MOTOROLA

APR23/D

Using the DSP56300

Direct Memory Access Controller

Motorola's High-Performance DSP Technology dsp

Using the DSP56300 Direct Memory Access Controller

by
Eliezer Sand
Motorola Semiconductor Israel Ltd.

Motorola, Incorporated
Semiconductor Products Sector
6501 William Cannon Drive West
Austin, TX 78735-8598

OnCE and Mfax are trademarks of Motorola, Inc.

Order this document by APR23/D.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

SECTION 1

INTRODUCTION

The Direct Memory Access (DMA) controller is the part of the DSP56300 core that permits data transfers between internal or external memory and/or internal or external I/O in any combination, without intervention of the core. Due to dedicated DMA address and data buses, as well as internal memory partitioning, a high level of isolation is achieved so that the DMA operation does not interfere with or slow down the core operation.

1.1 OVERVIEW

Because the DMA controller is part of the DSP56300 core, the Bus Interface Unit (BIU) receives all the needed controls from the DMA controller and can manage external activity with maximum flexibility and performance, depending on the profile of the tasks to be handled. The DMA controller has six channels, each with its own register set. All the registers are memory-mapped in the internal I/O memory space. **Table 1-1** shows the various types of data transfers the DMA controller can perform.

Table 1-1 DMA Controller Data Transfers

Location to/from Location			Minimum Clock Cycles per Single Word Transfer
Internal Memory	→	Internal Memory	2
External Memory	↔	Internal Memory	2 + wait states
External Memory	→	External Memory	2 + wait states
Internal Memory	↔	Internal I/O	2
External Memory	↔	Internal I/O	2 + wait states
Internal I/O	→	Internal I/O	2

Data transfer for one channel takes a minimum of 2 clock cycles per single word. The number of clocks per transfer can be larger if there is a contention between the core and the DMA activity (i.e., if they both access the same 1/4 K of internal RAM in the same cycle (or three cycles in Packing mode—see **Section 5.4** on page 5-5), or, if they both want to access external memory).

1.2 DMA REGISTERS

The DMA has six identical channels. Each channel has four dedicated registers:

- **DSR*i*** —DMA Source Register for channel *i* which holds the source base address for the next DMA transfer
- **DDR*i*** —DMA Destination Register for channel *i* which holds the destination base address for the next DMA transfer
- **DCO*i*** —DMA Counter for channel *i* which contains the number of DMA transfers left to perform
- **DCR*i*** —DMA Control Register for channel *i* which contains all the bits needed to control the operation of the channel

In addition to these channel-dedicated registers, there are also five common registers. The common registers include four DMA Offset Registers (DOR0, DOR1, DOR2, and DOR3) and one read-only DMA Status Register (DSTR). The DORs hold offset addresses to be used by any of the channels, as required, in specific addressing modes.

- **Table 1-2** describes the function of DMA Control Register (DCR) bits.
- **Table 1-3** on page 1-8 describes the different modes of the DMA Counter (DCO) which are selected by the addressing mode defined in the DCR.
- **Table 1-4** on page 1-9 describes the structure of the read only DSTR.

Table 1-2 DMA Control Register Structure

Bit No.	Bit Names	Value	Function
23	DE	0	Clearing the bit disables DMA transfers on this channel.
		1	Setting the bit initiates a transfer, or enables the initiation of a transfer by another source (e.g., by external interrupt).
22	DIE	0	Clearing this bit disables the interrupt-at-end-of-transfer function for this channel.
		1	Setting the bit enables the interrupt-at-end-of-transfer function for this channel.
21–19	DTM[2:0]	—	These bits define: • **Transfer mode:** Block, line or word per request • **Trigger source:** DMA request or DE with / without DE auto clear at end-of-transfer
		000	This value selects a block transfer by request with DE auto clear.
		001	This value selects a word transfer by request with DE auto clear.
		010	This value selects a line transfer by request with DE auto clear.
		011	This value selects a block transfer by DE with DE auto clear.
		100	This value selects a block transfer by request without DE auto clear.
		101	This value selects a word transfer by request without DE auto clear.
		110	This value is not defined and is reserved.
		111	This value is not defined and is reserved.
18–17	DPR[1:0]	—	These bits select the channel priority.
		00	This value selects Priority Level 0 (lowest).
		01	This value selects Priority Level 1.
		10	This value selects Priority Level 2.
		11	This value selects Priority Level 3 (highest).

Table 1-2 DMA Control Register Structure (Continued)

Bit No.	Bit Names	Value	Function
16	DCON	0	Clearing this bit disables Continuous mode.
		1	Setting this bit enables Continuous mode for this channel.
15–11	DRS[4:0]	—	These bits identify the DMA request source.
		00000	External ($\overline{\text{IRQA}}$ pin)
		00001	External ($\overline{\text{IRQB}}$ pin)
		00010	External ($\overline{\text{IRQC}}$ pin)
		00011	External ($\overline{\text{IRQD}}$ pin)
		00100	Transfer Done from channel 0
		00101	Transfer Done from channel 1
		00110	Transfer Done from channel 2
		00111	Transfer Done from channel 3
		01000	Transfer Done from channel 4
		01001	Transfer Done from channel 5
		01010–11111	Peripheral Request MDRQ0–Peripheral Request MDRQ21
10	D3D	0	Clearing this bit disables 3D mode.
		1	Setting this bit enables 3D mode.

Table 1-2 DMA Control Register Structure (Continued)

Bit No.	Bit Names	Value	Function		
9–7	DAM[5:3]	**If D3D = 0**			
		—	**Destination Addressing Mode**	**Counter Mode**	**Offset Select**
		000	Two-dimensional	B	DOR0
		001	Two-dimensional	B	DOR1
		010	Two-dimensional	B	DOR2
		011	Two-dimensional	B	DOR3
		100	No Update	A	None
		101	Postincrement-by-1	A	None
		110	Reserved		
		111	Reserved		
		Note: If the destination address generation mode specifies a different counter mode than the source address generation mode, then the counter mode is B.			
		If D3D = 1			
		—	**Destination Addressing Mode**	**Offset Select**	
		000	Two-dimensional	DOR0	
		001	Two-dimensional	DOR1	
		010	Two-dimensional	DOR2	
		011	Two-dimensional	DOR3	
		100	No Update	None	
		101	Postincrement-by-1	None	
		110	Three-dimensional	DOR0: DOR1	
		111	Three-dimensional	DOR2: DOR3	

Table 1-2 DMA Control Register Structure (Continued)

Bit No.	Bit Names	Value	Function		
6–4	DAM[2:0]	**If D3D = 0**			
		—	**Source Addressing Mode**	**Counter Mode**	**Offset Select**
		000	Two-dimensional	B	DOR0
		001	Two-dimensional	B	DOR1
		010	Two-dimensional	B	DOR2
		011	Two-dimensional	B	DOR3
		100	No Update	A	None
		101	Postincrement-by-1	A	None
		110	Reserved		
		111	Reserved		
		Note: If the source address generation mode specifies a different counter mode than the destination address generation mode, then the counter mode is B.			
		If D3D = 1			
		—	**Addressing Mode**	**Counter Mode**	**Offset Select**
		000	Source: 3D	C	Source: DOR0:DOR1
			Dest: See DAM[5:3]		Dest: See DAM[5:3]
		001	Source: 3D	D	Source: DOR0:DOR1
			Dest: See DAM[5:3]		Dest: See DAM[5:3]
		010	Source: 3D	E	Source: DOR0:DOR1
			Dest: See DAM[5:3]		Dest: See DAM[5:3]
		011	Source: 3D	Reserved	Source: DOR0:DOR1
			Dest: See DAM[5:3]		Dest: See DAM[5:3]
		100	Source: See DAM[5:3]	C	Source: See DAM[5:3]
			Destination: 3D		Dest: DOR2:DOR3
		101	Source: See DAM[5:3]	D	Source: See DAM[5:3]
			Destination: 3D		Dest: DOR2:DOR3
		110	Source: See DAM[5:3]	E	Source: See DAM[5:3]
			Destination: 3D		Dest: DOR2:DOR3
		111	Source: See DAM[5:3]	Reserved	Source: See DAM[5:3]
			Destination: 3D		Dest: DOR2:DOR3

Table 1-2 DMA Control Register Structure (Continued)

Bit No.	Bit Names	Value	Function
3–2	DDS[1:0]	—	These bits select the destination memory (X data, Y data, or program).
		00	X Memory Space
		01	Y Memory Space
		10	P Memory Space
		11	Reserved
		Note:	In Cache mode, a DMA-to-program memory space has some limitations (as described in the *DSP56300 Family Manual*).
1–0	DSS{1:0]	—	These bits select the source memory (X data, Y data, or program).
		00	X Memory Space
		01	Y Memory Space
		10	P Memory Space
		11	Reserved
		Note:	In Cache mode, a DMA-from-program memory space has some limitations (as described in the *DSP56300 Family Manual*).

Table 1-3 DMA Counter Structure

Addressing Mode	Bit Structure Description																							
	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
No update	Not used																							
Linear	DCO																							
2D	DCOH												DCOL											
3D mode C	DCOH												DCOM						DCOL					
3D mode D	DCOH								DCOM										DCOL					
3D mode E	DCOH								DCOM								DCOL							

Note: In No Update mode, the counter is not used. In Linear mode, the counter is used as a single register. In 2D mode, the counter is used as two registers DCOH and DCOL. In 3D mode, the counter is used as three registers DCOH, DCOM, and DCOL which can be different lengths, depending on the selected mode (i.e., C, D, or E).

Table 1-4 DMA Status Register Structure

Bit No.	Bit Names	Value	Function
23–12	These bits are reserved.		
11–9	DCH[2:0]	—	These bits identify the active channel number
		000	DMA Channel 0
		001	DMA Channel 1
		010	DMA Channel 2
		011	DMA Channel 3
		100	DMA Channel 4
		101	DMA Channel 5
		110	reserved
		111	reserved
8	DACT	0	The active DMA channel is disabled or awaiting DMA requests.
		1	The active DMA channel is performing a transfer.
7–6	These bits are reserved.		
5	DTD5	0	DMA channel 5 has not completed a transfer.
		1	DMA channel 5 has completed all requested transfers.
4	DTD4	0	DMA channel 4 has not completed a transfer.
		1	DMA channel 4 has completed all requested transfers.
3	DTD3	0	DMA channel 3 has not completed a transfer.
		1	DMA channel 3 has completed all requested transfers.
2	DTD2	0	DMA channel 2 has not completed a transfer.
		1	DMA channel 2 has completed all requested transfers.
1	DTD1	0	DMA channel 1 has not completed a transfer.
		1	DMA channel 1 has completed all requested transfers.
0	DTD0	0	DMA channel 0 has not completed a transfer.
		1	DMA channel 0 has completed all requested transfers.

SECTION 2

DMA USAGE BASIC EXAMPLES

These examples show the simplest use of one DMA channel. They assume no interference from other sources, such as other DMA channels, contentions with the core, etc.

2.1 TRANSFERRING FROM X MEMORY TO Y MEMORY

The following code causes DMA channel 0 to transfer a block of N words linearly from X internal memory to Y internal memory.

```
movep     #$source_addr,x:M_DSR0
movep     #$dest_addr,x:M_DDR0
movep     #$(N-1),x:M_DCO0              ; # of words to be transferred
movep     #$9802d4,x:M_DCR0             ; software-triggered,
                                        ; x linear -> y linear
nop                                     ; due to core pipeline
nop
jclr      #0,x:M_DSTR,*                 ; polling of DTD0
```

In this code, the value written to DSR0 is the address of the first data item to be transferred. The value written to DDR0 is the target address of the first data item to be transferred.

Note: It is not necessary to specify that the address is internal.

The DMA hardware evaluates this information automatically according to the specific DSP56300 family member memory map. The number of transfers to perform (N) is (DCO + 1); therefore, the value written to DCO0 is (N – 1).

The content of DCR0 is as follows:

1. **DE = 1** (i.e., start a transfer)
2. **DIE = 0** (i.e., disable an interrupt to the core at end of block)
3. **DTM = 011** (i.e., block transfer triggered by DE—Software-triggered mode)
4. **DPR = 00** (i.e., the priority level is zero—the lowest one)
5. **DCON = 0** (i.e., the continuous mode is not activated)
6. **DRS = 00000 (**In Software-triggered mode, this field is ignored.)
7. **D3D = 0** (i.e., non-three-dimensional mode)
8. **DAM = 101101** (i.e., source address is post-incremented by 1—Linear mode—and destination address is also Linear mode)
9. **DDS = 01** (i.e., the destination space is Y memory)
10. **DSS = 00** (i.e., the source space is X memory)

In this example, the core does not do anything while the DMA is transferring the block of data. The core polls the DTD bit of this channel and continues to execute the program after the DMA finishes transferring the data. Due to the chip pipeline, because the core

polls this bit until it equals 0, the chip must wait 2 clock cycles from the cycle in which DE is written until DTD is reset at the start of block transfer to indicate that the channel is busy. Another option is to poll the DACT bit (Bit 8) in the same DSTR until it is set. This bit indicates whether any DMA channel is actually transferring a data item. Because of the core and DMA pipelines, the core must wait 3 clock cycles (3 NOP instructions) until this bit is set and can be polled.

2.2 DOWNLOADING FROM EXTERNAL MEMORY

The example code presented in this section causes DMA channel 2 to transfer a block of N words linearly from external P memory to internal P memory. In this example, the trigger of the transfer is a request from Timer 0. This is an example of a program overlay in which the core works on a part of the program, while, in parallel, a DMA channel downloads another part of the program. It is common for an operating system to use a timer for scheduling such a DMA activity. In this example, the timer is programmed to use its most basic mode, and the DMA controller is programmed to generate a core interrupt at the end of transfer without clearing the DE bit. When the interrupt occurs, this code loads new parameters to the address registers to prepare them for the next download task that occurs when the next timer interrupt request occurs (i.e., $2 \times$ (num_comp) from the previous trigger). It doesn't matter whether the timer or the DMA is programmed first.

Note: The user must make sure that the program size N is smaller than the number of the timer cycles programmed in TCPR0. Otherwise, the DMA gets a new trigger before it finishes the previous task and ignores the new trigger.

```
M_TCSR0     EQU         $ffff8f     ;DSP56301 Timer0 control/status reg.
M_TLR0      EQU         $ffff8e     ;DSP56301 Timer0 load register
M_TCPR0     EQU         $ffff8d     ;DSP56301 Timer0 compare register
                                    ;
bclr        #9,SR                   ;enable interrupt priority levels 3,2,1
bset        #17,x:M_IPRC            ;enable DMA2 interrupt at priority
                                    ;level 1
movep       #$9,x:M_AAR0            ;AAR0 indicates to all external P space
                                    ;as static RAM
movep       #$1fffe2,x:M_BCR        ;2 wait states in area 0 access
movep       #$0,x:M_TLR0            ;initial value of the timer counter
movep       #$num_comp,x:M_TCPR0    ;number of CLK/2 cycles until a trigger
                                    ;is generated
movep       #$201,x:M_TCSR0         ;Timer0 enable at mode 0 + reload
movep       #$ext_addr,x:M_DSR2     ;external address
movep       #$int_addr,x:M_DDR2     ;internal address
movep       #$(N-1),x:M_DCO2        ;N is number of words to ne downloaded
movep       #$e082da,x:M_DCR2       ;block transfer triggered by timer0
...                                 ;p linear -> p linear
```

```
        ...                             ;DE is not cleared at end of block
        ...                             ;interrupt is generated at the
                                        ;end of block
                                        ;channel priority 0 (doesn't matter in
                                        ;the example)
        org p:I_DMA2
I_DMA2  movep   (pointer1),x:M_DSR2     ;interrupt routine: prepare parameters
        movep   (pointer2),x:M_DDR2     ;for next download
```

Note: Basically, the core can transfer a block of data from one memory space to another one. If the transfer is in data memory, then the DMA transfer saves core MIPS, but uses the same time (for transfer to/from peripheral the core does the transfer even faster using one cycle instead of two). For transfers to/from program space, however, the advantage of using the DMA is doubled, not only because it frees the core for other tasks, but also because it operates much faster. The DMA transfers a data item in 2 cycles, whereas the core must use a MOVEM instruction that takes 6 cycles.

2.3 GETTING DATA FROM THE ESSI RECEIVER

In the sample code provided in this section, DMA channel 5 is programmed to get data from the receiver of an ESSI peripheral. In this mode, the source address type must be No Update mode because the DMA reads the data register of the receiver and its address is constant. The data is transferred to a circular buffer in the internal X memory. In this example, the circular buffer is implemented using linear addressing; at the end of the buffer, an interrupt is issued and the DDR is re-programmed. The example in **Section 3.3.1** on page 3-5 shows an alternate way to implement a circular buffer with almost no intervention from the core.

Note: In this example, the DMA works indefinitely regardless of the content of the counter DCO5, but the counter is used to generate an interrupt to the core in order to jump to the top of the buffer after it is filled. The ESSI can be initialized before or after the programming of the DMA channel.

```
        M_PCRD  EQU     $ffffaf     ;56301 Port D control register
        M_RX1   EQU     $ffffa8     ;56301 ESSI1 receive data register
        M_CRB1  EQU     $ffffa6     ;56301 ESSI1 control register B
        M_CRA1  EQU     $ffffa5     ;56301 ESSI1 control register A

        bclr    #9,SR               ;enable interrupt priority levels 3,2,1
        bset    #23,x:M_IPRC        ;enable DMA5 interrupt at priority
                                    ;level 1
        movep   #$13,x:M_PCRD       ;enable ESSI1 SC0, SC1, SRD pins
        movep   #$180000,x:M_CRA1   ;24 bits per word, maximal frequency
        movep   #$02010c,x:M_CRB1   ;receiver enable, one bit clock
```

```
                                        ;SC0(RXC) and SC1(FSR) are outputs
        movep     #M_RX1,x:M_DSR5       ;address of the ESSI1 receive register
        movep     #$int_addr,x:M_DDR5   ;buffer top address in internal memory
        movep     #$(N-1),x:M_DCO5      ;N is the size of the circular buffer
        movep     #$ee62c0,x:M_DCR5     ;word transfer triggered by ESSI1 Rx
                                        ;x no update -> x linear
                                        ;DE is not cleared at end of block
                                        ;interrupt is generated at the
                                        ;end of block
                                        ;channel priority 3 (doesn't matter
                                        ;in the example)
I_DMA5
        movep     #$int_addr,x:M_DDR5   ;interrupt routine: point again to the
                                        ;circular buffer top address
```

Note: When the receiver of the ESSI is full, a trigger to the DMA is generated. This trigger is asserted until the receiver is read (by the DMA or the core). Basically, the user can program a DMA channel to transfer a block from memory to memory when a trigger from peripheral is asserted. Although the DMA operates correctly, there is a problem, because the peripheral does not deassert the Receiver Full (or Transmitter Empty) condition and can, therefore, potentially have an undetected overrun or underrun error.

SECTION 3

MULTI-DIMENSIONAL DMA TRANSFERS

The DMA is capable of transferring data in complex structures. The address mode of the source or the destination or both can be Two-Dimensional (2D) or Three-Dimensional (3D).

3.1 TWO-DIMENSIONAL (2D) TRANSFER

For a Two-Dimensional (2D) transfer, the DCO counter is divided into two counters DCOL and DCOH. An offset register is used to calculate the address jump. The total number of transfers is (DCOH + 1) × (DCOL + 1). The address for the DMA transfer is the current content of the address register (DSRi or DDRi). After the transfer is performed, if DCOL > 0, the address register is incremented by 1 and DCOL is decremented by 1. If DCOL = 0, the address register is loaded with the sum of its previous value and the content of the offset register. DCOH is decremented by 1 and DCOL is reloaded with the initial value that was written by the last core instruction. If DCOL = DCOH = 0 (i.e., the last transfer of the block), the address register is loaded with the sum of its previous value and the content of the offset register, and both DCOL and DCOH are reloaded with the initial values written by the last core instruction. With this last transfer of the block, the DMA either ceases operation or waits for a new trigger.

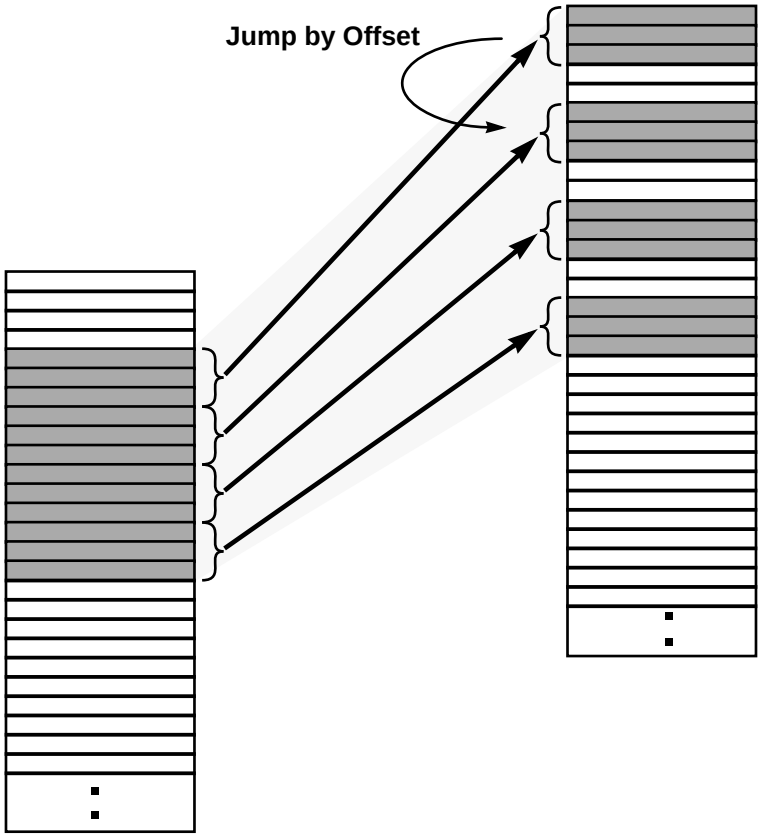


Figure 3-1 Linear to 2D Transfer

3.2 THREE-DIMENSIONAL (3D) TRANSFER

For a Three-Dimensional (3D) transfer, the DCO divides into three counters: DCOL, DCOM and DCOH. This transfer mode uses two offset registers to calculate the address jump. The total number of transfers is (DCOH + 1) × (DCOM + 1) × (DCOL + 1). The DMA transfer address is the current content of the address register (DSRi or DDRi). After performing the transfer, if DCOL > 0, the address register is incremented by 1 and DCOL is decremented by 1. If DCOL = 0 and DCOM > 0, the address register is loaded with the sum of its previous value and the content of the first offset register, DCOM is decremented by 1 and DCOL is reloaded with the initial value written by the last core instruction. If DCOL = DCOM = 0 and DCOH > 0, the address register is loaded with the sum of its previous value and the content of the second offset register, DCOH is decremented by 1, and both DCOL and DCOM are reloaded with the initial values written to the DCO by the last core instruction. If DCOL = DCOM = DCOH = 0 (i.e., the last transfer of the block), the address register is loaded with the sum of its previous value and the content of the second offset register, and DCOL, DCOM and DCOH are reloaded with the initial values written to the DCO by the last core instruction. After this last transfer of the block, the DMA either ceases operation or waits for a new trigger.

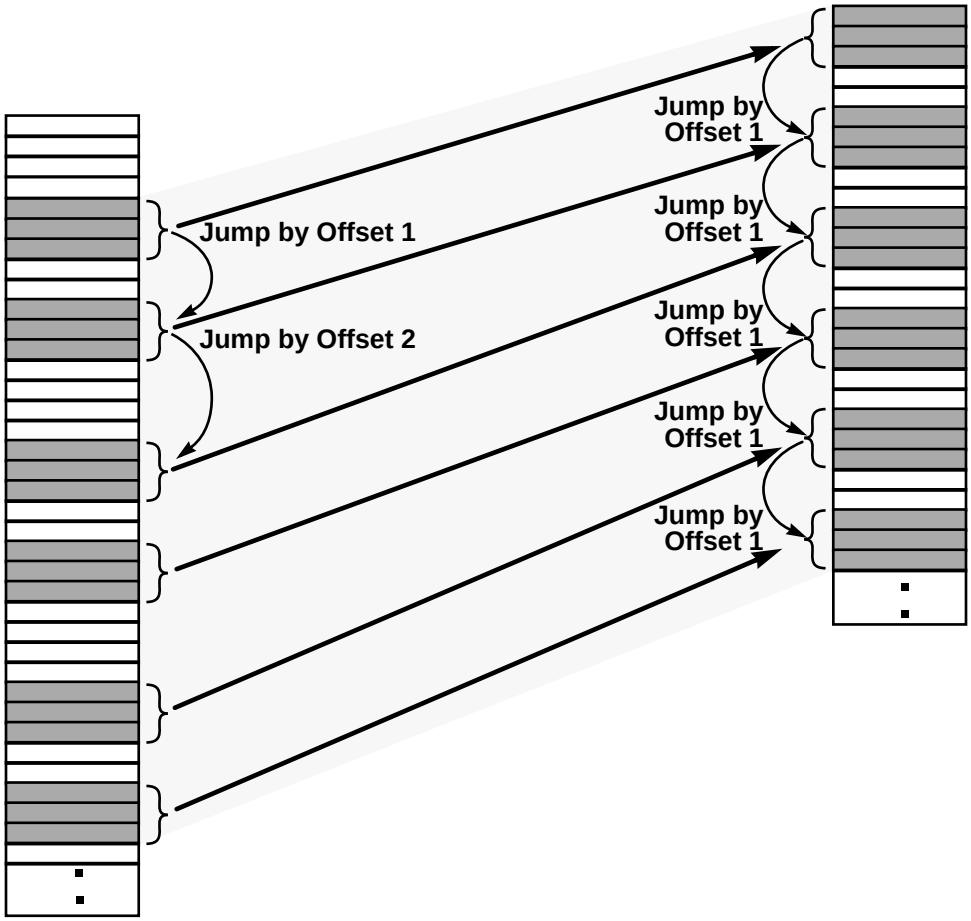


Figure 3-2 3D to 2D Transfer

3.3 EXAMPLES FOR OTHER ADDRESS TYPES

The following examples show how to use 2D and 3D Addressing modes to transfer data between other address types.

3.3.1 Circular Buffer Using 2D Addressing

The example code presented in this section transfers one data word per request until the end of the circular buffer, and then jumps back by a negative value offset register to the head of the buffer. Unlike the alternate code presented in **Section 2.3** on page 2-5, the following code does not require the core to handle any interrupts.

```
M_PCRD      EQU         $ffffaf         ; DSP56301 port D control register
M_RX1       EQU         $ffffa8         ; DSP56301 ESSI1 receive data register
M_CRB1      EQU         $ffffa6         ; DSP56301 ESSI1 control register B
M_CRA1      EQU         $ffffa5         ; DSP56301 ESSI1 control register A
movep       #$13,x:M_PCRD               ; enable ESSI1 SC0, SC1, SRD pins
movep       #$180000,x:M_CRA1           ; 24 bits per word, maximal frequency
movep       #$02010c,x:M_CRB1           ; receiver enable, one bit clock
movep       #$-(N-1),x:M_DOR0           ; this is the offset to get back
movep       #M_RX1,x:M_DSR5             ; address of the ESSI1 receive register
movep       #$int_addr,x:M_DDR5         ; buffer top address in internal memory
movep       #$(fff000+(N-1)),x:M_DCO5
movep       #$ae6040,x:M_DCR5           ; word transfer triggered by ESSI1 Rx
                                        ; x no update -> x 2D with DOR0
                                        ; DE is not cleared at end of block
                                        ; interrupt is not generated at the
                                        ; end of block
                                        ; channel priority 3 (doesn't matter
                                        ; in the example)
```

3.3.2 Transfers with Equidistant Offset

The DSP56300 core Address Generation Unit (AGU) has an address mode called Post-increment By Offset Nn that uses the syntax (Rn) + Nn. This mode can be used, for example, to perform a decimation of samples stored in a large array. Using the DMA to perform the same task can free the pointer Rn. If the user programs a DMA channel to be 2D with DCOL = 0, then for every transfer, the address register is updated by offset N stored in the offset register DORi. Programming the same channel as 3D with DCOH = 0 and DCOL = 0, but with DCOM > 0 and DORi = N (offset) and DORj = –(N × DCOM) implements a circular buffer that performs an equivalent decimation algorithm (i.e., the resulting address mode is similar to using the modulo modifier $mod_{Mn}((Rn) + Nn)$).

3.3.3 Transferring Data for 3D ESSI Transmitters

In the example code presented in this section, three ESSI transmitters send data simultaneously from a DMA channel by using the Line Transfer mode and a 2D addressing mode to wrap around back to the first transmitter. In this example, the 3D mode is used to allow the source to generate a circular buffer. The 3D mode is needed because when DCOL = 0, the destination (ESSI registers) uses an offset and the buffer must also use an offset. Therefore, this offset is 1. The second offset is used for wrapping back to top_of_buffer. An interrupt is needed only after the buffer is scanned the number of times indicated by the contents of DCOH (i.e., DCOH + 1). In this code, the first write operation to the transmitters is done by the core because after reset, the TDE bit (Transmitter Data Empty) in the ESSI Status Register (Bit 6 in SSISR) is cleared, and therefore a DMA request can not be issued for the first word. In order not to change the code between the first buffer pass and all other passes, the first write is taken by the core in all passes.

```
M_PCRC     EQU        $ffffbf        ; DSP56301 port C control register
M_TX00     EQU        $ffffbc        ; DSP56301 ESSI0 trans.0 data register
M_TX01     EQU        $ffffbb        ; DSP56301 ESSI0 trans.1 data register
M_TX02     EQU        $ffffba        ; DSP56301 ESSI0 trans.2 data register
M_CRB0     EQU        $ffffb6        ; DSP56301 ESSI0 control register B
M_CRA0     EQU        $ffffb5        ; DSP56301 ESSI0 control register A
bclr       #8,SR                     ; enable interrupt priority levels 3,2,1
bset       #13,x:M_IPRC              ; enable DMA0 interrupt at priority
                                     ; level 1
movep      #$2f,x:M_PCRC             ; enable five pins for ESSI0
movep      #$180000,x:M_CRA0         ; 24 bits per word, maximal frequency
movep      #$01d12c,x:M_CRB0         ; synchronous normal mode, one bit clock
                                     ; three transmitters
movep      #$fffffd,x:M_DOR2         ; DOR2 = -2
movep      #$1,x:M_DOR0              ; regular increment
movep      #$-buf_size,x:M_DOR1      ; for jumping back to the top of buffer
movep      #$int_buf,x:M_DSR0        ; buffer top address in internal memory
movep      #M_TX02,x:M_DDR0          ; transmitter 2 data register
movep      #$(2^12*B1+fc0003),x:M_DCO0; the counter is divided to three
                                     ; B1 = (buf_size / 3) -1 -> DCOM
                                     ; DCOL = 2, DCOH = $3f
movep      #$d65d20,x:M_DCR0         ; line transfer triggered by ESSI0 Tx
                                     ; x 3D with DOR0, DOR1 -> × 2D with DOR2
                                     ; DE is cleared at end of block
                                     ; interrupt is generated at the
                                     ; end of block
                                     ; channel priority 3 (doesn't matter
                                     ; in the example)
I_DMA0
bset       #23,x:M_DCR0              ; interrupt routine: start again
```

SECTION 4

OPERATION OF MULTIPLE DMA CHANNELS

Because a DMA channel can be triggered by one of thirty-two sources, multiple DMA tasks can be performed orthogonally or concurrently. This section discusses use of multiple DMA channels in applications.

4.1 INTRODUCTION

This section provides several examples that employ multiple DMA channels.

4.2 COLOR COMPRESSION EXAMPLE 1

Any DMA channel can be triggered by one of thirty-two sources: four external lines ($\overline{\text{IRQA}}$, $\overline{\text{IRQB}}$, $\overline{\text{IRQC}}$, and $\overline{\text{IRQD}}$), twenty-two event sources from peripherals, and six "end of block transfer of channel A triggers the start of work of channel B" triggers. These last six modes are useful if two DMA tasks must be performed sequentially, or when one DMA task is in progress while the core programs a second DMA channel with all the parameters for a second task. When the first task is done, the second task starts, and the core can program new parameters for the first DMA to be performed after the second DMA channel is done. In this way, two DMA tasks can be orthogonal to each other.

In color television, every pixel is represented by 3 bytes corresponding to the colors Red, Green, Blue. In the PAL television system, the R,G,B representation is transformed to a Y,U,V representation in which Y represents brightness and U and V represent the color components according to the following equations:

$$Y = 0.299 * R + 0.587 * G + 0.114 * B \quad (0.1)$$

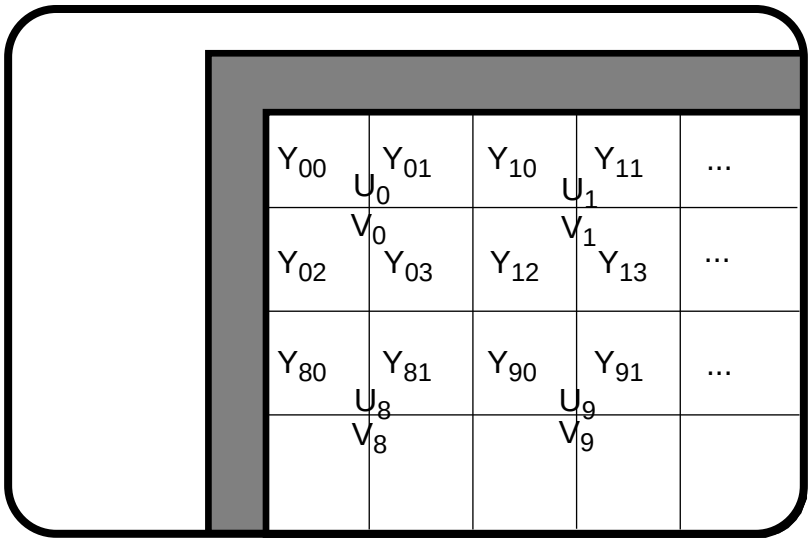
$$U = (B - Y) / 2.03 \quad (0.2)$$

$$V = (R - Y) / 1.14 \quad (0.3)$$

By using the Y,U,V form, a 50% compression can be achieved, because, for every 2×2 pixels, instead of 4 bytes each to represent R, G, and B, only 4 bytes are required for Y (one for each item) and one common byte each for U and V, without significantly reducing the picture quality. Standards like CCIR-601, MPEG, and H.261 support this Y, U, V representation. The data is stored according to the MPEG standard in 16-bit wide words according to the following memory map.

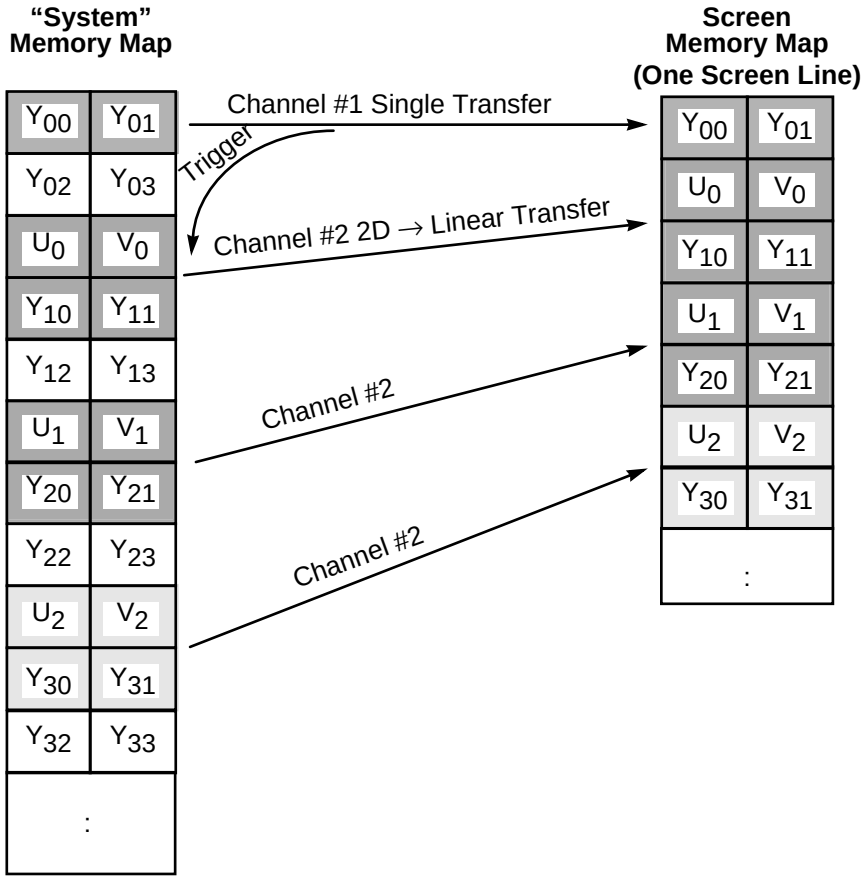
ADDRESS	DATA
000000	Y00, Y01
000001	Y02, Y03
000002	U0, V0
000003	Y10, Y11
000004	Y12, Y13
000005	U1, V1
000006	Y20, Y21
000007	Y22, Y23
000008	U2, V2

The representation in the screen memory is shown in **Figure 4-1**. A transfer from system memory to screen memory of one line is shown in **Figure 4-2**.



AA1371

Figure 4-1 Screen Memory Representation



AA1372

Figure 4-2 Transfer from System Memory to Screen Memory

In the example code listed below, the first DMA channel is needed to transfer the first data item of Y00,Y01 because it is irregular to the structure of other data items. The second DMA channel is triggered by the first one and transfers from 2D to linear when DCOL = 1 (i.e., one increments and one jumps).

```
movep     #$2,x:M_DOR0          ; jump two places
movep     #Sys_Mem,x:M_DSR1     ; address of first data item
                                ; in System mem
movep     #Line1,x:M_DDR1       ; address of first data item in Line mem
movep     #$0,x:M_DCO1          ; transfer one data item
movep     #$860240,x:M_DCR1     ; block transfer triggered by pin irqa_
                                ; (assuming pin is connected to Vsync)
                                ; clear DE after transfer
                                ; x no update -> x no update
                                ; no interrupt at end of block
movep     #Sys_Mem+2,x:M_DSR0   ; address of U0,V0
movep     #Line1+1,x:M_DDR0     ; address of second data item in Line mem
movep     #$11f001,x:M_DCO0     ; DCOH = $11f = 287, DCOL = 1
                                ; 288 pixel pairs, i.e. 576 pixels
                                ; in line
movep     #$c62a80,x:M_DCR0     ; block transfer triggered by DTD1
                                ; clear DE after transfer
                                ; x 2D with DOR0 -> x linear
                                ; interrupt at end of block
```

4.3 PRIORITIES BETWEEN CHANNELS

Each DMA channel has a priority defined by bits DPR1–DPR0 in the DCR of that channel. The priority can be 0, 1, 2 , or 3. Prioritization is needed because only one DMA channel can issue a transfer in a single clock cycle due to the fact that there is only one DMA Address Bus (DAB). The arbitration hardware in the DMA machine determines which channel is the next to issue a transfer on a per access basis (i.e., when the destination address is placed on DAB). If the active DMA channels have different priorities, then the channel with the highest priority issues a transfer and all the others wait. If there are some active channels with the same priority, a round-robin method is used (i.e., each channel issues one word periodically). The selected channel priority is used to compare the priority between the DMA and the core (discussed in **Section 5.5** on page 5-6). **Figure 4-3** on page 4-6 is an example of the priority between the channels. In the example, channels 0 and 1 have priority level 0; channels 2, 3, and 5 have priority level 1; and channel 4 has priority level 2.

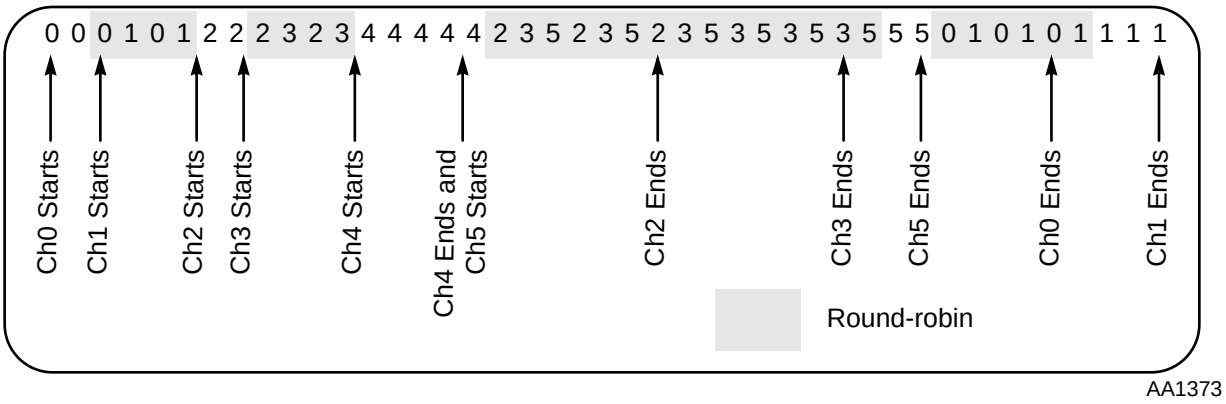


Figure 4-3 Example of Multi-channel Operation

Note: The round-robin algorithm promises a equality of transfer between channels that have the same priority, but it does not assure that the number of words transferred by channel A are exactly the same number of words transferred by channel B in the same time. This is because if a third channel with higher priority interferes in the middle, the system does not remember which channel was the last to issue a word before that. Therefore, a sequence of 23232444444232 is possible. In such a sequence, channel 2 transfers 5 words and channel 3 transfers only 3 words.

4.4 COLOR COMPRESSION EXAMPLE 2

The code presented in this section is an example of the same transfer from system memory to screen memory as described in **Section 4.2**, but with a window size QCIF (176 x 144 pixels). This size is a quarter of video recording quality in a screen. In this case, the data of the QCIF window is first copied to another temporary buffer in the format of system memory, and then the data is transferred again to the screen memory.

Because this process must be repeated, the first transfer must use 3D mode in order to return to the start of the buffer in the memory system. The challenge is that this example must use a counter with DCOL = 263, DCOM = 71 and DCOH = 0. This division can not be implemented in one counter because 12 bits are needed for DCOL, and more than 6 bits are needed for DCOM. The solution implemented here makes the first transfer using two channels with each transferring half of the block.

For the second transfer (from temp_buffer to system memory), the example must transfer addresses 0 and 2, 3 and 5, and so forth. This is effected by using a second set of DMA channel pairs that work in a round-robin fashion. The first channel transfers addresses 0, 3, 6, and so forth; the second channel transfers addresses 2, 5, 8, and so forth.

Because they use the highest priority (3), other DMA channels can not ruin the exactness of this round-robin. Both channels are triggered by DTD1 (i.e., the whole QCIF block is transferred to the temporary buffer). In this example, minimum core intervention is needed to reload channels 2 and 3 after half of the QCIF data is placed in the temp_buf (with a jump back to start of temp_buf for the interlaced lines). **Figure 4-4** illustrates the DMA transfers in this example.

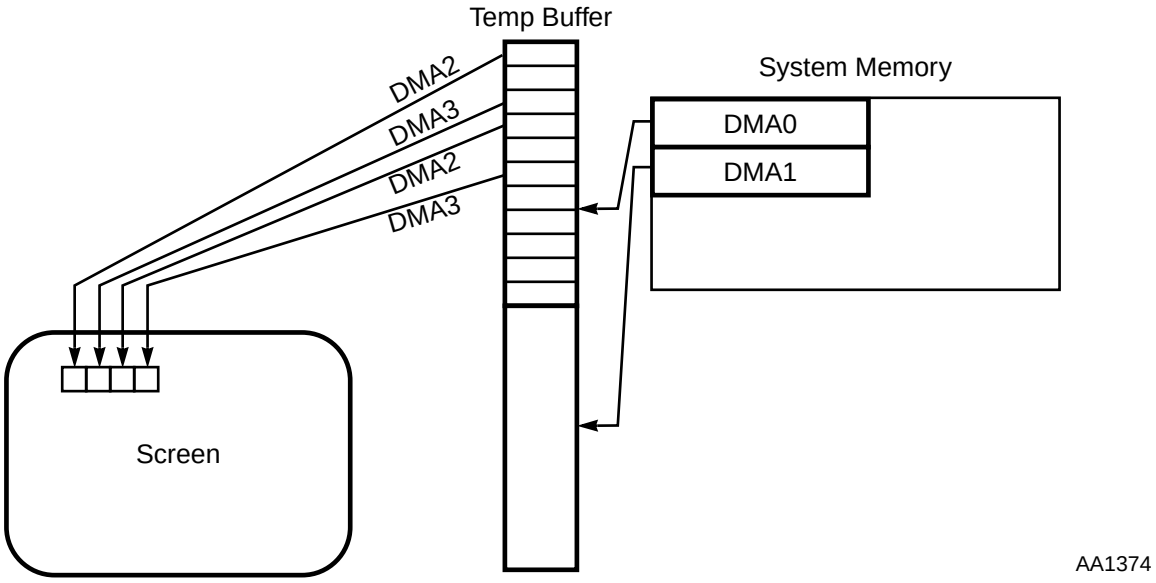


Figure 4-4 QCIF Block Transfer Example

```
movep     #264,x:M_DOR0             ; jump over half a line in system mem.
movep     #-(528*36),x:M_DOR1       ; jump back to start of system mem.
movep     #3,x:M_DOR2               ; jump of 3 in source temp_buf
movep     #2,x:M_DOR3               ; jump of 2 in screen memory
movep     #Sys_Mem,x:M_DSR0         ; address of first data in System mem
movep     #Temp_buf,x:M_DDR0        ; address of first data in Temp_buf
movep     #$023107,x:M_DCO0         ; DCOL = 176*1.5 - 1 = $107
                                    ; DCOM = 36 - 1 = $23, DCOH = 0 mode E
movep     #Sys_Mem+528*36,x:M_DSR1  ; address after 36 double lines
movep     #Temp_Mem+264*36,x:M_DDR1 ; address in temp_buf
                                    ; after 36 double
movep     #$023107,x:M_DCO1         ; same as DCO2
movep     #$a406a4,x:M_DCR0         ; block transfer triggered by irqa_
                                    ; (assuming pin is connected to Vsync)
                                    ; don't clear DE after transfer
                                    ; priority level is 2
                                    ; x 3D, DOR0,DOR1 DCO-E -> y linear
                                    ; no interrupt at end of block
movep     #$a426a4,x:M_DCR1         ; same but trigger by DTD0
movep     #Temp_buf,x:M_DSR2        ; address of Y00,Y01
movep     #Line1,x:M_DDR2           ; first address in screen memory
movep     #Temp_buf+2,x:M_DSR3            ; address of U0,V0
movep     #Line1+1,x:M_DDR3         ; second address in screen memory
movep     #$c5f000,x:M_DCO2         ; DCOL=0, DCOH = 88*36 - 1(half QCIF)
```

```
            movep       #$c5f000,x:M_DCO3
            movep       #$862931,x:M_DCR2          ; both channels with same mode
            movep       #$c62931,x:M_DCR3          ; block transfer triggered by DTD1
                                                   ; y 2D with DOR2 -> x 2D with DOR3
                                                   ; priority level 3 (for round-robin)
                                                   ; enable interrupt at channel 3
                                                   ; (for new setting of DSR2,3 and DDR2,3
                                                   ; to continue the transfer)
I_DMA3      jmp         long_subroutine
long_subroutine:
            checks which of the three settings is needed
            update DSR2, DSR3, DDR2, DDR3, DCR2, DCR3 to continue transferring
            rti
```

4.5 CONTINUOUS MODE

Each DMA channel can be in Continuous mode, depending on the value of the DCON bit in the DMA Control Register (DCTR) for the channel. If this bit is set while transferring data, then other channels with the same priority can not interfere until the whole line or block is transferred (according to the transfer mode). However, enabled channels with higher priority can access the bus and finish their word, line, or block in spite of Continuous mode selection. **Figure 4-5** is an example similar to **Figure 4-1** on page 4-4 except for the fact that channel 5 is in continuous mode.

Note: If a channel has a high priority, but is waiting for a trigger, another active channel that has data to transfer can do so, even if the first channel is in Continuous mode. So, if the user designs a system in which a block is transferred externally without interference as a word or line transfer, the design should also ensure that the triggering is continuous.

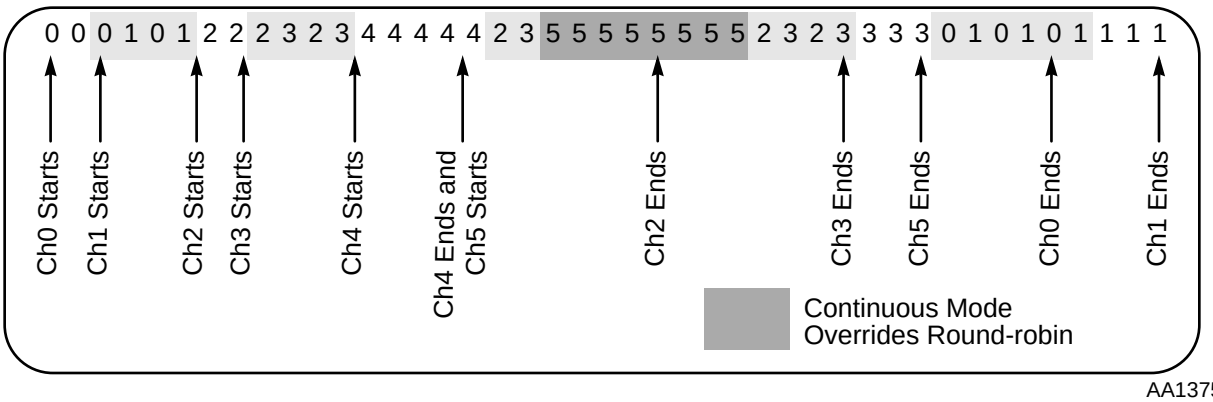


Figure 4-5 Example of Multi-channel Operation with Continuous Mode

SECTION 5

DMA AND CORE CONTENTION

Because the DSP core and the DMA controller can both access the internal memories independently, there is the possibility that the two can compete for access to the same space simultaneously. The same is true for access to the internal peripheral registers. Using the DMA controller efficiently requires that the programmer understand how to minimize such access contention and also understand the prioritization scheme that controls such accesses if they are unavoidable.

5.1 CONTENTION FOR INTERNAL MEMORY

Memories in the DSP563xx family are dual access because both the core and the DMA controller have separate address and data buses connected to the memories. The memories are designed as small modules each having 1/4 K word (for RAM) or 3 K words (for ROM). If the core and DMA access different memory modules, they can work in parallel without interfering with each other. If, however, the DMA and core attempt to access the same memory module, the DMA halts (its internal clocks are stopped) until the core finishes its access to the module. Therefore, to get maximum throughput from the DMA, it is advisable to ensure that the core and the DMA work on different memory modules (i.e., never access the same module simultaneously).

Note: DMA access to internal memory does not reduce available core MIPS.

5.2 CONTENTION FOR PERIPHERAL REGISTERS

The DMA can only access the data registers of DSP563xx peripherals. These registers are read-only or write-only by their nature. Typically, it is bad design practice to allow simultaneous access to a data register by both the core and the DMA controller. Although both core and DMA can *read* the same read-only data register in the same clock cycle, and both get the correct value from it with no delay, if both the core and the DMA *write* to the same write-only data register in the same clock cycle, only the core data is written to the register and the data from the DMA *is lost*. Secondly, because reading from or writing to a data register may affect some flags in a peripheral status register, allowing double accessing prevents the user from determining the cause of a status change.

5.3 PRIORITIES ON EXTERNAL ACCESS

As discussed in **Section 4**, every DMA channel has a priority defined in bits DPR[1:0] of the channel DMA Control Register (DCTR). The priority of the currently selected channel (which is, of course, the highest priority of the channels scheduled to transfer data) is referred to as the current DMA priority. Because there is only one external port, if the DMA is required to initiate an external access, its priority must be compared with the core priority. The priority of the core is defined by the Core Priority (CP[1:0]) bits in the Status Register (SR[23:22]). The relative priority between the DMA and the core is defined by the Core-DMA Priority (CDP[1:0]) bits in the Operation Mode Register (OMR[9:8]), as follows:

- If CDP[1:0] = 11, then DMA accesses always have lower priority than the core accesses.
- If CDP[1:0] = 10, then DMA accesses always have the same priority as the core accesses.
- If CDP[1:0] = 01, then DMA accesses always have higher priority than the core accesses.
- If CDP[1:0] = 00, then the priorities of the DMA and core are compared dynamically to decide which access is performed first.

If the DMA priority is programmed to be higher than the core priority (CDP[1:0] = 01), then the DMA performs the external access and the core waits until the DMA finishes all its required external accesses. If the DMA downloads a program from external to internal memory, the core can get the external bus to initiate one access in the destination slot of the DMA which does not need the external bus. However, if the core needs the external bus for some memory space accesses, its performance degrades substantially. If the DMA transfers a block from external memory to external memory, the core halts all its activity the first time it needs the external bus until the DMA finishes all its task. This mode of operation is useful only if the user must perform a critical task and it is acceptable for the core to be halted for a long period. If the DMA priority is higher than the core priority, and the DCON bit in the selected channel is set (i.e., Continuous mode is selected), then the core is not allowed to initiate an access in the next slot after the DMA has used the external bus. The Continuous mode is useful if the DMA is transferring a block from external DRAM memory to internal memory (or vice-versa). If the core were allowed to change the address during its access, this would result in a page miss for every DMA access and the whole block transfer would require a longer total time. However, if a DRAM refresh cycle is required, it is performed by the core when needed even if the DMA priority is higher than core priority and the DCON bit is set.

If the DMA priority is programmed to be equal to the core priority (CPD[1:0] = 10), then if the DMA and core both require external access, the core performs all its external accesses pertaining to the current instruction in the order P, X, Y, and then the DMA performs its access (one word at a time).

If the DMA priority is programmed to be lower than the core priority (CPD[1:0] = 11), then if the DMA requires an external access, it must wait until it gets a free slot in which the core does not need the external bus.

In the Dynamic Priority mode (CPD[1:0] = 00), it is possible that a DMA channel may be halted by the core having a higher priority in the source slot or the destination slot. If another DMA channel with higher priority is triggered, this raises the DMA priority and it can get the bus for finishing the halted transfer of the first channel (only if it was

started) and then the new channel gets the bus with equal priority or higher priority of the core as programmed in its DPR bits.

Note: Even if the new channel doesn't need external accesses, this push out of the stuck word of the first channel occurs, and be seen on the external bus, in order that the new channel is able to start working.

5.4 PACKING/UNPACKING MODE

The DMA can support external accesses to / from 8-bit wide external memory. This mode is activated by setting Bit 7 (BPAC) of the Address Attribute Register (AAR) being used for that DMA access. If this mode is selected, the DMA must program the external access to be 2D or 3D with DCOL = 0 and DORi = 3. The DMA issues the addresses as jumps of three (i.e., DAB, DAB + 3, DAB + 6, etc.). The Bus Interface Unit (BIU) halts the DMA internal clocks and generates three consecutive accesses at DAB, DAB + 1, and DAB + 2 for each address. For the first access, the least significant byte of the data is written to the eight least significant data pins of Port A, or these pins are sampled to generate the low byte of data. For the second access, the middle byte of the data is written to the eight least significant data pins of Port A, or these pins are sampled again to generate the middle byte of the data. For the third access, the most significant byte of the data is written to the eight least significant data pins of Port A, or these pins are sampled again to generate the high byte of data.

Some comments on Packing mode:

1. Packing is considered for DMA accesses only, and ignored during core accesses.
2. DAB + 1 and DAB + 2 should not cross the AAR bank border otherwise improper operation may result.
3. Arbitration between DMA channels on the external bus and between DMA and core does not take place during the packing access (i.e., the chip treats this access as one DMA external access with more wait states and does not stop it).
4. Arbitration on the external bus during the packing access is also not allowed (i.e., the chip does not yield mastership of the bus until the whole packing access is finished).
5. Packing mode is not allowed to Synchronous SRAM with zero wait states; otherwise, improper operation may result.
6. The DMA is halted by the BIU for three times the programmed number of wait states + 3.

Example of code using Packing mode (very similar to the code in **Section 2**):

```
M_TCSR0     EQU         $ffff8f                 ; DSP56301 Timer0 control/status reg.
            M_TLR0      EQU         $ffff8e     ; DSP56301 Timer0 load register
            M_TCPR0     EQU         $ffff8d     ; DSP56301 Timer0 compare register
            bclr        #9,SR                   ; enable interrupt priority levels 3,2,1
            bset        #17,x:M_IPRC            ; enable DMA2 interrupt at priority
                                                ; level 1
            movep       #$89,x:M_AAR0           ; AAR0 indicates to all external P space
                                                ; as static RAM and with packing mode
            movep       #$1fffe2,x:M_BCR        ; 2 wait states in area 0 access
            movep       #$0,x:M_TLR0            ; initial value of the timer counter
            movep       #$num_comp,x:M_TCPR0    ; number of CLK/2 cycles until a trigger
                                                ; is generated
            movep       #$201,x:M_TCSR0         ; Timer0 enable in mode 0 + reload
            movep       #$3,x:M_DOR3            ; jumps of 3 between the addresses
            movep       #$ext_addr,x:M_DSR2     ; external address
            movep       #$int_addr,x:M_DDR2     ; internal address
            movep       #4096*(N-1),x:M_DCO2    ; DCOH = N-1 when N is number of 24 bit
                                                ; words to be stored; DCOL=0
            movep       #$e082ba,x:M_DCR2       ; block transfer triggered by timer0
                                                ; p 2D with DOR3 -> p linear
                                                ; DE is not cleared at end of block
                                                ; interrupt is generated at the
                                                ; end of block
                                                ; channel priority 0 (doesn't matter in
                                                ; the example)
I_DMA2      movep       (pointer1),x:M_DSR2     ; interrupt routine: prepare parameters
            movep       (pointer2),x:M_DDR2     ; for next download
```

5.5 CORE ACCESSES THE DMA IN MID-OPERATION

None of the DMA address registers (DSRi, DDRi, DORi) or the counter (DCOi) should be written to by the core if it is not guaranteed that the channel using them is not in the middle of processing data. This is because the operation of the DMA is asynchronous to the core operation, and therefore the cycle in which the core writes to a register used by a current DMA activity can not be guaranteed. However, all the DMA registers can be read by the core in any stage and the value that is read reflects the status of the DMA in the exact cycle of the read operation (and of course it can be updated right afterwards). No control bit in DCRi register should be changed while the channel is active. The correct way to change parameters of a DMA channel while in operation (e.g., to allocate this channel to another task) is to disable it as follows:

```
            bclr        #M_DE,x:M_DCRi          ; clear DE bit in DCR
            jclr        #M_DTDi,x:M_DSTR,*      ; poll on Transfer Done
                                                ; to see that transfer was stopped
            change DMA registers
```

When a DMA channel is so disabled, the channel responds in one of the following two ways:

1. If the channel is programmed by the TM[2:0] bits to work in Block Transfer or Line Transfer mode and it is in the middle of a transfer, then the current word is stored in the destination, and the block or line is stopped in the middle. If the channel has captured a trigger but didn't start transferring data because of low priority compared to another channel, then the channel is stopped immediately without starting the transfer (i.e., the request is ignored).
2. If the channel is programmed by TM[2:0] bits to work in Word Transfer mode, and if a request was accepted, DTD is not asserted high until the current word is transferred, even if it must wait until other channels with higher priorities finish transferring a whole block before transferring the current word. If, however, the channel is disabled before a request is asserted, DTD is asserted immediately. This is because in Word Transfer mode, the DMA must support fast peripherals (see **Section 6**) in which any request that is already generated must be handled by the DMA.

5.6 DMA INITIALIZATION AFTER RESET

After hardware reset, all the peripherals and the DMA must be initialized. It does not matter whether a peripheral is programmed first or the DMA channel triggered by this peripheral is programmed first, because after the peripheral is initialized it can assert a trigger and this trigger remains asserted until the DMA channel actually accesses the peripheral data register. The same is true also for the HI32 because the DMA must assert an enable line before the first trigger can occur. The only module that must be initialized after its DMA channel is initialized is the timer, because it asserts one cycle trigger regardless of DMA access to any register, and if the timer is initialized first, the first trigger occurs earlier than expected.

5.7 WAIT INSTRUCTION

The WAIT instruction is defined as "Wait for Interrupt or DMA request". Therefore, DMA channels can be programmed to be triggered by peripherals and, after all the needed channels are enabled by setting DE in the control registers, the WAIT instruction is initiated. In this state, most of the internal clocks of the DSP56300 core and most of the internal clocks of the DMA are halted. When a request from peripheral triggers a programmed DMA channel, the core leaves the Wait processing state and the DMA starts transferring the data item(s). If a WAIT instruction is decoded by the core while

the DMA is transferring data, the DMA prevents the core from entering the Wait processing state. Therefore, when entering the Wait processing state, the user should assure that no DMA channels is transferring data. This can be guaranteed by polling bit DACT (Bit 8) of the DMA Status Register (DSTR) to make sure it is 0.

5.8 STOP INSTRUCTION

In the Stop processing state, all activity in the chip is stopped and all the clocks are halted. Before entering Stop, the DACT bit in DSTR must be polled to check that the DMA is not actually transferring data. After that, it must be also guaranteed (before entering the Stop processing state), that the DMA is not activated unintentionally after leaving the Stop processing state. Therefore, channels for which the request line is an external pin ($\overline{\text{IRQA}}$, $\overline{\text{IRQB}}$, $\overline{\text{IRQC}}$, $\overline{\text{IRQD}}$) must be disabled (clear DE bit) before entering the Stop processing state. Channels triggered by requests that can be asserted when leaving the Stop processing state must also be disabled before entering the Stop processing state (e.g., serial port transmitters, because the Stop processing state sets the Transmitter Empty condition and, therefore, issues a false request). Another possibility is to disable the peripherals individually (see the peripheral module descriptions in the device *User's Manual*), and then the DMA channels can remain active (DE bit set). More complex is the situation with DMA channels that are programmed to be triggered by HI32 (e.g., in the DSP56301). These channels must be disabled first and then the appropriate DTD bits in DSTR must be polled for 1 (i.e., the channels are inactive). After that, the HI32 itself must be programmed to initiate and individual reset by writing 000 in bits HM[2:0] in the DCTR and then polling the HACT bit in the DSR for 0. Only after all these conditions are met, can the chip be programmed to enter the Stop state correctly.

5.9 DEBUG MODE

Before entering the Debug mode, the chip checks to make sure that the current access is not in the middle of operation because of contention or external access wait states. When the access is finished (source or destination) the chip enters Debug mode. All the DMA activity in this mode is preserved and the OnCE™ module operation cannot affect it. Therefore, the DMA can resume normal operation after leaving the Debug mode.

SECTION 6

HI32 DMA OPERATION

This section provides examples of how to use DMA transfers with the HI32 in the Universal Bus (UB) mode and the HI32 in the PCI Bus mode.

6.1 INTRODUCTION

The DMA can be triggered from twenty-two different peripheral device sources. For a regular peripheral request, the trigger to the DMA remains set until the appropriate register at the peripheral is accessed by the DMA. Therefore, the peripheral can not generate a second request until the first one is served. There is, however, another method in which the peripheral (i.e., timer) generates a triggering pulse without checking whether the DMA serves this trigger. The last four source devices defined by the DRS bits (i.e., for DRS[4:0] = 111xx) are special because, in addition to the regular behavior of all the requesting devices, they can serve as “fast request sources”. The “fast peripheral” has a full duplex handshake to the DMA, enabling maximum throughput of a trigger every 2 clock cycles. This mode is functional only in the Word Transfer mode (DTM = 001 or 101). In the Fast Request mode, the DMA sets an “enable line” to the peripheral. If required, the peripheral sends the DMA a one cycle triggering pulse.This pulse resets the enable line. If the DMA arbitration mechanism decides by the priority algorithm that this trigger can be served in the next cycle, the enable line is set again even before the corresponding register in the peripheral is accessed. If the enable line is asserted, this means that the DMA can respond to a new request in all cases (i.e., even if its internal clocks are halted because of contention on the current executing transfer).

6.2 HI32 IN UNIVERSAL BUS MODE

The following sections describe the use of DMA transfers with the HI32 configured in Universal Bus (UB) mode.

6.2.1 HI32 DSP Side Registers in Universal Bus Mode

When the HI32 is configured as Universal Bus (UB) mode, the DSP side registers are:

- **DSP Control Register (DCTR)**—Bits 22–20 of the DCTR are the HI32 Mode bits (HM[2:0]). If HM[2:0] = 010, then the Universal Bus (UB) mode is selected. If HM[2:0] = 011, then the Enhanced Universal Bus (Enhanced UB) mode is selected. If HM[2:0] = 101, then the Self-configuration mode is selected. The Self-configuration mode is a special mode in which the slave HI32 can program its Host-side Registers in order to start working on the system bus. In UB mode, Bits 19–13 control all other HI32 activities. Bits 6–0 control the interrupts in the DSP and Host sides and flags to the Host side. These bits do not have to be changed from their state after reset if the HI32 is controlled by the DMA controller.

- **DSP Status Register (DSR)**—The DSR contains some status bits reflecting the value of bits in the HCTR and HCVR in the Host side. Bit 1 of this register is the Slave Transmit Data Request (STRQ) bit. It is set if the slave transmit data FIFO is not full. Bit 2 in this register is Slave Receive Data Request (SRRQ). It is set if the slave receive FIFO is not empty.
- **DSP Slave Transmit Data (DTXS) Register**—The DTXS can be written by the core or a DMA channel to transfer data from the DSP to the Host side Receiver register HRXS. Between DTXS and HRXS there is a 6-level FIFO. In the Pre-fetch mode (i.e., SFT = 0 in Host side register HCTR) the FIFO is used. In the Fetch mode (i.e., SFT = 1), the FIFO is bypassed and DTXS is connected (after synchronization) directly to HRXS.
- **DSP Receive Data Register (DRXR)**—The DRXR can be read by the core or DMA to transfer data from the Host side register HTXR into the DSP. Between HTXR and DRXR there is a 6-level FIFO.

6.2.2 HI32 DMA Operation in Universal Bus Mode

In the DSP56301, if the HI32 is connected in UB mode (or Enhanced UB mode), up to two DMA channels can be used to interact with it. One DMA channel can be programmed to be triggered by SRRQ = 1 (i.e., DRS = 11100 in the channel's control register) to transfer data from DRXR FIFO. Another DMA channel can be programmed to be triggered by STRQ = 1 (i.e., DRS = 11110 in the channel's control register) to transfer data to DTXS FIFO. Any HI32 data transfer controlled by a DMA channel can be performed at the highest level of DMA performance (i.e., one transfer every two DSP clock cycles).

Guidelines for such applications include:

- DMA channels triggered by the HI32 must be programmed to work in Word Transfer mode (i.e., TM = 001 or TM = 101 in the channel's control register).
- For the channel triggered by SRRQ, the source address must be programmed as No Update mode and the DSR must point to the DRXR. The destination address mode can be any (No Update, Linear, 2D, 3D).
- The channel triggered by STRQ must be programmed with destination address in No Update mode and the DDR must point to the DTXS. Any of the source address modes (No Update, Linear, 2D, or 3D) can be selected.

Note: Data coherency between the DSP side and the Host side can not be guaranteed if all the above requirements are not met.

Because of the handshake between the DMA and the HI32 (which is a fast peripheral), the user can program the HI32 before the DMA or the DMA before the HI32 and correct operation is guaranteed. To allocate a DMA channel that was connected to HI32 for another task, the channel DE must be cleared and then polled on the appropriate DTD bit in DSTR to assure that the channel is actually disabled. This process can take some time because if a trigger from the HI32 was already accepted by the channel, the channel can not set DTD until the word corresponding to this trigger can actually be transferred. To put the chip in the Stop state, follow the rules in **Section 5.8** on page 5-8.

6.2.3 DMA Code Example—HI32 in UB Mode

In the following example, two DSP56301 chips are connected to each other when chip A is master and chip B is slave. Port A of the master is connected to Port B of the slave and the slave is programmed as UB mode. Maximum frequency on the common bus is achieved if the $\overline{BS}$ of the master is connected to $\overline{HBS}$ of the slave, CLKOUT of the master is connected to EXTAL input of the slave, and, in the slave, the PLL multiplication and division factors are 1. In this case, accesses on the common bus can use 2 wait states.

The following DSP56301 slave code example uses two DMA channels. One channel reads the HI32 Receive FIFO and the other channel writes to HI32 Slave Transmit FIFO. The two channels have equal high priority. Control of each word is achieved by the handshake between the DMA and the HI32, but is totally transparent to the user. The 2D address mode is used in the example in order to maintain circular buffers in the internal memory. In this example, the master is the only master on Port A. The following codes require the connections shown in **Figure 6-1** on page 6-7 for correct code operation.

```
M_DCTR     EQU        $ffffc5    ; DSP56301 HI32 DSP control register
M_DPMC     EQU        $ffffc7    ; DSP56301 HI32 DSP PCI master control
                                 ; register
M_DPAR     EQU        $ffffc8    ; DSP56301 HI32 DSP PCI address register
M_DRXR     EQU        $ffffcb    ; DSP56301 HI32 DSP receive FIFO
M_DTXS     EQU        $ffffcd    ; DSP56301 HI32 DSP slave transmit FIFO

movep      #$5e8000,x:M_DCTR      ; enter the self configuration mode
                                  ; by writing HM[2:0]=101
                                  ; HRSP=1, HDRP=0, HTAP=1 configures
                                  ; HRST and HTA active low
                                  ; HDRQ active high
movep      #$000055,M_DPMC        ; this is the base address to be
                                  ; transferred to CBMA as GB10-GB3
clr        a
rep        #4
movep      a0,x:M_DPAR            ; first dummy access writes to CSTR/CCMR
                                  ; second dummy access is to CCCR/CRID
                                  ; third dummy access writes to CLAT
                                  ; fourth dummy access writes $00550000
                                  ; to CBMA
movep      #$0,x:M_DCTR           ; leave configuration, enter personal
                                  ; reset
movep      #$2e8000,x:M_DCTR      ; slave in UB mode (HM=010)
movep      #-bufr_size,x:M_DOR2   ; to roll over on receiver buffer
movep      #-buft_size,x:M_DOR3   ; to roll over on transmitter buffer
movep      #M_DRXR,x:M_DSR0
movep      #rec_buf,x:M_DDR0
movep      #bufr_size,x:M_DCO0    ; DCOH=0, DCOL= bufr_size
movep      #tran_buf,x:M_DSR1
movep      #M_DTXS,x:M_DDR1
movep      #buft_size,x:M_DCO1    ; DCOH=0, DCOL= buft_size
movep      #$eee140,x:M_DCR0      ; word transfer without clearing DE
                                  ; interrupt at end_of_block (for core to
                                  ; start processing on the buffer)
                                  ; priority level 3
                                  ; trig. by slave receiver FIFO not empty
                                  ; x no update -> x 2D with DOR2
movep      #$eef230,x:M_DCR1      ; same as channel 0 but:
                                  ;trig. by slave transmitter FIFO not full
                                  ;x 2D with DOR3 -> x no update
```

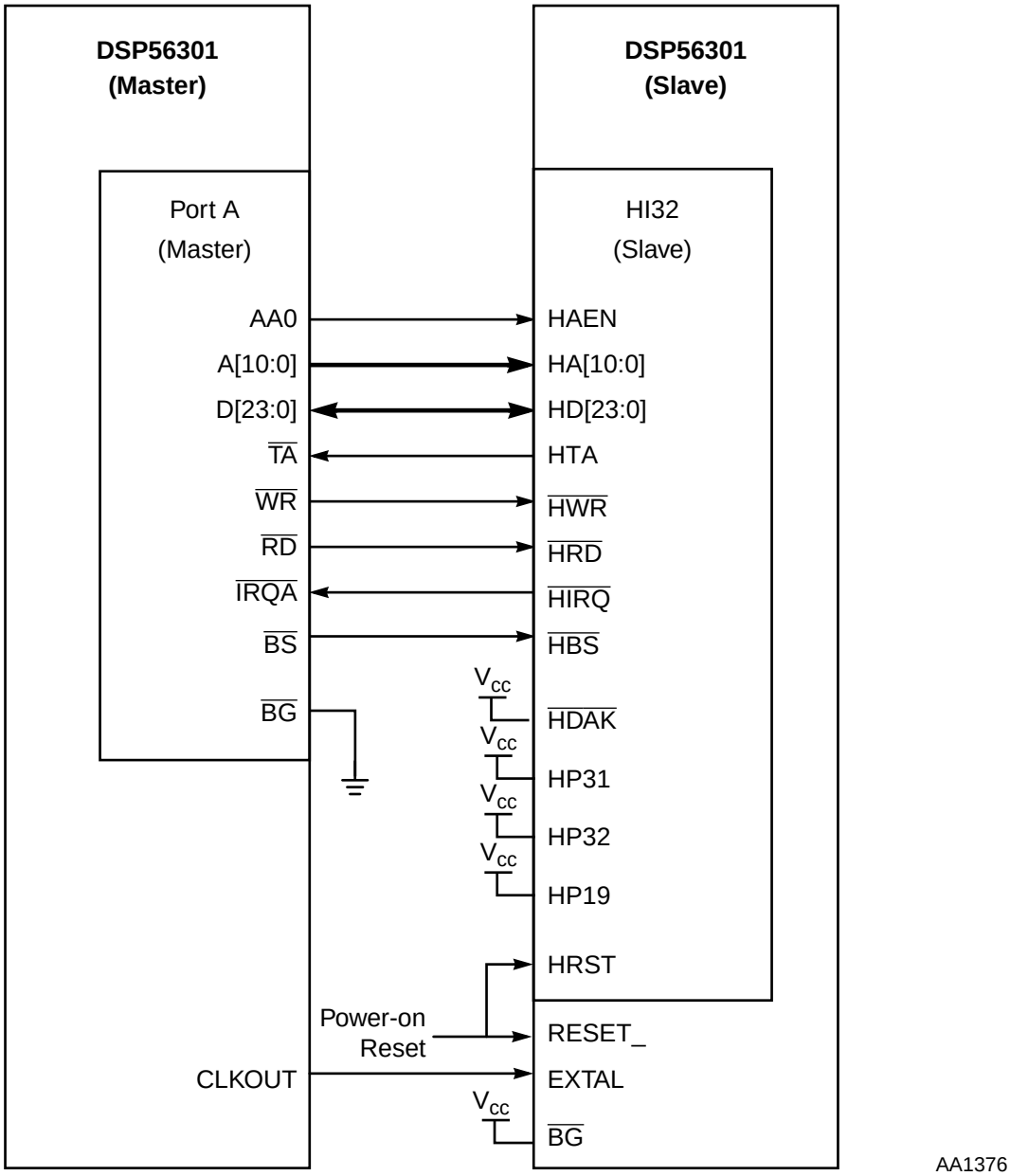


Figure 6-1 Synchronous Connection of DSP56301 Port A to DSP56301 HI32

In the DSP56301 master code example listed below, two DMA channels are used. One channel transfers a block of data from external memory to internal memory and the other DMA channel transfers a block of data from internal memory to external memory. Controllability is achieved by the connection of HTA of the slave to $\overline{TA}$ of the master. Because the DMA priority is low, it does not interfere with the core's external accesses. To save wait states on the external bus, the first DMA is triggered by $\overline{IRQA}$, which is connected to the slave's $\overline{HIRQ}$. The RREQ bit in HCTR is set so that whenever there is data in DRXR, the $\overline{HIRQ}$ pin is pulsed. In this way this DMA channel can access the bus only when it really has data.

```
movep     #$effc11,x:M_AAR0              ; accesses to slave DSP are to
                                         ; addresses X-$effxxx as SRAM
movep     #$000002,x:M_BCR               ; 2 wait states on the access
rep       #12                            ; this period is needed until
nop                                      ; slave finishes the self
movep     #-mas_size,x:M_DOR0            ; channel 4 is used to read HRXS data
movep     #$eff2af,x:M_DSR4              ; the address of HRXS is composed
                                         ; of: 12 bits from master AAR0,
                                         ; 1 don't care, 8 bits from
                                         ; slave's CBMA + '111'
movep     #mas_buf1,x:M_DDR4             ; DCOH = $fff, DCOL = mas_size
movep     #($fff000+mas_size),x:M_DCO4
move      #$eff2ac,r0                    ; the address of HCTR of the slave
movep     #mas_buf2,x:M_DSR5             ; HTXR address is the same as HRXS
movep     #$eff2af,x:M_DDR5
movep     #($fff000+mas_size),x:M_DCO5   ; RREQ=1 in HCTR of the slave
bset      #2,x:(r0)                      ; write RREQ=1 on the slave
movep     #$d80201,x:M_DCR5              ; software triggered block
                                         ; interrupt at the end of block
                                         ; priority level 0
                                         ; y 2D with DOR0 -> x no update
movep     #$c80044,x:M_DCR4              ; word transfer triggered by IRQA
                                         ; interrupt at end_of_block
                                         ; priority level 0
                                         ; x no update -> y 2D with DOR0
```

6.3 HI32 IN PCI BUS MODE

The following sections describe the use of DMA transfers with the HI32 configured in PCI Bus mode.

6.3.1 HI32 DSP Side Registers in PCI Bus Mode

When the HI32 is configured as PCI mode, the DSP side registers are:

- **DSP Control Register (DCTR)**—Bits 22–20 of the DCTR are the HI32 Mode bits (HM[2:0]). If HM[2:0] = 001, then the mode is PCI. If HM[2:0] = 101, then the mode is Self-configuration mode. The Self-configuration mode is a special mode in which the slave HI32 can program its Host-side Registers in order to start working on the system bus. Bits 19–13 are ignored in PCI mode. Bits 6–0 controls the interrupts in the DSP and Host sides and flags to the Host side. These bits do not have to be changed from their state after reset if the HI32 is controlled by the DMA controller.
- **DSP PCI Control Register (DPCR)**—The DPCR controls the HI32 PCI interrupts and interface logic.
- **DSP PCI Master Control Register (DPMC)**—The DPMC generates the two most significant bytes of the 32-bit PCI transaction address and controls the burst length and data transfer format.
- **DSP PCI Address Register (DPAR)**—The DPAR generates the two least significant bytes of the 32-bit PCI transaction address, the PCI bus command, and the PCI bus byte enables.
- **DSP Status Register (DSR)—**The DSR contains some status bits reflecting bits of registers HCTR and HCVR in the host side. Bit 1 in this register is Slave Transmit Data Request (STRQ). It is set if the slave transmit data FIFO is not full. Bit 2 in this register is Slave Receive Data Request (SRRQ). It is set if the there is data in the receive FIFO that was read by the HI32 slave.
- **DSP PCI Status Register (DPSR)**—The DPSR contains status bits and flags that the DSP56300 core can examine when the HI32 is in PCI mode. Bit 1 in this register is Master Transmit Data Request (MTRQ). It is set if the master transmit data FIFO is not full. Bit 2 in this register is Master Receive Data Request (MRRQ). It is set if there is data in the receive FIFO that was read by the HI32 master.
- **DSP Master Transmit Data FIFO (DTXM)**—The DTXM is the data register that can be written by the core or DMA for transferring data from the DSP to the Host side register HRXM. Between DTXM and HRXM there is a 4-level or 8-level FIFO

(depending whether all the 32 bits of PCI data are written to the FIFO). In the Pre-fetch mode (i.e., the bit SFT = 0 in the host side register HCTR) the FIFO is used. In the Fetch mode (i.e., SFT = 1), the FIFO is bypassed and DTXS is connected (after synchronization) directly to HRXS.

- **DSP Slave Transmit Data (DTXS) Register**—The DTXS is the data register that can be written by the core or DMA for transferring data from the DSP to the host side register HRXS. Between DTXS and HRXS there is a 3-level or 6-level FIFO (depending on whether all the 32 bits of the PCI data are written into the FIFO). In the Pre-fetch mode (i.e., bit SFT = 0 in the host side HCTR) the FIFO is used. In the Fetch mode (i.e., SFT = 1) the FIFO is bypassed and DTXS is connected directly (after synchronization) to HRXS.
- **DSP Receive Data Register (DRXR)**—The DRXR is the data register that can be read by the core or DMA for transferring data from the host side HTXR into the DSP. Between HTXR and DRXR there is a 6-level FIFO.

6.3.2 HI32 DMA Operation in PCI Mode

In the DSP56301, if the HI32 is connected in PCI mode, up to three DMA channels can be used to interact with it. The first DMA channel can be programmed to be triggered by STRQ = 1 in the DSR (i.e., DRS = 11110 in the channel's control register) to transfer data to DTXS FIFO. A second DMA channel can be programmed to be triggered by MTRQ=1 in DPSR (i.e., DRS = 11111) to transfer data to DTXM. A third DMA channel can be programmed to be triggered by SRRQ = 1 in DSR (i.e., DRS = 11100) *or* by MRRQ = 1 in DPSR (i.e., DRS = 11101) to transfer data from DRXR FIFO. Because only three FIFOs exist in the HI32, mixed data can be valid in the receive FIFO both for master receive and slave receive. Only one type can be handled by the DMA and the other one must be handled by the core interrupts or polling.

General guidelines for this type of operation include:

- The DMA channels that are triggered by the HI32 must be programmed to work in Word Transfer mode (i.e., TM = 001 or TM = 101 in the channel's control register).
- The channel triggered by SRRQ or MRRQ must be programmed with source address as No Update mode and the DSR to point on DRXR. The destination address mode can be any (No Update, Linear, 2D, or 3D).
- The channel triggered by STRQ must be programmed with destination address as No Update mode and the DDR to point on DTXS.

- The channel triggered by MTRQ must be programmed with destination address as No Update mode and the DDR to point to DTXM. The source address mode for both channels can be any (No Update, Linear, 2D, or 3D).

Note: Coherency of data between the DSP side and the host side can not be guaranteed if all these items are not met.

Because of the handshake between the DMA and the HI32, which is a fast peripheral, the user can program the HI32 before the DMA or the DMA before the HI32 and correct operation is guaranteed. To allocate a DMA channel that was connected to HI32 for another task, clear the DE of the channel and then poll the appropriate DTD bit in the DSTR to assure that the channel was actually disabled. This process can take some time because if a trigger from the HI32 was already accepted by the channel, the channel does not set DTD until the word correspond to this trigger can actually be transferred. To put the chip in the Stop state, follow the rules in **Section 5.8** on page 5-8.

6.3.3 DMA Code Example—HI32 in PCI Bus Mode

In the following example, two DSP56301 chips are connected to the PCI bus as two agents. The PCI bus frequency is 33 MHz and the internal frequency of each DSP56301 must be at least 5/3 of this frequency. The same code can be written to both devices. In the example, DMA channel 0 transfers a buffer from internal memory to DTXM, channel 1 transfers the slave received data from DRXR to internal memory, and channel 2 transfers a buffer from internal memory to DTXS. Master received data in DRXR is handled by interrupts. All the circular buffers in the internal memory are of equal size. For the slave receive and slave transmit transactions, the only requirement is initializing the DMA channels. For the master transmit transaction, the address phase is handled by interrupt I_HPMA. For the master receive transaction, all handling is through interrupts I_HPMA and I_HPMR. All the buffers are transferred repeatedly. Of course, in a true system, some method of semaphore or scheduling should be used to decide when the master actually wants to transmit or receive data.

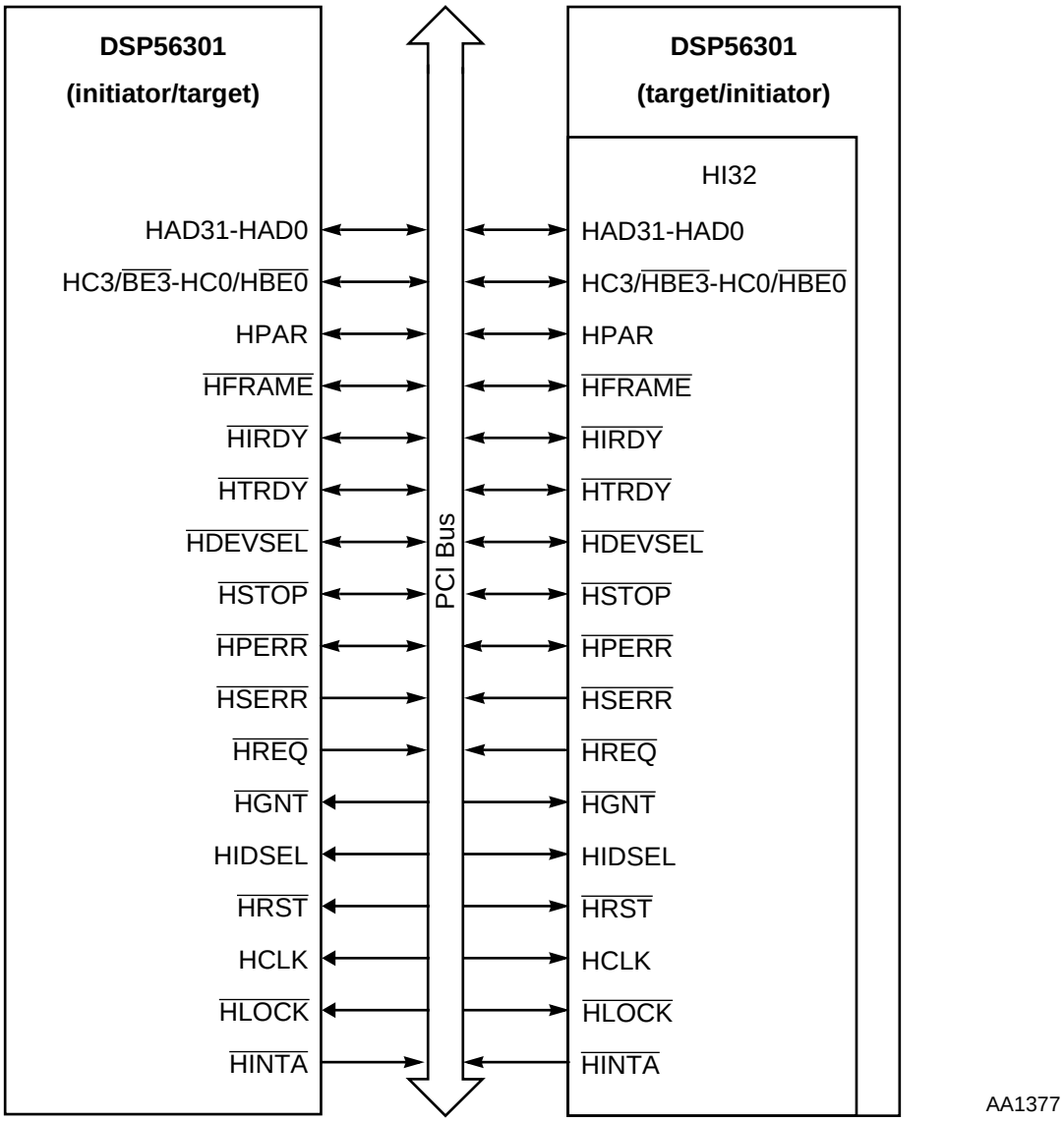


Figure 6-2 Connection of DSP56301 to PCI Bus

```
M_DCTR          EQU             $ffffc5    ; DSP56301 HI32 DSP control register
M_DPCR          EQU             $ffffc6    ; DSP56301 HI32 DSP PCI control register
        M_DPMC  EQU             $ffffc7    ; DSP56301 HI32 DSP PCI master control
                                           ; register
        M_DPAR  EQU             $ffffc8    ; DSP56301 HI32 DSP PCI address register
        M_DPSR  EQU             $ffffca    ; DSP56301 HI32 DSP PCI status register
        M_DRXR  EQU             $ffffcb    ; DSP56301 HI32 DSP receive FIFO
        M_DTXM  EQU             $ffffcc    ; DSP56301 HI32 DSP master transmit FIFO
        M_DTXS  EQU             $ffffcd    ; DSP56301 HI32 DSP slave transmit FIFO

        org     p:I_RESET
        jmp     >START
        org     p:I_HPMR                   : this is the interrupt Host PCI
                                           ; master receive
        movep   x:M_DRXR,x:(r6)+
        org     p:I_HPMA                   ; this is the interrupt Host PCI
                                           ; master address
        jsr     >HPMA_

        org     p:START
        move    #$0,sr                     ; enable interrupts
        bset    #1,x:M_IPRP                ; HI32's IPL=1
        movep   #$500000,x:M_DCTR          ; enter the self configuration mode
                                           ; by writing HM[2:0]=101
        movep   #base_addr,x:M_DPMC        ; this is the base address to be
                                           ; transferred to CBMA
        movep   #$000006,x:M_DPAR          ; write to CSTR/CCMR BM=1,MSE=1
        movep   #$0,x:M_DPAR               ; dummy write to CCCR/CRID
        movep   #$00f800,x:M_DPAR          ; write to CLAT
        movep   #$0,x:M_DPAR               ; write data from DPMC to CBMA
        movep   #$0,x:M_DPMC               ; return to personal reset state HM=000
        movep   #$040014,x:M_DPCR          ; MACE=1, MAIE=1, MRIE=1
        movep   #$100000,x:M_DCTR          ; PCI mode (HM=001)

        move    #buf_size,m6               ; initialization of the circular buffer
        move    #int_buf_mr,r6             ; for master receive (handled by I_HPMR)
                                           ; address int_buf_mr should be with
        move    #buf_size,r7               ; leading log2(buf_size-1) zeros
        movep   #-buf_size,x:M_DOR3
        movep   #int_buf_mt,x:M_DSR0
        movep   #M_DTXM,x:M_DDR0
        movep   r7,x:DCO0
        movep   #$aefa30,x:M_DCR0          ; DCOH=0, DCOL= buf_size-1
                                           ; word transfer triggered by MTRQ
                                           ; no interrupt at end of block
                                           ; priority level 3
                                           ; x 2D with DOR3 -> x no update
        movep   #M_DRXR,x:M_DSR1
        movep   #int_buf_sr,x:M_DDR1
        movep   r7,x:M_DCO1                ; same as DCO0
```

```
            movep       $aee1c4,x:M_DCR1            ; word transfer triggered by SRRQ
                                                    ; no interrupt at end of block
                                                    ; priority level 3
                                                    ; x no update -> y 2D with DOR3
            movep       #int_buf_st,x:M_DSR2
            movep       #M_DTXS,x:M_DDR2
            movep       r7,x:M_DCO2                 ; same as DCO1
            movep       #$aef231,x:M_DCR2           ; word transfer triggered by STRQ
                                                    ; no interrupt at ens of block
                                                    ; priority level 3
                                                    ; y 2D with DOR3 -> x no update
HPMA_
            jset        #1,x:M_DPSR,PCI_RD          ; if MTRQ=1 the master transmit FIFO
                                                    ; is empty and therefore master
                                                    ; transmit transaction should not
                                                    ; occur. In this case because I_HPMA
                                                    ; was generated, it must be that the
                                                    ; master receive FIFO is not full and
                                                    ; PCI master receive transaction can
                                                    ; occur
            movep       #$(3f0000+s_ad_h),x:M_DPMC  ; FC=0, BL=$3f+1 = 64 in write
                                                    ; trans.
                                                    ; with PCI slave address s_ad_h
            movep       #$(070000+s_ad_l),x:M_DPAR  ; BE_=0000, C=0111 (memory wr),
                                                    ; addr=slave address low register
            rti
PCI_RD
            movep       #$(2f0000+s_ad_h),x:M_DPMC  ; FC=0, BL=$2f+1 = 48 in rd.
                                                    ; trans.
                                                    ; with PCI slave address s_ad_h
            movep       #$(060000+s_ad_l),x:M_DPAR  ; BE_=0000, C=0110 (memory rd),
            rti                                     ; addr=slave address low register
```

How to reach us:

USA/EUROPE/LOCATIONS NOT LISTED:
Motorola Literature Distribution
P.O. Box 5405
Denver, Colorado 80217
(303) 675-2140
1-800-441-2447

Mfax™:
RMFAX0@email.sps.mot.com
TOUCHTONE (602) 244-6609

JAPAN:
Nippon Motorola Ltd.
SPD, Strategic Planning Office
4-32-1, Nishi-Gotanda
Shinagawa-ku, Tokyo 141, Japan
81-3-5487-8488

TECHNICAL RESOURCE CENTER:
1 (800) 521-6274

ASIA/PACIFIC:
Motorola Semiconductors H.K. Ltd.
8B Tai Ping Industrial Park
51 Ting Kok Road
Tai Po, N.T. Hong Kong. 852-26629298

INTERNET: http://www.motorola-dsp.com

DSP HELPLINE: dsphelp@dsp.sps.mot.com

APR23/D