TMS320C6x DMA Applications

APPLICATION REPORT: PRELIMINARY

David Bell
Jackie Brenner

June 19, 1998

IMPORTANT NOTICE

TRADEMARKS

Contents

Tables

Figures

TMS320C6x DMA Applications

Abstract

The on-chip Direct Memory Access (DMA) controller is used to transfer data between two locations in the memory map in the background of CPU operation. Typically the DMA will be used to transfer blocks of data between external and internal data memories, restructure portions of internal data memory, continually service a Multi-channel Buffered Serial Port (McBSP) or Analog Front End (AFE) circuit, or page program sections to internal program memory. There are four DMA channels that may be programmed to perform one or more of these tasks, while the CPU is executing a program. Channels may be configured to run continuously throughout the device's entire operation, with only one setup required. The fixed priority scheme between the channels allows for high-priority synchronous transfers to be performed during a low-priority block transfer. The DMA channels may communicate their status to the CPU through interrupts to provide the CPU with control over their operation. Most applications require only an initial setup of DMA control registers, with little intervention by the CPU to maintain their operation.

Background

The on-chip Data Memory Access (DMA) Controller is used to transfer data from one memory-mapped location to another without the intervention of the CPU. Data may be transferred between internal memory, peripherals, and external devices in the background of CPU operation allowing for the CPU to remain active during data transfers. There are four DMA channels, which may be independently configured to perform different types of transfers.

The DMA is highly flexible, in that there are many different types of transfers that may be done to enable faster throughput by the CPU through data organization. With the DMA, data may be transferred to and from internal program memory, internal data memory, an external memory space, an external Analog Front End (AFE) circuit, or the Multi-channel Buffered Serial Ports (McBSPs). Also by using the DMA, data currently in memory may be reorganized to increase the CPU's effectiveness.

Each channel of the DMA has the following set of registers that must be configured prior to beginning a data transfer:

- Primary Control Register – Used to configure the transfer
- Secondary Control Register – Used to enable interrupts to the CPU and to monitor the channels activity
- Transfer Counter Register – Used to keep track of the transferred elements
- Source Address Register – The memory location from which the element will be transferred
- Destination Address Register – The memory location to which the element will be transferred

In addition to these, there are several global DMA registers that may be used by any of the DMA channels to perform more complicated transfers:

- Global Address Registers (A, B, C, and D) – Used as either a Split Address, or as an address reload value
- Global Index Registers (A and B) – Used to control address updates during a transfer

- Global Count Reload Registers (A and B) – Used to reload the Transfer Counter Register of a DMA channel

Each of the global DMA registers may be used by any of the DMA channels, and more than one channel may use the same register at a time.

There is one additional DMA register, the Auxiliary Control Register, which is used to set the priority of the Auxiliary Channel with respect to the four main DMA channels and the CPU. The Auxiliary Channel is used by the Host Port to access the C62xx memory.

The TMS320C62xx Peripherals Reference Guide gives a complete description of the DMA structure, and should be used in conjunction with this document.

System Structure

The DMA may be used to access any location in the ‘C6x memory map. This includes internal data memory, internal program memory, on-chip peripherals, external memories, and external AFEs. Typically the DMA will be used to transfer blocks of data between external and internal data memories, restructuring portions of internal data memory, continually servicing a McBSP or AFE, or for program paging.

All accesses to external memory spaces must go through the External Memory Interface (EMIF). External memory types which are supported on the ‘C6x are Synchronous DRAM (SDRAM), Sync-Burst SRAM (SBSRAM), and asynchronous memories. To understand how to configure different memory spaces, see the TMS320C62xx Peripherals Reference Guide.

External AFE circuits will predominantly use the asynchronous memory interface of the ‘C6x. A typical AFE configuration will include a Data-In address, a Data-Out Address, a Read Sync signal, and a Write Sync signal. Synchronization events would be connected to one of the four external interrupt pins of the device (EXT_INT[7:4]).

The McBSP is the only on-chip peripheral that is likely to require servicing by the DMA. Each McBSP has a Data Receive Register (DRR), a Data Transmit Register (DXR), a transmit-event signal (XEVT), and a receive-event signal (REVT). The DRR and DXR are memory-mapped registers, and the events occur whenever data is transferred out (XEVT) or transferred in (REVT).

Internal Data memory is divided into several 16-bit banks. For the TMS320C6201 there are four banks and for the TMS320C6201B there are two blocks of four banks, with each block occupying half of the data memory. Each bank may only be accessed once per cycle, either by the DMA or by one of the CPU sides (A or B). If both the DMA and the CPU attempt to access the same bank during the same cycle, then the priority bit set in the DMA channel’s Priority Control Register will determine the order in which the access is granted.

Internal Program memory always gives the CPU priority over the DMA. In order for the DMA to access program memory, there must be time slots during which it can get in. These slots occur when a fetch packet (eight instructions) contains multiple execute packets (a group of instructions executed in one cycle). This leaves cycles in which the CPU is not requesting a fetch packet, and the DMA may access the program memory.

Data Relocation

The purpose of the DMA is to move data elements from one location to another. Through proper configuration of the DMA channel control registers, the data to be transferred can be moved in its current format or may be restructured to fit a particular application.

The simple case is a block move, in which a contiguous memory space is copied from one location to another, unaltered. This transfer requires the minimum amount of setup, and is usually performed either to transfer a program section from an external memory location to internal program memory, or to transfer a data section from external memory to internal data memory.

By taking advantage of some of the features of the DMA, a more complicated transfer may be performed in which a section of data is reorganized during the transfer. One example of this would be sorting, in which a data block, divided into contiguous frames of equal size, is reorganized in memory by ordinal location within a frame. In other words the first element of the first frame would be located next to the first element of the second frame. This type of transfer is frequently performed when multiple frames of data are arriving to the device via the serial port (or AFE), or when data arrays, located in external memory, are brought on-chip.

The following examples demonstrate how the DMA may be used to relocate and reorganize data.

Block Move Example

The block move is used to simply transfer a block of contiguous memory from one location to another. This is ordinarily done to move a data or program section from external memory to internal memory, where the CPU may do single-cycle accesses. For this transfer, four of the five basic registers mentioned in the Background section must be configured: The Primary Control Register, the Transfer Counter Register, the Source Address Register, and the Destination Address Register.

Consider an example in which a 1k block of contiguous 32-bit elements are transferred from off-chip memory located at the base address of CE2 (0x02000000) to the base of internal data memory (0x800000000). To initialize this transfer, the following values should be set for the four control registers:

Primary Control Register = 0x00000050

Source Address Register = 0x02000000

Destination Address Register = 0x80000000

Transfer Counter Register = 0x00000400

The individual fields of the Primary Control Register are shown in Figure 1 and the fields of the Transfer Counter Register are shown in Figure 2.

Figure 1: Primary Control Register Setup for Block Move Example

31 30	29 28	27	26	25	24	23 19	18 16
00	00	0	0	0	0	00000	000
DST RELOAD	SRC RELOAD	EMOD	FS	TCINT	PRI	WSYNC	RSYNC

15 14	13	12	11 10	9 8	7 6	5 4	3 2	1 0
00	0	0	00	00	01	01	00	00
RSYNC	INDX	CNT RELOAD	SPLIT	ESIZE	DST DIR	SRC DIR	STATUS	START

Figure 2: Transfer Counter Register Setup for Block Move Example

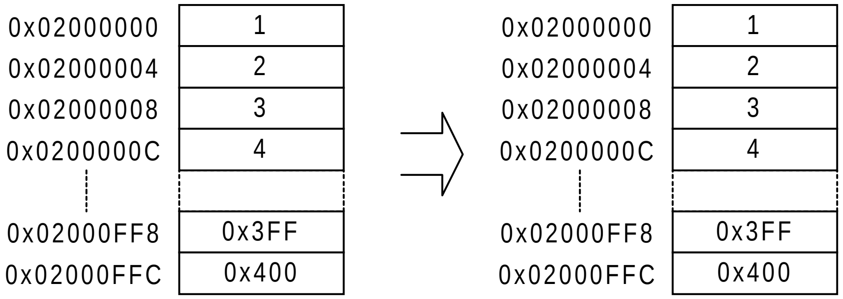
31 16	15 0
0x0000	0x0400
FRAME COUNT	ELEMENT COUNT

The settings of 01b in the DST DIR and SRC DIR bitfields will cause the DMA channel to increment both the source address and the destination address by one element size (4 bytes in this example) following the transfer of each element.

In order to initiate the transfer, a value of 01b needs to be written to the START bitfield.

Figure 3 shows the transfer that the above setup will perform.

Figure 3: Block Move Example Diagram



Extremely Large Block Move Example

Occasionally there is a need to perform a block move of a large section of memory containing more than 65535 elements (the maximum value of ELEMENT COUNT). This transfer type would typically be used to perform a data dump from external memory to an off-chip AFE, or to initialize a memory space from an AFE. This transfer is essentially the same as the basic block move in the previous example, except multiple frames must be used. Using the frame count in conjunction with the element count, it is possible to transfer a single block of up to 0xFFFE0001 (4.3G) elements. This is much greater than the 65535 possible using the element count alone.

For a large block transfer, the following are true:

- If the address is set to be adjusted using a programmable value ((SRC/DST)_DIR = 11b), the Frame Index must equal the Element Index.
- Frame synchronization must be disabled. This will prevent a synchronization event in the middle of the transfer.
- The number of elements transferred in the entire block is ((F - 1) x Er) + Ei, where:
  - F = The initial value of the Frame Count
  - Ei = The initial value of the Element Count
  - Er = Element Count Reload value

If the numbers of elements to be transferred is constant for a given application, then suitable values may be used explicitly in a program. For a majority of transfer lengths, many count and reload values will provide the same performance.

If the block length were not a fixed amount, but established during run-time, then an algorithm to determine the count and reload values during execution would be a more convenient solution.

Using the above information, a simple formula may be created to calculate F, Ei, and Er from a given block size. One possible formula is as follows:

- Ei = 15 LSBs of total element count. Fix to 0x8000 if 15 LSBs are all 0.
- Er = 0x8000 (fixed)
- F = total element count divided by Er, plus 1. Do not add 1 if Ei is forced to 0x8000.

The following C code performs the above calculations:

```
F       =(XFER_SIZE >> 15)+1;
Ei      = XFER_SIZE & 0x7FFF;
if (!Ei){
     Ei = 0x8000;
     F -= 1;
}
Er      = 0x8000;
```

For this set of equations, the maximum transfer size is 0x7FFF7FFF (2.15G). If larger transfers were to be performed, then a more complicated algorithm would need to be established.

For this example, assume that the entire memory space CE2 (16MB) is to be transferred to an off-chip peripheral located at 0x00400000 (CE0). This transfer is of 0x00400000 32-bit words. For this, the following values must be assigned to the DMA registers[1]:

Primary Control Register = 0x00000010

Source Address Register = 0x02000000

Destination Address Register = 0x00400000

Transfer Counter Register = 0x00808000

Global Count Reload Register A = 0x00008000

---

[1] The sample formulas were used to determine the Transfer Counter and Global Count Reload A values. If different formulas were used to obtain the count values, those numbers may be different.

The Primary Control Register for channel 0 should be configured as shown in Figure 4, and the Transfer Counter Register is shown in Figure 5.

Figure 4: Primary Control Register Setup for Extremely Large Block Move Example

31 30	29 28	27	26	25	24	23 19	18 16
00	00	0	0	0	0	00000	000
DST RELOAD	SRC RELOAD	EMOD	FS	TCINT	PRI	WSYNC	RSYNC

15 14	13	12	11 10	9 8	7 6	5 4	3 2	1 0
00	0	0	00	00	00	01	00	00
RSYNC	INDX	CNT RELOAD	SPLIT	ESIZE	DST DIR	SRC DIR	STATUS	START

Figure 5: Transfer Counter Register Setup for Extremely Large Block Move Example

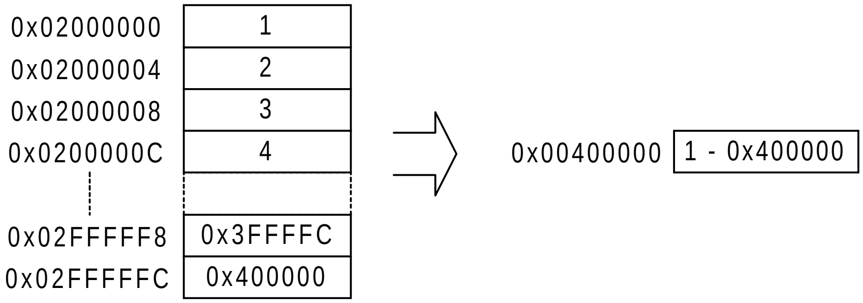
31 16	15 0
0x0080	0x8000
FRAME COUNT	ELEMENT COUNT

The settings of 01b in the SRC DIR bitfields will cause the DMA channel to increment the source address by one element size (4 bytes in this example) following each element. Since the destination is a fixed address, DST DIR is set to 00b.

In order to initiate the transfer, a value of 01b needs to be written to the START bitfield.

Figure 6 shows the transfer that the above setup will perform.

Figure 6: Extremely Large Block Move Example Diagram



Data-Sorting Transfer Example

When an application requires the use of multiple data arrays, it is often desirable to have the arrays arranged such that the first elements of each array are adjacent, the second elements are adjacent, and so on. Often this is not the format in which the data is presented to the device. Either data comes via a peripheral, in which the data arrays arrive one after the other, or the arrays are located in memory, with each array occupying a portion (frame) of contiguous memory spaces. For these instances, the DMA may be configured to reorganize the data into the desired format.

The following formulas may be used to set up a DMA channel to organize the data in memory by ordinal position:

- FRAME INDEX should be set to –(((E – 1) x F) – 1) x S
- ELEMENT INDEX should be set to F x S, where
  - F = The initial value of Frame Count
  - E = The initial value of Element Count, as well as the Element Count Reload value
  - S = The element size in bytes

This example focuses on the second case mentioned above, in which equal sized data arrays are located in external memory. For this transfer to give the desired results, it is necessary that the arrays are of the same size, and they reside in contiguous memory.

For this example it will be assumed that the data is located in 16-bit ROM, beginning at address 0x01600000 (Map 1, CE1). The DMA channel will be configured to bring four frames of 1k half-words from their locations in ROM to internal data memory beginning at 0x80000000[2]. The index values will be:

- FRAME INDEX = –(((1024 – 1) x 4) – 1) x 2 = 0xE00A
- ELEMENT INDEX should be set to 4 x 2 = 8

For this, the following values must be assigned to the DMA channel’s control registers:

---

[2] Note that on if this transfer is performed on the TMS320C6201, then each array will be located in its own memory bank. This will allow for multiple arrays to be accessed during the same cycle without any contention. See the TMS320C62xx Peripherals Reference Guide for details on the configuration of internal data memory.

Primary Control Register = 0x000001D0

Source Address Register = 0x01600000

Destination Address Register = 0x80000000

Transfer Counter Register = 0x00040400

Global Count Reload Register A = 0x00000400

Global Index Register A = 0xE00A0008

The Primary Control Register for channel 0 should be configured as shown in Figure 7, the Transfer Counter Register is shown in Figure 8, and Global Index Register A is shown in Figure 9.

Figure 7: Primary Control Register Setup for Data-Sorting Transfer Example

31 30	29 28	27	26	25	24	23 19	18 16
00	00	0	0	0	0	00000	000
DST RELOAD	SRC RELOAD	EMOD	FS	TCINT	PRI	WSYNC	RSYNC

15 14	13	12	11 10	9 8	7 6	5 4	3 2	1 0
00	0	0	00	01	11	01	00	00
RSYNC	INDX	CNT RELOAD	SPLIT	ESIZE	DST DIR	SRC DIR	STATUS	START

Figure 8: Transfer Counter Register Setup for Data-Sorting Transfer Example

31 16	15 0
0x0004	0x0400
FRAME COUNT	ELEMENT COUNT

Figure 9: Global Index Register A Setup for Data-Sorting Transfer Example

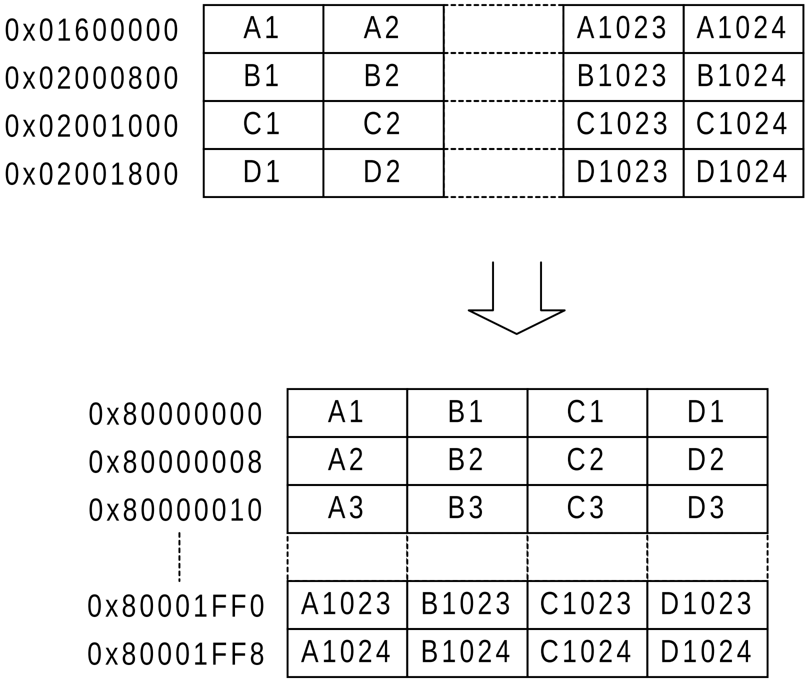
31 16	15 0
0xE00A	0x0008
FRAME INDEX	ELEMENT INDEX

The settings of 01b in the SRC DIR bitfield will cause the DMA channel to increment source address by one element size (2 bytes in this example) following each element. DST DIR being set to 11b will cause the destination address to be modified according to Global Index Register A (INDEX = 0). ELEMENT INDEX will be used following each element within each frame to increment the destination address by 8 bytes (4 elements). FRAME INDEX will be used following the last element of each frame to set the destination address to the first element of the subsequent frame.

In order to initiate the transfer, a value of 01b needs to be written to the START bitfield.

Figure 10 shows the transfer that the above setup will perform.

Figure 10: Data Sorting Example Diagram



0x01600000	A1	A2		A1023	A1024
0x02000800	B1	B2		B1023	B1024
0x02001000	C1	C2		C1023	C1024
0x02001800	D1	D2		D1023	D1024

0x80000000	A1	B1	C1	D1
0x80000008	A2	B2	C2	D2
0x80000010	A3	B3	C3	D3
0x80001FF0	A1023	B1023	C1023	D1023
0x80001FF8	A1024	B1024	C1024	D1024

Servicing a Peripheral

In many ‘C6x applications the DMA will be used to service a peripheral which is sending data to, and receiving data from the device. This peripheral is most commonly either a McBSP or an external AFE. In order for the DMA to effectively communicate with either of these, it must be configured to perform a synchronized data transfer. It must write only when the peripheral is able to accept new data, and read only when the peripheral has new data available.

There are three types of synchronization available to a DMA channel:

- Read Synchronization: Each read transfer waits for the selected event to occur before proceeding
- Write Synchronization: Each write transfer waits for the selected event to occur before proceeding
- Frame Synchronization: Each frame transfer waits for the selected event to occur before proceeding

Each DMA channel may be configured for Read Synchronization, Write Synchronization, both, or Frame Synchronization. If Frame Synchronization is used, then the Read Synchronization event triggers the frame transfer. Synchronization is established in the DMA channel’s Primary Control Register.

The events to the DMA from each McBSP are XEVT and REVT. XEVT is issued when a value has been copied from the Data Transmit Register (DXR) to the Transmit Shift Register (XSR), signifying that the most recent data has been transferred out. REVT is issued when a value has been copied from the Receive Buffer Register (RBR) to the Data Receive Register (DRR), indicating that a new data value has been received. An external AFE would typically have similar synchronization events arriving through one or more of the external interrupt pins (EXT_INT[7-4]).

A complete list of DMA synchronization events is given in Table 1.

Table 1: DMA Channel Synchronization Events

Event number (binary)	Event Acronym	Event Description
00000	None	No Synchronization
00001	TINT0	Timer 0 interrupt
00010	TINT1	Timer 1 interrupt
00011	SD_INT	EMIF SDRAM timer interrupt
00100	EXT_INT4	External interrupt pin 4
00101	EXT_INT5	External interrupt pin 5
00110	EXT_INT6	External interrupt pin 6
00111	ENX_INT7	External interrupt pin 7
01000	DMA_INT0	DMA channel 0 interrupt
01001	DMA_INT1	DMA channel 1 interrupt
01010	DMA_INT2	DMA channel 2 interrupt
01011	DMA_INT3	DMA channel 3 interrupt
01100	XEVT0	MCSP 0 transmit event
01101	REVT0	MCSP 0 receive event
01110	XEVT1	MCSP 1 transmit event
01111	REVT1	MCSP 1 receive event
10000	DSPINT	Host to DSP interrupt

In addition to properly synchronizing the peripherals, care must be taken to ensure that the data is being transferred to and from the correct location. This becomes an issue when performing transfers for 8- and 16-bit elements, particularly when operating in an endian mode that is different than the peripheral expects.

Synchronized Data Transfer Example

In order to transfer data to and from a McBSP[3], it is necessary to use Read Synchronization for reading the DRR and Write Synchronization for writing to the DXR.

---

[3] This information is valid for servicing an external AFE as well. The more frequent the accesses to the AFE, however, the more favorable a frame-synchronized transfer solution would be. It is usually important to keep the arbitration within the EMIF to a minimum.

Consider a variation of the Block Move example. Once again it is desired to bring a 1k block of 32-bit elements into data memory, beginning at address 0x80000000. Instead of bringing the data from a external memory space, however, it will be arriving through McBSP 0. Every time a new 32-bit data value arrives in McBSP 0 DRR Register, the event REVT0 will be set. The DMA channel servicing this data must therefore have its read transfers synchronized on this event (RSYNC = 01101b). To initialize this transfer, the following values should be set for the four control registers:

Primary Control Register = 0x00034040

Source Address Register = 0x018C0000

Destination Address Register = 0x80000000

Transfer Counter Register = 0x00000400

The individual fields of the Primary Control Register are shown in Figure 11.

Figure 11: Primary Control Register Setup for Synchronized Data Transfer Example

31 30	29 28	27	26	25	24	23 19	18 16
00	00	0	0	0	0	00000	011
DST RELOAD	SRC RELOAD	EMOD	FS	TCINT	PRI	WSYNC	RSYNC

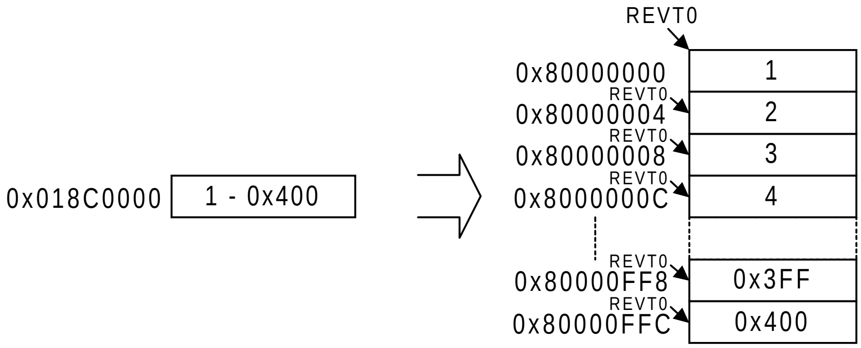
15 14	13	12	11 10	9 8	7 6	5 4	3 2	1 0
01	0	0	00	00	01	00	00	00
RSYNC	INDX	CNT RELOAD	SPLIT	ESIZE	DST DIR	SRC DIR	STATUS	START

The settings of 01b in the DST DIR bitfield will cause the DMA channel to increment the destination address by one element size (4 bytes in this example) following each element. Since the source is a fixed address, SRC DIR is set to 00b. RSYNC is set to REVT0 to synchronize the reading of the DRR.

In order to initiate the transfer, a value of 01b needs to be written to the START bitfield.

Figure 12 shows the transfer that the above setup will perform.

Figure 12: Synchronized Data Transfer Example Diagram



Split-Mode Transfer Example

A McBSP or AFE is more commonly used for bi-directional communication with the device, which means that the DMA will need to both read and write to the peripheral. Adding to the previous example, consider that at the same time 1k words are being received from the McBSP, 1k words are being transmitted to it as well. In order to facilitate this there needs to be a channel set up to transfer data from internal memory to McBSP 0 DXR. This channel must be write-synchronized on the event XEVT0.

While this could easily be done with another DMA channel, one of the features of the DMA is that a single channel may be used to perform two simultaneous transfers. This feature was designed for use with a peripheral, for which the transmit- and receive-data addresses are fixed. Using this feature, the previous example could be modified such that a single DMA channel is used to perform both reads from and writes to the McBSP.

Setting the SPLIT bitfield in the DMA Channel Primary Control Register enables a split-mode transfer and selects the location of the Split Address. Possible SPLIT values are listed in Table 2.

Table 2: DMA Channel SPLIT Settings

SPLIT Value	Split Address
00	Split mode disabled
01	DMA Global Address Register A
10	DMA Global Address Register B
11	DMA Global Address Register C

Global Address Registers A, B, and C may be used to hold the Split Address. This address is assumed to be on an even word boundary, as the three LSBs are reserved and fixed at zero. This address is used as the Split Source Address. The Split Destination Address is automatically set to be one word address greater than the Split Source Address. If an external peripheral is to be serviced by a DMA channel in split-mode, this addressing convention must be followed.

In this example a block of 1k 32-bit words will be transferred to McBSP 0, and a block of 1k 32-bit words will be transferred from McBSP 0 to memory using the same DMA channel.[4] The 1k data block to be transferred to the McBSP will begin at address 0x80001000, while the input data block will again be written to 0x80000000. For this, the following values must be assigned to the DMA registers:

Primary Control Register = 0x00634450

Source Address Register = 0x80001000

Destination Address Register = 0x80000000

Transfer Counter Register = 0x00000400

Global Address Register A = 0x018C0000

The individual fields of the Primary Control Register are shown in Figure 13.

---

[4] Note that the DXR address is the next adjacent memory address above the DRR.

Figure 13: Primary Control Register Setup for Split-Mode Data Transfer Example

31 30	29 28	27	26	25	24	23 19	18 16
00	00	0	0	0	0	01100	011
DST RELOAD	SRC RELOAD	EMOD	FS	TCINT	PRI	WSYNC	RSYNC

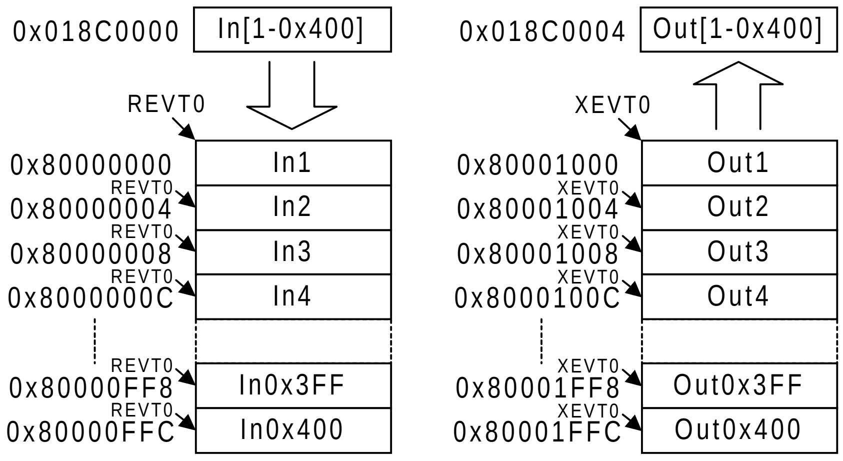
15 14	13	12	11 10	9 8	7 6	5 4	3 2	1 0
01	0	0	01	00	01	01	00	00
RSYNC	INDX	CNT RELOAD	SPLIT	ESIZE	DST DIR	SRC DIR	STATUS	START

The settings of 01b in the SRC DIR and DST DIR bitfields will cause the DMA channel to increment both the source address and the destination address by one element size (4 bytes in this example) following each element. Since the this is a split transfer, the elements from the source address will be written to the Split Destination Address (DXR) and the elements transferred from the Split Source Address (DRR) is written to the Destination Address. RSYNC is set to REVT0 and WSYNC is set to XEVT0.

In order to initiate the transfer, a value of 01b needs to be written to the START bitfield.

Figure 14 shows the transfer that the above setup will perform.

Figure 14: Split-Mode Transfer Example Diagram



Frame-Synchronized Data Transfer Example

If accesses to an external AFE are frequent, it may be beneficial to transfer elements in bursts, rather than making single accesses through the EMIF. Bursting makes more efficient use of the EMIF, as there are fewer cycles lost to arbitration between requesters. In order to facilitate bursting, it may be necessary to have an intermediate FIFO buffering system as part of the AFE. By using a FIFO interface, there will still be a Data-In address and a Data-Out address from the perspective of the DMA. The synchronization event would then be an external signal from the FIFO (or external control logic) indicating when the FIFO has sufficient data to burst a frame.

In order to perform a synchronized burst, the DMA channel must be configured with Frame Synchronization. The FS bitfield must equal 1 and the RSYNC[5] bitfield should be set to the desired synchronization event, both in the Primary Control Register.

Consider a modification of the previous (Split-Mode transfer) example, with an external AFE as the peripheral. If the elements arrive and depart through the AFE at a rate which does not seriously limit the bandwidth of the EMIF, then the only modification to the previous setup would be to replace the Global Address Register A value with the AFE Data-In address.[6]

If, however, servicing transmit- and receive-elements individually prevents the EMIF from allowing further accesses (either by the CPU or another DMA channel), then the bursting method should be used. When performing frame synchronized transfers, two DMA channels must be used, as split-mode transfers do not allow bursting.

For this example system, assume there is an input FIFO and an output FIFO, which are each capable of holding 1k 32-bit elements (one frame size). An external interrupt (EXT_INT4) selects when the frame of data is ready to be read from the input FIFO. The output FIFO will be written to as soon as the input frame is completed. In this fashion the input and output transfer rates will be identical.[7]

The AFE is mapped into CE0 space (Map 1), with the address of the input FIFO at 0x00400000, and the address of the output FIFO at 0x0040004. For the channel which services the input data the following values must be assigned to the DMA registers:

---

[5] Usually an external interrupt or a timer interrupt synchronizes the DMA channel, depending on whether the data transfer is internally or externally mastered.

[6] This assumes that the Data-Out address is one word size above the Data-In address.

[7] The input data transfer and the output data transfer could both have external synchronization, as well.

Primary Control Register = 0x06010040

Secondary Control Register = 0x00000008

Source Address Register = 0x00400000

Destination Address Register = 0x80000000

Transfer Counter Register = 0x00010400

The individual fields of the Primary Control Register are shown in Figure 15, the fields of the Secondary Control Register are shown in Figure 16, and the fields of the Transfer Counter Register are shown in Figure 17.

Figure 15: Primary Control Register Setup for Frame-Synchronized Data Transfer Example

31 30	29 28	27	26	25	24	23 19	18 16
00	00	0	1	1	0	00000	001
DST RELOAD	SRC RELOAD	EMOD	FS	TCINT	PRI	WSYNC	RSYNC

15 14	13	12	11 10	9 8	7 6	5 4	3 2	1 0
00	0	0	00	00	01	00	00	00
RSYNC	INDX	CNT RELOAD	SPLIT	ESIZE	DST DIR	SRC DIR	STATUS	START

Figure 16: Secondary Control Register Setup for Frame-Synchronized Data Transfer Example

31 19	18 16
XXXXXXXXXXXXX	000
Reserved	DMAC

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
WSYNC CLR	WSYNC STAT	RSYNC CLR	RSYNC STAT	WDROP IE	WDROP COND	RDROP IE	RDROP COND	BLOCK IE	BLOCK COND	LAST IE	LAST COND	FRAME IE	FRAME COND	SX IE	SX COND

Figure 17: Transfer Counter Register Setup for Frame-Synchronized Data Transfer Example

31 16	15 0
0x0001	0x0400
FRAME COUNT	ELEMENT COUNT

The settings of 01b in the DST DIR bitfield will cause the DMA channel to increment the destination address by one element size (4 bytes in this example) following each element. Since this is a frame synchronized transfer (FS = 1), the entire frame of elements from the source address will be read from the AFE as soon as the Read Synchronization event (RSYNC = EXT_INT4) is received. Setting TCINT to 1 will cause the DMA channel to generate an interrupt, which will occur at the end of a frame (FRAME IE = 1). This interrupt will be used to initiate the transfer to the output FIFO by another DMA channel.

For the DMA channel which services the output data the following values must be assigned to the DMA registers:

Primary Control Register = 0x0402X010[8]

Source Address Register = 0x80001000

Destination Address Register = 0x00400004

Transfer Counter Register = 0x00010400

The individual fields of the Primary Control Register are shown in Figure 18, and the fields of the Transfer Counter Register are shown in Figure 19.

Figure 18: Primary Control Register Setup for Frame-Synchronized Data Transfer Example (2)

31 30	29 28	27	26	25	24	23 19	18 16
00	00	0	1	0	0	00000	010
DST RELOAD	SRC RELOAD	EMOD	FS	TCINT	PRI	WSYNC	RSYNC

15 14	13	12	11 10	9 8	7 6	5 4	3 2	1 0
{*n*}[9]	0	0	00	00	00	01	00	00
RSYNC	INDX	CNT RELOAD	SPLIT	ESIZE	DST DIR	SRC DIR	STATUS	START

Figure 19: Transfer Counter Register Setup for Frame-Synchronized Data Transfer Example (2)

31 16	15 0
0x0001	0x0400
FRAME COUNT	ELEMENT COUNT

---

[8] The value of X is equal to 4**n*, where *n* is the DMA channel number servicing the input data. See note 9.

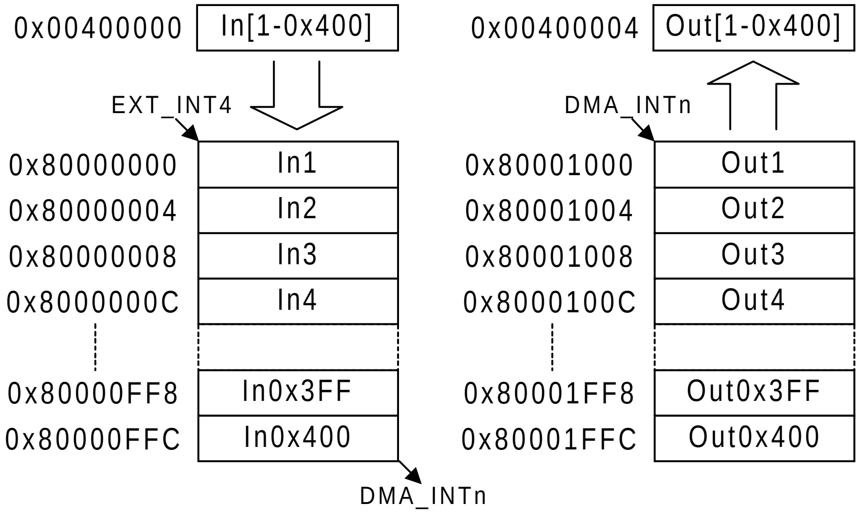
[9] The value of RSYNC for this channel depends on the channel servicing the input transfer, where *n* equals the DMA channel number.

The settings of 01b in the SRC DIR bitfield will cause the DMA channel to increment the source address by one element size (4 bytes in this example) following each element. Since this is a frame synchronized transfer (FS = 1), the entire frame of elements from the source address will be written to the AFE as soon as the Read Synchronization event (RSYNC = DMA_INT*n*) is received.

In order to initiate the two transfers, a value of 01b needs to be written to the START bitfield of each channel's Primary Control Register.

Figure 20 shows the transfer that the above setup will perform.

Figure 20: Frame-Synchronized Transfer Example Diagram



Endian Mode Considerations

When using a peripheral for element sizes other than 32-bits, endianness plays an important role. This is usually only true for the McBSPs, as external peripherals typically match the endianness of the entire system.

The McBSPs are inherently little endian. The DXR and DRR, being registers, have the Least Significant Byte (LSB) on the right and the Most Significant Byte (MSB) on the left (conceptually). The DXR and DRR of the McBSPs are depicted in Figure 21 and Figure 22, with the byte ordering for each endian mode shown.

Figure 21: DXR Byte Locations

	31			0
Little Endian	Byte 3	Byte 2	Byte 1	Byte 0
Big Endian	Byte 0	Byte 1	Byte 2	Byte 3

Figure 22: DRR Byte Locations

	31			0
Little Endian	Byte 3	Byte 2	Byte 1	Byte 0
Big Endian	Byte 0	Byte 1	Byte 2	Byte 3

The DXR always transmits assuming the element is located with its LSB at bit 0. When in little-endian mode, this is the base address of the DXR, so no matter what the element size, a write to the DXR base address will properly align the element. In big-endian mode, however, this is the upper portion of the register. Depending on the size of the element, the write must be made to the address of either Byte 2 (16-bit) or to Byte 3 (8-bit).

The DRR is configurable to either right- or left-justify the incoming data. The justification will determine the source address of the data element. For right-justified data (default), the source address will be Byte 0 in little-endian mode, and Byte 2 (16-bit) or Byte 3 (8-bit) in big-endian mode. For left-justified data, the reverse will be true.

Table 3 shows the possible endian mode, element size, and DRR justification combinations that may be encountered in a system. Only the source and destination addresses are given for each. All of the necessary configurations described previously still apply.

Table 3: Possible DMA Source and Destination Address for Servicing McBSP0[10]

Element Size	Endian Mode	DRR Justification	Source Address	Destination Address
8-bit	Little	Right	0x018C0000	0x018C0004
		Left	0x018C0003	0x018C0004
	Big	Right	0x018C0003	0x018C0007
		Left	0x018C0000	0x018C0007
16-bit	Little	Right	0x018C0000	0x018C0004
		Left	0x018C0002	0x018C0004
	Big	Right	0x018C0002	0x018C0006
		Left	0x018C0000	0x018C0006
32-bit	Little	Right	0x018C0000	0x018C0004
		Left	0x018C0000	0x018C0004
	Big	Right	0x018C0000	0x018C0004
		Left	0x018C0000	0x018C0004

---

[10] Note that the Source Addresses and Destination Addresses are identical for both the big- and little-endian modes when transferring 32-bit elements.

Repetitive DMA operation

When data flow requires that a peripheral be repetitively serviced throughout device operation, or when program sections are to be continuously swapped in and out of program memory, it is appropriate to configure the DMA to reprogram itself for subsequent transfers. This may be accomplished by running the DMA channel in auto-initialization mode and providing reload values for the source and destination addresses and for the transfer counter.

Once the DMA is programmed to run repetitively an important concern is how to effectively buffer the incoming and outgoing data. To keep memory usage to a minimum, the DMA should write over old data, instead of storing information that is no longer useful. This type of buffering is considered “circular”, as data is continuously cycling through the same memory space.

When a high throughput is required of the device, and time cannot be spared waiting on the DMA to transfer a frame of data to the device before processing, a ping-pong buffering system should be used. This scheme requires slightly more complicated reload settings, but allows for the CPU to be processing data while the DMA is transferring new data on-chip, and old data off-chip.

The following examples demonstrate these two buffering schemes.

Transferring Data To and From Circular Buffers

In many DSP applications, data is stored in on-chip circular buffers, in which new data is written directly over old data, so that a minimum amount of memory space will be consumed by an application. For such an application, it is desired for the DMA to continually bring frames of data to the same block of data memory. This may easily be accomplished by configuring the DMA channel(s) for the initial block move, and taking a few extra steps to allow for the DMA to reset itself after each transfer.

As an example of how to use circular buffering, the Frame-Synchronized Data Transfer Example will be modified to run continuously, with the input and output data buffers being reused for each frame of data. The address of the input FIFO is 0x00400000, and the address of the output FIFO is 0x0040004.

For the DMA channel that services the input data the Primary Control Register value must be modified to allow the channel to use an index for the Destination Address. Global Index Register A will be used to return the Destination Address to the beginning of the frame. The control registers for this channel should be:

Primary Control Register = 0x060100C0

Secondary Control Register = 0x00000008

Source Address Register = 0x00400000

Destination Address Register = 0x80000000

Transfer Counter Register = 0x00010400

Global Index Register A = 0xF0040004

The individual fields of the Primary Control Register are shown in Figure 23, the fields of the Secondary Control Register are shown in Figure 24, the fields of the Transfer Counter Register are shown in Figure 25, and the Global Index Register A is shown in Figure 26.

Figure 23: Primary Control Register Setup for Circular Buffering Example

31 30	29 28	27	26	25	24	23 19	18 16
00	00	0	1	1	0	00000	001
DST RELOAD	SRC RELOAD	EMOD	FS	TCINT	PRI	WSYNC	RSYNC

15 14	13	12	11 10	9 8	7 6	5 4	3 2	1 0
00	0	0	00	00	11	00	00	00
RSYNC	INDX	CNT RELOAD	SPLIT	ESIZE	DST DIR	SRC DIR	STATUS	START

Figure 24: Secondary Control Register Setup for Circular Buffering Example

31 19	18 16
XXXXXXXXXXXXX	000
Reserved	DMAC

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	0	0	0	0	1	0	0	0
WSYNC CLR	WSYNC STAT	RSYNC CLR	RSYNC STAT	WDROP IE	WDROP COND	RDROP IE	RDROP COND	BLOCK IE	BLOCK COND	LAST IE	LAST COND	FRAME IE	FRAME COND	SX IE	SX COND

Figure 25: Transfer Counter Register Setup for Circular Buffering Example

31 16	15 0
0x0001	0x0400
FRAME COUNT	ELEMENT COUNT

Figure 26: Transfer Counter Register Setup for Circular Buffering Example

31 16	15 0
0xF004	0x0004
FRAME INDEX	ELEMENT INDEX

The settings of 11b in the DST DIR bitfield will cause the DMA channel to modify the destination address using Global Index Register A (INDEX = 0). Following each element the address will be modified by four bytes using ELEMENT INDEX, and after each frame the destination address will return to 0x80000000 using FRAME INDEX. Since this is a frame synchronized transfer (FS = 1), the entire frame of elements from the source address will be read from the AFE as soon as the Read Synchronization event (RSYNC = EXT_INT4) is received. Setting TCINT to 1 will cause the DMA channel to generate an interrupt, which will occur at the end of a frame (FRAME IE = 1). This interrupt will be used to initiate the transfer to the output FIFO by another DMA channel.[11]

For the DMA channel that services the output data the Primary Control Register value must be modified to allow the channel to use an index for the Source Address. Global Index Register A will be used to return the Source Address to the beginning of the frame. The control registers for this channel should be:

Primary Control Register = 0x0402X030[12]

Source Address Register = 0x80001000

Destination Address Register = 0x00400004

Transfer Counter Register = 0x00010400

The individual fields of the Primary Control Register are shown in Figure 27, and the fields of the Transfer Counter Register are shown in Figure 28.

---

[11] Note that the FRAME COND bit must be manually cleared following each frame transfer by this channel. See the section on DMA Interrupt Service Routines for information on how to do this.

[12] The value of X is equal to 4**n*, where *n* is the DMA channel number servicing the input data. See note 13.

Figure 27: Primary Control Register Setup for Circular Buffering Example (2)

31 30	29 28	27	26	25	24	23 19	18 16
00	00	0	1	0	0	00000	010
DST RELOAD	SRC RELOAD	EMOD	FS	TCINT	PRI	WSYNC	RSYNC

15 14	13	12	11 10	9 8	7 6	5 4	3 2	1 0
{*n*}[13]	0	0	00	00	00	11	00	00
RSYNC	INDX	CNT RELOAD	SPLIT	ESIZE	DST DIR	SRC DIR	STATUS	START

Figure 28: Transfer Counter Register Setup for Circular Buffering Example (2)

31 16	15 0
0x0001	0x0400
FRAME COUNT	ELEMENT COUNT

The settings of 11b in the SRC DIR bitfield will cause the DMA channel to modify the source address using Global Index Register A (INDEX = 0). Following each element the address will be modified by four bytes using ELEMENT INDEX, and after each frame the destination address will return to 0x80001000 using FRAME INDEX. Since this is a frame synchronized transfer (FS = 1), the entire frame of elements from the source address will be written to the AFE as soon as the Read Synchronization event (RSYNC = DMA_INT*n*) is received.

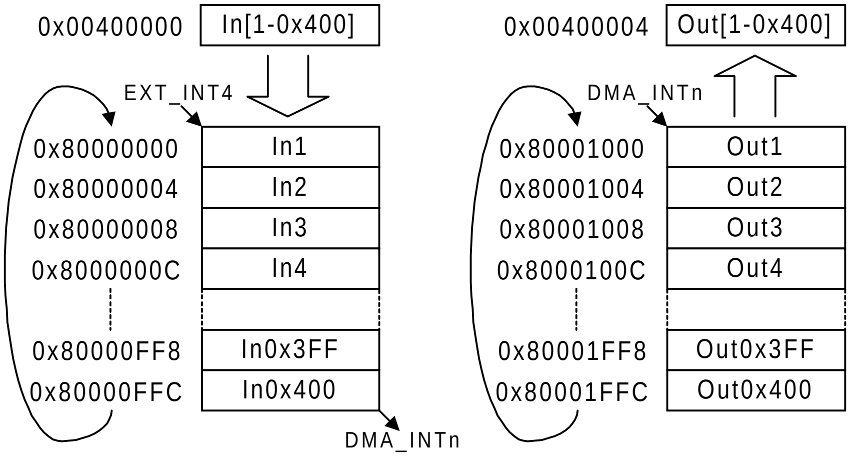
In order to initiate the two transfers, a value of 11b needs to be written to the START bitfield of each channel's Primary Control Register. Since the output buffer will not have valid data in it until after the CPU has processed the initial input buffer, the DMA channel servicing the output should not be started until after the first frame completes.

Figure 29 shows the transfer that the above setup will perform.

---

[13] The value of RSYNC for this channel depends on the channel servicing the input transfer, where *n* equals the DMA channel number.

Figure 29: Circular Buffer Example Diagram



Ping-Pong Transfer Example

The previous example provides an easy way to renew the buffer of data being operated on by the CPU. One drawback associated with the single pair of input and output buffers is that while the DMA is filling the input buffer or reading from the output buffer, the CPU cannot be accessing the same space.[14] One way to ensure that the CPU will not be operating on an incorrect set of data, or changing data which has not yet been moved off-chip, is to have a dual-buffering scheme. This means simply that there are two input buffers, and two output buffers. This will double the amount of internal memory consumed, but will greatly increase throughput for applications for which the CPU is dedicated to converting the input data set to an output data set, with few breaks. For this type of application, the DMA will be moving data to and from one pair of input/output buffers, while the CPU is operating on the other pair. As soon as both the CPU and DMA are finished they will switch input/output buffer pairs.

---

[14] This is true unless care is taken that the CPU is always ahead of the input data and behind the output data.

Consider a variation of the previous transfer in which there are two 1k-word input buffers and two 1k-word output buffers in data memory. The DMA will transfer 1k block of 32-bit words to one input buffer, located at 0x80000000, and a 1k block of 32-bit words from one output buffer, located at 0x80002000. After these sets of data are transferred, a new block is transferred to a second input buffer, at 0x80001000, and a new block is transferred from a second output buffer, at 0x80003000. The next pair of transfers will return to the original input/output pair. The CPU will compute data located in the first input buffer and store the results in the first output buffer following the first DMA transfer. Once completed, the CPU will use the second input buffer, storing the results in the second output buffer. The CPU will then switch back to the first pair and continue. The control registers from the previous example will be modified such that the source and destination addresses are reloaded to their original values following each block (two frames) of data transferred.

For the DMA channel that services the input data the Primary Control Register value must be modified to allow the channel to post-increment the Destination Address. Global Address Register B will be used to return the Destination Address to the beginning of the first buffer. The control registers for this channel should be:

Primary Control Register = 0x46010040

Secondary Control Register = 0x00000088

Source Address Register = 0x00400000

Destination Address Register = 0x80000000

Transfer Counter Register = 0x00020400

Global Address Register B = 0x80000000

The individual fields of the Primary Control Register are shown in Figure 30, the fields of the Secondary Control Register are shown in Figure 31, and the fields of the Transfer Counter Register are shown in Figure 32.

Figure 30: Primary Control Register Setup for Ping-Pong Transfer Example

31 30	29 28	27	26	25	24	23 19	18 16
01	00	0	1	1	0	00000	001
DST RELOAD	SRC RELOAD	EMOD	FS	TCINT	PRI	WSYNC	RSYNC

15 14	13	12	11 10	9 8	7 6	5 4	3 2	1 0
00	0	0	00	00	01	00	00	00
RSYNC	INDX	CNT RELOAD	SPLIT	ESIZE	DST DIR	SRC DIR	STATUS	START

Figure 31: Secondary Control Register Setup for Ping-Pong Transfer Example

31 19	18 16
XXXXXXXXXXXXX	000
Reserved	DMAC

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	1	0	0	0	1	0	0	0
WSYNC CLR	WSYNC STAT	RSYNC CLR	RSYNC STAT	WDROP IE	WDROP COND	RDROP IE	RDROP COND	BLOCK IE	BLOCK COND	LAST IE	LAST COND	FRAME IE	FRAME COND	SX IE	SX COND

Figure 32: Transfer Counter Register Setup for Ping-Pong Transfer Example

31 16	15 0
0x0002	0x0400
FRAME COUNT	ELEMENT COUNT

The settings of 01b in the DST DIR bitfield will cause the DMA channel to increment the destination address following each element. Since this is a frame-synchronized transfer (FS = 1), an entire frame of elements will be read from the AFE as soon as the Read Synchronization event (RSYNC = EXT_INT4) is received. Setting TCINT to 1 will cause the DMA channel to generate an interrupt, which will occur at the end of a frame (FRAME IE = 1). This interrupt will be used to initiate the transfer to the output FIFO by another DMA channel.[15] Following the second frame of each block, the destination address will be reloaded to 0x80000000, to allow the DMA to overwrite old data with new.

---

[15] Note that the FRAME COND bit must be manually cleared following each frame transfer by this channel. See the section on DMA Interrupt Service Routines for information on how to do this.

For the DMA channel that services the output data the Primary Control Register value must be modified to allow the channel to post-increment the Source Address. Global Address Register C will be used to return the Source Address to the beginning of the first output buffer. The control registers for this channel should be:

Primary Control Register = 0x2602X010[16]

Source Address Register = 0x80002000

Destination Address Register = 0x00400004

Transfer Counter Register = 0x00020400

Global Address Register C = 0x80002000

The individual fields of the Primary Control Register are shown in Figure 33, and the fields of the Transfer Counter Register are shown in Figure 34.

Figure 33: Primary Control Register Setup for Ping-Pong Transfer Example (2)

31 30	29 28	27	26	25	24	23 19	18 16
00	10	0	1	0	0	00000	010
DST RELOAD	SRC RELOAD	EMOD	FS	TCINT	PRI	WSYNC	RSYNC

15 14	13	12	11 10	9 8	7 6	5 4	3 2	1 0
{*n*}[17]	0	0	00	00	00	01	00	00
RSYNC	INDX	CNT RELOAD	SPLIT	ESIZE	DST DIR	SRC DIR	STATUS	START

Figure 34: Transfer Counter Register Setup for Ping-Pong Transfer Example (2)

31 16	15 0
0x0002	0x0400
FRAME COUNT	ELEMENT COUNT

---

[16] The value of X is equal to 4**n*, where *n* is the DMA channel number servicing the input data. See note 17.

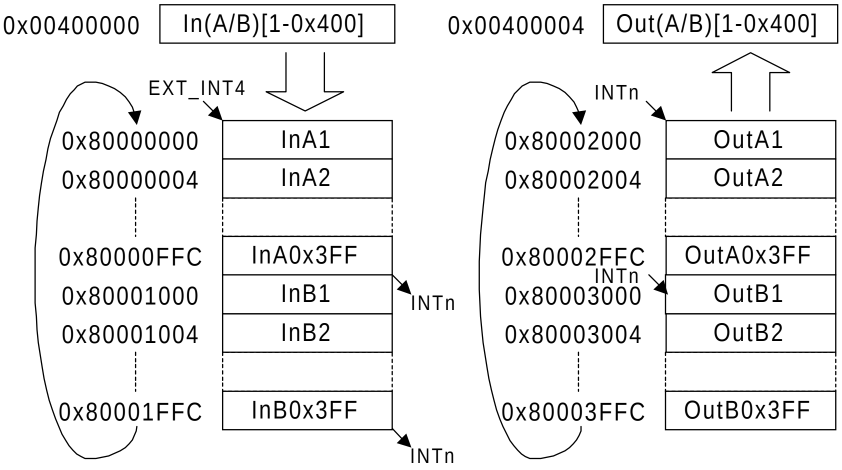
[17] The value of RSYNC for this channel depends on the channel servicing the input transfer, where *n* equals the DMA channel number.

The settings of 01b in the SRC DIR bitfield will cause the DMA channel increment the source address following each element. Since this is a frame-synchronized transfer (FS = 1), an entire frame of elements will be written to the AFE as soon as the Read Synchronization event (RSYNC = DMA_INT*n*) is received. Following the second frame of each block, the Source Address will be reloaded to 0x80002000.

In order to initiate the two transfers, a value of 11b needs to be written to the START bitfield of each channel's Primary Control Register. Since the output buffers will not have valid data in them until after the CPU has processed the initial two input buffers, the DMA channel servicing the output should not be started until after the first block completes. By enabling the BLOCK IE to generate an interrupt, the DMA channel servicing the output data may be started following the first input block (two frames).[18]

Figure 35 shows the transfer that the above setup will perform.

Figure 35: Ping-Pong Buffer Example Diagram



---

[18] An example Interrupt Service Routine that does this and clears the FRAME COND bit is given in the DMA Interrupt Service Routines section.

Program Paging Example

The methodology behind the Ping-Pong data transfer may also be applied to program paging. When a program requires more than 64kbytes of memory, and it is not desired to operate either from external memory or in cache mode, then it becomes necessary to implement program paging. Providing a DMA channel to bring in a block of code from external memory to internal program memory does this. In order to maintain efficiency with the CPU, it is necessary to have at least two sections of program space in the internal program memory. By doing so, existing code may be executed while new code is brought to the device.

In order to facilitate an efficient paging scheme, a program should be divided into sections, with the sections to be paged occupying the same amount of memory space.[19] These sections should then be stored in known external memory locations so that the DMA is capable of retrieving specific blocks of code. A typical breakdown of code would include:

- Interrupt Service Table which resides in either program memory or external memory[20]
- Main block of code which resides in program memory
- Program pages which reside in external memory

Bringing data into internal program memory with the DMA differs significantly from accessing internal data memory. The end result is that transfers to program memory are slower than transfers to data memory, and these transfers should be given time to complete. The primary factor in enabling a program page to be transferred more effectively is code parallelism. The less parallel a program is, the more frequent the accesses to the internal program memory space by the DMA can be. Any optimized loops in a program will slow the transfer.

The sample program in this example will include the following sections:

- Interrupt Service Table (1k), linked to the base address of internal program memory: 0x00000000 (Map 1). This section contains Interrupt

---

[19] For information on how to create program sections and organize them in memory, see the TMS320C6x Assembly Language Tools User's Guide and the TMS320C6x Optimizing C Compiler User's Guide.

[20] The location of the Interrupt Vector Table should depend upon the frequency of interrupts that need to be serviced. If interrupts were frequent, then it would be more efficient for the IST to be in internal program memory.

- Main block of code (15k), linked to 0x00000400 in internal program memory. This section contains main subroutine,
- Initialization section linked to 0x0000A000. This block of code sets up the EMIF, interrupts, data initialization, and any DMA transfers to be performed during the program that may be configured ahead of time. This section will be overwritten as it is only used once.
- Page1 section linked to external memory location 0x20000000 (CE2), with run-time location set to 0x00006000. This page of code will be brought into internal program memory multiple times throughout the program execution.
- Page2 section linked to 0x20004000, with run-time location set to 0x0000A000. This page of code will be brought into internal program memory multiple times throughout the program execution.
- Page3 section linked to 0x20008000, with run-time location set to 0x00006000. This page of code will be brought into internal program memory multiple times throughout the program execution.
- Page4 section linked to 0x2000C000, with run-time location set to 0x0000A000. This page of code will be brought into internal program memory multiple times throughout the program execution.

Each page (1-4) listed above is of length 16k (0x4000), and each page branches to the next, sequentially. In order to facilitate this, a DMA channel should be set up by the initialization code to transfer pages 1 through 4 to their run-time program space.

Program paging is typically a background transfer, as it is not desirable to interfere with the servicing of peripherals or other data transfers. Paging is normally done using a low-priority channel.

For this example, the DMA channel is set up as follows:

Primary Control Register = 0x9601A050

Secondary Control Register = 0x00000080

Source Address Register = 0x20000000

Destination Address Register = 0x00006000

Transfer Counter Register = 0x00011000

Global Address Register B = 0x20006000

Global Address Register C = 0x0000A000

Global Reload Register A = 0x00011000

The individual fields of the Primary Control Register are shown in Figure 36, the fields of the Secondary Control Register are shown in Figure 37, and the fields of the Transfer Counter Register are shown in Figure 38.

Figure 36: Primary Control Register Setup for Program Paging Example

31 30	29 28	27	26	25	24	23 19	18 16
10	01	0	1	1	0	00000	001
DST RELOAD	SRC RELOAD	EMOD	FS	TCINT	PRI	WSYNC	RSYNC

15 14	13	12	11 10	9 8	7 6	5 4	3 2	1 0
10	1	0	00	00	01	01	00	00
RSYNC	INDX	CNT RELOAD	SPLIT	ESIZE	DST DIR	SRC DIR	STATUS	START

Figure 37: Secondary Control Register Setup for Program Paging Example

31 19	18 16
XXXXXXXXXXXXX	000
Reserved	DMAC

15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0
WSYNC CLR	WSYNC STAT	RSYNC CLR	RSYNC STAT	WDROP IE	WDROP COND	RDROP IE	RDROP COND	BLOCK IE	BLOCK COND	LAST IE	LAST COND	FRAME IE	FRAME COND	SX IE	SX COND

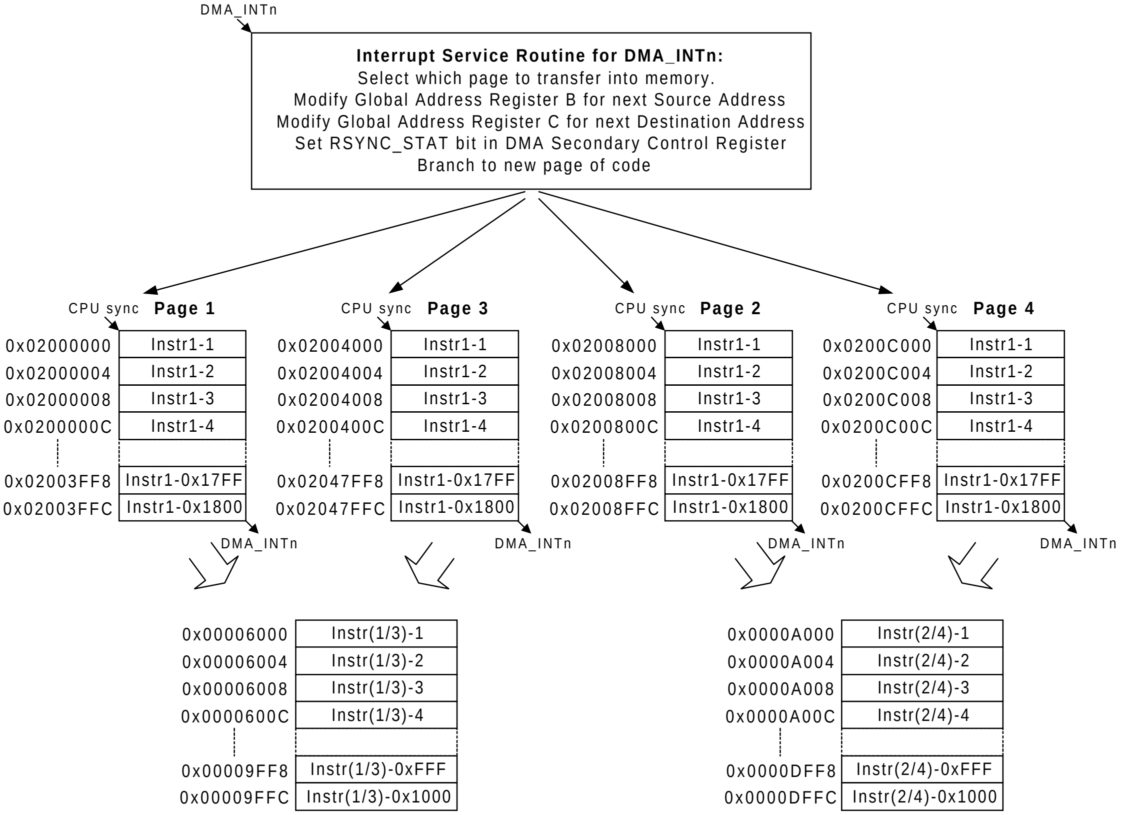
Figure 38: Transfer Counter Register Setup for Program Paging Example

31 16	15 0
0x0001	0x1800
FRAME COUNT	ELEMENT COUNT

The settings of 01b in the DST DIR and SRC DIR bitfields will cause the DMA channel to increment the destination and source addresses following each element. Since this is a frame-synchronized transfer (FS = 1), an entire page will be transferred to program memory as soon as the Read Synchronization event (RSYNC = EXT_INT6) is received. Setting TCINT to 1 will cause the DMA channel to generate an interrupt, which will occur at the end of a block (BLOCK_IE = 1). This interrupt will be used to let the CPU know that valid code is present. Following each block, Global Reload Registers should be set with the destination and source addresses for the subsequent transfer. External interrupt EXT_INT6 should not actually be used during program execution. Instead, the CPU should directly set the RSYNC STAT bit in the channel's Secondary Control Register to initiate each transfer. This will provide control between the CPU and the DMA. Each time the CPU finishes executing a page, it should set the RSYNC STAT bit, then poll the Interrupt Flag for the interrupt number of the DMA channel. If it is known that the CPU will complete before the DMA every time then CPU may be placed in IDLE to decrease the transfer time.

Figure 39 shows the transfer that the above setup will perform.

Figure 39: Program-Paging Example Diagram



DMA Interrupt Service Routines

Through configuration of a DMA channel's Primary and Secondary Control Register, it is possible for each DMA channel to interrupt the CPU when one or more conditions occur. When any of the enabled conditions occur, the interrupt flag for the DMA channel will be set. If this interrupt is enabled in the Interrupt Enable Register (IER), then the interrupt will be serviced. The conditions that may be used are given in Table 4.

Table 4: DMA Channel Condition Descriptions

Bitfield	Event	Occurs…
BLOCK	Block transfer complete	After the last write transfer in a block transfer is written to memory.
FRAME	Frame complete	After the last write transfer in each frame is written to memory.
LAST	Last frame	After all counter adjustments for the next to last frame in a block transfer complete.
WDROP RDROP	Dropped read/write synchronization	If a subsequent synchronization event occurs before the last one is cleared.
SX	Split transmit overrun receive	If the split-mode is enabled, and transmit-element transfers get seven or more element transfers ahead of receive-element transfers.

The IE bits in the channel's Secondary Control Register must be set for each condition to generate an interrupt to the CPU. The TCINT bit in the channel's Primary Control Register must also be set. This will cause an interrupt to occur whenever the enabled condition transitions from a "0" to a "1", which will be reported in the COND bitfields of the channel's Secondary Control Register.

If the IE bit for a condition is enabled, then the CPU must manually clear the COND bit in order to receive subsequent interrupts. This is to avoid confusion in the case that multiple events trigger the same interrupt. The most common way to perform this is to have an Interrupt Service Routine (ISR) which services each DMA Channel in use.[21] ISRs range in function from simplistic to complex, depending on the application. They are typically designed to be as short as possible, so that little time is taken away from the processor.

In order to enable a DMA-generated interrupt to be taken, several steps must be taken by the CPU. The Global Interrupt Enable (GIE) bit must be set in the Control Status Register (CSR) and the appropriate interrupt number's Interrupt Enable (IE*n*) bit and the Non-Maskable Interrupt Enable (NMIE) bit must be set in the Interrupt Enable Register (IER). The setting of NMIE is to prevent the processor to be interrupted until it is fully out of reset, and no interrupts may be taken until this is done. The GIE bit globally enables any enabled interrupt to be serviced by the CPU. This bit may be cleared to protect certain routines. The IE*n* bit is used to enable the specific interrupt number of the DMA channel in use.

The most common DMA Conditions used to interrupt the CPU are the BLOCK, FRAME, and LAST conditions. These are used primarily to modify RELOAD values in the DMA, or to update global variables used within a program. These are for planned services that occur during program execution. The remaining Conditions are used to service unplanned situations. (R/W)DROP is used in the instance that a synchronized transfer was skipped, and the SX Condition is used in the case that a split-mode transfer is not symmetric. An ISR for a DMA channel may address any number of these Conditions.

A typical sequence of events in servicing a DMA interrupt is:

- Read the Secondary Control Register
- Check COND bits to see which Condition generated the interrupt
- Clear the Condition(s) by writing “0” to the COND bits
- Write the Secondary Control Register[22]
- Perform necessary tasks to service the Condition

---

[21] For information on how to set up CPU interrupts, see the TMS320C62xx Peripherals Reference Guide and the TMS320C62x/C67x CPU and Instruction Set Reference Guide.

[22] In a synchronized transfer, it is a good idea to mask the RSYNC STAT and WSYNC STAT bits. If a synchronization event is serviced during the ISR, then writing a “1” to either may cause a spurious synchronization event. Writing a “0” has no effect.

- Resume program execution

An example of a basic ISR is one that could be used in the Ping-Pong Transfer Example. In that example, it was desired to initiate the second DMA channel (servicing the output data) after the second input frame had been completed. It was also required that the FRAME COND bit be cleared following each frame, as the FRAME IE bit is set. The following C code will perform what is necessary for the case where DMA channel 1 is servicing the input data and DMA channel 2 is servicing the output data:

```
/* DMA Channel 1 Interrupt Service Routine will
execute upon completion of a Frame Transfer by
Channel 1. Since Channel 1 is servicing the input
data, when it completes its transfer the CPU will
be free to begin executing code. */

interrupt void DMA_Ch1_ISR(void)

{

unsigned int sec_ctrl;

/* Read Channel 1 Secondary Control Register */

  sec_ctrl = REG_READ(DMA1_SECONDARY_CTRL_ADDR);

/* If the second frame has completed, start DMA
Channel 2 in auto-initialization mode, then clear
the Block Condition bit and disable Block Interrupt
Enable. */

  if (GET_BIT(&sec_ctrl, BLOCK_COND){

      DMA_AUTO_START(DMA_Ch2);

      RESET_BIT(&sec_ctrl, BLOCK_COND);

      RESET_BIT(&sec_ctrl, BLOCK_IE);

  }

/* Clear the FRAME COND bit in the DMA Channel 1
Secondary Control Register */

  RESET_BIT(&sec_ctrl, FRAME_COND);

  REG_WRITE(DMA1_SECONDARY_CTRL_ADDR, sec_ctrl);

} /* End DMA_Ch1_ISR */
```

Conclusion

The TMS320C6x DMA is a versatile tool that may be used to perform data transfers throughout device operation, with little setup required. Through proper initialization, both simple and complex data transfers may be run concurrently to provide data to the CPU, and to transmit data to external devices as well. The examples provided here offer some of the more common DMA applications for a system. Many of these transfers have multiple setups possible to accomplish the same tasks. The best way to set up multiple transfers using more than one DMA channel will depend on the combinations desired. It may be necessary to use a single channel to perform more than one type of transfer, using the CPU to continuously reprogram the channel. It is typically possible to configure those transfers that will occur throughout device operation only once, which will free the CPU to perform its application with little interruption.

Appendix.A

Included in the Appendix.A are code segments for each of the DMA transfer examples provided in this document. The following code may be modified as required for a given system, and many of these transfers may be used within the same system to perform multiple transfers.

Block Move Example Code

```
/* Set up the DMA Control Registers to perform the data transfer   */
/* of a block or data.                                             */
void
run_DMA(void)
{
unsigned int   dma_pri_ctrl   = 0;
unsigned int   dma_sec_ctrl   = 0;
unsigned int   dma_src_addr   = 0;
unsigned int   dma_dst_addr   = 0;
unsigned int   dma_tcnt       = 0;
unsigned int   dma_gcr        = 0;
unsigned int   dma_gcra       = 0;
unsigned int   dma_gcrb       = 0;
unsigned int   dma_gndxa      = 0;
unsigned int   dma_gndxb      = 0;
unsigned int   dma_gaddra     = 0;
unsigned int   dma_gaddrb     = 0;
unsigned int   dma_gaddrc     = 0;
unsigned int   dma_gaddrd     = 0;


      /* Transfer a block of size = XFER_SIZE from DBLOCK1 to   */
      /* DBLOCK2.                                               */

      /* Reset DMA Control Registers */
      dma_reset();

      /* Set up Global Configuration Registers for the DMA */
      dma_gcra = (unsigned int)reload_elm;
      dma_global_init(dma_gcr,dma_gcra,dma_gcrb,dma_gndxa,dma_gndxb,
                      dma_gaddra,dma_gaddrb,dma_gaddrc,dma_gaddrd);
      /* Set up DMA Ch2 to write 256k words (1MB) from CE2 to
         external AFE in CE0 */
      /* Set up DMA Primary Control Register */
      LOAD_FIELD(&dma_pri_ctrl, DMA_RELOAD_NONE, DST_RELOAD, DST_RELOAD_SZ);
      LOAD_FIELD(&dma_pri_ctrl, DMA_RELOAD_NONE, SRC_RELOAD, SRC_RELOAD_SZ);
      LOAD_FIELD(&dma_pri_ctrl, DMA_NO_EM_HALT , EMOD      , 1            );
      LOAD_FIELD(&dma_pri_ctrl, DMA_CPU_PRI    , PRI       , 1            );
      LOAD_FIELD(&dma_pri_ctrl, SEN_NONE       , WSYNC     , WSYNC_SZ     );
      LOAD_FIELD(&dma_pri_ctrl, SEN_NONE       , RSYNC     , RSYNC_SZ     );
      LOAD_FIELD(&dma_pri_ctrl, DMA_CNT_RELOADA, CNT_RELOAD, 1            );
      LOAD_FIELD(&dma_pri_ctrl, DMA_SPLIT_DIS  , SPLIT     , SPLIT_SZ     );
      LOAD_FIELD(&dma_pri_ctrl, DMA_ESIZE32    , ESIZE     , ESIZE_SZ     );
      LOAD_FIELD(&dma_pri_ctrl, DMA_ADDR_NO_MOD, DST_DIR   , DST_DIR_SZ   );
      LOAD_FIELD(&dma_pri_ctrl, DMA_ADDR_INC   , SRC_DIR   , SRC_DIR_SZ   );

      /* Set up DMA Tranfer Count Register */
      LOAD_FIELD(&dma_tcnt, XFER_SIZE, ELEMENT_COUNT, ELEMENT_COUNT_SZ);

      /* Set up Source and Destination Address Registers */
      dma_src_addr = (unsigned int)DBLOCK1;
      dma_dst_addr = (unsigned int)DBLOCK2;

      /* Store DMA Channel 2 registers */
      dma_init(DMA_CH2,
            dma_pri_ctrl,
            dma_sec_ctrl,
            dma_src_addr,
            dma_dst_addr,
            dma_tcnt);

      /* Start DMA Ch2 Transfer */
      DMA_START(DMA_CH2);

} /* End run_DMA */
```

Extremely Large Block Move Example Code

```
/* Set up the DMA Control Registers to perform the data transfer   */
/* of the large data block                                         */
void
run_DMA(void)
{
unsigned int   dma_pri_ctrl   = 0;
unsigned int   dma_sec_ctrl   = 0;
unsigned int   dma_src_addr   = 0;
unsigned int   dma_dst_addr   = 0;
unsigned int   dma_tcnt       = 0;
unsigned int   dma_gcr        = 0;
unsigned int   dma_gcra       = 0;
unsigned int   dma_gcrb       = 0;
unsigned int   dma_gndxa      = 0;
unsigned int   dma_gndxb      = 0;
unsigned int   dma_gaddra     = 0;
unsigned int   dma_gaddrb     = 0;
unsigned int   dma_gaddrc     = 0;
unsigned int   dma_gaddrd     = 0;
unsigned int   count_reload   = 0;
unsigned int   frame_cnt      = 0;
unsigned int   initial_elm    = 0;
unsigned int   reload_elm     = 0;


      /* Establish initial count value, and reload value, based */
      /* on the transfer size (XFER_SIZE) of the large block.   */
      /* The formulas used to calculate the initial and reload  */
      /* are as follows:                                        */
      /* Initial element count = 15 LSBs of total transfer size */
      /* Reload element count = 0x8000 (bit 15)                 */
      /* Frame count = bits 15 through 30, plus 1               */
      /* NOTE: The maximum size using this method is 0x7FFF7FFF */
      /*       For even larger sizes, new formulas must be used */

      frame_cnt = (XFER_SIZE >> 15) + 1;
      initial_elm = XFER_SIZE & 0x7FFF;         /* keep 15 LSBs */
      if (!initial_elm)       /* element count of 0 not allowed */
```

```
{
      initial_elm = 0x8000;
      frame_cnt -= 1;
}
reload_elm =  0x8000;

/* Reset DMA Control Registers */
dma_reset();

/* Set up Global Configuration Registers for the DMA */
dma_gcra = (unsigned int)reload_elm;
dma_global_init(dma_gcr,dma_gcra,dma_gcrb,dma_gndxa,dma_gndxb,
                dma_gaddra,dma_gaddrb,dma_gaddrc,dma_gaddrd);

/* Set up DMA Ch2 to write 256k words (1MB) from CE2 to
   external AFE in CE0 */
/* Set up DMA Primary Control Register */
LOAD_FIELD(&dma_pri_ctrl, DMA_RELOAD_NONE, DST_RELOAD, DST_RELOAD_SZ);
LOAD_FIELD(&dma_pri_ctrl, DMA_RELOAD_NONE, SRC_RELOAD, SRC_RELOAD_SZ);
LOAD_FIELD(&dma_pri_ctrl, DMA_NO_EM_HALT , EMOD      , 1            );
LOAD_FIELD(&dma_pri_ctrl, DMA_CPU_PRI    , PRI       , 1            );
LOAD_FIELD(&dma_pri_ctrl, SEN_NONE       , WSYNC     , WSYNC_SZ     );
LOAD_FIELD(&dma_pri_ctrl, SEN_NONE       , RSYNC     , RSYNC_SZ     );
LOAD_FIELD(&dma_pri_ctrl, DMA_CNT_RELOADA, CNT_RELOAD, 1            );
LOAD_FIELD(&dma_pri_ctrl, DMA_SPLIT_DIS  , SPLIT     , SPLIT_SZ     );
LOAD_FIELD(&dma_pri_ctrl, DMA_ESIZE32    , ESIZE     , ESIZE_SZ     );
LOAD_FIELD(&dma_pri_ctrl, DMA_ADDR_NO_MOD, DST_DIR   , DST_DIR_SZ   );
LOAD_FIELD(&dma_pri_ctrl, DMA_ADDR_INC   , SRC_DIR   , SRC_DIR_SZ   );

/* Set up DMA Tranfer Count Register */
LOAD_FIELD(&dma_tcnt, frame_cnt,   FRAME_COUNT,   FRAME_COUNT_SZ  );
LOAD_FIELD(&dma_tcnt, initial_elm, ELEMENT_COUNT, ELEMENT_COUNT_SZ);

/* Set up Source and Destination Address Registers */
dma_src_addr = (unsigned int)DBLOCK;
dma_dst_addr = (unsigned int)AFEout;

/* Store DMA Channel 2 registers */
dma_init(DMA_CH2,
      dma_pri_ctrl,
            dma_sec_ctrl,
            dma_src_addr,
            dma_dst_addr,
            dma_tcnt);

      /* Start DMA Ch2 Transfer */
      DMA_START(DMA_CH2);

} /* End run_DMA */
```

Data-Sorting Example Code

```
/* Set up the DMA Control Registers to perform a column-wise sort  */
/* of data arrays located in external memory.                      */
void
run_DMA(void)
{
unsigned int   dma_pri_ctrl   = 0;
unsigned int   dma_sec_ctrl   = 0;
unsigned int   dma_src_addr   = 0;
unsigned int   dma_dst_addr   = 0;
unsigned int   dma_tcnt       = 0;
unsigned int   dma_gcr        = 0;
unsigned int   dma_gcra       = 0;
unsigned int   dma_gcrb       = 0;
unsigned int   dma_gndxa      = 0;
unsigned int   dma_gndxb      = 0;
unsigned int   dma_gaddra     = 0;
unsigned int   dma_gaddrb     = 0;
unsigned int   dma_gaddrc     = 0;
unsigned int   dma_gaddrd     = 0;
unsigned int   count_reload   = 0;
unsigned int   el_index       = 0;
unsigned int   fr_index       = 0;
unsigned int   esize          = 0;

      /* Reset DMA Control Registers */
      dma_reset();

      /* Calculate index values, as well as the ESIZE, based on the  */
      /* number of elements per frame (EL_COUNT), the number of      */
      /* frames per block (FR_COUNT), and the number of bytes of each*/
      /* element (EL_SIZE).                                          */
      el_index = FR_COUNT * EL_SIZE;
      fr_index = -(((EL_COUNT - 1) * FR_COUNT) - 1) * EL_SIZE;
      if (EL_SIZE == 1) esize = 2;
      else if (EL_SIZE == 2) esize = 1;
      else esize = 0;
```

```
/* Set up Global Configuration Registers for the DMA */
LOAD_FIELD(&dma_gcra , FR_COUNT, FRAME_COUNT  , FRAME_COUNT_SZ  );
LOAD_FIELD(&dma_gcra , EL_COUNT, ELEMENT_COUNT, ELEMENT_COUNT_SZ);
LOAD_FIELD(&dma_gndxa, fr_index, FRAME_INDEX  , FRAME_INDEX_SZ  );
LOAD_FIELD(&dma_gndxa, el_index, ELEMENT_INDEX, ELEMENT_INDEX_SZ);
dma_global_init(dma_gcr,dma_gcra,dma_gcrb,dma_gndxa,dma_gndxb,
                dma_gaddra,dma_gaddrb,dma_gaddrc,dma_gaddrd);

/* Set up DMA Ch1 to perform a block transfer of XFER_SIZE elements */
/*   from AFE to INBUFFER */
/* Set up DMA Primary Control Register */
LOAD_FIELD(&dma_pri_ctrl, DMA_RELOAD_NONE, DST_RELOAD, DST_RELOAD_SZ);
LOAD_FIELD(&dma_pri_ctrl, DMA_RELOAD_NONE, SRC_RELOAD, SRC_RELOAD_SZ);
LOAD_FIELD(&dma_pri_ctrl, DMA_NO_EM_HALT , EMOD      , 1            );
LOAD_FIELD(&dma_pri_ctrl, DMA_CPU_PRI    , PRI       , 1            );
LOAD_FIELD(&dma_pri_ctrl, SEN_NONE       , WSYNC     , WSYNC_SZ     );
LOAD_FIELD(&dma_pri_ctrl, SEN_NONE       , RSYNC     , RSYNC_SZ     );
LOAD_FIELD(&dma_pri_ctrl, DMA_INDXA      , INDEX     , 1            );
LOAD_FIELD(&dma_pri_ctrl, DMA_CNT_RELOADA, CNT_RELOAD, 1            );
LOAD_FIELD(&dma_pri_ctrl, DMA_SPLIT_DIS  , SPLIT     , SPLIT_SZ     );
LOAD_FIELD(&dma_pri_ctrl, esize          , ESIZE     , ESIZE_SZ     );
LOAD_FIELD(&dma_pri_ctrl, DMA_ADDR_INDX  , DST_DIR   , DST_DIR_SZ   );
LOAD_FIELD(&dma_pri_ctrl, DMA_ADDR_NO_MOD, SRC_DIR   , SRC_DIR_SZ   );
SET_BIT(&dma_pri_ctrl,FS);            /* Set Frame Sync bit         */
SET_BIT(&dma_pri_ctrl,TCINT);         /* Allow Ch2 to interrupt CPU */
SET_BIT(&dma_pri_ctrl,EMOD);          /* Halt DMA with emu halt     */

/* Set up DMA Tranfer Count Register */
LOAD_FIELD(&dma_tcnt, FR_COUNT, FRAME_COUNT  , FRAME_COUNT_SZ  );
LOAD_FIELD(&dma_tcnt, EL_COUNT, ELEMENT_COUNT, ELEMENT_COUNT_SZ);

/* Set up Source and Destination Address Registers */
dma_src_addr = (unsigned int)AFEin;
dma_dst_addr = (unsigned int)INBUFFER1;

/* Store DMA Channel 1 registers */
dma_init(DMA_CH1,
      dma_pri_ctrl,
      dma_sec_ctrl,
      dma_src_addr,
            dma_dst_addr,
            dma_tcnt);


      /* Start DMA Ch1 Transfer */
      DMA_START(DMA_CH1);

} /* End run_DMA */
```

Synchronized Data Transfer Example Code

```
/* Set up the DMA Control Registers to perform the data transfers  */
/* from McBSP 0 to internal data memory.                           */
void
run_DMA(void)
{
unsigned int   dma_pri_ctrl   = 0;
unsigned int   dma_sec_ctrl   = 0;
unsigned int   dma_src_addr   = 0;
unsigned int   dma_dst_addr   = 0;
unsigned int   dma_tcnt       = 0;
unsigned int   dma_gcr        = 0;
unsigned int   dma_gcra       = 0;
unsigned int   dma_gcrb       = 0;
unsigned int   dma_gndxa      = 0;
unsigned int   dma_gndxb      = 0;
unsigned int   dma_gaddra     = 0;
unsigned int   dma_gaddrb     = 0;
unsigned int   dma_gaddrc     = 0;
unsigned int   dma_gaddrd     = 0;

      /* Reset DMA Control Registers */
      dma_reset();

      /* Set up Global Configuration Registers for the DMA */
      LOAD_FIELD(&dma_gcra, XFER_SIZE, ELEMENT_COUNT, ELEMENT_COUNT_SZ);
      dma_global_init(dma_gcr,dma_gcra,dma_gcrb,dma_gndxa,dma_gndxb,
                      dma_gaddra,dma_gaddrb,dma_gaddrc,dma_gaddrd);

      /* Set up DMA Ch1 to perform a block transfer of XFER_SIZE elements */
      /*   from McBSP 1 DRR to INBUFFER */
      /* Set up DMA Primary Control Register */
      LOAD_FIELD(&dma_pri_ctrl, DMA_RELOAD_GARB, DST_RELOAD, DST_RELOAD_SZ);
      LOAD_FIELD(&dma_pri_ctrl, DMA_RELOAD_NONE, SRC_RELOAD, SRC_RELOAD_SZ);
      LOAD_FIELD(&dma_pri_ctrl, DMA_NO_EM_HALT , EMOD      , 1            );
      LOAD_FIELD(&dma_pri_ctrl, DMA_CPU_PRI    , PRI       , 1            );
      LOAD_FIELD(&dma_pri_ctrl, SEN_NONE       , WSYNC     , WSYNC_SZ     );
      LOAD_FIELD(&dma_pri_ctrl, SEN_REVT0      , RSYNC     , RSYNC_SZ     );
      LOAD_FIELD(&dma_pri_ctrl, DMA_CNT_RELOADA, CNT_RELOAD, 1            );
      LOAD_FIELD(&dma_pri_ctrl, DMA_SPLIT_DIS  , SPLIT     , SPLIT_SZ     );
      LOAD_FIELD(&dma_pri_ctrl, DMA_ESIZE32    , ESIZE     , ESIZE_SZ     );
      LOAD_FIELD(&dma_pri_ctrl, DMA_ADDR_INC   , DST_DIR   , DST_DIR_SZ   );
      LOAD_FIELD(&dma_pri_ctrl, DMA_ADDR_NO_MOD, SRC_DIR   , SRC_DIR_SZ   );
      SET_BIT(&dma_pri_ctrl,TCINT);         /* Allow Ch2 to interrupt CPU */
      SET_BIT(&dma_pri_ctrl,EMOD);          /* Halt DMA with emu halt     */

      /* Set up DMA Tranfer Count Register */
      LOAD_FIELD(&dma_tcnt, XFER_SIZE, ELEMENT_COUNT, ELEMENT_COUNT_SZ);

      /* Set up Source and Destination Address Registers */
      dma_src_addr = MCBSP_DRR_ADDR(0);
      dma_dst_addr = (unsigned int)INBUFFER;

      /* Store DMA Channel 1 registers */
      dma_init(DMA_CH1,
            dma_pri_ctrl,
            dma_sec_ctrl,
            dma_src_addr,
            dma_dst_addr,
            dma_tcnt);

      /* Start DMA Transfers */
      DMA_START(DMA_CH1);

} /* End run_DMA */
```

Split-Mode Transfer Example Code

```
/* Set up the DMA Control Registers to perform to service McBSP 0.   */
void
run_DMA(void)
{
unsigned int   dma_pri_ctrl   = 0;
unsigned int   dma_sec_ctrl   = 0;
unsigned int   dma_src_addr   = 0;
unsigned int   dma_dst_addr   = 0;
unsigned int   dma_tcnt       = 0;
unsigned int   dma_gcr        = 0;
unsigned int   dma_gcra       = 0;
unsigned int   dma_gcrb       = 0;
unsigned int   dma_gndxa      = 0;
unsigned int   dma_gndxb      = 0;
unsigned int   dma_gaddra     = 0;
unsigned int   dma_gaddrb     = 0;
unsigned int   dma_gaddrc     = 0;
unsigned int   dma_gaddrd     = 0;

      /* Reset DMA Control Registers */
      dma_reset();

      /* Set up Global Configuration Registers for the DMA */
      LOAD_FIELD(&dma_gcra, XFER_SIZE, ELEMENT_COUNT, ELEMENT_COUNT_SZ);
      dma_gaddra = MCBSP_DRR_ADDR(0);
      dma_global_init(dma_gcr,dma_gcra,dma_gcrb,dma_gndxa,dma_gndxb,
                      dma_gaddra,dma_gaddrb,dma_gaddrc,dma_gaddrd);

      /* Set up DMA Ch1 to perform a block transfer of XFER_SIZE elements */
      /*   from McBSP 0 DRR to INBUFFER */
      /* Set up DMA Primary Control Register */
      LOAD_FIELD(&dma_pri_ctrl, DMA_RELOAD_NONE, DST_RELOAD, DST_RELOAD_SZ);
      LOAD_FIELD(&dma_pri_ctrl, DMA_RELOAD_NONE, SRC_RELOAD, SRC_RELOAD_SZ);
      LOAD_FIELD(&dma_pri_ctrl, DMA_NO_EM_HALT , EMOD      , 1            );
      LOAD_FIELD(&dma_pri_ctrl, DMA_CPU_PRI    , PRI       , 1            );
      LOAD_FIELD(&dma_pri_ctrl, SEN_XEVT0      , WSYNC     , WSYNC_SZ     );
      LOAD_FIELD(&dma_pri_ctrl, SEN_REVT0      , RSYNC     , RSYNC_SZ     );
      LOAD_FIELD(&dma_pri_ctrl, DMA_CNT_RELOADA, CNT_RELOAD, 1             );
      LOAD_FIELD(&dma_pri_ctrl, DMA_SPLIT_GARA , SPLIT     , SPLIT_SZ     );
      LOAD_FIELD(&dma_pri_ctrl, DMA_ESIZE32    , ESIZE     , ESIZE_SZ     );
      LOAD_FIELD(&dma_pri_ctrl, DMA_ADDR_INC   , DST_DIR   , DST_DIR_SZ   );
      LOAD_FIELD(&dma_pri_ctrl, DMA_ADDR_INC   , SRC_DIR   , SRC_DIR_SZ   );
      SET_BIT(&dma_pri_ctrl,TCINT);         /* Allow Ch1 to interrupt CPU */

      /* Set up DMA Tranfer Count Register */
      LOAD_FIELD(&dma_tcnt, XFER_SIZE, ELEMENT_COUNT, ELEMENT_COUNT_SZ);

      /* Set up Source and Destination Address Registers */
      dma_src_addr = (unsigned int)OUTBUFFER;
      dma_dst_addr = (unsigned int)INBUFFER;

      /* Store DMA Channel 1 registers */
      dma_init(DMA_CH1,
            dma_pri_ctrl,
            dma_sec_ctrl,
            dma_src_addr,
            dma_dst_addr,
            dma_tcnt);

      /* Start DMA Transfer */
      DMA_START(DMA_CH1);

} /* End run_DMA */
```

Frame-Synchronized Data Transfer Example Code

```
/* Set up the DMA Control Registers to perform the data transfers  */
/* between internal data memory and the external AFE.              */
void
run_DMA(void)
{
unsigned int   dma_pri_ctrl   = 0;
unsigned int   dma_sec_ctrl   = 0;
unsigned int   dma_src_addr   = 0;
unsigned int   dma_dst_addr   = 0;
unsigned int   dma_tcnt       = 0;
unsigned int   dma_gcr        = 0;
unsigned int   dma_gcra       = 0;
unsigned int   dma_gcrb       = 0;
unsigned int   dma_gndxa      = 0;
unsigned int   dma_gndxb      = 0;
unsigned int   dma_gaddra     = 0;
unsigned int   dma_gaddrb     = 0;
unsigned int   dma_gaddrc     = 0;
unsigned int   dma_gaddrd     = 0;
unsigned int   count_reload   = 0;

      /* Reset DMA Control Registers */
      dma_reset();

      /* Set up Global Configuration Registers for the DMA */
      dma_gcra = (unsigned int)XFER_SIZE;
      dma_gaddrb = (unsigned int)INBUFFER;
      dma_gaddrc = (unsigned int)OUTBUFFER;
      dma_global_init(dma_gcr,dma_gcra,dma_gcrb,dma_gndxa,dma_gndxb,
                      dma_gaddra,dma_gaddrb,dma_gaddrc,dma_gaddrd);

      /* Set up DMA Ch1 to perform a block transfer of XFER_SIZE elements */
      /*   from AFE to INBUFFER */
      /* Set up DMA Primary Control Register */
      LOAD_FIELD(&dma_pri_ctrl, DMA_RELOAD_GARB, DST_RELOAD, DST_RELOAD_SZ);
      LOAD_FIELD(&dma_pri_ctrl, DMA_RELOAD_NONE, SRC_RELOAD, SRC_RELOAD_SZ);
      LOAD_FIELD(&dma_pri_ctrl, DMA_NO_EM_HALT , EMOD      , 1            );
      LOAD_FIELD(&dma_pri_ctrl, DMA_CPU_PRI    , PRI       , 1            );
      LOAD_FIELD(&dma_pri_ctrl, SEN_NONE       , WSYNC     , WSYNC_SZ     );
      LOAD_FIELD(&dma_pri_ctrl, SEN_EXT_INT4   , RSYNC     , RSYNC_SZ     );
      LOAD_FIELD(&dma_pri_ctrl, DMA_CNT_RELOADA, CNT_RELOAD, 1            );
      LOAD_FIELD(&dma_pri_ctrl, DMA_SPLIT_DIS  , SPLIT     , SPLIT_SZ     );
      LOAD_FIELD(&dma_pri_ctrl, DMA_ESIZE32    , ESIZE     , ESIZE_SZ     );
      LOAD_FIELD(&dma_pri_ctrl, DMA_ADDR_INC   , DST_DIR   , DST_DIR_SZ   );
      LOAD_FIELD(&dma_pri_ctrl, DMA_ADDR_NO_MOD, SRC_DIR   , SRC_DIR_SZ   );
      SET_BIT(&dma_pri_ctrl,FS);            /* Set Frame Sync bit         */
      SET_BIT(&dma_pri_ctrl,TCINT);         /* Allow Ch2 to interrupt CPU */
      SET_BIT(&dma_pri_ctrl,EMOD);          /* Halt DMA with emu halt     */

      /* Set up DMA Secondary Control Register */
      SET_BIT(&dma_sec_ctrl,FRAME_IE); /* FRAME COND to generate interrupt*/

      /* Set up DMA Tranfer Count Register */
      LOAD_FIELD(&dma_tcnt, 1        , FRAME_COUNT  , FRAME_COUNT_SZ  );
      LOAD_FIELD(&dma_tcnt, XFER_SIZE, ELEMENT_COUNT, ELEMENT_COUNT_SZ);

      /* Set up Source and Destination Address Registers */
      dma_src_addr = (unsigned int)AFEin;
      dma_dst_addr = (unsigned int)INBUFFER;

      /* Store DMA Channel 1 registers */
      dma_init(DMA_CH1,
            dma_pri_ctrl,
            dma_sec_ctrl,
            dma_src_addr,
            dma_dst_addr,
            dma_tcnt);

      /* Start DMA Ch1 Transfer */
      DMA_START(DMA_CH1);

      /* Set up DMA Ch2 to perform a block transfer of XFER_SIZE elements */
      /*   from OUTBUFFER to AFE */
      /* Set up DMA Primary Control Register */
      LOAD_FIELD(&dma_pri_ctrl, DMA_RELOAD_NONE, DST_RELOAD, DST_RELOAD_SZ);
      LOAD_FIELD(&dma_pri_ctrl, DMA_RELOAD_GARC, SRC_RELOAD, SRC_RELOAD_SZ);
      LOAD_FIELD(&dma_pri_ctrl, DMA_NO_EM_HALT , EMOD      , 1            );
```

```
      LOAD_FIELD(&dma_pri_ctrl, DMA_CPU_PRI    , PRI       , 1            );
      LOAD_FIELD(&dma_pri_ctrl, SEN_NONE       , WSYNC     , WSYNC_SZ     );
      LOAD_FIELD(&dma_pri_ctrl, SEN_DMA_INT1   , RSYNC     , RSYNC_SZ     );
      LOAD_FIELD(&dma_pri_ctrl, DMA_CNT_RELOADA, CNT_RELOAD, 1            );
      LOAD_FIELD(&dma_pri_ctrl, DMA_SPLIT_DIS  , SPLIT     , SPLIT_SZ     );
      LOAD_FIELD(&dma_pri_ctrl, DMA_ESIZE32    , ESIZE     , ESIZE_SZ     );
      LOAD_FIELD(&dma_pri_ctrl, DMA_ADDR_NO_MOD, DST_DIR   , DST_DIR_SZ   );
      LOAD_FIELD(&dma_pri_ctrl, DMA_ADDR_INC   , SRC_DIR   , SRC_DIR_SZ   );
      SET_BIT(&dma_pri_ctrl,FS);            /* Set Frame Sync bit         */
      SET_BIT(&dma_pri_ctrl,EMOD);          /* Halt DMA with emu halt     */

      /* Set up DMA Tranfer Count Register */
      LOAD_FIELD(&dma_tcnt, 1        , FRAME_COUNT  , FRAME_COUNT_SZ  );
      LOAD_FIELD(&dma_tcnt, XFER_SIZE, ELEMENT_COUNT, ELEMENT_COUNT_SZ);

      /* Set up Source and Destination Address Registers */
      dma_src_addr = (unsigned int)OUTBUFFER;
      dma_dst_addr = (unsigned int)AFEout;

      /* Store DMA Channel 2 registers */
      dma_init(DMA_CH2,
            dma_pri_ctrl,
            dma_sec_ctrl,
            dma_src_addr,
            dma_dst_addr,
            dma_tcnt);

      /* Start DMA Ch2 Transfer */
      DMA_START(DMA_CH2);

} /* End run_DMA */
```

Circular Buffering Transfer Example Code

```
/* Set up the DMA Control Registers to perform the data transfers  */
/* between internal data memory and the external AFE, using cir-   */
/* ular buffering for both the input and output data.             */
void
run_DMA(void)
{
unsigned int   dma_pri_ctrl   = 0;
unsigned int   dma_sec_ctrl   = 0;
unsigned int   dma_src_addr   = 0;
unsigned int   dma_dst_addr   = 0;
unsigned int   dma_tcnt       = 0;
unsigned int   dma_gcr        = 0;
unsigned int   dma_gcra       = 0;
unsigned int   dma_gcrb       = 0;
unsigned int   dma_gndxa      = 0;
unsigned int   dma_gndxb      = 0;
unsigned int   dma_gaddra     = 0;
unsigned int   dma_gaddrb     = 0;
unsigned int   dma_gaddrc     = 0;
unsigned int   dma_gaddrd     = 0;
unsigned int   count_reload   = 0;
unsigned int   el_index       = 0;
unsigned int   fr_index       = 0;
unsigned int   esize          = 0;

      /* Reset DMA Control Registers */
      dma_reset();

      /* Calculate index values, as well as the ESIZE, based on the  */
      /* number of elements per frame (EL_COUNT) and the number of   */
      /* bytes of each element (EL_SIZE).                            */
      el_index = EL_SIZE;
      fr_index = -((EL_COUNT - 1) * EL_SIZE);
      if (EL_SIZE == 1) esize = 2;
      else if (EL_SIZE == 2) esize = 1;
      else esize = 0;
```

```
/* Set up Global Configuration Registers for the DMA */
dma_gcra = (unsigned int)XFER_SIZE;
LOAD_FIELD(&dma_gcra , 1       , FRAME_COUNT  , FRAME_COUNT_SZ  );
LOAD_FIELD(&dma_gcra , EL_COUNT, ELEMENT_COUNT, ELEMENT_COUNT_SZ);
LOAD_FIELD(&dma_gndxa, fr_index, FRAME_INDEX  , FRAME_COUNT_SZ  );
LOAD_FIELD(&dma_gndxa, el_index, ELEMENT_INDEX, ELEMENT_COUNT_SZ);
dma_global_init(dma_gcr,dma_gcra,dma_gcrb,dma_gndxa,dma_gndxb,
                dma_gaddra,dma_gaddrb,dma_gaddrc,dma_gaddrd);

/* Set up DMA Ch1 to perform a block transfer of XFER_SIZE elements */
/*   from AFE to INBUFFER, continuously. */
/* Set up DMA Primary Control Register */
LOAD_FIELD(&dma_pri_ctrl, DMA_RELOAD_NONE, DST_RELOAD, DST_RELOAD_SZ);
LOAD_FIELD(&dma_pri_ctrl, DMA_RELOAD_NONE, SRC_RELOAD, SRC_RELOAD_SZ);
LOAD_FIELD(&dma_pri_ctrl, DMA_NO_EM_HALT , EMOD      , 1            );
LOAD_FIELD(&dma_pri_ctrl, DMA_CPU_PRI    , PRI       , 1            );
LOAD_FIELD(&dma_pri_ctrl, SEN_NONE       , WSYNC     , WSYNC_SZ     );
LOAD_FIELD(&dma_pri_ctrl, SEN_EXT_INT4   , RSYNC     , RSYNC_SZ     );
LOAD_FIELD(&dma_pri_ctrl, DMA_INDXA      , INDEX     , 1            );
LOAD_FIELD(&dma_pri_ctrl, DMA_CNT_RELOADA, CNT_RELOAD, 1            );
LOAD_FIELD(&dma_pri_ctrl, DMA_SPLIT_DIS  , SPLIT     , SPLIT_SZ     );
LOAD_FIELD(&dma_pri_ctrl, esize          , ESIZE     , ESIZE_SZ     );
LOAD_FIELD(&dma_pri_ctrl, DMA_ADDR_INDX  , DST_DIR   , DST_DIR_SZ   );
LOAD_FIELD(&dma_pri_ctrl, DMA_ADDR_NO_MOD, SRC_DIR   , SRC_DIR_SZ   );
SET_BIT(&dma_pri_ctrl,FS);            /* Set Frame Sync bit         */
SET_BIT(&dma_pri_ctrl,TCINT);         /* Allow Ch2 to interrupt CPU */

/* Set up DMA Secondary Control Register */
SET_BIT(&dma_sec_ctrl,FRAME_IE); /* FRAME COND to generate interrupt*/

/* Set up DMA Tranfer Count Register */
LOAD_FIELD(&dma_tcnt, 1       , FRAME_COUNT  , FRAME_COUNT_SZ  );
LOAD_FIELD(&dma_tcnt, EL_COUNT, ELEMENT_COUNT, ELEMENT_COUNT_SZ);

/* Set up Source and Destination Address Registers */
dma_src_addr = (unsigned int)AFEin;
dma_dst_addr = (unsigned int)INBUFFER;

/* Store DMA Channel 1 registers */
dma_init(DMA_CH1,
      dma_pri_ctrl,
      dma_sec_ctrl,
      dma_src_addr,
      dma_dst_addr,
      dma_tcnt);

/* Start DMA Ch1 Transfer */
DMA_AUTO_START(DMA_CH1);

/* Set up DMA Ch2 to perform a block transfer of XFER_SIZE elements */
/*   from OUTBUFFER to AFE, continuously. */
/* Set up DMA Primary Control Register */
LOAD_FIELD(&dma_pri_ctrl, DMA_RELOAD_NONE, DST_RELOAD, DST_RELOAD_SZ);
LOAD_FIELD(&dma_pri_ctrl, DMA_RELOAD_NONE, SRC_RELOAD, SRC_RELOAD_SZ);
LOAD_FIELD(&dma_pri_ctrl, DMA_NO_EM_HALT , EMOD      , 1            );
LOAD_FIELD(&dma_pri_ctrl, DMA_CPU_PRI    , PRI       , 1            );
LOAD_FIELD(&dma_pri_ctrl, SEN_NONE       , WSYNC     , WSYNC_SZ     );
LOAD_FIELD(&dma_pri_ctrl, SEN_DMA_INT1   , RSYNC     , RSYNC_SZ     );
LOAD_FIELD(&dma_pri_ctrl, DMA_INDXA      , INDEX     , 1            );
LOAD_FIELD(&dma_pri_ctrl, DMA_CNT_RELOADA, CNT_RELOAD, 1            );
LOAD_FIELD(&dma_pri_ctrl, DMA_SPLIT_DIS  , SPLIT     , SPLIT_SZ     );
LOAD_FIELD(&dma_pri_ctrl, esize          , ESIZE     , ESIZE_SZ     );
LOAD_FIELD(&dma_pri_ctrl, DMA_ADDR_NO_MOD, DST_DIR   , DST_DIR_SZ   );
LOAD_FIELD(&dma_pri_ctrl, DMA_ADDR_INDX  , SRC_DIR   , SRC_DIR_SZ   );
SET_BIT(&dma_pri_ctrl,FS);            /* Set Frame Sync bit         */

/* Set up DMA Tranfer Count Register */
LOAD_FIELD(&dma_tcnt, 1       , FRAME_COUNT  , FRAME_COUNT_SZ  );
LOAD_FIELD(&dma_tcnt, EL_COUNT, ELEMENT_COUNT, ELEMENT_COUNT_SZ);

/* Set up Source and Destination Address Registers */
dma_src_addr = (unsigned int)OUTBUFFER;
dma_dst_addr = (unsigned int)AFEout;

/* Store DMA Channel 2 registers */
dma_init(DMA_CH2,
      dma_pri_ctrl,
      dma_sec_ctrl,
      dma_src_addr,
      dma_dst_addr,
            dma_tcnt);

      /* Start DMA Ch2 Transfer */
      DMA_AUTO_START(DMA_CH2);

} /* End run_DMA */
```

Ping-Pong Transfer Example Code

```
/* Set up the DMA Control Registers to perform the data transfers  */
/* between internal data memory and the external AFE, using a ping-*/
/* pong buffering scheme for both input and output data.           */
void
run_DMA(void)
{
unsigned int   dma_pri_ctrl   = 0;
unsigned int   dma_sec_ctrl   = 0;
unsigned int   dma_src_addr   = 0;
unsigned int   dma_dst_addr   = 0;
unsigned int   dma_tcnt       = 0;
unsigned int   dma_gcr        = 0;
unsigned int   dma_gcra       = 0;
unsigned int   dma_gcrb       = 0;
unsigned int   dma_gndxa      = 0;
unsigned int   dma_gndxb      = 0;
unsigned int   dma_gaddra     = 0;
unsigned int   dma_gaddrb     = 0;
unsigned int   dma_gaddrc     = 0;
unsigned int   dma_gaddrd     = 0;
unsigned int   count_reload   = 0;

      /* Reset DMA Control Registers */
      dma_reset();

      /* Set up Global Configuration Registers for the DMA */
      LOAD_FIELD(&dma_gcra, 2        , FRAME_COUNT  , FRAME_COUNT_SZ  );
      LOAD_FIELD(&dma_gcra, XFER_SIZE, ELEMENT_COUNT, ELEMENT_COUNT_SZ);
      dma_gaddrb = (unsigned int)INBUFFER1;
      dma_gaddrc = (unsigned int)OUTBUFFER1;
      dma_global_init(dma_gcr,dma_gcra,dma_gcrb,dma_gndxa,dma_gndxb,
                      dma_gaddra,dma_gaddrb,dma_gaddrc,dma_gaddrd);

      /* Set up DMA Ch1 to perform a block transfer of XFER_SIZE elements */
      /*   from AFE to INBUFFER */
      /* Set up DMA Primary Control Register */
      LOAD_FIELD(&dma_pri_ctrl, DMA_RELOAD_GARB, DST_RELOAD, DST_RELOAD_SZ);
      LOAD_FIELD(&dma_pri_ctrl, DMA_RELOAD_NONE, SRC_RELOAD, SRC_RELOAD_SZ);
      LOAD_FIELD(&dma_pri_ctrl, DMA_NO_EM_HALT , EMOD      , 1            );
      LOAD_FIELD(&dma_pri_ctrl, DMA_CPU_PRI    , PRI       , 1            );
      LOAD_FIELD(&dma_pri_ctrl, SEN_NONE       , WSYNC     , WSYNC_SZ     );
      LOAD_FIELD(&dma_pri_ctrl, SEN_EXT_INT4   , RSYNC     , RSYNC_SZ     );
      LOAD_FIELD(&dma_pri_ctrl, DMA_CNT_RELOADA, CNT_RELOAD, 1            );
      LOAD_FIELD(&dma_pri_ctrl, DMA_SPLIT_DIS  , SPLIT     , SPLIT_SZ     );
      LOAD_FIELD(&dma_pri_ctrl, DMA_ESIZE32    , ESIZE     , ESIZE_SZ     );
      LOAD_FIELD(&dma_pri_ctrl, DMA_ADDR_INC   , DST_DIR   , DST_DIR_SZ   );
      LOAD_FIELD(&dma_pri_ctrl, DMA_ADDR_NO_MOD, SRC_DIR   , SRC_DIR_SZ   );
      SET_BIT(&dma_pri_ctrl,FS);            /* Set Frame Sync bit         */
      SET_BIT(&dma_pri_ctrl,TCINT);         /* Allow Ch2 to interrupt CPU */

      /* Set up DMA Secondary Control Register */
      SET_BIT(&dma_sec_ctrl,FRAME_IE); /* FRAME COND to generate interrupt*/

      /* Set up DMA Tranfer Count Register */
      LOAD_FIELD(&dma_tcnt, 2       , FRAME_COUNT  , FRAME_COUNT_SZ  );
      LOAD_FIELD(&dma_tcnt, EL_COUNT, ELEMENT_COUNT, ELEMENT_COUNT_SZ);

      /* Set up Source and Destination Address Registers */
      dma_src_addr = (unsigned int)AFEin;
      dma_dst_addr = (unsigned int)INBUFFER1;

      /* Store DMA Channel 1 registers */
      dma_init(DMA_CH1,
            dma_pri_ctrl,
            dma_sec_ctrl,
            dma_src_addr,
            dma_dst_addr,
            dma_tcnt);

      /* Start DMA Ch1 Transfer */
      DMA_AUTO_START(DMA_CH1);

      /* Set up DMA Ch2 to perform a block transfer of XFER_SIZE elements */
      /*   from OUTBUFFER to AFE */
      /* Set up DMA Primary Control Register */
      LOAD_FIELD(&dma_pri_ctrl, DMA_RELOAD_NONE, DST_RELOAD, DST_RELOAD_SZ);
      LOAD_FIELD(&dma_pri_ctrl, DMA_RELOAD_GARC, SRC_RELOAD, SRC_RELOAD_SZ);
```

```
      LOAD_FIELD(&dma_pri_ctrl, DMA_NO_EM_HALT , EMOD      , 1            );
      LOAD_FIELD(&dma_pri_ctrl, DMA_CPU_PRI    , PRI       , 1            );
      LOAD_FIELD(&dma_pri_ctrl, SEN_NONE       , WSYNC     , WSYNC_SZ     );
      LOAD_FIELD(&dma_pri_ctrl, SEN_DMA_INT1   , RSYNC     , RSYNC_SZ     );
      LOAD_FIELD(&dma_pri_ctrl, DMA_CNT_RELOADA, CNT_RELOAD, 1            );
      LOAD_FIELD(&dma_pri_ctrl, DMA_SPLIT_DIS  , SPLIT     , SPLIT_SZ     );
      LOAD_FIELD(&dma_pri_ctrl, DMA_ESIZE32    , ESIZE     , ESIZE_SZ     );
      LOAD_FIELD(&dma_pri_ctrl, DMA_ADDR_NO_MOD, DST_DIR   , DST_DIR_SZ   );
      LOAD_FIELD(&dma_pri_ctrl, DMA_ADDR_INC   , SRC_DIR   , SRC_DIR_SZ   );
      SET_BIT(&dma_pri_ctrl,FS);            /* Set Frame Sync bit         */

      /* Set up DMA Tranfer Count Register */
      LOAD_FIELD(&dma_tcnt, 2        , FRAME_COUNT  , FRAME_COUNT_SZ  );
      LOAD_FIELD(&dma_tcnt, XFER_SIZE, ELEMENT_COUNT, ELEMENT_COUNT_SZ);

      /* Set up Source and Destination Address Registers */
      dma_src_addr = (unsigned int)OUTBUFFER1;
      dma_dst_addr = (unsigned int)AFEout;

      /* Store DMA Channel 2 registers */
      dma_init(DMA_CH2,
            dma_pri_ctrl,
            dma_sec_ctrl,
            dma_src_addr,
            dma_dst_addr,
            dma_tcnt);

      /* Start DMA Ch2 Transfer in DMA_Ch1_ISR after Output Buffer 1 has  */
      /* valid data in it (i.e. after first 2 input frames are received). */

} /* End run_DMA */
```

Program Paging Transfer Example Code

```
/* Set up the DMA Control Registers to perform the data transfers  */
/* from external memory to internal program memory. There are four */
/* program pages: EXTPAGE1, EXTPAGE2, EXTPAGE3, EXTPAGE4. These are*/
/* brought into program memory locations PAGE1 and PAGE2 to be     */
/* executed. Note that an interrupt service routine must be set up */
/* to adjust dma_gaddrb and dma_gaddrc to point to the next ex-    */
/* ternal and internal page locations.                             */
run_DMA(void)
{
unsigned int   dma_pri_ctrl   = 0;
unsigned int   dma_sec_ctrl   = 0;
unsigned int   dma_src_addr   = 0;
unsigned int   dma_dst_addr   = 0;
unsigned int   dma_tcnt       = 0;
unsigned int   dma_gcr        = 0;
unsigned int   dma_gcra       = 0;
unsigned int   dma_gcrb       = 0;
unsigned int   dma_gndxa      = 0;
unsigned int   dma_gndxb      = 0;
unsigned int   dma_gaddra     = 0;
unsigned int   dma_gaddrb     = 0;
unsigned int   dma_gaddrc     = 0;
unsigned int   dma_gaddrd     = 0;

      /* Reset DMA Control Registers */
      dma_reset();

      /* Set up Global Configuration Registers for the DMA */
      LOAD_FIELD(&dma_gcra, 1        , FRAME_COUNT  , FRAME_COUNT_SZ  );
      LOAD_FIELD(&dma_gcra, PAGE_SIZE, ELEMENT_COUNT, ELEMENT_COUNT_SZ);
      dma_gaddrb = (unsigned int)EXTPAGE2;
      dma_gaddrc = (unsigned int)INPAGE2;
      dma_global_init(dma_gcr,dma_gcra,dma_gcrb,dma_gndxa,dma_gndxb,
                      dma_gaddra,dma_gaddrb,dma_gaddrc,dma_gaddrd);

      /* Set up DMA Ch3 to perform an auto-initialized block transfer of  */
      /* PAGE_SIZE elements from external to internal program memory      */
```

```
    /* Set up DMA Primary Control Register */
    LOAD_FIELD(&dma_pri_ctrl, DMA_RELOAD_GARC, DST_RELOAD, DST_RELOAD_SZ);
    LOAD_FIELD(&dma_pri_ctrl, DMA_RELOAD_GARB, SRC_RELOAD, SRC_RELOAD_SZ);
    LOAD_FIELD(&dma_pri_ctrl, DMA_EM_HALT    , EMOD      , 1            );
    LOAD_FIELD(&dma_pri_ctrl, DMA_CPU_PRI    , PRI       , 1            );
    LOAD_FIELD(&dma_pri_ctrl, SEN_NONE       , WSYNC     , WSYNC_SZ     );
    LOAD_FIELD(&dma_pri_ctrl, SEN_EXT_INT6   , RSYNC     , RSYNC_SZ     );
    LOAD_FIELD(&dma_pri_ctrl, DMA_CNT_RELOADA, CNT_RELOAD, 1            );
    LOAD_FIELD(&dma_pri_ctrl, DMA_SPLIT_DIS  , SPLIT     , SPLIT_SZ     );
    LOAD_FIELD(&dma_pri_ctrl, DMA_ESIZE32    , ESIZE     , ESIZE_SZ     );
    LOAD_FIELD(&dma_pri_ctrl, DMA_ADDR_INC   , DST_DIR   , DST_DIR_SZ   );
    LOAD_FIELD(&dma_pri_ctrl, DMA_ADDR_INC   , SRC_DIR   , SRC_DIR_SZ   );
    SET_BIT(&dma_pri_ctrl,FS);            /* Set Frame Sync bit         */
    SET_BIT(&dma_pri_ctrl,TCINT);         /* Allow Ch3 to interrupt CPU */

    /* Set up DMA Secondary Control Register */
    LOAD_FIELD(&dma_sec_ctrl, DMAC_BLOCK_COND, DMAC_EN, DMAC_EN_SZ);
    SET_BIT(&dma_sec_ctrl,BLOCK_IE); /* FRAME COND to generate interrupt*/

    /* Set up DMA Tranfer Count Register */
    LOAD_FIELD(&dma_tcnt, 1        , FRAME_COUNT  , FRAME_COUNT_SZ  );
    LOAD_FIELD(&dma_tcnt, PAGE_SIZE, ELEMENT_COUNT, ELEMENT_COUNT_SZ);

    /* Set up Source and Destination Address Registers */
    dma_src_addr = (unsigned int)EXTPAGE1;
    dma_dst_addr = (unsigned int)INPAGE1;

    /* Store DMA Channel 3 registers */
    dma_init(DMA_CH3,
          dma_pri_ctrl,
          dma_sec_ctrl,
          dma_src_addr,
          dma_dst_addr,
          dma_tcnt);

    /* Start DMA Ch3 Transfer */
    DMA_AUTO_START(DMA_CH3);
    DMA_RSYNC_SET(DMA_CH3);
} /* End run_DMA */
```