ACTEL FPGA를 이용한 PCI 인터페이스 설계에 관한 연구
A Study on the Design of PCI Interface using ACTEL FPGA

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

Key Words : PCI (Peripheral Component Interconnect), FPGA(Field Programmable Gate Array), PC(Personal Computer), ISA(Industrial Standard Architecture), VESA(Video Electronics Standard Association),VGA(Video Graphic Array), CPU(중앙연산장치)

ABSTRACT : The Peripheral Component Interconnect(PCI) is a standard Local Bus for high speed processors and peripheral controllers. PCI is intended to meet the local bus requirements of next generation high performance computer systems. It is meaningless for PC applications not to connect PC Local Bus(PCI) and PCI interface circuit is necessary. We designed this PCI Interface circuit by ACTEL 1425 Chips and simulated by Viewlogic and conformed the operations of PCI by extracted delayed parameters after placement and routing.

1.서 론

PC의 버스 구조는 저속의 ISA, VESA 버스로부터 고속의 PCI 버스로 바뀌고 있고, 최근에는 ISA 버스와 함께 PC의 양대 버스기술로 자리잡아 온 PCI 버스를 인텔· 마이크로소프트(MS)·컴팩 등을 주축으로 한 PC업체가 향후 PC 플레폼 변화를 정의한 " PC 98" 사양서에서 ISA 버스를 기본사양에서 배제함에 따라 PCI 버스가 독보적인 PC용 확장 버스 기술로서 독주가 예상되고 있다. 이에 따라 사운드카드나 모뎀등 ISA버스를 사용하던 장치들이 PCI 장치로 변신하고 있으며, PC에서의 응용제품을 개발하려면 반드시 고속의 PCI 버스에 연결을 시켜야 의미가 있게 된다.[1,2,4]

PCI(peripheral component interconnect)는 초고속 프로세서(highspeed process)나 주변제어기(peripheral controller)의 로칼버스(local bus)로 차세대의 고성능 컴퓨터의 로칼버스 요구조건을 만족하므로, Digital Equipment (DEC Alpha), Motorora(Power PC), Intel(Pentium)등에서도 이 구조를 사용하고 있다.

PCI는 원래 프로세서 마더보드에 있는 고속장치 사이의 통신을 목적으로 하고 있다. 그러고 앞에 열거한 Pentium, Power PC, DEC Alpha등은 주변기기와 직접 연결(interface)하기를 원한다. 따라서 최근의 PCI는 I/O(입출력) 기기와의 연결에 관심이 있고, 사용자가 설계한 PC 응용제품은 PCI버스에 직접 연결되어야 원하는 시스템 성능을 낼 수 있게 된다. 또한 PCI는 고속 버스일 뿐 아니라 자동 컨피규레이션(autoconfiguration)에 해당하는 컨피규레이션 영역 레지스터 블록(configration space register block)을 포함하고 있어, 고성능 주변기기(SCSI, IDE, LAN, graphic, video)등에 별도장치 없이 인

터페이스를 제공하고 플러그 앤드 플레이도 가능하게 한다. 일반적으로 메모리와 중앙처리장치의 속도는 빠르고, 주변기기의 속도는 느리며, 이를 연결해주는 버스가 고속이어야 시스템의 성능을 제대로 발휘할 수 있으므로 PC 관련 응용제품을 개발하면 무엇보다도 이 PCI 버스에 연결을 시켜야만 의미가 있게 되는 것이다. 이런 관점에서 PCI 인터페이스 회로가 필요하며 여기서 고려한 PCI 인터페이스는 사용자장치가 주는 데이터를 PCI 인터페이스를 통하여 PCI 로칼 버스상에 데이터를 기록하거나 버스의 데이터를 읽을 수 있는 기능에 초점을 두어, 사용자 측에서의 인터페이스로 READY와 READ/WRITE 스트로브 핸드쉐이크 시컨스를 갖는 일반적인 32비트 디바이스를 설계한 것이며, 24비트의 사용자 디바이스 어드레스를 또한 제공한다. 그리고 33Mhz의 PCI 타이밍을 만족하도록 설계하였다.

논문의 구성은 제 1장 서론에 이어 제 2장 본론에서, PCI와 설계된 내용, 제 3장에서는 시뮬레이션 내용을 설명하고 제 4장에서 결론을 내렸다.

2. 본 론

2.1 PCI 시스템

Fig. 2.1은 전형적인 PCI시스템을 보여준다. CPU는 PCI 어뎁터를 통하여 PCI 버스에 연결되어 있고, 사용자 장치로 비디오 데이터가 입력되고, PCI 인터페이스를 통하여 PCI 버스에 연결되며, 초고속의 디스크와 비디오 컨트롤러에 전송할 수 있는 시스템으로, 본 논문에서는 PCI 인터페이스부를 설계하였다.

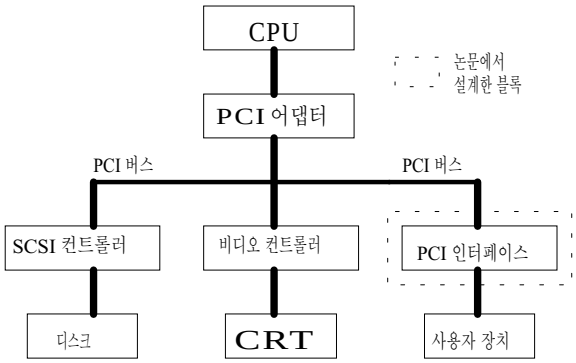

Fig. 2.1 PCI system

PCI는 3개의 어드레스 영역을 정의한다. PCI 하드웨어 컨피규레이션을 제공하도록 정의된 컨피규레이션 영역, 실제 데이터 전송에 있어서 디바이스에 의해 사용되는 메모리와 I/O 영역이다.

PCI는 OS 소프트웨어에 의해 컨피규레이션 영역을 읽기도 하고, 기록도 하여 초기화 한다. PCI 디바이스는 이 목적을 위하여 256바이트의 컨피규레이션 레지스터가 할당되어 있다. (모든 바이트가 전부 실제 로직으로 구현되어야 하는 것은 아니다. 정의되지 않은 레지스터는 읽었을 때 제로값을 돌려주면 된다.) 컨피규레이션 영역을 억세스 하는 것은 각각의 디바이스에 대하여 칩선택 신호인 IDSEL 제어핀을 통하여 외부 어드레스 디코딩을 요구하게 된다.

PCI에서 기본적인 버스 전송 메카니즘은 하나의 어드레스 위상과 여러개의 데이터 위상으로 구성된 버스트 전송이다. FRAME# 제어신호가 어서트(assert)되는 첫 번째 클록 사이클

이 어드레스 전송의 타이밍이 된다. FRAME#은 계속되는 데이터 타이밍에서도 그대로 액티브(active) 상태이고, 마지막 버스트 데이터 전송의 시작에서 디어서트(deassert) 된다. 각각의 데이터 페이즈는 IRDY#(initiator ready, 버스 마스터가 기록될 데이터를 공급하거나, 데이터를 읽을 준비가 되었을 때), TRDY#(target ready, 타겟 디바이스가 기록될 데이터를 받거나, 읽을 데이터를 공급하는)를 포함한다.

PCI에 연결된 디바이스는 버스 마스터가 요구하는 어떤 종류의 동작인가를 지시하는 버스 커맨드를 수행한다. 버스 커맨드는 버스 동작의 어드레스 페이즈동안 4 비트 CBE# 라인을 인코딩한다. 데이터 페이즈 동안에는 CBE# 라인은 워드 전송에 대해서 바이트 인에이블 신호이다.

PCI의 최대 버스전송률은 매 30ns당 32 비트 워드 혹은 132MB/sec (33Mhz*32bits/8bits)이다. 기본적인 PCI의 리드 라이트 동작은 Fig. 2.2 Fig. 2.3과 같다.

Fig.는 설계할 PCI 인터페이스와 PCI 버스, 그리고 사용자 디바이스 사이의 외부 신호를 나타낸다.

PCI〔0:31〕	멀티플렉스된 어드레스/데이타 버스
CBE#〔0:3〕	멀티플렉스된 버스 커맨드와 바이트 인에이블
PARITY	PCI〔0:31〕과 CBE#〔0:3〕의 짝수 페리티
FRAME#	데이터 프레임의 시작과 기간을 나타낸다.
IDSEL	컨피규레이션 동작동안의 칩선택신호로 사용
IRDY#	버스 마스터가 데이터 전송의 준비기 되어 있음을 나타냄
RST#	마스터 리셋 신호
CLK	마스터 클록 신호
DEVSEL#	어드레스가 디코드 되었음을 나타내는 인터페이스 FPGA에 의해 드라이브되는 신호
TRDY#	데이터 전송의 준비가 되었음을 나타내는 인터페이스 FPGA에 의해 드라이브되는 신호
STOP#	전송(버스트)을 포기하겠다는 의사를 나타내는 인터페이스 FPGA에 의해 드라이브 되는 신호

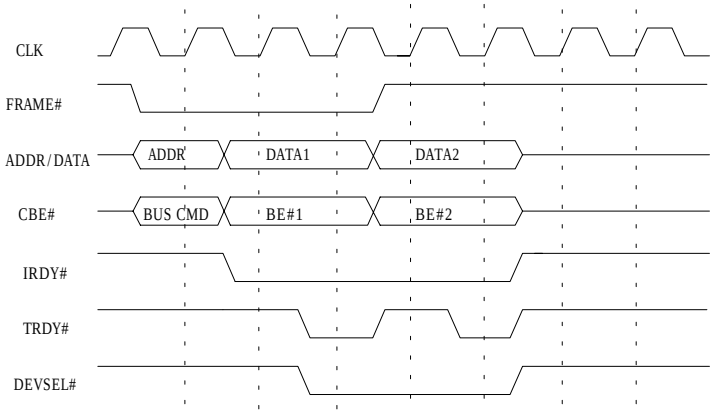

Fig. 2.2 PCI write operation

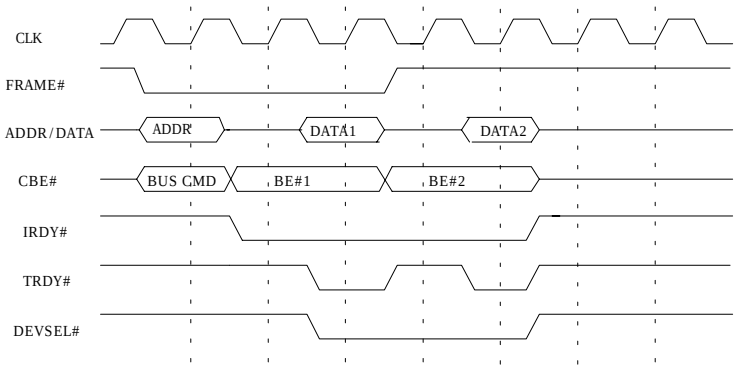

Fig. 2.3 PCI read operation

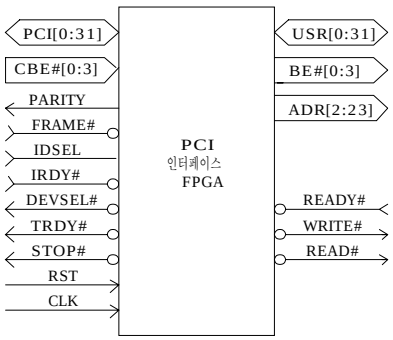

Fig. 2.4 Interface circuit i/o signals

인터페이스에 의해 드라이브되는 3개의 PCI 제어신호(DEVSEL#, TRDY#, STOP#)는 PCI 특별그룹에 의해 트라이스테이트(tristate)를 유지하도록 되어 있다. 이것은 Low(active)가 된 후에 FPGA가 플로팅(floating) 되기 전에 최소한 한 클록동안 High (inactive)를 유지하도록 드라이브한다.

사용자 디바이스와 인터페이스 되는 신호는 다음과 같다.

USR〔0:31〕	양방향성 사용자 데이터 버스
BE#〔0:3〕	라이트 리드 동작에 대한 사용자 바이트 인에이블
ADR〔2:23〕	사용자 어드레스 버스
READY#	사용자 디바이스가 데이터 전송의 준비가 되었음을 나타냄
WRITE#	사용자 라이트 스트로브
READ#	사용자 리드 스트로브

2.2 PCI 시스템 설계

PCI 인터페이스의 설계사양은 다음과 같다.

① 한 개의 FPGA로 일반적인 32 비트의 사용자장치와 완벽한 PCI 인터페이스

② 완전한 PCI 기준을 만족

- 컨피규레이션 스페이스 레지스터 파일

- 베이스 어드레스를 통한 디바이스 어드레싱
- 핸드쉐이킹 데이터 전송

③ 사용자장치에 대한 16MB 디멀티플렉스된 어드레스 라인
④ 사용자장치와 핸드쉐이크를 하는 간단한 READY, READ/WRITE 스트로브
⑤ 리드 라이트에 대한 고속 버스트 전송
⑥ 특별히 사용자의 요구에 만족시키기 위한 모듈 형태로 수정이 쉽도록 한다.

회로도 Fig. 2.5와 같이 11개의 모듈로 구성되어 있고, 최상위계층으로 Fig. 2.4와 같다. Fig. 2.5를 기준으로 설계된 각 모듈의 동작에 대하여 설명한다.

모듈 : dpads(PCI data bus pads)

최상위 계층 탑 레벨에서 dpads(PCI data bus pads)는 PCI 인터페이스 모듈(PCIIFACE)과 연결되며, PCI 인에이블 신호(PCI_ENABLE)에 따라 32비트의 PCI[0:31] 신호의 입력 혹은 사용자 장치로 부터의 어드레스 데이터를 PCI 버스에 올려 놓기 위한 출력 패드로 양방향성 트라이스테이트 패드로 구성하였다.

모듈 : PCI 인터페이스(PCI interface)

모듈 PCI INTERFACE(piface)는 32비트의 PCI 멀티플렉스된 어드레스/데이터를 PCI 버스와 사용자장치 사이를 인터페이스한다. PCI 버스로 부터의 데이터는 dpads(PCI data bus module)를 지나 piface(PCI interface)모듈에 입력된다. 입력된 신호는 다른 블록으로 가기전에 먼저 클록에 의해 동작하는 레지스터에 입력되어, PCI 데이터의 셋업 스펙(7ns)을 만족시키도록 한다.

또한 이 모듈은 PCI 읽기 동작시 32비트의 페리티 발생 블록을 포함한다. 발생된 페리티는 관련된 PCI_ENABLE 신호에 따라 PCI 버스로 출력되거나, PCI 버스로 부터 패리티 입력 패드로 작용한다. 그리고 발생된 패리티는 내부 리지스터에 저장되므로 한 클록 지연된다. RESET 신호는 파우워 온 후에 PCI신호가 드라이브 되지 않도록 한다.

모듈 : 사용자 인터페이스(User interface)

사용자 인터페이스(uiface) 모듈은 32비트 사용자 데이터 버스를 인터페이스한다. 이 모듈은 USR_ENABLE 신호에 의해 PCI[0:31] 데이터를 사용자 장치로 혹은 사용자 장치로 부터의 데이터를 PCI[0:31]로 보내는 양방향성 트라이스테이트 패드로 구성되어 있다.

모듈 : Configuration Space Register(Confspace)

CONFSPACE 모듈은 PCI에서 요구하는 컨피규레이션 스페이스의 레지스터를 하드웨어적으로 구현한다. 어드레스 비트에 의해 제어되는 4 to 1 멀티플렉서에 의해 컨피규레이션 읽기 동작시 CFG[0:31]의 컨피규레이션 영역의 데이터를 선택한다. 이 멀티플렉서의 출력은 구현되지 않은 영역의 레지스터를 읽을 때 제로가 읽히도록 해 준다. 또 커맨드(COMMAND[0:15])와 배이스 레지스터(BASE[0:31])는 다른 모듈에서 사용하기 때문에 별도의 버스를 통하여 전송된다.

신호 BVALID는 OS 소프트웨어에 의한 베이스 어드레스가 컨피규레이션 레지스터의 베이스 레지스터에 입력될 때 BVALID 신호가 출력된다. 파우어 온시 RESET 신호는 BVALID가 "False"가 되도록 한다.

컨피규레이션 레지스터중 7개를 구현하였는데, 구현된 컨피규레이션 해더 레지스터 중 4개는 밴더, 디바이스, 리비젼, 클레스 ID로 읽기만 가능하다. 이 레지스터의 골격을 쉽게 목표 디바이스에 맞게 수정할 수 있다.

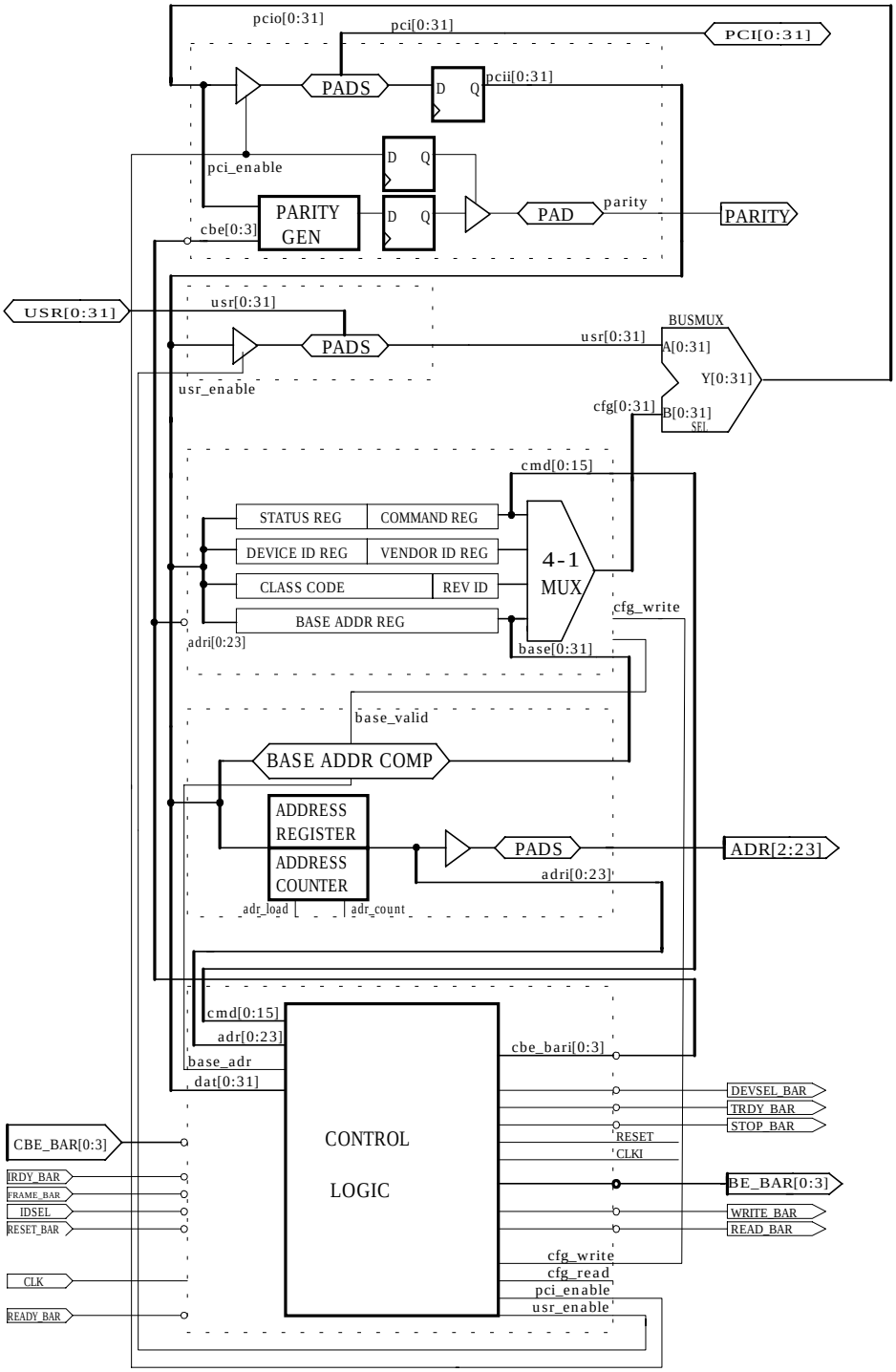

Fig. 2.5 PCI interface circuit

다음 레지스터의 내용을 간단히 설명한다.

Vendor ID	PCI SIG에 의해 허용된 디바이스 제작자를 확인
Device ID	벤더에 의해 허용된 특별한 디바이스 형태를 확인
Revision ID	벤더에 의해 허용된 리비젼 확인
Class Code	일반적인 디바이스 기능을 확인

나머지 3개의 레지스터는 읽기와 쓰기가 가능하고 다음 기능을 제공한다.

Command	PCI 사이클에서 응답하는 디바이스의 동작을 제어
Status	PCI 버스관련 상태 정보를 기록
Base	디바이스의 베이스 어드레스(O/S 에 의해 정해짐)를 규정

커맨드 레지스터는 파우어 온 리셋트시 컨피규레이션 영역의 읽기와 쓰기를 제외한 모든 PCI 동작으로 부터 디바이스를 논리적으로 분리하도록 클리어된다. 여기서 커맨드 레지스터의 비트 0와 1이 I/O와 메모리 억세스인 경우의 디바이스 응답을 제어하도록 하였다. 모든 다른 커맨드 레지스터 비트는 제로값을 갖고 있다.

상태 레지스터의 비트[10:9]는 디바이스가 자신의 PCI 어드레스를 디코딩하는 속도를 나타낸다. PCI 스펙은 Fast, Medium, Slow 의 3가지 응답을 제공한다. 여기서는 Medium 속도의 디코드를 사용하였다.

베이스 레지스터는 디바이스의 메모리 혹은 I/O 스페이스 요구를 결정하도록 OS에서 사용된다. 여기서 사용자장치는 16MB 어드레스 스페이스 (24 비트) 라고 가정하고, 상위 8비트의 베이스 어드레스만 읽기/쓰기가 가능하도록 구성되고, 하위 24 비트는 읽기 동작시 항상 제로값을 리턴 하도록 되어있다. 만약 디바이스에서 허용되는 어드레스 스페이스를 확장하거나, 줄일 경우 베이스 레지스터와 관련된 어드레스 비교기를 맞게 수정하면 된다.

모듈 : 버스 멀티플렉스(BUSMUX)

BUSMUX는 PCI 읽기 동작시 사용자 장치로 부터의 데이터 버스(USR[0:31])와 컨피규레이션 영역의 데이타 버스(CFG[0:31])를 선택하여 PCI[0:31]로 출력시킨다.

모듈 : 어드레스 인터페이스(Address interface)

어드레스 인터페이스 모듈은 버스 동작중 어드레스 위상에서 멀티플렉스된 어드레스/데이타 버스에서 어드레스를 로드한다.(ADRREG) 이 레지스터의 [2:9]비트는 버스트 동작시 어드레스 자동 증가를 위해 업 카운터로 구현하였다.

이 모듈은 또한 PCI 어드레스중 컨피규레이션 영역의 베이스 레지스터의 내용과 비교하는 베이스 어드레스 비교기를 포함한다. 비교기의 어드레스 입력은 빠른 어드레스 디코드를 위하여 어드레스 레지스터가 아니라 버스 인터페이스로 부터 직접 입력 된다. 따라서, 어드레스 로딩과 어드레스 디코딩은 연속적인 사이클이 아니라, 같은 클록 사이클에서 일어난다. 어드레스가 일치할 경우 BVALID 신호가 있을 때 어드레스 히트(address hit)라고 하며, Base_Adr가 셋트된다.

모듈 : PCI 컨트롤 패드입력(cipads)

PCI 컨트롤 패드 입력 모듈은 CBE#[0:3], irdy#, frame#, idsel, rst#, clk 등 제어에 필요한 신호의 입력 패드를 포함한다.

모듈 : 사용자 제어 입력 패드(uipadsl)

이 모듈은 사용자 장치와 통신을 하기위한 사용자로부터의 ready# 신호가 입력되는 패드를 포함한다.

모듈 : PCI 제어신호 출력 패드(copads)

PCI 인터페이스로 부터 버스로 출력되는 신호를, devsel#, trdy#, stop#를 포함한다.

모듈 : 사용자 제어신호 출력 패드(uopads)

PCI 인터페이스로부터 사용자 디바이스를 제어하기 위한 신호 be#[0:3],write#, read# 신호를 출력하는 패드를 포함한다.

모듈 : 제어(Control)

제어(CONTROL) 모듈은 PCI와 사용자 인터페이스의 외부 핸드쉐이크 신호, 내부 제어신호등 각 모듈에서 사용하는 모든 제어 신호를 만든다. 제어 모듈은 4개의 서브 모듈로 구성되어 있다.

① CBE#[0:3] 버스 명령 디코더 (CBEDECODA)

CBE#[0:3]을 디코드 하도록 하였고, Frame# 신호와 한 클록 지연된 Frame_dly 신호, 그리고 적절히 CBE#[0:3] 신호를 이용하여 컨피규레이션, 사용자 장치의 읽기, 쓰기 명령등 4개의 신호(cfgwr_cmd, cfgrd_cmd,usrrd_cmd, usrwr_cmd) 에 대해서만 설계하였다.

② PCI 제어(PCI control block : PCICNTRL)

frame#과 한 클록 지연된 frame_dly 신호를 앤딩하여, PCI 동작에서 어드레스 읽기 위상의 신호 adr_load 신호를 만든다. cfg_read 혹은 usr_read인 경우 trdy신호는 어써트되고, frame# 구간일 때 adr_count 신호를 만든다. trdy 신호와 frame_dly신호를 이용하여, PCI 동작의 마지막을 나타내는 last_cycle 신호를 만든다.

③ 컨피규레이션 제어(Configuration control module : CFGCNTRL)

컨피규레이션 읽기와 쓰기 신호 cfg_write, cfg_read 신호를 이용하여, cfg_read, cfg_write, cfg_ready 신호 등을 만든다.

④ 사용자 제어(User control block : usrcntrl)

userrd_cmd, usrwr_cmd 신호를 이용하여, usr_read, usr_write, usr_ready 신호를 만든다.

3.시뮬레이션

ACTEL FPGA 셀로 설계된 회로를 Viewlogic 시뮬레이터를 이용하여 로직 시뮬레이션을 한 후 ACTEL 1425A 칩으로 배치 배선을 한후 최종적으로 지연 파라미터에 의한 시뮬레이션으로 동작을 확인 하였다. 확인은 4가지 파형에 대하여 확인하였다.

컨피규레이션 레지스터 쓰기 동작

타이밍 1 : PCI 컨피규레이션 쓰기 동작은 클록 엣지 t0에서 시작한다.

FRAME#	frame의 시작을 나타내는 신호를 "L"로 어서트
PCI[0:31]	어드레스 값 포함
IDSEL	컨피규레이션에 대한 디바이스가 선택되었음을 나타냄
CBE#	컨피규레이션 쓰기 버스 커맨드 를 가리킴

CONTROL 모듈은 위의 조건을 디코드하여, adr_load신호를 만들어, 어드레스 값을 어드레스 레지스터/카운터에 로드시칸다. cfg_write_cmd는 컨피규레이션 쓰기 명령이 받아들여졌다는 것을 가리킨다.

클록 엣지 t1에서, FRAME#은 버스트 데이터가 있다는 것을 나타내도록 디어섯트 상태를 유지한다. 이 조건은 adr_count를 클록 엣지 t2에서 발생시켜, 다음 데이터 워드에 대하여 어드레스 레지스터를 자동으로 증가시킨다.

Fig. 3.1은 버스트 컨피규레이션 쓰기 동작으로 FRAME# 은 클록 t5에서 디어섯트되어, 버스트의 마지막 데이터 워드가 다음에 있다는 것을 가리킨다. 클록 엣지 t7은 프레임의 끝을 나타내고, 그때 DEVSEL# 과 cfg_write_frame은 클리어된다. 신호 sts_enable 은 전처럼 클록 엣지 t8까지 남아 있다.

타이밍 2 : 컨피규레이션 레지스터 읽기 동작

이 동작은 CBE# 버스 커맨드가 신호 cfg_read_cmd 로 디코드되는 것을 제외하고, 컨피규레이션 쓰기와 비슷하다. IRDY#를 어섯트 하는 것은 PCI 버스 마스터가 데이터를 받아들일 준비가 되어 있다는 것을 가리킨다.

복수의 데이터가 있다는 것을 나타내도록 클록 엣지 t1을 지나서도 FRAME#은 디어서트 상태로 남아있다. 클록 엣지 t2에서, adr_count신호가 발생되고 다음 워드 전송을 위하여 어드레스 카운터를 동작시킨다.

Fig. 3.2의 타이밍 2에서는 클록 엣지 t3에서 FRAME#이 디어섯트되어 다음 마지막 워드가 있다는 것을 가리킨다. 이것은 t4에서 TRDY#신호 t5에서 DEVSEL#, pci_enable, cfg_read 신호를 디어섯트 하도록 한다.

타이밍 3 : 사용자장치 쓰기동작

PCI 메모리(User) 쓰기 동작은 클록 엣지 t0에서 다음 PCI 인터페이스 조건에 따라 시작된다.

FRAME#	프레임의 시작을 나타내는 신호로 어섯트
PCI[0:31]	어드레스 값을 포함
CBE#	메모리 쓰기 명령

CONTROL모듈은 이 조건을 디코드하여, adr_load를 만든다. 그리고 어드레스 값을 로드하여, 어드레스 레지스터/카운터 에 입력한다. usr_write_cmd 는 메모리 쓰기명령이 받아들여 졌나를 가리킨다.

버스트 컨피규레이션 쓰기 동작에서는 FRAME# 신호가 버스트 조건을 가리키도록 클록 엣지 t1에서 어섯트 상태로 남아있다. 클록 엣지 t2에서, 첫 번째 데이터에 대하여 WRITE#와 TRDY# 신호가 발생된다. 마찬가지로 adr_count는 카운터를 한주기 진행시킨다. 사용자 디바이스는 첫 번째 데이터 워드를 처리하도록 클록 엣지 t3에서 한 주기 동안 READY# 신호를 디어섯트 한다.

READY#신호를 다시 어섯트 하는 것은 사용자 인터페이스가 다음 데이터 워드에 대해 준비가 되어 있음을 가리키고, 두 번째 WRITE#, TRDY#를 허용한다. adr_count 스트로브는 클록 엣지 t5에서 발생한다. 타이밍 6에서 FRAME#이 디어섯트 되는 것은 다음이 마지막 워드임을 나타낸다. 클록 엣지 t8에서 마지막 스트로브가 발생한다. 한번 더 사용자 디바이스의 READY# 상태를 나타낸다.

타이밍 4 : 사용자 장치 읽기 동작

이 동작은 단일 사용자 장치 쓰기와 CBE# 버스 커맨드는 usr_read_cmd 에 의해 디코드되는 것을 제외하고는 유사하다. FRAME# 신호는 한 워드 이상이 있음을 가리키도록 클록 엣지 t1이 지나도 어서트 상태로 남아있다. 클록 엣지 t4에서 adr_count 는 만들어져 다음 버스트 워드를 위하여 어드레스 카운터를 증가 시킨다.

클록 엣지 t5에서, READ# 유저데이타 스트로브는 첫 번째 데이터가 FPGA에 의해(그리고 PCI 버스) 받아들여 졌음을 사용자 디바이스에게 알리도록 점차적으로 디어섯트 된다. 이 예에서 FRAME#은 버스트의 마지막 워드가 요구되었다는 것을 가리키도록 디어섯트된다.

클록 엣지 t6에서 READ# 스트로브의 리어섯트는 사용자 디바이스에게 두 번째 리드데이

타

워드의 펫칭을 시작하도록(첫 번째와 같은 방법으로) 지시한다. 클록 엣지 t10에서 지연된 sts_enable 신호만을 제외하고 모든 신호를 디어섯트하여 끝낸다.

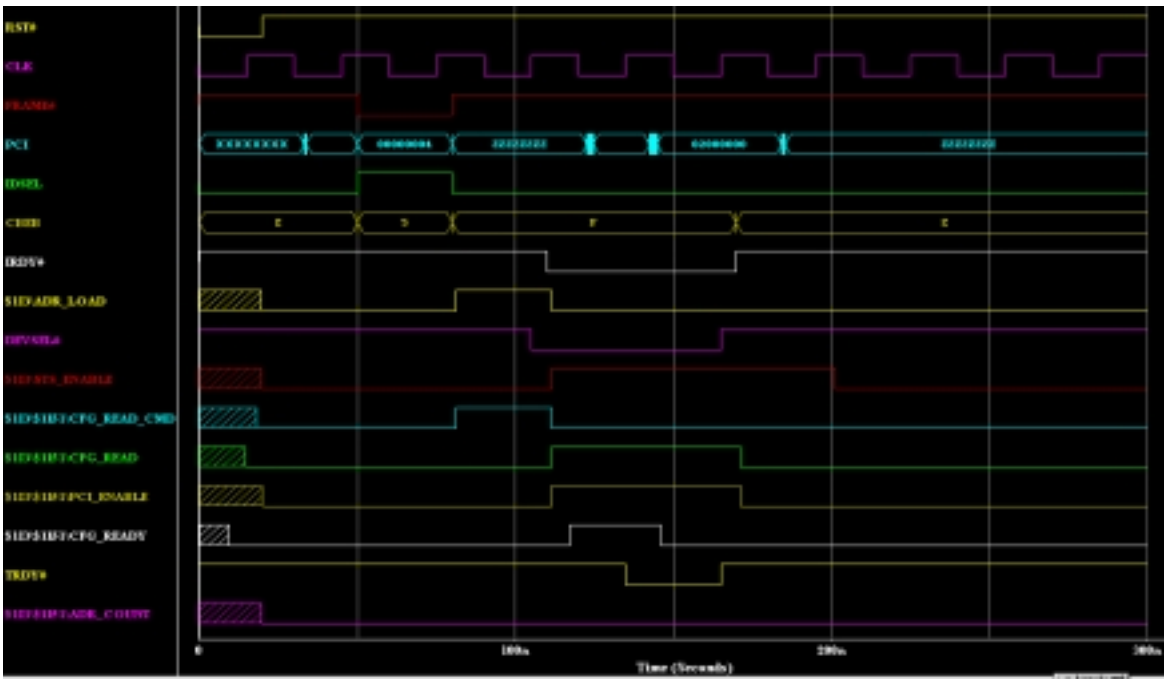

Fig. 3.1 Timing 1

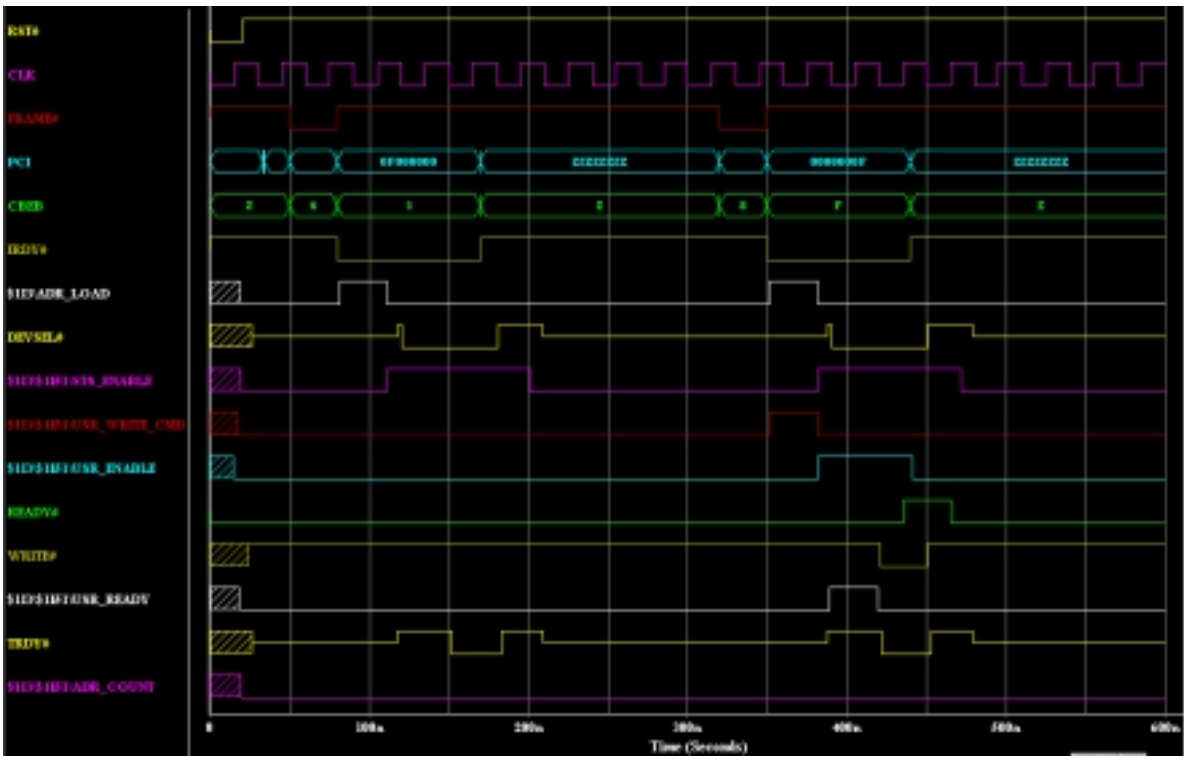

Fig. 3.2 Timing 2

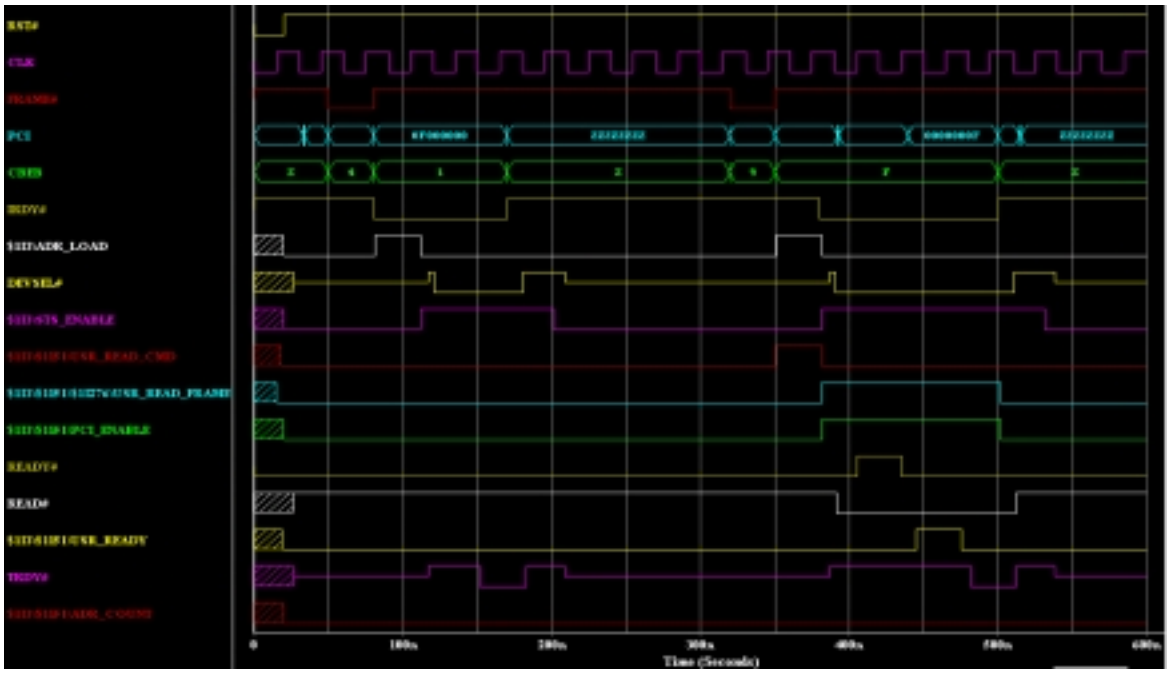

Fig. 3.3 Timing 3

Fig. 3.4 Timing 4

4 결 론

본 논문에서는 PC 응용제품을 PC에 의미있게 연결하기 위해서는 반드시 PCI 버스에 직접 연결이 되어야 하므로, 응용제품과 PCI 버스 사이의 인터페이스 회로를 ACTEL FPGA 회로로 구현 하였고, 시뮬레이션으로 동작을 하였다. 향 후 PC 응용제품을 개발함에 있어 기 구현된 PCI 인터페이스 회로와 함께 FPGA상에서 시스템 동작 확인을 할 수 있고, 궁극적으로는 응용제품과 PCI 인터페이스 회로를 함께 ASIC화가 가능하여 경쟁력 있는 IC개발의 핵심요소가 될 수 있다.

참고문헌

(1) Philips, 1995, "*Desktop Video Data Handbook*",Philips Semiconductors, Vol. 1. pp. 3-185~3-201.

(2) 장기혁 편역, 1986, "영상처리 시스템의 기초와 설계 제작", 도서출판 세운. pp.150~ 180.

(3) 오재광 저, 1995, "PC 인터페이스 제작과 실제", 크라운 출판사, pp. 120 ~ 150.

(4) 이주석, 1997, " DRAM을 사용한 가변사이즈 영상/저장 시스템 구현에 관한 연구", 한국통신학회 논문지, Vol.22, No.6, pp. 1185 ~ 1194.

(5) Wang, K. and Bryant, C. 1995, ″ Designing the MPC105 PCI Bridge/Memory Controller″. IEEE MICRO, Vol.15, No.2, pp. 44 ~ 49.