PCI 2.2 타겟 컨트롤러 설계 및 검증

서경호*, 최은주*, 서강덕*, 현유진*, 성광수***
*영남대학교 전자공학과 집적회로연구실
**영남대학교 전자정보공학부
e-mail:braham@yumail.ac.kr

Design and Verification of PCI 2.2 Target Controller

Kyung-Ho Seo*, Eun-Ju Choi*, Kwang-Duck Seo*,
Eugin Hyun*, Kwang-Su Seong**
*VLSI Lab, Dept of Electronic Engineering, Yeungnam University
**Dept of Electrical & Computer Science, Yeungnam University

요 약

PCI 2.2 마스터 디바이스가 타겟 디바이스로부터 데이터를 읽어 오고자 할 때 타겟 장치는 내부적으로 데이터를 준비해야 함으로 인해 PCI 버스가 데이터 전송 없이 점유되는 상황이 발생한다. 이를 위해 PCI 2.2 사양에서는 지연전송을 제안하여 전송 효율을 향상시켰지만 이 역시 타겟 디바이스가 얼마의 데이터를 미리 준비 해둘지를 알 수 없어 인해 버스 사용 및 데이터 전송 효율을 떨어뜨리는 원인을 제공한다. 본 논문에서는 먼저 이를 해결하기 위한 새로운 방법을 제안한다. 그리고 이 방법을 지원하는 PCI 타겟 컨트롤러와 로컬 디바이스를 설계하였다. 설계되어진 PCI 타겟 컨트롤러는 PCI 2.2를 전혀 모르는 사용자도 쉽게 PCI 인터페이스를 지원할 수 있도록 한 프로토콜 변환기로 사용될 수 있다. PCI 타겟 컨트롤러와 로컬 디바이스는 먼저 행위 모델로 설계하였으며 또한 이들을 검증하기 위한 테스트벤치를 설계 하였다. 이를 통해 제안되어진 방법의 성능을 측정하였으며 후에 다시 실제 하드웨어로 설계하였다. 설계되어진 하드웨어를 효과적으로 검증하기 위해 참조모델, 랜덤발생기, 비교엔진으로 구성된 랜덤 테스트 환경을 제안하였다. 이 검증 환경에서 수행된 결과를 비교함으로써 일반적인 테스트 벡터에서 발견하기 어려운 에러들을 발견할 수 있었다.

1. 서론

지난 10년간 PCI 2.2는 입출력 인터페이스의 표준안으로 자리 잡아왔다. 하지만 근래에 들어와서 주변장치들이 고성능이 됨으로 인해 보다 고속 데이터 전송이 가능한 입출력 표준안의 필요성이 대두되었다[1-7]. 이에 PCI SIG는 2000년 PCI-X를 제안하여 현재 서버나 워크스테이션 등에 사용되어지고 있다[8]. 그러나 아직 대부분의 PC는 PCI 2.2 버스를 지원하고 있으며 셋탑박스, 통신 시스템, 실시간 시스템 등 다양한 분야에서도 아직 응용되어지고 있다. 그리고 PCI-X 역시 PCI 2.2 프로토콜도 함께 지원하도록 규정되어 있다[8]. 따라서 PCI 2.2 버스의 성능이 전체 시스템의 성능에 많은 영향을 미칠 수 있다.

PCI 2.2 마스터 디바이스가 타겟 디바이스로부터 데이터를 읽어 오고자 할 때 타겟 장치는 내부적으로 데이터를 준비해야 함으로 인해 PCI 버스가 데이터 전송 없이 점유되는 상황이 발생한다. 이를 해결하기 위해 PCI 2.2 사양에서는 지연전송(Delayed transaction)을 제안하여 전송 효율을 향상시켰다[8]. 하지만 이 방법 역시 타겟 디바이스가 얼마의 데이터를 미리 준비 해둘지를 알 수 없어 버스 사용 및 데이터 전송 효율을 떨어뜨리는 원인을 제공한다.

본 논문에서는 먼저 이를 해결하기 위한 새로운 방법을 제안한다. 그리고 이 방법을 지원하는 PCI 타겟 컨트롤러와 로컬 디바이스를 설계하였다. PCI 타겟 컨트롤러와 로컬 디바이스는 먼저 행위 모델로 설계하였으며 또한 이들을 검증하기 위한 테스트벤치(test-bench)를 설계 하였다. 이를 통해 제안되어진 방법의 성능을 측정하였으며 후에 다시 이들은 실제 하드웨어로 설계하였다. 설계되어진 하드웨어를 효과적으로 검증하기 위해 참조모델(Reference Model), 랜덤발생기(Random Generator), 비교엔진

(Compare Engine)으로 구성된 랜덤 테스트 환경을 제안하였다.

2. PCI 2.2 개요

버스 마스터가 타겟 장치에 메모리 읽기 명령을 요구한 경우 타겟 장치는 데이터를 전송해주기 위해서 내부적으로 데이터를 준비할 시간이 필요하다. 특히 지역 버스를 가지는 PCI 장치인 경우 로컬 디바이스로부터 데이터를 읽어와야 할 시간이 필요하기 때문에 그 동안 아무런 데이터 전송 없이 PCI 버스를 점유하고 있어야 한다. 만약 지역 버스가 느리게 동작하는 경우라면 데이터를 전송하기 위해 PCI 버스를 장시간 점령해야하기 때문에 버스 사용 효율은 떨어지게 된다.

PCI 2.2 사양에서는 이렇게 내부적으로 데이터 준비 시간이 오래 걸리는 타겟 장치를 위해 그림 1과 같이 지연전송 메커니즘을 제안하고 있다[7]. 그림 1(a)에서 보면 먼저 버스 마스터 A가 타겟 장치 B에 메모리 읽기 명령을 요구하면 타겟 장치 B는 아직 데이터 전송을 할 수 없음을 알리기 위해 리트라이 종료로 프로토콜을 종료한다. 여기서 리트라이란 PCI 타겟 디바이스가 즉시 데이터를 전송 할 수 없을 때 프로토콜을 종료하는 방법으로 이 후 마스터 디바이스는 꼭 다시 데이터 전송을 요청을 해야 한다. 그 후 타겟 장치 B는 기억 해둔 어드레스를 이용하여 자체적으로 데이터를 준비하기 시작한다. 만약 타겟 장치에 의해 전송 할 데이터가 준비된 후에 버스 마스터 A가 똑같은 명령을 다시 요구해 오면 그림 1(b)과 같이 데이터를 전송해준다.

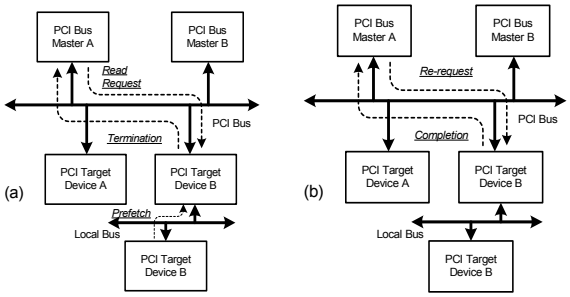

Figure 1. The Delayed Transaction Mechanism

이러한 지연전송은 타겟 장치 B가 버스 마스터 A에 의해 요청된 데이터를 자체적으로 준비를 하는 동안 버스 마스터 A가 다른 타겟 장치와 데이터 전송을 하거나 혹은 버스 마스터 B가 PCI 버스를 사용할 수 있기 때문에 PCI 버스의 사용 효율을 향상시킬 수 있다. 따라서 PLX와 같은 대부분의 PCI 디바이스는 지연전송을 지원한다[9]. 이러한 지연전송을 이용하면 효율적으로 버스를 사용 할 수 있음에도 불구하고 이 역시 타겟 장치가 몇 개의 데이터를 미리 읽어 두어야 하는지를 정확히 알 수가 없어 버스 사용 및 데이터 전송 효율을 떨어뜨리는 원인을 제공한다[10].

3. 제안된 방법

제안된 방법은 버스 마스터가 메모리 읽기 명령을 타겟 장치에 요구하기 전에 미리 몇 개의 데이터를 읽어갈지 정확하게 알려줌으로써 타겟 장치가 정확한 양의 데이터를 읽게 하는 방법이다.

그림 2(a)에서 메모리 읽기 명령을 수행하고자 하는 버스 마스터는 먼저 메모리 쓰기 명령을 이용하여 다음에 요구할 메모리 읽기 명령의 어드레스와 읽을 데이터 크기를 *PCR*(Prefech Control Register)에 저장하는데 이를 프리페치 요구라고 한다. 이 *PCR*은 제안된 방법을 지원하기 위해 정의된 내부 컨트롤 레지스터(Control Register)이다. *PCR*에 프리페치를 위한 어드레스와 데이터 크기 정보가 저장되면 타겟 장치는 이를 이용하여 로컬 디바이스로부터 데이터를 프리페치 하여 내부 버퍼에 저장한다. 그림 2(b)에서와 같이 버스 마스터는 곧 메모리 읽기 명령을 타겟 장치에 요구할 것이다. 이때 타겟 장치가 전송할 데이터를 아직 프리페치 하지 못했다면 리트라이 종료로 프로토콜을 종료하고, 만약 전송할 데이터가 준비되었다면 버스 마스터에게 데이터를 전송을 시작하면 된다.

제안된 방법은 타겟 장치가 몇 개의 데이터를 미리 읽어야 될지를 정확하게 알 수 있기 때문에 지연전송에서 발생하는 문제점을 해결 할 수 있다[10]. 즉 프리페치 요구를 이용한 방법은 지연전송에 비해 데이터 전송 효율을 향상시킬 뿐 아니라 PCI 버스와 지역 버스의 사용 효율을 향상시킬 수 있다.

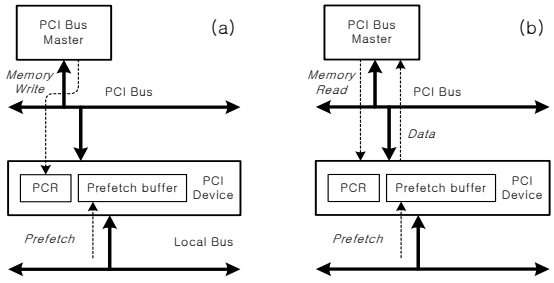

Figure 2. The Proposed Mechanism

4. 설계 및 검증

본 논문에서는 그림 3과 같이 *APCS*라고 명칭을 정한 간단한 로컬 인터페이스를 가지는 PCI 2.2 타겟 컨트롤러를 설계하였다. 설계되어진 *APCS*는 PCI 2.2를 전혀 모르는 사용자도 쉽게 PCI 인터페이스를 지원할 수 있도록 한 프로토콜 변환기이다. 또한 SRAM과 연결되어진 간단한 로컬 디바이스도 설계하였다. 이 로컬 디바이스는 *PCR*과 프리페치 버퍼를 가진다.

설계되어진 *APCS*와 로컬 디바이스는 지연전송과 제안

된 방법을 모두 지원하다. 만약 *APCS*가 마스터 디바이스로부터 메모리 읽기 명령어를 요청 받게 되는 경우 *APCS*는 먼저 *SAD[31:0]*으로 해당 주소와 *SCBE[3:0]*로 명령어를 로컬 디바이스에 전송한다. 또한 이때 *SAREQ*를 드라이브 하여 주소와 명령어가 유효함을 알린다. 이때 로컬 디바이스는 *SSTOP*을 드라이브 하여 데이터를 바로 전송하지 못함을 *APCS*에 알리면 *APCS*는 PCI 프로토콜을 리트라이로 종료 할 것이다. 하지만 로컬 디바이스는 *APCS*로부터 수신한 주소를 이용하여 *SRAM*으로부터 일정량의 데이터를 미리 읽어둔다. 후에 마스터 디바이스는 *APCS*를 통해 다시 똑같은 메모리 읽기 명령어를 요구해 올 것이다. 이때 로컬 디바이스가 프리페치 버퍼에 데이터를 가지고 있다면 *SDONE*를 이용하여 데이터 전송 개시를 알린다. 그러면 *APCS*는 *SDREQ*를 이용하여 데이터 전송 요청을 하고 로컬 디바이스는 *SD[31:0]*을 통해 이들을 전송한다. 이때 *SDONE*를 이용하여 전송하는 데이터가 유효함을 알린다. 만약 마스터 디바이스에 의해 PCI 버스 상에서 프로토콜이 종료하면 *APCS*는 *SLAST* 신호를 통해 로컬디바이스에 이를 알린다. 반대로 로컬 디바이스가 더 이상 데이터 전송을 못하는 경우 *SSTOP*을 이용하여 이를 *APCS*에 알린다. 이상은 로컬 디바이스가 지연 전송으로 응답한 경우이다.

다음은 프리페치 요구가 적용되어진 경우를 들어 보자. 먼저 로컬 마스트 디바이스는 메모리 읽기 명령어를 하기 전에 프리페치 요구를 요청한다. 그러면 *APCS*는 *SAD[31:0]*을 통해 로컬 디바이스의 *PCR* 주소와 프리페치 요구를 위한 정보를 전송한다. 곧 로컬디바이스는 *SRAM*으로부터 데이터를 정확한 양만큼 읽어 올 것이다.

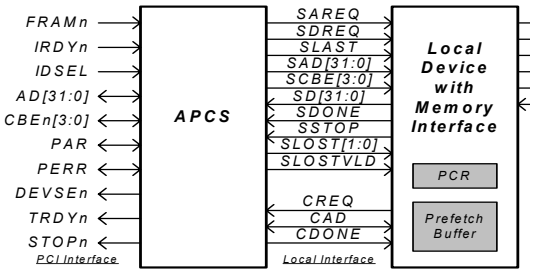

Figure 3. The top block of *APCS* and local device

본 논문에서는 설계 되어진 디바이스의 올바른 동작을 검증하기 위해 먼저 C 언어를 이용하여 *APCS*와 로컬 디바이스의 행위 모델을 설계하였다. 또한 이 행위 모델을 이용하여 제안된 프리페치를 이용한 방법의 성능도 시뮬레이션을 통해 측정하였다. *APCS*와 로컬 디바이스는 다시 Verilog HDL을 이용하여 실제 하드웨어로 코딩되었다.

설계되어진 *APCS*와 로컬 디바이스를 검증하기 위해 PCI 버스 마스터, PCI 버스 아비터, PCI 버스 모니터, 그리고 메모리의 행위 모델들로 구성된 테스트벤치를 그림 4(a)와 같이 설계하였다. 여기서 PCI 버스 모니터는 버스 마스터나 *APCS*가 PCI 프로토콜을 어기는 경우 이를 로그(log) 파일로 만들어 에러를 보고하는 프로토콜 감시자(checker)이다. 그림 4(b)와 같이 *PLI*(Programming language interface)를 통해 연결되어진 각 행위 모델들은 같은 클럭을 기반으로 동작된다. 여기서 *PLI*는 C 언어로 작성된 프로그램과 verilog HDL로 작성된 프로그램을 연결하기 위한 verilog 시뮤레이터(simulator)이다.

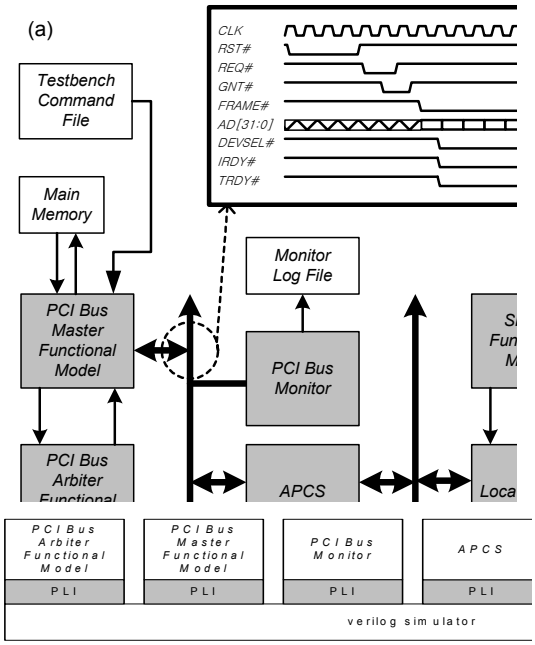

Figure 4. Simulation environment

일반적인 마이크로프로세스는 지원하는 모든 마이크로 명령어에 대해 올바르게 동작하는 지를 확인함으로써 검증을 할 수 있다. 그러나 *APCS*와 같은 컨트롤러인 경우 명령어에 기반을 두고 동작을 하는 게 아니라 프로토콜에 의해 동작을 하기 때문에 보다 유용한 검증 방법이 요구된다. 그래서 본 논문에서는 *APCS*를 효과적으로 검증 할 수 있는 어셈블러 명령어들을 정의 하였다. 이 명령어들은 테스트벤치의 명령어 파일에 기술이 되어지며 이는 사용자에 의해 직접 이루어 질 수도 있고 또는 랜덤생성기에 의해 자동으로 생성되어 질 수도 있다. 이 어셈블러 명령어는 크게 두개의 형태로 나누어진다. 첫 번째는 버스 마스터에 PCI 버스를 통해 데이터 전송을 지시하는 명령어이다. 두 번째는 실제 데이터 전송 시에 발생 할 수 있는 여러 가지 시나리오들을 만들기 위한 파라미터를 정의하기 위한 명령어이다. 예를 들면, PCI 버스나 로컬 버스의 대기 시간을 설정하는 파라미터를 들 수 있다. 몇몇 예제를 아래에 나타냈다.

- PCI_Wait_Max 4;
- PCI_Wait_Min 2;
- MWrite 0x100, 0x300, 84;

먼저 *'PCI_Wait_Max'*와 *'PCI_Wait_Min'*은 로컬 마스터가 데이터를 전송할 때 최대 대기 시간과 최소 대기 시간을 제어하기 위한 명령어이다. 다음의 *'MWrite'*는 PCI 마스터 디바이스가 메모리 읽기 명령어를 이용하여 데이터를 전송하라는 명령어이다. 즉 마스터 디바이스는 먼저 메모리 300h 번지로부터 84개의 데이터를 읽어 타겟 주소 100h 번지와 함께 *APCS*에 전송할 것이다. 곧 로컬 디바이스는 *APCS*로부터 데이터를 수신하여 *SRAM*의 100h 번지에 데이터를 저장 할 것이다.

다음으로 설계되어진 하드웨어를 보다 효과적으로 검증하기 위해 본 논문에서는 참조모델, 랜덤발생기, 그리고 비교엔진을 가지는 랜덤 환경을 그림 5와 같이 제안한다. 여기서 참조모델은 명령어 파일을 처리하기 위한 *APCS*의 동작모델로 클럭에 상관없이 동작되어 진다.

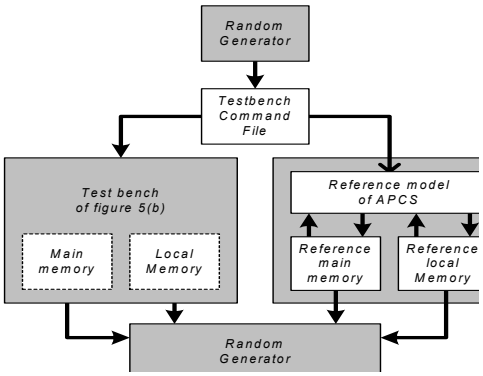

Figure 5. Verification environment for random testing

먼저 랜덤발생기는 데이터 전송을 위한 모든 시나리오를 무작위로 발생시켜 명령어 파일로 생성한다. 그러면 테스트벤치는 이 명령어 파일을 읽어 메모리간의 데이터 전송을 이룬다. 이 명령어 파일은 참조모델에도 똑같이 적용되어진다. 마지막으로 비교엔진은 메인메모리의 데이터와 참조모델의 메인메모리의 데이터를 그리고 SRAM의 데이터와 참조모델의 로컬 메모리의 데이터를 각각 비교한다.

5. 실험결과

*APCS*와 로컬 디바이스에 지연전송이 적용되어진 것에 비해 제안된 방법의 성능이 향상되었음은 앞선 연구[10]에서 시뮬레이션을 통해 증명하였다. 즉 100개의 명령어를 무작위로 선택하여 시뮬레이션 한 결과. 지연 전송에 비해 제안되어진 방법이 평균 10% 성능 향상됨을 알 수 있었다[10].

본 논문에서는 또한 실제로 설계되어진 *APCS*와 로컬 디바이스를 검증하기 위해 10,000개의 명령어를 입력하여 랜덤 환경에서 시뮬레이션 하였다. 표 1은 검증 결과 나타난 오류들을 통계 내어 놓은 것이다.

Table 1. The distribution of error patterns in the *APCS* controller under random testing.

구 분	개수
*APCS*의 PCI 버스 프로토콜 어김	3
*APCS*의 로컬 버스 프로토콜 어김	1
*APCS*의 데이터패스(Datapath) 오류	2
로컬디바이스 에러	2
기타	4
합계	12

6. 결론

본 논문에서는 데이터 전송효율을 향상시키는 새로운 방법을 소개하였으며 또한 이를 지원하는 PCI 타겟 컨트롤러와 로컬 디바이스를 설계하였다. 타겟 컨트롤러와 로컬 디바이스는 먼저 행위 모델로 설계하였으며 또한 이들을 검증하기 위한 테스트벤치를 설계 하였다. 또한 실제 하드웨어로 설계하였으며 이를 효과적으로 검증하기 위해 참조모델, 랜덤발생기, 비교엔진으로 구성된 랜덤 테스트 환경을 제안하였다. 이 검증 환경에서 수행된 결과를 비교함으로써 일반적인 테스트 벡터에서 발견하기 어려운 에러들을 발견할 수 있었다.

References

[1] Edward Solari and George Willse, “PCI hardware and software: architecture and design”, Annabooks. 1998.

[2] Don Anderson and Tom Shabnley, “PCI System Architecture”, Mindshare. 1999.

[3] Bradly K. Fawceet, "Designing PCI bus interfaces with programmable logic", Proceedings of the 8th Annual IEEE International ASIC Conference and Exhibit, pp. 321-324, 1995.

[4] Al Chame, "PCI bus in high speed I/O systems applications", Proceedings of the IEEE Conference on Aerospace, Vol. 4, pp. 505-504, 1998.

[5] E. Finkelstein and S. Weiss, "Implementation of PCI based systems using programmable logic", IEE Proceedings Circuits, Devices and Systems, Vol. 147, No. 3, pp. 171-174, 2000.

[6] http://www.pcisig.com

[7] PCI SIG, “PCI Local Bus Specification Revision 2.2”, PCI SIG. 1998.

[8] PCI SIG, “PCI-X Addendum to the PCI Local Bus Specification Revision 1.0a”, PCI SIG, 2000.

[9] http://www.plxtech.com.

[10] 현유진, 성광수, “PCI 2.2에서 프리페치 요구를 이용해서 데이터 전송 효율을 향상시키는 효과적인 방법”, 대한전자공학회논문지, 제41권, CI편, 제4호, pp. 319-326, 2004년 7월.