Application Report
SPRA994 − March 2004

TMS320C64x EDMA Architecture

Jeffrey Ward
Jamon Bowen

TMS320C6000 Architecture

ABSTRACT

The enhanced DMA (EDMA) controller of the TMS320C64x™ device is a highly efficient data transfer engine. To maximize bandwidth, minimize transfer interference, and fully utilize the resources of the EDMA, it is crucial to understand the architecture of the engine. Transfer requests (TRs) originate from many requestors, including sixty-four programmable EDMA channels, the level 2 (L2) memory controller, and other master peripherals. The EDMA controls access to resources and arbitrates between concurrent transfers. Understanding the interaction points for transfer requests in the EDMA architecture is crucial to creating a system that takes full advantage of EDMA's capabilities.

Contents

List of Figures

List of Tables

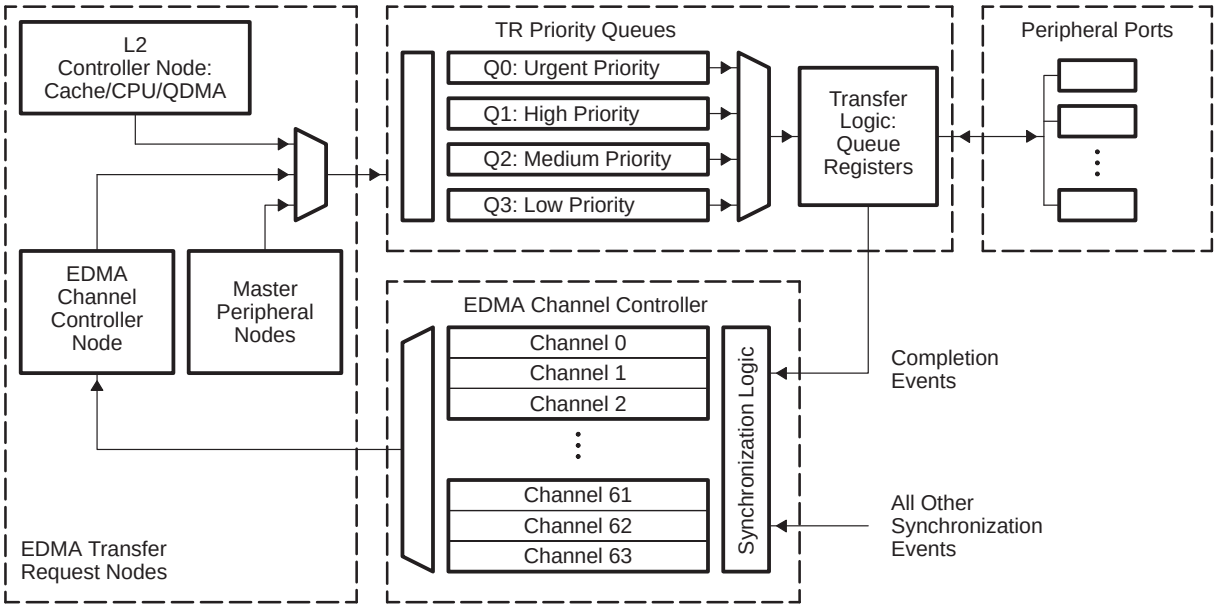

Figure 1. EDMA Architecture Overview

1 Introduction

The enhanced DMA (EDMA) controller of the TMS320C64x devices is a highly efficient data transfer engine, capable of handling up to 8 bytes per EDMA cycle, resulting in 2.4 Giga-bytes per second of total data throughput at a CPU rate of 600 MHz (the EDMA frequency being CPU frequency divided by two). The EDMA performs all data movement between the on-chip level-two (L2) memory, external memory (connected to the device through an external memory interface (EMIF), and the device peripherals. These data transfers include CPU-initiated and event-triggered transfers, master peripheral accesses, cache servicing, and non-cacheable memory accesses. The EDMA architecture has many features designed to service multiple high-speed data transfers simultaneously. With a working knowledge of this architecture and the ways in which data transfers interact and are performed, it is possible to create an efficient system and to maximize the bandwidth utilization of the EDMA.

2 EDMA Architecture

The most important thing to understand, prior to setting up the data movement in a system, is the architecture of the transfer engine. By understanding this architecture, it is possible to understand the stages through which a transfer is accomplished. The architecture is the key to knowing how multiple transfers (from multiple transfer requestors) interact with one another, and ultimately how they impact the system performance.

2.1 Data Transfer Overview

Each data transfer is initiated by a transfer request (TR), which contains all the information required to perform the transfer: source address, destination address, transfer priority, element count, etc. TRs are sorted into queues based on priority. Once at the head of the queue, the TR is moved into the EDMA transfer controller's queue registers, which perform the actual data movement defined by the TR.

The entire process of TR submission, priority queuing, and arbitration occurs at the speed of the EDMA, which is CPU frequency divided by two. Data movement at the peripheral occurs at the speed of the peripheral. The peripheral ports buffer data to isolate the high speed EDMA from the peripherals. This is a very efficient architecture, allowing the EDMA to service multiple simultaneous data transfers.

2.2 Transfer Requestors

There are up to three requestors of data transfers inside the DSP: the L2 cache/memory controller, the EDMA channels, and the master peripherals. The transfers requested are likely to be different due to the different tasks that each performs. However, the way each transfer request is handled by the EDMA transfer controller is the same, regardless of its requestor.

2.2.1 Level-Two Memory Controller

The L2 cache/memory controller performs many functions. It services CPU data accesses, submits quick DMA (QDMA) transfer requests, and maintains the coherency of the level-1 cache and the level-2 cache (if enabled). All communication between the CPU block and the rest of the device must pass through the L2 controller as depicted in Figure 2.

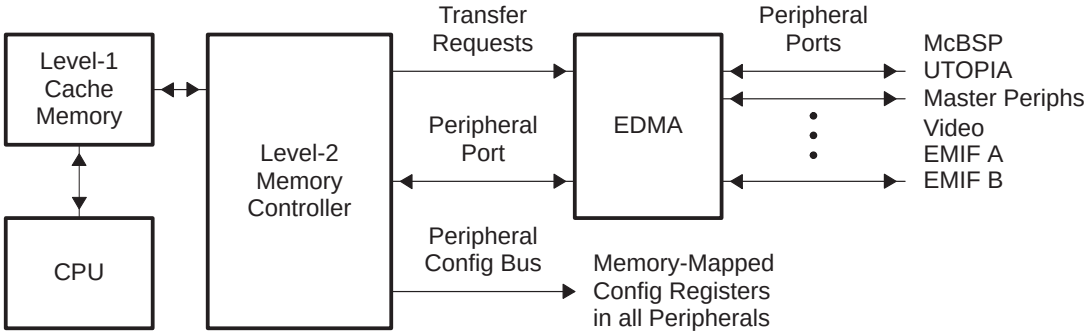

Figure 2. L2 Controller Functionality

The L2 controller directs QDMA requests and external memory accesses to the EDMA, L2 cache/memory accesses to the L2 memory, and memory-mapped control register accesses to the peripheral configuration (config) bus. In addition, if any L2 memory is set up as cache, it maintains the coherency of the data between the cache and the cacheable memory space(s). The L2 controller receives L2 memory accesses from the CPU side and from the EDMA side. In the case of contention, the EDMAWEIGHT register defines which requestor takes precedence. The EDMAWEIGHT register is documented in the *TMS320C6000 EDMA Controller Reference Guide* (literature number SPRU234).

Some accesses to the L2 controller result in an EDMA transfer request (TR); others do not. The L2 controller generates a TR for the following conditions:

- The CPU issues a QDMA transfer.
- The CPU accesses a non-cacheable external memory space.
- The L2 controller performs a cache allocation from external memory – the result of a CPU access to a cacheable memory space.

- The L2 controller performs a cache eviction to external memory – the result of a cache allocation which has no space to land in the cache memory.
- User-initiated cache operations (flush, clean, etc.).

The L2 controller submits all CPU and cache servicing transfer requests on the EDMA priority level set in the priority bits in the cache configuration (CCFG) register. QDMA transfers can be set to any priority on a per-transfer basis via the priority bits of the QDMA options register.

Note that some accesses and data paths do not pass through the EDMA. The L2 controller does not generate an EDMA transfer request for the following conditions:

- The CPU accesses memory in the L2 SRAM space. This access goes directly to L2 within the cache memory system; refer to *TMS320C64x DSP Two Level Internal Memory Reference Guide* (SPRU610).
- The CPU accesses a memory-mapped config register. This access passes through the config bus.
- The CPU accesses a cacheable external memory element that is allocated in L2 or L1 cache. This access goes directly to L2 within the cache memory system; refer to *TMS320C64x DSP Two Level Internal Memory Reference Guide* (SPRU610).

Details on programming the QDMA can be found in the *TMS320C6000 EDMA Controller Reference Guide* (literature number SPRU234). Information about configuring the L2 cache, defining cacheable and non-cacheable external memory spaces, and programming the cache configuration registers can be found in the *TMS320C64x Two-Level Internal Memory Reference Guide* (literature number SPRU610).

2.2.2 EDMA Channel Controller

There are sixty-four EDMA channels that can be configured in a special on-chip parameter RAM (PaRAM), with each channel corresponding to a specific synchronization event to trigger the transfer. The RAM-based structure of the EDMA allows for a great deal of flexibility. Each channel has a complete parameter set accessible via the peripheral config bus, which makes each channel's transfer parameters independent of one another. To allow for some interaction between transfers a linking mechanism is available to EDMA channels. Once fully exhausted, new channel parameters may be automatically loaded with a new set that is stored in the PaRAM via the linking mechanism.

One EDMA TR is issued per synchronization event received. The transfers requested by the EDMA channels are completely dependent on the configuration programmed by the user. Details on programming EDMA channels are not included in this document. For transfer examples see the examples section of the *TMS320C6000 EDMA Controller Reference Guide* (literature number SPRU234).

2.2.3 Master Peripherals

Master peripherals include the HPI, the PCI, and the EMAC. Master peripheral servicing is performed without any user intervention. These peripherals have a direct connection to the EDMA, with limited user-programmability. The direct connection allows master peripherals to submit transfer requests to the EDMA transfer controller in the same fashion as the L2 controller and the EDMA channels.

The requests made to the EDMA are dependent on the master activity, but consist of transfers between a master peripheral and the rest of the system memory. These transfer requests can transfer data between any location in the DSP's memory map and the master peripherals. The priority level of these transfers is determined by the TR control register (TRCTL), located in the master peripheral register set.

For information on programming the master peripherals, see the appropriate chapter referenced in the *TMS320C6000 DSP Peripherals Overview Reference Guide* (literature number SPRU190).

2.3 Transfer Request Bus

The transfer requestors to the EDMA are connected to the transfer controller (TC) via the transfer request (TR) bus. If multiple TRs arrive at the TR bus simultaneously, they are submitted in the order of their priority. This has little impact on performance because these requests are arbitrated quickly (in about 2-4 EDMA cycles) compared to data transfer rates.

2.4 Transfer Controller

Transfer requests are queued in the transfer controller based on their priority. The transfer controller is the portion of the EDMA that processes the TR and performs the actual data movement (see Figure 1).

Within the TC, the TR is shifted into one of the transfer request queues to await processing. The transfer priority level determines the queue to which it is submitted. There are four queues, corresponding to four priority levels, each with a depth of 16 entries: Q0 (urgent), Q1 (high), Q2 (medium), and Q3 (low). Each TMS320C64x transfer requestor is programmable such that it can submit TRs on any priority level.

Once the transfer request reaches the head of its queue, it is submitted to the queue registers to be processed. Only one TR from each priority queue can be serviced at a time by the address generation/transfer logic. The transfer logic can process transfers of different priorities concurrently. To maximize the data transfer bandwidth in a system, transfers should be distributed among all four priorities whenever possible. This topic is discussed at length in *TMS320C6000 EDMA IO Scheduling and Performance* (literature number SPRAA00).

The TC contains four queue register sets, one for each priority queue, which monitor the progress of a transfer. Within the register set for a particular queue, the current source address, destination address, and count are maintained for a transfer. These registers are not present in the device's memory map and are unavailable to the CPU.

The TC is connected to peripherals via peripheral ports. This is where the actual data movement occurs during a transfer.

2.5 Peripheral Ports

Peripherals involved in high speed data traffic (McBSP, UTOPIA, master peripherals − discussed in individual sections, Video Port, EMIF, and L2 controller) have ports that accept commands from the TC as shown in Figure 3. Each includes read and write FIFO command and data buffers between the high speed EDMA engine and the peripheral, which may operate at some lower frequency. The ports receive TC commands and access the peripherals directly, freeing the EDMA to service other transfers while waiting for a response from the peripheral. This design allows transfers to/from different peripherals on different priority levels to occur simultaneously.

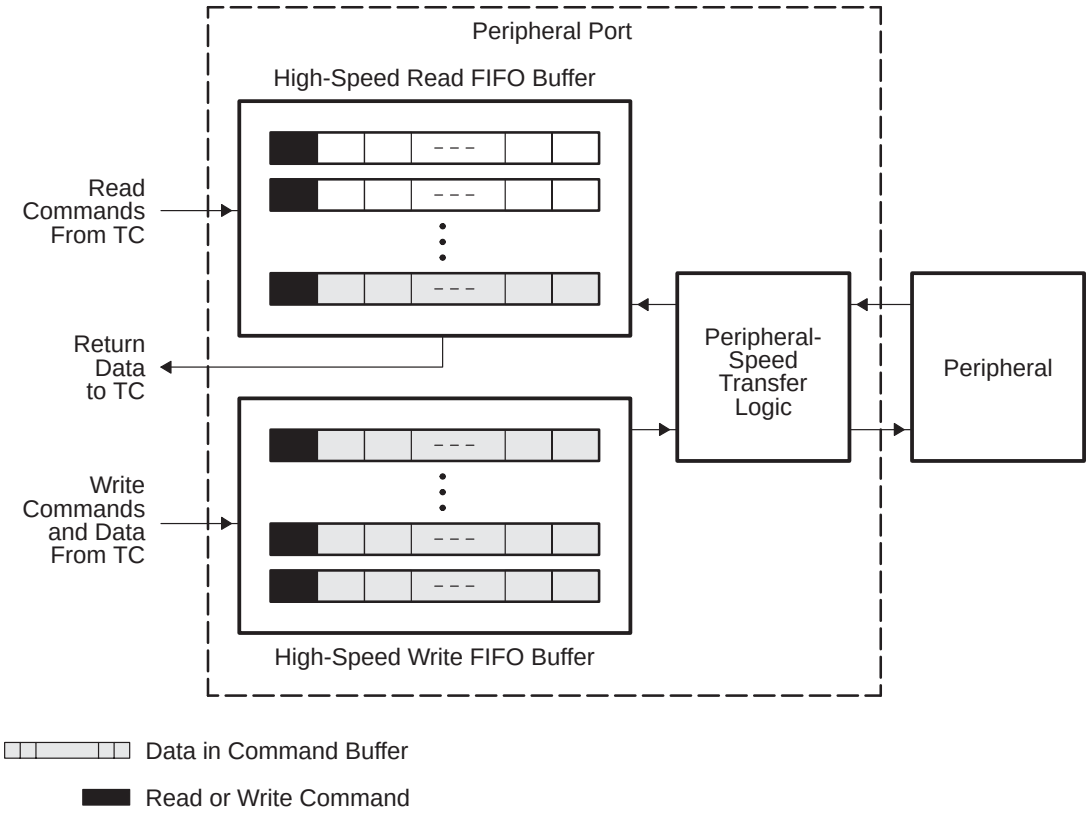

Figure 3. Peripheral Port Diagram

The number of command buffers in each peripheral port (as well as the default burst size of that port) is fixed in order to maximize efficiency.

Table 1. Command Buffers and Buffer Sizes

	Reads		Writes	
Peripheral†	**Command Buffers**	**Buffer Size (words)**	**Command Buffers**	**Buffer Size (words)**
L2 Memory Controller	8	2	8	2
TCP/VCP	8	2	8	2
McBSP 0/1/2	2	1	2	1
Utopia	2	16	2	8
EMIF A	4	16	4	32
EMIF B	4	4	4	8

† Peripheral availability varies by specific device. Refer to device data sheet.

Peripheral and EMIF ports service all commands in the order of their arrival. For example, suppose four read commands arrive followed by four write commands. The four reads are serviced followed by the four writes.

In contrast, the L2 port services reads and writes alternately, in the order of their arrival. Again, suppose four read commands arrive followed by four write commands. The first read is serviced, then the first write, followed by the second read, then the second write, and so on.

2.6 Transfer Controller Commands

To perform a transfer, the TC sends commands to source and destination ports for data to be read/written. These commands are for small bursts of data, which are less than or equal to the total transfer size of the submitted transfer request. The default burst size and the number of command buffers per port is shown in Table 1. The TC sends commands to the ports for data transfers, but the actual data movement doesn't occur until the port is ready. However, waiting for the port to become ready does not stall the TC. Therefore, if the different queues request transfers to/from different ports, the transfers can occur at the same time. Transfer commands made to the same port(s) are arbitrated by the TC according to priority.

To initiate a data transfer, the TC submits a command to the source or destination pipeline. There are three commands generated by the TC: pre-write, read, and write. Commands can be submitted to both pipelines once per cycle by any of the queue register sets. The TC arbitrates every cycle (separately for each pipeline) to allow the highest priority command that is pending to be submitted to the appropriate port. The pre-write command is issued to notify the destination that it is going to receive data. Once the destination has available space to accommodate the incoming data, it sends an acknowledgement to the EDMA that it is ready.

After receiving the acknowledgment from the destination, a read command is issued to the source port. Data is read at the maximum frequency of the source into the command buffer, and then passed to the EDMA routing unit to be sent to the destination. Once the routing unit receives the data, the data is sent along with a write command to its destination port.

Due to the EDMA's capability to wait for the destination's readiness to receive data, the source resource is free to be accessed for other transfers until the destination is ready. This provides an excellent utilization of resources and is referred to as write-driven processing. All write commands and data are sent from the EDMA to all resources on a single bus. The information is passed at the clock speed of the EDMA, and data from multiple transfers are interleaved based on priority when occurring simultaneously.

In this way, the EDMA transfer controller is services commands and data from multiple transfers simultaneously. Also, ports can service more than one active transfer if they have the bandwidth to do so, always giving precedence to the higher priority transfers. This especially useful for the L2 memory port which is the fastest port in the system.

The read data arrives on unique busses from each resource. This is to prevent contention and to ensure that data can be read at the maximum rate possible. Once the data arrives at the routing unit, the data that is available for the highest priority transfer is moved from its read bus to the write bus and sent to the destination port.

The queue register sets, command bus, and routing unit are depicted in Figure 4.

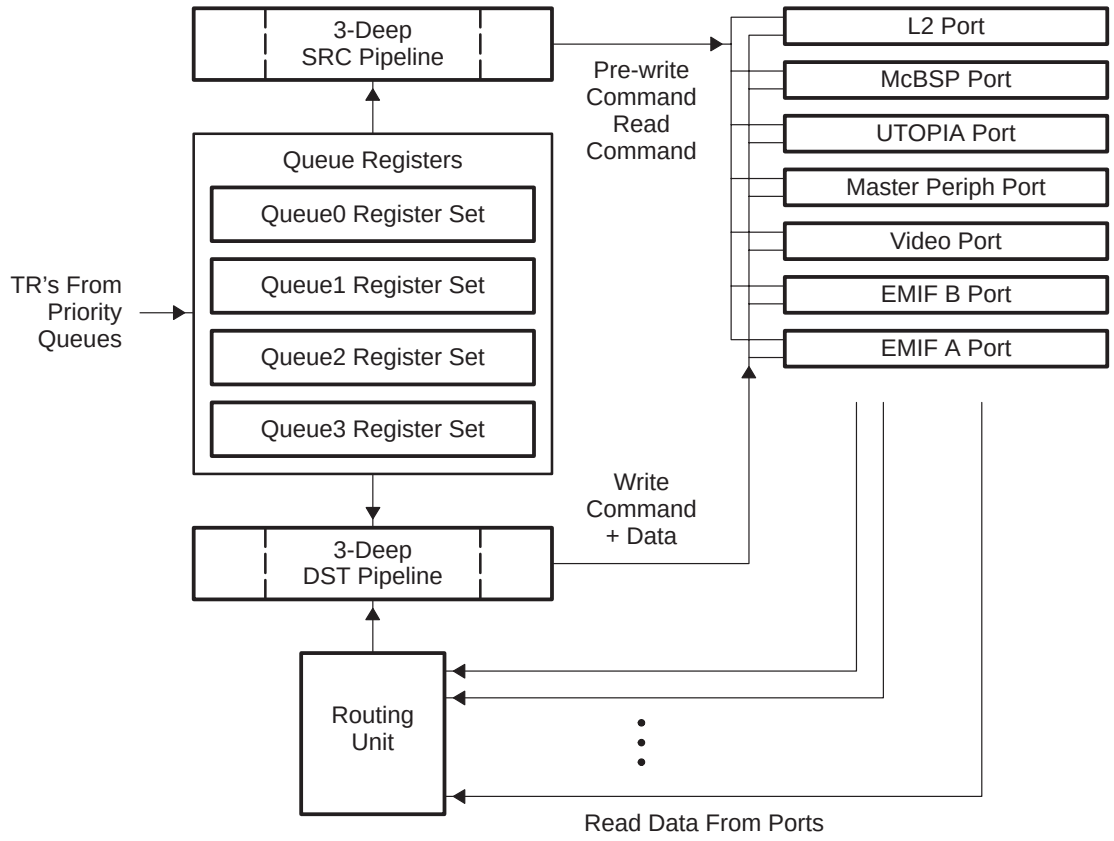

Figure 4. Command and Data Busses

3 Transfer Request Submission

Knowing how and when TRs are submitted is important to understand when scheduling data traffic in a system. The types of TRs submitted to the hardware differ slightly depending on the requestor, but all TRs contain the same essential information: source and destination addresses, element count, and the relationship between the elements within the source and destination regions (fixed, increment, decrement, or indexed.)

3.1 L2 Transfer Requests

The L2 transfer controller handles TR submission for CPU data accesses, L1 and L2 cache allocations from EMIF, L1 and L2 cache evictions to EMIF, and QDMA transfers.

3.1.1 CPU and Cache Transfer Requests

The L2 controller services CPU requests and maintains L2 cache coherency. The L2 cache is of programmable size, it can be disabled, and it resides between the CPU's L1 cache and the rest of the DSP's memory mapped space.

All cacheable memory spaces are serviced by the L1 cache and/or the L2 cache. If L2 cache size is zero (L2 cache is disabled), then cacheable memory is serviced by the L1 cache only. If L2 cache size is not zero (L2 cache is enabled), then cacheable memory is serviced by both L1 and L2 cache.

When determining system traffic, it is important to know when the L2 controller generates transfer requests. There are five basic CPU/L2 actions which trigger TR submission, listed above in section 2.2.1. However, one or more of these actions can be triggered based on CPU activity. To determine exactly what circumstances generate which TRs, refer to Table 2.

Table 2. EDMA/Cache Activity Due to CPU Accesses

Read/Write Destination	L1 Controller Action	L2 Cache Enabled/ Disabled	L2 Controller Action	Number of TRs Submitted	Number of Elements per TR
Internal registers	None	Don't care	None	0	–
Memory mapped control registers	Forward request to L2 controller	Don't care	Read from config bus – no EDMA action	0	–
L2 SRAM	Hit returns data; miss allocates one L1 cache line from L2 controller	Don't care	Read SRAM – no EDMA action	0	–
Non-cacheable EMIF	Forward request to L2 controller	Don't care	Submit TR to EDMA	1	1
Write access to cacheable EMIF	Hit, data lands in L1D; miss, data is passed onto L2	Disabled	Submit TR to EDMA	1	1
Read access to Cacheable EMIF	Hit returns data; miss allocates one L1 cache line from L2 controller	Disabled	Requests 1 L1 cache line from EDMA	1	L1 line size (64 bytes)
Cacheable EMIF	Hit returns data; miss allocates one L1 cache line from L2 controller	Enabled	Hit returns L1 cache line; miss allocates one L2 cache line from EDMA	2	½ L2 line size each (64 bytes each)

Note the two-level memory structure. A data request traverses the memory hierarchy until the data is found. The hierarchical data access sequence consists of the following:

1. The CPU requests data from the L1 controller.
2. The L1 controller checks L1 memory and requests data from the L2 controller if the data is not in L1.
3. The L2 controller checks L2 memory, and if the data is not in L2, requests data from the peripheral config bus or the EDMA (depending on the address range of the access.)
4. Data requests to the EDMA result in TRs.

Cache hits can reduce CPU wait states, and they have the added benefit of reducing EDMA traffic. For example, by using cache to access data in EMIF, the first request allocates one cache line from EMIF by submitting a TR. Subsequent hits to that cache line are returned quickly and no TR is issued. Furthermore, there is less EDMA latency when data is transferred in a block compared to transferring element by element.

Also note that a cache writeback to EMIF generates a single TR to write out modified data. These occur anytime there is no space in cache memory for a pending allocation, and the least recently used cache line contains dirty data.

For additional details on the two-level cache architecture of the TMS320C64x devices, including how to define memory spaces as cacheable, see the *TMS320C64x DSP Two-Level Internal Memory Reference Guide* (literature number SPRU610).

3.1.2 QDMA Transfer Requests

The L2 controller also submits TRs for QDMA transfers. QDMA transfers are initiated by writing to the QDMA pseudo-registers. Transfers are for simple block or frame transfers and take as little as one cycle to submit. For more information on QDMA transfers, consult *TMS320C64x DSP Two-Level Internal Memory Reference Guide* (literature number SPRU610).

3.2 EDMA Channel Transfer Requests

EDMA channels can be programmed to transfer data in a large variety of ways. Each channel is synchronized to a particular system event. One event corresponds to one TR submission, which transfers all or some of the data described by the parameter set. Due to the large number of configurations possible, the programming of an EDMA channel is not described in this document. For details see the *TMS320C6000 EDMA Controller Reference Guide* (literature number SPRU234). The amount of data transferred by a single TR is shown in Table 3.

Table 3. Data Transferred by an EDMA Channel Transfer Request

Source Dimension	Destination Dimension	Synchronization	Data Transferred by TR
1−D	1−D	Read/Write (FS=0)	1 element
1−D	1−D	Frame (FS=1)	Element count (one frame)
Other		Array (FS=0)	Element count (one array)
Other		Block (FS=1)	(Array count + 1) x element count

3.3 HPI Transfer Requests

The HPI controller submits TRs based on programmable actions performed by the host. To maximize the bandwidth available to host data transfers there are read and write FIFOs, each of which can contain eight 32-bit words. When possible, the HPI bursts multiple words between the HPI FIFOs and the physical memory. Table 4 describes the burst size of the TR submitted, depending on the host activity involved.

Table 4. Data Transferred by an HPI Transfer Request

Host Access	Situation	Data Transferred by TR
Non-auto-increment read	HPI reads HPID register	1 word
Non-auto-increment write	HPI writes HPID register	1 word
Auto-increment read	HPI reads HPID register and FIFO is empty	8 words
Auto-increment read	HPI reads HPID register, FIFO is less than or equal to half full, no outstanding TRs	4 words
Auto-increment write	HPI writes to HPID and data is the fourth element written since last TR issued	4 words

Data is transferred based on the host activity. If the host is performing individual accesses (accessing HPID in non-auto-increment mode), TRs are submitted for each individual element. If the host is performing burst transfers (accessing HPID in auto-increment mode) then TRs are submitted for multiple contiguous elements at a time. See the *TMS320C6000 DSP Host Port Interface Reference Guide* (literature number SPRU578) for additional information on the HPI, including a block diagram, register descriptions, pin listing, and waveforms.

3.4 PCI Transfer Requests

The PCI submits TRs based on actions performed by the PCI master, which could be an external host or the DSP. To maximize the bandwidth available there are 16 word read and write buffers for each type of access: master read, master write, slave read, and slave write. When possible (multiple word access to prefetchable memory), the PCI bursts multiple words between the FIFOs and the physical memory. Table 5 describes the burst size of the TR submitted, depending on the activity involved.

Table 5. Data Transferred by an PCI Transfer Request

PCI Access	Situation	Data Transferred by TR
PCI slave read/writes to non-Prefetchable memory	n/a	1 word
PCI slave writes to prefetchable memory	Slave write buffer is more then 1/4 full	4 words or transfer size (if <4 words)
PCI slave reads from prefetchable memory, first access	Slave read buffer is empty	16 words or transfer size (if <16 words)
PCI slave reads from prefetchable memory, subsequent accesses	Slave read buffer is less then 3/4 full	4 words or transfer remaining (if <4 words)
PCI master writes, first access	Master write buffer is empty	16 words or transfer size (if <16 words)
PCI master writes, subsequent accesses	Master write buffer is less then 3/4 full	4 words or transfer remaining (if <4 words)
PCI master reads	Master read buffer is more then 1/4 full	4 words or transfer remaining (if <4 words)

Data is transferred on the PCI bus may be of much larger burst sizes if the transfer buffers are serviced adequately. See the *TMS320C6000 DSP Peripheral Component Interconnect Reference Guide* (literature number SPRU518) for additional information on the PCI.

3.5 EMAC Transfer Requests

The ethernet media access controller (EMAC) module provides an efficient interface between the DSP core processor and the networked community via 10Base-T (10 Mbits/sec) or 100BaseTX (100 Mbits/sec) network connectivity. Transfer requests from the EMAC are submitted to the EDMA based upon packet movement between the DSP and the network.

The EMAC contains several input and output buffers to facilitate data flow. There are three input and three output buffers of 64 bytes each. This buffer size of 64 bytes (16 words) represents the maximum TR that the EMAC submits to the EDMA. For more information on the EMAC see *TMS320C6000 DSP Ethernet Media Access Controller (EMAC)/Management Data Input/Output (MDIO) Module Reference Guide* (literature number SPRU628.)

4 Priority Queue Allocation

To prevent any one transfer requestor from inundating the priority queues it is necessary to limit the number of outstanding TRs each register can submit. An outstanding TR is one that has been submitted to the priority queues and is awaiting processing. Once the TR enters the queue registers, it is no longer considered outstanding. For this purpose, each requestor has priority queue allocation register(s) to limit the number of TRs it can submit to each priority level. The default lengths allocated to the various requestors are listed below in Table 6. Keep in mind that the queue length per requestor is programmable, but the total queue length is not. The sum of all requestor lengths on a given priority must not exceed the total queue length of 16.

Table 6. Priority Queue Lengths

Queue	Priority	Total Queue Length	Requestor	Requestor Default Queue Length	Program Register Name
Q0	Urgent	16	L2 Controller/QDMA	6	L2ALLOC0
			EDMA Channels	2	PQAR0
			Master peripherals	0	TRCTL
				Sum = 8	
Q1	High	16	L2 Controller/QDMA	2	L2ALLOC1
			EDMA Channels	6	PQAR1
			Master peripherals	0	TRCTL
				Sum = 8	
Q2	Medium	16	L2 Controller/QDMA	2	L2ALLOC2
			EDMA Channels	2	PQAR2
			Master peripherals	4	TRCTL
				Sum = 8	
Q3	Low	16	L2 Controller/QDMA	2	L2ALLOC3
			EDMA Channels	6	PQAR3
			Master peripherals	0	TRCTL
				Sum = 8	

TR control registers (TRCTL) reside in the register sets of the master peripherals, and the exact name of the register may vary by peripheral.

For details on programming the priority queue allocation registers (L2ALLOCn, PQARn, TRCTL's) see the *TMS320C6000 DSP Peripherals Overview Reference Guide* (literature number SPRU190).

4.1 Transfer Requestor Stalls

If a requestor has submitted its maximum allotment of TRs for a given priority queue, its next TR of the same priority stalls the requestor. The stall is resolved when a TR on the priority level that caused the stall reaches the queue registers and is no longer outstanding.

If the L2 controller requestor experiences a TR stall, subsequent requests by the CPU will be stalled as well. For example, if the CPU submits three consecutive QDMA's to the medium priority queue (and L2ALLOC2 = 2), the L2 requestor is stalled until the first QDMA begins processing. While stalled, the L2 requestor cannot submit TRs on any priority for any action, including QDMA, CPU EMIF accesses, and cache accesses.

If the EDMA channel requestor is stalled, subsequent events do not generate TRs regardless of their priority. For example, if three low priority EDMA transfers are triggered consecutively (and PQAR3 = 2), the EDMA channel requestor is stalled. Subsequent events for any priority level cannot be submitted until the first low priority transfer begins servicing.

Note that if the EDMA channel requestor is stalled, the EDMA channel controller continues to receive synchronization events, and those events generate TRs once the controller is freed. Events are not lost during a stall unless the same particular event is received multiple times during the stall.

A TR stall scenario is depicted in the priority inversion section below. TR stalls severely inhibit efficient operation and must be avoided in a system.

4.2 Programming Priority Queue Allocation Registers

Care must be taken when programming the priority queue allocation registers for the three requestors (L2ALLOCn, PQARn, TRCTL's). This is to avoid both TR stalls and priority queue overrun (submitting more than 16 TRs to any one queue.)

Default values for the master peripherals' registers (TRCTL registers) are appropriate to handle activity from that master peripheral. It is generally unnecessary to change these values. The TRCTL registers contain bits for setting both the priority of transfers from the master peripheral and the queue allocation for that priority. Values for unused master peripherals can be ignored.

The allocation registers associated with EDMA and QDMA must be set appropriately in accordance with the amount of system traffic from their requestors. These limits should be programmed high enough to avoid stalling any requestor (especially for requestors that experience high traffic), but low enough so as not to exceed the total queue length (16) on any priority level.

For example, suppose priority queue Q1 is used for QDMA requests, EDMA channel requests, and HPI requests. Consider the three priority queue allocation registers associated with these transfer requests (L2ALLOC1, PQAR1, and the HPI's TRCTL.) Since the HPI's TRCTL register should retain the default allocation value of 4, there are 12 remaining spaces in the queue to be allocated between the QDMA and EDMA requestor. If the EDMA will experience more traffic on Q1 than the QDMA, PQAR1 may be set to 7 while L2ALLOC1 is set to 5.

Details on defining and scheduling system traffic can be found in *TMS320C6000 EDMA IO Scheduling and Performance* (literature number SPRAA00).

Avoid programming allocation registers to zero. Inadvertent TRs to zero-length queues will stall the requestor indefinitely.

5 Transfer Interaction and Arbitration

Knowing how multiple transfers interact once they are submitted is important for maximizing the performance obtained.

There are three places in the EDMA where transfers must be arbitrated: at the transfer requestor nodes, in the priority queues, and during active processing in the queue registers.

Arbitration at the requestor nodes is a simple matter. If a requestor is stalled, no TRs may be submitted. If requestors are not stalled, TRs are submitted in the order of their arrival. If TRs arrive simultaneously, they are submitted in a round-robin style. This has little significance as the delay is only 2−4 EDMA cycles per request.

In the priority queues, arbitration is also a fairly simple matter. TRs submitted to a single priority queue are processed serially in the order of their arrival. The priority queue is a simple FIFO in this respect. For this reason, long data transfers should not be placed on the same priority as short, time-critical transfers, because the short transfer could be queued behind the long transfer. If a large transfer is considered to be high priority, it is best to break the transfer up into multiple shorter bursts by using the linking or chaining capabilities of the EDMA. These capabilities are described in detail in the *TMS320C6000 DSP EDMA Controller Reference Guide* (literature number SPRU234). A specific example in Chapter 3 entitled “Breaking up Large Transfers with ATCC” can be found therein. The ATCC method neatly accomplishes this task by utilizing only one EDMA channel and parameter set.

By submitting multiple small TRs for one large transfer, the time-critical TRs (McBSP/HPI/etc.) can get in between the small TRs and not be queued for the full duration of the long transfer.

The final place TRs are arbitrated is in the queue registers. Each priority level has a queue register set which, when ready to service a transfer, fetches a TR from the head of its priority queue and begins processing.

The design of the queue registers and priority scheme makes the EDMA a very efficient transfer engine, capable of servicing up to four simultaneous transfers – one in each queue register set. Determining arbitration among the queue registers is generally a simple task. Commands are issued from the TRs to the peripheral ports. If two TRs attempt to utilize the same port, they are arbitrated by priority.

To make the EDMA more efficient, ports are only “busy” if they are actively transferring data. This is true for any port, but is mainly an advantage for the L2 port. Take the following example as a first approximation of how this occurs:

Assume that a TR is transferring data from the L2 memory to external memory on the EMIF. Since the L2 memory operates at a higher frequency than the EMIF, the L2 port will be “busy” only for short intervals during the transfer. This allows a TR that is servicing the McBSP to be serviced simultaneously, even though it utilizes the L2. Figure 5 illustrates this concept.

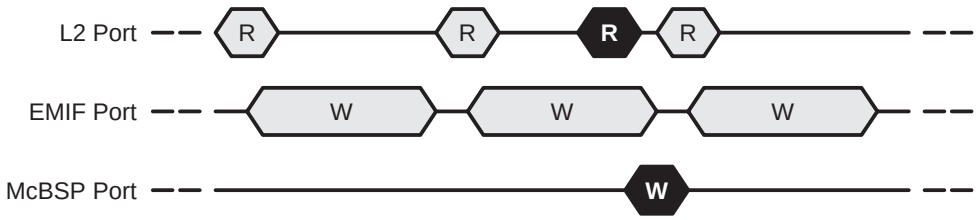

Figure 5. L2 Services Multiple Transfers

To expand on this example, remember that data is stored in the peripheral port’s command buffers in between these bursts. There are multiple command buffers in each port, and buffer size depends on the peripheral. The EDMA will burst data as long as there is space in the command buffers. The sizes of these command buffers are shown in Table 1.

Consider another transfer from the L2 port to EMIF-A on the TMS320C64x devices. The L2 port has 8 read command buffers of 2 words each. EMIF-A has 4 write command buffers of 32 words each. The activity at the ports is as follows:

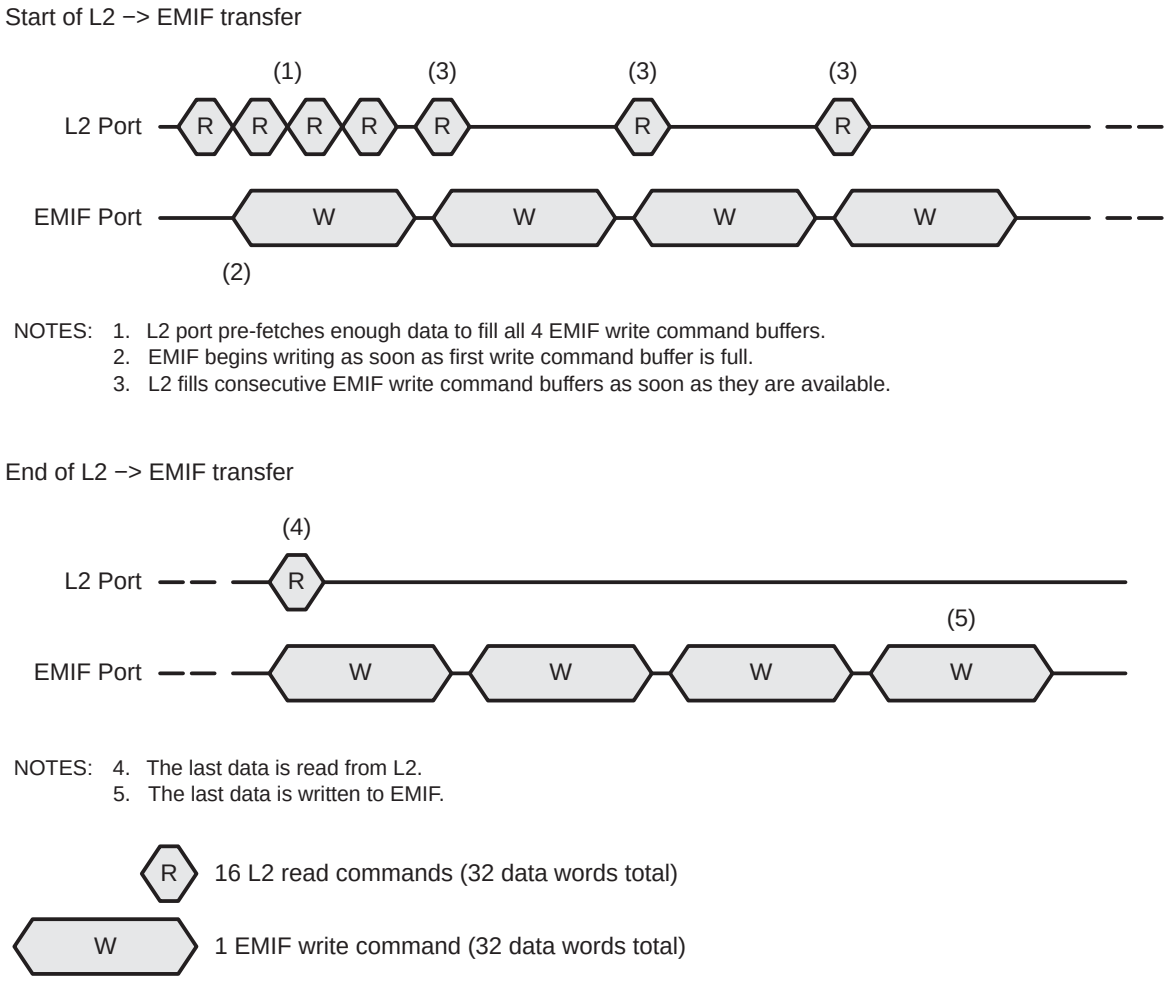

Figure 6. Port Activity is Determined by Command Buffers

The L2 port will be issued 16 read commands (of 2 words each). At this point, one full EMIF write command buffer will be filled (32 words), and the EMIF begins writing this data. At the same time, the L2 will still be reading data to fill up the remaining 3 EMIF write command buffers. This is the initial pre-fetch, during which time the L2 bandwidth is utilized fully.

After this pre-fetch, the L2 is utilized to keep the EMIF write buffers full. The L2 is only “busy” for a fraction of time for the remainder of the transfer. This fraction is roughly equal to the ratio of EMIF bandwidth to L2 bandwidth. Ideally, if the EMIF had a bandwidth of 800 MB/sec, and the L2 had a bandwidth of 2400 MB/sec, this transfer would only require about 33% of the L2’s bandwidth. This number is a very rough estimate, not accounting for data propagation delays and other factors, but it is a useful approximation.

At any time during this transfer, the L2 port may be diverted to the servicing of a higher priority transfer. However, because there is pre-fetched data stored in the EMIF command buffers, the L2 port may service the higher priority transfer and return to the EMIF transfer without interrupting EMIF data flow. By buffering data in the peripheral ports and utilizing write driven processing, the EDMA makes excellent utilization of resources and services multiple transfers efficiently.

6 Priority Inversion

Under certain circumstances, a low priority transfer appears to take precedence (as seen from the device pin perspective) over another transfer which the user intended to be of higher priority. This situation is known as priority inversion.

When defining types of priority inversion, the terms *low priority transfer* and *high priority transfer* are used to reference transfers of relative priority, rather than absolutely corresponding to transfers on priorities Q3 and Q1. For example, a transfer on priority Q0 (urgent priority) is a *high priority transfer* when compared to the relatively *low priority transfer* on Q2 (medium priority.)

There are four basic priority inversion scenarios:

6.1 Priority Inversion Due to Port Blocking

Priority inversion can occur when a port is in use for a low priority transfer. When a higher priority transfer request reaches the queue registers, it immediately has priority over the lower priority transfer to submit commands to the ports. However, because the ports are simple FIFO buffers, any remaining commands from the lower priority transfer must complete before the higher priority commands are serviced. Since commands are serviced in the order of their arrival, these commands can be from the read or write command buffers, or both.

From the pin perspective, it seems that the lower priority transfer is taking precedence, because the pins still service the transfer in progress until the buffered read/write commands are flushed.

Below is an example port blocking scenario. At some time before 0, transfer request A is active in the medium priority level, writing to EMIF-A (a 32-bit, 100 MHz SBSRAM.) It fills the EMIF-A peripheral port with write commands and data. The size of the write command buffer is determined from table 1 above.

Later, at time 0, transfer request B, which also writes to EMIF-A, arrives from priority queue 0. The destination pipe recognizes this and immediately gives commands from TR B priority. However, as illustrated at 640 ns and 1280 ns, EMIF-A will not begin servicing TR B’s write commands until TR A’s previously buffered commands are completed (flushed from the buffer.)

The delay depends on the operating frequency and bandwidth of the destination and the size and number of commands to be flushed. In this case, EMIF-A flushes 4 commands of 32 words each, which it takes a total of 1280 ns.

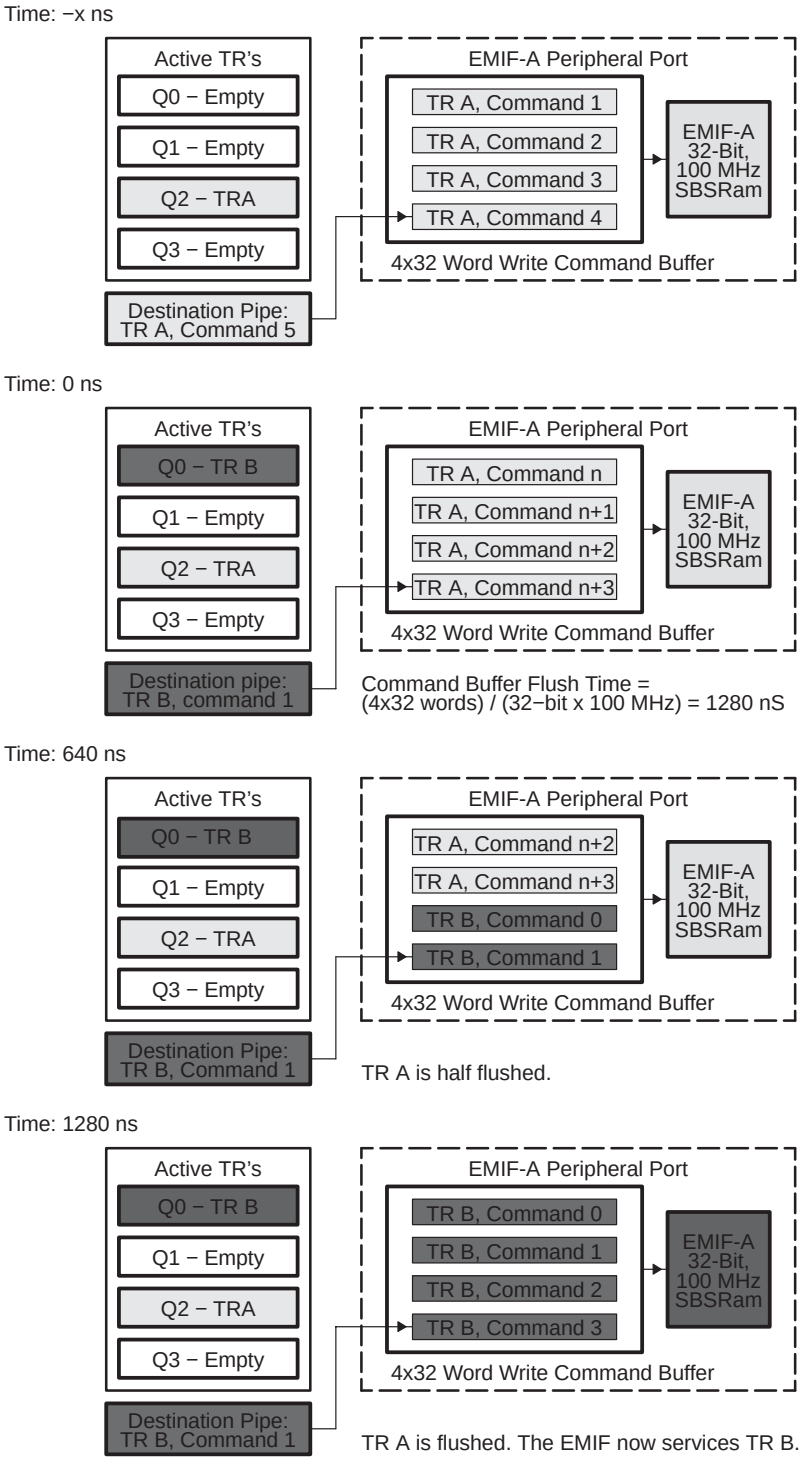

Figure 7. Example Port Blocking Scenario

This type of priority inversion is mainly a concern for EMIF memories which have the largest command buffers and longer flush times. The effects of this type of priority inversion can be minimized by proper system traffic scheduling.

6.2 Priority Inversion Due to Multiple High Priority Transfers

Priority inversion can occur if a high priority transfer is waiting for another transfer on the same priority level to complete. This is because there is only one set of queue registers per priority queue. While the high priority transfer is stalled in the queue, a lower priority transfer in progress could utilize resources that the pending high priority transfer is waiting to use.

From the pin perspective, it seems that the lower priority transfer is taking precedence. The status of the high priority transfer, stuck in the priority queue behind another high priority transfer in progress, is externally invisible.

Below is an example of priority inversion due to multiple high priority transfers. Assume that transfer request A is a relatively large, ongoing transfer in the transfer controller. At some later time, TRs B and C enter the queues, both intending to read from the same source. Even though TR C is of higher priority, TR B begins servicing first, because TR C must wait for TR A to finish processing in the Q0 queue registers.

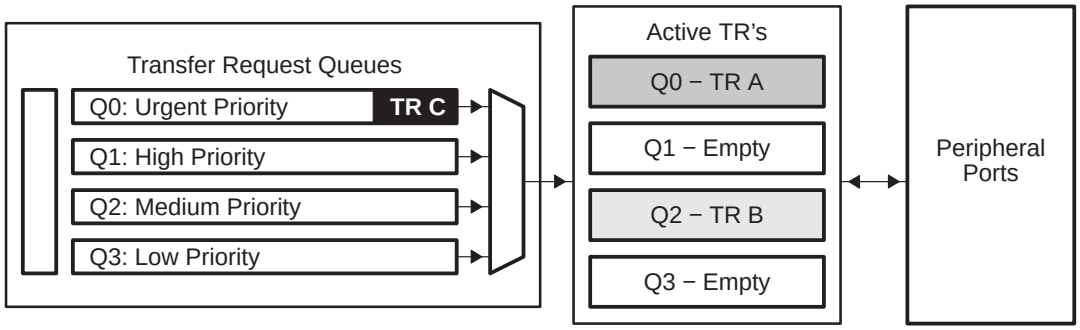

Figure 8. Example Multiple High Priority Scenario

This type of priority inversion is alleviated by properly scheduling transfers on all available priority levels, and by breaking up large, high priority transfers.

6.3 Priority Inversion Due to TR Stalls (TR Blocking)

Priority inversion can occur if a high priority requestor is stalled and therefore cannot submit a high priority TR. This case is especially applicable to the EDMA and L2 requestors. If the EDMA requestor is stalled (by submitting more than its allotment of TRs to a priority level), subsequent events on any priority are not serviced until the stall is resolved. Similarly, if the L2 requestor is stalled (by submitting more than its allotment of TRs to a priority level) subsequent L2 submissions are delayed.

From the pin perspective, it could seem that a lower priority transfer is taking precedence. The status of the stalled requestor is externally invisible (another reason stalls should be avoided.)

An example of priority inversion due to a transfer requestor stall is shown in Figure 9. The EDMA priority allocation register, PQAR2, is programmed to limit the number of TRs the EDMA channel controller can submit to the medium priority queue. As shown, the EDMA channel controller has attempted to submit a fourth TR to Q2, resulting in a stall. The high priority event on the bottom right is not serviced until the channel controller stall is resolved (when a Q2 TR enters the TC for processing.)

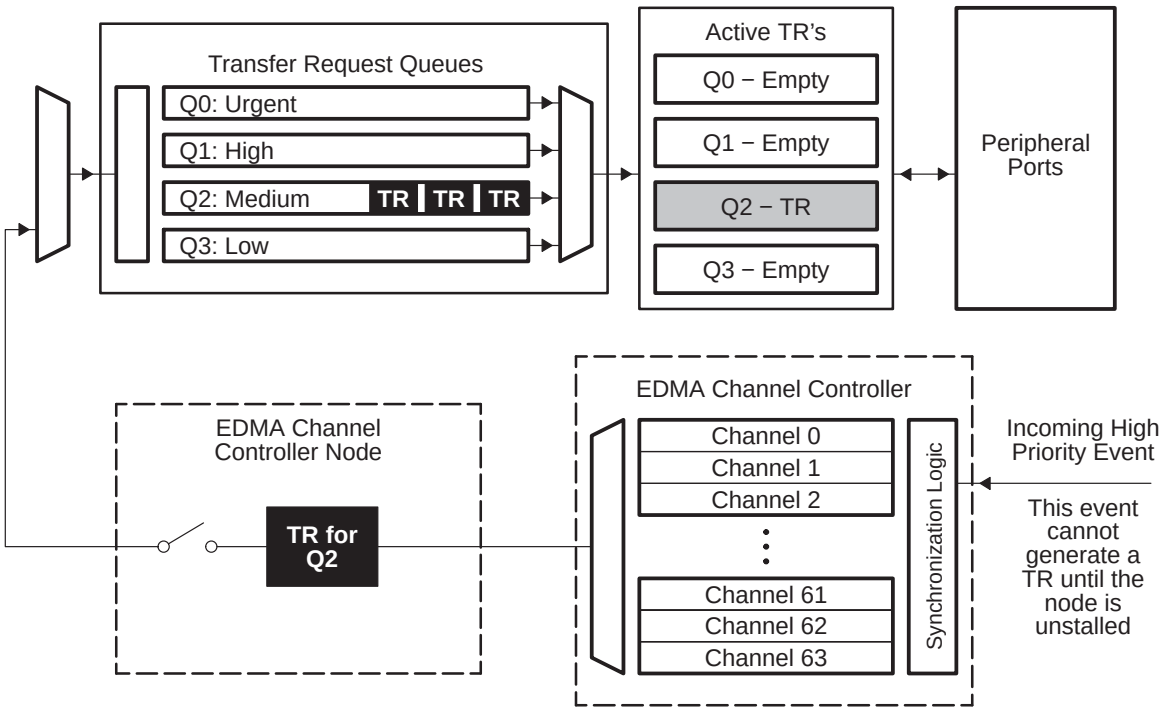

Figure 9. Example TR Stall Scenario

Note that the first event for each EDMA channel will not be lost, but multiple events to the same channel will only be counted once for the duration of the stall.

This type of priority inversion should be avoided by properly programming the priority queue allocation registers, thereby eliminating TR stalls.

6.4 Priority Inversion Due to Read/Write Parallelism

The read and write command pipelines in the EDMA transfer controller operate in parallel to maximize bandwidth and minimize transfer stalls. Unfortunately, this has a negative effect on the priority scheme in that, if a high priority transfer reads from a port and a low priority transfer writes to a port (or vice versa), the port receives both sets of commands. This is an issue because most ports have no priority decode mechanism so they simply service commands in the order of their arrival.

An example of priority inversion due to read/write parallelism is shown in Figure 10. Assume TR B is reading from EMIF-A on urgent priority, and TR A is writing to EMIF-A on medium priority. The source and destination pipelines submit commands separately, and the peripheral port simply services commands in the order of their arrival.

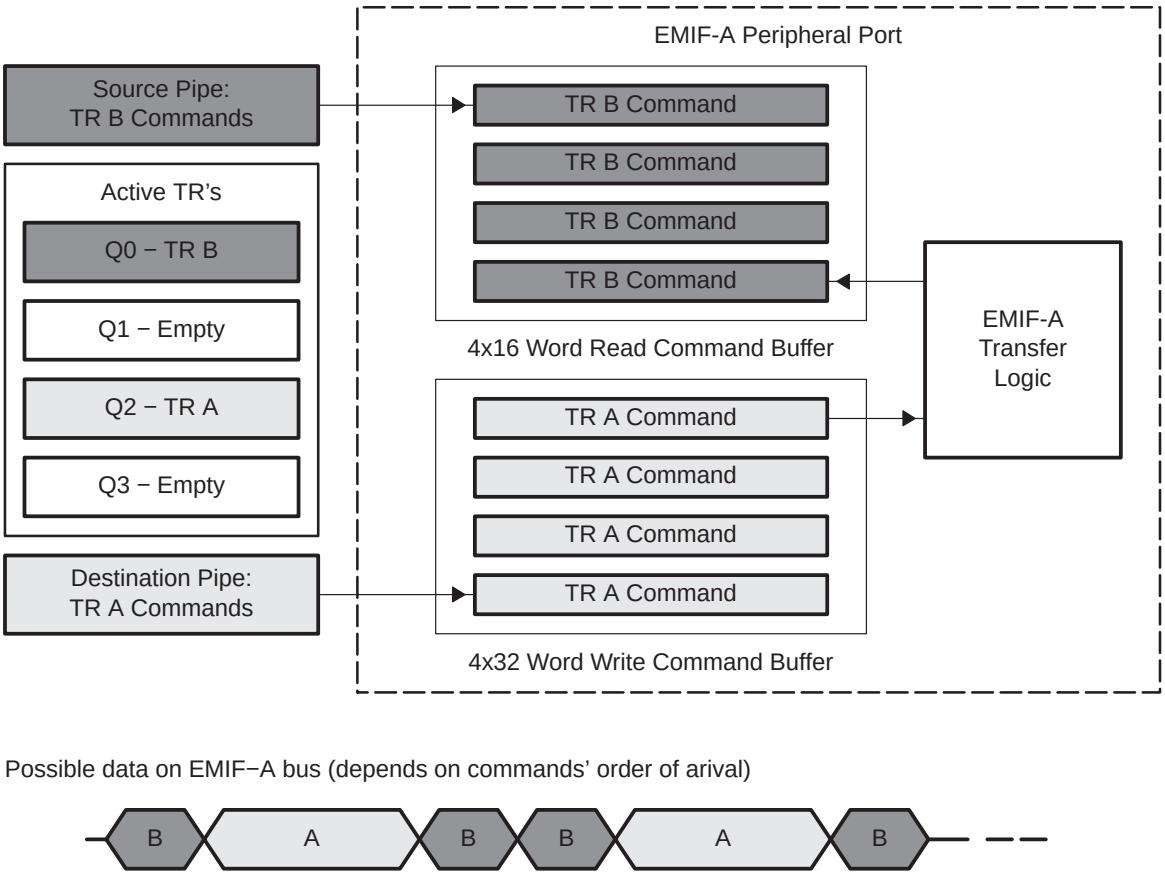

Figure 10. Example Parallel Read/Write Scenario

This type of priority inversion mainly affects memory ports which often service multiple medium to large transfers. However, the L2 port is generally much faster than the other ports involved and there is no noticeably degradation. This priority inversion can be avoided by properly scheduling transfers or adjusting priority levels.

7 Resolving Priority Inversion, TR Blocking/Stalls, and Port Blocking

Priority inversion, TR stalls/TR blocking, and port blocking scenarios can be minimized or even avoided completely by properly setting up traffic flow in a system. This is discussed in depth in *TMS320C6000 EDMA IO Scheduling and Performance* (literature number SPRAA00).

8 Conclusion

The TMS320C64x EDMA is a highly efficient data transfer engine. To be able to schedule system traffic, maximize bandwidth utilization and minimize transfer conflicts, it is important to understand the architecture of this transfer engine. Transfers start when a requestor (L2 controller, EDMA channel, HPI/PCI, EMAC) submits a transfer request for data to be transferred by the EDMA. These transfers can interact with one another at three different times: during submission by the requestor, within the transfer priority queues, and during active transferring within the priority queue registers. The first of the three has very little impact on performance, but the latter two can have an impact depending on the transfers' priority levels and properties. Transfer requests submitted to the same queue will be serviced in order. Active transfers (those in the queue registers) submit commands to the peripheral ports in order of priority. The ports have command buffers, and service commands in a FIFO manner. Because ports carry out transfers, the EDMA can service multiple requests simultaneously. By understanding the EDMA architecture, it is possible to maximize the data throughput and minimize the blocking of time-critical data transfers.

NOTE: This document is a revision of *TMS320C621x/TMS320C671x EDMA Queue Management Guidelines* (literature number SPRA720).

9 References

1. TMS320C6000 EDMA IO Scheduling and Performance (SPRAA00)
2. *TMS320C6000 DSP Peripherals Overview Reference Guide* (literature number SPRU190).
3. *TMS320C6000 EDMA Controller Reference Guide* (literature number SPRU234)
4. *TMS320C64x DSP Two-Level Internal Memory Reference Guide* (literature number SPRU610)

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