스누핑 로직을 이용한 프로세서간 통신에 관한 연구

김경남*, 장경선**, 강동수***
*충남대학교 컴퓨터공학과
**충남대학교 컴퓨터공학과
***충남대학교 컴퓨터공학과
e-mail : dadari@cnu.ac.kr

A Study on Inter-processor communication using snooping logic

Kyeong-Nam Kim*, Kyong-Sun Jhang**, Dong-Soo, Kang***
*Dept. of Computer Engineering, Chung-Nam National University
**Dept. of Computer Engineering, Chung-Nam National University
***Dept. of Computer Engineering, Chung-Nam National University

요 약

현재 프로세서 형태를 보면, 유니 프로세서에서 멀티 프로세서 형태로 바뀌고 있는 상태이다. 프로세서의 집적율이 높아질수록 발열량이 많아지고 성능 면에 있어서 큰 이점이 없기 때문에 작은 클럭으로도 동작할 수 있고 프로세서를 여러 개를 이용하여 여러 개의 일을 처리할 수 있도록 임베디드 시스템이나 PC 환경이 바뀌고 있으며 이러한 환경은 필수적으로 사용되고 있다. 멀티프로세서 환경의 큰 이점은 여러 개의 프로세스를 처리할 수 있는 것이며 대신 프로세서간의 정보교환이 정확해야 이러한 이점을 최대한 활용할 수 있다. 따라서 본 논문에서는 프로세서 간의 정보교환이나 통신을 위한 방법론에 대해 연구한다

1. 서론

프로세서의 발달로 인해 PC 나 PDA 등의 임베디드 기기 사용자들은 높은 수준의 프로그램을 사용할 수 있게 되었다.

이러한 프로세서의 발달은 각 제조사들의 프로세서 집적도를 계속적으로 높여가게 되었으며 일정 수준의 집적도에 도달하였을 때 많은 문제에 부딪히게 되었다. 그 문제는 바로 높은 발열량과 여기에 따른 많은 전력소비였다.

각 프로세서의 제조사들은 이러한 문제를 해결하기 위해 다각도로 노력하였으며 그 결과가 현재 많이 볼 수 있는 멀티프로세서 환경이다. 멀티프로세서는 유니 프로세서보다 낮은 클럭들을 갖고 있는 프로세서들을 연결하여 보다 나은 성능을 나타낼 수 있으며 많은 사용자들에게 높은 호응도를 보이고 있다.

멀티프로세서 환경이 사용자들에게 높은 호응을 보이고 있는 이유는 다름 아닌 여러 개의 프로세스를 실행하였을 경우 처리하는 매커니즘에 있다.

하나의 프로세서로 처리하였을 경우 발생하는 로드를 여러 개의 프로세서가 나누어 처리하는 매커니즘을 사용함으로써 많은 성능 향상을 가져온 것이다.

그런데, 현재 멀티프로세서 시스템 중 임베디드 분야에 대한 멀티프로세서 환경은 아직 활성화 되어 있지 않은 상태이며 환경을 구축하는데 있어 많은 부분이 제약적인 것을 그 이유로 들 수 있다.

그러므로 임베디드 환경에서 멀티프로세서가 갖는 여러 가지 제약적인 부분과 프로세서를 제어하는 매커니즘을 개발하는 것이 필수적이다.

따라서 본 논문에서는 임베디드 환경뿐만 아니라 PC 환경에서도 동작할 수 있도록 유니 프로세서로 작동하는 프로세서를 여러 개 연결하여 멀티프로세서 환경으로 동작할 수 있는 방법에 대하여 연구한다.

2. 프로세서의 한계

현재 사용하고 있는 대부분의 프로세서는 마스터 모듈만이 존재하며 이러한 모듈의 구성으로 인해 가지는 여러 가지 제약이 있다.[1]
예를 들어, 프로세서를 여러 개 사용하는 시스템의 경우 프로세서간 데이터를 주고 받거나 프로세스를 병렬적으로 처리하고 싶을 때 프로세서간의 커뮤니케이션이 어려워진다.

본 논문에서는 이러한 문제점을 해결하기 위해 스누핑 로직을 이용한 방법을 제안한다. 스누핑 로직이란 마스터 모듈인 프로세서뿐만 아니라 주변 장치들의 데이터 전송정보를 감시하는 로직을 말한다.

이러한 정보들은 기본적으로 해당 시스템의 버스 인터페이스 유닛 (BIU)를 통하여 전송이 되며 스누핑 로직은 버스 안에 발생하는 모든 정보를 감시하게 되는 것이다.

2.1 Snooping logic

스누핑 로직은 데이터를 보낼 프로세서, 즉 소스 프로세서로부터 데이터를 받아 이 데이터를 처리할 수 있는 타겟 프로세서에게 데이터를 전송하여 주는

일종의 연결고리가 된다. 소스 프로세서가 보낼 데이터를 메모리의 특정 영역에 쓰게 되면 스누핑 로직은 이 정보를 감시를 통해 알게 되고 해당 데이터를 보낼 프로세서를 선정하여 데이터를 전송하게 된다.

2.2 Inter-processor Interrupt (IPI)[2]

데이터를 받게 되는 타겟 프로세서는 언제 데이터가 올 것인지 알 수가 없다. 소스 프로세서로부터 데이터가 발생될 때까지 기다리게 되면 많은 손실이 발생하게 되며 이는 직접적으로 성능을 저하시키게 된다.

스누핑 로직을 이용한 방법에서는 이러한 문제가 발생되는 것을 방지하기 위하여 프로세서간 인터럽트 방식을 이용하여 데이터를 전송하는 방식을 사용하였다.

보통 프로세서는 외부로부터 인터럽트를 받을 수 있지만 반대로 인터럽트를 외부로 내보낼 수 없다. 그러므로 프로세서를 대신하여 인터럽트를 발생시킬 모듈이 필요하게 되며 이 모듈이 바로 스누핑 로직이다.

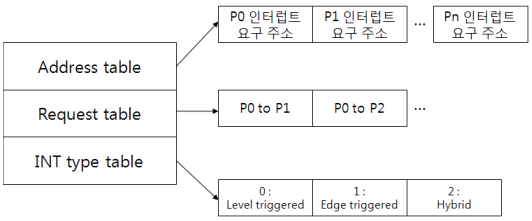
스누핑 로직은 특정 조건이 발생하면 데이터를 받을 프로세서에게 인터럽트를 발생시켜 해당 데이터에 대한 처리를 하도록 한다.

데이터를 받을 타겟 프로세서는 이를 위하여 해당 인터럽트에 대한 인터럽트 서비스 루틴을 갖고 있어 데이터를 처리할 수 있다.

2.3 테이블

스누핑 로직은 인터럽트를 발생시키거나 여기에 대한 응답을 받기 위해 따로 테이블을 갖고 있다.

이 테이블은 프로세서가 어느 영역에 데이터를 썼을 때 어느 프로세서에게 데이터를 전달할 것인지에 대한 여부와 인터럽트를 타겟 프로세서에게 요청하였을 때 여기에 대한 응답이 왔는지 정보를 갖고 있게 된다.



(그림 1) 테이블 구조

(그림 1)과 같이 스누핑 로직 내부의 테이블은 크게 주소 테이블, 인터럽트 요구 테이블, 그리고 인터럽트 타입으로 구성된다.

주소테이블은 가장 먼저 부팅 되는 프로세서로부터 정보를 받게 된다. 메모리의 어느 영역에 데이터를 썼을 때 어디로 인터럽트를 발생시킬지 여부가 이 테이블을 참조하여 이뤄지게 된다.

인터럽트 요구 테이블은 인터럽트를 요청한 정보, 그리고 인터럽트 요청에 대한 응답에 대한 정보를 갖고 있게 된다. 이 정보가 필요한 이유는 소스 프로세서에서 보낸 데이터가 제대로 처리되었는지에 대한 여부를 알 수 있는 중요한 정보이다.

마지막으로 인터럽트 타입 테이블은 인터럽트의 종류에 대해 저장해 놓은 테이블이다. 인터럽트의 발생 종류에 따라 level-triggered, edge-triggered 그리고 두 가지 방식을 혼합하여 사용하는 hybrid 방식으로 나뉘어지는데 이 정보를 갖고 있어야 하는 이유는 프로세서 마다 인터럽트를 처리하는 방식이 모두 다르기 때문이다. 그리고 스누핑 로직에서 인터럽트를 요청하고 응답을 받을 때 이 정보를 알고 있어야 하므로 따로 테이블을 생성하여 저장하도록 한다.

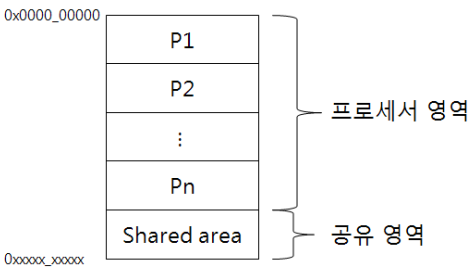
2.4 메모리 구조

본 논문에서 구성한 멀티프로세서 시스템은 기본적으로 유니 프로세서로 동작하는 프로세서들을 연결한 구조이다. 다른 말로 표현하면 시스템 상에 존재하는 모든 프로세서들이 유니 프로세서일 때와 마찬가지로 동작하기 때문에 여기에 대한 메모리 구조가 결정되어 있어야 한다는 것을 의미한다.

만약 4 개의 프로세서를 이용한 시스템을 구성한다면 각 프로세서들은 자신들이 가지는 고유의 메모리 영역이 있을 것이다. 이 메모리 영역은 프로세서 생산부터 이미 결정되어 있는 것이기 때문에 임의로 바꾸어 사용할 수 없다.

2.5 메모리 영역 설정

결국 이와 같은 문제로 메모리 영역을 나누어 프로세서간 메모리를 사용하는데 문제가 발생하지 않도록 구성해야 한다.



(그림 2) 메모리 구조

(그림 2)는 유니 프로세서들을 멀티 프로세서로 만들기 위한 메모리 구조이다. 메모리를 유니 프로세서의 구조에 맞게 고치는 것은 매우 작업이 어렵기 때문에 아예 프로세서들이 필요로 하는 영역을 구하여 각 프로세서가 접근할 수 있는 메모리 영역을 제한하는 것이다.

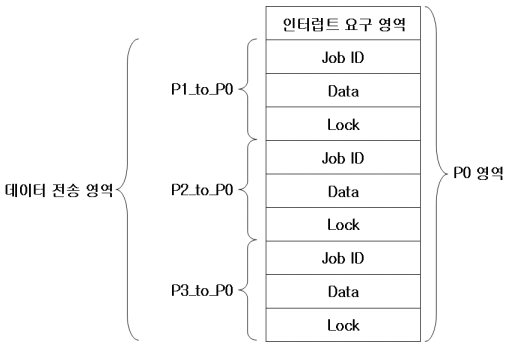
예를 들어 0 번지부터 3 만 번지까지 특정 프로세서가 써야 한다면 이 영역은 해당 프로세서가 사용하도록 하고 또 다른 프로세서가 0 번지부터 3 만 번지까지 또 써야 한다면 버스를 컨트롤 하는 agent 에서 0 에 3 만을 더하여 접근할 수 있도록 한다. 이것은 두

개의 프로세서가 모두 0 번지부터 3 만 번지를 접근하지만 실제로는 에이전트에 의해 하나의 프로세서는 0 번지부터 3 만 번지까지, 다른 프로세서는 3 만 1 번지부터 6 만 번지까지 접근하도록 수정하는 것이다. 이렇게 함으로써 각 프로세서들이 같은 주소를 내보내도 서로 다른 번지에 접근하도록 하여 메모리 사용시 충돌이 발생하는 문제를 해결할 수 있다.

2.6 공유영역

(그림 2)에서 보듯이 메모리 구조 중 특정 부분이 있다. 바로 공유 영역이라고 불리는 곳인데 이곳이 바로 스누핑 로직을 이용하여 프로세서간 통신을 지원하도록 하는 영역이다.

데이터를 보내기를 원하는 프로세서는 이 영역에 데이터를 쓰게 되고 스누핑 로직은 이 정보를 얻게 되면 바로 데이터를 보낼 프로세서를 결정하여 인터럽트를 발생시키게 된다.



(그림 3) 공유 영역 구조

(그림 3)는 공유 영역에 대한 상세 구조이다. 만약 4 개의 프로세서가 있다고 가정한다면 0 번째 프로세서 공유 영역은 위 그림과 같이 구성된다.

먼저 프로세서 0 의 공유영역은 프로세서 1 부터 프로세서 3 까지 접근할 수 있도록 나누고 다시 각각의 영역은 현재 전송된 데이터가 어떠한 job 에 대한 데이터인지 알 수 있도록 job ID 를 저장하는 영역을 갖는다. 그리고 여기에 대한 데이터를 데이터 영역에 저장하여 필요한 정보를 갖추게 된다.

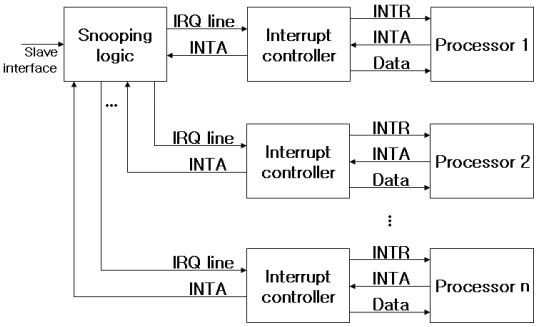
이렇게 해서 인터럽트를 요청하기 위한 준비가 끝나게 되며 마지막으로 인터럽트 요구 영역에 데이터를 쓰면 프로세서 0 가 인터럽트가 발생하게 된다. Lock 정보는 여러 개의 프로세서가 한 번에 같은 프로세서에게 데이터를 보낼 경우가 발생하게 되는데 이럴 경우 하나의 프로세서만 데이터를 쓰고 나머지 프로세서는 대기를 해야 하는데 이것을 나누어 놓기 위해 설정해 놓은 영역이다.

즉, 데이터를 보낼 소스 프로세서가 데이터를 받을 타겟 프로세서의 데이터 전송영역에 데이터를 쓸 때는 lock 을 걸어놓지 않다가 인터럽트 요구 영역에 데이터를 쓸 때 lock 을 걸어 다른 프로세서가 데이터를 쓰지 못하도록 방지하는 것이다.

3. 실험

본 논문에서 제안하고 있는 멀티프로세서 시스템을 직접 8086[3] 프로세서를 이용하여 실험을 해보았다.

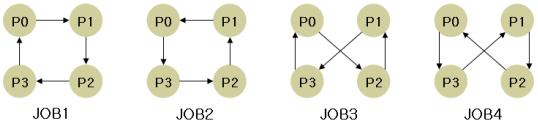
총 4 개의 프로세서를 연결하였으며 여기에서 동작하는 프로세스, 즉 job 의 수를 4 개를 설정하여 동작 유무를 확인하였다.



(그림 4) 스누핑 로직와 프로세서 연결도

(그림 4)처럼 스누핑 로직은 각 프로세서의 인터럽트 컨트롤러와 연결하였으며 슬레이브 인터페이스를 통해 가장 처음 동작하는 프로세서에 의해 내부의 테이블을 초기화하게 된다.

이렇게 초기화된 테이블을 기초로 하여 4 개의 잡을 직접 시스템에서 동작시켰으며 각 잡의 흐름은 아래 그림과 같다.



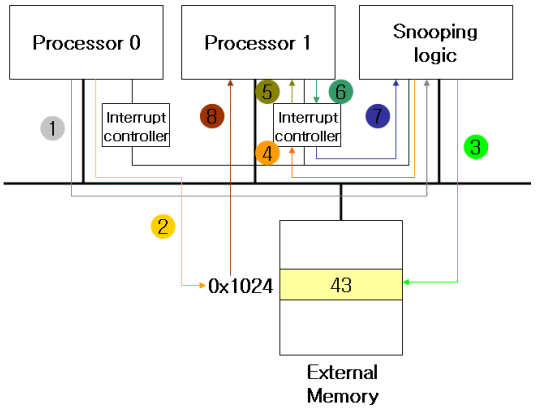
(그림 5) 각 job 에 대한 명세도

Job1 은 프로세서 0 부터 시작하여 프로세서 1, 프로세서 2, 프로세서 3 로 데이터를 전송하여 실행을 하게 되며 다시 프로세서 1 로 돌아와 같은 작업을 반복한다. Job2 은 이와 반대로 실행이 되며 job3 와 job4 도 서로 반대로 수행되도록 하여 모든 경우에 대하여 실험을 할 수 있도록 구성하였다.

만약 프로세서 0 에서 프로세서 1 으로 데이터를 전송하려면 프로세서 0 은 프로세서 1 의 데이터 전송영역에 데이터와 jobID 를 쓰게 되고 마지막으로 인터럽트 전송 영역에 데이터를 쓰게 된다. 스누핑 로직은 이 정보를 알게 되면 즉시 프로세서 1 에게 인터럽트를 요청하게 되고 인터럽트를 받은 프로세서 1 은 여기에 대한 응답을 주고 인터럽트 서비스 루틴을 수행하게 된다.

인터럽트 서비스 루틴은 이미 언급하였듯이, 자신의 데이터 전송영역에 쓰여진 프로세서 0 의 데이터를 읽어오는 루틴을 수행하게 되며 이것을 읽어온 뒤 처리를 하거나 바로 프로세서 2 에게 이 데이터를 전

달하게 된다. 이것은 구현의 차이이며 단지 실험을 위한 루틴으로 작성된 루틴이다.



(그림 6) 프로세서 0 과 프로세서 1 의 동작

프로세서간 통신을 위한 기법은 여러 가지가 있다.

그 중 스누핑 로직을 이용한 방법과 비교하여 볼 수 있는 기법은 MailBox[4]와 3DSP system[5]을 들 수 있다. Mailbox 는 Mailbox 라는 모듈을 이용하여 특정 메모리에 접근하였을 때 다른 프로세서에게 데이터를 전달하여 주는 방식이며 3DSP system 은 프로세서간 인터럽트 방식으로 데이터를 전달하여 주는 방식이다.

이 두 가지 방법과 비교를 했을 때 스누핑 로직을 이용한 방식과 성능상의 차이는 거의 없으나 현재 존재하는 임베디드 시스템이나 기타 시스템들이 갖추어야 할 것 중 하나가 확장성과 신뢰성인 것에 초점을 맞추었다.

만약 멀티프로세서 시스템을 구축하였다 하더라도 하나의 프로세서가 갑자기 동작을 하게 되지 않는다고 한다면 그 시스템은 이미 사용할 수 없게 된다.

하지만 스누핑 로직을 이용한 경우 다시 주소를 재설정 해주면 되기 때문에 이미 죽은 프로세서가 있을 경우 해당 프로세서로 데이터를 보내지 않으면 되며 만약 프로세서가 더 추가된다고 하더라도 테이블 정보만 바꾸어주면 되기 때문에 로직이나 하드웨어를 바꿀 필요가 없다.

Mailbox 나 3DSP system 의 경우 레지스터 등을 이미 고정하여 사용하거나 많은 인터럽트 라인을 사용함으로써 복잡도가 높아 이를 설계하는 설계자 입장에서는 매우 까다로운 문제가 아닐 수 없다. 스누핑 로직은 여기에 착안하여 좀 더 쉽고 빠르게 정보를 바꿀 수 있는 방식을 채택하도록 하였다.

4. 결론

본 논문에서 요약한 바와 같이 현재 프로세서 구조는 멀티프로세서 형태로 가고 있으며 여기에 대한 여러가지 제어 매커니즘들이 나오고 있다.

본 논문은 이러한 매커니즘 중 스누핑 로직을 이용하여 전혀 성능에 지장을 주지 않고 프로세서 간에 통신을 할 수 있는 매커니즘을 구성하였으며 성능면에서나 신뢰성, 확장성에 있어서 다른 방법론보다 좀 더 확장된 개념을 보였다. 앞으로도 이러한 실험은 계속될 것이며 멀티프로세서 제어를 위한 좀 더 성능이 개선된 방법이 필요하며 기술 연구가 지속적으로 추진 되어야 할 것이다.

5. Acknowledgement

본 연구는 교육과학기술부의 과학기술위성 3 호 개발사업의 지원을 받아 수행하였음.

참고문헌

[1] AMBA 2 Specification, http://www.arm.com/products/solutions/AMBA_Spec.html

[2][4] OMAP5910 Dual Core Processor Data Manual Texas Instruments, Jan. 2003

[3] CPU86 8088 FPGA IP Core Version 0.75, http://www.ht-lab.com/freecores/cpu8086/cpu86.html

[5] 3DSP. Application note : Inter-processor Communication