9

오픈 소스 소프트웨어 기반 멀티미디어 휴대단말 구현의 실제

국민대학교 ▌신동윤 · 홍대영 · 고원석 · 손주호 · 권동휘 · 임성수*

1. 서 론

오픈소스 소프트웨어를 활용한 휴대단말 구현은 실험적인 수준을 벗어나 상용화 수준에 이르러 있고, 이미 다양한 리눅스 기반의 휴대단말 제품들이 시장에 출시되었다. 리눅스 기반 휴대단말 제품은 기존의 PMP 기능 위주의 고사양 휴대단말에서 최근의 스마트폰 시장에 이르기까지 다양한 제품군으로 개발되고 있으며, 인텔과 퀄컴을 중심으로 MID(Mobile Internet Device) 제품 개발도 리눅스 기반으로 이루어지고 있다.

오픈소스 소프트웨어는 그 특성상 최적화보다는 기능 구현에 초점이 맞추어져 개발이 이루어지고 있기 때문에, 이를 상용 제품에 활용하기 위해서는 제품의 특징과 하드웨어 자원의 한계 등을 고려한 상당 수준의 최적화가 반드시 필요하다. 그럼에도 불구하고 최근까지 개발된 오픈소스 소프트웨어 중에는 기능과 소프트웨어 구조 및 개발 편의성 등에서 상용 소프트웨어 수준을 넘어서는 것들이 상당수 발견되고 있고, 그에 따라 오픈소스 소프트웨어만을 이용한 상용 제품 개발도 점차 가능해지고 있다.

본 논문에서는 오픈소스 소프트웨어인 리눅스를 기반으로 하여 카메라와 오디오 및 DMB 기능을 갖춘 멀티미디어 휴대단말 기기를 개발하는 과정을 실제 구현 사례를 들어 소개한다. 참조 플랫폼의 하드웨어는 Marvell 사의 PXA270 프로세서를 주 프로세서로 사용하고, 멀티미디어 기능을 제공하기 위한 각종 칩을 탑재하고 있으며 소프트웨어는 리눅스 커널 2.6 버전을 중심으로 다양한 오픈소스 소프트웨어를 사용하여 휴대단말을 구성한다.

본 논문의 구성은 다음과 같다. 2장에서는 멀티미디어 휴대단말 하드웨어 구조를 설명하고, 3장에서는 휴대단말 하드웨어의 동작을 지원하기 위한 다양한 커널 구현 기술을 소개한다. 4장에서는 휴대단말의 성능 최적화를 위한 구현 기술 및 GUI와 사용자 응용 소프트웨어를 구성하는 사례를 소개하고 마지막으로 5장에서 결론을 맺는다.

2. 멀티미디어 휴대단말 구조

본 논문의 작성을 위해 사용한 멀티미디어 휴대단말 하드웨어는 DMB를 갖춘 멀티미디어 스마트폰의 구조를 흉내 내어 설계 및 구현된 참조 하드웨어이다. 주 프로세서로 최대 624Mhz로 동작하는 Marvell 사의 PXA270 프로세서를 사용하였고, 카메라와 VGA급 LCD 디스플레이를 갖추고, 넥실리온(주)의 DMB 모듈을 통해 DMB 기능을 제공한다. 저장장치로는 SDRAM(128MB), NOR 플래시메모리(32MB), 그리고 NAND 플래시메모리(256MB)를 탑재하고 있으며, MMC와 SD 카드를 지원한다. 통신 기능으로는 무선랜과 CDMA 1x 혹은 HSDPA 기능을 제공한다. 그림 1은 본 논문의 작성을 위해 사용한 멀티미디어 휴대단말 하드웨어의 연결 구조를 보인다.

단말 하드웨어의 설계에 있어서 가장 중요한 부분 중 하나는 전력 공급을 위한 회로 설계이다. 원활한 전력 공급 및 전력 소모 관리를 위해 대부분의 상용 휴대 단말 제품에는 PMIC(Power Management IC)라 불리는 특별히 설계된 칩을 사용한다.

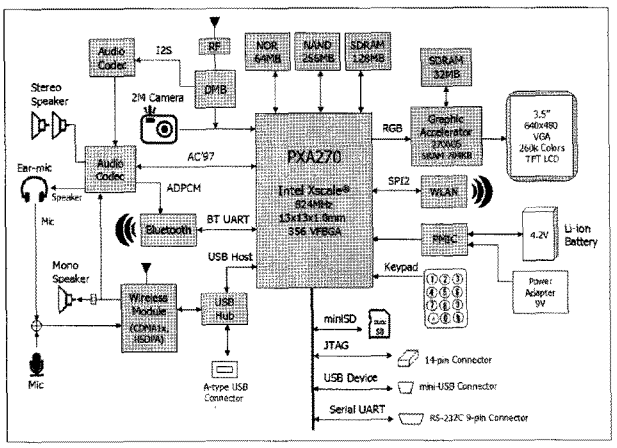


그림 1 시스템 블록도

* 종신회원

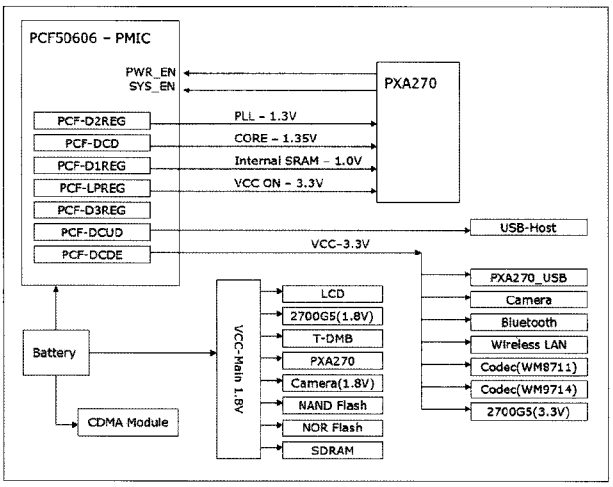


그림 2 PCF50606 전원 배분도

본 논문의 참조 플랫폼에서는 필립스 사의 PCF50606이라는 PMIC를 사용하였다. PCF50606은 시스템의 각 하드웨어들이 필요로 하는 전력을 공급해주는 역할과 시스템의 상태에 따라 전력 공급을 조절하는 전력 관리 기능을 갖추고 있을 뿐 아니라, 터치스크린 인터페이스를 갖추고 있어서 참조 플랫폼에서 터치 스크린을 사용할 수 있게 해준다.

그림 2는 참조 플랫폼에서 PCF50606을 통한 공급 예를 보인다. PCF50606의 전력 공급 부분들중 하나인 PCF-DCD에서 주 프로세서인 PXA270 core부분으로 0.9V에서 1.5v에 해당하는 전압이 공급되고, PCF-DCDE 부분에서 플랫폼의 주변 장치에 공급하기 위한 3.3v의 전압이 나온다.

3. 부트로더 및 리눅스 커널 포팅

커널을 컴파일하고 루트 파일시스템을 구축하기 위해서 컴파일러, 링커 및 기타 개발 도구들을 통합한 형태의 도구 모음인 툴체인을 사용한다. 리눅스 커널 및 관련 소프트웨어는 모두 GNU의 GCC(GNU Compiler Collection)라는 툴체인을 사용하는데 커널 버전 및 루트 파일시스템을 이루는 유틸리티들의 버전에 따라 요구되는 툴체인의 버전이 다르다. 통상적으로 리눅스 커널 2.4 버전 이상에서는 컴파일을 위해 GCC 2.95 이후 버전을 사용한다.

앞서 언급한 하드웨어를 동작시키기 위해서 가장 먼저 실행되는 소프트웨어는 부트로더이다. 부트로더는 시스템 구동을 위해 필요한 기본적인 하드웨어들을 초기화하며 커널을 로드하여 운영체제를 실행할 수 있는 기능을 제공한다. 본 논문에서 사용하는 참조 하드웨어에는 BLOB이라고 하는 오픈소스 부트로더를 수정하여 사용하였다.

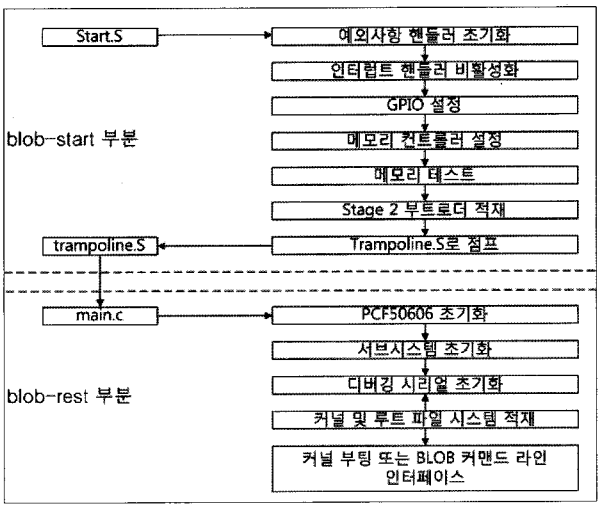


그림 3 부트로더(BLOB) 부팅 절차

본 논문에서는 리눅스 커널 2.6.1x 버전을 참조 하드웨어에서 구동 가능하도록 수정하여 사용하였다. 다음으로 부트로더 및 리눅스 커널을 포팅하기 위해 고려해야 하는 사항들을 소개한다.

3.1 부트로더의 기능 및 구성

본 논문에서 사용하는 부트로더는 BLOB이라고 하는 오픈소스 부트로더의 수정된 형태이다. 그림 3에 BLOB의 실행 절차를 보인다. BLOB의 실행은 크게 blob-start 부분과 blob-rest 부분으로 나누어져 있다. blob-start 부분에서는 시스템의 부팅을 위해 가장 기초적인 설정이 이루어지며 기타 하드웨어 종속적인 부분은 blob-rest 부분에서 이루어진다.

blob-start 부분에서는 ARM 프로세서 구동에 필요한 예외사항 핸들러(exception vector handler)를 초기화하는 작업을 가장 먼저 수행한다. 이 예외사항 핸들러 부분은 리눅스 커널이 부팅되면서 새로운 예외사항 핸들러로 수정된다. 따라서 이 단계의 예외사항 핸들러는 리눅스 커널이 아니라 부트로더가 동작하는데 필요한 핸들러라고 볼 수 있다. 다음 과정에서는 외부 인터럽트 발생을 금지시킨다. 따라서, 인터럽트 핸들러 동작도 역시 비활성화시킨다.

부트로더의 첫 번째 단계 설정 작업 중 가장 중요한 작업은 메모리 설정 작업이다. 주로 메모리 타이밍 관련한 설정 작업이 이루어지는데 DRAM 제어를 위한 타이밍 설정 작업이 가장 중요하며 시스템 성능 및 안정성에 큰 영향을 준다. 보통 DRAM은 동기적 제어 방식을 사용하고 플래시메모리 등은 비동기적 제어 방식을 많이 사용하기 때문에 DRAM 제어를 위한 타이밍 설정 작업이 훨씬 민감하며 프로세서 매뉴얼 및 DRAM 소자의 특성을 잘 파악하여야 한다. 메모리 설정과 아

울러 중요한 작업은 프로세서의 GPIO 설정 작업이다. GPIO는 필요에 따라 범용으로 설정 가능한 입출력 핀이다. 이 입출력 인터페이스에 어떤 장치가 연결되어 있느냐에 따라 그에 적절한 설정이 이루어져야 한다.

부트로더의 두 번째 단계 설정 작업 중 가장 중요한 작업은 PMIC의 설정 부분이다. PMIC는 앞 절에서 언급한 바와 같이 시스템의 각 부분에 전력 배분을 담당하는 칩이다. 따라서, 각 시스템 부분에서 필요로 하는 전력 공급을 위해 적절한 전압 설정을 해야 한다. 만약 이 부분이 정확하게 이루어지지 않으면 해당 장치는 오동작을 할 수 있다.

3.2 리눅스 커널의 설정

본 논문에서 사용하는 참조 단말 하드웨어에는 리눅스 커널 2.6.15를 기반으로 하여 하드웨어 특성에 맞게 수정한 커널을 사용한다. 일반적으로 리눅스 커널이 다양한 프로세서를 지원하고 있지만 각 프로세서에 알맞게 설정된 커널을 만들기 위해서 먼저 해당 프로세서를 위한 패치를 적용해야 한다. 본 참조 단말의 구현 과정에서는 이를 위해 ARM Linux 커뮤니티의 Marvell의 Mainstone(구 인텔의 참조 플랫폼) 참조 보드의 커널 패치를 사용하였다. Mainstone이 PXA270을 채용하고 있기 때문에 프로세서 핵심 부분을 위한 커널 설정은 기존 설정을 그대로 사용할 수 있다.

패치된 커널에 추가적으로 수정해야 할 부분들 중에서 가장 중요한 부분은 참조 플랫폼의 하드웨어 구조에 맞게 매모리 맵을 수정하는 부분이다. 표 1은 메모리 맵을 참조 플랫폼에 맞게 수정한 예이다. 그 밖에 수정해야 할 부분으로는 커널에서 출력하는 메시지를 보기 위해 사용하는 UART 설정과 관련된 부분이 있다.

표 1 메모리 맵 수정 예(arch/arm/mach-pxa/mainstone.c)

```
static struct map_desc mainstone_io_desc[] __initdata = {
        {       // CS0 for DOC
                .virtual        = 0xf0000000,
                .pfn            = __phys_to_pfn(0x00000000),
                .length         = 0x00100000,
                .type           = MT_DEVICE
        },
        {       // CS2 for DMB
                .virtual        = 0xf0200000,
                .pfn            = __phys_to_pfn(0x08000000),
                .length         = 0x00100000,
                .type           = MT_DEVICE
        },
         {      // CS3 for Reserved
                .virtual        = 0xf0300000,
                .pfn            = __phys_to_pfn(0x0C000000),
                .length         = 0x00100000,
                .type           = MT_DEVICE
        },
         {      // CS4 for 2700G (VLIO BUS)
                .virtual        = 0xeb000000,
                .pfn            = __phys_to_pfn(0x10000000),
                .length         = 0x04000000,
                .type           = MT_DEVICE
        },
        {       // CS2 for 2700G (SRAM BUS)
                .virtual        = 0xf0400000,
                .pfn            = __phys_to_pfn(0x14000000),
                .length         = 0x00100000,
                .type           = MT_DEVICE
        },
.....
```

3.3 핵심 디바이스 드라이버 구현

본 절에서는 참조 플랫폼에 탑재된 하드웨어 디바이스들 중에서 중요한 디바이스들에 대한 디바이스 드라이버를 구현한 방법에 대하여 설명한다.

3.3.1 LCD 및 프레임버퍼 디바이스 드라이버 구현

가. 프레임 버퍼 디바이스 드라이버

프레임 버퍼란 LCD를 통해 보여질 영상 정보를 저장하고 있는 메모리 영역을 말한다. 사용자 응용 프로그램은 open/ read/write/close 같은 시스템 콜을 사용하여 프레임 버퍼 디바이스 드라이버에서 제공하는 API에 접근한다.

프레임 버퍼 디바이스 드라이버를 참조 플랫폼에서 사용하기 위해서 LCD 디바이스 드라이버와는 독립적으로 LCD Controller 내의 레지스터 값들을 올바르게 설정해야 한다. 그림 4는 LCD Controller를 위한 레지스터 설정에 관련된 내용을 보여준다. 먼저 프레임 버퍼가 일정 시간 간격으로 LCD에 영상정보를 올바르게 출력하기 위한 픽셀 클럭(pixclock), 해상도(resolution) 설정, 픽셀 당 비트수(bpp)와 같은 레지스터들에 대한 정확한 설정이 필요하다(그림 4). 언급한 레지스터들의 설정이 끝나면 리눅스 커널이 메인 메모리 내에 프레임 버퍼로 사용하기 위한 영역을 할당받으며 이 과정들이 끝마치게 되면 응용프로그램이 /dev/fb0을 통해 접근할 수 있는 프레임 버퍼 장치 노드를 통해 LCD 화면에 출력될 이미지에 대한 내용을 쓰거나 읽어올 수 있다.

참조 플랫폼에서는 이러한 프레임 버퍼뿐만 아니라 추가적으로 2개의 Overlay를 지원한다. 해당 Overlay는 각각 /dev/fb1, /dev/fb2 디바이스 노드에 매핑 되어 있다. 디바이스 드라이버의 초기화 단계에서 프레임 버퍼 구조체의 초기화, 프레임 버퍼를 위한 메모리를 할당, 인터럽트 핸들러 등록, 리눅스 프레임 버퍼에 디바이스 등록의 과정을 수행한다. 응용 프로그램은 mmap( ) 인터페이스를 통하여 프레임 버퍼에 직접 접근할 수 있는 메모리 영역을 얻고, 이 메모리 영역을 통해 픽셀 값을 쓸 경우, 프레임 버퍼 디바이스 드라이버는 LCD 패널에 해당 픽셀을 출력한다.

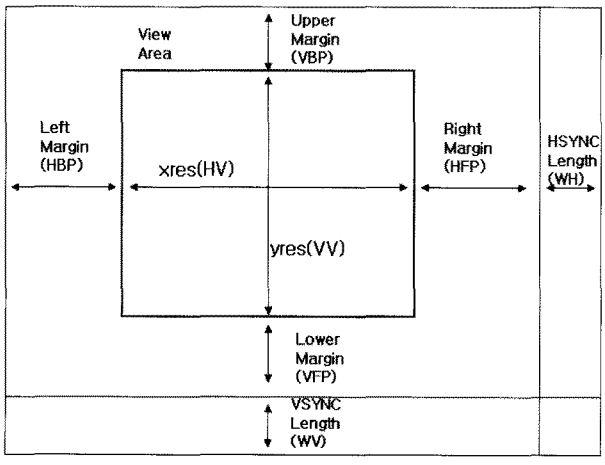


그림 4 LCD Controller 설정

나. LCD 디바이스 드라이버

참조 플랫폼에서 사용하는 LCD 패널은 TPO사의 TD035STEE1이다. 실제 LCD 패널은 아래 보이는 주소에 맵핑되어 있으며 해당 주소에 접근하여 LCD 패널 활성화를 위한 초기화 순서를 수행해야만 정상적으로 프레임버퍼에서 넘어오는 영상 데이터를 뿌릴 수 있다. 이러한 LCD 패널을 초기화를 위해 사용되는 통신채널은 모토롤라에서 개발한 SPI(Serial Peripheral Interface bus)이므로 해당 인터페이스를 사용하기 위한 설정도 미리 커널에서 활성화시켜주어야 한다.

프레임 버퍼에 대한 설정을 정확히 하고 응용프로그램에서 프레임버퍼에 접근할 수 있는 드라이버의 등록을 한 후, 프레임 버퍼에서 있는 영상을 LCD 패널에 뿌리기 위해서는 LCD 패널에 대한 하드웨어를 초기화하는 과정을 수행해야한다. 표 2는 참조 플랫폼의 메모리 맵안의 LCD 관련 메모리 영역에 대한 주소 범위를 보여준다.

LCD를 통하여 올바른 영상정보를 보기 위한 초기화 과정은 다음과 같은 과정을 순차적으로 수행해야 한다. (1) LCD Backlight에 전원을 인가하고, (2) LCD 패널의 초기화를 위한 통신 채널인 SPI 채널 활성화, (3) 마지막으로는 LCD 패널 초기화 과정(deep standy mode 상태, sleep mode off 상태, LCD 패널 레지스터 설정, LCD 패널 활성화)을 수행한다. 3개의 초기화 과정을 수행하면 프레임 버퍼에서 넘어오는 영상 데이터가 정상적으로 LCD 패널에 뿌려진다.

그림 5는 지금까지 언급한 프레임버퍼와 LCD 패널에 대한 설정과 초기화 과정을 거치고 정상적으로 디바이스 드라이버로 등록되었을 때, 응용 프로그램부터 LCD 패널까지 데이터가 이동하는 과정을 보여준다.

표 2 메모리맵 안의 LCD관련 메모리 영역

0x2000 0000 ~ 0x2FFF FFFF	PCMCIA/CF Slot 0 (256MByte)	Reserved	16 Bits wide
0x3000 0000 ~ 0x3FFF FFFF	PCMCIA Socket 1 (256MByte)	Reserved	16 Bits wide
0x4000 0000 ~ 0x43FF FFFF	Memory mapped registers(Peripherals)(64MByte)	Reserved	Reserved
0x4400 0000 ~ 0x47FF FFFF	Memory mapped registers(LCD) (64MByte)	Reserved	Reserved
0x4800 0000 ~ 0x4BFF FFFF	Memory mapped registers(Memory Ctl) (64MByte)	Reserved	Reserved

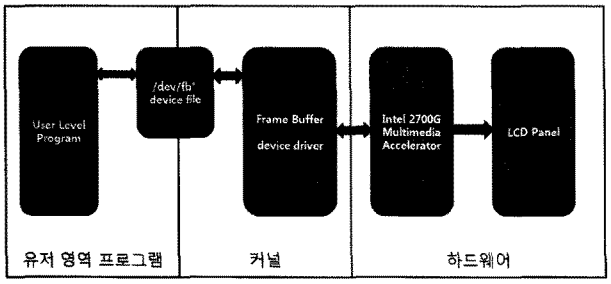


그림 5 프레임버퍼 제어구조

3.3.2 카메라 및 Video4Linux

본 논문에서 사용하는 참조 플랫폼에서는 멀티미디어기능을 제공하기 위해 카메라 모듈과 DMB 모듈을 탑재하고 있다. 카메라 모듈은 PixelPlus사의 PO1200N, DMB 모듈은 넥실리온사의 NX3301이다. NX3301을 통해 T-DMB, TPEG 기능을 이용할 수 있다. 카메라는 I2C 인터페이스를 통해 제어하며, DMB 모듈은 내부에 펌웨어 역할을 하는 마이크로 코드를 통해 구동된다. 두 모듈 모두 PXA270 프로세서의 Quick-capture 인터페이스를 통해 영상 정보를 처리하므로, 카메라와 DMB를 독립적으로 구동시키기 위한 Glue Logic이 하드웨어적으로 구현되어 있다(그림 6).

카메라 디바이스 드라이버는 크게 네 부분으로 구분되어 진다. 먼저 카메라와 PXA270간에 통신을 위해 사용되는 I2C인터페이스에 대한 부분, IC2 인터페이스를 통한 카메라 모듈의 초기화 부분 , PXA270의 QCI(Quick-capture interface)에 관한 초기화 부분, 마지막으로 카메라를 사용하기 위한 V4L의 기능 구현 부분으로 나뉜다.

I2C의 SCL, SDA를 통해 카메라 모듈에 대한 초기화 과정과 PXA270의 QCI의 초기화를 마치고 나면 카메라 모듈을 통해 사용자가 원하는 영상 포맷과 프레임 비율에 해당하는 영상정보들이 들어오게 된다. QCI는

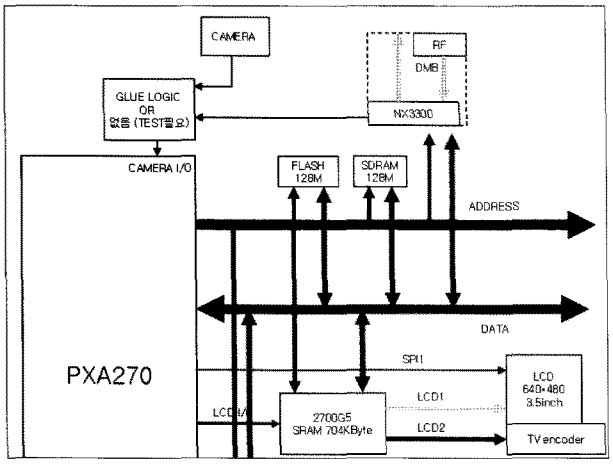


그림 6 카메라, DMB부분의 하드웨어 블록 다이어그램

CMOS 혹은 CCD 형태의 이미지센서를 제어하기 위해 PXA270에 내장된 컨트롤러로서 입력된 영상정보를 처리하기 위해서 3개의 FIFO를 사용하고, 각각의 FIFO 채널에 대해 DMA를 사용한 수신 설정을 하면 DMA 컨트롤러에서 지정한 주소로 데이터를 전송하게 된다. DMB 역시 마이크로 코드의 동작이 수행되면, 영상정보를 QCI를 통해 프로세서로 전송하게 되고, 이 과정은 카메라의 QCI 사용과정과 동일하게 수행된다.

V4L(Video4Linux)는 리눅스에서 비디오 디바이스를 제어하기 위해 표준으로 지정된 스펙이다. 비디오 디바이스는 튜너를 포함한 비디오 카드나 웹캠과 같이 영상정보를 입력할 수 있는 장치를 말하며, 참조 플랫폼에서는 카메라와 DMB의 영상 정보를 사용자레벨에서 제어하기 위하여 사용되므로, 참조 플랫폼에 탑재된 카메라와 DMB의 제어를 위한 V4L 부분을 구현해야 한다. 본 논문의 참조 단말 하드웨어에는 V4L을 이용하여 카메라와 DMB를 제어하도록 구현하였다. 표 3은 참조 플랫폼에서 V4L을 통하여 영상을 제어하기 위해 구현한 ioctl 명령어들이다.

3.3.3 AC97 및 오디오 디바이스 드라이버

본 논문에서 사용하는 참조 단말 보드는 Wolfson사의 WM9714 칩을 오디오 코덱으로 사용하고 있으며 이는 PXA270 프로세서에 내장되어 있는 AC97 컨트롤러와 AC-링크를 통해 연결되어 있다. AC-링크는 디지털 오디오, 모뎀, 마이크 입력, 코덱 레지스터 제어, 상태 정보 등을 전송하기 위한 시리얼 인터페이스이다. 참조 단말 보드에 대한 오디오 디바이스 드라이버는 AC-링크를 통해 오디오 코덱(WM9714)을 제어하며 드라이버의 초기화 부분은 그림 7과 같다. 먼저 오디오 코덱 내의 레지스터들을 초기화하기 위하여 오디오 코덱을 리셋하고 참조 단말 보드의 오디오 데이터 입

표 3 카메라 디바이스 드라이버에 구현된 Video4Linux ioctl 목록

ioctl	역할
VIDIOCGCAP	비디오 장치의 정보 반환
VIDIOCGWIN	capture 영역 정보 반환
VIDIOCSWIN	capture 영역 정보 설정
VIDIOCSPICT	화상 이미지 속성 설정
VIDIOCGPICT	화상 이미지 속성 반환
VIDIOCCAPTURE	capture 제어
VIDIOCGMBUF	mmap() 인터페이스 지원
WCAM_VIDIOCSINFOR	화상 이미지 속성 설정
WCAM_VIDIOCGINFOR	화상 이미지 속성 반환
WCAM_VIDIOCGCIREG	Quick-capture register 반환
WCAM_VIDIOCSCIREG	Quick-capture register 설정

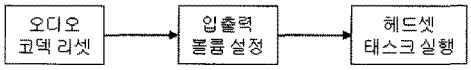


그림 7 오디오 디바이스 드라이버 초기화 절차

출력 각각에 대한 볼륨 크기를 설정한다. 마지막으로 하드웨어 오디오 코덱과 AC97 컨트롤러 사이에 오디오 입출력 데이터가 전달되는 경로를 나타내는 오디오 패스를 설정하기 위하여 헤드셋 태스크를 실행시킨다.

헤드셋 태스크는 참조 단말 보드 헤드셋 슬롯에 헤드셋이 삽입되었는지의 유무에 따라 오디오 입출력 패스를 보드에 내장된 마이크와 스피커 또는 헤드셋의 마이크와 스피커로 설정한다. 헤드셋 태스크는 드라이버 초기화 부분뿐만 아니라 헤드셋 슬롯에 대한 인터럽트가 발생하였을 때도 호출되어 적절한 오디오 패스 변경을 수행한다.

3.3.4 키패드 디바이스 드라이버

참조 단말 보드에는 총 35개의 키가 있으며 7x5 매트릭스 형태로 구성되어 있으며 이는 PXA270 프로세서의 키패드 컨트롤러에 의해 제어된다. 참조 단말 보드를 위한 키패드 디바이스 드라이버는 이러한 키패드 컨트롤러를 제어하도록 구현된다.

키패드와 같은 입력장치에 대한 리눅스 디바이스 드라이버는 역할에 따라 크게 두 가지로 분류되며 이에 대한 구조는 그림 8과 같다. 첫 번째는 응용 프로그램에게 사용자 입력을 전달하고 입력장치를 제어하는 부 드라이버의 등록 처리 방식에 대한 표준적인 절차를 제공하는 주 드라이버이며 두 번째는 입력 장치를 제어하여 사용자로부터 입력받은 데이터를 주 드라이버로 전달하는 부 드라이버이다.

모든 리눅스 시스템은 동일한 키패드 주 드라이버를 사용하며 이는 본 논문에서 사용하는 리눅스 커널 2.6.15에서 지원하고 있기 때문에 개발자가 별도로 작성 및 수정할 필요가 없다. 반면에, 실제적으로 하드웨어 장치로부터 입력 값을 얻어오는 부 드라이버는 해당 플랫폼의 키 구성에 따른 초기화 루틴과 키가 눌렸을 때 주 드라이버로 해당 키코드 값을 전달하는 인터럽트 루틴을 작성하여야 한다. 부 드라이버의 초기화 루틴에서 설정하는 항목으로는 입출력으로 사용되는 GPIO 핀의 모드, 키 디바운스 간격, 매트릭스 키패드의 행과 열의 수, 오토 스캔 유무, 멀티플 키 인식 유무 등이 있다.

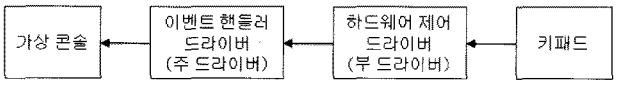


그림 8 키패드 드라이버 연결도

3.3.5 터치스크린 디바이스 드라이버

터치스크린 디바이스 드라이버는 PMIC인 PCF50606에서 제공하는 터치 기능을 사용할 수 있도록 작성되었다. 터치스크린 디바이스의 구조는 상위 어플리케이션과 연동하는 주 디바이스 드라이버와 터치스크린에 값이 입력되었을 경우 처리하는 부 디바이스로 구분된다. 터치스크린에 터치가 감지되었을 경우 인터럽트가 발생하고 부 디바이스 드라이버는 이 인터럽트에 의해 작동한다. 부 디바이스 드라이버는 내부 레지스터로부터 받은 포인팅 좌표 값을 주 디바이스에 전달하는 역할을 한다. 주 디바이스는 상위 응용프로그램과 노드로 연결되어, 부 디바이스로부터 전달받은 정보를 각각의 응용 프로그램에 맞게 전달하는 역할을 수행하므로 응용 프로그램내에서 터치스크린을 터치하였을 경우 직접 값을 전달하는 역할을 하는 것은 주 디바이스 드라이버가 된다.

일반적인 디바이스 드라이버의 경우 인터럽트가 발생하면 인터럽트핸들러가 직접 해당 인터럽트에 대한 처리를 담당한다. 하지만 터치스크린 디바이스의 경우 터치스크린에 한점을 포인팅할 때 한번 발생하는 일반적인 인터럽트 외에 드래그를 할 때 연속적으로 인터럽트가 발생하는 상황을 처리해야 한다.

드래그를 할 때 인터럽트가 중첩되어 연속해서 발생하는 경우, 이전 인터럽트 핸들러의 수행이 끝나지 않았음에도 불구하고, 새롭게 발생한 인터럽트에 의해 이전의 인터럽트 처리가 중단되는 경우가 생기게 되고, 이 때문에 이전 인터럽트 핸들러가 수행해야 하는 좌표값 전달 과정이 방해를 받게 된다.

인터럽트 핸들러에서 상위로 전달해야 하는 좌표값 전달 과정이 방해 받는 것을 방지하기 위해서 세마포어를 사용하여 이전 인터럽트 핸들러의 안전한 수행을 보장하도록 드라이버를 작성하였다.

터치스크린의 인터럽트 핸들러에서는 루프안에서 좌표 값을 읽어 상위 레벨로 전달하는 일을 수행하는 커널 쓰레드를 생성하여, 응용 프로그램에서 좌표값을 계속 읽어들일 수 있도록 한다.

그림 9는 터치스크린 디바이스 드라이버의 동작 구조를 보여준다. 세마포어를 통하여 중복된 인터럽트에

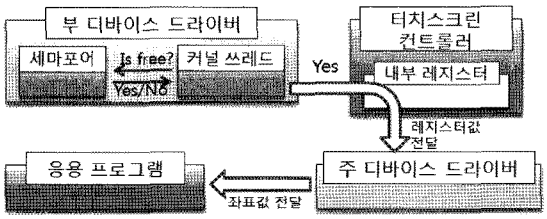


그림 9 터치스크린 디바이스 드라이버의 동작 구조

의한 방해를 받지 않는 커널 쓰레드는 폴링 방식으로 터치스크린의 좌표 값을 읽어 온다. 등록된 터치스크린의 드라이버는 제거되기 전까지 지속적으로 레지스터 값과 세마포어를 확인하고 결과를 주 디바이스 드라이버에 전달하는 역할을 한다. 전달된 값은 주 디바이스 드라이버를 통해 응용 프로그램에서 확인할 수 있다.

4. 성능 최적화를 위한 기술 및 응용 소프트웨어 구조

참조 하드웨어 플랫폼의 에너지소모 및 메모리사용 최적화를 위한 관리 모듈을 구현하였고, 참조 플랫폼의 GUI 시스템을 위한 소프트웨어 셋들을 다양하게 구성하여 여러 가지 사용자 응용 소프트웨어들을 사용할 수 있도록 하였다.

4.1 전력 소모 및 메모리 관리 최적화

4.1.1 전력 소모 관리

전력 관리 프로그램에서는 표 4와 같이 두 가지 방식으로 참조 플랫폼의 전력모드를 제어 한다. Suspend / wake-up 방식의 전력 모드 제어를 위해서 PXA270은 Normal mode, Idle mode, Deep-idle mode, Standby mode, Sleep mode, Deep-sleep mode 등의 6가지 전력 모드를 지원하고, 동적/정적으로 주 프로세서에 공급되는 전압 및 동작 클럭을 제어할 수 있는 DVFS 방식의 전력 제어도 가능하다.

4.1.2 메모리 관리

메모리 극빈(Out-of-Memory, OOM)상태란, 가상 메모리를 사용하는 시스템에서 가용 메모리가 부족하여 더 이상 메모리를 할당받을 수 없는 상태를 말한다. 커널 버전 2.4 이후의 리눅스 커널에는 메모리 극빈 상황이 발생했을 때 메모리를 확보하는 OOM Killer가 구현되어 있다. 하지만 커널 내에 구현된 OOM Killer는 PC 환경을 위해 만들어진 구조이기 때문에 임베디드 시스템 환경에는 적합하지 않다. 참조 플랫폼에서는 커널에 기본적으로 포함되어 있는 OOM Killer의 단점을 보완하기 위하여 사용자가 직접 희생 프로세스를 선택하는 사용자 수준(User-level) OOM Killer 개발하였다.

표 4 참조 플랫폼의 전력 제어 방식

Suspend / Wake-up 기능	Suspend / Wake-up 기능이란 주 프로세서, 그리고 이와 연결된 주변 장치들을 Suspend 모드로 Wake-up을 통한 normal 모드로 전환하는 것을 말한다.
DVFS (Dynamic Voltage Frequency Scaling)	DVFS는 동적으로 주 프로세서의 동작 클럭(frequency)와 전압(voltage)를 조절하기 위해 수행되어야 하는 모든 작업들을 포함하여 가리킨다.

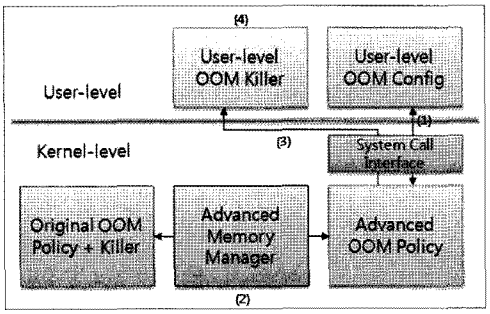


그림 10 메모리 관리 모듈의 동작 구조

그림 10은 임베디드 환경에서 발생할 수 있는 메모리 극빈 현상을 해결하기 위해 구현한 메모리 관리 모듈의 동작 구조와 각 부분에 대한 설명이다.

(1) 사용자 OOM을 사용하기 위한 속성(Policy, Threshold) 설정
(2) 설정된 Policy에 따른 메모리 관리 수행
(3) 메모리 극빈 상황이 발생하면 사용자 수준 OOM Killer가 실행되도록 시그널 전달
(4) 사용자가 희생 프로세스를 선택 후 종료

4.2 GUI 및 응용 소프트웨어 구조

참조 플랫폼에서 사용하는 GUI 시스템은 오픈 소스인 GPE(GPE Palmtop Environment)를 사용한다. GPE는 리눅스 기반의 PDA나 휴대 단말을 위한 어플리케이션을 개발할 수 라이브러리들을 제공할 뿐만 아니라 PIM(Personal Information Management), 동영상/오디오 플레이어, 웹 브라우징을 위한 사용자 프로그램들도 제공한다. X 윈도우 시스템에 기반한 GPE는 내부적으로 GTK+ 사용하며, 윈도우 관리를 위해 Matchbox를 사용한다. 이 외에도 멀티미디어를 위한 GStreamer 라이브러리, 웹 2.0을 위한 Webkit 라이브러리 등과 같이 특정 기능을 위한 라이브러리를 추가로 삽입하여 사용할 수 있다.

그림 11은 참조 플랫폼에서 동작되도록 구성된 GPE 기반 GUI 시스템의 기본 구조를 보여준다. GTK+의 올바를 구동을 위해서 Glib, ATK, Pango 등의 라이브러리 들이 필요하고, 윈도우 관리를 위한 Matchbox 라이브러리들을 구성함으로서 GPE 기반의 GUI 시스템을 구축하였다.

GPE 기반의 GUI 시스템외에 Qt/Embedded기반의 GUI 시스템도 구축하였다. Qt/Embedded는 Trolltech사에서 개발한 임베디드용 오픈 소스 GUI 라이브러리이다. 상용화 제품이므로, 소스의 업데이트 및 관리가 잘되어 있고, 이식성이 강하여 Qt/Embedded 기반의 GUI 시스템에서는 QT 라이브러리만을 사용하여 어떠

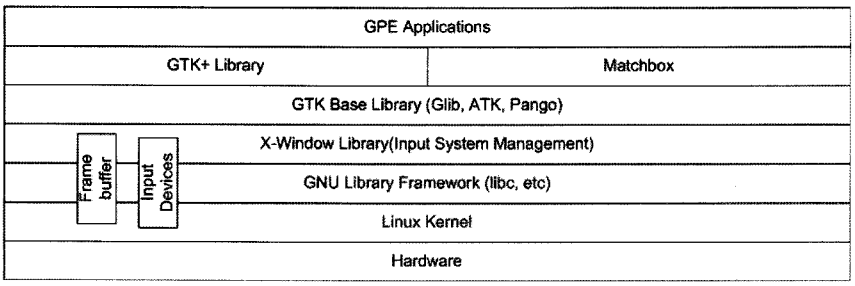


그림 11 GUI와 응용 소프트웨어의 구조

한 플랫폼 위에서도 동작하는 응용프로그램을 개발할 수 있다. 본 참조 플랫폼에서는 Qt/Embedded 기반에서 동작하는 OPIE라는 응용 프로그램 셋을 구축하였다.

5. 결 론

최근 오픈 소스 소프트웨어 기반의 여러 가지 프로젝트들이 상용화 제품에도 적용이 가능한 수준으로 개발, 진행되고 있다. 본 논문에서는 멀티미디어 기능 및 통신 기능을 충분히 지원하도록 만들어진 하드웨어 플랫폼위에 오픈 소스 소프트웨어의 대표라 할 수 있는 리눅스 운영체제를 사용할 수 있도록, 부트로더, 커널의 수정 및 각종 디바이스의 기능 지원을 위한 디바이스 드라이버를 구현한 사례를 소개하였다.

또한, 모바일 단말 환경에서 사용할 수 있는 다양한 응용 프로그램의 구동을 위해서 오픈 소스 기반의 GUI 시스템을 Qt와 GTK 기반으로 구축하는 사례를 보였다.

참고문헌

[1] J. Corbet, A Rubini and G Kroah-Hartman, Linux Device Drivers, 3rd Edition, O'Reilly, 2005

[2] C. Hallinan, Embedded Linux Primer: A Practical Real-World Approach, Prentice Hall PTR, 2006

[3] P. Raghavan, A. Lad, S. Neelakandan, Embedded Linux System Design and Development, AUERBACH, 2005

[4] S. Venkateswaran, Essential Linux Device Drivers, Prentice Hall PTR, 2008

[5] Intel PXA27X Processor Family Developer's Manual January 2006

[6] http://www.gtk.org/

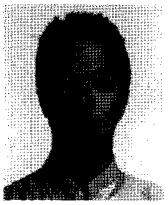
[7] http://www.xfree86.org/

[8] http://gpe.handhelds.org/

[9] http://www.gnome.or.kr/web/default/home

[10] SPECIAL REPORT: Linux kernel 2.6 arrives in embedded, http://linuxdevices.com/news/NS9430710378.html

[11] Migrating to Linux kernel 2.6 -- Part 2: Migrating device drivers to Linux kernel 2.6, http://linux-devices.com/articles/AT4389927951.html

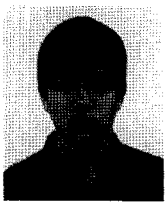


신동윤

2001～2005 국민대학교 컴퓨터과학 학사
2005～2007 국민대학교 전산과학 석사
2007～현재 국민대학교 컴퓨터공학 박사
관심분야 : 임베디드 시스템, 이동통신 단말, 컴퓨터 구조, 실시간 시스템, 시스템 성능 평가, 저전력 시스템
E-mail : naxelsdy@gmail.com

홍대영

2001～2005 국민대학교 컴퓨터과학 학사
2005～2008 국민대학교 전산과학 석사
2008～현재 국민대학교 컴퓨터공학 박사
관심분야 : 임베디드 시스템, 이동통신 단말, 컴퓨터 구조, 실시간 시스템, 시스템 성능 평가, 저전력 시스템
E-mail : 14vicente@gmail.com



고원석

2001～2007 국민대학교 컴퓨터과학 학사
2007～현재 국민대학교 전산과학 석사
관심분야 : 임베디드 시스템, 마이크로 커널, 컴퓨터 구조, 실시간 시스템
E-mail : magicyaba@gmail.com

손주호

1999～2008 국민대학교 컴퓨터공학 학사
2008～현재 국민대학교 전산과학 석사
관심분야 : 임베디드 시스템, 컴퓨터 구조, 시스템 가상화, 실시간 시스템
E-mail : sonjuhomail@gmail.com

권동휘

2000～2007 국민대학교 컴퓨터과학 학사
2007～현재 국민대학교 전산과학 석사
관심분야 : 임베디드 시스템, 이동통신 단말, 컴퓨터 구조, 실시간 저전력 시스템, 가상화 시스템
E-mail : kwondh61@gmail.com

임성수

1993 서울대학교 컴퓨터공학 학사
1995 서울대학교 컴퓨터공학 석사
2002 서울대학교 전기컴퓨터공학 박사
2000～2004 팜팜테크㈜ 기술총괄이사
2004～현재 국민대학교 컴퓨터학부 조교수
관심분야 : 임베디드 시스템, 이동통단말, 컴퓨터 구조, 실시간 시스템신
E-mail : sslim@kookmin.ac.kr