PCI DMA Unit Programming Interface V2.3

Xin Gao

Department of Electrical Engineering, University of Hawaii at Manoa
xingao@hawaii.edu

Abstract: This document briefly describes the programming interface of the PCI DMA unit. A general introduction is given in section 1. The registers are described in section 2. The mode of linked list descriptors is presented in section 3. Finally, an example is shown in section 4.

1. Introduction

The Block diagram of the PCI DMA unit is shown in Fig.1.

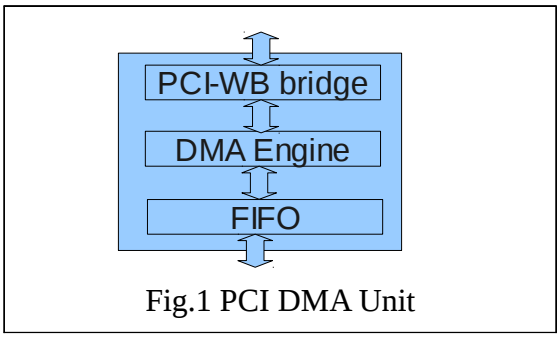

Fig.1 PCI DMA Unit

The PCI-WB bridge is from OPENCORES. Some configuration information of the core is shown below.

1) Two PCI images are implemented. Image0 is for the CONFIGURATION space while image1 is mapped to the MEMORY space. Each of the two implemented images covers 4KB PCI space. No address translation is enabled. The base addresses of the images are assigned by the software.

2) The PCI DMA unit supports both read and write operations.

3) Two WB images are implemented, each of which covers 2G space. So, totally the unit covers all the 4G space.

4) The values of some registers in the PCI configuration header are shown below in Table 1.

NOTE: In our implementation, we assume a little-endian data organization style. That is to say, on the PCI bus side, the most significant byte of the 32-bit data bus is for the higher address while the least significant byte is for the lower address.

Table 1: The Values of Some Configuration Header Registers

Name	Value
Vendor ID	16'h10EE
Device ID	16'h0001
Revision ID	8'h01
Subsys Vector ID	16'h10EE
Subsys ID	16'h0001
Max Lat	8'h1a
Min Gnt	8'h08

2. Registers

Currently, there are only six 32-bit registers in the PCI DMA unit. They are shown below.

1) BOARD ID Register

a) Address: {Image1_base_addr[31:12],12'h000}, where “Image1_base_addr” is the base address of PCI image1.

b) Function: The lower 6 bits of the register contain the ID of the board.

2) **CTRL Register**

a) Address: {Image1_base_addr[31:12],12'h004}, where “Image1_base_addr” is the base address of PCI image1.

b) Function:

CTRL[31]: This bit MUST be zero.

CTRL[30]: DMA start control. If this bit is “1”, the DMA operation starts. When the DMA operation finishes, an interrupt signal will be asserted on the PCI bus. The interruption signal will be held in the assertion state until this bit is cleared (i.e., “0”). Also, to start a new DMA operation, the software MUST first clear this bit before setting this bit to “1”.

CTRL[29]: Watchdog counter enable control. If this bit is “1”, the watchdog counter in the DMA engine will be enabled. Refer to “CTRL[1:0]” for more information.

CTRL[1:0]: Watchdog counter limit control. The watchdog counter determines how many clock cycles the DMA engine is allowed to wait for the data from (or to) the FIFOs (refer to Fig.1). This is specified in Table 2. If, during a DMA transfer, the total number of clock cycles the DMA engine wait for the data from (or to) the FIFOs is larger than the value specified by CTRL[1:0], the DMA transfer will be stopped with the interrupt signal asserted and the total number of transferred bytes returned.

Table 2: CTRL[1:0]

CTRL[1:0]	No. of Clock Cycles
2'b00	4
2'b01	8
2'b10	16
others	4

CTRL[28]: Interrupt enable bit. If this bit is “1”, an interrupt will be asserted on the PCI bus when the DMA transfer finishes.

CTRL[27:25]: DMA mode. The meanings of the different combinations are shown below in Table 3.

Table 3: DMA Mode

CTRL[27:25]	DMA Mode
3'b000	Read channel 0
3'b001	Read channel 1
3'b010	Read channel 2
3'b011	Read channel 3
3'b100	Write channel 0
3'b101	Write channel 1

3'b110	Write channel 2
3'b111	Write channel 3

CTRL[23:16]: Burst size control. In our implementation, the DMA transfer is broken into bursts to reduce the latency. The bits in CTRL[23:16] determine the size of the burst. This is specified below in Table 3. Note that larger burst size does not necessarily mean faster transfer. This is because the PCI write operations are temporarily stored in the FIFOs in the PCI-WB bridge. These write operations are released on the PCI bus only when the burst end is received.

Table 4: CTRL[23:16]

CTRL[23:16]	No. of Bytes in a Burst
8'h00	512
8'h01	256
8'h02	128
others	512

Other bits: All the other bits are not used yet.

c) NOTE: If the DMA engine is in the process of fulfilling a DMA transfer request, writing to the CTRL register is ignored and has NO effect.

3) DMA_POT Register

a) Address: {Image1_base_addr[31:12],12'h008}, where “Image1_base_addr” is the base address of PCI image1.

b) Function: This 32-bit register contains the pointer to the head of the list of the DMA operation descriptors. Note that this pointer is an address on the PCI bus.

c) NOTE: If the DMA engine is in the process of fulfilling a DMA transfer request, writing to the DMA_POT register is ignored and has NO effect.

4) DMA_LEN Register

a) Address: {Image1_base_addr[31:12],12'h00C}, where “Image1_base_addr” is the base address of PCI image1.

b) Function: This 32-bit register specifies the total number of bytes to be sent (or received) in the DMA request.

c) NOTE: If the DMA engine is in the process of fulfilling a DMA transfer request, writing to the DMA_LEN register has NO effect.

5) TRANSFERRED_SIZE Register

a) Address: {Image1_base_addr[31:12],12'h010}, where “Image1_base_addr” is the base address of PCI image1.

b) Function: This 32-bit register contains the total number of bytes that have been sent in the last completed DMA request.

c) NOTE: This register is NOT writable. Also, it should be read when the host CPU receives the interrupt that indicates the completion of the DMA request.

6) STATUS Register

a) Address: {Image1_base_addr[31:12],12'h014}, where “Image1_base_addr” is the base address of PCI image1.

b) Function:

STATUS[6]: Address Error. If any of the addresses involved in the DMA transfer (e.g., the content in the “**DMA_POT”** register, the content in the “**DMA_LEN**” register, etc) are not 32-bit aligned (i.e., the 2 LSBs are not “00”), the DMA transfer will be stopped and STATUS[6] will be set to “1”.

STATUS[5]: DMA completion flag. If the DMA transfer is finished (due to error or normal completion), this bit is set “1”.

STATUS[4]: Hardware error. If this bit is set to “1”, an unexpected hardware error (e.g., memory soft error) is met during the transfer and the state machine of the DMA transfer is destroyed.

STATUS[3:0]: Bus transfer error. If any of these four bits are set to “1”, a PCI or WB bus operation error is met during the transfer.

STATUS[7]: Watchdog counter overflowed. If the DMA transfer is stopped due to the overflow of the watchdog counter, this bit is set to “1”.

STATUS[15]: DMA is in operation. If this bit is set, a DMA operation is using the DMA engine.

STATUS[11]: Channel 3 up flag. If this bit is “1”, channel 3 is up and ready for data transfer.

STATUS[10]: Channel 2 up flag. If this bit is “1”, channel 2 is up and ready for data transfer.

STATUS[9]: Channel 1 up flag. If this bit is “1”, channel 1 is up and ready for data transfer.

STATUS[8]: Channel 0 up flag. If this bit is “1”, channel 0 is up and ready for data transfer.

STATUS[19]: error flag for channel 3. If this bit is “1”, an error has been detected during data transfer in the Aurora link in channel 3. To use this register, the user must first reset channel 3 after power up. Also, to record a new error, the user must first reset channel 3.

STATUS[18]: error flag for channel 2. If this bit is “1”, an error has been detected during data transfer in the Aurora link in channel 2. To use this register, the user must first reset channel 2 after power up. Also, to record a new error, the user must first reset channel 2.

STATUS[17]: error flag for channel 1. If this bit is “1”, an error has been detected during data transfer in the Aurora link in channel 1. To use this register, the user must first reset channel 1 after power up. Also, to record a new error, the user must first reset channel 1.

STATUS[16]: error flag for channel 0. If this bit is “1”, an error has been detected during data transfer in the Aurora link in channel 0. To use this register, the user must first reset channel 0 after power up. Also, to record a new error, the user must first reset channel 0.

Other bits: Not used yet. Reading returns 0.

c) *NOTE:* This register is NOT writable. Also, it should be read when the host CPU receives the interrupt that indicates the completion of a DMA transfer. Whenever a new DMA transfer starts, this register is automatically cleared.

7) Reset Register

a) Address: {Image1_base_addr[31:12],12'h058}, where “Image1_base_addr” is the base address of PCI image1.

b) Function: This register is for resetting the different channels. A “1” in bit[0] will reset channel 0. Bits[3:1] have similar operations.

c) *NOTE:* bits [31:4] are reserved and must be zero.

8) MON Register

a) Address: {Image1_base_addr[31:12],12'h05C}, where “Image1_base_addr” is the base address of PCI image1.

b) Function: This register is for generating the MON pulses.

MON[0]: “1” generates a pulse for MON_A0.

MON[1]: “1” generates a pulse for MON_B0.

MON[4:2]: Width of the pulse for MON_A. 3'b000: 16 clock cycles. 3'b001: 8 clock cycles. 3'b010: 4 clock cycles. 3'b011: 32 clock cycles. Others: 16 clock cycles.

MON[7:5]: Width of the pulse for MON_B. 3'b000: 16 clock cycles. 3'b001: 8 clock cycles. 3'b010: 4 clock cycles. 3'b011: 32 clock cycles. Others: 16 clock cycles.

Other bits: Not used yet. Reading returns 0.

c) NOTE: MON[0] and MON[1] are automatically cleared by the firmware. So the user is not required to manually clear these two bits.

8) START_END_PULSE Register

a) Address: {Image1_base_addr[31:12],12'h060}, where “Image1_base_addr” is the base address of PCI image1.

b) Function: This register is for holding the start and end pulses from the MON pins.

START_END_PULSE[6]: start_end_pulse_A.

START_END_PULSE[5]: start_pulse_A.

START_END_PULSE[4]: end_pulse_A.

START_END_PULSE[2]: start_end_pulse_B.

START_END_PULSE[1]: start_pulse_B.

START_END_PULSE[0]: end_pulse_B.

c) NOTE:“start_pulse_A” is generated from pin “MON_A2” and is simply the sampling of the pulse on the pin. The required minimum width of the pulse on “MON_A2” is longer than 15ns. Similarly, “end_pulse_A” is generated from pin “MON_A4”, “start_pulse_B” is generated from pin “MON_B2”, and “end_pulse_B” is generated from pin “MON_B4”. The “start_end_pulse_A” bit is asserted at the positive edge of “start_pulse_A” and negated at the positive edge of “end_pulse_A”. If the positive edge of “end_pulse_A” arrives more than 5 seconds later than the positive edge of “start_pulse_A”, a timeout will be generated and “start_end_pulse_A” will be cleared automatically. “start_end_pulse_B” is used to measure the interval between “start_pulse_B” and “end_pulse_B “ and works in a similar way as “start_end_pulse_A”.

9) Other Registers

The other addresses are reserved and must be zero.

3. Linked List Descriptors

Our PCI DMA unit works in the mode of linked list descriptors. In this mode, the DMA engine will fetch the real DMA request information from the PCI bus. This is illustrated in Fig. 2.

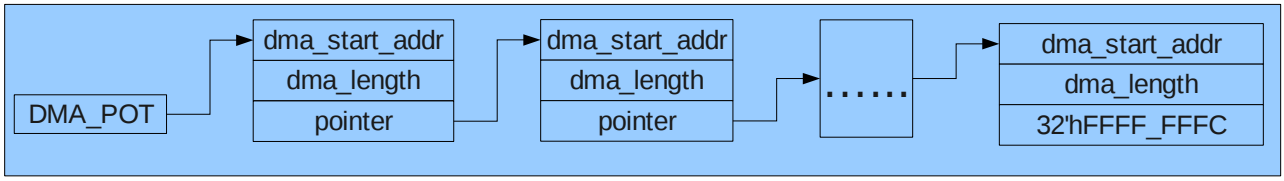

Fig.2 Linked List Descriptors

The “DMA_POT” register contains a pointer to the head of the list of the DMA operation descriptors. When the DMA engine starts a DMA transfer, it first uses the content in “DMA_POT” as an address to read “dma_start_addr”, “dma_length” and “pointer” from the PCI bus. Note that “DMA_POT” only contains the address of “dma_start_addr”. By adding 4 and 8 to the address of “dma_start_addr” we can obtain the addresses of “dma_length” and “pointer”.

The data in “dma_start_addr” specifies the starting address of the DMA transfer on the PCI bus. The data in “dma_length” specifies the total number of bytes that the DMA transfer is going to send or (receive) starting from the address specified by “dma_start_addr”.

If the content of “pointer” is 32'hFFFF_FFFC, the list is ended. Otherwise, the content of “pointer” contains the address of “dma_start_addr” of the next DMA operation descriptor in the list.

Note that the contents in “dma_start_addr”, “dma_length” and “pointer” MUST be 32-bit aligned (i.e., the 2 LSBs are “00”).

The mode of linked list descriptors allows us to fill different segments in the PCI bus space by issuing a single DMA command to the DMA engine. For example, if we write 1024 to the “DMA_LEN” register in the DMA engine and specify in the main memory that the value of “dma_length” of each DMA operation descriptor is 256, then the DMA engine will automatically fill 4 address segments each of which is as long as 256.

4. Example

In this section, we show an example from our testbench to illustrate the typical flow of a DMA transfer.

Our testbench is based on the one from OPENCORES. It was originally provided to test the PCI-WB bridge but can be modified to test our PCI DMA unit. The framework of the testbench is shown in Fig.3.

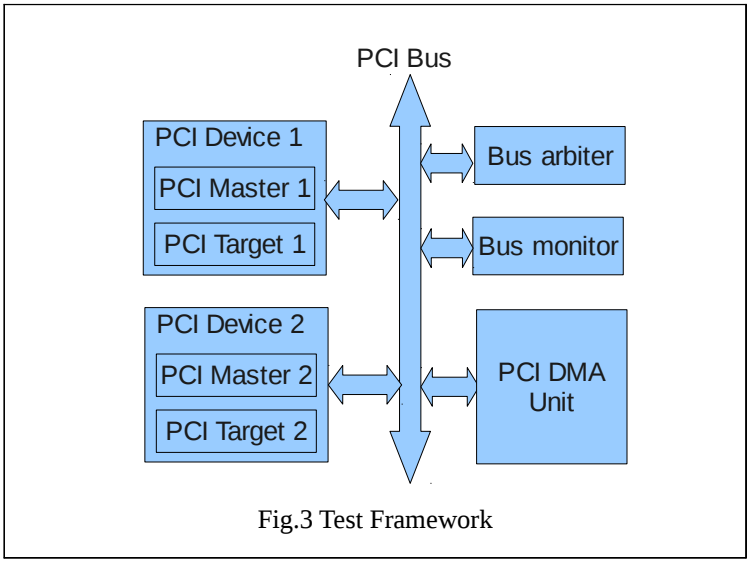

Fig.3 Test Framework

In Fig.3, “PCI Device 1” and “PCI Device 2” are behavioral PCI devices. Each of them consists of a behavioral PCI master and a behavioral PCI target. “Bus arbiter” is the arbiter of the PCI bus. “Bus monitor” is used to detect the bus errors during the simulation.

The flow of the simulation is show below.

1) Use “PCI Master 1” to configure “PCI Device 1”, “PCI Device 2”, and “PCI DMA Unit” with the configure cycles of the PCI bus.

2) Use “PCI Master 1” to set the information of PCI image1 of “PCI DMA Unit”. This includes setting the base address, address mask and translation address. This is done with the memory write cycles of the PCI bus. Refer to “PCI IP Core Specification” provided by OPENCORES for the address offsets of the registers.

3) Use “PCI Master 1” to set the content of the “interrupt control register” of the PCI-WB bridge in “PCI DMA Unit”. Refer to “PCI IP Core Specification” provided by OPENCORES for the address offset of the register. In this step, we still use the memory write cycles of the PCI bus.

4) Use “PCI Master 1” to write the linked list of DMA operation descriptors into the memory in “PCI Target 2”. This is done with the memory write cycles of the PCI bus.

5) Use “PCI Master 1” to write the “DMA_LEN” register in “PCI DMA Unit”. This is done with the memory write cycles of the PCI bus.

6) Use “PCI Master 1” to write the “DMA_POT” register in “PCI DMA Unit”. This is done with the memory write cycles of the PCI bus. After this step, the “DMA_POT” register contains an address which points to the head of the list of the DMA operation descriptors in the memory in “PCI Target 2”.

7) Use “PCI Master 1” to write the “CTRL” register in “PCI DMA Unit”. This is done with the memory write cycles of the PCI bus. CTRL[30] must be set to “1” to start the DMA transfer.

8) The DMA operation starts.

9) The DMA operation finishes and the interrupt signal on the PCI bus is asserted.

10) “PCI Master 1” reads the content of the “interrupt status register” of the PCI-WB bridge in “PCI DMA Unit” to confirm that an interrupt is generated by “PCI DMA Unit”. Refer to “PCI IP Core Specification” provided by OPENCORES for the address offset of the register. In this step, we use the memory read cycles of the PCI bus.

11) “PCI Master 1” reads the “ TRANSFERRED_SIZE” register to know how many bytes have been transferred. In this step, we use the memory read cycles of the PCI bus.

12) “PCI Master 1” reads the “STATUS” register to know whether any error has occurred during the transfer. In this step, we use the memory read cycles of the PCI bus.

13) “PCI Master 1” writes the “CTRL” register in “PCI DMA Unit”. This is done with the memory write cycles of the PCI bus. CTRL[30] is set to “0” to release the DMA engine and clear the interrupt.