논문 2007-44CI-4-1

Ad hoc 방식의 PCMCIA 접속에 의한 리눅스 기반의 무선 네트워크 시스템 구현

(An Implementation of the Embedded Linux System on the Wireless Network using Ad hoc PCMCIA Interface)

김 성 호*, 문 호 선*, 김 용 득**

(Sungho Kim, Ho Sun Moon, and Yong Deak Kim)

요 약

본 논문에서는 PCMCIA 접속을 위한 전용 제어 칩을 사용하지 않고, Ad hoc 기법을 활용한 ARM 프로세서 기반의 리눅스 시스템 구현방안을 제안한다. 먼저 PCMCIA접속 신호를 생성하기 위한 조합논리 소자의 구성 및 프로세서와의 접속을 위한 타이밍에 대해 기술하고, 구현된 하드웨어를 제어하기 위한 임베디드 리눅스 디바이스 드라이버에 대한 설계방안을 기술한다. 실질적인 시스템의 구현을 위해 S3C2410A(ARM9)프로세서 기반의 임베디드 리눅스 시스템을 구현하여 PCMCIA접속을 통한 무선네트워크를 구성하였다. 성능평가의 결과로 기존의 전용 제어 칩 내장 시스템의 97.9% ~102.49%의 수행능력을 가지고 있음을 확인할 수 있었으며, 제안된 방안으로 시스템의 성능 저하 없이 프로세서 기반 PCMCIA 접속 시스템이 간소화된다.

Abstract

An embedded system is implemented in this work by removing PCMCIA dedicate controller chip from ARM processor based embedded Linux system. In this paper, we propose PCMCIA interface architecture by using Ad hoc methods for wireless network. The proposed system is developed based on S3C2410A processor and it is interfaced with PCMCIA socket by using combinational digital logic circuits. It is interesting to observe that Ad hoc interface provides 97.9% ~ 102.49% performance when compared with dedicate controller systems. The results indicate that the proposed method simplifies the system without loss of performance.

Keywords: 임베디드 리눅스, PCMCIA, 무선 네트워크, Ad hoc, 디바이스 드라이버

I. 서 론

지난 80년대 후반부터, 수차례에 걸쳐 시장 형성 가능성을 모색해 왔던 무선 랜은 근래 들어 네트워크 시장의 주요 쟁점으로 등장하며 본격적인 시장형성이 진행되고 있다.

무선 랜의 장점은 이동성, 구축의 유연성, 비용의 절감, 확장성에 있으며, 이런 무선 랜은 기존의 유선 랜을 대체 또는 확장한 유연한 데이터 통신 시스템으로서, 무선 주파수 기술 을 이용하여 유선망 없이도 데이터를 주고받을 수 있는 기능의 제공, 즉 유선망에 구속됨이 없이 이더넷이나 토큰링과 같은 전통적인 랜 기술의 모든 장점과 기능을 그대로 제공한다.[1]

무선 랜 시스템은 특정 애플리케이션의 요구에 부응하기 위하여 다양한 형태의 토폴로지들로 구축될 수 있으며, 시스템 구축은 소수 사용자들에게 적합한 독립적인 네트워크 형태로부터 수천의 사용자들에게 로밍을 제공하는 넓은 지역에 이르기까지 쉽게 변경할 수 있다.[1]

또한 최근 몇 년간 임베디드 소프트웨어 기술 중 임베디드 운영체제 기술 변화의 큰 기조 중 하나는 기존

* 학생회원, ** 정회원, 아주대학교 전자공학부
(Dept. of Electronics Engineering Ajou University)
※ 본 연구는 정보통신부 및 정보통신연구진흥원의 해외교수요원초빙사업의 연구결과로 수행되었음
접수일자: 2006년12월20일, 수정완료일: 2007년6월21일

정보가전 시장과 여타의 임베디드 응용 산업을 주도 하던 전통 임베디드 실시간 운영체제들인 VxWorks, pSOS, QNX, VRTX 등에 비해서 임베디드 리눅스가 크게 약진한 것이다.[2] 그 이유는 각종 임베디드 응용에 사용되는 하드웨어의 양적, 질적 발전으로 인해 사용할 수 있는 하드웨어 자원이 늘어남에 따라, 임베디드 응용의 데스크톱화를 꼽을 수 있다. 이는 임베디드 리눅스가 전통 RTOS를 사용하는 것보다 많은 기능을 지원하고, 공개 소스를 이용해 개발속도가 빠르며 개발 시에 전통적인 RTOS를 사용하는 것보다 비용이 적게 드는 것에 기인한 것으로 볼 수 있다.[2]

본 논문의 Ⅱ장 본론에서는 PCMCIA 인터페이스와 시스템에 적용한 임베디드 리눅스 및 실질적인 Ad hoc 방식의 접속방안에 대해 알아보고, Ⅲ장에서는 제안된 방안에 따른 시스템 구현을 통해 제안된 시스템의 성능을 평가하고, 마지막으로 Ⅳ장에서는 결론을 논의한다.

Ⅱ. 본 론

1. PCMCIA 인터페이스

PCMCIA(Personal Computer Memory Card International Association) 카드는 음성, 데이터, 팩스 모뎀, 무선 통신, 네트워크 인터페이스 등과 같은 다양한 각종 메모리와 I/O 디바이스들과 호환이 가능한 것으로 PCMCIA는 PC 카드로 불리는 PCMCIA 카드의 표준화와 교환성을 위해 만들어졌다. PCMCIA의 주요 초점은 PC 카드의 기준과 IBM-PC와의 호환성에 맞춰졌다.[3~4]

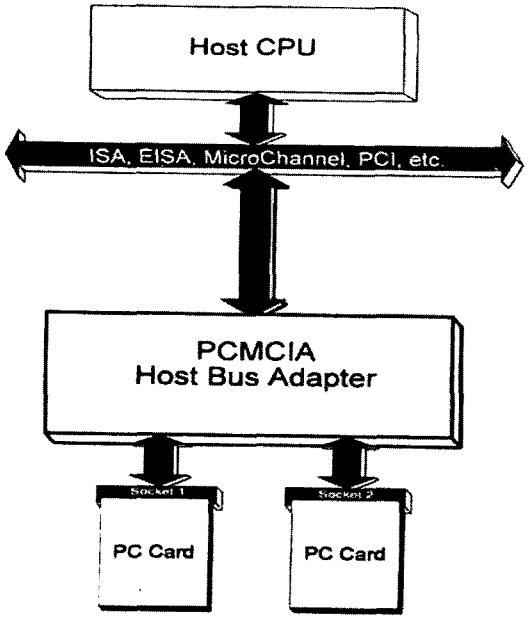


그림 1. PCMCIA 소켓과 호스트 CPU의 접속
Fig. 1. Interface between PCMCIA and Host CPU.

그림 1은 PCMCIA의 소켓과 PCMCIA 호스트 버스 연결기에 대해 나타낸다.[4]

PCMCIA 소켓은 다양한 장치와 연결이 되는데 이 연결은 호스트 버스와 PCMCIA소켓 사이에 있는 PCMCIA HBA(Host Bus Adapter)로 이루어진다. HBA는 호스트 버스와 PC 카드 소켓 사이에서 브리지 역할을 하며 호스트 버스 트랜잭션을 PC카드 쪽으로 전달하는 것이다. 그림 2는 16bit 호스트와의 메모리카드의 연결 방식을 보여준다.[4]

PCMCIA는 두께 및 용도에 따라 아래의 표와 같이 3가지의 종류가 존재하며 본 논문에서는 랜에 사용되는 Type2를 그 대상으로 하고 있다.[3]

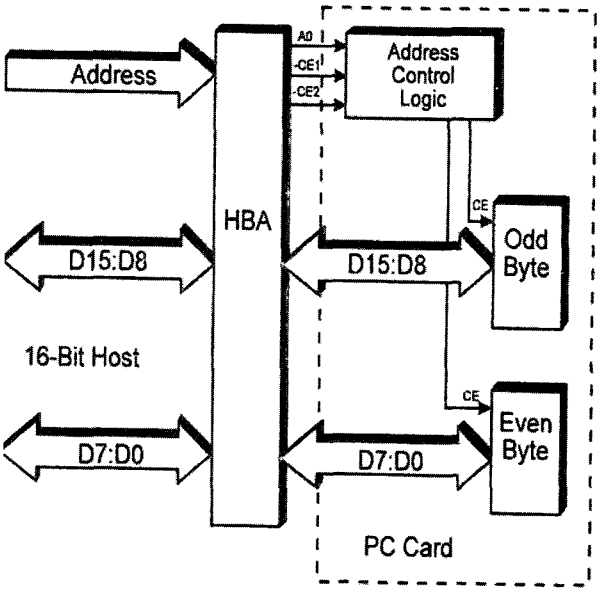


그림 2. 16-Bit 호스트와 메모리 카드의 연결
Fig. 2. Interconnection between 16-bit Host and Memory Card.

표 1. PCMCIA 카드의 종류
Table 1. Type of PCMCIA Card.

종류	크기	두께	용도
Type1	85.6mm, 54.0mm로 통일됨	3.3mm	메모리
Type2		5.0mm	모뎀, 랜
Type2		10.5mm	하드드라이브

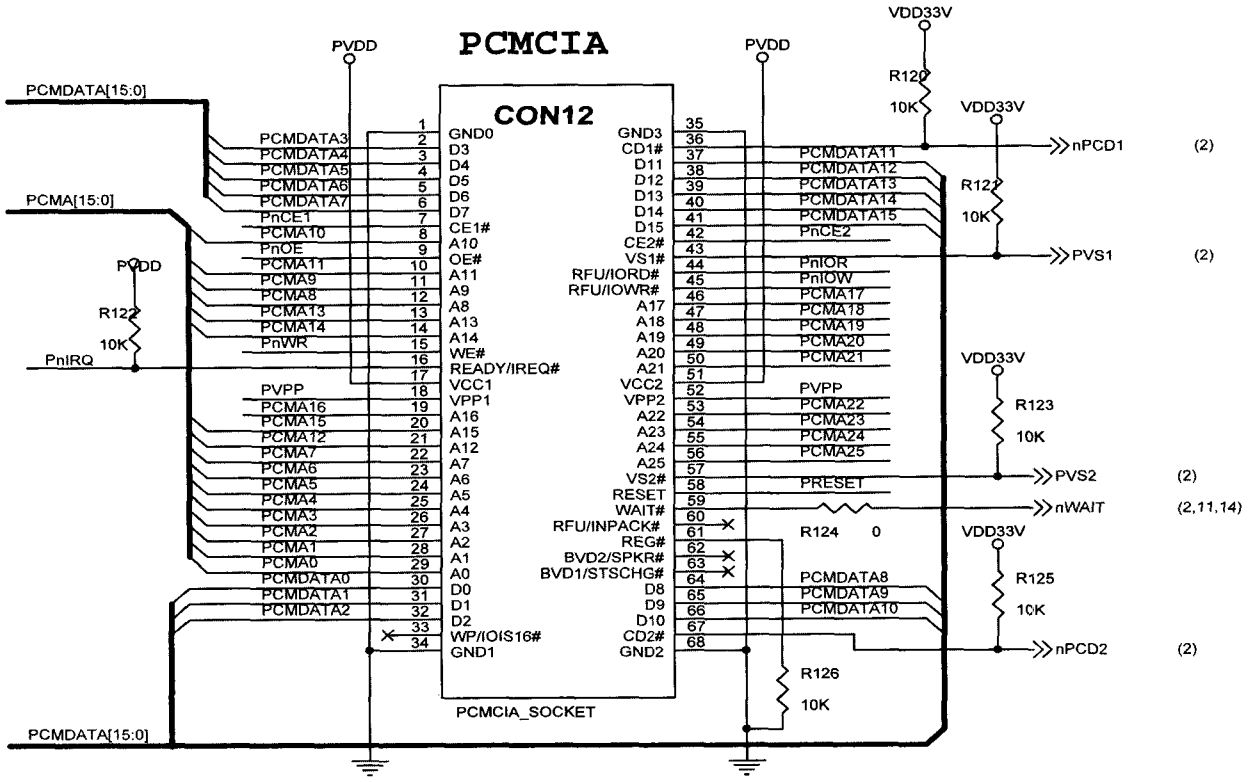


그림 3. PCMCIA 소켓과 CPU의 연결
Fig. 3. Interconnection between PCMCIA Socket and CPU.

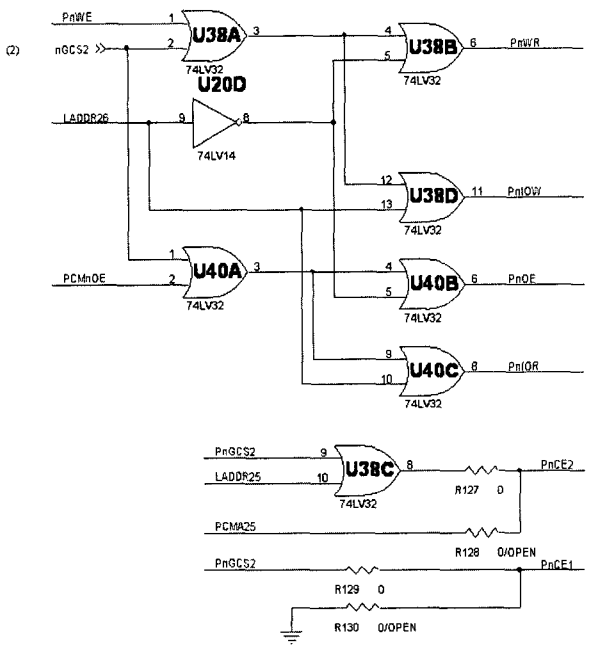


그림 4. Ad hoc 방식의 PCMCIA 제어 회로
Fig. 4. PCMCIA Control Circuit using Ad hoc method.

2. Ad hoc 방식의 PCMCIA 인터페이스의 구현

본 논문에서는 기존의 전용 제어 칩 대신에 최소한의 논리 게이트만을 사용하여 무선 랜 카드만을 지원하는 PCMCIA 인터페이스, 즉 Ad hoc 방식의 PCMCIA 인터페이스를 구현하였다.

Ad hoc 방식의 PCMCIA 인터페이스를 구현하기 위한 접속 회로는 그림 3과 그림 4와 같고 주요 신호는

표 2. PCMCIA 주요 인터페이스
Table 2. The PCMCIA Main Electrical Interface.

신호	내용
PCMA25:0	64MB의 메모리 및 I/O를 액세스 하기 위한 PCMCIA 주소버스
PnCE	Chip Enable을 위한 Active Low 신호 8bit/16bit 액세스 모드
PnIOR	I/O Output Enable을 위한 Active Low
PnIOW	I/O Write Enable을 위한 Active Low
PnOE	PCMCIA 카드로부터 메모리 읽는 동안 Low Assert 되는 신호
PnWR	PCMCIA 카드에 Write하는 동안 Low Assert 되는 신호
nWAIT	Card가 현재 처리중이거나 초기화중임을 나태내주는 Active Low 신호
PCMDATA15:0	16bit 양방향 데이터 버스

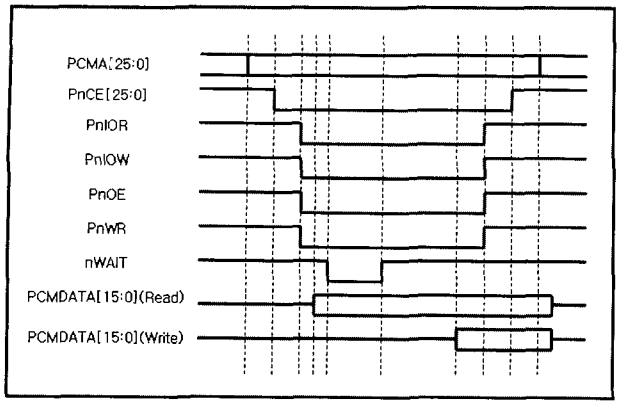


그림 5. Ad hoc 방식의 PCMCIA 제어신호의 타이밍 차트

Fig 5. Timing Chart of Main Signals which control PCMCIA Read/Write Operations.

표 2와 같다.

대부분의 임베디드 시스템에서는 다양한 종류의 PCMCIA 카드를 지원할 필요가 없고 동등한 성능이라면 가능한 한 최소 비용으로 설계하는 것이 유리하다.

그림 4는 기존의 전용 제어 칩 대신에 최소한의 논리 게이트만을 사용하여 구현한 회로로써 기존 방법보다 비용적인 면과 공간적인 면에서 훨씬 효율적임을 알 수 있다.

CPU에서는 Chip Select 신호(nGCS2)를 이용하여 PCMCIA 인터페이스를 선택하고 최상위 어드레스(LADDR26)를 이용하여 IO동작과 메모리 동작을 디코딩 하였다.

그림 5는 그림 4에 의해 만들어진 PCMCIA 제어 신호들의 타이밍 차트로써 PCMCIA 전용 컨트롤러를 사용했을 때와 비교 하여 동일한 Read/Write 타이밍을 가짐을 확인할 수 있다.

그림 6은 PCMCIA 인터페이스와의 전원분리를 위하여 별도의 전원 레귤레이터를 사용하였는데, 현재 사용되고 있는 두 종류의 PCMCIA 무선 랜 카드는 5V용과 3.3V용으로 구분되어 있으므로, VS(voltage Sense: Connector #43, #57)핀을 CPU의 범용 IO 입력으로 연결하여, 두 전원을 선택적으로 공급할 수 있도록 LTC1470을 사용하여 설계하였다.[4~5]

또한, 초반 입력 전원이 불안정하여 VS 핀을 통한 카드인식 이전에 잘못된 전원을 공급할 가능성을 배제하기위해 VS 핀을 Pull-Up 하였고, LTC1470의 전원 제어 핀은 Pull-Down 하였다. 이는 초기 전원 불안정 시 3.3V를 공급하여 최소한의 오류를 방지하기 위함이다. 무선 랜 카드는 삼성의 11Mbps 무선 랜 카드(Magic LAN, WIFI supported) 5V 및 3.3V를 사용하여 실험 하였다.

3. 임베디드 리눅스의 포팅

기본적으로 리눅스의 모든 소스는 공개되어 있으며, 소스와 내부 구조 및 모든 문서가 공개되어 있기 때문에 사용자의 요구에 따라 커널 및 전 계층에 걸친 프로그램의 변경 및 수정이 가능하다.[6]

리눅스는 안정적이며 기능이 검증되어 있다. 전 세계적으로 많은 리눅스 서버가 설치, 운영되고 있고, 안정성 및 보안에 관련한 문제들을 수많은 개발자들이 함께 공유하며 풀어나가고 있기 때문에 배포 버전에 오류가

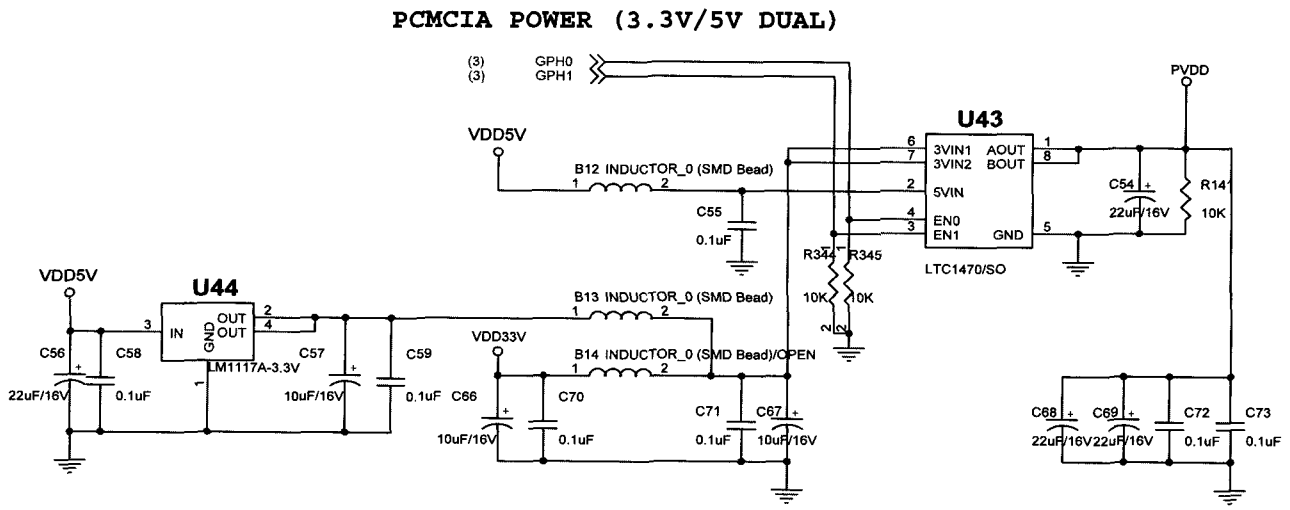


그림 6. 카드 자동인식에 의한 양 전원 공급 회로

Fig. 6. Schematic for 5V/3.3V Dual Power Supply by Detecting Card Type.

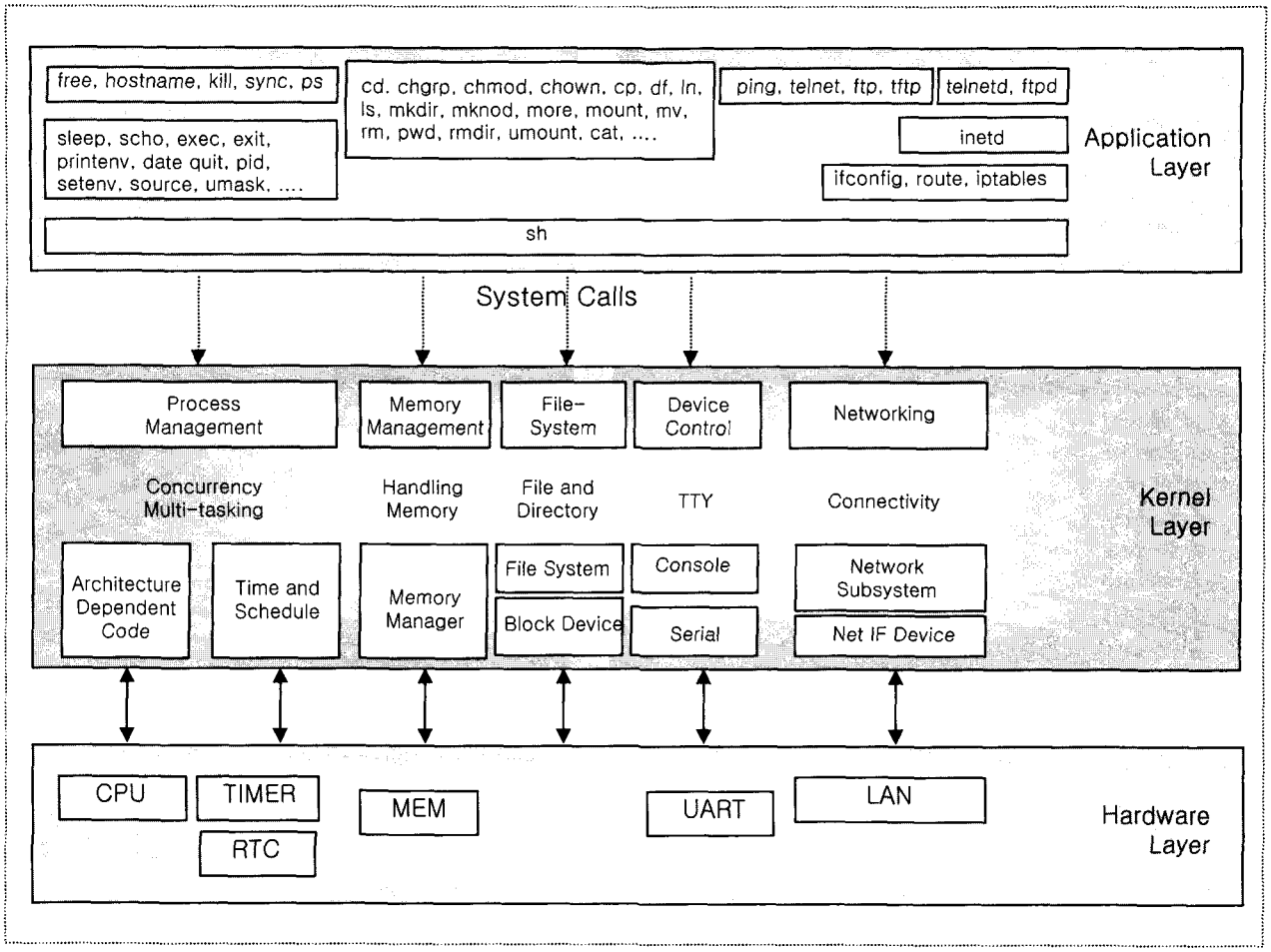


그림 7. 실험용보드에 탑재된 임베디드 리눅스 소프트웨어 계층구조
Fig. 7. The Architecture over view of embedded Linux using proposed systems.

포함될 가능성은 다른 운영체제에 비해 상대적으로 적다고 볼 수 있다.[7~8]

리눅스 커널은 효율적인 멀티태스킹, 페이지 기반의 가상 메모리 시스템, 공유 라이브러리, 인터넷 접속을 위한 TCP/IP, 여러 가지 종류의 파일 시스템을 기본적으로 지원하고 있어 어떤 응용 프로그램의 수행에도 문제가 없다.[9]

본 논문에서는 위와 같은 특징이 있는 임베디드 리눅스를 실험용보드에 포팅 하였다. 그림 7은 실험용보드에 탑재된 임베디드 리눅스 소프트웨어 계층구조를 나타낸다.[10] 임베디드 리눅스는 텔넷, FTP 등의 네트워킹 응용 프로그램과 다양한 네트워킹 커널 모듈을 기본적으로 제공한다. 본 구현에서는 이러한 네트워킹 기능을 사용하여 무선 네트워크 시스템을 구현하였다.

임베디드 리눅스의 디바이스 드라이버에는 특정 하드웨어에 제어명령을 전달하기 위해 동작하는 제어코드가 존재한다. 이 제어코드는 실제 임베디드 시스템 환경에서 특정 하드웨어 제어를 위해 사전에 할당되어 있는 메모리주소에 제어명령을 전달함으로써 하드웨어의 제어를 수행한다.[11~12]

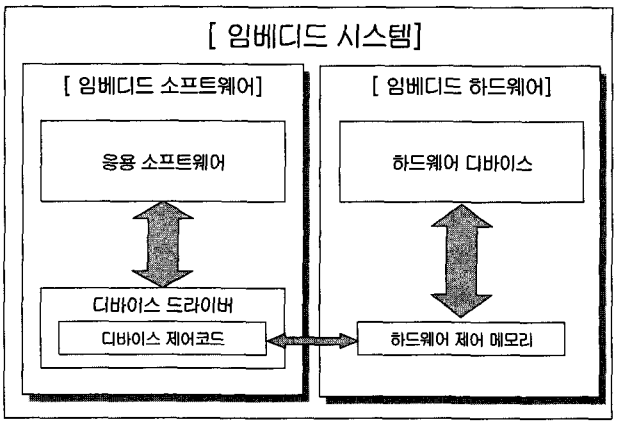


그림 8. 디바이스 드라이버의 하드웨어 제어
Fig. 8. Device Driver Hardware Control Flow.

그림 8은 이러한 과정을 표현하고 있다. 즉, 디바이스

```
/* BANK2, nGCS2 */
#define pBANK2_BASE             0x10000000
#define pPCMCIA_IO_BASE         pBANK2_BASE
#define vPCMCIA_IO_BASE         0xd2000000
#define pPCMCIA_ATTR_BASE       (pBANK2_BASE + (1<<26))
#define vPCMCIA_ATTR_BASE       0xd2800000
```

그림 9. PCMCIA 접속을 위한 주소 정의
Fig. 9. Address definition of the PCMCIA interface.

표 3. PCMCIA 접속을 위한 주소 설명
Table 3. Address description of the PCMCIA Interface.

정의	내용
pBANK2_BASE	PCMCIA 기본 주소
pPCMCIA_IO_BASE	IO 영역의 물리 주소
vPCMCIA_IO_BASE	IO 영역의 가상 주소
pPCMCIA_ATTR_BASE	Attribute 메모리 영역의 물리 주소
vPCMCIA_ATTR_BASE	Attribute 메모리 영역의 가상 주소

드라이버는 대상 하드웨어를 모니터링 하기위해, 해당 주소에 있는 데이터를 참조하고 이를 제어하기 위해서 해당 주소에 특정 데이터를 전달한다. 본 구현에서는 공개된 리눅스용 PCMCIA 무선 랜 디바이스 드라이버인 hostap를 기본으로 사용하였고, PCMCIA 디바이스 드라이버 없이 무선 랜 카드를 제어할 수 있도록 아래와 같이 hostap를 포팅 하였다.

그림 9는 PCMCIA 인터페이스를 위한 주소를 정의한 것이다. PCMCIA 카드는 모두 CIS(Card Information Structure)를 가지고 있고, 이 CIS 정보를 통하여 카드의 Manufacture ID와 특성을 알 수 있다. CIS는 메모리 영역에 할당되어 있고 CIS가 저장되어 있는 메모리 영역을 Attribute 메모리라 한다.[13] 본 구현에서의 메모리 영역은 이 Attribute 메모리에 접근하기 위한 용도로만 사용되었다. 본 구현에서는 MMU 기반의 CPU인 S3C2410A-266을 사용하였으므로 디바이스 드라이버는 물리 주소(physical address) 대신 가상 주소(virtual address)를 사용해야 한다.

그림 10은 무선 랜 카드를 검색하는 단계로써 PCMCIA 카드의 CIS 정보를 읽어 지원 가능한 무선 랜 카드인지 판단하는 부분이다. 이때 CIS 정보를 읽어

```
attr_mem = vPCMCIA_ATTR_BASE;
printk(KERN_INFO "S3C2410 PCMCIA adapter: "
  "IO addr=0x%lx, ATTR addr=0x%x, irq=%d\n",
  vPCMCIA_IO_BASE,attr_mem,IRQ_PCMCIA);
if(prism2_s3c2410_check_cis(attr_mem,
 S3C2410_MAX_ATTR_LEN,
 &cor_offset, &cor_index)) {
  printk(KERN_INFO "Unknown PC Card CIS - not a "
   "Prism2/2.5 card?\n");
    goto fail;
}
printk(KERN_DEBUG "Prism2/2.5 PC Card detected in PCMCIA
adapter\n");
```

그림 10. 무선 랜카드 검색
Fig. 10. Probing of the wireless LAN card.

```
dev = local->dev;
dev->irq = IRQ_PCMCIA;
dev->base_addr = vPCMCIA_IO_BASE;
local->attr_mem = attr_mem;
local->cor_offset = cor_offset;
if (prism2_init_dev(local))
  goto fail;
  if (prism2_hw_config(dev, 1)) {
    goto fail;
  }
```

그림 11. 무선 랜카드 초기화
Fig. 11. Initialization of the wireless LAN card.

```
#define HFA384X_OUTB(v,a)
  outb((v), dev->base_addr + (a))
#define HFA384X_INB(a)
  inb(dev->base_addr + (a))
#define HFA384X_OUTW(v,a)
  outw((v), dev->base_addr + (a))
#define HFA384X_INW(a)
  inw(dev->base_addr + (a))
#define HFA384X_INSW(a, buf, wc)
  insw(dev->base_addr + (a), buf, wc)
#define HFA384X_OUTSW(a, buf, wc)
  outsw(dev->base_addr + (a), buf, wc)
```

그림 12. 무선 랜카드의 접속 정의
Fig. 12. Access definition of the wireless LAN card.

오기 위해 Attribute 메모리 영역의 가상 주소가 사용되었다.

그림 11은 무선 랜 카드의 초기화 단계로써 리눅스 디바이스 드라이버 구조체에 PCMCIA 디바이스 정보를 등록하고 무선 랜 카드를 초기화 한다.

```
if (ev & HFA384X_EV_TXEXC) {
  prism2_txexc(dev);
  HFA384X_OUTW
    (HFA384X_EV_TXEXC, HFA384X_EVACK_OFF);
}
if (ev & HFA384X_EV_RX) {
  if (!prism2_rx(dev))
  HFA384X_OUTW
    (HFA384X_EV_RX, HFA384X_EVACK_OFF);
}
```

그림 13. 인터럽트 서비스 루틴
Fig. 13. Interrupt Service Routine.

그림 12는 무선 랜 카드의 액세스를 정의한 것으로써 Read/Write 시 IO 영역의 가상 주소가 사용된다. 무선 랜 카드를 액세스할 때 반드시 여기에 정의한 것을 사용하도록 되어 있다.

그림 13는 ISR(Interrupt Service Routine)으로써 외부 이벤트 발생 시 송신 동작에 해당하는 인터럽트인지 데이터의 수신에 해당하는 인터럽트인지를 검사하여 그림 12에서 정의한 매크로 함수의 호출을 통해 실질적인 데이터 송수신동작을 수행하는 PCMCIA 디바이스 드라이버의 핵심 부분이며, 무선 랜 패킷 전송은 모두 이 부분을 통해 이루어진다.

Ⅲ. 실 험

1, 실험용 보드의 구성

실험용 보드는 이번 성능 평가에 사용된 보드로서 ARM920T코어의 S3C2410A-266MHz MCU와 16MB의 NOR FLASH, 64MB SDRAM, 10Mbps 유선 랜 (CS8900A) 및 Ad hoc 방식으로 구현된 PCMCIA 인터페이스로 구성되어 있으며 주요 기능 블록은 그림 14와 같다.

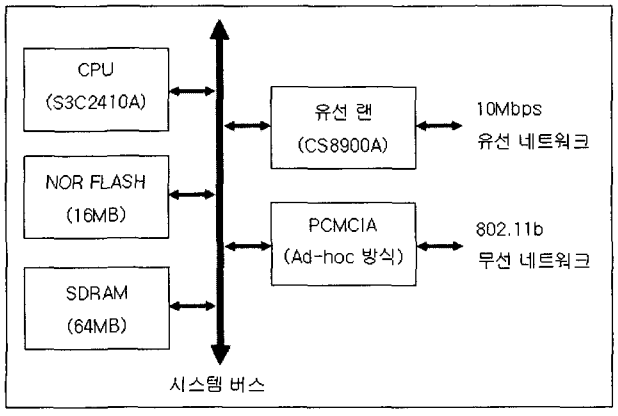


그림 14. 주요 기능 블록도
Fig. 14. Functional Block Diagram.

2. 실험 환경 및 결과

PCMCIA접속을 위해 제시한 실험용보드의 적용가능성을 검토하기 위하여 표 4와 같은 환경에서 기본적인 기능들을 실험 하였다.

그림 15는 본 논문에서 구현한 실험용 보드의 사진이며, 그림 16과 같이 세 대의 실험용 보드를 무선 네트워크에 접속하여 그 성능을 시험하였다.

비교대상보드는 SMDK2410을 사용하였다. 비교대상보드의 하드웨어 사양은 표 4의 실험용보드와 동일하고 비교대상보드의 운영체제 또한 실험용보드와 동일한 임베디드 리눅스를 사용 하였다. 단 비교대상보드는

표 4. 실험용보드 사양
Table 4. Specification of Test Bed.

항목	사양
CPU	S3C2410A-266
RAM	64MB SDRAM
FLASH	16MB(Nor Flash)
Network	CS8900 10-Mbps
	802.11b 무선 랜
Serial	RS-232C (3 Port)
File System	Ramdisk
Kernel	Linux-2.4.18
U-boot	1.1.4
host PC	P4 2.8GHz, Redhat9.0

그림 15. 실험용보드
Fig. 15. The picture of Test bed.

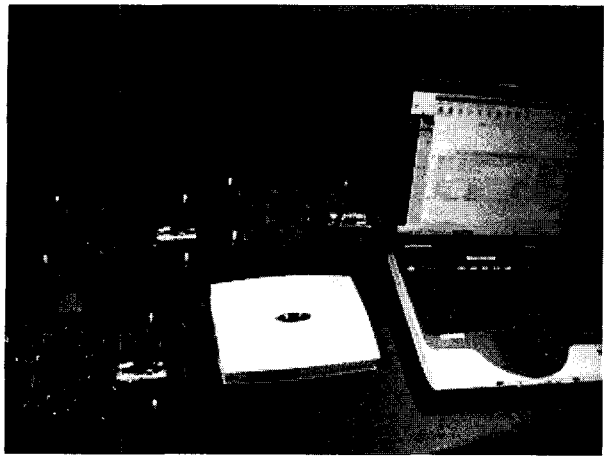


그림 16. 성능시험환경
Fig. 16. Test Environment.

PCMCIA 인터페이스 구현을 위해 전용 제어 칩을 사용하였고 실험용보드는 Ad hoc 방식을 사용하였다. 성능평가를 위한 측정 항목은 아래와 같다.

(1) 대용량 파일전송에 의한 전송 속도 비교
(2) 패킷 사이즈별 평균 왕복 시간
(3) 유선 네트워크와의 동시동작에 의한 성능저하

그림 17은 비교대상보드인 SMDK2410을 사용한 FTP서버에서의 대용량 파일 전송속도를 측정한 그림이며, 윈도우 기반서버의 로컬 네트워크 속도측정 기능을 이용하였으며, 평균 4.97Mbps의 성능을 보였다.

그림 18은 동일 환경에서의 실험용보드 성능측정 결과로서 평균 4.98Mbps의 전송속도를 보였으며, 비교대상보드와 동일한 특성을 나타내고 있다.

패킷크기별 평균왕복시간 측정실험에서는 표5와 같은 결과를 보였으며, 비교대상보드의 시간을 기준으로 실험용 보드의 성능을 측정하면 97.9%~102.49%의 거

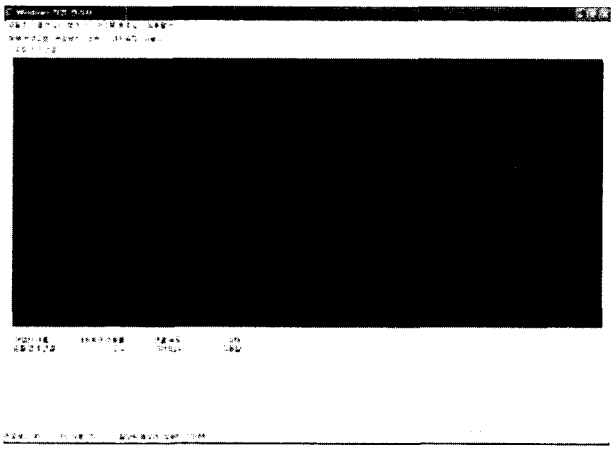


그림 17. 비교대상보드의 전송속도
Fig. 17. Transfer Speed in SMDK2410 Board.

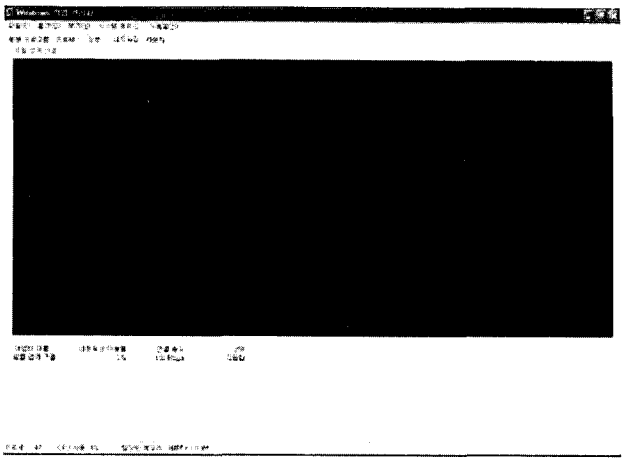


그림 18. 실험용보드에서의 전송속도
Fig. 18. Transfer speed in test board.

표 5. 패킷크기별 평균 왕복 이동 시간 비교
Table 5. Round Trip Time under variable packet size.

패킷크기	비교대상보드 (msec)	실험용보드 (msec)	성능 (%)
1000	4.100	4.202	102.49
2000	7.410	7.303	98.56
4000	11.520	11.735	101.87
8000	21.550	21.976	101.98
20000	52.153	51.082	97.95

표 6. 유선 랜에서의 동시 동작 성능평가
Table 6. Performance evaluation while Ethernet is running

패킷크기	비교대상보드 (msec)	실험용보드 (msec)	성능 (%)
1000	4.623	4.502	97.38
2000	7.524	7.322	97.32
4000	12.015	12.120	100.87
8000	22.552	22.486	99.71
20000	53.232	53.191	99.92

의 동일한 성능을 보임을 알 수 있다.

또한 유선네트워크와의 동시동작으로 인한 성능저하를 측정한 결과는 표 6과 같으며 그에 따른 성능저하가 발생하지 않음을 나타낸다.

IV. 결 론

본 논문에서는 기존의 전용 제어 칩 대신에 최소한의 논리게이트만을 사용하여 무선 랜 카드를 지원하는

PCMCIA 인터페이스를 구현하였다. 이러한 Ad hoc 방식은 기존 방법에 비해 전용 제어 칩을 사용하지 않으므로 비용적인 면과 공간적인 면에서 효율적이다. 그리고 실험용보드와 비교대상보드와의 성능비교를 통하여 Ad hoc 방식이 기존 방식에 비해 97.9%~102.49%의 거의 동일한 성능임을 보여, 성능 저하가 없음을 확인할 수 있었다.

또한 최근 임베디드 시스템의 소형화, 무선화, 저 전력화와 더불어 'embedded everywhere'라는 말이 암시하듯 유비쿼터스의 열풍으로 강력한 네트워크 기능이 요구되는 추세를 고려할 때, 전용 제어 칩을 사용하지 않고 메모리 타이밍을 이용한 구현은 CPU의 활용도를 높임은 물론, 조정 가능한 신호 타이밍을 사용함으로서 가변적인 인터페이스 개수 및 속도에 유연하게 대처할 수 있다는 점과 공개된 소스를 사용하는 임베디드 리눅스를 탑재하여 다양한 네트워킹 기능을 지원한다.

향후 다양한 시스템에서의 검증을 통해 시스템의 필요사양을 낮추고, 리눅스에서 지원하는 어플리케이션을 활용한다면 다양한 임베디드 시스템에 적용 할 수 있고, 이는 저가의 비용으로 전용 제어 칩으로 구현한 시스템과 동일한 성능을 얻을 수 있을 것으로 기대된다.

참 고 문 헌

[1] 황규희, "2003 기술산업 정보분석 무선 LAN", 2003.

[2] 이형석, 정영준, "임베디드 운영체제 커널 기술 동향", 전자통신동향분석 제 21권 제 1호, 2006년 2월

[3] 이한, "PCMCIA를 이용한 입원환자용 환자의료정보통신 시스템 구현", 석사학위논문, 아주대학교 대학원, 2004.

[4] Don Anderson, "PCMCIA System Architecture", Mindshare Press, 1994.

[5] Doug La Porte, "Voltage Interface Issues for PCMCIA Sockets", WESCON'95 Conference record, pp.153-160, NOV, 1995.

[6] 안효복, "ARM으로 배우는 임베디드 시스템.", 한빛미디어, 2006.

[7] 이무영, 조상영, 남상엽, "32비트 ARM9을 이용한 임베디드 시스템 설계와 응용", 상학당, 2005.

[8] www.kelp.or.kr

[9] Hollabaugh, "Embedded Linux hardware, software and Interfacing", 2001.

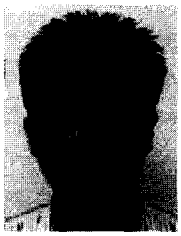
[10] www.kesl.org

[11] 최재현, 이우진, 정기원, "재사용성 및 신뢰성을 고려한 리눅스기반 임베디드 디바이스 드라이버 개발 기법", 정보처리학회논문지C, 제12-D권, 제7호, 1065-1070쪽, 2005년 12월

[12] Alessandro Rubini, "Linux Device Drivers.", O'Reilly, 2000.

[13] Michael T. Mori, W. Dean Welder, "the PCMCIA Developer's Guide 3rd Edition", Sycard Technology, pp.99-116, 1999.

저 자 소 개



김 성 호(학생회원)
1999년 아주대학교 화학공학과 학사 졸업.
2007년 아주대학교 전자공학과 석사 재학 중.
<주관심분야 : 임베디드 시스템, Embedded Linux, 컴퓨터>

문 호 선(학생회원)
1998년 아주대학교 산업공학과 학사 졸업.
2005년 아주학교 전자공학과 석박사통합과정 수료
<주관심분야 : 통신, RTOS, 임베디드 시스템, ITS>

김 용 득(정회원)
1971년 연세대학교 전자공학전공 (학사)
1973년 연세대학교 전자공학전공 (석사)
1978년 연세대학교 전자공학전공 (박사)
1979년~현재 아주대학교 전자공학부 정교수
1973년~1974년 불란서 E. S. E 전자공학 연구실
1973년~1974년 미국 STANFORD 대학교 연구교수
1981년~1982년 한국전자통신연구소 위촉연구위원
1994년~1998년 ITS 연구기획단연구위원, 전자부문총괄
<주관심분야 : 통신, 컴퓨터, ITS>