# IXP1200 네트웍프로세서에서의 스케쥴러의 설계

신상호[0] 임경수 안순신
고려대학교 전자공학과
{feagle, angus, sunshin}@dsys.korea.ac.kr

# Design of Scheduler in IXP1200 Network Processor

Shang-Ho Shin[0] Kyung-Soo Lim Sun-shin Ahn
Dept. of Electronics, Korea University

요 약

인터넷에서 저성능의 소프트웨어적인 처리 또는 고성능의 하드웨어 처리를 하는 장비의 단점을 보완하기 위해서 네트웍프로세서를 사용하는 방법이 등장하였다. 네트웍 프로세서를 이용해서 네트웍기능을 지원하기 위해서는 한정된 자원을 효율적으로 활용하기 위해서 스케쥴러가 요구된다. 네트웍프로세서에서 스케쥴러를 설계하기 위해서 필요한 사항과 구조등에 관해서 알아본다.

## 1. 서론

인터넷 트래픽을 처리하기 위해서 사용되는 장비는 초기에는 소프트웨어적인 방법을 이용해서 처리를 하였지만 트래픽의 증가로 인하여 고성능의 처리를 위해서 ASIC을 이용한 하드웨어적인 처리를 하였다. 하드웨어적인 방법은 성능은 뛰어나지만 변경이 어렵고 제작기간이 길어서 소프트웨어적인 방법이 가지는 유연성이 요구되었다. 이러한 사항을 만족시키기 위해서 나타난 것이 네트웍프로세서이다. 네트웍프로세서는 네트웍처리에 필요한 기능들을 포함하고 있는 프로세서로서 소프트웨어적인 유연성과 함께 하드웨어적인 고성능을 제공한다.

## 2. IXP1200 Networkprocessor의 구조

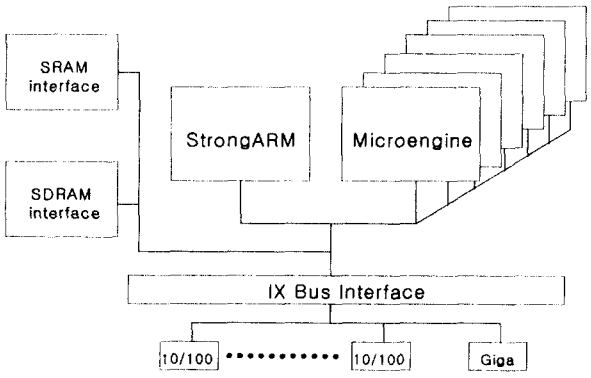


그림 1. IXP1200의 구조

IXP1200은 Intel사의 네트웍프로세서로서 그림 1과 같은 구조를 가진다. Forwarding 기능을 수행하는 6개의 Microengine과 라우팅프로토콜을 처리하는 한 개의 StrongARM프로세서로 이루어진다. 많은 프로세서를 병렬로 사용함으로써 고성능의 처리를 하며, 이들 각각을 프로그램에 의해서 기능을 제공함으로써 유연성을 제공한다. 또한 IX라는 버스구조를 통해서 입출력 포트와의 패킷전송을 수행한다. SRAM과 SDRAM에 대한 interface를 따로 가짐으로써 메모리 접근에 대한 효율성을 높이고 있다. Microengine은 RISC에 기초한 프로세서로서 각각이 최대 4개까지의 thread를 하드웨어적으로 지원한다. Thread를 사용하여 메모리 acces같은 딜레이가 긴 작업의 딜레이를 숨김으로써 실행효율을 높인다.

## 3. Receive Scheduler

Receive 기능을 수행할 때의 IXP1200의 전체적인 동작은 그림 2와 같다. 입출력 포트에서 IXP1200으로 패킷을 읽어 들이기 위해서 사용되는 버퍼인 RFIFO와 RFIFO에 저장된 패킷을 처리할 receive thred와 이들을 스케쥴링할 receive scheduler가 주요대상이 된다. Receive scheduler는 packet을 처리할 receive thread가 존재하는지를 receive thread의 종료상태를 나타내는 thread done register를 통해서 확인하고 패킷이 도착해 있는 포트를 입력포트의 패킷수신 상태를 나타내는 receive ready레지스터를 읽음으로써 확인한다. 이 두 가지가 모두 존재할

경우 패킷을 저장할 RFIFO 의 element를 할당하고 이것들을 정리해서 receive request를 보낸다. Receive state machine이 receive request에 기록된 내용을 통해서 입력 포트에서 지정된 RFIFO element로 패킷을 전송하고 전송이 완료되면 receive thread에게 signal을 보내서 패킷을 처리할 수 있도록 한다. 패킷처리를 끝낸 receive thread는 thread done register에 종료상태를 기록한다.

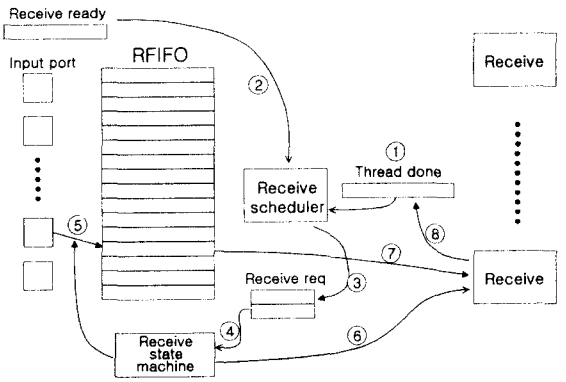


그림 2. Receive scheduler의 동작

Receive scheduler를 설계할 때 있어서 가장 중요한 사항은 receive scheduler의 동작속도가 입력 포트에 패킷이 도착하는 속도보다 빨라야 한다는 것이다. 스케줄러의 동작속도를 제한하는 요소는 우선 스케줄러 자체의 실행주기이다. Microengine은 RISC프로세서이므로 하나의 명령이 한 Cycle에 실행된다고 볼 수 있다. 딜레이가 긴 명령이라도 스케줄러에서 사용하는 경우에는 실행시간을 예측할 수 있으므로 스케줄러가 실행되는 시간을 결정할 수가 있다. 반면에 스케줄러가 스케줄링을 하기 위해서 필요한 자원인 receive thread는 contention이 증가할수록 딜레이의 증가로 receive thread의 실행주기가 늘어난다. 따라서receive scheduler는 사용할 수 있는 receive thread가 회수될 때까지 기다려야 하므로 스케줄러의 실행시간이 receive thread의 실행주기에 의해서 결정되게 된다. 부하가 적을 때는 상관없겠지만 부하가 커질 경우에는 receive thread의 동작속도에 의해서 스케줄러의 동작속도가 제한되고 그에 따라서 각각의 입력포트에 대한 처리가 지연될 수 있다는 것을 고려해서 스케줄러를 설계해야 한다.

스케줄러는 입력 포트의 패킷수신여부를 결정하기 위해서는 각각의 입력포트의 상태를 읽어 와야 한다. 이 정보를 읽어 들이는 방법으로는 스케줄러에서 주기적으로 읽어 들이는 polling에 의한 방법과 일정한 주기에 의해서 하드웨어에서 자동으로 읽어서 저장시키는 Autopush 두 가지의 방법이 있다. Autopush는 context swap없이 사용할 수 있지만 하드웨어에서 기록하고 있을 때 읽어 들이지 않도록 주의를 해야 한다. 주기적으로 읽어 들이는 방법은 필요할 때 읽어 들일 수 있지만 딜레이가 길어서 context swap을 해야 한다.

스케줄러에서 몇 개의 thread를 사용할 것인지는 사용하는 명령의 딜레이에 의해서 결정되며 딜레이를 제어할 수 있을 경우에는 여러 개의 thread를 사용할 필요는 없다. 스케줄러는 길이가 다른 부분과 비교해서 길지 않고 딜레이가 긴 명령어를 많이 사용하지 않기 때문에 딜레이를 제어할 수 있다. 따라서 특별한 문제가 없는 한 2개의 thread를 사용하여 만들 수 있다. Main과 sub 모듈로 만들면 context swap의 개념이 아닌 sub function을 호출하듯이 사용할 수 있다. Main에서는 주요기능을 실행하며 context switching을 시켜야 할 때라도 명령어의 종료를 기다려야 하는 명령어와 딜레이가 긴 작업은 sub thread에 모아서 실행시켜야 한다. 하지만 이 구조는 딜레이가 긴 명령이 dependency에 의해서 같은 주기 내의 긴 딜레이를 가지는 앞 명령어의 결과에 의해서 영향을 받을 때는 사용할 수 없다.

## 4. Transmit Scheduler

Transmit scheduler는 receive thread에서 처리가 끝난 패킷을 출력포트로 전송할 순서를 정하기 위해서 사용하는 스케줄러이다. Transmit scheduling의 처리과정은 그림 3과 같다.

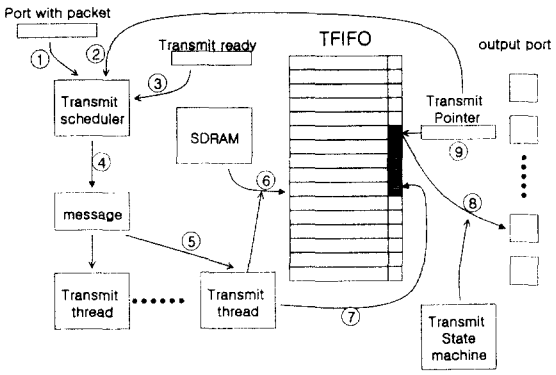


그림 3. Trasmit scheduler의 동작

Receive thread에서 처리가 끝나고 output queue에 저장된 패킷이 있는 포트에 관한 정보를 port with packet을 통해서 얻는다. 또한 전송준비가 되어 있는 output port에 관한 정보를 transmit ready 레지스터를 통해서 얻으며, transmit pointer를 통해서 TFIFO에서 전송이 이루어진 출력포트에 관한 정보를 얻는다. 이 정보를 이용하여 스케줄링을 해서 transmit thread에 message를 전달하면 transmit thread에서 메시지를 읽고서 지정된 TFIFO element로 SDRAM으로부터 패킷을 전송시킨다. 전송이 끝나면 validate bit을 체크한다. Transmit state machine은 transmit pointer 지정하고 있는 elemen의 validate bit이 set되어 있을 경우 전송을 시작하고 transmit pointer를 증가시킨다.

Transmit scheduler도 receive scheduler처럼 출력포트의 전송준비 상태를 읽어야 하며 출력될 패킷이 있는 출력포트에 관한 정보를 읽어야 한다. 또한 TFIFO에서 패킷이 전송대기중인 포트에 관한 정보는 스케줄러 내부에서 출력포트와 TFIFO의 바인딩정보에 의해서 관리하고 있어야 한다. 출력포트에서는 출력가능 상태라고 나오더라도 TFIFO에서 다른 패킷이 전송 대기중이라면 그 출력포트는 더 이상 패킷을 받을

수 없을 수 있기 때문이다 Transmit scheduler가 관리해야 할 자원중의 하나는 TFIFO element이다. 패킷이 TFIFO에 전송된 후 state machine에 의해서 출력포트로 전송되기 때문에 실제 전송이 일어나는 element와 SDRAM에서 읽어 들일 패킷을 저장할 element는 다르게 된다. 따라서 패킷을 읽어 들일 element의 포인터 in_ptr, 이전 주기의 스케쥴링에서 transmit pointer의 위치를 기록한 output pointer를 관리해야 한다. Output pointer를 유지하는 이유는 사용할 수 있는 TFIFO element의 개수가 몇 개인지를 확인하고 실제로 전송이 이루어진 포트를 확인함으로써 binding정보를 변경하기 위해서이다. 만약 스케쥴러가 최대한 할당한 element개수가 16보다 적다면 free element의 count는 필요하지 않게 된다. free element개수가 0보다 작을 경우에는 사용가능한 element가 생길 때까지 기다려야 한다. output pointer와 trasmit pointer를 비교함으로써 TFIFO에서 패킷이 대기중인 포트에 관한 정보를 변경시켜야 한다. 포트에 패킷이 대기중인 포트들에 대한 정보를 모아서 전송준비가 되어 있는 출력포트의 정보를 마스크 시켜서 실제로 전송이 가능한 포트들에 대해서 스케쥴링을 실행하여야 한다. 그림 4에서 TFIFO element를 관리하기 위해서 필요한 포인터를 보이고 있다.

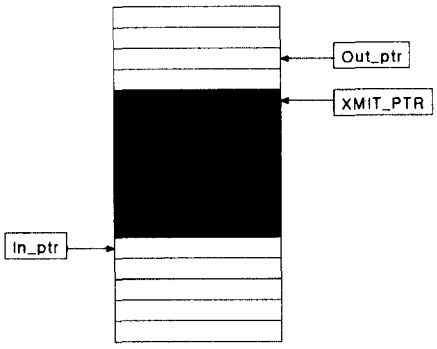


**그림 4. TFIFO element의 관리**

Transmit scheduler의 동작시간은 receive thread에서 패킷을 처리하는 시간보다 빨라야만이 패킷이 drop되지 않고 전송될 수 있다. 동작시간을 계산하는 방법은 receive scheduler에서 사용하는 방법과 같다. 또한 receive thread의 동작시간은 transmit scheduler의 동작시간보다 빨라야만 한다.

Transmit scheduler는 receive thread보다 읽어 들여야 하는 정보가 더 많고 그 정보가 dependency를 가지게 될 경우에는 main과 sub thread의 구조를 유지하기가 힘들어진다. 이것을 해결하기 위해서 4개의 thread를 사용할 경우에는 transmit scheduling thread간에 같은 포트나 TFIFO element에 대한 스케쥴링이 일어나지 않도록 해야 한다.

## 5. Context Switching의 효과

하드웨어 적으로 지원되는 context switching의 효과를 그림 5에 나타내었다.전체 명령의 중간중간에 존재하는 delay부분만을 모아서 하나로 모을 수 있으며 그 결과는 delay부분이 실제 instruction부분에 가려져서 프로세서가 delay가 긴 명령어에 의해서 대기하지 않고 계속 실행되는 것이다. 결과적으로는 하나의 thread가 실행되었을 때의 실행주기에서 딜레이 부분이 빠지게 되는 것이다. 하지만 이 결과를 위해서는 전체 코드 길이가 길거나 딜레이가 긴 명령어의 위치가 적절히 조절 되어야만 한다.

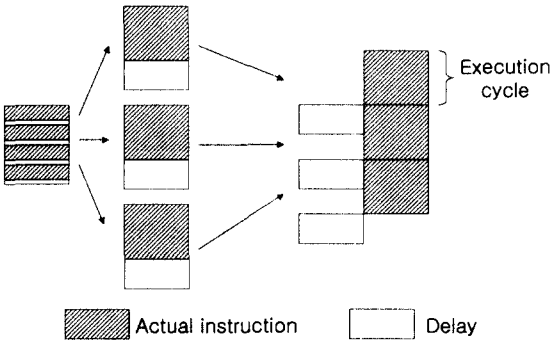


**그림 5. context switching의 효과**

## 6. 결론

네트웍이 발전함에 따라 빠른 속도와 함께 다양한 기능이 요구되어 지고 있다. 이러한 사항을 충족시킬 수 있는 것이 네트웍프로세서이다. 네트웍프로세서에서 스케쥴러를 설계할 때 전체적인 상황을 고려해서 기능을 정하고 동작주기를 맞추어야 한다. 하지만 빠르게 동작시키면서 많은 기능을 제공한다는 것은 상반되는 요구사항이기 때문에 성능을 정하고 그에 맞추어서 스케쥴러 및 처리 모듈의 성능을 맞추어야 한다. 이러한 시스템의 전반전인 사항을 고려하여 설계되어야 만이 스케쥴러가 좋은 효율을 성능을 내면서 시스템을 동작시킬 수 있다.

## 7. 참고 문헌

[1]. IXP1200 Network Processor Programmer's reference, Intel

[2]. IXP1200 Network Processor Hardware Reference Manual, Intel

[3]. TCP/IP Illustrated, Volume 1, Stevens

[4] Multithreading on a Superscalar Microprocessor, Manu Gulati

[5]. Agere, Inc. "The Challenge for next generation network processors", September 10, 1999