멀티미디어 응용을 위한 Nios II 임베디드 프로세서 시스템의 uClinux 디바이스 드라이버 구현

The Implementation of uClinux Device Driver of Nios II Embedded Processor System for Multimedia Application

김동진*, 박영석*
Dong-Jin Kim*, Young-Seak Park*

요 약

최근 정보통신기기 분야에서 임베디드 시스템이 널리 활용되고 있고, 그 사용범위와 영향력이 점점 커지고 있다. 이러한 임베디드 시스템에서 다양한 기능을 제공하고, 유지 보수를 용이하게 하기 위해서 임베디드 시스템을 위한 운영체제가 많이 개발되어 사용되고 있다. 그 중에서도 임베디드 리눅스는 저렴한 비용으로 구입 가능하고, 많은 장치 드라이버가 제공되며, 소스 코드가 공개되어 있어 이를 수정하면 자신만의 시스템을 구축할 수 있다. 본 연구에서는 멀티미디어 응용에 보편적으로 쓰이고, 가장 활용도가 높은 Touch Panel과 TFT-LCD를 사용하기 위한 디바이스 드라이버를 구현하였다. Altera Nios II 임베디드 시스템을 이용하여 하드웨어를 구성하였고, 멀티미디어 응용을 위한 uClinux기반 Nios II 시스템의 Frame Buffer, Touch Panel, I2S 디바이스 드라이버를 설계하여 실제 동작을 테스트 하였다.

Abstract

Recently, embedded processor systems have been widely used in the field of information communication devices and increased its use range and influence. The embedded systems are offered variety of functions, and its operating systems have been developed to make them easy to repair and maintain. Especially embedded linux is very cheap and provide a lot of equipment drivers. Also we can set up our own system because the source code is opened. In this paper, we describe the implementation of Touch panel and TFT-LCD device driver that are widely used for multimedia application. We designed the system hardware by using Altera Nios II embedded system. And we implemented the device drivers of frame buffer, touch panel and i2s based on uClinux for multimedia application, and tested actual operations of the integrated system.

Keywords : Nios II embedded system, device driver, uClinux, embedded linux

I. 서 론

유비쿼터스(Ubiquitous) 컴퓨팅에 대한 관심과 더불어 하루가 다르게 발전하는 전자·컴퓨터 기술은 시간의 흐름과 함께 각종 컴퓨터 및 전자기기의 고성능화, 소형화를 주도 하면서 임베디드 시스템(Embedded System) 개발을 위한 토대를 마련해 왔다. 이미 전자·정보통신 등의 많은 분야에서 임베디드 시스템이 널리 활용되고 있으며, 최근에는 항공, 우주, 국방, 의료, 멀티미디어 등의 첨단 분야에 이르기까지 그 사용범위와 영향력이 점점 커지고 있다. 이러한 임베디드 시스템에서 다양한 기능을 제공하고, 유지 보수를 용이하게 하기 위하여 임베디드 시스템용 운영체제가 많이 개발되어 사용되고 있다[1].

상용 운영체제인 유닉스 및 윈도우는 구입하는데 비용이 많이 들고, 소스 코드가 공개되지 않는데 반해 리눅스는 저렴한 비용으로 구입 가능하고, 많은 장치 드라이버가 제공되며, 소스 코드도 공개되어 있어 이를 수정하면 자신만의 시스템을 구축할 수 있다[2].

Altera Nios II 임베디드 프로세서 시스템은 사용자가 필요한 주변장치들과 심지어 코어의 기능까지 유연하게 설정하여 만들 수 있어 특정한 응용에 적합하도록 설계할 수 있다. 하지만 시스템의 설계 구현이 어려워 국내·외적으로 설계 구현에 대한 연구가 거의 보고되고 있지 않다.

본 논문에서는 리눅스의 장점과 Altera Nios II 임베디드 프로세서 시스템을 이용하여 멀티미디어 응용에 보편적으로 쓰이고, 가장 활용도가 높은 Touch Panel과 TFT-LCD를 사용하기 위한 디바이스 드라이버를 구현하였다. 각종 응용에 유연하고 적응성이 뛰어난 Altera사의 Nios II 임베디드 프로세서를 이용하여 시스템을 구성하고[3][4][5][6], 멀티미디어 응용을 위한 Nios II 시스템의 Frame Buffer,

*경남대학교 정보통신공학과
투고 일자 : 2009. 7. 15 수정완료일자 : 2009. 10. 27
게재확정일자 : 2009. 10.29
※ 본 연구는 2009년도 경남대학교 학술연구장려금 지원으로 이루어졌음.

Touch Panel Controller, I2S Controller 디바이스 드라이버를 설계하여 실제 동작을 테스트하였다.

Ⅱ장에서는 하드웨어 설계를 논의하고, Ⅲ장에서는 개발 환경 구축, Ⅳ장에서는 리눅스 커널 및 디바이스 드라이버 설계, 그리고 Ⅴ장에서는 실험 및 결과, 마지막으로 Ⅵ장은 결론을 요약한다.

Ⅱ. 하드웨어 설계

하드웨어 설계의 주요 도구로는 Quartus Ⅱ 설계 소프트웨어와 SOPC Builder 시스템 개발도구, Modelsim 시뮬레이션 소프트웨어 그리고 구현회로 검증을 위해서는 SignalTap Ⅱ 임베디드 논리해석기를 이용하였다[5][6]. 그림 1은 본 연구가 목적하는 시스템의 하드웨어 구성이다.

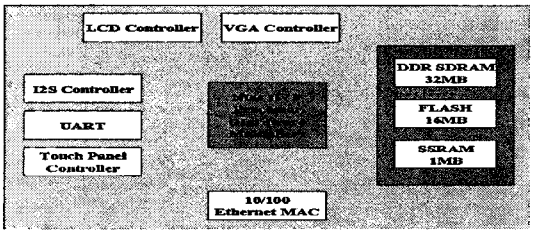

그림 85. 시스템 하드웨어 블록도

Fig. 1. Block diagram of system hardware

본 연구에서는 그림 1의 하드웨어를 바탕으로 멀티미디어 응용을 위한 다음의 디바이스 드라이버를 개발하였다.

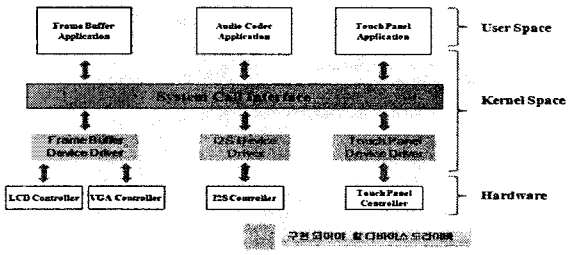

그림 86. 구현할 디바이스 드라이버

Fig. 2. Implementing device drivers

2.1 uClinux를 위한 하드웨어 최소 사항

uClinux를 위한 최소 시스템으로 프로세서의 경우 Nios Ⅱ f 또는 s 코어와 Hardware multiplier가 요구되고, Nios Ⅱ e 코어의 경우 사용 가능하나 처리 속도가 매우 늦기 때문에 권장하지 않는다. 또한 Hardware multiplier의 경우 Nios Ⅱ 프로세서의 옵션으로 포함되지 않을 경우 예상할 수 없는 hack이 발생할 수 있다. SDRAM의 경우 8MB 이상의 용량을 필요로 하며, 한 개의 Full Featured timer, Jtag 또는 Serial uart가 필요하다. 리눅스의 경우 irq0은 auto-detected를 의미하므로 Timer를 제외한 나머지 디바이스는 절대 irq0을 사용할 수 없다[7].

2.2 Nios Ⅱ 프로세서 및 주변장치 구성

Nios Ⅱ 프로세서 시스템을 설계하기 위해서는 목적 시스템을 상술하고, 구성하여 시스템을 생성하는 SOPC Builder 개발도구를 사용한다. 이는 Quartus Ⅱ 설계 소프트웨어 환경하에서 구동되며, SOPC 설계의 전 과정을 수행할 수 있게 한다.

Frame Buffer, Touch Panel 및 I2S 디바이스 드라이버를 구동하기 위한 하드웨어 구성은 그림 3와 같이 CPU(Nios Ⅱ Processor), Avalon tristate Bridge, Ethernet Controller(DP83848C), LCD Controller(TD043MTEA1), SDRAM Controller, SSRAM Controller, Touch panel Controller(AD7843), I2S Controller(WM8731 Audio Codec), Timer 등의 컴포넌트가 필요하다.

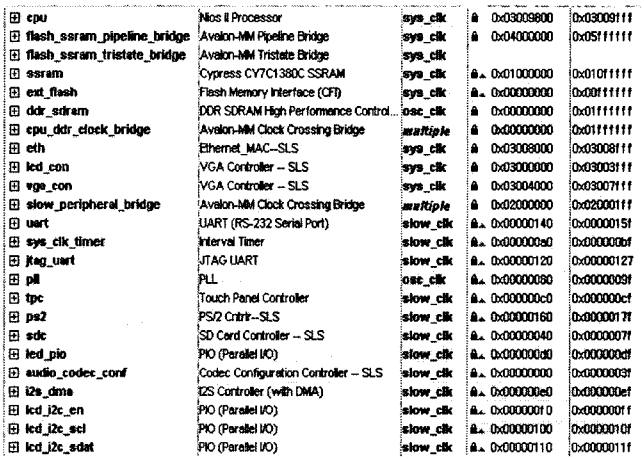

그림 87. SOPC Builder 시스템 구성

Fig. 3. SOPC Builder system configuration

SOPC Builder를 이용하여 시스템을 구성하고 Generate를 실행함으로써 SOPC Builder에서 구성한 시스템이 자동 생성되고, 이러한 하드웨어 정보들은 *.PTF라는 파일에 저장되게 된다. Quartus Ⅱ에서 Compilation을 실행하여 *.sof 파일을 생성함으로써 하드웨어 설계는 완료된다.

그림 4는 Quartus Ⅱ 소프트웨어로 구성한 시스템 회로도이다.

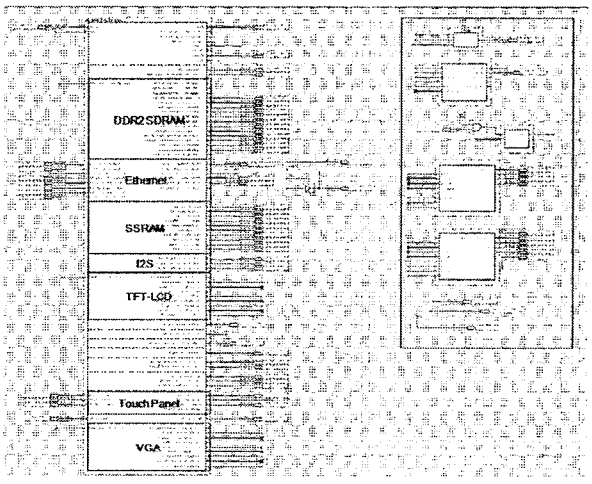

그림 88. 시스템 회로도

Fig. 4. System circuit diagram

III. 개발환경 구축

Nios II 임베디드 시스템의 개발 환경은 그림 5와 같이 Host 시스템과 Target 시스템으로 구성된다. Host 시스템은 Application을 개발하고 Compiling, Linking, Remote debugging 등의 기능을 담당하고 Target 시스템은 Host 시스템에서 개발한 Application을 테스트하는 기능을 담당한다.

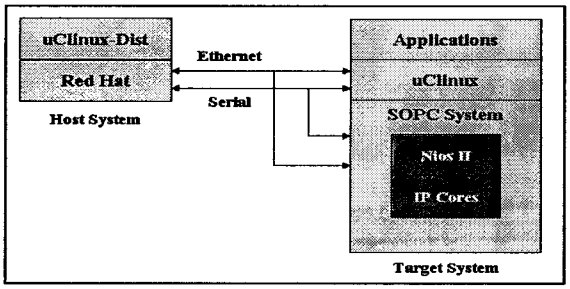

그림 89. 개발환경 구성도

Fig. 5. Development environment organization diagram

3.1 Binary Tool Chain 설치[7][8]

Host 시스템 구축을 위해 Host PC에 Fedora 7.0을 설치한다. Binary Tool Chain을 설치하기 위해 nioswiki[8]나 Altera 협력사로부터 nios2gcc를 다운받는다.

#> sudo tar jxfv nios2gcc-20080203.tar.bz2 -C

#> cd /root

#> vi .bash_profile

"PATH=$PATH:/opt/nios2/bin" 추가

#> source .bash_profile

3.2 uClinux-Dist[7][8][9]

3.2.1. uClinux-Dist 설치

Nios II 임베디드 프로세서는 MMU(Memery Management Unit)가 없기 때문에 일반적인 PC에서 사용하는 리눅스를 사용할 수 없고, MMU를 지원하지 않는 uClinux를 사용한다. 그러나 최근 Altera Nios II 8.0부터 MMU를 지원함으로써 일반 상용 Linux를 사용할 수 있게 되었다.

uClinux-Dist를 설치하기 위해서 uClinux-Dist Source를 nioswiki나 Altera 협력사로부터 다운받고, 다음과 같이 설치를 한다.

#> tar jxvf uClinux.tar.bz2

#> cd uClinux

#> tar jxvf uClinux-Dist.tar.bz2

3.2.2. Kernel Configuring 및 Compiling

커널 컴파일 명령을 통해서 커널 이미지를 생성하기 전에 커널 설정 과정이 필요하다. 커널 설정 방법은 menuconfig를 통해 설정을 한다.

#> make clean

#> make menuconfig

다음과 같은 menuconfig 초기화면이 나타난다.

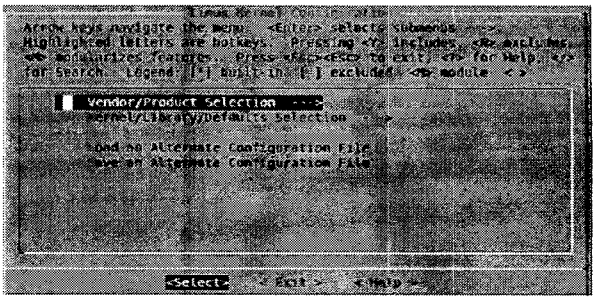

그림 90. Menuconfig 초기화면

Fig. 6. Initial screen of menuconfig

Vendor/Product Selection -->

Vendor(Altera)

Target Product(Nios2) 를 선택한다.

Kernel/Library/Defaults Selection -->

-- Kernel is linux-2.6.x

Libe Version(None)

[ ] Default all settings

[ ] Customize Kernel Settings

[ ] Customize Application/Library Settings

[ ] Update Default Vendor Settings

위와 같이 설정한다. 이때 처음 커널 이미지를 만들기 전까지 다른 커널 옵션들을 변경하지 않는다. 커널 설정을 저장하고 터미널로 빠져 나간다. *.ptf 파일을 이용하여 커널 컴파일 시 사용할 시스템 헤더 파일을 만든다. 이것은 SOPC Builder에서 만든 시스템의 중요한 정보들, 즉 각 컴포넌트에 대한 base address, spans, IRQ 등의 정보를 uClinux-dist/linux-2.6.x/include/asm-nios2/nios2.h 파일에 자동 저장된다.

#> make vendor_hwselect SYSPTF=*.ptf

#> romfs

다시 menuconfig를 통해 커널을 설정한다.

#> make menuconfig

Kernel/Library/Defaults Selection -->

-- Kernel is linux-2.6.x

Libe Version(None)

[*] Default all settings

[ ] Customize Kernel Settings

[ ] Customize Application/Library Settings

[ ] Update Default Vendor Settings

와 같이 설정한 후 커널을 컴파일 한다.

> make

컴파일이 완료되면 /nios2-linux/uClinux-Dist /image 디렉토리에 zImage라는 커널 이미지가 생성된다.

3.2.3. Customize Kernel 및 Application 설정

Frame Buffer, Touch Panel 및 I2S 디바이스 드라이버 테스트를 위한 Customize Kernel 및 Application 설정을 위해

menuconfig를 실행한다.

```
# > make menuconfig
Kernel/Library/Defaults Selection -->
    -- Kernel is linux-2.6.x
       Libe Version(None)
    [ ] Default all settings
    [*] Customize Kernel Settings
    [*] Customize Application/Library Settings
    [ ] Update Default Vendor Settings
```

리눅스 커널 부팅에 대한 address offset은 아래 메뉴에서 설정이 가능하다. zImage load address는 시스템의 sdram base address + link address offet가 된다.

```
Processor type and features -->
*** Platform dependant setup ***
  (0x00500000) Link address offset for booting
```

1) Device Driver 및 File Systems 설정

• Ethernet IP Driver

```
Networking support -->
  Networking options -->
   [*] Packet socket
   [*] UNIX domain sockets
   [*] TCP/IP networking
Device Drivers -->
  [*] Network device support -->
   [*] Ethernet (10 or 100Mbit) -->
    [*] SLS MAC Support
```

• NFS

```
File Systems -->
  [*] Network File Systems -->
   [*] NFS client support
    [*] NFS client support for NFS version 3
```

• VGA IP Driver

```
Device Drivers -->
  Graphics support -->
    [*] SLS VGA IP Support
```

• Touch Panel Controller IP Driver

```
Device Drivers -->
  Character devices -->
    [*] SLS TPC IP Support
```

• I2S IP Driver

```
Device Drivers -->
  Character devices -->
    [*] SLS I2S Support
```

• UART Driver

Consol 디바이스로 JTAG UART와 Serial UART를 동시에 사용할 수 없다.

만약 JTAG UART 사용을 원한다면 다음과 같이 설정한다.

```
Device Drivers -->
  Character devices -->
    [*] Altera JTAG UART Support
    [*] Altera JTAG UART Consol Support
```

만약 Serial UART 사용을 원한다면 SOPC Builder에서 Serial UART 설정에서 Baud Rate : 115200, Parity : None, Data Bits : 8, Stop Bits : 1로 설정하고, 다음과 같이 설정한다.

```
Device Drivers -->
  Character devices -->
    [*] Altera UART Support
     (4) Maximum number of Altera uart ports
      (115200) Default baudrate for Altera UART ports
```

2) Frame Buffer, Touch Panel, I2S 디바이스 드라이버 테스트를 위한 Application 설정

• Touch Panel

```
Miscellaneous Applications -->
    [*] touch panel
```

• JPEG Viewer

```
Miscellaneous Applications -->
  --Video tools
    [*] JPEG Viewer
```

• MP3 Player

```
Miscellaneous Applications -->
  --Audio tools
    [*] MP3Play
```

• NFS

```
BusyBox -->
  Linux System Utilities -->
    [*] Mount
    [*] Support mounting NFS file systems
  Networking Utilities -->
    [*] ifconfig
    [*] Enable status reporting output (+7k)
```

Ⅳ. 커널과 디바이스 드라이버

4.1 리눅스 커널[10]

커널(Kernel)이란 운영체제를 구성하고 있는 핵심(core)이다. 운영체제라고 하는 것은 커널과 디바이스 드라이버, 커맨드 쉘(Command Shell) 혹은 유저인터페이스(User Interface) 그리고 기본적인 파일들과 시스템 유틸리티를 포함한다. 커널은 시스템의 다른 모든 부분을 위한 기본적인 서비스를 제공하고, 하드웨어를 관리하며 시스템 자원을 분배한다. 다시 말하면, 타겟 보드(Target Board)의 램(Ram)에 상주하여, 시스템의 구동에 필요한 환경 설정과 수행되는 프로그램들을 스케줄링(Scheduling)하는 소프트웨어이다.

그림 7과 같이 커널은 프로세서 관리, 메모리 관리, 파일 시스템 관리, 디바이스 관리, 네트워크 관리의 5개의 기능 블록으로 구분할 수 있다.

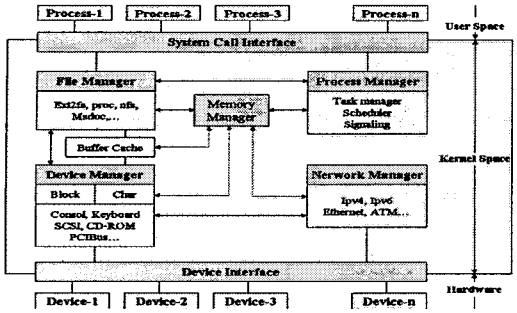

그림 91. 리눅스 커널의 구조

Fig. 7. Structure of linux kernel

4.2 운영체제와 커널[11]

그림 8과 같이 시스템 상에서 실행되는 응용 프로그램은 시스템 콜을 통하여 커널과 통신한다. 일반적으로 응용 프로그램에서 C 라이브러리 같은 함수를 호출하는데, 라이브러리 내부에서는 커널에게 특정 작업을 지시하기 위해 시스템 콜을 사용하게 된다. 일단 커널로 제어가 넘어가게 되면 커널 모드로 실행되어 커널 서브시스템(Sub system) 또는 디바이스 드라이버를 통해서 하드웨어를 제어하게 된다.

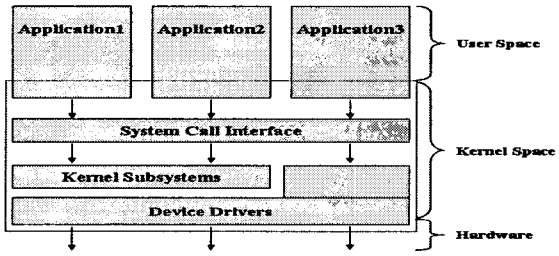

그림 92. 커널과 시스템 구성요소간의 연관 관계

Fig. 8. Relationship between kernel and system component parts

4.3 디바이스 드라이버[11]

디바이스 드라이버는 시스템이 지원하는 하드웨어를 응용 프로그램에서 사용할 수 있도록 커널에서 제공하는 라이브러리이다. 응용프로그램이 커널에게 자원 사용을 요청하고, 커널은 이런 요청에 따라 시스템을 관리한다.

4.3.1. 디바이스 파일

리눅스는 시스템에 있는 모든 자원을 파일 형식으로 표현한다. 이런 파일들은 /dev/ 디렉토리에 존재하며, 이들을 디바이스 파일이라고 한다. 각각의 디바이스 파일이 실질적인 하드웨어를 표현한다. 일반 파일의 목적이 데이터를 저장하는 데 있다면 디바이스 파일은 하드웨어 정보를 제공하는데 목적이 있다.

(1) 디바이스 파일 생성

디바이스 파일은 mknod 유틸리티에 의해서 생성된다. 디바이스 파일은 주로 "/dev/" 디렉토리에 만든다.

> mknod /dev/leds c 240 0

"leds"는 파일명이고, "c"는 문자 디바이스 드라이버, "240"은 주 번호, "0"부 번호이다.

4.3.2. 저수준 파일 입출력 함수

저수준 파일 입출력 함수는 리눅스 커널에서 제공하는 파일 관련 시스템 콜을 라이브러리 함수로 만든 것이다. 이것은 아주 기본적인 파일을 처리하는 함수로서, 디바이스 파일을 실질적으로 다룰 때 사용한다. 저수준 파일을 이용하여 디바이스 파일에 연관된 디바이스 드라이버 함수가 동작하게 된다.

표 1. 저수준 파일 입출력 함수

Table 1. Low level file input and output functions

저수준 파일 입출력 함수	기능
open()	파일이나 장치를 연다.
close()	열린 파일을 닫는다.
read()	파일에서 데이터를 읽어온다.
write()	파일에 데이터를 쓴다.
lseek()	파일의 쓰거나 읽기 위해위치를 변경한다.
ioctl()	read(), write()로 다루지 않는 특수한 제어를 한다.
fsync()	파일에 쓴 데이터를 실제 하드웨어 동기를 맞춘다.

4.3.3. 디바이스 드라이버의 종류

리눅스에서 사용되는 디바이스 드라이버는 크게 문자, 블록, 네트워크 디바이스 드라이버로 나눈다. 문자 디바이스 드라이버는 임의의 길이를 갖는 문자열이나 자료의 순차성을 지닌 장치를 다루는 디바이스 드라이버로서 버퍼 캐쉬를 사용하지 않는다. 블록 디바이스 드라이버는 일정 크기의 버퍼를 통해 데이터를 처리하는 디바이스로, 커널 내부의 파일 시스템에서 관리하고 내부적인 버퍼바이있는 디바이스 드라이버이다. 마지막으로 네트워크 디바이스 드라이버는 네트워크 계층과 연결되어 네트워크 통신을 통해 네트워크 패킷을 송수신할 수 있는 기능을 제공한다.

4.4 TFT-LCD 디바이스 드라이버 구현

Frame buffer란 리눅스 시스템에서 그래픽을 표현할 수 있는 하드웨어를 말한다. 즉, LCD Controller가 frame buffer 장치라고 할 수 있으며, 그 하드웨어를 user application이 제어할 수 있도록 만들어진 디바이스 드라이버를 frame buffer driver라고 할 수 있다. 본 연구에서 사용된 TFT-LCD의 특징은 다음과 같다.

ITEM	Description	Unit
Part Number	TD043MTEA1	-
Display Size	4.3	Inch
Aspect Ratio	15:9	-
Number of Dots	800×RGB×480	Dot
Color Arrangement	Stripe	-
Color Number	16Million	-

그림 93. TFT-LCD의 특징

Fig. 9. Characteristics of TFT-LCD

frame buffer device driver에서 LCD IP(Intellectual Property)에 대한 버퍼 어드레스(buffer address)가 SSRAM(0x05000000 ~ 0x050BB800)에 할당되어 있다. LCD IP는 16 bpp(bit per pixel)와 800×480 해상도로 설정되어 있고, 각각의 픽셀은 2byte의 RGB565로 설정되어야 한다[12].

그림 10은 LCD IP의 레지스터 맵(Register map)이다. LCD IP 내부 레지스터는 LCD Device Driver가 커널에 등록될 때 설정된다.

Register	Address	Description
CTRL	0x00	Control Reg
STAT	0x04	State Reg
HTIM	0x08	Horizontal Timing Reg
VTIM	0x0C	Vertical Timing Reg
HVLN	0x10	Horizontal & Vertical Length Reg
VBAa	0x14	Video Memory Base Address Reg
VBAb	0x18	Video Memory Base Address Reg
CUR_P	0x1C	Cursor Position Reg
CUR_A	0x20	Cursor Address Reg

그림 94. LCD Controller 레지스터 맵

Fig. 10. Register map of LCD controller

그림 11는 TFT-LCD 회로 구성도로써 I2S 통신으로 LCD 모듈을 설정하고, LCD의 동기신호에 의해 영상 데이터가 출력된다.

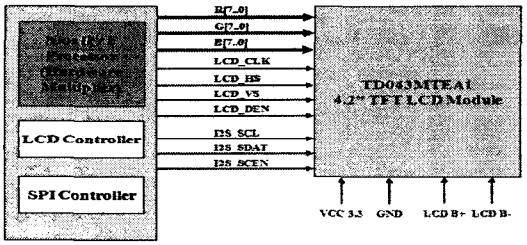

그림 95. TFT-LCD 회로 구성도

Fig. 11. TFT-LCD circuit diagram

그림 12는 본 연구에서 구현한 Frame Buffer 드라이버의 동작 순서도이다. uClinux가 부팅되면 struct platform_device *dev와 같은 형태로 LCD Controller의 플랫폼을 설정한다. platform_device 구조체에는 장치의 이름, ID, 리소스 정보 등이 저장되고 보드가 부팅될 때 platform_device로 정의된 장치들이 가장 먼저 인식된다. platform_device 구조체 안에는 struct resource 구조체가 있다. struct resource 구조체는 LCD Controller의 레지스터가 맵핑된 영역의 물리적 주소를 메모리에 할당한다. LCD 장치의 동작에 관한 정보는 struct fb_info *info 구조체와 같은 형태로 LCD의 동작 타이밍, LCD Controller 레지스터 정보, 가로 및 세로 크기 등에 대한 정보를 저장한다. 프레임버퍼 드라이버 등록은 regster_framebuffer 함수(driver/video/fbmem.c)를 호출함으로써 이루어진다. *regster_framebuffer* 함수는 *fb_info* 구조체를 인자로 받아서 호출된다. fb_info 구조체는 보드 초기화에서 등록한 platform_device 구조체 정보와 드라이버가 직접 설정하는 정보 등이 저장된다. 프레임버퍼 드라이버 사용은 static struct fb_ops slsfb_fbops 구조체를 통해서 application과 연결된다.

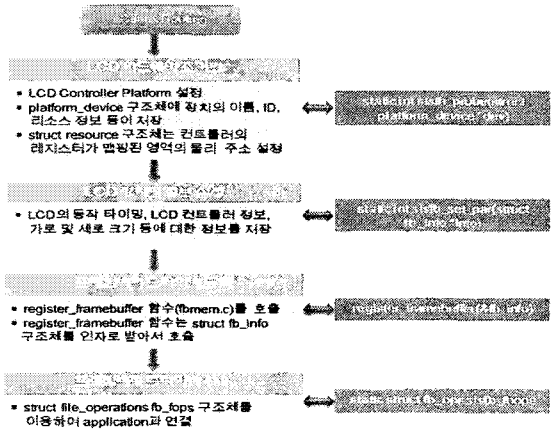

그림 96. Frame Buffer Driver 등록 순서

Fig. 12. Registration sequence of frame buffer driver

그림 13은 Frame buffer 드라이버의 전체 구조이다. user level에서 전송한 frame buffer data는 slsfb_fbops 구조체를 이용하여 "dev/fb0"라는 노드를 통해 frame buffer 드라이버에 전송되며, frame buffer 드라이버는 LCD Controller에게 데이터를 전달하며, 이로써 전송한 데이터가 TFT-LCD에 출력된다.

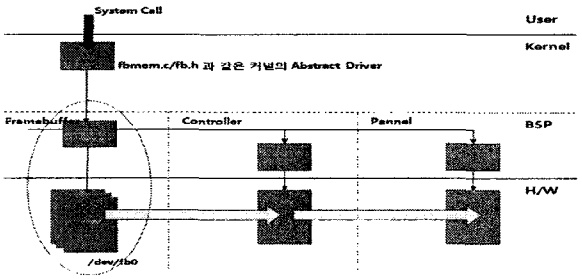

그림 97. Frame Buffer Driver 구조

Fig. 13. Structure of frame buffer driver

4.5 Touch Panel 디바이스 드라이버 구현

최근 멀티미디어 제품들에서 입력장치로 터치스크린(Touch screen)을 많이 사용하고 있다. 임베디드 리눅스는 PC 리눅스와 동일한 시스템을 사용하기 때문에 터치스크린 역시 마우스와 같은 장치로 봐야하고 처리되어야 하지만, 그 입력 방식의 차이점 때문에 다른 관점에서 처리되어야 한다. 특히 디바이스 드라이버 입장에서는 하드웨어 구조가 다르기 때문에 기존 마우스 처리와 다른 형태를 가지고 있다. 본 연구에서 사용되고 있는 Touch Pannel은 다음과 같은 특징을 가지고 있다.

ITEM	Description
Part Number	AD7843
Interface	4-wire touch screen interface

그림 98. Touch Panel의 특징

Fig. 14. Characteristic of touch panel

그림 15는 Touch Panel 회로도로써 사용자 입력을 받아서

2진 데이터 코드로 바꾸어준다.

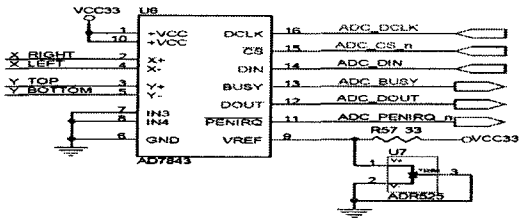

그림 99. Touch Panel 회로도

Fig. 15. Touch panel circuit diagram

그림 16은 Touch panel 디바이스 드라이버 설계시 사용될 Touch panel IP의 레지스터 맵이다. 먼저 컨트롤 레지스터를 세팅하고 XDR과 YDR 레지스터를 통해 터치 좌표 값을 얻을 수 있다.

Realster	Address	Description
CR	0x00	Control Register
XDR	0x04	X Coordinate Data Reg
YDR	0x08	Y Coordinate Data Reg
ISR	0x0C	Interrupt State Reg

그림 100. Touch Panel Controller의 레지스터 맵

Fig. 16. Register map of touch panel controller

그림 17는 본 연구에서 구현한 Touch Panel 디바이스 드라이버의 동작 순서이다. uClinux가 부팅되면 struct platform_device *dev와 같은 형태로 Touch Panel Controller의 플랫폼을 설정한다. platform_device 구조체에는 장치의 이름, ID, 리소스 정보 등이 저장되고 보드가 부팅될 때 platform_device로 정의된 장치들이 가장 먼저 인식되고, 이 구조체 안에는 struct resource 구조체가 있다. struct resource 구조체에는 Touch Panel Controller의 레지스터가 맵핑된 영역의 물리적 주소를 메모리에 할당한다. register_chardev 함수를 호출하여 Touch Panel 디바이스 드라이버를 커널에 등록하고, struct file_operations TPC_fops 구조체를 이용하여 application과 연결한다.

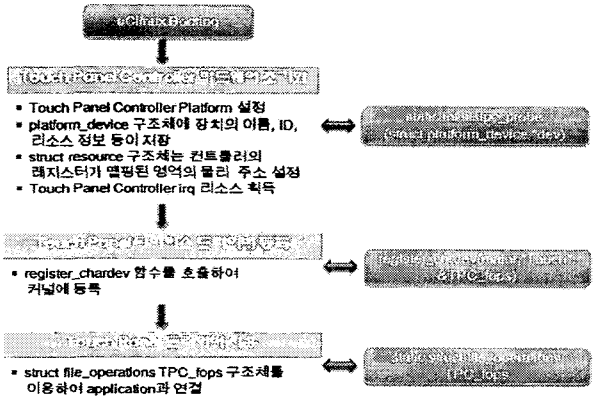

그림 101 Touch Panel 디바이스 드라이버 동작 순서

Fig. 17. Operation sequence of touch panel device driver

그림 18은 Touch Panel 디바이스 드라이버의 전체 구조이다. Touch Controller는 사용자 입력을 받아서 좌표값을 2진 데이터 코드로 바꿔주고, "dev/ts"라는 노드를 통하여 Touch Panel 드라이버에 전송된다. 전송된 좌표값은 TPC_fops 구조체를 통해 user application에 전달된다.

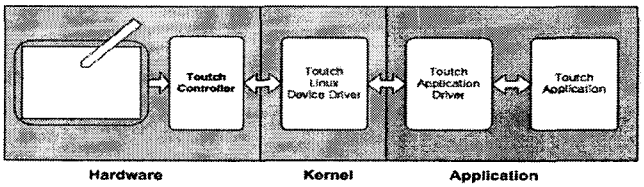

그림 102. Touch Panel 디바이스 드라이버 구조

Fig. 18. Structure of touch panel device driver

4.6 I2S 디바이스 드라이버 구현

I2S는 주로 IC간의 Digital Audio를 전송하기 위한 인터페이스이다. 통상 2채널의 오디오를 전송하기 위해 3라인으로 기본구성이 된다. 기본 클럭인 SCK(Serial Clock), 채널을 의미하는 WS(Word Select), 실제 오디오 데이터 SD(Serial Data)로 구성이 된다.

그림 19은 Audio Codec의 회로도이며, I2S 통신으로 오디오 코덱의 레지스터를 셋팅하게 된다.

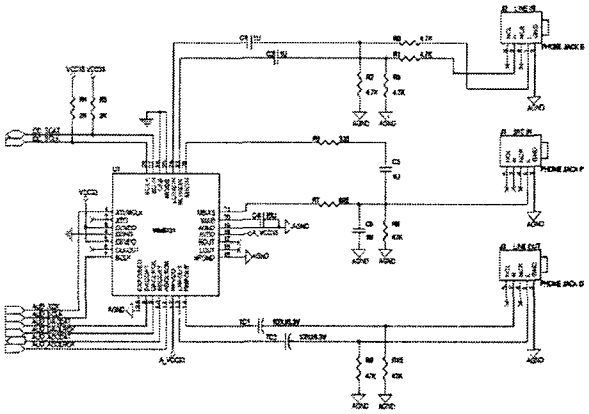

그림 103. WM8731 회로도

Fig. 19. WM8731 circuit diagram

그림 20은 WM8731 Audio Codec의 레지스터 맵이다.

Register	Address	Description
BIT_RATE	0x00	Buffer address
DATA_WIDTH	0x04	Number of words at buffer
NO_OF_WORD	0x08	Bit rate which ip will send data to audio codec
BUFF_ADD	0x0C	Data width of the input data

그림 104. Audio Codec 레지스터 맵

Fig. 20. Register map of audio codec

그림 21는 i2s 본 연구에서 구현한 디바이스 드라이버의 동작 순서이다. uClinux가 부팅되면 struct platform_device *dev

와 같은 형태로 I2S Controller의 플랫폼을 설정한다. platform_device 구조체에는 장치의 이름, ID, 리소스 정보 등이 저장되고 보드가 부팅될 때 platform_device로 정의된 장치들이 가장 먼저 인식되고, 이 구조체 안에는 struct resource 구조체가 있다. struct resource 구조체에는 I2S Controller의 레지스터가 맵핑된 영역의 물리적 주소를 메모리에 할당한다. register_chardev 함수를 호출하여 I2S 디바이스 드라이버를 커널에 등록하고, static struct file_operations sls_i2s_fops 구조체를 이용하여 application과 연결한다.

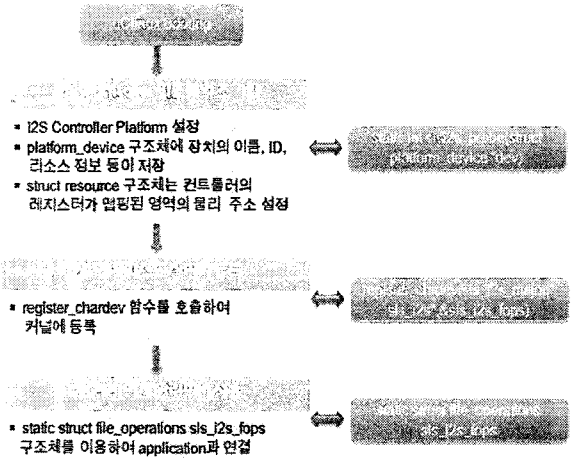

그림 105. I2S 디바이스 드라이버 동작 순서

Fig. 21. Operation sequence of I2S device driver

그림 22는 I2S 디바이스 드라이버의 전체 구조이다. user level에서 전송한 데이터는 sls_i2s_fops 구조체를 이용하여 "/dev/i2s"라는 노드를 통해 I2S 디바이스 드라이버에 전송되며, I2S 디바이스 드라이버는 I2S Controller를 통해 WM8731 codec 칩을 설정한다.

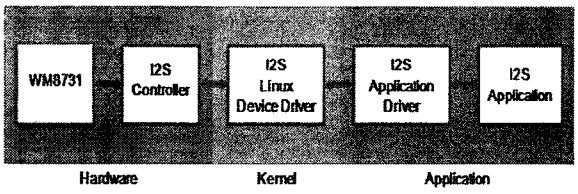

그림 106. I2S 디바이스 드라이버 구조

Fig. 22. Structure of I2S device driver

V. 실험 및 결과

본 논문에서는 멀티미디어 응용을 위한 Nios II 임베디드 프로세서 시스템의 uClinux 디바이스 드라이버를 구현하였다.

멀티미디어 응용에 보편적으로 쓰이고, 가장 활용도가 높은 Touch Panel과 TFT-LCD를 사용하기 위한 디바이스 드라이버를 구현하여 리눅스 커널 이미지를 만들었으며, Nios II 개발 보드에 포팅 하였다. JPEG Viewer와 MP3 Player를 이용하여 실제 동작을 테스트 하였다.

5.1 실험환경

본 논문의 실험에서는 Altera Cyclone III 3C25 디바이스가 장착된 개발보드를 사용하여 동작을 테스트하였다. 하드웨어 개발에는 Altera Design Suite 8.0을 사용하였으며, 소프트웨어 개발은 RED Hat Fedora 7 기반 uClinux Kernel 2.6.27 rc-4를 사용하였다.

표 2 실험 환경

Table 2. Experiment environment

▣ Host PC
▶ Development Board : Altera Nios II Embedded Evaluation Kit, Cyclone III Edition
▶ 하드웨어 개발 : 윈도우 기반 Quartus II 8.0
▶ 소프트웨어 개발 : RED Hat Fedora 7 기반 uClinux Kernel 2.6.27 rc-4
▣ Embedded system
▶ 프로세서 : Nios II / Full featured (150MHz)
▶ Flash memory : 16MB
▶ SDRAM : 32MB
▶ Audio Codec : WM8731
▶ Touch Panel : AD7843
▶ TFT-LCD : 4.3 inch (800×480)

그림 23은 본 논문에서 구현한 디바이스 드라이버를 테스트하기 위한 2만5천 게이트의 Cyclone III FPGA가 장착된 Nios II 개발보드이다. Frame Buffer를 이용하여 JPEG 그림 파일을 LCD에 출력하고 있다.

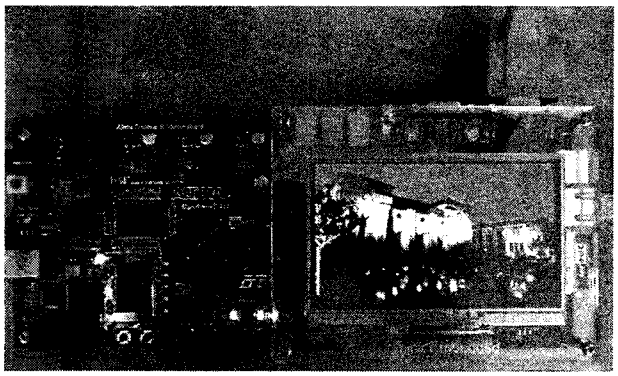

그림 23. 전체 시스템 구성

Fig. 23. Configuration of entire system

5.2 실험 절차 및 결과

5.2.1. uClinux 부팅

2장에서와 같이 Quartus II와 SOPC Builder로 설계한 하드웨어를 FPGA에 실장한다. 3장과 같이 uClinux-Dist 커널 이미지를 컴파일 명령을 통해 생성한 후 Nios II Command Shell에서 nios2-download 명령을 통해 타겟 보드의 SDRAM에 다운로드하면, 하이퍼터미널 창에 uClinux 부팅 메시지가 출력된다.

<Nios II Command Shell 창>

/> nios2-configure media_player_hw.sof

/> nios2-download -g zImage

```
uClinux/Nios II
Built 1 zonelists in Zone order, mobility grouping off.  Total
pages: 8128
Kernel command line:
PID hash table entries: 128 (order: 7, 512 bytes)
Dentry cache hash table entries: 4096 (order: 2, 16384 bytes)
Inode-cache hash table entries: 2048 (order: 1, 8192 bytes)
Memory available: 29960k/2511k RAM, 0k/0k
ROM (1413k kernel code, 1098k data)
TCP bind hash table entries: 1024 (order: 0, 4096 bytes)
TCP: Hash tables configured (established 1024 bind 1024)
TCP reno registered
NET: Registered protocol family 1
io scheduler noop registered
io scheduler deadline registered (default)
FB0: slsfb frame buffer device for SLS VGA
FB0: 800x480x65536 at 0x85000000
Registered SLS Touch Panel Controller(TPC) with 240 Major
Number
 sls_i2s registered
sls_i2s: probe of sls_i2s failed with error 1
ttyS0 at MMIO 0x2000140 (irq = 6) is a Altera
sls_i2s registered
sls_i2s: probe of sls_i2s failed with error 1
ttyS0 at MMIO 0x2000140 (irq = 6) is a Altera
UART
console [ttyS0] enabled
sls_sd_dev_setup :Major number 3 was ok
TCP cubic registered
NET: Registered protocol family 17
Freeing unused kernel memory: 892k freed (0x194000 -
0x272000)
Shell invoked to run file: /etc/rc
Command: hostname uClinux
Command: mount -t proc proc /proc
Command: mount -t sysfs sysfs /sys
Command: mount -t usbfs none /proc/bus/usb
mount: mounting none on /proc/bus/usb failed: No such file
or directory
Command: mkdir /var/tmp
Command: mkdir /var/log
Command: mkdir /var/run
Command: mkdir /var/lock
Command: mkdir /var/empty
Command: ifconfig lo 127.0.0.1
Command: route add -net 127.0.0.0 netmask 255.0.0.0 lo
Command: cat /etc/motd
Welcome to
          ____ _  _
         /  __| ||_|
    _   _| |  | |_  _ __  _   _ _  _
   | | | | |  | | || '_ \| | | |\ \/ /
   | |_| | |__| | || | | | |_| |/    \
   |  ___\____|_|_||_| |_|\____|\_/\_/
   | |
   |_|
For further information check:
http://www.uclinux.org/
Execution Finished, Exiting
Sash command shell (version 1.1.1)
```

그림 24. uClinux 부팅 메시지

Fig. 24. uClinux booting message

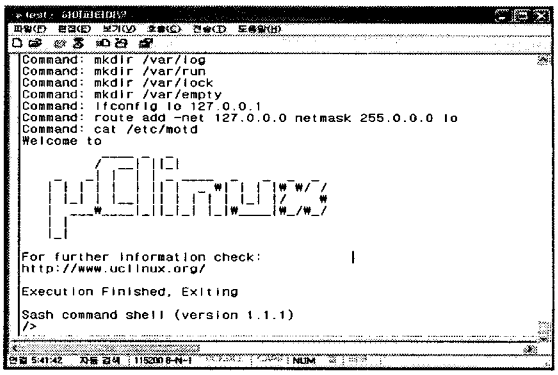

그림 25 uClinux 부팅 화면

Fig. 25. uClinux booting screen

그림 26은 리눅스가 부팅될 때 Target 시스템의 콘솔 창으로서, Frame Buffer, Touch Panel, I2S 디바이스 드라이버가 커널에 등록될 때 출력되는 메시지이다. 본 연구에서 구현한 디바이스 드라이버들은 리눅스로 부팅된 후 모듈로 리눅스 커널에 적재하는 것이 아니라 커널을 컴파일 할 때, 디바이스 드라이버도 같이 포함되어 컴파일 된다.

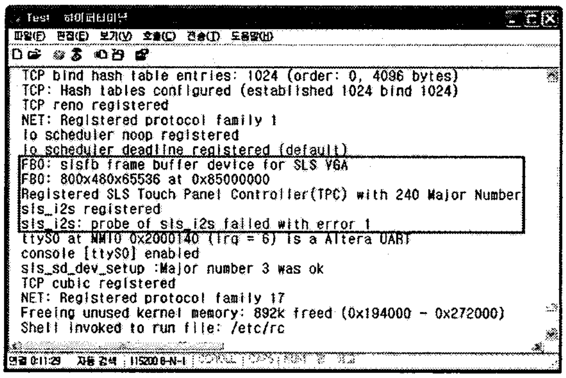

그림 26. Frame Buffer, Touch Panel, I2S 커널 등록 메시지

Fig. 26. Frame buffer, touch panel, I2S kernel registration message

5.2.2. NFS 마운트

JPEG 파일과 MP3 파일을 공유하기 위해 이더넷을 활성화하고, NFS(Network File System)을 마운트하기 위해 다음과 같이 커널을 설정하고, 타겟 시스템의 콘솔 창에서 명령을 실행한다.

```
<커널 설정>
 File Systems -->
   [*] Network File Systems -->
    [*] NFS client support
      [*] NFS client support for NFS version 3
<Tarket system 콘솔 창>
 /> ifconfig eth0 203.253.180.101
 /> mount -t nfs -n -o nolock,rsize=1024,
 wsize=1024 203.253.180.74:/home/nfs /mnt
```

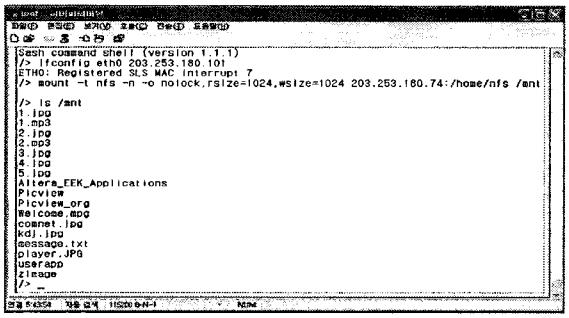

그림 27. NFS 마운트

Fig. 27. NFS mount

5.2.3. Frame Buffer 테스트

Frame Buffer 디바이스 드라이버를 테스트하기 위해 다음과 같이 Kernel을 설정하고, 타겟 시스템의 콘솔 창에서 명령을 실행한다.

<커널 설정>

Miscellaneous Applications -->

--Video tools

[*] JPEG Viewer

<Tarket system 콘솔 창>

/mnt> jpegview -S1 -f 1.jpg

SLS_VGA Driver is opened

read 1.jpg OK

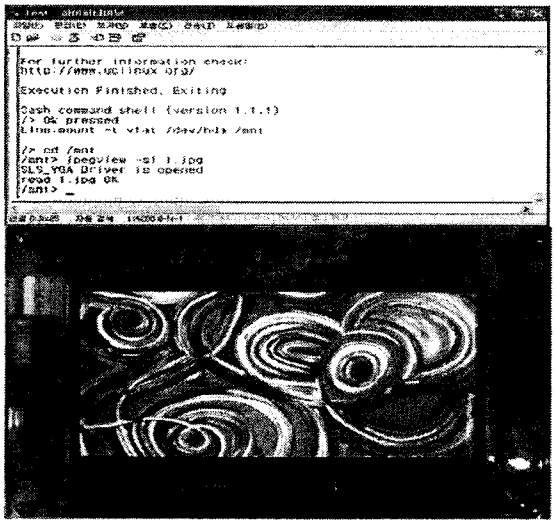

그림 28. Frame Buffer 동작 테스트

Fig. 28. Test operation of frame buffer

5.2.4. Touch Panel 디바이스 드라이버 테스트

Touch Panel 디바이스 드라이버를 테스트하기 위해 다음과 같이 Kernel을 설정하고, 타겟 시스템의 콘솔 창에서 명령을 실행한다.

<커널 설정>

Miscellaneous Applications -->

[*] touch panel

touchpanel application을 실행하고, LCD screen 위를 터치하면 터치된 좌표가 콘솔창에 출력된다.

<Tarket system 콘솔 창>

/> touchpanel &

[25]

/> SLS TPC tesing...

Touch LCD Area ((300,200),(500,280)) to terminate application

X co-ordinate=562,Y co-ordinate=217 ,Pen state =1

X co-ordinate=562,Y co-ordinate=217 ,Pen state =0

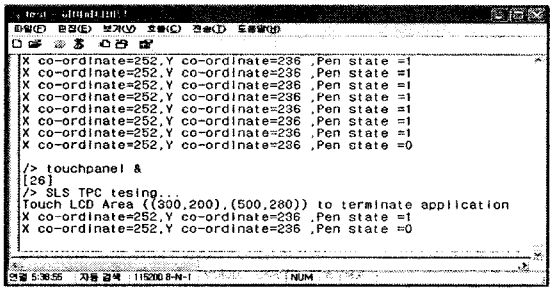

그림 29. Touch panel 디바이스 드라이버 실행 화면

Fig. 29. Execute screen of touch panel device driver

5.2.5. I2S 디바이스 드라이버 테스트

I2S 디바이스 드라이버를 테스트하기 위해 다음과 같이 Kernel을 설정하고, 타겟 시스템의 콘솔 창에서 명령을 실행한다.

<커널 설정>

Miscellaneous Applications -->

--Audio tools

[*] MP3Play

<Tarket system 콘솔 창>

/mnt> mp3play 1.mp3

SLS:mpa_ctrl.stream_buffer_size=2048

1.mp3: MPEG1-III (179766 ms)

Bit rate = 0x20

[melon]005..원더걸스.-.so hot. .

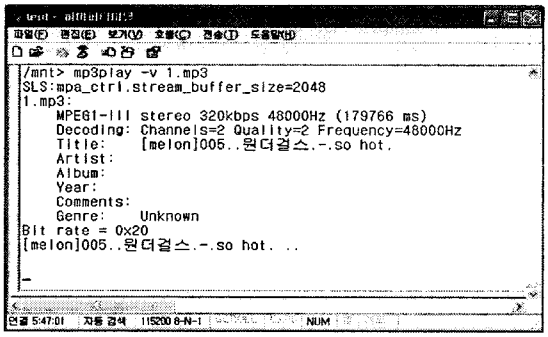

그림 30. MP3Play를 이용한 I2S 디바이스 드라이버 테스트

Fig. 30. Test of I2S device driver using MP3Play

Ⅵ. 결 론

전자·정보통신 등의 많은 분야에서 임베디드 시스템이 널리 활용되고 있으며, 최근에는 항공, 우주, 국방, 의료, 멀티미디어 등의 첨단 분야에 이르기까지 그 사용범위와 영향력이 점점 커지고 있다. 이러한 임베디드 시스템에서 다양한 기능을 제공하고, 유지 보수를 용이하게 하기 위하여 임베디드 시스템용 운영체제가 많이 개발되어 사용되고 있다.

본 연구에서는 초기 구입비와 라이센스(License)의 부담이 없고, 소스 코드의 공개로 전 세계 수많은 개발자에 의해 개발되고 있으며, 많은 장치 드라이버가 제공되는 uClinux를 Nios II 시스템에 포팅(Porting)하였다.

Altera Nios II 임베디드 프로세서 시스템은 각종 응용에 유연하고 적응성이 뛰어나지만, 시스템의 설계 구현이 어려워 국내·외적으로 설계 구현에 대한 연구가 거의 보고되고 있지 않다.

본 논문에서는 리눅스의 장점과 Altera Nios II 임베디드 프로세서 시스템을 이용하여 멀티미디어 응용에 보편적으로 쓰이고, 가장 활용도가 높은 Touch Panel과 TFT-LCD를 사용하기 위해 Touch Panel과 Frame Buffer 디바이스 드라이버를 구현하였으며, Audio Codec 제어를 위한 I2S 디바이스 드라이버도 구현하여 실제 동작을 테스트하였다.

앞으로의 연구 과제로 멀티미디어 컨텐츠(contents)를 저장할 수 있는 SD카드와 동영상 Player를 포팅하여 PMP(Personal Multimedia Player)와 같은 멀티미디어 플레이어를 구현할 필요가 있다.

참고문헌

[1] 신용진, "임베디드 시스템의 미래 '임베디드 리눅스'", 하이-엔드, pp.71-73, 2008

[2] 김명규, 한동원, 황승구, "리눅스를 이용한 임베디드 시스템의 기술 동향", 정보통신연구진흥원 학술정보, pp.1-9, 2000

[3] 박영석, PLD를 이용한 디지털 시스템의 설계, 경남대 지능형홈 인력양성사업팀, pp.1-480, 2005

[4] Altera Corp, Nios II Hardware Development Tutorial, V2.5, pp.1-41, 2007

[5] Altera Corp, Quartus II Development Software Handbook, V8.1, pp.1-2490, 2009

[6] Altera Corp, Embedded Design Handbook, V2.2, pp. 1-306, 2009

[7] SLS Corp, uClinux NEEK BSP User Guide, pp.1-54, 2008

[8] Nios Community Wiki, www.nioswiki.com

[9] Nios Community forum, www.alteraforum.com

[10] 다니엘 보에이, 마르코 체사티, 리눅스 커널의 이해, 한빛미디어, pp.29-846, 2006

[11] 유영창, 리눅스 디바이스 드라이버, 한빛미디어, pp.28-910, 2007

[12] SLS Corp, Graphic LCD Controller User Guide, V1.0, pp.3-26, 2009

[13] Altera Corp, Nios II : World's Most Versatile Embedded Processor, pp.2-11, Oct. 2004

[14] Tyson S. Hall, James O. Hamblen, "Using an FPGA Processor Core and Embedded Linux for Senior Design Projects", IEEE International Conference on MSE, pp.1-2, 2007

[15] David Lariviere, Stephen A. Edwards, "uClinux on the Altera DE2", Colombia University, pp.3-22 2008

[16] Zongqing Lu, Xiong Zhang, Chuiliang Sun, "An Embedded System with uClinux based on FPGA", IEEE Pacific-Asia Workshop on Computational and Industrial Application, pp.691-694, 2008

[17] Philipp Lutz, "Device drivers and Test application for a SOPC solution with Nios II softcore processor and uClinux", University of Applied Sciences, Augsburg, pp.1-56, 2008

김 동 진(Dong-Jin Kim)

2007년 2월 경남대 정보통신공학과(공학사)
2009년 2월 경남대 정보통신공학과(공학석사)
2009년 ~ 현재 경남대 첨단공학과 박사과정

※주관심분야 : FPGA설계, Embedded System

박 영 석(Young-Seak Park)

正 會 員

1979년 영남대학교 전자공학과(공학사)
1981년 한양대학교 전자공학과(공학석사)
1985년 한양대학교 전자공학과(공학박사)

1990년 ~ 1991년 일본 우정성 통신총합연구소(관서선단연구센터) 초빙과학자
1990년 ~ 1991년 일본 긴끼이동통신센터 객원연구원
2001년 ~ 2002년 미국 North Carolina 주립대학(NCSU) 교환교수
1985년 ~ 현재 경남대학교 정보통신공학과 교수

※주관심분야: Software Engineering, Web-based Software Design & Development, Pattern Recognition, Image Processing, Computer Network & Network Computing, Embedded Processor System HW/SW