송수신 분리형 TCP/IP Offload Engine을 위한 소프트웨어 및 하드웨어 모듈의 설계

(Design of Software and Hardware Modules for a TCP/IP Offload Engine with Separated Transmission and Reception Paths)

장 한 국 † 정 상 화 †† 최 영 인 †††
(Hankook Jang) (Sang-Hwa Chung) (Young-In Choi)

요 약 TCP/IP Offload Engine(TOE)는 TCP/IP 프로토콜을 네트워크 어댑터 상에서 처리함으로써 호스트 CPU의 프로토콜 처리 부하를 줄이는 기술이다. TOE의 구현 방안으로는 임베디드 프로세서를 사용한 소프트웨어 TOE, ASIC 기반의 하드웨어 TOE, 그리고 하드웨어와 소프트웨어 구현의 장점을 결합한 하이브리드 TOE 등이 제안되어 왔다. 본 논문에서는 하이브리드 방식의 TOE 구현을 위해 두 개의 프로세서 코어를 내장한 FPGA를 기반으로 임베디드 리눅스 기반의 소프트웨어 모듈 및 데이타 송수신에 필요한 하드웨어 모듈들을 설계하였다. 두 개의 프로세서 코어를 사용하여 송신 경로와 수신 경로를 분담하여 관리함으로써 리눅스 프로세스들 사이의 작업 전환 오버헤드를 줄일 수 있고, 송신과 수신 과정의 병렬 처리를 통해 단일 임베디드 프로세서의 성능 한계를 극복할 수 있다. 하드웨어 모듈은 패킷 헤더의 생성 및 처리, DMA를 사용한 데이타 수집 및 저장 등을 담당하여 송수신 성능을 향상시킨다. 본 논문에서는 프로세서 코어 내장형 FPGA가 장착된 TOE 네트워크 어댑터를 사용하여 송수신 분리형 TOE의 성능을 검증하였다.

키워드 : TCP/IP Offload Engine, FPGA, 프로세서 코어, 임베디드 리눅스, 송수신 분리

Abstract TCP/IP Offload Engine (TOE) is a technology that processes TCP/IP on a network adapter instead of a host CPU to reduce protocol processing overhead from the host CPU. There have been some approaches to implementing TOE: software TOE based on an embedded processor; hardware TOE based on ASIC implementation; and hybrid TOE in which software and hardware functions are combined. In this paper, we designed software modules and hardware modules for a hybrid TOE on an FPGA that had two processor cores. Software modules are based on the embedded Linux. Hardware modules are for data transmission (TX) and reception (RX). One core controls the TX path and the other controls the RX path of the Linux. This TX/RX path separation mechanism can reduce task switching overheads between processes and overcome poor performance of single embedded processor. Hardware modules deal with creating headers for outgoing packets, processing headers of incoming packets, and fetching or storing data from or to the host memory by DMA. These can make it possible to improve the performance of data transmission and reception. We proved performance of the TOE with separated transmission and reception paths by performing experiments with a TOE network adapter that was equipped with the FPGA having processor cores.

Key words : TCP/IP Offload Engine, FPGA, Processor Core, Embedded Linux, TX/RX Path Separation

· 본 연구는 산업자원부의 지역혁신 인력양성사업의 연구결과로 수행되었음
† 학생회원 : 부산대학교 컴퓨터공학과
hkjang@pusan.ac.kr
†† 종신회원 : 부산대학교 컴퓨터공학과 교수/컴퓨터및정보통신연구소 연구원
shchung@pusan.ac.kr
(Corresponding author임)
††† 비 회 원 : 부산대학교 컴퓨터공학과
chotbull@pusan.ac.kr
논문접수 : 2006년 5월 24일
심사완료 : 2006년 8월 17일

1. 서 론

인터넷으로 대표되는 컴퓨터 네트워크 환경에서는 TCP/IP 프로토콜이 보편적으로 사용되고 있다. TCP/IP 프로토콜은 오늘날의 운영체제들에는 기본적으로 포함되어 있으며, 호스트 CPU 상에서 처리하는 방식이 일반적으로 사용되고 있다. 이러한 방식은 100 Mbps 대

역폭의 Fast Ethernet 환경까지는 문제가 없었으나 Gigabit Ethernet 이상의 네트워크 환경에서는 호스트 CPU 상에 막대한 부하(load)를 발생시켜 전체 시스템의 성능을 저하시킨다는 문제점을 가진다. 특히, 네트워크의 물리적 대역폭이 Gbps를 넘어서 10 Gbps 이상으로 발전하면 TCP/IP 프로토콜을 처리하는 부하가 급격히 증가하여 단일 호스트 CPU로는 네트워크의 속도를 감당하기 어려운 상황이 발생하게 된다[1-5]. 이러한 문제점을 해결하는 방안으로서 호스트 CPU가 아닌 네트워크 어댑터에서 TCP/IP를 처리하는 TOE(TCP/IP Offload Engine) 기술이 제안되고 있다. TCP/IP를 네트워크 어댑터 상에서 처리하게 되면 호스트 CPU에 가해지는 부하가 줄어들게 되고, 호스트 CPU는 프로토콜 처리 이외의 실질적인 작업에 전념함으로써 전체 시스템의 성능이 향상되는 효과를 얻을 수 있다.

TOE를 구현하는 방식으로는 네트워크 어댑터에 탑재한 범용 임베디드 프로세서 상에서 소프트웨어로 TCP/IP를 처리하는 방식(소프트웨어 TOE)[6], 전용 칩(ASIC)을 개발하여 하드웨어로 TCP/IP를 처리하는 방식(하드웨어 TOE)[7-9], 그리고 TCP/IP 기능 중 일부는 소프트웨어로 처리하고 일부는 하드웨어로 처리하는 방식(하이브리드 TOE)[10,11] 등이 제안되고 있다. 소프트웨어 TOE는 하드웨어 TOE에 비해 구현이 쉽다는 장점을 가지는 반면, 임베디드 프로세서는 호스트 CPU에 비해 성능이 낮으므로 하드웨어 TOE에 비해 프로토콜을 처리하는 성능이 떨어지는 단점이 있다[12]. 하드웨어 TOE는 소프트웨어 TOE에 비해 통신의 성능은 우수하지만[13-15], 구현이 비교적 어렵고 개발에 비용이 많이 든다. 또한 TCP/IP의 상위 프로토콜까지 네트워크 어댑터로 내려서 처리하려고 할 때 상위 프로토콜을 처리하는 하드웨어 모듈을 새로 개발하여 TCP/IP 처리 모듈과 결합해야 하는 어려움이 발생한다. 하드웨어 TOE와 소프트웨어 TOE를 결합한 하이브리드 TOE는 많은 작업 부하로 인하여 임베디드 프로세서 상에서 성능을 확보하기 어려운 기능들은 하드웨어로 구현함으로써 하드웨어 TOE에 근접하는 성능을 제공하고, 연결 설정과 같이 통신의 성능에 영향을 크게 끼치지 않는 기능들은 임베디드 프로세서 상에서 소프트웨어로 처리함으로써 하드웨어 TOE에 비해 비교적 구현이 쉽다는 장점을 가진다. 또한 소프트웨어 구현을 통해 TCP/IP의 상위 프로토콜까지 오프로딩(offloading)하거나 새로운 기능을 추가하기가 용이하다.

본 논문에서는 두 개의 프로세서 코어를 내장한 FPGA 상에서 임베디드 리눅스 기반의 소프트웨어를 바탕으로 데이타 송수신 과정에 필요한 주요 기능들을 하드웨어로 처리하는 하이브리드 방식의 TOE를 설계하였다. 두 개의 프로세서 코어들은 각각 송신 기능과 수신 기능을 전담하여 처리함으로써 리눅스 프로세스들 사이의 작업 전환 오버헤드를 줄이고, 송신과 수신 과정의 병렬 처리를 통해 단일 임베디드 프로세서의 성능 한계를 극복한다. 하드웨어 모듈은 패킷 헤더의 생성 및 처리, DMA를 사용한 데이타 수집 및 저장 등을 담당하여 송수신 성능을 향상시킨다. 또한 본 논문에서는 하드웨어 모듈과 프로세서 코어들의 연동을 위한 메커니즘을 개발하였다. 그리고 본 논문에서는 프로세서 코어 내장형 FPGA가 장착된 TOE 네트워크 어댑터를 개발하고, 이를 사용한 실험을 통해 하이브리드 TOE를 위한 하드웨어 구현의 효과를 입증하였다.

본 논문은 다음과 같이 구성된다. 2장에서는 관련 연구를 소개하고, 3장에서는 하이브리드 TOE의 구조와 소프트웨어 모듈 및 하드웨어 모듈의 구현에 대해 설명한다. 4장에서는 실험 결과를 제시한다. 마지막으로 5장에서는 결론과 향후 연구를 제시한다.

2. 관련 연구

TOE를 개발하는 기존의 방안 중에서 소프트웨어 TOE 구현의 대표적인 사례로는 Intel사의 PRO/1000T IP Storage Adapter [6]가 있다. 이 제품은 기존의 Gigabit Ethernet 어댑터에 Intel 80200 StrongARM 프로세서를 장착하였으며, TCP/IP와 iSCSI (SCSI over IP) 프로토콜을 소프트웨어로 처리한다. 콜로라도 대학에서 이 어댑터를 사용하여 실험한 결과[12]에 따르면 단방향 대역폭이 최대 30 MB/s를 넘지 못하고 있는데, 이는 TCP/IP를 사용하는 일반적인 Gigabit Ethernet 어댑터가 최대 70 MB/s 정도의 단방향 대역폭을 보여주는 것과 비교했을 때 성능이 2배 이상 낮은 것이다.

하드웨어 TOE의 구현 사례로는 Alacritech사의 SLIC [7], Adaptec사의 NAC-7711[8], QLogic사의 ISP4010 [9] 등이 있으며, 이들 모두 Gigabit Ethernet을 지원하고 있다. 제작사들의 발표 자료에 의하면 하드웨어 기반 TOE 제품들은 대체로 100 MB/s 정도의 단방향 대역폭을 보유하고 있으며[13-15], 이는 Gigabit Ethernet의 최대 성능에 근접한 것이다. 그러나, 하드웨어 TOE 구현은 전용 ASIC 구현에 많은 시간과 비용이 소모되고 구현된 하드웨어에서 수정하거나 개선할 사항이 발생할 때마다 새로운 ASIC을 개발해야 한다는 단점을 가진다. 또한 RDMA(Remote Direct Memory Access) 등과 같은 상위 프로토콜까지 네트워크 어댑터에서 오프로딩하여 처리하려는 요구가 발생할 때에도 효과적으로 대응하기 어렵다.

소프트웨어 TOE 구현과 하드웨어 TOE 구현 사례들을 제외하면 본 논문에서 제안하는 바와 같이 하드웨어

모듈과 범용 임베디드 프로세서 상에서 운용되는 소프트웨어 모듈을 결합하여 TCP/IP 프로토콜을 처리하는 방식의 하이브리드 TOE는 아직까지 개발된 사례가 없다.

3. 송수신 분리형 TOE를 위한 소프트웨어 모듈 및 하드웨어 모듈 구현

본 장에서는 두 개의 프로세서 코어를 기반으로 송신 경로와 수신 경로를 분리한 TOE 구조에서 임베디드 리눅스 기반의 소프트웨어 모듈과 송수신 과정의 주요 기능을 담당하는 하드웨어 모듈의 설계 및 구현에 대해 설명한다.

3.1 하이브리드 TOE의 구조

그림 1은 본 논문의 하이브리드 TOE가 구현된 FPGA와 이에 기반한 하이브리드 TOE 네트워크 어댑터의 구조를 나타낸다. 본 논문에서는 두 개의 PowerPC 405 코어를 내장한 Xilinx사의 Virtex-II Pro FPGA를 사용하여 TOE를 구현하였다. 두 개의 프로세서 코어들은 각각 송신 처리를 전담하는 TX 프로세서와 수신 처리를 전담하는 RX 프로세서로 사용되며, 프로세서간 인터페이스를 사용하여 상대방 프로세서에게 작업을 요청할 수 있다. 이렇게 두 프로세서 코어가 송수신 과정을 분담하여 처리하는 메커니즘은 송신 프로세스와 수신 프로세스 사이의 스케줄링에 의한 작업 전환 오버헤드를 제거하여 호스트 CPU에 비해 성능이 떨어지는 임베디드 프로세서의 단점을 극복할 수 있게 한다. 각 코어들은 FPGA 외부에 PLB(Processor Local Bus) 버스를 통해 연결된 64MB 용량의 SDRAM과 OPB(On-chip Peripheral Bus) 버스를 통해 연결된 32MB 용량의 플래시 메모리를 독자적으로 가진다. 그리고 각 코어들은 300 MHz의 코어 클럭, 100 MHz의 PLB 클럭, 50 MHz의 OPB 클럭으로 동작한다.

FPGA 내부에서 TOE의 하드웨어는 TOE 모듈, 호스트 인터페이스, GbE(Gigabit Ethernet) 컨트롤러 등으로 구성된다. TOE 모듈은 하이브리드 TOE에서 하드웨어 구현의 핵심으로 송수신 기능 등을 처리하는 하드웨어 엔진과 프로세서 코어들과의 인터페이스를 담당하는 HW/SW 인터페이스로 구성된다. TOE 모듈은 TX 프로세서와 RX 프로세서의 OPB 버스에 양쪽으로 연결되며, 따라서 TX 프로세서와 RX 프로세서는 메모리에 접근하는 방식으로 TOE 모듈에 접근할 수 있다. 송신 파트(TX 프로세서 및 송신 하드웨어 모듈)과 수신 파트(RX 프로세서 및 수신 하드웨어 모듈)는 TOE 모듈 내의 연결정보 메모리를 통해 TCP 연결(connection)에 대한 정보를 공유한다. 호스트 인터페이스는 64bit/66MHz PCI 컨트롤러를 내장하여 호스트 CPU와 TOE 모듈 사이에서 인터페이스를 담당한다. GbE 컨트롤러는 Gigabit Ethernet MAC/PHY 칩과의 인터페이스를 담당하여 Ethernet 패킷의 송수신을 처리한다.

TOE 네트워크 어댑터를 사용한 통신의 진행 과정은 다음과 같다. 호스트 CPU의 사용자 프로그램이 TOE를 사용한 통신을 요청하면 이 요청은 호스트 운영체제의 TCP/IP 프로토콜 스택을 거치지 않고 TOE 어댑터의 호스트인터페이스로 직접 전달된다. 호스트 인터페이스는 이 요청을 TOE 모듈로 전달하고, TOE 모듈에서

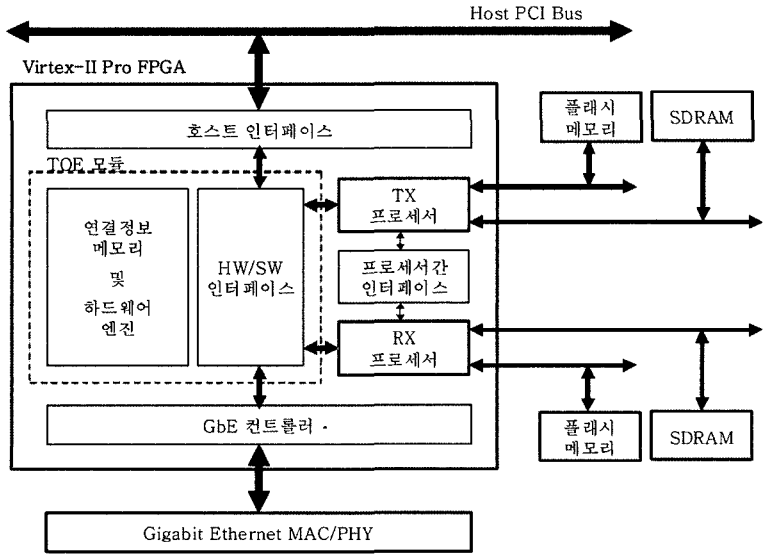

그림 1 하이브리드 TOE 네트워크 어댑터의 구조

하드웨어와 소프트웨어를 연동하여 요청된 작업을 처리한다. 요청된 작업이 데이타의 송신인 경우 TOE 모듈은 원격 노드로 전송할 데이타를 호스트 CPU의 메인 메모리에서 DMA 읽기를 사용하여 가져온다. TOE 모듈은 TCP/IP 헤더와 MAC 헤더를 생성하여 패킷을 완성하고, 패킷 생성이 끝나면 GbE 컨트롤러에 패킷의 전송을 요청한다. GbE 컨트롤러의 요청을 받은 Gigabit Ethernet MAC은 완성된 패킷을 DMA 읽기를 통해 가져가서 수신 노드로 전송한다. 수신 과정에서는 송신 과정의 역순으로 수신 패킷이 처리된다. 요청된 작업의 처리가 끝나면 호스트 인터페이스가 호스트 CPU에 인터럽트를 발생시켜 처리가 완료되었음을 알리고, 호스트 CPU는 호스트 인터페이스에 저장된 처리 결과를 읽어가서 사용자 프로그램에 전달한다.

3.2 듀얼 프로세서 코어에 기반한 송수신 분리형 TOE 소프트웨어 모듈

본 구현에 사용되는 소프트웨어 모듈은 TCP/IP 프로토콜 스택을 내장한 임베디드 리눅스(커널 버전 2.4.18), 디바이스 드라이버, 응용 프로그램의 세 가지로 범주로 구성된다. TX 프로세서와 RX 프로세서는 각자 독자적인 메모리 상에서 임베디드 리눅스 커널과 디바이스 드라이버, 응용 프로그램을 운용한다. 송수신 경로의 분리를 위한 커널은 동일한 커널 이미지를 기본으로 하고, 각 프로세서의 역할에 따라 일정한 함수들만을 수행하도록 구현한다. 디바이스 드라이버와 응용 프로그램은 커널 이미지와 분리된 램 디스크 형태로 탑재되어 각 프로세서에 의해 수행된다.

본 논문에서 제안하는 송수신 분리 구조에서는 하나의 TCP 연결에 대해 송신과 수신이 각각 다른 프로세서에 의해 관리되어야 하며, 두 프로세서들은 흐름 제어 등의 TCP 연결 관리를 위해 필요한 정보들을 공유해야 한다. 본 논문에서는 TX 프로세서와 RX 프로세서 사이에서 공유되는 연결 정보로서 임베디드 리눅스의 INET 소켓 구조체를 사용하며, FPGA 내부의 듀얼 포트 메모리를 사용하여 각 TCP 연결에 해당하는 INET 소켓 구조체를 유지한다.

그림 2는 일반적인 리눅스 커널의 프로토콜 스택 구조와 송수신 경로 분리를 위해 수정된 프로토콜 스택의 구조를 비교하여 보여준다. 일반 커널의 프로토콜 스택은 TCP 연결에 대해 SOCK_STREAM 타입을 사용한다. 반면 본 논문의 송수신 분리 구조에서는 송신을 처리하는 TX 커널과 수신을 처리하는 RX 커널이 INET 소켓 계층을 공유하는 구조를 가지며, BSD 소켓 계층에서 송신 경로와 수신 경로를 구분하여 처리하기 위해 SOCK_TOE_TX와 SOCK_TOE_RX 소켓 타입을 별도로 정의하였다. TX 프로세서와 RX 프로세서는 프로토콜 스택에 전달된 BSD 소켓 타입에 따라 리눅스 커널의 특정 오퍼레이션들을 수행한다.

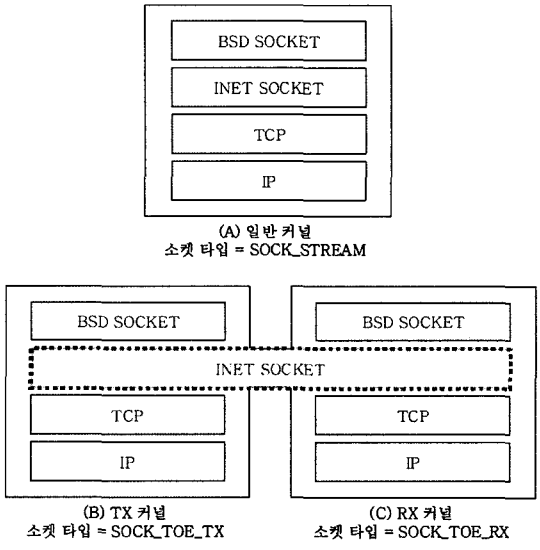

그림 2 송수신 경로 분리를 위한 TCP/IP 프로토콜 스택의 구조

3.3 연결 정보 메모리와 프로세서간 인터페이스

송수신 분리 구조에서는 두 프로세서들 사이에서 흐름 제어 등의 TCP 연결 관리에 필요한 정보를 공유하거나 상대 프로세서에 연동 작업을 요청하는 데 사용할 인터페이스가 필요하다. 본 연구에서는 프로세서들 사이의 연결 정보 공유 및 명령 전달 인터페이스로 FPGA 내부의 듀얼 포트 메모리를 사용한다. TCP 연결이 생성될 때 TX 프로세서와 RX 프로세서 사이의 듀얼 포트 메모리상에 연결 정보(INET 소켓 구조체)가 생성되고, 각 프로세서는 동일한 하드웨어 주소를 사용하여 연결 정보에 접근한다. 그림 3은 프로세서들 사이의 명령 전달 인터페이스와 연결 정보 공유 메모리를 포함한 전체 듀얼 포트 메모리의 맵 구성을 보여준다. 본 논문에서는 16 KB의 듀얼 포트 메모리 공간을 사용하고 있으며, 이 중에서 하위 2 KB의 영역을 COMMAND 영역으로 지정하여 프로세서들 사이에 명령을 전달하는 인터페이스로 사용한다. 나머지 14 KB의 영역은 2 KB 크기의 연결 정보를 저장하는 데 사용되며, 따라서 현재의 구현에서는 최대 7개의 연결을 관리할 수 있다.

COMMAND 영역 중에서 하위 1 KB의 영역은 RX 프로세서에서 TX 프로세서로의 명령 전달에 사용되고 TX 프로세서에서 RX 프로세서로의 명령 전달에는 512 byte가 사용된다. 나머지 512 byte는 ARP 패킷 처리와 RX 프로세서에서 생성된 SYN/ACK 패킷 처리 등의

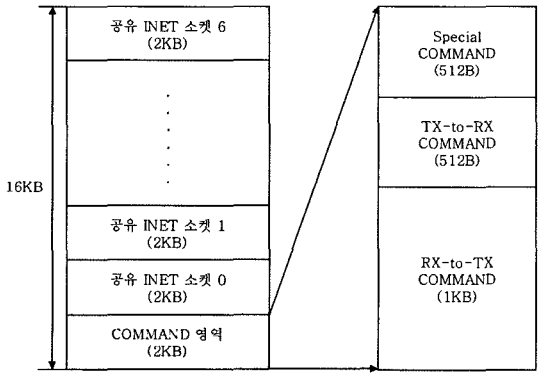

그림 3 듀얼 포트 메모리의 맵 구성

특수 명령을 위한 영역으로 사용된다. 두 프로세서 사이의 명령 전달에 사용되는 디스크립터(descriptor)는 4 byte의 COMMAND 필드와 4 byte의 INFO 필드가 결합된 구조체이다. 따라서 RX to TX 명령 영역은 128개의 명령 디스크립터 배열이 만들어지고, TX to RX 명령 영역은 64개의 명령 디스크립터 배열이 만들어진다. 디스크립터를 만들어 저장한 후 HW/SW 인터페이스 내에 예약된 메모리 영역에 전달을 요청하는 명령을 저장하면 HW/SW 인터페이스가 상대방 프로세서에 인터럽트를 걸어서 디스크립터의 도착을 알린다.

3.4 송수신 분리형 TOE를 위한 하드웨어 모듈의 설계 및 구현

그림 4는 송신을 처리하는 하드웨어 모듈의 내부 구조를 보여준다. 송신 하드웨어 모듈은 크게 호스트 인터페이스와의 연결 모듈, TX 프로세서와의 인터페이스, 수신 모듈과의 인터페이스, 패킷 생성 파트로 구성된다. TX 프로세서와의 인터페이스는 OPB 버스를 통한 접근을 지원하고 필요에 따라 TX 프로세서에 인터럽트를 거는 역할을 맡는다. 수신 모듈과의 인터페이스는 수신 모듈의 연동(coprocessing) 작업 요청을 TX 프로세서 및 송신 하드웨어 모듈에 전달하는 등의 역할을 맡는다. 패킷 생성 파트는 작업할 패킷의 헤더와 데이타 페이로드를 저장하는 패킷 버퍼, 호스트 메모리에서 데이타를 DMA 읽기로 가져오는 페이로드 생성모듈, TCP/IP/MAC 헤더 생성 모듈, 연결 정보 메모리와의 인터페이스, 연결 정보를 해석하고 헤더 생성 모듈에 정보를 제공하는 연결정보 디코더 등으로 구성된다. 헤더 생성 및 흐름 제어 등에 필요한 각종 정보는 송신 모듈과 수신 모듈이 공유하는 연결 정보 메모리에서 유지된다.

송신 하드웨어 모듈의 동작 과정은 다음과 같다. 사용자 프로그램이 send() 함수를 통해 데이타 송신을 요청하면 DMA 읽기를 통해 호스트 메모리에서 패킷 버퍼로 데이타를 가져오고, 데이타를 패킷 버퍼에 저장하는 동안 TCP 체크섬 생성을 위해 부분 체크섬을 계산한다. 데이타 복사가 끝나면 TCP 헤더, IP 헤더, MAC 헤더를 차례대로 생성하고 각 헤더를 패킷 버퍼에 저장한다. 패킷의 생성이 완료되면 GbE 컨트롤러에 패킷의 전송을 요청한다.

그림 5는 수신을 처리하는 하드웨어 모듈의 내부 구조를 보여준다. 수신 하드웨어 모듈은 전체적으로 송신 하드웨어 모듈과 유사한 구조를 가지며, 호스트 인터페이스와의 연결 모듈, RX 프로세서와의 인터페이스 파트, 송신 모듈과의 인터페이스 파트, 패킷 처리 파트로 구성된다. RX 프로세서와의 인터페이스는 OPB 버스를 통한 접근 지원과 인터럽트를 생성하는 역할을 맡는다. 송신 모듈과의 인터페이스 파트는 연결 설정 등과 같이 송신 모듈과의 연동 작업이 요구될 경우 송신 모듈에게 필요한 작업을 요청하는 등의 역할을 수행한다. 패킷 처리 파트는 MAC 헤더에서 패킷의 종류 판별, IP 주소

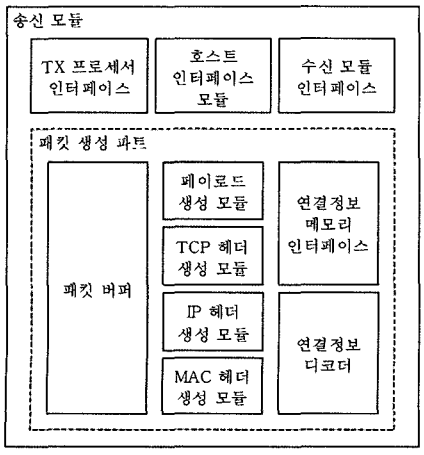

그림 4 송신 하드웨어 모듈의 내부 구조

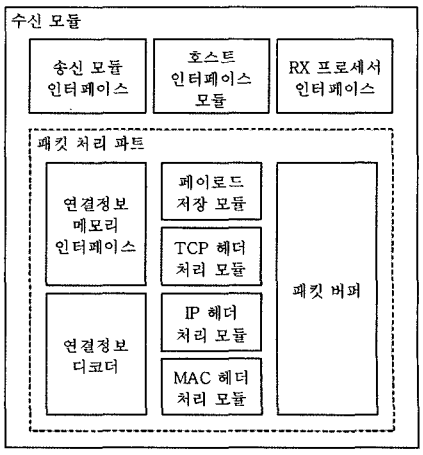

그림 5 수신 하드웨어 모듈의 내부 구조

검사 및 IP 헤더 체크섬 검사, TCP 헤더에서 순서 번호와 ACK 번호 점검, TCP 체크섬 검사 등의 작업을 수행한다. 그리고 수신 패킷이 데이타 패킷인 경우에는 헤더 처리가 끝나면 페이로드 저장 모듈이 DMA 쓰기를 통해 수신 데이타를 호스트 메모리의 사용자 버퍼에 저장한다. 헤더 처리를 위해 필요한 TCP 연결 정보는 송신 하드웨어 모듈과 마찬가지로 연결 정보 메모리에서 가져온다.

수신 하드웨어 모듈의 동작 과정은 다음과 같다. 사용자 프로그램이 recv() 함수를 통해 데이타 수신을 요청하면 해당TCP 연결의 상태를 수신 대기 상태에서 설정하여 데이타 패킷이 도착하기를 기다리게 한다. GbE 컨트롤러에서 패킷 수신이 발생하면 수신패킷이 패킷 버퍼로 복사되고, 이어서 MAC/IP/TCP 헤더에 대한 검사와 처리가 수행된다. 헤더 처리가 끝난 후 수신 패킷이 데이타를 포함하고 있으면 연결정보 메모리에서 대상 TCP 연결을 검색하여 사용자 버퍼의 정보를 획득하여 DMA 쓰기를 통해 사용자 버퍼로 데이타를 복사한다. 데이타 복사가 끝나면 호스트 CPU에 인터럽트를 걸어서 데이타의 수신을 알린다.

4. 실험 및 분석

본 논문의 실험에서는 TOE 구현에 있어서 임베디드 리눅스를 기반으로 하여 소프트웨어만으로 TCP/IP를 처리하는 경우와 송수신 기능의 일부를 하드웨어로 구현한 경우의 성능을 비교하였다. 실험에 사용된 두 노드는 TOE 네트워크 어댑터를 탑재하고, 스위치를 거치지 않고 직접 연결하였다. 각 노드는 1.8 GHz로 동작하는 Intel Xeon CPU와 512 MB의 메인 메모리 및 64bit/66MHz PCI 버스를 탑재하고 있다. 호스트 CPU의 운영체제로는 리눅스 커널2.4.7-10을 사용하였다.

그림 6은 소프트웨어만으로 구현된 TOE와 하드웨어 구현이 포함된 TOE의 최소 지연시간을 비교하고 있다. 소프트웨어만으로 구현된 TOE는 최소 지연시간이 약 680μs로 측정되었으며, 하드웨어 구현이 포함된 TOE는 최소 지연시간이 약 181μs를 보였다. 그리고 데이타의 크기가 증가할 때 하드웨어 구현이 포함된 경우에는 지연시간이 늘어나는 폭이 소프트웨어만으로 처리하는 경우에 비해 크지 않았다. 소프트웨어만으로 처리하는 경우의 지연시간 증가가 큰 이유는 호스트 메모리에서 FPGA 내부의 버퍼로 가져온 데이타를 리눅스가 관리하는 SDRAM으로 복사해서 패킷을 생성하고, 생성된 패킷을 다시 GbE 컨트롤러의 버퍼로 복사하는 과정을 거치기 때문이다. 그러나 하드웨어 구현이 포함된 경우에는 패킷 버퍼로 데이타를 가져온 후에는 더 이상 불필요한 복사가 발생하지 않으므로 지연시간의 증가가 크지 않았다. 이러한 결과는 데이타 송수신시 하드웨어 구현이 TOE의 성능을 크게 향상시킬 수 있다는 사실과 범용 임베디드 프로세서를 사용할 경우 소프트웨어만으로는 통신의 성능을 완전히 이끌어낼 수 없다는 사실을 입증하고 있다.

본 논문에서 하드웨어로 구현한 기능들은 통신의 성능을 더욱 향상시킬 수 있도록 최적화할 수 있는 여지가 남아있다. 먼저 송신 과정에서 TCP, IP, MAC 헤더들을 생성하는 과정을 살펴보면 현재 구현에서는 각 계층의 헤더를 순차적으로 생성하고 있으나 이를 병렬화함으로써 헤더 생성 시간을 줄일 수 있다. 그리고 현재의 구현에서는 DMA 읽기로 데이타를 완전히 가져온 이후에 헤더의 생성을 시작하고 있으나 차후에는 데이타를 가져오는 동안 헤더 생성에 필요한 기본 정보를 연결 정보 메모리에서 미리 읽어와 준비해 둠으로써 패킷을 생성하는 데 걸리는 시간을 전체적으로 줄일 수 있다. 또한 작은 크기의 데이타는 TOE 네트워크 어댑터가 DMA로 읽어오는 대신 호스트 CPU가 PIO(Processor I/O)로 직접 TOE 네트워크 어댑터로 전달하게 되면 DMA 초기화 시간을 절약할 수 있다. 수신 과정

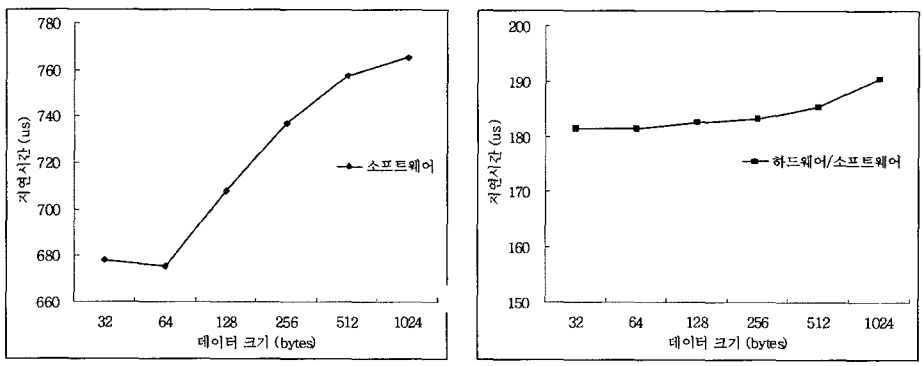

그림 6 소프트웨어 구현과 하드웨어 구현의 최소 지연시간 비교

에서는 수신 패킷을 패킷 버퍼에 저장하는 동안 헤더 처리를 완료함으로써 전체적인 수신 처리 시간을 단축할 수 있다.

5. 결론 및 향후 과제

본 논문에서는 두 개의 프로세서 코어를 내장한 FPGA 상에서 임베디드 리눅스에 내장된 TCP/IP 프로토콜 스택의 송수신 경로를 분리하고 송수신 기능의 일부를 하드웨어로 처리하는 TOE를 구현하였다. 송수신 분리 구조에서 두 프로세서의 연동을 위해 필요한 하드웨어 및 소프트웨어의 구조와 동작 메커니즘을 제시하고, 이를 위해 설계된 하드웨어 구조 및 소프트웨어 구조에 대해 설명하였다. 본 연구는 두 개의 임베디드 프로세서 코어를 사용해 송신과 수신을 분리하여 처리함으로써 TOE의 통신 부하를 덜고, 송수신 기능을 하드웨어로 처리함으로써 성능의 향상을 도모하는 연구의 기초를 제시한다. 그리고 소프트웨어만으로 TCP/IP를 처리하는 TOE 구현과 하드웨어 구현을 포함한 TOE의 성능을 비교한 실험을 통해 하드웨어 구현의 성능을 확인하였다. 실험의 결과 소프트웨어만으로 구현된 TOE는 최소 지연시간이 약 680μs이었으며, 하드웨어 구현이 포함된 TOE는 최소 지연시간이 약 181μs이었다.

본 연구의 향후 과제로는 하드웨어로 구현된 모듈의 성능을 개선하고 송수신 과정의 주요 기능에 대해 하드웨어 구현의 폭을 확대하고자 한다. 이미 하드웨어로 구현된 기능들 중에서 송신 과정에서는 TCP, IP, MAC 헤더들을 생성하는 과정을 최대한 병렬화하고, DMA 읽기로 데이타를 가져오는 동안 헤더 생성에 필요한 기본 정보를 미리 준비하며, 작은 크기의 데이타는 호스트 CPU가 PIO로 직접 TOE 네트워크 어댑터로 전달하는 등의 최적화를 통해 성능을 향상시킬 계획이다. 수신 과정에서는 수신 패킷을 패킷 버퍼에 저장하는 동안 헤더 처리를 완료하여 수신 처리 시간을 단축할 계획이다. 그리고 ACK 패킷의 생성과 처리, 흐름 제어 및 혼잡 제어에 필요한 기능들에 대한 하드웨어구현을 추가로 진행하여 TOE의 성능을 최대화하고자 한다.

참 고 문 헌

[1] N. Bierbaum, "MPI and Embedded TCP/IP Gigabit Ethernet Cluster Computing," Proceedings of 27th Annual IEEE Conference on Local Computer Networks 2002, pp. 733-734, Nov. 2002.

[2] J. Kay and J. Pasquale, "Profiling and Reducing Processing Overheads in TCP/IP," IEEE/ACM Transactions on Networking, Vol. 4, No. 6, pp. 817-828, Dec. 1996.

[3] R. A. F. Bhoedjang, T. Ruhl, and H. E. Bal, "User-Level Network Interface Protocols," IEEE Computer, Vol. 31, No. 11, pp. 53-60, Nov. 1998.

[4] B. S. Ang, "An evaluation of an attempt at offloading TCP/IP protocol processing onto an i960RN-based iNIC," Tech. Rep, HPL-2001-8, HP Labs, Jan. 2001.

[5] G. Banga and J. C. Mogul, "Scalable kernel performance for Internet servers under realistic loads," USENIX Annual Technical Conference, pp. 12, June 1998.

[6] Intel Corporation, "Intel PRO/1000T IP Storage Adapter," Data Sheet, 2003, available at http://www.intel.com

[7] Alacritech, Inc., "SLIC Technology Overview," Technical Review, 2002, available at http://www.alacritech.com.

[8] Adaptec, Inc., "Adaptec TOE NAC 7711," Data Sheet, 2003, available at http://graphics.adaptec.com.

[9] QLogic Corporation, "iSCSI Controller," Data Sheet, 2003, available at http://download.qlogic.com

[10] S.-C. Oh, H. Jang, and S.-H. Chung, "Analysis of TCP/IP protocol stack for a Hybrid TCP/IP Offload Engine," Proceedings of the 5th International Conference on Parallel and Distributed Computing, Applications and Technologies, pp. 406-409, 2004.

[11] H. Jang, S.-H. Chung, S.-C. Oh, "Implementation of a Hybrid TCP/IP Offload Engine Prototype," Proceedings of the 10th Asia-Pacific Computer Systems Architecture Conference, pp. 464-477, 2005.

[12] S. Aiken, D. Grunwald, A. R. Pleszkun, and J. Willeke, "A Performance Analysis of the iSCSI Protocol," Proceedings of the 20th IEEE/11th NASA Goddard Conference on Mass Storage Systems and Technologies, pp. 123-134, 2003.

[13] Lionbridge Technologies, Inc., "Alacritech SES-1001T: iSCSI HBA Competitive Analysis," Veri-Test Benchmark Report, 2004, available at http://www.veritest.com

[14] Adaptec, Inc., "Unleashing File Server Potential with Adaptec GigE NAC 7711," Benchmark Report, 2003, available at http://graphics.adaptec.com

[15] H. Ghadia, "Benefitsof full TCP/IP offload (TOE) for NFS Services," Proceedings of 2003 NFS Industry Conference, 2003, available at http://nfsconf.com

장 한 국

정보과학회논문지 : 시스템 및 이론
제 33 권 제 5 호 참조

정 상 화
정보과학회논문지 : 시스템 및 이론
제 33 권 제 5 호 참조

최 영 인
2005년 부산대학교 컴퓨터공학과 학사
2005년~현재 부산대학교 컴퓨터공학과 석사과정. 관심분야는 클러스터 시스템, TOE