2003년도 대한전자공학회 하계종합학술대회 논문집 제26권 제1호

TCP/IP 하드웨어와 CPU 와의 통신을 위한 Host Interface 의 구현

정여진, *임혜숙
이화여자대학교 과학기술대학원 정보통신학과
컴퓨터 네트워킹 하드웨어 연구실

Host Interface Implementation for TCP/IP Hardware Accelerator

Yeojin Jung, *Hyesook Lim
Information Electronics Engineering in Ewha W. University
*hlim@ewha.ac.kr

Abstract

TCP/IP 를 포함하는 데이터 네트워킹 프로토콜을 구현함에 있어, 기존에는 소프트웨어 방식으로 구현되었던 모듈들을 하드웨어로 구현하는 프로젝트를 수행하면서, CPU 와 하드웨어 모듈과의 통신을 중계하는 모듈을 구현하였다. 본 논문에서는 TCP/IP 하드웨어와 CPU 와의 통신을 위한 Host Interface 의 기능에 대해 다루고 구현 방식을 Control flow 와 Data flow 의 입장에서 설명하였다. 우선, Host Interface 의 기능을 설명하고 Host Interface 의 입출력 신호를 정의하였다. Host Interface 에서 이루어지는 CPU 와 하드웨어 모듈간의 통신을 제어정보 흐름과 데이터정보 흐름으로 나누고 제어흐름을 위해서는 Command/Status Register 를 두었고, 데이터 흐름을 위해서는 CPU 와 데이터 RAM 사이에 FIFO 를 두어 데이터의 흐름이 신속히 이루어지도록 하였다. 끝으로 Host Interface 와 주변 모듈들간의 통신에 대한 Testcases 에 대해서도 다루었다.

I. Introduction

Local Area Network(LAN)[1]에서 널리 사용되는 Ethernet의 링크 스피드는 과거 10Mbps로부터 최근에는 수Gbps로 성장하였는데, 이는 마이크로프로세서의 속도를 능가하는 것이며, 이에 따라 소프트웨어로 구현된 TCP/IP의 프로세싱은 CPU에 있어서 커다란 부하로 작용하게 되었다[2]. 따라서 전체 시스템의 프로세싱 속도의 향상을 위하여 기존의 소프트웨어로 구현되어 왔던 TCP/IP를 포함하는 데이터 네트워킹 프로토콜 모듈들을 점차로 하드웨어로 바꾸어나가고 있는 추세이다. 기존의 소프트웨어로 구현되어 있는 모듈을 하드웨어로 구현하기 위해서는 CPU와 하드웨어로 구현된 모듈간의 통신을 고려해야만 한다. 하드웨어 모듈들 간의 통신은 직접 연결된 버스를 통하여 이루어지는 반면에, CPU와의 통신은 메모리의 읽기/쓰기를 통하여서만 가능하기 때문이다. 따라서 하드웨어와 CPU는 직접 연결된 버스를 통하여 통신할 수 없고, 하드웨어 모듈들과 CPU와의 통신을 위한 모듈이 필요하게 된다. 본 논문은 CPU와 하드웨어 간에 통신을 담당하기 위해 구현된 Host Interface(HI)에 대하여 다룬다. HI의 구현은 TCP/IP 하드웨어 구현 프로젝트의 일부분으로 수행되었다.

II. Functions of Host Interface

HI 는 하드웨어 모듈들과 CPU 사이에 위치하여 TCP/IP 하드웨어 모듈들과 CPU 간의 통신을 담당하는 모듈이다. HI 에서 이루어지는 CPU 와 하드웨어 모듈간의 통신은 제어정보 흐름(control flow)과 데이터정보 흐름(data flow)으로 나누어 볼 수 있다[3]. CPU 는 하드웨어 모듈에게 명령을 내릴 수 있고 하드웨어 모듈들의 상태를 알 수 있어야 한다. 또한 하드웨어 모듈들이 패킷을 만드는 데 필요한 헤더 정보들을 알려주거나 하드웨어 모

듈들로부터 처리된 헤더 정보들을 받을 수 있어야 한다. 반대로 하드웨어 모듈들도 CPU 에게 명령을 내릴 수 있고 상태를 알릴 수 있어야 하며, CPU 로부터 패킷을 만들기 위한 헤더 정보를 얻거나, 패킷을 parsing 하여 얻은 헤더 정보를 CPU 에게 알려주어야 한다. 또 패킷의 수신, 에러에 대한 통계자료를 CPU 에게 제공할 수 있어야 한다. 이러한 과정이 제어정보 흐름에 속한다. 반면에 데이터정보 흐름은 패킷의 payload 에 해당하는 순수 데이터를 말한다. CPU 가 요청한 주소를 확인하는 과정을 통해 제어정보 흐름인지 데이터정보 흐름인지 구별하여, HI 는 그에 맞는 역할을 수행하게 된다. HI 는 Command/Status Register(CSR), Rx RAM, Tx RAM 이렇게 3 개의 메모리와 연결되어 있는데[4], CSR 는 CPU 와 하드웨어 모듈간의 상태, 제어정보와 헤더정보의 통신에 이용되고, Rx RAM, Tx RAM 은 순수 데이터가 저장되는 장소이다. 본 논문에서 구현된 HI 에서는 주소범위 0~100 까지 100 개의 entry 를 CSR 에 할당하였고, Rx RAM 과 Tx RAM 에 주소범위 300~20299, 20599~40599 까지 각각 80kbyte 의 데이터를 저장할 수 있도록 하였다.

HI 와 다른 블록간의 통신은 그림 1 에서 볼 수 있듯이, 크게 HI 와 CPU 와의 interface, HI 와 Rx RAM, Tx RAM 과의 interface, HI 와 다른 하드웨어 모듈간의 interface 로 나누어 볼 수 있다. AMBA(Advanced Microcontroller Bus Architecture)규약을 기준으로 하여 설계된 HI 와 CPU 간의 interface 를 살펴보면, HI 는 CPU 의 요청을 처리할 준비가 되었을 때 ready 신호를 보내게 된다. CPU 는 데이터의 읽기 또는 쓰기를 위해 제어 신호와 함께 액세스하기를 원하는 메모리의 주소를 보낸다. CPU 는 HI 가 준비가 되면, 그 다음 클럭에 쓰고 자 하는 데이터를 보내거나 읽고자 하는 데이터를 받게 된다. HI 와

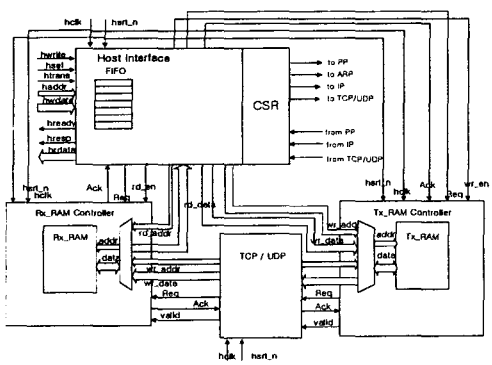

그림 1. **Host Interface** 와 주변 블록 간의 **Interface**

Data RAM 과의 interface 는 CPU 의 요청을 처리하기 위해 HI 가 RAM Controller 와 Handshaking 을 하고 데이터를 처리하는 부분이다. 또 HI 는 다른 하드웨어 모듈에게 CSR 에 저장된 제어정보를 제공하고 다른 하드웨어 모듈로부터 받은 상태정보를 CSR 에 쓰기 위한 interface 를 가지고 있다.

III. Control Flow

HI 는 CPU 나 하드웨어 모듈에서 들어온 명령, 상태정보, 헤더정보 등을 레지스터에 쓰고, 읽어 전달함으로써 제어정보 흐름을 관리하게 되는데, 이를 위하여 내부에 CSR 를 가진다. CSR 는 세 가지 종류의 레지스터로 나누어 지는데, Read Only(RO) Register, Read/Write(RW) Register, Clear on Read(COR) Register 가 그것이다. RO Register 는 CPU 는 읽기만 가능한 레지스터로, Rx source port number, Rx destination port number, Rx IP address, TOS, Protocol ID 등, 다른 하드웨어 모듈들이 CPU 에게 알려주고자 하는 헤더정보 등이 저장된다. RW Register 는 CPU 가 읽기 쓰기가 모두 가능한 레지스터로 Own MAC address, Own IP address, Own hardware type 등의 configuration register 들과 Tx destination address, Tx source port number, Tx destination port number 와 같은 패킷을 만들어 내보낼 때 필요한 정보들을 저장한 레지스터들이 이에 해당한다. 마지막으로 COR Register 는 다른 하드웨어 모듈들이 쓰고 CPU 가 읽어가는 레지스터로 status register, interrupt register, statistics register 등이 있으며, COR Register 는 CPU 가 읽어간 후에는 0 으로 clear 되어 CPU 가 읽어갔음을 나타내게 된다.

HI 에서의 제어정보 흐름은 다음과 같다. CPU 가 미리 정의된 Register Map 에 의하여 CSR 의 적절한 주소로 쓰기 명령을 내리면 HI 는 CPU 로부터 들어온 제어정보를 CSR 에 쓰고, 하드웨어 모듈은 레지스터에 직접 연결된 버스를 통해 제어정보를 전달 받는다. 반대로 하드웨어 모듈이 버스를 통해 상태정보를 CSR 에 쓰면 HI 는 CPU 가 그 주소의 데이터를 요청할 때 CPU 의 데이터버스에 데이터를 올려줌으로써 정보를 전달하게 된다. CPU 는 CSR 의 모든 레지스터를 읽어갈 수 있지만, COR Register 의 경우, 데이터를 읽어간 후에는 COR register 가 0 으로 clear 되는 동작이 뒤따라야 하고 쓰기 작업은 RW Register 에만 가능하다. 만약 CPU 가 RO Register 나 COR Register 에 쓰기 요청을 하면 HI 는 error response 를 띄우고 쓰기 요청은 무시된다.

IV. Data Flow

HI 는 Data Flow 를 처리함에 있어 AMBA AHB Slave 로 동작하도록 구현되었다. Rx RAM 과 Tx RAM 이 TCP/IP 하드웨어 칩 내부에 존재하여 CPU 의 요청에 따라 HI 가 적절한 동작을 하게 된다. 즉, CPU 가 AHB Master 가 되고 HI 는 AHB 의 Slave 가 되는 것이다. Data Flow 에는 CPU 가 RAM 에서 데이터를 읽어가는 Read Operation 과 CPU 가 RAM 에 데이터를 쓰는 Write Operation 이 있다. Read Operation 에 있어서 HI 는 데이터의 효율적인 제공을 위하여 HI 내부에 Rx FIFO 를 두고 CPU 와 Rx RAM 간 데이터 이동이 Rx FIFO 를 통해 이루어지도록 하였다. 또한 빠른 패킷의 처리를 위하여 Rx RAM 에서의 액세스 우선권은 TCP/UDP 하드웨어 모듈에 두었다. Write Operation 의 경우에도 CPU 와 Tx RAM 사이에 작은 사이즈의 Tx FIFO 를 두어 쓰기 작업이 원활히 이루어지도록 하였다. 각 Operation 에 대해서 자세히 살펴보면 다음과 같다.

■ Rx Direction

본 논문에서 구현된 HI 는 AMBA 규약에 따르고 있음을 앞에서 언급하였다. AMBA 규약에 따르면, Master 가 데이터를 요청하면 Slave 는 그 다음 클럭에 데이터를 받는 것을 기본으로 하고 있다[5]. 그림 2 는 AMBA AHB 의 multiple transfer 를 보여준다. Master 가 주소 A 에 대한 데이터 처리를 요청하면 Slave 는 ready 가 1 일 때, 바로 데이터를 처리하며, 바로 데이터를 처리할 수 없는 경우, ready 가 0 로 하여 wait state 를 삽입하도록 한다. CSR 을 액세스하는 경우 CSR 이 HI 의 내부에 존재하기 때문에 한 클럭에 데이터를 제공할 수 있지만 Rx RAM 과 같이 메모리를 사용하여 HI 외부에 메모리를 두는 경우에는 요청된 데이터를 바로 다음 클럭에 제공할 수 없는 문제점이 따른다. 본 논문에서 구현된 HI 에서는 CPU 의 데이터 요청 시 wait state 의 삽입 없이 효율적으로 데이터를 제공하기 위하여 CPU 의 데이터 요청 후 Rx RAM 에서 데이터를 직접 읽어오지 않고, 그림 3 과 같이 중간에 Rx FIFO 를 두어 미리 읽어온 데이터를 저장해 두었다가 CPU 가 데이터를 요청하면 Rx FIFO 에 있는 데이터를 제공하는 방식을 취하였다. Rx FIFO 는 8 개의 entry 를 가지고 있으며 유효한 데이터가 저장된 entry 수는 rx_fifo_counter 에 저장되어 있다. Rx FIFO 의 entry 가 8 개 이므로 rx_fifo_counter 의 값이 0 이면 Rx FIFO 가 비어 있는 상태이고, 8 이면 FIFO 가 유효한 데이터로 꽉 차 있는 상태를 나타낸다. Rx Direction 에 있어서 Rx FIFO 가 비어있지 않으면(유효한 데이터가 있으면) HI 는 ready 상태가 되고 CPU 의 읽기 요청했을 때 CPU 가 요청한 데이터가 Rx FIFO 에 저장된 데이터가 맞는지 주소를 확인한 다음, Rx FIFO 에서 데이터를 읽어와서 바로 데이터를 제공하게 된다. 동시에 HI 는 Background process 로 Rx RAM 에서 데이터를 읽어와 Rx FIFO 에 저장하는 일을 수행한다. 이 작업은 Rx FIFO 에 빈 entry 가 존재하는 한 수행된다. Rx RAM 에는 TCP/UDP 모듈이 패킷에서 처리한 데이터가 쓰여지거나 HI 가 Rx RAM 에서 데이터를 읽어가게 되고 이를 위하여 TCP/UDP 모듈과 HI 사이의 액세스 중재가 필요한데, Rx RAM Controller 가 그 역할을 담당한다. HI 는 Rx RAM 에서 데이터를 읽어오기 위하여 Rx RAM Controller 와 Handshaking 을 하여 Rx RAM 을 액세스할 수 있는 권리를 얻는다. Rx RAM 에의 액세스는 패킷의 빠른 처리를 위하여 TCP/UDP 모듈에 우선권을 두었다. 따라서 HI 는 TCP/UDP 모듈이 Rx RAM 을 액세스하고 있는 동안에는 Rx RAM 을 액세스할 수 없다. 이런 경우 CPU 에서 읽기 요청이 들어와도 Rx FIFO 가 비어있으면 CPU 의 요청을 바로 처리하지 못하고 wait state 를 띄워야만 한다. 또한 HI 가 Rx RAM 을 액세스하고 있는 동안에 TCP/UDP 모듈에서 액세스 요청이 들어오면 HI 는 하던 작업을 끝낸 후, 액세스 권리를 TCP/UDP 모듈로 넘겨야 한다.

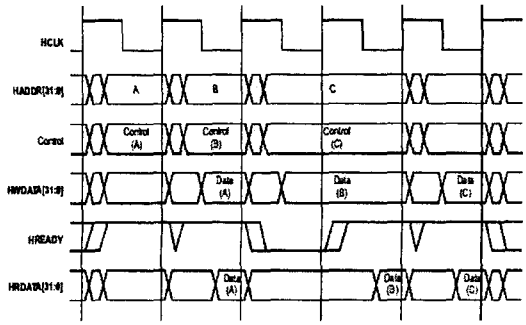

그림 2. AMBA AHB Multiple transfer

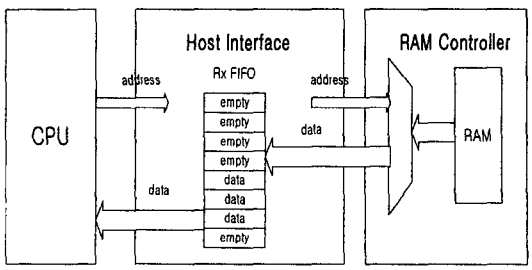

그림 3. Rx Direction Data Flow

■ **Tx Direction**

그림 4 에서처럼 Write operation 에서 사용되는 Tx FIFO 는 Rx FIFO 보다 작은 사이즈의 FIFO 를 사용하였다. Tx FIFO 역시 Rx FIFO 의 경우와 마찬가지로, tx_fifo_counter 에 의하여 entry 들이 관리된다. HI 는 Tx FIFO 가 비어있는 경우에 ready 가 되고 이때, CPU 에서 쓰기 요청과 주소가 들어오면 CPU 에서 들어온 주소를 확인한 후 Tx FIFO 에 들어오는 데이터를 쓰고, tx_fifo_counter 를 증가시킨다. 일단 Tx FIFO 에 유효한 entry 가 존재하면 HI 는 Tx RAM Controller 에 Tx RAM 의 액세스를 요청하게 되고, Tx RAM Controller 로부터 승인을 받으면 Tx FIFO 에서 데이터를 가져와 Tx RAM 에 쓰게 된다. 이와 같이 CPU 와 Tx RAM 간에 직접 데이터를 이동시키지 않고 Tx FIFO 를 거치도록 하면 HI 가 Tx RAM Controller 에게 액세스를 요청하고 승인 받아 데이터를 쓰기 시작하기까지 소요되는 두 세클럭 동안에도 CPU 에게 wait state 를 띄우지 않고 연속적으로 데이터를 받아 처리할 수 있게 되어 CPU 의 데이터 쓰기 작업을 원활하게 한다. 그러나 Tx RAM 액세스의 경우에도 HI 보다 TCP/UDP 하드웨어 모듈에 우선권이 있기 때문에 TCP/UDP 모듈이 Tx RAM 을 액세스 하고 있으면 기다려야 하고, HI 가 Tx RAM 을 액세스하고 있는 동안에도 TCP/UDP 모듈로부터 액세스 요청이 들어오면 HI 는 하던 작업을 마친 후, 메모리 액세스 권한을 넘겨야 한다.

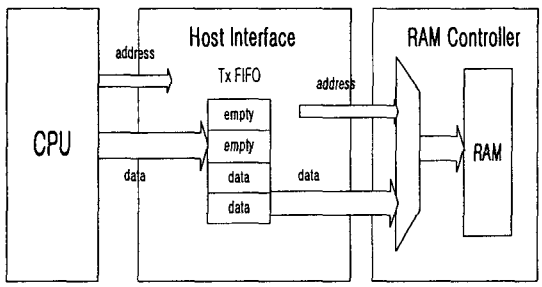

그림 **4. Tx Direction Data Flow**

V. Testcases

베릴로그로 구현한 HI 모듈을 검증하기 위하여 여러 케이스를 테스트하여 보았다. 테스트케이스들은 세 부분으로 나누어져 이루어졌는데 우선 CSR 의 동작을 검증하기 위하여 RW Register, COR Register, RO Register, 각각에 대하여 읽기 쓰기 요청을 하여 그에 따른 동작을 살펴보았다. 그 다음으로는 FIFO 의 동작을 검증하였다. Rx FIFO 가 비어있으면 Rx RAM Controller 로 request 를 보내 데이터를 읽어오고 Rx FIFO 의 데이터를 CPU 의 요청에 따라 제공하는지를 각각 테스트하고, CPU 에 데이터를 제공하는 동시에 Rx RAM 에서 데이터를 읽어와 Rx FIFO 에 쓰는지를 테스트 해 보았다. Tx FIFO 의 경우에도 CPU 에서 쓰기 요청을 받으면 CPU 로부터의 데이터를 Tx FIFO 에 쓰고 Tx RAM Controller 에 request 를 보내서 Tx RAM 에 데이터를 옮기는 과정을 각각, 그리고 동시에 제대로 동작하는지 테스트 하였다. 마지막으로 HI 와 Ram Controller 와의 동작을 테스트하였다. HI 의 Handshaking 과정과, TCP/UDP 모듈의 액세스 요청에 따른 액세스 양도가 제대로 이루어지는지 검토하였다.

VI. Conclusion

CPU 와 하드웨어로 구현된 모듈간의 통신을 위하여 AMBA AHB 버스에 기준한 HI 를 구현하였다. CPU 부터 들어오는 제어 정보들은 HI 의 내부에 존재하는 CSR 를 통해 하드웨어 모듈에 전달되도록 하였다. 데이터 정보의 경우에는 데이터 RAM 과 CPU 사이에 FIFO 를 구현하여, CPU 와 RAM 사이의 데이터 흐름이 FIFO 를 거쳐가게 함으로써 CPU 의 읽기/쓰기 요청 시 HI 가 RAM 과의 Handshaking 을 하는 동안에도 CPU 의 요청을 신속하게 처리할 수 있도록 하였다. 본 논문에서는 HI 가 CSR 액세스와 Data RAM 액세스 모두 AHB Slave 로 동작하도록 구현하였다. 앞으로는 HI 가 AHB Master 로 동작하여 CPU 를 거치지 않고 직접 외부에 있는 데이터 메모리로 Data 를 전송하는 DMA 방식을 사용하도록 구현해 보고자 한다.

Reference

[1] James F.Kurose, Keith W. Ross, " Computer Networking : A Top-Down Approach Featuring the Internet" , Addison Wesley, 2002

[2] " Introduction to TCP/IP Offload Engine(TOE)" , http://www.10gea.org

[3] 진교홍, 이정태, " 고속통신을 위한 TCP/IP 프로토콜의 하드웨어 설계 및 구현" , 한국정보과학학회지, Vol.12, No.1, pp135-153, Feb.1998.

[4] Wiznet 홈 페이지 hp://www.wiznet.co.kr

[5] AMBA™ Specification (Rev 2.0)