화상 데이터 전송을 위한 PCI 버스 DMA 인터페이스 설계에 관한 연구
A Study on the Design of PCI Bus DMA Interface for Video Data Transfer

이 주 석 (전자과)
Joo-Sock Lee(Dept. of Electronics)

Key Words : PCI (Peripheral Component Interconnect), DMA(Direct Memory Access), PC(Personal Computer), ISA(Industrial Standard Architecture), EISA(Extended ISA), MCA(Micro Channel Architecture), VESA(Video Electronics Standard Association), VGA(Video Graphic Array), CPU(중앙연산장치), IRDY(Initiator ready), TRDY(Target Ready), GUI(Graphic User Interface)

ABSTRACT : The PCI local bus master interface block has been designed in this paper. In case of real time video data processing, The data transfer rate is exponentially increased and the ISA bus which has a 16bit and 8MHz bus architecture cannot meet that transfer rate. This bottleneck condition makes the high speed microprocessor wait for the data from peripheral devices. Therefore, the system performance is determined not by a microprocessor speed but by a peripheral device's speed.

The PCI local bus architecture is focused on communication among the high bandwidth LSI devices. The PCI bus DMA interface was designed by Verilog HDL and synthesized with the Synopsis. The functional operation of the synthesized logic was verified with a cell delay calculator and the Cwave, verilog simulator. The expected PCI data operation with DMA was observed. The designed PCI DMA master interface operates as a PCI master device fully. It can initiates bus operation by itself. The DMA controller controls the data transfer between a system memories and a local memories. Therefore, the CPU can process other operation during the memory writing or reading operations.

1. 서론

PC(Personal Computer)가 탄생한 이래 마이크로프로세서의 기술은 급진적으로 발전하여 왔으며, 이에 따라 PC도 고성능, 고기능을 갖추어 왔고, 사용자는 복잡한 그래픽, LAN(Local Area Network)을 통한 대용량 데이터 전송, 실시간 비디오 처리 등을 요구하게 되었다. CPU(Central Processing Unit)의 발전속도는 조만간 클록주파수가 1GHz인 제품이 시제품으로 제작되고 있으나, 주변기기와의 대용량 데이터를 실시간에 처리, 전송하기 위한 버스의 발전속도는 이를 따라가지 못하고 있다. 해상도가 640×480인 영상데이터를 초당 30프레임(Frame)을 실시간에 전송하기 위해서는 9.21MB/Sec의 전송속도가 필요하고, 1280×1024 사이즈의 16비트 영상데이터를 전송하기 위해서는 78.64MB/Sec의 전송속도가 필요하게 된다. 따라서 저속의 ISA(Industrial Standard Architecture), VESA(Video Electronics Standard Association)버스로부터 132MB/Sec 전송이 가능한 고속의 PCI (

Peripheral Component Interconnect)버스로 바뀌고 있고, PC 플레폼 변화를 정의한 " PC 98" 에서 ISA 버스를 PC의 기본사양에서 배제함에 따라 PCI 버스가 독보적인 PC용 확장 버스 기술로서 독주가 예상되고 있다. 따라서 사운드카드나 모뎀, 비디오 카드 등 ISA버스를 사용하던 장치들이 PCI 장치로 변신하고 있으며, PC에서의 응용제품은 반드시 고속의 PCI 버스에 연결을 시켜야 의미가 있게 된다[8,9,10,11].

PCI는 초고속 프로세서(Highspeed Process)나 주변제어기(Peripheral Controller)의 로칼 버스(Local Bus)로 차세대의 고성능 컴퓨터의 로칼 버스 요구조건을 만족하므로, Digital Equipment (DEC Alpha), Motorora(Power PC), Intel(Pentium)등에서도 이 구조를 사용하고 있고, 사용자가 설계한 PC 응용제품은 반드시 PCI버스에 직접 연결되어야 원하는 시스템 성능을 낼 수 있게 된다[8]. 또한 PCI는 고속 버스일 뿐 아니라 자동 컨피규레이션(Autoconfiguration)에 해당하는 컨피규레이션 영역 레지스터 블록(Configuration Space Register block)을 포함하고 있어, 고성능 주변기기(SCSI, IDE, LAN, Graphic, video)등에 별도장치 없이 인터페이스를 제공하고 플러그 앤드 플레이도 가능하게 한다.

본 논문은 720×480 사이즈의 영상데이터를 PCI 버스를 이용하여 전송하기 위한 마스터 모드 DMA(Direct Memory Access) 전송이 가능한 인터페이스부를 설계한 것으로, Verilog HDL을 이용하여 기능 정의를 하였고, Synopsys를 이용하여 회로 합성을 하였으며, Cadence Tool을 이용한 시뮬레이션으로 설계된 회로의 동작을 검증하였다.

본 논문의 구성은 제 1장 서론에 이어 제 2장 본론에서 PCI에 대한 내용을 제 3장에서 설계된 내용과 제 4장 시뮬레이션 내용을 설명하고 마지막으로 결론을 내렸다.

2. 본 론

2. 1 PCI 시스템

Fig. 1은 전형적인 PCI시스템을 보여준다. CPU는 PCI 브리지를 통하여 PCI 버스에 연결되어 있고, 여러 가지 사용자 장치가 PCI 버스에 연결되어 있으며, 각 장치의 내부는 타겟 혹은 마스터 모드의 PCI 인터페이스 회로가 있어 PCI 버스에 연결되게 된다.

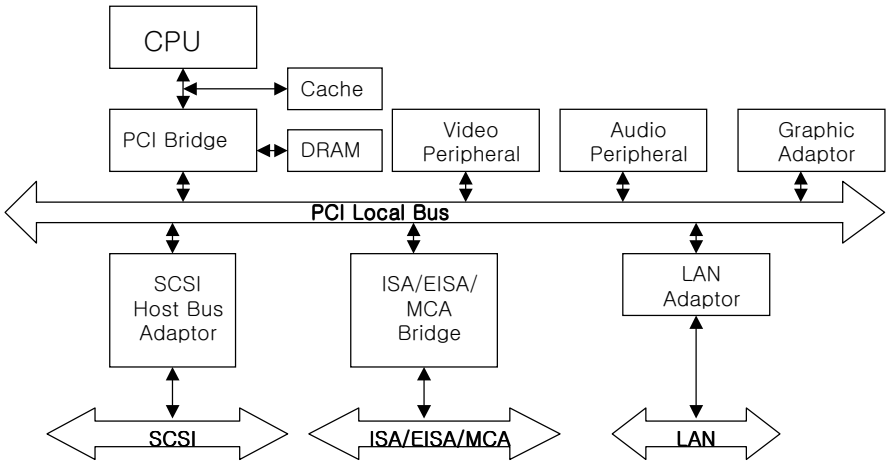

Fig. 1 The PCI system

PCI 버스의 시스템 클록은 33MHz로 32Bit 데이터를 전송하며 최대 132MB/Sec의 전송 속도가 가능하며, 버스를 64Bit로 확장하고 66MHz의 시스템 클록을 사용하면, 528MB/Sec 데이터 전송이 가능하다. PCI는 OS 소프트웨어에 의해 컨피규레이션 영역을 읽기도 하고, 기록도 하여 초기화하므로 플러그 앤 플레이가 가능하게 되는데 이를 위한 256 바이트의 컨피규레이션 레지스터가 할당되어 있다. 컨피규레이션 영역을 억세스 하는 것은 각각의 디바이스에 대하여 칩 선택 신호인 IDSEL 제어핀과 어드레스를 주고, 이 어드레스를 디코딩하여 매칭이 되는 장치를 찾게되며, 이 장치에 대한 정보를 제공하게 된다[7]. PCI에서 핀은 공유하여 사용하므로, 타겟 모드인 경우 최소 47개, 마스터 모드인 경우 최소 49개 핀이 필요하다. Fig. 2는 PCI 마스터 장치의 신호 구성을 나타낸다.

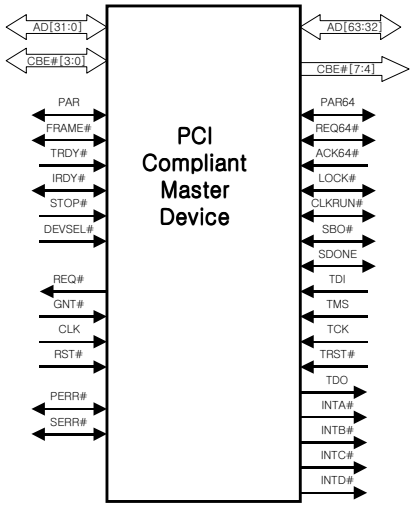

Fig. 2 Signals of PCI master device

2. 2 PCI 마스터 장치의 신호

CLK : 기준 클록으로 33MHz 혹은 66MHz
RST# : 리셋 신호
AD[31:0] : 어드레스와 데이터를 중첩한 32비트 신호선
CBE#[3:0]: 중첩된 명령어와 바이트 인에이블 신호
PAR : AD[31:0]와 CBE#[3:0]의 짝수 페리티 신호
FRAME# : PCI 버스 싸이클의 시작과 데이터 전송의 끝을 나타내는 신호
IRDY# : 마스터가 데이터를 전송할 준비가 되어있다는 신호
TRDY# : 선택된 타겟 장치가 데이터 전송의 준비가 되어있다는 신호
STOP# : 타겟이 마스터에게 현재 싸이클 중단을 요구하는 신호
LOCK# : 복수의 싸이클이 계속 수행되어야 함을 알리는 신호
IDSEL : 칩 선택 신호
DEVSEL : 어드레스에 의해 타겟이 선택되었음을 알리는 신호
REQ# : 마스터가 PCI 버스 사용을 요구하는 신호
GNT# : PCI 버스의 사용을 허가하는 신호
PERR# : 데이터 페리티 에러 신호

SERR# : 어드레스 페리티 에러, 데이터 페리티 에러가 발생했음을 알리는 신호
INTA#, INTB#, INTC#, INTD# : 인터럽트 신호
AD[63:32] : 64비트 확장 시 중첩된 어드레스/데이터 신호
CBE#[7:4] : 64비트 확장 시 추가된 32비트에 대한 명령어/바이트 인에이블 신호
REQ64# : 마스터가 64비트의 전송을 요구하는 신호
ACK64# : 타겟이 64비트로 데이터를 전송하려 할 때의 신호
PAR64# : AD[63:32]와 CBE#[7:4]에 해당하는 짝수 페리티 신호
SBO# : 캐쉬 상태를 나타내는 신호
SDONE : 캐쉬에서 현재 억세스를 위한 스눕(snoop) 상태
TDI, TDO, TCK, TMS, TRST# : JTEG 테스트를 위한 신호

2. 3 PCI 버스의 전송 동작

PCI에서 기본적인 버스 전송 메카니즘은 하나의 어드레스 위상과 여러 개의 데이터 위상으로 구성된 버스트 전송이다. FRAME# 제어신호가 "Low"로 되는 첫 번째 클록 사이클이 어드레스 전송의 타이밍이 된다. FRAME#은 계속되는 데이터 타이밍에서도 그대로 액티브 상태이고, 마지막 버스트 데이터 전송의 시작에서 "High"로 된다. 데이터 전송은 IRDY#(Initiator Ready, 버스 마스터가 기록될 데이터를 공급하거나, 데이터를 읽을 준비가 되었을 때), TRDY#(Target Ready, 타겟 디바이스가 기록될 데이터를 받거나, 읽을 데이터를 공급하는)가 "Low"로 되었을 때 일어난다.

PCI에 연결된 디바이스는 버스 마스터가 요구하는 동작이 어떤 동작인가를 마스터로부터 전송된 버스 커맨드에 따라 수행한다. 버스 커맨드는 어드레스 페이즈 동안 4 비트 CBE# 라인을 인코딩 하여 알 수 있고, 데이터 페이즈 동안에는 CBE# 라인은 워드 전송에 대한 바이트 인에이블 신호가 된다.

기본적인 PCI의 읽기와 쓰기 동작은 Fig. 3, Fig. 4와 같다.

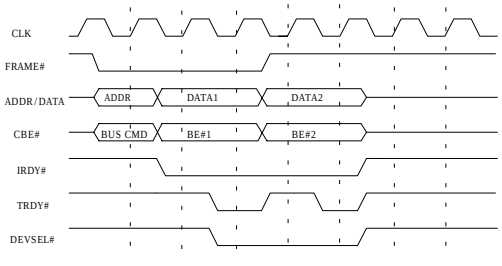

Fig. 3 The PCI Write Operation

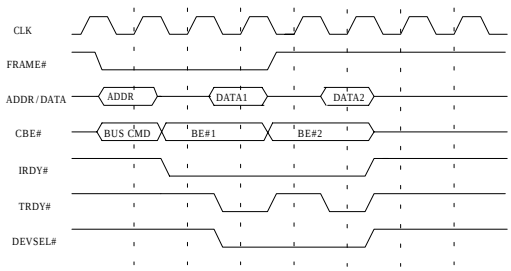

Fig. 4 The PCI Read Operation

3 PCI 버스 DMA 인터페이스 설계

PCI DMA 인터페이스 설계의 전체 구성은 Fig. 5와 같다. PCI 버스에 인터페이스 되는 장치는 크게 타겟 모드와 마스터 모드가 있는데, 마스터 모드는 버스의 사용을 요구할 수 있으며, 타겟은 마스터에 의해 선택되어 데이터를 마스터에 전송하거나 마스터로부터의 데이터를 수신하게 된다. PCI 버스에서 마스터 동작에는 두 가지 경우가 있다. 첫 번째 마스터 동작은 마스터용 FIFO(First In First Out)를 사용하여, 데이터의 송수신을 하는 것이며, 두 번째

경우는 CPU의 부담을 덜어주기 위한 DMA를 사용한 마스터 동작이 있다. 여기서는 CPU를 경유하지 않고 화상데이터를 메모리로 전송하기 위한 DMA를 사용한 마스터동작을 설계하였다.

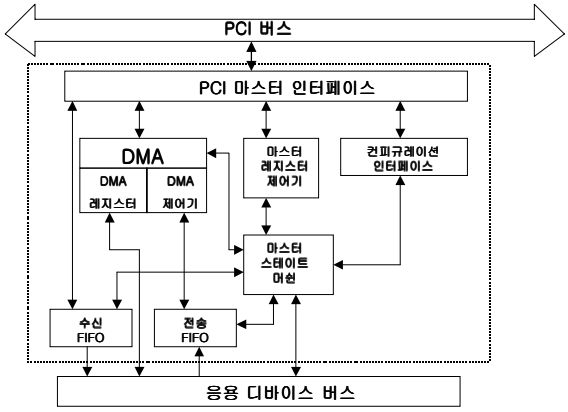

Fig. 5 The block diagram of PCI DMA interface

DMA는 메모리로부터 데이터를 직접 읽고, 쓸 수 있는 디바이스로 CPU가 메모리로부터의 데이터 전송에 관여하지 않으므로, 다른 작업을 할 수 있어 CPU의 부담을 줄일 수 있으며, 화상데이터는 수평 수직 위치에 관한 데이터로 DMA의 전송이 유리하다. Fig. 5에서 DMA를 이용한 메모리 전송을 하기 위해서는 DMA 레지스터에 필요한 값(시작점, 전송할 데이터 수, 상태)을 응용장치로부터 입력받아 셋팅한 후, 응용 장치로부터 데이터를 먼저 전송 FIFO에 저장하고, DMA가 버스의 사용 권한을 마스터 스테이트 머신에 요구하여 버스 사용권을 얻은 후 FIFO의 데이터를 PCI 버스를 경유하여 메모리로 전송하게 되고, 수신시에는 수신 FIFO 에 먼저 저장한 후 응용장치에서 읽게 된다.

Fig. 6은 구현된 DMA 스테이트 머신의 흐름도이며 DMA 제어기의 동작은 다음과 같다. 먼저 DMA 레지스터에 데이터를 입력하고, DMA 동작의 시작을 지시하는 레지스터에 '1'을 입력함으로써 시작된다. DMA를 이용하여 전송할 데이터의 개수가 일치하지 않거나, DMA 전송을 중지하는 레지스터가 셋팅 되면 DMA 사용을 중지한다. DMA 전송 방향은 전송방향을 나타내는 레지스터를 보고 판단하며, 메모리의 데이터 전송의 경우, 마스터 스테이트 머신에 버스 사용을 요구하여 버스 사용권을 획득한 후, 데이터를 PCI 버스를 통하여 메모리에 전송하며, 전송 중 에러가 발생하면 사용을 중단한다. DMA 레지스터는 목적지의 주소를 나타내는 어드레스 레지스터, 전송할 데이터의 숫자를 나타내는 워드 카운터, DMA 명령, DMA 상태, DMA 제어 레지스터로 구성되어 있다.

일반적으로 응용회로의 클록과 PCI 클록은 별개이기 때문에, 응용회로에서 PCI 버스에 직접 데이터를 전송할 수가 없다. 따라서 FIFO 메모리는 필요하며, FIFO 메모리는 SRAM을 이용하여 구현하였다.

Fig. 7은 구현된 마스터 스테이트 머신의 흐름도이다. 버스의 사용을 위해 REQ# 신호를 브리지에 주어 허가신호 GNT#를 받은 후, FRAME# 신호를 보고 버스 사용여부를 판단한다. 만약 PCI 버스가 사용중이라면 사용할 수 있을 때까지 FRAME# 신호의 구동을 유보한다. 사용중이 아니라면 FRAME#과 IRDY#를 이용하여 버스 사용권을 장악하고 TRDY#와 IRDY# 신호가 각각 "Low"일 때 데이터를 전송한다. 데이터 전송 시 마지막 데이터 전송인가 확인하여 마지막 데이터 전송 전에 FRAME# 신호를 "High"로 하여 버스 사용권을 반납

한다. 또 데이터 전송 중에 버스 사용의 중단을 요구하는 에러신호(SERR#, PERR#)가 발생하면, 즉시 버스 사용권을 반납한다.

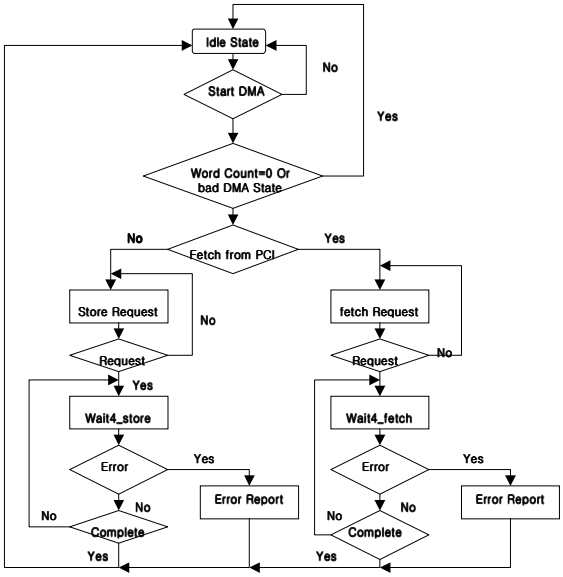

Fig. 6 The flowchart of DMA state machine

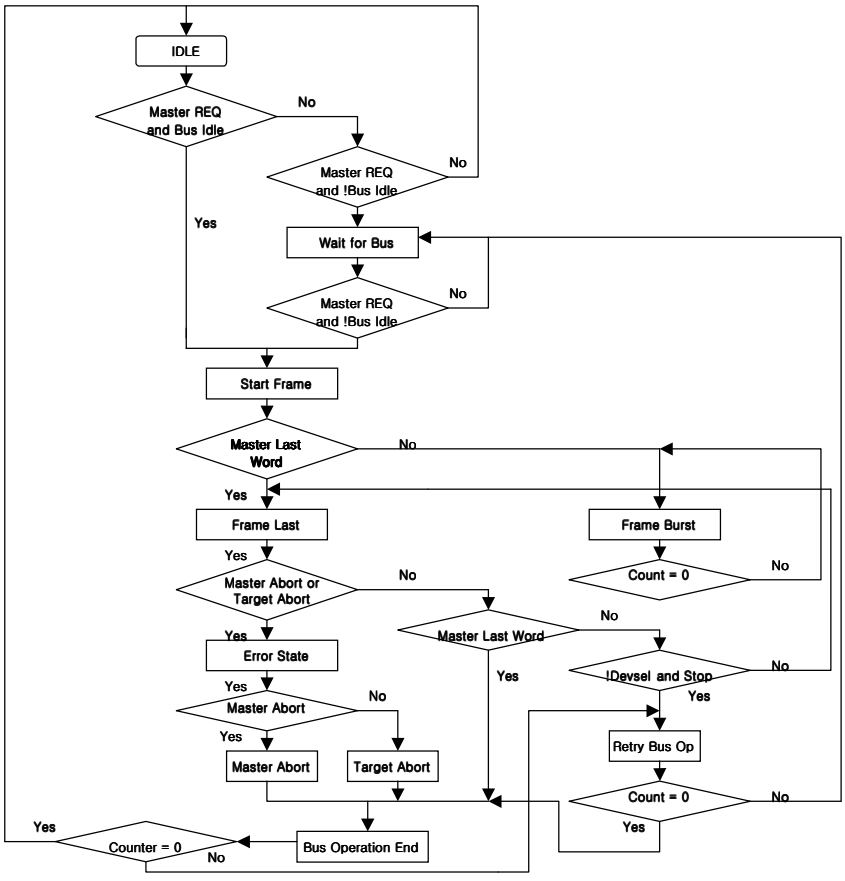

Fig. 7 The flowchart of master state machine

4. 시뮬레이션 결과

4. 1 검증방법

회로는 각 블록을 Verilog HDL을 사용하여 기능을 정의하고, 이를 Synopsys를 통하여 게이트 회로로 구현한 후 블록별로 시뮬레이션을 하였다. 또 각 블록을 결합하여 전체 시스템을 구성한 후, PCI의 기본 동작과 마스터 모드 DMA 전송에 대하여 시뮬레이션을 한 후 게이트의 전달 지연을 고려한 프리 레이아웃(Pre-Layout) 회로로 최종 동작을 확인하였다. 전체 시뮬레이션 방법은 데이터를 임의로 발생시켜 어레이를 구성한 후, 이 데이터를 응용회로의 데이터로 하여 입력하여 PCI의 DMA 마스터 기능으로 전송한 후, 전송된 데이터와 원래의 어레이 데이터를 비교하는 방법으로 전송여부를 판단하여 수행하였다.

4. 2 시뮬레이션 결과

4. 2. 1 컨피규레이션 레지스터 입출력 동작 확인

컨피규레이션 레지스터에 데이터를 읽고 쓰는 동작을 확인하기 위해 IDSEL(Chip Select) 신호와 AD[31:0], CBE#[3:0] 신호선에 데이터를 입력하여 동작을 확인하였다. Fig. 8은 컨피규레이션 레지스터에 값이 입력되고 있음을 나타내는 시뮬레이션 파형으로 DEVSEL# 신호가 "Low" 상태에서 IRDY#와 TRDY#가 "Low"로 되어 있으므로 데이터가 전송되고 있음을 알 수 있다. Fig. 9는 커멘드 레지스터에 0FFFh의 값이 입력되는 동작을 보여주고 있다. IDSEL 신호가 "High"인 영역에서 컨피규레이션 레지스터 쓰기 동작 명령어를 입력하고, 커멘드 레지스터에 해당하는 주소를 AD가 가리키며 다음 클록에서 값이 입력되게 하였고, 이때 컨피규레이션 레지스터 블록에서 값을 보고 동작함을 확인하였다.

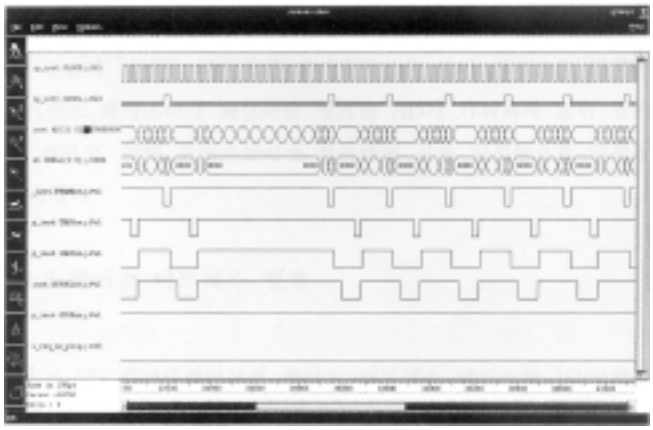

Fig. 8 The configuration input waveform

4. 2. 2 베이스 어드레스 설정

Fig. 10은 컨피규레이션 레지스터의 베이스 어드레스 레지스터 10H에 01000000H를 입력하여 기록하는 파형이다. IDSEL 핀이 "High"인 구간에서 컨피규레이션 쓰기 명령을 입력하고, 베이스 어드레스 레지스터에 해당하는 번지를 AD가 주어 입력되게 하였으며, 컨피규레

이션 레지스터의 10H번지의 데이터를 보고 동작을 확인하였다.

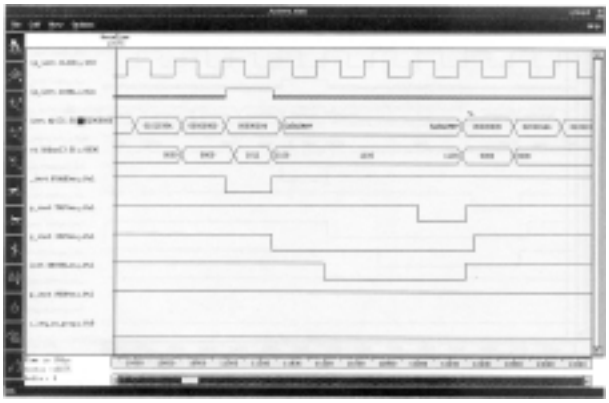

Fig. 9 The command register input waveform

Fig. 10 The base address input waveform

4. 2. 3 DMA 읽기 동작

Fig. 11은 DMA를 사용하여 메모리에서 데이터를 읽어 버스를 통해 외부 PCI 버스에 전달하는 과정을 보여주고 있다. DMA 전송을 위해서 필요한 레지스터 값을 각각 외부에서 입력하였고, 메모리의 데이터는 랜덤하게 외부에서 발생시켰다. 레지스터는 PCI_Addr, WordCount, command_reg, start_dma 순으로 입력하였고, 입력을 위해 MP_data_bus[31:0]와 스트로브 신호(MP_wordcount_reg, MP_PCI_reg, MP_start_dma)를 이용하였다. 레지스터 전송이 완료되고, Fig. 11의 66450에서 데이터 전송이 입력한 개수의 데이터가 발생함을 확인하였고, 이를 원래의 데이터와 비교하여 일치함을 확인하였다. 또 데이터 전송이 완료되기 바로 전에 FRAME# 신호가 "High"로 되고, DEVSEL#, IRDY# 신호들이 버스 사용이 끝나자 원래 상태로 "High"로 되었음을 확인하였다.

4.2.4 DMA 쓰기 동작

Fig. 12는 DMA를 사용하여 메모리로 데이터를 쓰는 동작 파형 결과이다. 입력 데이터는

랜덤하게 32비트 값을 설정하였고, 레지스터는 command_reg, PCI_Addr, WordCont, start_dma 순으로 입력하였다. 레지스터 전송이 완료되고 Fig. 12의 66450에서 데이터 전송이 입력한 개수의 데이터가 발생함을 확인하였고, 이를 원래의 데이터와 비교하여 일치함을 확인하였다. fetch_from_PCI는 start_dma 가 입력될 때 "Low"이므로, 데이터 방향이 메모리로 가고 있음을 확인할 수 있었다.

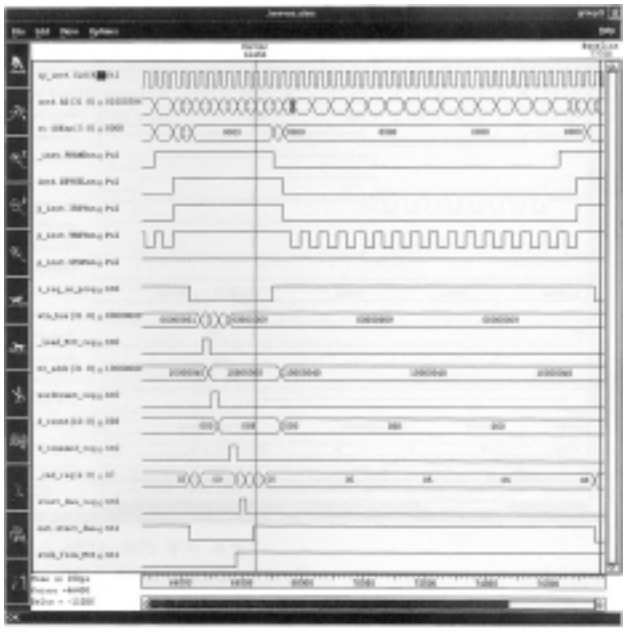

Fig. 11 DMA read operation waveform

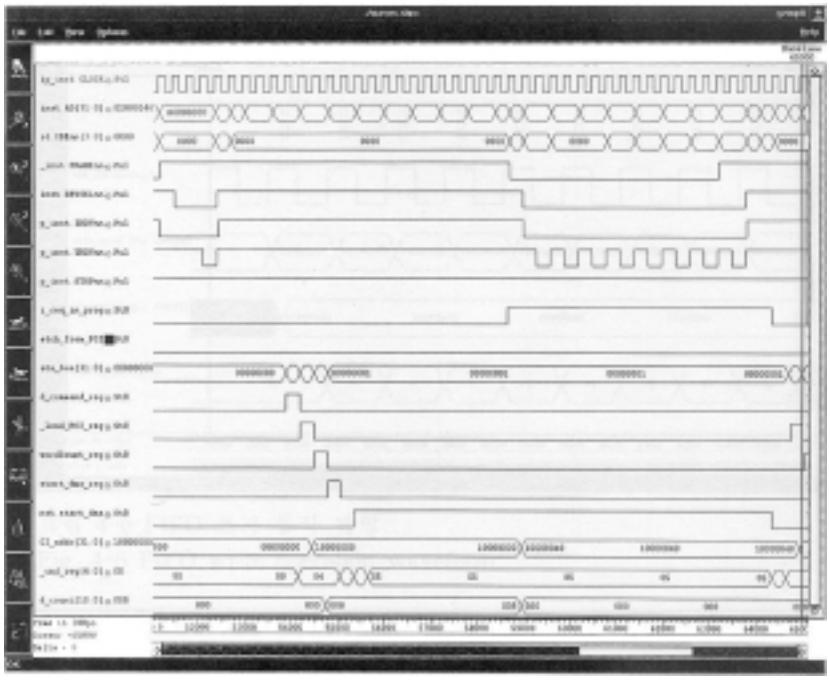

Fig. 12 DMA write operation waveform

4. 2. 5 FIFO 동작

DMA 전송을 위해서는 FIFO를 사용하는데 DMA에서 PCI버스를 통하여 데이터를 메모리에 전달하는 전송 FIFO의 동작을 확인하였다. Fig. 13은 데이터의 전송시 FIFO로 데이터가 쓰여짐을, Fig. 14는 전송 FIFO에서 데이터가 읽혀지는 상태에서 데이터가 나가는 것을 보여주고 있다. 각각에서 각 워드 카운트에 해당하는 데이터가 일치함을 확인할 수 있고, write_enable_pulse도 정상적으로 동작함을 확인하였다.

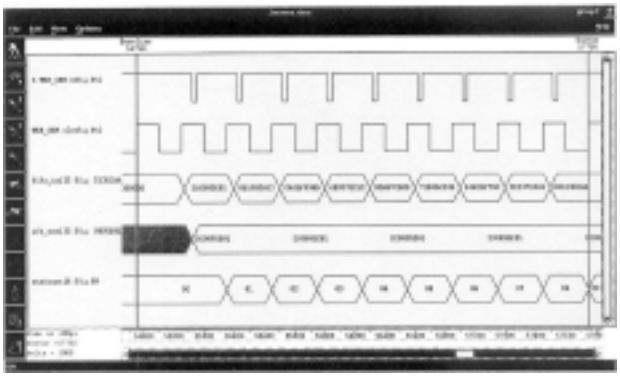

Fig. 13 FIFO write operation waveform

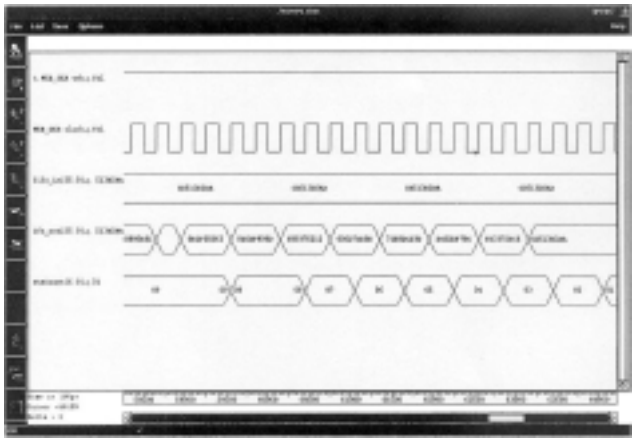

Fig. 14 FIFO read operation waveform

4 결 론

본 논문에서는 PCI 버스 DMA 마스터 인터페이스 블록을 설계하였다. PC 응용제품을 PC에 의미 있게 연결하기 위해서는 반드시 PCI 버스에 직접 연결해야 하며, 이를 위해 인터페이스 회로로, 마스터에 의해 지배를 받는 타겟 모드와 독자적으로 버스의 사용권을 획득하여 타겟으로 데이터 전송을 할 수 있는 마스터 모드 중 본 논문에서는 CPU의 부하를 줄이고 DMA를 이용하여 데이터 전송을 할 수 있는 마스터 모드 PCI 인터페이스 회로를 Verilog HDL로 구현하였고, 회로로 합성한 후, 시뮬레이션으로 동작을 확인하였다. 따라서 본 논문에서 설계된 회로는 향 후 PC 응용제품을 개발함에 있어 PCI DMA 마스터 인터페이스 회로로 사용할 수 있으며, 이를 이용하여 시스템 동작 확인을 할 수 있고, 궁극적으로는 응용제품과 PCI 인터페이스 회로를 함께 ASIC화하여 경쟁력 있는 IC개발의 핵심요소가 될 수 있다.

참고문헌

(1) 장기혁 편역, 1986, *영상처리 시스템의 기초와 설계 제작*, 도서출판 세운
(2) A.H.M. van Roermund et. al., Aug. 1989, A General-Purpose Programmable Video Signal Processor, *IEEE Trans. on Consumer Electronics*, Vol. 35, No. 3
(3) Philips, 1995, "*Desktop Video Data Handbook*"
(4) 오재광 저, 1995, *PC 인터페이스 제작과 실제*, 크라운 출판사
(5) Karl Wang, Chris Bryant et. al., Apr. 1995, Designing the MPC105 PCI Bridge/Memory Controller, *IEEE MICRO*, Vol. 15, No. 2.
(6) Tom Shanley and Don Anderson,1995, 3rd ed. *PCI System Architecture*, Addison Wesley Publishing Company,
(7) PCI SIG, June 1, 1995, *PCI Local Bus Specification Revision 2.1*
(8) QuickLogic, 1997, *QAN10 PCI Using the QL2003 FPGA Data Sheet*
(9) Actel Corporation, May 1998, *Core PCI Target+DMA Data Sheet*
(10) Mentor Graphics, September 8, 1997, *32-bit PCI Bus to 32-bit Backend PCI Core Data Sheet*
(11) Brooltree, 1997, *Bt848 Single-Chip Video Capture for PCI Data Sheet*