TMS320C672x DSP Dual Data Movement Accelerator (dMAX)

Reference Guide

Literature Number: SPRU795D
November 2005 – Revised October 2007

Texas Instruments

Contents

List of Figures

List of Tables

Read This First

About This Manual

This document provides an overview and describes the common operation of the data movement accelerator controller (referred to as dMAX throughout this document) in the digital signal processors (DSPs) of the TMS320C672x™ DSP family. This document also describes operations and registers unique to dMAX. The following chapters are included:

- Chapter 1 provides an overview of dMAX.
- Chapter 2 provides a list of registers and register descriptions that are used in dMAX.
- Chapter 3 presents transfer examples for dMAX.
- Chapter 4 provides performance and throughput data along with guidelines on how to obtain the best performance.

Project collateral discussed in this reference guide can be downloaded from http://www.ti.com/lit/zip/SPRU795.

Notational Conventions

This document uses the following conventions:

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

Related Documentation From Texas Instruments

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

TMS320C672x DSP Peripherals Overview Reference Guide (literature number SPRU723) describes peripherals available on the TMS320C672x™ DSPs.

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

TMS320c672x DSP CPU and Instruction Set Reference Guide (literature number SPRU733) describes the TMS320C672x™ CPU architecture, instruction set, pipeline, and interrupts for these digital signal processors.

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

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

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

Trademarks

TMS320C672x, C6000, TMS320C62x, TMS320C67x, Code Composer Studio are trademarks of Texas Instruments.

Introduction/Feature Overview

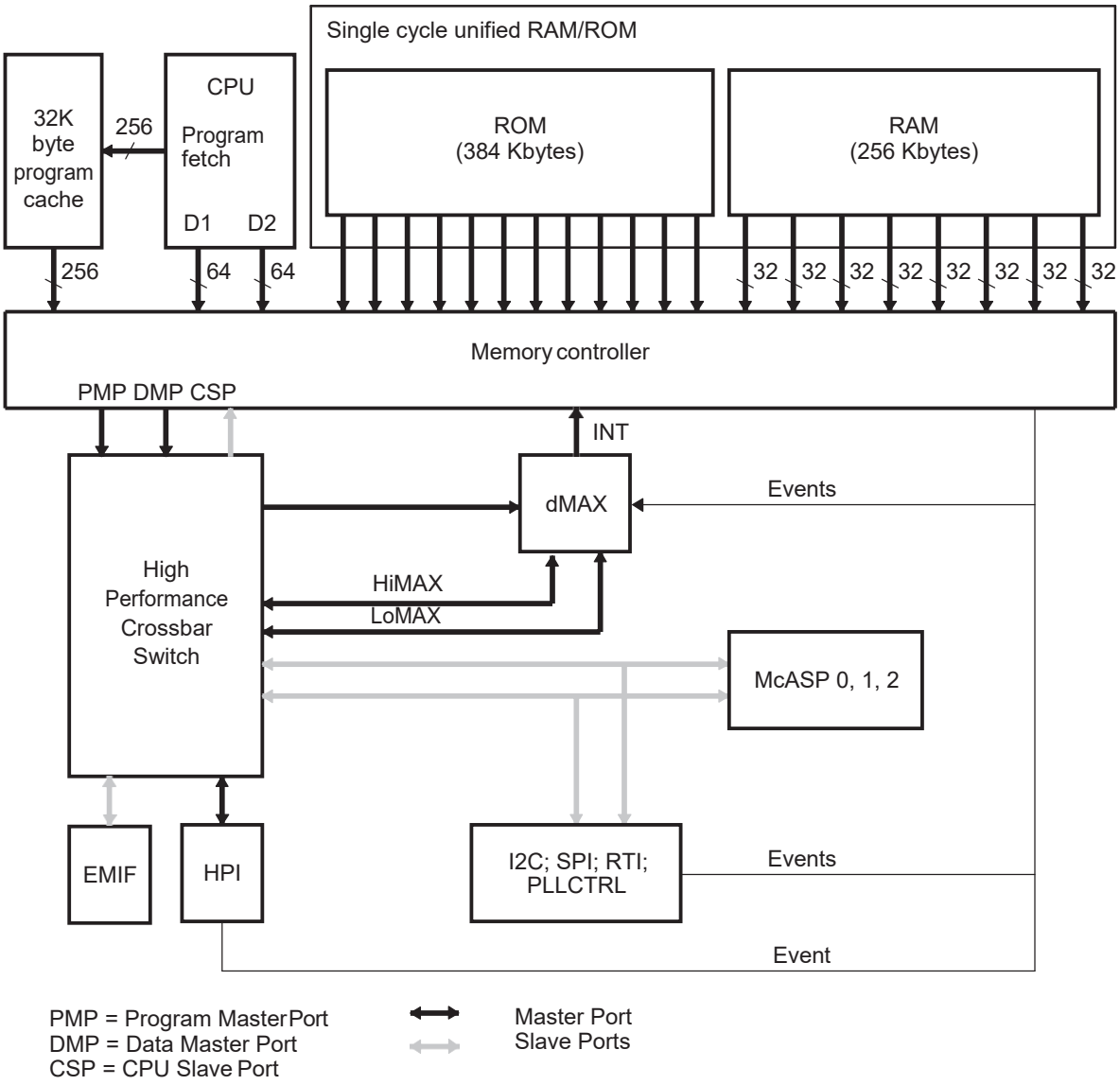
This chapter provides an overview of the data movement acceleration controller (dMAX) and its features.

Topic **Page**

1.1 Overview

The dMAX controller handles user-programmed data transfers between the internal data memory controller and the device peripherals on the C672x DSP, as shown in Figure 1-1. dMAX also allows movement of data to/from any addressable memory space, including internal memory, peripherals, and external memory. Additionally, it has a different architecture from the previous EDMA controller in the C621x/C671x devices.

Figure 1-1. TMS320C672x Block Diagram



The dMAX controller includes the capability to:

- Perform three-dimensional data transfers for advanced data sorting
- Manage a section of the memory as a circular buffer/FIFO with delay tap based reading and writing data
- Concurrently process two transfer requests (provided that they are to/from different source/destinations)

Figure 1-2 shows a block diagram of dMAX which includes:

- Event and interrupt processing registers
- Event encoder
- High priority event parameter RAM (PaRAM)
- Low priority event parameter RAM (PaRAM)
- Address generation hardware for high-priority events - MAX0 (HiMAX)
- Address generation hardware for low-priority events - MAX1 (LoMAX)

The TMS320C672x peripheral bus structure can be described logically as a high-performance crossbar switch with five master ports and five slave ports (shown in Figure 1-1). When accessing the slave ports, the MAX0 (HiMAX) module is always given the highest priority, followed by the MAX1 (LoMAX) module. If, for example, several masters, including MAX0 and MAX1, attempt concurrently to access the same slave port, the MAX0 module will be given the highest priority, followed by the MAX1 module.

Event signals are connected to bits of the dMAX Event Register (DER), and the bits in the DER reflect the current state of the event signals. An event is defined as a transition of the event signal. The dMAX Event Flag Register (DEFR) can be programmed individually for each event signal, to capture either low-to-high or high-to-low transitions of the bits in the DER (event polarity is individually programmable). Event polarity is programmable in the dMAX Event Polarity Register (DEPR).

An event is also a synchronization signal that can be used: 1) to trigger dMAX to start a transfer, or 2) to generate an interrupt to the CPU. All the events are sorted into two groups: a low-priority event group (the LoMAX module serves these requests) and a high-priority event group (the HiMAX module serves these requests).

Simultaneous occurrences of events are prioritized by the event encoder, which sorts them out and chooses the two highest priority events - one from each priority group. The event encoder then passes the events to the address-generation hardware. The priority of simultaneous events within a group is resolved according to the event number (an event with the lower number has higher priority within its group). dMAX can simultaneously process the two highest priority requests from each priority group.

Each PaRAM contains an event entry table section and a transfer entry table section. An event entry describes an event type and associates the event to either one of the transfer types or to an interrupt. If an event entry associates the event to one of the transfer types, the event entry will contain a pointer to the specific transfer entry in the transfer entry table. The transfer entry table may contain up to eight transfer entries. A transfer entry specifies details required by dMAX to perform the transfer. If an event entry associates the event to an interrupt, the event entry specifies which interrupt should be generated to the CPU when the event arrives.

Prior to enabling events and triggering a transfer, the event entry and transfer entry must be configured. The event entry must specify type of transfer, transfer details (type of synchronization, reload, element size, etc.), and should include a pointer to the transfer entry. The transfer entry must specify source, destination, counts, and indexes. If an event is sorted in the high-priority event group, the event entry and transfer entry must be specified in the high-priority parameter RAM. If an event is sorted in the low-priority event group, the event entry and transfer entry must be specified in the low-priority parameter RAM.

When an event is used to trigger a CPU interrupt, the event entry specifies which interrupt line should be used, and a transfer entry is not required. When an event is used to trigger a data transfer, the event entry specifies the type of transfer, transfer options, and points to the transfer entry. The transfer entry is stored in the parameter RAM, and is passed to the address generation hardware (MAX modules), which addresses the external memory interface (EMIF) and/or peripherals to perform the necessary read and write transactions.

Figure 1-2. dMAX Controller Block Diagram

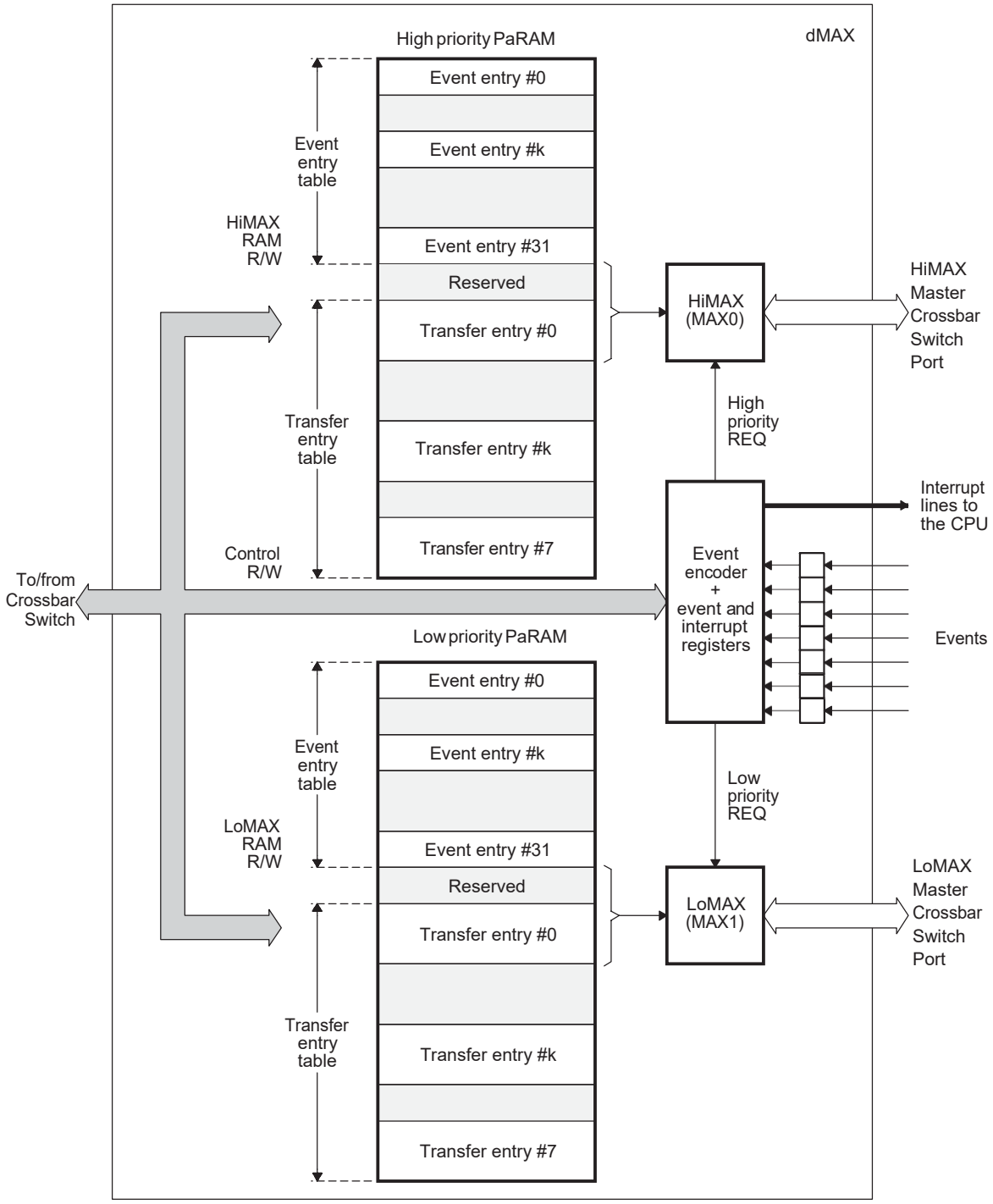


Table 1-1 summarizes the difference between dMAX and the C6000 EDMAs.

Table 1-1. Differences Between the C621x/C671x EDMA and C672x dMAX

Features	C621x/671x EDMA	dMAX
Maximum number of channels	16	16
Parameter RAM size	2048 bytes	1024 bytes (512b HiMAX + 512b LoMAX)
Alternate transfer complete interrupt	No	Yes
Transfer chaining	Only channels 8 to 11	No
Linking transfers	Yes	No (Values used for transfer reload are built inside a transfer entry)
Clock rate	EDMA clock rate equals CPU clock rate	dMAX clock rate equals of the CPU clock rate
Parameter storage for an event	6 words	11 words
Number of words in parameter RAM required to specify a data transfer with reload	18 words	11 words
CIER Register	Yes	No
CIPR Register Flag Clear	Write 1 to clear	dMAX Transfer Completion (DTCR) Register has similar functionality. Write 1 to clear.
Event Select Register	Yes	No (Event Entry Table used for similar purpose)
Priority Queue Status Register	Yes	No
Interrupt events to the CPU	1	8 dMAX handles CPU interrupts along with transfer events. One interrupt line (INT8) is dedicated for end of transfer notification. One interrupt line (INT7) is dedicated for FIFO status and error notifications.
Event Set Register	Yes	No. The CPU can initiate transfers by using dMAX Event Trigger (DETR) Register.
Event Clear Register	Yes	No.
Event Enable Register	Yes	Yes. Used only to enable events. Write 0 has no effect.
Event Disable Register	No	Yes. Write 1 to disable an event
QDMA transfers	Yes	No
Event polarity selection	No	Yes
Event Register (reflects current state of event signals)	No	Yes
Event Flag Register (Captures transitions on event signals captured in the Event Register)	Yes. (On 621x/671x, called ER)	Yes
3D transfer support	No	Yes
Independent index fields for source and destination for all transfer dimensions.	No	Yes
Size of index field for the first transfer dimension	16	16
Size of index field for the second transfer dimension	16	16
Internal Read/Write Path Width	64 bits	Each of two MAX modules has 32-bit wide path
Size of index field for the third transfer dimension	N/A	16
Frame index usage to derive the next frame start address	Frame index added to the start element address in a frame	Frame index added to the address of a last element in a frame
Priority levels for events	Yes. Set in the PRI bit field in the OPT parameter	Yes. If set, bits in DEHPR put events into the high-priority group. If set, bits in DELPR put events into the low-priority group.
Error notification to the CPU in case of FIFO overflow or underflow	No	Yes
Transfer indexes expressed in no. of elements	No	Yes
Circular buffer support	No	Yes
Table based multi-tap delay transfers	No	Yes

1.2 dMAX Terminology

The following definitions help to understand some of the terms used in this document:

- **dMAX:** Dual data movement accelerator. dMAX is composed of two equivalent modules, MAX0 and MAX1. The MAX modules can operate in parallel.
- **Element transfer:** An element transfer is the transfer of a single data element (8-,16-, or 32-bit) from source to destination. Each element can be transferred based on a synchronization event, if required. Element transfer is used in context with (1D) transfer.
- **Frame:** A group of elements comprise a frame. A frame can have staggered or contiguous elements. A frame can be transferred with or without a synchronizing event. Frame is used in context with one-dimensional (1D) transfer.
- **Event:** An event is a transition on an event signal latched in the dMAX Event Flag Register (DEFR). For example, data received by the McASP can trigger an event.
- **Event Entry:** If an event is used to trigger a data transfer, the event entry should be set to specify the type of transfer, the transfer options, and should include a pointer to a transfer entry. If the event is used to trigger an interrupt, the event entry only specifies which interrupt line should be used.

  The event entry uses only one word of memory space in the dMAX parameter RAM, within which is a one-to-one correspondence between the events and the event entries. A unique event entry is assigned to an event in each PaRAM (an event has one event entry in the high-priority PaRAM, and one event entry in the low-priority PaRAM). The event priority group decides which event entry will be passed to the HiMAX/LoMAX once an event arrives.

  All the events are sorted into low- or high-priority groups. The event encoder prioritizes all received events, and sorts the event with the highest priority from each group. The two highest priority events (one from each group) can be processed at the same time (HiMAX will process the highest priority request from the high priority event group, and LoMAX will process the highest priority request from the low priority event group). An event entry is programmable and defines how the corresponding event is going to be processed when it arrives.
- **Event Entry Table:** The event entry table contains all the event entries and occupies 32 words; it is located at the very beginning of the parameter RAM. There are two event entry tables, one for high priority events and the other for low priority events. If an event belongs to a high priority event group, then its event entry is located in the high-priority event entry table. If an event belongs to a low priority event group, then its event entry is located in the low-priority event entry table.
- **Event Priority Group:** dMAX events can be configured as either high-priority or low-priority. This splits events into two priority groups; high-priority or low-priority. The high-priority event group is serviced by the MAX0 module. The low-priority event group is serviced by the MAX1 module.
- **Transfer Entry:** The transfer entry table includes an 11-word long entry that defines transfer parameters such as source, destination, count, and indexes. There is enough space in each parameter RAM to keep transfer entries for eight different transfers (16 transfer entries total).
- **Quantum Transfer:** To improve system latency, long data transfers are divided into a number of smaller transfers (quantum transfers). The dMAX controller is always moving data in small sub-transfers called quantum transfers. If an event arrives while dMAX is performing a quantum transfer, the event will be serviced after the current quantum has been transferred.
- **Quantum Transfer Size Limit (QTSL):** The maximum size of a quantum transfer is programmable within the event entry for a given channel. It can be programmed to be 1, 4, 8, or 16 elements. The actual size of a quantum transfer is the smaller of the QTSL and the number of elements still to be transferred. Decreasing the QTSL will decrease the overall dMAX latency; increasing it will increase the dMAX data throughput.
- **Pending Event:** A pending event is an event latched in the DEFR that has not been processed by the dMAX controller.
- **Long Data Transfer:** A long data transfer occurs when the number of elements to be transferred after each synchronization event is larger than the QTSL.
- **One-Dimensional (1D) Transfer:** A group of elements makes up a 1D block. The number of elements in this block can be specified as well as the spacing between them. The spacing can be specified independently for both the source and the destination and can range from -32768 to +32767 elements).

- **Two-Dimensional (2D) Transfers:** A group of frames comprise a 2D block. The first dimension is the number of elements in a frame, and the second dimension is the number of frames. The number of frames in a 2D block can range from 1 to 65535. Either frames or the entire 2D block can be transferred at a time. Spacing between frames can be specified independently for source and destination (valid values for frame index: -32768 to 32767 elements).
- **Three-Dimensional (3D) Transfers:** A group of 2D blocks comprise a 3D block. The first dimension is the number of elements in a frame, and the second dimension is the number of such frames, and the third dimension is number of 2D blocks. The number of 2D blocks can range from 1 to 32767. Either a frame or the entire 3D block can be transferred at a time. Spacing between 2D blocks can be specified independently for source and destination (valid values for spacing between 2D blocks are: -32768 to 32767 elements).
- **One-Dimensional Burst (1DN) Transfer:** One-dimensional burst transfer is optimized for moving sequential data from one memory location to the other. This transfer does not support non-sequential source or destination.
- **SPI Slave Transfer:** SPI peripheral servicing requires that for a given SPI event, one element be read from the SPI input register and an element be written to the SPI output shift register. The SPI slave transfer provides this functionality.
- **FIFO (Circular Buffer):** A FIFO is defined by its base address, size, two watermarks and two pointers (read pointer and write pointer). The two pointers are continuously chasing each other as data is being written to and read from the buffer. Reads and writes to the buffer are asynchronous to each other. When the FIFO is filled with data, the pointers wrap around and new samples overwrite the old data.

  The FIFO size is specified in number of elements; it does not have to be a power of two.
- **Table-based Multi-tap Delay Transfer:** Many audio algorithms access large delay buffers in a non-sequential fashion. A table-based, multi-tap delay transfer reads/writes elements to/from a FIFO according to table of pre-defined delay tap offsets.

  Table based, multi-tap delay memory access patterns have arbitrary spacing between consecutive taps (defined by the delay tables), and have predictable contiguous spacing within a tap (the delay samples within a tap are contiguous).
- **dMAX Channel:** An event signal associated with the event entry and transfer entry used to transfer data.
- **Reference (Reload) Registers in Transfer Entry:** Values from the reference set of registers are used to load the active set of registers at the end of a transfer if reload is enabled. This facilitates the ping-pong buffering scheme.
- **Active Registers in Transfer Entry:** These active sets of registers are updated by dMAX during the course of a transfer. dMAX maintains the current transfer state information in the set of active registers.
- **Delay Table:** A delay table is referenced by a pointer in a transfer entry for a FIFO transfer. The table lists all required delays for table based multi-tap delay transfers (the delay offsets are referenced to the buffer pointers).

1.3 Initiating dMAX Transfers

There are two ways to initiate a data transfer using the dMAX controller:

- Event-triggered dMAX transfer (this is a more typical usage of dMAX)
- CPU-initiated dMAX transfer

An event-triggered dMAX transfer allows the submission of transfer requests to occur automatically, based on system events, without any intervention by the CPU. dMAX also includes support for CPU-initiated transfers for added control and robustness, and they can be used to start memory-to-memory transfers. To generate an event to dMAX, the CPU must create a transition on one of the bits from the dMAX Event Trigger Register (DETR), which are mapped to the dMAX Event Register (DER) based on the polarity.

Each dMAX transfer can be started independently. The CPU can also disable a dMAX channel by disabling the event associated with that channel.

- **Event-triggered dMAX Transfer:** If an event is enabled, and latched in the DEFR, the event encoder causes its event entry and its transfer entry to be passed to the address generation hardware, which performs the requested accesses. Although the event causes this transfer, it is very important that the event itself be enabled by the CPU. Writing a 1 to the corresponding bit in the dMAX Event Entry Register (DEER) enables an event. Alternatively, an event is still latched in the DEFR, even if its corresponding enable bit in the DEER is 0 (disabled). The dMAX transfer related to this event occurs as soon as it is enabled in the DEER.
- **CPU-initiated dMAX Transfer:** For CPU-initiated transfers, the CPU uses the DETR. To initiate a transfer, the CPU must create an appropriate edge on a bit in DETR (the appropriate edge depends on the polarity set for the event). A transition on a DETR signal will be latched in the DEFR. Just as with an event coming from a peripheral, the event entry and transfer entry in the dMAX parameter RAM corresponding to this event are passed to the address generation hardware, which performs the requested access as appropriate. CPU-initiated dMAX transfers are unsynchronized data transfers.

Prior to enabling events and triggering a transfer, the event entry and transfer entry must be configured. The event entry must specify: type of transfer, transfer details (type of synchronization, reload, element size, etc.), and should include a pointer to the transfer entry. The transfer entry must specify source, destination, counts, and indexes. If an event is sorted in the high-priority event group, the event entry and transfer entry must be specified in the high-priority parameter RAM. If an event is sorted in the low-priority event group, the event entry and transfer entry must be specified in the low-priority parameter RAM.

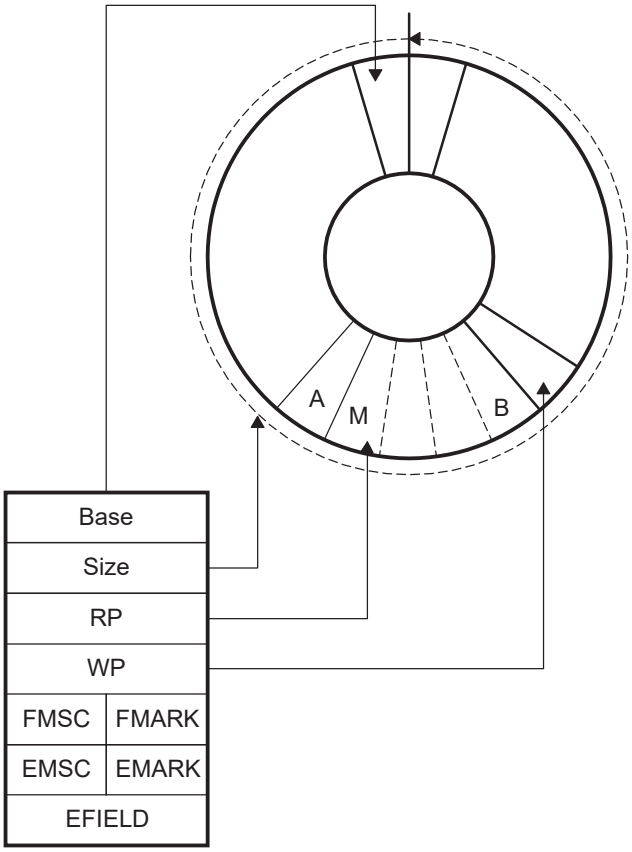
1.4 FIFO Implementation

The dMAX controller has the capability to utilize a section of the memory as a circular buffer/FIFO and supports dedicated transfer types to and from a FIFO (circular buffer). In this case, the FIFO is a block of memory (external or internal) in the DSP RAM defined by its base address and its size, and the size of elements that it holds (8-, 16-, or 32-bit). The size of the FIFO buffer is specified in terms of the maximum number of elements the buffer can hold; it does not have to be a power of two. The maximum size of a FIFO is limited to 1048576 elements.

When the FIFO buffer is filled with data, the buffer wraps around and new samples overwrite the old data.

A FIFO buffer is described by a FIFO descriptor, which can be located any place in the DSP memory. A block diagram of a FIFO and its descriptor is presented in Figure 1-3.

Figure 1-3. Parameters Defining a FIFO: Read Pointer, Write Pointer, FIFO Base Address, FIFO Size, EMARK, FMARK, FMSC, EMSC, and EFIELD



In FIFO read and write transfer entries, a FIFO is referenced by using a pointer to the FIFO descriptor.

In the discussion below, it is assumed that reads and writes to a FIFO are performed by the dMAX controller. If the CPU reads/writes data to a FIFO, the read and write pointers are not going to be automatically updated, and special care must be taken to keep the pointer values current.

The write pointer points to a FIFO location where dMAX will store the next incoming sample. The write pointer is automatically updated at the end of a transfer in which dMAX writes new samples to the FIFO. In Figure 1-3 the new samples are written to the buffer in a counter-clockwise direction, and the last sample written to the buffer is marked with (B). The write pointer is referenced to the base address of a buffer (the write pointer value is zero when it is pointing to a FIFO base address). Once the value in the write pointer reaches the size of the buffer, a write of a new sample to the buffer will force the pointer to wrap around to zero.

The read pointer points to a FIFO location from which dMAX will retrieve the next sample. At the end of a transfer in which samples are read from the FIFO, the read pointer is automatically updated. In Figure 1-3 the new samples are read from the FIFO in a counter-clockwise direction, and the last sample read from the buffer is marked with (A). The read pointer is referenced to the base address of a buffer (read pointer value is zero when it is pointing to a buffer base address). Once the value in the read pointer reaches the buffer size, a read from the buffer will force the pointer to wrap around to zero.

1.4.1 FIFO Watermarks

It is useful to detect conditions when the number of unread elements in a FIFO drops below a certain number or *grows* beyond another predefined number (these two levels will be referred to as FIFO watermarks).

A dedicated interrupt line is reserved to indicate the FIFO watermark conditions and FIFO errors to the CPU. The FIFO status interrupt line and the dMAX FIFO Status Register (DFSR) are used to notify the CPU about the status of FIFOs.

Two watermarks, EMARK and FMARK, are assigned to a FIFO along with parameters shown in Figure 1-3. A status bit is assigned to each watermark condition. The bit is set in the DFSR only when the watermark is reached.

When the number of unread samples in the FIFO becomes larger or equal than the pre-defined FMARK, dMAX will signal the watermark condition to the CPU by triggering a FIFO status interrupt, and by setting the Full Mark Status (FMSC) bit in the DFSR.

When the number of fresh samples in the FIFO (samples that have not been read from the FIFO) becomes equal or drops below than pre-defined EMARK, dMAX will signal the watermark condition to the CPU by triggering a FIFO status interrupt, and by setting the Empty Mark Status (EMSC) bit in the DFSR.

In order to receive a next watermark notification, the CPU needs to clear the EMSC or FMSC status bits in the DFSR. The dMAX controller will not trigger a new FIFO status CPU interrupt unless the status bits are cleared by the CPU for the last watermark condition reported.

1.4.2 FIFO Error Field

If an error occurs when reading or writing to a FIFO, dMAX uses the error field (EFIELD), within the FIFO descriptor, to indicate the error type to the CPU. If an error is detected, the dMAX controller will abort a transfer and a CPU intervention is required to resume operation.

The dMAX controller notifies the CPU about FIFO transfer error by setting both status bits assigned to the FIFO (FMSC and EMSC) in the DFSR, by writing error code to the EFIELD and triggering a FIFO status interrupt.

The dMAX controller reports three types of FIFO errors to the CPU:

- An overflow error is indicated and a FIFO transfer is aborted before unread samples from FIFO are overwritten. Each time before performing a write transfer, dMAX compares a bit value contained in the active COUNT0 bit field with the number of empty slots in the FIFO. If the number of empty slots in the FIFO is smaller than the value contained in the COUNT0 bit field, dMAX will abort the write transfer and flag an error to the CPU.
- An underflow error condition is flagged and a transfer is aborted when an attempt is made to read more than the number of unread samples stored in the FIFO. Each time before performing a read transfer, dMAX compares a bit value contained in the active COUNT0 bit field with the number of samples in the FIFO available for read. If the number of samples available for read in the FIFO is smaller than the value contained in the COUNT0 bit field, dMAX will abort the read transfer and flag an error to the CPU.
- In a table-based, multi-tap delay transfer, if a delay specified in the delay table is larger than number of samples stored in the FIFO, an error will be generated and the transfer will be aborted.

1.5 Types of dMAX Transfers

The dMAX controller provides for five types of data transfers:

- General purpose data transfer (covers one-dimensional (1D), two-dimensional (2D), and three-dimensional (3D) transfer)
- FIFO write transfer with support for table-based, multi-tap delay memory access
- FIFO read transfer with support for table-based, multi-tap delay memory access
- One-dimensional burst (1DN) transfer
- SPI slave transfer

CAUTION

1DN and SPI Slave transfers requires System Patch Version 2_00_00 or later: http://focus.ti.com/docs/toolsw/folders/print/sprc203.html

One-dimensional and two-dimensional transfers are implemented as special cases of a three-dimensional transfer, where counters for the higher dimensions are equal to zero (e.g., a 3D transfer with the third dimension counter equal to zero becomes a 2D transfer).

A dMAX transfer is described by its transfer entry. The entry defines the required transfer parameters such as source, destination counts, indexes, etc. For a general-purpose data transfer, an independent set of indexes for source and destination can be specified for each transfer dimension. This facilitates 1D-2D, 1D-3D, and 2D-3D element sorting.

The format for the transfer entry is different for different transfer types. The transfer entry contains a set of active registers that are continuously updated by dMAX during the course of a transfer. It also contains reference registers which are used to reload values in the active register set.

When a transfer is complete, an active address register can be reloaded from one of two sets of address reference registers. The reload option can be enabled or disabled by the RLOAD bit-field of the event entry (reload is explained in Section 1.9 and the RLOAD bit field is defined in Section 2.1.1.1). An active register set with the capability of reloading from one of two reference register sets facilitates implementation of various ping-pong buffering schemes.

A set of active registers for a general-purpose data transfer includes source address, destination address, and element counters for each transfer dimension (COUNT2, COUNT1 and COUNT0). There are two sets of reference registers that specify reference values for source, destination address, and one reference counter register. During a transfer, dMAX uses the active register set and the reference counter. If reload is enabled at the end of the transfer, active address registers are loaded from one of two sets of reference address registers. The address reference register set used during reload is specified by the PP bit (defined in Section 2.1.2.1) within the transfer entry.

A set of active registers for a FIFO transfer includes a linear address and frame and element counter. Two registers hold reference values for linear address. One register holds a reference value for frame and element counter. During a transfer, the dMAX controller uses the active register set, and a reference element counter. If reload is enabled, at the end of the transfer, active linear address is loaded from one of two reference address registers. The address reference register used during reload is specified by the PP bit within the transfer entry (the PP bit is defined in Section 2.1.2.2 and Section 2.1.2.3).

Transfer synchronization is specified by the SYNC bit in the event entry. Transfers can be either frame-synchronized (a frame of data is transferred after receiving a synchronizing event) or whole transfer can be completed after receiving a synchronization event. Element-synchronized transfers (one element is transferred after receiving a synchronization event) are considered a special case of frame synchronized transfers. By making frame size equal to one, and by selecting frame synchronization with the SYNC bit, a frame-synchronized transfer becomes element-synchronized. SPI slave transfer allows for servicing the SPI peripheral when used in slave mode. The peripheral servicing requires that for a given SPI event, one element be read from the SPI input register and an element be written to the SPI output shift register. The SPI slave transfer supports this functionality. A set of active registers for a SPI slave transfer includes source address, destination address, and element counter. For each input event, one element is read from the SPI input shift register (SPIBUF) and is stored in the destination address. Also, one element is read from the input address and moved to the SPI output shift register (SPIDAT0).

The input address and the output address are incremented by one after handling each event. There are two sets of reference registers that specify reference values for source address and destination address; also there is one reference counter register. During a transfer, dMAX uses the active register set. If reload is enabled at the end of the transfer, active address registers are loaded from one of two sets of reference address registers. The address reference register set used during reload is specified by the PP bit within the transfer entry.

One-dimensional burst transfer (1DN) is optimized for moving sequential data from one memory location to the other. This transfer does not support non-sequential source or destination. It also does not support reload feature. The data transfer happens in bursts. The burst length (number of elements transferred in a single burst) and the number of bursts for each transfer can be programmed. The transfer is designed to require minimal setup overhead and allow for fast data movement and therefore does not provide some of the features supported by general purpose transfers.

1.5.1 One-Dimensional Transfers

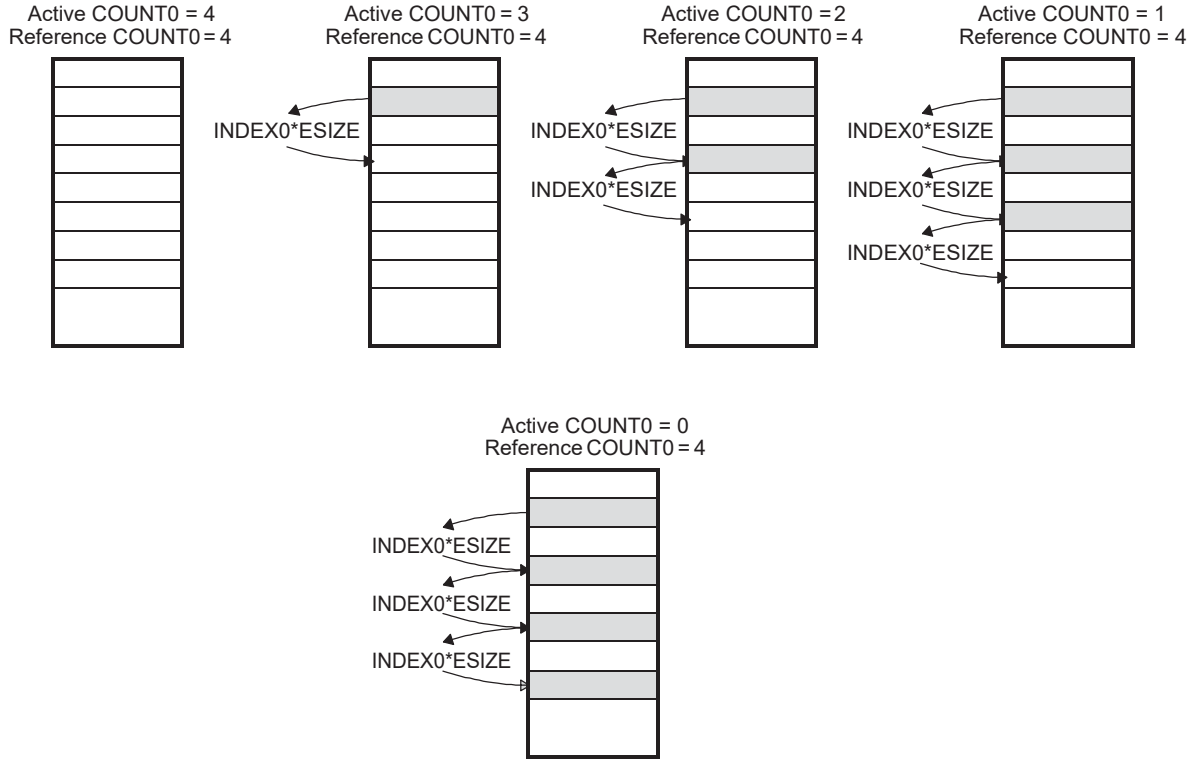
A transfer is one-dimensional (1D) when the COUNT2 and COUNT1 registers in a transfer entry are equal to zero and smaller or equal to one, respectively, and if the COUNT0 register is greater than zero. In this case, dMAX will transfer the number of elements specified by the COUNT0 bit field of the transfer entry. The maximum number of elements that can be moved by a 1D-transfer is 65535.

During the course of a transfer, dMAX updates the active parameters within transfer entry (active source address, active destination address and active element counter).

The distance between elements within the frame is specified by index0 and can be independently controlled for source and destination (source and destination index 0). After each element transfer, active source and destination addresses are updated by the product of element size and the appropriate index0. The source index0 and destination index0 are expressed in number of elements, and can be between -32768 to 32767.

A 1D transfer is graphically presented in Figure 1-4. The example presents transfer phases in a case when active COUNT0 and reference COUNT0 are set to four prior to transfer.

Figure 1-4. One-Dimensional Transfer



The active COUNT0 field within the transfer entry is decremented after each element transfer. A 1D transfer is complete when the active COUNT0 field is decremented to zero. In a 1D transfer, the value of the SYNC bit field of the event entry is ignored and COUNT0 number of elements is transferred after receiving an event.

After a transfer is complete, and when reload is enabled, the reference counters and a different set of address reference registers will be loaded in the active register set. There are two sets of reference registers for source and destination addresses, and one set of counter registers. These two sets of reference address registers facilitate ping-pong buffering. A new transfer will be automatically kicked off after a new event is received.

1.5.2 Two-Dimensional Transfers

A transfer is two-dimensional (2D) when the COUNT2 register field in a transfer entry is smaller or equal to one, and the COUNT1 and COUNT0 register fields are greater than zero. In this case, the dMAX controller will transfer a 2D block (COUNT1 number of frames, and each frame will have COUNT0 number of elements).

During course of a transfer, the dMAX controller updates the active parameters within transfer entry (active source address, active destination address and active element counter).

The size of COUNT1 and COUNT0 control bit-fields within the transfer entry is adjustable by setting the counter configuration bits in the event entry. There are two options for a 2D transfer:

- The maximum number of frames in a 2D block can be up to 65535. In this case frame can contain up to 255 elements.
- The maximum frame size of 65535 elements can be achieved if the maximum number of frames in a 2D block is limited to 255.

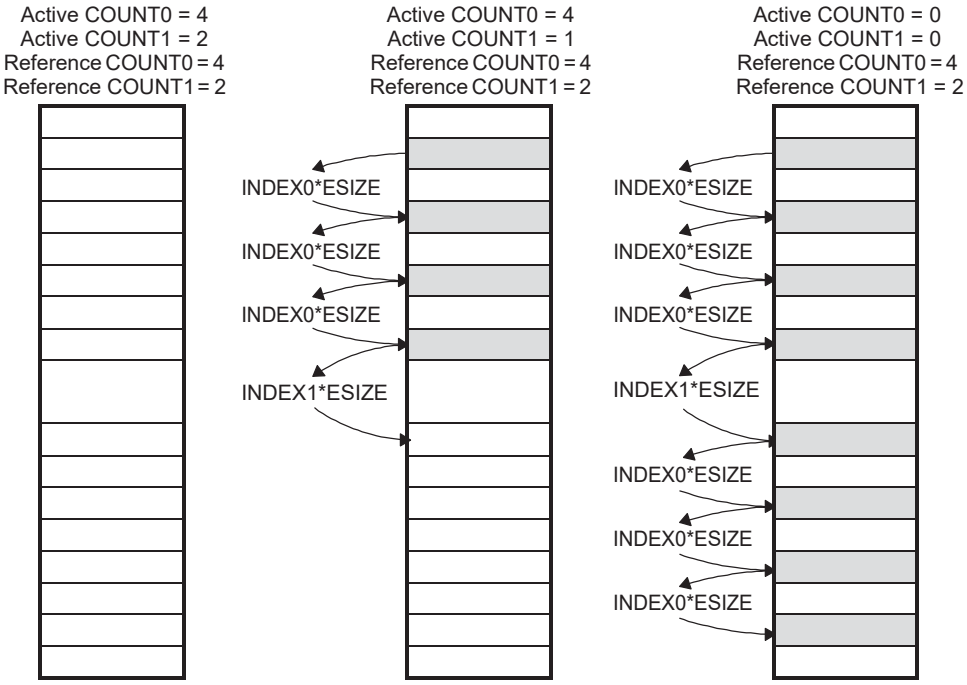
The frame index (index1) defines the distance, in number of elements, between the last element in a frame and the first element of the next frame, and it can be between -32768 and 32767. The frame index can be controlled independently for source and destination.

The distance between elements within the frame is specified by index0 and can be independently controlled for source and destination (source and destination index 0). The source index0 and destination index0 are expressed in number of elements, and can be between -32768 and 32767. While the active COUNT0 control register is greater than zero, source and destination address are updated after each element transfer by using the product of element size and the appropriate index0.

When the active COUNT0 control register is decremented to zero, the active COUNT1 control register is decremented by one and source and destination addresses are updated by using the product of element size and the appropriate index1. The active COUNT0 control register is then reloaded from the COUNT0 reference register.

A 2D transfer is graphically presented in Figure 1-5. The example presents transfer phases where active COUNT0 and reference COUNT0 are set to four, and active COUNT1 and reference COUNT1 are set to two prior to transfer. The left panel in Figure 1-5 presents the memory and counters prior to transfer. A phase after the first frame was transferred is presented by the middle panel.

Figure 1-5. A Two-Dimensional Transfer



After the transfer of each frame, the active COUNT1 register is decremented and the active COUNT0 register is reloaded from the COUNT0 reference register. A transfer is complete when the active COUNT1 field is decremented to zero.

A 2D transfer can be either frame-synchronized, which means that one frame of COUNT0 elements is moved after receiving a synchronization event, or whole transfer, which means that COUNT1 number of frames can be moved after receiving a synchronization event. When the SYNC bit field of the event entry is equal to zero, the transfer is frame-synchronized and only one frame is transferred after receiving an event. When the SYNC bit field of the event entry is equal to one (COUNT0 equal to one), the data becomes element-synchronized. When the SYNC bit field of the event is equal to one, the full 2D block is moved after receiving an event.

If reload is enabled after a transfer is complete, the reference counter and a different set of address reference registers will be loaded in the active register set. There are two sets of reference registers for source and destination addresses and one reference element counter register. Two sets of reference address registers facilitate ping-pong buffering. A new transfer will be automatically kicked off, after a new event is received.

1.5.3 Three-Dimensional Transfers

A transfer is three-dimensional (3D) when the COUNT2 register is greater than one, and COUNT1, and COUNT0 registers in a transfer entry are greater than zero. In this case, the dMAX controller will transfer COUNT2 number of 2D blocks. Each 2D block will have COUNT1 frames, and each frame will have COUNT0 number of elements.

During the course of a transfer, dMAX updates the active parameters within the transfer entry (active source address, active destination address, and active element counter).

The size of COUNT2, COUNT1 and COUNT0 control bit-fields within the transfer entry is adjustable by setting the counter configuration bits in the event entry. There are three options for 3D transfers:

- The maximum number of 2D blocks can be up to 32767. Each 2D block will have up to 255 frames, and each frame can have up to 255 elements;
- The maximum frame size can be up to 65535 elements. The maximum number of 2D blocks is limited to 127, and each 2-D block is composed of up to 255 frames.
- The maximum number of frames, within a 2D block, can be up to 65535. The number of 2D blocks is limited to 127, and each frame has up to 255 elements.

The 2D-block index (index2) defines the distance, in number of elements, between the last element in a 2D block and the first element of the next 2D block, and it can be between -32768 and 32767. The block index can be controlled independently for source and destination.

Frame index (index1) defines the distance, in number of elements, between the last element in a frame and the first element of the next frame, and it can be between -32768 and 32767. The frame index can be controlled independently for source and destination.

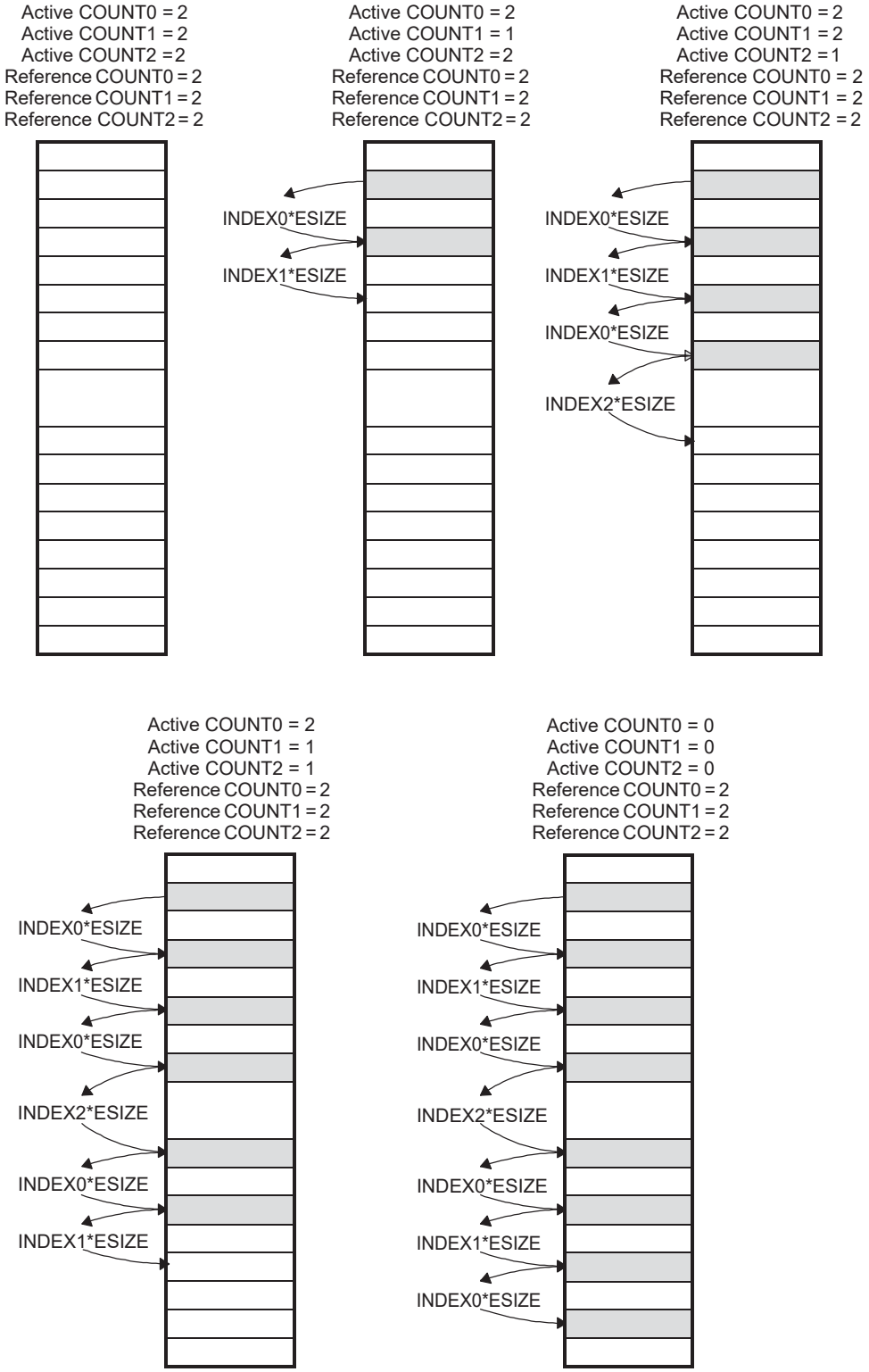
The distance between elements within the frame is specified by index0, and can be independently controlled for source and destination (source and destination index 0). The source index0 and destination index0 are expressed in number of elements, and can be between -32768 and 32767. While the active COUNT0 control register is greater than zero, source and destination addresses are updated by using the product of element size and the appropriate index0 after each element transfer.

When the active COUNT0 control register is decremented to zero, the active COUNT1 control register is decremented by one and the source and destination addresses are updated by using the product of element size and the appropriate index1. The active COUNT0 control register is then reloaded from one of two COUNT0 reference registers.

When the active COUNT1 control register is decremented to zero, the active COUNT2 control register is decremented by one and the source and destination addresses are updated by using the product of element size and the appropriate index2. The active COUNT1 control register is then reloaded from the COUNT1 reference register.

A 3D transfer is shown in Figure 1-6. The example presents phases after each frame transfer in a case when active COUNT0, COUNT1, COUNT2 and reference COUNT0, COUNT1, COUNT2 are set to two prior to transfer.

Figure 1-6. A Three-Dimensional Transfer



After the transfer of each frame, the active COUNT1 register is decremented and the active COUNT0 register is reloaded from the COUNT0 reference register. After transfer of the 2D block, the active COUNT2 register is decremented and the active COUNT1 register is reloaded from the COUNT1 reference register. A transfer is complete when the active COUNT2 field is decremented to zero.

Transfer synchronization is specified by the SYNC bit in the event entry. Transfers can be frame-synchronized (one frame of data is moved after receiving a synchronizing event), or whole transfer can be moved after receiving a synchronization event. Element-synchronized transfers (one element is transferred after receiving a synchronization event) are considered a special case of frame-synchronized transfers. By making the frame size equal to one, and by selecting frame-synchronization with the SYNC bit, a frame-synchronized transfer becomes element-synchronized.

If reload is enabled after a transfer is complete, the reference counter and a different set of address reference registers will be loaded in the active register set. There are two sets of reference registers for source and destination addresses and one reference element counter register. Two sets of reference address registers facilitate ping-pong buffering. A new transfer will be automatically kicked off, after a new event is received.

1.5.4 FIFO Transfers

The dMAX controller can move data between a two-dimensional linear address and a circular buffer (FIFO). dMAX supports FIFO write and FIFO read type of transfers. A FIFO transfer is specified by its transfer entry in the PaRAM. The transfer entry describes the transfer and contains a pointer to the descriptor of a FIFO which is used either as a transfer source or as a transfer destination.

Two-dimensional linear address space is described in the FIFO transfer entry by its address, two dimensional element counter, and two indexes - one for each dimension. The index0 is expressed in number of elements and represents distance between two elements within a frame. The index0 can be anywhere between -32768 to 32767. The index1 is also expressed in number of elements; it represents distance between two frames, and it can be between -32768 to 32767.

During the course of a transfer, dMAX updates the active parameters within transfer entry (active linear address and active element counter), and the appropriate pointer within the FIFO descriptor. For a FIFO write, dMAX will update the write pointer, and for a FIFO read, the read pointer gets updated.

When the active element counter (COUNT0) control register is greater than zero, the active linear address is updated after each element transfer by using the product of element size and the index0.

When the active element counter (COUNT0) control register is decremented to zero, the active frame counter (COUNT1) control register is decremented by one and the active linear address is updated by using the product of element size and the index1. The active element counter control register is then reloaded from the reference element counter register.

The appropriate pointer within the FIFO descriptor (read pointer for a FIFO read transfer or write pointer for a FIFO write) is updated only when the active frame counter control register is decremented to zero. The pointer is adjusted by adding reference element counter (COUNT0). The pointer adjustment is always calculated by modulo FIFO buffer size.

Values in the active frame counter (COUNT1) register must be always smaller or equal to the value specified in the reference frame counter (COUNT1R) register. If that is not the case, dMAX will ignore the transfer entry and will not perform the transfer.

During a FIFO read transfer, if the reference frame counter (COUNT1) is greater than zero, multiple sets of consecutive samples (taps) can be read from a FIFO and stored in the linear DSP memory without CPU involvement. The taps can be read anywhere from the FIFO. The tap position (tap delay) within the FIFO buffer is expressed in number of samples and is calculated by subtracting from the FIFO read pointer. All taps in a multi-tap delay read transfer have the same size (defined by reference element counter COUNT0). Tap delay values are stored in a dedicated table (delay table).

During a FIFO write transfer, if the reference frame counter COUNT1 is greater than zero, a data frame read from the linear DSP memory can be stored to the FIFO as a consecutive set of samples (taps). The taps can be stored anywhere in the FIFO buffer. The tap position (tap delay) within the FIFO buffer is expressed in number of samples and is calculated by subtracting from the FIFO write pointer. All taps in a multi-tap delay write transfer have the same number of elements (defined by reference element counter COUNT0). Tap delay values are stored in a dedicated table (delay table).

A FIFO transfer can be either frame-synchronized (COUNT0 elements moved after receiving a synchronization event) or whole transfer (can be moved after receiving a synchronization event). If the SYNC bit field of the event entry is equal to zero, the transfer is frame-synchronized and only one frame (COUNT0 elements) is transferred after receiving an event. If the frame size is equal to one, the data transfer becomes element synchronized. If the SYNC bit field of the event entry is equal to one, the whole transfer is synchronized to one event.

If reload is enabled after a transfer is complete, the reference counter and a linear address reference register will be loaded in the active register set. There are two sets of linear address reference registers and only one reference element counter register. Two sets of linear address reference registers facilitate ping-pong buffering. A new transfer will be automatically kicked off, after a new event is received.

Each FIFO transfer entry contains pointers to two delay tables. A dedicated delay table can be used with each of the two reference register sets (a different delay table can be used for ping and pong buffers). The delay tables are referenced from the transfer entry by pointers, and can be located anywhere in the DSP memory space. The same delay tables can be applied to different FIFOs by specifying the same pointers to delay tables in the FIFO transfer entries.

1.5.4.1 FIFO Write

A transfer entry for FIFO write includes: an active copy of source address, a pointer to FIFO descriptor, two source address indexes, an active and a reference copy of element counters, two sets of reference registers used to reload source address, and two pointers to delay tables.

An active copy of source address points to a DSP memory location which will be copied to the FIFO buffer. An active copy of source address is automatically updated by the dMAX controller during a course of a transfer. Instead of fully describing the FIFO buffer within the transfer entry, a pointer to the FIFO descriptor is used (for more detail on the FIFO descriptor see Section 1.4 and Section 2.2). The FIFO descriptor can be located anywhere in the DSP RAM.

An element counter is two-dimensional when the frame count register COUNT1 in a transfer entry is greater than one. In this case, COUNT1 number of frames will be transferred by the dMAX controller, and each frame will have COUNT0 samples. Each of these frames is stored to the FIFO as a set of consecutive elements (taps).

The tap position within the FIFO is determined by delay table entries. The tap delay is expressed in number of samples and it is calculated from the FIFO write pointer. The number of entries in the delay table is equal to value of reference frame count register COUNT1. There is one-to-one correspondence between delay table entries and taps (each delay table entry is assigned to one tap). The first delay read from the delay table corresponds to the first tap written to the FIFO; the second delay read from the delay table corresponds to the second tap written to the FIFO, and so on.

Source index1 defines the distance, in number of elements, between the last element in a frame and the first element of the next frame, and it can be between -32768 and 32767. The distance between elements within the frame is specified by index0. The source index0 is expressed in number of elements, and can be between -32768 and 32767. While the active COUNT0 control register is greater than zero, source address is updated after each element transfer by using the product of element size and the appropriate index0.

When the active element counter COUNT0 control register is decremented to zero, the active COUNT1 control register is decremented by one and source address is updated by using the product of element size and the index1. The active element counter COUNT0 control register is then reloaded from the COUNT0 reference register.

The write pointer within the FIFO descriptor is updated only when active frame counter COUNT1 control register is decremented to zero. The pointer is adjusted by adding reference element counter COUNT0. The pointer adjustment is always calculated by modulo FIFO buffer size.

An element counter is one-dimensional when the frame counter COUNT1 register in a transfer entry is smaller or equal to one. In case when reference frame counter COUNT1 is equal to zero, dMAX will ignore delay table and transfer COUNT0 elements from source to the location in the FIFO pointed by the write pointer.

An example of a FIFO write transfer is presented in Figure 1-7. In the figure it is assumed that the FIFO write pointer grows in a counter-clockwise direction when new samples are written to the FIFO. A transfer entry for a FIFO write shown in Figure 1-7 specifies:

- Linear source address (SRC) from which data will be copied to the FIFO. Currently sample (A) is at a memory location pointed to by SRC (Figure 1-7).
- Two source address indexes: sindex0 and sindex1 (for clarity, the indexes are not shown in Figure 1-7).
- Active and reference copy of frame and element counter. At the beginning of the transfer, active and reference frame counters are set to three (COUNT1), while active and reference element counters (COUNT0) are set to two.
- Pointer to a FIFO descriptor. The FIFO descriptor defines the destination FIFO. In case of FIFO write transfer the relevant FIFO descriptor parameters shown in Figure 1-7 are:
  - FIFO size (in number of elements)
  - FIFO base address, and
  - FIFO write pointer
- Pointers to two delay tables (for clarity reasons only one delay table is shown in Figure 1-7). There is one-to-one correspondence between delays from the delay table and taps written to the FIFO. The first table delay corresponds to the first frame; the second delay corresponds to the second frame etc.

Having the frame count equal to three means that in this transfer, three frames will be sorted to taps and stored to the FIFO. Since the element count is equal to two, each tap will have two elements. Source index0 dictates spacing between source elements read within a frame. The frame elements are written to subsequent locations within the circular buffer (FIFO). Location where tap is stored within the FIFO is dictated by the delay tables.

A FIFO write can be either frame-synchronized (a sync event is required to transfer each frame), or fully synchronized (one event can be used to synchronize the whole transfer).

Figure 1-7. Three-Frame FIFO Write Transfer (Prior to Transfer Start)

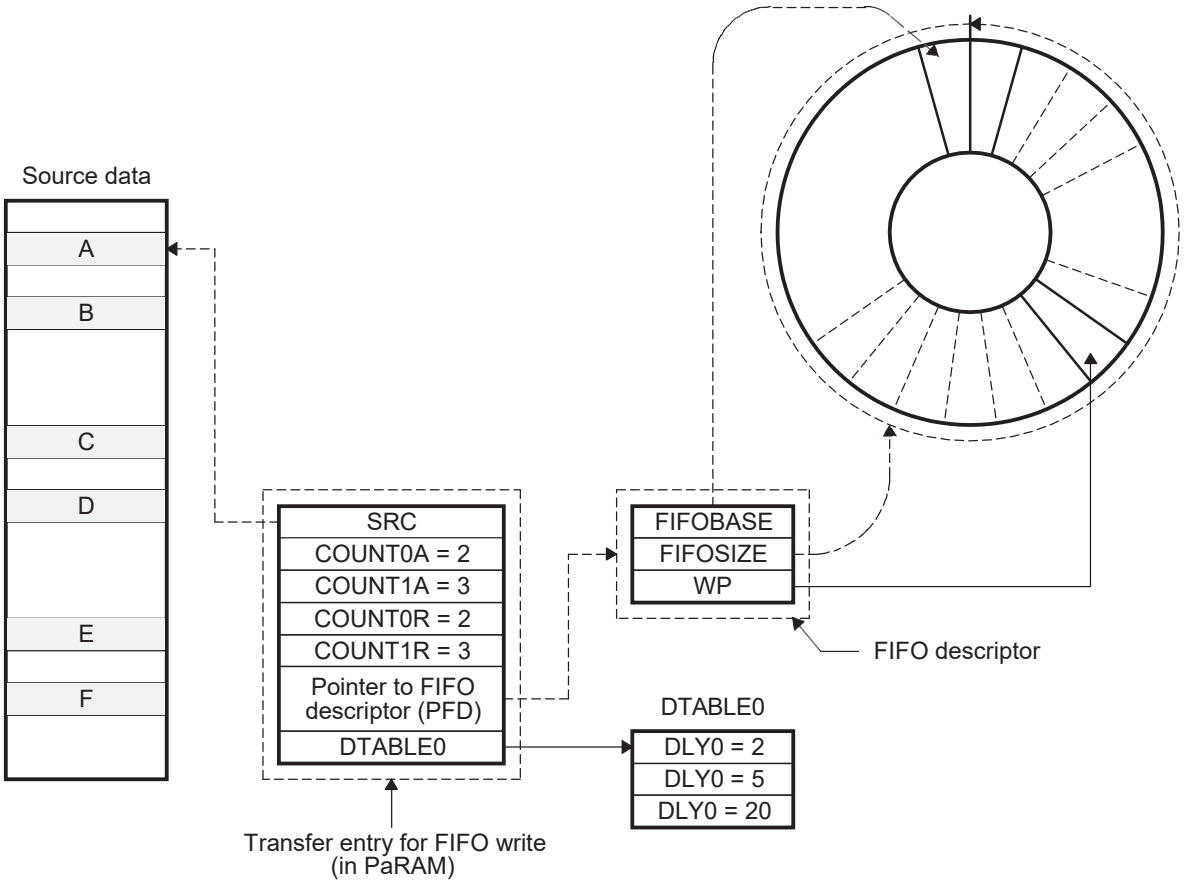


Figure 1-8 shows that since one frame is transferred, the active frame counter (COUNT1) is decremented by one.

Prior to transfer, the active SRC points to the first frame element (A). After transfer of the first element (A), the active element counter (COUNT0) is decremented by one, and the SRC is updated by the product of source index0 and element size. The SRC now points to the second frame element (B). After transfer of the second element (B), COUNT0 is decremented by one. Since COUNT0 is now equal to zero, the SRC is updated by the product of source index1 and element size, and the value of the active element counter is reloaded from the reference element counter.

The first entry from the delay table (DTABLE0) dictates the location within the FIFO where the first frame will be stored. The delay entry specifies destination location relative to the write pointer. Since the first value read from the delay table is equal to two, starting storing point for the first tap will be two samples behind the write pointer.

Before the second frame is transferred, the SRC points to the first element of the second frame (Figure 1-8). After transfer of the first element of the second frame, COUNT0 is decremented by one, and the SRC is updated by the product of source index0 and element size. The SRC now points to the second element of the second frame (D). After transfer of the second element (D), COUNT0 is decremented by one. Since the active element counter is now equal to zero, the SRC is updated by the product of source index1 and element size, and the value of the active element counter is reloaded from the reference element counter.

Figure 1-8. Three-Frame FIFO Write Transfer (After Transfer of the First Frame)

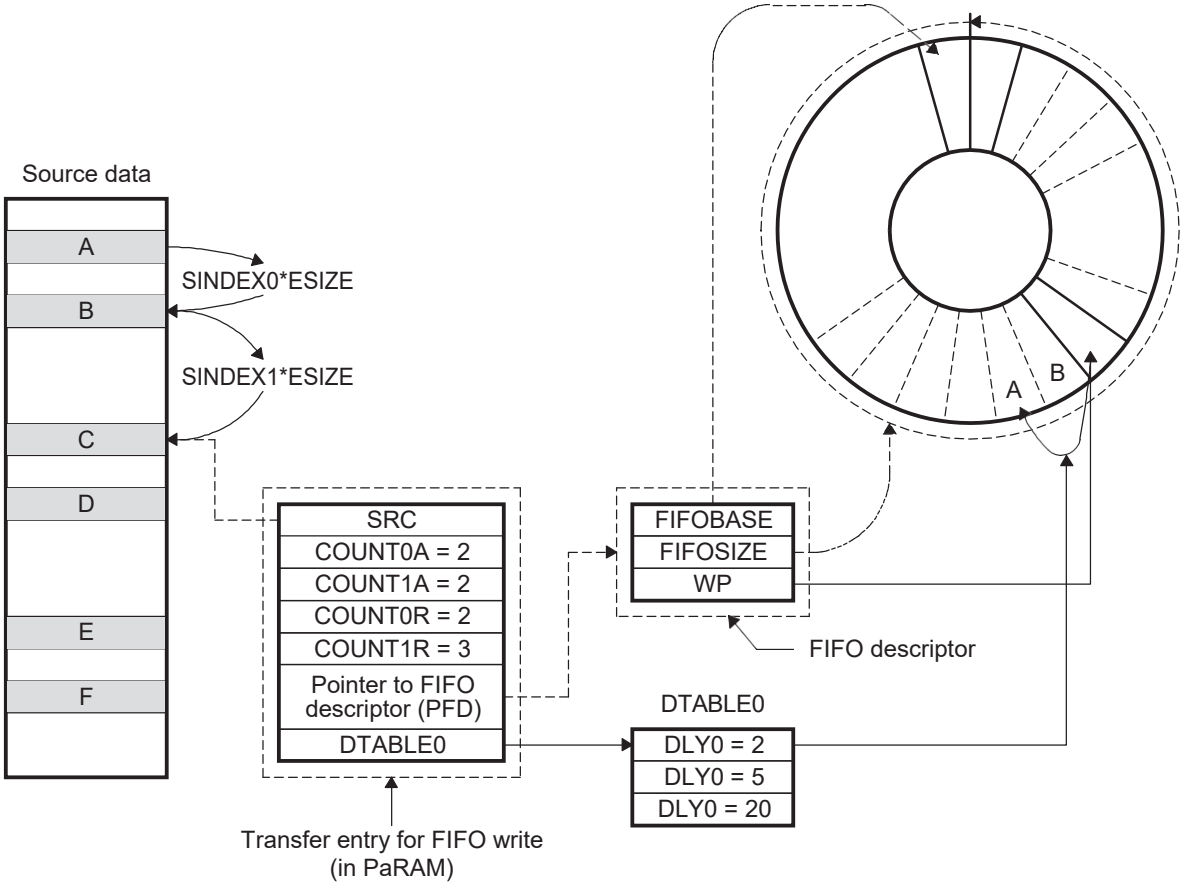
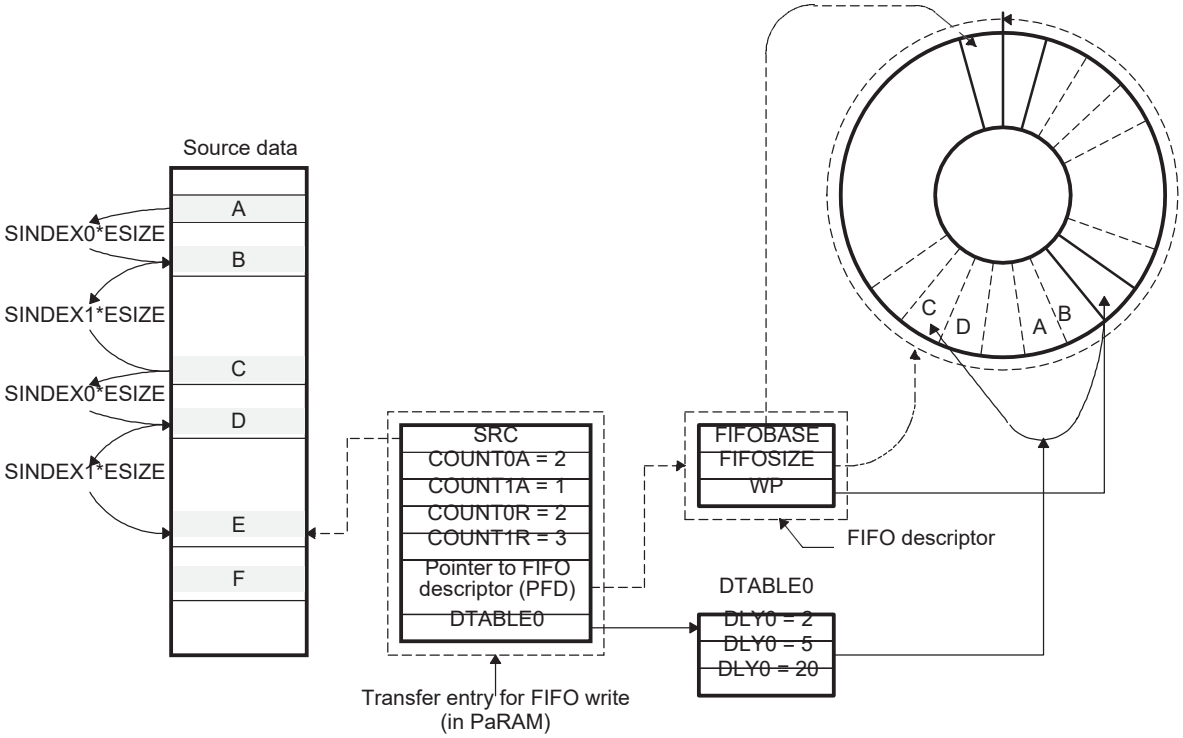


Figure 1-9 shows that after the second frame is transferred, COUNT1 is decremented by one.

The second entry from the delay table (DTABLE0) dictates the location within the FIFO where the second frame will be stored. The delay entry specifies destination location relative to the write pointer. Since the second value read from the delay table is equal to five, the starting storing point for the second tap will be five samples behind the write pointer.

Before the last frame is transferred, the active source address pointer (SRC) points to the first element of the third frame (E) (Figure 1-9). After transfer of the first element (E), COUNT0 is decremented by one, and the SRC is updated by product of source index0 and element size. The SRC now points to the second element of the third frame (F). After transfer of the second element (F), COUNT0 is decremented by one.

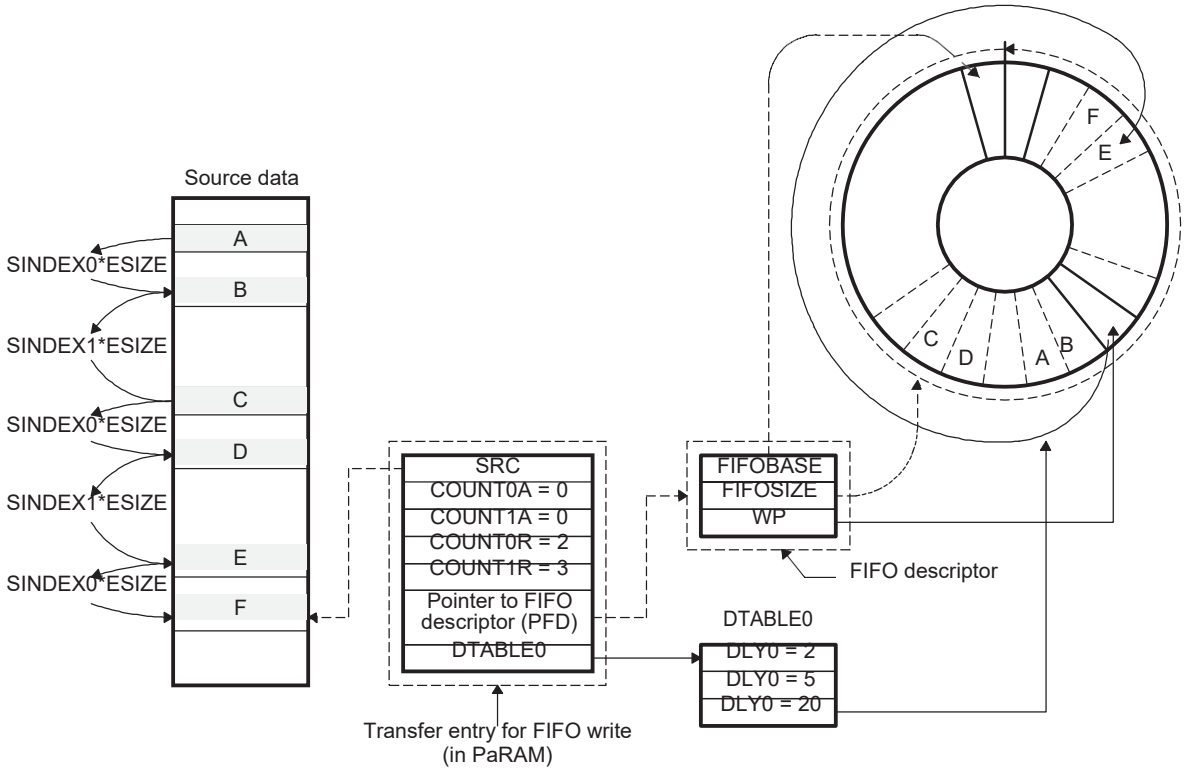
Figure 1-9. Three-Frame FIFO Write Transfer (After Transfer of the Second Frame)



The third entry from the delay table (DTABLE0) dictates the location within the FIFO where the third frame will be stored. The delay entry specifies the destination location relative to the write pointer. Since the third value read from the delay table is equal to 20, the starting storing point for the first tap will be 20 samples behind the write pointer.

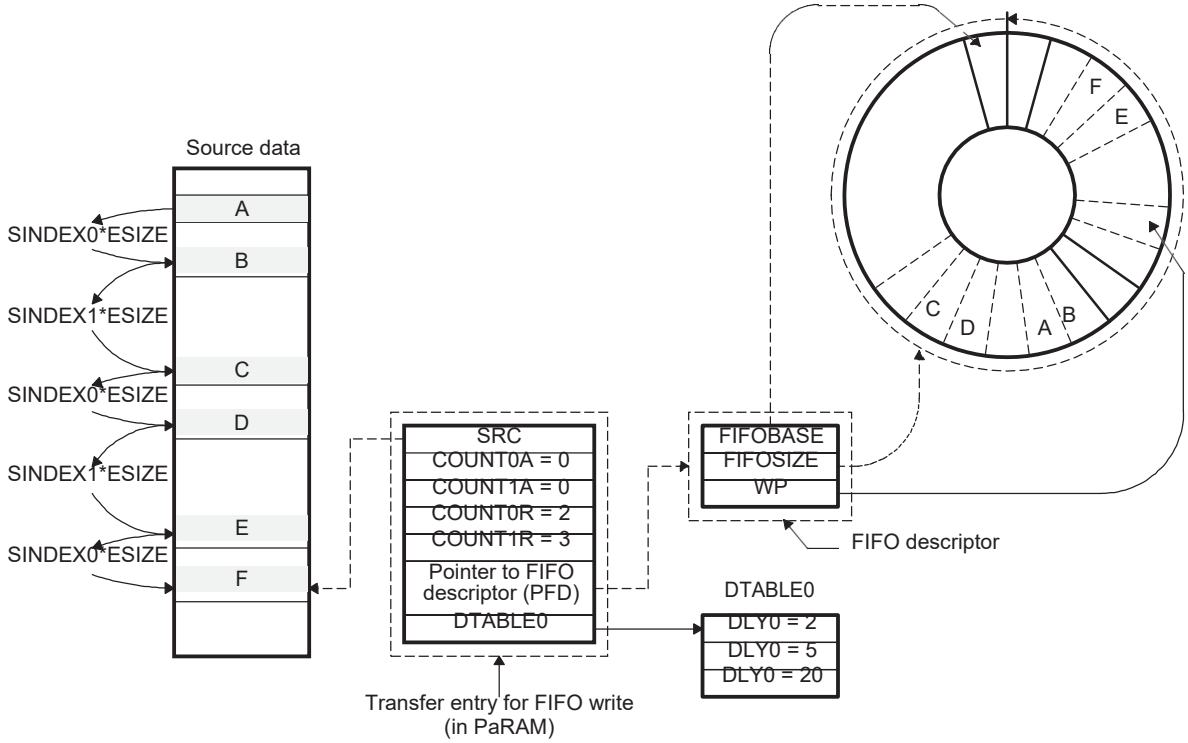
The results of what happens immediately after the last frame is transferred is presented in Figure 1-10. Since a frame is transferred, COUNT1 is decremented by one, and now equals zero.

Figure 1-10. Three-Frame FIFO Write Transfer (Immediately After Transfer of the Third Frame)



Since the active frame counter is equal to zero, the transfer is complete. After completing the whole transfer (frame counter decremented to zero), dMAX automatically updates the FIFO write pointer. Figure 1-11 presents the setup after the transfer is complete. The write pointer is incremented by reference frame size (reference value for COUNT0, which is two in this example). The value of the write pointer is calculated by modulo FIFO size.

Figure 1-11. Three-Frame FIFO Write Transfer (Transfer Complete)



If reload is enabled, the SRC can be reloaded from one of two reference registers, and value of the active element and frame counters can be loaded from the reference counter register. In the example, only one delay table is shown for clarity. If reload is enabled, an appropriate delay table pointer is used with each reload.

When the reference frame counter (COUNT1R) is equal to zero, dMAX ignores the delay table and data is written to FIFO starting from the location pointed to by the write pointer.

1.5.4.2 FIFO Read

A transfer entry for FIFO read includes: an active copy of destination address, a pointer to FIFO descriptor, two destination address indexes, an active and a reference copy of element counters, two sets of reference registers used to reload destination address, and two pointers to delay tables.

An active copy of destination address points to a DSP memory location where data read from the FIFO buffer will be stored. An active copy of destination address is automatically updated by dMAX during a course of a transfer. Instead of fully describing FIFO buffer within the transfer entry a pointer to FIFO descriptor is used (for more detail on the FIFO descriptor see Section 1.4 and Section 2.2). The FIFO descriptor can be located anywhere in the DSP RAM.

An element counter is two-dimensional when the element count register (COUNT1) in a transfer entry is greater than one. In this case, COUNT1 number of taps will be transferred by dMAX from the FIFO to linear memory, and each tap will have COUNT0 elements. Each tap read from the FIFO is a set of COUNT0 consecutive elements.

The tap position within the FIFO is determined by delay table entries. The tap position (tap delay) within the FIFO buffer is expressed in number of samples and is calculated from the FIFO read pointer. The number of entries in the delay table is equal to value of the reference frame count register (COUNT1). There is one-to-one correspondence between delay table entries and taps (each delay table entry is assigned to one tap). The first delay read from the delay table corresponds to the first tap read from the FIFO; the second delay read from the delay table corresponds to the second tap read from the FIFO, etc.

Destination index1 defines the distance, in number of elements, between the last element in a frame and the first element of the next frame; it can be between -32768 and 32767.

The distance between elements within the frame is specified by index0. The destination index0 is expressed in number of elements, and can be between -32768 and 32767. While the active COUNT0 control register is greater than zero, destination address is updated after each element transfer by using the product of element size and the appropriate index0.

When the active COUNT0 control register is decremented to zero, the active COUNT1 control register is decremented by one and destination address is updated by using the product of element size and the index1. The active COUNT0 control register is then reloaded from the COUNT0 reference register.

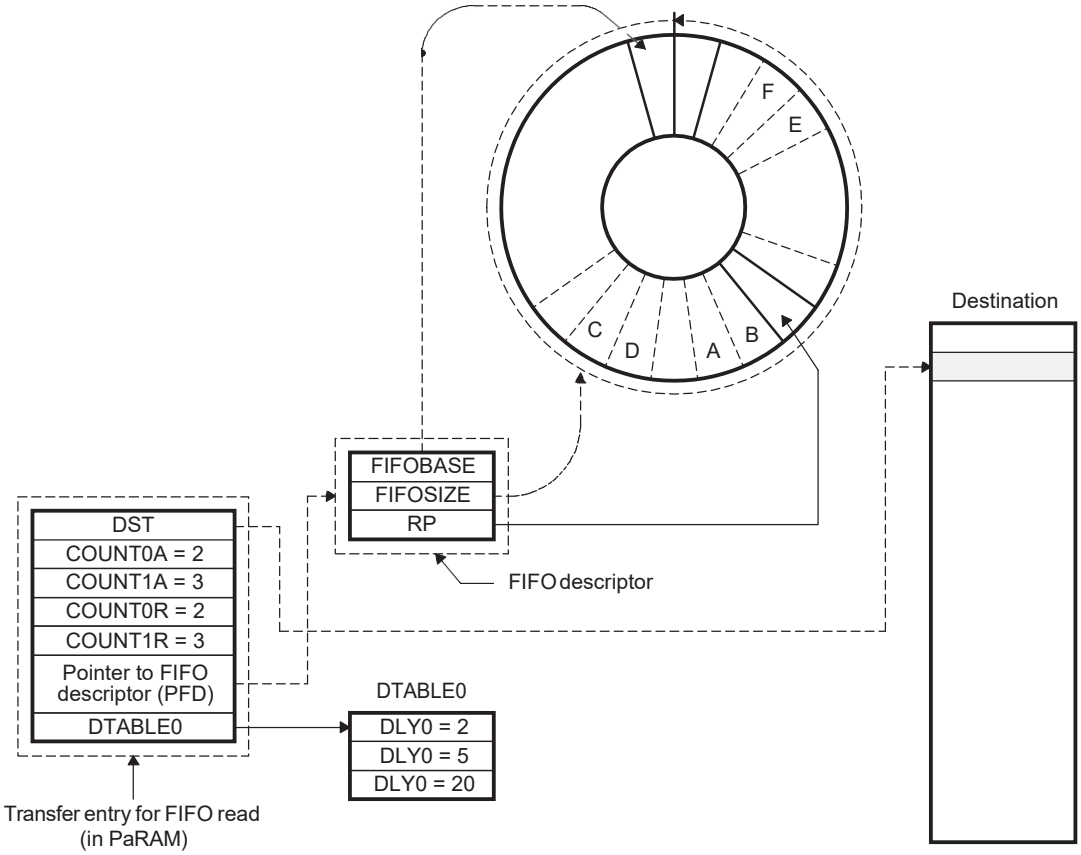
The read pointer within the FIFO descriptor is updated only when the active COUNT1 control register is decremented to zero. The pointer is adjusted by adding COUNT0. The pointer adjustment is always calculated by modulo FIFO buffer size.

An element counter is one-dimensional when the COUNT1 register in a transfer entry is smaller or equal to one. When reference COUNT1 is equal to zero, dMAX will ignore the delay table and transfer COUNT0 elements from a FIFO location pointed by the read pointer to memory location pointed by active destination address register.

An example of a FIFO read transfer is presented in Figure 1-12, where the FIFO read pointer moves in a counter-clockwise direction when samples are read from the FIFO. A transfer entry for FIFO read shown in Figure 1-12 specifies:

- A destination (DST) address to which data is going to be stored after being read from the FIFO.
- Two destination address indexes: dindex0 and dindex1 (for clarity the indexes are not shown in Figure 1-12).
- Active and reference copy of frame and element counter. At the beginning of the transfer, active and reference frame counters are set to three (COUNT1), while active and reference element counters (COUNT0) are set to two.
- Pointer to a FIFO descriptor. The FIFO descriptor defines the source FIFO. In case of FIFO read transfer the relevant FIFO descriptor parameters shown in Figure 1-12 are:
  - FIFO size (in number of elements)
  - FIFO base address, and
  - FIFO read pointer
- Pointers to two delay tables (for clarity, only one delay table is shown in Figure 1-12). There is one-to-one correspondence between delays from the delay table and taps read from the FIFO. The first table delay corresponds to the first tap; the second delay corresponds to the second tap, etc.

Figure 1-12. Three-Frame FIFO Read (Prior to Transfer Start)

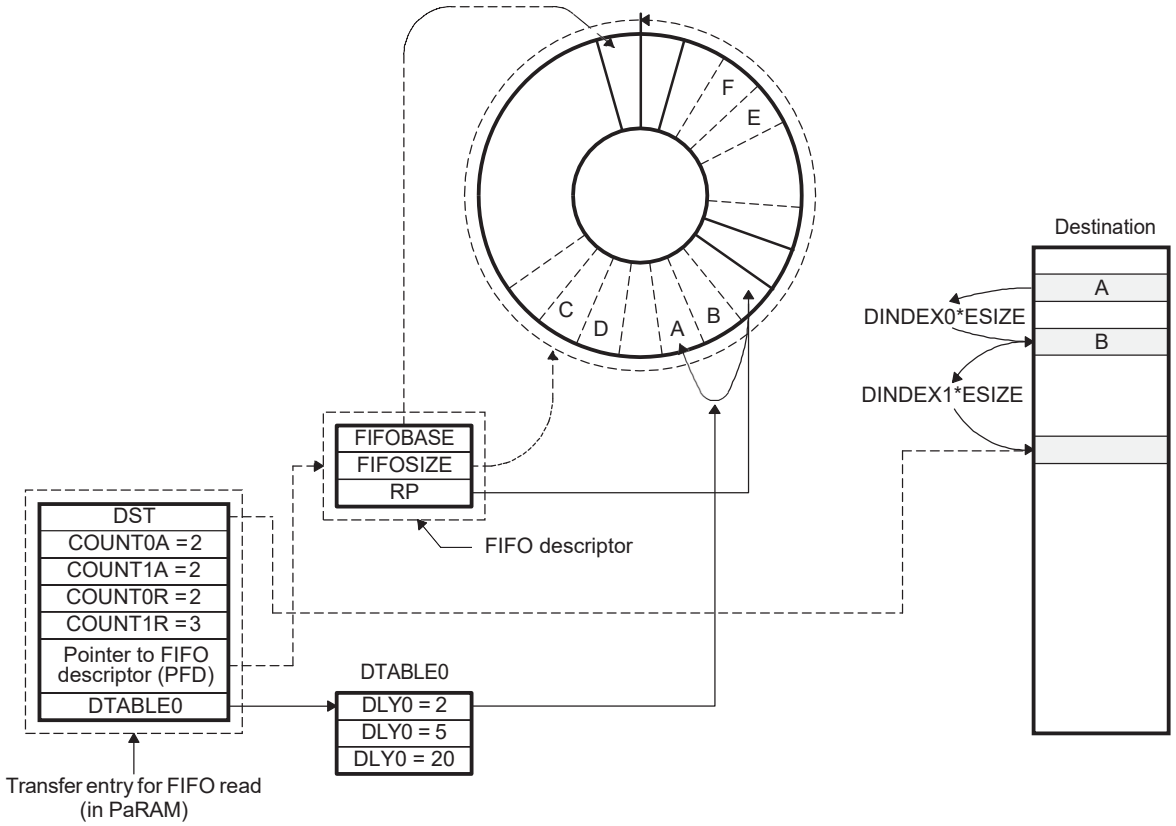


Having frame count equal to three means that in this transfer, three taps will be pulled out from the FIFO buffer and sorted out to the linear DSP memory. Since the element count is equal to two, each tap will have two elements. The samples within the tap are read from subsequent locations within the FIFO. The FIFO location from which the tap is retrieved is dictated by the delay tables. Destination index0 dictates spacing between frame elements when dMAX stores samples read from the FIFO to the linear DSP memory.

A FIFO read can be either frame-synchronized (a sync event is required to transfer each frame), or one event can be used to synchronize the whole transfer.

Figure 1-13 shows the results of the first frame being transferred. Since one frame is transferred, the active frame counter (COUNT1) is decremented by one.

Figure 1-13. Three-Frame FIFO Read (After Reading the First Tap)



The first entry from the delay table (DTABLE0) dictates location within the FIFO from which the first tap will be read. The delay entry specifies the source location relative to the read pointer. Since the first value read from the delay table is equal to two, the starting point for the first tap will be two samples behind the read pointer. Tap size is equal to frame size and it is equal to two, so the first tap read from the FIFO consists of elements (A) and (B).

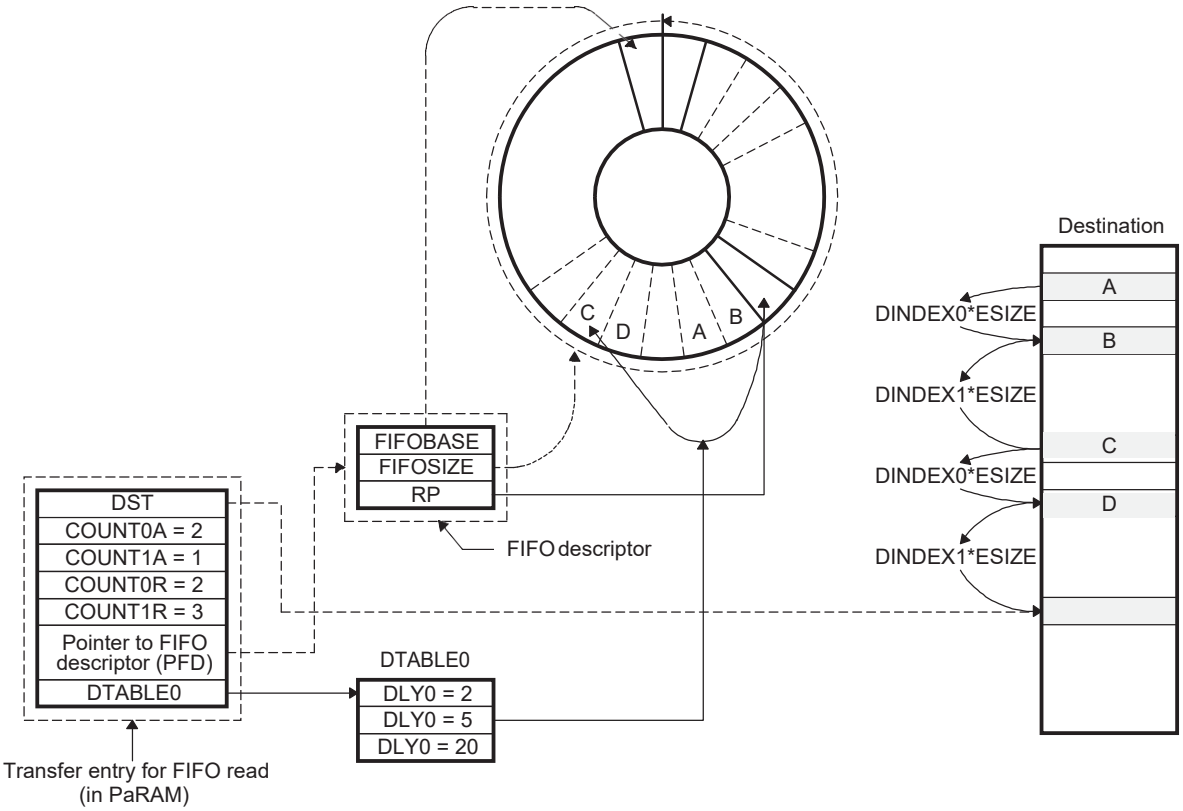
The first tap element (A) is transferred to the location pointed to by the active destination address. After transfer of the first element (A), the active element counter (COUNT0) is decremented by one, the active destination address (DST) is updated by product of destination index0 and element size, and the second tap element (B) is transferred. After transfer of the second element (B), the active element counter (COUNT0) is decremented by one. Since the active element counter is now equal to zero, the DST is updated by product of destination index1 and element size, and value of the active element counter is reloaded from the reference element counter.

The second entry from the delay table (DTABLE0) dictates location within the FIFO from which the second tap will be read. The delay entry specifies source location relative to the read pointer. Since the second value read from the delay table is equal to five, starting point for the second tap will be five samples behind the read pointer. Tap size is equal to frame size and it is equal to two, so the second tap read from the FIFO consists of elements (C) and (D).

The first element of the second tap (C) is transferred to the location pointed by the active destination address. After transfer of the first element (C) the active element counter (COUNT0) is decremented by one, and the active destination address DST is updated by product of destination index0 and element size, and the second tap element (D) is transferred. After transfer of the second element of the second tap (D), the active element counter (COUNT0) is decremented by one. Since the active element counter is now equal to zero, the active destination address DST is updated by product of destination index1 and element size, and value of the active element counter is reloaded from the reference element counter.

Figure 1-14 shows what happens immediately after the frame transfer. Since another frame is transferred, the active frame counter (COUNT1) is decremented by one.

Figure 1-14. Three-Tap FIFO Read (After Reading the Second Tap)

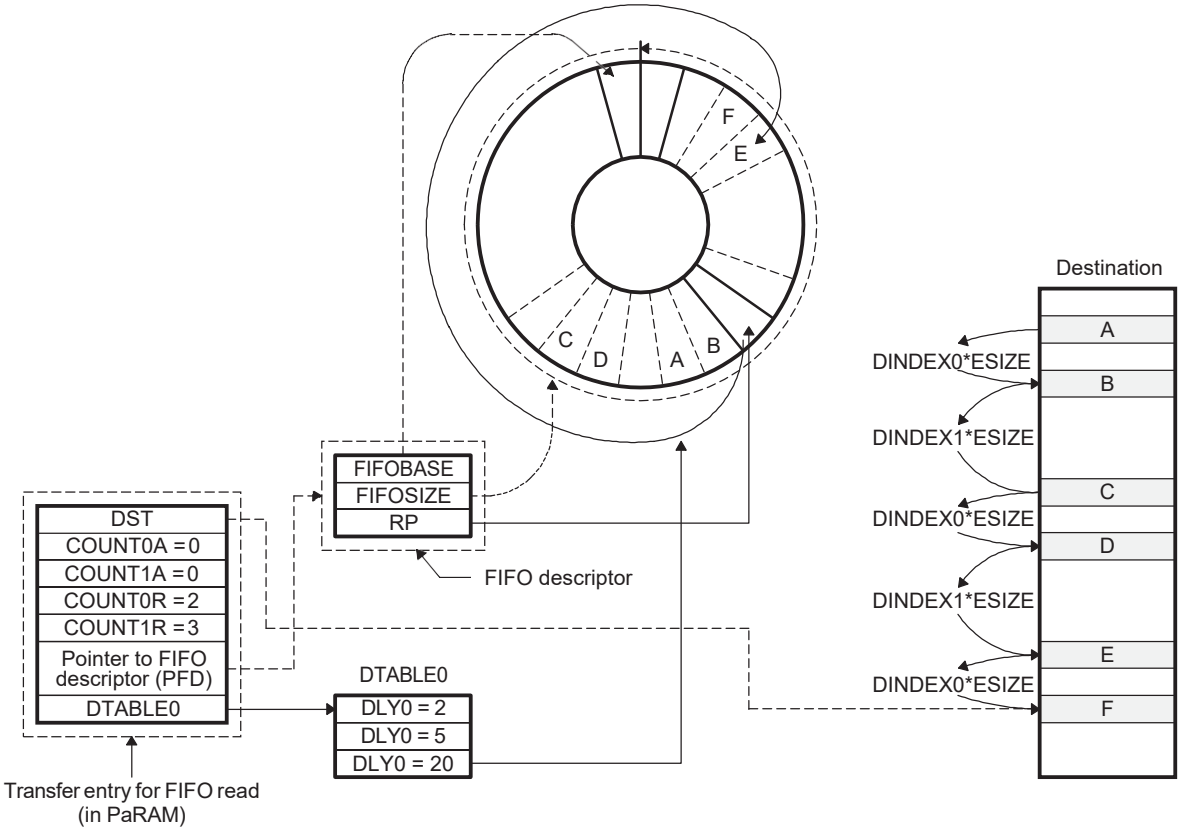


The third delay from the delay table (DTABLE0) dictates location within the FIFO from which the third tap will be read. The delay specifies source location relative to the read pointer. Since the third value read from the delay table is equal to 20, starting point for the third tap will be 20 samples behind the read pointer. Tap size is equal to frame size and it is equal to two, so the third tap read from the FIFO consists of elements (E) and (F).

The first element of the first tap (E) is transferred to the location pointed by the active destination address. After transfer of the first element (E), the active element counter (COUNT0) is decremented by one, and the active destination address DST is updated by product of destination index0 and element size, and the second element of the third tap (F) is transferred. After transfer of the second element (F) the active element counter (COUNT0) is decremented by one.

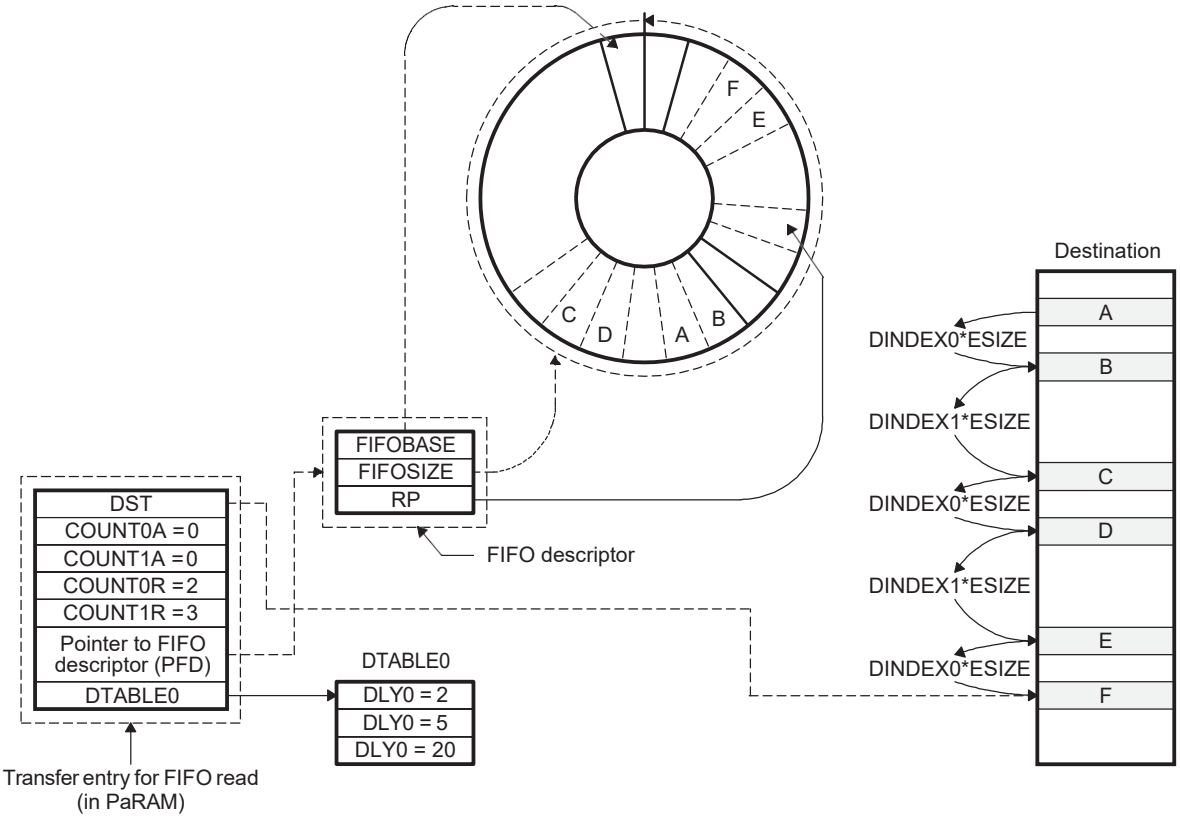
Figure 1-15 shows what happens immediately after the frame transfer. Since another frame is transferred, the active frame counter (COUNT1) is decremented by one and equals zero.

Figure 1-15. Three-Tap FIFO Read (Immediately After Reading the Third Tap)



When the active counter is equal to zero, the transfer is complete. After completing the whole transfer (frame counter decremented to zero), dMAX automatically updates the FIFO read pointer. Figure 1-16 shows the setup after the transfer is complete. The read pointer is incremented by reference frame size (reference value for COUNT0, which is two in this example). The value of the read pointer is calculated by modulo FIFO size.

Figure 1-16. Three-Tap FIFO Read (Transfer Complete)



If reload is enabled, the active destination address DST can be reloaded from one of two reference registers, and the value of the active element and frame counters can be loaded from the reference counter register. In the example, only one delay table is shown for clarity. If reload is enabled, an appropriate delay table pointer is used with each reload.

If the reference frame counter (COUNT1) is equal to zero, dMAX ignores the delay table and FIFO read starts at the location pointed by the read pointer .

1.5.5 One-Dimensional Burst (1DN) Transfers

The one-dimensional burst transfer (1DN) is optimized for doing fast transfer of sequential data from one memory location to the other. A 1DN transfer is specified by its transfer entry in the PaRAM. The transfer entry describes the transfer and contains pointer to the source and the destination address. The transfer entry also contains the CNT that defines the number of bytes that need to be transferred.

The 1DN transfer happens in bursts. A burst is synonymous to a quantum transfer and describes the number of bytes that the dMAX will transfer before it checks for a new event. If an event arrives while dMAX is performing a burst transfer, the event will be serviced after the burst has been completed. The burst length can be configured to be between 1-64 bytes. Unlike the general purpose transfers, the 1DN transfer does not yield to a higher priority pending transfer after transferring one burst. The 1DN transfer

only yields after a burst if there is a new event that has arrived. The 1DN transfer yields to a higher priority pending transfer after performing N bursts. The number of bursts (N) after which the 1DN transfer yields to a higher priority pending transfer is user configurable. The number of bursts can be configured in the transfer entry. The 1DN transfer can thus block a higher priority pending transfer and therefore should be used with caution.

During the course of a transfer, dMAX updates the parameters within the transfer entry (source address, destination address, byte counter and number of bursts). The maximum number of bytes that can be moved by a 1DN transfer is 65535.

After receiving an event, CNT number of bytes are transferred. A 1DN transfer is complete when the CNT field is decremented to zero.

1.5.6 SPI Slave Transfer

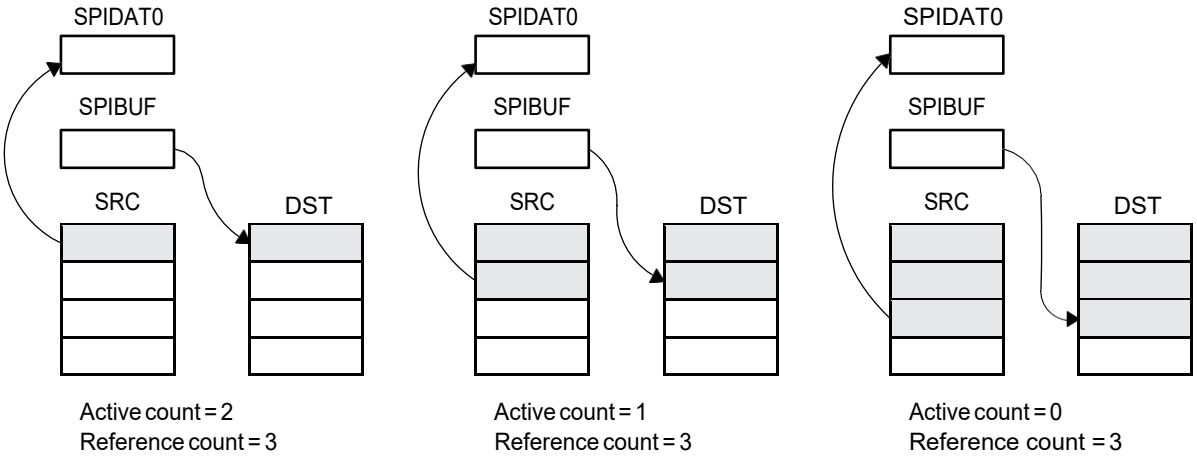
SPI slave transfer allows for servicing the SPI peripheral when used in slave mode. The peripheral servicing requires that for a given SPI event, one element be read from the SPI input register and an element be written to the SPI output shift register. The SPI slave transfer supports this functionality. The 672x DSP supports two SPI peripherals. The SPI peripheral to use for a give SPI slave transfer is configured in the event entry.

A set of active registers for a SPI slave transfer includes source address, destination address, and element counter. During course of a transfer, the dMAX controller updates the active parameters within transfer entry (active source address, active destination address and active element counter). The element size can be configured to be 8-bit or 16-bit in the event entry.

For each input event, one element is read from the SPI input shift register (SPIBUF) and is stored in the destination address. Also, one element is read from the input address and moved to the SPI output shift register (SPIDAT0). The input address and the output address are incremented by one after servicing each event. There are two sets of reference registers that specify reference values for source and destination address; there is also one reference counter register. During a transfer, dMAX uses the active register set.

A SPI slave transfer is graphically presented in Figure 1-17. The example presents transfer phases in a case when active count and reference count is set to 3.

Figure 1-17. SPI Slave Transfer



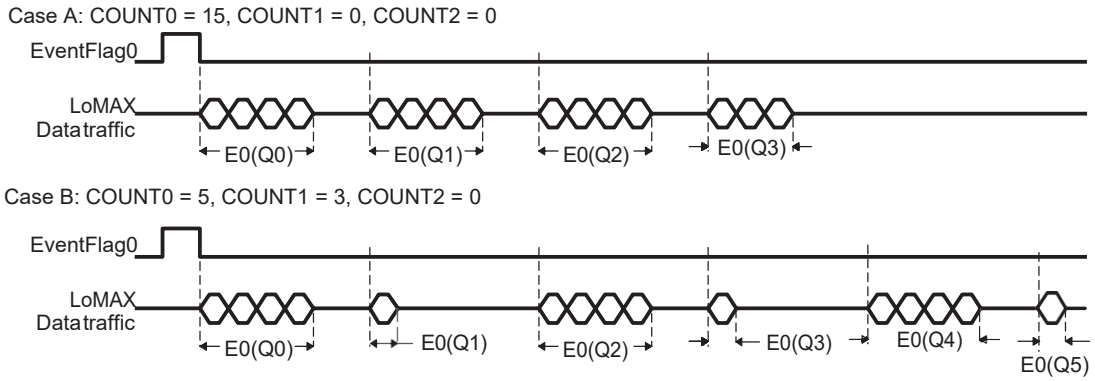
The maximum number of elements that can be transferred by a SPI slave transfer is 65535. A SPI slave transfer is complete when the active element count field is decremented to zero. If reload is enabled at the end of the transfer, active address registers are loaded from one of two sets of reference address registers. The address reference register set used during reload is specified by the PP bit within the transfer entry. If PP bit is 0, address reference register set 1 is loaded to the active address registers. If PP bit is 1, address reference register set 0 is loaded to the active address registers. The PP bit is toggled after loading the reference register set.

1.6 Quantum Transfers

Splitting a transfer into a number of sub-transfers allows the dMAX controller to quickly process higher-priority requests that arrived during the transfer, and therefore reduce the system latency. To improve system latency, dMAX always moves data in small sub-transfers called quantum transfers. The number of elements transferred during a quantum transfer is always less than or equal to the quantum transfer size limit (QTSL) which is specified in a transfer entry. A transfer longer than the quantum transfer size limit is called a long transfer.

An example of a long transfer is presented in Figure 1-18. In this example, an event is triggering a transfer of 15 elements, and the QTSL is set to four in the transfer entry.

Figure 1-18. An Example of a Long Transfer (Transfer Size is Equal to 15 Elements and Quantum Transfer Limit Size is Set to 4)



Transfer size of E0 is 15 elements
In the Event Entry QTSL is set to 4 elements
In the Event Entry SYNC = '1'

In CASE A the transfer is one-dimensional
In CASE B the transfer is two-dimensional

Legend:

Ex(Qy):	Quantum transfer (y), is a part of a long transfer initiated by an event on the event signal number (x).
Event Flagx:	A flag that corresponds to event number (x), from the EFR register

The QTSL is an upper bound to the number of elements that dMAX can transfer from source to destination without interruption. During a quantum transfer, dMAX may move from source to destination a number of elements that will not exceed what is specified in the QTSL bit field of the event entry. For general purpose data transfer, the size of the quantum transfer is equal to the minimum between the QTSL and the active element counter, COUNT0A.

In the example presented in the upper panel of Figure 1-18 (CASE A), a transfer will be broken into four quantum transfers: E0(Q0), E0(Q1), E0(Q2), and E0(Q3). The dMAX controller transfers four elements within each of the first three quantum transfers, and during the last quantum transfer, the remaining three elements are moved.

In the example presented in the lower panel of Figure 1-18 (CASE B), a transfer will be broken into six quantum transfers. In the example, the frame size is equal to five and the QTSL is equal to four, so the first quantum transfer moves the first four elements of a frame while the second quantum transfer moves the remaining one element. Therefore, dMAX, in CASE B, breaks transfer of each of three frames into two quantum transfers.

A FIFO transfer that crosses the upper FIFO boundary is always split into two quantum transfers (even when the transfer size is equal to or smaller than the QTSL specified in the event entry). The first of the two quantum transfers will fill the gap between the FIFO pointer and the upper FIFO boundary; the second quantum transfer will wrap around and move remaining elements starting from the lower FIFO boundary.

When a new event arrives while dMAX is performing a quantum transfer (and the quantum transfer is a part of a long transfer started by the event that belongs to the same priority group as the new event), the new event will be serviced after completion of the quantum transfer that is in progress. After servicing the event, if there are no other pending events in the same priority group, dMAX will continue processing previously interrupted data transfer (assuming that the event doesn't start another long transfer with higher priority than the interrupted long transfer).

If there are no pending events and two long transfers in progress are pending (assuming the two transfers belong to events from the same priority group), the transfer that corresponds to a higher priority event is going to be serviced first.

The QTSL is programmable and can be different for different channels. By decreasing the QTSL, overall dMAX latency will decrease; by increasing it, the dMAX data throughput will increase.

1.7 Element Size and Alignment

The element size can be independently set for each channel. The dMAX controller supports three different element size settings: 8-bit, 16-bit, and 32-bit.

The transfer counter always specifies the number of elements to be transferred. Source, destination indexes, and buffer size are also specified in number of elements.

CAUTION

Only 32-bit transfers are supported to and from the McASP DMA ports.

1.8 Source/Destination Address Updates

Since all indexes are specified in number of elements, in order to calculate correct address, the index value must be multiplied by element size. Indexes can take positive and negative values. After each transfer, source and destination addresses are updated by the product of the appropriate index and element size.

1.9 Reloading dMAX Transfers

The transfer entry contains a set of active registers, two sets of address reference registers, and only one reference counter register. In the active register set, dMAX maintains information about the state of a transfer (current source and destination addresses and current element count).

The reload option can be used with all transfer types supported by dMAX. If reload is enabled after a transfer is complete, a set of address reference registers will be loaded to the active register set. A new transfer will be automatically kicked off after a new event is received. A different set of address reference registers will be loaded into the active set of address registers each time a transfer completes. The PP control bit in the transfer entry indicates which reference set was last loaded in the set of active registers. If the PP bit in the transfer entry is cleared to zero, during the next reload, dMAX will move Reload1 set of registers into the active registers and set the PP bit to one. If the PP bit in the transfer entry is set to one, during the next reload, dMAX will move Reload0 set of registers into the active registers, and will clear the PP bit to zero.

For FIFO transfers, a reference register set consists of two linear address references and one reference element counter register. A FIFO transfer entry also contains two delay table pointers. There is one-to-one correspondence between reference address registers and delay-table pointers. The purpose of associating a delay-table pointer with a reference register set is to be able to alternate or use a different delay table every time a new transfer is initiated by reload. If reload is enabled, a different address reference register will be loaded in the active address register every time transfer completes. The PP control bit in the transfer entry indicates which reference register was loaded in the active register and which delay table is used in the current transfer.

1.10 dMAX Interrupt Generation

The dMAX controller can use:

- A dedicated interrupt line (INT8) to notify the CPU about data transfer completion.
- A dedicated interrupt line (INT7) to indicate the FIFO status conditions to the CPU.
- any event presented in Table 1-2 to trigger a CPU interrupt by using one of six interrupt lines (see Section 2.1.1.3).

1.10.1 Using an Event to Initiate a CPU Interrupt

An event from a peripheral or an external event can be used to generate an interrupt to the CPU via dMAX. In order to generate an interrupt, an event entry that corresponds to the event must be programmed appropriately. If an event is used to generate a CPU interrupt, a transfer entry is not required, and the event entry only needs to specify which interrupt line should be used to trigger the CPU interrupt.

1.10.2 End of Transfer Notification Interrupt to the CPU

The dMAX controller is responsible for generating transfer-complete interrupts to the CPU; it uses a single interrupt line to the CPU on behalf of all 16 possible channels. The various control registers and bit fields facilitate dMAX interrupt generation.

Upon completion of the entire channel transfer (counter for all dimensions expired to zero), the transfer-complete interrupt applies. The TCINT bit field within the event entry enables dMAX to notify the CPU at the end of a transfer. At the end of a transfer, if the TCINT bit field in the event entry is set, dMAX will set a bit in one of two dMAX Transfer Complete Registers (DTCR). Prior to setting the bit in the register, dMAX will verify if the bit was cleared; if the bit is zero, it will generate an interrupt to the CPU. The TCC value programmed in the event entry dictates the DTCR bit number that gets set. In the dMAX controller, the TCC field specifies the transfer complete pending bit, with values between 0-15. The TCC codes 0-7 correspond to bits 0-7 of the DTCR0, while TCC codes 8-15 correspond to bits 0-7 of the DTCR1.

If the TCC pending bit is not cleared by the CPU before dMAX attempts to set the bit again, dMAX will not generate a CPU interrupt.

If the TCINT bit is not set in the event entry, the TCC code will not be set and a CPU interrupt will not be generated at the end of a transfer.

If the TCINT bit in the event entry is set, at the end of transfer, dMAX will trigger a CPU interrupt (INT8), and if the interrupt is enabled, its interrupt service routine is executed.

The transfer complete code can be programmed to any value for any dMAX channel; no direct relation between the channel number and the transfer complete code value is needed. This allows multiple channels having the same transfer complete code value to cause the CPU to execute the same ISR (for different channels). Alternatively, the same channel can set multiple complete codes depending on the transfers performed.

1.10.2.1 Alternate Transfer Complete Interrupt

The dMAX controller allows an interrupt upon completion of intermediate transfers in a block, and enabled by the alternate transfer complete interrupt (ATCINT) field in the event entry. When it is enabled, an interrupt is set (and sent to the CPU) upon completion of each intermediate transfer of the current channel. At the end of an intermediate transfer, if the ATCINT bit field in the event entry is set, the dMAX controller will set a bit in one of two dMAX Transfer Complete Registers (DTCR). Prior to setting the bit in the DTCR, the dMAX controller will verify that the bit was cleared; only if the bit is zero will it generate an interrupt to the CPU. The TCC value programmed in the event entry dictates the DTCR bit number that gets set. In the dMAX controller, the TCC field specifies the transfer complete pending bit, with values between 0-15. The TCC codes 0-7 correspond to bits 0-7 of the DTCR0 register, while TCC codes 8-15 correspond to bits 0-7 of the DTCR1 register.

If the TCC pending bit is not cleared by the CPU before dMAX attempts to set the bit again, dMAX will not generate a CPU interrupt.

If the ATCINT bit is not set in the event entry, the TCC code will not be set and a CPU interrupt will not be generated upon completion of each intermediate transfer of the current channel.

The alternate transfer complete interrupt is not applicable if the whole transfer is synchronized to one synchronizing event (SYNC bit field in the event entry equal to one).

1.10.2.2 Processing of End of Transfer dMAX Interrupt by the CPU

Since the dMAX controller tracks the completion of the dMAX channel transfer, it sets the appropriate bit in the DTCR as per the transfer complete pending bit specified. The CPU ISR should read either dMAX Event Status Register (DESR) or DTCR to determine what, if any, events/channels have completed and perform the necessary operations. The read-only DESR will have a lower overhead than reading the DTCR, since the DESR is placed inside the CPU for faster access. The pending bit can only be cleared by writing to the DTCR. Writing a 1 to the relevant bit can clear the DTCR bits; writing a 0 has no effect.

By the time one interrupt is serviced, many others could have occurred and relevant bits set in the DTCR. Each of these bits in the DTCR would probably need different types of service. The ISR should check for all pending interrupts and continue until all the posted interrupts are serviced.

1.10.3 FIFO Status Notification Interrupt

1.10.3.1 FIFO Buffer Watermarks

A FIFO buffer descriptor contains two watermarks; empty mark (EMARK), and full mark (FMARK). If the number of fresh samples in the circular buffer becomes equal or drops below EMARK, a FIFO empty condition will be signaled to the CPU. If the number of fresh samples in the circular buffer becomes equal or rises above FMARK, a FIFO full condition will be signaled to the CPU. The watermark notifications can be enabled or disabled by writing to a bit in the event entry describing a FIFO transfer.

To signal a watermark condition to the CPU, the dMAX sets the FIFO status code pending bit in the dMAX FIFO Status Register (DFSR) and triggers a FIFO status interrupt to the CPU (INT7). The FIFO status code bit to be used for this notification is dictated by the value specified in the FIFO buffer descriptor. In order to receive further watermark notifications from the particular buffer, the CPU must clear the FIFO status code pending bit in the DFSR.

The Full Mark Status Code (FMSC) value programmed in the FIFO descriptor dictates the DFSR bit number that gets set to indicate a FIFO full status. In the dMAX controller, the FMSC field specifies the FIFO full status pending bit, with values between 0-15. The FMSC codes 0-7 correspond to bits 0-7 of the DFSR0, while FMSC codes 8-15 correspond to bits 0-7 of the DFSR1.

The Empty Mark Status Code (EMSC) value programmed in the FIFO descriptor dictates the DFSR bit number that gets set to indicate a FIFO empty status. The EMSC field specifies the FIFO empty status pending bit, with values between 0-15. The EMSC codes 0-7 correspond to bits 0-7 of the DFSR0, while EMSC codes 8-15 correspond to bits 0-7 of the DFSR1.

1.10.3.2 FIFO Buffer Error Notifications

The FIFO errors are discussed in Section 1.4.2. In case of a FIFO error, dMAX will abort the transfer. The controller will notify the CPU by triggering a FIFO status interrupt, setting an appropriate error flag in the FIFO descriptor, and by setting both status flags assigned to the FIFO (EMSF and FMSF FIFO status flags are set in the DFSR0 and DFSR1).

1.10.4 dMAX NMI Interrupt

dMAX has an interrupt line, dMAX NMI, that is hooked to the CPU INT1.interrupt. dMAX will never assert this dMAX NMI interrupt to the CPU.

1.11 Emulation Operation

During debug using the emulator, the CPU may be halted on an execute packet boundary for single stepping, benchmarking, profiling, or other debug uses. During an emulation halt, dMAX operations continue; toggling bits in the DETR will not trigger dMAX events.

1.12 Event Encoder

The event encoder uses information from two registers, DEHPR and DELPR, to sort all events in two priority groups. The event encoder then ranks all events within one group according to their event numbers (lower event number, higher event priority within its group). The highest priority event from the high-priority group and the highest priority event from the low-priority group can be then processed concurrently by dMAX.

A new event always takes priority over a long transfer. The event numbers are only used for arbitration among multiple pending events. For example, if there are multiple new events and multiple long transfers all pending then the new event with the lowest event number (i.e., highest priority) will be serviced first. Once all new events have been serviced, then the long transfer with the lowest number/highest priority will be transferred next.

1.12.1 Synchronization of dMAX Events

The dMAX Event Flag Register (DEFR) captures up to 31 separate events; therefore, it is possible for events to occur simultaneously on the dMAX event inputs. In such cases, the event encoder resolves the order of processing. This mechanism sorts simultaneous events and sets the priority of the events.

The dMAX controller sorts all events into two priority groups to which an event can belong: high priority and low priority. Two registers, the DEHPR and DELPR, establish the association of an event and priority group. Bits in the DER and DEFR are assigned to the events according to their event numbers (bit n in thesen registers correspond to event number n). There is a one-to-one correspondence between the event flags in the DEFR and 32 bits in the DEHPR. Setting a bit in the DEHPR puts the corresponding event in the high-priority event group. There is also a one-to-one correspondence between the event flags in the DEFR, and the 32 bits in the DELPR. Setting a bit in the DELPR puts the corresponding event in the low-priority event group.

For events arriving simultaneously within the same priority group, priority is determined by their event number (an event with lower event number has a higher priority within its group).

dMAX can simultaneously process one event from each priority group. Therefore, the two highest priority events (one from each group) can be processed at the same time.

Table 1-2 lists how the synchronization events are associated with event numbers in dMAX.

Table 1-2. dMAX Channel Synchronization Events

Event Number	Event Acronym	Address Offset in the Event Entry Table	Event Description
0	DETR[0]	0x00	The CPU triggers the event by creating an appropriate transition (edge) on bit0 in the DETR register.
1	DETR[16]	0x04	The CPU triggers the event by creating an appropriate transition (edge) on bit16 in the DETR register.
2	RTIREQ0	0x08	RTI DMA REQ[0]
3	RTIREQ1	0x0C	RTI DMA REQ[1]
4	MCASP0TX	0x10	MCASP0 TX DMA REQ
5	MCASP0RX	0x14	MCASP0 RX DMA REQ
6	MCASP1TX	0x18	MCASP1 TX DMA REQ
7	MCASP1RX	0x1C	MCASP1 RX DMA REQ
8	MCASP2TX	0x20	MCASP2 TX DMA REQ
9	MCASP2RX	0x24	MCASP2 RX DMA REQ
10	DETR[1]	0x28	The CPU triggers the event by creating an appropriate transition (edge) on bit1 in the DETR register.
11	DETR[17]	0x2C	The CPU triggers the event by creating an appropriate transition (edge) on bit17 in the DETR register.
12	UHPIINT	0x30	UHPI CPU_INT
13	SPI0RX	0x34	SPI0 DMA_RX_REQ
14	SPI1RX	0x38	SPI1 DMA_RX_REQ
15	RTIREQ2	0x3C	RTI DMA REQ[2]
16	RTIREQ3	0x40	RTI DMA REQ[3]
17	DETR[2]	0x44	The CPU triggers the event by creating an appropriate transition (edge) on bit2 in the DETR register.
18	DETR[18]	0x48	The CPU triggers the event by creating an appropriate transition (edge) on bit18 in the DETR register.
19	I2C0XEVT	0x4C	I2C 0 Transmit Event
20	I2C0REVT	0x50	I2C 0 Receive Event
21	I2C1XEVT	0x54	I2C 1 Transmit Event
22	I2C1REVT	0x58	I2C 1 Receive Event
23	DETR[3]	0x5C	The CPU triggers the event by creating an appropriate transition (edge) on bit3 in the DETR register.
24	DETR[19]	0x60	The CPU triggers the event by creating an appropriate transition (edge) on bit19 in the DETR register.
25	Reserved	0x64	Reserved
26	MCASP0ERR	0x68	AMUTEIN0 or McASP0 TX INT or McASP0 RX INT (error on MCASP0)
27	MCASP1ERR	0x6C	AMUTEIN1 or McASP1 TX INT or McASP1 RX INT (error on MCASP1)
28	MCASP2ERR	0x70	AMUTEIN2 or McASP2 TX INT or McASP2 RX INT (error on MCASP2)
29	OVLREQ[0/1]	0x74	Error on RTI
30	DETR[20]	0x78	The CPU triggers the event by creating an appropriate transition (edge) on bit20 in the DETR register.
31	DETR[21]	0x7C	The CPU triggers the event by creating an appropriate transition (edge) on bit21 in the DETR register.

1.12.2 Event Priority Processing Within the Same Event Priority Group

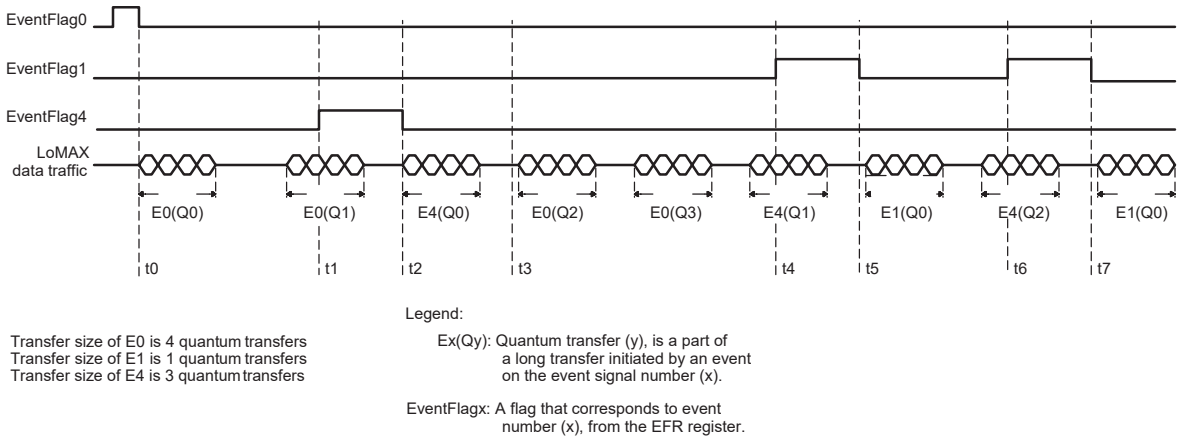
Within a priority group, events are prioritized according to the event numbers (lower event number has higher priority).

Long data transfers are always divided into a number of quantum transfers and serviced in the background. A long data transfer will always have a lower priority than a new event. If a new event arrives, the new event will be given higher priority than the long transfer. This is true even if the new event priority is lower than the priority of the event that triggered the long data transfer.

If an event arrives while dMAX is performing a quantum transfer (sub-transfer of a long data transfer), the new event will be serviced immediately after completion of the quantum transfer that is in progress. After servicing the new event, if there are no other pending events, dMAX will continue processing the previously interrupted data transfer.

A data traffic example illustrating the case when events arrive from three event signals (Event0, 1 and 4) sorted to the same priority event group is given in Figure 1-19. Within the group, event0 has the highest priority, followed by event1 and event4. A transition on event signal zero is latched in the Event Flag register (EFR), and EventFlag0 is set. The flag is cleared once dMAX starts processing the event. The event0 triggers a long data transfer E0. dMAX begins moving the first quantum transfer E0(Q0) of the long data transfer E0, at the time stamp t0. The EventFlag0 is automatically cleared.

Figure 1-19. A Data Traffic Example: All Events Arrive from Three Event Signals Sorted to the Lower Priority Event Group



While dMAX is moving the second quantum transfer E0(Q1) of the long transfer E0, a new event arrives at time stamp t1 (EventFlag4 gets set). New events always have higher priority than long data transfers. Even though Event4 has lower priority than the event that started the long data transfer E0, the new event will be serviced as soon as quantum transfer E0(Q1) is complete.

At the time stamp t2, dMAX starts servicing the new event and EventFlag4, which corresponds to the event, is automatically cleared. The new event triggers a new long data transfer E4. The dMAX controller begins moving the first quantum transfer E4(Q0) of the new long data transfer E4, at the time stamp t2.

After quantum transfer E4(Q0) is completed (time stamp t3), there are no new events waiting to be processed, but there are two long data transfers pending execution. Long pending data transfers are prioritized, among themselves, according to their triggering events. Since Event0 has higher priority than Event4, the long data transfer E0 is resumed. The long data transfer E0 is done after quantum transfers E0(Q2), and E0(Q3) are completed. After completing transfer E0, dMAX will switch back, and resume long data transfer E4.

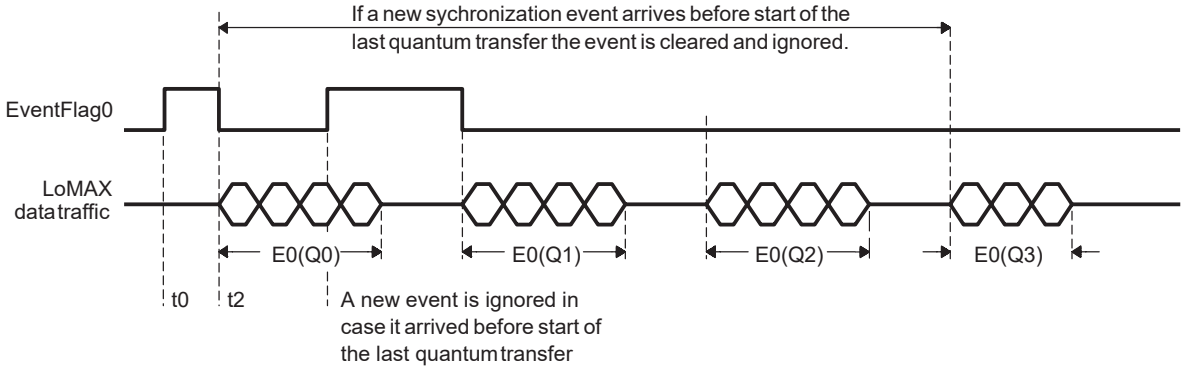
An event1 arrives at time stamp t4, while dMAX is transferring the second quantum transfer E4(Q1) of long data transfer E4. The new event is serviced as soon as the dMAX completes quantum transfer E4(Q1). The new event triggers a short data transfer which contains only one quantum transfer E1(Q0). dMAX resumes long data transfer E4, once quantum transfer E1(Q0) is complete.

At time stamp t6, dMAX would continue processing transfer E1, instead of resuming transfer E4, if number of elements synchronized to EventFlag1, that took place at time stamp t4, was larger than quantum transfer size limit.

If an event arrives while dMAX is performing a quantum transfer, the event will be serviced after completion of the quantum transfer that is in progress. After servicing the event, dMAX will continue processing the previously interrupted data transfer as long as there are no other pending transfers with higher priority than previously interrupted data transfer, and the new event did not trigger a long transfer with priority higher than the interrupted transfer.

An illustration of when a new event arrives, on the same channel, during a course of a long transfer is presented in Figure 1-20. An event at time stamp t0 triggers a long transfer (QTSL is set to four elements and transfer size is set to 15 elements). The transfer will be broken into four quantum transfers. The first three quantum transfers will be four words long. The last quantum transfer will be three words long. If a new event on the channel arrives before start of the last quantum transfer, dMAX will ignore the event. For example in Figure 1-20, a new event on the channel arrived during the first quantum transfer E0(Q0). The event will not affect progress of the transfer E0. dMAX clears the event flag right before start of the second quantum transfer E0(Q1). In other words, the event is ignored and dMAX continues with transfer E0 (second quantum transfer is performed). If a new event on the channel arrives during the last quantum transfer, the event will trigger a new transfer (assuming that the active element counter (COUNT0) is greater than zero when the last quantum is transferred).

Figure 1-20. A Data Traffic Example: A New Event Arrives During a Long Transfer



Transfer size of E0 is 15 elements
QTSL is set to 4 elements
SYNC is set to zero

Legend:

Ex(Qy): Quantum transfer (y), is a part of a long transfer initiated by an event on the event signal number (x).

EventFlagx: A flag that corresponds to event number (x), from the EFR register.

Register and Memory Description

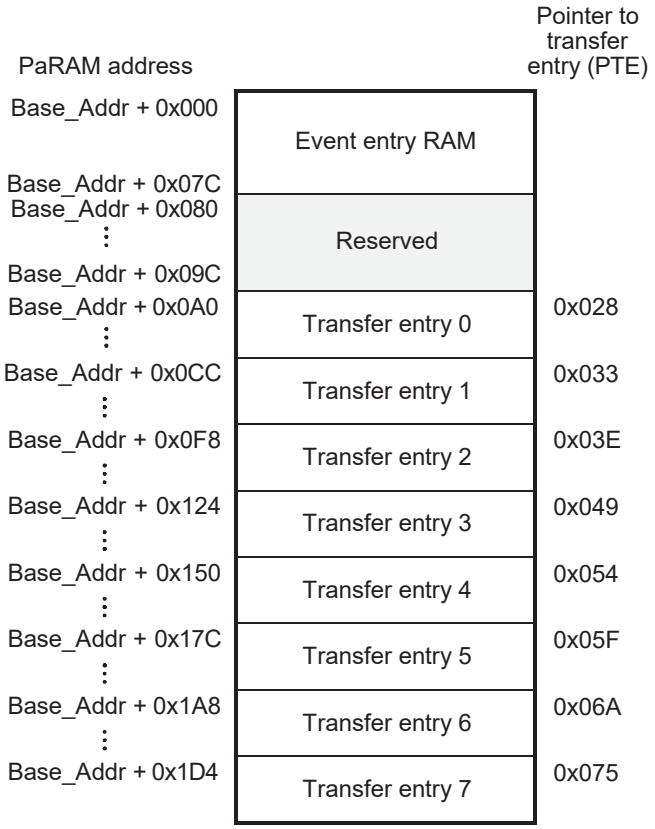
The memory for the dMAX controller is divided to high-priority and low-priority PaRAM. High-priority PaRAM contains the event entry table and transfer table for high-priority events, while low-priority PaRAM contains the event entry table and transfer table for low-priority events. Both high- and low-priority PaRAM memories are organized identically.

Topic **Page**

2.1 Parameter RAM (PaRAM)

Each PaRAM contains two sections: the event entry table section and the transfer entry table section. The PaRAM memory map is shown in Figure 2-1.

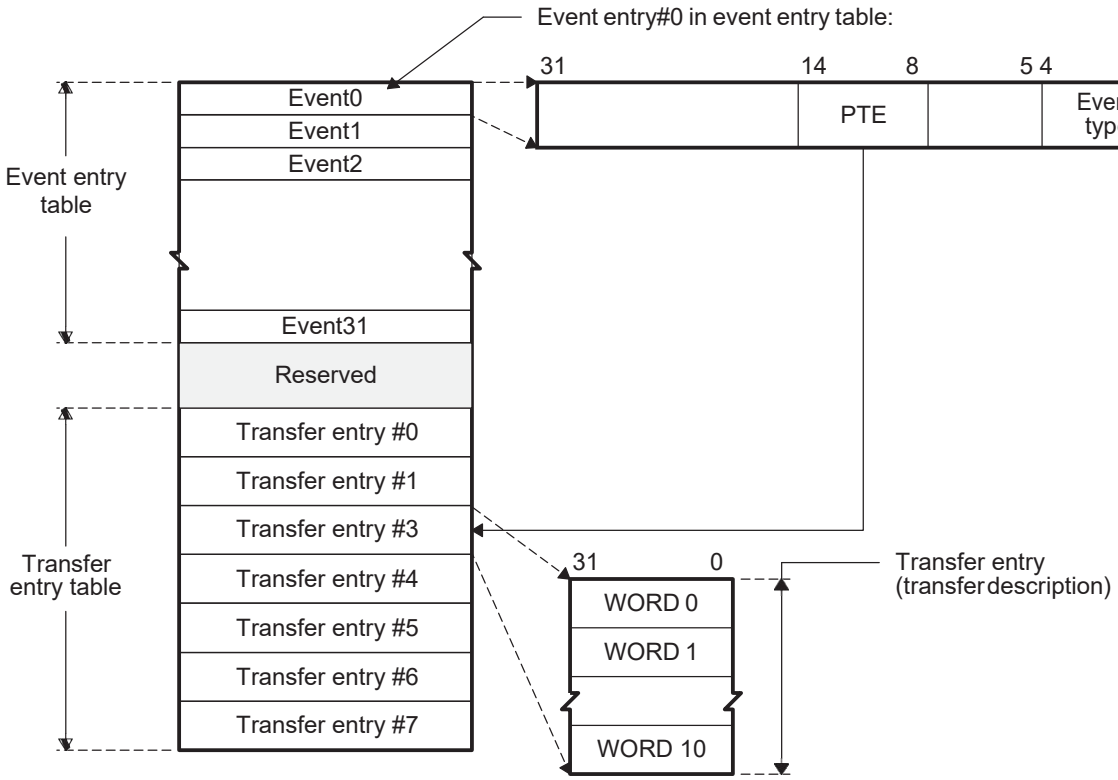
Figure 2-1. PaRAM Memory Map



The PaRAM memory size is equal to 128 words (32-bit words). Within the PaRAM, the event entry table is located between word offsets 0x00 and 0x1F; word offsets 0x20-0x27 are reserved and the transfer entry table ram utilizes the remainder. A pointer to transfer entry (PTE) specifies a word offset from the PaRAM base to the start of the transfer entry.

The dMAX PaRAM memory organization is shown in Figure 2-2.

Figure 2-2. PaRAM Memory Organization Block Diagram



An event entry describes an event type and associates the event to either one of transfer types or to an interrupt. If an event entry associates the event to one of the transfer types, the event entry will contain a pointer to the specific transfer entry in the transfer entry table (this is illustrated in Figure 2-2). If an event entry associates the event to an interrupt, the event entry specifies which interrupt should be generated to the CPU in case the event arrives. The transfer table may contain up to eight transfer entries. A transfer entry specifies details required by the dMAX controller to perform the transfer. The size of a transfer entry table is 88 words, and the transfer entry size is 11 words. A total of eight transfer entries can fit in both high-priority and low-priority PaRAMs.

2.1.1 Event Entry Table

When an event is processed, its corresponding event entry is passed to dMAX, and the controller uses the information to process the event. An event entry is programmable and associates the event either with a CPU interrupt or with one of the transfer types. In the first case, the event entry will specify which interrupt line should be used. In the second case, the event entry will dictate transfer options.

There is an event entry table for both high-priority and low-priority events. Each event can be associated with one of two event entries, based on event priority. One event entry is located inside the high-priority table and the other one is within the low-priority event table.

After receiving an event, dMAX uses the event entry currently associated with it. If the event currently belongs to the high-priority event group, dMAX is provided with the event entry from the high priority event entry table; the same happens with the low priority event group. If two events from different priority groups arrive at the same time, both event entries, from high- and low-priority event tables will be simultaneously provided to dMAX.

An event is sorted into a high- or low-priority group by setting DEHPR and DELPR registers. Each of the two event entry tables has 32 entries (each entry is one word long) and is stored starting from the word zero address in the PaRAM. There is a one-to-one correspondence between the events and the 31 entries in the table. Event entry 25 is not used (bit 25 in the DEFR is not associated to any event). The first entry in the event entry table is assigned to the event mapped to the LSB bit of the DEFR (to the event number zero). The events linked to subsequent bits of the DEFR are assigned to subsequent entries in the event entry table. The last entry (entry 31) in the event entry table is assigned to the event mapped to the MSB bit of the DEFR (event number 31).

Synchronization allows dMAX transfers to be triggered by events from peripherals. A channel puts a request for a data transfer only when its event entry specifies a transfer type, transfer options, and points to a transfer entry.

A new data transfer is initiated only when an event entry is programmed, and when dMAX receives its event or when the CPU manually synchronizes it (by creating a transition on one of the DTER register bits mapped into the DER register). The amount of data to be transferred depends on the configuration of the transfer entry.

Note: It is not recommended to access the active register set, indexes, reference counter (within the transfer entry), or event entry of a transfer in progress.

2.1.1.1 Event Entry for General Purpose Data Transfers

An event entry for a general purpose data transfer is presented in Figure 2-3 and explained in Table 2-1.

Figure 2-3. Event Entry for General Purpose Data Transfer

31	30	29	28	27	24	23	22	21	20	19	18	17	16
QTSL		SYNC	Reserved	TCC		ATCINT	TCINT	Reserved	RLOAD	CC		ESIZE	
R/W-0		R/W-0	R/W-0	R/W-0		R/W-0	R/W-0	R/W-0	R/W-0	R/W-0		R/W-0	

15	14	8	7	5	4	0
Reserved	PTE		Reserved		ETYPE	
R/W-0	R/W-0		R/W-0		R/W-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 2-1. Event Entry for General Purpose Data Transfer Field Descriptions

Bit	Field	Value	Description
31-30	QTSL[1:0]		Quantum Transfer Size Limit
		00	During a quantum transfer the dMAX controller may not move more than one element.
		01	During a quantum transfer the dMAX controller may not move more than four elements.
		10	During a quantum transfer the dMAX controller may not move more than eight elements.
		11	During a quantum transfer the dMAX controller may not move more than sixteen elements.
29	SYNC		Transfer Synchronization
		0	Transfer COUNT0 number of elements on every sync event
		1	Complete whole transfer on every sync event
28	Reserved	0	Reserved
27-24	TCC	15-0	Transfer Complete Code. Specifies code that is set in the DTCR0 or DTCR1 register after transfer (or transfer phase) is completed. TCC codes 0-7 correspond to bits 0-7 of the DTCR0. TCC codes 8-15 correspond to bits 0-7 of the DTCR1.
23	ATCINT		Enables alternate transfer mode
		0	Alternate transfer mode disabled
		1	The CPU gets notified after completion of each frame. A TCC pending bit is set in the DTCR0 or DTCR1 register and, if the bit in the DTCR register was previously cleared, the CPU interrupt is triggered at the end of each transfer phase.
22	TCINT		Transfer Completion Interrupt enable
		0	TCC code and Transfer Complete Interrupt disabled
		1	After completing a whole transfer, the dMAX controller sets a TCC pending bit. If the bit in the DTCR register was previously cleared, it triggers an interrupt to the CPU.
21	Reserved	0	Reserved
20	RLOAD		Reload Options
		0	No Reload after transfer is completed
		1	Reload active element counter, active SRC, and DST addresses when transfer is completed. If the PP control bit in the Transfer entry is equal to one, reference set zero (SRC0 and DST0) is loaded in the active address registers. If the PP control bit in the transfer entry is equal to zero, reference set one (SRC1 and DST1) is loaded in the active address registers.
19-18	CC[1:0]		Counter Configuration
		00	Counter bit field sizes within the transfer entry are defined as: COUNT2 =15-bits, COUNT1 = 8 bits, COUNT0 = 8 bits
		01	Counter bit field sizes within the transfer entry are defined as: COUNT2 =7-bits, COUNT1 = 8 bits, COUNT0 = 16 bits
		10	Counter bit field sizes within the transfer entry are defined as: COUNT2 =7-bits, COUNT1 = 16 bits, COUNT0 = 8 bits
		11	Reserved

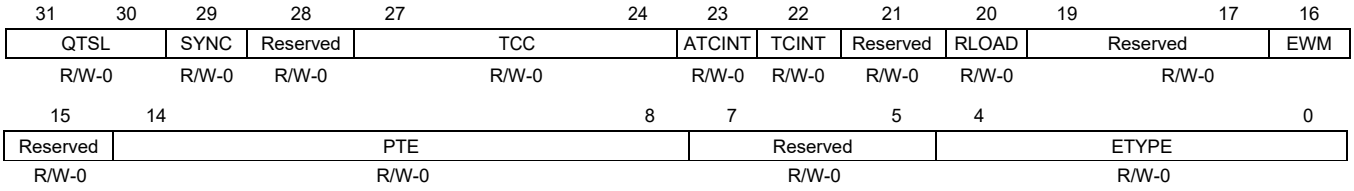
Table 2-1. Event Entry for General Purpose Data Transfer Field Descriptions (continued)

Bit	Field	Value	Description
17-16	ESIZE[1:0]		Element Size
		00	8-bit element
		01	16-bit element
		10	32-bit element
		11	Reserved
15	Reserved	0	Reserved
14-8	PTE		Pointer to Transfer Entry (PTE)
		0x75	These seven bits are used as a pointer to the location in the PaRAM where the transfer entry that corresponds to the event is stored.
		0x6A	
		0x5F	
		0x54	
		0x49	
		0x3E	
		0x33	
		0x28	
7-5	Reserved	0	Reserved
4-0	ETYPE	00011	Event type: General purpose data transfer

2.1.1.2 Event Entry for FIFO Transfers

The event entry for FIFO transfers is presented in Figure 2-4 and described in Table 2-2.

Figure 2-4. Event Entry for FIFO Transfer



LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 2-2. Event Entry for FIFO Transfer Field Descriptions

Bit	Field	Value	Description	
31-30	QTSL[1:0]		Quantum Transfer Limit	
		00	During a quantum transfer the dMAX controller may not move more than one element.	
		01	During a quantum transfer the dMAX controller may not move more than four elements.	
		10	During a quantum transfer the dMAX controller may not move more than eight elements.	
		11	During a quantum transfer the dMAX controller may not move more than 16 elements.	
29	SYNC		Transfer Synchronization	
		0	Transfer COUNT0 number of elements on every sync event	
		1	Transfer whole transfer on every sync event	
28	Reserved	0	Reserved	
27-24	TCC	15-0	Transfer Complete Code. Specifies code that is set in the DTCR0 or DTCR1 register after transfer (or transfer phase) is completed. TCC codes 0-7 correspond to bits 0-7 of the DTCR0. TCC codes 8-15 correspond to bits zero to seven of the DTCR1.	
23	ATCINT		Alternate Transfer Complete Interrupt enable	
		0	Alternate transfer mode disabled	
		1	The CPU gets notified after completion of each frame. A TCC pending bit is set in the DTCR0 or DTCR1 register and, if the bit in the DTCR register was previously cleared, the CPU interrupt is triggered at the end of each transfer phase.	
22	TCINT		Transfer Complete Interrupt enable	
		0	TCC code and Transfer Complete Interrupt disabled	
		1	After completing a whole transfer, the dMAX controller sets a TCC pending bit, and if the bit in the DTCR register was previously cleared, it triggers an interrupt to the CPU.	
21	Reserved	0	Reserved	
20	RLOAD		Reload Options	
		0	No Reload after transfer is completed	
		1	FIFO write case	FIFO read case
			Reload element active counter and active SRC address when transfer is completed. If the PP control bit in the Transfer entry is equal to one, reference address zero (SRC0) is loaded in the active address register. If the PP control bit in the Transfer entry is equal to zero, reference address one (SRC1) is loaded in the active address registers.	Reload element active counter and active DST address when transfer is completed. If the PP control bit in the Transfer entry is equal to one, reference address zero (DST0) is loaded in the active address register. If the PP control bit in the Transfer entry is equal to zero, reference address one (DST1) is loaded in the active address registers.
19-17	Reserved	0	Reserved	
16	EWM		Enable Watermark Notifications	
		0	Watermark notifications disabled (If FIFO write, FIFO full watermark; if FIFO read, FIFO empty watermark).	
		1	Watermark notifications enabled (If FIFO write, FIFO full watermark ; if FIFO read, FIFO empty watermark).	

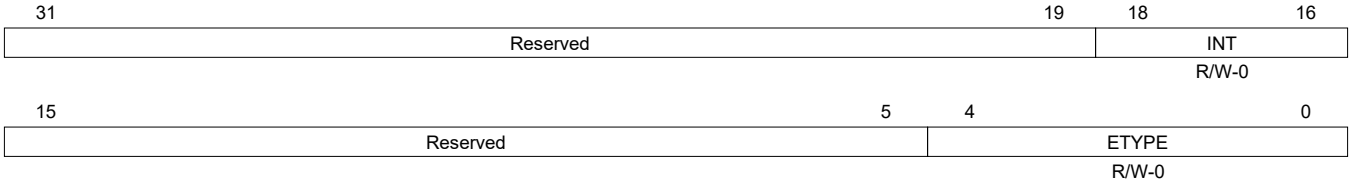
Table 2-2. Event Entry for FIFO Transfer Field Descriptions (continued)

Bit	Field	Value	Description
15	Reserved	0	Reserved
14-8	PTE		Pointer to Transfer Entry (PTE)
		0x75 0x6A 0x5F 0x54 0x49 0x3E 0x33 0x28	These seven bits are used as a pointer to the location in the PaRAM where the transfer entry that corresponds to the event is stored.
7-5	Reserved	0	Reserved
4-0	ETYPE		Event Type
		00100	Event Type: FIFO WRITE Transfer
		00101	Event Type: FIFO READ Transfer

2.1.1.3 Event Entry for Interrupt from dMAX Controller to the CPU

If an event is required to trigger an interrupt to the CPU, interrupt event entry is used. The interrupt event entry does not have an associated entry in the transfer entry table. The interrupt event entry is shown in Figure 2-5 and described in Table 2-3.

Figure 2-5. Event Entry for Interrupt from dMAX Controller to the CPU



LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 2-3. Event Entry for Interrupt from dMAX Controller to the CPU Field Descriptions

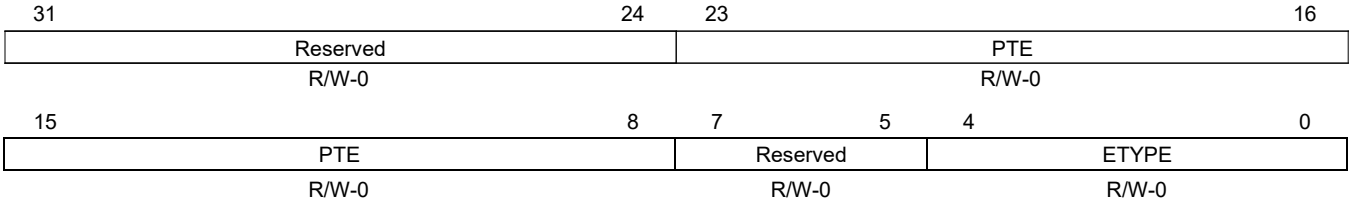
Bit	Field	Value	Description
31-19	Reserved	0	Reserved bits should be initialized to zero when configuring the Interrupt Event entry.
18-16	INT	010	Generate interrupt on INT9
		011	Generate interrupt on INT10
		100	Generate interrupt on INT11
		101	Generate interrupt on INT12
		110	Generate interrupt on INT13
		111	Generate interrupt on INT15
15-5	Reserved	0	Reserved bits should be initialized to zero when configuring the Interrupt Event entry.
4-0	ETYPE	00111	Generate an Interrupt to the CPU

Depending on the priority group of an event, the CPU interrupt can be triggered either by the HiMAX or LoMAX module.

2.1.1.4 Event Entry for One-Dimensional Burst Transfers

The event entry for one-dimensional burst transfers is presented in Figure 2-6 and described in Table 2-4.

Figure 2-6. Event Entry for One-Dimensional Burst Transfer



LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

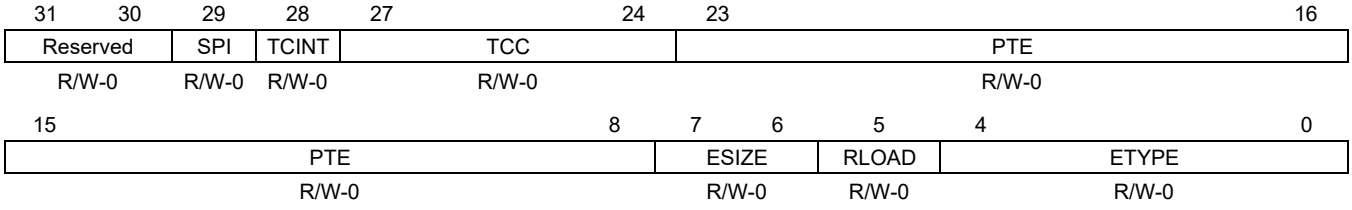
Table 2-4. Table Describing Bit Fields of Event Entry for One-Dimensional Burst Transfer

Bit	Field	Value	Description
31-24	Reserved	0	Reserved
23-8	PTE	0x1D4 0x1A8 0x17C 0x150 0x124 0xF8 0xCC 0xA0	These sixteen bits are used as a pointer to location in the PaRAM where the Transfer Entry that corresponds to the event is stored.
7-5	Reserved	0	Reserved
4-0	ETYPE	00110	Event type: One-Dimensional Burst Transfer

2.1.1.5 Event Entry for SPI Slave Transfers

The event entry for SPI slave transfers is shown in Figure 2-7 and described in Table 2-5.

Figure 2-7. Event Entry for SPI Slave Transfers



LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 2-5. Table Describing Bit Fields of Event Entry for SPI Slave Transfer

Bit	Field	Value	Description
31-30	Reserved	0	Reserved
29	SPI		SPI peripheral to use for the transfer
		0	SPI 0
		1	SPI 1
28	TCINT		Transfer Completion Interrupt Enable
		0	TCC code and Transfer Complete Interrupt disabled
		1	After completing a whole transfer, the dMAX controller sets a TCC pending bit and triggers an interrupt to the CPU.
27-24	TCC	15-0	Transfer Complete Code. Specifies code that is set in the DTCR0 or DTCR1 register after transfer is completed. TCC codes 0-7 correspond to bits 0-7 of the DTCR0. TCC codes 8-15 correspond to bits 0-7 of the DTCR1.
23-8	PTE		Pointer to Transfer Entry (PTE)
		0x1D4	
		0x1A8	
		0x17C	
		0x150	
		0x124	
		0xF8	
		0xCC	These sixteen bits are used as a pointer to location in the PaRAM where the Transfer Entry that corresponds to the event is stored.
		0xA0	
7-6	ESIZE		Element Size
		00	Reserved
		01	8-bit Element
		10	16-bit Element
		11	Reserved
5	RLOAD		Reload Options
		0	No Reload after the transfer is completed
		1	Reload active element counter, active SRC, and DST addresses when transfer is completed. If the PP control bit in the Transfer entry is equal to one, reference set zero (SRC0 and DST0) is loaded in the active address registers. If the PP control bit in the transfer entry is equal to zero, reference set one (SRC1 and DST1) is loaded in the active address registers.
4-0	ETYPE	00010	Event type: SPI slave data transfer

2.1.2 Transfer Entry Table

There are two transfer entry tables, each of which can hold up to eight transfer entries. One table is associated with high-priority events and the other is associated with low-priority events. The size of the transfer entry is fixed to 11 words. The format of the transfer entry varies, depending on the transfer type. A dedicated transfer entry is available for general purpose data transfers, and FIFO read/write.

The different transfer entry formats are described in the following sections.

Note: It is not recommended to access an active register set, indexes, reference counter (within the transfer entry), or event entry of a transfer in progress.

2.1.2.1 Transfer Entry for General Purpose Data Transfers

The transfer entry for general purpose data transfers is presented in Figure 2-8, Figure 2-9, and Figure 2-10 and described in Table 2-6.

Figure 2-8. Transfer Entry for General Purpose Data Transfer for CC=01 or CC=11

Word	31	30 29 28 27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
0	SRC ADDRESS (ACTIVE)				
1	DST ADDRESS (ACTIVE)				
2	PP	COUNT2 (ACTIVE)	COUNT1 (ACTIVE)	COUNT0 (ACTIVE)	
3	DST INDX 0			SRC INDX 0	
4	DST INDX 1			SRC INDX 1	
5	DST INDX 2			SRC INDX 2	
6	Reserved	COUNT2 (REFERENCE)	COUNT1 (REFERENCE)	COUNT0 (REFERENCE)	
7	SRC RELOAD ADDRESS0				
8	DST RELOAD ADDRESS0				
9	SRC RELOAD ADDRESS1				
10	DST RELOAD ADDRESS1				

Figure 2-9. Transfer Entry for General Purpose Data Transfer for CC=10

Word	31	30 29 28 27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
0	SRC ADDRESS (ACTIVE)				
1	DST ADDRESS (ACTIVE)				
2	PP	COUNT2 (ACTIVE)	COUNT1 (ACTIVE)		COUNT0 (ACTIVE)
3	DST INDX 0			SRC INDX 0	
4	DST INDX 1			SRC INDX 1	
5	DST INDX 2			SRC INDX 2	
6	Reserved	COUNT2 (REFERENCE)	COUNT1 (REFERENCE)		COUNT0 (REFERENCE)
7	SRC RELOAD ADDRESS0				
8	DST RELOAD ADDRESS0				
9	SRC RELOAD ADDRESS1				
10	DST RELOAD ADDRESS1				

Figure 2-10. Transfer Entry for General Purpose Data Transfer for CC=00

Word	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	SRC ADDRESS (ACTIVE)																															
1	DST ADDRESS (ACTIVE)																															
2	PP	COUNT2 (ACTIVE)															COUNT1 (ACTIVE)								COUNT0 (ACTIVE)							
3	DST INDX 0																SRC INDX 0															
4	DST INDX 1																SRC INDX 1															
5	DST INDX 2																SRC INDX 2															
6	Reserved	COUNT2 (REFERENCE)															COUNT1 (REFERENCE)								COUNT0 (REFERENCE)							
7	SRC RELOAD ADDRESS0																															
8	DST RELOAD ADDRESS0																															
9	SRC RELOAD ADDRESS1																															
10	DST RELOAD ADDRESS1																															

Table 2-6. Transfer Entry for General Purpose Data Field Descriptions

Word	Bit	Field	Description
0	31-0	SRC ADDRESS (ACTIVE)	Source address - updated by the dMAX controller during course of transfer
1	31-0	DST ADDRESS (ACTIVE)	Destination address - updated by the dMAX controller during course of transfer
2	31	PP	Reference bit. Keeps track of what was loaded in the active set of parameters during reload. It gets updated by the dMAX controller only during reload of active parameters (if reload in the event entry is enabled by setting RLOAD to one).
	Variable	COUNT2 (ACTIVE)	Counter2 value - Counter for the third dimension of transfer, updated by the dMAX controller during transfer. The size of this bit field depends on setting of the CC bit-field in the event entry.
	Variable	COUNT1 (ACTIVE)	Counter1 value - Counter for the second dimension of transfer, updated by the dMAX controller during transfer. The size of this bit field depends on setting of the CC bit-field in the event entry.
	Variable	COUNT0 (ACTIVE)	Counter0 value - Counter for the first dimension of transfer, updated by the dMAX controller during course of transfer. The size of this bit field depends on setting of the CC bit-field in the event entry.
3	31-16	DSTINDX0	Value used to update destination address after value in the first dimension element counter is decremented. The index represents the lower 16-bits of 32-bit offset that will be added to active address (upper 16 bits will be sign extended). Index is expressed in number of elements.
	15-0	SRCINDX0	Value used to update source address after value in the first dimension element counter is decremented. The index represents the lower 16-bits of 32-bit offset that will be added to active address (upper 16 bits will be sign extended). Index is expressed in number of elements.
4	31-16	DSTINDX1	Value used to update destination address after value in the second dimension element counter is decremented. The index represents the lower 16-bits of 32-bit offset that will be added to active address (upper 16 bits will be sign extended). Index is expressed in number of elements.
	15-0	SRCINDX1	Value used to update source address after value in the second dimension element counter is decremented. The index represents the lower 16-bits of 32-bit offset that will be added to active address (upper 16 bits will be sign extended). Index is expressed in number of elements.
5	31-16	DSTINDX2	Value used to update destination address after value in the third dimension element counter is decremented. The index represents the lower 16-bits of 32-bit offset that will be added to active address (upper 16 bits will be sign extended). Index is expressed in number of elements.
	15-0	SRCINDX2	Value used to update source address after value in the third dimension element counter is decremented. The index represents the lower 16-bits of 32-bit offset that will be added to active address (upper 16 bits will be sign extended). Index is expressed in number of elements.

Table 2-6. Transfer Entry for General Purpose Data Field Descriptions (continued)

Word	Bit	Field	Description
6	31	Reserved	Reserved
	Variable	COUNT2 (REFERENCE)	Counter2 Reference- This value is used to reload counter for the third dimension of transfer (in case reload is enabled). Size of this bit field depends on setting of the CC bit-field in the event entry.
	Variable	COUNT1 (REFERENCE)	Counter1 Reference - This value used to reload counter for the second dimension of transfer. Size of this bit field depends on setting of the CC bit-field in the event entry.
	Variable	COUNT0 (REFERENCE)	Counter0 Reference - This value used to reload active counter for the first dimension of transfer. Size of this bit field depends on setting of the CC bit-field in the event entry.
7	31-0	SRC RELOAD ADDRESS0	Source Address Reload 0 - Used by the dMAX controller to reload the active source address parameter at the end of transfer (if specified by RLOAD bit filed of the event entry).
8	31-0	DST RELOAD ADDRESS0	Destination Address Reload 0 - Used by the dMAX controller to reload the active destination address parameter at the end of transfer (if specified by RLOAD bit filed of the event entry).
9	31-0	SRC RELOAD ADDRESS1	Source Address Reload 1 - Used by the dMAX controller to reload the active source address parameter at the end of transfer (if specified by RLOAD bit filed of the event entry).
10	31-0	DST RELOAD ADDRESS1	Destination Address Reload 1 - Used by the dMAX controller to reload the active destination address parameter at the end of transfer (if specified by RLOAD bit filed of the event entry).

Active entries are modified during the transfer; at the end of the transfer they could be reloaded. If reload is not enabled, the transfer will complete as specified in the active register set and stop. When reload is enabled, the dMAX controller will still use the active register set. When the first transfer is completed, if the PP bit value is set to one, the dMAX controller will load the active address registers with a value from the reload0 set of registers and flip the PP bit to zero. After the transfer is complete, the dMAX controller will load the active address set with the value from reload1 set of registers, and flip the PP bit value to one. The PP bit always indicates the last set of reload registers that was loaded in the active set of address registers.

2.1.2.2 Transfer Entry for FIFO Write

The FIFO transfer entry is presented in Figure 2-11 and described in Table 2-7.

Figure 2-11. Transfer Entry for FIFO Write

Word	31	30 29 28 27 26 25 24 23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0	SRC ADDRESS (ACTIVE)		
1	PFD (Pointer to FIFO Descriptor)		
2	PP	COUNT1 (ACTIVE)	COUNT0 (ACTIVE)
3	Reserved		SRC INDX 0
4	Reserved		SRC INDX 1
5	Reserved	COUNT1 (REFERENCE)	COUNT0 (REFERENCE)
6	SRC RELOAD ADDRESS0		
7	SRC RELOAD ADDRESS1		
8	Pointer to Delay Table 0 (used with Reload0)		
9	Pointer to Delay Table 1 (used with Reload1)		
10	Reserved		

Table 2-7. Transfer Entry for FIFO Write Field Descriptions

Word	Bit	Field	Description
0	31-0	SRC ADDRESS (ACTIVE)	Source address - updated by the dMAX controller during course of transfer.
1	31-0	PFD	Specifies pointer to description of destination FIFO. The source FIFO descriptor can be placed anywhere in the DSP memory space.
2	31	PP	Reference bit. Keeps track of what was loaded in the active set of parameters during reload. It gets updated by the dMAX controller only during parameter reload (if reload is enabled in the event entry by setting RLOAD to one).
	30-16	COUNT1 (ACTIVE)	Counter1 value - Counter for the second dimension of transfer, updated by the dMAX controller during transfer. Transfer Entry will be ignored by the dMAX controller when: (active COUNT1) > (reference COUNT1).
	15-0	COUNT0 (ACTIVE)	Counter0 value - Counter for the first dimension of transfer, updated by the dMAX controller during course of transfer.
3	31-16	Reserved	Reserved
	15-0	SRC INDX 0	Value used to update source address after value in the first dimension element counter is decremented. The index represents the lower 16 bits of 32-bit offset that will be added to active address (upper 16 bits will be sign extended). Index is expressed in number of elements.
4	31-16	Reserved	Reserved
	15-0	SRC INDX 1	Value used to update source address after value in the second dimension element counter is decremented. The index represents the lower 16 bits of 32-bit offset that will be added to active address (upper 16 bits will be sign extended). Index is expressed in number of elements.
5	31	Reserved	Reserved
	30-16	COUNT1 (REFERENCE)	Counter1 Reference - This value used to reload active counter for the second dimension of transfer.
	15-0	COUNT0 (REFERENCE)	Counter0 Reference - This value used to reload active counter for the first dimension of transfer.
6	31-0	SRC RELOAD ADDRESS0	Source Address Reload 0 - Used by the dMAX controller to reload the active source address parameter at the end of transfer (if specified by RLOAD bit filed of the event entry).
7	31-0	SRC RELOAD ADDRESS1	Source Address Reload 1 - Used by the dMAX controller to reload the active source address parameter at the end of transfer (if specified by RLOAD bit filed of the event entry).
8	31-0	POINTER TO DELAY TABLE 0	This parameter is used as a pointer to table of delays used for non-sequential multi-tap delay. Table pointed by this entry is used in combination with reload0 register.
9	31-0	POINTER TO DELAY TABLE 1	This parameter is used as a pointer to table of delays used for non-sequential multi-tap delay. Table pointed by this entry is used in combination with reload1 register.
10	31-0	Reserved	Reserved

Active entries are modified during the course of a transfer; at the end of the transfer they could be reloaded.

If reload is not enabled, the transfer will complete as specified in the active register set, and stop. If reload is enabled, on the first execution, dMAX will use a set of active registers, and after transfer completion, dMAX will flip the PP bit and reload the active registers. When the first transfer is completed, if the PP bit value is set to one, dMAX will load the active address register with the value from the reload0 register and flip the PP bit to zero. After the transfer is complete, dMAX will load the active address parameter with the value from the reload1 register and flip the PP bit value to one. The PP bit always indicates which reload register was loaded in the active address register, and which delay table should be used.

If the PP control bit is zero, reload is enabled, and if the frame counter is greater than zero, dMAX will use a delay table pointed to by the pointer to Delay Table 0 zero from the FIFO transfer entry. If the PP control bit is one, reload is enabled, and if the frame counter is greater than zero, dMAX will use a delay table pointed to by the pointer to Delay Table 1 from the FIFO transfer entry. The maximum number of entries in a delay table is 32767 (since number of entries in a delay table equals to value of the reference value of the COUNT1).

2.1.2.3 Transfer Entry for FIFO Read

The transfer entry for moving data from circular buffer is presented in Figure 2-12 and described in Table 2-8.

Figure 2-12. Transfer Entry for FIFO Read

Word	31	30 29 28 27 26 25 24 23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8 7 6 5 4 3 2 1 0
0	DST ADDRESS (ACTIVE)		
1	PFD (Pointer to FIFO Descriptor)		
2	PP	COUNT1 (ACTIVE)	COUNT0 (ACTIVE)
3	Reserved		DST INDX 0
4	Reserved		DST INDX 1
5	Reserved	COUNT1 (REFERENCE)	COUNT0 (REFERENCE)
6	DST Reload Address0		
7	DST Reload Address1		
8	Pointer to Delay Table 0 (used with Reload0)		
9	Pointer to Delay Table 1 (used with Reload1)		
10	Reserved		

Table 2-8. Transfer Entry for FIFO READ Field Descriptions

Word	Bit	Field	Description
0	31-0	DST Address (ACTIVE)	Destination address - updated by the dMAX controller during the transfer.
1	31-0	PFD	Specifies pointer to description of destination FIFO; can be placed anywhere in the DSP memory space.
2	31	PP	Reference bit. Keeps track of what was loaded in the active set of parameters during reload. Gets updated by dMAX only during parameter reload (in case reload is enabled in the event entry by setting RLOAD to one).
	30-16	COUNT1 (ACTIVE)	Counter1 value - Counter for the second dimension of transfer, updated by dMAX during course transfer. Transfer Entry will be ignored by dMAX when: (active COUNT1) > (reference COUNT1).
	15-0	COUNT0 (ACTIVE)	Counter0 value - Counter for the first dimension of transfer, updated by dMAX during the transfer.
3	31-16	Reserved	Reserved
	15-0	DST INDX 0	Value used to update destination address after value in the first dimension element counter is decremented. The index represents the lower 16-bits of 32-bit offset that will be added to active address (upper 16 bits will be sign extended). Index is expressed in number of elements.
4	31-16	Reserved	Reserved
	15-0	DST INDX 1	Value used to update destination address after value in the second dimension element counter is decremented. The index represents the lower 16-bits of 32-bit offset that will be added to active address (upper 16 bits will be sign extended). Index is expressed in number of elements.
5	31	Reserved	Reserved
	30-16	COUNT1 (REFERENCE)	Counter1 Reference - This value is used to reload active counter for the second dimension of transfer.
	15-0	COUNT0 (REFERENCE)	Counter0 Reference - This value is used to reload active counter for the first dimension of transfer.
6	31-0	DST Reload Address0	Destination Address Reload 0 - Used by the dMAX controller to reload the active destination address parameter at the end of transfer (if specified by RLOAD bit filed of the event entry).
7	31-0	DST Reload Address1	Destination Address Reload 1 - Used by the dMAX controller to reload the active destination address parameter at the end of transfer (if specified by RLOAD bit filed of the event entry).
8	31-0	Pointer to Delay Table 0	This parameter is used as a pointer to table of delays used for non-sequential multi-tap delay. Table pointed by this entry is used in combination with reload0 register.
9	31-0	Pointer to Delay Table 1	This parameter is used as a pointer to table of delays used for non-sequential multi-tap delay. Table pointed by this entry is used in combination with reload1 register.
10	31-0	Reserved	Reserved

If reload is not enabled, the transfer will complete as specified in the active register set and stop. In case reload is enabled, on the first execution, dMAX will use a set of active registers, and after transfer completion, dMAX will flip the PP bit, and reload active registers. When the first transfer is completed, if the PP bit value is set to one, dMAX will load the active address register with the value from the reload0 register and flip the PP bit to zero. After the transfer is complete, dMAX will load the active address parameter with the value from the reload1 register and flip the PP bit value to one. The PP bit always indicates which reload register was loaded in the active address register, and which delay table should be used.

If the PP control bit is zero, reload is enabled, and if the frame counter is greater than 0, dMAX will use Delay Table 0 from the FIFO transfer entry. If the PP control bit is 1, reload is enabled, and if the frame counter is greater than 0, dMAX will use Delay Table 1 from the FIFO transfer entry. The maximum number of entries in a delay table is 32767 (the number of entries in a delay table equals to the value of the reference value of COUNT1).

2.1.2.4 Transfer Entry for One-Dimensional Burst Transfers

The transfer entry for One-Dimensional burst data transfers is presented in Figure 2-13 and described in Table 2-9.

Figure 2-13. Transfer Entry for One-Dimensional Burst Transfer

Word	31–24	23–16	15–8	7–6	5–4	3–1	0
0	EVNT	TCC	Reserved	NBURST		Reserved	TCINT
1	SRC						
2	DST						
3	Reserved	BURSTLEN	CNT				
4-10	Reserved						

Table 2-9. Transfer Entry for One-Dimensional Burst Transfer Description

Word	Bit	Field	Value	Description
0	31-24	EVNT		Event number (0-31)- Specifies the event number of the transfer.
	23-16	TCC		Transfer Complete Code. Specifies code that is set in the DTCR0 or DTCR1 register after transfer is completed. TCC codes 0-7 correspond to bits 0-7 of the DTCR0. TCC codes 8-15 correspond to bits 0-7 of the DTCR1.
	15-8	Reserved		Reserved
	7-4	NBURST		Number of bursts (0-15) – 1 to 16 bursts each of length BURSTLEN - updated by the dMAX controller during course of transfer
	3-1	Reserved		Reserved
	0	TCINT		Transfer Completion Interrupt Enable
			0	TCC code and Transfer Complete Interrupt disabled
			1	After completing a whole transfer, the dMAX controller sets a TCC pending bit and triggers an interrupt to the CPU.
1	31-0	SRC		Source address - updated by the dMAX controller during course of transfer.
2	31-0	DST		Destination address - updated by the dMAX controller during course of transfer
3	31-24	Reserved		Reserved
	23-15	BURSTLEN		Burst length in units of bytes (1-64)
	15-0	CNT		Count in unit of bytes - updated by the dMAX controller during course of transfer
4-10	31-0	Reserved		Reserved

2.1.2.5 Transfer Entry for SPI Slave Transfers

The transfer entry for SPI Slave transfers is presented in Figure 2-14 and described in Table 2-10.

Figure 2-14. Transfer Entry for SPI Slave Transfer

Word	31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	SRC ADDRESS (ACTIVE)																															
1	DST ADDRESS (ACTIVE)																															
2	PP	Reserved															COUNT (ACTIVE)															
3	Reserved																COUNT (REFERENCE)															
4	SRC RELOAD ADDRESS0																															
5	DST RELOAD ADDRESS0																															
6	SRC RELOAD ADDRESS1																															
7	DST RELOAD ADDRESS1																															
8-10	Reserved																															

Table 2-10. Transfer Entry for SPI Slave Transfer Description

Word	Bit	Field	Description
0	31-0	SRC ADDRESS (ACTIVE)	Source address - updated by the dMAX controller during course of transfer
1	31-0	DST ADDRESS (ACTIVE)	Destination address - updated by the dMAX controller during course of transfer
2	31	PP	Reference bit. Keeps track of what was loaded in the active set of parameters during reload. It gets updated by the dMAX controller only during reload of active parameters (if reload in the event entry is enabled by setting RLOAD to one).
	30-16	Reserved	Reserved
	15-0	COUNT (ACTIVE)	Count in unit of elements - updated by the dMAX controller during course of transfer
3	31-16	Reserved	Reserved
	15-0	COUNT (REFERENCE)	Reference Count in unit of elements. This value is used to reload the active count at the end of transfer (if specified by RLOAD bit filed of the event entry)
4	31-0	SRC RELOAD ADDRESS0	Source Address Reload 0 - Used by the dMAX controller to reload the active source address parameter at the end of transfer (if specified by RLOAD bit filed of the event entry)
5	31-0	DST RELOAD ADDRESS0	Destination Address Reload 0 - Used by the dMAX controller to reload the active destination address parameter at the end of transfer (if specified by RLOAD bit filed of the event entry)
6	31-0	SRC RELOAD ADDRESS1	Source Address Reload 1 - Used by the dMAX controller to reload the active source address parameter at the end of transfer (if specified by RLOAD bit filed of the event entry)
7	31-0	DST RELOAD ADDRESS1	Destination Address Reload 1 - Used by the dMAX controller to reload the active destination address parameter at the end of transfer (if specified by RLOAD bit filed of the event entry)
8-10	31-0	Reserved	Reserved

Active entries are modified during the transfer; at the end of the transfer they could be reloaded. If reload is not enabled, the transfer will complete as specified in the active register set and stop. When reload is enabled, the dMAX controller will still use the active register set. When the first transfer is completed, if the PP bit value is set to one, the dMAX controller will load the active address registers with a value from the reload0 set of registers and flip the PP bit to zero. After the transfer is complete, the dMAX controller will load the active address set with the value from reload1 set of registers, and flip the PP bit value to one. The PP bit always indicates the last set of reload registers that was loaded in the active set of address registers.

2.2 FIFO Descriptor

A FIFO (circular buffer) descriptor specifies:

- FIFO base address
- FIFO element size
- FIFO size (in number of elements)
- A read and write pointer
- Two status codes used to indicate FIFO full and FIFO empty conditions
- Two level marks (empty mark EMARK and full mark FMARK)
- Error Field (EFIELD) used to describe an error condition to the CPU

A FIFO descriptor is referenced by the pointer from a transfer entry (FIFO read transfer and FIFO write transfer can both use the same FIFO by referencing the same FIFO descriptor).

The FIFO descriptor is shown in Figure 2-15 and described in Table 2-11.

Figure 2-15. FIFO Descriptor

Word	31 30 29 28 27 26 25 24	23 22 21 20 19 18 17 16	15 14 13 12 11 10 9 8	7 6 5 4 3 2 1 0
0	FIFO BASE ADDRESS			
1	Reserved		WRITE POINTER	
2	Reserved	ES ZE Reserved	FIFO SIZE	
3	Reserved		READ POINTER	
4	Reserved FMSC	Reserved	FMARK	
5	Reserved EMSC	Reserved	EMARK	
6	Reserved FF	Reserved EF2	Reserved EF1	Reserved EF0

Table 2-11. FIFO Descriptor Field Descriptions

Word	Bit	Field	Value	Description
0	31-0	FIFO BASE ADDRESS		FIFO Base Address. Must be word-aligned.
1	31-20	Reserved	0	Reserved
	19-0	Write Pointer		Write Pointer
2	31-26	Reserved	0	Reserved
	25-24	ESIZE		FIFO Element Size
			00	8-bit element
			01	16-bit element
			10	32-bit element
			11	Reserved
	23-20	Reserved	0	Reserved
	19-0	FIFO SIZE		FIFO size in number of elements
3	31-20	Reserved	0	Reserved
	19-0	Read Pointer		FIFO Read Pointer
4	31-28	Reserved	0	Reserved
	27-24	FMSC		Full mark status code. This bit is set in the DFSR0 or DFSR1 register when number of samples in the FIFO is equal or larger than number specified by FMARK bit field. FMSC codes zero to seven correspond to bits zero to seven of the DFSR0. FMSC codes eight to 15 correspond to bits zero to seven of the DFSR1.
	23-20	Reserved	0	Reserved
	19-0	FMARK		If the number of new samples becomes equal or grows above full mark (FMARK), the condition is signaled by FMSC code of the buffer.

Table 2-11. FIFO Descriptor Field Descriptions (continued)

Word	Bit	Field	Value	Description
5	31-28	Reserved	0	Reserved
	27-24	EMSC		Empty mark status code. This bit is set in the DFSR0 or DFSR1 register when the number of samples in the FIFO is equal or smaller than number specified by EMARK bit field. EMSC codes zero to seven correspond to bits zero to seven of the DFSR0. EMSC codes eight to 15 correspond to bits zero to seven of the DFSR1.
	23-20	Reserved	0	Reserved
	19-0	EMARK		If the number of new samples becomes equal or falls below empty mark (EMARK), the condition is signaled by EMSC code of the buffer.
6	31-25	Reserved	0	Reserved
	24	FF	0-1	FIFO Full Flag. If the FIFO becomes full, this bit is automatically set by dMAX.
	23-17	Reserved	0	Reserved
	16	EF2	0-1h	Error Flag 2 bit gets set to indicate an overflow error. The overflow error is indicated if a FIFO write transfer is attempted and COUNT0 active is larger than the number of empty slots in the FIFO. In this case, the transfer will be aborted, the EF2 bit will be automatically set, and INT7 will be generated to the CPU, and both FIFO mark flags will be set in the DFSR.
	15-9	Reserved	0	Reserved
	8	EF1	0-1h	Error Flag 1 (EF1) bit gets set in case of a table-based multi-tap delay read transfer, and if a delay specified in the delay table is larger than the number of samples stored in the FIFO. In this case, the transfer will be aborted, the EF1 bit will be automatically set, and INT7 will be generated to the CPU and both FIFO mark flags will be set in the DFSR.
	7-1	Reserved	0	Reserved
	0	EFO	0-1h	Error flag 0 bit gets set if there is an underflow error condition. A FIFO read transfer is attempted, and the COUNT0 active of the attempted transfer is larger than the number of samples available for read in the FIFO. In this case, the transfer will be aborted, the EFO bit will be automatically set, and INT7 will be generated to the CPU and both FIFO mark flags will be sent in the DFSR.

2.3 dMAX Control Registers

The list of dMAX control registers is shown in Table 2-12.

Table 2-12. dMAX Control Registers

Register Address	dMAX Register Name	See	Register Description
0x6000 0008	DEPR	Section 2.3.7	The dMAX event polarity register (DEPR) controls the polarity-rising edge (low to high) or falling edge (high to low)-that sets the flag in the EFR register.
0x6000 000C	DEER	Section 2.3.5	The events can be enabled by writing a 1 to dMAX Event Enable Register (DEER).
0x6000 0010	DEDR	Section 2.3.6	The events can be disabled by writing a 1 to dMAX Event Disable Register (DEDR).
0x6000 0014	DEHPR	Section 2.3.8	An event is assigned to the high priority event group when the bit, which corresponds to the event, is set in the dMAX Event High Priority Register (DEHPR).
0x6000 0018	DELPR	Section 2.3.9	An event is assigned to the low priority event group when the bit, which corresponds to the event, is set in the dMAX Event Low Priority Register (DELPR).
0x6000 001C	DEFR	Section 2.3.4	The dMAX Event Flag Register (DEFR) indicates that an appropriate transition edge (specified in the Event Polarity Register) has occurred on the event signals. All events are captured in the event flag register, even when the events are disabled.
0x6000 0034	DER0	Section 2.3.1	The dMAX event register (DER0) reflects current value of the event signals 7-0.
0x6000 0054	DER1	Section 2.3.2	The dMAX event register (DER1) reflects current value of the event signals 15-8.
0x6000 0074	DER2	Section 2.3.3	The dMAX event register (DER2) reflects current value of the event signals 23-16.
0x6000 0094	DER3	Table 2-16	The dMAX event register (DER3) reflects current value of the event signals 31-24.
0x6000 0040	DFSR0	Section 2.3.10	dMAX FIFO status register 0. Writing a 1 to the DFSR0 register clears the corresponding bit. Writing 0 has no effect.
0x6000 0060	DFSR1	Section 2.3.11	dMAX FIFO status register 1. Writing a 1 to the DFSR1 register clears the corresponding bit. Writing 0 has no effect.
0x6000 0080	DTCR0	Section 2.3.12	dMAX transfer completion register 0. Writing a 1 to the DTCR0 register clears the corresponding bit. Writing 0 has no effect.
0x6000 00A0	DTCR1	Section 2.3.13	dMAX transfer completion register 1. Writing a 1 to the DTCR1 register clears the corresponding bit. Writing 0 has no effect.
-	DETR	Section 2.3.14	dMAX event trigger register. By toggling a bit in this register the CPU can trigger an event. To facilitate faster CPU access, the dMAX Event Trigger Register is not memory-mapped and is placed inside the CPU module.
-	DESR	Section 2.3.15	dMAX event status register. To facilitate low CPU access overhead this register mirrors TCC bits from DTCR0 and DTCR1 registers. The register also keeps track of dMAX controller activity. To facilitate faster CPU access, the dMAX Event Status Register is not memory-mapped and is placed inside the CPU module.

A set of two registers is provided to set event priority (DEHPR and DELPR), and to enable/disable events (DEER and DEDR). This eliminates the need for read/modify/write when enabling/disabling events or when setting event priorities. For example, writing '1' to DEHPR puts the corresponding event in the high-priority group, while writing '0' has no effect.

2.3.1 dMAX Event Register 0 (DER0)

Values in the DER0 reflect the current state (high or low) on the event signals zero to seven. The DER0 is a read-only register, and is shown in Figure 2-6 and described in Table 2-13.

Figure 2-16. dMAX Event Register 0 (DER0)

31								16
Reserved								
R-x								

15	8	7	6	5	4	3	2	1	0
Reserved		EVT7	EVT6	EVT5	EVT4	EVT3	EVT2	EVT1	EVT0
		R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 2-13. dMAX Event Register 0 (DER0) Field Descriptions

Bit	Field	Description
31-8	Reserved	Reserved
7-0	EVT*n*	Event 0-7 bits. **This is a read-only register.**

2.3.2 dMAX Event Register 1 (DER1)

Values in the DER1 reflect the current state (high or low) on the event signals eight to 15. The DER1 is a read-only register.

The dMAX event register 1 (DER1) is shown in Figure 2-17 and described in Table 2-14.

Figure 2-17. dMAX Event Register 1 (DER1)

31								16
Reserved								
R-x								

15	8	7	6	5	4	3	2	1	0
Reserved		EVT15	EVT14	EVT13	EVT12	EVT11	EVT10	EVT9	EVT8
		R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x

LEGEND: R = Read only; -*n* = value after reset

Table 2-14. dMAX Event Register 1 (DER1) Field Descriptions

Bit	Field	Description
31-8	Reserved	Reserved
7-0	EVT*n*	Event 15-8 bits. **This is a read-only register.**

2.3.3 dMAX Event Register 2 (DER2)

Values in the DER2 reflect the current state (high or low) on the event signals 16 to 23.

The DER2 is shown in Figure 2-18 and described in Table 2-15.

Figure 2-18. dMAX Event Register 2 (DER2)

31									16
Reserved									
15			**8**	**7**	**6**	**5**	**4**	**3**	**2**	**1**	**0**
Reserved				EVT23	EVT22	EVT21	EVT20	EVT19	EVT18	EVT17	EVT16
				R-x	R-x	R-x	R-x	R-x	R-x	R-x	R-x

LEGEND: R = Read only; -*n* = value after reset

Table 2-15. dMAX Event Register 2 (DER2) Field Descriptions

Bit	Field	Description
31-8	Reserved	Reserved
7-0	EVT*n*	Event 23-16 bits. **This is a read-only register.**

Table 2-16. dMAX Event Register 3 (DER3) Field Descriptions

Bit	Field	Description
31-8	Reserved	Reserved
7-0	EVT*n*	Event 31-24 bits. **This is a read only register.**

2.3.4 dMAX Event Flag Register (DEFR)

A transition on the event signal (event) is captured in the dMAX Event Flag Register (DEFR), even when the events are disabled.

Once an event has been posted in the DEFR, and the event is enabled in the DEER, the event flag is automatically cleared by dMAX, immediately after it starts processing the request. CPU can also clear the event flag by writing 1 to the DEFR register.

The DEFR register is shown in Figure 2-19 and described in Table 2-17.

Figure 2-19. dMAX Event Flag Register (DEFR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EF31	EF30	EF29	EF28	EF27	EF26	Reserved	EF24	EF23	EF22	EF21	EF20	EF19	EF18	EF17	EF16
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EF15	EF14	EF13	EF12	EF11	EF10	EF9	EF8	EF7	EF6	EF5	EF4	EF3	EF2	EF1	EF0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 2-17. dMAX Event Flag Register (DEFR) Field Descriptions

Bit	Field	Value	Description
31-26	EF*n*	0	Transition has not occurred since last clear
		1	Transition has occurred since last clear
		0	No changes in flag register
		1	Clears the corresponding bit to 0
25	Reserved	0	Reserved
24-0	EF*n*	0	Transition has not occurred since last clear
		1	Transition has occurred since last clear
		0	No changes in flag register
		1	Clears the correpsonding bit to 0

2.3.5 dMAX Event Enable Register (DEER)

To enable an event, the corresponding bit must be set in dMAX Event Enable Register (DEER). Any of the event bits in the DEER can be set to 1 to enable that corresponding event (writing 0 has no effect).

The event registers latch all events that are captured by dMAX, even if that event is disabled. Event processing procedure is analogous to an interrupt enable and interrupt pending processing, thus ensuring that dMAX does not drop any events. Re-enabling an event with a pending event signaled in the event flag register forces dMAX to process that event.

The DEER is shown in Figure 2-20and described in Table 2-18.

Figure 2-20. dMAX Event Enable Register (DEER)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EE31	EE30	EE29	EE28	EE27	EE26	Reserved	EE24	EE23	EE22	EE21	EE20	EE19	EE18	EE17	EE16
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0		R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EE15	EE14	EE13	EE12	EE11	EE10	EE9	EE8	EE7	EE6	EE5	EE4	EE3	EE2	EE1	EE0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 2-18. dMAX Event Enable Register (DEER) FIELD Descriptions

Bit	Field	Value	Description
31-26	EE*n*[31-26]		Event 26-31 enable bits.
		0	Writing a 0 has no effect.
		1	Any of the event bits can be set to 1 to enable that event.
25	Reserved	0	Reserved
24-0	EE*n*[24-0]		Event 0-24 enable bits.
		0	Writing a 0 has no effect.
		1	Any of the event bits can be set to 1 to enable that event.

2.3.6 dMAX Event Disable Register (DEDR)

In order to disable an event, the corresponding bit must be set in the dMAX Event Disable Register (DEDR). Any of the event bits in the DEDR can be set to 1 to disable that corresponding event (writing 0 has no effect).

The DEDR is shown in Figure 2-21 and described in Table 2-19.

Figure 2-21. dMAX Event Disable Register (DEDR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ED31	ED30	ED29	ED28	ED27	ED26	Reserved	ED24	ED23	ED22	ED21	ED20	ED19	ED18	ED17	ED16
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0		R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ED15	ED14	ED13	ED12	ED11	ED10	ED9	ED8	ED7	ED6	ED5	ED4	ED3	ED2	ED1	ED0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 2-19. dMAX Event Disable Register (DEDR) Field Descriptions

Bit	Field	Value	Description
31-0	ED*n*		Event 0-31 disable bits.
		0	Writing a 0 has no effect.
		1	Any of the event bits can be set to 1 to disable that event.

2.3.7 dMAX Event Polarity (DEPR)

The dMAX Event Polarity Register (DEPR) controls the polarity-rising edge (low to high) or falling edge (high to low) that sets the event flag in the DEFR register. To ensure recognition of the signal as an edge, the signal must maintain the new level for at least one dMAX clock cycle.

Each bit in the DEPR corresponds to one dMAX event. Writing a 1 to one of the event bits makes the event sensitive to a rising edge. Writing a 0 to one of the event bits makes the event sensitive to a falling edge.

The DEPR is shown in Figure 2-22 and described in Table 2-20.

Figure 2-22. dMAX Event Polarity (DEPR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EP31	EP30	EP29	EP28	EP27	EP26	Reserved	EP24	EP23	EP22	EP21	EP20	EP19	EP18	EP17	EP16
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EP15	EP14	EP13	EP12	EP11	EP10	EP9	EP8	EP7	EP6	EP5	EP4	EP3	EP2	EP1	EP0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 2-20. dMAX Event Polarity Register (DEPR) Field Descriptions

Bit	Field	Value	Description
31-0	EP*n*		Event 0-31 polarity select bits.
		0	Event flag is set on falling edge.
		1	Event flag is set on rising edge.

2.3.8 dMAX Event High Priority (DEHPR)

In the dMAX Event High-Priority Register (DEHPR), each event can be individually configured as a high-priority event by writing a 1 into the corresponding bit of the DEHPR. Writing 0 has no effect. The DEHPR is shown in Figure 2-23 and described in Table 2-21. The way to handle events that are placed in the high priority group is determined by the event with the lowest bit value. The event assigned to the lowest bit has the highest priority (within the high priority group, the event assigned to bit 0 position has the highest priority; the event assigned to bit 31 has the lowest priority).

Since only one dMAX register can be accessed at one time, an event will be sorted in the high or low priority group depending on the last write to the corresponding bit of either the DEHPR or DELPR.

Figure 2-23. dMAX Event High Priority (DEHPR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
EHP31	EHP30	EHP29	EHP28	EHP27	EHP26	Reserved	EHP24	EHP23	EHP22	EHP21	EHP20	EHP19	EHP18	EHP17	EHP16
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0		R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
EHP15	EHP14	EHP13	EHP12	EHP11	EHP10	EHP9	EHP8	EHP7	EHP6	EHP5	EHP4	EHP3	EHP2	EHP1	EHP0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 2-21. dMAX Event High Priority Register (DEHPR) Field Descriptions

Bit	Field	Value	Description
31-0	EHP*n*		Event 0-31 high-priority select bits.
		0	No effect
		1	Event set as a high-priority event.

2.3.9 dMAX Event Low Priority (DELPR)

In the dMAX event low-priority register (DELPR), each event can be individually configured as a low-priority event by writing a 1 into its corresponding bit. Writing zero has no effect. The DELPR is shown in Figure 2-24 and described in Table 2-22. Bit 0 position has the highest priority; bit 31 has the lowest priority. The way to handle of events that are placed in the low priority group is determined by the event with the lowest bit value. The event assigned to the lowest bit has the highest priority (bit 0 position has the highest priority; and the event assigned to bit 31 has the lowest priority within the low priority group).

Since only one dMAX controller register can be accessed at one time, an event will be sorted into the high or low priority group depending on the last write to the corresponding bit of either the DEHPR or DELPR.

Figure 2-24. dMAX Event Low Priority (DELPR)

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16
ELP31	ELP30	ELP29	ELP28	ELP27	ELP26	Reserved	ELP24	ELP23	ELP22	ELP21	ELP20	ELP19	ELP18	ELP17	ELP16
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
ELP15	ELP14	ELP13	ELP12	ELP11	ELP10	ELP9	ELP8	ELP7	ELP6	ELP5	ELP4	ELP3	ELP2	ELP1	ELP0
R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 2-22. dMAX Event Low Priority Register (DELPR) Field Descriptions

Bit	Field	Value	Description
31-0	ELP*n*		Event 0-31 high- priority select bits.
		0	No effect.
		1	Event set as a low-priority event.

2.3.10 dMAX FIFO Status Register 0 (DFSR0)

The dMAX FIFO Status Register (DFSR) is used by dMAX to report FIFO status codes (FSC) to the CPU. From the DFSR, the CPU reads the code set by dMAX. The CPU clears the DFSR pending bits by writing 1, writing 0 has no effect. The DFSR0 is shown in Figure 2-25 and described in Table 2-23.

Figure 2-25. dMAX FIFO Status Register 0 (DFSR0)

31	16
Reserved	

15	8	7	6	5	4	3	2	1	0
Reserved		FSC7	FSC6	FSC5	FSC4	FSC3	FSC2	FSC1	FSC0
		R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 2-23. dMAX FIFO Status Register 0 (DFSR0) Field Descriptions

Bit	Field	Value	Description
31-8	Reserved	0	Reserved
7-0	FSC*n*		FIFO status flags 7-0. The CPU reads status bit set by the dMAX controller. The CPU clears the flags by writing to this register.
		0	No effect
		1	Clears corresponding flag.

The FSC value programmed in the FIFO descriptor dictates the DFSR bit number that gets set. The FMSC and EMSC bit fields specify the FIFO Full and FIFO Empty conditions, respectively, with values between 0-15. The FIFO status codes 0-7 correspond to bits 0-7 of the DFSR0.

If the FSC pending bit is not cleared by the CPU before dMAX attempts to set the bit again, dMAX will not generate a FIFO status interrupt.

2.3.11 dMAX FIFO Status Register 1 (DFSR1)

The dMAX FIFO Status Register (DFSR1) is used by dMAX to report FIFO status codes to the CPU. From the DFSR, the CPU reads the code set by the dMAX controller. The CPU clears the DFSR bits by writing 1; writing 0 has no effect. The DFSR1 is shown in Figure 2-26 and described in Table 2-24.

Figure 2-26. dMAX FIFO Status Register 1 (DFSR1)

31	16
Reserved	

15	8	7	6	5	4	3	2	1	0
Reserved		FSC15	FSC14	FSC13	FSC12	FSC11	FSC10	FSC9	FSC8
		R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 2-24. dMAX FIFO Status Register 1 (DFSR1) Field Descriptions

Bit	Field	Value	Description
31-8	Reserved	0	Reserved
7-0	FSC*n*		FIFO status flags 16-8. The CPU reads status bit set by the dMAX controller. The CPU clears the flags by writing to this register.
		0	No effect
		1	Clears corresponding flag

The FSC value programmed in the FIFO descriptor dictates the DFSR bit number that gets set. The FMSC and EMSC bit fields specify the FIFO Full and FIFO empty conditions respectively, with values between 0-15. The FIFO Status codes 8-15 correspond to bits 0-7 of the DFSR1.

If the FSC pending bit is not cleared by the CPU before dMAX attempts to set the bit again, dMAX will not generate a FIFO status interrupt.

2.3.12 dMAX Transfer Completion Register 0 (DTCR0)

In the dMAX Transfer Completion Register (DTCR), the CPU clears the code set by dMAX. The CPU clears the DTCR bits by writing 1; writing 0 has no effect. The DTCR is shown in Figure 2-27 and described in Table 2-25.

Figure 2-27. dMAX Transfer Completion Register 0 (DTCR0)

31	16
Reserved	

15	8	7	6	5	4	3	2	1	0
Reserved		TCC7	TCC6	TCC5	TCC4	TCC3	TCC2	TCC1	TCC0
		R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 2-25. dMAX Transfer Completion Register 0 (DTCR0) Field Descriptions

Bit	Field	Value	Description
31-8	Reserved	0	Reserved
7-0	TCC*n*		Pending request flags 7-0. The CPU reads pending requests set by the dMAX controller from the DESR. The CPU clears the flags by writing to this register.
		0	No effect
		1	Clears corresponding flag

The TCC value programmed in the event entry dictates the DTCR bit number that gets set. The TCC field specifies the transfer complete pending bit, with values between 0-15. The TCC codes 0-7 correspond to bits 0-7 of DTCR0. If the TCC pending bit is not cleared by the CPU before dMAX attempts to set the bit again, dMAX will not generate a CPU interrupt. To determine which TCC values are set by the dMAX controller, the CPU may either read DTCR, or read the DESR (reading from the DESR is more efficient).

2.3.13 dMAX Transfer Completion Register 1 (DTCR1)

The DTCR Register (DTCR1) is used by the CPU to clear transfer completion codes set by dMAX. The CPU clears the DTCR bits by writing 1, writing 0 has no effect. The DTCR1 is shown in Figure 2-28 and described in Table 2-26.

Figure 2-28. dMAX Transfer Completion Register 1 (DTCR1)

31	16
Reserved	

15	8	7	6	5	4	3	2	1	0
Reserved		TCC15	TCC14	TCC13	TCC12	TCC11	TCC10	TCC9	TCC8
		R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 2-26. dMAX Transfer Completion Register 1 (DTCR1) Field Descriptions

Bit	Field	Value	Description
31-8	Reserved	0	Reserved
7-0	TCC*n*		Pending request flags 15-8. The CPU reads pending requests set by the dMAX controller from the DESR. The CPU clears the flags by writing to this register.
		0	No effect
		1	Clears the corresponding flag

The TCC value programmed in the event entry dictates the DTCR bit number that gets set. The TCC field specifies the transfer complete pending bit, with values between 0-15. The TCC codes eight to 15 correspond to bits zero to seven of the DTCR1. If the TCC pending bit is not cleared by the CPU before dMAX attempts to set the bit again, dMAX will not generate a CPU interrupt. To determine which TCC values are set by dMAX, the CPU may either read the DTCR, or read the DESR (reading from the DESR is more efficient).

2.3.14 dMAX Event Trigger Register (DETR)

The dMAX Event Trigger Register (DETR) is used by the CPU to trigger events. To facilitate faster CPU access, the DETR is not memory-mapped and is placed inside CPU module (there is no CPU overhead when writing to the DETR). The DETR is presented in Figure 2-29 and described in Table 2-27.

Figure 2-29. dMAX Event Trigger Register (DETR)

31	22	21	20	19	18	17	16
Reserved		ET21	ET20	ET19	ET18	ET17	ET16
		R/W-0	R/W-0	R/W-0	R/W-0	R/W-0	R/W-0

15	4	3	2	1	0
Reserved		ET3	ET2	ET1	ET0
		R/W-0	R/W-0	R/W-0	R/W-0

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 2-27. dMAX Event Trigger (DET) Register Field Descriptions

Bit	Field	Description
31-22	Reserved	Reserved
21	ET21	This bit is mapped to EVENT31 in the dMAX Event Register. Creating an appropriate edge on this bit in the DETR will trigger an EVENT31.
20	ET20	This bit is mapped to EVENT30 in the dMAX Event Register. Creating an appropriate edge on this bit in the DETR will trigger an EVENT30.
19	ET19	This bit is mapped to EVENT24 in the dMAX Event Register. Creating an appropriate edge on this bit in the DETR will trigger an EVENT24.
18	ET18	This bit is mapped to EVENT18 in the dMAX Event Register. Creating an appropriate edge on this bit in the DETR will trigger an EVENT18.
17	ET17	This bit is mapped to EVENT11 in the dMAX Event Register. Creating an appropriate edge on this bit in the DETR will trigger an EVENT11.
16	ET16	This bit is mapped to EVENT1 in the dMAX Event Register. Creating an appropriate edge on this bit in the DETR will trigger an EVENT1.
15-4	Reserved	Reserved
3	ET3	This bit is mapped to EVENT23 in the dMAX Event Register. Creating an appropriate edge on this bit in the DETR will trigger an EVENT23.
2	ET2	This bit is mapped to EVENT17 in the dMAX Event Register. Creating an appropriate edge on this bit in the DETR will trigger an EVENT17.
1	ET1	This bit is mapped to EVENT10 in the dMAX Event Register. Creating an appropriate edge on this bit in the DETR will trigger an EVENT10.
0	ET0	This bit is mapped to EVENT0 in the dMAX Event Register. Creating an appropriate edge on this bit in the DETR will trigger an EVENT0.

To trigger an event, the CPU must create an appropriate edge on a DETR bit which is mapped to the DER. The appropriate edge depends on the polarity set for the event (programmed in the DEPR).

Toggling bits in the DETR will trigger the dMAX events only if the CPU is not halted by emulation.

If the chip support library (CSL) for TMS320C672x is not used and it is necessary to access the DET register from the C program, the register must be declared as:

extern far cregister volatile unsigned int DETR;

An example of DETR usage to trigger an event zero is shown in Example 2-1.

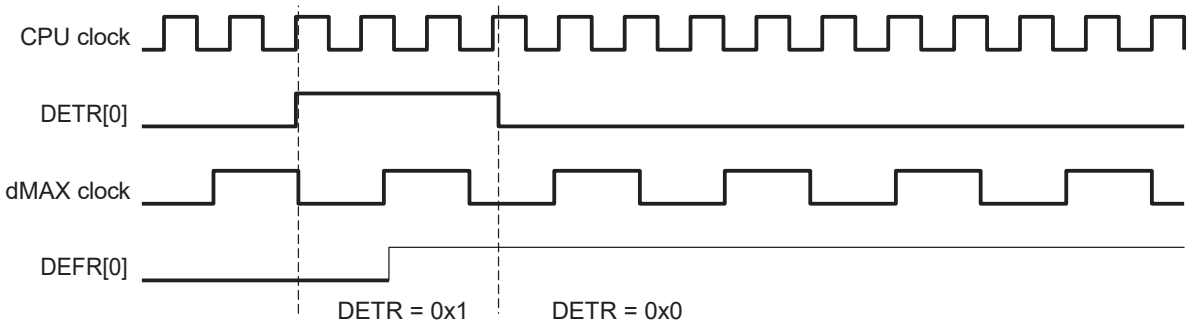
Example 2-1. Triggering Event0 by Writing to the DET Register

```
...
    DETR = 0x00000001;
    asm(" nop");
    asm(" nop");
    asm(" nop");
    DETR = 0x00000000;

...
```

A simplified timing diagram of triggering an event by writing to the DETR is shown in Figure 2-30. In this case, dMAX Event Polarity for event zero is set to '1' (even flag is set on rising edge of DET[0]). The CPU toggles the DET[0] by consecutively writing '1' and then '0' to bit0 of the DETR. The event flag zero in the dMAX Event Flag register (DEFR) gets set when the rising edge on DETR[0] is detected by dMAX (dMAX samples the DETR on every dMAX clock).

Figure 2-30. CPU Triggers Event by Writing to the DETR (when DEPR[0]=1) Timing Diagram



2.3.15 dMAX Event Status Register (DESR)

To facilitate faster CPU access, the dMAX Event Status register is not memory-mapped and is placed inside the CPU module. Unlike when reading the DTC0 and DTC1 registers, there is no CPU overhead when reading the DESR. The DESR is read-only and has the following application:

- To mirror DTCR0 and DTCR1 (dMAX Transfer Completion Register) bits. By reading the DESR[15:8] and DESR[31:24], the CPU has a fast way of accessing bits from the DTCR0 and DTCR1 registers.

Organization of the DES register is presented in Figure 2-31 and explained in Table 2-28.

Figure 2-31. dMAX Event Status (DES) Register

31	30	29	28	27	26	25	24	23	18	17	16
TCC15	TCC14	TCC13	TCC12	TCC11	TCC10	TCC9	TCC8	Reserved			
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0			

15	14	13	12	11	10	9	8	7	0
TCC7	TCC6	TCC5	TCC4	TCC3	TCC2	TCC1	TCC0	Reserved	
R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	R-0	

LEGEND: R/W = Read/Write; R = Read only; -*n* = value after reset

Table 2-28. dMAX Event Status Register (DESR) Field Descriptions

Bit	Field	Description
31-24	TCC[15:8]	These bits are read-only and correspond to the TCC bits from DTCR1 register.
23-16	Reserved	Reserved
15-8	TCC[7:0]	These bits are read-only and correspond to the TCC bits from DTCR0 register.
7-0	Reserved	Reserved

The DESR is read-only and the CPU can clear a transfer completion bit only by writing to the DTC0 or DTC1 registers. To access the DESR from the C program, if the Chip Support Library (CSL) for TMS320C672x is not used, the register must be declared as:

extern far cregister volatile unsigned int DESR;

An example of DESR register usage is shown in Example 2-2.

Example 2-2. Reading the DES Register

```
...
unsigned int IPR_val;
      IPR_val = DESR;

...
```

Transfer Examples

This chapter provides examples for various dMAX transfer types.

CAUTION

1DN and SPI slave transfers require System Patch v2.00 or later. You can download the patch at http://focus.ti.com/docs/toolsw/folders/print/sprc203.html.

Topic **Page**

3.1 Transfer Synchronization

The dMAX transfer can be frame-synchronized, or whole transfer can be performed after receiving a synchronization event. If frame synchronization is enabled (in the event entry the SYNC bit field is cleared) one event is required for the transfer of each frame. If frame synchronization is disabled (in the event entry the SYNC bit field is set) only one event is required to synchronize the whole transfer.

When frame synchronization is enabled (SYNC='0'), and the ATCINT bit field within the event entry is set, dMAX will notify the CPU after transfer of each frame is completed. When frame synchronization is enabled (SYNC='0'), and the ATCINT bit field within the event entry is cleared, dMAX will require a synchronization event to transfer each frame, but it will not notify the CPU after a frame transfer is complete. When frame synchronization is disabled (SYNC='1'), dMAX will notify the CPU after whole transfer is completed only if the TCINT (or ATCINT) bit field within the event entry is set.

The dMAX controller notifies the CPU about transfer status by using interrupt (INT8), and by setting a bit in the DTCR register. A bit that gets set is specified by the TCC bit field of the event entry.

3.2 General Purpose Transfer Examples

General-purpose data transfer covers 1-dimensional (1D), 2-dimensional (2D) and 3-dimensional (3D) transfer. The 1D and 2D transfers are implemented as special cases of 3D transfers, where counters for the higher dimensions are equal to zero. A 3D transfer with the third dimension counter equal to zero becomes a 2D transfer. For a general-purpose data transfer, an independent set of indexes for source and destination can be specified for each transfer dimension. This facilitates 1D-2D, 1D-3D, and 2D-3D element sorting.

3.2.1 Steps Required to Set Up a General Purpose Transfer

The following steps are required to set up a general-purpose dMAX transfer:

1. Priority of an event that will be used to trigger the general-purpose data transfer must be defined. To put the event into the high-priority group, a bit corresponding to the event in the DEHPR should be set to one. To put the event into the low-priority group, a bit corresponding to the event in the DELPR should be set to one.
2. The event signal edge (rising/falling) that will be used to trigger an event must be defined. To trigger an event on the rising edge of the event signal, a bit corresponding to the event in the DEPR should be set to one. To trigger an event on the falling edge of the event signal, a bit corresponding to the event in the DEPR should be cleared to zero.
3. If the event is sorted to the high-priority group, its event entry in the HiMAX PaRAM must be defined. If the event is sorted to the low-priority group, its event entry in the LoMAX PaRAM must be defined. The following bit fields in the event entry must be configured:
   - ETYPE bit field must be set to '00011' for general-purpose data transfer.
   - PTE bit field is used as a pointer to a location in the PaRAM where a transfer entry that corresponds to the event is stored.
   - ESIZE should define correct element size.
   - CC should define configuration of the counter field (correct widths of COUNT0, COUNT1 and COUNT2 bit fields) within the transfer entry.
   - If the RLOAD bit is set when a transfer is complete, an active counter and an active address register will be reloaded from one of two sets of address reference registers. If the RLOAD bit is cleared, dMAX will ignore new events after a transfer is complete.
   - When a transfer is complete, the TCINT bit should be set if a notification to the CPU is required. The ATCINT bit should be set if notification to the CPU is required after transfer of each frame.
   - The TCC indicates which bit in the DTCR is going to be set to indicate transfer status to the CPU.
   - If SYNC=1, dMAX will require one event to complete the transfer. If SYNC=0, dMAX will move only one frame of data after receiving each synchronization event.
   - The QTSL bit field defines granularity with which dMAX breaks up a large transfer into a number of smaller sub-transfers.
4.

Transfer entry must be properly configured (transfer must be defined by source/destination address, source/destination indexes and counter bit fields). The PTE bit field of the event entry points to the transfer entry.

When the event entry reload is enabled (RLOAD='1'), the PP bit in the transfer entry must be properly configured. When PP='0', after the transfer is completed, reload register set one is loaded in the set of active registers. When PP='1', after the transfer is completed, reload register set zero is loaded in the set of active registers.

5. The event must be enabled by setting a corresponding bit in the DEER.

Once an event is enabled, dMAX will perform a data transfer after an appropriate transition is detected on the event signal.

If ATCINT = '1' or TCINT='1', dMAX will signal transfer status to the CPU by using an interrupt (INT8), and by setting a bit, specified by the TCC bit field of the event entry, in the DTCR. The DTCR bits are mirrored in the DESR. The read-only DESR is located inside the CPU module and the CPU can access the register with minimum overhead (the fastest way of monitoring the DTCR is to read its copy from the DESR).

To keep receiving the notifications from a particular dMAX channel, after each notification, the CPU must clear the transfer completion bit used by the dMAX channel. The CPU should use the read-only DESR to find out which transfer completion bits were set, and it should clear them by writing '1' to the corresponding bits from the DTCR.

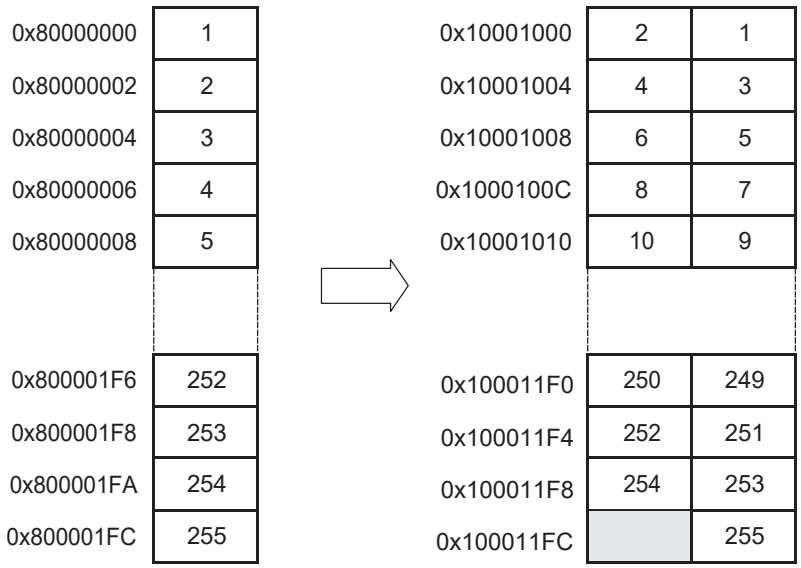
After enabling the event, it is not recommended to access the event entry, active register set, indexes, and the reference counter bit fields within the transfer entry.

3.2.2 EXAMPLE: 1D Block Move Transfer

Often during device operation it is necessary to transfer a block of data from one location to another, usually between on- and off-chip memories. The most basic transfer that can be performed by dMAX is a block move transfer.

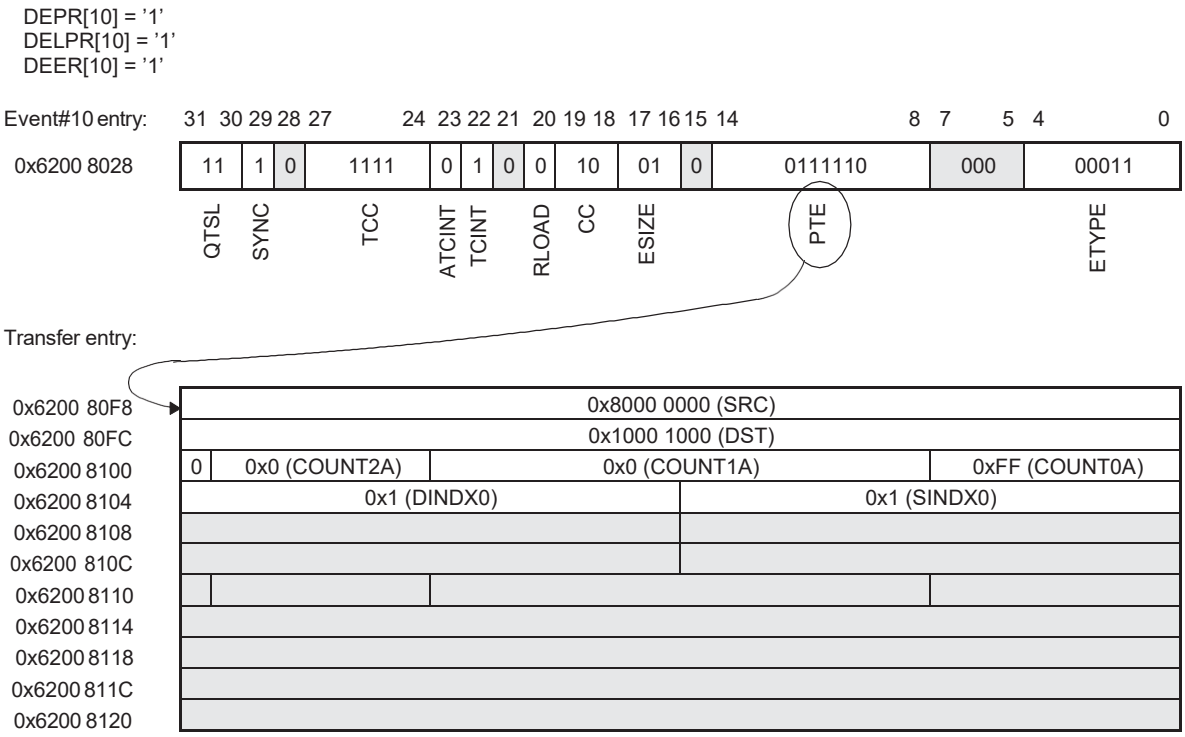
In this example, a section of data is to be copied from external memory to internal memory. The data block is 255 half-words and resides at address 0x80000000. It is to be transferred to internal address 0x10001000. The data transfer is shown in Figure 3-1.

Figure 3-1. Block Move Diagram



The parameters for this transfer are shown in Figure 3-2. Event 10 is used to trigger this transfer. Here, Event 10 is triggered by CPU write '0', followed by CPU write '1', to bit one of the DETR register (rising edge of the event signal 10 triggers an event since DEPR[10] = '1'). The event is processed by the LoMAX since DELPR[10] = '1'.

Figure 3-2. Event Entry and Transfer Entry for 1D Block Transfer



The whole transfer is completed after receiving one synchronization event (SYNC = '1' in the event entry); subsequent synchronization events will be ignored (reload is disabled RLOAD = '0' in the event entry). After completing the transfer, dMAX will notify the CPU by triggering an interrupt (INT8) and by setting a bit 7 in the DTCR1 (TCC bit field within the event entry is equal to 15).

The dMAX controller will split the transfer into 16 quantum transfers (QTSL='11' in the event entry). The first 15 quantum transfers will be 16 elements long, while the last quantum transfer size will move 15 elements.

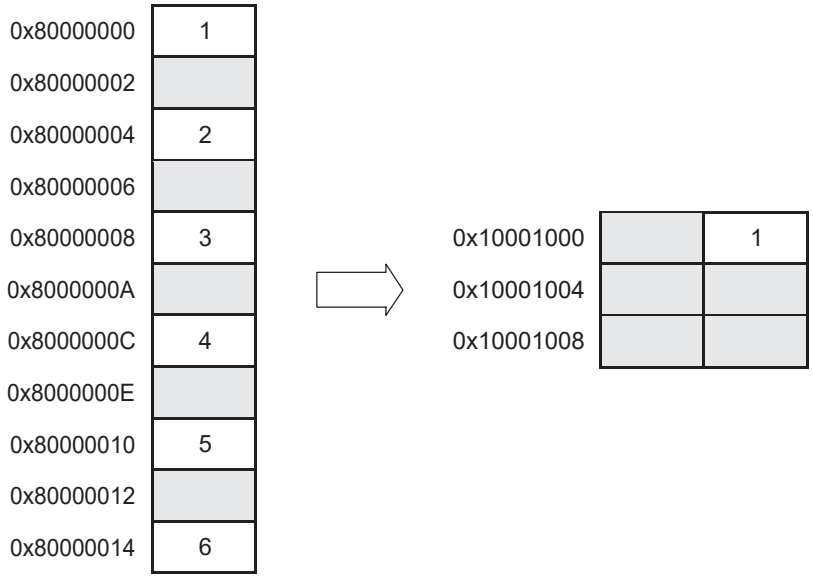
3.2.3 EXAMPLE: Element- Synchronized 1D Transfer

When the synchronization bit field within the event entry is set to zero (SYNC = '0'), dMAX requires a synchronization event to transfer each frame (frame size is equal to COUNT0 elements). Therefore, to implement an element-synchronized transfer (only one element is transferred after receiving a synchronization event) two-dimensional transfers must be used.

In this example, six data elements are to be copied from the external memory to internal memory. The six half-words of source data starts at address 0x80000000, with source index equal to two (source elements to be transferred are spaced by one element). The data are to be transferred to contiguous destination memory block starting at the internal address 0x10001000.

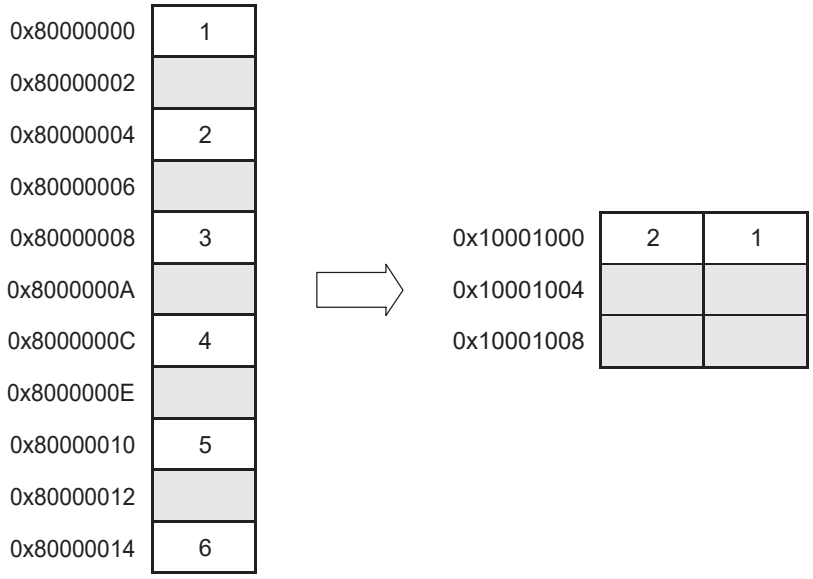
A memory snapshot of the data transfer after receiving the first synchronization event is shown in Figure 3-3.

Figure 3-3. Element-Synchronized 1D Transfer Diagram (After Receiving the First Synchronization Event)



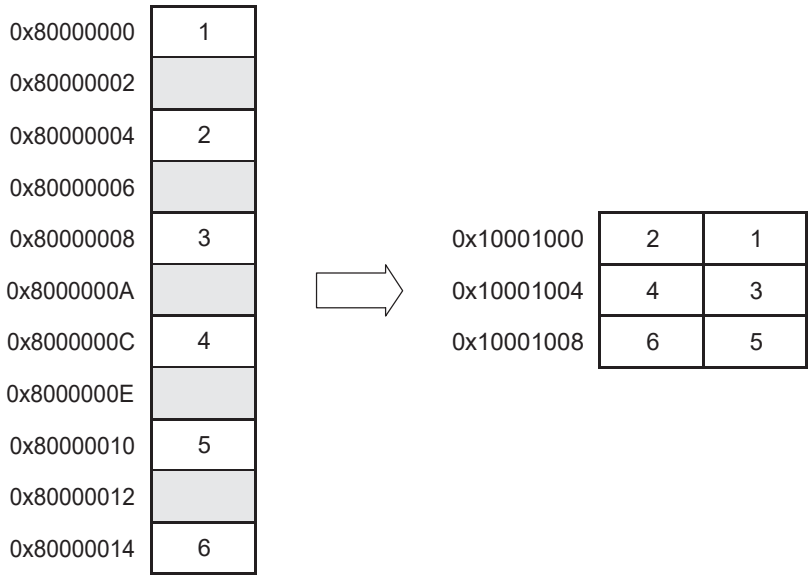
A memory snapshot for the data transfer after receiving the first two synchronization events is shown in Figure 3-4.

Figure 3-4. Element-Synchronized 1D Transfer Diagram (After Receiving the Second Synchronization Event)



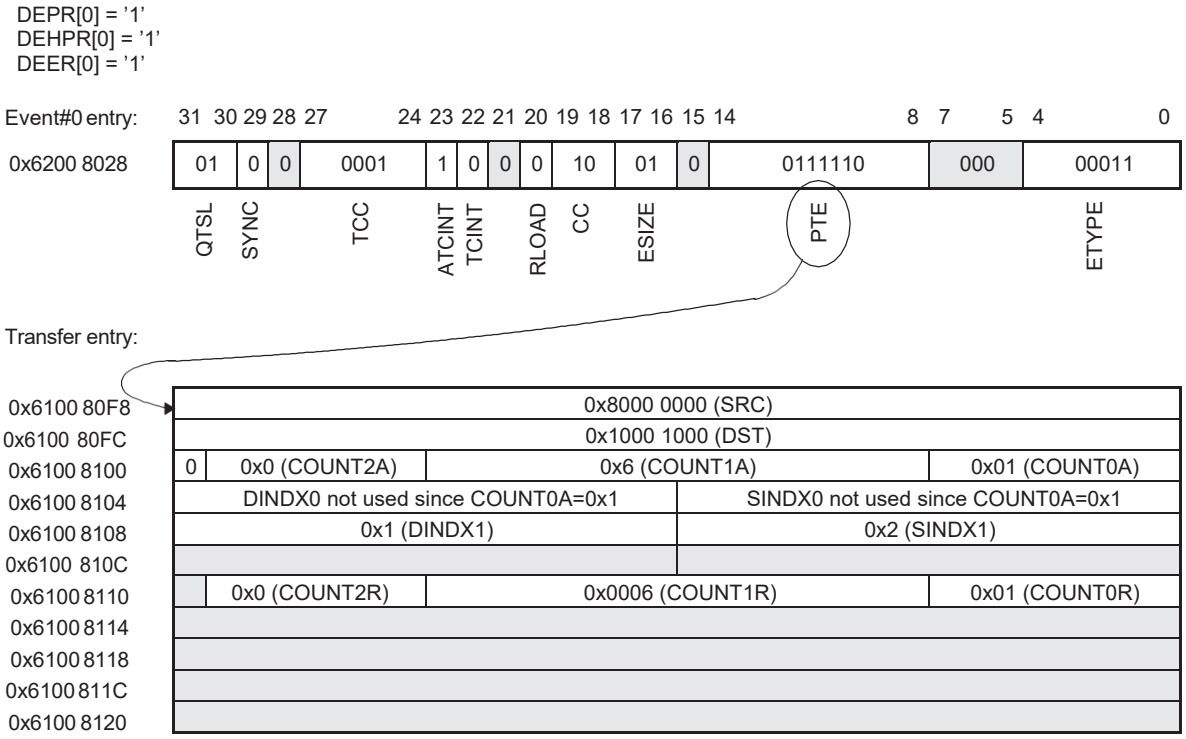
A memory snapshot for the data transfer after receiving all synchronization events is shown in Figure 3-5.

Figure 3-5. Element-Synchronized 1D Transfer Diagram (After Receiving Six Synchronization Events)



The parameters for this transfer are shown in Figure 3-6. Event zero is triggered by CPU write '0' followed by CPU write '1' to bit zero of the DETR (rising edge of the event signal 10 triggers an event since DEPR[0] = '1'). The event is processed by the HiMAX since DEHPR[0] = '1'.

Figure 3-6. Event Entry and Transfer Entry for Element-Synchronized 1D Transfer



Since frame size in this transfer is equal to one, source and destination zero indexes (SINDX0/DINDX0) are not used. Spacing between frames (in this case frame size is equal to one element) is dictated by source/destination one indexes (SINDX1/DINDX1).

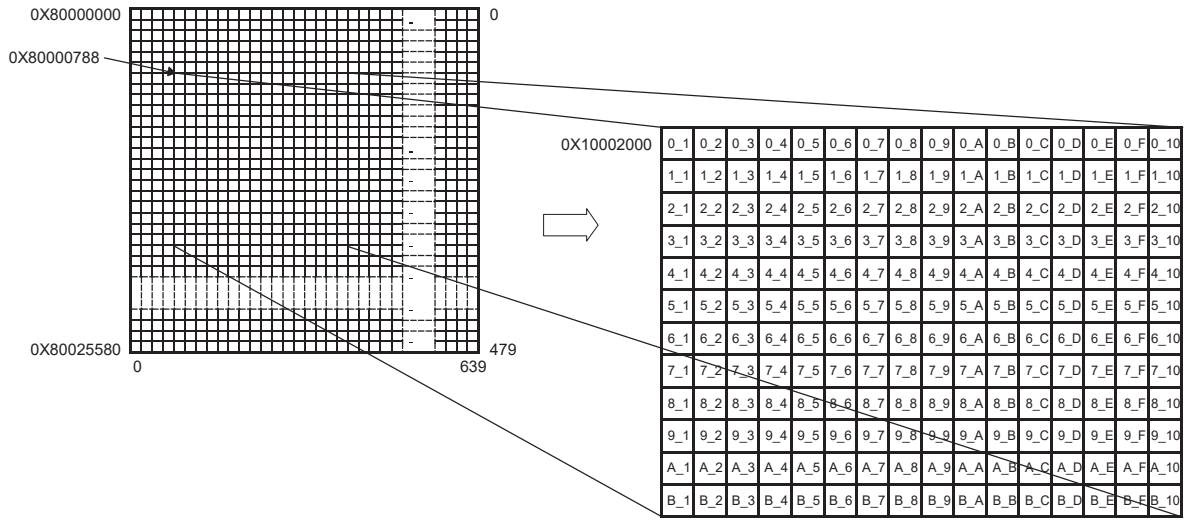
Here, the dMAX controller will move one element after receiving a synchronization event; a total of six synchronization events are required to complete the transfer. Since frame synchronization is enabled and ATCINT='1', dMAX will notify the CPU after transfer of each element has been complete, by triggering an interrupt and by setting bit one in the DTCR0 (TCC = 0x1 in the event entry).

After whole transfer is completed, dMAX will ignore subsequent synchronization events (reload is disabled RLOAD = '0' in the event entry).

3.2.4 EXAMPLE: Sub-frame Extraction

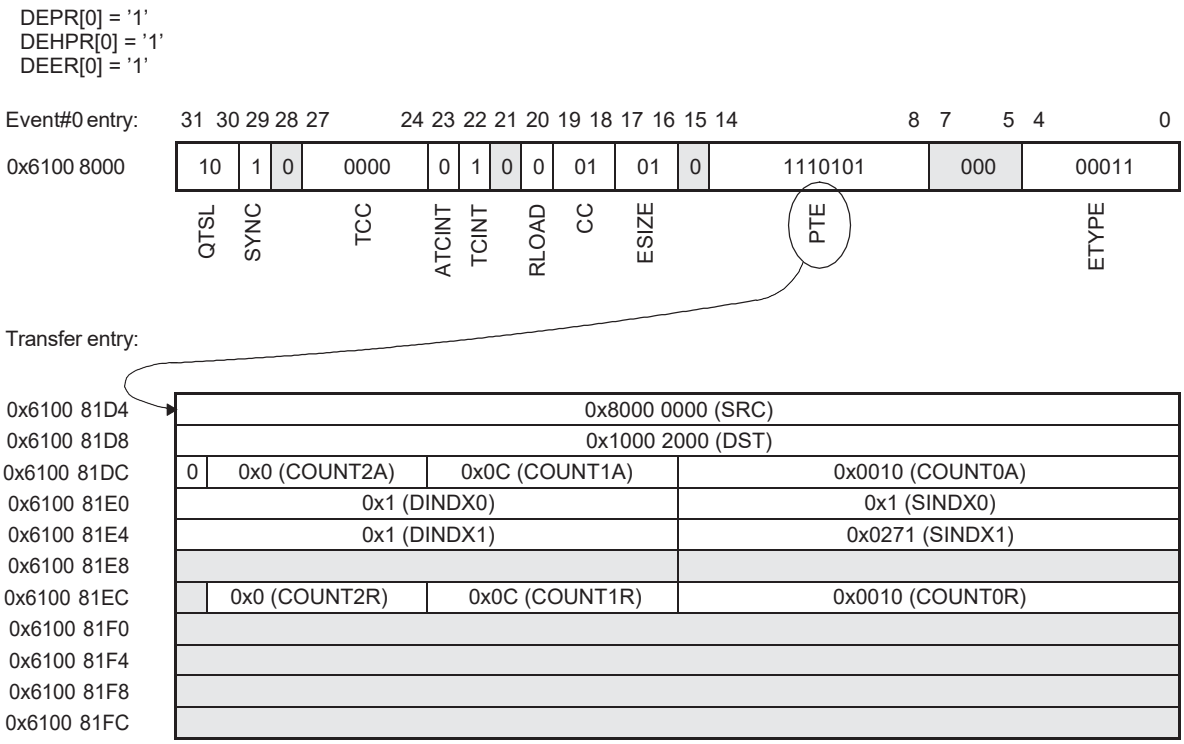
The dMAX controller has an efficient way of extracting a small frame of data from a larger one; by performing a 2-D to 1-D transfer, it can retrieve a portion of data for the CPU to process. In this example, a 640x480 sample frame of data is stored in external memory. Each sample is represented by a 16-bit half word. A 16 x 12 sub-frame is extracted for processing by the CPU. To facilitate more efficient processing time by the CPU, dMAX places the sub-frame in internal RAM. Figure 3-7 depicts the transfer of the sub-frame from external memory to internal DSP memory.

Figure 3-7. Sub-Frame Extraction



To perform this transfer, the CPU will trigger an event by creating a rising edge on a bit0 in DETR (by writing a '0', followed by writing a '1' to DETR[0]). The parameters required for dMAX to request this transfer are shown in Figure 3-8.

Figure 3-8. Event Entry and Transfer Entry for Sub-Frame Extraction Transfer



The whole transfer is completed after receiving one synchronization event (SYNC = '1' in the event entry), and subsequent synchronization events will be ignored (reload is disabled RLOAD = '0' in the event entry). After completing the transfer, dMAX will notify the CPU by triggering an interrupt (INT8) and by setting a bit zero in the DTCR0 (TCC bit field within the event entry is equal to zero).

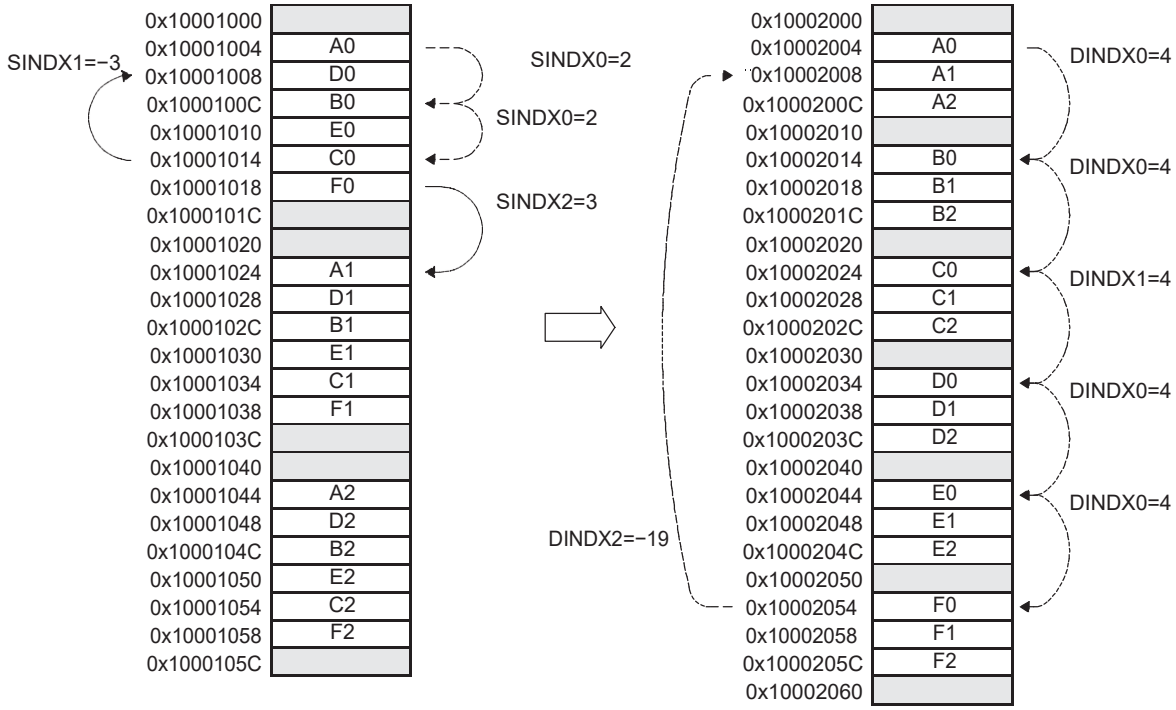
Since the maximum quantum transfer size is limited to eight elements (QTSL='10'), dMAX will split the transfer into 32 quantum transfers. The first quantum transfer will move the first eight elements from source to destination. The second quantum transfer will move the remaining four words of the first frame from the source to the destination. The third quantum transfer will move the first eight elements of the second frame, while the fourth quantum transfer moves the remaining four elements of the second frame. The last (32nd) quantum transfer will move the remaining four elements of the last (16th) frame.

At the destination, the frames are contiguous; therefore, both destination indexes, DINDX0 and DINDX1, are equal to one. At the source, elements within a frame are contiguous, while the last element of current frame is separated by 624 elements from the first element of the next frame. Therefore, SINDX0 = '1' and SINDX1 = 0x271.

3.2.5 EXAMPLE: Three Dimensional (3D) Data De-Interleaving

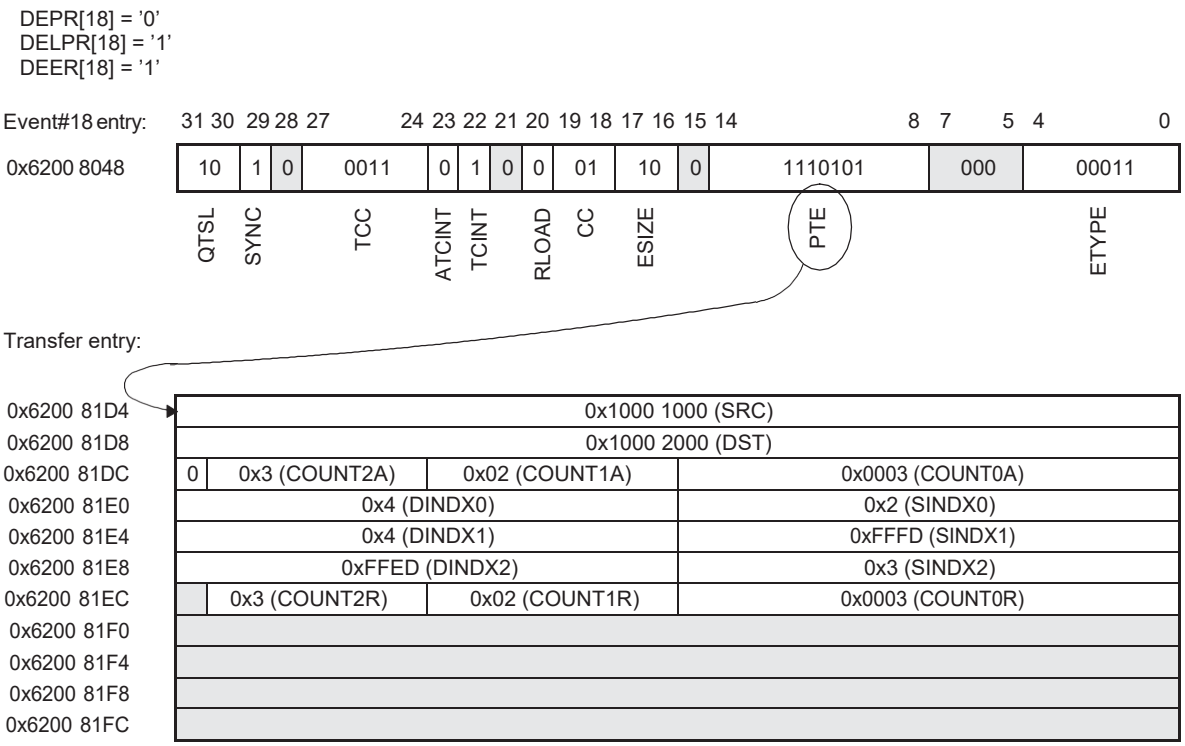
The dMAX controller supports three-dimensional (3D) transfers which are useful for sorting data and for servicing peripherals such as the McASP. In this example, dMAX is used to de-interleave the samples stored in the internal memory. Also, source and destination are both in the internal data memory. Figure 3-9 depicts the data de-interleaving transfer for this example.

Figure 3-9. 3D Data De-Interleaving



To perform this transfer, the CPU will trigger an event by creating a falling edge on a bit18 in DETR (by writing a '1', followed by writing a '0' to DETR[18]). The parameters required for dMAX to request this transfer are shown in Figure 3-10. The event is classified to the low-priority group by setting DELPR[18]='1', and is processed by the LoMAX module.

Figure 3-10. Event Entry and Transfer Entry for 3D Data De-Interleaving



The whole transfer is completed after receiving one synchronization event (SYNC = '1' in the event entry); subsequent synchronization events will be ignored (reload is disabled RLOAD = '0' in the event entry). After completing the transfer, dMAX will notify the CPU by triggering an interrupt (INT8) and by setting bit three in the DTCR0 (TCC bit field within the event entry is equal to three).

3.2.6 EXAMPLE: Ping-Pong Data Buffering Example

The CPU input and output buffers are continuously being filled or emptied in order for the CPU to process the data; therefore, it must match the pace of the dMAX controller very closely. The receive data must always be placed in memory before the CPU accesses it, and the CPU must provide the output data before dMAX transfers it. This is an unnecessary challenge. A simple technique which allows the CPU activity to be distanced from the dMAX controller activity is to use ping-pong buffering. This means that there are two sets of data buffers for all incoming and outgoing data streams. While dMAX is transferring data in to and out of the ping buffers, the CPU is manipulating the data in the pong buffers. When both the CPU and dMAX activity completes, they switch; dMAX then writes over the old input data and transfers the new output data.

The support for ping-pong buffering is built into the transfer parameters. The transfer entry describing a transfer has an active register set, and two reload parameter sets, ping and pong. After completing a transfer described by the active register set, dMAX will reload the active parameters with parameters from the reload set 1, if the PP bit is clear. If the PP bit is set, dMAX will reload the active parameter set with parameters from reload set 1.

In this example, transfers are triggered by the CPU writes to DETR. The rising edge of the DETR[18] triggers a transfer of one frame. Also in this example, dMAX uses ping-pong buffering; it is moving data from the DSP memory to two 16-bit wide FIFOs connected to asynchronous EMIF. One FIFO is mapped to address 0x9000 0000 and the second FIFO is mapped to address 0x9000 0002. After each synchronization event, dMAX moves one element from the DSP internal memory to each FIFO. The event entry and transfer entry for the configuration is shown in Figure 3-11.

Figure 3-11. Event Entry and Transfer Entry for Ping-Pong Data Buffering

DEPR[18] = '1'
DELPR[18] = '1'
DEER[18] = '1'

Event#18 entry:

	31 30	29	28	27 24	23	22	21	20	19 18	17 16	15	14 8	7 5	4 0
0x6200 8048	10	0	0	1111	0	1	0	1	01	01	0	1110101	000	00011
	QTSL	SYNC		TCC	ATCINT	TCINT		RLOAD	CC	ESIZE		PTE		ETYPE

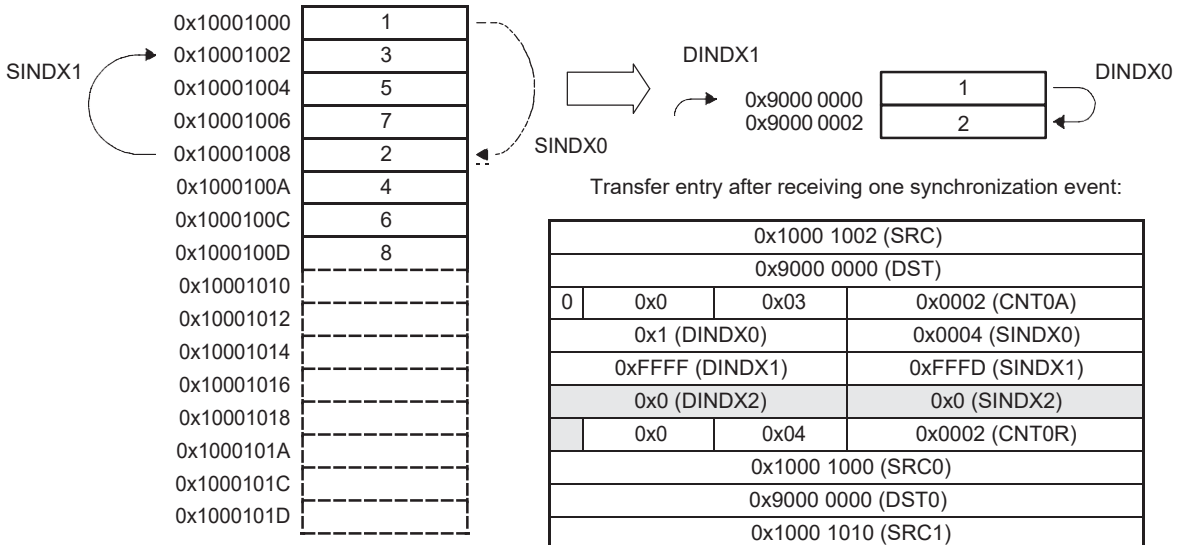
Transfer entry:

0x6200 81D4	0x1000 1000 (SRC)			
0x6200 81D8	0x9000 0000 (DST)			
0x6200 81DC	0	0x0 (COUNT2A)	0x04 (COUNT1A)	0x0002 (COUNT0A)
0x6200 81E0	0x1 (DINDX0)			0x4 (SINDX0)
0x6200 81E4	0xFFFF (DINDX1)			0xFFFD (SINDX1)
0x6200 81E8	0x0 (DINDX2)			0x0 (SINDX2)
0x6200 81EC		0x0 (COUNT2R)	0x04 (COUNT1R)	0x0002 (COUNT0R)
0x6200 81F0	0x1000 1000 (SRC0)			
0x6200 81F4	0x9000 0000 (DST0)			
0x6200 81F8	0x1000 1010 (SRC1)			
0x6200 81FC	0x9000 0000 (DST1)			

The data in the internal memory is organized in two ping and two pong buffers. The first ping buffer starts at address 0x10001000 and the second ping buffer starts at address 0x10001008. The data from the first ping buffer should be moved to the first FIFO. The data from the second ping buffer should be moved to the second FIFO. The pong buffers are organized similarly.

After receiving the first synchronization event, dMAX moves the first frame of two elements from the source ping buffers to the destination. The first frame consists of the first element from the first ping buffer and the first element from the second ping buffer. The first element is transferred to the FIFO at address 0x9000 0000, and the second element is transferred to the FIFO mapped at address 0x9000 0002. A memory snapshot after receiving the first synchronization event is shown in Figure 3-12.

Figure 3-12. Ping-Pong Data Buffering After Receiving the First Synchronization Event

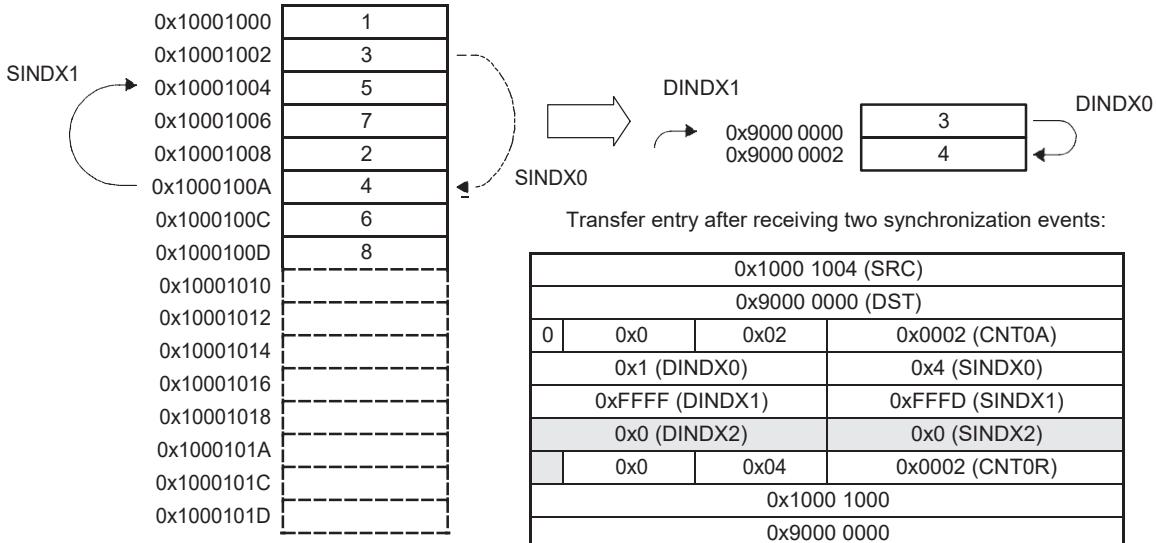


Transfer entry after receiving one synchronization event:

0x1000 1002 (SRC)			
0x9000 0000 (DST)			
0	0x0	0x03	0x0002 (CNT0A)
0x1 (DINDX0)			0x0004 (SINDX0)
0xFFFF (DINDX1)			0xFFFD (SINDX1)
0x0 (DINDX2)			0x0 (SINDX2)
	0x0	0x04	0x0002 (CNT0R)
0x1000 1000 (SRC0)			
0x9000 0000 (DST0)			
0x1000 1010 (SRC1)			
0x9000 0000 (DST1)			

After receiving the second synchronization event, dMAX will move the second frame of two elements from source ping buffers to destination. The second frame consists of the second element from the first ping buffer and the second element from the second ping buffer. The first frame element is transferred to the FIFO at address 0x9000 0000, and the second frame element is transferred to the FIFO mapped at address 0x9000 0002. A memory snapshot after receiving the second synchronization event is shown in Figure 3-13.

Figure 3-13. Ping-Pong Data Buffering After Receiving the Second Synchronization Event

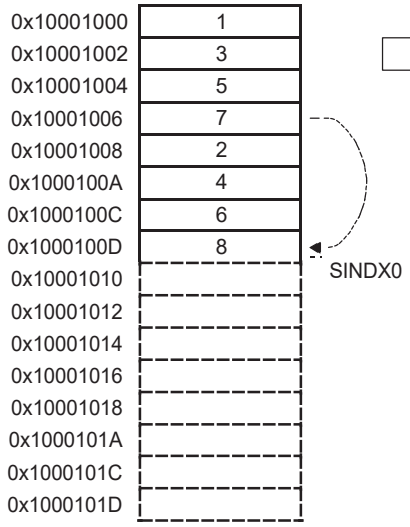


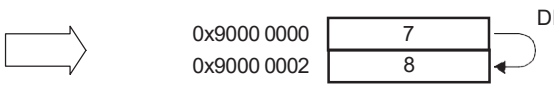
Transfer entry after receiving two synchronization events:

0x1000 1004 (SRC)			
0x9000 0000 (DST)			
0	0x0	0x02	0x0002 (CNT0A)
0x1 (DINDX0)			0x4 (SINDX0)
0xFFFF (DINDX1)			0xFFFD (SINDX1)
0x0 (DINDX2)			0x0 (SINDX2)
	0x0	0x04	0x0002 (CNT0R)
0x1000 1000			
0x9000 0000			
0x1000 1010			
0x9000 0000			

After receiving the fourth synchronization event, dMAX will move the fourth frame of two elements from source ping buffers to destination. The fourth frame consists of the fourth element from the first ping buffer and the fourth element from the second ping buffer. The first frame element is transferred to the FIFO at address 0x9000 0000, and the second frame element is transferred to the FIFO mapped at address 0x9000 0002. A memory snapshot after receiving the fourth synchronization event is shown in Figure 3-14.

Figure 3-14. Ping-Pong Data Buffering After Receiving the Fourth Synchronization Event





Transfer entry after receiving four synchronization events:

0x1000 1010 (SRC)			
0x9000 0000 (DST)			
1	0x0	0x04	0x0002 (CNT0A)
0x0001 (DINDX0)			0x0004 (SINDX0)
0xFFFF (DINDX1)			0xFFFD (SINDX1)
0x0 (DINDX2)			0x0 (SINDX2)
	0x0	0x04	0x0002 (CNT0R)
0x1000 0000 (SRC0)			
0x9000 0000 (DST0)			
0x1000 1010 (SRC1)			
0x9000 0000 (DST1)			

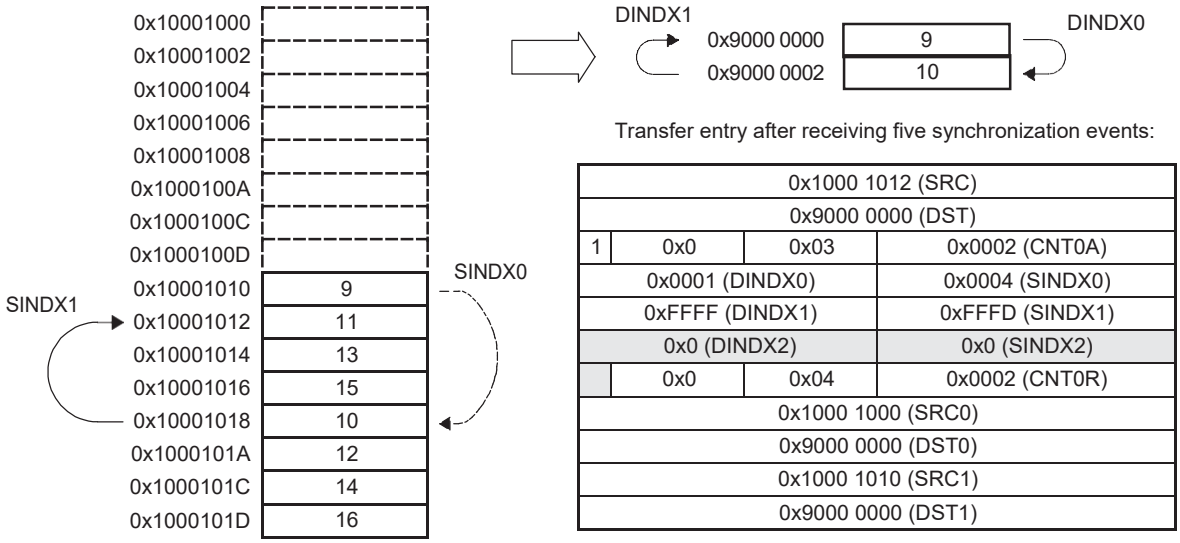
After transfer of the fourth ping frame, the transfer is complete and the counters of all three dimensions are expired. The dMAX controller sets bit 7 in the DTCR1 (since the TCC code specified in the event entry is equal to 15), and it triggers a CPU interrupt (INT8). Since reload is enabled (RLOAD bit is set to one in the event entry), dMAX reloads active counter values from the counter reference register.

Reload register set 0 (SRC0 and DST0) specifies the ping buffer, while reload register set 1 (SRC1 and DST1) specifies the pong buffer. During reload, dMAX swaps between the ping and pong buffers. The PP bit is used to determine which reload register set should be loaded in the set of active registers.

In this example, the PP bit was cleared prior to enabling the event; therefore, during the first reload, dMAX will load active source and destination registers from reload register set 1 (SRC1 and DST1 will be loaded in the active SRC and DST registers respectively). After reload is performed, dMAX sets the PP bit to 1.

After receiving the fifth synchronization event, dMAX will move the first frame of two elements from source pong buffers to destination. The frame consists of the first element from the first pong buffer and the first element from the second pong buffer. A memory snapshot after receiving the fifth synchronization event is shown in Figure 3-15.

Figure 3-15. Ping-Pong Data Buffering After Receiving the Fifth Synchronization Event



Transfer entry after receiving five synchronization events:

0x1000 1012 (SRC)			
0x9000 0000 (DST)			
1	0x0	0x03	0x0002 (CNT0A)
0x0001 (DINDX0)			0x0004 (SINDX0)
0xFFFF (DINDX1)			0xFFFD (SINDX1)
0x0 (DINDX2)			0x0 (SINDX2)
	0x0	0x04	0x0002 (CNT0R)
0x1000 1000 (SRC0)			
0x9000 0000 (DST0)			
0x1000 1010 (SRC1)			
0x9000 0000 (DST1)			

After receiving synchronization Event 8, dMAX moves the fourth frame of two elements from source pong buffers to destination. The frame consists of the fourth element from the first pong buffer and the fourth element from the second pong buffer. The first frame element is transferred to the FIFO at address 0x9000 0000, and the second frame element is transferred to the FIFO mapped to the FIFO at address 0x9000 0002.

After transfer of the fourth pong frame, the transfer is complete and the counters of all three dimensions are expired. The dMAX controller sets bit seven in the DTCR1 (since the TCC code specified in the event entry is equal to 15), and it triggers a CPU interrupt (INT8). Since reload is enabled (RLOAD bit is set to one in the event entry), the dMAX controller reloads active counter values from the counter reference register.

Since the PP bit is set, dMAX loads active source and destination registers from reload register set zero (SRC0 and DST0 will be loaded in the active SRC and DST registers, respectively). After reload is performed, dMAX clears the PP bit to zero .

3.3 FIFO Transfer Examples

The FIFO write and FIFO read transfers are used to write-to and read-from a software-defined FIFO in the device memory map. The FIFO transfers can be 1-dimensional (1D) or 2-dimensional (2D), with 1D transfers implemented as special cases of 2D transfers where the counter for the second dimension is equal to zero. Indexes for source (FIFO write) and destination (FIFO read) can be specified for each transfer dimension. In addition, a table of delay-tap offsets is used for 2D FIFO transfers to facilitate indexing into the FIFO. All FIFO transfers support reloading of the active address and counters at the completion of a transfer. A different set of delay-tap offset can also be reloaded to add further flexibility.

The following sections present the proper setup procedure for FIFO transfers and provide various example configurations. They include complete register configurations, along with graphical representations of each example transfer.

3.3.1 Steps Required to Set Up a FIFO Transfer

The following steps are required to set up a dMAX FIFO transfer:

1. The priority of an event that will be used to trigger the FIFO data transfer must be defined. To put the event into the high-priority group, the corresponding bit in the DEHPR should be set to one. To put the event into the low-priority group the corresponding bit in the DELPR should be set to one.
2. The event signal edge (rising/falling) that will be used to trigger an event must be defined. To trigger an event on the rising edge of the event signal, the corresponding bit in the DEPR should be set to one. To trigger an event on the falling edge of the event signal, the corresponding bit in the DEPR should be cleared to zero.
3. If the event is sorted to the high-riority group, its event entry in the HiMAX PaRAM must be defined. If the event is sorted to the low-priority group, its event entry in the LoMAX PaRAM must be defined. The following bit fields in the event entry must be configured:
   - The ETYPE bit field must be set to '00100' for a FIFO write transfer, or '00101' for a FIFO read transfer.
   - The PTE bit field must be set to point to the location in the PaRAM where the event's transfer entry will be stored.
   - The EWM bit field should be set to either enable (EWM == 1) or disable (EWM == 0) FIFO watermark notifications.
   - The RLOAD bit field should be set to 1 to enable reloading of the active counters and address when the transfer is complete, or set to 0 to avoid reloading.
   - The TCINT bit should be set if a notification to the CPU is required (only when the transfer is complete). The ATCINT bit should be set if notification to the CPU is required after transfer of each frame.
   - The TCC field should be set to indicate the desired bit to be set in the DTCR to report transfer status to the CPU.
   - The SYNC bit should be set to have the entire transfer complete after one event. Alternatively, the SYNC bit should be cleared to have only one frame complete after each event.
   - The QTSL bit field should be set to define the granularity with which dMAX breaks up a large transfer into a number of smaller sub-transfers.
4. The following fields in the transfer entry must be properly configured:
   - The active and reference address and counters should be programmed, along with the element indexes.
   - The pointer to the FIFO descriptor (PFD) should be programmed to point to the FIFO descriptor's base address.
   - Pointers to delay tables should be programmed if performing a 2D transfer. 1D transfers do not use delay tables and instead always read and write directly from/to the FIFO read and write pointer locations.
   - If RLOAD is enabled in the associated event entry, the PP bit in the transfer entry must also be properly configured. When PP='0', reload register set one is loaded in the set of active registers after the transfer is completed. When PP='1', reload register set zero is loaded in the set of active registers after the transfer is completed.

5. A FIFO descriptor should be placed somewhere in the device memory to define the characteristics of the FIFO. The following fields of the FIFO descriptor should be programmed:
   - The FIFO base address (FBA) field should be programmed to specify where the FIFO should begin in memory.
   - The FIFO element size (ESIZE) field should be programmed to select the size of elements the FIFO will hold.
   - The FIFO SIZE field should be programmed to specify the number of elements that the FIFO can hold.
   - The read pointer (RP) and write pointer (WP) should be programmed to the desired initial values, in terms of element offsets from the base element.
   - The two status codes, FMSC and EMSC, should be programmed to select the code to report when indicating FIFO full and FIFO empty conditions. These fields should still be set to facilitate error reporting even if watermarks are disabled. If watermarks are not enabled, these fields should be set to the same value to conserve codes.
   - The two level marks, EMARK and FMARK, should be programmed to select the desired watermark levels. It is not required to program these fields if watermarks are not enabled.
   - Finally, the FIFO Full bit (FF) should be set to 1 if WP = RP and it is desired to start with the FIFO in the full state. Otherwise (and in most instances) FF should be cleared to 0. The FF bit should not be subsequently modified.
6. Delay tables should be placed somewhere in the device memory to define the delay-tap offset values to be used for a 2D transfer. These tables simply consist of a list of 20-bit values aligned on 32-bit boundaries. Delay tables are not required for 1D transfers.
7. The event must be enabled by setting the event's corresponding bit in the DEER register.

Once an event is enabled, dMAX will begin the data transfer when it detects the appropriate transition in the event signal.

If ATCINT = '1' or TCINT='1', dMAX will signal the transfer status to the CPU by generating an interrupt on line INT8. In addition, dMAX will set a bit, specified by the TCC bit field of the event entry, in the DTCR. The DTCR bits are mirrored in the DESR. The read-only DESR is located inside the CPU module and can be accessed by the CPU with minimum overhead. The fastest way of monitoring the DTCR is to read its copy from the DESR.

To keep receiving notifications from a particular dMAX channel, the CPU must clear the transfer completion bit used by the dMAX channel after each notification. The CPU should use the read-only DESR to find out which transfer completion bits have been set, and it should clear them by writing '1' to the corresponding bits in the DTCR.

The dMAX controller will also generate an interrupt, INT7, to the CPU when the FIFO status changes in the following ways:

- A FIFO error occurs
- A FIFO watermark is reached and watermarks are enabled in the event entry

Along with generating INT7, dMAX will also set notification bits in one or more places. For watermark notifications, the programmed bit in one of the DFSRs will be set. For error notifications, both EMSC and FMSC bits will be set in the appropriate DFSR, along with an error code in the error field (EFIELD) within the corresponding FIFO descriptor.

To keep receiving FIFO status notifications, the CPU must clear the FIFO status bit(s) used by the dMAX channel after each notification. The CPU should clear these bits by writing '1' to the corresponding bits in the appropriate DFSR.

After enabling the event, it is not recommended to access the event entry, active register set, indexes, and the reference counter bit fields within the transfer entry.

3.3.2 EXAMPLE: 1D FIFO Write Transfer

A 1D FIFO write transfer is the most basic transfer that can be performed by dMAX to move data into a FIFO. This transfer simply copies data from the source address into the FIFO in a continuous stream, starting at the write pointer.

In this example, a section of data is copied from internal memory to a FIFO in external memory. The FIFO is programmed in the FIFO descriptor to have a size of 20 elements and an element size of 16-bits. In addition, the write pointer and the read pointer are both set to 0 at the start of the transfer. The data block to be placed into the FIFO is four half-words and resides at address 0x10001000. The base address of the FIFO is set to 0x80000000. The state of the FIFO before the transfer is shown in Figure 3-16, and the state of the FIFO after the transfer is shown in Figure 3-17.

Figure 3-16. 1D FIFO Write Diagram (Before Transfer)

Source address	Data
0x1000 1000	1
0x1000 1002	2
0x1000 1004	3
0x1000 1006	4

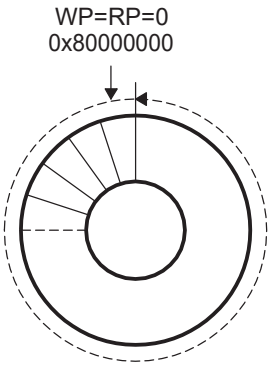
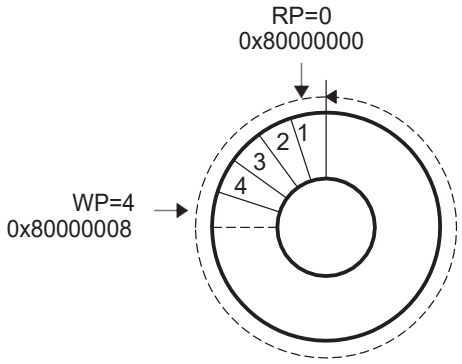


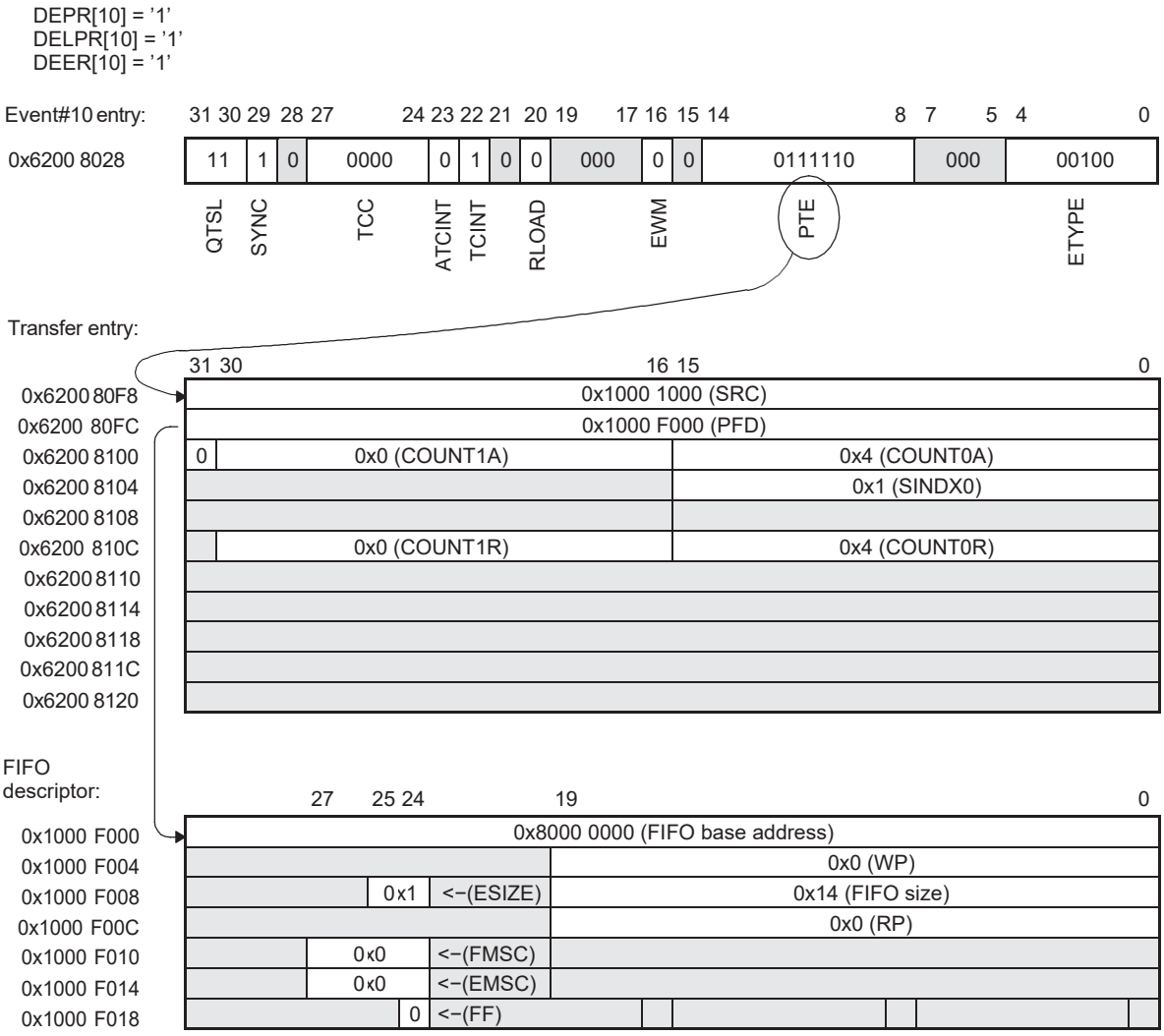
Figure 3-17. 1D FIFO Write Diagram (After Transfer)

Source address	Data
0x1000 1000	1
0x1000 1002	2
0x1000 1004	3
0x1000 1006	4



The parameters for this transfer are shown in Figure 3-18. Event 10 is used by the transfer, and the event is triggered by a CPU writing a '0' followed by a '1' to bit one of the DETR. A rising edge on event signal 10 triggers the event because DEPR[10] = '1'. The event is processed by the LoMAX because DELPR[10] = '1'.

Figure 3-18. Event Entry, Transfer Entry, and FIFO Descriptor for 1D FIFO Write



The entire transfer completes on receiving one synchronization event (SYNC = '1' in the event entry), and subsequent synchronization events will be ignored because reload is disabled in the event entry. After completing the transfer, dMAX notifies the CPU by generating an interrupt (INT8) and by setting bit zero in the DTCR (TCC bit field within the event entry is equal to zero).

Because WP = RP, the FF bit must be programmed as 0 to tell dMAX that the FIFO is empty, not full. Although watermarks are disabled in the event entry, the FMSC and EMSC should still be programmed because they are also used for error reporting. FMSC and EMSC are set to the same value in this example to conserve status bits. If watermarks are used, these fields should instead be set to different values.

The dMAX controller will complete this transfer using one quantum transfer (QTSL='11' in the event entry).

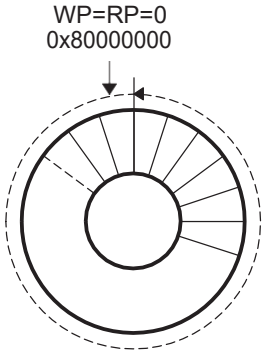
3.3.3 EXAMPLE: 2D FIFO Write Transfer with Reload

Two-dimensional (2D) FIFO write transfers allow data to be written into a FIFO at user-programmed delay-tap offsets from the write pointer (WP). In this example, four data elements are to be copied from the internal memory to the FIFO in external memory using a set of delay-tap offsets (0, 2). The transfer is then reloaded, and another four data elements are copied into the FIFO using a different set of delay-tap offsets (0, 6). The eight half-words of source data to be transferred start at address 0x10001000, and the 20-element FIFO starts at address 0x8000000. A source index of 1 is used for the first dimension of the transfer, while a source index of two is used for the second dimension. Finally, SRC RELOAD ADDRESS 1 is set to 0x1000100A to provide a new active address for the reloaded (second) transfer.

A memory snapshot of the data transfer before receiving the first synchronization event is shown in Figure 3-19.

Figure 3-19. 2D FIFO Write Transfer Diagram (Before First Synchronization Event)

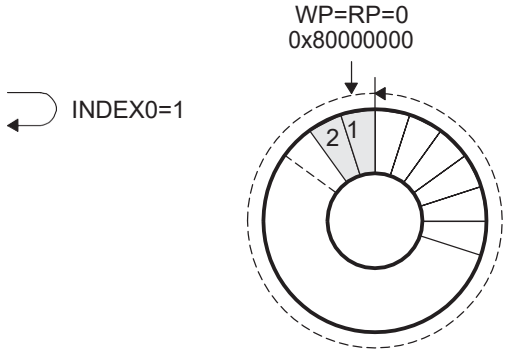
Source address	Data
0x1000 1000	1
0x1000 1002	2
0x1000 1004	
0x1000 1006	3
0x1000 1008	4
0x1000 100A	5
0x1000 100C	6
0x1000 100E	
0x1000 1010	7
0x1000 1012	8



A memory snapshot of the data transfer after receiving the first synchronization event is shown in Figure 3-20. A delay-tap offset of 0 is used when writing this frame into the FIFO.

Figure 3-20. 2D FIFO Write Transfer Diagram (After Receiving the First Synchronization Event)

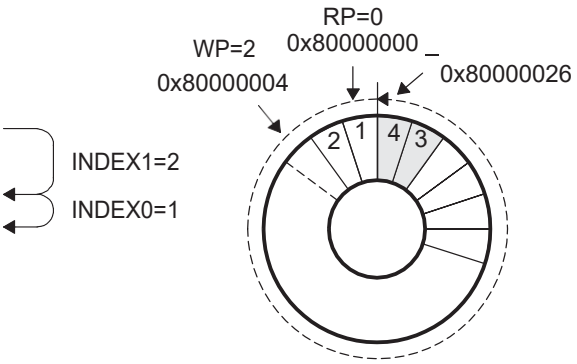
Source address	Data
0x1000 1000	1
0x1000 1002	2
0x1000 1004	
0x1000 1006	3
0x1000 1008	4
0x1000 100A	5
0x1000 100C	6
0x1000 100E	
0x1000 1010	7
0x1000 1012	8



A memory snapshot of the data transfer after receiving the second synchronization event is shown in Figure 3-21. A delay-tap offset of 2 is used when writing this frame into the FIFO. This means that the first element in this frame is written two slots behind the location of the WP. After transferring this frame, the WP is incremented by 2.

Figure 3-21. 2D FIFO Write Transfer Diagram (After Receiving the Second Synchronization Event)

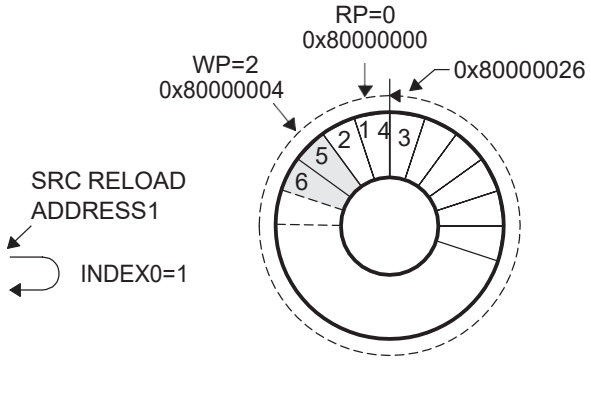
Source address	Data
0x1000 1000	1
0x1000 1002	2
0x1000 1004	
0x1000 1006	3
0x1000 008	4
0x1000 100A	5
0x1000 100C	6
0x1000 100E	
0x1000 1010	7
0x1000 1012	8



After the second frame has been transferred, the active address is reloaded with SRC RELOAD ADDRESS 1, and the counters are reloaded with the reference counters. In addition, the second delay table is used for the next transfer. A memory snapshot for the data transfer after receiving the third synchronization event is shown in Figure 3-22. A delay-tap offset of 0 is used when writing this frame into the FIFO.

Figure 3-22. 2D FIFO Write Transfer Diagram (After Receiving Three Synchronization Events)

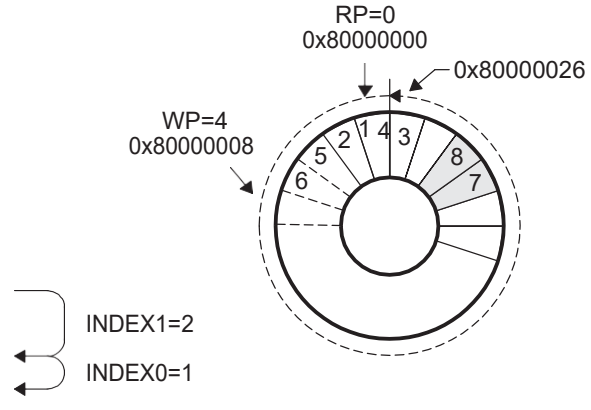
Source address	Data
0x1000 1000	1
0x1000 1002	2
0x1000 1004	
0x1000 1006	3
0x1000 1008	4
0x1000 100A	5
0x1000 100C	6
0x1000 100E	
0x1000 1010	7
0x1000 1012	8



A memory snapshot for the data transfer after receiving the fourth and final synchronization event is shown in Figure 3-23. A delay-tap offset of 6 is used when writing this frame into the FIFO. This means that the first element in this frame is written six slots behind the location of the WP. After transferring this frame, the WP is incremented by 2.

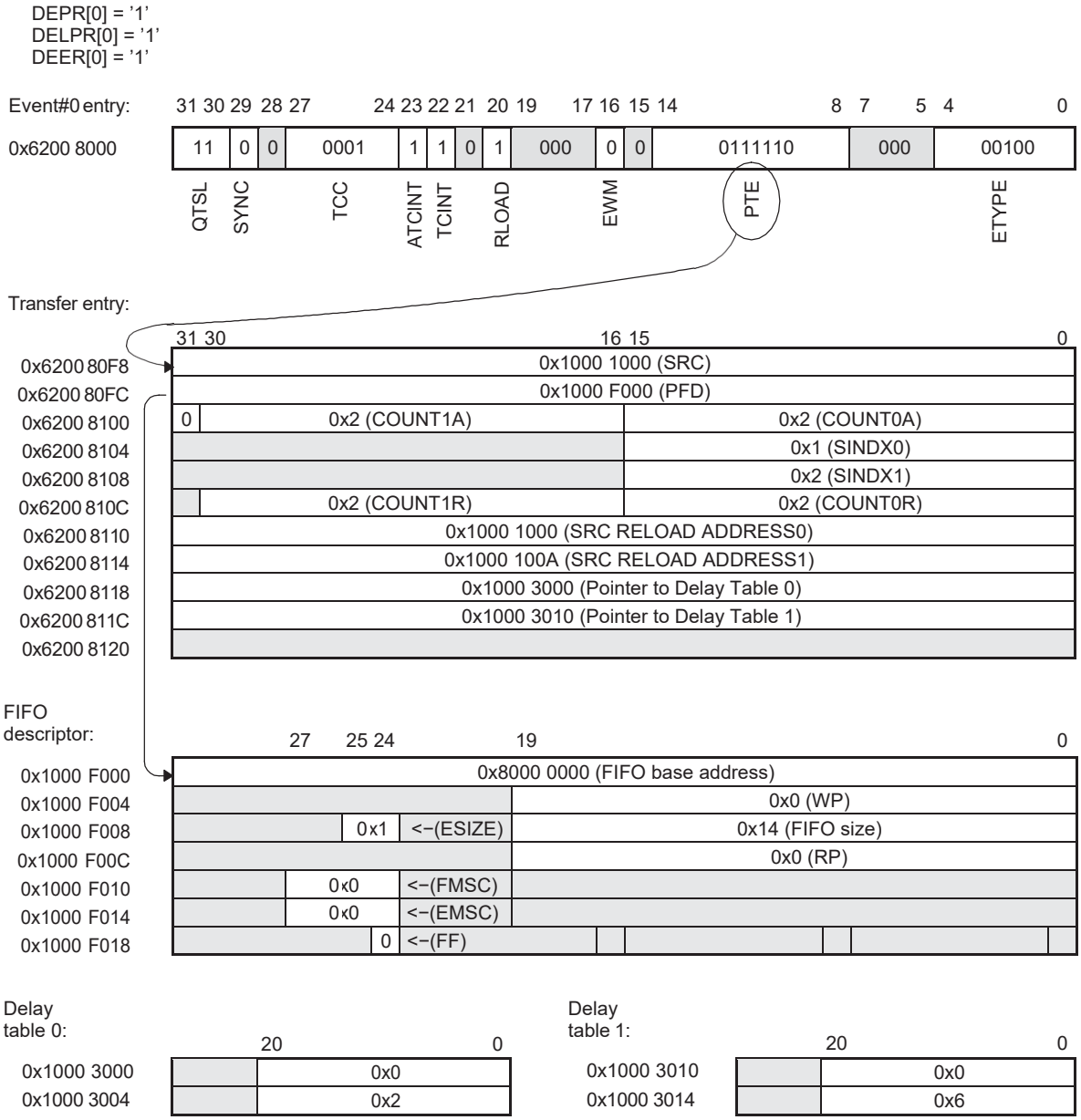
Figure 3-23. 2D FIFO Write Transfer Diagram (After Receiving All Synchronization Events)

Source address	Data
0x1000 1000	1
0x1000 1002	2
0x1000 1004	
0x1000 1006	3
0x1000 1008	4
0x1000 100A	5
0x1000 100C	6
0x1000 100E	
0x1000 1010	7
0x1000 1012	8



The parameters for this transfer are shown in Figure 3-24. Event zero is used to trigger this transfer. In this case, the event zero is triggered by a CPU write '0' followed by a CPU write '1' to bit zero of the DETR (rising edge of the event signal ten triggers an event since DEPR[0] = '1'). The event is processed by the LoMAX since DELPR[0] = '1'.

Figure 3-24. Event Entry, Transfer Entry, FIFO Descriptor, and Delay Tables for 2D FIFO Write Transfer



Here, dMAX will move two elements after receiving each synchronization event; a total of two synchronization events are required to complete the transfer. Since frame synchronization is enabled and ATCINT='1', dMAX will notify the CPU after completion of each frame of the transfer by triggering an interrupt and by setting bit 1 in the DTCR (TCC =0x1 in the event entry).

As in the previous example, the FF bit must be programmed as 0 to tell the dMAX that the FIFO is empty, not full. Although watermarks are disabled in the event entry, the FMSC and EMSC should still be programmed because they are also used for error reporting. FMSC and EMSC are set to the same value in this example to conserve status bits. If watermarks are used, these fields should instead be set to different values.

After the whole transfer is completed dMAX will reload the active address and counters and wait for subsequent synchronization events. This example only shows one reload and the following two synchronization events. In practice, dMAX will continue to reload the source address and counters at the completion of each subsequent transfer.

3.3.4 EXAMPLE: 1D FIFO Read Transfer

A 1D FIFO read transfer is the most basic transfer that can be performed by dMAX to move data out of a FIFO. This transfer simply copies data from the FIFO to the destination address in a continuous stream starting at the read pointer.

This example shows how to retrieve the FIFO data that was written to the FIFO in the 1D FIFO write transfer example. It picks up at the completion of the 1D FIFO write transfer example and uses the same FIFO descriptor. The FIFO has a size of 20 elements and an element size of 16 bits. In addition, the write pointer points to element 4 and the read pointer points to element 0 at the start of this transfer. The data block to be moved from the FIFO is four half-words and will be placed at address 0x10002000. The base address of the FIFO is 0x80000000. The state of the memory before the transfer is shown in Figure 3-25, and the state of the memory after the transfer is shown in Figure 3-26.

Figure 3-25. 1D FIFO Read Diagram (Before Transfer)

Dest address	Data
0x1000 2000	
0x1000 2002	
0x1000 2004	
0x1000 2006	

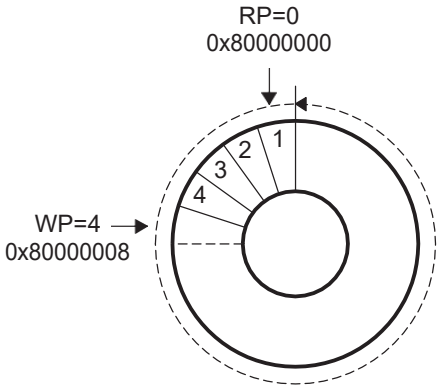
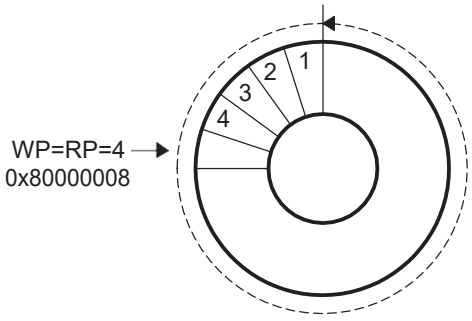


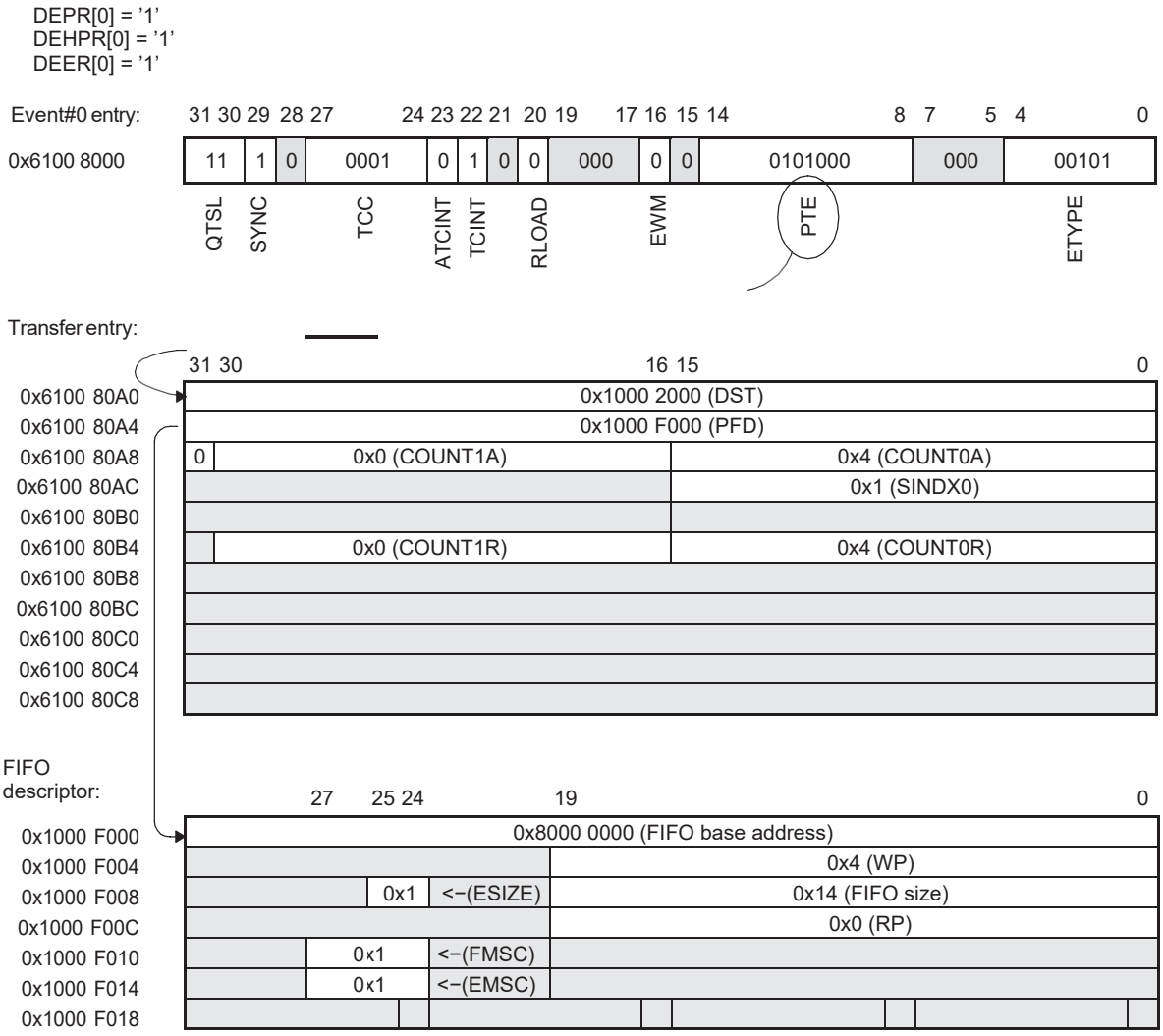
Figure 3-26. 1D FIFO Read Diagram (After Transfer)

Dest address	Data
0x1000 2000	1
0x1000 2002	2
0x1000 2004	3
0x1000 2006	4



The parameters for this transfer are shown in Figure 3-27. The FIFO descriptor does not need to be reprogrammed if this transfer is performed directly after the transfer in the 1D FIFO write transfer example. Event 0 is used by the transfer, and the event is triggered by the CPU writing a '0' followed by a '1' to bit zero of the DETR. A rising edge on event signal 0 triggers the event because DEPR[0] = '1'. The event is processed by the HiMAX because DEHPR[0] = '1'.

Figure 3-27. Event Entry, Transfer Entry, and FIFO Descriptor for 1D FIFO Read



The entire transfer completes on receiving one synchronization event (SYNC = '1' in the event entry); subsequent synchronization events will be ignored because reload is disabled in the event entry. After completing the transfer, dMAX notifies the CPU by generating an interrupt (INT8) and by setting bit 1 in the DTCR (TCC bit field within the event entry is equal to 1).

Although watermarks are disabled in the event entry, the FMSC and EMSC fields should still be programmed because they are also used for error reporting. FMSC and EMSC are set to the same value in this example to conserve status bits. If watermarks are used, these fields should instead be set to different values.

The dMAX controller will complete this transfer using one quantum transfer (QTSL='11' in the event entry).

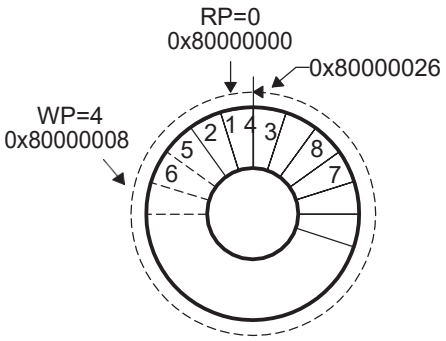
3.3.5 EXAMPLE: 2D FIFO Read Transfer with Reload

2D FIFO read transfers allow data to be read from the FIFO at user-programmed, delay-tap offsets from the read pointer (RP). This example picks up where the 2D FIFO write transfer leaves off, draining the eight elements that were copied into the FIFO. The same FIFO descriptor is used. In the first transfer, four data elements are copied from the FIFO into internal memory using a set of delay-tap offsets (0, 2). The transfer is then reloaded, and another four data elements are copied from the FIFO using a different set of delay-tap offsets (0, 6). The eight half-words are transferred to address 0x10002000, and the 20-element FIFO starts at address 0x8000000. A destination index of 1 is used for the first dimension of the transfer, while a destination index of 2 is used for the second dimension. Finally, DST RELOAD ADDRESS 1 is set to 0x1000200A to provide a new active address for the reloaded (second) transfer.

A memory snapshot of the data transfer before receiving the first synchronization event is shown in Figure 3-28.

Figure 3-28. 2D FIFO Read Transfer Diagram (Before First Synchronization Event)

Source address	Data
0x1000 2000	
0x1000 2002	
0x1000 2004	
0x1000 2006	
0x1000 2008	
0x1000 200A	
0x1000 200C	
0x1000 200E	
0x1000 2010	
0x1000 2012	

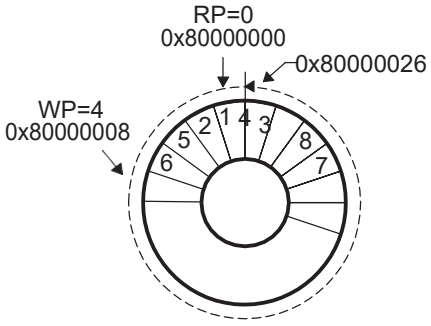


A memory snapshot of the data transfer after receiving the first synchronization event is shown in Figure 3-29. A delay-tap offset of 0 is used when reading this frame from the FIFO.

Figure 3-29. 2D FIFO Read Transfer Diagram (After Receiving the First Synchronization Event)

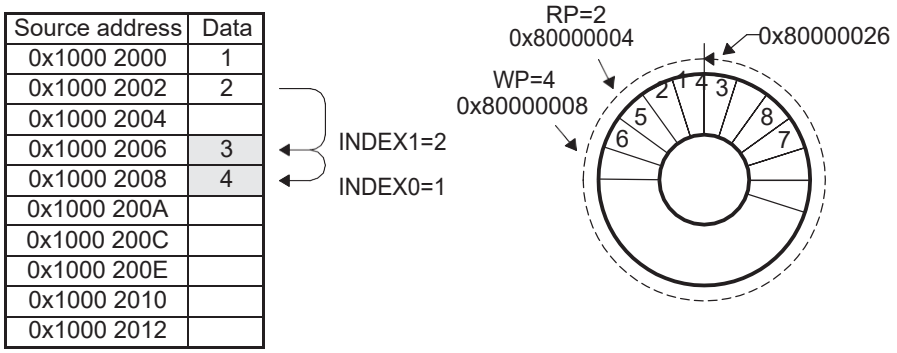
Source address	Data
0x1000 2000	1
0x1000 2002	2
0x1000 2004	
0x1000 2006	
0x1000 2008	
0x1000 200A	
0x1000 200C	
0x1000 200E	
0x1000 2010	
0x1000 2012	

INDEX0=1



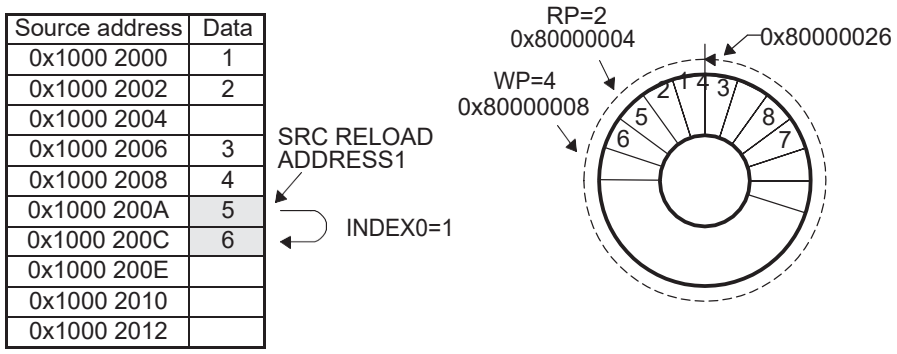
A memory snapshot for the data transfer after receiving the second synchronization event is shown in Figure 3-30. A delay-tap offset of 2 is used when reading this frame from the FIFO. This means that the first element in this frame is read from two slots behind the RP. After transferring this frame, the RP is incremented by 2.

Figure 3-30. 2D FIFO Read Transfer Diagram (After Receiving the Second Synchronization Event)



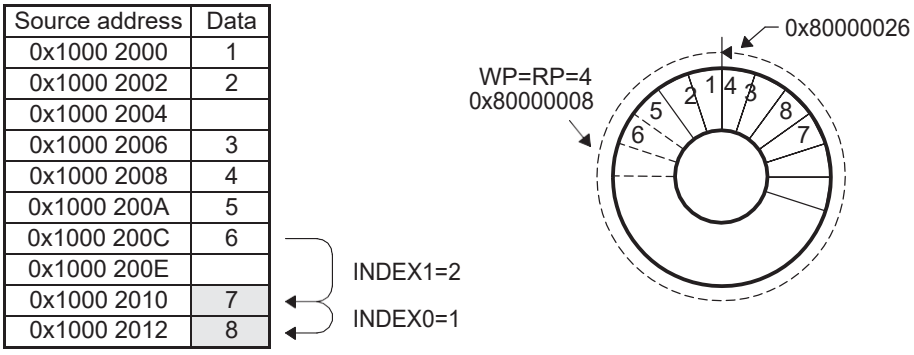
After the second frame has been transferred, the active address is reloaded with DST RELOAD ADDRESS 1, and the counters are reloaded with the reference counters. In addition, the second delay table is used for the next transfer. A memory snapshot for the data transfer after receiving the third synchronization event is shown in Figure 3-31. A delay-tap offset of 0 is used when reading this frame from the FIFO.

Figure 3-31. 2D FIFO Read Transfer Diagram (After Receiving Three Synchronization Events)



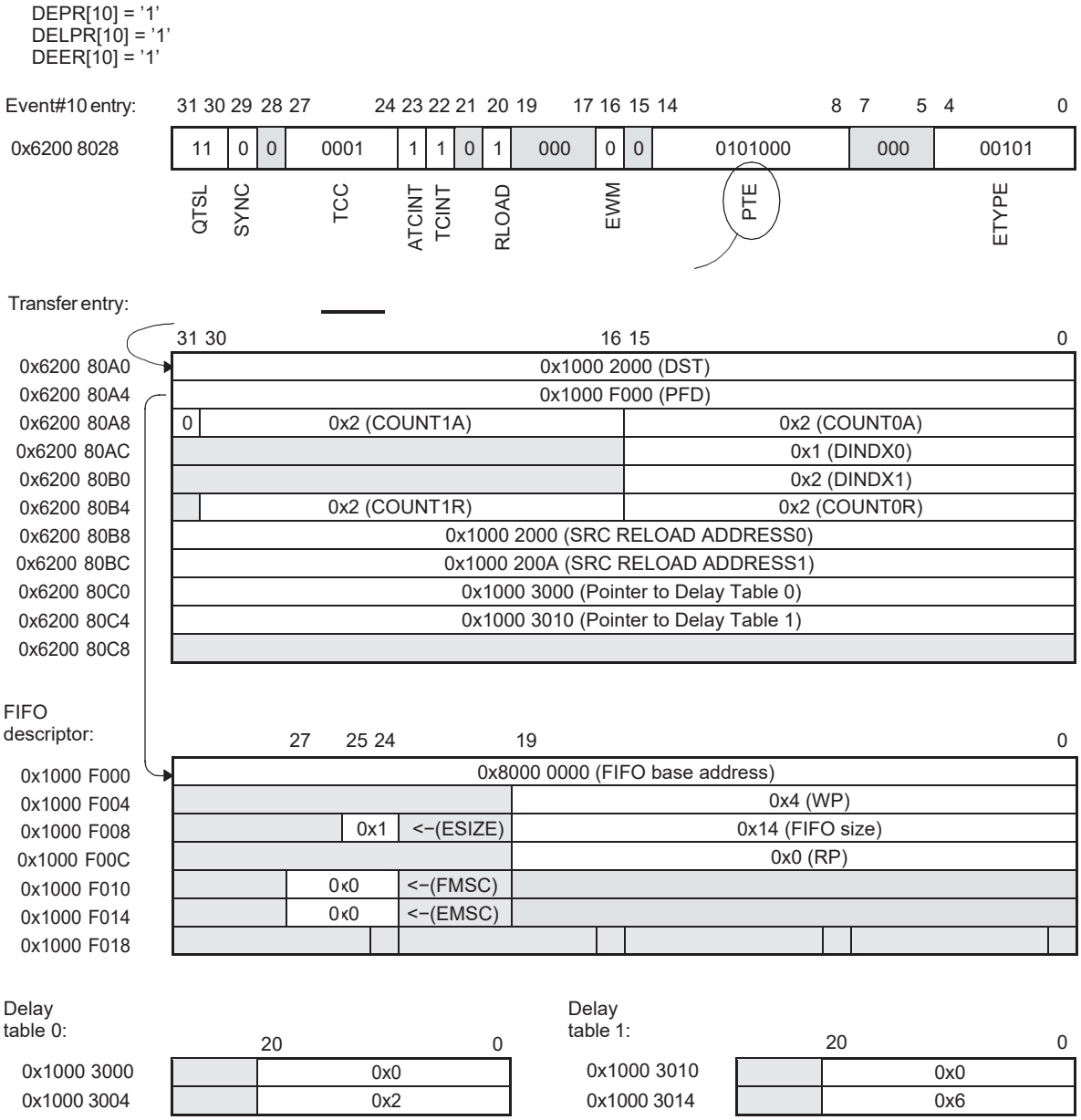
A memory snapshot for the data transfer after receiving the fourth and final synchronization event is shown in Figure 3-32. A delay-tap offset of 6 is used when reading this frame from the FIFO. This means that the first element in this frame is read from six slots behind the location of the RP. After transferring this frame, the RP is incremented by 2.

Figure 3-32. 2D FIFO Read Transfer Diagram (After Receiving All Synchronization Events)



The parameters for this transfer are shown in Figure 3-33. Event 10 is used to trigger this transfer. In this case the Event 10 is triggered by a CPU write '0,' followed by a CPU write '1' to bit one of the DETR (rising edge of the event signal ten triggers an event since DEPR[10] = '1'). The event is processed by the LoMAX since DELPR[10] = '1'.

Figure 3-33. Event Entry, Transfer Entry, FIFO Descriptor, and Delay Tables for 2D FIFO Read Transfer



In this example, dMAX will move two elements after receiving each synchronization event; a total of two synchronization events are required to complete the transfer. Since frame synchronization is enabled and ATCINT='1,' dMAX will notify the CPU after completion of each frame of the transfer, by triggering an interrupt and by setting bit 1 in the DTCR (TCC =0x1 in the event entry).

Although watermarks are disabled in the event entry, the FMSC and EMSC should still be programmed because they are also used for error reporting. FMSC and EMSC are set to the same value in this example to conserve status bits. If watermarks are used, these fields should instead be set to different values.

After the whole transfer is completed, dMAX will reload the active address and counters and wait for subsequent synchronization events. This example only shows one reload and the following two synchronization events. In practice, dMAX controller will continue to reload the destination address and counters at the completion of each subsequent transfer.

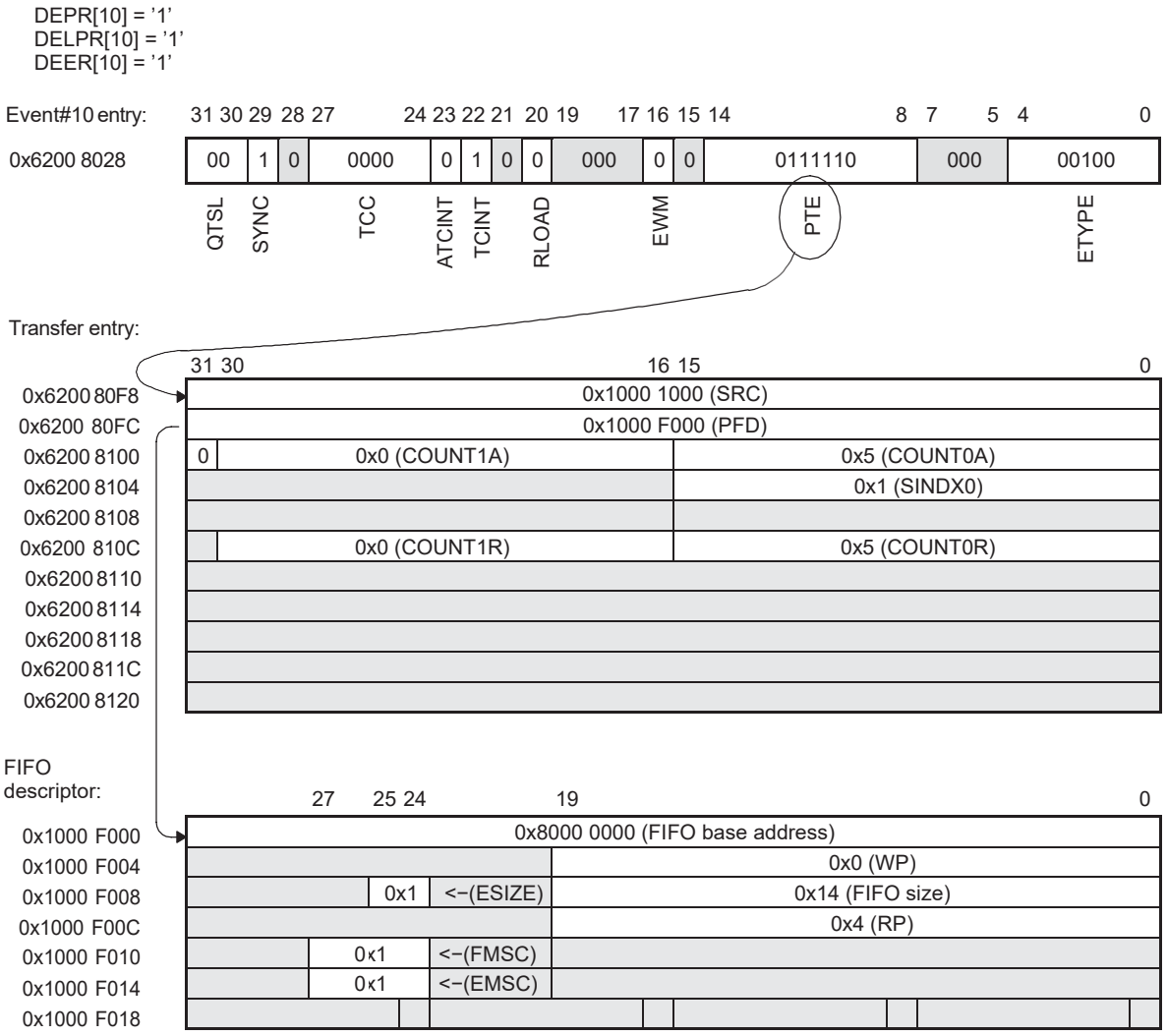
3.3.6 EXAMPLE: FIFO Overflow Error

A FIFO overflow error occurs when a FIFO write transfer attempts to transfer more elements to the FIFO than there are free slots to be written. When this occurs, the FIFO write transfer will be terminated prior to transferring any elements and dMAX will notify the CPU by generating an interrupt (INT7). In addition, both EMSC and FMSC bits will be set in the appropriate DFSR along with the EF2 bit within the FIFO descriptor.

In this example, a FIFO overflow error is demonstrated by attempting to copy a section of data from internal memory to a FIFO without enough free slots in external memory. The FIFO is programmed in the FIFO descriptor to have a size of 20 elements and an element size of 16 bits. In addition, the write pointer is set to 0 and the read pointer is set to 4 at the start of the transfer. This leaves four slots in the FIFO free to be written. The data block to be placed into the FIFO is five half-words and resides at address 0x10001000. The base address of the FIFO is set to 0x80000000.

The parameters for this transfer are shown in Figure 3-34. Event 10 is used by the transfer, and the event is triggered by a CPU writing a '0,' followed by a '1' to bit one of the DETR. A rising edge on event signal 10 triggers the event because DEPR[10] = '1'. The event is processed by the LoMAX because DELPR[10] = '1'.

Figure 3-34. Event Entry, Transfer Entry, and FIFO Descriptor for FIFO Overflow Error



Although watermarks are disabled in the event entry, the FMSC and EMSC should still be programmed because they are also used for error reporting. FMSC and EMSC are set to the same value in this example to conserve status bits. If watermarks are used, these fields should instead be set to different values.

The state of the FIFO before the transfer is shown in Figure 3-35, and the state of the FIFO after the transfer is shown in Figure 3-36.

Figure 3-35. FIFO Overflow Error Diagram (Before Receiving Synchronization Event)

Source address	Data
0x1000 1000	1
0x1000 1002	2
0x1000 1004	3
0x1000 1006	4
0x1000 1008	5

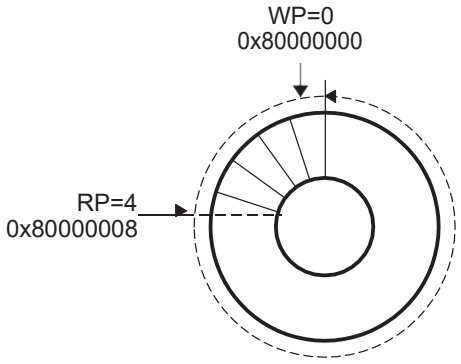
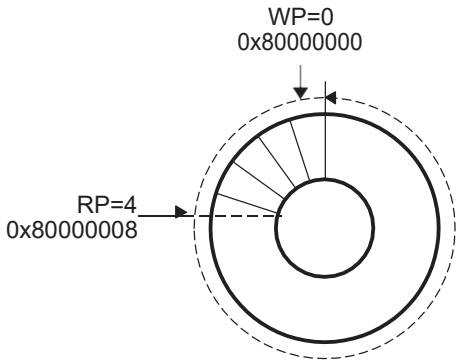


Figure 3-36. FIFO Overflow Error Diagram (After Receiving Synchronization Event)

Source address	Data
0x1000 1000	1
0x1000 1002	2
0x1000 1004	3
0x1000 1006	4
0x1000 1008	5



Because there are not enough free slots in the FIFO to perform the transfer, no elements are written to this FIFO and the read and write pointer remain unchanged. Instead, dMAX generates INT7 to the CPU, writes a 1 to the bits in the DFSR specified by the EMSC and FMSC fields, and sets the FIFO overflow error bit in the FIFO descriptor. It is important to note that even though there are enough elements in the FIFO to complete one quantum transfer (QTSL = 0, 4 elements), no elements are written because the entire transfer cannot be successfully completed. The state of the FIFO status registers before the transfer is shown in Figure 3-37, and their state after the transfer is shown in Figure 3-38.

Figure 3-37. dMAX FIFO Status Registers Before FIFO Overflow Error Occurs

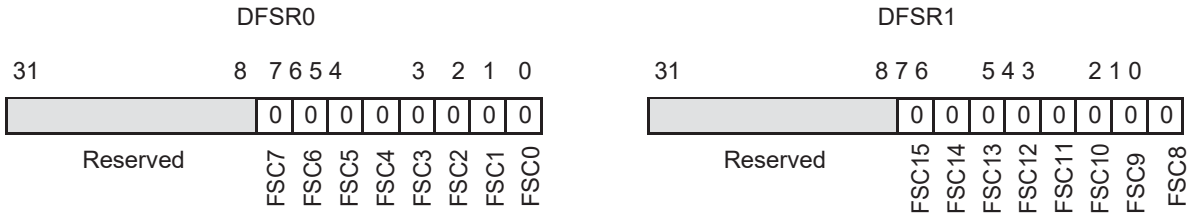
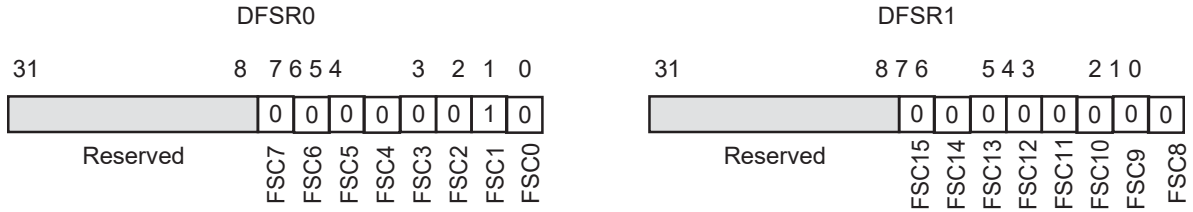


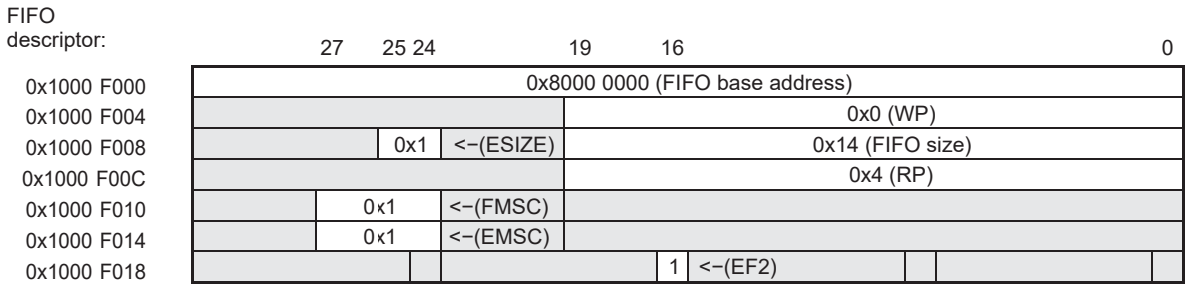
Figure 3-38. dMAX FIFO Status Registers After FIFO Overflow Error Occurs



Only the FSC1 bit is set because both EMSC and FMSC are equal to 1. After the error occurs, this bit should be manually cleared to 0 in order to receive error notifications on subsequent transfers.

The state of the FIFO descriptor after the FIFO overflow error occurs is shown in Figure 3-39.

Figure 3-39. FIFO Descriptor After FIFO Overflow Error Occurs



Note that the only field that has changed is EF2. It is set to 1 to indicate the FIFO overflow error has occurred. This bit and FIFO status bits should be cleared by the CPU after the error occurs.

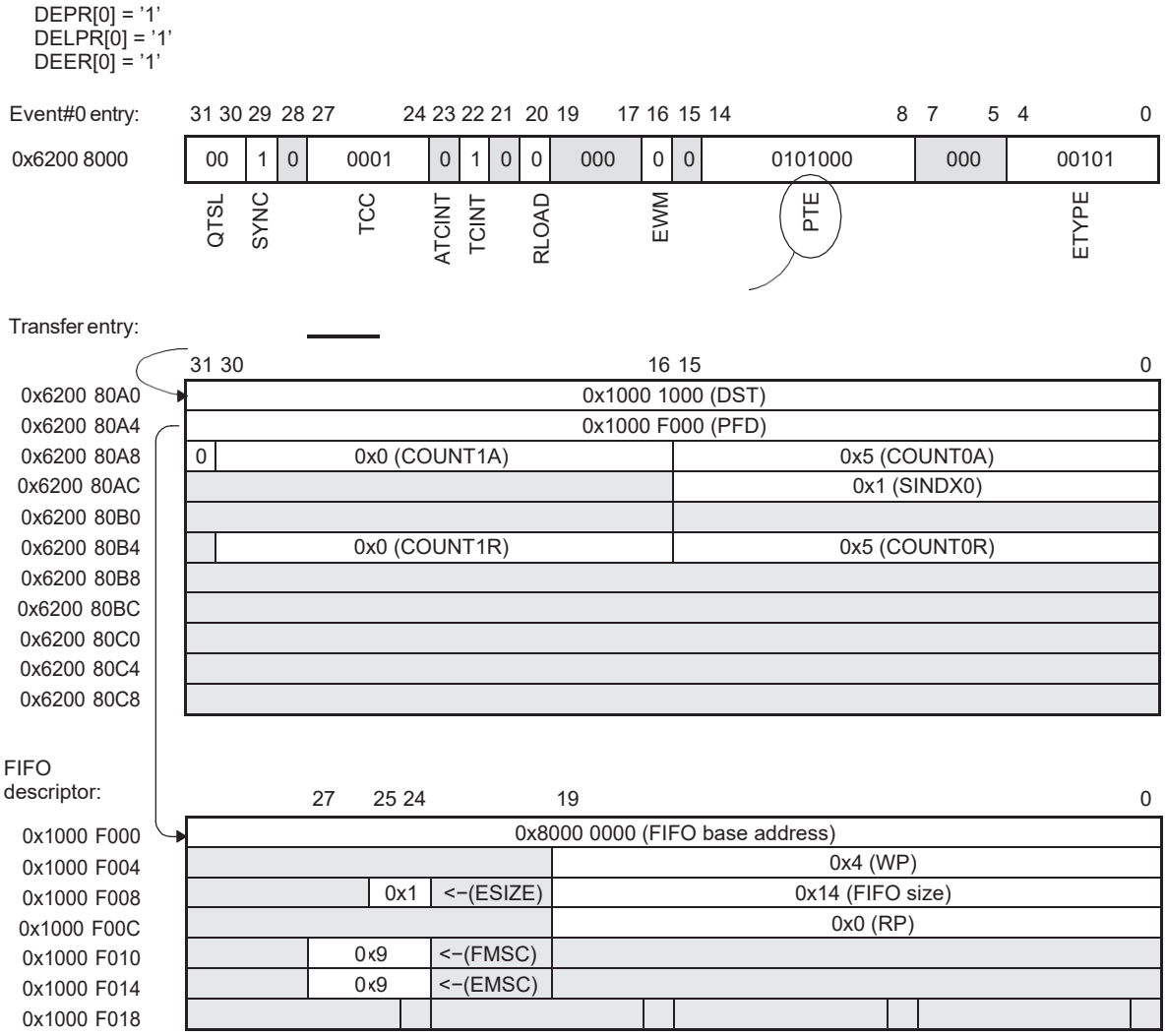
3.3.7 EXAMPLE: FIFO Underflow Error

A FIFO underflow error occurs when a FIFO read transfer attempts to read more elements from the FIFO than are available. When this occurs, the FIFO read transfer will be terminated prior to reading any elements and dMAX will notify the CPU by generating an interrupt (INT7). In addition, both EMSC and FMSC bits will be set in the appropriate DFSR along with the EF0 bit within the FIFO descriptor.

In this example, a FIFO underflow error is demonstrated. The FIFO is programmed in the FIFO descriptor to have a size of 20 elements and an element size of 16 bits. In addition, the write pointer is set to 4 and the read pointer is set to 0 at the start of the transfer. Therefore, there are only four elements available to read. The data block to be read from the FIFO is five half-words and will be placed at address 0x10001000. The base address of the FIFO is set to 0x80000000.

The parameters for this transfer are shown in Figure 3-40. Event 0 is used by the transfer, and the event is triggered by a CPU writing a '0,' followed by a '1' to bit 0 of the DETR. A rising edge on event signal 0 triggers the event because DEPR[0] = '1'. The event is processed by the LoMAX because DELPR[0] = '1'.

Figure 3-40. Event Entry, Transfer Entry, and FIFO Descriptor for FIFO Underflow Error



Although watermarks are disabled in the event entry, the FMSC and EMSC should still be programmed because they are also used for error reporting. FMSC and EMSC are set to the same value in this example to conserve status bits. If watermarks are used, these fields should instead be set to different values.

The state of the FIFO before the transfer is shown in Figure 3-41, and the state of the FIFO after the transfer is shown in Figure 3-42.

Figure 3-41. FIFO Underflow Error Diagram (Before Receiving Synchronization Event)

Source address	Data
0x1000 1000	
0x1000 1002	
0x1000 1004	
0x1000 1006	
0x1000 1008	

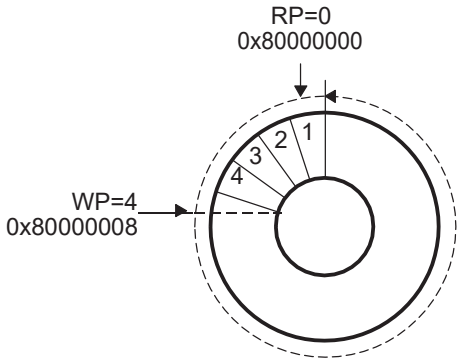
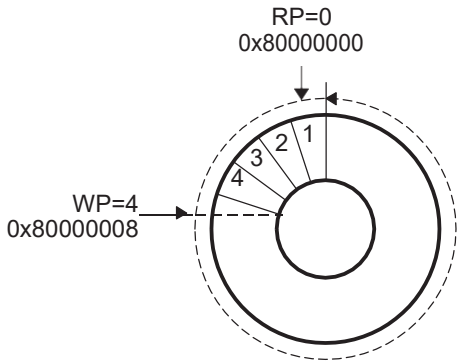


Figure 3-42. FIFO Underflow Error Diagram (After Receiving Synchronization Event)

Source address	Data
0x1000 1000	
0x1000 1002	
0x1000 1004	
0x1000 1006	
0x1000 1008	



Because there are not enough elements in the FIFO to perform the entire read transfer, no elements are read from this FIFO and the read pointer and write pointer remain unchanged. Instead, dMAX generates INT7 to the CPU, writes a 1 to the bits in the DFSR specified by the EMSC and FMSC fields, and sets the FIFO underflow error bit in the FIFO descriptor. It is important to note that even though there are enough elements in the FIFO to complete one quantum transfer (QTSL = 0, 4 elements), no elements are read because the entire transfer cannot be successfully completed. The state of the FIFO status registers before the transfer is shown in Figure 3-43, and their state after the transfer is shown in Figure 3-44.

Figure 3-43. dMAX FIFO Status Registers Before FIFO Underflow Error Occurs

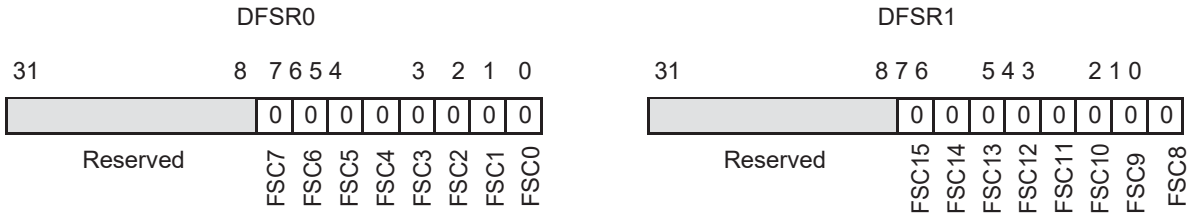
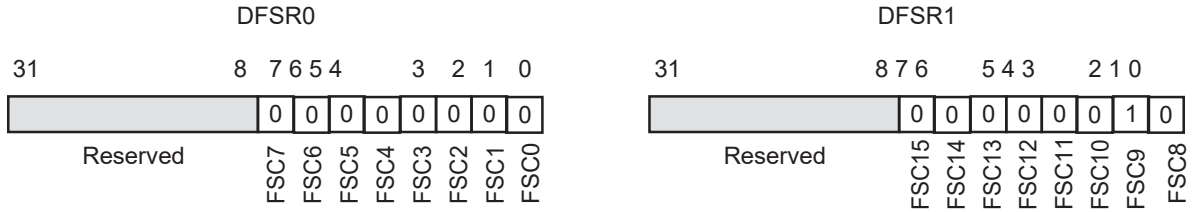


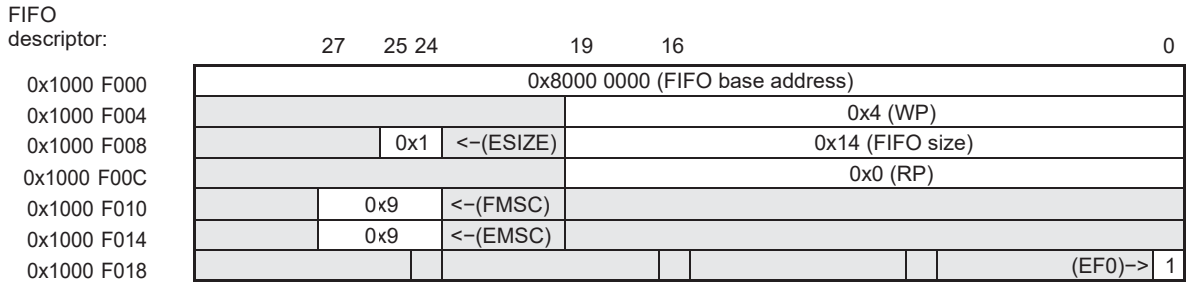
Figure 3-44. dMAX FIFO Status Registers After FIFO Underflow Error Occurs



Only the FSC9 bit is set because both EMSC and FMSC are equal to 9. After the error occurs, this bit should be manually cleared to 0 in order to receive error notifications on subsequent transfers.

The state of the FIFO descriptor after the FIFO overflow error occurs is shown in Figure 3-45.

Figure 3-45. FIFO Descriptor After FIFO Overflow Error Occurs



Note that the only field that has changed is EF0. It is set to 1 to indicate that the FIFO underflow error has occurred. This bit and the FIFO status bits should be manually cleared by the CPU.

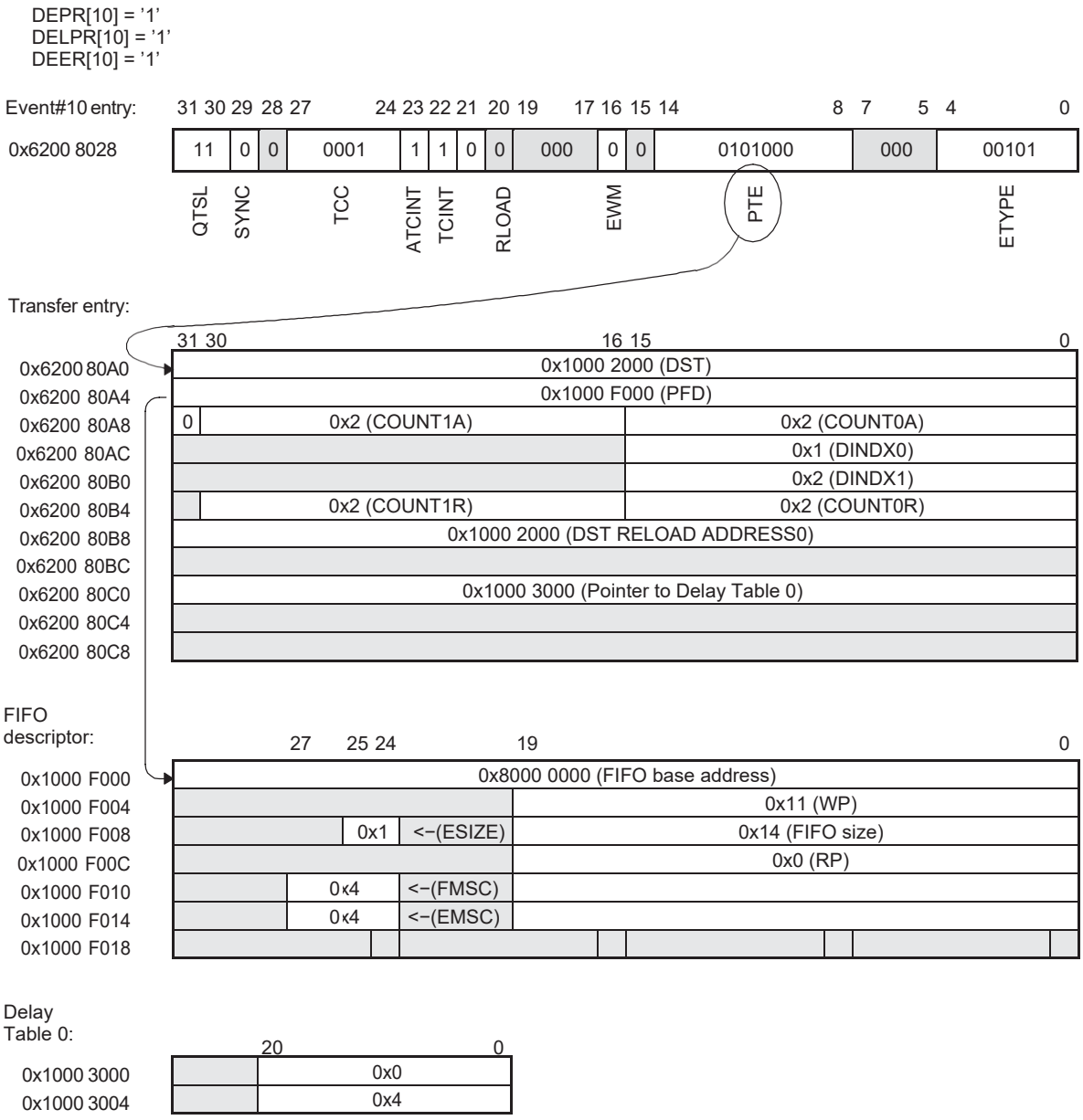
3.3.8 EXAMPLE: FIFO Delay-Tap Error

A FIFO delay-tap error occurs when a FIFO read transfer attempts to read from a delay-tap that is larger than the number of old samples stored in the FIFO. When this occurs, the FIFO read transfer will be terminated prior to reading any elements from that delay-tap and any subsequent delay-taps. The dMAX controller will then notify the CPU by generating an interrupt (INT7), setting the EMSC and FMSC bits in the appropriate DFSR, and setting the EF1 bit within the FIFO descriptor.

In this example, a FIFO delay-tap error is demonstrated by attempting to perform a 2D FIFO read transfer using a delay-table that contains a delay-tap larger than the number of old samples in the FIFO. The FIFO is programmed in the FIFO descriptor to have a size of 20 elements and an element size of 16 bits. In addition, the write pointer is set to 17 and the read pointer is set to 0 at the start of the transfer. Therefore, the largest valid delay-tap value is 4. The data block to be read from the FIFO is four half-words and will be placed at address 0x1000 2000. The base address of the FIFO is set to 0x80000000.

In the transfer, four data elements are to be copied from the FIFO into internal memory using a set of delay-tap offsets (0, 4). The four half-words are to be transferred to address 0x10002000, and the 20-element FIFO starts at address 0x8000000. A destination index of 1 is used for the first dimension of the transfer, while a destination index of 2 is used for the second dimension. The parameters for this transfer are shown in Figure 3-46. Event 10 is used to trigger this transfer. In this case, event 10 is triggered by a CPU write '0' followed by a CPU write '1' to bit one of the DETR register (the rising edge of event signal ten triggers an event since DEPR[10] = '1'). The event is processed by the LoMAX since DELPR[10] = '1'.

Figure 3-46. Event Entry, Transfer Entry, FIFO Descriptor, and Delay Tables for FIFO Delay-Tap Error



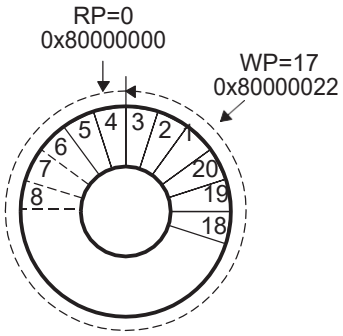
In this example, dMAX will attempt to move two elements after receiving each synchronization event, and a total of two synchronization events are required to complete the transfer. Since frame synchronization is enabled and ATCINT='1,' dMAX will notify the CPU after completion of each frame of the transfer by triggering an interrupt and by setting bit 1 in the DTCR (TCC =0x1 in the event entry).

The FMARK and EMARK do not need to be programmed because watermarks are disabled in the event entry. However, the FMSC and EMSC should still be programmed because they are also used for error reporting. FMSC and EMSC are set to the same value in this example to conserve status bits. If watermarks are used, these fields should instead be set to different values.

A memory snapshot of the data transfer before receiving the first synchronization event is shown in Figure 3-47.

Figure 3-47. FIFO Delay-tap Error Diagram (Before First Synchronization Event)

Source address	Data
0x1000 2000	
0x1000 2002	
0x1000 2004	
0x1000 2006	
0x1000 2008	

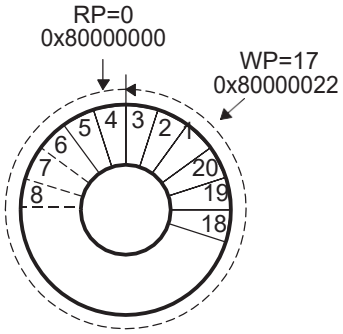


A memory snapshot of the data transfer after receiving the first synchronization event is shown in Figure 3-48. A delay-tap offset of 0 is used when reading this frame from the FIFO.

Figure 3-48. FIFO Delay-Tap Error Diagram (After Receiving the First Synchronization Event)

Source address	Data
0x1000 2000	4
0x1000 2002	5
0x1000 2004	
0x1000 2006	
0x1000 2008	

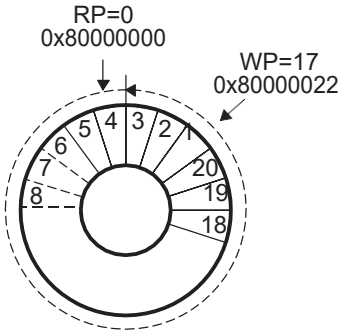
INDEX0=1



A memory snapshot for the data transfer after receiving the second synchronization event is shown in Figure 3-49. A delay-tap offset of 4 is used when attempting to read this frame from the FIFO. This means that the first element in the frame is attempted to be read from four slots behind the RP.

Figure 3-49. FIFO Delay-Tap Error Diagram (After Receiving the Second Synchronization Event)

Source address	Data
0x1000 2000	4
0x1000 2002	5
0x1000 2004	
0x1000 2006	
0x1000 2008	



Because there are not enough old samples in the FIFO to perform a read using a delay-tap of 4, no further elements are read from this FIFO and the read pointer and write pointer remain unchanged. Instead, dMAX generates INT7 to the CPU, writes a 1 to the bits in the DFSR specified by the EMSC and FMSC fields, and sets the FIFO delay-tap error bit (EF1) in the FIFO descriptor. The state of the FIFO status registers before the transfer is shown in Figure 3-50, and their state after the transfer is shown in Figure 3-51.

Figure 3-50. dMAX FIFO Status Registers Before FIFO Delay-Tap Error Occurs

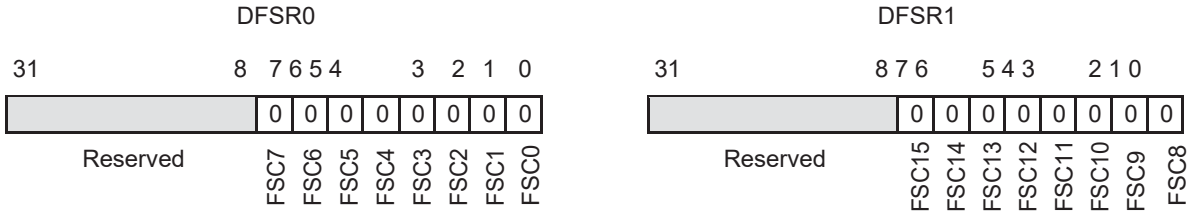
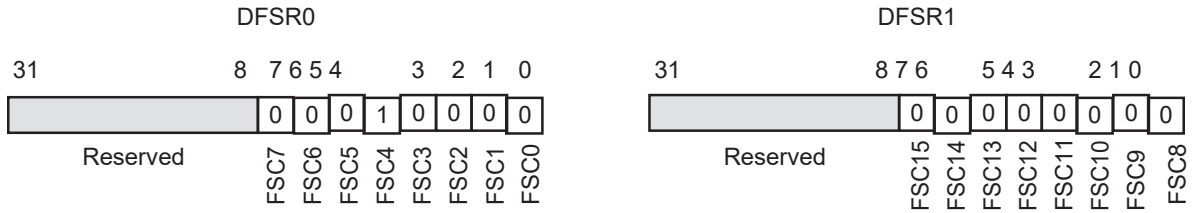


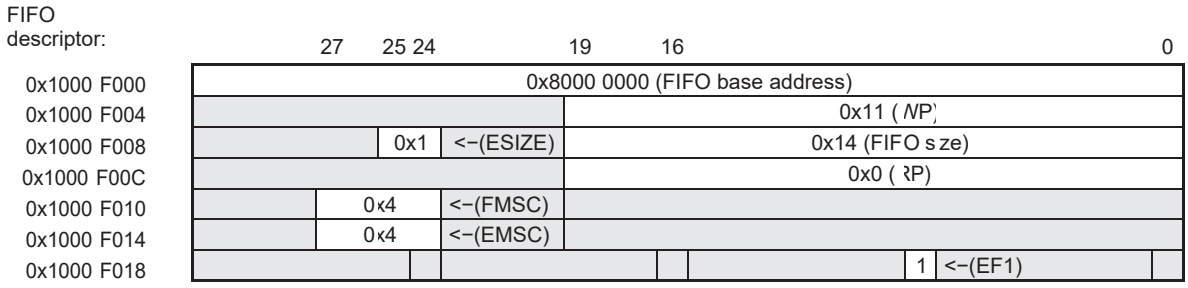
Figure 3-51. dMAX FIFO Status Registers After FIFO Delay-Tap Error Occurs



Only the FSC4 bit is set because both EMSC and FMSC are equal to 4. After the error occurs, this bit should be manually cleared to 0 in order to receive error notifications on subsequent transfers.

The state of the FIFO Descriptor after the FIFO delay-tap error occurs is shown in Figure 3-52.

Figure 3-52. FIFO Descriptor After FIFO Delay-Tap Error Occurs



Note that the only field that has changed is EF1. It is set to 1 to indicate that the FIFO delay-tap error has occurred. This bit and the FIFO status bits should be cleared by the CPU.

3.4 One-Dimensional Burst Transfers

The one-dimensional burst transfer (1DN) is optimized for doing fast transfer of sequential data from one memory location to the other. The 1DN transfer happens in bursts. The burst length (BURSTLEN) can be configured in the transfer entry to be between 1-64 bytes. The number of bursts (NBURSTS) can be configured in the transfer entry and can be between 1-16 bursts. The 1DN transfer does not yield to a higher priority pending transfer after transferring one burst. The 1DN transfer only yields after a burst if there is a new event that has arrived. The 1DN transfer yields to a higher priority pending transfer after performing NBURSTS.

3.4.1 Steps Required to Set Up a One-Dimensional Burst Transfer

The following steps are required to set up a One-Dimensional burst dMAX transfer:

1. Priority of an event that will be used to trigger the general-purpose data transfer must be defined. To put the event into the high-priority group, a bit corresponding to the event in the DEHPR should be set to one. To put the event into the low-priority group, a bit corresponding to the event in the DELPR should be set to one.
2. The event signal edge (rising/falling) that will be used to trigger an event must be defined. To trigger an event on the rising edge of the event signal, a bit corresponding to the event in the DEPR should be set to one. To trigger an event on the falling edge of the event signal, a bit corresponding to the event in the DEPR should be cleared to zero.
3. If the event is sorted to the high-priority group, its event entry in the HiMAX PaRAM must be defined. If the event is sorted to the low-priority group, its event entry in the LoMAX PaRAM must be defined. The following bit fields in the event entry must be configured:
   a. ETYPE bit field must be set to '00110' for 1DN data transfer.
   b. PTE bit field is used as a pointer to a location in the PaRAM where a transfer entry that corresponds to the event is stored.
4. Transfer entry must be properly configured. The PTE bit field of the event entry points to the transfer entry. The following fields in the transfer entry must be configured.
   a. When a transfer is complete, the TCINT bit should be set if a notification to the CPU is required. The TCC indicates which bit in the DTCR is going to be set to indicate transfer status to the CPU.
   b. NBURST field must be set to specify the number of bursts after which the 1DN transfer will yield to any higher priority pending transfer.
   c. BURSTLEN should be set to specify the number of bytes transferred for each burst. Please note the unit for BURSTLEN field is bytes and not elements.
   d. SRC (Source address pointer), DST (Destination address pointer) and CNT (Transfer length in number of bytes) fields should be specified to describe the transfer. Please note the unit for CNT is number of bytes and not number of elements.
   e. EVNT field must be set to indicate the event number of the transfer. The valid values are 0-31.
5. The event must be enabled by setting a corresponding bit in the DEER.

Once an event is enabled, dMAX will perform a data transfer after an appropriate transition is detected on the event signal. If TCINT='1', dMAX will signal transfer status to the CPU by using an interrupt (INT8), and by setting a bit, specified by the TCC bit field of the transfer entry, in the DTCR. The DTCR bits are mirrored in the DESR. The read-only DESR is located inside the CPU module and the CPU can access the register with minimum overhead (the fastest way of monitoring the DTCR is to read its copy from the DESR).

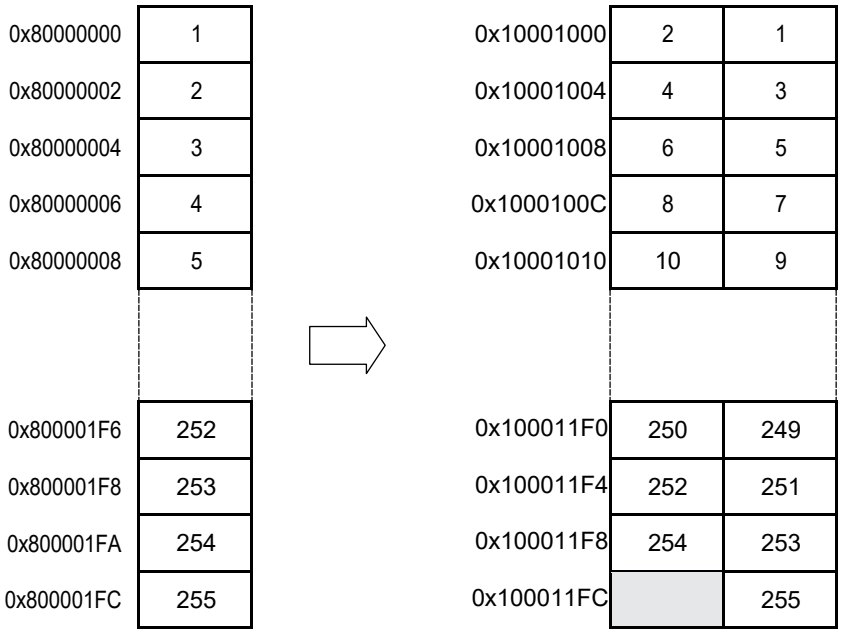
To keep receiving the notifications from a particular dMAX channel, after each notification, the CPU must clear the transfer completion bit used by the dMAX channel. The CPU should use the read-only DESR to find out which transfer completion bits were set, and it should clear them by writing '1' to the corresponding bits from the DTCR.

After enabling the event, it is not recommended to access the event entry and the transfer entry. Please note the One-Dimensional burst transfer is frame synchronized transfer in essence and the entire transfer is completed after receiving the event signal.

3.4.2 Example: One-Dimensional Burst Transfer

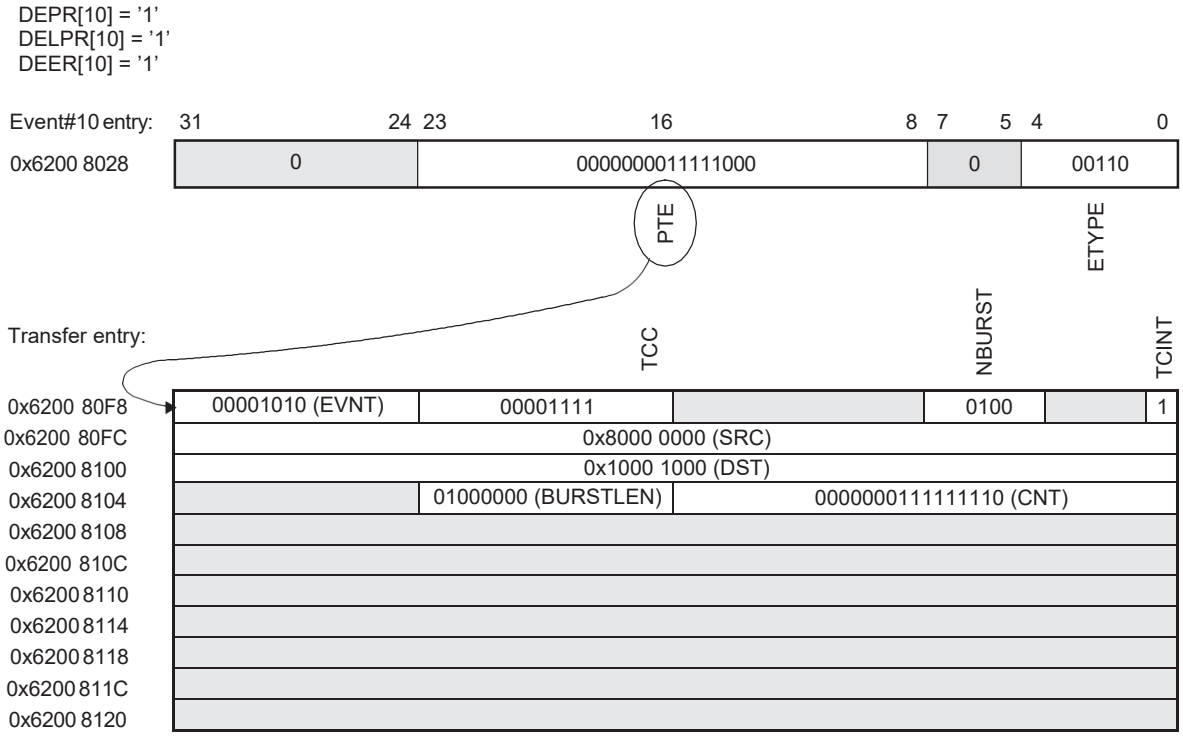
Often during device operation it is necessary to transfer a block of data from one location to another, usually between on- and off-chip memories. The most efficient transfer that can be performed by dMAX to perform a block move is the 1DN transfer. In this example, a section of data is to be copied from external memory to internal memory. The data block is 255 half-words and resides at address 0x80000000. It is to be transferred to internal address 0x10001000. The data transfer is shown in Figure 3-53.

Figure 3-53. 1DN Block Move Diagram



The parameters for this transfer are shown in Figure 3-54. Event 10 is used to trigger this transfer. Here, Event 10 is triggered by CPU write '0', followed by CPU write '1', to bit one of the DETR register (rising edge of the event signal 10 triggers an event since DEPR[10] = '1'). The event is processed by the LoMAX since DELPR[10] = '1'.

Figure 3-54. Event Entry and Transfer Entry for 1DN Transfer



The whole transfer is completed after receiving one synchronization event. Subsequent synchronization events will be ignored. After completing the transfer, dMAX will notify the CPU by triggering an interrupt (INT8) and by setting bit 7 in the DTCR1 (TCC bit field within the event entry is equal to 15).

The dMAX controller will split the transfer into burst transfers. The NBURST setting is 4 (5 bursts) and the BURSTLEN is 64. First 320bytes will be transferred as 5 bursts of 64bytes. After every burst, the dMAX will check if a new event has arrived. If a new event has arrived, the dMAX will yield to perform the transfer for the new event. If no new event arrives, the dMAX will check if there is any higher priority pending transfer after the initial 5 bursts. If a higher priority transfer is pending, the dMAX will yield to perform the higher priority transfer. If no higher priority transfer is pending, the dMAX will continue performing the 1DN transfer and will transfer the remaining 190bytes as 2 bursts of 64bytes and one burst of 62bytes.

3.5 SPI Slave Transfer

SPI slave transfer allows for servicing the SPI peripheral when used in slave mode. The peripheral servicing requires that for each input event, one element is read from the SPI input shift register (SPIBUF) and is stored in the destination address. Also, one element is read from the input address and moved to the SPI output shift register (SPIDAT0).

3.5.1 Steps Required to Set Up a SPI Slave Transfer

The following steps are required to set up a SPI Slave dMAX transfer:

1. Priority of an event that will be used to trigger the SPI slave data transfer must be defined. To put the event into the high-priority group, a bit corresponding to the event in the DEHPR should be set to one. To put the event into the low-priority group, a bit corresponding to the event in the DELPR should be set to one.
2. The event signal edge (rising/falling) that will be used to trigger an event must be defined. To trigger an event on the rising edge of the event signal, a bit corresponding to the event in the DEPR should be set to one. To trigger an event on the falling edge of the event signal, a bit corresponding to the event in the DEPR should be cleared to zero.
3. If the event is sorted to the high-priority group, its event entry in the HiMAX PaRAM must be defined. If the event is sorted to the low-priority group, its event entry in the LoMAX PaRAM must be defined. The following bit fields in the event entry must be configured:
   a. ETYPE bit field must be set to '00010' for SPI slave data transfer.
   b. If the RLOAD bit is set, when a transfer is complete, an active counter and an active address register set will be reloaded from one of two sets of address reference registers. If the RLOAD bit is cleared, dMAX will ignore new events after a transfer is complete.
   c. ESIZE defines the element size for the transfer. Element size can be 8-bit element or 16-bit element.
   d. PTE bit field is used as a pointer to a location in the PaRAM where a transfer entry that corresponds to the event is stored.
   e. The TCC indicates which bit in the DTCR is going to be set to indicate transfer status to the CPU.
   f. When a transfer is complete, the TCINT bit should be set if a notification to the CPU is required.
   g. The SPI field indicates which SPI peripheral to use for the data transfer.
4. Transfer entry must be properly configured (transfer must be defined by source/destination address, and count bit fields). The PTE bit field of the event entry points to the transfer entry. When the event entry reload is enabled (RLOAD='1'), the PP bit in the transfer entry must be properly configured. When PP='0', after the transfer is completed, reload register set one is loaded in the set of active registers. When PP='1', after the transfer is completed, reload register set zero is loaded in the set of active registers.
5. The event must be enabled by setting a corresponding bit in the DEER.

Once an event is enabled, dMAX will perform a data transfer after an appropriate transition is detected on the event signal. If TCINT='1', dMAX will signal transfer status to the CPU by using an interrupt (INT8), and by setting a bit, specified by the TCC bit field of the event entry, in the DTCR. The DTCR bits are mirrored in the DESR. The read-only DESR is located inside the CPU module and the CPU can access the register with minimum overhead (the fastest way of monitoring the DTCR is to read its copy from the DESR).

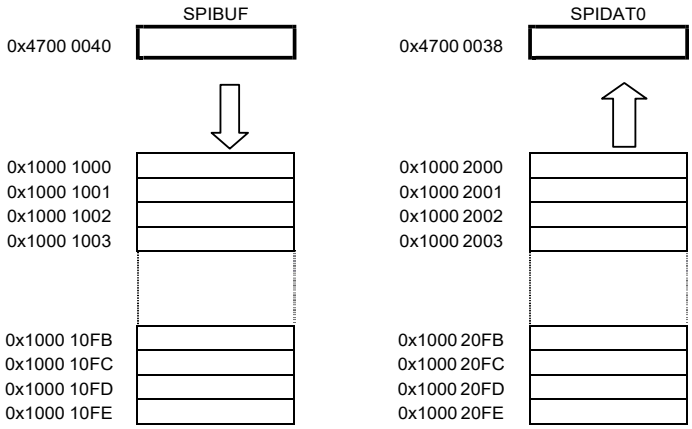
To keep receiving the notifications from a particular dMAX channel, after each notification, the CPU must clear the transfer completion bit used by the dMAX channel. The CPU should use the read-only DESR to find out which transfer completion bits were set, and it should clear them by writing '1' to the corresponding bits from the DTCR.

After enabling the event, it is not recommended to access the event entry, and the active register set in the transfer entry.

3.5.2 Example: SPI Slave Transfer

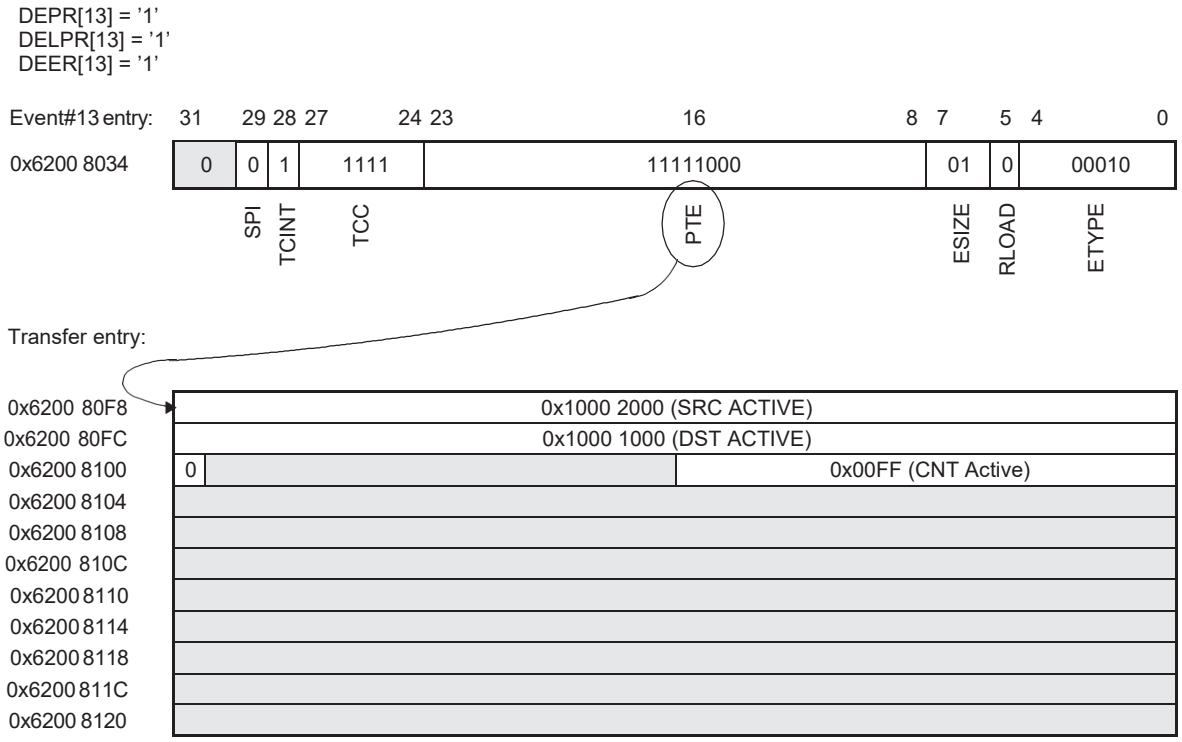
In this example, SPI0 is configured as the SPI slave. The SPI is configured to do 8bit element transfers. A section of data memory at 0x10001000 is updated with the data being received from the SPI master. A section of data memory at 0x10002000 contains the data that needs to be transmitted back to the SPI master. The data block is 255 8-bit-words. The data transfer is shown in Figure 3-55.

Figure 3-55. SPI Slave Transfer Diagram



The parameters for this transfer are shown in Figure 3-56. Event 13 is the dMAX event generated by the SPI0 peripheral. Event entry 13 is used for the SPI slave transfer to support SPI0 peripheral.

Figure 3-56. Event Entry and Transfer Entry for SPI Slave Transfer



After receiving one synchronization event, the dMAX will read one element from the SPIBUF register and move it to the destination address. Also, one element will be read from the source address and written to the SPIDAT0 register. After the transfer, the source and the destination address are incremented by 1 and the active count is decremented by 1. The transfer is complete when the active count becomes 0. After completing the transfer, dMAX will notify the CPU by triggering an interrupt (INT8) and by setting bit 7 in the DTCR1 (TCC bit field within the event entry is equal to 15). As the RLOAD is not enabled, any subsequent synchronization events will be ignored.

3.6 Examples of Servicing Peripherals

An important capability of the dMAX controller is its ability to service peripherals in the background of the CPU operation, without requiring any CPU intervention. Through proper initialization of the dMAX channels, they can be configured to continuously service on- and off-chip peripherals throughout the device operation. Each event available to dMAX has its own dedicated channel, and all channels operate simultaneously. This means that all data streams can be handled independently with little or no consideration for what else is going on in dMAX.

Since all dMAX channels are always synchronized, there are no special setups required to configure a channel to properly service a particular event. The only requirements are to use the proper channel and configure event entry and transfer entry for a particular transfer and to enable the channel's event in the DEER.

When programming a dMAX channel to service a peripheral, it is necessary to know how data is to be presented to the DSP. Data is always provided with some kind of synchronization event, and is either one element per event (non-bursting), or multiple elements per event (bursting).

3.6.1 EXAMPLE: Servicing McASP Peripheral

Higher bandwidth applications require that multiple data elements be presented to the DSP for every sync event. This frame of data can either be from multiple sources that are working simultaneously or a single high-throughput peripheral that streams data to/from the DSP.

In this example, a McASP receives samples from three stereo audio channels and presents them to the DSP. Each sample is represented by a 24-bit element, so a whole 32-bit word is read from McASP (element size is 32 bits). On each event the McASP receives either three samples from left or three samples from right channel. On subsequent events, samples from left and from right channels alternate. For example, on every odd event, samples from the left channels are received, while on every even event samples from the right channels are received.

CAUTION

Only 32-bit transfers are supported to and from the McASP DMA ports.

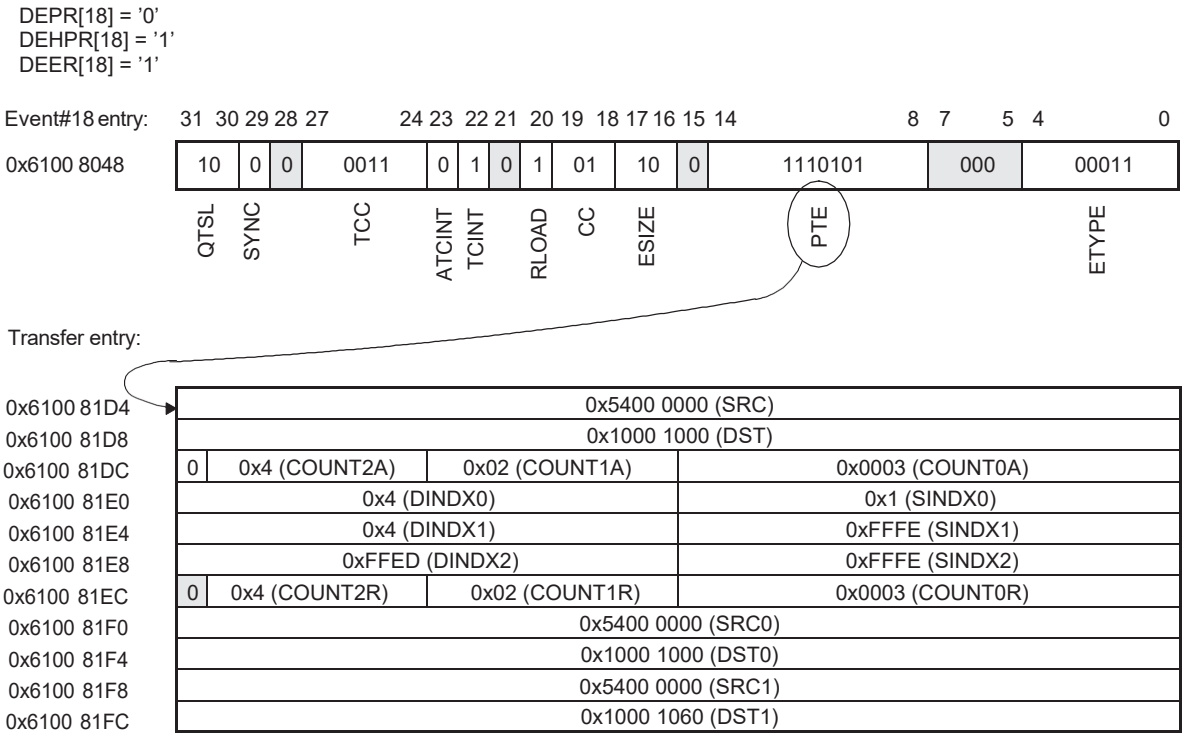
The dMAX controller must sort the McASP data for processing by the CPU. The data is sorted into two separate buffers (left and right buffer) per each stereo channel.

The McASP has a FIFO style programmer's model in that each read from the McASP DMA port results in reading each successive serializer programmed as a receiver, and each write to the McASP port results in a write to each successive serializer programmed as a transmitter, regardless of the address used. The McASP includes the logic to automatically cycle through the serializers after each DMA request (skipping over any serializers that are either disabled or programmed the 'opposite' way). It also includes logic to make sure that there are exactly as many reads per receive DMA event as there are serializers programmed to receive, and exactly as many writes per transmit DMA event as there are serializers programmed to transmit.

This example utilizes the ping-pong buffering scheme. This way, dMAX can keep fetching a next block of data, while the current block is being processed by the CPU.

The event entry and transfer entry for configuration is shown in Figure 3-57.

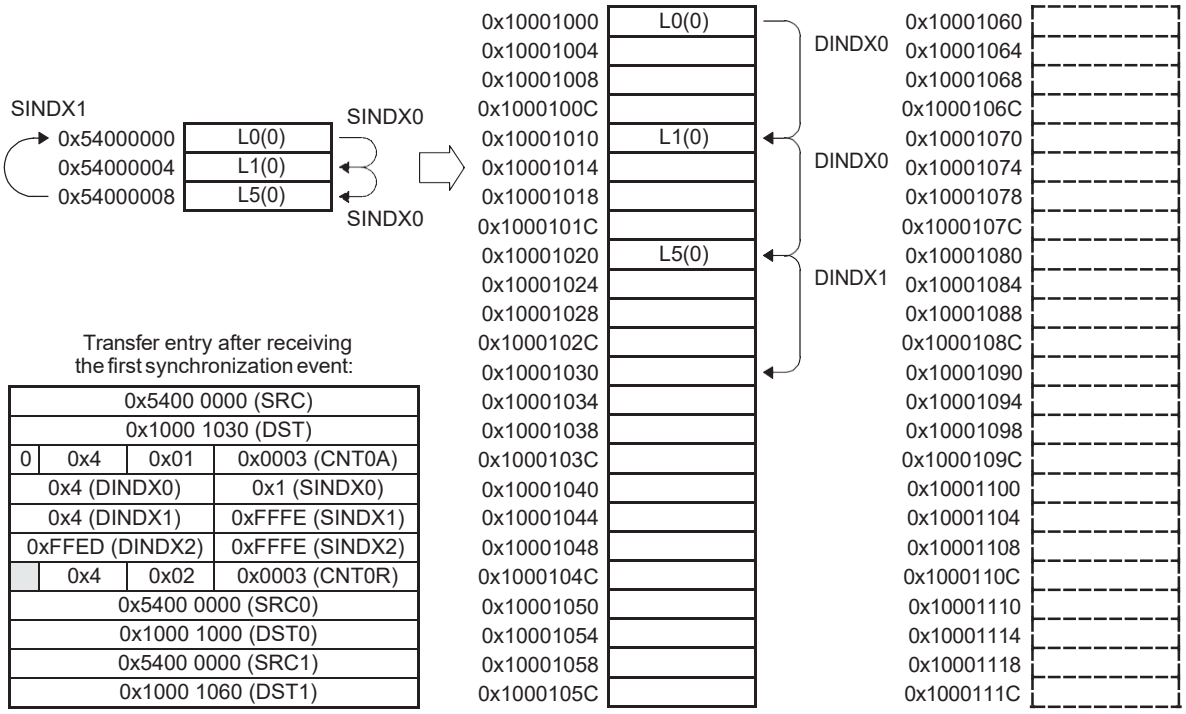
Figure 3-57. Event Entry and Transfer Entry for McASP Transfer



The McASP0 is configured to receive samples from three stereo channels. Here, McASP0 channels zero, one, and five are used.

After receiving the first three left samples from the three channels (L0(0), L1(0) and L5(0)) the McASP triggers an event to dMAX, which then moves the three samples from McASP and sorts them to the left ping buffers of each of the three channels. A memory snapshot after receiving the first event is shown in Figure 3-58.

Figure 3-58. McASP Receive Example After Receiving the First Synchronization Event

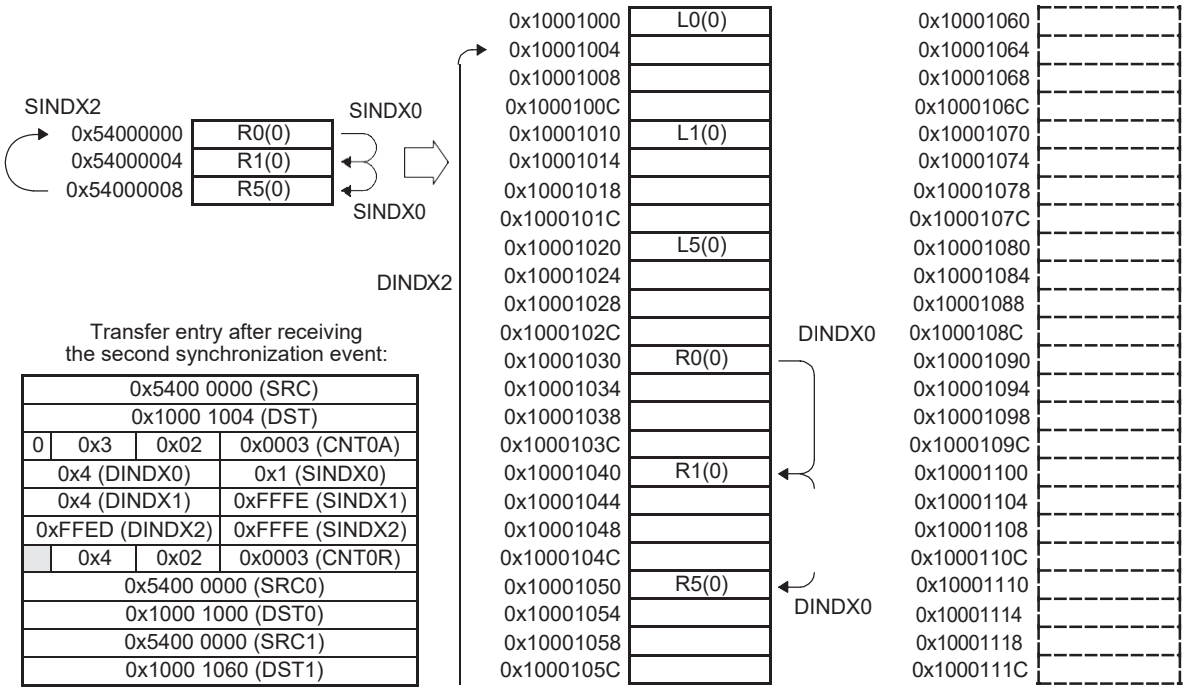


Transfer entry after receiving the first synchronization event:

0x5400 0000 (SRC)			
0x1000 1030 (DST)			
0	0x4	0x01	0x0003 (CNT0A)
0x4 (DINDX0)			0x1 (SINDX0)
0x4 (DINDX1)			0xFFFE (SINDX1)
0xFFED (DINDX2)			0xFFFE (SINDX2)
	0x4	0x02	0x0003 (CNT0R)
0x5400 0000 (SRC0)			
0x1000 1000 (DST0)			
0x5400 0000 (SRC1)			
0x1000 1060 (DST1)			

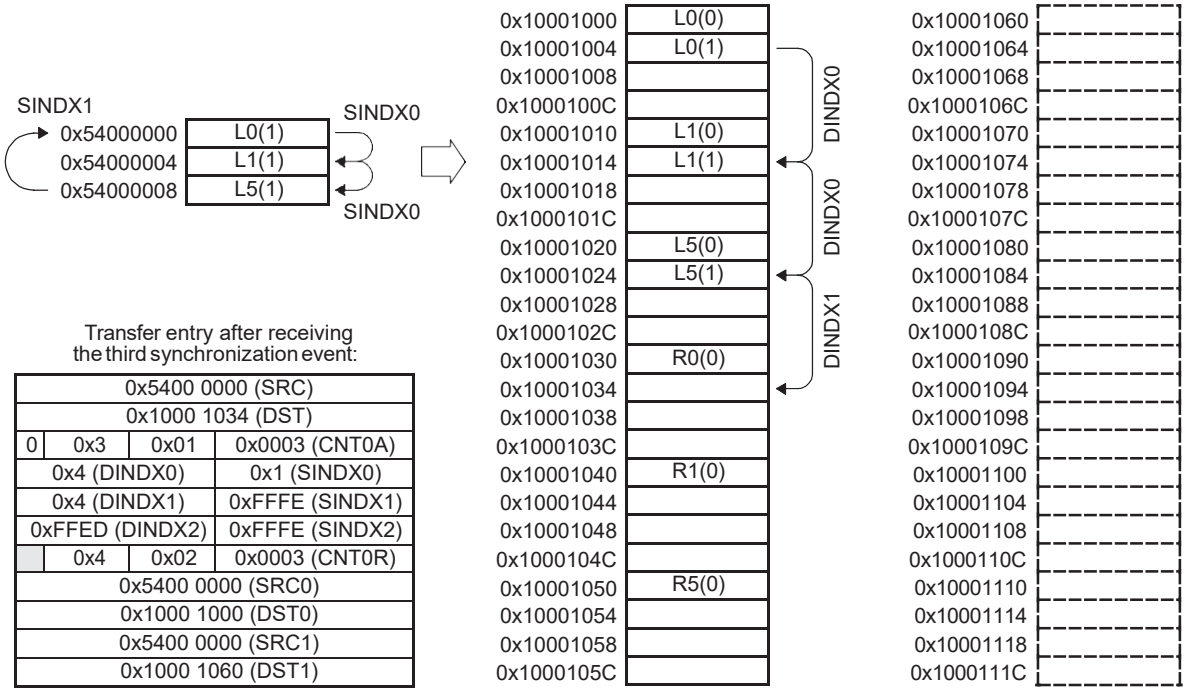
After receiving the first three right samples from the three channels (R0(0), R1(0) and R5(0)) the McASP triggers a second event to dMAX, which then transfers and sorts the three samples to the right ping buffers of each of the three channels. A memory snapshot after receiving the second event is shown in Figure 3-59.

Figure 3-59. McASP Receive Example After Receiving the Second Synchronization Event



After receiving the next three left samples from the three channels (L0(1), L1(1) and L5(1)) the McASP will trigger a third event to dMAX, which then transfersand sorts the three samples to the left ping buffers of each of the three channels. A memory snapshot after receiving the second event is shown in Figure 3-60.

Figure 3-60. McASP Receive Example After Receiving the Third Synchronization Event

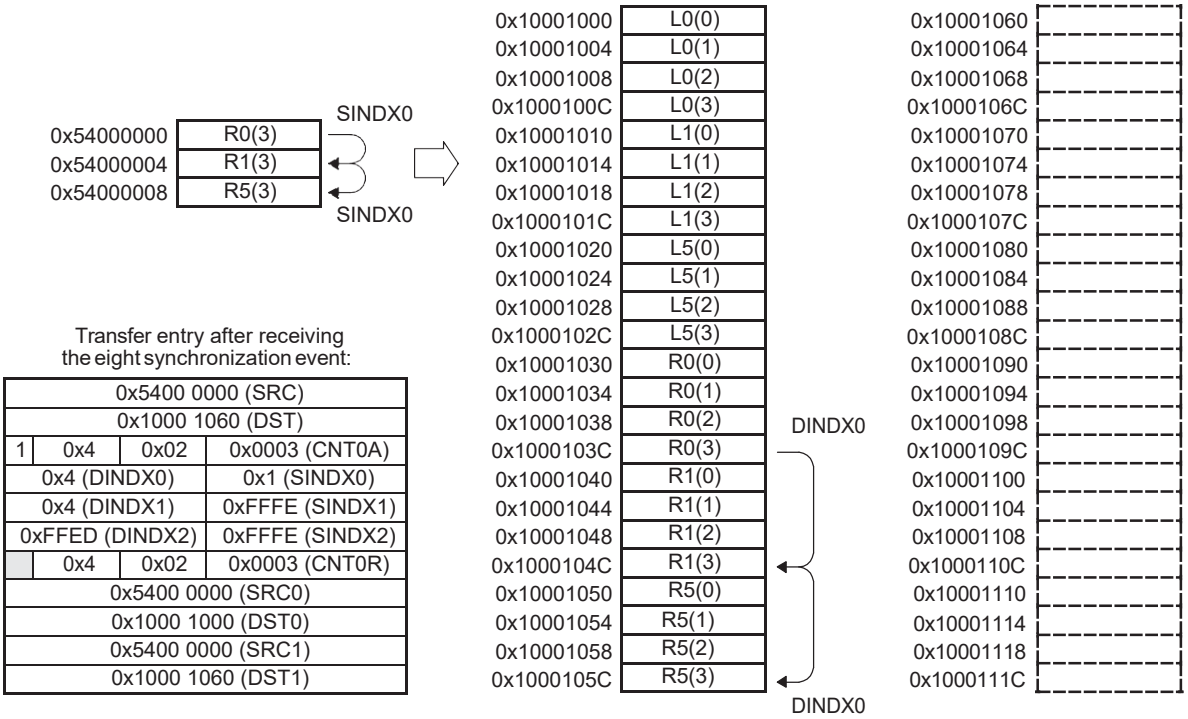


Transfer entry after receiving the third synchronization event:

0x5400 0000 (SRC)			
0x1000 1034 (DST)			
0	0x3	0x01	0x0003 (CNT0A)
0x4 (DINDX0)			0x1 (SINDX0)
0x4 (DINDX1)			0xFFFE (SINDX1)
0xFFED (DINDX2)			0xFFFE (SINDX2)
	0x4	0x02	0x0003 (CNT0R)
0x5400 0000 (SRC0)			
0x1000 1000 (DST0)			
0x5400 0000 (SRC1)			
0x1000 1060 (DST1)			

After receiving eight synchronization events, the McASP will receive four samples from both the left and right channels in each block. A memory snapshot after receiving eight synchronization events is shown in Figure 3-61.

Figure 3-61. McASP Receive Example After Receiving the Eight Synchronization Events



Transfer entry after receiving the eight synchronization event:

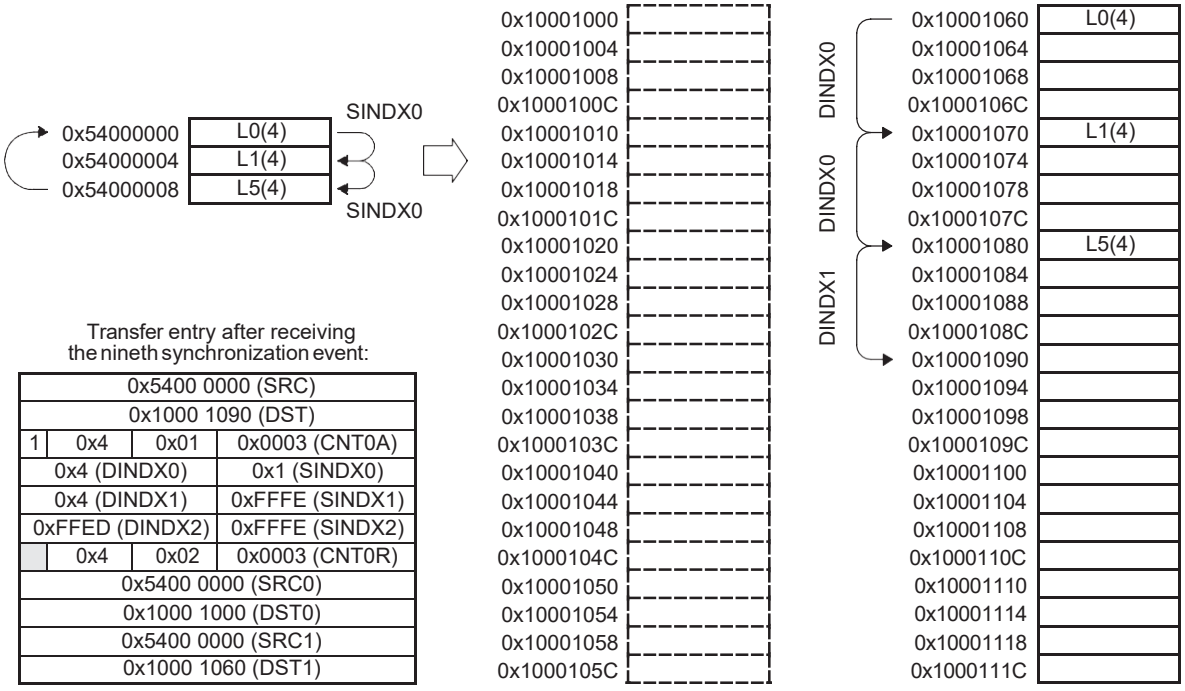
0x5400 0000 (SRC)			
0x1000 1060 (DST)			
1	0x4	0x02	0x0003 (CNT0A)
0x4 (DINDX0)			0x1 (SINDX0)
0x4 (DINDX1)			0xFFFE (SINDX1)
0xFFED (DINDX2)			0xFFFE (SINDX2)
	0x4	0x02	0x0003 (CNT0R)
0x5400 0000 (SRC0)			
0x1000 1000 (DST0)			
0x5400 0000 (SRC1)			
0x1000 1060 (DST1)			

After receiving eight synchronization events all active counters will expire, the transfer is complete, and dMAX sets the bit three in the DTCR0 (TCC is equal to three in the event entry), and it triggers a CPU interrupt. Now, the CPU can start processing block of data in the ping buffers.

Since reload is enabled (RLOAD set to one in the event entry), dMAX will load the active counter from the reference counter, and it will load set of active registers from reload register set one (since the PP bit was cleared before enabling the event). The dMAX controller will also flip the PP bit to one during reload.

After receiving three left samples L0(4), L1(4) and L5(4), dMAX generates Event 9. A memory snapshot after receiving this event is shown in Figure 3-62. The following seven synchronization events will be used to fill the pong buffers.

Figure 3-62. McASP Receive Example After Receiving the Nine Synchronization Event



Transfer entry after receiving the nineth synchronization event:

0x5400 0000 (SRC)			
0x1000 1090 (DST)			
1	0x4	0x01	0x0003 (CNT0A)
0x4 (DINDX0)			0x1 (SINDX0)
0x4 (DINDX1)			0xFFFE (SINDX1)
0xFFED (DINDX2)			0xFFFE (SINDX2)
	0x4	0x02	0x0003 (CNT0R)
0x5400 0000 (SRC0)			
0x1000 1000 (DST0)			
0x5400 0000 (SRC1)			
0x1000 1060 (DST1)			

Once pong buffers are full and the counters of all three dimensions are expired, dMAX sets bit 3 in the DTCR0 (since the TCC code specified in the event entry is equal to three), and it triggers a CPU interrupt (INT8). Since reload is enabled (RLOAD bit is set to one in the event entry), dMAX reloads active counter values from the counter reference register.

Since the PP bit is set, dMAX loads active source and destination registers from reload register set zero (SRC0 and DST0 will be loaded in the active SRC and DST registers, respectively). The dMAX controller clears the PP bit to zero after reload is performed.

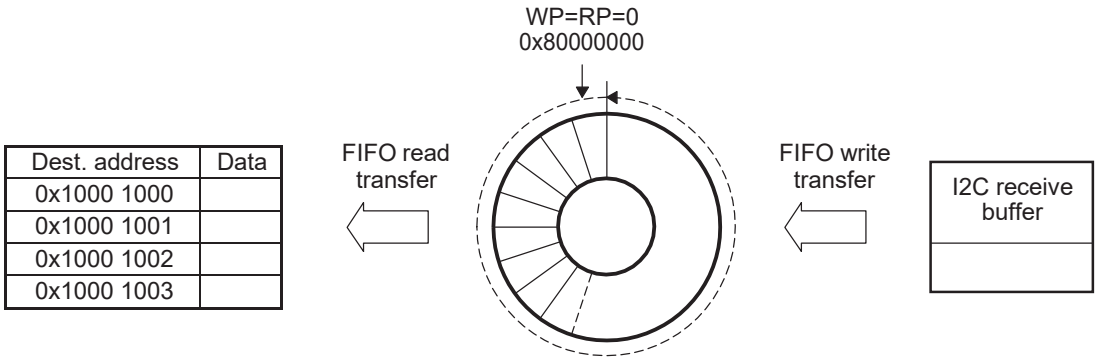
3.6.2 EXAMPLE: Servicing I2C Peripherals (FIFO FMARK Watermark)

The FIFO FMARK watermark can be used to detect when the number of samples in a FIFO reaches or exceeds a programmed value. At the completion of a FIFO write transfer in which EWM = 1 in the event entry, dMAX compares the read and write pointers to determine if the number of samples in the FIFO is greater-than or equal-to the value programmed into the FMARK field. If so, dMAX sets a specified bit in one of the DFSRs and generates an interrupt (INT7) to the CPU. The CPU can then trigger a FIFO read transfer to drain the samples from the FIFO.

This example demonstrates using a FIFO write transfer to copy data from an I2C peripheral that has been set up to generate an event to dMAX on each element that it receives. An FMARK watermark is used by the FIFO write transfer to notify the CPU when a sufficient number of samples have been placed in the FIFO. In processing the FMARK interrupt, the CPU manually triggers a FIFO read transfer to drain a block of data from the FIFO. At the completion of the FIFO read transfer, the CPU is again notified that the block of data has been read from the FIFO and is ready for processing.

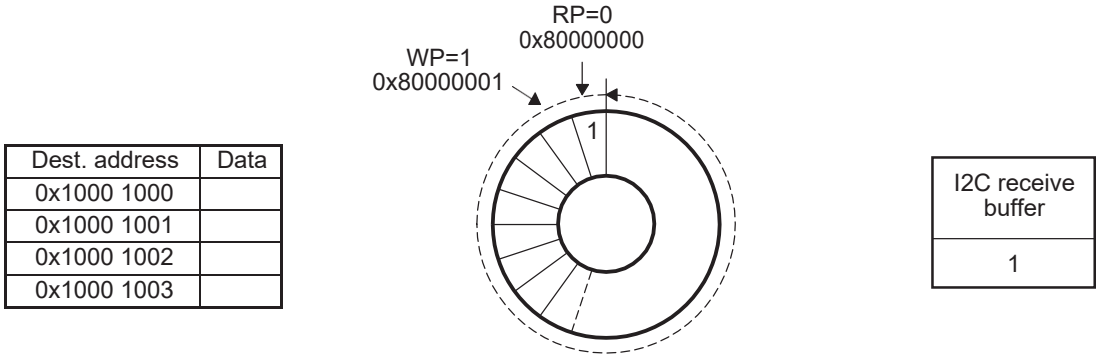
Figure 3-63 shows the state of the memories before the I2C has received any data.

Figure 3-63. FIFO FMARK Example Diagram (Before First I2C Event)



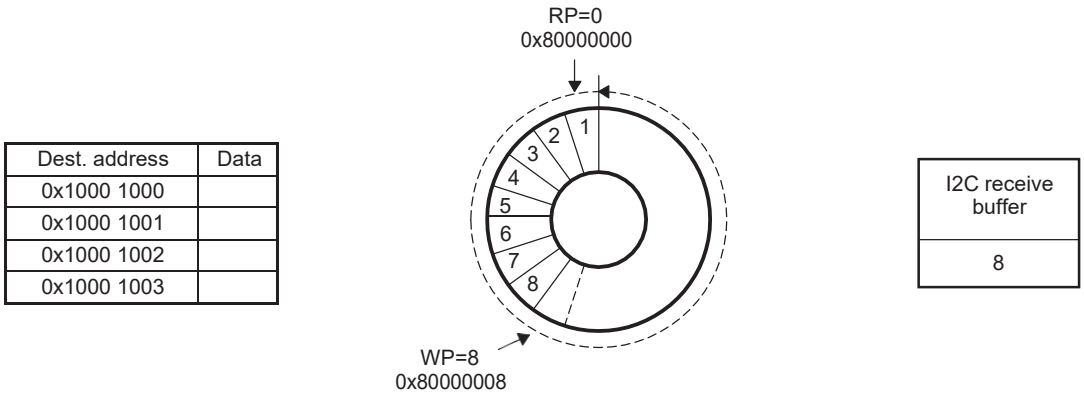
Upon receiving the first data element, the I2C generates an event to dMAX, which in turn performs a FIFO write transfer to place the received data into the FIFO. Figure 3-64 shows the state of the memories at the completion of this transfer.

Figure 3-64. FIFO FMARK Example Diagram (After First Synchronization Event from the I2C)



After the first element is transferred into the FIFO, the FIFO write transfer reloads its counters and address and waits for subsequent events to be generated by the I2C. On each new event from the I2C, the FIFO write transfer will be triggered to copy the new data element into the FIFO. Following receipt of the first element, seven more elements are transferred into the FIFO in the same fashion. At the completion of the eighth transfer, the FMARK watermark is reached and dMAX notifies the CPU. Figure 3-65 shows the state of the memories immediately after the eighth transfer completes.

Figure 3-65. FIFO FMARK Example (After the Eighth Element Has Been Transferred)



At this point, dMAX notifies the CPU by posting the FMSC (set to 1 in this example) to the DFSR and generating an interrupt (INT7) to the CPU. Figure 3-66 shows the state of the DFSRs before posting the FMSC, and Figure 3-67 shows their state after posting.

Figure 3-66. dMAX FIFO Status Registers Before FIFO FMARK is Reached

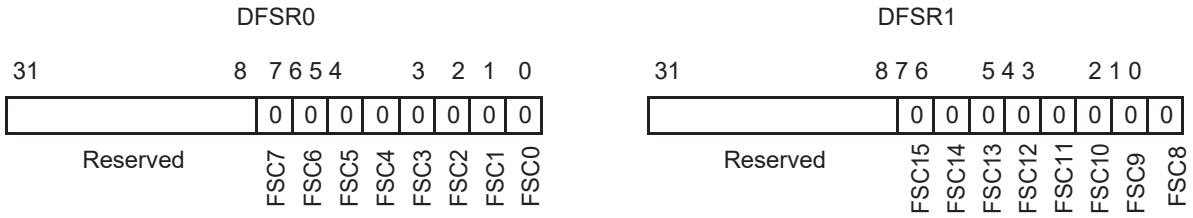
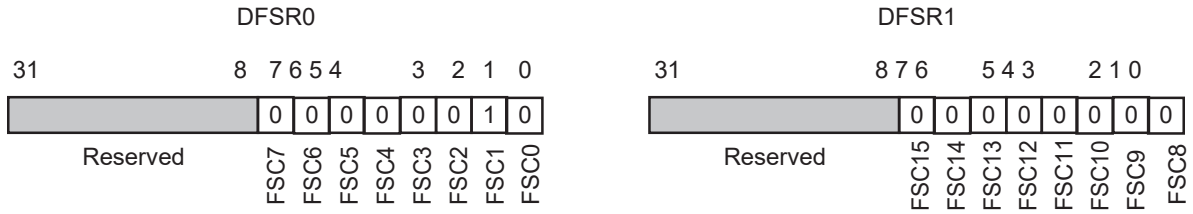


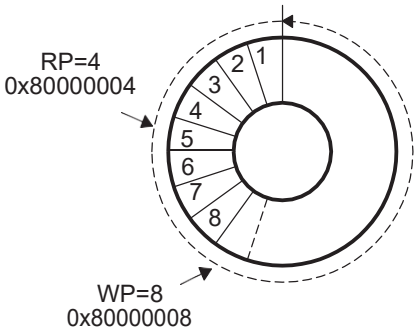
Figure 3-67. dMAX FIFO Status Registers After FIFO FMARK is Reached



The ISR (not shown) that processes the FMARK interrupt then clears the FSC1 bit and triggers a FIFO read transfer to drain four samples from the FIFO. Figure 3-68 shows the state of the memories after the completion of the FIFO read transfer.

Figure 3-68. FIFO FMARK Example (After FIFO Read Transfer)

Dest. address	Data
0x1000 1000	1
0x1000 1001	2
0x1000 1002	3
0x1000 1003	4



I2C receive buffer
8

The CPU can now process the block of four elements while the FIFO write transfer continues to fill the FIFO with data from the I2C. When the number of elements in the FIFO again reaches 8, another FMARK interrupt will be generated to the CPU, and the next block of data can be drained from the FIFO. Ping-pong buffering is not used in this example but could be easily implemented by changing the DST RELOAD ADDRESS 1 field to point to a second buffer.

Figure 3-69 and Figure 3-70 show the event entries and transfer entries for the FIFO write transfer and FIFO read transfer, respectively. Figure 3-71 shows the FIFO descriptor that is common to both transfers.

Figure 3-69. Event Entry and Transfer Entry for FIFO FMARK Example (FIFO Write Transfer)

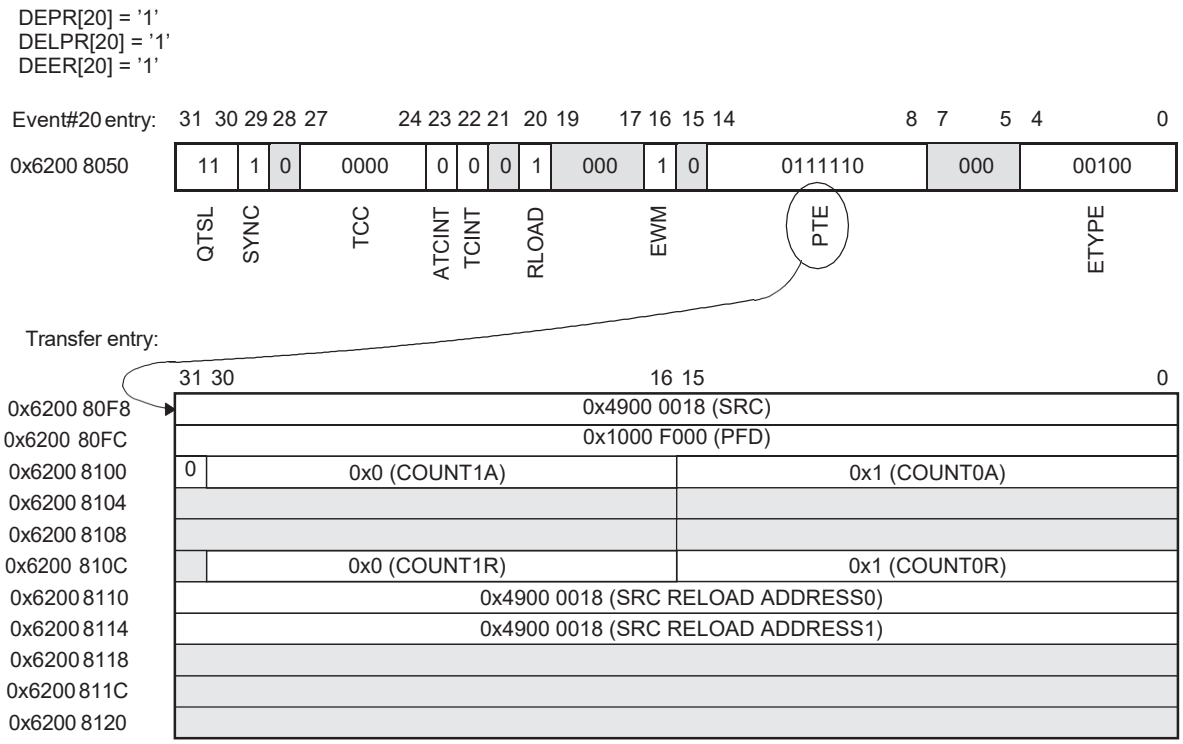


Figure 3-70. Event Entry and Transfer Entry for FIFO FMARK Example (FIFO Read Transfer)

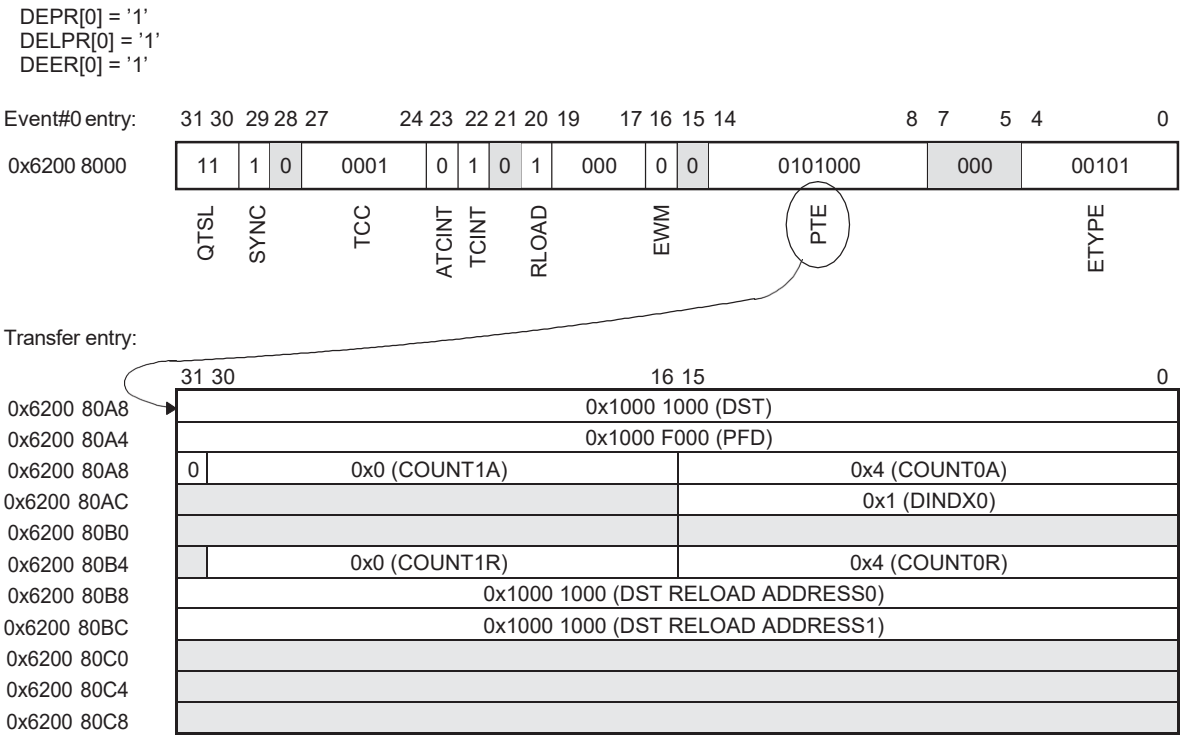


Figure 3-71. FIFO Descriptor for FIFO FMARK Example

FIFO descriptor:

	27	25 24	19	16	0
0x1000 F000	0x8000 0000 (FIFO base address)				
0x1000 F004			0x0 (WP)		
0x1000 F008		0x0	<−(ESIZE)	0x14 (FIFO size)	
0x1000 F00C			0x0 (RP)		
0x1000 F010		0x1	<−(FMSC)	0x8 (FMARK)	
0x1000 F014		0x0	<−(EMSC)		
0x1000 F018		0	<−(FF)		

Event Entry 20, which corresponds to the I2C 0 Receive Event signal, is configured as the FIFO write transfer, and Event Entry 0, which corresponds to a CPU generated event, is configured as the FIFO read transfer. Watermarks are enabled in the FIFO write transfer by setting the WME field to 1, and watermarks are disabled in the FIFO read transfer. Instead, the transfer complete interrupt is enabled for the FIFO read transfer by setting the TCINT field to 1. Reload is enabled for both transfers to allow continuous operation.

The source address field for the FIFO write transfer should point to the receive buffer of the desired I2C peripheral. In this example, address 0x49000018 is used, but the I2C peripheral and device data manual should be consulted to verify the correct address. In addition, the SRC RELOAD ADDRESS0 and SRC RELOAD ADDRESS1 fields in the FIFO write transfer entry are both set to the I2C receive buffer so that each reloaded transfer reads from the same memory location.

The FMARK field in the FIFO descriptor is set to 8 to notify the CPU when there are eight new elements in the FIFO. Also, the FF bit is set to 0 to indicate to dMAX that the FIFO is empty at the start of the transfer. This is necessary because the read and write pointers are equal at the start of the transfer.

3.6.3 EXAMPLE: Servicing I2C Peripherals (FIFO EMARK Watermark)

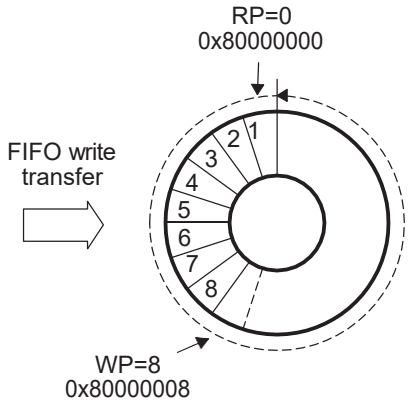
The FIFO EMARK watermark can be used to detect when the number of samples in a FIFO reaches or falls below a programmed value. At the completion of a FIFO read transfer in which EWM = 1 in the event entry, dMAX compares the read and write pointers to determine if the number of samples in the FIFO is less-than or equal-to the value programmed into the EMARK field. If so, dMAX sets a specified bit in one of the DFSRs and generates an interrupt (INT7) to the CPU. The CPU can then trigger a FIFO write transfer to fill more samples into the FIFO.

This example demonstrates using a FIFO read transfer to copy data from a FIFO to an I2C peripheral that has been set up to generate an event to dMAX when it is ready to accept the next element. An EMARK watermark is used by the FIFO read transfer to notify the CPU when the number of elements in FIFO is running low. In processing the EMARK interrupt, the CPU triggers a FIFO write transfer to fill the FIFO with a block of data. At the completion of the FIFO write transfer, the CPU is again notified that the block of data has been placed into the FIFO.

Figure 3-72 shows the state of the memories before the I2C requests any data.

Figure 3-72. FIFO EMARK Example Diagram (Before First I2C Event)

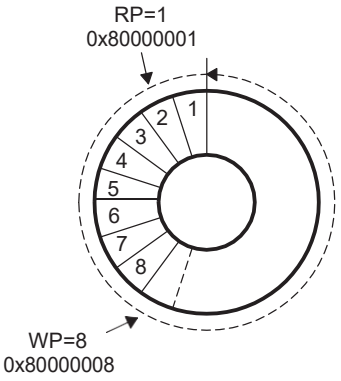
Source address	Data
0x1000 1000	9
0x1000 1001	10
0x1000 1002	11
0x1000 1003	12



When the I2C is ready for a new data element, it generates an event to dMAX, which in turn performs a FIFO read transfer to copy one element from the FIFO to the I2C transmit buffer. Figure 3-73 shows the state of the memories at the completion of this transfer.

Figure 3-73. FIFO EMARK Example Diagram (After First Synchronization Event from the I2C)

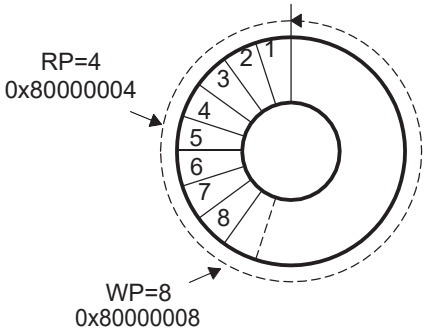
Source address	Data
0x1000 1000	9
0x1000 1001	10
0x1000 1002	11
0x1000 1003	12



After the first element is transferred from the FIFO, the FIFO read transfer reloads its counters and address and waits for subsequent events to be generated by the I2C. On each new event from the I2C, the FIFO read transfer will be triggered to copy another data element from the FIFO to the I2C. After reading the first element, three more elements are read one-by-one from the FIFO in the same fashion. At the completion of the fourth transfer, the EMARK watermark is reached and dMAX notifies the CPU. Figure 3-74 shows the state of the memories immediately after the fourth transfer completes.

Figure 3-74. FIFO EMARK Example (After the Fourth Element has been Transferred)

Source address	Data
0x1000 1000	9
0x1000 1001	10
0x1000 1002	11
0x1000 1003	12



At this point, dMAX notifies the CPU by posting the EMSC (set to 0 in this example) to the DFSR and generating an interrupt (INT7) to the CPU. Figure 3-75 shows the state of the DFSRs before posting the EMSC, and Figure 3-76 shows their state after posting.

Figure 3-75. dMAX FIFO Status Registers Before FIFO EMARK is Reached

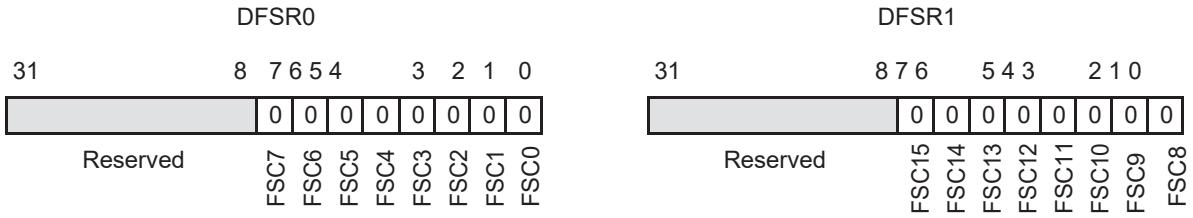
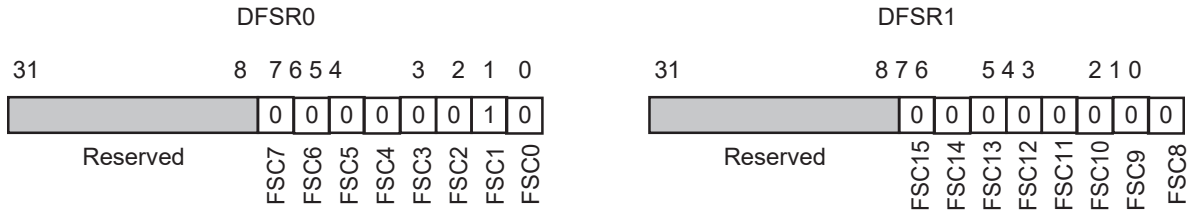


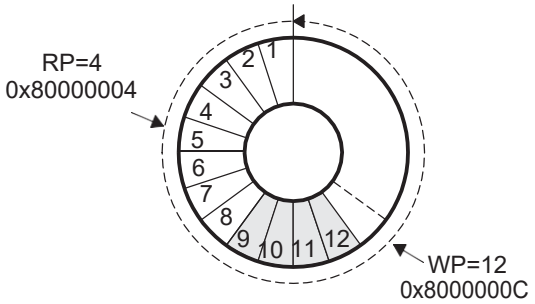
Figure 3-76. dMAX FIFO Status Registers After FIFO EMARK is Reached



The ISR (not shown) processes the EMARK interrupt then clears the FSC0 bit and triggers a FIFO write transfer to fill four more samples into the FIFO. Figure 3-77 shows the state of the memories after the completion of the FIFO write transfer.

Figure 3-77. FIFO EMARK Example (After FIFO Write Transfer)

Source address	Data
0x1000 1000	9
0x1000 1001	10
0x1000 1002	11
0x1000 1003	12



I2C transmit buffer
8

The CPU can now place the next block of data into the source address buffer while the FIFO read transfer continues to drain the FIFO to the I2C. When the number of elements in the FIFO again falls to 4, another EMARK interrupt will be generated to the CPU, and the next block of data can be filled into the FIFO. Ping-pong buffering is not used in this example but could be easily implemented by changing the SRC RELOAD ADDRESS 1 field to point to a second buffer.

Figure 3-78 and Figure 3-79 show the event entries and transfer entries for the FIFO read transfer and FIFO write transfer, respectively. Figure 3-80 shows the FIFO descriptor that is common to both transfers.

Figure 3-78. Event Entry and Transfer Entry for FIFO EMARK Example (FIFO Read Transfer)

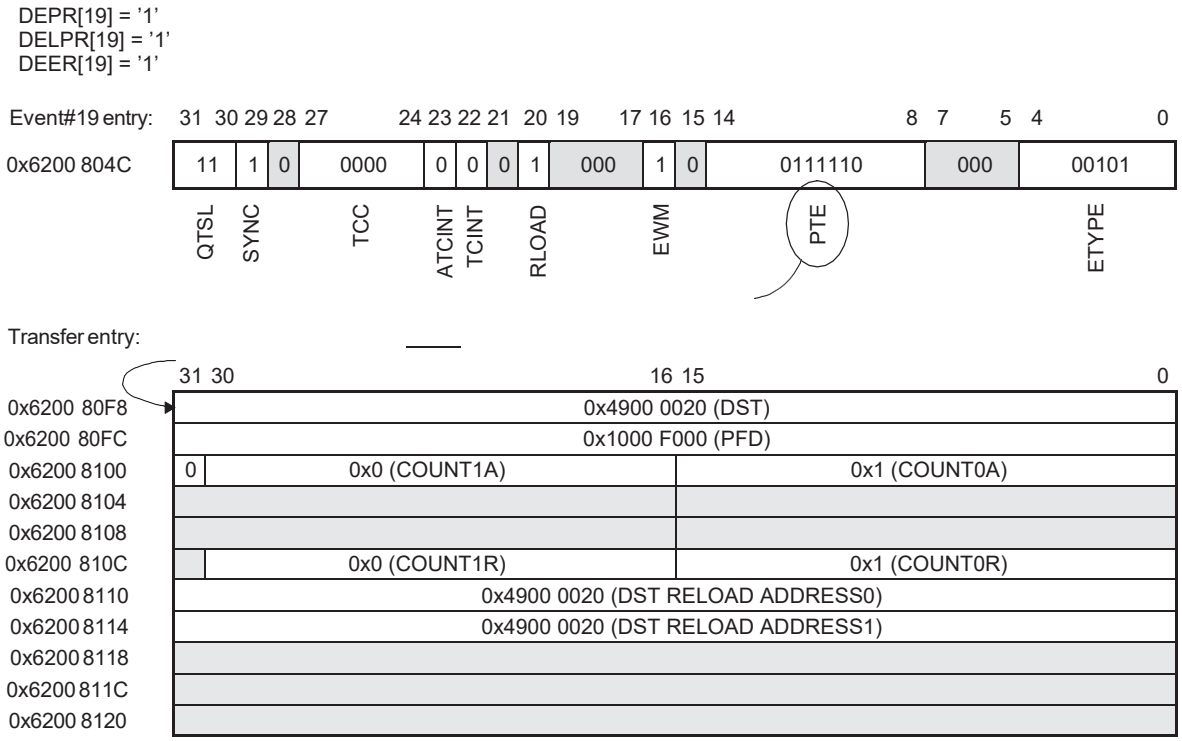


Figure 3-79. Event Entry and Transfer Entry for FIFO EMARK Example (FIFO Write Transfer)

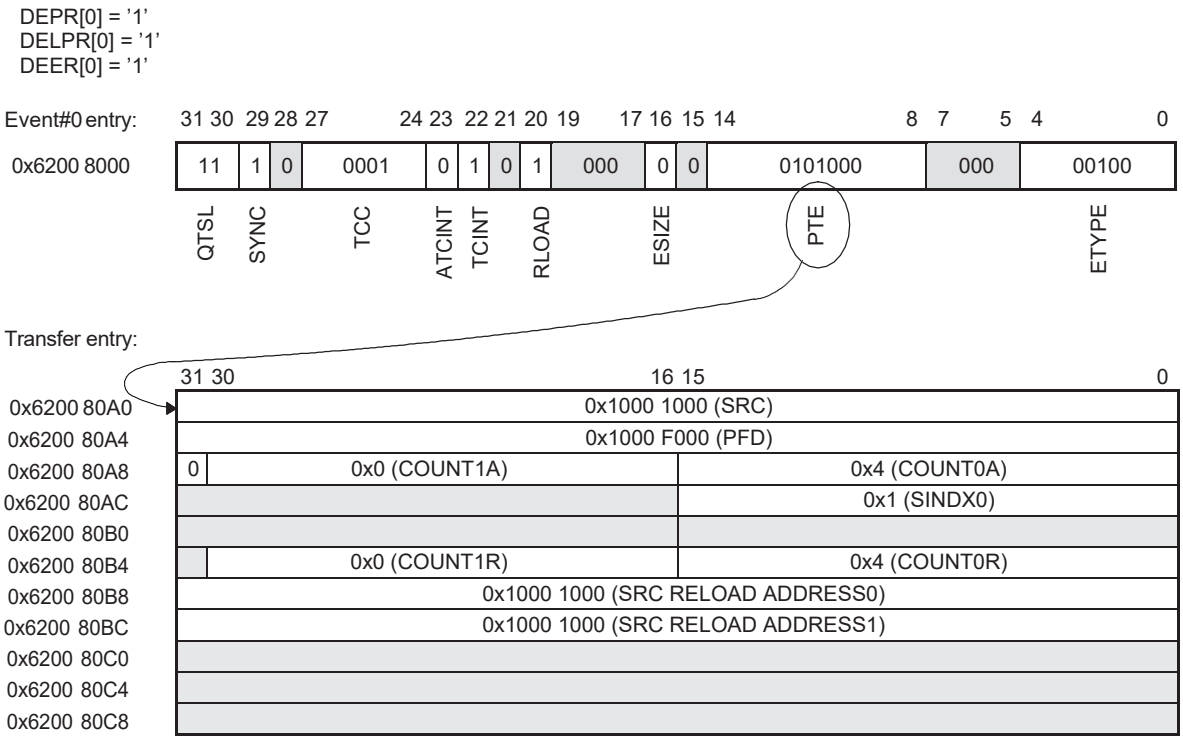
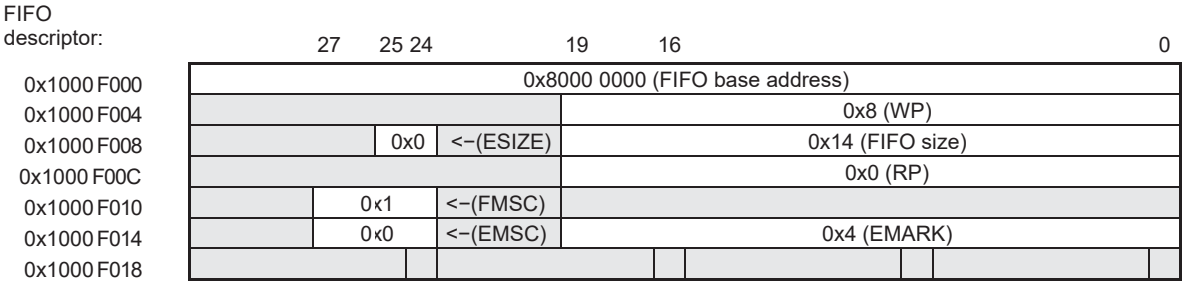


Figure 3-80. FIFO Descriptor for FIFO FMARK Example



Event Entry 19, which corresponds to the I2C transmit event signal, is configured as the FIFO read transfer, and Event Entry 0, which corresponds to a CPU generated event, is configured as the FIFO write transfer. Watermarks are enabled in the FIFO read transfer by setting the WME field to 1, and watermarks are disabled in the FIFO write transfer. Instead, the transfer complete interrupt is enabled for the FIFO Write Transfer by setting the TCINT field to 1. Reload is enabled for both transfers to allow continuous operation.

The destination address field for the FIFO read transfer should point to the transmit buffer of the desired I2C peripheral. In this example, address 0x49000020 is used, but the device data manual should be consulted to verify the correct address. In addition, the DST RELOAD ADDRESS0 and DSDT RELOAD ADDRESS1 fields in the FIFO read transfer entry are both set to the I2C transmit buffer so that each reloaded transfer writes to the same memory location.

The EMARK field in the FIFO descriptor is set to 4 to notify the CPU when there are only four elements left in the FIFO.

3.7 Example of Using dMAX Events to Generate a CPU Interrupt

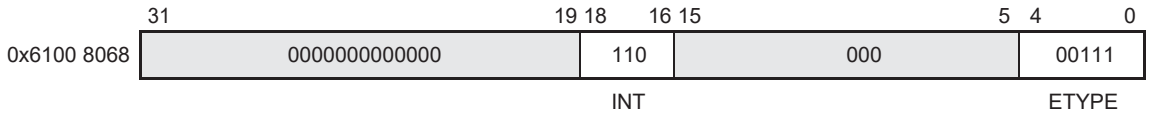
Instead of triggering a data transfer, a dMAX event can be used to trigger a CPU interrupt. In other words, a transition of the event signal (rising or falling edge) can be used to trigger a CPU interrupt.

In order to interpret an event as an interrupt to the CPU, a dedicated event entry is used. In the example, a rising edge of EVENT26 is used to trigger CPU interrupt INT13 (DEPR[26]='1'). An event entry is illustrated in Figure 3-81. In this case the event priority is sorted into the high priority group (DEHPR[26] = '1'), so the event will be processed, and the interrupt will be generated by the HiMAX.

Figure 3-81. Event Used to Trigger CPU Interrupt INT13 Example

DEPR[26] = '1'
DEHPR[26] = '1'
DEER[26] = '1'

Event#26 entry:



The HiMAX module will trigger a CPU interrupt INT13 for a rising edge on the event signal 26.

3.7.1 Using External Signals to Trigger a CPU Interrupt

The McASP has an AMUTEIN signal which is not intended to be a fully controlled GIO pin, but rather to be connected in parallel to one of the device level general purpose interrupt pins. The TMS320C672x has no dedicated general-purpose interrupt pins. Therefore, there is a multiplexer for each McASP which allows the AMUTEIN input for that McASP to be sourced from one of the other I/O pins on the device. Eight different choices are available, and at least one of these should be a spare pin in most applications. Also, it is possible on TMS320C672x to use only one GIO pin to source AMUTEIN to all three McASP modules simultaneously to further conserve pins.

By configuring the CFGMCASP0 register, an external signal can be selected to drive the event signal 26. This way a transition of the external signal can be used to trigger a CPU interrupt.

By configuring the CFGMCASP1 register, an external signal can be selected to drive the event signal 27. This way a transition of the external signal can be used to trigger a CPU interrupt.

By configuring the CFGMCASP2 register, an external signal can be selected to drive the event signal 28. This way a transition of the external signal can be used to trigger a CPU interrupt.

3.8 Examples of dMAX Usage for Delay-Based Effects

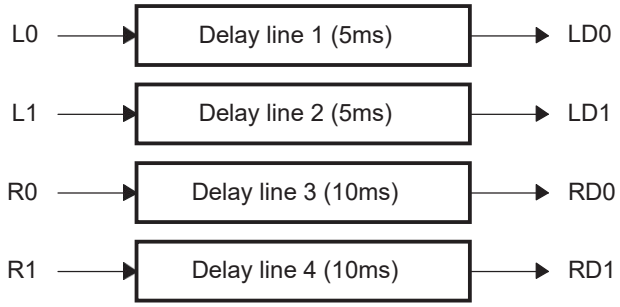
The dMAX controller is a powerful engine that offloads the CPU by bringing the data required to produce delay-based effects into the internal DSP memory. dMAX supports circular addressing and table-guided, multi-tap delay FIFO transfers.

Support for the table-guided, multi-tap delay FIFO transfer enables dMAX to split one FIFO into multiple sections (each section can correspond to either a different channel or to different audio effects). This means that support for the table-guided, multi-tap delay FIFO transfers lets dMAX use only one FIFO per system, such that all required FIFO reads can be handled by only one FIFO read transfer parameter entry, and all required FIFO writes can be handled by only one FIFO write transfer parameter entry. Therefore, two FIFO transfer parameters are required to describe all required FIFO transfers in the system.

An example of using dMAX in a simple, real application is given in this section, using a delay effect on four channels of audio. This basic example is used to illustrate dMAX usage, but the methodology can be extended to the most complicated delay-based effects.

A block diagram of a delay effect on four channels is shown in Figure 3-82. Samples from four input channels L0, L1, R0, and R1 are fed into four delay lines. Each delay line delays the input channel data by a different amount. The output data from the delay lines LD0, LD1, RD0, and RD1 are sent out.

Figure 3-82. Block Diagram of the Delay Effect on Four Input Channels

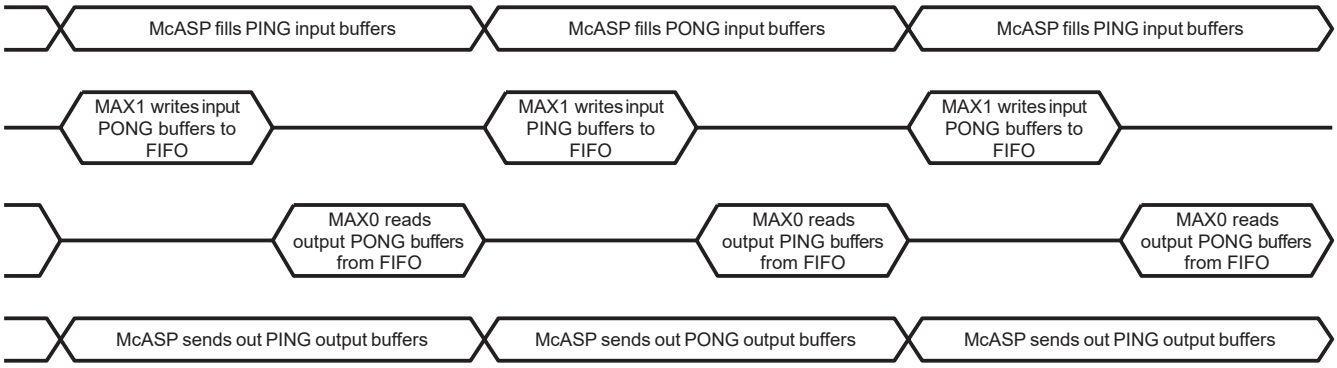


The data samples are received and sent out by McASP. Since this example is used to illustrate typical dMAX usage to produce delay-based effects, the McASP dMAX transfers are not discussed in detail. For more information on how to set up dMAX to receive/send data from the McASP please refer to the example covered in Section 4.7, Servicing the McASP.

To improve dMAX efficiency, data received from the input channels is transferred to the delay lines in blocks. In this example, a block size of four samples is used. When this block of input data (four samples) is received from each input channel, the McASP notifies the CPU by an interrupt. The CPU then initiates a dMAX transfer to store the fresh samples to the FIFO. The CPU also initiates a dMAX transfer to fetch the required delays from the FIFO (the data retrieved from the delay lines will be sent out).

The sequence of events in this example is presented in Figure 3-83.

Figure 3-83. Sequence of Events for Processing



The double-buffering scheme is utilized. Double buffering consists of two input buffers (input ping and pong buffers) and two output buffers (output ping and pong buffers). The fresh samples are stored by the McASP into the input ping (pong) buffer while dMAX is transferring data from the input pong (ping) buffer to the FIFO. The output data is sent out by the McASP from the output ping (pong) buffer while the dMAX is filling the output pong (ping) buffer.

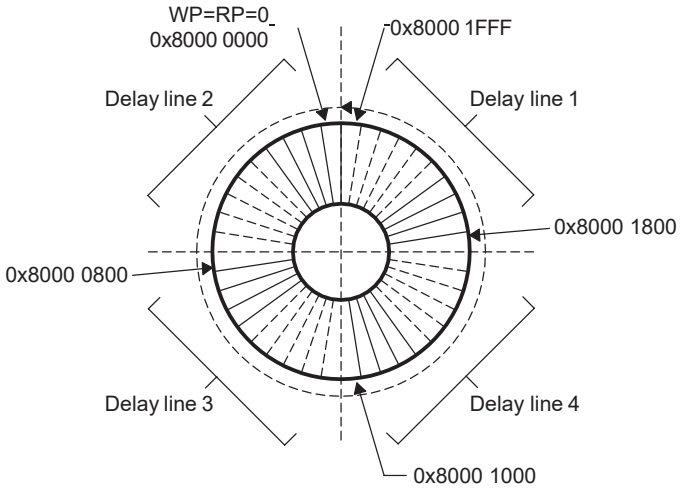
Note in Figure 3-83, that the CPU triggers two FIFO transfers sequentially. The CPU first triggers the FIFO write transfer (executed by the MAX1), followed by the FIFO read transfer (executed by the MAX0). The FIFO transfers are triggered sequentially in this example for clarity only.

When FIFO read and FIFO write transfers are handled by different MAX sub-modules, then both transfers can be triggered by the CPU at the same time, and the two MAX sub-modules will execute the two transfers concurrently. When both FIFO read and FIFO write transfers are handled by the same MAX module, the transfers can be triggered at the same time by the CPU, but the MAX sub-module will execute the transfers sequentially.

To accommodate four different delay lines, the FIFO is split into four sections. Each of the four sections operates as a separate delay line and is assigned to a different input channel. If the sample rate is 48KHz, it is more than enough to reserve space for 1024 elements for each delay line (1024 elements at a sample rate of 48KHz gives the maximum delay of ~21ms). Since there are four delay lines, and each delay line is 1024 elements long, the FIFO size is equal to 4096 elements. In this example, the incoming samples are 16 bits wide; therefore, the FIFO size is 8192 bytes.

A block diagram of the FIFO divided into four sections (four delay lines) is presented in Figure 3-84. By using appropriate delays in the delay guide table for the table-guided, multi-tap delay FIFO transfer, dMAX is capable of accessing each FIFO quadrant. Therefore, only one FIFO read transfer entry is required to complete all FIFO reads, and only one FIFO write transfer entry is required to complete all FIFO writes in the system.

Figure 3-84. FIFO Descriptor and Block Diagram of FIFO



FIFO descriptor:

0x1000 F000	0x8000 0000 (FIFO base address)			
0x1000 F004				0x0 (WP)
0x1000 F008		1	ESIZE	0x1000 (FIFO size)
0x1000 F00C				0x0 (RP)
0x1000 F010		0x1	FMSC	0x1000 (FMARK)
0x1000 F014		0x0	EMSC	0x0000 (EMARK)
0x1000 F018	0	FF		

In this example the FIFO is placed in the external SDRAM starting from 0x8000 0000 and the FIFO descriptor is placed in the internal data memory starting from 0x1000 F000. The FIFO is split into four quadrants of 1024 elements (Figure 3-84). Each FIFO quadrant corresponds to one delay line from Figure 3-83. In this example, element size is equal to 16 bits.

This example illustrates how to, by using table-guided, multi-tap delay FIFO transfers:

- Set up one FIFO write transfer entry to move four blocks of fresh samples on all four channels to their appropriate delay lines
- Set up one FIFO read transfer entry to retrieve four blocks of delayed samples from the appropriate delay lines

The dMAX controller transfers a block of the fresh samples received from each channel to the corresponding FIFO quadrant, retrieves the required delays from each FIFO section, and prepares blocks of the delayed data to be sent out.

To begin operation, the FIFO content should be initialized to zero. The FIFO read poiner (RP) and write pointer (WP) are pointing to the beginning of the FIFO in the FIFO descriptor (Figure 3-84).

A FIFO read transfer can easily be aborted due to FIFO transfer error, so it is important to pay close attention when initializing FIFO pointers RP and WP. Two cases should be considered: reading from an empty FIFO, and trying to read delayed old samples from a FIFO full of new samples. For instance, if both pointers are pointing to the same location (e.g., RP=WP=0), and the FIFO read transfer executes first, the transfer will be aborted because the FIFO is empty and FIFO read attempts a read from an empty FIFO. Another example of when multi-tap delay FIFO read is aborted, is when both pointers are pointing to the same location (e.g., RP=WP=0), and the FIFO Full bit is set in the FIFO descriptor. In this case, the FIFO is full of new samples, and the multi-tap delay FIFO read transfer attempts to retrieve old samples, so the transfer is aborted.

3.8.1 Writing a Block of Fresh Samples to Each FIFO Quadrant

To write a block of fresh samples to the beginning of each of four delay lines (each FIFO quadrant) the CPU initiates a table-guided, multi-tap delay FIFO write transfer.

Only one FIFO write transfer entry is required to move the four blocks of fresh data to the FIFO. After the CPU triggers Event 18, dMAX moves four taps of data (each tap/block is four samples) from the ping/pong input buffers in the linear memory to the FIFO locations listed in the delay table. Each entry in the FIFO write delay table specifies the location to which block of data must be written. The locations in the FIFO write delay table are specified as offsets from the FIFO write pointer. Each of these four delays represents the beginning of one of the four delay lines.

The FIFO is split in the four quadrants and each quadrant corresponds to one delay line. The fresh block of data is always stored at the beginning of the delay line and each delay line is 0x400 elements deep. Therefore, the FIFO write delay table entries for the delay lines zero, one, two, and three are 0x000, 0xC00,0x800, and 0x400, respectively.

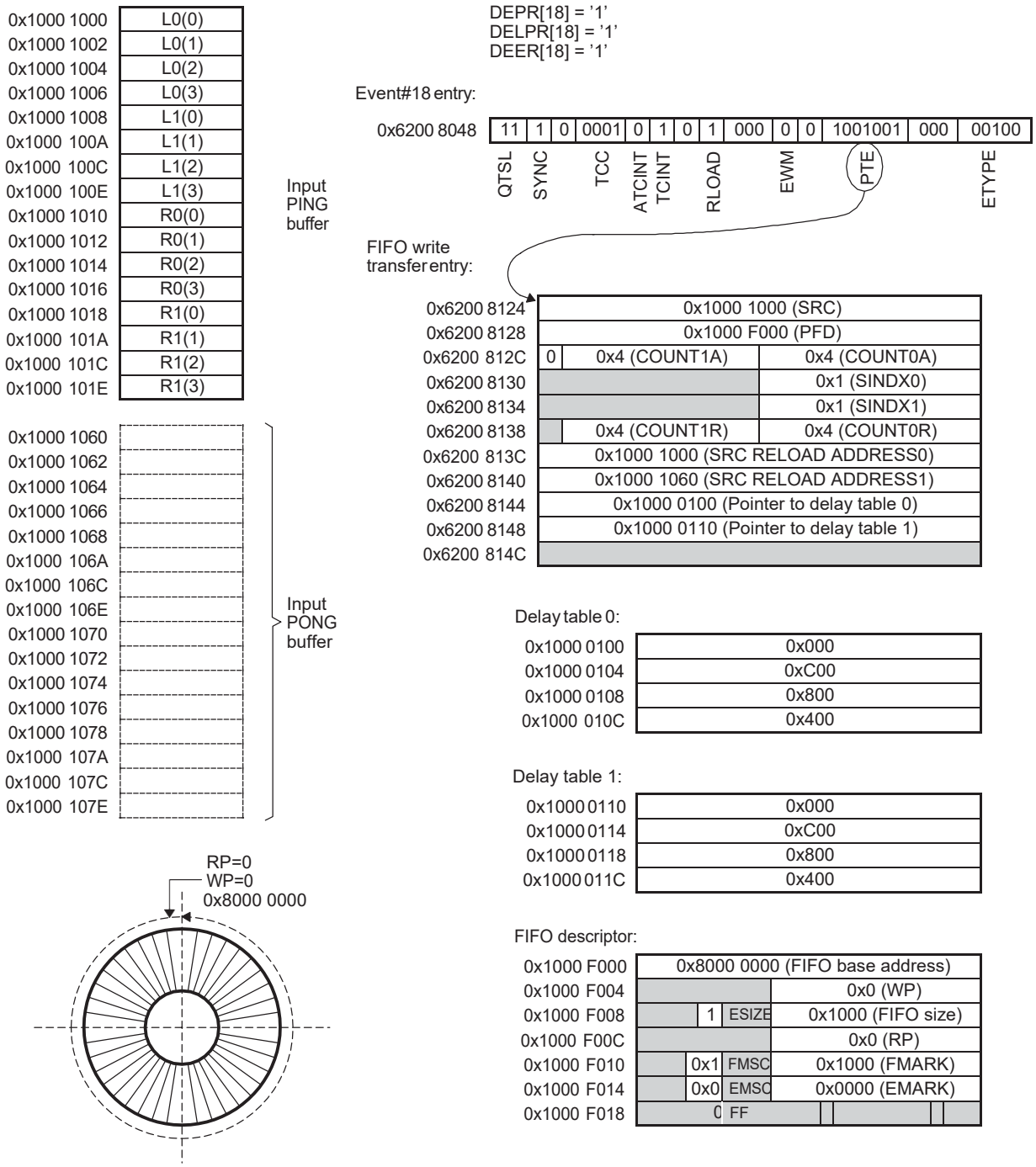
In this example, the CPU uses Event 18 to trigger the table-guided, multi-tap delay FIFO write. Event 18 is assigned to the low-priority group (processed by MAX1). The block size is equal to four, so the first counter dimension (tap size) in the FIFO write transfer entry is set to four. Since one block of fresh samples is moved to the beginning of each delay line, the second counter dimension (number of taps) in the FIFO write transfer entry is set to four.

The SYNC bit field in the event entry is set to one, therefore only one synchronization event is required to complete the whole transfer. The TCINT bit field is set to one, therefore, dMAX will notify the CPU when all four taps are stored to the FIFO by triggering INT8 and by setting the TCC code one in the DESR (and DTCR0). Since reload is enabled (RLOAD is set to one in the event entry), the FIFO write transfer will alternate between input ping and input pong buffers in each subsequent transfer. After the FIFO write is complete, dMAX will use the source address of the pong buffer (0x1000 1060) in the subsequent transfer as a starting address from which data will be moved to the FIFO.

The event entry, FIFO write descriptor and the FIFO transfer descriptor prior to transfer are shown in Figure 3-85.

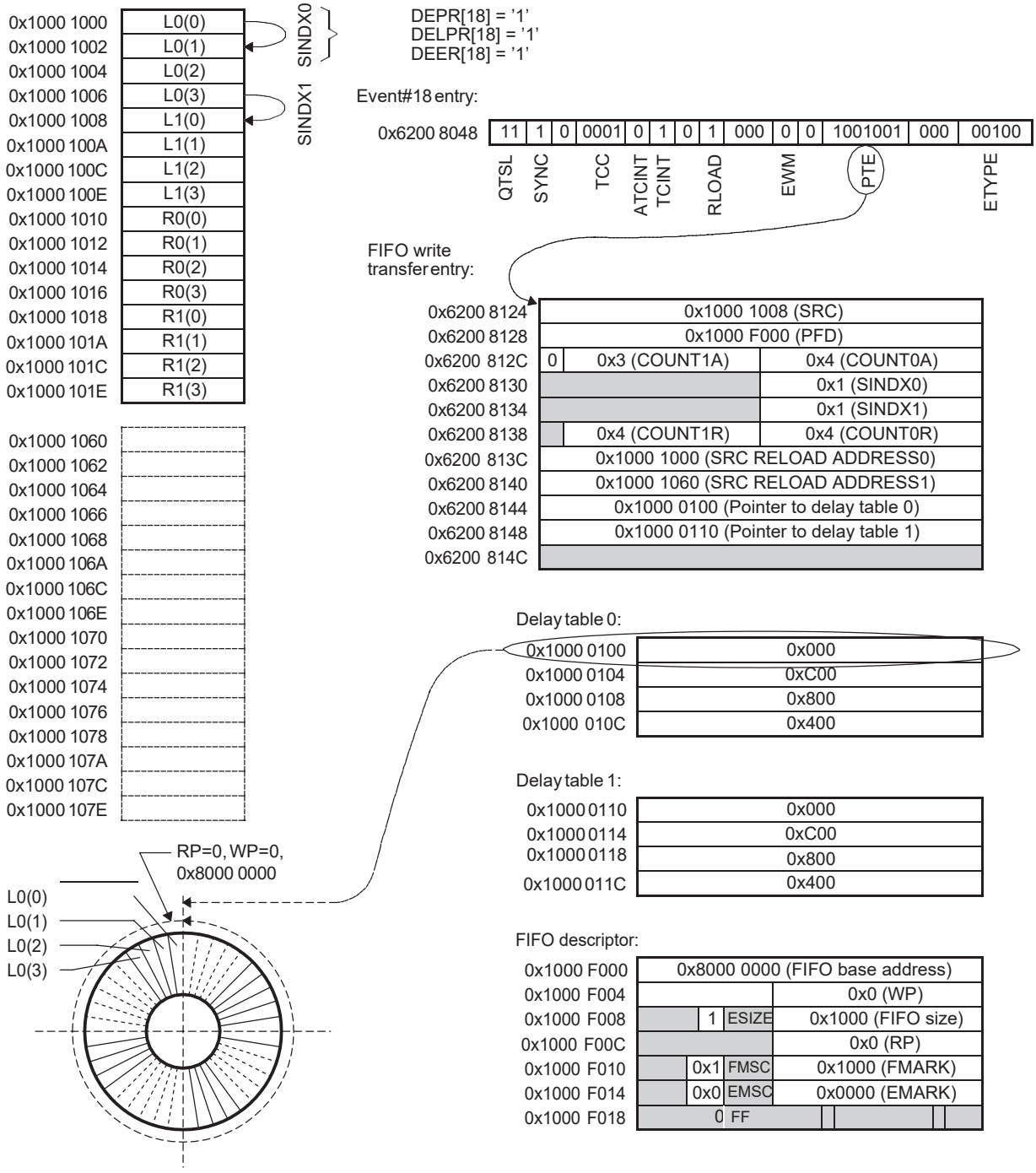
The data received from the McASP is stored starting from location 0x10001000 (ping buffer), or 0x10001060 (pong buffer). After a block of four samples is received from each of four channels, the McASP notifies the CPU by an interrupt.

Figure 3-85. Table-Guided Multi-tap Delay FIFO Write Transfer. Situation Before Transfer Start



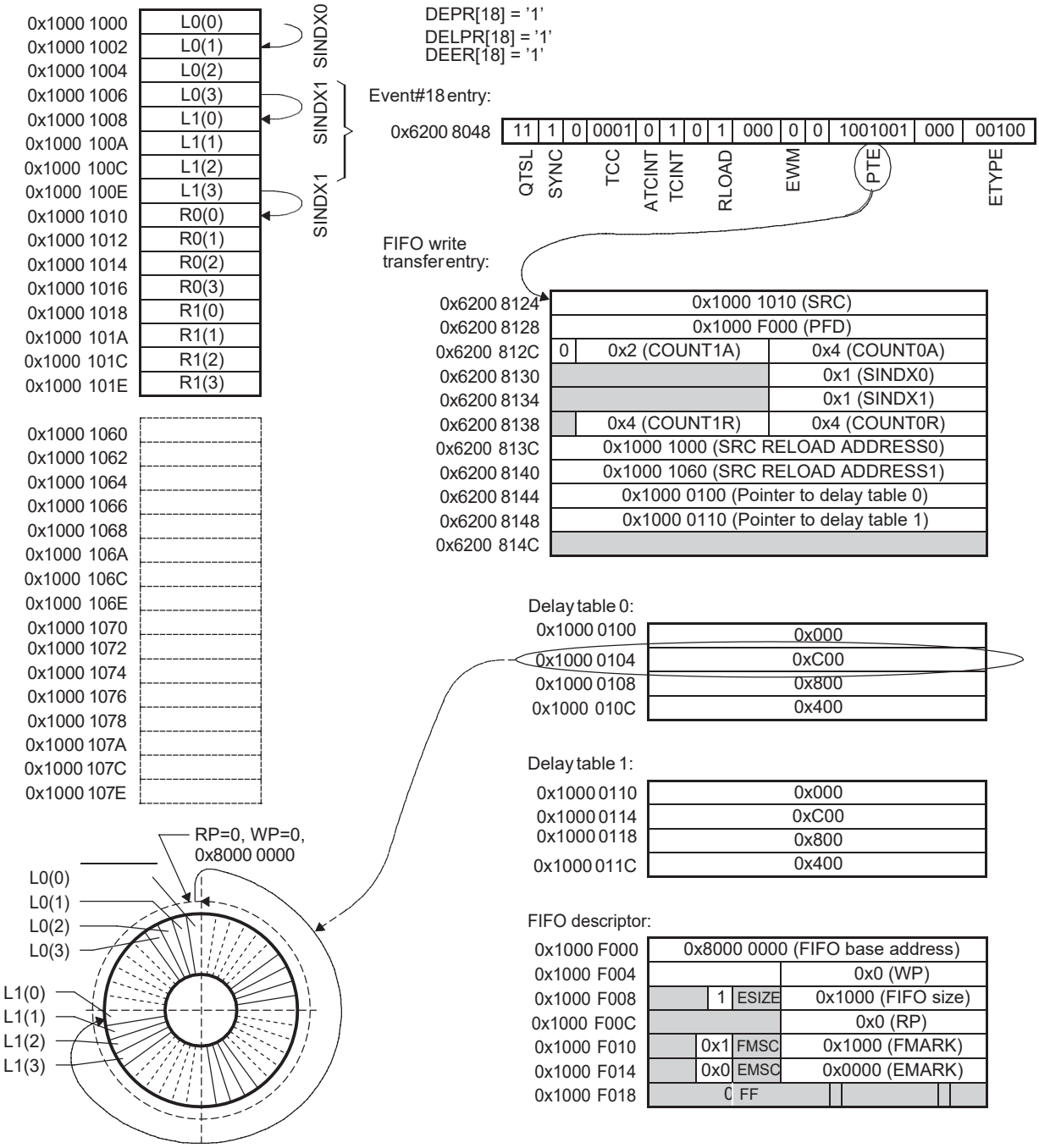
The condition after the first block of data is transferred to the FIFO is shown in Figure 3-86 (FIFO is growing in the counter clock wise direction). The first block of fresh samples from the input ping buffer is written to the beginning of the first delay line. The first entry in the FIFO write delay table is used as a pointer to the FIFO location to which the block of fresh data received from the first channel needs to be stored (the beginning of the first FIFO quadrant).

Figure 3-86. Condition After Fresh Block of Data from the First Channel Moved to the First Delay Line



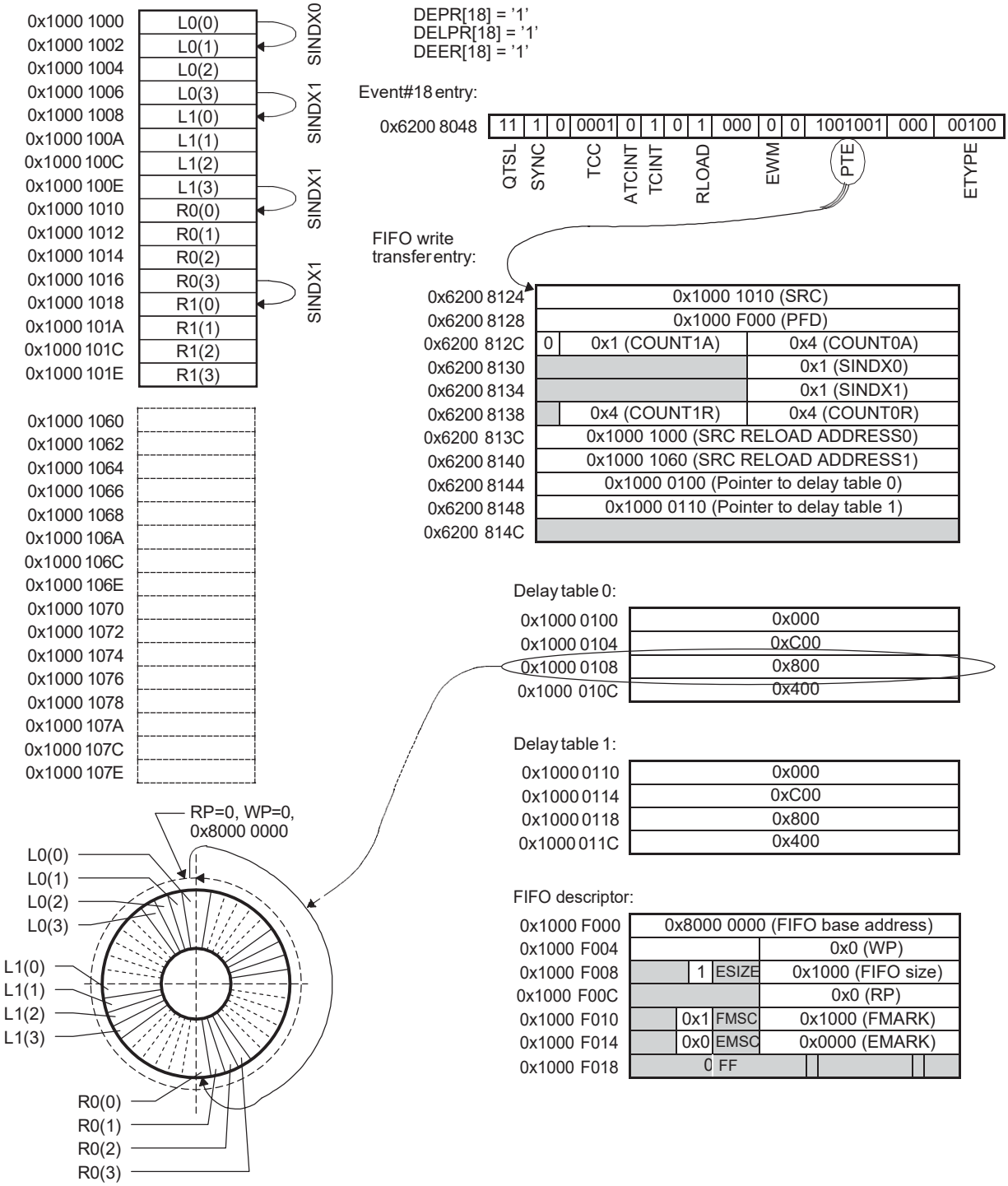
The condition after the second block of data is transferred to the FIFO is shown in Figure 3-87 (FIFO is growing in the counter clock wise direction). The second block of fresh samples is written to the beginning of the second delay line. The second entry in the FIFO write delay table is used as a pointer to the FIFO location to which the block of fresh data, received from the second channel, needs to be stored (the beginning of the second FIFO quadrant).

Figure 3-87. Condition After Fresh Block of Data From the Second Channel Moved to the Second Delay Line



The condition after the third block of data is transferred to the FIFO is shown in Figure 3-88 (FIFO is growing in the counter-clockwise direction). The third block of fresh samples is written to the beginning of the third delay line. The third entry in the FIFO write delay table is used as a pointer to the FIFO location to which the block of fresh data, received from the third channel, needs to be stored (the beginning of the third FIFO quadrant).

Figure 3-88. Condition After Fresh Block of Data From the Third Channel is Moved to the Third Delay Line

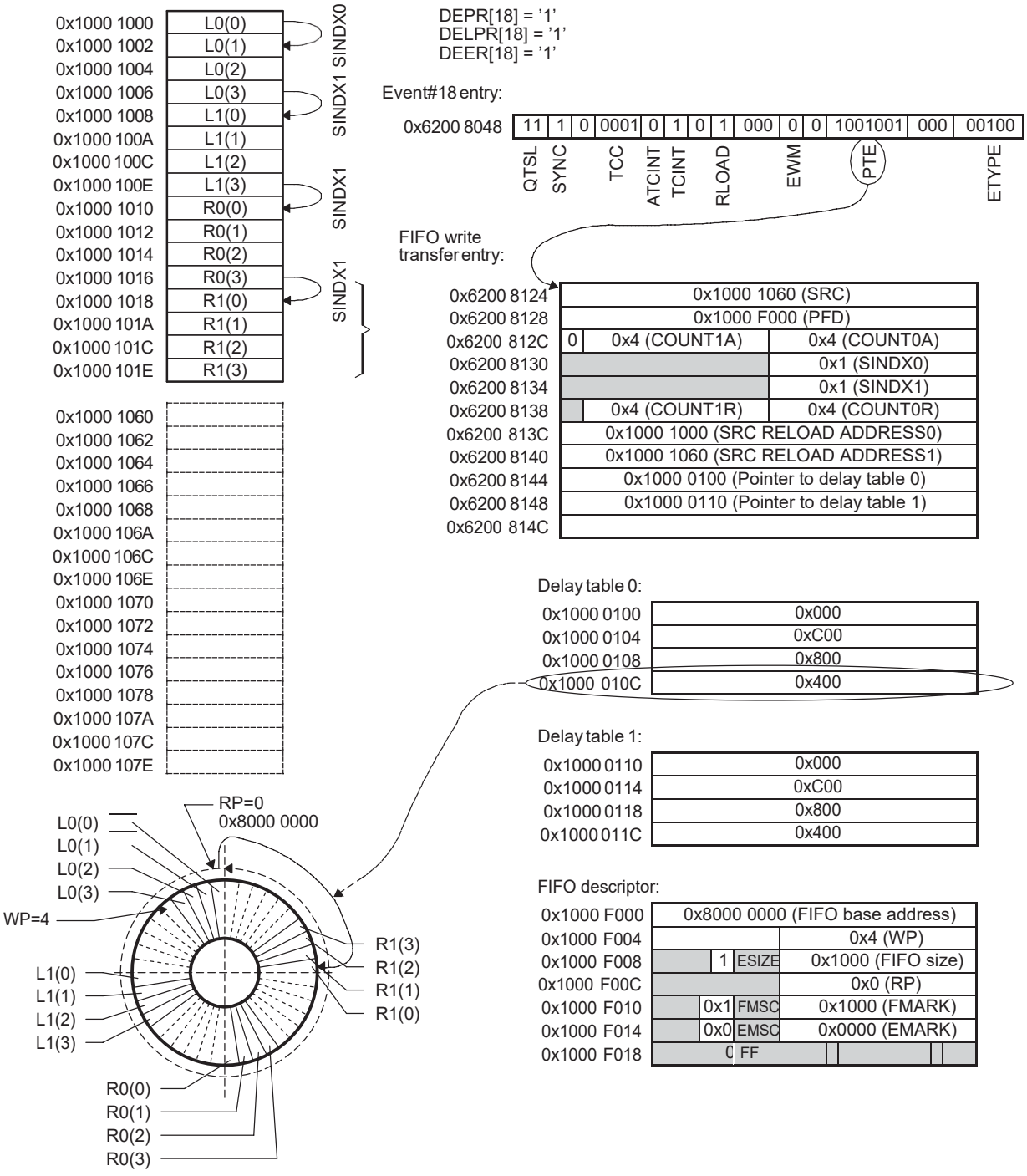


The condition after the last block of data is transferred to the FIFO is shown in Figure 3-89 (FIFO is growing in the counter-clockwise direction). The last block of fresh samples is written to the beginning of the fourth delay line which has an offset of 0x400 samples from the FIFO Write Pointer (during transfer of the last block of data the WP is still equal to zero). The fourth entry in the FIFO write delay table is used as a pointer to the FIFO location to which the block of fresh data, received from the fourth channel, needs to be stored (the beginning of the fourth FIFO quadrant).

After completion of the last phase, dMAX sets the TCC bit to 1 in the DESR and DTCR0 register and triggers INT8 to the CPU. dMAX also updates the FIFO Write Pointer, and reloads the pong source address (0x1000 1060) to be used as source for the next transfer.

By setting RLOAD bit field to 1, dMAX can move fresh data from the ping or pong input buffer to the delay lines within the FIFO. This double buffering comes in handy since dMAX can move data from the input ping (pong) buffer to the FIFO while the McASP is filling the input pong (ping) buffer.

Figure 3-89. Condition After a Fresh Block of Data From the Fourth Channel is Moved to the Fourth Delay Line



3.8.2 Reading a Block of Delayed Samples from Each FIFO Quadrant

To read a block of delayed samples from each of four delay lines (each FIFO quadrant) the CPU initiates a table-guided, multi-tap delay FIFO read transfer.

In this example, the CPU uses Event 17 to trigger a table-guided, multi-tap delay FIFO read. Event 17 is assigned to the high-priority group and is processed by MAX0. Since block size is equal to four, in this example, the first counter dimension (tap size) in the FIFO read transfer entry is set to four. The FIFO contains four delay lines, and one block of delayed samples is required from each FIFO quadrant; therefore, the second counter dimension (number of taps) in the FIFO read transfer entry is set to four.

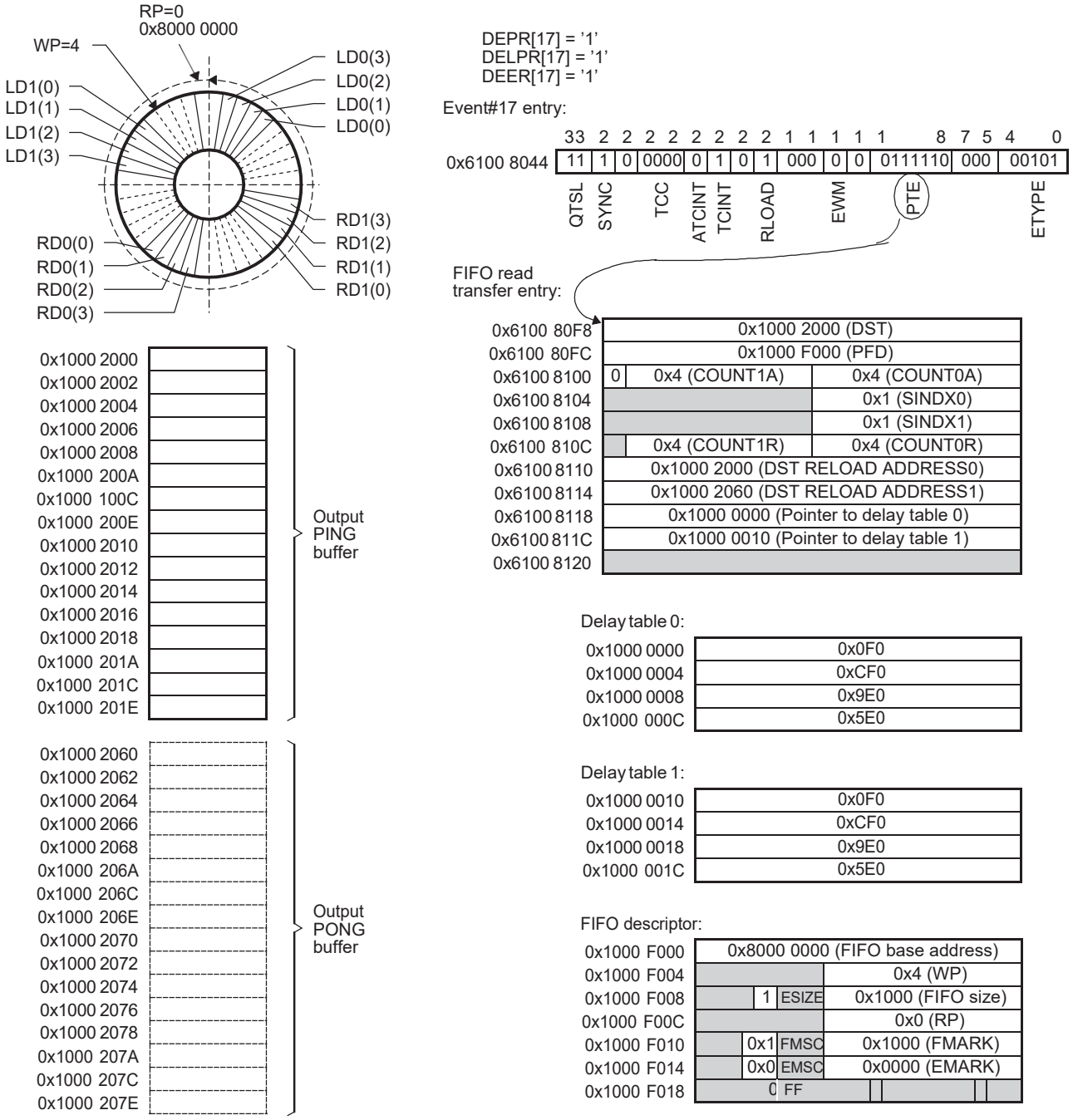
After the CPU triggers Event 17, dMAX will read four taps of data (each tap is four samples long) from the FIFO locations specified by the delay table. Each entry in the FIFO read delay table specifies location/offset from which the block of data must be read (the entries from the FIFO read delay table specify offsets from the FIFO RP).

A delay of 5ms, at a sample rate of 48KHz, translates into a delay of 0xF0 samples. A delay of 10ms, at a sample rate of 48KHz, translates into delay of 0x1E0 samples. Since the FIFO is divided into four quadrants of 0x400 elements, positions of the four delays (LD0,LD1, RD0, and RD1) relative to the FIFO RP are consecutive: 0x0F0, 0xCF0, 0x9E0 and 0x5E0 elements behind the RP (the FIFO is growing counter-clockwise).

Only one transfer entry is required to retrieve all required delays from the FIFO. The SYNC bit field in the event entry is set to 1, so only one synchronization event is required for the whole transfer. The TCINT bit field is set to 1, so dMAX will notify the CPU when all four taps are retrieved from the FIFO by triggering INT8 and by setting the TCC code to zero in the DESR (and DTCR0). Since reload is enabled (RLOAD is set to 1 in the event entry), in the subsequent transfer, dMAX will use the destination address of the pong buffer (0x1000 2060) to store delayed samples read from the FIFO.

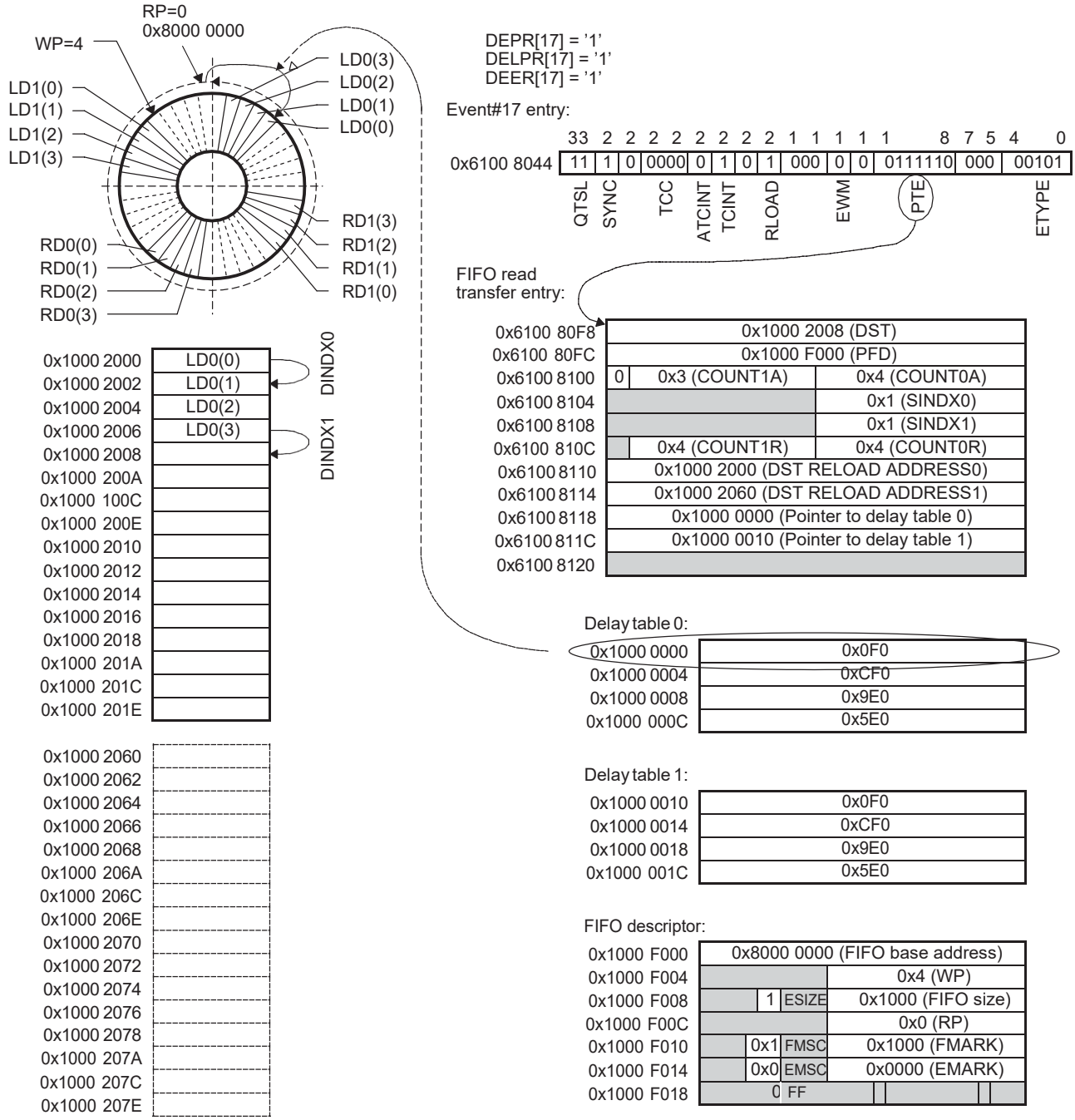
The output ping buffer, FIFO descriptor, and FIFO read transfer descriptor prior to transfer start are presented in Figure 3-90.

Figure 3-90. Reading Delayed Block of Samples From the FIFO Using Table Guided Multi-tap Delay FIFO Read Transfer. Situation Before Transfer Start



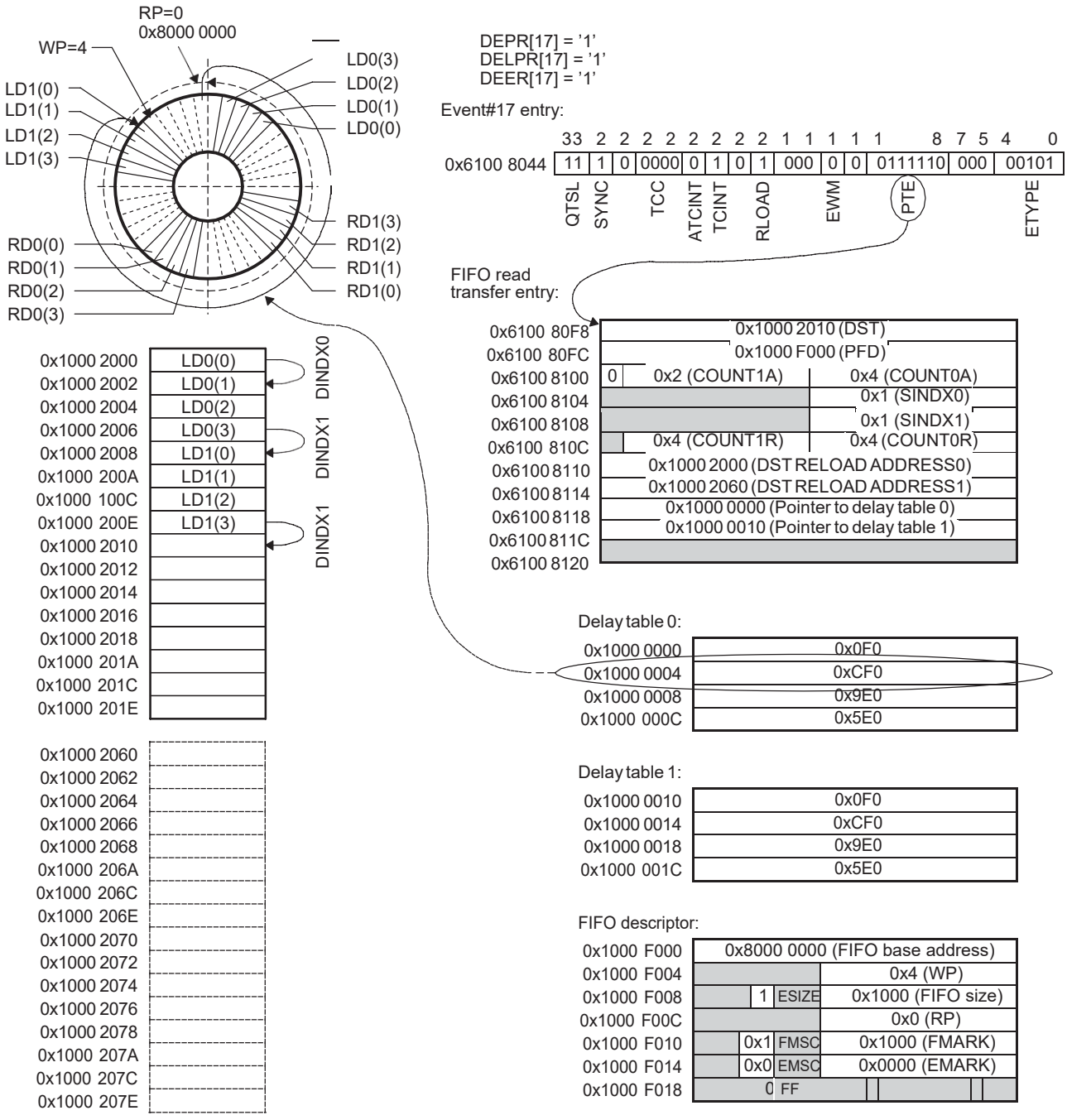
The condition after the first block of delayed data is read from the FIFO is shown in Figure 3-91 (FIFO is growing in the counter-clockwise direction). The first block of delayed samples (output from the delay line 1) is retrieved from the FIFO, and copied to the output ping buffer.

Figure 3-91. Condition After Delayed Block of Data is Retrieved From the First Delay Line



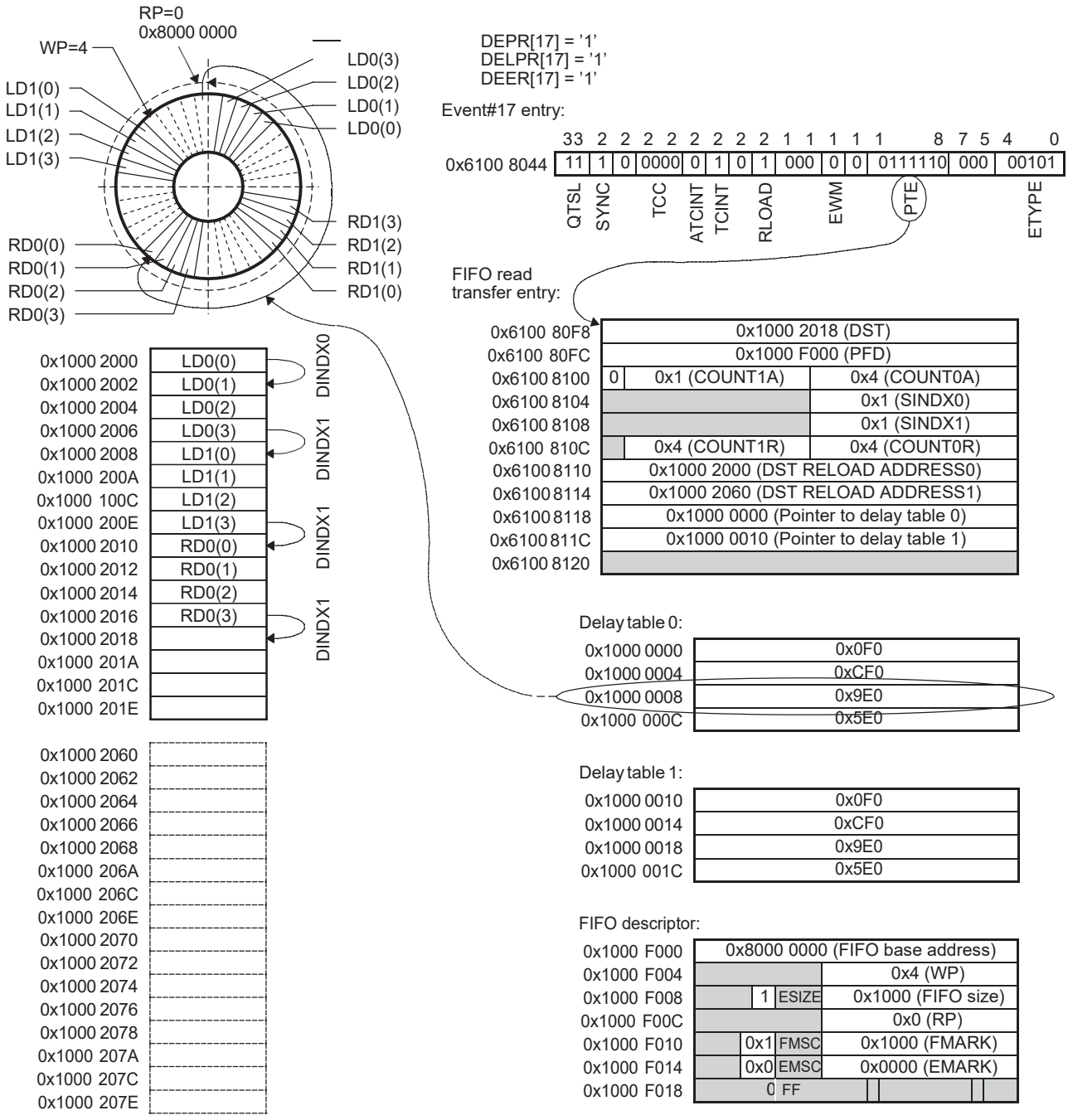
The condition after the second block of delayed data is read from the FIFO is shown in Figure 3-92 (FIFO is growing in the counter-clockwise direction). The second block of delayed samples (output from the delay line 2) is retrieved from the FIFO, and copied to the output ping buffer. The second entry in the FIFO read-delay table is used as a pointer to the FIFO location from which the second block of data needs to be retrieved. The second block of data resides within the second FIFO quadrant.

Figure 3-92. Condition After Delayed block of Data is Retrieved From the Second Delay Line



The condition after the third block of delayed data is read from the FIFO is shown in Figure 3-93 (FIFO is growing in the counter-clockwise direction). The third block of delayed samples (output from the delay line 3) is retrieved from the FIFO, and copied to the output ping buffer. The third entry in the FIFO read-delay table is used as a pointer to the FIFO location from which the third block of data needs to be retrieved. The third block of data resides within the third FIFO quadrant.

Figure 3-93. Condition After Delayed Block of Data is Retrieved From the Third Delay Line



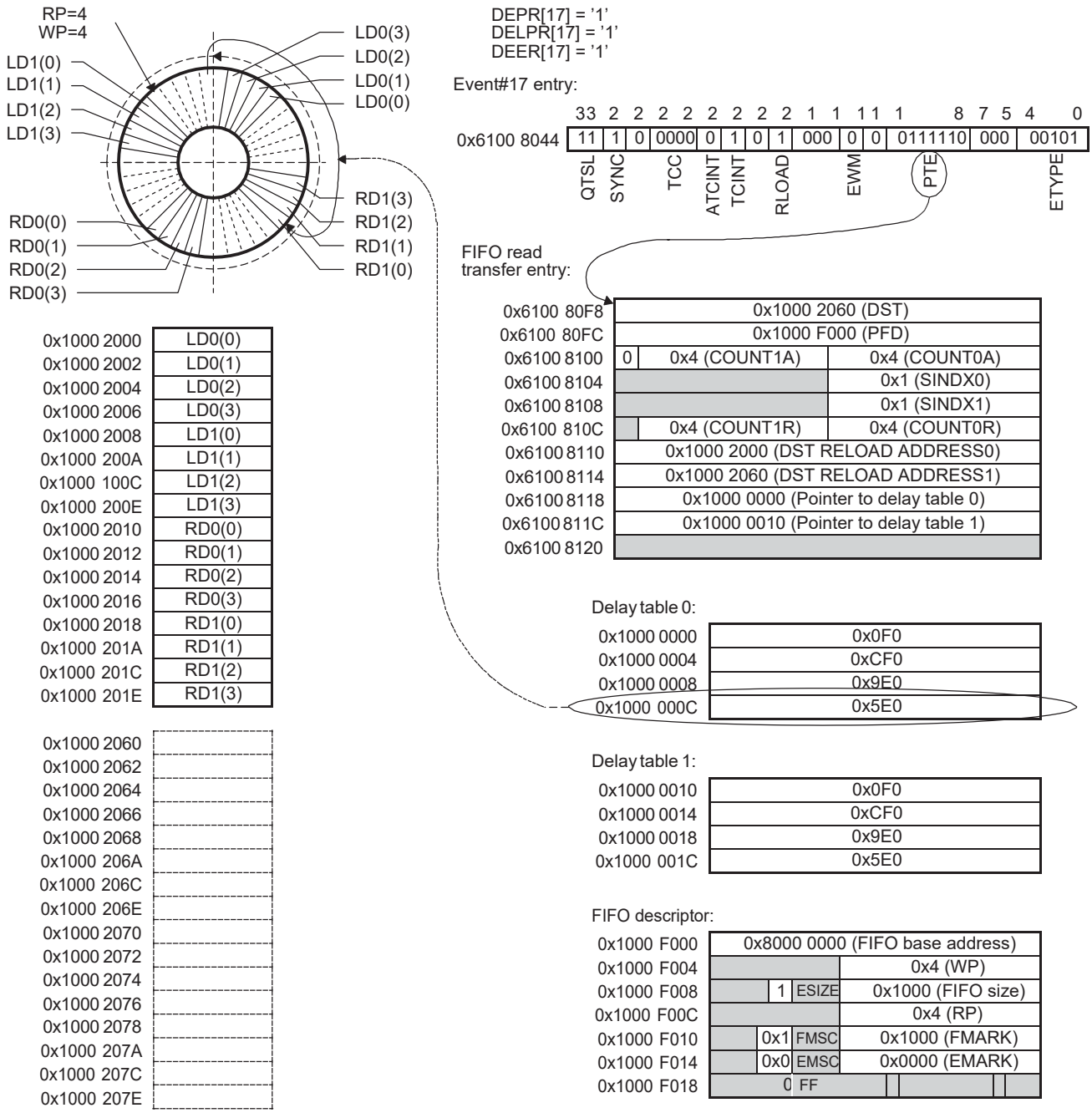
The condition after the last block of delayed data is read from the FIFO is shown in Figure 3-94 (FIFO is growing in the counter-clockwise direction). The fourth block of delayed samples (output from the delay line 4) is retrieved from the FIFO, and copied to the output ping buffer. The fourth entry in the FIFO read delay table is used as a pointer to the FIFO location from which the fourth block of data needs to be retrieved. During the read of the last block of data, the RP is still equal to zero. The fourth block of data resides within the fourth FIFO quadrant.

After completion of the last phase, dMAX sets the TCC bit to zero in the DTCR0 (bit 8 in the DESR) and triggers INT8 to the CPU. dMAX also updates the FIFO RP, and reloads the pong destination address (0x1000 2060) to be used as destination for the next transfer.

The four delayed blocks of data LD0, LD1, RD0 and RD1 (each block contains four samples) are now ready to be sent out by the McASP. Since this example is used to illustrate typical dMAX usage to produce delay-based effects, the McASP dMAX transfers are not discussed in detail. For more information on how to set up dMAX to send data to the McASP please refer to example given in Section 3.6.1.

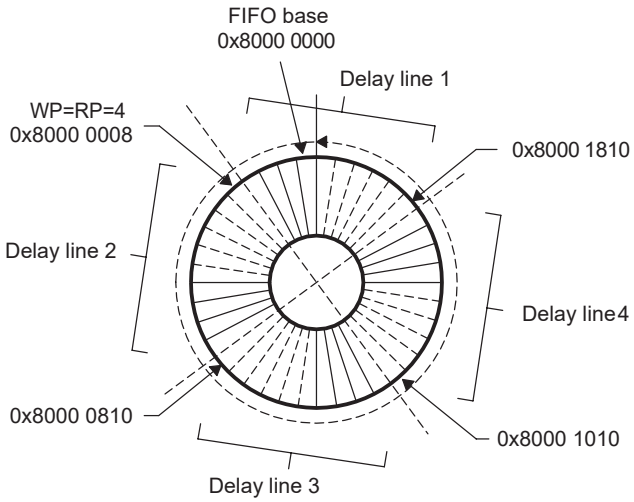
By setting RLOAD bit field to 1, each subsequent transfer alternates between the ping and pong output buffer (ping and pong buffers are switched after each transfer). This double buffering comes in handy since dMAX can move data from the FIFO to the ping output buffer while the McASP is moving data out from the pong output buffer.

Figure 3-94. condition After a Block of Data is Retrieved From the Fourth Delay Line



After a pair of multi-tap delay FIFO read and multi-tap delay FIFO write transfers is complete, the four quadrants representing the four delay lines are rotated as shown in Figure 3-95 (beginning of each delay line within the FIFO moved counter-clockwise by four samples).

Figure 3-95. FIFO Descriptor and Block Diagram of FIFO After Moving Four Delay TAPS to Four Delay Lines



FIFO descriptor:

0x1000 F000	0x8000 0000 (FIFO base address)			
0x1000 F004				0x4 (WP)
0x1000 F008		1	ESIZE	0x1000 (FIFO size)
0x1000 F00C				0x4 (RP)
0x1000 F010		0x1	FMSC	0x1000 (FMARK)
0x1000 F014		0x0	EMSC	0x0000 (EMARK)
0x1000 F018	0	FF		

In this example, the FIFO is split into four quadrants and each FIFO quadrant is used as an independent delay line. The advantage of splitting a FIFO into several sections is that one FIFO write transfer entry, and only one FIFO read transfer entry are sufficient to describe all FIFO transfers in the system.

An inefficient alternative to this approach would be to have four independent FIFO descriptors and four pairs of FIFO read and FIFO write transfer entries for each of four delay lines.

The same methodology used in this example can be extended to any number of effects applied to any number of channels. The methodology from this example can also be extended to a case where a cascade of different effects is applied to the input data. The FIFO can be divided into a number of different sections and each of these sections can be assigned to hold different data. Each audio effect from the cascade can be assigned to a different FIFO section. In this case, the CPU should be used to calculate the outputs from each processing stage in the cascade, while dMAX should be used to maintain the FIFO. Maintaining the FIFO entails fetching data required by the CPU to process the next block, and storing the latest outputs from each processing stage to the appropriate FIFO sections. Only one pair of FIFO transfer descriptors is required to describe all FIFO transfers in the system.

The double buffering is built into the FIFO transfers. If the reload option is enabled in the FIFO transfer event entry, dMAX will alternate between the ping and pong buffers in each subsequent FIFO transfer.

Each of the two buffers used for double buffering is associated with a different delay table (ping and pong buffer can have different delay tables). When dMAX alternates between the ping and pong buffers, it also alternates between the two delay tables used during the FIFO transfer. This feature comes in handy when it is required to change the delay table on the fly during FIFO transfer (e.g., to implement the low frequency oscillator-LFO effect). The CPU can only modify the pointer to the FIFO delay table used by the ping (pong) buffer while dMAX is performing pong (ping) transfer.

dMAX Controller Performance

This chapter provides performance and throughput data for dMAX, along with guidelines to follow to obtain the best performance. Also, the attached spreadsheet contains performance calculations and can be downloaded from this link http://www-s.ti.com/sc/techlit/spru795d.zip.

Topic **Page**

4.1 Overview

The dMAX controller handles user-programmed data transfers between the internal data memory controller and the device peripherals on the C672x DSP. The performance data is given in the number of dMAX clocks required to perform a specific transfer, which enables you to easily scale and calculate data throughput for different dMAX clock frequencies.

The dMAX performance is characterized by:

- The maximum data throughput (in Mbytes/s)
- Number of dMAX clocks required to perform a transfer
- Latencies required to start a transfer

System performance is affected by a variety of factors:

- The number of tasks handled by the dMAX controller. All the tasks compete for the dMAX controller resources
- dMAX controller clock to EMIF clock ratio and dMAX controller clock to CPU clock ratio
- dMAX controller competes with the CPU and the UHPI for resources (external memory interface (EMIF)), internal memory
- If the resource accessed is within the EMIF, it is susceptible to stalls such as SDRAM page misses, and asynchronous, not-ready conditions

The data presented in this chapter represents the best-case scenario when there are no resource conflicts during transfer. Many of these factors are within the system designer's control. This chapter uses a best-case performance to give the system designer an idea of the upper band of available bandwidth.

The performance discussed below is for only one MAX module; however, using both MAX modules can effectively double the performance.

4.2 Guidelines for Getting the Best dMAX Performance

The dMAX controller includes features such as the capability to:

- Perform three-dimensional data transfers for advanced data sorting
- Manage a section of the memory as a circular buffer/FIFO, with delay tap based reading and writing data
- Concurrently process two transfer requests (provided that they are to/from different source/destinations)

When splitting tasks between MAX0 and MAX1 modules during the system design process, the tasks should be split between the two dMAX modules to minimize system latency

The Transfer Completion Code (TCC) bit field is used to notify the Interrupt Service Routine (ISR) which data transfer has been completed. The CPU can read the TCC from the DESR, DTRC0, and DTCR1, and each read provides the same information. However, since the DESR is internal and the CPU access to it is minimal, reads from it are more efficient. The DESR is read-only and to clear a TCC code, the CPU must perform a write to the DTCR0 or DTCR1 (depending on the TCC that needs to be cleared).

The data required by the CPU should be placed in the fast internal memory. To obtain the maximum performance, the CPU should access only data from the fast internal memory. The activities should be divided so that dMAX and the CPU complement each other. While the CPU is processing a current block of data from the internal memory, the dMAX controller should bring to the internal memory, the next block of data that will be required by the CPU.

4.2.1 General Purpose Transfer: Best Performance Tips

The following are tips on generating the best performance from general-purpose transfers.

- To get the highest throughput, use large QTSL values and maximize COUNT0. If COUNT0 is greater than the QTSL specified in the event entry, the transfer will be split in several quantum transfers. Specifying a large QTSL value will allow dMAX to move more data during each quantum transfer, and therefore to achieve higher performance.
- Use of small QTSL values works better if low latency is required. If a new dMAX request arrives in the middle of a data transfer, small QTSL values can help to reduce the dMAX reaction time. If small QTSL values are used, dMAX will use smaller quantum transfers. Smaller quantum transfers take less time to complete so the new request is processed more quickly.
- The burst type of transfers (where INDEX0 is equal to one) have the maximum throughput. To achieve the maximum performance, the burst transfers should be used wherever possible. Data sorting transfers (where INDEX0 $\neq$ 1) are slower than moving sequential data.

4.2.2 FIFO Transfer: Best Performance Tips

The following are tips on generating the best performance from FIFO transfers:

- When specifying a FIFO transfer use a block size (tap size) larger than one. Elements within the block (tap) should be consecutive (use INDX0=1). Fulfilling this requirement helps to maximize utilization of dMAX, external memory interface (EMIF) and the CPU for the following reasons:
  1. The SDRAM memory is optimized for burst accesses of consecutive data. The circular buffer (FIFO) holding long delay-line samples is usually placed in the external memory (SDRAM).
  2. The dMAX controller is optimized to achieve the maximum performance when moving blocks of consecutive data (INDEX0=1).
  3. The CPU is most efficient when processing data in blocks.
- When possible, use large QTSL values for higher performance. Use of small QTSL values works better in case low dMAX latency is required
- Specify the block size (tap size) top be a multiple of the QTSL value. This helps to best utilize dMAX bandwidth to move the maximum amount of data in minimum number of quantum transfers.
- Specify the FIFO size to be a multiple of the block size.
  Data is moved to/from FIFO in blocks; therefore, if a FIFO size is not a multiple of block size, a transfer crossing the upper FIFO boundary will be split into two quantum transfers. The first quantum transfer will move the first part of block data and will end at the upper FIFO boundary; and the second quantum transfer would wrap around the upper FIFO boundary back to the FIFO base and move the remaining data. This helps to best utilize dMAX bandwidth to move the maximum amount of data in the minimum number of quantum transfers.
- Specify the value for each entry within the delay table to be a multiple of the block size. The entries from the delay table specify offsets relative to the FIFO pointer. The data block should be transferred to/from the FIFO locations pointed to by the delay table entries.
  If a specified delay value is not a multiple of block size, the data transfer might cross the upper FIFO boundary. If that happens, the transfer will be split into two quantum transfers. The first quantum transfer will move the first part of block data and it will end at the upper FIFO boundary; the second quantum transfer will wrap around the upper FIFO boundary back to the FIFO base and move the remaining data.
  If the delay value is a multiple of block size, transfer of a block of data always aligns with the FIFO boundary. This helps to best utilize dMAX bandwidth to move the maximum amount of data in the minimum number of quantum transfers.
- Put the FIFO descriptor in the internal memory for faster access by dMAX.
- Put delay tables (tables that guide the 2D FIFO transfer) in the internal memory for faster access by dMAX.

4.2.3 One-Dimensional Burst Transfer: Best Performance Tips

The following are tips on generating the best performance from One-Dimensional burst transfers.

- To get the highest throughput, use large BURSTLEN. If CNT is greater than the BURSTLEN specified in the transfer entry, the transfer will be split in several burst transfers. Specifying a large BURSTLEN value will allow dMAX to move more data during each burst transfer, and therefore to achieve higher performance.
- Use large NBURST. Specifying a large NBURST will allow dMAX to perform more burst transfers before it checks for any higher priority pending transfer. This allows the dMAX to achieve higher performance.
- Use of small BURSTLEN values works better if low latency is required. If a new dMAX request arrives in the middle of a data transfer, small BURSTLEN values can help to reduce the dMAX reaction time.
- Use of small NBURST values avoids blocking higher priority pending transfers. Chose the NBURST carefully to not starve the higher priority pending transfers and also to achieve maximum performance.

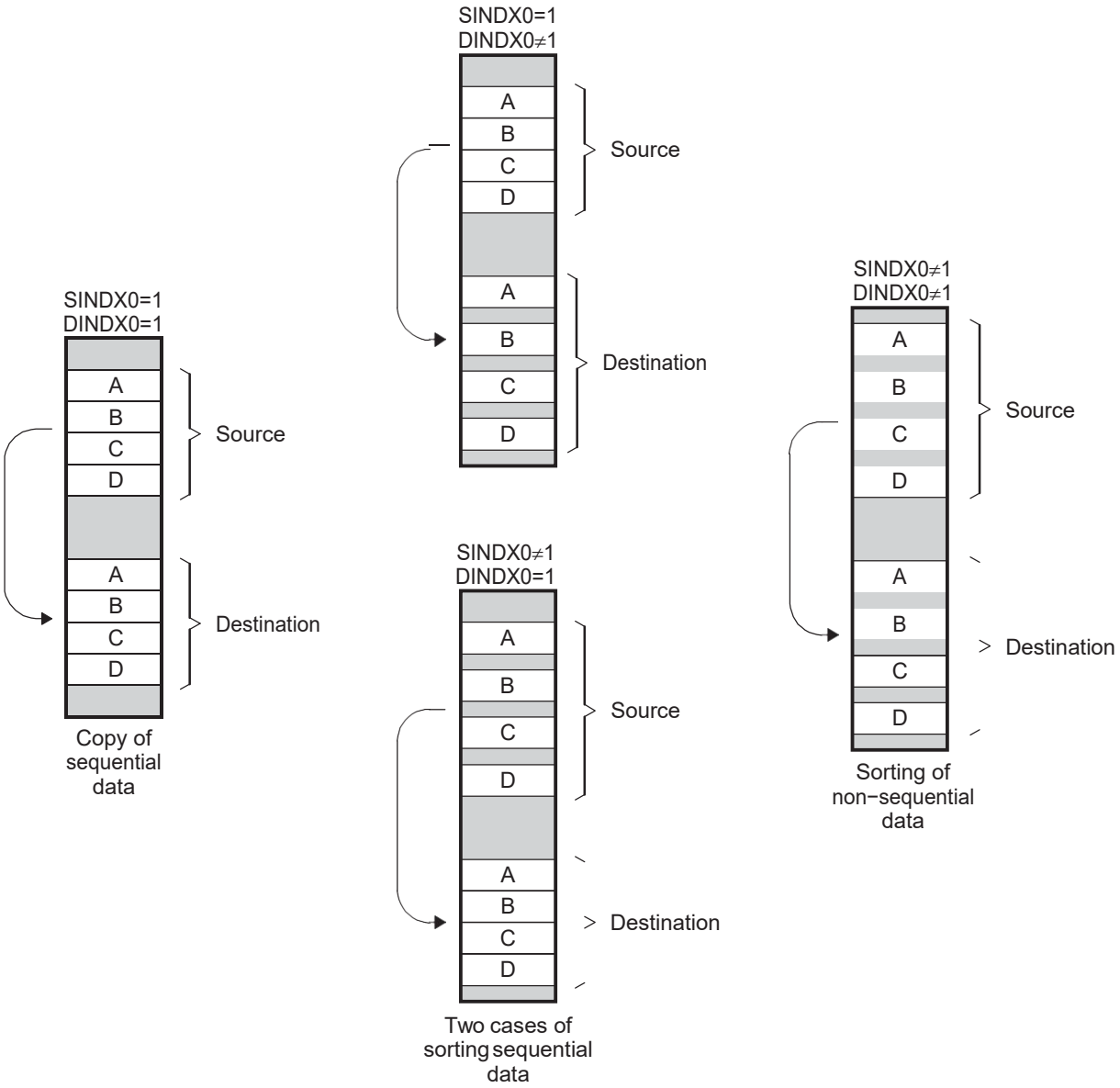
4.3 General Performance Transfer Performance

General-purpose data transfer performance is evaluated for three types of transfers, all of which have different levels of performance:

- Sequential data copy form source to destination (SINDX0=1 and DINDX0=1)
- Sorting of sequential data (SINDX0=1 and DINDX0≠1 or SINDX0≠1 and DINDX0=1)
- Sorting of non-sequential Data (SINDX0≠1 and DINDX0≠1)

The three type of transfers are depicted in Figure 4-1.

Figure 4-1. Three Transfer Types Used to Collect Performance Data



For sequential data copy, dMAX moves a block of sequential data from the source to the destination (shown in the left panel).

For sequential data sorting (presented in the middle panel), dMAX either sorts a block of sequential data from the source to the non-sequential locations at the destination, or sorts non-sequential source data to sequential locations at the destination.

Sequential data sorting covers two cases:

- The sequential source data (SINDX0 =1) is sorted out to non-sequential destination locations (DINDX0≠1), illustrated by the upper-middle panel.
- The non-sequential source data (SINDX0 ≠1) is sorted out to sequential destination locations (DINDX0=1), illustrated by the lower-middle panel.

In non-sequential data sorting, both source and destination locations are non-sequential: SINDX0≠1 and DINDX0≠1 (shown in the right panel).

The transfer duration is expressed in the number of dMAX clocks required to complete the transfer. The following formula can be used to calculate bandwidth in bytes/s:

$$Data\_Tput[Bytes/s] = \frac{dMAX\_Clock\_Freq[MHz] \times Transfer\_Size[Number\ of\ Elements] \times Element\_SIZE[Bytes]}{Transfer\ Duration[Number\ of\ dMAX\ clocks]}$$

where *Data_Tput* is data throughput expressed in Bytes/s, *dMAX_Clock_Freq* is frequency of the dMAX controller in MHz, *Transfer_Size* is length of transfer in number of elements, and *Element_SIZE* is element size expressed in bytes.

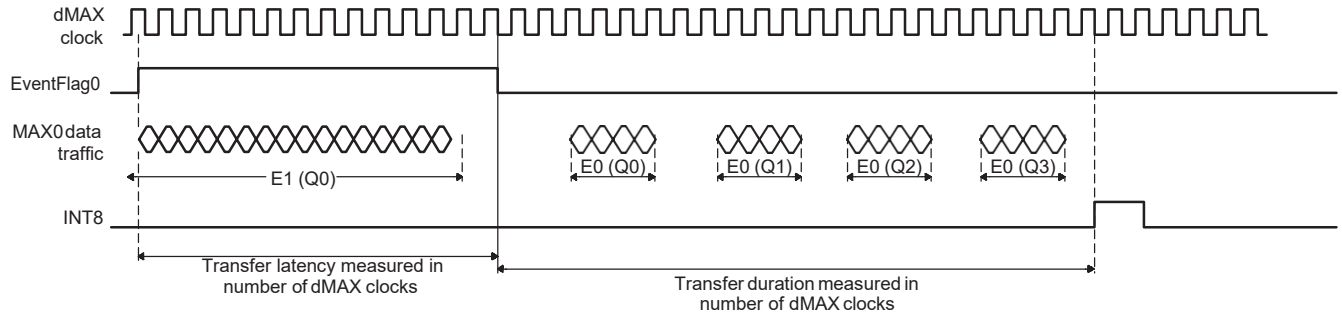
4.4 Transfer Duration and Latency

The transfer duration is measured in the number of dMAX clocks required to complete a data transfer. For performance measurements purposes, the transfers are synchronized by using one synchronization event (only one synchronization event is used to synchronize the complete data transfer). The transfer duration is counted from the moment the event flag is cleared to the moment the CPU receives transfer-completed notification, as shown in Figure 4-2. Transfer E0 presented in Figure 4-2 consists of quantum transfers (E0(Q0), E0(Q1), E0(Q2) and E0(Q3)), and is synchronized to Event0.

The delay that the MAX module requires to respond to an event is described by latency. If the MAX module is not busy, a new transfer request is serviced quickly with very low latency. If the MAX module is in the middle of a transfer and another transfer request arrives, the new request will have to wait until completion of the quantum transfer currently in progress.

The transfer latency measured in dMAX clocks is shown in Figure 4-2. In this example, the Event0 is used to trigger a transfer E0. The MAX0 module is busy at the time when the Event0 arrives, so the new event must wait until the current quantum transfer E1(Q0) completes. The latency is measured (expressed in number of dMAX clocks) from the triggering to the moment when dMAX starts processing the new event (the moment when the event flag gets cleared by dMAX).

Figure 4-2. Transfer Latency and Duration Measured in Number of dMAX Clocks



The transfer duration measurements are collected by using the Real Time Interrupt (RTI) module. The RTI counter value is recorded immediately after triggering a dMAX transfer. The interrupt flag is polled by the CPU. Immediately after dMAX indicates transfer completion by setting the interrupt flag, the CPU records the new RTI counter value. A transfer duration is then calculated by subtracting the two recorded counter values. Measured values are averaged over several runs, and the average value is used as transfer duration

4.5 General Purpose Transfer Latency

The latency is not fixed and depends on overall dMAX loading. When the dMAX controller is busy, the new event must wait to be processed until dMAX completes the current quantum transfer. If several events in the same priority group arrive at the same time, the latency for the lowest priority event will be equal to sum of quantum transfer durations of all higher-priority pending events.

The dMAX controller transfers throttled by the EMIF asynchronous ready signal, or dMAX transfers with large QTSL values, increase dMAX latency by extended use of the MAX module resources. To improve dMAX latency, granularity with which the dMAX controller breaks a long transfer into a number of small quantum transfers can be increased, and a smaller quantum transfer size limit can be used. The granularity is controlled by the QTSL value within the event entry. Using small quantum transfers allows dMAX to evaluate the queue of events waiting to be processed more frequently. More frequent checking of the event queue reduces latency, but increases the overall transfer time. The overall transfer time is longer when the QTSL value is small, because dMAX moves a large number of small quantum transfers. In other words, using smaller QTSL values improves system latency, but worsens system throughput.

Using large QTSL values improves system throughput, but worsens system latency. The overall transfer time is shorter when the QTSL value is large because dMAX moves a small number of large quantum transfers.

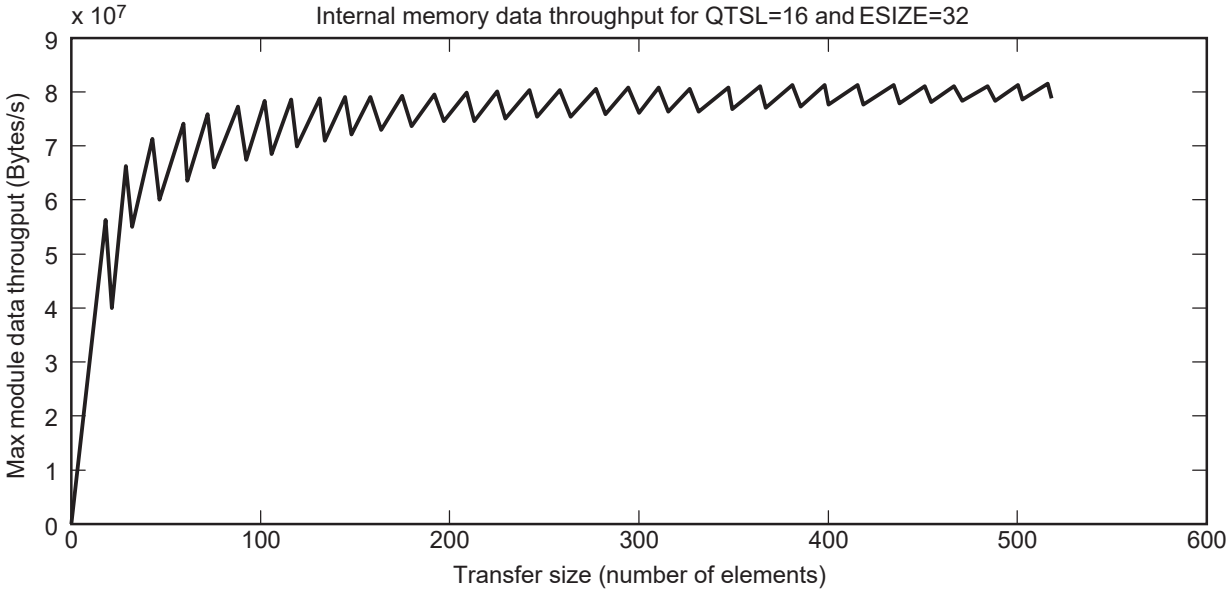
4.6 Transfers within the Internal Memory

The internal memory is clocked at the same frequency as the CPU. Therefore, when both source and destination are in the internal memory, dMAX transfers result in the highest throughput.

4.6.1 Copy of Sequential Data (SINDX0=1 and DINDX0=1)

The number of dMAX clock cycles required to copy a sequential data block from source in the internal data memory, to a different location in the internal data memory of the DSP is presented in Table 4-1. The cycle numbers are presented for different transfer counts, and different QTSL numbers. Figure 4-3 shows the MAX module data throughput for copy of a sequential block of data when both source and destination are in the internal memory (element size is 32-bit and QTSL is equal to 16).

Figure 4-3. MAX Module Data Throughput for Copy of a Sequential Block of Data when Both Source and Destination are in Internal Memory (SINDX0=1 and DINDX0=1)



Note that MAX module performance does not steadily increase with the burst size increase. Performance for dMAX can be described as an exponential saw tooth (Figure 4-3). The performance reaches peaks when the first dimension of the transfer counter (COUNT0) is a multiple of the QTSL value. If the COUNT0 value is not a multiple of QTSL, the last quantum transfer will not completely utilize the MAX bandwidth. When this is the case, all the quantum transfers except the last one moves QTSL number of elements, while the last quantum transfer moves the remaining elements of the transfer - which is smaller than the QTSL value. To clarify, the MAX module performance reaches the local maximums for transfers that move QTSL number of elements in each quantum transfer. The MAX module performance reaches the local minimums for transfers that move only one element in the last quantum transfer.

For example, it takes ~156 dMAX clocks (QTSL is set to 16) to transfer a burst of 16 sequential 32-bit elements from source to destination in the internal memory. In this case, the transfer size is equal to the QTSL value, and the MAX module will move all the elements in one quantum transfer. Increasing the transfer size by four elements (from 16 to 20 elements) requires splitting the transfer into two quantum transfers (the first quantum transfer of 16 elements and the second quantum transfer of four elements), and this increases total transfer duration to ~250 of dMAX clocks. Increasing the transfer size by another 12 elements (from 20 to 32 elements) still requires splitting the transfer into two quantum transfers (now both quantum transfers move 16 elements), and this increases total transfer duration to ~273 of dMAX clocks.

Table 4-1. MAX Module Performance for Copy of a Block of Sequential Elements when both Source and Destination are in Internal Memory

No. dMAX Clocks		Transfer Size (COUNT0) in Number of Elements																			
		4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	128	256	512	1024
ESIZE = 8 bits																					
QTSL	1	379	714	1052	1385	1725	2057	2397	2730	3070	3403	3740	4075	4411	4748	5083	5417	10796	21546	43054	86057
	4	130	215	301	305	475	561	645	730	821	901	990	1075	1160	1245	1335	1421	2795	5545	11051	22061
	8	130	131	217	218	308	304	394	395	482	483	572	570	658	659	747	746	1451	2858	5676	11306
	16	130	131	131	132	223	224	224	225	314	316	316	317	408	408	408	409	778	1513	2989	5929
ESIZE = 16 bits																					
QTSL	1	379	714	1052	1385	1725	2057	2397	2730	3070	3403	3740	4075	4411	4748	5083	5417	10796	21546	43054	86057
	4	130	218	309	395	483	570	660	745	837	923	1010	1098	1188	1273	1365	1451	2858	5675	11307	22571
	8	130	132	225	225	316	317	407	409	500	501	592	593	684	685	775	777	1518	2985	5932	11817
	16	130	132	140	143	232	233	242	243	332	335	341	344	433	433	441	440	844	1640	3244	6441
ESIZE = 32 bits																					
QTSL	1	379	715	1052	1385	1725	2057	2397	2730	3070	3403	3740	4075	4411	4748	5083	5417	10796	21546	43054	86057
	4	138	225	318	409	503	593	689	777	871	961	1056	1145	1243	1329	1423	1513	2989	5929	11822	23593
	8	138	143	235	243	334	339	435	441	536	543	637	640	735	741	837	841	1641	3240	6442	12840
	16	138	143	151	156	250	256	270	273	368	374	384	392	482	490	501	507	974	1900	3754	7465

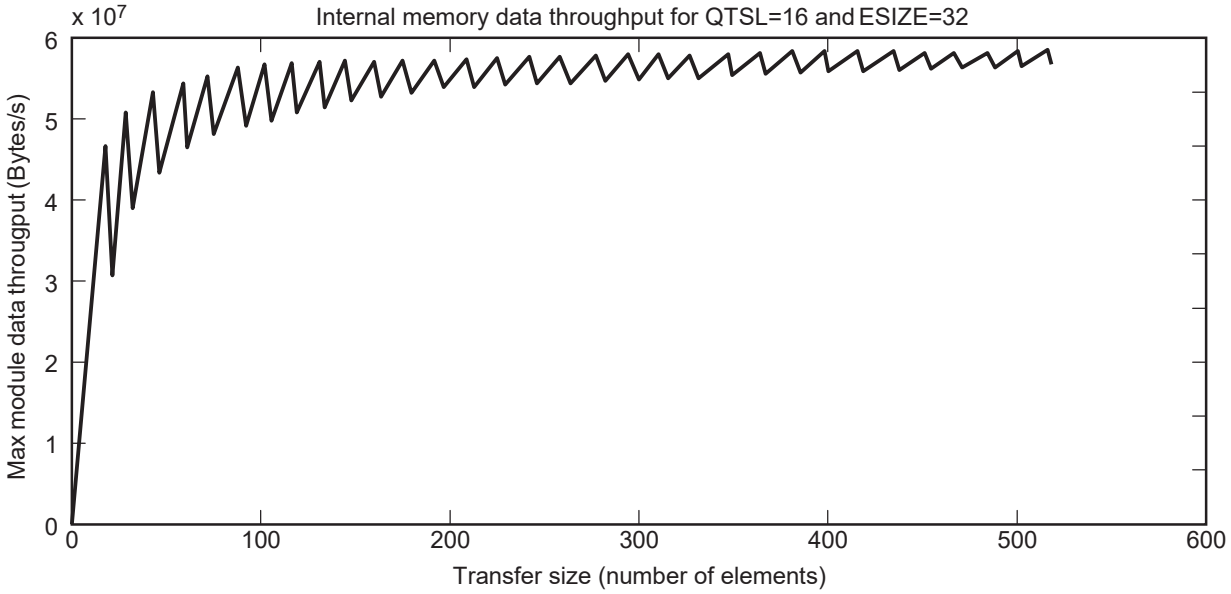
Transfers within the Internal Memory

4.6.2 Sorting of Sequential Data (SINDX0≠1 and DINDX0=1, or SINDX0=1 and DINDX0≠1)

The number of dMAX clock cycles required to sort sequential data blocks to non-sequential locations, when both source and destination locations are in the internal DSP memory, is presented in Table 4-2. The cycle numbers are presented for different transfer counts, and different QTSL numbers.

Figure 4-4 shows the MAX module data throughput for sorting a sequential block of data when both source and destination are in the internal memory (element size is 32-bit and QTSL is equal to 16).

Figure 4-4. MAX Module Data Throughput for Sorting of Sequential Block of Data when both Source and Destination are in Internal Memory (SINDX0≠1, DINDX0=1)



Note that MAX module performance does not steadily increase with the burst size increase. Performance for dMAX can be described as an exponential saw tooth. The performance reaches peaks when the first dimension of the transfer counter (COUNT0) is a multiple of the QTSL value. If COUNT0 value is not a multiple of QTSL, the last quantum transfer will not completely utilize the MAX bandwidth. When this is the case, all quantum transfers except the last one have size equal to QTSL, while the last quantum transfer moves the remaining elements of the transfer - which is smaller than QTSL value. To restate this, the MAX module performance reaches the local maximums for transfers that move QTSL number of elements in each quantum transfer. The MAX module performance reaches the local minimums for transfers that move only one element in the last quantum transfer.

Table 4-2. MAX Module Performance for Sorting of Sequential Elements when both Source and Destination are in Internal Memory

No. dMAX Clocks		Transfer Size (COUNT0) in Number of Elements																			
		4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	128	256	512	1024
ESIZE = 8 bits																					
QTSL	1	409	768	1138	1496	1865	2224	2589	2952	3318	3680	4046	4408	4776	5136	5502	5864	11690	23336	46635	93224
	4	150	250	359	460	569	673	779	883	989	1091	1199	1302	1409	1509	1619	1724	3404	6762	13484	26923
	8	150	164	269	283	390	401	511	520	630	642	749	763	870	881	991	1000	1965	3880	7725	15400
	16	150	164	176	192	300	311	326	340	448	461	476	489	596	609	623	636	1233	2425	4813	9577
ESIZE = 16 bits																					
QTSL	1	410	768	1138	1496	1866	2224	2594	2952	3322	3680	4050	4408	4778	5136	5506	5864	11694	23336	46638	93224
	4	150	254	360	465	575	676	785	889	1000	1101	1210	1315	1420	1525	1635	1739	3435	6825	13610	27177
	8	150	165	274	288	393	410	515	532	638	654	762	774	881	897	1002	1019	1997	3949	7854	15658
	16	150	165	185	195	303	320	337	351	455	473	491	504	608	626	643	657	1271	2493	4943	9837
ESIZE = 32 bits																					
QTSL	1	410	771	1138	1500	1866	2226	2594	2956	3322	3683	4050	4407	4778	5136	5506	5864	11694	23339	46638	93227
	4	150	257	369	475	582	690	800	905	1017	1123	1231	1337	1448	1553	1665	1771	3498	6955	13868	27691
	8	150	166	278	295	403	422	528	546	655	671	784	800	906	922	1034	1050	2062	4073	8110	16170
	16	150	166	189	205	312	331	351	363	473	488	512	527	634	653	673	685	1335	2617	5198	10345

Transfers within the Internal Memory

4.6.3 Sorting of Non-Sequential Data (SINDX0≠1 and DINDX0≠1)

The number of dMAX clock cycles required to sort non-sequential data when both source and destination locations are in the internal data memory of the DSP is presented in Table 4-3. The cycle numbers are presented for different transfer counts, and different QTSL numbers.

Table 4-3. MAX Module Performance for Sorting of Non-Sequential Elements when both Source and Destination are in Internal Memory

No. dMAX Clocks		Transfer Size (COUNT0) in Number of Elements																			
		4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	128	256	512	1024
ESIZE = 8 bits																					
QTSL	1	443	834	1231	1629	2026	2417	2816	3211	3608	4004	4404	4794	5191	5589	5985	6377	12715	25389	50731	101419
	4	182	315	458	589	732	863	1006	1137	1280	1411	1554	1685	1828	1959	2102	2233	4432	8809	17583	35113
	8	183	222	359	402	543	585	721	764	905	947	1083	1126	1267	1309	1445	1489	2940	5832	11628	23208
	16	183	222	271	307	448	488	536	577	718	755	806	844	985	1022	1073	1111	2181	4315	8589	17131
ESIZE = 16 bits																					
QTSL	1	443	834	1231	1629	2026	2417	2816	3211	3608	4004	4404	4794	5191	5589	5985	6377	12715	25389	50731	101419
	4	183	315	455	589	728	863	1002	1137	1279	1411	1552	1685	1826	1959	2099	2233	4430	8809	17581	35113
	8	183	223	365	403	544	587	727	769	906	948	1085	1128	1268	1310	1448	1489	2940	5836	11629	23212
	16	183	223	271	312	448	493	537	576	717	760	805	843	984	1027	1072	1110	2180	4314	8588	17130
ESIZE = 32 bits																					
QTSL	1	443	833	1231	1628	2026	2416	2817	3212	3608	4005	4403	4793	5191	5588	5986	6376	12717	25388	50731	101419
	4	183	314	458	588	732	862	1006	1136	1280	1410	1554	1684	1828	1958	2102	2232	4431	8808	17583	35112
	8	183	224	361	402	545	587	722	764	907	949	1084	1126	1269	1311	1446	1488	2942	5832	11630	23208
	16	183	224	271	311	449	492	537	575	716	759	804	842	983	1026	1071	1109	2179	4313	8587	17129

Transfers within the Internal Memory

4.7 Transfers Between the Internal Memory and McASP

To evaluate data throughput between the McASP and the internal memory, two cases of transfers are considered:

1. In the first case, the burst of sequential data is moved between the McASP DMA port and sequential locations in the internal memory; the dMAX throughput is higher. In this case, CPU involvement is required to interleave/deinterleave data. For McASP receive, the CPU must de-interleave data since subsequent elements transferred from the McASP DMA port belong to different serializers. With a McASP transmit, the CPU must interleave data since consecutive elements to be sent to the McASP DMA port belong to different serializers.
2. In the second case, a burst of sequential data is moved from the McASP DMA port to non-sequential locations in the internal memory for McASP receive (or data from non-sequential locations in the internal memory is moved to the McASP DMA port for McASP transmit). In this case the dMAX controller throughput is lower. For a McASP receive, dMAX moves and de-interleaves data from the McASP DMA port to a set of receive buffers (since subsequent elements transferred from the McASP DMA port belong to different serializers). For a McASP transmit, dMAX interleaves data from a set of transmit buffers and moves the data to the McASP DMA port (since consecutive elements to be sent to the McASP DMA port belong to different serializers).

MAX module performance is presented in the number of dMAX clocks required to complete a transfer. The data presented can be used to determine the loading of dMAX when servicing the McASP peripheral and/or the maximum sampling rate in the system.

For example, if dMAX is performing data interleave/de-interleave, the tables from Section 4.6.2 should be used. In this case, each serializer pin has an associated buffer in the DSP memory, and dMAX moves the data between the buffer and the McASP peripheral. If one serializer pin is used per each clock domain (total of six pins), the total number of dMAX clocks required to move the McASP data for all six clock domains can be calculated from Table 4-4 and Table 4-5. It takes ~137 dMAX clocks to receive a sample from the McASP DMA port, and it takes ~142 dMAX clocks to send a sample to the McASP DMA port. Assuming that there are three receive and three transmit McASP pins, it will take ~837 dMAX clocks to service all six clock domains (one serializer pin per clock domain). If dMAX runs at 150MHz, it takes ~5.58s to execute 837 dMAX clocks.

4.7.1 Copy of Sequential Data (SINDX0=1 and DINDX0=1)

In this case only one receive buffer, and one transmit buffer is used for all serializer pins from the same clock domain. The dMAX controller achieves higher throughput since it is bursting on both ends (source and destination). The data in the buffers is interleaved, so before processing the CPU must perform data sorting.

4.7.1.1 McASP Receive

The number of dMAX clock cycles required to copy sequential data from McASP DMA port to sequential destination locations in the internal data memory is presented in Table 4-4. The cycle numbers are presented for different transfer counts, and different QTSL numbers.

Since the dMAX controller bursts on both ends (source and destination) the throughput is higher. However, the CPU must de-interleave the data before processing, since subsequent elements transferred from the McASP DMA port belong to different serializers.

Table 4-4. MAX Module Performance for Copy of Block of Sequential Elements from McASP DMA Port Source to Destination in Internal Memory

No. dMAX Clocks		Transfer Size (COUNT0) in Number of Elements															
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
ESIZE = 32 bits																	
QTSL	1	130	208	293	378	462	540	625	705	794	872	957	1038	1126	1204	1289	1369
	4	130	130	131	132	222	223	222	223	310	314	314	314	402	405	406	405
	8	130	130	131	132	138	139	143	140	231	231	231	231	238	239	239	241
	16	130	130	131	132	138	139	143	140	147	147	148	148	155	156	156	156

4.7.1.2 McASP Transmit

The number of dMAX clock cycles required to copy sequential data from source in the internal data memory to McASP DMA port is presented in Table 4-5. The cycle numbers are presented for different transfer counts, and different quantum transfer size limit (QTSL) numbers.

Since the dMAX controller bursts on both ends (source and destination) the throughput is higher, but before start the CPU must prepare and order data in the internal memory so that each subsequent element, within a block to be transferred, belongs to different serializer

Table 4-5. MAX Module Performance for Copy of a Block of Sequential Elements from Source in the Internal Memory to McASP DMA Port Destination

No. dMAX Clocks		Transfer Size (COUNT0) in Number of Elements															
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
ESIZE = 32 bits																	
QTSL	1	130	215	304	385	476	560	645	729	821	905	990	1074	1163	1249	1336	1420
	4	130	130	137	137	228	226	232	230	320	321	327	323	417	413	419	419
	8	130	130	137	137	137	138	145	146	235	236	238	238	242	240	246	247
	16	130	130	137	137	137	138	145	146	147	147	153	154	154	155	162	163

4.7.2 Sorting of Sequential Data (SINDX0=1 and DINDX0≠1 or SINDX0≠1 and DINDX0=1)

The dMAX controller achieves lower throughput than discussed in Section 4.7.1 since it is bursting on the McASP DMA port end and it is sorting data on the memory end. In this case each serializer pin has associated buffer in the DSP memory, and dMAX moves the data between the buffer and the McASP peripheral

4.7.2.1 McASP Receive

The number of dMAX clock cycles required to sort sequential data from source in the McASP to non-sequential locations in the internal data memory of the DSP is presented in Table 4-6. The cycle numbers are presented for different transfer counts, and different QTSL numbers.

Table 4-6. MAX Module Performance for Sorting of Sequential Elements from McASP DMA Port Source to Non-Sequential Destination in Internal Memory

No. dMAX Clocks		Transfer Size (COUNT0) in Number of Elements															
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
ESIZE = 32 bits																	
QTSL	1	137	221	312	403	494	585	677	762	853	945	1036	1121	1212	1303	1395	1484
	4	137	140	147	150	241	248	255	256	353	352	359	365	457	463	470	469
	8	137	140	147	150	156	157	164	169	261	263	274	274	282	283	288	294
	16	137	140	147	150	156	157	164	169	173	178	184	183	190	194	198	204

4.7.2.2 McASP Transmit

The number of dMAX clock cycles required to sort non-sequential data from source in the internal DSP memory to McASP DMA port destinations is presented in Table 4-7. The cycle numbers are presented for different transfer counts, and different QTSL numbers.

Table 4-7. MAX Module Performance for Sorting of Non-Sequential Data from Source in the Internal Memory to McASP DMA Port Destination

No. dMAX Clocks		Transfer Size (COUNT0) in Number of Elements															
		1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
ESIZE = 32 bits																	
QTSL	1	137	228	325	416	515	606	703	795	892	983	1078	1172	1264	1359	1452	1544
	4	142	150	159	166	260	270	280	287	384	392	403	411	508	515	528	535
	8	139	150	159	166	176	183	191	199	294	307	314	321	334	341	348	354
	16	142	150	160	166	176	183	191	199	209	216	225	232	242	249	262	264

4.8 Transfers Between Internal Memory and EMIF SDRAM

Throughput to the SDRAM interface depends on the SDRAM setting and on clock ratio between dMAX and the external memory interface (EMIF). In the following sections, the assumption is that ratio between dMAX and the EMIF clock domains is 1.5.

4.8.1 Copy of Sequential Data (SINDX0=1 and DINDX0=1)

The number of dMAX clock cycles required to copy a block of sequential data from source (SINDX0 = 1) in the internal data memory to different location in the SDRAM memory (DINDX0 = 1) when EMIF is 32-bit wide is presented in Table 4-8. The cycle numbers are presented for different transfer counts, and different QTSL numbers. Note that for writes to the SDRAM performance numbers look very similar to performance numbers from Table 4-1 describing writes to the internal memory. Writes to the SDRAM and to the internal memory have similar performance due to the internal FIFOs inside bus bridges.

The number of dMAX clock cycles required to copy a block of sequential data from source (SINDX0 = 1) in the SDRAM memory to sequential locations in the internal memory (DINDX0 = 1) when EMIF is 32-bit wide is presented in Table 4-9. The cycle numbers are presented for different transfer counts, and different QTSL numbers.

The number of dMAX clock cycles required to copy a block of sequential data from source (SINDX0 = 1) in the internal data memory to different location in the SDRAM memory (DINDX0 = 1) when EMIF is 16-bit wide is presented in Table 4-10. The cycle numbers are presented for different transfer counts, and different QTSL numbers.

The number of dMAX clock cycles required to copy a block of sequential data from source (SINDX0 = 1) in the SDRAM memory to different location in the internal memory (DINDX0 = 1) when EMIF is 16-bit wide is presented in Table 4-11. The cycle numbers are presented for different transfer counts, and different QTSL numbers.

Note that the MAX module performance doesn't steadily increase with the burst size increase. dMAX performance can be described as an exponential sawtooth. The performance reaches peaks when the first dimension of the transfer counter (COUNT0) is a multiple of the QTSL value. In case COUNT0 value is not a multiple of the QTSL, the last quantum transfer will not completely utilize the MAX module bandwidth. In this case, all quantum transfers except the last one have size equal to QTSL, while the last quantum transfer moves the remaining elements of the transfer - which is smaller than the QTSL value. In other words, the MAX module performance reaches the local maximums for transfers that move the QTSL number of elements in each quantum transfer. The MAX module performance reaches the local minimums for transfers that move only one element in the last quantum transfer.

Table 4-8. MAX Module Performance for Copy of a Block of Sequential Elements when Source is in Internal Memory and Destination is in the SDRAM (EMIF is 32-bit wide)

No. dMAX Clocks		Transfer Size (COUNT0) in Number of Elements																			
		4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	128	256	512	1024
ESIZE = 8 bits																					
QTSL	1	379	714	1052	1385	1725	2057	2397	2730	3070	3403	3740	4075	4411	4748	5083	5417	10796	21546	43054	86057
	4	130	215	301	305	475	561	645	730	821	901	990	1075	1160	1245	1335	1421	2795	5545	11051	22061
	8	130	130	216	217	307	307	392	397	483	483	568	570	659	659	744	748	1454	2859	5678	11308
	16	130	131	131	132	223	224	224	225	314	316	316	317	408	408	408	409	778	1513	2989	5929
ESIZE = 16 bits																					
QTSL	1	379	714	1052	1385	1725	2057	2397	2730	3070	3403	3740	4075	4411	4748	5083	5417	10796	21546	43054	86057
	4	130	218	309	395	483	570	660	745	837	923	1010	1098	1188	1273	1365	1451	2858	5675	11307	22571
	8	130	138	223	225	315	317	406	410	498	505	591	594	683	685	774	777	1518	2989	5934	11817
	16	130	138	138	139	232	234	239	241	331	337	339	340	430	437	439	441	841	1641	3244	6442
ESIZE = 32 bits																					
QTSL	1	378	716	1054	1387	1726	2057	2396	2728	3067	3400	3738	4076	4412	4747	5086	5417	10796	21547	43050	86057
	4	137	226	322	413	506	593	690	781	874	962	1058	1147	1242	1329	1426	1516	2990	5932	11822	23594
	8	137	139	238	240	338	343	438	440	538	540	638	642	738	742	838	842	1642	3242	6443	12842
	16	137	139	154	156	254	257	266	273	371	375	383	392	485	487	502	504	973	1897	3757	7466

Table 4-9. MAX Module Performance for Copy of a Block of Sequential Elements when Source is in SDRAM and Destination is in Internal Memory (EMIF is 32-bit wide)

No. dMAX Clocks		Transfer Size (COUNT0) in Number of Elements																			
		4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	128	256	512	1024
ESIZE = 8 bits																					
QTSL	1	449	850	1254	1652	2056	2458	2863	3260	3664	4067	4470	4867	5272	5674	6078	6477	12917	25790	51538	103030
	4	150	247	356	455	560	657	761	865	969	1067	1171	1269	1373	1476	1578	1679	3314	6567	13112	26180
	8	150	150	254	254	358	358	464	463	568	568	674	674	779	778	884	883	1726	3404	6766	13494
	16	150	150	151	154	259	260	262	263	369	368	374	374	480	479	485	485	930	1817	3594	7145
ESIZE = 16 bits																					
QTSL	1	449	851	1254	1652	2056	2459	2863	3260	3664	4067	4471	4867	5272	5674	6078	6476	12915	25768	51535	103030
	4	150	254	362	463	572	673	781	883	992	1094	1202	1303	1412	1513	1621	1724	3407	6763	13494	26941
	8	150	153	262	263	372	374	482	485	591	595	704	706	813	818	926	928	1817	3593	7145	14254
	16	150	153	163	163	270	274	284	284	285	391	394	404	509	514	524	525	1006	1965	3885	7725
ESIZE = 32 bits																					
QTSL	1	449	851	1254	1652	2056	2459	2863	3260	3664	4067	4471	4867	5272	5674	6078	6476	12915	25768	51535	103030
	4	157	262	376	485	599	707	821	928	1043	1151	1265	1372	1487	1595	1710	1817	3594	7146	14256	28466
	8	157	164	277	285	397	404	517	525	637	645	758	765	877	885	997	1005	1966	3885	7727	15411
	16	157	165	174	182	296	303	318	326	437	443	456	465	577	584	598	606	1171	2298	4555	9066
ESIZE = 64 bits																					

Table 4-10. MAX Module Performance for Copy of a Block of Sequential Elements when Source is in Internal Memory and Destination is in the SDRAM (EMIF is 16-bit wide)

No. dMAX Clocks		Transfer Size (COUNT0) in Number of Elements																			
		4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	128	256	512	1024
ESIZE = 8 bits																					
QTSL	1	378	716	1053	1387	1726	2057	2396	2728	3067	3403	3738	4076	4413	4747	5086	5417	10796	21547	43050	86057
	4	130	215	300	384	475	560	644	730	820	904	990	1075	1160	1244	1335	1420	2794	5544	11822	22060
	8	130	130	220	219	307	307	392	396	483	483	568	570	659	659	711	747	1454	2860	6445	11307
	16	130	130	137	137	222	223	227	229	314	315	317	319	406	407	412	409	782	1517	3758	5933
ESIZE = 16 bits																					
QTSL	1	379	714	1052	1385	1725	2057	2397	2730	3070	3403	3740	4075	4411	4748	5083	5417	10796	21546	43054	86057
	4	130	218	309	395	483	570	660	745	837	923	1010	1098	1188	1273	1365	1451	2858	5675	11307	22571
	8	130	137	222	225	314	317	406	410	498	505	591	594	683	685	774	777	1518	2989	5934	11817
	16	130	137	138	139	230	237	239	241	331	337	339	340	430	437	439	441	841	1641	3244	6442
ESIZE = 32 bits																					
QTSL	1	378	716	1054	1387	1726	2057	2396	2728	3067	3400	3738	4076	4412	4747	5086	5417	10796	21547	43050	86057
	4	137	225	322	413	506	594	690	777	874	961	1058	1148	1242	1332	1426	1517	2990	5930	11822	23596
	8	137	139	238	240	338	341	438	440	538	540	638	642	738	742	838	842	1642	3242	6443	12842
	16	137	139	154	156	254	257	267	274	371	375	384	392	485	489	502	504	974	1899	3755	7466

Table 4-11. MAX Module Performance for Copy of a Block of Sequential Elements when Source is in SDRAM and Destination is in Internal Memory (EMIF is 16-bit wide)

No. dMAX Clocks		Transfer Size (COUNT0) in Number of Elements																			
		4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	128	256	512	1024
ESIZE = 8 bits																					
QTSL	1	455	859	1269	1677	2088	2493	2903	3307	3717	4127	4535	4941	5351	5755	6165	6576	13108	26172	52313	104580
	4	150	254	358	462	568	670	780	881	990	1091	1198	1302	1407	1510	1621	1721	3405	6761	13486	26929
	8	150	150	260	260	365	369	475	476	583	586	690	691	800	801	904	910	1777	3501	6960	13875
	16	150	150	156	158	267	268	273	275	384	386	390	392	497	503	504	510	979	1913	3787	7530
ESIZE = 16 bits																					
QTSL	1	455	859	1269	1677	2088	2493	2903	3307	3717	4127	4535	4941	5351	5755	6165	6576	13108	26172	52313	104580
	4	156	260	371	476	587	693	801	910	1021	1126	1236	1341	1451	1556	1666	1772	3501	6956	13877	27712
	8	156	158	270	274	388	393	504	510	622	626	738	744	856	861	972	977	1914	3785	7532	15022
	16	156	158	170	173	288	291	303	306	420	423	436	438	552	555	568	570	1103	2154	4270	8493
ESIZE = 32 bits																					
QTSL	1	455	859	1270	1678	2087	2492	2903	3307	3717	4127	4535	4941	5351	5755	6165	6576	13108	26172	52313	104580
	4	163	275	395	510	629	743	864	977	1097	1212	1331	1446	1565	1680	1800	1915	3787	7530	15024	30005
	8	163	173	295	305	427	437	560	570	691	702	824	833	955	966	1088	1097	2159	4265	8500	16946
	16	157	173	194	209	327	341	358	372	490	505	524	536	654	668	689	700	1363	2681	5323	10604

4.8.2 Sorting of Sequential Data (SINDX0≠1 and DINDX0=1 or SINDX0 = 1 and DINDX0≠1)

The number of dMAX clock cycles required to sort non-sequential data from source (SINDX0 ≠ 1) in the internal data memory to sequential locations in the SDRAM (DINDX0 = 1) is presented in Table 4-12 for 32-bit EMIF and in Table 4-14 for 16-bit EMIF. The cycle numbers are presented for different transfer counts, and different QTSL numbers.

The number of dMAX clock cycles required to sort sequential data from source (SINDX0 = 1) in the SDRAM to non-sequential locations in the internal data memory (DINDX0 ≠ 1) is presented in Table 4-13 for 32-bit EMIF and in Table 4-15 for 16-bit EMIF. The cycle numbers are presented for different transfer counts, and different QTSL numbers.

Note that dMAX module performance doesn't steadily increase with the burst size increase. dMAX performance can be described as an exponential sawtooth. The performance reaches peaks when the first dimension of the transfer counter (COUNT0) is a multiple of the QTSL value. If the COUNT0 value is not a multiple of the QTSL, the last quantum transfer will not completely utilize the MAX module bandwidth. In this case, all quantum transfers except the last one have size equal to the QTSL, while the last quantum transfer moves the remaining elements of the transfer - which is smaller than the QTSL value. The MAX module performance reaches the local maximums for transfers that move the QTSL number of elements in each quantum transfer. The MAX module performance reaches the local minimums for transfers that move only one element in the last quantum transfer.

Table 4-12. MAX Module Performance for Sorting Block of Non-Sequential Elements from Source in Internal Memory to Sequential Locations at Destination in SDRAM (EMIF is 32-bit wide)

No. dMAX Clocks	Transfer Size (COUNT0) in Number of Elements																			
	4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	128	256	512	1024
ESIZE = 8 bits																				
QTSL 1	410	776	1146	1513	1882	2250	2618	2985	3354	3724	4090	4460	4826	5194	5562	5931	11821	23597	47149	94249
QTSL 4	163	280	398	516	634	752	870	988	1106	1224	1342	1460	1578	1696	1814	1932	3820	7596	15148	30252
QTSL 8	163	190	313	340	458	491	609	635	760	787	905	938	1056	1082	1207	1234	2430	4810	9582	19114
QTSL 16	163	190	224	252	373	401	434	461	583	611	644	672	793	821	854	881	1727	3401	6767	13481
ESIZE = 16 bits																				
QTSL 1	410	780	1146	1512	1882	2251	2618	2984	3354	3724	4090	4460	4826	5194	5562	5931	11821	23597	47149	94249
QTSL 4	163	280	403	519	638	755	880	997	1114	1231	1356	1473	1590	1707	1832	1949	3852	7661	15277	30509
QTSL 8	163	196	314	346	465	497	615	648	766	799	917	950	1068	1101	1220	1252	2460	4876	9708	19372
QTSL 16	163	196	229	256	379	407	440	472	591	624	654	684	805	834	867	900	1757	3468	6893	13740
ESIZE = 32 bits																				
QTSL 1	410	780	1146	1512	1882	2251	2618	2984	3354	3724	4090	4460	4826	5194	5562	5931	11821	23597	47149	94249
QTSL 4	166	2876	410	528	652	770	894	1012	1136	1252	1378	1496	1620	1736	1862	1978	3915	7786	15531	31019
QTSL 8	166	196	320	354	478	510	628	662	785	819	943	975	1093	1127	1250	1284	2527	5004	9967	19884
QTSL 16	166	196	236	263	386	419	458	485	610	643	676	710	834	861	901	928	1818	3592	7146	14248

Table 4-13. MAX Module Performance for Sorting of Block of Sequential Locations from Source in SDRAM to Non-Sequential Destination Locations in Internal Memory (EMIF is 32-bit wide)

No. dMAX Clocks	Transfer Size (COUNT0) in Number of Elements																			
	4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	128	256	512	1024
ESIZE = 8 bits																				
QTSL 1	475	907	1340	1770	2205	2634	3070	3500	3934	4365	4796	5228	5661	6089	6525	6954	13874	27709	55375	110708
QTSL 4	169	287	411	528	652	775	898	1017	1140	1256	1383	1504	1627	1745	1869	1985	3933	7819	15605	31163
QTSL 8	169	181	305	317	442	453	579	591	718	729	855	867	993	1005	1131	1143	2252	4455	8878	17714
QTSL 16	169	181	196	209	334	347	367	375	504	513	532	546	669	683	704	710	1387	2726	5419	10795
ESIZE = 16 bits																				
QTSL 1	476	908	1340	1770	2205	2634	3070	3500	3934	4365	4796	5228	5661	6089	6525	6954	13874	27709	55375	110708
QTSL 4	169	293	417	535	660	784	908	1029	1154	1274	1399	1523	1647	1765	1890	2014	3983	7918	15794	31544
QTSL 8	169	183	310	326	451	465	592	607	733	747	873	890	1016	1029	1155	1170	2302	4554	9071	18097
QTSL 16	169	183	204	218	341	357	376	389	517	530	551	563	689	706	722	740	1438	2827	5614	11181
ESIZE = 32 bits																				
QTSL 1	476	908	1340	1770	2205	2634	3070	3500	3934	4365	4796	5228	5661	6089	6525	6954	13874	27709	55375	110708
QTSL 4	171	299	424	549	678	801	931	1055	1180	1305	1434	1557	1686	1810	1936	2060	4080	8110	16179	32320
QTSL 8	172	190	319	334	464	484	611	629	758	778	905	922	1053	1072	1199	1217	2395	4744	9451	18866
QTSL 16	171	190	210	231	357	374	396	416	541	559	581	600	727	745	766	787	1533	3019	5997	11950

Table 4-14. MAX Module Performance for Sorting Block of Non-Sequential Elements from Source in Internal Memory to Sequential Locations at Destination in SDRAM (EMIF is 16-bit wide)

No. dMAX Clocks		Transfer Size (COUNT0) in Number of Elements																			
		4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	128	256	512	1024
ESIZE = 8 bits																					
QTSL	1	410	776	1146	1513	1882	2250	2618	2985	3354	3724	4090	4460	4826	5194	5562	5931	11821	23597	47149	94249
	4	163	280	398	516	634	752	870	988	1106	1224	1342	1460	1578	1696	1814	1932	3820	7596	15148	30252
	8	163	189	313	340	458	491	609	635	760	787	905	938	1056	1082	1207	1234	2430	4810	9582	19114
	16	163	190	222	252	373	401	434	461	583	611	644	672	793	821	854	881	1727	3401	6767	13481
ESIZE = 16 bits																					
QTSL	1	410	780	1146	1512	1882	2251	2618	2984	3354	3724	4090	4460	4826	5194	5562	5931	11821	23597	47149	94249
	4	163	281	401	519	638	755	880	997	1114	1231	1356	1473	1590	1707	1852	1949	3852	7661	15277	30509
	8	163	196	313	346	465	497	615	648	766	799	917	950	1068	1101	1220	1252	2460	4876	9708	19372
	16	163	196	229	256	379	407	440	472	591	624	654	684	805	834	867	900	1757	3468	6893	13740
ESIZE = 32 bits																					
QTSL	1	410	780	1146	1512	1882	2251	2618	2984	3354	3724	4090	4460	4826	5194	5562	5931	11821	23597	47149	94249
	4	165	287	410	528	652	770	894	1012	1136	1252	1378	1496	1620	1736	1862	1978	3915	7786	15531	31019
	8	166	196	320	354	478	510	628	662	785	819	943	975	1093	1127	1250	1284	2527	5004	9967	19884
	16	165	196	236	263	386	419	458	485	610	643	676	710	834	861	901	928	1819	3592	7146	14248

Table 4-15. MAX Module Performance for Sorting of Block of Sequential Locations from Source in the SDRAM to Non-Sequential Destination Locations in Internal Memory (EMIF is 16-bit wide)

No. dMAX Clocks		Transfer Size (COUNT0) in Number of Elements																			
		4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	128	256	512	1024
ESIZE = 8 bits																					
QTSL	1	484	917	1360	1795	2235	2672	3110	3545	3988	4423	4862	5299	5738	6174	6616	7052	14067	28095	56183	112321
	4	169	291	417	535	659	784	908	1027	1154	1275	1398	1520	1647	1765	1889	2014	3983	7917	15799	31546
	8	169	183	307	322	450	465	589	603	728	747	871	885	1010	1029	1154	1166	2297	4551	9068	18094
	16	169	183	204	216	339	355	373	388	512	530	546	564	688	702	722	734	1432	2823	5609	11179
ESIZE = 16 bits																					
QTSL	1	484	917	1360	1795	2235	2672	3110	3545	3988	4423	4862	5299	5738	6174	6616	7052	14067	28095	56183	112321
	4	171	297	424	547	678	801	929	1054	1180	1303	1434	1557	1683	1809	1936	2059	4080	8110	16180	32318
	8	171	190	320	336	464	485	611	629	755	779	904	923	1052	1072	1198	1217	2397	4745	9457	18865
	16	171	190	212	231	358	375	399	417	543	561	583	602	728	746	768	788	1534	3020	5998	11950
ESIZE = 32 bits																					
QTSL	1	483	918	1360	1795	2235	2672	3110	3545	3988	4423	4862	5299	5738	6174	6616	7052	14067	28095	56183	112321
	4	176	310	443	574	706	835	973	1103	1234	1364	1500	1627	1763	1894	2027	2155	4273	8496	16951	33852
	8	177	203	334	360	494	518	654	676	812	836	971	996	1131	1155	1290	1311	2585	5128	10221	20391
	16	177	203	230	254	386	412	440	462	595	622	650	673	806	831	859	882	1725	3403	6765	13483

4.9 One-Dimensional Burst Transfer Performance

To evaluate data throughput for One-Dimensional Burst transfers, four cases of transfers are considered:

- In the first case, the burst of sequential data is moved between sequential locations in the internal memory; the dMAX throughput is higher.
- In the second case, the burst of sequential data is moved between sequential locations in the internal memory to sequential locations in the external memory.
- In the third case, the burst of sequential data is moved between sequential locations in the external memory to sequential locations in the internal memory.
- In the fourth case, the burst of sequential data is moved between sequential locations in the external memory.

MAX module performance is presented in the number of dMAX clocks required to complete a transfer. The data presented can be used to determine the loading of dMAX when performing 1DN transfers using different NBURSTS, BURSTLEN and different combination of source and destination locations.

Throughput to the SDRAM interface depends on the SDRAM setting and on clock ratio between dMAX and the external memory interface (EMIF). In the following sections, the assumption is that ratio between dMAX and the EMIF clock domains is 1.5. For transfer cases that involve SDRAM, throughput numbers are provided for the case when EMIF is 16-bit wide and for the case when EMIF is 32-bit wide.

Table 4-16. MAX Module Performance for Moving Sequential Data - Both Source and Destination are in Internal Memory

No. dMAX Clocks		Transfer Size (CNT) in Number of Bytes									
		4	16	32	64	128	256	384	512	768	1024
Burst Length = 4 bytes											
NUMBURST	1	57	216	428	849	1700	3397	5091	6785	10180	13572
	4	57	107	212	419	835	1668	2500	3331	4997	6659
	8	57	107	177	345	692	1380	2067	2757	4132	5508
	16	57	107	177	310	618	1234	1850	2466	3698	4930
Burst Length = 16 bytes											
NUMBURST	1	57	62	121	240	475	947	1420	1892	2837	3779
	4	57	62	86	132	261	513	772	1028	1539	2052
	8	58	62	86	132	223	443	664	884	1325	1763
	16	58	62	86	132	223	408	629	811	1217	1619
Burst Length = 32 bytes											
NUMBURST	1	58	62	72	135	272	540	807	1077	1612	2148
	4	58	62	72	100	163	324	485	643	965	1283
	8	58	62	72	100	163	288	447	573	856	1140
	16	57	62	72	100	163	288	412	534	820	1066
Burst Length = 64 bytes											
NUMBURST	1	58	62	72	86	170	335	503	667	1000	1332
	4	58	62	72	86	135	226	359	450	674	898
	8	58	62	72	86	135	226	321	415	639	828
	16	58	62	72	86	135	226	321	415	604	792

Table 4-17. MAX Module Performance for Moving Sequential Data - Source is in Internal Memory and Destination is in External Memory (EMIF is 32 bits Wide)

No. dMAX Clocks		Transfer Size (CNT) in Number of Bytes									
		4	16	32	64	128	256	384	512	768	1024
Burst Length = 4 bytes											
NUMBURST	1	55	216	427	850	1700	3396	5091	6787	10180	13572
	4	55	107	212	419	835	1668	2499	3331	4995	6659
	8	55	107	175	347	692	1379	2067	2754	4132	5508
	16	55	107	175	310	618	1234	1850	2466	3698	4930
Burst Length = 16 bytes											
NUMBURST	1	55	62	121	240	475	947	1420	1892	2835	3779
	4	55	62	86	132	259	514	772	1028	1539	2051
	8	55	62	86	132	223	443	664	884	1323	1763
	16	55	62	86	132	223	408	626	811	1215	1619
Burst Length = 32 bytes											
NUMBURST	1	55	62	70	136	272	539	807	1074	1612	2148
	4	55	62	70	100	163	324	483	643	963	1283
	8	55	62	70	100	163	287	447	571	856	1140
	16	55	62	70	100	163	287	412	534	819	1066
Burst Length = 64 bytes											
NUMBURST	1	55	62	70	86	170	335	501	667	1000	1332
	4	55	62	70	86	132	226	357	450	674	898
	8	55	62	70	86	133	226	321	415	639	828
	16	55	62	70	86	133	226	321	415	604	792

Table 4-18. MAX Module Performance for Moving Sequential Data - Source is in Internal Memory and Destination is in External Memory (EMIF is 16 bits Wide)

No. dMAX Clocks		Transfer Size (CNT) in Number of Bytes									
		4	16	32	64	128	256	384	512	768	1024
Burst Length = 4 bytes											
NUMBURST	1	55	216	427	849	1700	3395	5091	6787	10180	13572
	4	55	107	212	419	835	1668	2499	3331	4996	6659
	8	55	107	174	346	692	1379	2067	2755	4132	5508
	16	55	107	175	310	618	1234	1850	2466	3698	4930
Burst Length = 16 bytes											
NUMBURST	1	55	62	121	240	475	947	1420	1892	2835	3779
	4	55	62	86	132	260	515	772	1028	1539	2051
	8	55	62	86	132	223	443	664	884	1323	1763
	16	55	62	86	132	223	408	627	811	1213	1619
Burst Length = 32 bytes											
NUMBURST	1	55	62	70	137	272	539	807	1075	1612	2148
	4	55	62	70	100	163	324	483	643	963	1283
	8	55	62	70	100	163	287	447	571	856	1140
	16	55	62	70	100	163	287	412	534	818	1066
Burst Length = 64 bytes											
NUMBURST	1	55	62	70	86	170	335	500	667	1000	1332
	4	55	62	70	86	142	253	393	504	753	1005
	8	55	62	70	86	142	253	365	476	728	950
	16	55	62	70	86	142	254	365	476	699	922

Table 4-19. MAX Module Performance for Moving sequential Data - Source is in External Memory and Destination is in Internal Memory (EMIF is 32 bits Wide)

No. dMAX Clocks		Transfer Size (CNT) in Number of Bytes									
		4	16	32	64	128	256	384	512	768	1024
Burst Length = 4 bytes											
NUMBURST	1	72	282	556	1110	2215	4425	6631	8845	13262	17687
	4	72	173	341	677	1349	2696	4039	5383	8071	10772
	8	72	173	304	606	1207	2409	3607	4804	7207	9619
	16	72	173	303	567	1134	2264	3393	4517	6773	9039
Burst Length = 16 bytes											
NUMBURST	1	72	81	156	316	628	1251	1876	2500	3747	4994
	4	72	82	122	206	410	821	1227	1636	2451	3266
	8	72	81	121	206	375	746	1120	1493	2235	2979
	16	72	82	122	206	375	712	1083	1419	2128	2835
Burst Length = 32 bytes											
NUMBURST	1	72	81	91	177	352	701	1049	1397	2093	2788
	4	72	81	91	142	245	484	724	965	1444	1926
	8	72	82	91	142	245	448	690	892	1337	1780
	16	72	81	91	142	245	450	655	856	1301	1709
Burst Length = 64 bytes											
NUMBURST	1	72	82	91	111	218	434	652	868	1300	1731
	4	72	82	91	111	181	327	507	652	975	1299
	8	72	81	91	111	181	327	471	616	939	1227
	16	72	81	91	111	182	327	471	616	903	1191

Table 4-20. MAX Module Performance for Moving Sequential Data - Source is in External Memory and Destination is in Internal Memory (EMIF is 16-bit wide)

No. dMAX Clocks		Transfer Size (CNT) in Number of Bytes									
		4	16	32	64	128	256	384	512	768	1024
Burst Length = 4 bytes											
NUMBURST	1	73	291	577	1156	2308	4610	6917	9223	13836	18446
	4	73	182	361	722	1442	2884	4325	5763	8650	11529
	8	73	181	325	653	1300	2594	3893	5191	7782	10380
	16	73	181	325	614	1227	2453	3678	4901	7349	9806
Burst Length = 16 bytes											
NUMBURST	1	73	86	170	338	674	1347	2020	2690	4035	5378
	4	73	86	134	231	461	916	1372	1829	2739	3651
	8	73	86	133	230	423	842	1262	1682	2523	3362
	16	73	86	133	231	422	807	1227	1611	2416	3219
Burst Length = 32 bytes											
NUMBURST	1	73	86	102	202	400	796	1192	1589	2381	3173
	4	73	86	102	165	292	580	870	1157	1732	2309
	8	73	86	102	166	292	544	833	1088	1625	2165
	16	73	86	102	166	292	544	797	1049	1589	2093
Burst Length = 64 bytes											
NUMBURST	1	73	86	102	135	266	531	795	1059	1586	2115
	4	73	86	102	135	230	423	650	843	1263	1683
	8	73	86	102	135	230	423	616	807	1227	1611
	16	73	86	102	135	230	423	614	807	1190	1575

Table 4-21. MAX Module Performance for Moving Sequential Data - Source is in External Memory and Destination is in External memory (EMIF is 32-bit Wide)

No. dMAX Clocks		Transfer Size (CNT) in Number of Bytes									
		4	16	32	64	128	256	384	512	768	1024
Burst Length = 4 bytes											
NUMBURST	1	79	303	602	1204	2404	4806	7207	9610	14412	19212
	4	79	194	385	772	1538	3079	4614	6154	9228	12305
	8	79	195	351	699	1396	2791	4186	5579	8363	11151
	16	79	196	350	664	1324	2644	3968	5292	7939	10584
Burst Length = 16 bytes											
NUMBURST	1	79	85	167	328	653	1301	1949	2601	3897	5194
	4	79	86	135	233	464	927	1389	1852	2779	3704
	8	79	86	135	233	431	861	1291	1722	2580	3453
	16	79	86	135	233	432	829	1258	1657	2484	3310
Burst Length = 32 bytes											
NUMBURST	1	79	86	95	188	369	737	1102	1468	2202	2933
	4	79	86	95	160	293	583	874	1165	1744	2326
	8	79	86	96	160	293	555	847	1113	1669	2223
	16	79	86	96	160	293	556	823	1091	1643	2173
Burst Length = 64 bytes											
NUMBURST	1	79	86	95	114	226	447	670	891	1337	1782
	4	79	86	95	113	212	408	618	813	1218	1626
	8	79	86	95	113	212	408	602	800	1205	1594
	16	79	86	95	113	212	408	603	799	1189	1580

Table 4-22. MAX Module Performance for Moving Sequential Data - Source is in External Memory and Destination is in External Memory (EMIF is 16-bit Wide)

No. dMAX Clocks		Transfer Size (CNT) in Number of Bytes									
		4	16	32	64	128	256	384	512	768	1024
Burst Length = 4 bytes											
NUMBURST	1	78	304	604	1205	2406	4809	7211	9613	14417	19220
	4	78	198	3989	793	1590	3176	4762	6352	9524	12700
	8	79	199	363	724	1446	2888	4333	5774	8657	11544
	16	79	198	363	687	1374	2744	4116	5489	8231	10973
Burst Length = 16 bytes											
NUMBURST	1	79	91	177	352	699	1397	2096	2792	4184	5578
	4	79	90	153	276	549	1096	1642	2191	3284	4376
	8	79	91	153	275	520	1040	1562	2086	3123	4162
	16	79	90	153	275	521	1016	1535	2030	3043	4054
Burst Length = 32 bytes											
NUMBURST	1	79	91	107	209	419	832	1247	1662	2490	3318
	4	79	90	106	196	377	752	1127	1501	2251	3000
	8	79	91	107	196	377	739	1112	1477	2212	2948
	16	79	91	106	196	377	739	1101	1462	2199	2919
Burst Length = 64 bytes											
NUMBURST	1	79	91	107	139	285	576	868	1160	1744	2327
	4	78	91	106	139	286	576	868	1159	1744	2327
	8	79	91	107	139	285	577	868	1159	1744	2326
	16	79	90	106	139	286	577	868	1160	1744	2326

4.10 SPI Slave Transfer Performance

SPI slave transfer allows for servicing the SPI peripheral when used in slave mode. The peripheral servicing requires that for each input event, one element is read from the SPI input shift register (SPIBUF) and is stored in the destination address. Also, one element is read from the input address and moved to the SPI output shift register (SPIDAT0).

The performance data in Table 4-23 provides the dMAX clocks taken to service one event from the SPI peripheral. The dMAX clocks are provided for various combinations of source and destination locations (IRAM or SDRAM).

Table 4-23. MAX Module Performance for Handling One SPI Event

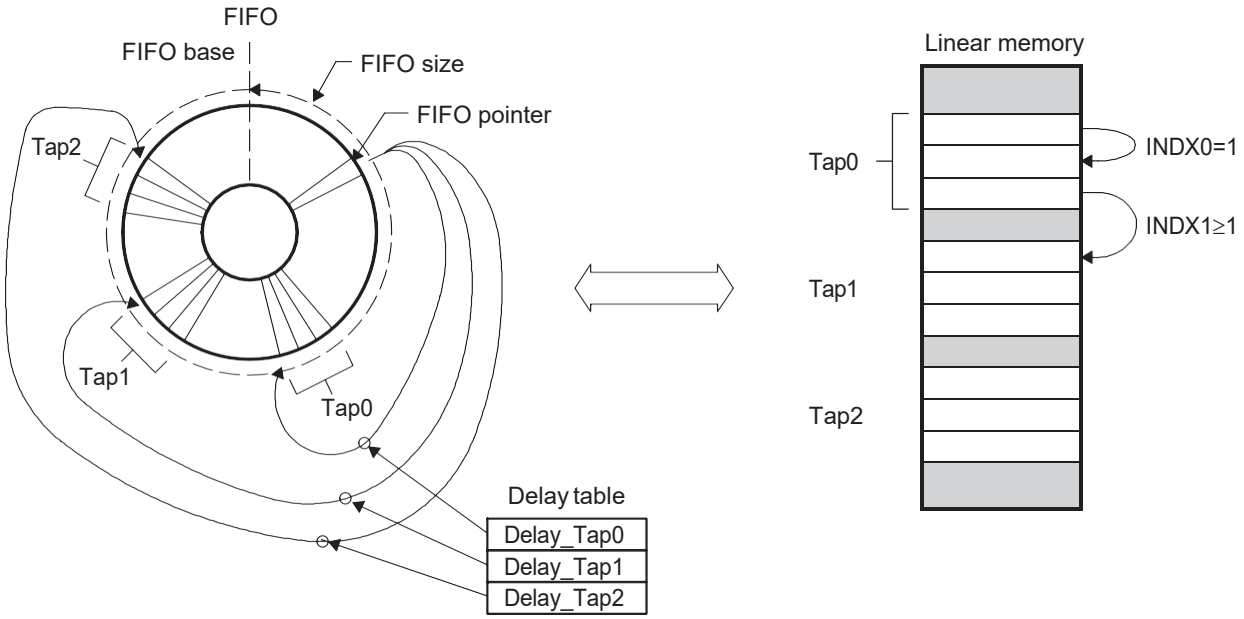
No dMAX Clocks		Destination Location	
		IRAM	SDRAM
Source Location			
8 bit Element	IRAM	63	63
	SDRAM	81	84
16bit Element	IRAM	63	63
	SDRAM	81	84

4.11 FIFO Transfer Performance

A table-guided, multi-tap delay FIFO transfer usually moves a number of taps between the FIFO and the linear memory. The transfer performance depends on a number of factors such as element size, QTSL value, number of taps (blocks), and tap (block) size. If the FIFO is in the external memory, the performance also depends on ratio between dMAX and the EMIF clock.

The table-guided FIFO transfer is presented in Figure 4-5.

Figure 4-5. Table-Guided Multi-tap Delay FIFO Transfer



In a FIFO read, a delay table guides dMAX to retrieve only taps at specified offsets from the FIFO Read Pointer (RP). In a FIFO write, a delay table guides dMAX to write taps at specified offsets from the FIFO Write Pointer (WP).

In Figure 4-5 three taps of three elements are moved between the FIFO and the linear memory. The delay table in Figure 4-5 contains three entries, and each entry is associated with one of the three taps. In a FIFO read transfer, the data flow direction is from left to right and a FIFO RP is used. In a FIFO write transfer, data flow direction is from right to left and an FIFO WP is used.

In the following sections, sequential element spacing within a tap (INDX0=1) in the linear memory is assumed. Taps in the linear memory can be either sequential (INDX1=1), or there could be some spacing between the taps (INDX1>1).

Since the number of taps can vary from application to application in this section, performance is evaluated per tap. The FIFO transfer performance is evaluated by the number of dMAX clocks required to transfer one data block (one tap) between linear memory and the FIFO.

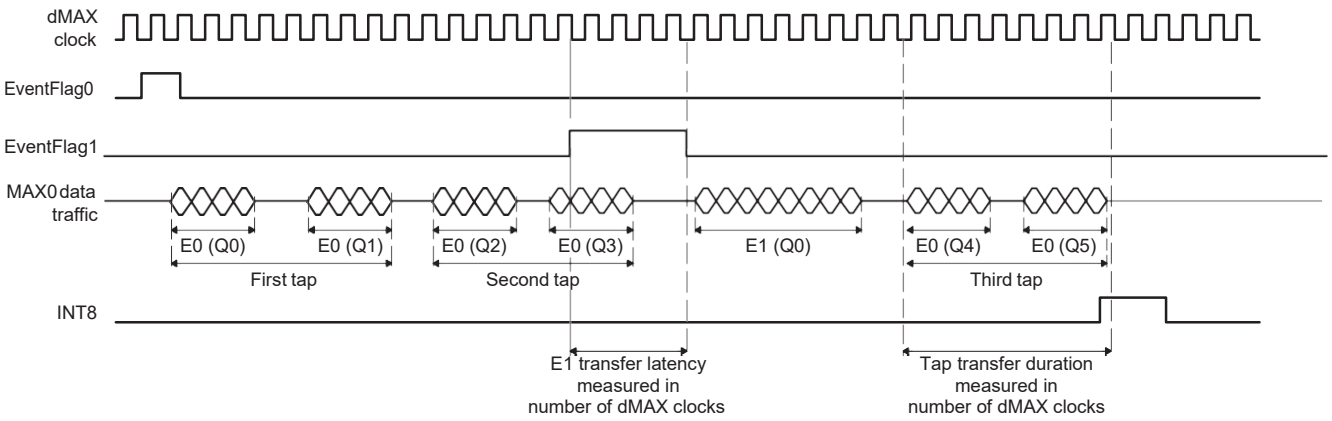
4.12 Transfer Duration and Latency

The performance is measured by number of dMAX clocks required to complete transfer of one tap (block of data). For performance measurements purposes, the FIFO transfers of ten or more taps are synchronized by using one synchronization event (one sync event is used to synchronize the complete transfer). The transfer duration is measured in number of dMAX clocks and is counted from the moment the event flag is cleared to the moment the CPU received notification that transfer completed as shown in Figure 4-6. The number of dMAX clocks required for transfer of one tap is calculated by dividing the number of dMAX clocks required for the whole transfer by number of taps. The data presented represents the best case scenario when there are no resource conflicts during transfer.

The FIFO Transfer E0 presented in Figure 4-6 consists of three taps. In this example, tap size is eight elements, and the QTSL value is set to four. Therefore, each tap transfer has two quantum transfers. The whole FIFO transfer consists of three taps and is broken into six quantum transfers. During the fourth quantum transfer E0(Q3), a new event arrives (Event1). The new event will be processed after completion of the quantum transfer E0(Q3). The delay that the MAX module requires to respond to an event is described by latency. If the MAX module is not busy, a new transfer request is serviced quickly with very low latency. If the MAX module is in the middle of a transfer and another transfer request arrives, the new request will have to wait until completion of the quantum transfer currently in progress.

In FIFO transfers, the latency depends on the QTSL value specified in the event entry. A new event will have to wait until the current quantum transfer in progress is completed. How long the new event will have to wait depends on the size of the quantum transfer currently in progress, and on quantum transfer sizes of all events with higher priority than the current event (in case higher priority events are also waiting to be processed).

Figure 4-6. Transfer Latency and Tap Transfer Duration Measured in Number of dMAX Clocks



The latency is not fixed and depends on overall dMAX controller loading. If dMAX is busy, the new event must wait to be processed until dMAX completes the current quantum transfer. If several events in the same priority group arrive at the same time, the latency for the lowest priority event will be equal to sum of quantum transfer durations of all higher priority pending events.

Using large QTSL values improves system throughput, but worsens system latency. The overall transfer time is shorter when the QTSL value is large, since MAX moves a small number of large quantum transfers.

4.13 FIFO Read

Performance numbers in this section are given in number of dMAX clocks required to move one tap/block of data from the source FIFO to the destination. The performance data is collected using the following assumptions:

- The destination locations within a tap are assumed to be consecutive (DINDX0 = 1).
- The FIFO size is a multiple of tap size
- Entries from the delay table- Tap delays (offsets to the tap location calculated from the Read Pointer) are multiple of the tap size

The number of dMAX clocks required to move multiple blocks of data from the FIFO can be calculated from the tables given in the following sections. For a specific block size (tap size), time required to read one tap from the FIFO is given in Table 4-24, Table 4-25, or Table 4-26 depending on location of the FIFO and the EMIF size (time required to transfer one tap is expressed in number of dMAX clocks). The number of dMAX clocks required to transfer N taps from the FIFO can be calculated by simply multiplying by N the number read from these same tables.

The FIFO read transfers of ten or more taps are used to measure performance. The per tap performance was calculated by dividing total number of dMAX clocks required to transfer all the taps by number of taps transferred.

4.13.1 FIFO Read Transfers Within Internal Memory

The average number of dMAX clock cycles required to transfer one tap (block) of data from the FIFO in the internal data memory of the DSP to destination in the internal data memory is presented in Table 4-24.

Table 4-24. FIFO Read MAX Module Performance for Moving Various Tap Sizes When Both Source FIFO and Destination Locations are in Internal Memory

No. dMAX Clocks		Tap Size (COUNT0) in Number of Elements																	
		4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	128	256
ESIZE = 8 bits																			
QTSL	1	555	1090	1626	2162	2697	3232	3767	4303	4837	5372	5907	6442	6976	7511	8045	8579	17138	34256
	4	156	293	430	567	704	841	977	1114	1251	1388	1524	1661	1798	1934	2070	2207	4389	8741
	8	156	158	295	297	434	436	572	575	711	714	850	852	988	991	1127	1129	2236	4445
	16	156	158	160	162	299	301	303	305	441	444	446	448	584	586	588	590	1160	2296
ESIZE = 16 bits																			
QTSL	1	555	1090	1626	2162	2697	3232	3767	4303	4837	5372	5907	6442	6976	7511	8045	8579	17138	34256
	4	158	297	436	575	714	853	992	1130	1269	1408	1546	1685	1824	1962	2101	2239	4453	8869
	8	158	162	301	305	444	448	587	591	730	733	872	876	1015	1019	1157	1161	2300	4572
	16	158	162	166	170	309	313	317	321	460	464	468	471	610	614	619	622	1223	2423
ESIZE = 32 bits																			
QTSL	1	559	1098	1638	2178	2717	3256	3795	4334	4873	5412	5951	6490	7028	7567	8105	8643	17266	34512
	4	162	305	448	591	734	877	1019	1162	1305	1447	1590	1733	1875	2018	2161	2303	4581	9125
	8	162	170	313	321	464	472	615	623	766	774	916	924	1067	1074	1217	1225	2428	4828
	16	162	169	178	186	329	336	346	352	495	503	511	519	662	670	678	686	1351	3679

4.13.2 FIFO Read Transfers Between Internal Memory and EMIF SDRAM

Throughput to the SDRAM interface depends on the SDRAM setting and on clock ratio between dMAX and the EMIF. In the following sections, the assumption is that ratio between dMAX and the EMIF clock domains is 1.5.

The average number of dMAX clock cycles required to transfer one tap (block) of data from the FIFO in the SDRAM to destination in the internal data memory of the DSP is presented in Table 4-25 for a 32-bit EMIF and in Table 4-26 for a 16-bit EMIF.

Table 4-25. FIFO Read MAX Module Performance for Moving Various Tap Sizes When Source FIFO is in SDRAM and Destination is in Internal Memory (EMIF is 32-bit wide)

No. dMAX Clocks		Tap Size (COUNT0) in Number of Elements																	
		4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	128	256
ESIZE = 8 bits																			
QTSL	1	619	1219	1820	2419	3021	3620	4222	4822	5422	6022	6623	7223	7823	8423	9024	9625	19233	38447
	4	171	324	477	630	784	937	1090	1243	1396	1549	1703	1856	2008	2162	2314	2468	4916	9815
	8	171	174	327	330	483	486	639	642	795	798	952	954	1108	1111	1264	1267	2515	5013
	16	171	174	176	179	333	336	336	339	495	498	498	501	658	660	660	663	1312	2594
ESIZE = 16 bits																			
QTSL	1	619	1219	1819	2419	3020	3621	4222	4822	5422	6022	6623	7223	7823	8423	9024	9625	19233	38447
	4	174	330	486	642	798	954	1110	1266	1423	1579	1734	1890	2046	2230	2359	2515	5012	9937
	8	174	179	336	339	498	501	660	664	823	825	986	987	1146	1151	1308	1311	2607	5169
	16	174	179	183	189	345	350	355	360	516	521	525	531	687	692	696	702	1387	2758
ESIZE = 32 bits																			
QTSL	1	642	1230	1837	2443	3049	3656	4261	4868	5476	6080	6684	7294	7898	8506	9112	9718	19413	38791
	4	178	339	501	663	825	987	1151	1312	1474	1636	1798	1960	2122	2283	2446	2608	5200	10380
	8	178	189	350	360	512	532	692	702	863	873	1034	1044	1205	1215	1376	1386	2755	5493
	16	178	189	198	208	369	381	390	399	561	575	582	591	753	767	774	783	1551	3086

Table 4-26. FIFO Read MAX Module Performance for Moving Various Tap Sizes when Source FIFO Is In the SDRAM and Destination is in Internal Memory (EMIF is 16-bit wide)

No. dMAX Clocks		Tap Size (COUNT0) in Number of Elements																	
		4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	128	256
ESIZE = 8 bits																			
QTSL	1	627	1239	1852	2464	3076	3687	4300	4914	5528	6142	6757	7371	7985	8599	9213	9827	19648	39282
	4	173	327	483	639	795	951	1180	1264	1420	1576	1732	1888	2044	2200	2356	2513	5006	10198
	8	173	177	332	336	491	495	650	654	809	813	969	973	1128	1132	1286	1291	2564	5109
	16	173	177	180	185	339	345	348	352	508	513	516	519	676	681	684	687	1359	2691
ESIZE = 16 bits																			
QTSL	1	627	1239	1852	2464	3076	3687	4300	4914	5528	6142	6757	7371	7985	8599	9213	9828	19648	39282
	4	177	336	495	654	814	973	1132	1291	1450	1609	1769	1927	2086	2244	2404	2563	5108	10198
	8	177	185	345	352	513	519	681	687	850	855	1018	1024	1185	1192	1353	1360	2704	5362
	16	177	185	192	201	360	368	376	384	543	551	559	567	726	734	742	751	1483	2948
ESIZE = 32 bits																			
QTSL	1	630	1243	1855	2468	3080	3691	4303	4914	5530	6144	6758	7373	7986	8600	9216	9831	19661	39321
	4	185	352	520	688	856	1024	1192	1360	1528	1696	1864	2032	2200	2367	2536	2704	5393	10766
	8	185	201	368	384	551	568	734	750	917	934	1100	1117	1284	1300	1466	1483	2948	5878
	16	185	201	217	233	400	417	432	448	616	633	649	664	831	849	864	880	1744	3471

4.14 FIFO Write Transfer

Performance numbers in this section are given in number of dMAX clocks required to move one tap/block of data from the source to the FIFO. The performance data is collected using the following assumptions:

- It is also assumed that FIFO size is multiple of tap size.
- The elements within source taps are assumed to be consecutive (SINDX0 = 1).
- Entries from the delay table-Tap delays (offsets to the tap location calculated from the WP) are multiple of the tap size.

The number of dMAX clocks required to move multiple blocks of data to the FIFO can be calculated from the tables given in the following sections. For a specific block size (tap size), time required to write one tap to the FIFO is given in Table 4-27, Table 4-28, or Table 4-29 depending on location of the FIFO and EMIF size (time required to transfer one tap is expressed in number of dMAX clocks). The number of dMAX clocks required to transfer N taps to the FIFO can be calculated by simply multiplying by N the number read from these same tables.

4.14.1 FIFO Write Transfers Within the Internal Memory

The average number of dMAX clock cycles required to transfer one tap (block) of data to the FIFO in the SDRAM from source in the internal data memory is presented in Table 4-27.

Table 4-27. FIFO Write MAX Module Performance for Various Tap Sizes when Source Data and Destination FIFO are in Internal Memory

No. dMAX Clocks		Tap Size (COUNT0) in Number of Elements																	
		4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	128	256
ESIZE = 8 bits																			
QTSL	1	560	1100	1639	2179	2718	3257	3797	4336	4874	5413	5952	6491	7029	7568	8106	8645	17269	34517
	4	158	296	434	572	710	848	985	1123	1261	1399	1536	1674	1812	1949	2987	2224	4422	8806
	8	158	160	298	300	438	440	578	580	717	719	857	859	997	999	1136	1138	2253	4477
	16	158	160	162	165	302	304	307	308	446	448	450	452	589	592	594	596	1169	2313
ESIZE = 16 bits																			
QTSL	1	560	1100	1639	2179	2718	3257	3797	4336	4874	5413	5952	6491	7029	7568	8106	8645	17270	34517
	4	160	300	440	580	720	860	999	1139	1279	1418	1558	1698	1837	1977	2117	2256	4486	8934
	8	160	165	305	308	448	452	592	596	736	740	880	883	1023	1027	1167	1171	2317	4606
	16	160	165	168	173	312	317	320	324	464	468	472	476	615	620	624	627	1233	2441
ESIZE = 32 bits																			
QTSL	1	564	1108	1651	2195	2738	3281	3824	4368	4910	5427	5996	6539	7081	7624	8166	8709	17397	34773
	4	165	308	452	596	740	884	1028	1171	1315	1459	1602	1746	1890	2033	2177	2321	4614	9192
	8	165	173	317	324	468	476	620	628	772	780	924	931	1075	1083	1227	1234	2445	4867
	16	165	173	181	189	332	340	348	357	500	508	516	524	668	675	684	692	1361	2567

4.14.2 FIFO Write Transfers Between Internal Memory and EMIF SDRAM

Throughput to the SDRAM interface depends on the SDRAM setting and on clock ratio between dMAX and EMIF. In the following sections, the assumption is that the ratio between dMAX and the EMIF clock domains is 1.5.

The average number of dMAX clock cycles required to transfer one tap (block) of data from source in the internal data memory of the DSP to the destination FIFO in the SDRAM is presented in Table 4-28 for a 32-bit EMIF and in Table 4-29 for a 16-bit EMIF.

Table 4-28. FIFO Write MAX Module Performance for Various Tap Sizes when Source is in Internal Memory and Destination FIFO is in SDRAM (EMIF is 32-bit wide)

No. dMAX Clocks		Tap Size (COUNT0) in Number of Elements																	
		4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	128	256
ESIZE = 8 bits																			
Q	1	560	1100	1639	2179	2718	3257	3796	4335	4874	5413	5952	6490	7029	7567	8106	8644	17269	34517
T	4	158	296	434	572	710	847	985	1123	1260	1398	1536	1674	1811	1949	2086	2224	4422	8806
S	8	158	160	298	300	438	439	577	579	717	719	857	859	996	998	1136	1138	2253	4477
L	16	158	160	162	164	301	304	306	308	446	447	449	451	589	591	593	595	1169	2313
ESIZE = 16 bits																			
Q	1	557	1099	1639	2176	2718	3257	3796	4335	4874	5413	5952	6490	7029	7567	8106	8644	17270	34517
T	4	157	297	437	577	717	857	999	1139	1278	1418	1558	1697	1837	1977	2116	2256	4485	8934
S	8	157	161	301	305	445	449	589	593	733	737	877	881	1022	1027	1166	1170	2317	4605
L	16	157	161	165	169	309	313	317	321	461	465	469	473	613	617	621	625	1232	2441
ESIZE = 32 bits																			
Q	1	561	1108	1651	2195	2738	3281	3825	4367	4910	5453	5996	6538	7081	7623	8166	8709	17397	34773
T	4	161	305	449	593	737	881	1027	1171	1314	1459	1602	1746	1890	2033	2176	2320	4614	9190
S	8	161	169	313	321	465	473	617	625	769	777	921	929	1075	1083	1226	1234	2445	4861
L	16	161	169	177	185	329	337	345	353	497	505	513	521	665	673	681	689	1361	2397

Table 4-29. FIFO Write MAX Module Performance for Various Tap Sizes when Source is in Internal Memory and Destination FIFO is in SDRAM (EMIF is 16-bit wide)

No. dMAX Clocks		Tap Size (COUNT0) in Number of Elements																	
		4	8	12	16	20	24	28	32	36	40	44	48	52	56	60	64	128	256
ESIZE = 8 bits																			
Q	1	560	1100	1639	2179	2718	3257	3796	4335	4874	5413	5952	6490	7029	7567	8106	8644	17269	34517
T	4	158	296	434	572	710	847	985	1123	1260	1398	1536	1674	1811	1949	2086	2224	4422	8806
S	8	158	160	298	300	438	439	577	579	717	719	857	859	996	998	1136	1138	2253	4477
L	16	158	160	162	164	301	304	306	308	446	447	449	451	589	591	593	595	1169	2313
ESIZE = 16 bits																			
Q	1	557	1099	1639	2179	2718	3257	3796	4335	4874	5413	5952	6490	7029	7567	8106	8644	17270	34517
T	4	157	297	437	577	717	857	999	1139	1278	1418	1558	1697	1837	1977	2116	2256	4485	8934
S	8	157	161	301	305	445	449	589	593	733	737	877	881	1022	1027	1166	1170	2317	4605
L	16	157	161	165	169	309	313	317	321	461	465	469	473	613	617	621	625	1232	2441
ESIZE = 32 bits																			
Q	1	561	1108	1651	2195	2738	3281	3825	4367	4910	5453	5996	6538	7081	7623	8166	8709	17397	34773
T	4	161	305	449	593	737	881	1027	1171	1314	1459	1602	1746	1890	2033	2176	2320	4614	9190
S	8	161	169	313	321	465	473	617	625	769	777	921	929	1075	1083	1226	1234	2445	4861
L	16	161	169	177	185	329	337	345	353	497	505	513	521	665	673	681	689	1361	2697

Revision History

This document has been revised from SPRU795C to SPRU795D because of the following technical change(s):

Table A-1. Changes in this Revision

Location	Additions/Modifications/Deletions
Figure 3-56	Changed the PTE value from 0111110 to 11111000

IMPORTANT NOTICE AND DISCLAIMER

TI PROVIDES TECHNICAL AND RELIABILITY DATA (INCLUDING DATASHEETS), DESIGN RESOURCES (INCLUDING REFERENCE DESIGNS), APPLICATION OR OTHER DESIGN ADVICE, WEB TOOLS, SAFETY INFORMATION, AND OTHER RESOURCES “AS IS” AND WITH ALL FAULTS, AND DISCLAIMS ALL WARRANTIES, EXPRESS AND IMPLIED, INCLUDING WITHOUT LIMITATION ANY IMPLIED WARRANTIES OF MERCHANTABILITY, FITNESS FOR A PARTICULAR PURPOSE OR NON-INFRINGEMENT OF THIRD PARTY INTELLECTUAL PROPERTY RIGHTS.

These resources are intended for skilled developers designing with TI products. You are solely responsible for (1) selecting the appropriate TI products for your application, (2) designing, validating and testing your application, and (3) ensuring your application meets applicable standards, and any other safety, security, or other requirements. These resources are subject to change without notice. TI grants you permission to use these resources only for development of an application that uses the TI products described in the resource. Other reproduction and display of these resources is prohibited. No license is granted to any other TI intellectual property right or to any third party intellectual property right. TI disclaims responsibility for, and you will fully indemnify TI and its representatives against, any claims, damages, costs, losses, and liabilities arising out of your use of these resources.

TI’s products are provided subject to TI’s Terms of Sale (www.ti.com/legal/termsofsale.html) or other applicable terms available either on ti.com or provided in conjunction with such TI products. TI’s provision of these resources does not expand or otherwise alter TI’s applicable warranties or warranty disclaimers for TI products.

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