H.264/AVC 인코더 용 PCI 인터페이스의 구현

정회원 박 경 오*, 김 태 현*, 종신회원 황 승 훈*, 홍 유 표*°

An Implementation of a PCI Interface for H.264/AVC Encoder

Kyoungoh Park*, Taehyun Kim* *Regular Members*,
Seung-Hoon Hwang*, Youpyo Hong*° *Lifelong Members*

요 약

H.264/AVC 비디오 압축 표준은 DMB, 디지털 TV 및 각종 차세대 방송, 통신 및 가전 분야에 채택 되어 왔고, 최근 감시카메라용 DVR 분야에서도 사실상의 표준으로 자리 잡아가고 있다. PC 기반 DVR의 경우 PC와의 데이터 전송 채널은 통상적으로 PCI 버스를 이용하는 반면, SOC용으로 사용되는 H.264/AVC 코덱은 대개 AMBA 버스를 기반으로 하여 호스트 인터페이스가 수행된다. 본 논문에서는 AHB 버스를 시스템 버스로 이용하는 H.264/AVC 코덱을 효과적으로 PCI 버스로 연결해 주기 위한 인터페이스 모듈 설계 및 실험 결과를 제시하였다.

Key Words : PCI, AHB, H.264/AVC

ABSTRACT

H.264/AVC video compression standard has been adopted for DMB, digital TV and various next generation broadcasting, communication and consumer electronics applications, and modern DVR system is also based on H.264/AVC standard. Although PC-based DVRs use PCI bus for main interface typically, H.264/AVC codec for SOCs use AHB bus for host interface. In this paper, we present an implementation of PCI to AHB interface module for H.264/AVC codec to efficiently communicate with a PC and experimental results.

Ⅰ. 서 론

차세대 DTV, DMB 등에서 채택되어져 온 H.264/AVC는 최근 DVR 세트시장에서도 그 영향력을 넓혀가고 있다. DVR 세트는 크게 PC 카드 방식과 Stand-Alone 방식으로 나뉘는데, 이 중 PC 카드 방식은 PC의 PCI 버스 슬롯에 DVR 세트를 장착하여 구동하는 방식을 말한다. 통상적인 SoC 멀티미디어 코덱은 대중적인 SoC 임베디드 CPU인 ARM CPU에 호환 가능하도록 AMBA 2.0 AHB 버스 인터페이스를 외부 인터페이스로 사용한다. 본 연구에서 사용한 H.264/AVC 인코더도 AHB 버스 인터페이스를 사용하므로, 멀티미디어 코덱 용 PCI 버스 인터페이스의 구현은 PCI 버스와 AHB 버스 간 브릿지 회로를 설계하는 것을 의미한다. 본 연구에서는 PCI Specification 2.2 버전[1]을 참고하였다. [2-5]에서는 특정 목적의 PCI 버스 회로 설계를 제시하고 있으며, [6]에서는 AHB와 PCI 버스 간 브리지 회로의 설계를 제시하고 있으나 멀티미디어 코덱용이 아닌 다양한 목적의 브리지 회로를 제시 하고 있다. 본 논문에서는 AHB 버스 신호를 PCI 버스 신호로 바꾸어 주는 멀티미디어 코덱 용 PCI 버스 인터페이스 회로의 설계 및 구현 결과를 제시하였다.

※ 본 논문은 동국대학교의 지원으로 이루어졌음.
* 동국대학교 전자전기공학부 (yhong@dgu.edu)), (° : 교신저자)
논문번호 : KICS2010-07-340, 접수일자 : 2010년 7월 29일, 논문최종접수일자 : 2010년 9월 6일

Ⅱ. PCI 버스 인터페이스 구조

그림 1은 PCI 버스 인터페이스의 구조를 보여주고 있다. PCI 버스 인터페이스는 이니셰이터(initiator)와 타겟(target)으로 구성 되어 있으며 일반적인 버스 인터페이스와 비교 해 보았을 때, 이니셰이터는 마스터, 타겟은 슬레이브에 해당 한다. 이니셰이터가 수행하는 역할은 타겟에게 데이터 읽기/쓰기 요청, 쓰기 데이터 전송 및 컨트롤 신호를 만들어 주는 역할이다. 타겟이 수행하는 역할은 이니셰이터가 요청한 읽기/쓰기 요청을 받아들여 읽기 데이터 전송을 해 주는 역할이다. 호스트-PCI 브리지는 CPU 로컬 버스와 PCI 인터페이스 간의 통신을 담당하며, 이니셰이터와 타겟의 역할을 모두 수행한다. 또한, PCI 버스의 아비트레이션도 수행한다.

PCI 버스의 데이터 전송은 전송 목적에 따라 컨피규레이션 억세스와 데이터 억세스로 구분 된다. 컨피규레이션 억세스란 PCI 버스에 접속하는 디바이스의 특성, 종류, 동작 방식 등을 설정하는 것을 의미한다. 데이터 억세스는 일반적인 데이터 전송을 의미하는 것으로, 버스트 억세스를 지원한다. PCI 버스에 연결되어져 있는 디바이스들은 동일한 주소 체계를 가지므로 각 디바이스 별로 다른 주소를 할당 받아야 하는데, 이를 베이스 어드레스라고 하며 이는 호스트에서 설정 해 주게 되어 있다.

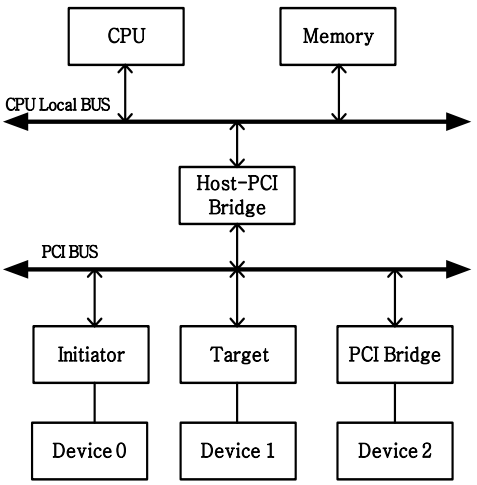

그림 1. PCI 버스 인터페이스 구조

Ⅲ. PCI 인터페이스 구현

PCI 인터페이스 설계 시 각 디바이스의 특성에 따라 이니셰이터 혹은 타겟중 하나를 선택하여 설계하나 간혹 이니셰이터와 타겟 모두를 설계하는 경우도 있다. 이니셰이터로 설계하는 경우는 해당 디바이스가 빈번한 읽기/쓰기 데이터 억세스를 할 때 이며, 타겟으로 설계하는 경우는 큰 용량의 데이터를 연속인 주소에 쓰거나 읽는 경우이다. 이 때 타겟으로 설계하는 경우 주소 계산 및 버스트 억세스 조절 등을 호스트에서 해 줌으로 호스트에서의 조절이 용이한 장점이 있다. 일반적인 경우 이니셰이터로 설계하는 경우가 많지만, 본 논문의 경우 H.264/AVC 인코더용 PCI 인터페이스를 설계하는 것이기 때문에 빈번한 데이터 억세스는 거의 없고, 대용량의 비트스트림을 메인 메모리의 비트스트림 공간의 연속적인 주소에 쓰는 것이 주된 목표인 관계로 타겟으로 설계를 결정하였다.

3.1 설계 블록

그림 2는 본 논문의 PCI 버스 인터페이스 모듈의 블록도이다. 각 블록 별 설명은 아래와 같다.

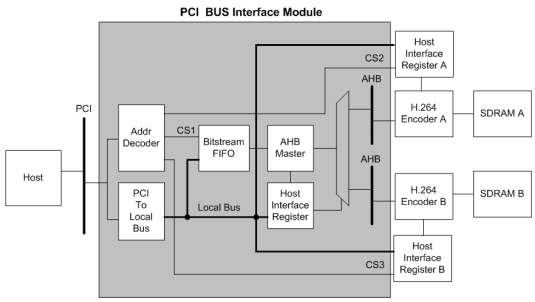

그림 2. PCI 버스 인터페이스 모듈 블록도

3.1.1 어드레스 디코더

PCI 인터페이스 모듈은 크게 비트스트림을 저장하는 비트스트림 FIFO 영역과 인코더 및 호스트 인터페이스 레지스터 영역으로 나눌 수 있다. 각 영역에 대한 접근은 PCI 버스 주소에 따라 결정되기 때문에 어드레스 디코더 모듈에서 PCI 버스의 주소 값을 이용하여 각 모듈별 칩 셀렉트(CS)를 조정한다.

3.1.2 PCI to 로컬 버스

PCI 버스 신호를 로컬 버스 신호로 변환 하거나 그 역 과정을 수행하는 역할을 하는 모듈이다. PCI 버스 신호는 어드레스/데이터 신호가 동일한 신호 선을 사용하기 때문에 어드레스 페이즈와 데이터 페이즈를 구분하여 어드레스/데이터 신호를 분리 해 주어야 한다. 또한, 데이터 신호의 경우 쓰기/읽기 신호가 동일한 신호 선으로 되어 있기 때문에 이를 입력 신호와

출력 신호로 구분 해주어야 한다. 로컬 버스 신호는 비트스트림 FIFO, 호스트 인터페이스 레지스터, 인코더용 호스트 인터페이스 레지스터에 접근 할 때 사용이 되는 신호이다.

3.1.3 비트스트림 FIFO

PCI 버스 인터페이스 모듈 내의 비트스트림 FIFO는 비트스트림이 저장된 메모리(SDRAM)에서 전송되어져 온 비트스트림을 임시로 저장하는 모듈이다. PCI 버스를 통해서 호스트에게 비트스트림을 전송 할 때 전송 속도를 높이기 위해서 버스트 모드(연속적인 데이터의 전송)를 사용 하는데, 이를 위해서는 호스트에게 건네 줄 데이터가 미리 일정 분량 이상 준비 되어 있어야 하므로 비트스트림 FIFO가 필요하다. 비트스트림 FIFO의 크기는 96x32bit로, PCI 버스 클럭과 AHB 버스 클럭의 주파수가 다르기 때문에 듀얼 포트 SRAM을 사용하였고, AHB 버스와 PCI 버스 인터페이스 모두 32bit 폭의 데이터 신호를 사용하기 때문에 32bit 폭의 FIFO를 사용 하였다. 또한, PCI 버스를 통해서 FIFO의 데이터를 가져 갈 때 FIFO 언더플로우가 발생 될 가능성을 줄이기 위해 크기를 96으로 결정했다.

3.1.4 AHB 마스터

SDRAM의 비트스트림을 AHB 버스를 통하여 비트스트림 FIFO에 전송하는 역할을 하는 모듈이다. 또한 AHB 신호를 로컬 버스 신호로 변환을 한다. 전송 속도 증가를 위해 AHB 마스터 모듈은 DMA기능을 지원하며, 버스트 8로 비트스트림을 읽어 온다.

3.1.5 호스트 인터페이스 레지스터

PCI 버스 인터페이스 모듈의 비트스트림 전송 설정에 관련된 정보가 저장된 모듈로서, 호스트는 비트스트림의 크기 정보, 시작 주소 정보, SDRAM A를 택할 것인지 B를 택할 것인지에 대한 정보와 전송 시작 명령을 해당 레지스터에 설정하게 되면 설정 값이 AHB 마스터에 입력이 된다. 입력 후 AHB 마스터는 설정된 시작 주소부터 설정된 크기만큼의 비트스트림을 SDRAM A 또는 B로부터 비트스트림 FIFO로 읽어 온다.

3.2 비트스트림 전송 규약

그림 3에서는 AHB 버스 인터페이스와 PCI 버스 인터페이스 간의 연결을 나타내었다.

인코더는 인코딩 시 비트스트림이 생성되면 SDRAM 비트스트림 영역의 시작 주소부터 연속된주

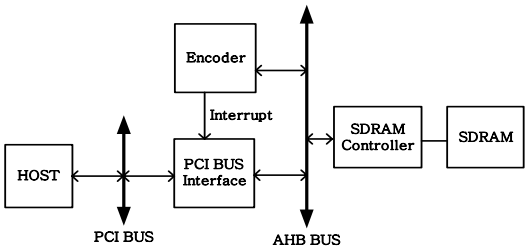

그림 3. 인터페이스 연결도

소로 저장 한다. 그렇게 한 프레임에 대한 인코딩이 완료 되면, SDRAM의 비트스트림 영역에는 한 프레임에 대한 비트스트림이 저장 되어있게 된다. 한 프레임에 대한 비트스트림 저장이 완료 되면 인코더는 인터럽트를 PCI 버스 인터페이스 모듈로 보낸다. 인터럽트를 받은 PCI 버스 인터페이스 모듈은 AHB 버스를 통해서 비트스트림의 시작 주소와 크기 정보를 읽어온 후 호스트 인터페이스 레지스터에 시작 주소 및 크기 정보를 입력한다. 이와 동시에 PCI 인터페이스 모듈은 PCI 버스를 통해서 호스트에게 인터럽트를 전달하고, 인터럽트를 인지한 호스트는 호스트 인터페이스 레지스터를 읽어, 읽어 와야 할 비트스트림의 시작 주소와 크기를 확인한다. 시작 주소와 크기를 확인 한 호스트는 PCI 인터페이스 모듈의 AHB 마스터 모듈에 읽어 와야 할 비트스트림의 시작 주소와 크기 정보를 입력한 후 AHB 마스터에 읽기 DMA를 시작하라는 명령을 내린다. AHB 마스터는 입력된 비트스트림 시작 지점부터 비트스트림의 크기 만큼 AHB 버스를 통해 SDRAM으로부터 비트스트림을 읽어온 후 비트스트림 FIFO에 저장한다. 이 때 AHB 마스터는 FIFO가 오버플로우 되지 않도록 비트스트림 FIFO의 여유 공간이 AHB 마스터가 사용하는 버스트 크기인 버스트 8로 읽어올 수 있는 크기 8*32bit 이하가 되면 비트스트림을 읽어오는 동작을 잠시 멈춘 후 여유 공간이 8*32bit을 초과하면 읽기 동작을 개시한다. 호스트는 AHB 마스터에 읽기 DMA 개시 명령을 내린 직후부터 비트스트림 FIFO에서 비트스트림을 읽어 온다. 호스트는 한 프레임에 해당하는 비트스트림을 모두 읽어 오면 다음 프레임에서 위의 동작을 반복한다.

3.3. 비트스트림 FIFO 언더플로우 방지

전술한 비트스트림 전송 규약에서 호스트가 비트스트림을 비트스트림 FIFO에서 읽어올 때, 호스트가 비트스트림 FIFO의 비트스트림을 읽어 가는 속도 보다 AHB 마스터가 비트스트림 FIFO에 비트스트림을 쓰

는 속도가 느린 경우가 발생 될 수 있다. 이 상태가 유지 될 경우 비트스트림 FIFO에 언더플로우가 발생 될 수 있다. 이러한 언더플로우의 방지를 위해 PCI 버스 표준[1]의 리트라이(Retry), 디스커넥트(Disconnect) 기능을 이용 하였다.

리트라이란 호스트가 읽기/쓰기 억세스를 요청하였을 때 타겟이 억세스를 받을 준비가 되어 있지 않은 경우, 타겟이 호스트에게 준비가 되어 있지 않음을 알린 후 재전송을 요청하는 기능이다. 이 기능을 이용하여 호스트가 비트스트림 전송을 개시 한 시점에 비트스트림 FIFO에 데이터가 8*32bit 이상 저장되어 있지 않은 경우, 리트라이 기능을 사용하였다. 8*32를 기준으로 삼은 이유는 호스트가 비트스트림을 비트스트림 FIFO에서 읽어올 때 버스트 8로 읽어오기 때문 이다.

디스커넥트는 리트라이와 유사하지만 전송을 개시한 시점이 아니라 버스트 모드로 읽기/쓰기 중에 타겟이 더 이상 데이터를 받지 못하거나 보내줄 데이터가 없는 경우, 호스트에게 억세스 중단을 요청하는 기능이다. 이 기능을 이용하여 비트스트림 전송 도중 비트스트림 FIFO의 잔여 데이터가 8*32bit 이하이고, 메모리에서 비트스트림을 모두 가져오지 못했을 경우 디스커넥트 기능을 이용하여 억세스를 중단하도록 했다. 해당 기능들은 비트스트림 FIFO에 남아있는 데이터의 개수를 확인하여 언더플로우가 발생하기 전에 리트라이, 디스커넥트 기능을 사용 하도록 구현 하였다.

Ⅳ. 구현 및 실험 결과

제안된 PCI 버스 인터페이스 모듈은 베릴로그를 이용하여 구현 되었으며, 그림 4는 시뮬레이션을 통해 측정 된 초당 데이터 전송률을 보여주고 있다. PCI 버스의 동작 주파수는 33/66MHz 중 선택이 가능토록 규정되어 있으며[1], 본 실험에서는 33MHz를 사용하였다. 일단 PCI 타겟 모듈만의 독립적 성능측정하기 위하여 코덱 동작과 별도로 PCI 모듈만을 시뮬레이션 하였으며, AHB 버스의 동작 주파수를 바꾸어 보며 최대 데이터 전송률을 측정하였다.

실험 결과, AHB 버스의 동작 주파수에 따라서 최대 전송 비트율이 증가하다가 110MHz부터 더 이상 증가하지 않았다. 이는 110MHz 부터는 AHB 버스의 데이터 전송률이 PCI 버스의 데이터 전송률보다 높아져 PCI 버스의 데이터 전송률이 한계치에 도달한 것으로 측정되었다. PCI 버스 표준에서 규정 된 이론적으로 보장 된 최대 전송 속도는 1,056Mbps 이며[1], 설계된 PCI모듈의 시뮬레이션 상 최대 전송 속도는 993Mbps로 측정되었다.

H.264/AVC 인코더와의 연동 테스트를 위하여 FPGA 테스트를 진행하였으며, 테스트 시 사용한 보드는 그림 5와 같으며 자일링스사의 FPGA인 XC4VLX100 칩을 사용하고 호스트는 NXP사의 PNX1702 칩을 사용하였다. FPGA 합성 결과 AHB의 동작 주파수가 40MHz로 나타나, PCI 클럭은 33MHz, AHB 클럭은 40MHz의 주파수로 인코더 연동 비트스트림 전송 테스트 결과 10Mbps의 비트스트림 실시간 전송이 가능함을 확인하였다. 시뮬레이션 결과에 비해 낮은 데이터 전송률을 보인 이유는 인코더에서 사용하는 메모리 대역폭에 따라서 비트스트림 전송을 위한 메모리 대역폭이 줄어들었기 때문으로 판단된다.

제안된 PCI 인터페이스 모듈의 사이즈는 TSMC 0.13μm 공정 라이브러리를 이용하여 시놉시스 디자인 컴파일러 툴로 합성한 결과 총 게이트 수가

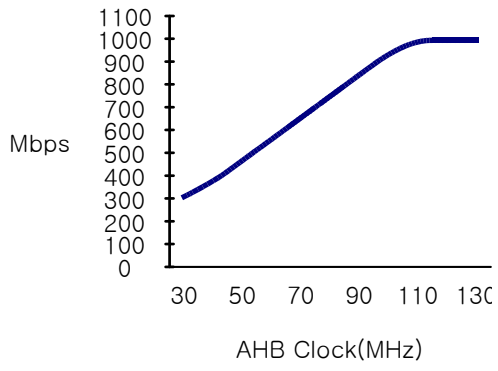

그림 4. 시뮬레이션 결과

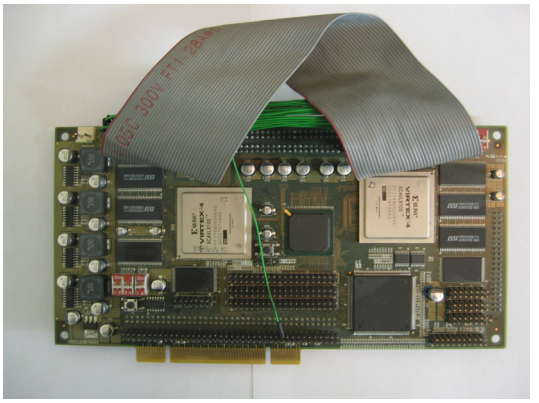

그림 5. PCI 인터페이스 회로 테스트용 FPGA 보드

13.371K로 측정되었으며, SRAM은 24Kbit 이다.

본 연구와 기존 연구를 비교 해 보았을 때, [3]은 메모리 억세스 시의 병목 현상을 줄이기 위해 FIFO를 사용하여 데이터를 메모리에서 프리페치 해 오는 방법을 제안한 연구로써 합성 결과 및 동작 결과를 제시하지 않고 있다. [4]의 경우 FPGA를 사용한 PCI 회로의 설계에 대해 제안한 것으로 합성 결과 및 동작 속도 등을 제시하지 않고 있다. [5]는 PCI 버스를 이용한 SDRAM 컨트롤러 설계에 대해 제안하고 있지만 정상 동작 여부만 제시하였으며 [6]에서도 AHB/PCI 버스 간 브리지 회로를 설계 하였으나 결과에 대해서는 정상 동작 여부만 제시되어 있어서 본 연구결과와의 정량적인 직접적 비교는 불가능하였다.

V. 결 론

본 연구를 통하여 H.264/AVC 인코더와 호스트와의 인터페이스 기능을 수행하는 PCI 인터페이스 회로가 설계되었다. PCI 인터페이스 모듈 내부의 비트스트림 FIFO와 버스트 모드를 이용함으로서 최대 993Mbps의 데이터 전송 및 인코더 연동 비트스트림 전송 시 AHB 버스 동작 주파수 40MHz에서 10Mbps의 비트스트림 전송이 가능하도록 구현 되었다. 또한, 리트라이 및 디스커넥트 기능을 이용하여 비트스트림 FIFO의 언더플로우를 방지하였다.

참 고 문 헌

〔1〕 "PCI Local Bus Specification Revision 2.2", *PCI SIG*. 1998.

〔2〕 Edward Solari and George Willse, "PCI Hardware and Software architecture and design", 4th ed., Annabooks. 1998.

〔3〕 I. S. LEE, J. Y. Kang, H. Y. Kim, "병목 현상 제거 및 효율적인 PCI 회로 설계에 관한 연구" *한국통신학회 논문지*, 제27권, 4C호, pp. 365-370, 2002.

〔4〕 E.Finkelstein and S. Weiss, "implementation of PCI based systems using programmable logic" *IEEE Proceedings Circuits, Devices and Systems*, Vol.147, No.3, pp 171-174, 2000.

〔5〕 Q. Daqiang, H. Bing, L.Dandan, "Design of SDRAM Controller in High-Speed Data Acquisition Based On PCI Bus" 8th International Conference on Electronic Measurement and Instruments, Vol.1, pp.822-825, 2007.

〔6〕 W. Zhonghai, Y. Yizheng, W. Jinxiang, Y. Mingyan, "Designing AHB/PCI Bridge", *Proceedings of 4th International Conference on ASIC 2001*, pp. 578~580, 2001

박 경 오 (Kyoungoh Park) 정회원

2009년 2월 동국대학교 전자공학과 학사

2009년 3월~현재 동국대학교 전자공학과 석사 과정

<관심분야> 비디오 코덱 칩 설계

김 태 현 (Taehyun Kim) 정회원

2010년 2월 동국대학교 전자공학과 학사

2010년 3월~현재 동국대학교 전자공학과 석사 과정

<관심분야> 비디오 코덱 칩 설계

황 승 훈 (Seung-Hoon Hwang) 종신회원

1999년 연세대학교 박사

1999년~2005년 LG전자 책임연구원

2003년~2005년 영국 사우스햄턴대학교 Visiting Research Fellow

2005년~현재 동국대학교 교수

<관심분야> 무선 및 이동통신 시스템 및 요소기술, cognitive radio, 밀리미터파통신

홍 유 표 (Youpyo Hong) 종신회원

1991년 2월 연세대학교 전기공학과 학사

1993년 5월 University of Southern California 전기공학과 석사

1998년 8월 University of Southern California 컴퓨터공학과 박사

1998년 7월~1999년 2월 Synopsys, Hillsboro, Senior Engineer

1999년 3월~현재 동국대학교 교수

<관심분야> 멀티미디어 칩 설계, SOC 설계