Application Report
SPRA720 - February 2001

TMS320C621x/TMS320C671x EDMA Queue Management Guidelines

David Bell

TMS320C6000 Applications

ABSTRACT

The enhanced DMA (EDMA) controller of the TMS320C621x and TMS320C671x devices is a highly efficient data transfer engine, controlling all of the data movement beyond the level-two memory of the device. There are sixteen channels and a quick DMA (QDMA) available to perform programmable data transfers, giving a great deal of flexibility. With this flexibility comes the responsibility to intelligently program the data movement efficiently. By understanding the underlying EDMA architecture and the means by which all transfers are performed, this task is not difficult to accomplish. It is important to know what types of transfers are performed by the EDMA, how they are requested, and how multiple transfers interact while being serviced. This knowledge allows transfers to be programmed so they take full advantage of the EDMA and obtain the maximum bandwidth available from the device.

Contents

List of Figures

List of Tables

1 Introduction

The enhanced DMA (EDMA) controller of the TMS320C621x and TMS320C671x devices is a highly efficient data transfer engine, capable of maintaining up to 1200 Mbytes per second (MB/s) of data throughput during operation. (The 1200 MB/s measurement assumes a clock rate of 150 MHz.) The EDMA handles all data movement between the level-two memory and the device peripherals, including cache-servicing, non-cacheable memory accesses, user-programmed data transfers, and host accesses. The EDMA architecture has many features designed to facilitate multiple data transfers simultaneously. With some knowledge of this architecture and the way in which data transfers are performed, it is possible to maximize the bandwidth utilization of the EDMA in any system.

2 EDMA Architecture

The most important thing to understand, prior to setting up the data movement of a system, is the architecture of the transfer engine. By understanding this architecture, it is possible to then understand the stages through which this transfer is accomplished. The architecture is the key to knowing how multiple transfers by multiple transfer requestors interact with one another, and ultimately impact the system performance. The EDMA is the transfer engine of the C6x1x devices (TMS320C6211, TMS320C6711, and TMS320C6712 devices). A block diagram of the EDMA is shown in Figure 1. This figure depicts the transfer requestors (level-two cache/memory controller, the EDMA channels, and the host port interface), the means by which they submit transfer requests to be performed (transfer request chain), and the transfer controller (transfer crossbar). It is important to note that the diagram is showing the transfer request structure, not the actual data transfer paths. Details on the data busses are provided later in the document. For additional architecture details, see the *TMS320C6000 Peripherals Reference Guide* (SPRU190).

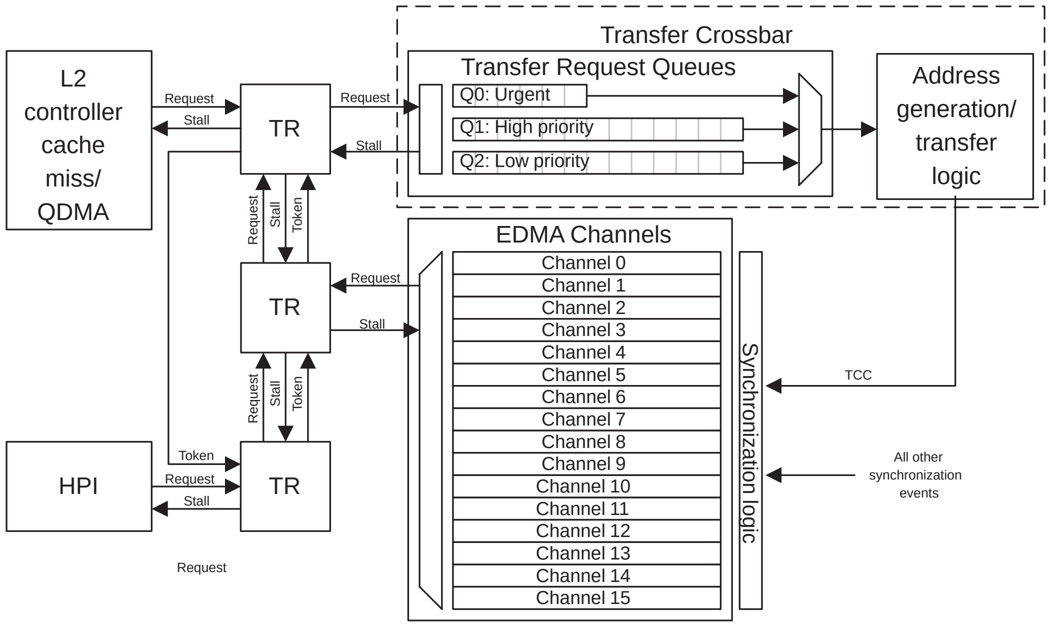

Figure 1. EDMA Architecture

2.1 Transfer Requestors

There are three requestors of data transfers inside the DSP: The level-two (L2) cache/memory controller, the EDMA channels, and the host port interface. Although the transfers requested are likely to be different due to the different tasks that each performs, the way each transfer request is handled by the EDMA is the same.

L2 Controller

The primary functions of the level-two (L2) cache controller are to manage all accesses to the L2 cache/SRAM, to maintain the coherency of the data within the cache, and to control the communication between the CPU and the rest of the chip (and the rest of the system via the on-chip peripherals). The transfers requested of the EDMA by the L2 are for cache misses, data flushes from the cache to its physical memory location, and accesses to non-cacheable memory. This functionality is not programmable, with the exception of defining non-cacheable memory and configuring portions of L2 as SRAM.

The CPU may initiate transfers as needed during system operation through a set of memory-mapped registers. The registers are referred to as Quick DMA (QDMA) registers, as the CPU is able to quickly dispatch a data transfer request without needing to configure a specific EDMA channel. The QDMA is essentially a seventeenth EDMA channel that is synchronized by the CPU. The parameters of the QDMA registers are identical to those of an EDMA channel's parameter set, with the exception that there is no element count reload, and no link address.

EDMA Channels

There are sixteen 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, since channels are orthogonal to one another. Each channel has a complete parameter set and does not rely on any shared resources. Once fully exhausted, the channel parameters may be reloaded (via the linking mechanism) with a new set that has been saved in the PaRAM.

The transfers requested by the EDMA channels are completely dependant on the configuration programmed by the user. Details on programming EDMA channels and the QDMA are not detailed in this document. For information and transfer examples see the *TMS320C6000 Peripherals Reference Guide* (SPRU190) or *TMS320C6000 EDMA: Applications Examples* (SPRA636).

HPI

Host port servicing is performed without any user intervention. The HPI has a direct connection to the EDMA and is not programmable by the user. The requests made to the EDMA are dependent on the host activity, but consist of transfers between the HPI data register (HPID) and the rest of the system memory. An EDMA channel invisible to the user is set aside for this task and is not configurable. The host port is capable of transferring data to or from any location in the DSPs memory map.

2.2 Transfer Request Chain

All transfer requestors to the EDMA are connected to the transfer request (TR) chain. A TR, once submitted, is shifted through the chain to the transfer crossbar (TC), where it is prioritized and processed. The TR can be submitted for a single data element or for a large number of elements, depending on the transfer required by the requestor.

The TR chain provides an inherent priority scheme to the requestors. Assuming each makes a submission on the same cycle, the requestor closest to the TC (downstream requestor) arrives first, and the farthest (upstream requestor) arrives last. Once a request is within the request chain, it has priority over new submissions, so the requests at the end of the chain do not get starved for servicing.

To prevent possible deadlock situations that would occur if a downstream requestor were held off from submission due to continuous submissions by upstream requestors, there is a round-robin token scheme implemented within the TR chain's logic. A token is passed around the TR chain (for the token, it is a loop) in the downstream direction. The transfer request node that has the token inverts the priority levels of its two requestors. Rather than giving priority to an existing request in the TR chain, located in the upstream node, priority is given to the local requestor to submit a new request. Although this is a safeguard implanted into the EDMA, the high bandwidth of the EDMA, relative to the speed at which requests are submitted, has shown this to be inconsequential.

2.3 Transfer Crossbar

Once a transfer request is at the end of the request chain, it is sent to the Transfer Crossbar (TC), shown in Figure 2. The TC is the portion of the EDMA that processes the TR. The TC includes separate read data busses from each peripheral on which data is read from a peripheral. A single global write bus is connected to all peripherals to send the transfer data to their destinations.

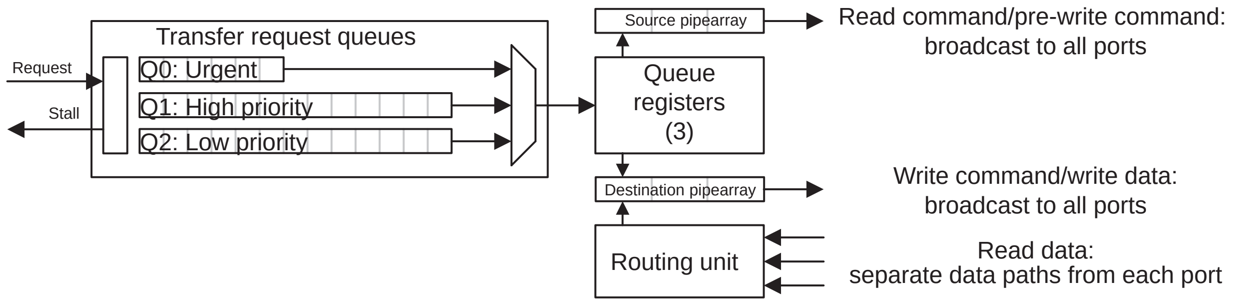
Transfer request queues
Request
Stall
Q0: Urgent
Q1: High priority
Q2: Low priority
Source pipearray
Queue registers (3)
Destination pipearray
Routing unit
Read command/pre-write command: broadcast to all ports
Write command/write data: broadcast to all ports
Read data: separate data paths from each port

Figure 2. Transfer Crossbar

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 three queues, corresponding to three priority levels:

- Urgent (Q0): reserved for cache service requests submitted by the L2 controller.
- High (Q1): used for host port servicing and high-priority EDMA transfers.
- Low (Q2): used for low-priority EDMA transfers.

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 queue can be serviced at a time by the address generation/transfer logic. The transfer logic can process TRs within separate registers, so multiple transfers can be going on at the same time. To maximize the data transfer bandwidth in a system, all three queues should be utilized.

There are three 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 devices' memory maps and are unavailable to the CPU.

The queue registers essentially function as a traditional DMA. They maintain the transfer parameters (source, destination, count, etc.) during the data transfer. These “channels” are essentially traditional DMA channels that are dynamically programmed by the DSP itself during device operation. The clear advantage to this architecture is that all transfers can be initialized after boot and will arrange themselves via the DSP hardware as needed. To get the most benefit out of this architecture, it is necessary to understand how these transfers take place.

The queue registers send requests for data to be transferred. These requests are for small bursts of data, which are less than or equal to the total data burst size of the submitted transfer request. The actual burst size, shown in Table 1, depends on the port performing the data reads or writes and is fixed by the hardware to maximize performance. This allows transfers initiated by different queues to occur simultaneously to one another. Because the registers send *requests* for data transfers, the actual data movement occurs as soon as the ports are ready. Therefore, if the different queues request transfers to/from different ports, the transfers can occur at the same time. Transfer requests made to the same port(s) are arbitrated according to priority. The burst size of the L2 memory is effectively infinite because the memory operates at the same frequency as the EDMA and is capable of being accessed on every cycle. The burst size is therefore set by the other peripheral involved in the data transfer or the total burst size originally submitted to the EDMA.

Table 1. TMS320C621x/C671x Peripheral Default Burst Sizes

	Default Burst Size	
Peripheral	**Read**	**Write**
L2 cache/memory	N/A (infinite)	N/A (infinite)
EMIF	4	16
McBSP	1	1
HPI	4	4

To initiate a data transfer, each queue register set submits its command request to the appropriate pipeline. There are three commands generated by the queue registers: pre-write, read, and write. Commands can be submitted to both pipelines once per cycle by any of the queue registers. The TC arbitrates every cycle (separately for each pipeline) to allow the highest priority command that is pending to be submitted. The pre-write command is issued to notify the destination that it is going to receive data. All ports have a small buffer available to receive a burst of data at the internal clock rate. Once the destination has available space to accommodate the incoming data, it sends an acknowledgement to the EDMA that it is ready.

After receiving the acknowledgment from the destination, a read command is issued to the data source. Data is read at the maximum frequency of the source and passed to the EDMA routing unit to be sent to the destination.

Once the routing unit receives the data, the data is sent along with a write command to its destination.

Due to the EDMA's capability to wait for the destination'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, as resources do not depend on one another, 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 interlaced when occurring simultaneously. Provided that multiple transfers (from different queues) have different sources, the transfers occur simultaneously.

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

3 TR Submission

Knowing how and when TRs are submitted is important to understand what "makes up" a TR and how it is processed by the hardware. 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.

L2 Transfer Requests

The L2 controller can make several different cache and data requests, always using the urgent priority (Q0). All requests are based on CPU and cache activity and are not programmable. There are four possible actions that trigger a TR submission:

- **Cache miss:** *Read in one cache line from the EMIF*

  Any time a miss occurs to the level-two cache, whether a read or a write, the L2 controller will issue TRs for one cache line to be read from its true location in external memory. To most efficiently service the cache miss, the line is transferred in two parts. For each L2 miss, two TRs are submitted. The first TR requests the portion of the line containing the missed data, and the second TR requests the remaining portion of the line. For a read miss, the missed data is passed to the CPU, allowing it to unstall, as soon as the data from the first TR is received. On a write miss, the data is written to its L2 location as soon as the data from the first TR is received. The L2 line size is 128 bytes. The TRs submitted on a cache miss correspond to those shown in Table 2.

Table 2. L2 Cache Miss TR Submissions

Missed Data Location	First TR Request	First TR Burst Size	Second TR Request	Second TR Burst Size
First ¼ L2 line	First ½ L2 line	64 bytes	Back ½ L2 line	64 bytes
Second ¼ L2 line	Back ¾ L2 line	96 bytes	Front ¼ L2 line	32 bytes
Third ¼ L2 line	Back ½ L2 line	64 bytes	Front ½ L2 line	64 bytes
Fourth ¼ L2 line	Back ¼ L2 line	32 bytes	Front ¾ L2 line	96 bytes

Note that if L2 is configured as all SRAM, then the cache miss is based on the L1 line size. For the level-one program **cache** (L1P) the line size is 256 bytes, and for the level-one data cache (L1D), the line size is 128 bytes. L1 cache lines are serviced with a single TR only. The lines are not broken into two parts.

- **Cache line eviction:** *Write out one cache line to the EMIF*

  When a cache miss occurs and a new L2 line is read from external memory, and the line in L2 to be replaced is dirty (the data has been modified), it must be evicted. In this instance, the L2 controller issues a TR to write out one full line to its location in external memory. The evicted line is not broken up into multiple sections to be transferred out, as the eviction is serviced most quickly by an unbroken burst. In the case of an L2 cache miss with an eviction, three TRs are submitted: one to write out the dirty cache line, and two to bring in the new line (as defined above).

  Note that if L2 is configured as all SRAM, then the eviction is based on the L1 line size. The eviction can only occur in the case of a read miss, as write misses do not result in a line fetch. In the case of an L1D read miss, two TRs are submitted: one to write out the dirty cache line, and one to bring in the new line. L1P cannot evict a cache line as the cache is read-only from the CPU.

- **Cache flush/clean:** *Write out multiple cache lines, as appropriate*

  There are a number of cache functions that are available to the CPU to maintain cache coherency with shared memory, clean the cache of old data, etc. Several of these functions are flush, clean, and writeback, all of which cause any modified data in the cache to be written back to their locations in external memory. For each of these functions, TRs are submitted for each cache line to be written back. Flush and clean commands are available for both the level-one and level-two caches.

- **Long distance access:** *Read or write one element from a non-cacheable location*

  External memory can be defined as a cacheable or non-cacheable region. Accesses to cacheable regions cause TRs to be submitted as defined in the first two bullets. Accesses to non-cacheable regions are performed individually with a single TR submission, and the TR is submitted for a single element only. For long distance reads, the CPU is unstalled as soon as the data value is retrieved. For long distance writes, there is a write buffer that temporarilty stores the write data while the TR is submitted. The EDMA reads the data from this write buffer and transfers it to its non-cacheable memory location. The buffer is four elements deep and allows the CPU to continue running without stalling.

For additional details on the two-level cache architecture of the C621x and C671x devices, see the *TMS320C6000 Peripherals Reference Guide* (SPRU190).

QDMA Transfers

QDMA transfer requests, while controlled by separate hardware, are submitted using the same TR node as the L2 controller TRs. From a functional point of view, the QDMA submission can be seen as a function of the L2 controller. A single TR on either Q1 or Q2 is submitted whenever a QDMA pseudo-register is written to. The writing of a QDMA pseudo-register is equivalent to an EDMA channel receiving a single synchronization event. QDMA transfers can be any frame- or block-synchronized transfer, using the same parameters as an EDMA channel.

EDMA Channels

EDMA channels can be programmed to transfer data in a large variety of ways. Each channel is synchronized to a particular system event. When an event arrives, one TR is submitted to transfer 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 Peripherals Reference Guide* (SPRU190) or *TMS320C6000 EDMA: Applications Examples* (SPRA636). The number of elements transferred by a single TR is shown in Table 3.

Table 3. Elements Transferred by EDMA Channel Transfer Request

Source Dimension	Destination Dimension	Synchronization	Elements Transferred
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) × element count

HPI

The HPI controller submits TRs based on the actions performed by the host. TRs are always submitted to the high-priority queue (Q1). To maximize the bandwidth available to host data transfers there are read and write FIFOs implemented in the HPI FIFO, 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. HPI Access Transfer Burst Sizes

Host Access	Situation	Words Transferred
Non-auto-increment read	HPI reads HPID register	1
Non-auto-increment write	HPI writes to HPID register	1
Auto-increment read	HPI reads HPID register and FIFO is empty	8
Auto-increment read	HPI reads HPID register, FIFO <= half full, no outstanding TRs	4
Auto-increment write	HPI writes to HPID and data is the fourth element written since last TR issued	4

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. To guarantee that data written by the host during a burst write always reach their destination, there is protection hardware built into the HPI. The HPI will always flush any remaining data left without a pending TR when the host terminates the write burst.

The HPI detects the conclusion of a write burst under several conditions. The host can write to a register other than the data register (either the HPIC or HPIA), the host can change transfer directions by reading from the HPID, or the transfer can timeout when the host is inactive for a particular amount of time. The HPI has an internal timer that counts the number of cycles between HPID accesses. If this count exceeds 128 CPU cycles, then the burst transfer is determined to be over and the data is flushed to its destination.

See the *TMS320C6000 Peripherals Reference Guide* (SPRU190) for additional information on the HPI, including a block diagram, register descriptions, pin listing, and waveforms.

4 Priority Queue Allocation

Knowing how much TR space is allocated for each queue can prevent stalls due to overflow. Because EDMA channels and QDMAs are programmable, the possibility exists for the EDMA to be inundated with a large number of requests simultaneously. To prevent a single requestor from using all of the EDMA resources, the number of outstanding TRs by these requestors is limited. For the high- and low-priority queues (Q1 and Q2, respectively) the number of outstanding TRs is shown in Table 5. This number does *not* include any TR currently present in the queue registers. The urgent priority queue (Q0) does not have any allocation restrictions, as its size is equal to the maximum number of L2 TRs that can be outstanding at any given time.

Table 5. EDMA and QDMA TR Limits

Transfer Requestor	TR Allotment
QDMA	3
EDMA	8

After the requestor has its maximum allotment of TRs in either priority queue, the next TR of the same priority is stalled until a space is free in the priority queue. For the QDMA, this means a CPU stall. The CPU is stalled until the TR is submitted, which occurs when the next QDMA TR reaches the queue registers. For the EDMA channels, a TR stall blocks other TRs from being submitted on either priority level. No TRs on either priority level are submitted until the next TR reaches the queue registers.

For illustration, suppose the CPU submits QDMA transfer requests one after the next on the same priority level, level 1. There is an active transfer already occurring on this priority level. The first three QDMA TRs get submitted to Q1 to await processing. When the CPU writes to a QDMA pseudo-register to submit the fourth QDMA transfer, there is a CPU stall. The CPU unstalls when the transfer currently in the queue registers completes, the first QDMA TR moves from the priority queue to the queue register, and the latest TR can be submitted.

To illustrate the same situation with the EDMA channels, suppose that there is a large transfer ongoing (in the queue registers) on priority level 2. Ten synchronization events all arrive within a short amount of time while the transfer is in progress. The first nine events received are for transfers using priority level 2, and the last event received is for a transfer on priority level 1. The first eight events will generate TR submissions as expected. Since the quota for level 2 submissions is reached, when the ninth event arrives the TR is not be submitted and the EDMA channel controller stalls. Since the controller stalls, even though the tenth event is for a transfer using priority level 1, it too stalls. The TRs generated by the final two events received are submitted when the current transfer being processed completes and the TR from the first event moves from the priority queue to the queue registers. Note that, although the EDMA controller stalls, the EDMA continues to receive synchronization events. System events are not lost during a stall unless the same particular event is received multiple times during the stall.

5 Transfer Arbitration

Knowing how multiple transfers interact once they are submitted is important for controlling the performance obtained. The interaction is different, depending on whether the TRs are on the same or different priority level. There are significant throughput benefits:

- Interaction within the same priority level: Effects of TR blocking by a long transfer

  TRs submitted on the same priority level are processed sequentially, with the first TR submitted to the system being processed first. Because of this, a TR for a large data transfer can potentially block other TRs of the same priority from being processed. It is for this reason that large transfers should not be requested on the same priority level as small time-critical transfers.

  Time critical transfers are typically those submitted by the EDMA channels to service serial port data, or TRs submitted by the HPI for host servicing. These transfers should be performed on Q1 (high priority). Small EDMA transfers may also be done on this priority level, as they do not block other TRs for long periods of time.

  All other transfers should be done on Q2. If a large transfer is considered a ”high” priority, it is possible to break the transfer up into multiple short bursts by using the linking and chaining capability of the EDMA. By submitting multiple small TRs for one large transfer, the time-critical TRs (McBSP/HPI/etc.) can get in between and not be starved for the full transfer time.

- Interaction between different priority levels: It is possible for transfers on different priority levels to occur simultaneously.

  There are separate queue register sets for each priority level. Each of these queue registers can contain an “active” TR—one that is currently being processed. The queue registers submit read and write requests to the DSP resources (L2 memory, EMIF, McBSPs, HPI, etc.) to transfer bursts of data. The data bursts are short (between 1 and 16 elements) and optimized for the resource and the direction. Because the queue registers issue burst requests to the resources, and the resources then provide the data as fast as they can, multiple active transfers can be ongoing at the same time.

  Each resource has a dedicated read bus from the resource to the routing unit. Data can arrive at the resource speed as fast as it becomes available. From the routing unit, there is a single global write bus providing data to the resources at the EDMA clock rate (faster than the peripheral rates). All peripherals have a built in FIFO, allowing them to receive the data bursts. Provided that the resources are different, data can be simultaneously transferred.

6 Conclusion

The C6000™ EDMA is a highly efficient data transfer engine capable of achieving and maintaining the device’s maximum data throughput. To ensure that the maximum bandwidth for system data transfers is achieved, it is necessary for the user to efficiently manage the queue resources available. Transfers are performed by a resource (L2 controller, EDMA channel, or HPI) submitting a transfer request for an element ,or group of elements, to be transferred by the EDMA. These transfer requests 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 depend on the priority levels on which the transfers are requested. Since there are different priority levels on which a transfer can take place, it is necessary to understand the underlying architecture to intelligently decide which to use for all programmable transfers. L2 controller and HPI submitted transfer requests occur on fixed priority levels that are not programmable. They are important to understand for efficiently programming the EDMA channels, and for QDMA to not disrupt this background servicing. While transfers submitted on the same priority level are transferred in the order they are submitted, transfers submitted on different priority levels occur simultaneously. With foresight into the EDMA architecture, it is possible to maximize the number of simultaneous transfers occurring throughout the device operation, and minimize the blocking of time-critical data transfers. Doing this allows the full device bandwidth to be achieved.

7 References

1. *TMS320C6000 Peripherals Reference Guide* (SPRU190).
2. *TMS320C6000 EDMA: Applications Examples* (SPRA636).