논문 2004-41CI-4-1

PCI 2.2에서 프리페치 요구를 이용해서 데이터 전송 효율을 향상시키는 효과적인 방법

(Method to improve the Data Transfer Efficiency in the PCI 2.2 using Prefetch Request)

현 유 진*, 성 광 수**

(Eugin Hyun and Kwang-Su Seong)

요 약

PCI 2.2 버스 마스터가 메모리 읽기 명령으로 타겟 장치에 데이터 전송을 요구하면 타겟 장치는 내부적으로 데이터 준비하는데 시간이 필요하므로 데이터 전송 없이 장시간 PCI 버스를 점유하는 상황이 발생할 수 있다. 이는 PCI 버스 사용 효율 및 데이터 전송 효율을 떨어뜨리게 되며 이를 해결하기 위해 PCI 2.2에서는 지연 트랜잭션 메커니즘을 이용한다. 그러나 이 방법은 타겟 장치가 프리패치해야 할 정확한 데이터의 양을 알 수 없기 때문에 데이터 전송 효율이 떨어진다. 본 논문에서는 메모리 읽기 명령을 수행하고자 하는 버스 마스터가 메모리 쓰기 명령어를 이용하여 타겟 장치에게 읽어올 데이터의 양을 미리 알려주는 프리페치 요구를 이용해 보다 효율적으로 데이터를 전송하는 방법을 제안한다. 모의실험 결과 제안된 방법이 지연 트랜잭션에 비해 데이터 전송 효율이 평균 10 % 향상되었다.

Abstract

When the PCI 2.2 bus master requests data using Memory Read command, the target device may hold PCI bus without data transfer for a long time because the target device requires time to prefetch data internally. Because the PCI bus usage efficiency and the data transfer efficiency are decreased due to this situation, the PCI specification recommends to use the Delayed Transaction mechanism to improve the performance. But the mechanism doesn't fully improve performance because the target device doesn't know prefetch data size exactly. In this paper, we propose a new method to transfer data efficiently when the bus master reads data from the target device. The bus master informs the target device the exact read data size using prefetch request using Memory Write command. The simulation result shows that the proposed method has the higher data transfer efficiency than the Delayed Transaction about 10%.

Keywords: PCI bus, PCI 2.2, 전송효율

I. 서 론

1992년 PCI 2.1 표준안이 만들어진 이후 지금까지 약 10년간 PCI 버스가 I/O 시스템에 사용되어 왔다. PCI 버스는 CPU와 독립적으로 동작하도록 되어 있고 일반적으로 32bit 버스 폭을 지원하고 33Mhz에 동작 한다[1-7]. 그러나 근래에 들어와서 CPU와 메모리의 성능이 향상되고 네트워크 카드 및 디스크 컨트롤러 등을 포함한 주변 장치들이 고성능화됨에 따라 이들을 연결하는 PCI 버스가 전체 시스템의 병목현상이 되고 있다[8]. 이에 PCI SIG에서는 보다 효율적인 버스활용을 위해 PCI-X를 2000년에 제안하였다[9]. 현재 PCI-X 인터페이스는 고속 데이터 전송을 요구하는 서버나 워크스테이션 등에 사용되어지고 있다. 하지만 대부분의 PC는 아직까지 PCI 2.2 버스를 지원하고 있으며 셋톱박스, 통신

* 학생회원, 영남대학교 대학원 전자공학과
(Department of Electronic Engineering)
** 정회원, 영남대학교 전자정보공학부
(School of Electrical Engineering and Computer Science)
※ 이 논문은 2002학년도 영남대학교 학술연구조성비 자원에 의한 것임
접수일자: 2004년1월20일, 수정완료일: 2004년5월10일

시스템, 실시간 시스템, 공장 자동화 시스템 등 다양한 분야에서도 PCI 2.2 버스가 응용되고 있다. 또한 PCI-X는 PCI 2.2 프로토콜을 지원하는 장치에 옵션으로 추가 지원되는 프로토콜이기 때문에 모든 PCI-X 장치는 PCI 2.2 프로토콜을 꼭 지원하여야 한다[9]. 따라서 PCI 2.2 버스의 성능이 전체 시스템의 성능에 많은 영향을 미칠 수 있다.

PCI 2.2 버스 마스터가 타겟 장치로부터 데이터를 읽어 오고자 할 때 타겟 장치는 내부적으로 전송할 데이터를 준비하여야 한다. 이때 PCI 버스에 비해 느리게 동작하는 지역 장치를 가지는 PCI 2.2 타겟 장치나 하위 PCI 버스를 가지는 PCI-PCI 브리지(Bridge)인 경우에는 타겟 장치가 데이터 전송 없이 PCI 버스를 점유하는 상황이 발생하기 때문에 버스 사용 효율 및 데이터 전송 효율이 떨어지게 된다. PCI 2.2에서는 이를 해결하기 위해 지연 트랜잭션(Delayed Transaction)을 사용한다. 즉, 버스 마스터로부터 메모리 읽기(Memory Read) 명령어를 요구받은 PCI 2.2 타겟 장치나 PCI-PCI 브리지는 데이터를 바로 전송하지 못함을 마스터에 알리며 프로토콜을 종료한 후, 지역 버스나 하위 PCI 버스를 통해 데이터를 프리페치(Prefetch)한다. 얼마 후 같은 버스 마스터로부터 같은 메모리 읽기 명령을 요구받게 되면 PCI 2.2 타겟 장치나 PCI-PCI 브리지는 프리페치 해둔 데이터를 버스 마스터에게 전송하게 된다. 그러나 이 방법에서는 PCI 2.2 타겟 장치가 프리페치 해야 할 데이터의 양을 정확히 알 수 없다는 문제점이 있다.

본 논문에서는 메모리 읽기 명령어를 수행하고자 하는 버스 마스터가 메모리 쓰기(Memory Write) 명령어를 이용하여 읽어올 데이터의 양을 타겟 장치에 미리 알려주는 프리페치 요구(Prefetch Request)를 이용해 보다 효율적으로 데이터를 전송하는 방법을 제안한다. 본 논문 Ⅱ장에서는 PCI 2.2 버스의 개요에 대해 살펴보고, Ⅲ장에서 지연 트랜잭션 메커니즘에 대해 소개한다. 그리고 Ⅳ장에서 제안된 방법을 소개하며 Ⅴ장에서 시뮬레이션 결과를 보여주고 Ⅵ장에서 결론을 맺는다.

Ⅱ. PCI 2.2의 개요

PCI 버스에서는 하나의 버스로 어드레스와 데이터를 전송한다. 먼저 데이터 전송을 원하는 버스 마스터는 버스의 사용을 조종 및 관리하는 아비터(Arbiter)에게 버스 사용권을 요구한다. 만약 PCI 버스를 사용하고자 하는 버스 마스터가 아비터로부터 버스 사용권을 받았을 때 다른 버스 마스터가 PCI 버스를 사용하지 않는다면 버스 사용을 알리는 시작 신호와 함께 타겟 장치의 어드레스를 PCI 버스로 전송한다. 그러면 PCI 버스에 존재하는 모든 타겟 장치는 이 어드레스를 읽어 해독을 하고, 만약 그 어드레스가 특정 타겟 장치의 영역이라면 그 장치는 응답 신호를 발생한다. 그 다음부터 데이터는 한번 혹은 연속으로 전송된다. 이때 기본적으로 모든 장치는 32비트로 데이터를 전송하며 확장된 장치는 64 비트로 데이터를 전송할 수 있다[4][5][7].

만약 모든 데이터가 전송되어졌거나 버스 마스터나 타겟 장치의 내부적인 문제가 발생할 경우 프로토콜이 종료되어져야 하는데, 이는 버스 마스터의 직접적인 원인에 의해 종료되어 질 수도 있고 타겟 장치의 요청에 의해 종료되어 질 수도 있다. 먼저 버스 마스터가 직접 프로토콜을 종료하는 경우를 살펴보면 완료 종료(Completion Termination)와 타임아웃 종료(Timeout Termination)를 들 수 있다. 일반적인 경우에 버스 마스터가 전송하고자 하는 데이터를 모두 전송하면 프로토콜을 종료하게 되는데 이를 완료 종료라 하며 아주 정상적인 경우이다[4][5][7]. PCI 2.2 사양에서는 버스 마스터가 장시간 버스를 점유 할 수 없도록 하기 위해 내부에 타이머를 두도록 하고 있다. 만약 버스 마스터가 이 타이머가 종료된 시점에 아비터로부터 버스 사용권을 계속 받지 못했다면 즉시 프로토콜을 종료하여야 한다. 이를 타임아웃 종료라고 한다. 버스 마스터가 다시 버스를 사용하고자 하는 경우 아비터로부터 버스 사용권을 받아야한다[7].

다음은 타겟 장치가 프로토콜 종료를 요청하는 경우로 리트라이 종료(Retry Termination)와 디스커넥트 종료(Disconnect Termination)를 들 수 있다. 버스 마스터가 타겟 장치에 데이터 전송을 요구할 때 타겟 장치의 내부적인 문제 때문에 바로 데이터를 전송하여 줄 수 없다면 이를 버스 마스터에 알려 주어야 한다. 그러면 버스 마스터는 즉시 프로토콜을 종료하여야 한다. 이를 리트라이 종료라 한다. 다시 말해 리트라이 종료에서는 데이터 전송이 일어나기 전에 프로토콜이 종료된다. 얼마의 시간이 지난 후에 버스 마스터는 반드시 타겟 장치에 다시 데이터 전송을 요구해야 한다[7].

이번에는 데이터를 전송하는 도중에 타겟 장치의 내부적인 문제로 인해 더 이상 데이터 전송을 할 수 없는 경우를 고려해보자. 이를 경우 역시 타겟 장치는 버스 마스터에 프로토콜 종료를 요청해야 하고 버스 마스터는 즉시 프로토콜을 종료하여야 한다. 이를 디스커넥트

종료라고 한다. 만약 마스터 장치가 나머지 데이터의 전송을 원한다면 얼마의 시간이 지난 후에 다시 타겟 장치에게 요구해올 것이다[7].

PCI 장치에서 가장 보편적으로 사용되는 명령어는 메모리 읽기 명령어와 메모리 쓰기 명령어로 PCI 버스 마스터가 타겟 장치의 메모리에 접근하고자 할 때 사용되는 명령어이다. 즉 타겟 장치의 메모리에 데이터를 전송하고자 할 때는 메모리 쓰기 명령어를, 반대로 버스 마스터가 타겟 장치의 메모리로부터 데이터를 읽고자 할 때는 메모리 읽기 명령어를 이용한다[7].

메모리 읽기 명령어는 다시 3가지의 명령어로 나누어지는데, 메모리 읽기(Memory Read), 메모리 읽기 라인(Memory Read Line), 그리고 메모리 읽기 멀티플(Memory Read Multiple)이 그것들이다. 다음은 이들 3개 명령어에 대해 설명한다. 마스터 장치로부터 메모리 읽기 명령을 요청 받은 타겟 장치는 32 비트 데이터를 한 개 준비하는데 반해, 메모리 읽기 라인 명령을 요청 받은 경우에는 캐시 라인 크기(Cache line size) 만큼의 데이터를 내부적으로 준비하게 된다. 이때 캐시 라인 크기라 함은 메인 CPU의 내부 데이터 캐시 메모리의 라인 크기를 나타내며 PC에서 일반적인 경우 16 바이트 정도이다. 다음은 메모리 읽기 멀티플 명령을 요청 받은 경우로, 이때 타겟 장치는 캐시 라인 크기의 정수배 만큼 데이터를 전송하기 위해 준비한다[7]. 그러나 마스터 장치가 타겟 장치에 데이터 전송을 요구할 때 이 규칙을 지키지 않아도 된다. 예를 들어 마스터 장치가 메모리 읽기 명령어를 이용해 캐시 라인 크기보다도 큰 데이터를 읽어도 되고, 메모리 읽기 멀티플을 이용해 32 비트 데이터만 읽어도 된다. 그러나 성능 향상을 위해서는 위의 규칙들을 지키는 것이 좋다.

III. 지연 트랜잭션

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 지역 장치로부터 데이터를 읽어 와야 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

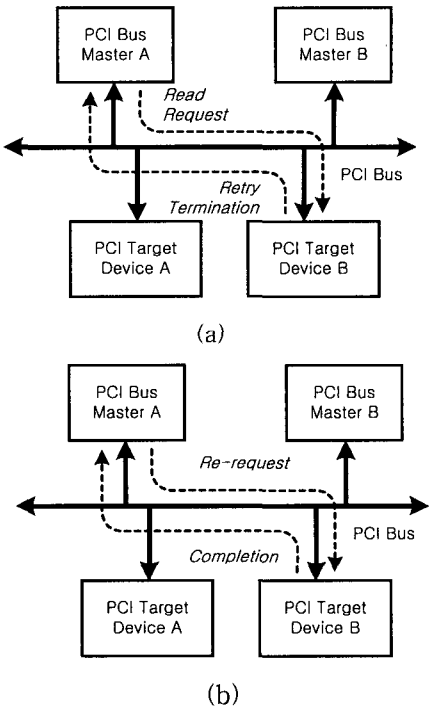


그림 1. 지연 트랜잭션 메커니즘
Fig. 1. Delayed Transaction mechanism.

시간이 오래 걸리는 타겟 장치를 위해 그림 1과 같이 지연 트랜잭션 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. 타겟 장치 B는 기억 해둔 어드레스를 이용하여 자체적으로 데이터를 준비하기 시작하는데 이를 프리페치라고 한다[7]. 만약 타겟 장치에 의해 전송할 수 있는 데이터가 준비된 후에 버스 마스터 A가 똑같은 메모리 읽기 명령을 다시 요구해 오면 그림 1(b)과 같이 준비된 데이터를 전송해준다.

이러한 지연 트랜잭션 메커니즘은 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.

이러한 지연 트랜잭션을 이용하면 효율적으로 버스를 사용 할 수 있음에도 불구하고 다음과 같은 문제점이 있다. 첫 번째, 버스 마스터가 읽어가려는 데이터에 비해 타겟 장치가 프리페치 해둔 데이터가 적은 경우 타겟 장치는 더 이상 전송해 줄 수 있는 데이터가 없으므로 디스커넥트 종료로 프로토콜을 종료하게 된다. 하지만 일반적으로 버스 마스터는 나머지 데이터를 전송

받기 위해 다시 요청해 올 것이다. 얼마 후 버스 마스터는 아비터로부터 다시 버스 사용권을 획득한 후 나머지 데이터 전송을 위해 다시 읽기 명령으로 데이터를 요구하고 이에 대해 지연 트랜잭션으로 처리되어야 한다. 결국 버스 마스터가 다시 버스 사용권을 획득하는 시간과 타겟 장치가 리트라이 종료로 프로토콜을 종료 한 후 다시 프리페치 해야 하므로 전체 데이터 전송 효율이 떨어질 것이다. 이는 버스 마스터가 한번에 많은 양의 데이터를 읽어오고자 할 때 더욱 그러할 것이다.

두 번째는 버스 마스터가 타겟 장치에 의해 프리페치 되어진 데이터 보다 적은 양의 데이터를 읽고 프로토콜을 종료하는 경우이다. 이때 타겟 장치가 아직 하위 버스로부터 데이터를 프리페치 하는 중이라면 즉시 종료하고, 또한 타겟 장치가 이미 프리페치 해둔 남은 데이터를 모두 폐기하여야 한다[7]. 왜냐면 버스 마스터가 타겟 장치로부터 남은 데이터를 다시 읽어갈 것이라는 걸 보장 할 수 없기 때문이다. 만약 버스 마스터가 원하는 데이터를 모두 읽고 완료 종료로 종료한 경우라면 더 이상 데이터를 읽어갈 필요가 없기 때문에 타겟 장치가 남은 데이터를 폐기하여도 아무런 문제가 없다. 그런데 만약 버스 마스터가 많은 양의 데이터를 읽어 가려고 하였지만 버스 마스터의 내부적인 문제나 타임아웃 종료로 불가피하게 프로토콜을 종료한 경우를 고려해 보아야 한다. 이 경우 버스 마스터는 분명히 남은 데이터 전송을 위해 다시 메모리 읽기 명령을 요구 해 올 것이다. 하지만 이미 타겟 장치는 남은 데이터를 모두 폐기하였기 때문에 전송해 줄 데이터가 없다. 이는 결국 버스 사용 및 데이터 전송 효율을 떨어뜨리게 된다.

III. 제안된 방법

본 논문에서는 앞장에서 설명한 지연 트랜잭션 메커니즘에서 버스 사용 및 데이터 전송 효율을 떨어뜨리는 문제점을 해결하기 위한 새로운 방법을 제안한다. 제안된 방법은 버스 마스터가 메모리 읽기 명령을 타겟 장치에 요구하기 전에 미리 몇 개의 데이터를 읽어갈지 정확하게 알려줌으로써, 타겟 장치가 정확한 양의 데이터를 프리페치 할 수 있게 하는 방법이다. 제안된 방법을 설명하기 위해 지역 버스를 가지는 PCI 타겟 장치를 그림 2에 나타내었다.

그림 2(a)에서 메모리 읽기 명령을 수행하고자 하는 버스 마스터는 먼저 메모리 쓰기 명령을 이용하여 다음에 요구할 메모리 읽기 명령의 어드레스와 읽을 데이터

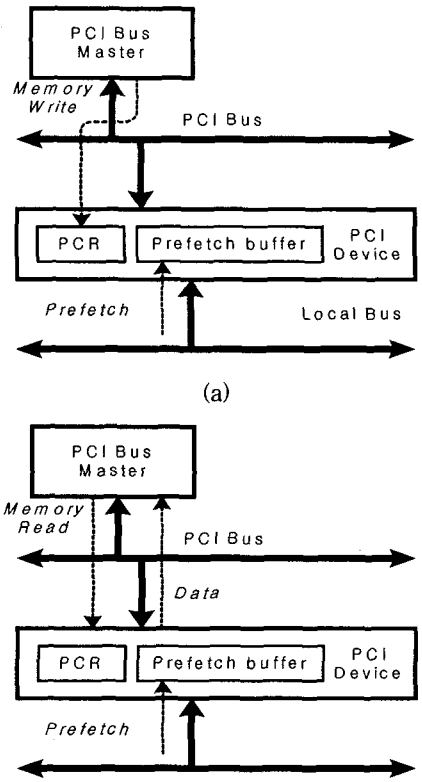


그림 2. 제안된 방법. (a) 메모리 쓰기 명령어를 이용한 프리페치 요구. (b) 메모리 읽기 명령 요구

Fig. 2. The proposed mechanism. (a) Prefetch Request with Memory Write command. (b) Memory Read request.

크기를 PCR(Prefech Control Register)에 저장하는데 이를 프리페치 요구라고 한다. 이 PCR은 제안된 방법을 지원하기 위해 정의된 내부 컨트롤 레지스터(Con-trol Register)이다. PCR에 프리페치를 위한 어드레스와 데이터 크기 정보가 저장되면 타겟 장치는 이를 이용하여 지역 장치로부터 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 프리페치 해야 될지를 정확하게 알 수 있기 때문에 지연 트랜잭션 메커니즘에서 발생하는 문제점을 해결 할 수 있다. 먼저 지연 트랜잭션에서 버스 마스터가 읽어가려는

데이터에 비해 타겟 장치가 프리페치 해둔 데이터가 적거나 많기 때문에 발생하는 문제를 해결 할 수 있다. 물론 지연 트랜잭션에서도 타겟 장치가 항상 많은 양의 데이터를 프리페치 해 둔다면 전송 효율이 향상될 수 있다. 그러나 그렇게 될 경우 버스 마스터가 요구할 데이터의 양이 얼마인지도 모른 채 항상 많은 양의 데이터를 프리페치 해야 하기 때문에 내부 버퍼 및 지역 버스 효율이 떨어지게 된다.

다음은 버스 마스터가 많은 양의 데이터를 읽어 가려고 하였지만 타임아웃 종료이나 버스 마스터 내부 버퍼의 공간 부족 등으로 인해 불가피 하게 프로토콜을 종료한 경우를 고려해보자. 이 경우 제안된 방법에서는 타겟 장치가 나머지 데이터를 폐기하지 않고 버스 마스터가 다시 읽어 갈 때까지 기다리게 된다. 또한 버스 마스터가 요구한 모든 데이터를 타겟 장치가 프리페치 하지 못했다면 나머지 데이터를 계속 프리페치를 한다. 왜냐면 버스 마스터가 얼마만큼의 데이터를 읽어 갈지를 미리 정확히 알려주었기 때문에 얼마 후 남은 데이터 전송을 하기 위해 반드시 다시 메모리 읽기 명령을 요구 할 것이기 때문이다. 따라서 제안된 방법은 지연 트랜잭션 메커니즘에 비해 데이터 전송 효율을 향상시킬 뿐 아니라 PCI 버스와 지역 버스의 사용 효율을 향상시킬 수 있다.

V. 모의실험

본 실험을 위해 그림 3 (a)과 같이 아비터, PCI 2.2 버스 마스터, PCI 2.2 타겟 장치, 그리고 지역 장치를 C 언어를 이용하여 클럭 단위로 동작하는 행위 모델(Behavioral model)로 구현하였다. 그리고 각 C 모델들을 Verilog 시뮬레이터 PLI(Programming language in -terface)를 이용해 연결하였다.

전체의 시뮬레이션 환경을 살펴보면 그림 3(b)과 같다. 테스트 벡터는 사용자가 직접 파일로 작성하거나 혹은 랜덤 발생기에 의해 파일로 자동 생성된다. 이 파일에는 PCI 버스 마스터가 수행하기 위한 명령어들이 열거되어 있다. 예를 들면, 지역 메모리 300번지로부터 84 바이트의 데이터를 읽어 PCI 타겟 장치 100번지로 전송하도록 하려면 mWrite 0x100, 0x300, 84 라고 코딩한다. 이 명령어를 테스트 벡터 파일로부터 읽은 버스 마스터는 아비터로부터 버스 사용권을 획득한 후 메모리 쓰기 명령어를 수행한다. 메모리 쓰기 명령어를 모

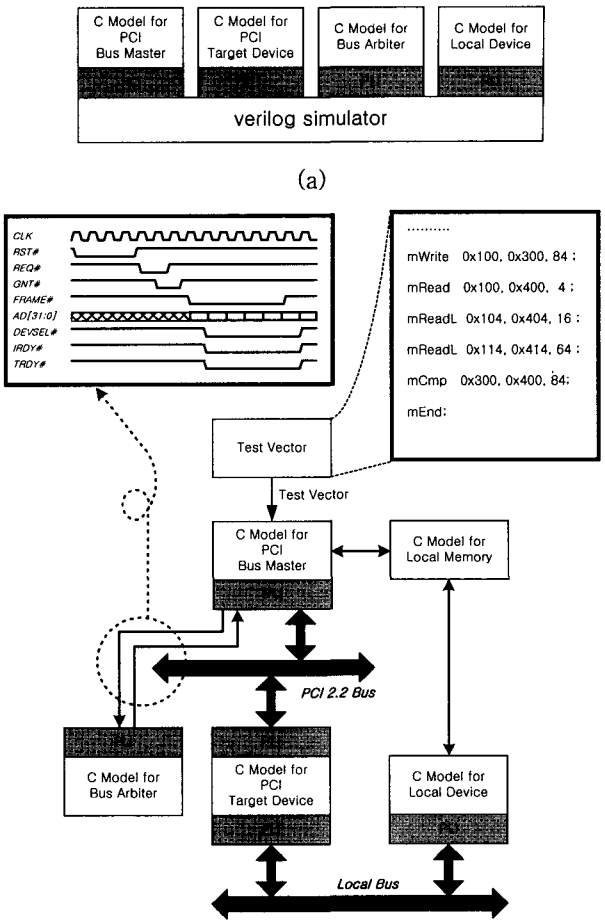


그림 3. 모의실험 환경
Fig. 3. Simulation environment.

두 수행하면, PCI 버스 마스터는 테스트 벡터에서 다음 명령어를 읽어 수행하게 된다.

그림 4는 버스 마스터가 메모리 읽기 멀티플 명령어를 이용하여 데이터 전송을 요구할 때 전송 데이터의 크기를 증가하면서 그때 소요되는 클럭 수를 측정한 것이다. 지연 트랜잭션이 적용된 경우 타겟 장치가 지역 장치로부터 프리페치 할 데이터의 크기를 16 바이트부터 512 바이트까지 증가시키면서 시뮬레이션 하였다. 이와 함께 전송할 데이터의 크기를 512 바이트부터 4K 바이트까지 증가시키며 시뮬레이션 하였다. 이때 지역 버스와 PCI 버스의 지연은 없도록 하였으며 마스터에서 타임아웃이 발생하지 않도록 하였다. 그림 4의 가로축은 전송할 데이터의 크기이며 세로축은 전송이 완료될 때까지 소요된 클럭수이다. (i)~(v)는 지연 트랜잭션이 적용된 경우로 각각 16 바이트, 32 바이트, 64 바이트, 128 바이트, 그리고 512 바이트씩 데이터를 프리페치 하는 경우이고, (vi)는 제안된 방법이 적용된 경우이다.

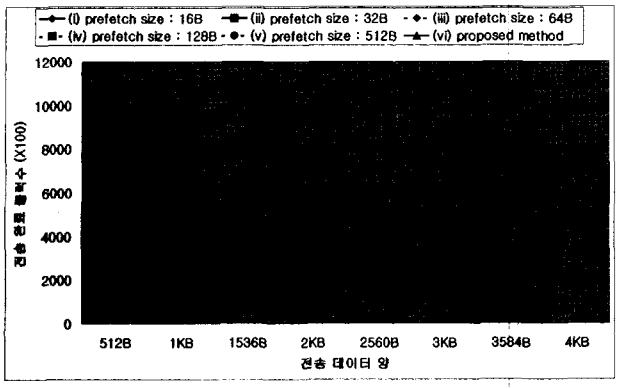


그림 4. 전송할 데이터 크기 변화와 프리페치 데이터 크기 변화에 따라 데이터 전송에 소요되는 클럭수

Fig. 4. The number of clock cycle to transfer data with varying the size of the transferred data and the size of the prefetch data.

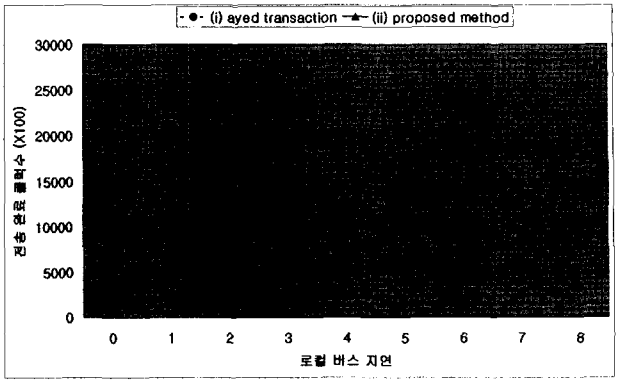


그림 5. 지역 버스 지연에 따라 4K 바이트 데이터 전송에 소요되는 클럭수

Fig. 5. The number of clock cycle to transfer 4K byte data with respect to local bus wait cycle.

데이터의 양이 많을수록 전송 시간이 늘어나며 지연 트랜잭션에서는 프리페치 크기가 클수록 전송 시간이 줄어듦을 알 수 있다. 프리페치 크기가 512바이트 이상이면 제안된 방법과 유사한 성능을 보이고 있다.

그림 5는 지역 버스 지연에 따른 데이터 전송 속도를 나타낸 것이다. 명령어는 메모리 읽기 멀티플을 이용하였고 전송할 데이터의 크기는 4K 바이트로 하였다. 지연 트랜잭션이 적용된 경우 데이터 전송 속도를 최대로 하기 위해 프리페치 할 데이터의 크기를 4K 바이트로 하였다. 또한 PCI 버스의 지연은 없도록 하였으며 버스 마스터에서 타임아웃이 발생하지 않도록 하였다. 그림 5의 가로축은 지역 버스 지연 클럭이며 세로축은 전송이 완료되는데 소요된 클럭 수이다. (i)는 지연 트랜잭션이 적용된 경우이고 (ii)는 제안된 방법이 적용된 경우이다.

지연 트랜잭션을 이용한 방법과 제안된 방법 모두 지역 버스의 지연이 커짐에 따라 데이터 전송 시간이 증가됨을 알 수 있다. 지역 버스 지연이 0에서 7 클럭일 때까지는 지연 트랜잭션과 제안된 방법의 성능이 유사하지만 지역 버스 지연이 8인 경우에는 지연 트랜잭션이 적용된 경우의 전송 시간이 많이 소요됨을 알 수 있다. 그 이유는 지역 버스 지연으로 인해 타겟 장치가 PCI 버스 마스터에게 데이터를 8 사이클 내로 전송하지 못해 타겟 장치는 디스커넥트로 PCI 프로토콜을 종료하여야 하며 그로 인해 타겟 장치에 의해 현재 이루어지는 프리페치는 중단되고 프리페치된 모든 데이터는 폐기되어야 하기 때문이다.

다음으로 버스 마스터가 내부적인 문제로 프로토콜을 종료해야 하거나 버스 사용을 장시간 점유 할 수 없어 타임아웃 종료로 프로토콜을 종료해야 하는 경우이다. 버스 마스터는 메모리 읽기 멀티플 명령어를 이용하여 4K 바이트만큼의 데이터 전송을 요구하였으며, 이때 타겟에 지연 트랜잭션이 적용된 경우 프리페치 할 데이터의 크기를 4K 바이트로 하였다. 또한 PCI 버스 지연은 없도록 시뮬레이션 하였다. 그림 6에서와 같이, 버스 마스터가 중간에 프로토콜 종료 없이 4K 바이트만큼의 데이터를 한번에 모두 전송 받는 것부터 최대 32번 나누어 전송 받도록 시뮬레이션 하였다. 그림 6의 가로축은 버스 마스터에 의한 분할 전송 횟수이며 세로축은 전송이 완료된 클럭 수이다. (i)는 지연 트랜잭션이 적용된 경우이고 (ii)는 제안된 방법이 적용된 경우이다.

실험 결과 한번에 모든 데이터가 전송된 경우에는 두 경우가 거의 비슷한 성능을 가지지만, 중간에 전송이 끊기는 횟수가 많을수록 제안된 방법의 성능이 나음을 알 수 있었다. 그 이유는 지연 트랜잭션이 적용된 타겟 장치가 비록 4K 바이트만큼의 데이터를 프리페치 하였다 하더라도 버스 마스터가 중간에 프로토콜을 종료하면 타겟 장치는 프리페치 해둔 데이터를 모두 폐기해야 하지만, 제안된 방법에서는 비록 버스 마스터가 중간에 프로토콜을 종료하더라도 타겟 장치는 데이터를 폐기하지 않고 버스 마스터가 다시 요구하기를 기다리기 때문이다.

마지막으로 전송할 데이터의 크기를 4 바이트에서 4K 바이트까지 무작위로 100개를 선택되도록 하여 시뮬레이션을 하였다. 이때 지역 버스 지연은 0에서 8로 무작위로 선택되도록 하였고 PCI 버스 지연은 없도록 시뮬레이션 하였다. 버스 마스터 장치가 타겟 장치로부터 데이터를 4 바이트만큼 읽고자 할 땐 메모리 읽기

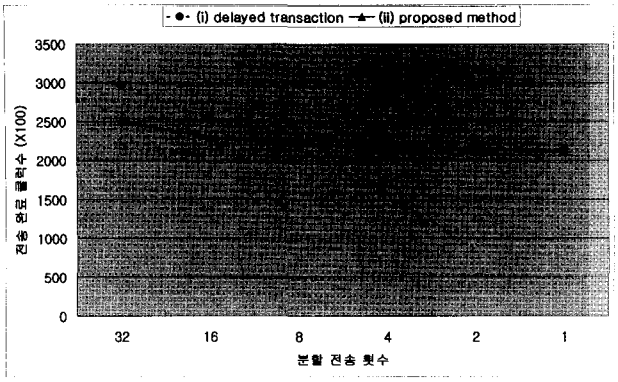


그림 6. 버스 마스터에 의한 전체 데이터가 분할 전송됨에 따라 4K 바이트 데이터 전송에 소요되는 클럭수

Fig. 6. The number of clock cycle to transfer data with respect to the number of disconnect due to the bus master.

명령어를, 16 바이트 이하를 읽고자 할 땐 메모리 읽기 라인 명령어를, 그리고 그 이상의 데이터를 읽고자 할 땐 메모리 읽기 멀티플 명령어가 사용하도록 하였다. 또한 마스터 장치가 데이터를 읽을 때 중간에 프로토콜 종료 없이 한번에 모든 데이터를 읽어 갈지, 아니면 최대 32번까지 나누어 전송 받을지도 무작위로 선택되도록 하였다. 그 결과 제안된 방법이 지연 트랜잭션에 비해 평균 10 % 성능 향상이 됨을 알 수 있었다.

VI. 결 론

PCI 2.2 버스 마스터가 메모리 읽기 명령으로 타겟 장치에 데이터 전송을 요구하면 타겟 장치는 내부적으로 전송할 데이터를 준비하는데 시간이 걸리게 되므로 전체 성능을 저하시키는 원인이 된다. 이를 위해 PCI 2.2에서는 지연 트랜잭션을 이용하지만, 이 역시 타겟 장치가 몇 개의 데이터를 프리페치 하는지 정확히 알 수 없다는 단점이 있다.

본 논문에서는 프리페치 요구 방법을 이용해 전송 효율을 향상하는 방법을 제안하였다. 메모리 읽기 명령을 수행하고자 하는 버스 마스터는 메모리 쓰기 명령어를 이용하여 읽어올 데이터의 양을 타겟 장치에 미리 알려주는 프리페치 요구를 타겟 장치에 먼저 요청함으로써 효율적으로 데이터를 전송을 할 수 있다. 실험 결과 제안된 메커니즘은 지연 트랜잭션 메커니즘에 비해 데이터 전송 효율이 평균 10 % 향상되었음을 확인 할 수 있었다.

참 고 문 헌

[1] 동역 메카트로닉스 연구소 기술 정보실, "*PCI 버스 해설과 인터페이스 카드 설계*", 국제 테크노 정보 연구소, 2001.

[2] Edward Solari and George Willse, "*PCI hardware and software: architecture and design*", Anna -books, 1998.

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

[4] Bradly K. Fawcett, "Designing PCI bus interfaces with programmable logic", Proceedings of the Eighth Annual IEEE International ASIC Confer -ence and Exhibit, pp. 321-324, September 1995.

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

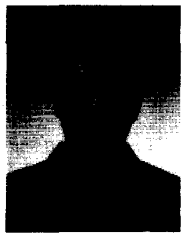
[6] E. Finkelstein and S. Weiss, "Implementation of PCI-based systems using programmable logic", IEE Proceedings Circuits, Devices and Systems, Vol. 147, no. 3. pp 171-174, June 2000.

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

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

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

저 자 소 개



현 유 진(학생회원)
제37권 SD편 제10호 참조
2001년 영남대학교 대학원 전자공학과 석사 졸업.
2003년 영남대학교 대학원 전자공학과 박사 수료
<주관심분야: 디지털 시스템 설계, 집적회로 및 CAD>

성 광 수(정회원)
제37권 SD편 제2호 참조
현재 영남대학교 전자정보공학부 조교수