PCI 2.2 Data 전송 효율을 향상시키기 위한 메커니즘

현 유 진, 성 광 수
영남대학교 전자공학과
전화 : 053-810-3935 / 핸드폰 : 016-748-5516

Mechanism for Improving Data Rate on PCI 2.2 Interface

Eugin Hyun, Kwang-Su Seong
VLSI Lab. Dept. of Electronic Engineering, Yeungnam University
E-mail : braham@yumail.ac.kr

Abstract

The PCI 2.2 spec introduces Delayed Transaction mechanism to improve system performance for target device with slow local bus. But this mechanism has some restriction since target device doesn't know prefetch data size. So, we propose a new mechanism, which target device prefetch exact data on local bus, to improve data rate on PCI or local interface. The simulation results showed that the proposed mechanism more improves system performance than the Delayed Transaction mechanism.

I. 서론

1981년 IBM 호환 PC 초창기 때부터 사용되어온 ISA/EISA I/O 버스 이후, 1992년 PCI 2.1 표준안이 만들어지면서 10년간 PCI 버스가 I/O 시스템에 사용되어 왔다. PCI 버스는 CPU와 독립적으로 동작하도록 되어 있고, 완전한 32 비트 버스 폭을 지원함으로 인해 데이터의 전송 속도가 빠르다[1][2][3][4].

지금까지 PCI 버스는 Revision 2.3까지 소개 되었으며, 보다 효율적인 버스활용을 위해 PCI X spec이 2000년에 소개되었다[5][6]. 또한 기존의 병렬 전송에서 발생하는 데이터 전송 속도의 한계를 극복하기 위해, 직렬 인터페이스를 이용한 PCI Express라는 표준안이 2002년에 소개되었다[7].

현재 PCI 버스는 IBM PC뿐 아니라 셋톱박스, 통신 시스템, 실시간 시스템, 공장 자동화 시스템 등 다양한 분야에서 응용되고 있다. 특히 보다 효율적인 데이터 전송을 요구하는 서버, 워크스테이션 등에서는 PCI X 인터페이스를 사용하고 있다. 또한 향후 PC는 PCI Express 인터페이스로 사용되어질 전망이다.

하지만 PCI X와 PCI Express 인터페이스를 지원하는 시스템의 경우도 호환성을 위해 PCI 2.2 인터페이스를 지원하여야 한다. 이러한 호환은 전체 데이터 전송 효율을 떨어뜨리게 되는 결과를 낳는다.

본 논문에서는 PCI 2.2 프로토콜에서 전체 데이터 전송 효율을 떨어뜨리는 원인을 분석하고 이를 효과적으로 개선하기 위한 메커니즘을 제안하고자 한다.

II. PCI 2.2 특징[4]

PCI 버스에서는 하나의 버스로 어드레스와 데이터를 전송한다. 먼저 버스 마스터가 어드레스를 전송하면 해당하는 타겟 디바이스가 응답을 하고, 그 다음부터 데이터는 한번 혹은 연속으로 전송된다. PCI에서는 모든 디바이스가 32비트 데이터 버스 폭을 가지며, 확장된 디바이스는 추가적으로 64bit 버스 폭을 가질 수 있다.

PCI 디바이스는 3가지의 어드레스 영역을 가지고 있다. Configuration 어드레스 영역, I/O 어드레스 영역, 메모리 어드레스 영역이다. 이러한 각각의 어드레스 영역에 접근하기 위해 각각의 명령어가 제공되어 진다. 모든 타겟 디바이스는 PCI에서 제공하는 기본적인 명령어를 받아들일 수 있어야한다.

먼저 configuration 어드레스 영역에 접근하기 위한 명령어로, PCI 디바이스에 내장된 configuration 레지스터에 Write 혹은 Read 위한 명령어이다.

두 번째는 I/O 어드레스 영역에 접근하기 위한 명령어로 기존의 I/O 장치와 호환을 위한 명령어이다.

마지막으로 메모리 어드레스 영역에 접근하기 위한 명령어로, 버스 마스터가 특정 메모리 디바이스에 Write를 명령하는 경우 해당 디바이스는 데이터를 받아들이면 되고, Read 명령어인 경우 원하는 데이터를 버스 마스터로 전송해주면 된다. 대부분의 PCI 디바이스는 메모리 명령어를 이용하여 데이터를 전송한다.

III. Delayed Transaction

버스 마스터가 타겟 디바이스에 Read 명령어를 요구한 경우, 타겟 디바이스는 데이터를 바로 전송해주기 위해서 (이를 "Immediate 응답"이라고 한다) 내부적으로 데이터를 준비할 시간이 필요하다. 특히 로컬 버스를 가지는 PCI 디바이스인 경우, 로컬 디바이스로부터 데이터를 전송 받아야 하는 시간이 필요하기 때문에, 그 동안 PCI 버스를 점유하고 있어야 한다. 만약 로컬 버스가 느리게 동작하는 경우라면 한 개의 데이터를 전송하기 위해 PCI 버스를 장시간 점령해야하기 때문에, 버스 사용 효율은 떨어지게 된다. 또한 여러 개의 데이터 전송을 연속으로 요구하는 메모리 Read 명령어인 경우에는 더욱 그러할 것이다. 이는 하위에 다른 PCI 버스를 연결하는 PCI-PCI 브리지인 경우에도 마찬가지이다. 더욱이, PCI 2.2 spec에서는 타겟 디바이스가 장시간 버스를 점유하는걸 방지하기 위해 최소한 8 사이클 내에 데이터를 전송하도록 규정하고 있다. 그렇지 못한 경우에는 즉시 프로토콜을 종료하여야 한다[4].

따라서 PCI 2.2에서는 이렇게 내부적으로 데이터 준비 시간이 오래 걸리는 디바이스의 경우 그림 1과 같이 "Delayed Transaction" 메커니즘을 제안하고 있다[4].

그림 1(a)에서 보듯이 먼저 버스 마스터가 타겟 디바이스에 메모리 Read 명령어를 요구한다(Delayed Request). 그러면 타겟 디바이스는 아직 데이터 전송을 할 수 없음을 알린다(Delayed Response). 하지만 타겟 디바이스는 버스 마스터가 보내 준 어드레스를 이용하여 데이터를 준비하기 시작한다. 만약 전송 할 수 있는 데이터가 준비되었을 때 버스 마스터가 다시 똑같은 메모리 Read 명령어를 요구해 오면 그림 1(c)와 같이 데이터를 전송해준다(Delayed Completion). 이러한 메커니즘은 타겟 디바이스가 내부적으로 데이터를 준비를 하는 동안 버스 마스터는 다른 동작을 하든지, 혹은 다른 버스 마스터가 PCI 버스를 활용할 수 있기 때문에 버스의 효율을 향상시킬 수 있다.

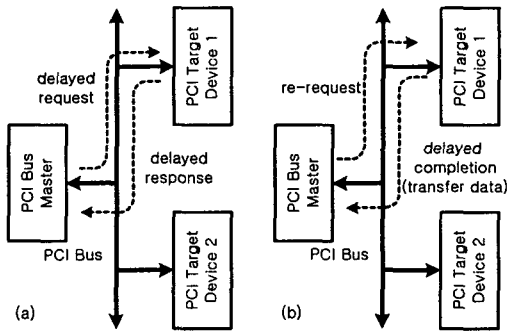

그림 1. Delayed Transaction 메커니즘.

일반적으로 Delayed Transaction을 지원하는 디바이스인 경우, 메모리 Read 명령어를 받은 후 미리 일정 양의 데이터를 준비하게 된다(이를 "Prefetch"라고 한다). 특히 PCI-PCI 브리지는 원래의 마스터 디바이스로부터는 메모리 Read 명령어를 받은 경우에 하위 버스로부터 일정 양의 데이터를 Prefetch하게 된다.

이러한 Prefetch를 이용한 Delayed Transaction이 아주 효율적인 버스 사용을 할 수 있음에도 불구하고 몇 가지 제약이 있다.

먼저 버스 마스터가 읽어가려는 데이터에 비해 타겟 디바이스가 Prefetch 해둔 데이터가 적다면, 프로토콜은 타겟에 의해 즉시 종료하게 된다. 버스 마스터는 다시 버스 사용권을 획득 후, 나머지 데이터 전송을 위해 Delayed Request를 할 것이다. 여기서 버스 마스터가 다시 버스 사용권을 획득하기 위한 시간만큼 데이터 전송 효율은 떨어 질 것이다. 이는 버스 마스터가 많은 양의 데이터를 요구할 때 더욱 자주 그러할 것이다.

반대로 버스 마스터가 Prefetch한 데이터 보다 더 적은 양을 읽고 프로토콜을 종료하면, 타겟 디바이스는 즉시 Prefetch를 종료하고 남은 데이터를 폐기하여야 한다[4]. 왜냐면 버스 마스터가 남은 데이터를 읽어 갈 것이라는 보장을 할 수 없기 때문이다.

여기서 또 하나 고려 해보아야 할 것은, 버스 마스

터가 많은 양의 데이터를 읽어 가려고 하였지만 버스 마스터의 내부적인 문제나 혹은 버스를 장시간 점유할 수 없음으로 인해 불가피하게 프로토콜을 종료한 경우이다. 이 경우 버스 마스터는 분명히 남은 데이터 전송을 위해 다시 메모리 Read 명령어를 요구 해 올 것이다. 하지만 이미 타겟 디바이스는 남은 모든 데이터를 폐기하였기 때문에 Delayed Response로 응답한 다음 다시 Prefetch를 시작하여야 할 것이다. 이는 결국 버스 사용 및 데이터 전송 효율을 떨어뜨리게 되는 것이다.

IV. 제안된 메커니즘

본 논문에서는 이러한 PCI 2.2 버스에서 발생하는 이러한 문제들을 해결하기 위한 메커니즘을 그림 2와 같이 제안하고자 한다.

메모리 Read 명령어를 수행하고자 하는 버스 마스터는 먼저 그림 (a)와 같이 메모리 Write 명령어를 이용하여, 다음에 수행할 메모리 Read 명령어의 어드레스와 데이터 크기를 전송해준다. 이때 메모리 Write 명령어의 주소는 타겟 디바이스 내부에 설계되어진 Prefech Control Register(PCR)의 주소이다. 이 PCR은 타겟 디바이스가 로컬 디바이스로부터 데이터를 Prefech하기 위해 사용되어지는 내부 Control Register 이다. 즉 PCR이 Enable 되면 타겟 디바이스는 PCR에 저장된 어드레스와 데이터 크기 정보를 바탕으로 로컬 디바이스로부터 메모리에 데이터를 Prefetch한다.

버스 마스터는 곧 그림 (b)와 같이 메모리 Read 명령을 요구 할 것이다. 만약 이 시점에 타겟 디바이스의 메모리에 전송할 데이터가 준비되어 있다면 바로 응답하여 데이터 전송을 시작하면 된다. 만약 메모리에 전송할 데이터가 준비되어 있지 않다면, 마치 Delayed Response와 같이 응답하면 된다.

제안된 메커니즘은 타겟 디바이스가 몇 개의 데이터를 Prefetch 해야 될지를 정확하게 알 수 있기 때문에, Delayed Transaction에서 발생하는 제약들을 해결 할 수 있다.

먼저 Delayed Transaction에서 버스 마스터가 읽어 가려는 데이터에 비해 타겟 디바이스가 Prefetch 해둔 데이터가 적거나 많기 때문에 발생하는, 버스 사용 및 데이터 전송 효율 저하를 해결 할 수 있다. 물론 Delayed Transaction에서도 항상 많은 양의 데이터를 Prefetch하는 경우에는 전송 효율이 향상된다. 그러나 그렇게 될 경우, 전송될 데이터 크기에 상관없이 항상 많은 양의 데이터를 Prefetch 함으로 인해 로컬 버스와 내부 메모리의 효율이 떨어 질 것이다.

그 다음은 버스 마스터가 많은 양의 데이터를 읽어 가려고 하였지만 버스 마스터의 내부적인 문제나 버스 사용권을 장시간 점유 할 수 없음으로 인해 불가피하게 프로토콜을 종료한 경우이다. 이 경우 제안된 메커니즘에서는 계속해서 Prefetch를 하여 메모리에 저장해 둔다. 버스 마스터는 분명히 남은 데이터 전송을 하기 위해 다시 메모리 Read 명령어를 요구 할 것이고, 타겟 디바이스는 Prefetch 해둔 데이터를 전송하면 된다.

결국 제안된 방법은 Delayed Transaction 메커니즘에 비해 데이터 전송 효율을 향상시킬 뿐 아니라, PCI 버스와 로컬 버스의 사용 효율을 향상시킬 수 있다.

이러한 메커니즘을 지원하기 위해서는 PCI 디바이스가 PCR을 가지도록 설계되어져야 할 뿐 아니라, 디바이스 드라이브도 이를 지원하도록 설계되어져야 한다. 즉 OS가 제안된 메커니즘을 지원하는 PCI 디바이스에 메모리 Read 명령어를 내리기 위해 디바이스 드라이브를 호출 할 때, 디바이스 드라이브는 PCR에 Write한 후 메모리 Read 명령어를 수행하도록 지시하여야 한다.

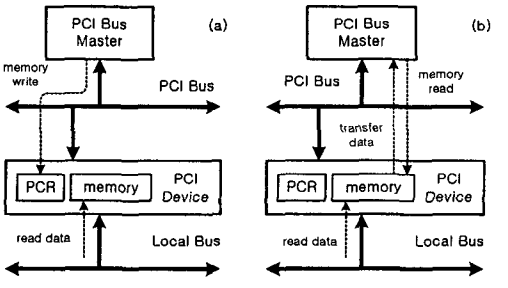

그림 2. 제안된 메커니즘.

V. 모의 실험 및 결과

본 실험을 위해 PCI 2.2 버스 마스터, 타겟 디바이스, 그리고 로컬 디바이스를 C 언어를 이용하여 동작 모델로 구현하였으며, 테스트 환경을 위해 버스 기능 모델도 C 언어로 구현하였다. 또한 타겟 디바이스는 Delayed Transaction과 제안된 메커니즘을 모두 지원하도록 설계되었다.

타겟 디바이스의 내부 메모리는 4K이며, 로컬 버스의 지연은 무작위(0~8)로 하였으며 PCI 버스는 지연없이 동작하도록 한 다음 메모리 Read 명령어를 이용하여 시뮬레이션 하였다. 각 시뮬레이션 결과 그림에서 (i)는 타겟 디바이스가 Delayed Transaction으로 응답한 경우이고, (ii)는 제안된 메커니즘을 적용한 경우이다.

그림 3은 버스 마스터가 요구하는 데이터의 크기를

증가시킬 때 데이터 전송 속도를 시뮬레이션 한 것이다. 여기서 Delayed Transaction을 위한 Prefech 크기는 16 바이트로 설정하였다. 이는 캐시 라인 크기로 PCI 2.2 spec에서 소개된 일반적인 Prefech 크기이다 [4]. 실험 결과, 전송할 데이터가 많을수록 제안된 메커니즘이 Delayed Transaction에 비해 훨씬 높은 데이터 전송 효율을 가짐을 알 수 있다.

그림 4는 Delayed Transaction에서 Prefech 크기를 변화시킬 때의 데이터 전송 속도를 측정한 것이다. 전송할 데이터 크기는 3K에서 4K 바이트 범위에서 무작위로 선택하였다. 여기에서 보듯이, Prefech 데이터 크기가 작을 때는 Delayed Transaction의 성능이 많이 떨어지지만, 3K 바이트 이상 Prefetch를 하는 경우 제안된 메커니즘과 거의 같아짐을 알 수 있다.

그림 5는 버스 마스터가 내부적인 문제나 버스 사용권을 장시간 점유 할 수 없음으로 인해 불가피하게 프로토콜을 종료한 경우로, 앞의 경우와 동일조건에서 시뮬레이션 되었다. 한번에 모든 데이터가 전송된 경우에는 두 경우가 거의 비슷한 성능을 가지지만, 중간에 전송이 끊기는 횟수가 많을수록 제안된 메커니즘의 성능이 조금 좋음을 알 수 있었다.

VI. 결론

PCI 2.2 spec에서는 전체 데이터 전송 효율을 떨어뜨리는 메모리 Read 명령어를 효과적으로 수행하기 위해 Delayed Transaction이라는 메커니즘을 제안하였다. 이는 아주 효율적인 버스 사용을 할 수 있음에도 불구하고 데이터 전송 효율을 저하시키는 몇 가지 제약이 있다.

이에 본 논문에서는 보다 효율적인 버스 사용과 데이터 전송 효율을 향상하기 위한 새로운 메커니즘을 제안하였다. 메모리 Read 명령어를 수행하고자 하는 버스 마스터는 미리 타겟 디바이스에게 읽어올 데이터의 양을 알려 줌으로써, 타겟 디바이스가 Prefetch에서 발생하는 여러 가지 제약 사항을 해결하였다.

본 시뮬레이션을 위해 로컬 디바이스를 C 언어를 이용하여 동작 모델로 구현하였으며, 테스트 환경을 위해 버스 기능 모델도 C 언어로 구현하였다. 실험 결과 Delayed Transaction 메커니즘에 비해 상당 부분 성능 향상이 있음을 확인 할 수 있었다.

참고 문헌

[1] "PCI 버스 해설과 인터페이스 카드 설계.", 동역메카트로닉스연구소.

[2] Edward Solari and George Willse, "PCI hardware and software : architecture and design", Annabooks, 1998.

[3] Don Anderson and Tom Shabnley, "PCI System Architecture.", Mindshare, 1999.

[4] "PCI Local Bus Specification", PCI SIG, Revision 2.2, 1998.

[5] "PCI Local Bus Specification", PCI SIG, Revision 2.3, 2002.

[6] "PCI-X Addendum to the PCI Local Bus Specification", PCI SIG, Revision 1.0a, 2000.

[7] "PCI Express Base Specification", PCI SIG, Revision 1.0a, 2003.

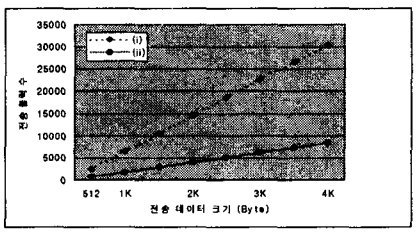

그림 3. 데이터 크기에 따른 전송 속도

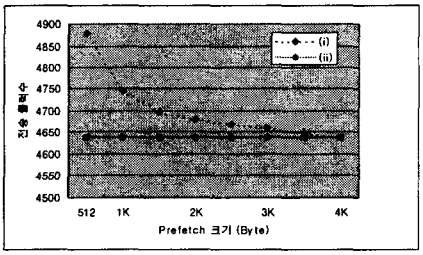

그림 4. Prefetch 크기에 따른 전송 속도

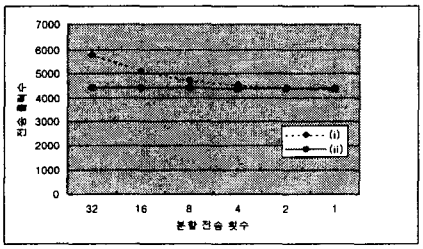

그림 5. 분할 전송 횟수에 따른 전송 속도

감사의 글

이 논문은 2002년도 영남대학교 학술연구 조성비에 의한 지원과 IDEC의 부분적인 지원에 의한 것임.