Application Report
SPRA614A - August 2000

TMS320C620x/TMS3206701 DMA and CPU: Data Access Performance

David Bell
Eric Biscondi

C6000 Applications

ABSTRACT

In a real-time system, data flow is important to understand and control 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. This application note provides relevant information for Texas Instruments (TI™) TMS320C620x devices (such as the 'C6201(B), 'C6202(B), 'C6203, 'C6204, 'C6205, and 'C6701). In general, one important guideline is to use the direct memory access (DMA) controller for background off-chip data accesses. The CPU should only be used for sporadic (non-periodic) accesses to individual locations, otherwise the system can incur performance degradation. Although the CPU and the DMA controller function independently of one another, when both are performing simultaneous data accesses it is necessary to properly schedule and configure them in order to minimize conflict and waiting while meeting real-time requirements. The timing information provided in this document 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. There are a number of guidelines that can be used to maximize available bandwidth.

Contents

List of Figures

List of Tables

1 TMS320C620x DMA and CPU: Data Access Performance

1.1 DMA Structure

The structure of the DMA is important to understand when incorporating it into a system. By knowing how the different DMA channels interact with one another, as well as the rest of the device, data flow will be easier to control. There was a structural improvement made for the 1.5V DMA for performance considerations, as outlined below.

1.1.1 1.8V and 2.5V TMS320C620x Devices[1]

The 1.8V/2.5V DMA consists of four independently programmable channels, as well as an auxiliary channel for servicing the host port interface. Each channel can access any portion of the memory map. Figure 1 shows a block diagram of this DMA.

1. Devices include the TMS320C6201(B), C6202, C6701

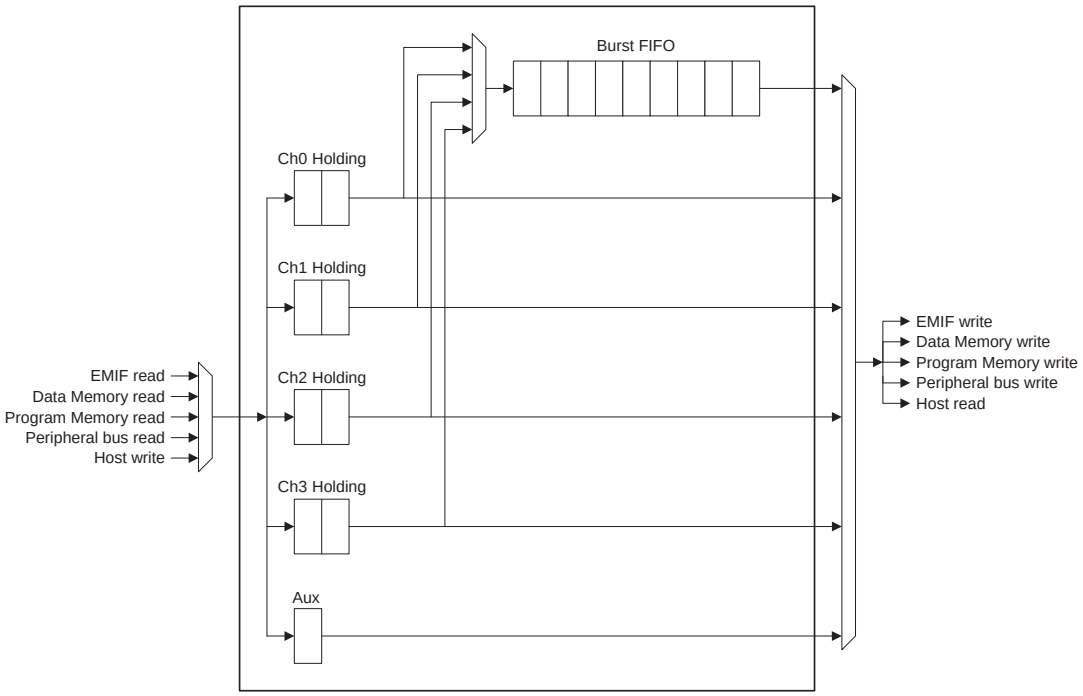

Figure 1. 1.8V/2.5V DMA Controller Data Bus Block Diagram

1.1.1.1 Holding Registers

Each DMA channel has a pair of holding registers, used to temporarily store the transfer element, so that the element is read from the source without being dependent on the destination. The holding registers function in one of two ways:

1. Split mode: The two registers serve as separate transmit and receive data stream holding registers for split mode. For both transmit and receive read transfers no subsequent read transfer request is issued until the associated write transfer request completes.
2. Non-split mode: The two registers serve as a 2-deep FIFO, in which a subsequent read can be issued without waiting for the associated write transfer to complete. However, because there are two holding registers, read transfers can only get one transfer ahead of write transfers.

1.1.1.2 Internal FIFO

A 9-deep FIFO holding path is provided to facilitate bursting for improved data rate by a DMA channel. This FIFO combines with the channel's holding registers to become an 11-deep FIFO. For a channel to obtain control of the internal FIFO, the following must be true:

- The channel does not have read or write synchronization enabled (frame synchronization can be enabled)
- 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 above conditions

A burst transfer by the DMA will be performed for each frame for unsynchronized and frame-synchronized transfers. Each frame is treated as a separate burst.

Since the 1.8V/2.5V DMA allows a higher priority channel to begin prior to the lower priority channel completing all pending writes, there is a potential for the higher priority channel to not gain access of the FIFO. When the source of the higher priority transfer is the same as the destination of the lower priority channel the FIFO is unable to flush its data. The priority scheme of the DMA considers the DMA channel number to determine which channel gets access of a resource. Since the higher priority channel owns the common resource, the low priority channel is unable to access it for its pending writes. The data remains in the shared FIFO, which prevents the higher priority channel from obtaining its use.

The impact of this is that the interrupting high-priority channel is not able to burst properly, since it does not have possession of the FIFO. Instead it uses 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 are transmitted at a time.

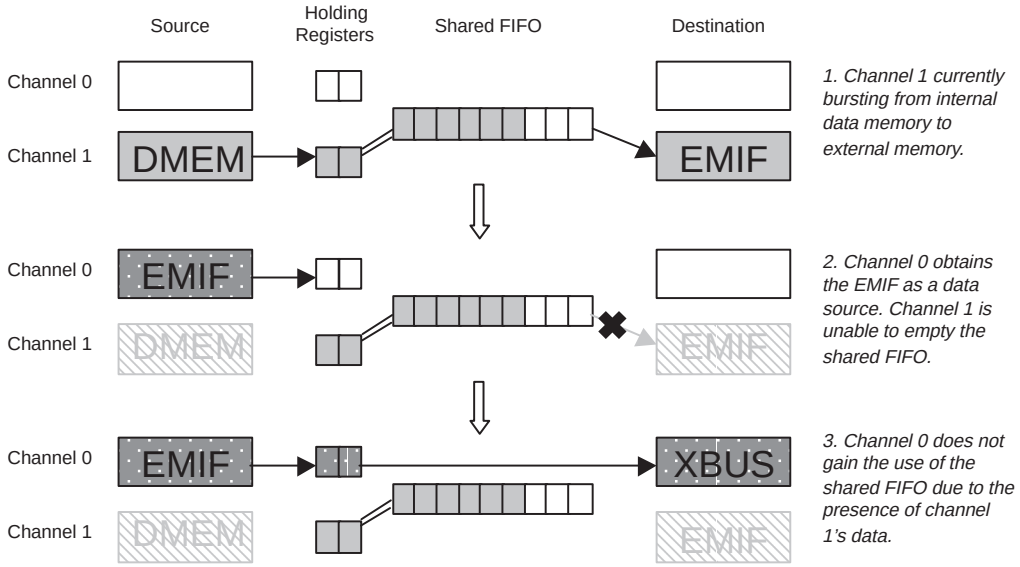

Figure 2. Shared FIFO Resource Problem

1.1.2 1.5V TMS320C620x Devices[2]

The structure of the DMA was redesigned for the 'C6203 (the first 1.5 V C6000 DSP) for 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 3 shows the internal data movement paths of the 1.5V DMA controller, including data buses and internal FIFOs.

2. Devices include the TMS320C6202B, 'C6203, 'C6204, 'C6205

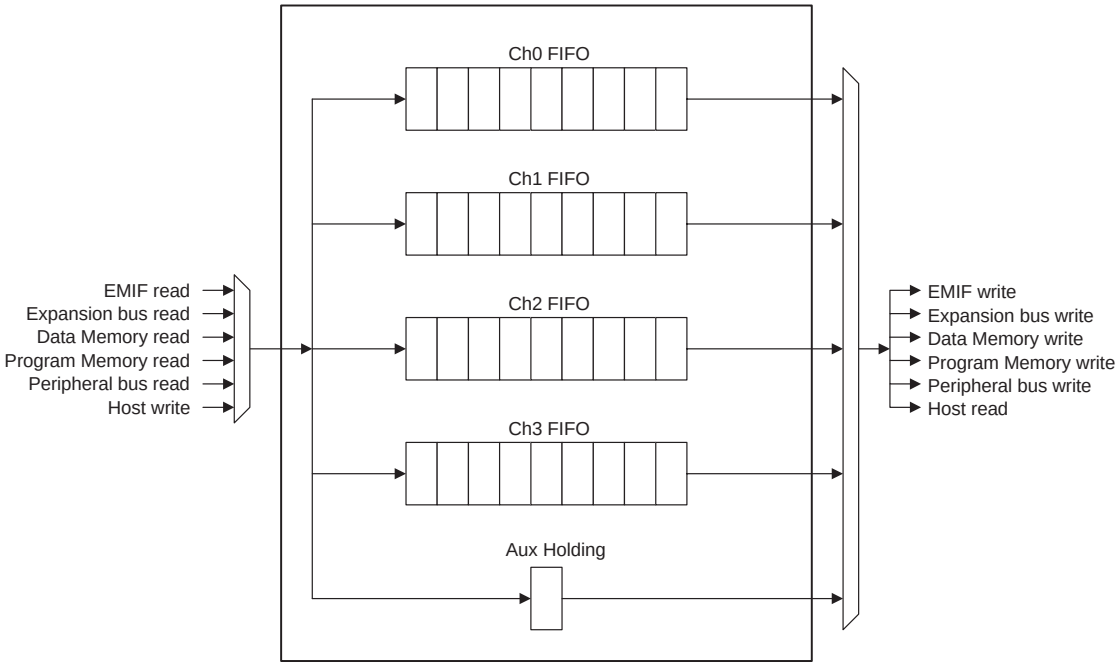

Figure 3. 1.5V DMA Controller Data Bus Block Diagram

1.1.2.1 Channel FIFOs

In these devices, each DMA channel has a dedicated 9-deep FIFO to facilitate bursting to high-speed memories. Each channel has a dedicated FIFO to reduce the arbitration required for switching between high-speed bursting channels. The individual operation by any channel is unchanged from that of the 1.8V/2.5V 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. The structure of the 1.5V DMA removes this bandwidth-limiting resource conflict that can occur with the 1.8V/2.5V 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 maintains its data within its FIFO until the higher priority transfer completes. When it once again gains access of its destination resource the transfer resumes.

In all other situations the behavior of the 1.5V DMA is identical to that of the 1.8V/2.5V devices.

1.1.2.2 Split Mode

When operating in split-mode, a 1.5V DMA channel behaves identically to a 1.8V/2.5V DMA channel. Only the first two cells of the channel's FIFO are used, effectively becoming the two holding registers.

1.1.3 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 Accessing Data

The data to be accessed by the CPU and the DMA can be located in internal data memory, on-chip peripherals, or in external memory. Whenever the CPU data paths and the DMA contend for any of these resources, arbitration determines the order in which the requests are serviced. Data Path A always has priority over data path B, and the priority level of the DMA with respect to the CPU is based on the priority (PRI) bit of the DMA channel's primary control register. This arbitration is valid for all resources. Figure 4 shows the CPU, data memory controller, and peripheral bus connections.

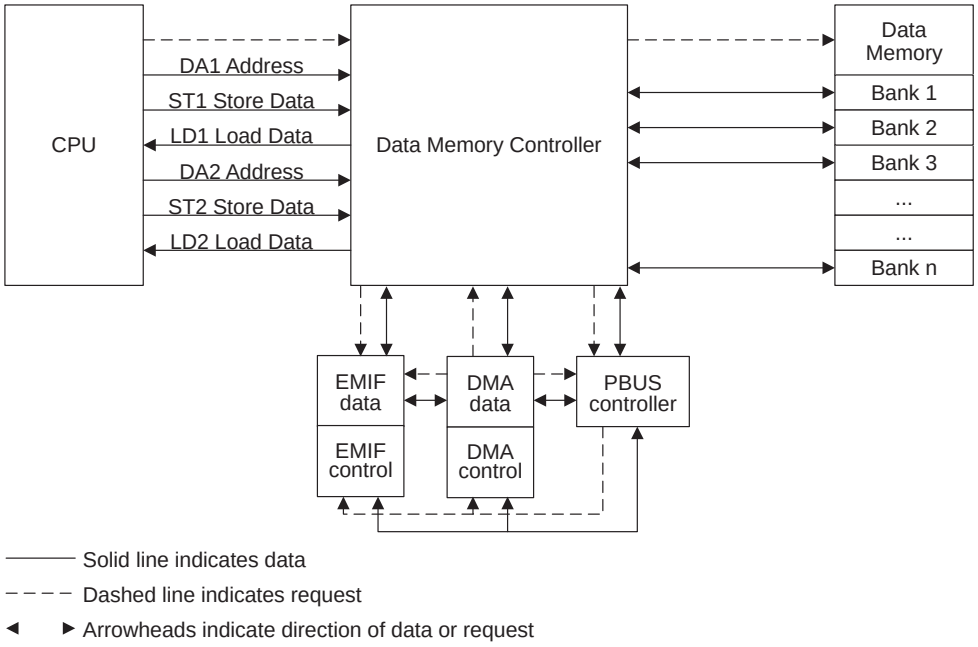

Figure 4. TMS320C620x Data Paths

2.1 Internal Data Memory

Internal data memory consists of high-speed SRAMs, which are divided into several 16-bit wide banks. Each bank can be accessed once per cycle, with distinct banks accessible in parallel. An access takes more than one cycle only if multiple requestors contend for the same memory bank. In this case a lower-priority requestor will be stalled until all of the banks it requests are free. Arbitration for each bank occurs every cycle. The physical arrangement, as well as the number, of data memory banks varies slightly between the DSPs.

2.1.1 TMS320C6201 Configuration

The internal data memory of the ’C6201 consists of four 16-bit wide banks. The DMA or CPU can access each bank once per cycle and multiple banks in the same cycle. The maximum data access each cycle is two 16-bit accesses and a 32-bit access.

2.1.2 TMS320C6201B/C6202(B)/C6203/C6204/C6205 Configuration

The internal data memory of the other ’C62x devices is modified to maximize the data accesses that can be performed each cycle by the three possible sources. Instead of four banks, the data memory consists of eight 16-bit wide banks. These are divided into two blocks of four banks, with the first four in the lower half of data memory, and the last four in the upper half. This configuration doubles the number of banks that can be accessed in a single cycle. With this new memory configuration, the maximum data access each cycle is three 32-bit accesses.

2.1.3 TMS320C6701 Configuration

The internal data memory of the ’C6701 consists of two blocks of eight 16-bit wide banks, allowing parallel 64-bit loads by the CPU in the same cycle as a data access by the DMA. With the new memory configuration, the maximum data access each cycle is two 64-bit CPU accesses (LDDW only) and a 32-bit DMA access.

2.2 Peripheral Bus

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

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

Table 1. CPU Stalls for Peripheral Register Accesses

CPU Access	CPU Stall
Single	5
Parallel	9

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

2.3 External Memory Interface (EMIF)

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

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

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 amount 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.

2.3.1 Memory Timings

The cycle timings for each memory type are in the appendix of this document, or 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 is specific to the type of memory. The beginning of the access is marked by the transition of the memory’s chip enable (/CE) to active (low).

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

Access times to asynchronous memory are user-defined using programmable setup, strobe, and hold values. Read and write accesses can use different settings for each field. For a complete description, see the *TMS320C6000 Peripherals Reference Guide*.

Table 2. EMIF Data Access Completion Timings in CLKOUT1 Cycles

Memory Type:	CE_READ_HOLD	CE_WRITE_HOLD
Asynchronous	7 –READ_HOLD	4 if WRITE_HOLD = 0 3 if WRITE_HOLD > 0
SDRAM	0	0
SBSRAM[3]	4	4

3. The numbers provided in this (and all other) table are for the 1/2x rate SBSRAM. The TMS320C6201(B) and 'C6701 devices also support a 1x SBSRAM interface, the numbers for which can vary by 1-2 cycles. Due to the complexity of providing the 1x SBSRAM interface at the maximum device frequency, and the fact that this mode is not commonly used, the 1x-specific timings are not included in this document.

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

The appendix contains the cycle timings for each external memory type, with reference points for all measurements in this document shown. The beginning and end points of the cycles correspond to the reference points in the datasheet for /CE transitions.

2.4 CPU Accesses

The CPU is accesses data in external memory using the load and store operations. 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 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. CE_HOLD values (CE_READ_HOLD or CE_WRITE_HOLD) are provided in Table 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 can have to wait until the appropriate phase of the external memory clock, which is half the rate of the CPU clock.

Table 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 increases if the EMIF is currently completing a previous access. The number of cycles of additional delay depends on whether the CPU has a higher priority than the current access, as well as how close to completion the access is. 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.

2.4.1 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.

2.5 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 4. These switching times are valid for CPU/CPU, DMA/CPU, CPU/DMA, and DMA/DMA access boundaries.

Table 4. External Switching Time between Accesses by Different Requestors

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

The switching times signify the number of CLKOUT1 cycles between the end of one access and the beginning of another. Within this document the beginning of an access is defined to be the rising edge of CLKOUT1 identified in the data sheet to reference the /CE transition from 1 to 0. The end of an access is defined to be the rising edge of CLKOUT1 used to reference the /CE transition from 0 to 1. Memory cycle diagrams are provided in the appendix with the beginning and end of all memory accesses marked.

2.5.1 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 4, depending on whether one channel interrupts another, or one channel completes and a suspended channel resumes (or begins). The switching time between accesses by different channels is equal to the values given in Table 5 plus an additional offset shown in Table 5.

Table 5. Additional Switching Time between External DMA Accesses

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

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

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

2.5.2 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.[4]

2.5.2.1 Multiple Frames and Auto-initialization

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

Once a frame or block boundary is reached, the current burst ends and a new burst begins. As shown in Table 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 6. CLKOUT1 Cycles between External Frame Bursts

Memory Type	Cycles between reads	Cycles between writes
Asynchronous	1	1
SDRAM	16	10
SBSRAM	4	4

2.5.2.2 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 7 shows the time between memory accesses (/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 7. Burst Interruption by McBSP/Host Service

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

4. External accesses by the DMA auxiliary channel act as a DMA channel interrupting another DMA channel, described previously.

2.6 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 8 shows the time delay from a synchronization event to the beginning of a data access.

Table 8. DMA Synchronization Timings

	CLKOUT1 cycles for Memory Type:		
Timing	**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[5]	16	20	17
WSYNC_STAT bit set to beginning of external write	7	11	8

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

2.7 Transferring to/from the 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. Instead it must read several elements, then write several elements.

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

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

Table 9. Burst Size for Shared Resource

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

6. 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 10 lists the number of cycles between reads and writes for transfers between external memories, in addition to those provided in Table 4.

Table 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 11.

Table 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 in and the latencies in Table 10 and Table 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 Bandwidth Calculation

If the system activity is known, then you can derive the total bandwidth required by the system from the information presented in this document. Such an analysis allows a designer to know that all data I/O is properly performed, with no missed samples.

3.1 Simple Example

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

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

First you should find out the bandwidth requirements for the individual data streams. describes all of the timing parameters used in the calculations, along with their location in this document.

Table 12. Simple Example Timing Parameter Descriptions

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	Figure A–3, p. 25	Cycle count from the beginning of the access to the beginning of the first read data phase.
CE_read_hold	4	Table 2, p. 10	Cycle count from the last strobe in a read burst from SBSRAM to the end of the access
CE_write_hold	4	Table 2, p. 10	Cycle count from the last strobe in a write burst to SBSRAM to the end of the access
read_frame_gap	4	Table 6, p. 13	Time between read bursts for multi-frame transfer
read_to_write	6	Table 4, p. 11	Switching time between a SBSRAM read access and a SBSRAM write access
DMA_hp_read_to_lp_write	15	Table 5, p. 12	Additional switching time between a high priority read access and a low priority write access
start_to_sync	1	Table 8, p. 14	Time from setting START = 01b to the setting of RSYNC_STAT
RSYNC_STAT_to_read	11	Table 8, p. 14	Latency from the setting of RSYNC_STAT to the 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

3.1.1 Channel 0 Burst Time:

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

$2 \cdot (CE\_read\_setup + 2 \cdot element\_count0 + CE\_read\_hold)$

$= 2 \cdot (4 + 2 \times 20 + 4) = 96$ cycles

3.1.2 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

3.1.3 Channel 1 Burst Time:

DMA channel 1 performs only a single burst, which requires the following number of cycles to complete:

2 · element_count1 + CE_write_hold

= 2 x 40 + 4 = 84 cycles

3.1.4 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

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

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

Table 13. Complex Example Timing Parameter Descriptions

Parameter	Value	Location	Description
element_count	128	N/A	Number of elements per frame
Setup	2	Memory Timings, p. 9	Read/write setup time (same in this example)
Strobe	4	Memory Timings, p. 9	Read/write strobe time (same in this example)
Hold	1	Memory Timings, p. 9	Read/write hold time (same in this example)
CE_read_hold	6	Table 2, p. 10	Cycle count from the last strobe in a read burst from asynchronous memory to the end of the access
CE_write_hold	3	Table 2, p. 10	Cycle count from the last strobe in a write burst to asynchronous memory to the end of the access
mcbsp_read_interruption	12	Table 7, p. 13	Burst interruption caused by a McBSP read
mcbsp_write_interruption	14	Table 7, p.13	Burst interruption caused by a McBSP write
write_to_read	1	Table 4, p. 11	Switching time between an asynchronous write access to an asynchronous read access
read_to_write	2	Table 4, p. 11	Switching time between an asynchronous read access to an asynchronous write access
DMA_lp_write_to_hp_read	4	Table 5, p. 12	Additional switching time between a low priority write access and a high priority read access
DMA_hp_read_to_lp_write	15	Table 5, p. 12	Additional switching time between a high priority read access and a low priority write access
RSYNC_STAT_to_read	9	Table 8, p. 14	Latency from the setting of RSYNC_STAT to the beginning of a read access

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

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

3.2.1 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.

3.2.2 Input Data Transfer Time:

The parallel input data requires the following number of cycles every 2000 cycles:

(setup + strobe) · element_count + hold · (element_count – 1) + CE_read_hold

= (2 + 4) · 128 + 1 · (128 – 1) + 6 = 901 cycles.

3.2.3 Output Data Transfer Time:

The parallel output data requires the following number of cycles every 3000 cycles:

(setup + strobe) · element_count + hold · (element_count – 1) + CE_write_hold

= (2 + 4) · 128 + (1) · 127 + 3 = 898 cycles

3.2.4 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.

3.2.5 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 insures 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 this example, transfer priority is ranked according to frequency. Based on the numbers shown in the timeslice calculation the priority ranking is as follows:

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.

3.2.6 McBSP Data Transfer Overhead:

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

2 · (CE_read_hold + mcbsp_read_interruption) for the McBSP reads and

2 · (CE_write_hold + mcbsp_write_interruption) for the McBSP writes

= 2 · (6 + 12 + 3 + 14) = 70 cycles every 12000 cycles.

3.2.7 Input Data Transfer Overhead:

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

4 · (CE_write_hold + write_to_read + DMA_lp_write_to_hp_read)

= 4 · (3 + 1 + 4) = 32 cycles every12000 cycles.

3.2.8 Output Data Transfer Time:

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

6 · (read_to_write + DMA_hp_read_to_lp_write)

= 6 · (2 + 15) = 102 cycles every 12000 cycles.

3.2.9 Total Bandwidth Utilization:

Again, the total bandwidth required by the system is:

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

= ((6 x901) + (4 x898) +70 + 32 + 102) / 12000 · 100%

= 9202 /1200 · 100%

= 77%

3.2.10 Device Considerations 1.8V/2.5V vs. 1.5V:

For the 1.5V C6000 devices, 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 1.8V/2.5V devices, however, some additional steps must be taken due to the shared FIFO present in the DMA. As described section 1.1.1.2 Internal FIFO, 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). In 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:

$6 \cdot (\text{RSYNC\_STAT\_to\_read})$

$= 6 \cdot 9 = 54$ 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 {sync event} to the setting of RSYNC_STAT) within 2846 / 6 = 475 cycles[7].

4 Bandwidth Optimization

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

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

4.1 Maximize DMA Bursts

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

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

7. 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.

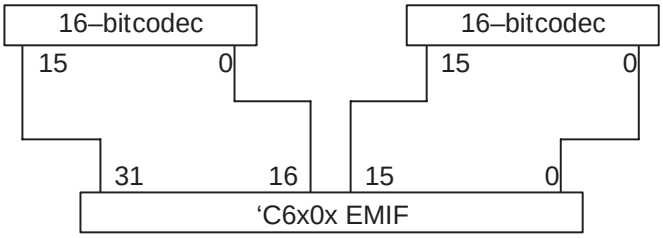

Figure 5. Combining External Peripherals

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 6. When an odd 16-bit data element arrives, latch it into one halfword on the data bus. When an even 16-bit data element arrives, access both elements with a 32-bit transfer. Thus, the bandwidth is effectively doubled.

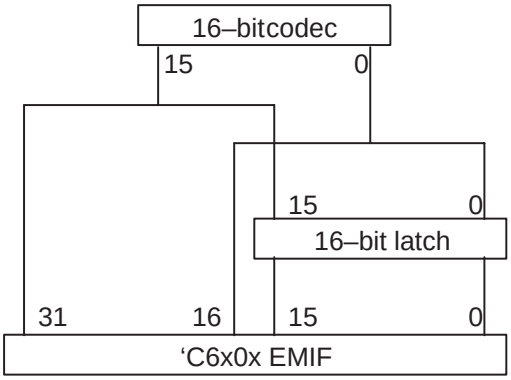

Figure 6. Converting a 16-bit Peripheral to 32-bit

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

4.2 Minimizing CPU/DMA Conflict

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

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

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

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

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

5 Conclusion

In any real-time system, efficient use of data resources is a critical issue. In order to schedule data accesses by multiple requestors, you should understand not only how long a data request takes to complete, but also how the different transfers interact with one another. By understanding the timings involved with accessing memory, switching between requestors for a data resource, and synchronizing data transfers, maximizing the efficiency of the 'C6000 system will become an achievable task. Using the data provided in this document allows for an analysis to be done on the data I/O for a system. With knowledge of the I/O requirements of the system, as well as the performance capability of the DSP, it is possible to optimize the performance of the DMA and CPU.

Appendix A External Memory Access Timings

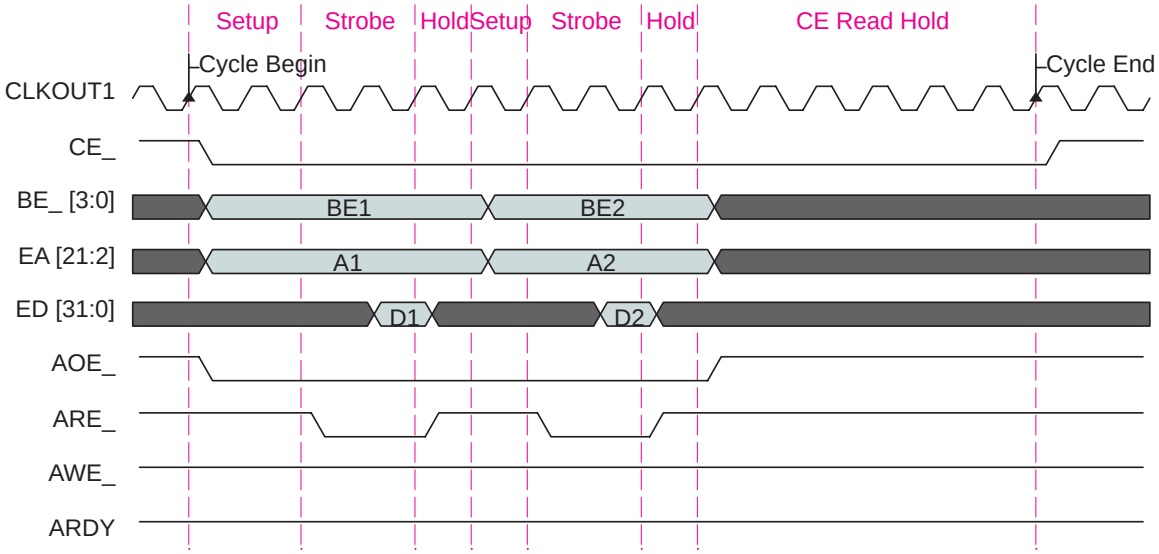

Figure A–1. Asynchronous Memory Read Cycle Timings (Setup = 1, Strobe = 2, Hold = 1)

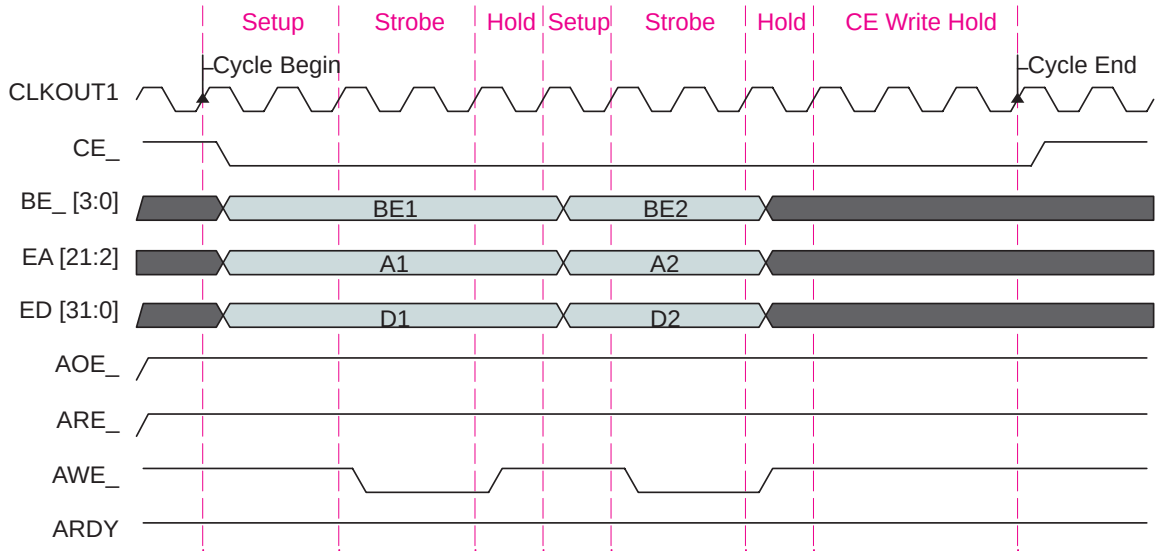

Figure A–2. Asynchronous Memory Write Cycle Timings (Setup = 1, Strobe = 2, Hold = 1)

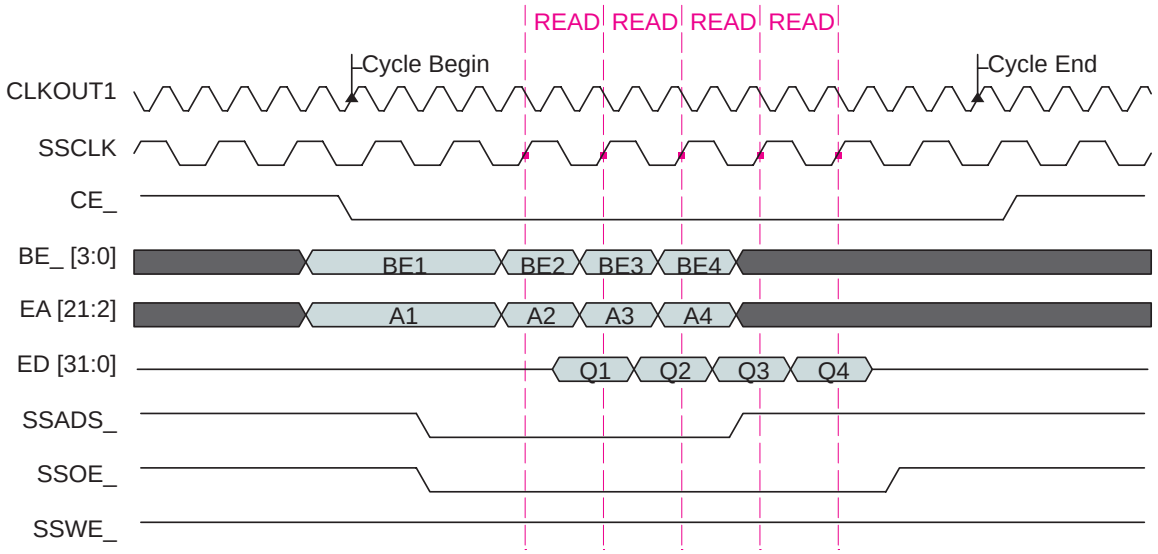

Figure A–3. 1/2x Rate SBSRAM Read Cycle Timings

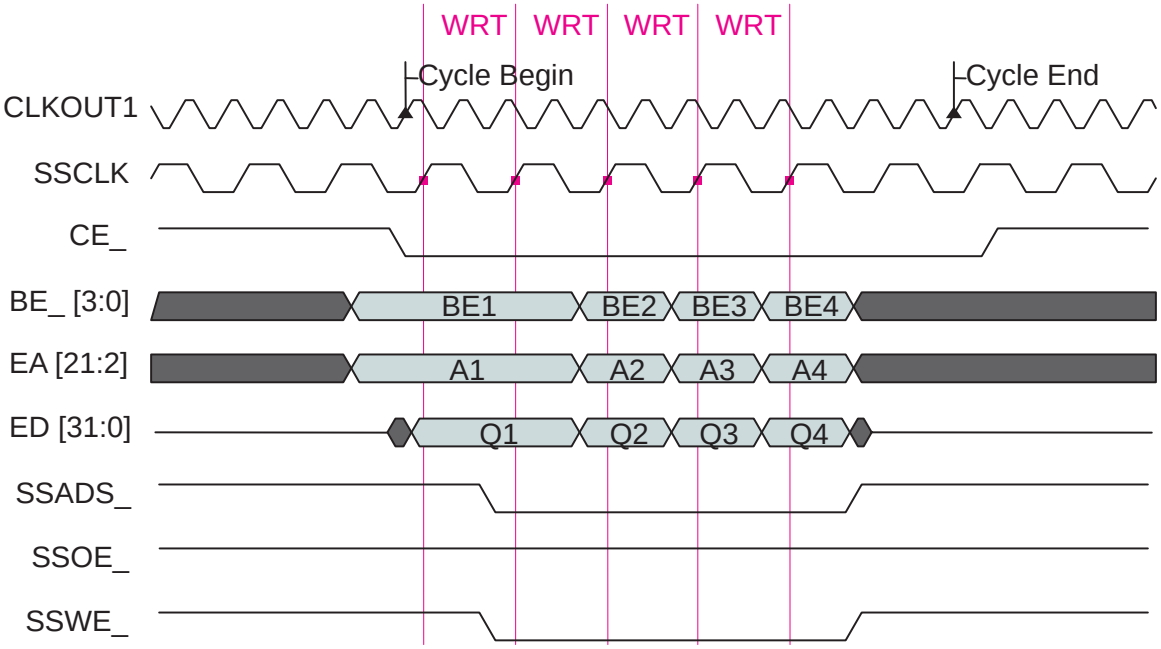

Figure A–4. 1/2x Rate SBSRAM Write Cycle Timings

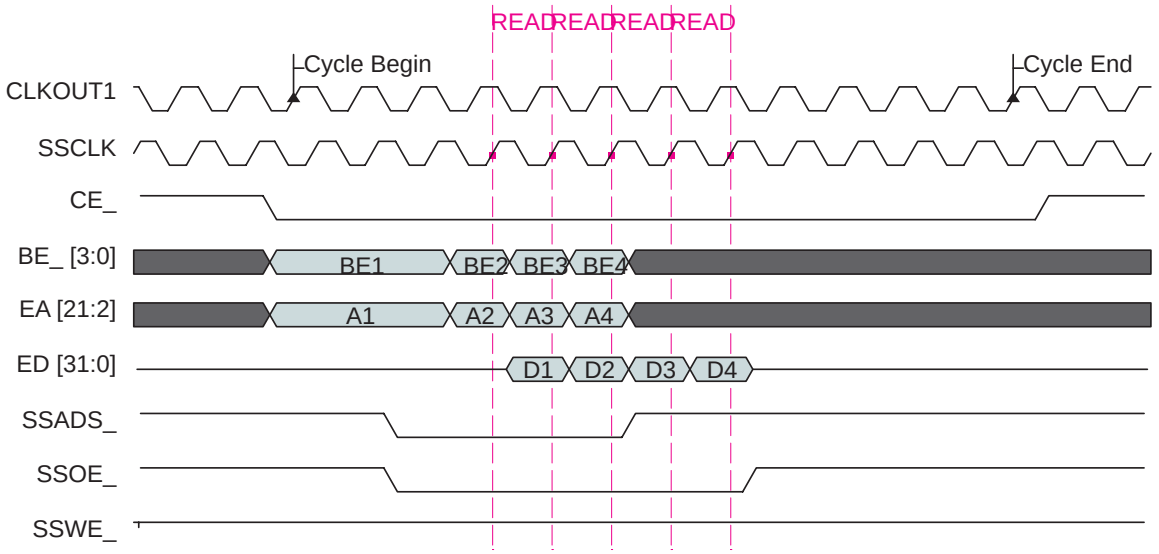

Figure A–5. 1x Rate SBSRAM Read Cycle Timings

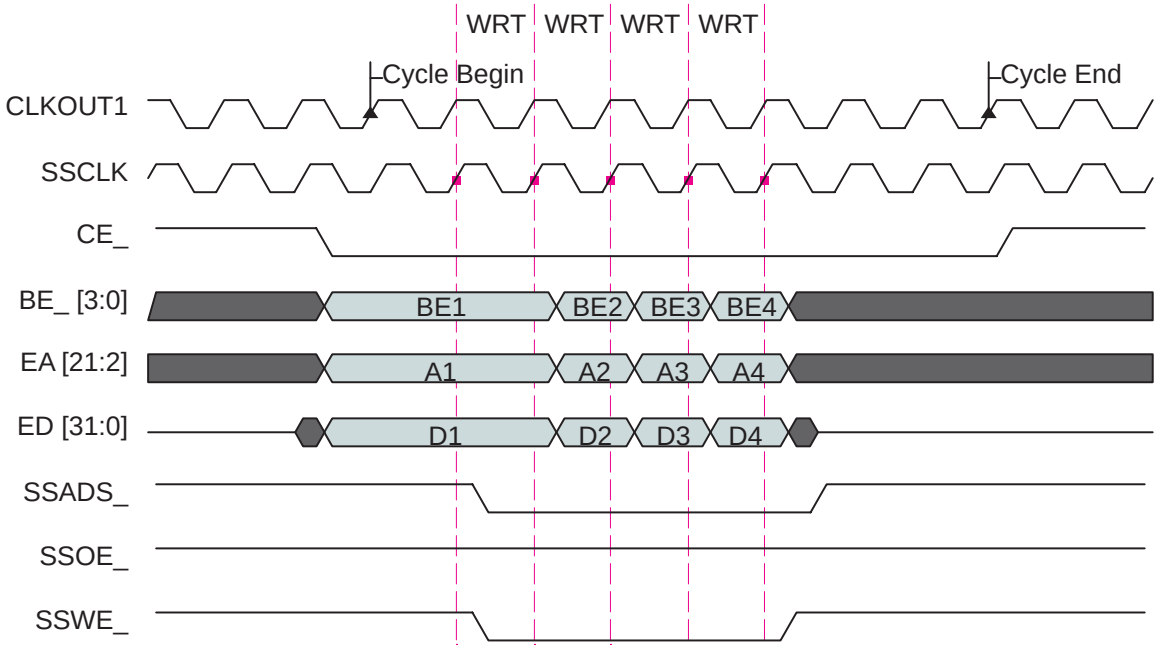

Figure A–6. 1x Rate SBSRAM Write Cycle Timings

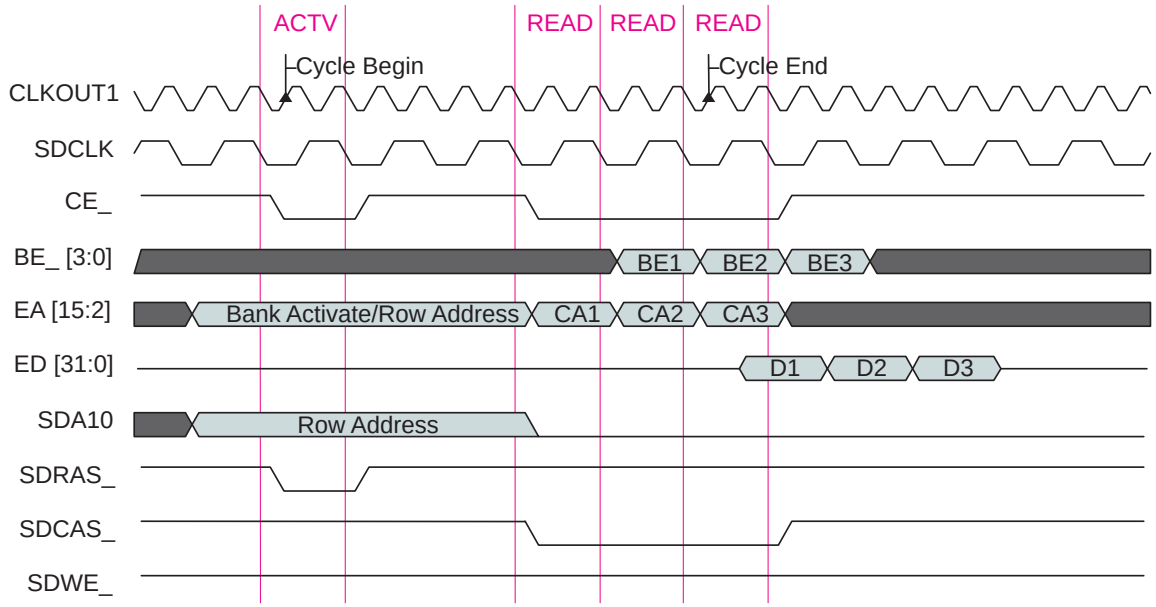

Figure A–7. SDRAM Read Cycle Timings with Row Activation

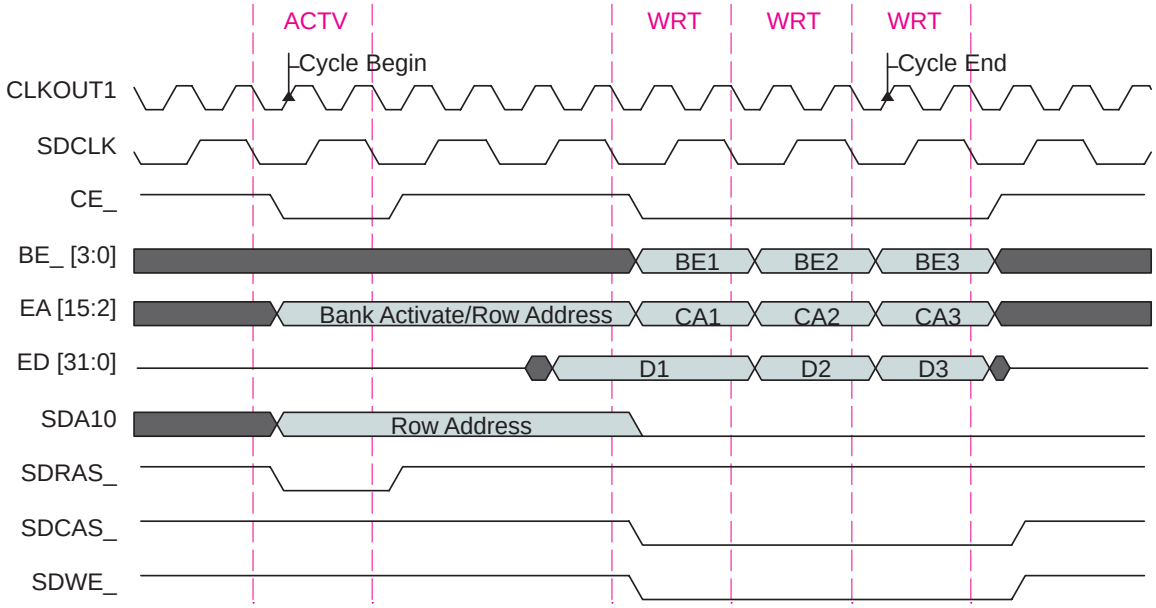

Figure A–8. SDRAM Write Cycle Timings with Row Activation

IMPORTANT NOTICE

Texas Instruments and its subsidiaries (TI) reserve the right to make changes to their products or to discontinue any product or service without notice, and advise customers to obtain the latest version of relevant information to verify, before placing orders, that information being relied on is current and complete. All products are sold subject to the terms and conditions of sale supplied at the time of order acknowledgment, including those pertaining to warranty, patent infringement, and limitation of liability.

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

Customers are responsible for their applications using TI components.

In order to minimize risks associated with the customer's applications, adequate design and operating safeguards must be provided by the customer to minimize inherent or procedural hazards.

TI assumes no liability for applications assistance or customer product design. TI does not warrant or represent that any license, either express or implied, is granted under any patent right, copyright, mask work right, or other intellectual property right of TI covering or relating to any combination, machine, or process in which such semiconductor products or services might be or are used. TI's publication of information regarding any third party's products or services does not constitute TI's approval, warranty or endorsement thereof.