x86 기반 임베디드 리눅스를 위한 DOC 파일시스템

이병권* 김석일* 전중남*
*충북대학교 전자계산학과
sonic747@just.cbu.ac.kr, {ksi, joongnam}@cbu.ac.kr

Building DOC Filesystem for x86-based Embedded Linux System

ByungKwon Lee*, Sukil Kim*, Joongnam jeon*
*Dept of Computer Science, ChungBuk National University

요 약

x86기반 임베디드 리눅스 시스템의 저장장치로 단일-칩 플래시 디스크인 DOC(DiskOnChip) 시스템이 많이 사용되고 있다. 본 연구에서는 DOC 스스로 부팅하도록 부팅이미지, 커널이미지, 루트파일시스템을 설치하는 과정을 설명한다. DOC는 자체 기능으로 에러탐색 및 수정기능과 파일시스템으로 TrueFFS가 인터페이스로 동작한다. 또한, 구성된 DOC 저장 시스템에 GUI 구현할 수 있도록 Qt-E 계층을 추가하여 시스템 개발자는 단지 어플리케이션을 설치함으로써 쉽게 임베디드시스템을 구성할 수 있다.

1. 서론

최근 임베디드 시스템이 가전제품의 일부분에 포함되어 가전 및 산업제품에 지능화 및 고급화를 가속화하고 있다[1]. 이러한 임베디드 시스템 구성에 안정적인 운영체제의 선택과 각종 컨텐츠 데이터를 효과적으로 저장할 수 있는 저장장치 시스템을 선정하기 위하여 프로세서의 종류, 어플리케이션 및 데이터의 크기, 재기록 여부 등, 여러 가지 사항을 고려해야 한다[2][3][4].

임베디드 소프트웨어 구조를 구현하는데 있어 어려운 문제는 코드가 스스로 수정 가능해야 한다는 것과, 제품을 작동 불능이 되지 않도록 하면서 오류로부터 회복 할 수 있게 만들어야 하고 전원 이상으로 인한 코드 파괴를 피해야 한다[6][7]. 이를 위해 임베디드 시스템에서는 빠른 유지보수를 위해 재부팅 없이 프로그램이 가능한 장치로써 주로 플래시메모리를 사용한다.

본 연구에서는 현재 경제성과 안정성을 가지고 있다는 평가를 받는 운영체제로서 임베디드 리눅스와 x86 기반에서 동작되고 시스템의 컨텐츠를 저장하는 장치로 단일칩 플래시 디스크 저장 장치인 DOC (DiskOnChip)을 선정하여[5] 파일시스템을 구성하는 과정을 설명한다. DOC는 스스로 에러 탐색(EDC)과 수정(ECC)이 가능한 기능을 포함하고 있다. 또한, DOC통한 임베디드 시스템 환경에서의 GUI(Graphic User Interface) 설계를 쉽게 할 수 있도록 임베디드 GUI계층을 포함하는 과정도 함께 설명한다.

2절에서는 DOC 기능과 블록구조의 특징에 대하여 3절에서는 임베디드 GUI 툴킷(ToolKit)인 Qt-E를 설명한다. 4절에서는 리눅스에서 DOC설치를 위해 고려사항과 설치과정을 설명하고 5절에서는 Qt-E를 시스템에 포함하기 위한 환경설정과 설치과정을 6절에서 결론으로 마친다.

2. DOC(DiskOnChip)

DOC은 다양한 용량이 지원되는 고성능, 단일-칩 플래시디스크로서, 빠르고 크기가 작아 적은 공간을 차지하는 저가형의 플래시디스크 솔루션이다. 또한, DOC은 특별한 외부장치나 점퍼를 세팅하지 않고 소켓에 장착하고 전원을 넣는 것만으로 동작하며

EDC/ECC (Error Detection Code/Error Correction Code) 구조를 포함해 에러 탐색 및 수정할 수 있다. 또한, 부트 블록(IPL)을 가지고 있어 스스로 시스템 부팅이 가능하다[그림1].

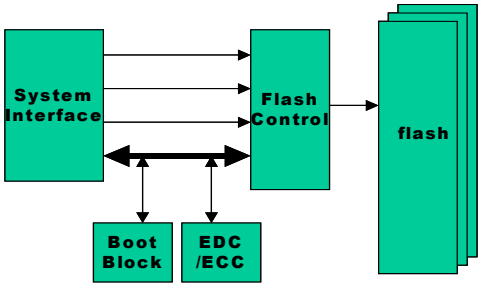

그림 1. DOC(DiskOnChip) 구성도

DOC은 TrueFFS 드라이버를 포함해 wearing leveling을 통해 Flash Array의 동일한 위치에 연속적으로 데이터를 쓰는 것을 방지하여 일반 Flash Component에서 있을 수 있는 부분적인 열화(烈火)를 방지할 수 있는 단일칩 플래시디스크 솔루션이다[5]. 그림2은 시스템 계층에서 위치한 TrueFFS와 DOC이다.

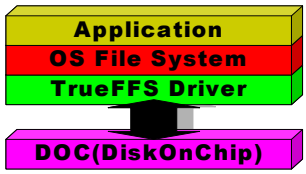

그림 2. TrueFFS 와 DOC

3. Qt 임베디드

Qt는 데스크톱 개발시스템 분야에서 시스템 구축에 급속한 성장을 보인 GUI(graphic user interface) 툴킷 중 하나이다. 현재 리눅스 데스크톱 환경인 KDE(K Desktop Environment)의 개발의 근간이 되기도 했다. Qt는 GNU C++ 기반의 GUI Library로서 MS-Windows, Unix, Linux, Mac, Zaurus, iPaq, Cassiopeia, Generic PDA등을 지원하며 서로 다른 플랫폼에서 소스코드의 호환을 보장한다. Qt의 그래픽 X 환경은 Qt/x11과 Qt/Embedded의 형태의 서로 구분되는 계층구조를 갖는다[그림3].

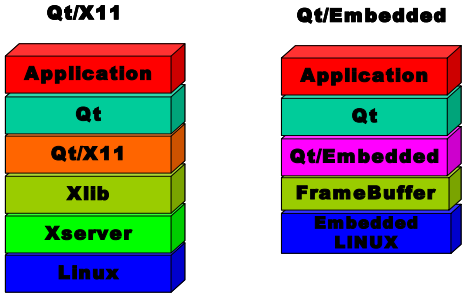

그림 3. Qt 플랫폼 계층구조

현재 일반적으로 사용하는 리눅스는 데스크톱 환경은 XFree86를 사용하고 있다. XFree86은 x86용 UNIX 계열의 OS용으로 개발된 그래픽 환경 전용 소프트웨어이다. 여러 가지 비디오 카드 칩에 대응하고 기능면도 충실하지만 일부의 비디오 칩에는 대응하지 않고 칩에 따라 리눅스용 드라이버의 호완성에 차이가 있다. 또한, 인스톨이나 트러블 슈팅에 어느 정도의 전문 지식이 필요하다. 리눅스의 데스크톱으로 XFree86를 사용하여 임베디드 시스템의 GUI를 구현하기란 트러블 슈팅의 문제는 물론, XFree86의 가용 용량이 너무 커 자원(resource)을 많이 차지하는 문제를 갖는다. 현재 XFree86는 약 20Mbyte 정도의 크기 가지고 있기 때문에 덩치 너무 크다. 이런 단점을 보안하기 위해 QT Embedded 에서는 프레임버퍼(FrameBuffer)라는 계층을 제공하여 XServer, X11을 제거해 성능을 높이고, GUI에 필요한 라이브러리의 저장 공간을 절약하는 효과를 보인다. 프레임버퍼를 이용하여 X-윈도우 없이 커널 수준에서 직접 그래픽 카드의 비디오 램에 접근하는 방식을 사용할 수 있다. 프레임버퍼가 지원하는 해상도는 Gray Scale, 8bit Color, 24bit Color을 제공하고 호스트(host) 환경에서 타겟(target) 환경처럼 시뮬레이션해 볼 수 있는 VFB (Virtual Frame Buffer)제공한다. 임베디드 리눅스 시스템에서 Qt Embedded 라이브러리를 이용해 개발된 프로그램동작을 위한 GUI를 지원할 수 있는 VGA환경인 프레임버퍼에 관한 환경을 설정해야 한다. 설정은 LILO 혹은 GRUB와 같은 Boot Loader에 옵션을 설정하여 인자로 넘겨주는 것이 일반적인 방법이다. 옵션 이름은 보통 VGA인데, 그래픽 카드 제조회사에 따라 옵션 이름이 바뀔 수 있다. 기본적인 VESA2.0 규격을 준수하는 그래픽 카드라면, VGA 옵션으로 다음과 표1 같은 해상도가 주어진다.

표 1. 해상도별 Frame Buffer 설정 값

Color	640x480	800x600	1024x768	1280x1024	1600x1200
256(8bit)	0301	0303	0305	0307	031C
32768	0310	0313	0316	0319	031D
65535(16bit)	0311	0314	0317	031A	031E
16.8M(24bit)	0312	0315	0318	031B	031F

적당한 Color는 16bit Color, 해상도는 800X600 혹은 1024x768을 선택하면 충분하다. 자세한 설정 방법은 표4에서 설명한다.

4. 리눅스용 DOC(DiskOnChip) 설치

DOC가 장착된 임베디드 플랫폼으로 SBC는 AXIOM-TEK(주)에서 개발한 산업용 전용 보드로 기본 베이

스는 x86 이다. 산업 전용 보드이기 때문에 보드에 기본적인 장치만 있을 뿐, 주변장치는 거의 분리된 상태로 동작한다. SBC에는 기존 데스크톱 리눅스를 그대로 사용하여 커널 옵션 및 패치로 환경 설정 후 사용할 수 있다.

4-1 부팅디스크 작성

DOC는 NAND형의 플래시 디바이스로 제조 중에 bad blocks이 포함 될 수 있다. bad blocks를 저장하기 위한 테이블(BBT)이 DOC 안에 파일 형태로 포함되어있다. 현재 리눅스에서는 BBT 테이블의 내용을 읽지 못하므로, DOC 디바이스를 사용하려면 x86용 DOS DOC 툴을 적재한 DOS로 부팅 가능한 디스켓이 필요하다. DOS로 부팅 후 다음 아래 명령어를 사용해 세그먼트 d000에 있는 BBT(Bad Block Table)를 docbbt.txt에 (1)과 같이 복사한다.

```
A:\dformat/win:d000/noformat /log:docbbt.txt (1)
```

이와 같은 명령어로 리눅스에서 DOC 장치를 포맷시 사라지는 DOC의 정보를 파일 형태로 저장 해놓고 다음에 DOC의 정보를 복원하기 위해서이다.

4-2 DOC를 위한 커널 작성

리눅스 플랫폼에서 DOC설치를 위해 제공하는 더미(dummy) 오브젝트 형태의 TreFFS드라이버와 리눅스 커널인 kernel-2.4.2-i386.tar.gz를 획득한 후 특정폴더(/tmp)에 압축해제 한다. 리눅스 커널이 준비되면 DOC을 커널에 추가하기 위해 패치 작업이 필요하다.

▪ 커널 패치

명령(2)로 패치 작업이 끝나면 make xconfig, make menuconfig, make config 세 가지 방법 중 한 가지를 선택하여 DOC 인식을 필요한 옵션을 설정 한다[표2].

```
#>patch_linux linux-2_2-patch driver-patch      (2)
```

▪ 리눅스 커널 DOC 추가 옵션(menuconfig 사용)

표 2. DOC를 위한 커널옵션 선택

```
▸ Loadable module support
 ->[*] Enable loadable module support
 ->[*] Set version information on all symbols for modules
 ->[*] Set version information on all symbols
       for modules
▸ Block devices
 ->[*] Loopback device Support
 -><*> RAM disk support
 ->[*] Initial RAM disk(initrd) support
 ->[M] M-System DOC device support
```

▪ 장치 노드(node) 만들기

DOC 인식을 위해 장치들이 위치한 폴더(/dev)에 명령어(3)으로 노드를 생성 한다. mknod_fl은 미리 작성된 쉘(shell)스크립트로 메이져넘버(majorNumber), 마이너넘버(minorNumber)를 생성하고, 노드 작성시 세심한 작업이 필요하다.

```
#> mknod_fl (/dev/msys/fla, fla[1-4])      (3)
```

▪ 커널 컴파일

DOC가 포함된 커널이미지(bzImage)를 생성하기 위해 명령 make dep->make clean->make bzImage ->make install->make modules->make modules_install 순서로 컴파일 과정을 한다.

▪ DOC 멀티 부팅 설정

DOC가 추가된 커널로 재-부팅 해야 DOC를 포맷과 부팅디스크생성, 루트파일시스템을 생성할 수 있다. 이를 위해 파일(/etc/lilo.conf)에 다음 표3 와 같이 추가한다.

표 3. 부트로더에 DOC 마운트(mount)

```
image=/boot/flash-2.4.0 #DoC 커널이미지
label=DOC(DiskOnChip) # 커널 이름
root=/dev/hda5 #linux root filesystem
```

또한, 파일(lilo.conf)을 수정 후 부트로더에 추가된 커널정보의 적용을 위해 명령(4)를 반드시 수행해야 한다.

```
#>./sbin/lilo                                   (4)
```

4-3 리눅스 DOC 시스템

리눅스를 재부팅 한 후 커널 컴파일 작업에서 생성된 DOC 모듈(doc.o) 추가 명령(5)를 수행한다.

```
#>insmod doc                                    (5)
```

▪ DOC을 위한 부트로더 설정

프레임 버퍼를 사용한다는 것을 커널에 알려야 한다. 파일(/etc/lilo.conf)을 열어서 표4와 같이 수정한다.

표 4. DOC 부팅을 위한 LILO 설정

```
boot=/dev/msys/fla1
compact
install=/boot/doc.b
map=/boot/System.map-2.2.14-5.0
disk=/dev/msys/fla1
bios=0x80
prompt
delay=50
time=50
vga=0x314 //800x600, 65535 Color 16bit
image=/boot/vmlinuz-2.2.14-5.0
label=linux
root=/dev/msys/fla1
initrd=/boot/initrd-2.2.14-5.0.img
```

▪ 커널 이미지 복사 및 루트파일시스템 작성

현재 DOC가 추가된 리눅스로 부팅된 상태이고 부팅 과정에 사용된 리눅스 운영체제관련 이미지를 DOC 장치에 복사(커널이미지, 루트파일시스템)해야 한다. 명령(6)를 사용해 미리 정의한 쉘 스크립트로 명령을 수행한다.

```
#>./mkdocimg redhat-kernl.root.files.copy  (6)
```

▪ DOC를 통한 부팅

DOC를 부팅위한 부트로더 적용명령어(7)를 실행 후 운영체제를 재시동한다. 부팅 환경이 변경되어 좌측 상단에 펭귄 모양의 LOGO를 볼 수 있다. VGA 모드가 맞지 않는다면 적절한 다른 숫자를 넣으라는 화면과 함께 부팅이 멈춘다.

```
#>./sbin/lilo                    (7)
```

5. Qt GUI 설치

Trolltech 사에서 제공하는 임베디드 Qt 툴킷을 다운받아 압축 해제한다. 표5와 같이 설치시 라이브러리 경로와 매크로 정의를 .bash_profile(Readhat 경우)에 작성하면 쉽게 Qt GUI를 작성할 수 있다.

표 5. Qt 환경설정

```
QTDIR=~/qte-2.3.2
LD_LIBRARY_PATH=~/qte-2.3.2/lib:$LD_LIBRARY_PATH
export QTDIR LD_LIBRARY_PATH
```

Qt 임베디드 툴킷을 설치시 타겟 환경에 맞는 해상도, Qt라이브러리, VFB 지원여부 선정하여 Makefile 작성한다. 자세한 환경설정 관련은 파일 INSTALL에서 확인할 수 있다.

또한, Qt로 작성된 프로그램에 대하여 자동으로 Makefile를 작성하게 만들어 줄 수 있는 tmake를 설치한다. 표6에서는 tmake를 bash_profile에 정의한 부분이다.

표 6. tmake 환경설정

```
TMAKEPATH=~/tmake-1.8/lib/qws/linux-x86-g++
PATH=$PATH:~/tmake-1.8/bin
export TMAKEPATH PATH
```

최종적으로 설치명령(make)를 수행해 Qt 툴킷을 설치할 수 있다. DOC 시스템을 바탕으로 한 GUI 어플리케이션을 쉽게 작성하기 위해 Qt-임베디드 계층을 DOC 상위에 구성했다.

DOC 장치 및 드라이버, 리눅 스OS, 프레임버퍼, Qt 임베디드 시스템, Qt 임베디드 어플리케이션의 계층은 그림4와 같다. 본 플랫폼 형태의 구조로 임베디드 시스템 개발자는 자신의 어플리케이션을 상위계층의 Qt 임베디드만을 사용하여 쉽고 빠르게 개발할 수 있다.

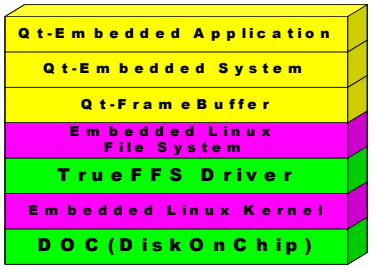

그림 4. DOC and GUI 임베디드

8. 결 론

운영체제를 탑재한 임베디드 시스템은 운영체제의 선정에 안정성과 경제성을 고려한 신중한 선택이 선행되어야할 것이다. 이어 운영체제를 저장할 수 있는 저장장치의 시스템은 운영체제의 에이전트로서 반드시 고려해야 할 중요한 문제일 것이다.

본 연구에서는 단일칩 플래시 타입의 데이터 저장장치인 DOC에 임베디드 리눅스와 루트 파일시스템, 어플리케이션을 구조화하였다. DOC는 TrueFFS란 드라이버를 통해 플래시 저장 장치인 DOC에게 인터페이스 제공하고 또한, EDC/ECC으로 에러를 탐색 및 수정을 자동으로 수행한다. 또한, DOC시스템을 기반으로 한 GUI 툴킷인 Qt를 사용하여 임베디드 어플리케이션 사용자가 쉽게 임베디드 시스템을 개발할 있도록 DOC GUI 시스템을 구성하였다. 향후 연구과제로 실시간 운영체제를 탑재한 리눅스 운영체제를 탑재하여 더 정확한 동작으로 구동될 수 있을 것이다.

참고문헌

[1] 임채덕 외6인, “임베디드 소프트웨어의 기술동향 및 산업발전전망, ”Institute of Information Technology Assessment, 2002.

[2] Embedded System (http://www.embedded.com)

[3] 최병욱 외3인,“임베디드 리눅스,” 홍릉출판사2002

[4] Karim Yaghmour, "Building Embedded LINUX SYSTEMS," O'RELLY, 2003.

[5]M-SYS, "http://www.m-sys.com/content/products/

[6] Gatliff, Bill, "The How-To's of Flash: Implement-ing Downloadable Firmware, "A paper presented at the Embedded Systems Conference, San Jose, 25 September 2000.

[7] Sumner, Scott A. "From PROM to Flash, "Embedded System Programming. July 2000,

[8] Matt Welsh, Matthias Kalle Dalheimer, Terry Dawson, Lar Kaufman, "Running Linux,"ORELLY, 1999.