프리페치 요구를 지원하는 PCI 2.2 타겟 컨트롤러 설계 및 검증

현 유 진[†] · 성 광 수[††]

요 약

PCI 2.2 마스터 디바이스가 타겟 디바이스로부터 데이터를 읽어 오고자 할 때 타겟 디바이스는 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 버스 사용 및 데이터 전송 효율을 떨어뜨리는 원인을 제공한다. 이에 앞선 연구에서는 이를 해결하기 위한 프리페치 요구를 이용하는 새로운 방법을 제안하였다. 본 논문에서는 이 방법을 지원하는 PCI 타겟 컨트롤러와 로컬 디바이스를 설계하였다. 설계된 PCI 타겟 컨트롤러는 간단한 로컬 인터페이스를 가질 뿐 아니라 PCI 2.2를 전혀 모르는 사용자도 쉽게 PCI 인터페이스를 지원할 수 있도록 설계 되었다. 또한 설계된 하드웨어를 효과적으로 검증하기 위한 방법으로 기본 동작 검증, 설계 기반 검증, 그리고 랜덤 테스트 검증을 제안하였다. 이러한 검증을 위해 테스트 벤치와 테스트 벤치를 동작시키는 위한 명령어를 제안하였다. 그리고 랜덤 테스트를 위해 참조 모델, 랜덤 발생기, 비교 엔진으로 구성된 테스트 환경을 구축하였으며 이를 이용해 코너 케이스를 효과적으로 검증할 수 있다. 또한 제안된 테스트 환경을 통해 시뮬레이션 한 결과, 프리페치 요구를 이용한 제안된 방법이 지연 전송에 비해 데이터 전송 효율이 평균 9% 향상되었다.

키워드 : PCI, PCI 2.2, 컨트롤러, 프리페치, 검증

Design and Verification of PCI 2.2 Target Controller to support Prefetch Request

Eugin Hyun[†] · Kwang-Su Seong[††]

ABSTRACT

When a PCI 2.2 bus master requests data using Memory Read command, a target device may hold PCI bus without data to be transferred for long time because a target device needs time to prepare data internally. Because the usage efficiency of the PCI bus and the data transfer efficiency are decreased due to this situation, the PCI specification recommends to use the Delayed Transaction mechanism to improve the system performance. But the mechanism cann't fully improve performance because a target device doesn't know the exact size of prefetched data. In the previous work, we propose a new method called Prefetch Request when a bus master intends to read data from the target device. In this paper, we design PCI 2.2 controller and local device that support the proposed method. The designed PCI 2.2 controller has simple local interface and it is used to convert the PCI protocol into the local protocol. So the typical users, who don't know the PCI protocol, can easily design the PCI target device using the proposed PCI controller. We propose the basic behavioral verification, hardware design verification, and random test verification to verify the designed hardware. We also build the test bench and define assembler instructions. And we propose random testing environment, which consist of reference model, random generator, and compare engine, to efficiently verify corner case. This verification environment is excellent to find error which is not detected by general test vector. Also, the simulation under the proposed test environment shows that the proposed method has the higher data transfer efficiency than the Delayed Transaction about 9%.

Key Words : PCI, PCI 2.2, Controller, Prefetch, Verification

1. 서 론

지난 10년간 PCI 2.2는 입출력 인터페이스의 표준안으로 자리 잡아왔다. 하지만 근래에 들어와서 주변 디바이스들이 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 사용되고 있다[8]. 그러나 아직 대부분의 PC는 PCI 2.2 버스를 지원하고 있으며 셋탑박스, 통신 시스템, 실시간 시스템

※ 이 논문은 2005학년도 영남대학교 학술연구조성비 지원에 의한 것임.
† 정 회 원 : DGIST IT연구부 HW연구팀 연구원
†† 정 회 원 : 영남대학교 공과대학 전자정보공학부 부교수
논문접수 : 2005년 5월 23일, 심사완료 : 2005년 10월 24일

등 다양한 분야에서도 아직 응용되고 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 디바이스는 내부적으로 데이터를 준비해야 함으로 인해 데이터 전송 없이 PCI 버스를 점유하는 상황이 발생한다. 이를 해결하기 위해 PCI 2.2 사양에서는 지연전송(Delayed transaction)을 제안하여 전송 효율을 향상시켰다[7]. 하지만 이 방법 역시 타겟 디바이스가 얼마의 데이터를 미리 준비 해둘지를 알 수 없어 버스 사용 및 데이터 전송 효율을 떨어뜨리는 원인을 제공한다. 이를 해결하기 위해 앞선 연구에서는 프리페치 요구를 이용하는 새로운 방법이 제안되었다[9].

본 논문에서는 프리페치 요구를 지원하는 PCI 타겟 컨트롤러와 로컬 디바이스를 설계하였다. 또한 설계한 하드웨어를 효과적으로 검증하기 위해 기본 동작 검증, 설계 기반 검증, 그리고 랜덤 테스트 검증을 제안한다. 이러한 검증을 위해 테스트 벤치와 테스트 벤치를 동작시키는 위한 명령어를 제안한다.

또한 랜덤 테스트 검증을 위해 참조모델, 랜덤발생기, 비교엔진으로 구성된 테스트 환경을 구축하였다. 이를 이용해 코너 케이스(corner case)를 효과적으로 검증을 할 수 있었다.

이러한 테스트 환경을 통해 제안된 방법과 지연 전송의 성능을 시뮬레이션 및 분석을 할 수 있었다.

본 논문의 2장에서는 PCI 2.2의 개요에 대해 살펴보고 3장에서는 설계한 컨트롤러와 이를 검증하기 위해 제안된 방법을 소개할 것이다. 그리고 4장에서는 시뮬레이션 결과를 보여주며 5장에서 결론을 맺는다.

2. PCI 2.2 개요

PCI 버스는 하나의 버스로 어드레스와 데이터를 전송을 한다. 데이터 전송을 원하는 마스터 디바이스는 먼저 아비터(Arbiter)로부터 버스 사용권을 얻어야 한다. 버스 마스터가 버스 사용권을 얻고 이때 다른 버스 마스터 디바이스가 PCI 버스를 사용하지 않는다면 데이터 전송을 시작할 수 있다. 즉 버스 마스터는 버스 사용을 알리는 신호와 함께 명령어 그리고 타겟 디바이스의 어드레스를 PCI 버스로 전송한다. 이때 사용되는 신호선이 <표 1>의 FRAMEn, CBEn[3:0], 그리고 AD[31:0]이다. 그러면 PCI 버스에 연결된 모든 타겟 디바이스는 이 명령어와 어드레스를 읽어 해독하고 만약 그 어드레스가 특정 타겟 디바이스의 영역이면 그 디바이스는 응답신호를 발생한다. 이때 사용되는 신호선이 DEVSELn이다. 그 다음부터 데이터는 한번 혹은 연속으로 전송된다. 이때 마스터 디바이스는 IRDYn 신호선을, 타겟 디바이스는 TRDYn 신호선을 이용하여 데이터 전송하거나 혹은 전송받을 준비가 되어 있음을 알린다. 즉 IRDYn와 TRDYn 신호선이 동시에 활성(active) 되어 있을 때 데이터가 전송되는 것이다[7].

〈표 1〉 PCI 2.2 데이터 전송에 사용되는 인터페이스의 주요 신호

신호이름	의 미
FRAMEn	PCI 프로토콜의 시작과 끝을 알리는 신호
IRDYn	마스터 디바이스가 데이터 전송할 준비가 되었다는 신호
AD[31:0]	타겟 디바이스에 보낼 어드레스와 데이터를 전송하는 신호
CBEn[3:0]	타겟 디바이스에 보낼 명령어와 바이트 선택 정보를 전송하는 신호
DEVSELn	마스터 디바이스로부터 데이터 전송 요청을 받은 타겟 디바이스가 응답하는 신호
TRDYn	타겟 디바이스가 데이터 전송할 준비가 되었다는 신호
STOPn	타겟 디바이스가 프로토콜을 종료함을 알리는 신호

모든 PCI 디바이스는 디바이스의 상태 확인 및 동작 제어를 할 수 있는 컨피규레이션 레지스터(configuration register)를 내장하고 있다. 컨피규레이션 쓰기와 읽기 명령어는 이 레지스터와 데이터 전송을 하기 위한 명령어이다[7]. 시스템이 초기화 되고나면 호스트 브리지는 먼저 이 명령어를 이용하여 모든 PCI 디바이스들의 컨피규레이션 레지스터를 각각 세팅한다.

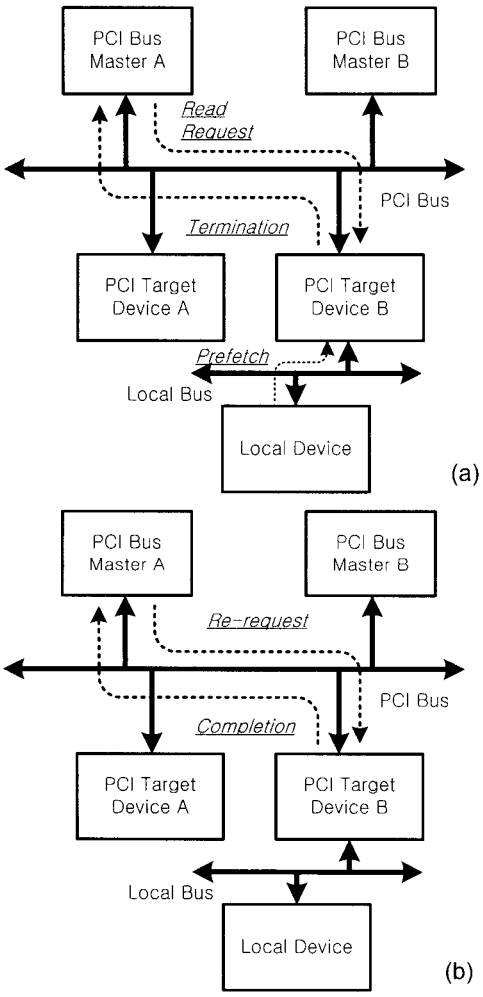
PCI 디바이스에서 데이터 전송을 위해 가장 보편적으로 사용되는 명령어는 메모리 읽기 명령어와 메모리 쓰기 명령어이다. 이 명령어를 PCI 버스 마스터가 타겟 디바이스의 메모리에 접근하고자 할 때 사용되어 진다. 즉 버스 마스터가 타겟 디바이스의 메모리에 데이터를 전송하고자 할 때는 메모리 읽기 명령어를, 반대로 타겟 디바이스의 메모리로부터 데이터를 읽고자 할 때는 메모리 읽기 명령어를 이용한다[7].

이러한 메모리 읽기 명령을 버스 마스터가 타겟 디바이스에 요구한 경우, 타겟 디바이스는 데이터를 전송해주기 위해서 내부적으로 데이터를 준비할 시간이 필요하다. 특히 로컬 버스(local bus)를 가지는 PCI 디바이스인 경우 로컬 디바이스로부터 데이터를 읽어 와야 할 시간이 필요하기 때문에 그 동안 아무런 데이터 전송 없이 PCI 버스를 점유하고 있어야 한다. 만약 로컬 버스가 느리게 동작하는 경우라면 데이터를 전송하기 위해 PCI 버스를 장시간 점령해야 하기 때문에 버스 사용 효율은 떨어지게 된다. PCI 2.2 사양에서는 이렇게 내부적으로 데이터 준비 시간이 오래 걸리는 타겟 디바이스를 위해 (그림 1)과 같이 지연전송 메커니즘을 제안하고 있다[7]. (그림 1)(a)에서 보면 먼저 버스 마스터 A가 타겟 디바이스 B에 메모리 읽기 명령을 요구하면 타겟 디바이스 B는 아직 데이터 전송을 할 수 없음을 알리기 위해 STOPn 신호를 이용하여 리트라이(Retry)로 프로토콜을 종료한다. 여기서 리트라이란 PCI 타겟 디바이스가 즉시 데이터를 전송 할 수 없을 때 프로토콜을 종료하는 방법으로 이 경우 마스터 디바이스는 꼭 다시 데이터 전송을 요청을 해야 한다[7]. 그 후 타겟 디바이스 B는 기억 해둔 어드레스를 이용하여 자체적으로 데이터를 준비하기 시작하는

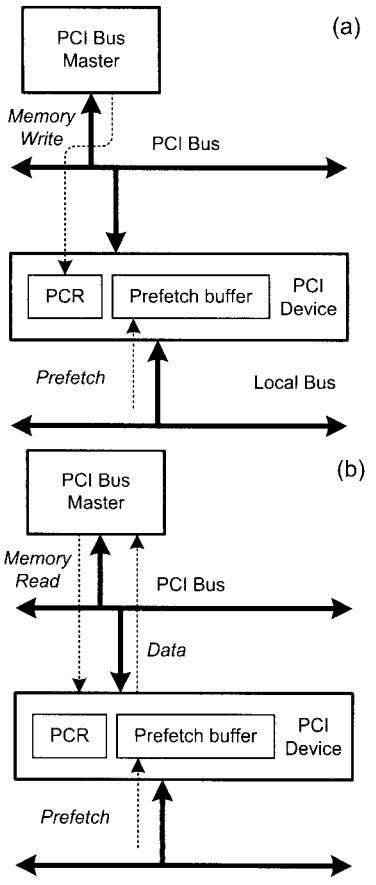
데 이를 프리페치(Prefetch)라고 한다[7]. 만약 타겟 디바이스에 의해 전송 할 데이터가 준비된 후에 버스 마스터 A가 똑같은 메모리 읽기 명령을 다시 요구해 오면 (그림 1)(b)과 같이 데이터는 전송된다.

이러한 지연전송은 타겟 디바이스 B가 버스 마스터 A에 의해 요청된 데이터를 자체적으로 준비를 하는 동안, 버스 마스터 A가 다른 타겟 디바이스와 데이터 전송을 하거나 혹은 버스 마스터 B가 PCI 버스를 사용할 수 있다. 이는 PCI 버스의 사용 효율을 향상시킬 수 있기 때문에, PLX사에서 제작된 PCI 컨트롤러 등에서는 지연전송을 지원한다[10]. 이러한 지연전송을 이용하면 효율적으로 버스를 사용할 수 있음에도 불구하고, 타겟 디바이스가 몇 개의 데이터를 미리 읽어 두어야 하는지를 정확히 알 수가 없어 버스 사용 및 데이터 전송 효율을 떨어뜨리는 원인을 제공한다. 이는 앞선 연구에서 분석되었고 이를 효과적으로 해결하기 위한 방법도 소개하였다[9]. 버스 마스터가 메모리 읽기 명령을 타겟 디바이스에 요구하기 전에 미리 몇 개의 데이터를 읽어갈지 알려줌으로써 타겟 디바이스가 정확한 양의 데이터를 읽게 하는 방법이다. 그 과정을 간단히 살펴보면 (그림 2)(a)에서 버스 마스터는 먼저 메모리 쓰기 명령을 이용하여 다음에 요구할 메모리 읽기 명령의 어드레스와 읽어올 데이터의 크기를 메모리 쓰기 명령어의 데이터로 보낸다. 이때 메모리 쓰기 명령어의 타겟 주소는 타겟 디바이스의 *PCR*(Prefech Control Register) 주소이고 메모리 쓰기를 이용한 이런 요청을 프리페치 요구(Prefetch Request)라고 한다. 이 *PCR*은 제안된 방법을 지원하기 위해 정의된 내부 컨트롤 레지스터(Control Register)이다. 타겟 디바이스는 데이터를 수신한 후 *PCR*에 프리페치를 위한 어드레스와 데이터 크기 정보가 저장한다. 그리고 타겟 디바이스는 이를 이용하여 로컬 디바이스로부터 데이터를 프리페치 하여 내부 버퍼에 저장한다. (그림 2)(b)에서와 같이 버스 마스터는 곧 메모리 읽기 명령을 타겟 디바이스에 요구할 것이다. 이때 타겟 디바이스가 전송할 데이터를 아직 프리페치 하지 못했다면 리트라이 종료로 프로토콜을 종료하고 만약 전송할 데이터가 준비되었다면 버스 마스터에게 데이터를 전송하면 된다.

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



(그림 1) 지연 전송 메커니즘, (a) 지연 전송 요청, (b) 지연 전송 완성



(그림 2) 프리페치 요구를 이용하는 제안된 방법, (a) 프리페치 요구, (b) 메모리 읽기 명령 완성

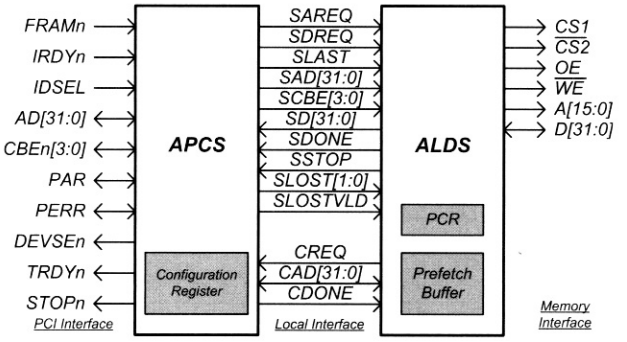
수 있다[9].

이 방법은 PCI 2.2 프로토콜을 어기지 않고 사용하기 때문에, 기존 PCI 디바이스 드라이버와도 호환이 유지된다. 즉, PCI 2.2 디바이스로부터 메모리 읽기 명령어를 수행하기 전에 메모리 쓰기 명령어를 먼저 수행하도록 하면 되기 때문이다.

3. PCI 2.2 컨트롤러 설계 및 검증

3.1 설계

본 논문에서는 (그림 3)과 같이 로컬 인터페이스를 가지는 PCI 2.2 타겟 컨트롤러- 이하 이를 *APCS*(Advanced PCI Controller for Slave Device)라 명칭 함-를 설계하였다. 설계된 *APCS*는 내부에 따로 버퍼를 두지 않고 단지 PCI 프로토콜을 로컬 프로토콜로 변환해 주는 기능을 한다. 즉 PCI 2.2를 전혀 모르는 사용자도 *APCS*를 이용하면 쉽게 PCI 인터페이스를 지원하는 디바이스를 설계할 수 있다. 또한 SRAM과 연결된 로컬 디바이스-이하 *ALDS*(Advanced Local Device for SRAM)라 명칭 함-도 설계하였다. 이 *ALDS*는 지연전송과 제안된 방법을 모두 지원하며 이를 위해 *PCR*과 프리페치 버퍼를 내부에 가지고 있다. <표 2>는 *APCS*의 로컬 인터페이스 신호를 나타낸다.



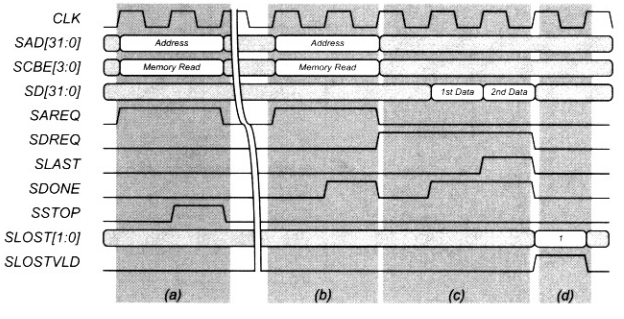
(그림 3) 설계된 APCS와 ALDS의 블록 다이어그램

〈표 2〉 APCS의 로컬 인터페이스의 신호

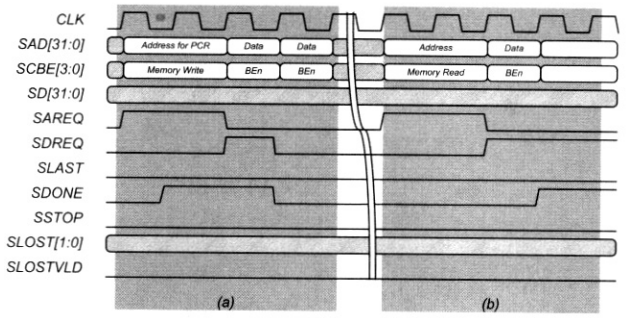
신호이름	의 미
SAREQ	APCS가 로컬 디바이스로 어드레스를 보낸다는 신호
SDREQ	APCS가 로컬 디바이스로 데이터를 보낼 준비가 되었거나, 로컬 디바이스로부터 데이터를 받을 준비가 되었다는 신호
SLAST	APCS와 로컬 디바이스 간에 전송되는 마지막 데이터를 알리는 신호
SAD[31:0]	APCS가 로컬 디바이스로 보내는 어드레스와 데이터를 전송하는 신호
SCBE[3:0]	APCS가 로컬 디바이스로 보내는 명령어와 바이트 선택 정보를 전송하는 신
SD[31:0]	로컬 디바이스가 APCS로 보내는 어드레스와 데이터를 전송하는 신호
SDONE	로컬 디바이스가 APCS로부터 데이터를 받을 준비가 되었거나, 로컬 디바이스로 데이터를 보낼 준비가 되었음을 알리는 신호
SSTOP	로컬 디바이스가 APCS에 더 이상 데이터를 전송할 수 없음을 알리는 신호

*APCS*와 *ALDS*의 동작 과정 중, 먼저 지연 전송이 *ALDS*에 적용된 경우를 살펴보면 (그림 4)와 같다. *APCS*가 마스터 디바이스로부터 메모리 읽기 명령어를 요청 받게 되는 경우 *APCS*는 먼저 *SAD[31:0]*와 *SCBE[3:0]*를 통해 해당 주소와 명령어를 *ALDS*로 전송한다. 이때 *SAREQ*를 드라이브 하여 주소와 명령어가 유효함을 알린다(구간 a). 그러면 로컬 디바이스는 *SSTOP*을 드라이브 하여 데이터를 바로 전송하지 못함을 *APCS*에 알린다. 곧 *APCS*는 PCI 프로토콜을 리트라이로 종료 할 것이다. 하지만 로컬 디바이스는 *APCS*로부터 수신한 주소를 이용하여 *SRAM*으로부터 일정량의 데이터를 미리 프리페치 한다. 얼마 후 마스터 디바이스는 *APCS*를 통해 다시 똑같은 메모리 읽기 명령어를 요구해 올 것이다. 이때 로컬 디바이스인 *ALDS*가 프리페치 버퍼에 데이터를 가지고 있다면 *SDONE*를 이용하여 데이터 전송 개시를 알린다(구간 b). 그러면 *APCS*는 *SDREQ*를 이용하여 데이터 전송 요청을 하고 *ALDS*는 *SD[31:0]*을 통해 이들을 전송한다. 이때 *SDONE*를 이용하여 전송하는 데이터가 유효함을 알린다. 만약 마스터 디바이스에 의해 PCI 버스 상에서 프로토콜이 종료되면 *APCS*는 *SLAST* 신호를 통해 로컬 디바이스 *ALDS*에 이를 알린다(구간 c). 반대로 *ALDS*가 더 이상 데이터 전송을 못하는 경우 *SSTOP*을 이용하여 이를 *APCS*에 알린다. 이렇게 PCI 버스에서 데이터 전송이 종료되는 경우, *APCS*가 *ALDS*로부터 읽어왔음에도 불구하고 PCI 마스터 디바이스로 모든 데이터를 전송하지 못 할 수 있다. 이는 두 인터페이스의 프로토콜 타이밍이 다름으로 인해 설계상 나타나는 것으로 최대 2개이다. *APCS*는 이를 보상하기 위해 메모리 읽기 명령어를 수행 한 후에 *SLOST[1:0]*를 이용하여 PCI 버스 마스터 디바이스로 전송되지 못한 데이터의 개수를 로컬 디바이스에 알려준다(구간 d). (그림 4)에서는 *APCS*가 로컬 디바이스로부터 2개의 데이터를 읽어 왔지만, PCI 마스터가 프로토콜을 종료함으로 인해 마지막 데이터는 전송되지 못한 경우이다.

다음은 프리페치 요구가 적용된 경우로 (그림 5)에 나와 있다. 먼저 로컬 마스트 디바이스는 메모리 읽기 명령어를 하기 전에 프리페치 요구를 요청한다. 그러면 *APCS*는 *SAD[31:0]*을 통해 *ALDS*의 *PCR* 주소와 프리페치 요구를



(그림 4) 지연 전송으로 동작하는 APCS와 ALDS의 타이밍 다이어그램



(그림 5) 제안된 방법으로 동작하는 APCS와 ALDS의 타이밍 다이어그램

위한 정보를 *ALDS*로 전송한다(구간 a). 곧 *ALDS*는 *SRAM*으로부터 데이터를 프리페치 할 것이다. 얼마 후 마스터 디바이스는 메모리 읽기 명령어를 *APCS*에 요구할 것이고 그 후에 과정은 앞에 소개된 (그림 4)와 같다(구간 b).

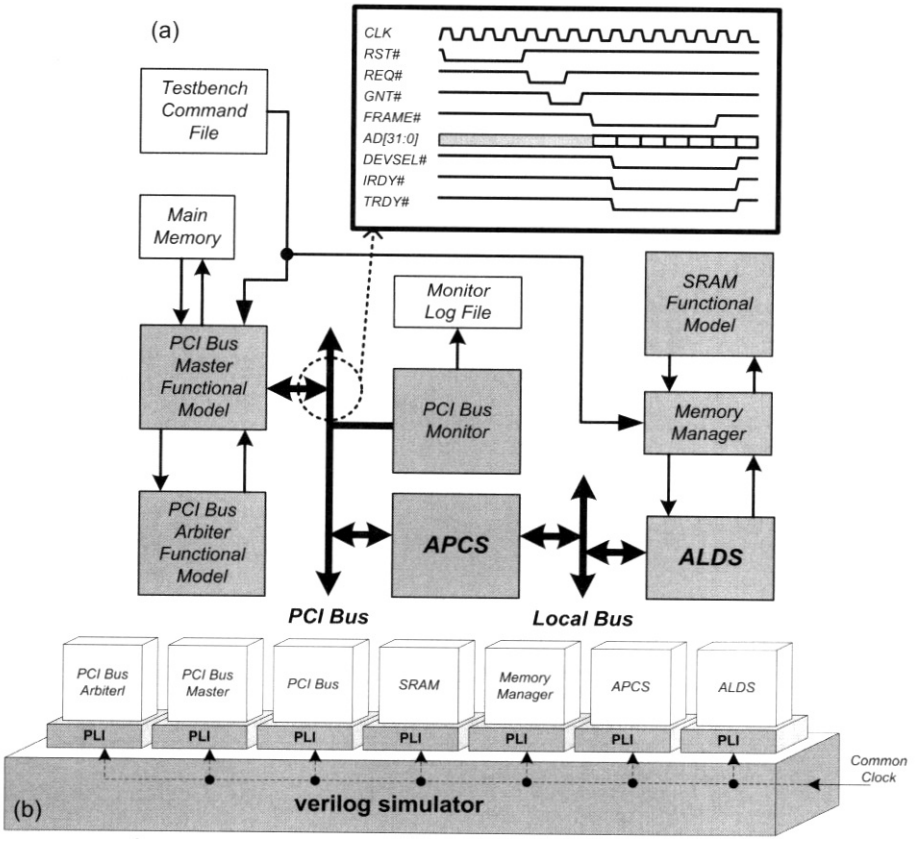
3.2 검증

본 논문에서는 설계된 디바이스의 올바른 동작을 검증하기 위해 먼저 C 언어를 이용하여 *APCS*와 *ALDS*의 행위 모델(Behavioral model)을 설계하였다. 설계된 두 디바이스의 동작은 (그림 6)의 테스트 벤치(test bench)를 통해 검증되었다. 또한 이 행위 모델을 이용하여 제안된 프리페치의 성능도 시뮬레이션을 통해 측정하였다. *APCS*와 *ALDS*는 다시 Verilog HDL을 이용하여 실제 하드웨어로 코딩되었다. 이들 역시 (그림 6)의 테스트 벤치를 통해 테스트 되었다.

테스트 벤치는 (그림 6)(a)와 같이 PCI 버스 마스터, PCI 버스 아비터, PCI 버스 모니터, 그리고 메모리의 행위 모델들로 구성되어있다. 여기서 PCI 버스 모니터는 버스 마스터나 *APCS*가 PCI 프로토콜을 어기는 경우 이를 로그(log) 파일로 만들어 에러로 보고하는 프로토콜 감시자(checker)이다. 이들 행위 모델들은 (그림 6)(b)와 같이 *PLI*(Programming language interface)를 통해 연결되어 공통 클럭을 기반으로 동작된다. 여기서 *PLI*는 C 언어로 작성된 프로그램과 verilog HDL로 작성된 프로그램을 연결하기 위한 인터페이스이다[11].

일반적인 마이크로프로세서는 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 어셈블러와 유사한 명령어들을 정의 하였다.

이들 명령어는 크게 세 개의 형태로 나누어진다. 첫 번째는 버스 마스터에 PCI 버스를 통해 데이터 전송을 지시하는 명령으로 <표 3>에 기술되어있다. 먼저 '*MWrite*', '*MRead*', '*CWrite*', 그리고 '*CRead*' 명령어는 버스 마스터 모델로 하여금 *APCS*에 메모리 쓰기 및 읽기, 컨피규레이션 쓰기 및 읽기 명령어를 수행하라고 지시하는 명령어이다. 그리고 두



(그림 6) 시뮬레이션 환경, (a) 테스트 벤치, (b) PLI를 이용한 연결

번째 타입은 <표 4>와 같이 메인 메모리와 로컬 메모리를 접근하기 위한 명령어이다. '*MSave*' 명령어는 메인 메모리의 특정 주소에 특정 데이터를 저장하기 위한 것이고 '*MSaveDefault*'는 메인 메모리 전체에 무작위로 데이터를 모두 채워 넣기 위한 명령어로 뒤에 소개될 랜덤 테스트 방법에 유용하게 사용된다. '*MDisplay*'는 메인 메모리의 특정 주소의 데이터 값을 화면으로 보기 위한 명령어이다. 그 다음으로 '*LSave*', '*LSaveDefault*', '*LDisplay*'는 로컬 메모리인 SRAM에 적용되는 명령어로 로컬 프로토콜 매니저에 의해 실행된다. 마지막으로 '*CompareMem*' 명령어는 메인 메모리의 특정 주소의 데이터와 로컬 메모리의 특정 주소의 데이터를 비교하는데 사용되는 명령어이다. 비교 후 오류가 있는 경우 로그 파일로 에러를 보고한다.

마지막 타입은 실제 데이터 전송 시에 발생 할 수 있는 여러 가지 시나리오들을 만들기 위한 명령어로 <표 5>에 나와 있다. PCI 버스 마스터 디바이스가 실제 데이터 전송 도중 지연을 줄 수 있는데, 이를 위해 '*MWaitMax*'와 '*MWaitMin*' 명령어를 정의하였고 이는 테스트 벤치의 버스 마스터 모델에 의해 사용되어 최대 지연과 최소 지연을 조절 한다. 또한 *ALDS*가 *SRAM*와 데이터 전송을 할 때도 지연이 발생할 수 있는 이를 위해 '*LWaitMax*'와 '*LWaitMin*' 명령어를 이용하면 된다. 또한 마스트 디바이스가 데이터 전송을 위한 한 개의 명령어를 수행하고 다음 명령어를 수행할 때까지 대기해야 되는 시간도 조절 할 필요가 있는데 이를 위한 명령어가 '*MCmdWaitMax*'와 '*MCmdWaitMin*'이다. 마지막으로 마스터 디바이스가 *APCS*로 전송하는 데이터에 패리티 에러(Parity error)를 부여하여 *APCS*가 이를 제대로 체크하는지를 확인해야 하는데 이를 위한 명령어가

〈표 3〉 데이터 전송을 위한 기본적인 명령어

명령어	의 미
MWrite	버스 마스터 모델로 하여금 APCS에 메모리 쓰기 명령어를 요청하게 함
MRead	버스 마스터 모델로 하여금 APCS에 메모리 읽기 명령어를 요청하게 함
CWrite	버스 마스터 모델로 하여금 APCS에 컨피규레이션 쓰기 명령어를 요청하게 함
CRead	버스 마스터 모델로 하여금 APCS에 컨피규레이션 읽기 명령어를 요청하게 함

〈표 4〉 메인 메모리와 로컬 메모리를 관리하기 위한 기본적인 명령어

명령어	의 미
MSave	메인 메모리의 특정 주소에 특정 데이터를 저장
MSaveDefault	메인 메모리 전체에 데이터를 무작위로 저장
MDisplay	메인 메모리의 특정 주소의 데이터를 화면으로 확인
LSave	로컬 메모리의 특정 주소에 특정 데이터를 저장
LSaveDefault	로컬 메모리 전체에 데이터를 무작위로 저장
LDisplay	로컬 메모리의 특정 주소의 데이터를 화면으로 확인
CompareMem	메인 메모리의 특정 주소의 데이터와 로컬 메모리의 특정 주소의 데이터를 비교

〈표 5〉 기타 프로토콜을 지원하기 위한 기본적인 명령어

명령어	의 미
MWaitMax	PCI 버스 마스터의 데이터 전송 시 발생하는 최대 지연 시간 설정
MWaitMin	PCI 버스 마스터의 데이터 전송 시 발생하는 최소 지연 시간 설정
LWaitMax	ALDS가 로컬 메모리와 데이터 전송 시 발생하는 최대 지연 시간 설정
LWaitMin	ALDS가 로컬 메모리와 데이터 전송 시 발생하는 최소 지연 시간 설정
MCmdWait Max	버스 마스터가 하나의 명령어를 수행하고 다음 명령어를 수행하는데 까지 걸리는 최대 지연 시간 설정
MCmdWait Min	버스 마스터가 하나의 명령어를 수행하고 다음 명령어를 수행하는데 까지 걸리는 최소 지연 시간 설정
MSend ParityErr	버스 마스터가 APCS에 패리티 에러가 있는 데이터를 전송하게 지시
MDetect ParityErr	버스 마스터가 APCS로부터 전송 받은 데이터가 패리티 에러가 있다고 보고

*MSendParityErr*이다. 또한 마스터 디바이스가 *APCS*로부터 전송 받은 데이터를 패리티 에러가 있음을 *APCS*에 보고해야 되는 상황도 발생할 수 있는데 이를 위한 명령어가 *MDetectParityErr*이다.

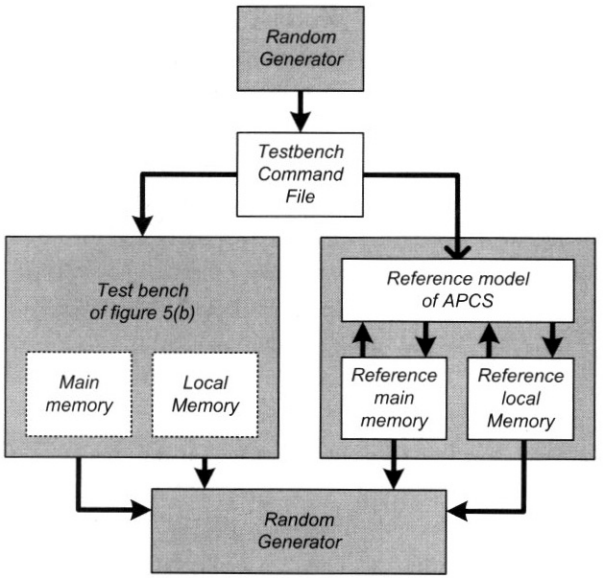
설계된 *APCS*와 *ALDS*는 제안된 테스트 벤치와 명령어를 이용하여 검증하고자 하는 시나리오를 완성하여 테스트 벤치의 명령어 파일로 저장하면 PCI 마스터 디바이스나 로컬 프로토콜 매니저가 이를 읽어 해당 명령어를 수행하게 된다.

본 논문에서는 검증 과정을 모두 3단계, 즉 기본 동작 검증, 설계 기반 검증, 그리고 랜덤 테스트 검증으로 나누었다.

첫 번째 단계는 설계된 하드웨어의 기본 동작 검증으로 *APCS*와 *APDL*이 PCI 명령어를 지원하는 지와 올바른 PCI 프로토콜 및 로컬 프로토콜로 동작하는지를 검증하는 단계이다. 아래에 명령어 파일의 간단한 예제가 나와 있다.

- *MSave 0x300, 0x12345678;*
- *PCI_Wait_Max 4;*
- *PCI_Wait_Min 2;*
- *MWrite 0x100, 0x300, 84;*
- *CompareMem 0x100, 0x300, 84;*

먼저 '*MSave*' 명령어에 의해 메인 메모리 300h 번지에 데이터 12345678h이 저장된다. 그리고 '*PCI_Wait_Max*'와 '*PCI_Wait_Min*'에 의해 PCI 버스 마스터 디바이스의 전송 지연 시간이 결정된다. 마지막으로 '*MWrite*'에 의해 PCI 마스터 디바이스는 먼저 메모리 300h 번지로부터 84개의 데이터를 읽어 타겟 주소 100h 번지와 함께 *APCS*에 전송할 것이다. 곧 *ALDS*는 다시 *APCS*로부터 데이터를 수신하여 *SRAM*의 100h 번지에 이들을 저장 할 것이다. 그 후 메인 메모리 300h 번지와 데이터와 *SRAM*의 100h 번지의 데이터 84개를 비교하게 된다. 그 결과 이상이 있는 경우 에로로 보고된다.



(그림 7) 랜덤 테스트를 위한 검증 환경

두 번째 단계는 설계 기반 검증이다. 설계된 *APCS*와 *ALDS*의 데이터패스(Datapath), 컨트롤유닛(Control Unit)을 모든 패스(path), 특히 상태도(State machine)의 모든 분기를 트랙(Track)할 수 있도록 명령어들을 구성하여 검증한다. 이는 설계된 데이터패스, 컨트롤유닛, 그리고 상태도에서 실제로 절대로 발생하지 않는 조건들이 설계 단계에서 삽입되었는지를 검증하기 위해서이다.

마지막 단계는 설계된 하드웨어의 동작의 코너 케이스 등을 효과적으로 검증하기 위한 방법으로 앞의 두 단계를 통해서 발견되지 못한 오류를 위한 랜덤 테스트 검증이다[12]. 이를 위해 본 논문에서는 참조모델, 랜덤 발생기, 그리고 비교엔진을 가지는 랜덤 환경을 (그림 7)과 같이 제안한다. 여기서 참조모델은 명령어 파일을 처리하기 위한 *APCS*와 *ALDS*의 동작모델로 클럭에 상관없이 동작된다. 먼저 랜덤 발생기는 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 적용된다. 마지막으로 비교엔진은 메인 메모리의 데이터와 참조모델의 메인 메모리의 데이터를 그리고 SRAM의 데이터와 참조모델의 로컬 메모리의 데이터를 각각 비교한다.

3.3 성능 측정

지금까지 소개되어진 명령어들은 설계되어진 *APCS*와 *ALDS*를 검증하는 데 사용되어질 뿐 아니라, 지연전송과 프리페치 요구를 이용한 방법의 성능 비교를 테스트 하는데도 사용되어 진다. 성능을 측정하는 시나리오 및 시뮬레이션 조건은 앞선 연구[9]에서 자세히 소개되어 있다.

첫 번째로, 버스 마스터가 메모리 읽기 멀티플 명령어를 이용하여 데이터 전송을 요구할 때 전송 데이터의 크기를 증가하면서 그때 소요되는 클럭 수를 측정한 것이다. 이때 지연 트랜잭션이 적용된 경우 타겟 장치가 로컬 디바이스로부터 프리페치 할 데이터의 크기를 증가시키면서 시뮬레이션 하였다.

두 번째로, 지역 버스 지연에 따른 데이터 전송 속도를 나타낸 것이다. 이때 전송할 데이터의 크기 및 지연 트랜잭션이 적용된 경우의 프리페치 할 데이터의 크기를 최대로 하였다.

다음으로 버스 마스터가 내부적인 문제로 프로토콜을 종료해야 하거나 버스 사용을 장시간 점유 할 수 없어 프로토콜을 종료해야 하는 경우이다. 이때 역시 최대 크기의 데이터 전송을 요구하였으며, 이때 타겟에 지연 트랜잭션이 적용된 경우 프리페치 할 데이터의 크기도 최대로 하였다.

마지막으로 시나리오는 전송할 데이터의 크기를 무작위로 선택되도록 하였다.

4. 실험결과

4.1 성능 측정 결과

설계된 하드웨어를 이용하여 성능을 측정하기 위해 전송할 데이터의 크기, 로컬 버스의 지연을 무작위로 선택하도록 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. 버스 마스터 장치가 타겟 장치로부터 데이터를 4 바이트만큼 읽고자 할 땐 메모리 읽기 명령어를, 16 바이트 이하를 읽고자 할 땐 메모리 읽기 라인 명령어를, 그리고 그 이상의 데이터를 읽고자 할 땐 메모리 읽기 멀티플 명령어가 사용하도록 하였다. 시뮬레이션 결과 프리페치 요구를 이용한 제안된 방법은 지연전송이 적용된 것에 비해 약 9% 정도 향상되었음을 확인 할 수 있다. 이 결과는 단순한 동작 모델을 이용하여 시뮬레이션을 한 앞선 연구[9]의 10% 성능 향상 결과와 거의 같음을 알 수 있다.

4.2 검증 결과

설계된 하드웨어는 기본 동작 검증과 설계 기반 검증을 실행하였다. <표 6>는 검증 결과로 나타난 오류들을 설계 관점에서 통계 내어 놓은 것이다. 외부 인터페이스의 프로토콜을 관리하고 설계된 데이터 패스를 제어하기 위한 상태도에서 가장 많은 오류가 발생함을 확인할 수 있다.

〈표 6〉 기본 동작 검증과 설계 기반 검증 결과

구 분	오류 개수
코딩 에러 및 오타	32
데이터 패스 추가	4
데이터 패스 신호 추가 및 수정	8
컨트롤 유닛 신호 추가 및 수정	18
상태도 추가	6
상태도의 제어 신호 추가 및 수정	58
합 계	126

〈표 7〉 랜덤 테스트 검증 결과

구 분	오류 개수
APCS의 PCI 프로토콜 어김	4
APCS의 로컬 프로토콜 어김	2
ALDS의 로컬 프로토콜 어김	2
기 타	4
합 계	12

다음은 랜덤 테스트 검증을 위해 10,000개의 데이터 전송 명령어를 수행하여 랜덤 환경에서 시뮬레이션 한 결과로 <표 7>에 나타내었다. 즉 앞서 동작 검증과 설계 기반 검증을 통해서 찾지 못하는 오류를 랜덤 테스트 검증을 통하면 쉽게 찾을 수 있음을 알 수 있다.

설계되어진 *APCS*와 *ALDS*는 PCI 2.2 컨트롤러의 동작 주파수인 33MHz에서 동작됨을 확인 할 수 있었다. *APCS*는 내부에 프리페치 버퍼를 따로 두지 않고 단순한 프로토콜 변환기로써 설계되어 졌기 때문에 크기 또한 작기 때문에 시스템 온칩(SoC)의 IP로써 사용되어 질 수 있을 것으로 보인다. 또한 *ALDS* 역시 단순한 SRAM 인터페이스를 지원하는 로컬 디바이스 예로, 응용 분야에 따라 설계자에 의해 차후 설계되어 질 수 있다.

5. 결 론

본 논문에서는 앞선 연구에서 소개된 프리페치 요구를 이용하여 데이터 전송 효율을 향상시키는 방법을 지원하는 PCI 타겟 컨트롤러와 로컬 디바이스를 설계하였다. 설계된 PCI 타겟 컨트롤러는 간단한 로컬 인터페이스를 가질 뿐 아니라 PCI 2.2를 전혀 모르는 사용자도 쉽게 PCI 인터페이스를 지원할 수 있도록 설계 되었다. 또한 설계된 하드웨어를 효과적으로 검증하기 위해 기본 동작 검증, 설계 기반 검증, 그리고 랜덤 테스트 검증을 제안하였다. 이러한 검증을 위해 테스트 벤치와 테스트 벤치를 동작시키기 위한 명령어를 제안하였다. 또한 랜덤 테스트를 위해 참조모델, 랜덤발생기, 비교엔진으로 구성된 테스트 환경을 구축하였으며 이를 이용해 코너 케이스를 효과적으로 검증하였다. 즉, 랜덤 테스트 환경을 통해 설계되어진 하드웨어를 검증한 결과를 비교함으로써 일반적인 테스트 벡터에서 발견하기 어려운 12개의 오류들을 발견할 수 있었다. 또한 제안된 테스트 환경을 통해 시뮬레이션 한 결과, 프리페치 요구를 이용한 제안된 방법이 지연 전송에 비해 데이터 전송 효율이 평균 9% 향상되었다.

참 고 문 헌

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

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

[11] Cadence, Verilog-XL Reference version 3.4, Cadence, 2002.

[12] Michael Keating and Pierre Bricaud, Reuse Methodology manual for Soc designs, Kluwer Academic Publishers, 1999.

현 유 진

e-mail : braham@dgist.ac.kr

1999년 영남대학교 전자공학과(학사)

2001년 영남대학교 전자공학과(공학석사)

2005년 영남대학교 전자공학과(공학박사)

2005년~현재 DGIST IT연구부 HW연구팀 연구원

관심분야 : 디지털 시스템, SoC, 컴퓨터 입출력 컨트롤러, 임베디드 시스템

성 광 수

e-mail : kssung@yu.ac.kr

한양대학교 전자공학과(학사)

한국과학기술연구원 전자공학과(석사)

한국과학기술연구원 전자공학과(박사)

현 재 영남대학교 공과대학 전자정보공학부 부교수

관심분야 : 디지털 시스템, 집적회로 및 CAD, 임베디드 시스템