TMS320C620x/C670x DSP Program and Data Memory Controller/ Direct Memory Access (DMA) Controller Reference Guide

Literature Number: SPRU577A
September 2004

Texas Instruments

IMPORTANT NOTICE

Texas Instruments Incorporated and its subsidiaries (TI) reserve the right to make corrections, modifications, enhancements, improvements, and other changes to its products and services at any time and to discontinue any product or service without notice. Customers should obtain the latest relevant information before placing orders and should verify that such information is current and complete. All products are sold subject to TI's terms and conditions of sale supplied at the time of order acknowledgment.

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

TI assumes no liability for applications assistance or customer product design. Customers are responsible for their products and applications using TI components. To minimize the risks associated with customer products and applications, customers should provide adequate design and operating safeguards.

TI does not warrant or represent that any license, either express or implied, is granted under any TI patent right, copyright, mask work right, or other TI intellectual property right relating to any combination, machine, or process in which TI products or services are used. Information published by TI regarding third-party products or services does not constitute a license from TI to use such products or services or a warranty or endorsement thereof. Use of such information may require a license from a third party under the patents or other intellectual property of the third party, or a license from TI under the patents or other intellectual property of TI.

Reproduction of information in TI data books or data sheets is permissible only if reproduction is without alteration and is accompanied by all associated warranties, conditions, limitations, and notices. Reproduction of this information with alteration is an unfair and deceptive business practice. TI is not responsible or liable for such altered documentation.

Resale of TI products or services with statements different from or beyond the parameters stated by TI for that product or service voids all express and any implied warranties for the associated TI product or service and is an unfair and deceptive business practice. TI is not responsible or liable for any such statements.

Following are URLs where you can obtain information on other Texas Instruments products and application solutions:

Products		**Applications**	
Amplifiers	amplifier.ti.com	Audio	www.ti.com/audio
Data Converters	dataconverter.ti.com	Automotive	www.ti.com/automotive
DSP	dsp.ti.com	Broadband	www.ti.com/broadband
Interface	interface.ti.com	Digital Control	www.ti.com/digitalcontrol
Logic	logic.ti.com	Military	www.ti.com/military
Power Mgmt	power.ti.com	Optical Networking	www.ti.com/opticalnetwork
Microcontrollers	microcontroller.ti.com	Security	www.ti.com/security
		Telephony	www.ti.com/telephony
		Video & Imaging	www.ti.com/video
		Wireless	www.ti.com/wireless

Mailing Address: Texas Instruments
Post Office Box 655303 Dallas, Texas 75265

Preface

Read This First

About This Manual

This document describes the program memory modes, program and data memory organizations, and the program and data memory controller in the TMS320C620x/C670x digital signal processors (DSPs) of the TMS320C6000™ DSP family.

This document also describes the operation of the direct memory access (DMA) controller and the DMA and CPU data access performance to the internal memory, the peripherals, and the external memory.

Notational Conventions

This document uses the following conventions.

- Hexadecimal numbers are shown with the suffix h. For example, the following number is 40 hexadecimal (decimal 64): 40h.
- Registers in this document are shown in figures and described in tables.
  - Each register figure shows a rectangle divided into fields that represent the fields of the register. Each field is labeled with its bit name, its beginning and ending bit numbers above, and its read/write properties below. A legend explains the notation used for the properties.
  - Reserved bits in a register figure designate a bit that is used for future device expansion.

Related Documentation From Texas Instruments

The following documents describe the C6000™ devices and related support tools. Copies of these documents are available on the Internet at www.ti.com. *Tip:* Enter the literature number in the search box provided at www.ti.com.

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

TMS320C6000 DSP Peripherals Overview Reference Guide (literature number SPRU190) describes the peripherals available on the TMS320C6000™ DSPs.

TMS320C6000 Technical Brief (literature number SPRU197) gives an introduction to the TMS320C62x™ and TMS320C67x™ DSPs, development tools, and third-party support.

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

TMS320C6000 Code Composer Studio Tutorial (literature number SPRU301) introduces the Code Composer Studio™ integrated development environment and software tools.

Code Composer Studio Application Programming Interface Reference Guide (literature number SPRU321) describes the Code Composer Studio™ application programming interface (API), which allows you to program custom plug-ins for Code Composer.

TMS320C6x Peripheral Support Library Programmer's Reference (literature number SPRU273) describes the contents of the TMS320C6000™ 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 Chip Support Library API Reference Guide (literature number SPRU401) describes a set of application programming interfaces (APIs) used to configure and control the on-chip peripherals.

Trademarks

Contents

Figures

Tables

Chapter 1

TMS320C620x/C670x Internal Memory

This chapter describes the program memory modes, program and data memory organizations, and the program and data memory controller in the TMS320C620x/C670x digital signal processors (DSPs) of the TMS320C6000™ DSP family.

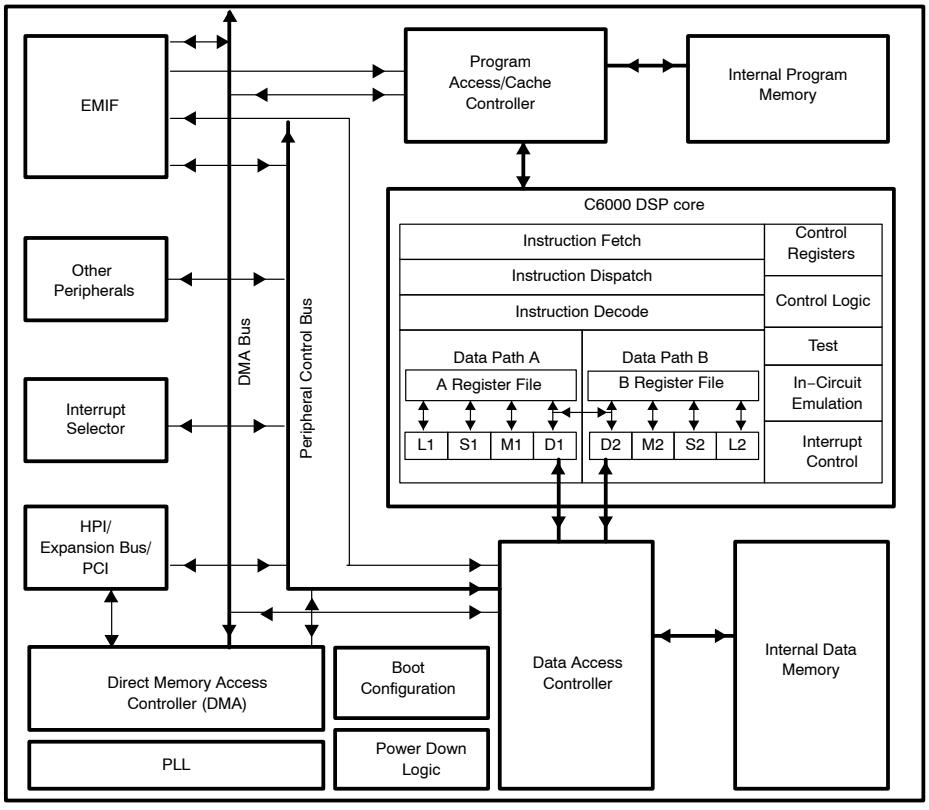
Topic **Page**

1.1 Program Memory Controller

The program memory controller, shown in Figure 1–1, 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 1–1. TMS320C620x/C670x DSP Block Diagram



1.2 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) DSP, the memory/cache program space is expanded to the sizes shown in Table 1–1. In addition, the C6202(B)/C6203(B) DSP 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 1–1 and Table 1–2 compare the internal memory and cache configurations available on the current C620x/C670x devices. Figure 1–2 shows a block diagram of the connections between the C6201/C6204/C6205/C6701 CPU, program memory controller (PMEMC), and memory blocks. Figure 1–3 shows a block diagram of the connections between the CPU, PMEMC, and memory blocks in the C6202/C6202B/C6203(B) DSP. For the C6202(B)/C6203(B) DSP, there are two program memory controllers, PMEM1 and PMEM0. The PMEM1 controller handles all accesses to program memory block 1 (SRAM and cache), as well as all cache operations and external accesses. The PMEM0 controller always accesses program memory block 0 (SRAM only). The addresses shown in Figure 1–2 and Figure 1–3 are for operation in memory map mode 1.

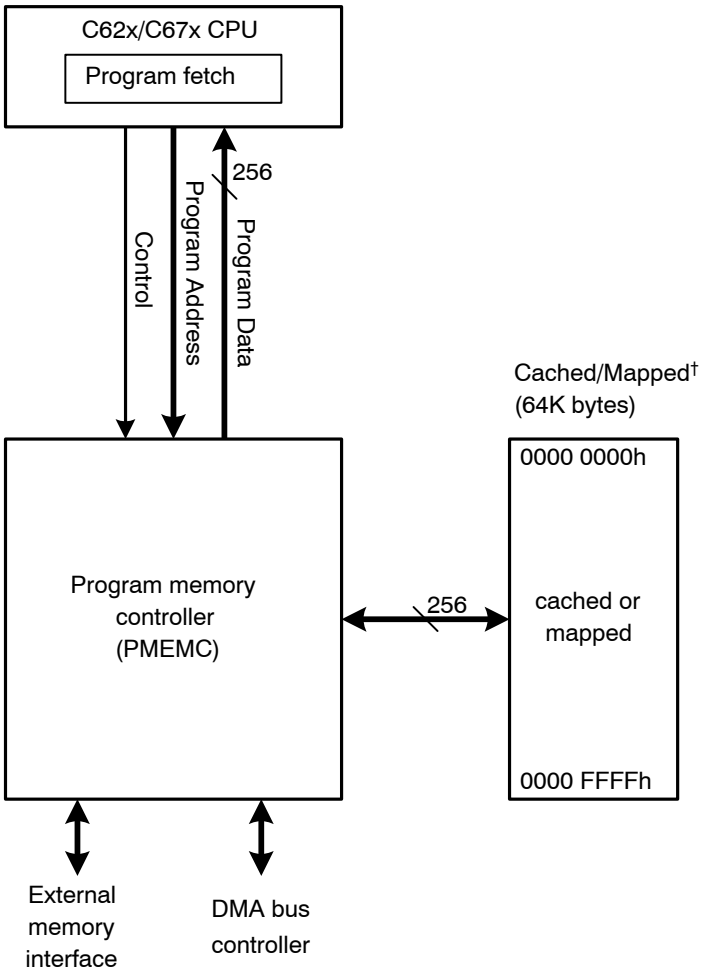
Table 1–1. TMS320C620x/C670x DSP Internal Memory Configurations

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 1–2. TMS320C620x/C670x DSP Cache Architectures

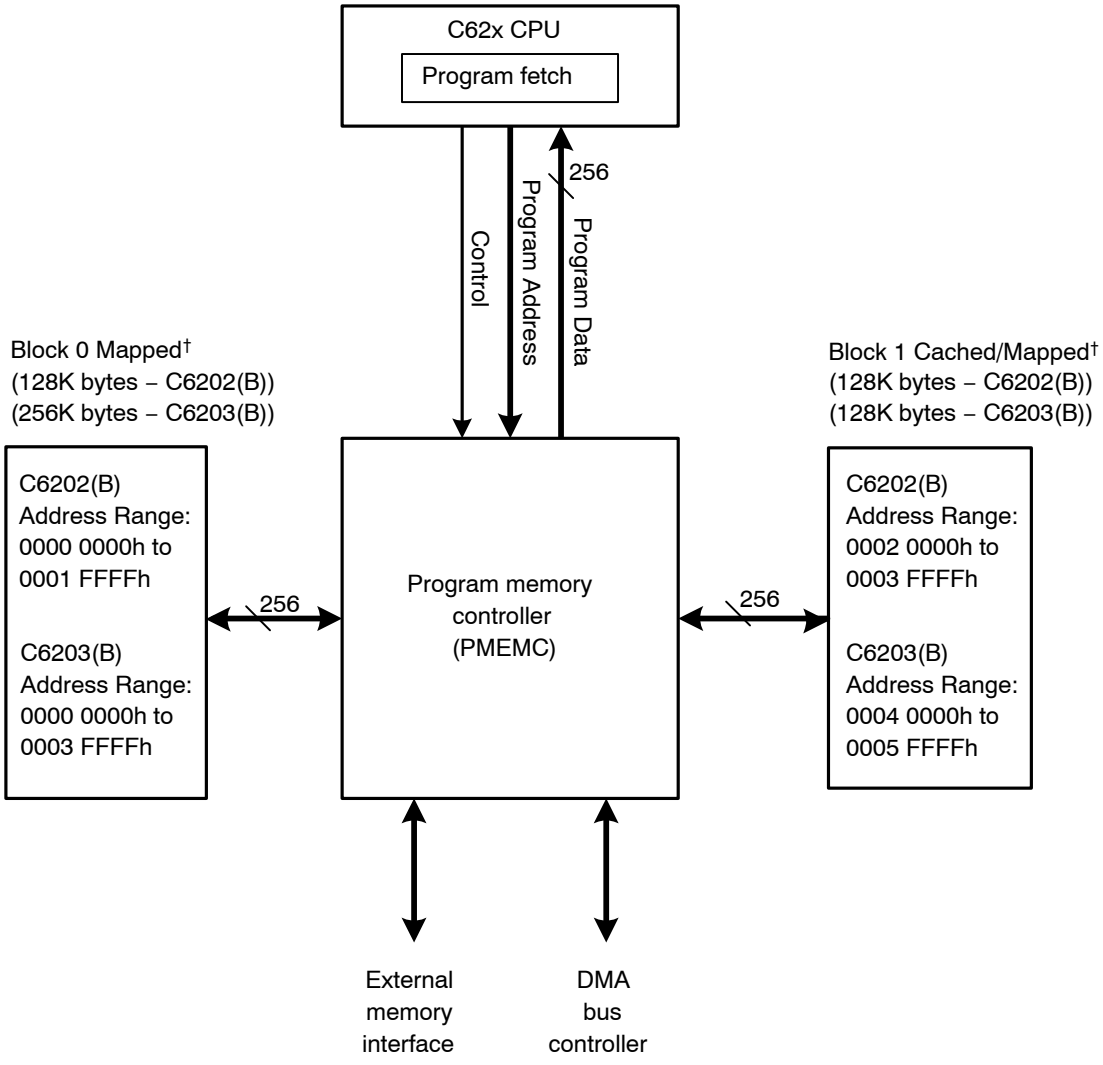
Cache Space	Size (Bytes)	Associativity	Line Size (Bytes)
C6201 program	64K	Direct mapped	32
C6701 program	64K	Direct mapped	32
C6202(B) program	128K	Direct mapped	32
C6203(B) program	128K	Direct mapped	32
C6204 program	64K	Direct mapped	32
C6205 program	64K	Direct mapped	32

Figure 1–2. Program Memory Controller Block Diagram (C6201/C6204/C6205/C6701 DSP)



† Addresses shown in Map 1 mode (section 1.2.2)

Figure 1–3. Program Memory Controller Block Diagram (C6202(B)/C6203(B) DSP)



† Addresses shown in Map 1 mode (section 1.2.2)

1.2.1 Internal Program Memory Modes

The cached/mapped block of the internal program memory can be used in any of four modes as selected by the program cache control (PCC) field in the CPU control status register (CSR), as shown in Table 1–3. 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 1.2.2 for a detailed description of the memory mapped operations.)

- **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 1–3. Internal Program Memory Mode Summary

PCC Field	Memory Mode	Description
000	Mapped	Cache disabled (default state at reset)
001	–	Reserved
010	Cache enabled	Cache accessed and updated on reads
011	Cache freeze	Cache accessed but not updated on reads
100	Cache bypass	Cache not accessed or updated on reads
101–111	–	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 4 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. You 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
```

1.2.2 Memory Mapped Operation

When the PCC field in CSR is programmed for mapped mode, all of the internal program RAM is mapped into internal program space. Table 1–4 shows the address space for the internal program RAM for the map mode selected at device reset.

Table 1–4. Internal Program RAM Address Mapping in Memory Mapped Mode

Device	Block†	Map 0	Map 1
C6201	–––	0140 0000h – 0140 FFFFh	0000 0000h – 0000 FFFFh
C6202(B)	Block 0	0140 0000h – 0141 FFFFh	0000 0000h – 0001 FFFFh
	Block 1	0142 0000h – 0143 FFFFh	0002 0000h – 0003 FFFFh
C6203(B)	Block 0	0140 0000h – 0143 FFFFh	0000 0000h – 0003 FFFFh
	Block 1	0144 0000h – 0145 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

† C6201/C6204/C6205/C6701 DSP 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) DSP, 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.

1.2.3 Cache Operation

When the PCC field in CSR 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 1–5 shows the addresses occupied by the C6202/C6202B/C6203(B) RAM that is not used for cache, for each map mode.

Table 1–5. Internal Program RAM Address Mapping in Cache Mode (TMS320C6202(B)/C6203(B) DSP)

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) DSP, 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.

1.2.4 Cache Architecture

The C620x/C670x cache is a direct mapped architecture. The width of the cache (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 DSP, the 64K byte cache contains 2K fetch packets (2K lines). For the C6202(B)/C6203(B) DSP, the 128K-byte cache contains 4K fetch packets (4K lines).

1.2.4.1 Cache Usage of CPU Address

How the cache uses the fetch packet address from the CPU is shown in Figure 1−4 for the C6201/C6204/C6205/C6701 DSP and in Figure 1−5 for the C6202(B)/C6203(B) DSP.

Figure 1−4. Logical Mapping of Cache Address (C6201/C6204/C6205/C6701 DSP)

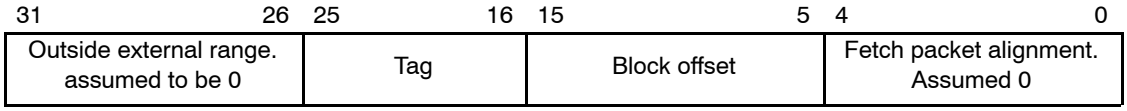
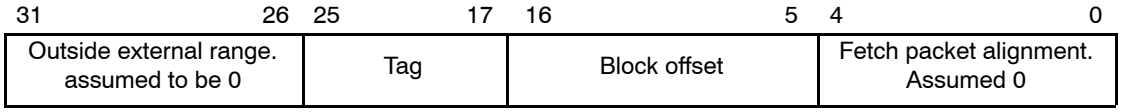


Figure 1−5. Logical Mapping of Cache Address (C6202(B)/C6203(B) DSP)



- **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 DSP, 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) DSP, 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 DSP, bits 25−16. The cache has a separate 2K × 11 tag RAM that holds all the tags.
  - For C6202(B)/C6203(B) DSP, bits 25−17. The cache has a separate 4K × 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.

1.2.4.2 Cache Invalidation

The tag RAM contains a valid bit for each line of the cache. When a program enables the cache, the PMEMC clears the valid bit for each tag entry. This invalidates the entire contents of the program cache and prepares it for use.

Programs can change the current cache mode by writing to the PCC field in the CSR. The PMEMC only invalidates its contents when it transitions out of the mapped-memory mode. In other words, the PMEMC invalidates the cache's contents when programs write 010b to PCC and the PCC previous value was 000b.

The PMEMC halts the CPU while it initializes its tags. On the 6202(B) and C6203(B) DSPs, the PMEMC allows DMA accesses to proceed to Block 0 during this initialization.

1.2.4.3 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.

1.2.5 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.

1.2.6 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 word 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. While the CPU is executing from external memory, IPRAM block 1 cannot be accessed using the DMA. The PMEM1 memory controller is used by the CPU to fetch instructions from the EMIF; therefore, while performing a fetch from external memory, DMA access to PMEM1 is limited.

1.2.7 Illegal Access to Program Memory

An access to a section of memory that does not return a ready indication is not allowed. Possible requestors are: CPU program fetches, CPU loads and stores, programmed DMA channels or HPI/PCI/XBUS host mastering of the DMA through the auxiliary DMA. This type of access can create a stall indefinitely. When a requestor has created a program memory stall, other requestors are unable to access this program memory space. For C6202/C6203 DSP, if an access generates a program memory block 0 stall, other requestors may still access program memory block 1 and vice versa.

1.3 Data Memory Controller

As shown in Figure 1−1 (page 1-2), the data memory controller connects:

- The CPU and direct memory access (DMA) controller to internal data memory and performs the necessary arbitration.
- The 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.

1.3.1 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 described in section 1.4 and illustrated in Figure 1−6, Figure 1−7, Figure 1−8, and Figure 1−9. 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 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 3, *DMA and CPU Data Access Performance*, for a description of data access performance to the internal data memory, the EMIF, and the peripheral bus.

1.4 Internal Data Memory

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

1.4.1 TMS320C6201/C6204/C6205 DSP

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 1–6, Figure 1–6, and Table 1–7. 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 1–6. Data Memory Organization (C6201/C6204/C6205 DSP)

	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 1–6. Data Memory Controller Interconnect to Other Banks (C6201/C6204/C6205 DSP)

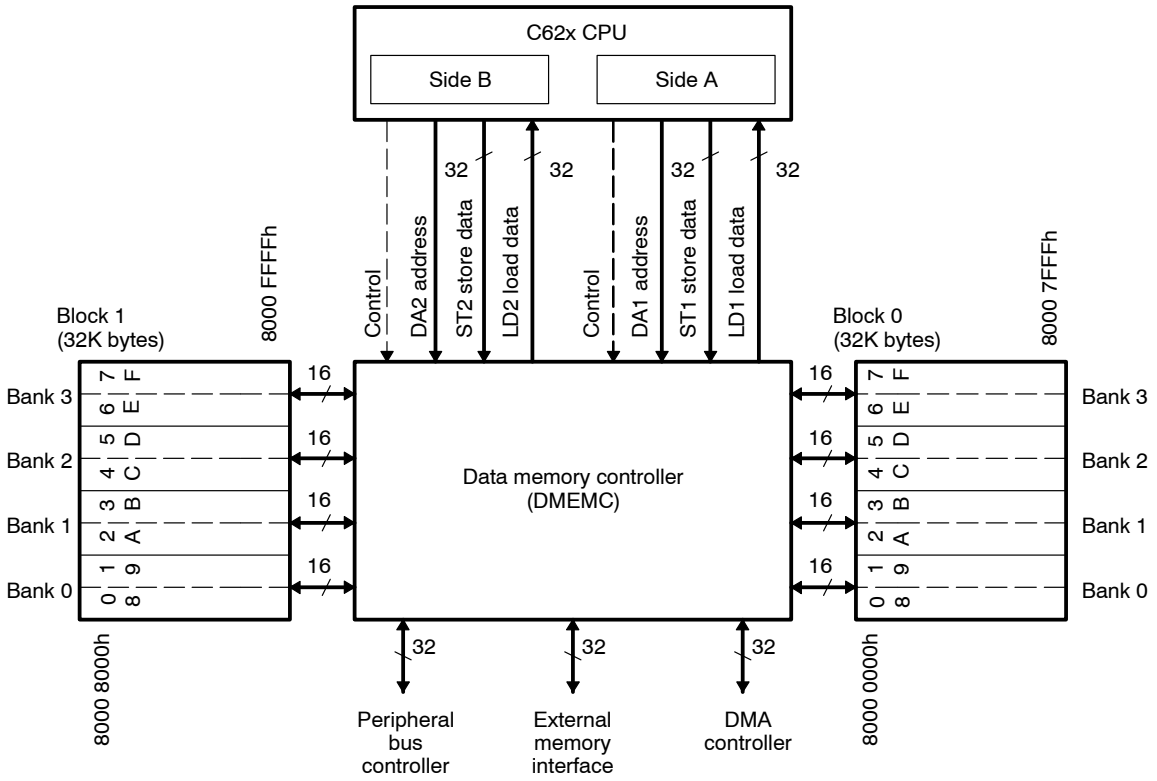


Table 1–7. Internal Data RAM Address Mapping (C6201/C6204/C6205 DSP)

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

1.4.2 TMS320C6701 DSP

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 1–8, Figure 1–7, and Table 1–9. 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 1–8. Data Memory Organization (C6701 DSP)

	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 1–7. Data Memory Controller Interconnect to Other Blocks (C67x CPU)

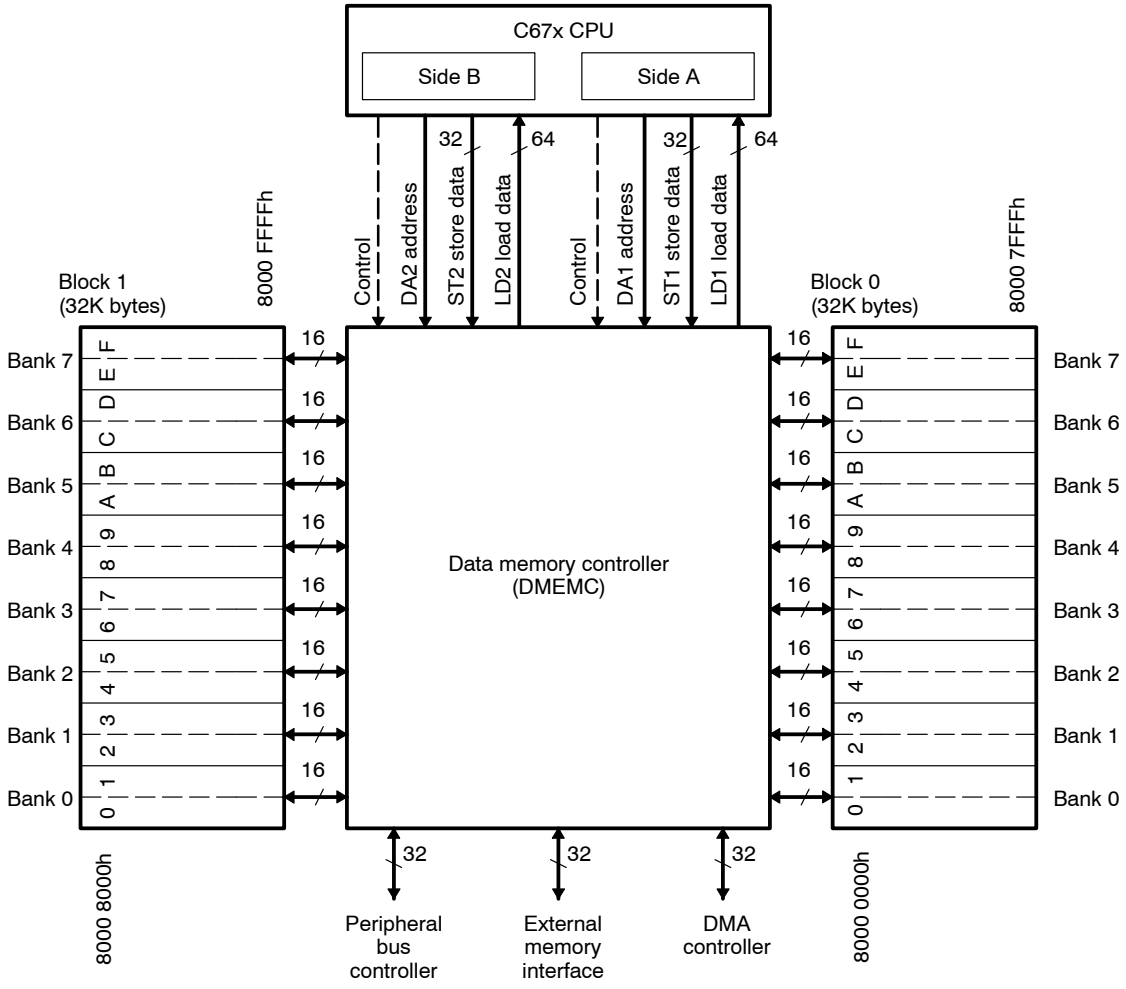


Table 1–9. Internal Data RAM Address Mapping (C6701 DSP)

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

1.4.3 TMS320C6202(B) DSP

The C6202(B) data memory controller (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 1–8 shows a block diagram of the connections between the C6202(B) CPU, DMEMC, and memory blocks. Table 1–10 shows the memory range occupied by each block of internal data RAM.

Figure 1–8. Data Memory Controller Interconnect to Other Blocks (C6202(B) DSP)

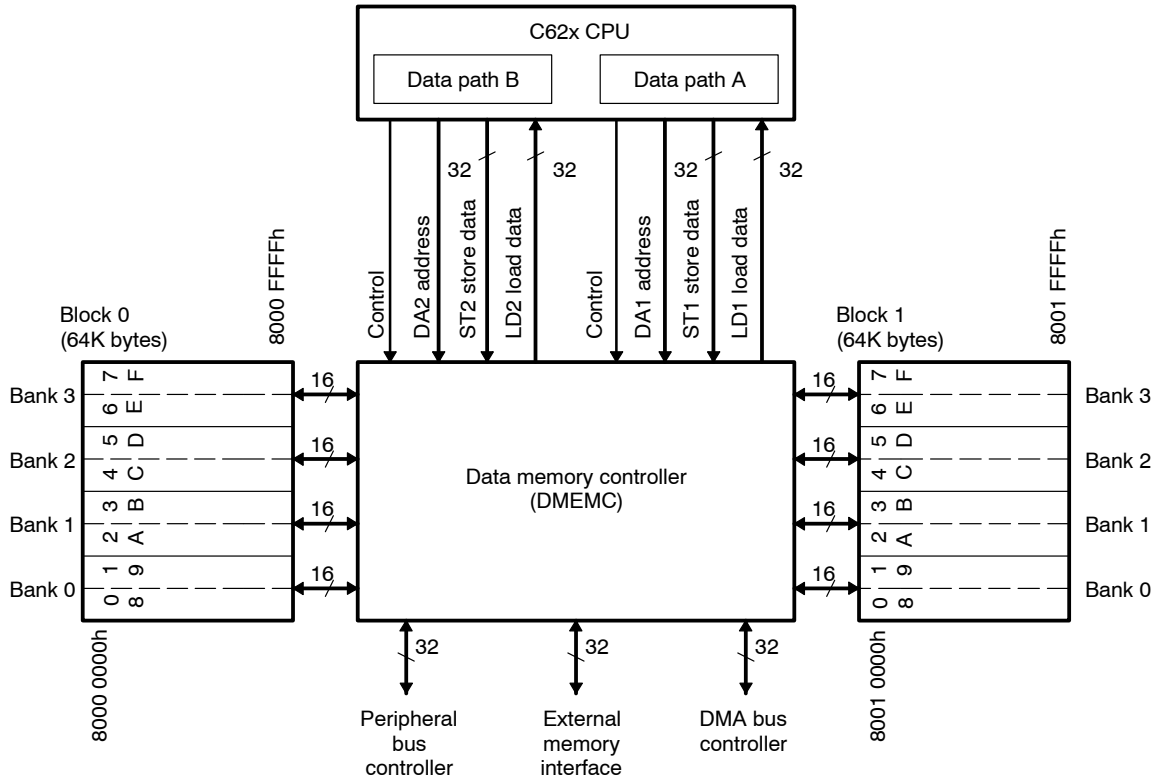


Table 1–10. Internal Data RAM Address Mapping (C6202(B) DSP)

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

1.4.4 TMS320C6203(B) DSP

The C6203(B) data memory controller (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 1–9 shows a block diagram of the connections between the C6203(B) CPU, DMEMC, and memory blocks. Table 1–11 shows the memory range occupied by each block of internal data RAM.

Figure 1–9. Data Memory Controller Interconnect to Other Blocks (C6203(B) DSP)

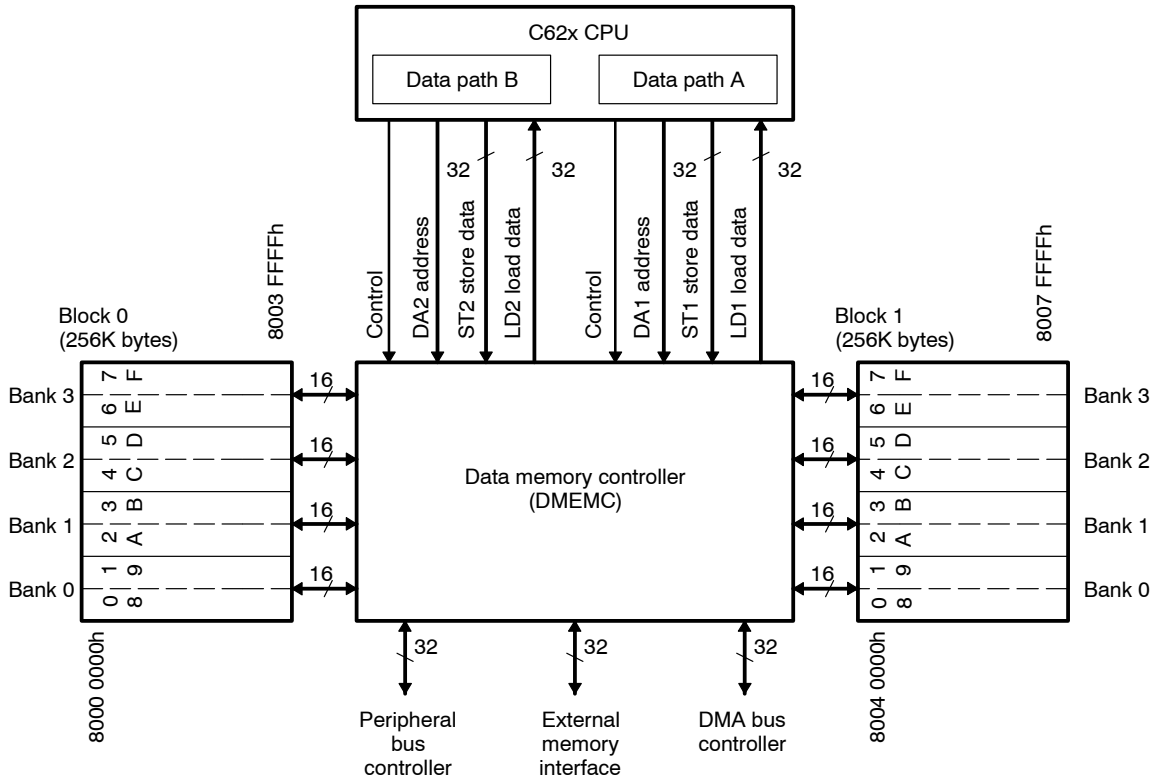


Table 1–11. Internal Data RAM Address Mapping (C6203(B) DSP)

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

1.4.5 Data Alignment

The following data alignment restrictions apply:

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

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

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

Bytes: There are no alignment restrictions on byte accesses.

1.4.6 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.

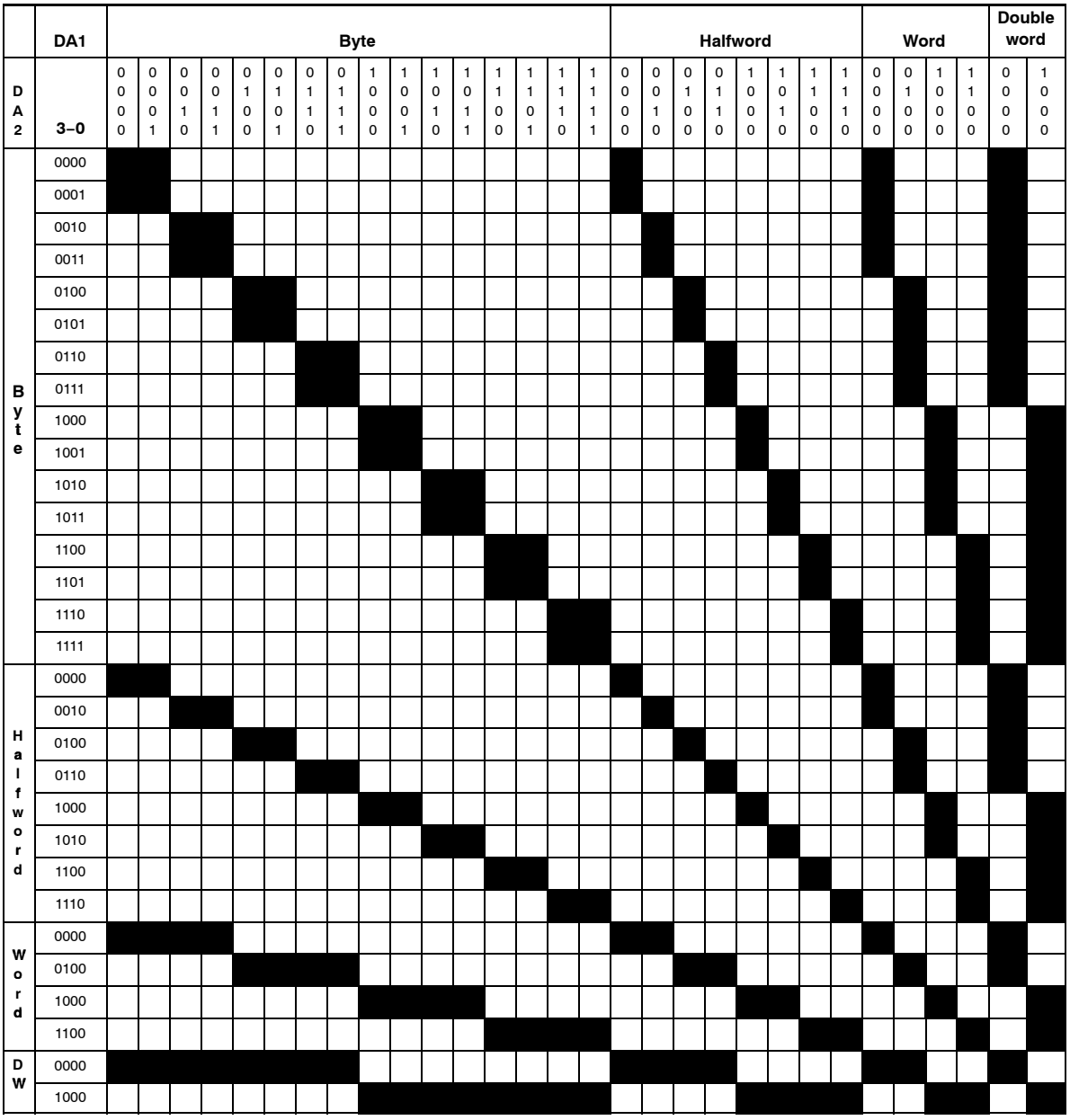
Loads and stores from the same execute packet are seen by the data memory controller during the same CPU cycle. Loads and stores from future or previous CPU cycles do not cause wait states for the internal data memory accesses in the current cycle. Thus, internal data memory access causes a wait state 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 1–10 and Figure 1–11 show what access conditions cause internal memory conflicts when the CPU makes two data accesses (on DA1 and DA2).

Figure 1–10. Conflicting Internal Memory Accesses to the Same Block (C6201/C6202(B)/C6203(B)/C6204/C6205 DSP)

	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 are shown in shaded areas.

Figure 1−11. Conflicting Internal Memory Accesses to the Same Block (C6701 DSP)



Note: Conflicts are shown in shaded areas.

1.4.7 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 1−10 and Figure 1−11 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. 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.

1.4.8 Illegal Access to Data Memory

An access to a section of memory that does not return a ready indication is not allowed. Possible requestors are: CPU program fetches, CPU loads and stores, programmed DMA channels or HPI/PCI/XBUS host mastering of the DMA through the auxiliary DMA. This type of access can create a stall indefinitely. When a requestor has created a data memory stall, other requestors are unable to access this data memory space.

1.4.9 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 halfword data resident in memory is identical for little-endian and big-endian data. Table 1–12 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 1–13 and Table 1–14 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 1–12. Register Contents After Little-Endian or Big-Endian Data Loads (C620x/C670x DSP)

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 1–13. Register Contents After Little-Endian or Big-Endian Data Loads (C6701 DSP only)

Instruction	Address Bits (2–0)	Big-Endian Memory Result	Little-Endian Memory Result
LDDW	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 1–14. Memory Contents After Little-Endian or Big-Endian Data Stores (C620x/C670x DSP)

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.

1.5 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.

1.5.1 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 1–15. 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 1–15. 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	XXXX XXXX	XXXX XXXX
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.

1.5.2 CPU Wait States

Isolated peripheral bus controller accesses from the CPU cause 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.

1.5.3 Arbitration Between the CPU and the DMA Controller

As shown in Figure 1–6, Figure 1–7, Figure 1–8, and Figure 1–9, 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 2

Direct Memory Access (DMA) Controller

This chapter describes the operation of the direct memory access (DMA) controller in the digital signal processors (DSPs) of the TMS320C6000™ DSP family.

Topic **Page**

2.1 Overview

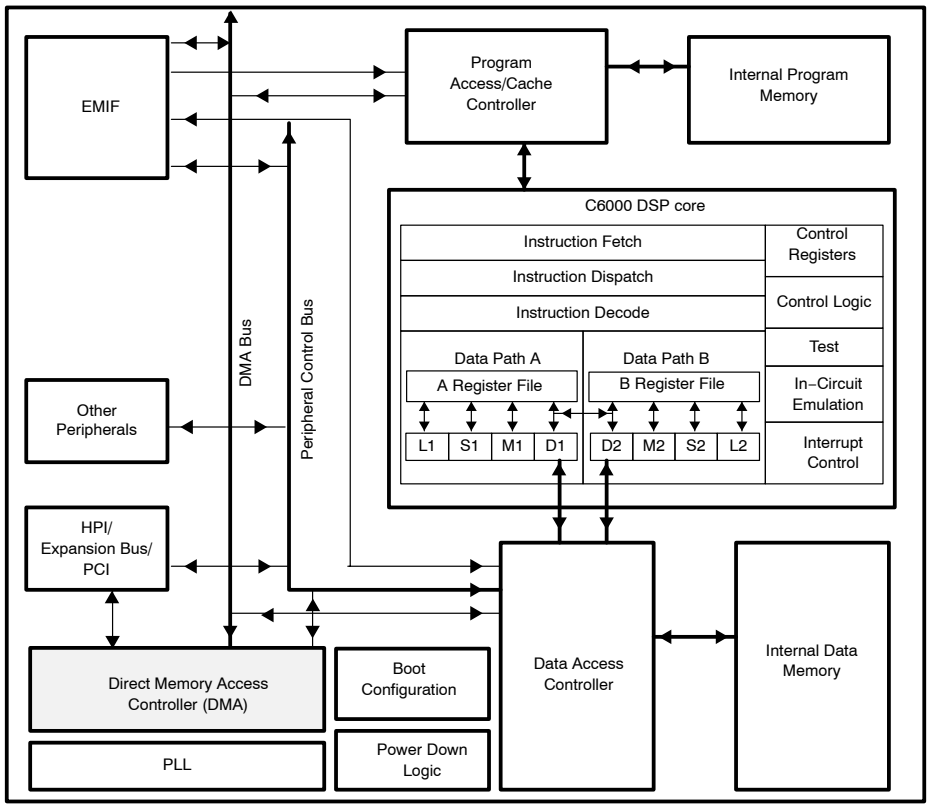
The DMA controller transfers data between regions in the memory map without intervention by the CPU. The DMA controller allows movement of data to and from internal memory, internal peripherals, or external devices to occur in the background of CPU operation. The DMA controller has four independent programmable 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). Requests are sent to one of these possible resources:

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

Figure 2−1 shows the C6000 DSP block diagram with the DMA.

Figure 2–1. DMA Controller Interconnect to TMS320C6000 Memory-Mapped Modules



2.2 DMA Terminology

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

- **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-channel mode, data elements are read from the source address, and written to the split destination address.
- **Receive element transfer:** In split-channel mode, data elements are read from the split source address, and written to the destination address.

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.
- **Four channels:** The DMA controller can keep track of the contexts of four independent 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.
- **Multiframe transfer**: Each block transfer can consist of multiple frames of a programmable size.

- **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.
- **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.
- **Autoinitialization:** Once a block transfer is complete, a DMA channel can automatically reinitialize itself for the next block transfer.
- **Event synchronization:** Each read, write, or frame transfer may be initiated by selected events.
- **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.

2.3 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 primary control register (PRICTL).

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

2.3.1 Register Access Protocol

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

1) Set up DMA channel primary control register (PRICTL) with START = 00b.
2) Set up DMA channel secondary control register (SECCTL) with:

   WSYNCCLR = 1

   RSYNCCLR = 1
3) Set up DMA channel source address register (SRC), DMA channel destination address register (DST), and DMA global count reload register (GBLCNT).
4) Start the DMA channel by writing a 01b or 11b to the START field of PRICTL.

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

2.3.2 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 PRICTL, 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 nonautoinitialization: It is possible to switch from an autoinitialized transfer to a nonautoinitialized transfer to complete DMA activity on a particular channel. To switch modes, the active channel should be paused by restoring PRICTL with START = 10b, then restarted in the new mode by restoring PRICTL with START = 01b.

If the active channel is operating in split-channel mode, then it is necessary to ensure that the switch from autoinitialization to nonautoinitialization 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.

2.3.3 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 register (XFRCNT) and the DMA channel source and destination address registers (SRC and DST) have associated reload registers, as selected by the associated reload fields, DSTRLD and SRCRLD, in PRICTL.

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 global 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 XFRCNT, the 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 2.9.1.2, *Shared FIFO*, 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 is 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 mode operation described in section 2.6, 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 2.12.6 for more explanation of the DMA channel transfer counter.

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.

2.4 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.

Up to 31 events are available and may be selected by the RSYNC and WSYNC fields in the DMA channel primary control register (PRICTL). If the frame synchronization bit is set (FS = 1) in PRICTL, then the event selected by RSYNC field enables an entire frame, and WSNYC field must be cleared to 00000b. If a channel is set up to operate in split-channel mode (SPLIT ≠ 00b), RSYNC and WSYNC must be set to nonzero values. If the value of these fields is cleared 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 the RSYNC and WSYNC fields and events is shown in Table 2–1.

Table 2−1. 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
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
10011−11111	Reserved	

2.4.1 Latching of DMA Channel Event Flags

The DMA channel secondary control register (SECCTL) contains status and clear fields for read (RSYNC) and write (WSYNC) synchronization events. Care must be taken if software is used to poll and clear the status/conditions in SECCTL during a synchronized DMA transfer. To avoid inadvertently setting an extra RSYNC/WSYNC event during a synchronized DMA transfer, you should only write zeros to the STAT and CLR fields.

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 SECCTL. If no synchronization is selected, the STAT bit is always read as 1. A single event can trigger multiple actions.

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 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. Manipulation of events has priority over any simultaneous automated setting or clearing of events.

2.4.2 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 RSYNCSTAT field when the DMA completes the request for the first read transfer in the new frame.

2.4.3 Synchronization Control

The DMA of the C6202/C6203/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 in SECCTL. WSPOL affects write-synchronized transfers and RSPOL affects read- and frame-synchronized transfers.

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

Figure 2−2 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_INT*n* 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_INT*n* are ignored.

3) Same as 1

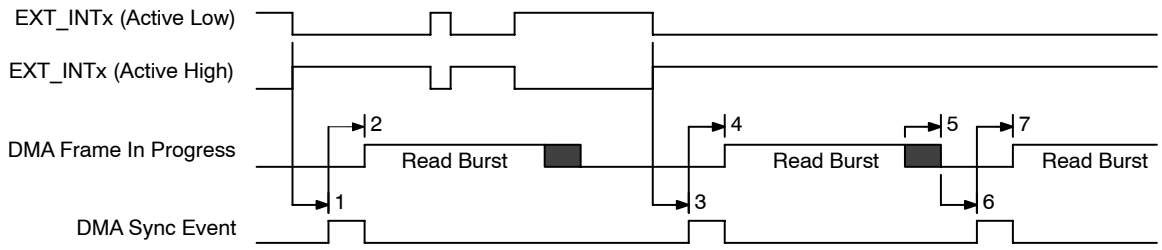
4) Same as 2

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

6) Because EXT_INT*n* 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 2–2. 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 ($\overline{HF}$ active) to less than half-full ($\overline{HF}$ 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 $\times$8 mode).

These new features are only used by the DMA when WSPOL, RSPOL, or FSIG are properly configured. If all fields are cleared to 0 (default), the C6202/C6203/C6204/C6205 DMA functions identically to the C6201 DMA.

2.5 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.

2.5.1 Basic Address Adjustment

The SRCDIR and DSTDIR fields in the DMA channel primary control register (PRICTL) can do the following:

- set the index to increment by element size
- set the index to decrement by element size
- use a global index value
- not affect either the DMA channel source or destination address registers

By default, the SRCDIR and DSTDIR 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.

2.5.2 Address Adjustment with the Global Index Registers

The DMA global index register (GBLIDX) is selected by the INDEX field in the DMA channel primary control register (PRICTL). 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 (ELEIDX) is contained in the 16 LSBs of the selected GBLIDX. The adjustment value for the end of the frame (FRMIDX) is contained in the 16 MSBs of the selected GBLIDX. Both of these fields contain signed 16-bit values that can range from –32768 to 32767.

2.5.3 Element Size, Alignment, and Endianness

By using the ESIZE field in the DMA channel primary control register (PRICTL), 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 address register (SRC), DMA channel destination address register (DST), and any associated reload registers.
- ELEIDX and FRMIDX in the DMA global index register (GBLIDX)

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.

2.5.4 Using a Frame Index to Reload Addresses

In an autoinitialized, single-frame block transfer, the frame index (FRMIDX) in the DMA global index register (GBLIDX) 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 10 bytes from a static external address to alternating locations (skipping one byte between each two bytes):

- In DMA channel primary control register (PRICTL):
  - SRCDIR = 00b, the static source address
  - DSTDIR = 11b, the programmable index value
- In DMA global index register (GBLIDX):
  - ELEIDX = 10b, the 2-byte destination stride
  - FRMIDX = $-(9 \times 2) = -18$ = FFEEh, restart destination for the transfer at the same location by moving 18 bytes.

2.5.5 Transferring a Large Single Block

Element count (ELECNT) can be used in conjunction with frame count (FRMCNT) to allow single-frame block transfers of more than 65535 bytes. The product of ELECNT and FRMCNT forms a larger effective element count. The following must be performed:

- If the address is to be adjusted using a programmable value (SRCDIR and/or DSTDIR = 11b), frame index (FRMIDX) must equal element index (ELEIDX) if the address adjustment is determined by a DMA global index register (GBLIDX). 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 (FS = 0, in PRICTL). 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 FRMCNT
  Er = ELECNT reload value
  Ei = Initial value of ELECNT

  Thus, to transfer 128K + 1 elements, you could set:

  F = 5
  Er = 32K
  Ei = 1

2.5.6 Sorting

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

- ELEMENT INDEX is set to $F \times S$
- FRAME INDEX is set to $-(((E - 1) \times F) - 1) \times S$

  where:

  E = Initial value of ELECNT (the number of elements per frame), initial value of the ELEMENT COUNT RELOAD
  F = Initial value of FRMCNT (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 = $3 \times 2 = 6$ and FRAME INDEX = $-(((4 - 1) \times 3) - 1) \times 2$ = FFF0h. Assume that the source address is not modified and the destination increments starting at 8000 0000h. Table 2–2 shows the data in the order in which it is transferred, and Table 2–3 shows how the data appears in memory after transfers are finished.

Table 2–2. 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 2–3. 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

2.6 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. The DMA global address register (GBLADDR) selected by the SPLIT field in the DMA channel primary control register (PRICTL) determines the address of the peripheral that is to be accessed for split transfer. Split-channel operation consists of transmit element transfers and receive element transfers. 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 in PRICTL. 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 in PRICTL.
  - **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.

When a DMA channel is operating in split mode, only one element count and one frame count are used for both the transmit and receive transfers. The end of frame or end of block is set following the last transfer. When the channel operating in split mode is servicing a McBSP, this will normally be the last receive transfer because the transmit transfers will normally run ahead of the receive transfers. The transfer counters will be modified after the transmit transfer, so that if autoinitialization is enabled, the transfer counters may indicate that another transfer has begun before the receive portion of the split-mode transfer has completed. 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 reinitializes) 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 exceed 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.

When a DMA channel is operating in split mode, only one element counter and one frame counter are used for both the transmit and receive transfers. The end of frame or end of block event is set following the last transfer. When the channel operating in split mode is servicing a McBSP, this will normally be the last receive transfer, because the transmit transfers will normally run ahead of the receive transfers. The transfer counters are modified after the transmit transfer, so that if autoinitialization is enabled, the transfer counters may indicate that another transfer has begun before the receive portion of the split-mode transfer has completed.

2.7 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 (PRICTL). The AUXPRI field in the DMA auxiliary control register (AUXCTL) 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.

- **Priority between DMA channels:** The DMA controller has a fixed priority scheme, with channel 0 having highest priority and channel 3 having lowest priority. The auxiliary channel can be given a priority anywhere within this hierarchy.

2.7.1 Priority Between DMA Channels

The fields in the DMA auxiliary control register (AUXCTL) affect the auxiliary channel. The priority assigned to each DMA channel determines which DMA channel performs a read or write transfer first, when two or more channels are ready to perform transfers. A channel's priority field should be modified only when that channel is paused or stopped.

The priority of the auxiliary channel is configurable by programming the CHPRI field in AUXCTL. By default, CHPRI contains 0000b at reset. This value sets the auxiliary channel as highest priority, followed by channel 0, channel 1, channel 2, and 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-channel 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 be 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.

2.7.2 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. The write transfer must complete before the higher priority channel will be able to start its transfer. Arbitration of the higher priority channel will occur as soon as the write from the lower priority channel completes. For example, if the lower priority channel's write is blocked by the CPU, the higher priority channel will not be able to start until the CPU releases the contending resource and the write is able to complete. This occurs even if the higher priority channel is accessing a different resource. See Chapter 3, *DMA and CPU Data Access Performance,* for more detail.

2.8 DMA Channel Condition Determination

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

The SECCTL also provides the means to enable the DMA channels to interrupt the CPU through their corresponding interrupt enable (IE) bit. 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 primary control register (PRICTL) is set, the logical OR of all enabled conditions forms the DMA_INT*n* signal. Otherwise, the DMA_INT*n* remains inactive. This logic is shown in Figure 2–3. 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 SXCOND, WDROPCOND, and RDROPCOND bits in SECCTL 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 conditions associated IE bit is set, that condition bit cannot be cleared automatically. A zero must be written to the COND bit to clear the bit. 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 SECCTL 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.

Table 2–4 describes each of the condition flags in SECCTL. Depending upon the system application, these conditions can represent errors. The last frame condition (LASTCOND) can be used to change the reload register values for autoinitialization. The frame index and element count reload are used every frame. Thus, you must wait to change these values until all but the last frame transfer in a block transfer finishes. Otherwise, the current block transfer is affected.

Figure 2–3. Generation of DMA Interrupt for Channel n *From Conditions*

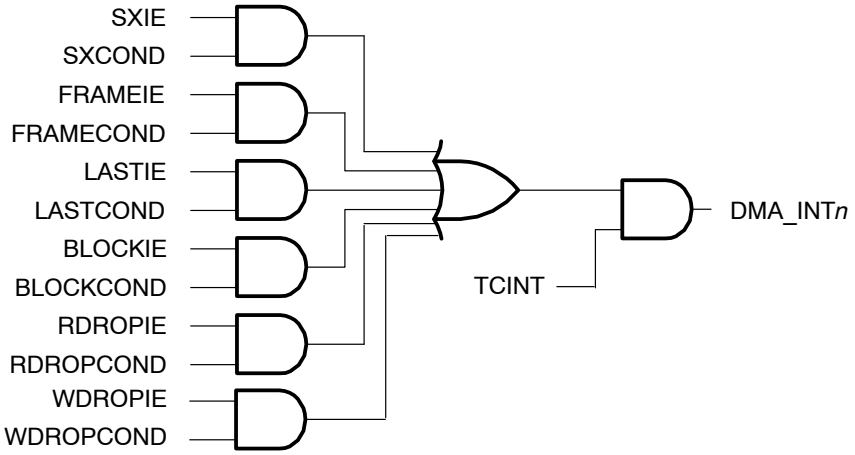


Table 2–4. DMA Channel Secondary Control Register (SECCTL) Condition Descriptions

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

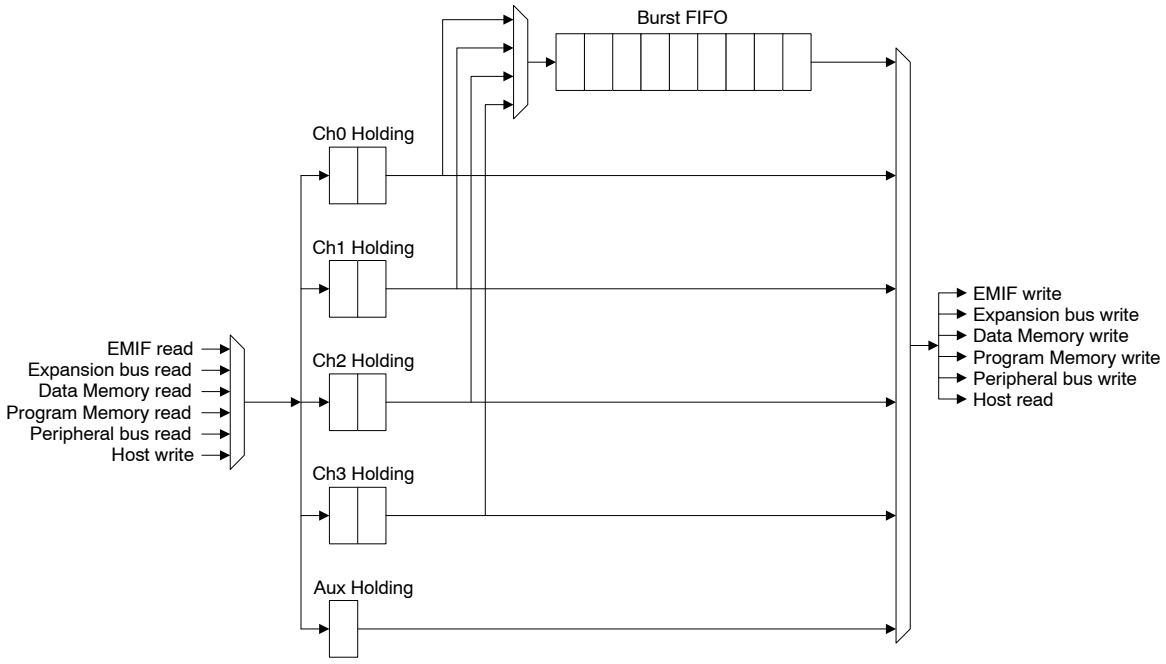
2.9 DMA Controller Structure

The C6000 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-channel mode for peripheral support. The DMA of the C6201/C6701/C6202 devices accomplish this by providing a pair of holding registers to each channel and a 9-deep shared FIFO. The DMA of the C6202B/C6203(B)/C6204/C6205 devices was modified to improve performance in certain instances; rather than a shared FIFO, each DMA channel has a dedicated 9-deep FIFO available to use.

2.9.1 TMS320C6201/C6701/C6202 DMA Structure

Figure 2−4 shows the internal data movement paths of the C6201/C6701/C6202 DMA controller, including data buses and internal holding registers.

Figure 2−4. DMA Controller Data Bus Block Diagram (C6201/C6701/C6202 DSP)



2.9.1.1 Read and Write Buses

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

- EMIF
- Expansion bus (C6202 DSP only)/host port interface (C6201/C6701 DSP)
- Internal data memory
- Internal program memory, block 0
- Internal program memory, block 1 (C6202 DSP 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.

2.9.1.2 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.

2.9.1.3 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-channel mode or not, additional restrictions can apply.

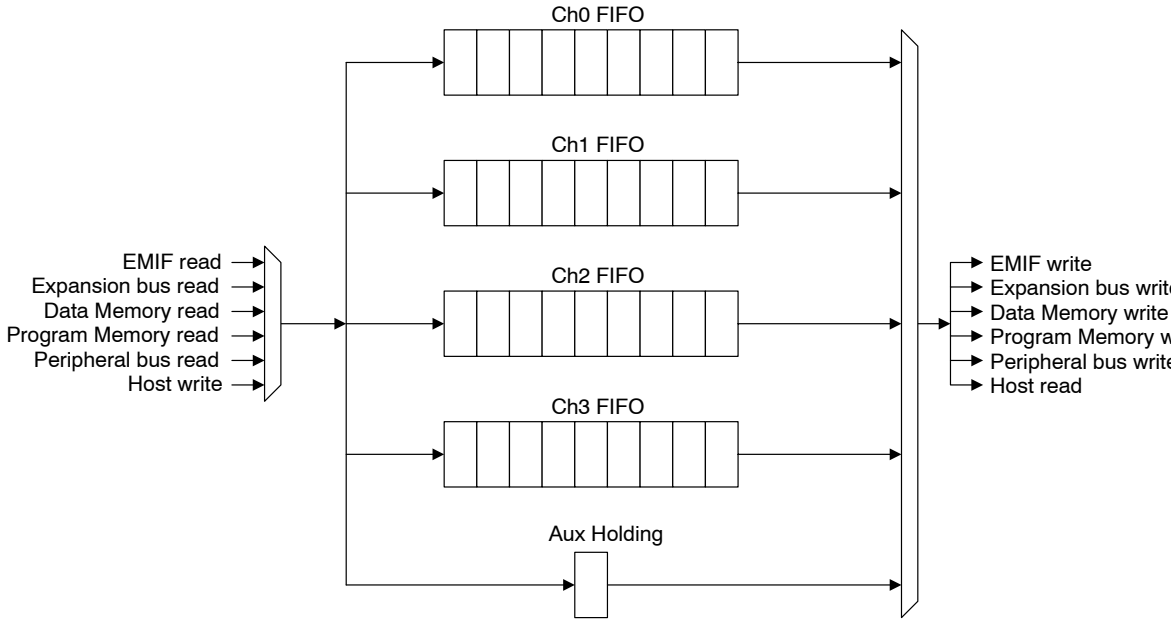
In split-channel mode, the two registers serve as separate transmit and receive data stream holding registers for split-channel mode. For both the transmit and receive-read transfer, no subsequent read transfer request is issued until the associated write transfer request completes.

In nonsplit-channel 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.

2.9.2 TMS320C6202B/C6203(B)/C6204/C6205 DMA Structure

The structure of the C6202B/C6203(B)/C6204/C6205 DMA was redesigned 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 2−5 shows the internal data movement paths of the DMA controller, including data buses and internal FIFOs.

Figure 2−5. DMA Controller Data Bus Block Diagram (C6202B/C6203(B)/C6204/C6205 DSP)



2.9.2.1 Read and Write Buses

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

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

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.

2.9.2.2 Channel FIFOs

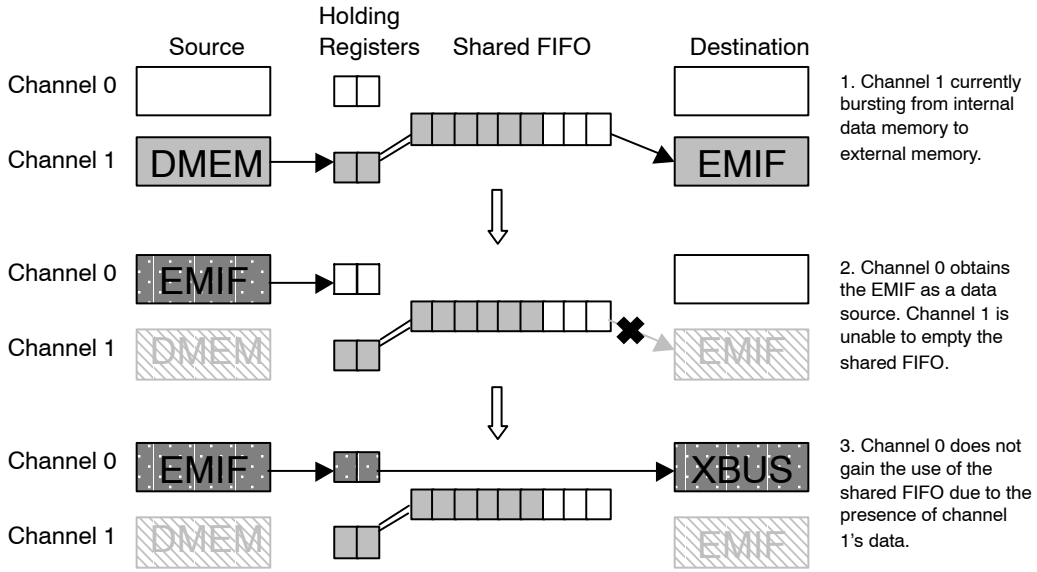
Each DMA channel has a dedicated 9-deep FIFO to facilitate bursting to high-speed memories. Each channel owns its own FIFO, which reduces the arbitration required for switching between high-speed bursting channels. The individual operation by any channel is unchanged from that of the C6201/C6701/C6202 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 C6201/C6701/C6202 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.

As with the shared FIFO DMA, a higher priority DMA channel can still be stalled by a lower priority channel if the lower priority channel is unable to complete its write to a resource. Arbitration of the higher priority channel occurs as soon as the write from the DMA completes.

The effect of this is that the interrupting high-priority channel is not able to burst properly, as shown in Figure 2–6, 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 2–6. Shared FIFO Resource Problem



The new structure of this DMA removes this bandwidth-limiting aspect of the C6201/C6701/C6202 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 these DMA channels is identical to those of the C6201/C6701/C6202 device.

2.9.2.3 Split-Channel Mode

When operating in split-channel mode, these DMA channels behave identically to a C6201/C6701/C6202 DMA. 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.

2.9.3 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.

2.9.4 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 3*, DMA and CPU Data Access Performance.*

2.10 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 DMACEN field in the DMA channel secondary control register (SECCTL), the DMAC pin can reflect the status of RSYNCSTAT, WSYNCSTAT, BLOCKCOND, or FRAMECOND or be treated as a high or low general-purpose output. If the DMAC pin reflects RSYNCSTAT or WSYNCSTAT 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.

2.11 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 to pause during this time or to continue running by setting the EMOD bit in the DMA channel primary control register (PRICTL). 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.

2.12 DMA Controller Registers

The DMA controller registers are listed in Table 2−5. See the device-specific datasheet for the memory address of these registers.

Table 2−5. DMA Controller Registers

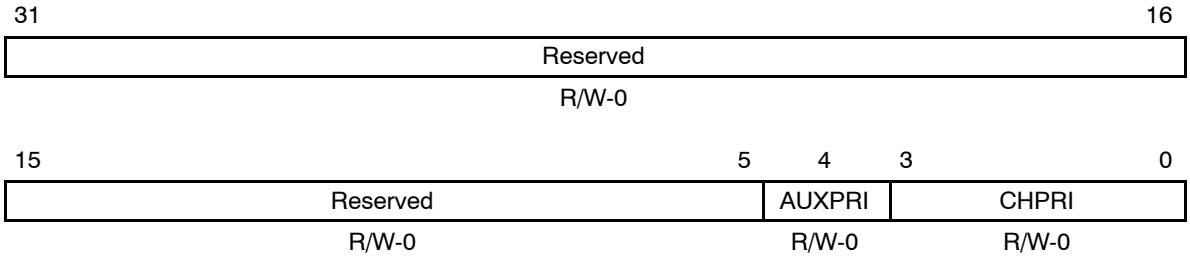
Acronym	Register Name	Section
AUXCTL	DMA auxiliary control register	2.12.1
PRICTL0−3	DMA channel primary control registers	2.12.2
SECCTL0−3	DMA channel secondary control registers	2.12.3
SRC0−3	DMA channel source address registers	2.12.4
DST0−3	DMA channel destination address registers	2.12.5
XFRCNT0−3	DMA channel transfer counter registers	2.12.6
GBLCNTA−B	DMA global count reload registers	2.12.7
GBLIDXA−B	DMA global index registers	2.12.8
GBLADDRA−D	DMA global address registers	2.12.9

2.12.1 DMA Auxiliary Control Register (AUXCTL)

The DMA auxiliary control register (AUXCTL) shown in Figure 2−7 and described in Table 2−6 affects the auxiliary channel. The priority assigned to each DMA channel determines which DMA channel performs a read or write transfer first, when two or more channels are ready to perform transfers.

A channel's priority field should be modified only when that channel is paused or stopped.

Figure 2−7. DMA Auxiliary Control Register (AUXCTL)



Legend: R/W = Read/write; -*n* = value after reset

Table 2−6. DMA Auxiliary Control Register (AUXCTL) Field Descriptions

Bit	*field*†	*symval*†	Value	Description
31−5	Reserved	−	0	Reserved. The reserved bit location is always read as 0. A value written to this field has no effect.
4	AUXPRI	OF(*value*)		Auxiliary channel priority mode bit.
		DEFAULT CPU	0	CPU priority
		DMA	1	DMA priority
3−0	CHPRI	OF(*value*)	0−Fh	DMA channel priority mode bits.
		DEFAULT HIGHEST	0	Fixed channel priority mode auxiliary channel highest priority
		2ND	1h	Fixed channel priority mode auxiliary channel 2nd-highest priority
		3RD	2h	Fixed channel priority mode auxiliary channel 3rd-highest priority
		4TH	3h	Fixed channel priority mode auxiliary channel 4th-highest priority
		LOWEST	4h	Fixed channel priority mode auxiliary channel lowest priority
		−	5h−Fh	Reserved

† For CSL implementation, use the notation DMA_AUXCTL_*field_symval*

2.12.2 DMA Channel Primary Control Registers (PRICTL0–3)

The DMA channel primary control register (PRICTL) shown in Figure 2–8 and described in Table 2–7 controls each DMA channel independently.

Figure 2–8. DMA Channel Primary Control Register (PRICTL)

31–30	29–28	27	26	25	24
DSTRLD	SRCRLD	EMOD	FS	TCINT	PRI
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

23–19	18–14	13	12
WSYNC	RSYNC	INDEX	CNTRLD
R/W-0	R/W-0	R/W-0	R/W-0

11–10	9–8	7–6	5–4	3–2	1–0
SPLIT	ESIZE	DSTDIR	SRCDIR	STATUS	START
R/W-0	R/W-0	R/W-0	R/W-0	R-0	R/W-0

Legend: R = Read only; R/W = Read/write; -*n* = value after reset

Table 2–7. DMA Channel Primary Control Register (PRICTL) Field Descriptions

Bit	*field*†	*symval*†	Value	Description
31–30	DSTRLD	OF(*value*)	0–3h	Destination address reload for autoinitialization bits.
		DEFAULT NONE	0	Do not reload during autoinitialization.
		B	1h	Use DMA global address register B as reload.
		C	2h	Use DMA global address register C as reload.
		D	3h	Use DMA global address register D as reload.
29–28	SRCRLD	OF(*value*)	0–3h	Source address reload for autoinitialization bits.
		DEFAULT NONE	0	Do not reload during autoinitialization.
		B	1h	Use DMA global address register B as reload.
		C	2h	Use DMA global address register C as reload.
		D	3h	Use DMA global address register D as reload.

† For CSL implementation, use the notation DMA_PRICTL_*field_symval*

Table 2–7. DMA Channel Primary Control Register (PRICTL) Field Descriptions (Continued)

Bit	*field*†	*symval*†	Value	Description
27	EMOD	OF(*value*)		Emulation mode bit.
		DEFAULT NOHALT	0	DMA channel keeps running during an emulation halt.
		HALT	1	DMA channel pauses during an emulation halt.
26	FS	OF(*value*)		Frame synchronization bit.
		DEFAULT DISABLE	0	Disable
		RSYNC	1	RSYNC event used to synchronize entire frame
25	TCINT	OF(*value*)		Transfer controller interrupt enable bit.
		DEFAULT DISABLE	0	Interrupt is disabled.
		ENABLE	1	Interrupt is enabled.
24	PRI	OF(*value*)		Priority mode bit.
		DEFAULT CPU	0	CPU priority
		DMA	1	DMA priority
23–19	WSYNC	OF(*value*)	0–1Fh	Write transfer synchronization bit.
		DEFAULT NONE	0	No synchronization
		TINT0	1h	Timer 0 interrupt event
		TINT1	2h	Timer 1 interrupt event
		SDINT	3h	EMIF SDRAM timer interrupt event
		EXTINT4	4h	External interrupt event 4
		EXTINT5	5h	External interrupt event 5
		EXTINT6	6h	External interrupt event 6
		EXTINT7	7h	External interrupt event 7

† For CSL implementation, use the notation DMA_PRICTL_*field_symval*

Table 2−7. DMA Channel Primary Control Register (PRICTL) Field Descriptions (Continued)

Bit	*field*†	*symval*†	Value	Description
		DMAINT0	8h	DMA channel 0 interrupt event
		DMAINT1	9h	DMA channel 1 interrupt event
		DMAINT2	Ah	DMA channel 2 interrupt event
		DMAINT3	Bh	DMA channel 3 interrupt event
		XEVT0	Ch	McBSP 0 transmit event
		REVT0	Dh	McBSP 0 receive event
		XEVT1	Eh	McBSP 1 transmit event
		REVT1	Fh	McBSP 1 receive event
		DSPINT	10h	DSP interrupt event
		XEVT2	11h	McBSP 2 transmit event
		REVT2	12h	McBSP 2 receive event
		–	13h−1Fh	Reserved
18−14	RSYNC	OF(*value*)	0−1Fh	Read synchronization bit.
		DEFAULT NONE	0	No synchronization
		TINT0	1h	Timer 0 interrupt event
		TINT1	2h	Timer 1 interrupt event
		SDINT	3h	EMIF SDRAM timer interrupt event
		EXTINT4	4h	External interrupt event 4
		EXTINT5	5h	External interrupt event 5
		EXTINT6	6h	External interrupt event 6
		EXTINT7	7h	External interrupt event 7
		DMAINT0	8h	DMA channel 0 interrupt event
		DMAINT1	9h	DMA channel 1 interrupt event
		DMAINT2	Ah	DMA channel 2 interrupt event

† For CSL implementation, use the notation DMA_PRICTL_*field_symval*

Table 2−7. DMA Channel Primary Control Register (PRICTL) Field Descriptions (Continued)

Bit	*field*†	*symval*†	Value	Description
		DMAINT3	Bh	DMA channel 3 interrupt event
		XEVT0	Ch	McBSP 0 transmit event
		REVT0	Dh	McBSP 0 receive event
		XEVT1	Eh	McBSP 1 transmit event
		REVT1	Fh	McBSP 1 receive event
		DSPINT	10h	DSP interrupt event
		XEVT2	11h	McBSP 2 transmit event
		REVT2	12h	McBSP 2 receive event
		−	13h−1Fh	Reserved
13	INDEX	OF(*value*)		Selects the DMA global data register to use as a programmable index.
		DEFAULT	0	Use DMA global index register A.
		A		
		B	1	Use DMA global index register B.
12	CNTRLD	OF(*value*)		Transfer counter reload for autoinitialization and multiframe transfers.
		DEFAULT	0	Reload with DMA global count reload register A.
		A		
		B	1	Reload with DMA global count reload register B.
11−10	SPLIT	OF(*value*)	0−3h	Split-channel mode enable bits.
		DEFAULT	0	Split-channel mode is disabled.
		DISABLE		
		A	1h	Split-channel mode is enabled; use DMA global address register A as split address.
		B	2h	Split-channel mode is enabled; use DMA global address register B as split address.
		C	3h	Split-channel mode is enabled; use DMA global address register C as split address.

† For CSL implementation, use the notation DMA_PRICTL_*field_symval*

Table 2−7. DMA Channel Primary Control Register (PRICTL) Field Descriptions (Continued)

Bit	*field*†	*symval*†	Value	Description
9−8	ESIZE	OF(*value*)	0−3h	Element size bits.
		DEFAULT 32BIT	0	32 bit
		16BIT	1h	16 bit
		8BIT	2h	8 bit
		−	3h	Reserved
7−6	DSTDIR	OF(*value*)	0−3h	Destination address modification after element transfers.
		DEFAULT NONE	0	No modification.
		INC	1h	Increment by element size in bytes.
		DEC	2h	Decrement by element size in bytes.
		IDX	3h	Adjust using DMA global index register selected by INDEX bit.
5−4	SRCDIR	OF(*value*)	0−3h	Source address modification after element transfers.
		DEFAULT NONE	0	No modification.
		INC	1h	Increment by element size in bytes.
		DEC	2h	Decrement by element size in bytes.
		IDX	3h	Adjust using DMA global index register selected by INDEX bit.
3−2	STATUS	OF(*value*)	0−3h	DMA channel status bit.
		DEFAULT STOPPED	0	Stopped
		RUNNING	1h	Running without autoinitialization
		PAUSED	2h	Paused
		AUTORUNNING	3h	Running with autoinitialization

† For CSL implementation, use the notation DMA_PRICTL_*field_symval*

Table 2–7. DMA Channel Primary Control Register (PRICTL) Field Descriptions (Continued)

Bit	*field*†	*symval*†	Value	Description
1–0	START	OF(*value*)	0–3h	DMA channel operation mode bit.
		DEFAULT STOP	0	Stop
		NORMAL	1h	Start without autoinitialization
		PAUSE	2h	Pause
		AUTOINIT	3h	Start with autoinitialization

† For CSL implementation, use the notation DMA_PRICTL_*field_symval*

2.12.3 DMA Channel Secondary Control Registers (SECCTL0–3)

The DMA channel secondary control register (SECCTL) shown in Figure 2–9 and described in Table 2–8 controls each DMA channel independently.

The SECCTL of the C6202(B)/C6203(B)/C6204/C6205 DSP 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.

Figure 2–9. DMA Channel Secondary Control Register (SECCTL)

31	30	29	28	27	26	25	24
Reserved							
R/W-0							

23	22	21	20	19	18	17	16
Reserved		WSPOL†	RSPOL†	FSIG†	DMACEN		
R/W-0		R/W-0	R/W-0	R/W-0	R/W-0		

15	14	13	12	11	10	9	8
WSYNCCLR	WSYNCSTAT	RSYNCCLR	RSYNCSTAT	WDROPIE	WDROPCOND	RDROPIE	RDROPCOND
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

7	6	5	4	3	2	1	0
BLOCKIE	BLOCKCOND	LASTIE	LASTCOND	FRAMEIE	FRAMECOND	SXIE	SXCOND
R/W-1	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

† Available only on C6202(B)/C6203(B)/C6204/C6205 devices; on C6201/C6701 devices, these bits are reserved, read only, with a default value of 0.

Legend: R/W = Read/write; -*n* = value after reset

Table 2–8. DMA Channel Secondary Control Register (SECCTL) Field Descriptions

No.	*field*†	*symval*†	Value	Description
31–22	Reserved	–	0	Reserved. The reserved bit location is always read as 0. A value written to this field has no effect.
21	WSPOL	OF(*value*)		**For C6202(B)/C6203(B)/C6204/C6205 DSP:** Write synchronization event polarity bit. This bit is valid only if EXT_INT*n* is selected.
		DEFAULT ACTIVEHIGH	0	Active high
		ACTIVELOW	1	Active low

† For CSL implementation, use the notation DMA_SECCTL_*field_symval*

Table 2−8. DMA Channel Secondary Control Register (SECCTL) Field Descriptions (Continued)

No.	*field*†	*symval*†	Value	Description
20	RSPOL	OF(*value*)		**For C6202(B)/C6203(B)/C6204/C6205 DSP:** Read and frame synchronization event polarity bit. This bit is valid only if EXT_INT*n* is selected.
		DEFAULT ACTIVEHIGH	0	Active high
		ACTIVELOW	1	Active low
19	FSIG	OF(*value*)		**For C6202(B)/C6203(B)/C6204/C6205 DSP:** Level/edge detect mode bit. This bit must be cleared to 0 for nonframe-synchronized transfers.
		DEFAULT NORMAL	0	Edge detect mode (FS = 1 or FS = 0).
		IGNORE	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.
18−16	DMACEN	OF(*value*)	0−7h	DMA action complete pins reflect status and condition.
		DEFAULT LOW	0	DMAC pin is held low.
		HIGH	1h	DMAC pin is held high.
		RSYNCSTAT	2h	DMAC reflects RSYNCSTAT.
		WSYNCSTAT	3h	DMAC reflects WSYNCSTAT.
		FRAMECOND	4h	DMAC reflects FRAMECOND.
		BLOCKCOND	5h	DMAC reflects BLOCKCOND.
		–	6h−7h	Reserved
15	WSYNCCLR	OF(*value*)		Write synchronization status clear bit.
		DEFAULT NOTHING	0	No effect.
		CLEAR	1	Clear write synchronization status.

† For CSL implementation, use the notation DMA_SECCTL_*field_symval*

Table 2–8. DMA Channel Secondary Control Register (SECCTL) Field Descriptions (Continued)

No.	*field*†	*symval*†	Value	Description
14	WSYNCSTAT	OF(*value*)		Write synchronization status bit.
		DEFAULT CLEAR	0	Synchronization is not received.
		SET	1	Synchronization is received.
13	RSYNCCLR	OF(*value*)		Read synchronization status clear bit.
		DEFAULT NOTHING	0	No effect.
		CLEAR	1	Clear read synchronization status.
12	RSYNCSTAT	OF(*value*)		Read synchronization status bit.
		DEFAULT CLEAR	0	Synchronization is not received.
		SET	1	Synchronization is received.
11	WDROPIE	OF(*value*)		Write synchronization dropped interrupt enable bit.
		DEFAULT DISABLE	0	WDROP condition does not enable DMA channel interrupt.
		ENABLE	1	WDROP condition enables DMA channel interrupt.
10	WDROPCOND	OF(*value*)		Write drop condition bit.
		DEFAULT CLEAR	0	WDROP condition is not detected.
		SET	1	WDROP condition is detected.
9	RDROPIE	OF(*value*)		Read synchronization dropped interrupt enable bit.
		DEFAULT DISABLE	0	RDROP condition does not enable DMA channel interrupt.
		ENABLE	1	RDROP condition enables DMA channel interrupt.

† For CSL implementation, use the notation DMA_SECCTL_*field_symval*

Table 2–8. DMA Channel Secondary Control Register (SECCTL) Field Descriptions (Continued)

No.	*field*†	*symval*†	Value	Description
8	RDROPCOND	OF(*value*)		Read drop condition bit.
		DEFAULT CLEAR	0	RDROP condition is not detected.
		SET	1	RDROP condition is detected.
7	BLOCKIE	OF(*value*)		Block transfer finished interrupt enable bit.
		DISABLE	0	BLOCK condition does not enable DMA channel interrupt.
		DEFAULT ENABLE	1	BLOCK condition enables DMA channel interrupt.
6	BLOCKCOND	OF(*value*)		Block transfer finished condition bit.
		DEFAULT CLEAR	0	BLOCK condition is not detected.
		SET	1	BLOCK condition is detected.
5	LASTIE	OF(*value*)		Last frame finished interrupt enable bit.
		DEFAULT DISABLE	0	LAST condition does not enable DMA channel interrupt.
		ENABLE	1	LAST condition enables DMA channel interrupt.
4	LASTCOND	OF(*value*)		Last frame finished condition bit.
		DEFAULT CLEAR	0	LAST condition is not detected.
		SET	1	LAST condition is detected.
3	FRAMEIE	OF(*value*)		Frame complete interrupt enable bit.
		DEFAULT DISABLE	0	FRAME condition does not enable DMA channel interrupt.
		ENABLE	1	FRAME condition enables DMA channel interrupt.

† For CSL implementation, use the notation DMA_SECCTL_*field_symval*

Table 2–8. DMA Channel Secondary Control Register (SECCTL) Field Descriptions (Continued)

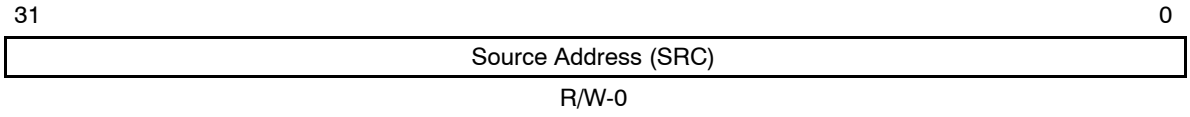
No.	*field*†	*symval*†	Value	Description
2	FRAMECOND	OF(*value*)		Frame complete condition bit.
		DEFAULT CLEAR	0	FRAME condition is not detected.
		SET	1	FRAME condition is detected.
1	SXIE	OF(*value*)		Split transmit overrun receive interrupt enable bit.
		DEFAULT DISABLE	0	SX condition does not enable DMA channel interrupt.
		ENABLE	1	SX condition enables DMA channel interrupt.
0	SXCOND	OF(*value*)		Split transmit condition bit.
		DEFAULT CLEAR	0	SX condition is not detected.
		SET	1	SX condition is detected.

† For CSL implementation, use the notation DMA_SECCTL_*field_symval*

2.12.4 DMA Channel Source Address Registers (SRC0–3)

The DMA channel source address (SRC) register shown in Figure 2–10 and described in Table 2–9 holds the address for the next read transfer.

Figure 2–10. DMA Channel Source Address Register (SRC)



Legend: R/W = Read/write; -*n* = value after reset

Table 2–9. DMA Channel Source Address Register (SRC) Field Descriptions

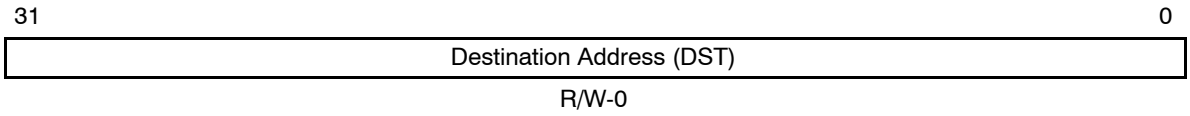
Bit	Field	*symval*†	Value	Description
31–0	SRC	OF(*value*)	0–FFFF FFFFh	This 32-bit source address specifies the address for the next read transfer.
		DEFAULT	0	

† For CSL implementation, use the notation DMA_SRC_SRC_*symval*

2.12.5 DMA Channel Destination Address Registers (DST0–3)

The DMA channel destination address register (DST) shown in Figure 2–11 and described in Table 2–10 holds the address for the next write transfer.

Figure 2–11. DMA Channel Destination Address Register (DST)



Legend: R/W = Read/write; -*n* = value after reset

Table 2–10. DMA Channel Destination Address Register (DST) Field Descriptions

Bit	Field	*symval*†	Value	Description
31–0	DST	OF(*value*)	0–FFFF FFFFh	This 32-bit destination address specifies the address for the next write transfer.
		DEFAULT	0	

† For CSL implementation, use the notation DMA_DST_DST_*symval*

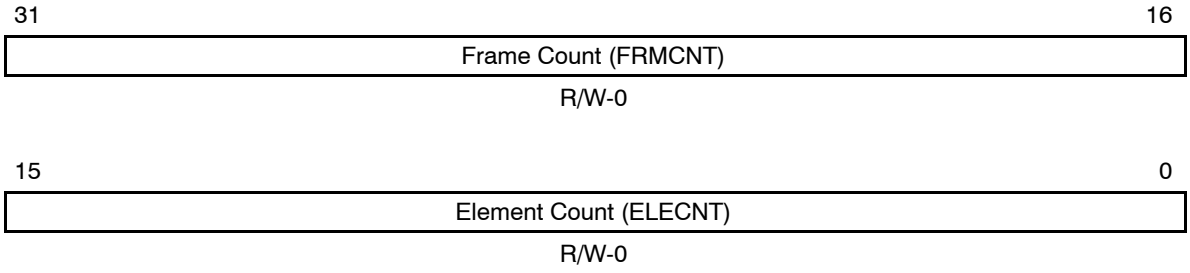
2.12.6 DMA Channel Transfer Counter Registers (XFRCNT0–3)

The DMA channel transfer counter register (XFRCNT) shown in Figure 2–12 and described in Table 2–11 contains fields that represent the number of frames and the number of elements per frame to be transferred.

Frame Count (FRMCNT) field: This 16-bit unsigned value 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, FRMCNT is reloaded with the 16 MSBs in the DMA global count reload register (GBLCNT) selected by the CNTRLD field in the DMA channel primary control register (PRICTL). Initial values of 0 and 1 in FRMCNT have the same effect of transferring a single frame.

Element Count (ELECNT) field: This 16-bit unsigned value sets the number of elements per frame. The maximum number of elements per frame transfer is 65535. This counter is decremented after the read transfer of each element. Once the last element in each frame is reached, ELECNT is reloaded with the 16 LSBs in the DMA global count reload register (GBLCNT) selected by the CNTRLD field in the DMA channel primary control register (PRICTL). This reloading is unaffected by autoinitialization mode. Before a block transfer begins, the counter and selected DMA 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 ELECNT is initialized as 0, operation is undefined.

Figure 2−12. DMA Channel Transfer Counter Register (XFRCNT)



Legend: R/W = Read/write; -*n* = value after reset

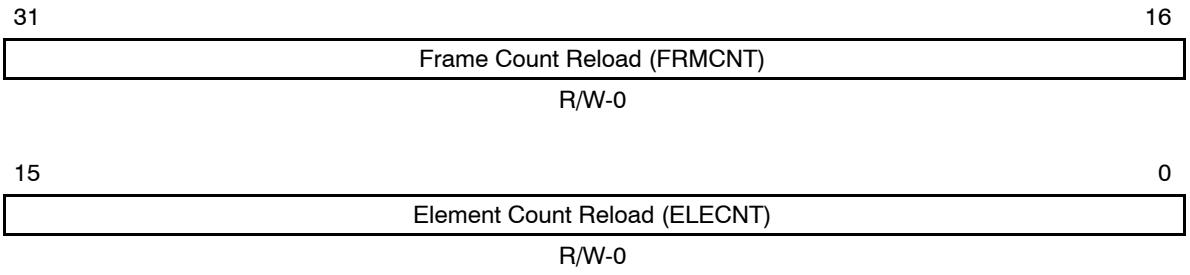
Table 2−11. DMA Channel Transfer Counter Register (XFRCNT) Field Descriptions

Bit	***field***†	***symval***†	Value	Description
31−16	FRMCNT	OF(*value*)	0−FFFFh	A 16-bit unsigned value specifies the number of frames in a block transfer. Values of 0 and 1 have the same effect of transferring a single frame.
		DEFAULT	0	
15−0	ELECNT	OF(*value*)	0−FFFFh	A 16-bit unsigned value that specifies the number of elements in a frame.
		DEFAULT	0	

† For CSL implementation, use the notation DMA_XFRCNT_*field_symval*

2.12.7 DMA Global Count Reload Registers (GBLCNTA–B)

Figure 2–13. DMA Global Count Reload Register (GBLCNT)



Legend: R/W = Read/write; -*n* = value after reset

Table 2–12. DMA Global Count Reload Register (GBLCNT) Field Descriptions

Bit	*field*†	*symval*†	Value	Description
31–16	FRMCNT	OF(value)	0–FFFFh	This 16-bit value reloads the FRMCNT bits in the DMA channel transfer counter register (XFRCNT).
		DEFAULT	0	
15–0	ELECNT	OF(value)	0–FFFFh	This 16-bit value reloads the ELECNT bits in the DMA channel transfer counter register (XFRCNT).
		DEFAULT	0	

† For CSL implementation, use the notation DMA_GBLCNT_*field_symval*

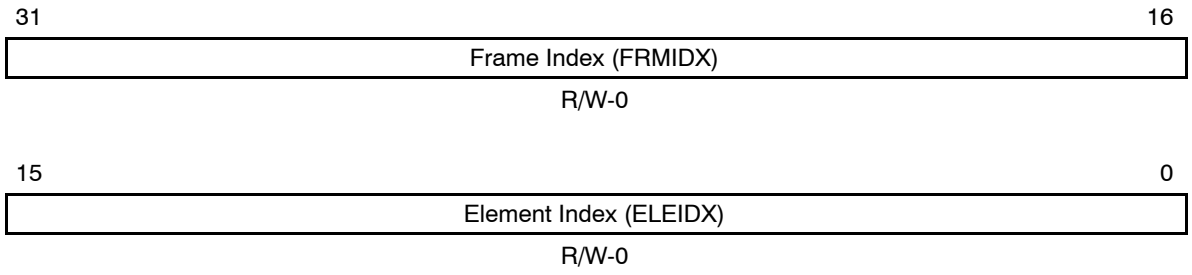
2.12.8 DMA Global Index Registers (GBLIDXA–B)

The DMA global index register (GBLIDX) shown in Figure 2–14 and described in Table 2–13 is selected by the INDEX field in the DMA channel primary control register (PRICTL).

- **Element Index (ELEIDX) field:** This 16-bit signed value specifies the element index used for an address offset to the next element in a frame. The index value can range from –32768 to 32767. For element transfers, except the last one in a frame, ELEIDX determines the amount to be added to the DMA channel source or the destination address register; as selected by the SRCDIR or DSTDIR field in PRICTL after each read or write transfer, respectively.

- **Frame Index (FRMIDX) field:** This 16-bit signed value specifies the frame index used for an address modification. The index value can range from –32768 to 32767. If the read or write transfer is the last in a frame, FRMIDX (and not ELEIDX) is used for address adjustment. This adjustment occurs in both single frame and multiframe transfers, including transfers after the last frame in a block.

Figure 2−14. DMA Global Index Register (GBLIDX)



Legend: R/W = Read/write; -*n* = value after reset

Table 2−13. DMA Global Index Register (GBLIDX) Field Descriptions

Bit	*field*†	*symval*†	Value	Description
31−16	FRMIDX	OF(*value*)	0−FFFFh	This 16-bit signed value specifies the frame index used for an address modification. This value is used by the DMA for address updates in both single frame and multiframe transfers.
		DEFAULT	0	
15−0	ELEIDX	OF(*value*)	0−FFFFh	This 16-bit signed value specifies the element index used for an address offset to the next element in a frame.
		DEFAULT	0	

† For CSL implementation, use the notation DMA_GBLIDX_*field_symval*

2.12.9 DMA Global Address Registers (GBLADDRA−D)

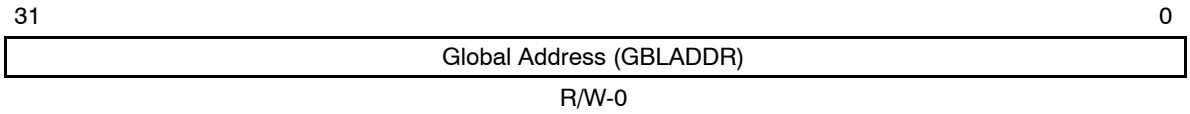
The DMA global address register (GBLADDR) shown in Figure 2−15 and described in Table 2−14 is selected by the SPLIT field in the DMA channel primary control register (PRICTL). The GBLADDR determines the address of the peripheral that is to be accessed for split transfer:

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

The two LSBs should be fixed at 0 to force alignment at a word address. The third LSB should also be 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.

Split-channel mode cannot be used with the expansion bus. The source address, destination address, nor the split address can be within the expansion bus I/O memory range.

Figure 2−15. DMA Global Address Register (GBLADDR)



Legend: R/W = Read/write; -*n* = value after reset

Table 2−14. DMA Global Address Register (GBLADDR) Field Descriptions

Bit	Field	*symval*†	Value	Description
31−0	GBLADDR	OF(*value*)	0−FFFF FFFFh	This 32-bit address specifies the address of the peripheral that is to be accessed for the split transfer.
		DEFAULT	0	

† For CSL implementation, use the notation DMA_GBLADDR_GBLADDR_*symval*

Chapter 3

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.

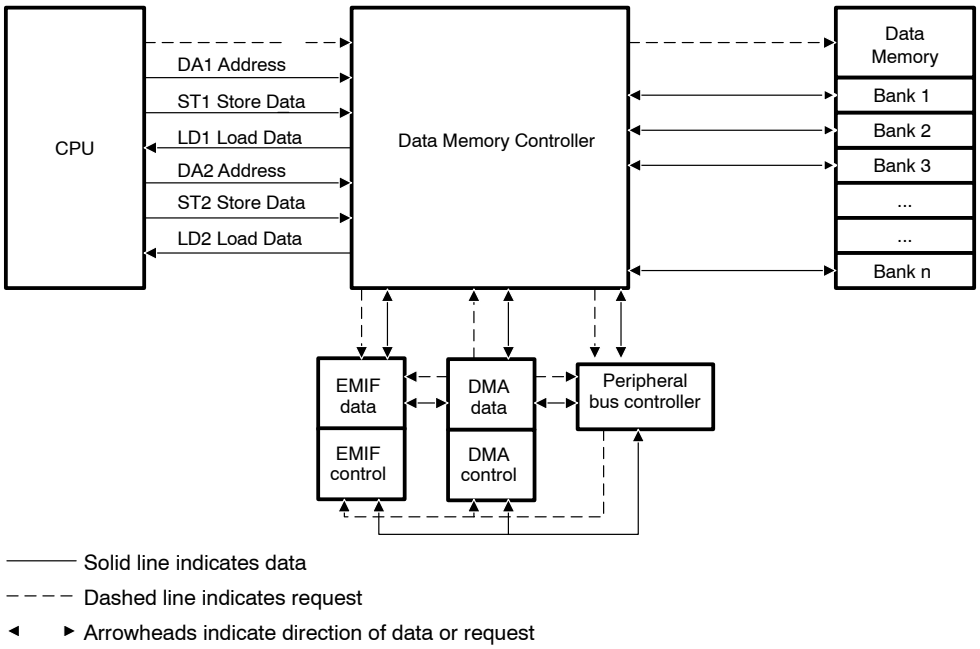
In a real-time system, it is important to understand data flow and control it to achieve high performance. By analyzing the timing characteristics for accessing data and switching between data requestors, it is possible to maximize the achievable bandwidth in any system. In general, one important guideline is to use the DMA controller for background off-chip data accesses. The CPU should only be used for sporadic (nonperiodic) 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 chapter 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 chapter also provides guidelines on how to maximize the available bandwidth.

Topic **Page**

3.1 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 primary control register (PRICTL). This arbitration is valid for all resources. Figure 3–1 shows the CPU, data memory controller, and peripheral bus connections.

Figure 3–1. Data Paths



3.1.1 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.

3.1.2 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 3–1. A single CPU access to the peripheral bus stalls the CPU for 5 cycles, and parallel CPU accesses stall the CPU for 9 cycles.

DMA accesses to the peripheral bus are pipelined, allowing the DMA to access peripheral bus every 3 cycles.

Table 3–1. CPU Stalls For Peripheral Register Accesses

CPU Access	CPU Stall
Single	5
Parallel	9

3.1.3 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.

3.1.3.1 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 ($\overline{CE}$) to active (low).

The time used by the EMIF at the end of an external data access is provided in Table 3–2. Table 3–2 shows the number of CLKOUT1 cycles between the external strobe for a particular memory ($\overline{AOE}$, $\overline{AWE}$, $\overline{SSOE}$, $\overline{SSWE}$, or $\overline{SDCAS}$) and $\overline{CE}$ 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.

Note:

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

Table 3–2. EMIF Data Access Completion Timings in CLKOUT1 (CPU Clock) Cycles

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

After the $\overline{CE}$ is reasserted 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 $\overline{CE}$ signals). These timings vary slightly depending on the memory type, the requestors, and the situation of the switching request.

3.1.3.2 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 3–3 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 ($\overline{CE}$_READ_HOLD or $\overline{CE}$_WRITE_HOLD) are provided in Table 3–2. 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 3–3. 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 in Table 3–3. 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.

3.1.3.3 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.

3.1.4 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.

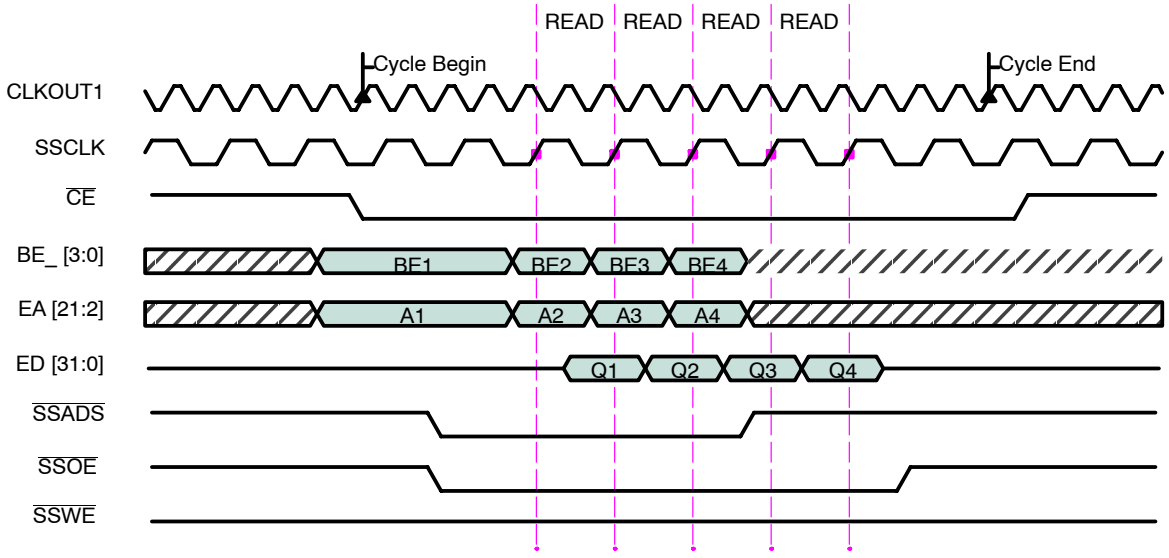
For accesses to external memory through the EMIF, the number of cycles in between accesses depends on the memory type and the direction of the accesses. Since memory timings and latencies vary according to different memory types, and since the external memories do not run off the CPU clock, the switching times are not uniform across all memory types. The delay times between external accesses are provided in Table 3–4. These switching times are valid for CPU/CPU, DMA/CPU, CPU/DMA, and DMA/DMA access boundaries.

Table 3–4. External Switching Time between Accesses by Different Requestors

Current Access		Subsequent Access					
		ASRAM		SDRAM		SBSRAM	
		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 $\overline{CE}$ 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 $\overline{CE}$ transition from 0 to 1. Figure 3–2 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 $\overline{CE}$ transitions.

Figure 3–2. 1/2× Rate SBSRAM Read Cycle Timing Diagram



3.1.4.1 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 3–4, 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 3–4, plus an additional offset shown in Table 3–5.

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.

Table 3–5. 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

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.

3.1.4.2 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 Autoinitialization

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 autoinitialization, 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 3–6 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 3–6. 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 3–7 shows the time between memory accesses ($\overline{CE}$ 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 3−7. Burst Interruption by McBSP/Host Service

Current Access		Burst Cycles Idle When Servicing McBSP/Host		
		McBSP Read/ Host Write	**McBSP Write/ Host Read**	**McBSP Read and 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 is described in section 3.1.4.1, *Switching Between DMA Channels*.

3.1.5 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 3−8 shows the time delay from a synchronization event to the beginning of a data access.

Table 3−8. 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.

3.1.6 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 (for example, asynchronous memory) then only a few elements burst at a time. If a high-speed memory is being read (for example, internal data memory) then more of the FIFO is used.

Table 3–9 lists the number of elements per burst when reading from, and writing to, the same resource.

Table 3–9. 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 3−10 lists the number of cycles between reads and writes for transfers between external memories, in addition to those provided in Table 3−4.

Table 3−10. 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 3−11.

Table 3−11. 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 3−10 and Table 3−11, 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.

3.1.7 DMA Port Crossing

The DMA has 4−6 master ports, all of which are listed below:

- Data Memory
- Program Memory Block 0
- Program Memory Block 1 (on C6202/C6203 DSP only)
- Expansion Bus (XBUS) I/O (on C6202/C6203/C6204 DSP only)
- External Memory Interface (EMIF)
- Internal Peripheral Bus (peripheral control registers including McBSP data registers)

The DMA auxiliary port is a slave port and should be considered a requestor much like a programmed DMA channel. DMA accesses/bursts are not permitted to cross a port boundary.

3.2 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 section. Such an analysis allows a designer to make sure that all data processing can be performed in the time allotted for it.

3.2.1 Simple Bandwidth Calculation Example Using Timing Information

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

- DMA channel 0 performs an unsynchronized transfer of 32-bit data from 1/2× 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 1/2× 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 determine 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 3−12.

Table 3−12. Timing Parameter Descriptions for Simple Bandwidth Calculation 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 3−2, page 3-5	Cycle count from the last strobe in a read burst from SBSRAM to the end of the access
CE_write_hold	4	Table 3−2, page 3-5	Cycle count from the last strobe in a write burst to SBSRAM to the end of access
read_frame_gap	4	Table 3−6, page 3-9	Time between read bursts for multi-frame transfer
read_to_write	6	Table 3−4, page 3-7	Switching time between a SBSRAM read access and a SBSRAM write access

Table 3−12. Timing Parameter Descriptions for Simple Bandwidth Calculation Example (Continued)

Parameter	Value	Location	Description
DMA_hp_read_to_lp_write	15	Table 3−5, page 3-8	Additional switching time between a high priority read access and a low priority write access
start_to_sync	1	Table 3−8, page 3-10	Time from setting START = 01b to the setting of RSYNC_STAT
RSYNC_STAT_to_read	11	Table 3−8, page 3-10	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:

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

  $= 2 \times [4 + (2 \times 20) + 4] = 96$ *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:

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

  $= (2 \times 40) + 4 = 84$ *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 $\overline{CE}$ 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*

3.2.2 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 15 μs.
- 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 3–13 describes all of the timing parameters used in the calculations, along with their location in this chapter.

Table 3−13. Timing Parameter Descriptions For Complex Bandwidth Calculation Example

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

= [(2 + 4) $\times$ 128] + [1 $\times$ (128 $-$ 1)] + 6 = 901 *cycles*

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

[(setup + strobe) $\times$ *element_count] + [hold* $\times$ *(element_count* $-$ 1*)]*
+ CE_write_hold

= [(2 + 4) $\times$ 128] + [1 $\times$ (128 – 1)] + 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.

3.2.2.1 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 3–14, transfer priority is ranked according to frequency. Based on the numbers shown in the timeslice calculation the priority ranking is as shown.

Table 3−14. 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:

$2 \times$ *(CE_read_hold + mcbsp_read_interruption) for McBSP reads and*
$2 \times$ *(CE_write_hold + mcbsp_write_interruption) for McBSP writes*

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

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

$4 \times$ *(CE_write_hold + write_to_read + DMA_lp_write_to_hp_read)*

$= 4 \times (3 + 1 + 4) = 32$ *cycles* every 12000 cycles

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

$6 \times$ *(read_to_write + DMA_hp_read_to_lp_write)*

$= 6 \times (2 + 15) = 102$ *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* $\times$ *100%*

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

3.2.2.2 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 DSP), 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 DSP) devices, however, some additional steps must be taken due to the shared FIFO present in the DMA. As described in section 2.9.2.2, *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:

$6 \times$ *(RSYNC_STAT_to_read)*

$= 6 \times 9$
= 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.

3.3 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 design CPU and DMA activity to work as efficiently as possible to meet performance goals.

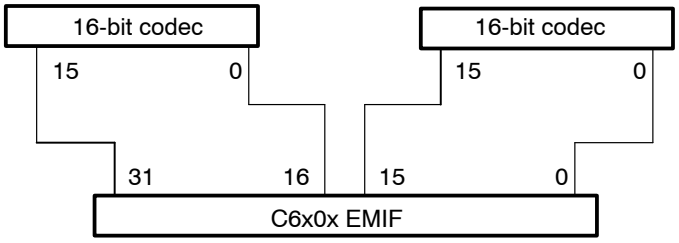
There are typically multiple ways to implement tasks, both with the CPU and with the DMA. Understanding the implications of the different options can allow the best to be chosen.

3.3.1 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-bit 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 3–3. This allows for both to be accessed simultaneously. This requires synchronization of the data streams to insure that valid data is always read.

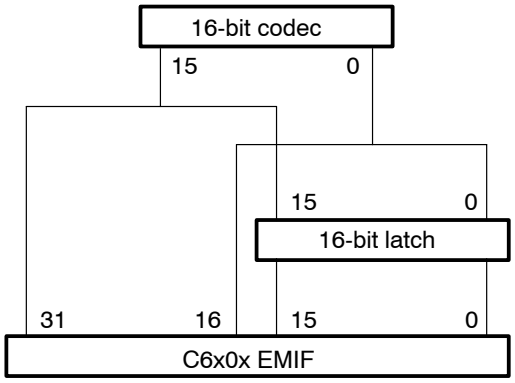
Figure 3–3. Combining External Peripherals



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 3–4. 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.

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.

Figure 3–4. Converting a 16-Bit Peripheral to 32-Bit



3.3.2 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 (that is, 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, these "lost" cycles can be minimized if the DMA burst is interrupted only once per group of CPU accesses.

Appendix A

Revision History

Table A−1 lists the changes made since the previous version of this document.

Table A−1. Document Revision History

Page	Additions/Modifications/Deletions
1-11	Changed the title in section 1.2.4.2 to Cache Invalidation. Changed section 1.2.4.2 to: The tag RAM contains a valid bit for each line of the cache. When a program enables the cache, tee PMEMC clears the valid bit for each tag entry. This invalidates the entire contents of the program cache and prepares it for use. Programs can change the current cache mode by writing to the PCC field in CSR. The PMEMC only invalidates its contents when it transitions out of the mapped-memory mode. In other words, the PMEMC invalidates the cache's contents when programs write 010b to PCC and the PCC previous value was 000b. The PMEMC halts the CPU while it initializes its tags. On the C6202(B) and C6203(B) DSPs, the PMEMC allows DMA accesses to proceed to Block 0 during this initialization.
2-33−2-51	Updated symbolic values (*symval*) of the bits in Table 2−6 through Table 2−14.

Index

A

B

C

D

E

F

G

I

L

M

O

P

R

S

T

U

W

X