상용 디지털 TV를 위한 임베디드 리눅스 시스템

문 상 필[†] · 서 대 화[††]

요 약

아날로그 TV에 비해 새로운 방식의 디지털 TV는 단순한 영상과 음성의 처리뿐만 아니라, 데이터를 처리 및 저장해야 한다. 그리고 필요에 따라 데이터를 요구하기 위한 리턴 채널을 관리해야 한다. 이처럼 많은 기능을 동시에 충족시켜줘야 하기 때문에 디지털 TV시스템에서는 운영체제가 필요하다. 임베디드 리눅스는 오픈 소스로서 가격 경쟁력을 확보할 수 있으며, 공개된 장치 드라이버와 응용프로그램을 재사용 가능하며, 오픈 소스 공동체를 통해 문제를 쉽게 해결할 수 있고, 쉘, 파일 시스템을 이용하여 편리한 개발 환경을 제공해 주는 이점이 있다. 본 논문에서는 디지털 TV 시스템을 위한 임베디드 리눅스를 이식하기 위해 교차 개발 환경과 커널을 빅 엔디언으로 변경하였고, 커널 구동에 필수적인 장치들을 재설계하였으며, 커널을 메모리에 적재할 수 있도록 시스템 메모리 맵을 재설정하였다. 또한 시스템 장치제어를 위한 디바이스 드라이버를 작성하였다.

Embedded Linux for Commercial Digital TV System

Sang-Pil Moon[†] · Dae-Wha Seo[††]

ABSTRACT

A Digital TV system is necessary for data processing as well as video and audio processing. Especially in the case of interactive broadcasting, it should manage returning channel created by the Internet, PSTN, and so on. Because of many functionalities and multitasking jobs, it needs an Operating System. Embedded Linux as open source program can increase a cost effectiveness in market and has many advantages - reusable device drivers and application programs, more convenient developing environment using shell and file system, and easy problem resolution within Open Source Community. In this paper, we modified Embedded Linux kernel and cross developing environment for a big-endian system, redesigned devices for kernel execution, and configured system memory map in order to load a linux kernel. Also we developed an device driver for entire system control.

키워드 : 임베디드 리눅스(Embedded Linux), 디지털 TV(Digital TV)

1. 서 론

텔레비전의 다매체 성향에 따라 지상파방송의 경쟁력 강화를 위하여 디지털 방송이 본격적으로 추진되고 있다. 이에 따라 디지털 TV 시스템의 수요가 급격히 증가하고 있다.

디지털 TV 시스템을 설계함에 있어 기존의 아날로그 TV 시스템에 사용되는 마이크로 컨트롤러를 디지털 TV 시스템에 적용할 경우 디지털 방송이 요구하는 다양한 수신 능력을 발휘하지 못한다.

다채널·고화질을 추구하던 아날로그 TV 시스템과는 달리 디지털 TV 시스템은 영상과 음성의 처리뿐만 아니라 데이터 채널로 입력받은 정보를 처리해야 한다. 따라서 고성능 프로세서 및 대량의 데이터를 저장하기 위한 고용량 메모리가 필요하며 이러한 자원을 효율적으로 활용하기 위해 임베디드 운영체제가 제공되어야 한다.

디지털 TV 시스템에 적용된 임베디드 운영체제로 pSOS™와 같은 상용 운영체제가 있다. 그러나 상용 운영체제를 사용할 경우 TV 시스템을 생산할 때마다 로열티를 지불해야 되므로 생산 비용이 증가한다. 이에 반해 오픈소스 OS인 임베디드 리눅스를 사용하게 되면 생산 비용을 절감 할 수 있다. 뿐만 아니라 소스가 공개된 장치 드라이버와 응용 프로그램들을 재사용 가능하기 때문에 개발 기간을 단축할 수 있으며, 모듈 기능과 쉘·파일 시스템을 이용하여 보다 편리한 시스템 개발 환경을 구축할 수 있다. 또한 시스템 개발 중 문제가 발생할 경우 오픈 소스 공동체를 통해 해결할 수 있다.

† 준 회 원 : 경북대학교 대학원 전자공학과
†† 종신회원 : 경북대학교 전자전기공학부 교수
논문접수 : 2003년 7월 7일, 심사완료 : 2003년 11월 28일

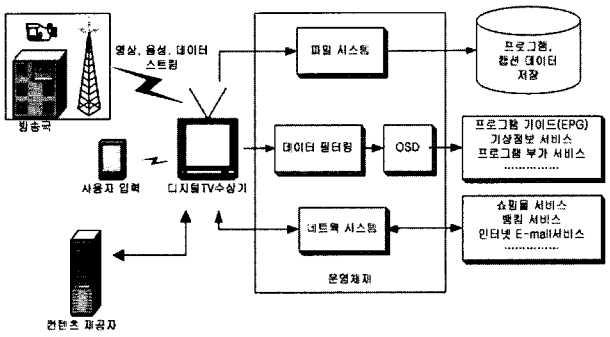
본 논문에서는 한국형 디지털 방송 수신이 가능한 LCD 프로젝션 TV에 임베디드 리눅스 환경구축에 대해 기술한다. 이 시스템에 리눅스를 적용하기 위해 교차 개발 환경과 커널을 빅 엔디언으로 변경하였고, 커널 구동에 필수적인 장치들을 재설계하였으며, 커널을 메모리에 적재할 수 있도록 시스템 메모리 맵을 재설정하였다. 또한 전체 시스템 장치제어를 위한 디바이스 드라이버를 작성하였다.

본 논문의 구성은 다음과 같다. 2장에서는 디지털 TV 운영체제의 역할 및 필요성에 관해 기술하고, 3장에서는 상용 디지털 TV 시스템의 구성 및 하드웨어에 관해 고찰한다. 4장에서는 리눅스를 적용하기 위한 이식과정을 제시하고, 5장에서는 개발 환경을 보이고, 6장에서는 연구 결과물로서 장치 제어 결과와 최종적으로 출력되는 화면을 보여주고, 마지막으로 결론을 제시한다.

2. 디지털 TV 운영체제

디지털 TV는 아날로그 TV에 비해 고화질, 고음질서비스를 제공한다. 주사선수와 수평해상도의 증가로 고화질 서비스를 제공하며, AC3·MPEGII등의 오디오기술을 통해 고음질서비스를 제공한다. 하지만 보다 혁신적인 장점은 디지털 기술을 이용한 데이터 방송 서비스 제공이다.

데이터 방송 서비스는 프로그램 시청 중에 제공 받을 수 있는 종속형 서비스, 프로그램 시청과 관계없는 독립형 서비스, 리턴 채널을 활용한 양방향서비스로 구분된다. 이러한 서비스가 가능하기 위해서는 데이터의 처리가 필요하다. 즉, 수신된 디지털 스트림에서 추출된 데이터를 구분하여 저장해야 하며, 동시에 필요한 부분을 화면에 출력하기위해 데이터를 재구성해야 한다. 필요에 따라서는 현재 방송중인 프로그램을 하드디스크에 저장하는 작업을 해야 한다. 이외에 본격적인 양방향 서비스가 시작되면 전화선·인터넷을 통해 생성되는 리턴 채널 관리 및 전자상거래(T-commerce)와 같은 애플리케이션 수행을 위해 더 많은 데이터 처리가 요구된다[1].



(그림 1) 디지털 TV 운영체제 역할

디지털 TV 수상기에 수신된 디지털 방송 신호는 영상, 음성, 데이터를 포함하는 하나의 스트림 형태로 수신된다. 수신된 신호는 RF 모듈에 의해 복조되어 영상은 MPEG II(Video) 신호로, 음성은 AC3·MPEG II(Audio)로 필터링되며, 데이터는 PAT, CAT, PMT, TSDT(ATSC) 혹은 NIT, SDT, EIT, TDT(DVB)들이 혼합된 형태로 분류된다[2][3]. 예를 들어, EPG(Electronic Program Guide)와 같은 서비스를 위해서는 각각의 테이블로부터 수집된 정보들이 재구성 및 저장되고, OSD(On-Screen Display)구동 장치에 의해 프레임 형태로 생성되어 화면에 출력된다. 이 과정에서, 지속적으로 수신되는 프로그램 정보를 재구성해야 하며, 동시에 이를 분류 및 저장해야 한다. 또한 화면에 출력하기 위한 프레임을 구성해야 한다. 이러한 작업들을 동시에 실행시켜 주기 위해 운영체제가 필요하다(그림1).

시판되고 있는 디지털 TV시스템은 Windriver사의 pSOS™와 같은 상용 운영체제를 탑재하고 있다. 이러한 상용 운영체제는 안정성 및 시스템 최적화가 이루어져 있는 장점이 있지만, 한대의 TV를 생산할 때마다 지불해야 하는 로열티의 부담으로 제품 가격이 상승하게 된다. 이에 반해 오픈 소스 OS인 임베디드 리눅스를 적용할 경우 로열티의 부담이 없으므로 가격 경쟁력을 높일 수 있고, 다음과 같은 장점이 있다.

- 공개된 장치 드라이버 및 응용 프로그램 재사용 가능.
- 모듈 기능 및 쉘, 파일 시스템 지원으로 편리한 개발 환경 구성.
- 개발 과정에서 문제 발생시, 오픈 소스 공동체를 통한 빠른 해결.

이러한 리눅스의 장점을 살려 디지털 TV 시스템에 이식하기 위해서는 시스템 구성에 관해 세밀히 분석해야 하고, 시스템의 목적에 맞게 운영체제를 구성 및 최적화해야 한다. 다음 장에서는 리눅스 이식과정의 준비단계로서 상용 디지털 TV 시스템 구성 및 기능에 관해 기술한다.

3. 상용 디지털 TV 시스템

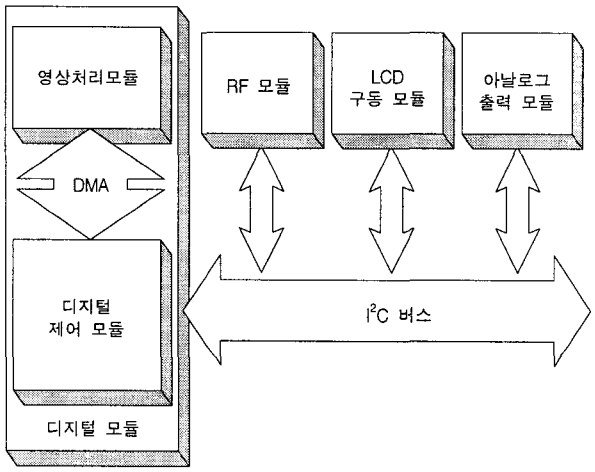
3.1 디지털 TV 시스템 구성

디지털 TV 시스템은 디지털 제어 모듈, 영상처리 모듈, RF 모듈, LCD 구동 모듈, 아날로그 출력 모듈로 구성되어 있다(그림 2).

디지털 제어 모듈은 디지털 TV 시스템에 장착된 여러 모듈들과 장치들을 제어하고, EPG 정보 처리 등의 여러 응용 프로그램을 수행하는 역할을 한다. 디지털 제어 모듈에는 시스템 제어 프로세서(MCU), SDRAM, 플래시 메모리,

시리얼 버스, I²C(Inter IC)버스 인터페이스 등으로 구성되어 있다.

영상처리 모듈은 RF 모듈에서 추출된 데이터를 디코딩하여 영상, 음성, 텍스트 정보로 분류하여 처리하는 모듈이다. 이 모듈은 데이터를 디코딩하는 신호 처리 장치(SPU)와 화면에 출력 될 데이터 프레임을 저장할 SDRAM으로 구성된다. RF모듈은 변조된 디지털 방송 신호를 수신하여 디지털 데이터 스트림으로 복조한다. LCD 구동 모듈은 스크린 확대 방식의 TV에서 광원으로 사용되는 LCD 패널을 제어한다. 아날로그 출력모듈은 음성 출력, 편향판 제어, 외부 출력 신호등을 제어한다. RF · LCD · 아날로그 출력 모듈들은 I²C 버스를 통해 디지털 제어 모듈에 의해 제어되고, 영상 처리 모듈은 외부 버스를 이용해 DMA로 제어된다.



(그림 2) 디지털 TV 시스템 모듈 구성

임베디드 리눅스를 이식할 때, 직접적인 영향을 주는 모듈은 CPU, 메모리, 그리고 외부 입출력 장치와의 인터페이스를 포함하는 디지털 제어 모듈이다. 다음 절에서는 디지털 제어 모듈의 하드웨어 요소들에 대해 기술한다.

3.2 하드웨어 요소 고찰

디지털 TV 시스템에서 리눅스 커널의 역할은 시스템 자원을 효율적으로 관리하여 다중 프로그램 환경을 제공하는 것이다. 운영체제가 하드웨어 자원을 효율적으로 관리하기 위해서는 하드웨어의 기본 요소에 대한 분석이 필수적이다. 따라서 임베디드 리눅스를 디지털 TV 시스템에 이식할 때, 다음과 같은 하드웨어 요소들의 기능과 특성에 대한 분석이 뒷받침되어야 한다.

- CPU 구조 및 기능
- 메모리 접근 방식
- 메모리 맵
- 인터럽트 컨트롤러
- 타이머
- 시리얼 버스 인터페이스
- 외부 버스 인터페이스

커널을 새로운 목표 시스템에 이식하기 위해서는 시스템 CPU 구조 및 기능을 분석해야 하고 이를 통해 어셈블리어 수준의 저급언어에 대한 이해가 필요하다. 즉, 명령어 체계에 대한 이해, 시스템 레지스터 및 상태 레지스터의 파악, CPU 동작 모드, 예외 처리 상황에 관한 분석이 요구된다.

메모리 장치에 관한 고찰은 메모리 맵 설정과 접근 방식에 대한 이해이다. 시스템이 구동되기 전의 임베디드 리눅스 커널은 플래시 롬과 같은 영구 기억소자에 저장된다. 그리고 커널이 동작되기 위해서는 램으로 커널 코드를 옮겨야 한다. 또한 동작 중에 외부 장치를 접근하는 일이 자주 발생한다. 이러한 플래시 롬, 램, 외부장치들은 시스템 메모리 맵이라는 선형적인 주소 체계를 가지고 있고 동일한 방식으로 접근이 가능하다. 따라서 시스템 메모리 맵을 설정하기 위한 기억 소자의 크기 및 특징에 관한 고찰이 필요하다. 뿐만 아니라 CPU가 메모리에 접근하는 방식에 대한 이해가 요구된다. 이식해야할 목표 시스템의 CPU는 빅 엔디언 메모리 접근 방식을 사용하는지, 혹은 리틀 엔디언 방식을 사용하는가에 대한 사전 인지가 필요하다.

그리고 운영체제의 수행을 위해 타이머, 인터럽트 컨트롤러, 버스 인터페이스 등의 시스템 장치에 관한 분석이 요구된다. 운영체제 커널 핵심 기능 중의 하나인 스케줄링을 위해서는 정해진 시간 간격마다 인터럽트를 생성해주는 타이머와 CPU에게 타이머 인터럽트 발생을 알려주는 인터럽트 컨트롤러가 필요하다. 그리고, 각각의 장치들을 연결하는 시스템 버스와 전체 시스템 제어를 위해 MCU와 외부 장치 사이의 연결 통로인 외부 버스 인터페이스에 관한 분석이 필요하다.

3.2.1 CPU 구조 및 기능

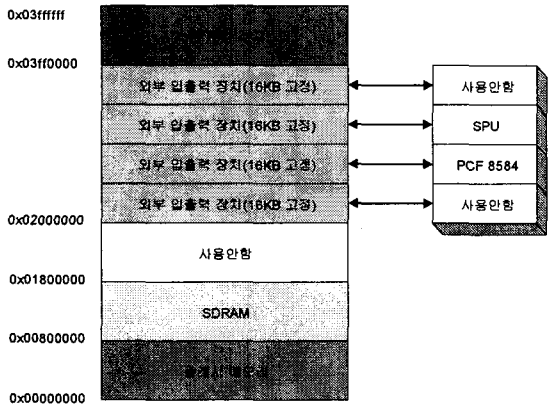
디지털 제어 모듈에 적용된 프로세서 코어는 ARM7TDMI로 저전력의 16/32bit RISC 프로세서이다. 이 코어 로직은 MMU를 포함하고 있지 않다. 따라서 리눅스 이식 후 응용 프로그램 작성 시, 가상 메모리를 사용할 수 없었으며, 메모리 보호 기능도 적용할 수 없었다. 실제로 커널의 개입 없이 장치 드라이버의 맵핑된 영역을 바로 접근 할 수도 있었다. 그리고 ARM7TDMI 프로세서 코어는 6개의 동작 모드로 구성되며, 37개의 시스템 레지스터와 6개의 상태 레지스터를 지니고 있고, FIQ, IRQ, Abort, Software Interrupt, Undefined Instruction Trap등의 5가지 예외 상황을 두고 있다[4].

3.2.2 메모리 접근 방식

ARM7TDMI 프로세서 코어는 리틀 엔디언과 빅 엔디언 메모리 접근 방식 모두를 지원한다. 그러나 디지털 TV 시스템에 적용된 MCU는 빅 엔디언 메모리 접근 방식만을 지원한다[5].

3.2.3 메모리 맵

디지털 제어 모듈의 메모리 자원은 전원이 차단되었을 경우에도 운영체제 및 실행 프로그램을 보관할 플래시 메모리, 실제 코드가 수행될 SDRAM, 외부 장치에 데이터를 전송하기 위한 입출력 영역, 그리고 MCU내의 내부 시스템 레지스터와 장치 레지스터를 들 수 있다. 이러한 영역들은 사용자 입장에서 선형적인 주소 영역으로 볼 수 있다.



(그림 3) 시스템 메모리 맵

메모리 맵은 MCU 내부의 시스템 관리 레지스터에 의해 설정된다. 그리고 디지털 제어 모듈의 16M바이트 SDRAM, 8M바이트 플래시 메모리, 영상 처리 모듈의 SPU, I^2C 버스 인터페이스 장치, 내부 집적 장치 및 시스템 레지스터로 구성된다. 각각은 뱅크로 구분 할 수 있으며, 메모리 종류, 크기 및 버스 폭, 접근 클록 타임에 따라 시스템 관리 레지스터의 외부 버스 제어 레지스터, 갱신 레지스터, 뱅크 제어 레지스터에 의해 설정 가능하다(그림 3).

3.2.4 인터럽트 컨트롤러

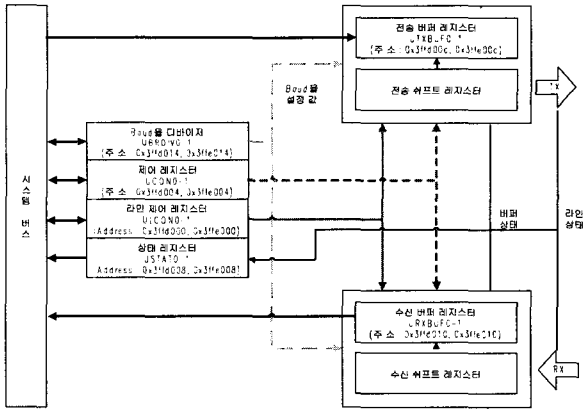
디지털 제어 모듈에 적용된 인터럽트 컨트롤러는 21개의 인터럽트 자원을 가지고 있다. MCU 내부 장치로부터 17개, 외부 장치로부터 4개의 인터럽트 요청을 받는다. 인터럽트 컨트롤러는 내부 혹은 외부 장치에서 발생된 인터럽트 신호가 전달될 때, 인터럽트 팬딩 레지스터에서 해당하는 번호의 인터럼트 요구를 CPU에게 알리는 역할을 한다[5]. 인터럽트가 처리되는 과정은 인터럽트 컨트롤러에 의해 요청이 있었음을 전달받은 CPU가 인터럽트 종류에 따라 동작 모드를 IRQ 혹은 FIQ 모드로 전환하여 처리하게 된다. 즉 인터럽트 요구가 팬딩된 것들 중에서 우선순위가 높은 인터럽트에 해당하는 서비스 루틴으로 분기하여 실행하고, 처리가 끝나면 CPU의 원래 모드로 복귀하는 과정으로 끝이 난다[4].

3.2.5 타이머

디지털 제어 모듈에 적용된 타이머는 32비트 인터벌 모드 타이머이다. 리눅스 커널 스케쥴링 및 커널 타임의 기본 단위인 10msec 클록을 생성하기 위해서 타이머 장치의 카운터 및 데이터 레지스터 값을 설정하여 100Hz의 주기를 가지는 클럭 신호가 발생되도록 해야 한다.

3.2.6 시리얼 버스 인터페이스

커널 초기화 과정에서 출력되는 커널 메시지, 프로그램 오류와 관련된 메시지 등의 디버깅 정보를 보기 위해 콘솔 장치가 필요하다. 임베디드 시스템에서는 별도의 콘솔장치가 없기 때문에 UART(Universal Asynchronous Receiver Transmitter)와 같은 시리얼 포트 장치를 이용하여 외부의 PC로 데이터를 전송하여 커널로부터 출력되는 메시지를 받아볼 수 있다. 본 논문에서 적용된 MCU는 두개의 동일한 UART 포트를 내장하고 있으며 구성 및 컨트롤 블록 다이어그램은 (그림 4)와 같다.

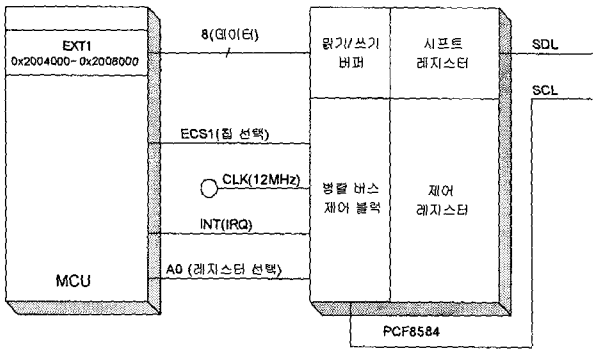


(그림 4) UART 구성 및 컨트롤 블록 다이어그램

MCU 메모리 맵에 매핑된 RX · TX 버퍼 레지스터와 쉬프트 레지스터는 실제적인 데이터 송 · 수신에 필요한 공간을 제공한다. 그리고 라인 제어 레지스터, UART 컨트롤 레지스터, 상태 레지스터는 데이터 전송에 필요한 송 · 수신 모드 결정, 에러 체크 등의 역할을 한다. 예를 들어 송신의 경우, TX 버퍼 레지스터에 전달된 데이터는, 상태 레지스터를 통해 현재 데이터 전송이 가능한지 혹은 오류 여부를 확인한 후, UART 컨트롤 레지스터에 의해 설정된 모드와 Baud율 레지스터에 의해 정해진 속도로 시리얼 라인으로 데이터를 전송한다.

3.2.7 외부 버스 인터페이스

디지털 TV 시스템의 RF 모듈, LCD 구동 모듈, 아날로그 출력 모듈은 I^2C 버스를 통해 제어된다. 따라서 MCU의 병렬 외부 버스와 I^2C 시리얼 버스 라인 사이의 인터페이스 장치가 필요하다. 본 논문에서 적용된 버스 인터페이스 장치는 필립스사의 PCF8584이다. (그림 5)는 MCU와 I^2C 시리얼 버스 라인의 연결 상태를 보여주고 있다.



(그림 5) MCU 와 I2C 시리얼 버스 라인의 연결 상태

MCU의 외부 버스 영역으로 지정된 메모리에 PCF8584의 읽기 및 쓰기 버퍼가 맵핑 되어있다. 외부 버스 영역(EXT1)에 쓰여진 데이터는 맵핑된 PCF8584장치의 읽기 및 쓰기 버퍼로 전송되고 시프트 레지스터에 의해 SDL(Serial Data Line)로 전송된다. 이와 동시에 병렬 버스 제어 블록을 통해 SCL(Serial Clock Line)로 클록이 전송된다. 이러한 데이터 및 클록의 전송은 I^2C 버스 프로토콜을 만족 시켜주기 위해 제어 레지스터에 의해 PCF 알고리즘을 통해 제어된다. 여기에서 I^2C 버스 프로토콜은 버스를 통해 연결된 칩들 사이의 통신을 위한 버스 규약, 즉 Slave 주소와 모드, ACK, 전송 시작 및 종료 조건 등을 정의해 놓은 것이다[6]. 그리고 PCF 알고리즘은 I^2C버스 프로토콜을 만족시키기 위해 인터페이스 장치인 PCF8584를 제어하는 알고리즘이다[7].

4. 리눅스 이식

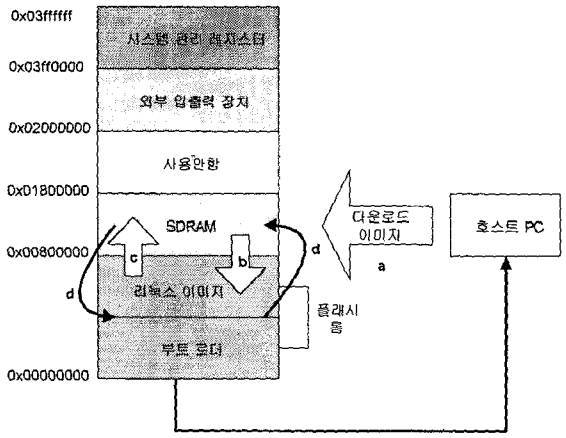
이 장에는 하드웨어 구성에 관한 분석을 바탕으로 하여 임베디드 리눅스 이식 과정에 관해 기술한다. 리눅스를 디지털 제어 모듈에 이식하는 과정은 커널 초기화 코드 재구성, 장치 드라이버 생성으로 이루어진다.

4.1 커널 초기화 코드 재구성

리눅스를 이식하기 위해서는 하드웨어에 의존적인 커널 초기화 코드를 재구성해야 한다. 재구성할 부분은 리눅스 이미지 로딩을 위한 메모리 맵 설정 및 변경, 커널 동작의 필수 장치 재구성 등이다.

4.1.1 메모리 맵 설정 및 변경

리눅스 이미지는 압축된 상태로 디지털 제어 모듈의 플래시 메모리에 적재되고, 실행 시에 SDRAM으로 이동되어 압축 해제된다. 그리고 압축 해제된 커널 코드를 메모리에 재배치하여 로딩 과정에서 메모리 영역을 차지하였던 코드를 제거해야 한다. 이러한 일련의 과정은 시스템 메모리 맵이라는 선형적인 주소 내에서 이루어진다.



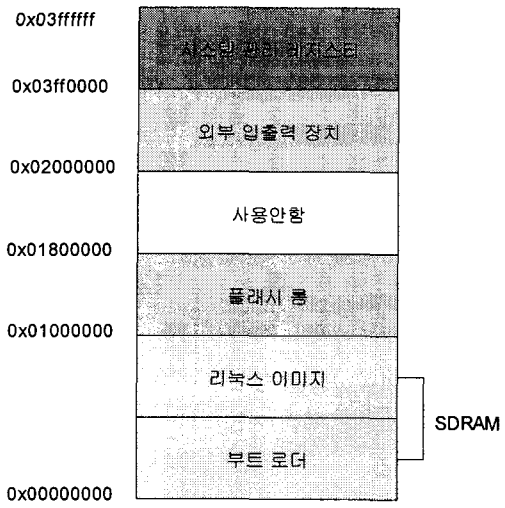
(그림 6) 메모리 맵 설정과정

(그림 6)은 메모리 맵 설정 과정을 보여 주고 있다. 시스템 초기화 시, CPU는 플래시 롬 영역만을 인식하고 가장 하위 부분에 삽입된 부트로더부터 실행한다. 부트로더는 리눅스 커널 이미지를 호스트 PC로부터 디지털 제어 모듈로 다운로드하여 플래시 롬과 같은 영구 저장장치에 저장 및 실행하는 역할을 한다.

<과정 a>는 부트로더가 호스트 PC로부터 리눅스 이미지를 디지털 제어 모듈의 SDRAM 영역으로 옮기는 과정을 나타낸다. 그리고 SDRAM에 다운로드된 리눅스 이미지는 플래시 롬의 부트로더 다음 영역에 저장된다<과정 b>.

이후 시스템이 초기화 되고, 부트 로더는 플래시 롬에 저장된 리눅스 이미지내의 실행 코드(head.o)로 제어를 넘겨준다. 제어권을 넘겨받은 코드(head.o)가 실행되면서 플래시 롬 전체 내용을 SDRAM 영역으로 옮긴다<과정 c>.

플래시 롬에서 SDRAM 영역으로 이미지가 복제되면 시스템 관리 레지스터를 이용해 SDRAM과 플래시 롬의 시작 주소를 바꾼다<과정 d>. 즉, 0x0로 시작되는 플래시 롬 뱅크 시작 주소와 0x800000로 시작되는 SDRAM 뱅크 시작 주소를 서로 바꾼다. 이렇게 하면, 플래시 롬에 있는 리눅스 이미지가 SDRAM 영역으로 로드된 후, SDRAM 영역의 리눅스 코드(head.o)를 프로그램 카운터 변경없이 수행할 수 있게 된다. (그림 7)은 최종 메모리 맵 상태를 나타낸다.



(그림 7) 최종 메모리 맵 상태

플래시 롬에서 SDRAM 영역으로 복사된 리눅스 이미지는 압축 해제를 통해 실행 가능한 형태가 된다. 리눅스 이미지의 압축을 풀기 위해 bss_section과 user_stack, malloc 영역 등의 압축 해제 코드를 실행하기 위한 메모리 공간이 필요하다. 메모리 영역의 할당은 이미지(vmlinux) 생성에 사용되는 링크 스크립트(vmlinux.lds)를 통해 각각의 영역이 할당된다. 그리고 이러한 영역들은 리눅스 이미지에 포함된 최초 실행코드(head.o)에 의해 디지털 TV 시스템의 SDRAM에 생성된다. 실제적인 압축 해제는 생성된 영역을 이용하여 압축 해제 코드(misc.o)에 포함된 gunzip 도구를 이용하여 이루어진다.

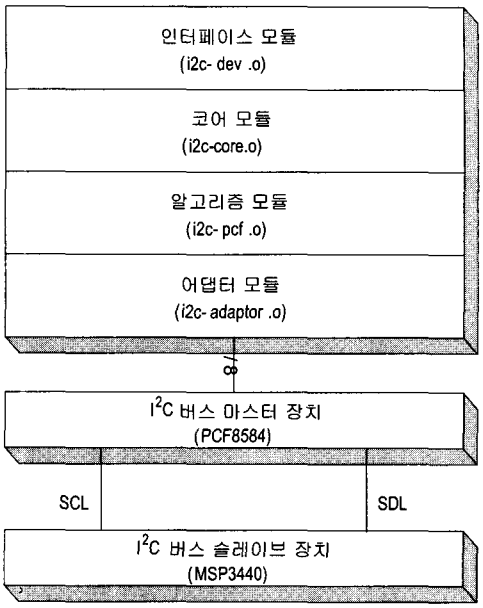
압축이 풀려진 커널 코드가 위치하는 부분은 malloc영역 이후 주소부터이다. 그리고 malloc 영역 이전까지의 주소에 들어 있는 코드들은 커널 압축이 풀리면 필요 없게 되므로 이 영역을 덮어쓰는 재배치 과정을 거쳐야 한다. 커널 코드를 재배치 할 경우 재배치 코드가 malloc영역 이하의 주소에 위치하고, 재배치 과정에서 덮어쓰기가 되기 때문에 재배치 코드를 압축이 풀린 코드 이후로 옮겨 두어야 한다.

4.1.2 커널 동작의 필수 장치

커널 동작의 필수 장치는 인터럽트 컨트롤러, 타이머, 콘솔장치 등이 있다. 이러한 장치들은 MCU 내부 레지스터로 매핑되어 있고, 각 장치와 관련된 레지스터의 재구성을 통해 커널 동작에 필요한 기능을 제공한다.

4.2 장치 드라이버 설계

디지털 TV 시스템을 구동하기 위한 장치에는 커널 동작의 필수 장치 외에 영상 처리 장치와 아날로그 제어 장치가 있다. 본 논문에서는 디지털 TV 시스템의 핵심 버스로 이용된 I^2C 버스 인터페이스인 PCF8584의 장치 드라이버를 설계 및 구현하였다.



(그림 8) PCF8584 장치 드라이버 구조

PCF8584 장치 드라이버는 일반적인 리눅스 장치 드라이버와 같이 모듈 구조로 이루어진다. 인터페이스 모듈, 코어 모듈, 알고리즘 모듈, 어댑터 모듈로 구성되어 있으며(그림 9), 각각의 기능은 다음과 같다.

- 인터페이스 모듈
  - 장치 드라이버와 응용프로그램사이의 인터페이스 제공.
  - 장치 파일 오퍼레이션 정의(open, read, write, ioctl, release).
  - 장치 파일 초기화(/dev/i2c0) 및 생성.
- 코어 모듈
  - 어댑터의 추가/제거 함수 정의.
  - I^2C 버스 기능 정의(transfer(), master_send/receive(), control()).
  - /proc 상태정보 파일 시스템에 관한 초기화.
- 알고리즘 모듈
  - PCF8584 장치 초기화.
  - PCF8584 버스 인터페이스 장치의 기능 함수 정의 (i2c_start/stop(), wait_for_pin()).
  - 어댑터 모듈
  - 메모리에 매핑된 주소를 이용 실제 데이터 전송 (setbyte(), getbyte()).

PCF8584 인터페이스 장치를 구동하기 위해 모듈의 초기화가 필요하다. 일반적인 리눅스 시스템은 장치 드라이버를

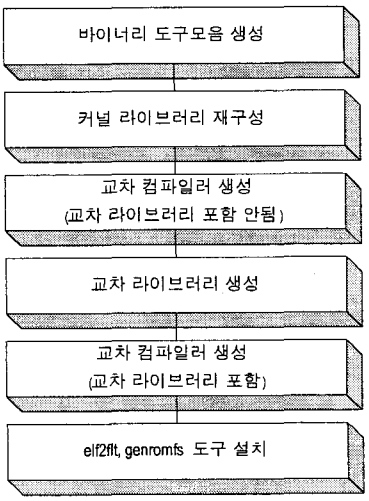
커널 동작 중에 모듈 형태로 삽입 하고, 그 과정에서 드라이버를 초기화한다. 그러나 하나의 이미지로 생성되어 롬에 삽입되는 임베디드 리눅스의 경우에는 부팅 과정에서 초기화 되고, 동적 삽입이 아닌 정적 연결에 의해 동작한다. 4가지 모듈 내에 포함된 초기화 루틴은 이미지 링크 과정에서 '.init '섹션 내의 심볼인 '_init '으로 저장되어 부팅과정의 do_initcalls() 함수를 통해 초기화된다.

장치 구동 과정은 인터페이스 모듈, 코어 모듈, 알고리즘 모듈, 어댑터 모듈의 순서로 이루어진다. 응용프로그램으로부터 요청(open, write, read, ioctl)을 받은 인터페이스 모듈은 전달받은 메시지를 인자로 하여 코어 모듈내의 I^2C 버스로의 기능적 인터페이스 함수(master_send(), master_receive(), transfer())를 호출한다. 호출 받은 기능적 인터페이스 함수는 I^2C 버스에 연결된 아날로그 장치의 slave 주소, 플래그, 메시지 길이 등을 포함하는 데이터를 생성하여 알고리즘 모듈로 전달한다. 알고리즘 모듈은 버스 상태 및 slave 주소를 확인하고, PCF8584 상태 및 제어 레지스터를 설정한다. 마지막으로 어댑터 모듈은 실제로 맵핑된 영역에 데이터를 쓰고(setbyte()), 읽어오는(getbyte())역할을 한다.

5. 개발 환경 구축

리눅스를 이식하기 위해 이미지가 생성될 호스트 PC에 교차 개발 환경을 구성하였고, 이식의 전단계로 비슷한 스펙을 갖는 테스트 보드를 활용하여 실험 환경을 구축하였으며, 이를 바탕으로 디지털 TV 시스템에 리눅스를 이식하기 위한 개발 환경을 구축하였다.

5.1 교차 개발 환경 구축



(그림 9) 교차 개발 환경 구축 과정

(그림 10)은 교차 개발 환경을 구축하는 과정을 나타낸다. 첫 번째, 교차 컴파일러 및 라이브러리, 커널 모듈 생성 등 전 과정에서 사용될 링커, 어셈블러와 오브젝트 도구(objcopy, objdump)등의 바이너리 도구 모음을 생성한다. 이 도구 모음은 리눅스 이미지, 실행 바이너리 코드, 라이브러리와 같은 목적코드 등의 생성 및 변경에 사용된다.

두 번째, 교차 컴파일러, 교차 라이브러리가 커널 라이브러리를 참조하므로 아키텍처 의존적인 부분을 수정한다. 실제로 MCU가 빅 엔디언 메모리 접근 방식만을 지원하기 때문에 리틀 엔디언 방식으로 작성된 커널 라이브러리를 재구성하였다.

세 번째, 교차 개발 환경 구축 과정에서 가장 중요한 부분이 교차 컴파일러 및 라이브러리 생성이다. 교차 컴파일러와 교차 라이브러리 사이에는 상호 의존적인 관계가 생성된다. 즉, 교차 컴파일러(X-gcc)를 생성하기 위해서는 교차 라이브러리(X-glibc)가 필요하다. 또한 교차 라이브러리를 생성하기 위해서는 교차 컴파일러를 필요로 하는 등의 상호 의존성을 가진다. 이러한 문제를 해결하기 위해 교차 컴파일러 생성과정에서 더미 라이브러리(libchack)를 이용하여, 교차 라이브러리를 참조 하지 않는 교차 컴파일러를 생성한다. 생성된 교차 컴파일러를 이용하여 교차 라이브러리 코드를 컴파일 해 교차 라이브러리를 생성한다. 이렇게 생성된 교차 라이브러리를 포함하는 정상적인 교차 컴파일러를 재생성한다[8].

네 번째, 이식된 리눅스는 읽기만 가능한 롬 파일 시스템을 기반으로 동작하기 때문에 바이너리 형태의 파일 시스템 이미지를 생성하기 위한 도구를 설치해야 한다. 그리고 부가 정보가 들어있어 크기가 큰 ELF(Executable and Linkable format) 포맷 대신 경량인 BFLT(Binary Flat format)를 사용하기 위해 실행 바이너리를 변환하기 위한 도구를 설치한다[9].

5.2 개발 환경

개발 환경은 snds100 테스트용 보드, 원격 디버깅 장비인 EPI사의 JEENI^{TM}(Jtag EmbeddedICE EtherNet Interface), 한국형 디지털 방송 수신이 가능한 프로젝션 방식의 상용 디지털 TV 보드로 이루어졌고, 각각의 구성은 <표 1>과 같다.

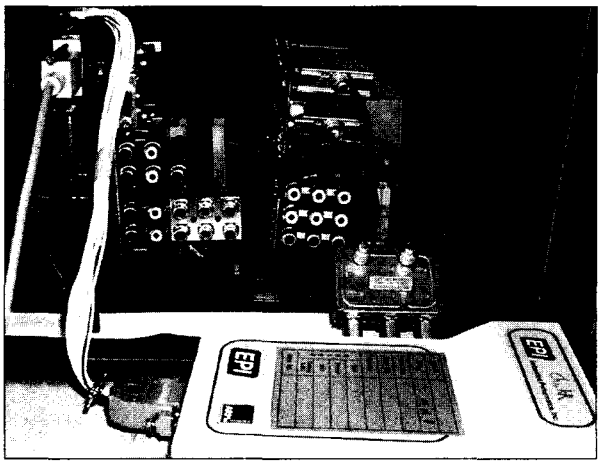
개발 과정은 호스트 PC에서 생성된 리눅스 이미지를 시리얼 라인을 통해 실험 보드내의 메모리에 적재하여 실행하는 방식으로 진행되었다. 그리고 JEENI^{TM}를 이용하여 원격 디버깅 환경을 구성하였다.

개발 환경 구성과정에서 원격 디버깅용으로 사용되었던 JEENI^{TM} 장비의 표준 케이블과 개발 보드내의 포트 핀 배열에 차이가 있어 케이블을 자체 제작하였다(그림 12). 또

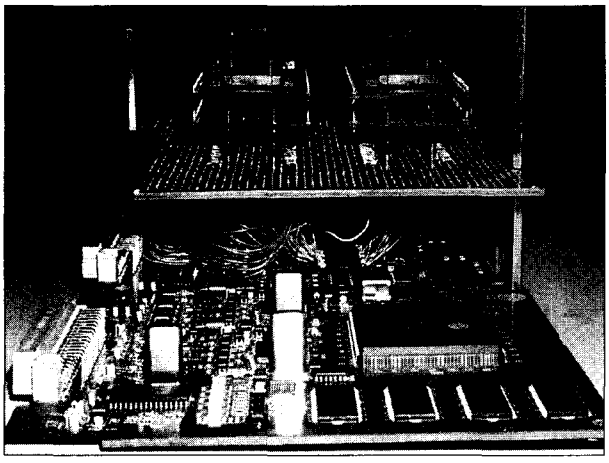
〈표 1〉 개발 환경

	구 성	기 능
snds100 시험 보드	삼성 s3c4510B(ARM 7TDMI), 16MByte SDRAM, 1MByte 플래시 롬, 상태 LED, RJ45 컨넥터.	네트워크 애플리케이션 개발 보드
Digital TV 보드	삼성 s3c4500(ARM 7TDMI), 16MByte SDRAM, 8MByte 플래시 롬, pcf8584, SPU(LG 전자), 32MByte SDRAM.	디지털 TV 시스템 제어 모듈
JEENI	두개의 하드웨어 브레이크 포인트, 이더넷 호스트 인터페이스, 내부 ARM 710A 캐쉬 프로세서.	원격 디버깅을 위한 인터페이스 제공
소프트웨어	gcc-2.95.3, glibc-2.1.3, binutils-2.12, genromfs-0.5.1, elf2flt, gdb-5.0, ADS(ARM™ Developer Suite) 평가 버전 1.1.	교차 개발 환경 구축 및 디버깅을 위한 소프트웨어

한 롬 내부의 부트 블록이 얘기치 않은 일로 지워지는 것을 막기 위해 부트 블록 보호기능이 적용되어 있었고, 로직 형태의 신호로 그 기능을 제거 할 수 없는 관계로 보드에서 롬을 분리하여 소켓에 연결시킨 형태로 수정하였다(그림 13).



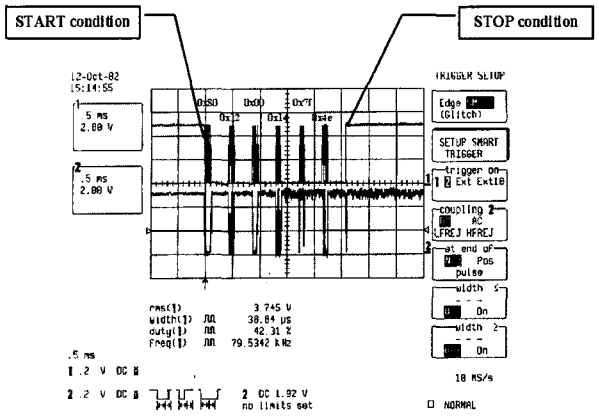
(그림 12) 개발 환경



(그림 13) 개발 보드

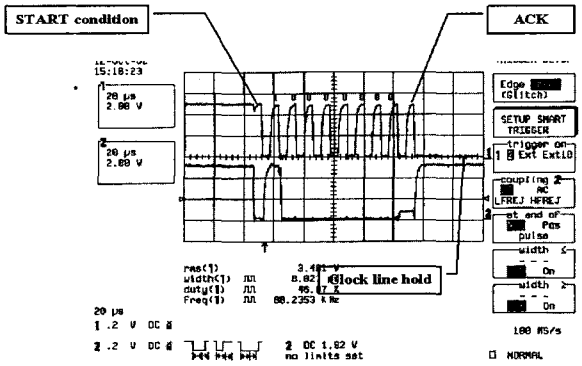
6. 구현 및 실행결과

리눅스가 이식된 시스템의 동작여부 확인을 위해 PCF85 84 인터페이스를 장치를 이용하여 음성 처리기를 제어하였다. 적용된 장치는 MSP3440으로 음성 데이터를 시스템 외부로 출력하기 위해 아날로그 형태로 변환하는 기능을 한다. 결과 확인은 시스템에서 출력되는 비프음과 I²C 버스 신호 측정을 통해 이루어졌다.



(그림 14) 전체 데이터(6byte) 출력

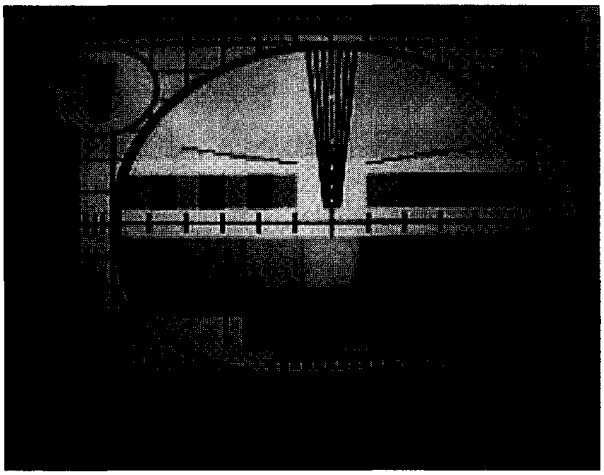
(그림 14)는 특정 주파수의 비프음 한번을 출력하기 위해 보내는 전체 데이터(6Byte)를 출력한 화면이다. 처음 펄스는 MSP3440G의 slave 주소인 0x80, 다음이 sub 주소를 나타낸다. 제어 레지스터를 나타내는 0x12, 비프 기능을 나타내는 0x00 0x14이다. 그리고 볼륨은 0x7f(MAX)이고, 주파수는 0x4E이다[10].



(그림 15) slave주소

MSP3440의 slave 주소는 0x80이다. (그림 15)에서 첫 부분의 Logic High에서 Low로 떨어지는 부분이 START 상태를 나타낸다. 데이터 라인이 high에서 low로 상태가 변한 후, 클록 라인의 상태가 변하였으므로 START 조건을 만족한다. 첫 번째 클록의 데이터 값이 high 그리고 이후의 값들이 모두 low이므로 이 값은 0x80을 나타낸다. 또한 9번째 클록이 ACK를 나타내는데, 이 클록 이후 또 다른 바이트를 보내기 전에 PCF8584에 의해서 클록 라인이 Logic low로 유지된다.

(그림 16)은 리눅스가 적용된 상용 디지털 TV 시스템의 화면 출력 결과이다.



(그림 16) 화면 결과

7. 결 론

최근 디지털 TV시스템의 수요 증가에 따라 여러 가지 상용 모델이 출시되고 있다. 이러한 디지털 TV시스템에서는 여러 가지 작업을 동시에 처리하는 다중 프로그램 환경을 제공해야하므로 운영체제를 필요로 한다.

기존의 디지털 TV 시스템은 고비용의 상용 운영체제를 사용하고 있다. 코드 최적화 및 안정성의 측면에서 장점을 가지고 있지만, 한대의 TV를 생산할 때마다 로열티를 지불해야 하므로 생산 비용이 증가하는 단점이 있다. 이에 반해 오픈 소스 운영체제인 리눅스를 적용할 경우, 로열티 부담이 없으므로 생산 비용을 줄일 수 있고, 리눅스가 지닌 장점을 살려 편리한 개발 환경을 구축 할 수 있다.

본 논문에서 임베디드 리눅스가 적용된 상용 디지털 TV 시스템은 한국형 디지털 방송 수신이 가능한 프로젝션 TV로서 디지털 제어모듈, RF모듈, 영상처리모듈, 아날로그 처리모듈로 구성되어 있다. 그리고 각각의 모듈 중에서 리눅스 이식의 핵심이 되는 부분은 주처리장치, 플래시 롬, SDRAM을 포함하는 디지털 제어 모듈이다.

연구 과정은 먼저 리눅스 이식을 위한 준비 단계로 디지털 제어 모듈의 하드웨어 요소를 고찰 하였고, 이를 바탕으로 리눅스 커널 코드를 시스템에 적합하게 재설계하였으며, 전체 시스템 제어를 위해 I^2C 버스 인터페이스 장치 드라이버를 설계 및 구현하였다. 그리고 교차 개발 환경 구축, 스펙이 비슷한 테스트 환경 구축, 개발 보드 재구성의 과정으로 개발 환경을 구축하여 임베디드 리눅스를 이식하였다. 마지막으로 동작은 I2C 버스 인터페이스 장치를 통해 연결된 아날로그 출력 모듈내의 외부 음성 출력 제어장치의 비프음 확인과 디지털 오실로스코프를 이용한 버스 데이터 측정으로 확인하였다.

참 고 문 헌

[1] "KBS 2001년 기술 연구보고서-데이터 방송", http://tri.kbs.co.kr/pdf/publish/01/6.pdf.

[2] "Digital Video Broadcasting (DVB) : Specification for Service Information (SI) in DVB systems," ETSI EN 300 468 V1.5.1.

[3] "ATSC Digital Television Standard, Rev.B," ATSC Standard A/53B with Amendment 1.

[4] Steve Furber, Stephen B. "ARM system-on-chip architecture," Addison-Wesley, 2000.

[5] "s3c4510B user manual," http://www.samsung.com/Products/Semiconductor/SystemLSI/Networks/PersonalNTASSP/CommunicationProcessor/S3C4510B/um_s3c4510b_rev1.pdf.

[6] "I^2C bus specification", http://www.semiconductors.philips.com/acrobat/literature/9398/39340011.pdf.

[7] "PCF8584 specification," http://www.semiconductors.philips.com/acrobat/datasheets/PCF8584_4.pdf.

[8] Craig Hollabaugh, "Embedded Linux, Hardware, Software, and Interfacing," Addison Wesley, 2002.

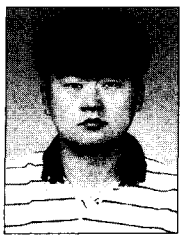
[9] "Embedded Linux/Microcontroller Project," http://www.uclinux.org/description.

[10] "MSP3440G data sheet," http://www.micronas.com/products/documentation/consumer/msp34x0g/downloads/msp34x0g_1ds.pdf.

[11] John L. Hennessy, David A. Patterson, "Computer organization and design : the hardware/software interface," 2nd ed. Morgan Kaufmann Publishers, 1998.

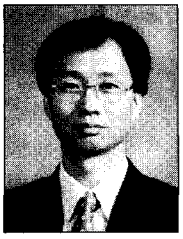
[12] Richard Stones, neil Matthew, "Beginning Linux programming," 2nd ed. Wrox press, 1999.

[13] 권수호, "Linux programming Bible", 글로벌, 2002.



문 상 필

e-mail : octor@palgong.knu.ac.kr
2002년 경북대학교 전자전기공학부 (공학사)
2002년~현재 경북대학교 전자공학과 석사과정
관심분야 : 임베디드 시스템, 운영체제, 디바이스 드라이버



서 대 화

e-mail : dwseo@ee.knu.ac.kr
1981년 경북대학교 전자공학과(학사)
1983년 한국과학기술원 전산학과(석사)
1993년 한국과학기술원 전산학과(박사)
1981년~1995년 한국전자통신연구소 시스템 S/W 연구실 근무
1998년~1999년 University of California Irvine 연구 교수
1995년~현재 경북대학교 공과대학 전자전기공학부 부교수
관심분야 : 임베디드 시스템, 모바일 컴퓨팅, 병렬분산운영체제