802.11 Wireless LAN 환경에 적합한 프로세서 구조

전성재[O], 홍인표, 이용주, 이용석, 정진우[1]
연세대학교 전기전자공학과 프로세서 연구실
삼성종합기술원 Comm. & Network Lab[1]
{sjjun[O], necross, lemann}@dubiki.yonsei.ac.kr, yonglee@yonsei.ac.kr, jjoung@samsung.com

A Processor Architecture for 802.11 Wireless LAN Environment

Sungjae Jun[O], Inpyo Hong, Yongjoo Lee, Yongsurk Lee, Jinoo Joung[1]
Processor Lab., Dept. of Electrical and Electronic Engineering, Yonsei University, Seoul, Korea
Comm. & Network Lab., Samsung Advanced Institute of Technology, Kiheung, Korea[1]

요 약

최근 휴대폰, PDA, 노트북 등의 모바일 제품의 인기에 따라 모바일에 대한 소비자의 관심이 증대되고 있으며. 대형 네트워크 장비보다 소형의 개인 휴대용의 모바일 제품의 성장세가 두드러지고 있다. 이러한 추세에 따라 무선랜에 대한 관심도 증대되고 있다. 본 논문에서는 기존의 ARM 프로세서를 기반으로 802..11 무선랜 환경에 맞는 네트워크 프로세서 구조에 대한 연구를 수행하였다. 그 결과 전송과 수신이 빈번하게 동시에 일어나는 무선랜 환경에서는 multi-threading을 처리할 수 있는 프로세서가 구조(SMT)가 superscalar 구조에 비해 높은 성능 향상폭을 보여주었다

1. 서 론

현재 무선랜은 11Mbps를 지원하는 802.11b규격이 규격화되어 시장에서 상용적으로 쓰이고 있다. 그러나 기하급수적으로 증가하는 트래픽을 처리하기 위하여 네트워크의 속도규격이 높아지고 있으며 이에 따라 무선랜에서도 24Mbps~54Mbps 속도를 지원하는 802.11a, 802.11g 규격들이 제품이 등장하거나 표준화가 진행중이다. 이처럼 높아지는 속도를 처리하기 위하여 장비들의 성능요구치도 높아지고 있다.

본 논문에서는 일반적인 wire-lan이 아닌 802.11 무선랜 MAC layer operation에 최적화된 네트워크 프로세서 구조에 대해 연구를 수행하였다. 네트워크 프로세서의 기본구조는 embedded system에서 가장 많이 사용되는 ARM processor를 기본으로 하였고 본 연구실에서 보유중인 ARM호환 SMT 시뮬레이터를 이용하여 시뮬레이션을 진행하였다.

2. 802.11 wireless LAN

이번 연구에 쓰인 802.11 wireless 패킷의 종류는 일반적으로 AP에서 쓰이는 관리 프레임을 없애고, 유저 데이터를 위한 프레임만을 구성하였다. 연구에 쓰인 프레임의 종류는 다음의 표와 같다.

표 1 프레임 종류

프레임 종류	기능
User Data 프레임	일반적인 user data를 포함
RTS 프레임	채널의 상황을 보장하기 위한 프레임
CTS 프레임	RTS 프레임에 대한 응답
ACK 프레임	특정 프레임의 성공적인 수신을 알림

먼저 User Data 프레임[1]은 다음과 같은 구조를 가진다.

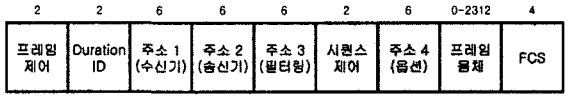

그림 1 User Data Frame

RTS와 CTS의 프레임 포맷은 다음과 같다.

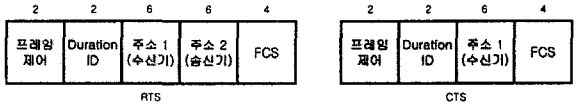

그림 2 RTS, CTS Frame

RTS(Request To Send)프레임은 RTS-CTS(Clear To Send) 클리어링 교환 프레임으로서 전송될 프레임이 RTS 임계값 보다 클 때, 주고 받게 된다. ACK 프레임은 단순전송, RTS/CTS 교환 다음의 전송 프레임, 조각화된 프레임을 포함한 모든 데이터 전송에 사용된다. ACK의 프레임 포맷은 CTS 프레임 포맷과 같다.

본 연구에서는 user data transfer에 관한 시뮬레이션만을 진행하였기 때문에 비경쟁기반의 패킷전송과 관리프레임의 처리는 제외하였다. User data transfer의 기본적인 특징은 경쟁기반의 패킷전송이다. 하지만 몇몇의 중요한 packet은 경쟁기반에 그대로 두면 전송의 지연이 생기고 또한 보내려는 스테이션에서 그 packet을 계속 갖고 있어야 하기 때문에 비경쟁기반으로 packet을 보낼 필요가 있다. IEEE802.11x에서는 SIFS, PIFS(PCF Inter-Frame Space)(PCF : Point Coordination Function), DIFS를 통하여 경쟁 및 비경쟁 채널을 설정한다. 시간 구간을 다르게 함으로써 패킷 간의 경쟁을 조절할 수 있다. SIFS의 경우는 아토믹 동작과 같은 경우에 사용되는 시간 간격으로서 가장 우선도가 높은 패킷을 전송할 때 사용된다.PIFS의 경우는 상용 서비스등에서 사용될 예정으

로 있는 시간 간격이고, DIFS는 우선권이 없는 패킷의 전송에 사용되는 시간 간격이다

3. Simulation Program

Simulation 프로그램은 IEEE 802.11 spec에 맞게 packet를 주고받는 과정을 구현하도록 작성되었다. PC 상의 프로그램 simulation 환경에서는 packet을 주고받을 air가 없기 때문에 그 역할을 파일이 수행하도록 하였다. 아래 그림을 보면

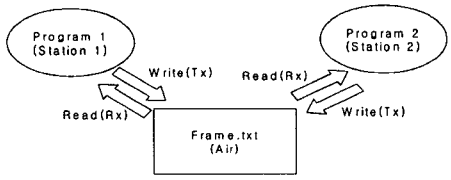

그림 3 시뮬레이션 개념

서로 데이터를 주고받을 station이 각각 s1, s2이라하면 s1은 자기가 s2로 보낼 data를 IEEE 802.11 spec에 따라 packet으로 만든 다음, 정해진 파일(여기서는 frame.txt)에 쓰게 된다. 이 과정이 실제 환경에서는 작성된 packet을 air로 송신하는 과정이라 할 수 있다. s2는 정해진 파일에서 data를 읽어 들이고, packet에 따라 spec에 지정된 동작들을 수행하게 되며, 파일에 저장되는 packet은 IEEE 802.11의 MAC layer의 packet이다.
이 simulation 프로그램은 신뢰성을 높이기 위하여 실제 802.11 system 환경에서 사용 중인 application을 참고로 하여 작성하였다. 실제 system에서 사용 중인 application이다 보니 simulation에서는 구현하기 힘든 하드웨어에 직접적으로 접근하고 관리하는 부분이 다수 삽입 되 있었고, 이런 부분은 각각 상황에 맞춰 다른 코드로 대체하였다.

4. Microprocessor Architecture

프로세서의 처리 성능을 가속하는 데에는 여러 가지 방법이 있지만, 크게 두 가지 분류로 볼 수 있다. 작업의 parallelism을 높여서 병렬 처리하는 방법과 클럭 주파수를 올려서 빠르게 처리하는 방법이다. 그러나 클럭 주파수의 상승은 곧 회로의 전력 소모량 증가로 이어지고 높은 동작 주파수를 갖는 회로의 설계는 많은 설계 복잡도를 유발하므로, 현재의 프로세서 구조는 다중 스레딩 기술(multi-threading)을 실시간 응용에 적용[2][3][4]하는 방향으로 연구되고 있다. 본 연구에서는 multi-threading 기술을 활용하여 parallelism을 통한 성능 향상을 꾀하도록 한다. 그래서 하나의 통일된 마이크로아키텍처 모델을 기반으로 하여 다음과 같은 네 가지 형태의 아키텍처 모델을 시뮬레이션 하였다

- **1-issue in-order scalar processor**
  한 사이클에 하나의 명령어만을 순차적으로 이슈하는 가장 단순한 프로세서 구성.
- **2-issue in-order superscalar processor**
  한 사이클에 최대 2개의 명령어를 순차적으로 실행하는 방식. Dependency 관계를 추적하여 true dependency가 없는 경우 순서대로 명령어를 이슈하며, 한 시점에 하나의 스레드만이 프로세서 내에서 active한 상태가 된다. 스레드 변경을 위해서는 일정한 penalty cycle이 소모된다.
- **1-issue in-order fine-grain multi-threading processor**
  Scalar 프로세서와 같이 한 사이클에 하나의 명령어를 실행할 수 있으나, 실행할 명령어를 여러 스레드로부터 선택할 수 있으므로, 분기 예측 오류, 캐쉬 미스, true dependency 발생 등과 같이 특정 스레드에서 명령어를 이슈할 수 없는 경우에 다른 스레드의 명령어를 이슈할 수 있다. 매 사이클마다 지연시간 없이 스레드를 스위칭 하는 효과를 갖는다.
- **2-issue in-order simultaneous multi-threading processor**
  여러 개의 스레드로부터 최대 2개까지의 명령어를 이슈하여 실행한다. Fine-grain multi-threading과 다른 점은 한 사이클에 여러 개의 스레드로부터 전송된 명령을 동시에 이슈할 수 있다는 것이다.

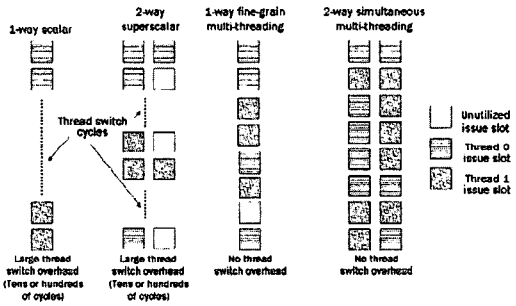

그림 4 Processor architecture model

5. 시뮬레이션

multi-threading 기술로 인하여 성능 향상 효과를 얻기 위해서는 workload가 thread로 적절하게 나뉘어 있어야 한다. 이를 위하여 다음과 같은 workload를 구성하였다.

① Rx + gen_frame
프레임을 전송하기 위하여 payload를 알맞은 크기로 분할하고 헤더를 생성하는 등의 과정을 수행하는 동안, 해당 스테이션으로 user 프레임이 수신되는 상황

② gen_frame + tx_frag
프레임의 크기가 threshold보다 큰 경우, 프레임을 fragmentation하게 되는데, 현재 fragment를 전송하면서 다음 fragment를 생성하는 과정을 수행하고 있는 상황

③ gen_frame + update_NAV
프레임을 전송하기 위하여 payload를 알맞은 크기로 분할하고 헤더를 생성하는 등의 과정을 수행하는 동안 user 프레임이 수신되었는데 다른 스테이션이 destination으로 설정되어 있어 NAV만을 업데이트하고 종료하는 상황

④ gen_frame + Rx_CRC_error
프레임을 전송하기 위하여 payload를 알맞은 크기로 분

할하고 헤더를 생성하는 등의 과정을 수행하는 동안 user 프레임이 수신되었는데 CRC 체크 결과 에러가 발생하여 버려지는 상황

⑤ generate_multiple_fragments
프레임의 크기가 threshold보다 큰 경우, 프레임을 fragmentation하게 되는데, 여러 개의 fragment를 각각 하나의 thread로 간주하여 동시에 fragmentation하는 동작

⑥ Rx + CRC
802.11 MAC의 알고리즘에 따른 Rx동작과 수신된 프레임의 CRC 체크를 별도의 thread로 할당하여 동시에 수행하는 동작

⑦ gen_frame + Rx_RTS
프레임을 전송하기 위하여 생성하고 있는 중에 다른 스테이션으로부터 RTS 프레임이 수신되어 미리 정해진 채널 상황에 따라서 CTS 프레임을 만들어 전송하는 동작

6. 결과

아래의 표는 각 workload별로 마이크로아키텍처에 따른 성능을 작업을 완료하는 데에 소요된 사이클 수로 나타낸 것이다. 위에서 설명한 workload를 1에서 7까지의 번호로 구분하여 표기한 것이며, 5번 workload는 동시에 2개의 thread가 fragment를 생성하는 경우부터 동시에 4 thread가 fragmentation을 하는 경우까지를 시뮬레이션 하였다. 성능 향상 정도는 현재의 embedded processor와 같은 구성을 가진 STSI를 1로 보았을 때, 다른 마이크로아키텍처의 상대적인 성능을 수치적으로 표현하였다.

표 2 마이크로아키텍처에 따른 각 workload의 수행 사이클 수

		STSI	STMI	MTSI	MTMI
1		52087	40273	41362	31356
2		36081	28111	33081	25424
3		50033	38435	39807	28427
4		51603	39826	38539	28450
5	2 frags	64676	49698	43325	30990
	3 frags	97024	74557	62397	36712
	4 frags	129372	99416	82236	44203
6		19759	15434	17396	13381
7		5627	4966	3981	3282

표 3 마이크로아키텍처에 따른 성능 향상 정도

		STSI	STMI	MTSI	MTMI
1		1	1.29	1.25	1.66
2		1	1.15	1.42	1.74
3		1	1.28	1.09	1.41
4		1	1.30	1.25	1.76
5	2 frags	1	1.15	1.34	1.59
	3 frags	1	1.29	1.33	1.81
	4 frags	1	1.30	1.49	2.08
6		1	1.30	1.55	2.64
7		1	1.30	1.57	2.92
average		1	1.15	1.53	1.90

Embedded processor로 구현 가능한 정도의 복잡도를 유지하기 위해서 in-order 방식을 채택하였기 때문에 하나의 thread에서 2개까지의 명령어를 이슈하는 STMI (superscalar) 방식의 성능 향상 폭이 23%로 그다지 크지 않다. 이에 비하여 1개의 ALU만을 사용하는 MTSI (fine-grain multi-threading) 방식의 경우에는 2개의 ALU를 갖는 superscalar 구성에 비하여 높은 성능을 보임으로써 hardware utilization 효율이 뛰어남을 볼 수 있다. Multi-threading을 지원하고 2개의 ALU를 사용하여 issue width로 2로 늘려준 경우에는 91%의 성능 향상 효과를 나타냈다. 이 경우 hardware overhead중에 큰 것은 여러 개의 register file과 추가적인 ALU 정도가 된다. 이러한 hardware overhead를 비교하기 위하여 ARM9 HDL 모델을 합성해서 나온 수치를 기반으로 각 architecture의 gate수를 추정해보았고 그 표는 다음과 같다.

표 4 각 마이크로아키텍처의 area overhead 추정치

	ARM9	STMI	MTSI	MTMI
Control block	10.5K	10K	10K	10K
ALU	4.3K	8.6K	4.3K	8.6K
Shifter	1.1K	1.1K	1.1K	1.1K
Register file	22K	22K	66K	66K
Multiplier	10K	0K	0K	0K
기타	12.9K	12.9K	12.9K	12.9K
Total	60.8K	54.6K	94.3K	98.6K
overhead	0	-10.2%	55.1%	62.2%

위의 표들를 비교해보면 MTSI에서 MTMI로 넘어갈 때에는 area overhead가 큰 차이가 없을걸로 예상되나, 성능은 50% 가량 차이가 나므로 MTMI가 802.11 응용 프로그램을 가속하는 데에는 가장 효율적이다.

7. 결론

본 연구에서는 802.11 wireless LAM 상황에서 네 가지 종류의 마이크로아키텍처를 비교해보았다. 그중에서 가장 복잡한 MTMI 구성은 현재의 일반 내장형 프로세서의 1.6배의 면적을 가지면서 1.9배의 성능을 낼 수 있는 것으로 파악되었다. 이러한 MTMI 구성은 다른 아키텍처 옵션과의 성능/면적 효율을 감안하면 가장 우수하다

8. 참고문헌

[1] Wireless LAN Medium Acesss Control(MAC) and Physical Layer(PHY) Specifications

[2] Steve Kleiman and Joe Eykholt, "Interrupts as threads," ACM SIGOPS Operating Systems Review, Volume 29 Issue 2, April 1995

[3] J. Kreuzinger, A. Schulz, M. Preffer, Th. Ungerer, U.Brinkschulte and C.Krakowski, "Real-time Scheduling on Multithreaded Processors," Seventh International Conference on Real-Time Computing Systems and Applications, December, 2000

[4] U. Brinkschulte, J. Kreuzinger, M.Pfeffer, Th. Ungerer, "A Scheduling Technique Providing a Strict Isolation of Real-time Threads," Proceedings of the Seventh International Workshop on Object-Oriented Real-Time Dependable Systems, pp. 334~340, January, 2002