



# Macrocad Development Inc. VCM 6404 Scatter/Gather DMA Core Model

**Description:**

Macrocad's VCM 6404 model is a synthesizable general purpose DMA Controller. Synchronous design and small module size assures worry free synthesis. Well commented code provides insight into operations. A test bench is included. DMA transfers can be "throttled" to provide balanced access. DMA Chaining, (Linked lists, or Scatter/Gather) is supported. Transfer errors can be detected and can generate the DMA completion Interrupt. The DMA controller can be set to program itself for the next link in the list, including the first link. An interrupt is generated in the case of an aborted DMA transfer, and the link is also aborted. Designed to work with the VCM 4601 PCI core from Macrocad.

**Features:**

- ◆ PCI 2.2 Compliant
- ◆ Synchronous design
- ◆ Synthesizable RTL code
- ◆ 66MHz operations
- ◆ 4 DMA channels
- ◆ Approximately 8500 ASIC gates
- ◆ Verilog or VHDL source code
- ◆ Functional partitioning
- ◆ Designed *by* hardware engineers *for* hardware engineers
- ◆ Internal buffers implemented with SRAM
- ◆ Expandable register set
- ◆ Support for linked lists and scatter gather
- ◆ Byte boundary addressing for both source and destination addresses
- ◆ Address hold feature
- ◆ Bursts are supported
- ◆ DMA pacing option for system bandwidth control
- ◆ DMA retry support

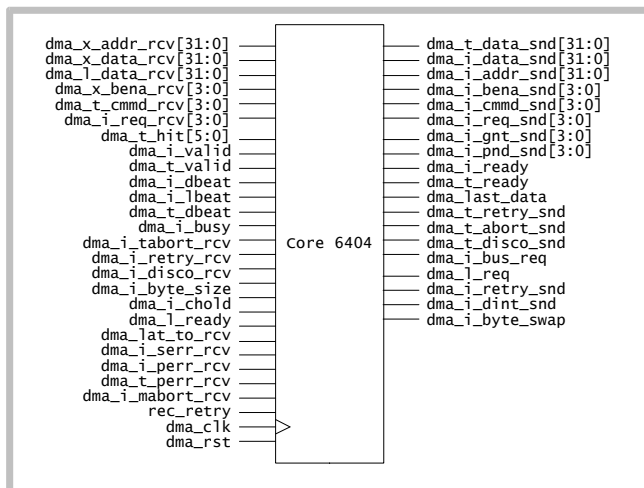


Figure 1

## DMA Core Description

Macrocad's VCM 6404 is a synthesizable DMA controller behavior model. This model provides the system designer with a drop in DMA function. This model is functionally partitioned to provide the most flexibility for ASIC and FPGA implementations.

The DMA interface works with a request / acknowledge handshake. Thus, the acceptance of a request does not indicate that the transaction completed successfully. There is another indication that the transaction was successful. There are separate target and initiator state machines as well as separate handshake signals. Data paths are not always different for target and initiator transactions.

## DMA Signal Descriptions

The DMA interface is comprised of generic signals which can be connected to the user's application interface logic. These signals are shown below. The signals can be used for a local or side bus, the main DMA transfer bus, or signals can be considered global in that they apply to all the DMA logic.

### DMA Bus Signal Group

Name	Range	Type	Bus	Description
dma_t_data_snd	[31:0]	Output	M	Target read data to main bus
dma_i_data_snd	[31:0]	Output	ML	Initiator write data to bus
dma_i_addr_snd	[31:0]	Output	ML	Initiator Address to bus
dma_i_bena_snd	[3:0]	Output	ML	Initiator Byte enables to bus
dma_i_cmmnd_snd	[3:0]	Output	ML	Initiator command
dma_i_req_snd	[3:0]	Output	Global	DMA channel request
dma_i_gnt_snd	[3:0]	Output	Global	DMA channel grant
dma_i_pnd_snd	[3:0]	Output	Global	DMA channel pending
dma_i_ready		Output	M	Initiator is ready
dma_t_ready		Output	M	Target register access is executed
dma_i_last_data		Output	ML	last beat request
dma_t_retry_snd		Output	M	Target Signal a retry
dma_t_abort_snd		Output	M	Target signal an abort
dma_t_disco_snd		Output	M	Target signal a disconnect
dma_i_bus_req		Output	M	Request main bus for DMA transfer
dma_l_req		Output	L	Request local bus for DMA transfer
dma_i_retry_snd		Output	M	Initiator Request a retry of previous access
dma_i_dint_snd		Output	M	DMA interrupt
dma_i_byte_swap		Output	Global	Swap byte lanes to Big-Endian
dma_x_addr_rcv	[31:0]	Input	M	Initiator address from bus
dma_x_data_rcv	[31:0]	Input	M	Initiator data from main bus
dma_l_data_rcv	[31:0]	Input	L	Initiator data from local bus
dma_x_bena_rcv	[3:0]	Input	M	Initiator byte enables from bus
dma_t_cmmnd_rcv	[3:0]	Input	M	Target command
dma_i_req_rcv	[3:0]	Input	Global	DMA channel request
dma_t_hit	[5:0]	Input	M	Target hit, 6 ranges supported
dma_i_valid		Input	ML	Initiator data/address/byte enables valid
dma_t_valid		Input	M	Target <write> data is valid
dma_i_dbeat		Input	ML	Initiator data beat

Name	Range	Type	Bus	Description
dma_i_lbeat		Input	ML	Initiator last data beat of burst
dma_i_busy		Input	ML	Initiator busy, DMA transfer in progress
dma_t_dbeat		Input	M	Target data beat
dma_i_tabort_rcv		Input	M	Received DMA transfer abort
dma_i_retry_rcv		Input	M	Retry received from interface
dma_i_disco_rcv		Input	M	Disconnect received from interface
dma_i_byte_size		Input	Global	Size for address hold function is 8 bits
dma_i_chold_rcv		Input	M	Hold the DMA channel, stall arbitration
dma_l_ready		Input	L	Local read data valid
dma_i_lat_to_rcv		Input	M	Latency timeout, relinquish bus
dma_i_serr_rcv		Input	M	Initiator DMA address parity (serious) error
dma_i_perr_rcv		Input	M	Initiator DMA transfer parity error
dma_t_perr_rcv		Input	M	Target DMA transfer parity error
dma_i_mabort_rcv		Input	M	Initiator DMA transfer abort
dma_clk		Input	Global	Clock
dma_rst		Input	Global	Reset

Table 1

## Pin Connections

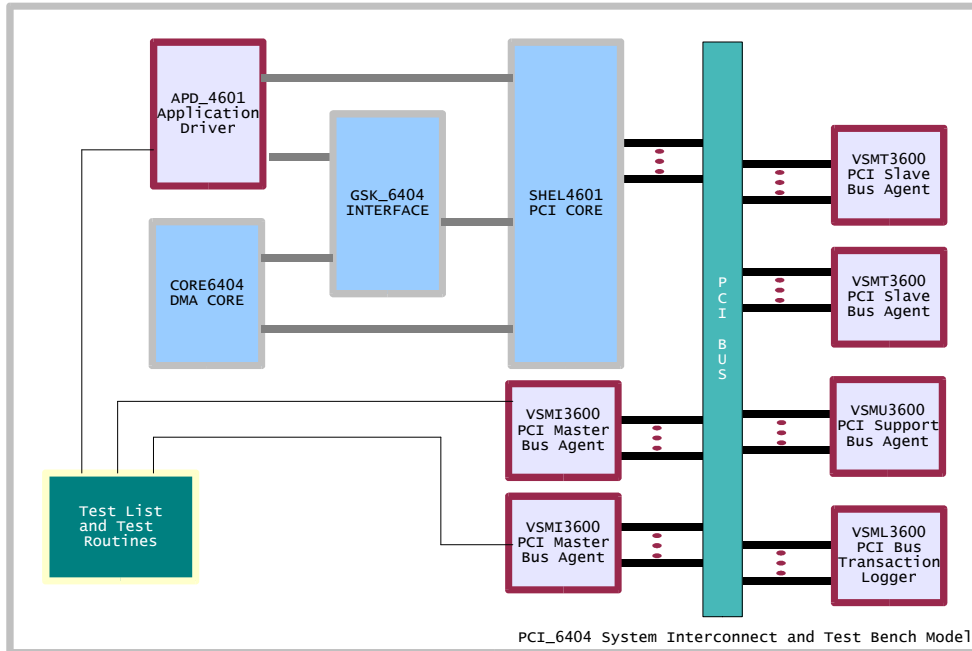
The VCM6404 DMA core supports a main bus, sometimes referred to as the memory bus, and a local bus, or peripheral bus. This enables the DMA to transfer data to and from slow peripherals connected to the local bus, while not slowing down the memory or main bus.

There are separate control handshakes for the local bus. These are `dma_i_req`, local request, and `dma_i_ready`, local ready. The data path for the local bus uses `dma_i_data_rcv`, local data read, when the read phase of the DMA transfer is active. Other signals for DMA transfers are shared between the local bus and main bus. Other signals such as the DMA channel requests and grants can be connected as desired in the system. In some cases, some main bus signals might be used, or shared with the local bus, such as `dma_i_tabort_rcv`, abort the DMA transfer, `dma_i_dint_snd`, DMA interrupt, etc. Programming operations are supported only through the main bus. No access to the register set is available via the local bus.

The basic handshake consists of a four deep pipeline if bursting is supported. The DMA will make a request, and look for a valid signal return. This valid, (`dma_i_valid`) indicates that the device connected to the DMA has accepted this request, (data, address and byte enables), and that the DMA core can make another request if needed. The request information must remain on the bus until a valid signal is detected. The `dma_i_last_data` signal indicates the last request in a burst. The requests do not indicate that the transaction has been executed, just that the request has been made. The `dma_i_dbeat` signal will indicate the execution of a beat of a DMA burst transfer. Thus the DMA always looks for the `dma_i_dbeat` signal to know if the transaction is executed. This is important in the read phase of the transfer. The DMA core needs to know when the read data is valid, and which byte lanes to write this read data into. Thus the return of the `dma_i_dbeat` signal indicates valid read data, address and byte enable information to the DMA core.

## PCI Example

An example of a PCI implementation using the DMA core is shown below.



The PCI application and the DMA core share the PCI application interface. The signals interface between the PCI core and the application and DMA through the gasket interface. This is a simple multiplexor to switch between the application and the DMA core for signals which are inputs to the application side of the PCI core. The PCI core application side outputs are broadcast to both the application and the DMA core.

## DMA Functional Description

The VCM\_6404 DMA controller will automatically transfer bytes from the source to the destination. The current address for the source is determined by the Src register. Likewise, the current address for the destination is determined by the Dst register. The number of bytes to be transferred is indicated in the Btc register. Each channel has one of these 16 bit registers. The byte enables are calculated based on the address, (source or destination), and the number of bytes left to be transferred. Buffered, (burst), transfers are supported, but fly-by transfers are not supported. The DMA controller can operate in chaining mode. In this case when the DMA reaches the end of the current link's transfer the byte counter (Btc) will be 0x0000. In this case, if chaining is enabled, and the end of the chain has not yet been reached, the DMA will read the next link's record into the Btc, Src, Dst, Nxt, and Ctl registers for the active channel. Scatter/Gather operations and repeat buffer (auto-initialize) operations are easily supported with this option.

The DMA can execute transfers between two memory bus mounted devices, or between a local bus device and a memory bus device, or between two local bus devices. The byte alignments work regardless of the source or destination being on the main bus or local bus. All devices are considered to be 32 bit devices to this DMA controller. If the local bus mounted device is 16 or 8 bits, a translation will need to be made external to the DMA, such that the accesses happen properly.

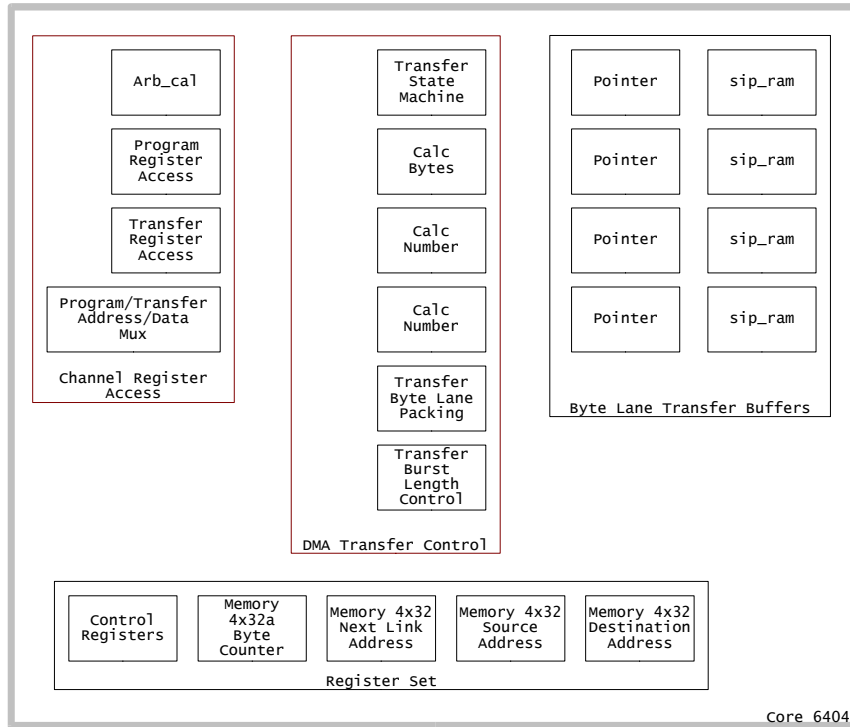


Figure 2

### Byte Alignment on Transfers

The alignment of data during transfers is flexible, and efficient. The incoming data (from the source) is aligned before storing it in the temporary buffer. The buffer is made up of 4 byte wide single port rams. These are individually accessed on a byte lane basis for both read and write phases of the transfer. The buffer control logic will write the byte lanes indicated by the byte enables, but will 'pack' the data so that the next byte will be adjacent to the last one written. In this way, the buffer size does not need to be larger than the burst limit. Data alignment is required for DMA transfers. This DMA controller aligns and packs on a burst read of the source, then unpacks and re-aligns for the write to the destination. Any combination of alignment of source and destination data is possible with this method, and full burst performance is maintained. However, links containing program information should be on 64 byte boundaries. No alignment is done for reading program links.

### Register Set

Each channel has Source address, Destination address, Byte count, Next record location, and Control registers to determine the operation of the transfer. There are 128 address byte locations reserved for these registers, (0x00 to 0x7F).

Name:	SIZE:	ADDR: - 1e	DESCRIPTION:	@RESET
Btc_0	32	0x00000000	Byte count for channel 0	0x00000000
Src_0	32	0x00000010	Source Address for channel 0	0x00000000
Dst_0	32	0x00000020	Destination Address for channel 0	0x00000000
Nxt_0	32	0x00000030	Next Link's Address for channel 0	0x00000000
Ctl_0	32	0x00000040	Control and Status Register for channel 0	0x00000000
Btc_1	32	0x00000004	Byte count for channel 1	0x00000000
Src_1	32	0x00000014	Source Address for channel 1	0x00000000
Dst_1	32	0x00000024	Destination Address for channel 1	0x00000000
Nxt_1	32	0x00000034	Next Link's Address for channel 1	0x00000000
Ctl_1	32	0x00000044	Control Register for channel 1	0x00000000
Btc_2	32	0x00000008	Byte count for channel 2	0x00000000
Src_2	32	0x00000018	Source Address for channel 2	0x00000000
Dst_2	32	0x00000028	Destination Address for channel 2	0x00000000
Nxt_2	32	0x00000038	Next Link's Address for channel 2	0x00000000
Ctl_2	32	0x00000048	Control Register for channel 2	0x00000000
Btc_3	32	0x0000000c	Byte count for channel 3	0x00000000
Src_3	32	0x0000001c	Source Address for channel 3	0x00000000
Dst_3	32	0x0000002c	Destination Address for channel 3	0x00000000
Nxt_3	32	0x0000003c	Next Link's Address for channel 3	0x00000000
Ctl_3	32	0x0000004c	Control Register for channel 3	0x00000000
Int	32	0x00000050	Interrupt mask and status register	0x00000000
Arb	32	0x00000060	Arbitration between channels	0x00000000

Table 2a

Name:	SIZE:	ADDR: - 1e	DESCRIPTION:	@RESET
Btc_0	32	0x00000000	Byte count for channel 0	0x00000000
Btc_1	32	0x00000004	Byte count for channel 1	0x00000000
Btc_2	32	0x00000008	Byte count for channel 2	0x00000000
Btc_3	32	0x0000000c	Byte count for channel 3	0x00000000
Src_0	32	0x00000010	Source Address for channel 0	0x00000000
Src_1	32	0x00000014	Source Address for channel 1	0x00000000
Src_2	32	0x00000018	Source Address for channel 2	0x00000000
Src_3	32	0x0000001c	Source Address for channel 3	0x00000000
Dst_0	32	0x00000020	Destination Address for channel 0	0x00000000
Dst_1	32	0x00000024	Destination Address for channel 1	0x00000000
Dst_2	32	0x00000028	Destination Address for channel 2	0x00000000
Dst_3	32	0x0000002c	Destination Address for channel 3	0x00000000
Nxt_0	32	0x00000030	Next Link's Address for channel 0	0x00000000
Nxt_1	32	0x00000034	Next Link's Address for channel 1	0x00000000
Nxt_2	32	0x00000038	Next Link's Address for channel 2	0x00000000
Nxt_3	32	0x0000003c	Next Link's Address for channel 3	0x00000000
Ctl_0	32	0x00000040	Control and Status Register for channel 0	0x00000000
Ctl_1	32	0x00000044	Control Register for channel 1	0x00000000
Ctl_2	32	0x00000048	Control Register for channel 2	0x00000000
Ctl_3	32	0x0000004c	Control Register for channel 3	0x00000000
Int	32	0x00000050	Interrupt mask and status register	0x00000000
Arb	32	0x00000060	Arbitration between channels	0x00000000

**Table 2b**

**Register Access**

Tables 2a and 2b assume that the register set is accessed from 0x00000000 to 0x00000063. The registers use a select feature so they can be physically aliased to other address ranges. Registers are always accessed as a 32 bit data width. A write to a register in the register set is accessed by activating the dma\_t\_hit[2] signal along with dma\_t\_dbeat and the proper write command code. A read access is accomplished by activating the dma\_t\_hit[2] signal along with the proper read command code. Five address bits are used to select the registers, dma\_x\_addr\_rcv[6:2]. Programming register access and DMA transfers are mutually exclusive, and access to the registers should be arbitrated external to the DMA core.



**Figure 3**

Figure 3 shows the address bits used for register access. The selection of the range for the register set is accomplished with the dma\_t\_hit[2] input. Thus the register set address range can be placed anywhere within the total address range of the system. The address range required is 128 bytes, (7 address bits).

## Register Bit Descriptions

Name:	OP:	BIT:	DESCRIPTION:
Btc_x	R/W	31:0	Transfer count register:
	R/W	31:16	RESERVED
	R/W	15:0	This register provides the transfer count (in bytes) for the channel's DMA transfers. The transfer terminates when the value in the byte counter equals 0x0000. If this register (which is also the byte counter) is programmed with the value 0x0001, then one byte will be transferred. Thus the maximum number of bytes which can be transferred is 65535.
Src_x	R/W	31:0	Source address register: Source (read location) address counter register for DMA transfer operations. The address assumes little endian byte boundary referencing. The source address counter increments, decrements, or is static (depending on the control register program value) when the transfers occur.
Dst_x	R/W	31:0	Destination address register: Destination (write location) address counter register for DMA transfer operations. The address assumes little endian byte boundary referencing. The destination address counter increments, decrements, or is static (depending on the control register program value) when the transfers occur.
Nxt_x	R/W	31:0	Next pointer address register: This register holds the location for the next set of register values (link) in the chain of operations. If chaining is enabled, when the current link's transfer is complete, the DMA controller will read in the data from the location indicated in this buffer. The Src, Dst, Btc, Nxt and Ctl registers for the active channel will be updated with these values. If the control register <Ctl> bit 9 is set to a '1', this indicates that this is the final link in the chain, and no next link will be executed at the end of this link.

---

Table 3a

Name:	OP:	BIT:	DESCRIPTION:
Ctl_x	-	31:0	Control register:
	RO	31:22	RESERVED
	R/W	21	CycRetry: Cycle retry, disconnect, or timeout warning
	R/W	20	CycAbort: Cycle abort detected during DMA transfer
	R/W	19:18	DstSel: Destination select 3 = RESERVED 2 = Local bus 1 = IO 0 = memory
	R/W	17:16	SrcSel: Source select 3 = RESERVED 2 = Local bus 1 = IO 0 = memory
	RO	15	RESERVED
	RO	14	DmaActive: DMA active, or pending
	R/W	13	FetNexRec: Fetch Next Record This bit (if set to 1) will force the DMA controller to fetch the next transfer record in the chain. It is used to start a link process.
	R/W	12	ChanEn: DMA channel enable 1 = DMA channel enabled 0 = DMA channel disabled
	R/W	11	TransMod: Transfer Mode 1 = Block transfer mode, for memory to memory transfers 0 = Demand transfer mode, for DMA devices capable of requesting service
	R/W	10	IntMode: Interrupt Mode 1 = Interrupt when all links in chain have completed 0 = Interrupt on each chain link completion
	R/W	9	ChainMod: Chaining mode 1 = Chaining disabled 0 = Chaining enabled
	R/W	8:6	DatTransLim: Data transfer limit code 7 = 32 bytes 6 = 2 bytes 5 = 1 byte 4 = Reserved 3 = 16 bytes 2 = Reserved 1 = 8 bytes 0 = 4 bytes
	R/W	5:4	DestDir: Destination address register update mode 3 = Reserved 2 = Hold 1 = Reserved 0 = Increment
	R/W	3:2	SrcDir: Destination address register update mode 3 = Reserved 2 = Hold 1 = Reserved 0 = Increment
-	1:0	Reserved	

Table 3b

## Control Register Bit Descriptions

	Control register bit descriptions
CycRetry	PCI retry, timeout or disconnect detected.
CycAbort	PCI master or target abort detected.
DstSel	The data destination selection bits determine the destination of the data transfer. This can be either the PCI or local bus, and can be either IO or memory for either destination.
SrcSel	The data destination selection bits determine the destination of the data transfer. This can be either the PCI or local bus, and can be either IO or memory for either destination.
DmaActive	This bit indicates the the DMA channel is active, or pending. This means that is the DMA byte counter is not 0x0000, or if there is a link pending, (not at end of chain), this bit will be set.
FetNexRec	Setting this bit will cause the DMA to read the link descriptor is the channel is enabled, and the byte count is 0x0000.
ChanEn	This bit enables the DMA channel. It can be written at any time during a transfer.
TransMod	Selecting block mode operation will ignore the DMA request and DMA grant signals. Demand mode will use the DMA request and grant signals to pace, or throttle the DMA transfers. These signals are used by the DMA arbitration logic.
IntMode	Interrupts can be generated either at the end of the link, or at the end of the chain. This bit selects the interrupt behavior.
ChainMod	This bit enables chaining mode. The DMA will not fetch the next link descriptor unless chining mode is enabled. This bit is cleared for the last link descriptor in order to end the chain of links.
DatTransLim	These bits determine the PCI burst limit. This is useful in tuning system performance. Note that this limit can be programmed and changed on a per link basis. The upper limit is determined by the size of the PCI transfer buffer size. This default is 32 bytes.
DestDir	The destination address register can be controlled to increment, decrement, or be static. Bits 5:4 determine the destination address generation behavior.
SrcDir	The source address register can be controlled to increment, decrement, or be static. Bits 3:2 determine the source address generation behavior.

Name:	OP:	BIT:	DESCRIPTION:
Arb	R/W	7:0	Arb: Arbitration control register for all channels
		7	Reserved
		6	PrioOpt: Priority Option 1 = High priority will request transfers as long as it has data to transfer. 0 = Upon completion of transfer, a high priority device will suppress request for one grant to other device.
		5:4	PrioGrps: Priority Groups 3 = Reserved 2 = Channel 1/0 Priority 1 = Channel 3/2 priority 0 = Round Robin
		3:2	PrioChan3_2: Priority between channels 3 and 2 3 = Reserved 2 = Channel 2 priority 1 = Channel 3 Priority 0 = Round Robin
		1:0	PrioChan1_0: Priority between channels 1 and 0 3 = Reserved 2 = Channel 0 priority 1 = Channel 1 Priority 0 = Round Robin
Int	R/W	31:0	Int: Arbitration control register for all channels
		31:8	RESERVED
		7:4	Interrupt mask These 4 bits will enable the associated interrupt.
		7	enable interrupt channel 3
		6	enable interrupt channel 2
		5	enable interrupt channel 1
		4	enable interrupt channel 0
		3:0	Interrupt status and clear
			When read, indicate active interrupt channels
		3	if = 1, interrupt active for channel 3
		2	if = 1, interrupt active for channel 2
		1	if = 1, interrupt active for channel 1
		0	if = 1, interrupt active for channel 0
			When written, clear the selected channel's interrupt
		3	if = 1, clear interrupt for channel 3
		2	if = 1, clear interrupt for channel 2
		1	if = 1, clear interrupt for channel 1
		0	if = 1, clear interrupt for channel 0

Table 3c

## DMA Transfer Handshake

The DMA controller acts like a target, (slave) or as an initiator, (master). In the target mode, the register set can be accessed. In the initiator mode, the DMA can either transfer data, or it can load new transfer information from memory buffers which have been set up prior to the DMA being activated. In order to accomplish these initiator transfers, the DMA core asserts the dma\_i\_bus\_req signal at the start of a transfer. Once it detects that dma\_i\_busy has been asserted, it drops the dma\_i\_bus\_req. The dma\_i\_bus\_req signal is used to start the transfer, but the transfer will continue bursting until the end of burst is reached, at which point the DMA core will assert the last signal. Advancing through the burst to the next address and data set, is accomplished by the device connected to the DMA core. It advanced to the next word of the burst by asserting dma\_i\_valid. It indicates that a transaction is complete by asserting dma\_i\_dbeat. The last completed beat of the burst is indicated by detection of dma\_i\_lbeat. The DMA core will execute read, (i.e. write data into the transfer buffer) when it is doing a read phase of the transfer, and it detects an dma\_i\_dbeat. It will then look at the byte enables coming in, (dma\_x\_bena\_rcv), then pack the incoming data on dma\_x\_data\_rcv, and write it to the DMA transfer temporary buffer. When a write phase of the DMA transfer is in progress, the DMA core

assumes the data is written when the `dma_i_valid` signal is asserted. It does not update the byte counter until the `dma_i_dbeat` signal indicates a completed cycle.

## Programming Considerations

The sequence of programming and executing a series of DMA transfers can happen in two ways. The programming agent can program the registers directly, or can write the program information to a location in memory called a link descriptor table. The programming agent can then have the DMA controller program itself, by writing the next address register. This will point the DMA controller to the location containing the link descriptor table. The programming agent then writes to the control register, setting the fetch next record bit active. This starts the DMA which will fetch the link descriptor table which should contain the dma transfer register values. This operation will write the DMA transfer registers, (`byt`, `src`, `dst`, `nxt`, and `ctl`) with the values in the link descriptor table. The DMA controller will then request data transfers until the byte counter is decremented to `0x0000` which indicates that there are no more bytes to transfer in this link, for this channel. If there are no other links to execute, no further DMA operations will be executed, since this is defined as the end of a chain. The DMA will also stop requesting transfers if an abort is detected, or if the control register is programmed to disable the channel's DMA transfer prematurely, (setting control register bit 12 to 0).

It is important to remember that the source, destination, byte count, next record, and control registers are all programmed when the next link's programming record is fetched. Thus control over the DMA process is established prior to the DMA process starting.

In short, the end of a chain can be determined in a variety of ways, but it is best to accomplish this by setting the chaining disable bit in the control register field in the last link's program record. The fetch next record bit in this field should be set to 0.

An interrupt can be generated at this time to show that the chain is complete.

If an interrupt is not used, then it is important to be able to determine when the end of the last link has occurred. This can be done in the following way. Do not set the chaining disable bit in the last link's control register program record. Set the fetch next record bit to 0, (disable fetching another program record). When the end of the link is reached, the chaining disable bit will be set by the DMA logic, and this bit can then be monitored to show the end of the chain. Since this bit can be programmed also, this can interfere with this indication. In this case, always set this bit to 0 in the program records, and then it can be used to indicate the end of the link in the last link of the chain.

Also, the byte counter value being equal to 0 indicates the end of a link, so if the chaining disable bit is a 1 (for whatever reason) these two conditions would also indicate the end of the last link in the chain.

If for some reason, the programming agent desires to disable a DMA transfer prior to the end of chain, it should not write the byte count register to `0x00000000`, but rather clear the DMA enable bit, (12) in the control register for that channel.