TMS320C6000 Peripherals Reference Guide

Literature Number: SPRU190D
February 2001

PRINTED WITH SOY INK™

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.

TMS320C6x Peripheral Support Library Programmer’s Reference (literature number SPRU273) describes the contents of the ’C6x peripheral support library of functions and macros. It lists functions and macros both by header file and alphabetically, provides a complete description of each, and gives code examples to show how they are used.

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.

TMS320C6x Evaluation Module Reference Guide (literature number SPRU269) provides instructions for installing and operating the 'C6x evaluation module. It also includes support software documentation, application programming interfaces, and technical reference material.

TMS320C62x Multichannel Evaluation Module User's Guide (literature number SPRU285) provides instructions for installing and operating the 'C62x multichannel evaluation module. It also includes support software documentation, application programming interfaces, and technical reference material.

6TMS320C62x Multichannel Evaluation Module Technical Reference (SPRU308) provides provides technical reference information for the 'C62x multichannel evaluation module (McEVM). It includes support software documentation, application programming interface references, and hardware descriptions for the 'C62x McEVM.

TMS320C6201/6701 Evaluation Module Technical Reference (SPRU305) provides provides technical information that describes the 'C6x evaluation module functionality. It includes a description of host software utilities and a complete application programming interface reference.

TMS320C6000 DSP/BIOS User's Guide (literature number SPRU303) describes how to use DSP/BIOS tools and APIs to analyze embedded real-time DSP applications.

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.

TMS320C6203 Digital Signal Processor Data Sheet (literature number SPRS086) describes the features of the TMS320C6203 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.

PC is a trademark of International Business Machines Corporation.

PowerQUICC is a trademark of Motorola.

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.

Contents

Figures

Tables

Chapter 1

Introduction

The TMS320C6000™ platform of devices consists of the first off-the-shelf digital signal processors (DSPs) that use advanced very long instruction word (VLIW) to achieve high performance through increased instruction-level parallelism. The VelociTI™ 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. The features, memory, and peripherals of the C6000 devices are described.

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 includes 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	systems (PCS)	Speech vocoding
Channel multiplexing	Personal digital assistants (PDA)	Text-to-speech
Data encryption	Speaker phones	Voice mail
Digital PBXs	Spread spectrum communications	
Digital speech interpolation (DSI)	Digital subscriber loop (xDSL)	
DTMF encoding/decoding	Video conferencing	
Echo cancellation	X.25 packet switching	

1.2 Overview of the TMS320C6000 Platform of DSPs

With a performance of up to 4800 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, with the VelociTI™ architecture, execute up to eight 32-bit instructions per cycle. The device's core CPU consists of two general-purpose register files, A and B, which have a 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 (32–bit) and double-precision (64–bit) instructions (C67x only).

❑ 32 x 32–bit integer multiply with 32– or 64–bit result

❑ Pin-compatible fixed-point and floating-point DSPs.

The C64x device brings the highest level of performance, with the VelociTI.2 extensions to the VelociTI architecture. Some key features include:

❑ Register file enhancement

❑ Data path extensions

❑ Packed data processing

❑ Additional functional unit hardware

❑ Increased orthogonality.

For more information on features and options of the TMS320C6000, see the *TMS320C6000 CPU and Instruction Set Reference Guide (SPRU 189)*.

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).

TMS320C620x™/C670x™: The C620x and C670x devices 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.

TMS320C621x™/C671x™/C64x™: The C621x/C671x/C64x 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 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 these functions:

- ❑ The CPU and the enhanced direct memory access (EDMA) controller access to internal memory, and performance of the necessary arbitration.
- ❑ The CPU data access to the EMIF
- ❑ The CPU accesses to on-chip peripherals
- ❑ Sending requests to EMIF for an L2 data miss.

The internal SRAM of the C621x/C671x/C64x 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

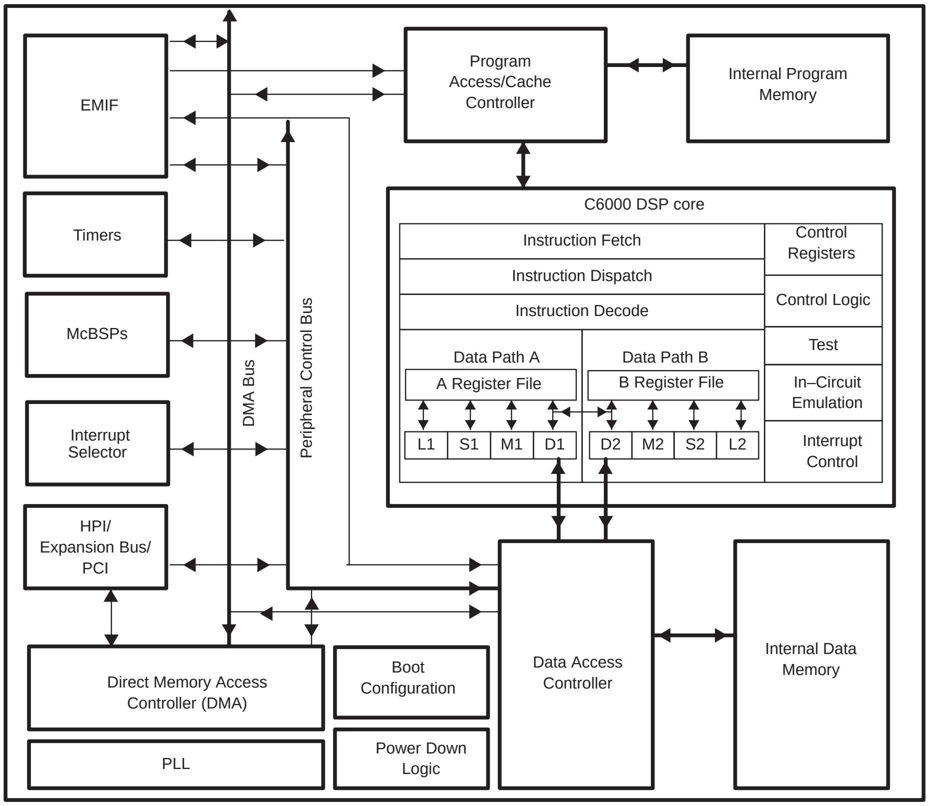
Peripheral	C6201	C6202(B) C6203(B)	C6204	C6205	C621x	C6414	C6415	C6701	C671x
Direct memory access (DMA) controller	Y	Y	Y	Y	N	N	N	Y	N
Enhanced direct memory access (EDMA) controller	N	N	N	N	Y	Y	Y	N	Y
Host-port interface (HPI)	Y	N	N	N	Y	Y	Y†	Y	Y
Expansion bus (XBUS)	N	Y	Y	N	N	N	N	N	N
PCI	N	N	N	Y	N	N	Y†	N	N
External memory interface (EMIF)	1	1	1	1	1	2	2	1	1
Boot configuration	Y	Y	Y	Y	Y	Y	Y	Y	Y
Multichannel buffered serial ports (McBSPs)	2	3	2	2	2	3	3†	2	2
UTOPIA	N	N	N	N	N	N	Y†	N	N
Interrupt selector	Y	Y	Y	Y	Y	Y	Y	Y	Y
32-bit timers	2	2	2	2	2	3	3	2	2
Power-down logic	Y	Y	Y	Y	Y	Y	Y	Y	Y
GPIO peripheral	N	N	N	N	N	Y	Y†	N	N

† The C6415 peripheral set is selected at device reset. For details, see *Chapter 11, Boot Modes and Configuration*, and the specific device datasheet.

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.

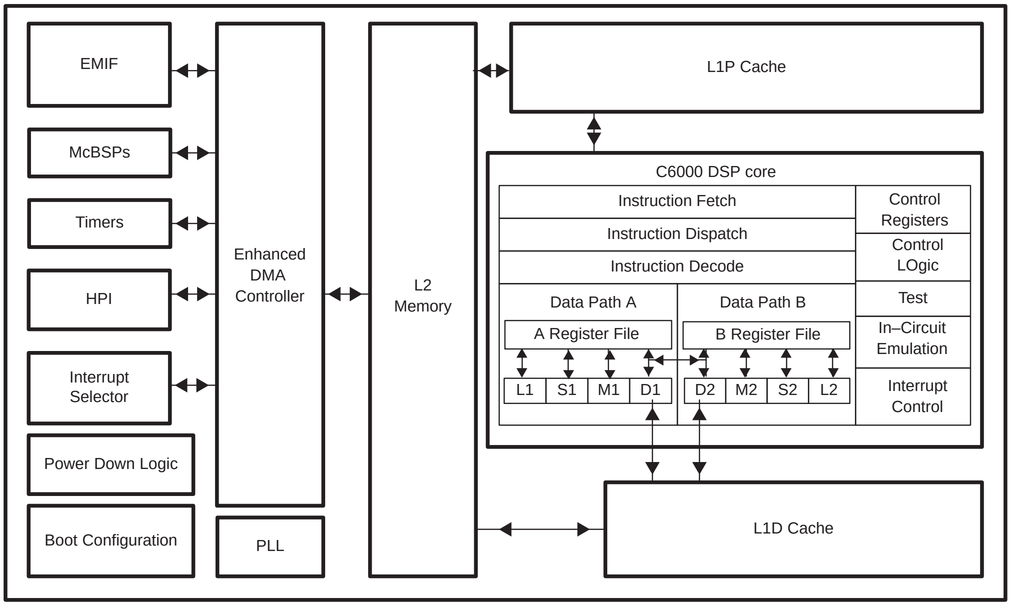
Figure 1–1 shows the peripherals in the block diagram for the TMS320C620x/C670x devices. Figure 1–2 shows a block diagram for the TMS320C621x/C671x devices. Figure 1–3 shown a block diagram for the TMS320C64x devices.

Figure 1–1. TMS320C620x/C670x Block Diagram



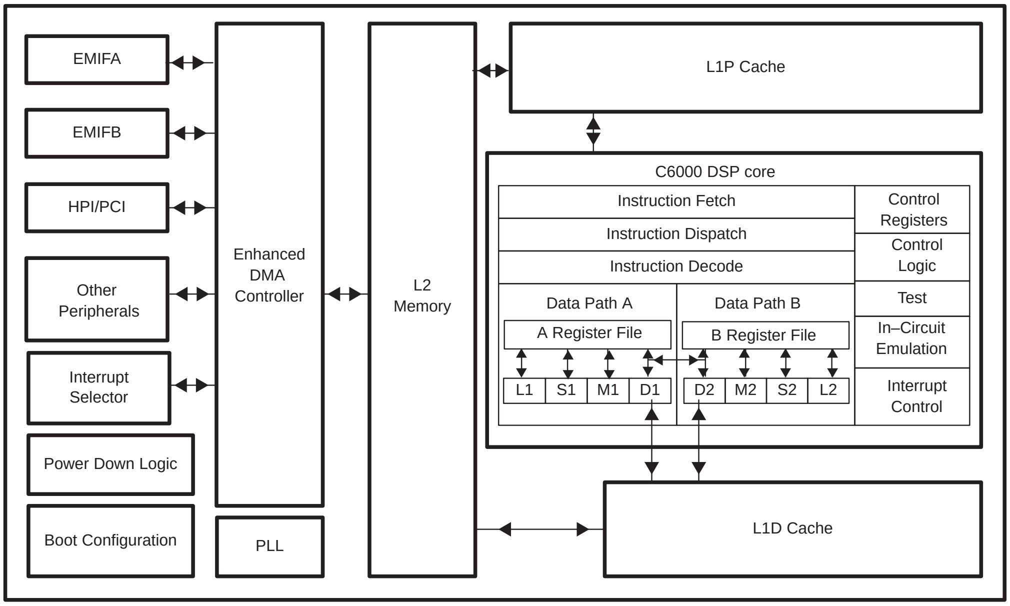
Note: Refer to the specific device datasheet for its peripheral set.

Figure 1–2. TMS320C621x/C671x Block Diagram



Note: Refer to the specific device datasheet for its peripheral set.

Figure 1–3. TMS32064x Block Diagram



Note: Refer to the specific device datasheet for its peripheral set.

The main functions of the devices shown in the preceding figures are summarized as follows:

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 16 (C621x/C671x) or 64 (C64x) 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.

PCI: The PCI module supports connection of the C6000 device to a PCI host via the integrated PCI master/slave bus interface.

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

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

Boot Configuration: The TMS320C6000 devices 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 these capabilities:

- ❑ 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 analog-to-digital (A/D) and digital-to-analog (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.

UTOPIA: The UTOPIA (Universal Test and Operations Interface for ATM) peripheral allows the DSP to be a slave ATM Controller device. The interface supports the UTOPIA level 2 interface and conforms to the ATM Forum standard specification af-phy-0039.0000. The UTOPIA interfaces to an ATM master.

Timer: The C6000 devices have 32-bit general-purpose timers that are used to perform these functions:

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

GPIO: The GPIO peripheral provides a dedicated set of general-purpose input/output signals to the C6000 device. Transitions on these signals can be used to generate interrupts to the CPU and synchronization events to the EDMA.

Interrupt Selector: The C6000 peripheral set produces interrupt sources. The interrupt selector allows the user to choose which interrupts their 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

TMS320C620x/C670x Internal Program and Data Memory

This chapter describes the TMS320C620x/C670x™ program and data memory controller. Program memory modes, including cache operation and bootload operation, are discussed. Also described in the chapter are the program and data memory organizations in the C620x/C670x devices.

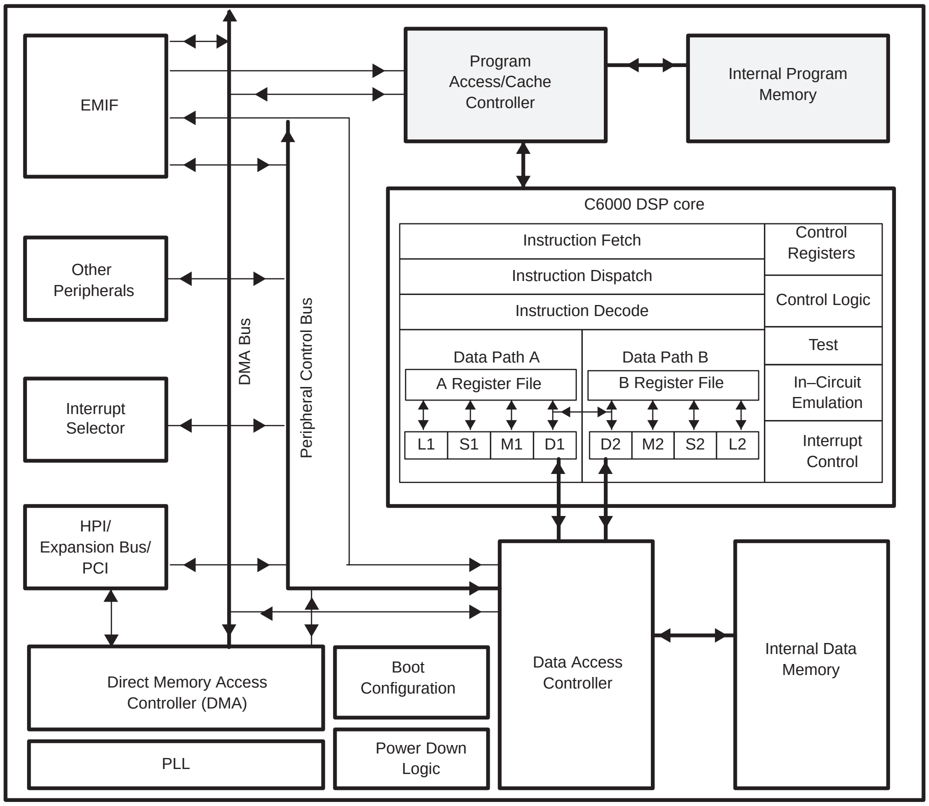
Topic **Page**

2.1 Program Memory Controller

The program memory controller, shaded 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. TMS320C620x/C670x Program Memory Controller in the Block Diagram



Note: Refer to the specific device datasheet for its peripheral set.

2.2 Internal Program Memory

The TMS320C6201/C6204/C6205/C6701 internal program memory is user-configurable as cache or memory-mapped program space. It 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.

In the TMS320C6202(B)/C6203(B), the memory/cache program space is expanded to the sizes shown in Table 2–1. In addition, the C6202(B)/C6203(B) provides another block of memory that operates as a memory-mapped block. These two blocks can be accessed independently. This allows the CPU to perform program fetch from one block of program memory, without interfering with a DMA transfer from the other block.

Table 2–1 and Table 2–2 compare the internal memory and cache configurations available on the current TMS320C6x0x devices. Figure 2–2 shows a block diagram of the connections between the C6201/ C6204/C6205/C6701 CPU, PMEMC, and memory blocks. Figure 2–3 shows a block diagram of the connections between the CPU, PMEMC, and memory blocks in the C6202/C6202B/C6203(B). The addresses shown in Figure 2–2 and Figure 2–3 are for operation in memory map mode 1.

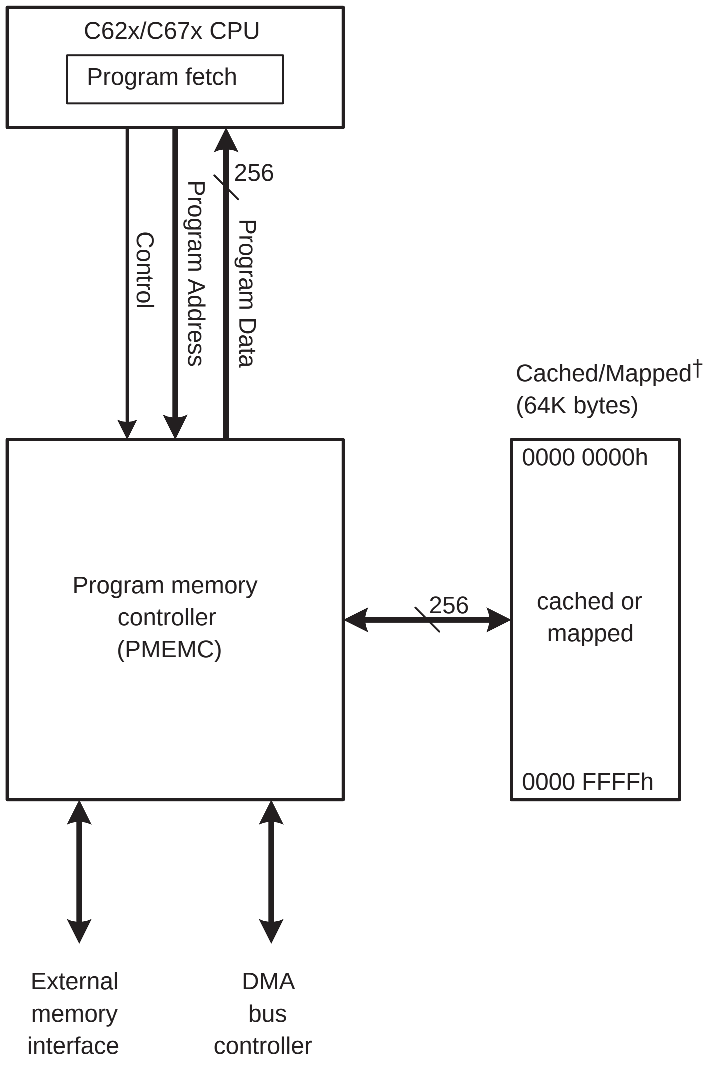
Table 2–1. TMS320C6000 Internal Memory Configurations

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(B)	6200	Harvard	384K	128K (map) 128K (map/cache)	128K (map)
C6203(B)	6200	Harvard	896K	256K (map) 128K (map/cache)	512K (map)
C6204	6200	Harvard	128K	64K (map/cache)	64K (map)
C6205	6200	Harvard	128K	64K (map/cache)	64K (map)

Table 2–2. TMS320C6000 Cache Architectures

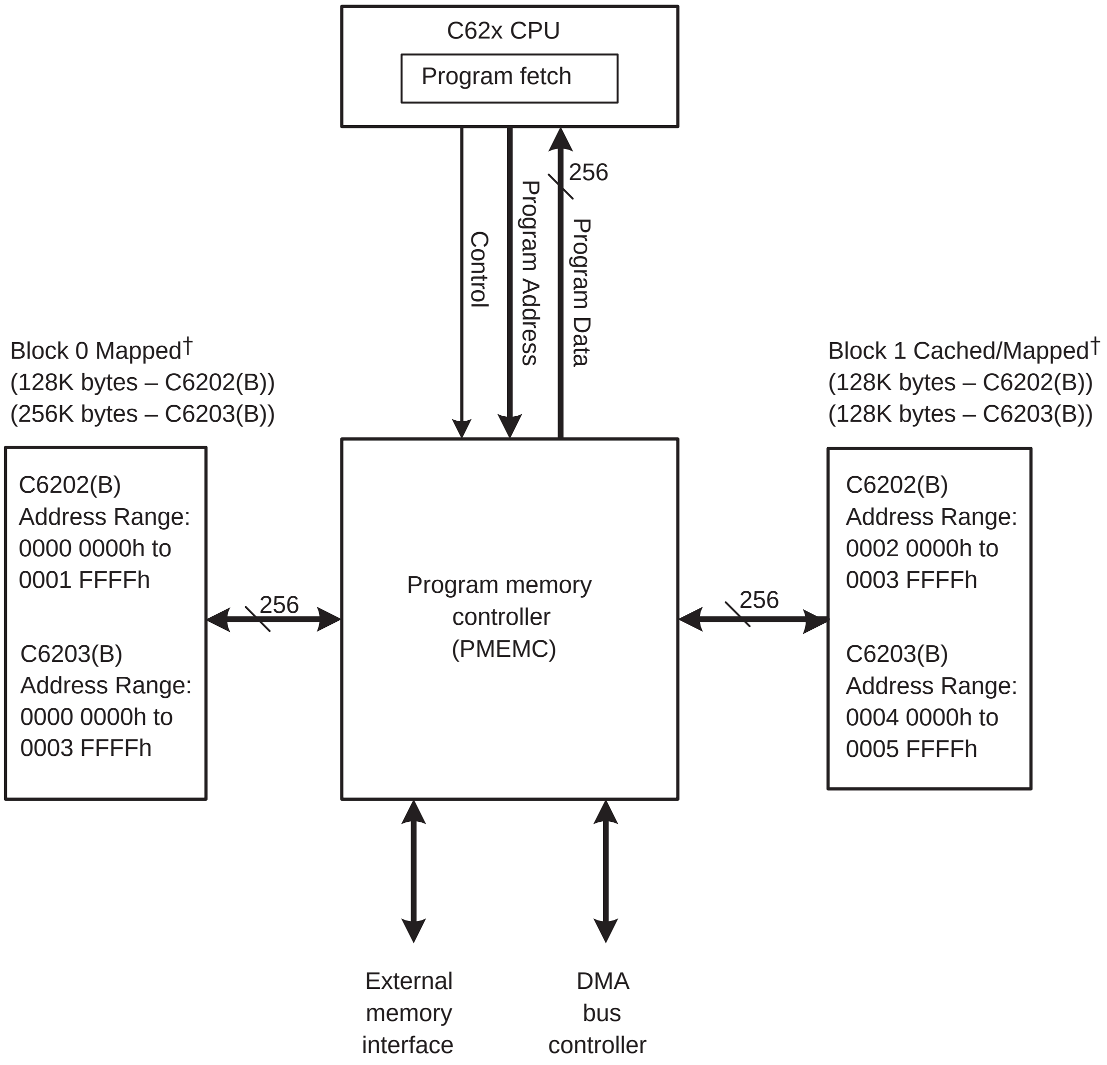
Cache Space	Size (Bytes)	Associativity	Line Size (Bytes)
C6201 program	64K	Direct mapped	32
C6701 program	64K	Direct mapped	32
C6202(B) program	128K	Direct mapped	32
C6203(B) program	128K	Direct mapped	32
C6204 program	64K	Direct mapped	32
C6205 program	64K	Direct mapped	32

Figure 2–2. TMS320C6201/C6204/C6205/C6701 Program Memory Controller Block Diagram



† **Note:** Addresses shown in Map 1 mode (section 2.2.2)

Figure 2–3. TMS320C6202(B)/C6203(B) Program Memory Controller Block Diagram


C62x CPU
Program fetch
Control
Program Address
Program Data
256
Block 0 Mapped†
(128K bytes – C6202(B))
(256K bytes – C6203(B))
C6202(B) Address Range: 0000 0000h to 0001 FFFFh
C6203(B) Address Range: 0000 0000h to 0003 FFFFh
256
Program memory controller (PMEMC)
256
Block 1 Cached/Mapped†
(128K bytes – C6202(B))
(128K bytes – C6203(B))
C6202(B) Address Range: 0002 0000h to 0003 FFFFh
C6203(B) Address Range: 0004 0000h to 0005 FFFFh
External memory interface
DMA bus controller

† **Note:** Addresses shown in Map 1 mode (section 2.2.2)

2.2.1 Internal Program Memory Modes

The cached/mapped block of 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 2–3. The modes are:

- ❑ **Mapped:** In mapped mode, program fetches from the cached/mapped block of 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 section 2.2.2 for a detailed description of the memory mapped operations. See Chapter 11, *Boot Modes and Configuration*, 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. 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 2–3. 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 the operation mode of the PMEMC is changed, 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, modify line four of the routine to correspond the the value of PCC that is to be 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. The user must ensure that no interrupts occur during the execution of this assembly routine to prevent unexpected modification of the CSR.

```
   .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 Memory Mapped Operation

When the PCC field of the CPU control status register is programmed for Mapped mode, all of the internal program RAM is mapped into internal program space. Table 2–4 shows the address space for the internal program RAM for the map mode selected at device reset.

Table 2–4. Internal Program RAM Address Mapping in Memory Mapped Mode

Device	**Block** (See Note)	**Map 0**	**Map 1**
C6201	——	0140 0000h – 0140 FFFFh	0000 0000h – 0000 FFFFh
C6202(B)	Block 0 Block 1	0140 0000h – 0141 FFFFh 0142 0000h – 0143 FFFFh	0000 0000h – 0001 FFFFh 0002 0000h – 0003 FFFFh
C6203(B)	Block 0 Block 1	0140 0000h – 0143 FFFFh 0144 0000h – 0145 FFFFh	0000 0000h – 0003 FFFFh 0004 0000h – 0005 FFFFh
C6204	——	0140 0000h – 0140 FFFFh	0000 0000h – 0000 FFFFh
C6205	——	0140 0000h – 0140 FFFFh	0000 0000h – 0000 FFFFh
C6701	——	0140 0000h – 0140 FFFFh	0000 0000h – 0000 FFFFh

Note: TMS320C6201/C6204/C6205/C6701 has only one block of internal program memory.

In mapped mode, both the CPU and the DMA can access all locations in the RAM. Any access outside of the address space that the internal RAM is mapped to is forwarded to the EMIF. 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.

For the C6202(B)/C6203(B), 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. The DMA cannot cross between Block 0 and Block 1 in a single transfer. Separate DMA transfers must be used to cross block boundaries.

2.2.3 Cache Operation

When the PCC field of the CPU Control Status Register is programmed for one of the Cache modes, all internal program memory in the C6201/C6204/C6205/C6701 device is used as a cache. For the C6202(B)/C6203(B) device, block 1 operates as a cache while block 0 remains mapped into internal program space. Table 2–5 shows the addresses occupied by the C6202/C6202B/C6203(B) RAM that is not used for cache, for each map mode.

Table 2–5. TMS320C6202(B)/C6203(B) Internal Program RAM Address Mapping in Cache Mode

Device	**Block**	**Map 0**	**Map 1**
C6202(B)	0	0140 0000h – 0141 FFFFh	0000 0000h – 0001 FFFFh
C6203(B)	0	0140 0000h – 0143 FFFFh	0000 0000h – 0003 FFFFh

Any CPU or DMA access to the memory range that was occupied by the cache RAM returns undefined results. For the C6202(B)/C6203(B), as in the map mode simultaneous accesses to block 0 by the CPU and DMA stalls the DMA until the CPU has completed its access. It is necessary to ensure that all DMA accesses to block 1 have completed before the cache is enabled.

2.2.4 Cache Architecture

The C620x/C670x cache is a direct mapped architecture. The width of the cache (the line size) is 256 bits, or eight 32-bit instructions. Each line in the cache is one fetch packet. Therefore, for the C6201/C6204/C6205/C6701, the 64K byte cache contains 2K fetch packets (2K lines). For the C6202(B)/C6203(B) device, the 128k-byte cache contains 4K fetch packets (4K lines).

2.2.4.1 Cache Usage of CPU Address

How the cache uses the fetch packet address from the CPU is shown in Figure 2–4 for the C6201/C6204/C6205/C6701 and Figure 2–5 for the C6202(B)/C6203(B). These functions are described below:

❑ **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).

❑ **Tag block offset:** The device characteristics are as follows:

- For the C6201/C6204/C6205/C6701, any external address maps to only one of the 2K lines. Any two fetch packets that are separated by an integer multiple of 64K bytes map to the same line. Thus bits 15:5 of the CPU address create the 11-bit block offset that determines the specific line, of the 2K lines, to which any particular fetch packet maps.
- For the C6202(B)/C6203(B), any external address maps to only one of the 4K lines. Any two fetch packets that are separated by an integer multiple of 128K bytes map to the same line. Thus bits 16:5 of the CPU address create the 12-bit block offset that determines the specific line, of the 4K lines, to which any particular fetch packet maps.

❑ **Tag:** The cache assumes a maximum external address space of 64M bytes (from 0000 0000h–03FF FFFFh). The following bits of the address correspond to the tag that determines the original location of the fetch packet in external memory space:

- For C6201/C6204/C6205/C6701, bits 25:16. The cache has a separate 2K $\times$ 11 tag RAM that holds all the tags.
- For C6202(B)/C6203(B), bits 25:17. The cache has a separate 4K x 10 tag RAM that holds all the tags.

Each address location in the tag RAM contains the tag, plus a valid bit that is used to record line validity information.

Figure 2–4. Logical Mapping of Cache Address (C6201/C6204/C6205/C6701)

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

Figure 2–5. Logical Mapping of Cache Address (C6202(B)/C6203(B))

31 – 26	25 – 17	16 – 5	4 – 0
Outside external range. assumed to be 0	Tag	Block offset	Fetch packet alignment. Assumed 0

2.2.4.2 Cache Flush

A dedicated valid bit in each address location of the tag RAM indicates whether the contents of the corresponding cache line is valid data. During a cache line, all of the valid bits are cleared to indicate that no cache lines 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. The CPU is halted during a cache flush. For the C6202(B)/C6203(B), a DMA access to block 0 while the cache is flushed continues without stalling.

2.2.4.3 Line 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 the tag field 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 line, sets the valid bit, sets the tag to bits of the requested address, and delivers this fetch packet to the CPU after all eight instructions are available.

2.2.5 Bootload Operation

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 11, Boot Modes and Configuration*, for more information on bootloading code.

2.2.6 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. Only 32–bit words accesses can be made to the program memory via the DMA. 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 to the same program memory block. The maximum throughput to the DMA is one access every other cycle. In 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.

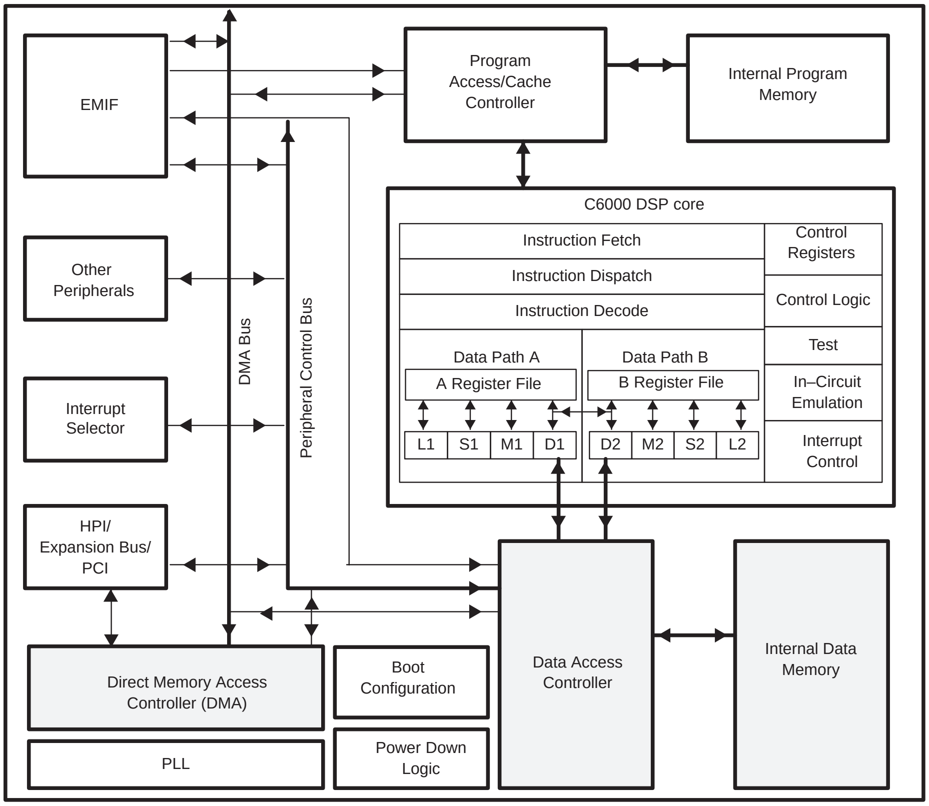
2.3 Data Memory Controller

As shown in Figure 2–6, 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 2–6. TMS320C620x/C670x Block Diagram



Note: Refer to the specific device datasheet for its peripheral set.

2.3.1 Data Memory Access

The data memory controller services all CPU and DMA controller data requests to internal data memory. The directions of data flow and the master (requester) and slave (resource) relationships between the modules are shown in *Section 2.4* Figure 2–7, Figure 2–8, Figure 2–9, and Figure 2–10. These figures show:

- 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.

See *Chapter 5, DMA and CPU Data Access Performance* for a detailed description of data access performance to the internal data memory, the EMIF, and the peripheral bus.

2.4 Internal Data Memory

The following sections describe the memory organization of each C620x and C670x device.

2.4.1 TMS320C6201/C6204/C6205

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, as shown in Table 2–6, Figure 2–7, and Table 2–7. 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. Since accesses to different blocks never cause performance penalties, it is not necessary to consider the address within a block if simultaneous accesses occur to different blocks. Both CPU and DMA can simultaneously access data that resides in different banks within the same block without a performance penalty. The two CPU data ports, A and B, can simultaneously access neighboring 16-bit data elements inside the block without a resource conflict. To avoid performance penalties, it is necessary to give attention to address LSBs when the two accesses involve data in the same block. With this memory configuration, the maximum data access each cycle is three 32-bit accesses made by CPU data port A, B, and the DMA controller to different banks.

Table 2–6. Data Memory Organization (TMS320C6201/C6204/C6205)

	Bank 0		**Bank 1**		**Bank 2**		**Bank 3**	
First address (Block 0)	80000000	80000001	80000002	80000003	80000004	80000005	80000006	80000007
	80000008	80000009	8000000A	8000000B	8000000C	8000000D	8000000E	8000000F
	•	•	•	•	•	•	•	•
	•	•	•	•	•	•	•	•
	•	•	•	•	•	•	•	•
	80007FF0	80007FF1	80007FF2	80007FF3	80007FF4	80007FF5	80007FF6	80007FF7
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
	80008008	80008009	8000800A	8000800B	8000800C	8000800D	8000800E	8000800F
	•	•	•	•	•	•	•	•
	•	•	•	•	•	•	•	•
	•	•	•	•	•	•	•	•
	8000FFF0	8000FFF1	8000FFF2	8000FFF3	8000FFF4	8000FFF5	8000FFF6	8000FFF7
Last address (Block 1)	8000FFF8	8000FFF9	8000FFFA	8000FFFB	8000FFFC	8000FFFD	8000FFFE	8000FFFF

Figure 2–7. Data Memory Controller Interconnect to Other Banks (TMS320C6201/C6204/C6205)

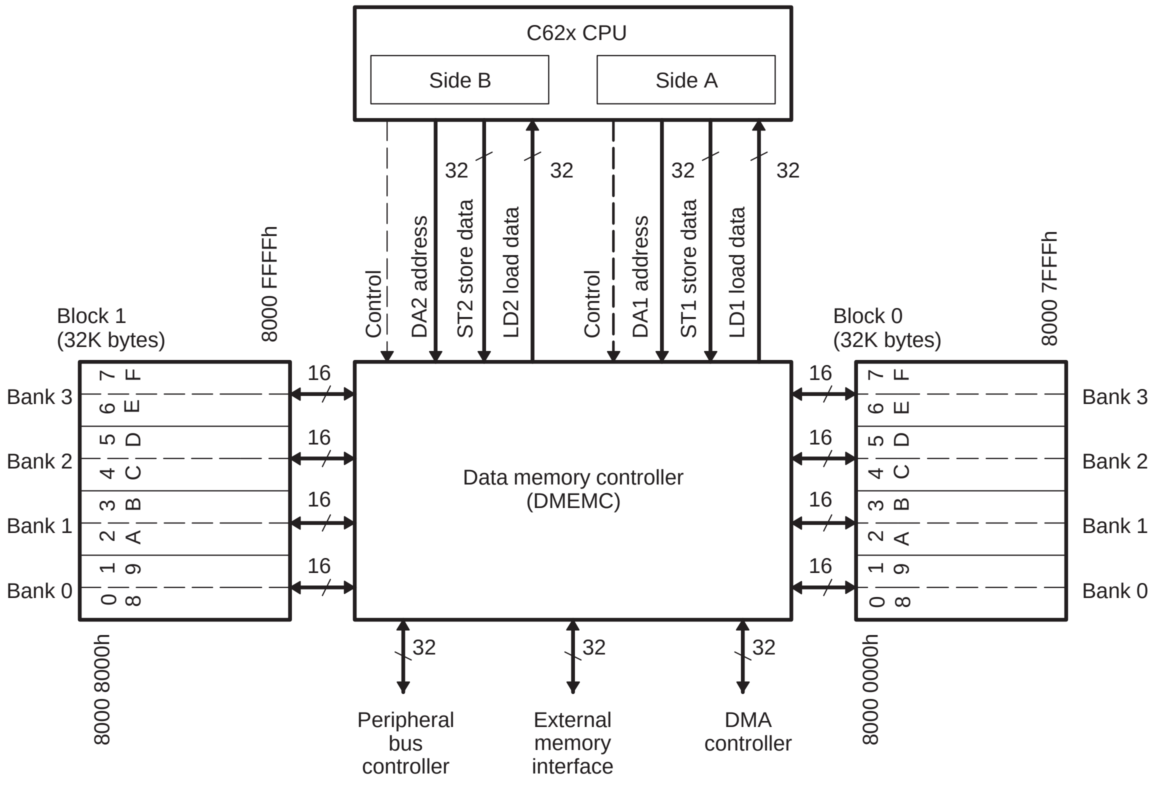


Table 2–7. Internal Data RAM Address Mapping

Block 0	8000 0000h – 8000 7FFFh
Block 1	8000 8000h – 8000 FFFFh

2.4.2 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, as shown in Table 2–8, Figure 2–8, and Table 2–9. 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. Since accesses to different blocks never cause performance penalties, it is not necessary to consider the address within a block if simultaneous accesses occur to different blocks. It is only necessary to give attention to the address when the two accesses occur in the same block. 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. The two CPU data ports, A and B, can simultaneously access neighboring 16-bit data elements inside the same block without a resource conflict. To avoid performance penalties, it is necessary to give attention to address LSBs when two accesses involve data in the same block. With this memory configuration, the maximum data access each cycle is two 64-bit CPU accesses (LDDW only) and a 32-bit DMA access.

Table 2–8. Data Memory Organization (TMS320C6701)

	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 2–8. Data Memory Controller Interconnect to Other Blocks (TMS320C6701)

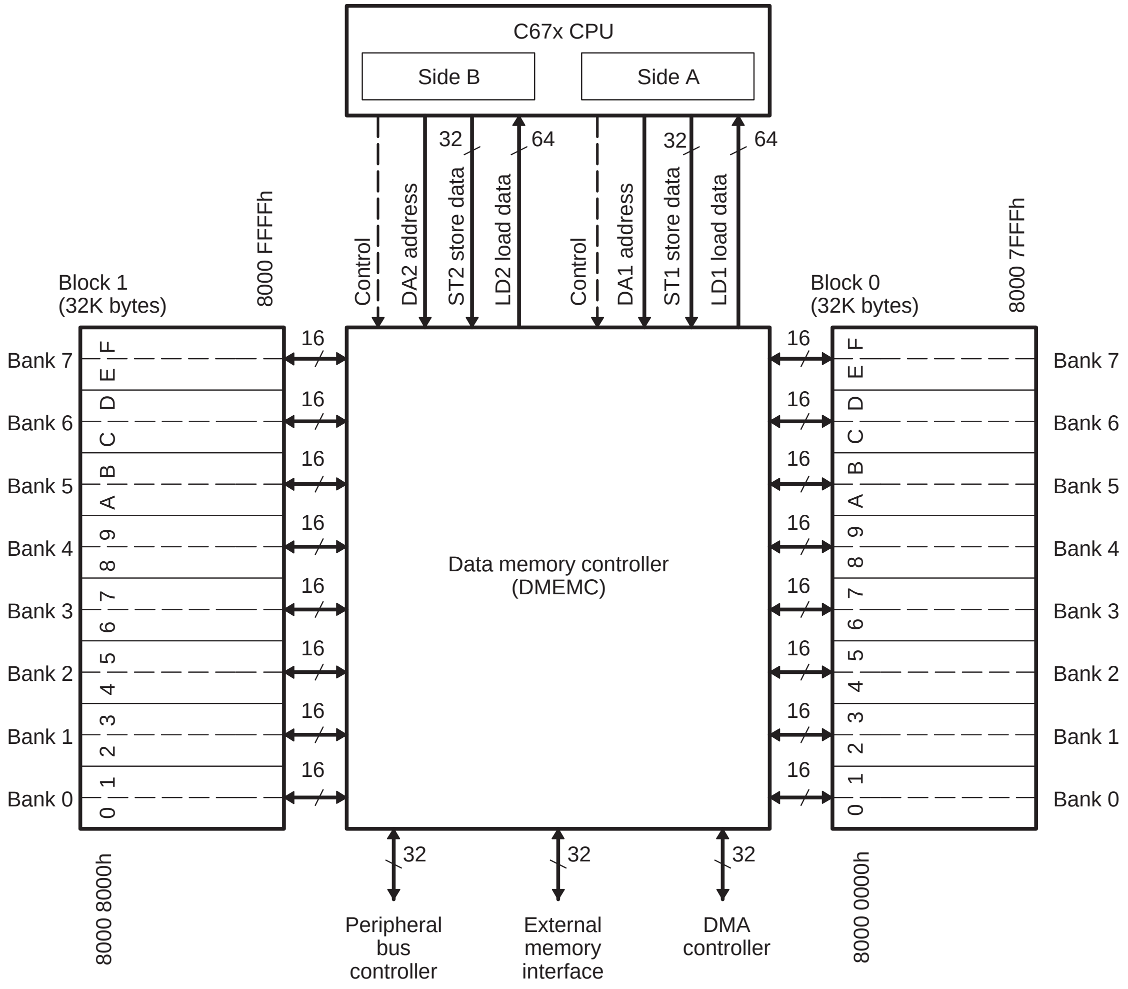


Table 2–9. Internal Data RAM Address Mapping

Block 0	8000 0000h – 8000 7FFFh
Block 1	8000 8000h – 8000 FFFFh

2.4.3 TMS320C6202(B)

The TMS320C6202(B) data memory controller (DMEMC) provides all of the functionality available in the TMS320C6201 revision 3. The C6202(B) 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(B) 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 2–9 shows a block diagram of the connections between the C6202(B) CPU, DMEMC, and memory blocks. Table 2–10 shows the memory range occupied by each block of internal data RAM.

Figure 2–9. TMS320C6202(B) Data Memory Controller Block Diagram

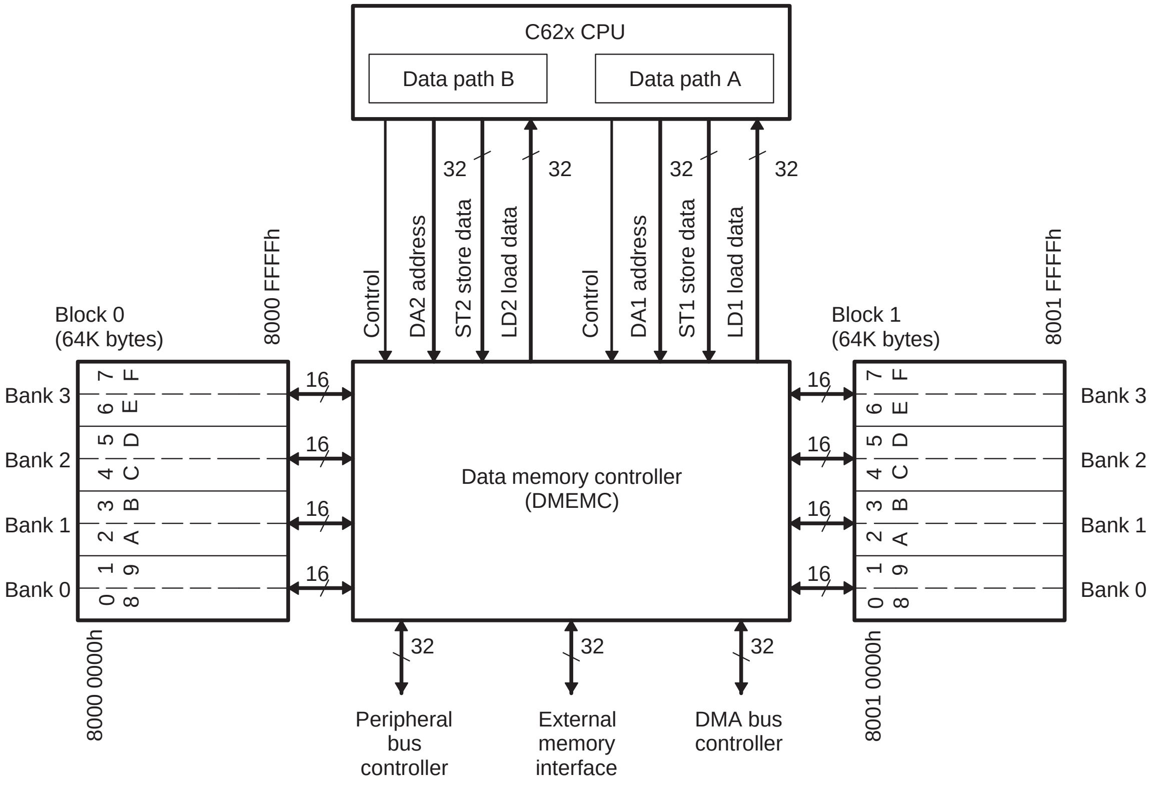


Table 2–10. Internal Data RAM Address Mapping

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

2.4.4 TMS320C6203(B)

The TMS320C6203(B) data memory controller (DMEMC) provides all of the functionality available in the TMS320C6201 revision 3. The C6203(B) DMEMC contains 512K bytes of RAM organized in two blocks of four banks each. Each bank is 16 bits wide. The DMEMC for the C6203(B) 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 2–10 shows a block diagram of the connections between the C6203(B) CPU, DMEMC, and memory blocks. Table 2–11 shows the memory range occupied by each block of internal data RAM.

Figure 2–10. TMS320C6203(B) Data Memory Controller Block Diagram

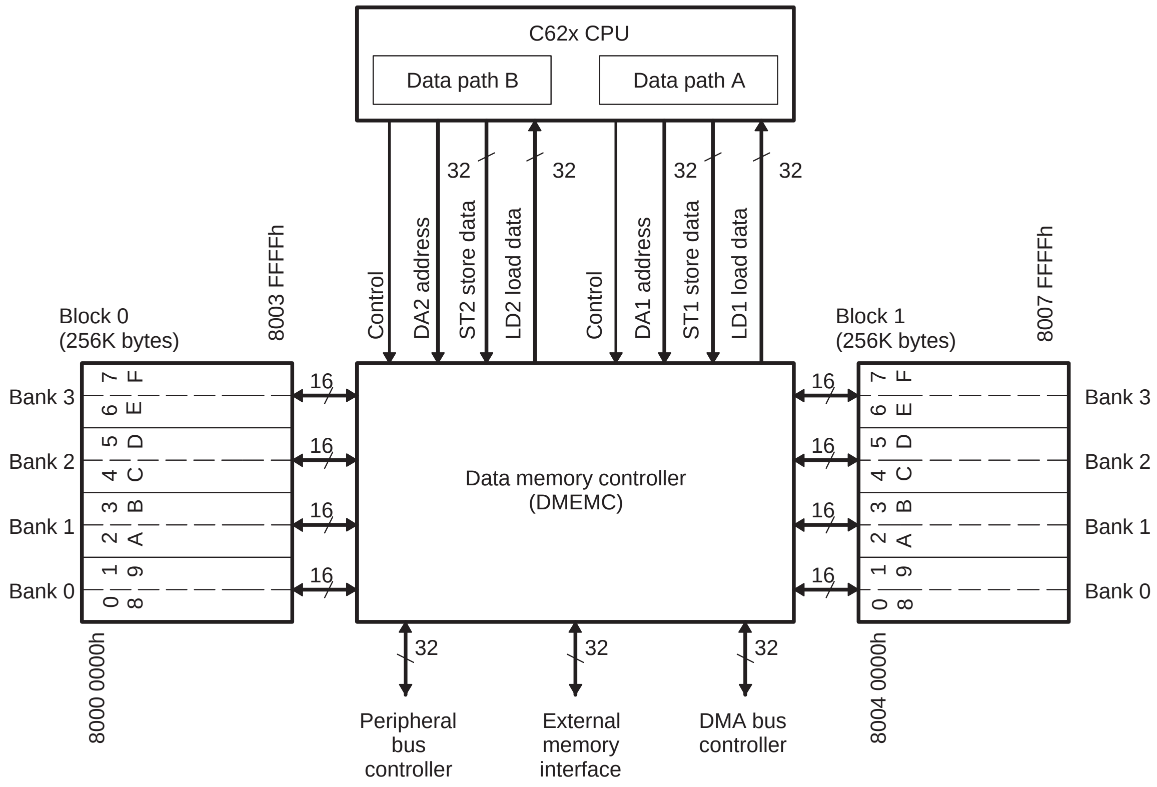


Table 2–11. Internal Data RAM Address Mapping

Block 0	8000 0000h – 8003 FFFFh
Block 1	8004 0000h – 8007 FFFFh

2.4.5 Data Alignment

The following data alignment restrictions apply:

Doublewords: (C6701 only) Doublewords are aligned on even 8-byte (double-word) 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.

2.4.6 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 execute 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 2–11 and Figure 2–12 show what access conditions cause internal memory conflicts when the CPU makes two data accesses (on DA1 and DA2).

Figure 2–11. Conflicting Internal Memory Accesses to the Same Block (TMS320C6201/C6202(B)/C6203(B)/C6204/C6205)

	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 2–12. 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.

2.4.7 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. Figure 2–11 and Figure 2–12 can be used 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 3, *TMS320C621x/C671x/C640x 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.

2.4.8 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 = 0 selects big-endian. Byte ordering within word and half word data resident in memory is identical for little-endian and big-endian data. Table 2–12 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 2–13 and Table 2–14 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 2–12. Register Contents After Little-Endian or Big-Endian Data Loads (TMS320C620x and TMS320C670x)

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 2–13. 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 2–14. Memory Contents After Little-Endian or Big-Endian Data Stores (TMS320C620x/C670x)

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.

2.5 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.

2.5.1 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 2–15. 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 2–15. 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 nibbles correctly written,
? indicates nibbles with undefined value after write.

2.5.2 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.

2.5.3 Arbitration Between the CPU and the DMA Controller

As shown in Figure 2–7, Figure 2–8, Figure 2–9, and Figure 2–10, 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

TMS320C621x/C671x/C64x Two-Level Internal Memory

The TMS320C621x/C671x/C64x provides a two-level memory architecture for the internal program bus and the data bus. The two-level memory consists of the program cache and the data cache.

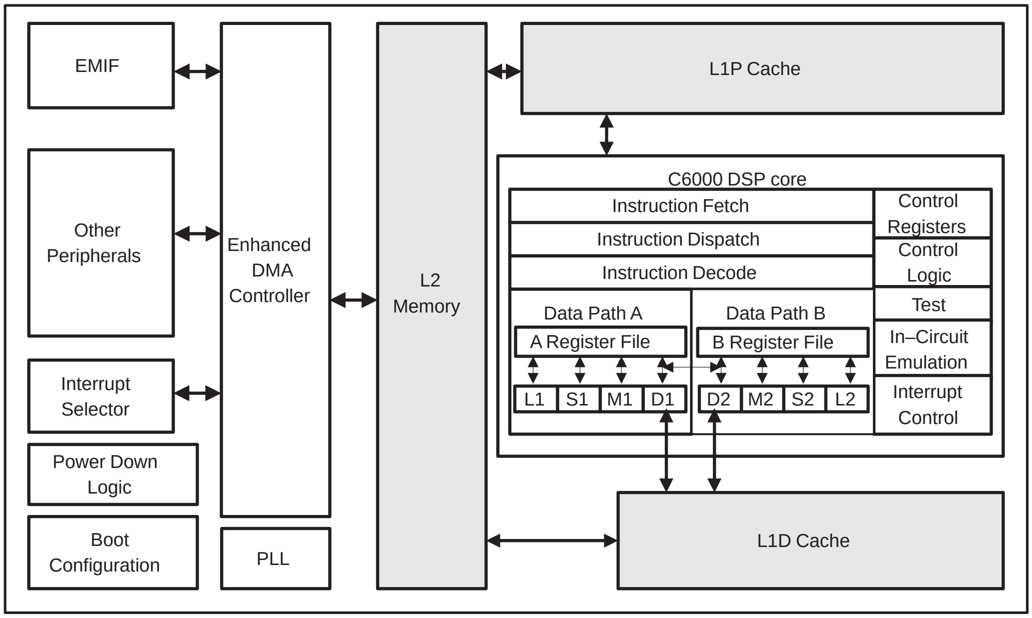
Topic **Page**

3.1 Overview

The TMS320C621x/C671x/C64x has a two-level memory architecture for program and data. The first level program cache is designated L1P, and the first level data cache is designated L1D. Both the program and data memory share the second level memory, designated L2. L2 is configurable as partial cache and partial SRAM.

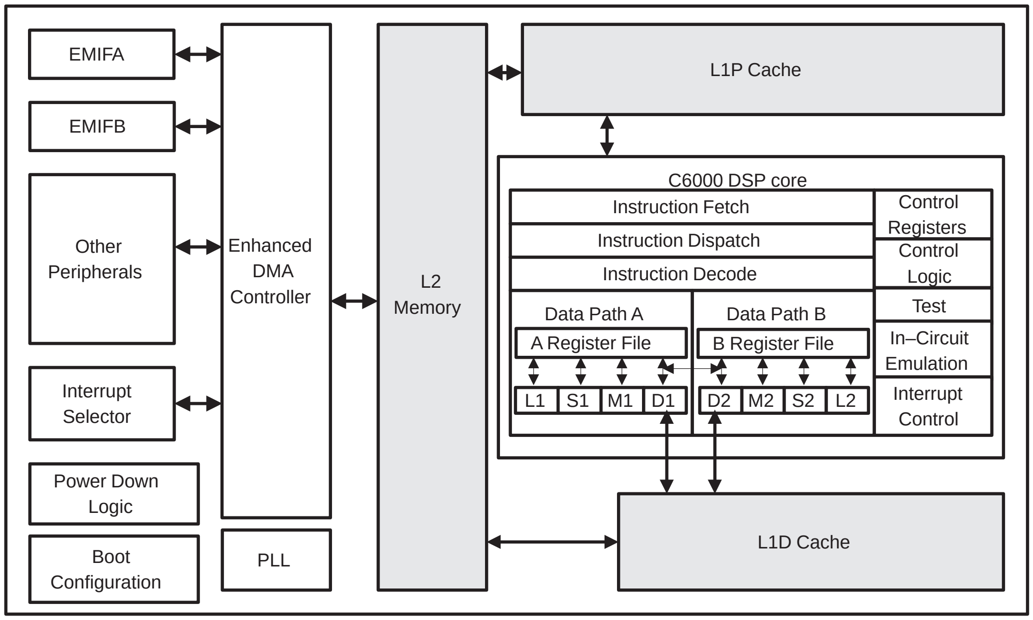
Figure 3–1 and Figure 3–2 show the block diagram of the C621x/C671x and C64x, respectively. Table 3–1 summarizes the internal layout of these devices. Figure 3–3 illustrates the bus connections between the CPU, internal memories, and the enhanced DMA for the TMS320C6000.

Figure 3–1. TMS320C621x/C671x Block Diagram



Note: Refer to the specific device datasheet for its peripheral set.

Figure 3–2. TMS320C64x Block Diagram



Note: Refer to the specific device datasheet for its peripheral set.

Figure 3–3. TMS320C6000 Two-Level Internal Memory Block Diagram

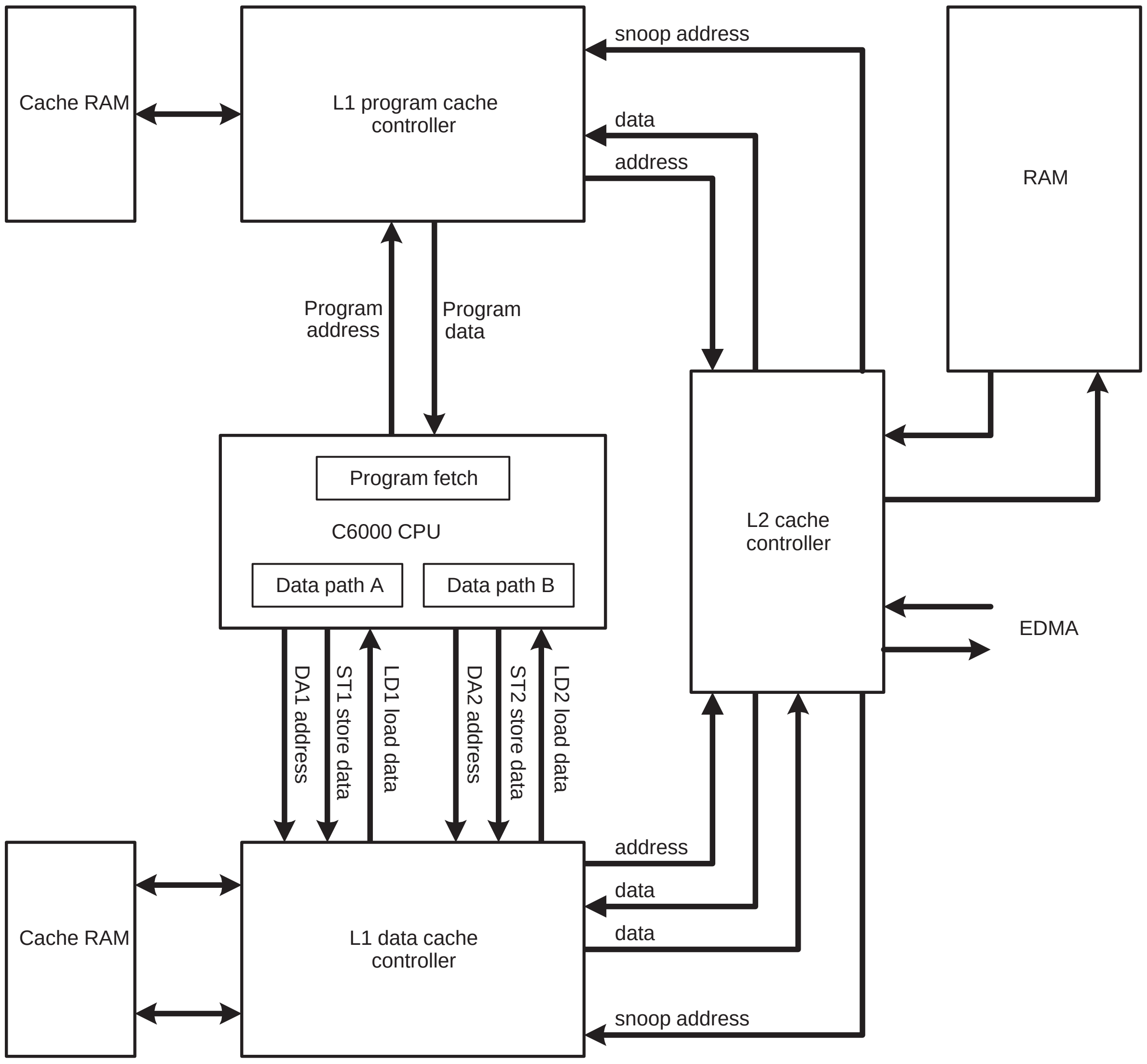


Table 3–1. TMS320C621x/C671x/C64x Internal Memory Details

	TMS320C621x/C671x	**TMS320C64x**
Internal Memory Structure	Two Level	Two Level
L1P size	4 Kbytes	16 Kbytes
L1P organization	Direct mapped	Direct mapped
L1P CPU access time	1 cycle	1 cycle
L1P line size	64 bytes	32 bytes
L1P banking	1 x 256 bit bank	8 x 32 bit banks
L1P read miss allocation	1 line allocated in L1P	1 line allocated in L1P
L1P read hit allocation	Data read from L1P	Data read from L1P
L1P write miss allocation	L1P writes not supported	L1P writes not supported
L1P write hit allocation	L1P writes not supported	L1P writes not supported
L1P –> L2 request size	2 fetches/L1P line	1 fetch/L1P line
L1P Protocol	Read Allocate	Read Allocate; Pipelined Misses
L1P Memory	Single Cycle RAM	Single cycle RAM
L1P –> L2 request CPU stall	5 cycles	8 cycles
L1D size	4 Kbytes	16 Kbytes
L1D organization	2-way set associative	2-way set associative
L1D CPU access time	1 cycle	1 cycle
L1D line size	32 bytes	64 bytes
L1D replacement strategy	2-way Least Recently Used	2-way Least Recently Used
L1D banking	64 bit wide dual ported RAM	8 x 32 bit banks
L1D read miss allocation	1 line allocated in L1D	1 line allocated in L1D
L1D read hit allocation	Data read from L1D	Data read from L1D
L1D write miss allocation	No allocation in L1D, data sent to L2	No allocation in L1D, data sent to L2
L1D write hit allocation	Data is written into hit L1D location	Data is written into hit L1D location
L1D Protocol	Read Allocate	Read allocate, Pipelined Misses
L1D –> L2 request size	2 fetches/L1D line	2 fetch/L1D line
L1D –> L2 request CPU stall	4 cycles	6 cycles / SRAM 8 cycles / L2 Cache hit

Table 3–1. TMS320C621x/C671x/C64x Internal Memory Details

	TMS320C621x/C671x	TMS320C64x
L2 size	64 Kbytes	1024Kbytes
L2 Cache Size	0/16/32/48/64KB	0/32/64/128/256KB
L2 SRAM Size	64/48/32/16/0 KB	1024/992/960/896/768KB
L2 organization	1/2/3/4-way set associative.	4-way set associative Cache
L2 line size	128 bytes	128 bytes
L2 replacement strategy	1/2/3/4-way Least Recently Used	4-way Least Recently Used
L2 banking	4 x 64 bit banks	8 x 64 bit banks
L2-L1P protocol	Coherency snoop invalidates	Coherency snoop invalidates
L2-L1D protocol	Coherency snoop	Coherency snoops
L2 protocol	Read and Write Allocate	Read and Write Allocate
L2 read miss allocation	Data read from EDMA, stored in L2	Data read from EDMA, stored in L2;
L2 read hit allocation	Data read from L2	Data read from L2
L2 write miss allocation	Data is read from EDMA into LRU L2 line and modified with new data	Data is read from EDMA into LRU L2 line and modified with new data
L2 write hit allocation	Data is written into hit L2 location	Data is written into hit L2 location
L2 –> L1P read path width	256 bit	256bit
L2 –> L1D read path width	128 bit	256 bit
L1D –> L2 write path width	32 bit	64 bit
L1D –> L2 victim path width	128 bits	256 bit
L2 –> EDMA read path width	64 bit	64 bit
L2 –> EDMA write path width	64 bit	64 bit

3.2 TMS320C621x/C671x/C64x Cache Definitions

The following is a list of terms that relate to the operation of the TMS320C621x/C671x/C64x two-level caches. These terms are used throughout the document.

Allocate – Allocation refers to the process of finding a location in the cache to store new, uncached data.

Associativity – Associativity refers to the number of line frames in each set.

Clean – When a cache line holds data that is valid and not modified, that cache line is clean.

Coherency – When a system uses a two-level memory system it is possible for the same address to be stored in both levels at the same time. If the lower level data has been modified, but the higher level memory has not been updated with the modification, then the two levels are said to be incoherent. If the same data resides in both levels of the memory hierarchy, then the two levels are said to be coherent.

Direct mapped cache – A direct mapped cache contains only one line that a specific higher level memory can be cached in. This does not mean that the cache contains as many lines as the memory it is caching, but instead that a memory location will only be cached in one specific line. For example if a memory location is cached in a direct mapped cache at location A, then cleaned out of the cache and then re-cached, it still can only be cached in location A.

DMA – Direct memory access, this is a transaction of a block of data from one memory space to another. It executes on the EDMA engine.

Dirty – When a cache line holds data that is valid and modified by the CPU but the modification has not been updated in the next higher level memory, that cache line is dirty.

Eviction – When a victim line contains dirty data, the data must be written out to the next level memory to maintain coherency. This process of writing dirty data to the next level memory is eviction. This is caused by a matching line address missing and being brought in to cache.

Execute Packet – An execute packet consists of a block of instructions that begin execution in parallel in a single cycle.

Fetch Packet – A fetch packet is a block of 8 instructions that are fetched in a single cycle some or all of these instructions are not necessarily executed in that cycle.

Higher Level Memory – A memory that is closer to the system memory. In this case the L2 is the higher level memory. It is closer to the physical memory external to the processor.

Hit – A cache hit occurs when the data for a requested memory location is present in the cache. A cache hit minimizes stalling, since the data can be fetched from the cache much faster than from the source memory.

Least recently used (LRU) allocation – For a set associative cache, least recently used allocation refers to the method used to choose which line frame to allocate space in on an allocation. When all of the lines in the set that the address maps to contain valid data, the line that was least recently read or written is allocated to store new data. In other words, the most recently used stays.

Line – A cache line is the amount of data fetched from the next higher level memory on a cache miss. The cache line can be larger than the size of the data request that forced the cache miss. For example a load byte instruction could force the L1D to miss but the cache will fetch an entire cache line from L2, not just the requested byte.

Line Frame – A line frame is a location in a cache that holds cached data (one line), an associated address tag and line state data. The state data indicates if the line is valid or dirty.

Load through – When a CPU request misses both a first level and the second level cache, the data is fetched from the external memory and stored to both the first level and second level cache simultaneously. A cache that stores data ,and sends that data to the lower cache at the same time, is a load-through cache. Using a load-through cache reduces the stall time compared to a cache which first stores the data, then sends it to the next lower level cache.

Long distance access – Accesses made by the CPU which are to a non-cacheable memory space are long distance accesses, and should be thought of as accesses directly to external memory. These are also used when a memory mapped control register is accessed.

Lower Level Memory – This is a memory that is closest to the CPU and furthest away from the physical memory system. In this case it would be the L1D and L1P being the lowest-level memories.

Memory Ordering – This is the process of defining what order the data will be processed in memory. There are two types, strong and relaxed. In the strong case all read and write operations take place strictly in program order. When accesses occur in parallel, the writes always take place before reads. If there are multiple reads or multiple writes, the data from the A datapath must be read first or written first depending on whether the execute packet is made up of loads or stores in parallel. Strong memory ordering is intuitive to normal software data flow.

Miss – A cache miss occurs when the data for a requested memory location is not in the cache. A miss causes the CPU to stall while the data is fetched from the next ‘higher’ level cache.

Miss Pipelining – When a single miss occurs many events have to occur to fetch the missing line into the lower-level memory. This includes fetching the data, writing to the tag memories, and writing back the evicted line. When multiple misses occur, this overhead is spread out over all of the misses. Only the overhead is seen for the first miss. Subsequent misses see much less overhead per miss.

Read allocate – A read allocate cache only allocates space in the cache on a read miss. A write miss does not cause an allocate to occur, instead the write data is passed on to the next ‘higher’ level cache.

Set – A set is a collection of line frames in which a line can reside. A direct mapped cache contains one line per set, an n-way set-associative cache contains n lines per set.

Set-associative cache – A set-associative cache contains multiple lines that each higher level memory location can be cached in. For example, memory location X can be cached in a set-associative cache at line A, B, or C. Location X is fetched and stored in line A, then it is cleaned out of the cache. When X is re-fetched, it can be cached in line A, B, or C depending on which line is allocated. Lines A, B, and C make up a set in this cache.

Snoop – Snooping is a method by which a high-level memory queries a lower-level memory to determine if the two memories contain data for the same address. In a two-level memory system, when data is moved out of the higher-level memory the lower-level memory must be snooped to determine if the address which is being evicted is incoherent with the same address in the lower-level memory.

Tag – The tag is a storage element containing the most significant bits of the line address. These are stored in the tag memories. The tag memories are queried to determine whether a range of addresses is present in the cache. This gives either a hit or a miss.

Thrash – When the CPU makes alternating accesses to multiple addresses which all map to the same cache line, each access will force the previous access out of the cache. Any subsequent access to the first location will again have to fetch the data from the next higher level memory since the data was evicted from the cache. This process of loading data into the cache, writing over the data, and reloading the data from the next-level memory is referred to as “thrashing the cache”.

Valid – When a cache line holds data that has been fetched from the next level memory, that cache line is valid.

Victim – When space is allocated in a cache, but all of the lines in the set that the address maps to contain valid data, then the data in the least recently used line is overwritten with the new data. The line that is overwritten is known as the victim line.

Writeback – A writeback cache will modify only its data on a write hit; it will not notify the next higher level memory that the write occurred. Thus the cache and the next -level memory will be incoherent, but the cache will hold the correct data.

Write allocate – A write-allocate cache allocates space for either a read or write miss. On a write miss, the space is allocated in the cache and the data is written into the cache; the data is not sent to the next level cache.

Write Merging – If writes have the same double word address, they can be merged into a single write. For example, two word stores to the same double word could be merged into a single write.

3.3 TMS320C621x/C671x Two-Level Memory

3.3.1 L1P Description

On the TMS320C621x/C671x, the L1P is a 4 Kbyte direct mapped cache with a 64 byte line size and 64 sets. A 32-bit CPU program address is divided into three pieces to determine the physical L1P location where the data can reside. The six least significant bits indicate the byte offset of the first word in the program fetch packet. Since the CPU requires one fetch packet per fetch, the five least significant bits of this offset are ignored and only the upper bit is used to determine which half of the line is sent to the CPU. The next six bits of the program address are used to indicate in which set the data can reside. Since the L1P is a direct mapped cache, each the data for each address can only reside in one of the possible 64 sets. The remaining 20 most significant address bits are used as a unique tag to label what data is currently residing in that cache line. Figure 3–4 illustrates a 32 bit address divided into the tag, set, and offset fields.

Figure 3–4. TMS320C621x/C671x L1P Address Allocation

31 12	11 6	5 0
Tag	Set Index	Offset

For example, the address 0x801ef183 would be split into the following: Offset = 0x3, Set Index = 0x6; Tag = 0x801ef.

The L1P operation is controlled by the CPU control status register (CSR), L1P flush base address register (L1PFBAR), L1P flush word count register (L1PFWC), and the cache configuration register (CCFG). Refer to section 3.5 for a detailed description of the operation of these registers.

An L1P hit completes in one cycle, without stalling the CPU. An L1P miss that hits in L2 stalls the CPU for 5 cycles. An L1P miss that misses in L2 stalls the CPU until the L2 retrieves the data from external memory and transfers the data to the L1P, which then returns the data to the CPU.

3.3.2 L1D Description

The TMS320C621x/C671x L1D is a 4 Kbyte, two-way set associative cache with a 32-byte line size and 64 sets. A physical address from the CPU is divided into four regions to select the appropriate cache set to allocate the data in and to select the correct word from that set. The lower two address bits are a word offset into the address. The next three bits select the word in the set that contains the requested data. The next six bits identify the appropriate set to search for the requested data. The remaining 21 bits are the tag value for the address. Figure 3–5 illustrates a 32 bit address divided into the tag, set, word, and offset fields.

Figure 3–5. TMS320C621x/C671x L1D Address Allocation

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

The L1D operation is controlled by the CPU control status register (CSR), L1D flush base address register (L1DFBAR), L1D flush word count register (L1DFWC) and the cache configuration register (CCFG). Refer to section 3.6 for a detailed description of the operation of these registers.

An L1D hit completes in one cycle, without stalling the CPU. An L1D miss which hits in L2 stalls the CPU for 4 cycles. An L1D miss which misses in L2 stalls the CPU until the L2 retrieves the data from external memory and transfers the data to the L1D, which then returns the data to the CPU.

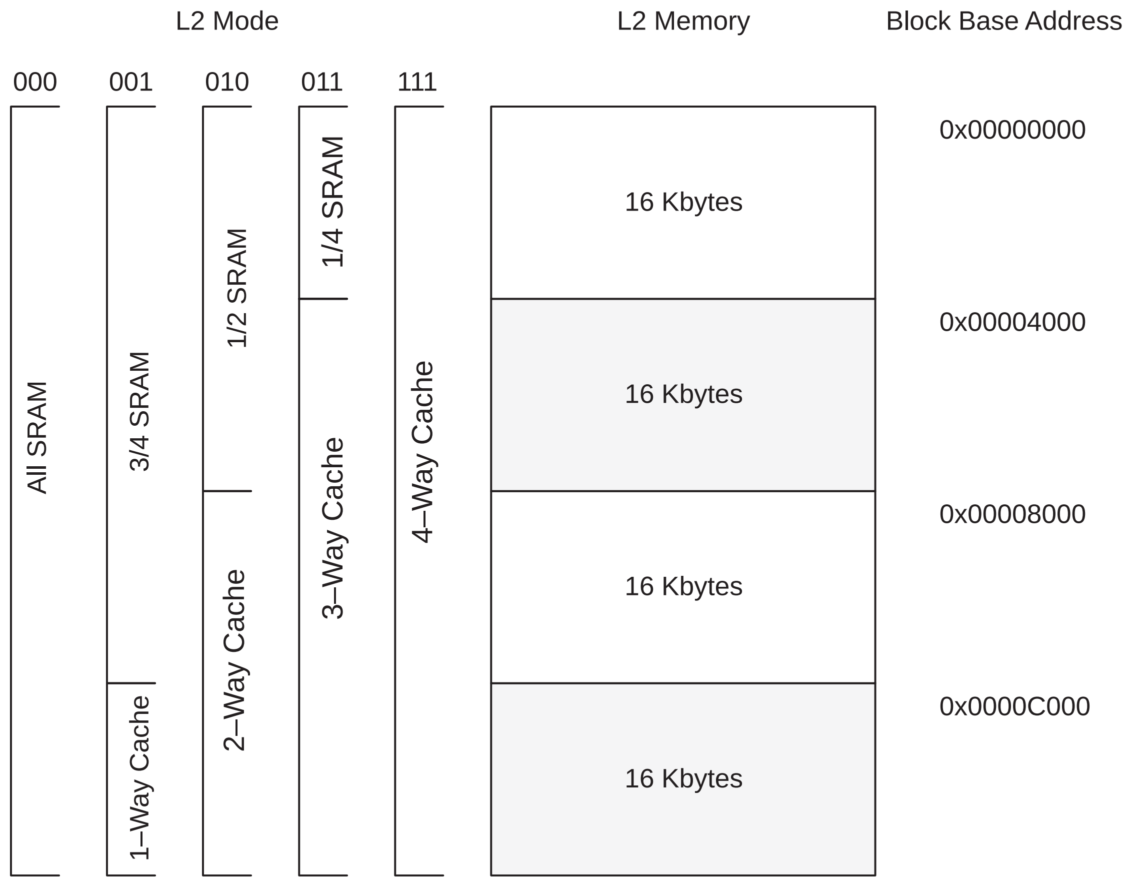
A write buffer exists between the L1D and L2 caches. This buffer can hold the data for up to four write-misses words. As long as this buffer is not full, then new L1D write miss or victim operations can be accepted without stalling the CPU. A L1D write miss, or a snoop that hits an entry in the write buffer, will stall the CPU until the buffer has been flushed.

The write path between the L1D and L2 is 32 bits wide for the C621x/C671x. This path can transmit two 16-bit write misses to different L2 banks in one cycle. If the two write misses are to the same bank, then they will be serialized and transmitted in 2 cycles. An L1D victim write requires 2 cycles, since only one half of the L1D line can be transmitted down the victim path in each cycle.

3.3.3 L2 Description

The L2 operates in four operation modes, depending on the state of the CCFG register. Figure 3–6 shows the division of the L2 SRAM into mapped memory space and cache for each TMS320C621x/C671x L2 Mode. It also shows how the memory configuration for the L2 affects the proportion of cache and SRAM.

Figure 3–6. TMS320C621x/C671x L2 Memory Configuration



In the TMS320C621x/C671x, the L2 is a 64 Kbyte memory that can operate in several modes. The L2 operation is controlled by the cache configuration register (CCFG), the L2 flush base address register (L2FBAR), the L2 flush word count register (L2FWC), the L2 clean base address register (L2CBAR), the L2 clean word count register (L2CWC), the L2 flush register (L2FLUSH), and the L2 clean register (L2CLEAN). Refer to section 3.7 for a detailed description of the operation of these registers and of the L2. Figure 3–7, Figure 3–8, and Figure 3–9 show how the 32-bit address is split into the tag, set index and offset fields.

Figure 3–7. TMS320C621x/C671x L2 Address Allocation 64K/48K Cache (L2MODE = 011b or 111b)

31 — 15	14 — 5	4 — 0
Tag	Set Index	Offset

Figure 3–8. TMS320C621x/C671x L2 Address Allocation 32K Cache (L2MODE = 010b)

31 — 14	13 — 5	4 — 0
Tag	Set Index	Offset

Figure 3–9. TMS320C621x/C671x L2 Address Allocation 16K Cache (L2MODE = 001b)

31 — 13	12 — 5	4 — 0
Tag	Set Index	Offset

3.3.4 TMS320C621x/C671x Data Alignment

The following data alignment restrictions apply:

❑ **Double words –** For C67x, double words are aligned on even 8-byte (double word) boundaries, and always start at a byte address where the three LSBs are 0. Double words are used only on loads triggered by the LDDW instruction. Store operations on the C67x do not use double words. Load and store operations in the C62x do not use double words.

❑ **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.

3.3.5 Control Registers

In addition to the aforementioned cache control registers, the memory attribute registers (MARs) control the TMS320C621x/C671x cache operation. Figure 3–10 shows the format of the memory attribute registers for the TMS320C621x and TMS320C671x devices. Refer to section 3.7.4 for a detailed description of the MARs bit field and its effect on the caches.

Figure 3–10. TMS320C621x/C671x L2 Memory Attribute Registers (MAR0 – MAR15)

31 … 1	0
Reserved	CE
RW,+x	RW,+0

Table 3–2 lists the control registers that affect the TMS320C621x/C671x and their address in the TMS320C621x/C671x memory map.

Table 3–2. TMS320C621x/C671x Internal Memory Control Registers Addresses

Register Address, Bytes	Register Mnemonic	Register Name
01840000h	CCFG	Cache configuration register
01844000h	L2FBAR	L2 flush base address register
01844004h	L2FWC	L2 flush word count register
01844010h	L2CBAR	L2 clean base address register
01844014h	L2CWC	L2 clean word count register
01844020h	L1PFBAR	L1P flush base address register
01844024h	L1PFWC	L1P flush word count register
01844030h	L1DFBAR	L1D flush base address register
01844034h	L1DFWC	L1D flush word count register
01845000h	L2FLUSH	L2 flush register
01845004h	L2CLEAN	L2 clean register
01848200h	MAR0	Controls CE0 range 80000000h – 80FFFFFFh
01848204h	MAR1	Controls CE0 range 81000000h – 81FFFFFFh
01848208h	MAR2	Controls CE0 range 82000000h – 82FFFFFFh
0184820Ch	MAR3	Controls CE0 range 83000000h – 83FFFFFFh

Table 3–2. TMS320C621x/C671x Internal Memory Control Registers Addresses (Continued)

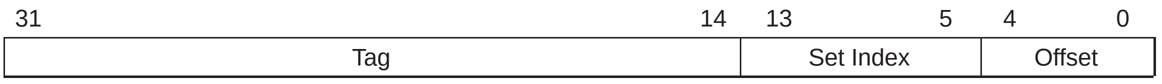
Register Address, Bytes	Register Mnemonic	Register Name
01848240h	MAR4	Controls CE1 range 90000000h – 90FFFFFFh
01848244h	MAR5	Controls CE1 range 91000000h – 91FFFFFFh
01848248h	MAR6	Controls CE1 range 92000000h – 92FFFFFFh
0184824Ch	MAR7	Controls CE1 range 93000000h – 93FFFFFFh
01848280h	MAR8	Controls CE2 range A0000000h – A0FFFFFFh
01848284h	MAR9	Controls CE2 range A1000000h – A1FFFFFFh
01848288h	MAR10	Controls CE2 range A2000000h – A2FFFFFFh
0184828Ch	MAR11	Controls CE2 range A3000000h – A3FFFFFFh
018482C0h	MAR12	Controls CE3 range B0000000h – B0FFFFFFh
018482C4h	MAR13	Controls CE3 range B1000000h – B1FFFFFFh
018482C8h	MAR14	Controls CE3 range B2000000h – B2FFFFFFh
018482CCh	MAR15	Controls CE3 range B3000000h – B3FFFFFFh

3.4 TMS320C64x Two-Level Memory

3.4.1 L1P Description

The L1P on the C64x is a 16 Kbyte direct mapped cache with a 32-byte line size and 512 sets. A 32-bit CPU program address is divided into three pieces to determine the physical L1P location where the data can reside. The five least significant bits indicate the byte offset of the program fetch packet. The next nine bits of the program address are used to indicate in which set the data can reside. Since the L1P is a direct mapped cache, each address's data can only reside in one of the possible 512 sets. The remaining 18 most significant address bits are used as a unique tag to label what data is currently residing in that cache line. Figure 3–11 illustrates a 32-bit address divided into the tag, set, and offset fields.

Figure 3–11. TMS320C64x L1P Address Allocation



The L1P operation is controlled by the CPU control status register (CSR), the L1P flush base address register (L1PFBAR), the L1P flush word count register (L1PFWC) and the cache configuration register (CCFG). Refer to section 3.5 for a detailed description of the operation of these registers.

An L1P hit completes in one cycle, without stalling the CPU. An L1P miss which hits in L2 stalls the CPU from 0 to 7 cycles, depending on the parallelism of the execute packet and the execute phase of the pipeline when the miss occurs. (For more information on the phase of the C6000 pipeline refer to the *SPRU189 TMS320C6000 CPU Instruction Set Reference Guide,* which is described in the *Related Documentation from Texas Instruments* section.)

The performance of the pipelining is constant, regardless of the alignment of the execute packets. On the C64x architecture, execute packets can span fetch packet boundaries. For example, three execute packets of 5, 5 and 6 instructions only occupy two fetch packets.

An L1P miss that misses in L2 cache stalls the CPU until the L2 retrieves the data from external memory and transfers the data to the L1P, which then returns the data to the CPU. This delay will depend upon the type of external memory used to hold the external program.

3.4.2 L1D Description

The TMS320C64x L1D is a 16 Kbyte cache, with a 64-bit wide write bus from the L1D to the L2 memory. It is a two-way set associative cache with a 64 byte line size and 128 sets. A physical address from the CPU is divided into four regions to select the appropriate cache set to allocate the data in and to select the correct word from that set. The lower two address bits are a word offset into the address. The next four bits select the word in the set that contains the requested data. The next seven bits identify the appropriate set to search for the requested data. The remaining 20 bits are the tag value for the address. Figure 3–12 illustrates a 32-bit address divided into the tag, set, word, and offset fields.

Figure 3–12. TMS320C64x L1D Address Allocation

31 13	12 6	5 2	1 0
Tag	Set Index	Word	Offset

The L1D operation is controlled by the CPU control status register (CSR), the L1D flush base address register (L1DFBAR), the L1D flush word count register (L1DFWC) and the cache configuration register (CCFG). Refer to section 3.6 for a detailed description of the operation of these registers.

An L1D hit completes in one cycle, without stalling the CPU. An L1D miss which hits in L2 stalls the CPU for between 2 and 8 cycles. The 2 cycle penalty refers to the steady state case of the average penalty for a large number of continuous misses (See section 3.4.3). The worst case of a single miss in L1D, where L2 is configured as cache and there is a L2 cache hit causes a 8 cycle stall. An L1D miss which misses in L2 stalls the CPU until the L2 retrieves the data from external memory and transfers the data to the L1D, which then returns the data to the CPU. The external miss penalty will vary depending on the type and width of external memory used to hold external data.

The L1D is a read allocate cache only. Any write misses to L1D are passed on to L2 and do not cause a line to be fetched into L1D.

When a read miss occurs that causes a victim to occur of a dirty line, the least recently used line is written back to L2 and then overwritten in L1D. This write-back operation can be pipelined with the read request and other outstanding write-backs.

If there are two accesses to a line in the same cycle, only one miss penalty will occur. Similarly if there are two accesses in succession to the same line, and the first one is a miss, the second access will not incur any additional miss penalty.

3.4.3 Pipelining Cache Misses

Unlike the C621x/C671x architecture, the C64x architecture provides a mechanism to reduce the penalty for first level cache misses. This technique is known as “miss pipelining”. If misses occur in clusters, the retrieval of the missed lines can be overlapped. This amortizes the overhead of a cache miss and causes the average miss penalty to be significantly less than that of an isolated miss. An isolated cache miss causes a CPU stall of 8 clock cycles. If another cache miss occurs in either the same clock cycle or in the next clock cycle, the number of stalls incurred will not be 8, as in the first miss, but only 2. This technique reduces the average stalls per miss from 8 clock cycles to 5 clock cycles. The averaging effect improves when there are more cache misses in parallel. This technique is used in both L1P and L1D and is explained in sections 3.5.1 and 3.6.3 respectively.

3.4.4 Memory Banking Structure

The C64x has a different memory banking structure than that of the C621x/C671x. The C621x/C671x least significant bit (LSB) banks are on 16-bit boundaries. Therefore any memory accesses where the 2 LSBs of the addresses are the same will access the same bank. If these memory accesses are in parallel, the CPU will stall for one cycle. On the C64x the memory banks are on 32-bit boundaries. Therefore any memory accesses where the 3 LSBs of the addresses are the same will access the same bank. C64x memory accesses to the same bank will cause a stall except in the following cases:

- ❑ The accesses are to the same bank, but they are in the same L1D cache line. In other words, they are adjacent accesses to the same 32-bit address, but the 2 LSBs are not the same.
- ❑ There will be no stall if the parallel accesses are 8 or 16 bits wide to the same 32 bit bank, provided the accesses have different 2 LSBs. On the C621x/C671x, any parallel 8-bit wide accesses to the same 16-bit bank but with different first LSB would cause a stall. This is not the case on the C64x.
- ❑ Non-aligned accesses will never cause a memory bank stall as they cannot occur in parallel.

3.4.5 Memory Ordering Support

The C64x enforces a strong memory ordering model in the CPU and its local memories such as L1D, and L2. The L1P is read only and thus its memory ordering with respect to L1D and L2 is irrelevant. Strong memory ordering enforces the order in which the data is written to and read from memory. Strong memory ordering enforces the exact same order in which the data is requested in the program flow. This program order is defined in section 3.2 of the cache definitions.

This ordering restriction guarantees correctness of the program execution and any complex interactions with peripherals. This currently includes any host processor interactions with the local memory (L2). Section 3.4.6 describes certain C64x stall cases that are not implemented on the C621x/C671x.

3.4.6 L1D – L2 Write Buffer

A write buffer exists between the L1D and L2 caches. There can be up to four non-mergeable write misses outstanding in the write buffer without stalling the CPU. If a write miss occurs when the write buffer is full, the CPU will stall until the L2 is no longer busy processing read or write misses. This is done to guarantee correct memory ordering requirements.

The write buffer allows merging of write requests. It will merge two write misses into a single transaction providing the following rules are obeyed:

- The double-word address of the two accesses is the same
- The two accesses are to L2 configured as SRAM
- The oldest write has just been placed in the write buffer queue
- The newest write has not been placed in the buffer.queue

When two stores are adjacent and L2 is configured as SRAM, the two stores are merged into a single double word store. This case is quite common. By merging smaller writes together, the likelihood of a full store buffer decreases. This in turn reduces the chance of a CPU stall taking place.

Once an access has been written into the write buffer, it is invisible to the L2 snoop hardware. The snoop hardware cannot see what is in the write buffer, so it waits until the buffer empties. This behavior prevents memory ordering violations. The L2 snoop hardware also queries what data is present in L1D; this prevents data coherency problems.

Here are some examples of the behaviors mentioned above:

1) A write miss occurs in L1D and this miss is passed into the write buffer. In a subsequent cycle, a write hit occurs in L1D. The L2 cannot snoop the write buffer so both the CPU and L1D hit is stalled until the write buffer has emptied. The ordering rule being observed here is that all writes must occur in the order that they are dispatched in the program flow.

2) A write miss occurs in L1D and is passed into the write buffer. In the next cycle, a read miss occurs in L1D. The CPU and L1D will stall until the write buffer has been emptied. The strong memory ordering rule being observed here is that writes have to complete before reads.

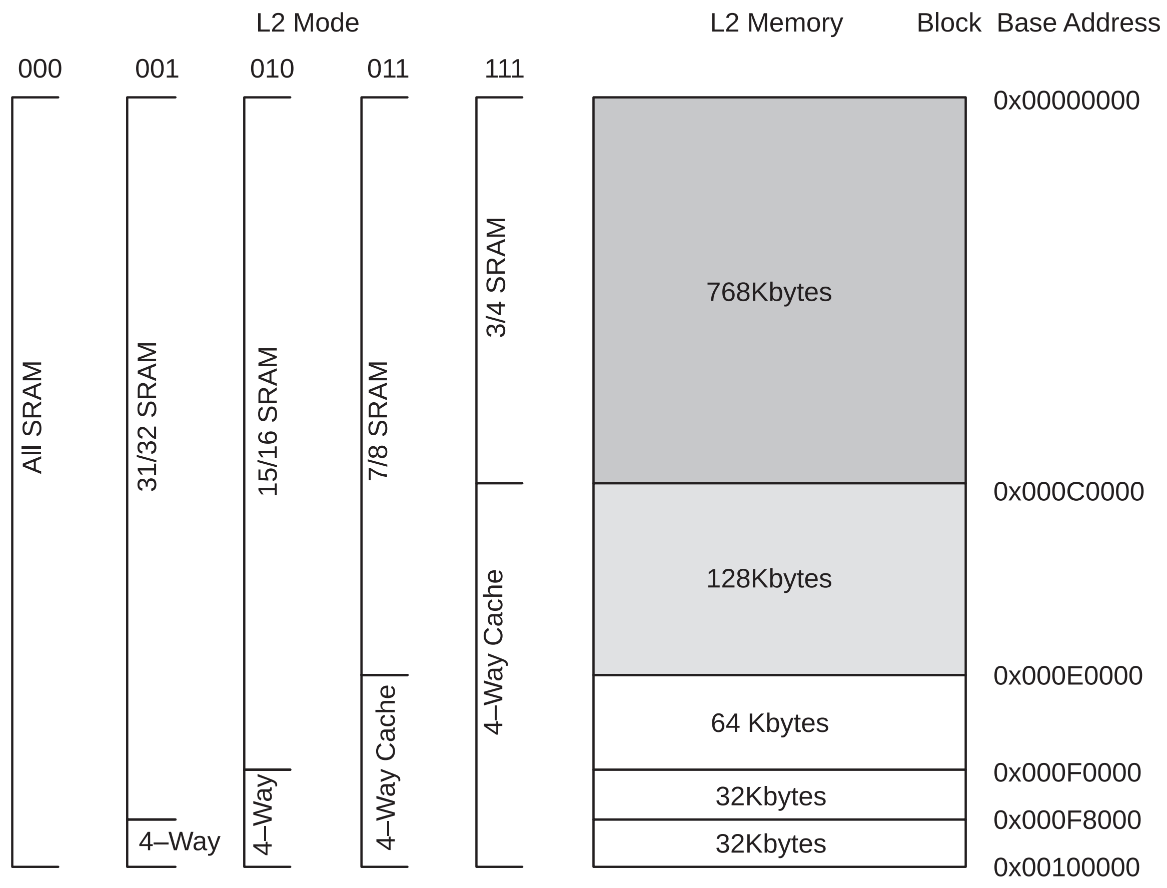
3) A host CPU writes to an area of L2 memory. The L2 will snoop L1D for the memory addresses sought by the host and will invalidate the relevant lines in L1D.

4) A host CPU reads an area of L2 memory. The L2 will snoop L1D for the memory addresses sought by the host and will copy back the data from the lines that contain the data for the addresses sought by the host CPU.

3.4.7 L2 Description

The L2 operates in five modes, depending on the state of the CCFG register. Figure 3–13 shows the division of the L2 SRAM into mapped memory space and cache for each TMS320C64x L2 Mode.

Figure 3–13. TMS320C64x L2 Memory Configuration



The C64x L2 is a 1024 Kbyte memory that can operate in several modes. The L2 operation is controlled by the cache configuration register(CCFG), the L2 flush base address register (L2FBAR), the L2 flush word count register (L2FWC), the L2 clean base address register (L2CBAR), the L2 clean word count register (L2CWC), the L2 flush register (L2FLUSH), and the L2 clean register (L2CLEAN). Refer to section 3.7 for a detailed description of the operation of these registers and of the L2. Figure 3–14, Figure 3–15, Figure 3–16, and Figure 3–17 show how the 32-bit address is separated out into the tag, set index, and offset fields for the various CCFG modes.

Figure 3–14. TMS320C64x L2 Address Allocation, 256K Cache (L2MODE = 111b)

31 17	16 5	4 0
Tag	Set Index	Offset

Figure 3–15. TMS320C64x L2 Address Allocation, 128K Cache (L2MODE = 011b)

31 16	15 5	4 0
Tag	Set Index	Offset

Figure 3–16. TMS320C64x L2 Address Allocation, 64K Cache (L2MODE = 010b)

31 15	14 5	4 0
Tag	Set Index	Offset

Figure 3–17. TMS320C64x L2 Address Allocation, 32K Cache (L2MODE = 001b)

31 14	13 5	4 0
Tag	Set Index	Offset

3.4.8 TMS320C64x Data Alignment

The following data alignment restrictions apply:

❑ **Doublewords –** The C64x can access double words on any byte boundary. For doublewords aligned on an even 8-byte boundary an LDDW instruction is used for loads and an STDW is used for double word stores. For doublewords not aligned on an even 8-byte boundary, the C64x can use the LDNDW and STNDW instructions for loads and stores respectively.

❑ **Words –** The C64x can access words on any byte boundary. For words aligned on an even 4-byte (word) boundary an LDW instruction is use for loads and an STW is used for stores. Words are aligned on even 4-byte (word) boundaries, and always start at a byte address where the two LSBs are 0. For words not aligned on an even 4-byte boundary, the C64x can use the LDNW and STNW instructions for loads and stores respectively.

❑ **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.

Note: The LDNDW, STNDW, LDNW, STNW instructions can occur on any byte boundary, but only one of these instructions can occur per cycle. A single non–aligned access can cause two L1D misses. If a non-aligned access occurs that has parts from different lines, these two accesses are not forced to be strongly ordered with respect to one another.

3.4.9 Control Registers

The L1P and L1D caches may need to change modes more frequently within an application than the L2. Their functionality is controlled via the PCC and DCC bits in the internal CPU control status register(CSR). Figure 3–18 shows the control status register (CSR). Section 3.5 and section 3.6 explain the operation of the PCC and DCC bits, respectively.

Figure 3–18. TMS320C64x Control Status Register (CSR)

31 24	23 16
CPU-ID	Revision ID
R	R

15 10	9	8	7 5	4 2	1	0
PWRD	SAT	EN	PCC	DCC	PGIE	GIE
RW,+0	R,C,+0	R,+0	RW,+0	RW,+0	RW,+0	RW,+0

Along with other cache control registers, the memory attribute registers (MARs) control the C64x cache operation. Figure 3–19 shows the format of the memory attribute registers for the C64x. Refer to section 3.7.4 for a detailed description of the MARs bit field and its effects on the caches. Bit CE allows the address range to be cached in L2 cache.

Figure 3–19. TMS320C64x L2 Memory Attribute Registers (MAR0 – MAR255)

31 1	0
Reserved	CE
RW,+x	RW,+0

Table 3–3 shows the MAR registers for the various ranges in the entire memory map.

The C64x MAR registers are compatible with the C621x/C671x MAR registers. They have the same register address corresponding to the same memory range. For example, MAR128 on C64x at address 0x01848200 controls address range 80000000h to 80FFFFFFh. On C621x/C671x this same register was named MAR0.

The L2 allocation registers (L2ALLOC0, L2ALLOC1, L2ALLOC2, and L2ALLOC3) shown in Figure 3–20 allow the user to specify the maximum number of outstanding L2 transfer requests on each priority level transfer request queue. The defaults for the L2 allocate field (L2ALLOC) of the L2ALLOC0, L2ALLOC1, L2ALLOC2 and L2ALLOC3 registers are 6,2,2,2 respectively. Users can specify which of the transfer priority levels the L2 should use for all cache related accesses by programming the P field of the CCFG register (section 3.7). For example, to use the high priority level (Queue 1) for L2 controller transfers, P must be set to 001b and the L2ALLOC field in the L2ALLOC1 must be set to a non-zero value. See *Chapter 6, EDMA Controller*, for details.

Figure 3–20. TMS320C64x L2 Allocation Registers (L2ALLOC0 – L2ALLOC3)

L2ALLOC0

31 – 3	2 – 0
Reserved	L2ALLOC
R,+0	RW,+110

L2ALLOC1

31 – 3	2 – 0
Reserved	L2ALLOC
R,+0	RW,+010

L2ALLOC2

31 – 3	2 – 0
Reserved	L2ALLOC
R,+0	RW,+010

L2ALLOC3

31 – 3	2 – 0
Reserved	L2ALLOC
R,+0	RW,+010

Table 3–3 lists all the C64x control registers.

Table 3–3. TMS320C64x Internal Memory Control Registers Addresses

Register Address, Bytes	Register Mnemonic	Register Description
01840000h	CCFG	Cache configuration register
01842000h	L2ALLOC0	L2 allocation register 0
01842004h	L2ALLOC1	L2 allocation register 1
01842008h	L2ALLOC2	L2 allocation register 2
0184200Ch	L2ALLOC3	L2 allocation register 3
01844000h	L2FBAR	L2 flush base address register
01844004h	L2FWC	L2 flush word count register
01844010h	L2CBAR	L2 clean base address register
01844014h	L2CWC	L2 clean word count register
01844020h	L1PFBAR	L1P flush base address register
01844024h	L1PFWC	L1P flush word count register
01844030h	L1DFBAR	L1D flush base address register
01844034h	L1DFWC	L1D flush word count register
01845000h	L2FLUSH	L2 flush register
01845004h	L2CLEAN	L2 clean register
01848000h to 0184817Ch	MAR0 to MAR95	Reserved
01848180h	MAR96	Controls EMIFB CE0 range 60000000h – 60FFFFFFh
01848184h	MAR97	Controls EMIFB CE0 range 61000000h – 61FFFFFFh
01848188h	MAR98	Controls EMIFB CE0 range 62000000h – 62FFFFFFh
0184818Ch	MAR99	Controls EMIFB CE0 range 63000000h – 63FFFFFFh
01848190h	MAR100	Controls EMIFB CE1 range 64000000h – 64FFFFFFh
01848194h	MAR101	Controls EMIFB CE1 range 65000000h – 65FFFFFFh
01848198h	MAR102	Controls EMIFB CE1 range 66000000h – 66FFFFFFh
0184819Ch	MAR103	Controls EMIFB CE1 range 67000000h – 67FFFFFFh
018481A0h	MAR104	Controls EMIFB CE2 range 68000000h – 68FFFFFFh
018481A4h	MAR105	Controls EMIFB CE2 range 69000000h – 69FFFFFFh
018481A8h	MAR106	Controls EMIFB CE2 range 6A000000h – 6AFFFFFFh
018481ACh	MAR107	Controls EMIFB CE2 range 6B000000h – 6BFFFFFFh
018481B0h	MAR108	Controls EMIFB CE3 range 6C000000h – 6CFFFFFFh
018481B4h	MAR109	Controls EMIFB CE3 range 6D000000h – 6DFFFFFFh
018481B8h	MAR110	Controls EMIFB CE3 range 6E000000h – 6EFFFFFFh
018481BCh	MAR111	Controls EMIFB CE3 range 6F000000h – 6FFFFFFFh
018481C0h to 018481FCh	MAR112 to MAR127	Reserved

Table 3–3. TMS320C64x Internal Memory Control Registers Addresses (Continued)

Register Address, Bytes	Register Mnemonic	Register Description
01848200h	MAR128	Controls EMIFA CE0 range 80000000h – 80FFFFFFh
01848204h	MAR129	Controls EMIFA CE0 range 81000000h – 81FFFFFFh
01848208h	MAR130	Controls EMIFA CE0 range 82000000h – 82FFFFFFh
0184820Ch	MAR131	Controls EMIFA CE0 range 83000000h – 83FFFFFFh
01848210h	MAR132	Controls EMIFA CE0 range 84000000h – 84FFFFFFh
01848214h	MAR133	Controls EMIFA CE0 range 85000000h – 85FFFFFFh
01848218h	MAR134	Controls EMIFA CE0 range 86000000h – 86FFFFFFh
0184821Ch	MAR135	Controls EMIFA CE0 range 87000000h – 87FFFFFFh
01848220h	MAR136	Controls EMIFA CE0 range 88000000h – 88FFFFFFh
01848224h	MAR137	Controls EMIFA CE0 range 89000000h – 89FFFFFFh
01848228h	MAR138	Controls EMIFA CE0 range 8A000000h – 8AFFFFFFh
0184822Ch	MAR139	Controls EMIFA CE0 range 8B000000h – 8BFFFFFFh
01848230h	MAR140	Controls EMIFA CE0 range 8C000000h – 8CFFFFFFh
01848234h	MAR141	Controls EMIFA CE0 range 8D000000h – 8DFFFFFFh
01848238h	MAR142	Controls EMIFA CE0 range 8E000000h – 8EFFFFFFh
0184823Ch	MAR143	Controls EMIFA CE0 range 8F000000h – 8FFFFFFFh
01848240h	MAR144	Controls EMIFA CE1 range 90000000h – 90FFFFFFh
01848244h	MAR145	Controls EMIFA CE1 range 91000000h – 91FFFFFFh
01848248h	MAR146	Controls EMIFA CE1 range 92000000h – 92FFFFFFh
0184824Ch	MAR147	Controls EMIFA CE1 range 93000000h – 93FFFFFFh
01848250h	MAR148	Controls EMIFA CE1 range 94000000h – 94FFFFFFh
01848254h	MAR149	Controls EMIFA CE1 range 95000000h – 95FFFFFFh
01848258h	MAR150	Controls EMIFA CE1 range 96000000h – 96FFFFFFh
0184825Ch	MAR151	Controls EMIFA CE1 range 97000000h – 97FFFFFFh
01848260h	MAR152	Controls EMIFA CE1 range 98000000h – 98FFFFFFh
01848264h	MAR153	Controls EMIFA CE1 range 99000000h – 99FFFFFFh
01848268h	MAR154	Controls EMIFA CE1 range 9A000000h – 9AFFFFFFh
0184826Ch	MAR155	Controls EMIFA CE1 range 9B000000h – 9BFFFFFFh
01848270h	MAR156	Controls EMIFA CE1 range 9C000000h – 9CFFFFFFh
01848274h	MAR157	Controls EMIFA CE1 range 9D000000h – 9DFFFFFFh
01848278h	MAR158	Controls EMIFA CE1 range 9E000000h – 9EFFFFFFh
0184827Ch	MAR159	Controls EMIFA CE1 range 9F000000h – 9FFFFFFFh
01848280h	MAR160	Controls EMIFA CE2 range A0000000h – A0FFFFFFh
01848284h	MAR161	Controls EMIFA CE2 range A1000000h – A1FFFFFFh

Table 3–3. TMS320C64x Internal Memory Control Registers Addresses (Continued)

Register Address, Bytes	Register Mnemonic	Register Description
01848288h	MAR162	Controls EMIFA CE2 range A2000000h – A2FFFFFFh
0184828Ch	MAR163	Controls EMIFA CE2 range A3000000h – A3FFFFFFh
01848290h	MAR164	Controls EMIFA CE2 range A4000000h – A4FFFFFFh
01848294h	MAR165	Controls EMIFA CE2 range A5000000h – A5FFFFFFh
01848298h	MAR166	Controls EMIFA CE2 range A6000000h – A6FFFFFFh
0184829Ch	MAR167	Controls EMIFA CE2 range A7000000h – A7FFFFFFh
018482A0h	MAR168	Controls EMIFA CE2 range A8000000h – A8FFFFFFh
018482A4h	MAR169	Controls EMIFA CE2 range A9000000h – A9FFFFFFh
018482A8h	MAR170	Controls EMIFA CE2 range AA000000h – AAFFFFFFh
018482ACh	MAR171	Controls EMIFA CE2 range AB000000h – ABFFFFFFh
018482B0h	MAR172	Controls EMIFA CE2 range AC000000h – ACFFFFFFh
018482B4h	MAR173	Controls EMIFA CE2 range AD000000h – ADFFFFFFh
018482B8h	MAR174	Controls EMIFA CE2 range AE000000h – AEFFFFFFh
018482BCh	MAR175	Controls EMIFA CE2 range AF000000h – AFFFFFFFh
018482C0h	MAR176	Controls EMIFA CE3 range B0000000h – B0FFFFFFh
018482C4h	MAR177	Controls EMIFA CE3 range B1000000h – B1FFFFFFh
018482C8h	MAR178	Controls EMIFA CE3 range B2000000h – B2FFFFFFh
018482CCh	MAR179	Controls EMIFA CE3 range B3000000h – B3FFFFFFh
018482D0h	MAR180	Controls EMIFA CE3 range B4000000h – B4FFFFFFh
018482D4h	MAR181	Controls EMIFA CE3 range B5000000h – B5FFFFFFh
018482D8h	MAR182	Controls EMIFA CE3 range B6000000h – B6FFFFFFh
018482DCh	MAR183	Controls EMIFA CE3 range B7000000h – B7FFFFFFh
018482E0h	MAR184	Controls EMIFA CE3 range B8000000h – B8FFFFFFh
018482E4h	MAR185	Controls EMIFA CE3 range B9000000h – B9FFFFFFh
018482E8h	MAR186	Controls EMIFA CE3 range BA000000h – BAFFFFFFh
018482ECh	MAR187	Controls EMIFA CE3 range BB000000h – BBFFFFFFh
018482F0h	MAR188	Controls EMIFA CE3 range BC000000h – BCFFFFFFh
018482F4h	MAR189	Controls EMIFA CE3 range BD000000h – BDFFFFFFh
018482F8h	MAR190	Controls EMIFA CE3 range BE000000h – BEFFFFFFh
018482FCh	MAR191	Controls EMIFA CE3 range BF000000h – BFFFFFFFh
01848100h to 018481FCh	MAR192 to MAR255	Reserved

3.5 L1P Operation

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/010b. All other values for PCC are reserved, as shown Table 3–4.

Table 3–4. Level 1 Program Cache Mode Setting

Cache Mode	PCC Value	Description
Cache enable	010/000	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.

There are two methods for user controlled invalidation of data in the L1P. With the first method, writing a 1 to the IP bit of the cache configuration register (CCFG) invalidates all cache tags in the L1P tag RAM. This is a write-only bit, and 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. The CCFG is shown in Figure 3–28 and described in Table 3–9.

The second method for invalidating the L1P requires the L1PFBAR register and L1PFWC register. This is useful for invalidating a block of data in the L1P. The user must first write a word-aligned address into the L1PFBAR register. This value is the starting address for the invalidation. The number of words invalidated is equal to the value written into the L1PFWC register. The L1P searches for, and invalidates, all lines whose memory address falls within the range from L1PFBAR to L1PFBAR+L1PFWC–1. If L1PFBAR or L1PFWC are not aligned to the L1P line size, 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 the user should take care to ensure that the L1PFBAR register is set up correctly prior to writing the L1PFWC. When the entire invalidation is complete, the L1PFWC register will contain the value 0. The L1PFBAR and L1PFWC are shown in Figure 3–21 and Figure 3–22, respectively. Refer to section 3.7.11 for a summary of this L1P flush operation.

Figure 3–21. L1P Flush Base Address Register (L1PFBAR)

31 … 0
L1P Flush Base Address
RW,+x

Figure 3–22. L1P Flush Word Count Register (L1PFWC)

31 … 16	15 … 0
Reserved	L1P Flush Word Count
R,+x	RW,+x

3.5.1 TMS320C64x L1P Miss Pipelining

The C64x L1P cache supports the pipelining of misses. When there is an isolated L1P miss, the CPU is stalled for 8 cycles. However, the overhead of fetching the missed line from L1P can be spread out if there are multiple misses.

Before discussing the effect of pipelining multiple misses, let us first examine the effect of the number of execute packets in a fetch packet for a single miss. A fetch packet consists of eight instructions. An execute packet is a set of instructions that are executed in parallel. When an execute packet equals a fetch packet, eight instructions execute in a single cycle. This case illustrates one cycle of execution per fetch packet.

More than one execute packet can reside in a fetch packet. For example, you can have a fetch packet that consists of two execute packets and each execute packet contains four instructions. In this case there are two cycles of execution in the fetch packet of eight instructions. One of the two execution cycles per fetch packet replaces one of the stall cycles of the missed fetch packet.

As more execute packets are placed in each fetch packet, more stalls are overlapped with the execute cycles. For the case where there are eight execute packets per fetch packet or one instruction executed per cycle, no L1P misses will be observed by the user. But if there are fewer execute packets per fetch packet while number of instructions executed per cycle increases, the number of stalls per cycle also increases. This increase is a non-linear function of the number of instructions executed per cycle.

Now let us look at multiple misses in L1P. When multiple misses in L1P occur, they are submitted to the L1P cache controller so that, in the limit, the user will see only 3 misses per cycle, as opposed to 8 for an isolated L1P miss. More than two instructions executed per cycle will yield one or more stalls per execute cycle. This is summarized in Table 3–6.

Table 3–5. Miss Penalties for Large Numbers of Sequential Execute Packets

Instructions Per Execute Packet	Average Stalls Per Execute Packet
1	0.00
2	0.25
3	0.84
4	1.68
5	2.10
6	2.72
7	3.34
8	4.00

3.6 L1D Operation

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 of the CPU control and status register (CSR) are 000b and 010b. All other values for DCC are reserved, as shown in Table 3–6.

Table 3–6. Level 1 Data Cache Mode Settings

Cache Mode	DCC Value	Description
Cache enable	000/010	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.

Operation on a cache miss depends upon 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 uses the fetch address to decode the set mapped to by the data. The L1D controller allocates space in the line that was least recently used and stores the new data into that line. If the data in the allocated line has been modified but the corresponding address has not been updated (because the cache line is dirty), that data is evicted 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 the data in the allocated line has not been modified (because the cache line is valid but not dirty) or the cache line is invalid, the new data is simply written into the allocated line. If two read misses occur in the same cycle, they are serialized by the L1D so that only one request at a time is presented to the L2. The mechanism of miss pipelining is relevant here, as the miss penalty of the two misses is overlapped to reduce the average per miss.

3.6.1 Read Allocate

The L1D is a read-allocate cache. 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 is also a write-back cache. Write hits in the L1D are stored in the L1D and no access is performed to the L2 in this case, which causes the L1D to be dirty. If this line is evicted in a future access then the dirty data is updated in the L2. In some cases the user may need to force modified data out of the L1D cache via user-controlled invalidation. One instance where this would be necessary is a context switch, where all modified data would be moved to external memory. Since the L1D and L2 could be incoherent due to write hits in the L1D, the user should perform an L1D invalidation to force any dirty L1D data into the L2.

3.6.2 L1D Invalidation

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 cache tags in the L1D tag RAM. This is a write-only bit, a read of this bit will always return 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 register and L1DFWC register. This is useful for invalidating a block of data in the L1D. The user 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 invalidates, all lines whose memory address falls within the range from L1DFBAR to L1DFBAR+L1DFWC-1. 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, 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-1 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 the user should take care to ensure that the L1DFBAR register is set up correctly prior to writing the L1DFWC. When the invalidation is complete, the L1DFWC register will contain the value 0. The second method is preferred for writing data that has been cached in the L1D to the external memory space. The L1DFBAR and L1DFWC are shown in Figure 3–23 and Figure 3–24.

Figure 3–23. L1D Flush Base Address Register (L1DFBAR)

31	0
L1D Flush Base Address	
RW,+x	

Figure 3–24. L1D Flush Word Count Register (L1DFWC)

31	16	15	0
Reserved		L1D Flush Word Count	
R,+x		RW,+x	

3.6.3 TMS320C64x L1D Miss Pipelining

The C64x L1D supports pipelining misses. This means that if multiple misses occur in the same cycle or in adjacent cycles, the overhead of a single miss can be spread out over multiple misses. The initial miss incurs all the overhead of a miss penalty. This includes all cache tag reads and address resolutions (to check for cache hits), as well as RAM accesses and L2 requests. If more than one miss occurs in the same cycle, the second miss request can begin directly after the first. If more than two misses occur, a pipeline fills with various stages of these multiple misses. This is shown in Figure 3–25 and Figure 3–26. Once this pipeline has been totally filled, the incremental cost of a new miss averages only 2.0 cycles per miss. This compares with six cycles for a single miss to L2 SRAM, and eight cycles for a single miss to L2 Cache.

Figure 3–25 shows four L1D misses when L2 is configured as SRAM. The pipeline signals are explained in Figure 3–27. In this scenario, the CPU requests data in clock cycle 0 for read1 and read 2. In clock cycle 1 the data is looked for in L1D. The data is not present in L1D so in cycle 2 a miss is recorded for both read1 and read2. Also in cycle 1, the CPU requests the data for read3 and read4. In cycle 3, there is an L2 request for the data for read1 and a miss is recorded for both read3 and read4. In cycles 4, 5 and 6 there are L2 requests for the data for read2, read3, and read4 respectively. In cycles 7 and 8, the data for read1 is found in L2 and placed in L1. In cycles 9 and 10, the data for read2 is found in L2 and placed in L1. In cycle 11, the data from read1 and read2 is placed in the register file. Also, in cycle 11 and in 12, the data from read3 is found in L2 and placed in L1. In cycles 13 and 14, the data from read4 is found in L2 and placed in L1. In cycle 15, the data from read3 and read4 is placed in the register file. For these four misses, the CPU was stalled for a total of 12 clock cycles. The average of this is 3 cycles, instead of 6 cycles, for a single miss.

Figure 3–26 shows four L1D misses when the L2 segment is configured as cache. The pipeline signals are explained in Figure 3–27. In this scenario, the CPU requests data in clock cycle 0 for read1 and read 2. In clock cycle 1 the data is looked for in L1D. The data is not present in L1D so in cycle 2 a miss is recorded for both read1 and read2. Also in cycle 1 the CPU requests the data for read3 and read4. In cycle 3, there is an L2 request for the data for read1 and a miss is recorded for both read3 and read4. In cycles 4, 7 and 8 there are L2 requests for the data for read2, read3, and read4 respectively. In cycles 9 and 10, the data for read1 is found in L2 and placed in L1. In cycles 11 and 12, the data for read2 is found in L2 and placed in L1. In cycle 13, the data from read1 and read2 is placed in the register file. Also in cycle 13 and in cycle 14 the data from read3 is found in L2 and placed in L1. In cycles 15 and 16 the data from read4 is found in L2 and placed in L1. In cycle 17, the data from read3 and read4 is placed in the register file. For these four misses, the CPU was stalled for a total of 14 clock cycles. This averages 4.67 cycles instead of 8 cycles for a single miss.

Figure 3–25. L1D – L2 SRAM 4 Read Miss Pipeline

Clock Cycle	2	3	4	5	6	7	8	9	10	11	12	13	14	15
Read1	Miss1	L2Req1				Data1a	Data1b			RFwr				
Read2	Miss2		L2req2					Data2a	Data2b	RFwr				
Read3	L1 Tagrd	Miss3		L2req3						Data3a	Data3b			RFwr
Read4	L1 Tagrd	Miss4			L2req4							Data4a	Data4b	RFwr
CPU Stalls	0	1	1	1	1	1	1	1	1	1	1	1	1	0

Figure 3–26. L1D – L2 CACHE 4 Read Miss Pipeline

Clock Cycle	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17
Read1	Miss1	L2Req1	L2 Tagrd	Addrgen				Data1a	Data1b			RFwr				
Read2	Miss2		L2req2	L2 Tagrd	Addrgen					Data2a	Data2b	RFwr				
Read3	L1 Tagrd	Miss3				L2req3	L2 Tagrd	Addrgen				Data3a	Data3b			RFwr
Read4	L1 Tagrd	Miss4						L2req4	L2 Tagrd	Addrgen				Data4a	Data4b	RFwr
CPU Stalls	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	0

Figure 3–27. Key to Read Miss Pipeline Signals

Signal	Meaning	Signal	Meaning
Addrgen	Address Generation	L2reqn	L2 memory request signal
L1 Tagrd	L1 Tag Memory Read	RFwr	Register file write
		Missn	Cycle to declare a miss
Data	Data Read	L2 Tagrd	L2 tag memory read
CPU Stage	CPU execution pipeline stage	Readn	Actual CPU action; LDH, LDB, etc.

Using the pipeline diagrams in Figure 3–25 and Figure 3–26, Table 3–7 and Table 3–8 can be drawn up for a varying number of intermediate misses, M. These show how, as the number of parallel and consecutive misses increases, the number of stalls changes. For small numbers of misses, there is a larger overhead for each miss. However as the number of misses increases, each new miss causes a smaller overhead.

Table 3–7. Cycles Per Miss for Different Numbers of Misses to L1D from L2 Cache

Number of Misses	Total Stall Cycles Take	Mean Cycles Per Miss
1	8	8
2	10	5.0
3	12	4.0
4	14	3.5
>4 even	$6 + 2 \times M$†	$2 + 6/M$†

† M = Number of intermediate misses.

Table 3–8. Cycles Per Miss for Different Numbers of Misses to L1D from L2 SRAM

Number of Misses	Total Stall Cycles Take	Mean Cycles Per Miss
1	6	8
2	8	4.5
3	10	3.33
4	12	3.0
>4 even	$4 + 2 \times M$†	$2 + 4/M$†

† M = Number of intermediate misses.

The configuration of L2 (whether it is SRAM or cache) will affect the miss penalty of an access to L1D. If misses can be queued, the miss penalty is reduced in both cases. For example, let's assume that there are 32 misses (M = 32), all occurring in a single block. By substituting M into the equation above, the average miss penalty is only 2.21 cycles per miss when L2 is cache, and 2.12 cycles per miss when L2 is SRAM. If there is a miss in L2 cache, the number of cycles will depend on the external memory type or peripheral used and its bandwidth connection to the external memory port.

3.7 L2 Operation

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 upon the selected operation mode of the L2.

Writing to the L2MODE field of the cache configuration register (CCFG) sets the L2 operation mode. The L2 can function as a mapped SRAM and enabled cache. In cache mode, the L2 does not support freeze or bypass operation. Figure 3–28 shows the CCFG register for C621x/C671x and C64x. Table 3–9 and Table 3–10 describe the bit fields of this register for the C621x/C671x and C64x, respectively. Refer to section 3.3.3 for details of the L2 operation according to L2MODE for the TMS320C621x/C671x and to section 3.4.7 for details of the L2 operation according to L2MODE for the TMS320C64x.

Figure 3–28. Cache Configuration Register (CCFG)

31 29	28 10	9	8	7 3	2 0
P†	Reserved	IP	ID	Reserved	L2MODE
RW, +000	R,+x	W,+0	W,+0	R,+0 0000	RW,+000

† Applicable on C64x only. On C621x/C671x, bit field P is reserved, R, +000b.

Table 3–9. TMS320C621x/C671x Cache Configuration Register (CCFG) Field Description

Field	Description

L2MODE	L2 Operation Mode
	L2MODE = 000b: No L2 Cache / All SRAM mode
	L2MODE = 001b: 1-way cache / ¾ mapped SRAM
	L2MODE = 010b: 2-way cache / ½ mapped SRAM
	L2MODE = 011b: 3-way cache / ¼ mapped SRAM
	L2MODE = 111b: 4-way cache
	L2MODE = other: Reserved

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

ID	Invalidate L1D
	ID = 0: Normal L1D operation
	ID = 1: All L1D lines invalidatedInvalidate LIP

Table 3–10. TMS320C64x Cache Configuration Register (CCFG) Field Description

Field	Description

L2MODE	L2 Operation Mode
	L2MODE = 000b: No L2 Cache / All SRAM mode 1Mbyte
	L2MODE = 001b: 4-way Cache / 31/32 mapped SRAM
	L2MODE = 010b: 4-way Cache / 15/16 mapped SRAM
	L2MODE = 011b: 4-way Cache / 7/8 mapped SRAM
	L2MODE = 111b: 4-way Cache / ¾ mapped SRAM
	L2MODE = other: Reserved

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

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

P	L2 Requestor Priority P = 000b: L2 controller requests are placed on urgent priority level P = 001b: L2 controller requests are placed on high priority level P = 010b: L2 controller requests are placed on medium priority level P = 011b: L2 controller requests are placed on low priority level

The reset value of the L2MODE field is 000b, thus the L2 RAM is configured as mapped SRAM at reset to support data boot-loading. Any L2 RAM that is configured as cache is no longer in the memory map. For example, in L2 mode 010b the address range from 00010000h to 0001FFFFh is no longer available in the TMS320C64x memory map. The associativity of the L2 cache RAM is a function of the L2 Mode on the C671x and C621x but stays at four-way for the C64x architecture. On C621x/C671x each ¼ of SRAM added in the cache increases the associativity by one line per set. To ensure coherency and data integrity on an L2 mode switch, the user must perform a series of operations.

Table 3–11 specifies the required operations when either adding or removing L2 memory as mapped SRAM. Failure to follow these guidelines could result in data loss and undefined L2 operation.

Table 3–11. L2 Mode Switch Guidelines

To Switch From	To	Perform the following
Mode with L2 SRAM	Mode with less L2 mapped SRAM	1) Use EDMA to transfer any data needed out of the L2 SRAM space to be converted into cache. 2) Perform clean of L2 (L2CLEAN). 3) Wait until completion of EDMA or L2 clean, whichever is last. 4) Read priority queue status register (PQSR) of the EDMA to make sure they have completed 5) Switch L2 mode via write to CCFG. 6) Force CCFG modification via read from CCFG. 7) Execute 8 cycles of NOP.
Any L2 mode	Mode with more L2 mapped SRAM	1) Perform clean of L2 cache (L2CLEAN). 2) Wait until completion of L2 clean. 3) Switch L2 mode via write to CCFG. 4) Force CCFG modification via read from CCFG. 5) Wait for all EDMA priority queue status register (PQSR) to be come zero. 6) Execute 8 cycles of NOP.

3.7.1 L2 Interfaces

The L2 Controller services requests from three different requestors: L1P, L1D, and EDMA. Since the L1P only sends read requests, a single data bus transfers data from the L2 to the L1P. The L1D to L2 interface consists of a read bus from the L2 to the L1D and a write bus from the L1D to the L2. The L2 transfers data to and from the EDMA through separate read and write busses.

3.7.2 L2 Organization

The L2 SRAM is organized as multiple banks. Two accesses can be serviced at the same time if the two accesses do not use the same bank.

The L2 SRAM is made up of four 64-bit wide memory banks. 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 following priority is assigned to the events.

1) L1P or L1D (Hit)

2) EDMA read/writes

3) Read Fill (cache service in L2)

4) Victim Write (cache write-back from L2 to memory via EDMA)

5) Snoop (data merging with L1D)

When an L1P and L1D access to the L2 collide, the L1P request is always given priority. On a collision between a CPU and an EDMA request to the L2 the EDMA request is prioritized over the CPU request.

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.

3.7.3 L2 Read Requests

When an L2 location is enabled as a cache, the operation is similar to the L1D cache. On a read request to the L2 the data is sent to the requestor if a hit occurs. If the data is not in the L2 the requestor is stalled and the Least Recently Used(LRU) line is allocated for the new data. If the allocated line contains valid data the L1D is snooped. The L1D must be snooped even if an L1P miss supplied the L2 miss address because the evicted L2 line could be cached in the L1D. If the L1D returns data both the matching L1D line and evicted L2 line are invalidated, otherwise only the evicted L2 line is invalidated. Both the L2 and L1D caches must be invalidated on an L1D match to maintain coherency between the caches. If the L1D returns dirty data or if the evicted L2 line contains dirty data that data is evicted to the external memory and the required data is requested from the Enhanced DMA. The L2 is a load through cache, thus on an L1/L2 read miss the requested data is forwarded to the L1 by the L2 when it is available from the EDMA. This load through mechanism reduces the CPU stall time. Out-of-order fetching is used to minimize the CPU stall time. The necessary data is fetched first and the other side of the line is brought in afterwards. When an L1 cache requests data from the L2 that also misses the L2 the portion of fetched L2 line which contains the L1 line is requested from the EDMA first followed by the remaining L2 line portions.

3.7.4 L2 Write Requests

On a write request to the L2, operation varies according to the state of the address in the cache. On a write hit to the L2, the data in the L2 is modified with the new data from the CPU and the line is marked as dirty. The L2 is a write-allocate cache thus on a write miss the LRU line is allocated in the L2 for the new data. If that line contains valid data then the same snoop mechanism used on a L2 read miss is implemented to evict the data to external memory. The L2 line that contains the data that is to be modified is fetched from the external memory, the new data from the L1 is merged with the data from external memory and the line is marked as valid and dirty. The L2 is a writeback cache, therefore it may be required that the user force an L2 invalidation to maintain coherency with the external memory. Section 3.7.11 describes L2 invalidation methods.

The memory attribute registers (MARs) can be programmed to turn on or off 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 L2 line of data. Each of the CE spaces is divided into ranges that can be made cacheable by the least significant bit of a MAR register. If the CE bit of a MAR register is set, the L2 and L1s cache the corresponding address range. If the CE bit of a MAR register is not set, the L2, L1D and L1P will not cache the corresponding address range. When an EMIF address is not cacheable any access to that address by the CPU will bypass all caches and fetch the data directly from the external memory.

Table 3–12 describes the operation of the MAR field. At reset, the MAR registers are set to 0. To begin caching data in the L2 and L1s, you must set the appropriate MAR register to 1. The MARs are shown in Figure 3–10 and Figure 3–19. Refer to Table 3–2 and Table 3–3 for details of the MAR ranges.

Table 3–12. Memory Attribute Register(MAR) Bit Field Description

Field	Description
CE	Cacheability Enable
	CE = 0: Memory range is NOT cacheable
	CE = 1: Memory range is cacheable

The MAR registers define operation for the EMIF space. For C64x, both EMIFA and EMIFB spaces are included. Accesses by the EMIF to addresses which are not enabled as cacheable (CE = 0) are long distance accesses. On these accesses only the requested data size is fetched, not an entire cache line. Addresses in the L2 that are operating in mapped mode are always cacheable by the L1s.

CAUTION

The reset value of the memory attribute registers configures all external memory as not cacheable. In order to use the L1 and L2 caches to store any external data the appropriate CE field must first be set to 1.

3.7.5 L1D Cache in all L2 SRAM Mode

When the L2 SRAM is configured to be in all SRAM mode (L2MODE = 000b in the CCFG), accesses to addresses not in the L2 SRAM occur naturally but have two different modes of operation. The mode of operation depends on the cacheability of the region. If the region is not cacheable (CE = 0 in the MAR register), accesses to that region of memory will only return the exact piece of requested data. No other data will be cached in L1D or L2.

If the region is cacheable (CE = 1 in the MAR register), the data element is still loaded into the CPU register. In addition, the line that contains the data element will be allocated in L1D but not L2, since L2 is set in all SRAM mode. See Table 3–1 for the line size.

3.7.6 L1D and L2 Host-Processor Interface

Any access to the L2 SRAM from a remote CPU via the PCI or the Host Port Interface(HPI) will have to conform to strong memory ordering rules. Remote accesses of local memories that are reads cause the data in L1D to be copied back to L2. If the host access is a write, the line in L1D is snooped and invalidated and written back if needed to support the remote request, so the next time it is accessed by the CPU, the new line in L2 will be allocated again in L1D.

3.7.7 Host Access of L2 registers

The L2 registers are not available via the EDMA. This means an EDMA transfer cannot be configured to write data into the L2 control register address space. Thus the HPI and so a host cannot execute any operations on the DSP. This includes all types of L2 flushes and cleans. To perform this, the host would have to interrupt the CPU core to perform the flush or clean operation.

3.7.8 External Coherency

The C621x/C671x and C64x do not perform any external (off chip) coherency checking. The L2 does not perform any snoops in the external address space, covered by the MAR registers. If data from an external memory space is cached in L2 and the data in the external memory is modified by a external event, the data cache in L2 will not be aware of this.

This coherency maintenance would have to be performed by some form of interrupt service routine to cause the invalidation by cleaning of the cached image of external memory before it was modified. Once the data was modified this space would automatically be updated in L2 by the natural cache mechanism when the data was needed by the CPU.

3.7.9 EDMA Service

EDMA accesses are only allowed to L2 space that is configured as mapped SRAM. When the EDMA makes a read request to the L2, the L1D is snooped for the L2 address and the EDMA transfer is stalled until a response is returned. If the L1D returns data, that data is placed in the L2 and the EDMA request proceeds. The C621x/C671x and C64x behave differently in this case. For the C621x/C671x the L1D line is invalidated to maintain coherency. In the case of the C64x the L1D line is not invalidated. At a future time when the EDMA writes more data to this address, the appropriate line in L1D will be invalidated. This maintains coherency but in a different way than in C621x/C671x. The L1P is not snooped for data when an EDMA read request is received because the CPU cannot modify data in L1P so the L1P’s data will not be incoherent.

On the C621x/C671x when the EDMA makes a write request to the L2, both the L1P and the L1D are snooped for the L2 address. 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. This is true in the case of program writes to L2, L1P will be invalidated, but in the L1D on the C64x the affected lines are not invalidated.

Any access to L2 SRAM space by the EDMA to an address which is not present in the L2 SRAM, due to L2MODE settings, may corrupt the L2 cache state.

3.7.10 EDMA Coherency

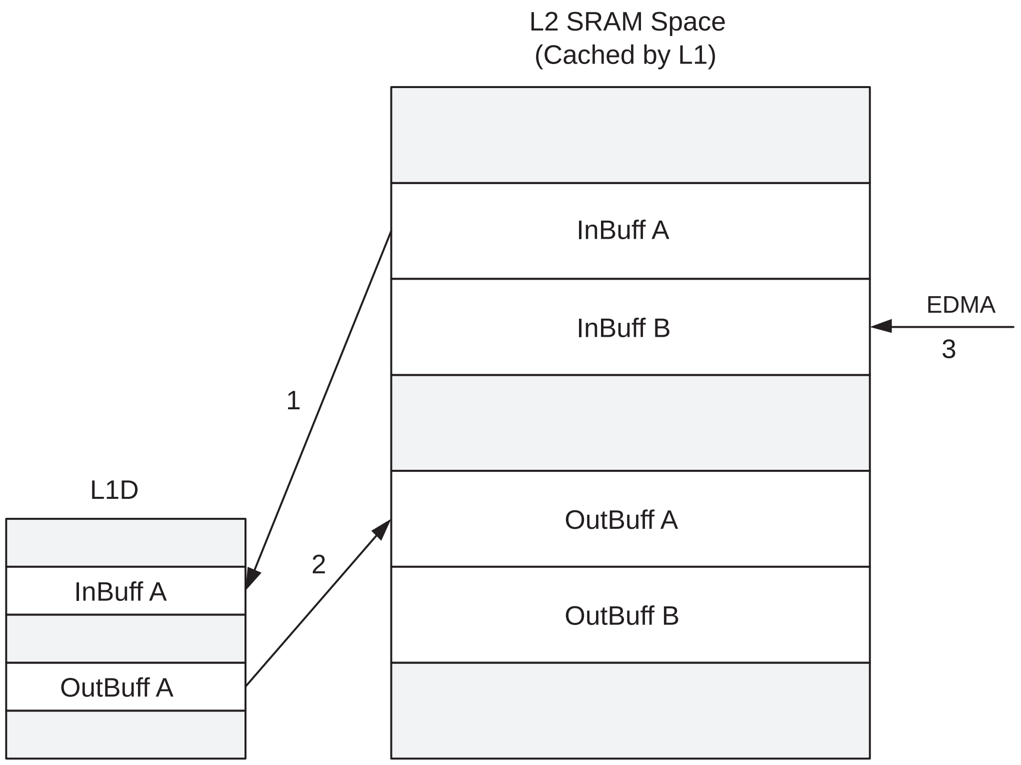
On the C621x/C671x, when data is written into L2 SRAM using the EDMA, the data is then cacheable using L1D. Accesses to the locations in L2 will cause allocation of the appropriate lines in L1D. This data can be read or written back to and from the memory space. After processing this section of data it may be desirable to transfer this data out again to external memory. When this occurs a snoop notices the EDMA accessing the addresses of the lines which are allocated in L1D, these L1D lines are cleaned out of L1D into L2 before they are transferred out by the EDMA. In other words these lines are invalidated in L1D and the data copied back to L2. The next time any data is transferred into the same place, reads to the same memory locations will cause new misses. This is an automatic process and maintains coherency between the data being transferred in and out using the EDMA. It forces the L1D cache to always read the newest data in the L2 and also forces the EDMA to always read the freshest data in L2.

This C64x device functions differently. The same coherency is maintained except at the point when the data from L2 is allocated in L1D and the L2 locations are read by the EDMA. In the C621x/C671x, the data is ‘cleaned’ out of L1D; the lines are invalidated and the data is copied back to L2. In the C64x, the data is still copied back, but the lines are not invalidated. Therefore any future access to those lines in L1D will result in a read hit, as opposed to a read miss in the C621x/C671x case. Though the data has been read out using the EDMA the data can still be accessed in L1D. On the C621x/C671x this would result in read misses.

The example in Figure 3–29 and Figure 3–30 illustrates how the L1–L2 caches work in conjunction with DMAs to L2 on the C64x. For this example, assume the L2 space is configured solely as SRAM (L2MODE = 000b).

Note: The parenthetical numbers in this section refer to these two figures.

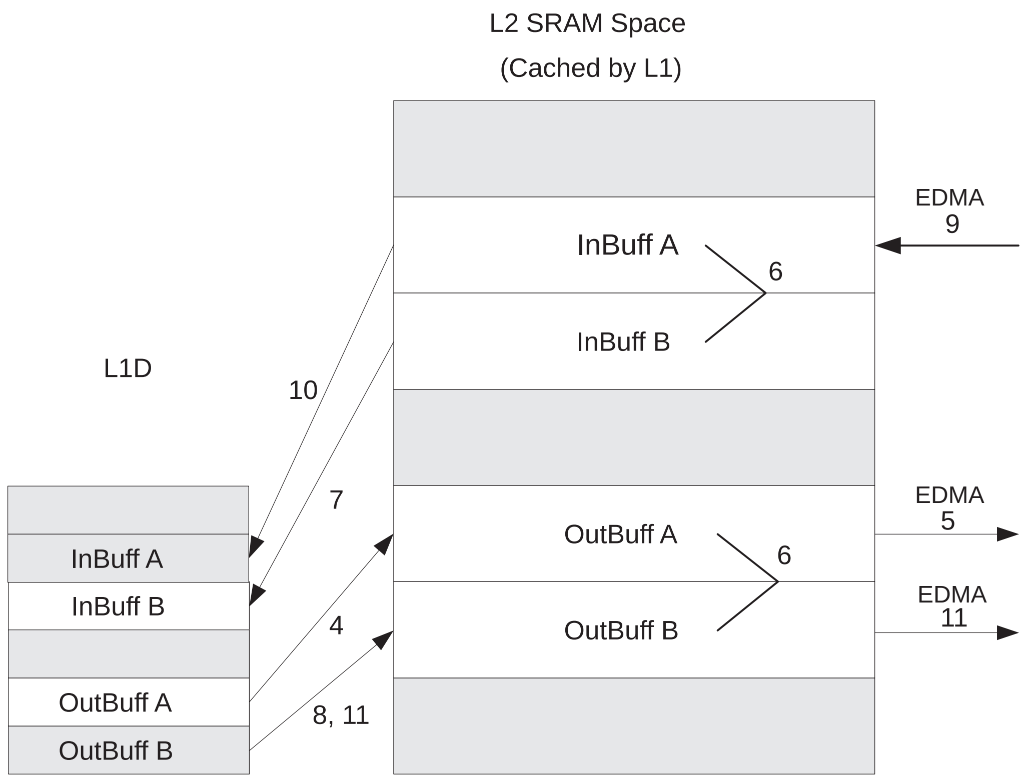
Figure 3–29. Coherency Diagram, Buffers A, B Allocated in L1D



Note: The numbers in the figure refer to the process described in this section.

The process begins with a buffer of data, InBuffA, which has been written to L2 via the EDMA controller. For this example, it is assumed that an interrupt from the EDMA controller was sent to the DSP to inform it that data was ready. As the L2 space is accessed **(1)**, it is cached into the L1D memory. As the DSP processes the data, it might write data to a separate output buffer, OutBuffA, in the L2 space. For this example also, assume that the DSP first read OutBuffA, thus caching it into L1D **(2)** . This process sets the L2 copy of the L1D tags to note that the lines containing OutBuffA data are in L1D. If this had not occurred, then OutBuffA would only be resident in L2, and writes to OutBuffA would simply fall through the aforementioned write buffer to L2. Write data to OutBuffA is captured in the L1D image of OutBuffA, without L2's knowledge. Note that in the background, DMA accesses may be continuing to a separate input buffer (InBuffB) **(3)** .

Figure 3–30. Coherency Example Part 2, Data Written Back and Transferred by EDMA



Note: The numbers in the figure refer to the process described in this section.

Once processing is complete, the DSP or possibly a peripheral that requires the data in OutBuffA, performs a DMA access to L2. The L2 queries its copy of the L1D tags, and, detecting that the data has been cached into L1D, performs a snoop cycle to L1D, and waits for a response. The L1D image of OutBuffA is returned to L2 **(4)**, but left valid in L1D. The data is written to the L2 SRAM, and then passed to the I/O subsystem to service the DMA **(5)**. If the DSP requires further use of OutBuffA, the access will hit L1D, and thus performance is preserved.

As typical with many DSP applications, double buffering is incorporated into this example to illustrate the coherency issues **(6**). As the process continues, the DSP is interrupted once InBuffB is ready to be processed. The DSP reads InBuffB, caching it into L1D **(7)**. Again, this process sets the L2 copy of the L1D tags to note that the lines containing OutBuffB are in L1D. As data is processed, the DSP writes it to OutBuffB **(8)**. As before, assume the DSP reads OutBuffB first, thus caching it into L1D. Writes to OutBuffB will be captured in L1D's image of OutBuffB.

If OutBuffB has not been read first, the writes to OutBuffB would simply pass through to L2 directly, and leave L1D unmodified. The reads of the output buffer are included to illustrate the significance of the snoop cycles the cache performs for applications which perform this type of access.

As before, in the background DMA accesses may continue to buffers that may be resident in L1D **(9)**, as shown with InBuffA above. As DMA accesses are performed to L2, the appropriate lines in L1D are snoop-invalidated out of L1D, **(10)** to prevent the DSP from using stale data. As DMA requests are made of OutBuffB, L1D’s copy of OutBuffB will be snooped **(11)** and the most up-to-date data is passed to the I/O subsystem. In this manner, the cache hardware maintains coherency with L2, such that the programmer’s view of the device is just L2 SRAM.

The EDMA mechanisms for C621x/C67x and C64x are shown in Table 3–13 and Table 3–14, respectively.

Table 3–13. TMS320C621x/C671x EDMA Mechanism

Action	Effect on L1D	Action Taken by L2
EDMA writes data to L2 SRAM, bufferA	Invalidation of lines in L1D	Snoops monitor writes to bufferA, L2 invalidates those lines in L1D
Process data in bufferA	Lines allocated in L1D from bufferA	Nothing
EDMA reads data from L2 SRAM from bufferA	Lines in L1D from bufferA are invalidated	Nothing

Table 3–14. TMS320C64x EDMA Mechanism

Action	Effect on L1D	Action Taken by L2
EDMA writes data to L2 SRAM, bufferA	Invalidation of lines in L1D	Snoops monitor writes to bufferA, L2 invalidates those lines in L1D
Process data in bufferA	Lines allocated in L1D from bufferA	Nothing
EDMA reads data from L2 SRAM from bufferA	Nothing	Nothing

3.7.11 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, any modified L2 locations are copied to external memory. 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 modified data from the L2 to the external memory space and snoops data from the L1D. In addition, the clean operation invalidates all lines in the L1P, L1D, and L2. Refer to Table 3–17 for a summary of the L2 flush and clean operations.

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 3–31 shows the L2FLUSH register. Table 3–15 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 3–32 shows the L2CLEAN register. Table 3–17 describes the operation of the L2CLEAN register.

Figure 3–31. L2 Flush Register (L2FLUSH)

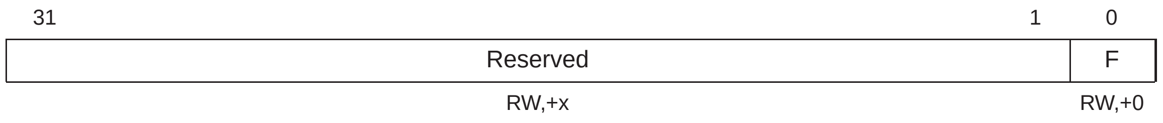


Table 3–15. L2 Flush Register Fields Description

Field	Description

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

Figure 3–32. L2 Clean Register (L2CLEAN)

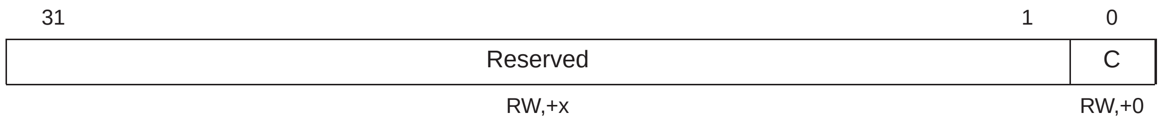


Table 3–16. L2 Clean Register Fields Description

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 the L2 for all lines whose external memory address falls within the range from L2FBAR to L2FBAR+L2FWC-1 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 the user should take care to ensure that the L2FBAR register is set up correctly prior to writing the L2FWC. When the flush is complete, the L2FWC register will contain the value 0. Figure 3–33 shows the L2FBAR register. Figure 3–34 shows the L2FWC register.

Figure 3–33. L2 Flush Base Address Register (L2FBAR)

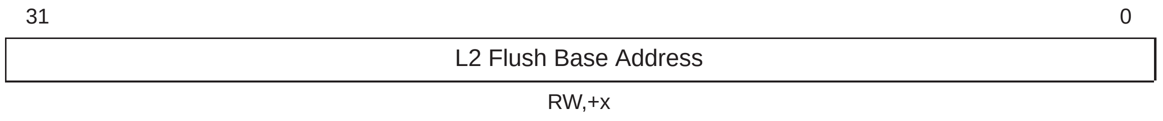
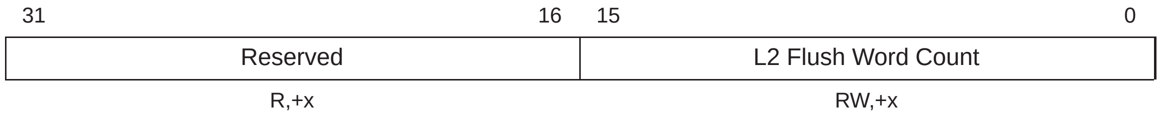


Figure 3–34. L2 Flush Word Count Register (L2FWC)



To clean an address range 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 the L2 for all lines whose external memory address falls within the range from L2CBAR to L2CBAR+L2CWC-1 and copies that data, through the EDMA, to external memory space (as well as invalidating those L2 lines). 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 the user should take care to ensure that the L2CBAR register is set up correctly prior to writing the L2CWC. When the invalidation is complete, the L2CWC register will contain the value 0.

The user can submit multiple CLEAN or FLUSH operations. The first one submitted will show the current flush or clean word count in the L2FWC register. When this counts down to zero, it is loaded with the next value in the list of flush or clean requests. A consistent zero count in this register will show that all operations have completed.

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-1 are saved to the external memory space. Figure 3–35 shows the L2CBAR register. Figure 3–36 shows the L2CWC register. Refer to Table 3–17 for a summary of the L2 range flush and clean operations.

Figure 3–35. L2 Clean Base Address Register (L2CBAR)

31   0
L2 Clean Base Address
RW,+x

Figure 3–36. L2 Clean Word Count Register (L2CWC)

31 16	15 0
Reserved	L2 Clean Word Count
R,+x	RW,+x

Table 3–17. Cache Flush/Clean Summary

Cache Operation	L2 Register Usage	L1P Effect	L1D Effect	L2 Effect
L2 Clean	L2CLEAN	Invalidated (completely)	Modified data sent to L2 and all data invalidated	Invalidated (completely); Modified data sent to external memory
L2 Flush	L2FLUSH	None	Modified data sent to L2 and invalidated	Modified data written back to external memory (but kept valid)
L2 Range Flush	L2FBAR/ L2FWC	None	Modified data in selected address range sent to L2 and invalidated in L1D.	Modified data written back to external memory (but kept valid)
L2 Range Clean	L2CBAR/ L2CWC	All addresses in selected range invali-dated	Modified data in selected address range sent to L2 and invalidated in L1D	Modified data written back to external memory and invalidated
L1P Flush	L1PFBAR/ L1PFWC	All addresses in selected range invali-dated	None	None
L1D Flush	L1DFBAR/ L1DFWC	None	Modified data in selected address range sent to L2 and invalidated in L1D.	Data updated in L2

Figure 3–37 through Figure 3–40 illustrate the operation of the C621x/C671x/C64x two-level memory system for CPU read and write requests, and EDMA read and write requests. These figures only demonstrate the operation and do not necessarily describe exact ordering of operation.

Figure 3–37. Two-Level Cache CPU Read Operation Flowchart

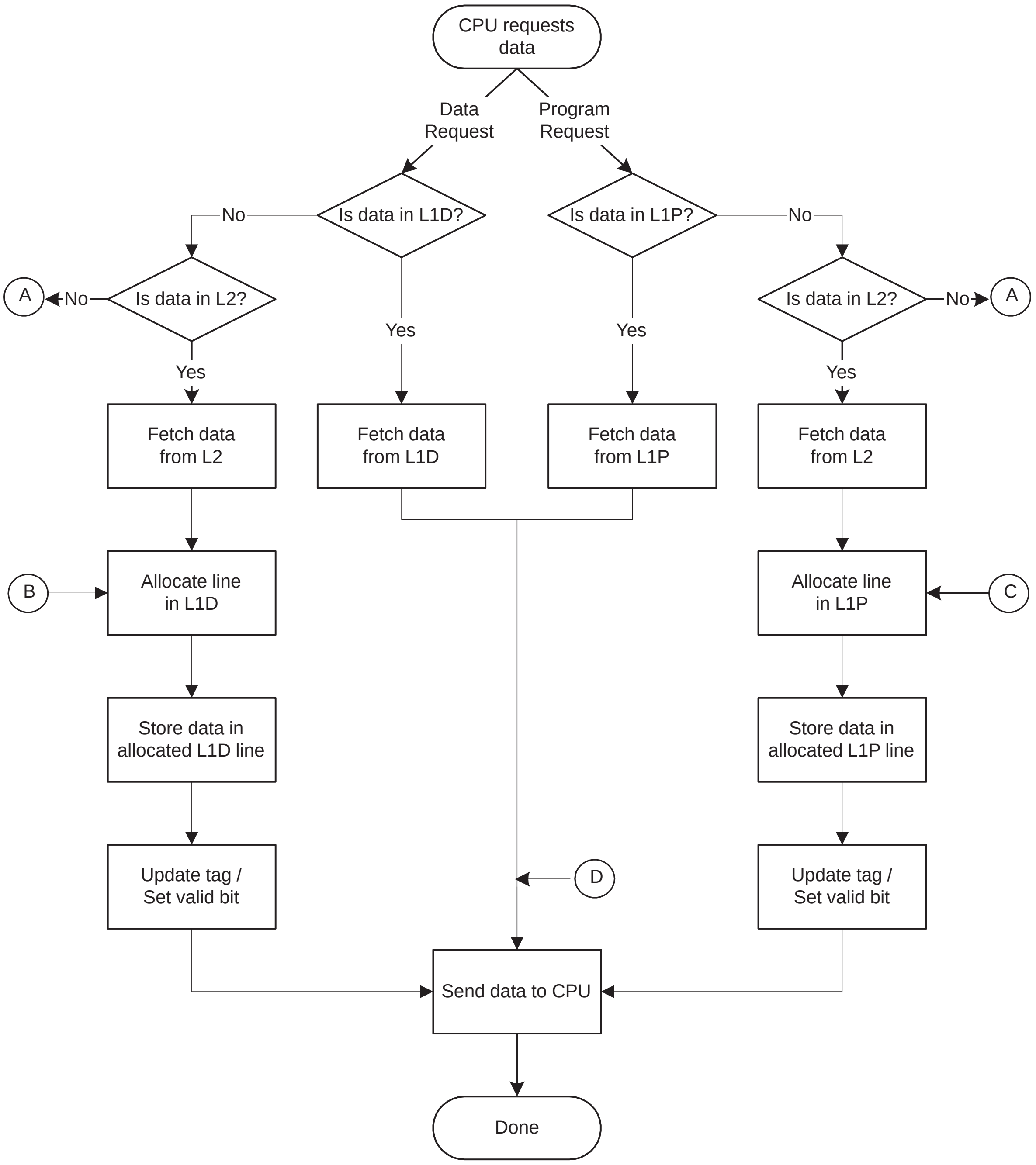


Figure 3–37. Two-Level Cache CPU Read Operation Flowchart (Continued)

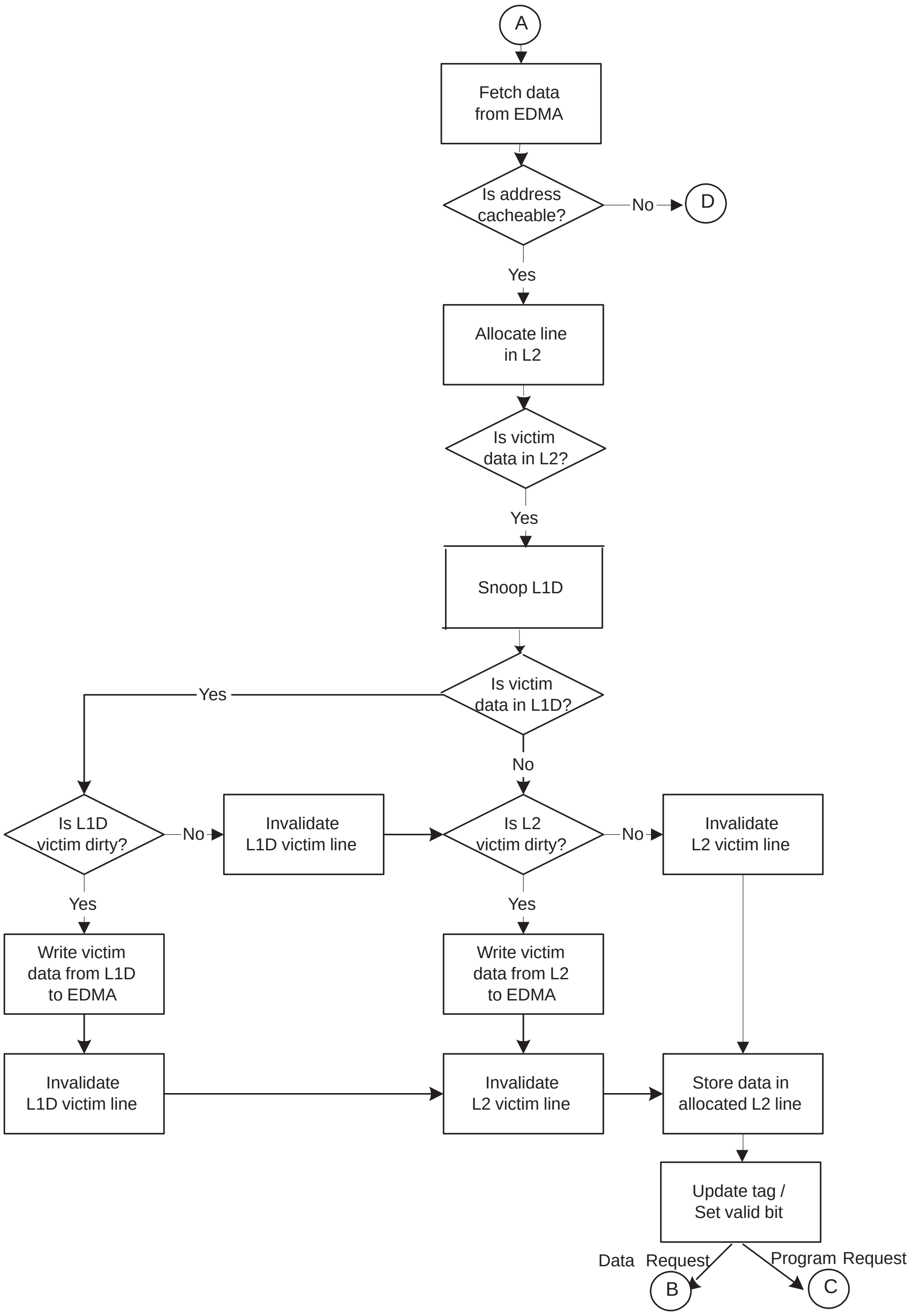

A
Fetch data from EDMA
Is address cacheable?
No
D
Yes
Allocate line in L2
Is victim data in L2?
Yes
Snoop L1D
Is victim data in L1D?
Yes
No
Is L1D victim dirty?
No
Invalidate L1D victim line
Is L2 victim dirty?
No
Invalidate L2 victim line
Yes
Write victim data from L1D to EDMA
Yes
Write victim data from L2 to EDMA
Invalidate L1D victim line
Invalidate L2 victim line
Store data in allocated L2 line
Update tag / Set valid bit
Data Request
B
Program Request
C

Figure 3–38. Two-Level Cache CPU Write Operation Flowchart

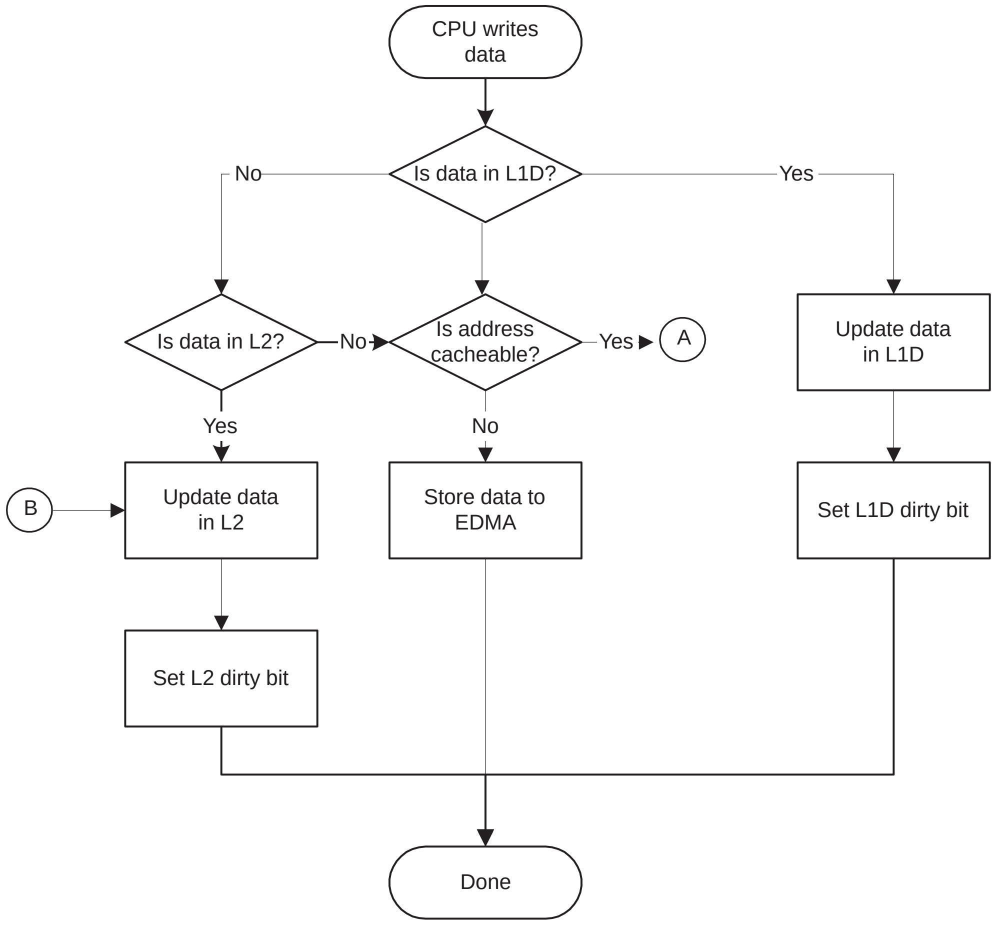


Figure 3–38. Two-Level Cache CPU Write Operation Flowchart (Continued)

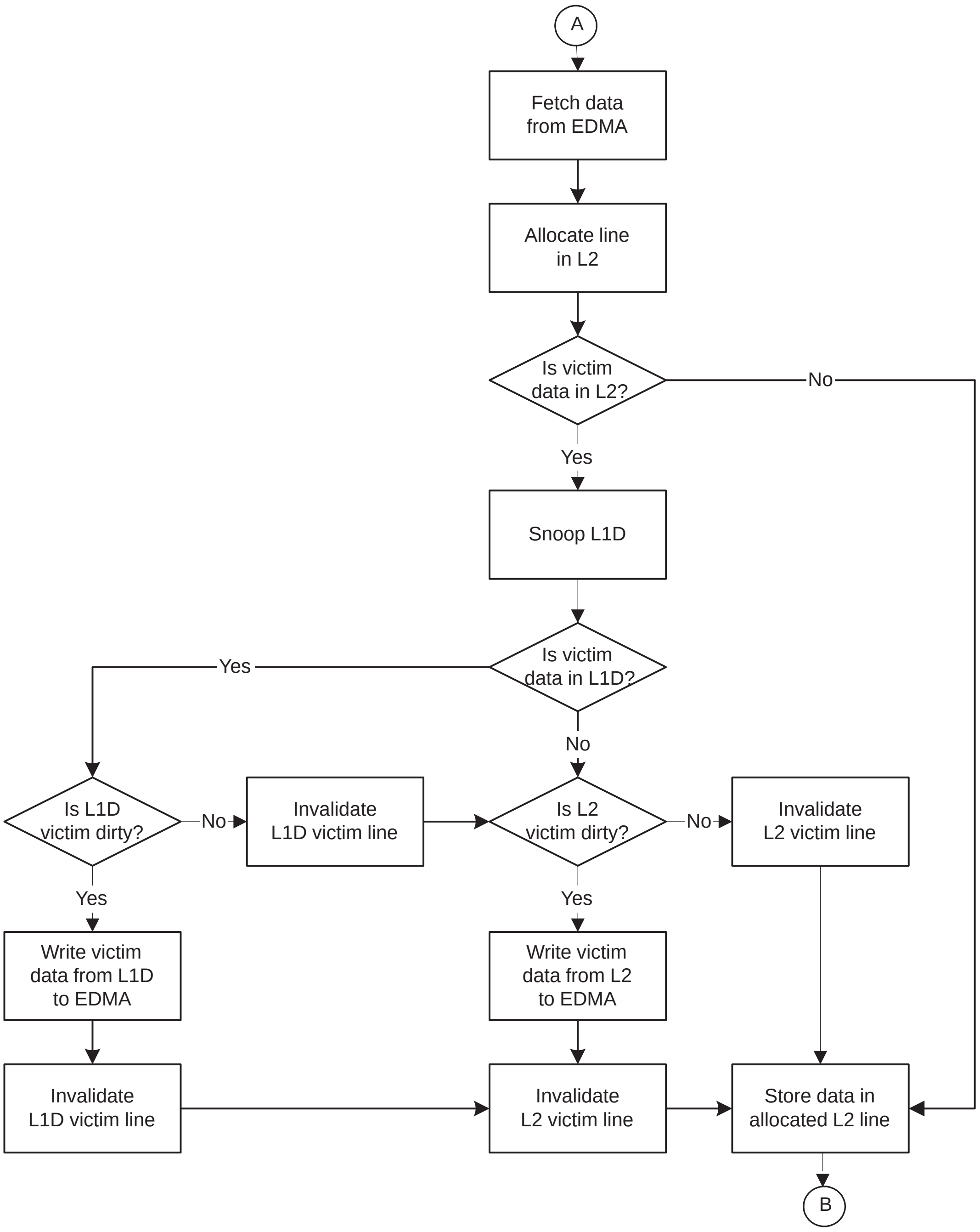

A
Fetch data from EDMA
Allocate line in L2
Is victim data in L2?
No
Yes
Snoop L1D
Is victim data in L1D?
Yes
No
Is L1D victim dirty?
No
Invalidate L1D victim line
Is L2 victim dirty?
No
Invalidate L2 victim line
Yes
Yes
Write victim data from L1D to EDMA
Write victim data from L2 to EDMA
Invalidate L1D victim line
Invalidate L2 victim line
Store data in allocated L2 line
B

Figure 3–39. Two-Level Memory EDMA Read Operation Flowchart

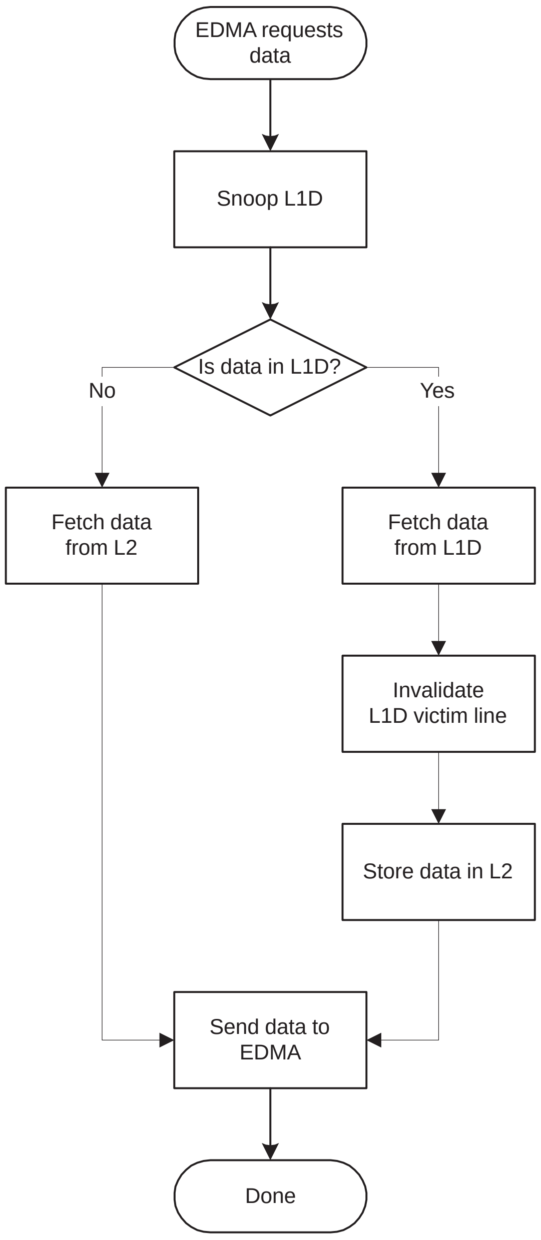
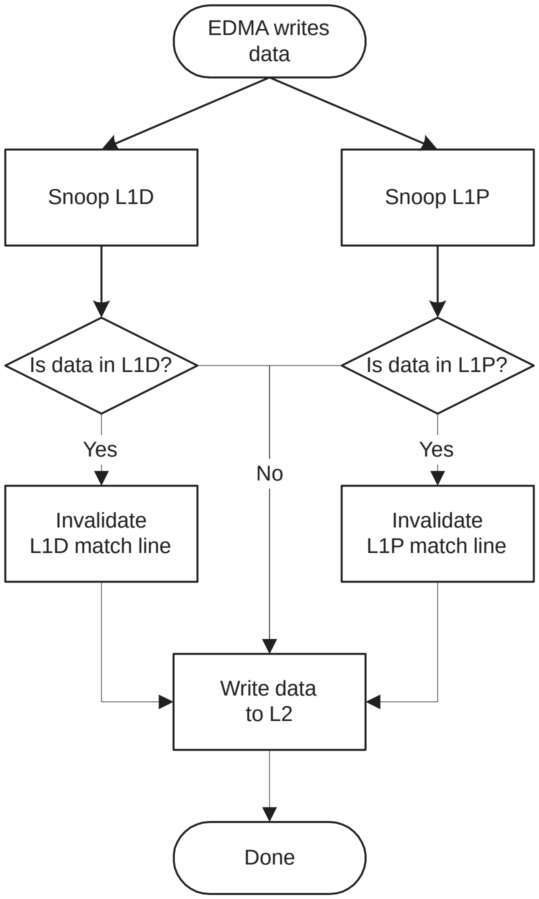


Figure 3–40. Two-Level Memory EDMA Write Operation Flowchart



Chapter 4

Direct Memory Access (DMA) Controller

This chapter describes the direct memory access channels and registers available for the TMS320C620x/C670x devices.

Topic **Page**

4.1 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 written to the split destination address. See section 4.8 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 4.8 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 4.11, *DMA Controller Structure*, on page 4-38 for more information.
- ❑ **Four channels:** The DMA controller can keep track of the contexts of four independent block transfers. See section 4.2, *DMA Registers*, on

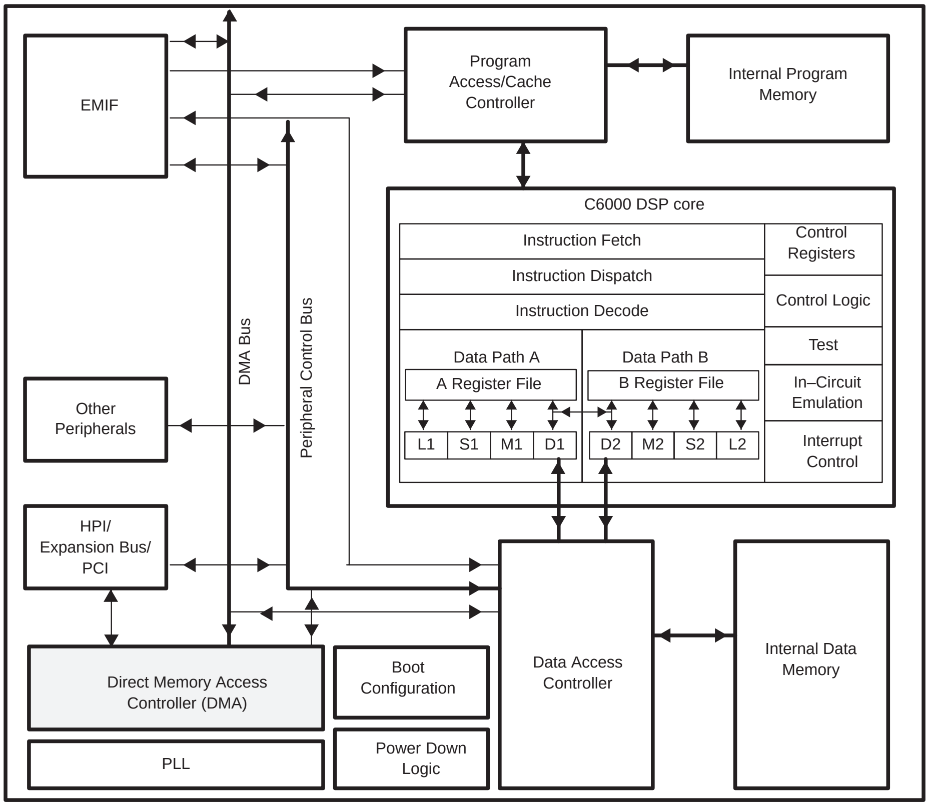
page 4-5 for more information about saving the contents of multiple block transfers.

- **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 4.8 on page 4-30 for more information.
- **Multiframe transfer**: Each block transfer can consist of multiple frames of a programmable size. See Section 4.5, *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 4.7.1 on page 4-24 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 4.7.3 on page 4-25 for more information.
- **Autoinitialization:** Once a block transfer is complete, a DMA channel can automatically reinitialize itself for the next block transfer. See section 4.4.1 on page 4-15 for more information.
- **Event synchronization:** Each read, write, or frame transfer may be initiated by selected events. See Section 4.6 on page 4-19 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 4.10 on page 4-35 for more information.

Figure 4–1 shows the TMS320C6000 block diagram with the DMA-related components shaded. Figure 4–1 summarizes the differences between the DMAs in different C6000 devices.

Figure 4–1. DMA Controller Interconnect to TMS320C6000 Memory-Mapped Modules



Note: Refer to the specific device datasheet for its peripheral set.

Table 4–1. Differences in TMS320C6000 DMAs

Features	Supported on Device	Described in Section
WSPOL, RSPOL, FSIG	ALL C620x/C670x, except C6201/C6701	4.2.1.2

4.2 DMA Registers

The DMA registers configure the operation of the DMA controller. How the DMA control registers are mapped in memory is shown in Table 4–2, which is in address order, and Table 4–3, which is in register name order.

These registers include the DMA global data, count reload, index, and address registers, as well as independent control registers for each channel.

Table 4–2. DMA Control Registers by Address

Hex Byte Address	Abbreviation	Name	Described in Section
0184 0000	PRICTL0	DMA channel 0 primary control	4.2.1
0184 0004	PRICTL2	DMA channel 2 primary control	4.2.1
0184 0008	SECCTL0	DMA channel 0 secondary control	4.10
0184 000C	SECCTL2	DMA channel 2 secondary control	4.10
0184 0010	SRC0	DMA channel 0 source address	4.7
0184 0014	SRC2	DMA channel 2 source address	4.7
0184 0018	DST0	DMA channel 0 destination address	4.7
0184 001C	DST2	DMA channel 2 destination address	4.7
0184 0020	XFRCNT0	DMA channel 0 transfer counter	4.5
0184 0024	XFRCNT2	DMA channel 2 transfer counter	4.5
0184 0028	GBLCNTA	DMA global count reload register A	4.5
0184 002C	GBLCNTB	DMA global count reload register B	4.5
0184 0030	GBLIDXA	DMA global index register A	4.7.2
0184 0034	GBLIDXB	DMA global index register B	4.7.2
0184 0038	GBLADDRA	DMA global address register A	4.8
0184 003C	GBLADDRB	DMA global address register B	4.8
0184 0040	PRICTL1	DMA channel 1 primary control	4.2.1
0184 0044	PRICTL3	DMA channel 3 primary control	4.2.1
0184 0048	SECCTL1	DMA channel 1 secondary control	4.10
0184 004C	SECCTL3	DMA channel 3 secondary control	4.10
0184 0050	SRC1	DMA channel 1 source address	4.7
0184 0054	SRC3	DMA channel 3 source address	4.7
0184 0058	DST1	DMA channel 1 destination address	4.7
0184 005C	DST3	DMA channel 3 destination address	4.7
0184 0060	XFRCNT1	DMA channel 1 transfer counter	4.5
0184 0064	XFRCNT3	DMA channel 3 transfer counter	4.5
0184 0068	GBLADDRC	DMA global address register C	4.8
0184 006C	GBLADDRD	DMA global address register D	4.8
0184 0070	AUXCTL	DMA auxiliary control register	4.9.1

Table 4–3. DMA Control Registers by Register Name

Name	Abbreviation	Hex Byte Address	Described in Section
DMA auxiliary control register	AUXCTL	0184 0070	4.9.1
DMA channel 0 destination address	DST0	0184 0018	4.7
DMA channel 0 primary control	PRICTL0	0184 0000	4.2.1
DMA channel 0 secondary control	SECCTL0	0184 0008	4.10
DMA channel 0 source address	SRC0	0184 0010	4.7
DMA channel 0 transfer counter	XFRCNT0	0184 0020	4.5
DMA channel 1 destination address	DST1	0184 0058	4.7
DMA channel 1 primary control	PRICTL1	0184 0040	4.2.1
DMA channel 1 secondary control	SECCTL1	0184 0048	4.10
DMA channel 1 source address	SRC1	0184 0050	4.7
DMA channel 1 transfer counter	XFRCNT1	0184 0060	4.5
DMA channel 2 destination address	DST2	0184 001C	4.7
DMA channel 2 primary control	PRICTL2	0184 0004	4.2.1
DMA channel 2 secondary control	SECCTL2	0184 000C	4.10
DMA channel 2 source address	SRC2	01840014	4.7
DMA channel 2 transfer counter	XFRCNT2	0184 0024	4.5
DMA channel 3 destination address	DST3	0184 005C	4.7
DMA channel 3 primary control	PRICTL3	0184 0044	4.2.1
DMA channel 3 secondary control	SECCTL3	0184 004C	4.10
DMA channel 3 source address	SRC3	0184 0054	4.7
DMA channel 3 transfer counter	XFRCNT3	0184 0064	4.5
DMA global address register A	GBLADDRA	0184 0038	4.8
DMA global address register B	GBLADDRB	0184 003C	4.8
DMA global address register C	GBLADDRC	0184 0068	4.8
DMA global address register D	GLLADDRD	0184 006C	4.8
DMA global count reload register A	GBLCNTA	0184 0028	4.5
DMA global count reload register B	GBLCNTB	0184 002C	4.5
DMA global index register A	GBLIDXA	0184 0030	4.7.2
DMA global index register B	GBLIDXB	0184 0034	4.7.2

4.2.1 DMA Channel Control Registers (PRICTL and SECCTL)

The DMA channel primary control register (PRICTL) and secondary control registers (SECCTL) contain fields that control each DMA channel independently. Several new fields have been added to SECCTL. The fields in these registers are described in this section.

4.2.1.1 DMA Channel Primary Control Register (PRICTL)

The DMA channel primary control register (PRICTL) fields are shown in Figure 4–2. Descriptions of these fields are summarized in Table 4–4.

Figure 4–2. DMA Channel Primary Control Register (PRICTL)

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 4–4. DMA Channel Primary Control Register (PRICTL) Field Descriptions

No.	Field	Description	Section
31 to 30	DST RELOAD	Destination address reload for autoinitialization DST RELOAD = 00b: do not reload during autoinitialization DST RELOAD = 01b: use DMA global address register B as reload DST RELOAD = 10b: use DMA global address register C as reload DST RELOAD = 11b: use DMA global address register D as reload	4.4.1.1
29 to 28	SRC RELOAD	Source address reload for autoinitialization SRC RELOAD = 00b: do not reload during autoinitialization SRC RELOAD = 01b: use DMA global address register B as reload SRC RELOAD = 10b: use DMA global address register C as reload SRC RELOAD = 11b: use DMA global address register D as reload	4.4.1.1
27	EMOD	Emulation mode EMOD = 0: DMA channel keeps running during an emulation halt EMOD = 1: DMA channel pauses during an emulation halt	4.13
26	FS	Frame synchronization FS = 0: disable FS = 1: RSYNC event used to synchronize entire frame	4.6

Table 4–4. DMA Channel Primary Control Register (PRICTL) Field Descriptions (Continued)

No.	Field	Description	Section

25	TCINT	Transfer controller interrupt
		TCINT = 0: interrupt disabled
		TCINT = 1: interrupt enabled
			4.10

24	PRI	Priority mode: DMA versus CPU
		PRI = 0: CPU priority
		PRI = 1: DMA priority
			4.9

23 to 19	WSYNC	Write transfer synchronization
		WSYNC = 00000b: no synchronization
		WSYNC = other: sets synchronization event
			4.6

18 to 14	RSYNC	Read synchronization
		RSYNC = 00000b: no synchronization
		RSYNC = other: sets synchronization event
			4.6

13	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
			4.7.2

12	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	4.4.1.1

11 to 10	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
			4.8

9 to 8	ESIZE	Element size
		ESIZE = 00b: 32-bit
		ESIZE = 01b: 16-bit
		ESIZE = 10b: 8-bit
		ESIZE = 11b: reserved
			4.7.3

Table 4–4. DMA Channel Primary Control Register (PRICTL) Field Descriptions (Continued)

No.	Field	Description	Section
7 to 6	DST DIR	Destination address modification after element transfers DST DIR = 00b: no modification DST DIR = 01b: increment by element size in bytes DST DIR = 10b: decrement by element size in bytes DST DIR = 11b: adjust using DMA global index register selected by INDEX	4.7.1
5 to 4	SRC DIR	Source address modification after element transfers SRC DIR = 00b: no modification SRC DIR = 01b: increment by element size in bytes SRC DIR = 10b: decrement by element size in bytes SRC DIR = 11b: adjust using DMA global index register selected by INDEX	4.7.2

3 to 2	STATUS	STATUS = 00b: stopped
		STATUS = 01b: running without autoinitialization
		STATUS = 10b: paused
		STATUS = 11b: running with autoinitialization
			4.4

1 to 0	START	START = 00b: stop
		START = 01b: start without autoinitialization
		START = 10b: pause
		START = 11b: start with autoinitialization
			4.4

4.2.1.2 DMA Channel Secondary Control Register (SECCTL)

The DMA Channel Secondary Control Register (SECCTL) fields are shown in Figure 4–3. These fields are summarized in Table 4–5.

The DMA channel secondary control register (SECCTL) of the C6202(B)/C6203(B)/C6204/C6205 has been expanded to include three new fields: WSPOL, RSPOL, and FSIG. These fields are used to add control to a frame-synchronized data transfer. The new fields are shown in gray in Figure 4–3.

Figure 4–3. DMA Channel Secondary Control Register (SECCTL)

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

Note: * WSPOL, RSPOL, and FSIG bit fields are not available to the C6201 and C6701 devices. These bitfields are R+0 on the C6201 and C6701 devices.

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

No.	Field	Description	Section
21	WSPOL	Write synchronization event polarity (not applicable for C6201 and C6701 devices). Selects the polarity of an external sync event: 1 = active low, 0 = active high This field is valid only if EXT_INTx is selected.	4.6.3
20	RSPOL	Read and frame synchronization event polarity (not applicable for C6201 and C6701 devices). Selects the polarity of an external sync event: 1 = active low, 0 = active high. This field is valid only if EXT_INTx is selected.	4.6.3

Table 4–5. DMA Channel Secondary Control Register (SECCTL) Field Descriptions (Continued)

No.	Field	Description	Section

19	FSIG	Level/edge detect mode selection. FSIG must be set to “0” for non-frame-synchronized transfers (not applicable for C6201 and C6701 devices).
		
		FSIG = 0: Edge detect mode (FS = 1 or FS = 0).
		FSIG = 1: Level detect mode (valid only when FS = 1).
		In level detect mode, synchronization inputs received during a frame transfer are ignored unless still set after the frame transfer completes.
			4.6.3

18 to 16	WSYNC CLR	Write synchronization status clear: Read as 0 write 1 to clear write synchronization status.	4.6.1
15	DMAC EN	DMA action complete pins reflect status and condition. 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	4.12
14	WSYNC STAT	Write synchronization status. WSYNC STAT = 0: Synchronization is not received. WSYNC STAT = 1: Synchronization is received.	4.6.1
13	RSYNC CLR	Read synchronization status clear: Read as 0 write 1 to clear read synchronization status.	
12	RSYNC STAT	Read synchronization status. RSYNC STAT = 0: Synchronization is not received. RSYNC STAT = 1: Synchronization is received.	4.6.1
11	WDROP IE	Write synchronization dropped interrupt enable. WDROP IE = 0: WDROP condition does not enable DMA channel interrupt. WDROP IE = 1: WDROP condition enables DMA channel interrupt.	4.10.1
10	WDROP COND	Write drop condition. WDROP COND = 0 WDROP condition is not detected. WDROP COND = 1 WDROP condition is detected.	4.10

Table 4–5. DMA Channel Secondary Control Register (SECCTL) Field Descriptions (Continued)

No.	Field	Description	Section
9	RDROP IE	Read synchronization dropped interrupt enable. RDROP IE = 0: RDROP condition does not enable DMA channel interrupt. RDROP IE = 1: RDROP condition enables DMA channel interrupt.	4.10.1
8	RDROP COND	Read drop condition. RDROP COND = 0: RDROP condition is not detected. RDROP COND = 1: RDROP condition is detected.	4.10
7	BLOCK IE	Block transfer finished interrupt enable. BLOCK IE = 0: BLOCK condition does not enable DMA channel interrupt BLOCK IE = 1: BLOCK condition enables DMA channel interrupt	4.10.1
6	BLOCK COND	Block transfer finished condition. BLOCK COND = 0: BLOCK condition is not detected. BLOCK COND = 1: BLOCK condition is detected.	4.10
5	LAST IE	Last frame finished interrupt enable. LAST IE = 0: LAST condition does not enable DMA channel interrupt. LAST IE = 1: LAST condition enables DMA channel interrupt.	4.10.1
4	LAST COND	Last frame finished condition. LAST COND = 0: LAST condition is not detected. LAST COND = 1: LAST condition is detected.	4.10
3	FRAME IE	Frame complete interrupt enable. FRAME IE = 0: FRAME condition does not enable DMA channel interrupt FRAME IE = 1: FRAME condition enables DMA channel interrupt	4.10.1
2	FRAME COND	Frame complete condition. FRAME COND = 0: FRAME condition is not detected. FRAME COND = 1: FRAME condition is detected.	4.10
1	SX IE	Split transmit overrun receive interrupt enable. SX IE = 0: SX condition does not enable DMA channel interrupt. SX IE = 1: SX condition enables DMA channel interrupt.	4.10.1
0	SX COND	Split transmit condition. SX COND = 0: SX condition is not detected. SX COND = 1: SX condition is detected.	4.10

4.2.2 Register Access Protocol

The following steps should be followed when setting up a DMA transfer:*

1) Set up primary control register with START = 00b.

2) Set up secondary control register with:
   WSYNC_CLR = 1 and
   RSYNC_CLR = 1.

3) Set up source/destination address registers and count register.

4) Start the DMA channel by writing a 01b or 11b to the START field of the primary control register.

   Note: The STATUS field in the PRICTL should equal 00b before configuring the DMA channel.

4.3 Memory Map

The DMA controller assumes the device memory map shown in *Chapter 11, Boot Modes and Configuration*. Requests are sent to one of these possible resources:

- ❑ Expansion bus (C6202(B)/C6203(B)/C6204 only)
- ❑ Host port interface (C6201/C6701 only)
- ❑ PCI (C6205 only)
- ❑ External memory interface
- ❑ Internal program memory, block 0
- ❑ Internal program memory, block 1 (C6202(B)/C6203(B) only)
- ❑ Internal peripheral bus
- ❑ Internal data memory.

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

4.4 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 (PRICTL).

Once the value of START has been modified, the primary control register should not be modified again until STATUS is equal to START.

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 PRICTL 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.

4.4.1 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, autoinitialization is enabled. In this case, after completion of a block transfer, the selected DMA channel registers are reloaded and the DMA channel is restarted . If restarting after a pause, START must be rewritten as 11b for autoinitialization to be enabled.

Switching from autoinitialization to non-autoinitialization: It is possible to switch from an autoinitialized transfer to a non-autoinitialized transfer to complete DMA activity on a particular channel. To switch modes, the active channel should be paused by restoring the primary control register with START = 10b, then restarted in the new mode by restoring the primary control register with START = 01b.

If the active channel is operating in split-mode, then it is necessary to ensure that the switch from autoinitialization to non-autoinitialization does not occur at a frame boundary. If the channel is paused with the transmit source in frame n and the receive destination in frame n – 1; then the channel must be restarted with autoinitialization (START = 11b), then repaused before the switching of modes occurs. This is to ensure that both transmit and receive data streams both complete the same number of frames.

4.4.1.1 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. 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 DST RELOAD and SRC RELOAD in the DMA channel primary control register (see Figure 4–2).

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 4.11.1.2, 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 4.8, the source and destination addresses are independently reloaded when the associated transmit or receive element transfers are completed.

The DMA channel transfer counter reload can be rewritten only after the next-to-last frame in the current block transfer is 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 4.5 for more explanation of the DMA channel transfer counter.

Note: You cannot switch from a non-XBUS or non-PCI *src/dst* address to an XBUS or PCI *src/dst* address during autoinitialization. Similarly, you cannot switch from an XBUS or PCI to a non-XBUS or non-PCI src/dst address.

4.5 Transfer Counting

The DMA channel transfer counter register (XFRCNT), shown in Figure 4–4 contains fields that represent the number of frames and the number of elements per frame to be transferred. Figure 4–5 shows the DMA global count reload register (GLBLCNT).

FRAME COUNT field: 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 65535. 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 4.4.1.1). Initial values of 0 and 1 in FRAME COUNT have the same effect of transferring a single frame.

ELEMENT COUNT field: 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 65535. 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 4–4. DMA Channel Transfer Counter Register (XFRCNT)

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

Figure 4–5. DMA Global Count Reload Register (GBLCNT) Used As Transfer Counter Reload)

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

4.6 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 4–6. This is similar to the fields in the interrupt selector (see section 14.5, *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 4–6. 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 4–6. 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	

4.6.1 Latching of DMA Channel Event Flags

The DMA channel secondary control register (described in Table 4–5) 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. User manipulation of events has priority over any simultaneous automated setting or clearing of events.

4.6.2 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.

4.6.3 Synchronization Control

The DMA of the C6202(B)/C6203(B)/C6204/C6205 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 4–6 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 synchronization event. During the synchronization event, transitions on EXT_INTx are ignored.

3) Same as 1

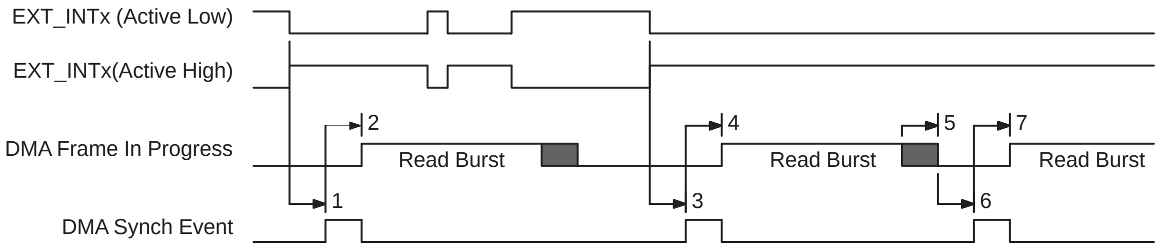
4) Same as 2

5) After a read burst completes internally, a delay of 32 CPU clock cycles are inserted before checking whether EXT_INTx is still active.

6) Because EXT_INTx is still active after the burst and delay, a new synchronization event is registered inside the DMA.

7) The new DMA synchronization event triggers another burst.

Figure 4–6. 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, when the ‘C6202 is reading from 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.

After a frame is completed, the DMA waits 32 CPU clock cycles before checking to determine if the flag is still active. If it is still active, the next frame will be synchronized based on the active flag. This delay is needed to give the external FIFO time to update its flags and give the flag time to propagate through the internal registers before being registered inside the DMA. For example, a FIFO typically takes approximately 1 to 3 FIFO clock cycles to update its flag externally. Depending on the divide ratio of the output XFCLK, this can translate to as long as 24 CPU cycles (for x8 mode).

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(B)/C6203(B)/C6204/C6205 DMA functions identically to the C6201 DMA.

4.7 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, it can be programmed to effectively treat the various elements in a frame as though they were coming from separate sources and group each source's data together.

The DMA channel source address (SRC) register, (shown in Figure 4–7) holds the address for the next read transfer. The DMA Channel destination address (DST) register, shown in Figure 4–8, holds the address for the next write transfer.

Figure 4–7. DMA Channel Source Address Register (SRC)

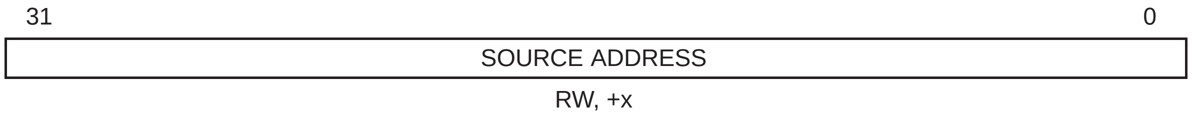
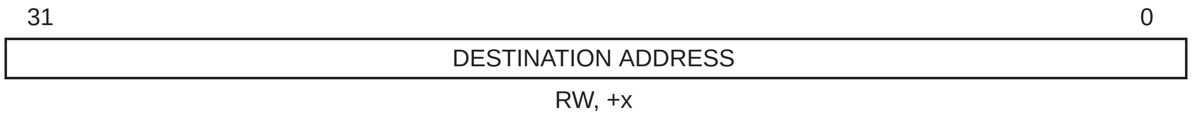


Figure 4–8. DMA Channel Destination Address Register (DST)



4.7.1 Basic Address Adjustment

As indicated in Table 4–4, the SRC DIR and DST DIR fields can do the following: set the index to increment by element size, decrement by element size, use a global index value, or not affect either the DMA channel source or destination address registers. 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.

4.7.2 Address Adjustment With the Global Index Registers

The particular DMA global index register (GBLIDX) shown in Figure 4–9 is selected via the INDEX field in the DMA channel primary control register. Unlike basic address adjustment, this mode allows different adjustment amounts depending upon 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 4–9. DMA Global Index Register (GBLIDX)

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

These fields affect address adjustment as follows.

- ❑ **ELEMENT INDEX field:** 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 field:** 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.

4.7.3 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 are 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. It is also necessary to 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 in big-endian mode an address ending in 11b selects the least significant byte .

4.7.4 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.

4.7.5 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) $\times$ Er). The effective element count is ((F – 1) $\times$ 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.

4.7.6 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 4–7 shows the data in the order in which it is transferred, and Table 4–8 shows how the data appears in memory after transfers are finished.

Table 4–7. 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 4–8. 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

4.8 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.

4.8.1 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, reinitialization 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.

4.8.2 Split Address Generation

The DMA global address register (GBLADDR), shown in Figure 4–10, 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 4–10. DMA Global Address Register (GBLADDR) 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.

Note: Split-mode cannot be used with the expansion bus. Neither the source address, destination address, nor the split address can be within the expansion bus I/O memory range.

4.9 Resource Arbitration and Priority Configuration

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

- DMA channels
- CPU program and data accesses.

The resources include:

- Internal data memory
- Internal program memory
- 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.
- P**riority 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.

4.9.1 DMA Auxiliary Control Register (AUXCTL) 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 4–11 and are summarized Table 4–9.

Figure 4–11. DMA Auxiliary Control Register (AUXCTL)

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

Table 4–9. DMA Auxiliary Control Register (AUXCTL)Field Descriptions

No.	Field	Description

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

3 to 0	CH PRI	DMA channel priority. In the case when the auxiliary channel is used to service the expansion bus host port operation, CH PRI must be 0000b (auxiliary channel highest priority) as shown below. 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

The priority assigned each DMA channel determines which DMA channel performs a read or write transfer first, when 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.

For read and write transfers, arbitration between channels occurs independently every CPU clock cycle. 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: A channel's PRI field should be modified only when that channel is paused or stopped.

4.9.2 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. See *Chapter 5, DMA and CPU Data Access Performance,* for more detail.

4.10 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.

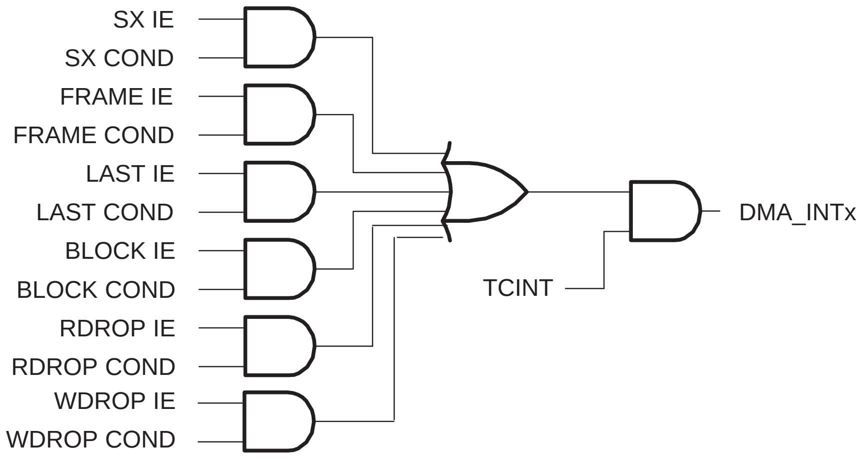
These registers also provide the means to enable the DMA channels to interrupt the CPU through their corresponding interrupt enable (IE) fields. If a condition flag and its corresponding IE bit are set, that condition is enabled to contribute to the status of the interrupt signal from the associated 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 4–12. 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 cannot be cleared automatically. A zero must be written to the COND bit to clear it. If the associated IE bit is not set, 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 4–12. Generation of DMA Interrupt for Channel x From Conditions



4.10.1 Definition of DMA Channel Secondary Control Register (SECCTL) Conditions

Table 4–10 describes each of the condition flags in the DMA channel secondary control register (SECCTL).

Depending upon 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, the user 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 4–10. DMA Channel Secondary Control Register (SECCTL) 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	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	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

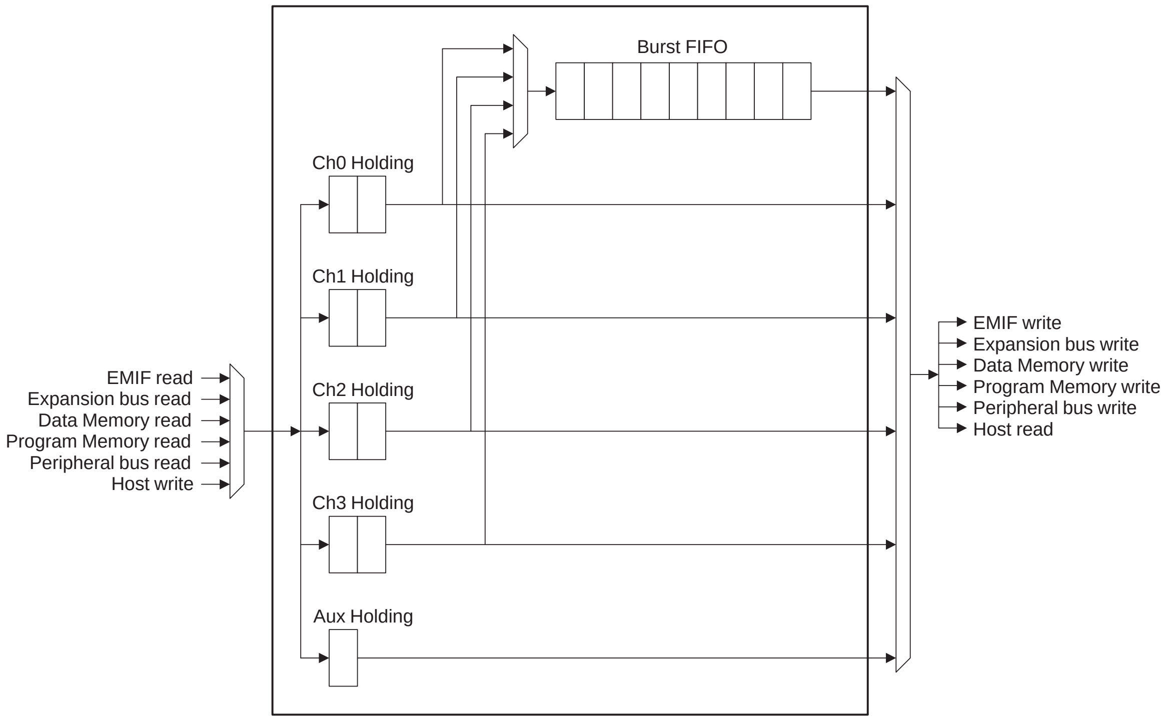
4.11 DMA Controller Structure

The TMS320C6000 generation DMA consists of four user-programmable channels and an auxiliary channel. Each channel is capable of bursting to high-speed memories and can be used in split-mode for peripheral support. The DMA of the C6201/C6701/C6202 (1.8V core supply devices) accomplish this by providing a pair of holding registers to each channel and a 9-deep shared FIFO. The DMA was modified slightly for the C6202B/C6203(B)/C6204/C6205 (1.5V core devices) to improve performance in certain instances; and rather than a shared FIFO, each DMA channel has a dedicated 9-deep FIFO available to use.

4.11.1 TMS320C6201/C6701/C6202 (1.8V devices) DMA Structure

Figure 4–13 shows the internal data movement paths of the 1.8V device DMA controller, including data buses and internal holding registers.

Figure 4–13. DMA Controller Data Bus Block Diagram for 1.8V Device



4.11.1.1 Read and Write Buses

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

- ❑ EMIF
- ❑ Expansion bus (C6202)/host port interface (C6201/C6701)
- ❑ Internal data memory
- ❑ Internal program memory, block 0
- ❑ Internal program memory, block 1 (C6202 only).
- ❑ 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, its buses have a different naming convention. 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.

4.11.1.2 Shared 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, a channel in that mode cannot use the FIFO. If only frame synchronization is enabled, that channel can still use the FIFO.
- ❑ 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 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.

4.11.1.3 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 upon whether the DMA controller is in split mode or not, additional restrictions can apply:

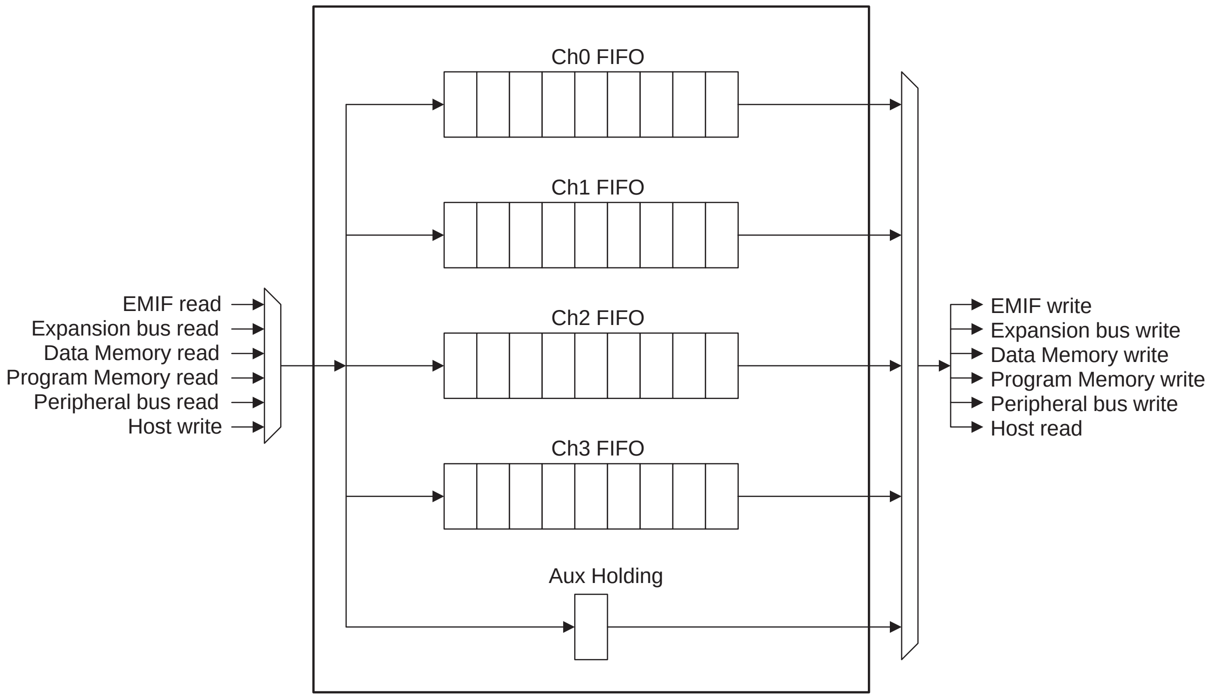
In split mode, the two registers serve as separate transmit and receive data stream holding registers for split mode. For both the transmit and receive-read transfer, no subsequent read transfer request is issued until the associated write transfer request completes.

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.

4.11.2 TMS320C6202B/C6203(B)/C6204/C6205 (1.5V Devices) DMA Structure

The structure of the 1.8V device DMA was redesigned for the 1.5V devices to obtain performance improvements. By removing the arbitration for a single burst FIFO, multiple bursting channels are more able to co-exist without loss of throughput. Figure 4–14 shows the internal data movement paths of the 1.5V device DMA controller, including data buses and internal FIFOs.

Figure 4–14. DMA Controller Data Bus Block Diagram for 1.5V Device



4.11.2.1 Read and Write Buses

As with the 1.8V device DMA, each DMA channel can independently select one of these sources and destinations:

- EMIF
- Expansion bus (C6202B/C6203(B)/C6204) / PCI (C6205)
- Internal data memory
- Internal program memory, block 0
- Internal program memory, block 1 (C6202B/C6203(B) only)
- Internal peripheral bus.

The auxiliary channel read and write buses are identical to those described in Figure 4–13.

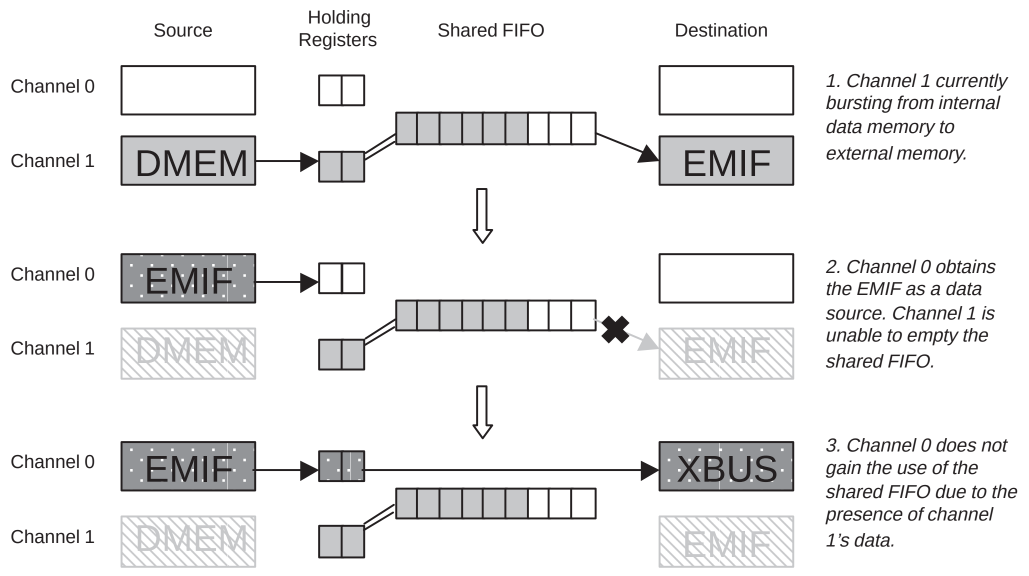
4.11.2.2 Channel FIFOs

Each 1.5V device DMA channel has a dedicated 9-deep FIFO to facilitate bursting to high-speed memories. Each channel owns its own FIFO, which reduces the arbitration required for switching between high-speed bursting channels. The individual operation by any channel is unchanged from that of the 1.8V device DMA. The benefit of multiple FIFOs comes into play only when switching between channels.

Dedicated FIFOs allow for a seamless transition from one bursting DMA channel to another. Since the 1.8V device DMA allows a higher-priority channel to begin prior to the lower priority channel completing all pending writes, potentially the higher-priority channel will not gain access to the FIFO. When the source of the higher-priority transfer is the same as the destination of the lower-priority channel, the FIFO will be unable to flush its data. The priority scheme of the DMA considers the DMA channel number to determine which channel gets access to a resource. Since the higher-priority channel will own the common resource, the lower-priority channel will be unable to access it for its pending writes. The data will remain in the shared FIFO, which prevents the higher-priority channel from obtaining its use.

The effect of this is that the interrupting high-priority channel is not able to burst properly, as shown in Figure 4–15, since it does not have possession of the FIFO. Instead it will use its holding registers as a 2-deep FIFO, which is not deep enough to facilitate bursting. Instead of a continuous burst of data, only two elements will be transmitted at a time.

Figure 4–15. Shared FIFO Resource Problem


Source
Holding Registers
Shared FIFO
Destination
Channel 0
Channel 1
DMEM
EMIF
1. Channel 1 currently bursting from internal data memory to external memory.
Channel 0
Channel 1
EMIF
DMEM
EMIF
2. Channel 0 obtains the EMIF as a data source. Channel 1 is unable to empty the shared FIFO.
Channel 0
Channel 1
EMIF
XBUS
DMEM
EMIF
3. Channel 0 does not gain the use of the shared FIFO due to the presence of channel 1's data.

The new structure of the 1.5V device DMA removes this bandwidth-limiting aspect of the 1.8V device DMA. By providing each channel with its own FIFO, it is not necessary for the lower priority channel to flush all of its pending data for the higher priority channel to be capable of high-speed bursts. The lower priority channel will maintain its data within its FIFO until the higher priority transfer completes. When it once again gains access to its destination resource, the transfer will resume.

In all other situations, the behavior of the 1.5V device DMA channels is identical to those of the 1.8V device.

4.11.2.3 Split Mode

When operating in split-mode, a 1.5V device DMA channel behaves identically to a 1.8V device DMA channel. Only the first two cells of the channel's FIFO are used, effectively becoming the two holding registers. Each cell serves as a holding register for the separate transmit data stream, and receive data stream. For both the transmit-read and receive-read transfer, no subsequent read transfer request is issued until the associated write transfer request completes.

4.11.3 Operation

Reads and writes by the DMA are independent from one another. This allows for a source to be accessed even if the destination is not ready, and vice versa. A DMA channel continues to issue read requests until its holding registers are full, or in the case of a burst transfer until the FIFO is full. Likewise the channel issues write requests until its holding registers are empty. In the situation where the DMA is both reading from and writing to the same resource, the write will have priority.

4.11.4 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 of 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.

For a detailed description, see *Chapter 5*, *DMA and CPU Data Access Performance.*

4.12 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.

4.13 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 5

DMA and CPU Data Access Performance

This chapter describes the DMA and CPU data access performance to the internal memory, the peripherals, and the external memory.

Topic **Page**

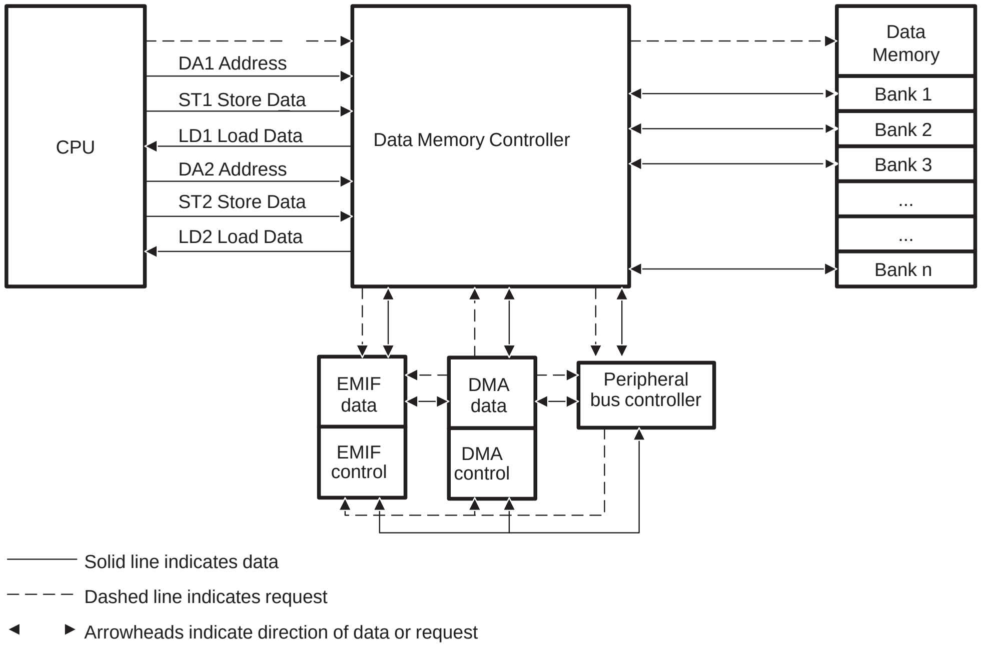
5.1 Overview

In a real-time system, it is important to understand data flow and control it to achieve high performance. By analyzing the timing characteristics for accessing data and switching between data requestors, it is possible to maximize the achievable bandwidth in any system. In general, one important guideline is to use the DMA controller for background off-chip data accesses. The CPU should only be used for sporadic (non-periodic) accesses to individual locations, otherwise the system can incur performance degradation. Although the CPU and the DMA controller function independently of one another, when both are performing simultaneous data accesses it is necessary to properly schedule and configure them in order to minimize conflict and waiting while meeting real-time requirements. This section provides the necessary information to understand how the different data requestors affect one another, as well as the amount of time required to perform data accesses. Due to the flexibility of the CPU and the DMA, code structure and DMA activity can be tailored to maximize data I/O bandwidth for particular situations. This section also provides guidelines on how to maximize the available bandwidth.

5.2 Accessing Data

The data to be accessed by the CPU and the DMA can be located in internal data memory, on-chip peripherals, or in external memory. Whenever the CPU data paths and the DMA contend for any of these resources, arbitration determines the order in which the requests are serviced. Data path A always has priority over data path B, and the priority level of the DMA with respect to the CPU is based on the priority (PRI) bit of the DMA channel's primary control register. This arbitration is valid for all resources. Figure 5–1 shows the CPU, data memory controller, and peripheral bus connections.

Figure 5–1. TMS320C620x/C670x Data Paths



5.2.1 Internal Data Memory

Internal data memory consists of high-speed SRAMs, which are divided into several 16-bit wide banks. Each bank can be accessed once per cycle, with distinct banks accessible in parallel. An access takes more than one cycle only if multiple requestors contend for the same memory bank. In this case a lower-priority requestor will be stalled until all of the banks it requests are free. For example during a CPU access to data memory bank1 and bank 2, a DMA access (configured to be lower priority) to bank 2 will be stalled until the CPU access to bank 2 is completed. Arbitration for each bank occurs every cycle. The physical arrangement, as well as the number, of data memory banks varies slightly between the DSPs. For more details, see Chapter 2, *TMS320C620x/C670x Internal Program and Data Memory.* The maximum data access rate to the internal data memory is also described in section 2.4.

5.2.2 Peripheral Bus

The on-chip peripherals are configured via memory-mapped control registers accessible through the peripheral bus controller. The peripheral bus controller performs word accesses only, which affects writes to a peripheral register. A write of a byte or halfword is treated as a 32-bit word. The values written to the non-selected bytes are undefined. On reads, individual bytes can be accessed, as the CPU or DMA extracts the appropriate bytes.

Accesses across the peripheral bus occur in multiple cycles, and all accesses are serialized. A CPU access to the peripheral bus results in a CPU stall of several cycles, as shown in Table 5–1. A single CPU access to the peripheral bus stalls the CPU for five cycles, and parallel CPU accesses stall the CPU for nine cycles.

Table 5–1. CPU Stalls For Peripheral Register Accesses

CPU Access	CPU Stall
Single	5
Parallel	9

DMA accesses to the peripheral bus are pipelined, allowing the DMA to access peripheral bus every three cycles.

5.2.3 External Memory Interface (EMIF)

The external memory interface (EMIF) connects the DSP to external memory, such as synchronous dynamic RAM (SDRAM), synchronous burst static RAM (SBSRAM), and asynchronous memory. The EMIF also provides 8-bit-wide and 16-bit-wide memory read capability to support low-cost ROM memories (flash, EEPROM, EPROM, and PROM).

The EMIF supports burst capability to facilitate data transfers to/from high-speed memories. The DMA exercises this functionality through the use of its internal FIFO. Using the DMA, it is possible to access external memories at the rate of one data element per memory clock cycle. The CPU must wait for each data element required by the current execute packet before proceeding to the next execute packet. Thus data requests to the EMIF by the CPU are done one at a time, rather than in bursts, and do not take advantage of the burst capability of the EMIF.

To achieve its high-throughput for burst transfers, the EMIF has multiple internal pipeline stages. Due to this pipeline, there is latency incurred for a data transfer request both at the beginning of the burst request and at the end of the burst request. The number of cycles required for the actual data access depends on the type of memory being accessed.

To lessen the effects of memory access latencies, frequent data accesses to the EMIF should be performed by the DMA in bursts. Also, if there is potential for a frequent number of interruptions to burst activity by a higher priority requestor, the arbitration bit (RBTR8) can be set in the EMIF global control register. Setting this bit ensures that a minimum of eight accesses of a current burst is serviced before a higher priority requestor can use the EMIF. This functionality reduces the number of cycles lost to arbitration.

The number of cycles required to access an external memory location depends on two factors:

- **Type of external memory:** Different memory types have different cycle timings for data accesses.
- **Current EMIF activity:** If another resource is currently accessing external memory, the EMIF requires multiple cycles to flush its pipeline.

5.2.3.1 Memory Timings

The cycle timings for each memory type are provided in the data sheet for the particular C6000 device.

The access latency required for an external memory to be able to either return or receive a data element is defined in the datasheet, and it is specific to the type of memory. The beginning of the access is marked by the transition of the memory's chip enable ($\overline{\text{CE}}$) to active (low).

The time used by the EMIF at the end of an external data access is provided in Table 5–2. This table shows the number of CLKOUT1 cycles between the external strobe for a particular memory ($\overline{\text{AOE}}$, $\overline{\text{AWE}}$, $\overline{\text{SSOE}}$, $\overline{\text{SSWE}}$, or $\overline{\text{SDCAS}}$) and $\overline{\text{CE}}$ returning high for each of the memory types. These cycle counts are referred to as CE_READ_HOLD for reads, and CE_WRITE_HOLD for writes.

Access times to asynchronous memory are user-defined using programmable setup, strobe, and hold values. Read and write accesses can use different settings for each field. For a complete description, see Chapter 14, *Interrupt Selector and External Interrupts.*

Note:

The SBSRAM data provided in all the tables in this chapter are for the 1/2x rate SBSRAM. The TMS320C6201 and C6701 devices also support a 1x SBSRAM interface, the data for which can vary by 1-2 cycles. The 1x speed SBSRAM option is not commonly used. It is currently difficult to find devices that conform to the timing requirements. Therefore, the 1x-specific timing are not included in this chapter.

Table 5–2. EMIF Data Access Completion Timings in CLKOUT1 (CPU Clock) Cycles

Memory Type	$\overline{\text{CE}}$_READ_HOLD	$\overline{\text{CE}}$_WRITE_HOLD
Asynchronous	7 – READ_HOLD	4 if WRITE_HOLD = 0 3 if WRITE_HOLD > 0
SDRAM	0	0
SBSRAM	4	4

After the $\overline{\text{CE}}$ is re-asserted high at the end of a memory access, multiple cycles occur before another external access can begin due to arbitration within the EMIF. When the EMIF switches between requestors (or requests by the same requestor) there can be multiple cycles between external accesses (between active $\overline{\text{CE}}$ signals). These timings vary slightly depending on the memory type, the requestors, and the situation of the switching request.

5.2.3.2 CPU Accesses

The CPU uses the load and store operations to access data in external memory. Since accesses to external memory require multiple cycles to complete, the CPU stalls during the E3 stage of the pipeline. The data memory controller handles CPU accesses to the EMIF, with each request passed individually to the EMIF. The data memory controller waits until previous accesses have completed before issuing subsequent requests. This protocol prevents the CPU from bursting data accesses.

Table 5–3 provides the number of cycles for which the CPU stalls for an external access. SETUP, STROBE, and HOLD values are user-programmable fields of each CE control register in the EMIF. The CE_HOLD values ($\overline{\text{CE}}$_READ_HOLD or $\overline{\text{CE}}$_WRITE_HOLD) are provided in Table 5–2. TRP and TRCD are user-programmable fields of the SDRAM control register in the EMIF. The SDRAM and

SBSRAM timings have a range of two cycles due to the fact that the CPU request may have to wait until the appropriate phase of the external memory clock, which is half the rate of the CPU clock.

Table 5–3. CPU Stalls for Single External Data Accesses

Memory Type	Load	Store
Asynchronous	SETUP + STROBE + HOLD + CE_HOLD – 5	6
SDRAM (active row)	17 or 18	7 or 8
SDRAM (inactive row)	2 • (TRP + TRCD) + (25 or 26)	2 • (TRP + TRCD) + (15 or 16)
SBSRAM	15 or 16	7 or 8

The number of CPU stall cycles is at least the number given Table 5–3. However, the number of CPU stall cycles increases if the EMIF is currently completing a previous access. The number of cycles of additional delay depends upon whether the CPU has a higher priority than the current access, as well as how close the access is to completion. If the current access is a CPU store to asynchronous memory, the maximum number of additional cycles for which the CPU is stalled is SETUP+ STROBE+ HOLD+ CE_HOLD – 5. This maximum is obtained if both CPU accesses are submitted in parallel. For every cycle of delay between the two accesses, subtract one from the additional stall value (until the additional delay is zero). All other loads and stores do not result in an additional delay to a CPU access.

5.2.3.3 DMA Accesses

The DMA can burst data to and from external memory at the rate of one element per memory clock cycle. The DMA’s internal FIFO allows for data reads to be pipelined. Provided that the FIFO does not completely fill (which occurs if DMA writes are held off by a higher priority requestor), the DMA does not need to wait for a request to complete before issuing another.

A DMA access to external memory can achieve the maximum throughput rate for any external memory. By pipelining accesses, the time required to access a frame of data is equal to one memory clock per element for synchronous memories and the user-programmed settings for asynchronous memory.

5.2.4 Resource Contention

When multiple requestors (DMA or CPU) access the same resource, the resource’s memory controller arbitrates which accesses the memory location first. Data resources that can have multiple requestors include internal data memory banks, peripheral registers, and external memory (EMIF). The expansion bus is only accessible with the DMA.

Arbitration is performed every cycle within the memory controllers for each resource. For internal data memory and peripheral register accesses there is no delay when switching between requestors. A lower-priority request will take place on the cycle immediately following the high-priority request.

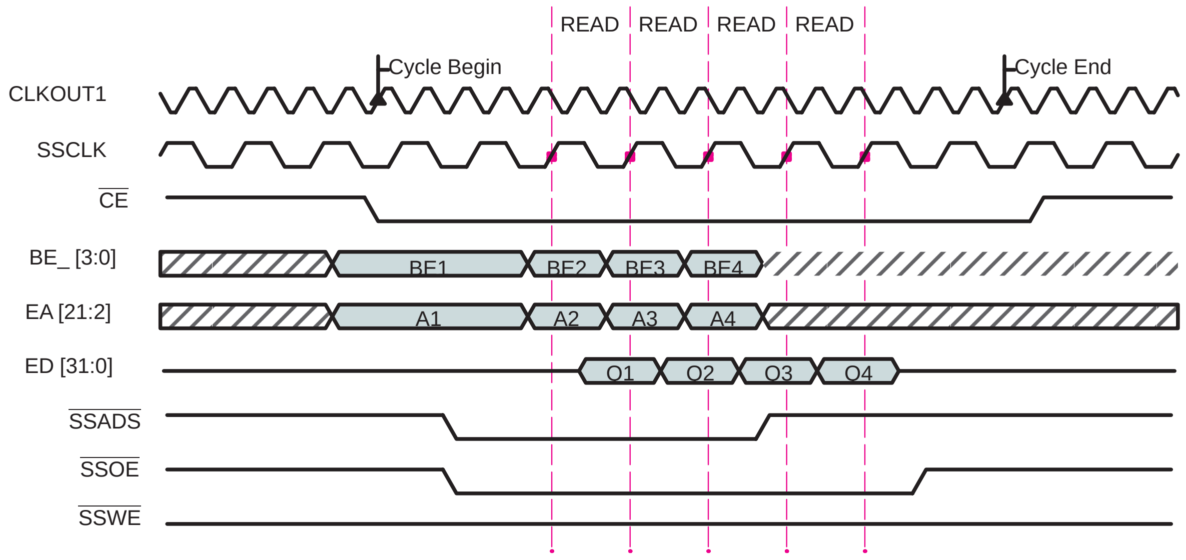
For accesses to external memory through the EMIF, the number of cycles in between accesses depends on the memory type and the direction of the accesses. Since memory timings and latencies vary according to different memory types, and since the external memories do not run off the CPU clock, the switching times are not uniform across all memory types. The delay times between external accesses are provided in Table 5–4. These switching times are valid for CPU/CPU, DMA/CPU, CPU/DMA, and DMA/DMA access boundaries.

Table 5–4. External Switching Time between Accesses by Different Requestors

		Subsequent Access					
		ASRAM		**SDRAM**		**SBSRAM**	
Current Access		**Read**	**Write**	**Read**	**Write**	**Read**	**Write**
ASRAM	Read	1–2	1–2	5–7	5–7	2–3	2–3
	Write	1	1	1	1	2–3	2–3
SDRAM	Read	12–15	12–15	16–18	18–20	13–17	13–17
	Write	4–5	4–5	10	8	5–7	5–7
SBSRAM	Read	2–4	2–4	7–9	7–9	4–6	4–6
	Write	2–3	2–3	5–7	5–7	4	4

The switching times signify the number of CLKOUT1 (CPU Clock) cycles between the end of one access and the beginning of another. Within this chapter, the beginning of an access is the rising edge of CLKOUT1 (CPU Clock) immediately preceding the rising edge of the memory clock used in the data sheet to reference the $\overline{\text{CE}}$ transition from 1 to 0. The end of an access is the rising edge of CLKOUT1 (CPU Clock) immediately preceding the rising edge of the memory clock used in the datasheet to reference the $\overline{\text{CE}}$ transition from 0 to 1. Figure 5–2 shows an example of the beginning and the end of an SBSRAM transfer. For the C6201 SBSRAM, the memory clock SSCLK is used to reference the $\overline{\text{CE}}$ transitions

Figure 5–2. 1/2x Rate SBSRAM Read Cycle Timings



5.2.4.1 Switching Between DMA Channels

Switching between DMA channels is different than switching between CPU data paths or between the CPU and the DMA. Arbitration is handled within the DMA for channels that are active at the same time. Arbitration is done in two places: the read port and the write port. Only one DMA channel can read at a time, and only one channel can write at a time. Priority is always granted to the lowest-numbered DMA channel. The Auxiliary channel’s priority is programmable.

A DMA channel requests a read access as soon as it is started by the CPU, or when its read-synchronization event is received. A write access is requested as soon as data arrives at one of the channel’s holding registers, and after any write-synchronization event. A channel’s request to either the read- or write-controller is either passed to the desired resource or held due to a higher priority channel using the port. Arbitration is handled every cycle.

This arbitration takes only one cycle and has virtually no impact for internal transfers. For transfers to or from external memory, this is more noticeable. The number of cycles between accesses by different channels depends on three factors:

❑ Memory type
❑ Transfer direction
❑ Channel priority

Switching times between accesses depends on memory type and direction. The time varies from the values in Table 5–4, depending on whether one channel interrupts another, or one channel completes and a suspended channel resumes (or begins). The switching time between accesses by different channels is equal to the values given in Table 5–4, plus an additional offset shown in Table 5–5.

Table 5–5. Additional Switching Time between External DMA Accesses

	Subsequent DMA Access			
	Higher-Priority Channel		**Lower-Priority Channel**	
Current DMA Access	**Read**	**Write**	**Read**	**Write**
Read	2–4	11–15	2–4	8–15
Write	0–4	0–4	4–8	0–4

If a DMA channel’s request is passed to the read- or write-controller when a lower priority channel is transmitting data, the lower priority channel’s new requests are suspended and the higher priority channel is permitted to transfer.

Since most data accesses to external memory use the DMA, knowledge of the time required to switch between DMA channels is important. A DMA channel accessing external memory will hold off all other requests for external memory until that access is complete. The switching time depends on the type of memory accessed, the directions of both the current and subsequent transfers, and whether or not either DMA channel uses the internal FIFO to burst.

5.2.4.2 Burst Interruptions

External DMA bursts can be interrupted without another requestor taking over the EMIF. Examples of this include transfer frame boundaries, servicing of a McBSP by a higher-priority channel, or internal auxiliary channel accesses.

Multiple Frames and Auto-initialization

The DMA channels can be configured to transmit multiple frames of equal length, constituting a block. Also each channel can be optionally configured to perform auto-initialization, where some or all address and counter registers are reloaded at the end of each block, in order to continue transmission without CPU intervention. These functions of the DMA complete quickly, each requiring only one cycle within the DMA block.

Once a frame or block boundary is reached, the current burst ends and a new burst begins. As shown in Table 5–6 the number of inactive cycles during an external burst depends on the type of memory being accessed, as well as the direction of the transfer.

Table 5–6. CLKOUT1 (CPU Clock) Cycles between External Frame Bursts

Memory Type	Cycles Between Reads	Cycles Between Writes
Asynchronous	1	1
SDRAM	16	10
SBSRAM	4	4

Servicing a McBSP or Host Access

When a higher priority channel services a McBSP, or when the auxiliary channel (higher priority) services a host access, it temporarily assumes control of the read port and the write port of the DMA from the currently bursting channel. The amount of time lost from this interruption depends on the memory type being accessed, as well as the direction of the current burst. Table 5–7 shows the time between memory accesses ($\overline{\text{CE}}$ inactive) for each type of external memory. The cycle counts assume that the data source and destination for the McBSP or host transfers are internal data memory.

Table 5–7. Burst Interruption by McBSP/Host Service

		Burst Cycles Idle When Servicing McBSP/Host		
Current Access		McBSP Read/ Host Write	McBSP Write/ Host Read	McBSP Read & Write/ Host peripheral access
ASRAM	Read	12	14	22
	Write	2	2	13
SDRAM	Read	28	30	38
	Write	16	14	28
SBSRAM	Read	16	18	26
	Write	10	8	22

Internal Auxiliary Channel Access

External accesses by the DMA auxiliary channel act as a DMA channel that is interrupting another DMA channel. This described in *section 5.2.4.1 Switching Between DMA Channels*.

5.2.5 DMA Synchronization

Each DMA channel can be synchronized by a variety of events to control its data movement. There is latency involved with performing a synchronized transfer that should be understood when computing response time of a system. Table 5–8 shows the time delay from a synchronization event to the beginning of a data access.

Table 5–8. DMA Synchronization Timings

Timing	CLKOUT1 (CPU Clock) Cycles for Memory Type		
	ASRAM	SDRAM	SBSRAM
Internal event to setting of (R/W)SYNC_STAT bit	1	1	1
External event to setting of (R/W)SYNC_STAT bit	4	4	4
RSYNC_STAT bit set to beginning of external read	9	12	11
RSYNC_STAT bit set to beginning of external write†	16	20	17
WSYNC_STAT bit set to beginning of external write	7	11	8

† Measurement valid for read-synchronized or frame-synchronized internal-to-external DMA transfer.

5.2.6 Transferring To/From Same Resource

The DMA can transfer data to and from the same resource. Two primary applications for this would be to restructure data in internal memory, or to burst data between its external source and an external buffer.

Using the same resource reduces the throughput achievable by the DMA. This situation results because the DMA cannot read from and write to the same resource simultaneously.

DMA writes are given a higher priority than DMA reads. A DMA channel issues write requests as long as there is data in its holding registers. Because of this, a channel that attempts to burst to the same resource from which it is reading cannot capitalize on the DMA FIFO. Since the write requests begin as soon as data is in the channel’s holding registers, and the write request has priority over the read requests, the number of elements buffered during the read burst depends solely on the speed of the memory being read. If a slow memory is being read (i.e. asynchronous memory) then only a few elements burst at a time. If a high-speed memory is being read (i.e. internal data memory) then more of the FIFO is used.

Table 5–9 lists the number of elements per burst when reading from, and writing to, the same resource.

Table 5–9. *Burst Size for Shared Resource*

Read Memory	**Elements/Burst**
Internal Data Memory	5
Asynchronous	3*
SDRAM	6
SBSRAM	5

Note: Burst size is 2 when READ_HOLD = 3

When the DMA uses the same resource for both source and destination, switching latency exists between reading and writing. This latency depends on the transition. Table 5–10 lists the number of cycles between reads and writes for transfers between external memories, in addition to those provided in Table 5–4.

Table 5–10. *Additional Switching Time for External-to-External Transfers*

Burst Transition	**Additional Cycles**
Read to Write	0 – 4
Write to Read	1 – 4

Latencies also exist between read bursts and write bursts when transferring between locations in internal data memory. While each data access can be performed in a single cycle, there is switching time between the read requests and the write requests, as provided in Table 5–11.

Table 5–11. *Switching Time for Internal-to-Internal Transfers*

Burst Transition	**CLKOUT1 Cycles**
Read to Write	8
Write to Read	9

Due to the burst sizes and the latencies in Table 5–10 and Table 5–11, the throughput of a transfer with a shared resource for the source and destination is maximized for frame sizes equal to a multiple of the above burst sizes.

5.3 Bandwidth Calculation

If the system activity is known, then the total bandwidth required by the system can be derived from the information presented in this chapter. Such an analysis allows a designer to make sure that all data processing can be performed in the time allotted for it.

5.3.1 Simple Timing Use Example

The following simple example illustrates how to use the timing information in this chapter. Consider the following:

- ❑ DMA channel 0 performs an unsynchronized transfer of 32-bit data from ½x SBSRAM to internal data memory with a frame count of 2 and an element count of 20.
- ❑ DMA channel 1 performs an unsynchronized transfer of 32-bit data from internal data memory to ½x clock rate SBSRAM with a frame count of 1 and an element count of 40.
- ❑ DMA channel 1 is started immediately after channel 0.

First it is necessary to find out the bandwidth requirements for the individual data streams. All of the timing parameters used in the calculations, along with their location in this chapter, are described in Table 5–12.

Table 5–12. Timing Parameter Descriptions For Simple Example

Parameter	Value	Location	Description
element_count0	20	N/A	Number of elements per frame for channel 0
element_count1	40	N/A	Number of elements per frame for channel 1
CE_read_setup	4	N/A	Cycle count from the beginning of the access to beginning of the first read data phase.
CE_read_hold	4	Table 5–2, p. 5-6	Cycle count from the last strobe in a read burst from SBSRAM to the end of the access
CE_write_hold	4	Table 5–2, p. 5-6	Cycle count from the last strobe in a write burst to SBSRAM to the end of access
read_frame_gap	4	Table 5–6, p. 5-11	Time between read bursts for multi-frame transfer
read_to_write	6	Table 5–4, p. 5-8	Switching time between a SBSRAM read access and a SBSRAM write access
DMA_hp_read_to_lp_write	15	Table 5–5, p. 5-10	Additional switching time between a high priority read access and a low priority write access

Table 5–12. Timing Parameter Descriptions For Simple Example (Continued)

Parameter	Value	Location	Description
start_to_sync	1	Table 5–8, p. 5-12	Time from setting START = 01b to the setting of RSYNC_STAT
RSYNC_STAT_to_read	11	Table 5–8, p. 5-12	Latency from the setting of RSYNC_STAT to beginning of a read access

Since channel 1 is started after channel 0, it waits until channel 0's transfer completes before beginning its data transfer. The total transfer time equals the transfer time of channel 0 plus the transfer time of channel 1 plus the time between transfers, or:

Channel 0 Burst Time + Channel 0 Overhead + Channel 1 Burst Time + Channel 1 Overhead

The functions of each are summarized as follows:

❑ **Channel 0 Burst Time:** DMA channel 0 performs two burst transfers, one for each frame. The cycle time required for all bursts is:

$2 \bullet [CE\_read\_setup + (2 \bullet element\_count0) + CE\_read\_hold]$

$= 2 \bullet [4 + (2 \bullet 20) + 4] = 96$ *cycles*

❑ **Channel 0 Overhead:** The first frame starts after the RSYNC_STAT bit is set. Since channel 0 performs an unsynchronized transfer, RSYNC_STAT is set 1 cycles after START = 01b is written to the channel's primary control register. The time between frames must also be included in the overhead calculation, since there is a small number of cycles between bursts. This delay is calculated as:

$Start\_to\_sync + RSYNC\_STAT\_to\_read + read\_frame\_gap$

$= 1 + 11 + 4 = 16$ *cycles*

❑ **Channel 1 Burst Time:** DMA channel 1 performs only a single burst, which requires the following number of cycles to complete:

$(2 \bullet element\_count1) + CE\_write\_hold$

$= (2 \bullet 40) + 4 = 84$ *cycles*

- **Channel 1 Overhead:** Since channel 1 is started during channel 0's transfer, the delay from starting the channel to the actual beginning of the transfer is not apparent. Rather than the time delay from the setting of the START field to the beginning of the transfer (as for Channel 0), the overhead consists only on the delay between channel 0's transfer and channel 1's transfer. This delay is calculated by:

  read_to_write + DMA_hp_read_to_lp_write

  = 6 + 15 = 21 *cycles*

- **Total Transfer Time:** The total time required for these transfers, from the setting of START = 01b in channel 0's primary control register to the end of the $\overline{\text{CE}}$ period for channel 1 is:

  Channel 0 Burst Time + Channel 0 *Overhead + Channel* 1 *Burst Time + Channel* 1 *Overhead*

  = 96 + 16 + 84 + 21 = 217 *cycles*

5.3.2 Complex Bandwidth Calculation Example

A more complex example involves calculating the bandwidth requirement of a system, ensuring that the system requirements do not exceed the capabilities of the device. The system involves the following transfers:

- Full-duplex serial data transferred to/from a McBSP at 48 kHz
- Data input from an asynchronous memory source with setup = 2, strobe = 4, hold = 1. Data arrives in frames of 128 elements every 10 µs.
- Data output from an asynchronous memory source with setup = 2, strobe = 4, hold = 1. Data is output in frames of 128 elements every 15µs.
- The CPU is restricted to internal memory and is running at 200 MHz.

First, calculate the bandwidth requirements for the individual data streams. Then, consideration for the interaction between the DMA transfers should be included. Table 5–13 describes all of the timing parameters used in the calculations, along with their location in this document.

Table 5–13. Timing Parameter Descriptions For Complex Example

Parameter	Value	Location	Description
element_count	128	N/A	Number of elements per frame
Setup	2	Memory Timings, p. 5-5	Read/write setup time (same in this example)
Strobe	4	Memory Timings, p. 5-5	Read/write strobe time (same in this example)
Hold	1	Memory Timings, p. 5-5	Read/write hold time (same in this example)
CE_read_hold	6	Table 5–2, p. 5-6	Cycle count from the last strobe in a read burst from asynchronous memory to the end of access
CE_write_hold	3	Table 5–2, p. 5-6	Cycle count from the last strobe in a write burst to asynchronous memory to the end of access
mcbsp_read_interruption	12	Table 5–7, p. 5-11	ASRAM burst interruption caused by a McBSP read
mcbsp_write_interruption	14	Table 5–7, p. 5-11	ASRAM burst interruption caused by a McBSP write
write_to_read	1	Table 5–4, p. 5-8	Switching time between an asynchronous write access to an asynchronous read access
read_to_write	2	Table 5–4, p. 5-8	Switching time between an asynchronous read access to an asynchronous write access
DMA_lp_write_to_hp_read	4	Table 5–5, p. 5-10	Additional switching time between a low priority write access and a high priority read access
DMA_hp_read_to_lp_write	15	Table 5–5, p. 5-10	Additional switching time between a high priority read access and a low priority write access
RSYNC_STAT_to_read	9	Table 5–8, p. 5-12	Latency from the setting of RSYNC_STAT to beginning of an ASRAM read access

Since this is a bandwidth calculation, rather than a latency (or completion time) calculation, the worst case interaction of the DMA transfers must be taken into account. The bandwidth requirement of the system will be equal to the transfer time required by each of the channels plus any arbitration latency introduce by channel interaction during a timing window of the least common denominator of the transfer times. This calculation is represented by:

(Input data transfer time + Output data transfer time + McBSP data transfer overhead + Input data transfer overhead + Output data transfer overhead) / Timeslice • 100%

McBSP Data Transfer Time: The serial output data requires a transfer from internal data memory to the McBSP once per 4167 cycles.

The serial input data requires a transfer from the McBSP to internal data memory once per 4167 cycles.

Input Data Transfer Time: The parallel input data requires the following number of cycles every 2000 cycles:

[(setup + strobe) • element_count] + [hold • (element_count – 1)]

+ CE_read_hold

= [(2 + 4) • 128] + [1 • (128 – 1)] + 6 = 901 cycles

Output Data Transfer Time: The parallel output data requires the following number of cycles every 3000 cycles:

[(setup + strobe) • element_count] + [hold • (element_count – 1)]

+ CE_write_hold

= [(2 + 4) • 128] + [(1) • 127] + 3 = 898 cycles

Timeslice Calculation: The serial sync event arrives roughly every 4000 cycles, the parallel input every 2000 cycles, and the parallel output every 3000 cycles. Considering all events in a 12000 window is therefore adequate for the entire system. The least common denominator of the transfer times is 12000.

5.3.2.1 DMA Channel Selection

The DMA channels used for each of the data transfers directly impacts the performance of the system. Typically the transfers in a system should be ranked in priority such that short bursts (such as McBSP servicing) are given the highest priority and long bursts (typically background paging) are given the lowest priority. For transfers that are of similar burst lengths the more frequent transfer is given priority. This ensures that data being sampled at a high frequency is never missed. System constraints and special cases can require that a different priority scheme be used. For the example in Table 5–14, transfer priority is ranked according to frequency. Based on the numbers shown in the time-slice calculation the priority ranking is as shown.

Table 5–14. DMA Channel Selection Priority

Data Transfer	Burst Size	Event Frequency	DMA channel
McBSP	1	1/4000 cycles	0
Parallel Input	128	1/2000 cycles	1
Parallel Output	128	1/3000 cycles	2

Based on this, channel 0 should be used for the serial data, channel 1 for the parallel input data, and channel 2 for the parallel output data.

McBSP Data Transfer Overhead: DMA channel 0 will interrupt either channel 1 or 2 twice if serial frames are synchronized at the same time, but potentially four separate times. This worst-case results in the following number of additional cycles:

2 • *(CE_read_hold + mcbsp_read_interruption) for McBSP reads and*
2 • *(CE_write_hold + mcbsp_write_interruption) for McBSP writes*

= 2 • [(6 + 12) +(3 + 14)] = 70 *cycles every* 12000 *cycles*

Input Data Transfer Overhead: DMA channel 1 interrupts channel 2 four times, resulting in the following number of additional cycles:

4 • *(CE_write_hold + write_to_read + DMA_lp_write_to_hp_read)*

= 4 • (3 + 1 + 4) = 32 *cycles every* 12000 *cycles*

Output Data Transfer Time: DMA channel 2 trails channel 1 six times, resulting in the following number of additional cycles:

6 • *(read_to_write + DMA_hp_read_to_lp_write)*

= 6 • (2 + 15) = 102 *cycles every* 12000 *cycles*

Total Bandwidth Utilization: Again, this is the total bandwidth required by the system:

(Input data transfer time + Output data transfer time + McBSP data transfer overhead + Input data transfer overhead + Output data transfer overhead)/Timeslice · 100%)

= ((6 • 901) + (4 • 898) + 70 + 32 + 102) / 12000 • 100%
= 9202 / 12000 • 100%
= 77%

5.3.2.2 Comparison of 1.8V/2.5V Devices to 1.5V Device

For the 1.5 V C6000 devices (such as C6202B/C6203(B)/C6204/C6205), the bandwidth analysis ends here. Since only 9132 out of every 12000 cycles are required for the transfers in the system, or 76%, there is no problem servicing the I/O data streams.

For the 1.8 V (such as C6201/C6701/C6202) devices, however, some additional steps must be taken due to the shared FIFO present in the DMA. As described in section 4.11.2.2 *Channel FIFOs*, having the shared FIFO can reduce throughput when a high-priority burst transfer interrupts a lower-priority burst transfer, when the source of the high-priority transfer is the same resource as the destination of the low-priority transfer. This condition exists in the above example when channel 1 interrupts channel 2. To solve this problem, the CPU must be used to control the burst interruption to insure that channel 1 always has use of the shared FIFO when active.

To do this, the channel should no longer be synchronized on the external event, but rather on an unused synchronization event. The CPU should be configured to receive an interrupt from the external event (previously used to synchronized channel 1). The ISR for the interrupt should perform the following tasks:

❑ Pause channel 2
❑ Set RSYNC_STAT for channel 1
❑ Unpause channel 2

The ISR executes six times per 12000 cycles. This switching time replaces the 32 cycles previously described for channel 1 interrupting channel 2. Instead, the number of cycles will be:

6 • *(RSYNC_STAT_to_read)*

= 6 • 9
= 54 *cycles every* 12000 *cycles*

This changes the total cycle requirement to 9202 – 32 + 54 = 9224 cycles every 12000 cycles, which is still 77% bandwidth utilization. The required cycle count will grow, however, if the ISR for channel 1 is delayed from execution. 2846 cycles remain in each 12000 window for the ISR to occur six times. In order to provide the CPU intervention, the ISR must complete (CPU interrupt {external event} to the setting of RSYNC_STAT) within 2846/6 = 475 cycles.

Note:

This cycle count can be an average if it is guaranteed that the completion time for six ISRs not exceed 2846 cycles, the “free” cycles accounted for in the calculation.

5.4 Bandwidth Optimization

Understanding the time requirements of a system is crucial to building it successfully. By knowing the time requirements for data I/O, and utilizing the DSP timing information presented in the previous sections, it is possible to tailor CPU and DMA activity to work as efficiently as possible to meet performance goals.

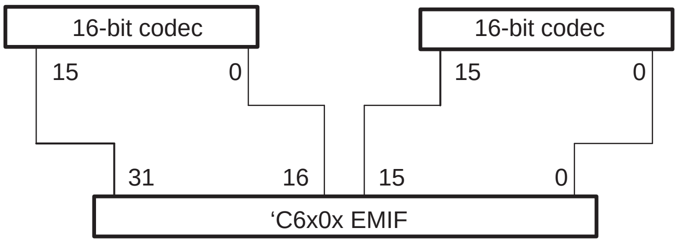
There are typically multiple ways to implement tasks, both with the CPU and with the DMA. Understanding the implications of the different options can allow the best to be chosen.

5.4.1 Maximize DMA Bursts

The most important things to consider when accessing external memory is that bursts are the most efficient way to access data. Data bursts are performed through a non-synchronized or frame-synchronized DMA transfer. Each frame is one continuous burst. To maximize data bandwidth, data should always be transferred using the largest frame size possible, and should be transferred as 32-bit elements regardless of the data size that will be used by the CPU.

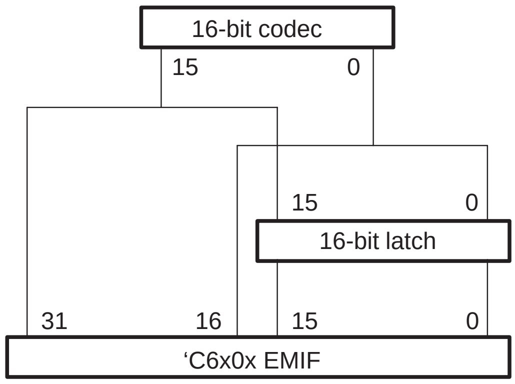
Accessing 32-bit data with the DMA can be accomplished when the data source is 16-or 8-bit by adding or organizing system hardware. If multiple 16-bit codecs are providing data I/O for the system, then two codecs can be located per word address, with one on the lower 16-bits and the other on the upper 16-bits, as shown in Figure 5–3. This allows for both to be accessed simultaneously. This requires synchronization of the data streams to insure that valid data is always read.

Figure 5–3. Combining External Peripherals



If only one 16-bit data I/O source is present, the system bandwidth is greatly improved by providing an external latch, as shown in Figure 5–4. When an odd 16-bit data element arrives, latch it into one halfword on the data bus. When an even 16-bit data element arrives, access both elements with a 32-bit transfer. Thus, the bandwidth is effectively doubled.

Figure 5–4. Converting a 16-bit Peripheral to 32-bit



If the cost is acceptable, an external FIFO could be placed between the data source and the DSP to buffer a frame of data. A DMA channel could then burst a full frame of data elements when the FIFO fills. By bursting data the bandwidth of the system is maximized.

5.4.2 Minimizing CPU/DMA Conflict

As the CPU is optimized for internal accesses, it cannot burst data from external memory. CPU data accesses should therefore be restricted to internal data memory as much as possible. Using the DMA to page data in and out of internal memory allows better processing speeds to be achieved.

Conflict between DMA channels, and between the CPU and DMA should be minimized. When a high-priority DMA or CPU access interrupts a DMA burst, cycles are lost as the burst is broken. By performing all CPU data accesses in a single block (i.e. one after the next in a small section of code), rather than dispersed throughout a section of code, each data requestor can have the full system bandwidth. The DMA burst is only interrupted a single time the transfer rate is not heavily impacted.

In some systems the above solutions may not be practical, particularly if bandwidth is restricted by hardware or by asynchronous events. If conflict cannot be completely avoided, then bandwidth can still be efficiently used.

Sometimes the CPU must be used to access external memory. The most frequent example is when all DMA channels are heavily used to perform other tasks. In this case, it is more inefficient to save the context of a DMA channel, then page data to internal memory, then restore the channel's context.

If the CPU must be used to access external memory, the accesses should be performed consecutively—either one serial instruction after another or in parallel. This reduces the effect of interrupting a DMA burst to/from external memory. Since there is a loss of several cycles in between accesses by the DMA and CPU, as described in , these “lost” cycles can be minimized if the DMA burst is interrupted only once per group of CPU accesses.

Chapter 6

EDMA Controller

This chapter describes the enhanced DMA controller (EDMA) for the TMS320C621x/C671x/C64x™. EDMA transfer parameters, types and performance are discussed. This chapter also describes the quick DMA (QDMA) for fast data requests by the CPU.

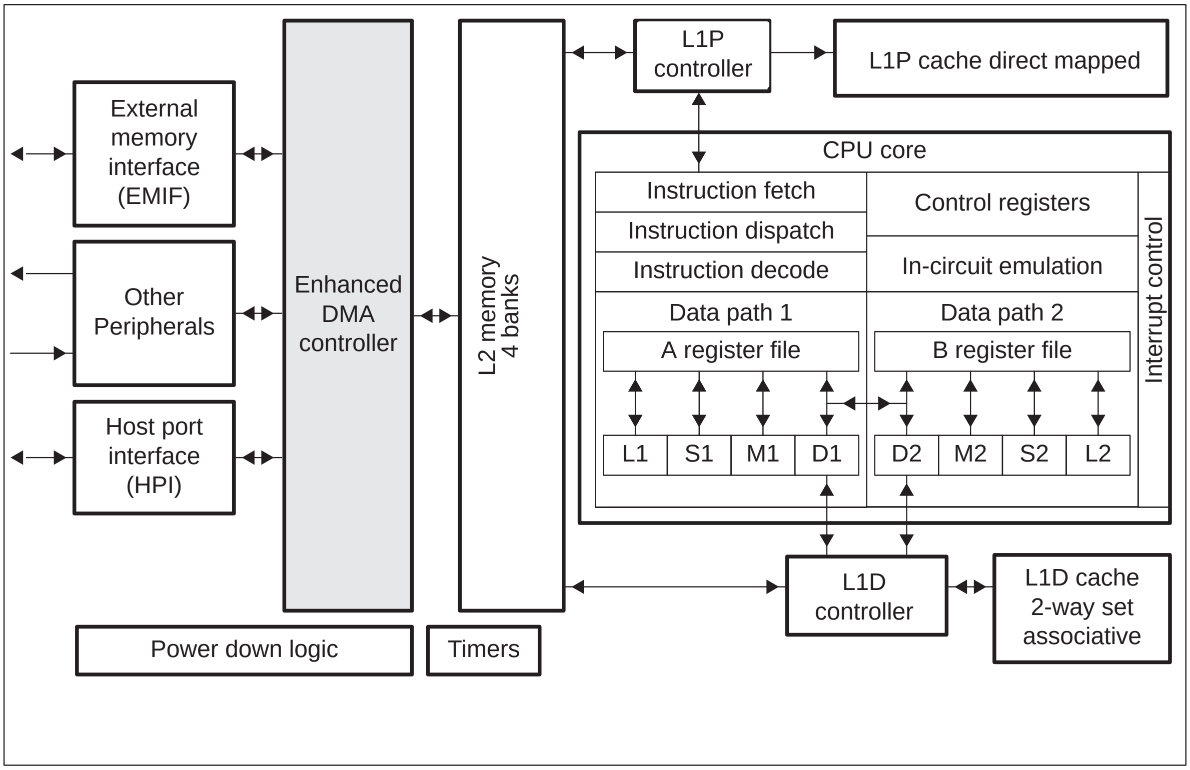
Topic **Page**

6.1 Overview

The enhanced direct memory access (EDMA) controller handles all data transfers between the level-two (L2) cache/memory controller and the device peripherals on the TMS320C621x/C671x/C64x. These data transfers include cache servicing, non-cacheable memory accesses, user-programmed data transfers, and host accesses.

The EDMA controller in the C621x/C671x/C64x is different in architecture to the previous DMA controller in the C620x/C670x devices. The EDMA includes several enhancements to the DMA in that it provides 64 channels (C64x) or 16 channels C621x/C671x, with programmable priority, and the ability to link and chain data transfers. The EDMA allows movement of data to/from any addressable memory spaces, including internal memory (L2 SRAM), peripherals, and external memory.

Figure 6–1. TMS320C621x/C671x/C64x Block Diagram

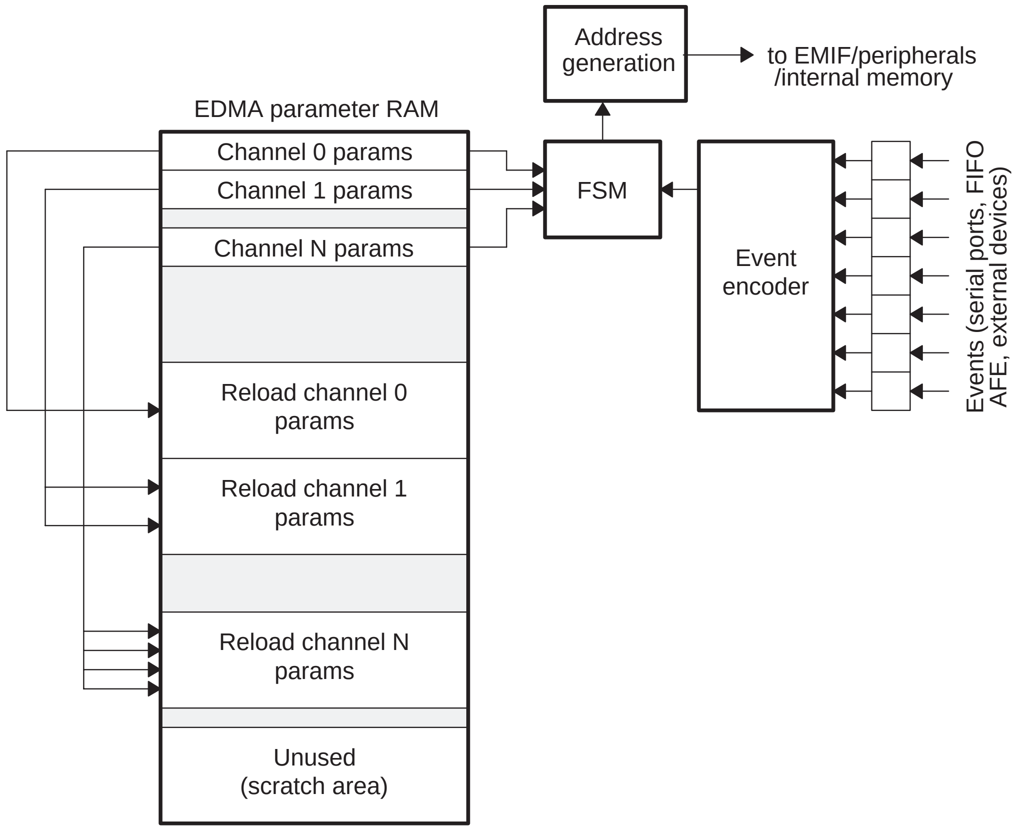


The EDMA controller comprises:

- ❑ 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 are stored in the EDMA parameter RAM, and are passed onto the address generation hardware, which address the EMIF and/or peripherals to perform the necessary read and write transactions.

The EDMA has the capability of performing fast and efficient transfers by accepting a quick DMA (QDMA) request from the CPU. A QDMA transfer is best suited for applications that require quick data transfers, such as data requests in a tight loop algorithm. See section 6.19 for more details. Table 6–1 summarizes the difference between the C6000 EDMAs.

Table 6–1. Differences in TMS320C6000 EDMAs

Features	**Supported on Device**	**Described in Section**
Number of channels	C64x has 64 channels. C621x/C671x has 16 channels	6.1
CIPR, CIER, CCER, ER, EER, ECR, ESR registers	For C64x, each of these registers are expanded into two registers (low and high) to support 64 channels.	6.3
Transfer Chaining on all channels	C64x only. On C621x/C671x, only channels 8 to 11 can be chained	6.15
Supports Peripheral Device Transfers	C64x only	6.16
Programmable priority queue allocation	C64x only	6.17.2.2
EDMA transfers possible on all priority queues	C64x only. On C621x/C671x, EDMA cannot transfer on Q0.	6.17.2.2
L2 controller transfers possible on all priority queues	C64x only. On C621x/C671x, L2 controller transfers on Q0 only.	6.17.2.2
Event polarity selection	C64x only	6.3.5
Alternate Transfer Complete Chaining and Interrupt	C64x only	6.14.2, 6.15.3

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 synchronization event if required. The term ’element transfer’ is used in context with 1-dimensional transfer, which is described below.
- **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 1-dimensional (1D) transfer, which 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. 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. For 1-dimensional transfers, a group of frames form a block. For 2-dimensional transfers, a group of arrays form a block.
- **1-dimensional (1D) transfer:** A group of frames comprise a 1D block. The number of frames (frame count, FRMCNT) in a block can be between 1 and 65536 (corresponding to an FRMCNT value range of 0 to 65535). The number of elements per frame can be between 1 and 65535. Either elements or full frames can be transferred at a time.
- **2-dimensional (2D) transfer:** A group of arrays comprise a 2D block. The first dimension is the number of contiguous elements in an array, and the second dimension is the number of such arrays. The number of arrays (array count, FRMCNT) in a block can range from 1 to 65536 (corresponding to an FRMCNT value range of 0 to 65535). Either arrays or the entire block can be transferred at a time.

6.3 Event Processing and EDMA Control Registers

Each of the 64 channels (C64x) or 16 channels (C621x/C671x) in the EDMA has a specific synchronization event associated with it. 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 6–2. These synchronization events are discussed in detail in section 6.7.1.

Table 6–2. EDMA Control Registers

Byte Address	Acronym	Register Name	Section
01A0 FF9Ch	EPRH	Event polarity high register †	6.3.5
01A0 FFA4h	CIPRH	Channel interrupt pending high register †	6.14
01A0 FFA8h	CIERH	Channel interrupt enable high register †	6.14
01A0 FFACh	CCERH	Channel chain enable high register †	6.15
01A0 FFB0h	ERH	Event high register †	6.3.1
01A0 FFB4h	EERH	Event enable high register †	6.3.2
01A0 FFB8h	ECRH	Event clear high register †	6.3.3
01A0 FFBCh	ESRH	Event set high register †	6.3.4
01A0 FFC0h	PQAR0	Priority queue allocation register 0 †	6.17.2.2
01A0 FFC4h	PQAR1	Priority queue allocation register 1 †	6.17.2.2
01A0 FFC8h	PQAR2	Priority queue allocation register 2 †	6.17.2.2
01A0 FFCCh	PQAR3	Priority queue allocation register 3 †	6.17.2.2
01A0 FFDCh	EPRL	Event polarity low register †	6.3.5
01A0 FFE0h	PQSR	Priority queue status register	6.17.1
01A0 FFE4h	CIPR	Channel interrupt pending register (C621x/C671x)	6.14
	CIPRL	Channel interrupt pending low register (C64x)	
01A0 FFE8h	CIER	Channel interrupt enable register (C621x/C671x)	6.14
	CIERL	Channel interrupt enable low register (C64x)	
01A0 FFECh	CCER	Channel chain enable register(C621x/C671x)	6.15
	CCERL	Channel chain enable low register (C64x)	
01A0 FFF0h	ER	Event register(C621x/C671x)	6.3.1
	ERL	Event low register (C64x)	
01A0 FFF4h	EER	Event enable register(C621x/C671x)	6.3.2
	EERL	Event enable low register (C64x)	
01A0 FFF8h	ECR	Event clear register(C621x/C671x)	6.3.3
	ECRL	Event clear low register (C64x)	
01A0 FFFCh	ESR	Event set register(C621x/C671x)	6.3.4
	ESRL	Event set low register (C64x)	

† Applicable to C64x only.

6.3.1 Event Register (ER, ERL, ERH)

All events are captured in the event register (ER), even when the events are disabled. The C621x/C671x has only one event register (ER). The C64x has two event registers, event low register (ERL) and event high register (ERH) for the 64 channels. For the remaining of this chapter, the term “event register” or “ER” refers to the ER for C621x/C671x or the ERL/ERH for C64x. The ER shown in Figure 6–3 contains one bit for each event. Section 6.7.1 describes the type of synchronization events and the EDMA channels associated with each of them.

Figure 6–3. Event Register (ER, ERL, ERH)

C621x/C671x: 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

C64x: Event Low Register (ERL) for events 0 to 31

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EVT31	EVT30	EVT29	EVT28	EVT27	EVT26	EVT25	EVT24	EVT23	EVT22	EVT21	EVT20	EVT19	EVT18	EVT17	EVT16
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

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

C64x: Event High Register (ERH) for events 32 to 63

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EVT63	EVT62	EVT61	EVT60	EVT59	EVT58	EVT57	EVT56	EVT55	EVT54	EVT53	EVT52	EVT51	EVT50	EVT49	EVT48
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

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EVT47	EVT46	EVT45	EVT44	EVT43	EVT42	EVT41	EVT40	EVT39	EVT38	EVT37	EVT36	EVT35	EVT34	EVT33	EVT32
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

6.3.2 Event Enable Register (EER, EERL, EERH)

In addition to the event register, the EDMA controller also provides the user the option of enabling/disabling events. Any of the event bits in the event enable register shown in Figure 6–4 can be set to ‘1’ to enable that event. The C621x/C671x has only one event enable register (EER). The C64x has two event enable registers, event enable low register (EERL) and event enable high register (EERH) for the 64 channels. For the remaining of this chapter, the term “event enable register” or “EER” refers to the EER for C621x/C671x or the EERL/EERH for C64x.

For the C621x/C671x, events 8–11 are only available for chaining of EDMA events. Therefore they are enabled in the Channel Chain Enable Register, described in section 6.15. Bits 8–11 in the EER are read-only for C621x/C671x. 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.

Figure 6–4. Event Enable Register (EER, EERL, EERH)

C621x/C671x: Event Enable Register (EER)

31–16
Reserved
R, +0

15	14	13	12	11–8	7	6	5	4	3	2	1	0
EE15	EE14	EE13	EE12	Reserved	EE7	EE6	EE5	EE4	EE3	EE2	EE1	EE0
RW,+0	RW,+0	RW,+0	RW,+0	R, +0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0

C64x: Event Enable Low Register (EERL) for events 0 to 31

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EE31	EE30	EE29	EE28	EE27	EE26	EE25	EE24	EE23	EE22	EE21	EE20	EE19	EE18	EE17	EE16
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
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

C64x: Event Enable High Register (EERH) for events 32 to 63

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EE63	EE62	EE61	EE60	EE59	EE58	EE57	EE56	EE55	EE54	EE53	EE52	EE51	EE50	EE49	EE48
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
EE47	EE46	EE45	EE44	EE43	EE42	EE41	EE40	EE39	EE38	EE37	EE36	EE35	EE34	EE33	EE32
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

6.3.3 Event Clear Register (ECR, ECRL, ECRH)

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. The C64x has two event clear registers, event clear low register (ECRL) and event clear high register (ECRH), for the 64 channels. For the remaining of this chapter, the term “event clear register” or “ECR” refers to the ECR for C621x/C671x, or the ECRL/ECRH for C64x. 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.

Figure 6–5. Event Clear Register (ECR, ECRL, ECRH)

C621x/C671x: Event Clear Register (ECR)

31–16
Reserved
RW,+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

C64x: Event Clear Low Register (ECRL) for Events 0 to 31

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EC31	EC30	EC29	EC28	EC27	EC26	EC25	EC24	EC23	EC22	EC21	EC20	EC19	EC18	EC17	EC16
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
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

C64x: Event Clear High Register (ECRH) for events 32 to 61

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EC63	EC62	EC61	EC60	EC59	EC58	EC57	EC56	EC55	EC54	EC53	EC52	EC51	EC50	EC49	EC48
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
EC47	EC46	EC45	EC44	EC43	EC42	EC41	EC40	EC39	EC38	EC37	EC36	EC35	EC34	EC33	EC32
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

6.3.4 Event Set Register (ESR, ESRL, ESRH)

The CPU can also set events by way of the event set register (ESR) shown in Figure 6–6. The C64x has two event set registers, event set low register (ESRL) and event set high register (ESRH), for the 64 channels. For the remaining of this chapter, the term “event set register” or “ESR” refers to the ESR for C621x/C671x or the ESRL/ESRH for C64x. Writing a ‘1’ to one of the 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 ESR bit is set and is not triggered by the associated event. See also section 6.19 for a description of the QDMA, an alternative way to perform CPU-initiated EDMA transfers.

Figure 6–6. Event Set Register (ESR, ESRL, ESRH)

C621x/C671x: Event Set Register (ESR)

31															16
Reserved															
RW,+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

C64x: Event Set Low Register (ESRL) for events 0 to 31

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ES31	ES30	ES29	ES28	ES27	ES26	ES25	ES24	ES23	ES22	ES21	ES20	ES19	ES18	ES17	ES16
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
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

C64x: Event Set High Register (ESRH) for events 32 to 61

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
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ES63	SC62	ES61	ES60	ES59	ES58	ES57	ES56	ES55	ES54	ES53	ES52	ES51	ES50	ES49	ES48
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
ES47	ES46	ES45	ES44	ES43	ES42	ES41	ES40	ES39	ES38	ES37	ES36	ES35	ES34	ES33	ES32
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

6.3.5 Event Polarity Register (EPRL, EPRH) (C64x)

An event is signaled to the EDMA controller by positive-edge triggering (low-to-high transition) on one of its event inputs. For the C64x the event polarity can be changed to falling-edge triggering (high-to-low transition) through setting the corresponding bits in the event polarity low register (EPRL) or event polarity high register (EPRH) to ‘1’. Figure 6–7 shows the EPRL and EPRH.

Figure 6–7. Event Polarity Register (EPRL, EPRH) (C64x)

C64x: Event Polarity Low Register (EPRL) for events 0 to 31

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EP31	EP30	EP29	EP28	EP27	EP26	EP25	EP24	EP23	EP22	EP21	EP20	EP19	EP18	EP17	EP16
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
EP15	EP14	EP13	EP12	EP11	EP10	EP9	EP8	EP7	EP6	EP5	EP4	EP3	EP2	EP1	EP0
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

C64x: Event Polarity High Register (EPRH) for events 31 to 64

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EP63	EP62	EP61	EP60	EP59	EP58	EP57	EP56	EP55	EP54	EP53	EP52	EP51	EP50	EP49	EP48
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
EP47	EP46	EP45	EP44	EP43	EP42	EP41	EP40	EP39	EP38	EP37	EP36	EP35	EP34	EP33	EP32
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 64 events (C64x) or 16 events (C621x/C671x) 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 transfer is determined by its EDMA parameters stored in the parameter RAM of the EDMA controller. Parameter RAM is discussed in the next section. For events arriving simultaneously, the channel with the highest event number submits its transfer request first.

6.5 Parameter RAM (PaRAM)

Unlike the C620x/C670x DMA controller, which is a register-based architecture, the EDMA controller is a RAM-based architecture. EDMA channels are configured in a parameter table. The table is a 2K-byte block of internal parameter RAM (PaRAM) located within the EDMA. The table consists of six-word parameter sets (entries), for a total of 85 entries.

The contents of the 2K byte PaRAM, shown in Table 6–3, comprises:

- For C621x/C671x there are 16 transfer parameter entries for the 16 EDMA events. For C64x. there are 64 transfer parameter entries for the 64 EDMA events. Each entry is six words or 24 bytes.
- Remaining transfer parameter sets are used for linking transfers. Each set or entry is 24 bytes.
- 8 bytes of unused RAM that can be used as scratch pad area. 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.

Once an event is captured, its parameters are read from one of the top 64 entries (C64x) or 16 entries (C621x/C671x) in the PaRAM. These parameters are then sent to the address generation hardware.

Table 6–3. EDMA Parameter RAM Contents

Address	Parameters
01A0 0000h to 01A0 0017h	Parameters for event 0 (6 words)
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)
01A0 0120h to 01A0 0137h	Parameters for event 12 (6 words)
01A0 0138h to 01A0 014Fh	Parameters for event 13 (6 words)
01A0 0150h to 01A0 0167h	Parameters for event 14 (6 words)
01A0 0168h to 01A0 017Fh	Parameters for event 15 (6 words)
01A0 0180h to 01A0 0197h	Parameters for event 16† (6 words)
01A0 0198h to 01A0 01AFh	Parameters for event 17† (6 words)
...	...
...	...
01A0 05D0h to 01A0 05E7h	Parameters for event 62† (6 words)
01A0 05E8h to 01A0 05FFh	Parameters for event 63† (6 words)
01A0 0600h to 01A0 0617h	Reload/link parameters for event N (6 words)
01A0 0618h to 01A0 062Fh	Reload/link parameters for event M (6 words)
…	…
01A0 07E0h to 01A0 07F7h	Reload parameters for event Z (6 words)
01A0 07F8h to 01A007FFh	Scratch pad area (2 words)

† The C64x devices support up to 64 synchronization events. For the C621x/C671x device, these PARAM locations (01A0 0180h – 01A0 05FFh) can be used for reload/link parameters.

6.5.1 EDMA Transfer Parameter Entry

Each parameter entry of an EDMA event is organized in six 32-bit words or 24 bytes as shown in Figure 6–8. Access to the EDMA parameter RAM is provided only via the peripheral bus. These parameters are shown in Table 6–4.

Figure 6–8. Parameter Storage for an EDMA Event

31 16	15 0	
Options (OPT)		Word 0
SRC Address (SRC)		Word 1
Array/frame count (FRMCNT)	Element count (ELECNT)	Word 2
DST address (DST)		Word 3
Array/frame index (FRMIDX)	Element index (ELEIDX)	Word 4
Element count reload (ELERLD)	Link address (LINK)	Word 5

Table 6–4. EDMA Channel Parameters

		As defined for…		
Offset Address (bytes)	**Parameter**	**1-D transfer**	**2-D transfer**	**Section**
0	Options	Transfer configuration options.		6.6.1
4	Source address	The address from which data is transferred.		6.6.2
8	Element count	The number of elements per frame.	The number of elements per array.	6.6.3
10	Frame count (1D), Array count (2D)	The number of frames per block minus one.	The number of arrays per frame minus one.	6.6.4
12	Destination address	The address to which data is transferred.		6.6.2
16	Element index	The address offset of elements within a frame.	—	6.6.5
18	Frame index (1D), Array index (2D)	The address offset of frames within a block.	The address offset of arrays within a frame.	6.6.5
20	Link address	The PaRAM address containing the parameter set to be linked.		6.6.7
22	Element count reload	The count value to be loaded at the end of each frame.†	—	6.6.6

† This field is only valid for element-synchronized transfers.

Note: The offset provided assumes little endian mode of operation. All control registers are 32 bits wide, and the physical location of parameters that share a single register are fixed, regardless of endian mode. Control registers should always be accessed as 32-bit words. The specific offset address entries that this note applies to are 8, 10, 16, 18, 20, and 22.

6.6 EDMA Transfer Parameters

Depending on the options associated with a transfer, the transfer parameters can be updated by the EDMA. The following sections describe the parameters shown in Table 6–5 that are common to the C621x/C671x/C64x devices, and those in Table 6–6 that are unique to the C64x.

6.6.1 Options Parameter (OPT)

The options parameter (OPT) in the EDMA channel entry is a 32-bit field as shown in Figure 6–9.

Figure 6–9. Options (OPT) Bit-Fields

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

15	14 13	12	11	10 5	4	3	2	1	0
rsvd	TCCM†	ATCINT†	rsvd	ATCC†	rsvd	PDTS†	PDTD†	LINK	FS
R,+0	RW,+00	RW,+0	R,+0	RW,+0	R,+0	RW,+0	RW,+0	RW,+0	RW,+0

† Applies to C64x only. On C621x/C671x, these bits are reserved, R+0.

Table 6–5. EDMA Channel Options Parameter (OPT) Description (C621x/C671x/C64x)

Bit No.	Field	Description	Section

31–29	PRI	Priority levels for EDMA events
		PRI=000b; Reserved; Urgent priority.
		For C621x/C671x, this level is reserved ONLY for L2 requests and not valid for EDMA transfer requests.
		For C64x, this level is available for CPU and EDMA transfer requests.
		PRI=001b; High priority EDMA transfer
		PRI=010b; Medium priority EDMA transfer (C64x);
		Low priority EDMA transfer (C621x/C671x).
		PRI=011b; Low priority EDMA transfer (C64x);
		reserved (C621x/C671x)
		PRI=100b to 111b; reserved
			6.17

28–27	ESIZE	Element size
		ESIZE=00b; 32-bit word
		ESIZE=01b; 16-bit half-word
		ESIZE=10b; 8-bit byte
		ESIZE=11b; reserved
			6.9

26	2DS	Source dimension 2DS = 0; 1-dimensional source. 2DS = 1; 2-dimensional source.	6.8 and 6.11
25–24	SUM	Source address update mode SUM = 00b; Fixed address mode. No source address modification SUM = 01b; Source address increment depends on 2DS, and FS bit-fields SUM = 10b; Source address decrement depends on 2DS, and FS bit-fields SUM = 11b; Source address modified by the element index/frame index depending on 2DS, and FS bits.	6.11

Table 6–5. EDMA Channel Options Parameter (OPT) Description (C621x/C671x/C64x) (Continued)

Bit No.	Field	Description	Section
23	2DD	Destination dimension 2DD = 0; 1-dimension destination. 2DD = 1; 2-dimensional destination.	6.8 and 6.11
22–21	DUM	Destination address update mode DUM = 00b; Fixed address mode. No destination address modification DUM = 01b; Destination increment depends on 2DD, and FS bit-fields DUM = 10b; Destination decrement depends on 2DD, and FS bit-fields DUM = 11b; Destination modified by the element index/frame index depending on 2DD, and FS bits.	6.11
20	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.	6.14 and 6.15
19–16	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, when the current set is exhausted. For C64x, the 4-bit TCC code is used in conjunction with bit field TCCM for a 6-bit transfer complete code.	6.14 and 6.15
1	LINK	Link LINK=0; Linking of event parameters disabled. Entry not reloaded. LINK=1; Linking of event parameters enabled. After the current set is exhausted, the channel entry is reloaded with the parameter set specified by the link address. The link address must be on a 24-byte boundary and within the EDMA PaRAM. The link address is a 16-bit address offset from the PaRAM base address.	6.6.7 and 6.12
0	FS	Frame synchronization FS=0; Channel is element/array synchronized. FS=1; Channel is frame synchronized. The relevant event for a given EDMA channel is used to synchronize a frame.	6.7

Table 6–6. EDMA Channel Options Field Description (C64x only)

Bit No.	Field	Description	Section
14–13	TCCM	Most significant bits of transfer complete code. (C64x only) For C64x, bits TCCM work in conjunction with the 4-bit TCC code to provide a 6-bit transfer complete code. TCCM:TCC = 000000b to 111111b; 6-bit code is used to set the relevant bit in CIPRL or CIPRH provided TCINT = 1, when the current set is exhausted.	6.14 6.15
12	ATCINT	Alternate transfer complete interrupt (C64x only) ATCINT = 0; Alternate transfer complete indication disabled. CIPR bits are not set upon completion of intermediate transfers in a block. ATCINT = 1; The relevant CIPR bit is set upon completion of intermediate transfers in a block. The bit (position) set in the CIPR is the ATCC value specified.	6.14 6.15
10 – 5	ATCC	Alternate transfer complete code (C64x only) ATCC = 000000b to 111111b; 6-bit code is used to set the relevant bit in CIPRL or CIPRH (i.e., CIP[ATCC] bit) provided ATCINT = 1, upon completion of an intermediate transfer in a block.	6.14 6.15

3	PDTS	Peripheral device transfer (PDT) mode for source (C64x only).
		PDTS = 0; PDT read disabled.
		PDTS = 1; PDT read enabled.
			6.16

2	PDTD	Peripheral device transfer (PDT) mode for destination (C64xonly).
		PDTD = 0; PDT write disabled.
		PDTD = 1; PDT write enabled.
			6.16

6.6.2 SRC/DST Address (SRC/DST)

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 6.11.

6.6.3 Element Count (ELECNT)

Element count is a 16-bit unsigned value that specifies the number of elements in a frame (for 1D transfers) 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. The EDMA performs no transfers if element count is zero. Details in section 6.10.

6.6.4 Frame/Array Count (FRMCNT)

Frame/array count is also a 16-bit unsigned value and it specifies the number of frames in a 1D block or number of arrays in a 2D block. The term ’frame count’ applies to 1D transfers, while the term ’array count’ applies to 2D transfers. The maximum number of frames/arrays in a block is 65536. Therefore a frame/array count of 0 is actually one frame/array and frame/array count of 1 corresponds to 2 frames/arrays. Details in section 6.10.

6.6.5 Element Index (ELEIDX) and Frame/Array Index (FRMIDX)

The 16-bit signed value specified in the element and frame/array 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 term ’frame index’ applies to 1D transfers, while the term ’array index’ applies to 2D transfers. 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 1D 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/array index provides an offset to the next frame/array in a block.

6.6.6 Element Count Reload (ELERLD)

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 1D element 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 6.10.1.

6.6.7 Link Address (LINK)

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 (C621x/C671x), or 01A0 0600h to 01A0 07F7h (C64x). 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 6.12. In addition to the reload parameter space, the entry of any unused EDMA channel can also be used for linking. The EDMA can always have up to 85 programmed entries regardless of the number of channels actually used.

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. This allows the submission of transfer requests to occur automatically based on system events, without any intervention by the CPU. CPU-initiated transfer is included in the design for added control and robustness. QDMA transfers, discussed in section 6.19, are the preferred method of issuing CPU-synchronized data transfers.

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. In addition to event enable via EER, the completion of a transfer can also trigger another EDMA transfer through chaining and the CCER. See section 6.15 for details.

6.7.1 Synchronization of EDMA Transfers

All EDMA channels are tied to a specific synchronization event. Synchronization allows EDMA transfers to be triggered by events either from peripherals, interrupts from external hardware, or an EDMA transfer completion event. A channel only requests a data transfer when it receives its event or when the CPU manually synchronizes it (by writing to the ESR). The amount of data to be transferred depends on the channel’s configuration. A channel can submit an entire frame/block when frame/block-synchronized, or a subset of a frame (element or array, depending on dimension) when element/array-synchronized.

Table 6–7 and Table 6–8 list the synchronization events associated with each of the programmable EDMA channels for the C621x/C671x and C64x, respectively.

The association of an event to a channel is fixed. Unlike the existing C6201-type DMA, each of EDMA channels have one specific 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 (EXT_INTn). The source of C621x/C671x synchronization events is listed in Table 6–7. The source of C64x synchronization events, listed in Table 6–8, is a superset of the C621x/C671x events. 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 6–7. EDMA Channel Synchronization Events – TMS320621x/C671x

EDMA Channel Number	Event Acronym	Event Description
0	DSPINT	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

† Channels 8 to 11 are used for transfer chaining only. See section 6.15 for details.

Table 6–8. EDMA Channel Synchronization Events – TMS320C64x [†]

EDMA Channel Number	Event Acronym	Event Description
0	DSPINT	Host-to-DSP interrupt
1	TINT0	Timer 0 interrupt
2	TINT1	Timer 1 interrupt
3	SD_INTA	EMIFA SDRAM timer interrupt
4	GPINT4/EXT_INT4	GPIO event 4/External interrupt 4
5	GPINT5/EXT_INT5	GPIO event 5/External interrupt 5
6	GPINT6/EXT_INT6	GPIO event 6/External interrupt 6
7	GPINT7/EXT_INT7	GPIO event 7/External interrupt 7
8	GPINT0	GPIO event 0
9	GPINT1	GPIO event 1
10	GPINT2	GPIO event 2
11	GPINT3	GPIO event 3
12	XEVT0	McBSP0 transmit event
13	REVT0	McBSP0 receive event
14	XEVT1	McBSP1 transmit event
15	REVT1	McBSP1 receive event
16	–	None
17	XEVT2	McBSP2 transmit event
18	REVT2	MCBSP2 receive event
19	TINT2	Timer 2 interrupt
20	SD_INTB	EMIFB SDRAM timer interrupt
21	PCI	PCI Wakeup Interrupt
22	–	None
23	–	None
24	–	None
25	–	None
26	–	None
27	–	None
28	–	None
29	–	None
30	–	None

[†] In addition to the events shown in Table 6–8, each of the 64 channels can also be synchronized with the transfer completion or alternate transfer completion events. See section 6.15, *Chaining EDMA Channels by an Event*.

Table 6–8. EDMA Channel Synchronization Events – TMS320C64x(Continued)[†]

EDMA Channel Number	Event Acronym	Event Description
31	–	None
32	UREVT	Utopia Receive Event
33	–	None
34	–	None
35	–	None
36	–	None
37	–	None
38	–	None
39	–	None
40	UXEVT	UTOPIA transmit event
41	–	None
42	–	None
43	–	None
44	–	None
45	–	None
46	–	None
47	–	None
48	GPINT8	GPIO event 8
49	GPINT9	GPIO event 9
50	GPINT10	GPIO event 10
51	GPINT11	GPIO event 11
52	GPINT12	GPIO event 12
53	GPINT12	GPIO event 13
54	GPINT14	GPIO event 14
55	GPINT15	GPIO event 15
56	–	None
57	–	None
58	–	None
59	–	None
60	–	None
61	–	None
62	–	None
63	–	None

[†] In addition to the events shown in Table 6–8, each of the 64 channels can also be synchronized with the transfer completion or alternate transfer completion events. See section 6.15, *Chaining EDMA Channels by an Event*.

6.8 Types of EDMA Transfers

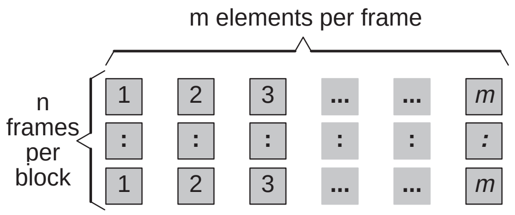
The EDMA provides for two types of data transfers, 1-dimensional (1D) and 2-dimensional (2D) transfers. This is selected by setting the 2DD and 2DS bits in the event's options field. 2DD when set to 1 represents two-dimensional transfer on the destination. Similarly, a 2-D transfer on the source is performed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

The number of dimensions a transfer has determines the makeup of a frame of data. In a 1-D transfer, frames are made up of a number of individual elements. In a 2-D transfer, blocks are made up of a number of arrays, each of which is made up of a number of elements. See Appendix A for figures that provide a detailed representation of all types of EDMA transfers.

6.8.1 1-Dimensional Transfers

For 1D transfers, a group of elements equal to element count constitute a frame. Transfers focus on individual elements. Each frame of data to be transferred has a single dimension associated with it indicating the number of elements per frame. EDMA channels may be configured to transfer multiple frames (or a block of frames), but each frame is handled individually. Frame count is the number of frames in a 1D transfer. A 1-D transfer can be considered two dimensional, with the second dimension fixed at 1. A sample 1-D frame is shown in Figure 6–10, with an element count of *m*.

Figure 6–10. 1-D Transfer Data Frame



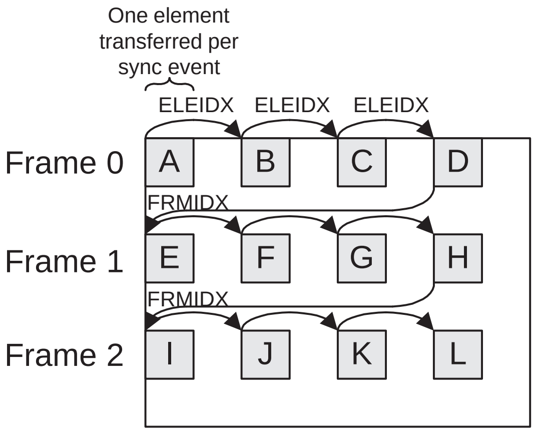
The elements within a block can either be all located at the same address, at contiguous addresses, or at a configurable offset from one another. The addresses of elements within a frame can be located at a specific distance apart (determined by element index, ELEIDX), while address of the first element of each frame is a set distance from a particular element of the previous frame (determined by frame index, FRMIDX). 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 (ELERLD) in the transfer entry.

Transfers may be submitted either one element at a time, when element synchronized(FS = 0), or one frame at a time, when frame-synchronized (FS = 1).

6.8.1.1 Element Synchronized 1D Transfer (FS=0)

If a channel is configured to be a 1-D element synchronized transfer, the source and destination addresses are updated within the parameter table following the transfer request submission for each element. Therefore the element index (ELEIDX) and frame index (FRMIDX) are based on the difference between element addresses. Figure 6–11 shows the concept of a 1D element synchronized transfer with 4 elements in each frame (ELECNT = 4) and a total of three frames (FRMCNT = 2).

Figure 6–11. Transfer with Element Synchronication (FS=0)



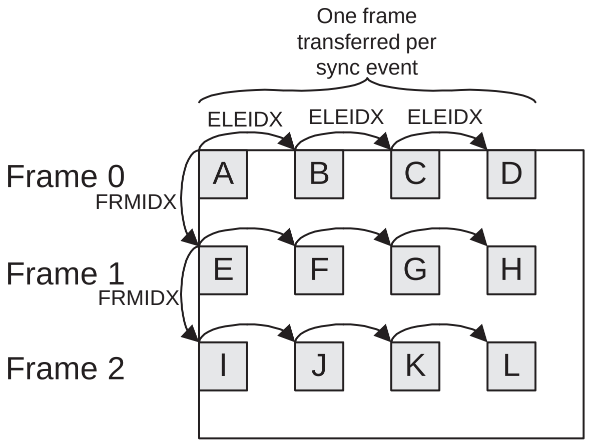
Each element in a frame is transferred from its source to destination address upon receiving the channel-specific sync event. After the channel receives a sync event, it sends off a transfer request for DMA service. The EDMA controller then decrements the element count (ELECNT) by 1 in the parameter RAM. When a channel sync event occurs and ELECNT = 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 (reload with the 16-bit value in ELERLD) and frame count (FRMCNT) decrements by 1. User-specified element index (ELEIDX) is used to compute the address of the next element in a frame. Similarly, frame index (FRMIDX) 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 1D transfer. Specific updates are described in sections 6.10 and 6.11.

If linking is enabled (LINK=1, see section 6.12), 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.

6.8.1.2 Frame Synchronized 1D Transfer (FS=1)

Frame-synchronized 1D transfer allows a channel to request the transfer of an entire frame of elements. The frame index no longer represents the difference between the address of the last element of a frame and the address of the first element of the subsequent frame, but rather the difference between the starting addresses of each frame. A frame-synchronized 1-D transfer is functionally identical to an array-synchronized 2-D transfer (assuming ELEIDX equals the number of bytes per element). The address indexing for a frame-synchronized 1-D transfer is shown in Figure 6–12.

Figure 6–12. 1D Transfer With Frame Synchronization (FS=1)



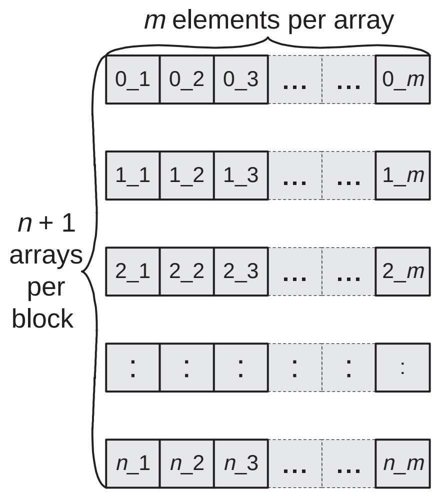
Here, the element transfer in each frame is not synchronized, but instead each frame transfer is synchronized by the channel event. The FS bit (in options field) should be set to ‘1’ to enable frame-synchronized transfer. A user-specified element index (ELEIDX) can be used to stagger elements in a frame. Frame index (FRMIDX) can be added to the start element address in a frame to derive the next frame start address. Element count reload (ELERLD) does not apply to a 1D frame-synchronized transfer (FS = 1). The address modification and count modification depends on the type of update modes chosen. They are mentioned here only for an understanding of a 1D transfer. Specific updates are described in sections 6.10 and 6.11.

If linking is enabled (LINK = 1, see section 6.12), 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.

6.8.2 2-Dimensional Transfers

The 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, hence element index (ELEIDX) 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. Arrays can be offset from one another by a fixed amount. A sample 2-D frame is shown in Figure 6–13, with an array count of *n* and an element count of *m*.

Figure 6–13. 2-D Transfer Data Block

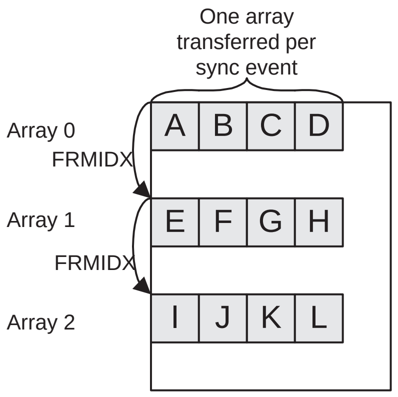


The offset of the arrays is determined by the array index (FRMIDX), the value of which depends on the synchronization mode of the transfer. Transfers can be submitted either one array at a time, when array synchronized (FS = 0), or one block at a time, when block synchronized (FS = 1).

6.8.2.1 Array Synchronized 2D Transfer (FS = 0)

A channel that is configured to perform a 2-D transfer with array synchronization updates its source and destination registers after the transfer request for each *array* is submitted. The array index (FRMIDX) is the difference between the starting addresses for each array of the block, as shown in Figure 6–14. FRMIDX is used for all address update modes except fixed address update mode (SUM/DUM = 00b).

Figure 6–14. 2-D Transfer with Array Synchronization (FS = 0)



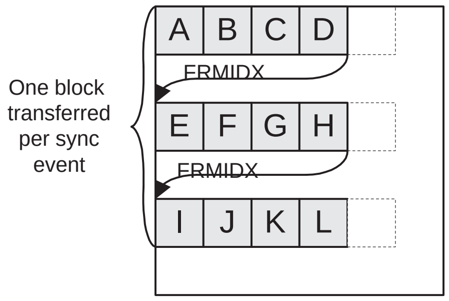
Upon receiving a synchronization event, an array (contiguous group of elements) is transferred. The example shows 4 elements in an array (ELECNT = 4) and the number of arrays to be transferred is 3 (FRMCNT = 2). Frame count (FRMCNT) decrements after each array is transferred. Frame index is added to an array’s start address to derive the next array’s start address. The actual address modification and count modification depends upon the type of update modes selected (SUM/DUM). These modes are mentioned here only for an understanding of a 2D transfer. Specific updates are described in section 6.11 and section 6.12.

When FRMCNT reaches zero and if linking is enabled (LINK = 1, see section 6.12), 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.

6.8.2.2 Block Synchronized 2D Transfer (FS=1)

For a 2-D transfer, the complete block gets transferred when the channel's event occurs and FS = 1. Block synchronization causes the array index (FRMIDX) to be implemented by the address generation/transfer logic. This address update is transparent to the user and is not reflected in the parameter RAM. The address is updated after each element in a burst. The logic first updates the addresses according to the setting of SUM/DUM. If an element is the last in a particular array and an update mode is selected (SUM/DUM ≠ 00b), the address(es) are indexed according to the array index. The index is added to the address after the address update occurs. FRMIDX is therefore equal to the space between arrays of a block, as shown in Figure 6–15.

Figure 6–15. 2-D Transfer with Block Synchronization (FS=1)



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 block sync arrives. See section 6.12 for details on linking.

6.9 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.

When transferring a burst of elements to or from a 64 bit wide peripheral (e.g. L2 or EMIFA), 64-bit elements are transferred regardless of the ESIZE programmed. This allows the EDMA to maximize the available bandwidth.

6.9.1 Fixed Address Mode Transfer Considerations

The maximum EDMA element size is 32-bit. However, the following data paths are 64-bit wide:

- ❑ L2 SRAM
- ❑ EMIFA (64-bit EMIF, C64x only)

Care must be taken when performing a burst access via the 64-bit data paths that have the following EDMA configurations:

- ❑ Element size is 32-bit (ESIZE = 00b)
- ❑ Fixed address mode (SUM or DUM = 00b in the options parameter)
- ❑ Frame-synchronized access (FS = 1 in the options parameter), or two-dimensional source or destination transfer (either 2DS or 2DD is set to 1).
- ❑ Element count is greater than 1 (ELECNT > 1).

Accesses to a 64-bit-wide data bus with the above EDMA configurations are fixed on a 64-bit boundary. For example, when performing an N number of 32-bit accesses to the L2 SRAM or EMIFA in fixed address mode (ELECNT = N, N>1), the EDMA actually performs N/2 number of 64-bit accesses to the fixed doubleword address. Thus it is actually a 64-bit doubleword that is transferred.

For a write to a 64-bit-wide data bus with the above conditions, the 32-bit word is written to both word 0 and word 1 of the fixed doubleword address. For example, a 32-bit write to the L2 SRAM address 0x00000000 updates both word 0 (at address 0x00000000) and word 1 (at address 0x00000004) with the new data.

For a read from a 64-bit-wide data bus with the above conditions, both word 0 and word 1 of the fixed doubleword address are extracted. For example, when performing an EDMA word transfer from the L2 SRAM fixed address 0x00000000 to an external memory via a 32-bit EMIF, the extracted data from the L2 SRAM will show up at the EMIF pins ED[31:0] as “word 0”, “word 1”, “word 0”, “word 1”…etc.

The above considerations only apply to accesses to a 64-bit-wide data bus. For EDMA fixed address mode word access to a 32-bit internal register or a 32-/16-/8-bit external memory device, the address is fixed on a 32-bit word boundary. Reads and writes are only performed to the word address specified.

6.10 Element and Frame/Array Count Updates

The EDMA parameter RAM has 16-bit unsigned values of element count (ELECNT) and frame/array count (FRMCNT) each. Additionally, it also holds 16-bit signed values each for the element index (ELEIDX) and frame/array index (FRMIDX). 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 ELECNT and FRMCNT are updated in the corresponding event's transfer entry depending on the type of transfer (1D or 2D) and the synchronization type as shown in Table 6–9.

Table 6–9. EDMA Element and Frame/Array Count Updates

Synchronization	Transfer Mode	Element Count Update	Frame/Array Count Update†
Element (FS=0)	1D; (2DS&2DD=0)	–1 (reload if ELECNT = 1) See section 6.10.1	–1 (if element count = 1)
Array (FS=0)	2D; (2DS|2DD=1)	None	–1
Frame (FS=1)	1D; (2DS&2DD=0)	None	–1
Block (FS=1)	2D; (2DS|2DD=1)	None	None

† Frame count update applies to 1D transfers. Array count update applies to 2D transfers. No frame/array count update occurs if the frame/array count is zero (FRMCNT = 0).

6.10.1 Element Count Reload (ELERLD)

There is a special condition for reloading the element count for element synchronized (FS = 0) 1D 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 6–11. Therefore, the EDMA controller keeps track of the element count to update the address. When an element sync event occurs at the end of a frame (ELECNT = 1), the EDMA controller sends off the transfer request, and reloads the ELECNT 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 (transparent to users) tracks the address directly.

6.11 Source/Destination (SRS/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 6–10 provide for a variety of data structures that can be created. The source and/or destination address is updated depending on whether frame/block sync (FS) is enabled, or dimension (2DS/2DD) of the transfer. All address updates should occur after the current transfer request is sent. Therefore, these updates are used to set the EDMA parameters for the next event. Table 6–12 shows the possible address update modes for a transfer.

Table 6–10. Address Update Modes

SUM/DUM	1-D	2-D
00: No mod	All elements located at the same address.	All elements in an array are at the same address.
01: Increment	All elements are contiguous, with subsequent elements located at a higher address than the previous.	All elements within an array are contiguous, with subsequent elements located at a higher address than the previous. Arrays are offset by FRMIDX.
10: Decrement	All elements are contiguous, with subsequent elements located at a lower address than the previous.	All elements within an array are contiguous, with subsequent elements located at a lower address than the previous. Arrays are offset by FRMIDX.
11: Index	All elements within a frame are offset from one another by ELEIDX. Frames are offset by FRMIDX.	Reserved

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 1D 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 6–11 shows the amount by which the source address is modified for each of the combinations of FS, 2DD/2DS, and SUM parameters. Table 6–12 shows the destination address updates that are possible. Appendix A provides figure representations of all types of transfers.

Note that when either the source or the destination is a 2D transfer *and* the transfer is block synchronized (FS=1), it means that the complete block of data is transferred on a sync event. Therefore, address updates are not applicable in this case because updates are transparent to the users. If LINK = 1 and the link conditions outlined in Table 6–13 are met, no address updates occur. Instead, the link parameters are copied directly to the event parameter.

Table 6–11. EDMA SRC Address Parameter Updates

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

Note: ELECNT: Element count
ELEIDX: 16-bit signed element index value
FRMCNT: Frame/array count
FRMIDX: 16-bit signed frame index value(1D transfers)
FRMIDX: 16-bit signed array index value (2D transfers)
ESIZE:element size in bytes

Table 6–12. EDMA DST Address Parameter Updates

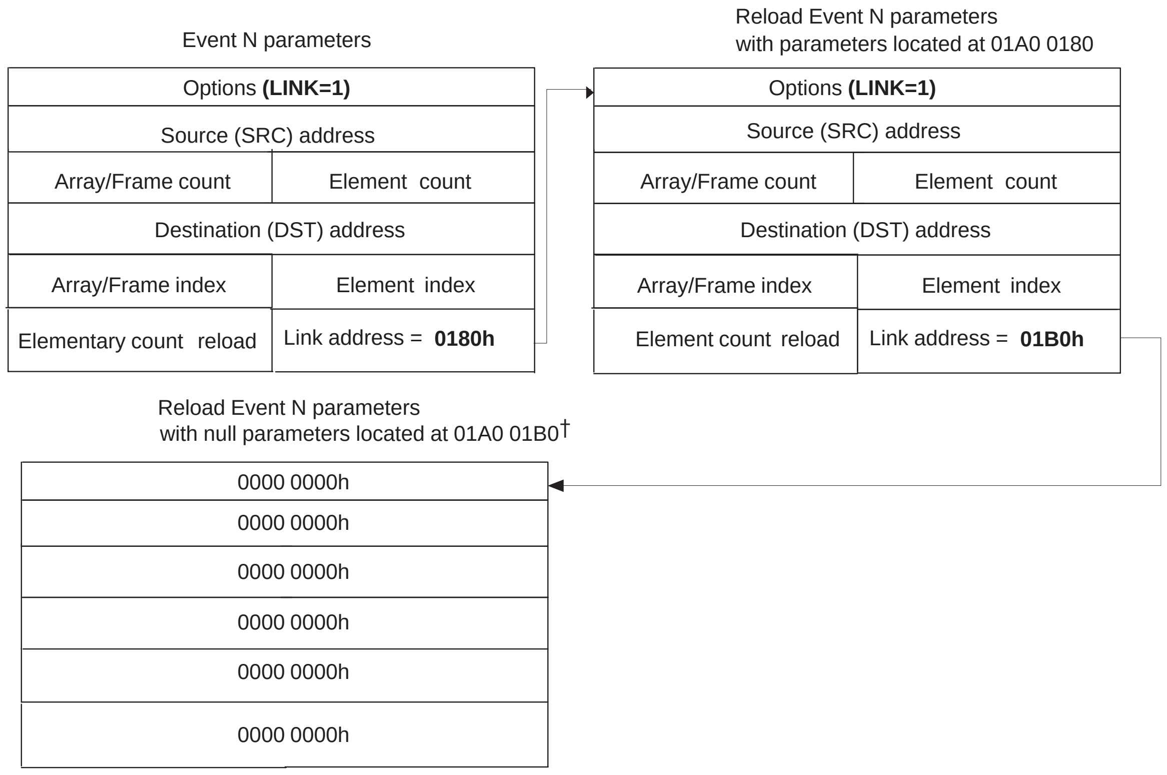
Frame Sync	Transfer Type (2DS:2DD)	Destination Update Mode (DUM)			
		00	01	10	11
FS = 0	00	None	+ESIZE; Increment by element size.	–ESIZE; Decrement by element size.	+ELEIDX or +FRMIDX if ELECNT = 1 Add signed ELEIDX to each element in a frame except the last. Add signed FRMIDX to the last element in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

Note: ELECNT: Element count
ELEIDX: 16-bit signed element index value
FRMCNT: Frame/array count
FRMIDX: 16-bit signed frame index value (1D transfers)
FRMIDX: 16-bit signed array index value (2D transfers)
ESIZE:element size in bytes

6.12 Linking EDMA Transfers

The EDMA controller provides linking, a feature especially useful for complex sorting, circular buffering type of applications. If LINK = 1, upon completion of a transfer, the EDMA link feature reloads the current transfer parameters with the parameter pointed to by the 16-bit link address. 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 6–16.

Figure 6–16. Linked EDMA Transfer



† See section 6.13 for details on null parameters

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 6–13 shows the channel completion conditions when the linking of parameters is performed. 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. The last entry should be linked to a NULL parameter set. See section 6.13 for details.

Table 6–13. Channel Completion Conditions

LINK = 1	**1D Transfers**	**2D Transfers**
Element/array sync (FS = 0)	Frame count == 0 && Element count == 1	Frame count == 0
Frame sync (FS = 1)	Frame count == 0	Always

Linking an entry to itself replicates the behavior of autoinitialization to facilitate the use of circular buffering and repetitive transfers. After an EDMA channel exhausts its current entry, the parameter set is reloaded and the transfer begins again.

Once the channel completion conditions are met for an event, the transfer parameters located at the link address are loaded into one of the 16 event parameter space (C621x/C671x) or 64 event parameter space (C64x) 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.

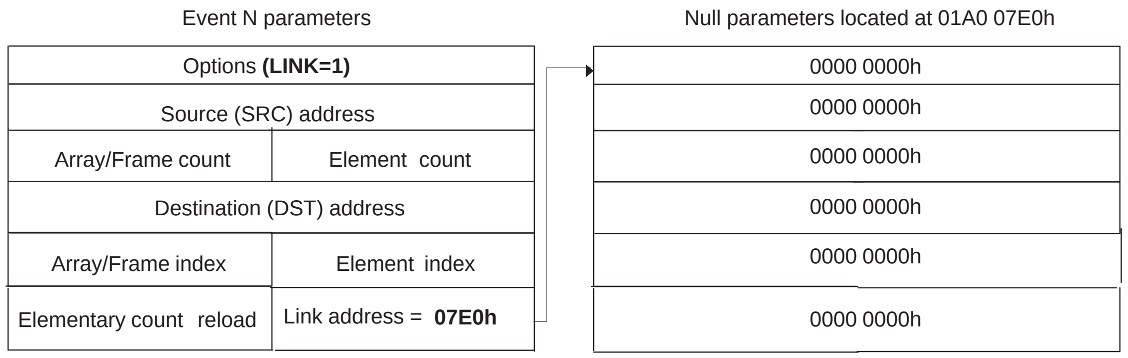
Any entry in the PaRAM can be used for a linked transfer parameter set. Entries in the first 16 (C621x/C671x) or 64 (C64x) locations should only be used for linking if the corresponding event and chain event are disabled.

6.13 Terminating an EDMA Transfer

All EDMA transfers are terminated by linking to a NULL parameter set after the last transfer.

The NULL parameter set serves as the termination point of any EDMA transfer. A NULL parameter set is defined as an EDMA parameter set where all the parameters (options, source/destination address, frame/element count, etc.) are set to zero. Multiple EDMA transfers can link to the same terminating NULL parameter set. Therefore only one NULL parameter set is required in the EDMA parameter RAM. Figure 6–17 is an example of an EDMA transfer termination.

Figure 6–17. Terminating EDMA Transfers



6.14 EDMA Interrupt Generation

The EDMA controller is responsible for generating transfer-completion interrupts to the CPU. Unlike the C620x/C670x 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 (C621x/C671x) or 64 channels (C64x). 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 is provided, the EDMA controller sets a bit in the channel interrupt pending register (CIPR) shown in Figure 6–18. The C64x has two channel interrupt pending registers, channel interrupt pending low register (CIPRL) and channel interrupt pending high register (CIPRH), for the 64 channels. For the remaining of this chapter, the term “channel interrupt pending register” or “CIPR” refers to the CIPR for C621x/C671x, or the CIPRL/CIPRH for C64x.

The CIPR bit number that gets set is dictated by the transfer complete code 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 6–19. Since the C64x has 64 channels, it has two channel interrupt enable registers, channel interrupt enable low register (CIERL) and channel interrupt enable high register (CIERH). For the remaining of this chapter, the term “channel interrupt enable register” or “CIER” refers to the CIER for C621x/C671x, or the CIERL/CIERH for C64x.

To configure the EDMA for any channel (or QDMA request) to interrupt the CPU:

❑ Set CIE*n* to ‘1’ in the CIER
❑ Set TCINT to ‘1’ in channel options
❑ Set Transfer Complete Code to *n* in channel options

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 6–18. Channel Interrupt Pending Register (CIPR, CIPRL, CIPRH)

C621x/C671x: 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

C640x: Channel Interrupt Pending Low Register (CIPRL)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CIP31	CIP30	CIP29	CIP28	CIP27	CIP26	CIP25	CIP24	CIP23	CIP22	CIP21	CIP20	CIP19	CIP18	CIP17	CIP16
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
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

C640x: Channel Interrupt Pending High Register (CIPRH)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CIP63	CIP62	CIP61	CIP60	CIP59	CIP58	CIP57	CIP56	CIP55	CIP54	CIP53	CIP52	CIP51	CIP50	CIP49	CIP48
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
CIP47	CIP46	CIP45	CIP44	CIP43	CIP42	CIP41	CIP40	CIP39	CIP38	CIP37	CIP36	CIP35	CIP34	CIP33	CIP32
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 6–19. Channel Interrupt Enable Register (CIER, CIERL, CIERH)

C621x/C671x: 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

C64x: Channel Interrupt Enable Low Register (CIERL)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CIE31	CIE30	CIE29	CIE28	CIE27	CIE26	CIE25	CIE24	CIE23	CIE22	CIE21	CIE20	CIE19	CIE18	CIE17	CIE16
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
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

C64x: Channel Interrupt Enable High Register (CIERH)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CIE63	CIE62	CIE61	CIE60	CIE59	CIE58	CIE57	CIE56	CIE55	CIE54	CIE53	CIE52	CIE51	CIE50	CIE49	CIE48
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
CIE47	CIE46	CIE45	CIE44	CIE43	CIE42	CIE41	CIE40	CIE39	CIE38	CIE37	CIE36	CIE35	CI34	CIE33	CIE32
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

In the C621x/C671x, the transfer complete code is specified in the TCC field, with values between 0000b to 1111b. In the C64x, which has a total of 64 channels, the transfer complete code is expanded to a 6-bit value that accomodates the 64 channels. The 6-bit transfer complete code of the C64x is comprised of the new TCCM bits (most significant bits of the transfer complete code), in addition to the TCC field in the options parameter. The transfer complete code is directly mapped to the CIPR bits as shown in Table 6–14 for the C621x/C671x and Table 6–15 for the C64x. For example, if TCC = 1100b (and also TCCM = 00 for the C64x), CIPR[12] (C621x/C671x) or CIPRL[12] (C64x) 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 transfer complete code to be any value in Table 6–14 for any EDMA channel. In other words, there need not necessarily be a direct relation between the channel number and the transfer complete code value. This allows multiple channels having the same transfer complete code value to cause the CPU to execute the same ISR (for different channels). Alternatively, the same channel can set multiple complete codes depending on the transfers performed.

Table 6–14. Transfer Complete Code (TCC) to EDMA Interrupt Mapping

TCC in Options (TCINT=1)	CIPR Bits Set	TCC in Options (TCINT=1)	CIPR Bits Set
0000b	CIP0	1000b	CIP8
0001b	CIP1	0001b	CIP9
0010b	CIP2	0010b	CIP10
0011b	CIP3	0011b	CIP11
0100b	CIP4	0110b	CIP12
0101b	CIP5	1101b	CIP13
0110b	CIP6	1110b	CIP14
0111b	CIP7	1111b	CIP15

Table 6–15. C64x Transfer Complete Code (TCC) to EDMA Interrupt Mapping

TCC in Options (TCINT=1)	CIPRL Bits Set†	TCC in Options (TCINT=1)	CIPRH Bits Set†
000000b	CIP0	100000b	CIP32
000001b	CIP1	100001b	CIP33
000010b	CIP2	100010b	CIP34
000011b	CIP3	100011b	CIP35
000100b	CIP4	100110b	CIP36
...	...	...	...
...	...	...	...
...	...	...	...
011110b	CIP30	111110b	CIP62
011111b	CIP31	111111b	CIP63

† Bit fields CIP[0:31] correspond to bits 0 to 31 in CIPRL.
Bit fields CIP[32:63] correspond to bits 0 to 31 in CIPRH.

6.14.1 EDMA Interrupt Servicing by the CPU

Since the EDMA controller is aware of when the EDMA channel transfer is complete, it sets the appropriate bit in the CIPR as per the transfer complete code 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. The ISR should check for all pending interrupts and continue until all the posted interupts are serviced.

6.14.2 TMS320C64x Alternate Transfer Complete Code Interrupt

In addition to the transfer complete interrupt, the C64x EDMA allows channel interrupt upon completion of intermediate transfers in a block. This is referred to as the alternate transfer complete interrupt. For example, in a 1D element-synchronized transfer, alternate transfer complete interrupt may be generated upon transfer completion of each element.

Two new fields, the alternate transfer complete interrupt (ATCINT) and the alternate transfer complete code (ATCC), are added to the options parameters. The function of the alternate transfer interrupt is similar to the function of the transfer complete interrupt. Similar to the TCCM:TCC, the ATCC can be set to any values betwen 000000b to 111111b (see Table 6–14).

To enable alternate transfer complete interrupt, configure the EDMA channel options parameter as follows:

- ❑ Set CIEn to ’1’ in the CIER
- ❑ Set ATCINT to ’1’ in channel options
- ❑ Set ATCC to ’n’ in channel options.

When alternate transfer complete interrupt is enabled by ATCINT, an interrupt is set (and sent to the CPU if CIER is set) upon completion of each intermediate transfer of the current channel. Upon completion of the entire channel transfer (See channel completion conditions in Table 6–13), the transfer complete interrupt applies instead, provided transfer complete interrupt is enabled by TCINT. Alternate transfer complete interrupt does not apply to 2-dimensional frame-synchronized transfers, since there are no intermediate transfers in this mode.

6.15 Chaining EDMA Channels by an Event

The channel chaining capability for the EDMA allows the completion of an EDMA channel transfer to trigger another EDMA channel transfer. Table 6–13 shows the channel complete conditions.

Chaining is different from linking. The EDMA link feature (section 6.12) reloads the current channel parameter with the linked parameter. The EDMA chaining feature does not modify or update any channel parameters. It simply provides a synchronization event to the chained channel. The channel chain enable register (CCER) is shown in Figure 6–20.

Figure 6–20. Channel Chain Enable Register (CCER, CCERL, CCERH)

C621x/C671x: Channel Chain Enable Register (CCER)

31–16
Reserved
R, +0

15–12	11	10	9	8	7–0
Reserved	CCE11	CCE10	CCE9	CCE8	Reserved
R,+0	RW,+0	RW,+0	RW,+0	RW,+0	R,+0

C64x: Channel Chain Enable Low Register (CCERL) for Events 0 to 31

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CCE31	CCE30	CCE29	CCE28	CCE27	CCE26	CCE25	CCE24	CCE23	CCE22	CCE21	CCE20	CCE19	CCE18	CCE17	CCE16
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
CCE15	CCE14	CCE13	CCE12	CCE11	CCE10	CCE9	CCE8	CCE7	CCE6	CCE5	CCE4	CCE3	CCE2	CCE1	CCE0
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

C64x Channel Chain Enable High Register (CCERH) for Events 32 to 63

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
CCE63	CCE62	CCE61	CCE60	CCE59	CCE58	CCE57	CCE56	CCE55	CCE54	CCE53	CCE52	CCE51	CCE50	CCE49	CCE48
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
CCE47	CCE46	CCE45	CCE44	CCE43	CCE42	CCE41	CCE40	CCE39	CCE38	CCE37	CCE36	CCE35	CCE34	CCE33	CCE32
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

6.15.1 TMS320C621x/C671x EDMA Transfer Chaining

For the C621x/C671x, 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. By setting the TCC to 8, 9, 10, or 11, any EDMA channel can synchronize any of these four channels.

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 6–20 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.

For example, if CCER[8] = 1 and TCC = 1000b are specified for EDMA channel 4, an external interrupt on EXT_INT4 initiates the EDMA transfer. Once channel 4 transfer is complete (all of the parameters are exhausted), 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’.

6.15.2 TMS320C64x EDMA Transfer Chaining

The C64x EDMA transfer chaining is an expansion of the C621x/C671x transfer chaining. Any of the 64 transfer completion codes of the C64x EDMA can be used to trigger another channel transfer. The user-specified transfer complete code is expanded to a 6-bit value TCCM:TCC. The 4 bits in the TCC field (bits 19 to 16) of the options parameter are the least significant bits of the transfer complete code, while the new TCCM bit fields are the most significant bits of the transfer complete code. For example, if the transfer complete code (TCCM:TCC) is 010001b (i.e. TCCM = 01, TCC = 0001b) and CCERL[17] = 1 is specified for EDMA channel 4, the completion of the channel 4 transfer will initiate the next transfer specified by EDMA channel 17, provided that the channel 4 TCINT = 1.

6.15.3 TMS320C64x Alternate Transfer Chaining

The alternate transfer complete interrupt field (ATCINT) and alternate transfer complete code (ATCC), described in section 6.14.2, allow the C64x EDMA to perform channel chaining upon completion of intermediate transfers in a block. The function of the alternate transfer chaining is similar to the function of the transfer complete chaining.

When alternate transfer complete chaining is enabled, the next EDMA channel (specified by the ATCC of the current channel) is synchronized upon completion of each intermediate transfer of the current channel. Upon completion of the entire channel transfer (see channel completion conditions in Table 6–13), transfer complete chaining applies instead, provided transfer complete chaining is enabled. Alternate transfer complete chaining does not apply to 2-dimensional block-synchronized transfers, since there are no intermediate transfers in this mode. Alternate transfer chaining allows one channel to trigger another once for each transfer request it makes (i.e. once per sync event received), rather than only once per block.

To enable alternate transfer complete chaining, configure the EDMA channel parameter as follows:

❑ Set ATCINT = 1 in the options parameter

❑ Set the ATCC value to the next EDMA channel in the chain

❑ Set the relevant bit in the channel chain enable register (bit CCER[ATCC]).

Note: Alternate transfer complete code chaining does not affect linking operations described in section 6.12.

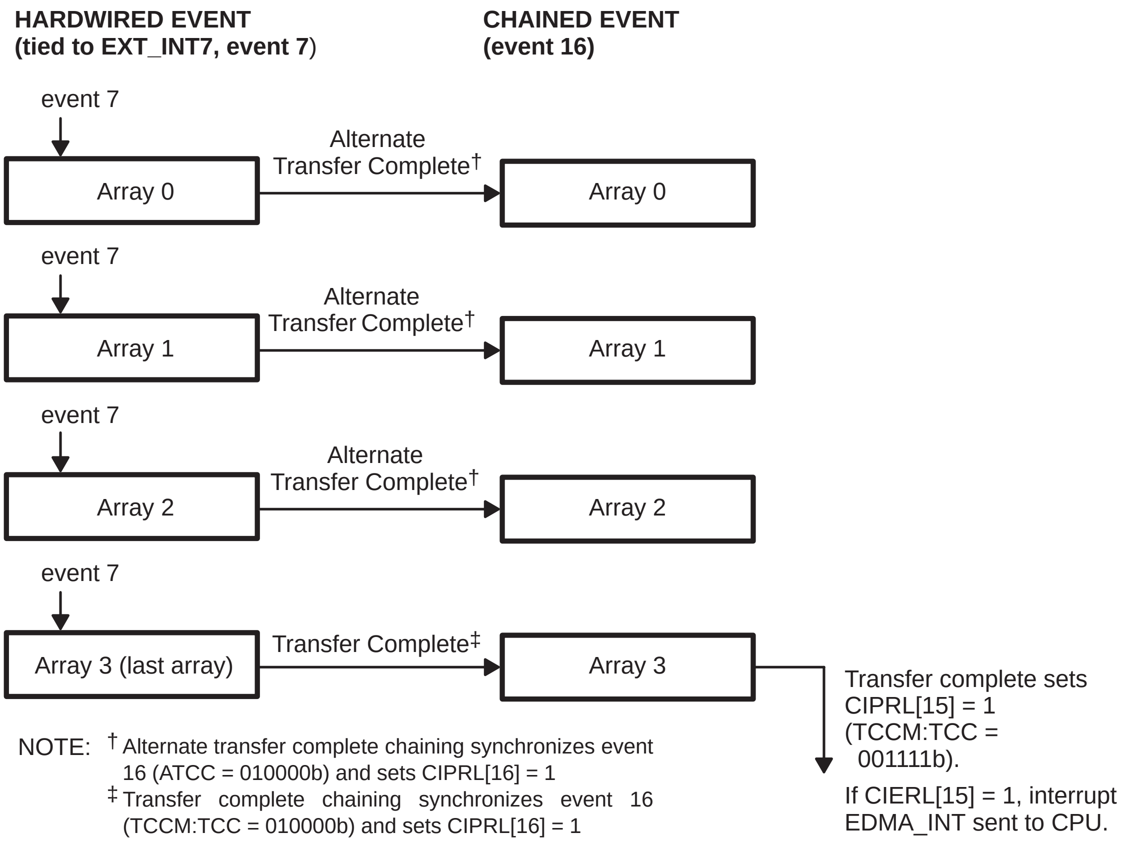
6.15.4 C64x Alternate Transfer Chaining Example

The following examples explain the alternate transfer complete chaining function in detail.

6.15.4.1 Servicing Input/Output FIFOs with a Single Event

Most common systems for ADSL, networking, and video applications require the use of a pair of external FIFOs that must be serviced at the same rate. One FIFO is used to buffer data input, and the other is used to buffer data output. The EDMA channels that are used to service these FIFOs can be set up for 2D transfers. While each FIFO is serviced with a different set of parameters, both can be signaled from a single event. For example, an external interrupt pin can be tied to the status flags of one of the FIFOs. When this event arrives, the EDMA needs to perform servicing for *both* the input and output streams. Without the alternate transfer complete chaining feature this would have required two events, and thus two external interrupt pins. The alternate transfer complete chaining feature allows the use of a single external interrupt pin (e.g. EXT_INT7). The EDMA setup and illustration for this example is shown in Figure 6–21.

Figure 6–21. Alternate Transfer Complete Chaining Example


HARDWIRED EVENT
(tied to EXT_INT7, event 7)
CHAINED EVENT
(event 16)
event 7
Array 0
Alternate Transfer Complete†
Array 0
event 7
Array 1
Alternate Transfer Complete†
Array 1
event 7
Array 2
Alternate Transfer Complete†
Array 2
event 7
Array 3 (last array)
Transfer Complete‡
Array 3
Transfer complete sets CIPRL[15] = 1 (TCCM:TCC = 001111b).
If CIERL[15] = 1, interrupt EDMA_INT sent to CPU.

NOTE: † Alternate transfer complete chaining synchronizes event 16 (ATCC = 010000b) and sets CIPRL[16] = 1
‡ Transfer complete chaining synchronizes event 16 (TCCM:TCC = 010000b) and sets CIPRL[16] = 1

SETUP

Channel 7 Parameters for Chaining

- ❑ Enable Transfer Complete Chaining:
  TCINT = 1
  TCCM:TCC = 010000b
- ❑ Enable Alternate Transfer Complete Chaining
  ATCINT = 1
  ATCC = 010000b

Channel 16 Parameters for Chaining

- ❑ Enable Transfer Complete Chaining:
  TCINT = 1
  TCCM:TCC = 001111b
- ❑ Disable Alternate Transfer Complete Chaining
  ATCINT = 0
  ATCC = don’t care

Event Enable Register (EER)

- ❑ Enable Channels 7:
  EERL[7] = 1

Channel Chain Enable Register (CCER)

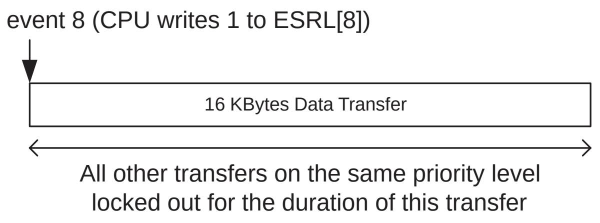
- ❑ Enable Chaining to Channel 16:
  CCERL[16] = 1

An EXT_INT7 event triggers a channel 7 array transfer. Upon completion of each intermediate array transfer of channel 7, alternate transfer complete chaining sets bit CIPRL[16] (specified by channel 7 ATCC) and provides a synchronization event to channel 16. Upon completion of the last array transfer of channel 7, transfer complete chaining—not alternate transfer complete chaining—sets bit CIPRL[16] (specified by its TCCM:TCC) and provides a synchronization event to channel 16. The completion of channel 16 sets bit CIPRL[15] (specified by its TCCM:TCC), which can generate an interrupt to the CPU if CIERL[15] = 1.

6.15.4.2 Breaking up Large Transfers with ATCC

Another very useful feature of the alternate transfer completion code is for breaking up large transfers. A large transfer may lock out other transfers of the same priority level (section 6.17) for the duration of the transfer. For example, a large transfer with high priority from the internal memory to the external memory via the EMIF may lock out other EDMA transfers on the high priority queue. In addition, this large high priority transfer may lock out the EMIF for a long period of time from lower priority channels.Therefore large transfers should be done on a low priority level. Figure 6–22 shows the EDMA setup and illustration of an example single large block transfer.

Figure 6–22. Single Large Block Data Transfer


event 8 (CPU writes 1 to ESRL[8])
16 KBytes Data Transfer
All other transfers on the same priority level locked out for the duration of this transfer

EDMA Channel 8 SETUP

Element Count (ELECNT) = 4069 (16 KB)

Frame Count (FRMCNT) = 0 (1 frame)

ATCC = don’t care

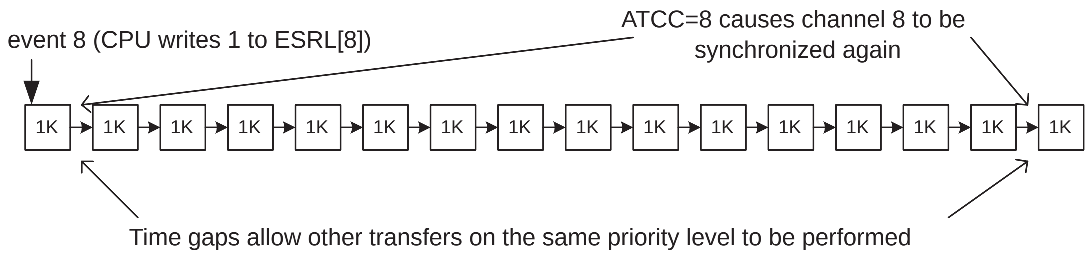
ATCINT = 0

1D transfer of 16 KByte elements

A solution to the above problem is to use the alternate transfer completion code to break up a large transfer into smaller transfers. For example, to move a single large block of memory (16K bytes), the EDMA is set up to actually perform a 2D array synchronized transfer. The element count is set to a “reasonable” value, where reasonable is derived from the amount of time it would take to move this smaller amount of data. Assume 1K byte is a reasonable small transfer in this example. The EDMA is set up to transfer 16 arrays of 1K byte elements, for a total of 16K byte elements. The ATCC field in the options parameter is set to the same value as the channel number. In this example, EDMA channel 8 is used and ATCC is also set to 8. The transfer complete code TCCM:TCC may be set to a different value to cause an interrupt to the CPU at the end of the transfer.

The CPU starts the EDMA transfer by writing to the appropriate bit of the event set register (ESRL[8]). The EDMA transfers the first 1K-byte array. Upon completion of the first array (an intermediate transfer), alternate transfer complete code chaining generates a synchronization event to channel 8, a value specified by the ATCC field. This ATCC-generated synchronization event causes EDMA channel 8 to transfer the next 1K-byte array. This process continues until the transfer parameters are exhausted, at which point the EDMA will have completed the 16K-byte transfer. This method breaks up a large transfer into smaller packets, thus providing natural time slices in the transfer such that other events may be processed. Figure 6–23 shows the EDMA setup and illustration of the broken up smaller packet transfers.

Figure 6–23. Smaller Packet Data Transfers


event 8 (CPU writes 1 to ESRL[8])
ATCC=8 causes channel 8 to be synchronized again
1K 1K 1K 1K 1K 1K 1K 1K 1K 1K 1K 1K 1K 1K 1K 1K
Time gaps allow other transfers on the same priority level to be performed

EDMA Channel 8 SETUP

Element Count (ELECNT) = 256 (1KB)	ATCC = 8
Array Count (FRMCNT) = 15 (16 frames)	ATCINT = 1
Array Index (FRMIDX) = 1024 (1KB strides)	
Array sync (FS) = 0	
CCER[8] = 1†	*2D transfer of 16 1KByte arrays*

† If another channel that has a system event (such as channel 4) is used, both the event enable and channel chain enable must be set (EER[4] and CCER[4]) for the channel to transfer after both the external sync and the ATCC are received.

6.16 Peripheral Device Transfers (TMS320C64x only)

The C64x EDMA supports the peripheral device transfer mode (PDT), which provides an efficient way to transfer large amounts of data between an external peripheral device and an external memory device that share the same data pins. In normal operation, this type of transfer requires an EMIF read of the external source followed by an EMIF write to the external destination. When PDT is enabled, data is driven by the external source directly, and written to the external destination in the same data bus transaction. Refer to Chapter 10, *External Memory Interface* for a detailed description of PDT.

PDT transfers are classified in terms of the memory on the EMIF. A PDT write is a transfer from a peripheral to memory (memory is physically written). To enable a PDT write from an external peripheral source to an external memory destination, the PDTD field in the EDMA options field must be set to 1.

A PDT read is a transfer from memory to a peripheral (memory is physically read). To enable a PDT read from an external memory source to an external peripheral destination, the PDTS field in the EDMA options field must be set to 1.

PDT writes and PDT reads are mutually exclusive. In other words, PDTS and PDTD cannot both be set to 1.

6.17 Resource Arbitration and Priority Processing

The EDMA channels can have programmable priority. The PRI bit in options specifies the priority levels. The highest priority available in the system is level 0 or the urgent priority. L2 requests comprise of data and program requests from the CPU, L1 and L2 controllers. Table 6–16 shows the available priority levels for the different requestors.

Table 6–16. Programmable Priority Levels for Data Requests

PRI(31:29)	C621x/C671x Priority Level	C621x/C671x Requestors	C64x Priority Level	C64x Requesters
000b	Level0; urgent priority	L2 controller	Level0; urgent priority	L2 controller, EDMA, QDMA
001b	Level1; high priority	EDMA, QDMA and/or HPI	Level1; high priority	L2 controller, EDMA, QDMA
010b	Level2; low priority	EDMA, QDMA	Level 2; medium priority	L2 controller, EDMA, QDMA and/or HPI/ PCI
011b	Reserved	Reserved	Level 3, low priority	L2 controller, EDMA, QDMA
100b –111b	Reserved	Reserved	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 different levels of priority. Refer to section 6.18.

6.17.1 Priority Queue Status Register (PQSR)

The priority queue status register (PQSR) shown in Figure 6–24 (C621x/C671x) and Figure 6–25 (C64x) indicates whether the transfer request queue is empty on the priority level queues. Status bits PQ in the PQSR provide the status of the queues. A ’1’ in the PQ bit indicates that there are no requests pending in the respective priority level queue. If PQSR[0] is ’1’, this means all L2 requests for data movement have been completed and there are no requests pending in the priority level 0 queue.

Figure 6–24. Priority Queue Status Register(PQSR)(C621x/C671x)

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

Figure 6–25. Priority Queue Status Register(PQSR)(C64x)

31	3	2	1	0
rsvd	PQ3	PQ2	PQ1	PQ0
R, +0	R, +1	R, +1	R, +1	R, +1

The PQ 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 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.

6.17.2 Transfer Request Queue Length

6.17.2.1 TMS320C621x/C671x Transfer Request Queues

The C621x/C671x has three transfer request queues: Q0, Q1, and Q2. The different priority level transfer requests (PRI field in the options parameter) are sorted into Q0, Q1, and Q2 as shown in Table 6–17. Urgent priority queue Q0 is reserved for L2 controller transfer requests. The lower priority queues Q1 and Q2 can be used for EDMA, QDMA, and HPI transfers. Table 6–17 also shows how each queue is divided among the different requesters.

Table 6–17. Transfer Request Queues (C621x/C671x)

Queue	Priority Level (PRI)	Total Queue Length (fixed)	Maximum Queue Length Available to Requester	
Q0	0; urgent priority	6	L2 controller	6
Q1	1; high priority	13	EDMA QDMA HPI	8 3 2
Q2	2; low priority	11	EDMA QDMA	8 3

6.17.2.2 Priority Queue Allocation Registers (PQAR) (C64x only)

The C64x has four transfer request queues—Q0, Q1, Q2, and Q3. The different priority level transfer requests (PRI field) are sorted into the corresponding queues, as shown in Table 6–18. Table 6–18 shows how Q0, Q1, Q2, and Q3 are divided among the different requesters. The queue length available to EDMA requests is programmable via the priority queue allocation registers, (PQARs), shown in Figure 6–26, Figure 6–27, Figure 6–28, and Figure 6–29.

L2 requests on C64x can be programmed on any one of the queues via the cache configuration register (CCFG). The queue length available for L2 requests is programmable via the L2 allocation registers. See Chapter 3, *TMS320C621x/C671x/C64x Two-Level Internal Memory* for details.

Table 6–18. Transfer Request Queues (C64x)

Queue	Priority Level (PRI)	Total Queue Length (fixed)	Default Queue Length Available to Requester	
Q0	0; urgent priority	16	L2 controller/QDMA	6 (programmable in L2ALLOC0)
			EDMA	2 (programmable in PQAR0)
Q1	1; high priority	16	EDMA	6 (programmable in PQAR1)
			L2 controller/QDMA	2 (programmable in L2ALLOC1)
Q2	2; medium priority	16	EDMA	2 (programmable in PQAR2)
			L2 controller QDMA	2 (programmable in L2ALLOC2)
			HPI/PCI	4
Q3	3; low priority	16	EDMA	6 (programmable in PQAR3)
			L2 controller/QDMA	2 (programmable in L2ALLOC3)

Figure 6–26. Priority Queue Allocation Register 0 (PQAR0) (C64x only)

31	3	2	1	0
rsvd		PQA2	PQA1	PQA0
R,+0		RW,+1	RW,+1	RW,+0

Figure 6–27. Priority Queue Allocation Register 1 (PQAR1) (C64x only)

31	3	2	1	0
rsvd		PQA2	PQA1	PQA0
R,+0		RW,+1	RW,+1	RW,+0

Figure 6–28. Priority Queue Allocation Register 2 (PQAR2) (C64x only)

31	3	2	1	0
rsvd		PQA2	PQA1	PQA0
R,+0		RW,+0	RW,+1	RW,+0

Figure 6–29. Priority Queue Allocation Register 3 (PQAR3) (C64x only)

31	3	2	1	0
rsvd		PQA2	PQA1	PQA0
R,+0		RW,+1	RW,+1	RW,+0

A new transfer request submission to a full queue will cause an EDMA stall until the existing request in the queue is processed to make room in the queue. See section 6.21 for details on transfer requests submission.

Note: Writes to the PQSRs should be performed only when the corresponding queues are empty of all outstanding transfer requests. This can be achieved by event disabling and by checking the PQ bits in the PQSR (section 6.17.1).

6.18 EDMA Performance

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

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

6.19 Quick DMA (QDMA)

Quick DMA (QDMA) provides one of the most efficient ways to move data. QDMA 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 to issue single, independent transfers to quickly move data.

6.19.1 QDMA Registers

Since the QDMA is used for quick, one-time transfers it does not have the capability to reload a count or link. The count reload/link address register is therefore not available to the QDMA. The QDMA can be used for chaining transfers. The QDMA registers are not updated during or after a transfer by the hardware. They retain the values that were submitted. All EDMA transfers are submitted using frame synchronization (1D) or block synchronication (2D). See section 6.19.3.

The QDMA consists of two sets of memory mapped, write-only registers, similar to an EDMA parameter entry. Figure 6–30 shows the first set, which is a direct mapping of the five QDMA registers required to configure a transfer. There is no count reload, no link address, and the LINK field of the options parameter is reserved. Writing to the QDMA registers configures, but does not submit, a QDMA transfer request. Figure 6–31 shows the pseudo registers for this set. Writing to the pseudo registers submits a transfer request.

Figure 6–30. QDMA Registers[‡]

QDMA_OPT 0200 0000h

31 … 0
QDMA options[†]
W, + 0

QDMA_SRC 0200 0004h

31 … 0
SRC address
W, + 0

QDMA_CNT 0200 0008h

31 … 16	15 … 0
Array/frame count	Element count
W, + 0	W, + 0

QDMA_DST 0200 000ch

31 … 0
DST address
W, + 0

QDMA_IDX 0200 0010h

31 … 16	15 … 0
Arrary/frame index	Element index
W, + 0	W, + 0

[†] The fields in this register are shown in Figure 6–32.
[‡] For C64x, QDMA registers are readable and writeable (RW).

Figure 6–31. QDMA Pseudo Registers

QDMA_S_OPT 0200 0020h

31 … 0
QDMA options[†]
W, + 0

QDMA_S_SRC 0200 0024h

31 … 0
SRC address
W, + 0

QDMA_S_CNT 0200 0028h

31 … 16	15 … 0
Arrary/frame count	Element count
W, + 0	W, + 0

QDMA_S_DST 0200 002Ch

31 … 0
DST address
W, + 0

QDMA_S_IDX 0200 0030h

31 … 16	15 … 0
Arrary/frame index	Element index
W, + 0	W, + 0

[†] The fields in this register are shown in Figure 6–32.

Figure 6–32. QDMA Options Register (QDMA_OPT, QDMA_S_OPT)

31 29	28 27	26	25 24	23	22 21	20	19 16	15	14	13	12 1	0
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Rsvd	TCC5	TCC4	Rsvd	FS
W, +0	W, +0	W, +0	W, +0	W, +0	W, +0	W, +0	W, +0	W,+0	W,+0	W,+0	W,+0	W,+0

Note: TCCM apply to C64x only. For C621x/C671x, bit 15 is reserved W, +0.

Although the QDMA mechanism does not support event linking, it supports completion interrupts, as well as QDMA transfer complete chaining with EDMA events. QDMA completion interrupts are enabled and set in the same way as EDMA completion interrupts; through the use of the TCINT and TCC fields in the options register, and the CIPR and CIER of the EDMA. QDMA transfer-complete chaining and alternate transfer-complete chaining with EDMA events are enabled through setting the appropriate bits in the QDMA options register and CCER of the EDMA. QDMA transfer requests have the same priority restrictions as the EDMA. See section 6.17 for details.

6.19.2 QDMA Register Access

For C621x/C671x, each of the QDMA registers is write only. Reads of the QDMA registers will return invalid data. For C64x, the QDMA registers are readable and writeable. 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.

6.19.3 Initiating a QDMA Transfer

A QDMA transfer requires only one to five cycles to submit, depending on the number of registers that need to be configured. A typical QDMA transfer is performed by writing four of the parameter values to their registers followed by the write of the fifth parameter to its corresponding pseudo-register. All QDMA transfers are submitted using frame synchronization (1D) or block synchronization (2D), therefore the QDMA always requests a transfer of one complete frame (1D) or block (2D) of data. The value in the FS field of the QDMA options register is “don’t care.” There are no intermediate transfers in a QDMA transfer. Only one request is sent for any QDMA submission, and the number of elements transferred is shown in Table 6–19.

Table 6–19. QDMA Transfer Length

Transfer Dimension	Elements Transferred
1-D to 1-D	One frame, regardless of frame count
Other	One block, all arrays transferred

Thus, a typical submission sequence might look like the following:

```
QDMA_SRC = SOME_SRC_ADDRESS;
QDMA_DST = SOME_DST_ADDRESS;
QDMA_CNT = (NUMFRAME–1)<<16 | NUM_ELEMENTS;// 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
```

6.19.4 QDMA Performance

The QDMA mechanism is extremely efficient at submitting DMA requests. Stores to the QDMA registers are passed to L2 as regular writes rather than peripheral writes. A QDMA transfer requires only one to five cycles (one cycle write for each of the five QDMA registers) to submit, depending upon the number of registers that need to be configured. Therefore, the QDMA registers can be used within the context of tight loop algorithms if desired.

Furthermore, all of the QDMA registers retain their value after the request is submitted, so if a second transfer is to be performed with any of the same parameter settings, they do not need to be rewritten by the CPU. Only the changed registers must be rewritten, with the final parameter written to the appropriate pseudo-register to submit the transfer. As a result, subsequent QDMA requests can be processed in as little as one cycle per request.

6.19.5 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 2–3 EDMA cycles, as this is how long it takes to submit a transfer. The L2 controller includes a 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 lower levels as described in section 6.17. 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 requests with level 0 or reserved values 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.

6.20 Emulation Operation

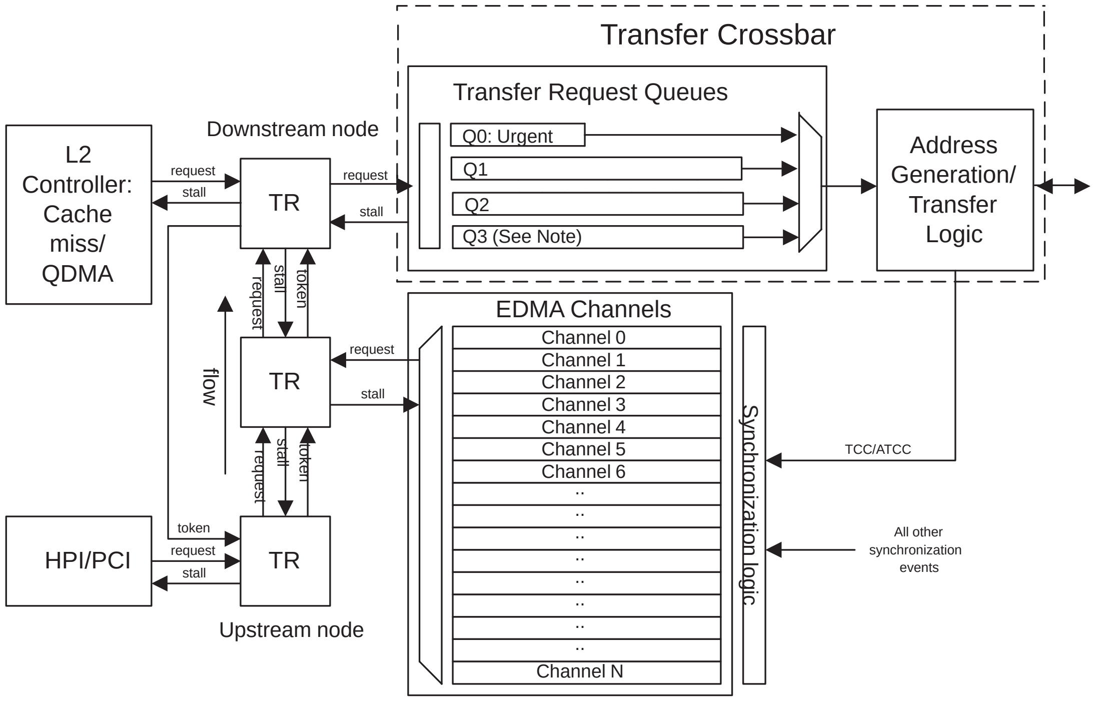
During debug using the emulator, the CPU may be halted on an execute packet boundary for single stepping, benchmarking, profiling, or other debug uses. During an emulation halt, EDMA operations continue.

6.21 Transfer Request Submission

6.21.1 Request Chain

All transfer requestors to the EDMA are connected to the transfer request chain. This is shown in Figure 6–33. A transfer request, once submitted, is shifted through the chain to the transfer crossbar (TC), where it is prioritized and processed. The transfer request can be for a single data element or for a large number of elements, as described in sections 6.7 and 6.8.

Figure 6–33. EDMA Block Diagram



Note: Q3 is available to C64x only. See section 6.17.2.2 for details.

The request chain provides an inherent priority scheme to the requestors. Assuming each makes a submission on the same cycle, the requestor closest to the TC (downstream requestor) arrives first, and the farthest (upstream requestor) arrives last. Once a request is within the request chain, it has priority over new submissions, such that the requests at the end of the chain do not get starved for servicing.

To prevent possible deadlock situations that would occur if a downstream requestor was held off from submission due to continuous submissions by upstream requestors, there is a round-robin scheme implemented within the chain's logic. A token is passed around the chain (for the token, it is a loop) in the downstream direction every CPU clock cycle. The transfer request node that has the token inverts the priority levels of its two requestors. Rather than giving priority to an existing request in the chain, located in the upstream node, priority is given to the local requestor with the token to submit a new request. Although this is a safeguard implanted into the EDMA, the high bandwidth of the EDMA relative to the speed at which requests are submitted has shown this to be inconsequential.

The requesters include the L2 controller, the EDMA channels, and the HPI/PCI.

6.21.1.1 L2 Controller Transfer Requests

The L2 controller submits all transfer requests for cache servicing, for accessing non-cacheable memory, and for QDMA transfers. See Chapter 3, *TMS320C621x/C671x/C64x* for details on the cacheability of memory.

For C621x/C671x, cache servicing requests are always made on the urgent priority level and are not visible to the user. For read requests, the cache controller always requests an L2 line in two parts, requesting the “missed” portion of the line first. The data transfers requested are based on the data location within the L2 line, as shown in Table 6–20. For write requests, as a result of flush/clean operations or eviction, the burst size is one complete L2 line.

Table 6–20. TMS320C621x/C671x Cache Controller Data Transfers

Data Location	First Transfer	Second Transfer
First 1/4	Front 1/2 line	Back 1/2 line
Second 1/4	Back 3/4	Front 1/4 line
Third 1/4	Back 1/2 line	Front 1/2 line
Fourth 1/4	Back 1/4 line	Front 3/4 line

For TMS320C64x, cache servicing requests can be made on any priority levels as specified in the P bits in the CCFG register. For read requests, the cache controller always requests an L2 line in two bursts of 64 Kbytes each, requesting the “missed” portion of the line first. For write requests, as a result of flush/clean operations or eviction, the cache controller transfers one complete L2 line in two bursts of 64 Kbytes each.

For both C621x/C671x and C64x, transfer requests by the L2 controller for non-cacheable memory are always equal to a single element and are used to load/store data from/to a non-cacheable location in external memory. These requests are also submitted with an urgent priority and are invisible to the user.

QDMA transfer requests have the same restrictions as the EDMA channels. See section 6.19 for details.

6.21.1.2 HPI/PCI Transfer Requests

The HPI/PCI automatically generates transfer requests to service host activity. These transfer request submissions are submitted only with a high priority and are invisible to the user. The HPI/PCI submits a transfer request for a single element read or write for fixed mode host accesses and a transfer request for a short data burst for autoincrement transfers. The burst size is always for eight or fewer elements. See section 6.17 for available HPI transfer request priority.

6.21.1.3 EDMA Channel Transfer Requests

The EDMA channel transfers can be submitted with either high, medium (C64x only), or low priority; with the recommendation that high priority be reserved for short bursts and single element transfers and low priority be used for longer (background) block moves. It is also recommended that transfers be divided between the two priority levels when applicable, as this helps to maximize the device performance.

6.21.2 Transfer Crossbar

Once a transfer request is at the end of the request chain, it is sent to the transfer crossbar (TC). Within the TC, the transfer request is shifted into one of the transfer request queues to await processing. The queue to which it is submitted is determined by the priority associated with it. The C621x/C671x has three fixed-length queues (Q0, Q1, Q2). The C64x has four priority level queues with programmable lengths. See section 6.17.2.2.

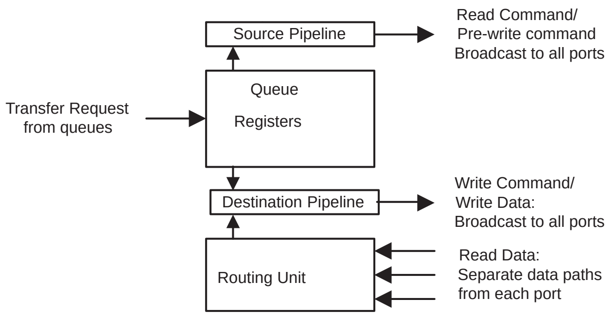
Once the transfer request reaches the head of its queue, it is submitted to the address generation/transfer logic to be processed. Only one transfer request from each queue can be serviced at a time by the address generation/transfer logic. To maximize the data transfer bandwidth in a system, all queues should be utilized.

6.21.3 Address Generation/Transfer Logic

The address generation/transfer logic block, shown in Figure 6–34, controls the transferring of data by the EDMA. There is one register set for each priority queue, which monitors the progress of a transfer. Within the register set for a particular queue, the current source address, destination address, and count are maintained for a transfer. These registers are not memory-mapped and are not available to the CPU.

The queue registers essentially function as a traditional DMA. They maintain the transfer parameters (source, destination, count, etc) during the data transfer. The queue registers send requests for data to be transferred. These requests are for small bursts, which are less than or equal to the total data size of the submitted transfer request. The actual size depends on the port performing the data reads or writes and is fixed by the hardware to maximize performance. This allows transfers initiated by different queues to occur simultaneously to one another. Due to the fact that the registers send *requests* for data transfers, the actual data movement occurs as soon as the ports are ready. So if the different queues request transfers to/from different ports then the transfers can occur at the same time. Transfer requests made to the same port(s) are arbitrated for priority.

Figure 6–34. Address Generation/Transfer Logic Block Diagram


Source Pipeline
Read Command/ Pre-write command Broadcast to all ports
Queue Registers
Transfer Request from queues
Destination Pipeline
Write Command/ Write Data: Broadcast to all ports
Routing Unit
Read Data: Separate data paths from each port

Each queue register set submits its transfer request to the appropriate source/destination pipeline to initiate a data transfer. There are three commands generated by the queue registers: pre-write, read, and write. Commands can be submitted to both source/destination pipelines once per cycle by any of the queue registers. The TC arbitrates every cycle (separately for each pipeline) to allow the highest priority command that is pending in the source/destination pipeline to be submitted. The pre-write command is issued to notify the destination that it is going to receive data. All ports have a small buffer available to receive a burst of data at the internal clock rate. Once the destination has available space to accommodate the incoming data, it sends an acknowledgement to the EDMA that it is ready.

After receiving the acknowledgement from the destination, a read command is issued to the data source. Data is read at the maximum frequency of the source and passed to the EDMA routing unit to be sent to the destination.

Once the routing unit receives the data, the data is sent along with a write command to its destination.

Due to the EDMA's capability to wait for the destination to be ready to receive data, the source resource is free to be accessed for other transfers until the destination is ready. This provides an excellent utilization of resources, and is referred to as write-driven processing. All commands and write data are sent from the EDMA to all resources on a single bus. The information is passed at the clock speed of the EDMA, and data from multiple transfers are interlaced when occurring simultaneously. Provided that multiple transfers (from different queues) have different sources, the transfers occur simultaneously.

The read data arrives on unique busses from each resource. This is to prevent contention and to ensure that data can be read at the maximum rate possible. Once the data arrives to the routing unit, the data that is available for the highest priority transfer is moved from its read bus to the write bus and sent to the destination.

6.22 Transfer Examples

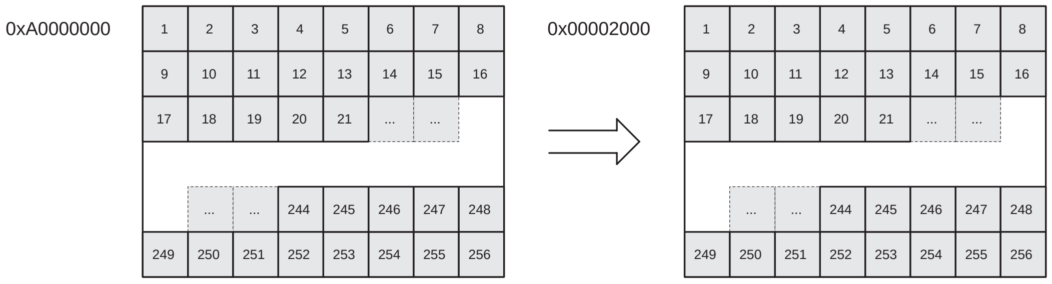
A wide variety of transfers can be performed by the EDMA depending on the parameter configuration. The more basic transfers can be performed either by an EDMA channel, or by submitting a QDMA. More complicated transfers or repetitive transfers require an EDMA channel to be used.

6.22.1 Block Move Example

The most basic transfer that can be performed by the EDMA is that of a block move. Often during device operation it is necessary to transfer a block of data from one location to another, usually between on- and off-chip memory.

In this example, a section of data is to be copied from external memory to internal L2 SRAM. The data block is 256 words and resides at address 0xA0000000 (CE2). It is to be transferred to internal address 0x00002000 (L2 block 0). The data transfer is shown in Figure 6–35.

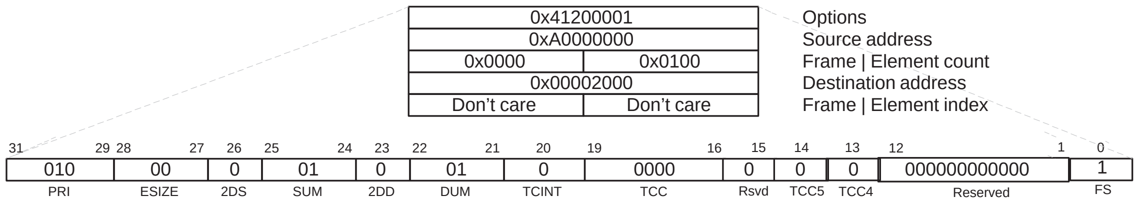
Figure 6–35. Block Move Diagram



The fastest way to perform this transfer is through a QDMA request. The QDMA request can be submitted in several different ways, the most basic being a frame-synchronized 1-D to 1-D transfer. This type of transfer is valid for block sizes of less than 64k elements. The transfer must be frame-synchronized so that all of the elements are transferred once the entry is submitted. QDMA submits all requests as frame-synchronized transfers, regardless of the FS bit value.

The parameters for this transfer are shown in Figure 6–36. Those that must be configured are QDMA options, source address, destination address, and element count.

Figure 6–36. Block Move QDMA Parameters



Note: PRI (bits 31:29) is set to 010b (C621x/C671x) or 011b (C64x) for low priority background transfer. TCCM (bits 14:13) are reserved on C621x/C671x.

The source address for the QDMA is set to the start of the data block in external memory, and the destination address is set to the start of the data block in L2. Since all data is contiguous, SUM and DUM are both set to 01b (increment). The priority (PRI) is set to low-priority for background transfer.

The CPU requires four cycles to submit the request for this transfer, one cycle for each register write. Fewer registers are required if any of the QDMA registers are already configured, with a minimum of one cycle. Three of the QDMA parameters must be written to their proper QDMA registers and one parameter must be written to its pseudo-register, which initiates the transfer. A sample QDMA submission for the above transfer is as follows:

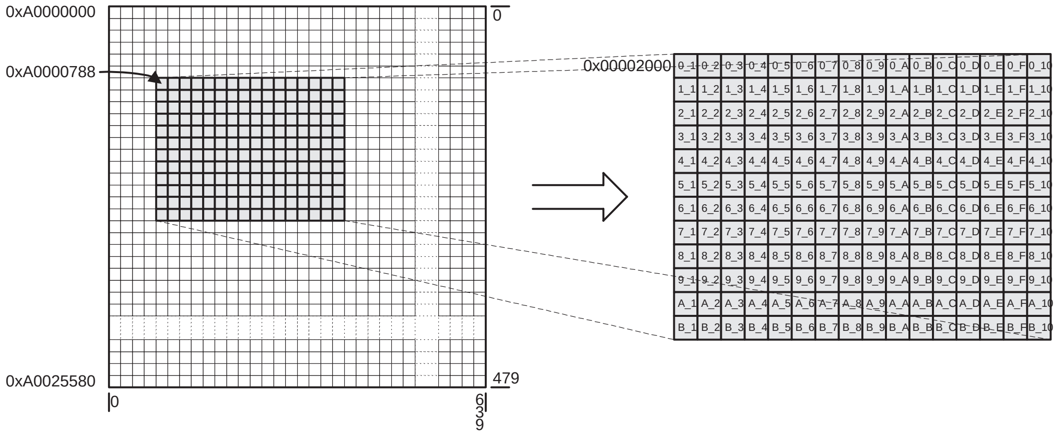
```
…
QDMA_SRC   = 0xA0000000;  /* Set source address      */
QDMA_DST   = 0x00002000;  /* Set destination address */
QDMA_CNT   = 0x00000100;  /* Set frame/element count */
QDMA_S_OPT = 0x41200001;  /* Set options and submit  */
…
```

A block that contains greater than 64k elements requires the use of both element count and array/frame count. Since the element count field is only 16 bits, the largest count value that can be represented is 65535. Any count larger than this needs to be represented with an array count as well. In order to transmit this amount of data a QDMA can still be used. Rather than a frame-synchronized 1-D to 1-D transfer, the QDMA needs to be configured as a block-synchronized (FS=1) 2-D to 2-D transfer.

6.22.2 Sub-frame Extraction Example

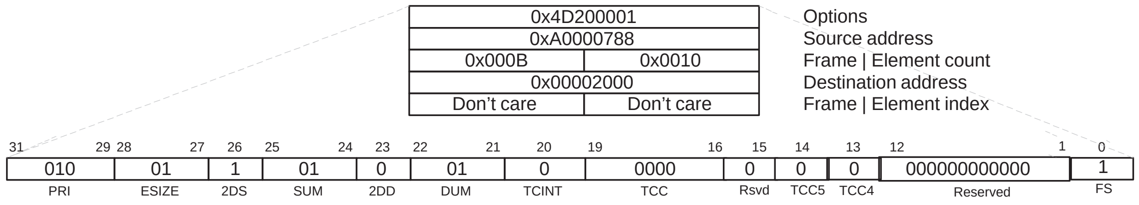
The EDMA has an efficient way of extracting a small frame of data from a larger one. By performing a 2-D to 1-D transfer, the EDMA can retrieve a portion of data for the CPU to process. In this example, a 640 x 480-pixel frame of video data is stored in external memory, CE2. Each pixel is represented by a 16-bit halfword. A 16 x 12-pixel sub-frame of the image is extracted for processing by the CPU. To facilitate more efficient processing time by the CPU, the EDMA places the sub-frame in internal L2 SRAM. Figure 6–37 depicts the transfer of the sub-frame from external memory to L2.

Figure 6–37. Sub-Frame Extraction


0xA0000000
0xA0000788
0xA0025580
0
479
0
639
0x00002000
0_1 0_2 0_3 0_4 0_5 0_6 0_7 0_8 0_9 0_A 0_B 0_C 0_D 0_E 0_F 0_10
1_1 1_2 1_3 1_4 1_5 1_6 1_7 1_8 1_9 1_A 1_B 1_C 1_D 1_E 1_F 1_10
2_1 2_2 2_3 2_4 2_5 2_6 2_7 2_8 2_9 2_A 2_B 2_C 2_D 2_E 2_F 2_10
3_1 3_2 3_3 3_4 3_5 3_6 3_7 3_8 3_9 3_A 3_B 3_C 3_D 3_E 3_F 3_10
4_1 4_2 4_3 4_4 4_5 4_6 4_7 4_8 4_9 4_A 4_B 4_C 4_D 4_E 4_F 4_10
5_1 5_2 5_3 5_4 5_5 5_6 5_7 5_8 5_9 5_A 5_B 5_C 5_D 5_E 5_F 5_10
6_1 6_2 6_3 6_4 6_5 6_6 6_7 6_8 6_9 6_A 6_B 6_C 6_D 6_E 6_F 6_10
7_1 7_2 7_3 7_4 7_5 7_6 7_7 7_8 7_9 7_A 7_B 7_C 7_D 7_E 7_F 7_10
8_1 8_2 8_3 8_4 8_5 8_6 8_7 8_8 8_9 8_A 8_B 8_C 8_D 8_E 8_F 8_10
9_1 9_2 9_3 9_4 9_5 9_6 9_7 9_8 9_9 9_A 9_B 9_C 9_D 9_E 9_F 9_10
A_1 A_2 A_3 A_4 A_5 A_6 A_7 A_8 A_9 A_A A_B A_C A_D A_E A_F A_10
B_1 B_2 B_3 B_4 B_5 B_6 B_7 B_8 B_9 B_A B_B B_C B_D B_E B_F B_10

To perform this transfer, the CPU can issue a QDMA request for a block-synchronized (FS=1) 2-D to 1-D transfer. Since the source is 2-D and the transfer is block-synchronized, the QDMA requests a transfer of the entire sub-frame. The parameters required for the QDMA registers to request this transfer are shown in Figure 6–38.

Figure 6–38. Sub-Frame Extraction QDMA Parameters



0x4D200001		Options
0xA0000788		Source address
0x000B	0x0010	Frame | Element count
0x00002000		Destination address
Don't care	Don't care	Frame | Element index

31–29	28–27	26	25–24	23	22–21	20	19–16	15	14	13	12–1	0
010	01	1	01	0	01	0	0000	0	0	0	000000000000	1
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Rsvd	TCC5	TCC4	Reserved	FS

Note: PRI (bits 31:29) is set to 010b (C621x/C671x) or 011b (C64x) for low priority transfer. TCCM is reserved on C621x/C671x.

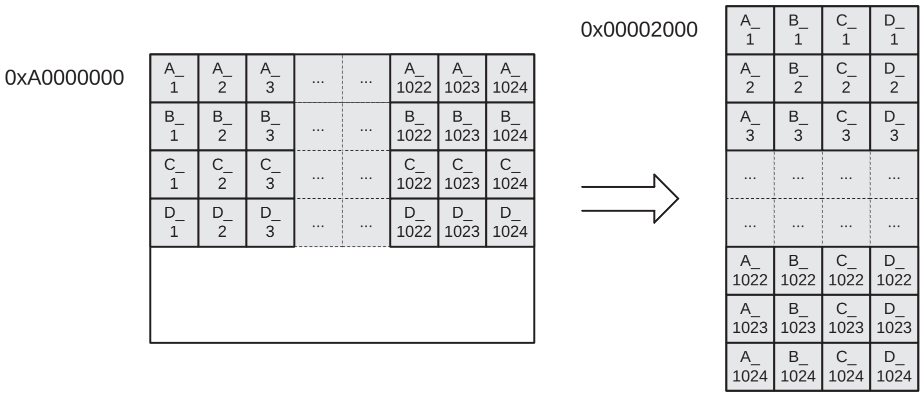
All of the address updates occur within the address generation/transfer logic. The array index provided is therefore the space *between* arrays of the sub-frame. Since each array of the video image is 640 pixels in length and each array of the sub-frame is 16 pixels in length, the array index is set to 2 bytes/element * (640 – 16) elements = 1248 bytes (0x4=0). The sub-frame is transferred to a block of contiguous memory. The element count (ELECNT) is set to 16, the number of elements per sub-frame array, and the array count (FRMCNT) is set to 11, one less than the number of arrays. The QDMA request is sent to the low-priority queue so that it does not interfere with any data acquisition that could be occurring.

Inversely, a 1-D to 2-D transfer can be used to perform the *insertion* of a sub-frame into a larger frame of data. For instance, with this example the sub-frame could be inserted back into the larger image after some processing by the CPU.

6.22.3 Data Sorting Example

Many applications require the use of multiple data arrays; it is often desirable to have the arrays arranged such that the first elements of each array are adjacent, the second elements are adjacent, and so on. Often this is not the format in which the data is presented to the device. Either data is transferred via a peripheral, with the data arrays arriving one after the other, or the arrays are located in memory, with each array occupying a portion (frame) of contiguous memory spaces. For these instances, the EDMA can be configured to reorganize the data into the desired format. Figure 6–39 shows the data sorting of element arrays.

Figure 6–39. Data Sorting Example Diagram


0xA0000000
A_1 A_2 A_3 ... ... A_1022 A_1023 A_1024
B_1 B_2 B_3 ... ... B_1022 B_1023 B_1024
C_1 C_2 C_3 ... ... C_1022 C_1023 C_1024
D_1 D_2 D_3 ... ... D_1022 D_1023 D_1024
0x00002000
A_1 B_1 C_1 D_1
A_2 B_2 C_2 D_2
A_3 B_3 C_3 D_3
A_1022 B_1022 C_1022 D_1022
A_1023 B_1023 C_1023 D_1023
A_1024 B_1024 C_1024 D_1024

The following values can be used to determine the fields required to use QDMA requests to organize the data in memory by ordinal position:

- ❑ F = The initial value of Frame Count
- ❑ E = The initial value of Element Count, as well as the Element Count Reload value
- ❑ S = The element size in bytes

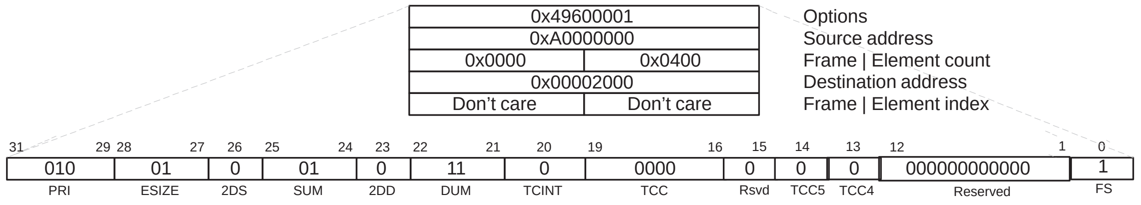
The QDMA can again be used to transfer this data. However, due to the arrangement of the data in the destination, it is not possible to accomplish this with a single submission. Instead a separate QDMA transfer request is submitted for each frame. If it is necessary to use an EDMA channel to perform this transfer, then an entry must be provided for each frame in the transfer in PaRAM. Also, the transfer must use the chaining feature to self-synchronize each frame on the completion of the previous. See section 6.15 for details.

This example focuses on the second case mentioned above, in which equal sized data arrays are located in external memory. It is not necessary for the arrays to be of equal length. In the case that the lengths vary, each QDMA submission or each EDMA reload parameter set in PaRAM would contain the corresponding new count value.

For this example it is assumed that the 16-bit data is located in external RAM, beginning at address 0xA0000000 (CE2). The QDMA is used to bring four frames of 1k half-words from their locations in RAM to internal data memory beginning at 0x00008000. The index value required is ELEIDX = F x S = 4 x 2 = 8.

Since separate QDMA transfer requests are to be submitted for each frame, only the ELEIDX value is used in the QDMA parameters. The CPU updates the destination address for each new frame. For the first frame of data, the values shown in Figure 6–40 are assigned to the QDMA registers:

Figure 6–40. Sorting QDMA Parameters



0x49600001		Options
0xA0000000		Source address
0x0000	0x0400	Frame | Element count
0x00002000		Destination address
Don’t care	Don’t care	Frame | Element index

31–29	28–27	26	25–24	23	22–21	20	19–16	15	14	13	12–1	0
010	01	0	01	0	11	0	0000	0	0	0	000000000000	1
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Rsvd	TCC5	TCC4	Reserved	FS

Note: PRI (bits 31:29) is set to 010b (C621x/C671x) or 011b (C64x) for low priority transfer. TCCM is reserved on C621x/C671x.

The QDMA submitted with the above parameters only transfers the first frame. For each subsequent frame the CPU must perform two stores to change the source address and the destination address manually. It is *not* necessary for the CPU to wait for each frame to complete before submitting a request for the next. The subsequent transfer requests submitted are stored in the transfer queues to await processing.

To summarize, the CPU performs four writes to configure the options field, the source address, the count, and the destination address (or any four of the five fields). The CPU then performs a write to the index pseudo-register (or the register still not configured) to submit the transfer request for the first frame. For each additional frame the CPU increments the source address by E x S (= 1024 x 2 = 2048) and stores this value to the source address register, and also increments the destination address by S and stores this value to the destination address *pseudo*-register to submit the transfer request.

If it is desired to have the EDMA notify the CPU when all of the transfers have completed, then the transfer request for the last frame should also have a modified options field to include a Transfer Complete Code value (and have TCINT = 1). See section 6.14 for details.

6.22.4 Peripheral Servicing Examples

An important capability of the EDMA is its ability to service peripherals in the background of CPU operation, without requiring any CPU intervention. Through proper initialization of the EDMA channels, they can be configured to continuously service on- and off-chip peripherals throughout the device operation. Each event available to the EDMA has its own dedicated channel, and all channels operate simultaneously. This means that all data streams can be handled independently with little or no consideration for what else is going on in the EDMA.

Since all EDMA channels are always synchronized, there are no special setups required to configure a channel to properly service a particular event. The only requirements are to use the proper channel for a particular transfer and to enable the channel’s event in the EER or CCER (unless the CPU synchronizes the channel).

When programming an EDMA channel to service a peripheral, it is necessary to know how data is to be presented to the DSP. Data is always provided with some kind of synchronization event, and is either one element per event (non-bursting), or multiple elements per event (bursting).

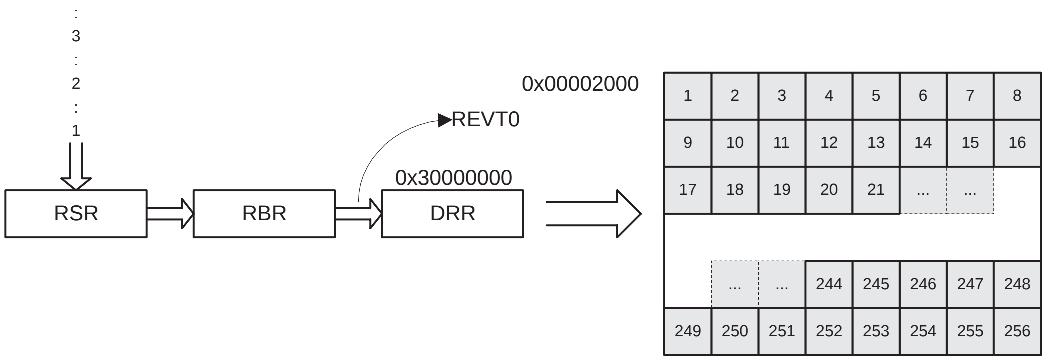
6.22.5 Non-bursting Peripherals

Non-bursting peripherals include the on-chip McBSPs and many external devices such as codecs. Regardless of the peripheral, the EDMA channel configuration is the same.

The on-chip McBSPs are the most commonly used peripherals in a C6000 system. EDMA channels 12 and 13 are dedicated to McBSP0 transmit and receive events, and channels 14 and 15 are dedicated to McBSP1 transmit and receive events. The transmit and receive data streams are treated independently by the EDMA. While a standard DMA channel could be used in split-mode to handle transmit and receive data, there are a number of restrictions with this due to the sharing of resources. The EDMA channels do not have these restrictions. Although most serial applications call for similar data formats both to and from the McBSP, this is not a requirement for reliable operation with the EDMA. The transmit and receive data streams can have completely different counts, data sizes, and formats.

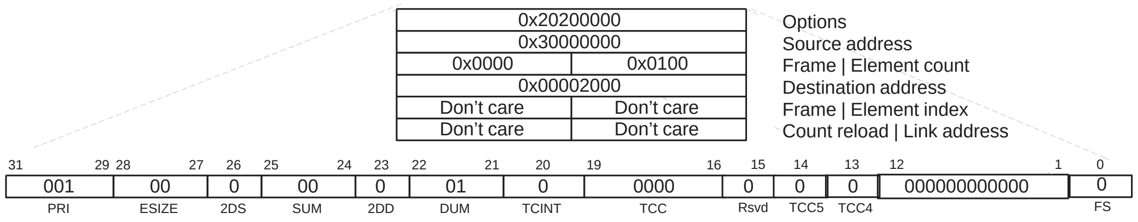
If the previous block move example were changed such that the 256 words were received by McBSP0 to be transferred to internal L2 SRAM, this is easily handled by EDMA channel 13, which is synchronized by REVT0. A block diagram of this transfer is shown in Figure 6–41.

Figure 6–41. McBSP Servicing for Incoming Data



To transfer the incoming data stream to its proper location in L2 memory, the EDMA channel must be set up for a 1-D to 1-D transfer with element synchronization (FS = 0). Since an event (REVT0) is generated for every word as it arrives, it is necessary to have the EDMA issue the transfer request for each element individually. The channel entry for this transfer is shown in Figure 6–42.

Figure 6–42. EDMA Parameters for Servicing Incoming McBSP Data



0x20200000		Options
0x30000000		Source address
0x0000	0x0100	Frame | Element count
0x00002000		Destination address
Don’t care	Don’t care	Frame | Element index
Don’t care	Don’t care	Count reload | Link address

31–29	28–27	26	25–24	23	22–21	20	19–16	15	14	13	12–1	0
001	00	0	00	0	01	0	0000	0	0	0	000000000000	0
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Rsvd	TCC5	TCC4		FS

Note: TCCM is reserved on C621x/C671x.

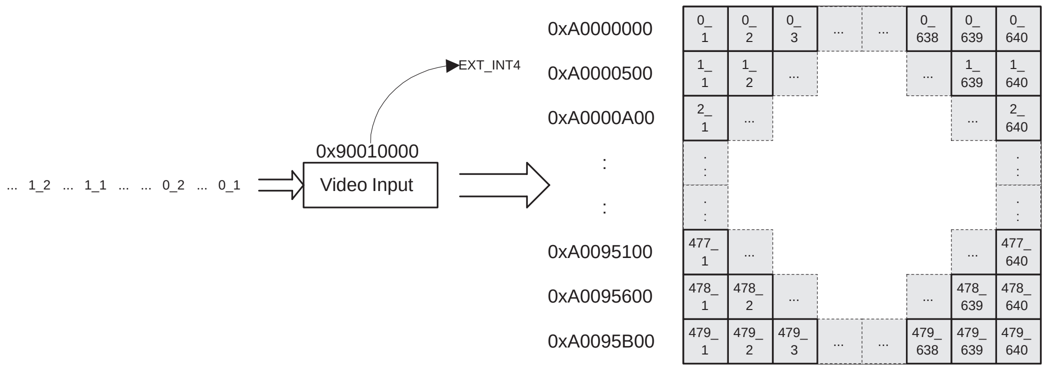
The source address of the EDMA channel is set to the DRR address for McBSP0, and the destination address is set to the start of the data block in L2. Since the address of the DRR is fixed, SUM is set to 00b (no modification). The destination address is left at 01b (increment) as in the previous example. The priority level chosen in this example is based on the premise that serial data is typically a high priority, so that samples are not missed. Each transfer request by this channel are made on the high-priority queue (Q1).

6.22.5.1 Bursting Peripherals

Higher bandwidth applications require that multiple data elements be presented to the DSP for every sync event. This frame of data can either be from multiple sources that are working simultaneously or a single high-throughput peripheral that streams data to/from the DSP.

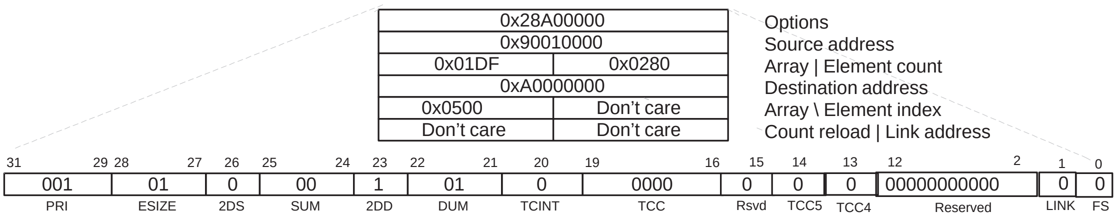
In this example a video framer is receiving a video frame from a camera and presenting it to the DSP one array at a time. The video image is 640 x 480 pixels, with each pixel represented by a 16-bit element. The image is to be stored in external memory. A diagram depicting this is shown in Figure 6–43.

Figure 6–43. Bursting Peripheral



To transfer data from an external peripheral to an external buffer one array at a time based on EXT_INT4, channel 4 must be configured. There are two types of transfers that are suitable for this: a 1-D to 1-D transfer with frame synchronization (FS = 1) or a 1-D to 2-D transfer with array synchronization (FS = 0); they are functionally identical. Due to the nature of the data (a video frame made up of arrays of pixels) the destination is essentially a 2-D entity. The parameter options to service the incoming data with a 1-D to 2-D transfer are shown in Figure 6–44.

Figure 6–44. EDMA Parameters to Service Peripheral Bursts



Note: TCCM is reserved on C621x/C671x.

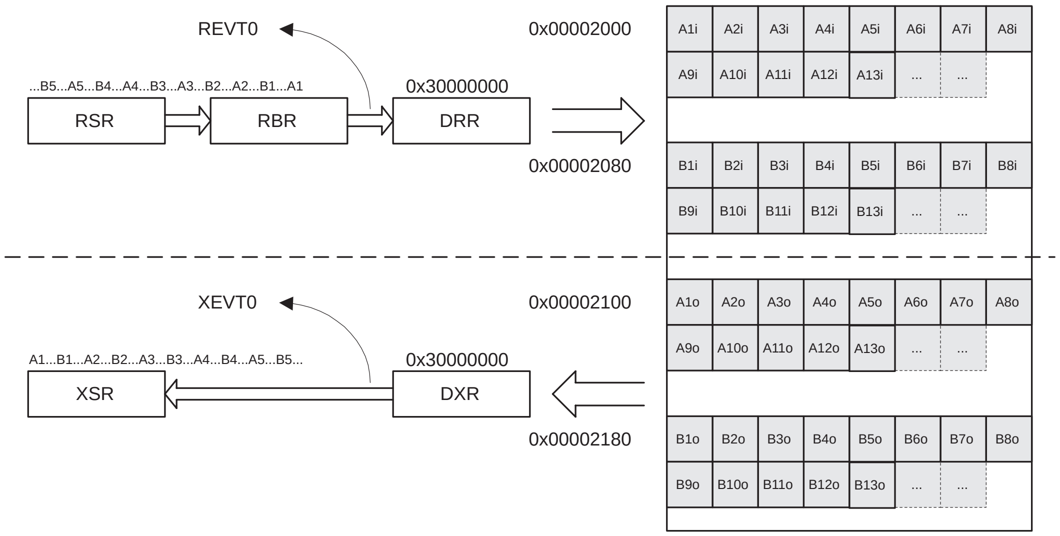
The source address is set to the location of the video framer peripheral, and the destination address to the start of the data buffer. Since the input address is static, SUM is set to 00b. The destination is made up of arrays of contiguous, linear elements. Therefore DUM is set to 01b (increment). The element count is equal to the number of pixels in an array, 640. The array count is equal to one less than the total number of arrays in the block, 479. An array index, equal to the difference between the starting addresses of each array, is required. Since each pixel is represented by a halfword, the array index is equal to twice the element count, or 1280 bytes.

6.22.5.2 Continuous Operation

Configuring an EDMA channel to receive a single frame of data can be useful, and is applicable to some systems. A majority of the time, however, data is going to be continuously transmitted and received throughout the entire operation of the DSP. In this case, it is necessary to implement some form of linking such that the EDMA channels continuously reload the necessary parameter sets.

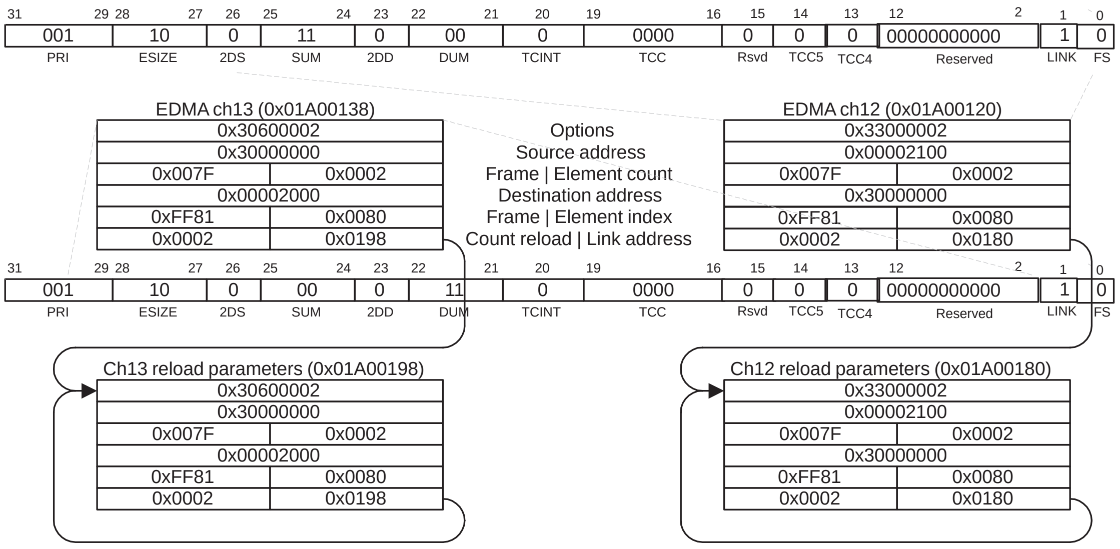
In this example, McBSP0 is configured to transmit and receive data on a T1 array. To keep the example simple, only two channels are active for both transmit and receive data streams. Each channel receives packets of 128 elements. The packets are transferred from the serial port to L2 memory and from L2 memory to the serial port, as shown in Figure 6–45.

Figure 6–45. Continuous McBSP Servicing by EDMA


REVT0
0x00002000
...B5...A5...B4...A4...B3...A3...B2...A2...B1...A1
0x30000000
RSR
RBR
DRR
0x00002080
A1i A2i A3i A4i A5i A6i A7i A8i
A9i A10i A11i A12i A13i ... ...
B1i B2i B3i B4i B5i B6i B7i B8i
B9i B10i B11i B12i B13i ... ...
XEVT0
0x00002100
A1...B1...A2...B2...A3...B3...A4...B4...A5...B5...
0x30000000
XSR
DXR
0x00002180
A1o A2o A3o A4o A5o A6o A7o A8o
A9o A10o A11o A12o A13o ... ...
B1o B2o B3o B4o B5o B6o B7o B8o
B9o B10o B11o B12o B13o ... ...

The McBSP generates REVT0 for every element received and XEVT0 for every element transmitted. To service the data streams, EDMA channels 12 and 13 must be set up for 1-D to 1-D transfers with element synchronization (FS = 0). In order to service the McBSP continuously throughout DSP operation, the channels must be linked to a duplicate entry in the parameter RAM. After all frames have been transferred, the EDMA channels reload and continue. The channel entries for these transfers are shown in Figure 6–46.

Figure 6–46. EDMA Parameters for Continuous McBSP Servicing



Note: TCCM is reserved on C621x/C671x.

Receive Channel

EDMA channel 13 is used to service the incoming data stream of McBSP0. As in the previous example, the source address is set to that of the DRR register, and the destination address is set to the first element of the data block. Since there are two data channels being serviced, A and B, and they are to be located separately within the L2 SRAM, the destination address update mode uses element and frame indexing (DUM = 11b). The element index is set to the offset between the first element of each channel's data section and the frame index is the offset between the second element of channel A and the first element of channel B. Since elements are 8-bit, the ESIZE field is set to 10b.

In order to facilitate continuous operation, a copy of the channel entry is placed in parameter RAM at address 0x01A01980. The LINK option is set and the link address is provided in the entry. Upon exhausting channel 13's element and frame counts, the parameters located at the link address are loaded into channel 13's parameter set and operation continues. This function continues throughout DSP operation until halted by the CPU.

The parameter table must keep track of the element count within the frame since each element is sent individually (FS = 0). It is therefore required that an element count reload is provided in the parameter set. This value is reloaded to the element count field every time the element count reaches zero.

Transmit Channel

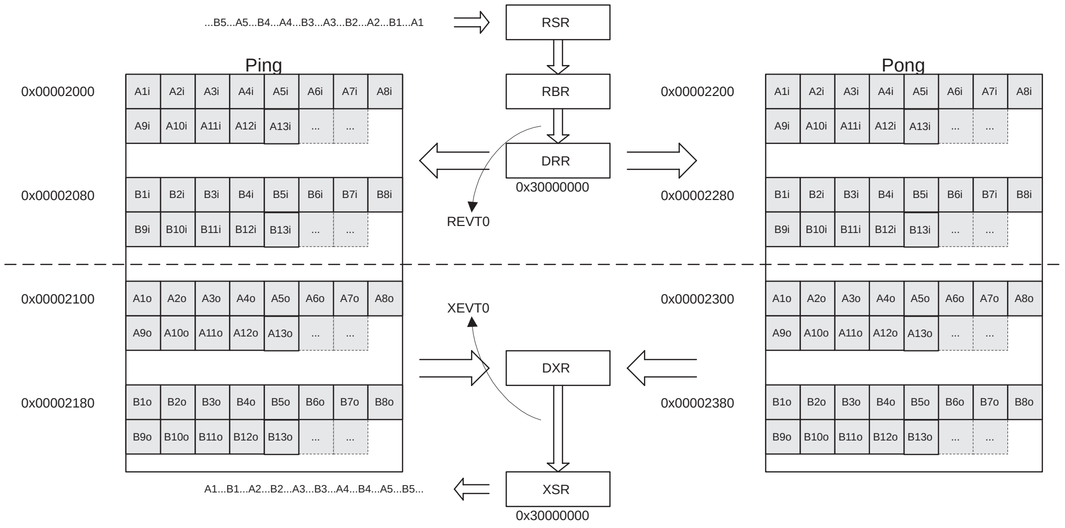
EDMA channel 12 services the outgoing data stream of McBSP0. Its configuration is essentially the opposite of channel 13's for this application since the input and output data is symmetrical. The element and frame counts are identical, as are the index values. The options are reversed, such that the source is updated using the programmed index values while the destination address is held constant. The source address provided to the channel is that of the beginning of channel A's output data, and the destination address is that of the DXR. Linking is also used to allow for continuous operation by the EDMA channel, with a duplicate entry in the parameter RAM.

6.22.5.3 Ping-Pong Buffering

Although the configuration presented above allows the EDMA to service a peripheral continuously, there are a number of restrictions it presents to the CPU. Since the input and output buffers are continuously being filled/emptied, in order for the CPU to process the data, it must match the pace of the EDMA very closely. The EDMA receive data must always be placed in memory before the CPU accesses it, and the CPU must provide the output data before the EDMA transfers it. Though not impossible, this is an unnecessary challenge. It is particularly difficult in a two-level cache system.

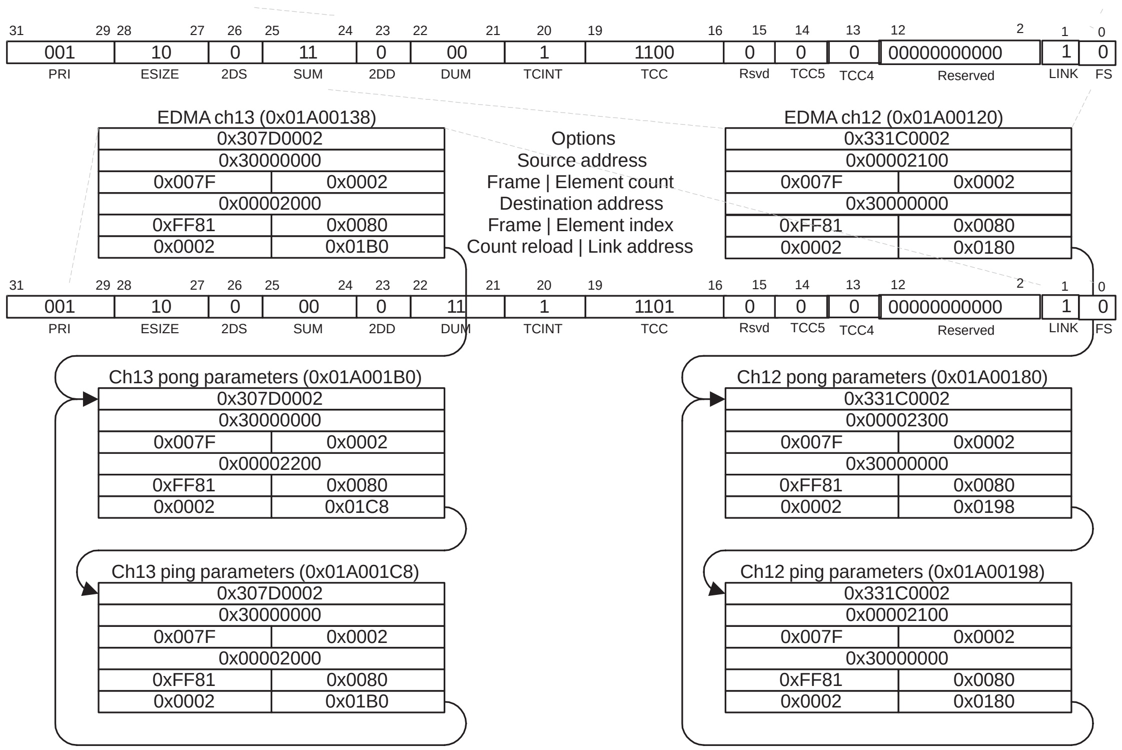
A simple technique to implement, which allows the CPU activity to be distanced from the EDMA activity, is to use ping-pong buffering. This simply means that there are multiple (usually two) sets of data buffers for all incoming and outgoing data streams. While the EDMA is transferring data in to and out of the ping buffers, the CPU is manipulating the data in the pong buffers. When both CPU and EDMA activity completes, they switch. The EDMA then writes over the old input data and transfers the new output data. The ping-pong scheme for this example is shown in Figure 6–47.

Figure 6–47. Ping-Pong Buffering for McBSP Data


...B5...A5...B4...A4...B3...A3...B2...A2...B1...A1
RSR
RBR
DRR
0x30000000
REVT0
XEVT0
DXR
XSR
0x30000000
A1...B1...A2...B2...A3...B3...A4...B4...A5...B5...
Ping
Pong
0x00002000
0x00002080
0x00002100
0x00002180
0x00002200
0x00002280
0x00002300
0x00002380
A1i A2i A3i A4i A5i A6i A7i A8i
A9i A10i A11i A12i A13i ... ...
B1i B2i B3i B4i B5i B6i B7i B8i
B9i B10i B11i B12i B13i ... ...
A1o A2o A3o A4o A5o A6o A7o A8o
A9o A10o A11o A12o A13o ... ...
B1o B2o B3o B4o B5o B6o B7o B8o
B9o B10o B11o B12o B13o ... ...

To change the continuous operation example such that a ping-pong buffering scheme is used, the EDMA channels need only a moderate change. Instead of one parameter set, there are two; one for transferring data to/from the ping buffers, one for transferring data to/from the pong buffers. As soon as one transfer completes, the channel loads the entry for the other and the data transfers continue. The EDMA channel configuration required for this is shown in Figure 6–48.

Figure 6–48. EDMA Parameters for Ping-Pong Buffering



Note: TCCM is reserved on C621x/C671x.

Each channel has two parameter sets, ping and pong. The EDMA channel is initially loaded with the ping parameters. The link address for the ping entry is set to the PaRAM offset of the pong parameter set, and vice versa. The channel options, count values, index values are all identical between the ping and pong parameters for each channel. The only differences are the link address provided and the address of the data buffer in internal memory.

Synchronization with the CPU

In order to utilize the ping-pong buffering technique, it is necessary to signal to the CPU when it can begin to access the new data set. After the CPU finishes processing an input buffer (ping), it waits for the EDMA to complete before switching to the alternate (pong) buffer.

In this example, both channels provide their channel numbers as their report word and set TCINT to ‘1’ to generate an interrupt after completion. When channel 13 fills an input buffer CIP13 is set to ‘1’ and when channel 12 empties an output buffer CIP12 is set to ‘1’. The CPU must manually clear these bits.

With the channel parameters set as above, the CPU can simply poll the CIPR to determine when to switch. The EDMA and CPU could alternatively be configured such that the channel completion interrupts the CPU. By doing this, the CPU would be able to service a background task while waiting for the EDMA to complete.

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.

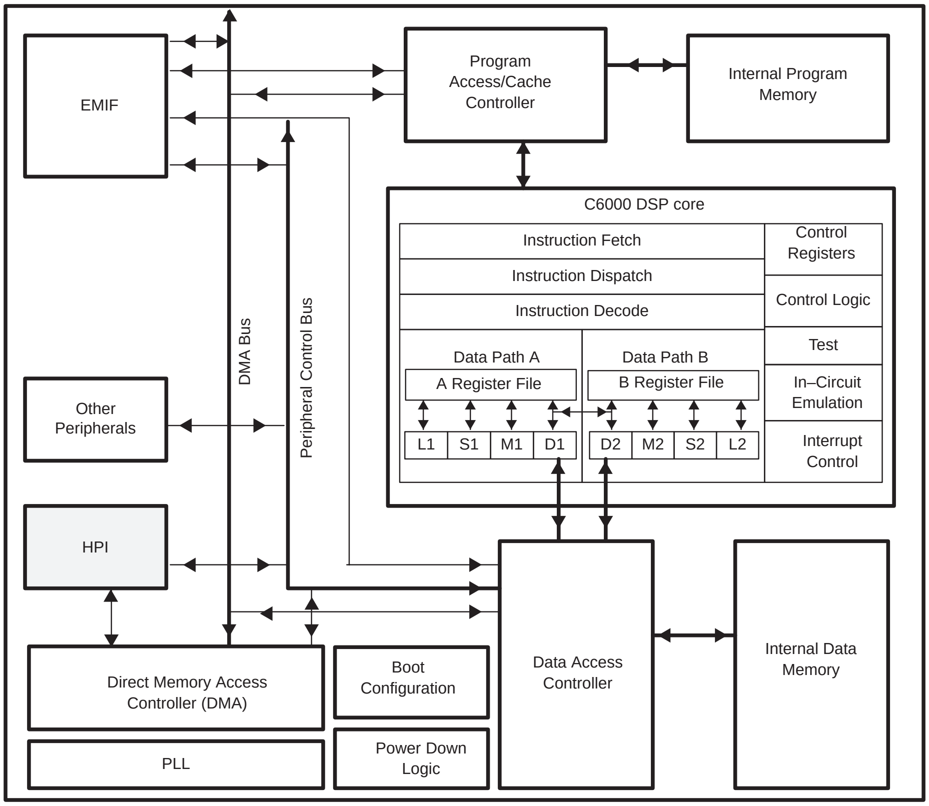
Topic **Page**

7.1 Overview

The host-port interface (HPI) is a 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/EDMA controller. 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. For the C64x, the CPU can also access the HPIA.

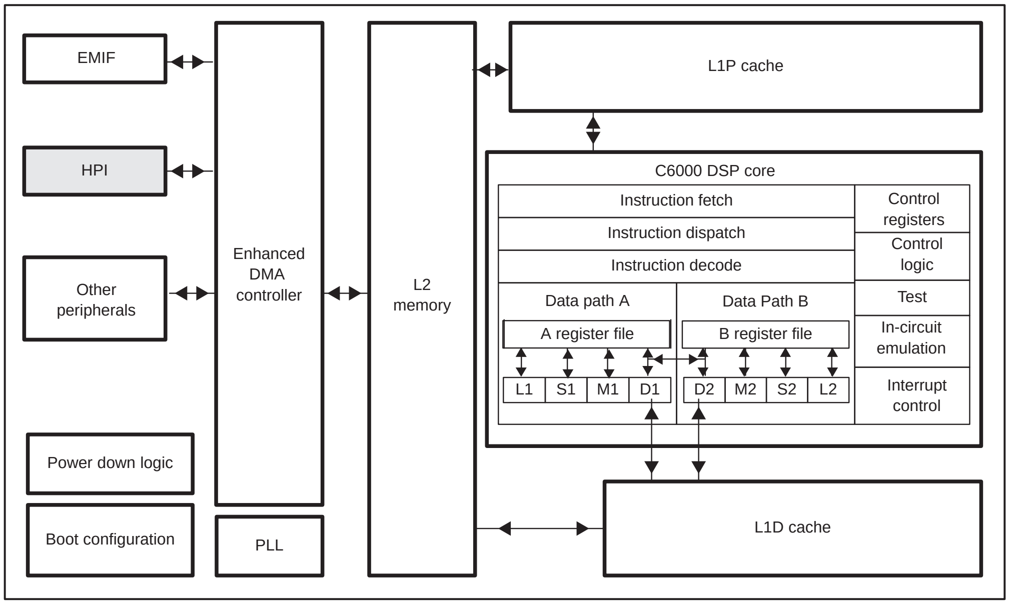
Figure 7–1 shows the host-port components in the block diagram of the C620x/C670x. Figure 7–2 shows the HPI in the C621x/C671x/C64x block diagram. Table 7–1 summarizes the differences between the C6000 HPIs.

Figure 7–1. TMS320C620x/C670x Block Diagram



Note: Refer to the specific device datasheet for its peripheral set.

Figure 7–2. TMS320C621x/C671x/C64x Block Diagram



Note: Refer to the specific device datasheet for its peripheral set.

Table 7–1. Differences Between the C62x/C67x and C64x HPI

	C62x/C67x		**C64x**		
Features	**C620x/C670x**	**C621x/C671x**	**C64x HPI16**	**C64x HPI32**	**Section**
Data Bus Width	16–bit	16–bit	16-bit	32–bit	7.2
Byte enable $\overline{\text{HBE[1:0]}}$ pins	yes	no	no	no	7.3.4
HHWIL	used	used	used	not used	7.3.3
Single halfword access support	yes	no	no	no	7.6.6
HPIA access	By host only	By host only	By host or CPU. HPIA consists of HPIAR and HPIAW.	By host or CPU. HPIA consists of HPIAR and HPIAW.	7.5.1
$\overline{\text{HRDY}}$ operation	Not–ready after each word access	Not–ready only if internal read/ write buffers not ready	Same as C621x/C671x	Same as C621x/C671x	7.4.1 7.4.2 7.4.3
Internal read buffer	no	yes, 8–deep	yes, 16–deep	yes, 16–deep	7.4.2
Internal write buffer	no	yes, 8–deep	yes, 32–deep Flushes after internal timer times out	yes, 32–deep Flushes after internal timer times out	7.4.2 7.4.3

7.2 HPI External Interface

The following sections discuss the external interface of the C6000 HPIs. See section 7.3 for a detailed description of the interface signals.

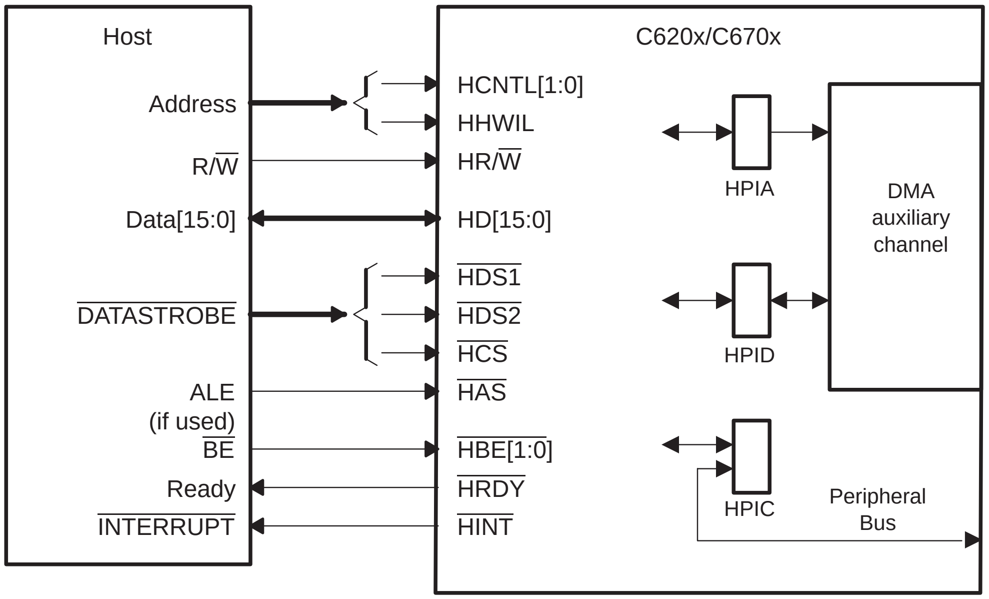
7.2.1 TMS320C620x/C670x HPI

Figure 7–3 is a simplified diagram of the C620x/C670x 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 two data strobes ($\overline{\text{HDS1}}$ and $\overline{\text{HDS2}}$), the read/write select (HR/$\overline{\text{W}}$), and the address strobe ($\overline{\text{HAS}}$) 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 $\overline{\text{HINT}}$ 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.

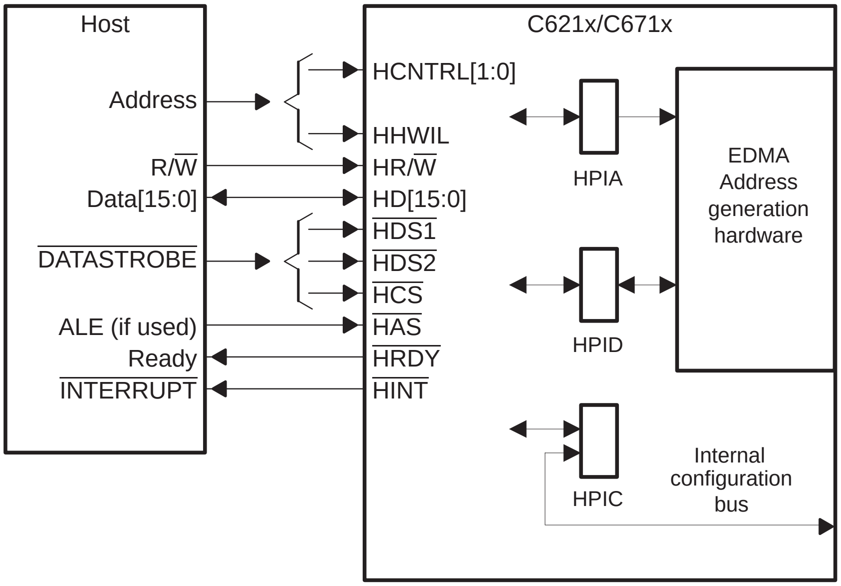
The HPI ready pin ($\overline{\text{HRDY}}$) allows insertion of host wait states. Wait states may be necessary, depending on latency to the point in the memory map accessed via the HPI, as well as on the rate of host access. The rate of host access can force not-ready conditions if the host attempts 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.

7.2.2 TMS320C621x/C671x HPI

The C621x/C671x pin interface (shown in Figure 7–4) is similar to the C620x HPI interface, except that byte enables ($\overline{\text{HBE[1:0]}}$ in C620x/C670x) are not supported. All accesses through the 16-bit data bus HD[15:0] have to be in pairs.

Unlike the C620x HPI interface, which uses the DMA auxiliary channel to perform accesses, the C621x/C671x HPI ties directly into internal address generation hardware. No specific EDMA channel is used for performing C621x/C671x HPI accesses. Instead the internal address generation hardware, which is not visible to users, handles the read/write requests and accesses.

Figure 7–4. HPI Block Diagram of TMS320C621x/C671x



7.2.3 TMS320C64x HPI16 or HPI32

As shown in Figure 7–5, the C64x has 32 external data pins HD[31:0]. As a result, the C64x HPI supports either a 16-bit or 32-bit external pin interface. The C64x HPI is called the HPI16 when operating as a 16-bit-wide host port, and it is called the HPI32 when operating as a 32-bit-wide host port. The C64x selects either the HPI16 or the HPI32 via the boot and device configuration pins at reset. See *Chapter 11, Boot Modes and Configuration*, for details.

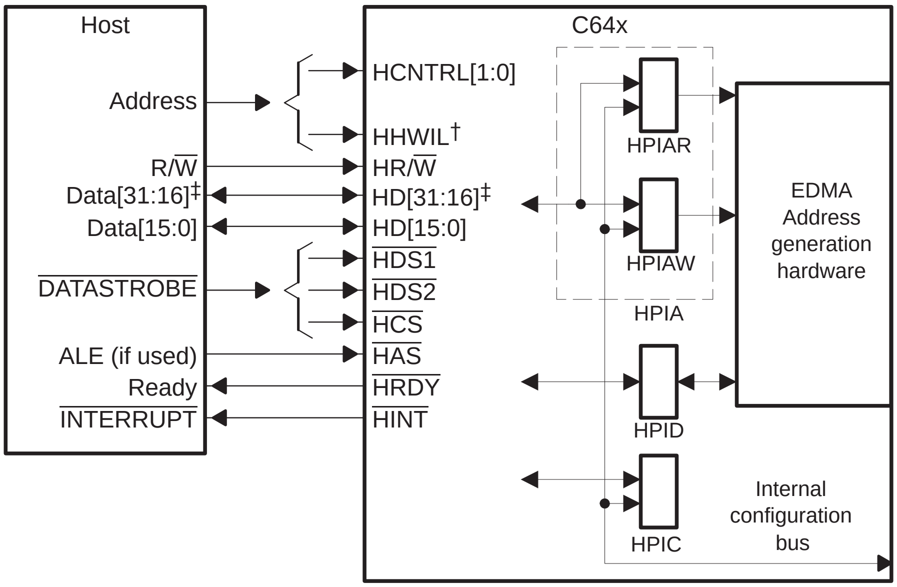
The HPI16 is an enhanced version of the C621x/C671x HPI. The HPI16 provides 32-bit data to the CPU with a 16-bit external interface. In addition to all the C621x/C671x HPI functions, the HPI16 allows the DSP to access the HPI address register HPIA. Furthermore, as shown in Figure 7–5, the HPIA is separated into two registers, HPIA Write (HPIAW) and HPIA Read (HPIAR). See section 7.5.1 for details.

The HPI32 has similar functions to the HPI16. The followings are the only differences between the HPI16 and the HPI32:

- **Data Bus Size:** As the name implies, the HPI16 has a 16-bit data bus. The HPI16 combines successive 16-bit transfers to provide 32-bit data to the CPU. For compatibility with other C6000 devices, the HPI16 uses HD[15:0] as data pins regardless of the endian mode selected at reset. The HPI32 has a 32-bit data bus. With this increased bus width, all transfers consist of one 32-bit word instead of two consecutive 16-bit halfwords. As a result, throughput is increased when the HPI operates in HPI32 mode.

- **HHWIL Input:** The HHWIL input is used on the HPI16 to identify the first or second halfword of a word transfer. The HHWIL is not used on the HPI32, as all data transfers are performed in 32-bit words.

Figure 7–5. HPI Block Diagram of TMS320C64x



† HHWIL applies to HPI16 only.
‡ HD[31:16] apples to HPI32 only.

7.3 HPI Signal Descriptions

The external HPI interface signals implement a flexible interface to a variety of host devices. Table 7–2 lists the HPI pins and their functions. The remainder of this section discusses the pins in detail.

Table 7–2. HPI External Interface Signals

Signal Name	Signal Type†	Signal Count	Host Connection	Signal Function
HD[15:0] or HD[31:0]‡	I/O/Z	16 or 32‡	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 and 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 multi-plexed 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
‡ HD[31:16] applies to C64x or HPI32 only.
§ HHWIL does not apply to C64x HPI32.
¶ $\overline{\text{HBE[1:0]}}$ applies to C620x/C670x only.

7.3.1 Data Bus: HD[15:0] or HD[31:0]

HD[15:0] or HD[31:0] is a parallel, bidirectional, 3-state data bus. HD is placed in the high-impedance state when it is not responding to an HPI read access. Pins HD[31:16] apply to the C64x HPI32 only. See section 7.2.3.

7.3.2 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 7–3 describes the HCNTL[1:0] bit functions.

Table 7–3. 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) in autoincrement mode. The HPI address register (HPIA) is postincremented by a word address (four byte address-es).
1	1	Host reads or writes to the HPI data register (HPID) in fixed address mode. HPI address register (HPIA) is not affected.

7.3.3 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 $\overline{\text{HBE[1:0]}}$ are removed from the C621x/C671x and C64x HPI, HHWIL in combination with HWOB specify the half-word position in the data register, HPID. This is shown in Table 7–4 along with the LSB address bits depending on endianness. HHWIL does not apply to the C64x HPI32.

Table 7–4. 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

7.3.4 Byte Enables: $\overline{\text{HBE[1:0]}}$ (C620x/C670x only)

On HPID writes, the value of $\overline{\text{HBE[1:0]}}$ indicates which bytes of the 32-bit word are written. On HPID writes, $\overline{\text{HBE0}}$ enables the least significant byte in the half-word and $\overline{\text{HBE1}}$ enables the most significant byte in the halfword. Table 7–5 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 halfword 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.

Note: The HPI only performs word reads. Therefore the byte enables $\overline{\text{HBE[1:0]}}$ are don't care during a host read access.

Table 7–5. Byte Enables for HPI Data Write Access (C620x/C670x only)

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

7.3.5 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.

7.3.6 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.

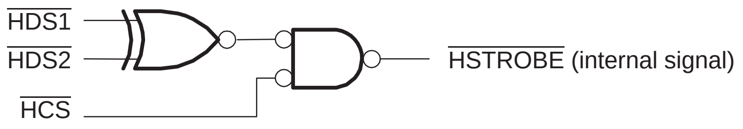
7.3.7 Strobes: $\overline{\text{HCS}}$, $\overline{\text{HDS1}}$, $\overline{\text{HDS2}}$

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

- ❑ A single strobe output with read/write select (HR/$\overline{\text{W}}$)
- ❑ Separate read and write strobe outputs. In this case, read or write select can be done by using different addresses.

Figure 7–6 shows the equivalent circuit of the $\overline{\text{HCS}}$, $\overline{\text{HDS1}}$, and $\overline{\text{HDS2}}$ inputs.

Figure 7–6. 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 an active high 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 7.3.8 for a description of $\overline{\text{HAS}}$.
- The rising edge of $\overline{\text{HSTROBE}}$ latches the $\overline{\text{HBE[1:0]}}$ input (C620x/C670x only) 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).

7.3.8 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).

7.3.9 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 signal is described in more detail in section 7.5.5. The HINT bit is set to 0 when the chip is being reset. Thus, the $\overline{\text{HINT}}$ pin is high at reset.

7.4 HPI Bus Access

7.4.1 HPI Bus Access for C620x/C670x

HPI access timing in different cases for the C620x/C670x HPI are shown in Figure 7–7, Figure 7–8, Figure 7–9, and Figure 7–10. $\overline{\text{HSTROBE}}$ represents the internally generated strobe described in Figure 7–6. Control signals: HCNTL[1:0], HR/$\overline{\text{W}}$, HHWIL, and $\overline{\text{HBE}}$[1:0] are inputs typically driven by the host. HCNTL[1:0] and HR/$\overline{\text{W}}$ should have the same values for both halfword accesses. HHWIL must be low for the first halfword transfer and high for the second. For correct operation, the host should monitor and detect HRDY low before any HPI transfer (this includes HPID, HPIA, and HPIC accesses).

7.4.1.1 Latching Control Signals

The control signals are latched differently, depending upon whether $\overline{\text{HAS}}$ is used. If $\overline{\text{HAS}}$ is tied high and not used (Figure 7–7 and Figure 7–9), the falling edge of $\overline{\text{HSTROBE}}$ latches the control signals. If $\overline{\text{HAS}}$ is used (Figure 7–8 and Figure 7–10), the falling edge of $\overline{\text{HAS}}$ latches these control signals. In the latter case, the falling edge of $\overline{\text{HAS}}$ must precede the falling edge of $\overline{\text{HSTROBE}}$.

7.4.1.2 C620x/C670x HPID Read

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, $\overline{\text{HRDY}}$ goes not–ready (high). Once data is available, $\overline{\text{HRDY}}$ goes ready (low). Data is set up at the falling edge of $\overline{\text{HRDY}}$ and held until the rising edge of $\overline{\text{HSTROBE}}$. Therefore the host should not end a read cycle ($\overline{\text{HSTROBE}}$ rising edge) until $\overline{\text{HRDY}}$ is detected ready (low).

$\overline{\text{HRDY}}$ goes not–ready (high) in one of the following conditions:

- After $\overline{\text{HCS}}$ falling edge if prefetch from the previous autoincrement address mode access has not completed (Case 1 in Figure 7–7 and Figure 7–8)
- After $\overline{\text{HSTROBE}}$ falling edge if $\overline{\text{HAS}}$ is not used and valid data is not in HPID (Case 2 in Figure 7–7 and Figure 7–8)
- After $\overline{\text{HAS}}$ falling edge if $\overline{\text{HAS}}$ is used, and valid data is not in HPID (Case 2 in Figure 7–7 and Figure 7–8)
- After $\overline{\text{HSTROBE}}$ rising edge if the read access is in autoincrement address mode

Fixed Address Mode HPID Read (HCNTL[1:0] = 11b)

For a fixed address mode read, the HPI sends the read request to the DMA auxiliary channel and $\overline{\text{HRDY}}$ becomes not–ready (high). $\overline{\text{HRDY}}$ remains not–ready (high) until the requested data is loaded into HPID. Since the DMA auxiliary channel performs word reads, the data is already present in the HPID at the beginning of the second halfword read access. Thus, the second halfword HPID read never encounters a not–ready condition, and $\overline{\text{HRDY}}$ remains low.

Autoincrement Address Mode HPID Read (HCNTL[1:0] = 10b)

For the first autoincrement address mode read, the HPI sends the read request to the DMA auxiliary channel and $\overline{\text{HRDY}}$ goes not–ready, as shown in Case 2 in Figure 7–7 and Figure 7–8. The data pointed to by the next address is fetched immediately upon completion of the current read. $\overline{\text{HRDY}}$ becomes not–ready (high) when the HPI is busy pre–fetching data.

7.4.1.3 C620x/C670x HPID Write

On a write, the host must set up data and $\overline{\text{HBE}}$[1:0] on the rising edge of $\overline{\text{HSTROBE}}$. The C620x/C670x HPI provides 32–bit data to the CPU through a 16–bit external interface by automatically combining two successive halfword transfers.

During an HPID write access, two halfword portions of the HPID are transferred from the host. At the end of this write access, (with the second rising edge of $\overline{\text{HSTROBE}}$), the contents of HPID are transferred as a 32–bit word to the address specified by HPIA.

The host should not end (with an $\overline{\text{HSTROBE}}$ rising edge) a write cycle until $\overline{\text{HRDY}}$ is detected ready (low). $\overline{\text{HRDY}}$ goes not–ready (high) in one of the following conditions:

- After $\overline{\text{HCS}}$ falling edge if the previous write access has not yet completed
- After $\overline{\text{HSTROBE}}$ rising edge to service the HPID write

See Figure 7–9 and Figure 7–10 for details on HPID write.

7.4.1.4 C620x/C670x HPIC or HPIA Access

For correct operation, a HPIC or HPIA write must occur before switching from HPID read to HPID write, or vice versa. The host should monitor and detect $\overline{\text{HRDY}}$ low before reading or writing to the HPIA/HPIC.

Figure 7–7. HPI Read Timing (HAS Not Used, Tied High)

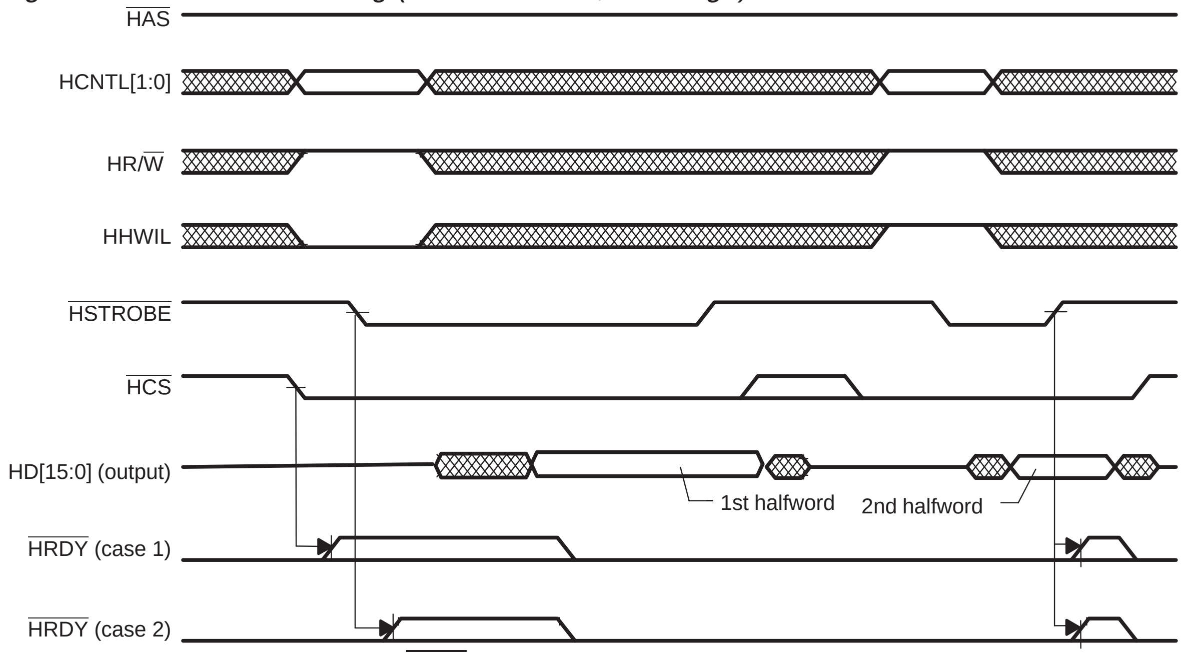
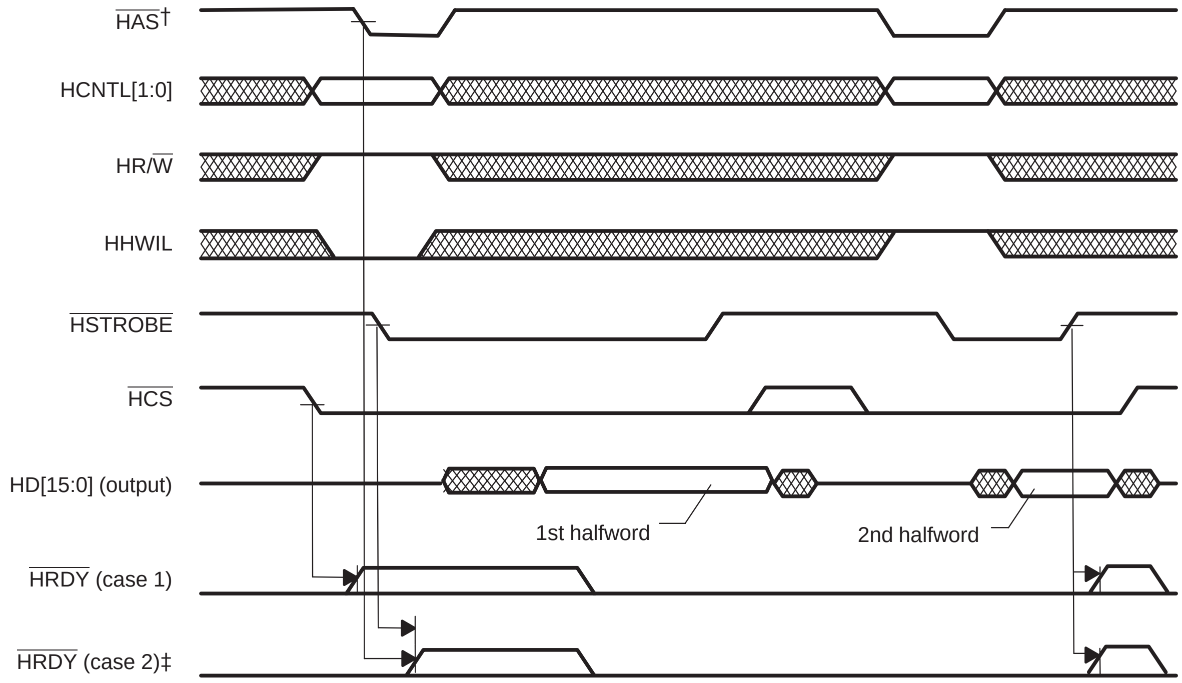


Figure 7–8. HPI Read Timing (HAS Used)



† For correct operation, strobe the HAS signal only once per HSTROBE cycle.
‡ For C620x/C670x, if HAS is used, HRDY goes not ready after HAS falling edge. For all other devices, HRDY goes not–ready after HSTROBE falling edge, even if HAS is used.

Figure 7–9. HPI Write Timing ($\overline{\text{HAS}}$ Not Used, Tied High)

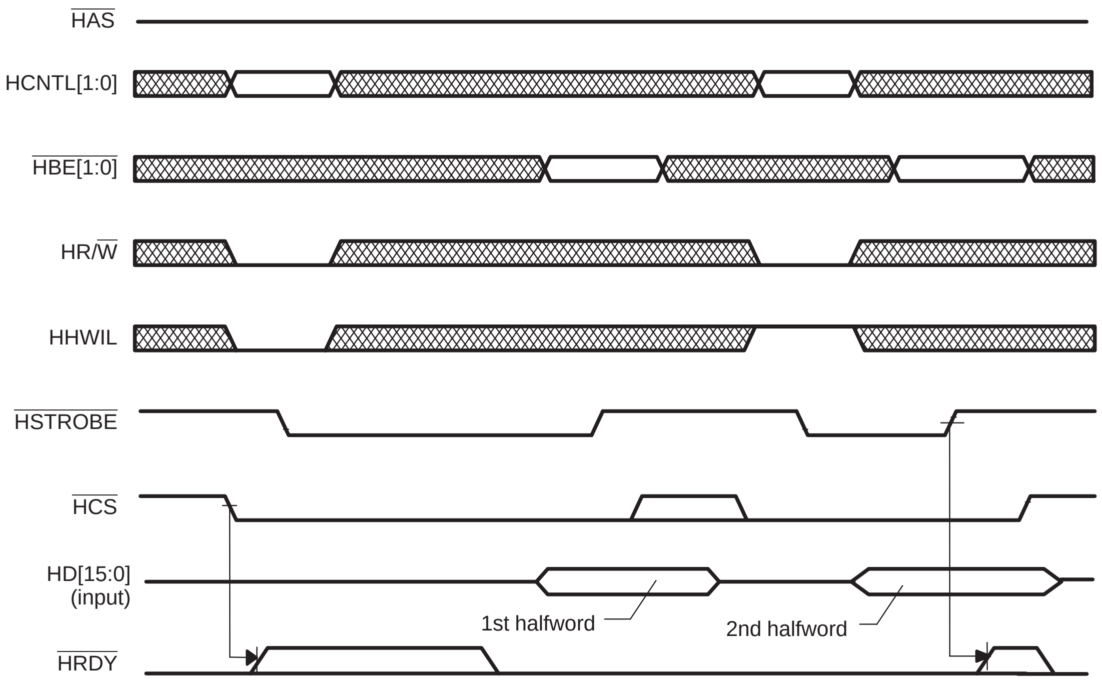
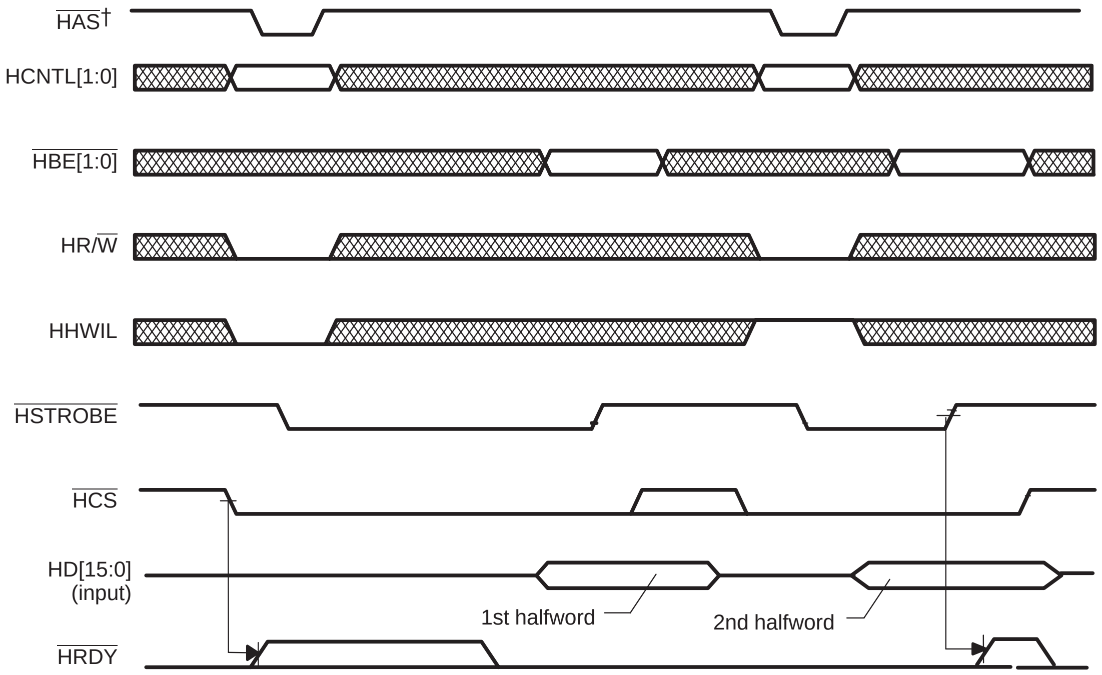


Figure 7–10. HPI Write Timing ($\overline{\text{HAS}}$ Used)



† For correct operation, strobe the $\overline{\text{HAS}}$ signal only once per $\overline{\text{HSTROBE}}$ cycle.

7.4.2 HPI Bus Access for C621x/C671x

The C621x/C671x HPI shares the same bus interface as the C620x/C670x HPI. In addition, the C621x/C671x HPI has internal read and write buffers to improve throughput in both read and write accesses. For correct operation, the host should monitor and detect $\overline{\text{HRDY}}$ low before any HPI transfer (this includes HPID, HPIA, and HPIC accesses).

7.4.2.1 Latching Control Signals

Similar to the C620x/C670x, the C621x/C671x HPI latches control signals differently depending upon whether $\overline{\text{HAS}}$ is used. See section 7.4.1.1.

7.4.2.2 C621x/C671x HPI Read

Fixed Address Mode HPID Read (HCNTL[1:0] = 11b)

For a fixed address mode read, the HPI sends the read request to the EDMA internal address generation hardware and $\overline{\text{HRDY}}$ becomes not–ready (high). This is shown in Case 2 in Figure 7–7 and Figure 7–8. $\overline{\text{HRDY}}$ remains not–ready until the requested data is loaded into HPID. Since the EDMA internal address generation hardware performs word reads, the data is already present in the HPID at the beginning of the second halfword read access. Thus, the second halfword HPID read never encounters a not–ready condition, and $\overline{\text{HRDY}}$ remains low.

Autoincrement Address Mode HPID Read (HCNTL[1:0] = 10b)

The C621x/C671x HPI has an internal read buffer that helps to improve throughput in autoincrement mode. For the first host read of the HPID, the HPI deasserts $\overline{\text{HRDY}}$ high (not ready) after $\overline{\text{HSTROBE}}$ falling edge, as shown in Case 2 in Figure 7–7 and Figure 7–8. During this time, the EDMA internal address generation hardware begins fetching enough words to fill the internal read buffer. As soon as the first valid data is ready, the HPI presents it on the HD bus and $\overline{\text{HRDY}}$ goes ready. If the next data is already present in the internal read buffer when the current read finishes, $\overline{\text{HRDY}}$ remains ready at the beginning of the next host read ($\overline{\text{HSTROBE}}$ falling edge). The HPI continues to perform subsequent data fetches and places the data in the internal read buffer before the actual host read occurs. This reduces the subsequent host read data access time. A read cycle is terminated by a host write to the HPIA or HPIC, at which point the HPI automatically empties the internal read buffer, and $\overline{\text{HRDY}}$ goes not–ready. Note that a read cycle does not have to be terminated immediately when a read burst completes. The normal process of reinitializing the HPIA register for a new address range will force the internal read buffer to flush in anticipation of a new read command.

For all modes of HPI read, the host should not end ($\overline{\text{HSTROBE}}$ rising edge) a read cycle until $\overline{\text{HRDY}}$ is detected ready (low). $\overline{\text{HRDY}}$ goes not–ready (high) in one of the following conditions:

- After $\overline{\text{HCS}}$ falling edge
- After $\overline{\text{HSTROBE}}$ falling edge for first halfword transfer (HHWIL low)
- After $\overline{\text{HSTROBE}}$ rising edge for second halfword transfer (HHWIL high).

7.4.2.3 C621x/C671x HPID Write

On a write, the host must set up data on the rising edge of $\overline{\text{HSTROBE}}$. All C621x/C671x writes must consist of two successive halfword transfers.

Fixed Address Mode HPID Write (HCNTL[1:0] = 11b)

This operation is identical to the C620x/C670x fixed address mode HPID write. See section 7.4.1.3 for details.

Autoincrement Address Mode HPID Write (HCNTL[1:0] = 10b)

The C621x/C671x HPI has an internal write buffer that helps to improve throughput in autoincrement mode. At the end of a word write in autoincrement mode, the data is copied from the HPID to the internal write buffer to wait for service by the EDMA internal address generation hardware. The DSP does not actually service this host write until the internal write buffer is half full, or if the write cycle is terminated. A write cycle is terminated by a host access to the HPIA or the HPIC, at which point the DSP services the HPI by transferring all remaining elements from the internal write buffer to their destinations.

Since the data is copied to the internal write buffer, the HPID is immediately ready for the next data write from the host. Thus under normal conditions, $\overline{\text{HRDY}}$ remains ready at the beginning of the next host write access ($\overline{\text{HSTROBE}}$ active).

The host should not end (with an $\overline{\text{HSTROBE}}$ rising edge) a write cycle until $\overline{\text{HRDY}}$ is detected ready (low). $\overline{\text{HRDY}}$ goes not–ready (high) in one of the following conditions:

- After $\overline{\text{HCS}}$ falling edge
- After $\overline{\text{HSTROBE}}$ rising edge for second halfword transfer (HHWIL high)
- After $\overline{\text{HSTROBE}}$ falling edge for first halfword transfer (HHWIL low) if the internal write buffer is full.

7.4.2.4 C621x/C671x HPIC or HPIA Access

For correct operation, a HPIC or HPIA write must occur before switching from HPID read to HPID write, or vice versa. A HPIC or HPIA register write terminates a burst read/write access in autoincrement mode. $\overline{\text{HRDY}}$ may go not–ready while the internal read buffer is being emptied (for reads), or while the internal write buffer is being serviced by the DSP (for writes).

7.4.3 HPI Bus Access for C64x

With a 32–bit data bus, the C64x HPI is an enhanced version of the C621x/C671x HPI. The C64x HPI can be configured at reset to operate in either HPI16 or HPI32 mode. See *Chapter 11 Boot Modes and Configuration*.

The HPI16 operation is similar to the C621x/C671x 16–bit HPI. See section 7.4.2. In addition to the operation described in section 7.4.2, the C64x HPI internal write buffer will flush when an internal timer times out after 128 CPU clock cycles.

The HPI32 operation is similar to the HPI16 operation, with exceptions due to the expanded 32–bit data bus. Since the HPI32 data bus is expanded to 32 bits, all read and write transfers consist of one 32–bit word access (HD[0:31]) instead of two consecutive 16–bit halfword accesses. Also, the HHWIL signal is not used on the HPI32 interface. The HPI32 read and write timings are shown in Figure 7–11, Figure 7–12, Figure 7–13, and Figure 7–14. The remaining HPI32 operations are identical to the HPI16.

Figure 7–11. HPI32 Read Timing ($\overline{\text{HAS}}$ Not Used, Tied High) for C64x only

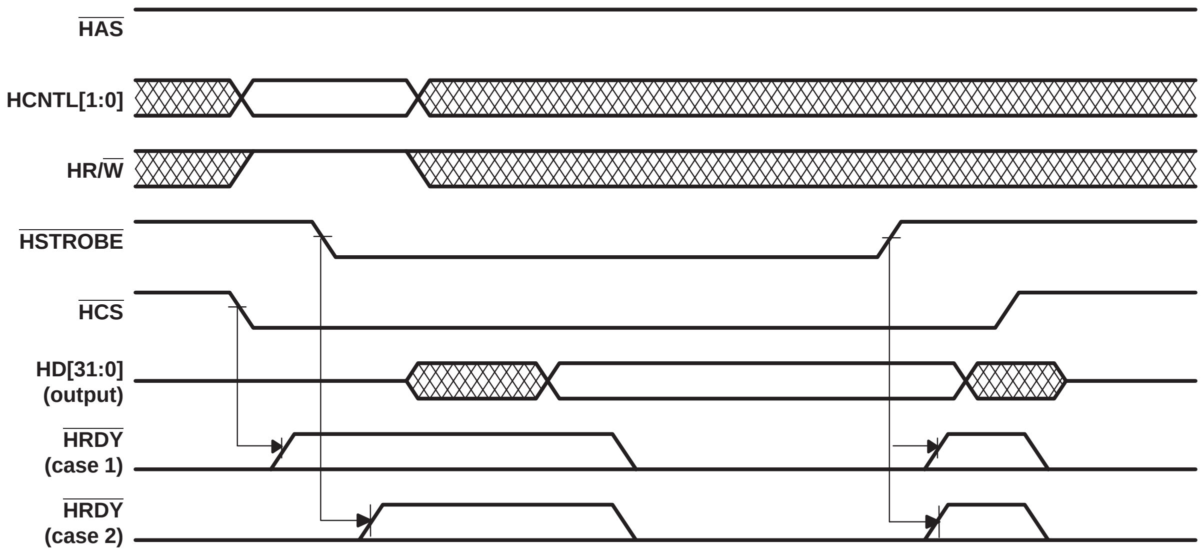
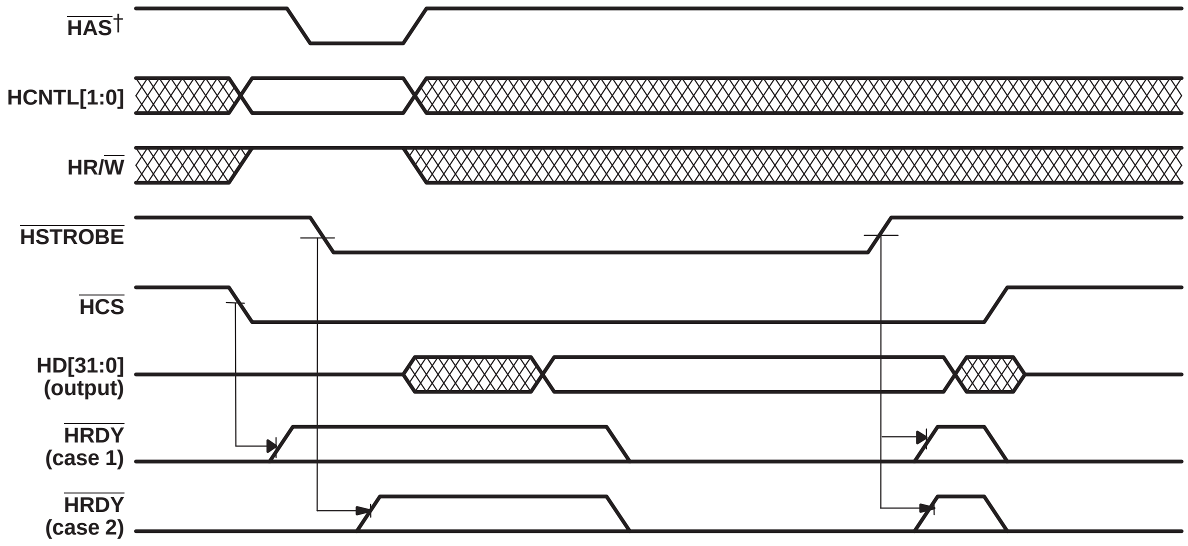


Figure 7–12. HPI32 Read Timing ($\overline{\text{HAS}}$ Used) for C64x only



† For correct operation, strobe the $\overline{\text{HAS}}$ signal only once per $\overline{\text{HSTROBE}}$ cycle.

Figure 7–13. HPI Write Timing ($\overline{\text{HAS}}$ Not Used, Tied High) for C64x only

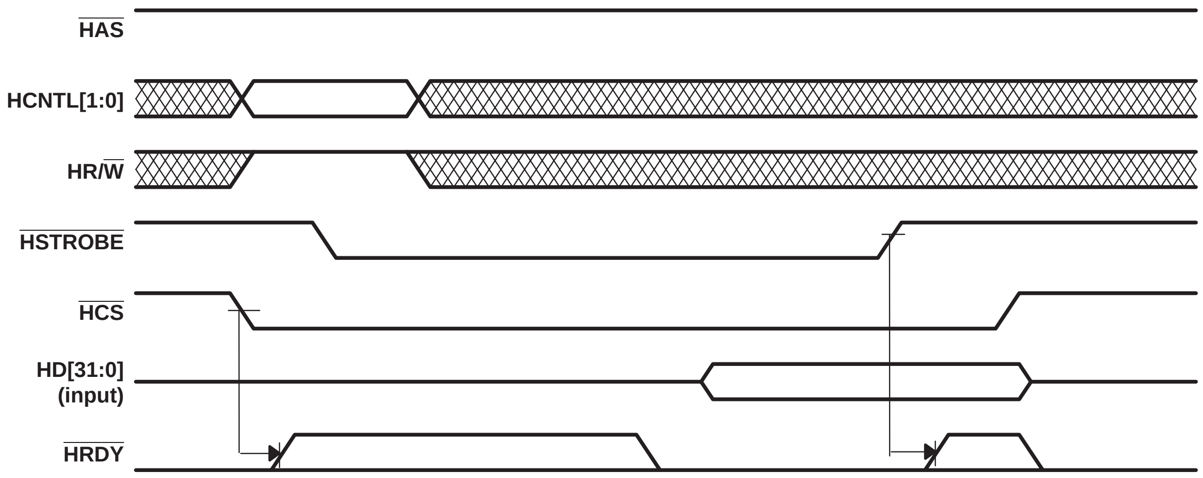
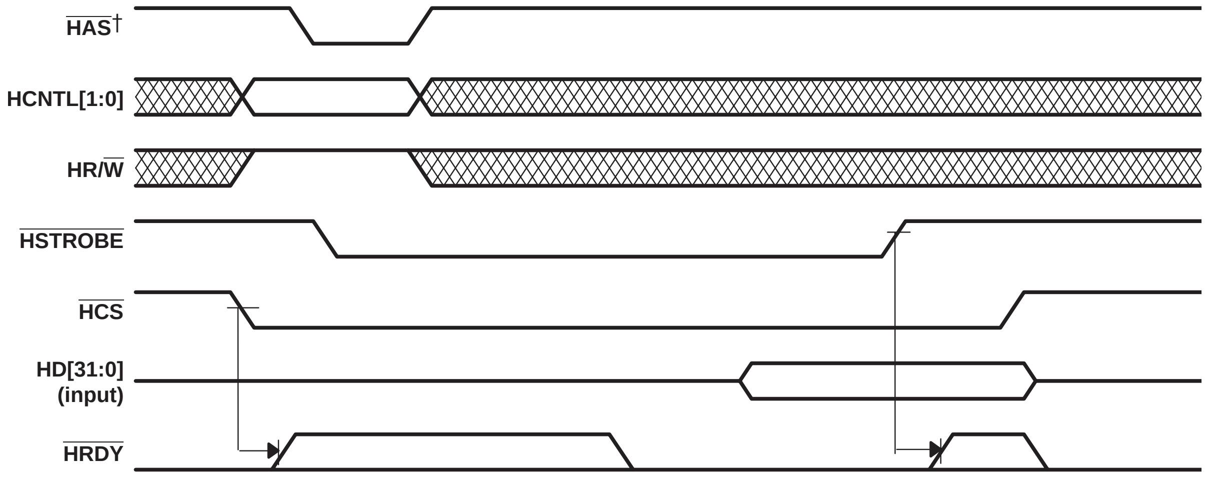


Figure 7–14. HPI Write Timing ($\overline{\text{HAS}}$ Used) for C64x only



† For correct operation, strobe the $\overline{\text{HAS}}$ signal only once per $\overline{\text{HSTROBE}}$ cycle.

7.5 HPI Registers

The registers that the HPI uses for communication between the host device and the CPU are summarized in Table 7–6 (C62x/C67x) and Table 7–7 (C64x). 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. The HPIA and HPIC are discussed in detail in the following sections.

Table 7–6. HPI Registers for C62x/C67x

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

Table 7–7. HPI Registers for C64x

Register Abbreviation	Register Name	Host Read/Write Access	CPU Read/Write Access	CPU Read/Write (Hex Byte Address)
HPID	HPI data	RW	–	–
HPIC	HPI control	RW	RW	01880000h
HPIA (HPIAW)†	HPI address (write)	RW	RW	0188 0004h
HPIA (HPIAR)†	HPI address (read)	RW	RW	0188 0008h

† Host access to the HPIA updates both HPIAW and HPIAR. The CPU can access HPIAW and HPIAR independently.

7.5.1 HPI Address Register (HPIA)

The HPIA contains the address of the memory accessed by the HPI at which the current access occurs. This address is a 30-bit word address, so the two LSBs are unaffected by HPIA writes and are always read as 0. The C62x/C67x HPIA register is only accessible by the host. It is not mapped to the DSP memory.

The C64x HPIA register is accessible by both the host and the CPU. Furthermore, the HPIA register is separated into two registers internally: the HPI address write register (HPIAW), and the HPI address read register (HPIAR). By separating the HPIA into HPIAW and HPIAR internally, the CPU can update the read and write memory address independently to allow the host to perform read and write to different address ranges.

For the C64x, a host access to the HPIA register is identical to the operation of the C62x/C67x HPI. The HCNTL[1:0] control bits are set to 01b to indicate an access to the HPIA register. A host write to the HPIA register updates *both* the HPIAW and HPIAR internally. A host read of the HPIA returns the value in the most recently used HPIAx register. For example, if the most recent HPID access was a read, then an HPIA read by the external host will return the value in HPIAR. If the most recent HPID access was a write, then an HPIA read by the external host will return the value in HPIAW.

Systems which update HPIAR/HPIAW internally via the CPU must not allow HPIA updates via the external bus and vice versa. The HPIAR/HPIAW registers can be read independently by both the CPU and the external host. The system must not allow HPID accesses via the external host while the DSP is updating the HPIAR/W registers internally. This can be controlled by any conveninet means, including the use of general–purpose input/output pins to perform handshaking between the host and the DSP.

7.5.2 HPI Control Register (HPIC)

The HPIC register, shown in Figure 7–15 and summarized in Table 7–8, 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 7–15. HPIC Register

31	30 24	23	22 21	20	19	18	17	16
rsvd†	rsvd	rsvd†	rsvd	FETCH	HRDY	HINT	DSPINT	HWOB
HRW,CRW,+0	HR,CR,+0	HRW,CRW,+0	HR,CR,+0	HRW,CR,+0	HR,CR,+0	HR,CR,+0	HRW,CR,+0	HRW,CR,+0

15	14 8	7	6 5	4	3	2	1	0
rsvd†	rsvd	rsvd†	rsvd	FETCH	HRDY	HINT	DSPINT	HWOB
HRW,CRW,+0	HR,CR,+0	HRW,CRW,+0	HR,CR,+0	HRW,CR,+0	HR,CR,+1	HR,CRW,+0	HRW,CRW,+0	HRW,CR,+0

† For C62x/C67x, bits 7, 15, 23, 31 are read-only; HR,CR,+0. For C64x, bits 7, 15, 23, and 31 are writable fields and must be written with 0. Otherwise, operation is undefined.

Table 7–8. 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. For HPI32, HWOB is not used and the value of HWOB is irrelevant.	7.6
DSPINT	The host processor-to-CPU/DMA interrupt	7.5.4
HINT	DSP-to-host interrupt. The inverted value of this bit determines the state of the CPU $\overline{\text{HINT}}$ output.	7.5.5
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.	7.5.3
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.	7.5.3

7.5.3 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.

7.5.4 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 the user programs the selection of the DSPINT interrupt with interrupt selector, the CPU detects the transition of DSPINT as an interrupt condition. Unlike a host write, a CPU write of DSPINT = 1 when DSPINT = 0 has no effect. The CPU can clear the DSPINT bits by writing a 1 to DSPINT when DSPINT = 1. Neither a host, nor a CPU write of DSPINT = 0, affects the DSPINT bit or signal in any case.

7.5.5 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 half-word reads by the host can yield different data if the CPU changes the state of this bit between the two read operations.

7.6 Host Access Sequences

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

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 in the C620x/C670x or the internal address generation hardware in the C621x/C671x/C64x. For the 16-bit HPI, 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 must 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. The C64x HPI32 combines two halfword transfers into a single-word transfer.

7.6.1 Initialization of HPIC and HPIA

Before any data access, the HPIC and HPIA must be initialized. On the C62x/C67x, only the host has access to the HPIA register. On the C64x, either the host or the CPU can be used to initialize the HPIC and HPIA registers. The following sections discuss the host initialization sequence for the 16-bit-wide host port (C62x/C67x HPI and C64x HPI16) and the 32-bit-wide host port (HPI32) respectively.

7.6.1.1 Initialization of HPIC and HPIA – C62x/C67x HPI and C64x HPI16

Before accessing data, the HWOB bit of the HPIC register and the HPIA must be initialized (in this order, because HWOB affects the HPIA access). After initializing HWOB, the host (or the CPU for C64x) can write to HPIA with the correct halfword alignment. Table 7–9 and Table 7–10 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 7–9. 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.
For the C64x, a host write to HPIA updates both HPIAR and HPIAW internally.

Table 7–10. 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.
For the C64x, a host write to HPIA updates both HPIAR and HPIAW internally.

7.6.1.2 Initialization of HPIC and HPIA — HPI32

For the HPI32, either the host or the CPU can be used to initialize the HPIC and HPIA. All accesses are 32-bit wide. The HWOB bit in the HPIC is not used. Therefore it may not be necessary to initialize the HPIC if the default value is desired. Table 7–11 summarizes the HPIC and HPIA initialization sequence for HPI32.

Table 7–11. Initialization of HPIC and HPIA

	Value During Access			Value After Access		
Event	**HD**	**HR/$\overline{\text{W}}$**	**HCNTL[1:0]**	**HPIC**	**HPIA**	**HPID**
Host writes HPIC	00000000	0	00	00080008	????????	????????
Host writes HPIA	80001234	0	01	00080008	80001234	????????

Note: The “?” in this table indicate the value is unknown.

7.6.2 HPID Read Access in Fixed Address Mode

Assume that once the HPI is initialized, the host wishes to perform a read access to an address in fixed address mode. Assume that the host wants to read the word at address 80001234h and that the word value at that location is 789ABCDEh. The following sections discuss a HPID read access in fixed address mode for the 16-bit-wide host port (C62x/C67x HPI and C64x HPI16) and the 32-bit-wide host port (HPI32), respectively.

7.6.2.1 HPID Read in Fixed Address Mode — C62x/C67x HPI and C64x HPI16

The host must read the 32-bit HPID in two 16-bit halfwords. Table 7–12 and Table 7–13 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 (C620x/C670x) or the internal address generation hardware (C621x/C671x/C64x). 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 requested data is loaded into HPID. Because all internal 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 7–12. Data Read Access to HPI in Fixed Address Mode: 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 half-word Data not ready	????	xx	1	11	1	0	00010001	80001234	????????
Host reads HPID 1st half-word 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: The “?” in this table indicate the value is unknown.

Table 7–13. Data Read Access to HPI in Fixed Address Mode: 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 half-word Data not ready	????	xx	1	11	1	0	00000000	80001234	????????
Host reads HPID 1st half-word Data ready	789A	xx	1	11	0	0	00080008	80001234	789ABCDE
Host reads HPID 2nd half-word	BCDE	xx	1	11	0	1	00080008	80001234	789ABCDE

Note: The “?” in this table indicate the value is unknown.

7.6.2.2 HPID Read in Fixed Address Mode — HPI32

The host access sequence to the HPID of the HPI32 is similar to the sequence for the HPI16. The difference is that an HPI32 host access is done in one 32-bit word instead of two 16-bit halfwords. Table 7–14 shows an example of this read access in fixed address mode. In this example, the host reads the word at address 80001234h, which has a value of 789ABCDEh.

Table 7–14. Data Read Access in Fixed Address Mode for HPI32

	Value During Access				Value After Access		
Event	**HD**	**HR/$\overline{\text{W}}$**	**HCNTL[1:0]**	**$\overline{\text{HRDY}}$**	**HPIC**	**HPIA**	**HPID**
Host reads HPIC Data not ready	????????	1	11	1	00000000	80001234	????????
Host writes HPID Data ready	789ABCDE	1	11	0	00080008	80001234	789ABCDE

Note: The "?" in this table indicate the value is unknown.

7.6.3 HPID Read Access in Autoincrement Mode

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. For the C62x/C67x. 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.

7.6.3.1 HPID Read in Autoincrement Mode – C62x/C67x HPI and C64x HPI16

Table 7–15 summarizes a read access with autoincrement. After the first halfword 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. For the C62x/C67x, prefetching begins on the rising edge of $\overline{\text{HSTROBE}}$ in the second halfword read. The C64x HPI has an internal read buffer that allows prefetching to occur to fill the internal buffer upon the first HPID read access ($\overline{\text{HSTROBE}}$ falling edge). See section 7.4.2.2 for details.

Table 7–15. 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

† For the C64x, prefetch occurs immediately after the first host read request. (Row 1 in this table).

Note: The “?” in this table indicate the value is unknown.

Table 7–16. 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

† For the C64x, prefetch occurs immediately after the first host read request. (Row 1 in this table).

Note: The “?” in this table indicate the value is unknown.

7.6.3.2 HPID Read in Autoincrement Mode – HPI32

Table 7–17 summarizes a read access with autoincrement for the HPI32. In autoincrement mode, the first HPID read access causes the HPI to not only fetch the current data, but also to prefetch extra data to fill the internal read buffer. This throughput improvement internal read buffer is discussed in section 7.4.2.2.

Table 7–17. Read Access to HPI with Autoincrement for HP132

	Value During Access				Value After Access		
Event	**HD**	**HR/$\overline{\text{W}}$**	**HCNTL[1:0]**	**$\overline{\text{HRDY}}$**	**HPIC**	**HPIA**	**HPID**
Host reads HPID Data not ready. Prefetch data to fill internal read buffer	????????	1	10	1	00000000	80001234	????????
Host reads HPID Data ready	789ABCDE	1	10	0	00080008	80001234	789ABCDE
Address autoin-crement. Next data ready	????????	?	??	0	00080008	80001238	87654321

Note: The “?” in this table indicate the value is unknown.

7.6.4 Host Data Write Access Without Autoincrement

7.6.4.1 HPID Write in Fixed Address Mode – C62x/C67x and C64x HPI16

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 7–18 and Table 7–19 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. For the C620x/C670x the $\overline{\text{HBE[1:0]}}$ pair is enabled only for the transfer of the 16 LSBs. For the C621x, C671x and C64x HPI16, the $\overline{\text{HBE[1:0]}}$ pins do not exist. Only word writes are allowed, and all 16-bit write accesses must be made in pairs. The entire 32-bit word is transferred.

Table 7–18. Data Write Access to HPI in Fixed Address Mode: 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

† For C620x/C670x HPI, wxyz represents a "don't care" value on the HD pins. The $\overline{\text{HBE[1:0]}}$ value indicates that only 16-bit is transferred. For C621x/C671x and C64x HPI, however, wxyz should be 0000 on the HD pins. The entire 32-bit word is transferred.

Note: The "?" in this table indicate the value is unknown.

Table 7–19. Data Write Access to HPI in Fixed Address Mode: 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

† For C620x/C670x HPI, wxyz represents a "don't care" value on the HD pins. The $\overline{\text{HBE[1:0]}}$ value indicates that only 16-bit is transferred. For C621x/C671x and C64x HPI, however, wxyz should be 0000 on the HD pins. The entire 32-bit word is transferred.

Note: The "?" in this table indicate the value is unknown.

7.6.4.2 HPID Write in Fixed Address Mode — HPI32

HPID write of the HPI32 is similar to the HPI16. However, the host can write to the HPID register in one 32-bit write access. Table 7–20 summarizes an HPID write access for HPI32 in fixed address mode.

Table 7–20. Data Write Access to HPI in Fixed Address Mode for HPI32

	Value During Access				Value After Access			
Event	**HD**	**HR/$\overline{\text{W}}$**	**HCNTL[1:0]**	**$\overline{\text{HRDY}}$**	**HPIC**	**HPIA**	**HPID**	**Location 80001234**
Host writes HPID Waiting for previous access to complete.	00005566	0	11	1	00000000	80001234	????????	00000000
Host writes HPID. Ready	00005566	0	11	0	00080008	80001234	00005566	00000000
Waiting for access to complete.	????????	?	??	0	00080008	80001234	87654321	00005566

Note: The “?” in this table indicate the value is unknown.

7.6.5 HPID Write Access in Autoincrement Mode

7.6.5.1 HPID Write in Autoincrement Mode – C62x/C67x HPI and C64x HPI16

Table 7–21 and Table 7–22 summarize a host data write with autoincrement for HWOB = 1 and HWOB = 0, respectively. These examples are identical to the ones in section 7.6.4, except for the HCNTL[1:0] value and a subsequent write 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.

For the C64x HPI in autoincrement mode, data written by the host is immediately copied from the HPID to the internal write buffer. Therefore if the internal write buffer is not full, $\overline{\text{HRDY}}$ remains ready, and row 4 and 7 in Table 7–21 and Table 7–22 do not apply. In addition, the DSP only services the HPI write access in autoincrement mode when the internal write buffer is half full, or when the write cycle is terminated. Locations 80001234h and 80001238h in Table 7–16 and Table 7–17 do not get updated to the correct values (00005566h, 33000000h) until the internal write buffer is serviced.

Table 7–21. 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

† For C620x/C670x HPI, wxyz, rs, and nopq represent don't care values on the HD pins. For C621x/C671x, and C64x HPI, however, wxyz+0000, rs=00, and nopq=0000 on the HD pins. The entire 32-bit word is transferred.

Note: The "?" in this table indicate the value is unknown.

Table 7–22. 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

† For C620x/C670x HPI, wxyz, rs, and nopq represent don't care values on the HD pins. For C621x/C671x, and C64x HPI, however, wxyz+0000, rs=00, and nopq=0000 on the HD pins. The entire 32-bit word is transferred.

Note: The "?" in this table indicate the value is unknown.

7.6.5.2 HPID Write in Autoincrement Mode – HPI32

As described in section 7.6.5.1, data written in autoincrement mode by the C64x HPI host is immediately copied from the HPID to the internal write buffer. Therefore if the internal write buffer is not full, $\overline{\text{HRDY}}$ remains ready. The DSP only services the HPI write access in autoincrement mode when the internal write buffer is half full, or when the write cycle is terminated. Locations 80001234h and 80001238h in Table 7–21 and Table 7–22 do not get updated to the correct values (00005566h, 33000000h) until the internal write buffer is serviced. Table 7–23 summarizes a HPID write in autoincrement mode for the HPI32.

Table 7–23. Write Access to HPI with Autoincrement: HPI32

	Value During Access					Value After Access				
Event	**HD**	**HR/$\overline{\text{W}}$**	**HCNTL[1:0]**	**$\overline{\text{HRDY}}$**	**HHWIL**	**HPIC**	**HPIA**	**HPID**	**Location 80001234**†	**Location 80001238**†
Host writes HPID Waiting for previous access to complete	00005566	0	10	1	0	00000000	80001234	????????	00000000	00000000
Host writes HPID Ready	00005566	0	10	0	0	00080008	80001234	00005566	00000000	00000000
Host writes HPID Ready	33000000	0	10	0	1	00080008	80001238	33000000	00000000	00000000

† Location 80001234h and 80001238h do not get updated until the HPI internal write buffer is serviced. This occurs when the internal write buffer is half full, or when the write cycle terminates.

‡ The data in HPID is immediately copied to the internal write buffer if it is not full. Therefore, $\overline{\text{HRDY}}$ is ready.

Note: The “?” in this table indicate the value is unknown.

7.6.6 Single Halfword Cycles (C620x/C670x only)

In normal operation, every transfer must consist of two halfword accesses. However, the C620x/C670x HPI allows single halfword accesses to speed up operation. These can be useful in performing the following tasks:

❑ Writes to and reads from HPIC: In Table 7–9, 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 7–9, 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.

7.7 HPI Transfer Priority Queue — TMS320C621x/C671x/C64x

All C621x/C671x HPI transfers are placed in the high priority transfer queue, Q1. All C64x HPI transfers are placed in the medium priority queue, Q2. Refer to *Chapter 6 EDMA Controller,* Section 6.17 *Resource Arbitration and Priority Processing*, for details on transfer priority.

7.8 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) upon the rising edge of the $\overline{\text{RESET}}$ signal. Although the device is not in reset during these boot modes, the CPU itself is in reset until the boot completes. See *Chapter 11, Boot Modes and Configuration*, for more details.

Chapter 8

Expansion Bus

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

Topic **Page**

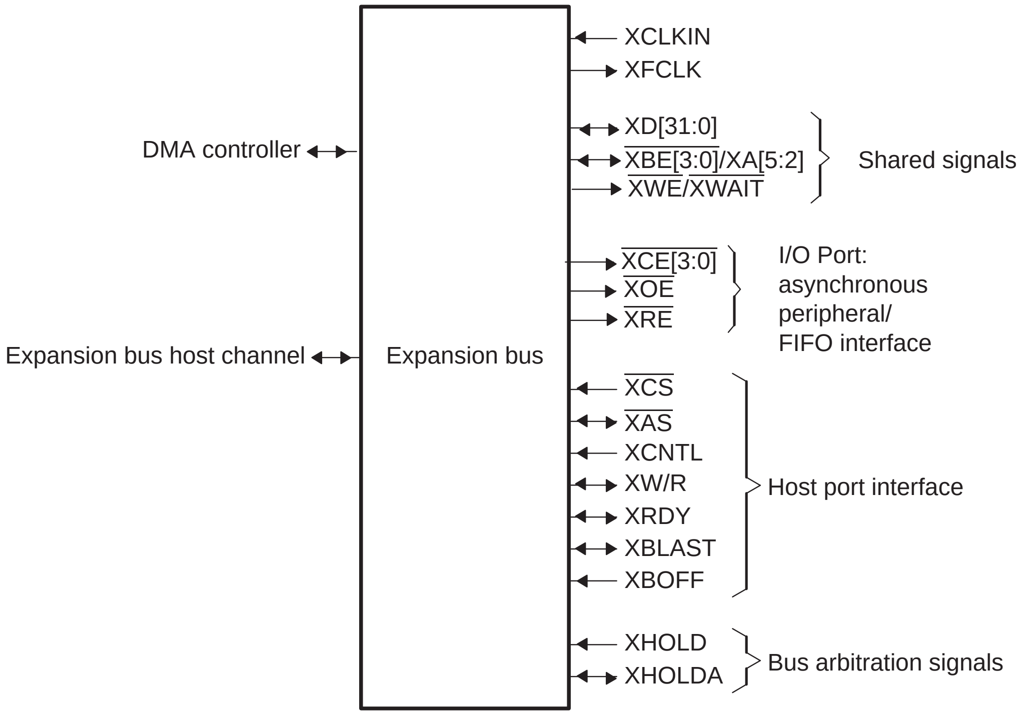
8.1 Overview

The expansion bus (XBUS) 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 XBUS offers a flexible bus arbitration scheme, implemented with two signals, XHOLD and XHOLDA. The XBUS 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 XBUS has two major sub blocks—the I/O port and host port interface. A block diagram of the XBUS is shown in Figure 8–1.

Figure 8–1. Expansion Bus Block Diagram

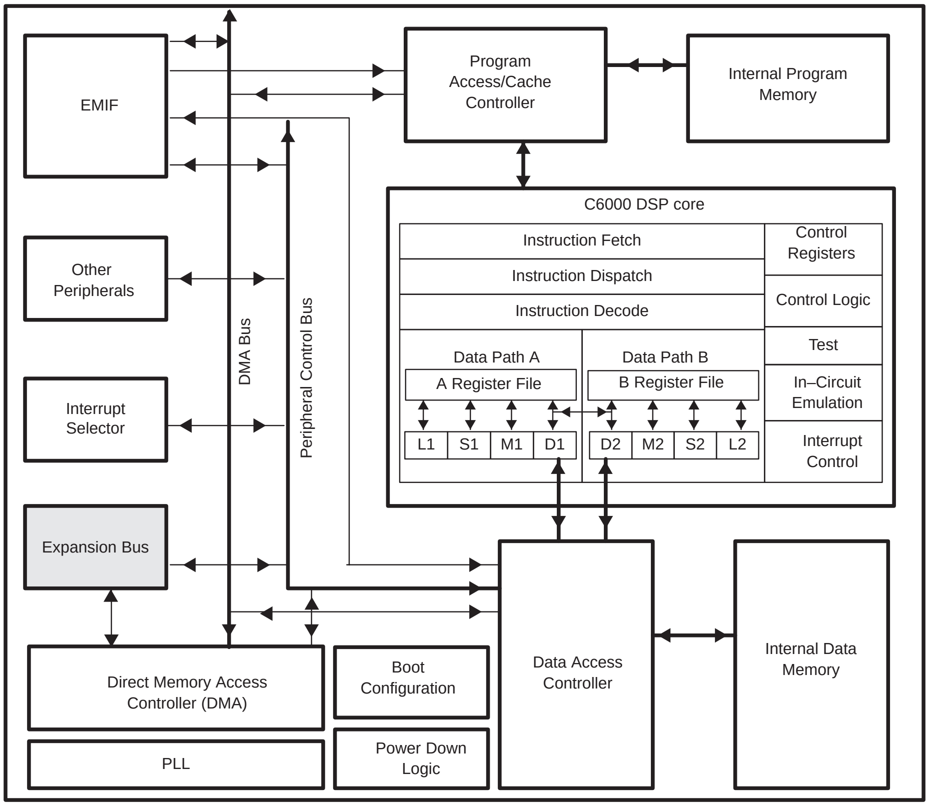


The I/O port has two modes of operation, which can coexist in a single system: asynchronous I/O mode and synchronous FIFO mode. These modes are selectable for each of four XCE spaces in the XBUS. The first mode (asynchronous I/O mode) provides output strobes, which are highly programmable like the asynchronous signals of the external memory interface (EMIF). The XBUS 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 XBUS I/O port and DSP memory is provided through the DMA controller.

The second sub-block of the XBUS 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 XBUS 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 XBUS 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 XBUS 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 the C620x/C670x chip-level block diagram.

Figure 8–2. Expansion Bus Interface in TMS320C620x/C670x Block Diagram



Note: Refer to the specific device datasheet for its peripheral set.

8.2 Expansion Bus Signals

Table 8–1 lists the XBUS signals and their functionality in each mode. If only the I/O port of the XBUS is used (or if the XBUS is not used at all), the XBUS signals should be pulled inactive according to Table 8–2.

Table 8–1. Expansion Bus Signals

	I/O Port Mode (Non-Exclusive)				**Mutually Exclusive Host Port Modes**			
XBUS 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	$\overline{\text{RE}}$				
$\overline{\text{XWE/XWAIT}}$	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

Table 8–2. Signal State for Disabled Host Port

XBUS Signal	I/O Port Mode (I/O/Z)	External Connection
XD[31:0]	I/O/Z	According to system (See *section 8.7)*
XFCLK	O	N/C
XCLKIN	I	Pull up
XCE[3:0]	O	N/C
$\overline{\text{XBE[3:0]}}$/XA[5:2]	O/Z	Pull down
$\overline{\text{XOE}}$	O	N/C
$\overline{\text{XRE}}$	O	N/C
$\overline{\text{XWE}}$	O	N/C
$\overline{\text{XAS}}$	I/O/Z	Pull up
XRDY	I/O/Z	Pull up
XW/R	I/O/Z	Pull up
XBLAST	I/O/Z	Pull up if BLPOL = 0; Pull down if BLPOL = 1
XHOLD	I/O/Z	Pull down
XHOLDA	I/O/Z	Pull down
XCNTL	I	Pull up
XBOFF	I	Pull down
$\overline{\text{XCS}}$	I	Pull up

8.3 Expansion Bus Registers

Control of the XBUS and the peripheral interfaces is maintained through memory-mapped registers within the XBUS. The memory-mapped registers are shown in Table 8–3.

Table 8–3. Expansion Bus Memory Mapped Registers

Byte Address, Hex	Abbreviation	Name	Described in Section
0188 0000	XBGC	Expansion Bus Global Control Register	8.3.2
0188 0004	XCECTL1	XCE1 Space Control Register	8.3.3
0188 0008	XCECTL0	XCE0 Space Control Register	8.3.3
0188 000c	XBHC	Expansion Bus Host Port Interface Control Register	8.5.1.5
0188 0010	XCECTL2	XCE2 Space Control Register	8.3.3
0188 0014	XCECTL3	XCE3 Space Control Register	8.3.3
0188 0018	—	Reserved	—
0188 001c	—	Reserved	—
0188 0020	XBIMA	Expansion Bus Internal Master Address Register	8.5.1.3
0188 0024	XBEA	Expansion Bus External Address Register	8.5.1.4

8.3.1 Expansion Bus Host Port Registers

The external master on the XBUS uses the XCNTL signal to select which internal register is being accessed. The state of this pin selects whether access is made to the XBUS 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 8–4 summarizes the registers that the XBUS host port uses for communication between the host device and the CPU.

Table 8–4. Expansion Bus Host Port Registers

Register Abbreviation	Register Name	Host Read/ Write Access	DSP 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 (XBGC)

The XBUS global control register (XBGC), shown in Figure 8–3 and described in Table 8–5, configures parameters of the XBUS that are common to all interfaces. The XBUS global control register must not be modified while I/O port transactions are in progress.

Figure 8–3. Expansion Bus Global Control Register (XBGC)

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 8–5. Expansion Bus Global Control Register (XBGC) Field Description

Field	Description	Section
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 $\overline{XCE3}$ is selected for FIFO mode, then XOE acts as FIFO output enable and $\overline{XCE3}$ acts as FIFO read enable. *XOE is disabled in all other XCE spaces regardless of MType setting.*	8.4.2

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.
		8.4.2

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.
		8.4.2

XARB	Arbitration mode, set by boot-mode selection
	XARB=0: internal arbiter disabled
	XARB=1: internal arbiter enabled
		8.6

8.3.3 XCE Space Control Registers (XCExCTL)

The four XCE space control registers (XCExCTL), shown in Figure 8–4 and described in Table 8–6, correspond to the four XCE memory spaces supported by the XBUS.

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

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 8–6. Expansion Bus XCE(0/1/2/3) Space Control Register (XCExCTL) Field Description

Field	Description	Section
Read Setup/ Write Setup	Setup width. Number of CLKOUT1 cycles of setup time for byte–enable/address ($\overline{\text{XBE}}$/XA) and chip enable ($\overline{\text{XCE}}$) before read strobe or write strobe falls. For asynchronous read accesses, this is also the setup time of $\overline{\text{XOE}}$ before $\overline{\text{XRE}}$ falls.	8.4.1
Read Strobe/ Write Strobe	Strobe width. The width of read strobe ($\overline{\text{XRE}}$) and write strobe ($\overline{\text{XWE}}$) in CLKOUT1 cycles.	8.4.1
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.	8.7
Read Hold/ Write Hold	Hold width. Number of CLKOUT1 cycles that byte–enable/address ($\overline{\text{XBE}}$/XA) and chip enable ($\overline{\text{XCE}}$) are held after read strobe or write strobe rises. For asynchronous read accesses, this is also the hold time of $\overline{\text{XCE}}$ after $\overline{\text{XRE}}$ rising.	8.4.1

8.4 Expansion Bus I/O Port Operation

For external I/O port accesses on the XBUS, 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 XBUS I/O port can only be done through the DMA channels 0 through 3. The DMEMC does not have direct access to the XBUS. Therefore, load and store (LD/ST) commands to the memory spaces of the XBUS 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 XBUS 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 8–7).

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 8–8).

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 8–8). 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

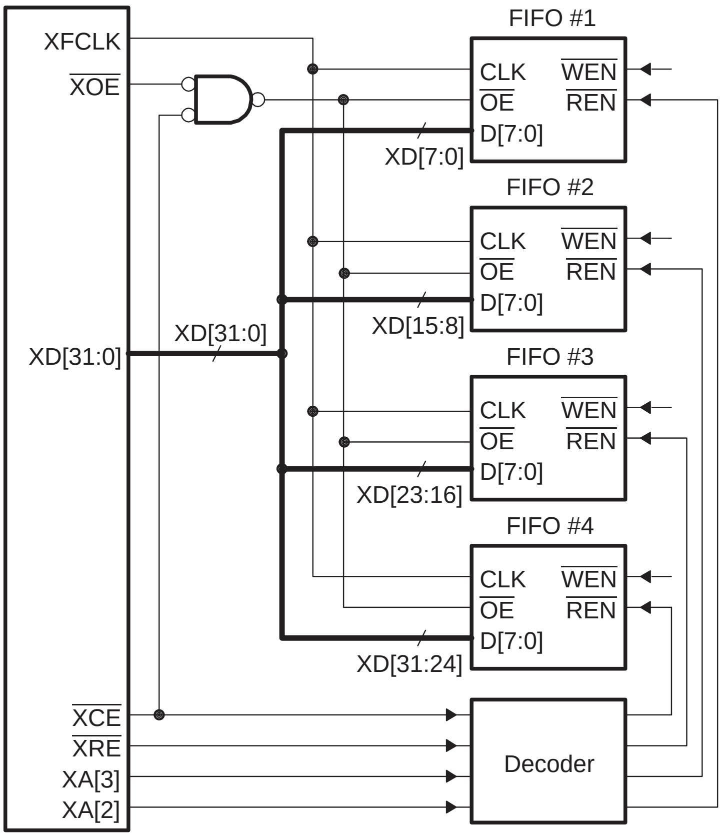


Table 8–7. Addressing Scheme – Case When Expansion Bus is Interfaced to Four 8-Bit FIFOs

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

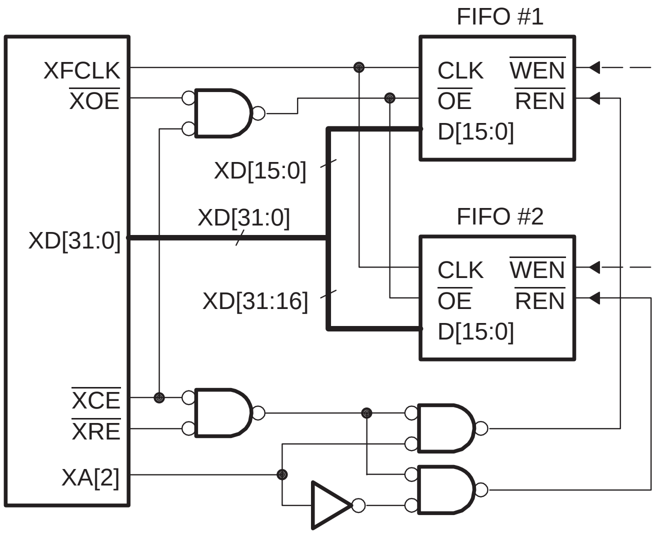


Table 8–8. Addressing Scheme – Case When the Expansion Bus is Interfaced to Two 16-Bit FIFOs

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 XBUS 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 Synchronous FIFO Modes

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

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

A pin description of synchronous FIFO is in Table 8–9.

Table 8–9. Synchronous 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

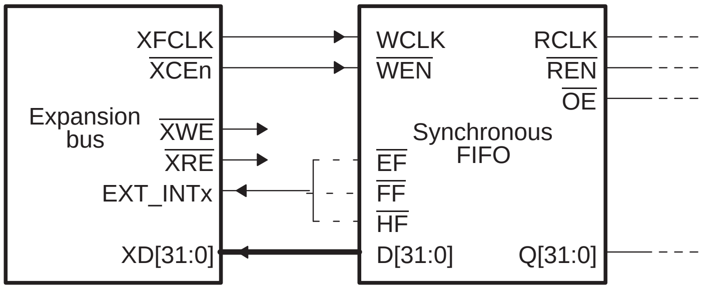


Figure 8–8. Read and Write FIFO Interface With Glue

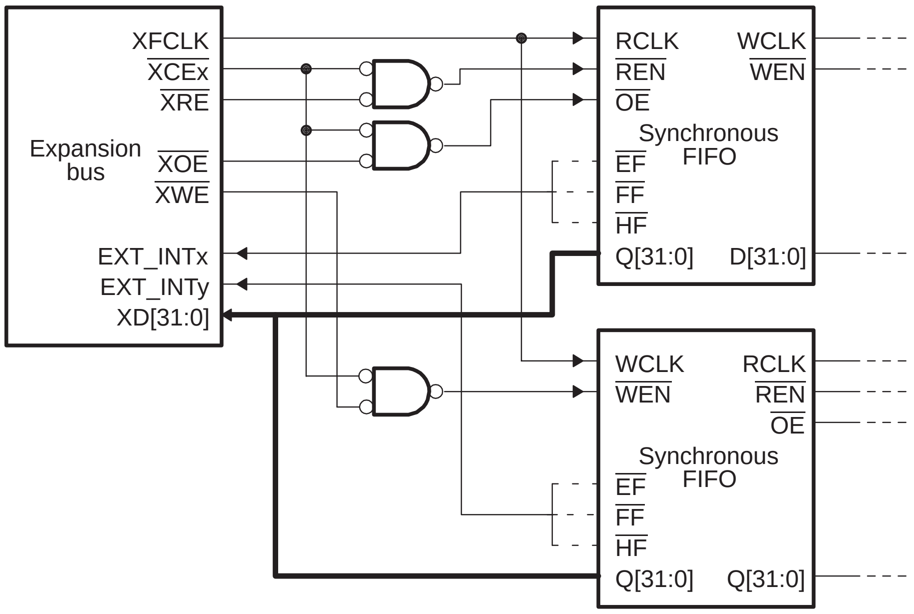
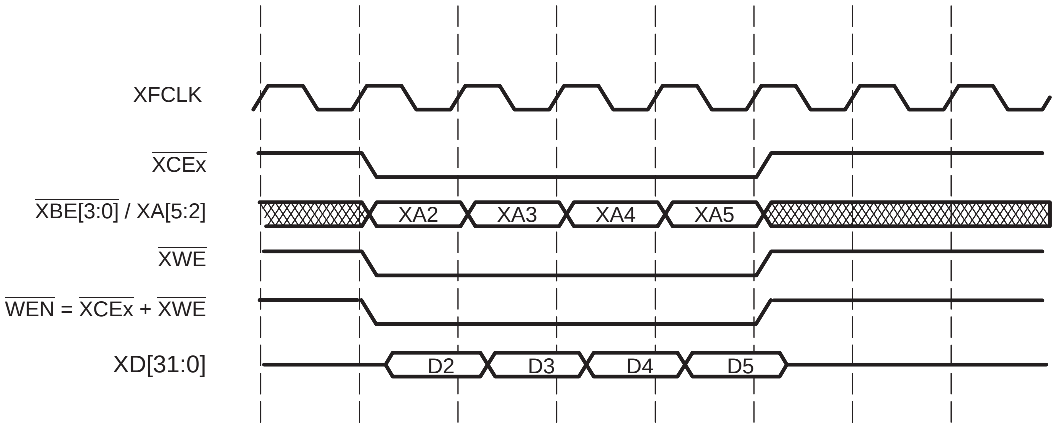


Figure 8–9. FIFO Write Cycles



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

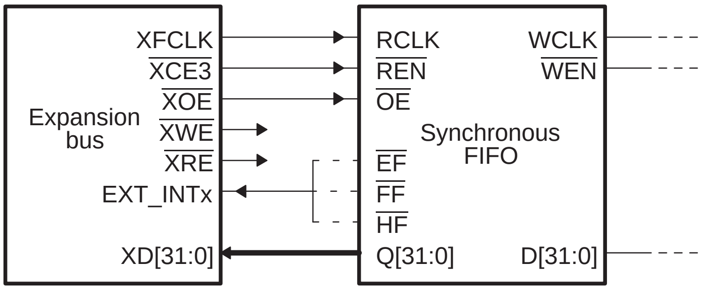


Figure 8–11. FIFO Read Mode – Read Timing (glueless case)

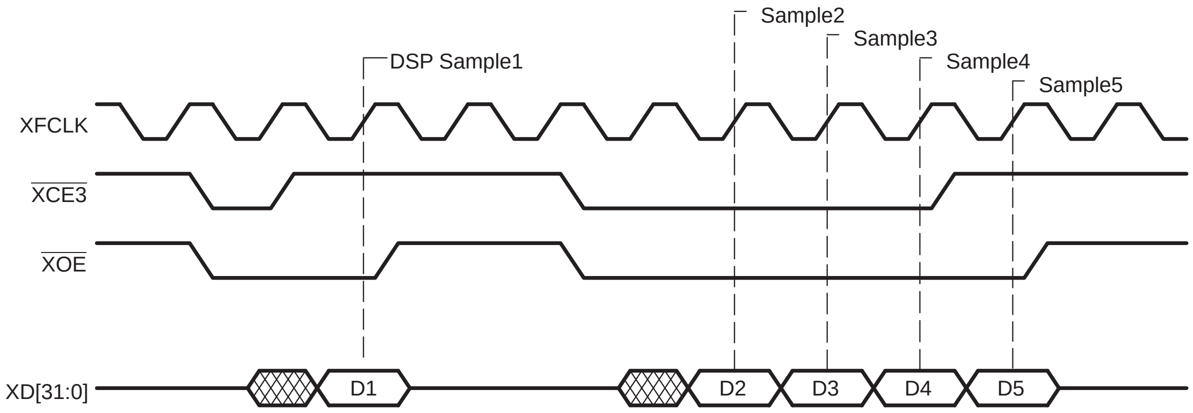
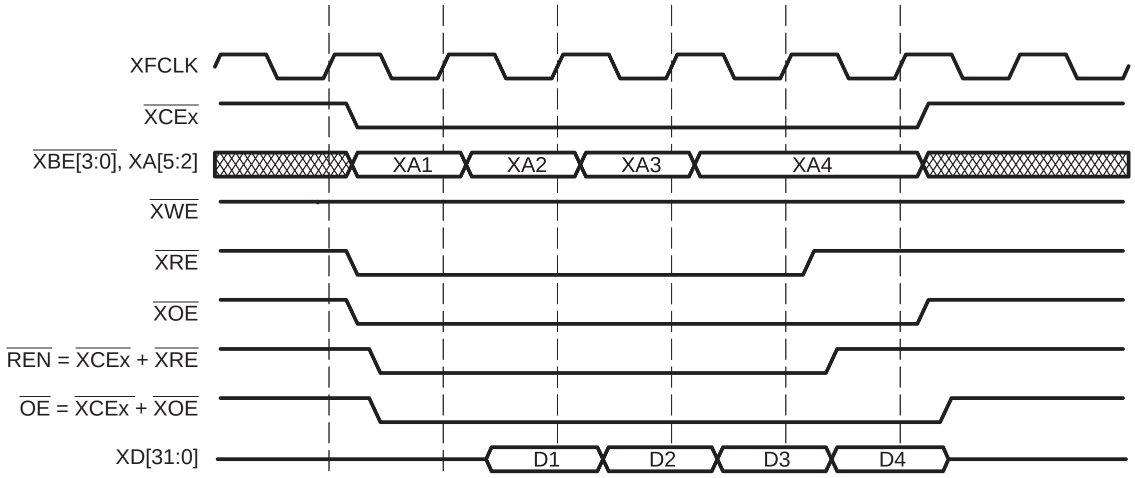


Figure 8–12. FIFO Read Mode – With Glue


XFCLK
$\overline{\text{XCEx}}$
$\overline{\text{XBE[3:0]}}$, XA[5:2]
XA1
XA2
XA3
XA4
$\overline{\text{XWE}}$
$\overline{\text{XRE}}$
$\overline{\text{XOE}}$
$\overline{\text{REN}}$ = $\overline{\text{XCEx}}$ + $\overline{\text{XRE}}$
$\overline{\text{OE}}$ = $\overline{\text{XCEx}}$ + $\overline{\text{XOE}}$
XD[31:0]
D1
D2
D3
D4

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 XBUS 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 8–10 and Table 8–11.

Table 8–10. 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 8–11. 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 8–12, Table 8–13, and Table 8–14.

Table 8–12. 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 8–13. Content of 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 8–14. Content of DMA Secondary Control Register

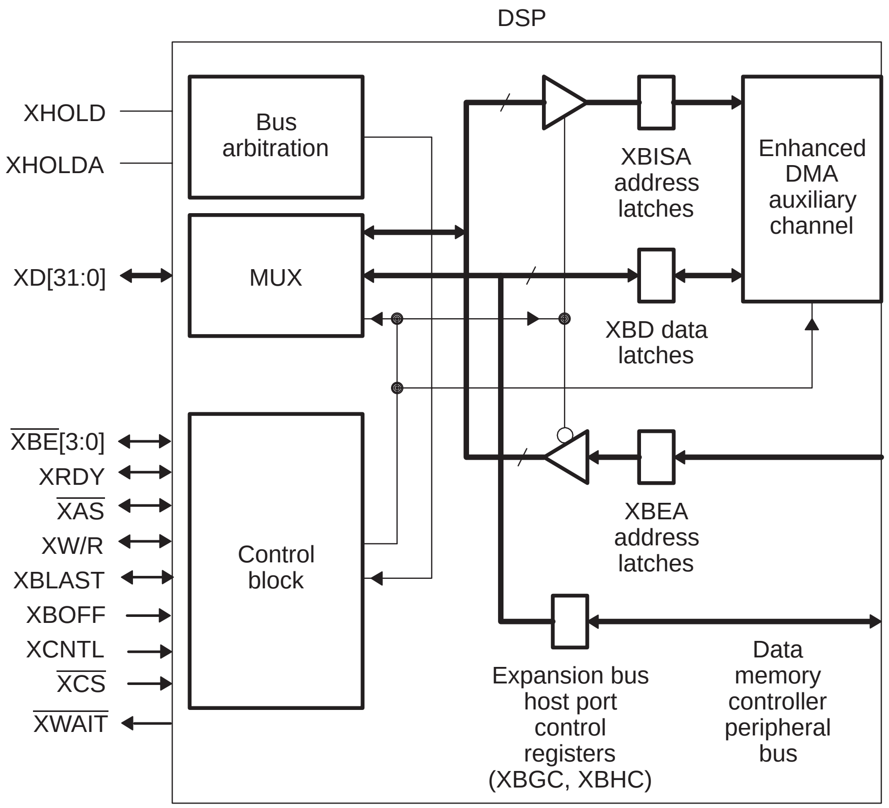
Reserved	**SYNC CNTL**	**DMAC EN**	**WSYNC CLR/WSYNC STAT/RSYNC CLR/RSYNC STAT/WDROP IE/WDROP COND/WDROP COND/RDROP IE/RDROP COND/BLOCK IE/BLOCK COND/LAST IE/LAST COND/FRAME IE**
0000 0000 00	001	0 00	0000 0000 0000 0000

8.5 Expansion Bus Host Port Operation

The Expansion Bus (XBUS) Host Port has two modes of operation which enable interfaces to external processors, PCI bridge chips, or other external peripherals. These two operation modes 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 XBUS host port block diagram is shown in 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 (XBD)

The expansion bus data (XBD) register, shown in Figure 8–14, contains the data that was read from the memory accessed by the XBUS 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. The host can perform single 32–, 16–, or 8–bit accesses to the XBD. Bursts longer than one word to the XBD should always be 32–bits wide.

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

Figure 8–14. Expansion Bus Data Register (XBD)

31 ... 0
XBD
HRW,+0000 0000 0000 0000 0000 0000 0000 0000

8.5.1.2 Expansion Bus Internal Slave Address Register (XBISA)

The expansion bus internal slave address (XBISA) register is used when the external XBUS 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 host can only perforem 32–bit accesses to the XBISA. The XBISA register is shown in Figure 8–15 and described in Table 8–15.

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

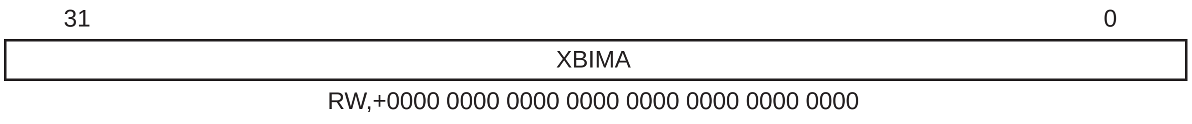
This register is used when the host port operates either in synchronous or asynchronous mode. The DSP does not have access to the XBISA register content. Burst transfers in the synchronous host-port mode are always expected to occur with autoincrement (AINC bit should be set to zero). In auto-increment mode (AINC = 0), operation is undefined if an external host attempts to access the last 2 word locations in the Internal Data RAM. This is because the DSP tries to pre-fetch data from reserved locations above the Internal Data RAM.

8.5.1.3 Expansion Bus Internal Master Address Register (XBIMA)

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 DSP 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 (XBIMA)



8.5.1.4 Expansion Bus External Address Register (XBEA)

The expansion bus external address (XBEA) register is set by the DSP when it is ready to initiate transfer on the XBUS. 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 XBEA register specifies where the data is accessed in the external slave memory map. 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 (XBEA)

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

8.5.1.5 Expansion Bus Host Port Interface Control Register (XBHC)

The expansion bus host port interface control (XBHC) register (shown in Figure 8–18 and described in Table 8–16) 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 Register (XBHC)

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 8–16. Expansion Bus Host Port Interface Control Register (XBHC) 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 Start = 11 reserved 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 XBUS registers reflect the state of the aborted transfer. Using this state information, you can restart the transfer.
INTSRC	The XBUS 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 32-bit words 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 XBUS.

The XBUS has the capability to initiate and receive burst transfers.

Table 8–17 lists pin function in the XBUS synchronous host port mode:

Table 8–17. Expansion Bus Pin Description (Synchronous Host Port Mode)

Signal Symbol	Signal Type	Signal Count	Signal Name	Signal Function
XCLKIN	I	1	Clock Input	XBUS clock (maximum clock speed is 1/4 of the CPU clock speed.
$\overline{\text{XCS}}$	I	1	Chip Select	Selects the DSP 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 XBUS. The DSP asserts XHOLDA when control is granted. Case 2 (Internal bus arbiter disabled) The DSP wakes up from reset as slave on the bus. XHOLD is asserted by the DSP to request use of the XBUS. The XBUS arbiter asserts XHOLDA when control is granted.
XHOLDA	I/O/Z	1	Hold acknowledge	Case 1 (Internal bus arbiter disabled) The DSP wakes up from reset as slave on the bus. The XBUS arbiter asserts XHOLDA when control is granted in response to XHOLD. The bus should not be granted to the DSP unless requested by XHOLD. Case 2 (Internal bus arbiter enabled) The DSP wakes up from reset as master of the bus. XHOLDA is asserted by the DSP when control is granted in response to XHOLD.
XD[31:0]	I/O/Z	32	Address/ data bus	Data

Table 8–17. Expansion Bus Pin Description (Synchronous Host Port Mode) (Continued)

Signal Symbol	Signal Type	Signal Count	Signal Name	Signal Function
XBLAST	I/O/Z	1	Burst last	Signal driven by the current XBUS 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] Note: For XBD access: 8-bit data must be byte-aligned 16-bit data must be halfword-aligned 32-bit data must be word-aligned. For XBISA access, all $\overline{\text{XBE[3:0]}}$ must be active low.
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 DSP owns the bus. When the DSP does not own the bus, XRDY is not driven until a request is made to the DSP.
XBOFF	I	1	Bus Back-Off	When asserted, suspends the current access and the DSP releases ownership of the XBUS.
$\overline{\text{XWAIT}}$	O	1	Wait	Ready output when the DSP intitiates transfers on the XBUS.

8.5.2.1 TMS320C62x Master on the Expansion Bus

When the C62x is the master of the XBUS, 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 XBUS 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 XFRCT field of the expansion bus host port control (XBHC) register field is set to control the number of 32-bit words being transferred.
   Note: Only 32-bit transfers are supported by the XBUS when the DSP is the master in synchronous host port mode.

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 DSP is mastering the bus. In this case internal bus arbiter is disabled (XHOLD is output and XHOLDA is input) and DSP wakes up from reset as slave on the XBUS.

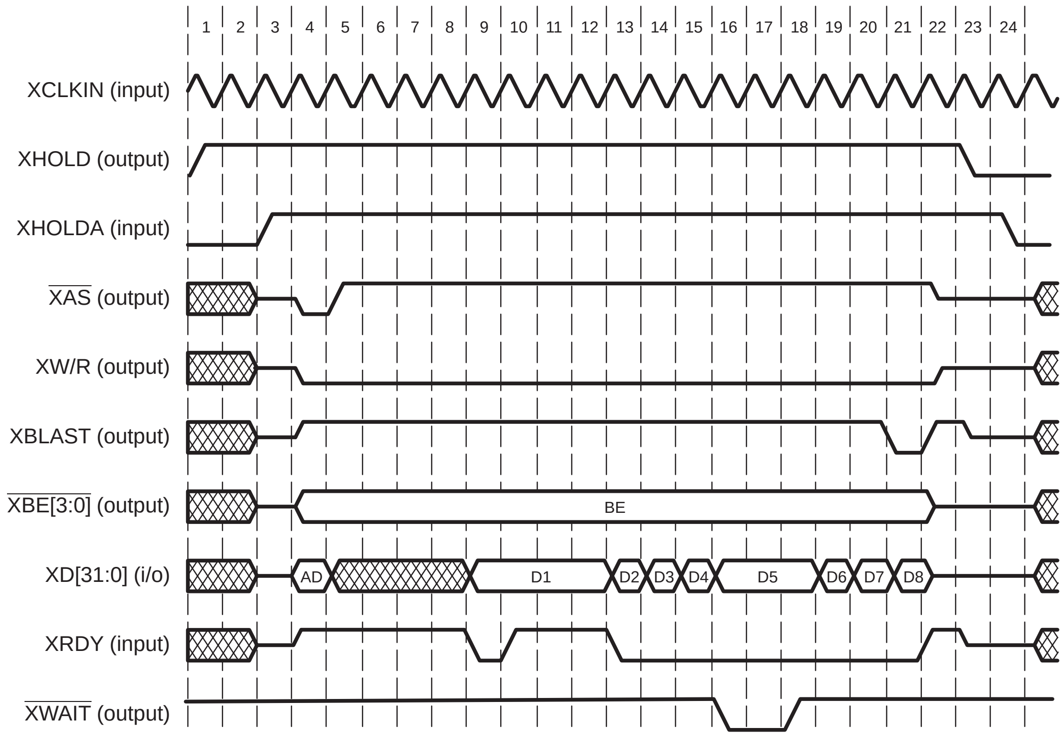
The $\overline{\text{XWAIT}}$ signal prevents data overflow/underflow when the DSP is a master on the XBUS. The $\overline{\text{XWAIT}}$ signal, which is multiplexed with the $\overline{\text{XWE}}$ signal, can be thought of as a ready output when the DSP initiates transfers on the XBUS. When the DSP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

Note: $\overline{\text{XWAIT}}$ is an output only signal in synchronous host port mode.

Burst Read Transfer

The timing presented in Figure 8–19 can be referenced for a visual description of the steps required to complete a burst read initiated by the DSP and throttled by the $\overline{\text{XWAIT}}$ and XRDY signals.

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



This is the step-by-step description of the events marked above the waveforms in Figure 8–19:

1) The DSP requests the XBUS by asserting XHOLD output.

2) The DSP waits for the XBUS.

3) The external bus arbiter asserts the XHOLDA signal, and the DSP 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 XBUS.

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

6) Same as step 5.

7) Same as step 5.

8) Same as step 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 step 10

12) Same as step 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 XBUS. 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 DSP asserts the XBLAST.

22) The recovery cycle.

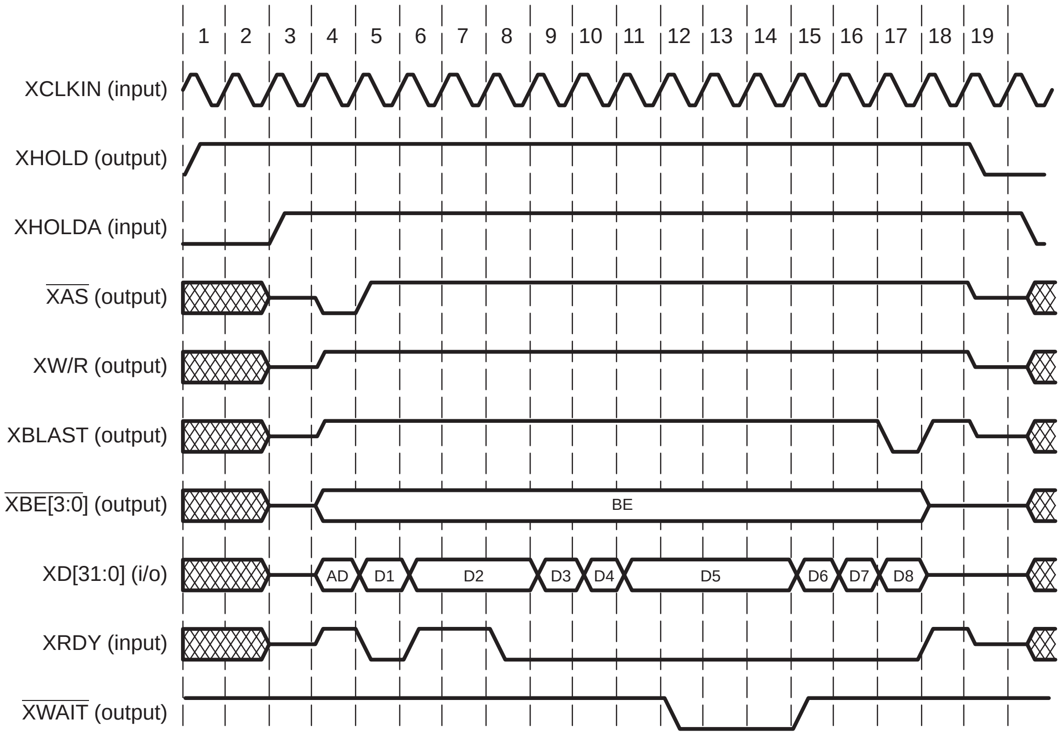
23) The DSP negates the XBUS request (XHOLD), and turns off the outputs.

Note: $\overline{\text{XWAIT}}$ is an output only signal in synchronous host port mode.

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 DSP and throttled by the $\overline{\text{XWAIT}}$ and XRDY signals.

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



This is the step-by-step description of the events marked above the waveforms in Figure 8–20:

1) The DSP requests the XBUS (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 XBUS.

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 DSP is holding data D2 on the XBUS 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 step12.

14) Same as step 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.

Note: $\overline{\text{XWAIT}}$ is an output only signal in synchronous host port mode.

To prevent contention on the XBUS, one recovery state between the last data transfer and next address cycle is inserted.

Preventing Deadlocks with Backoff

The XBUS has the XBOFF signal to prevent deadlocks while the DSP is performing a master transfer. When asserted, XBOFF suspends the current access and causes the DSP to release ownership of the XBUS. 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 one of the following conditions exists:

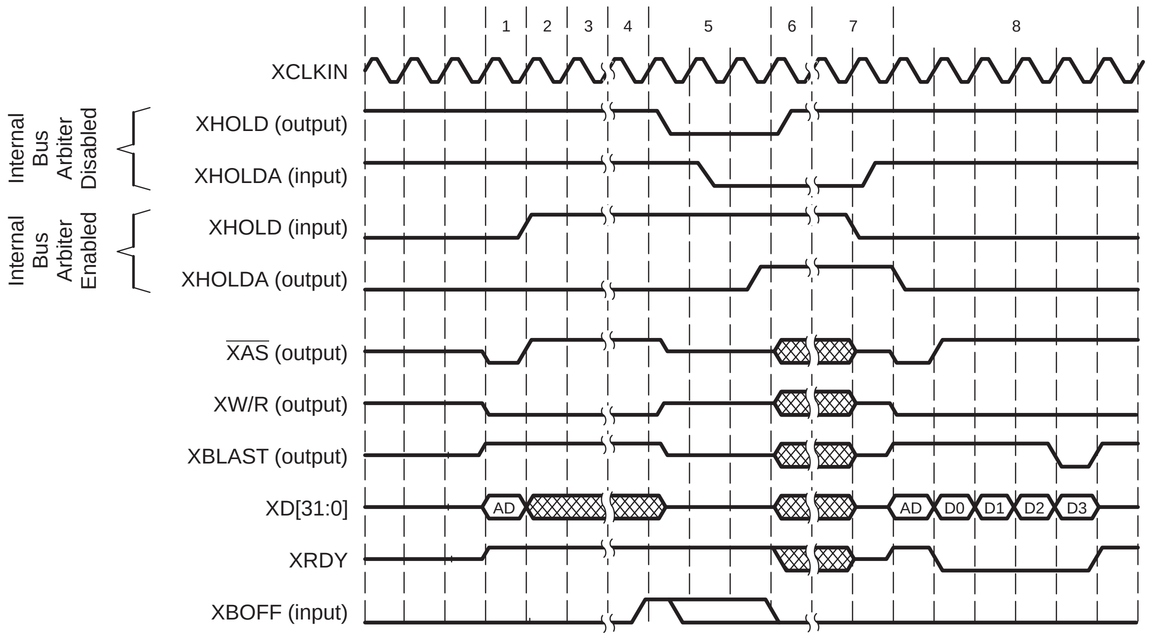
1) The external device is requesting the XBUS (XHOLD = 1), when the internal bus arbiter is enabled (XARB = 1)

or

2) The DSP is the XBUS 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 DSP 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 XBUS 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 DSP is the XBUS 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 DSP is still holding the XBUS waiting for XRDY to become low.
4) The external device asserts XBOFF, indicating a potential deadlock condition.
5) The DSP responds by releasing the XBUS. 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 the DSP responds to XBOFF. Figure 8–21 shows the fastest response time, one cycle.
6) The XBUS ownership changes. The new master drives the XBUS. 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 XBUS 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 TMS320C62x 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 XBUS 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. Bursts longer than one word must take place with auto-incrementing of the XBISA register. Therefore, the AINC bit must always be cleared to 0.

To read/write from the DSP memory space, the host must follow the following sequence:

1) The host writes the transfer source/destination address to the XBISA register and clears the AINC bit (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 XBISA XW/R signal. The XBISA register is autoincremented since bit one of the XBISA register must be cleared by the external host.
3) If the transfer is a burst, dictated by the BLAST signal, data is continuously read or written. If a multi-word burst is being done, all $\overline{\text{XBE}}$x signals must be low, because only 32-bit word bursts are allowed. If less than 32 bits are transferred (specified by $\overline{\text{XBE}}$x signals), then only single element transfers are allowed.

Note: $\overline{\text{XWAIT}}$ is not used in slave mode.

Cycle Description

Each access initiated by the external host can be broken up into distinct categories:

❑ **Address phase (Ta):** During the address phase, the DSP 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 DSP determines if the external master is doing a read or write cycle (XW/R input) and which XBUS register is being accessed (via the XCNTL input).

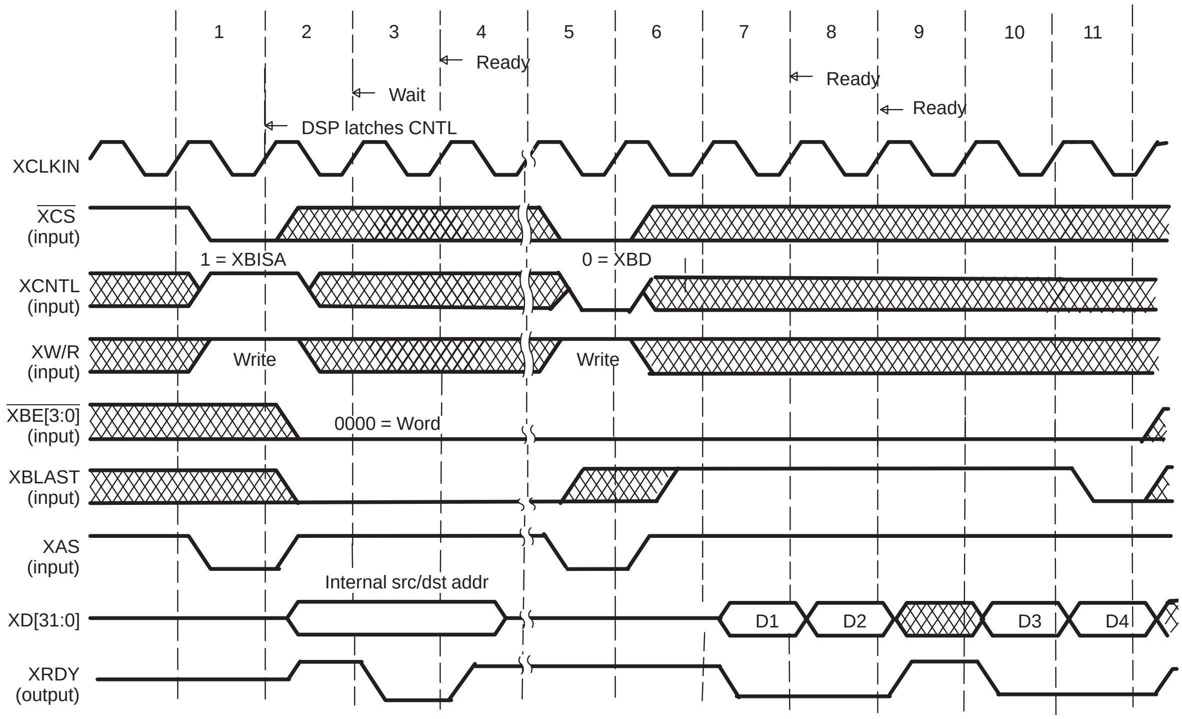
❑ **Wait/data phase (Tw/Td):** Immediately after the address phase, the transaction enters either the wait phase or data phase. For a read cycle, there is at least one wait phase before the DSP presents the data to the external host. This is controlled via the XRDY output of the DSP. If the XRDY signal is high, this indicates to the external host that the DSP 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 DSP asserts XRDY signal, indicating that read data should be latched by the external host or that write data has been latched by the DSP.

❑ **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 DSP 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. Expansion Bus Master Writes a Burst of Data to the DSP



The boot configuration for XBLAST and XW/R: BLPOL = 0 and RWPOL = 0. See Table 8–17 for more details.

The step-by-step description of the events marked above the waveforms in Figure 8–22 follows:

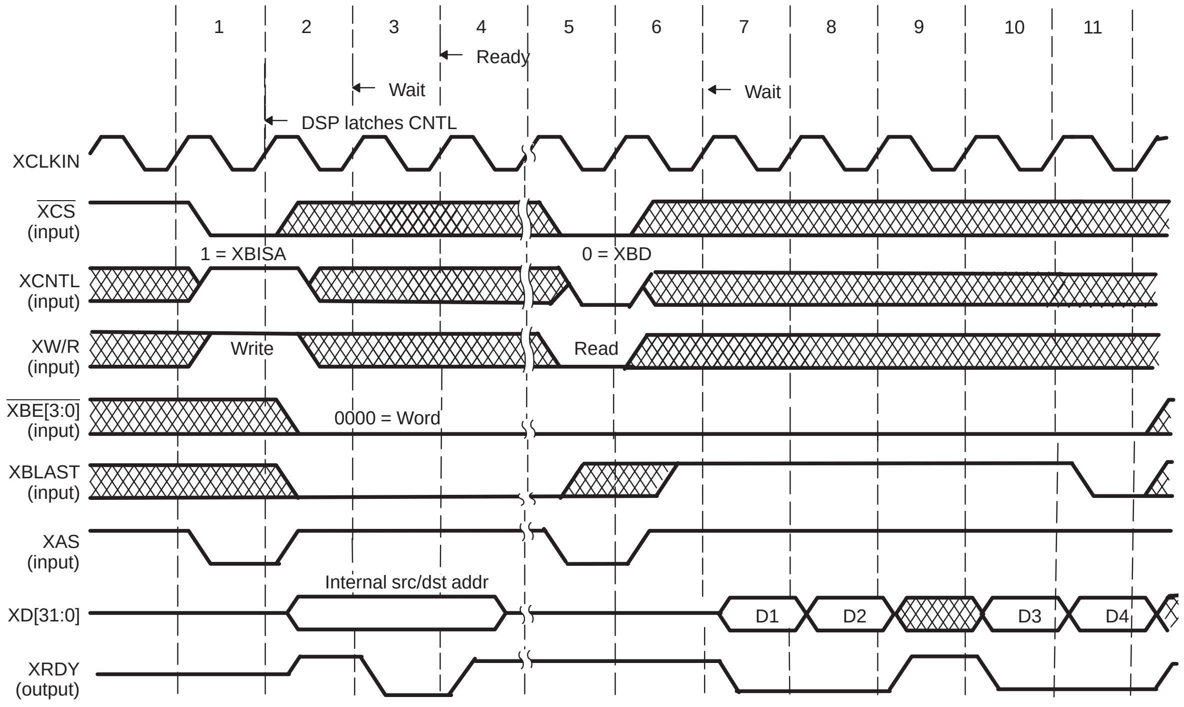
1) The $\overline{\text{XCS}}$, $\overline{\text{XAS}}$ and XCNTL signals are low, low, and high respectively, indicating XBISA register as the destination for the following transaction. The XW/R is high specifying that a write access is taking place.
2) The DSP begins driving the XRDY output in response to a transfer initiated by the external host. A high XRDY indicates that the DSP is not ready.
3) The data is written to the XBISA register when the DSP asserts the XRDY-output low.
4) The DSP inserts one or more not-ready cycles (XRDY=1) between the address phase and the first data phase.
5) 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.
6) The DSP inserts one not-ready cycle (XRDY=1).
7) The XBUS 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.
8) Same as step 7.
9) The DSP is not ready to accept next data, which is indicated by XRDY high.
10) Same as step 7.
11) The XBUS 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.

Note: $\overline{\text{XWAIT}}$ is not used in slave mode.

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 DSP



The boot configuration for XBLAST and XRW: BLPOL = 0 and RWPOL = 0. See Table 8–17 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 DSP begins driving the XRDY output in response to a transfer initiated by the external host. A high XRDY indicates that the DSP is not ready.
3) The data is written to the XBISA register when the DSP asserts the XRDY output low.
4) The DSP inserts one or more not-ready cycles (XRDY=1) between the address phase and the first data phase.
5) 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.
6) The DSP inserts one not-ready cycle (XRDY=1).
7) The ‘DSP presents the valid data, and drives XRDY low.
8) Same as step 7.
9) The DSP is not ready to present the next data, which is indicated by XRDY high.
10) Same as step 7.
11) The XBUS master indicates that the last read transaction is taking place by asserting the XBLAST signal.

Note: $\overline{\text{XWAIT}}$ is not used in slave mode.

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 XBUS operates in the asynchronous host port mode is given in Table 8–18.

Table 8–18. 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 DSP 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 DSP 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] Note: 8-bit data must be byte-aligned**.** 16-bit data must be halfword-aligned**.** 32-bit data must be word-aligned**.**
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 DSP 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 XBUS host port operates in the asynchronous mode, every transaction initiated by the host on the XBUS 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 DSP 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 upon what is written to the AINC bit during step 1.

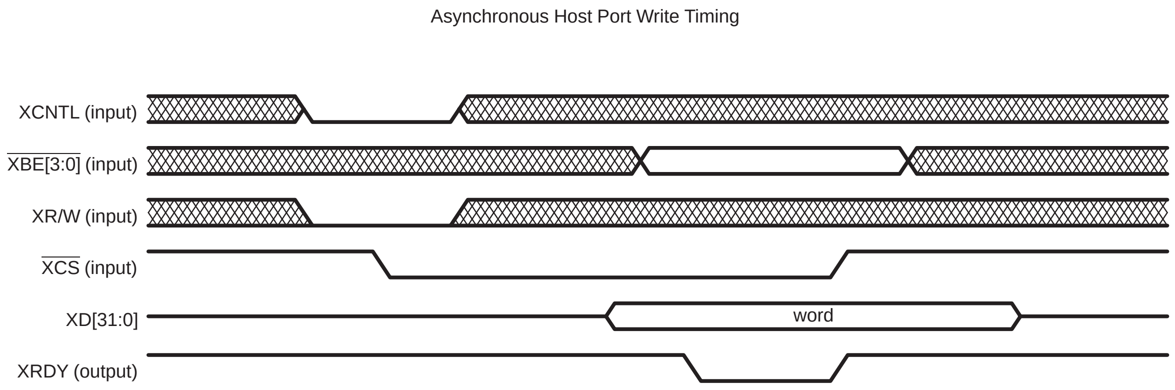
If the XBUS host port is configured to operate in asynchronous mode the $\overline{\text{XCS}}$ signal is used for four purposes:

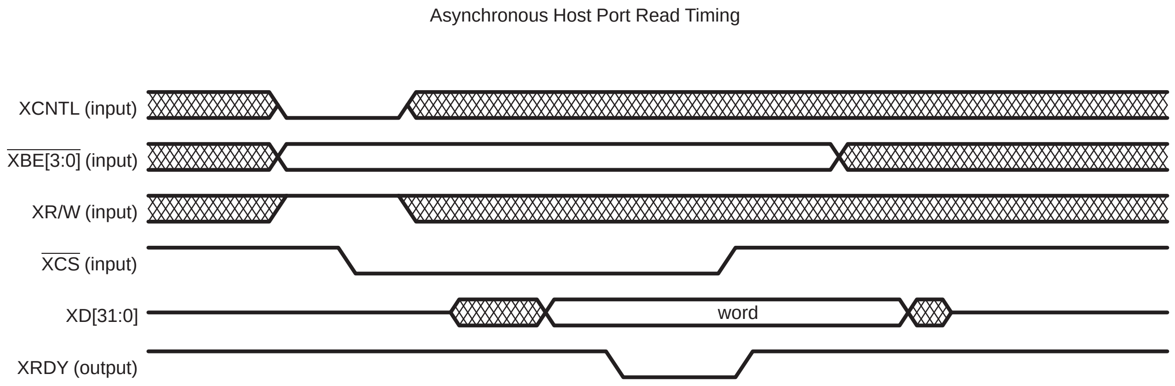
1) To select the XBUS host port as a target of an external master.
2) On a read, the falling edge of $\overline{\text{XCS}}$ initiates read accesses.
3) On a write, the rising edge of $\overline{\text{XCS}}$ initiates write accesses.
4) The $\overline{\text{XCS}}$ falling edge latches XBUS host port control inputs including: XW/R and XCNTL.

The XRDY signal of the DSP 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 XBUS 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 expansion 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 (XBGC) register indicates if the internal bus arbiter is enabled or disabled. This is shown in Table 8–19.

Table 8–19. 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 DSP wakes up from reset as the bus master. If internal bus arbiter is disabled, the DSP wakes up from reset as the bus slave. The DMA controller releases the XBUS between frames if a DMA block transfer is in progress. When the DSP releases the XBUS, 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

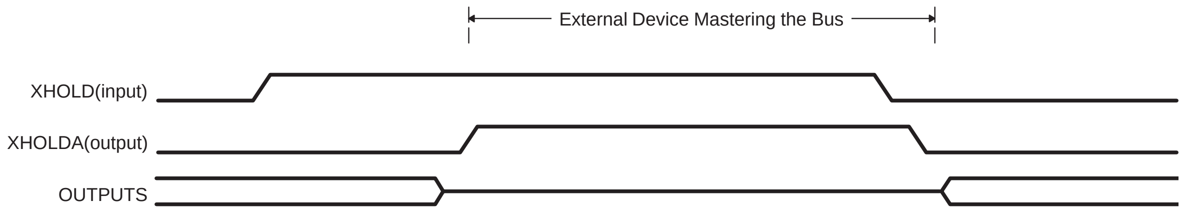
In this mode the DSP owns the XBUS by default. The DSP wakes up from reset as the master of the XBUS, and all other devices must request the bus from DSP. This mode is preferred when connecting one DSP to a PCI interface chip.

When the DSP owns the XBUS, both XHOLD (input) and XHOLDA (output) are low. XHOLD is asserted by an external device to request use of the XBUS. The DSP asserts XHOLDA when bus request is granted. The XBUS 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 XBUS to the requester only if no internal transfer requests to the XBUS are pending.

During a synchronized slave single word write to the XBD, if XHOLD input is de-asserted quickly enough after XBLAST = 0 (in the same cycle) the transfer gets corrupted. No single word slave write with the simultaneous removal of the XHOLD and assertion of the XBLAST should be allowed (XARB =1). In these cases the XHOLD should be registered before connecting it to the XHOLD input on the DSP.

Figure 8–25. Timing Diagrams for Bus Arbitration – XHOLD/XHOLDA (Internal Bus Arbiter Enabled)



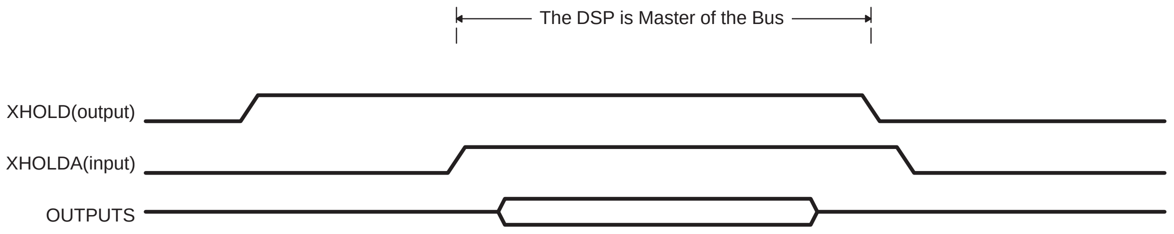
8.6.2 Internal Bus Arbiter Disabled

In this mode, the DSP acts as slave on the XBUS by default. This mode is preferred if the DSP is interfacing to an external host, or if multiple DSPs are connected to a PCI interface chip.

When the DSP owns the XBUS, both XHOLD (output) and XHOLDA (input) are high. To request the XBUS (for example to access a FIFO) the DSP asserts XHOLD. The external XBUS arbiter asserts XHOLDA when control is granted. The XBUS should not be granted to the DSP 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 XBUS 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 XBUS, 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 XBUS 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 8–20.

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)

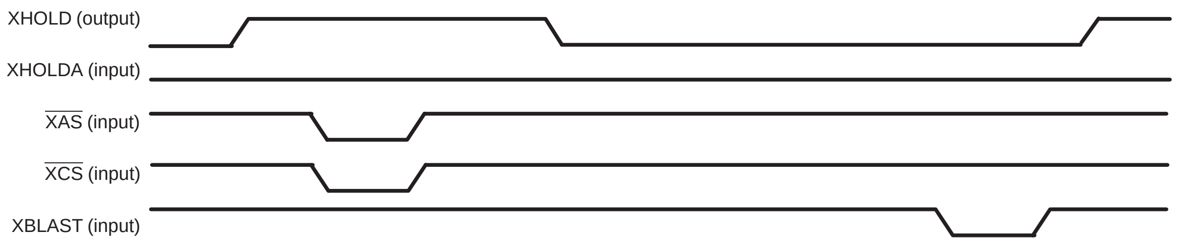

XHOLD (output)
XHOLDA (input)
$\overline{\text{XAS}}$ (input)
$\overline{\text{XCS}}$ (input)
XBLAST (input)

Table 8–20 shows possible scenarios that can happen when the internal bus arbiter is disabled (XARB =0).

Table 8–20. 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 XBUS in progress	DMA request to the XBUS I/O 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 keeps 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 XBUS I/O port, and auxiliary DMA requests are pending	After the DSP gets the XBUS 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 DMA transfer is completed, the DSP starts the auxiliary DMA transfer and does not drop the XHOLD between these two transfers.
		Auxiliary 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 8–19). 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 8–20. 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 XBUS I/O port pending	The DSP asserts the XHOLD, and once it gets the XBUS the transfer starts.
		DMA request to XBUS I/O port, and auxiliary DMA requests are pending	After the DSP gets the XBUS the pending auxiliary DMA request is executed first (since for the XBUS, 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 between these two transfers.
		Auxiliary DMA request pending	The DSP asserts the XHOLD, and once it gets the XBUS the transfer starts.
YES	N/A	DMA transfer to XBUS I/O port in progress	XBOFF is ignored if a DMA transfer to the XBUS I/O port is in progress.
		Auxiliary DMA transfer in progress	The DSP releases ownership of the XBUS as soon as possible. After that, the DSP requests the XBUS to complete the transfer interrupted by the XBOFF.
		Auxiliary DMA transfer in progress, and DMA request to XBUS I/O port pending	The DSP stops the current auxiliary DMA transfer in progress, and starts executing the pending DMA transfer to the XBUS I/O port. After the pending DMA transfer is completed, the DSP releases the XBUS to the external device. Some time afterwards, the DSP requests the XBUS to complete the transfer interrupted by the XBOFF.

8.6.3 Expansion Bus Requestor Priority

The auxiliary DMA channel for the XBUS 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 DSP 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 XBUS, DMA transactions should be set up so that the frame length is as short as possible. The size of frame transfers to the XBUS I/O port define the longest amount of time that host transactions can be blocked from accessing the XBUS.

8.7 Boot Configuration Control via Expansion Bus

The pull up/pull down resistors on the expansion bus XD[31:0] pins are used to determine the boot and device configurations during reset. The boot and device configurations include:

- ❑ Boot-mode of the device
- ❑ Lendian mode selection
- ❑ FIFO mode
- ❑ Internal expansion bus arbiter enable/disable
- ❑ Expansion bus host port mode
- ❑ Polarity of read/write XW/R and XBLAST control signals
- ❑ Memory type used in each XBUS memory space.

See *Chapter 11, Boot Modes and Configuration*, for details.

Chapter 9

PCI

This chapter describes the PCI port of the TMS320C6000™ devices. Also refer to the PCI Specification revision 2.2 for details on PCI interface.

Topic **Page**

9.1 Overview

The PCI port supports the following PCI features:

- ❑ Conforms to PCI specification revision 2.2
- ❑ Conforms to power management interface specification revision1.1
- ❑ Meets requirements of PC99
- ❑ PCI master/slave interface
- ❑ 32-bit address/data bus at 33 MHz
- ❑ Single function device
- ❑ Medium address decode
- ❑ PCI access to all on-chip RAM, peripherals, and external memory (via EMIF)
- ❑ Supports memory read, memory read multiple, memory read line, and memory write commands
- ❑ Unlimited slave-access burst lengths
- ❑ Master transfers of up to 64K bytes
- ❑ Single-word transfers for I/O read/writes
- ❑ Single-word transfers for configuration register access
- ❑ 5-V or 3.3-V input signaling, 3.3-V output signaling
- ❑ Many configuration register contents (subsystem ID/subsystem vendor ID, etc.) initialized from an external serial EEPROM at PCI reset.
- ❑ Supports 4-wire serial EEPROM interface
- ❑ EEPROM interface used directly by PCI port without DSP intervention on PCI reset. DSP software control of EEPROM after PCI reset.
- ❑ PCI interrupt request under DSP program control
- ❑ DSP interrupt via PCI I/O cycle
- ❑ DSP power control via software
- ❑ Peripheral power control via software
- ❑ Software-controlled assertion of PME from D0, D1, D2, $D3_{hot}$
- ❑ Hardware-controlled assertion of PME on power wakeup active from $D3_{cold}$. Optional hardware-controlled assertion of PME from D0, D1, D2, $D3_{hot}$.
- ❑ Supports D0, D1, D2, $D3_{hot}$, $D3_{cold}$ power management modes
- ❑ Implements PCI power management control status register “sticky” bits from logic powered by $3.3V_{aux}$
- ❑ Four FIFOs for efficient data transfer (master write, master read, slave write, slave read)
- ❑ Independent master/slave operation
- ❑ Independent slave read/slave write operation
- ❑ Three PCI base address registers (prefetchable memory, non-prefetchable memory, I/O)
- ❑ Disconnect with retry on memory read line, memory read multiple to prefetchable memory
- ❑ No wait states inserted by DSP on PCI master or slave transactions
- ❑ TMS320C6000 contains the logic required to implement a fully compliant PCI Specification revision 2.2 bursting master/slave with access into the DSP’s memory map (peripherals, on-chip RAM, and external through the EMIF).

The PCI Module does not support:

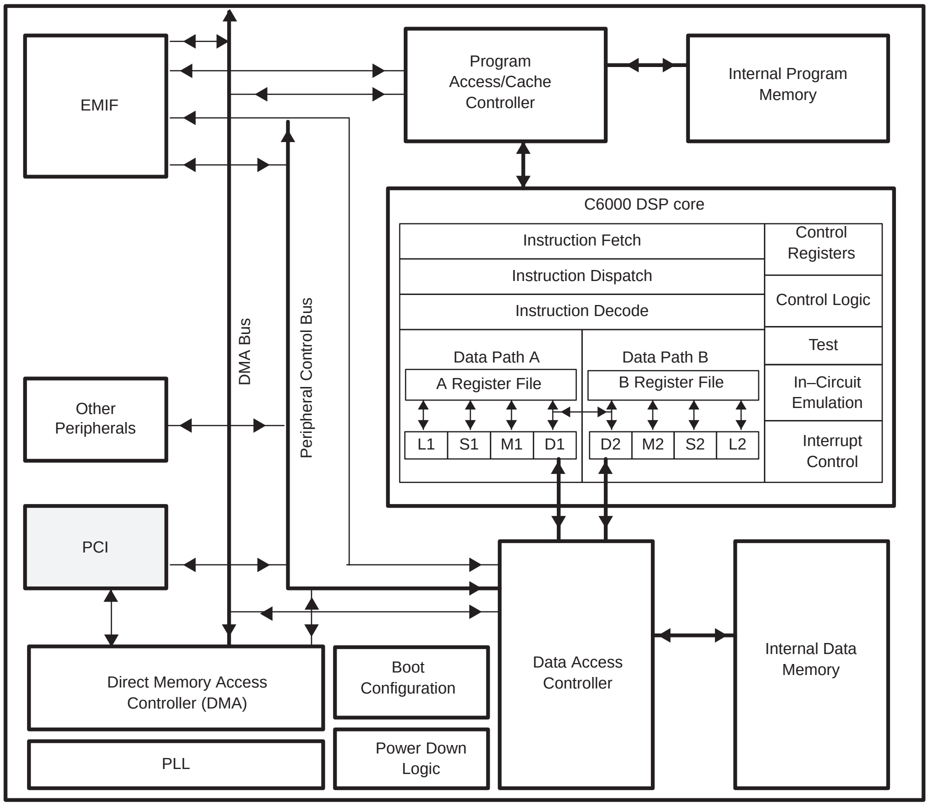
- PCI special cycles
- PCI interrupt acknowledge cycles
- PCI lock
- PCI memory caching
- 64-bit bus operation
- Operation at frequencies greater than 33 MHz
- Master address/data stepping
- Master combining (for write posting)
- Collapsing
- Merging
- Cache line-wrap accesses
- Reserved accesses
- Message-signaling Interrupts
- Vital product data
- Compact PCI hot swap
- Master-initiated I/O cycles
- Master-initiated configuration cycles

The PCI port for the TMS320C6000 supports connection of the DSP to a PCI host via the integrated PCI master/slave bus interface. For C62x/C67x, the PCI port interfaces to the DSP via the auxiliary channel of the DMA Controller. For C64x, the PCI port interfaces to the DSP via the EDMA internal address generation hardware. This architecture allows for both PCI Master and slave transactions, while keeping the DMA/EDMA channel resources available for other applications.

The C62x/C67x PCI port provides the auxiliary DMA with a source/destination address in the DSP memory. Address decode is performed by the DMA to select the appropriate interface (data memory, program memory, register I/O, or external memory). The auxiliary channel of the DMA controller should be programmed for the highest priority in order to achieve the maximum throughput on the PCI interface. Figure 9–1 depicts the PCI interface to the auxiliary channel of the DMA controller in C62x/C67x.

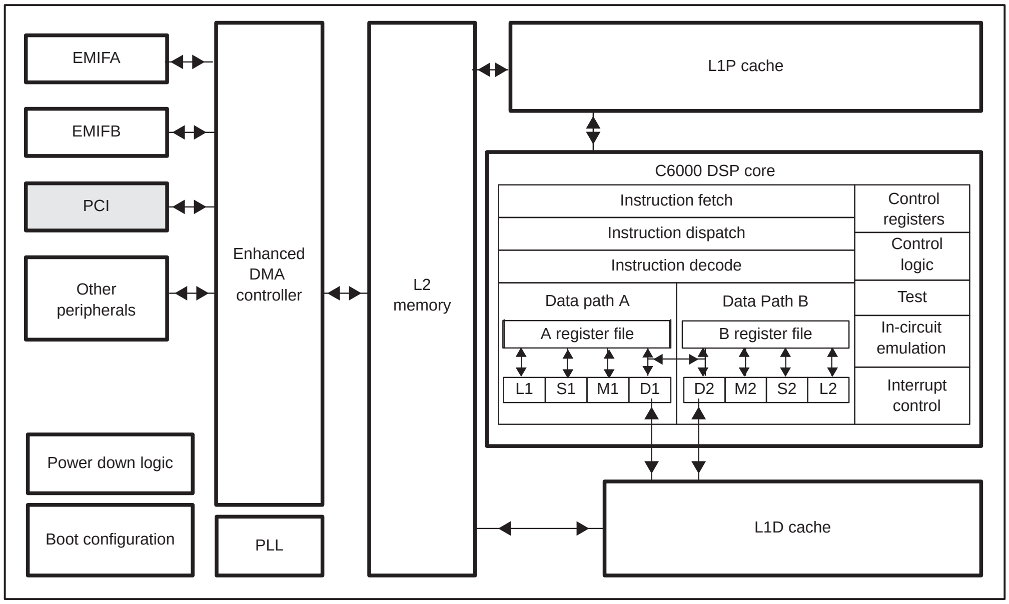
The C64x uses the EDMA internal address generation hardware to perform address decode instead. Figure 9–2 shows the C64x PCI interface to the EDMA.

Figure 9–1. TMS320C62x/C67x Block Diagram



Note: Refer to the specific device datasheet for its peripheral set.

Figure 9–2. TMS320C64x Block Diagram



Note: Refer to the specific device datasheet for its peripheral set.

Table 9–1 lists the differences between the C62x/C67x and C64x PCI ports.

Table 9–1. Differences Between the C62x/C67x and C64x PCI

Features	C62x/C67x PCI	C64x PCI	Section
Memory-mapped register location	01A4xxxxh and 01A8xxxxh address range	01C0xxxxh and 01C2xxxxh address range	9.3.3
Internal transfer	Auxiliary DMA	EDMA internal address generation hardware	9.1
Byte addressing	Word-aligned Master transactions	Doubleword-aligned master transactions	9.5
HALT register	Supported	Does not apply	9.9.7
Supported EEPROM size, bits	1K, 2K, 4K, 16K	4K	9.13
PCI bus clock. Minimum requirement.	$\geqq$ 4P†	$\geqq$ 8P†	9.1
Power management support	yes	no	9.15

† P= CPU clock period

9.2 PCI Architecture

The PCI port supports four types of PCI data transactions:

- **Slave Writes**—External PCI master writes to DSP slave
- **Slave Reads**—External PCI master reads from DSP slave
- **Master Writes**—DSP master writes to external slave
- **Master Reads**—DSP master reads from external slave

The PCI port block diagram, shown in Figure 9–3, consists of the following primary blocks:

- **PCI Bus Interface Unit**

  The PCI bus protocol is implemented in the PCI bus interface unit (PBIN). To maximize PCI bus bandwidth, the PCI interface does not insert wait states for slave or master burst transactions. If the corresponding FIFO goes full or empty, the PCI interface will disconnect the current transfer.

  The PCI bus interface unit will insert the following delays on the PCI bus:

  - **Slave Writes**:
    Zero wait state initial transfer
    Zero wait state subsequent transfers
    Disconnect if FIFO is full, or previous frame is not complete
  - **Slave Reads:**
    Prefetchable: Initial access disconnected with retry
    Up to 16 wait states inserted for single-word transfers
    Zero wait state initial transfer (prefetchable retry)
    Zero wait state subsequent transfers (prefetchable retry)
    Disconnect if FIFO is empty, or other slave read frame is in progress.
  - **Master Writes:**
    Zero wait state initial transfer
    Zero wait state subsequent transfers
  - **Master Reads:**
    Zero wait state initial transfer
    Zero wait state subsequent transfers

- **EEPROM Controller**

  The EEPROM controller interfaces to the 4-wire serial EEPROM interface. On PCI reset, this controller reads the EEPROM and provides the PCI bus interface unit with the configuration data. During normal operation, the EEPROM can be accessed by the DSP via the memory-mapped registers.

- **DSP Slave Write Block**

  The DSP slave write block contains a multiplexer, and a FIFO to transfer data (written by the external PCI master) from the PCI bus interface unit to the DSP.

- **DSP Slave Read Block**

  The DSP slave read block contains a multiplexer, and a FIFO to transfer data from the DSP to the PCI bus interface unit. The external PCI master is the requester of this data.

- **DSP Master Block**

  The DSP master block is divided into the read and write portions. The write portion of the DSP master block contains a data multiplexer, and a FIFO for DSP master writes. It transfers data from the DSP (master) to an external slave via the PCI bus interface unit. The read portion of the DSP master block contains a data multiplexer, and a FIFO for DSP master reads. It transfers data from the PCI bus interface unit to the DSP (master). The device cannot perform both read and write operations simultaneously.

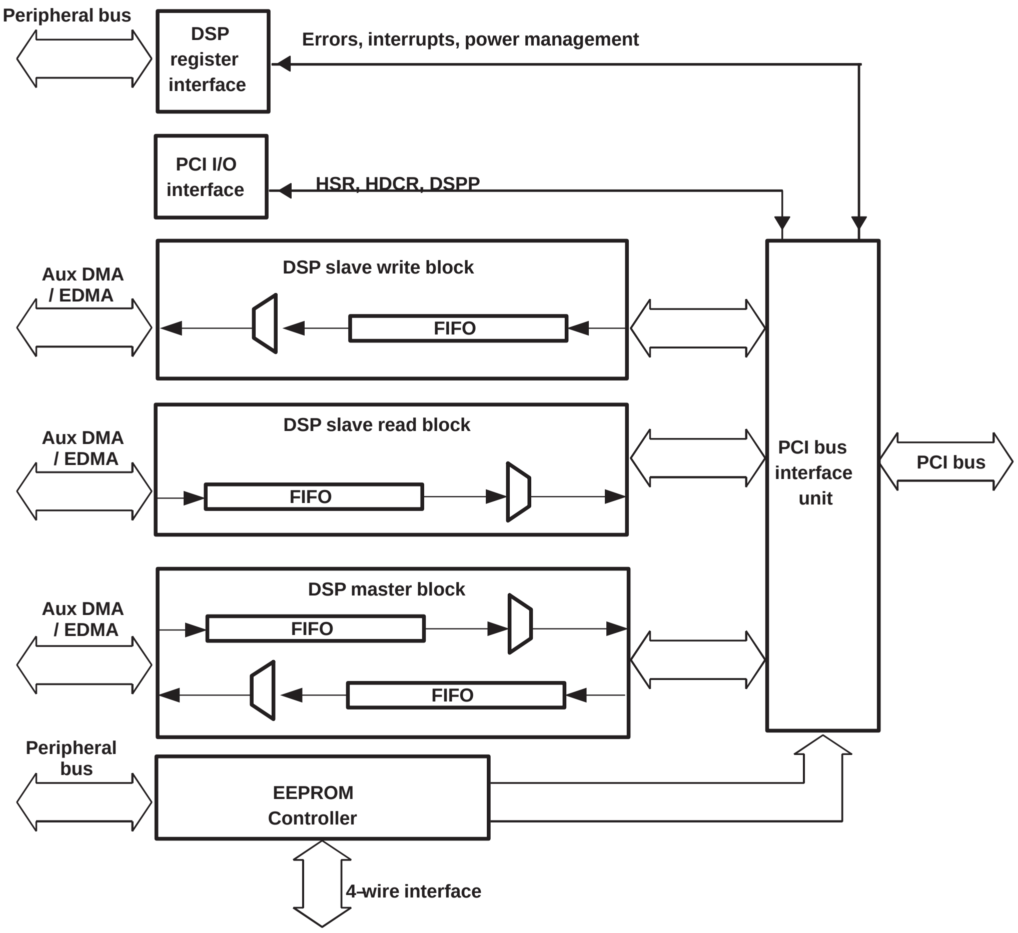
- **PCI I/O Interface**

  The I/O interface contains the PCI I/O mapped registers. These registers control the DMA/EDMA page for slave transactions, indicate the host status, and can interrupt or reset the DSP core.

- **DSP Register Interface**

  The DSP register interface contains DSP memory-mapped registers for the control of the master interface, PCI host interrupts, and power management.

Figure 9–3. PCI Port Block Diagram



9.3 PCI Registers

There are three types of PCI registers as follows:

- **PCI Configuration Registers**—accessible by external PCI host only.
- **PCI I/O Registers**—accessible by external PCI host only.
- **PCI Memory-Mapped Peripheral Registers**—accessible by DSP, and accessible by external PCI host via base address registers.

In addition, there are three resets related to the PCI registers. Each can have a different effect on the PCI registers, as indicated in the register descriptions in the following sections. These are the three resets:

- **$\overline{\text{RESET}}$**—the main chip reset pin
- **Warm Reset**—generated by the PCI host (or power management event)
- **$\overline{\text{PRST}}$**—the PCI reset signal.

9.3.1 PCI Configuration Registers (Accessible by External PCI Host Only)

The DSP supports all standard PCI configuration registers. These registers, which can only be accessed from the external host PCI, contain the standard PCI configuration information (vendor ID, device ID, class code, revision number, base addresses, power management, etc.).

Depending on the boot and device configuration settings at device reset, the PCI configuration registers can be autoloaded from an EEPROM at power-on reset or can be initialized with default values at power-on reset.

If no EEPROM is present, the PCI configuration registers will be initialized with default data (see section 9.16). If an EEPROM is present, and autoinitialization is configured, the PCI Configuration Registers cannot be properly accessed by the host until they are fully read from the EEPROM. PCI host access to the PCI configuration registers before the completion of autoinitialization will result in a disconnect with retry.

The CFGDONE and CFGERR bits in the PCI interrupt source register (PCIIS) and in the DSP reset source/status register (RSTSRC) indicate the status of the PCI configuration registers autoinitialization. See section 9.10.1 and section 9.10.3 for details. The PCI port asserts retries to prevent the host from performing any reads or writes to the PCI configuration registers until CFGDONE = 1, indicating that the PCI configuration registers have successfully been initialized with EEPROM or default values.

Table 9–2 shows the PCI configuration registers. Reads from the reserved fields return zeros, and writes to them have no effect. These registers conform to the PCI Specification revision 2.2. Refer to this specification and Power Management Specification revision 1.1 for more details on the registers and their operation.

The registers shaded in Table 9–2 can be autoloaded from the EEPROM at device power up. All registers are reset or loaded at PCI reset ($\overline{\text{PRST}}$).

In addition to the values indicated above, the EEPROM can also contain values for the power data, which can be selected by the power management control/status register (PMCSR). The power data stored in EEPROM are as follows:

- PC_D0 Power consumed D0
- PC_D1 Power consumed D1
- PC_D2 Power consumed D2
- PC_D3 Power consumed D3
- PD_D0 Power dissipated D0
- PD_D1 Power dissipated D1
- PD_D2 Power dissipated D2
- PD_D3 Power dissipated D3

Table 9–2. PCI Configuration Registers

Address	Byte 3	Byte 2	Byte 1	Byte 0
00h read only	Device ID		Vendor ID	
04h read/write	Status		Command	
08h read only	Class Code			Revision ID
0Ch read/write	Reserved	Header Type	Latency Timer	Cache Line Size
10h read/write	Base 0 Address (4 Mbyte prefetchable)			
14h read/write	Base 1 Address (8 Mbyte non-prefetchable)			
18h read only	Base 2 Address (4 words I/O)			
24h read only	Reserved			
2Ch read only	Subsystem ID		Subsystem Vendor ID	
30h read only	Reserved			
34h read only	Reserved			Capabilities Pointer
38h read only	Reserved			
3Ch read/write	Max Latency	Min_Grant	Interrupt Pin	Interrupt Line
40h read only	Power Management Capabilities		Next_Item_Ptr	Cap_ID
44h read only	Power Data	Reserved	Power Management Control/Status	
48h FFh	Reserved			

Note: Shaded registers can be autoloaded from EEPROM at autoinitialization.

See section 9.13.2 for details on the EEPROM memory map. See section 9.16 for a detailed description of the PCI configuration registers.

9.3.2 PCI I/O Registers (Accessible by External PCI Host Only)

The PCI I/O registers are located in the PCI host I/O space. They can only be accessed by the PCI host in the base 1 or base 2 address ranges specified in the base 1 address register and the base 2 address register, respectively. The base 1 address register and base 2 address register are PCI configuration registers, described in section 9.3.1. The locations of the different base addresses are discussed in section 9.4. All PCI I/O registers are byte-addressable. Table 9–3 shows the PCI I/O register locations accessed via the I/O base address (base 2 address).

Table 9–3. PCI I/O Registers Accessed via I/O Space (Base 2 Memory)

	Register/Port Accessed	
Address†	**Reads**	**Writes**
I/O Base Addr + 00h	HSR	HSR
I/O Base Addr + 04h	HDCR	HDCR
I/O Base Addr + 08h	DSPP	DSPP
I/O Base Addr + 0Ch	Reserved	Reserved

† I/O Base Addr is specified in the Base 2 Address Register. See section 9.3.1.

For processors that do not support I/O access, it can also access the PCI I/O registers via non-prefetchable reads and writes to the base 1 memory. The memory-mapping is as shown in Table 9–4. The DSP cannot access the I/O registers at these locations. They are accessible by the PCI host only. No memory reads/writes or DSP memory-mapped register access will occur at these locations for base 1 access. Base 0 access to these locations will be mapped to the DSP memory-mapped registers (not I/O registers).

Table 9–4. PCI I/O Registers Accessed via Base 1 Memory

C62x/C67x Address	C64x Address	Register/Port Accessed
0x01A7FFF0	0x01C1FFF0	HSR
0x01A7FFF4	0x01C1FFF4	HDCR
0x01A7FFF8	0x01C1FFF8	DSPP
0x01A7FFFC	0x01C1FFFC	Reserved

9.3.2.1 Host Status Register (HSR)

The host status register (HSR) is shown in Figure 9–4 and summarized in Table 9–5.

Figure 9–4. Host Status Register (HSR)

31 … 5	4	3	2	1	0
Reserved	EEREAD	CFGERR	INTAM	INTAVAL	INTSRC
HR, +0	HR, +x	HR, +0	HR/W,+1	HR, +0	HRW,+0

Table 9–5. Host Status Register (HSR) Bit Field Description

Bits	Name	Reset Source	Description
0	INTSRC	$\overline{\text{PRST}}$	PCI IRQ source active since last HSR clear. This bit, when 1, indicates that the DSP asserted the $\overline{\text{PINTA}}$ interrupt by writing the INTREQ bit in the RSTSRC register, and the INTAM bit in the HSR was a 0. This bit can be cleared by the PCI Host by writing a 1 to this bit. This will also negate the $\overline{\text{PINTA}}$ signal. Reads INTSRC = 0: $\overline{\text{PINTA}}$ was not asserted after last clear INTSRC = 1: $\overline{\text{PINTA}}$ was asserted after last clear Writes INTSRC = 0: no affect INTSRC = 1: deassert $\overline{\text{PINTA}}$

1	INTAVAL	$\overline{\text{PRST}}$	Indicates the current $\overline{\text{PINTA}}$ pin value. Writes to this bit have no effect. $\overline{\text{PINTA}}$ can be deasserted by the PCI host by writing the INTSRC field in the HSR with a 1 or by the DSP by writing a 1 to the INTRST field in the RSTSRC.
			INTAVAL = 0: $\overline{\text{PINTA}}$ is not asserted (inactive)
			INTAVAL = 1: $\overline{\text{PINTA}}$ is asserted (active)

2	INTAM	$\overline{\text{PRST}}$	$\overline{\text{PINTA}}$ mask. Disables DSP assertion of $\overline{\text{PINTA}}$. INTAM = 0: $\overline{\text{PINTA}}$ can be asserted by the DSP setting the INTREQ field in RSTSRC. INTAM = 1: $\overline{\text{PINTA}}$ will not be asserted. Only written by the PCI host (during D1, D2, D3 the $\overline{\text{PINTA}}$ is masked by the power management logic).

3	CFGERR	$\overline{\text{PRST}}$	Indicates an autoinitialization configuration error occurred
			CFGERR = 0: No error.
			CFGERR = 1: Autoinitialization configuration error.

4	EEREAD	$\overline{\text{PRST}}$	Indicates if the PCI configuration registers were initialized from EEPROM or not. EEREAD = 0: default values for configuration used EEREAD = 1: EEPROM values for configuration used

9.3.2.2 Host-to-DSP Control Register (HDCR)

The Host-to-DSP control register (HDCR) is shown in Figure 9–5 and summarized in Table 9–6.

Figure 9–5. Host-to-DSP Control Register (HDCR)

31 … 3	2	1	0
Reserved	PCIBOOT	DSPINT	WARMRESET
HR, +0	HR, +0	HW, +0	HW, +0

Table 9–6. Host-to-DSP Control Register (HDCR) Bit Field Description

Bits	Name	Reset Source	Description
0	WARMRESET	$\overline{\text{RESET}}$	DSP warm reset. Host write only. WARMRESET = 0: A write of 0 is ignored. WARMRESET = 1: Resets the DSP. The DSP will be held in reset for 16 PCI cycles. The DSP cannot be accessed until 16 PCI clocks after WARMRESET is written. WARMRESET only applies in D0. WARMRESET should not be used if the core is in power management state D1, D2 or D3 (I/O access is disabled in these states).
1	DSPINT	$\overline{\text{PRST}}$	DSP interrupt. Host writes only. Reads return 0. DSPINT = 0: A write of 0 is ignored. DSPINT = 1: Generates a host interrupt to the DSP. The interrupt will be generated to the core via the HOSTSW bit in the PCIIS register. If booting from the PCI interface, this interrupt will take the core out of reset. In all other cases, the DSP core must have its clock running and the HOSTSW bit in the PCIIEN register unmasked in order to latch the interrupt.
2	PCIBOOT	$\overline{\text{RESET}}$	PCI boot mode. Host read only. PCIBOOT = 0: Indicates DSP does not boot from the PCI. PCIBOOT = 1: Indicates DSP boot via the PCI.

9.3.2.3 DSP Page Register (DSPP) Bit Field Description

The DSP page register (DSPP) is shown in Figure 9–6 and summarized in Table 9–7.

Figure 9–6. DSP Page Register (DSPP)

31 … 11	10	9 … 0
Reserved	MAP	PAGE
HR, +0	HR, +x	HRW, +0

Table 9–7. DSP Page Register (DSPP) Bit Field Description

Bits	Name	Reset Source	Description
9:0	PAGE	$\overline{\text{PRST}}$	Locates a 4M-byte memory window within DSP address map for prefetchable (base 0) memory accesses.

10	MAP	$\overline{\text{RESET}}$	Indicates which memory map is being used by the DSP (C62x/C67x only)
			MAP = 0: Map 0
			MAP = 1: Map 1

9.3.3 PCI Memory-Mapped Peripheral Registers

Table 9–8 shows all the registers that are mapped into DSP space for DSP control/observation of the PCI interface. These registers can be accessed by both the DSP and the PCI host.

Table 9–8. PCI Memory-Mapped Peripheral Registers

DSP Data Space Address		Register Accessed		
C62x/C67x	**C64x**	**Reads**	**Writes**	**Section**
01A4 0000h	01C0 0000h	RSTSRC	RSTSRC	9.10.3
01A4 0004h	01C0 0004h	PMDCSR	PMDCSR	9.15.4
01A4 0008h	01C0 0008h	PCIIS	PCIIS	9.10.1
01A4 000Ch	01C0 000Ch	PCIIEN	PCIIEN	9.10.2
01A4 0010h	01C0 0010h	DSPMA	DSPMA	9.9.1
01A4 0014h	01C0 0014h	PCIMA	PCIMA	9.9.2
01A4 0018h	01C0 0018h	PCIMC	PCIMC	9.9.3
01A4 001Ch	01C0 001Ch	CDSPA	–	9.9.4
01A4 0020h	01C0 0020h	CPCIA	–	9.9.5
01A4 0024h	01C0 0024h	CCNT	–	9.9.6
01A4 0028h	01C0 0028h	HALT†	HALT†	9.9.7
01A8 0000h	01C2 0000h	EEADD	EEADD	9.13.4
01A8 0004h	01C2 0004h	EEDAT	EEDAT	9.13.4
01A8 0008h	01C2 0008h	EECTL	EECTL	9.13.4

† HALT register applies to C62x/C67x only.

9.4 TMS320C6000/PCI Memory Map

The PCI port has full visibility into the DSP's memory map via three base address registers:

- **Base 0:** 4M-byte prefetchable maps to all of DSP memory with the DSP Page register. Prefetch reads have all bytes valid.
- **Base 1**: 8M-byte non-prefetchable maps to DSP's memory-mapped registers. Non-prefetch supports byte enables.
- **Base 2:** 16-byte I/O contains I/O registers for the PCI host

These three registers belong to the group of PCI configuration registers. PCI host accesses to DSP (prefetchable) memory are mapped to a 4M-byte window in the PCI memory space. The PCI port contains a PCI I/O register — the DSP page register (DSPP) — that specifies the address mapping from the PCI address to the DSP address. This address mapping is used when the DSP is a slave on the PCI local bus.

The DSPP, described in section 9.3.2.3, is used to locate the 4M-byte window within DSP's memory map. Bits 21:0 of the PCI address are concatenated with bits 9:0 of the DSPP register to form the DSP address for PCI slave access to the DSP. This is shown in Figure 9–7.

Figure 9–7. PCI Base Slave Address Generation (Prefetchable)

31 22	21 0
DSPP Register (9:0)	Current PCI Address 21:0

The PCI base 1 register on the DSP is configured for an 8M-byte non-prefetchable region. This memory is mapped into the DSP memory-mapped registers (0180 0000h). Bits 22:0 of the PCI address are concatenated with a fixed offset to map the base 1 access into the memory mapped registers. This is shown in Figure 9–8.

Figure 9–8. PCI Base 1 Slave Address Generation (Non-prefetchable)

31 23	22 0
0000 0001 1	Current PCI Address 22:0

Base address register 2 is configured for a 16-byte I/O region for the PCI host to access the PCI I/O registers. See section 9.3.2.

The PCI bus interface provides two access methods for PCI host access to DSP’s memory. The 4M-byte Base 0 region is used for prefetchable data, and the 8M-byte base 1 region is used for non-prefetchable (register) access. All transfers to the non-prefetchable region will transfer single words and then disconnect.

Data access to the prefetchable region may be transferred in bursts limited mainly by the host system setup (PCI bridge latency timer, burst length count). Prior to transferring data, the PCI host must first write the DSP page register (DSPP) to locate the 4M-byte window within DSP’s memory map.

PCI master transactions issued by the DSP will attempt using bursts. Through disconnects, however, the external slave can force the DSP master to perform single-word transfers.

Internal to the DSP, all data transfers are handled by the auxiliary channel of the DMA controller (C62x/C67x) or the EDMA internal address generation hardware (C64x).

Note: Users must ensure that no PCI transactions will cross port boundaries of the DMA/EDMA controller. A port boundary is the address boundary between external memory and internal memory, between external memory and the peripheral address space, or between internal memory and the peripheral address space.

Note: Users must ensure that no PCI transactions will cross port boundaries of the DMA/EDMA controller. A port boundary is the address boundary between external memory and internal memory, between external memory and the peripheral address space, or between internal memory and the peripheral address space.

9.5 Byte Addressing

The PCI interface is byte-addressable. The PCI interface can read and write 8-bit bytes, 16-bit halfwords, 24-bit words, and 32-bit words. Words are aligned on an even four-byte boundary. Words always start at a byte address where the two LSBs are 00. Halfwords always start at a byte address where the last LSB is 0.

PCI slave transactions are fully byte-addressable.

PCI master transactions must start on a word-aligned address for C62x/C67x. For C64x, PCI master transactions must start on a doubleword-aligned address.

9.6 PCI Address Decode

The TMS320C6000 PCI port supports “medium” address decode of the PCI address for memory and I/O cycles. The $\overline{\text{PDEVSEL}}$ signal is asserted two PCI clock periods after $\overline{\text{PFRAME}}$ is sampled and asserted.

9.7 PCI Transfers to/from Program Memory (TMS320C62x/C67x)

The CPU has priority over the DMA (and auxiliary DMA) for access to program memory. Since the CPU can access the program memory on every CPU clock, it can possibly lock out the DMA from accessing the program memory.

If the PCI port is requesting program memory transfers via the auxiliary channel, all the other four DMA channels are halted. Thus, no DMA channel activity will occur during the PCI port requests, even if a DMA channel is accessing a different memory (EMIF/peripheral/data memory). If the CPU has higher priority, all DMA activity can be blocked while the CPU is executing a tightly coded routine from program memory.

The HALT register prevents this situation. The HALT register prevents the PCI port from performing master/slave auxiliary channel requests. If the HALT bit is set, all auxiliary transfers are prevented. Any current PCI master transaction will complete its DMA cycle. The PCI transaction will not commence until the HALT bit is deasserted. This prevents DMA lockup when a PCI transaction is in progress and the DSP is executing a packed section of code. The other DMA channels are free to access memory when the PCI master is halted.

The HALT register does not apply to C64x, because the DMA lockup condition does not apply. The C64x EDMA uses a priority queue implementation. Only EDMA transfers placed in the same priority queue as the PCI transfer will be stalled in the above condition. Refer to *Chapter 6, EDMA Controller,* for details on priority queues.

9.8 Slave Transfers

9.8.1 DSP Memory Slave Writes

The slave write FIFO in the DSP slave write block (section 9.2, Figure 9–3) is used to efficiently handle PCI host writes to the DSP slave.

The address for a DSP slave write is derived from the PCI address concatenated with the fixed offset in the DSPP register, as described in section 9.4. No wait states will be inserted by the slave PCI port. DSP slave writes execute with zero wait states on all data phases for both single and burst accesses. The PCI interface supports unlimited length memory burst transfers.

Slave write access to DSP will only be disconnected when the FIFO is full, or when the FIFO is not empty from a previous PCI slave write frame. Slave reads and master read/write transactions have no effect on slave write PCI transactions.

Internally, the auxiliary DMA or EDMA internal address generation hardware service the slave write FIFO when:

❑ FIFO is at least half-full
❑ PCI transaction has terminated ($\overline{\text{PFRAME}}$ deasserted).

The DSP slave write address is autoincremented internally. DSP memory writes continue until there are no longer any valid data in the FIFO. This applies to both single and burst PCI transactions. For single access transactions, the internal transfer request will be made after the PCI transaction has terminated.

9.8.2 DSP Memory Slave Reads

Similar to slave writes, the slave read FIFO in the DSP slave read block (section 9.2, Figure 9–3) is used to efficiently handle PCI host reads from the DSP slave. The PCI slave read interface supports unlimited length memory burst transfers. All PCI slave accesses to DSP must be 32-bit word-aligned.

The PCI port uses the cache line size and PCI command to determine the number of bytes transferred for a slave read. The type of PCI access is indicated by the $\overline{\text{PCBEx}}$ signals during the address phase. The following slave read commands are supported:

❑ Memory read
❑ Memory read multiple
❑ Memory read line

All of the above PCI slave reads can be prefetchable and non-prefetchable.

9.8.2.1 Non-Prefetchable Slave Reads

For non-prefetchable slave Reads, the PCI port inserts wait cycles until a single word is written to the FIFO. The word is then transferred on the PCI bus, and the cycle is terminated regardless if the command was a memory read, a memory read multiple or a memory read line.

9.8.2.2 Prefetchable Slave Reads

Prefetchable Memory Read

For these reads, the PCI port inserts wait cycles until the requested word is ready. The PCI port adheres to the 16-clock rule. If data is not ready in 16 PCI clocks, the memory read is disconnected with retry.

Prefetchable Memory Read Multiple and Prefetchable Memory Read Line

These requests are initially terminated on the PCI bus with a disconnect with retry. Subsequently, the auxiliary DMA or EDMA internal address generation hardware services the PCI port by transferring read data to the FIFO. The PCI slave transfer occurs when the original master re-attempts the initial transfer. Non-related slave requests are terminated with a disconnect.

For memory read line commands, the number of bytes transferred to the slave read FIFO is based on the cache-line size register. For memory read multiple commands, the DSP slave read will continuously fill the FIFO with data until the PCI master terminates the transaction ($\overline{\text{PFRAME}}$ de-asserted), at which point the last PCI valid data sample is transferred and the FIFO is flushed.

The auxiliary DMA or EDMA internal address generation hardware bursts until the slave read FIFO is full. Subsequent burst size is half of the FIFO length, or the number of words left to be transferred.

Memory read line and memory read multiple commands transfer data with zero wait states inserted by the DSP PCI port, when the requestor retries the command and the FIFO has data. The FIFO request is also terminated if the PCI transaction is disconnected prematurely by the master.

9.8.3 PCI Target Initiated Termination

The DSP issues target terminations under these conditions:

- ❑ Data transfer with disconnect if the master issues a burst memory access with an addressing mode that is not supported.
- ❑ Retry with data transfer if the master attempts a burst access to the configuration space (see section 9.3.1)
- ❑ Retry with data transfer if the master attempts a burst access to the I/O space (see section 9.3.2).
- ❑ Disconnect for slave memory or I/O writes and a transaction is waiting in the internal slave read FIFO.
- ❑ Disconnect for slave memory reads if the PCI address value does not match the address in the internal read prefetch buffer.
- ❑ Once a prefetch has been started, retry for all other memory and I/O reads until the original posted transaction is repeated by the PCI bus master and the prefetched data is transferred.

The PCI interface meets all 16-clock and 8-clock rules for data transfers within single access and burst accesses.

9.9 Master Transfers

Master transfers are initiated under DSP control. The following PCI memory-mapped peripheral registers are used to configure a DSP master transfer:

❑ DSP master address register (DSPMA)
❑ PCI master address register (PCIMA)
❑ PCI master control register (PCIMC)

The following PCI memory-mapped peripheral registers indicate the status of the current master transfer:

❑ Current DSP address register (CDSPA)
❑ Current PCI address register (CPCIA)
❑ Current byte count register (CCNT).

For C62x/C67x, the PCI transfer halt register (HALT) allows the DSP to terminate internal transfer requests to the auxiliary DMA channel.

9.9.1 DSP Master Address Register (DSPMA)

The DSP master address register (DSPMA) contains the DSP's address for the location of destination data for DSP master reads, or the address location of source data for DSP master writes. The register also contains bits to control the address modification. The DSPMA is shown in Figure 9–9 and summarized in Table 9–9.

Figure 9–9. DSP Master Address Register (DSPMA)

31 … 2	1	0
ADDRMA	AINC	Rsvd
RW, +0	RW, +0	R, +0

Table 9–9. DSP Master Address Register (DSPMA) Bit Field Description

Bits	Name	Reset Source	Description

1	AINC	$\overline{\text{RESET}}$ WARM	Autoincrement mode of DSP master address
			AINC = 0: Autoincrement of ADDRMA enabled
			AINC = 1: ADDRMA will not autoincrement

31:2	ADDRMA	$\overline{\text{RESET}}$ WARM	DSP's word address for PCI master transactions

9.9.2 PCI Master Address Register (PCIMA)

The PCI master address register (PCIMA) contains the PCI word address (C62x/C67x) or doubleword address (C64x). For DSP master reads, PCIMA contains the source address. For DSP master writes, the PCIMA contains the destination address. The PCIMA is shown in Figure 9–10 and summarized in Table 9–10.

Figure 9–10. PCI Master Address Register (PCIMA)

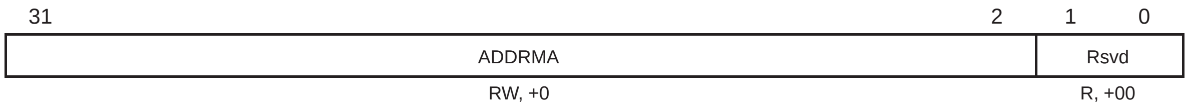


Table 9–10. PCI Master Address Register (PCIMA) Bit Field Description

Bits	Name	Reset Source	Description
31:2	ADDRMA	$\overline{\text{RESET}}$ WARM	PCI word address (C62x/C67x) or doubleword address (C64x) for PCI master transactions.

9.9.3 PCI Master Control Register (PCIMC)

The PCI master control register (PCIMC) contains:

- ❑ Start bits to initiate the transfer between DSP and PCI
- ❑ The transfer count, which specifies the number of bytes to transfer (65K bytes maximum)
- ❑ Reads indicate transfer status

The PCIMC is shown in Figure 9–11 and summarized in Table 9–11.

Figure 9–11. PCI Master Control Register (PCIMC)

31 – 16	15 – 2	1 – 0
CNT	Reserved	START
RW, +0000h	R, +0	RW, +00

Table 9–11. PCI Master Control Register (PCIMC) Bit Field Description

Bits	Name	Reset Source	Description
1:0	START	$\overline{\text{RESET}}$ WARM $\overline{\text{PRST}}$	Start the read or write master transaction START = 00b: transaction not started/flush current transaction START = 01b: Start a master write transaction START = 10b: Start a master read transaction to prefetchable memory START = 11b: Start a master read transaction to non-prefetchable memory START will return to 00b when the transaction is complete. If the START command is changed during a transfer, the transfer will stop and the FIFOs will be flushed. If the PCI bus is reset during a transfer, the transfer will stop and the FIFOs will be flushed. (A CPU interrupt can be generated on a $\overline{\text{PRST}}$ transition.) START will only get set if bits 31:16 ≠ 0000h
31:16	CNT	$\overline{\text{RESET}}$ WARM	Transfer Count . It specifies the number of bytes to transfer

9.9.4 Current DSP Address Register (CDSPA)

The current DSP address register (CDSPA) contains the current DSP address for master transactions. The CDSPA is shown in Figure 9–12 and summarized in Table 9–12.

Figure 9–12. Current DSP Address (CDSPA)

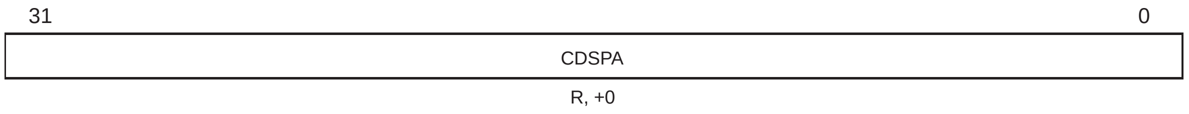


Table 9–12. Current DSP Address (CDSPA) Bit Field Description

Bits	Name	Reset Source	Description
31:0	CDSPA	$\overline{\text{RESET}}$ WARM	The current DSP address for master transactions

9.9.5 Current PCI Address Register (CPCIA)

The current PCI address register (CPCIA) contains the current PCI address for master transactions. The CPCIA is shown in Figure 9–13 and summarized in Table 9–13.

Figure 9–13. Current PCI Address Register (CPCIA)

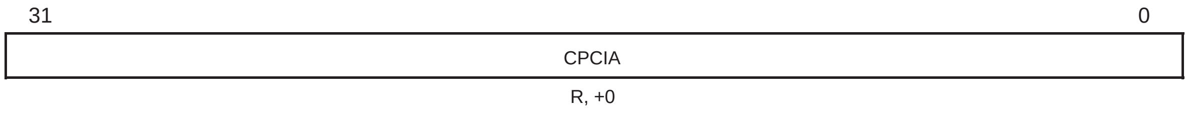


Table 9–13. Current PCI Address Register (CPCIA) Bit Field Description

Bits	Name	Reset Source	Description
31:0	CPCIA	$\overline{\text{RESET}}$ WARM $\overline{\text{PRST}}$	The current PCI address for master transactions

9.9.6 Current Byte Count Register (CCNT)

The current byte count register CCNT contains the number of bytes left on the current master transaction. The CCNT is shown in Figure 9–14 and summarized in Table 9–14.

Figure 9–14. Current Byte Count Register (CCNT)

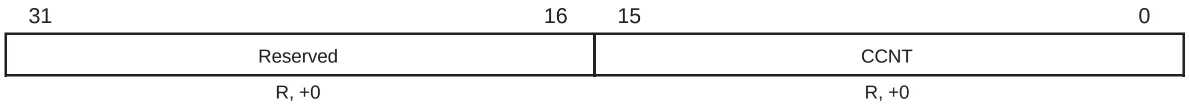


Table 9–14. Current Byte Count Register (CCNT) Bit Field Description

Bits	Name	Reset Source	Description
15:0	CCNT	$\overline{\text{RESET}}$ WARM $\overline{\text{PRST}}$	The number of bytes left on the master transaction.

9.9.7 PCI Transfer Halt Register (HALT) — C62x/C67x only

The PCI transfer halt register (HALT) allows the C62x/C67x DSP to terminate internal transfer requests to the auxiliary DMA channel. The HALT register is shown in Figure 9–15 and summarized in Table 9–15.

Figure 9–15. PCI Transfer Halt Register (HALT)

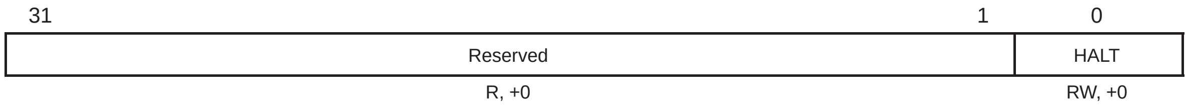


Table 9–15. PCI Transfer Halt Register (HALT) Bit Field Description

Bits	Name	Reset Source	Description
0	HALT	$\overline{\text{RESET}}$ WARM $\overline{\text{PRST}}$	Halt internal transfer requests HALT = 0: No effect. HALT = 1: HALT prevents the PCI port from performing master/slave auxiliary DMA transfer requests.

9.9.8 DSP Master Writes

The master write FIFO in the DSP master block (see section 9.2, Figure 9–3) is used to efficiently handle DSP master writes to an external slave. The master write interface supports burst lengths of up to 65K bytes.

Master writes are initiated under DSP control via the DSP master address register (DSPMA), the PCI master address register (PCIMA), and the PCI master control register (PCIMC).

For DSP master writes, the ADDRMA field in the DSPMA contains the word-aligned source address (DSP address). If AINC = 0 in the DSPMA, the source address is autoincremented by 4 bytes after each internal data transfer. The PCIMA contains the word-aligned (C62x/C67x) or doubleword-aligned (C64x) destination address (PCI address). An internal register keeps track of the PCI master address.

A master write is initiated by enabling the START bits in the PCIMC. The auxiliary DMA or EDMA transfers data from the source address (pointed to by DSPMA) to the master write FIFO. It either fills up the FIFO or transfers only the number of words desired if that number is less than the FIFO length. Subsequent internal data transfers are performed when the FIFO is half-full or less. Internal data transfer continues until the FIFO is full or until the transfer is completed.

Once the FIFO has valid data, a PCI bus request is made and data is transferred from the FIFO to the PCI slave. DSP master writes execute with zero wait states on all data phases of both single and burst accesses. The PCI command/byte enable signals ($\overline{\text{PCBEx}}$) indicate the master write bytes on the PCI interface.

Internal data transfers will stop once all master write data has been transferred from the DSP source to the master write FIFO. For C62x/C67x, internal data transfers and the current PCI bus cycle can also be terminated by asserting the HALT bit in the HALT register. The PCI bus Interface monitors the PCI interface for disconnects, retries and target-aborts. The PCI port complies with PCI Specification revision 2.2 and retries the exact same cycle during retries.

If the cycle is terminated with a master abort or a target abort, the current transfer will be terminated internally and externally on both the PCI bus. The master Write FIFO is flushed and either the master abort (PCIMASTER) or target abort (PCITARGET) bit will be set in PCI interrupt source register (PCIIS). These error conditions can generate a CPU interrupt if the corresponding bits are set in the PCI interrupt enable register (PCIIEN).

If the PCI latency timer (specified in the PCI configuration register space, section 9.3.1) times out, the PCI master will give up the bus. The master will request the bus later, and complete the necessary transfers.

The progress of the transfer can be polled by reading the PCI master control register (PCIMC). The START field goes to 00b when the transfer is complete on both the DSP and PCI side. Alternatively, the master can be programmed to generate an interrupt upon completion of a frame transfer by setting the MASTEROK bit in the PCIIEN.

9.9.9 DSP Master Reads

The master read FIFO in the DSP master block (section 9.2, Figure 9–3) is used to efficiently handle DSP master reads from an external slave. The master read interface supports burst lengths of up to 64K bytes.

Master reads are initiated under DSP control via the DSP master address register (DSPMA), the PCI master address register (PCIMA), and the PCI master control register (PCIMC).

For DSP master reads, the PCIMA contains the external PCI slave source address. The ADDRMA field in the DSPMA contains the word-aligned destination address (DSP address). If AINC = 0 in the DSPMA, the destination address is autoincremented by 4 bytes after each internal data transfer.

A master read is initiated by enabling the START bits in the PCIMC. The PCI port performs a PCI bus request. Once a PCI bus request is granted, a PCI bus cycle is initiated. The type of cycle initiated depends on the number of bytes to be transferred and the cache line size. The following master read commands are supported:

- ❑ Memory read
- ❑ Memory read multiple
- ❑ Memory read line

The user can initiate two types of reads, based on the START bits in the PCIMC. Prefetchable reads (START = 10b) use the memory read multiple and memory read line commands for transfers greater than one word. A memory read command is used for transfers of one word.

Non-prefetchable reads (START = 11b) always use a memory read command. A transfer size of N words is broken up into N one-word read cycles on the PCI bus. Users should read from prefetchable memory whenever possible.

No wait states are inserted on the initial data phase or subsequent data phases of the PCI master read access. Read data is written to the master read FIFO. An internal auxiliary DMA or EDMA data transfer request occurs when the FIFO is at least half full, or when the PCI transaction has terminated. The auxiliary DMA channel or EDMA transfers data from the master read FIFO to the DSP destination address (ADDRMA field in the DSPMA). All master read transactions are word-aligned for C62x/C67x, and doubleword-aligned for C64x.

Auxiliary DMA or EDMA transfers continue until there are no longer any valid data in the FIFO. This applies to both single access and burst PCI transactions. For single access transactions, the internal data transfer occurs after the PCI transaction has terminated ($\overline{\text{PFRAME}}$ deasserted).

Internal data transfers will stop once all master read data has been transferred from the master read FIFO to the DSP destination. For C62x/C67x, internal data transfers and the current PCI bus cycle can also be terminated by asserting the HALT bit in the HALT register. The PCI bus interface monitors the PCI interface for disconnects, retries and target-aborts. The PCI port complies with PCI Specification revision 2.2 and retries the exact same cycle during retries.

If the cycle is terminated with a master abort or a target abort, the current transfer will be terminated both internally and externally on the PCI bus. The master Read FIFO is flushed and either the master abort (PCIMASTER) or target abort (PCITARGET) bit will be set in PCI interrupt source register (PCIIS). These error conditions can generate a CPU interrupt if the corresponding bits are set in the PCI interrupt enable register (PCIIEN).

If the PCI latency timer (specified in the PCI configuration register space, section 9.3.1) times out, the PCI master will give up the bus. The master will request the bus later, and complete the necessary transfers.

The progress of the transfer can be polled by reading the PCI master control register (PCIMC). Alternatively, the master can be programmed to generate an interrupt upon completion of frame transfer by setting the MASTEROK bit in the PCIIEN.

9.10 Interrupts and Status Reporting

The PCI port can generate the following CPU interrupts:

- **PCI_WAKEUP:** This is dedicated to power wake-up events.
- **ADMA_HLT (C62x/C67x only):** This interrupt is generated when the auxiliary DMA is halted (DMAHALTED). This interrupt only applies to the C62x/C67x.
- **DSPINT:** This interrupt is asserted, and enabled in the PCI interrupt enable register (PCIIEN), when any of these events occurs: PWRMGMT, PCITRAGET, PCIMASTER, HOSTSW, PWRLH, PWRHL, MASTEROK, CFGDONE, CFGERR, EERDY, $\overline{\text{PRST}}$.

The PCI master/slave interface status/errors are shown in the PCI interrupt source register (PCIIS). All status/error conditions can generate a CPU interrupt if they are enabled in the DSP interrupt enable register (PCIIEN). Status bits in the PCIIS will still be set, even if the interrupt is not enabled. If an enabled interrupt occurs, it sends a DSPINT interrupt to the DSP. Writing a 1 to the corresponding PCIIS bit(s) clears the interrupt. If an interrupt bit is still set, a new interrupt will then occur. Thus, upon a PCI interrupt, the user should perform the following in the interrupt service routine:

- Read the PCIIS register
- Clear the appropriate PCIIS bits by writing a 1 to them

9.10.1 PCI Interrupt Source Register (PCIIS)

The PCI interrupt source register (PCIIS) shows the status of the interrupt sources. Writing a 1 to the bit(s) clears the condition. Writes of zero to, and reads from, the bit(s) have no effect. The PCIIS is shown in Figure 9–16 and summarized in Table 9–16.

Figure 9–16. PCI Interrupt Source Register (PCIIS)

31 13	12	11	10	9	8	7	6	5	4	3	2	1	0
Rsvd	DMA HALTED	$\overline{\text{PRST}}$	Rsvd	EERDY	CFG ERR	CFG DONE	MASTER OK	PWRHL	PWRLH	HOSTSW	PCI MASTER	PCI TARGET	PWR MGMT
R, +0	RW, +0	RW, +0	R, +0	RW, +0	RW, +0	RW, +0	RW, +0	RW, +0	RW, +0	RW, +0	RW, +0	RW, +0	RW, +0

Table 9–16. PCI Interrupt Source Register (PCIIS) Bit Field Description

Bits	Name	Reset Source	Description
0	PWRMGMT	$\overline{\text{RESET}}$ WARM	PWRMGMT = 0: No power management state transition interrupt PWRMGMT = 1: Power management state transition interrupt (will not be set if the DSP clocks are not running)
1	PCITARGET	$\overline{\text{RESET}}$ WARM	PCITARGET = 0: No target abort received PCITARGET = 1: Target abort received
2	PCIMASTER	$\overline{\text{RESET}}$ WARM	PCIMASTER = 0: No master abort received PCIMASTER = 1: Master abort received
3	HOSTSW	$\overline{\text{RESET}}$ WARM	HOSTSW = 0: No host software requested interrupt HOSTSW = 1: Host software requested interrupt (this bit must be set after boot from PCI to wake up DSP)

4	PWRLH	$\overline{\text{RESET}}$
		WARM
			PWRLH = 0: No low-to-high transition on PWRWKP
			PWRLH = 1: Low-to-high transition on PWRWKP

5	PWRHL	$\overline{\text{RESET}}$
		WARM
			PWRHL = 0: No high-to-low transition on PWRWKP
			PWRHL = 1: High-to-low transition on PWRWKP

6	MASTEROK	$\overline{\text{RESET}}$ WARM	MASTEROK = 0: No PCI master transaction complete interrupt MASTEROK = 1: PCI master transaction complete interrupt

7	CFGDONE	$\overline{\text{RESET}}$
		WARM
			CFGDONE = 0: Configuration of PCI Configuration Registers not complete
			CFGDONE = 1: Configuration of PCI Configuration Registers is complete.
			– set after an initialization due to $\overline{\text{PRST}}$ asserted.
			– set after WARM if initialization has been done

Table 9–16. PCI Interrupt Source Register (PCIIS) Bit Field Description (Continued)

Bits	Name	Reset Source	Description

8	CFGERR	$\overline{\text{RESET}}$
		WARM
			CFGERR = 0: No Checksum failure during PCI autoinitialization
			CFGERR = 1: Checksum failed during PCI autoinitialization.
			– set after an initialization due to $\overline{\text{PRST}}$ asserted and checksum error
			– set after WARM if initialization has been done, but had checksum error.

9	EERDY	$\overline{\text{RESET}}$ WARM	EERDY = 0: The EEPROM is not ready to accept a new command EERDY = 1: The EEPROM is ready to accept a new command and the data register can be read.
11	$\overline{\text{PRST}}$	$\overline{\text{RESET}}$ WARM	$\overline{\text{PRST}}$ = 0: No change of state on PCI reset $\overline{\text{PRST}}$ = 1: PCI reset (/PRST) changed state
12	DMA- HALTED	$\overline{\text{RESET}}$ WARM	DMAHALTED = 0: Auxiliary DMA transfers are not halted. DMAHALTED = 1: Auxiliary DMA transfers have stopped (C62x/C67x only, reserved on C64x)

9.10.2 PCI Interrupt Enable Register (PCIIEN)

The bits in the PCI interrupt enable register PCIIEN enable the PCI interrupts. In order for the DSP to see the interrupts, the DSP software must also set the appropriate bits in the control status register (CSR) and interrupt enable register (IER).

The only interrupt enabled after device reset ($\overline{\text{RESET}}$) is the HOSTSW interrupt. In this way, the PCI host can wake up the DSP by writing the DSPINT bit in the Host-to-DSP control register (HDCR). The PCIIEN is shown in Figure 9–17 and summarized in Table 9–17.

Figure 9–17. PCI Interrupt Enable Register (PCIIEN)

31 12	11	10	9	8	7	6	5	4	3	2	1	0
Rsvd	$\overline{\text{PRST}}$	Rsvd	EERDY	CFG ERR	CFG DONE	MASTER OK	PWRHL	PWRLH	HOSTSW	PCI MASTER	PCI TARGET	PWR MGMT
R, +0	RW, +0	R, +0	RW, +0	RW, +0	RW, +0	RW, +0	RW, +0	RW, +0	RW, +1	RW, +0	RW, +0	RW, +0

Table 9–17. PCI Interrupt Enable Register (PCIIEN) Bit Field Description

Bits	Name	Reset Source	Description
0	PWRMGMT	$\overline{\text{RESET}}$ WARM	PWRMGMT = 0: Power management state transition interrupts not enabled PWRMGMT = 1: Power management state transition interrupt enabled
1	PCITARGET	$\overline{\text{RESET}}$ WARM	PCITARGET = 0: PCI target abort interrupt not enabled PCITARGET = 1: PCI target abort interrupt enabled
2	PCIMASTER	$\overline{\text{RESET}}$ WARM	PCIMASTER = 0: PCI master abort interrupt not enabled PCIMASTER = 1: PCI master abort interrupt enabled
3	HOSTSW	$\overline{\text{RESET}}$ WARM	HOSTSW = 0: Host software requested interrupts not enabled HOSTSW = 1: Host software requested interrupt enabled

4	PWRLH	$\overline{\text{RESET}}$
		WARM
			PWRLH = 0: Low-to-high PWRWKP interrupts not enabled
			PWRLH = 1: Low-to-high PWRKWP interrupts enabled

5	PWRHL	$\overline{\text{RESET}}$
		WARM
			PWRHL = 0: High-to-low PWRWKP interrupts not enabled
			PWRHL = 1: High-to-low PWRWKP interrupts enabled

6	MASTEROK	$\overline{\text{RESET}}$ WARM	MASTEROK = 0: PCI master transaction complete interrupts not enabled MASTEROK = 1: PCI master transaction complete interrupts enabled
7	CFGDONE	$\overline{\text{RESET}}$ WARM	CFGDONE = 0: Configuration complete interrupts not enabled CFGDONE = 1: Configuration complete interrupts enabled
8	CFGERR	$\overline{\text{RESET}}$ WARM	CFGERR = 0: Configuration error interrupts not enabled CFGERR = 1: Configuration error interrupts enabled

Table 9–17. PCI Interrupt Enable Register (PCIIEN) Bit Field Description (Continued)

Bits	Name	Reset Source	Description
9	EERDY	$\overline{\text{RESET}}$ WARM	EERDY = 0: EEPROM ready interrupts not enabled EERDY = 1: EEPROM ready interrupts enabled
11	$\overline{\text{PRST}}$	$\overline{\text{RESET}}$ WARM	$\overline{\text{PRST}}$ = 0: $\overline{\text{PRST}}$ transition interrupts not enabled $\overline{\text{PRST}}$ = 1: $\overline{\text{PRST}}$ transition interrupts enabled

9.10.3 DSP Reset Source/Status Register (RSTSRC)

The DSP reset source/status register (RSTSRC) shows the reset status of the DSP. It gives the DSP visibility to which reset source caused the last reset. The RSTSRC is shown in Figure 9–18 and summarized in Table 9–18. The $\overline{\text{RST}}$,. $\overline{\text{PRST}}$, and WARMRST fields are cleared by a read of the RSTSRC.

Figure 9–18. DSP Reset Source/Status Register (RSTSRC)

31 7	6	5	4	3	2	1	0
Reserved	CFGERR	CFGDONE	INTRST	INTREQ	WARMRST	$\overline{\text{PRST}}$	$\overline{\text{RST}}$
R, +0	R, +0	R, +0	W, +0	W, +0	R, +0	R, +0	R, +1

Table 9–18. DSP Reset Source/Status Register (RSTSRC) Bit Field Description

Bits	Name	Reset Source	Description
0	$\overline{\text{RST}}$	$\overline{\text{RESET}}$	Indicates a device reset ($\overline{\text{RESET}}$) occurred since last RSTSRC read. Cleared by read of RSTSRC. Writes have no effect. $\overline{\text{RST}}$ = 0: No device reset ($\overline{\text{RESET}}$) since last RSTSRC read $\overline{\text{RST}}$ = 1: device reset ($\overline{\text{RESET}}$) has occurred since last RSTSRC read.
1	$\overline{\text{PRST}}$	$\overline{\text{RESET}}$	Indicates occurrence of a $\overline{\text{PRST}}$ reset since last RSTSRC read or $\overline{\text{RESET}}$ assertion. Cleared by read of RSTSRC or by $\overline{\text{RESET}}$ active. Writes have no effect. When PRST is held active (low), this bit will always read as 1. $\overline{\text{PRST}}$ = 0: No $\overline{\text{PRST}}$ reset since last RSTSRC read $\overline{\text{PRST}}$ = 1: $\overline{\text{PRST}}$ reset has occurred since last RSTSRC read.
2	WARMRST	$\overline{\text{RESET}}$	A host software reset of DSP or a power management warm reset occurred since last RSTSRC read or last RESET. It is set by a host write of 0 to WARMRST bit of the HDCR or a power management request from D2 or D3. Cleared by read of RSTSRC or $\overline{\text{RESET}}$ assertion. Writes have no effect. WARMRST = 0: No warm reset since last RSTSRC read or $\overline{\text{RESET}}$ WARMRST = 1: Warm reset since last RSTSRC read or $\overline{\text{RESET}}$
3	INTREQ	$\overline{\text{RESET}}$ WARM	Request a DSP-to-PCI interrupt when written with a 1. Causes assertion of $\overline{\text{PINTA}}$ if the INTAM bit in HSR is 1. Writes of 0 have no effect. Always reads as 0.
4	INTRST	$\overline{\text{RESET}}$ WARM	When a 1 is written to this bit, $\overline{\text{PINTA}}$ is deasserted. Writes of 0 have no effect. Always reads as 0.

Table 9–18. DSP Reset Source/Status Register (RSTSRC) Bit Field Description (Continued)

Bits	Name	Reset Source	Description
5	CFGDONE	$\overline{\text{RESET}}$	EEPROM has finished loading PCI configuration registers CFGDONE = 0: Configuration registers have not been loaded CFGDONE = 1: Configuration registers load from EEPROM is complete

6	CFGERR	$\overline{\text{PRST}}$	An error occurred when trying to load the configuration registers from EEPROM. (Checksum failure)
			CFGERR = 0: No configuration error
			CFGERR = 1: Checksum error during EE autoinitialization

9.10.4 PCI Interrupts

The PCI port can generate interrupts to both the CPU and the PCI host (via the $\overline{\text{PINTA}}$ pin). The following sections describe these two types of interrupts.

9.10.4.1 Host Interrupt to the DSP

The PCI host generates an interrupt to the DSP by writing the DSPINT bit in the PCI I/O Host-to-DSP control register. Writing this bit will cause the HOSTSW interrupt if it is enabled in the PCI interrupt enable register (PCIIEN).

9.10.4.2 DSP to Host Interrupt

The DSP can generate an interrupt to the PCI host via the $\overline{\text{PINTA}}$ pin. Interrupts to the host are generated only under DSP software control.

The interrupt is generated by writing a 1 to the INTREQ bit of the DSP reset source/status register (RSTSRC). This will cause the $\overline{\text{PINTA}}$ pin to be asserted on the local PCI bus, if the INTAM bit of the HSR is 0. The $\overline{\text{PINTA}}$ pin is negated by writing a 1 to the INTRST bit in the RSTSRC.

9.11 Reset

9.11.1 PCI Reset of DSP

The PCI host can reset the DSP via the Host-to-DSP control register (HDCR). Setting the WARMRESET bit to 1 causes a DSP reset, which resets all of the internal CPU and peripheral logic.

9.11.2 FIFO Resets

The PCI FIFOs and control logic are held in reset when either the DSP is reset or the PCI pin $\overline{\text{PRST}}$ is asserted.

9.11.3 PCI Configuration Register Reset

The PCI configuration registers read from the EEPROM are initialized on the PCI bus ($\overline{\text{PRST}}$) reset. Neither the DSP core reset, nor the power on reset, affects them.

9.12 Boot Configuration for PCI Port

The following PCI port configurations, along with other device configurations, are determined at device reset via the boot configuration pins:

- **EEPROM autoinitialization (EEAI)**
  Determines if PCI uses default values or read configure values from EEPROM
- **EEPROM size selection (EESZ[2:0])**
  Determines EEPROM size

For details on device and PCI boot configurations, refer to *Chapter 11, Boot Modes and Configuration,* and the device datasheet. The EEPROM interface is discussed in section 9.13.

9.12.1 PCI Boot

The PCI port supports boot from the PCI bus. The CPU is held in reset while the remainder of the device awakes from reset. During this period, the PCI host can initialize DSP's memory as necessary through the PCI. Once the PCI host is finished with all necessary initialization, it writes a 1 to the DSPINT bit of the host-to-DSP control register (HDCR) to remove the DSP core from it's reset state. The DSP then begins execution from address 0.

The sequence of events for PCI boot is as follows:

1) PCI BOOTMODE is selected via configuration pins at reset. Refer to *Chapter 11, Boot Modes and Configuration,* and the device datasheet for details.
2) The PCI interface autoinitializes the PCI configuration registers via EEPROM (if selected)
3) PCI Host sets memory and I/O enable
4) The PCI master writes the DSP page register (DSPP).
5) The PCI master transfers the data to DSP memory-mapped space, starting at address 0h.
6) The PCI master can also access data memory, peripheral registers, EMIF.
7) The PCI master writes the DSPINT bit of the HDCR with a 1 to release the DSP from reset.
8) DSP begins executing code from the program memory-mapped at 0h.

9.13 EEPROM Interface

The DSP supports the 4-wire serial EEPROM interface. The I^2C and SPI interfaces are not supported. The interface consists of the pins shown in Table 9–19.

Table 9–19. EEPROM Serial Interface

Pin	I/O/Z	Description
XSP_CLK	O	Serial EEPROM Clock
XSP_CS	O	Serial EEPROM Chip Select
XSP_DI†	I	Serial EEPROM Data In
XSP_DO	O	Serial EEPROM Data out

† The XSP_DI pin should be pulled down.

The serial EEPROM clock is derived from the DSP peripheral clock. Normally, it will be divided down by 2048 to drive the XSP_CLK pin. The XSP_CLK pin will only be active during EEPROM access to minimize power.

For C62x/C67x, the state of the boot configuration pins EESZ[2:0] at power-on reset determines if a serial EEPROM is present, and if so, what size. See *Chapter 11, Boot Modes and Configuration,* for details on EESZ[2:0]. Table 9–20 summarizes the EEPROM sizes supported by the C62x/C67x. The C64x only supports 4K EEPROM, and EESZ[2:0] does not exist.

Table 9–20. TMS320C62x/C67x EEPROM Size Support

EESZ[2:0]	EEPROM Size Supported on C62x/C67x
000	No EEPROM present
001	1K
010	2K
011	4K
100	16K
All others	Reserved

The EEPROM interface will access the EEPROM as a 16-bit device only. The ORG pin of the EEPROM must be connected to V_{CC}.

9.13.1 PCI Autoinitialization from EEPROM

The DSP allows some of the PCI configuration registers to be loaded from an external serial EEPROM. The PCI port without DSP intervention performs the autoinitialization process.

The state of the boot configuration pins EEAI and EESZ[2:0] (C62x/C67x only) at device reset determine if autoinitialization is enabled. Table 9–21 shows how the EEAI pin selects autoinitialization at reset.

Table 9–21. EEPROM Autoinitialization (EEAI)

EEAI	Autoinitialization
0	Use default values
1	Read values from EEPROM

Autoinitialization is enabled if:

1) Configuration pin EEAI = 1: Autoinitialization enabled.
2) (C62x/C67x only) Configuration pins EESZ[2:0] ≠ 00b. Indicates an EEPROM is present.
3) PCI operation is selected.

If any of these conditions are not met, default values are used for the PCI configuration registers and the EEPROM is not accessed. When all of these conditions are met, the contents from the EEPROM are loaded into some of the PCI configuration registers by the PCI interface (section 9.3.1). The size of the serial EEPROM is required to determine the serial protocol. See *Chapter 11, Boot Modes and Configuration*, for more details.

9.13.2 EEPROM Memory Map

The DSP requires a specific format for the data stored in the serial EEPROM. The first 28 bytes of the EEPROM are reserved for autoinitialization of PCI configuration registers. The remaining locations are not used for autoinitialization and can be used for storing other data. The EEPROM is always accessed as a 16-bit device. Table 9–22 summarizes the EEPROM memory map for the first 28 bytes. See also section 9.3.1 for details.

Table 9–22. EEPROM Memory Map

Address	Contents (msb … lsb)
0h	Vendor ID
1h	Device ID
2h	Class Code [7:0]/Revision ID
3h	Class Code [23:8]
4h	Subsystem Vendor ID
5h	Subsystem ID
6h	Max_Latency/Min_Grant
7h	PC_D1/PC_D0 (power consumed D1, D0)
8h	PC_D3/PC_D2 (power consumed D3, D2)
9h	PD_D1/PD_D0 (power dissipated D1, D0)
Ah	PD_D3/PD_D2 (power dissipated D3, D2)
Bh	Data_scale (PD_D3….PC_D0)
Ch	0000 0000 PMC[14:9] , PMC[5], PMC[3]
Dh	Checksum

9.13.3 EEPROM Checksum

The configuration data contained in the EEPROM is checked against a checksum. The checksum is a 16-bit cumulative XOR of the configuration data words contained in the EEPROM starting with an initial value of AAAAh. The user must ensure that the proper 16-bit checksum value is written to address 0Dh when programming the EEPROM.

Checksum = AAAA XOR Data(00h) XOR Data(01h)….. XOR Data(ODh)

If the checksum fails, the CFGERR bit in the PCIIS and in the HSR registers are set, and optionally, an interrupt to the DSP is generated. The DSP may or may not catch the interrupt, depending on the state of the core at the time. If the PCI is booting the device, the core is held in reset and will miss the interrupt.

The EEREAD bit in the HSR is set if EEPROM autoinitialization is used at power-on reset.

If the serial EEPROM is not accessed for PCI configuration purposes (i.e., EEAI = 0, EESZ[2:0] = 00b at reset), then the checksum is not performed.

Failed checksums will result in the PCI configuration registers being initialized with default data. Refer to the specific PCI configuration registers to determine their default values (section 9.16).

After successful PCI configuration register initialization (auto or default), the DSP reset source/status register (RSTSRC) configuration hold bit CFGDONE is updated to allow DSP to respond to reads, rather than terminating the cycle with disconnect retry. The CFGERR bit in the RSTSRC will be set if a checksum error occurred and default values for the PCI configuration registers are being used.

9.13.4 DSP EEPROM Interface

The EEPROM can also be used by the DSP via three memory-mapped registers: EEPROM Address Register (EEADD), EEPROM Data Register (EEDAT), and EEPROM Control Register (EECTL). These registers are shown in Figure 9–19, Figure 9–20, and Figure 9–21, respectively. They are summarized in Table 9–23, Table 9–24, and Table 9–25.

Figure 9–19. EEPROM Address Register (EEADD)

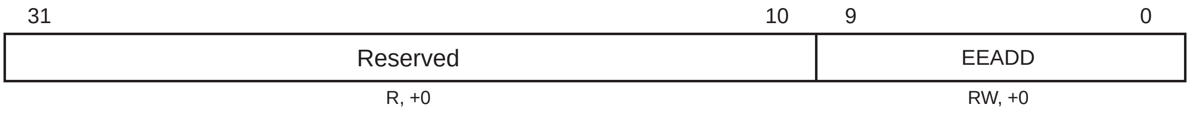


Table 9–23. EEPROM Address Register (EEADD) Bit Field Description

Bits	Name	Reset Source	Description
9:0	EEADD	$\overline{\text{RESET}}$ WARM	EEPROM address

Figure 9–20. EEPROM Data Register (EEDAT)

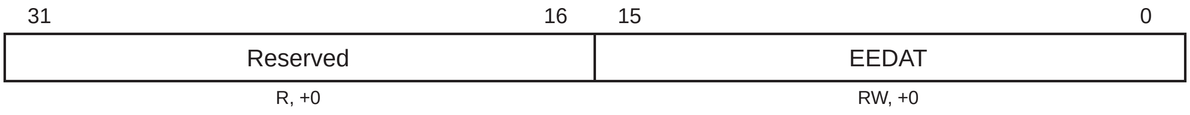


Table 9–24. EEPROM Data Register (EEDAT) Bit Field Description

Bits	Name	Reset Source	Description
15:0	EEDAT	$\overline{\text{RESET}}$ WARM	EEPROM data

Figure 9–21. EEPROM Control Register (EECTL)

31 … 9	8	7	6	5 … 3	2	1 … 0
Reserved	CFGDONE	CFGERR	EEAI	EESZ	READY	EECNT
R, +0	R, +0	R, +0	R, +x	R, +x	R, +0	RW, +00

Table 9–25. EEPROM Control Register (EECTL) Bit Field Description

Bits	Name	Reset Source	Description
1:0	EECNT	$\overline{\text{RESET}}$ WARM	EEPROM op code. Writes to this field cause the serial operation to commence
2	READY	$\overline{\text{RESET}}$ WARM	Indicates that the EEPROM is ready for a new command. Cleared on writes to EECNT READY = 0: EEPROM not ready for new command READY = 1: EEPROM ready for new command

5:3	EESZ	$\overline{\text{RESET}}$	Indicates the state of the EESZ[2:0] pins at power-on reset
			EESZ = 000b: No EEPROM
			EESZ = 001b: 1K (C621x/C671x only)
			EESZ = 010b: 2K (C621x/C671x only)
			EESZ = 011b: 4K
			EESZ = 100b: 16K (C621x/C671x only)

6	EEAI	$\overline{\text{RESET}}$	Indicates state of EEAI pin at power-on reset. EEAI = 0: PCI uses default values EEAI = 1: Read PCI configuration register values from EEPROM

7	CFGERR	$\overline{\text{RESET}}$	Checksum failed error
			CFGERR = 0: No checksum error.
			CFGERR = 1: Checksum error.

8	CFGDONE	$\overline{\text{RESET}}$	Configuration Done
			CFGDONE = 0: Configuration not done
			CFGDONE = 1: Configuration done

The DSP EEPROM interface is available immediately after reset. The CFGDONE bit in the RSTSRC indicates when the EEPROM has been read for PCI autoinitialization.

Serial EEPROM device operation is controlled by seven instructions. The instruction opcode consists of 2 bits. Valid opcodes are presented in Table 9–26.

Table 9–26. EEPROM Command Summary

Op Code	Instruction	Description
10	READ	Reads data at specified address
00 (Address = 11XXXX)	EWEN	Write enable
11	ERASE	Erase memory at address
01	WRITE	Write memory at address
00 (Address = 10XXXX)	ERAL	Erases all memory locations
00 (Address = 01XXXX)	WRAL	Writes all memory locations
00 (Address = 00XXXX)	EWDS	Disables programming instructions

The EEPROM control register EECTL has fields for the 2-bit opcode, as well as read-only bits that indicate the size of the EEPROM (EESZ latched from the EESZ[2:0] bits on power-on reset). The READY bit in the EECTL indicates when the last operation is complete, and the EEPROM is ready for a new instruction. The READY bit is cleared when a new op code is written to the EECNT field. An interrupt can also be generated on EEPROM command completion. The EERDY bit in the PCIIS and PPIIEN registers control the operation of the interrupt.

The EEDAT register is used to clock out user data to the EEPROM on writes and store EEPROM data on reads. For EEPROM writes, data written to the EEDAT is immediately transferred to an internal register. A DSP read from the EEDAT at this point will *not* return the value of the EEPROM data just written. The write data (stored in the internal register) is shifted out on the pins as soon as the two-bit op code is written to the EECNT field of the EECTL. For EEPROM reads, data is available in the EEDAT as soon as READY = 1 in the EECTL.

The EEPROM protocol is as follows:

1) Wait for the CFGDONE bit in RSTSRC to be set. The READY bit in the EECTL, and the EERDY bit in the PCIIS will be set as well.
2) Write EEPROM address to EEADD (address register, the EESZ determines which bits are significant)
3) For EEPROM reads, skip this step. For EEPROM writes (instruction WRITE/ WRAL), write data to EEDAT. This data is immediately transferred to an internal register. Therefore a DSP read from the EEDAT will return invalid data.
4) Write the two-bit op code to the EECNT field of the EECTL.
5) The EEPROM interface then clocks out the EEPROM serial sequence
6) Poll for READY = 1 in the EECTL register, or wait for interrupt (EERDY = 1 in PCIIS).
7) For EEPROM writes, skip this step. For EEPROM reads (instruction READ), read data from EEDAT.

The EEPROM serial sequence is initiated on writes to the EECNT op code field. If the EECNT field is written before the current command is complete (READY = 1), the command will be executed after the current command completes. However, the EEDAT field and the READY field of the EECTL from the previous command will be corrupted. Users should always poll for READY to be asserted before issuing new commands to the EEPROM controller.

9.14 Error Handling

The PCI configuration registers allow the DSP to handle error conditions. The following sections describe the handling of different error conditions. Refer to section 9.16 for a detailed discussions of the register fields.

9.14.1 PCI Parity Error Handling

If the DSP is mastering the bus, the data parity reported bit (bit 15) in the PCI status register (one of the PCI configuration registers) is set under either of the following conditions:

- Parity error during the data phase of a read transaction
- $\overline{\text{PPERR}}$ has been asserted by the target during the data phase of a write transaction.

The data parity detected bit (bit 8) in the PCI status register will be set under any of the following conditions:

- DSP is PCI bus master, and it detects a data parity error during a read transaction.
- DSP is PCI bus target, and it detects a data parity error during a write transaction.
- An address parity error is detected.

The PCI port will assert $\overline{\text{PPERR}}$ if the parity error reporting enable bit (bit 6) in the PCI command register (one of the PCI configuration registers) is set and the data parity detected bit (bit 8 of PCI status register) is set. The assertion of $\overline{\text{PPERR}}$ will remain valid until the second clock after the cycle in which the error occurred.

If a parity error is detected during a transfer involving DSP, the transaction will be allowed to complete unless the PCI port is the master and a target disconnect is detected. The DSP will not master abort due to a parity error.

The PCI bus interface provides parity generation and verification for PCI bus data, and PCI address parity. The PCI port asserts $\overline{\text{PSERR}}$ or $\overline{\text{PPERR}}$ for one PCI clock period and sets a flag in the PCI Command Register if it identifies a parity error.

9.14.2 PCI System Error Handling

An internal system error occurs if any of the following conditions are true:

- An address parity error is detected on the PCI bus (even if DSP is not the target of the transaction) and the parity error reporting enable bit (bit 6) is set in the PCI command register (one of the PCI configuration registers).
- DSP detected $\overline{\text{PPERR}}$ asserted, while mastering the bus.
- DSP received a target abort (disconnect without retry) while mastering the bus.

DSP asserts $\overline{\text{PSERR}}$ if the system error reporting enable bit (bit 8) in the PCI command register is set, and the internal system error flag is set.

DSP will halt and wait for software or hardware reset after $\overline{\text{PSERR}}$ has been asserted.

DSP will set the signaled system error bit (bit 14) in the PCI status register whenever $\overline{\text{PSERR}}$ is asserted.

9.14.3 PCI Master Abort Protocol

In the event that a master abort occurs while DSP is the master on the PCI bus, the current transfer is terminated on both the PCI bus and the Auxiliary DMA or EDMA interface. The received master abort signal will be set in the PCI status register. The PCIMASTER bit in PCIIS register will be set, and optionally an interrupt generated.

A received master abort will reset the START bits in the PCIMC register to 00b. Any master transactions in progress will stop.

9.14.4 PCI Target Abort Protocol

In the event that a target abort occurs while DSP is the master on the PCI bus, the current transfer is terminated on the PCI bus and Auxiliary DMA or EDMA interface. The received target abort signal will be set in the status register. The PCITARGET bit in the PCIIS register will be set, and optionally an interrupt generated.

The target abort follows the same procedure as disabling a master transaction. The received target abort signal is used to reset the START bits in the PCIMC to 00b. Further writes to DSP’s memory will be prevented. The interrupt will indicate to the user that the transfer did not succeed.

9.15 Power Management (TMS320C62x/C67x only)

9.15.1 Power Management for PCI

PCI Power Management Specification revision 1.1 defines power management states $D0_{\text{unitialized}}$, $D0_{\text{active}}$, D1, D2, $D3_{\text{hot}}$, and $D3_{\text{cold}}$. These power management states are discussed below.

$D0_{\text{unitialized}}$: Entered upon power up of the chip or any assertion of $\overline{\text{PRST}}$. The PCI configuration registers have not been initialized from EEPROM. When the DSP is in this state and autoinitialization is enabled, a PCI configuration register read or write will generate a retry. If autoinitialization is not enabled, default values are loaded into the registers and PCI accesses can proceed normally. Once the configuration registers are loaded from EEPROM, the host can initialize the base register and I/O address register. This state is exited and enters $D0_{\text{active}}$ after the configuration registers have been initialized from EEPROM or default values, and the PCI I/O access enable bit (bit 0) and/or memory access enable bit (bit 1) in the PCI command register (one of the PCI configuration registers) are set.

$D0_{\text{active}}$: This is the normal operating state. In this state, the device supports full operation, and all peripherals are available. Transitions from $D0_{\text{active}}$ are accomplished by a power management request, $\overline{\text{PRST}}$ assertion, or removal of V_{DDcore}. If the transition from $D0_{\text{active}}$ is a power management request, the PCI can generate an interrupt to the DSP via the PWRMGMT bit in the PCIIS.

D1: This is the first power management state. The exact operation of the chip in this mode is determined by DSP software. D1 power consumption is less than D0, but it is the responsibility of the DSP software to reduce chip power. The memory and I/O access enable bits of the PCI configuration command register will be disabled by a hardware mask to prevent memory and I/O cycles. DSP will respond to PCI configuration accesses. Transitions from D1 are accomplished by a power management request, $\overline{\text{PRST}}$ assertion, or removal of V_{DDcore}. When the transition from D1 is a power management request, an interrupt to the DSP can be generated via the PWRMGMT bit in the PCIIS. This will allow wake up from DSP power down PD1 mode only, not PD2 or PD3.

D2: This is the next power management state. The exact operation of the chip in this mode is determined by DSP software. D2 power consumption is less than that of D1, but it is the function of the DSP software to reduce chip power. The memory and I/O access enable bits of the PCI configuration command register will be disabled by a hardware mask to prevent memory and I/O cycles. DSP will respond to PCI configuration accesses. Transitions from D2 are accomplished by a power management request, $\overline{\text{PRST}}$ assertion, or removal of V_{DDcore}. When the transition from D2 is a power management request, an internal DSP warm reset is generated. This will allow wake up from DSP power down PD1, PD2 or PD3 mode.

D3$_{hot}$: This is the next power management state. The exact operation of the chip in this mode is determined by DSP software. D3$_{hot}$ power consumption should be less than that of D2, but it is the responsibility of the DSP software to reduce chip power. The memory and I/O access enable bits of the PCI configuration command register will be disabled by a hardware mask to prevent memory and I/O cycles. DSP will respond to PCI configuration accesses. Transitions from D3$_{hot}$ are accomplished by a power management request, $\overline{\text{PRST}}$ assertion, or removal of V_{DDcore}. When the transition from D3$_{hot}$ is a power management request, an internal DSP warm reset is generated. This will allow wake up from DSP PD1, PD2 or PD3 modes.

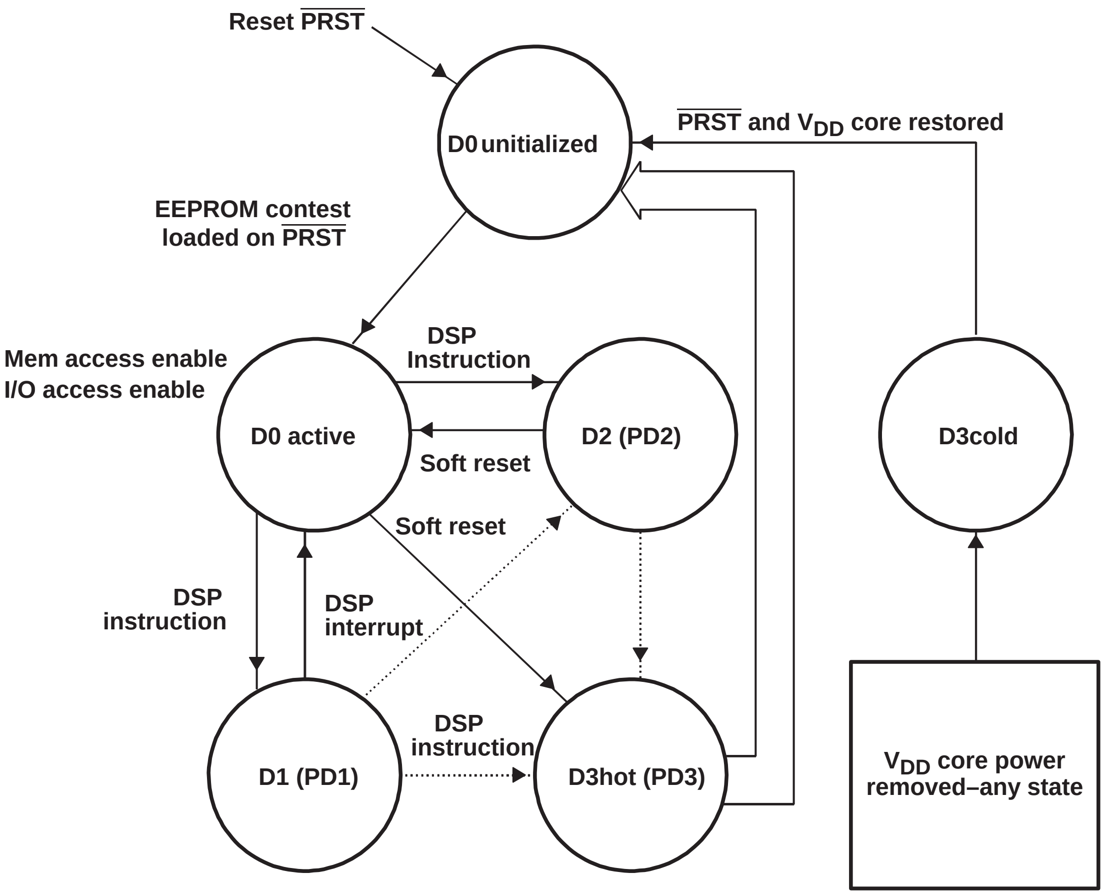
D3$_{cold}$: V_{DDcore} is removed and the device is totally shut down. The most significant power savings is achieved in this mode. A small amount of logic is powered from 3.3 V_{aux} to assert PME upon assertion of $\overline{\text{PWR\_WKP}}$. Transition from D3$_{cold}$ to D0$_{unitialized}$ is accomplished by restoring V_{DDcore} and a $\overline{\text{RESET}}$ or $\overline{\text{PRST}}$.

Figure 9–22 illustrates DSP's power state transitions. The DSP's power management strategy is different from PCI Power Management Specification revision 1.1 in the following ways:

- ❑ DSP core power savings modes can only be changed via software instructions. To transition into a power-savings mode, the DSP core must first be fully operational in order to execute the software. Direct transitions from D1 to D2, from D1 to D3$_{hot}$, or from D2 to D3$_{hot}$ are not supported, as shown by the dotted lines. The core must always transition to D0 $_{active}$ before proceeding to the desired power state.
- ❑ It is important to remember that the DSP software determines the power reduction mechanism for states D1, D2, and D3$_{hot}$.

Transitions from D3$_{hot}$ to D0 will generate an internal DSP reset and an internal PCI reset. The requested state will transition to D0$_{uninitialized}$.

Figure 9–22. PCI Port Power Management State Transition Diagram



Since power management of DSP is performed by DSP software, no power management is supported unless boot is from valid external memory or until the software is downloaded into internal DSP RAM and DSP is released from reset. The DSP software, either internal or external, must be able to handle the host initiated power management requests.

The PCI external host can not issue a PME until the DSP is in the D0 $_{\text{active}}$ state, as is required by the Power Management Specification revision 1.1.

9.15.2 DSP Power Management Strategy

The DSP responds to PCI bus activity to trigger DSP software-controlled transitions between the power management states. The DSP software is responsible for deciding the exact operation of the device in the power savings mode. The states are as follows:

- **D0$_{\textbf{active}}$:** active state, no power savings
- **D1:** DSP PD1 mode where CPU is halted (except for the interrupt logic).
- **D2:** DSP PD2 mode (CPU off, peripherals off).
- **D3$_{\textbf{hot}}$:** DSP PD3 mode (CPU off, peripherals off, PLL disabled).
- **D3$_{\textbf{cold:}}$** similar to D3$_{hot}$, except V$_{DDcore}$ is now removed. A small amount of logic powered by 3.3 V$_{aux}$ can be left powered to generate a PME on assertion of $\overline{\text{PWR\_WKP}}$.

The host requests for power state transitions by writing to the PWRSTATE bit of the PMCSR (one of the PCI configuration registers). If the PWRSTATE bit is different than that of the CURSTATE bit of the PMDCSR (one of the PCI peripheral registers), one of the following events will occur, depending on the current power management state of the DSP:

- **D0 or D1 state:** an interrupt is generated to wake up the core via the PWRMGMT bit in the PCIIS. Upon wake up, the DSP software must read the requested state (REQSTATE in the PMDCSR). The software must then update the bit CURSTATE in the PMDCSR. At this time the software can shut down peripherals, etc. as appropriate, with the execution of the appropriate power savings instruction. Direct transitions from D1 to D2, or from D1 to D3, are not supported because the DSP power savings are under software control.

- **D2 or D3 state:** a warm reset is generated to the DSP core. The warm reset will wake up the core from any power-savings mode that it may be in. The PCI I/O and memory access enable bits in the PCI command register maintain their states during the warm reset. After the core has waked up from the warm reset, it will be in the D0$_{active}$ state. Upon wake up, the DSP software must read the request state (REQSTATE in the PMDCSR). The software must then update the CURSTATE bit in the PMDCSR. At this time, the software can shut down peripherals, etc. as appropriate with the execution of the appropriate power savings instruction. As noted earlier, direct transitions from D2 to D3 are not supported because the DSP's power savings are under software control.

The I/O access enable (bit 0) and memory access enable (bit 1) bits in the PCI command register are disabled by hardware in D1, D2 and D3 to disable memory and I/O cycles, as indicated in the Power Management Specification revision 1.1. If a PME to D0 is requested, the bits will be enabled and take on their original values.

9.15.3 DSP Resets

This section discusses the various types of resets, which is vital to the understanding of DSP's power management strategy. Each type of reset and its effect is discussed below:

Power on reset ($\overline{\textbf{RESET}}$): This is the pin reset applied during power up or hard reset. The state of the autoinitialization pins are sampled on the rising edge of the reset (EESZ[2:0], EEAI). All logic on the DSP is reset including the core and the peripherals. The EEPROM is autoinitialized on $\overline{\text{PRST}}$, not $\overline{\text{RESET}}$.

Warm reset: This reset applies to the core and peripherals. The PCI host can generate the warm reset by setting WARMRESET = 1 in the HDCR register. The warm reset can also be generated on power management requests from D2 or D3. All logic on the DSP is reset including the core and the peripherals. However, the PCI Configuration Registers will maintain their state. In addition, the Memory Access Enable (bit 1) and I/O Access Enable (bit 0) bits of the PCI Command Register (one of the PCI Configuration Registers) are not affected by warm reset.

In summary, warm reset is generated by any one of the following:

- ❑ D2 or D3 power management requests (D2WARMONWKP and D3WARMONWKP bits set in the PMDCSR)
- ❑ Write to the WARMRESET bit of the HDCR by the PCI I/O during state D0
- ❑ $\overline{\text{PWR\_WKP}}$ if D2 or D3 and D2WARMONWKP, D3WARMONWKP set.

PCI reset ($\overline{\textbf{PRST}}$): This reset applies to the PCI Bus Interface Unit (section 9.2, Figure 9–3) and the PCI FIFOs. PCI reset ($\overline{\text{PRST}}$) initializes the PCI Configuration Registers to their default states (if EEAI = 0), or autoinitializes them with values from the EEPROM (if EEAI = 1). The PCI base address registers (part of the PCI configuration registers), the memory access enable (bit 1) and I/O access enable (bit 0) bits of the PCI command register are also reset by $\overline{\text{PRST}}$.

$\overline{\text{PRST}}$ is generated from the $\overline{\text{PRST}}$ pin, or from power management requests of D3 to D0.

The reset to the DSP core is the logical AND of the power-on reset ($\overline{\text{RESET}}$) and the warm reset, both of which are active low.

9.15.4 DSP Support for Power Management

The power management DSP control/status register (PMDCSR) is one of the PCI memory-mapped peripheral registers that allows power management control. The PMDCSR is shown in Figure 9–23 and summarized in Table 9–27.

Figure 9–23. Power Management DSP Control/Status Register (PMDCSR)

31 19	18 11	10	9	8	7	6	5	4	3 2	1 0
Rsvd	HWPMECTL	D3WARM ONWKP	D2WARM ONWKP	PMEEN	$\overline{\text{PWRWKP}}$	PMESTAT	PMEDRVN	AUXDE-TECT	CURSTATE	REQSTATE
R, +0	RW, +88h	R, +x	R, +x	RW, +x	R, +x	R,+x; W, +0	R, +0	R,+x	RW,+00	R, +00

Table 9–27. Power Management DSP Control/Status Register (PMDCSR) Bit Field Description

Bits	Name	Reset Source	Description
1:0	REQSTATE	$\overline{\text{RESET}}$ $\overline{\text{PRST}}$	Last requested power state. Last value written by host to PCI PWRSTATE bits in the PMCSR. Cleared to 00b on $\overline{\text{RESET}}$ or $\overline{\text{PRST}}$.
3:2	CURSTATE	$\overline{\text{RESET}}$ (unaffected by $\overline{\text{PRST}}$ warm reset)	Current power state. Reflects the current power management state of the chip. On changing state, the chip must change CURSTATE. The value written here will be used for PCI reads of PWRSTATE bits in the PMCSR.

4	AUXDETECT	$\overline{\text{RESET}}$
		WARM
			3.3V$_{aux}$DET pin value
			Status of PCIs 3.3 V$_{aux}$DET pin.
			AUXDETECT = 0: 3.3 V$_{aux}$DET is low
			AUXDETECT = 1: 3.3 V$_{aux}$DET is high

5	PMEDRVN	$\overline{\text{RESET}}$ WARM	PME driven high. The DSP has driven the PME pin active high. Set whenever the PMEEN and PMESTAT bits in the PMCSR are high. Cleared by DSP read of PMDCSR, but would be set if PMEEN and PMESTAT are both still high.
6	PMESTAT	None $\overline{\text{RESET}}$ WARM	PMESTAT sticky bit value Reads return the current status of the PMESTAT sticky bit in PMCSR. Write a 1 to this bit to force PMESTAT in the PMCSR to 1. If PMESTAT and PMEEN are written with a 1 at the same time the clear to the PMEEN and PMESTAT will occur. Write of 0 have no effect.
7	$\overline{\text{PWRWKP}}$	None	$\overline{\text{PWRWKP}}$ pin value $\overline{\text{PWRWKP}}$ = 0: $\overline{\text{PWR\_WKP}}$ pin is low $\overline{\text{PWRWKP}}$ = 1: $\overline{\text{PWR\_WKP}}$ pin is high

Table 9–27. Power Management DSP Control/Status Register (PMDCSR) Bit Field Description (Continued)

Bits	Name	Reset Source	Description

8	PMEEN	None	PME assertion enable
			Reads return current value of PMEEN bit in PMCSR sticky bit.
			PMEEN = 0: PMEEN bit in PMCSR is 0; PME assertion is disabled.
			PMEEN = 1: PMEEN bit in PMCSR is 1; PME assertion is enabled.

		$\overline{\text{RESET}}$ WARM	Writes of 1 clear both the PMEEN and PMESTAT sticky bits of PMCSR. Writes of 0 have no effect
9	D2WARMONWKP	$\overline{\text{RESET}}$	Warm reset from D2 D2WARMONWKP = 0: no warm reset will be generated on $\overline{\text{PWR\_WKP}}$ asserted (low). Default. D2WARMONWKP = 1: warm reset will be generated on $\overline{\text{PWR\_WKP}}$ asserted if the current state is D2. Warm resets will only be generated from $\overline{\text{PWR\_WKP}}$ if the above conditions are true and: • $\overline{\text{PRST}}$ (PCI Reset) is deasserted • PCLK is active.
10	D3WARMONWKP	$\overline{\text{RESET}}$	Warm reset from D3 D3WARMONWKP = 0: no warm reset will be generated on $\overline{\text{PWR\_WKP}}$ asserted (low). Default. D3WARMONWKP = 1: warm reset will be generated on $\overline{\text{PWR\_WKP}}$ asserted if the current state is D3. Warm resets will only be generated from $\overline{\text{PWR\_WKP}}$ if the above conditions are true and: • $\overline{\text{PRST}}$ (PCI reset) is deasserted • PCLK is active.
18:11	HWPMECTL	$\overline{\text{RESET}}$	Hardware PME control Allows PME to be generated automatically by hardware on active $\overline{\text{PWR\_WKP}}$ if the corresponding bit is set. Bit Set — Assert PME from hardware 0x11 — Current state = 00 0x12 — Current state = 01 0x13 — Current state = 10 0x14 — Current state = 11 0x15 — Requested state = 00 0x16 — Requested state = 01 0x17 — Requested state = 10 0x18 — Requested state = 11

9.15.4.1 PMCSR Sticky Bits (PMESTAT and PMEEN)

The PMESTAT and PMEEN bits in the PMCSR are powered from the 3.3 V_{aux} input. These bits are cleared on power-on transitions of 3.3 V_{aux}. $\overline{\text{PRST}}$, warm reset, and $\overline{\text{RESET}}$ do not affect the state of these bits. Therefore they are called “sticky” bits.

In power managed PCs, the 3.3 V_{aux} is applied during $D3_{cold}$. Sticky bits are only reset during initial power up.

In power non-managed PCs, the 3.3 V_{aux} transitions with device I/O power V_{DD}. Sticky bits are reset every time the system transitions from $D3_{cold}$ to D0. They are also reset on $\overline{\text{PRST}}$ reset.

PMESTAT bit

The PMESTAT bit is set to a 1 by assertion of $\overline{\text{PWR\_WKP}}$ when in $D3_{cold}$ ($\overline{\text{RESET}}$ active) or by a DSP write to the PMESTAT bit of the PMDCSR if not in $D3_{cold}$, regardless of the value of PMEEN.

The PMESTAT sticky bit is cleared by writing a 1 to the PMESTAT bit in the PMCSR (one of the PCI configuration registers). It is also cleared by a DSP write of 1 to the PMEEN bit of the PMDCSR. PMCSR writes when PMESTAT is a 0 have no effect on this bit. If the 3.3 V_{aux}DET pin is low, no support of PME assertion from $D3_{cold}$, then a $\overline{\text{PRST}}$ clears this bit. If the 3.3 V_{aux}DET pin is high, support of PME assertion from $D3_{cold}$, the a $\overline{\text{RESET}}$, $\overline{\text{PRST}}$ or warm reset does not affect this bit.

PMEEN bit

The PMEEN bit is set to 1 only when PCI configuration register (PMCSR) is written with the PMEEN set to 1. The DSP cannot set the PMEEN bit via the PMDCSR.

The PMEEN bit is cleared by a DSP write of 1 to PMEEN bit of the PMDCSR. PMCSR writes with the PMEEN = 0 will also clear the PMEEN bit. The DSP can observe the value of PMEEN by reading the PMEEN bit in the PMDCSR. If the 3.3 V_{aux}DET pin is low, no support of PME assertion from $D3_{cold}$, then a $\overline{\text{PRST}}$ clears this bit. If the 3.3 V_{aux}DET pin is high, support of PME assertion from $D3_{cold}$, then a $\overline{\text{RESET}}$, $\overline{\text{PRST}}$ or warm reset does not affect this bit.

The output of the PMEEN bit is used to prevent assertion of the PME pin when PMEEN is a 0. The PME pin may be asserted only if PMEEN is a 1.

The PMEDRVN bit in the PMDCSR indicates to the DSP that the PME pin was driven active. This bit is cleared by DSP reads of PMDCSR only, but would be immediately set again if the PMEEN and PMESTAT in the PMCSR are still both set.

9.15.4.2 3.3 V_{aux} Presence Detect Status Bit (AUXDETECT)

The 3.3 V_{aux}DET pin is used to indicate the presence of 3.3 V_{aux} when V_{DDcore} is removed. The DSP can observe this pin by reading the AUXDETECT bit in the PMDCSR. The PMEEN bit in the PMCSR is held clear by the 3.3 V_{aux}DET pin being low.

9.15.4.3 PCI Port Response to $\overline{\text{PWR\_WKP}}$ and PME Generation

The PCI port responses differently to an active $\overline{\text{PWR\_WKP}}$ input, depending on whether V_{DDcore} is alive when 3.3 V_{aux} is alive. The PCI port response to $\overline{\text{PWR\_WKP}}$ is powered by 3.3 V_{aux}.

When V_{DDcore} is alive and 3.3 V_{aux} is alive (i.e., all device power states but $D3_{cold}$), bits are set in the PCIIS register for the detection of the $\overline{\text{PWR\_WKP}}$ high-to-low and low-to-high transition. The $\overline{\text{PWR\_WKP}}$ signal is directly connected to DSP's PCI_WAKEUP interrupt. See section 9.10.

When V_{DDcore} is shut down and 3.3 V_{aux} is alive (i.e. in $D3_{cold}$), $\overline{\text{PWR\_WKP}}$ transition causes the PMESTAT bit in the PMCSR to be set (regardless of PMEEN). If PMEEN is set, $\overline{\text{PWR\_WKP}}$ activity also causes the PME pin to be asserted and held active.

The PCI port can also generate PME depending on the HWPMECTL bits in the PMDCSR. The PME can be generated from any state or on transition to any state on active $\overline{\text{PWR\_WKP}}$ signal if the corresponding bit in the HWPMECTL is set.

Transitions on the $\overline{\text{PWR\_WKP}}$ pin can cause a CPU interrupt (PCI_WAKEUP, see section 9.10). The PWRHL and PWRLH bits of the PCIIS indicate a high-to-low or low-to-high transition on the $\overline{\text{PWR\_WKP}}$ pin. If the corresponding interrupts are enabled in the PCIIEN, a PCI_WAKEUP interrupt is generated to the CPU.

If 3.3 V_{aux} is not powered, the PME pin is in high-impedance. Once PME is driven active by the DSP, it is only deasserted when the PMESTAT bit of the PMCSR is written with a 1 or the PMEEN bit is written with a 0. Neither $\overline{\text{PRST}}$, $\overline{\text{RESET}}$, or warm reset active can cause PME to go into high-impedance if it was already asserted before the reset.

9.15.4.4 DSP Interrupt Indicating that PWRSTATE has Changed

If a PCI PMCSR write causes a change in the PWRSTATE field, a CPU interrupt is generated. The PWRMGMT bit in the PCIIS is set when the PWRSTATE is different than the current state. The CPU interrupt cannot be generated if the DSP clocks are not running. PMCSR changes due to $\overline{\text{RESET}}$ do not cause a CPU interrupt. A CPU interrupt will occur when $\overline{\text{PRST}}$ occurs, since a $\overline{\text{PRST}}$ assertion causes an implicit write of 0 to the PWRSTATE bit of PMCSR.

9.16 PCI Configuration Registers Bit Field Descriptions

This section discusses the PCI configuration registers in detail. These registers are only accessible from the external host PCI. Table 9–28 to summarize the bit fields in the PCI configuration registers. Table 9–51 ists the power management states in the PWRDATA register. See section 9.3.1.

Table 9–28. Vendor ID Register Bit Field Description

Bits	Access	Default	Description
15:0	R	104Ch	Device manufacturer ID

Table 9–29. Device ID Register Bit Field Description

Bits	Access	Default	Description
15:0	R	A106h	Vendor-defined device ID

When the PWRSTATE bit of the PMCSR indicates D1, D2 or D3, the DSP must not respond to PCI activity to its I/O or memory spaces, and must not assert $\overline{\text{PINTA}}$. The DSP hardware will monitor the PWRSTATE bit and mask the I/O access enable bit and the memory access enable bit in the PCI command register, and prevent $\overline{\text{PINTA}}$ assertion when current PWRSTATE is D0, D1, D2 or D3. The PCI command register bit fields are summarized in Table 9–30.

Table 9–30. PCI Command Register Bit Field Description

Bits	Access	Default	Description
0	R/W	0	I/O Access Enable
1	R/W	0	Memory Access Enable
2	R/W	0	Master functionality
3	R	0	No Special Cycle recognition
4	R	0	Memory Write and Invalidate is not supported
5	R	0	Not a VGA device
6	R/W	0	Parity Error Reporting Enable
7	R	0	Data Stepping is not used
8	R/W	0	System Error Reporting Enable
9	R/W	0	Master back-to-back transaction
15:10	R	00h	Reserved

Table 9–31. PCI Status Register Bit Field Description

Bits	Access†	Default	Description
3:0	R	00h	Reserved
4	R	1	Capabilities List implemented (Power Management)
5	R	0	33 MHz maximum frequency
6	R	0	Reserved
7	R	0	Fast Back-to-Back capable
8	R/W–C	0	Master Data Parity Detected Error. Writing 1 resets this bit. Writing 0 has no effect.
10:9	R	01	Device Select signal timing: Medium
11	R/W–C	0	Signaled Target Abort. Writing 1 resets this bit. Writing 0 has no effect.
12	R/W–C	0	Received Target Abort. Writing 1 resets this bit. Writing 0 has no effect.
13	R/W–C	0	Received master Abort. Writing 1 resets this bit. Writing 0 has no effect.
14	R/W–C	0	Signaled System Error. Writing 1 resets this bit. Writing 0 has no effect.
15	R/W–C	0	Data Parity Reported Error. Writing 1 resets this bit. Writing 0 has no effect.

† R/W–C indicates that writing a 1 resets this bit. Writing 0 has no effect.

Table 9–32. Revision ID Register Bit Field Description

Bits	Access	Default	Description
7:0	R	01h	Device-specific revision ID

Table 9–33. Class Code Register Bit Field Description

Bits	Access	Default	Description
7:0	R	00h	Register-level programming interface
15:8	R	00h	Sub-class.
23:16	R	00h	Base Class of device.

Table 9–34. Cache Line Size Register Bit Field Description

Bits	Access	Default	Description
7:0	R/W	00h	Cache Line Size

Table 9–35. Latency Timer Register Bit Field Description

Bits	Access	Default	Description
7:0	R/W	00h	Latency Timer

Table 9–36. Header Type Register Bit Field Description

Bits	Access	Default	Description
6:0	R	00h	Configuration layout for typical PCI master/target device.
7	R	0h	Single function device.

Table 9–37. Base 0 Address Register Bit Field Description

Bits	Access	Default	Description
31:0	R/W	FFC0 0008	Mask for 4 Mbytes, prefetchable memory

Table 9–38. Base 2 Address Register Bit Field Description

Bits	Access	Default	Description
31:0	R/W	FF80 0000	Mask for 8 Mbytes, non-prefetchable memory

Table 9–39. Base 1 Address Register Bit Field Description

Bits	Access	Default	Description
31:0	R/W	FFFF FFF1	Mask for 16 Bytes, I/O space

Table 9–40. Subsystem ID Register Bit Field Description

Bits	Access	Default	Description
15:0	R	00h	Add-in board or subsystem identifier

Table 9–41. Subsystem Vendor ID Register Bit Field Description

Bits	Access	Default	Description
15:0	R	00h	Add-in board or subsystem vendor identifier

Table 9–42. Capabilities Pointer Register Bit Field Description

Bits	Access	Default	Description
7:0	R	040h	Offset to the Power Management Capability Block

Table 9–43. Interrupt Line Register Bit Field Description

Bits	Access	Default	Description
7:0	R/W	000h	Interrupt Line routing information

Table 9–44. Min_Gnt Register Bit Field Description

Bits	Access	Default	Description
7:0	R	00h	Minimum grant

Table 9–45. Max_Lat Register Bit Field Description

Bits	Access	Default	Description
7:0	R	00h	Maximum Latency

Table 9–46. Capability ID Register Bit Field Description

Bits	Access	Default	Description
7:0	R	01h	PCI Power Management ID

Table 9–47. Next Item Pointer Register Bit Field Description

Bits	Access	Default	Description
7:0	R	00h	Next Item in Capabilities List Pointer (‘0’ – last item)

Table 9–48. Power Management Capabilities Register (PMC) Bit Field Description

Bits	Access	Default	Description
15	R	No default value	PME_Support in D3$_{cold}$ state. Read value depends on presence of 3.3V on 3.3V$_{aux}$DET pin. 0: 3.3V$_{aux}$DET pin is low 1: 3.3V$_{aux}$DET pin is high
14:11	R	0h	PME_Support: states that board may assert PME.
10	R	0h	D2_Support
9	R	0h	D1_Support
8:6	R	000	Auxiliary Supply Max Current – Because the PWRDATA register is implemented, this field returns 000.
5	R	0h	Device Specific Initialization is or is not required.
4	R	0	Reserved
3	R	0	PME Clock: PCI clock is not required to assert PME.
2:0	R	010	Version of Power Management

Table 9–49. Power Management Control/Status Register (PMCSR) Bit Field Description

Name	Bits	Access†	Default	Description
PMESTAT	15	R/W–C	User-defined or 0	Power Management Event Status PMESTAT = 0: No Power management event has occurred. PMESTAT = 1: Power management event has occurred. If PMEEN is 1, then the PME pin is also asserted. Writing 1 clears this bit. Writing 0 has no effect. If $3.3V_{aux}$DET is low, this bit resets to 0. If $3.3V_{aux}$DET is high, this bit is 0 on power reset. PMESTAT is a sticky bit (i.e., maintained when main PCI bus power is off) powered from PCI $3.3V_{aux}$ pin.

DATASCALE	14:13	R	00	Data Scale
				Scaling factor for data read from PWRDATA register. This field changes with DATASEL values, and is initialized with PWRDATA initialization data.
				DATASCALE = 00b: Reserved
				DATASCALE = 01b: 0.1 Watt * value from PWRDATA
				DATASCALE = 10b: 0.01 Watt * value from PWRDATA
				DATASCALE = 11b: 0.001 Watt * value from PWRDATA

Table 9–49. Power Management Control/Status Register (PMCSR) Bit Field Description (Continued)

Name	Bits	Access†	Default	Description

DATASEL	12:9	R/W	0h	Data Select
				Select bits for PWRDATA register, DATASCALE field:
				DATASEL = 0000b: PWRDATA reads return D0 power consumed
				DATASEL = 0001b: PWRDATA reads return D1 power consumed
				DATASEL = 0010b: PWRDATA reads return D2 power consumed
				DATASEL = 0011b: PWRDATA reads return $D3_{hot}$ power consumed
				DATASEL = 1000b: PWRDATA reads return D0 power dissipated
				DATASEL = 1001b: PWRDATA reads return D1 power dissipated
				DATASEL = 1010b: PWRDATA reads return D2 power dissipated
				DATASEL = 1011b: PWRDATA reads return $D3_{hot}$ power dissipated
				others: PWRDATA reads return 0

PMEEN	8	R/W	User-defined or 0	PME assertion enable 1 enables assertion of PME pin, 0 disables PME assertion. If $3.3V_{aux}$DET is low, this bit resets to 0. If $3.3V_{aux}$DETis high, this bit is 0 on power up reset. PMEEN is a sticky bit (i.e., maintained when main PCI bus power is off) powered from PCI's $3.3V_{aux}$ pin.

Table 9–49. Power Management Control/Status Register (PMCSR) Bit Field Description (Continued)

Name	Bits	Access[†]	Default	Description
	7:2	R	00h	Reserved

PWRSTATE	1:0	R/W	00b	Power State
				Reads of PWRSTATE return current power state as provided by the CURSTATE bit of the PMDCSR.
				Writes of PWRSTATE are the requested power state by the host. Writes where the bits are not the same as the value currently in the REQSTATE field of the PMDCSR cause a power management interrupt or warm reset to the DSP and update REQSTATE with the value from these bits. Writes where the value is the same as the current value of REQSTATE do not cause the interrupt.
				PWRSTATE = 00b: Power state D0
				PWRSTATE = 01b: Power State D1
				PWRSTATE = 10b: Power State D2
				PWRSTATE = 11b: Power State D3
				Only Power State writes to legal transitions will be processed. All writes will terminate properly on the PCI bus, but illegal transitions will not be processed by the core. If software attempts to write an unsupported, optional state to this field, the write operation must complete normally on the bus; however, the data is discarded and no state change occurs.

† R/W–C indicates that writing a 1 resets this bit. Writing 0 has no effect.

Table 9–50. Power Data Register (PWRDATA) Bit Field Description

Name	Bits	Access	Default	Description
PWRDATA	7:0	R	00h	Power Data Reads return power consumed or dissipated in device power management state selected by PMCSR DATA-SEL bits. The implemented subset is shown in Table 9–51.

Table 9–51. Power Data Register (PWRDATA) DATASCALE Description

DATASEL	PWRDATA, PMCSR DATASCALE Reported
0000	Power consumed in D0 state
0001	Power consumed in D1 state
0010	Power consumed in D2 state
0011	Power consumed in $D3_{hot}$ state
0100	Power dissipated in D0 state
0101	Power dissipated in D1 state
0110	Power dissipated in D2 state
0111	Power dissipated in $D3_{hot}$ state
1XXX	Returns 0

Chapter 10

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**

10.1 Overview

The external memory interfaces (EMIFs) of all TMS320C6000 devices support a glueless interface to a variety of external devices, including:

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

The TMS320C620x/C670x EMIF services requests of the external bus from four requesters:

- On-chip program memory controller that services CPU program fetches
- On-chip data memory controller that services CPU data fetches
- On-chip DMA controller
- External shared-memory device controller (via EMIF arbitration signals)

If multiple requests arrive simultaneously, the EMIF prioritizes them and performs the necessary number of operations. A block diagram of the C620x/C670x is shown in Figure 10–1, and the signals shown there are summarized in Table 10–2. The C620x/C670x EMIF has a 32–bit data bus interface.

The C621x/C671x/C64x services requests of the external bus from two requestors:

- On-chip enhanced direct-memory access (EDMA) controller
- External shared-memory device controller

A block diagram of the C621x/C671x is shown in Figure 10–2. A block diagram of the C64x is shown in Figure 10–3.

The C64x EMIF offers additional flexibility by replacing the SBSRAM mode with a programmable synchronous mode, which supports glueless interfaces to the following:

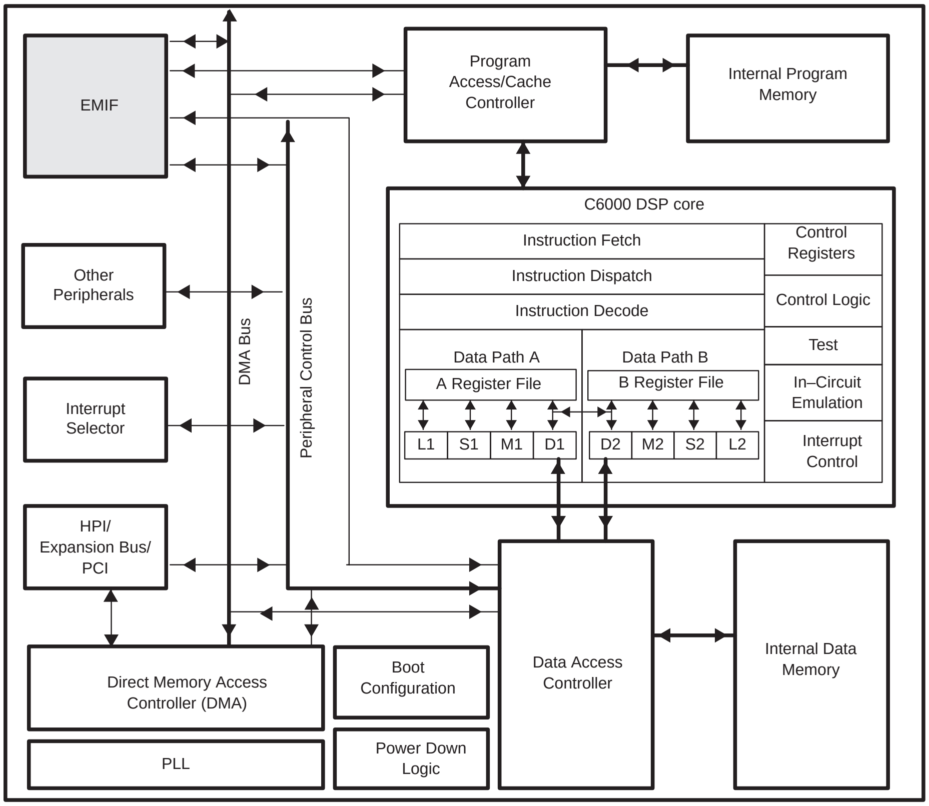
- ZBT (Zero Bus Turnaround) SRAM
- Synchronous FIFOs
- Pipeline and flow-thru SBSRAM

The C64x has two EMIFs, EMIFA and EMIFB, per device. The suffix “A” and “B” indicate the EMIF data bus width as follows:

- EMIFA – 64-bit data bus interface
- EMIFB – 16-bit data bus interface

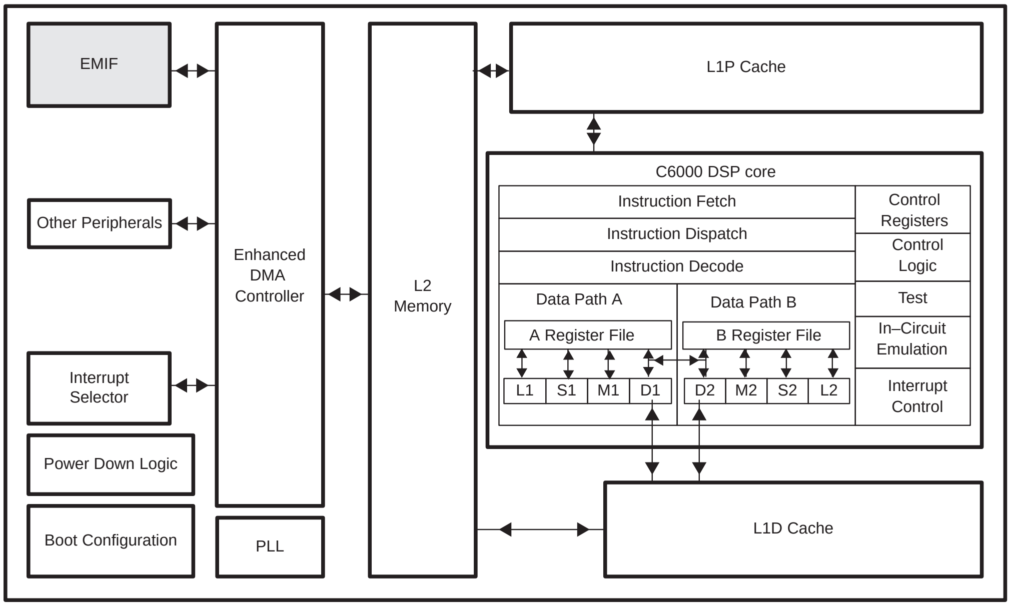
In this chapter, EMIFA and EMIFB are also referred to as EMIF.

Figure 10–1. External Memory Interface in the TMS320C620x/C670x Block Diagram



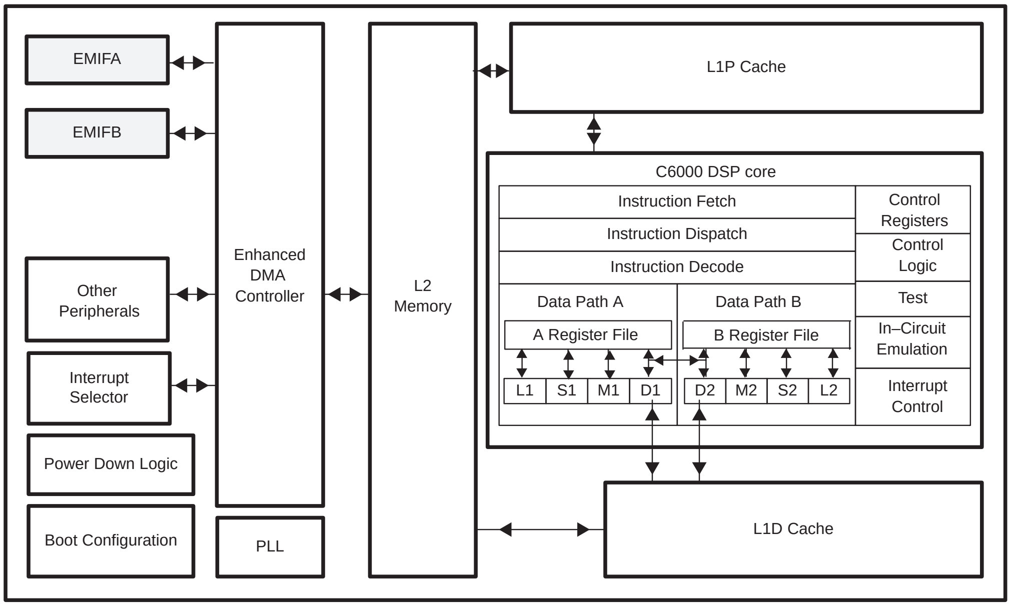
Note: Refer to the specific device datasheet for its peripheral set.

Figure 10–2. External Memory Interface in the TMS320C621x/C671x Block Diagram



Note: Not all peripherals exist on all C621x/C671x devices. Refer to the specific device datasheet for its peripheral set.

Figure 10–3. External Memory Interface in the TMS320C640x Block Diagram



Note: Refer to the specific device datasheet for its peripheral set.

Table 10–1 summarizes the differences between the C6000 EMIF devices.

Table 10–1. TMS320C6000 EMIF Summary

	C62x/C67x			C64x	
Features	**C6201 C6701**	**Other C620x Other C670x**	**C621x C671x**†	**EMIFA**	**EMIFB**
Bus Width	32	32	32†	64	16
Number of Memory Spaces	4	4	4	4	4
Addressable Space (Mbytes)	52	52	512†	1024	256
Synchronous Clocking	CPU clock and/or 1/2x CPU clock	1/2x CPU clock	Independent ECLKIN	Independent ECLKIN, 1/4x CPU clock or 1/6x CPU clock	Independent ECLKIN, 1/4x CPU clock or 1/6x CPU clock
Width Support	32 bit; 8-/16-bit ROM	32 bit; 8-/16-bit ROM	8-/16-/32-bit†	8–, 16–, 32–, or 64–bit	8–/16–bit
Supported memory type at CE1	Asynchronous memory	Asynchronous memory	All types	All types	All types
Control signals	Separate	Muxed synchronous signals	Muxed all control signals	Muxed all control signals	Muxed all control signals
Synchronous memory in system	Both SDRAM and SBSRAM	Either SDRAM or SBSRAM	Both SDRAM and SBSRAM	All synchronous	All synchronous
Additional registers	—	—	SDEXT	SDEXT CExSEC	SDEXT CExSEC
PDT support	No	No	No	Yes	Yes
ROM/Flash	✓	✓	✓	✓	✓
Asynchronous memory I/O	✓	✓	✓	✓	✓
Pipeline SBSRAM	✓	✓	✓	✓	✓
Flow thru SBSRAM				✓	✓
ZBT SRAM				✓	✓
Standard Syn-chronous FIFO				✓	✓
FWFT FIFO				✓	✓

† C6712 EMIF is only 16-bit wide. Addressable space is 256 Mbytes. It supports memory widths of 8 bits and 16 bits.

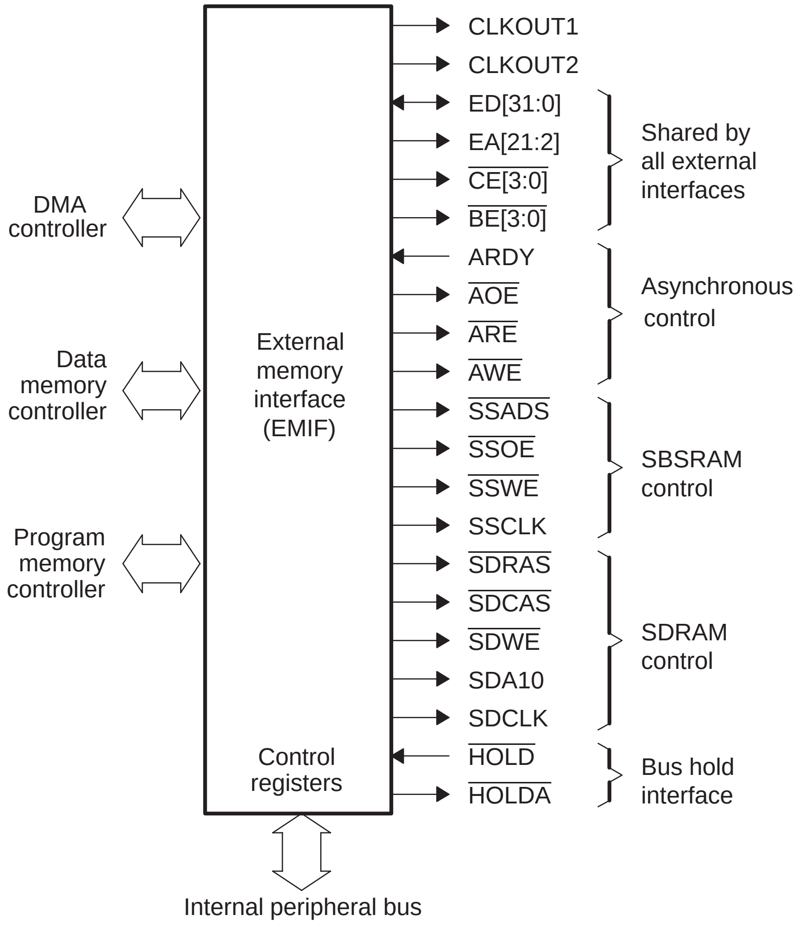
10.1.1 *EMIF Signals*

The following sections describe the EMIF signals on the TMS320C6000 devices.

10.1.1.1 TMS320C6201/C6701 External Memory Interface

The EMIF signals of the C6201/C6701 are shown in Figure 10–4. The C6201/C6701 provides separate clock and control signals for the SBSRAM and SDRAM interface. The SDRAM runs off SDCLK, while the SBSRAM runs off SSCLK. All three memory types (SDRAM, SBSRAM, and asynchronous devices) can be included in a system. Asynchronous interface is supported on all CE spaces, but CE1 is used for asynchronous interface only.

Figure 10–4. TMS320C6201/C6701 External Memory Interface

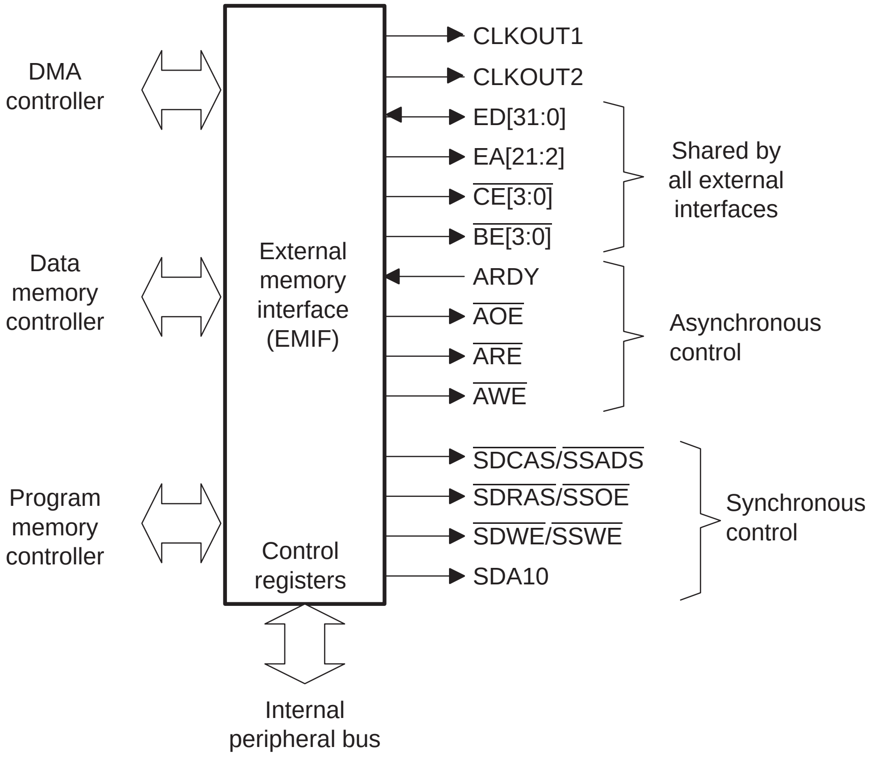


10.1.1.2 TMS320C6202(B)/C6203(B)/C6204/C6205 External Memory Interface

The EMIF signals are shown in Figure 10–5. These C620x/C670x devices have combined the SDRAM and SBSRAM signals. Only one of these two memory types can be used in a system. These memories run off CLKOUT2 (EMIF clock cycle), which is equal to half the CPU clock rate.

Asynchronous interface is supported on all CE spaces, but CE1 is used for asynchronous interface only.

Figure 10–5. TMS320C6202(B)/C6203(B)/C6204/C6205 External Memory Interface


DMA controller
Data memory controller
Program memory controller
External memory interface (EMIF)
Control registers
Internal peripheral bus
CLKOUT1
CLKOUT2
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 control
$\overline{\text{SDCAS}}$/$\overline{\text{SSADS}}$
$\overline{\text{SDRAS}}$/$\overline{\text{SSOE}}$
$\overline{\text{SDWE}}$/$\overline{\text{SSWE}}$
SDA10
Synchronous control

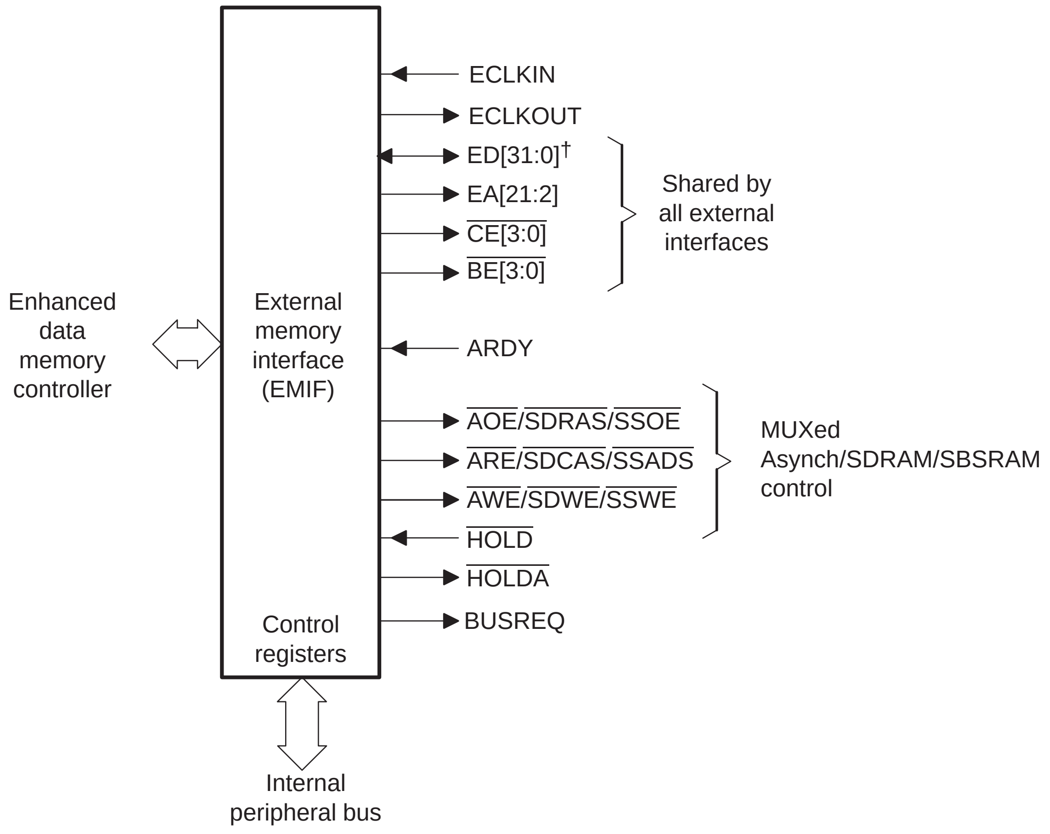
10.1.1.3 TMS320C621x/C671x External Memory Interface

The EMIF signals of the C621x/C671x are shown in Figure 10–6. The C621x/C671x has the following features:

❑ The C621x/C671x 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 C621x/C671x should operate off of ECLKOUT (EMIF clock cycle). If desired, the CLKOUT2 output can be routed back to the ECLKIN input.

- ❑ The SDRAM, SBSRAM, and asynchronous signals are combined. All three memory types can be included in a system, since no background refresh is performed.
- ❑ Unlike the C620x/C670x, the C621x/C671x EMIF space CE1 supports all three types of memory.
- ❑ 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 10–6. TMS320C621x/C671x 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
Enhanced data memory controller
External memory interface (EMIF)
ARDY
$\overline{\text{AOE}}/\overline{\text{SDRAS}}/\overline{\text{SSOE}}$
$\overline{\text{ARE}}/\overline{\text{SDCAS}}/\overline{\text{SSADS}}$
$\overline{\text{AWE}}/\overline{\text{SDWE}}/\overline{\text{SSWE}}$
MUXed Asynch/SDRAM/SBSRAM control
$\overline{\text{HOLD}}$
$\overline{\text{HOLDA}}$
BUSREQ
Control registers
Internal peripheral bus

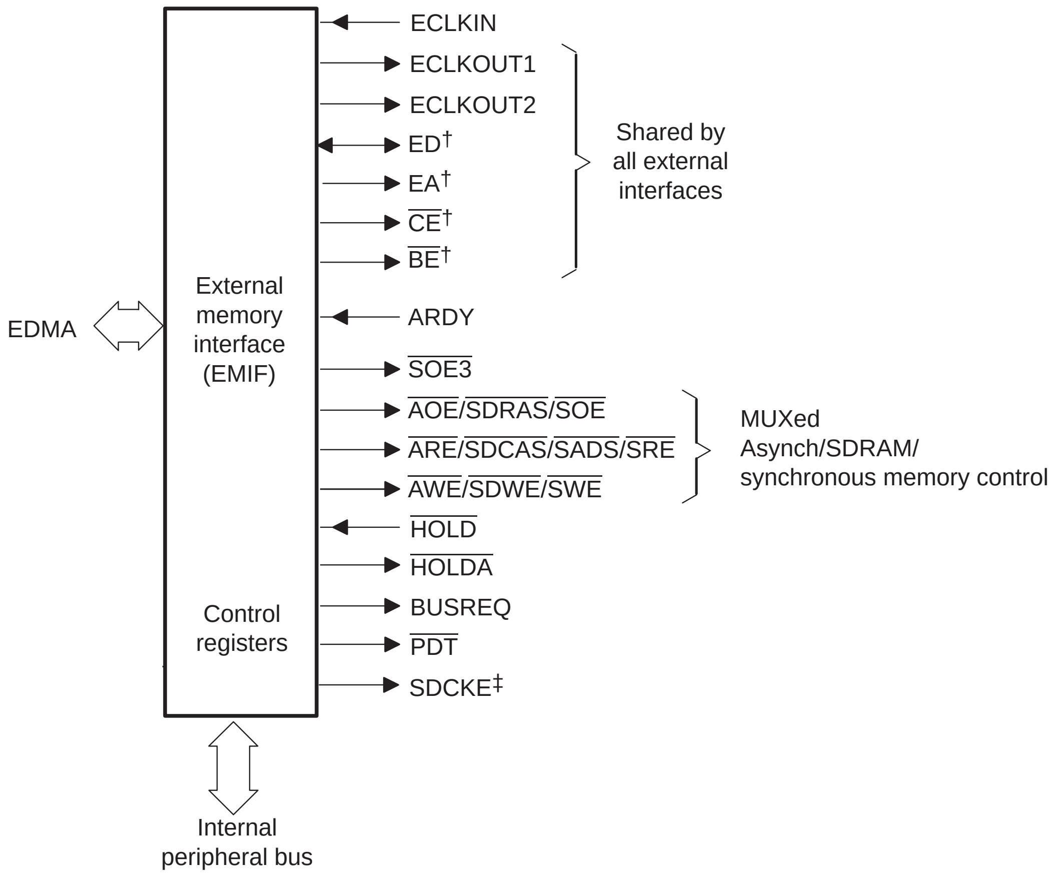
† C6712 uses ED[15:0].

10.1.1.4 TMS320C64x External Memory Interface

The EMIF signals of the C64x are shown in Figure 10–7. These signals apply to both EMIFA and EMIFB with the exception of the SDCKE signal, which applies to EMIFA only. The C64x EMIF is an enhanced version of the C621x EMIF. It includes all the C621x/C671x features plus the following new features:

- The data bus on EMIFA is 64 bit wide. The data bus on EMIFB is 16-bit wide.
- The EMIF clocks ECLKOUTx are generated internally based on the EMIF input clock. At device reset, users can configure one of the following three clocks as the EMIF input clock: internal CPU clock rate divide by 4, internal CPU clock rate divide by 6, or external ECLKIN. See *Chapter 11 Boot Modes and Configuration* for details. All of the memories interfacing with the C64x should operate off of ECLKOUTx (EMIF clock cycle). The ECLKOUT1 frequency equals to EMIF input clock frequency. The ECLKOUT2 frequency is programmable to be EMIF input clock frequency divided by 1, 2, or 4.
- The SBSRAM controller is replaced with a more flexible programmable synchronous memory controller. The SBSRAM control pins are also replaced with synchronous control pins.
- The $\overline{\text{PDT}}$ pin provides external-to-external transfer support.

Figure 10–7. TMS320C64x External Memory Interface (EMIFA and EMIFB)


ECLKIN
ECLKOUT1
ECLKOUT2
ED†
EA†
$\overline{\text{CE}}$†
$\overline{\text{BE}}$†
Shared by all external interfaces
EDMA
External memory interface (EMIF)
ARDY
$\overline{\text{SOE3}}$
$\overline{\text{AOE}}$/$\overline{\text{SDRAS}}$/$\overline{\text{SOE}}$
$\overline{\text{ARE}}$/$\overline{\text{SDCAS}}$/$\overline{\text{SADS}}$/$\overline{\text{SRE}}$
$\overline{\text{AWE}}$/$\overline{\text{SDWE}}$/$\overline{\text{SWE}}$
MUXed Asynch/SDRAM/ synchronous memory control
$\overline{\text{HOLD}}$
$\overline{\text{HOLDA}}$
BUSREQ
$\overline{\text{PDT}}$
SDCKE‡
Control registers
Internal peripheral bus

† See Table 9–2 for ED, EA, $\overline{\text{CE}}$, and $\overline{\text{BE}}$ pins on EMIFA and EMIFB.
‡ SDCKE applies to EMIFA only.

Table 10–2. EMIF Signal Descriptions

Device Group†‡ 1	2	3	4	5	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 Group 2 devices.
			✓	✓	CLKOUT4	O/Z	Clock output. Runs at 1/4 the CPU clock rate. On C64x, the CLKOUT4 pin is MUXed with the GP1 (general purpose input/output 1 pin). By default, it functions as CLKOUT4.
			✓	✓	CLKOUT6	O/Z	Clock output. Runs at 1/6 the CPU clock rate. On C64x, the CLKOUT6 pin is MUXed with the GP2 (general purpose input/output 2 pin). By default, it functions as CLKOUT6.
		✓	✓	✓	ECLKIN	I	EMIF clock input. Must be provided by the system on C621x/C671x. Optionally provided by the system on C64x.
		✓			ECLKOUT	O	EMIF clock output. All EMIF I/O are clocked relative to ECLKOUT.
			✓	✓	ECLKOUT1	O/Z	EMIF output clock at EMIF input clock (ECLKIN, CPU/4 clock, or CPU/6 clock) frequency.
			✓	✓	ECLKOUT2	O/Z	EMIF output clock at EMIF input clock (ECLKIN, CLKOUT4, or CLKOUT6) frequency divide by 1, 2, or 4.
				✓	ED[15:0]	I/O/Z	EMIF 16-bit data bus I/O
✓	✓	✓			ED[31:0]§	I/O/Z	EMIF 32-bit data bus I/O§
			✓		ED[63:0]	I/O/Z	EMIF 64-bit data bus I/O
				✓	EA[20:1]	O/Z	External address output. Drives bits 20–1 of the byte address. (Effectively a half-word address.)
✓	✓	✓			EA[21:2]	O/Z	External address output. Drives bits 21–2 of the byte address. (Effectively a word address.)
			✓		EA[22:3]	O/Z	External address output. Drives bits 22–3 of the byte address. (Effectively a double-word 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

† "M" indicates a multiplexed output signal.
‡ Group1 devices include: C6201/C6701.
Group 2 devices include: all C620x/C670x *except* C6201/C6701.
Group 3 devices include: C621x/C671x.
Group 4 devices include: C64x EMIFA.
Group 5 devices include: C64x EMIFB.
§ C6712 has a 16-bit bus; therefore, ED[31:16] do not apply.

Table 10–2. EMIF Signal Descriptions (Continued)

Device Group[†‡]					Pin	(I/O/Z)	Description
1	**2**	**3**	**4**	**5**			
				✓	$\overline{\text{BE[1:0]}}$	O/Z	Active-low byte enables.On C64x, byte-enables go active for only the appropriate byte lane for both writes and reads.
✓	✓	✓			$\overline{\text{BE[3:0]}}$	O/Z	Active-low byte enables. Individual bytes and halfwords can be selected for write cycles. For read cycles, all four byte-enables are active.
			✓		$\overline{\text{BE[7:0]}}$	O/Z	Active-low byte enables.On C64x, byte-enables go active for only the appropriate byte lane for both writes and reads.
✓	✓	✓	✓	✓	ARDY	I	Ready. Active-high asynchronous ready input used to insert wait states for slow memories and peripherals.
✓	✓	M	M	M	$\overline{\text{AOE}}$	O/Z	Active-low output enable for asynchronous memory interface
✓	✓	M	M	M	$\overline{\text{AWE}}$	O/Z	Active-low write strobe for asynchronous memory interface
✓	✓	M	M	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	Active low 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	M	M	$\overline{\text{SDRAS}}$	O/Z	Active-low row address strobe for SDRAM memory interface
✓	M	M	M	M	$\overline{\text{SDCAS}}$	O/Z	Active-low column address strobe for SDRAM memory interface
✓	M	M	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.
			✓		SDCKE	O/Z	SDRAM clock enable (used for self-refresh mode). If SDRAM is not in the system, SDCKE can be used as a general-purpose output.
✓					SDCLK	O/Z	SDRAM interface clock. Runs at 1/2 the CPU clock rate. Equivalent to CLKOUT2.
			M	M	$\overline{\text{SADS}}$/$\overline{\text{SRE}}$	O/Z	Synchronous memory address strobe or read enable (selected by RENEN in CE space secondary control register).

[†] “M” indicates a multiplexed output signal.

[‡] Group1 devices include: C6201/C6701.
Group 2 devices include: all C620x/C670x *except* C6201/C6701.
Group 3 devices include: C621x/C671x.
Group 4 devices include: C64x EMIFA.
Group 5 devices include: C64x EMIFB.

[§] C6712 has a 16-bit bus; therefore, ED[31:16] do not apply.

Table 10–2. EMIF Signal Descriptions (Continued)

Device Group†‡					Pin	(I/O/Z)	Description
1	**2**	**3**	**4**	**5**			
			M	M	$\overline{\text{SOE}}$	O/Z	Synchronous memory output enable
			✓	✓	$\overline{\text{SOE3}}$	O/Z	Synchronous memory output enable for $\overline{\text{CE3}}$ (intended for glueless FIFO interface).
			M	M	$\overline{\text{SWE}}$	O/Z	Synchronous memory write enable
✓	✓	✓	✓	✓	$\overline{\text{HOLD}}$	I	Active-low external bus hold (3-state) request
✓	✓	✓	✓	✓	$\overline{\text{HOLDA}}$	O	Active-low external bus hold acknowledge
		✓	✓	✓	BUSREQ	O	Active-high bus request signal. Indicates pending refresh or memory access.
			✓	✓	$\overline{\text{PDT}}$	O/Z	Peripheral data transfer data. This signal is active during the data phase of PDT transfers.

† "M" indicates a multiplexed output signal.
‡ Group1 devices include: C6201/C6701.
Group 2 devices include: all C620x/C670x *except* C6201/C6701.
Group 3 devices include: C621x/C671x.
Group 4 devices include: C64x EMIFA.
Group 5 devices include: C64x EMIFB.
§ C6712 has a 16-bit bus; therefore, ED[31:16] do not apply.

10.2 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 10–3.

Table 10–3. EMIF Memory-Mapped Registers

Byte Address		Abbreviation	EMIF Register Name
EMIF/EMIFA	EMIFB‡		
0180 0000h	01A8 0000h	GBLCTL	EMIF global control
0180 0004h	01A8 0004h	CE1CTL	EMIF CE1 space control
0180 0008h	01A8 0008h	CE0CTL	EMIF CE0 space control
0180 000Ch	01A8 000Ch		Reserved
0180 0010h	01A8 0010h	CE2CTL	EMIF CE2 space control
0180 0014h	01A8 0014h	CE3CTL	EMIF CE3 space control
0180 0018h	01A8 0018h	SDCTL	EMIF SDRAM control
0180 001Ch	01A8 001Ch	SDTIM	EMIF SDRAM refresh control
0180 0020h	01A8 0020h	SDEXT§	EMIF SDRAM extension§
0180 0024h to 0180 0040h	01A8 0024h to 01A8 0040h	–	Reserved
0180 0044h	01A8 0044h	CE1SEC¶	EMIF CE1 space secondary control¶
0180 0048h	01A8 0048h	CE0SEC¶	EMIF CE0 space secondary control¶
0180 004Ch	01A8 004Ch	–	Reserved
0180 0050h	01A8 0050h	CE2SEC¶	EMIF CE2 space secondary control¶
0180 0054h	01A8 0054h	CE3SEC¶	EMIF CE3 space secondary control¶

‡ EMIFB exists only on C64x.
§ Register does not exist on C620x/C6701 devices.
¶ Register exists only on C64x.

10.2.1 Global Control Register (GBLCTL)

The EMIF global control register (shown in Figure 10–8 and summarized in Table 10–4) configures parameters common to all the CE spaces.

Figure 10–8. EMIF Global Control Register (GBLCTL)

C6201/C6701:

31–16
Rsv
R, +0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Rsv	Rsv[†]	Rsv[†]	Rsv[†]	Rsv	ARDY	HOLD	HOLDA	NO HOLD	SDCEN	SSCEN	CLK1EN	CLK2EN	SSCRT	RBTR8	MAP
R,+0	RW,+0	RW,+1	RW,+1	R, +0	R, +x	R, +x	R, +x	RW, +0	RW, +1	RW, +1	RW, +1	RW, +1	RW, +0	RW, +0	R, +x

C6202(B)/C6203(B)/C6204/C6205:

31–16
Rsv
R, +0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Rsv	Rsv[†]	Rsv[†]	Rsv[†]	Rsv	ARDY	HOLD	HOLDA	NO HOLD	SDCEN	SSCEN	CLK1EN	Rsv	Rsv	RBTR8	MAP
R,+0	RW,+0	RW,+1	RW,+1	R, +0	R, +x	R, +x	R, +x	RW, +0	RW, +1	RW, +1	RW, +1	RW, +1	R, +0	RW, +0	R, +x

C621x/C671x:

31–16
Rsv
R, +0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Rsv	Rsv[†]	Rsv[†]	Rsv[†]	BUSREQ	ARDY	HOLD	HOLDA	NO HOLD	Rsv	Rsv	CLK1EN	CLK2EN	Rsv	Rsv	Rsv
R,+0	RW,+0	RW,+1	RW,+1	R, +0	R, +x	R, +x	R, +x	RW, +0	R, +1	R, +1	RW, +1	RW, +1	R, +0	R, +0	R, +0

C64x:

31–20	19–18	17	16
Rsv	EK2RATE	EK2HZ	EK2EN
R,+0	RW, +10	RW, +0	RW, +1

15–14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
Rsv	BRMODE	Rsv	BUSREQ	ARDY	HOLD	HOLDA	NO HOLD	EK1HZ	EK1EN	CLK4EN	CLK6EN	Rsv	Rsv	Rsv
R,+0	RW,+1	R,+0	R, +0	R, +x	R, +x	R, +x	RW, +0	RW, +1	RW, +1	RW, +1	RW, +1	RW, +1	R, +0	R, +0

† The reserved bit fields should always be written with their default values when modifying the GBLCTL.

Table 10–4. EMIF Global Control Register (GBLCTL) Field Descriptions

Field	Description	Apply to Device Group[†]
MAP	Map mode, contains the value of the memory map mode of the device MAP = 0: map 0 selected. External memory located at address 0. MAP = 1: map 1 selected. Internal memory located at address 0.	1,2
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.	1,2

SSCRT	SBSRAM clock rate select
	SSCRT = 0: SSCLK runs at 1/2x CPU clock rate
	SSCRT = 1: SSCLK runs at 1x CPU clock rate
		1

CLK1EN	CLKOUT1 enable
	CLK1EN = 0: CLKOUT1 held high
	CLK1EN = 1: CLKOUT1 enabled to clock
		1,2,3

CLK2EN	CLKOUT2 enable
	CLK2EN = 0: CLKOUT2 held high
	CLK2EN = 1: CLKOUT2 enabled to clock
	
	CLKOUT2 is enabled/disabled using SSCEN/SDCEN on the C6202(B)/C6203(B)/C6204/C6205
		1,3

CLK4EN	CLKOUT4 enable
	CLK4EN = 0, CLKOUT4 held high
	CLK4EN = 1, CLKOUT4 enabled to clock
	
	For C64x, CLKOUT4 pin is muxed with GP1 pin. Upon exiting reset, CLKOUT4 is enabled and clocking. After reset, CLKOUT4 maybe be configured as GP1 via the GPIO enable register GPEN.
		4,5

CLK6EN	CLKOUT6 enable
	CLK6EN = 0, CLKOUT6 held high
	CLK6EN = 1, CLKOUT6 enabled to clock
	
	For C64x, CLKOUT6 pin is muxed with GP2 pin. Upon exiting reset, CLKOUT6 is enabled and clocking. After reset, CLKOUT6 maybe be configured as GP2 via the GPIO enable register GPEN.
		4,5

SSCEN	SSCLK enable
	SSCEN = 0: SSCLK held high
	SSCEN = 1: SSCLK enabled to clock
	
	SSCEN enables CLKOUT2 on the C6202(B)/C6203(B)/C6204/C6205 if SBSRAM is used in the system (specified by the MTYPE field in in the CE space control register).
		1,2

[†] Group1 devices include:	C6201/C6701.
Group 2 devices include:	all C620x/C670x *except* C6201/C6701.
Group 3 devices include:	C621x/C671x.
Group 4 devices include:	C64x EMIFA.
Group 5 devices include:	C64x EMIFB.

Field	**Description**	**Apply to Device Group**†

SDCEN	SDCLK enable
	SDCEN = 0: SDCLK held high
	SDCEN = 1: SDCLK enabled to clock
	SDCEN enables CLKOUT2 on C6202(B)/C6203(B)/C6204/C6205 if SDRAM is used in system (specified by the MTYPE field in in the CE space control register).
		1,2

EK1EN	ECLKOUT1 enable
	ECLK1EN = 0, ECLKOUT1 held low
	ECLK1EN = 1, ECLKOUT1 enabled to clock
		4,5

EK2EN	ECLKOUT2 enable
	ECLK2EN = 0, ECLKOUT2 held low
	ECLK2EN = 1, ECLKOUT2 enabled to clock
		4,5

EK1HZ	ECLKOUT1 High-Z control
	ECLK1HZ = 0, ECLKOUT1 continues clocking during Hold
	ECLK1HZ = 1, ECLKOUT1 High-Z during Hold
		4,5

EK2HZ	ECLKOUT2 High-Z control
	ECLK2HZ = 0, ECLKOUT2 continues clocking during Hold
	ECLK2HZ = 1, ECLKOUT2 High-Z during Hold
		4,5

EK2RATE	ECLKOUT2 Rate. ECLKOUT2 runs at:
	ECLK2RT = 00, 1x EMIF input clock (ECLKIN, CPU/4 clock, or CPU/6 clock) rate
	ECLK2RT = 01, 1/2x EMIF input clock (ECLKIN, CPU/4 clock, or CPU/6 clock) rate
	ECLK2RT = 10, 1/4x EMIF input clock (ECLKIN, CPU/4 clock, or CPU/6 clock) rate
		4,5

NOHOLD	External HOLD disable
	NOHOLD = 0: hold enabled
	NOHOLD = 1: hold disabled
		1,2,3,4,5

HOLDA	HOLDA = 0: $\overline{\text{HOLDA}}$ output is low. External device owns EMIF. HOLDA = 1: $\overline{\text{HOLDA}}$ output is high. External device does not own EMIF.	1,2,3,4,5
HOLD	HOLD = 0: $\overline{\text{HOLD}}$ input is low. External device requesting EMIF. HOLD = 1: $\overline{\text{HOLD}}$ input is high. No external request pending.	1,2,3,4,5
ARDY	ARDY = 0: ARDY input is low. External device not ready. ARDY = 1: ARDY input is high. External device ready.	1,2,3,4,5
BUSREQ	BUSREQ = 0; BUSREQ ouput is low. No access/refresh pending. BUSREQ = 1; BUSREQ output is high. Access/refresh pending or in progress. See also BRMODE.	3,4,5
BRMODE	Bus Request Mode BRMODE = 0, BUSREQ indicates memory access pending or in progress BRMODE = 1, BUSREQ indicates memory access or refresh pending/in progress	4,5

† Group1 devices include:	C6201/C6701.
Group 2 devices include:	all C620x/C670x *except* C6201/C6701.
Group 3 devices include:	C621x/C671x.
Group 4 devices include:	C64x EMIFA.
Group 5 devices include:	C64x EMIFB.

In order to support as many common programming practices as possible between the C620x/C670x devices, SSCEN and SDCEN are used in

C6202(B)/C6203(B)/C6204/C6205 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 C621x/C671x contains the BUSREQ field to indicate if the EMIF has access/refresh pending or in progress. For C64x, the BRMODE field determines if BUSREQ shows memory refresh status. If BRMODE = 0, BUSREQ only indicates memory access status. If BRMODE = 1, BUSREQ indicates both memory access and refresh status.

The C64x has bit fields (EK1EN, EK2EN, EK1HZ, EK2HZ, and EK2RATE) to control ECLKOUTx outputs, as shown in Table 10–4. See also section 10.14.

10.2.2 EMIF CE Space Control Registers

The CE space control registers (CExCTL) are shown in Figure 10–9 and summarized in Table 10–5. These registers correspond to the CE memory spaces supported by the EMIF. For all EMIF, there are four CE space control registers corresponding to the four external CE signals.

The MTYPE field identifies the memory type for the corresponding CE space. If the MTYPE field selects a synchronous memory type (SDRAM, SBSRAM for C62x/C67x, or programmable synchronous for C64x), the remaining fields in the register have no 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 10.8.

The MTYPE field in the CExCTL 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(B)/C6203(B)/C6204/C6205, only one synchronous memory type is supported in a system. This is because these devices support SDRAM background refresh and they have shared synchronous memory control signals. SBSRAM accesses could be corrupted during SDRAM refresh if both memory types are present. Therefore, software must ensure that when a CE space is set up as a synchronous memory type, no other CE spaces are set up as a different synchronous memory type. For example, if a CE space is set as SBSRAM, then no other CE spaces should be set as SDRAM. Similarly, if a CE space is set as SDRAM, then no other CE spaces should be set as SBSRAM.

The programmed values in the CExCTL refer to CLKOUT1 for the C620x/C670x, ECLKOUT for the C621x/C671x, and ECLKOUT1 for the C64x.

Figure 10–9. TMS320C62x/C67x/C64x EMIF CE Space Control Register (CExCTL)

C620x/C670x:

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

C621x/C671x/C64x:

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 4	3	2 0
TA	Read strobe	MTYPE	Write hold MSB‡	Read hold
R, +11	RW, +111111	RW, +0010†	RW, +0	RW, +011

† MTYPE default value is RW, +0000.
‡ For C621x/C671x, this field is reserved. R,+0.

Table 10–5. EMIF CE Space Control Registers (CExCTL) 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 C620x/C670x
	
	<u>MTYPE Definitions for C620x/C670x</u>
	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
	
	<u>MTYPE Definitions for C621x/C671x/C64x‡</u>
	MTYPE = 0000b: 8-bit-wide asynchronous interface
	MTYPE = 0001b: 16-bit-wide asynchronous interface
	MTYPE = 0010b: 32-bit-wide asynchronous interface
	MTYPE = 0011b: 32-bit-wide SDRAM
	MTYPE = 0100b: 32-bit-wide SBSRAM (C621x/C671x)
	32-bit-wide programmable synchronous memory (C64x)
	MTYPE = 1000b: 8-bit-wide SDRAM
	MTYPE = 1001b: 16-bit-wide SDRAM
	MTYPE = 1010b: 8-bit-wide SBSRAM (C621x/C671x)
	8-bit-wide programmable synchronous memory (C64x)
	MTYPE = 1011b: 16-bit-wide SBSRAM (C621x/C671x)
	16-bit-wide programmable synchronous memory (C64x)
	MTYPE = 1100b: 64-bit-wide asynchronous interface (C64x only)
	MTYPE = 1101b: 64-bit-wide SDRAM (C64x only)
	MTYPE = 1110b: 64-bit-wide programmable synchronous memory (C64x only)

TA	Turn-around time (C621x/C671x/C64x only). 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).

† Clock cycles are in terms of CLKOUT1 for C620x/C670x, ECLKOUT for the C621x/C671x, and ECLKOUT1 for the C64x.
‡ 32-bit and 64–bit interfaces (MTYPE=0010b, 0011b, 0100b, 1100b, 1101b, 1110b) do not apply to C6712 and C64x EMIFB.

For the C64x, a secondary CE space control register is added for the programmable synchronous interface. This register controls the cycle timing of programmable synchronous memory accesses, and the clock used for synchronization for the specific CE space. The CE space secondary control register (CExSEC) is shown in Figure 10–10 and summarized in Table 10–6. The CExSEC applies only to C64x programmable synchronous memory interface.

Figure 10–10. CE Space Secondary Control Register (CExSEC) – TMS320C64x Only

31–16
Reserved
R, +0

15–9	8	7	6	5	4	3–2	1–0
Reserved	Rsv†	Rsv	SNCCLK	RENEN	CEEXT	SYNCWL	SYNCRL
R,+0	RW, +0	R, +0	RW, +0	RW, +0	RW, +0	RW, +00	RW, +10

† RW reserved bit fields should always be written with their default values when modifying the CExSEC.

Table 10–6. CE Space Secondary Control Register (CExSEC) Field Descriptions (TMS320C64x only)

Field	**Description**

SYNCRL	Synchronous interface data read latency
	
	SYNCRL = 00: 0 cycle read latency
	SYNCRL = 01: 1 cycle read latency
	SYNCRL = 10: 2 cycle read latency
	SYNCRL = 11: 3 cycle read latency

SYNCWL	Synchronous interface data write latency
	
	SYNCWL = 00: 0 cycle write latency
	SYNCWL = 01: 1 cycle write latency
	SYNCWL = 10: 2 cycle write latency
	SYNCWL = 11: 3 cycle write latency

CEEXT	CE extension register CEEXT = 0: CE goes inactive after the final command has been issued (not necessarily when all the data has been latched). CEEXT = 1: On read cycles, the CE signal will go active when $\overline{\text{SOE}}$ goes active and will stay active until $\overline{\text{SOE}}$ goes inactive. The $\overline{\text{SOE}}$ timing is controlled by SYNCRL. (used for synchronous FIFO reads with glue, where $\overline{\text{CE}}$ gates $\overline{\text{OE}}$)
RENEN	Read Enable Enable RENEN = 0: ADS mode. $\overline{\text{SADS}}$/$\overline{\text{SRE}}$ signal acts as $\overline{\text{SADS}}$ signal. $\overline{\text{SADS}}$ goes active for reads, writes, and deselect. Deselect is issued after a command is completed if no new commands are pending from the EDMA. (used for SBSRAM or ZBT SRAM interface) RENEN = 1: Read Enable mode. $\overline{\text{SADS}}$/$\overline{\text{SRE}}$ signal acts as $\overline{\text{SRE}}$ signal. $\overline{\text{SRE}}$ goes low only for reads. No deselect cycle is issued. (used for FIFO interface)
SNCCLK	Synchronization Clock SNCCLK = 0, control/data signals for this CE space are synchronized to ECLKOUT1 SNCCLK = 1, control/data for this CE space are synchronized to ECLKOUT2

10.2.3 EMIF SDRAM Control Register

The SDRAM control register (shown in Figure 10–11) 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 10–11 and described in Table 10–7. These registers should not be modified while accessing SDRAM.

Figure 10–11. EMIF SDRAM Control Register (SDCTL)

C620x/C670x:

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

C621x/C671x:

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

C64x:

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	11–1	0
TRC	Reserved	SLFRFR†
RW, +1111	R, +000 0000 0000	RW, +0

† SLFRFR only applies to EMIFA. Bit 0 is reserved, RW+0, on EMIFB.

Table 10–7. EMIF-to-SDRAM Control Register (SDCTL) Field Descriptions

Field	Description
TRC	Specifies the t_{RC} value of the SDRAM in EMIF clock cycles TRC = t_{RC} / t_{cyc}[†] – 1
TRP	Specifies the t_{RP} value of the SDRAM in EMIF clock cycles TRP = t_{RP} / t_{cyc}[†] – 1
TRCD[‡]	Specifies the t_{RCD} value of the SDRAM in EMIF clock cycles[‡] TRCD = t_{RCD} / t_{cyc}[†] – 1

INIT	Forces initialization of all SDRAM present
	INIT = 0: no effect
	INIT = 1: initialize SDRAM in each CE space configured for SDRAM. EMIF automatically changes INIT back to 0 after SDRAM initialization is performed.

RFEN	Refresh enable
	RFEN = 0: SDRAM refresh disabled
	RFEN = 1: SDRAM refresh enabled

SDWID	SDRAM column width select (C620x/C670x)
	SDWID = 0: 9 column address pins (512 elements per row)
	SDWID = 1: 8 column address pins (256 elements per row)

SDCSZ	SDRAM column size (C621x/C671x/C64x)
	SDCSZ = 00: 9 column address pins (512 elements per row)
	SDCSZ = 01: 8 column address pins (256 elements per row)
	SDCSZ = 10: 10 column address pins (1024 elements per row)
	SDCSZ = 11: reserved

SDRSZ	SDRAM row size (C621x/C671x/C64x)
	SDRSZ = 00: 11 row address pins (2048 rows per bank)
	SDRSZ = 01: 12 row address pins (4096 rows per bank)
	SDRSZ = 10: 13 row address pins (8192 rows per bank)
	SDRSZ = 11: reserved

[†] t_{cyc} refers to the EMIF clock period, which is equal to CLKOUT2 period for the C620x/C670x, ECLKOUT period for the C621x/C671x, or ECLKOUT1 period for the C64x.
[‡] On the C64x, TRCD specifies the number of ECLKOUT1 cycles between an ACTV command and a READ or WRT command (CAS). The specified separation is maintained while driving write data one cycle earlier.

Table 10–7. EMIF-to-SDRAM Control Register (SDCTL) Field Descriptions (Continued)

Field	Description

SDBSZ	SDRAM bank size (C621x/C671x/C64x)
	
	SDBSZ = 0: one bank-select pin (two banks)
	SDBSZ = 1: two bank-select pins (four banks)

SLFRFR	Self-refresh mode (C64x)
	
	If SDRAM is used in the system,
	SLFRFR = 0: self-refresh mode disabled
	SLFRFR = 1: self-refresh mode enabled
	
	If SDRAM is not used,
	SLFRFR = 0: general-purpose output SDCKE = 1
	SLFRFR = 1: general-purpose output SDCKE = 0

† t_{cyc} refers to the EMIF clock period, which is equal to CLKOUT2 period for the C620x/C670x, ECLKOUT period for the C621x/C671x, or ECLKOUT1 period for the C64x.
‡ On the C64x, TRCD specifies the number of ECLKOUT1 cycles between an ACTV command and a READ or WRT command (CAS). The specified separation is maintained while driving write data one cycle earlier.

The EMIF automatically clears the INIT field to zero after it performs SDRAM initialization. When RESET goes inactive, none of the CE spaces will be configured as SDRAM, so the INIT field will quickly change from 1 to 0. The CPU should initialize all of the CE space control registers and the SDRAM extension register before it sets the INIT bit back to 1.

For C64x, the SLFRFR bit enables the self refresh mode, in which the EMIF places the external SDRAM in a low power mode to maintain valid data while consuming a minimal amount of power. If SDRAM is not in use by the system, then the SLFRFR bit can be used to control SDCKE as a general-purpose output. See section 10.5.5 for details.

10.2.4 EMIF SDRAM Timing Register (SDTIM)

The SDRAM timing register controls the refresh period in terms of EMIF clock cycles. 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 SDINT (synchronization event to EDMA and interrupt source to CPU) is asserted. See section 10.5.4 for more information on SDRAM refresh.

Figure 10–12 and Table 10–8 describe the fields of the SDRAM timing register.

The C621x/C671x/C64x 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 10–12. EMIF SDRAM Timing Register (SDTIM)

31 26	25 24	23 12	11 0
Reserved	XRFR‡	COUNTER	PERIOD
R, +0000 00	R, +00† RW,+00‡	R, +0000 1000 0000† R, +0101 1101 1100‡	RW, +0000 0100 0000† RW, +0101 1101 1100‡

† Applies to C620x/C670x
‡ Applies to C621x/C671x/C64x

Table 10–8. EMIF SDRAM Timing Register Field Descriptions

Field	Description
PERIOD	Refresh period in EMIF clock cycles†
COUNTER	Current value of the refresh counter
XRFR‡	Extra refreshes: controls the number of refreshes performed to SDRAM when the refresh counter expires.‡

† For C620x/C670x, EMIF clock cycles = CLKOUT2 cycles.
For C621x/C671x, EMIF clock cycles = ECLKOUT cycles.
For C64x, EMIF clock cycles = ECLKOUT1 cycles.
‡ Applies to C621x/C671x/C64x only.

For the C621x/C671x/C64x, the initial value for the COUNTER field and also the PERIOD field has been increased to 0x5DC (1500 clock cycles). With a 10-ns EMIF cycle time, there is a 15-μs time between refresh operations. SDRAMs typically require 15.625 μs per refresh.

The XRFR field controls the number of refreshes that take place when the counter reaches zero (“00” for 1, “01” for 2, “10” for 3, and “11” for 4). As an example, since all banks must be deactivated to perform a refresh, it might be desirable to perform two refreshes half as often.

For the C621x/C671x/C64x, all refresh requests are considered high priority. When it is time to refresh, the refresh is performed immediately (though transfers in progress are allowed to complete). All banks are deactivated before a refresh command is issued. When the refresh command is complete, the banks are *not* restored to their state before refresh.

10.2.5 TMS320C621x/C671x/C64x SDRAM Extension Register (SDEXT)

The SDRAM extension register of the C621x/C671x/C64x allows programming of many parameters of SDRAM. The programmability offers two distinct advantages. First, it allows an interface to a wide variety of SDRAMs and is not limited to a few configurations or speed characteristics. Second, the EMIF can maintain seamless data transfer from external SDRAM due to features like hidden precharge and multiple open banks. Figure 10–13 shows the SDRAM extension register and Table 10–9 discusses these parameters.

Figure 10–13. TMS320C621x/C671x/C64x SDRAM Extension Register (SDEXT)

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,+01	RW,+1	RW,+11† RW,+10‡	RW,+101	RW,+11	RW,+1	RW,+10	RW,+01	RW,+1	RW,+111	RW,+1

† Applies to C621x/C671x.
‡ Applies to C64x.

Table 10–9. TMS320C621x/C671x/C64x SDRAM Extension Register (SDEXT) 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} / t_{cyc}[‡] – 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} / t_{cyc}[‡] – 1
THZP	Specifies t_{HZP} (also known as t_{ROH}) value of the SDRAM in ECLKOUT cycles THZP = t_{HZP} / t_{cyc}[‡] – 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

[†] For C64x, ECLKOUT referenced in this table is equivalent to ECLKOUT1.
[‡] t_{cyc} refers to the EMIF clock period, which is equal to ECLKOUT period for the C621x/C671x, ECLKOUT1 period for C64x.

10.3 Memory Width and Byte Alignment

10.3.1 C620x/C670x Memory Width and Byte Alignment

The C620x/C670x EMIF supports 32-bit-wide ASRAM, SDRAM, and SBSRAM interface in both big- and little-endian modes. CE1 space supports x16 and x8 read-only memory (ROM) interfaces. The packing format in ROM is always little-endian, regardless of the value of the LENDIAN configuration bit. Table 10–10 summarizes the addressable memory ranges on the C620x/C670x device.

Table 10–10. TMS320C620x/C670x Addressable Memory Ranges

Memory type	Memory width	Maximum addressable bytes per CE space	Address output on EA[21:2]	Represents
ASRAM	x32	4M	A[21:2]	Word address
SBSRAM	x32	4M	A[21:2]	Word address
SDRAM	x32	16M	See section 10.5	Word address

10.3.2 C621x/C671x Memory Width and Byte Alignment

The C621x/C671x EMIF supports memory widths of 8 bits, 16 bits, and 32 bits, including reads and writes of both big- and little-endian devices. The C6712 EMIF supports memory widths of 8 bits and 16 bits only. 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 EA2, regardless of the width of the device. Accesses to 8-bit memories have logical address bit 0 output on EA2. Table 10–11 summarizes the addressable memory ranges on the C621x/C671x device.

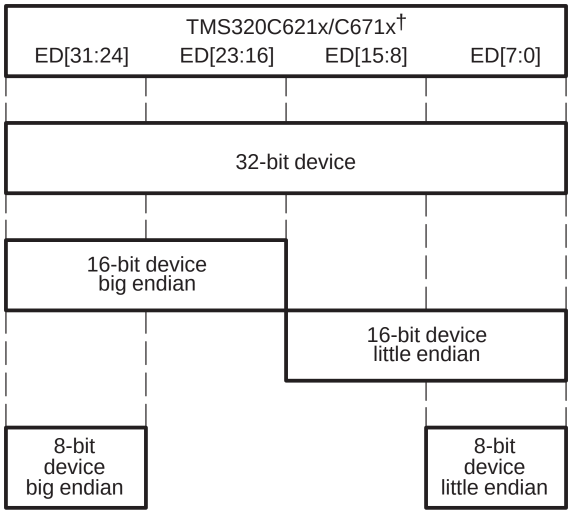
Table 10–11. TMS320C621x/C671x Addressable Memory Ranges

Memory type	Memory width	Maximum addressable bytes per CE space	Address output on EA[21:2]	Represents
ASRAM	x8	1M	A[19:0]	Byte address
	x16	2M	A[20:1]	Halfword address
	x32†	4M	A[21:2]	Word address
SBSRAM	x8	1M	A[19:0]	Byte address
	x16	2M	A[20:1]	Halfword address
	x32†	4M	A[21:2]	Word address
SDRAM	x8	32M	See section 10.5	Byte address
	x16	64M	See section 10.5	Halfword address
	x32 †	128M	See section 10.5	Word address

† 32-bit interface does not apply to C6712.

For C621x/C671x, 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 10–14.

Figure 10–14. TMS320C621x/C671x Byte Alignment by Endianness



† ED[31:16] do not apply to C6712.

10.3.3 C64x Memory Width and Byte Alignment

EMIFA supports memory widths of 8 bits, 16 bits, 32 bits, and 64 bits. EMIFB supports memory widths of 8 bits and 16 bits. Table 10–12 summarizes the addressable memory ranges on the C64x device. Both big- and little-endian formats are supported.

Table 10–12. TMS320C64x Addressable Memory Ranges

Memory type	Memory width	Maximum addressable bytes per CE space	Address output on EA[22:3] (EMIFA) EA[20:1] (EMIFB)†	Represents
ASRAM	x8	1M	A[19:0]	Byte address
	x16	2M	A[20:1]	Halfword address
	x32	4M	A[21:2]	Word address
	x64	8M	A[22:3]	Doubleword address
Programmable Sync Memory	x8	1M	A[19:0]	Byte address
	x16	2M	A[20:1]	Halfword address
	x32	4M	A[21:2]	Word address
	x64	8M	A[22:3]	Doubleword address
SDRAM	x8	32M	See section 10.5	Byte address
	x16	64M	See section 10.5	Halfword address
	x32	128M	See section 10.5	Word address
	x64	256M	See section 10.5	Doubleword address

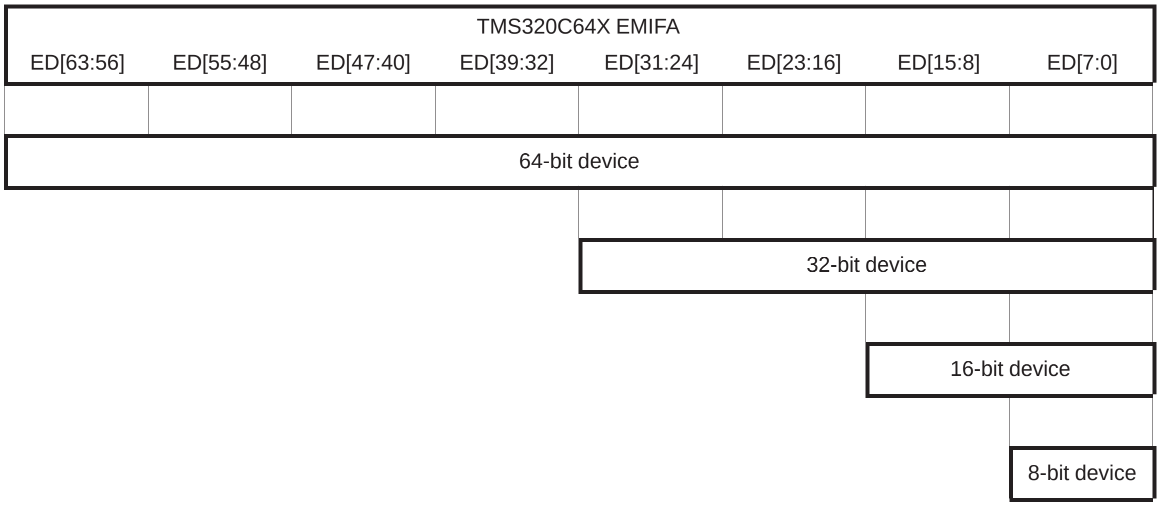
† The shaded rows (x32 and x64 interface) do not apply to EMIFB.

Similar to the C621x/C671x, packing and unpacking is automatically performed by the C64x EMIF for accesses to external memories of less than 64 bits (EMIFA) or 16 bits (EMIFB).

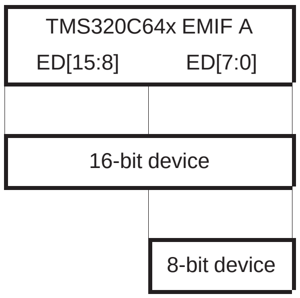
Figure 10–15 shows the byte lane used on C64x. The external memory is always right aligned to the ED[7:0] side of the bus. The endianness mode determines whether byte lane 0 (ED[7:0]) is accessed as byte address 0 (little endian) or as byte address N (big endian), where 2^N is memory width in bytes. ressed as either byte address 0 (big endian) or as byte address N (little endian).

Figure 10–15. TMS320C64x Byte Alignment by Endianness

EMIFA (64-bit bus):



EMIFB (16-bit bus):



10.4 Command-to-Command Turnaround Time

All C6000 EMIF have a one cycle command–to–command turnaround time. At least 1 data dead cycle is always included between commands so that read data and write data are never driven in the same cycle.

10.5 SDRAM Interface

The C6000 EMIF supports SDRAM commands shown in Table 10–13. Table 10–14 shows the signal truth table for the SDRAM commands. The 16M-bit and 64M-bit SDRAM interfaces are shown in Figure 10–16, Figure 10–17, Figure 10–18, and Figure 10–19. Table 10–15, Table 10–16, and Table 10–17 list all of the possible SDRAM configurations available via the C6000. Table 10–18 summarizes the pin connection and related signals specific to SDRAM operation.

The C620x/C670x EMIF allows programming of the SDRAM column size. A single page can be open in each CE space.

The C621x/C671x/C64x 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 C621x/C671x/C64x is able to open up to four pages of SDRAM simultaneously. The pages can all be in different banks of a single CE space, or distributed across multiple CE spaces. Only one page can be open per bank at a time. The C621x/C671x/C64x can interface to any SDRAM that has 8 to 10 column address pins, 11 to 13 row address pins, and two or four banks.

In addition to all the C621x/C671x SDRAM interface features, the C64x EMIF supports the SDRAM self refresh mode, and supports the Least Recently Used (LRU) page replacement strategy instead of random replacement strategy for better performance. See section 10.5.5 and section 10.5.2 for details.

Table 10–13. EMIF SDRAM Commands

Command	Function
DCAB	Deactivate (also known as PRECHARGE) 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
SLFREFR‡	Self-refresh mode‡

† TMS320C621x/C671x/C64x only
‡ TMS320C64x only

Table 10–14. Truth Table for SDRAM Commands

SDRAM:	**CKE**	$\overline{\textbf{CS}}$	$\overline{\textbf{RAS}}$	$\overline{\textbf{CAS}}$	$\overline{\textbf{W}}$	**A[19:16]**	**A[15:11]**	**A10**	**A[9:0]**
16-bit[†] EMIF:	**SDCKE**	$\overline{\textbf{CE}}$	$\overline{\textbf{SDRAS}}$	$\overline{\textbf{SDCAS}}$	$\overline{\textbf{SDWE}}$	**EA[20:17][‡]**	**EA[16:12]**	**EA11**	**EA[10:1]**
32-bit[†] EMIF:	**SDCKE[§]**	$\overline{\textbf{CE}}$	$\overline{\textbf{SDRAS}}$	$\overline{\textbf{SDCAS}}$	$\overline{\textbf{SDWE}}$	**EA[21:18][¶]**	**EA[17:13]**	**EA12[#]**	**EA[11:2]**
64-bit[†] EMIF:	**SDCKE**	$\overline{\textbf{CE}}$	$\overline{\textbf{SDRAS}}$	$\overline{\textbf{SDCAS}}$	$\overline{\textbf{SDWE}}$	**EA[22:19][‡]**	**EA[18:14]**	**EA13**	**EA[12:3]**
ACTV	H	L	L	H	H	0001b OR 0000b[¶]	Bank/Row	Row	Row
READ	H	L	H	L	H	X	Bank/Col	L	Col
WRT	H	L	H	L	L	X	Bank/Col	L	Col
MRS	H	L	L	L	L	L	L/Mode	Mode	Mode
DCAB	H	L	L	H	L	X	X	H	X
DEAC	H	L	L	H	L	X	Bank/X	L	X
REFR	H	L	L	L	H	X	X	X	X
SLFREFR	L	L	L	L	H	X	X	X	X

Bank = Bank Address
Row = Row Address
Col = Column Address
L = 0b = Low;
H = 1b = High
Mode = Mode Select
X = Previous value

[†] 16-bit EMIF includes C64x EMIFB
32-bit EMIF includes all C62x/C67x EMIF.
64-bit EMIF includes C64x EMIFA.

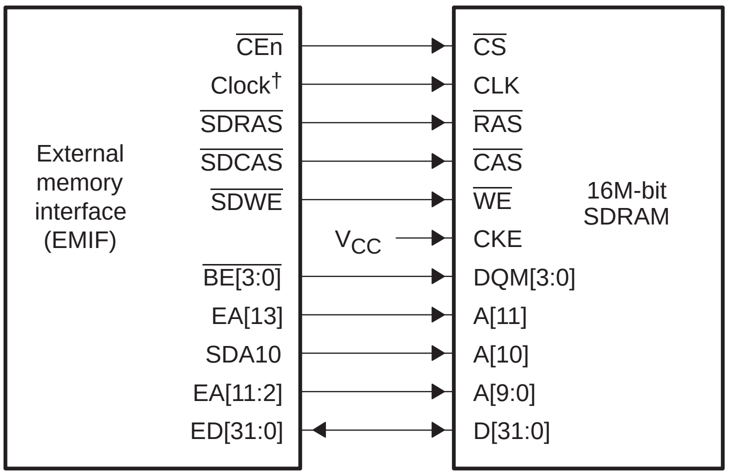
[‡] For C64x, upper address bits are used during ACTV to indicate non-PDT (0001b) vs. PDT (0000b) access. During all other accesses, address bits indicated with X hold previous value.

[§] SDCKE does not exist on C62x/C67x.

[¶] For C62x/C67x, upper address bits are reserved for future use. Undefined.

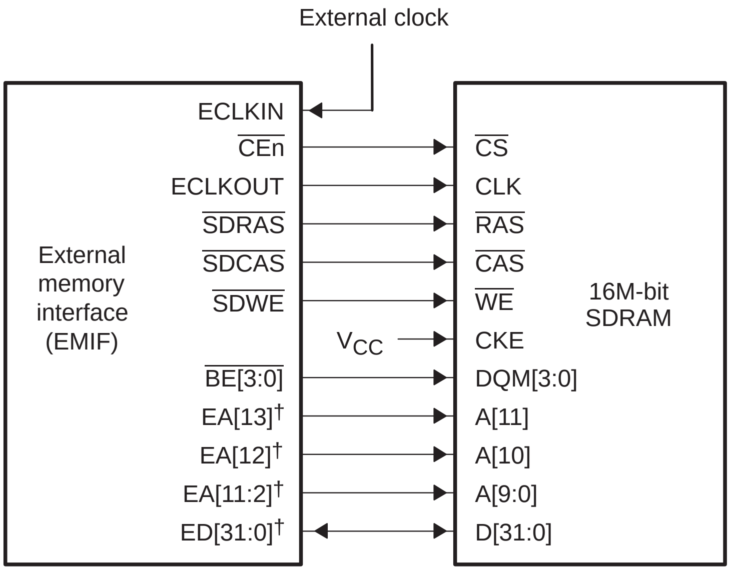
[#] SDA10 is used on C620x/C670x. EA12 is used on C621x/C671x.

Figure 10–16. TMS320C620x/C670x EMIF to 16M-Bit SDRAM Interface



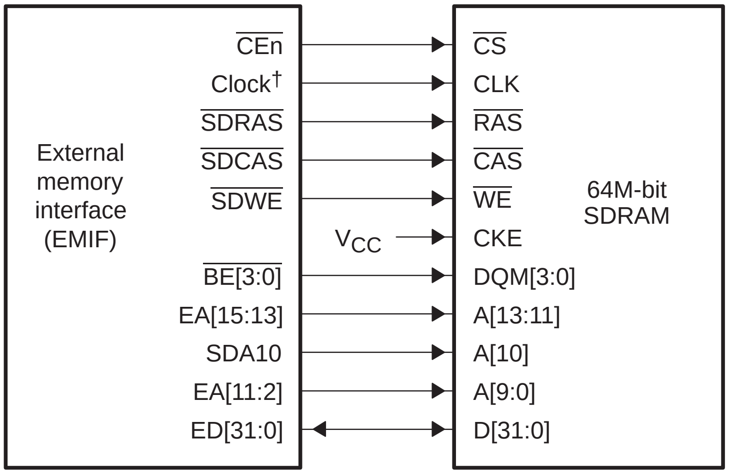
[†] Clock = SDCLK for C6201/C6701.
= CLKOUT2 for all other C620x/C670x.

Figure 10–17. TMS320C621x/C671x EMIF to 16M-Bit SDRAM Interface



[†] C6712,EA[12:1] and ED[15:0] are used instead.

Figure 10–18. TMS320C620x/C670x EMIF to 64M-Bit SDRAM Interface



† Clock = SDCLK for C6201/C6701.
= CLKOUT2 for all other C620x/C670x.

Figure 10–19. TMS320C64x EMIFA to 64M-Bit SDRAM Interface

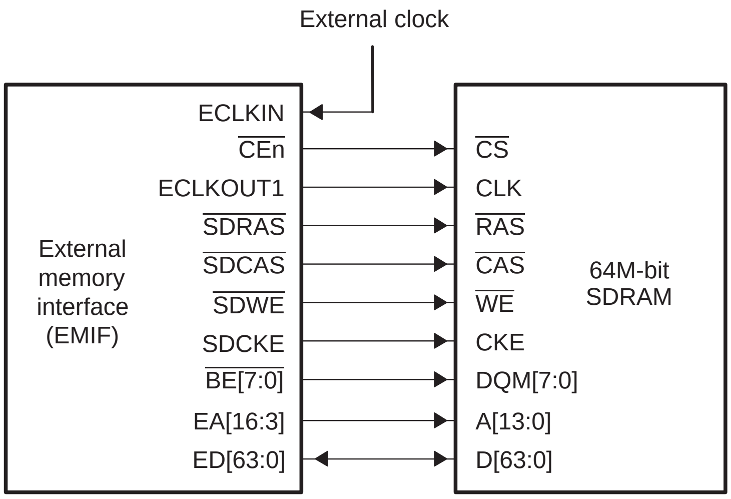


Table 10–15. TMS320C620x/C670x Compatible SDRAM

SDRAM Size	Banks	Width	Depth	Devices/ CE	Addressable Space, MBytes		Column Address	Row Address	Bank Select	Pre-charge
16M bit	2	x8	1M	4	8M	SDRAM	A8–A0	A10–A0	A11	A10
						EMIF	EA10–EA2	SDA10, EA11–EA2	EA13	SDA10
	2	x16	512K	2	4M	SDRAM	A7–A0	A10–A0	A11	A10
						EMIF	EA9–EA2	SDA10, EA11–EA2	EA13	SDA10
64M bit	4	x16	1M	2	16M	SDRAM	A7–A0	A11–A0	A13–A12	A10
						EMIF	EA9–EA2	SDA10, EA13–EA2	EA15–EA14	SDA10
	4	x32	512K	1	8M	SDRAM	A7–A0	A10–A0	A12–A11	A10
						EMIF	EA9–EA2	SDA10, EA11–EA2	EA14–EA13	SDA10
128M	4	x32	1M	1	16M	SDRAM	A7–A0	A11–A0	A13–A12	A10
						EMIF	EA9–EA2	EA13–EA2	EA15–EA14	EA12

Table 10–16. TMS320C621x/C671x Compatible SDRAM

SDRAM Size	Banks	Width	Depth	Max Devices/ CE	Address-able space, MBytes		Column Address	Row Address	Bank Se-lect	Pre-charge
16M bit	2	x4	2M	8	16M	SDRAM	A9–A0	A10–A0	A11	A10
						EMIF	EA11–EA2	EA12–EA2	EA13	EA12
	2	x8	1M	4	8M	SDRAM	A8–A0	A10–A0	A11	A10
						EMIF	EA10–EA2	EA12–EA2	EA13	EA12
	2	x16	512K	2	4M	SDRAM	A7–A0	A10–A0	A11	A10
						EMIF	EA9–EA2	EA12–EA2	EA13	EA12
64M bit	4	x4	4M	8	64M	SDRAM	A9–A0	A11–A0	A13–A12	A10
						EMIF	EA11–EA2	EA13–EA2	EA15–EA14	EA12
	4	x8	2M	4	32M	SDRAM	A8–A0	A11–A0	A13–A12	A10
						EMIF	EA10–EA2	EA13–EA2	EA15–EA14	EA12
	4	x16	1M	2	16M	SDRAM	A7–A0	A11–A0	A13–A12	A10
						EMIF	EA9–EA2	EA13–EA2	EA15–EA14	EA12
	4	x32†	512K	1	8M	SDRAM	A7–A0	A10–A0	A12–A11	A10
						EMIF	EA9–EA2	EA12–EA2	EA14–EA13	EA12
128M bit	4	x8	4M	4	64M	SDRAM	A9–A0	A11–A0	A13–A12	A10
						EMIF	EA11–EA2	EA13–EA2	EA15–EA14	EA12
	4	x16	2M	2	32M	SDRAM	A8–A0	A11–A0	A13–A12	A10
						EMIF	EA10–EA2	EA13–EA2	EA15–EA14	EA12
	4	x32	1M	1	16M	SDRAM	A7–A0	A11–A0	A13–A12	A10
						EMIF	EA9–EA2	EA13–EA2	EA15–EA14	EA12
256M bit	4	x8	8M	4	128M	SDRAM	A9–A0	A12–A0	A14–A13	A10
						EMIF	EA11–EA2	EA14–EA2	EA16–EA15	EA12
	4	x16	4M	2	64M	SDRAM	A8–A0	A12–A0	A14–A13	A10
						EMIF	EA10–EA2	EA14–EA2	EA16–EA15	EA12

† The x32 Width does not apply to C6712.

Table 10–17. TMS320C64x Compatible SDRAM

SDRAM Size	Banks	Width	Depth	Max Devices /CE	Address-able space, MBytes		Column Address	Row Address	Bank Select	Pre-charge
16M bit	2	x4	2M	16	32M	SDRAM	A9–A0	A10–A0	A11	A10
						EMIFA	EA12–EA3	EA13–EA3	EA14	EA13
						EMIFB	EA10–EA1	EA11–EA1	EA12	EA11
	2	x8	1M	8	16M	SDRAM	A8–A0	A10–A0	A11	A10
						EMIFA	EA11–EA3	EA13–EA3	EA14	EA13
						EMIFB	EA9–EA1	EA11–EA1	EA12	EA11
	2	x16	512K	4	8M	SDRAM	A7–A0	A10–A0	A11	A10
						EMIFA	EA10–EA3	EA13–EA3	EA12	EA11
64M bit	4	x4	4M	16	128M	SDRAM	A9–A0	A11–A0	A13–A12	A10
						EMIFA	EA12–EA3	EA14–EA3	EA16–EA15	EA13
						EMIFB	EA10–EA1	EA12–EA1	EA14–EA13	EA11
	4	x8	2M	8	64M	SDRAM	A8–A0	A11–A0	A13–A12	A10
						EMIFA	EA11–EA3	EA14–EA3	EA16–EA15	EA13
						EMIFB	EA9–EA1	EA12–EA1	EA14–EA13	EA11
	4	x16	1M	4	32M	SDRAM	A7–A0	A11–A0	A13–A12	A10
						EMIFA	EA10–EA3	EA14–EA3	EA16–EA15	EA13
						EMIFB	EA8–EA1	EA12–EA1	EA14–EA13	EA11
	4	x32	512K	2	16M	SDRAM	A7–A0	A10–A0	A12–A11	A10
						EMIFA	EA10–EA3	EA13–EA3	EA15–EA14	EA13
						EMIFB	–	–	–	–
128M bit	4	x8	4M	8	128M	SDRAM	A9–A0	A11–A0	A13–A12	A10
						EMIFA	EA12–EA3	EA14–EA3	EA16–EA15	EA13
						EMIFB	EA10–EA1	EA12–EA1	EA14–EA13	EA11
	4	x16	2M	4	64M	SDRAM	A8–A0	A11–A0	A13–A12	A10
						EMIFA	EA11–EA3	EA14–EA3	EA16–EA15	EA13
						EMIFB	EA9–EA1	EA12–EA1	EA14–EA13	EA11
						SDRAM	A7–A0	A11–A0	A13–A12	A10
	4	x32	1M	2	32M	EMIFA	EA10–EA3	EA14–EA3	EA16–EA15	EA13
						EMIFB	EA8–EA1	EA12–EA1	EA14–EA11	EA11
256M bit	4	x8	8M	8	256M	SDRAM	A9–A0	A12–A0	A14–A13	A10
						EMIFA	EA12–EA3	EA15–EA3	EA17–EA16	EA13
						EMIFB	EA10–EA1	EA13–EA1	EA15–EA14	EA11
	4	x16	4M	4	128M	SDRAM	A8–A0	A12–A0	A14–A13	A10
						EMIFA	EA11–EA3	EA15–EA3	EA17–EA16	EA13
						EMIFB	EA9–EA1	EA13–EA1	EA15–EA14	EA11

Table 10–18. SDRAM 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. (C620x/C670x 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.
$\overline{\text{BEx}}$	DQMx	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 $\overline{\text{BE}}$ outputs.
$\overline{\text{CE3}}$, $\overline{\text{CE2}}$, $\overline{\text{CE1}}$[‡], 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. $\overline{\text{CE1}}$ does not support SDRAM on C620x/C670x.
SDCKE[§]	CKE	CKE clock enable. For C64x, SDCKE is connected to CKE to minimize SDRAM power consumption when self-refresh mode is enabled. For C62x/C67x SDRAM interface, CKE is tied high (on the SDRAM device) since the Self Refresh command is not supported.
CLKOUT2	CLK	SDRAM clock input. Runs at 1/2 the CPU clock rate.Used for synchronous memory interface on the C6202(B)/C6203(B)/C6204/C6205.
SDCLK	CLK	SDRAM clock input. Runs at 1/2 the CPU clock rate.Used for SDRAM interface on C6201/C6701
ECLKOUT	CLK	SDRAM clock input. Used for synchronous memory interface on the C621x/C671x/C64x. For C621x/C671x, runs at ECLKIN rate. For C64x, ECLKOUT1 is used. ECLKOUT1 runs at EMIF input clock rate (ECLKIN, CPU/4 clock, or CPU/6 clock).

[†] SDA10 is used on C620x/C670x. EA12 is used on C621x/C671x. EA13 is used on C64x EMIFA. EA11 is used on C64x EMIFB.
[‡] For C620x/C670x, $\overline{\text{CE1}}$ does not support SDRAM.
[§] SDCKE exists on C64x only.

Table 10–19 is an overview of similarities and differences on the C6000 SDRAM interface.

Table 10–19. TMS320C6000 SDRAM Interface Summary

	C62x/C67x			C64x	
	C6201 C6701	**Other C620x C670x**[†]	**C621x C671x**[‡]	**EMIFA**	**EMIFB**
Interface width	32-bit	32-bit	32-, 16-, 8-bit[‡]	64-, 32-, 16-, 8-bit	16-, 8-bit
SDRAM clock	SDCLK	CLKOUT2	ECLKOUT	ECLKOUT1	ECLKOUT1
Registers for SDRAM timing parameters	SDCTL SDTIM	SDCTL SDTIM	SDCTL SDTIM SDEXT	SDCTL SDTIM SDEXT	SDCTL SDTIM SDEXT
SDRAM Control signals	Dedicated SDRAM control signals	MUXed with SBSRAM control signals	MUXed with SBSRAM and Async control signals.	MUXed with Async and Programmable Sync control signals.	MUXed with Async and Programmable Sync control signals.
Number of open pages	Single open page per CE space	Single open page per CE space	4 open pages in any CE space	4 open pages in any CE space	4 open pages in any CE space
Programmable SDRAM configuration	8- or 9-column address bits	8- or 9-column address bits	column, row, and bank size	column, row, and bank size	column, row, and bank size
Burst Mode	Not supported. Performs bursts by issuing back-to-back commands	Not supported. Performs bursts by issuing back-to-back commands	Supports SDRAM burst mode with a 4-word burst	Supports SDRAM burst mode with a 4-word burst	Supports SDRAM burst mode with a 4-word burst
Background refresh	Yes	Yes	No	No	No
Precharge pin	SDA10	SDA10	EA12	EA13	EA11
SDRAM Self-Refresh Mode	No	No	No	Yes	No
Page Replacement	Fixed	Fixed	Random	LRU	LRU

[†] This column applies to all C620x/C670x devices except C6201/C6701.
[‡] C6712 interfaces to 8- and 16-bit SDRAM only.

10.5.1 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 eight 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.

10.5.2 Monitoring Page Boundaries

SDRAM is a paged memory type, thus 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 row in internal page register(s), then performs compares against that address for subsequent accesses to any SDRAM CE space.

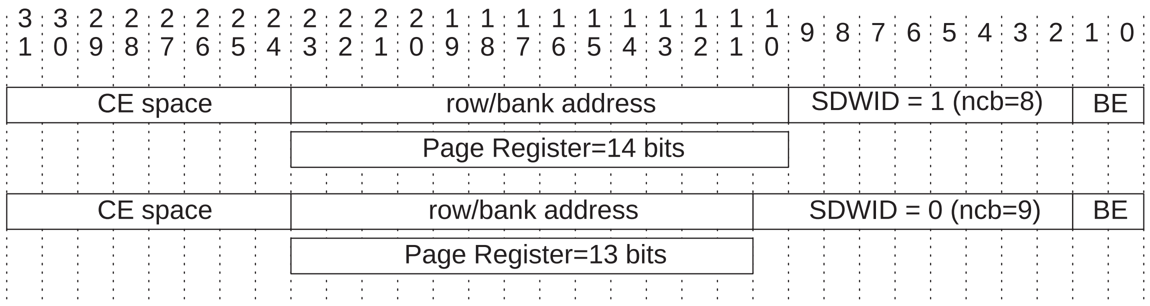
For all C6000 devices, ending the current access is not a condition that forces the active SDRAM row to be closed. The EMIF leaves the active row open until it becomes necessary to close it. This decreases the deactivate–reactivate overhead and allows the interface to capitalize fully on the address locality of memory accesses.

10.5.2.1 C620x/C670x Monitoring Page Boundaries

The C620x/C670x storage and comparison is performed independently for each CE space. The C620x/C670x EMIF has 4 internal page registers. Each page register corresponds to a single CE space. If a given CE space is configured for SDRAM operation (by the MTYPE field), the corresponding page register is used for accesses to that CE space. If the CE space is not configured for SDRAM operation, the corresponding page register is not used. Therefore, the C620x/C670x devices can support a single open page per CE space.

Address bits are compared, during an SDRAM access, to determine whether the page is open. The number of address bits compared is a function of the page size programmed in the SDWID field in the EMIF SDRAM control register. If SDWID = 0, the EMIF expects CE spaces configured as SDRAM to have a page size of 512 elements (i.e., number of column address bits = NCB = 9). Thus, the logical byte address bits compared are 23–11. Logical addresses with the same bits 23:11 belong to the same page. If SDWID = 1, the EMIF expects CE spaces with SDRAM to have a page size of 256 elements (NCB = 8). Thus, the logical byte address bits compared are 23–10. The logical address bits 25 to 24 (and above) determine the 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. Figure 10–20 details how a 32–bit logical address maps to the page register.

Figure 10–20. TMS320C620x/C670x Logical Address to Page Register Mapping



Note: ncb = number of column address bits

10.5.2.2 C621x/C671x Monitoring Page Boundaries

For the C621x/C671x, 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 controls which logical address bits are compared to determine if a page is open: SDCSZ (which controls NCB), SDRSZ (which controls the number of row address bits, or NRB), and SDBSZ (which controls the number of bank address bits, or NBB) Logical address bits above the bank address bit are not used as part of the page comparison, nor are they used when issuing the row/column commands to the external SDRAM. This implies that the maximum addressable space is limited by the specific configuration of SDRAM used.

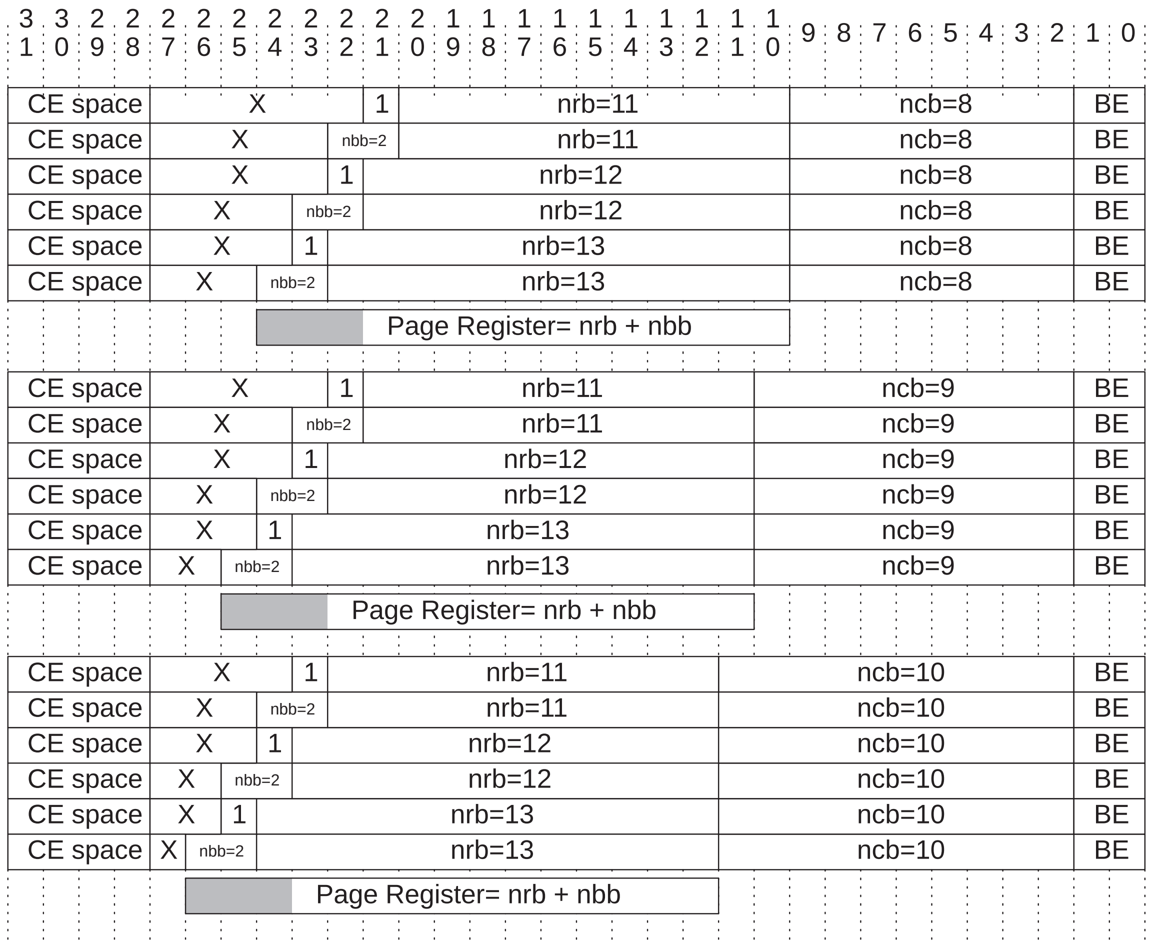
For example, a typical 2–bank $\times$ 512K $\times$ 16–bit SDRAM has settings of 8 column address bits, 11 row address bits, and 1 bank bit, With this configuration, the maximum amount of addressable space per CE space is $2^{(NCB+NRB+NBB+2)}$, or 4 Mbytes .

Note: The *+2* term is appropriate for calculating the addressable space in terms of bytes for a 32 bit interface. If only 16 bits of the bus are populated then *+1* is used, and if only 8 bits of the bus are populated then *+0* is used.

Figure 10–21 details how a 32–bit logical address maps to the page register. For 16– or 8–bit interfaces, the BE portion of the logical address is reduced to 1 bit for 16–bit SDRAM and 0 bits for 8–bit SDRAM. The NCB/NRB/NBB (and page register) shift accordingly.

The C621x/C671x EMIF employs a random page replacement strategy when necessary. This occurs when the total number of external SDRAM banks (not devices) is greater than 4, since the EMIF only contains 4 page registers. This can occur when multiple CE spaces of SDRAM are used. When the number of total banks of SDRAM is less than or equal to 4, the page replacement strategy is fixed since SDRAM requires that only 1 page can be open within a given bank. If a page miss is detected either during an access where a different page was previously accessed in the same CE space (fixed replacement), or if a page must be closed within a different CE space to allow a page register to be assigned for the current access (random replacement), the C621x/C671x performs a DEAC command and starts a new row access.

Figure 10–21. TMS320C621x/671x Logical Address to Page register Mapping


31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
CE space | X | 1 | nrb=11 | ncb=8 | BE
CE space | X | nbb=2 | nrb=11 | ncb=8 | BE
CE space | X | 1 | nrb=12 | ncb=8 | BE
CE space | X | nbb=2 | nrb=12 | ncb=8 | BE
CE space | X | 1 | nrb=13 | ncb=8 | BE
CE space | X | nbb=2 | nrb=13 | ncb=8 | BE
Page Register= nrb + nbb
CE space | X | 1 | nrb=11 | ncb=9 | BE
CE space | X | nbb=2 | nrb=11 | ncb=9 | BE
CE space | X | 1 | nrb=12 | ncb=9 | BE
CE space | X | nbb=2 | nrb=12 | ncb=9 | BE
CE space | X | 1 | nrb=13 | ncb=9 | BE
CE space | X | nbb=2 | nrb=13 | ncb=9 | BE
Page Register= nrb + nbb
CE space | X | 1 | nrb=11 | ncb=10 | BE
CE space | X | nbb=2 | nrb=11 | ncb=10 | BE
CE space | X | 1 | nrb=12 | ncb=10 | BE
CE space | X | nbb=2 | nrb=12 | ncb=10 | BE
CE space | X | 1 | nrb=13 | ncb=10 | BE
CE space | X | nbb=2 | nrb=13 | ncb=10 | BE
Page Register= nrb + nbb

Note: ncb = number of column address bits; nrb = number of row address bits; nbb = number of bank address bits.

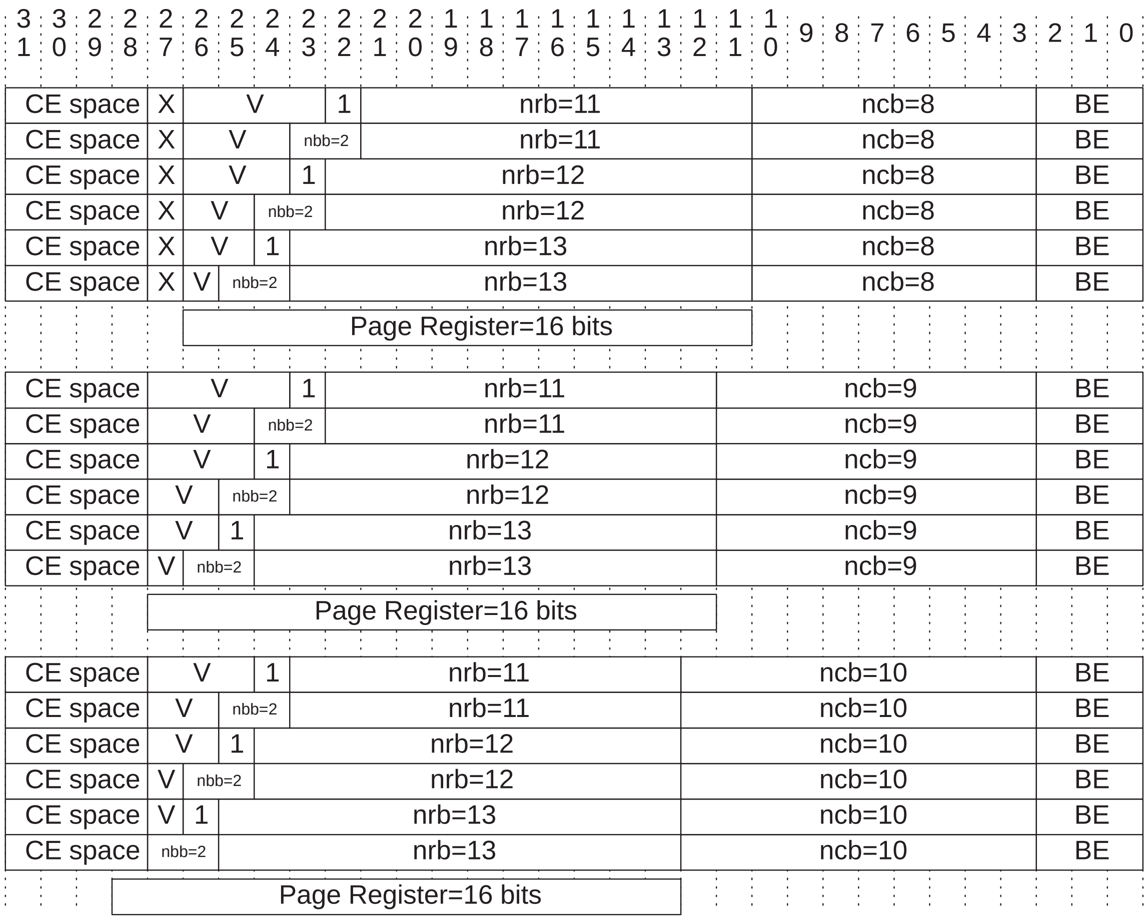
10.5.2.3 TMS320C64x Monitoring Page Boundaries

The C64x is similar to the C621x/C671x SDRAM paging scheme in that up to four pages of SDRAM can be opened simultaneously. This can be all in one CE space, or spread across multiple CE spaces. As in the C621x/C671x, the number of column address bits controls the number of the least significant address bit stored in the page register. However, the page register always stores 16 bits of address (instead of being limited by the number of row address bits plus the number of bank address bits (NRB+NBB)). Therefore, logical address bits above the bank address bit are used as part of the page comparison. Also, address bits above the bank bits are used when issuing the row/column commands to the external SDRAM. This allows more flexible designs and external visibility into the internal address aliasing. For 32–, 16– or 8–bit interfaces on EMIFA, the BE portion of the logical address is reduced to 2 bits for 32–bit SDRAM, 1 bit for 16–bit SDRAM, and 0 bits for 8–bit SDRAM. The NCB/NRB/NBB (and page register) shift accordingly.

The C64x EMIF employs a least recently used (LRU) page replacement strategy when necessary. This occurs when the total number of external SDRAM banks (not devices) is greater than 4, since the EMIF only contains 4 page registers. This can occur when multiple CE spaces of SDRAM are used. When the number of total banks of SDRAM is less than or equal to 4, then the page replacement strategy is fixed since SDRAM requires that only 1 page can be open within a given bank. If a page miss is detected (either during an access where a different page was previously accessed in the same CE space (fixed replacement) or if a page must be closed within a different CE space to allow a page register to be assigned for the current access (LRU replacement), the C64x performs a DEAC command and starts a new row access.

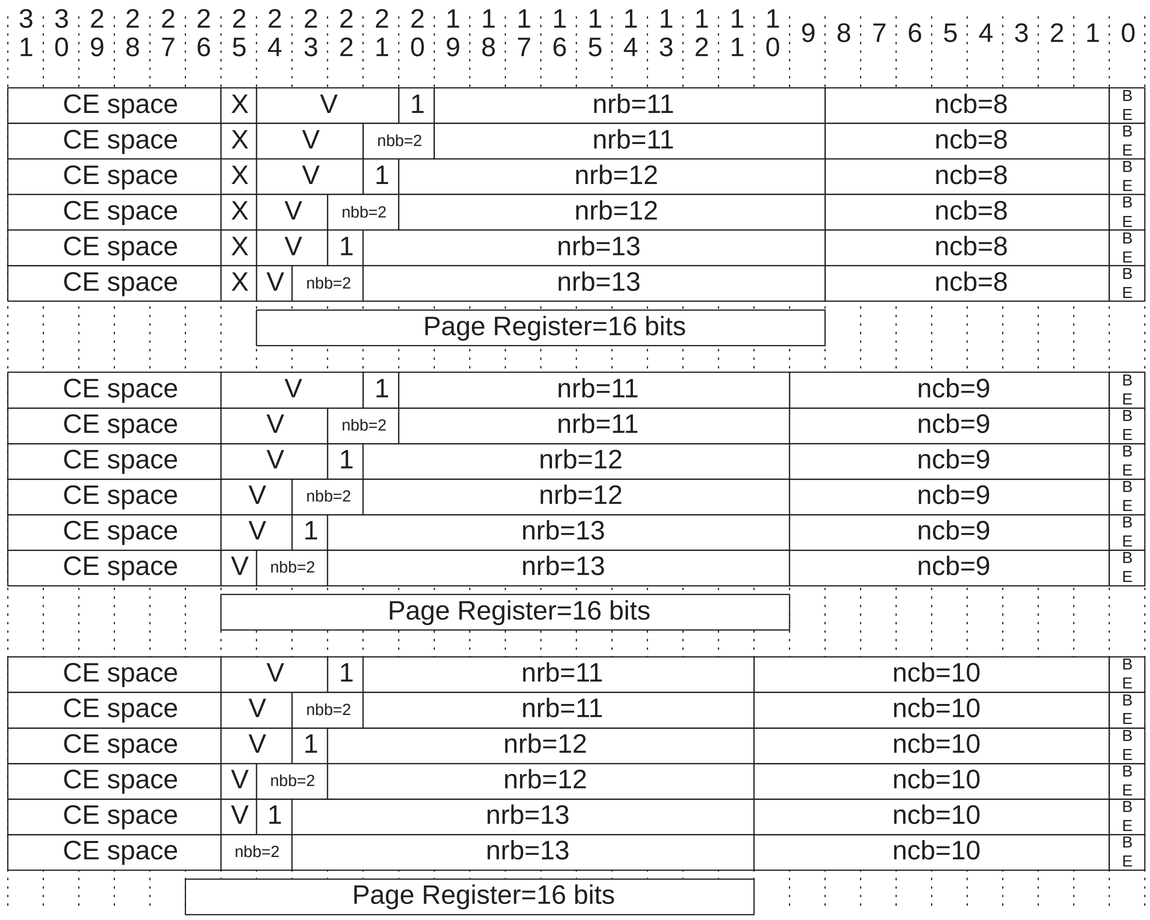
Figure 10–22 details how a 64–bit logical address maps to the page register. Figure 10–23 details how a 16–bit logical address maps to the page register.

Figure 10–22. TMS320C64x Logical Address to Page Register Mapping for EMIFA


31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
CE space | X | V | 1 | nrb=11 | ncb=8 | BE
CE space | X | V | nbb=2 | nrb=11 | ncb=8 | BE
CE space | X | V | 1 | nrb=12 | ncb=8 | BE
CE space | X | V | nbb=2 | nrb=12 | ncb=8 | BE
CE space | X | V | 1 | nrb=13 | ncb=8 | BE
CE space | X | V | nbb=2 | nrb=13 | ncb=8 | BE
Page Register=16 bits
CE space | V | 1 | nrb=11 | ncb=9 | BE
CE space | V | nbb=2 | nrb=11 | ncb=9 | BE
CE space | V | 1 | nrb=12 | ncb=9 | BE
CE space | V | nbb=2 | nrb=12 | ncb=9 | BE
CE space | V | 1 | nrb=13 | ncb=9 | BE
CE space | V | nbb=2 | nrb=13 | ncb=9 | BE
Page Register=16 bits
CE space | V | 1 | nrb=11 | ncb=10 | BE
CE space | V | nbb=2 | nrb=11 | ncb=10 | BE
CE space | V | 1 | nrb=12 | ncb=10 | BE
CE space | V | nbb=2 | nrb=12 | ncb=10 | BE
CE space | V | 1 | nrb=13 | ncb=10 | BE
CE space | nbb=2 | nrb=13 | ncb=10 | BE
Page Register=16 bits

Note: ncb = number of column address bits; nrb = number of row address bits; nbb = number of bank address bits.

Figure 10–23. TMS320C64x Logical Address to Page Register Mapping for EMIFB


31 30 29 28 27 26 25 24 23 22 21 20 19 18 17 16 15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
CE space | X | V | 1 | nrb=11 | ncb=8 | BE
CE space | X | V | nbb=2 | nrb=11 | ncb=8 | BE
CE space | X | V | 1 | nrb=12 | ncb=8 | BE
CE space | X | V | nbb=2 | nrb=12 | ncb=8 | BE
CE space | X | V | 1 | nrb=13 | ncb=8 | BE
CE space | X | V | nbb=2 | nrb=13 | ncb=8 | BE
Page Register=16 bits
CE space | V | 1 | nrb=11 | ncb=9 | BE
CE space | V | nbb=2 | nrb=11 | ncb=9 | BE
CE space | V | 1 | nrb=12 | ncb=9 | BE
CE space | V | nbb=2 | nrb=12 | ncb=9 | BE
CE space | V | 1 | nrb=13 | ncb=9 | BE
CE space | V | nbb=2 | nrb=13 | ncb=9 | BE
Page Register=16 bits
CE space | V | 1 | nrb=11 | ncb=10 | BE
CE space | V | nbb=2 | nrb=11 | ncb=10 | BE
CE space | V | 1 | nrb=12 | ncb=10 | BE
CE space | V | nbb=2 | nrb=12 | ncb=10 | BE
CE space | V | 1 | nrb=13 | ncb=10 | BE
CE space | nbb=2 | nrb=13 | ncb=10 | BE
Page Register=16 bits

Note: ncb = number of column address bits; nrb = number of row address bits; nbb = number of bank address bits.

10.5.3 Address Shift

The same EMIF pins determine the row and column address, thus the EMIF interface appropriately shifts the address in row and column address selection. Table 10–20 and shows the translation between bits of the byte address and how they appear on the EA pins for row and column addresses on the C620x/C670x. SDRAMs use the address inputs for control as well as address.

The following factors apply to the address shifting process for the C620x/C670x:

❑ The address shift is controlled completely by the SDWID field, which is programmed according to the column size of the SDRAM.

- ❑ The upper address bits (EA[14:11] when SDWID=0, or EA[15:10] when SDWID=1) are latched internally by the SDRAM controller during a RAS cycle. This ensures that the SDRAM bank select inputs are 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 during READ or WRT commands. This prevents autoprecharge from occurring following a READ or WRT command.

Table 10–20. TMS320C620x/C670x Byte Address to EA Mapping for SDRAM $\overline{\text{RAS}}$ and $\overline{\text{CAS}}$

EMIF Pins				**EA [21:17]**	**EA 16**	**EA 15**	**EA 14**	**EA 13**	**SDA 10**	**EA 11**	**EA 10**	**EA 9**	**EA 8**	**EA 7**	**EA 6**	**EA 5**	**EA 4**	**EA 3**	**EA 2**
SDRAM Pins	**SDRAM Width**	**SDWID**	**DRAM Cmd**			**A 13**	**A 12**	**A 11**	**A 10**	**A 9**	**A 8**	**A 7**	**A 6**	**A 5**	**A 4**	**A 3**	**A 2**	**A 1**	**A 0**
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	L†	░ 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	L†	░ 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.

† L = Low; SDA10 is driven low during READ or WRT commands to disable autoprecharge.

The following factors apply to the address shifting process for the C621x/C671x:

- ❑ 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
- ❑ The address bits corresponding to the bank select bits are latched internally by the SDRAM controller during a RAS cycle. The bank select bits are EA[13+n:13] for SDRSZ=00b (11 row pins), EA[14+n:14] for SDRSZ=01b (12 row pins), or EA[15+n:15] for SDRSZ=11b (13 row pins); where n=0 when SDBSZ=0, and n=1 when SDBSZ=1. This ensures that the SDRAM bank select inputs are correct during READ and WRT com-

mands. Thus, the EMIF maintains these values as shown in both row and column addresses.

- ❑ EA12 is connected directly to A10 signal, instead of using a dedicated pre-charge pin SDA10.

Table 10–21 describes the addressing for a 8-, 16-, and 32-bit-wide C621x/C671x SDRAM interface. The address presented on the pins are shifted for 8-bit and 16-bit interfaces.

Table 10–21. TMS320C621x/C671x Byte Address-to-EA Mapping for 8-, 16-, and 32-Bit Interface

# of column address bits	Interface bus width	DRAM Cmd	EA [21:17]	EA 16	EA 15	EA 14	EA 13	EA 12	EA 11	EA 10	EA 9	EA 8	EA 7	EA 6	EA 5	EA 4	EA 3	EA 2
				A 14	A 13	A 12	A 11	A 10	A 9	A 8	A 7	A 6	A 5	A 4	A 3	A 2	A 1	A 0
8	8	$\overline{\text{RAS}}$		22	21	20	19	18	17	16	15	14	13	12	11	10	9	8
		$\overline{\text{CAS}}$		Bank‡				L†	L	L	7	6	5	4	3	2	1	0
	16	$\overline{\text{RAS}}$		23	22	21	20	19	18	17	16	15	14	13	12	11	10	9
		$\overline{\text{CAS}}$		Bank‡				L†	L	L	8	7	6	5	4	3	2	1
	32	$\overline{\text{RAS}}$		24	23	22	21	20	19	18	17	16	15	14	13	12	11	10
		$\overline{\text{CAS}}$		Bank‡				L†	L	L	9	8	7	6	5	4	3	2
9	8	$\overline{\text{RAS}}$		23	22	21	20	19	18	17	16	15	14	13	12	11	10	9
		$\overline{\text{CAS}}$		Bank‡				L†	L	8	7	6	5	4	3	2	1	0
	16	$\overline{\text{RAS}}$		24	23	22	21	20	19	18	17	16	15	14	13	12	11	10
		$\overline{\text{CAS}}$		Bank‡				L†	L	9	8	7	6	5	4	3	2	1
	32	$\overline{\text{RAS}}$		25	24	23	22	21	20	19	18	17	16	15	14	13	12	11
		$\overline{\text{CAS}}$		Bank‡				L†	L	10	9	8	7	6	5	4	3	2
10	8	$\overline{\text{RAS}}$		24	23	22	21	20	19	18	17	16	15	14	13	12	11	10
		$\overline{\text{CAS}}$		Bank‡				L†	9	8	7	6	5	4	3	2	1	0
	16	$\overline{\text{RAS}}$		25	24	23	22	21	20	19	18	17	16	15	14	13	12	11
		$\overline{\text{CAS}}$		Bank‡				L†	10	9	8	7	6	5	4	3	2	1
	32	$\overline{\text{RAS}}$		26	25	24	23	22	21	20	19	18	17	16	15	14	13	12
		$\overline{\text{CAS}}$		Bank‡				L†	11	10	9	8	7	6	5	4	3	2

Legend:

░	Bit is internally latched during an ACTV command.
▓	Reserved for future use. Undefined.
▨	Bit may not be driven. The number of address bits driven during a $\overline{\text{RAS}}$ cycle is equal to the number of (row bits + bank-select bits).

† L = Low; EA12 is driven low during READ or WRT commands to disable autoprecharge.

‡ During CAS cycle for READ or WRT command, only the bank select address bits (1 or 2 bits, controlled by SDBSZ) are driven to valid values. The address bit(s) used are determined by the number of row address bits and number of bank address bits.

The following factors apply to the address shifting process for the C64x:

❑ The address shift is controlled completely by the column size field (SDCSZ), and is unaffected by the bank and row size fields. The address bits above the bank select bits are used internally to determine whether a page is opened.
❑ The address bits above the precharge bit (EA[18:14] on EMIFA, and EA[16:12] on EMIFB) are latched internally by the SDRAM controller during a RAS cycle. This ensures that the SDRAM bank select inputs are correct during READ and WRT commands. Thus, the EMIF maintains these values as shown in both row and column addresses.
❑ EA13 is the precharge pin for EMIFA. EA11 is the precharge pin for EMIFB.

Table 10–22 describes the addressing for a 8-, 16-, 32-, and 64-bit-wide SDRAM interface. The 32-bit and 64-bit SDRAM interfaces do not apply to EMIFB.

Table 10–22. TMS320C64x Byte Address to EA Mapping for 8-, 16-, 32-, 64-bit Interface

# of column address bits	Inter-face bus width	DRAM Cmd	EMIFB																			
			EA 20	EA 19	EA 18	EA 17[‡]	EA 16	EA 15	EA 14	EA 13	EA 12	EA 11	EA 10	EA 9	EA 8	EA 7	EA 6	EA 5	EA 4	EA 3	EA 2	EA 1
			EMIFA																			
			EA 22	EA 21	EA 20	EA 19[‡]	EA 18	EA 17	EA 16	EA 15	EA 14	EA 13	EA 12	EA 11	EA 10	EA 9	EA 8	EA 7	EA 6	EA 5	EA 4	EA 3
			A 19	A 18	A 17	A 16	A 15	A 14	A 13	A 12	A 11	A 10	A9	A8	A7	A6	A5	A4	A3	A2	A1	A0
8	8	$\overline{RAS}$	L	L	L	H/L	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8
		$\overline{CAS}$	L	L	L	H/L	23	22	21	20	19	L[†]	L	L	7	6	5	4	3	2	1	0
	16	$\overline{RAS}$	L	L	L	H/L	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9
		$\overline{CAS}$	L	L	L	H/L	24	23	22	21	20	L[†]	L	L	8	7	6	5	4	3	2	1
	32	$\overline{RAS}$	L	L	L	H/L	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10
		$\overline{CAS}$	L	L	L	H/L	25	24	23	22	21	L[†]	L	L	9	8	7	6	5	4	3	2
	64	$\overline{RAS}$	L	L	L	H/L	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11
		$\overline{CAS}$	L	L	L	H/L	26	25	24	23	22	L[†]	L	L	10	9	8	7	6	5	4	3
9	8	$\overline{RAS}$	L	L	L	H/L	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9
		$\overline{CAS}$	L	L	L	H/L	24	23	22	21	20	L[†]	L	8	7	6	5	4	3	2	1	0
	16	$\overline{RAS}$	L	L	L	H/L	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10
		$\overline{CAS}$	L	L	L	H/L	25	24	23	22	21	L[†]	L	9	8	7	6	5	4	3	2	1
	32	$\overline{RAS}$	L	L	L	H/L	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11
		$\overline{CAS}$	L	L	L	H/L	26	25	24	23	22	L[†]	L	10	9	8	7	6	5	4	3	2
	64	$\overline{RAS}$	L	L	L	H/L	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12
		$\overline{CAS}$	L	L	L	H/L	27	26	25	24	23	L[†]	L	11	10	9	8	7	6	5	4	3
10	8	$\overline{RAS}$	L	L	L	H/L	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10
		$\overline{CAS}$	L	L	L	H/L	25	24	23	22	21	L[†]	9	8	7	6	5	4	3	2	1	0
	16	$\overline{RAS}$	L	L	L	H/L	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11
		$\overline{CAS}$	L	L	L	H/L	26	25	24	23	22	L[†]	10	9	8	7	6	5	4	3	2	1
	32	$\overline{RAS}$	L	L	L	H/L	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12
		$\overline{CAS}$	L	L	L	H/L	27	26	25	24	23	L[†]	11	10	9	8	7	6	5	4	3	2
	64	$\overline{RAS}$	L	L	L	H/L	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13
		$\overline{CAS}$	L	L	L	H/L	28	27	26	25	24	L[†]	12	11	10	9	8	7	6	5	4	3

Legend: ▨ Bit is internally latched during an ACTV command.

[†] L = Low; logical address A10 is driven low during READ or WRT commands to disable autoprecharge.

[‡] EA19 (EMIFA) and EA17 (EMIFB) are used during ACTV to indicate non-PDT vs. PDT access. For non-PDT access, this bit is 1. For PDT access, this bit is 0 during ACTV.

10.5.4 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 C620x/C670x, 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 C620x/C670x EMIF SDRAM controller prioritizes SDRAM refresh requests with other data access requests posted to it from the EMIF requesters. The following rules apply:

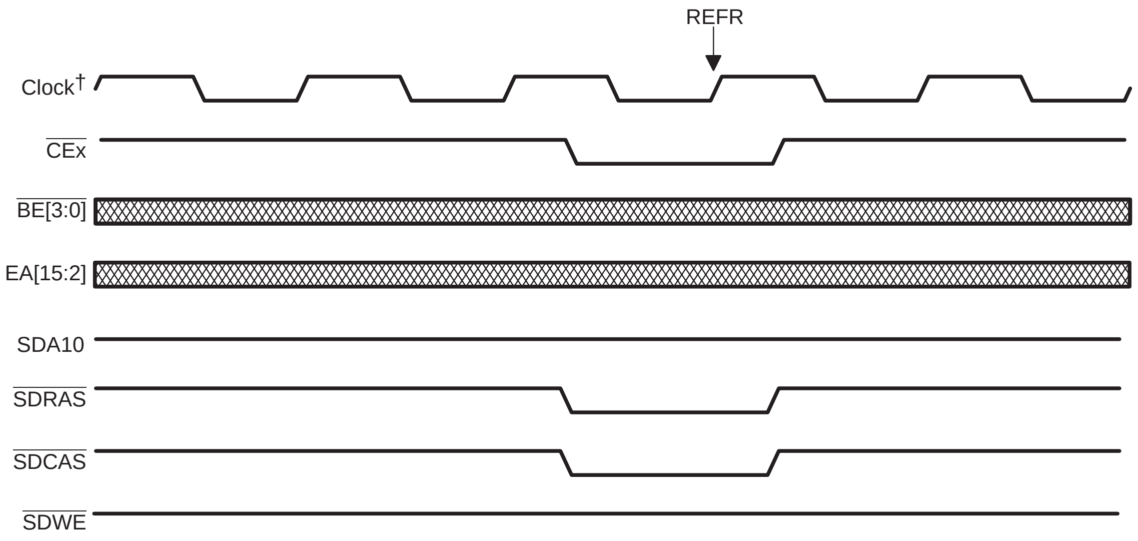
❑ A counter value of 11 invalidates the page information register, forcing the controller to close the current SDRAM page. The value 11 indicates an urgent refresh condition. Thus, following the DCAB command, the EMIF SDRAM controller performs three REFR commands, thereby decrementing the counter to 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 C620x/C670x EMIF, the C621x/C671x/C64x REFR requests are considered high priority, and no distinction exists between urgent and trickle refresh. Transfers in progress are allowed to complete. The C621x/C671x/C64x 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 10–24 shows the timing diagram for an SDRAM refresh.

Figure 10–24. SDRAM Refresh


REFR
Clock†
$\overline{\text{CEx}}$
$\overline{\text{BE[3:0]}}$
EA[15:2]
SDA10
$\overline{\text{SDRAS}}$
$\overline{\text{SDCAS}}$
$\overline{\text{SDWE}}$

† Clock = SDCLK for C6201/C6701.
= CLKOUT2 for all C620x/C670x except C6201/C6701.
= ECLKOUT for C621x/C671x.
= ECLKOUT1 for C64x.

10.5.5 SDRAM Self Refresh Mode (C64x Only)

The SLFRFR bit of the SDRAM control register (SDCTL) forces the EMIF to place the external SDRAM in a low-power mode, called Self Refresh, in which the SDRAM maintains valid data while consuming a minimal amount of power. This mode is entered when a 1 is written to the SLFRFR bit and SDRAM exists in the system. When the SLFRFR bit is set, the refresh enable bit RFEN in the SDCTL must be written with a 0 simultaneously. When the SLFRFR bit is asserted, all open pages of SDRAM are closed (DCAB issued to all CE spaces). In addition, a REFRESH command is issued on the same cycle that the SDCKE signal is driven low.

It is the user's responsibility to ensure that the SLFRFR bit is turned on/off at appropriate times. To exit SLFRFR mode, write a 0 to the SLFRFR bit and then immediately read back before performing other accesses. As long as SLFRFR = 1, user should ensure that no SDRAM accesses are performed.

During self refresh mode, the SDRAM clock (ECLKOUT1) can be turned off, if the system does not use the Hold interface, or if ECLKOUT1 is not used elsewhere in the system. ECLKOUT1 must be re-enabled before exiting self refresh mode. The EMIF ensures that the SDRAM is in the self refresh state for at least TRAS cycles, where TRAS is defined in the SDEXT register. In addition, the EMIF ensures the time from SDCKE high to the next ACTV command is at least 16 ECLKOUT1 cycles.

If SDRAM is not in use in the system, the SDCKE pin can be used as a general-purpose output. The inverse of SLFRFR bit is driven on the SDCKE pin.

If a Hold request is detected, then before acknowledging this request with HOLDA, the EMIF will assert the SDCKE output (as long as TRAS requirement has been met) and clear the SLFRFR bit to wake the SDRAM from reset. If SDRAM is not in use by the system, then Hold will have no effect on the state of the SDCKE output or the SLFRFR field.

The effects of the SLFRFR bit with an SDRAM in the system are summarized here:

- ❑ Write to SLFRFR while not in hold causes Self refresh mode entry/exit.
- ❑ Write to SLFRFR while in hold: write to SLFRFR is ignored, bit is not written.
- ❑ If HOLD request occurs while SLFRFR = 1, the EMIF ensures that the device has been in self refresh mode at least TRAS cycles. Then the EMIF exits self refresh mode (deasserts SLFRFR). After 16 ECLKOUT1 cycles, the EMIF will acknowledge the HOLD request.

Note: The EMIF SDCKE signal must be connected to the SDRAM CKE signal for proper SLFRFR operation.

10.5.6 Mode Register Set

The C620x/C670x 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 C620x/C670x EMIF always uses a mode register value of 0030h during an MRS command. Figure 10–25 shows the mapping between mode register bits, EMIF pins, and the mode register value. Table 10–23 shows the JEDEC standard SDRAM configuration values selected by this mode register value. Figure 10–27 shows the timing diagram during execution of the MRS command.

Figure 10–25. TMS320C620x/C670x 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		
011			0	000		

Table 10–23. TMS320C620x/C670x 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 C621x/C671x/C64x uses a mode register value of either 0032h or 0022h. The register value and description are shown in Figure 10–26 and summarized in Table 10–24. 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.

Figure 10–26. TMS320C621x/C671x/C64x 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		
010 or 011[‡]			0	010		

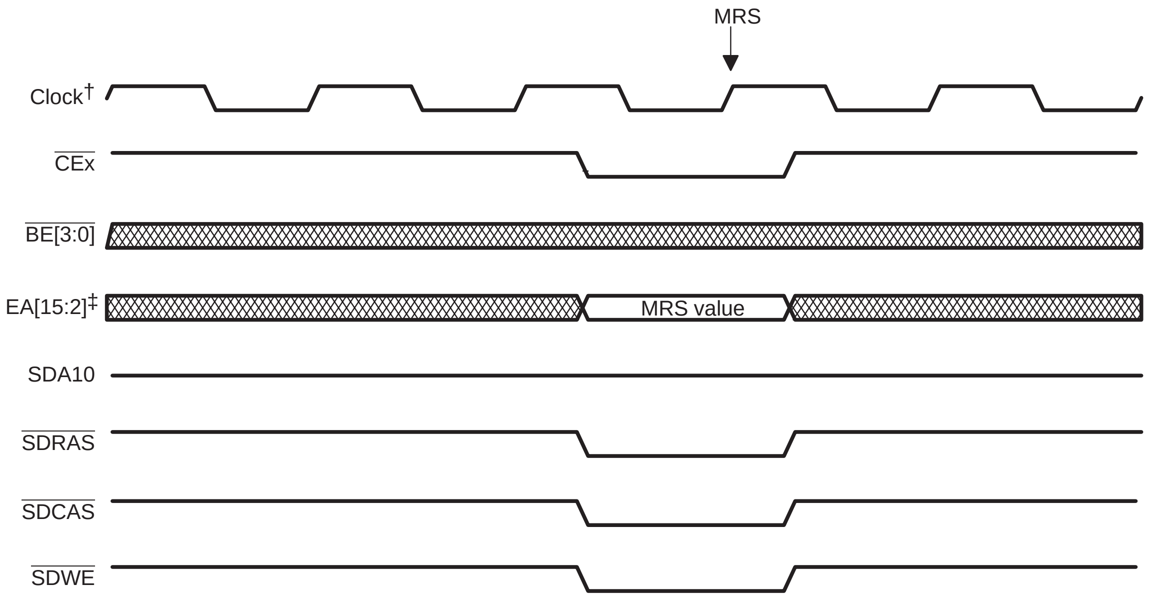
[†] For C64x EMIFA, EA[16:3] is used instead.
For C64x EMIFB, EA[14:1] is used instead.

[‡] If CASL = 1, bit 4 is 1.
If CASL = 0, bit 4 is 0.

Table 10–24. TMS320C621x/C671x/C64x 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 10–27. SDRAM Mode Register Set: MRS Command



† Clock = SDCLK for C6201/C6701.
= CLKOUT2 for all C620x/C670x except C6201/C6701.
= ECLKOUT for C621x/C671x.
= ECLKOUT1 for C64x.

‡ For C64x EMIFA, EA[16:3] is used instead.
For C64x EMIFB, EA[14:1] is used instead.

10.5.7 Timing Requirements

Several SDRAM timing parameters decouple the EMIF from SDRAM speed limitations. For the C620x/C670x, three of these parameters are programmable via the EMIF SDRAM control register; the remaining two are assumed to be static values, as shown in Table 10–25. 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 10–25. TMS320C620x/C670x SDRAM Timing Parameters

Parameter	Description	Value in EMIF Clock 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

[†] EMIF clock cycles = CLKOUT2 cycles for C620x/C670x

The C621x/C671x/C64x has additional programmable timing parameters that are programmable via the SDRAM control register and the SDRAM extension register, as shown in Table 10–26. These parameters are a superset of the parameters of the C620x/C670x. Consult the manufacturer’s data sheet for the particular SDRAM.

The C621x/C671x/C64x also allows the user to program other functional parameters of the SDRAM controller. These parameters are listed in Table 10–9 under the SDRAM extension register field descriptions. These parameters are not explicitly spelled out in the timing parameters of a data sheet, but the user must ensure that the parameters are programmed to a valid value. The recommended values for these SDRAM parameters are shown in Table 10–27.

Table 10–26. TMS320C621x/C671x/C64x SDRAM Timing Parameters

Parameter	Description	Value in EMIF Clock Cycles[†]
tRC	REFR command to ACTV, MRS, or subsequent REFR command	TRC + 1
tRCD	ACTV command to READ or WRT command	TRCD + 1
tRP	DCAB/DEAC command to ACTV, MRS, or REFR command	TRP + 1
tCL	CAS latency of the SDRAM	TCL + 2
tRAS	ACTV command to DEAC/DCAB command	TRAS + 1
tRRD	ACTV bank A to ACTV bank B (same CE space)	TRRD + 2
tWR	Write recovery, time from last data out of C6000 (write data) to DEAC/DCAB command	TWR + 1
tHZP	High Z from precharge, time from DEAC/DCAB to SDRAM outputs (read data) in high Z	THZP + 1

[†] EMIF clock cycles = ECLKOUT cycles for C621x/C671x
= ECLKOUT1 cycles for C64x.

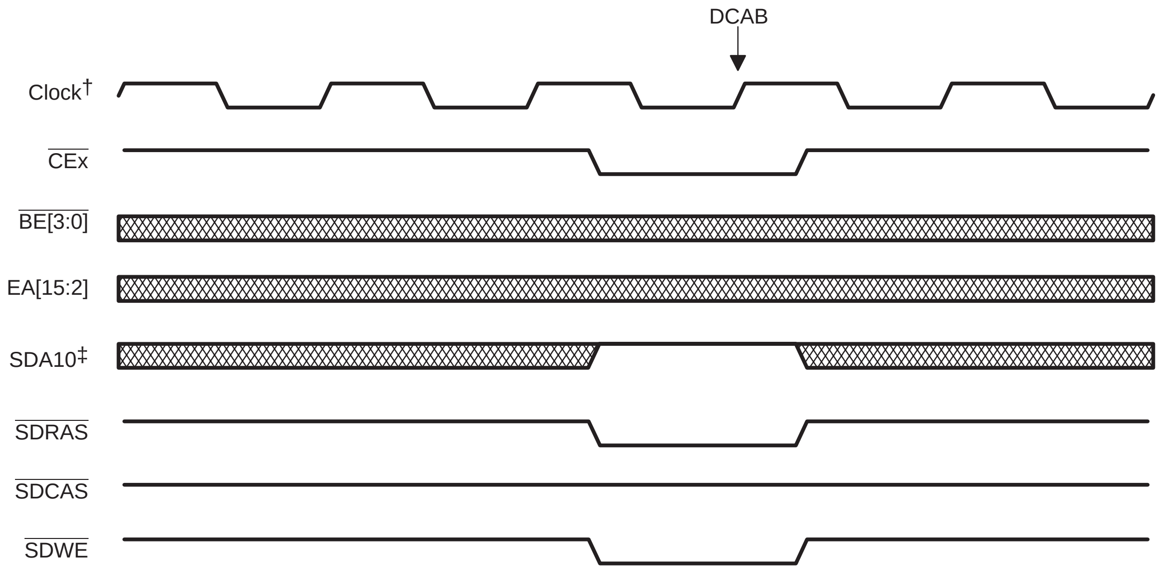
Table 10–27. TMS320C621x/C671x/C64x Recommended Values for Command–to–Command Parameters

Parameter	Description	Value in EMIF clock cycles	Suggested value for CL=2	Suggested value for CL=3
READ to READ	READ command to READ command. Used to interrupt a READ burst for random READ addresses	RD2RD + 1	RD2RD = 0	RD2RD = 0
READ to DEAC	Used in conjunction with t_{HZP}. Specifies the minimum amount of time between READ command and DEAC/DCAB command	RD2DEAC + 1	RD2DEAC = 1	RD2DEAC = 1
READ to WRITE	READ to WRITE command. The value programmed in this parameter depends on t_{CL}. READ to WRITE should be CAS latency plus 2 cycles (in EMIF clock cycles) to provide 1 turnaround cycle before WRITE command.	RD2WR + 1	RD2WR = 3	RD2WR = 4
BEx high before write interrupting read	Specifies the number of cycles that the BEx outputs should be high before a write is allowed to interrupt a read. This is related to READ to WRITE parameter.	R2WDQM + 1	R2WDQM = 1	R2WDQM = 2
WRITE to WRITE	Number of cycles between a WRITE interrupting a WRITE. Used for random WRITEs.	WR2WR + 1	WR2WR = 0	WR2WR = 0
WRITE to DEAC	Number of cycles between a WRITE command and a DEAC/DCAB command	WR2DEAC + 1	WR2DEAC = 1	WR2DEAC = 1
WRITE to READ	Number of cycles between a WRITE command and a READ command	WR2RD + 1	WR2RD = 0	WR2RD = 0

10.5.8 SDRAM Deactivation (DCAB and DEAC)

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 C620x/C670x, 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 10–28 shows the timing diagram for SDRAM deactivation.

Figure 10–28. TMS320C6000 SDRAM DCAB — Deactivate all Banks

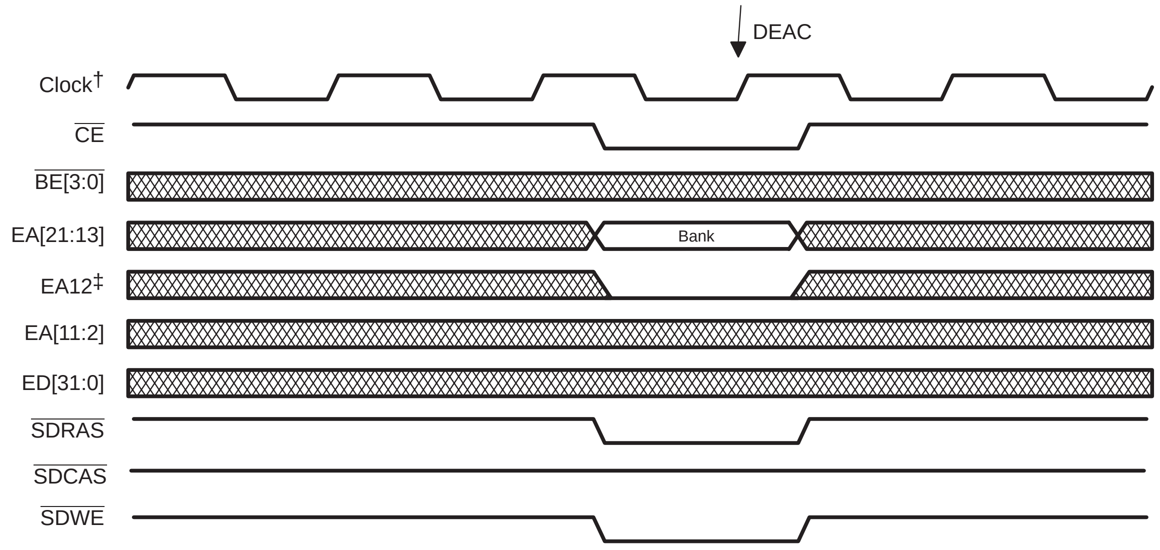


† Clock = SDCLK for C6201/C6701.
= CLKOUT2 for all C620x/C670x except C6201/C6701.
= ECLKOUT for C621x/C671x.
= ECLKOUT1 for C64x

‡ SDA10 applies to C620x/C670x only. On C621x/C671x, EA12 is used. On C64x EMIFA, EA13 is used. On C64x EMIFB, EA11 is used.

The C621x/C671x/C64x also supports the DEAC command, whose operation is depicted in Figure 10–29, 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 C621x/C671x/C64x still supports the DCAB command to close all pages prior to REFR and MRS commands.

Figure 10–29. TMS320C621x/C671x SDRAM DEAC — Deactivate Single Bank

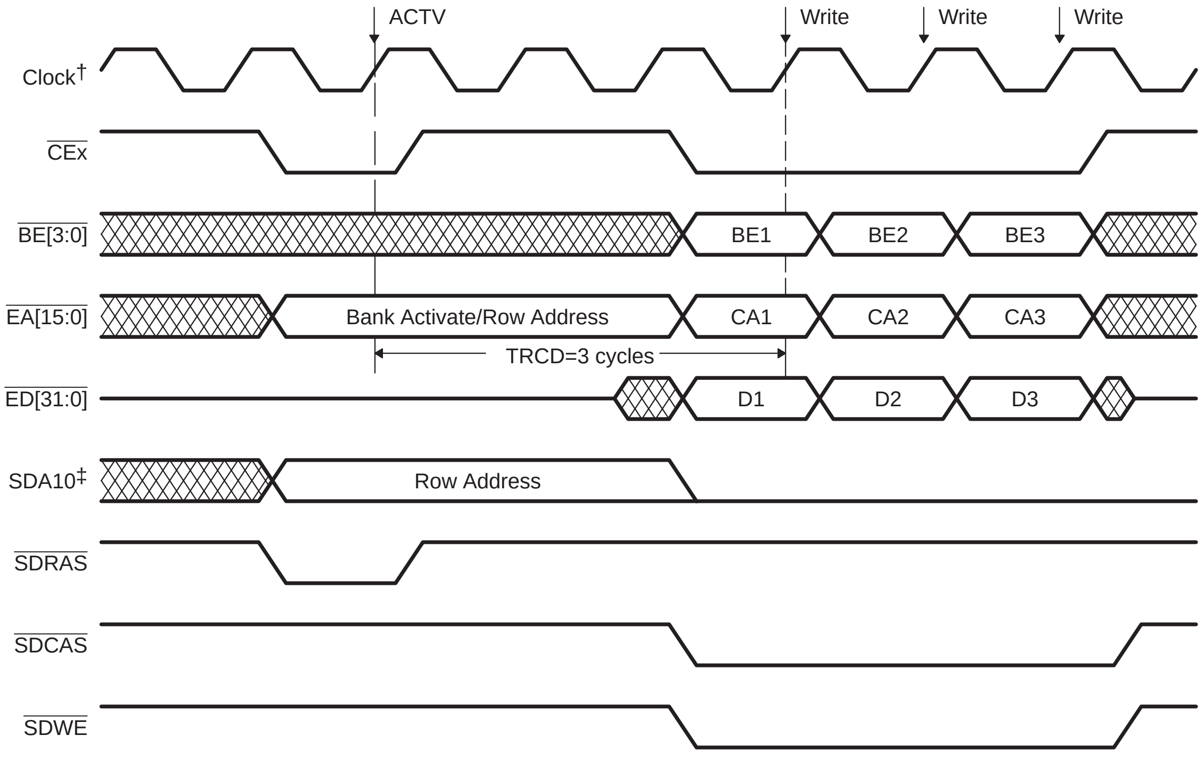


† Clock = ECLKOUT for C621x/C671x.
= ECLKOUT1 for C64x
‡ For C64x EMIFA, EA13 is used. For C64x EMIFB, EA11 is used.

10.5.9 Activate (ACTV)

The EMIF automatically issues the Activate (ACTV) command before a read or write to a new row of SDRAM. The ACTV command opens up a page of memory, allowing future accesses (reads or writes) with minimum latency. When an ACTV command is issued by the EMIF, a delay of t_{RCD} is incurred before a read or write command is issued. Figure 10–30 shows an example of an ACTV command before an SDRAM write. In this example, t_{RCD} = 3 EMIF clock cycles. The ACTV command for SDRAM reads are identical. Reads or writes to the currently active row and bank of SDRAM can achieve much higher throughput than reads or writes to random areas, because every time a new page is accessed the ACTV command must be issued.

Figure 10–30. ACTV Command Before an Example SDRAM Write



† Clock = SDCLK for C6201/C6701
= CLKOUT2 for all C620x/C670x except C6201/C6701
= ECLKOUT for C621x/C671x
= ECLKOUT1 for C64x

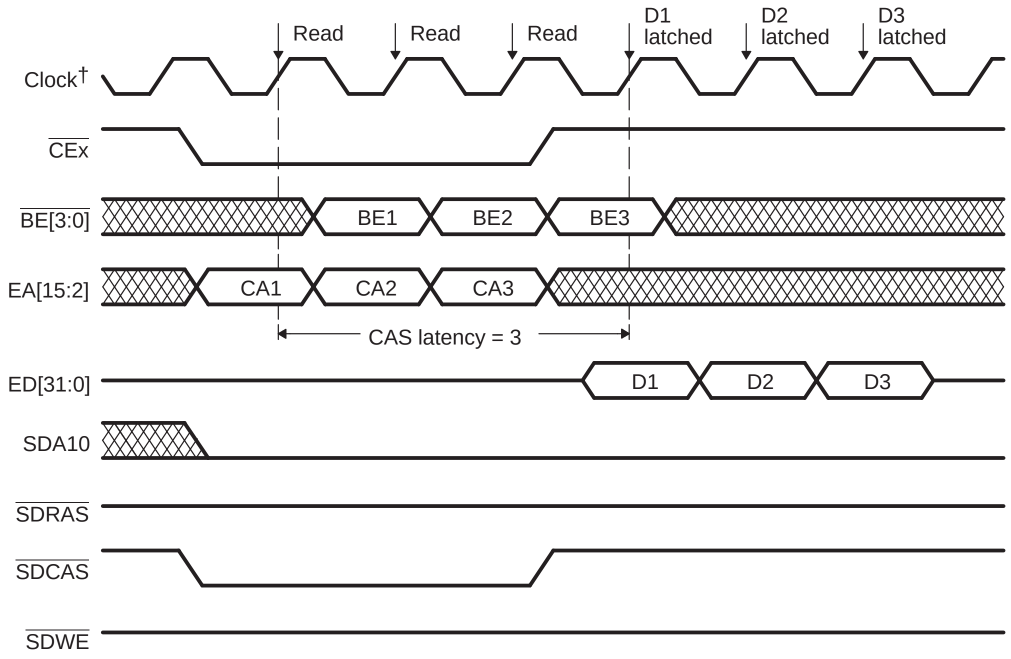
‡ SDA10 applies to C620x/C670x only. On C621x/C671x, EA12 is used. On C64x EMIFA, EA13 is used. On C64x EMIFB, EA11 is used.

10.5.10 SDRAM Read

10.5.10.1 TMS320C620x/C670x SDRAM Read

During an SDRAM read, the selected bank is activated with the row address during the ACTV command. Figure 10–31 shows the timing for the C620x/C670x 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 C620x/C670x, 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 invalid. The values on EA[15:13] during column accesses and execution of the DCAB command are the values latched during the ACTV command.

Figure 10–31. TMS320C620x/C670x SDRAM Read

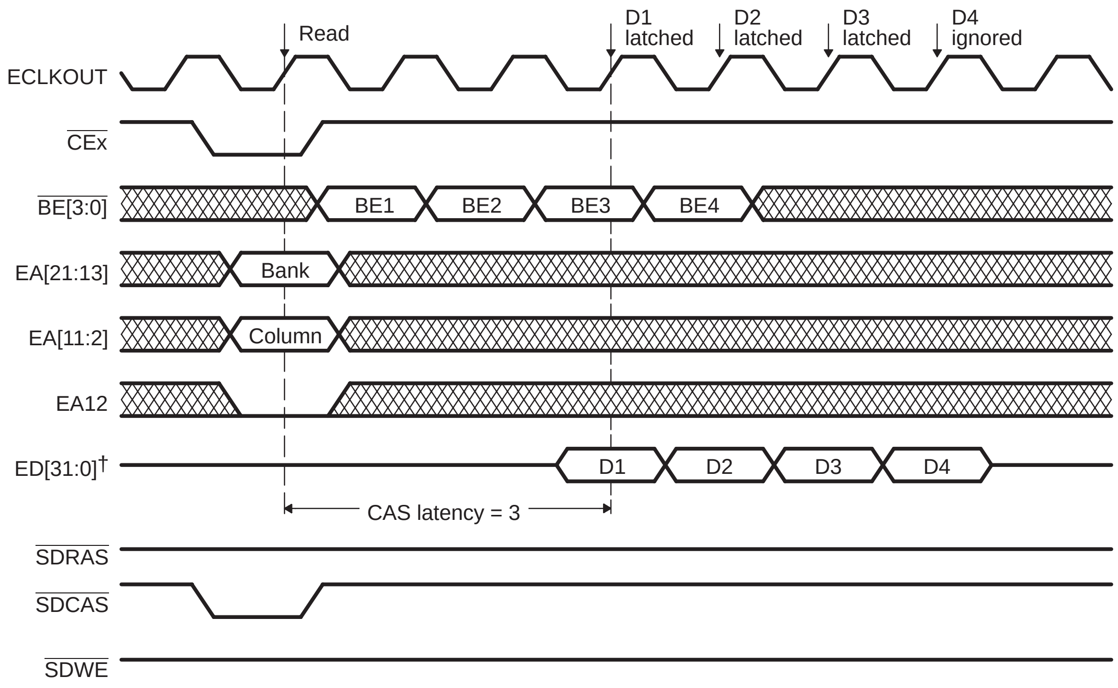


† Clock = SDCLK for C6201/C6701.
= CLKOUT2 for all C620x/C670x except C6201/C6701

10.5.10.2 TMS320C621x/C671x SDRAM Read

Figure 10–32 shows the C621x/C671x performing a three word read burst from SDRAM. The C621x/C671x 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 10–32, 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 10–32. TMS320C621x/C671x SDRAM Read



† ED[31:16] do not apply to C6712.

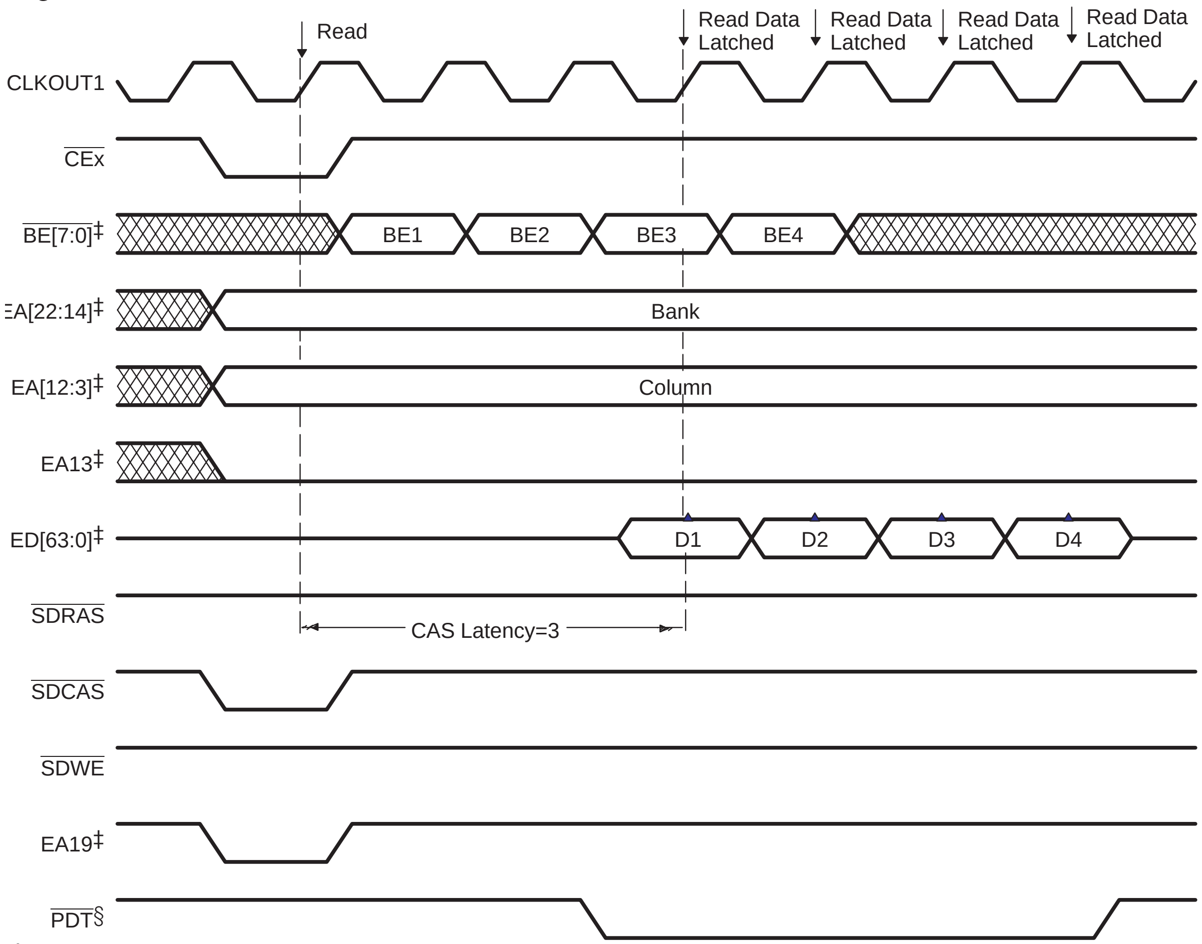
10.5.10.3 TMS320C64x SDRAM Read

The C64x also uses a burst length of four. It differs from the C621x/C671x SDRAM Read as follows:

- During ACTV, EA19 (EMIFA) or EA17 (EMIFB) is driven high for a non-PDT access, and driven low for a PDT access.
- For a PDT access, $\overline{\text{PDT}}$ is asserted during the data phase of the SDRAM read.

Figure 10–33 shows the C64x performing a three doubleword (EMIFA) or half-word (EMIFB) read burst from SDRAM for a PDT access. During ACTV (not shown in Figure 10–33), $\overline{\text{SDRAS}}$ goes active low, and EA19 (EMIFA) or EA17 (EMIFB) is driven low for a PDT access. The ACTV command is not issued for future access to the same open page.

Figure 10–33. TMS320C64x SDRAM Read



† EA19 is used as the PDT address pin for EMIFA. EA17 is used as the PDT address pin for EMIFB.
‡ For EMIFB, $\overline{\text{BE}}$[1:0], EA[20:12], EA[10:1], EA11, and ED[15:0], respectively, are used instead.
§ $\overline{\text{PDT}}$ only goes active during $\overline{\text{PDT}}$ transfers (section 10.9). During normal read/write transactions, the $\overline{\text{PDT}}$ signal will not be asserted.

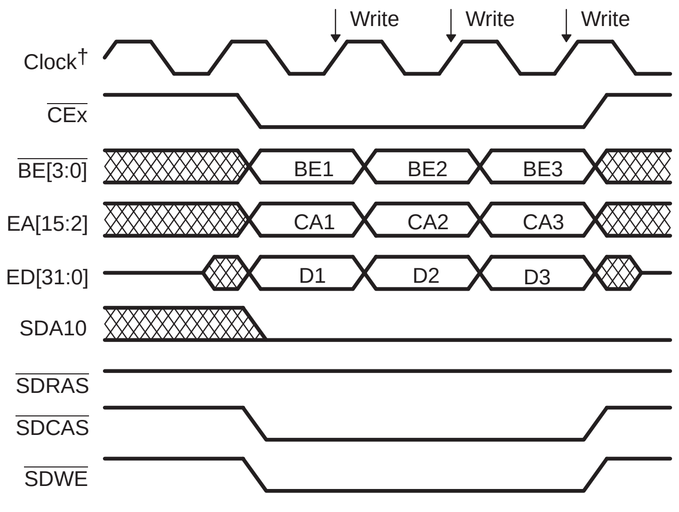
10.5.11 SDRAM Write

10.5.11.1TMS320C620x/C670x SDRAM Write

All SDRAM writes have a burst length of one on the C620x/C670x. 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 10–34 shows the timing for a three-word write on the C620x/C670x. 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 C620x/C670x 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 10.5.2). The values on EA[15:13] (if SDWID=1) or EA[14:13] (if SDWID=0) during column accesses and the DCAB command are the values latched during the ACTV command.

If a page boundary is crossed during the course of an access, the EMIF performs a DCAB command and starts a new row access. If a write burst crosses a page boundary, the $\overline{\text{CAS}}$ and $\overline{\text{WE}}$ signals stay active for one additional cycle before the DCAB command. The $\overline{\text{BEx}}$ signals are inactive high during this additional cycle to prevent the EMIF from incorrectly writing an extra word.

Figure 10–34. TMS320C620x/C670x SDRAM Three-Word Write

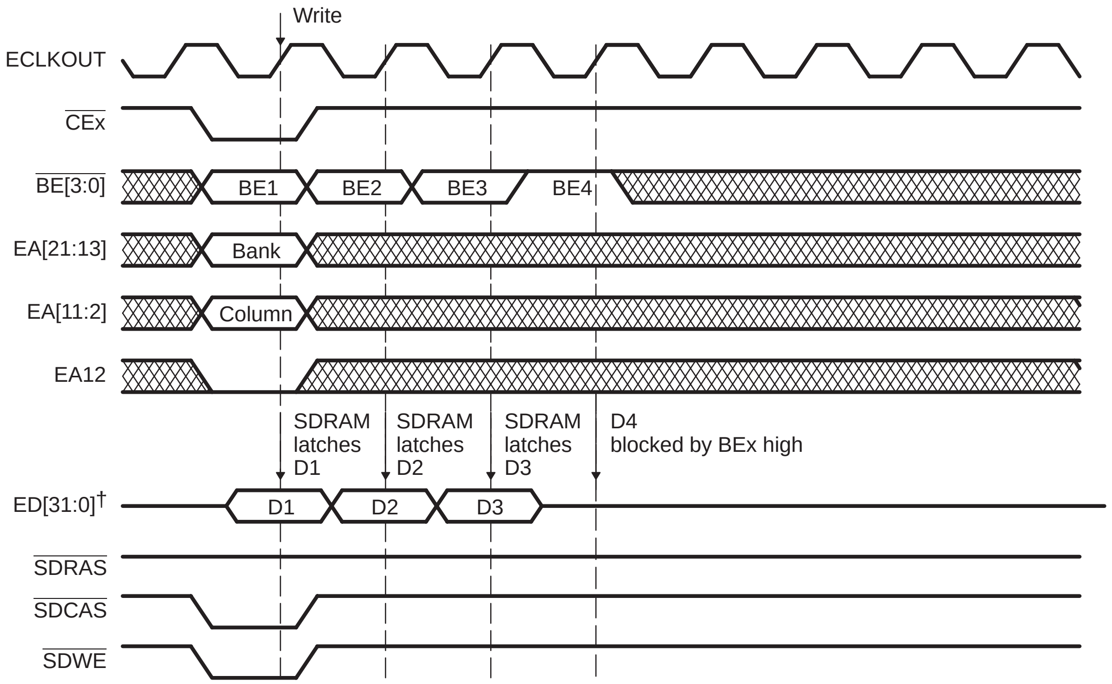


† Clock = SDCLK for C6201/C6701.
= CLKOUT2 for all other C620x/C670x except C6201/C6701.

10.5.11.2 TMS320C621x/C671x SDRAM Write

All SDRAM writes have a burst length of four on the C621x/C671x. 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 10–35 shows the timing for a three-word write on the C621x/C671x. Since the default C621x/C671x write-burst length is four words, the last write is masked out via the byte enable signals. On the C621x/C671x, 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 C621x/C671x, 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 10.5.2). The values on the bank select bits (see section 10.5.3) during column accesses and during the DEAC command are the values latched during the ACTV command.

Figure 10–35. TMS320C621x/C671x SDRAM Three-Word Write



† ED[31:16] do not apply to C6712.

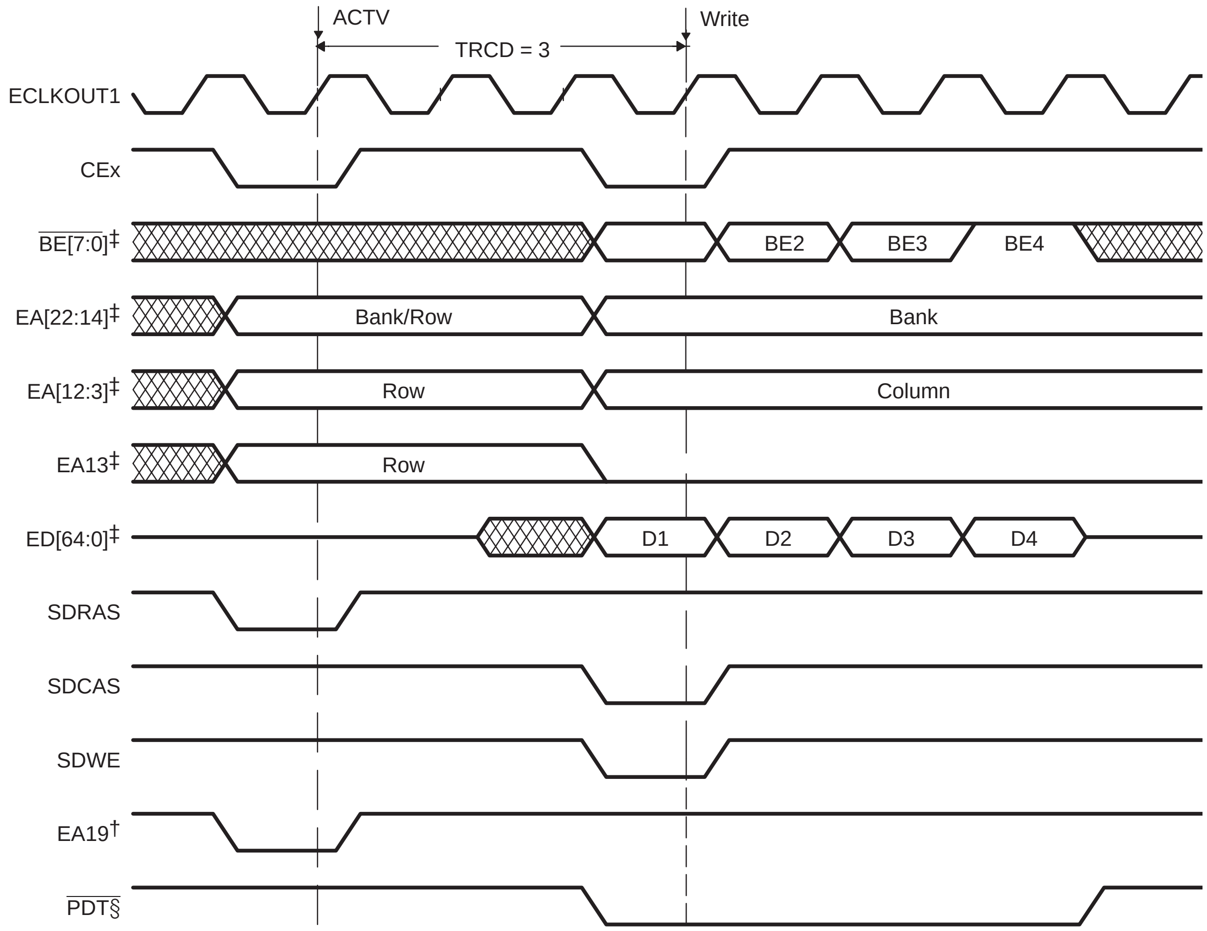
10.5.11.3TMS320C64x SDRAM Write

The C64x also uses a burst length of four. It differs from the C621x/C671x SDRAM read as follows:

❑ During ACTV, EA19 (EMIFA) or EA17 (EMIFB) is driven high for a non-PDT access, and driven low for a PDT access.
❑ For a PDT access, $\overline{\text{PDT}}$ is asserted during the data phase of the SDRAM write.
❑ Write data is driven one cycle early.

Figure 10–36 shows the timing for a three doubleword (EMIFA) or halfword (EMIFB) write on the C64x for a PDT access. An ACTV command is not issued for future access to the same open page.

Figure 10–36. TMS320C64x SDRAM Write



† EA19 is used as the PDT address pin for EMIFA. EA17 is used as the PDT address pin for EMIFB.
‡ For EMIFB, $\overline{\text{BE[1:0]}}$, EA[20:12], EA[10:1], EA11, and ED[15:0], respectively, are used instead.
§ $\overline{\text{PDT}}$ only goes active during $\overline{\text{PDT}}$ transfers (section 10.9). During normal read/write transactions, the $\overline{\text{PDT}}$ signal will not be asserted.

10.6 SBSRAM Interface

The C6000 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. Besides supporting the SBSRAM interface, the programmable synchronous interface on the C64x supports additional synchronous device interfaces. See section 10.8 for details on the C64x interface with the other synchronous devices. This section discusses the SBSRAM interface on all the C6000 devices.

For the C6201/C6701, the SBSRAM interface can run at either the CPU clock speed or at 1/2 of this rate. The selection is made based on the setting of the SSCRT bit in the EMIF global control register. For all other C620x/C670x devices (except C6201/C6701), the interface operates at 1/2 rate only. For the C621x/C671x, the SBSRAM runs off of ECLKOUT. For the C64x, the SBSRAM runs off of ECLKOUT1 or ECLKOUT2.

The four SBSRAM control pins are latched by the SBSRAM on the rising EMIF clock edge to determine the current operation. These pins are listed in Table 10–28. These signals are valid only if the chip select line for the SBSRAM is low.

Table 10–28. 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

† For C64x, SBSRAM control signals are renamed as $\overline{\text{SADS}}/\overline{\text{SRE}}$, $\overline{\text{SOE}}$, and $\overline{\text{SWE}}$, respectively.
‡ For C64x, SBSRAM interface can run off of either ECLKOUT1 or ECLKOUT2.

For the C620x/C670x, 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 C621x/C671x 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 C621x/C671x is assumed to be a four word burst. If non–incrementing addressing is required for a given access, the C621x/C671x 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 C6000 devices. Table 10–29 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 C621x/C671x strobes a new address into the SBSRAM. This is also done if the burst order should be non-incrementing or reverse order burst. After performing a read or write command, the C621x/C671x EMIF issues a deselect command to the SBSRAM if no accesses are pending to that CE space.

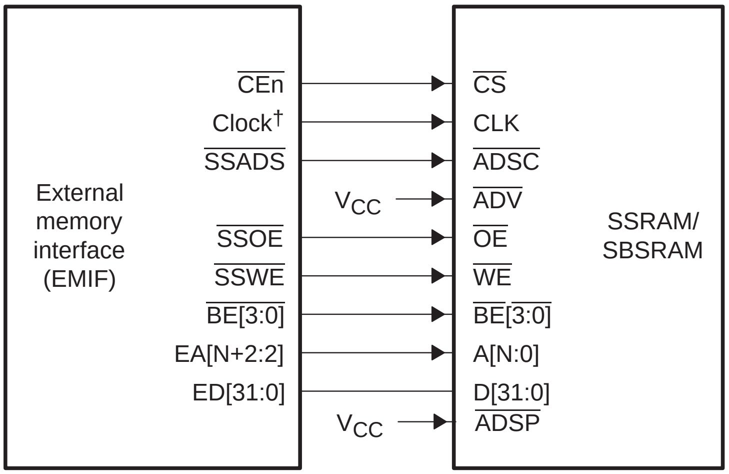
The C64x interface does not support the burst feature of the SBSRAM. On the C64x, an address is strobed into the SBSRAM on every cycle. After performing a read or write command, the C64x EMIF issues a deselect command to the SBSRAM if no accesses are pending to that CE space. The C64x also supports programmable read and write latency to allow a flexible interface to different types of synchronous memories.

The SBSRAM interfaces on the C620x/C670x are shown in Figure 10–37. The SBSRAM interfaces on the C621x/C671x are shown in Figure 10–38. The SBSRAM interfaces on the C64x are shown in Figure 10–39.

Table 10–29. TMS320C621x/C671x SBSRAM in Linear Burst Mode

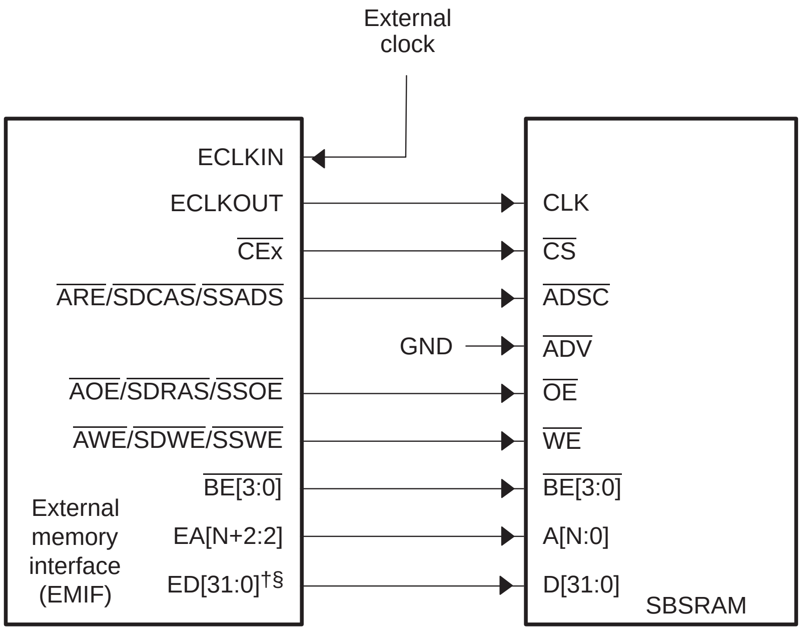
	Case 1	**Case 2**	**Case 3**	**Case 4**
SBSRAM Address	A[1:0]	A[1:0]	A[1:0]	A[1:0]
EMIF Address	EA[3:2]	EA[3:2]	EA[3:2]	EA[3:2]
First address	00	01	10	11
	01	10	11	00
	10	11	00	01
Fourth Address	11	00	01	10

Figure 10–37. TMS320C620x/C670x SBSRAM Interface



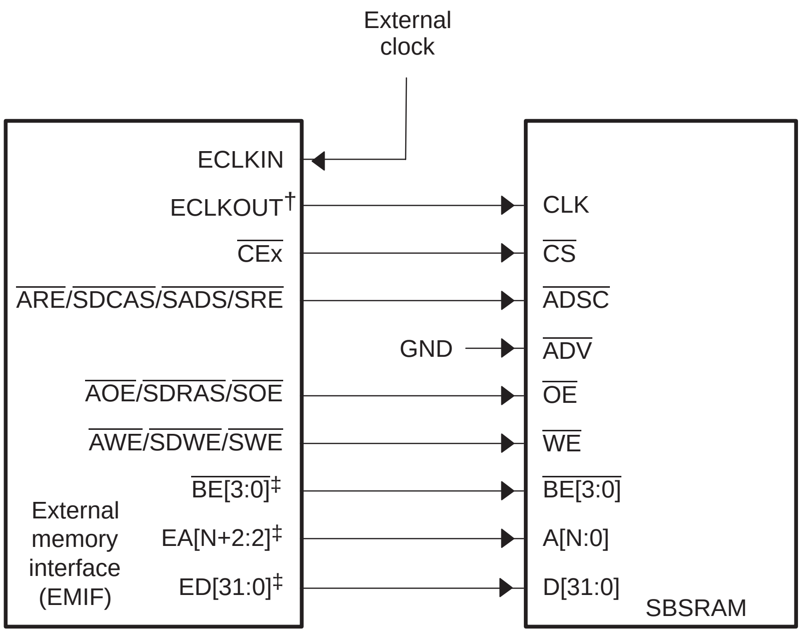
[†] Clock = SSCLK for C6201/C6701.
= CLKOUT2 for all C620x/C670x except C6201/C6701.

Figure 10–38. TMS320C621x/C671x SBSRAM Interface



[†] ED[31:16] do not apply to C6712.

Figure 10–39. TMS320C64x SBSRAM Interface



† ECLKOUTx used is selected by the SNCCLK bit in the CExSEC register.
‡ For interface to a 64–bit data bus, BE[7:0], EA[N+3:3, and ED[63:0] are used.
For interface to a 16–bit data bus, BE[1:0], EA[N+1:1], and ED[15:0] are used.

SBSRAMs are latent by their architecture, since read data follows address and control information by two cycles. 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.

Table 10–30 provides an overview of similarities and differences on the C6000 SBSRAM interface.

Table 10–30. TMS320C6000 SBSRAM Interface Summary

	C62x/C67x			C64x	
	C6201/C6701	Other C620x/C670x[†]	C621x/C671x	EMIFA	EMIFB
Interface width	32-bit	32-bit	32-, 16-, 8-bit	64-, 32-, 16-, 8-bit	16-, 8-bit
SBSRAM clock	SSCLK (1/2x or 1x CPU rate)	CLKOUT2	ECLKOUT	ECLKOUT1 or ECLKOUT2[‡]	ECLKOUT1 or ECLKOUT2[‡]
SBSRAM control signals	Dedicated SDRAM control signals	MUXed with SDRAM control signals.	MUXed with SDRAM and Async control signals.	MUXed with SDRAM and Async control signals.	MUXed with SDRAM and Async control signals.
Burst Mode	Not supported. Performs bursts by issuing back-to-back commands	Not supported. Performs bursts by issuing back-to-back commands	Supports SBSRAM burst mode with a 4-word burst	Not supported. Performs bursts by issuing back–to–back commands. Still issues deselect command	Not supported. Performs bursts by issuing back–to–back commands. Still issues deselect command
Programmable latency	No	No	No	Read, Write	Read, Write

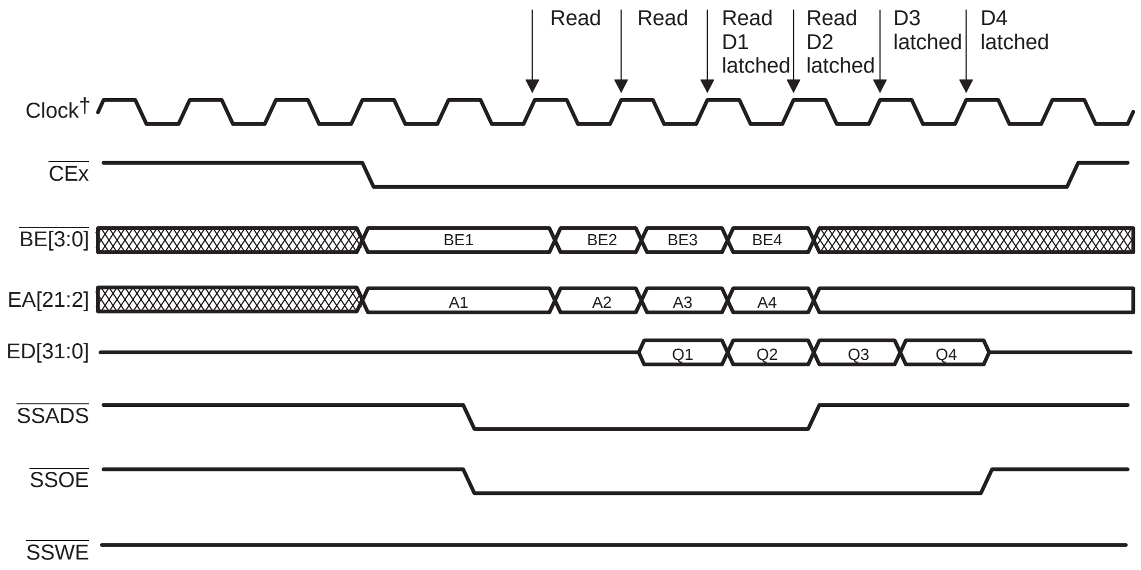
† This column applies to all C620x/C670x devices except C6201/C6701.
‡ The ECLKOUTx used is selected by the SNCCLK bit in the CExSEC register.

10.6.1 SBSRAM Reads

10.6.1.1 C620x/C670x SBSRAM Reads

Figure 10–40 shows a four-word read of an SBSRAM for the C620x/ C670x. 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 EMIF clock cycle.

Figure 10–40. TMS320C620x/C670x SBSRAM Four-Word Read

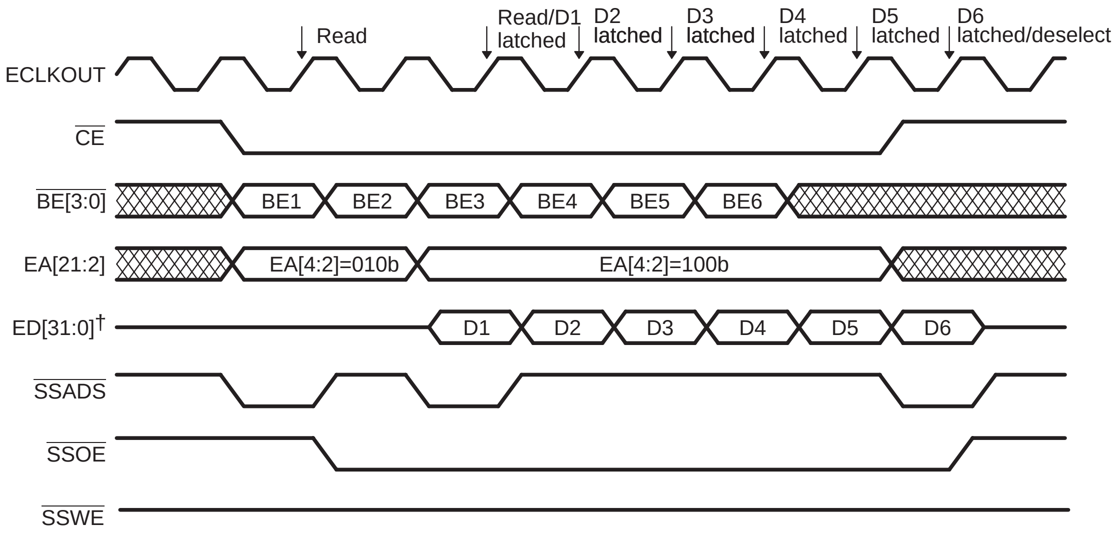


[†] Clock = SSCLK for C6201/C6701.
= CLKOUT2 for all C620x/C670x except C6201/C6701.

10.6.1.2 C621x/C671x SBSRAM Read

Figure 10–41 shows the timing for C621x/C671x 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 10–41. TMS320C621x/C671x SBSRAM Six-Word Read



† ED[31:16] do not apply to C6712.

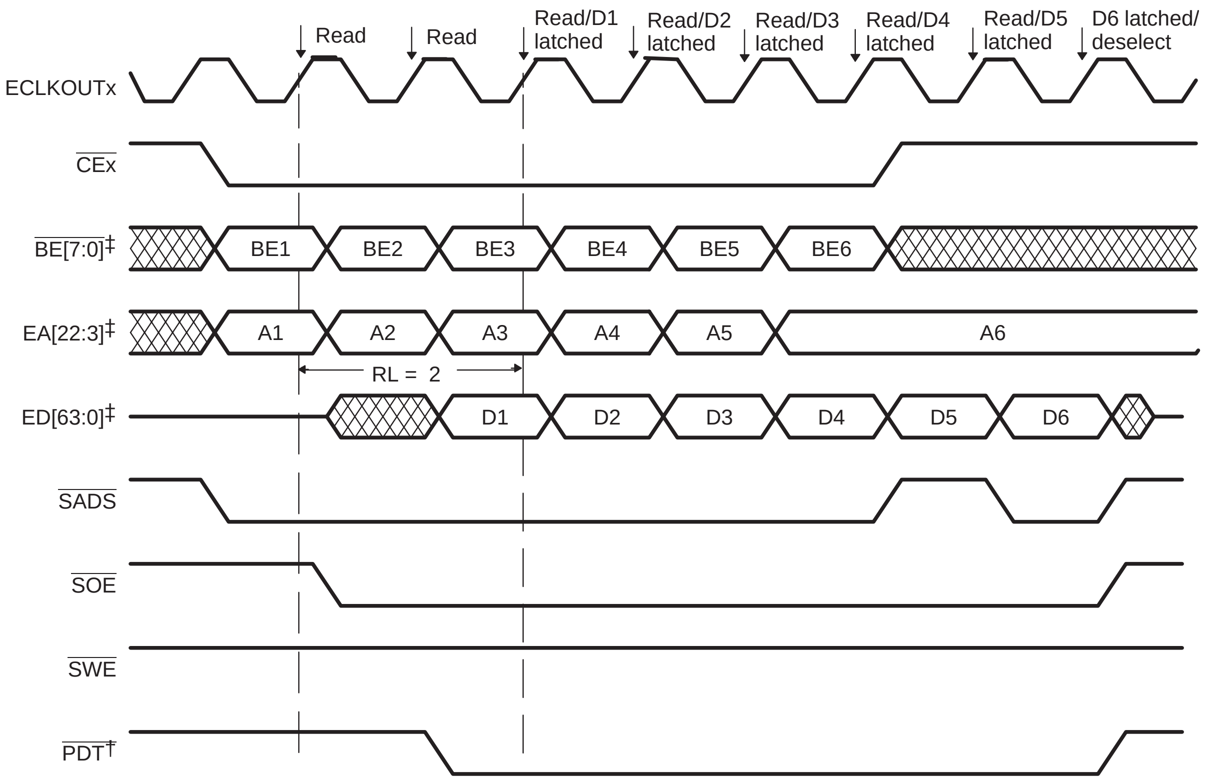
10.6.1.3 TMS320C64x SBSRAM Read

Figure 10–42 shows the timing for C64x six–element (doubleword for EMIFA, halfword for EMIFB) read with a two-cycle read latency. Every access strobes a new address into the SBSRAM, indicated by the $\overline{\text{SADS}}$ strobe low. The C64x EMIF issues a deselect cycle at the end of the burst transfer.

For the standard SBSRAM interface, the following fields in the CExSEC register must be set to their default state:

- SYNCRL = 10b; 2 cycle read latency
- SYNCWL = 00b; 0 cycle write latency
- CEEXT = 0; CE goes inactive after the final command has been issued
- RENEN = 0; $\overline{\text{SADS}}$/$\overline{\text{SRE}}$ signal acts as $\overline{\text{SADS}}$ signal

Figure 10–42. TMS320C64x SBSRAM Six-Element Read



† For PDT read from SBSRAM, $\overline{\text{PDT}}$ is asserted low during the data phase. For normal read/write transaction, the $\overline{\text{PDT}}$ signal is not asserted.

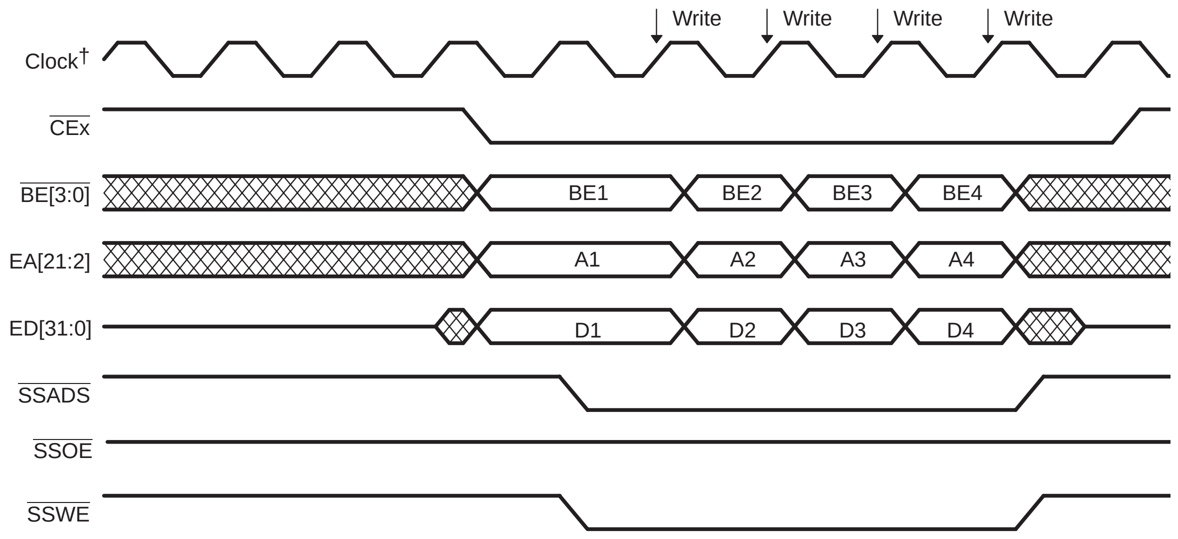
‡ For EMIFB, $\overline{\text{BE[1:0]}}$, EA[20:1], and ED[15:0], respectively, are used instead.

10.6.2 SBSRAM Writes

10.6.2.1 C620x/C670x SBSRAM Write

Figure 10–43 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 EMIF clock cycle.

Figure 10–43. TMS320C620x/C670x SBSRAM Four-Word Write

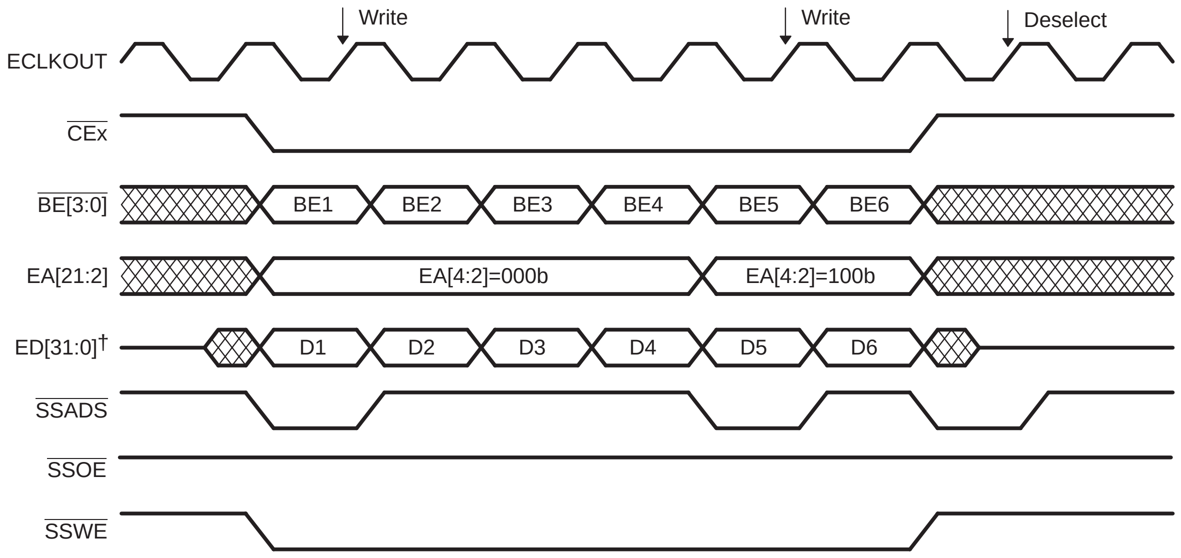


† Clock = SSCLK for C6201/C6701.
= CLKOUT2 for all C620x/C670x except C6201/C6701.

10.6.2.2 C621x/C671x SBSRAM Write

Figure 10–44 shows a C621x/C671x six-word write to SBSRAM. The new address is strobed into SBSRAM on the fifth cycle to prevent the SBSRAM's internal burst counter from rolling over to 000b.

Figure 10–44. TMS320C621x/C671x SBSRAM Write



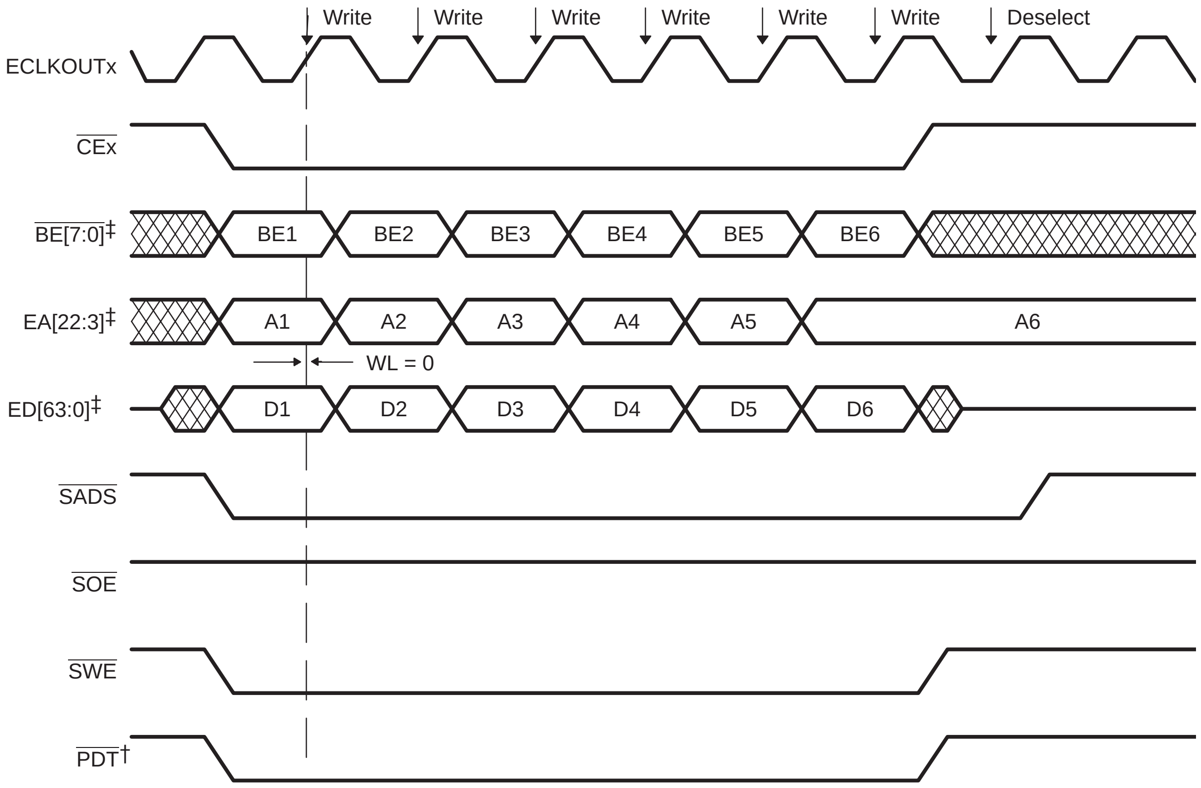
† ED[31:16] do not apply to C6712.

10.6.2.3 C64x SBSRAM Write

Figure 10–45 shows a C64x six-element (doubleword for EMIFA, halfword for EMIFB) write to SBSRAM. Every access strobes a new address into the SBSRAM. The C64x EMIF issues a deselect cycle at the end of the burst transfer.

Refer to section 10.6.1.3 for the CExSEC register setting for the C64x SBSRAM write interface.

Figure 10–45. TMS320C64x SBSRAM Six-Element Write



† For PDT write to SBSRAM, $\overline{\text{PDT}}$ is asserted low during the data phase, and the C64x puts data in high impedance. For normal read/write transaction, the $\overline{\text{PDT}}$ signal is not asserted.
‡ For EMIFB, $\overline{\text{BE}}$[1:0], EA[20:1], and ED[15:0], respectively, are used instead.

10.7 Programmable Synchronous Interface (TMS320C64x)

The C64x EMIF offers additional flexibility by replacing the SBSRAM interface with a programmable synchronous interface. The programmable synchronous interface supports glueless interfaces to the following devices:

- ❑ Pipelined and Flow-Through SBSRAM
- ❑ Zero bus turnaround (ZBT) synchronous pipeline SRAM
- ❑ Synchronous FIFOs in standard and first word fall through (FWFT) mode.

The programmable synchronous interface is configured by the CE space secondary control register (CExSEC). The bit fields in the CExSEC control the cycle timing, and the clock used for programmable synchronous interface synchronization. See section 10.2.2 for a detailed description of the bit fields.

In order to support different synchronous memory types, the C64x SBSRAM interface is a combination of the C620x/C670x and C621x/C671x interfaces. A new command is issued every cycle for SBSRAM bursts (similar to the C620x/C670x), and a deselect cycle is issued at the end of the burst (similar to the C621x/C671x). The RENEN field in the CExSEC register should be set to 0 for SBSRAM interface to enable the $\overline{\text{SADS}}$ signal. See section 10.6 for details on SBSRAM interface.

Table 10–31 shows the programmable synchronous interface pins.

Table 10–31. TMS320C64x Programmable Synchronous Pins

EMIF Signal	Signal Function
$\overline{\text{SADS}}$/$\overline{\text{SRE}}$	Address strobe/Read enable (selected by RENEN)
$\overline{\text{SOE}}$	Output enable
$\overline{\text{SOE3}}$	Output enable for $\overline{\text{CE3}}$. The $\overline{\text{SOE3}}$ pin is not muxed with other signals. (useful for glueless FIFO interface)
$\overline{\text{SWE}}$	Write Enable
ECLKOUT1	Synchronous interface clock, runs at 1x EMIF input clock rate
ECLKOUT2	Synchronous interface clock, runs at 1x, 1/2x, or 1/4x EMIF input clock rate

10.7.1 ZBT SRAM Interface

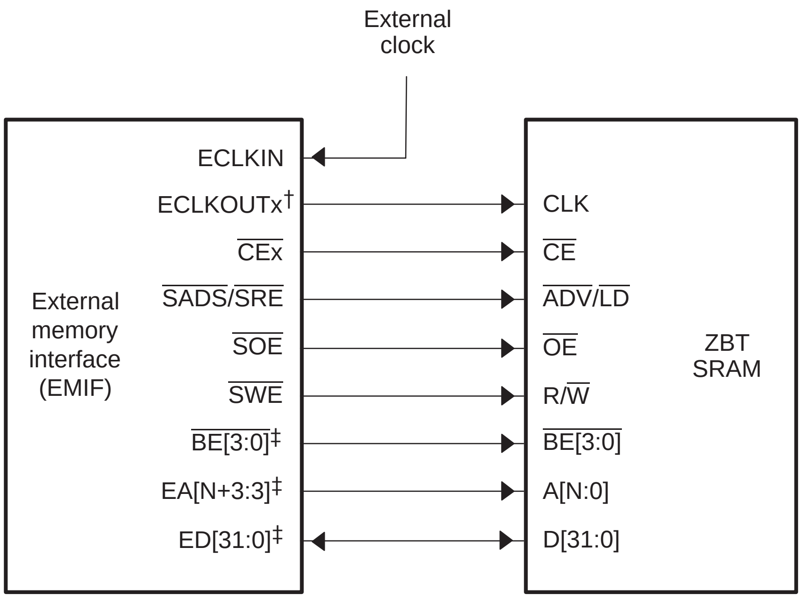
The programmable synchronous mode supports zero bus turnaround (ZBT) SRAM interface.

For ZBT SRAM interface, the following fields in the CExSEC register must be set as:

- ❑ SYNCRL = 10b; 2 cycle read latency
- ❑ SYNCWL = 10b; 2 cycle write latency
- ❑ CEEXT = 0; CE goes inactive after the final command has been issued
- ❑ RENEN = 0; $\overline{\text{SADS}}/\overline{\text{SRE}}$ signal acts as $\overline{\text{SADS}}$ signal.

Figure 10–46 shows the ZBT SRAM interface.

Figure 10–46. TMS320C64x ZBT SRAM Interface



† ECLKOUTx used is selected by the SNCCLK bit in the CExSEC register.
‡ For EMIFB, $\overline{\text{BE[1:0]}}$, EA[N+1:1], and ED[15:0] are used instead to interface to a 16–bit ZBT SRAM.

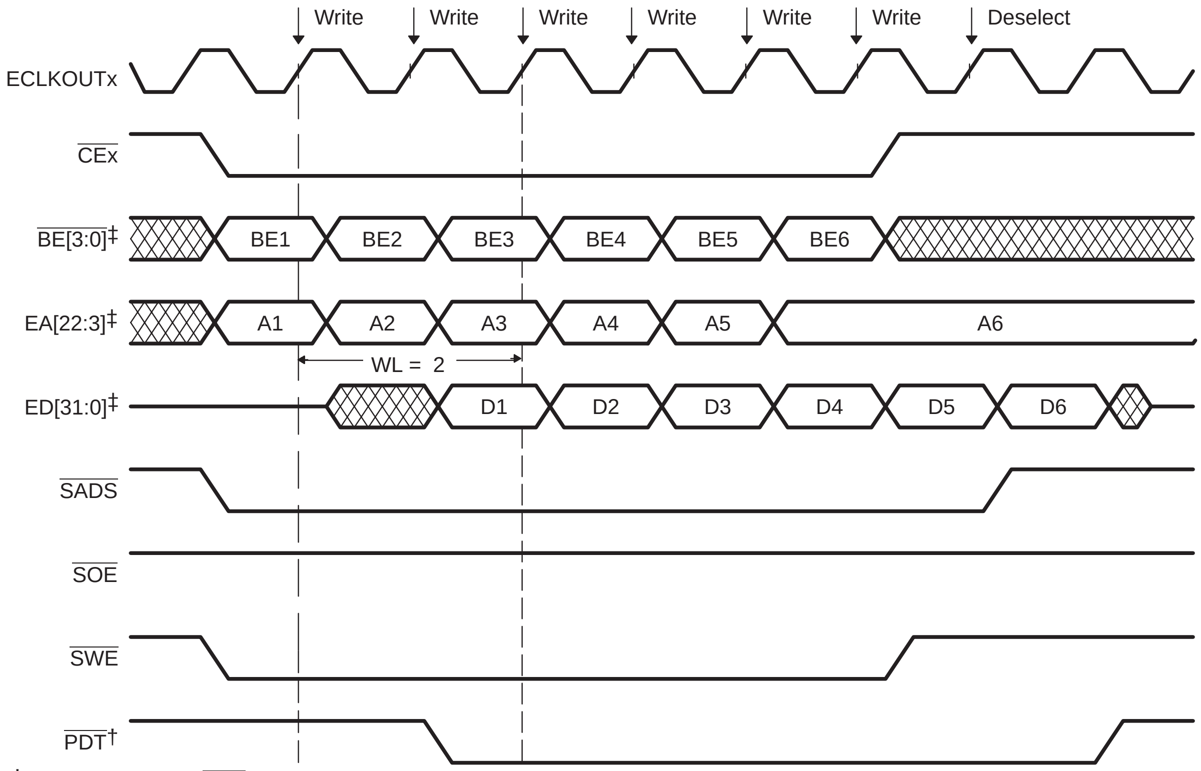
10.7.1.1 ZBT SRAM Read

ZBT SRAM read waveforms are identical to the SBSRAM read waveforms, since the register settings corresponding to the reads are the same. Refer to section 10.6.1.3 for details.

10.7.1.2 ZBT SRAM Write

For ZBT SRAM writes, the control signal waveforms are exactly the same as standard SRAM writes. The write data, however, is delayed by two cycles, as controlled by SYNCWL = 10b. Figure 10–47 shows the ZBT SRAM write timing.

Figure 10–47. TMS320C64x ZBT SRAM Six-Element Write



† For PDT transfers, $\overline{\text{PDT}}$ is asserted low during the data phase, and data is in high impedance. For normal read/write transaction, the $\overline{\text{PDT}}$ signal is not asserted.

‡ For EMIFB, $\overline{\text{BE[1:0]}}$, EA[21:1], and ED[15:0] are used instead.

10.7.2 Synchronous FIFO Interface

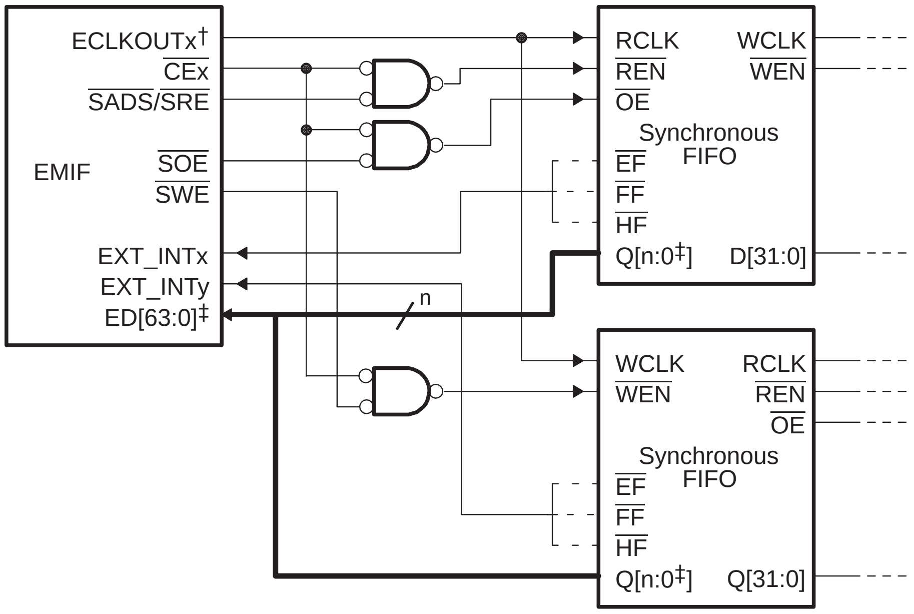
The programmable synchronous mode supports both standard timing synchronous FIFO interface, and first word fall through (FWFT) FIFO interface.

For synchronous FIFO interface, the following field in the CExSEC register must be set as stated:

- RENEN = 1; $\overline{\text{SADS}}$/$\overline{\text{SRE}}$ signal acts as $\overline{\text{SRE}}$ signal

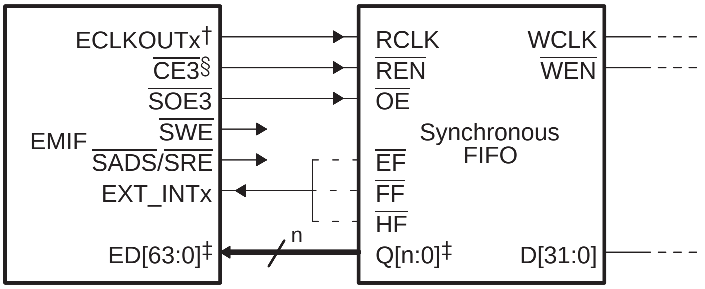
Figure 10–48 shows the synchronous FIFO interface with glue. Figure 10–49 and Figure 10–50 show the glueless synchronous FIFO interface at CE3 space, using the dedicated $\overline{\text{SOE3}}$ pin.

Figure 10–48. TMS320C64x Read and Write Synchronous FIFO Interface With Glue



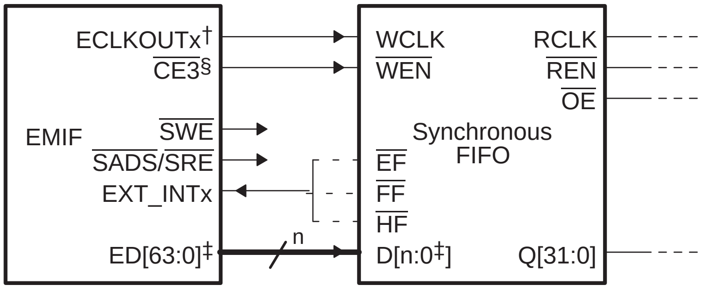
† ECLKOUTx used is selected by the SNCCLK bit in the CExSEC register.
‡ The MTYPE field selects the interface to be 8, 16, 32, or 64 bits wide. For EMIFB, only 8– and 16–bit interfaces are available. Therefore only ED[15:0] is used.

Figure 10–49. TMS320C64x Glueless Synchronous FIFO Read Interface in CE3 Space



† ECLKOUTx used is selected by the SNCCLK bit in the CExSEC register.
‡ The MTYPE field selects the interface to be 8, 16, 32, or 64 bits wide. For EMIFB, only 8– and 16–bit interfaces are available. Therefore only ED[15:0] is used.
§ Writes to CE3 must not be performed in this interface, since writes to CE3 will also cause CE3 to go active, causing data contention.

Figure 10–50. TMS320C64x Glueless Synchronous FIFO Write Interface in CE3 Space



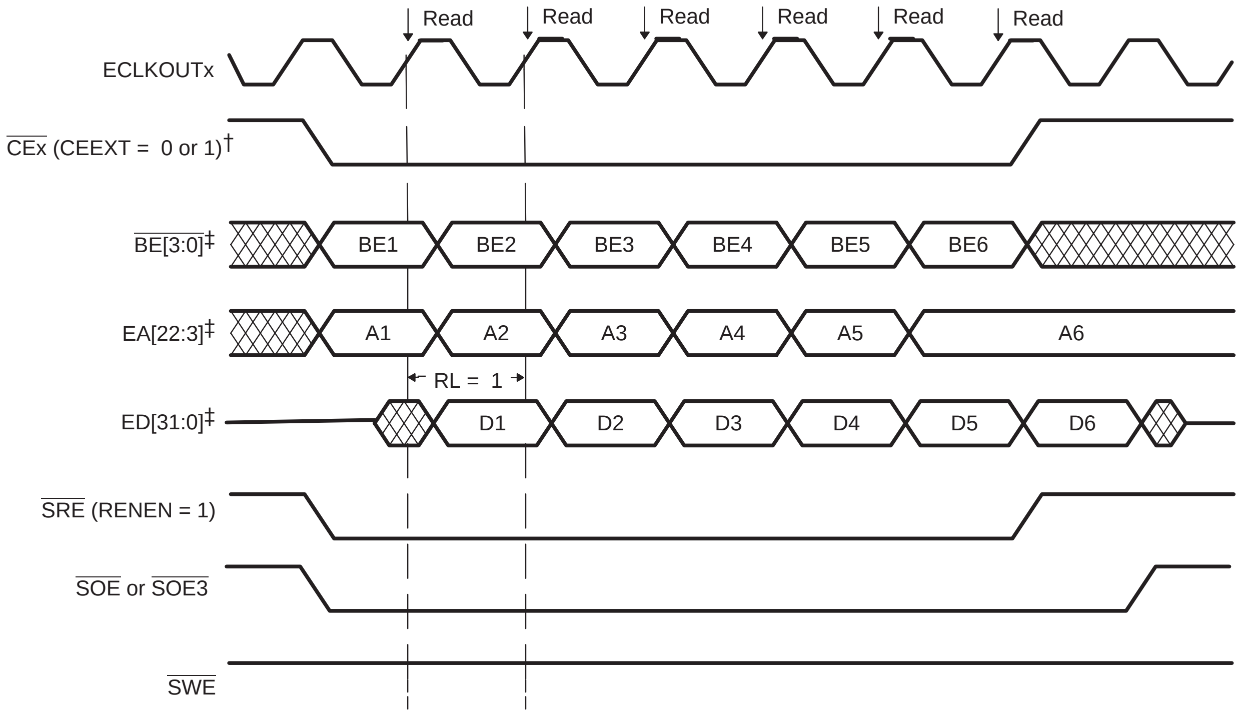
† ECLKOUTx used is selected by the SNCCLK bit in the CExSEC register.
‡ The MTYPE field selects the interface to be 8, 16, 32, or 64 bits wide. For EMIFB, only 8– and 16–bit interfaces are available. Therefore only ED[15:0] is used.
§ Reads to CE3 must not be performed in this interface, since reads to CE3 will also cause CE3 to go active, causing data contention.

10.7.2.1 Standard Synchronous FIFO Read

Figure 10–51 shows a C64x six-word read from a standard synchronous FIFO. The CExSEC register settings are as follows:

- ❑ SYNCRL = 01b; one cycle read latency
- ❑ RENEN = 1; $\overline{\text{SADS}}/\overline{\text{SRE}}$ signal acts as $\overline{\text{SRE}}$ signal
- ❑ CEEXT = 0; used for glueless FIFO interface
  = 1; used for FIFO interface with glue

Figure 10–51. TMS320C64x Standard Synchronous FIFO Read


Read
Read
Read
Read
Read
Read
ECLKOUTx
$\overline{\text{CEx}}$ (CEEXT = 0 or 1)†
$\overline{\text{BE[3:0]}}$‡
BE1
BE2
BE3
BE4
BE5
BE6
EA[22:3]‡
A1
A2
A3
A4
A5
A6
RL = 1
ED[31:0]‡
D1
D2
D3
D4
D5
D6
$\overline{\text{SRE}}$ (RENEN = 1)
$\overline{\text{SOE}}$ or $\overline{\text{SOE3}}$
$\overline{\text{SWE}}$

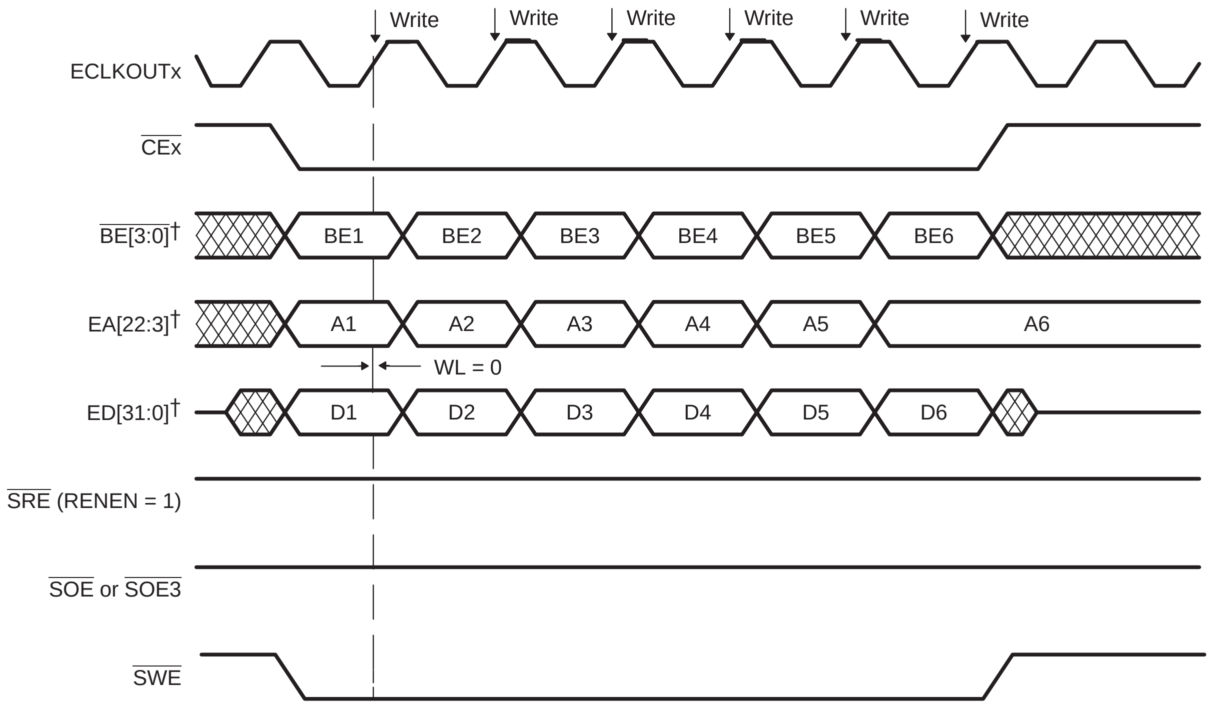
† CEEXT = 0 for glueless synchronous FIFO interface. CEEXT = 1 for interface with glue.
‡ For EMIFB, $\overline{\text{BE[1:0]}}$, EA[21:1], and ED[15:0] are used instead.

10.7.2.2 Standard Synchronous FIFO Write

Figure 10–52 shows a C64x six-word write to a standard synchronous FIFO. The CExSEC register settings are as follows:

- ❑ SYNCWL = 00b; zero cycle write latency
- ❑ RENEN = 1; $\overline{\text{SADS}}/\overline{\text{SRE}}$ signal acts as $\overline{\text{SRE}}$ signal

Figure 10–52. TMS320C64x Standard Synchronous FIFO Write



† For EMIFB, $\overline{\text{BE[1:0]}}$, EA[21:1], and ED[15:0] are used instead.

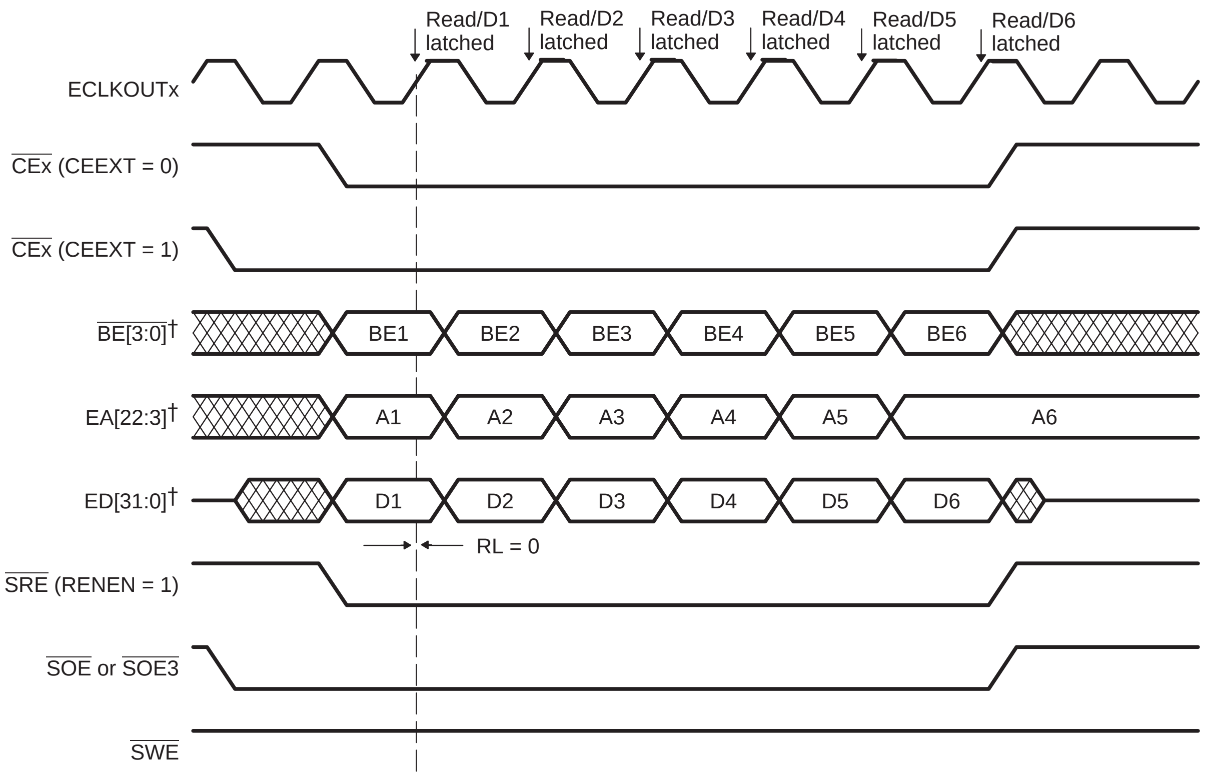
10.7.2.3 FWFT Synchronous FIFO Read

Figure 10–53 shows a C64x six-word read from a FWFT synchronous FIFO. The CExSEC register settings are as follows:

- ❑ SYNCRL = 00b; zero cycle read latency
- ❑ RENEN = 1; $\overline{\text{SADS}}/\overline{\text{SRE}}$ signal acts as $\overline{\text{SRE}}$ signal
- ❑ CEEXT = 0; used for glueless FIFO interface
  = 1; used for FIFO interface with glue

SYNCRL = 0 causes the $\overline{\text{SOE}}$ (or $\overline{\text{SOE3}}$) signal go active a cycle before the read command begins. If CEEXT=1, the $\overline{\text{CE}}$ signal will go active at the same time as the $\overline{\text{SOE}}$ signal.

Figure 10–53. TMS320C64x FWFT Synchronous FIFO Read



† For EMIFB, $\overline{\text{BE[1:0]}}$, EA[21:1], and ED[15:0] are used instead.

10.7.2.4 FWFT Synchronous FIFO Write

The FWFT Synchronous FIFO Write timing is identical to the standard synchronous FIFO write timing. See section 10.7.2.2.

10.8 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 10–32 lists the asynchronous interface pins.

Table 10–32. 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 10–54 shows an interface to standard SRAM, and Figure 10–56, Figure 10–57, and Figure 10–58 show interfaces to 8-, 16-, and 32-bit ROM for the C6000. For C620x/C670x, 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.

The C621x/C671x/C64x allows width of less than 32 bits on any CE space, as shown in the MTYPE description of the CExCTL register. Figure 10–55 shows the C621x/C671x interface to 16-bit asynchronous SRAM. The only difference on the C621x/C671x 16–bit interface is that ED[31:16] are used instead of ED[15:0]. The asynchronous interface signals on the C621x/C671x/C64x 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 time, 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.

Figure 10–54. EMIF to 32-bit SRAM Interface

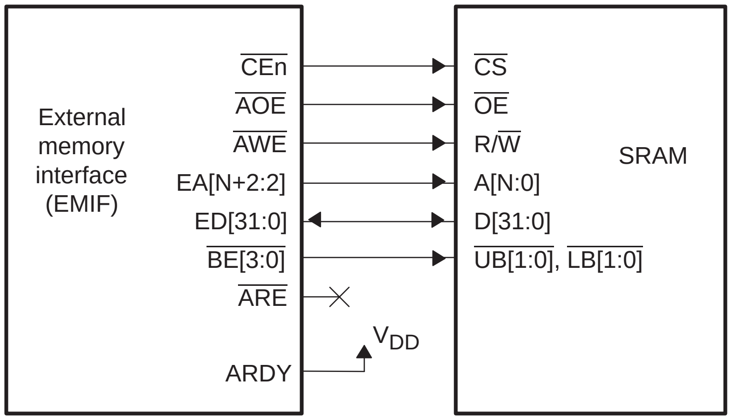
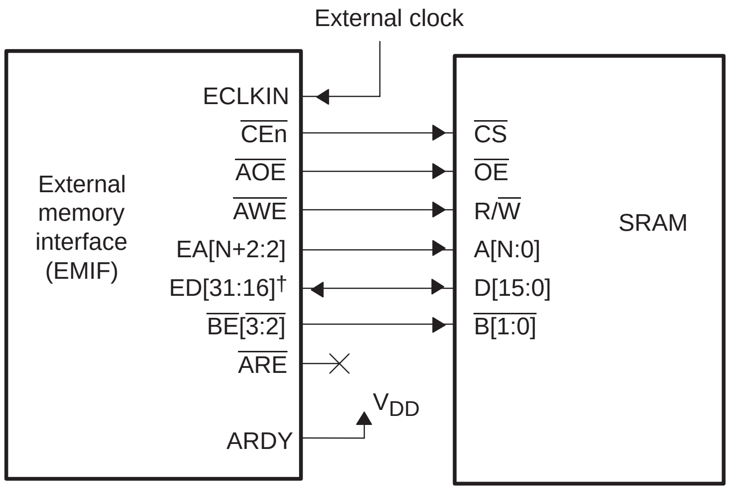
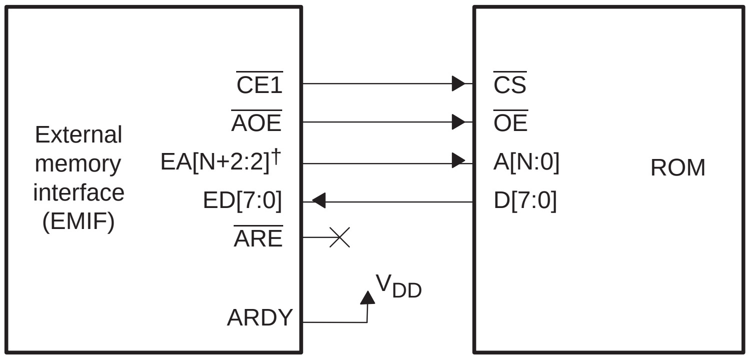


Figure 10–55. TMS320C621x/C671x EMIF to 16-bit SRAM (Big Endian)[†]



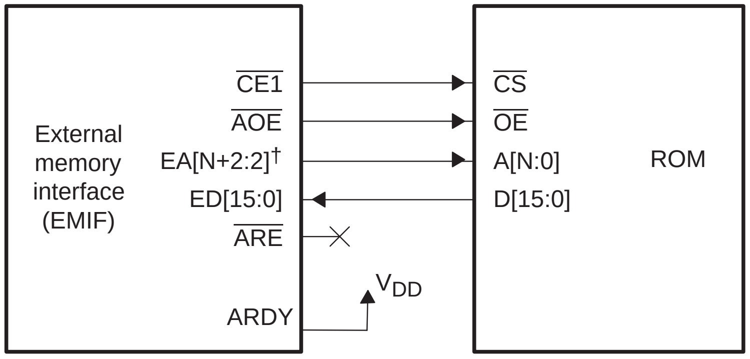
[†] Figure 10–55 does not apply to C6712 because ED[31:16] do not exist on C6712.

Figure 10–56. EMIF to 8-Bit ROM Interface



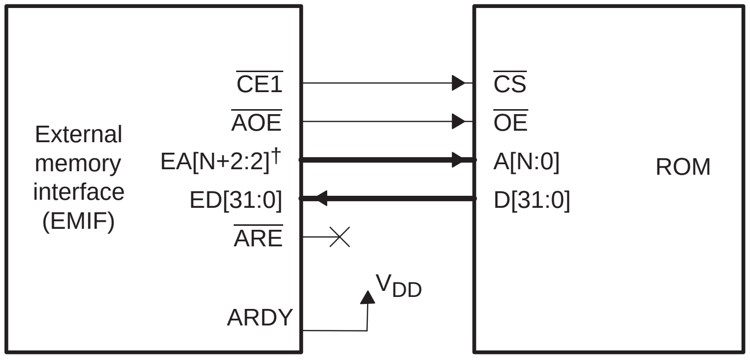
[†] For C64x EMIFA, EA[N+3:3] is used.
For C64x EMIFB, EA[N+1:1] is used.

Figure 10–57. EMIF to 16-Bit ROM Interface



[†] For C64x EMIFA, EA[N+3:3] is used.
For C64x EMIFB, EA[N+1:1] is used.

Figure 10–58. EMIF to 32-Bit ROM Interface



[†] For C64x EMIFA, EA[N+3:3] is used.
For C64x EMIFB, EA[N+1:1] is used.

Table 10–33 is an overview of similarities and differences on the C6000 ASRAM interface.

Table 10–33. TMS320C6000 ASRAM Interface Summary

	C620x/C670x	C621x/C671x	C64x	
			EMIFA	EMIFB
Interface width	32-bit ASRAM; x32, x16, x8 ROM	32-, 16-, 8-bit	64-, 32-, 16-, 8-bit	16-, 8-bit
Internal sychronization	CLKOUT1	ECLKOUT	ECLKOUT1	ECLKOUT1
Control signals	Dedicated ASRAM control signals	ASRAM control signals are MUXed with SDRAM and SBSRAM control signals.	ASRAM control signals are MUXed with SDRAM and programmable syn-chronous control signals.	ASRAM control signals are MUXed with SDRAM and programmable sychronous control signals.
Memory Endianness	Packing format in ROM is little-endian only	Supports both little- or big-endian	Supports both little- or big-endian	Supports both little- or big-endian

10.8.1 TMS320C620x/C670x 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 10–34 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 10–34. Byte Address to EA Mapping for Asynchronous Memory Widths

	EA Line																			
	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	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2
×16	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1
×8	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0

10.8.1.1 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, regardless of the access size. For example, a byte read will result in the EMIF performing 4 byte accesses on the external bus, but only one byte is returned from the EMIF to the requestor. 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.

10.8.1.2 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.

10.8.2 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 ($\overline{\text{CE}}$ low, address valid) and the activation of the read or write strobe
- **Strobe:** The time between the activation and deactivation of the read ($\overline{\text{ARE}}$) or write strobe ($\overline{\text{AWE}}$)
- **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 $\overline{\text{CE}}$ signal)

For the C620x/C670x these parameters are programmable in terms of CPU clock cycles via fields in the EMIF CE space control registers. For the C621x/C671x/C64x, these parameters are programmed in terms of ECLKOUT (or ECLKOUT1) cycles. Separate set-up, strobe, and hold timing parameters are available for read and write accesses. Minimum values for ASRAM are as follows:

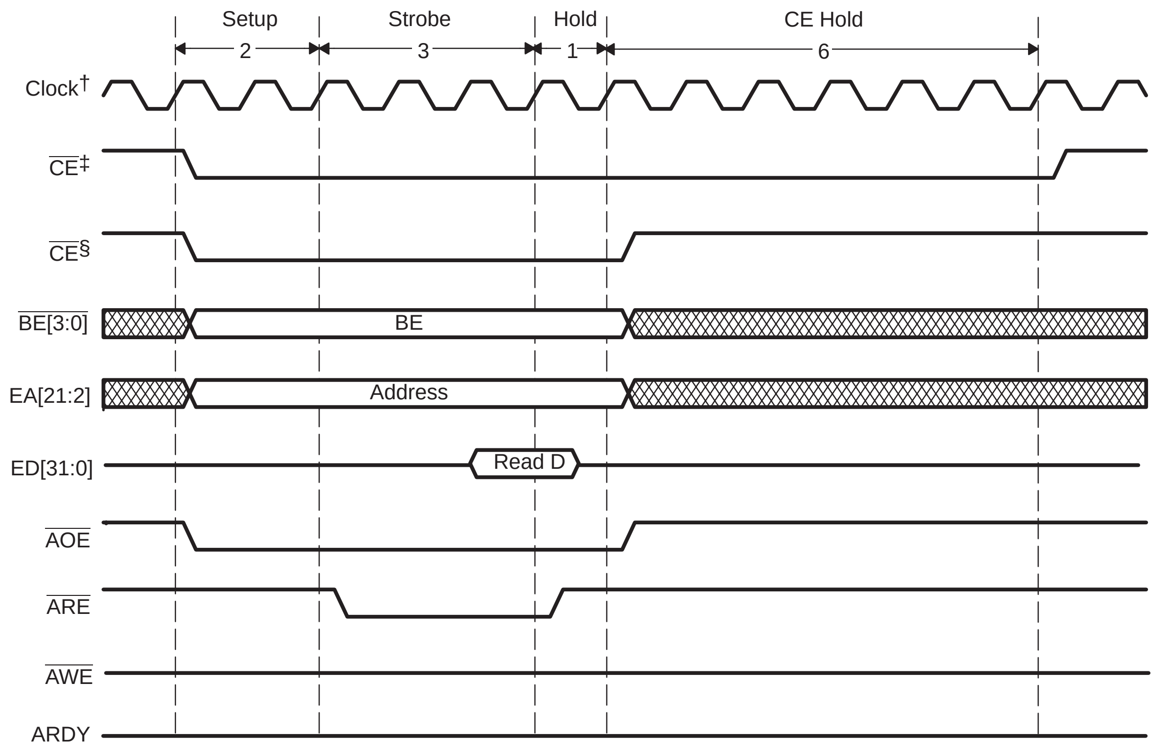
- SETUP ≥ 1 (0 treated as 1)
- STROBE ≥ 1 (0 treated as 1)
- HOLD ≥ 0
- On the C620x/C670x first access in a set of consecutive accesses or a single access, the setup period has a minimum count of 2.

10.8.3 Asynchronous Reads

Figure 10–59 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.
  - For C620x/C670x, the first access has a setup period minimum value of 2. After the first access, setup has a minimum value of 1.
- 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 or 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 C620x/C670x, 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 C621x/C671x/C64x, the $\overline{\text{CEn}}$ signal goes high just after the programmed hold period.

Figure 10–59. Asynchronous Read Timing Example



† Clock = CLKOUT1 for C620x/C670x
= ECLKOUT for C621x/C671x
= ECLKOUT1 for C64x

‡ $\overline{\text{CE}}$ waveform for C620x/C670x

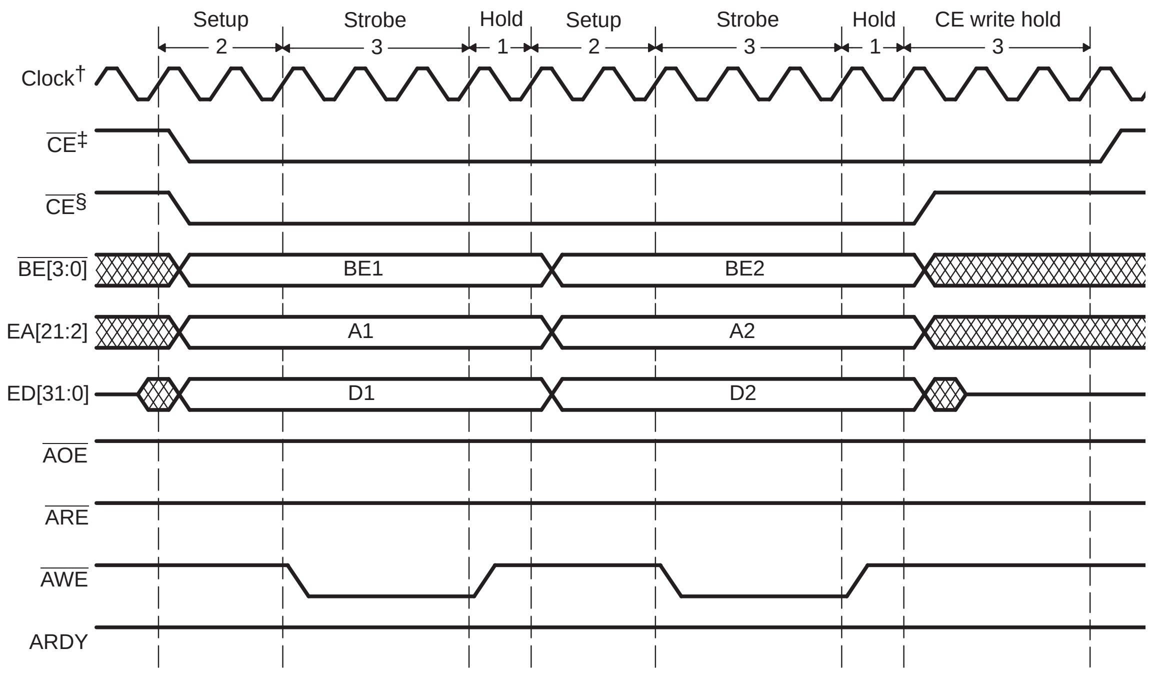
§ $\overline{\text{CE}}$ waveform for C621x/C671x/C64x

10.8.4 Asynchronous Writes

Figure 10–60 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 C620x/C670x, the first access has a setup period 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 C620x/C670x, 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 for four more cycles. This does not affect performance and merely reflects the EMIF's overhead.
- For the C621x/C671x/C64x, the $\overline{\text{CEn}}$ signal goes high immediately after the programmed hold period.

Figure 10–60. Asynchronous Write Timing Example



† Clock = CLKOUT1 for C620x/C670x
= ECLKOUT for C621x/C671x
= ECLKOUT1 for C64x
‡ $\overline{\text{CE}}$ waveform for C620x/C670x
§ $\overline{\text{CE}}$ waveform for C621x/C671x/C64x

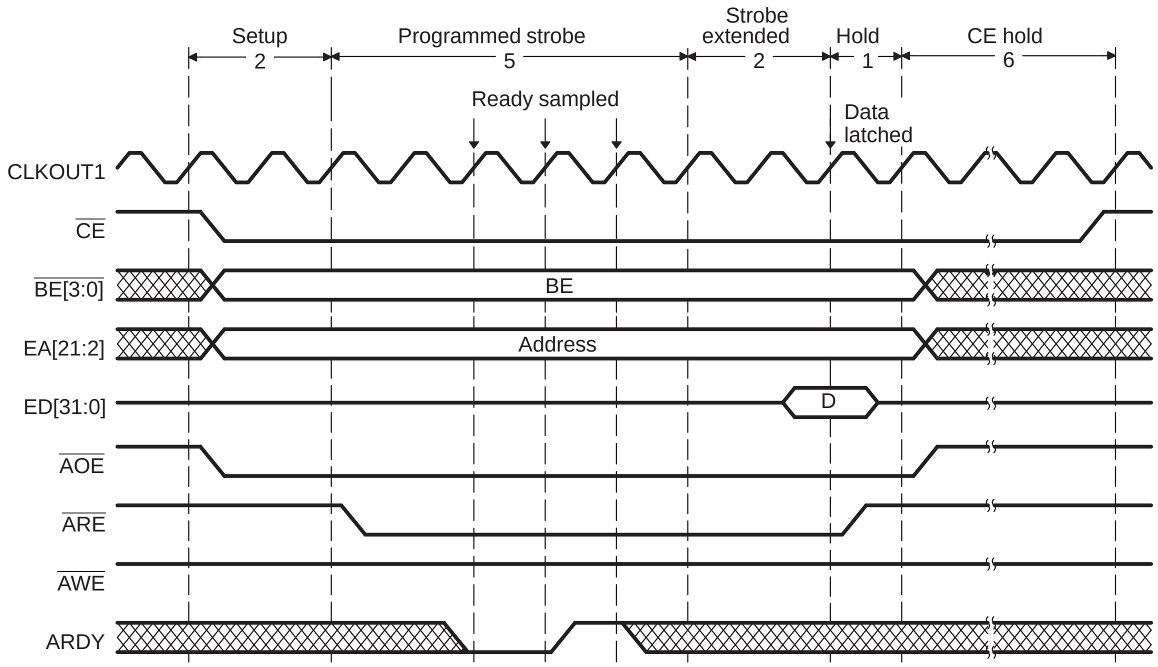
10.8.5 Ready Input

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 (C620x/C670x), ECLKOUT (C621x/C671x), or ECLKOUT1 (C64x). This synchronization allows an asynchronous ARDY input while avoiding metastablility.

- ❑ **TMS320C620x/C670x 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.

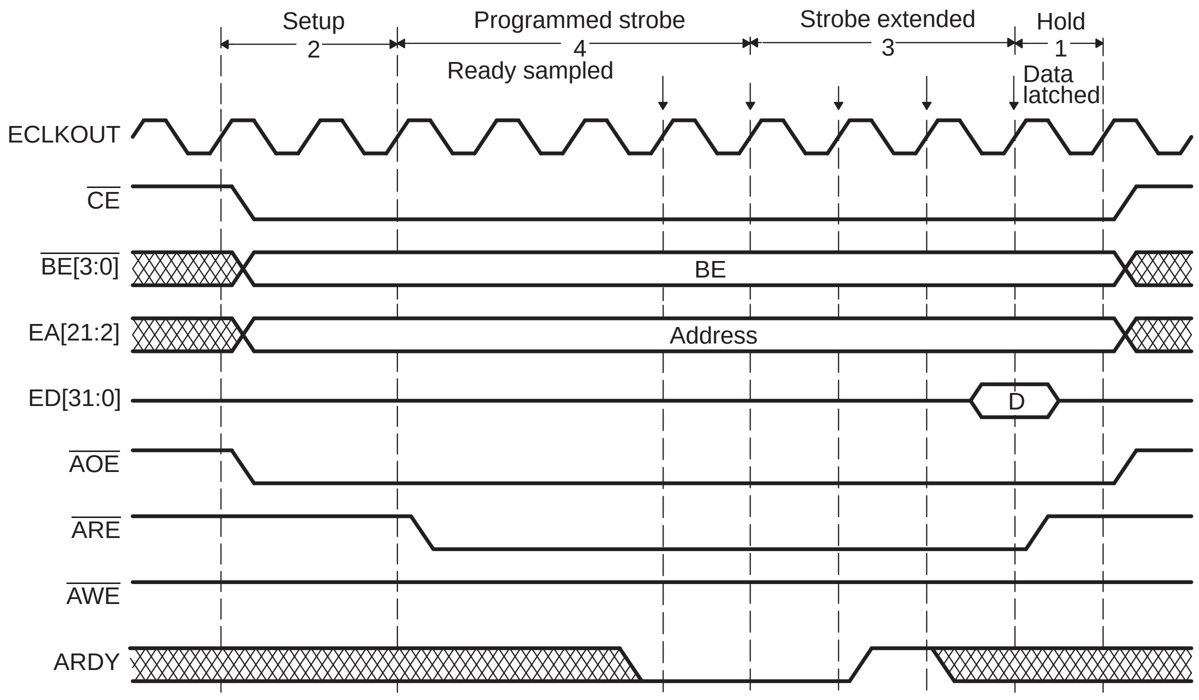
The read cycle in Figure 10–61 illustrates ready operation for the C620x/C670x.

Figure 10–61. TMS320C620x/C670x Ready Operation



- **TMS320C621x/C671x/C64x 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 C621x 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 10–62, although data is latched by the C621x after the second cycle.

Figure 10–62. TMS320C621x/C671x Ready Operation



10.9 Peripheral Device Transfers (PDT) (TMS320C64x)

A Peripheral Device Transfer (PDT) allows the user to directly transfer data from an external peripheral (such as a FIFO) to another external memory (such as SDRAM), and vice versa. Normally, this type of transfer would require an EMIF read of a peripheral followed by an EMIF write to memory, or an EMIF read of a memory followed by an EMIF write to a peripheral.

In a typical system, however, both the peripheral and memory are connected to the same physical data pins, and thus an optimization can be made. In a PDT write transfer, data is driven by the peripheral directly, and written to the memory in the same bus transaction. In a PDT read transfer, data is driven by the memory directly, and written to the peripheral in the same bus transaction. Typically the memory device will be mapped to an addressable location via a $\overline{\text{CEx}}$ signal. The peripheral device is normally not memory–mapped (it doesn't use a $\overline{\text{CEx}}$ signal). It is activated with the $\overline{\text{PDT}}$ signal and optionally a combination of other control signals (via external logic).

PDT transfers are classified in terms of the memory on the EMIF. A PDT write is a transfer from a peripheral to memory (memory is physically written). A PDT read is a transfer from memory to a peripheral (memory is physically read). For a PDT read, the EMIF ignores the read data on the external bus. For a PDT write, the EMIF data bus is three–stated during the transaction to allow the external peripheral or memory to drive the data bus.

In a PDT transaction, the EMIF:

1) Generates normal read control signals for a PDT read, or generates normal write control signals for a PDT write. For example, for a PDT read from CE0 with an SDRAM, the EMIF asserts $\overline{\text{CE0}}$ and generates the SDRAM read control signals. The EMIF does not explicitly generate the control signals to the destination in a PDT read. For a PDT write to CE0 with an SDRAM, the EMIF asserts $\overline{\text{CE0}}$ and generates the SDRAM write control signals. The EMIF does not explicitly generate the control signals to the source in a PDT write.

2) Generates PDT control signal $\overline{\text{PDT}}$, and the PDT address pin if SDRAM is involved in the PDT transaction. $\overline{\text{PDT}}$ is asserted low during the data phase of the transaction.

3) Drives EMIF data outputs (ED pins) to high–impedance.

To perform a PDT transfer, the PDTS or PDTD bits must be set appropriately in the EDMA options parameter. Refer to *Chapter 6, EDMA Controller*, for details. The $\overline{\text{PDT}}$ pin and a PDT address pin are used for PDT transfers. The PDT address pin used in PDT is:

- ❑ EMIFA: EA19
- ❑ EMIFB: EA17

The PDT address pin is only used if SDRAM is either the source of a PDT read transfer, or the destination of a PDT write transfer. In that case, the PDT address pin goes active during the ACTV cycle (RAS cycle).

10.9.1 PDT Write

A PDT write transfer refers to a transfer from a peripheral to memory, in which the memory is physically written. Figure 10–63 shows an example of a PDT transfer from a FWFT FIFO to an SDRAM. Note in this example that no glue is required. However, if both read and write PDT transactions are required on the same bus, glue will be required to properly create the $\overline{\text{REN}}$ signal and $\overline{\text{WEN}}$ signal for the FIFO.

To enable a PDT write transfer, the PDTD bit in the EDMA options field must be set to 1.

Figure 10–63. PDT Write Transfer From FIFO to SDRAM

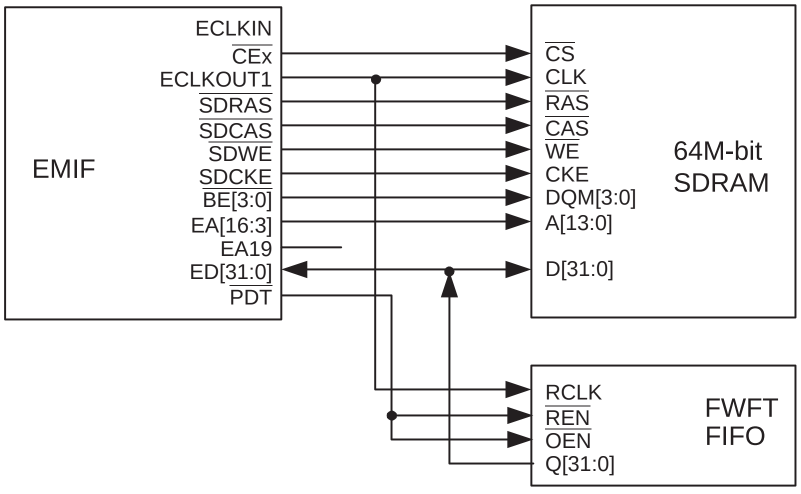


Figure 10–63 assumes that a FWFT FIFO is used, which has a zero cycle read latency. The $\overline{\text{PDT}}$ signal goes active ONLY during the data phase of the transaction. Refer to the SDRAM timing diagrams in section 10.5.11.2 for details. The peripheral (FIFO in this example) must drive data immediately when the $\overline{\text{PDT}}$ signal goes low.

Both the PDT address pin and the $\overline{\text{PDT}}$ pin assertion/deassertion are timed according to destination memory clock. If the destination memory is SDRAM, ECLKOUT1 is used.

A PDT write transfer procedure is as follows:

1) If the destination address is to a CE space not configured as SDRAM, then the PDT address pin is not used. If the destination address is to a CE space set as SDRAM, then:
   - If the access is to a closed page, then during the ACTV cycle, the PDT address pin will be asserted low.
   - If the access is to an open page previously accessed without a PDT operation, then the page will be closed and reopened, with the PDT address pin asserted low during the ACTV cycle.
   - If the access is to an open page previously accessed with a PDT operation, then the access will go directly to the data phase.
   - The PDT address bit is used to give the system advance warning that a PDT transaction is pending. This may be useful to activate bus switches or other external logic which will control the actual PDT transfer.
2) Normal write control signals are generated to the appropriate CE space.
3) The write transaction proceeds as normal except:
   - EMIF data outputs remain High-Z. Therefore the memory will read data from the peripheral device, instead of data from the EMIF.
   - $\overline{\text{PDT}}$ asserted low on the cycle that data is to be latched by the destination device. This implies that the peripheral must drive valid data on the same cycle that $\overline{\text{PDT}}$ is active.

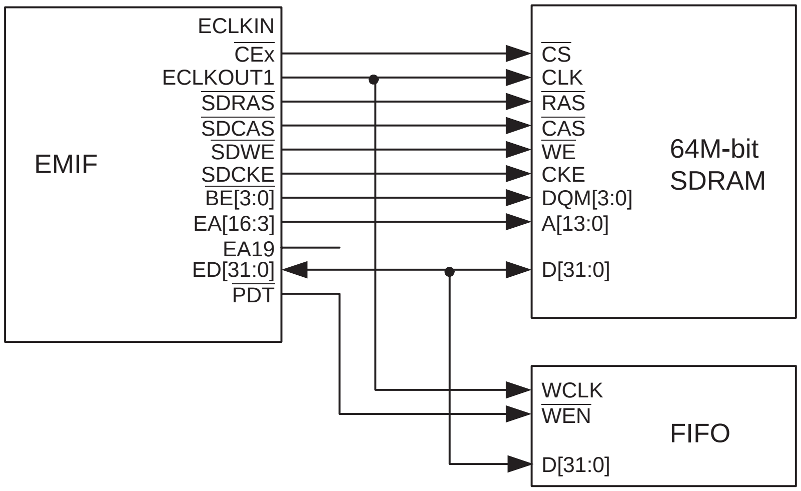
The $\overline{\text{PDT}}$ signal can be tied directly to the $\overline{\text{REN}}$ and/or $\overline{\text{OE}}$ signal of the source peripheral. Alternatively, $\overline{\text{PDT}}$ can be gated through external logic to accomplish more complicated transactions.

10.9.2 PDT Read

A PDT read transfer refers to a transfer from a memory to a peripheral, in which the memory is physically read. Figure 10–64 shows an example of a PDT transfer from an SDRAM to a FIFO. Note in this example that no glue is required. However, if both read and write PDT transactions are required on the same bus, glue will be required to properly create the $\overline{\text{REN}}$ signal and $\overline{\text{WEN}}$ signal for the FIFO.

To enable a PDT read transfer, the PDTS bit in the EDMA options field must be set to 1.

Figure 10–64. PDT Read Transfer From SDRAM to FIFO



Refer to section 10.5.10.3 for details on the SDRAM read timing diagram.

Both the PDT address pin and the $\overline{\text{PDT}}$ pin assertion/deassertion are timed according to source memory clock. If the source memory is SDRAM, then ECLKOUT1 is used.

A PDT read transfer procedure is as follows:

1) If the source address is to a CE space not configured as SDRAM, then the PDT address pin is not used. If the source address is to a CE space set as SDRAM, then:
   - If the access is to a closed page, then during the ACTV cycle, the PDT address pin will be asserted low.
   - If the access is to an open page previously accessed without a PDT operation, then the page will be closed and reopened, with the PDT address pin asserted low during the ACTV cycle.
   - If the access is to an open page previously accessed with a PDT operation, then the access will go directly to the data phase.
   - The PDT address bit is used to give the system advance warning that a PDT transaction is pending. This may be useful to activate bus switches or other external logic which will control the actual PDT transfer.
2) Normal read control signals are generated to the appropriate CE space.
3) The read transaction proceeds as normal except:
   - EMIF ignores data at the ED pins
   - $\overline{\text{PDT}}$ asserted low on the cycle that data is to be returned by the source device.

10.10 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). During active $\overline{\text{RESET}}$, the clock outputs behave as follows:

- ❑ SSCLK, SDCLK: driven high or low
- ❑ CLKOUT1: continues clocking unless the values on the PLL configuration pins are changed.
- ❑ CLKOUT2: On the C620x/C670x, CLKOUT2 is driven high or low. On the C621x/C671x, CLKOUT2 continues clocking.
- ❑ Other CLKOUTx: On the C64x, all CLKOUTx continues clocking.
- ❑ ECLKOUT: On the C621x/C671x, ECLKOUT will continue to clock as long as ECLKIN is provided. ECLKIN should be provided during reset in order to drive EMIF signals to the correct reset values.
- ❑ ECLKOUTn: On the C64x, the EK1HZ and EK2HZ bits in the EMIF Global Control Register determines the state of ECLKOUT1 and ECLKOUT2 during reset.

10.11 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 BUSREQ, $\overline{\text{HOLDA}}$, and the clock outputs (CLKOUT1, CLKOUT2, ECLKOUT, SDCLK, and/or SSCLK, depending on the device). For the C64x, the EKxHZ bits in the GBLCTL register determine the state of the ECLKOUTx signals while $\overline{\text{HOLDA}}$ is asserted. 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 (C621x/C671x/C64x 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. For C64x, the BRMODE bit in the GBLCTL register indicates the bus request mode (section 10.2.1).

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.

10.11.1 Reset Considerations With the Hold Interface

For the C62x/C67x, if a Hold request is pending ($\overline{\text{HOLD}}$ low) upon exiting reset, the EMIF outputs will be driven for a brief period of time (< 5 CLKOUT2/ECLKOUT cycles) in the default state. That is, active-low output strobes will be high and address outputs are driven low. If other memory controller devices are connected on the bus, there is a potential for data contention if the outputs are driven at opposite states during this short amount of time. If multiple C62x/C67x devices are connected on the same bus and come out of reset simultaneously, then the brief period of time that the output buffers are all driven will not result in device damage, since the outputs are all driven to the same logic level.

For the C64x, if a hold request is pending upon exiting reset, none of the EMIF output signals are driven active. All output signals stay in a high-impedance state. The $\overline{\text{HOLDA}}$ signal is asserted immediately.

10.12 Memory Request Priority

10.12.1 TMS320C620x/C670x Memory Request Priority

The C620x/C670x EMIF has multiple requestors competing for the interface. Table 10–35 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 10–35. TMS320C620x/C670x 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

† DMC = Data Memory Controller
‡ PMC = Program Memory Controller

10.12.2 TMS320C621x/C671x/C64x Memory Request Priority

The C621x/C671x/C64x 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 10–36. TMS320C621x/C671x/C64x EMIF Prioritization of Requests

Priority	**Requestor**
Highest	External hold
	Mode register set
	refresh
Lowest	EDMA†

† Refer to *Chapter 6 EDMA Controller,* for details on prioritization within the EDMA.

10.13 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 (C620x/C670x only)
- 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.

10.14 Clock Output Enabling

To reduce electromagnetic interference (EMI) radiation, the C62x/C67x 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. ECLKOUT on the C621x/C671x cannot be disabled using software.

For the C64x, the operation of the EMIF and device clocks is highly flexible. The CLKOUTx and ECLKOUTx can be disabled by setting the appropriate bits (CLK4EN, CLK6EN, EK1EN, EK2EN) in the EMIF global control register (GBLCTL). The ECLKOUT2 can be configured to run at 1x, 1/2x, or 1/4x the ECLKIN rate for the generic synchronous interface. In addition, the EK1HZ and EK2HZ bits in the GBLCTL configure the output EMIF clock behavior during hold. Table 10–36 summarizes the function of the EKxEN and EKxHZ bits. See also section 10.2.1.

On the C64x, the CLKOUT4 and CLKOUT6 pins are MUXed with the general–purpose input/output (GPIO) pins GP1 and GP2, respectively. When these pins are configured as GPIO pins by setting the GPIO Enable Register (GPEN), the corresponding CLKxEN bits in the GBLCTL are ignored.

Table 10–37. EMIF Output Clock (ECLKOUTx) Operation

EKxEN	EKxHZ	ECLKOUTx Behavior
0	0	ECLKOUTx remains low
0	1	ECLKOUTx low, except during Hold. In high–impedance during Hold.
1	0	ECLKOUTx clocking
1	1	ECLKOUTx clocking, except during Hold. In high–impedance during Hold.

10.15 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.

10.16 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 15, Power-Down Logic,* for further details on power-down modes.

For the C621x/C671x/C64x, refreshes are issued to SDRAM if ECLKIN is provided.

Chapter 11

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**

11.1 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.
- Device configurations

11.2 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. Most 3-state outputs are placed in the high-impedance state, and most 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. For reset timing refer to the specific device datasheets.

11.3 Memory Map

11.3.1 TMS320C6201/C6204/C6205/C6701 Memory Map

The C6201/C6204/C6205/C6701 has two memory maps, MAP 0 and MAP 1, which are summarized in Table 11–1. The maps differ in that MAP 0 has external memory mapped at address 0, and MAP 1 has internal memory mapped at address 0. For descriptions, refer to *Chapter 2 Internal Program and Data Memory.*

Table 11–1. TMS320C6201/C6204/C6205/C6701 Memory Map Summary

		Description of Memory Block in ...	
Address Range (Hex)	**Size (Bytes)**	**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 (C6201/C6701) or XBUS (C6204) 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 – 019F FFFF	256K	Internal peripheral bus interrupt selector registers	
01A0 0000 – 01A3 FFFF	256K	Reserved	
01A4 0000 – 01A8 FFFF	320K	Internal peripheral bus PCI registers (C6205 only)†	
01A9 0000 – 01FF FFFF	6M–576K	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 – 3FFF FFFF	1G–64M	Reserved	
4000 0000 – 4FFF FFFF	256M	Expansion bus XCE0 (C6204 only)†	
5000 0000 – 5FFF FFFF	256M	Expansion bus XCE1 (C6204 only)†	
6000 0000 – 6FFF FFFF	256M	Expansion bus XCE2 (C6204 only)†	
7000 0000 – 7FFF FFFF	256M	Expansion bus XCE3 (C6204 only)†	
8000 0000 – 8000 FFFF	64K	Internal data RAM	
8001 0000 – FFFF FFFF	2G–64K	Reserved	

Note: † Reserved for other devices.

11.3.2 TMS320C6202(B) Memory Map

The C6202(B) has two memory maps that are supersets of the C6201/C6701 memory maps. All valid C6201/C6701 address ranges are valid on the C6202(B). The memory maps for the C6202(B) are shown in Table 11–2.

Table 11–2. TMS320C6202(B) Memory Map Summary

Address Range (Hex)	Size (Bytes)	Description of Memory Block In…	
		MAP 0	**MAP 1**
0000 0000 – 0003 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	
8002 0000 – FFFF FFFF	2G–128K	Reserved	

11.3.3 TMS320C6203(B) Memory Map

The C6203(B) Memory Map is very similar to the memory map of the C6202. The differences exist because of the increased amount of internal memory available on the C6203(B). The memory maps for the C6203(B) are shown in Table 11–3.

Table 11–3. TMS320C6203(B) Memory Map Summary

Address Range (Hex)	Size (Bytes)	Description of Two Memory Blocks	
		Map 0	**Map 1**
0000 0000–0005 FFFF	384K	External memory interface CE0	Internal program RAM
0006 0000–003F FFFF	4M–384K	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 0000–0145 FFFF	384K	Internal program RAM	External memory interface CE1
0146 0000–017F FFFF	4M–384K	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 0000–8007 FFFF	512K	Internal data RAM	
8008 0000–FFFF FFFF	2G–128K	Reserved	

11.3.4 TMS320C621x/C671x Memory Map

The C621x/C671x has only one memory map, which is shown in Table 11–4. 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 C620x/C670x and C621x/C671x are located at the same addresses in both processors. The external memory address ranges begin at location 8000 0000h in the C621x/C671x, which is the location of internal data memory in the C620x.

Table 11–4. TMS320C621x/C671x 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 – 01FF FFFF	6M–256K	Reserved
0200 0000 – 0200 0033	52	QDMA registers
0200 0034 – 2FFF FFFF	736M–52	Reserved
3000 0000 – 3FFF FFFF	256M	McBSP 0/1 data
4000 0000 – 7FFF 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

11.3.5 TMSC64x Memory Map

Similar to the C621x, the C64x has only one memory map, which is shown in Table 11–5. 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 C621x and C64x are located at the same addresses in both processors. The external memory address ranges begin at 0x600000000 for EMFB, and begin at 0x80000000 for EMIFA.

Table 11–5. TMS320C64x Memory Map Summary

Address Range (Hex)	Size (Bytes)	Description of Memory Block
0x00000000 – 0x000FFFFF	1M	Internal RAM (L2)
0x00100000 – 0x017FFFFF	23M	Reserved
0x01800000 – 0x0183FFFF	256K	Internal configuration bus EMIFA registers
0x01840000 – 0x0187FFFF	256K	Internal configuration bus L2 control registers
0x01880000 – 0x018BFFFF	256K	Internal configuration bus HPI registers
0x018C0000 – 0x018FFFFF	256K	Internal configuration bus McBSP0 registers
0x01900000 – 0x0193FFFF	256K	Internal configuration bus McBSP1 registers
0x01940000 – 0x0197FFFF	256K	Internal configuration bus timer 0 registers
0x01980000 – 0x019BFFFF	256K	Internal configuration bus timer 1 registers
0x019C0000 – 0x019FFFFF	256K	Internal configuration bus interrupt selector registers
0x01A00000 – 0x01A3FFFF	256K	Internal configuration bus EDMA RAM and registers
0x01A40000 – 0x01A7FFFF	256K	Internal configuration bus McBSP2 registers
0x01A80000 – 0x01ABFFFF	256K	Internal configuration bus EMIFB registers
0x01AC0000 – 0x01AFFFFF	256K	Internal configuration bus timer 2 registers
0x01B00000 – 0x01B3FFFF	256K	Internal configuration bus GPIO registers
0x01B40000 – 0x01B7FFFF	256K	Internal configuration bus UTOPIA registers† (C6415 only)
0x01B80000 – 0x01BFFFFF	512K	Reserved
0x01C00000 – 0x01C3FFFF	256K	Internal configuration bus PCI registers† (C6415 only)
0x01C40000 – 0x01FFFFFF	4M–256K	Reserved
0x02000000 – 0x02000033	52	QDMA registers

† Address range is reserved for C6414.

Table 11–5. TMS320C64x Memory Map Summary (Continued)

Address Range (Hex)	Size (Bytes)	Description of Memory Block
0x02000034 – 0x2FFFFFFF	736M–52	Reserved
0x30000000 – 0x33FFFFFF	64M	McBSP0 data
0x34000000 – 0x37FFFFFF	64M	McBSP1 data
0x38000000 – 0x3BFFFFFF	64M	McBSP2 data
0x3C000000 – 0x3FFFFFFF	64M	UTOPIA queues† (C6415 only)
0x40000000 – 0x5FFFFFFF	512M	Reserved
0x60000000 – 0x63FFFFFF	64M	EMIFB CE0
0x64000000 – 0x67FFFFFF	64M	EMIFB CE1
0x68000000 – 0x6BFFFFFF	64M	EMIFB CE2
0x6C000000 – 0x6FFFFFFF	64M	EMIFB CE3
0x70000000 – 0x7FFFFFFF	256M	Reserved
0x80000000 – 0x8FFFFFFF	256M	EMIFA CE0
0x90000000 – 0x9FFFFFFF	256M	EMIFA CE1
0xA0000000 – 0xAFFFFFFF	256M	EMIFA CE2
0xB0000000 – 0xBFFFFFFF	256M	EMIFA CE3
0xC0000000 – 0xFFFFFFFF	1G	Reserved

† Address range is reserved for C6414.

11.3.6 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 11–6. When the boot configuration selects MAP 1, this memory is internal. When the device mode is in MAP 0, the memory is external. When external memory is selected, the boot configuration also determines the type of memory at the reset address. These options effectively provide alternative reset values to the appropriate EMIF control registers.

The C621x/C671x/C64x always has internal RAM at address 0, regardless of the boot configuration.

11.4 Boot Configuration

Several device settings are configured at reset to determine how the device operates. These settings include the boot configuration, the input clock mode, endian mode, and other device-specific configurations. For the C620x/C670x, the boot configuration is determined by the BOOTMODE[4:0] values. Table 11–6 lists all the values for BOOTMODE[4:0], as well as the associated memory maps and boot processes for C620x/C670x. For example the value 00000b on BOOTMODE[4:0] selects memory map 0; 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 11–6. TMS320C620x/C670x 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	Host boot (HPI/XBUS/PCI)
00111	MAP 1	Internal	Host boot (HPI/XBUS/PCI)
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 11–6. TMS320C620x/C670x Boot Configuration Summary

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	

See the following sections for details on the boot configuration of each of the C6000 devices.

11.4.1 TMS320C6201/C6701 Boot and Device Configuration

The C6201/C6701 latches the following configurations during device reset:

- ❑ **Boot Configuration:** The dedicated BOOTMODE[4:0] pins determine the device boot configurations as shown in Table 11–6.
- ❑ **Input Clock Mode:** The on-chip PLL frequency multiplier is configured through static CLKMODE input pins. Refer to the datasheet for details.
- ❑ **Endian Mode:** The LENDIAN input pin is used to configure the device to operate in either big endian (LENDIAN = 0) or little endian (LENDIAN = 1) mode.

11.4.2 TMS320C6202(B)/C6203(B)/C6204 Boot and Device Configuration

The input clock mode of the C6202(B)/C6202/C6204 is configured through the CLKMODE input pins at reset. Refer to the datasheet for details. The pull-up/pull-down resistors on the XBUS are used to determine the boot configuration (pins XD[4:0]) and other device configurations (pins XD[31:5]) at reset. The XD[4:0] lines directly map to BOOTMODE[4:0] described in Table 11–6. Reserved configuration fields should be pulled-down. Detailed descriptions of boot and device configurations are shown in Figure 11–1 and Table 11–7 .

Figure 11–1. TMSC6202(B)/C6203(B)/C6204 Boot and Device Configuration via Pull-Up/ Pull-Down Resistors on XD[31:0]

31	30 28	27	26 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

† All reserved fields should be pulled down.

Table 11–7. TMSC6202(B)/C6203(B)/C6204 Boot and Device Configuration Description

XD Bit	Field	Description

30:28
26:24
22:20
18:16
	MTYPE3
	MTYPE2
	MTYPE1
	MTYPE0
		Memory type:
		MTYPE=010b: 32-bit wide asynchronous interface
		MTYPE=101b: 32-bit wide FIFO interface
		MTYPE=other: reserved

13	BLPOL	Determines polarity of the XBLAST signal when the DSP is a slave on the XBUS.
		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.

12	RWPOL	Determines polarity of XBUS read/write signal.
		RWPOL=0: X$\overline{\text{R}}$/W. Write is active-high.
		RWPOL=1, XR/$\overline{\text{W}}$. Read is active-high.

11	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.

Table 11–7. TMSC6202(B)/C6203(B)/C6204 Boot and Device Configuration Description (Continued)

XD Bit	Field	Description

10	XARB	XBUS arbiter (status in XBGC):
		XARB = 0: Internal XBUS arbiter is disabled
		XARB = 1: Internal XBUS arbiter is enabled.

9	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 $\overline{\text{XCE3}}$ for FIFO read mode. XOE is disabled in all other XCE spaces.
8	LEND	Little endian mode: LEND = 0: system operates in big endian mode LEND = 1: system operates in little endian mode
4:0	BOOT-MODE	Dictates the boot-mode of the device. See Table 11–6.

11.4.3 TMS320C6205 Boot and Device Configuration

The C6205 CLKMODE0 input pin is used in conjunction with the EMIF data bus pins to determine the input clock mode at reset. For details refer to the datasheet. The pull-up/pull-down resistors on the EMIF data bus are also used to determine the boot mode selection (pins ED[4:0]), and other device configurations (pins ED[31:5]), at reset. The ED[4:0] lines directly map to BOOTMODE[4:0] described in Table 11–6. Reserved configuration fields should be pulled down. Detailed descriptions of boot and device configuration are shown in Figure 11–2 and Table 11–8. Refer to Chapter 9, *PCI*, for details on PCI boot.

Figure 11–2. TMS320C6205 Boot and Device Configuration via Pull-Up/Pull-Down Resistors on ED[31:0]

31	30–28	27	26–24	23	22–16
PLL_CONF2	Reserved	PLL_CONF1	Reserved	PLL_CONF0	Reserved†

15	14–9	8	7–5	4–0
EEAI	Reserved	LEND	EESZ	BOOTMODE

† All reserved fields should be pulled down.

Table 11–8. TMSC6205 Boot and Device Configuration Description

ED Bit	Field	Function
31	PLL_CONF [2]	On-chip PLL is enabled or disabled by CLKMODE0 pin
27	[1]	Case 1 When CLKMODE0 = 0 On-chip PLL bypass

23	[0]	Case 2 When CLKMODE0 = 1
		PLL_Conf = 000b, CPU clock = CLKIN x 1 (PLL Bypass)
		PLL_Conf = 001b, CPU clock = CLKIN x 4
		PLL_Conf = 010b, CPU clock = CLKIN x 8
		PLL_Conf = 011b, CPU clock = CLKIN x 10
		PLL_Conf = 100b, CPU clock = CLKIN x 6
		PLL_Conf = 101b, CPU clock = CLKIN x 9
		PLL_Conf = 110b, CPU clock = CLKIN x 7
		PLL_Conf = 111b, CPU clock = CLKIN x 11
		
		Note that CLKMODE0 acts as a PLL enable pin while the ED pins (pins 31, 27, and 23) choose the various PLL multiply options.

15	EEAI	EEPROM autoinitialization EEAI = 0: PCI uses default values EEAI = 1: Read configure value from EEPROM

8	LEND	LEND = 1: Little Endian
		LEND = 0: Big Endian

7:5	EESZ	EEPROM size selection (EEPROM is always 16-bit):
		
		EESZ = 000b No EEPROM
		EESZ = 001b 1K
		EESZ = 010b 2K
		EESZ = 011b 4K
		EESZ = 100b 16K
		EESZ = others Reserved

4:0	BOOTMODE	Dictates the boot-mode of the device. See Table 11–6.

11.4.4 TMSC621x/C671x Boot and Device Configuration

For the C621x/C671x, the pull-up/pull-down resistors on the host-port, along with the CLKMODE0 input pin, are used to determine the boot and device configurations at reset. The C6712 has dedicated configuration pins LENDIAN, BOOTMODE[1:0], and CLKMODE0.

- ❑ **Boot Configuration:** Pins HD[4:3] of the host-port are used to determine the boot configuration at reset. Only two of the five BOOTMODE bits are required because the C621x/C671x only has one memory map, which places internal memory at address 0. The HD[4:3] pins (or BOOTMODE[1:0] pins on C6712) map to the BOOTMODE[4:3] pins of the C620x/C670x. The complete boot configuration shown in Table 11–6 is significantly reduced for the C621x/C671x as shown in Table 11–9.
- ❑ **Input Clock Mode:** The on-chip PLL frequency multiplier is configured through the CLKMODE0 input pin. Refer to the datasheet for details on input clock mode.
- ❑ **Endian Mode**: The HD[8] pin of the host port (or LENDIAN pin on C6712) is latched at reset to configure the device to operate in either big endian (HD[8] = 0) or little endian (HD[8] =1) mode.

Table 11–9. TMS320C621x/C671x Boot Configuration Summary

HD[4:3] (C6211/C6711) BOOTMODE [1:0] (C6712)	C620x/C670x equivalent BOOTMODE[4:0]	Boot Process
00	00xxx	Host-port interface
01	01xxx	8-bit ROM with default timings
10	10xxx	16-bit ROM with default timings
11	11xxx	32-bit ROM with default timings

11.4.5 C64x Boot and Device Configuration

For the C64x, the following configurations are latched during device reset:

❑ **Input Clock Mode:** The on–chip PLL frequency multiplier is configured through CLKMODE input pins. Refer to the specific device datasheet for details (see *Related Documentation From Texas Instruments).*

❑ **Boot Configuration:** The pull up/down resistors on the EMIFB address bus, pin BEA[19:18], determine the boot configuration, as shown in Table 11–10. Refer to the specific device datasheet for the internal pull up/down resistors (See *Related Documentation From Texas Instruments).*

❑ **Device Configuration:** The following sections provide detailed descriptions on device–specific configurations.

Table 11–10. TMS320C64x Boot Configuration Summary

BOOTMODE	Boot Process
00	None
01	Host Boot†
10	EMIFB 8–bit ROM with default timings
11	Reserved

† For C6414, HPI is used for host boot.
For C6415, HPI is used for host boot if PCI_EN = 0, and PCI is used for host boot if PCI_EN = 1.

11.4.5.1 C6414 Device Configuration

The C6414 device configurations are determined by the pull up/down resistor on the HPI data pin HD5, along with the pull up/down resistors on the EMIFB address bus (BEA[20:1]) latched at device reset.

HPI Bus Width (HD5 pin)

The pull–up/down resistors on the HD5 pin determines the HPI bus width (HPI_WIDTH) as follows:

❑ HPI_WIDTH = 0: The HPI bus width is 16 bits. HPI operates in HPI16 mode.

❑ HPI_WIDTH = 1: The HPI bus width is 32 bits. HPI operates in HPI32 mode.

Other Device Configurations (BEA[20:1])

Table 11–13 and Table 11–11 show the rest of the C6414 device configurations through the EMIFB address bus pull up/down resistors.

Figure 11–3. TMS320C6414 Boot and Device Configuration via Pull–Up/Pull–Down Resistors on BEA[20:1]

20	19 18	17 16	15 14	13 1
LEND	BOOTMODE	AECLKIN_SEL	BECLKIN_SEL	Reserved†

†All reserved fields should be pulled down.

Table 11–11. TMS320C6414 Boot and Device Configuration Description

BEA Bit	Field	Description
20	LEND	Little endian mode LEND = 0: system operates in big endian mode LEND = 1: system operates in little endian mode
19:18	BOOTMODE	Dictates the bootmode of the device. See Table 11–10.

17:16	AECLKIN_SEL	EMIFA input clock select
		AECLKIN_SEL = 00b: EMIFA runs at AECLKIN rate.
		AECKLIN must be provided by system.
		AECLKIN_SEL = 01b: EMIFA runs at 1/4th of CPU clock rate
		AECLKIN_SEL = 10b: EMIFA runs at 1/6th of CPU clock rate
		AECLKIN_SEL = 11b: Reserved

15:14	BECLKIN_SEL	EMIFB input clock select
		BECLKIN_SEL = 00b: EMIFB runs at BECLKIN rate.
		BECKLIN must be provided by system.
		BECLKIN_SEL = 01b: EMIFB runs at 1/4th of CPU clock rate
		BECLKIN_SEL = 10b: EMIFB runs at 1/6th of CPU clock rate
		BECLKIN_SEL = 11b: Reserved

11.4.5.2 C6415 Device Configuration

The C6415 device configurations are determined by the PCI_EN pin, the MCBSP2_EN pin, the pull up/down resistor on the HPI data pin HD5, and the pull up/down resistors on the EMIFB address bus (BEA[20:1]) latched at device reset.

Peripheral Selection (PCI_EN, MCBSP2_EN, UTOPIA_EN)

Some peripherals on the C6415 are mutually exclusive. Users can select the desired peripherals via the following device configuration pins at reset:

- PCI Enable (PCI_EN)
- McBSP2 Enable (MCBSP2_EN)
- UTOPIA Enable (UTOPIA_EN field at BEA[11])

Table 11–12 summarizes the selection between HPI, PCI, McBSP2, and GPIO (pins 8 to 15) based on PCI_EN and MCBSP2_EN. Table 11–13 summarizes the selection between UTOPIA and McBSP1 via the pull-up/down resistor on UTOPIA_EN (BEA[11]) at reset.

Table 11–12. TMS320C6415 HPI, PCI, McBSP2, and GPIO Selection

PCI_EN	MCBSP2_EN	Peripherals Selected			
		HPI	PCI	McBSP2 †	GPIO †
0	0	Yes		Yes	GPIO Pins 0 to 15
0	1	Yes		Yes	GPIO Pins 0 to 15
1	0		Yes		GPIO Pins 0 to 8 only (No GP[9:15])
1	1		Yes. No EEPROM	Yes	GPIO Pins 0 to 8 only (No GP[9:15])

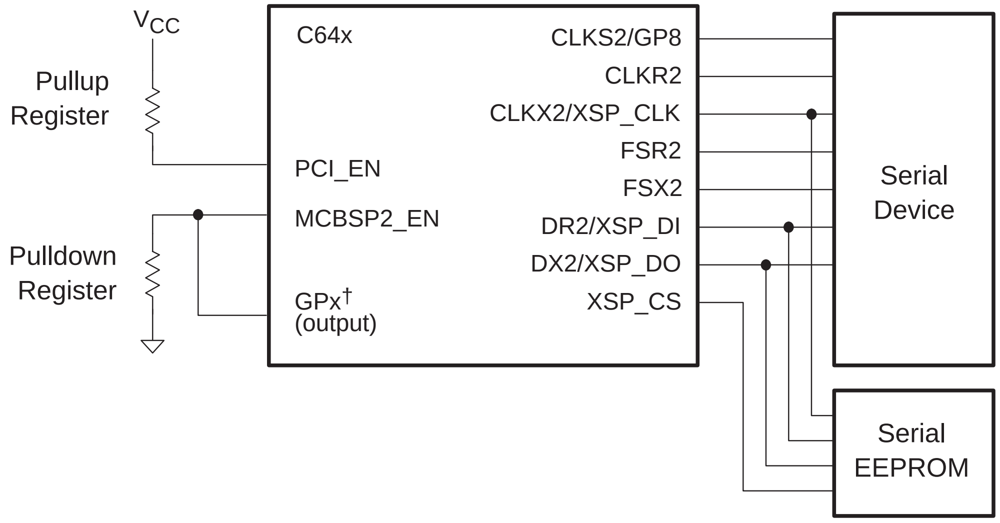
† GPIO pin GP8 is MUXed with McBSP2 external clock source pin CLKS2. To use this pin as GP8, bit 8 in the GPIO Enable Register (GPEN) must be set to 1. To use this pin as CLKS2, bit 8 in the GPEN must be set to 0.

Table 11–13. TMS320C6415 UTOPIA and McBSP1 Selection

UTOPIA_EN (BEA[11])	Peripherals Selected	
	UTOPIA	McBSP1
0		Yes
1	Yes	

UTOPIA_EN is latched at reset only. The PCI_EN pin must be driven valid (pulled up/down) at all times, and the user must not switch its value throughout device operation. MCBSP2_EN must be driven valid at all times, but the user can change its value dynamically after device reset. For example, at reset, the user can initialize PCI via the serial EEPROM by setting PCI_EN = 1 and MCBSP2_EN = 0. Upon exiting reset, the serial EEPROM may not be needed. If not, the user can change the MCBSP2_EN input to 1 to disable the serial EEPROM interface and enable the McBSP2 interface. Figure 11–4 shows an example of this interface. One of the GPIO pins (GP0 to GP7) is tied to the MCBSP2_EN configuration pin. At reset, PCI_EN is pulled up to 1, and MCBSP2_EN is pulled down to 0 to enable the PCI EEPROM interface. Upon exiting reset, configure the GPx as an output, and write a 1 to MCBSP2_EN to enable the McBSP2 interface.

Figure 11–4. McBSP2/EEPROM Selection Interface



Note: After reset, use one of the general purpose pins (GP0 to GP7) to set McBSP2_EN = 1 to enable the McBSP2 interface, and disable the EEPROM interface.

HPI Bus Width (HD5 pin)

If HPI operation is selected (PCI_EN = 0), a pull–up/down resistor on the HD5 pin determines the HPI bus width (HPI_WIDTH) as follows:

- HPI_WIDTH = 0: The HPI bus width is 16 bits. HPI operates in HPI16 mode.
- HPI_WIDTH = 1: The HPI bus width is 32 bits. HPI operates in HPI32 mode.

HPI_WIDTH is “don’t care” if PCI operation is selected instead (PCI_EN = 1).

Other Device Configurations (BEA[20:1])

Figure 11–5 and Table 11–14 show the rest of the C6415 device configurations through the EMIFB address bus pull up/down resistors.

Figure 11–5. TMS320C6415 Boot and Device Configuration via Pull–Up/Pull–Down Resistors on BEA[20:1]

20	19 18	17 16	15 14	13	12	11	10 1
LEND	BOOTMODE	AECLKIN_SEL	BECLKIN_SEL	EEAI	rsvd	UTOPIA_EN	Reserved†
							0001000000

† **CAUTION:** To operate correctly, all reserved fields must be pulled up/down as shown: BEA7 must be pulled up, and all other reserved bits (BEA12, BEA[10:8], BEA[6:1]) must be pulled down.

Table 11–14. TMS320C6415 Boot and Device Configuration Description

BEA Bit	**Field**	**Description**
20	LEND	Little endian mode LEND = 0: system operates in big endian mode LEND = 1: system operates in little endian mode
19:18	BOOTMODE	Dictates the bootmode of the device. See Table 10–10.

17:16	AECLKIN_SEL	EMIFA input clock select
		AECLKIN_SEL = 00b: EMIFA runs at AECLKIN rate.
		AECKLIN must be provided by system.
		AECLKIN_SEL = 01b: EMIFA runs at 1/4th of CPU clock rate
		AECLKIN_SEL = 10b: EMIFA runs at 1/6th of CPU clock rate
		AECLKIN_SEL = 11b: Reserved

15:14	BECLKIN_SEL	EMIFB input clock select
		BECLKIN_SEL = 00b: EMIFB runs at BECLKIN rate.
		BECKLIN must be provided by system.
		BECLKIN_SEL = 01b: EMIFB runs at 1/4th of CPU clock rate
		BECLKIN_SEL = 10b: EMIFB runs at 1/6th of CPU clock rate
		BECLKIN_SEL = 11b: Reserved

13	EEAI	EEPROM Autoinitialization EEAI = 0: PCI uses default values. EEAI must be set to 0 if PCI_EN is disabled (PCI_EN = 0). EEAI = 1: Read configure value from 4K EEPROM. MCBSP2_EN must be 0 to enable EEPROM operation

11	UTOPIA_EN	UTOPIA Enable
		The UTOPIA and McBSP1 peripherals on the C6415 are mutually exclusive. This field enables either UTOPIA or McBSP1.
		UTOPIA_EN = 0: UTOPIA disabled. McBSP1 enabled.
		UTOPIA_EN = 1: UTOPIA enabled. McBSP1 disabled.

11.5 Boot Processes

The boot process is determined by the boot configuration selected, as described in section 11.4. 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 C621x/C671x.

  Note: Operation is undefined if invalid code is located at address.

❑ **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. On C62x/C67x, 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. For the C620x/C670x, these values are expected to be stored in little endian format in the external memory, typically a ROM device. For the C621x/C671x, the data should be stored in the endian format that the system is using (either big or little endian). The C64x only supports 8 bit ROM boot, where data should be stored in the endian format that the system is using.

  The transfer is automatically done by the DMA/EDMA as a single-frame block transfer from the ROM to 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.

  - **C620x/C670x:** The DMA copies 64K bytes from CE1 to address 0, using default ROM timings. After the transfer the CPU begins executing from address 0.
  - **C621x/C671x/C64x:** The EDMA copies 1K bytes from the beginning of CE1 (EMIFB CE1 on C64x) 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 internal configuration registers, such as those that control the EMIF or other peripherals. 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 host 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, depending on the host interface peripheral that is available on the device. Refer to the device datasheet for the specific peripheral set. One of the following host interfaces is used for host boot:

- **Host Port Interface:** For devices with HPI, the HPI can be used for host boot. The HPI is always a slave interface, and needs no special configuration.
- **Expansion Bus:** For devices with XBUS, the XBUS can be used for the host boot. The type of host interface is determined by a set of latched signals during rest.
- **PCI:** For devices with PCI, the PCI can be used for the host boot.

Chapter 12

Multichannel Buffered Serial Port

This chapter describes the operation and hardware of the multichannel buffered serial port (McBSP). It also includes register definitions and timing diagrams for the McBSP.

Topic **Page**

12.1 Features

The multichannel buffered serial port (McBSP) for the TMS320C6000™ is based on the standard serial port interface used on the TMS320C2x, C3x, C5x, and C54x™ devices. The McBSP provides these functions:

- 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 analog-to-digital (A/D) and digital-to-analog (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.

All C6000 devices have the same McBSP. However, the C621x/C671x and C64x McBSP has additional features and enhancements that are summarized in Table 12–1.

Table 12–1. Enhanced Features on TMS320C6000 McBSP

Features	Supported on Device	Section
DXENA DX Enabler	C621x, C671x, C64x	12.6.5
RWDREVRS/ XWDREVRS 32-bit data reversal	C621x, C671x, C64x	12.3.9
RMCME/XMCME Enhanced multichannel selection mode	C64x	12.6.2, 12.6.4
FREE, SOFT Emulation control	C621x, C671x, C64x	12.2.1

12.2 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 12–1. Seven pins, listed in Table 12–2, connect the control and data paths to external devices.

Figure 12–1. McBSP Block Diagram

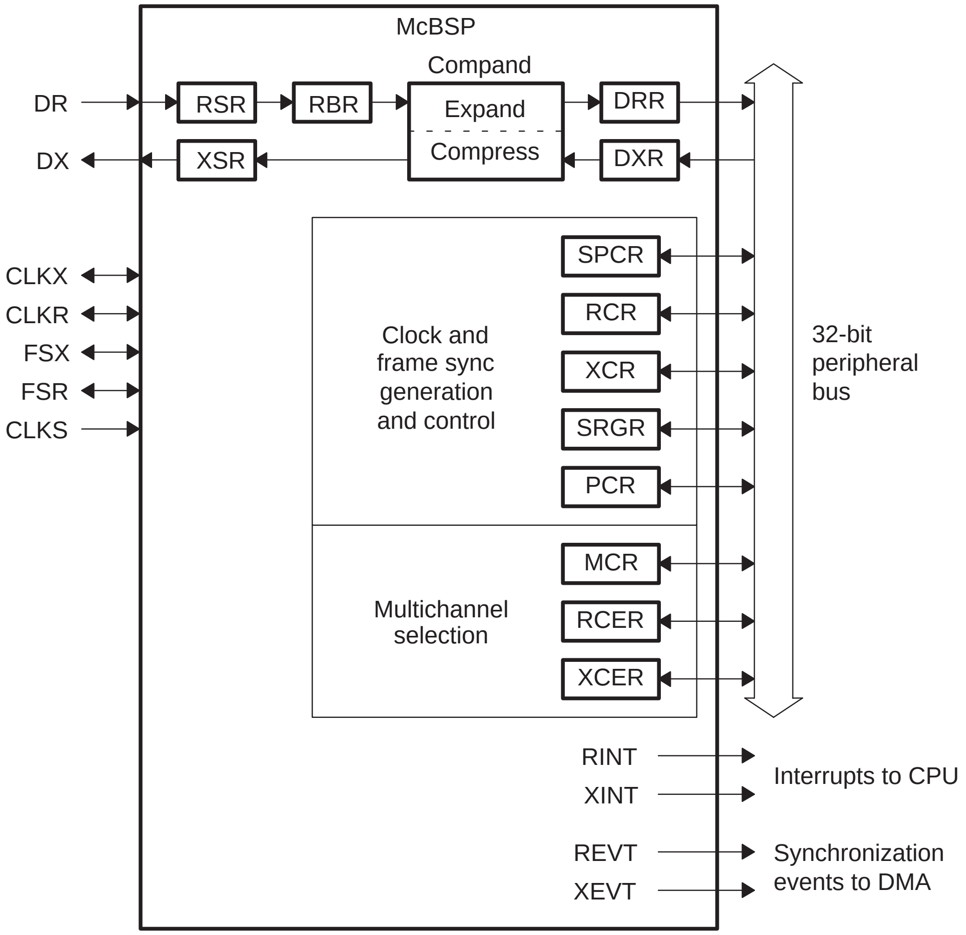


Table 12–2. McBSP Interface Signals

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

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 C6000 communicates to the McBSP via 32-bit-wide control registers accessible via the internal peripheral bus. Either the CPU or the DMA/EDMA 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/EDMA controller. This allows simultaneous internal data movement and external data communications.

The McBSP registers are listed in Table 12–3. 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/EDMA controller via the four signals shown in Table 12–4.

For devices with the EDMA peripheral (TMS320C621x/C671x/C64x), the data receive and transmit registers (DRR and DXR) are also mapped to memory locations 3xxxxxxxh shown in Table 12–5. The DRR and DXR locations 018Cxxxxh/0190xxxxh/01A4xxxxh are accessible via the peripheral bus, while the 3xxxxxxxh locations are accessible via the EDMA bus. Both the CPU and the EDMA in these devices can access the DRR and DXR in all the memory-mapped locations shown in Table 12–5. An access to *any* EDMA bus location in Table 12–5 is equivalent to an access to the DRR/DXR of the corresponding McBSP. For example, a read from any location in 30000000h–33FFFFFFh is equivalent to a read from the DRR of McBSP0 at 018C0000h. A write to any location in 30000000h–33FFFFFFh is equivalent to a write to the DXR of McBSP0 at 018C0004h. The user has a choice of reading from/writing to the DRR and DXR in either the 3xxxxxxxh or the

018Cxxxxh/0190xxxxh/01A4xxxxh location. It is recommended that the user 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 accessible only via the peripheral bus, and thus are mapped only to the 018Cxxxxh/0190xxxxh/01A4xxxxh locations.

Table 12–3. McBSP Registers

Hex Byte Address					
McBSP 0	**McBSP 1**	**McBSP 2**§	**Abbreviation**	**McBSP Register Name**	**Section**
–	–	–	RBR	Receive buffer register†	12.2
–	–	–	RSR	Receive shift register†	12.2
–	–	–	XSR	Transmit shift register†	12.2
018C 0000	0190 0000	01A4 0000	DRR	Data receive register‡¶	12.2
018C 0004	0190 0004	01A4 0004	DXR	Data transmit register¶	12.2
018C 0008	0190 0008	01A4 0008	SPCR	Serial port control register	12.2.1
018C 000C	0190 000C	01A4 000C	RCR	Receive control register	12.2.2
018C 0010	0190 0010	01A4 0010	XCR	Transmit control register	12.2.2
018C 0014	0190 0014	01A4 0014	SRGR	Sample rate generator register	12.5.1.1
018C 0018	0190 0018	01A4 0018	MCR	Multichannel control register	12.6.1
018C 001C	0190 001C	01A4 001C	RCER RCERE0	Receive channel enable register Enhanced receive channel enable register 0||	12.6.3.1 12.6.4.1
018C 0020	0190 0020	01A4 0020	XCER XCERE0	Transmit channel enable register Enhanced transmit channel enable register 0||	12.6.3.1 12.6.4.1
018C 0024	0190 0024	01A4 0024	PCR	Pin control register	12.2.1
018C 0028	0190 0028	01A4 0028	RCERE1	Enhanced receive channel enable register 1☆	12.6.4.1
018C 002C	0190 002C	01A4 002C	XCERE1	Enhanced transmit channel enable register 1☆	12.6.4.1
018C 0030	0190 0030	01A4 0030	RCERE2	Enhanced receive channel enable register 2☆	12.6.4.1
018C 0034	0190 0034	01A4 0034	XCERE2	Enhanced transmit channel enable register 2☆	12.6.4.1
018C 0038	0190 0038	01A4 0038	RCERE3	Enhanced receive channel enable register 3☆	12.6.4.1
018C 003C	0190 003C	01A4 003C	XCERE3	Enhanced transmit channel enable register 3☆	12.6.4.1

† The RBR, RSR, and XSR are not directly accessible via the CPU or the DMA/EDMA controller.
‡ The CPU and DMA/EDMA controller can only read this register; they cannot write to it.
§ Applicable only to C6202(B), C6203(B), and C64x.
¶ For TMS320C621x/C671x/C64x, the DRR and DXR are also mapped at addresses shown in Table 12–5.
|| For C64x, enhanced multichannel selection mode only, RCER and XCER are replaced by RCERE0 and XCERE0, respectively.
☆ RCERE1, XCERE1, RCERE2, XCERE2, RCERE3, and XCERE3 are available on C64x only for enhanced multichannel selection mode.

Table 12–4. McBSP CPU Interrupts and DMA Synchronization Events

Interrupt Name	Description	Section
RINT	Receive interrupt to CPU	12.3.3
XINT	Transmit interrupt to CPU	12.3.3
REVT	Receive synchronization event to the DMA/ EDMA controller	12.3.2.1
XEVT	Transmit synchronization event to the DMA/ EDMA controller	12.3.2.2

Table 12–5. TMS320C621x/C671x/C64x Data Receive and Transmit Registers (DRR/DXR) Mapping

		Accessible Via	
Serial Port	Register	Peripheral Bus	EDMA Bus
McBSP 0	DRR	0x018C0000	0x30000000–0x33FFFFFF
	DXR	0x018C0004	0x30000000–0x33FFFFFF
McBSP 1	DRR	0x01900000	0x34000000–0x37FFFFFF
	DXR	0x01900004	0x34000000–0x37FFFFFF
McBSP 2†	DRR	0x01A40000	0x38000000—x3BFFFFFF
	DXR	0x01A40004	0x38000000—x3BFFFFFF

† Applicable only to C64x.

12.2.1 Serial Port Configuration

The serial port is configured via the serial port control register (SPCR) and the pin control register (PCR) shown in Figure 12–2 and Figure 12–3, respectively. The SPCR and PCR contain McBSP status control bits. Table 12–6 and Table 12–7 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 12.8).

Figure 12–2. Serial Port Control Register (SPCR)

31–26	25	24	23	22	21–20	19	18	17	16
reserved	FREE[‡]	SOFT[‡]	$\overline{\text{FRST}}$	$\overline{\text{GRST}}$	XINTM	XSYNCERR[†]	$\overline{\text{XEMPTY}}$	XRDY	$\overline{\text{XRST}}$
R, +0	RW, +0	RW, +0	RW, +0	RW, +0	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

[†] Writing a 1 to XSYNCERR or RSYNCERR will set the error condition when the transmiter or receiver ($\overline{\text{XRST}}$=1 or $\overline{\text{RRST}}$=1), respectively, are enabled. Thus, it is used mainly for testing purposes or if this operation is desired.
[‡] Available in the C621x/C671x/C64x only.

Table 12–6. Serial Port Control Register (SPCR) Field Descriptions

Bit No.	Name	Function	Section
25	FREE	Serial clock free running mode (C621x/C671x/C64x only) This bit is used in conjunction with SOFT bit to determine state of serial port clock during emulation halt. FREE = 0: During emulation halt, SOFT bit determines operation of McBSP. FREE = 1: During emulation halt, serial clocks continue to run.	12.2.1
24	SOFT	Serial clock emulation mode (C621x/C671x/C64x only) This bit is used in conjunction with FREE bit to determine state of serial port clock during emulation halt. This bit has no effect if FREE=1. SOFT = 0: In conjunction with FREE = 0, serial port clock stops immediately during emulation halt, thus aborting any transmissions. SOFT = 1: In conjunction with FREE = 0, during emulation halt, serial port clock stops after completion of current transmission.	12.2.1
23	$\overline{\text{FRST}}$	Frame sync generator reset $\overline{\text{FRST}}$ = 0: Frame sync generation logic is reset. Frame sync signal is not generated by 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.	12.5.3
22	$\overline{\text{GRST}}$	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 programmed values in sample rate generator register (SRGR).	12.5.1.2

Table 12–6. Serial Port Control Register (SPCR) Field Descriptions (Continued)

Bit No.	Name	Function	Section
21–20	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	12.3.3
19	XSYNCERR	Transmit synchronization error XSYNCERR = 0: No frame synchronization error XSYNCERR = 1: Frame synchronization error detected by McBSP	12.3.7.2 12.3.7.5
18	$\overline{\text{XEMPTY}}$	Transmit shift register (XSR) empty $\overline{\text{XEMPTY}}$ = 0: XSR is empty. $\overline{\text{XEMPTY}}$ = 1: XSR is not empty.	12.3.7.4
17	XRDY	Transmitter ready XRDY = 0: Transmitter is not ready. XRDY = 1: Transmitter is ready for data to be written to DXR.	12.3.2
16	$\overline{\text{XRST}}$	Transmitter reset. This resets or enables transmitter. $\overline{\text{XRST}}$ = 0: Serial port transmitter is disabled and is in reset state. $\overline{\text{XRST}}$ = 1: Serial port transmitter is enabled.	12.3.1
15	DLB	Digital loopback mode DLB = 0: Digital loopback mode disabled DLB = 1: Digital loopback mode enabled	12.5.2.5 12.5.2.6 12.5.3.2

14–13	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
			12.3.8

Table 12–6. Serial Port Control Register (SPCR) Field Descriptions (Continued)

Bit No.	Name	Function	Section
12–11	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.	12.7

7	DXENA	DX Enabler – applicable only for C621x/C671x/C64x device. Enable extra delay for DX turn-on time. This bit controls Hi-Z enable on DX pin, not the data itself, so only first bit of data is delayed.
		DXENA = 0: DX enabler is off.
		DXENA = 1: DX enabler is on.
			12.6.5

5–4	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	12.3.3
3	RSYNCERR	Receive synchronization error RSYNCERR = 0: No frame synchronization error RSYNCERR = 1: Frame synchronization error detected by McBSP	12.3.7.2 12.3.7.5
2	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.	12.3.7.1

Table 12–6. Serial Port Control Register (SPCR) Field Descriptions (Continued)

Bit No.	Name	Function	Section
1	RRDY	Receiver ready RRDY = 0: Receiver is not ready. RRDY = 1: Receiver is ready with data to be read from DRR.	12.3.2
0	$\overline{\text{RRST}}$	Receiver reset. This resets or enables receiver. $\overline{\text{RRST}}$ = 0: Serial port receiver is disabled and is in reset state. $\overline{\text{RRST}}$ = 1: Serial port receiver is enabled.	12.3.1

Figure 12–3. 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 12–7. Pin Control Register (PCR) Field Descriptions

Bit No.	Name	Function	Section
13	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.	12.8
12	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.	12.8

Table 12–7. Pin Control Register (PCR) Field Descriptions (Continued)

Bit No.	Name	Function	Section
11	FSXM	Transmit frame synchronization mode	12.5.3.3
		FSXM = 0: Frame synchronization signal is provided by an external source. FSX is an input pin.	12.8
		FSXM = 1: Frame synchronization generation is determined by the sample rate generator frame synchronization mode bit FSGM in the SRGR.	
10	FSRM	Receive frame synchronization mode	12.5.3.2
		FSRM = 0: Frame synchronization signals are generated by an external device. FSR is an input pin.	12.8
		FSRM = 1: Frame synchronization signals are generated internally by the sample rate generator. FSR is an output pin except when GSYNC = 1 (see section 12.5.1.1) in SRGR.	
9	CLKXM	Transmitter clock mode	12.5.2.7
		CLKXM = 0: Transmitter clock is driven by an external clock with CLKX as an input pin.	12.8
		CLKXM = 1: CLKX is an output pin and is driven by the internal sample rate generator.	
		During SPI mode (CLKSTP in SPCR is a nonzero value):	12.7
		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.	
8	CLKRM	Receiver clock mode	12.5.2.6
		Case 1: Digital loopback mode not set (DLB = 0) in SPCR	12.8
		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.	

Table 12–7. Pin Control Register (PCR) Field Descriptions (Continued)

Bit No.	Name	Function	Section
6	CLKS_STAT	CLKS pin status. Reflects the value on the CLKS pin when selected as a general-purpose input.	12.8
5	DX_STAT	DX pin status. Reflects the value driven onto the DX pin when selected as a general-purpose output.	12.8
4	DR_STAT	DR pin status. Reflects the value on the DR pin when selected as a general-purpose input.	12.8

3	FSXP	Transmit frame synchronization polarity
		FSXP = 0: Frame synchronization pulse FSX is active high
		FSXP = 1: Frame synchronization pulse FSX is active low
			12.3.4.1
			12.8

2	FSRP	Receive frame synchronization polarity
		FSRP = 0: Frame synchronization pulse FSR is active high
		FSRP = 1: Frame synchronization pulse FSR is active low
			12.3.4.1
			12.8

1	CLKXP	Transmit clock polarity CLKXP = 0: Transmit data driven on rising edge of CLKX CLKXP = 1: Transmit data driven on falling edge of CLKX	12.3.4.1 12.8
0	CLKRP	Receive clock polarity CLKRP = 0: Receive data sampled on falling edge of CLKR CLKRP = 1: Receive data sampled on rising edge of CLKR	12.3.4.1 12.8

12.2.2 Receive and Transmit Control Registers: RCR and XCR

The receive control register (RCR), shown in Figure 12–4, configures parameters of the receive operations. The RCR fields are summarized in Table 12–8.

12.2.2.1 Receive Control Register: RCR

Figure 12–4. 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	3–0
Reserved	RFRLEN1	RWDLEN1	RWDREVRS†	Reserved
R, +0	RW, +0	RW, +0	RW, +0	R, +0

† RWDREVRS 32-bit reversal feature is applicable only to the C621x/C671x/C64x device. For all other C6000 devices, it is reserved (R, +0).

Table 12–8. Receive Control Register (RCR) Field Descriptions

Bit No.	Name	Function	Section

31	RPHASE	Receive phases
		RPHASE = 0: Single phase frame
		RPHASE = 1: Dual phase frame
			12.3.4.2

30–24	RFRLEN2	Receive frame length in phase 2 RFRLEN2 = 000 0000b: 1 word per phase RFRLEN2 = 000 0001b: 2 words per phase • • • RFRLEN2 = 111 1111b: 128 words per phase	12.3.4.3

23–21	RWDLEN2	Receive element length in phase 2
		RWDLEN2 = 000b: 8 bits
		RWDLEN2 = 001b: 12 bits
		RWDLEN2 = 010b: 16 bits
		RWDLEN2 = 011b: 20 bits
		RWDLEN2 = 100b: 24 bits
		RWDLEN2 = 101b: 32 bits
		RWDLEN2 = 11Xb: Reserved
			12.3.4.4

20–19	RCOMPAND	Receive companding mode. RCOMPAND = 00b: No companding. Data transfer starts with MSB first. RCOMPAND = 01b: No companding, 8-bit data. Transfer starts with LSB first. Applicable to 8-bit data (RWDLEN=000b), or 32-bit data in data reversal mode. RCOMPAND = 10b: Compand using µ-law for receive data. Applicable to 8-bit data only (RWDLEN=000b). RCOMPAND = 11b: Compand using A-law for receive data. Applicable to 8-bit data only (RWDLEN=000b).	12.3.9 12.4
18	RFIG	Receive frame ignore RFIG = 0: Unexpected receive frame synchronization pulses restart the transfer. RFIG = 1: Unexpected receive frame synchronization pulses are ignored.	12.3.6.1

17–16	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
			12.3.4.6

Table 12–8. Receive Control Register (RCR) Field Descriptions (Continued)

Bit No.	Name	Function	Section
14–8	RFRLEN1	Receive frame length in phase 1 RFRLEN1 = 000 0000b: 1 word per phase RFRLEN1 = 000 0001b: 2 words per phase • • • RFRLEN1 = 111 1111b: 128 words per phase	12.3.4.3

7–5	RWDLEN1	Receive element length in phase 1
		RWDLEN1 = 000b: 8 bits
		RWDLEN1 = 001b: 12 bits
		RWDLEN1 = 010b: 16 bits
		RWDLEN1 = 011b: 20 bits
		RWDLEN1 = 100b: 24 bits
		RWDLEN1 = 101b: 32 bits
		RWDLEN1 = 11Xb: Reserved
			12.3.4.4

4	RWDREVRS	Receive 32-bit bit reversal feature. (C621x/C671x/C64x only). 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; RCOMPAND should be set to 01b; else operation is undefined.	12.3.9

12.2.2.2 Transmit Control Register: XCR

The transmit control register (XCR), shown in Figure 12–5, configures parameters of the transmit operations. The XCR fields are summarized in Table 12–9.

Figure 12–5. 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
Reserved	XFRLEN1	XWDLEN1	XWDREVRS†	Reserved
R, +0	RW, +0	RW, +0	RW, +0	R, +0

† XWDREVRS 32-bit reversal feature is applicable only to the C621x/C671x/C64x device. For all other C6000 devices, it is reserved (R, +0).

Table 12–9. Transmit Control Register (XCR) Field Descriptions

Bit No.	Name	Function	Section

31	XPHASE	Transmit phases
		XPHASE = 0: Single phase frame
		XPHASE = 1: Dual phase frame
			12.3.4.2

30–24	XFRLEN2	Transmit frame length in phase 2
		XFRLEN2 = 000 0000b: 1 word per phase
		XFRLEN2 = 000 0001b: 2 words per phase
		•
		•
		XFRLEN2= 111 1111b: 128 words per phase
			12.3.4.3

23–21	XWDLEN2	Transmit element length in phase 2
		XWDLEN2 = 000b: 8 bits
		XWDLEN2 = 001b: 12 bits
		XWDLEN2 = 010b: 16 bits
		XWDLEN2 = 011b: 20 bits
		XWDLEN2 = 100b: 24 bits
		XWDLEN2 = 101b: 32 bits
		XWDLEN2 = 11Xb: Reserved
			12.3.4.4

20–19	XCOMPAND	Transmit companding mode. XCOMPAND = 00b: No companding. Data transfer starts with MSB first. XCOMPAND = 01b: No companding, 8-bit data. Transfer starts with LSB first. Applicable to 8-bit data (RWDLEN=000b), or 32-bit data in data reversal mode. XCOMPAND = 10b: Compand using µ-law for receive data. Applicable to 8-bit data only (RWDLEN=000b). XCOMPAND = 11b: Compand using A-law for receive data. Applicable to 8-bit data only (RWDLEN=000b).	12.3.9 12.4

Table 12–9. Transmit Control Register (XCR) Field Descriptions (Continued)

Bit No.	Name	Function	Section
18	XFIG	Transmit frame ignore XFIG = 0: Unexpected transmit frame synchronization pulses restart the transfer. XFIG = 1: Unexpected transmit frame synchronization pulses are ignored.	12.3.6.1

17–16	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
			12.3.4.6

14–8	XFRLEN1	Transmit frame length in phase 1 XFRLEN1 = 000 0000b: 1 word per phase XFRLEN1 = 000 0001b: 2 words per phase • • • XFRLEN1 = 111 1111b: 128 words per phase	12.3.4.3

7–5	XWDLEN1	Transmit element length in phase 1
		XWDLEN1 = 000b: 8 bits
		XWDLEN1 = 001b: 12 bits
		XWDLEN1 = 010b: 16 bits
		XWDLEN1 = 011b: 20 bits
		XWDLEN1 = 100b: 24 bits
		XWDLEN1 = 101b: 32 bits
		XWDLEN1 = 11Xb: Reserved
			12.3.4.4

4	XWDREVRS	Transmit 32-bit bit reversal feature (C621/C671/C64x only). 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; XCOMPAND should be set to 01b; else operation is undefined.	12.3.9

12.3 Data Transmission and Reception

As shown in Figure 12–1 on page 12-4, 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/EDMA controller.

Transmit data is written by the CPU or the DMA/EDMA 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.

12.3.1 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 12–10 shows the state of the McBSP pins when the serial port is reset by these methods.

Table 12–10. 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. See section 12.5.1 for more information on sample rate generator and its default at reset. When the device is pulled out of reset, the serial port remains in the reset condition ($\overline{\text{RRST}}$ = $\overline{\text{XRST}}$ = $\overline{\text{FRST}}$ = $\overline{\text{GRST}}$ = 0). In this reset condition, the serial port pins can be used as general-purpose I/O (see section 12.8).
- **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. See section 12.5.1.2 for details on the McBSP and sample-rate generator reset and initialization procedure.

12.3.2 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 or EDMA controller (REVT and XEVT)
- Using the interrupts to the CPU (RINT and XINT) that the events generate.

Note: Reading the DRR and writing to DXR affects RRDY and XRDY, respectively.

12.3.2.1 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/EDMA controller. Once that data has been read by either the CPU or the DMA/EDMA 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/EDMA 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.

12.3.2.2 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/EDMA 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/EDMA 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/EDMA controller (via XEVT). Also, the transmit interrupt (XINT) to the CPU can be driven by XRDY if XINTM = 00b (default value) in the SPCR.

12.3.3 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 12.6.3.3 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 12.5.3.4 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 12.3.7.2 and 12.3.7.5 for more details on synchronization error.

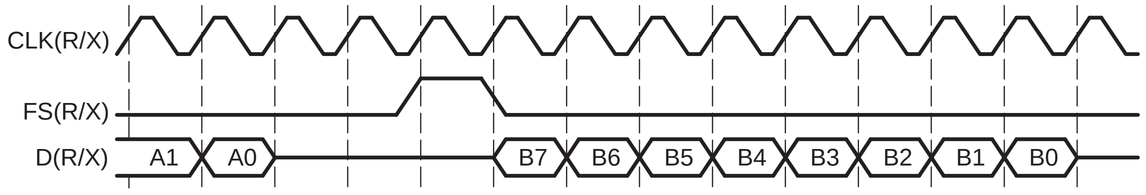
12.3.4 Frame and Clock Configuration

Figure 12–6 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 12–6. Frame and Clock Operation



12.3.4.1 Frame and Clock Operation

Receive and transmit frame sync pulses (FSR/X), and clocks (CLKR/X), can either be generated internally by the sample rate generator (see section 12.5.1) or be driven by an external input. The source of frame sync and clock are selected by programming the mode bits, FS(R/X)M and CLK(R/X)M respectively, in the PCR. FSR is also affected by the GSYNC bit in the SRGR (see section 12.5.3.2 for details).

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 12–36 on page 12-47). 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 12.3.4.6 for more information.

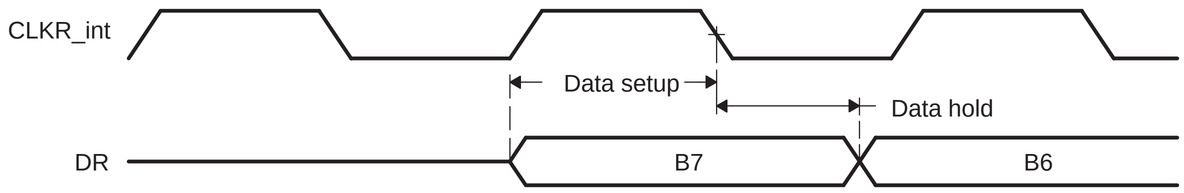
FSRP, FSXP, CLKRP, and CLKXP configure the polarities of FSR, FSX, CLKR, and CLKX, as indicated in Table 12–7. All frame sync signals (FSR_int and FSX_int) internal to the serial port are active high. If the serial port is configured for external frame synchronization (FSR/FSX are inputs to the McBSP) 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 12–36 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 12–7 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 12–7. Receive Data Clocking



12.3.4.2 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 12–8 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 12–11 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 12–8. Dual-Phase Frame Example

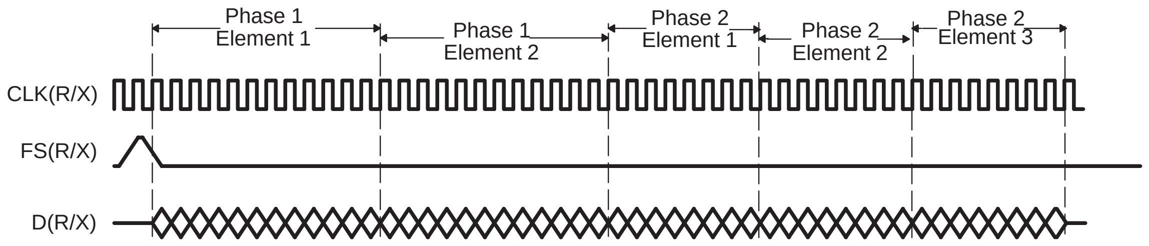


Table 12–11. RCR/XCR Fields Controlling Elements per Frame and Bits per Element

Serial Port McBSP0/1	Frame Phase	RCR/XCR field Control	
		Elements per Frame	**Bits per Element**
Receive	1	RFRLEN1	RWDLEN1
Receive	2	RFRLEN2	RWDLEN2
Transmit	1	XFRLEN1	XWDLEN1
Transmit	2	XFRLEN2	XWDLEN2

12.3.4.3 Frame Length: (R/X)FRLEN(1/2)

Frame length specifies the maximum number of serial elements or logical time slots or channels that are available for transfer per frame synchronization signal. In multichannel selection mode, the frame length value is independent of (and perhaps different from) the actual number of channels that the DSP is programmed to receive or transmit per frame via the MCR, RCER, and XCER registers. See section 12.6 for details on multichannel selection mode operation. The 7-bit (R/X)FRLEN(1/2) field in the (R/X)CR supports up to 128 elements per phase in a frame, as shown in Table 12–12. (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 12–8, (R/X)FRLEN1 = 1 or 0000001b and (R/X)FRLEN2 = 2 or 0000010b.

Table 12–12. McBSP Receive/Transmit Frame Length 1/2 Configuration

(R/X)PHASE	(R/X)FRLEN1	(R/X)FRLEN2	Frame Length
0	$0 \leq n \leq 127$	x	Single-phase frame; (n+1) words per frame
1	$0 \leq n \leq 127$	$0 \leq m \leq 127$	Dual-phase frame; (n+1) plus (m+1) words per frame

12.3.4.4 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 12–11. Table 12–13 shows how the value of these fields selects particular element lengths in bits. For the example in Figure 12–8, (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 12–13. 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

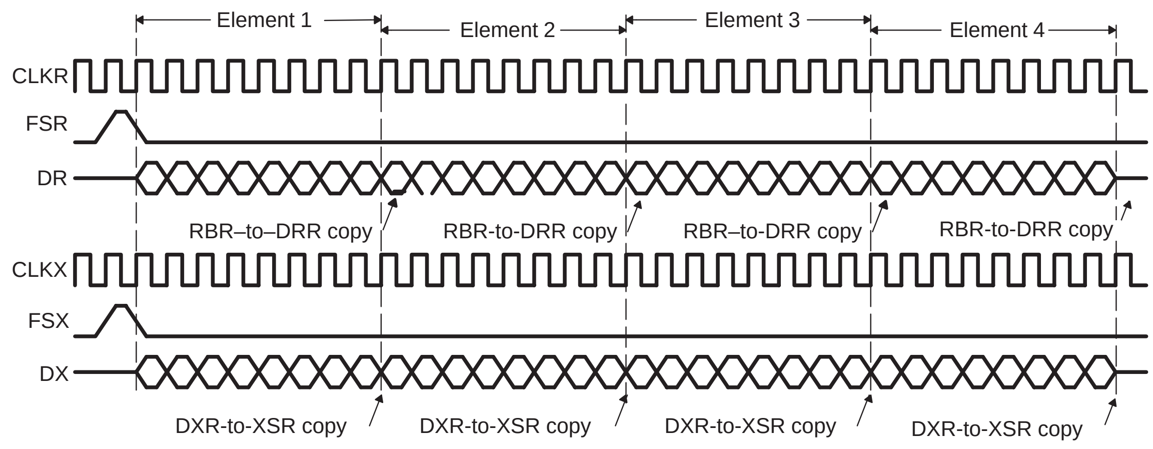
12.3.4.5 Data Packing using Frame Length and Element Length

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 12–9. 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/EDMA controller. Four reads of DRR and four writes of DXR are necessary for each frame.

Figure 12–9. Single-Phase Frame of Four 8-Bit Elements

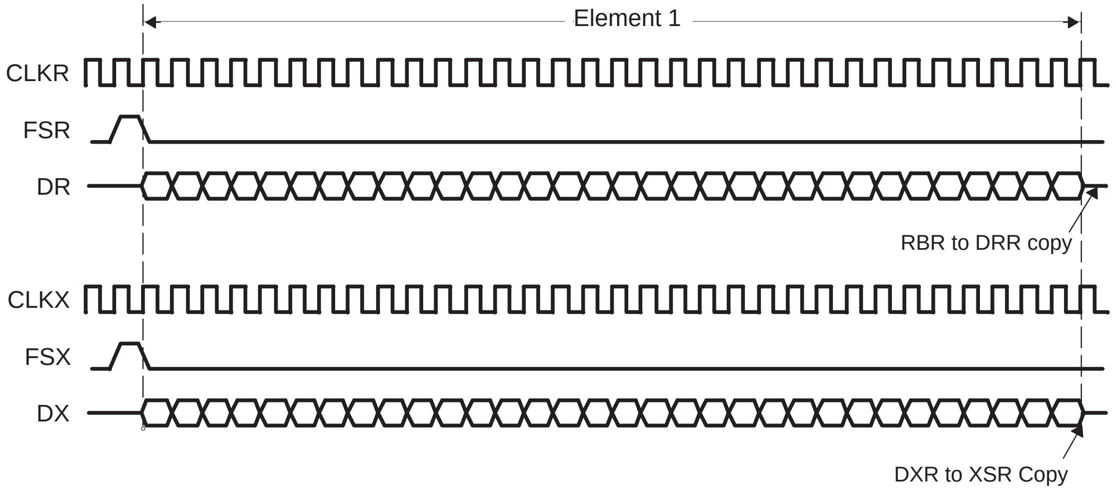


The example in Figure 12–9 can also be viewed as a data stream of a single-phase frame of one 32-bit data element, as shown in Figure 12–10. In this case:

- ❑ (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/EDMA 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.

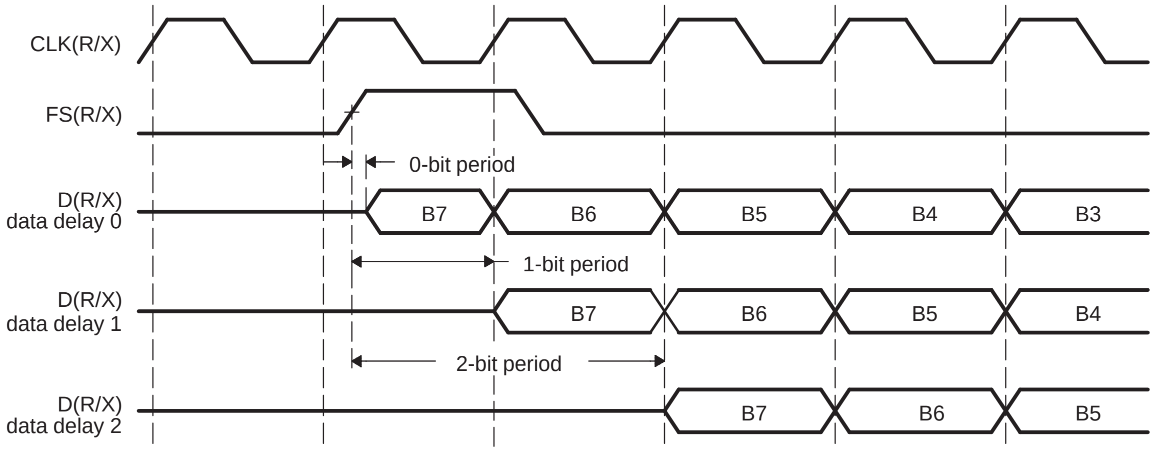
Figure 12–10. Single-Phase Frame of One 32-Bit Element



12.3.4.6 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 12–8 and Table 12–9, and shown in Figure 12–11. Typically, a 1-bit delay is selected because data often follows a 1-cycle active frame sync pulse.

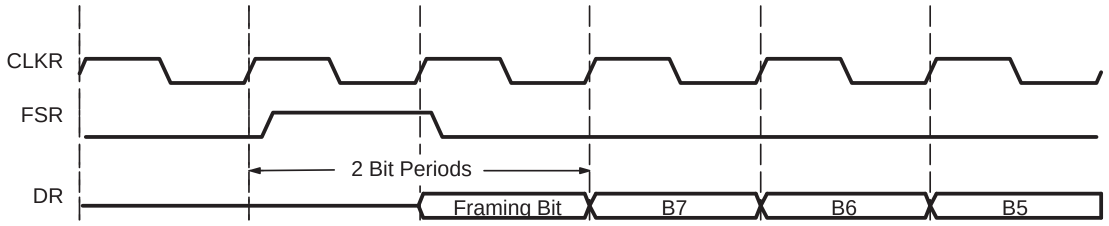
Figure 12–11. Data Delay



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 12–12. 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 12–12. 2-Bit Data Delay Used to Discard Framing Bit

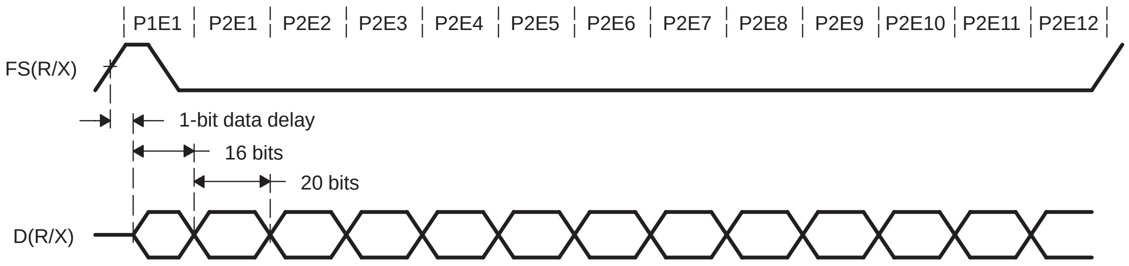


12.3.4.7 Multiphase Frame Example: AC97

Figure 12–13 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 12–13. AC97 Dual-Phase Frame Format†

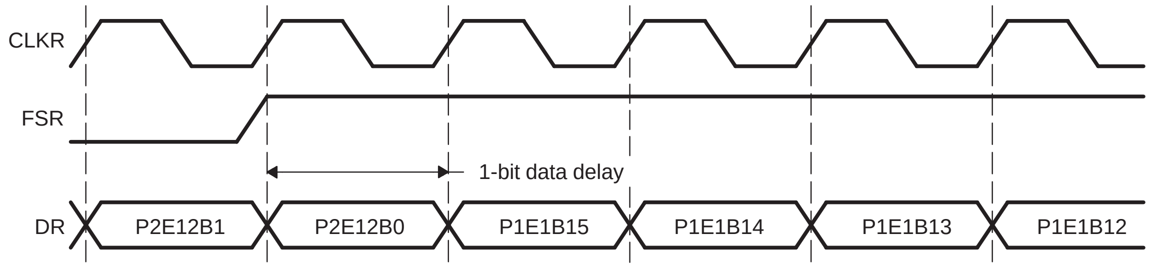


† PxEy denotes phase x and element y.

Figure 12–13 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 12–14, 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 12–14. AC97 Bit Timing Near Frame Synchronization†



† † PxEyBz denotes phase x, element y, and bit z.

12.3.5 McBSP Standard Operation

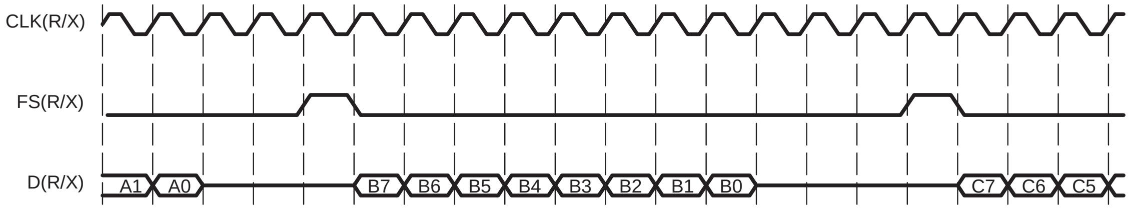
During a serial transfer, there are typically periods of serial port inactivity between packets or transfers. The receive and transmit frame synchronization pulse occurs for every serial transfer. When the McBSP is not in the reset state and has been configured for the desired operation, a serial transfer 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 12–15 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

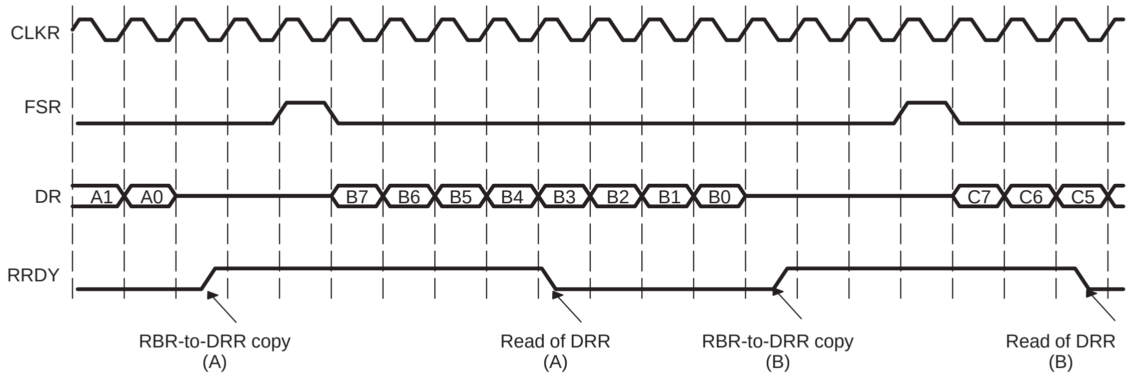
Figure 12–15. McBSP Standard Operation



12.3.5.1 Receive Operation

Figure 12–16 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.

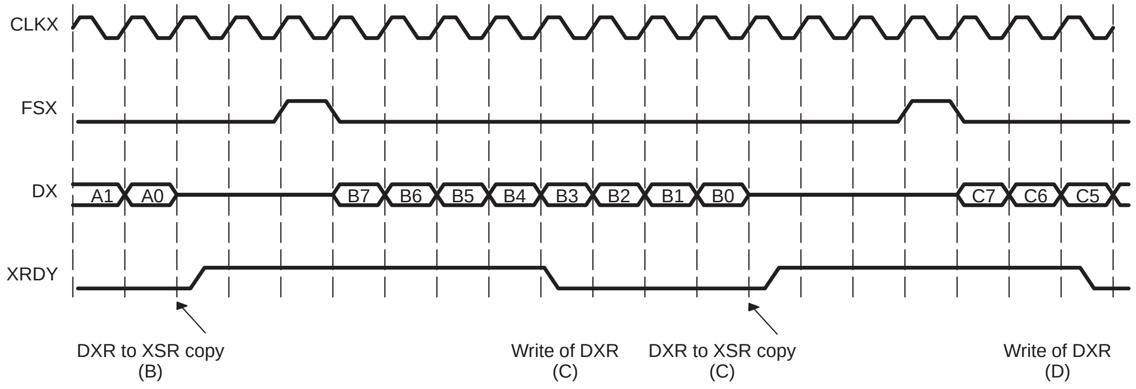
Figure 12–16. Receive Operation



12.3.5.2 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 12–17 illustrates serial transmission. See section 12.3.7.4 for information on transmit operation when the transmitter is pulled out of reset ($\overline{\text{XRST}}$ = 1).

Figure 12–17. Transmit Operation



12.3.5.3 Maximum Frame Frequency

The frame frequency is determined by the following equation, which calculates the period between frame synchronization signals:

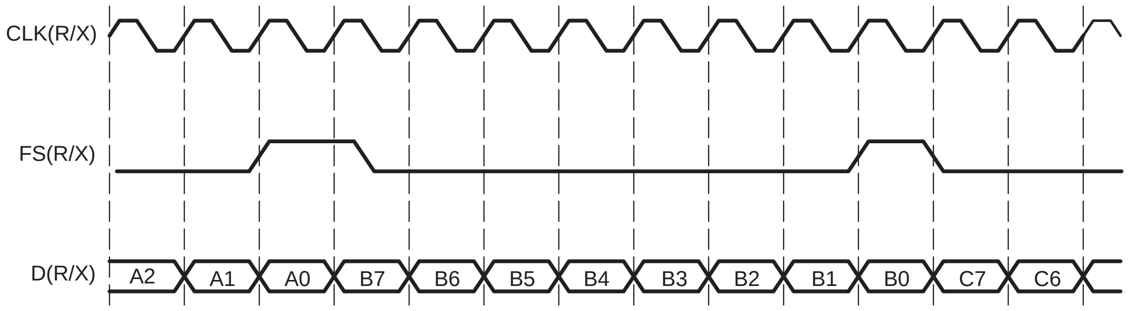
$$\textit{Frame frequency} = \frac{\textit{Bit-clock frequency}}{\textit{Number of bit clocks between frame sync signals}}$$

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 12–18 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 12–18. 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 12–11.

12.3.6 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.

12.3.6.1 Frame Sync Ignore and Unexpected Frame Sync Pulses

RFIG and XFIG are used to ignore unexpected internal or external 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 12–19 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 12.3.7.2 and 12.3.7.5.

Figure 12–19. Unexpected Frame Synchronization With (R/X)FIG = 0

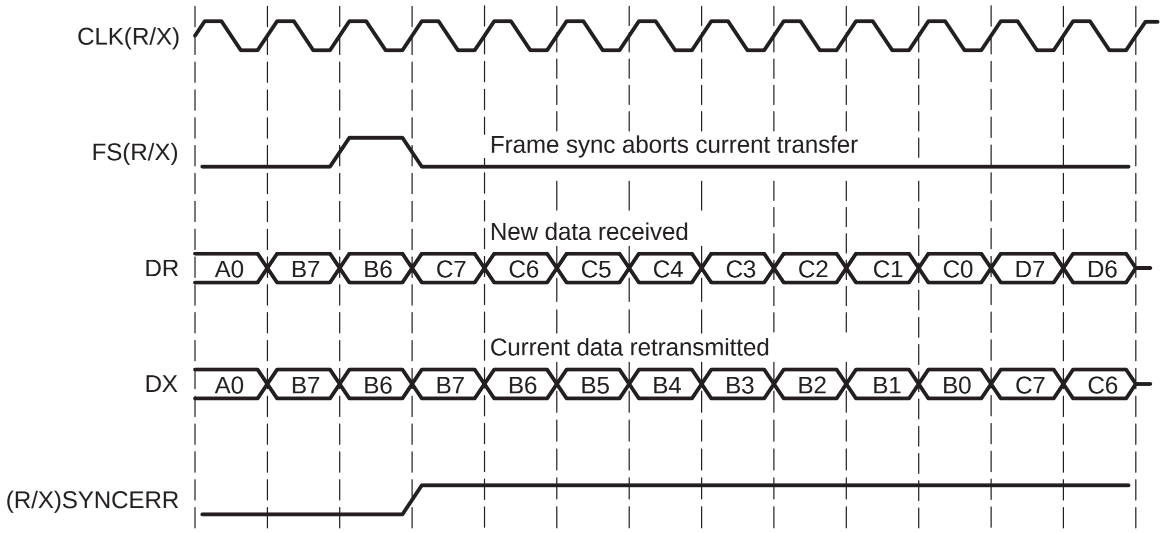
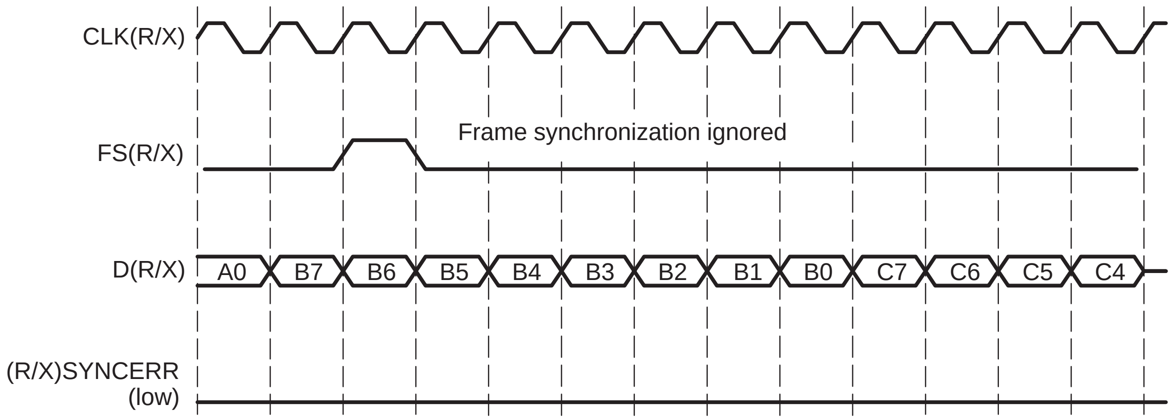

CLK(R/X)
FS(R/X)
Frame sync aborts current transfer
New data received
DR
A0 B7 B6 C7 C6 C5 C4 C3 C2 C1 C0 D7 D6
Current data retransmitted
DX
A0 B7 B6 B7 B6 B5 B4 B3 B2 B1 B0 C7 C6
(R/X)SYNCERR

Figure 12–20 shows McBSP operation when unexpected internal or external 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 12–20. Unexpected Frame Synchronization With (R/X)FIG = 1



12.3.6.2 Data Packing using Frame Sync Ignore Bits

Section 12.3.4.5 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 12–21. 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 12–22 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 12–21. Maximum Frame Frequency Operation With 8-Bit Data

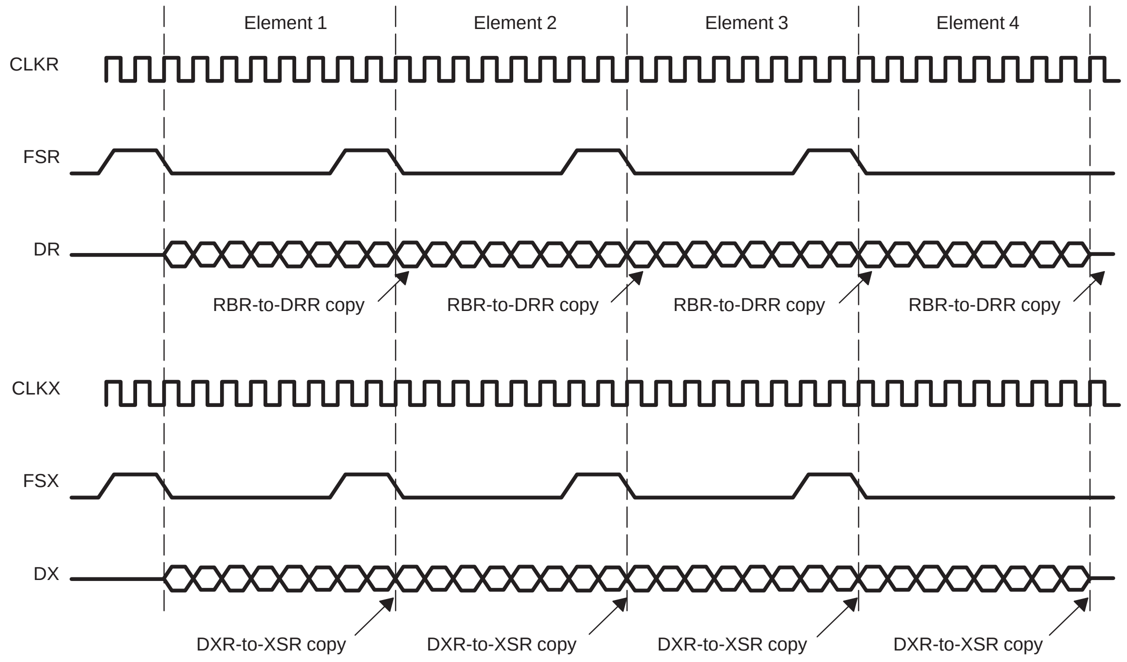
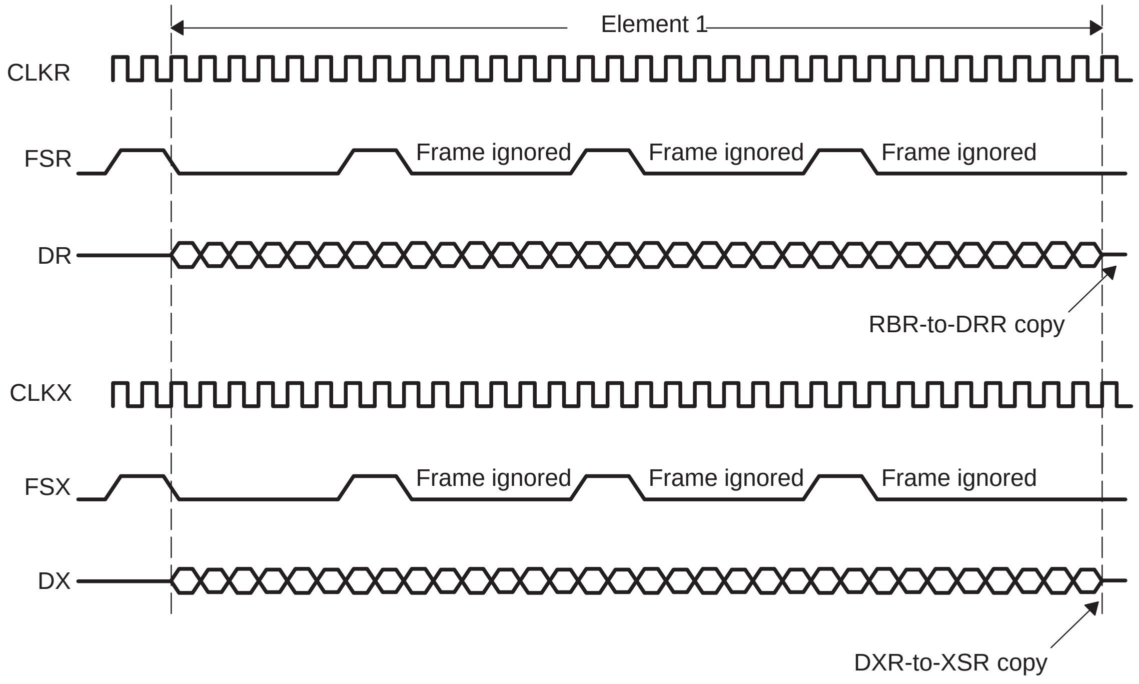


Figure 12–22. Data Packing at Maximum Frame Frequency With (R/X)FIG = 1



12.3.7 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)

12.3.7.1 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 12–24.

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 12–23 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 12–23. Serial Port Receive Overrun

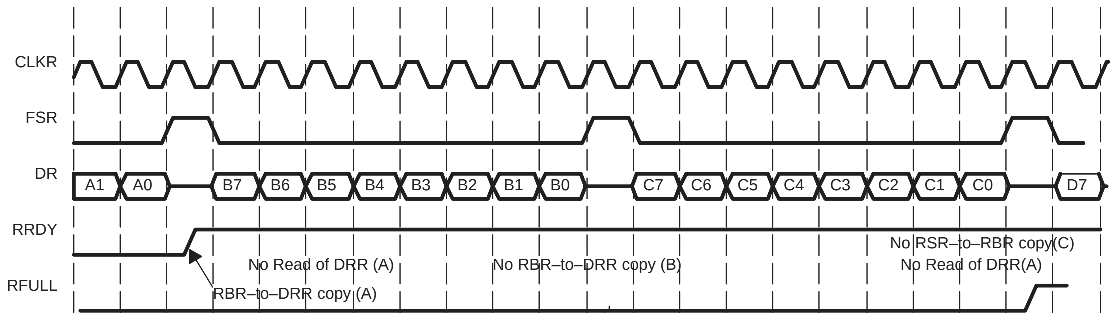
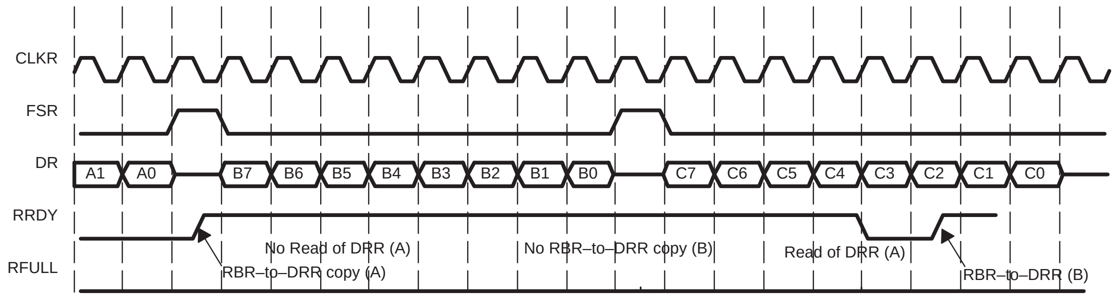


Figure 12–24 shows the case in which RFULL is set but the overrun condition is averted by reading the contents of DRR at least two and a half cycles before the next element, C, is completely shifted into RSR. This ensures that a RBR-to-DRR copy of data B occurs before the next element is transferred from RSR to RBR.

Figure 12–24. Serial Port Receive Overrun Avoided



12.3.7.2 Unexpected Receive Frame Synchronization: RSYNCERR

Figure 12–25 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 12.3.6.1 and shown in Figure 12–20. 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 12–19 for maximum packet frequency. Figure 12–26 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 12–25. Decision Tree Response to Receive Frame Synchronization Pulse

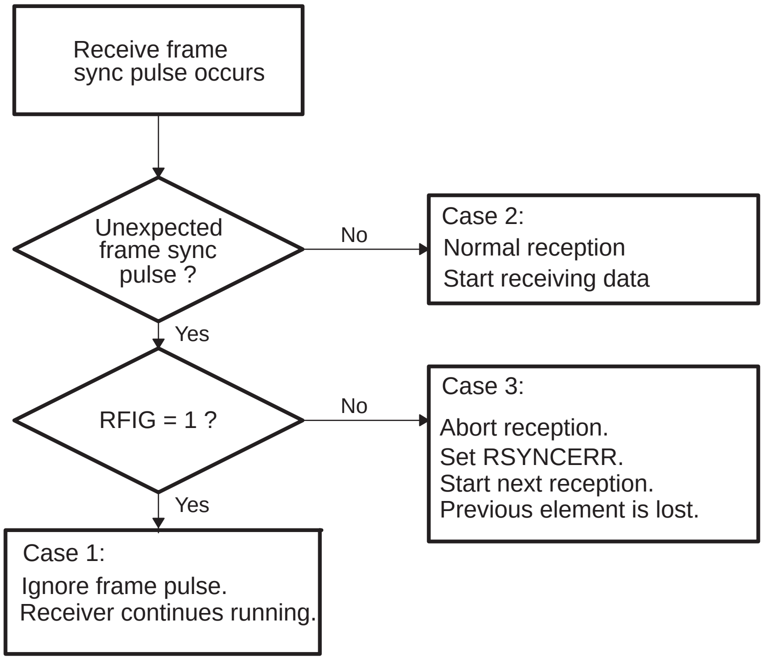
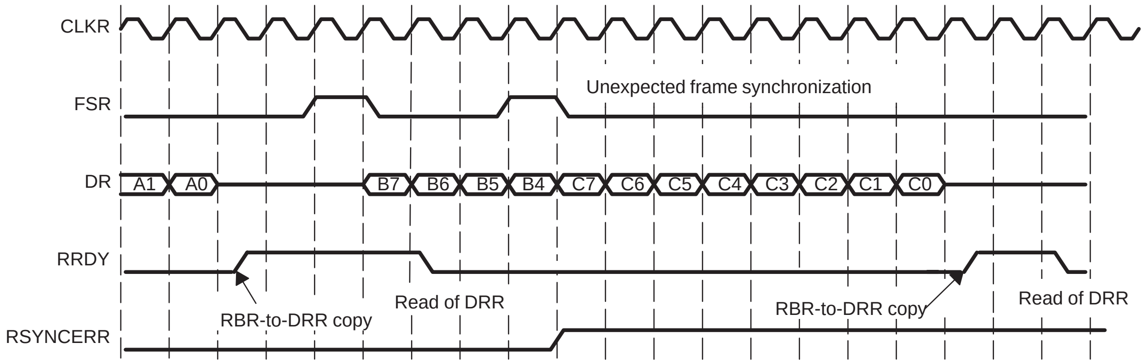


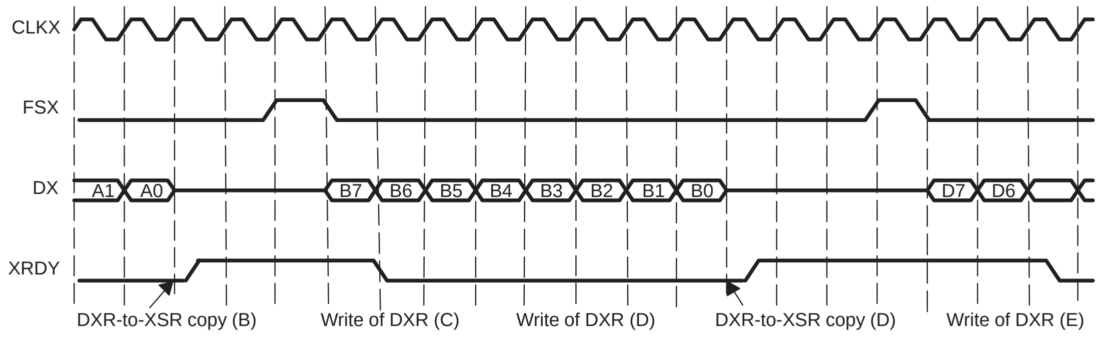
Figure 12–26. Unexpected Receive Synchronization Pulse



12.3.7.3 Transmit With Data Overwrite

Figure 12–27 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/EDMA controller can avoid overwriting by synchronizing data writes with XEVT.

Figure 12–27. Transmit With Data Overwrite



12.3.7.4 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 FSX (generated by an external device, or by the internal sample rate generator) until a new element is loaded into DXR by the CPU or the DMA/EDMA controller. $\overline{\text{XEMPTY}}$ is deactivated ($\overline{\text{XEMPTY}}$ = 1) when this new element in DXR is transferred to XSR. In the case when the FSX is generated by a DXR–to–XSR copy (FSXM=1 in the PCR and FSGM=0 in SRGR), the McBSP does not generate any new frame sync until new data is written to the DXR and a DXR–to–XSR copy occurs.

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 12–28 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 12–29 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

Figure 12–28. Transmit Empty

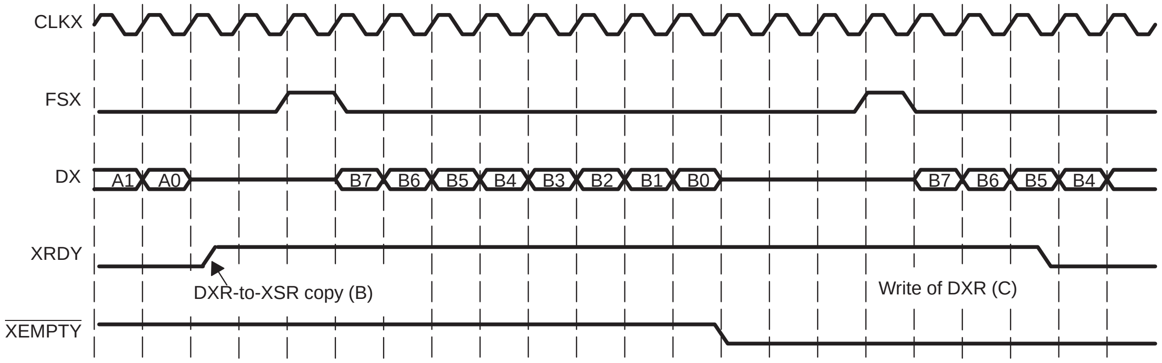
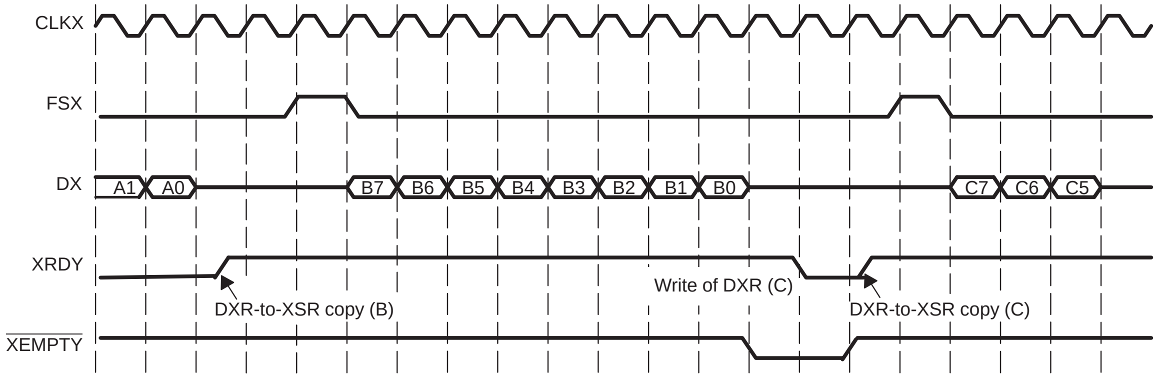


Figure 12–29. Transmit Empty Avoided



12.3.7.5 Unexpected Transmit Frame Synchronization: XSYNCERR

Figure 12–30 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 12.3.6.1 and shown in Figure 12–20. 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 12.3.5.3. 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.

- 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 12–19. Figure 12–31 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 12–30. Response to Transmit Frame Synchronization

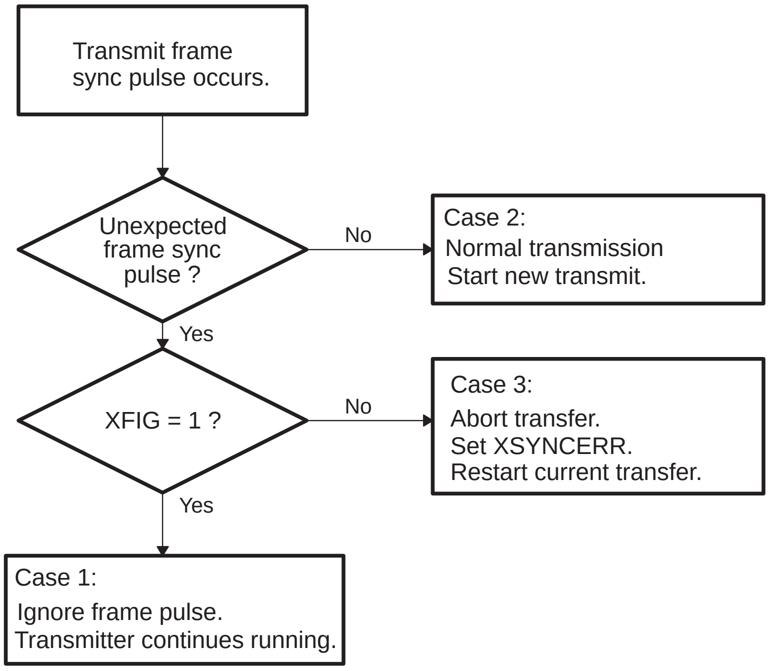
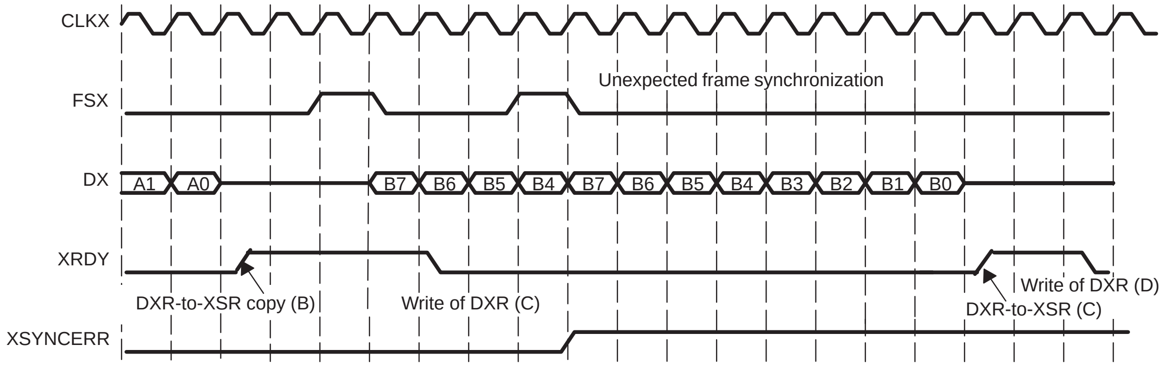


Figure 12–31. Unexpected Transmit Frame Synchronization Pulse



12.3.8 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 12–14 summarizes the effect that various values of RJUST have on an example 12-bit receive data value ABCh.

Table 12–14. 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

12.3.9 32-Bit Bit Reversal: (R/X)WDREVRS

The 32-bit bit reversal feature is only available on the C621x/C671x/C64x device. Normally all transfers are sent and received with the MSB first. However, you can reverse the receive/transmit bit ordering of a 32-bit element (LSB first) by setting all of the following:

- (R/X)WDREVRS = 1 in the receive/transmit control register RCR/XCR.
- (R/X)COMPAND = 01b in the RCR/XCR.
- (R/X)WDLEN(1/2) = 101b in the RCR/XCR to indicate 32-bit elements.

When you set the register fields as above, the bit ordering of the 32-bit element is reversed before being received by or sent from the serial port. If the (R/W)WDREVRS and (R/X)COMPAND fields are set, but the element size is not set to 32-bit, operation is undefined.

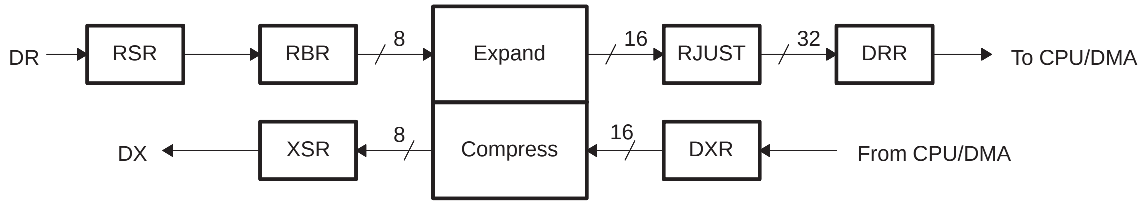
12.4 µ-Law/A-Law Companding Hardware Operation

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 12–8 and Table 12–9. Compression occurs during the process of copying data from DXR to XSR and expansion occurs from RBR to DRR, as shown in Figure 12–32.

Figure 12–32. Companding Flow



For transmit data to be compressed, it should be 16-bit, left-justified data, such as LAW16 as shown in Figure 12–33. 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 12–34.

Figure 12–33. Companding Data Formats

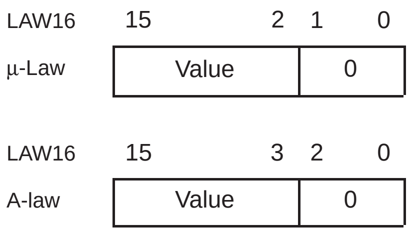
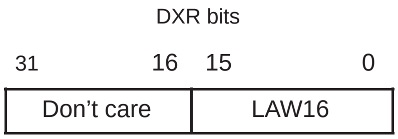


Figure 12–34. Transmit Data Companding Format in DXR



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 12–15.

Table 12–15. Justification of Expanded Data in DRR

	DRR Bits	
RJUST	31 16	15 0
00	0	LAW16
01	sign	LAW16
10	LAW16	0
11	Reserved	

12.4.1 Companding Internal Data

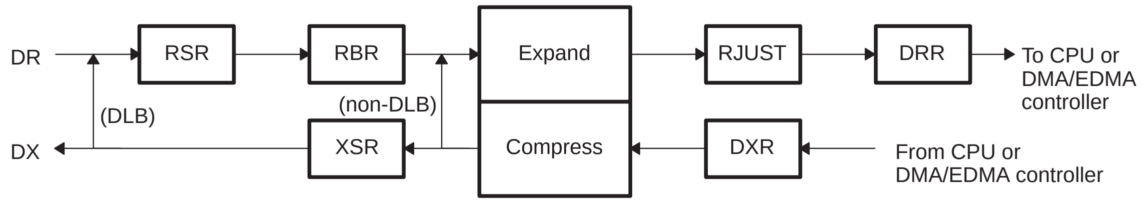
If the McBSP is otherwise unused, the companding hardware can compand internal data. This hardware can be used to:

- 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 12–35 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/EDMA 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/EDMA controller to these conversions, respectively. Here, the time for this companding depends on the serial bit rate selected.

Figure 12–35. Companding of Internal Data


DR
RSR
RBR
Expand
RJUST
DRR
To CPU or DMA/EDMA controller
(DLB)
(non-DLB)
DX
XSR
Compress
DXR
From CPU or DMA/EDMA controller

12.4.1.1 Bit Ordering

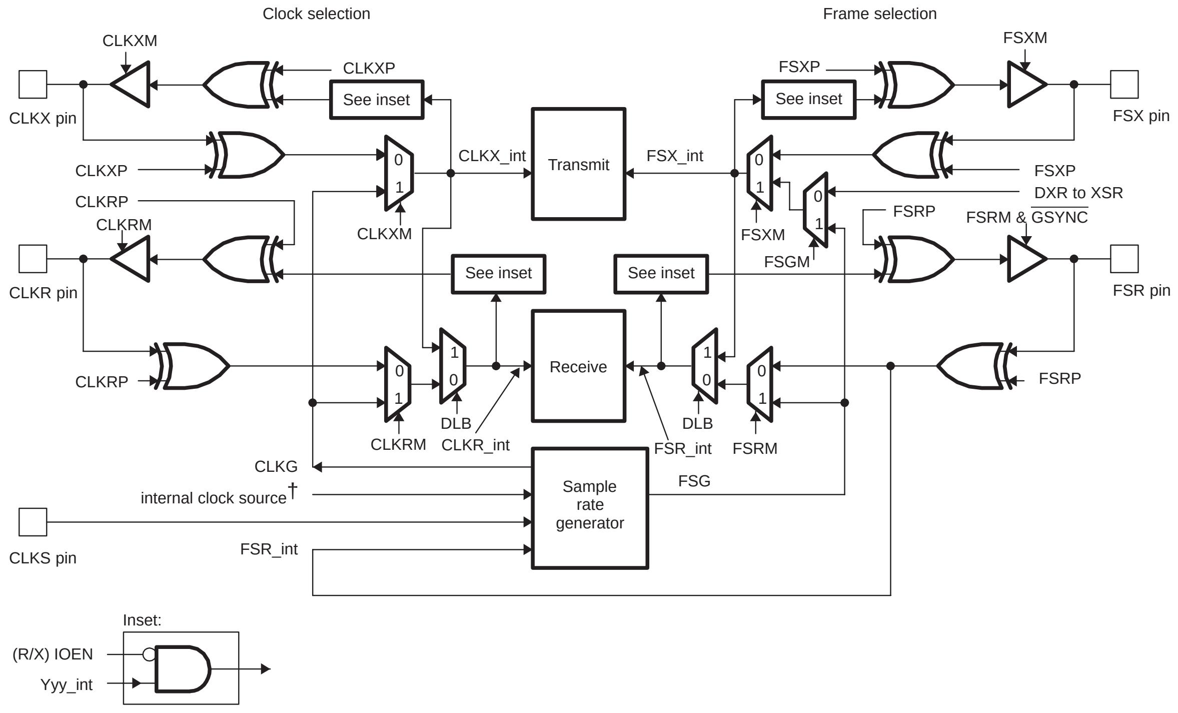
Normally, all transfers on the McBSP are sent and received with the MSB first. However, certain 8-bit data protocols (that do not use companded data) require the LSB to be transferred first. By setting the (R/X)COMPAND = 01b in the (R/X)CR, the bit ordering of 8-bit elements is reversed (LSB first) before 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.

For the C621x/C671x/C64x, a 32–bit bit reversal feature is also available. See section 12.3.9.

12.5 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 12–36 is a block diagram of the clock and frame selection circuitry.

Figure 12–36. Clock and Frame Generation



† Internal clock source:
CPU clock for C620x/C670x
CPU/2 clock for C621x/C671x
CPU/4 clock for C64x.

12.5.1 Sample Rate Generator Clocking and Framing

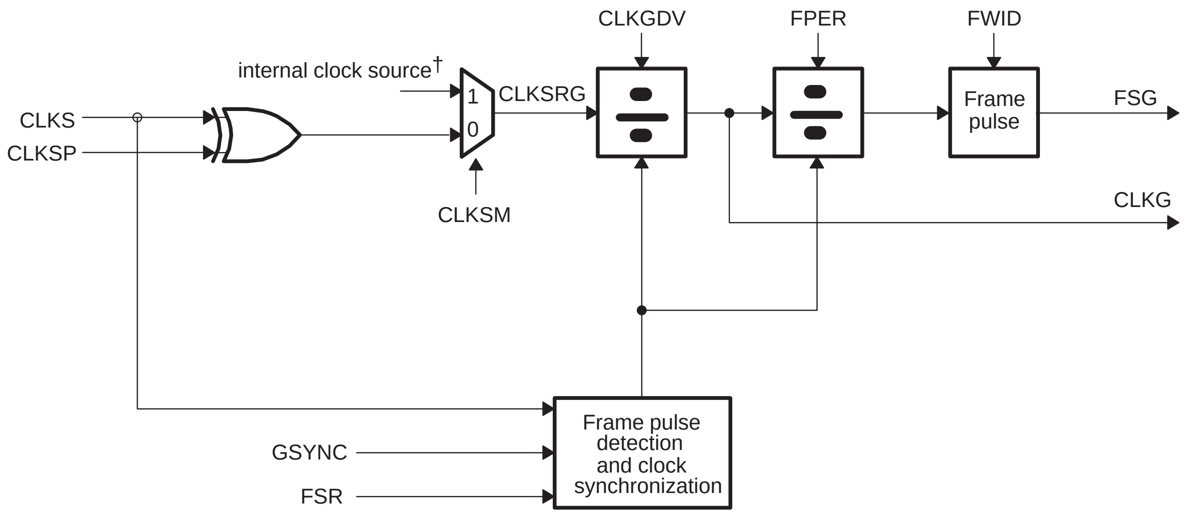
The sample rate generator is composed of a 3-stage clock divider that provides a programmable data clock (CLKG) and framing signal (FSG), as shown in Figure 12–37. CLKG and FSG are McBSP internal signals that can be programmed to drive receive and/or transmit clocking, CLK(R/X), and framing, FS(R/X). 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 sample rate generator is not used when CLKX, FSX, CLKR, FSR are driven by an external source. Therefore, the $\overline{\text{GRST}}$ field in SPCR does not need to be enabled ($\overline{\text{GRST}}$=1) for this setup. 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 12.3.1.

Figure 12–37. Sample Rate Generator



† Internal clock source:
CPU clock for C620x/C670x
CPU/2 clock for C621x/C671x
CPU/4 clock for C64x.

12.5.1.1 Sample Rate Generator Register (SRGR)

The sample rate generator register (SRGR) shown in Figure 12–38 and summarized in Table 12–16, controls the operation of various features of the sample rate generator. This section describes the fields in the SRGR.

Figure 12–38. 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 12–16. Sample Rate Generator Register (SRGR) Field Summary

Bit No.	**Name**	**Function**	**Section**
31	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.	12.5.2.4
30	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.	12.5.2.3
29	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.	12.5.2.1
28	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.	12.5.3.3
27–16	FPER	Frame period. This field’s value plus 1 determines when the next frame sync signal should become active. Valid values: 0 to 4095	12.5.3.1

Table 12–16. Sample Rate Generator Register (SRGR) Field Summary (Continued)

Bit No.	Name	Function	Section
15–8	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	12.5.3.1
7–0	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	12.5.2.2

12.5.1.2 McBSP and Sample Rate Generator Reset Procedure

The McBSP and sample rate generator reset and initialization procedure is as follows:

1) Ensure that no portion of the McBSP is using the internal sample-rate generator signals CLKG and FSG (If necessary, set /RRST and/or /XRST to 0). Set /FRST=/GRST=0 in the SPCR. If the device has been reset (/RRST=/XRST=/FRST=/GRST=0), this step is not required. CLKG and FSG are inactive low when /GRST=0.
2) Program the SRGR as required. Other control registers can be programmed if the respective portion (receiver/transmitter) is in reset.
3) Wait two CLKSRG clocks for proper internal synchronization.
4) To use the sample rate generator, set /GRST=1 and wait two CLKG bit clocks for synchronization. Skip this step if the internal sample rate generator is not used.
5) Set up data acquisition as desired (set up DMA/EDMA or CPU data transfers).
6) Set /XRST and/or /RRST to 1 to enable the corresponding section of the serial port. If the internal sample rate generator is not used, the McBSP is now ready to transmit and/or receive (skip step 7 and 8).
7) On the next rising edge of CLKSRG, CLKG transitions to 1 and starts clocking with a frequency equal to 1/(CLKGDV+1) of the sample rate generator input clock (see section 12.5.2.1). If an external device supplies the frame synchronization signals, the McBSP is now ready to transmit and/or receive. If the McBSP is the frame master, see also step 8.
8) Set /FRST=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.

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 12.4.1.

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 12.6.3.2 for more information.

12.5.2 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 C620x/C670x uses the CPU clock as the internal clock source to the sample rate generator. The C621x/C671x uses the CPU/2 clock as the internal clock source, while the C64x uses the CPU/4 clock as the internal clock source to the sample rate generator.

❑ 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 12–37) generates CLKG and FSG (see section 12.5.2.3).

12.5.2.1 Input Clock Source Mode: CLKSM

The CLKSM bit in the SRGR selects either the internal 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.

12.5.2.2 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.

Refer to the timing requirements in the device data sheet to determine the maximum McBSP bit rate. CLKGDV should be set appropriately to ensure that the McBSP clock rate does not exceed the bit rate limit.

12.5.2.3 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.

12.5.2.4 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 12–39 and Figure 12–40 show this operation with various polarities of CLKS and FSR. These figures assume that FWID is 0, for a FSG = 1 CLKG wide.

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.

Figure 12–39. CLKG Synchronization and FSG Generation When GSYNC = 1 and CLKGDV = 1

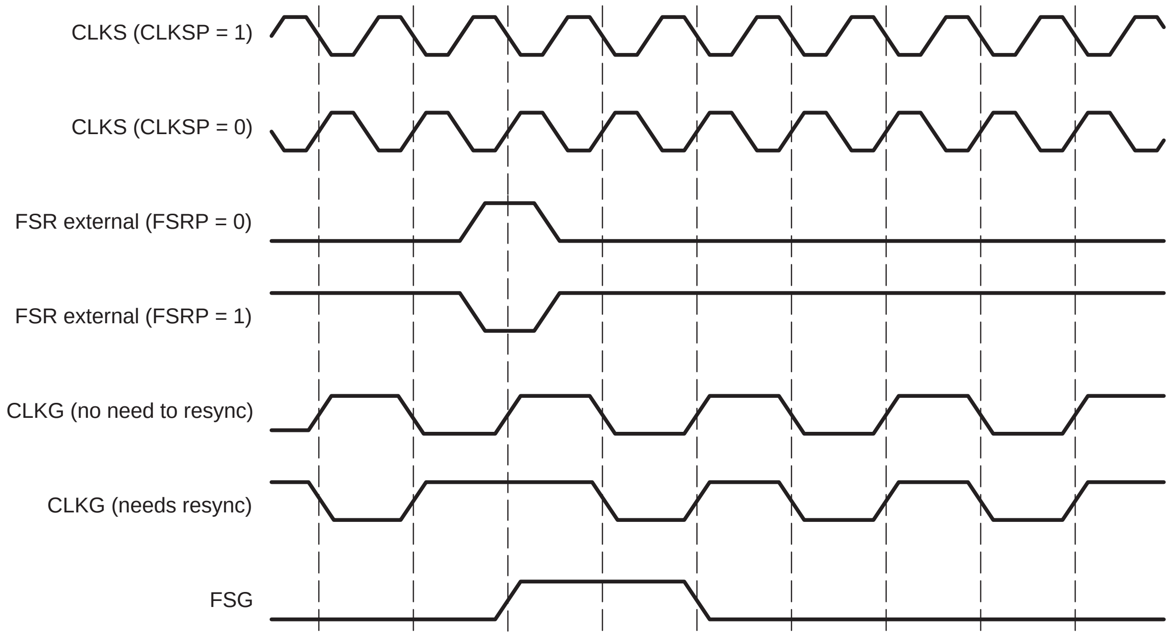
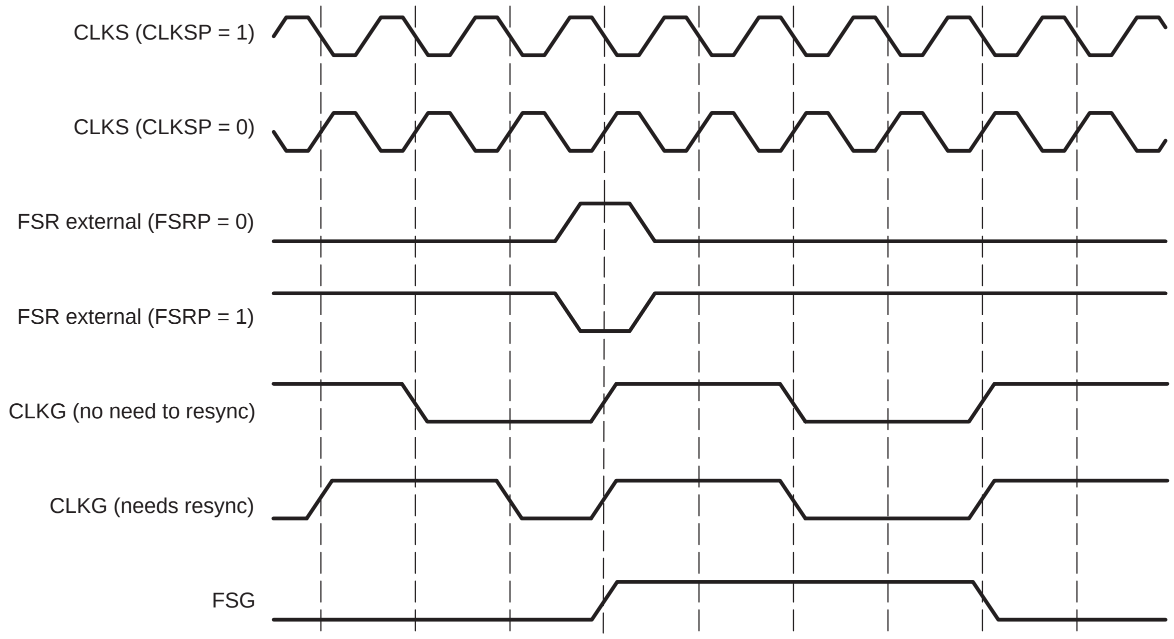


Figure 12–40. CLKG Synchronization and FSG Generation When GSYNC = 1 and CLKGDV = 3



12.5.2.5 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 12–36. DLB mode allows testing of serial port code with a single DSP device. DLB mode cannot be used when the McBSP is in clock stop mode (CLKSTP=1x in the SPCR). CLKK and FSX must be enabled as outputs (CLKXM=FSXM=1) in DLB mode.

12.5.2.6 Receive Clock Selection: DLB, CLKRM

Table 12–17 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 12–17. 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 12–18.	High impedance
1	1	CLKX_int drives CLKR_int as selected and is inverted. See Table 12–18.	Output. CLKR (same as CLKX) inverted as determined by CLKRP before being driven out.

12.5.2.7 Transmit Clock Selection: CLKXM

Table 12–18 shows how the CLKXM bit in the PCR selects the transmit clock and whether the CLKX pin is an input or output.

Table 12–18. 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.

12.5.3 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:

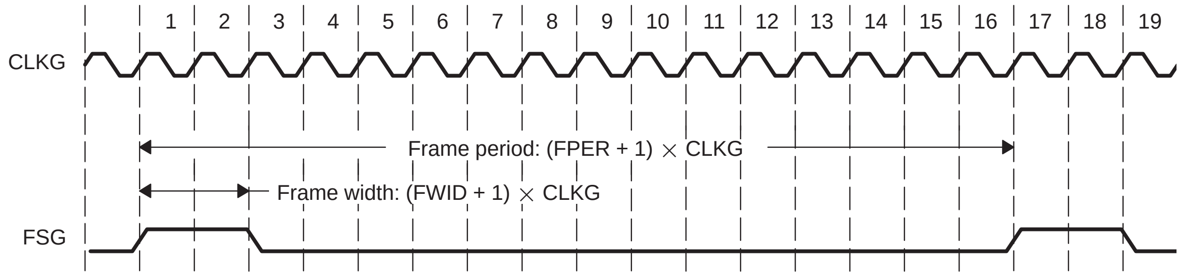
- ❑ A frame pulse with a programmable period between sync pulses and a programmable active width specified in the sample rate generator register (SRGR).
- ❑ The transmitter can trigger its own frame sync signal that is generated by a DXR-to-XSR copy. This causes a frame sync to occur on every DXR-to-XSR 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.

12.5.3.1 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 12–41 shows a frame of 16 CLKG periods (FPER = 15 or 00001111b).

Figure 12–41. 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

12.5.3.2 Receive Frame Sync Selection: DLB, FSRM, GSYNC

Table 12–19 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 12–19. 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 12–20.	High impedance
1	X	1	FSX_int drives FSR_int and is selected as shown in Table 12–20.	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 12–20.	Output. Receive (same as transmit) frame synchronization is inverted as determined by FSRP before being driven out.

12.5.3.3 Transmit Frame Sync Signal Selection: FSXM, FSGM

Table 12–20 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 12–20. 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.

12.5.3.4 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.

12.5.4 Stopping Clocks

There are two ways to stop serial clocks between data transfers. One is using the SPI CLKSTP mode where clocks are stopped between single-element transfers. This is described in section 12.7.

The other method is when the clocks are inputs to the McBSP (CLKXM or CLKRM = 0) and the McBSP operates in non-SPI mode. This means that clocks can be stopped between frames.

There are two scenarios:

❑ **CLKR, CLKX, FSR, and FSX are all inputs to McBSP.**
If the external device stops the serial clock between data transfers, it needs to restart the clock at least three CLKR/CLKX cycles before the next frame sync to allow proper synchronization.

- **CLKR/CLKX are inputs; FSR/FSX are outputs generated by McBSP**. If the external device stops the serial clock between data transfers, the McBSP interprets it as a slow-down serial clock. Ensure that there are no glitches on the CLKR/X lines as the McBSP may interpret them as clock edge transitions. Since restarting the serial clock is equivalent to a normal clock transition after a slow CLKR/X cycle, it is not necessary to restart the serial clock a few cycles early for internal synchronization.

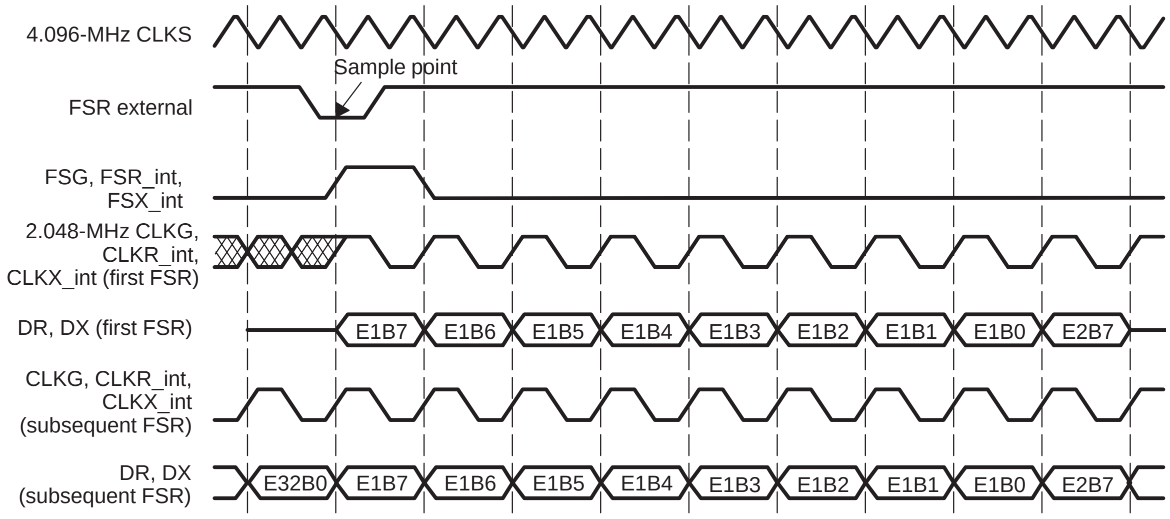
12.5.5 Clocking Examples

12.5.5.1 Double-Rate ST-BUS Clock

Figure 12–42 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 12–42. ST-BUS and MVIP Example

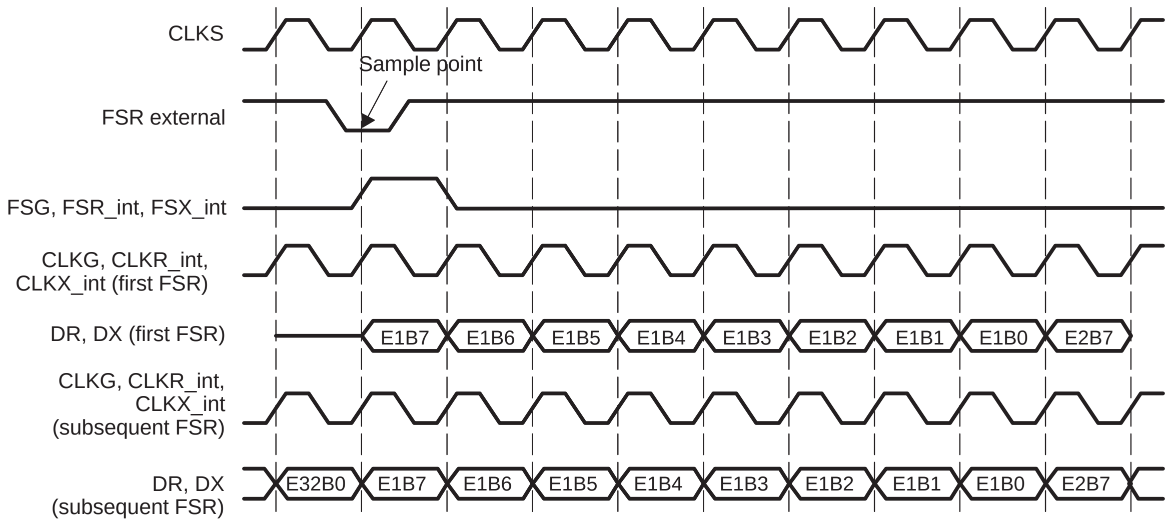


12.5.5.2 Single-Rate ST-BUS Clock

The example in Figure 12–43 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 12–43. 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.

12.5.5.3 Double-Rate Clock

The example in Figure 12–44 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.

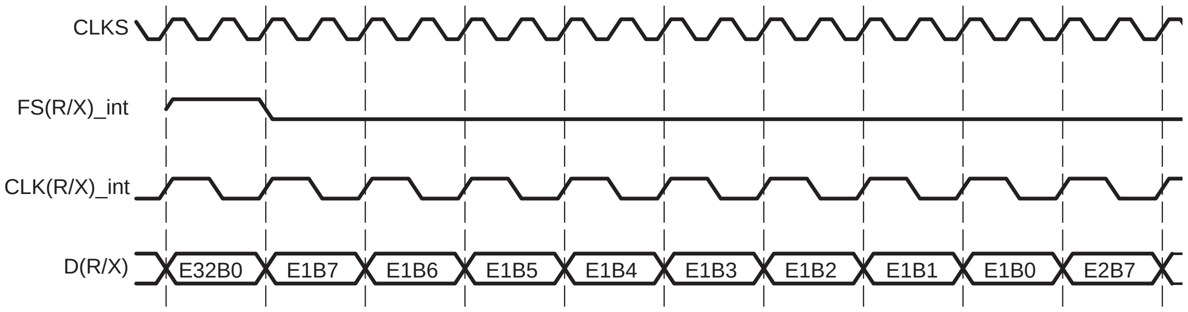
❑ GSYNC = 0: CLKS drives CLKG. CLKG runs freely and is not resynchronized by FSR.

❑ 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 12–44. Double-Rate Clock Example



12.6 Multichannel Selection Operation

The multichannel selection mode allows the McBSP to select independent channels (elements) for transmit and receive in a single-phase frame. Each frame represents a time-division multiplexed data stream. For all of the C6000 McBSP, up to 32 elements in a bit stream of up to 128 elements can be enabled at any given time. The C64x McBSP is an enhanced version that can also select up to 128 elements at any given time (see section 12.6.4).

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.

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).

Note:

For the C64x enhanced multichannel selection mode only, the XCER and RCER are replaced by XCERE0 and RCERE0, respectively. Additional registers XCERE1, XCERE2, XCERE3, RCERE1, RCERE2 and RCERE3 are also used in this mode.

12.6.1 Multichannel Control Register (MCR)

The multichannel control register (MCR) contains fields that control the multichannel selection mode. The MCR fields are shown in Figure 12–45. Descriptions of these fields are summarized in Table 12–21. The enhanced 128-channel selection mode (selected by the RMCME and XMCME fields), which allows the McBSP to select 128 channels at any time, is only available on the C64x (section 12.6.4).

Figure 12–45. Multichannel Control Register (MCR)

31 26	25	24 23	22 21	20 18	17 16
reserved	XMCME†	XPBBLK	XPABLK	XCBLK	XMCM
R, +0000 00	RW, +0	RW, +00	RW, +00	R, +000	RW, +00

15 10	9	8 7	6 5	4 2	1	0
reserved	RMCME†	RPBBLK	RPABLK	RCBLK	Rsvd	RMCM
R, +0000 00	RW, +0	RW, +00	RW, +00	R, +000	R, +0	RW, +0

Note: †XMCME and RMCME are only available on C64x. These bit fields are reserved (R, +0) on all other C6000 devices.

Table 12–21. Multichannel Control Register (MCR) Field Descriptions

Bit No.	Name	Function	Section
25	XMCME	Enhanced transmit multichannel selection enable (C64x only). See descriptions for RMCME.	12.6.4

24–23	XPBBLK	Transmit partition B subframe
		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
			12.6.3

22–21	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
			12.6.3

Table 12–21. Multichannel Control Register (MCR) Field Descriptions (Continued)

Bit No.	Name	Function		Section
20–18	XCBLK	Transmit current subframe		12.6.3.2
		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		
17–16	XMCM	Transmit multichannel selection enable		12.6.2 12.6.4
		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. In normal multichannel selection mode (RMCME = XMCME = 0), required elements are selected by enabling XP(A/B)BLK and XCER appropriately. In enhanced multichannel selection mode (For C64x only, RMCME = XMCME = 1), required elements are selected by enabling XCERE0 – XCERE3 appropriately. These selected elements are not masked. DX is always driven.	
		XMCM = 10b:	All elements are enabled but masked. In normal multichannel selection mode (RMCME = XMCME = 0), required elements are selected by enabling XP(A/B)BLK and XCER appropriately. In enhanced multichannel selection mode (For C64x only, RMCME = XMCME = 1), required elements are selected by enabling XCERE0 – XCERE3 appropriately. These selected elements are unmasked.	
		XMCM = 11b:	All elements are disabled and therefore masked by default. In normal multichannel selection mode (RMCME = XMCME = 0), 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. In enhanced multichannel selection mode (For C64x only, RMCME = XMCME = 1), required elements are selected by enabling RCERE0 – RCERE3 appropriately; selected elements can be unmasked by XCERE0 – XCERE3. XMCM = 11b is used for symmetric transmit and receive operation.	

Table 12–21. Multichannel Control Register (MCR) Field Descriptions (Continued)

Bit No.	Name	Function	Section
9	RMCME	Enhanced receive multichannel selection enable (C64x only) RMCME operates in conjunction with XMCME. The RMCME and XMCME bit values need to be the same. RMCME = 0 and XMCME = 0: Normal multichannel selection mode (Default value). Maximum 32 channels can be enabled at one time. Multichannel operation is comparable to the C620x McBSP. RMCME = 1 and XMCME = 1 Enhanced multichannel selection mode. Maximum 128 channels can be enabled at one time. All other modes reserved. Multichannel operation is undefined if RMCME ≠ XMCME.	12.6.4

8–7	RPBBLK	Receive partition B subframe
		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
			12.6.3

6–5	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
			12.6.3

Table 12–21. Multichannel Control Register (MCR) Field Descriptions (Continued)

Bit No.	Name	Function	Section

4–2	RCBLK	Receive current subframe
		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
			12.6.3.2

0	RMCM	Receive multichannel selection enable RMCM = 0: All channels are enabled. RMCM = 1: All elements are disabled. In normal multichannel selection mode (RMCME = XMCME = 0), required channels are selected by enabling RP(A/B)BLK and RCER appropriately. In enhanced multichannel selection mode(RMCME = XMCME = 1), required channels are selected by enabling RCERE0 – RCERE3 appropriately.	12.6.2 12.6.4

12.6.2 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.

12.6.3 Enabling and Masking of Channels in Normal Multichannel Selection Mode

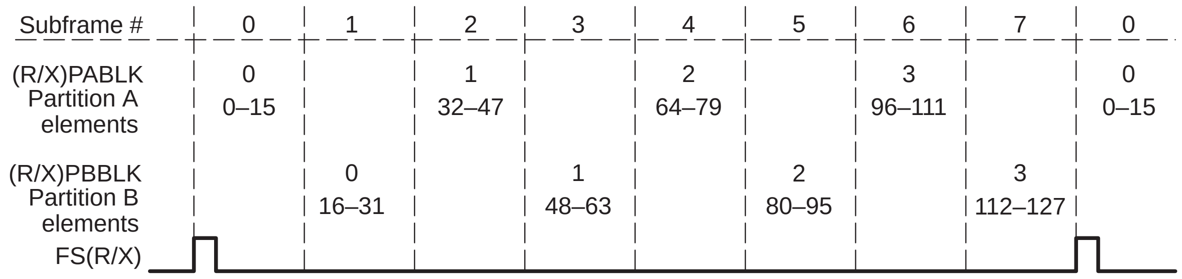
This section describes how to enable the channels in normal multichannel selection mode. For the C64x, see also section 12.6.4 for the enhanced multichannel selction mode.

For all C6000 devices, 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 12–46. 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 12–46. 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 in normal multichannel selection mode:

❑ 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. Select desired receive channels by setting the RCERA-RCERD. Symmetric operation occurs when a device transits 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/BBLK 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).

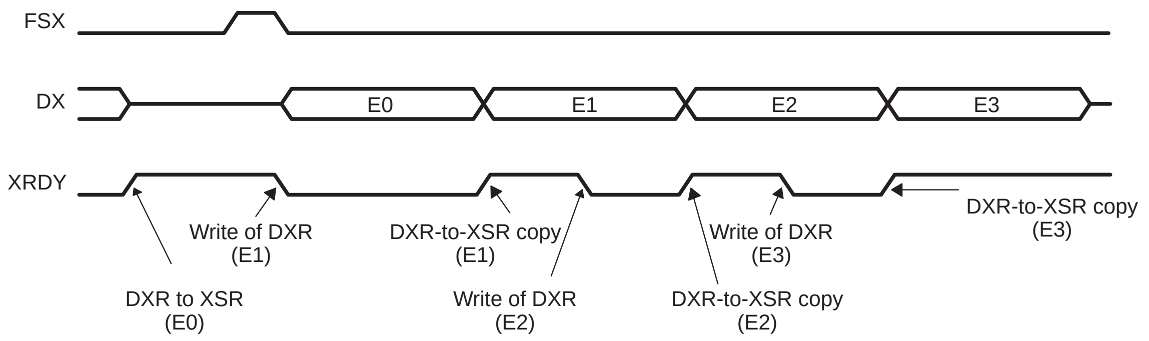
Figure 12–47 shows the activity on the McBSP pins for all of the preceding XMCM values with the following conditions:

- (R/X)PHASE = 0: Single-phase frame for multichannel selection enabled
- FRLEN1 = 011b: 4-element frame
- WDLEN1 = Any valid serial element length

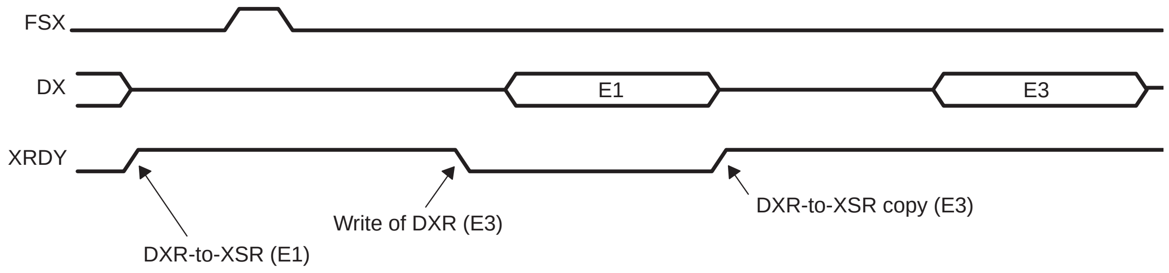
In the following illustrations, the arrows indicating the occurrence of events are only sample indications.

Figure 12–47. XMCM Operation

(a) XMCM = 00b



(b) XMCM = 01b, XPABLK = 00b, XCER = 1010b



(c) XMCM = 10b, XPABLK = 00b, XCER = 1010b

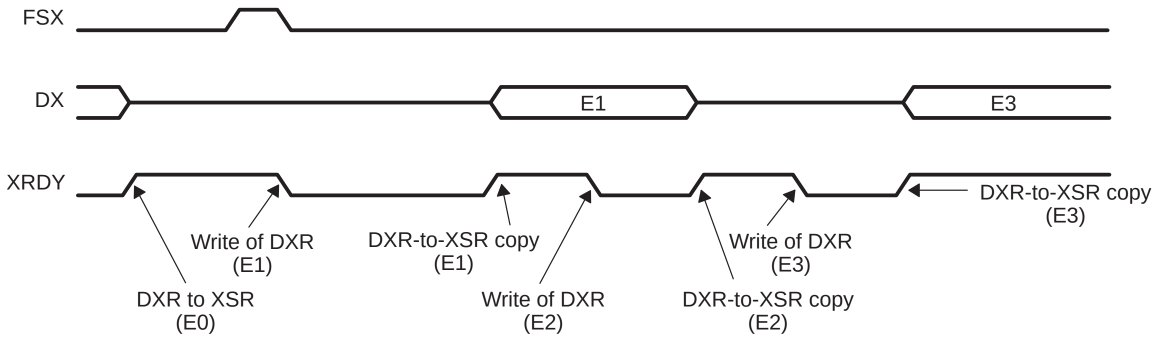
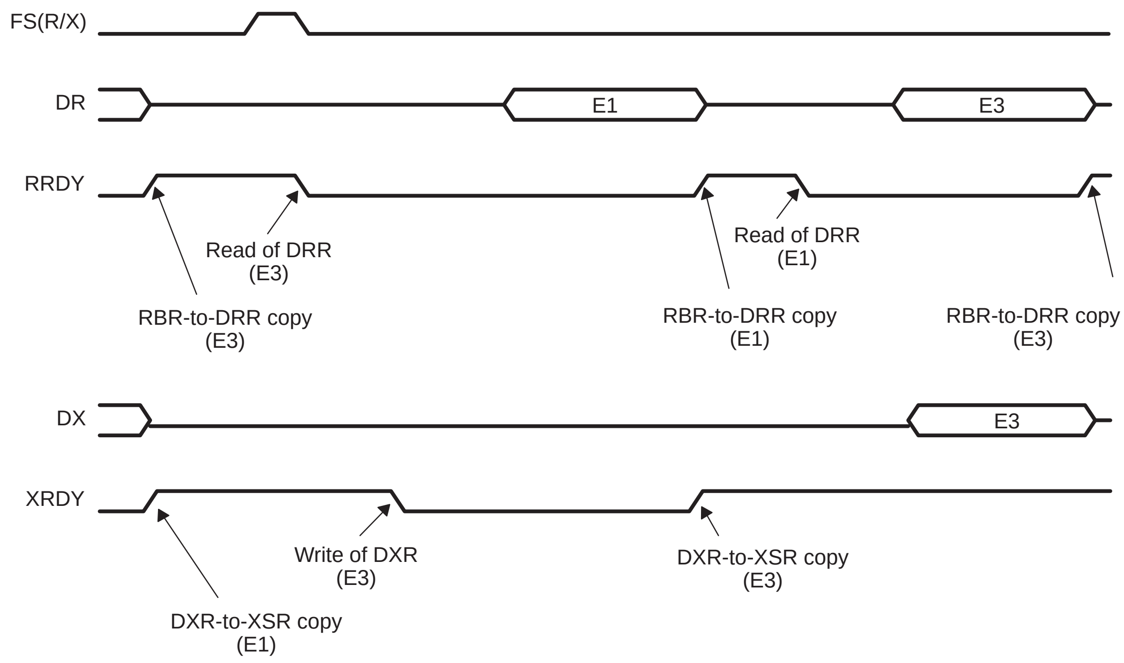


Figure 12–47. XMCM Operation (Continued)

(d) XMCM = 11b, RPABLK = 00b, XPABLK = X, RCER = 1010b, XCER = 1000b



12.6.3.1 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 12–48 and Figure 12–49. The (R/X)CEA and (R/X)CEB register fields shown in Table 12–22 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 12–48. 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

Table 12–22. Receive 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
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

Figure 12–49. 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 12–23. Transmit Channel Enable Register Field Description

Name	Function
XCEA*n* $0 \le n \le 15$	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
XCEB*n* $0 \le n \le 15$	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

12.6.3.2 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 12.6.3.3 for information about these interrupts).

Note:

The user 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.

12.6.3.3 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.

12.6.4 Enhanced Multichannel Selection Mode (C64x only)

In addition to the normal multichannel selection mode, the C64x McBSP has the enhanced multichannel selection mode, which allows up to 128 channels to be enabled at any given time. The enhanced multichannel selection mode is selected by setting the enhanced receive multichannel selection enable bit (RMCME), and enhanced transmit mulitichannel selection enable bit (XMCME), in the MCR to 1. This mode works in conjunction with six additional enhanced receive/transmit channel enable registers in the C64x McBSP: RCERE1, RCERE2, RCERE3, XCERE1, XCERE2, and XCERE3. The RCER and XCER described in section 12.6.3.1 are replace by the RCERE0 and XCERE1, respectively.

When RMCME = XMCME = 0, the C64x McBSP is in the normal multichannel selection mode. See sections 12.6.2 and 12.6.3 for a detailed description. In normal multichannel selection mode, The RCERE1–RCERE3 and XCERE1–XCERE3 are not used. RCERE0 and XCERE0 function as RCER and XCER, respectively.

When RMCME and XMCME = 1, the C64x McBSP has 128-channel selection capability. The registers RCERE0–RCERE3 and XCERE0–XCERE3 are used to enable up to 128 channels. Since up to 128 channels can be selected at one time, the (R/X)P(A/B)BLK and (R/X)CBLK values in the MCR are “don’t cares” and have no effect in this mode. Perform the following to enable up to 128 channels:

- ❑ Enable the selected channels in the XCERE0–XCERE3 and RCERE0–RCERE3.
- ❑ Set RMCME = XMCME = 1 in the MCR.
- ❑ Set RMCM, XMCM in the MCR as desired.

The following are descriptions of how each XMCM value affects operation in the enhanced multichannel selection mode (similar to its function in normal multichannel selection mode):

- ❑ 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 XCERE0–XCERE3. 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 XCERE0–XCERE3 (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 oc-

cur at their respective times. However, DX is driven only for those elements that are selected via XCERE0–XCERE3 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. Select desired receive channels by setting the RCERE0–RCERE3. The elements enabled in XCERE0–XCERE3 can be either a subset of or the same as those selected in RCERE0–XCERE3. 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 RCERE0–RCERE3. 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 RCERE0–RCERE3 (and *not* the number of elements programmed in RFRLEN1). For transmitting, DXR is loaded, and DXR-to-XSR copy occurs for all the elements that are enabled via RCERE0–RCERE3. However, DX is driven only for those elements that are selected via XCER. Therefore, if XINTM = 00b, transmit interrupts to the CPU would be generated the same number of times as the number of elements selected in RCERE0–RCERE3 (not XCERE0–XCERE3).

12.6.4.1 Channel Enable Registers for Enhanced Multichannel Selection

The enhanced receive channel enable registers (RCERE0, RCERE1, RCERE2, RCERE3) and the enhanced transmit channel enable registers (XCERE0, XCERE1, XCERE2, XCERE3) are used to enable any of the 128 elements for receive and transmit, respectively. Partitions A and B do not apply to the enhanced multichannel selection mode. Therefore, the bit fields in the enhanced receive/transmit channel enable registers are numbered from 0 to 127, representing the 128 channels. Table 12–24 shows the 128 channels in a multichannel data stream and their corresponding enable bits in registers R/XCEREx. Figure 12–50 and Figure 12–51 show the RCEREx and XCEREx registers, respectively. Table 12–25 describes the bit fields in these registers.

Figure 12–50. Enhanced Receive Channel Enable Registers

Enhanced Receive Channel Enable Register 0 (RCERE0)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RCE 31	RCE 30	RCE 29	RCE 28	RCE 27	RCE 26	RCE 25	RCE 24	RCE 23	RCE 22	RCE 21	RCE 20	RCE 19	RCE 18	RCE 17	RCE 16
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
RCE 15	RCE 14	RCE 13	RCE 12	RCE 11	RCE 10	RCE 9	RCE 8	RCE 7	RCE 6	RCE 5	RCE 4	RCE 3	RCE 2	RCE 1	RCE 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

Enhanced Receive Channel Enable Register 1 (RCERE1)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RCE 63	RCE 62	RCE 61	RCE 60	RCE 59	RCE 58	RCE 57	RCE 56	RCE 55	RCE 54	RCE 53	RCE 52	RCE 51	RCE 50	RCE 49	RCE 48
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
RCE 47	RCE 46	RCE 45	RCE 44	RCE 43	RCE 42	RCE 41	RCE 40	RCE 39	RCE 38	RCE 37	RCE 36	RCE 35	RCE 34	RCE 33	RCE 32
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

Enhanced Receive Channel Enable Register 2 (RCERE2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RCE 95	RCE 94	RCE 93	RCE 92	RCE 91	RCE 90	RCE 89	RCE 88	RCE3 87	RCE 86	RCE 85	RCE 84	RCE 83	RCE 82	RCE 81	RCE 80
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
RCE 79	RCE 78	RCE 77	RCE 76	RCE 75	RCE 74	RCE 73	RCE 72	RCE 71	RCE 70	RCE 69	RCE 68	RCE 67	RCE 66	RCE 65	RCE 64
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

Enhanced Receive Channel Enable Register 3 (RCERE3)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
RCE 127	RCE 126	RCE 125	RCE 124	RCE 123	RCE 122	RCE 121	RCE 120	RCE 119	RCE 118	RCE 117	RCE 116	RCE 115	RCE5 114	RCE 113	RCE 112
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
RCE 111	RCE 110	RCE 109	RCE 108	RCE 107	RCE 106	RCE 105	RCE 104	RCE 103	RCE 102	RCE 101	RCE 100	CE 99	RCE 98	RCE 97	RCE 96
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 12–51. Enhanced Transmit Channel Enable Registers

Enhanced Transmit Channel Enable Register 0 (XCERE0)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
XCE 31	XCE 30	XCE 29	XCE 28	XCE 27	XCE 26	XCE 25	XCE 24	XCE 23	XCE 22	XCE 21	XCE 20	XCE 19	XCE 18	XCE 17	XCE 16
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
XCE 15	XCE 14	XCE 13	XCE 12	XCE 11	XCE 10	XCE 9	XCE 8	XCE 7	XCE 6	XCE 5	XCE 4	XCE 3	XCE 2	XCE 1	XCE 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

Enhanced Transmit Channel Enable Register 1 (XCERE1)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
XCE 63	XCE 62	XCE 61	XCE 60	XCE 59	XCE 58	XCE 57	XCE 56	XCE 55	XCE 54	XCE 53	XCE 52	XCE 51	XCE 50	XCE 49	XCE 48
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
XCE 47	XCE 46	XCE 45	XCE 44	XCE 43	XCE 42	XCE 41	XCE 40	XCE 39	XCE 38	XCE 37	XCE 36	XCE 35	XCE 34	XCE 33	XCE 32
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

Enhanced Transmit Channel Enable Register 2 (XCERE2)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
XCE 95	XCE 46	XCE 45	XCE 44	XCE 43	XCE 42	XCE 41	XCE 40	XCE3 9	XCE 38	XCE 37	XCE 36	XCE 35	XCE3 4	XCE 33	XCE 80
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
XCE 79	XCE 78	XCE 77	XCE 76	XCE 75	XCE 74	XCE 73	XCE 72	XCE 71	XCE 70	XCE 69	XCE 68	XCE 67	XCE 66	XCE 65	XCE 64
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

Enhanced Transmit Channel Enable Register 3 (XCERE3)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
XCE 127	XCE 126	XCE 125	XCE 124	XCE 123	XCE 122	XCE 121	XCE 120	XCE 119	XCE 118	XCE 117	XCE 116	XCE 115	XCE 114	XCE 113	XCE 112
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
XCE 111	XCE 110	XCE 109	XCE 108	XCE 107	XCE 106	XCE 105	XCE 104	XCE 103	XCE 102	XCE 101	XCE 100	XCE 99	XCE 98	XCE 97	XCE 96
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 12–24. Enhanced Receive/Transmit Channel Enable Register (R/XCEREx) Field Description

Name	Function
RCE*n* 0 ≤ *n* ≤ 127	Receive channel enable RCE*n* = 0: Disables reception of the *n*th channel RCE*n* = 1: Enables reception of the *n*th channel.
XCE*n* 0 ≤ *n* ≤ 127	Transmit channel enable XCE*n* = 0: Disables transmission of the *n*th channel. XCE*n* = 1: Enables transmission of the *n*th channel.

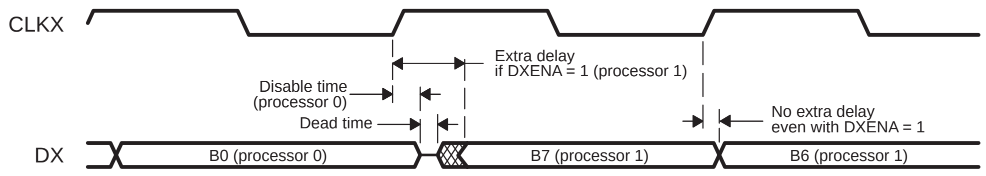
Table 12–25. Channel Enable Bits (in RCEREx/XCEREx) for a 128-Channel Data Stream

	Channel Number of a 128-Channel Data Stream							
	0 - 15	**16 - 31**	**32 - 47**	**48 - 63**	**64 - 79**	**80 - 95**	**96 - 111**	**112–127**
Register	RCERE0 XCERE0	RCERE0 XCERE0	RCERE1 XCERE1	RCERE1 XCERE1	RCERE2 XCERE2	RCERE2 XCERE2	RCERE3 XCERE3	RCERE3 XCERE3
Channel	R/XCE0 to R/XCE15	R/XCE16 to R/XCE31	R/XCE32 to R/XCE47	R/XCE48 to R/XCE63	R/XCE64 to R/XCE79	R/XCE80 to R/XCE95	R/XCE96 to R/XCE111	R/XCE112 to R/XCE127

12.6.5 DX Enabler: DXENA

The DX enabler is only available for the C621x/C671x/C64x 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 12–52.

Figure 12–52. 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 12–52, 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 some 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 some extra CPU cycles before the data is seen on the TDM data line. Therefore only the first bit of data is delayed. Refer to the specific device datasheet for the exact amount of delay.

12.7 SPI Protocol: CLKSTP

A system conforming to this protocol has a master-slave configuration. 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 12–53 and Figure 12–54, respectively.

Figure 12–53. SPI Configuration: McBSP as the Master

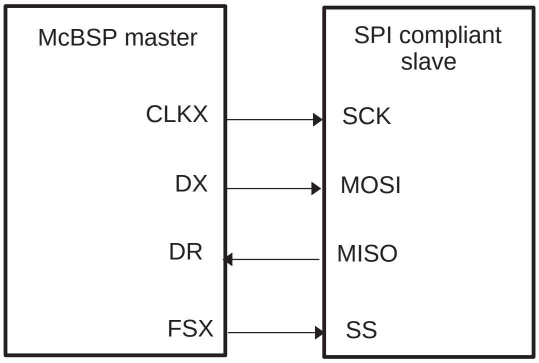
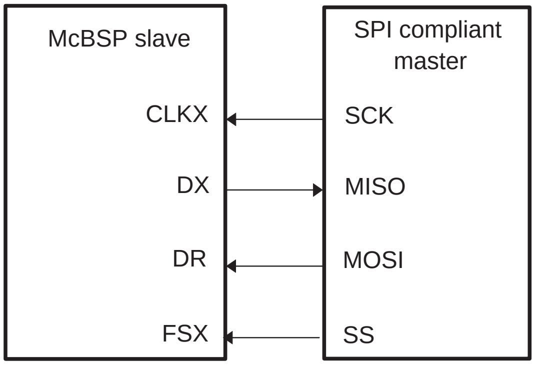


Figure 12–54. SPI Configuration: McBSP as the Slave



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

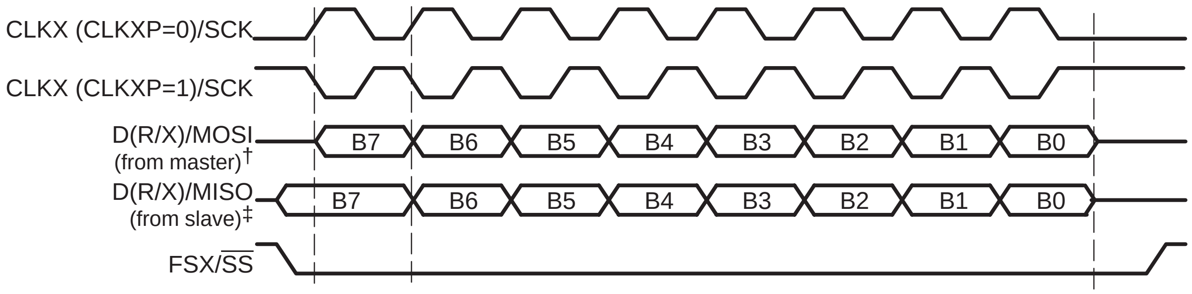
(CLKSTP) in conjunction with the CLKXP bit in the PCR allows serial clocks to be stopped between transfers using one of four possible timing variations, as shown in Table 12–26. Figure 12–55 and Figure 12–56 show the timing diagrams of the two SPI transfer formats and the four timing variations.

Note: The digital loopback mode (DLB =1 in the SPCR cannot be used in conjuntion with the clock stop mode (CLKSTP = 1x).

Table 12–26. SPI-Mode Clock Stop Scheme

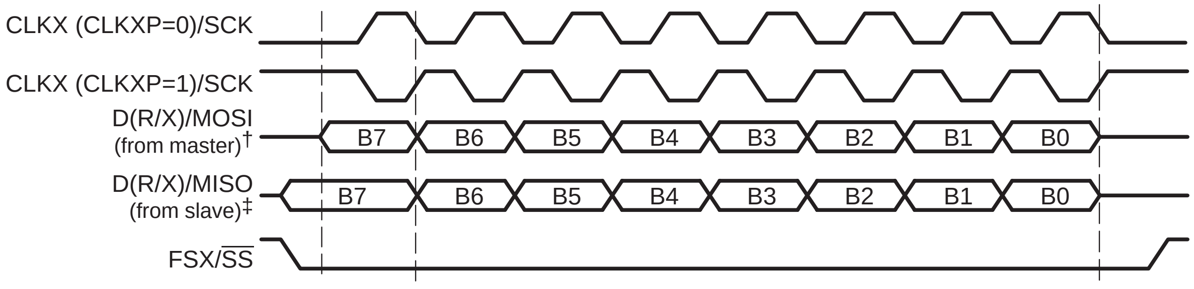
CLKSTP	CLKXP	Clock Scheme
0X	X	Clock stop mode disabled. Clock enabled for non-SPI mode.
10	0	Low inactive state without delay. The McBSP transmits data on the rising edge of CLKX and receives data on the falling edge of CLKR.
11	0	Low inactive state with delay. The McBSP transmits data one-half cycle 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 12–55. SPI Transfer with CLKSTP = 10b



† 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.

Figure 12–56. 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 12–26. 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 12–26.

Figure 12–55 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 12–56 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. The FSR and CLKR signals should not be used in SPI mode. These do not function as SPI signals like the FSX and CLKX signals. 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).

12.7.1 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 (FSGM=0 in SRGR). Therefore to receive an element in SPI master mode, the McBSP must also simultaneously transmit an element (write to the DXR) in order to generate the necessary slave enable FSX. The FSX needs to be asserted (low) to enable the slave before the McBSP starts shifting out data on the DX pin. Refer to the MOSI and FSX waveforms in Figure 12–55 and Figure 12–56. Therefore, XDATDLY and RDATDLY must be programmed to 1. When the McBSP is the SPI master, an XDATDLY value of 0 or 2 causes undefined operation. An RDATDLY of 0 causes the recevied data to be shifted incorrectly.

As the SPI master, the McBSP generates CLKX and FSX through the internal sample rate generator. As discussed in section 12.5.2.1, 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).

12.7.2 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 12–54. 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 12–55 and Figure 12–56 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 12–26.

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.

12.7.3 McBSP Initialization for SPI Mode

The operation of the serial port during device reset, transmitter reset, and receiver reset is described in section 12.3.1. 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 12–3 as required when the serial port is in the reset state ($\overline{\text{XRST}}$ = $\overline{\text{RRST}}$ = 0). Write the desired value into the CLKSTP field in the SPCR. Table 12–26 shows the various CLKSTP modes.
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) Depending upon whether the CPU or DMA services the McBSP, perform step (a) if the CPU is used, or step (b) if the DMA is used.
   a) If the CPU is used to service the McBSP. set $\overline{\text{XRST}}$ = $\overline{\text{RRST}}$ = 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 bitfields 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.
6) Wait two bit clocks for the receiver and transmitter to become active.

12.8 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 12–3 shows the PCR bits that configure each of the McBSP pins as general-purpose inputs or outputs. Table 12–27 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 12–27. 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 13

Timers

This chapter describes the 32-bit timer functionality, registers, and signals.

Topic **Page**

13.1 Overview

The device has 32-bit general-purpose timers that can be used to:

- Time events
- Count events
- Generate pulses
- Interrupt the CPU
- Send synchronization events to the DMA.

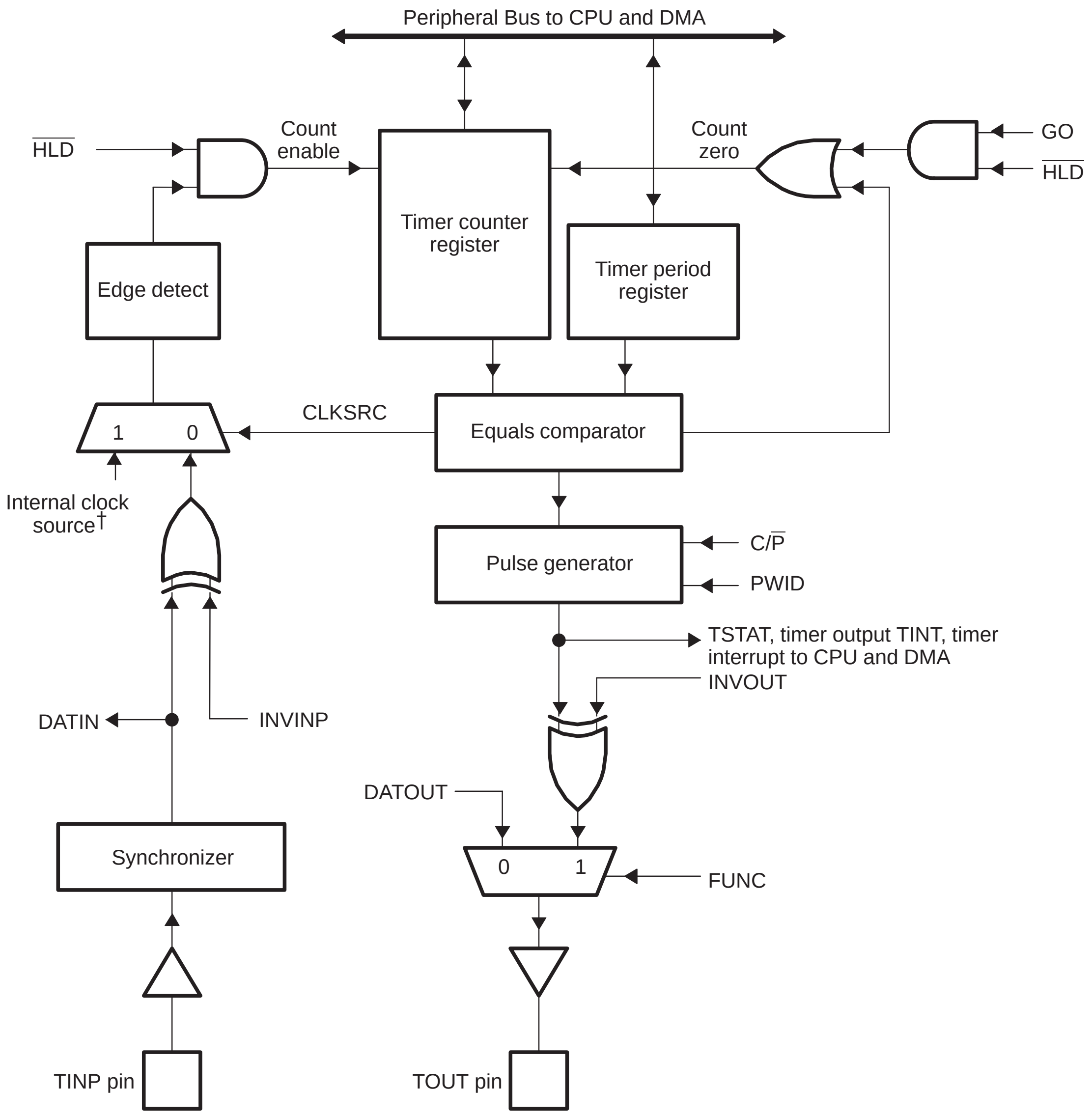
The timers have two signaling modes and can be clocked by an internal or an external source. The timers have an input pin and an output pin. The input and output pins, (TINP and TOUT) can function as timer clock input and clock output. They can also be configured for general-purpose input and output, 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. Table 13–1 summarizes the differences between the C6000 timers. Figure 13–1 shows a block diagram of the timers.

Table 13–1. Differences in TMS320C6000 Timers

Features	C620x/C670x	C621x/C671x	C64x	Section
Emulation halt support	yes	no	no	13.10
Internal timer input clock source frequency	CPU rate/4	CPU rate/4	CPU rate/8	13.5

Figure 13–1. Timer Block Diagram



† C62x/C67x uses CPU/4 clock as the internal clock source to the timer.
C64x uses CPU/8 clock as the internal clock source to the timer.

13.2 Timer Registers

Table 13–2 describes the three registers that configure timer operation.

Table 13–2. Timer Registers

Hex Byte Address			Name and Abbreviation	Description	Section
Timer 0	**Timer 1**	**Timer 2**			
01940000	01980000	01AC0000	Timer Control (CTL)	Determines the operating mode of the timer, monitors the timer status, and controls the function of the TOUT pin.	13.2.1
01940004	01980004	01AC0004	Timer Period (PRD)	Contains the number of timer input clock cycles to count. This number controls the TSTAT signal frequency.	13.2.2
01940008	01980008	01AC0008	Timer Counter (CNT)	Current value of the incrementing counter	13.2.3

13.2.1 Timer Control Register (CTL)

Figure 13–2 shows the timer control register. Table 13–3 describes the fields in this register.

Figure 13–2. Timer Control Register (CTL)

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	R, +X	RW, +0	RW, +0	RW, +0

Table 13–3. Timer Control Register (CTL) Field Descriptions

No.	Bitfield	Description	Section
31–12	Rsvd	Reserved.	
11	TSTAT	Timer status. Value of timer output.	13.6

10	INVINP	TINP inverter control. Only affects operation if CLKSRC = 0.
		INVINP = 0: Uninverted TINP drives timer.
		INVINP = 1: Inverted TINP drives timer.
			13.5

9	CLKSRC	Timer input clock source CLKSRC = 0: External clock source drives the TINP pin. CLKSRC = 1: Internal clock source. For C62x/C67x: CPU clock/4 For C64x: CPU clock/8	13.5
8	$C/\overline{P}$	Clock/pulse mode $C/\overline{P}$ = 0: Pulse mode. TSTAT is active one CPU clock after the timer reaches the timer period. PWID determines when it goes inactive. $C/\overline{P}$ = 1: Clock mode. TSTAT has a 50% duty cycle with each high and low period one countdown period wide.	13.6
7	$\overline{HLD}$	Hold. Counter may be read or written regardless of $\overline{HLD}$ value. $\overline{HLD}$ = 0: Counter is disabled and held in the current state. $\overline{HLD}$ = 1: Counter is allowed to count.	13.3
6	GO	GO bit. Resets and starts the timer counter. GO = 0: No effect on the timers. GO = 1: If $\overline{HLD}$ = 1, the counter register is zeroed and begins counting on the next clock.	13.3
5	Rsvd	Reserved.	
4	PWID	Pulse width. Only used in pulse mode ($C/\overline{P}$ = 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.	13.6
3	DATIN	Data in: Value on TINP pin	13.5, 13.9
2	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.	13.9

Table 13–3. Timer Control Register (CTL) Field Descriptions (Continued)

No.	Bitfield	Description	Section

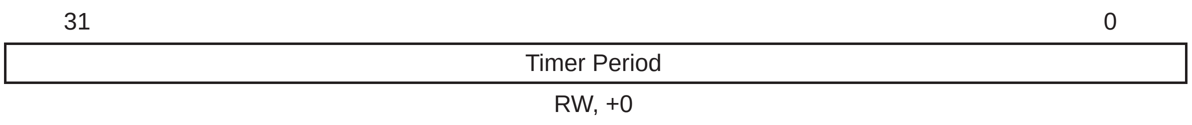
1	INVOUT	TOUT inverter control. Used only if FUNC = 1.
		
		INVOUT = 0: Uninverted TSTAT drives TOUT.
		INVOUT = 1: Inverted TSTAT drives TOUT.
			13.9

0	FUNC	Function of TOUT pin FUNC = 0: TOUT is a general-purpose output pin. FUNC = 1: TOUT is a timer output pin.	13.6

13.2.2 Timer Period Register (PRD)

The timer period register (Figure 13–3) contains the number of timer input clock cycles to count. This number controls the frequency of TSTAT.

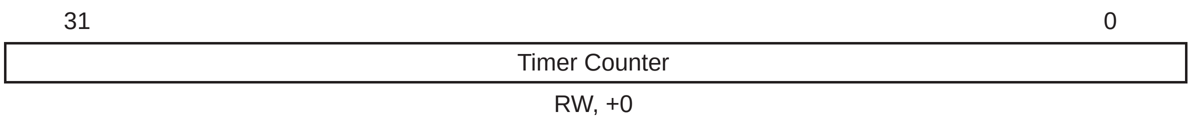
Figure 13–3. Timer Period Register (PRD)



13.2.3 Timer Counter Register (CNT)

The timer counter register (Figure 13–4) 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 13–4. Timer Counter Register (CNT)



13.3 Resetting the Timers and Enabling Counting: GO and $\overline{\text{HLD}}$

Table 13–4 shows how the GO and $\overline{\text{HLD}}$ enable basic features of timer operation.

Table 13–4. 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 (PRD).
3) Write the desired value to the timer control register (CTL). Do not change the GO and $\overline{\text{HLD}}$ bits of the CTL in this step.
4) Start the timer by setting the GO and $\overline{\text{HLD}}$ bits of the CTL to 1.

13.4 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 13–1. 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 (PRD), 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) for C62x/C67x. 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.

13.5 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.
- ❑ Internal clock source, selected by CLKSRC = 1. The C62x/C67x uses CPU clock/4 as an internal clock source. The C64x uses CPU clock/8 as an internal clock source.

13.6 Timer Pulse Generation

The two basic pulse generation modes are pulse mode and clock mode, as shown in Figure 13–5 and Figure 13–6, respectively. You can select the mode with the C/$\overline{\text{P}}$ bit of the timer control register (CTL). Note that in pulse mode, PWID in the CTL 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. Table 13–5 gives equations for various TSTAT timing parameters in pulse and clock modes.

Figure 13–5. Timer Operation in Pulse Mode (C/$\overline{P}$ = 0)

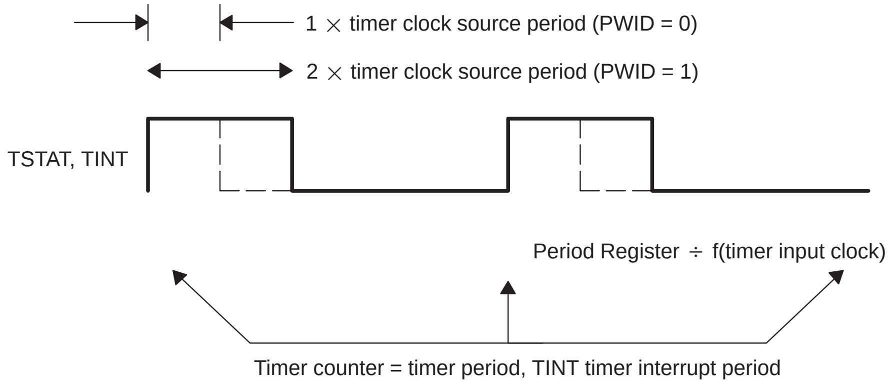

1 × timer clock source period (PWID = 0)
2 × timer clock source period (PWID = 1)
TSTAT, TINT
Period Register ÷ f(timer input clock)
Timer counter = timer period, TINT timer interrupt period

Figure 13–6. Timer Operation in Clock Mode (C/$\overline{P}$ = 1)

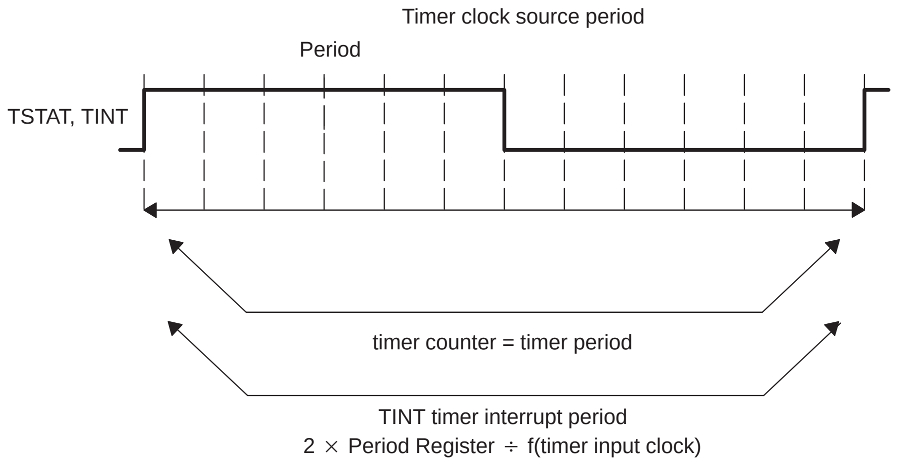

Timer clock source period
Period
TSTAT, TINT
timer counter = timer period
TINT timer interrupt period
2 × Period Register ÷ f(timer input clock)

Table 13–5. TSTAT Parameters in Pulse and Clock Modes

Mode	Frequency	Period	Width High	Width Low
Pulse	$\frac{\textit{f (clock source)}}{\textit{timer period register}}$	$\frac{\textit{timer period register}}{\textit{f (clock source)}}$	$\frac{\textit{(PWID + 1)}}{\textit{f (clock source)}}$	$\frac{\textit{timer period register – (PWID + 1)}}{\textit{f (clock source)}}$
Clock	$\frac{\textit{f (clock source)}}{\textit{2 * timer period register}}$	$\frac{\textit{2 * timer period register}}{\textit{f (clock source)}}$	$\frac{\textit{timer period register}}{\textit{f (clock source)}}$	$\frac{\textit{timer period register}}{\textit{f (clock source)}}$

13.7 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 timer counter register (CNT) is set to a value greater than the value of the timer period register (PRD) 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 CNT and new status updates to the timer control register (CTL).

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.

13.8 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.

13.9 Timer Pins as General-Purpose Input/Output

Upon device reset, the timer pins TINP and TOUT are general-purpose input and output (I/O) pins, respectively. By configuring the timer control register (CTL), the TINP and TOUT pins can operate as general-purpose pins even when the timer is running.

The TINP pin is always a general-purpose input pin if the timer is not running. If the timer is running, the TINP pin is a general-purpose input pin if CLKSRC = 1 in the CTL, which indicates that an internal clock source is used instead of the TINP pin. When TINP is a general-purpose input pin, the input value is readable on the DATIN bit field.

The TOUT pin is a general-purpose output pin if FUNC = 0 in the CTL, independent of timer operation. The FUNC bit field, as shown in Figure 13–1, selects either the DATOUT or the TSTAT value to be driven on the TOUT pin.

13.10 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. For C620x/C670x, 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. For C621x/C671x/C64x, the timer continues counting during emulation halt regardless of clock source.

Chapter 14

Interrupt Selector and External Interrupts

This chapter describes the interrupt selector and registers available.

Topic **Page**

14.1 Overview

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 14–1 summarizes the differences between the interrupt selectors of the C6000 devices.

Table 14–1. Differences in C6000 Interrupt Selectors

Features	Supported on Device	Section
Available Interrupts	C6000 devices have different available interrupts according to their peripheral sets	14.2
IACK and INUM pins	C620x/C670x only	14.3
EXT_INT4–7 pins	All C6000. On C64x, these pins are MUXed with the GPIO peripheral pins	14.3

14.2 Available Interrupt Sources

Table 14–2, Table 14–3, and Table 14–4 list the available interrupts of the C620x/C670x, C621x/C671x, and C64x, respectively. For more information on interrupts, including the interrupt vector table, see the *TMS320C6000 CPU and Instruction Set Reference Guide*.

Table 14–2. TMS320C620x/C670x 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†
	PCI_WAKEUP	PCI wake up interrupt‡
10010b	RINT2	McBSP 2 receive interrupt†
	ADMA_HLT	Auxiliary DMA halted interrupt‡
other		Reserved

† Only available on the C6202(B), C6203(B)
‡ Only available on the C6205

Table 14–3. TMS320C621x/C671x 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

Table 14–4. TMS320C64x 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_INTA	EMIFA SDRAM timer interrupt
00100b	GPINT4/EXT_INT4	GPIO interrupt 4/External interrupt 4
00101b	GPINT5/EXT_INT5	GPIO interrupt 5/External interrupt 5
00110b	GPINT6/EXT_INT6	GPIO interrupt 6/External interrupt 6
00111b	GPINT7/EXT_INT7	GPIO interrupt 7/External interrupt 7
01000b	EDMA_INT	EDMA channel (0 through 63) 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
10000b	GPINT0	GPIO interrupt 0
10001b	XINT2	McBSP 2 transmit interrupt
10010b	RINT2	McBSP 2 receive interrupt
10011b	TINT2	Timer 2 interrupt
10100b	SD_INTB	EMIFB SDRAM timer interrupt
10101b	PCI_WAKEUP	PCI wakeup interrupt
10110b	Reserved	Not used
10111b	UINT	UTOPIA interupt
other		Reserved

14.3 External Interrupt Signal Timing

EXT_INT4–7 and NMI are dedicated external interrupt sources. (EXT_INT4–7 pins are MUXed with the GPIO peripheral pins on C64x.) In addition, the FSR and FSX can be programmed to directly drive the RINT and XINT signals. Because these signals are asynchronous, they are synchronized before being sent to either the DMA/EDMA or CPU. Refer to the *TMS320C6000 CPU and Instruction Set Reference Guide* (SPRU189) and the device datasheet for details on external interrupt signals timing.

For the C620x/C670x, 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.

Note:

The IACK and INUM pins do not exist on C621x/C671x/C64x. They only exist on C620x/C670x.

14.4 Interrupt Selector Registers

Table 14–5 shows the interrupt selector registers. The interrupt multiplexer registers determine the mapping between the interrupt sources described in section 14.2 and the CPU interrupts 4 through 15 (INT4–INT15). The external interrupt polarity register sets the polarity of external interrupts.

Table 14–5. Interrupt Selector Registers

Byte Address	Abbreviation	Name	Description	Section
019C0000h	MUXH	Interrupt multiplexer high	Selects which interrupts drive CPU interrupts 10–15 (INT10–15)	14.4.2
019C0004h	MUXL	Interrupt multiplexer low	Selects which interrupts drive CPU interrupts 4–9 (INT4–INT9)	14.4.2
019C0008h	EXTPOL	External interrupt polarity	Sets the polarity of the external interrupts (EXT_INT4–EXT_INT7)	14.4.1

14.4.1 External Interrupt Polarity Register (EXTPOL)

The external interrupt polarity register, shown in Figure 14–1, allows the user 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–INT7) 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 14–1. External Interrupt Polarity Register (EXTPOL)

31 … 4	3	2	1	0
Rsvd	XIP7	XIP6	XIP5	XIP4
R, +0	RW, +0	RW, +0	RW, +0	RW, +0

14.4.2 Interrupt Multiplexer Register

The INTSEL fields in the interrupt multiplexer registers, shown in Figure 14–2 for the low range and Figure 14–3 for the high range, 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 14–2, Table 14–3, or Table 14–4, the user can map any interrupt source to any CPU interrupt. Default mapping of interrupt sources to CPU interrupts are shown in Table 14–6.

Figure 14–2. Interrupt Multiplexer Low Register (MUXL)

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 14–3. Interrupt Multiplexer High Register (MUXH)

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 14–6. 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 (C62x/C67x)
			SD_INTA	EMIFA SDRAM timer interrupt (C64x)
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 C621x/C671x
‡ Reserved on C64x

14.5 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 15

Power-Down Logic

The power-down modes are described in this chapter.

Topic **Page**

15.1 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, the Power–Down Modes can be used to achieve significant power savings without losing any data or operation context. PD1, PD2, and PD3 are three power–down modes available on the C6000 to perform this function. In addition to PD1, PD2, and PD3 the C6202(B)/C6203(B) also has a peripheral power–down mode, as discussed in section 15.4. Table 15–1 summarizes the differences between the power–down modes in the C6000 devices.

Table 15–1. Differences in C6000 Power–Down Modes

Features	**Supported on Device**	**Section**
PD pin	C620x/C670x only	15.2
Peripheral Power–Down Mode	C6202(B)/C6203(B) only	15.4

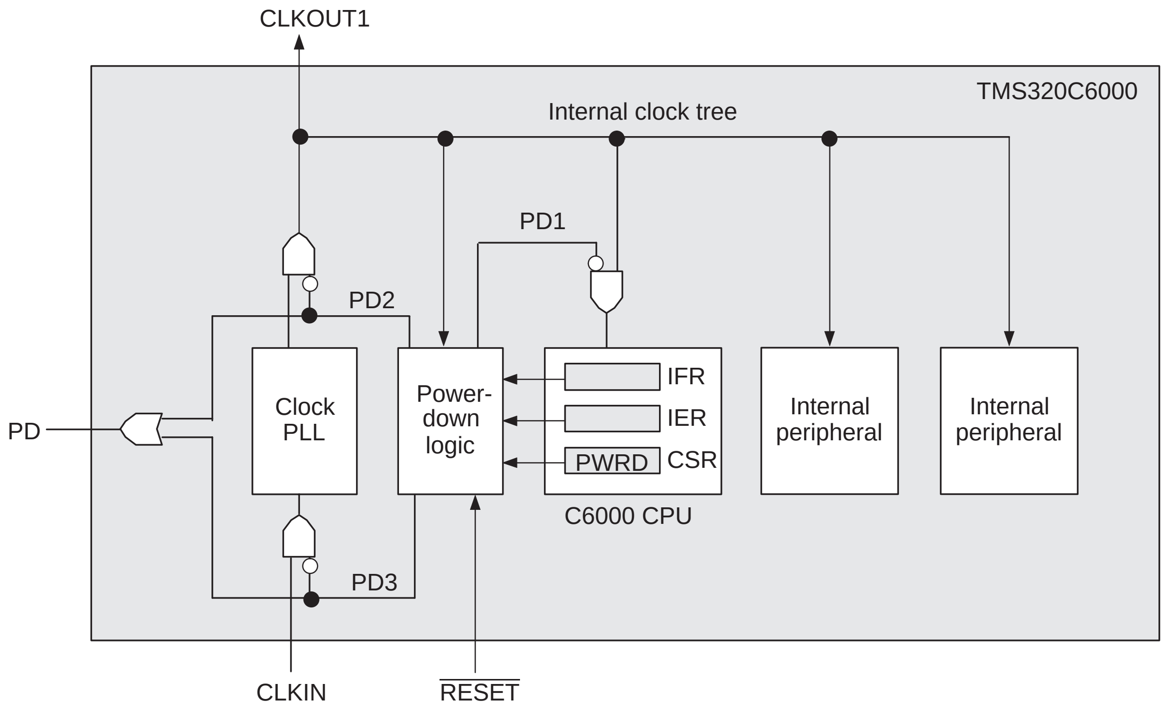
15.2 Power–Down Mode Descriptions

Figure 15–1 shows the power–down logic on a C6000 device. 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. During PD1, DMA/EDMA transactions can proceed between peripherals and internal memory.

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 15–1). 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 C620x/C670x, both the PD2 and PD3 signals also assert the PD pin for external recognition of these two power–down modes. Although the C621x/C671x/C64x 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 15–1. Power-Down Mode Logic



15.3 Triggering, Wake-Up, and Effects

The power–down modes and their wake–up methods are programmed by setting the PWRD field (bits 10–15) of the control status register (CSR). The PWRD field of the CSR is shown in Figure 15–2 and described in Table 15–2. When writing to the CSR, all bits of the PWRD field should be set at the same time. Logic 0 should be used when writing to the reserved bit (bit 15) of the PWRD field. The CSR is discussed in detail in the *TMS320C6000 CPU and Instruction Set Reference Guide (SPRU189)*.

Figure 15–2. 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	
	RW, +0	RW, +0	RW, +0	RW, +0	RW, +0	RW, +0	

Note: Refer to the *TMS320C6000 CPU and Instruction Set Reference Guide (SPRU189)* for other bit fields in the CSR.

Table 15–2. 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	wake by a device reset
011100	PD3	wake by a device reset
other	reserved	—

Power-down mode PD1 takes effect eight to nine clock cycles after the instruction that caused the power down (by setting the PWRD 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 execution resumed here
```

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 5. Wake-up by an enabled interrupt executes the corresponding interrupt service fetch packet (ISFP) first, prior to returning to the instruction following the NOP 5. CSR register GIE bit and interrupt enable register (IER) 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.

PD2 and PD3 modes can only be aborted by device reset. Table 15–3 summarizes all the power–down modes.

Table 15–3. 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 functional I/O "freeze" in the last state when the PLL clock is turned off.†
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 functional I/O "freeze" in the last state when the PLL clock is turned off.† Following reset, the PLL needs time to re-lock, just as it does following power-up.

† When entering PD2 and PD3, all functional I/O will remain in the previous state. However, for peripherals which are asynchronous in nature (HPI) or peripherals with an external clock source (McBSP, XBUS, timers, C621x/C671x/C64x EMIF, UTOPIA, PCI), output signals may transition in response to stimulus on the inputs. Peripheral operation is not guaranteed under these conditions.

15.4 Peripheral Power-Down Mode for TMS320C6202(B)/C6203(B)

In addition to the power down modes common to all of the C6x devices, the C6202(B)/C6203(B) 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 15–4 shows the peripheral power down register address location, and Figure 15–3 shows the register fields.

Table 15–4. Peripheral Power-Down Memory-Mapped Register

Byte Address	**Abbreviation**	**Field**
019C 0200h	PDCTL	Peripheral Power-Down Control

Figure 15–3. Peripheral Power-Down Control Register (PDCTL) for TMS320C6202(B)/C6203(B)

31 5	4	3	2	1	0
Reserved	PDMCSP2	PDMCSP1	PDMCSP0	PDEMIF	PDDMA
R,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0

Table 15–5 lists and describes the fields in the peripheral power-down control-register (PDCTL).

Table 15–5. Peripheral Power-Down Control Register Bit Field Descriptions

Bit No.	Field	Description	Section
4	PDMCSP2	Enable/disable internal McBSP2 clock PDMCSP2=0: internal McBSP2 clock allowed to clock PDMCSP2=1: internal McBSP2 clock disabled, McBSP2 is not functional	15.4
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.	15.4
2	PDMCSP0	Enable/disable internal McBSP0 clock PDMCSP0=0: Internal McBSP0 clock allowed to clock. PDMCSP0=1: Internal McBSP0 clock disabled. McBSP0 is not functional.	15.4
1	PDEMIF	Enable/disable internal EMIF clock PDEMIF=0: internal EMIF clock allowed to clock PDEMIF=1: Internal EMIF clock disabled. EMIF is not functional. HOLD condition which exists at power down will remain active and external clocks continue to clock.	15.4
0	PDDMA	Enable/disable internal DMA clock PDDMA=0: internal DMA clock allowed to clock PDDMA=1: internal DMA clock disabled. DMA is not functional	15.4

The user must be careful to not disable a portion of the device which is being used, since the peripheral becomes non–operational once disabled. 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. While a peripheral is in power–down mode, no writes to the peripheral's registers will occur; and reads from the peripheral will produce invalid data.

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 15–1.

Example 15–1. 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 16

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 3/5 V** and supports both standard 3-volt and 5-volt target system power inputs.

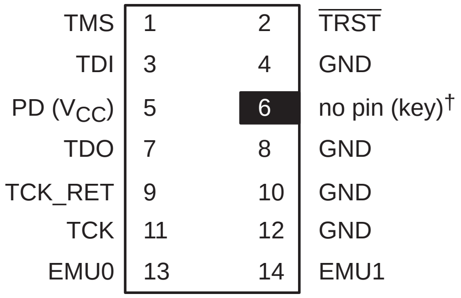
The term *JTAG,* as used in this book, refers to TI scan-based emulation, which is based on the IEEE 1149.1 standard.

Topic **Page**

16.1 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 16–1. Table 16–1 describes the emulation signals.

Figure 16–1. 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 16–1. 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

16.2 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.

16.3 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

16.4 JTAG Emulator Cable Pod Logic

Figure 16–2 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.

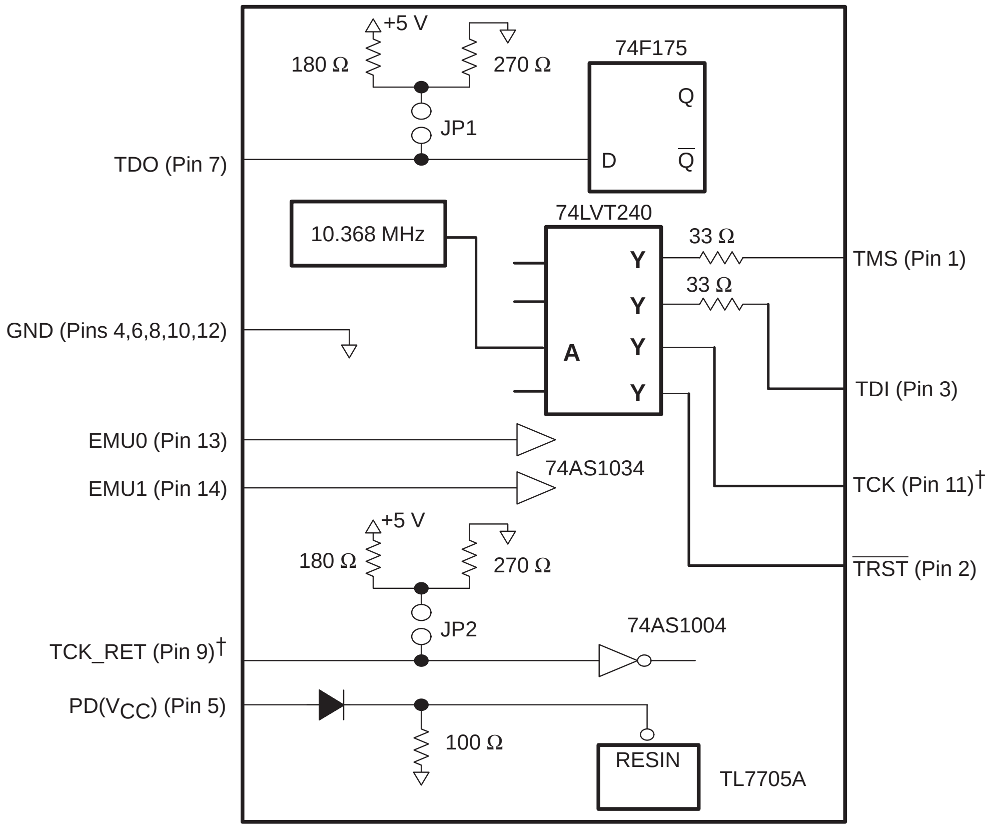
❑ Signal TCK is driven with a 74LVT240 device. Because of the high-current drive (32 mA I_{OL}/I_{OH}), this signal can be parallel-terminated. If TCK is tied to TCK_RET, then you can use the parallel terminator in the pod.

❑ 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 16–2. 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.

16.5 JTAG Emulator Cable Pod Signal Timing

Figure 16–3 shows the signal timings for the emulator cable pod. Table 16–2 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 16–3. JTAG Emulator Cable Pod Timings

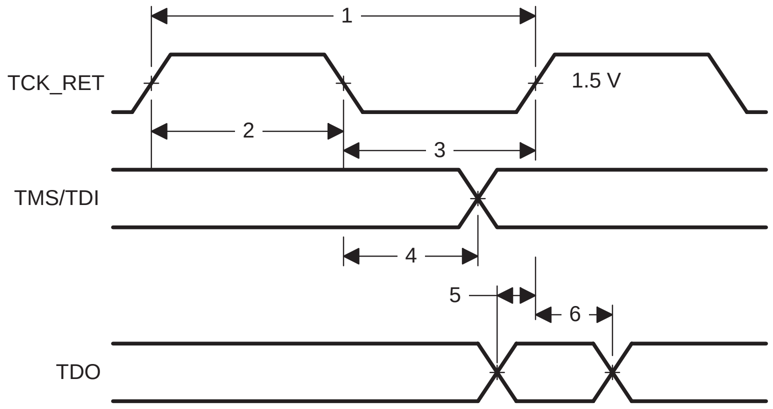


Table 16–2. Emulator Cable Pod Timing Parameters

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

16.6 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 16–2 (on page 16-5):

$t_{d(TMSmax)}$	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.

$$\begin{aligned} t_{pd\,(TCK\_RET–TMS/TDI)} &= \frac{\left[t_{d\,(TMSmax)} + t_{su\,(TTMS)}\right]}{t_{(TCKfactor)}} \\ &= \frac{[20ns + 10ns]}{0.4} \\ &= 75ns\ (13.3\ MHz) \end{aligned}$$

$$\begin{aligned} t_{pd\,(TCK\_RET–TDO)} &= \frac{\left[t_{d\,(TTDO)} + t_{su\,(TDOmin)}\right]}{t_{(TCKfactor)}} \\ &= \frac{[15ns + 3ns]}{0.4} \\ &= 45ns\ (22.2\ MHz) \end{aligned}$$

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:

$$\begin{aligned} t_r &= 5(R_{pullup} \times N_{devices} \times C_{load\_per\_device}) \\ &= 5(4.7\ k\Omega \times 16 \times 15\ pF) \\ &= 5.64\ \mu s \end{aligned}$$

Refer to the device datasheet for the actual R_{pullup} value.

16.7 Connections Between the Emulator and the Target System

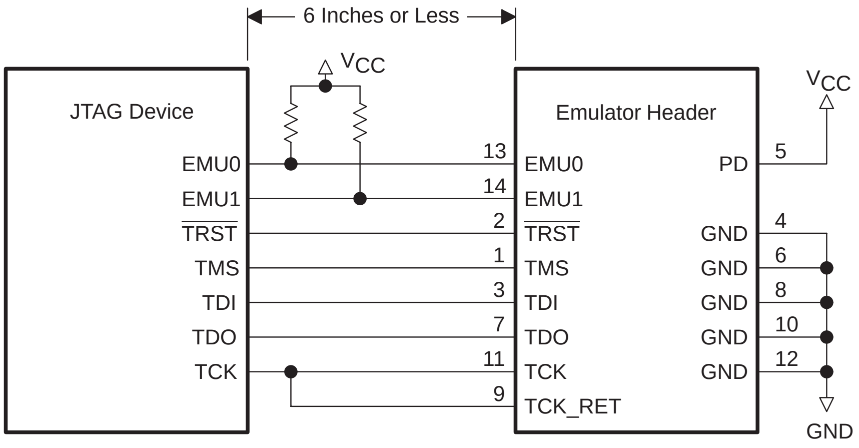
It is extremely important to provide high-quality signals between the emulator and the JTAG target system. Depending upon the situation, you must supply the correct signal buffering, test clock inputs, and multiple processor interconnections to ensure proper emulator and target system operation.

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.

16.7.1 Buffering Signals

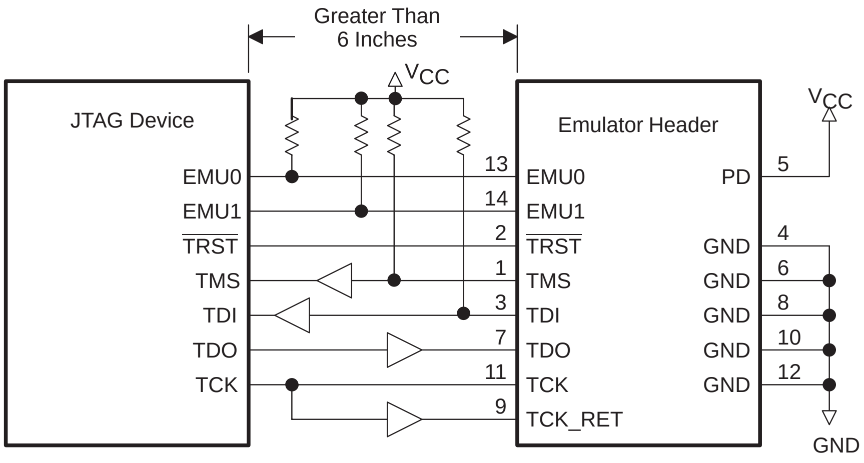
If the distance between the emulation header and the JTAG target device is greater than six inches, the emulation signals must be buffered. If the distance is less than six inches, no buffering is necessary. The following illustrations depict these two situations.

❑ **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. Refer to the device datasheet for the recommeded resistor value.

- ❑ **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.

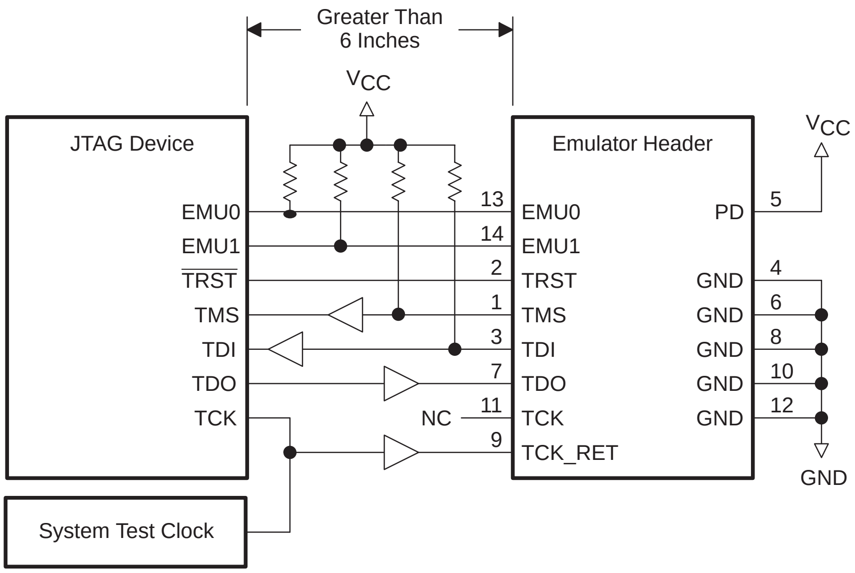


  - ■ The EMU0 and EMU1 signals must have pullup resistors connected to V_{CC} to provide a signal rise time of less than 10 µs. Refer to the device datasheet for the recommeded resistor value.
  - ■ The input buffers for TMS and TDI should have pullup resistors connected to V_{CC} to hold these signals at a known value when the emulator is not connected. Refer to the device datasheet for the recommeded resistor value.
  - ■ 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.

16.7.2 Using a Target-System Clock

Figure 16–4 shows an application with the system test clock generated in the target system. In this application, the TCK signal is left unconnected.

Figure 16–4. 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.

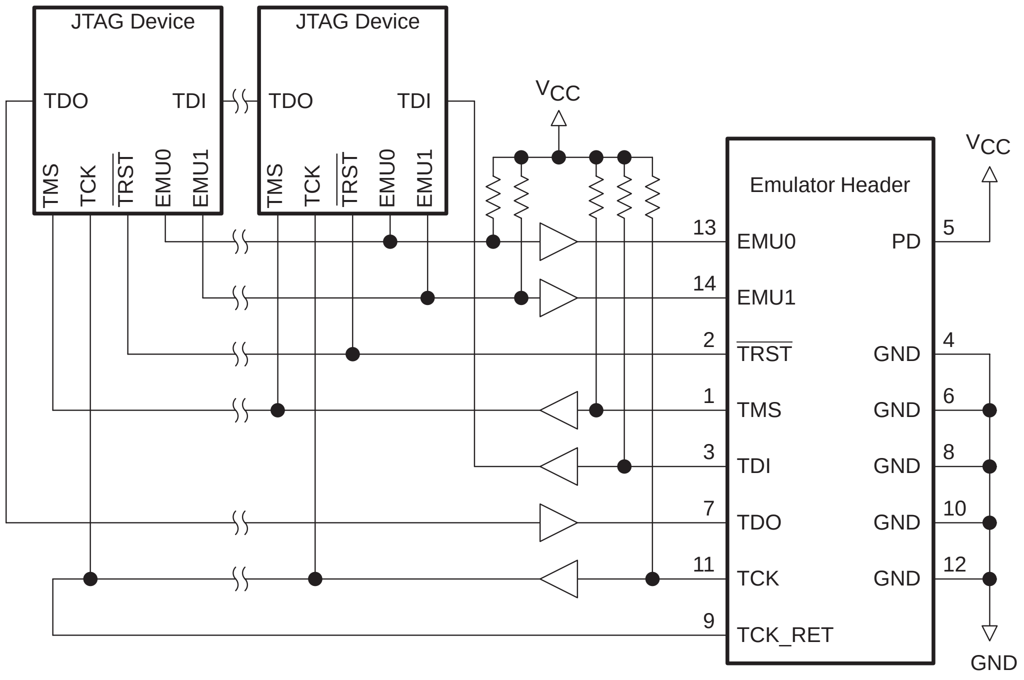
❑ In some cases, you may have other devices in your system that require a test clock when the emulator is not connected. The system test clock also serves this purpose.

16.7.3 Configuring Multiple Processors

Figure 16–5 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. Refer to the device datasheet for the recommeded resistor value.
- 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 16-8 and page 16-9).

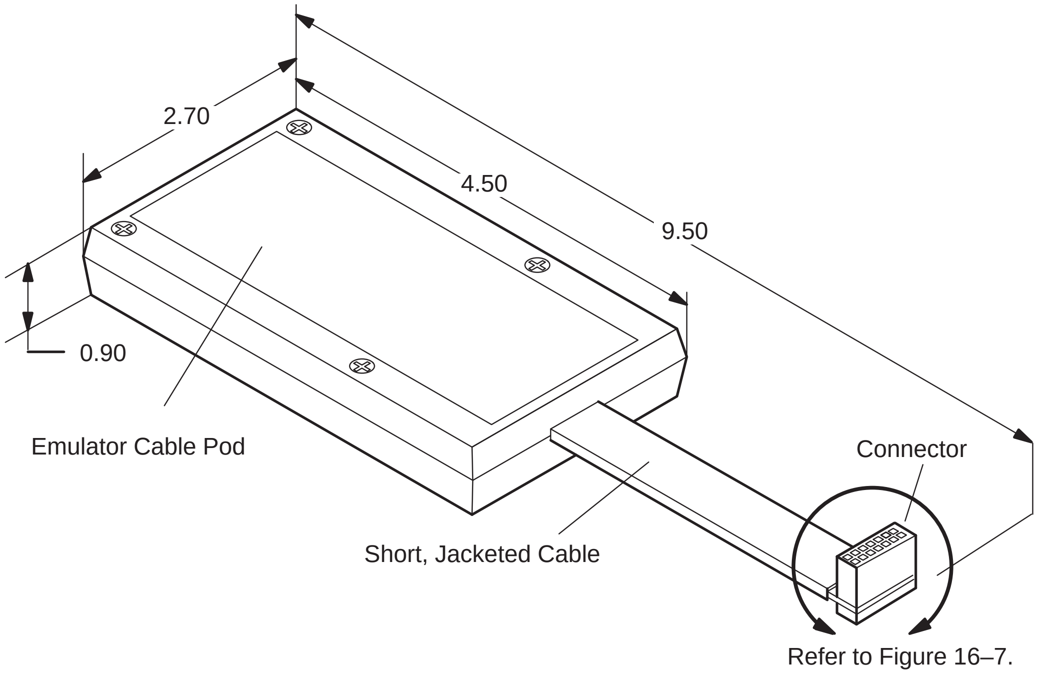
Figure 16–5. Multiprocessor Connections



16.8 Mechanical Dimensions for the 14-Pin Emulator Connector

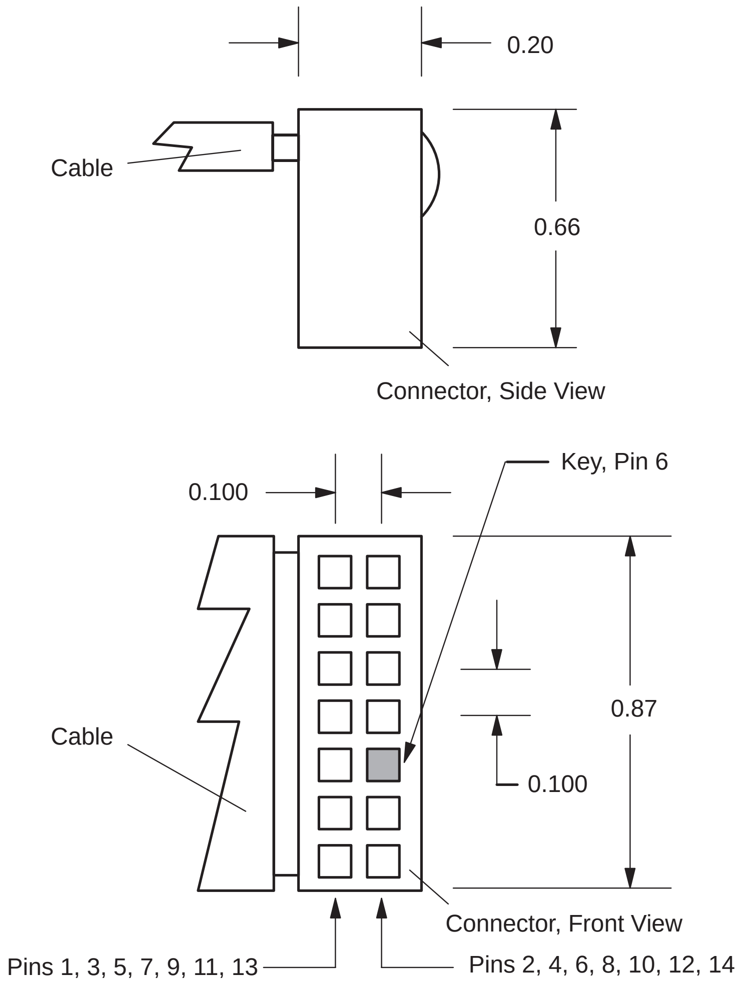
The JTAG emulator target cable consists of a 3-foot section of jacketed cable, an active cable pod, and a short section of jacketed cable that connects to the target system. The overall cable length is approximately 3 feet 10 inches. Figure 16–6 and Figure 16–7 (page 16-13) show the mechanical dimensions 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 16–6. Pod/Connector Dimensions



Note: All dimensions are in inches and are nominal dimensions, unless otherwise specified.

Figure 16–7. 14-Pin Connector Dimensions



Note: All dimensions are in inches and are nominal dimensions, unless otherwise specified.

16.9 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.

16.9.1 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.

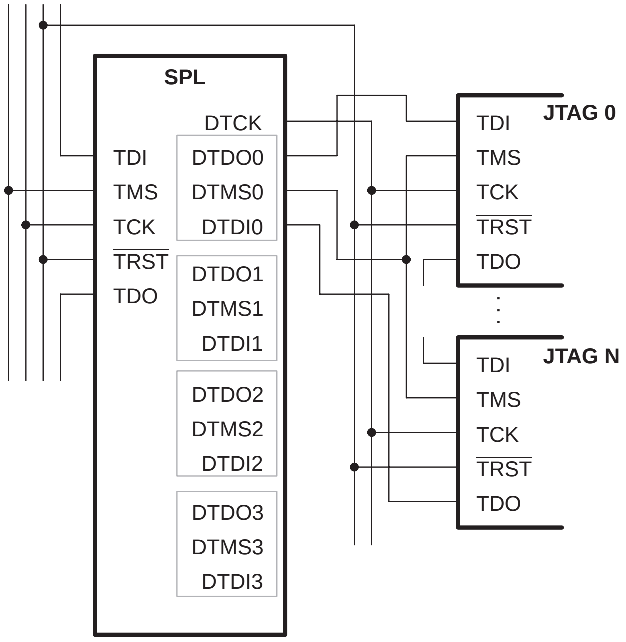
A system of multiple, secondary JTAG scan paths has better fault tolerance and isolation than a single scan path. Since an SPL has the capability of adding all secondary scan paths to the main scan path simultaneously, it can support global emulation operations, such as starting or stopping a selected 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 16–8 shows you how to connect a secondary scan path to an SPL.

Figure 16–8. Connecting a Secondary JTAG Scan Path to an SPL[†]



[†] Voltage translators should be used between the SPL (5V) and the C6000 (3V).

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.

16.9.2 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.

16.9.3 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:

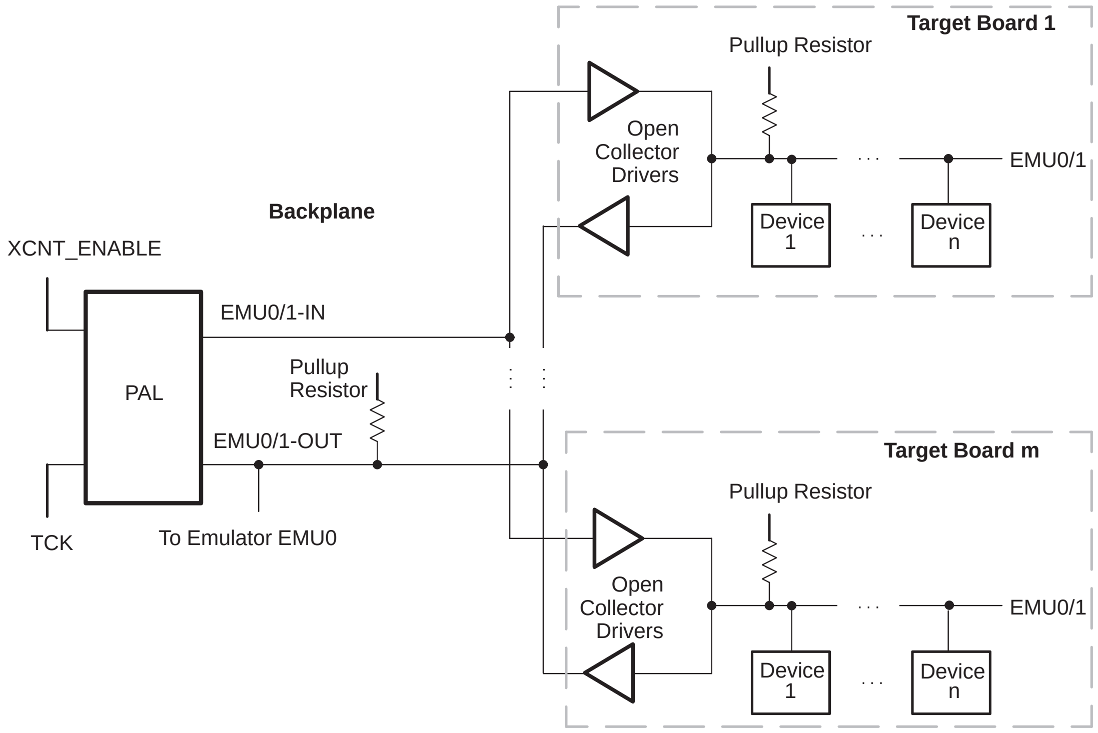
- **Signal Event**
  The EMU0/1 pins can be configured via software to signal internal events. In this mode, driving one of these pins low can cause devices to signal such events. To enable this operation, the EMU0/1 pins function as open-collector sources. External devices such as logic analyzers can 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 16–9 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 16–9. EMU0/1 Configuration



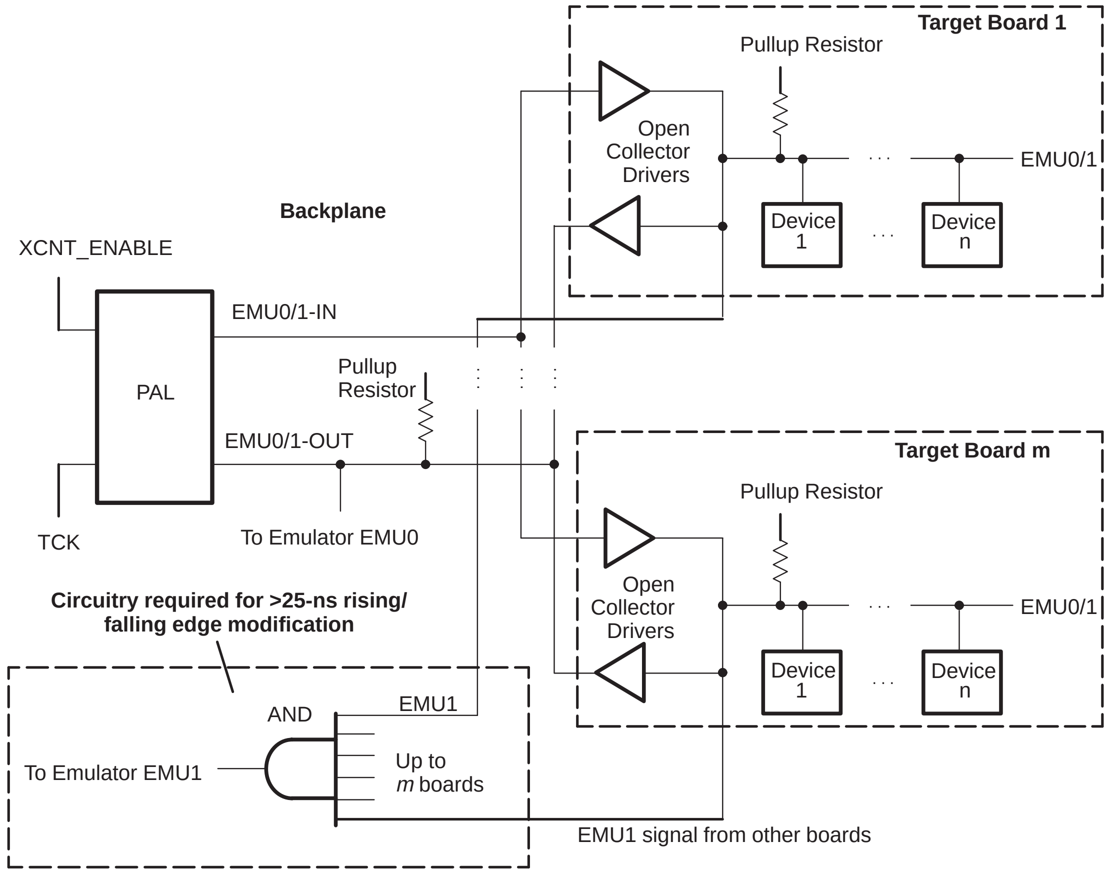
Notes:
1) The low time on EMUx-IN should be at least one TCK cycle and less than 10 μs. Software will set the EMUx-OUT pin to a high state.
2) To enable the open-collector driver and pullup resistor on EMU1 to provide rising/falling edges of less than 25 ns, the modification shown in this figure is suggested. 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 16–9 and Figure 16–10 , and the timing shown in Figure 16–11:

- ❑ 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.

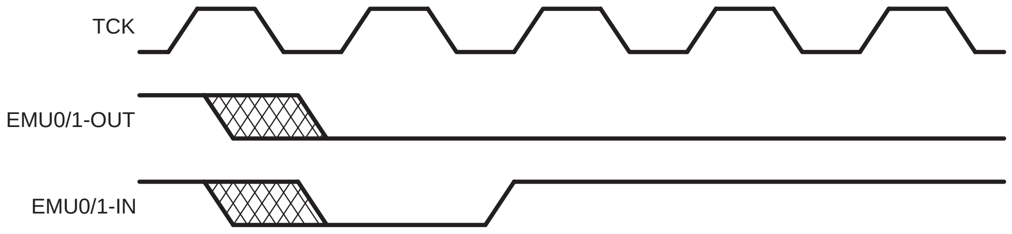
- ❑ The bused EMU0/1 signals go into a PAL® device, whose function is to generate a low pulse on the EMU0/1-IN signal when a low level is detected on the EMU0/1-OUT signal. This pulse must be longer than one TCK period to affect the devices, but less than 10 μs to avoid possible conflicts or retriggering, 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 16–10. EMU0/1 Configuration With Additional AND Gate to Meet Timing Requirements



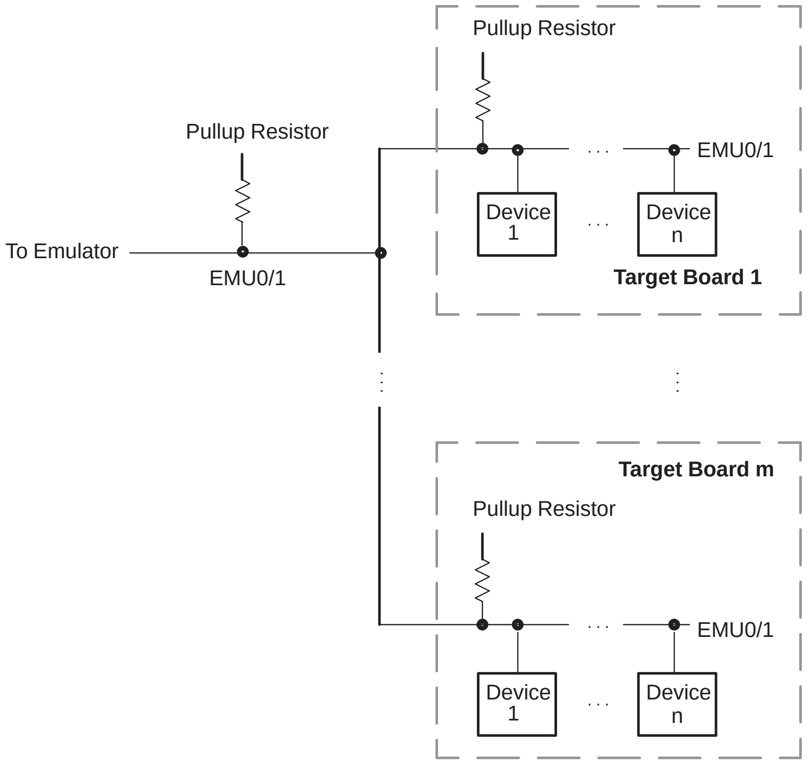
Notes:
1) The low time on EMUx–IN should be at least one TCK cycle and less than 10 μs. Software will set the EMUx–OUT pin to a high state.
2) To enable the open-collector driver and pullup resistor on EMU1 to provide rising/falling edges of less than 25 ns, the modification shown in this figure is suggested. 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 16–11. 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 16–12 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 16–12. EMU0/1 Configuration Without Global Stop

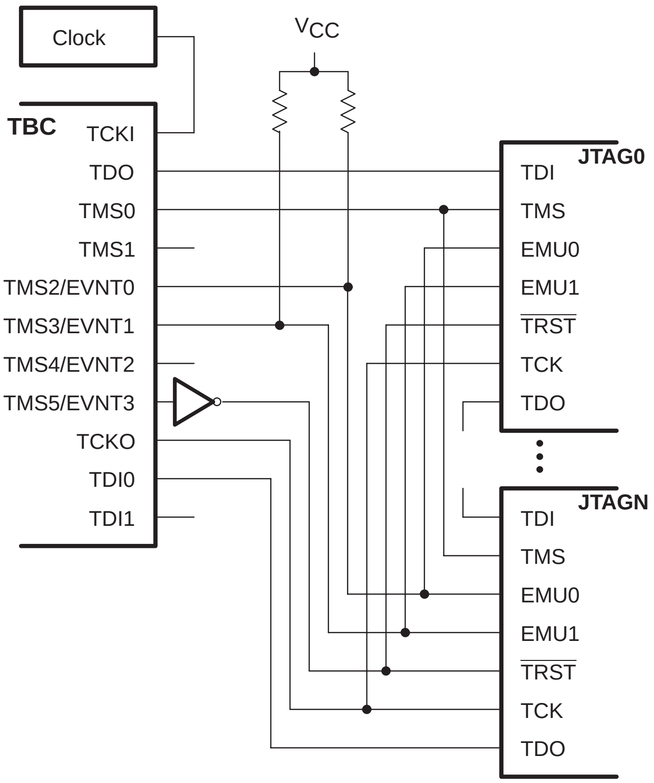


Note: The open-collector driver and pullup resistor on EMU1 must be able to provide rising/falling edges of less than 25 ns. Rising edges slower than 25 ns can cause the emulator to detect false edges during the RUNB command or when the external counter selected from the debugger analysis menu is used. If this condition cannot be met, then the EMU0/1 signals from the individual boards should be ANDed together (as shown in Figure 16–10) to produce an EMU0/1 signal for the emulator.

16.9.4 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 16–13 shows the scan path connections of *n* devices to the TBC.

Figure 16–13. TBC Emulation Connections for n JTAG Scan Paths[†]



[†] Voltage translators should be used between the TBC (5V) and the C6000 (3V).

In the system design shown in Figure 16–13, 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.

Chapter 17

General Purpose Input/Output

This chapter describes the general purpose input/output (GPIO) peripheral.

Topic **Page**

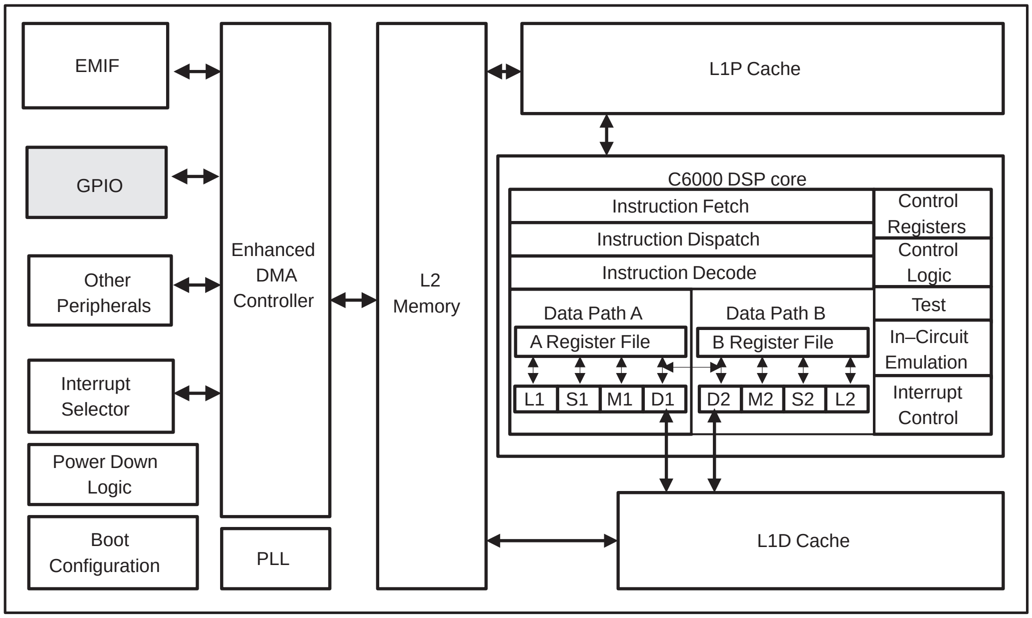
17.1 Overview

The general-purpose input/output (GPIO) peripheral provides dedicated general-purpose pins that can be configured as either inputs or outputs. When configured as an output, the user can write to an internal register to control the state driven on the output pin. When configured as an input, the user can detect the state of the input by reading the state of an internal register.

In addition, the GPIO peripheral can produce CPU interrupts and EDMA events in different interrupt/event generation modes.

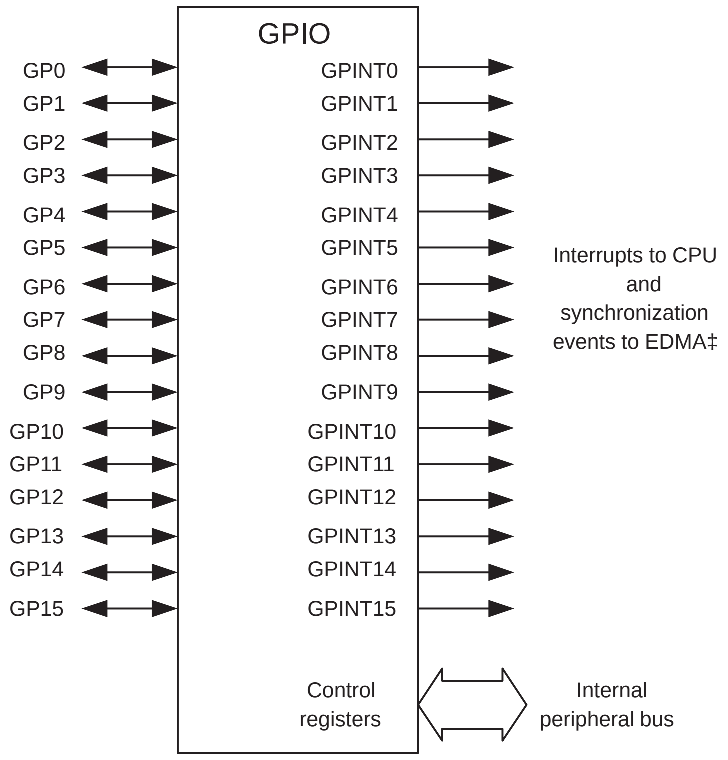
Figure 17–1 shows the GPIO peripheral in the TMS320C64x. Figure 17–2 shows the GPIO peripheral block diagram.

Figure 17–1. TMS320C64x Block Diagram



Note: Refer to the specific device datasheet for its peripheral set.

Figure 17–2. GPIO Peripheral Block Diagram[†]



† Some of the GPx pins are MUXed with other device signals. Refer to the specific device datasheet for details.

‡ All GPINTx are synchronization events to the EDMA. Only GPINT0 and GPINT[4:7] are available as interrupts to the CPU.

Some GPIO pins are MUXed with other device pins. Refer to the specific device datasheet for details on specific MUXing. GPINT[0:15] are all synchronization events to the EDMA. However, only GPINT0 and GPINT[4:7] are available as interrupt sources to the CPU.

17.2 GPIO Registers

The GPIO peripheral is configured through the registers shown in Table 17–1.

Table 17–1. GPIO Registers

Acronym	Register Name	Address	Section
GPEN	GPIO Enable Register	01B0 0000h	17.2.1
GPDIR	GPIO Direction Register	01B0 0004h	17.2.2
GPVAL	GPIO Value Register	01B0 0008h	17.2.3
—	Reserved	01B0 000Ch	—
GPDH	GPIO Delta High Register	01B0 0010h	17.2.4
GPHM	GPIO High Mask Register	01B0 0014h	17.2.5
GPDL	GPIO Delta Low Register	01B0 0018h	17.2.4
GPLM	GPIO Low Mask Register	01B0 001Ch	17.2.5
GPGC	GPIO Global Control Register	01B0 0020h	17.2.6
GPPOL	GPIO Interrupt Polarity Register	01B0 0024h	17.2.7

17.2.1 GPIO Enable Register (GPEN)

The GPIO enable register (GPEN) enables the GPIO pins for general-purpose input/output functions. To use any of the GPx pins in general-purpose input/output mode, the corresponding GPxEN bit must be set to 1. The GPEN is shown in Figure 17–3 and described in Table 17–2.

Figure 17–3. GPIO Enable Register

31–16
Reserved
R, +0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GP15 EN	GP14 EN	GP13 EN	GP12 EN	GP11 EN	GP0 EN	GP9 EN	GP8 EN	GP7 EN	GP6 EN	GP5 EN	GP4 EN	GP3 EN	GP2 EN	GP1 EN	GP0 EN
RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+0	RW,+1	RW,+1	RW,+1	RW,+1	RW,+1	RW,+0	RW,+0	RW,+1

Table 17–2. GPIO Enable Register (GPEN) Bit Field Description

No.	Field	Description
15:0	GPxEN	GPIO Mode enable GPxEN = 0; GPx pin is disabled as general-purpose input/output pin. It does not function as a GPIO pin and defaults to high impedance state. GPxEN = 1; GPx pin is enabled as general-purpose input/output pin. It defaults to high impedance state.

Some GPIO signals are MUXed with other device signals. For these MUXed signals, the signal functionality is controlled by the following:

- ❑ **Device configuration inputs**: At reset, device configuration inputs select the MUXed signal to operate as either a GPIO pin or in the other mode.
- ❑ **GPEN register bit fields**: A GPxEN = 1 indicates that the GPx pin will operate as a GPIO signal controlled by the remaining GPIO registers. A GPxEN = 0 indicates that the pin is disabled as a GPIO pin; it will operate in the other mode.

For details on signal configuration for a specific device, refer to the device datasheet and *Chapter 11, Boot Modes and Configuration*.

17.2.2 GPIO Direction Register (GPDIR)

The GPIO direction register (GPDIR) determines if a given GPIO pin is an input or an output. GPxDIR only applies if the corresponding GPIO signal is enabled via the GPxEN bit field. The GPDIR is shown in Figure 17–4 and described in Table 17–3. By default, all the GPIO pins are configured as input pins.

Figure 17–4. GPIO Direction Register (GPDIR)

31–16
Reserved
R, +0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GP15 DIR	GP14 DIR	GP13 DIR	GP12 DIR	GP11 DIR	GP10 DIR	GP9 DIR	GP8 DIR	GP7 DIR	GP6 DIR	GP5 DIR	GP4 DIR	GP3 DIR	GP2 DIR	GP1 DIR	GP0 DIR
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 17–3. GPIO Direction Register (GPDIR) Bit Field Description

No.	Field	Description
15:0	GPxDIR	GPx Direction. Controls direction (input or output) of GPIO pin. Applies when the corresponding GPxEN bit in the GPEN register is set to 1. GPxDIR = 0; GPx pin is an input GPxDIR = 1; GPx pin is an output

17.2.3 GPIO Value Register (GPVAL)

The GPIO value register (GPVAL) indicates the value to be driven on a given GPIO output pin, or the value detected on a given GPIO input pin. The GPVAL is shown in Figure 17–5. Table 17–4 shows the GPxVAL field description depending upon the direction of the GPIO pin.

Figure 17–5. GPIO Value Register (GPVAL)

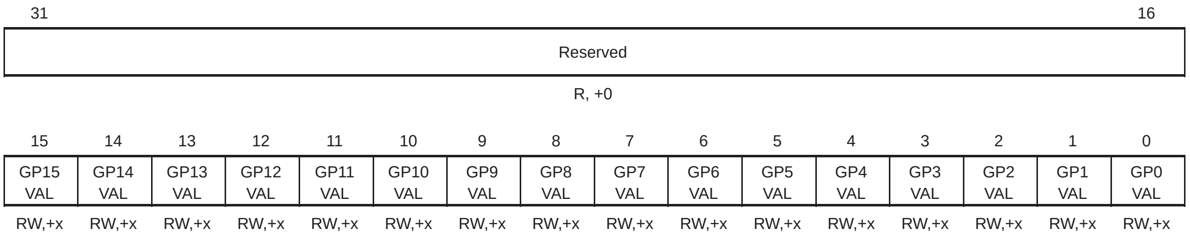


Table 17–4. GPIO Value Register (GPVAL) Bit Field Description

No.	Field	GPxDIR	Description
15:0	GPxVAL	0	Value detected at GPx input. Applies when the corresponding GPxEN bit in the GPEN register is set to 1. GPxVAL = 0; A value of 0 is latched from the GPx input pin GPxVAL = 1; A value of 1 is latched from the GPx input pin
		1	Value driven on GPx output. Applies when the corresponding GPxEN bit in the GPEN register is set to 1. GPxVAL = 0; GPx signal is driven low. GPxVAL = 1; GPx signal is driven high

17.2.4 GPIO Delta Registers (GPDH, GPDL)

The GPIO Delta High Register (GPDH) indicates whether a given GPIO input has undergone a transition from low to high. Similarly, the GPIO Delta Low Register (GPDL) indicates whether a given GPIO input has undergone a transition from high to low. If the given GPIO pin is configured as an output, the corresponding bit in the GPDH and GPDL maintains its previous value. Writing a ‘1’ to the corresponding field clears the bit, writing a ‘0’ has no effect. The GPDH is shown in Figure 17–6 and described in Table 17–5. The GPDL is shown in Figure 17–7 and described in Table 17–6.

Figure 17–6. GPIO Delta High Register (GPDH)

31–16
Reserved
R, +0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GP15 DH	GP14 DH	GP13 DH	GP12 DH	GP11 DH	GP10 DH	GP9 DH	GP8 DH	GP7 DH	GP6 DH	GP5 DH	GP4 DH	GP3 DH	GP2 DH	GP1 DH	GP0 DH
RW,+x	RW,+x	RW,+x	RW,+x	RW,+x	RW,+x	RW,+x	RW,+x	RW,+x	RW,+x	RW,+x	RW,+x	RW,+x	RW,+x	RW,+x	RW,+x

Table 17–5. GPIO Delta High Register (GPDH) Bit Field Description

No.	Field	Description
15:0	GPxDH	GPx Delta High. A low-to-high transition is detected on the GPx input. Applies when the corresponding GPx pin is enabled as an input (GPxEN = 1, GPxDIR = 0) GPxDH = 0; a low-to-high transition is not detected on GPx GPxDH = 1; a low-to-high transition is detected on GPx

Figure 17–7. GPIO Delta Low Regiser (GPDL)

31–16
Reserved
R, +0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GP15 DL	GP14 DL	GP13 DL	GP12 DL	GP11 DL	GP10 DL	GP9 DL	GP8 DL	GP7 DL	GP6 DL	GP5 DL	GP4 DL	GP3 DL	GP2 DL	GP1 DL	GP0 DL
RW,+x	RW,+x	RW,+x	RW,+x	RW,+x	RW,+x	RW,+x	RW,+x	RW,+x	RW,+x	RW,+x	RW,+x	RW,+x	RW,+x	RW,+x	RW,+x

Table 17–6. GPIO Delta Low Register (GPDL) Bit Field Description

No.	Field	Description
15:0	GPxDl	GPx Delta Low. A high-to-low transition is detected on the GPx input. Applies when the corresponding GPx pin is enabled as an input (GPxEN = 1, GPxDIR = 0). GPxDL = 0; a high-to-low transition is not detected on GPx. GPxDL = 1; a high-to-low transition is detected on GPx.

17.2.5 GPIO Mask Registers (GPHM, GPLM)

The GPIO high mask register (GPHM) and the GPIO low mask register (GPLM) are used to enable a given general-purpose input to cause a CPU interrupt, or an EDMA event generation. If a GPHM or GPLM bit is disabled, the value or transition on the corresponding GPx pin will not cause an interrupt/event generation. If the mask bit is enabled, the corresponding GPx input may cause an interrupt/event to be generated depending upon the interrupt mode selected in the GPIO Global Control Register. Refer to section 17.4 for details on the function of the GPHM and GPLM in interrupt/event generation. Figure 17–8 and Figure 17–9 show the GPHM and GPLM, respectively. These registers are described in Table 17–7 and Table 17–8.

Figure 17–8. GPIO High Mask Register (GPHM)

31–16
Reserved
R, +0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GP15 HM	GP14 HM	GP13 HM	GP12 HM	GP11 HM	GP10 HM	GP9 HM	GP8 HM	GP7 HM	GP6 HM	GP5 HM	GP4 HM	GP3 HM	GP2 HM	GP1 HM	GP0 HM
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 17–7. GPIO High Mask Register (GPHM) Bit Field Description

No.	Field	Description
15:0	GPxHM	GPx high mask. Enable interrupt/event generation based on either the corresponding GPxDH or GPxVAL bit in the GPDH and GPVAL registers, respectively. Applies when the corresponding GPxEN bit is enabled as an input (GPxEN = 1, GPxDIR = 0) GPxHM = 0; Interrupt/event generation disabled for GPx. The value or transition on GPx does not cause an interrupt/event generation. GPxHM = 1; Interrupt/event generation enabled for GPx.

Figure 17–9. GPIO Low Mask Register (GPLM)

31–16
Reserved
R, +0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
GP15 LM	GP14 LM	GP13 LM	GP12 LM	GP11 LM	GP10 LM	GP9 LM	GP8 LM	GP7 LM	GP6 LM	GP5 LM	GP4 LM	GP3 LM	GP2 LM	GP1 LM	GP0 LM
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 17–8. GPIO Low Mask Register (GPLM) Bit Field Description

No.	Field	Description
15:0	GPxLM	GPx low mask. Enable interrupt/event generation based on either the corresponding GPxDL or *inverted* GPxVAL bit in the GPDL and GPVAL registers, respectively. Applies when the corresponding GPxEN bit is enabled as an input (GPxEN = 1, GPxDIR = 0) GPxLM = 0; Interrupt/event generation disabled for GPx. The value or transition on GPx does not cause an interrupt/event generation. GPxLM = 1; Interrupt/event generation enabled for GPx.

17.2.6 GPIO Global Control Register (GPGC)

The GPIO Global Control Register (GPGC) configures the interrupt/event generation of the GPIO peripheral. The GPGC is shown in Figure 17–10 and described in Table 17–9.

Figure 17–10. GPIO Global Control Register (GPGC)

31–6	5	4	3	2	1	0
Reserved	GP0M	GPINT0M	Rsv	GPINTPOL	LOGIC	GPINTDV
R,+0	RW,+0	RW,+0	R,+0	RW,+0	RW,+0	RW,+0

Table 17–9. Global Control Register (GPGC)Bit Field Description

No.	Field	Description	Section
5	GP0M	GP0 Output Mode. Applies only if GP0 is configured as an output (GP0DIR = 1 in the GPDIR register). GP0M = 0; GPIO Mode—GP0 output is based on GP0 value (GP0VAL in GPVAL register) GP0M = 1; Logic Mode—GP0 output is based on the value of internal Logic Mode interrupt/event signal GPINT.	17.3 17.4.3
4	GPINT0M	GPINT0 interrupt/event generation mode. GPINT0M = 0; Pass Through Mode—GPINT0 interrupt/event generation is based on GP0 input value (GP0VAL in the GPVAL register). GPINT0M = 1; Logic Mode—GPINT0 interrupt/event generation is based on GPINT.	17.4 17.4.3
2	GPINTPOL	GPINT Polarity. Applies to Logic Mode (GPINT0M = 1) only. GPINTPOL = 0; GPINT is active (high) when the logic combination of the GPIO inputs is evaluated true. GPINTPOL = 1; GPINT is active (high) when the logic combination of the GPIO inputs is evaluated false.	17.4.2
1	LOGIC	GPINT Logic. Applies to Logic Mode (GPINT0M = 1) only. LOGIC = 0; OR Mode—GPINT is generated based on the logical-OR of all GPx events enabled in the GPHM or GPLM registers. LOGIC = 1; AND Mode—GPINT is generated based on the logical-AND of all GPx events enabled in the GPHM or GPLM registers.	17.4.2
0	GPINTDV	GPINT Delta/Value Mode. Applies to Logic Mode (GPINT0M = 1) only. GPINTDV = 0; Delta Mode—GPINT is generated based on a logic combination of *transitions* on the GPx pins. The corresponding bits in the GPHM and/or GPLM registers must be set. GPINTDV = 1; Value Mode—GPINT is generated based on a logic combination of *values* on the GPx pins. The corresponding bits in the GPHM and/or GPLM registers must be set.	17.4.2

17.2.7 GPIO Interrupt Polarity Register (GPPOL)

The GPIO interrupt polarity register (GPPOL) selects the polarity of the GPINTx interrupt/event signals in Pass Through Mode (section 17.4.1). Refer to specific device datasheet and the *SPRU190 TMS320C6000 Peripherals Reference Guide in the Related Documentation section* for details on interrupt/ event mapping. The GPINT0M bit in the GPIO global control register (GPGC) must be set to 0 to use GPINT0 in Pass Through Mode. The GPPOL is shown in Figure 17–11 and described in Table 17–10.

Figure 17–11. GPIO Interrupt Polarity Register (GPPOL)

31 8	7	6	5	4	3 2 1	0
Reserved	GPINT7POL	GPINT6POL	GPINT5POL	GPINT4POL	Reserved	GPINT0POL
R,+0	RW,+0	RW,+0	RW,+0	RW,+0	R+W	RW,+0

Table 17–10. GPIO Interrupt Polarity Register (GPPOL) Bit Field Description

No.	Field	Description
7:30	GPINTxPOL	GPINTx Polarity. Applies to Pass Through Mode only. GPINTxPOL = 0; GPINTx is asserted (high) based on a rising edge of GPx (effectively based on the value of the corresponding GPxVAL) GPINTxPOL = 1; GPINTx is asserted (high) based on a falling edge of GPx (effectively based on the inverted value of the corresponding GPxVAL)

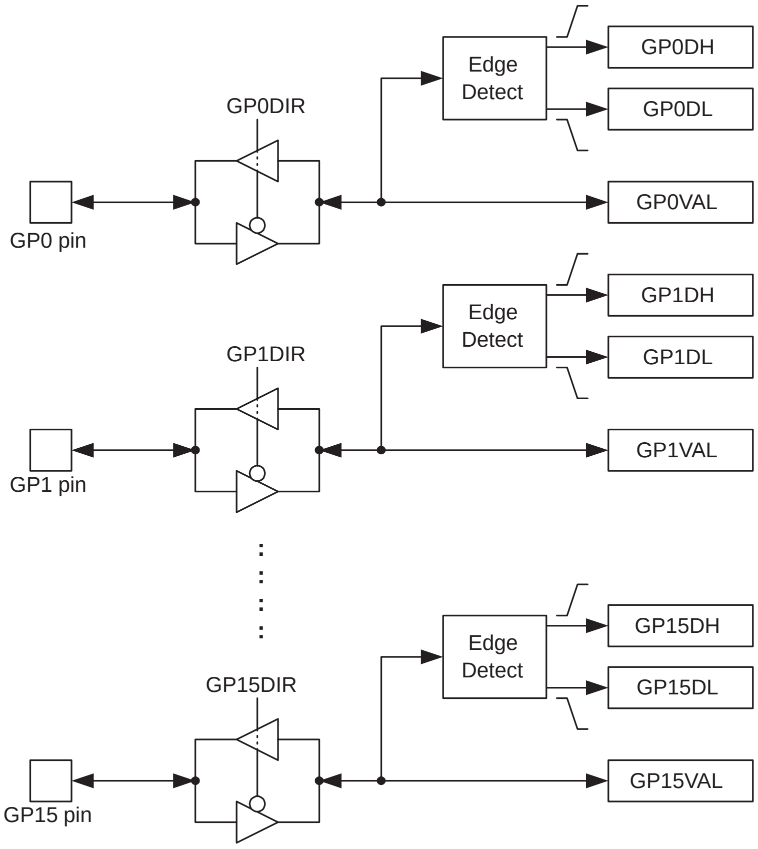
17.3 General Purpose Input/Output Function

A GPIO pin can operate as a general-purpose input/output once it is enabled in the GPIO enable register GPEN. The user can independently configure each GPIO pin as either an input or an output via the GPDIR register. When configured as an output (GPxDIR = 1), the value in the GPxVAL bit in the GPVAL register is driven on the corresponding GPx pin. When configured as an input (GPxDIR = 0), the state of the input can be read from the corresponding GPxVAL bit. Refer to section 17.2.1 , section 17.2.2 , and section 17.2.3 for details on the GPEN, GPDIR, and GPVAL registers.

In addition to the general-purpose input/output function, the edge detect logic in the GPIO peripheral reflects whether a transition has occurred on a given GPIO signal that is configured as an input (GPxDIR = 0). GPIO signal transition is reflected in the GPIO delta registers, GPDH or GPDL, respectively. The GPxDH bit in the GPDH is set to 1 when the corresponding enabled input undergoes a transition from low to high. Similarly, the GPxDL bit in the GPDL is set to 1 when the corresponding enabled input undergoes a transition from low to high.

Figure 17–12 shows the general-purpose input/output and edge detect logic of the GPIO peripheral.

Figure 17–12. General-Purpose Input/Output Functiuonal Block Diagram



To configure GP0 as a general-purpose output, in addition to setting GP0DIR = 1, the GP0M bit in the GPIO Global Control Register must also be set to 0. See section 17.4.3 for details on GP0 configurations.

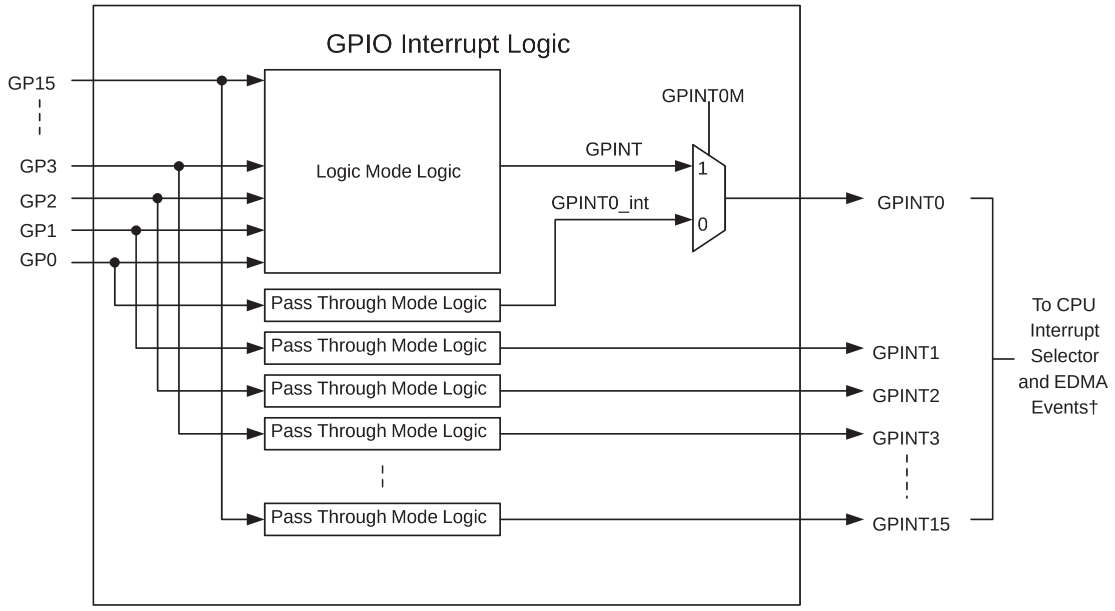
17.4 Interrupt and Event Generation

The GPIO peripheral can generate interrupts to the CPU, and synchronization events to the EDMA, in two modes:

- ❑ Pass Through Mode
- ❑ Logic Mode

The Pass Through Mode allows each GPx signal configured as an input to directly trigger a CPU interrupt and an EDMA event. The Logic Mode allows the user to determine which GPIO signals will be used as inputs to a semi-programmable logic function. The output of this logic function, GPINT, is MUXed with the Pass Through Mode internal output GPINT0_int to generate a CPU interrupt and an EDMA event, GPINT0. In addition, the Logic Mode output GPINT can be driven out of the GP0 pin for use at the board level (section 17.4.3). Figure 17–13 shows the GPIO interrupt/event generation logic.

Figure 17–13. GPIO Interrupt and Event Generation Block Diagram



† All GPINTx are synchronization events to the EDMA. Only GPINT0 and GPINT[4:7] are available as interrupts to the CPU.

17.4.1 Pass Through Mode

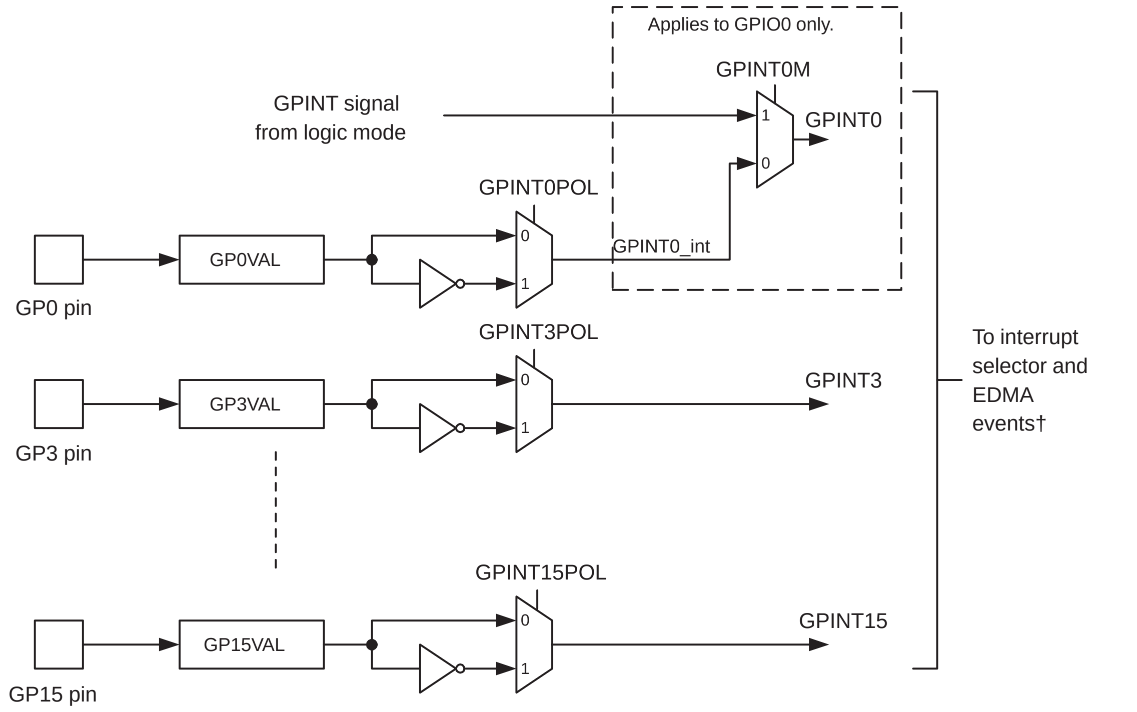
The Pass Through Mode applies to all GPIO signals. In Pass Through Mode, a transition on the GPx input pin can generate an interrupt event to the CPU and a synchronization event to the EDMA. Note that although all GPINTx are synchronization events to the EDMA, only GPINT0 and GPINT[4:7] are available as interrupts to the CPU. Figure 17–14 shows the Pass Through Mode interrupt/event generation block diagram. The user must configure the following bits correctly to use a GPx pin in the Pass Through Mode:

- ❑ GPxEN = 1: enable GPx to function as a GPIO pin.
- ❑ GPxDIR = 0: the GPx pin is an input
- ❑ Set GPINTxPOL = 0 if an interrupt/event is desired upon a rising edge transition on the corresponding GPx pin. Set GPINTxPOL = 1 if an interrupt/event is desired upon a falling edge transition on the corresponding GPx pin.

As shown in Figure 17–14, to use the GP0 in Pass Through Mode the GPINT0M bit in the GPGC register must also be set to 0. The GPINT0_int output from the Pass Through Mode logic is MUXed with the GPINT output from the Logic Mode logic to generate the GPINT0 interrupt/event. This is shown in Figure 17–13 and Figure 17–14. Refer to section 17.2.6 and section 17.4.3 for details.

If a GPx is configured as an output, the corresponding GPINTx signal is disabled.

Figure 17–14. GPINTx Generation in Pass Through Mode



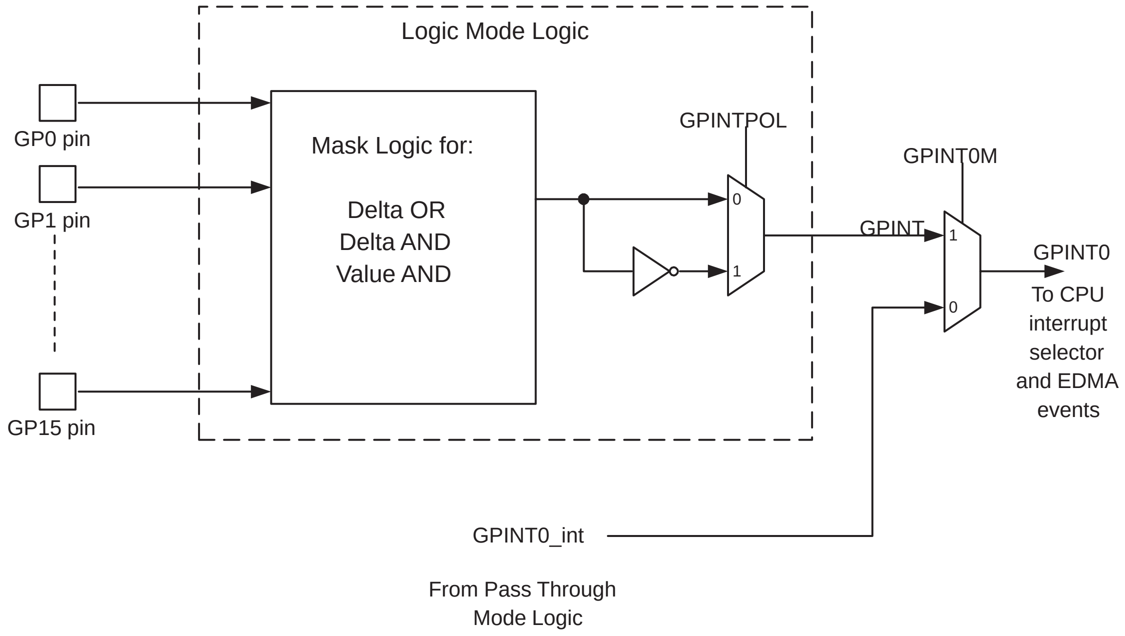
† All GPINTx are synchronization events to the EDMA. Only GPINT0 and GPINT[4:7] are available as interrupts to the CPU.

17.4.2 Logic Mode

In the Logic Mode, an interrupt/event is generated based on a logic combination of the GPIO inputs. The output of this logic function, GPINT, can be generated upon detection of a specific edge (rising, falling, or both) on any GPIO input signal(s), or upon detection of specific value(s) on any GPIO input signal(s). Disabled GPIO signals or enabled GPIO outputs cannot be used for interrupt/event generation. The Logic Mode output GPINT is MUXed with the Pass Through Mode output GPINT0_int to generate a CPU interrupt and an EDMA event. In order to use the Logic Mode to generate an interrupt/event, GPINT0M in the GPGC register must be set to 1. The GPINT signal can also be driven out of the GP0 pin for use at the board level. See section 17.4.3.

Figure 17–15 shows the block diagram of the Logic Mode logic. By default, GPINT is asserted (high) when the logic combination of the input(s) is evaluated true. By setting GPINTPOL = 1 in the GPGC register, GPINT is asserted (high) when the logic combination of the input(s) is evaluated false. This negative function of the GPINT is useful in indicating signal de-assertions at the GPIO pins.

Figure 17–15. Logic Mode Interrupt/Event Generation Block Diagram



The GPINT generation can operate in 1 of 3 modes: Delta OR, Delta AND, or Value AND mode. The GPINT generation is configured via two control bits in the GPGC register—GPINTDV and LOGIC, in addition to the mask bits in the GPHM and GPLM registers. The GPINTDV bit in the GPGC divides the Logic Mode into either Delta or Value Mode as follows:

- ❑ Delta Mode – Inputs to the interrupt/event mask logic are sourced from the GPDH and GPDL registers. GPINT is caused by the logic combination of the *transition* on the GPIO pin(s).
- ❑ Value Mode – Inputs to the interrupt/event mask logic are sourced from the GPxVAL register. GPINT is caused by the logic combination of the *value* on the GPIO pin(s).

The source to the Logic Mode mask logic is gated by the GPHM and GPLM registers. In Delta Mode, the GPxDH bit is gated with the GPxHM bit, and the GPxDL bit is gated with the GPxLM bit. In Value Mode, the value from the pin is gated with the GPxHM bit and the inverted value from the pin is gated with the GPxLM bit.

The LOGIC bit in the GPGC controls whether an interrupt/event is generated based on ALL the mask outputs being true or ANY one of the mask outputs being true:

- ❑ OR Mode – Interrupt/event generated based on ANY one of the mask outputs being true.
- ❑ AND Mode – Interrupt/event generated based on ALL of the mask outputs being true.

Table 17–11 summarizes the three modes in Logic Mode and the setup of the GPINTDV and LOGIC bits in the GPGC.

Table 17–11. Logic Mode Truth Table

GPINTDV	LOGIC	Logic Mode Description	Section
0	0	Delta OR	17.4.2.1
0	1	Delta AND	17.4.2.2
1	0	Reserved	—
1	1	Value AND	17.4.2.3

In summary, the GPIO global control register (GPGC) must be configured as follows in Logic Mode:

- ❑ GPINT0M = 1 to enable Logic Mode interrupt/event generation. The interrupt/event signal to the DSP (GPINT0) is based on the logic function output GPINT.
- ❑ GPINTPOL = 0 if the interrupt/event is based upon the logic evaluating true; or
  GPINTPOL = 1 if the interrupt/event is based upon the logic evaluating false.
- ❑ GPINTDV = 0 for Delta Mode, or GPINTDV = 1 for Value Mode.
- ❑ LOGIC = 0 for OR Mode, or LOGIC = 1 for AND Mode.

17.4.2.1 Delta OR Mode *(GPINTDV = 0; LOGIC = 0)*

Delta OR Mode allows the generation of an interrupt/event upon the first transition among a set of enabled GPIO inputs. The logic function output GPINT is driven active when ANY of the GPxDH or GPxDL bits and the corresponding GPxHM or GPxLM bits are set. Since the GPxDH and GPxDL bits operate independently from one another and have separate masks (GPxHM and GPxLM), an interrupt can be generated based on a signal transitioning to a specific state (high or low) or transitioning at all (either state).

To generate an interrupt/event in Delta OR Mode, the GPINTDV and LOGIC bits in the GPIO global control register must be configured as follows:

- GPINTDV = 0; Delta Mode
- LOGIC = 0; OR Mode

In addition, the GPHM and GPLM registers must be configured properly to enable the corresponding GPxDH and GPxDL bits to be inputs to the logic function. The following examples show the GPHM and GPLM setup. All the given GPIO pins in these examples are enabled as inputs (GPxEN = 1; GPxDIR = 0).

Example 1: GPINT based on low-to-high transition on GP1

- GPHM setup: GP1HM = 1
- GPLM setup: don’t care
- GPINT generation is caused by GP1DH = 1
  - If GP1 is high when entering this mode, a high-to-low transition (GP1DL = 1) followed by a low-to-high transition (GP1DH = 1) on GP1 will generate GPINT.
  - If GP1 is low when entering this mode, a low-to-high transition (GP1DH = 1) on GP1 will generate GPINT.

Example 2: GPINT based on any transition on GP1

- GPHM setup: GP1HM = 1
- GPLM setup: GP1LM = 1
- GPINT generation is caused by GP1DH = 1 or GP1DL = 1
  - Regardless of the initial state of GP1, the first transition on GP1 will generate GPINT. This first transition can either be a low-to-high transition (GP1DH = 1), or a high-to-low transition (GP1DL = 1).

Example 3: GPINT based on a low-to-high transition on GP1 or GP2

- GPHM setup: GP1HM = 1, GP2HM = 1
- GPLM setup: don't care
- GPINT generation is caused by GP1DH = 1 or GP2DH = 1
  - The first low-to-high transition (GPxDH = 1) on either GP1 or GP2 will generate GPINT.

Example 4: GPINT based on a low-to-high transition on GP1 or a high-to-low transition on GP2

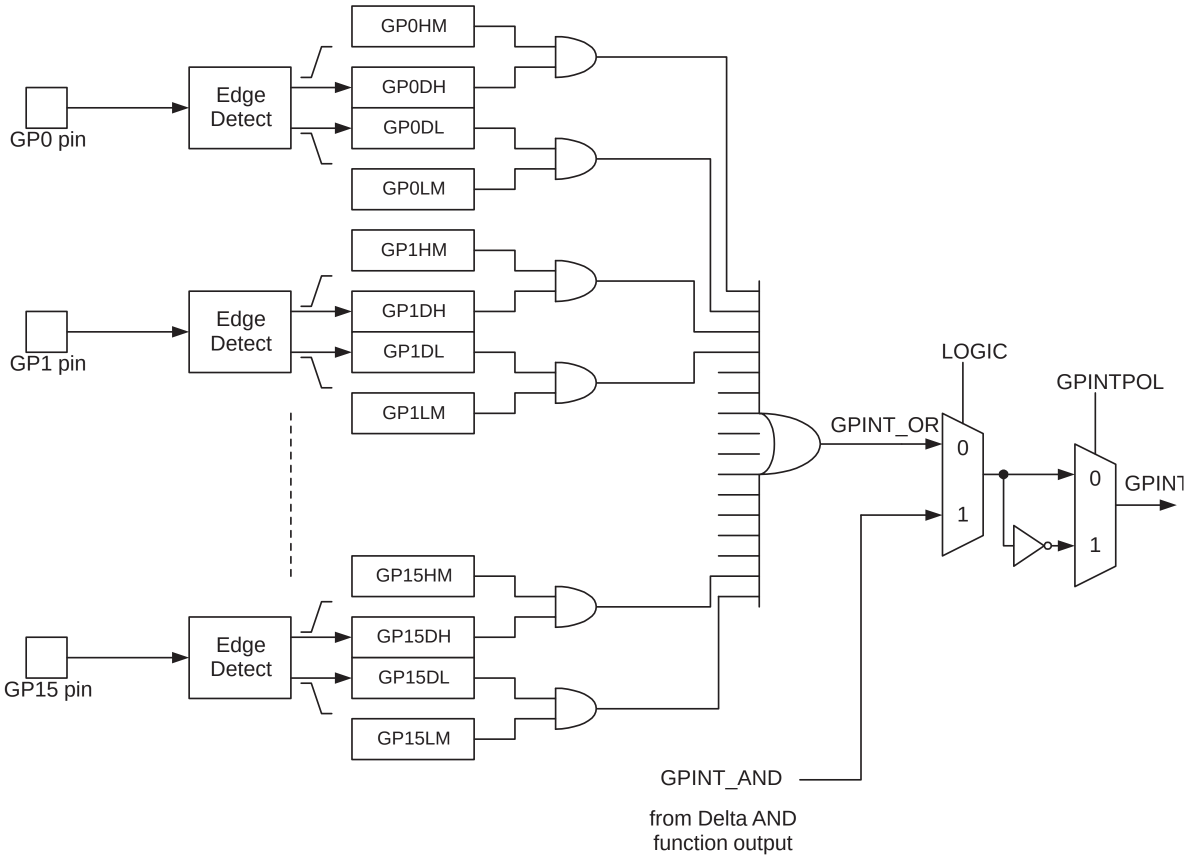
- GPHM setup: GP1HM = 1, GP2HM = don't care
- GPLM setup: GP1LM = don't care, GP2LM = 1
- GPINT generation is caused by GP1DH = 1 or GP2DL = 1
  - The first low-to-high transition on GP1 (GP1DH = 1), or a high-to-low transition on GP2 (GP2DL = 1), will generate GPINT.

Example 5: GPINT based on any transition on GP1 or any transition on GP2

- GPHM setup: GP1HM = 1, GP2HM = 1
- GPLM setup: GP1LM = 1, GP2LM = 1
- GPINT generation is caused by GP1DH, GP1DL, GP2DH, or GP2DL = 1.
  - The first transition on GP1 (GP1DH or GP1DL = 1) or the first transition on GP2 (GP2DH or GP2DL = 1) will generate GPINT.

Figure 17–16 shows the block diagram of the Delta OR mode.

Figure 17–16. GPINT Generation, Delta OR Mode



17.4.2.2 Delta AND Mode *(GPINTDV = 0; LOGIC = 1)*

Delta AND Mode allows the generation of an interrupt/event after all of a set of specified signals have undergone some specified transitions. GPINT is driven active when BOTH of the following conditions are true:

❑ ALL of the GPxDH bits are asserted for the group of GPIO signals with the GPxHM bits set.

❑ ALL of the GPxDL bits are asserted for the group of GPIO signals with the GPxLM bits set.

Since the GPxDH and GPxDL bits operate independently from one another and have separate masks (GPxHM and GPxLM), GPINT can be generated based on a signal transitioning from one state to another and back to the original state.

To generate an interrupt/event in Delta AND Mode, the GPINTDV and LOGIC bits in the GPIO Global Control Register must be configured as follows:

- GPINTDV = 0; Delta Mode
- LOGIC = 1; AND Mode

In addition, the GPHM and GPLM registers must be configured properly to enable the corresponding GPxDH and GPxDL bits to be inputs to the logic function. The following examples show the GPHM and GPLM setup. All the given GPIO pins in these examples are enabled as inputs (GPxEN = 1; GPxDIR = 0).

Example 1: GPINT based on low-to-high transition on GP1

- GPHM setup: GP1HM = 1
- GPLM setup: don’t care
- GPINT generation is caused by GP1DH = 1
  - If GP1 is high when entering this mode, a high-to-low transition (GP1DL = 1) followed by a low-to-high transition (GP1DH = 1) on GP1 will generate GPINT.
  - If GP1 is low when entering this mode, a low-to-high transition (GP1DH = 1) on GP1 will generate GPINT.

Example 2: GPINT based on a low-to-high and a high-to-low transition on GP1

- GPHM setup: GP1HM = 1
- GPLM setup: GP1LM = 1
- GPINT generation is caused by GP1DH = 1 and GP1DL = 1
  - If GP1 is high when entering this mode, a high-to-low transition (GP1DL = 1) followed by a low-to-high transition (GP1DH = 1) on GP1 will generate GPINT.
  - If GP1 is low when entering this mode, a low-to-high transition (GP1DH = 1) followed by a high-to-low transition (GP1DL = 1) on GP1 will generate GPINT.

Example 3: GPINT based on a low-to-high transition on both GP1 and GP2

- GPHM setup: GP1HM = 1, GP2HM = 1
- GPLM setup: don’t care
- GPINT generation is caused by GP1DH = 1 and GP2DH = 1
  - Both GP1 and GP2 must undergo a low-to-high transition (GP1DH = 1 and GP2DH = 1) to generate GPINT. If either (or both) signal starts out

high, GPINT is not generated until this signal undergoes a high-to-low followed by a low-to-high transition.

Example 4: GPINT based on a low-to-high transition on GP1 and a high-to-low transition on GP2

❑ GPHM setup: GP1HM = 1, GP2HM = don’t care
❑ GPLM setup: GP1LM = don’t care, GP2LM = 1
❑ GPINT generation is caused by GP1DH = 1 and GP2DL = 1

- Regardless of the initial state, GP1 must undergo a low-to-high transition (GP1DH = 1), AND GP2 must undergo a high-to-low transition (GP2DL = 1).

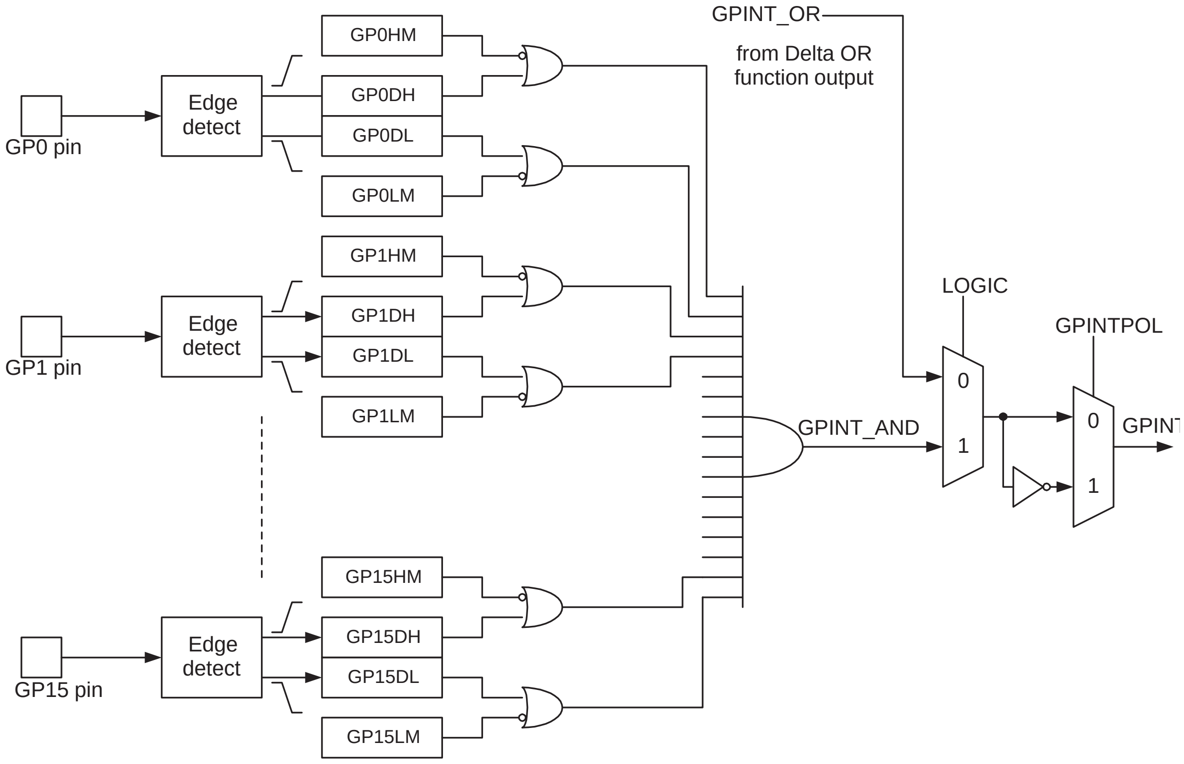
Example 5: GPINT based on low-to-high and high-to-low transitions on both GP1 and GP2

❑ GPHM setup: GP1HM = 1, GP2HM = 1
❑ GPLM setup: GP1LM = 1, GP2LM = 1
❑ GPINT generation is caused by GP1DH, GP1DL, GP2DH, and GP2DL = 1.

- Regardless of initial state, both GP1 and GP2 must undergo transition from original state back to original state.

Figure 17–17 shows the functional block diagram of the Delta AND mode.

Figure 17–17. GPINT Generation, Delta AND Mode



Note: The functional block diagram shows the mask logic as an OR with an inverter on the mask bit. This forces the OR to evaluate true when the mask bit is disabled. This is strictly a functional block diagram. The actual implementation prevents the GPINT from being asserted in the case that all of the mask bits are disabled.

17.4.2.3 *Value* AND *Mode (GPINTDV = 1; LOGIC = 1)*

Value AND Mode allows the generation of an interrupt/event based on a set of signals matching some given values. GPINT is driven active when BOTH of the following conditions are true:

- All of the GPxVAL bits are high for the group of GPIO signals with the GPxHM bits set.
- All of the GPxVAL bits are low for the group of GPIO signals with the GPxLM bits set.

To generate an interrupt/event in Value AND Mode, the GPINTDV and LOGIC bits in the GPIO Global Control Register must be configured as follows:

- GPINTDV = 1; Value Mode
- LOGIC = 1; AND Mode

In addition, the GPHM and GPLM registers must be configured properly to enable the corresponding GPxVAL bits to be inputs to the logic function. No GPINT is generated if any given GPIO signal has both GPxHM and GPxLM asserted. This is because no GPx (and the corresponding GPxVAL) can be both high and low simultaneously. The following examples show the GPHM and GPLM setup. All the given GPIO pins in these examples are enabled as inputs (GPxEN = 1; GPxDIR = 0).

Example 1: GPINT based on GP1 = 1

- GPHM setup: GP1HM = 1
- GPLM setup: GP1LM = 0
- GPINT generation is caused by GP1 = 1
  - If GP1 is high (GPxVAL = 1) when entering this mode, GPINT is immediately asserted.
  - If GP1 is low (GPxVAL = 0) when entering this mode, a low-to-high transition (GPxVAL = 1) on GP1 will generate GPINT.

Example 2: No GPINT generated when GPxHM = GPxLM = 1

- GPHM setup: GP1HM = 1
- GPLM setup: GP1LM = 1
- No GPINT is generated because GP1 can never simultaneously be both low (GPxVAL = 0) and high (GPxVAL = 1).

Example 3: GPINT based on GP1 = GP2 = 1

- GPHM setup: GP1HM = 1, GP2HM = 1
- GPLM setup: GP1LM = 0, GP2LM = 0
- GPINT generation is caused by GP1VAL = 1 and GP2VAL = 1
  - If GP1 = GP2 = 1 when entering this mode, GPINT is immediately asserted.
  - If GP1 = 1 and GP2 = 0 when entering this mode, a low-to-high transition on GP2 (GP2VAL = 1), as GP1 stays high, will generate GPINT.
  - If GP1 = GP2 = 0 when entering this mode, GPINT is generated when both GP1 and GP2 become high. If GP1 transitions high (GP1VAL = 1) then low (GP1VAL = 0) before GP2 transitions high, no GPINT is generated.

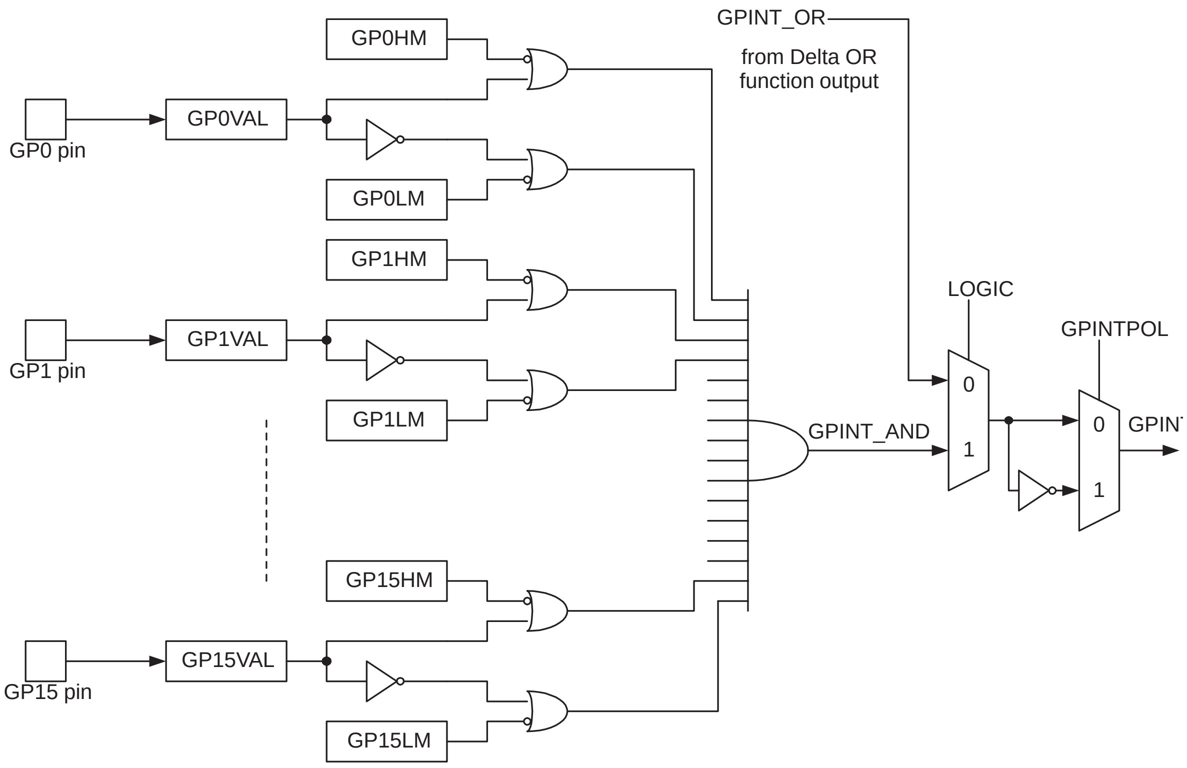
Example 4: GPINT based on GP1 = 1 and GP2 = 0

❑ GPHM setup: GP1HM = 1, GP2HM = 0
❑ GPLM setup: GP1LM = 0, GP2LM = 1
❑ GPINT generation is caused by GP1VAL = 1 and GP2VAL = 0

■ As in previous examples, both GP1 and GP2 must simultaneously be at the defined state: GP1VAL = 1 and GP2VAL = 0.

Figure 17–18 shows the functional block diagram of the Value AND Mode.

Figure 17–18. GPINT Generation, Value AND Mode



Note: The functional block diagram shows the mask logic as an OR with an inverter on the mask bit. This forces the OR to evaluate true when the mask bit is disabled. This is strictly a functional block diagram. The actual implementation prevents the GPINT from being asserted in the case that all the mask bits are disabled.

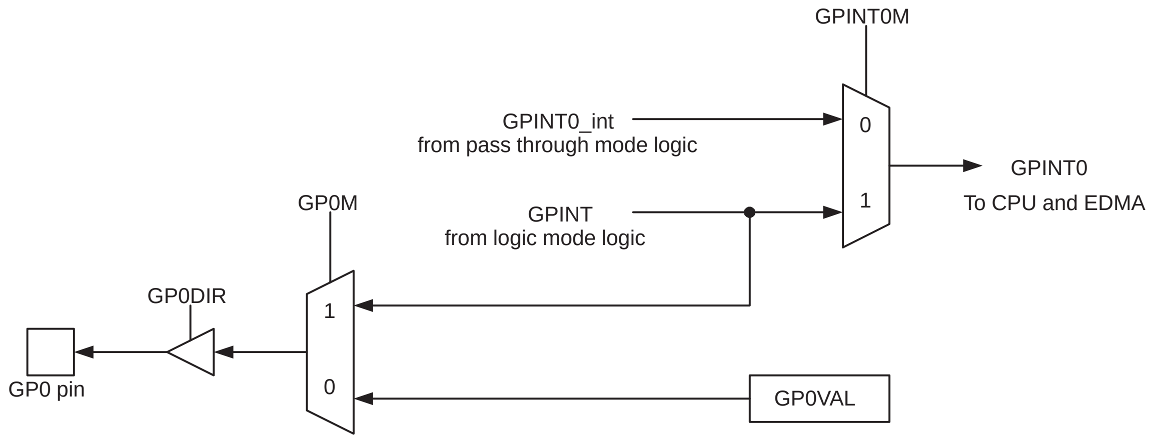
17.4.3 GPINT Muxing With GP0 and/or GPINT0

The logic function output signal GPINT can be used by both the DSP and an external device as follows:

- GPINT can generate a CPU interrupt and an EDMA event via GPINT0.
- In addition, if GP0 is configured as an output, GPINT can be driven out on GP0 to be used by external devices.

Figure 17–19 shows the connection of the GPINT signal.

Figure 17–19. GPINT Connection to GP0 and GPINT0



When GP0 is configured as an output (GP0DIR = 1), the GP0M bit controls whether the GP0 signal operates in GPIO Mode or in Logic Mode. In GPIO Mode (GP0M = 0), the value of the GP0VAL bit is driven out on GP0. In Logic Mode (GP0M = 1), GPINT is driven out on GP0. When GP0 is configured as an input, GP0M has no effect.

The GPINT0M bit controls whether the GPINT0 signal operates in Pass Through Mode or in Logic Mode. In Pass Through Mode, the GPINT0_int value from the Pass Through Mode logic is used to generate an interrupt/event to the CPU and EDMA. See section 17.4.1 and Figure 17–14 for details on the GPINT0_int signal. In Logic Mode, the Logic Mode output GPINT is used instead to generate an interrupt/event to the CPU and EDMA.

If GP0 is configured as an output, Logic Mode is still supported and GPINT can be generated. However, Pass Through Mode is disabled if GP0 is configured as an output. No GPINT0_int is generated.

17.5 GPIO Interrupts/Events

The GPIO peripheral generates interrupts and events to the CPU and EDMA, respectively, via the internal GPINTx signals. The GPIO interrupts/events are summarized in Table 17–12. GPINT1 – GPINT15 can only be used in Pass Through Mode, while GPINT0 can be used in either Pass Through or Logic Mode. All GPINTx are available as synchronization events to the EDMA. Only GPINT0 and GPINT[4:7] are available as interrupt sources to the CPU.

Table 17–12. GPIO Interrupts to CPU and Events to EDMA

Interrupt/Event Name	Description
GPINT0	GPINT0 is the interrupt/event output from Pass Through Mode or Logic Mode. In Pass Through Mode, GPINT0 reflects the value of GP0 or $\overline{\text{GP0}}$ (GPINT0_int). In Logic Mode, GPINT0 reflects the logic function output GPINT.
GPINT[1:15]	GPINT[1:15] are the interrupt outputs from Pass Through Mode. They reflect the value of GP[1:15] or $\overline{\text{GP[1:15]}}$ in Pass Through Mode.

Chapter 18

UTOPIA Level 2 Interface

This chapter describes the UTOPIA (Universal Test and Operations Interface for Asynchronous Transfer Mode [ATM]) interface on the C64x devices.

Topic **Page**

18.1 Overview

The C6000 UTOPIA peripheral is an ATM controller (ATMC) slave device that interfaces to a master ATM controller. The UTOPIA port conforms to the ATM Forum standard specification af-phy-0039.000. Specifically, this interface supports the UTOPIA Level 2 interface that allows 8-bit slave operation up to 50MHz for both transmit and receive operations.

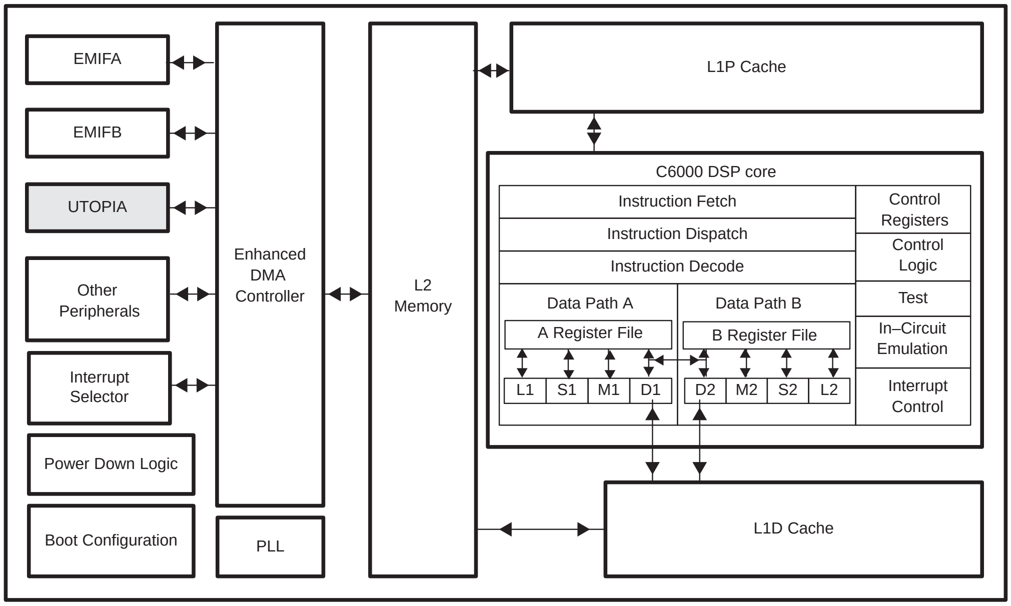
The UTOPIA slave interface relies on the master ATM controller to provide the necessary control signals such as the Clock, Enable and Address values. Only cell-level handshaking is supported.

Both the CPU and the EDMA can service the UTOPIA peripheral. The ATM Adaptation Layer (AAL) commonly called as Segmentation and Re-assembly (SAR) functions should be performed in software.

All references to the term ”slave devices” are analogous to multi-PHYs (MPHYs) as referenced in the ATM Forum specification.

Figure 18–1 shows the UTOPIA interface on some of the C64x.

Figure 18–1. TMS320C64x Block Diagram

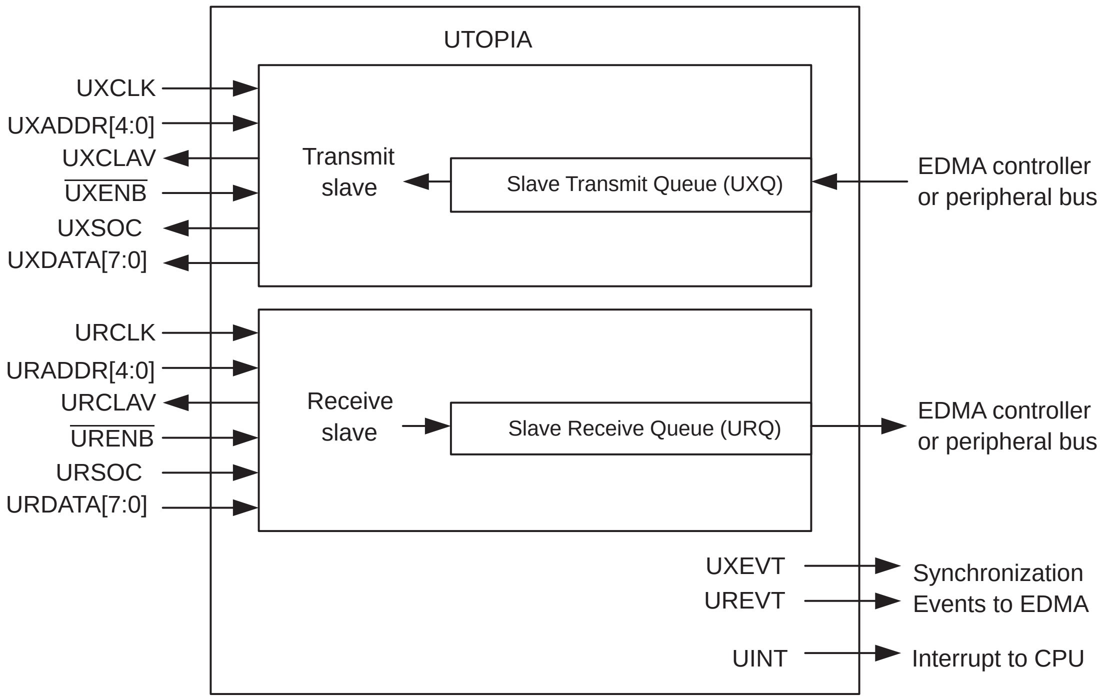


Note: Refer to the specific device datasheet for its peripheral set.

18.2 UTOPIA Interface Signals and Registers

The UTOPIA slave consists of the transmit interface and the receive interface. Figure 18–2 shows the UTOPIA block diagram. The interface signals are described in section 18.4.1.

Figure 18–2. UTOPIA Block Diagram


UTOPIA
UXCLK
UXADDR[4:0]
UXCLAV
$\overline{\text{UXENB}}$
UXSOC
UXDATA[7:0]
Transmit slave
Slave Transmit Queue (UXQ)
EDMA controller or peripheral bus
URCLK
URADDR[4:0]
URCLAV
$\overline{\text{URENB}}$
URSOC
URDATA[7:0]
Receive slave
Slave Receive Queue (URQ)
EDMA controller or peripheral bus
UXEVT
UREVT
Synchronization Events to EDMA
UINT
Interrupt to CPU

The UTOPIA port is configured via the configuration registers listed in Table 18–1. The data for transmit and receive queues are accessible via the EDMA controller or CPU at the data port addresses shown in Table 18–2. The UTOPIA sends notification of important interrupts to the CPU via UINT and synchronization events to the EDMA controller via the UXEVT and UREVT signals.

Table 18–1. UTOPIA Configuration Register

Acronym	Register Name	Address	Section
UCR	UTOPIA Control Register	01B4 0000h	18.2.1
–	Reserved	01B4 0004h	–
–	Reserved	01B4 0008h	–
UIER	UTOPIA Interrupt Enable Register	01B4 000Ch	18.6
UIPR	UTOPIA Interrupt Pending Register	01B4 0010h	18.6
CDR	Clock Detect Register	01B4 0014h	18.7
EIER	Error Interrupt Enable Register	01B4 0018h	18.8.1
EIPR	Error Interrupt Pending Register	01B4 001Ch	18.8.1

Table 18–2. Utopia Data Port Address

Acronym	Queue Name	Address	Section
URQ	UTOPIA Receive Queue	3C00 0000h	18.4.5 and 18.5.2
UXQ	UTOPIA Transmit Queue	3D00 0000h	18.4.3 and 18.5.1

18.2.1 UTOPIA Control Register (UCR)

The UTOPIA interface is configured via the UTOPIA Control Register (UCR). The UCR contains UTOPIA status control bits. The UCR is shown in Figure 18–3 and summarized in Table 18–3.

Figure 18–3. UTOPIA Control Register (UCR)

31	30 29	28 22	21 18	17	16
BEND	Reserved	SLID	XUDC	Rsvd	UXEN
RW, +0	R, +0	RW,+0	RW, +0	R,+0	RW, +0

15	14	13 6	5 2	1	0
Rsvd	MPHY	Reserved	RUDC	Rsvd	UREN
R, +0	RW, +1	R, +0	RW,+0	R, +0	RW, +0

Table 18–3. UTOPIA Control Register (UCR) Bit Field Description

Bit	Field	Description	Section
31	BEND	Endian Mode for data transferred via UTOPIA interface. BEND=0: Data is assembled to conform to Little-endian format BEND=1: Data is assembled to conform to Big-endian format	18.9
28:22	SLID	Slave ID: Applicable in MPHY mode. SLID is a programmable 5-bit PHY address used to identify the UTOPIA in a MPHY set up. Does not apply to single-PHY slave operation.	18.4.7
21:18	XUDC	Transmit User-Defined Cell Valid values: 0 to 11. The remaining values are reserved. XUDC = 0: The XUDC feature is disabled. The UTOPIA interface transmits a normal ATM cell of 53 bytes. XUDC = 1 to 11: The Utopia interface transmits the programmed number (1 to 11) of bytes as extra header. A UDC may have a minimum of 54 bytes (XUDC=1) up to a maximum of 64 bytes (XUDC=11).	18.3
16	UXEN	UTOPIA Transmitter Enable. UXEN = 0: The UTOPIA port transmitter is disabled and in reset state. UXEN = 1: The UTOPIA port transmitter is enabled.	18.11.1

14	MPHY	UTOPIA Receive/Transmit Multi-PHY mode
		MPHY = 0: Single PHY mode selected for Receive and Transmit UTOPIA
		MPHY = 1: Multi-PHY mode selected for Receive and Transmit UTOPIA. Default state.
			18.4.7

5:2	RUDC	Receive User-Defined Cell Valid values: 0 to 11. The remaining values are reserved. RUDC = 0: The RUDC feature is disabled. The Utopia interface expects a normal ARM cell of 53 bytes. RUDC = 1 to 11: The Utopia interface expects to receive the programmed number (1 to 11) of bytes as extra header. A UDC may have a minimum of 54 bytes (RUDC=1) up to a maximum of 64 bytes (RUDC=11).	18.3
0	UREN	UTOPIA Receiver Enable. UREN = 0: The UTOPIA port receiver is disabled and in reset state. UREN = 1: The UTOPIA port receiver is enabled.	18.11.1

18.3 UTOPIA Cell Transfer Format

The ATM Forum specification for UTOPIA Level 2 specifies the order in which header and payload information is sent across the ATM-PHY interface. The header information is sent first followed by the 48-byte payload. A standard ATM cell is 53 bytes (5-byte header + 48-byte payload). The UTOPIA peripheral also supports a non-standard ATM cell (R/XUDC = 1 to 11) of size 54 to 64 bytes. Figure 18–4 and Figure 18–5 show the standard and non-standard cell transfer format, respectively, with reference to time.

Figure 18–4. Standard UTOPIA Cell Transfer Format for 8-Bit Mode

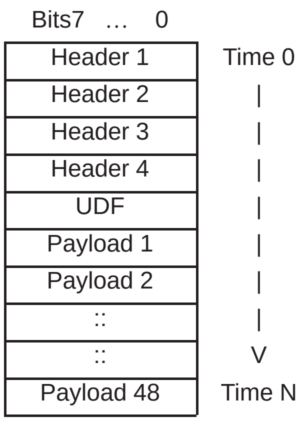
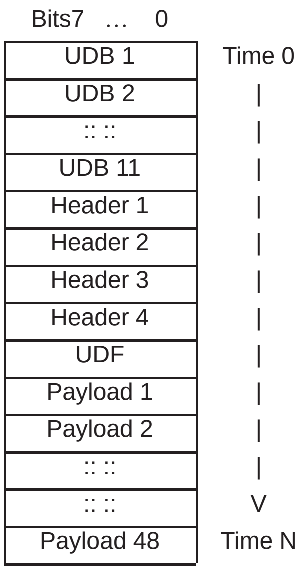


Figure 18–5. Non-Standard UTOPIA Cell Transfer Format for 8-Bit Mode

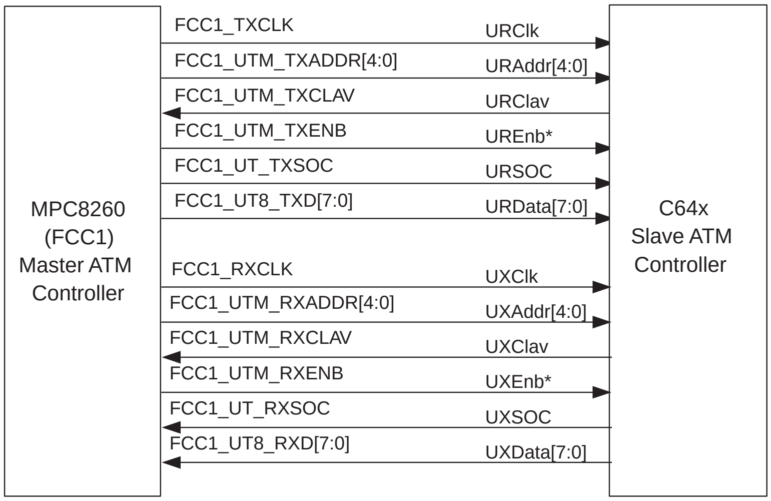


For the C6000, each ATM cell must be aligned on a word-boundary. Therefore, each ATM cell (53-Byte) in the DSP memory (internal or external) and in the UTOPIA transmit/receive queues is padded with dummy bytes as necessary before the ATM header. The standard 56-byte cell-packet consists of the 53-byte ATM cell, plus three bytes of dummy data before the ATM header. This 56-Byte packet is referred to as a cell-packet. See also section 18.10.

18.4 UTOPIA Slave ATM Controller

The UTOPIA interface can be used as an ATM controller slave in either a single- or multi-PHY (MPHY) configuration. As a slave, the clock, address, and enable signals of the transmit and receive interfaces are driven by the master. An example configuration is shown in Figure 18–6.

Figure 18–6. TMS320C64x UTOPIA Slave Interface to Motorola PowerQUICC™ Master in 8-Bit Mode



18.4.1 UTOPIA Slave Pins

As a slave device in an ATM system, the UTOPIA will perform all ATM cell transfers as and when directed by the master. The clock, address, and enable signals are inputs. The master can configure the slave’s address in the UTOPIA Control Register (UCR) through the HPI/PCI interface. The pins and their direction relevant to the UTOPIA slave interface are shown in Table 18–4.

The slave responds when it detects its assigned address on the address bus by asserting its UXCLAV or URCLAV signal, if indeed a cell is available for transmit or receive, respectively. If the slave does not have a cell to transmit or cell space to receive, it does not assert the relevant CLAV signal. But the master continues to poll the remaining slaves/PHYs in the system on the address bus.

Table 18–4. Slave UTOPIA Pin Descrition

Pin	Direction	Description
		UTOPIA TRANSMIT INTERFACE (Slave mode)
UXCLK	In	UTOPIA Transmit Clock. An input driven by the master in the system. Transmit data and transmit control signals are synchronous to this clock.
UXADDR[4:0]	In	5-bit address input driven by the master ATM Controller to identify each of the slave devices (up to 31) in the ATM system.
UXCLAV	Out	Transmit Cell Available status output signal of the slave. For cell-level handshake, the following is true: 0: Indicates that the slave does not have a complete cell available for transmit 1: Indicates that the slave has a complete cell available to transmit.
$\overline{\text{UXENB}}$	In	UTOPIA Transmit Interface Enable input signal. Asserted active low by the master to indicate that the slave should put first byte of valid data and assert SOC signal in the next clock cycle.
UXSOC	Out	Transmit Start-Of-Cell signal (active high) output by the slave on rising edge of UXCLK to indicate that the first valid byte of the cell is available on the Transmit Data Bus UXDATA[7:0].
UXDATA[7:0]	Out	8-bit Transmit Data Bus. Slave transmits ATM cells to the master using this bus on rising edge of UXCLK.
		UTOPIA RECEIVE INTERFACE (Slave mode)
URCLK	In	UTOPIA Receive Clock is an input signal driven by the ATM master. Receive data and control signals are sampled and synchronous to this clock.
URADDR[4:0]	In	5-bit address bus input driven by the master to select a slave.
URCLAV	Out	Receive Cell Available status signal is an output from the slave to indicate that it has space available to receive a cell from the master. For cell-level handshake, the following is true: 0: No space available to receive a cell from the master 1: Space available to receive a cell from the master
$\overline{\text{URENB}}$	In	UTOPIA Receive Interface Enable. An active low signal driven by the master to enable the receive interface of the slave. It indicates to the slave to sample Receive Data and SOC signal in the next clock cycle or thereafter.
URSOC	In	Receive Start-Of-Cell signal driven by the master to indicate that the first valid byte of the cell is available on the Receive Data Bus for the slave to sample.
URDATA[7:0]	In	8-bit UTOPIA Receive Data Bus. Data from the master is received on this bus. Data is sampled on the rising edge of URCLK.

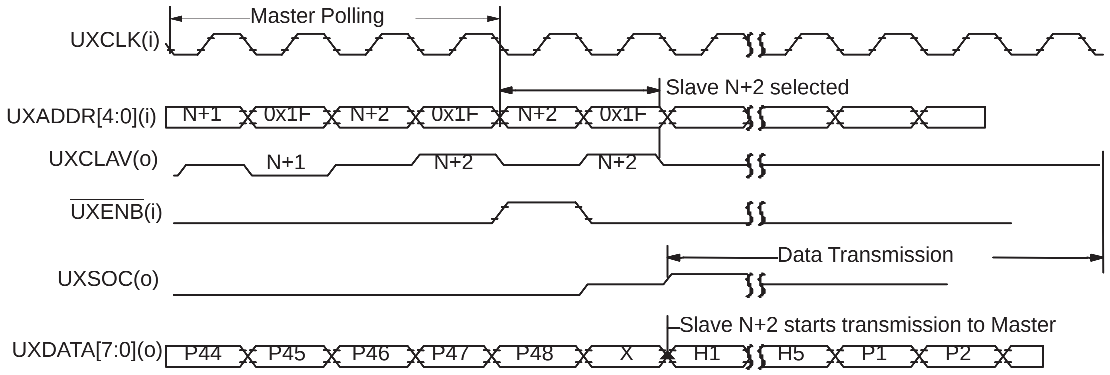
18.4.2 Slave-Transmit Operation

The UTOPIA slave-transmit block consists of a UTOPIA Level 2 pin interface that interfaces internally to the slave-transmit queue. The UTOPIA slave–transmit block diagram is shown in Figure 18–2.

The slave–transmit queue can be accessed via the UXQ data part, as shown in Table 18–2. The CPU/EDMA services the slave–transmit queue with 32–bit writes when a transmit interrupt/event is generated by the UTOPIA transmit section.

When the UTOPIA slave interface detects its address on the transmit address bus UXADDR[4:0], it drives the single UXCLAV signal to indicate to the master whether or not a cell is available for transmit. In the following cycles when the master chooses (after completion of any on-going data transfers) to get the data from this UTOPIA slave, the master asserts the slave address along with the Enable signal, $\overline{\text{UXENB}}$. Next, the slave starts transmitting the data on its UXDATA[7:0] pins. It does so by asserting the start-of-cell signal, UXSOC. Figure 18–7 shows the UTOPIA slave transmit interface timing. The clock for the UTOPIA slave transmit interface, UXCLK, is an input driven by the external master.

Figure 18–7. ATM Controller Slave Transmit Timing



18.4.3 Slave-Transmit Queue

The Slave-Transmit Queue facilitates the UTOPIA interface to be ready to transmit data whenever the master requests one. The slave-transmit queue generates a UXEVT when it is not full. This transmit event triggers the EDMA controller to perform 32-bit writes to the slave-transmit queue. A total of 14 word writes are required to fill one standard ATM cell-packet in the queue.

As soon as the first write to the queue occurs, the transmit event UXEVT is cleared. The next UXEVT event is generated if the queue is not full. This allows for the EDMA to begin the next cell-packet write without having to wait for the current cell to be fully written to the queue. This process repeats as described below.

The transmit event (UXEVT) is generated and cleared as follows:

1) UXEVT is generated when the queue is not full. The queue is not full when there is space for at least one cell-packet (56B).

2) UXEVT is cleared when the first write (by the EDMA) of that cell occurs

3) UXEVT is regenerated immediately (without waiting for the previous cell to be fully written) if the queue is not full.

4) Go to step 2.

The CPU can also be used to service the UTOPIA via the UINT signal. See section 18.5 for details.

The UTOPIA slave will agree for transmission to the master by asserting its UXCLAV signal when there is at least one cell available in the slave-transmit queue. If the slave cannot provide the next cell in a contiguous fashion, it de-asserts its UXCLAV in the cycle following the completion of the current cell transmission. The UXCLAV remains asserted if the slave has another cell available to transfer to the master. The master may disable RxEnb* on its side (connected to the $\overline{\text{UXENB}}$ pin for this ATMC slave), which causes the UTOPIA slave to hold off the next cell transfer until the master indicates as such.

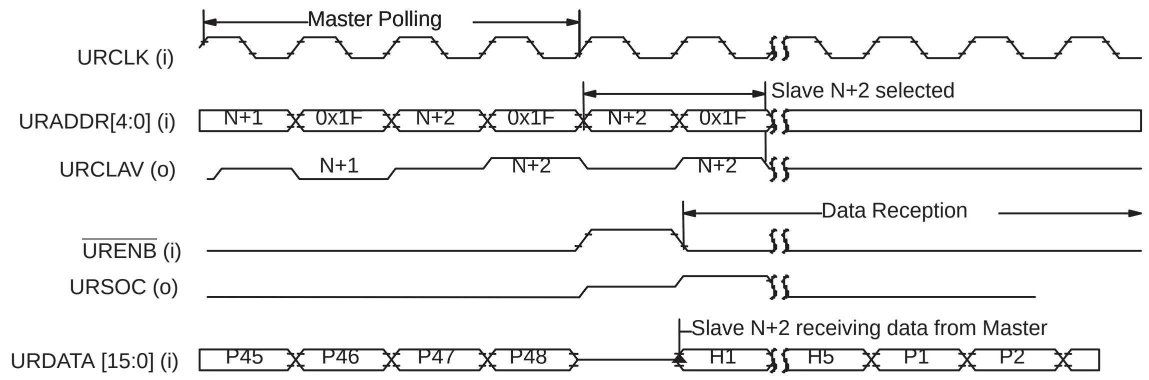
18.4.4 Slave-Receive Operation

The UTOPIA slave-receive block consists of a UTOPIA II pin interface that interfaces internally to a Slave-Receive Queue. The UTOPIA slave-receive block diagram is shown in Figure 18–2.

The Slave-Receive Queue can be accessed via the URQ data port, as shown in Table 18–2. The CPU/EDMA controller services the slave-receive queue with 32-bit reads when a receive interrupt/event is generated by the UTOPIA receive section.

When the master polls for slaves in the system that can receive its cells, the UTOPIA slave responds with an active cell-available signal on its URCLAV pin if it has space in the slave-receive queue to receive a complete cell. The master can choose to transmit to this slave or continue to poll to find a suitable slave for its data. In any case, the UTOPIA slave responds to its assigned address by asserting its appropriate URCLAV state a cycle after its address is detected on URADDR[4:0] bus. The master will then output the slave address that has an active RCLAV and also provides the enable signal ($\overline{\text{URENB}}$ on slave) to enable slave-receive operation. The UTOPIA receive slave will start receiving data in the cycle when the master asserts its start-of-cell (SOC) signal. The bytes are assembled into words and written into the slave-receive queue. Figure 18–8 shows the UTOPIA slave receive interface timing. The clock for the UTOPIA slave receive interface, URCLK, is an input driven by the external master.

Figure 18–8. ATM Controller Slave Receive Timing



18.4.5 Slave-Receive Queue

When the master initiates the transfers to the slave, the slave-receive queue generates an UREVT to the EDMA or UINT to the CPU (if desired) when at least one cell worth of data is available. The event is cleared as soon as the first read is performed by the EDMA or CPU. The next event is generated when the next cell is fully available and the process repeats. In summary, the receive event is generated and cleared as follows:

1) UREVT is generated when a complete cell is available

2) UREVT is cleared when the first read (by the EDMA) of that cell occurs

3) UREVT is regenerated when the next complete cell is available in the Slave-Receive Queue.

4) Go to step 2.

The CPU can also be used to service the UTOPIA via the UINT signal. See section 18.5 for details.

The UTOPIA slave will agree for reception from the master by asserting its UR-CLAV signal when there is at least one cell space available in the queue. If the slave cannot receive the next cell immediately, it de-asserts its URCLAV at least 4 URCLK cycles before the end of this cell transfer. If it remains asserted, it indicates that the slave can receive another cell from the master. The master may disable TxEnb* on its side ($\overline{\text{URENB}}$ for this ATMC slave) at its discretion.

18.4.6 UTOPIA Events Generation

The UTOPIA transmit and receive queues generate not-full and not-empty events to the EDMA. The events are generated when the queues have space available for at least one cell and not when the queues are completely full or empty. This allows for more throughput and better performance because the transmit and receive data can be continuously transferred without having to wait for a full/empty queue. Refer to section 18.4.3 and section 18.4.5 for details on the generation of these events. Either the EDMA or the CPU can be used to service the UTOPIA in response to these events, as discussed in section 18.5 and section 18.6, respectively. Typically, EDMA is used to service UTOPIA for performance considerations.

18.4.7 Multi-PHY (MPHY) Operation

The UTOPIA interface supports multi-PHY operation as per UTOPIA Level 2 specification. The MPHY mode is enabled when the MPHY bit in the UTOPIA Control Register (UCR) is set to 1 (default state). In MPHY mode, the 5-bit SLID (Slave ID) field in the UCR indicates the PHY address of the UTOPIA. Either the DSP or the external master can program the SLID field. The programming interface can be either the HPI/PCI.

MPHY operation is based on cell-level handshaking. As shown in Figure 18–7 and Figure 18–8, the external ATM master polls for available slave devices before the beginning of the actual data transaction. The UTOPIA output signals URCLAV, UXCLAV, UXSOC, and UXDATA[7:0] are in high-impedance state when the UTOPIA slave is not selected by the master. When the UTOPIA slave detects its address at the UXADDR[4:0] or URADDR[4:0] pins, it asserts the UXCLAV or URCLAV, respectively.

Note: When used in single-PHY mode (MPHY=0 in UCR), there is no need to program the address.

18.5 EDMA Servicing UTOPIA

Typically, the EDMA is used to service the UTOPIA interface. Table 18–5 lists the UTOPIA synchronization events to the EDMA and their corresponding EDMA channel. The following sections describe the EDMA setup to service the UTOPIA transmitter and receiver, respectively.

Table 18–5. EDMA Synchronization Events from UTOPIA

EDMA Event	EDMA Channel	Synchronization Event Description
UXEVT	40	Transmit event from the UTOPIA to EDMA. UXEVT is asserted if the Transmit Queue has space for at least one cell-packet.
UREVT	32	Receive event from the UTOPIA to EDMA. UREVT is asserted if a complete cell-packet is available in the Receive Queue.

18.5.1 EDMA Setup for UTOPIA Transmitter

As mentioned in section 18.4.6, the UTOPIA transmitter generates an UXEVT synchronization event to the EDMA when at least one cell-packet space is available in the Slave-Transmit Queue. EDMA channel 40 is dedicated to the UXEVT event. Per UXEVT synchronization event, one frame of cell-packet data is transferred to the Slave-Transmit Queue. A standard cell-packet comprises of 14 words (56 bytes), while a non–standard cell-packet comprises of 14, 15, or 16 words (56, 60, or 64 bytes). The EDMA access to UTOPIA is always 32-bit.The EDMA source address should point to the UTOPIA source buffer in the DSP memory (internal or external). The EDMA destination address should point to the Slave-Transmit Queue data port UXQ. In summary, the EDMA should have the following parameters setup for transmit operation:

❑ ESIZE = 00b; 32-bit elements

❑ SUM = 01b; Source address in autoincrement mode

❑ DUM = 01b; Destination address in autoincrement mode

❑ FS = 1; Frame synchronized transfer. A complete cell-packet is transferred to the UXQ per UXEVT synchronization event.

❑ SRC Address = Starting address of source buffer

❑ DST Address = Starting address of UXQ (3D00 0000h)

❑ Element Count = 14, 15, or 16; Transfer a cell-packet per UXEVT event

Other EDMA parameters can be setup as desired.

18.5.2 EDMA Setup for UTOPIA Receiver

As mentioned in section 18.4.6, the UTOPIA receiver generates an UREVT synchronization event to the EDMA when the slave-receive queue has space for at least one cell-packet. EDMA channel 23 is dedicated to the UREVT event. Per UREVT synchronization event, one frame of cell-packet data is read from the Slave-Receive Queue via the data port URQ. A standard cell-packet comprises of 14 words (56 bytes), while a non–standard cell-packet comprises of 14, 15, or 16 words (56, 60, or 64 bytes). The EDMA destination address should point to the destination buffer in the DSP memory (internal or external). In summary, the EDMA should have the following parameters setup for receive operation:

- ❑ ESIZE = 00b; 32-bit elements
- ❑ SUM = 01b; Source address in autoincrement mode
- ❑ DUM = 01b; Destination address in autoincrement mode
- ❑ FS = 1; Frame synchronized transfer. A complete cell-packet is read from the URQ per UREVT synchronization event.
- ❑ SRC Address = Starting address of URQ (3C00 0000h)
- ❑ DST Address = Starting address of destination buffer
- ❑ Element Count = 14, 15, or 16; Transfer a cell-packet per UREVT event

Other EDMA parameters can be set up as desired.

18.6 CPU Servicing UTOPIA Interface

In addition to the EDMA, the CPU can also be used to service the UTOPIA interface but is not recommended. The CPU can access the slave-transmit queue and the slave-receive queue via the data ports UXQ and URQ, respectively, as listed in Table 18–2. The UTOPIA interface generates a single CPU interrupt, UINT, for both the transmit and receive interface.

The relevant interrupts for each queue are enabled in the UTOPIA Interrupt Enable Register (UIER), which is shown in Figure 18–9 and summarized in Table 18–6. Interrupts are captured in the UTOPIA Interrupt Pending Register (UIPR), which is shown in Figure 18–10 and summarized in Table 18–7. The transmit interrupt and receive interrupt are generated on the same conditions that a UXEVT or UREVT is generated to the EDMA. See section 18.4.3 and section 18.4.5 for details.

Figure 18–9. UTOPIA Interrupt Enable Register (UIER)

31 17	16
Reserved	RQIE
R, +0	RW, +0

15 1	0
Reserved	XQIE
R, +0	RW, +0

Table 18–6. UTOPIA Interrupt Enable Register (UIER) Bit Field Description

No.	Field	Description
16	RQIE	Receive Queue Interrupt Enable RQIE = 0: Receive Queue Interrupt disabled. No interrupts are sent to the CPU upon the UREVT event. RQIE = 1: Receive Queue Interrupt enabled. Upon UREVT, interrupt UINT is sent to the CPU interrupt selector.
0	XQIE	Transmit Queue Interrupt Enable XQIE = 0: Transmit Queue Interrupt disabled. No interrupts are sent to the CPU upon the UXEVT event. XQIE = 1: Transmit Queue Interrupt enabled. Upon UXEVT, interrupt UINT is sent to the CPU interrupt selector.

Figure 18–10. UTOPIA Interrupt Pending Register (UIPR)

31–17	16
Reserved	RQIP
R, +0	RW, +0

15–1	0
Reserved	XQIP
R, +0	RW, +0

Table 18–7. UTOPIA Interrupt Pending Register (UIPR) Bit Field Descrition

No.	Field	Description

16	RQIP	Receive Queue Interrupt Pending
		RQIP = 0: No Receive Queue Interrupt pending.
		RQIP = 1: Receive Queue Interrupt pending.

0	XQIP	Transmit Queue Interrupt Pending
		XQIP = 0: No Transmit Queue Interrupt pending.
		XQIP = 1: Transmit Queue Interrupt pending.

18.6.1 Interrupt Generation and Servicing

The conditions that generate transmit and receive events to the EDMA are exactly the conditions that generate transmit and receive interrupts to the CPU. Therefore, the relevant interrupt pending bit in the UIPR is equivalent to the UXEVT/UREVT to the EDMA. But the interrupt to the CPU is generated only if the relevant enable bit is set in the UIER. For better system performance, the EDMA should be used to service the UTOPIA. Table 18–8 lists the UTOPIA events that cause the CPU interrupt, UINT.

Table 18–8. CPU Interrupt from UTOPIA

UTOPIA Event	CPU Interrupt	Interrupt Number	Interrupt Description
UXEVT	UINT	23	Transmit interrupt from the Slave-Transmit Queue to the CPU
UREVT	UINT	23	Receive interrupt from the Slave-Receive Queue to the CPU

Note: A single UTOPIA Interrupt is generated to the CPU, if the corresponding bit is set in the UTOPIA Interrupt Enable Register (UIER). The interrupt from individual queues for both transmit and receive is read from the UTOPIA Interrupt Pending Register (UIPR).

The generation and clearing of interrupts is as follows:

- The slave-transmit queue and slave-receive queue generate ready interrupts that set the relevant UIPR bit as shown in Figure 18–10.
- If the relevant bit in the UIER is set and the UTOPIA interrupt to the CPU called UINT (mapped to interrupt number 23) is enabled in the CPU's Interrupt Enable Register (IER), the CPU will be interrupted.
- Within the Interrupt Service Routine (ISR),
  - Read the UIPR to find which queue(s) generated the interrupt.
  - Service the queue(s) as required by performing a cell-packet read from the URQ or a cell-packet write to the UXQ (14 words for a standard cell-packet; 14, 15, or 16 words for a non–standard cell-packet). To service the receiver, a URQ read should always start at address 3C00 0000h. To service the transmitter, a UXQ write should always start at address 3D00 0000h.
  - The ISR clears the bit in UIPR upon servicing that interrupt, thereby enabling recognition of further interrupts from the same queue. Writing a '1' to the relevant bit clears that interrupt in UIPR; writing a '0' has no effect. Table 18–8 shows the interrupt numbers allocated and Table 18–2 shows the location of data ports.

18.7 UTOPIA Clocking and Clock Detection

The transmit and receive clock for the UTOPIA interface is supplied by an external clock source such as the master ATM controller. This allows for accurate clocks as required by most applications. Internal to the DSP, the UTOPIA registers and queues are synchronized to the DSP peripheral clock at a CPU/4 rate.

The Clock Detect Register (CDR) and the UTOPIA clock detection feature allows the DSP to detect the presence of the URCLK and/or UXCLK. The CDR is shown in Figure 18–11 and described in Table 18–9.

Figure 18–11. Clock Detect Register (CDR)

31 – 24	23 – 16
Reserved	XCCNT
R, +0	RW, +FF

15 – 8	7 – 0
Reserved	RCCNT
R, +0	RW, +FF

Table 18–9. Clock Detect Register (CDR) Bit Field Description

No.	Field	Description
23:16	XCCNT	Transmit Clock Count: number of peripheral clock cycles in which the external UTOPIA transmit clock (UXCLK) must have a low to high transition to avoid a reset of the transmit interface. XCCNT = 0: Transmit Clock Detect feature is disabled. XCCNT = 1 to 255: Transmit Clock Detect feature is enabled. XCCNT is the number of peripheral clock cycles before which the next UTOPIA clock edge (UXCLK) must be present. If a UXCLK clock edge is undetected within XCCNT peripheral clock cycles, the transmit UTOPIA port will be reset by hardware. The XCF error bit (XCFP) in the Error Interrupt Pending Register (EIPR) will be set.
7:0	RCCNT	Receive Clock Count: number of peripheral clock cycles in which the external UTOPIA receive clock must have a low to high transition to avoid a reset of the receive interface. RCCNT = 0: Receive Clock Detect feature is disabled. RCCNT = 1 to 255: Receive Clock Detect feature is enabled. RCCNT is the number of peripheral clock cycles before which the next UTOPIA clock edge (URCLK) must be present. If a URCLK clock edge is undetected within RCCNT peripheral clock cycles, the receive UTOPIA port will be reset by hardware. The RCF error bit (RCFP) in the Error Interrupt Pending Register (EIPR) will be set.

If a URCLK or a UXCLK edge is not detected within the respective time period specified in the CDR, an error bit, RCFP or XCFP, respectively, is set in the Error Interrupt Pending Register (EIPR). In addition, the RCPP and XCPP bits in the EIPR indicate the presence of the URCLK and UXCLK, respectively. This is shown in Figure 18–12. See section 18.8 for usage of these interrupts to the CPU.

18.8 Special Transfer Conditions

This section explains how the UTOPIA slave interface handles some of the error conditions.

- ❑ Runt Cells: Runt cells are those cells that are shorter than the standard ATM cell (53 bytes for 8-bit mode). This occurs when the device that sends data asserts SOC in the middle of a cell transfer. In the C64x UTOPIA, runt cells are handled in hardware. If the receive section of the UTOPIA detects a SOC before a complete cell is received, the byte count is reset and the runt cell is overwritten by the next new data. In the transmit direction, the user cannot force SOC. In other words, intentional runt cell generation is not supported.

- ❑ Absence of UTOPIA Clocks: If for any reason during a cell transfer in either direction, the receive clock (URCLK) or transmit clock (UXCLK) stops toggling, the corresponding section of the UTOPIA may be reset depending on the duration of absence. This means the queues are returned to their reset state, and all control registers are reset. In addition, the Receive Clock Failed (RCFP) or Transmit Clock Failed (XCFP) bit, respectively, will be set in the Error Interrupt Pending Register (EIPR). An interrupt UINT is generated to the CPU if the corresponding bits in the Error Interrupt Enable Register (EIER) are set. This helps in recovering from conditions where the master card (that supplies the clocks) is pulled from the system.

  The Receive Clock Present (RCPP) and Transmit Clock Present (XCPP) bits in the EIPR are set if the URCLK and UXCLK are detected, regardless of the state of the UTOPIA port. If the corresponding bits are enabled in the EIER, an interrupt UINT is generated. This is useful in re-enabling the UTOPIA ports once the UTOPIA clocks are detected.

- ❑ Abrupt Reset: In slave mode, typically the master issues a reset command via the management interface to ensure graceful stop of transfers. But if this is violated, data corruption will occur and the system software should comprehend this. Summarily, abrupt reset causes data loss/corruption. It is the user’s responsibility to avoid such conditions.

- ❑ Queue Read/Write Stall Conditions: There are two potential stall conditions when the user attempts to read or write to the transmit/receive queues when they are not ready (i.e. when UXEVT and UREVT are not active). They are:

  - Writes to a FULL Slave-Transmit Queue: Writing to the transmit queue that is full will render the queue not ready. In other words, writes are stalled until the queue is drained and space is available for further writes. Therefore, data is not overwritten. When such a condition occurs, the error status bit Transmit Queue Stall (XQSP) in the EIPR will be set to indicate the stall condition. The XQSP bit a read-only bit. It is cleared once the queue has space available and writes can continue.

  - Reads from an EMPTY Slave-Receive Queue: Attempting to read a queue that has no data results in stalling that operation until valid data is available. This also sets the Receive Queue Stall (RQSP) bit in the EIPR to indicate the read stall condition. This is a user error because a read access is performed when there is no active UREVT. The RQSP bit is a read-only bit. It is cleared as soon as valid data is available in the receive queue and the read is performed.

18.8.1 Error Interrupt Registers (EIPR, EIER)

The UTOPIA error conditions are recorded in the Error Interrupt Pending Register (EIPR), which is shown in Figure 18–12 and summarized in Table 18–10. A user write of ‘1’ to the XCPP, XCFP, RCPP, or RCFP field clears the corresponding bit. A user write of ‘0’ has no effect. The XQSP and RQSP fields are read-only bits, and they are cleared automatically by the UTOPIA interface once the error conditions go away. The error conditions in the EIPR can generate an interrupt to the CPU if the corresponding bits are set in the Error Interrupt Enable Register (EIER), shown in Figure 18–13 and summarized in Table 18–11.

Figure 18–12. Error Interrupt Pending Register (EIPR)

31 19	18	17	16
Reserved	XCPP	XCFP	XQSP
R, +0	RW, +0	RW, +0	R, +0

15 3	2	1	0
Reserved	RCPP	RCFP	RQSP
R, +0	RW, +0	RW, +0	R, +0

Table 18–10. Error Interrupt Pending Register (EIPR) Bit Field Description

No.	Field	Description
18	XCPP	Transmit Clock Present Interrupt Pending. XCPP indicates if the UTOPIA transmit clock UXCLK is present. XCPP is valid regardless if the transmit interface is enabled or not (UXEN = don't care). XCPP = 0: UXCLK not present. XCPP = 1: UXCLK present. If the corresponding bit in the EIER is set, an interrupt UINT is sent to the CPU.
17	XCFP	Transmit Clock Failed Interrupt Pending. XCFP is activated only when the UTOPIA transmit interface is enabled (UXEN = 1). XCFP = 0: UXCLK present. XCFP= 1: UXCLK failed. No UXCLK is detected for a period longer than that specified in the XCCNT field of the CDR. If the corresponding bit in the EIER is set, an interrupt UINT is sent to the CPU.
16	XQSP	Transmit Queue Stall Interrupt Pending. XQSP = 0: No Transmit Queue stall condition. XQSP = 1: Transmit Queue stalled—a write is performed to a full transmit queue. The write is stalled until the queue is drained and space is available. Data is not overwritten. XQSP is cleared once the queue has space available and writes can continue.
2	RCPP	Receive Clock Present Interrupt Pending. RCPP indicates if the UTOPIA receive clock URCLK is present. RCPP is valid regardless if the receive interface is enabled or not (UREN = don't care). RCPP = 0: URCLK not present. RCPP = 1: URCLK present. If the corresponding bit in the EIER is set, an interrupt UINT is sent to the CPU.
1	RCFP	Receive Clock Failed Interrupt Pending. RCFP is activated only when the UTOPIA receive interface is enabled (UREN = 1). RCFP = 0: URCLK present. RCFP = 1: URCLK failed. No URCLK is detected for a period longer than that specified in the RCCNT field of the CDR. If the corresponding bit in the EIER is set, an interrupt UINT is sent to the CPU.
0	RQSP	Receive Queue Stall Interrupt Pending RQSP = 0: No Receive Queue stall condition. RQSP = 1: Receive Queue stalled—a read is performed from an empty receive queue. The read is stalled until valid data is available in the queue. RQSP is cleared as soon as valid data is available and the read is performed.

Figure 18–13. Error Interrupt Enable Register (EIER)

31–19	18	17	16
Reserved	XCPE	XCFE	XQSE
R, +0	RW, +0	RW, +0	RW, +0

15–3	2	1	0
Reserved	RCPE	RCFE	RQSE
R, +0	RW, +0	RW, +0	RW, +0

Table 18–11. Error Interrupt Enable Register (EIER) Bit Field Description

No.	**Field**	**Description**

18	XCPP	Transmit Clock Present Interrupt Enable.
		XCPP = 0: Transmit Clock Present interrupt disabled.
		XCPP = 1: Transmit Clock Present interrupt enabled

17	XCFP	Transmit Clock Failed Interrupt Enable.
		XCFP = 0: Transmit Clock Failed interrupt disabled.
		XCFP = 1: Transmit Clock Failed interrupt enabled.

16	XQSP	Transmit Queue Stall Interrupt Enable.
		XQSP = 0: Transmit Queue Stall interrupt disabled.
		XQSP = 1: Transmit Queue Stall interrupt enabled.

2	RCPP	Receive Clock Present Interrupt Enable.
		RCPP = 0: Receive Clock Present interrupt disabled.
		RCPP = 1: Receive Clock Present interrupt enabled.

1	RCFP	Receive Clock Failed Interrupt Enable.
		RCFP = 0: Receive Clock Failed interrupt disabled.
		RCFP = 1: Receive Clock Failed interrupt enabled.

0	RQSP	Receive Queue Stall Interrupt Enable
		RQSP = 0: Receive Queue Stall interrupt disabled.
		RQSP = 1: Receive Queue Stall interrupt enabled.

18.9 Endian Considerations

For 8-bit operation, bytes are assembled into words in the UTOPIA queues. Device endian configuration is selected during reset. See Chapter 11*, Boot Modes and Configuration*, of *TMS320C6000 Peripherals Reference Guide* (SPRU190) for details. Pin-level endian configuration ensures the desired endian mode for the DSP/CPU. In order for the UTOPIA interface to present the data transferred across its interface to the DSP/CPU in accordance with the DSP's device endian mode, the endian bit BEND in UCR has to be programmed. By default/reset, the UTOPIA data is presented to the DSP in little-endian format. If big-endian is preferred, BEND should be programmed to '1'. The data bytes are swapped in hardware based on BEND value.

The following sections explain how the data to/from the ATM buffers are stored in the UTOPIA queues based on endianness.

18.10 Slave-Mode Endian Data Formats

When the DSP is a UTOPIA Slave in a system, it only communicates to the master. Therefore the communication from the slave’s perspective is always point-to-point. The standard cell-packet transfer format is shown in Figure 18–4. A non-standard cell-packet transfer format is shown in Figure 18–5.

Depending on the user-defined cell and endian format chosen [(R/X)UDC and BEND in UCR], bytes are placed in the transmit and receive queues as shown in Figure 18–14 through Figure 18–19.

The cell-packet format in Figure 18–14 through Figure 18–19 indicate the data stored in the DSP memory (internal or external) and in the transmit/receive queues. Only the ATM data, including the header and payload information but not the reserved bytes (as shown in Figure 18–4 and Figure 18–5), is actually sent or received across the UTOPIA pins.

Figure 18–14. Little-Endian (BEND=0) and RUDC/XUDC=0 in 8-Bit UTOPIA Slave Mode

Queue Bits	31:24	23:16	15:8	7:0	
Address n	Header 1	Dummy	Dummy	Dummy	Word 0
Address n+4	UDF	Header 4	Header 3	Header 2	Word 1
Address n+8	Payload 4	Payload 3	Payload 2	Payload 1	|
::	::	::	::	::	:
Address n+48	Payload 44	Payload 43	Payload 42	Payload 41	|
Address n+52	Payload 48	Payload 47	Payload 46	Payload 45	Word 13

Figure 18–15. Big-Endian (BEND=1) and RUDC/XUDC=0 in 8-Bit UTOPIA Slave Mode

Queue Bits	31:24	23:16	15:8	7:0	
Address n	Dummy	Dummy	Dummy	Header 1	Word 0
Address n+4	Header 2	Header 3	Header 4	UDF	Word 1
Address n+8	Payload 1	Payload 2	Payload 3	Payload 4	|
::	::	::	::	::	:
Address n+48	Payload 41	Payload 42	Payload 43	Payload 44	|
Address n+52	Payload 45	Payload 46	Payload 47	Payload 48	Word 13

Figure 18–16. Big-Endian (BEND=1) and RUDC/XUDC=1 in 8-Bit UTOPIA Slave Mode

Queue Bits	31:24	23:16	15:8	7:0	
Address n	Dummy	Dummy	UDB 1	Header 1	Word 0
Address n+4	Header 2	Header 3	Header 4	UDF	Word 1
Address n+8	Payload 1	Payload 2	Payload 3	Payload 4	|
::	::	::	::	::	:
Address n+48	Payload 41	Payload 42	Payload 43	Payload 44	|
Address n+52	Payload 45	Payload 46	Payload 47	Payload 48	Word 13

Figure 18–17. Little-Endian (BEND=0) and RUDC/XUDC=7 in 8-Bit UTOPIA Slave Mode

Queue Bits	31:24	23:16	15:8	7:0	
Address n	UDB 4	UDB 3	UDB 2	UDB 1	Word 0
Address n+4	Header 1	UDB 7	UDB 6	UDB 5	Word 1
Address n+8	UDF	Header 4	Header 3	Header 2	:
Address n+12	Payload 4	Payload 3	Payload 2	Payload 1	|
::	::	::	::	::	:
Address n+56	Payload 48	Payload 47	Payload 46	Payload 45	Word 14

Figure 18–18. Little-Endian (BEND=0) and RUDC/XUDC=11 in 8-Bit UTOPIA Slave Mode

Queue Bits	31:24	23:16	15:8	7:0	
Address n	UDB 4	UDB 3	UDB 2	UDB 1	Word 0
Address n+4	UDB 8	UDB 7	UDB 6	UDB 5	Word 1
Address n+8	Header 1	UDB 11	UDB 10	UDB 9	Word 2
Address n+12	UDF	Header 4	Header 3	Header 2	:
Address n+16	Payload 4	Payload 3	Payload 2	Payload 1	|
::	::	::	::	::	:
Address n+60	Payload 48	Payload 47	Payload 46	Payload 45	Word 15

Figure 18–19. Big-Endian (BEND=1) and RUDC/XUDC=11 in 8-Bit UTOPIA Slave Mode

Queue Bits	31:24	23:16	15:8	7:0	
Address n	UDB 1	UDB 2	UDB 3	UDB 4	Word 0
Address n+4	UDB 5	UDB 6	UDB 7	UDB 8	Word 1
Address n+8	UDB 9	UDB 10	UDB 11	Header 1	Word 2
Address n+12	Header 2	Header 3	Header 4	UDF	:
Address n+16	Payload 1	Payload 2	Payload 3	Payload 4	|
::	::	::	::	::	:
Address n+60	Payload 45	Payload 46	Payload 47	Payload 48	Word 15

18.11 UTOPIA Reset

The UTOPIA interface is in reset state during device reset. The UTOPIA interface can also be reset through software by programming the UREN and UXEN bits in the UTOPIA Control Register (UCR) when the device is out of reset.

Table 18–12 shows the reset values of the UTOPIA pins. The UTOPIA pins have no internal pull-up or pull-down resistors. The pins should be pulled externally to bring inputs to a known state when not driven. At reset, all outputs are driven to a high-impedance state to facilitate MPHY operation. The address pins are pulled high to give the address of a null PHY/slave (which is 11111b) during reset.

Table 18–12. UTOPIA Pin REset Values

UTOPIA Pins	ATM Controller SLAVE (Dir)	Reset Values
UXCLK	In	Low
UXADDR[4:0]	In	High
UXCLAV	Out	Hi-Z
$\overline{\text{UXENB}}$	In	High
UXSOC	Out	Hi-Z
UXDATA[7:0]	Out	Hi-Z
URCLK	In	Low
URADDR[4:0]	In	High
URCLAV	Out	Hi-Z
$\overline{\text{URENB}}$	In	High
URSOC	In	Low
URDATA[7:0]	In	Low

18.11.1 UTOPIA Slave Enable Sequence

A device reset or a programmable reset via the UCR resets the UTOPIA interface. To initialize the UTOPIA interface for slave operation, the following steps are required:

- ❑ The UTOPIA master device in the system provides the clock input to URCLK and UXCLK. UTOPIA port can not be initialized without these clocks.
- ❑ Ensure that DSP/chip is out of reset.
- ❑ Program EDMA channel(s) for data transmission and reception to/from the UTOPIA interface.
- ❑ Set up the UTOPIA configuration registers as required.
- ❑ Either the DSP or the external ATM master can write the address of this slave/PHY in the UCR. The external ATM master can write to the UCR via the HPI/PCI or McBSP interfaces.
- ❑ Take the interface out of reset by setting the UREN bit to 1 to enable Receive interface and UXEN to 1 to enable the transmit interface. Note that the transmit and receive interfaces are independent of each other. In typical systems, both are used.

18.12 ATM Adaptation Layer (AAL) Functions

The UTOPIA interface provides a standard hardware interface between an ATM Layer device (master) and a PHY device (slave). The ATM adaptation layer functions such as segmentation and re-assembly (SAR) for AAL2, AAL5 should be implemented in software.

Appendix A

EDMA Transfers

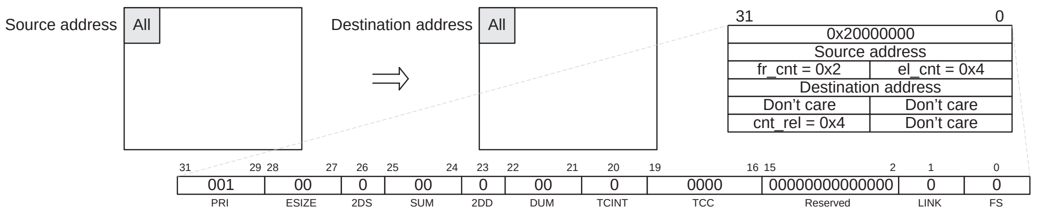
This appendix describes all of the different types of EDMA transfers.

Topic **Page**

A.1 Element Synchronized 1-D to 1-D Transfers

The possible 1-D to 1-D transfers, along with the necessary parameters using element synchronization (FS=0), are shown in Figure A–1 through Figure A–16. For each, only one element is transferred per synchronization event.

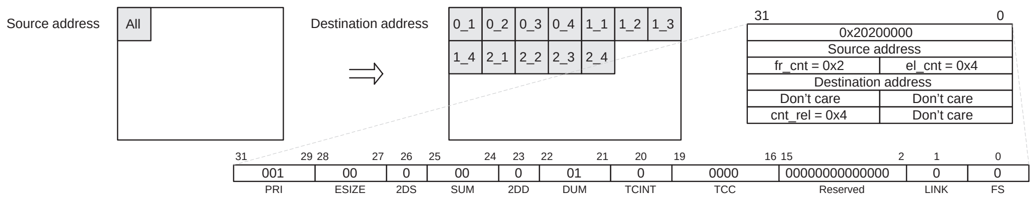
Figure A–1. Element Synchronized 1-D (SUM=00b) to 1-D (DUM=00b)



31	0
0x20000000	
Source address	
fr_cnt = 0x2	el_cnt = 0x4
Destination address	
Don’t care	Don’t care
cnt_rel = 0x4	Don’t care

31 29	28 27	26	25 24	23	22 21	20	19 16	15 2	1	0
001	00	0	00	0	00	0	0000	00000000000000	0	0
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

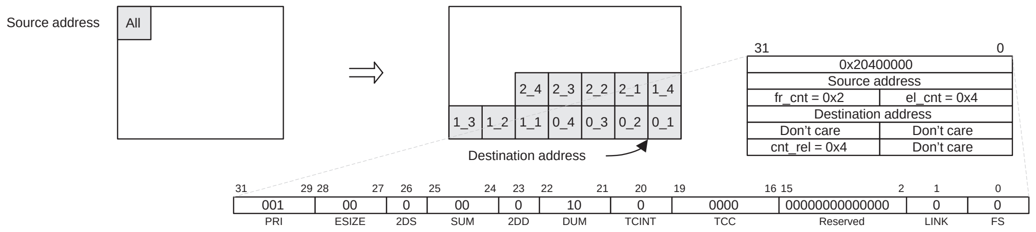
Figure A–2. Element Synchronized 1-D (SUM=00b) to 1-D (DUM=01b)



31	0
0x20200000	
Source address	
fr_cnt = 0x2	el_cnt = 0x4
Destination address	
Don’t care	Don’t care
cnt_rel = 0x4	Don’t care

31 29	28 27	26	25 24	23	22 21	20	19 16	15 2	1	0
001	00	0	00	0	01	0	0000	00000000000000	0	0
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

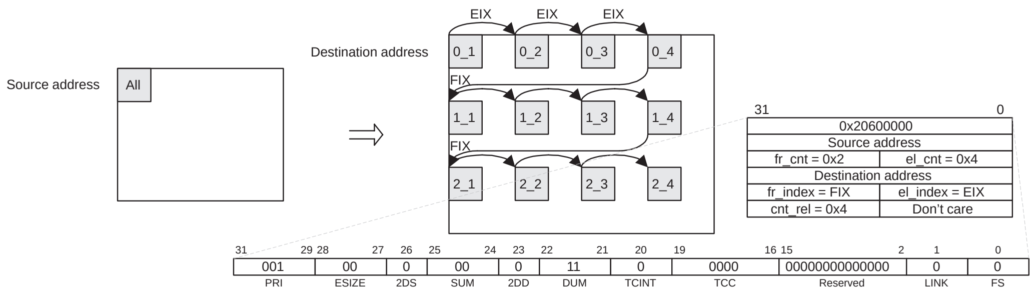
Figure A–3. Element Synchronized 1-D (SUM=00b) to 1-D (DUM=10b)



31	0
0x20400000	
Source address	
fr_cnt = 0x2	el_cnt = 0x4
Destination address	
Don’t care	Don’t care
cnt_rel = 0x4	Don’t care

31 29	28 27	26	25 24	23	22 21	20	19 16	15 2	1	0
001	00	0	00	0	10	0	0000	00000000000000	0	0
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

Figure A–4. Element Synchronized 1-D (SUM=00b) to 1-D (DUM=11b)



31	0
0x20600000	
Source address	
fr_cnt = 0x2	el_cnt = 0x4
Destination address	
fr_index = FIX	el_index = EIX
cnt_rel = 0x4	Don’t care

31 29	28 27	26	25 24	23	22 21	20	19 16	15 2	1	0
001	00	0	00	0	11	0	0000	00000000000000	0	0
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

Figure A–5. Element Synchronized 1-D (SUM=01b) to 1-D (DUM=00b)

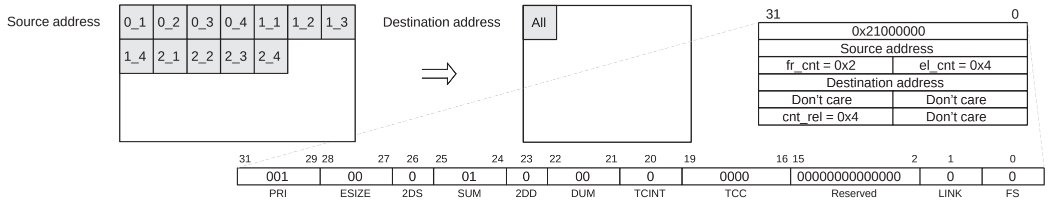


Figure A–6. Element Synchronized 1-D (SUM=01b) to 1-D (DUM=01b)

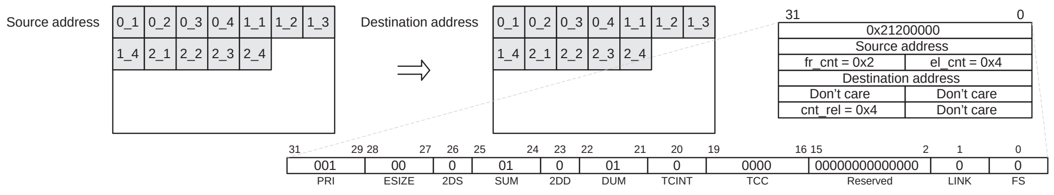


Figure A–7. Element Synchronized 1-D (SUM=01b) to 1-D (DUM=10b)

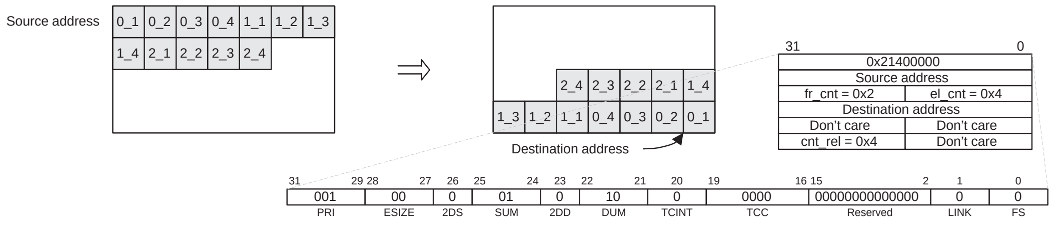


Figure A–8. Element Synchronized 1-D (SUM=01b) to 1-D (DUM=11b)

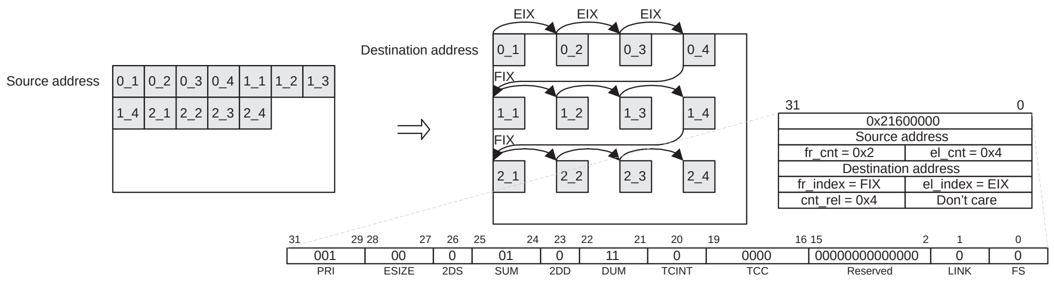
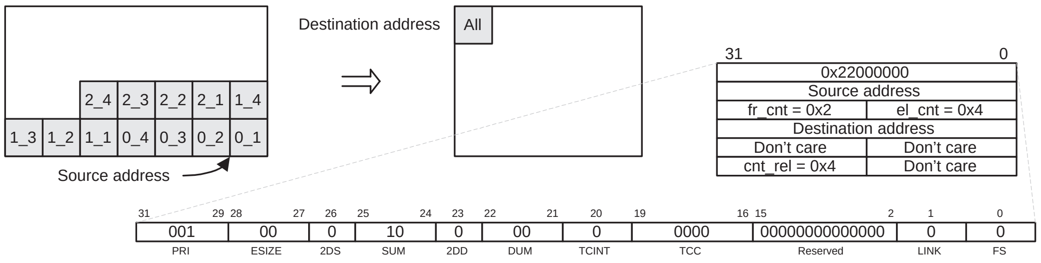


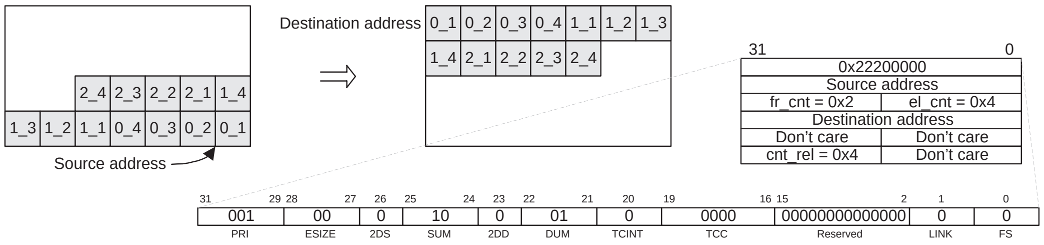
Figure A–9. Element Synchronized 1-D (SUM=10b) to 1-D (DUM=00b)



31	0
0x22000000	
Source address	
fr_cnt = 0x2	el_cnt = 0x4
Destination address	
Don't care	Don't care
cnt_rel = 0x4	Don't care

31–29	28–27	26	25–24	23	22–21	20	19–16	15–2	1	0
001	00	0	10	0	00	0	0000	00000000000000	0	0
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

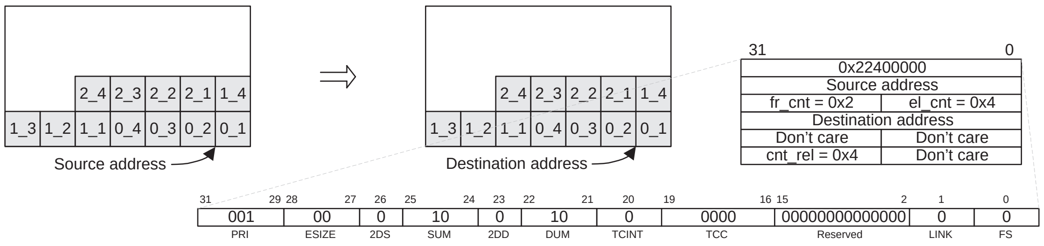
Figure A–10. Element Synchronized 1-D (SUM=10b) to 1-D (DUM=01b)



31	0
0x22200000	
Source address	
fr_cnt = 0x2	el_cnt = 0x4
Destination address	
Don't care	Don't care
cnt_rel = 0x4	Don't care

31–29	28–27	26	25–24	23	22–21	20	19–16	15–2	1	0
001	00	0	10	0	01	0	0000	00000000000000	0	0
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

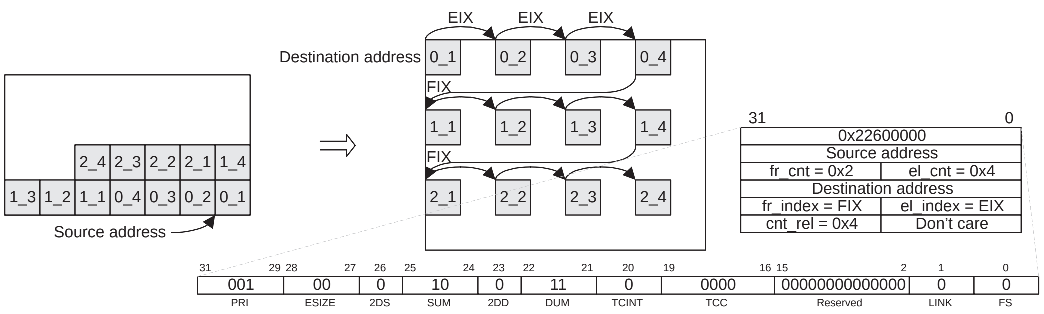
Figure A–11. Element Synchronized 1-D (SUM=10b) to 1-D (DUM=10b)



31	0
0x22400000	
Source address	
fr_cnt = 0x2	el_cnt = 0x4
Destination address	
Don't care	Don't care
cnt_rel = 0x4	Don't care

31–29	28–27	26	25–24	23	22–21	20	19–16	15–2	1	0
001	00	0	10	0	10	0	0000	00000000000000	0	0
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

Figure A–12. Element Synchronized 1-D (SUM=10b) to 1-D (DUM=11b)



31	0
0x22600000	
Source address	
fr_cnt = 0x2	el_cnt = 0x4
Destination address	
fr_index = FIX	el_index = EIX
cnt_rel = 0x4	Don't care

31–29	28–27	26	25–24	23	22–21	20	19–16	15–2	1	0
001	00	0	10	0	11	0	0000	00000000000000	0	0
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

Figure A–13. Element Synchronized 1-D (SUM=11b) to 1-D (DUM=00b)

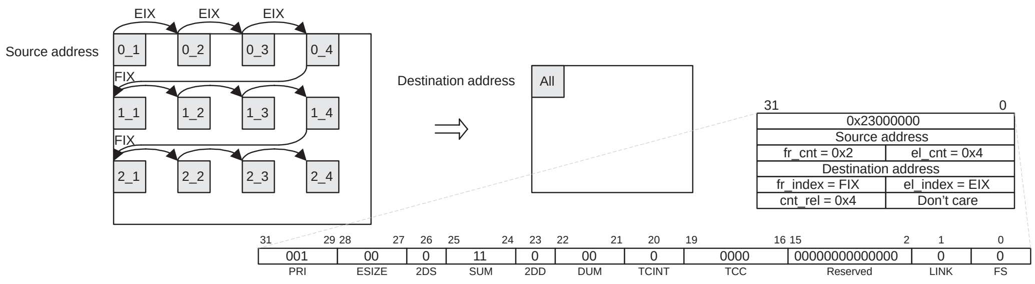


Figure A–14. Element Synchronized 1-D (SUM=11b) to 1-D (DUM=01b)

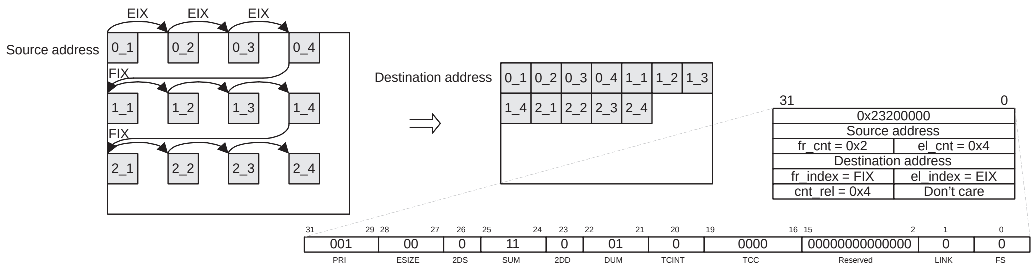


Figure A–15. Element Synchronized 1-D (SUM=11b) to 1-D (DUM=10b)

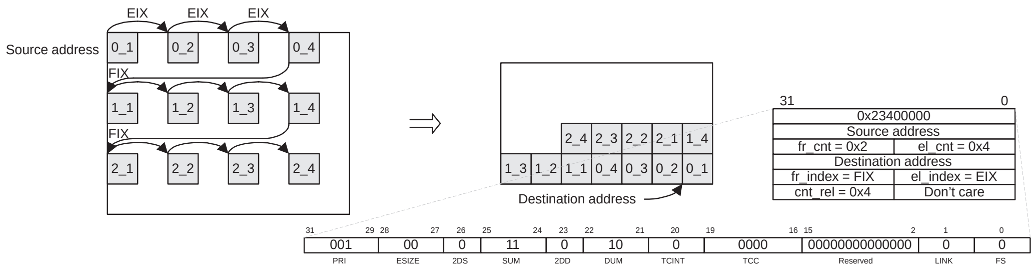
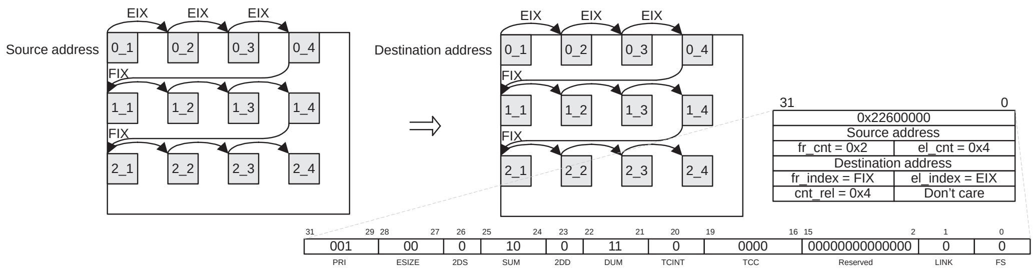


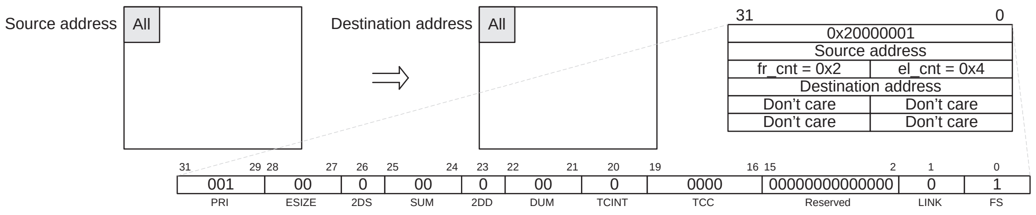
Figure A–16. Element Synchronized 1-D (SUM=11b) to 1-D (DUM=11b)



A.2 Frame-Synchronized 1-D to 1-D Transfers

The possible 1-D to 1-D transfers, along with the necessary parameters using frame synchronization (FS = 1) are shown in Figure A–17 through Figure A–32. For each, an entire frame of elements is transferred per synchronization event.

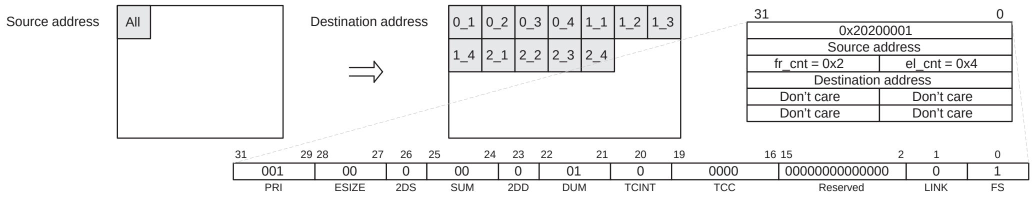
Figure A–17. Frame-synchronized 1-D (SUM=00b) to 1-D (DUM=00b)



31	0
0x20000001	
Source address	
fr_cnt = 0x2	el_cnt = 0x4
Destination address	
Don’t care	Don’t care
Don’t care	Don’t care

31 29	28 27	26	25 24	23	22 21	20	19 16	15 2	1	0
001	00	0	00	0	00	0	0000	00000000000000	0	1
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

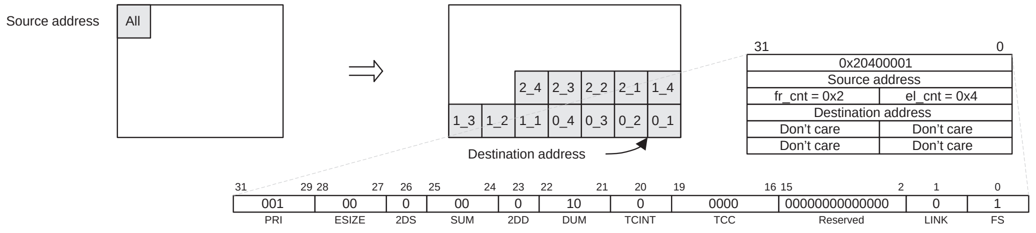
Figure A–18. Frame-synchronized 1-D (SUM=00b) to 1-D (DUM=01b)



31	0
0x20200001	
Source address	
fr_cnt = 0x2	el_cnt = 0x4
Destination address	
Don’t care	Don’t care
Don’t care	Don’t care

31 29	28 27	26	25 24	23	22 21	20	19 16	15 2	1	0
001	00	0	00	0	01	0	0000	00000000000000	0	1
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

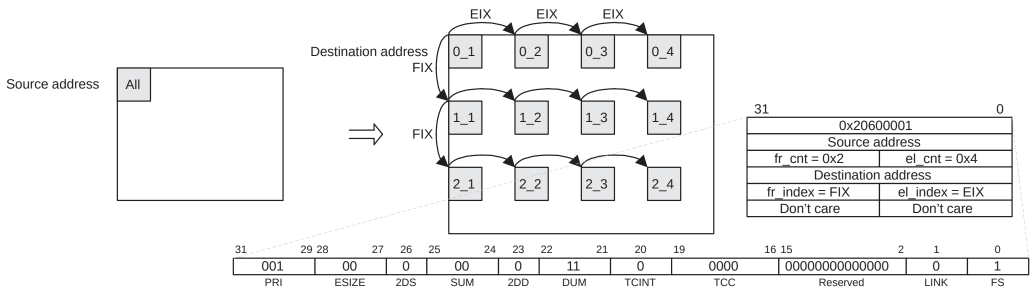
Figure A–19. Frame-synchronized 1-D (SUM=00b) to 1-D (DUM=10b)



31	0
0x20400001	
Source address	
fr_cnt = 0x2	el_cnt = 0x4
Destination address	
Don’t care	Don’t care
Don’t care	Don’t care

31 29	28 27	26	25 24	23	22 21	20	19 16	15 2	1	0
001	00	0	00	0	10	0	0000	00000000000000	0	1
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

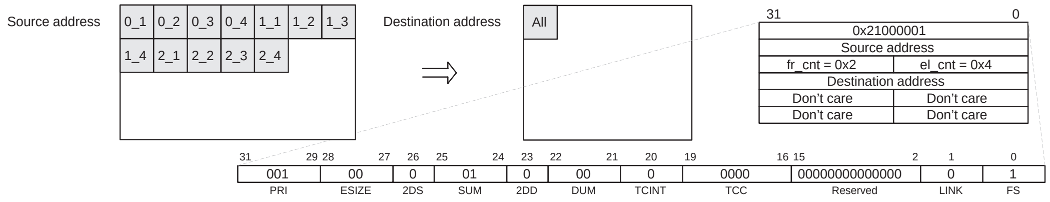
Figure A–20. Frame-synchronized 1-D (SUM=00b) to 1-D (DUM=11b)



31	0
0x20600001	
Source address	
fr_cnt = 0x2	el_cnt = 0x4
Destination address	
fr_index = FIX	el_index = EIX
Don’t care	Don’t care

31 29	28 27	26	25 24	23	22 21	20	19 16	15 2	1	0
001	00	0	00	0	11	0	0000	00000000000000	0	1
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

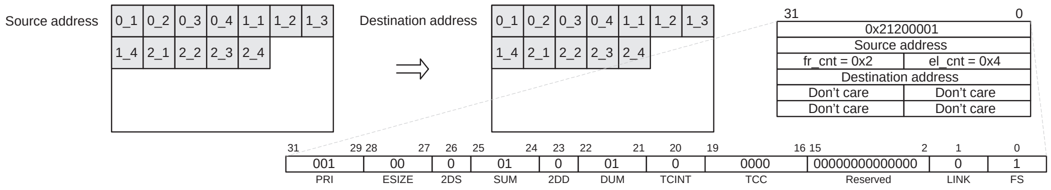
Figure A–21. Frame-synchronized 1-D (SUM=01b) to 1-D (DUM=00b)



0x21000001	
Source address	
fr_cnt = 0x2	el_cnt = 0x4
Destination address	
Don't care	Don't care
Don't care	Don't care

PRI (31–29)	ESIZE (28–27)	2DS (26)	SUM (25–24)	2DD (23)	DUM (22–21)	TCINT (20)	TCC (19–16)	Reserved (15–2)	LINK (1)	FS (0)
001	00	0	01	0	00	0	0000	00000000000000	0	1

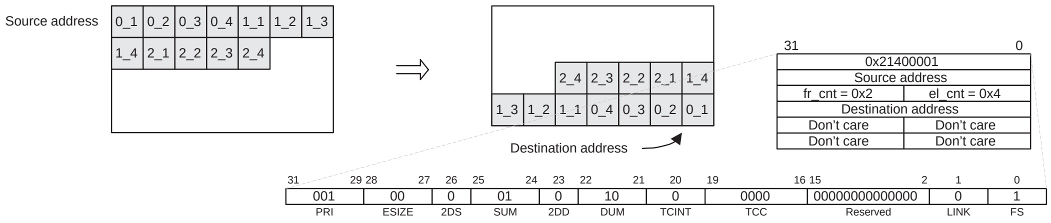
Figure A–22. Frame-synchronized 1-D (SUM=01b) to 1-D (DUM=01b)



0x21200001	
Source address	
fr_cnt = 0x2	el_cnt = 0x4
Destination address	
Don't care	Don't care
Don't care	Don't care

PRI (31–29)	ESIZE (28–27)	2DS (26)	SUM (25–24)	2DD (23)	DUM (22–21)	TCINT (20)	TCC (19–16)	Reserved (15–2)	LINK (1)	FS (0)
001	00	0	01	0	01	0	0000	00000000000000	0	1

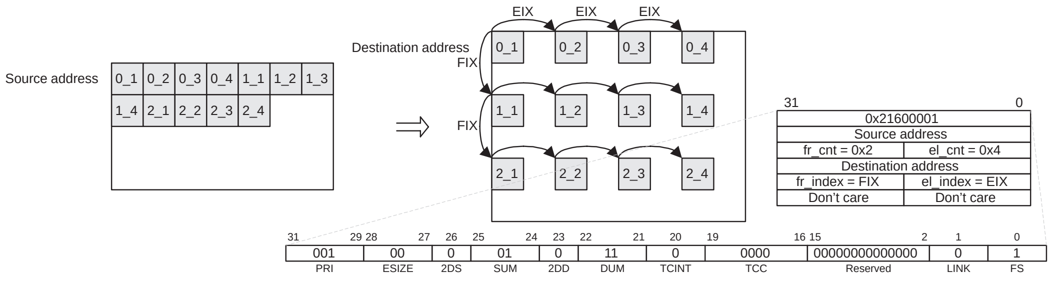
Figure A–23. Frame-synchronized 1-D (SUM=01b) to 1-D (DUM=10b)



0x21400001	
Source address	
fr_cnt = 0x2	el_cnt = 0x4
Destination address	
Don't care	Don't care
Don't care	Don't care

PRI (31–29)	ESIZE (28–27)	2DS (26)	SUM (25–24)	2DD (23)	DUM (22–21)	TCINT (20)	TCC (19–16)	Reserved (15–2)	LINK (1)	FS (0)
001	00	0	01	0	10	0	0000	00000000000000	0	1

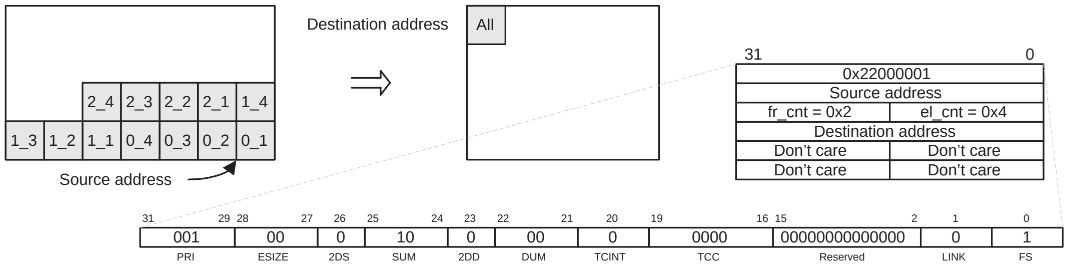
Figure A–24. Frame-synchronized 1-D (SUM=01b) to 1-D (DUM=11b)



0x21600001	
Source address	
fr_cnt = 0x2	el_cnt = 0x4
Destination address	
fr_index = FIX	el_index = EIX
Don't care	Don't care

PRI (31–29)	ESIZE (28–27)	2DS (26)	SUM (25–24)	2DD (23)	DUM (22–21)	TCINT (20)	TCC (19–16)	Reserved (15–2)	LINK (1)	FS (0)
001	00	0	01	0	11	0	0000	00000000000000	0	1

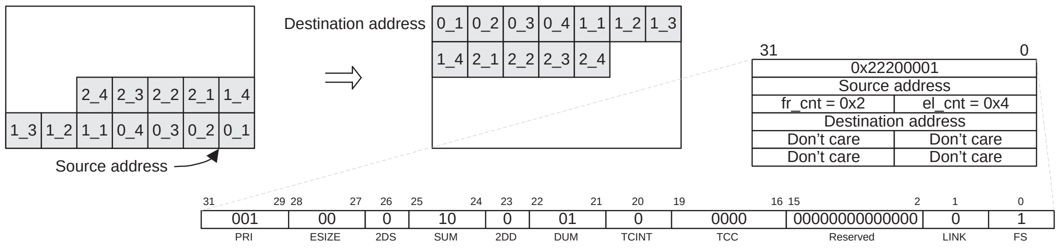
Figure A–25. Frame-synchronized 1-D (SUM=10b) to 1-D (DUM=00b)



31	0
0x22000001	
Source address	
fr_cnt = 0x2	el_cnt = 0x4
Destination address	
Don’t care	Don’t care
Don’t care	Don’t care

31 29	28 27	26	25 24	23	22 21	20	19 16	15 2	1	0
001	00	0	10	0	00	0	0000	00000000000000	0	1
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

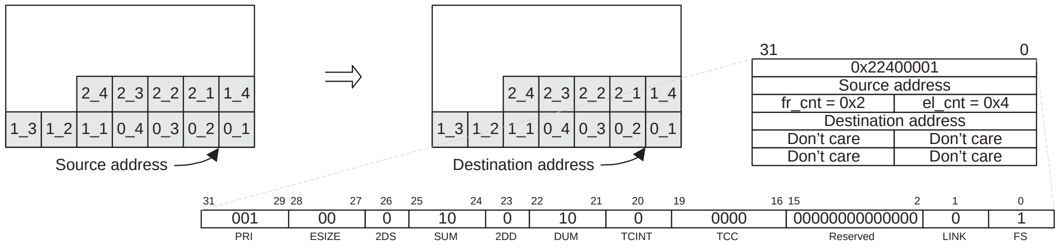
Figure A–26. Frame-synchronized 1-D (SUM=10b) to 1-D (DUM=01b)



31	0
0x22200001	
Source address	
fr_cnt = 0x2	el_cnt = 0x4
Destination address	
Don’t care	Don’t care
Don’t care	Don’t care

31 29	28 27	26	25 24	23	22 21	20	19 16	15 2	1	0
001	00	0	10	0	01	0	0000	00000000000000	0	1
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

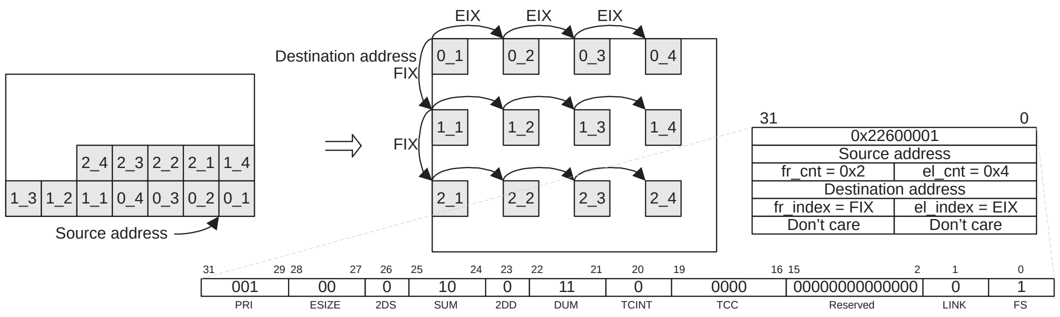
Figure A–27. Frame-synchronized 1-D (SUM=10b) to 1-D (DUM=10b)



31	0
0x22400001	
Source address	
fr_cnt = 0x2	el_cnt = 0x4
Destination address	
Don’t care	Don’t care
Don’t care	Don’t care

31 29	28 27	26	25 24	23	22 21	20	19 16	15 2	1	0
001	00	0	10	0	10	0	0000	00000000000000	0	1
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

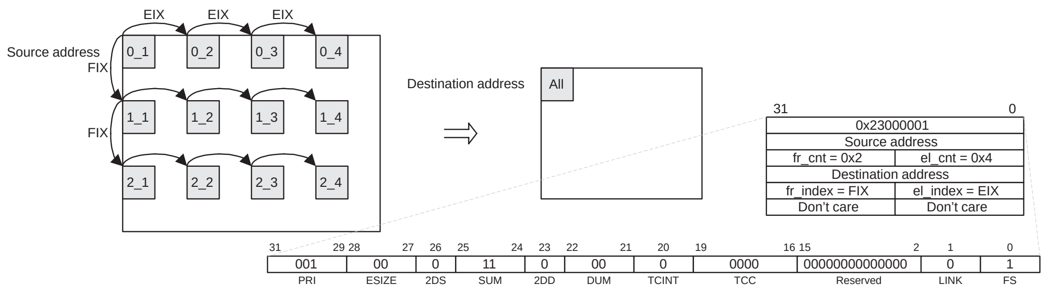
Figure A–28. Frame-synchronized 1-D (SUM=10b) to 1-D (DUM=11b)



31	0
0x22600001	
Source address	
fr_cnt = 0x2	el_cnt = 0x4
Destination address	
fr_index = FIX	el_index = EIX
Don’t care	Don’t care

31 29	28 27	26	25 24	23	22 21	20	19 16	15 2	1	0
001	00	0	10	0	11	0	0000	00000000000000	0	1
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

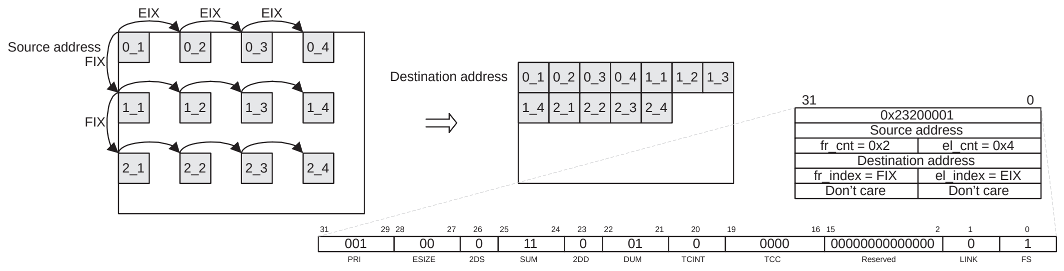
Figure A–29. Frame-synchronized 1-D (SUM=11b) to 1-D (DUM=00b)



0x23000001	
Source address	
fr_cnt = 0x2	el_cnt = 0x4
Destination address	
fr_index = FIX	el_index = EIX
Don't care	Don't care

31 29	28 27	26	25 24	23	22 21	20	19 16	15 2	1	0
001	00	0	11	0	00	0	0000	00000000000000	0	1
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

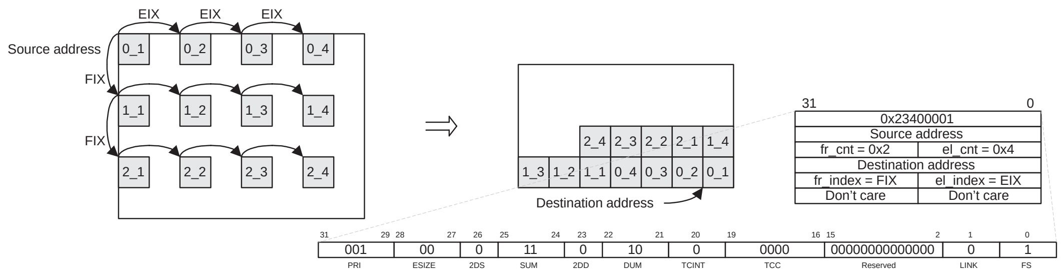
Figure A–30. Frame-synchronized 1-D (SUM=11b) to 1-D (DUM=01b)



0x23200001	
Source address	
fr_cnt = 0x2	el_cnt = 0x4
Destination address	
fr_index = FIX	el_index = EIX
Don't care	Don't care

31 29	28 27	26	25 24	23	22 21	20	19 16	15 2	1	0
001	00	0	11	0	01	0	0000	00000000000000	0	1
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

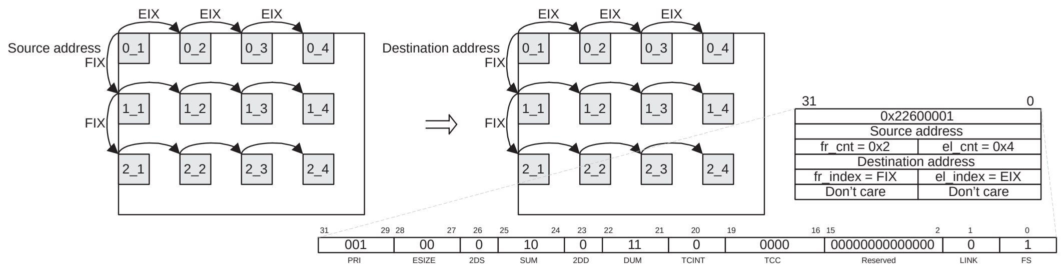
Figure A–31. Frame-synchronized 1-D (SUM=11b) to 1-D (DUM=10b)



0x23400001	
Source address	
fr_cnt = 0x2	el_cnt = 0x4
Destination address	
fr_index = FIX	el_index = EIX
Don't care	Don't care

31 29	28 27	26	25 24	23	22 21	20	19 16	15 2	1	0
001	00	0	11	0	10	0	0000	00000000000000	0	1
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

Figure A–32. Frame-synchronized 1-D (SUM=11b) to 1-D (DUM=11b)



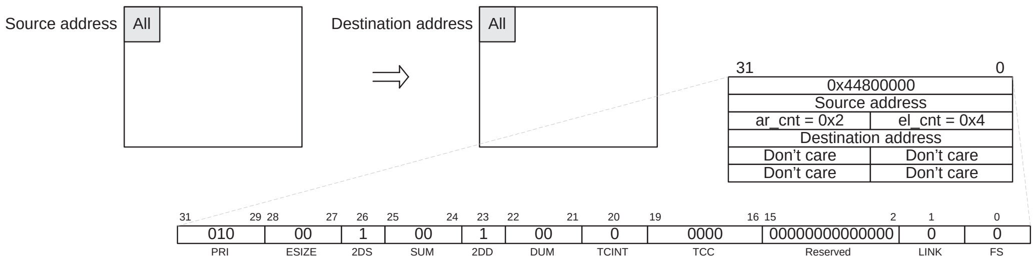
0x22600001	
Source address	
fr_cnt = 0x2	el_cnt = 0x4
Destination address	
fr_index = FIX	el_index = EIX
Don't care	Don't care

31 29	28 27	26	25 24	23	22 21	20	19 16	15 2	1	0
001	00	0	10	0	11	0	0000	00000000000000	0	1
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

A.3 Array Synchronized 2-D to 2-D Transfers

The possible 2-D to 2-D transfers, along with the necessary parameters using array synchronization (FS=0), are shown in Figure A–33 through Figure A–41. For each, a single array of elements is transferred per synchronization event.

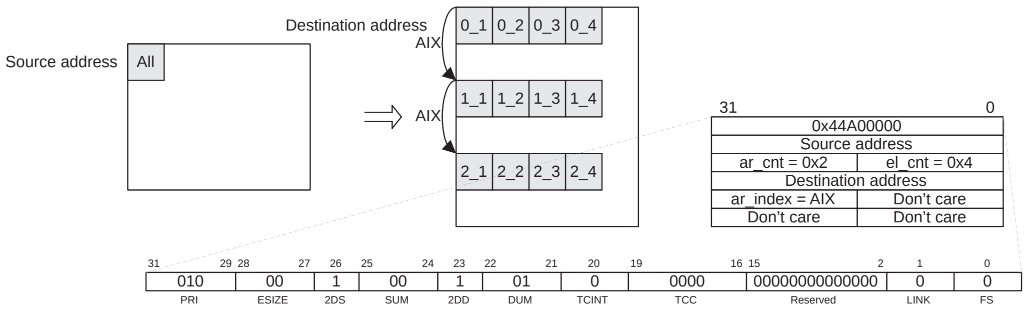
Figure A–33. Array Synchronized 2-D (SUM=00b) to 2-D (DUM=00b)



31	0
0x44800000	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
Don’t care	Don’t care
Don’t care	Don’t care

31–29	28–27	26	25–24	23	22–21	20	19–16	15–2	1	0
010	00	1	00	1	00	0	0000	00000000000000	0	0
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

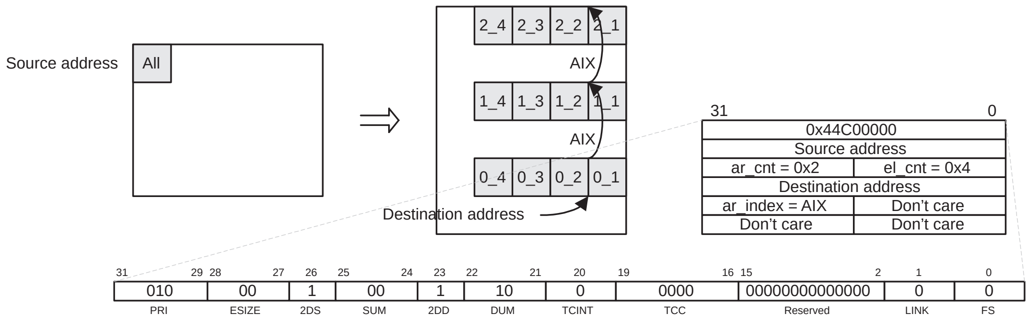
Figure A–34. Array Synchronized 2-D (SUM=00b) to 2-D (DUM=01b)



31	0
0x44A00000	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
ar_index = AIX	Don’t care
Don’t care	Don’t care

31–29	28–27	26	25–24	23	22–21	20	19–16	15–2	1	0
010	00	1	00	1	01	0	0000	00000000000000	0	0
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

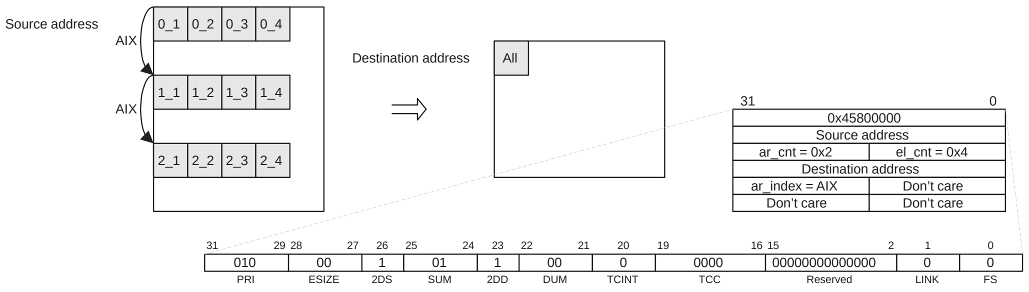
Figure A–35. Array Synchronized 2-D (SUM=00b) to 2-D (DUM=10b)



31	0
0x44C00000	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
ar_index = AIX	Don’t care
Don’t care	Don’t care

31–29	28–27	26	25–24	23	22–21	20	19–16	15–2	1	0
010	00	1	00	1	10	0	0000	00000000000000	0	0
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

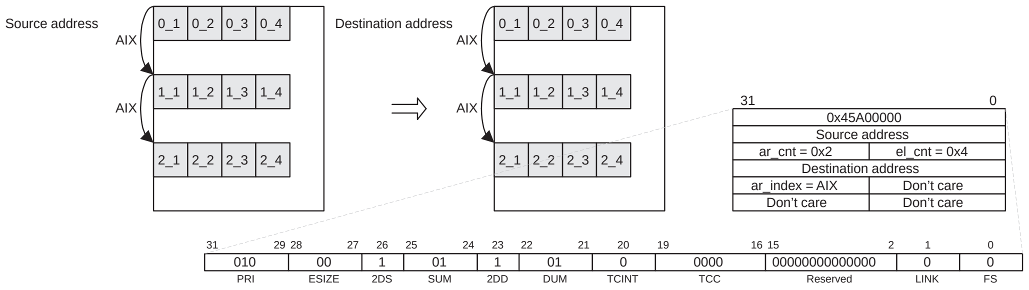
Figure A–36. Array Synchronized 2-D (SUM=01b) to 2-D (DUM=00b)



31	0
0x45800000	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
ar_index = AIX	Don't care
Don't care	Don't care

31–29	28–27	26	25–24	23	22–21	20	19–16	15–2	1	0
010	00	1	01	1	00	0	0000	00000000000000	0	0
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

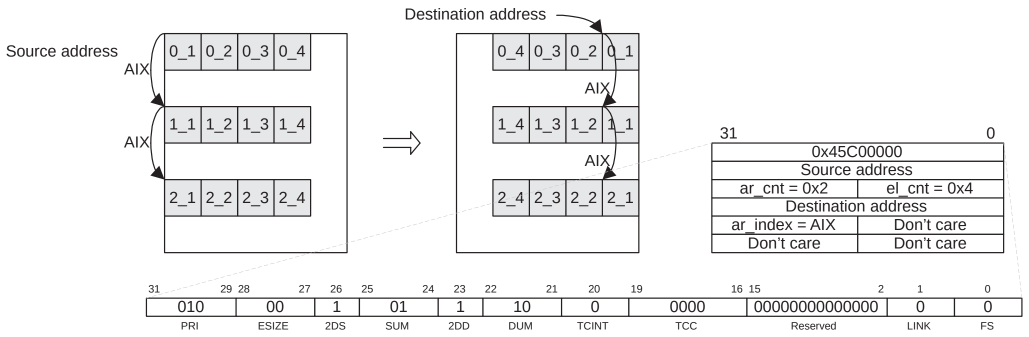
Figure A–37. Array Synchronized 2-D (SUM=01b) to 2-D (DUM=01b)



31	0
0x45A00000	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
ar_index = AIX	Don't care
Don't care	Don't care

31–29	28–27	26	25–24	23	22–21	20	19–16	15–2	1	0
010	00	1	01	1	01	0	0000	00000000000000	0	0
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

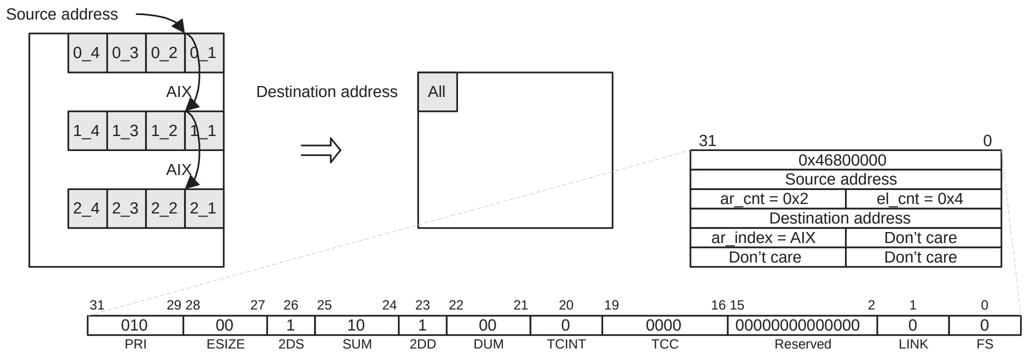
Figure A–38. Array Synchronized 2-D (SUM=01b) to 2-D (DUM=10b)



31	0
0x45C00000	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
ar_index = AIX	Don't care
Don't care	Don't care

31–29	28–27	26	25–24	23	22–21	20	19–16	15–2	1	0
010	00	1	01	1	10	0	0000	00000000000000	0	0
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

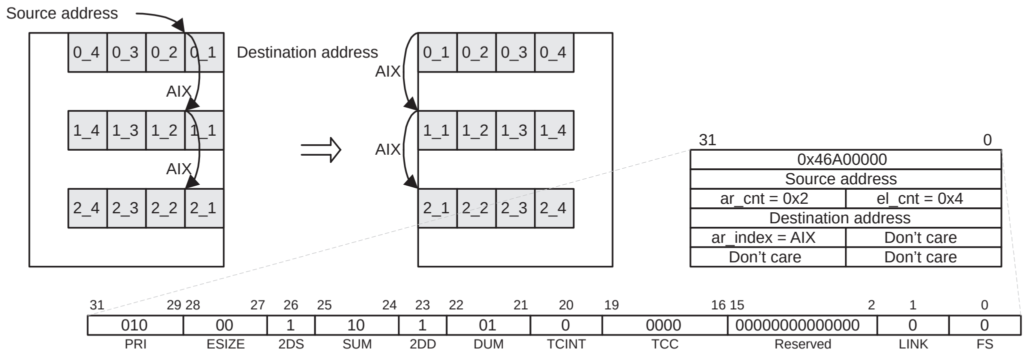
Figure A–39. Array Synchronized 2-D (SUM=10b) to 2-D (DUM=00b)



31	0
0x46800000	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
ar_index = AIX	Don’t care
Don’t care	Don’t care

31–29	28–27	26	25–24	23	22–21	20	19–16	15–2	1	0
010	00	1	10	1	00	0	0000	00000000000000	0	0
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

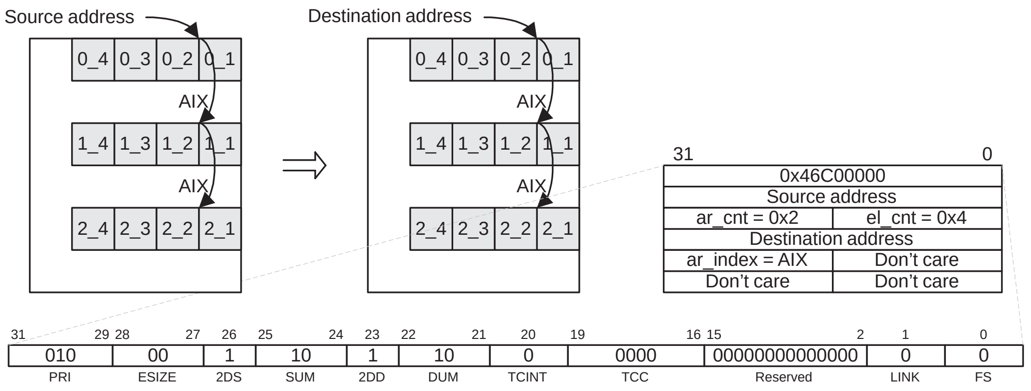
Figure A–40. Array Synchronized 2-D (SUM=10b) to 2-D (DUM=01b)



31	0
0x46A00000	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
ar_index = AIX	Don’t care
Don’t care	Don’t care

31–29	28–27	26	25–24	23	22–21	20	19–16	15–2	1	0
010	00	1	10	1	01	0	0000	00000000000000	0	0
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

Figure A–41. Array Synchronized 2-D (SUM=10b) to 2-D (DUM=10b)



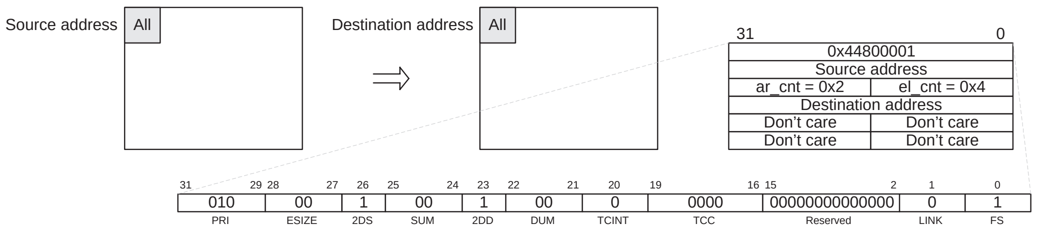
31	0
0x46C00000	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
ar_index = AIX	Don’t care
Don’t care	Don’t care

31–29	28–27	26	25–24	23	22–21	20	19–16	15–2	1	0
010	00	1	10	1	10	0	0000	00000000000000	0	0
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

A.4 Block-Synchronized 2-D to 2-D Transfers

The possible 2-D to 2-D transfers, along with the necessary parameters using block synchronization (FS=1), are shown in Figure A–42 through Figure A–50. For each, an entire block of arrays is transferred per synchronization event.

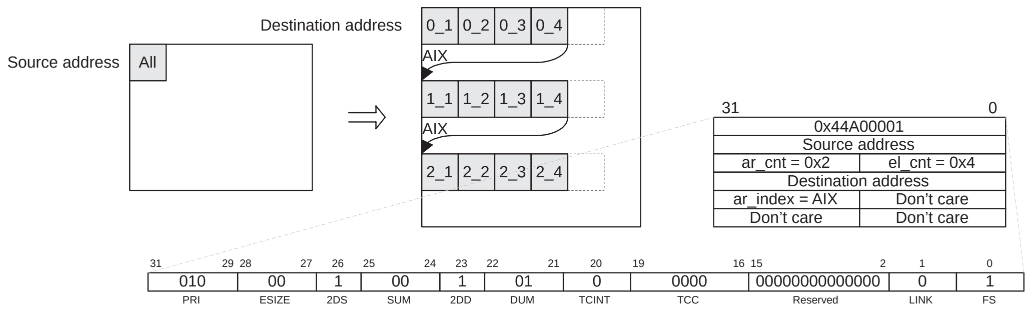
Figure A–42. Block-synchronized 2-D (SUM=00b) to 2-D (DUM=00b)



31	0
0x44800001	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
Don't care	Don't care
Don't care	Don't care

31 29	28 27	26	25 24	23	22 21	20	19 16	15 2	1	0
010	00	1	00	1	00	0	0000	00000000000000	0	1
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

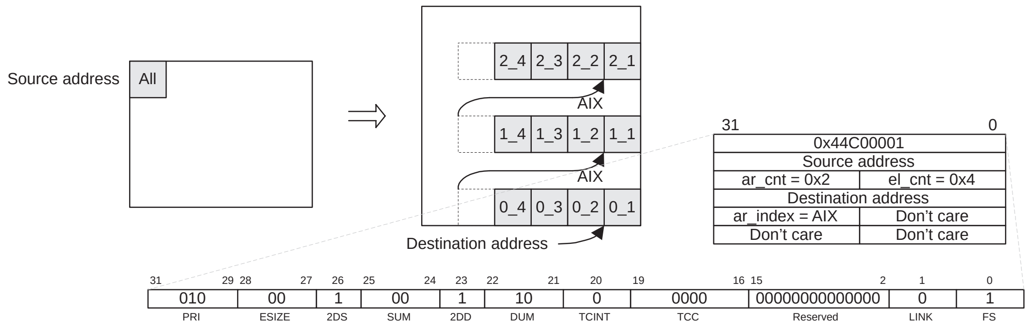
Figure A–43. Block-synchronized 2-D (SUM=00b) to 2-D (DUM=01b)



31	0
0x44A00001	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
ar_index = AIX	Don't care
Don't care	Don't care

31 29	28 27	26	25 24	23	22 21	20	19 16	15 2	1	0
010	00	1	00	1	01	0	0000	00000000000000	0	1
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

Figure A–44. Block-synchronized 2-D (SUM=00b) to 2-D (DUM=10b)

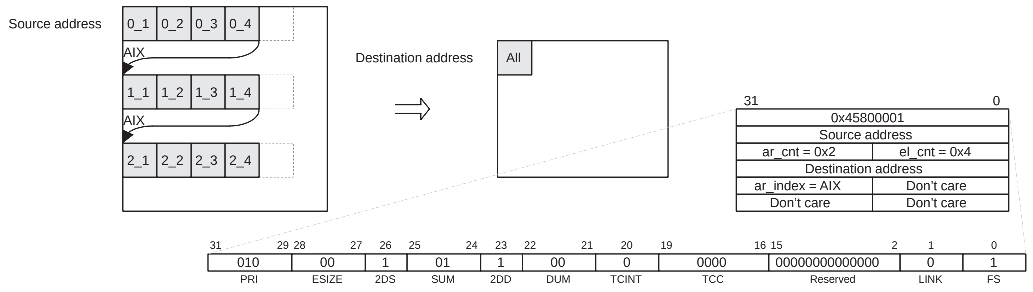


31	0
0x44C00001	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
ar_index = AIX	Don't care
Don't care	Don't care

31 29	28 27	26	25 24	23	22 21	20	19 16	15 2	1	0
010	00	1	00	1	10	0	0000	00000000000000	0	1
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

Note: The AIX is a negative value in this example.

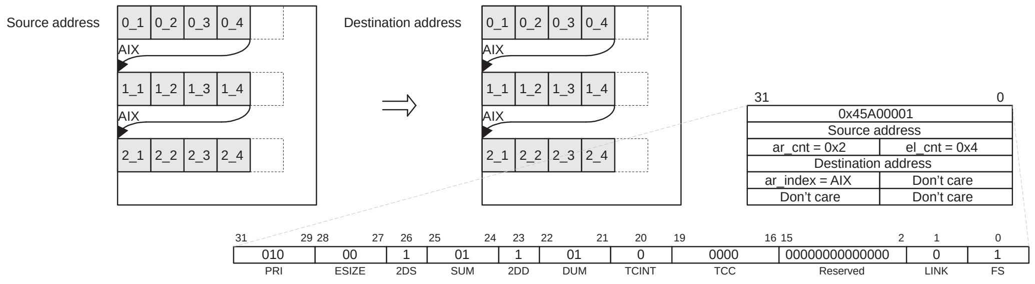
Figure A–45. Block-synchronized 2-D (SUM=01b) to 2-D (DUM=00b)



31	0
0x45800001	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
ar_index = AIX	Don’t care
Don’t care	Don’t care

31–29 PRI	28–27 ESIZE	26 2DS	25–24 SUM	23 2DD	22–21 DUM	20 TCINT	19–16 TCC	15–2 Reserved	1 LINK	0 FS
010	00	1	01	1	00	0	0000	00000000000000	0	1

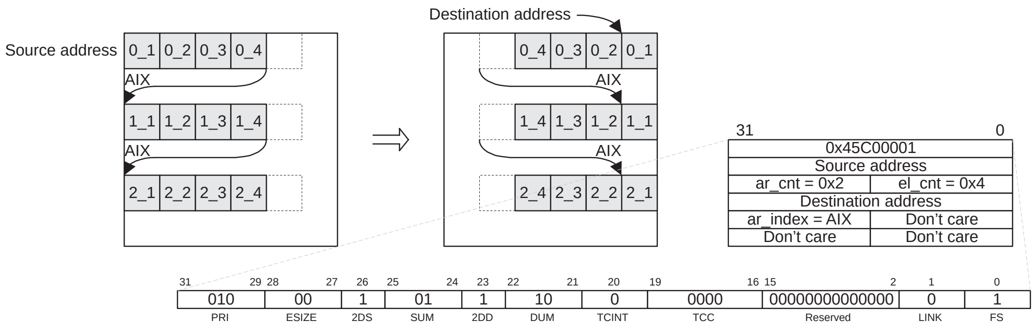
Figure A–46. Block-synchronized 2-D (SUM=01b) to 2-D (DUM=01b)



31	0
0x45A00001	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
ar_index = AIX	Don’t care
Don’t care	Don’t care

31–29 PRI	28–27 ESIZE	26 2DS	25–24 SUM	23 2DD	22–21 DUM	20 TCINT	19–16 TCC	15–2 Reserved	1 LINK	0 FS
010	00	1	01	1	01	0	0000	00000000000000	0	1

Figure A–47. Block-synchronized 2-D (SUM=01b) to 2-D (DUM=10b)



31	0
0x45C00001	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
ar_index = AIX	Don’t care
Don’t care	Don’t care

31–29 PRI	28–27 ESIZE	26 2DS	25–24 SUM	23 2DD	22–21 DUM	20 TCINT	19–16 TCC	15–2 Reserved	1 LINK	0 FS
010	00	1	01	1	10	0	0000	00000000000000	0	1

Figure A–48. Block-synchronized 2-D (SUM=10b) to 2-D (DUM=00b)

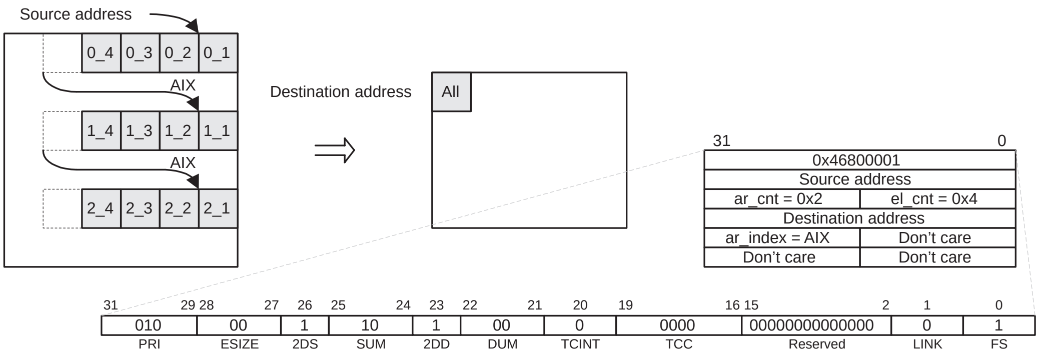


Figure A–49. Block-synchronized 2-D (SUM=10b) to 2-D (DUM=01b)

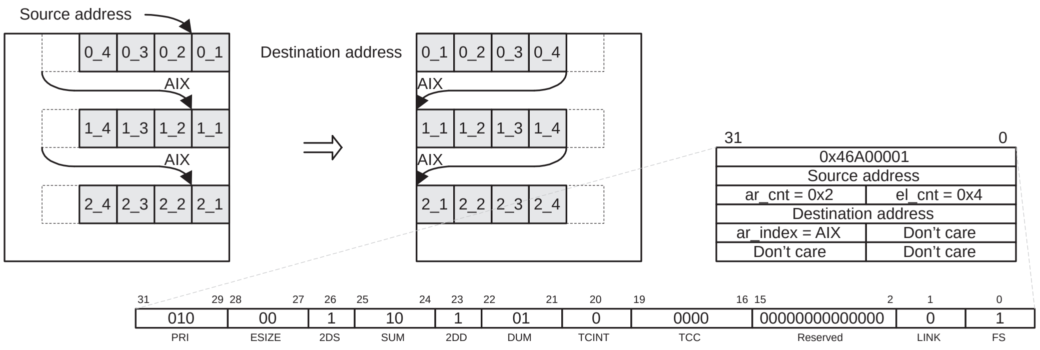
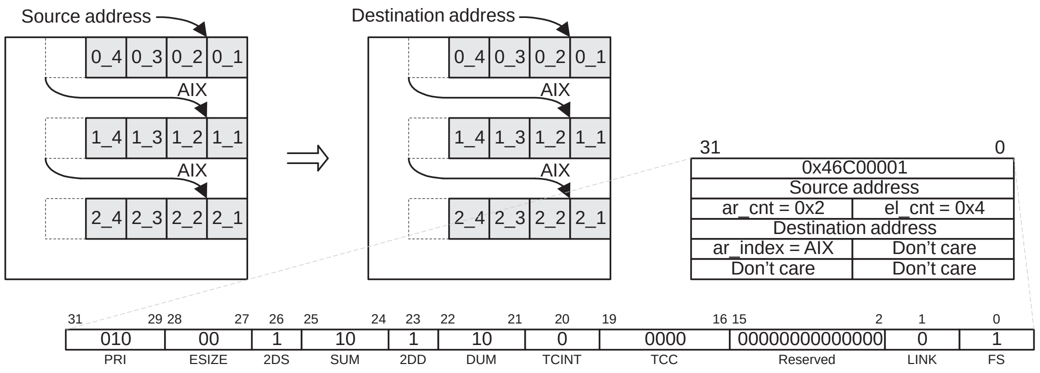


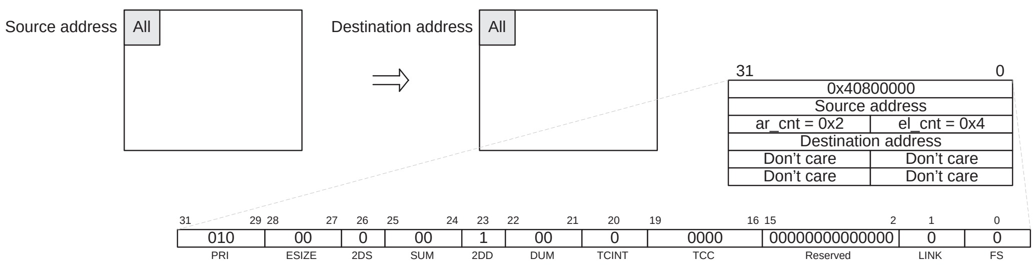
Figure A–50. Block-synchronized 2-D (SUM=10b) to 2-D (DUM=10b)



A.5 Array Synchronized 1-D to 2-D Transfers

The possible 1-D to 2-D transfers, along with the necessary parameters using array synchronization (FS=0), are shown in Figure A–51 through Figure A–59. For each, a single array of elements is transferred per synchronization event.

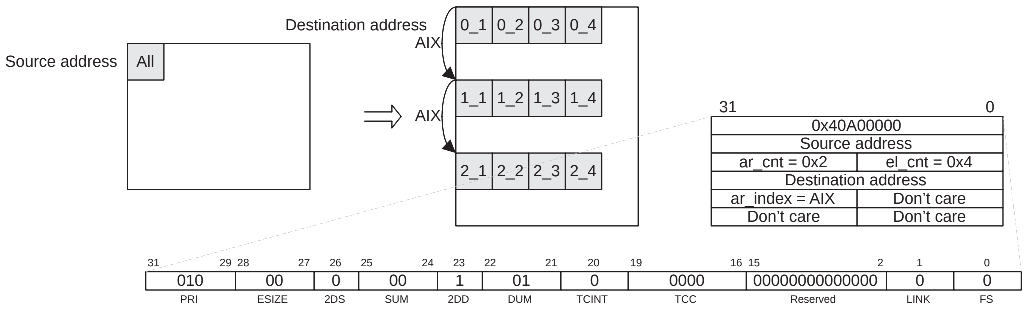
Figure A–51. Array Synchronized 1-D (SUM=00b) to 2-D (DUM=00b)



31	0
0x40800000	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
Don't care	Don't care
Don't care	Don't care

31–29	28–27	26	25–24	23	22–21	20	19–16	15–2	1	0
010	00	0	00	1	00	0	0000	00000000000000	0	0
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

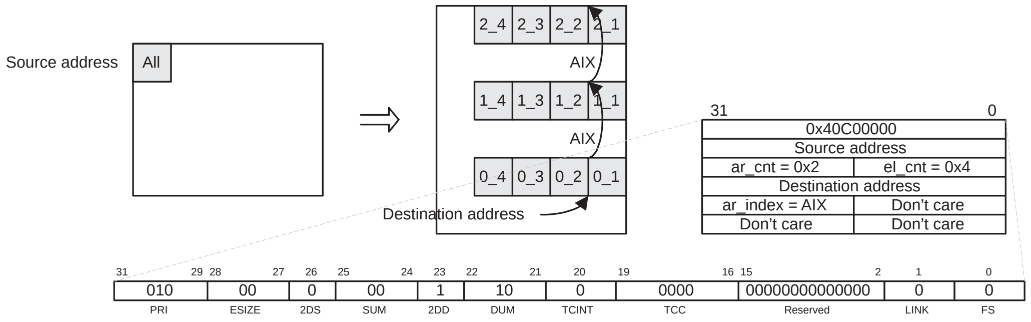
Figure A–52. Array Synchronized 1-D (SUM=00b) to 2-D (DUM=01b)



31	0
0x40A00000	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
ar_index = AIX	Don't care
Don't care	Don't care

31–29	28–27	26	25–24	23	22–21	20	19–16	15–2	1	0
010	00	0	00	1	01	0	0000	00000000000000	0	0
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

Figure A–53. Array Synchronized 1-D (SUM=00b) to 2-D (DUM=10b)

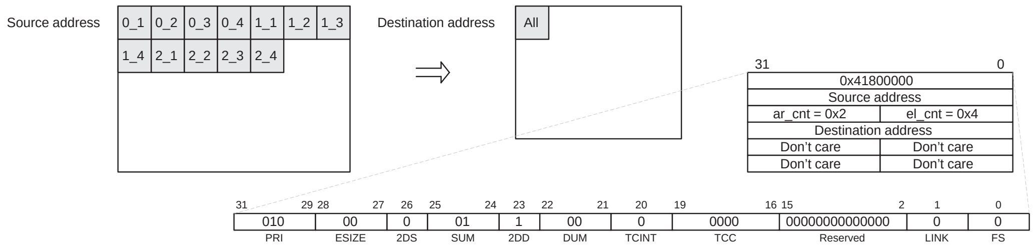


31	0
0x40C00000	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
ar_index = AIX	Don't care
Don't care	Don't care

31–29	28–27	26	25–24	23	22–21	20	19–16	15–2	1	0
010	00	0	00	1	10	0	0000	00000000000000	0	0
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

Note: AIX is a negative value in this example.

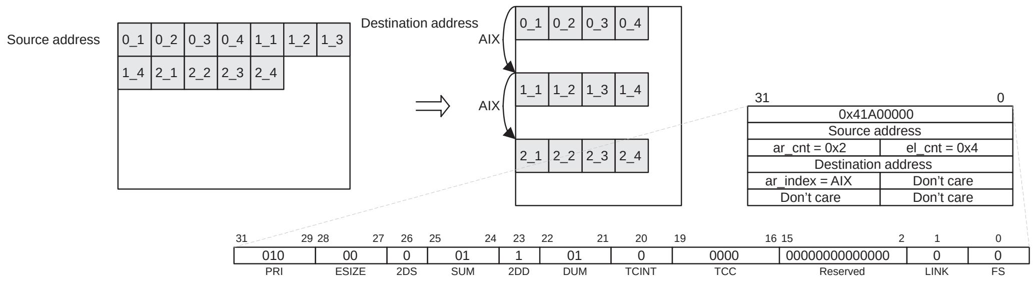
Figure A–54. Array Synchronized 1-D (SUM=01b) to 2-D (DUM=00b)



31	0
0x41800000	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
Don’t care	Don’t care
Don’t care	Don’t care

31 29	28 27	26	25 24	23	22 21	20	19 16	15 2	1	0
010	00	0	01	1	00	0	0000	00000000000000	0	0
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

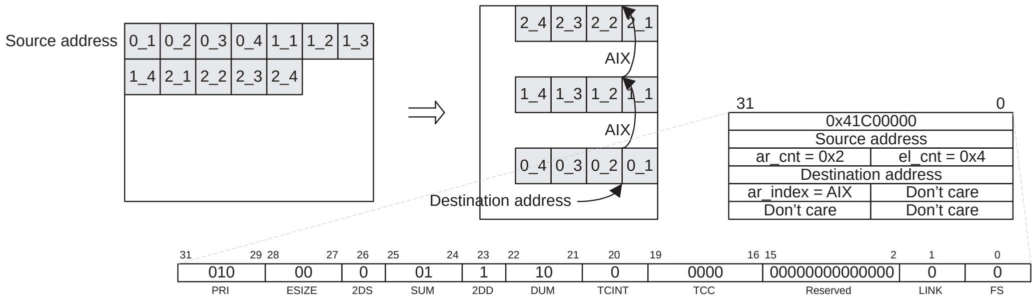
Figure A–55. Array Synchronized 1-D (SUM=01b) to 2-D (DUM=01b)



31	0
0x41A00000	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
ar_index = AIX	Don’t care
Don’t care	Don’t care

31 29	28 27	26	25 24	23	22 21	20	19 16	15 2	1	0
010	00	0	01	1	01	0	0000	00000000000000	0	0
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

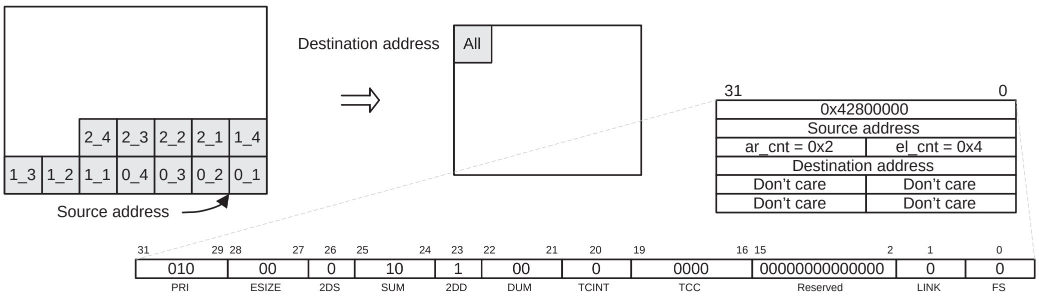
Figure A–56. Array Synchronized 1-D (SUM=01b) to 2-D (DUM=10b)



31	0
0x41C00000	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
ar_index = AIX	Don’t care
Don’t care	Don’t care

31 29	28 27	26	25 24	23	22 21	20	19 16	15 2	1	0
010	00	0	01	1	10	0	0000	00000000000000	0	0
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

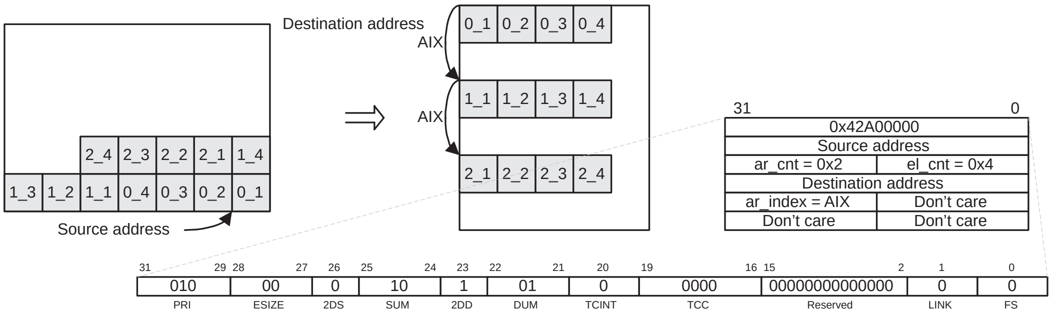
Figure A–57. Array Synchronized 1-D (SUM=10b) to 2-D (DUM=00b)



31	0
0x42800000	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
Don’t care	Don’t care
Don’t care	Don’t care

31 29	28 27	26	25 24	23	22 21	20	19 16	15 2	1	0
010	00	0	10	1	00	0	0000	00000000000000	0	0
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

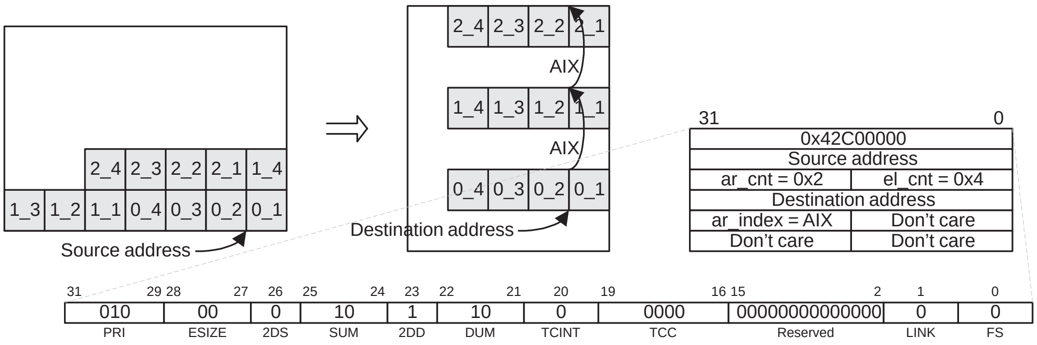
Figure A–58. Array Synchronized 1-D (SUM=10b) to 2-D (DUM=01b)



31	0
0x42A00000	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
ar_index = AIX	Don’t care
Don’t care	Don’t care

31 29	28 27	26	25 24	23	22 21	20	19 16	15 2	1	0
010	00	0	10	1	01	0	0000	00000000000000	0	0
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

Figure A–59. Array Synchronized 1-D (SUM=10b) to 2-D (DUM=10b)



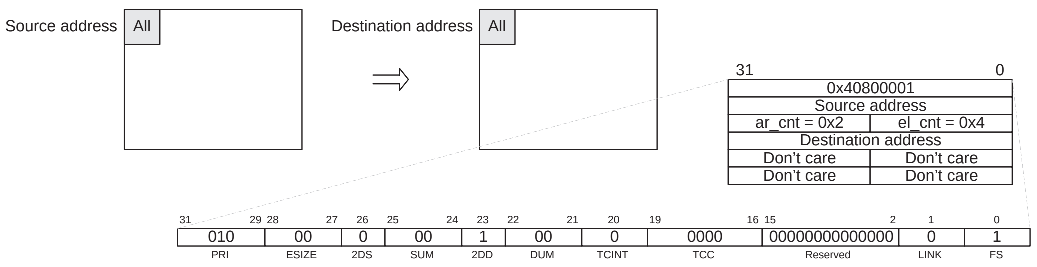
31	0
0x42C00000	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
ar_index = AIX	Don’t care
Don’t care	Don’t care

31 29	28 27	26	25 24	23	22 21	20	19 16	15 2	1	0
010	00	0	10	1	10	0	0000	00000000000000	0	0
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

A.6 Block-Synchronized 1-D to 2-D Transfers

The possible 1-D to 2-D transfers, along with the necessary parameters using block synchronization (FS=1), are shown in Figure A–60 through Figure A–68. For each, an entire block of arrays is transferred per synchronization event.

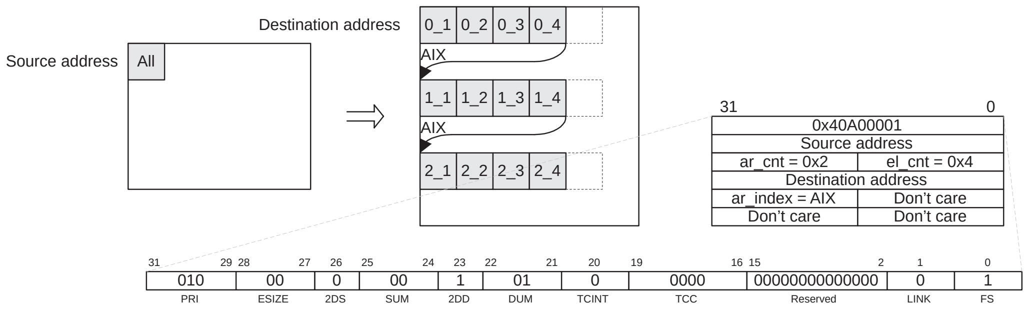
Figure A–60. Block-Synchronized 1-D (SUM=00b) to 2-D (DUM=00b)



31	0
0x40800001	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
Don't care	Don't care
Don't care	Don't care

31 29	28 27	26	25 24	23	22 21	20	19 16	15 2	1	0
010	00	0	00	1	00	0	0000	00000000000000	0	1
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

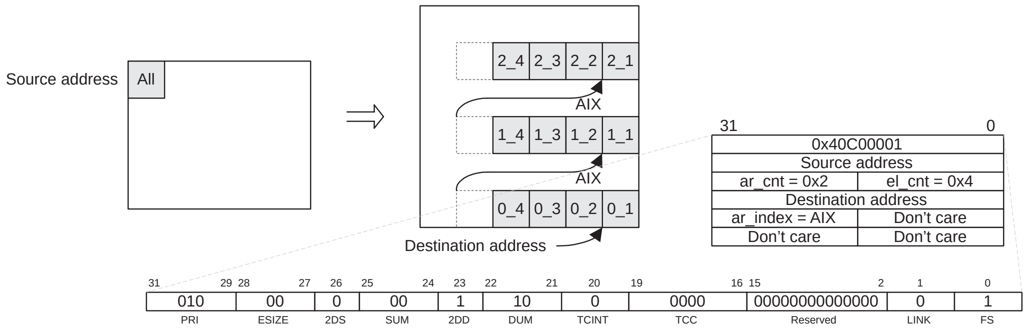
Figure A–61. Block-Synchronized 1-D (SUM=00b) to 2-D (DUM=01b)



31	0
0x40A00001	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
ar_index = AIX	Don't care
Don't care	Don't care

31 29	28 27	26	25 24	23	22 21	20	19 16	15 2	1	0
010	00	0	00	1	01	0	0000	00000000000000	0	1
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

Figure A–62. Block-Synchronized 1-D (SUM=00b) to 2-D (DUM=10b)

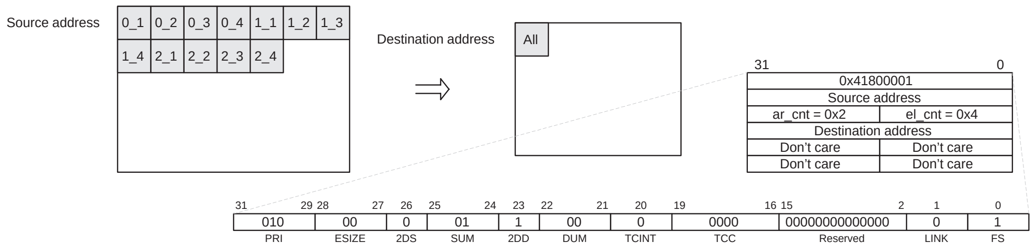


31	0
0x40C00001	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
ar_index = AIX	Don't care
Don't care	Don't care

31 29	28 27	26	25 24	23	22 21	20	19 16	15 2	1	0
010	00	0	00	1	10	0	0000	00000000000000	0	1
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

Note: AIX is a negative value in this example.

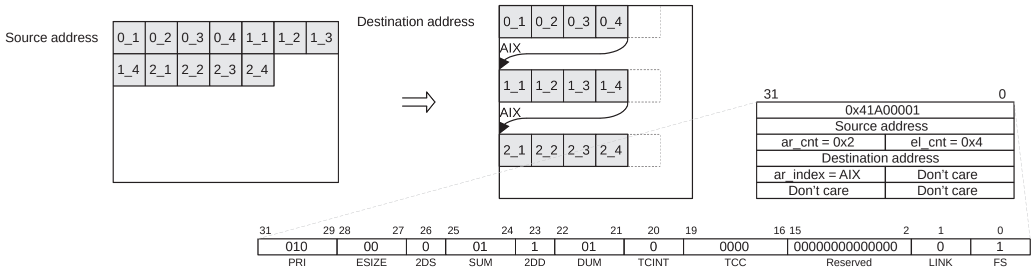
Figure A–63. Block-Synchronized 1-D (SUM=01b) to 2-D (DUM=00b)



31	0
0x41800001	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
Don't care	Don't care
Don't care	Don't care

31–29	28–27	26	25–24	23	22–21	20	19–16	15–2	1	0
010	00	0	01	1	00	0	0000	00000000000000	0	1
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

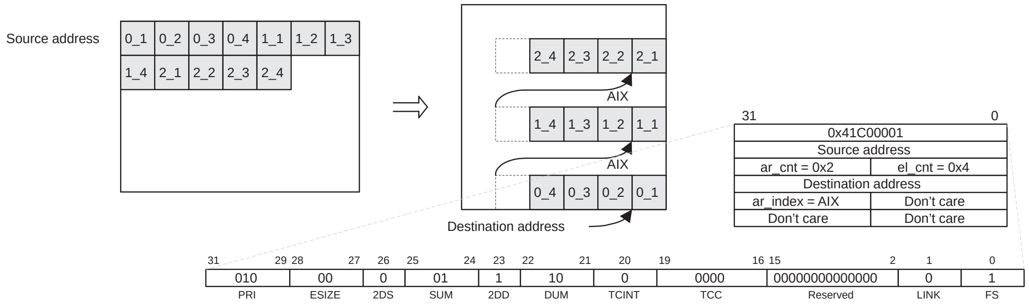
Figure A–64. Block-synchronized 1-D (SUM=01b) to 2-D (DUM=01b)



31	0
0x41A00001	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
ar_index = AIX	Don't care
Don't care	Don't care

31–29	28–27	26	25–24	23	22–21	20	19–16	15–2	1	0
010	00	0	01	1	01	0	0000	00000000000000	0	1
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

Figure A–65. Block-synchronized 1-D (SUM=01b) to 2-D (DUM=10b)

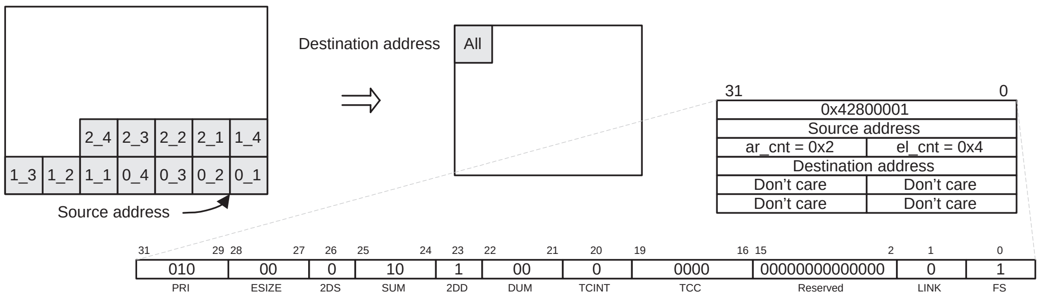


31	0
0x41C00001	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
ar_index = AIX	Don't care
Don't care	Don't care

31–29	28–27	26	25–24	23	22–21	20	19–16	15–2	1	0
010	00	0	01	1	10	0	0000	00000000000000	0	1
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

Note: AIX is a negative value in this example.

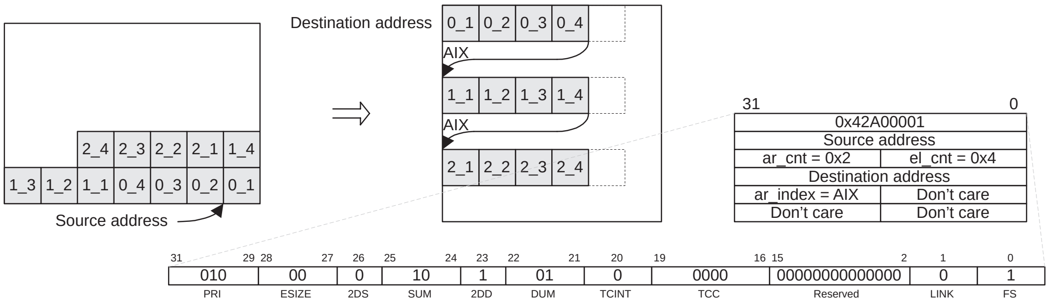
Figure A–66. Block-synchronized 1-D (SUM=10b) to 2-D (DUM=00b)



31	0
0x42800001	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
Don't care	Don't care
Don't care	Don't care

31–29	28–27	26	25–24	23	22–21	20	19–16	15–2	1	0
010	00	0	10	1	00	0	0000	00000000000000	0	1
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

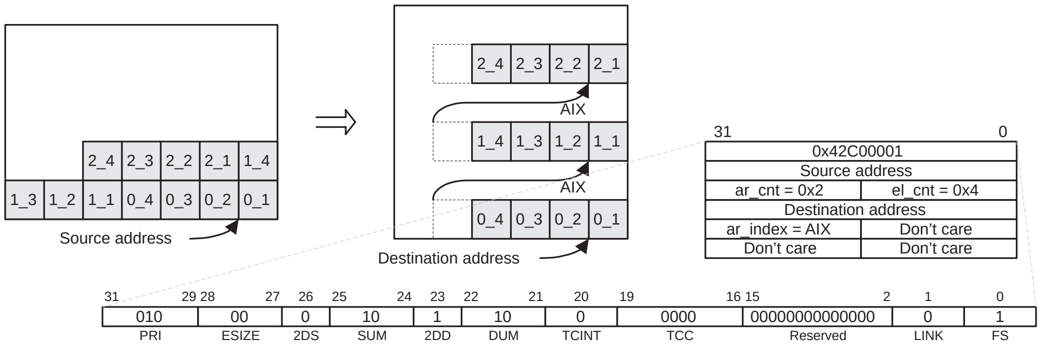
Figure A–67. Block-synchronized 1-D (SUM=10b) to 2-D (DUM=01b)



0x42A00001	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
ar_index = AIX	Don’t care
Don’t care	Don’t care

31–29	28–27	26	25–24	23	22–21	20	19–16	15–2	1	0
010	00	0	10	1	01	0	0000	00000000000000	0	1
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

Figure A–68. Block-synchronized 1-D (SUM=10b) to 2-D (DUM=10b)



0x42C00001	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
ar_index = AIX	Don’t care
Don’t care	Don’t care

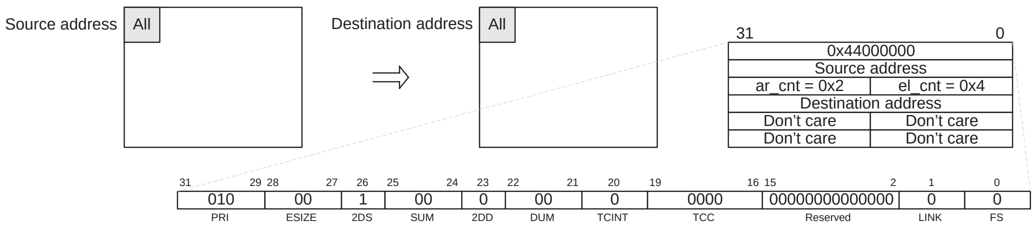
31–29	28–27	26	25–24	23	22–21	20	19–16	15–2	1	0
010	00	0	10	1	10	0	0000	00000000000000	0	1
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

Note: AIX is a negative value in this example.

A.7 Array Synchronized 2-D to 1-D Transfers

The possible 2-D to 1-D transfers, along with the necessary parameters using array synchronization (FS=0), are shown in Figure A–69 through Figure A–77. For each, a single array of elements is transferred per synchronization event

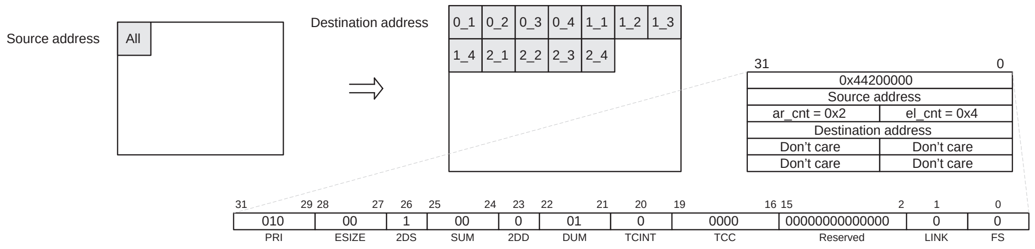
Figure A–69. Array Synchronized 2-D (SUM=00b) to 1-D (DUM=00b)



0x44000000	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
Don’t care	Don’t care
Don’t care	Don’t care

PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS
31–29	28–27	26	25–24	23	22–21	20	19–16	15–2	1	0
010	00	1	00	0	00	0	0000	00000000000000	0	0

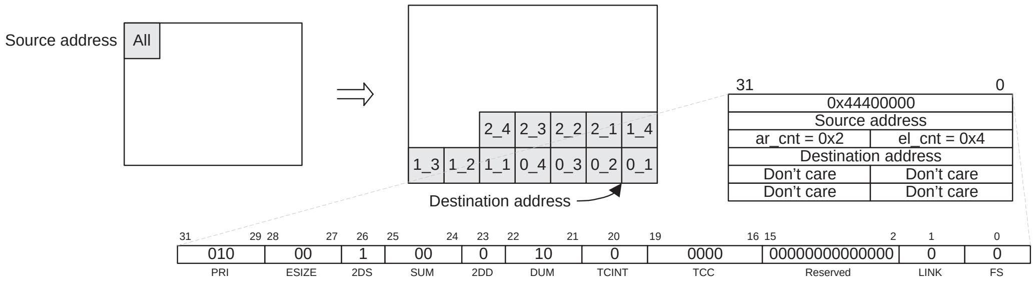
Figure A–70. Array Synchronized 2-D (SUM=00b) to 1-D (DUM=01b)



0x44200000	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
Don’t care	Don’t care
Don’t care	Don’t care

PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS
31–29	28–27	26	25–24	23	22–21	20	19–16	15–2	1	0
010	00	1	00	0	01	0	0000	00000000000000	0	0

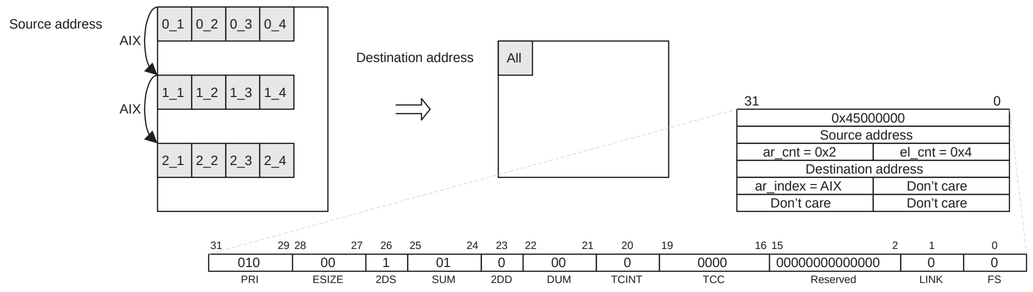
Figure A–71. Array Synchronized 2-D (SUM=00b) to 1-D (DUM=10b)



0x44400000	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
Don’t care	Don’t care
Don’t care	Don’t care

PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS
31–29	28–27	26	25–24	23	22–21	20	19–16	15–2	1	0
010	00	1	00	0	10	0	0000	00000000000000	0	0

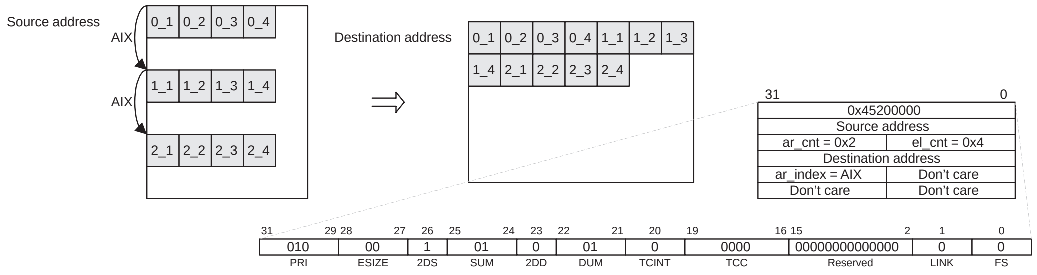
Figure A–72. Array Synchronized 2-D (SUM=01b) to 1-D (DUM=00b)



31	0
0x45000000	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
ar_index = AIX	Don't care
Don't care	Don't care

31–29	28–27	26	25–24	23	22–21	20	19–16	15–2	1	0
010	00	1	01	0	00	0	0000	00000000000000	0	0
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

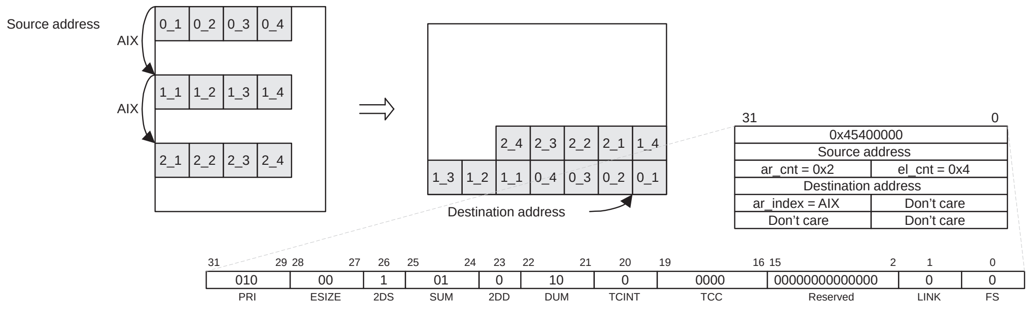
Figure A–73. Array Synchronized 2-D (SUM=01b) to 1-D (DUM=01b)



31	0
0x45200000	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
ar_index = AIX	Don't care
Don't care	Don't care

31–29	28–27	26	25–24	23	22–21	20	19–16	15–2	1	0
010	00	1	01	0	01	0	0000	00000000000000	0	0
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

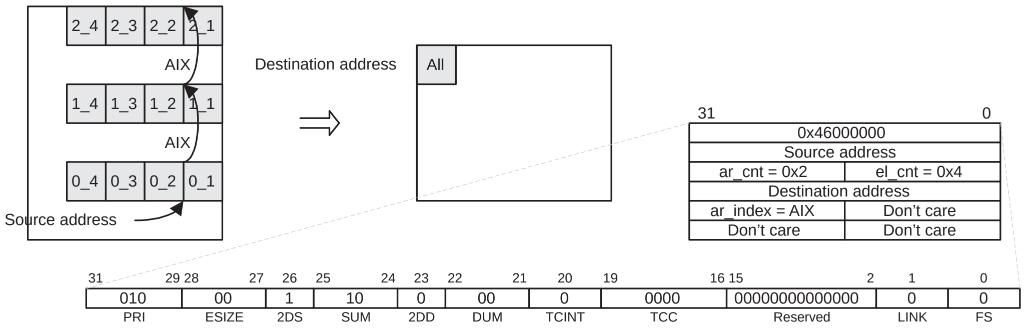
Figure A–74. Array Synchronized 2-D (SUM=01b) to 1-D (DUM=10b)



31	0
0x45400000	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
ar_index = AIX	Don't care
Don't care	Don't care

31–29	28–27	26	25–24	23	22–21	20	19–16	15–2	1	0
010	00	1	01	0	10	0	0000	00000000000000	0	0
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

Figure A–75. Array Synchronized 2-D (SUM=10b) to 1-D (DUM=00b)

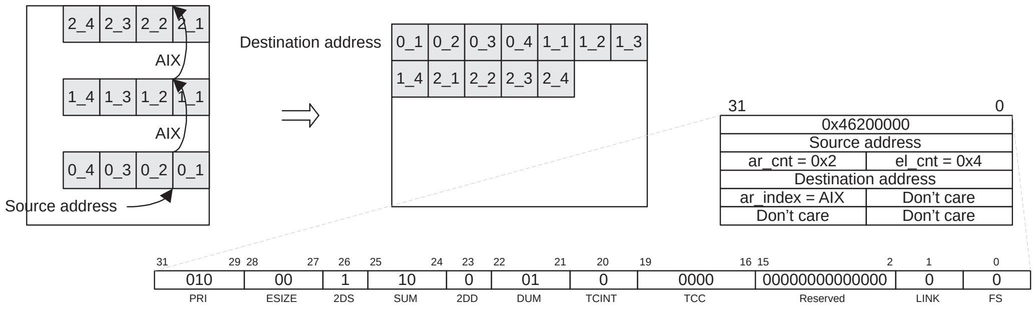


31	0
0x46000000	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
ar_index = AIX	Don't care
Don't care	Don't care

PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS
31–29	28–27	26	25–24	23	22–21	20	19–16	15–2	1	0
010	00	1	10	0	00	0	0000	00000000000000	0	0

Note: AIX is a negative value in this example.

Figure A–76. Array Synchronized 2-D (SUM=10b) to 1-D (DUM=01b)

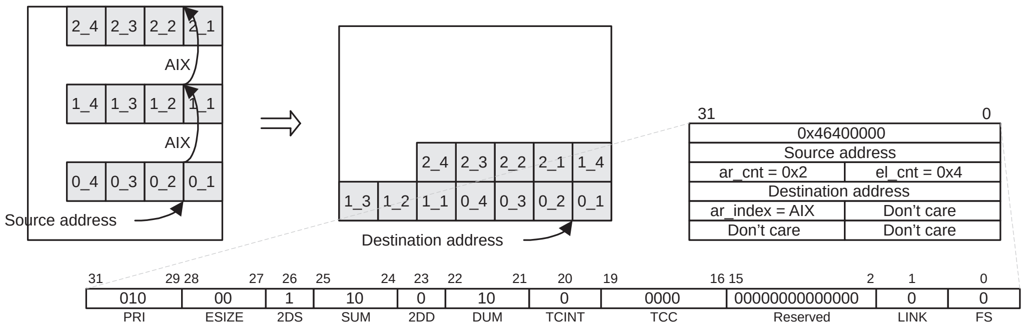


31	0
0x46200000	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
ar_index = AIX	Don't care
Don't care	Don't care

PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS
31–29	28–27	26	25–24	23	22–21	20	19–16	15–2	1	0
010	00	1	10	0	01	0	0000	00000000000000	0	0

Note: AIX is a negative value in this example.

Figure A–77. Array Synchronized 2-D (SUM=10b) to 1-D (DUM=10b)



31	0
0x46400000	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
ar_index = AIX	Don't care
Don't care	Don't care

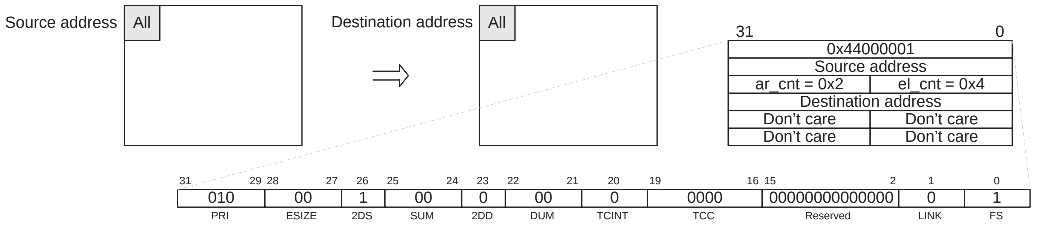
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS
31–29	28–27	26	25–24	23	22–21	20	19–16	15–2	1	0
010	00	1	10	0	10	0	0000	00000000000000	0	0

Note: AIX is a negative value in this example.

A.8 Block-Synchronized 2-D to 1-D Transfers

The possible 2-D to 1-D transfers, along with the necessary parameters using block synchronization (FS=1) are shown in Figure A–78 through Figure A–86. For each, an entire block of arrays is transferred per synchronization event.

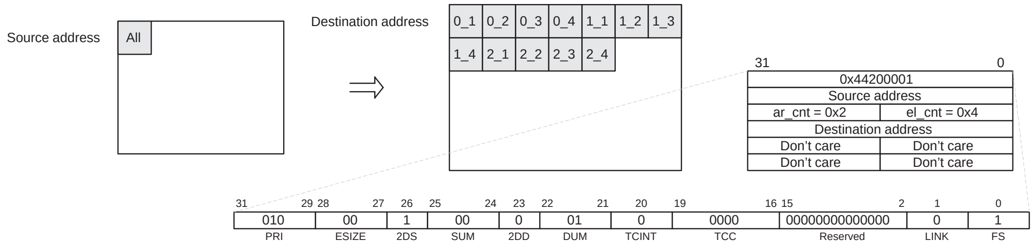
Figure A–78. Block-synchronized 2-D (SUM=00b) to 1-D (DUM=00b)



0x44000001	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
Don’t care	Don’t care
Don’t care	Don’t care

31 29	28 27	26	25 24	23	22 21	20	19 16	15 2	1	0
010	00	1	00	0	00	0	0000	00000000000000	0	1
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

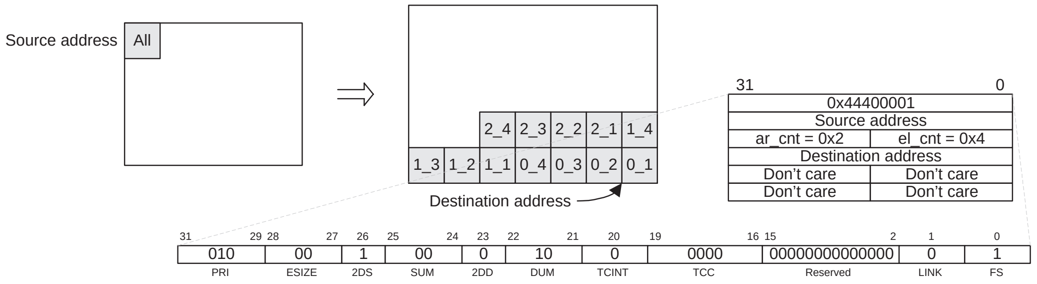
Figure A–79. Block-synchronized 2-D (SUM=00b) to 1-D (DUM=01b)



0x44200001	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
Don’t care	Don’t care
Don’t care	Don’t care

31 29	28 27	26	25 24	23	22 21	20	19 16	15 2	1	0
010	00	1	00	0	01	0	0000	00000000000000	0	1
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

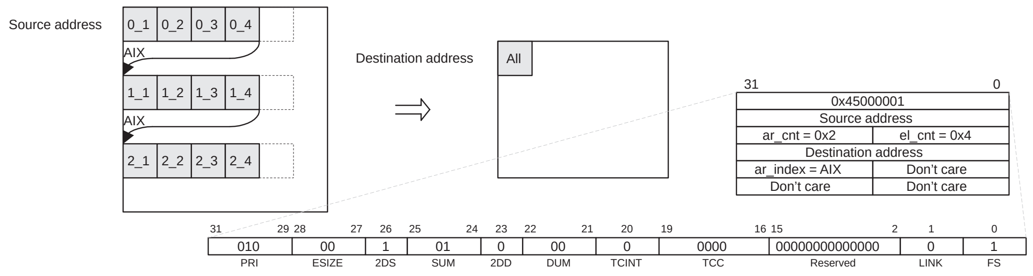
Figure A–80. Block-synchronized 2-D (SUM=00b) to 1-D (DUM=10b)



0x44400001	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
Don’t care	Don’t care
Don’t care	Don’t care

31 29	28 27	26	25 24	23	22 21	20	19 16	15 2	1	0
010	00	1	00	0	10	0	0000	00000000000000	0	1
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

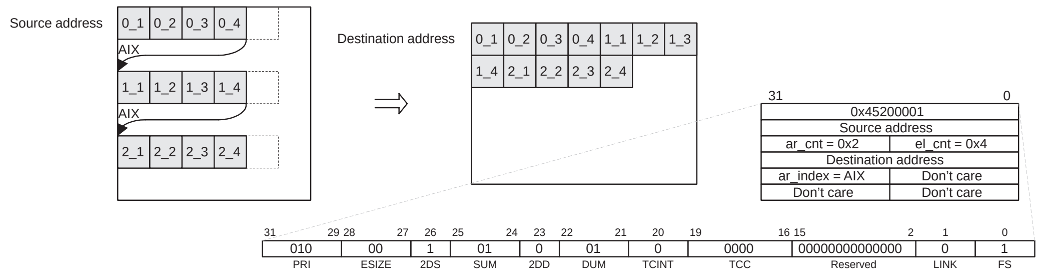
Figure A–81. Block-synchronized 2-D (SUM=01b) to 1-D (DUM=00b)



0x45000001	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
ar_index = AIX	Don't care
Don't care	Don't care

31–29	28–27	26	25–24	23	22–21	20	19–16	15–2	1	0
010	00	1	01	0	00	0	0000	00000000000000	0	1
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

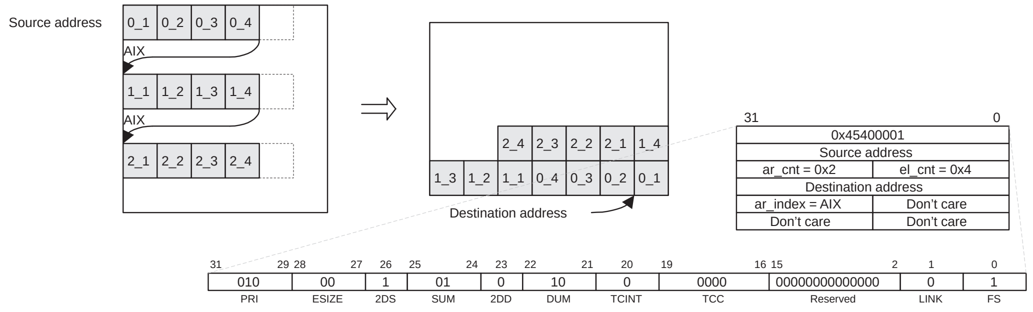
Figure A–82. Block-synchronized 2-D (SUM=01b) to 1-D (DUM=01b)



0x45200001	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
ar_index = AIX	Don't care
Don't care	Don't care

31–29	28–27	26	25–24	23	22–21	20	19–16	15–2	1	0
010	00	1	01	0	01	0	0000	00000000000000	0	1
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

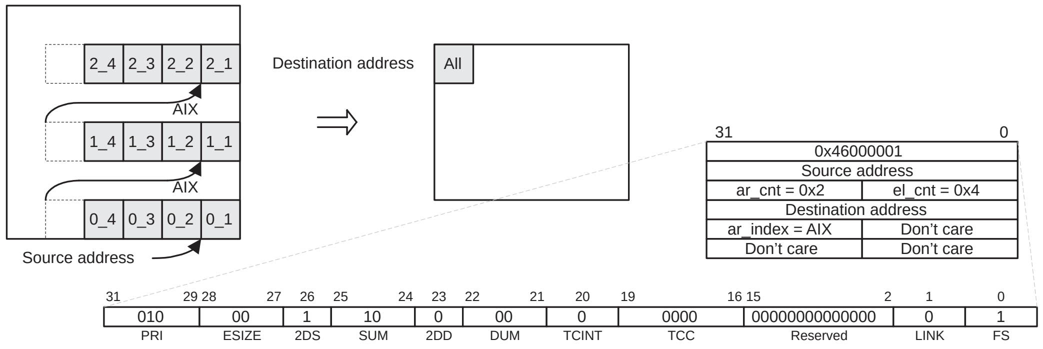
Figure A–83. Block-synchronized 2-D (SUM=01b) to 1-D (DUM=10b)



0x45400001	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
ar_index = AIX	Don't care
Don't care	Don't care

31–29	28–27	26	25–24	23	22–21	20	19–16	15–2	1	0
010	00	1	01	0	10	0	0000	00000000000000	0	1
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

Figure A–84. Block-synchronized 2-D (SUM=10b) to 1-D (DUM=00b)

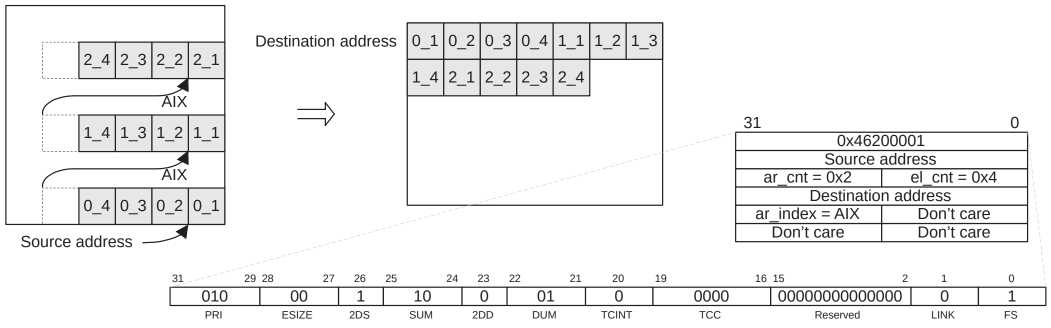


0x46000001	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
ar_index = AIX	Don’t care
Don’t care	Don’t care

31–29	28–27	26	25–24	23	22–21	20	19–16	15–2	1	0
010	00	1	10	0	00	0	0000	00000000000000	0	1
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

Note: AIX is a negative value in this example.

Figure A–85. Block-synchronized 2-D (SUM=10b) to 1-D (DUM=01b)

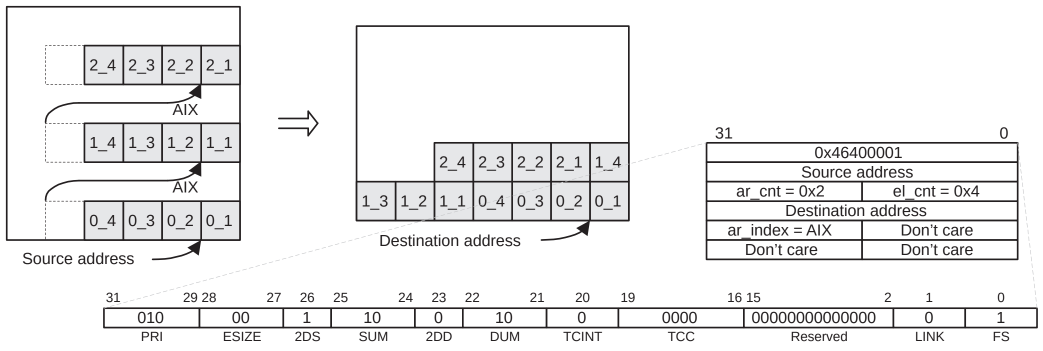


0x46200001	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
ar_index = AIX	Don’t care
Don’t care	Don’t care

31–29	28–27	26	25–24	23	22–21	20	19–16	15–2	1	0
010	00	1	10	0	01	0	0000	00000000000000	0	1
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

Note: AIX is a negative value in this example.

Figure A–86. Block-synchronized 2-D (SUM=10b) to 1-D (DUM=10b)



0x46400001	
Source address	
ar_cnt = 0x2	el_cnt = 0x4
Destination address	
ar_index = AIX	Don’t care
Don’t care	Don’t care

31–29	28–27	26	25–24	23	22–21	20	19–16	15–2	1	0
010	00	1	10	0	10	0	0000	00000000000000	0	1
PRI	ESIZE	2DS	SUM	2DD	DUM	TCINT	TCC	Reserved	LINK	FS

Note: AIX is a negative value in this example.

Index