TCP/IP Hardware Accelerator를 위한 Host Interface의 설계

준회원 정 여 진*, 정회원 임 혜 숙**

Host Interface Design for TCP/IP Hardware Accelerator

Yeojin Jung* *Associate Member*, Hyesook Lim** *Regular Members*

요 약

빠른 데이터 처리를 위하여 기존에는 소프트웨어방식으로 구현되었던TCP/IP를 고속의 하드웨어로 구현함에 있어, TCP/IP 하드웨어와 외부 블록간의 통신을 중계하는 블록인 Host Interface를 구현하였다. Host Interface는 TCP/IP 하드웨어와 외부 블록의 중간에 위치하여 외부 블록과의 통신을 위해 AMBA AHB 규약을 따른다. Host Interface는 내부의 Command/Status Register를 통하여 CPU와 TCP/IP 하드웨어 간의 명령, 상태, 헤더 정보 등을 전달하는데 이 때에는 AMBA AHB의 Slave로서 동작한다. Data Flow를 위해서 Host Interface는 AMBA AHB의 Master로서 동작하는데, 데이터 흐름의 방향에 따라 Data Flow는 데이터를 수신하는 Receive Flow와 데이터를 패킷으로 만들어 보내는 Transmit Flow로 나뉜다. Rx Flow의 경우, UDP 블록이나 TCP Buffer로부터 받은 데이터를 내부의 작은 RxFIFO를 통해 외부 RxRAM에 써서 CPU가 읽어갈 수 있도록 하고, Tx Flow의 경우에는 외부 TxRAM에서 전송할 데이터를 읽어 와서TxFIFO를 거쳐 UDP Buffer나 TCP Buffer에 씀으로써 패킷을 만들어 보내도록 한다. 외부 RAM의 액세스에는 Command/Status Register에 위치한 Buffer Descriptor의 정보를 이용하게 된다. Host Interface는 이러한Data Flow의 원활한 흐름을 위해서 여러 세부 기능들을 수행하게 된다. Host Interface의 기능을 검증하기 위하여 여러 testcase들이 수행되었으며, 0.18 마이크론 기술을 사용하여 synthesis한 결과, 내부의 Command/Status Register와 FIFO를 모두 포함하여 약 173K 게이트가 소요됨을 보았다.

Key Words : TCP/IP Hardware Accelerator

ABSTRACT

TCP/IP protocols have been implemented in software program running on CPU in end systems. As the increased demand of fast protocol processing, it is required to implement the protocols in hardware, and Host Interface is responsible for communication between external CPU and the hardware blocks of TCP/IP implementation. The Host Interface follows AMBA AHB specification for the communication with external world. For control flow, the Host Interface behaves as a slave of AMBA AHB. Using internal Command/Status Registers, the Host Interface receives commands from CPU and transfers hardware status and header information to CPU. On the other hand, the Host Interface behaves as a master for data flow. Data flow has two directions, Receive Flow and Transmit Flow. In Receive Flow, using internal RxFIFO, the Host Interface reads data from UDP FIFO or TCP buffer and transfers data to external RAM for CPU to read. For Transmit Flow, the Host

* 이화여자대학교 SoC Design Lab. (surya@ewhain.net)
** 이화여자대학교 SoC Design Lab. (hlim@ewha.ac.kr)
논문번호 : 040091-0225, 접수일자 : 2004년 2월 26일
※ 본 연구는 삼성전자의 지원으로 수행되었습니다

Interface reads data from external RAM and transfers data to UDP buffer or TCP buffer through internal TxFIFO. TCP/IP hardware blocks generate packets using the data and transmit. Buffer Descriptor is one of the Command/Status Registers, and the information stored in Buffer Descriptor is used for external RAM access. Several testcases are designed to verify TCP/IP functions. The Host Interface is synthesized using the 0.18 micron technology, and it results in 173 K gates including the Command/Status Registers and internal FIFOs.

Ⅰ. 서 론

인터넷은 세계 각지에 흩어져 있는 엔드 시스템들을 연결하여, 서로 데이터 교환이 가능하게 한 거대한 컴퓨터 네트워크로서 인터넷에 연결된 엔드 시스템의 수적인 면에서 지난 10년간 매년 두 배에 가까운 빠른 성장을 거듭하고 있다. 하루가 다르게 발전하고 있는 인터넷에서 고속의 데이터 처리를 요구하는 응용 프로그램의 증가와 더불어 이더넷으로 연결되어 있는 링크 속도가 Gigabit 이상으로 빨라짐에 따라, 소프트웨어로 구현된 TCP/IP[1] processing은 전체 통신 시스템에 과중한 부하를 안겨주게 되었으며, 새로운 병목점으로 등장하게 되었다[2][3].

본 논문은 '고속의 데이터 처리를 위한 TCP/IP hardware accelerator 구현' 프로젝트의 일부분으로 TCP/IP 하드웨어와 CPU, 외부 RAM과의 통신을 위한 Host Interface를 다룬다. Host Interface(HI)는 그림 1에 보이는 바와 같이, CPU와 외부 RAM, TCP/IP 하드웨어와의 통신을 담당하는 블록이다. 하드웨어로 구현된 블록들 간의 통신은 직접 연결된 버스를 통해 이루어지는 반면에 CPU와의 통신은 메모리 읽기/쓰기를 통해서 이루어진다. 따라서 CPU와 TCP/IP 하드웨어 간의 통신을 위해서는 매개 역할을 하는 블록이 필요하게 되는데, 이러한 기능을 담당하는 블록이 HI이다. HI와 CPU, 외부 RAM 간의 통신은 AMBA (Advanced Microcontroller Bus Architecture) 규약[4]을 기준으로 설계되었다.

HI는 control flow와 data flow를 가진다. HI가 AMBA Bus의 Slave로서 동작하는 Control flow는 CPU로부터 하드웨어 블록들에 내려지는 명령이나 보내는 패킷을 위한 헤더 정보, 혹은 하드웨어 블록들로부터 CPU로 보내지는 상태 정보 및 받은 패킷의 헤더 정보 등 패킷의 송수신에 필요한 control 신호들을 전송하기 위한 것으로, HI 내부의 Command/Status Register(CSR)을 통해 통신이 이루어진다. 수신한 패킷의 데이터 또는 송신할 패킷의 데이터에 해당하는 data flow는 외부 RAM과 TCP사이의 flow로 HI는 외부 RAM과 TCP/IP 하드웨어 사이에서 data를 전달하는 역할을 한다. 이 때에는 HI가 AMBA Bus의 Master로서 동작하며, 외부 RAM은 Slave로서 동작한다.

본 논문은 다음과 같이 구성되었다. 2장에서 HI가 외부 인터페이스와의 통신에 따르는 AMBA AHB Specification에 대하여 다룬 후, 3장에서 HI의 동작을 TCP와의 Receive(Rx) Flow와 Transmit (Tx) Flow, 그리고 AMBA AHB와의 동작으로 나누어 설명한다. 4장에서는 HI와 다른 블록 간의 인터페이스에 대해 다루었다. 5장에서 HI 구현 후 테스트한 케이스들에 대하여 언급하고 그 Synthesis Report를 6장에서 보였다. 끝으로 7장에서 결론을 맺는다.

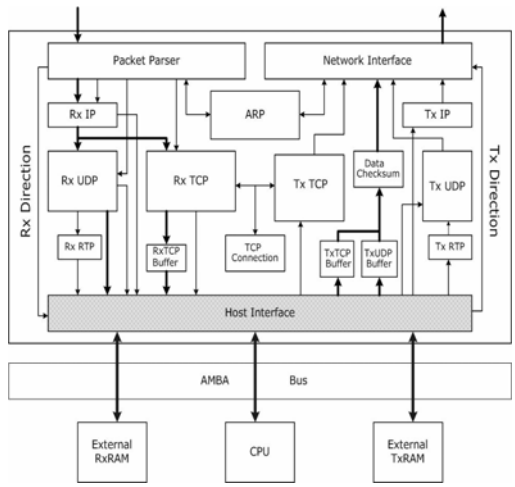

그림 1. TCP/IP hardware accelerator의 블록다이아그램

Ⅱ. AMBA Specification

본 논문에서 구현된 HI는 TCP 하드웨어가 외부와의 통신을 위해 AMBA 규약을 따름을 1장에서 언급하였다. AMBA AHB를 통해 통신을 하는 블록들은 그 역할에 따라 Master와 Slave로 나뉘게 된다. Master는 통신을 위해 AMBA Bus의 사용을 요청하여 액세스권을 얻은 블록이고, Slave는 Master의 요청에 의해 Master와 통신을 하게 되는

블록이다. Master와 Slave간의 통신은 그림 2의 신호들을 통해서 이루어진다. 우선 AMBA Bus를 사용하기를 원하는 블록은 AMBA Arbiter에게 bus 사용을 위한 요청을 하게 되고, 그에 대한 hgrant를 받음으로써 통신이 가능하게 된다. Hgrant를 받은 Master는 Slave에 control 신호들과 함께 읽거나 쓰기를 원하는 데이터와 주소를 보내고 Slave는 이에 대한 응답으로 Slave의 상태를 나타내는 신호들을 내보내게 된다. 본 논문에서 구현된 HI에서는 HI가 Master로 동작하는 경우, bus 점유를 보장 받기 위해 hbusreq와 함께 내보내는 hlock 신호는 0으로 고정하였다. 그밖에 bus의 size를 의미하는 hsize는 word를 사용하였고 htransfer는 IDLE, NONSEQ, SEQ를 지원하였다. IDLE은 데이터의 전송이 이루어지지 않고 있는 상태를 의미한다. NONSEQ는 한 번에 전송하는 데이터 chunk의 첫 데이터임을 나타내고, 그 이후의 데이터는 SEQ가 된다. Hburst는 하나의 워드 전송을 의미하는 SINGLE, 한번에 4 words씩 전송하는 INCR4만을 지원하도록 하였다. Hready는 Master의 요청에 대한 Slave의 처리상태를 나타내며, hresp는 OKAY와 ERROR만을 지원한다. 그림 3은 AMBA AHB의 multiple transfer의 timing을 보여 준다. AMBA AHB의 Master는 AMBA AHB Arbitration 블록으로부터 hgrant를 받으면 주소와 함께 control 신호들을 내보내게 되고 Slave가 ready이면 다음 클럭에 쓰려는 데이터를 내보내거나 읽고자 하는 데이터를 받게 된다. 그림 3에서 보이는 바와 같이 Slave가 ready가 되지 않은 경우(hready 신호가 low인 경우)에는 Slave가 ready가 될 때까지 주소, control신호들과 쓰려는 데이터를 유지하고 있어야 한다.

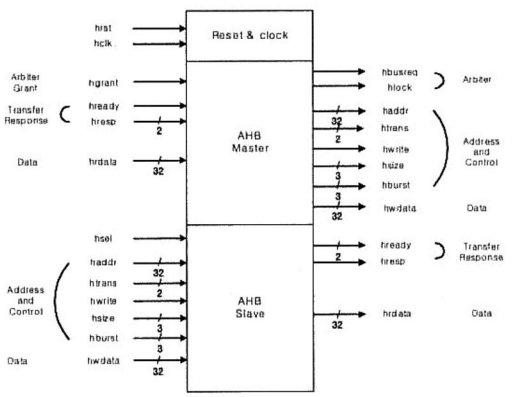

그림 2. AMBA Signals

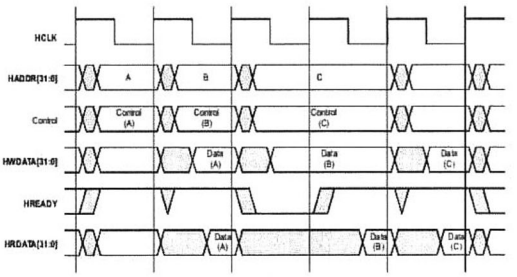

그림 3. AMBA AHB Multiple transfer

Ⅲ. Functions

3.1 Command/Status Resgister

HI는 CPU나 TCP 하드웨어간의 명령, 상태 정보 및 헤더 정보들을 레지스터에 쓰고, 읽어 전달함으로써 control flow를 관리하게 되는데[5], 이를 위하여 내부에 Command/Status Register(CSR)을 가진다. CSR은 동작에 따라 Read Only(RO) 레지스터, Read/Write(RW) 레지스터, Clear on Read(COR) 레지스터의 세 가지 종류의 레지스터로 나누어진다. RO 레지스터는 CPU가 읽기만 가능한 레지스터로, 받은 패킷의 헤더 정보들이 저장되는 레지스터가 이에 속한다. RW 레지스터는 CPU가 읽고 쓰기가 가능한 레지스터로 TCP Connection setup을 위한 Configuration 레지스터들과 보내고자 하는 패킷의 헤더 정보들이 저장된 레지스터이다. 마지막으로 COR 레지스터는 CPU가 읽어간 후에는 Clear되는 레지스터로 Interrupt 레지스터나 Statistics 레지스터들이 이에 해당한다.

CSR을 저장하고 있는 정보의 성격에 따라 나누어 보면, TCP Connection setup을 위한 Configuration 레지스터, Header information 레지스터, Buffered Direct Memory Access(BDMA)에 사용되는 레지스터로 구분된다. Configuration 레지스터에는 자기 자신의 MAC address, IP address, header type과 각종 timer 등이 저장되고, 송수신 패킷의 헤더를 parsing하여 얻은 정보들이 Header Information 레지스터에 저장된다. HI는 데이터의 송수신시, 액세스하고자 하는 외부 RAM의 주소나액세스 가능한 RAM의 크기 등 BDMA를 위한 정보들이 필요한데 이러한 정보들은 BDMA 레지스터에 저장된다. HI는 CPU에 의해 CSR에 쓰여진 이 정보들을 이용하여 외부 TxRAM로부터 데이터를 읽어오거나 외부 RxRAM에 데이터를 쓰게 된다.

TCP Connection이 처음 열리게 되면, HI는 CSR의 Configuration 레지스터의 값들을 TCP에게 valid 신호와 함께 보낸다. 또한, 매 패킷 단위로 헤더 정보들을 TCP로 알려주거나 TCP로부터 받아 CSR에 저장한다.

HI의 Control flow는 CPU의 Slave로서 동작한다. HI의 Slave mode는 hsel와 함께 enable되며, 그림 4의 state machine에 따라 동작하게 된다. HI의 Slave 동작은 CPU가 HI의 CSR에 데이터를 쓰거나 읽기 위해서 액세스하는 경우가 된다. CPU의 control 신호에 따라 state가 변하며, 각 state에서 CPU가 원하는 CSR의 데이터를 읽어 전달하거나, CPU가 쓰고자 하는 데이터를 CSR에 적게 된다. CSRError는 CPU가 쓰기가 허용되지 않는 RO 레지스터나 COR 레지스터에 쓰고자 했을 때 high가 되어 CSR_ERROR state가 되며, 이 state에서는 hready가 low이고 hresp가 ERROR가 되어 Master인 CPU에게 CPU의 요청이 잘못 되었음을 알린다.

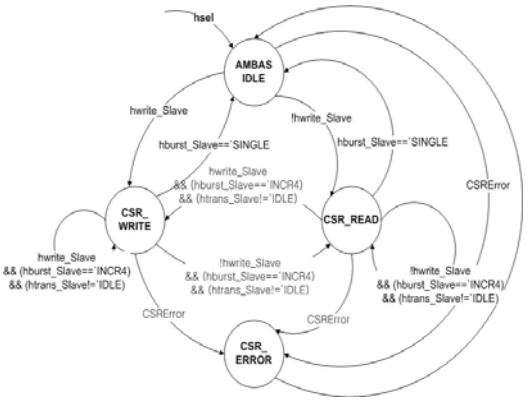

그림 4. CSR Access FSM (AMBA Slave Mode)

• Buffered Direct Memory Access(BDMA)

HI는 데이터의 저장에 있어서 BDMA 방식을 사용한다. 즉 외부 RAM을 HI가 직접 액세스하여 데이터를 읽고 씀으로써 CPU와 HI사이의 데이터 교환이 이루어지는 방식이다. HI는 외부 RAM을 직접 액세스하기 위해서 외부 RAM에 대한 정보를 알아야 하는데 이러한 정보를 가지고 있는 레지스터가 Buffer Descriptor(BD)이다. 아래의 그림 5는 이러한 BD의 data structure를보이고 있다[5]. 본 논문에서 구현한 HI에서는 Rx Flow와 Tx Flow를 위하여 각각 5개의 BD를 가지고 있다. 각 BD는 그림에서 보이는 바와 같이 데이터가 저장된 주소를 알려주는 Buffer Pointer와 그 BD가 가리키는 외부

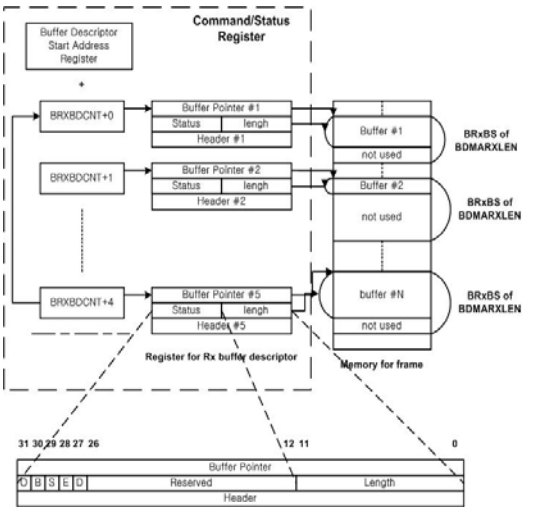

그림 5. Data Structure of Buffer Descriptor

RAM의 주소에 저장된 데이터의 크기를 지칭하는 length, 그리고 상태를 나타내는 status로 이루어져 있다. 또 각 BD는 저장된 데이터와 관련된 헤더 정보들을 가지고 있다. 각 BD는 status field에 ownership bit을 가지고 있는데 이 bit은 BD의 ownership을 결정하게 된다. HI는 이 ownership bit이 1일 때 BD를 사용하여 데이터를 읽어오거나 쓸 수 있게 된다. Rx Flow를 예로 들어보면 CPU는 BD에 HI가 수신한 데이터를 저장할 수 있는 외부 RAM의 주소와 크기 등을 쓰고 이 정보를HI가 사용할 수 있도록 ownership을 1로 한다. HI는 이 BD의 ownership이 1임을 확인하고 이 BD가 가리키는 곳에 수신한 데이터를 저장하고 다시 ownership을 0으로 하여 CPU가 이 정보로 데이터를 외부 RAM에서 읽어갈 수 있도록 한다. 첫번째 BD의 ownership이 CPU로 넘어가면 HI는 두번째 BD의 ownership을 확인하여 데이터를 쓰게 되고, 5개의 BD의 ownership을 모두 잃어버리면 ownership을 가질 때까지 RxTCP Buffer에서 데이터를 읽어오지 않는다. CPU는 BD의 정보를 이용하여 데이터를 읽어 처리한 후에는 ownership을 HI에게 돌려주어 HI가 계속 데이터를 외부 RAM에 저장할 수 있도록 해주어야 한다. 이러한 BD 역시 HI의 CSR에 저장되게 되고, CPU가 이 정보를 업데이트하기 위해서는 AMBA Bus의 Master로서 AMBA Bus의 Slave가 되는 HI를 액세스하여야 한다.

• Reset Generation

HI는 두 개의 리셋에 의해 동작하는데, 하나는 외부에서 들어오는 리셋이고 다른 하나는 TCP/IP

하드웨어를 동작시키기 위해 내부에서 생성하는 리셋이다. HI는 외부에서 들어오는 리셋에 의해 HI 내부의 CSR과 HI가 Slave로 동작하는데 필요한 블록들이 초기화되고, HI의 Master로서 동작하는 블록들과 TCP/IP 하드웨어 블록들은 내부에서 생성된 리셋에 의해 초기화된다. 외부에서 들어오는 리셋이 disable되면 CPU는 HI의 CSR에 Configuration 값들을 쓰게 되고, 충분한 시간 후에 HI의 내부에서 생성되는 리셋을 disable시킴으로써 TCP/IP 하드웨어 블록들이 동작하게 된다.

5.2 Data Flow

Data flow는 패킷의 순수한 데이터 부분에 해당되는데, 수신된 패킷의 데이터를 외부 RxRAM에 쓰는 Rx Flow와 전송하고자 하는 패킷의 데이터를 외부 TxRAM으로부터 읽어오는 Tx Flow가 있다. HI와 TCP 간의 data flow는 Rx Flow와 Tx Flow를 위하여 각각 별도의 FIFO와 bus를 가지지만 AMBA Bus에서의 data flow는 Rx Flow와 Tx Flow가 하나의 버스를 통해 이루어지게 된다. 다음에서는 Data flow를 Rx Flow와 Tx Flow, 그리고 AMBA Bus와의 동작으로 나누어 서술한다.

5.2.1 Rx Flow

Rx Flow는 외부로부터 패킷을 받아 처리하는 flow로, 그림 6는 Rx Flow의 블록다이아그램이다. Rx Flow는 Packet Parser에서 parsing된 데이터가 RxTCP를 거쳐 RxTCP Buffer에 저장되고, RxTCP Buffer에 저장된 데이터를 HI가 읽어 와서 CPU가 읽어갈 수 있도록 외부 RxRAM에 저장하는 일련의 flow를 지칭한다.

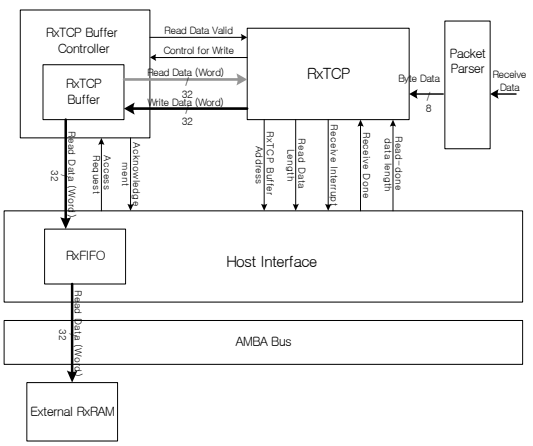

그림 6. Rx Flow의 블록다이아그램

• Data Transfer

RxTCP는 수신한 데이터를 RxTCP Buffer에 쓰고, in-order로 저장된 데이터의 크기가 미리 정의된 threshold 값보다 커지면 HI에게 Receive 신호와 함께 데이터가 저장된 RxTCP Buffer의 주소와 첫 워드의 유효 바이트와 함께 읽어갈 데이터의 크기를 알려준다. 이를 Receive interrupt라고 한다. HI는 RxTCP로부터 Receive interrupt를 받으면 RxTCP Buffer부터 데이터를 읽어 와서 내부의 작은 FIFO를 거쳐 외부 RxRAM에 저장하고 Rcvd_Done과 함께 외부 RxRAM으로 전송한 데이터의 크기를 RxTCP에게 알려주게 된다. 데이터를 저장할 수 있는 외부 RxRAM의 크기가 RxTCP가 읽어가라고 한 데이터 크기보다 작은 경우에는 외부 RxRAM 크기만큼만 저장한다. CSR의 BD 레지스터에는 CPU에 의해 데이터를 저장할 수 있는 외부 RxRAM의 크기와 주소가 저장되어 있는데, HI는 이 BD 레지스터의 정보를 이용하여 지정된 위치의 외부 RxRAM에 데이터를 쓰게 된다. 만약 RxTCP가 전달하는 데이터가 정해진 크기보다 큰 경우에는 정해진 크기만큼만 전달하며 나머지 데이터는 다음번 Receive interrupt가 왔을 때 전달된다. 그림 7은 RxTCP Buffer로부터 데이터를 읽어 와서 RxFIFO에 저장하는 과정을 보이고 있다. HI는 'RXTCPin_IDLE' 상태에서 RxTCP Buffer로부터 acknowledgement를 받으면 'RxTCPin_DATA' 상태가 되어 RxTCP Buffer로부터 받은 데이터를 RxFIFO에 저장하고, 데이터를 모두 다 읽어 왔거나 데이터를 저장할 수 있는 외부 RAM의 공간이 다 차서 더 이상 데이터를 저장할 수 없게 되면 'RXTCPin_IDLE' 상태로 돌아간다. 데이터를 모두 다 읽어오지 못한 상태에서 RxFIFO가 찼거나 RxTCP Buffer로부터 acknowledgement를 받지 못

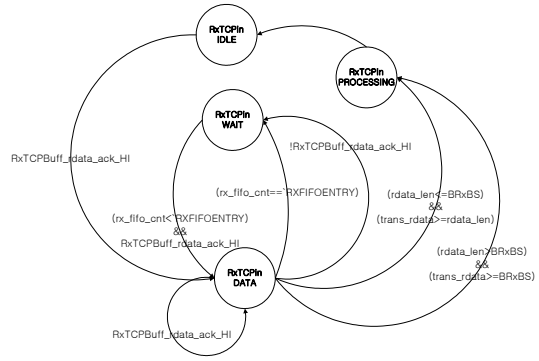

그림 7. Rx Data Flow FSM

한 경우에는 ''RxTCPin_WAIT'상태에서 RxFIFO에 데이터를 저장할 공간이 생기고 RxTCP Buffer로부터 acknowledgement를 받을 때까지 기다렸다가 조건이 충족되면 다시 'RxTCPin_DATA' 상태로 돌아가서 데이터를 읽어오는 일을 계속 수행한다.

• RxTCP Buffer Access Arbitration

HI는 RxTCP로부터 받은 정보를 이용하여 RxTCP Buffer에서 데이터를 읽어 와서 HI 내부의 RxFIFO에 데이터를 저장한다. RxTCP Buffer의 접근은 RxTCP Buffer Controller와의 handshaking을 통해 이루어진다. 그림 8은 HI와 RxTCP Buffer의 동작을 보여주고 있다. HI는 RxTCP로부터 데이터를 읽어가라는 interrupt를 받으면, RxFIFO에 데이터를 저장할 빈 공간이 있는지 확인하고 RxTCP Buffer Controller에게 RxTCP Buffer에 접근하기 위해 request를 보낸다. HI는 RxTCP Buffer Controller에게서 acknowledgment를 받으면, 그 다음 클럭에 유효한 데이터를 받게 된다. RxTCP Buffer의 접근권은 RxTCP에 우선권이 있기 때문에 HI와 RxTCP가 동시에 RxTCP Buffer를 접근하기를 원하면 RxTCP Buffer Controller는 RxTCP 에게 우선권을 주게 되며, HI가 RxTCP Buffer를 사용하고 있는 동안이라도 RxTCP가 RxTCP Buffer를 사용하기를 원하면 HI는 RxTCP Buffer의 접근권을 빼앗기게 된다. HI는 acknowledgment를 받았을 때만 데이터를 읽어올 수 있다.

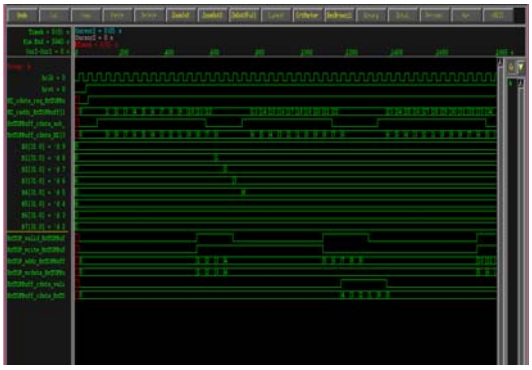

그림 8. Handshaking with RxTCP Buffer Controller

5.2.2 Tx Flow

Tx Flow는 그림 9에서 보이는 바와 같이 데이터를 전송하기 위해 외부 TxRAM에서 데이터를 읽어와 TxTCP Buffer에 저장하고 그 데이터를 Network Interface(NI)로 하여금 전송하게 하는 일련의 과정을 말한다. HI는 TxFIFO에서 전송할 데이터를

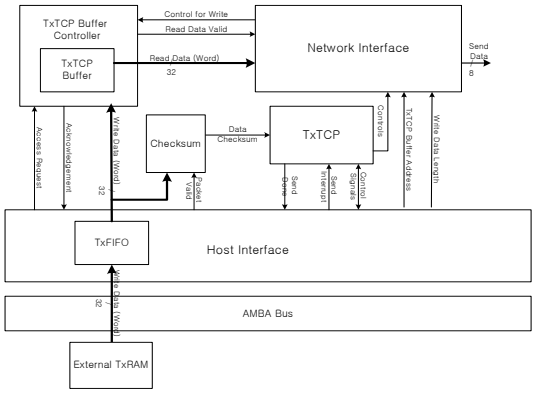

그림 9. Tx Flow의 블록다이아그램

TxTCP Buffer에 저장한 후, 매 패킷 단위로 TxTCP에게 전송할 데이터가 저장된 TxTCP Buffer의 주소와 저장된 데이터의 크기 및 데이터의 첫 워드의 유효바이트를 알려줘야 한다. HI는 Tx Flow를 위해서 세그멘테이션, Invalid Byte에 따른 데이터 전송 및 전송의 중지, TxTCP Buffer의 주소 관리 기능을 수행하여야 한다.

• Data Transfer

HI는 외부 TxRAM에서 워드단위로 데이터를 읽어 와서 TxFIFO를 거쳐 TxTCP Buffer에 저장한다. HI는 BD register에서 데이터 chunk의 첫 워드 중 유효한 바이트에 대한 정보를 읽고, 하나의 패킷에 실려 전송될 데이터를 TxTCP Buffer에 저장한 후 이 정보를 TxTCP에게 알려주어야 한다. HI는 외부 TxRAM에서 TxTCP Buffer로 워드 단위로 데이터를 전송하므로 전송되는 데이터의 첫 워드에는 무효한 데이터가 들어있을 수 있다. HI는 이러한 무효한 바이트 수를 TxTCP에게 알려줌으로써, TxTCP가 TxTCP Buffer에서 읽어온 워드 데이터를 바이트 단위로 NI에게 전송할 때 유효한 바이트부터 전송할 수 있도록 한다. 또한 TxTCP Buffer에 한 패킷에 해당하는 데이터를 저장할 때 무효 바이트 수를 고려함으로써 실제로 HI가 전송하는 데이터 크기는 유효한 데이터 크기보다 커질 수 있다. 그림 10은 무효 바이트의 수에 따라 전달되는 데이터가 달라짐을 그림으로 나타낸 것이다. 그림 10의 예를 보면, 같은 163byte의 데이터를 전달하더라도 전송되는 데이터의 첫 워드의 무효 바이트가 0이거나 1인 경우는 41word의 데이터만 전달되면 되지만, 무효바이트가 2이거나 3인 경우에는 42word의 데이터를 전달해야만 전달하고자 하는 163byte의 데이터를 제대로 전달할 수 있다.

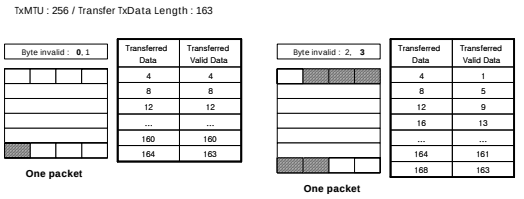

그림 10. Byte Invalid에 따른 Data Transfer

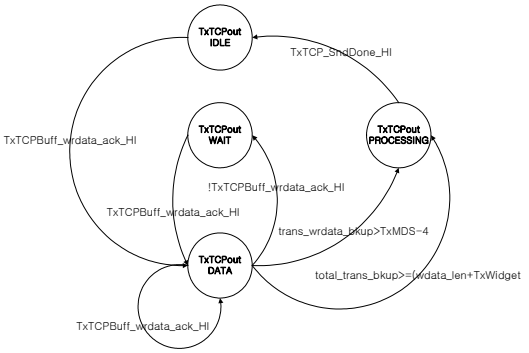

그림 11. TxData Flow FSM

• Segmentation

그림 11은 TxFIFO에 저장된 데이터를 TxTCP Buffer에 저장하는 과정을 나타낸 FSM이다. HI는 TxFIFO의 엔트리가 있음을 확인하고 TxTCP Buffer에 데이터 쓰기를 요청한 후, TxTCP Buffer Controller로부터 acknowledgement를 받으면 TxFIFO의 데이터를 차례로 TxTCP Buffer에 저장하게 된다. 이 과정에서 HI가 전송해야 할 데이터 크기가 Maximum Transfer Unit(MTU)보다 큰 경우, HI에서 데이터를 MTU보다 작은 데이터로 잘라 하나의 패킷에 들어갈 데이터 단위로 처리함으로써 TxTCP가 데이터를 세그멘테이션 해야 하는 부담을 덜도록 하였다. HI가 이러한 기능을 수행하는 또 한가지 이유는 HI가 패킷 단위의 데이터를 TxTCP Buffer에 저장할 때 이 데이터를 Checksum 블록도 받아보도록 하여 패킷에 들어가는 데이터의 checksum을 별도의 메모리 읽기 없이 수행하도록 하기 위해서이다. 전송하는 데이터의 첫 워드의 무효 바이트를 고려하여 HI가 전송하게 되는 총 데이터 크기를 MTU로 자르게 되면, 전송한 데이터 중 유효한 데이터 크기는 MTU와 같거나 MTU보다 작게 된다. 그림 12은 TxMTU가 256byte이고 전달하고자 하는 데이터의 크기가576byte일 때 첫 워드의 무효한 바이트 수에 따라 패킷 당TxTCP로 전달되는 데이터의 크기와 그 데이터 중 유효한 데이터의 크기를 보이고 있다. 무효 바이트가 0인 경우(그림 12의 ⓐ)에는 576byte의 데이터는 TxMTU의 크기(256byte)를 가지는 두 개의 패킷과 나머지 64byte의 크기를 가지는 하나의 패킷으로 세크멘테이션되어 TxTCP Buffer에 저장되며 이 때 저장된 데이터는 모두 유효한 데이터이다. 반면에 무효바이트가 2인 경우(그림 12의 ⓒ)에는 앞의 경우처럼 세 개의 패킷(두개의 MTU 크기의 패킷과 나머지 데이터로 이루어진 하나의 패킷)으로 세그멘테이션되어 TxTCP에게 전달되지만 첫 패킷의 유효 데이터는 첫 워드의 2개의 무효바이트로 인하여 256byte가 아닌 254byte가 된다. 두번째 패킷은 전달되는 256byte의 데이터가 모두 유효하지만 세번째 패킷은 워드 단위로 전달하기 때문에 나머지 66byte의 데이터를 전달하기 위하여 17word (68byte)의 데이터가 전달되고, 전달되는 마지막 워드의 하위 2byte의 데이터도 무효한 데이터가 된다.

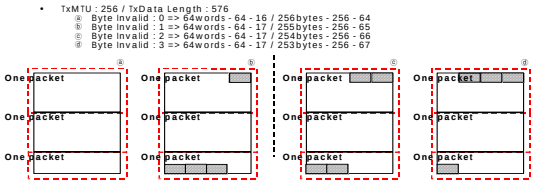

그림 12. Segmentation

• TxTCP Buffer Address Management

HI는TxTCP Buffer의 주소를 관리하는 기능을 수행한다. HI는 TxTCP Buffer에 저장한 뒤 TxTCP에게 데이터가 저장된 주소를 알려주어 TxTCP는 TxTCP Buffer의 주소를 관리할 필요가 없게 한다. TxTCP는 전송한 패킷에 대한 정보를 테이블에 저장하고 있다가 전송한 패킷에 대한 ACK를 받으면, HI에게 비워도 되는 TxTCP Buffer의 끝 주소를 알려준다. HI는 TxTCP로부터 비워도 되는 버퍼의 주소를 받음으로써 쓸 수 있는 버퍼 공간을 확보하게 된다. HI가 버퍼에 데이터를 쓰다가 버퍼가 가득 차면 데이터 전송을 중지하고 버퍼가 빌 때까지 기다려야 한다. TxTCP로부터 비워도 되는 버퍼주소를 받아, 쓰기 가능한 끝 주소를 갱신한 다음에야 데이터 전송이 다시 시작된다. HI는 TxTCP Buffer가 가득 찬 경우 외에도 TxTCP의 Transmit Window가 가득 차서 더 이상 TxTCP가 전송할 패킷에 대한 정보를 처리할 수 없는 경우에도 데이터 전송을 중지하게 되고 TxTCP로부터 데이터 전송을 다시 시작해도 좋다는 신호를 받은 후에야 데이터 전송을 다시 시작하게 된다.

• TxTCP Buffer Access Arbitration

HI는 전송하고자 하는 데이터를TxFIFO에서 읽어 와서 TxTCP Buffer에 저장하고 TxTCP가 읽어가도록 한다. 전송하고자 하는 전체 데이터를MTU를 넘지 않는 데이터 크기로 잘라 패킷 단위로 관리한다. HI가 TxTCP Buffer에 데이터를 쓰는 과정도 RxTCP Buffer를 액세스 하는 경우와 같이 TxTCP Buffer Controller와의 handshaking과정을 통해 이루어진다. 그림 13은 HI와 TxTCP Buffer Controller와의 동작을 보여주고 있다. HI는 TxFIFO에 데이터가 있음을 확인하고 TxTCP Buffer Controller에게 TxTCP Buffer에 접근하기 위해 request를 보낸다. TxTCP Buffer Controller에게서 acknowledgment를 받으면, 그 다음에 쓸 데이터를 데이터 버스 위에 올린다. Controller는 acknowledgment를 보내면서 데이터 버스의 데이터를 샘플링하고 다음 클럭에 데이터가 저장되게 된다. TxTCP Buffer의 접근권 역시 TxTCP에 우선권이 있기 때문에 HI와 TxTCP가 동시에 TxTCP Buffer를 접근하기를 원하면 TxTCP Buffer Controller는 TxTCP 에게 우선권을 주게 되며, HI가 TxTCP Buffer를 사용하고 있는 동안이라도 TxTCP가 TxTCP Buffer를 사용하기를 원하면 TxTCP Buffer의 접근권을 빼앗기게 된다. HI는 acknowledgment를 받았을 때만 데이터를 쓸 수 있다.

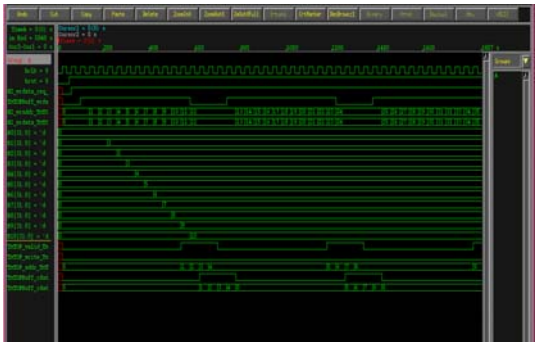

그림 13. Handshaking with TxTCP Buffer Controller

5.2.3 AMBA Operation

HI와 외부 RAM간의 데이터 전송에 있어서HI는 Master로 동작하게 된다. hsel신호가 disable된 상태에서, AMBA Arbiter에게 bus의 접근을 요청하고 hgrant를 얻어 외부 RAM에 데이터를 쓰거나 외부 RAM에서 데이터를 읽어오게 된다. 수신된 데이터를 외부 RxRAM에 저장하는 Rx Flow와 데이터를 전송하기 위해서 외부 TxRAM으로부터 데이터를 읽어오는 Tx Flow가 하나의 AMBA Bus를 통해서 이루어지게 된다. 그림 14의 좌측 state는 Rx Flow에 해당하고 우측 state는 Tx Flow에 해당한다. FSM의 각 state에 따라 데이터 전송을 위한 적절한 control 신호들을 내보내게 된다. 데이터 전송은 워드 단위로 이루어지며 한 번의 hgrant에 대하여 최대 4워드씩 전송하도록 설계되었다.

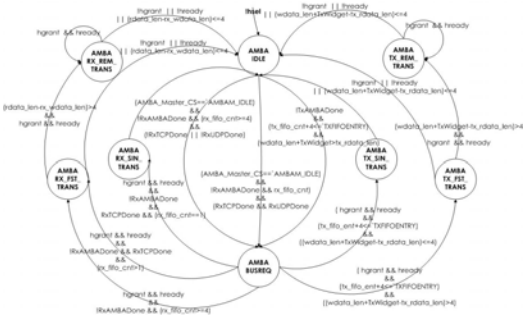

그림 14. AMBA Data Transfer FSM (AMBA Master Mode)

• AMBA Bus Arbitration

HI는 내부에Rx Flow를 위한 RxFIFO와 Tx Flow를 위한 TxFIFO를 가지고 있어 TCP와 HI 간의 데이터 전송은 서로 다른 패스를 통해 이루어진다. 그러나 각 FIFO와 외부 RAM간의 데이터 전송은 하나의 AMBA Bus를 통해 이루어지므로 두 flow 사이에서 arbitration이 이루어져야 한다. 본 논문에서 구현한 HI에서는 Rx Flow에 우선권을 주고 두 flow간의 arbitration을 위해서는 Round Robin 방식을 채택하였다. 그림 15는 AMBA Bus Arbitration을 보이고 있는데, HI 내부에 하나의 flow만이 존재할 때는 그 flow를 위해서 AMBA Bus가 사용되고, IDLE한 상태에서 두 flow가 모두 AMBA Bus를 사용해야 하는 경우에는 Rx Flow를 위한 데

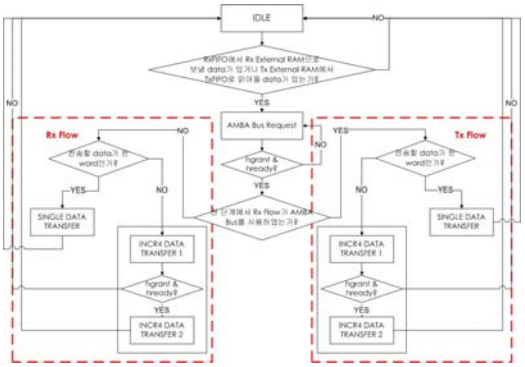

그림 15. AMBA Bus Arbitration

이터 전송을 먼저 하고 그 다음에는 Tx Flow를 위해 AMBA Bus를 사용하는 식으로 번갈아 사용되는 방식이다.

Ⅳ. Interface

• Interface with AMBA

HI는 AMBA Bus를 이용하여 수신한 데이터를 외부 RxRAM에 쓰거나 외부 TxRAM의 데이터를 읽어 와서 TxTCP로 전달하게 된다. 이때 그림 16의 신호들을 사용하게 되며 이 신호들은 AMBA Bus 요청을 위한 신호들과 AMBA의 Master가 Slave에게 데이터를 쓰거나 데이터를 읽어올 때 사용되는 제어 신호들, 데이터 버스, Master에 대한 Slave의 응답 신호들로 구성된다. 이 신호들은 HI가 AMBA의 Master로 동작하는지 Slave로 동작하는지에 따라 입출력이 결정된다.

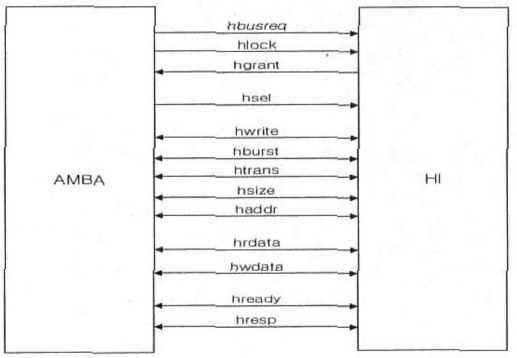

그림 16. Interface with AMBA

• Interface with Hardware Blocks

아래 그림 17는 HI와 TCP 하드웨어 블록 간의 인터페이스를 나타낸 그림이다. HI 와 TCP와의 인터페이스는 크게 TCP Connection 블록과의 인터페이스와 RxTCP, RxTCP Buffer Controller와의 인터페이스, TxTCP, TxTCP Buffer Controller, Checksum 블록과의 인터페이스로 나뉠 수 있다. HI는 TCP Connection 블록에게 CPU의 명령과 configuration 값들을 준다. RxTCP, TxTCP 블록과는 configuration 값들을 전해주는 신호들과 각 패킷의 송수신에 따른 헤더 정보와 buffer 정보, 그에 따른 응답 신호들이 오고 간다. 각 Buffer Controller와는 handshaking 및 데이터 읽기, 쓰기에 관련된 인터페이스를 가지며 Checksum 블록이 매 패킷 단위로 checksum을 계산할 수 있도록Checksum 블록에게 packet valid 신호를 보낸다. Checksum 블록은 packet valid 신호로 각 패킷을 구별하며 HI가 TxTCP Buffer에게 쓰는 데이터를 받아서 checksum을 계산한다.

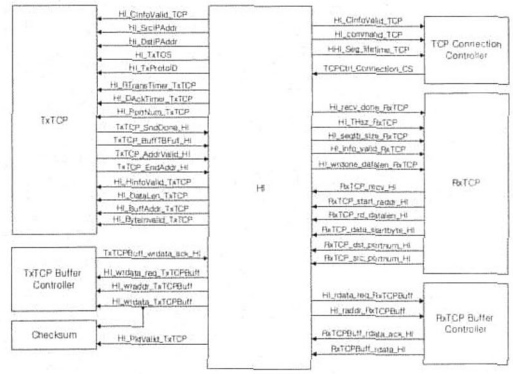

그림 17. Interface with TCP

Ⅴ. Testcases

HI를 위하여 다음의 testcase들을 테스트 해 보았다.

AMBA Bus와의 동작
◆ 다양한 데이터 크기에 따른 AMBA control 신호들의 동작 및 타이밍 확인 ◆ Rx Flow와 Tx Flow사이에서의 AMBA Bus 사용을 위한 arbitration 확인
Data Transfer를 위한 내부 동작
◆ 데이터 송신 및 수신에 따라 BD의 Ownership을 잃어버리는 경우 다음 BD로의 change 확인 ◆ 수신된 데이터가 BD에 지정된 크기보다 큰 경우 BD에 지정된 크기만큼만 외부 RxRAM에 저장하고 RxTCP에 저장한 크기에 대한 응답 확인 ◆ RxFIFO와 TxFIFO의 동작 확인 ◆ 전송하고자 하는 데이터가 MTU보다 큰 경우 MTU로 세그먼테이션 확인 ◆ 전송하고자 하는 데이터의 첫 워드 중 무효한 바이트에 따른 전달 데이터 크기 처리 확인 ◆ TxTCP Buffer의 주소 할당 및 TxTCP에서의 ACK에 따른 valid address 업데이트 확인 ◆ TxTCP Buffer가 full이거나 TxTCP Transmit window가 full인 경우 데이터 전달 중지 및 valid address 업데이트로 인한 재전송 시작 확인
다른 하드웨어 블록과의 동작
◆ CSR로부터 각 정보 및 명령들을 TCP블록에 전달하는지 확인 ◆ RxTCP Buffer Controller, TxTCP Buffer Controller와의 *hand shaking* 동작 확인 ◆ TCP 블록으로부터의 interrupt에 따른 response 확인

Ⅵ. Synthesis Report

Synopsys와 Samsung 0.18 마이크론 기술을 이용하여 Synthesis한 결과, HI의 area는 HI 내부의 CSR과 RxFIFO, TxFIFO를 포함하여 173731 게이트가 소요됨을 보았다.

Ⅶ. Conclusion

인터넷이 보편화되고 멀티미디어 서비스와 같은 고속의 데이터 처리를 요하는 응용 프로그램의 증가 및 링크 속도의 향상으로 엔드 시스템에서의 데이터 처리 속도 또한 고속이 요구되고 있다. 본 연구실에서는 소프트웨어로 구현되어 전체 통신 시스템의 병목점으로 작용하고 있는 TCP/IP 프로토콜을 하드웨어로 구현하는 프로젝트를 수행하였다. 기존의 소프트웨어로 구현되었던 TCP/IP 프로토콜을 하드웨어로 구현함에 있어, CPU나 외부 RAM과 TCP/IP 하드웨어 블록과의 통신을 위해서 Host Interface를 구현하였다. HI는 외부 RAM, CPU와 TCP 하드웨어 사이에서의 통신을 중계하는 역할을 담당한다. HI는 AMBA Specification에 따라 외부 RAM이나 CPU등의 외부 인터페이스와 통신을 하고 TCP 블록과는 직접 연결된 버스를 통하여 통신을 하게 된다. HI는 명령, 상태의 전달 등의 control flow를 위하여 내부에 CSR을 가지며, 이때에는 AMBA Bus의 Slave로 동작한다. 외부 RAM과 TCP 사이에서 전달되는 순수 데이터인data flow의 전달을 위해서는 AMBA Bus의 Master로서 동작한다. HI는 Rx Flow와 Tx Flow의 데이터에 대하여 MTU와 데이터를 저장할 수 있는 RAM 크기에 따라 세그멘테이션을 수행하고, AMBA Bus 이용에 있어서 두 Flow간에 AMBA Bus Arbitration을 수행하는 기능을 담당한다.

참 고 문 헌

[1] James F. Kurose, Keith W. Ross "Computer Networking: A Top-Down Approach Fraturing 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] Ethernet Controller Data Sheet Version 1.0

[5] Wiznet, http://wiznet.co.kr

정 여 진(Yeojin Jung) 준회원

2002년 2월 이화여자대학교 정보통신학과 졸업

2002년 3월~현재 이화여자대학교 정보통신학과 석사과정

<관심분야> 컴퓨터 네트워킹을 위한 스위치/라우터 칩의 설계, TCP/IP 관련 하드웨어 설계 등

임 혜 숙(Hyesook Lim) 정회원

1986년 2월 서울대학교 제어계측학과 졸업

1986년 8월~1989년 2월 삼성 휴렛 팩커드 연구원

1991년 2월 서울대학교 제어계측학과 석사

1996년 12월 The University of Texas at Austin Electrical and Computer Engineering 박사

1996년 11월~2000년 7월 Lucent Technologies Member of Technical Staff

2000년 7월~2002년 2월 Cisco Systems Hardware Engineer

2002년 3월~현재 이화여자대학교 정보통신학과 조교수

<관심분야> 컴퓨터 네트워킹을 위한 스위치/라우터 칩의 설계, TCP/IP 관련 하드웨어 설계, scalable video coding 등