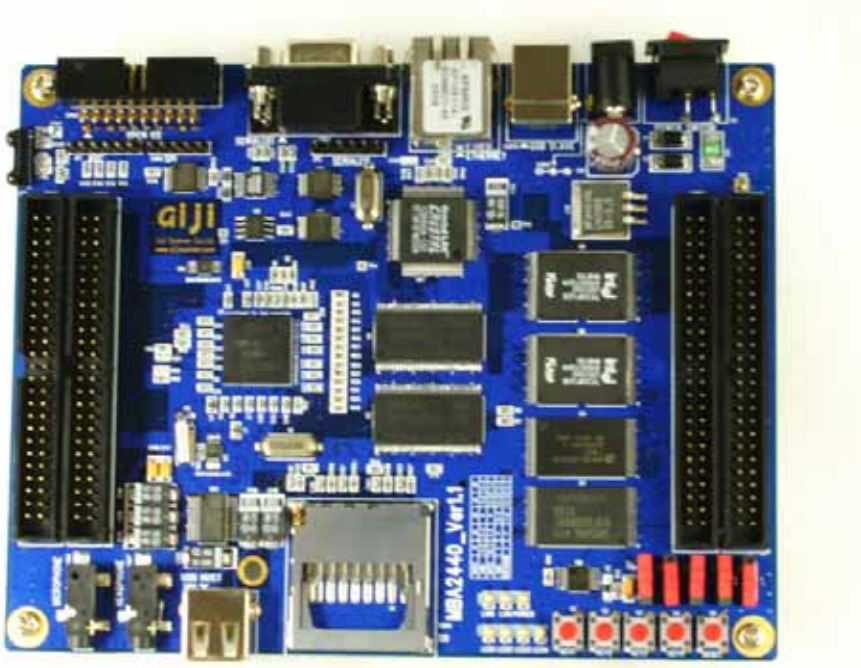
MBA2440 Reference Board Manual



Version 1.0

aiji 아이지시스템(주)
Embedded System Solution Provider

The present document has been developed within Embedded Division of AIJI System and may be further elaborated for the purposes of AIJI System. This document is provided for future development work within AIJI System and membership companies of AIJI System only.

Document Summary

지은이: 아이지시스템 임베디드 사업부
펴낸곳: 아이지시스템

전화: 031-223-6611
팩스: 031-223-6613
openice@aijisystem.com
www.aijisystem.com, www.mculand.com
경기도 수원시 팔달구 인계동 1122-10 삼호 타워 2층

Keywords: MBA2440 Reference Board Manual
- Version : V1.0 (1. Jul. 2005)
Editor Information
- Name: Choong Heon Kim
- E-mail: honeyme@aijisystem.com
- Tel.: +82-31-231-9868
- Address: Samho-Park-Tower Bldg. 2F, 1122-10, Ingye-Dong, Paldal-Gu, Suwon-City, Gyeonggi-Do, Korea
Version Information
- v1.0 :

Copyright Notification

aiji 아이지시스템(주)
Copyright @2005 AIJI System Co., Ltd.

이후로 언급된 회사명 및 제품명은 각 회사의 고유 상표 또는 등록된 상표입니다..
이 문서의 내용은 사전 경고 없이 변경될 수 있습니다.
이 문서의 내용은 아이지시스템의 지적 재산입니다. 아이지시스템의 허락 없이 무단으로 복제, 출판되는 것을 금합니다.

No part may be reproduced except as authorized by written permission.
The copyright and the foregoing restriction extend to reproduction in all media.
© 2005, Embedded Div. AIJI System
All rights reserved.

Contents

1 Introduction

이 문서는 AIJI System의 MBA2440 보드를 위한 매뉴얼입니다.

MBA2440은 삼성의 S3C2440 MCU를 기반으로 하여 여러 가지 하드웨어 디바이스들을 제어하여 mobile system이나 그 외에 embedded system 개발을 위한 reference를 제공하는 것에 초점을 맞추었습니다.

MBA2440은 reference board로서 MBA2440 하드웨어의 정상적인 동작 여부와 널리 쓰이는 OS인 Linux와 WinCE와 국산 RTOS인 iRTOS의 기본적인 동작 여부를 확인하는 것만을 목표로 구현하였습니다.

이 문서에서 설명되어 있지 않은 부분은 테스트 하지 않은 사항이므로 개발자 본인이 직접 테스트 해야 함을 숙지하시기 바랍니다.

이 외에 **다양한 어플리케이션 구현과 그에 따른 하드웨어의 설정 및 변경은 이 MBA2440 보드를 사용하는 개발자의 역량에 따라 언제든지 가능하며, 그에 따른 개발 과정에 대해서도 개발자가 직접 담당해야 할 부분임을 숙지!!**하기 바랍니다.

이 문서는 초보자들을 대상으로 하여 다음과 같은 사항을 목적으로 작성하였습니다.

☞ 보드를 처음 접하여 기본적으로 보드를 동작시킬 수 있는 방법들을 쉽게 따라 할 수 있도록 설명
☞ 보드를 위한 firmware와 OS를 개발 하기 위한 개발 환경 구축 방법 설명
☞ 개발 환경을 이용하여 프로그램을 컴파일 하는 방법을 설명

문제점 해결이나 궁금한 사항을 해결하려면 이 보드 매뉴얼과 CD로 제공된 문서들을 자세히 읽어 보기를 권합니다.

MBA2440에 관련된 소스나 자료들은 (www.mculand.com) 웹사이트를 통해 지속적으로 업데이트 할 예정입니다. 많은 이용을 바랍니다.

저희가 제공한 것 이외에 대한 어플리케이션 구현과 그에 따른 하드웨어의 설정 및 변경에 대한 문의나 기술 지원은 상황에 따라 솔루션 개발에 해당 되는 사항임을 알려 드립니다.

1.1 System Overview

MBA2440의 핵심은 당연히 MCU인 S3C2440이다. 이 MCU에 대한 자세한 설명은 S3C2440 datasheet를 참조하기 바란다.

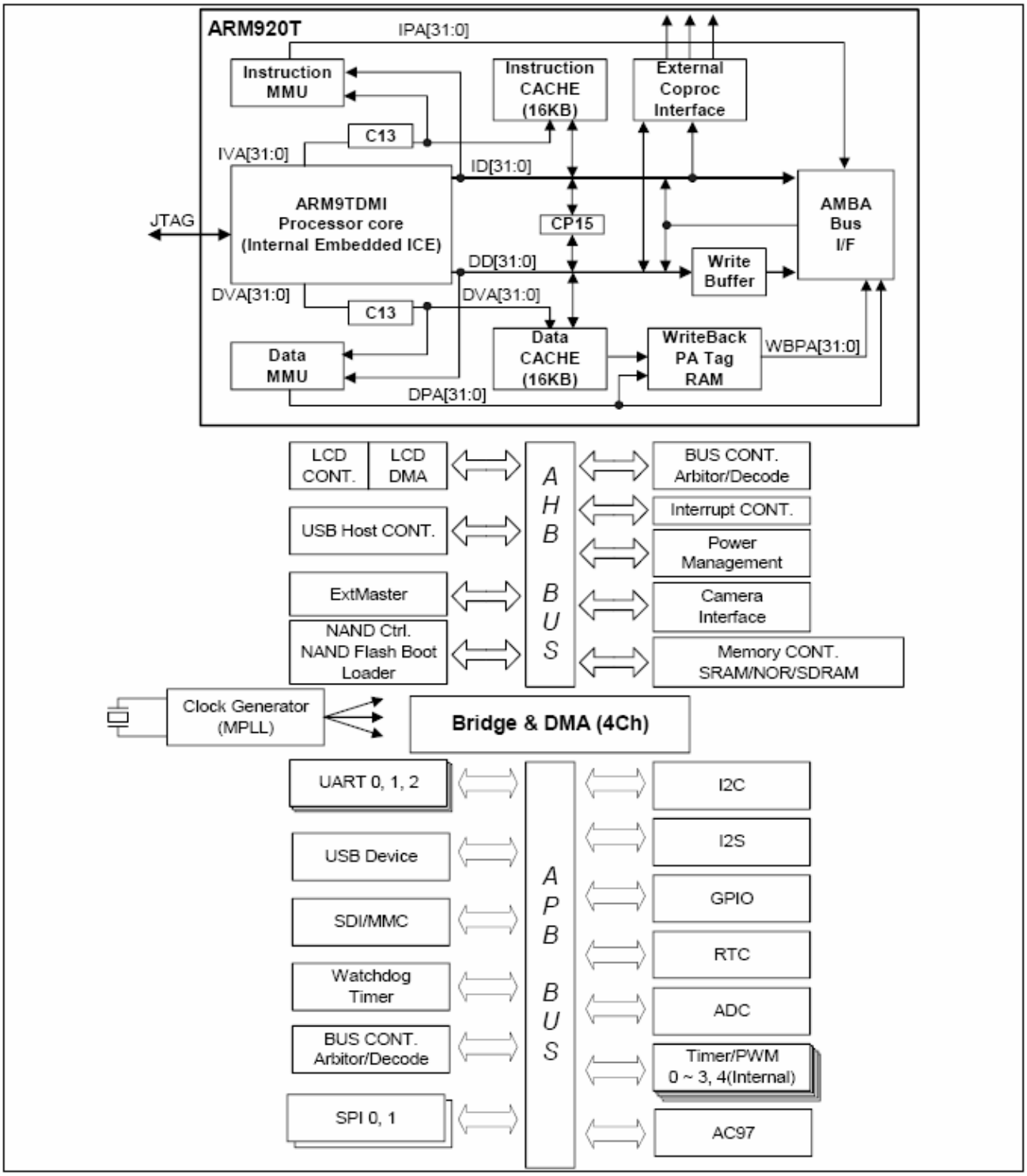
이 장에서는 S3C2440의 기본적인 설명과 함께 architecture의 구조 및 특성에 대해서 간단하게 언급하고자 한다.

S3C2440은 ARM920T core 기반으로 삼성에서 개발한 16/32bit RISC 마이크로 컨트롤러로서 저전력 소모와 높은 performance를 필요로 하는 어플리케이션과 hand-held 디바이스에 적합하다. 또 한, 이러한 시스템을 개발함에 있어서 소요되는 비용 절감을 위해 S3C2440 칩 내부에 여러 가지 다양한 IP들(LCD controller, NAND controller USB controller등등..)이 포함되어 있으며 이 IP들간의 연동을 위해 AMBA(Advanced Micro controller Bus Architecture) bus가 구현 되어 있다.

S3C2440의 특징을 요약하면 다음과 같다.

- CPU clock 300MHz(1.20V for core), 400MHz(1.30V for core)
- 메모리 전원 1.8V/2.5V/3.3V, 3.3V의 외부 I/O
- 16KB의 I/D cache와 MMU
- 외부 메모리 컨트롤러(SDRAM 제어 및 chip select logic)
- 1개의 LCD 전용 DMA 채널이 있는 LCD 컨트롤러(4K color STN, 256K color TFT)
- 4개의 외부 DMA 채널
- 3개의 UART 채널과 2개의 SPI 채널
- IIC/IIS(AC97 I/F) 지원
- SD host version 1.0/MMC protocol version 2.11
- 2개의 USB host 채널/ 1개의 USB device 채널(version 1.1) (USB isochronous 지원 안함)
- 4개의 PWM timer 채널/ 1개의 내부 timer
- WDT(Watch Dog Timer)
- 130개의 multi-functional GPIO/ 24개의 외부 인터럽트 채널
- 8개의 10bit ADC/ Touch screen I/F
- RTC(Real Time Clock)

[그림 1.1]은 S3C2440의 block diagram이다.

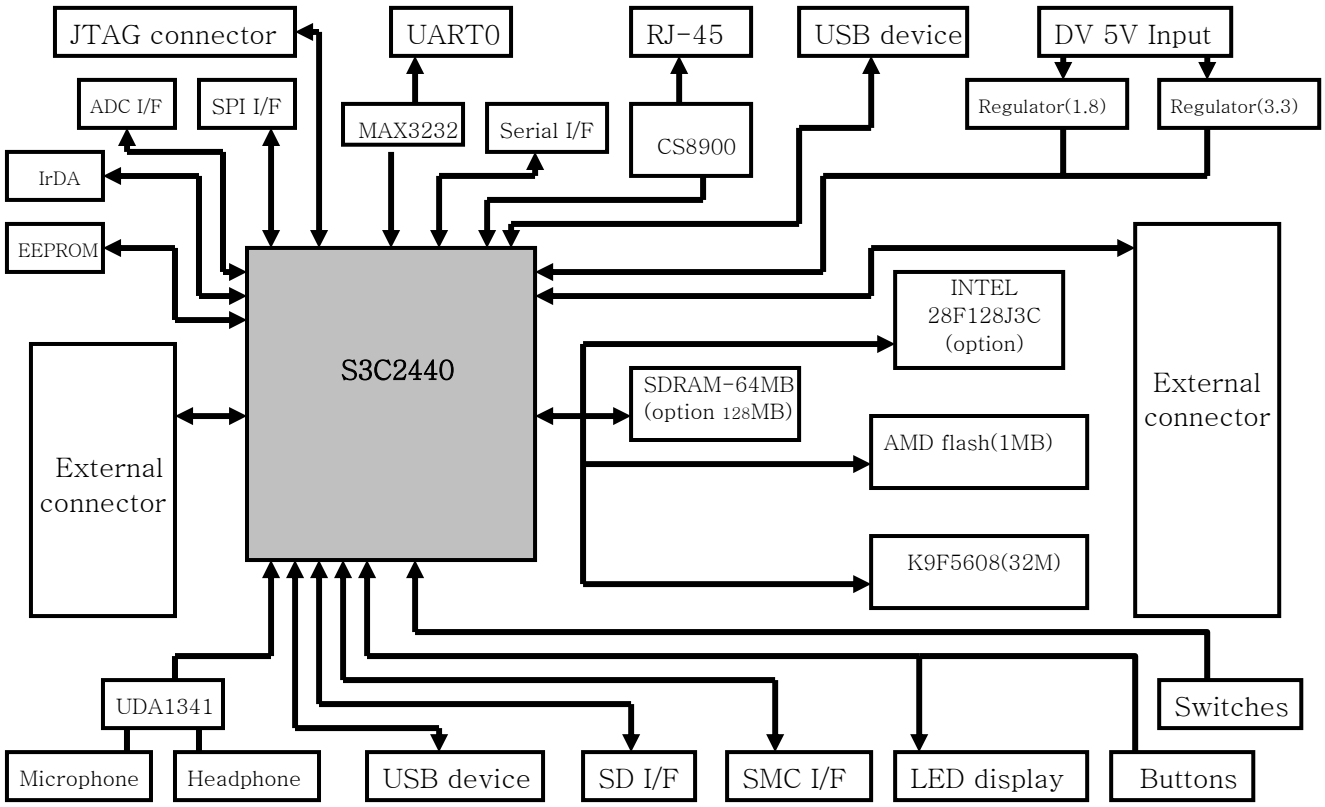


[그림 1.1] S3C2440 Block Diagram

1.2 MBA2440 Specification

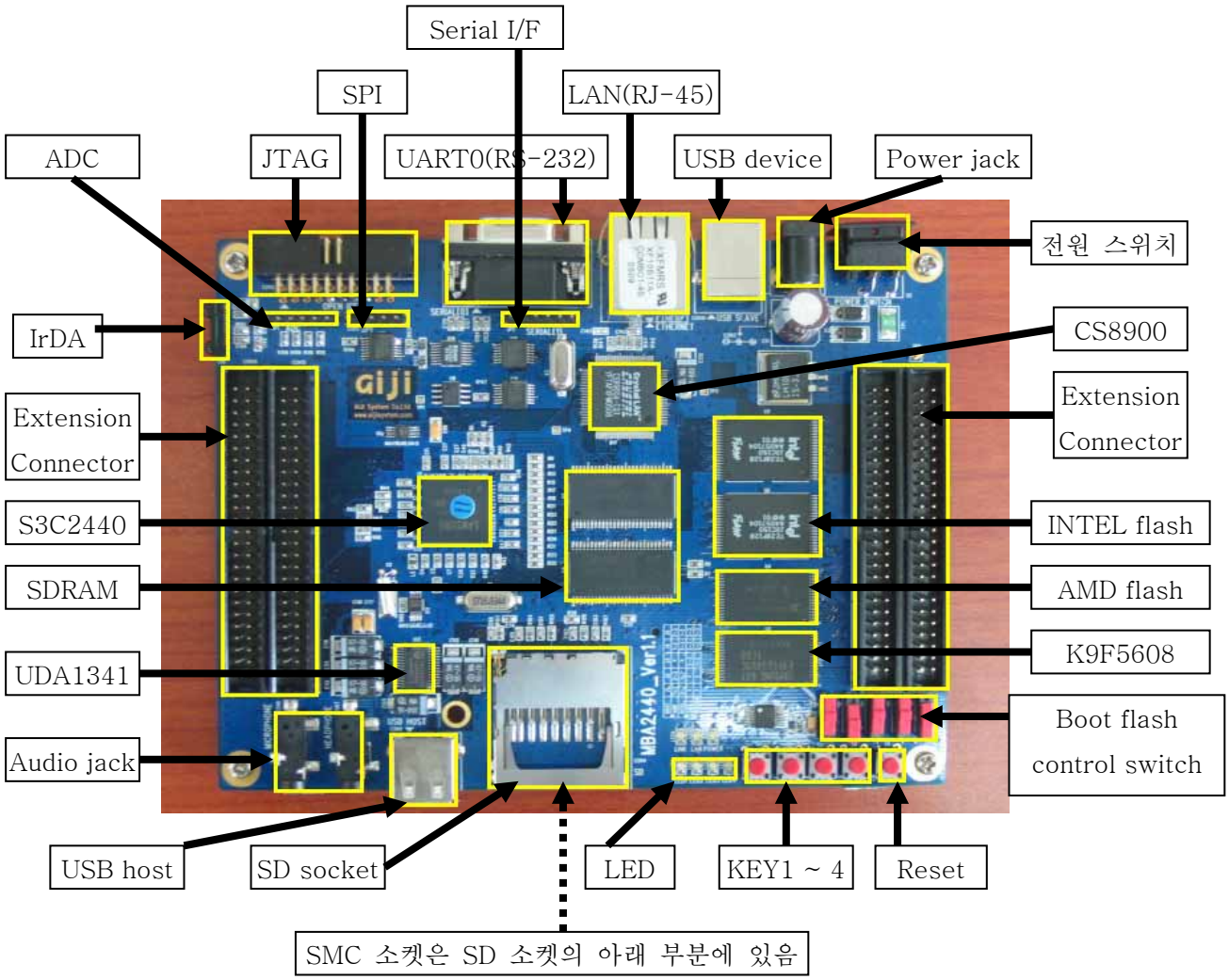
S3C2440를 기반으로 한 MBA2440 보드는 S3C2440와 여러 가지 디바이스들을 하드웨어로 연결하고 이를 이용하여 여러 가지 solution들을 개발할 수 있음을 보여 주기 위한 reference 보드이다.

MBA2440의 block diagram은 [그림 1.2]와 같다.



[그림 1.2] MBA2440 Block Diagram

[그림 1.3]은 MBA2440의 실제 모습과 각 부 명칭을 보여준다.



[그림 1.3] MBA2440 실제 모습

각각의 디바이스에 대해서 간략하게 알아보자.

☞ SDRAM

- 시스템 메모리로 사용됨. 기본적으로 64MB 지원. 128MB로 확장이 가능. nGCS6(0x30000000)에 연결

☞ AMD flash(AM29LV800BB)

- NOR 플래시 메모리로서 boot flash로 사용하기 위한 플래시 메모리. 1MB 크기.
- 점퍼 세팅에 따라 nGCS0(0x0) 또는 nGCS4(0x20000000)에 연결됨

☞ INTEL flash(28F128J3C)

- NOR 플래시 메모리로서 boot flash로 사용 또는 비휘발성 저장장치로 사용. 구매시 option 사항임. 32MB 크기
- 점퍼 세팅에 따라 nGCS0(0x0) 또는 nGCS1(0x08000000)에 연결됨

☞ NAND flash(K9F5608)

- boot flash 또는 비휘발성 저장 장치로 사용. 32MB 크기. S3C2440 NAND controller에 연결

☞ UART0

- MAX3232를 통해 시리얼 콘솔로 사용됨. RS-232 connector

☞ Serial I/F

- S3C2440 UART1에 직접 연결되어 있는 6 pin connector.
  Serial chip(MAX3232) 연결되어 있지 않음

☞ EEPROM

- IIC interface의 KS24C080 serial EEPROM. 1KB 크기.

☞ IrDA

- 적외선 시리얼 통신 포트. S3C2440 UART2에 연결.

☞ ADC I/F

- S3C2440 ADC controller에 직접 연결되어 있는 6 pin connector

☞ SPI I/F

- S3C2440 SPI controller에 직접 연결되어 있는 5 pin connector

☞ JTAG

- 20핀 JTAG용 connector. 여러 에뮬레이터 장비 연결 등에 사용.

☞ CS8900

- 10BASE-T Ethernet controller. RJ-45 connector에 연결. 20MHz clock source 연결.

☞ USB host , device

- S3C2440 USB host, device controller에 연결

☞ Power

- 2A이상의 5V 어댑터 사용. Regulator를 통해 1.3V(core 전압), 3.3v(VDD, 디바이스 전압) 공급

☞ UDA1341

- audio codec chip. S3C2440 IIS에 연결.

☞ SD I/F

- S3C2440 SDIO controller에 연결. SD 소켓.

☞ SMC I/F

- K9F5608과 같이 S3C2440 NAND controller에 연결. Chip select 점퍼(J3)에 의해서 선택
  SD 소켓 아랫면에 SMC 소켓이 있음

☞ Extension Connector

- 총 200핀의 connector. S3C2440의 address, data bus 및 대부분의 컨트롤 시그널 연결

☞ LED display

- power, Ethernet 상태 표시 LED와 직접 제어할 수 있는 4개의 LED

☞ Buttons

- reset 버튼 및 외부 시그널 입력을 위한 4개의 버튼(KEY1 ~ 4)

☞ Boot flash control switch

- MBA2440 부트롬 설정과 NAND 플래시 설정을 위한 5개의 스위치

☞ S3C2440

- MBA2440의 핵심인 MCU. 400MHz clock(1.3V core 전압)
- 16.9344MHz System clock을 사용
- 32.768KHz RTC clock 사용

1.3 MBA2440 H/W configuration

MBA2440을 효과적으로 사용하기 위해 하드웨어적으로 설정해야 할 부분들에 대해서 알아보자.

하드웨어 설정에 관해서는 다음의 세 가지로 구분할 수 있다.

- System Memory Configuration
- Boot flash Configuration
- NAND controller Configuration

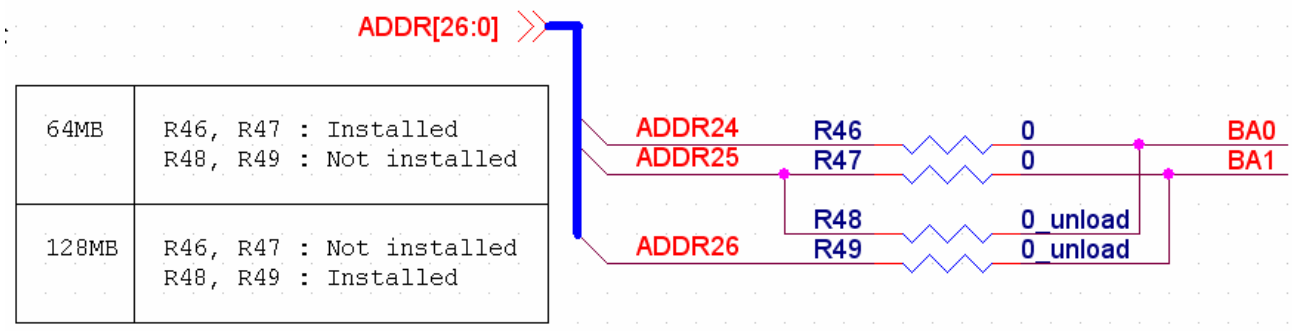
1.3.1 System Memory Configuration

MBA2440은 제품 출고 당시 기본적으로 64MB의 시스템 메모리를 사용하도록 32MB의 삼성 메모리 칩(K4S561632E-TC75) 2개를 보드에 붙여 출시한다. 그러나, 개발자의 요청에 따라 128MB로 업그레이드하여 보드를 출시하기도 한다.

개발자가 직접 칩을 구할 수 있다면 128MB로 업그레이드 할 수 있다. 이 경우에는 64MB의 삼성 칩(K4S510632E-TC75) 2개를 사용하게 된다.

64MB에서 128MB로 업그레이드하게 되면, 당연히 address bus에도 변화가 생겨야 한다. 64MB 메모리를 접근할 수 있는 address bus로는 128MB 크기의 메모리 영역 전부를 접근할 수 없기 때문이다.

64MB	R46, R47 : Installed R48, R49 : Not installed
128MB	R46, R47 : Not installed R48, R49 : Installed



[그림 1.4]

[그림 1.4]에서 보는 바와 같이, **64MB일 경우**에는 보드 뒷면에 저항 **R46, R47에 0Ω 저항**이 달려 있어 short 되어 있다. **128MB로 업그레이드 할 경우**에는 R46, R47을 open 시키고 **R48, R49를 short** 시켜 주어야 한다.

위와 같이 저항의 상태를 변경한 후, 64MB의 SDRAM 두 개를 붙이면 128MB로 업그레이드 된다.

1.3.2 Boot flash Configuration

MBA2440은 여러 개의 flash 메모리가 있다.

flash memory란 무엇인가?

Flash memory는 하나의 칩으로 구성되어 있으며, 시스템 메모리인 SDRAM과 다르게 비 휘발성 메모리이다. 보드의 전원이 꺼져도 데이터들을 그대로 저장하는 ROM의 장점과 데이터의 입출력이 자

유로운 RAM의 장점을 모두 갖추고 있기에 임베디드 시스템에서 데이터 저장 매체로서 가장 많이 쓰이고 있다.

이 flash memory는 내부 방식에 따라 NOR형과 NAND형 두 가지로 나뉘어 진다.

NOR flash는 셀이 병렬로 연결된 방식이다. 이는 NAND에 비해 데이터 access time이 짧아 읽기 속도가 빠르기 때문에 휴대폰에서 주로 사용되는 메모리이다. 그러나, NAND에 비해 제조 단가가 비싸다.

NAND flash는 셀이 직렬로 연결된 방식이다. 고용량 집적도 구현이 쉬워 NOR에 비해 고용량 메모리로서 월등한 기능을 수행하며, 쓰기 속도가 NOR에 비해 빠르기 때문에 음성이나 동영상 멀티미디어를 처리하는데 주로 쓰인다.

MBA2440에서는 NOR flash memory로,

- **AM29LV800BB** (AMD flash) : 1MB [기본 사항]
- **INTEL 28F128J3C** (INTEL flash) : 32MB(16MB x 2) [옵션 사항]

를 사용할 수 있으며,

NAND flash memory로,

- **K9F5608** (SOP) : 32MB [기본 사항]
- **K9S1208** (SMC) : 64MB [옵션 사항]

를 사용할 수 있다.

MBA2440은 이 4개의 플래시 메모리중 하나를 boot flash memory로 사용할 수 있다.

Boot flash란 전원을 켰을 때 S3C2440에서 0x0 번지로 인식되며, 제일먼저 수행되어야 할 코드를 저장하여 전원이 켜짐과 동시에 이 코드가 수행될 수 있도록 하기 위한 flash를 말한다.

어떤 flash 메모리를 boot flash로 사용하느냐는 보드 우측 하단의 5개의 boot flash control switch를 통해 설정할 수 있다.

이에 대한 설정 방법을 보드에 silk로 간단하게 표시하였다.

이 boot flash 설정표와 boot flash control switch를 자세히 보면 다음과 같다.

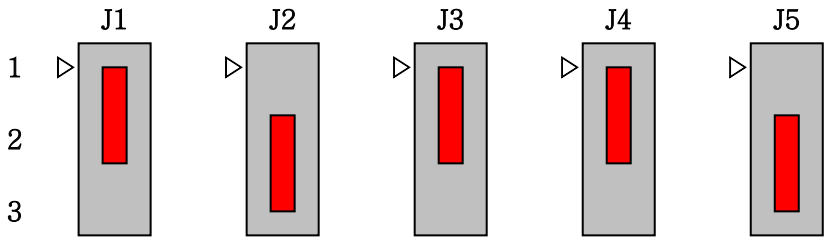
표에서 x 표시는 don't care의 의미이다.

[표 1.1] boot flash 설정표

BOOT	J1	J2	J3	J4	J5
AMD	1-2	2-3	X	1-2	2-3
INTEL	2-3	1-2	X	2-3	1-2
SMC	x	x	2-3	2-3	2-3
K9F5608	x	x	1-2	2-3	2-3

Boot flash control switch는 보드 우측 하단 모서리 부분에 있다.

기본적으로 보드가 출시될 당시에는 아래와 같은 [그림 1.5]처럼 설정되어 있다.



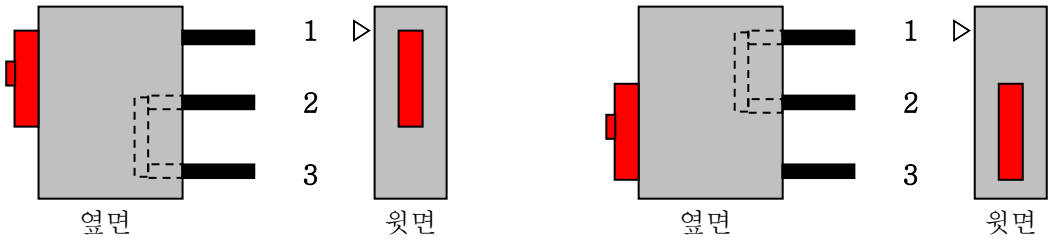
[그림 1.5]

[그림 1.5]의 설정은 AM29LV800 NOR flash를 Boot flash로 설정한 것이며, 제어 가능한 NAND flash는 SOP(K9F5608) NAND flash이다.

여기서 주의할 점이 있다.

하나의 boot flash control switch는 3개의 핀이 보드에 연결되어 있는데,

아래 [그림 1.6]에서 보는 바와 같이 스위치가 위로 올라가 있으면 실제로는 2번 핀과 3번 핀이 연결이 되며, 스위치가 아래로 내려가 있으면 실제로는 1번 핀과 2번 핀이 서로 연결된다는 점이다.



스위치가 1-2일 경우, 2-3번 핀이 연결 **스위치가 2-3일 경우, 1-2번 핀이 연결**

[그림 1.6]

이 문서에서는, **핀의 연결 상태를 기준으로 설명하지 않고, 스위치의 위치를 기준으로 설명한다는 점을 숙지**하기 바란다.

이제 각각의 스위치에 대해서 알아보자. **(이 부분부터는 회로도를 같이 참조하면서 볼 것)**

가장 먼저 알아보아야 할 스위치는 J4와 J5이다.

J4, J5는 S3C2440의 OM0, OM1 시그널을 제어한다. OM0, OM1 시그널은 nGCS0에 연결될 boot flash의 data bus를 설정하기 위한 시그널이다. 자세한 사항은 S3C2440 매뉴얼 5장의 memory controller 부분을 참조하기 바란다.

OM0 시그널은 J4를 통해 제어하며, **OM1 시그널은 J5를 통해 제어**한다.

OM0와 OM1이 모두 ground에 연결(J4, J5 모두 2-3로 설정)되면, **NAND flash를 boot flash로 설정**하게 된다.

OM0가 VDD에 연결(J4가 1-2로 설정)되고 OM1이 ground에 연결(J5가 2-3로 설정)되면, **data bus가 16비트로 설정** 된다. 이것은 **AMD flash를 boot flash로 사용할 경우에 해당**한다. AMD flash는 data bus가 16비트로 연결되어 있음을 회로도를 통해 확인할 수 있다.

OM0가 ground에 연결(J4가 2-3로 설정)되고 OM1이 VDD에 연결(J5가 1-2로 설정)되면, data bus가 32비트로 설정 된다. 이것은 **INTEL flash를 boot flash로 사용할 경우에 해당**한다. INTEL flash는 data bus가 32비트로 연결되어 있음을 회로도를 통해 확인할 수 있다.

다음으로 J1, J2를 알아보자

J1, J2는 두 개의 NOR flash memory 중 어떤 NOR flash를 boot flash로 사용할지를 결정한다.

J1이 1-2로, J2가 2-3로 연결된 경우에는 회로도에서 nCE16 시그널이 S3C2440의 nGCS0에 연결된다. nCE16은 AM29LV800BB flash memory의 nCE에 연결된 시그널이다. 그러므로 **AM29LV800BB flash가 boot flash로 설정**된다.

J1이 2-3로, J2가 1-2로 연결된 경우에는 회로도에서 nCE32 시그널이 S3C2440의 nGCS0에 연결된다. nCE32는 INTEL 28F128J3C flash memory의 nCE0에 연결된 시그널이다. 그러므로 **INTEL 28F128J3C flash가 boot flash로 설정**된다.

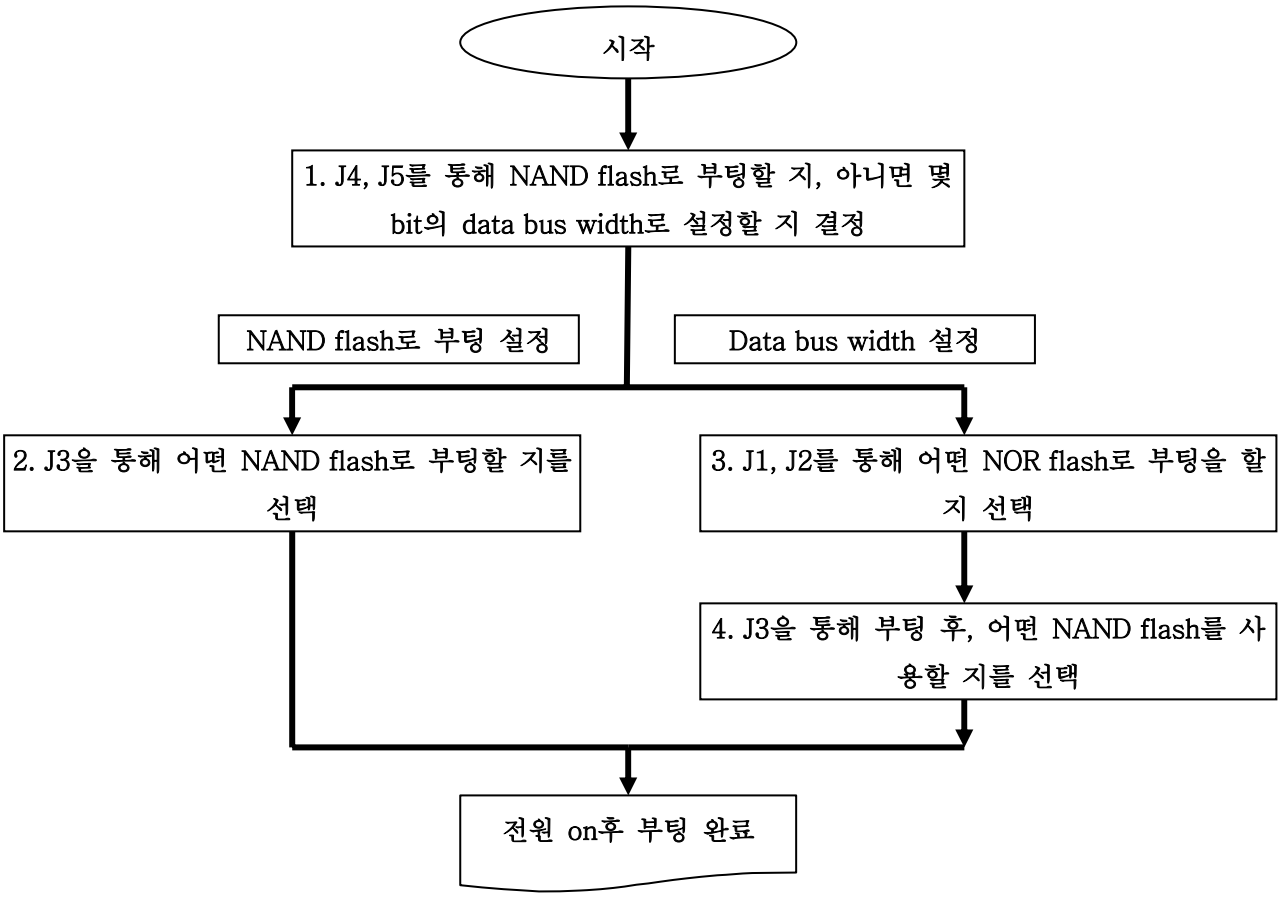
당연히, J1, J2가 둘 다 1-2로 설정된다면 S3C2440은 어떤 NOR flash를 boot flash로 잡을지 종잡을 수 없기에 동작하지 않게 되며, 둘 다 2-3으로 설정이 되면 nGCS0로 연결된 플래시 메모리가 없기에 boot flash가 없는 보드나 마찬가지가 되므로 동작하지 않는다. 이를 주의하자!!!

J4, J5를 통해 NAND flash를 boot flash로 설정한 경우, J3은 두 개의 NAND flash memory 중 어떤 NAND flash를 boot flash로 사용할지를 결정한다.
J4, J5를 통해 NOR flash를 boot flash로 설정한 경우, J3은 어떤 NAND flash를 제어할 지를 결정한다.

J4, J5를 통해 NAND flash를 boot flash로 설정한 상태에서,
J3이 1-2로 연결된 경우에는 S3C2440 NAND controller를 통해 K9F5608 NAND flash를 제어할 수 있게 된다.
J3이 2-3으로 연결된 경우에는 S3C2440 NAND controller를 통해 K9S1208(SMC) NAND flash를 제어할 수 있게 된다.

NAND flash memory는 S3C2440의 NAND controller를 통해서만 제어하기 때문에 동시에 두 NAND 플래시를 제어할 수 없다.

boot flash control switch 세팅을 하기 위해서는 다음과 같은 순서로 setting을 한다.



[그림 1.7]

점퍼 설정에 관한 쉬운 이해를 도모하기 위해 [그림 1.7]과 같이 간단한 순서도를 그렸다.

1.3.2.1 NOR flash를 Boot flash으로 사용하기

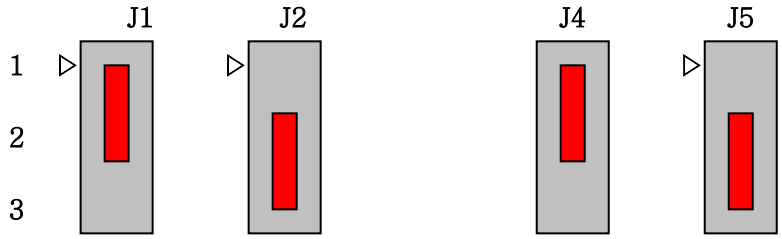
NOR flash를 Boot flash로 설정하기 위해서 위 순서도를 기준으로 설명하고자 한다.

AM29LV800BB를 Boot flash으로 사용할 경우 (그림 1.8)

1. AM29LV800BB로 설정하고 싶다면, J4를 1-2로, J5를 2-3으로 설정하여 data bus를 16비트로 설정한다.

2. J1을 1-2로, J2를 2-3으로 설정하여 nGCS0를 AM29LV800BB의 nCE에 연결한다.

3. 부팅 후, J3를 설정하여 test program이나 Linux, WinCE에서 제어 하고자 하는 NAND flash를 선택한다. J3은 NOR flash를 boot flash로 설정하는데 아무런 영향을 주지 않는다.
   J3가 1-2일 경우에는 K9F5608 NAND flash 제어가 가능하며,
   J3가 2-3일 경우에는 K9S1208 NAND flash 제어가 가능하다.



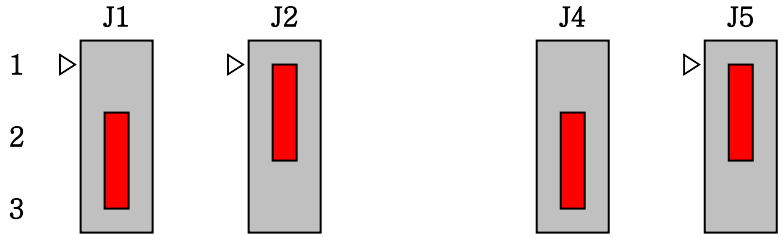
[그림 1.8] AMD flash를 boot flash로 설정

INTEL 28F128J3C를 Boot flash로 사용할 경우(그림 1.9)

1. E28F128J3C로 설정하고 싶다면, J4를 2-3로, J5를 1-2으로 설정하여 data bus를 32비트로 설정한다.

2. J1을 2-3로, J2를 1-2으로 설정하여 nGCS0를 E28F128J3C의 nCE0에 연결한다.

3. 부팅 후, J3를 설정하여 test program이나 Linux, WinCE에서 제어 하고자 하는 NAND flash를 선택한다. J3은 NOR flash를 boot flash로 설정하는데 아무런 영향을 주지 않는다.
   J3가 1-2일 경우에는 K9F5608 NAND flash 제어가 가능하며,
   J3가 2-3일 경우에는 K9S1208 NAND flash 제어가 가능하다.



[그림 1.9] INTEL flash를 boot flash로 설정

1.3.2.2 NAND flash를 Boot flash로 사용하기

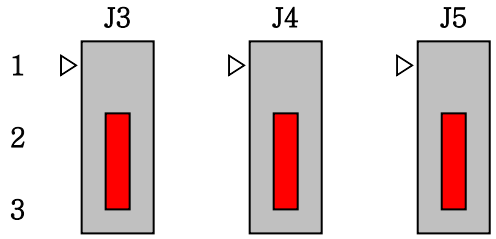
NAND flash를 Boot flash로 설정하기 위해서 위 순서도를 기준으로 설명하고자 한다.
([그림 1.10]. [그림 1.11])

1. J4, J5를 모두 2-3으로 설정한다.
2. J3을 통해 K9F5608, K9S1208(SMC) 둘 중 하나를 Boot flash로 설정한다.
   J3가 1-2일 경우에는 K9F5608 NAND flash가 boot flash로 설정되며,
   J3가 2-3일 경우에는 K9S1208 NAND flash가 boot flash로 설정된다.

NAND flash로 부팅할 경우, S3C2440의 nGCS0에 연결되어 있는 NOR flash에는 접근이 불가능하게 된다. nGCS1이나 nGCS4에 연결되어 있는 NOR flash는 제어가 가능하다. 제어하고자 하는 NOR flash가 nGCS0에 연결되지 않도록 J1, J2를 설정한다.

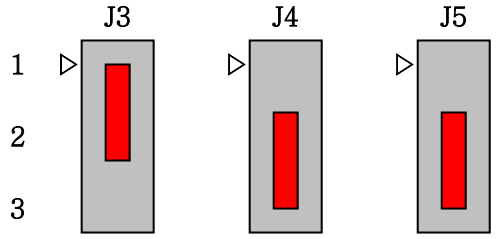
J1, J2는 NAND flash를 boot flash로 설정하는데 아무런 영향을 주지 않는다.

SMC(K9S1208)를 Boot flash로 사용할 경우



[그림 1.10] SMC(K9S1208)을 boot flash로 설정

SOP(K9F5608)를 Boot flash로 사용할 경우



[그림 1.11] SOP(K9F5608)을 boot flash로 설정

1.3.3 NAND controller Configuration

S3C2440은 NAND controller가 있다. 이 NAND controller는 NAND flash control signal을 통해 NAND flash memory를 쉽게 제어할 수 있게 해준다.

S3C2440에 연결할 수 있는 NAND flash memory의 종류는 다양하다. 그러나, S3C2440 NAND controller는 삼성에서 제공하고 있는 NAND flash memory들과의 원활한 연동을 하도록 최적화 되어 있다.

NAND controller는 연결되는 NAND flash memory의 종류에 따라 **normal mode**와 **advanced mode**로 설정을 하게 되며, 이에 따라 memory page size, memory address cycle, I/O bus width의 설정이 달라지게 된다. 이 설정은 **소프트웨어 상에서 설정하는 것이 아니라 하드웨어상에서 설정**을 하게 되어 있다.

이를 설정하는 시그널들이 바로 S3C2440에 있는 **NCON, GPG13(EINT21), GPG14(EINT22), GPG15(EINT23)** 시그널이다.

NAND FLASH MEMORY CONFIGURATION TABLE

NCON0	GPG13	GPG14	GPG15
0: Normal NAND	0: 256Words	0: 3-Addr	0: 8-bit bus width
	1: 512Bytes	1: 4-Addr	
1: Advance NAND	0: 1Kwords	0: 4-Addr	1: 16-bit bus width
	1: 2Kbytes	1: 5-Addr	

Note

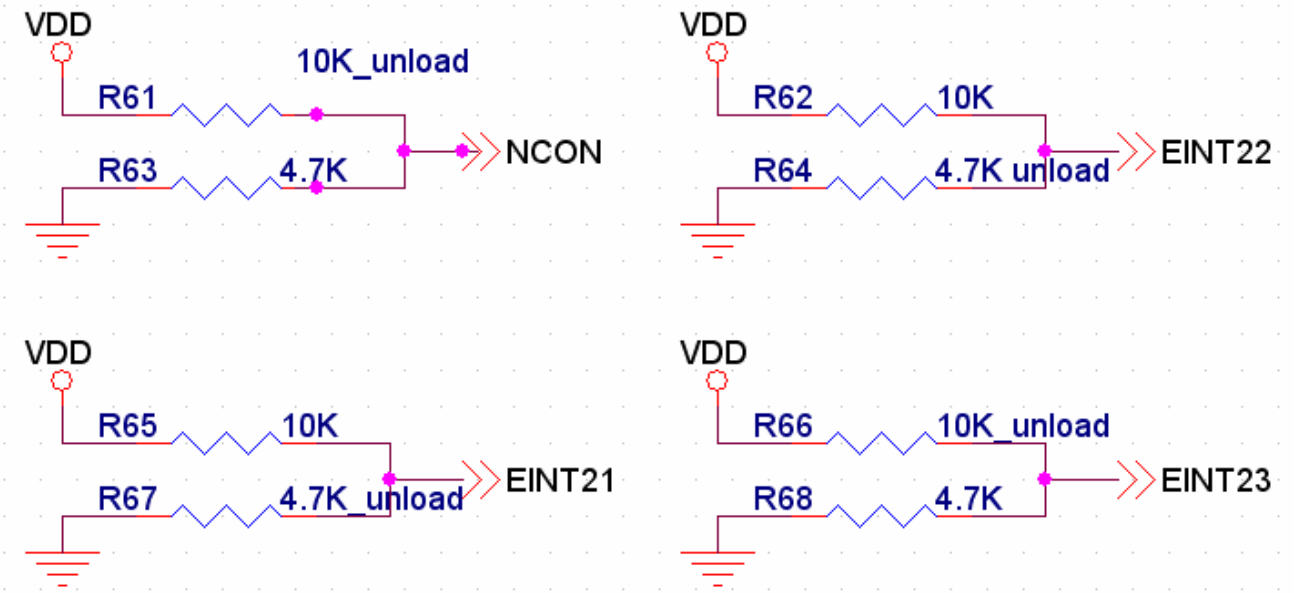
With above 4-bit, Possible total combinations are 16, but not all the value can be used.

Example) Nand flash configuration setting.

Parts	Page size/Total size	NCON0	GPG13	GPG14]	GPG15
K9S1208V0M-xxxx	512Byte / 512Mbit	0	1	1	0
K9K2G16U0M-xxxx	1KW / 2Gbit	1	0	1	1

Normal mode 또는 advanced mode에서 동작하기 위한 하드웨어 설정은 위와 같다.

MBA2440은 NAND controller 하드웨어 설정을 위해 회로도를 [그림 1.12]과 같이 구성하였다.



[그림 1.12]

MBA2440은 normal mode로 동작하도록 설정되어 있다.

K9K2G8U0M 과 같은 NAND flash memory를 연결하여 사용하려면, Advanced mode로 설정 해야 한다. Advanced mode의 설정은 NCON = 1, GPG13 = 1, GPG14 = 1, GPG15 = 0으로 시그널을 연결하면 된다.

위 회로도 상에서 R63을 unload 시키고, R61에 저항을 연결하면 된다.

자세한 사항은 S3C2440의 datasheet를 참조하기 바란다.

2 MBA2440 Quick Start

이 장에서는 MBA2440을 처음 접하는 개발자가 PC와 MBA2440에 케이블 연결을 하고 보드 구매시 세팅되어 있는 firmware나 OS를 쉽게 동작시켜 볼 수 있도록 동작 순서 및 방법을 소개한다.

먼저, 기본이 되는 firmware를 구동시키는 방법에 대해서 알아보고, 다음으로는 Linux와 WinCE를 구동시키는 방법에 대해서 알아본다.

2.1 Environment Setup

MBA2440을 처음 구동시키기 위해서는 데스크탑 PC와의 연동이 필요하며, PC에서 수행되어야 할 프로그램들을 설치하거나 설정해 주는 일이 가정 먼저 선행 되어야 한다.

이 장에서는 설명하고자 하는 기본적인 환경 구축 내용은 Windows XP 상에서 이루어지는 작업이며 다음과 같은 어플리케이션들이 설치되어 있어야 한다.

◈ **DNW, 하이퍼 터미널 시리얼 에뮬레이터**
: 리눅스의 minicom과 같은 시리얼 콘솔용 프로그램의 설정

◈ **Windows 용 TFTP 프로그램**
: 리눅스의 TFTP 프로그램과 같은 역할을 하는 windows 용 TFTP 서버 프로그램

이 환경 구축 내용은 Windows 2000 이상의 OS에서는 문제가 없으나 Windows 98에서는 환경 구축이 제대로 되지 않을 수 있음을 미리 밝힌다.

먼저, Windows OS상에서의 Serial 통신 에뮬레이터인 DNW와 하이퍼터미널을 설정하는 방법을 알아본다.

Linux OS상에서의 Serial 통신 에뮬레이터 설정 방법에 대해서는 이 문서의 3.3.4 Minicom 설정을 참고 하기 바란다.

다음으로는, LAN을 통해 보드에 이미지를 전송하기 위해 TFTP 서버 프로그램의 설정에 대하여 알아본다. Linux OS상에서의 TFTP 서버 프로그램의 설정에 대해서는 이 문서의 3.3.2 리눅스 상에서 TFTP 서버 설치를 참조 하기 바란다.

TFTP의 사용을 위해서 Windows PC, Linux PC, MBA2440의 IP 주소 설정도 필요하다.
주소 설정은 사설 네트워크망을 구성하여 사용한다.
이에 대한 사항은 3.1 개발 환경 구축 예를 참고하여 환경을 구축하기 바란다.

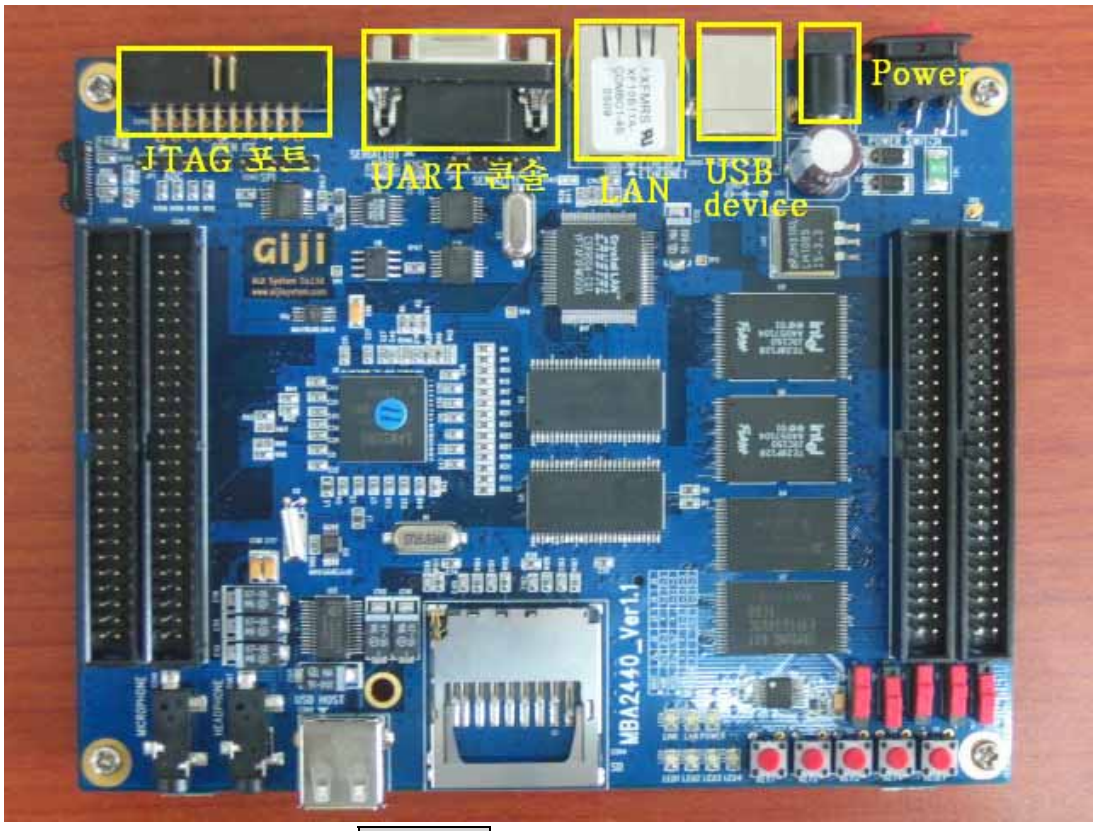
먼저, 케이블 연결에 대해 알아보자.

2.1.1 케이블 연결

PC와 MBA2440에는 여러 개의 케이블이 연결된다. MBA2440과 PC간에 연결되어야 할 케이블들에

대해 알아본다.

[그림 2.1]은 연결해야 할 포트들을 표시한 것이다.



[그림 2.1] 연결해야 할 port들

2.1.1.1 Power 케이블

제일 먼저 연결해야 할 케이블이 power 케이블이다. MBA2440에 전원을 공급하기 때문이다. 5V/2A 어댑터를 연결한다. 기본적으로 연결되어 있어야 한다.

2.1.1.2 UART 콘솔용 케이블

UART0 포트에 UART 콘솔용 케이블을 연결한다.

보드 구매시 제공하는 시리얼 케이블을 이용하면 된다. 이 케이블은 direct 케이블 이다.

시리얼 케이블의 한쪽은 MBA2440의 UART0 콘솔용 포트에 연결하고, 다른 쪽은 PC의 COM1이나 COM2에 연결하면 된다. 대부분 COM1에 연결한다.

이 케이블로 PC와 MBA2440이 연결되어 있어야 DNW나 하이퍼 터미널, minicom으로 MBA2440을 모니터링 할 수 있다. 이 케이블은 기본적으로 연결하도록 하자.

2.1.1.3 LAN 케이블 연결

LAN케이블은 두 종류가 있다. cross케이블은 PC와 MBA2440을 직접 연결할 때 사용한다.

허브를 이용하여 MBA2440과 PC를 연결할 때에는 direct 케이블을 이용한다.

LAN 케이블이 연결되어 있어야 u-boot의 “tftp” 명령이나 리눅스 부팅 후의 NFS 파일 시스템을 사용할 수 있다.

2.1.1.4 JTAG 케이블 연결

OpenICE-A900이나 OpenICE-A1000등의 디버거 장비를 연결하기 위한 JTAG 포트가 MBA2440에 있다. 이 포트를 통해 디버거 장비와의 연동이 가능하다. 디버거 장비를 사용할 때 연결하자.

2.1.1.5 USB device

보드 구매시 제공하는 USB device용 케이블은 MBA2440 test program의 마지막 메뉴 USB download를 실행 할 때 연결한다.

PC의 USB 포트에 케이블을 연결하고 다른 한 면은 MBA2440 USB device 포트에 연결한다.

USB 케이블을 이용해 다운로드 할 때에는, 반드시 Windows OS PC에서 DNW 프로그램을 통해서 다운로드해야 함을 유의한다.

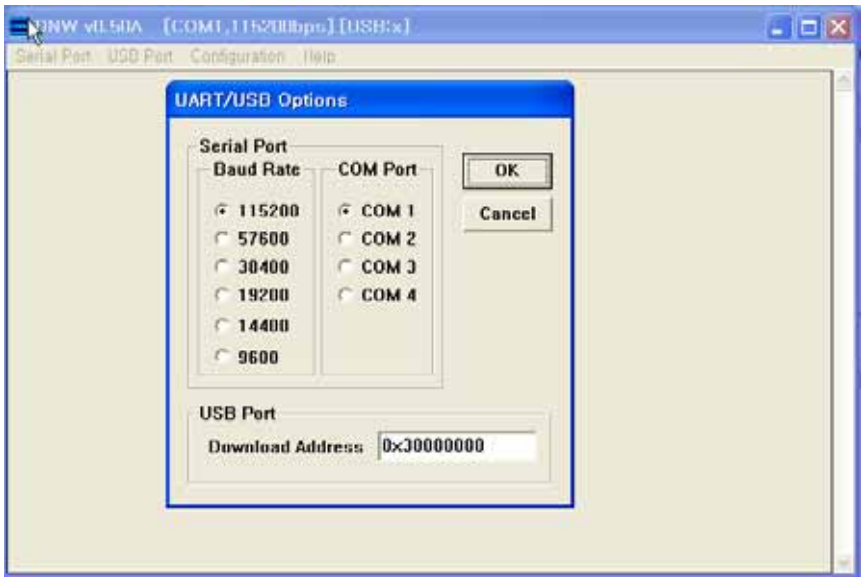
USB 케이블은 보드의 전원 케이블로도 사용이 가능하다.

자세한 사항은 4.3.18 USB download를 참조하라.

2.1.2 DNW 설정

DNW 프로그램은 제공한 CD의 utility/DNW 디렉토리 안에 **dnw.exe** 파일이다. 이 파일을 실행 시키면 된다.

[그림 2.2]과 같이 설정한다. 시리얼 케이블이 연결되어 있는 COM port를 정확히 설정하는 것을 잊지 말아야 한다. 이 문서에서는 PC의 COM1 포트에 케이블을 연결한다.



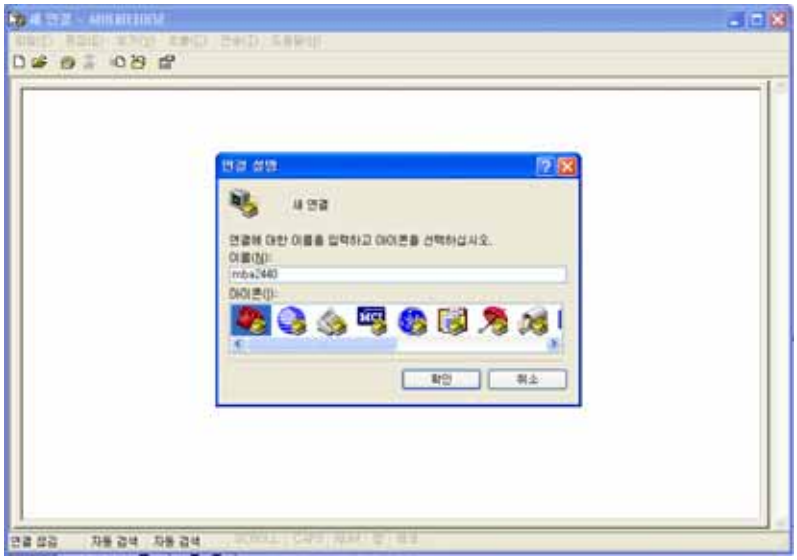
[그림 2.2]

설정이 끝나면 OK 버튼을 누른다. 그 다음 Serial Port->Connect를 누르면 연결된다.

2.1.3 하이퍼터미널 설정

1. 시작 -> 프로그램 -> 보조프로그램 -> 통신 -> 하이퍼터미널을 누른다.
2. 위치 정보를 설정하는 창이 나오는데 취소 버튼을 누른다.

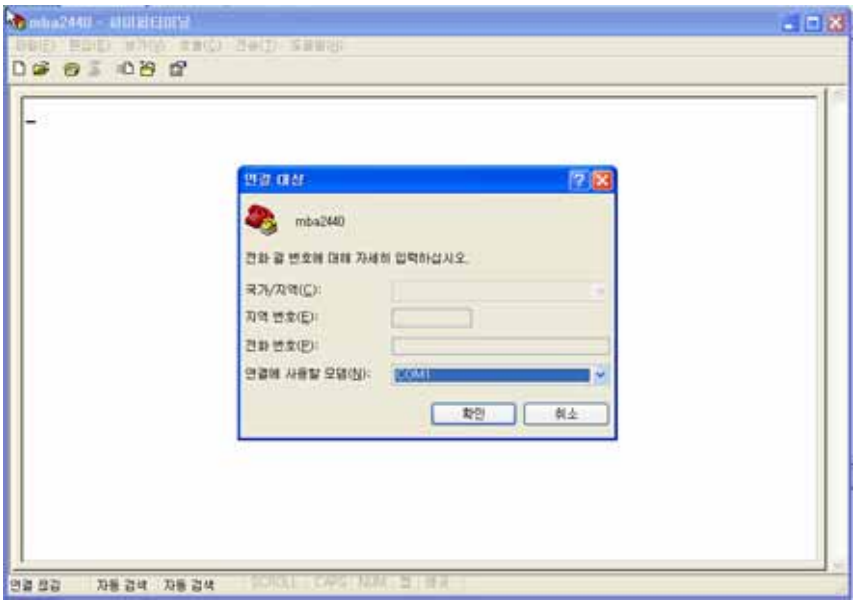
3. 하이퍼 터미널이 뜨고 연결 설명 창이 나오는데 mba2440 이라고 적는다. 이것은 원하는 임의의 글을 적으면 된다. [그림 2.3]



[그림 2.3]

4. 확인 버튼을 누르면 이전과 같은 위치 정보를 설정하는 창이 나오는데 취소 버튼을 누른다.

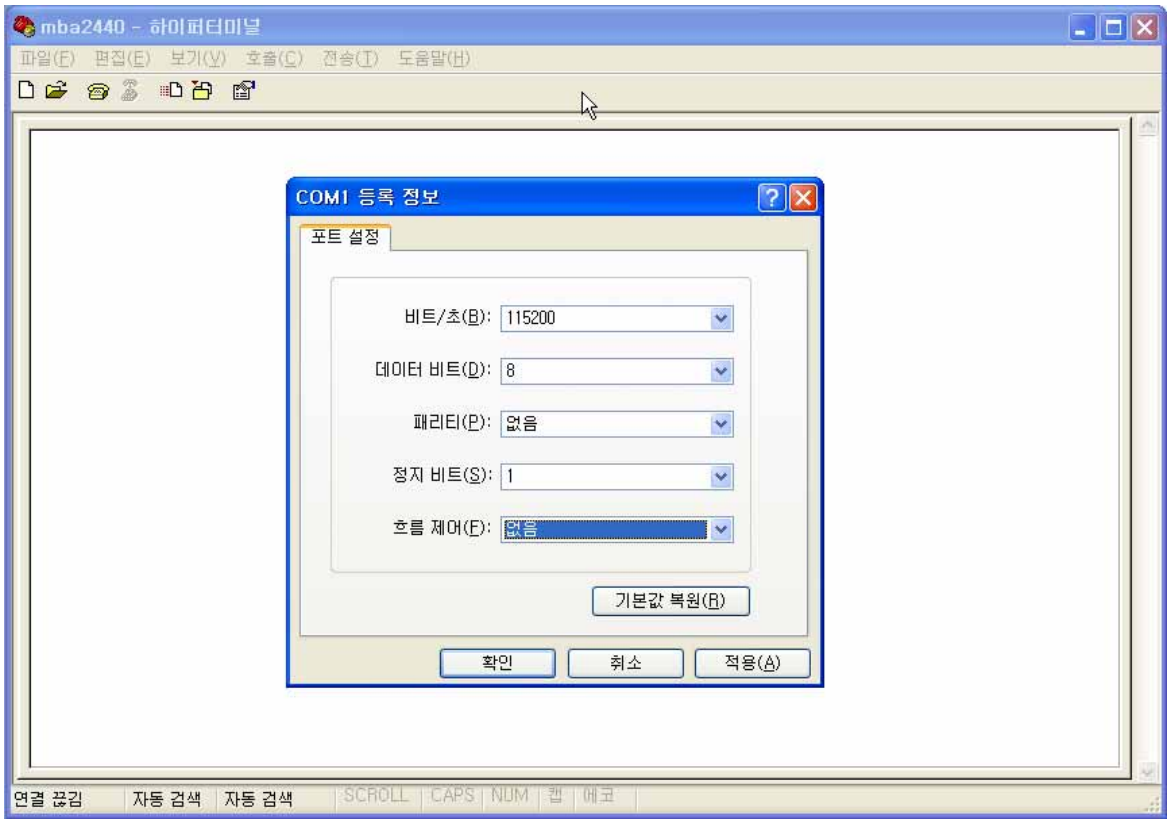
5.다음은 연결 대상을 설정하는 창이 나오는데 연결에 사용할 모뎀은 케이블이 연결되어 있는 COM port로 설정한다. 여기서는 COM1으로 선택하고 확인을 누른다. [그림 2.4]



[그림 2.4]

6. COM1 등록 정보를 설정하는 창이 나온다. 여기서 boudrate, data bit등의 정보를 설정한다. [그림 2.5]와 같이 설정한다.

설정이 완료되면 확인 버튼을 누른다.



[그림 2.5]

2.1.4 Windows용 TFTP 서버 프로그램 설치

MBA2440의 firmware인 u-boot는 큰 용량의 이미지 파일을 외부로부터 받아오기 위해 LAN을 이용한다. 이 때, 사용되는 것이 TFTP 서버와 클라이언트 프로그램이다.

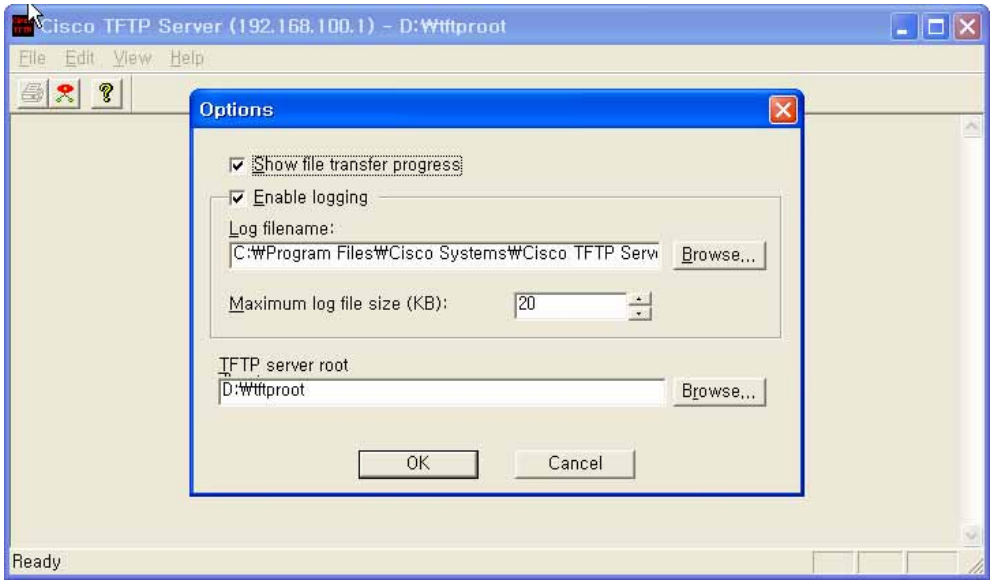
MBA2440에서 수행되는 OS 이미지들은 대부분 덩치가 크며, 데스크탑 PC에 존재한다. 이 이미지를 데스크탑 PC에서 MBA2440으로 가져오기 위해서는 데스크탑 PC에 TFTP 서버 프로그램이 존재해야 한다. 제공한 CD에는 Windows용 TFTP 서버 프로그램이 있다. 이 장에서는 이를 설치하는 방법을 알아본다.

CD에 있는 **utility/TFTP4win 디렉토리에 TFTPServer.exe 파일**이 있다. 이 파일을 실행시켜 TFTP 서버 프로그램을 설치한다.

설치가 완료된 후에는 이 **TFTP 서버 프로그램의 root 디렉토리를 설정**해 준다. **파일을 TFTP로 보내기 위해서는 이 root 디렉토리 안에 보내고자 하는 파일이나 이미지를 넣어 두어야 함을 유의**하자.

이 문서에서는 **D:/tftproot 를 root 디렉토리로 설정**한다.

D: 드리이브 밑에 tftproot 디렉토리를 생성한 후, View -> Option을 클릭하고 root 를 설정한다.


Cisco TFTP Server (192.168.100.1) - D:₩tftproot
File Edit View Help
Options
Show file transfer progress
Enable logging
Log filename:
C:₩Program Files₩Cisco Systems₩Cisco TFTP Serv
Browse...
Maximum log file size (KB):
20
TFTP server root
D:₩tftproot
Browse...
OK
Cancel
Ready

[그림 2.6]

2.2 MBA2440 Test Program

MBA2440을 위해 제공하는 firmware는 4가지 이다.

- **MBA2440 테스트 프로그램**
  : 보드에 있는 각각의 디바이스들을 테스트하며 제어 방법을 배울 수 있는 프로그램
- **U-boot 부트로더 프로그램**
  : 리눅스, 또는 WinCE를 구동하기 위해 필요한 부트로더
- **nboot 부트로더 프로그램**
  : NAND flash를 boot flash로 설정하고 WinCE를 구동할 경우 필요한 부트로더
- **eboot 부트로더 프로그램**
  : WinCE 개발시 사용되는 부트로더.

이 장에서는 MBA2440 test program을 처음 구동시키는 방법에 대해서만 다루며 자세한 사항은 4장. MBA2440 Test Program에서 다룬다.

U-boot 부트로더 프로그램은 MBA2440에 있어서 중요한 부트로더로서 자세한 사항은 5장. U-Boot에서 다룬다.

Eboot 부트로더 프로그램과 **Nboot 부트로더 프로그램**은 WinCE용 부트로더로서 7장. WinCE 에서 다룬다.

MBA2440 test 프로그램은
제공한 CD의 firmware/test_program 폴더 안에 MBA2440_test_v1.0.zip 압축파일이다.
앞서 설명한 바와 같이, 이 프로그램은 MBA2440에 있는 여러 가지 디바이스들을 테스트 할 수 있도록 되어 있으며, 이 프로그램 소스를 이용하여 개발자가 원하는 프로그램을 개발할 때, 예제 프로그램으로서 유용하게 쓰일 수 있다.

MBA2440은 출고시, AMD flash에 MBA2440 test 프로그램이 적재되어 출고된다. 그러므로, AMD flash를 boot flash로 설정한 후 전원을 켜면, DNW나 하이퍼 터미널, 리눅스의 minicom등의 콘솔을 통해 MBA2440 test 프로그램의 메뉴를 확인할 수 있다.

MBA2440 test 프로그램은 보드 출고 시 AMD flash의 0번지에 적재되어 있다.
MBA2440 test 프로그램은 AMD flash를 boot flash로 설정한 경우에 정상적으로 동작하도록 프로그래밍하였다.
INTEL flash를 boot flash로 설정하고 INTEL flash에 test program을 적재한 후 부팅하거나, NAND flash를 boot flash로 설정하고 NAND flash에 적재한 후 부팅하는 경우에는 특정 메뉴들은 제대로 동작하지 않거나 전혀 동작하지 않게 됨을 유의하기 바란다.

보드의 케이블이 정상적으로 연결되어 있다면, AMD flash를 boot flash로 설정한다.

(1.3.2.1 NOR flash를 Boot flash로 사용하기 참조)

DNW나 minicom 같은 serial console emulator를 실행한 후, 동작 가능하도록 설정한다.

(2.1.1 DNW 설정, 2.1.2 하이퍼터미널 설정, 3.3.4 Minicom 설정 참조)

보드의 전원을 켠다.

[그림 2.7]

[그림 2.7]은 전원을 켠 후, MBA2440 test 프로그램이 DNW를 통해 메뉴 메시지를 출력한 모습이다.

2.3 WinCE 부팅하기

이 장에서는 MBA2440 출고시 K9F5608 NAND flash에 적재되어 있는 u-boot와 WinCE 이미지를 이용하여 WinCE를 실행하는 방법, TFTP를 이용하여 WinCE 이미지를 다운로드하여 실행하는 방법에 대해서 알아본다.

Eboot로 실행하는 방법에 대해서는 7.2 Eboot boot loader에서 다루며, nboot로 실행하는 방법은 7.3 Nboot boot loader에서 다룬다.

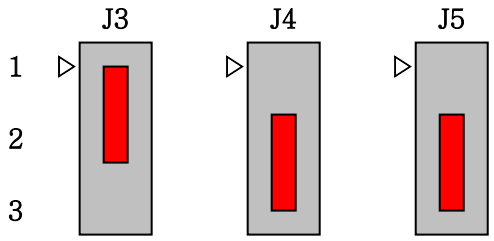
u-boot, WinCE 이미지는 제공한 CD에 있다.

2.3.1 U-boot로 WinCE 부팅하기

K9F5608 NAND flash를 boot flash로 설정하고 전원을 켜면 u-boot가 부팅 된다. U-boot는 부팅 후 2초를 기다린 후, 아무런 입력이 없으면 WinCE를 부팅시킨다. 그러나, 2초가 되기 전에 아무 키나 누르게 되면, u-boot 는 대기상태에 돌입하여 명령어 입력을 기다린다.

보드와 PC간의 케이블들을 정상적으로 연결한다. (2.1.1 케이블 연결 참조)

그림과 같이 스위치를 설정한다.



[그림 2.8]

여기서 잠깐!!

J1, J2에 대해 참고할 사항이 있다. J1, J2는 NAND flash를 boot flash로 설정하는 것에는 아무런 영향을 미치지 않는다.

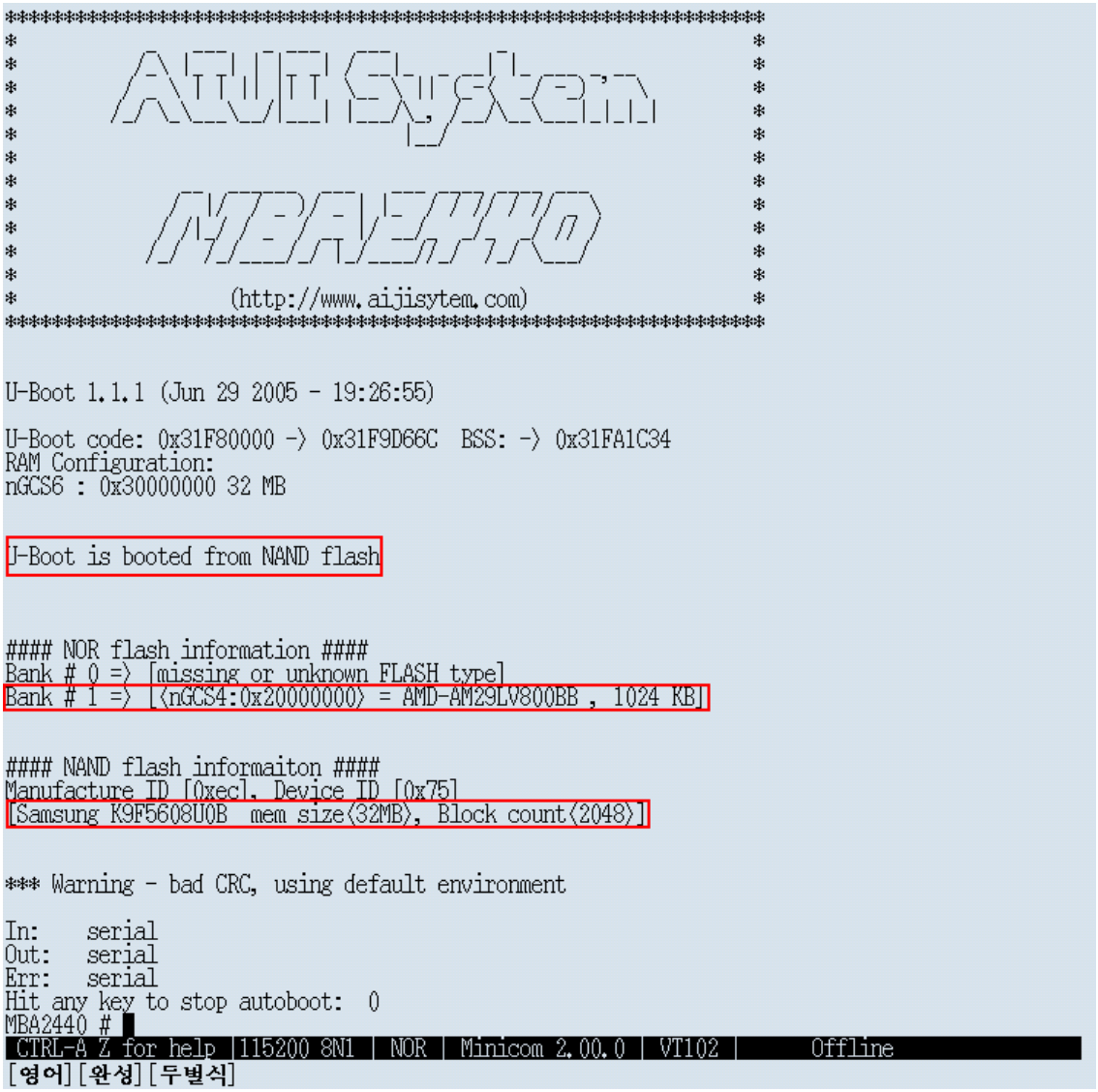
NAND flash를 boot flash로 설정하게 되면,

S3C2440은 물리주소 0x0 번지를 nGCS0가 아닌 stepping stone buffer로 인식(S3C2440 memory map 참조)하기 때문에 u-boot가 부팅하고 나면 nGCS0에 연결되어 있는 NOR flash로는 접근 및 제어가 불가능하다.

그러나, nGCS0가 아닌 다른 chip select pin에 연결되어 있는 NOR flash는 접근 및 제어가 가능하다. 개발자가 u-boot를 NAND flash상에서 동작 시킬 경우에는 자신이 제어하기 원하는 NOR flash를 nGCS0가 아닌 다른 chip select pin에 연결되도록 J1, J2 스위치를 설정해야 한다. 이점을 유의하기 바란다

자, 이제 WinCE를 실행시켜 보자.

보드의 전원을 켜보자.



[그림 2.9]

[그림 2.9]은 전원을 켠 후, 키를 입력하여 프롬프트가 출력된 모습이다. [그림 2.9]에서 보는 바와 같이 **U-boot가 NAND flash로 부팅**하였고, **NAND flash information을 통해서 어떤 NAND flash인지 확인**할 수 있으며 **NOR flash memory로 AMD flash를 인식**하였음을 알 수 있다.

전원을 켜면, 2초를 카운트한다. 2초 안에 아무 키든지 누르면 MBA2440 # 프롬프트가 뜨면서 명령어를 입력하기를 기다린다.

이 때, **"bootrom wince" 명령을 실행** 시키면 K9F5608 NAND flash에 적재되어 있는 WinCE 이미지를 메모리에 복사한 후, WinCE 이미지를 수행하게 된다.

전원을 켠 후에 2초 동안 아무런 입력이 없으면, u-boot는 2초가 지나고 나서 K9F5608 NAND flash에 있는 WinCE 이미지를 메모리로 복사한 후, WinCE를 부팅한다. 복사하는 동안에 LED2의 불이 빠르게 깜박거리는 것을 확인 할 수 있다. 메모리로 복사하는 동안 약 15초 이내의 시간이 소요된다.

[그림 2.10]

[그림 2.10]은 전원을 켠 후, WinCE이미지를 K9F5608 NAND flash에서 메모리로 복사하는 모습이다. 복사가 완료되면, WinCE를 부팅하게 되며 LCD를 통해서 WinCE 동작 여부를 확인할 수 있다.

프롬프트 상에서 bootrom명령어 말고 **NAND flash read 명령어(nandr)로도 리눅스를 수행**할 수 있다.
먼저, 커널 이미지를 메모리에 적재한다.

```
MBA2440 #  nandr  t  30200000  200000  1400000
Read start point : block[128] page[0] size = 0x1400000
Reading data ...-
Read compelete(0x1400000): from 0x200000(128) NAND flash to 0x30200000 SDRAM
```

nandr 명령에서

30200000은 WinCE 이미지를 적재할 메모리의 주소이고,

200000은 WinCE 이미지가 위치한 NAND flash의 주소이며

1400000은 복사하고자 하는 데이터 사이즈이다.

각, **숫자는 모두 16진수(hex) 단위**이다.

마지막으로 WinCE 이미지를 수행한다. 복사된 커널 이미지의 주소는 0x30200000 이다.

```
MBA2440 #  go  wince        (또는 go 30200000)
## Starting WinCE kernel at 0x08lx ... ?Windows CE Kernel for ARM (Thumb Enabled) Built on Jun 7
ProcessorType=0920  Revision=0
sp_abt=ffff5000 sp_irq=ffff2800 sp_undef=ffffc800 OEMAddressTable = 8c201480

Windows CE Firmware Init
INFO: Initializing system interrupts...
.....
```

2.3.2 TFTP로 WinCE 부팅하기

u-boot 상에서 tftp 명령을 이용하여 WinCE 이미지를 메모리에 적재한 후 WinCE를 실행해 보자.

먼저, TFTP로 WinCE를 실행하기 위해서는 네트워크 연결이 먼저 설정되어 있어야 한다. 이에 관한 사항은 3.1 개발 환경 구축 예를 먼저 참조하기 바란다.

WinCE 이미지를 TFTP 서버의 root 디렉토리에 복사한다. (2.1.3 Windows용 TFTP 서버 설치, 3.3.2 리눅스 상에서 TFTP 서버 설치 참조)

WinCE 이미지의 이름은 NK.nb0라 하자.

2.3.2절에서와 같이 u-boot의 TFTP 서버 주소 설정을 자신의 환경에 맞게 설정한다. (IP 주소 설정에 관해서는 3.1 개발 환경 구축 예를 참조)

TFTP 서버가 windows상에서 동작하고 있을 경우에는 MBA2440의 u-boot 환경 변수 serverip를 IP 주소를 192.168.100.1로 설정하며, **리눅스상에서 동작하고 있을 경우에는 IP주소를 192.168.100.2로 설정**한다.

MBA2440의 IP주소는 192.168.100.3이다.

u-boot는 기본적으로 TFTP 서버가 리눅스 상에서 동작하는 경우로 설정하였다.

만약, TFTP 서버가 Windows상에서 동작한다면 u-boot 환경 변수 중에 TFTP 서버 주소를 수정해 준다.

setenv 명령으로 수정을 한 후, printenv 명령으로 수정사항을 확인한다.

```
MBA2440 #  setenv  serverip  192.168.100.1
MBA2440 #  printenv
bootdelay=3
baudrate=115200
ethaddr=08:00:3e:26:0a:5b
stdin=serial
stdout=serial
stderr=serial
filesize=1c9dab
```

```
fileaddr=30800000
netmask=255.255.255.0
ipaddr=192.168.100.3
serverip=192.168.100.1

Environment size: 195/131068 bytes
```

ipaddr 는 MBA2440 자신의 IP주소이며, serverip는 TFTP 서버의 IP주소이다.

WinCE 이미지를 tftp 명령을 통해 다운로드 한다.

```
MBA2440 #   tftp   30200000   NK.nb0
TFTP from server 192.168.100.2; our IP address is 192.168.100.3
Filename 'NK.nb0'.
Load address: 0x30200000
Loading:/
done
Bytes transferred = 20971520 (1400000 hex)
```

WinCE 이미지를 부팅한다.

```
MBA2440 #   go   30200000          (또는 go wince)

## Starting application at 0x30200000 ...┐FMD::FMD_Init
FMD:: BOOT_LOADER not defied
FMD:: NOSYSCALL not defined
FMD:: NOBINFS not defined
FMD::FMD_Init Done
FMD::FMD_OEMIoControl = 0x71c24
PWR: Process Attach
>PWR_Init(602F1C0)
.....
```

2.4 리눅스 부팅하기

리눅스를 실행시키기 위해서는 MBA2440에 적합하게 수정된 u-boot 부트로더를 사용해야 한다. 이 u-boot는 데스크탑 PC의 바이오스와 같은 역할을 담당한다.

이 장에서는 TFTP를 이용하여 커널 이미지와 램디스크를 다운로드하여 리눅스를 실행하는 방법과 NAND flash에 리눅스 커널, 램디스크 이미지를 write하고 u-boot로 리눅스를 실행하는 방법에 대해서 알아본다.

u-boot, 리눅스 커널, 파일 시스템 이미지는 제공한 CD에 있다.

2.4.1 TFTP로 리눅스 부팅하기

K9F5608 NAND flash를 boot flash로 설정한 후 부팅하고 나서 u-boot의 tftp 명령을 통해 커널 이미지와 파일 시스템 이미지를 메모리로 적재한 후 리눅스를 실행 할 수 있다.

먼저, TFTP로 리눅스를 실행하기 위해서는 네트워크 연결이 먼저 설정되어 있어야 한다. 이에 관한 사항은 3.1 개발 환경 구축 예를 먼저 참조하기 바란다.

리눅스 커널, 램디스크 이미지를 TFTP 서버의 root 디렉토리에 복사한다. (2.1.4 Windows용 TFTP 서버 설치, 3.3.2 리눅스 상에서 TFTP 서버 설치 참조)

커널 이미지는 zImag.bin 이며, **램디스크 이미지는 ram2440-ram.gz**라 하자.

TFTP 서버가 windows상에서 동작하고 있을 경우에는 MBA2440의 u-boot 환경 변수 serverip를 IP 주소를 192.168.100.1로 설정하며, **리눅스상에서 동작하고 있을 경우에는 IP주소를 192.168.100.2로 설정**한다.

u-boot는 기본적으로 TFTP 서버가 리눅스 상에서 동작하는 경우로 설정하였다.

만약, TFTP 서버가 Windows상에서 동작한다면 u-boot의 TFTP 서버 주소를 수정해 준다.

setenv 명령으로 수정을 한 후, printenv 명령으로 수정사항을 확인한다.

```
MBA2440 #  setenv  serverip  192.168.100.1
MBA2440 #  printenv
bootdelay=3
baudrate=115200
ethaddr=08:00:3e:26:0a:5b
stdin=serial
stdout=serial
stderr=serial
filesize=1c9dab
fileaddr=30800000
netmask=255.255.255.0
ipaddr=192.168.100.3
```

```
serverip=192.168.100.1

Environment size: 195/131068 bytes
```

ipaddr 는 MBA2440 자신의 IP주소이며, serverip는 TFTP 서버의 IP주소이다.

커널 이미지를 tftp 명령을 통해 다운로드 한다.

```
MBA2440 #  tftp  30c00000  zImage.bin
TFTP from server 192.168.100.2; our IP address is 192.168.100.3
Filename 'zImage.bin'.
Load address: 0x30c00000
Loading:/
done
Bytes transferred = 746208 (b62e0 hex)
```

램디스크 이미지를 tftp 명령을 통해 다운로드 한다.

```
MBA2440 #  tftp  30800000  ram2440-ram.gz
TFTP from server 192.168.100.2; our IP address is 192.168.100.3
Filename 'ram2440.gz'.
Load address: 0x30800000
Loading:-
done
Bytes transferred = 2782866 (2a7692 hex)
```

리눅스 커널을 수행한다.

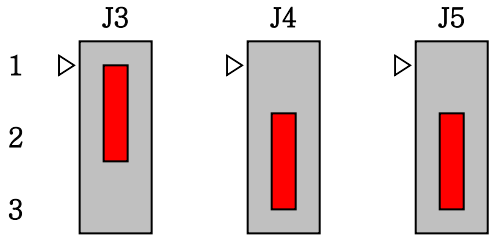
```
MBA2440 #  go  linux        (또는 go 30c00000)
Uncompressing Linux..................................................... done, booting the kern.
Linux version 2.4.20_elfin-d1.5 (root@localhost.localdomain) (gcc version 2.95.3 20010315 (releaT
CPU: ARM/CIRRUS Arm920Tid(wb) revision 0
.....
```

2.4.2 NAND flash상에서 리눅스 부팅하기

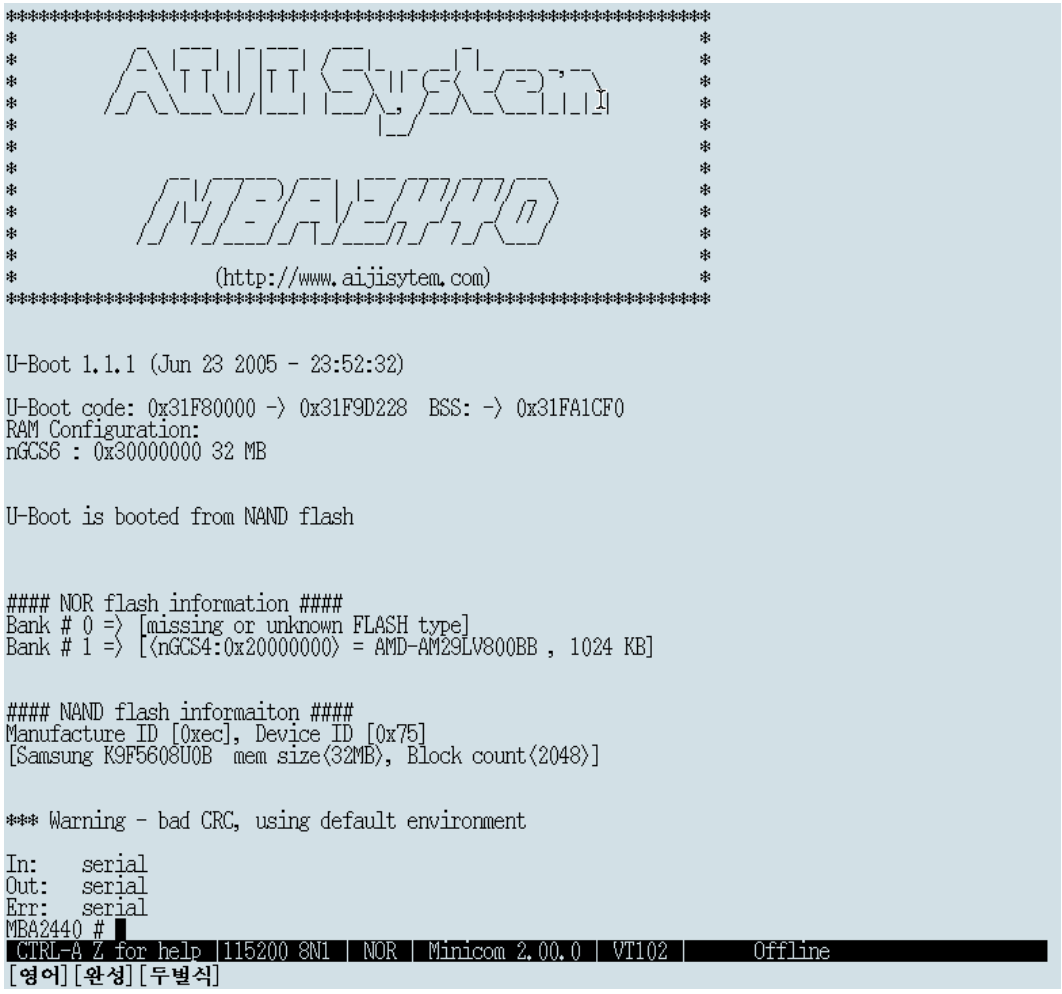
K9F5608 NAND flash를 boot flash로 설정하면 부팅시 u-boot가 동작하게 된다. K9F5608 NAND flash에 u-boot, 리눅스 커널, 램디스크를 write하여 적재하고, 적재된 이미지를 이용하여 리눅스를 실행시켜 보자.

이 장에서는 K9F5608 NAND flash를 예로 설명하지만, K9S1208 SMC를 이용하여도 무방하다.

[그림 2.11]과 같이 스위치를 설정하고 보드의 전원을 켠다.



[그림 2.11]



[그림 2.12]

[그림 2.12]에서 보는 바와 같이 **U-boot가 NAND flash로 부팅**하였으며 **NAND flash information을 통해서 어떤 NAND flash인지 확인**할 수 있다.

TFTP를 이용하여 u-boot와 커널 이미지, 램디스크 이미지를 다운로드 한다. 이 세가지 이미지는 제공한 CD에 있으며 TFTP 서버의 root 디렉토리에 복사해 주어야 함을 잊지 말자.

u-boot 이미지는 u-boot.bin, 커널 이미지는 zImage.bin, 램디스크 이미지는 ram2440-ram.gz 라 하자

u-boot 이미지를 TFTP를 통해 다운로드 한다.

```
MBA2440 #  tftp  30000000  u-boot.bin
```

```
TFTP from server 192.168.100.2; our IP address is 192.168.100.3
Filename 'mba2440/u-boot.bin'.
Load address: 0x30000000
Loading:-
done
Bytes transferred = 120492 (1d6ac hex)
```

커널 이미지를 다운로드 한다.

```
MBA2440 #  tftp  30080000  zImage.bin
TFTP from server 192.168.100.2; our IP address is 192.168.100.3
Filename 'mba2440/zImage.bin'.
Load address: 0x30080000
Loading:/
done
Bytes transferred = 746208 (b62e0 hex)
```

램디스크 이미지를 다운로드 한다.

```
MBA2440 #  tftp  30200000  ram2440-ram.gz
TFTP from server 192.168.100.2; our IP address is 192.168.100.5
Filename 'mba2440/ram2440.1.2.gz'.
Load address: 0x30200000
Loading:-
done
Bytes transferred = 2782866 (2a7692 hex)
```

NAND flash를 모두 erase한 후, 다운로드한 이미지를 한꺼번에 write한다.

```
MBA2440 #  nande  all
erase all blocks [0 ~ 2047]
Erasing block .../
Complete erasing block



MBA2440 #  nandw  t  30000000  0  600000
Write start point : block[0] page[0] size = 0x600000
Writing data .../
Write compelete(0x600000): from 0x30000000 SDRAM to 0x0(0) NAND flash
```

"bootrom linux" 명령을 수행하면 NAND flash에 적재되어 있는 커널 이미지와 램디스크 이미지를 메모리에 복사한 후, 커널을 수행하게 된다.

[그림 2.13]

[그림 2.13]는 NAND flash로부터 이미지들을 복사하는 모습이다.

bootrom 명령어 말고 **NAND flash read 명령어(nandr)로도 리눅스를 수행**할 수 있다.

먼저, 커널 이미지를 메모리에 적재한다.

```
MBA2440 #  nandr  t  30c00000  80000  180000
Read start point : block[32] page[0] size = 0x180000
Reading data ...-
Read compelete(0x180000): from 0x80000(32) NAND flash to 0x30c00000 SDRAM
```

nandr 명령에서

30c00000은 커널 이미지를 적재할 메모리의 주소이고,

80000은 커널 이미지가 위치한 NAND flash의 주소이며

180000은 복사하고자 하는 사이즈이다.

각, **숫자는 모두 16진수(hex) 단위**이다.

다음으로는 램디스크 이미지를 메모리에 적재한다.

```
MBA2440 #  nandr  t  30800000  200000  400000
Read start point : block[128] page[0] size = 0x400000
```

```
Reading data .../
Read compelete(0x400000): from 0x200000(128) NAND flash to 0x30800000 SDRAM
```

마지막으로 커널 이미지를 수행한다. 복사된 커널 이미지의 주소는 0x30c00000 이다.

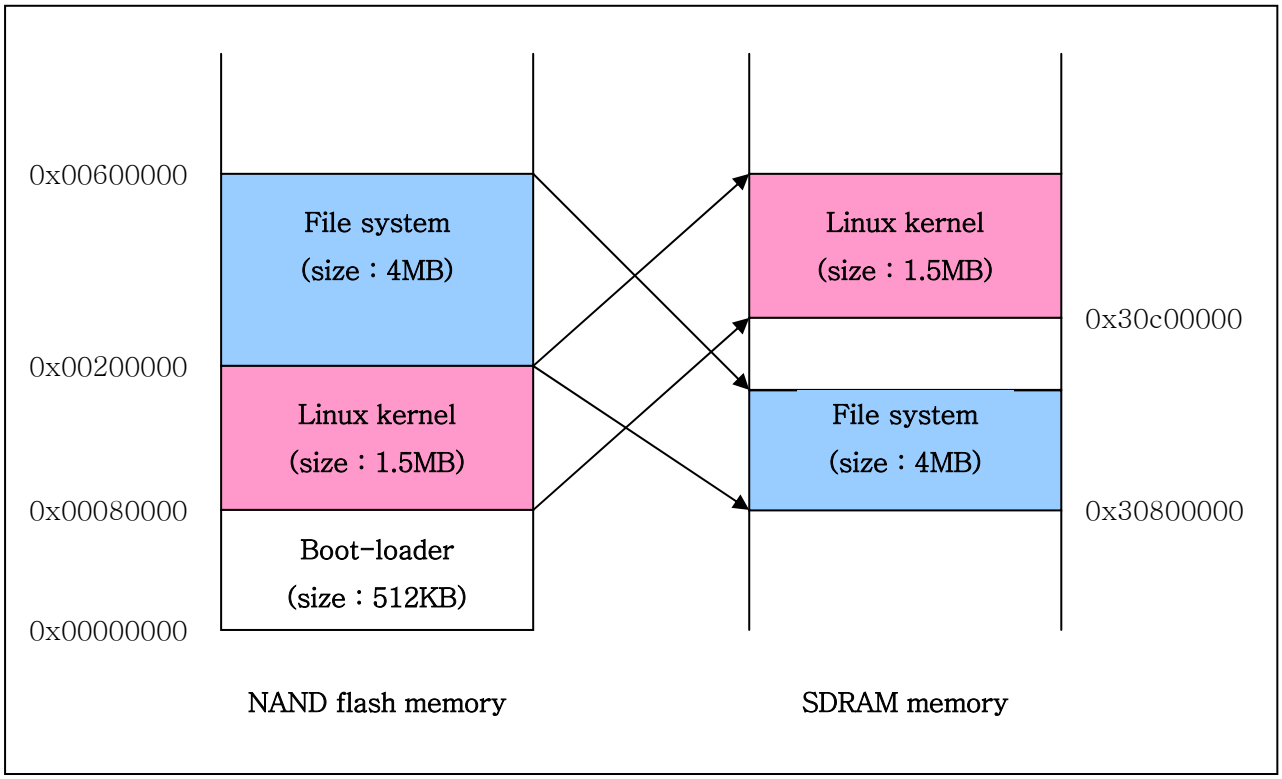
```
MBA2440 #   go   linux          (또는 go 30c00000)
Uncompressing Linux..... done, booting the kern.
Linux version 2.4.20_elfin-d1.5 (root@localhost.localdomain) (gcc version 2.95.3 20010315 (releaT
CPU: ARM/CIRRUS Arm920Tid(wb) revision 0
.....
```

```
iic_elfin_init: initialized iic-bus at 0xf4000000.
ELFIN UDA1341 audio driver initialized
NAND device: Manufacture ID: 0xec, Chip ID: 0x75 (Samsung NAND 32MB 3,3V)
Creating 3 MTD partitions on "NAND 32MB 3,3V":
0x00000000-0x00030000 : "NAND partition 0 : Bootloader"
0x00030000-0x00200000 : "NAND partition 1 : Kernel"
0x00200000-0x02000000 : "NAND partition 2"
usb.c: registered new driver hub
S3C2440 USBD Controller Core Initialized
USB Function Ethernet Driver Interface
NET4: Linux TCP/IP 1.0 for NET4.0
IP Protocols: ICMP, UDP, TCP, IGMP
IP: routing cache hash table of 512 buckets, 4Kbytes
TCP: Hash tables configured (established 2048 bind 2048)
IP-Config: No network devices available.
NET4: Unix domain sockets 1.0/SMP for Linux NET4.0.
NetWinder Floating Point Emulator V0.95 (c) 1998-1999 Rebel.com
RAMDISK: Compressed image found at block 0
Freeing initrd memory: 4096K
EXT2-fs warning: mounting unchecked fs, running e2fsck is recommended
VFS: Mounted root (ext2 filesystem).
Freeing init memory: 76K
INIT: version 2.74 booting
SIOCSIFADDR: No such device
eth0: unknown interface: No such device
SIOCSIFNETMASK: No such device
eth0: unknown interface: No such device
SIOCADDRT: NeI嵐근깁f55)ÿStarting system logger:  syslogd
Starting portmapper:  portmap
Starting network Starting network SIOCSIFADDR: No such device
eth0: unknown interface: No such device
SIOCSIFNETMASK: No such device
SIOCADDRT: No such device




+-------------------------------------+
|       Welcome to MBA2440  !!!       |
+-------------------------------------+

Linux login: root
[root@Linux /root]$
 CTRL-A Z for help |115200 8N1 | NOR | Minicom 2.00.0 | VT102 |      Offline
[영어][완성][두벌식]
```

[그림 2.14]

부팅 후 Linux login: 에 root를 입력하면 linux prompt가 뜨는 것을 볼 수 있다.

[그림 2.15]는 위 과정을 쉽게 이해할 수 있는 NAND flash와 SDRAM 메모리 간의 memory map이다.



[그림 2.15]

이제, 리눅스의 세계로 빠져 봅시다!!!!

3 개발 환경 구축

MBA2440을 이용하여 임베디드 시스템을 개발하는데 있어서 반드시 필요한 개발 환경들이 있다. 이러한 개발 환경들에는 어떤 것들이 있으며 어떻게 개발 환경을 구축하는지 알아 보는 것이 이 장의 목적이다.

이 장에서는 MBA2440을 제어하기 하기 위한 **firmware 개발 환경**, **임베디드 리눅스 개발 환경**, **WinCE 개발 환경**으로 크게 구분하여 다루고자 한다.

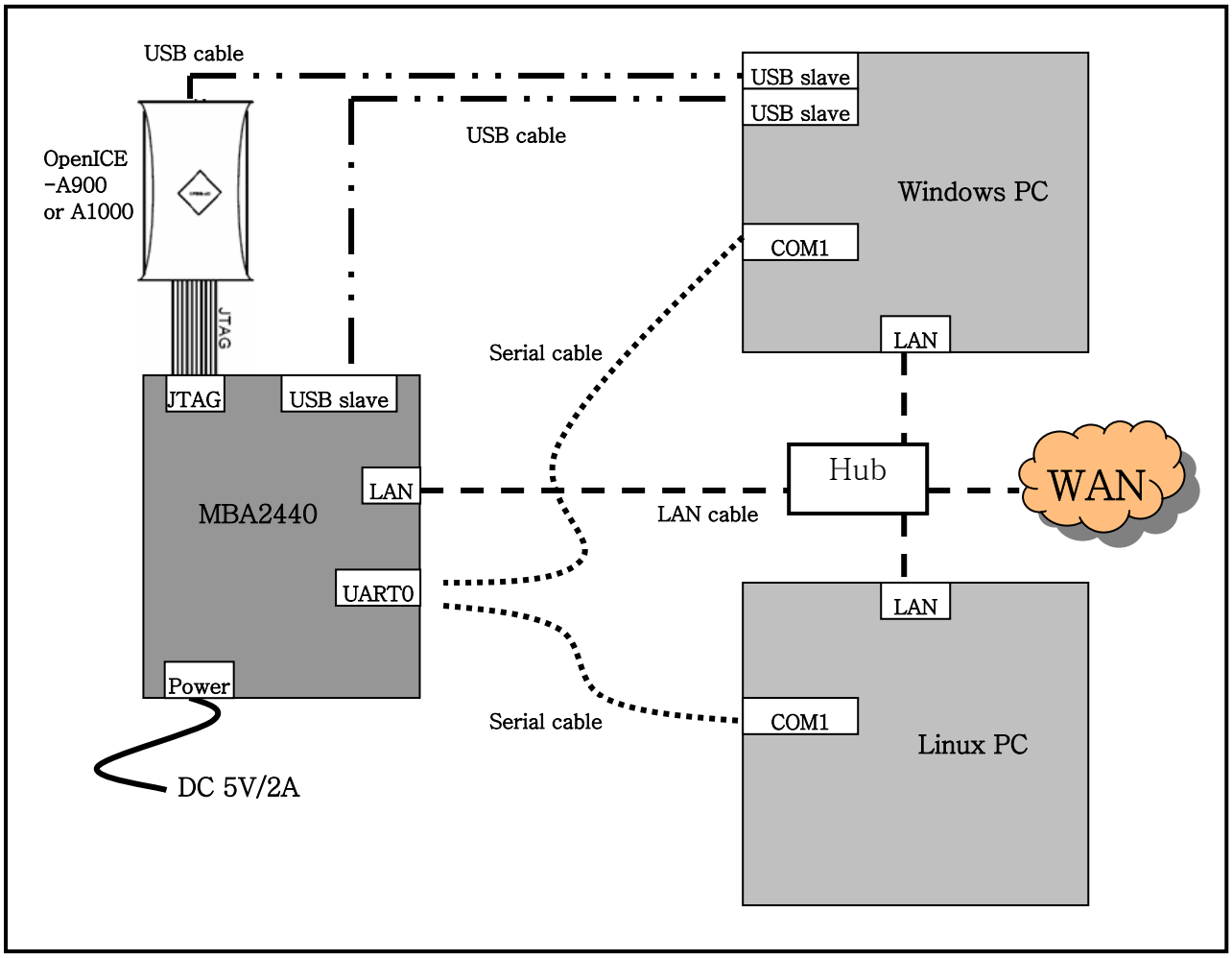
개발 환경을 구축하는 플랫폼은 크게 Windows OS와 Linux OS이다.

Linux OS상에서는 U-boot bootloader와 임베디드 리눅스를 개발하기 위한 환경을 구축하며,

Windows OS 상에서는 MBA2440 test program 개발 환경과 WinCE를 개발하기 위한 환경을 구축한다.

3.1 개발 환경 구축 예

이 문서에서는 개발 환경을 [그림 3.1]과 같이 구성하였다.



[그림 3.1] 개발 환경 구축 예

개발 환경을 구축함에 있어서 network 설정은 개발의 편의성을 위해서 중요한 부분이다.

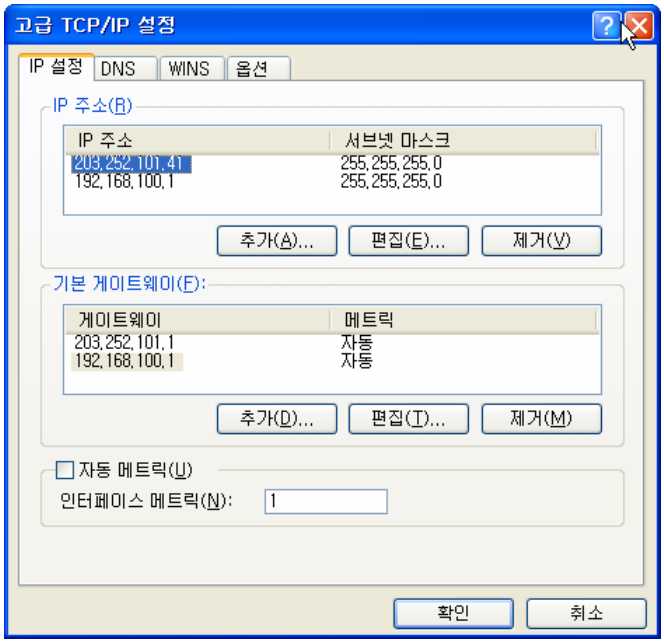
이 문서에서는 다음과 같은 사설 네트워크 주소를 할당하고 이에 맞춰서 모든 작업을 하게 될 것이다. 이 네트워크 주소는 개발자의 편의에 따라 수정이 가능한 것이다. 다만, 문서를 보는데 있어서 편의성을 위해 미리 정하고자 한다.

Windows가 설치된 데스크탑 PC	192.168.100.1
Linux가 설치된 데스크탑 PC	192.168.100.2
MBA2440 board	192.168.100.3

Windows나 Linux는 하나의 네트워크 카드에 자신이 기본적으로 사용하는 IP 주소 외에 또 다른 IP를 부여할 수 있다. 물론, 하나의 네트워크 카드에 주어지는 IP 주소는 서로 다른 대역이어야 한다.

☞ Windows 상에서는 자신의 네트워크 카드에 해당하는 "로컬 영역 연결"의 등록 정보에서 고급 버튼을 누르면 "고급 TCP/IP 설정"창이 뜬다.

이 "고급 TCP/IP 설정"창에서 사설 IP 주소와 기본 게이트웨이 주소를 추가해 준다. (추가 버튼을 누른다) 이 문서에서는 Windows PC에 192.168.100.1을 추가해 주었다.



[그림 3.2]

☞ Linux 상에서는 먼저 /etc/sysconfig/network-scripts/ 밑에 ifcfg-eth0:1 이라는 이름의 파일을 하나 생성한다.

생성한 후 파일 안에 다음과 같은 내용을 적는 후 저장하고 빠져 나온다.

```
DEVICE=eth0:1
ONBOOT=yes
BOOTPROTO=none
IPADDR=192.168.100.2
NETMASK=255.255.255.0
GATEWAY=192.168.100.1
TYPE=Ethernet
USERCTL=no
PEERDNS=no
NETWORK=192.168.100.0
BROADCAST=192.168.100.255
```

다음 명령어를 수행하여 network을 재동작 시킨다.

```
[root@localhost]#  /etc/rc.d/init.d/network  restart
```

ifconfig 명령을 통해 하나의 네트워크 카드에 서로 다른 대역의 두 개의 IP가 할당되었음을 확인할 수 있다.

```
[root@localhost]#  ifconfig
eth0      Link encap:Ethernet  HWaddr 00:20:ED:35:DD:CE
          inet addr:203.252.101.42  Bcast: 203.252.101.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1679305 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1757078 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:464794008 (443.2 Mb)  TX bytes:947711009 (903.8 Mb)
          Interrupt:11 Base address:0xa000 Memory:e2000000-e2000038

eth0:1    Link encap:Ethernet  HWaddr 00:20:ED:35:DD:CE
          inet addr:192.168.100.2  Bcast:192.168.100.255  Mask:255.255.255.0
          UP BROADCAST RUNNING MULTICAST  MTU:1500  Metric:1
          RX packets:1679305 errors:0 dropped:0 overruns:0 frame:0
          TX packets:1757078 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:100
          RX bytes:464794008 (443.2 Mb)  TX bytes:947711009 (903.8 Mb)
          Interrupt:11 Base address:0xa000 Memory:e2000000-e2000038

lo        Link encap:Local Loopback
          inet addr:127.0.0.1  Mask:255.0.0.0
          UP LOOPBACK RUNNING  MTU:16436  Metric:1
          RX packets:378001 errors:0 dropped:0 overruns:0 frame:0
          TX packets:378001 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:0
          RX bytes:110928505 (105.7 Mb)  TX bytes:110928505 (105.7 Mb)
```

MBA2440의 IP설정은 u-boot, eboot, embedded linux, WinCE 컴파일시에 설정하거나 구동 후 설정할 수 있다.

3.2 Firmware 개발 환경

MBA2440을 위한 firmware들 중 u-boot와 eboot를 제외한 **MBA240 test program, nboot는 ADS1.2를 기반으로 개발된 firmware들**이다. 이 firmware들을 수정하게 될 경우에는 ADS 1.2를 설치하고 이를 통해 수정한 후 재 컴파일 해야 한다.

ADS 1.2는 ARM사에서 개발한 ARM core 전용 Development Suite version 1,2이다.

이 개발 툴은 Windows, SunSPARC workstation, HP workstation, Redhat linux 플랫폼상에서 사용 가능하다.

자세한 사항은 ARM사의 홈페이지를 참조하기 바란다.

ADS 1.2는 직접 구매하여 설치한다. 설치 방법에 대해서는 다루지 않는다. 구매 후, 직접 매뉴얼을 참조하여 설치하기 바란다.

이 문서에서는 MBA240 test program, nboot에 관련된 사항을 설명할 때에는 Windows OS 상에서 ADS 1.2를 설치한 상황에서 모든 것을 설명할 것이다.

3.3 임베디드 리눅스 개발 환경

이 장에서는 임베디드 리눅스를 개발 하기 위한 환경을 구축하는 방법을 설명한다. **3.3장에서 설명하는 모든 개발 환경 구축 내용은 RedHat 9.0 리눅스 OS를 full option으로 설치한 데스크탑 PC상에서 이루어지는 작업이며 root 계정으로 수행하는 작업**이다. 개발자들도 이와 같은 환경에서 이 문서를 참조하기를 바란다.

이 장에서 다루는 임베디드 리눅스 개발 환경을 구축함으로써, 우리는 MBA2440을 위한 u-boot 부트로더와 임베디드 리눅스 시스템을 개발 할 수 있다.

개발 환경은 다음과 같이 구분할 수 있다.

◈ **크로스 컴파일러 설치**

: MBA2440을 위한 리눅스 소스를 컴파일 할 수 있는 컴파일러를 설치

◈ **tftp 서버 설치**

: trivial FTP로서 MBA2440 부트로더 상에서 파일 수신을 위한 서버를 설치

◈ **NFS 서버 설치**

: Network File System을 사용하기 위한 서버 설치

◈ **serial console(minicom) 설정**

: MBA2440을 모니터링 할 수 있는 리눅스의 serial 통신 emulator 설정

3.3.1 크로스 컴파일러 설치

3.3.1.1 크로스 컴파일러란?

크로스 컴파일러란, 데스크탑에 설치된 리눅스 OS에서 제공하는 컴파일러 외에 MBA2440을 위해 수정된 리눅스 소스를 컴파일 하기 위한 컴파일러를 말한다.

일반 데스크탑에 설치된 리눅스 OS 예를 들면, RedHat 9.0에서 제공하는 gcc 컴파일러는 소스를 컴파일 하게 되면 데스크탑에서 동작할 수 있는 실행 이미지를 생성하게 된다. 이 실행 이미지는 MBA2440에서는 동작하지 않는다. 데스크탑에 있는 CPU가 이해할 수 있도록 컴파일 했기 때문에 MBA2440의 CPU인 S3C2440은 이 실행 파일을 이해하지 못한다.

그러므로, 컴파일 할 때 당연히 MBA2440이 이해할 수 있도록 컴파일 하여야 MBA2440에서 동작할 수 있다. 이를 위한 컴파일러를 크로스 컴파일러라 말한다.

비단, MBA2440 보드 뿐만 아니라 많은 임베디드 시스템을 위한 컴파일러나 platform builder등도 크로스 컴파일러라 할 수 있다.

3.3.1.2 크로스 컴파일러 gcc-2.95.3 설치

제공한 CD에 linux/toolchain 디렉토리에 크로스 컴파일러 두 가지가 있다.

그 중, 한 가지가 arm-gcc-2.95.3.tar.bz2 이다.

이 크로스 컴파일러는 리눅스 2.4 버전 이하의 소스를 컴파일 할 때 사용하는 크로스 컴파일러 이다.

먼저, CD에서 압축된 크로스 컴파일러를 가져온다.

```
[root@localhost]#  pwd
/root
[root@localhost]#  cp  /mnt/cdrom/toolchain/arm-gcc-2.95.3.tar.bz2  ./
[root@localhost]#  ls
arm-gcc-2.95.3.tar.bz2
```

압축된 크로스 컴파일러의 압축을 해제하면 2.95.3 이라는 디렉토리가 생성된다.

```
[root@localhost]#  tar  cjf  arm-gcc-2.95.3.tar.bz2
[root@localhost]#  ls
2.95.3  arm-gcc-2.95.3.tar.bz2
```

/usr/local/ 디렉토리 밑에 arm 이라는 디렉토리를 만든 후, 생성된 2.95.3 디렉토리를 이동 시킨다.

```
[root@localhost]#  mkdir  /usr/local/arm
[root@localhost]#  ls  /usr/local
arm  bin  etc  games  include  lib  libexec  man  sbin  share  src
[root@localhost]#  mv  2.95.3  /usr/local/arm
[root@localhost]#  ls  /usr/local/arm
2.95.3
```

이제는 PATH 환경 변수에 이 크로스 컴파일러의 full path를 넣어 준다.
PATH 환경 변수 설정은 /root/.bash_profile 파일에서 설정해 준다

```
[root@localhost]#  vi  /root/.bash_profile
```

파일을 열면, 아래와 같은 내용이 보인다. 그 내용에 굵은 글씨체로 되어 있는 내용을 추가해 준다.

```
# .bash_profile
# Get the aliases and functions
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/bin
PATH=$PATH:/usr/local/arm/2.95.3/bin/

BASH_ENV=$HOME/.bashrc
USERNAME="root"
```

```
export USERNAME BASH_ENV PATH
```

추가한 후, 저장하고 빠져 나오고, 그 다음으로는 변경된 환경 변수를 적용시킨다.

```
[root@localhost]#  source  /root/.bash_profile
```

적용 되었는지 확인하기 위해 arm-linux- 를 입력한 후 tab 키를 쳐본다. 아래와 같이 화면에 뜨면 정상적으로 환경 변수가 적용이 된 것이다.

```
[root@localhost]#  arm-linux-
arm-linux-addr2line   arm-linux-g77      arm-linux-objcopy    arm-linux-strings
arm-linux-ar          arm-linux-gasp     arm-linux-objdump    arm-linux-strip
arm-linux-as          arm-linux-gcc      arm-linux-protoize   arm-linux-unprotoize
arm-linux-c++         arm-linux-gcj      arm-linux-ranlib
arm-linux-c++filt     arm-linux-ld       arm-linux-readelf
arm-linux-g++         arm-linux-nm       arm-linux-size
```

3.3.1.3 크로스 컴파일러 gcc-3.3 설치

CD에서 제공하는 또 다른 linux용 크로스 컴파일러는 **arm-gcc3.3.tar.bz2** 이다. 이 크로스 컴파일러는 Linux 2.6버전을 컴파일 하기 위한 것이다.

먼저, CD에서 압축된 크로스 컴파일러를 가져온다.

```
[root@localhost]#  pwd
/root
[root@localhost]#  cp  /mnt/cdrom/toolchain/arm-gcc3.3.tar.bz2  ./
[root@localhost]#  ls
arm-gcc3.3.tar.bz2
```

압축된 크로스 컴파일러의 압축을 해제하면 **arm-gcc3.3** 이라는 디렉토리가 생성된다.

```
[root@localhost]#  tar  cjf  arm-gcc3.3.tar.bz2
[root@localhost]#  ls
arm-gcc3.3  arm-gcc3.3.tar.bz2
```

/usr/local/ 디렉토리 밑에 arm 이라는 디렉토리를 만든다. 이미 생성되어 있다면 만들 필요는 없다.
arm 디렉토리에 arm-gcc3.3 디렉토리를 이동 시킨다.

```
[root@localhost]#  mkdir  /usr/local/arm
[root@localhost]#  ls  /usr/local
arm  bin  etc  games  include  lib  libexec  man  sbin  share  src
```

```
[root@localhost]#  mv  arm-gcc3.3  /usr/local/arm
[root@localhost]#  ls /usr/local/arm
arm-gcc3.3
```

이제는 PATH 환경 변수에 이 크로스 컴파일러의 full path를 넣어 준다.

PATH 환경 변수 설정은 /root/.bash_profile 파일에서 설정해 준다

```
[root@localhost]#  vi  /root/.bash_profile
```

파일을 열면, 아래와 같은 내용이 보인다. 그 내용에 굵은 글씨체로 되어 있는 내용을 추가해 준다.

```
# .bash_profile

# Get the aliases and functions
if [ -f ~/.bashrc ]; then
    . ~/.bashrc
fi

# User specific environment and startup programs

PATH=$PATH:$HOME/bin
PATH=$PATH:/usr/local/arm/2.95.3/bin/
PATH=$PATH:/usr/local/arm/arm-gcc3.3/bin/

BASH_ENV=$HOME/.bashrc
USERNAME="root"

export USERNAME BASH_ENV PATH
```

추가한 후, 저장하고 빠져 나오고, 그 다음으로는 변경된 환경 변수를 적용시킨다.

```
[root@localhost]#  source  /root/.bash_profile
```

적용 되었는지 확인하기 위해 armv5l-linux- 를 입력한 후 tab 키를 쳐본다. 아래와 같이 화면에 뜨면 정상적으로 적용이 된 것이다.

```
[root@localhost]#  armv5l-linux-
armv5l-linux-addr2line   armv5l-linux-cpp       armv5l-linux-gcov      armv5l-linux-ranlib
armv5l-linux-ar          armv5l-linux-g++       armv5l-linux-ld        armv5l-linux-readelf
armv5l-linux-as          armv5l-linux-gcc       armv5l-linux-nm        armv5l-linux-size
armv5l-linux-c++         armv5l-linux-gcc-3.3   armv5l-linux-objcopy   armv5l-linux-strings
armv5l-linux-c++filt     armv5l-linux-gccbug    armv5l-linux-objdump   armv5l-linux-strip
```

3.3.2 리눅스 상에서 TFTP 서버 설치

3.3.2.1 TFTP란?

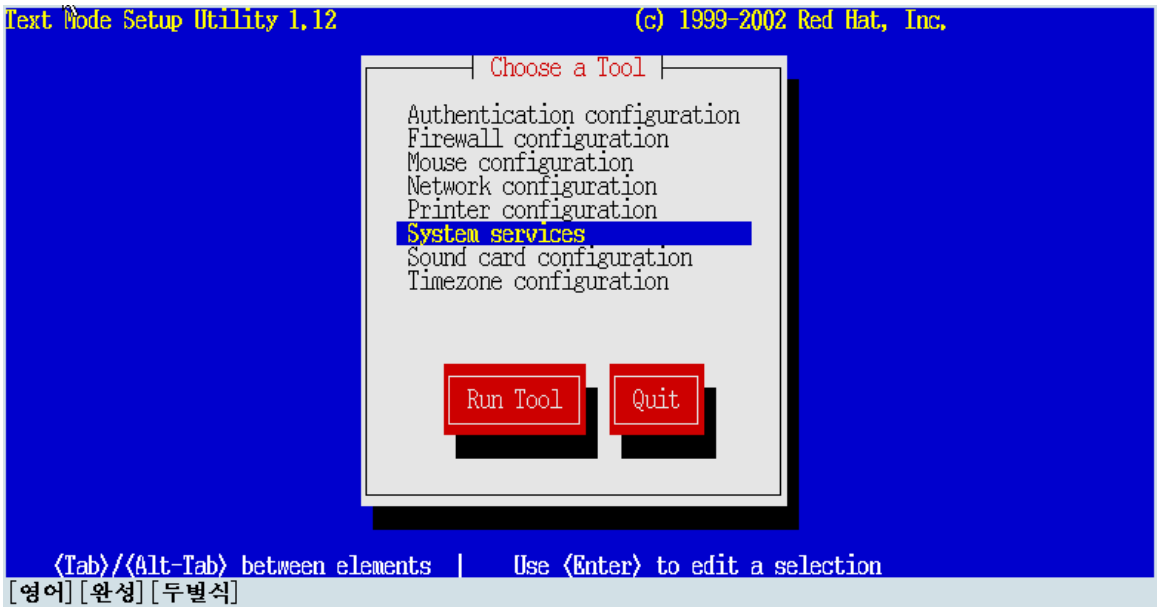
TFTP란 FTP 프로토콜의 하나로서 파일을 송수신 하기 위한 UDP 기반의 프로토콜이다.

임베디드 시스템을 개발하는 과정에서 MBA2440 u-boot에서 리눅스의 커널이나 파일 시스템, WinCE 이미지같이 큰 데이터를 PC로부터 받아 와야 할 경우가 빈번하게 발생하게 되는데 이 경우 아주 유용하게 사용할 수 있는 네트워크 프로토콜이다.

이 프로토콜을 기반으로 서버로 동작하는 TFTP 서버를 설치한다.

3.3.2.2 TFTP 서버 설치

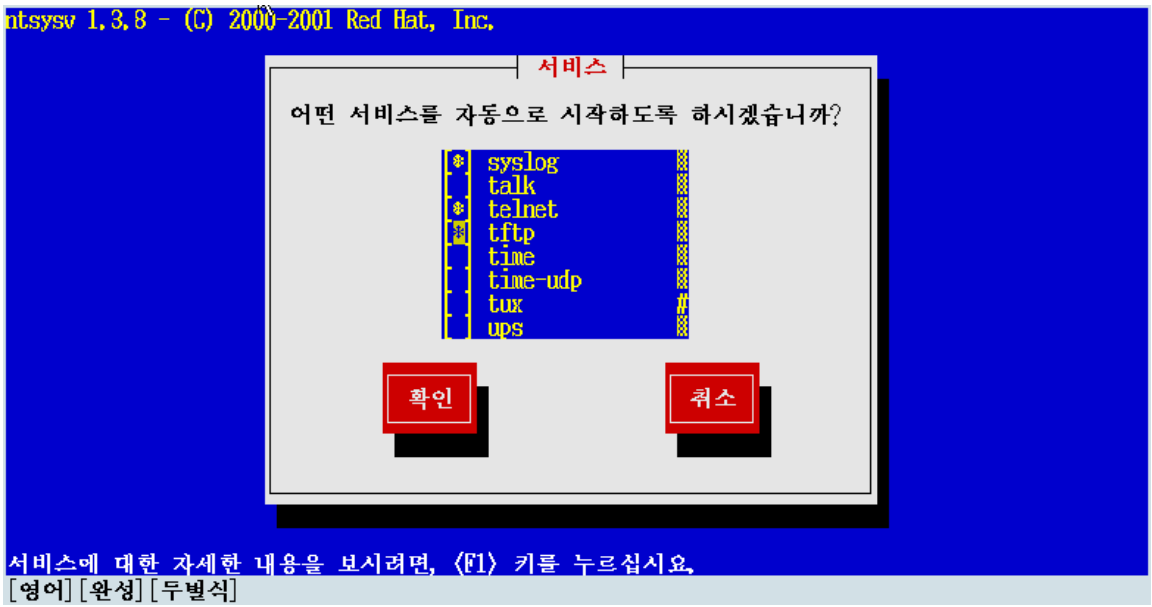
RedHat 9.0을 full option으로 설치했을 경우에는 TFTP가 설치되어 있기 때문에 사용하기만 하면 된다. 리눅스 PC에서 콘솔 창을 띄운 후, setup 명령을 입력하면 [그림 3.3]과 같이 text mode의 utility setup 창이 뜨고 여러 개의 메뉴가 뜨게 되는데, 이 중, System services 를 선택하고 enter를 치면 사용하고자 하는 서비스들을 선택할 수 있는 창이 뜬다.



[그림 3.3]

이 서비스 메뉴 중에서 TFTP를 찾아 선택한 후, "*"표가 체크되어 있지 않다면 [그림 3.4]와 같이 스페이스 바를 눌러 "*"표를 체크한다. 다음으로는 tab키를 눌러 '확인'으로 이동하고 enter 키를 쳐서 설정을 적용한다.

[그림 3.3]인 이전 메뉴로 돌아간 것을 볼 수 있는데, tab키를 눌러 'Quit' 으로 이동하고 enter키를 치고 빠져 나온다.



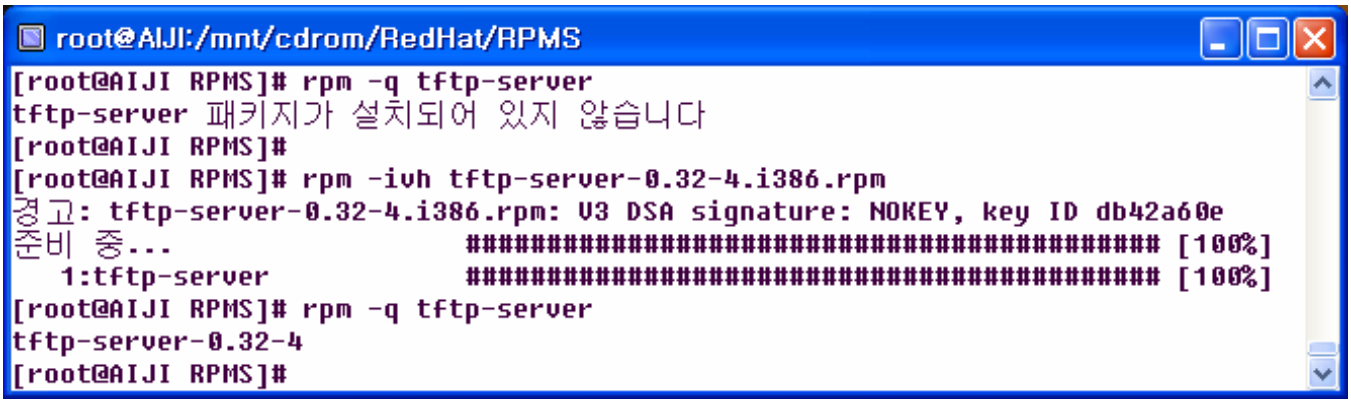
[그림 3.4]

설치되어 있지 않을 경우에는 RedHat CD에서 rpm을 찾아 설치하면 된다.

RedHat CD는 3장으로 구성되어 있는데, 이 CD에 있는 파일들을 검색하여

tftp-server-0.32-4.i386.rpm

을 찾아 /root 디렉토리에 복사한 후, rpm 명령으로 설치한다.

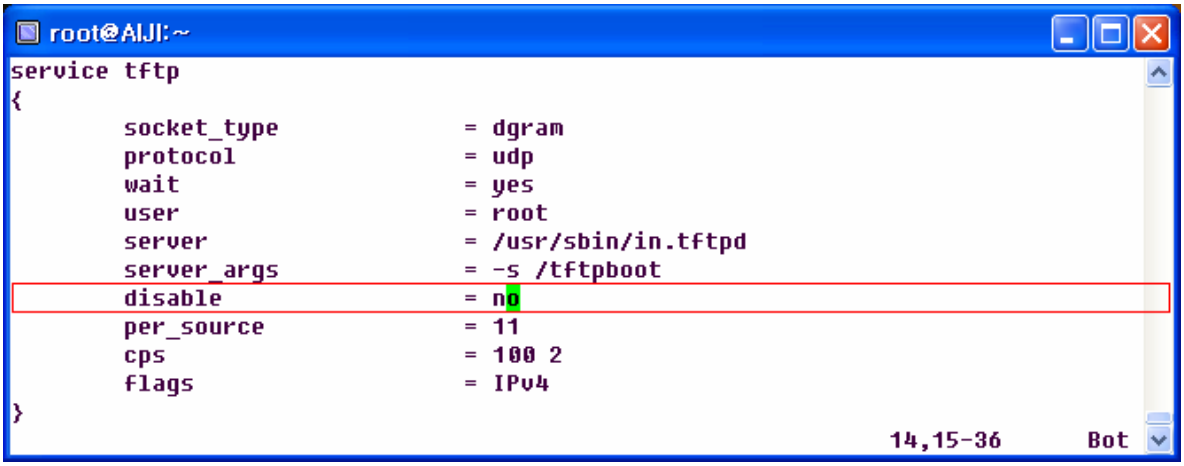


[그림 3.5]

3.3.2.3 TFTP 설정

/etc/xinetd.d/tftp 파일에서 수정해야 할 사항들이 있다.

```
[root@localhost]#  vi  /etc/xinetd.d/tftp
```



[그림 3.6]

[그림 3.6]과 같이 disable 을 no로 설정한다.

server_args 는 TFTP서버의 root 디렉토리를 설정하는 부분이다. **이 디렉토리 안에 자신이 전송하고자 하는 파일이나 이미지등을 넣어두어야 TFTP로 파일을 전송할 수 있음을 기억**하기 바란다.

Root 디렉토리는 개발자의 필요에 따라 원하는 위치의 디렉토리의 절대 경로를 /tftpboot 대신에 입력해 주면 된다.(ex: -s /aiji)

3.3.2.4 xinetd 재시작

TFTP 서비스를 사용 가능하도록 xinetd 데몬을 재시작 한다.

[그림 3.7]

3.3.3 NFS 설정

3.3.3.1 NFS란?

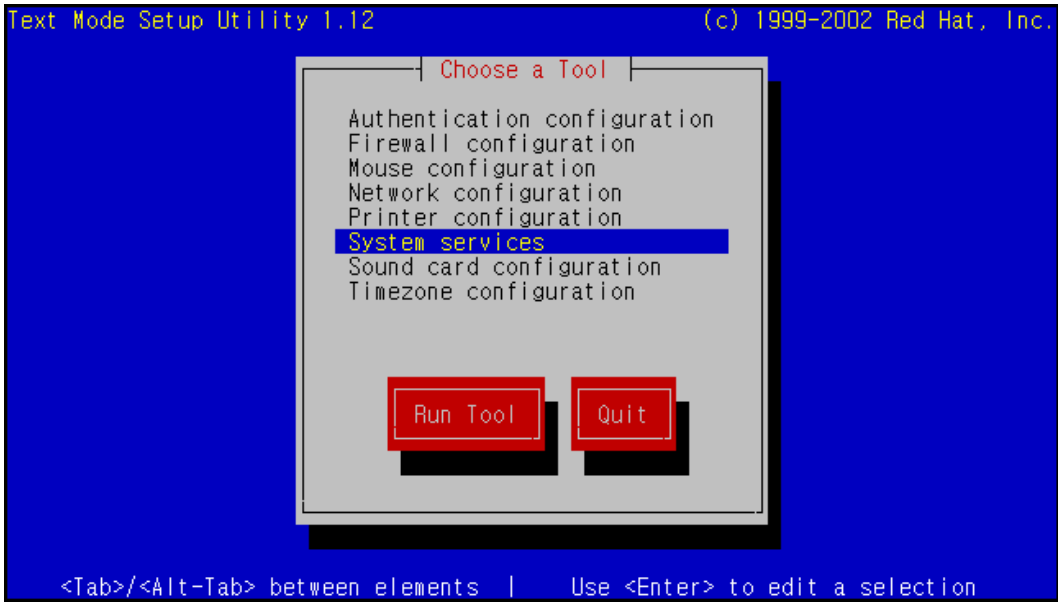
NFS란 파일 시스템 중의 하나로서 RPC를 이용하여 원격 호스트의 디렉토리를 마치 유저 자신의 로컬 디렉토리인 것처럼 사용할 수 있게 해주는 파일 시스템이다.

이 파일 시스템은 원격 호스트 상에서 NFS를 지원해 주어야 할 뿐만 아니라, 유저가 사용하는 시스템에서도 지원해 주어야 한다.

3.3.3.2 NFS 서비스

이 NFS 역시 RedHat 9.0을 full option으로 설치했을 경우, setup 명령을 통해 서비스를 사용할 수 있다.

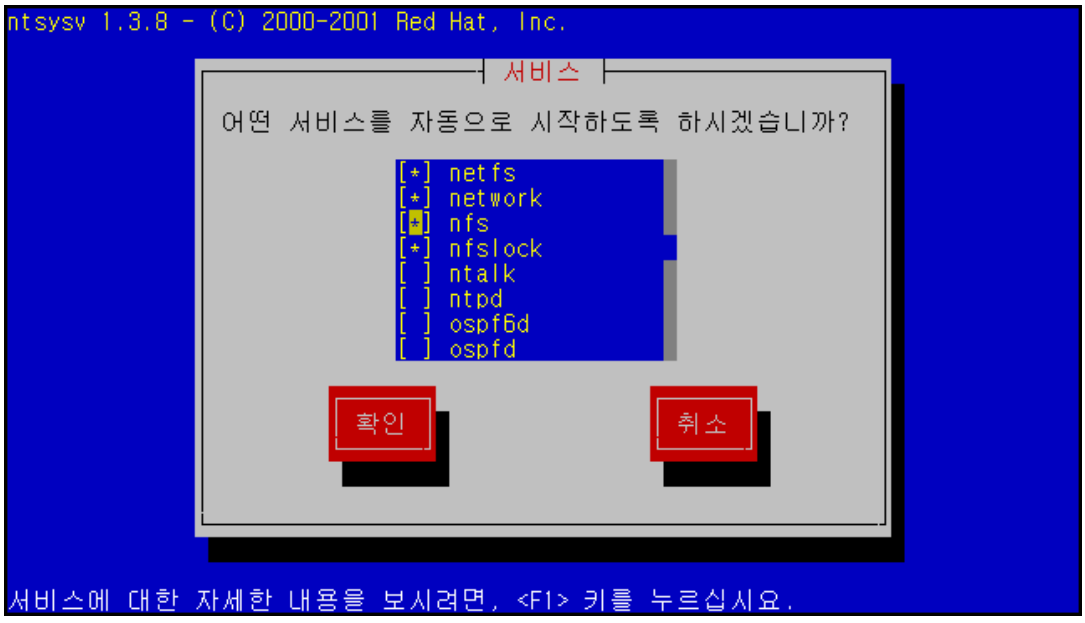
TFTP의 경우와 마찬가지로 [그림 3.8]과 같이 하면 된다.



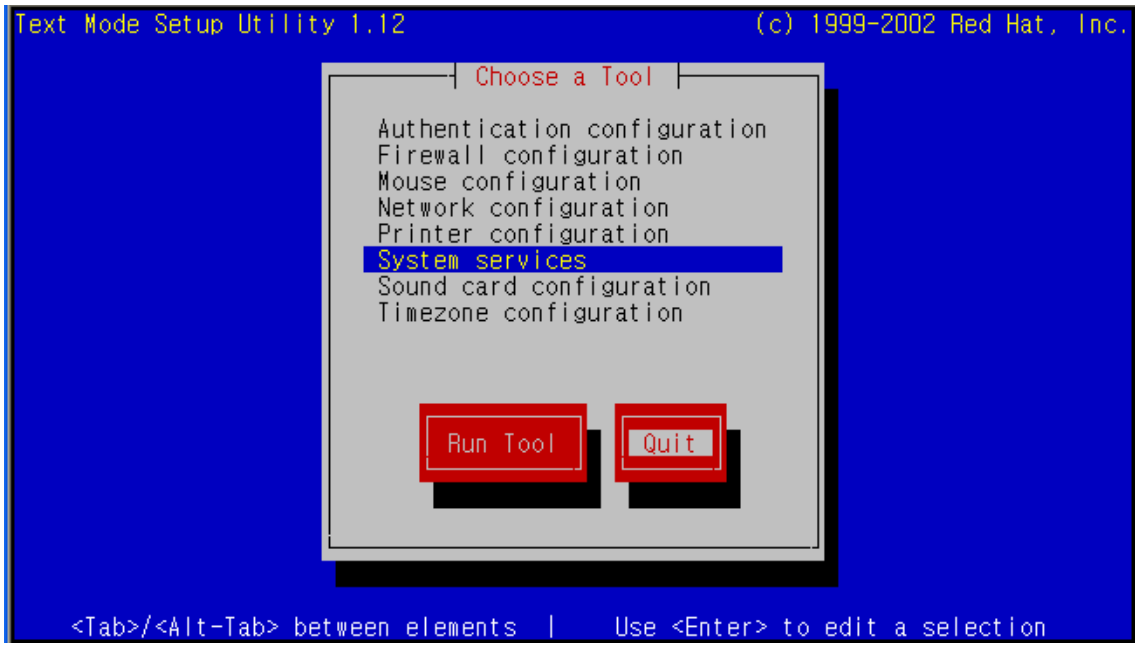
[그림 3.8]

리눅스상에서 터미널 창 하나를 띄우고 setup을 실행하면 위와 같은 창이 뜬다. NFS 서비스에 대한 지원을 여기서 해주게 된다. System services로 이동 후 엔터키를 누른다. 그러면 여러 시스템 서비스 지원을 위한 창으로 이동하게 된다.

창 이동 후 여러 서비스가 나타나는데 이 중 nfs 와 nfslock을 선택해 준다. 선택은 스페이스 바를 누르면 * 표시가 나타났다 없다 졌다 한다. * 가 생기면 선택된 것이다. 확인 후 빠져나오면 된다.



[그림 3.9]



[그림 3.10]

3.3.3.3 NFS 설정

NFS로 공유하고자 하는 디렉토리를 하나 만든다. 여기서는 nfsroot 라는 디렉토리를 최상위 디렉토리에(“/”)에 하나 만들고 설명할 것이다.

```
[root@localhost]#  mkdir  /nfsroot
[root@localhost]#  cd  /
[root@localhost]#  chmod  777  /nfsroot
[root@localhost]#  chown  nobody  /nfsroot
[root@localhost]#  chgrp  nobody  /nfsroot
```

다음으로는 /etc/exports 파일을 수정한다.

```
[root@localhost]#  vi  /etc/exports
```

```
/nfsroot                localhost(rw,no_root_squash)
/nfsroot                192.168.100.0/24(rw, no_root_squash)
```

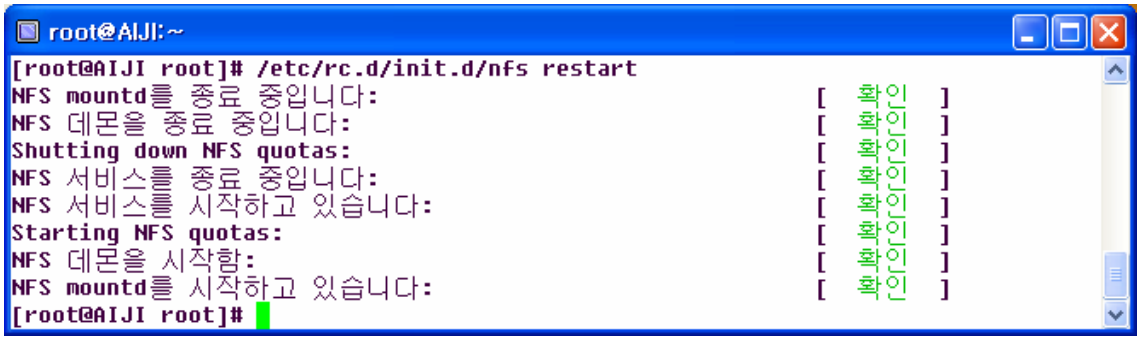
/nfsroot는 앞서 말한 공유하고자 하는 디렉토리명이다.
192.168.100.0/24 는 192.168.100 대역의 IP주소를 갖는 모든 컴퓨터는 접근을 허용한다는 의미이다.
()안의 내용에 대한 설명은 다음과 같다.

Option name	Description
ro	Read only
rw	Read/Write 가능
no_root_squash	root의 자격으로 파일 시스템 접근하도록 마운트

root_squash	root 자격으로 접근하면 anonymous UID/GID로 접근 허가
insecure	암호 인증 하지 않음

환경 설정이 끝나고 나면 설정을 확인하고 nfs 데몬을 다시 시작한다.

[그림 3.11]



[그림 3.12]

3.3.3.4 NFS 동작 확인

NFS가 제대로 동작하는지 확인 해보자.

MBA2440을 linux로 부팅한다. 부팅하는 방법은 2.4 리눅스 부팅하기를 참조한다.

부팅이 완료되었다면, minicom이나 DNW 콘솔 창을 통해서 리눅스 프롬프트를 볼 수 있다.

리눅스 PC가 NFS를 지원하도록 제대로 설정하였다면, 콘솔 창에서 다음과 같이 수행해 본다.

```
[root@localhost root]#  mount  -t  nfs  192.168.100.2:/nfsroot  /mnt/nfs/
```

192.168.100.2는 리눅스 PC의 IP주소이다.

:/nfsroot 는 리눅스 PC에서 NFS의 root 디렉토리의 절대 경로 주소이다.

/mnt/nfs/ 은 MBA2440 에서 동작하는 리눅스 파일 시스템에 존재하는 디렉토리로서, 이 디렉토리에 리눅스 PC의 /nfsroot 디렉토리가 서로 네트워크를 통해 마운트 되는 것이다.

리눅스 PC상에서 /nfsroot 디렉토리를 ls 명령으로 본 내용과 MBA2440 보드의 콘솔창에서(minicom 이나 DNW) /mnt/nfs 디렉토리를 ls 명령으로 본 내용이 동일함을 확인할 수 있다.

```
[root@localhost root]# cd /nfsroot
[root@localhost nfsroot]# ls
ram2440.gz  root-mba2440.cramfs  zImage-cramfs.bin  zImage.bin
[root@localhost nfsroot]#
```

[영어][완성][두벌식]

[그림 3.13] 리눅스 PC의 /nfsroot 디렉토리를 본 모습

```
[root@Linux /root]$mount -t nfs 192.168.100.2:/nfsroot /mnt/nfs/
nfs warning: mount version older than kernel
[root@Linux /root]$cd /mnt/nfs/
[root@Linux nfs]$ls
zImage.bin            ram2440.gz            zImage-cramfs.bin
root-mba2440.cramfs
[root@Linux nfs]$
```

[영어][완성][두벌식]

[그림 3.14] miniom상에서 /mnt/nfs 디렉토리를 본 모습

MBA2440 보드의 콘솔창에서 /mnt/nfs 에 있는 파일을 /root 디렉토리로 복사하면 네트워크를 통해 그대로 복사되며, 그 반대도 가능하다.

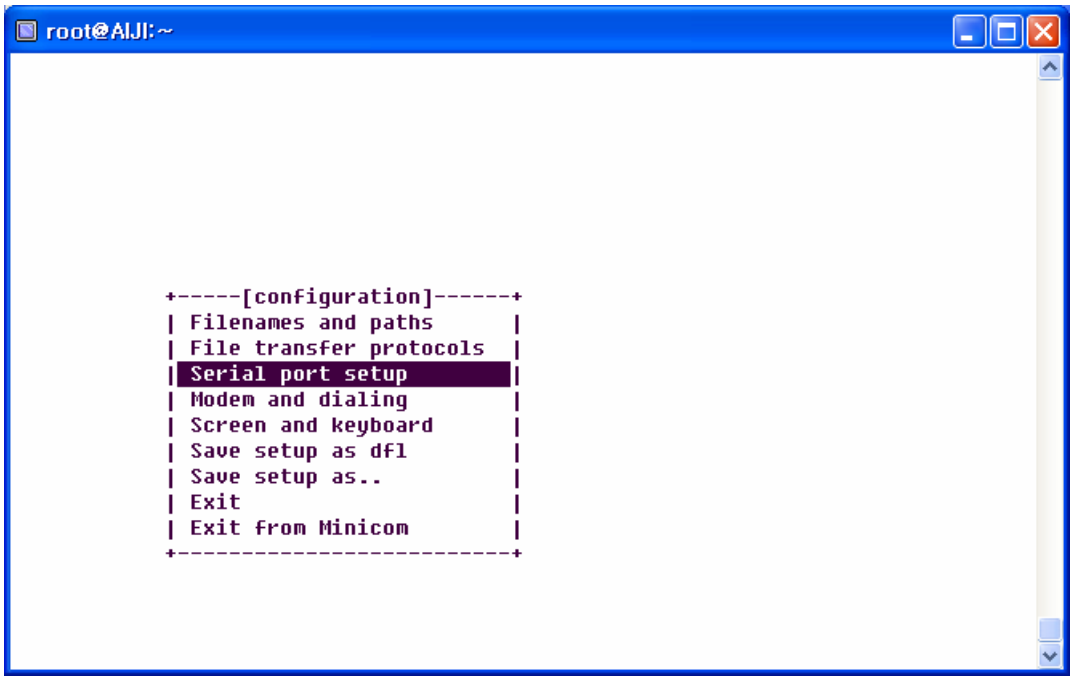
직접 테스트 해보기 바란다.

3.3.4 Minicom 설정

리눅스의 Minicom은 윈도우의 하이퍼터미널이나 새롬 데이터 맨과 같은 시리얼 통신 에뮬레이터로서 MBA2440의 모니터 역할을 하는 유틸리티이다.

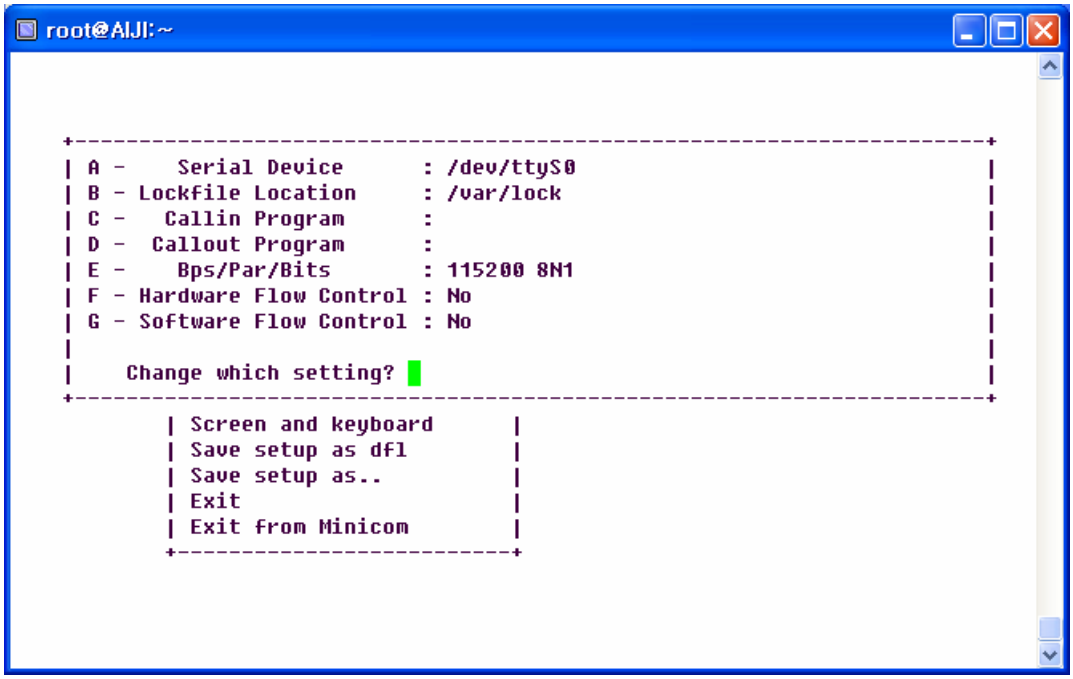
Minicom 설정을 위한 명령으로 다음의 명령을 수행한다.

```
[root@localhost]#  minicom -s
```



[그림 3.15]

[그림 3.15]에서 보는 바와 같이 Serial port setup을 선택한다. PC의 COM1일 경우에는 /dev/ttyS0, COM2일 경우에는 /dev/ttyS1이다.

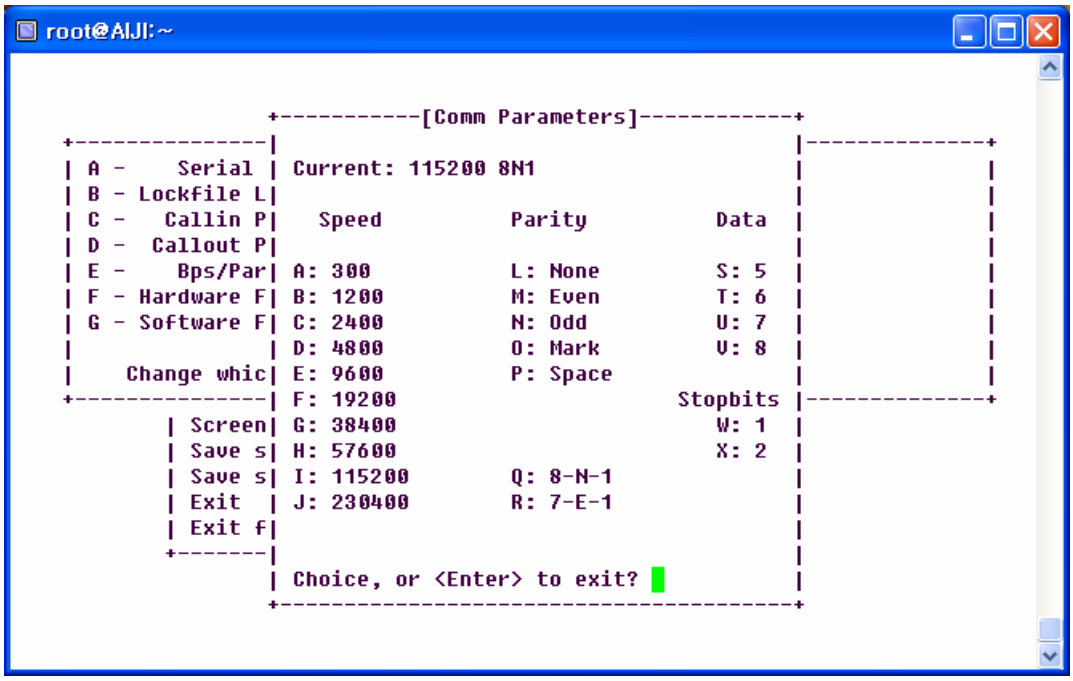


[그림 3.16]

Parameters를 설정한다.

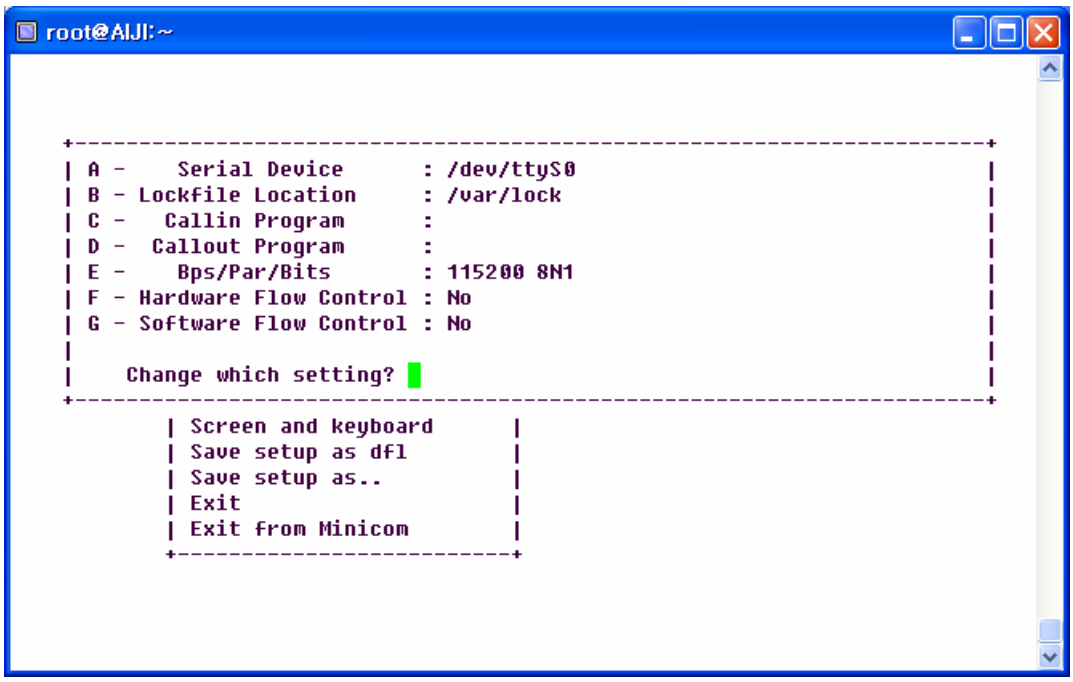
설정해야 할 값들 : [Baudrate(speed) : 115200 Parity : none Data bits : 8 Stop bits : 1]

E를 누르면 [그림 3.17]과 같은 모습을 볼 수 있다.



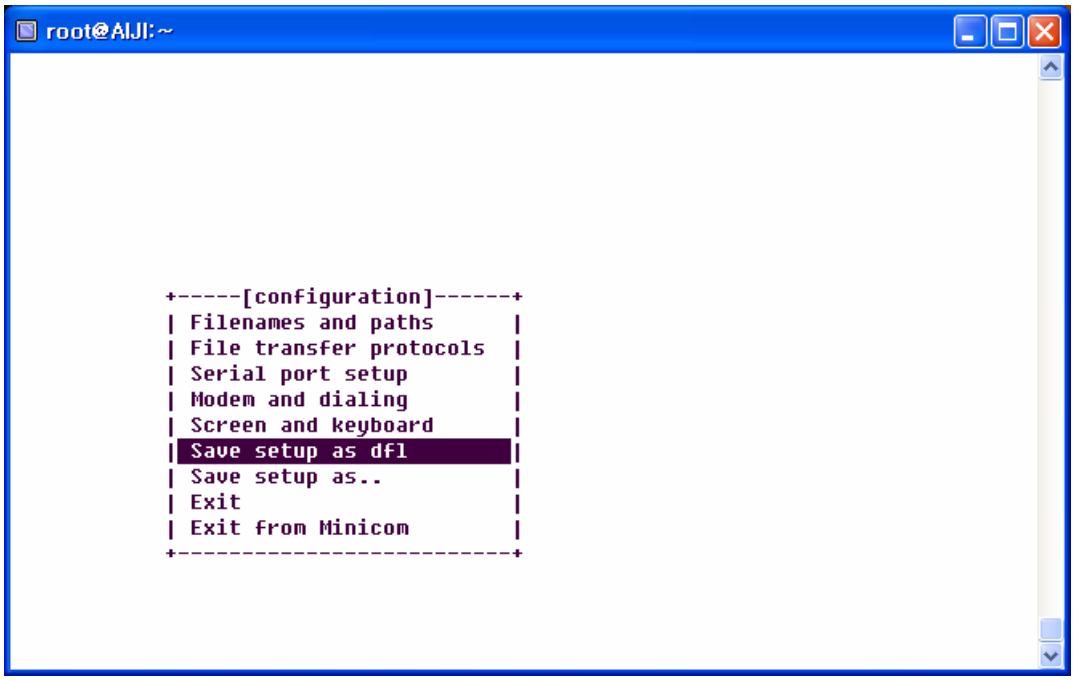
[그림 3.17]

I (115200 선택)와 Q (8-N-1 선택)를 누르고 엔터를 친다.



[그림 3.18]

다음으로 F와 G를 눌러 Hardware Flow Control과 Software Flow Control을 No로 설정한다.



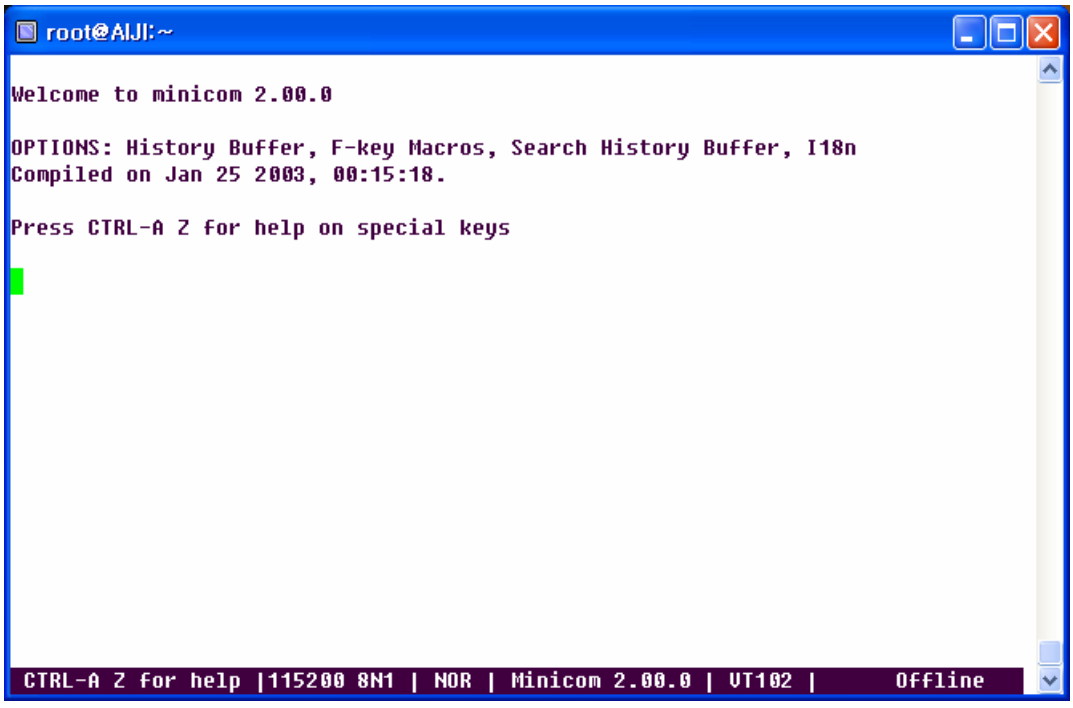
[그림 3.19]

설정을 저장한다. Save setup as df1을 선택하고 enter를 친다. 저장이 완료되면 Exit from mincom을 선택하고 enter를 친 후, 빠져 나온다.

이제부터는 minicom 이라는 명령만 입력하면 설정된 값대로 minicom이 동작하게 된다.

Minicom을 다시 실행해 보자.

```
[root@localhost]#   minicom
```

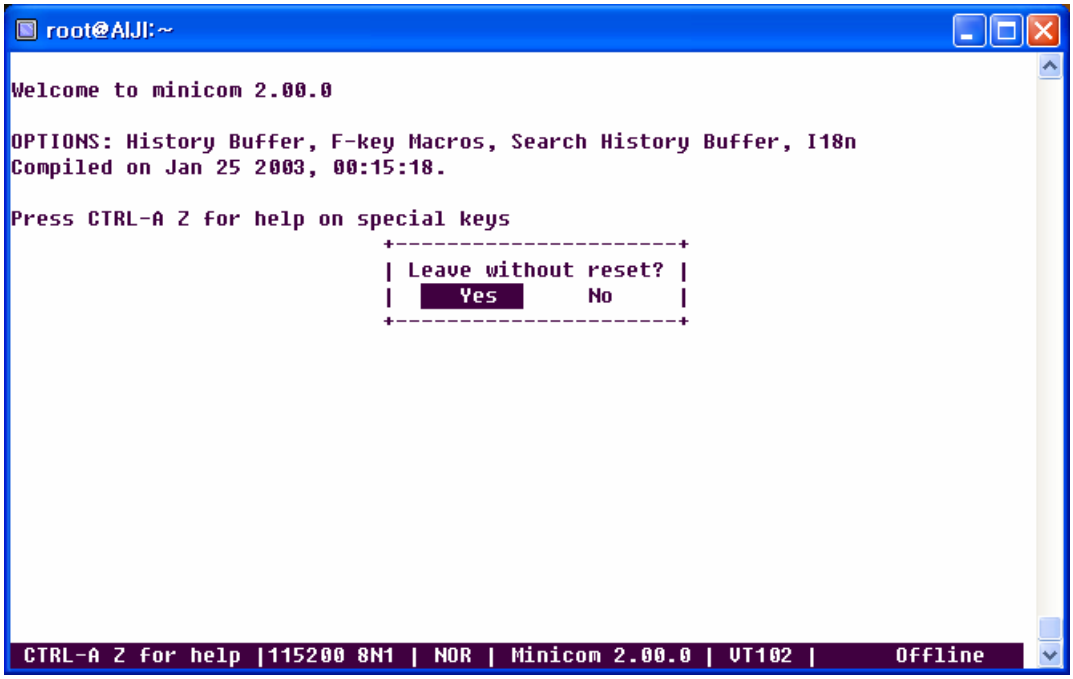


[그림 3.20]

이 상황에서 Ctrl+ A를 누른 후 Z키를 누르면 다음과 같은 minicom 메뉴가 나온다.

각 메뉴에 대한 설명은 하지 않는다. 본인이 직접 한번 해보기 바란다.

Minicom이 동작하는 상황에서 Ctrl+A를 누른 후 X를 누르면 minicom을 종료하게 된다.



[그림 3.21]

3.4 WinCE 개발 환경

제공한 CD의 WinCE5.0 디렉토리에는 MBA2440을 위한 WinCE 개발을 위해 WinCE 5.0용 BSP와 컴파일된 이미지가 포함되어 있다.

WinCE를 개발하기 위해서는 WinCE 5.0 platform builder가 필요하다. 이는 개발자가 직접 구매해야 하며, 이 문서에서는 설치 방법에 대해서는 다루지 않는다.

MBA2440 용 WinCE 5.0 BSP를 컴파일하는 방법과 eboot 또는 nboot를 이용하여 MBA2440상에서 WinCE를 부팅하는 방법은 7장. WinCE를 참조하기 바란다.

3.5 OPENice 디버거 장비

이 장에서는 AIJISystem에서 개발한 MDS장비에 대하여 소개하고 이를 이용하여 MBA2440의 flash memory나 메모리에 실행 파일을 올리고 동작시키는 방법에 대해서 알아본다.

3.5.1 OPENice-A900, OPENice-A1000

OPENice는 아이지 시스템에서 국산 기술로 개발한 국내 유일의 MDS 장비로서, 타 MDS 장비와 비교하였을 때, 전혀 손색이 없는 기능을 가지고 있다.

OPENice의 특징은 다음과 같다.

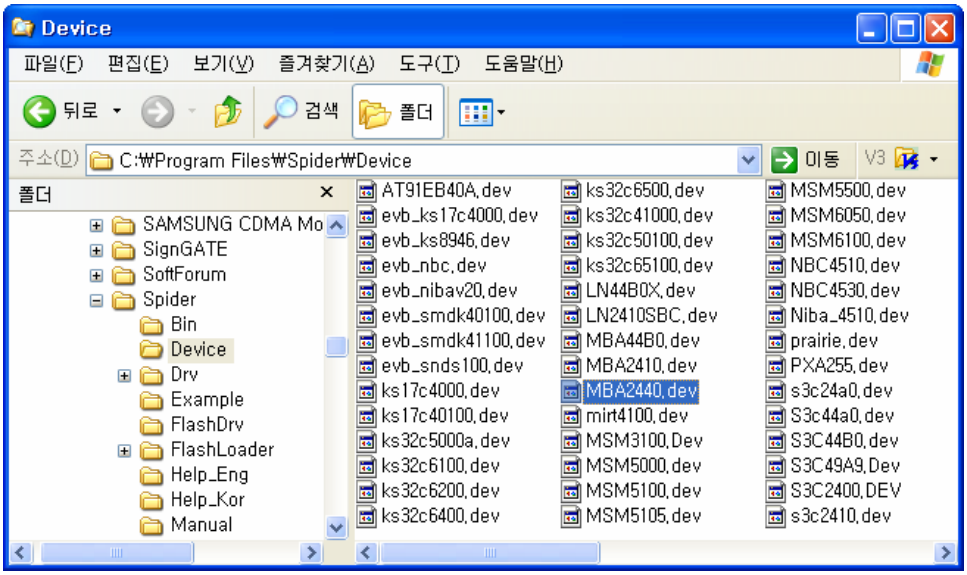
- **다양한 ARM core Processor 지원**
  - OPENice-A1000: ARM7, ARM9, ARM10, ARM11, XScale
  - OPENice32-A900[1]: ARM7, ARM9, ARM10, XScale
- **CDMA/GSM/GPRS/UMTS 디바이스 지원**
  - ARM core 기반의 휴대폰용 프로세서 지원
    (OPENice-A1000 및 OPENice32-A900에서 지원)
- **멀티 코어/ 프로세서 지원**
- **편리한 GUI SMU/SFR 설정 윈도우 제공**
- **JTAG 인터페이스**
- **고속 플래쉬 다운로딩 및 디버깅**
  - OPENice-A1000 및 OPENice32-A900에서 지원
- **Stand Alone 플래쉬 프로그래밍 (128MB)**
  - OPENice-A1000 만 지원
- **USB을 통한 고속 이미지 다운로딩**
  - USB 2.0: OPENice-A1000
  - USB 1.1: OPENIce32-A900
- **간편한 펌웨어 업그레이드**
- **전용 디버거 (Spider) 제공 및 타 디버거 호환**
  - 전용 디버거 : Spider – OPENice-A1000 및 OPENice32-A900 용
  - 호환 디버거: ARM SDT/ADS, IAR EWARM, Linux GDB,
    Windows CE Platform Builder

[1] OPENice32-A900은 XScale의 일부 디바이스는 지원하지 않음. 또한 신규 디바이스 지원 하지 않음 (특히 멀티 코어(ARM + non ARM based core).

3.5.1.1 Device file 설치

OPENice를 위한 디버거인 Spider를 설치하고 나면, MBA2440용 device file을 Spider를 위해 설치해야 한다. 이 device file은 OPENice를 통해 MBA2440을 제어하려 할 때, 보드 초기화를 위한 S3C2440 레지스터들을 setup하기 위한 정보들을 담고 있다.

제공한 CD의 utiliy/devicefile 디렉토리에 있는 MBA2440.dev 파일을 C:/Program files/Spider/Device 디렉토리에 복사한다.



[그림 3.22]

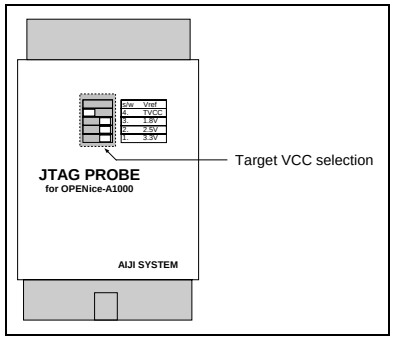
3.5.1.2 PC와의 연결

OPENice는 PC와 USB로 통신한다. OPENice에 전원을 공급한 뒤 USB 드라이버 설치 화면에서 드라이버를 설치한 후 사용한다. 자세한 사항은 OPENice 매뉴얼의 “제품 설치”를 참조하라.

3.5.1.3 MBA2440과의 연결

OPENice는 기본적으로 타겟 보드의 20핀 또는 14핀으로 구성된 JTAG 커넥터에 연결하여 사용할 수 있다. MBA2440은 20핀 커넥터이다.

OPENice-A1000의 경우에는 JTAG probe가 있는데 probe의 점퍼 세팅은 다음과 같이 한다.



[그림 3.23]

OPENice의 JTAG 케이블을 MBA2440의 JTAG port에 연결한다.

3.5.2 Boot flash에 이미지 쓰기

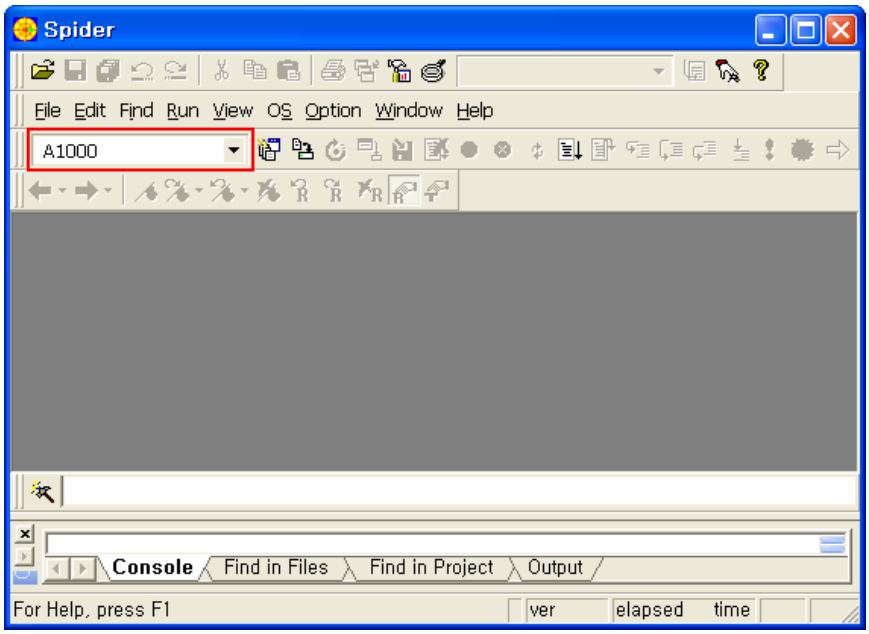
OPENice를 위한 환경 설정이 완료되었다고 가정하고 Boot flash(AMD flash나 INTEL flash)에 이미지를 write하는 방법에 대해 알아본다. (NAND flash write는 제공하지 않음. U-boot 이용할 것)

OPENice 환경 설정은 OPENice 매뉴얼을 참고하기 바란다.

3.5.2.1 AMD flash에 이미지 쓰기

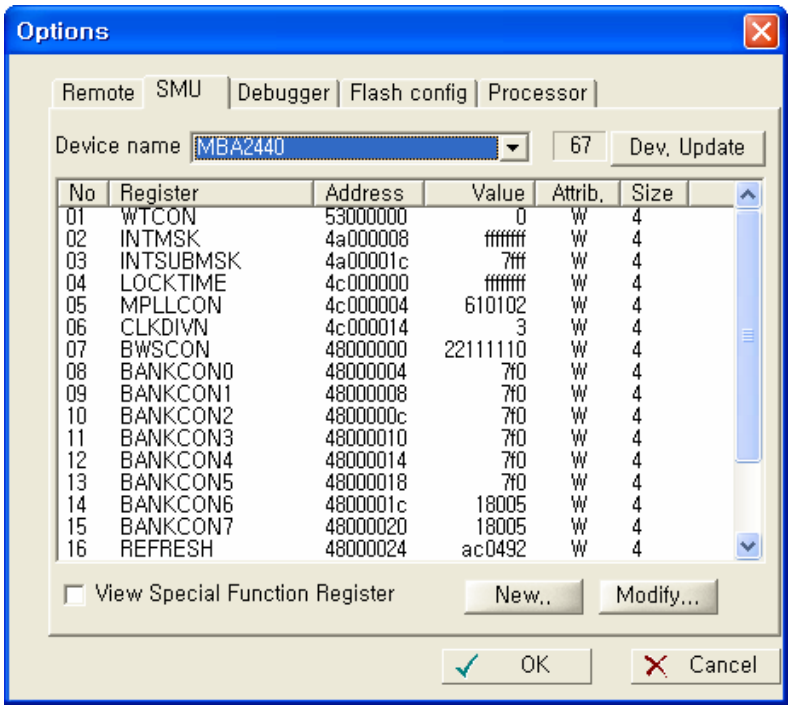
1. 케이블을 연결한다.
2. MBA2440의 boot flash 설정을 AMD flash로 설정한다.(1.3.2.1 NOR flash를 Boot flash로 사용하기 참조)
3. MBA2440의 전원을 켠다.
4. Spider를 실행 시키고 나서 좌측 상단에 해당 OPENice를 선택한다.

   이 문서에서는 A1000을 선택하였다.



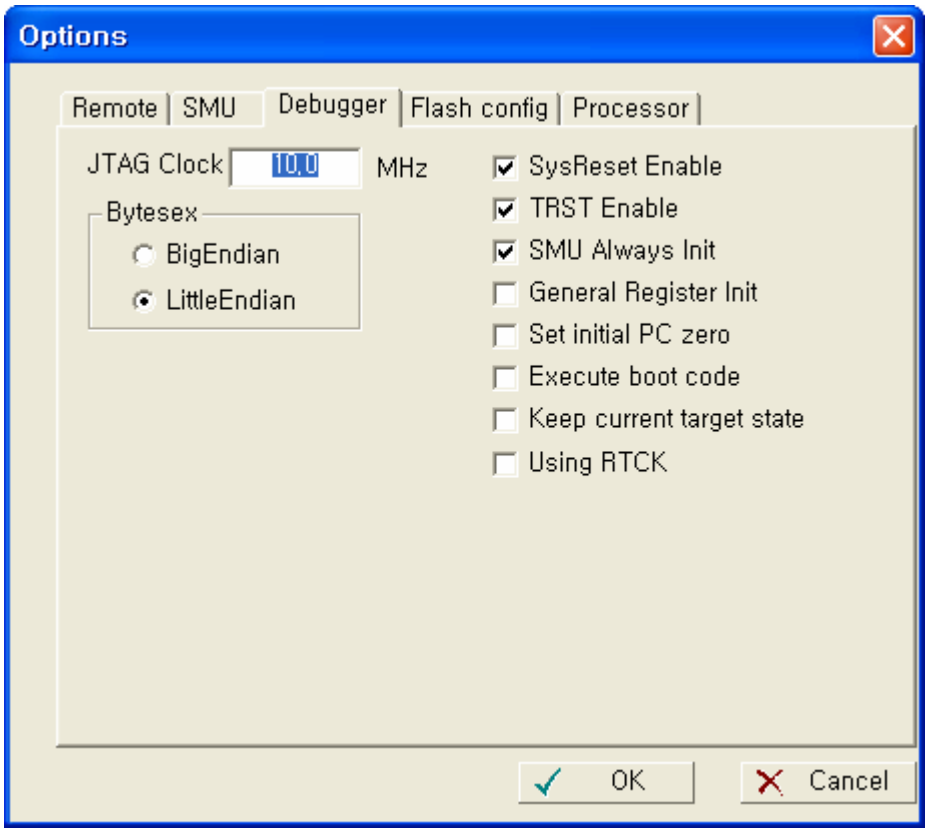
[그림 3.24]

5. Spider의 Option->Configure Interface (Cntl+E)를 선택하면 Options 창이 뜨는데 SMU 탭에서 MBA2440을 선택한다.



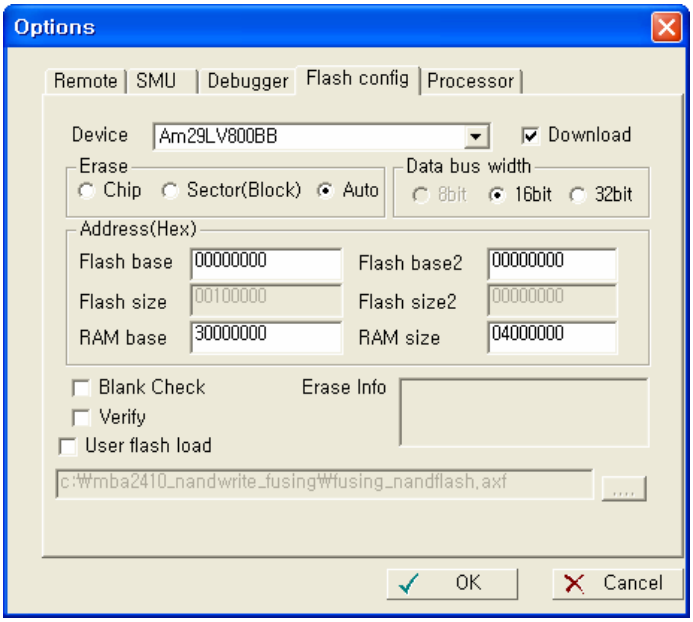
[그림 3.25]

6. Options 창의 Debugger 탭을 [그림 3.26]과 같이 설정한다.



[그림 3.26]

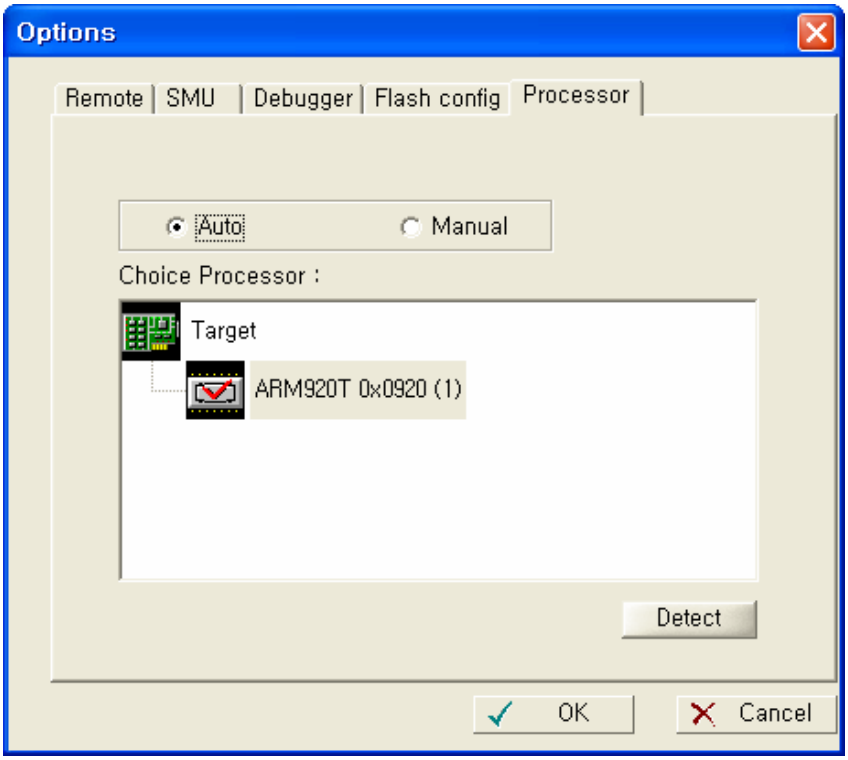
7. Options 창의 Flash config 탭을 [그림 3.27]과 같이 설정한다.



[그림 3.27]

[그림 3.27]에서 Device로는 AMD flash를 선택하였고 Download가 체크되어 있으며, Erase는 Auto, Data bus width는 16bit, Flash base는 0x0, RAM base는 0x30000000, RAM size는 0x4000000(64MB)로 설정되어 있음을 확인할 수 있다.

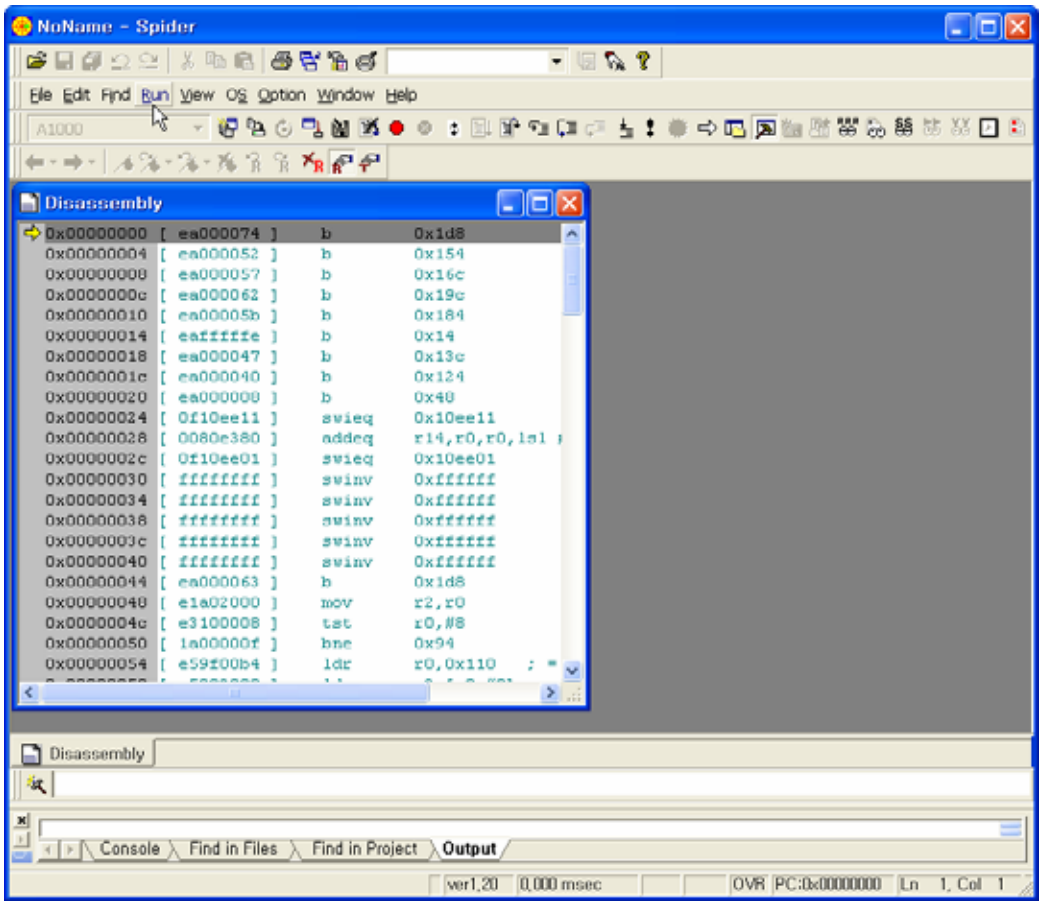
8. Options 창의 Processor 탭에서 ARM920T core를 detect한다.



[그림 3.28]

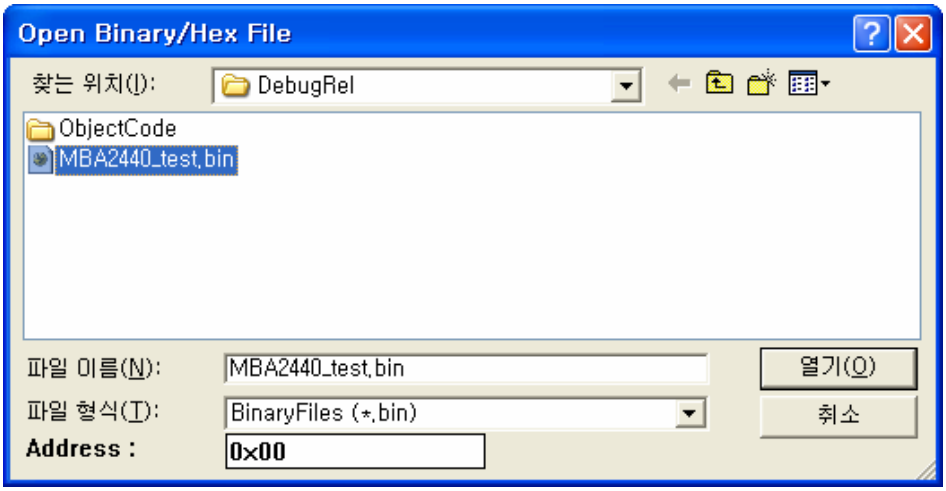
[그림 3.28]에서 Detect 버튼을 누른다. PC와 OPENice, OPENice와 MBA2440이 제대로 연결되어 있다면 [그림 3.28]과 같은 모습을 볼 수 있다.

9. Options 창에서 OK 버튼을 누르면, [그림 3.29]와 같이 초기화된 모습을 볼 수 있다.



[그림 3.29]

10. Spider의 File->Download Bin/Hex File을 선택한다.



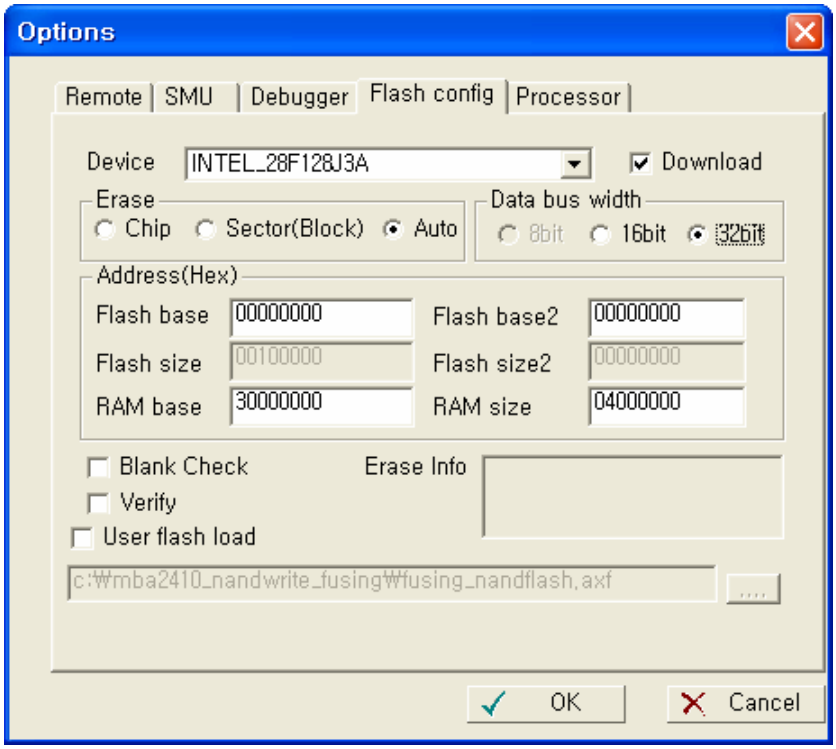
[그림 3.30]

[그림 3.30]에서 보는 바와 같이, 원하는 파일을 선택한다. 이 때, Address에 AMD flash의 원하는 위치의 주소를 적어준다. 0x0번지에 write하고자 한다면 그림과 같이 0x0번지를 적는다.

열기 버튼을 누르면, Spider에서 AMD flash의 write할 영역을 erase한 후 write하게 된다.

3.5.2.2 INTEL flash에 이미지 쓰기

INTEL flash는 MBA2440의 옵션 사항이다. 만약, MBA2440에 INTEL flash를 부착하였다면, 3.5.2.1 AMD flash에 이미지 쓰기의 내용을 그대로 따라 하되, 7번 사항만 [그림 3.31]과 같이 설정해 주면 된다.



[그림 3.31]

Device를 INTEL_28F128J3A 로 선택하고, Data bus width는 반드시 32bit 이어야 함에 유의하자.

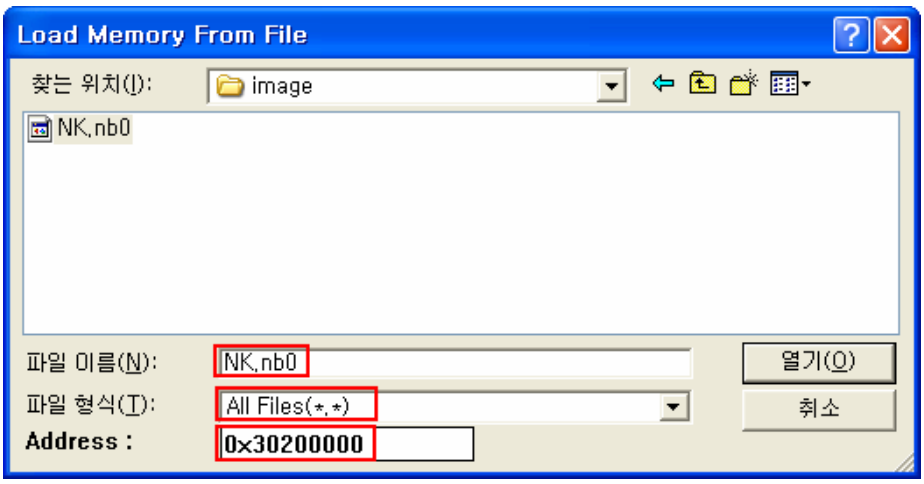
이 외에도 Spider 설치시 같이 제공되는 FlashUp 유틸리티를 이용하여 NOR flash에 write할 수 있다. 자세한 사항은 Spider 매뉴얼을 참조하기 바란다.

3.5.3 메모리에 이미지 적재

WinCE 이미지의 경우에는 SDRAM 메모리의 0x30200000번지에 적재하고 PC 값을 0x30200000번지로 세팅한 후, Spider의 Run->Go를 선택하면 WinCE가 부팅된다.

이처럼, SDRAM 메모리에 직접 이미지를 올리는 방법에 대해 알아본다.

1. Spider에서 Option->Configure Interface를 선택하고 3.5.2.1 AMD flash에 이미지 쓰기의 1번에서 9번을 따라한다.
2. Spider의 초기화면에서 File->Load Memory From File을 선택한다.
3. Load Memory From File창에서 [그림 3.32]와 같이 WinCE 이미지를 선택한다.

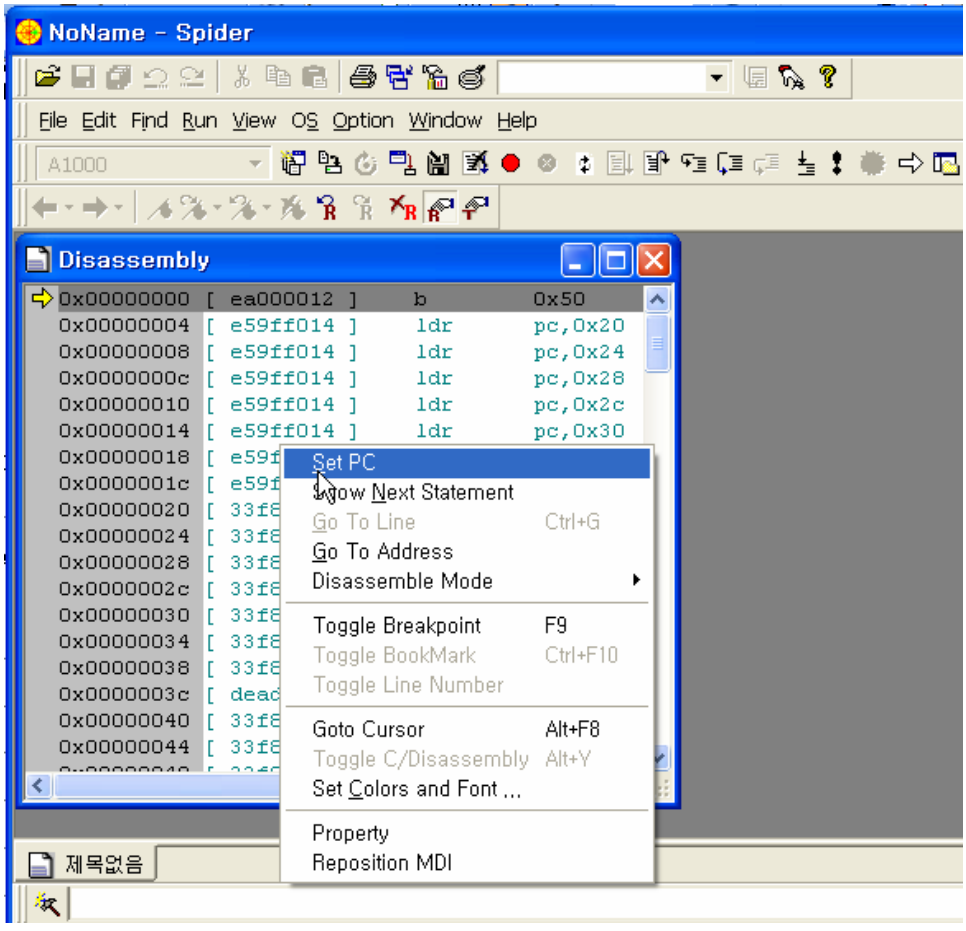


[그림 3.32]

Address는 적재하고자 하는 Address인 0x30200000으로 설정한다.

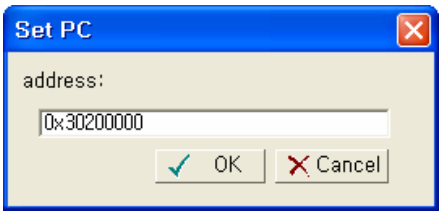
열기 버튼을 누르면, JTAG 포트를 통해 WinCE 이미지 NK.nb0가 0x30200000 번지로 적재된다. 이미지가 크기 때문에 download하는데 시간이 소요된다.

4. 다운로드가 완료된 후에 PC값을 0x30200000으로 설정한다.



[그림 3.33]

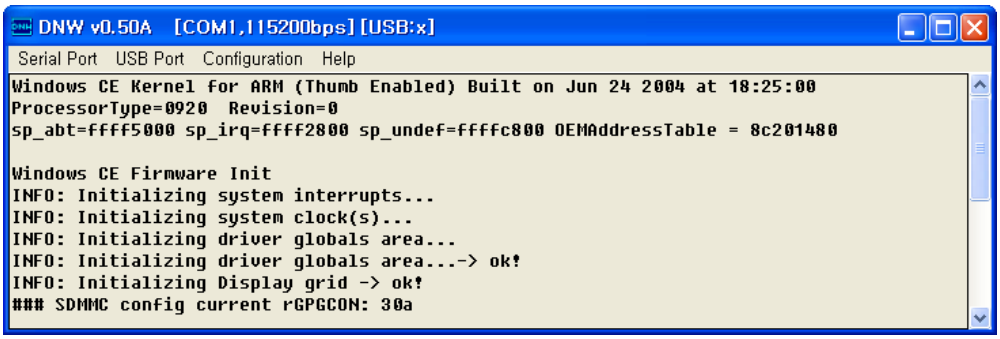
[그림 3.33]에서 보는 바와 같이, Disassembly창에 마우스를 대고 오른쪽 버튼을 클릭한 후, Set PC를 클릭한다.



[그림 3.34]

[그림 3.34]처럼 Set PC 창에서 0x30200000 주소를 세팅한 후, OK 버튼을 누른다.

5. Spider의 Run->Go를 선택하거나, 또는 F8을 누르면 WinCE 이미지가 부팅됨을 확인할 수 있다.

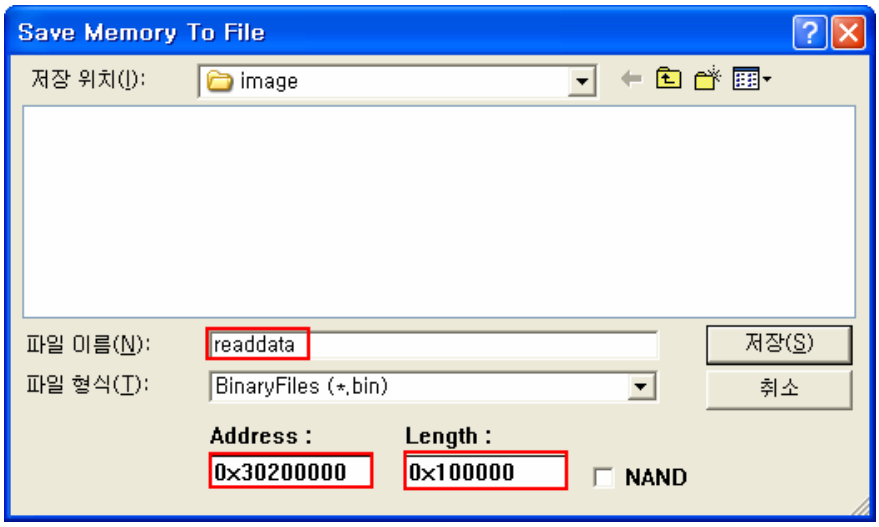


[그림 3.35]

3.5.4 메모리에서 데이터 읽기

MBA2440의 SDRAM 메모리에 있는 데이터를 PC로 읽어올 수 있다.

1. Spider에서 Option->Configure Interface를 선택하고 3.5.2.1 AMD flash에 이미지 쓰기의 1번에서 9번을 따라한다.
2. Spider의 초기화면에서 File->Save Memory To FIle을 선택한다.
3. Save Memory To FIle창에서 읽어오고 싶은 영역의 시작 주소와 크기, 저장될 파일 이름을 설정해 준다.



[그림 3.36]

Address는 읽어오고 싶은 영역의 시작주소를 입력하고, Length에는 크기를 입력하며, 파일 이름에는

저장될 파일명을 입력한다.

입력이 완료되면 저장 버튼을 누른다. 크기에 따라 시간이 많이(길게는 30~40분) 소요될 수 있으므로 완료될 때까지 충분히 기다린다.

3.5.5 OPENice 사용시 유의 사항

OPENice와 MBA2440을 연동할 경우 주의해야 할 사항들에 대해서 알아본다.

1. JTAG를 통해 OPENice와 MBA2440이 연결되어 있는 경우

- reset 버튼은 동작하지 않는다.

OPENice의 reset 버튼을 누르면 MBA2440도 같이 reset이 된다.

- 전원을 껐다가 켜면 boot flash에 있는 부트 코드는 동작하지 않는다.

부트 코드를 동작 시키고 싶다면, Set PC창에서 address를 0x0으로 설정한 후, F8키를 누른다.

2. Spider에서 Option->Configure Interface를 선택하고 Option창에서 OK를 버튼을 누르면 OPENice를 통해 MBA2440을 초기화 하게 된다.

그러나, 초기화가 되지 않고 에러 메시지가 출력된다면, 점검을 해 본다.

- 518 에러 메시지

이 에러의 경우는 Option창의 Debugger 탭에서 SysReset Enable이 체크되어 있지 않거나 JTAG reset 시그널이 제대로 연결되어 있지 않은 경우이다.

- 683 에러 메시지

Option창의 SMU 탭에서 타겟 보드에 맞는 device file인지 체크해 본다.

- 778 에러 메시지

보드에 전원이 들어가 있는지, Option창의 Processor 탭에서 CPU core가 제대로 detect 되었는지 체크해 본다.

4 MBA2440 Test Program

이 장에서는 MBA2440 test firmware 프로그램에 대해 구체적으로 알아본다.

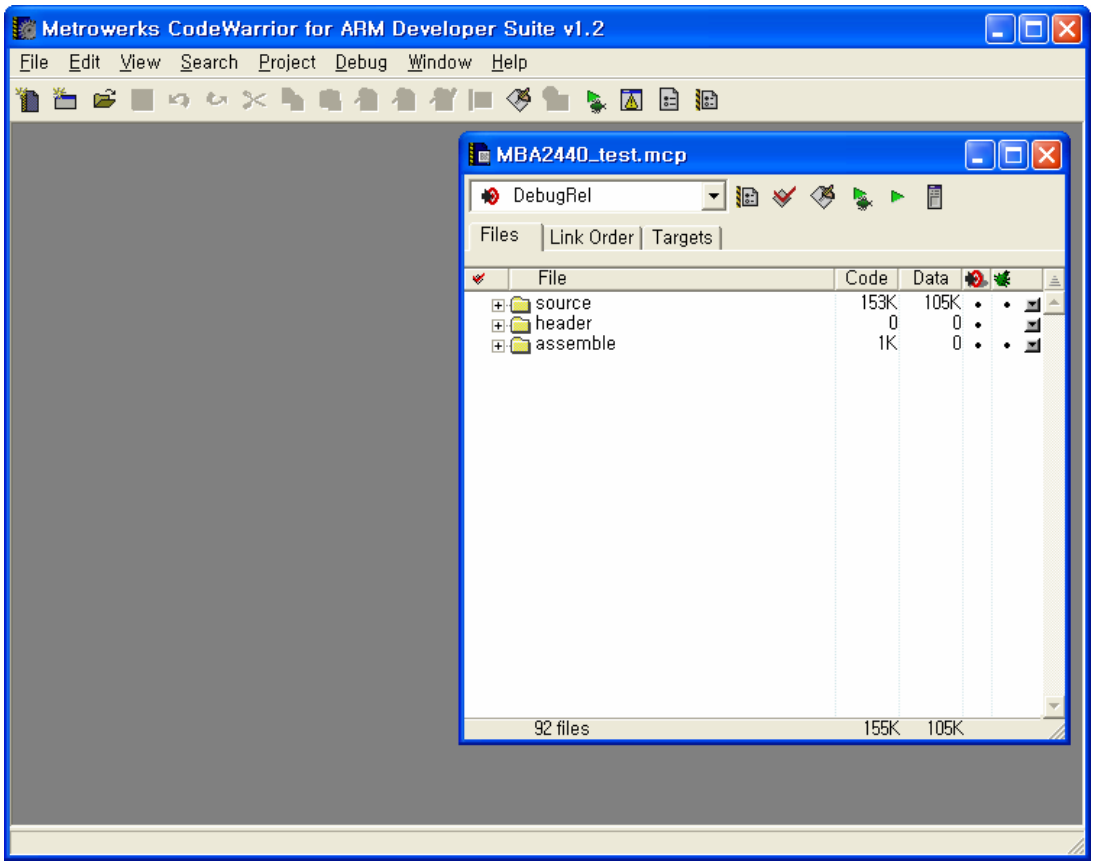
4.1 프로그램 컴파일 하기

MBA2440 test 프로그램은 ADS1.2로 개발되었다. 이 장에서는 **ADS1.2가 PC에 설치되어 있으며 ADS1.2의 사용법을 알고 있다고 가정하고 설명**할 것이다.

제공한 CD에서 firmwares/test_program 디렉토리에 **MBA2440_test_v1.0.zip** 압축파일이 있다. 이 압축파일이 MBA2440 test 프로그램의 소스이다.

압축을 해제하면 MBA2440_test_v1.0 디렉토리가 생성된다.

MBA2440_test_v1.0/MBA2440_test 디렉토리 안에 MBA2440_test.mcp 파일이 있다. 이 파일을 더블클릭하면 ADS1.2가 실행되며 이미 생성되어 있는 project창을 볼 수 있다. [그림 4.1]

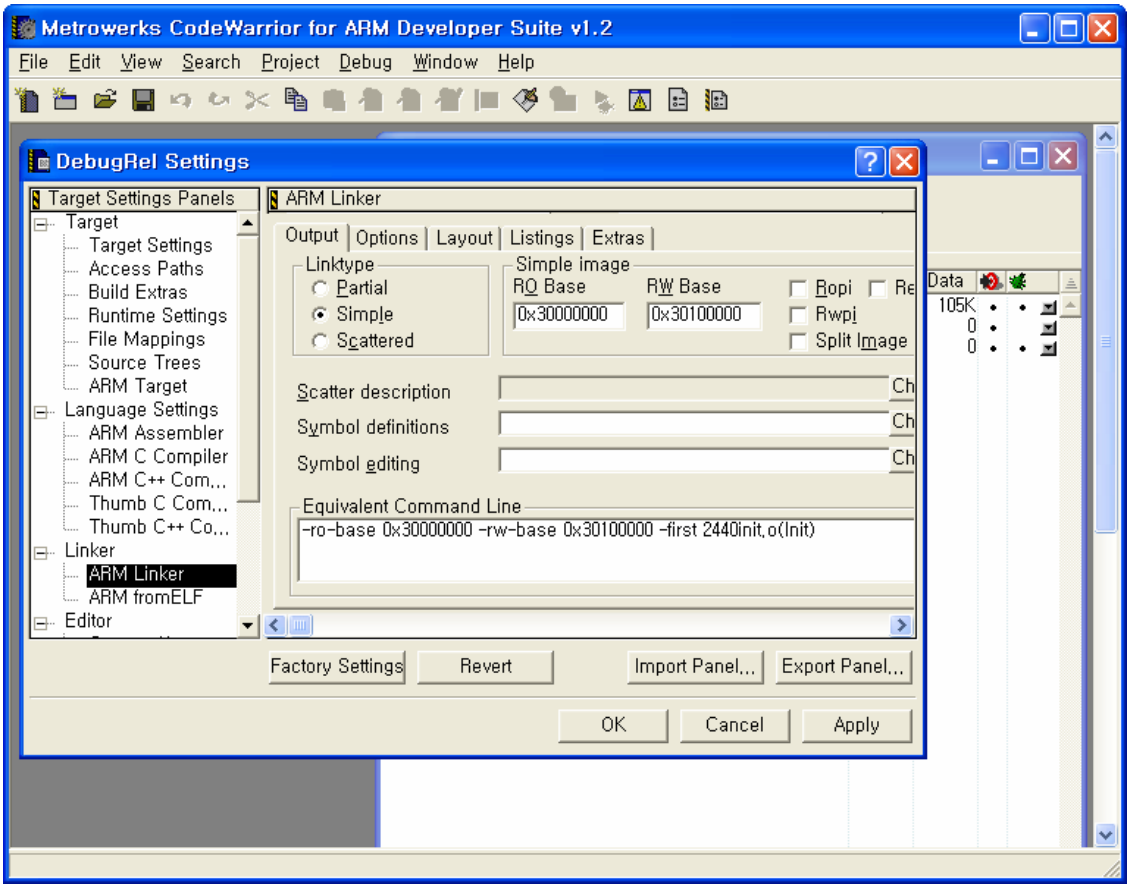


[그림 4.1]

이 소스는 컴파일 옵션에서 RO_BASE는 0x0으로 RW_BASE는 0x30000000으로 설정하여 컴파일 하였다. 컴파일된 이미지는 반드시 0x0 번지에 write해야 동작이 됨을 유의한다.

만약, boot flash상에서 동작시키지 않고 SDRAM에서 동작 시키고 싶다면 RO_BASE는 0x30000000(이 주소는 SDRAM 물리주소의 시작번지이다) RW_BASE는 0x30100000으로 설정한다.

RW_BASE를 0x30100000으로 설정하는 이유는, test 프로그램 이미지가 대개 0x30000000 ~ 0x3003ffff 영역 안에 적재가 되기 때문이다. 이 영역에 다른 데이터를 쓰게 되면 test 프로그램이 동작하지 않는다. 0x40000 크기의 test 프로그램 이미지 영역 외에 여유로운 메모리 공간을 위해 1M 정도의 영역을 비워 놓기 위해 RW_BASE를 0x30100000으로 설정한다. 자세한 사항은 소스의 초기화 부분(2440init.s)을 분석하기 바란다.



[그림 4.2]

4.1.1 테스트 프로그램 설정

CD로 제공한 소스는 SDRAM 64MB에 맞춰져 있다.

SDRAM이 128MB로 확장된 MBA2440 보드라면 소스에서 메모리와 관련된 부분의 수정이 필요하다.

제공한 소스에서 **Option.inc 파일과 Option.h 파일을 수정**해야 한다.

먼저, Option.inc 파일을 수정하자.

이 파일에 있는 내용들은 보드 초기화 하기 위한 어셈블리 코드에서 참조해야 할 내용들을 정의해 놓은 것이다.

```
;=========================================
; NAME: OPTION.A
```

```
; DESC: Configuration options for .S files
; HISTORY:
; 02.28.2002: ver 0.0
; 03.11.2003: ver 0.0      attached for 2440.
; jan E, 2004: ver0.03   modified for 2440A01.
; 04.15.2005: modified for MBA2440
;=========================================
                    GBLA    MBA2440_MEM_SIZE
;MBA2440_MEM_SIZE    SETA    0x4000000
MBA2440_MEM_SIZE     SETA    0x8000000
```

MBA2440_MEM_SIZE 매크로 정의하는 부분에서 0x4000000으로 정의된 값을 0x8000000으로 수정해야 한다.

";" 표시는 어셈블리 코드에서는 주석처리를 의미한다.

다음으로 Option.h 파일을 수정한다.

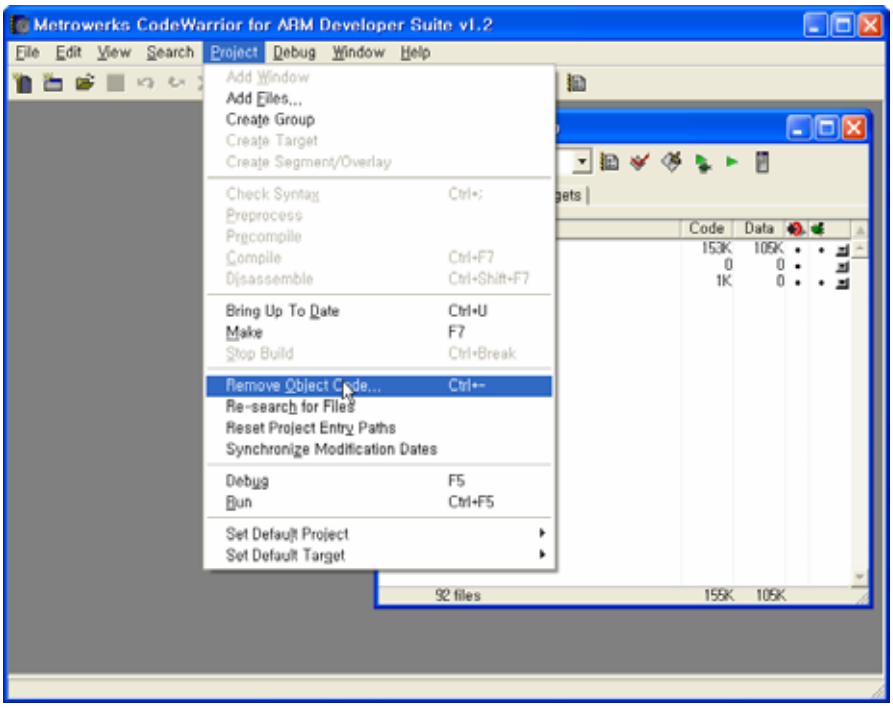
```
/************************************************************
 NAME: option.h
 DESC: To measuure the USB download speed, the WDT is used.
         To measure up to large time, The WDT interrupt is used.
 HISTORY:
 Feb.20.2002:Shin, On Pil: Programming start
 Mar.25.2002:purnnamu: S3C2400X profile.c is ported for S3C2440X.
 Jan.E.2004:DonGo: Modified for S3C2440a.
 04.15.2005: modified for MBA2440
 ************************************************************/

#ifndef __OPTION_H__
#define __OPTION_H__



//#define MBA2440_MEM_SIZE              0x4000000
#define MBA2440_MEM_SIZE              0x8000000
```

MBA2440_MEM_SIZE 매크로 정의하는 부분에서 0x4000000으로 정의된 값을 0x8000000으로 수정한다.

4.1.2 컴파일 하기

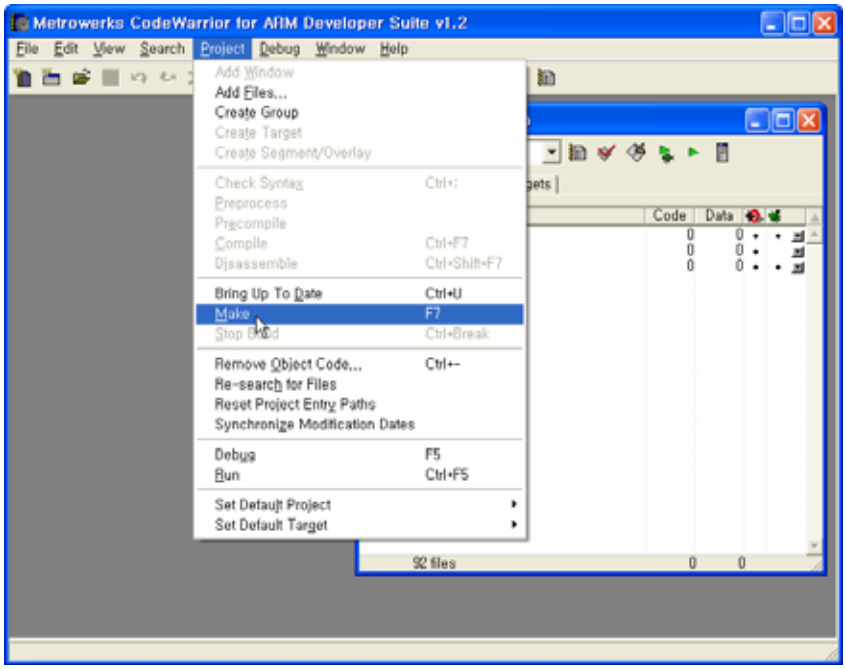
제공한 소스는 시스템 메모리를 64MB로 설정하여 컴파일 되어 있다. 그러나, 다시 컴파일 해보자.

Project->Remove Object Code를 클릭한 후, Remove Object Code창에서 All Targets 버튼을 클릭한다. [그림 4.3]



[그림 4.3]

Project->Make를 선택하거나 F7버튼을 눌러 컴파일 한다. [그림 4.4]

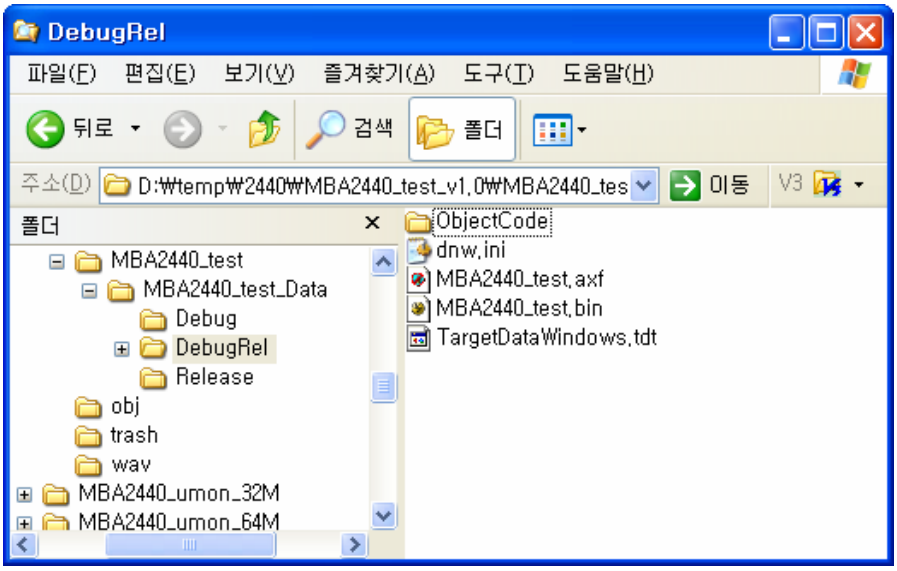


[그림 4.4]

컴파일이 완료되었다면,

MBA2440_test_v1.0/MBA2440_test/MBA2440_test_Data/DebugRel 에

MBA2440_test.bin 파일과 MBA2440_test.axf 파일이 생성되어 있음을 확인할 수 있다.



[그림 4.5]

MBA2440_test.bin 이미지 파일은

RO_BASE를 0x0으로 설정하여 컴파일 하였다면, AMD NOR flash의 0x0번지에 write한다.

RO_BASE를 0x30000000으로 설정하여 컴파일 하였다면, 메모리의 0x30000000 번지에 적재한 후, 동작 시킨다.

메모리에 적재한 후 동작시키는 방법은 OPENice를 이용하는 방법(3.5.3 메모리에 이미지 적재 참조)이나 u-boot를 이용하는 방법(5.2.2.4 NOR flash command 참조)이 있다.

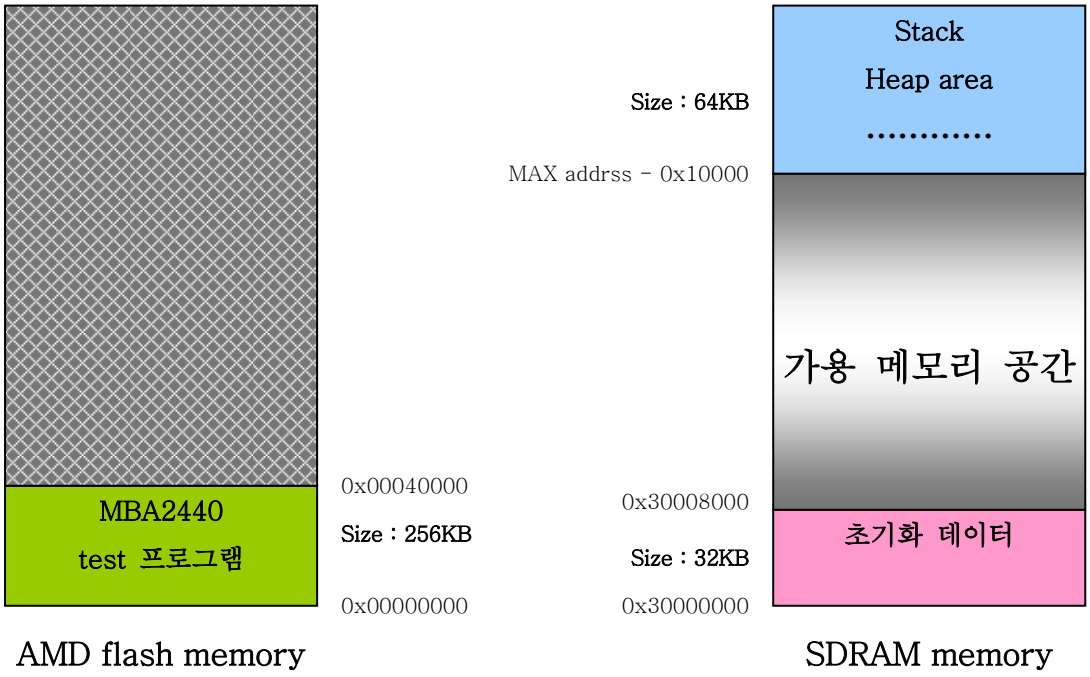
4.2 MBA2440 Test 프로그램의 유의할 점

MBA2440 Test 프로그램은 AMD flash를 boot flash로 설정한 환경하에서 제대로 동작하도록 프로그래밍되어 있다.

다음의 두 가지 경우에 따라 테스트 프로그램의 memory map이 다르며, NOR flash 테스트 메뉴에서 AMD flash의 테스트 가능 여부가 달라진다.

4.2.1 0x0 번지를 RO_BASE로 설정한 경우

소스를 컴파일 할 때, RO_BASE를 0x0번지로 설정하게 되면, [그림 4.6]과 같은 메모리 맵을 구성하게 된다. (이 경우, RW_BASE는 메모리 시작 주소인 0x30000000번지를 설정한다)



[그림 4.6]

[그림 4.6]에서 보는 바와 같이, RO_BASE를 0x0으로 설정하고 컴파일한 경우, MBA2440 테스트 프로그램코드는 AMD flash의 0x0 ~ 0x3ffff 번지 상에서 동작하게 된다.

MBA2440 test 프로그램 이미지의 크기는 256KB(0x40000) 정도이나, 개발자가 수정하거나 프로그램을 더 추가하게 될 경우 크기가 달라진다는 점을 유의하기 바란다.

그러나, 자신의 초기화 데이터나 heap, stack등은 SDRAM상에 존재하게 된다.

이 경우, NOR test 메뉴에서 AM29LV800 테스트 메뉴는 정상적으로 동작하지 않게 됨을 유의하기 바란다. 이유는 **MBA2440 test 프로그램 코드가 AMD flash상에서 동작하고 있기 때문**이다.

또 한, **SDRAM의 경우에는 초기화 데이터 부분이나, heap, stack등이 구축되어 있는 영역은 사용할 수 없다**. 이 영역이 수정되면 MBA2440 test 프로그램이 동작을 멈추게 될지도 모른다.

USB download 메뉴에서도 파일이나 이미지를 다운로드 할 때, SDRAM의 가용공간 영역에 다운로드 해야 함을 유의하기 바란다.

4.2.2 0x30000000번지를 RO_BASE로 설정한 경우

이 경우에는 **SDRAM의 메모리 맵에서 0x30000000 ~ 0x300fffff 번지와 SDRAM 맨 상위의 0x10000 번지를 제외한 부분만 사용이 가능**하다.

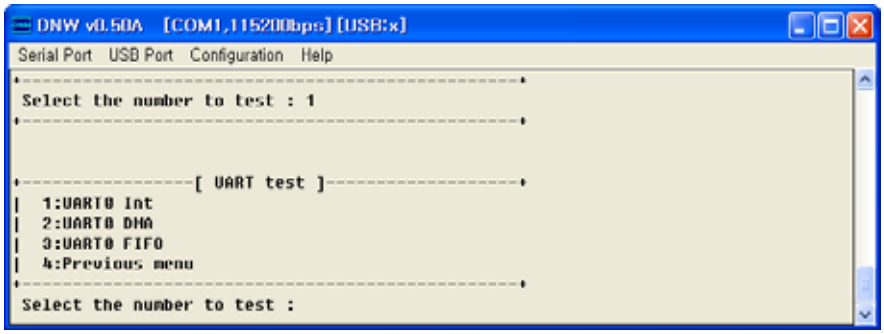
MBA2440 test 프로그램 코드가 AMD flash에서 동작하지 않기 때문에 AMD flash 테스트 메뉴가 정상적으로 동작하게 된다.

USB download 메뉴에서 파일이나 이미지를 다운로드 할 때, SDRAM의 가용공간 영역에 다운로드 해야 함을 유의하기 바란다.

4.3 Test 명령어

이 장에서는 MBA2440 Test 프로그램의 명령어에 대해서 설명한다. 자세한 설명은 하지 않는다. 명령어의 자세한 동작에 대해서는 직접 Test 프로그램 소스를 분석하기 바란다.

4.3.1 UART



[그림 4.7]

[UART test] 메뉴는 UART를 테스트 하기 위한 메뉴이다.

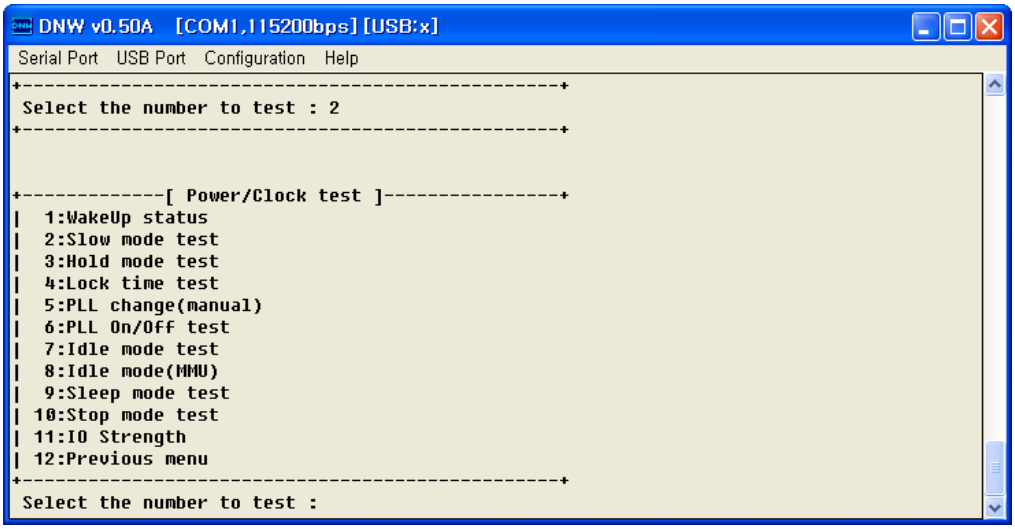
메뉴에서 보는 바와 같이 1~3번은 UART0를 인터럽트 모드, DMA 모드, FIFO 모드로 동작시키면서 테스트 한다.

각 메뉴를 선택한 후, 출력되는 메시지를 따라 테스트 해 본다.Serial chip이 UART0에만 연결되어 있다.

UART1을 테스트 하고자 할 때에는 SERIAL[1] 6핀 커넥터의 시그널을 serial chip에 연결해서 테스트할 수 있다.

UART2는 IrDA에 연결되어 있다.

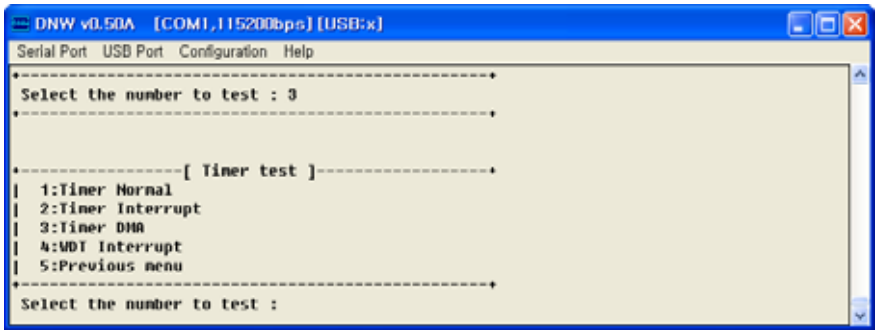
4.3.2 Power/Clock



[그림 4.8]

CPU의 동작 모드나 CPU clock을 테스트 할 수 있다. 각 메뉴를 선택한 후, 출력되는 메시지를 따라 테스트 해 본다.

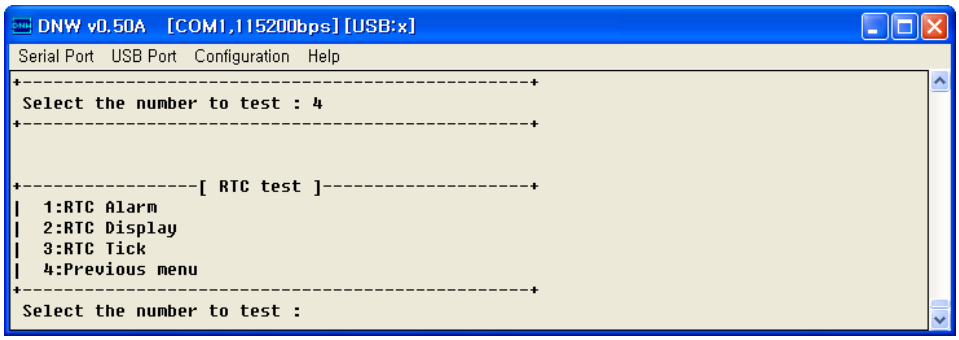
4.3.3 Timer



[그림 4.9]

Timer 테스트이다. 각 메뉴를 선택한 후, 출력되는 메시지를 따라 테스트 해 본다.

4.3.4 RTC



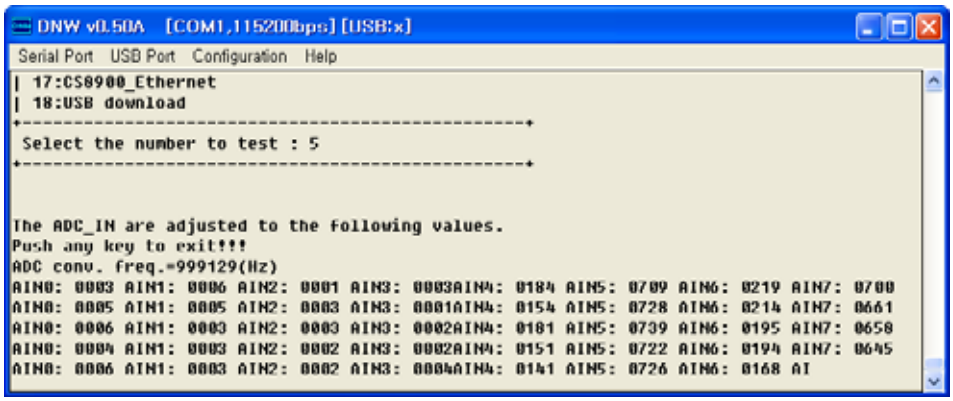
[그림 4.10]

RTC 테스트 이다. RTC를 이용해 alarm 테스트를 해 볼 수 있다.

2번 메뉴를 통해 시간을 설정하고 설정한 시간을 기준으로 1초마다 시간이 출력되는 것을 볼 수 있다.

각 메뉴를 선택한 후, 출력되는 메시지를 따라 테스트 해 본다.

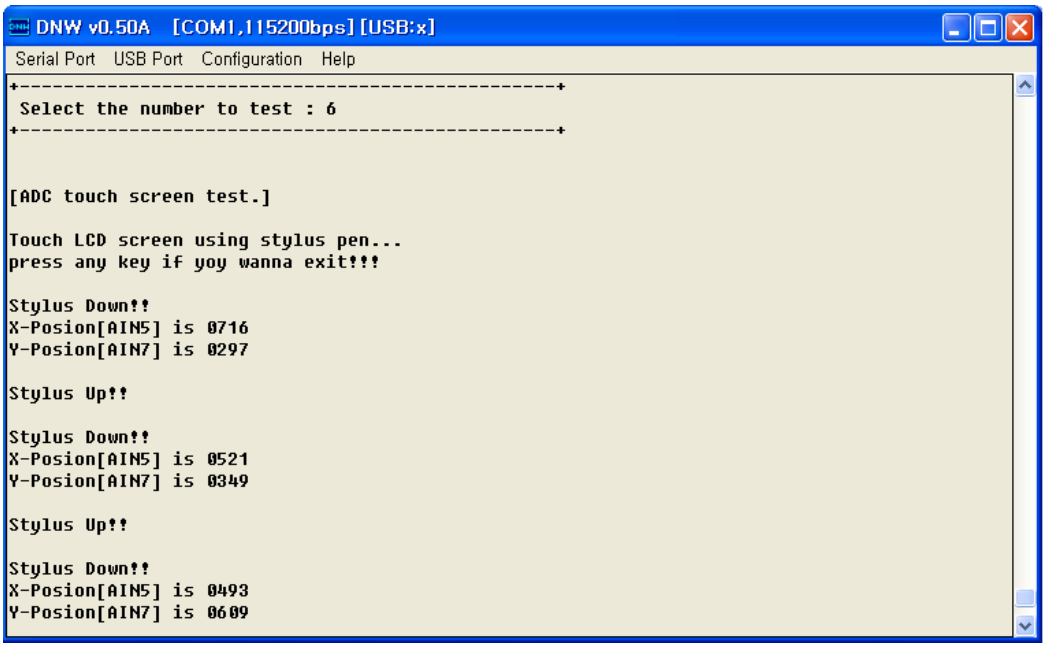
4.3.5 ADC test



[그림 4.11]

ADC 테스트 이다. 메뉴를 선택함과 동시에 출력되는 값으로 각 Ain0 ~ 7이 조정된다.
엔터키를 치면 동작이 중지 된다.

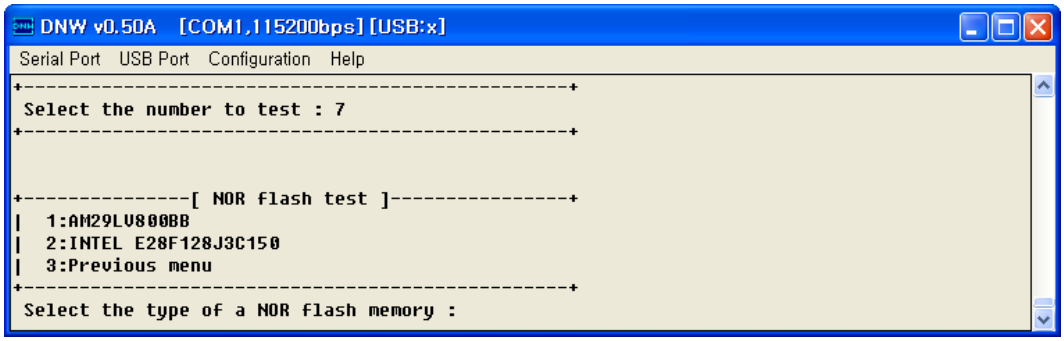
4.3.6 Touch Screen test



[그림 4.12]

Touch screen 테스트이다. 메뉴 선택한 후, LCD를 스타일러스 펜으로 접촉하면 좌표값이 출력된다.

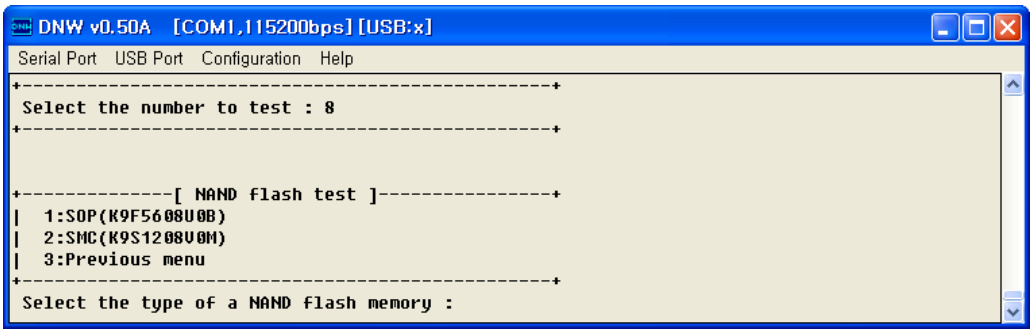
4.3.7 NOR flash test



[그림 4.13]

NOR flash를 테스트한다. 만약, boot flash로 설정된 NOR 플래시에 쓰기를 할거나, ID를 읽으려 할 때에는 제대로 동작하지 않게 된다. 이 점을 유의하기 바란다.

4.3.8 NAND flash test



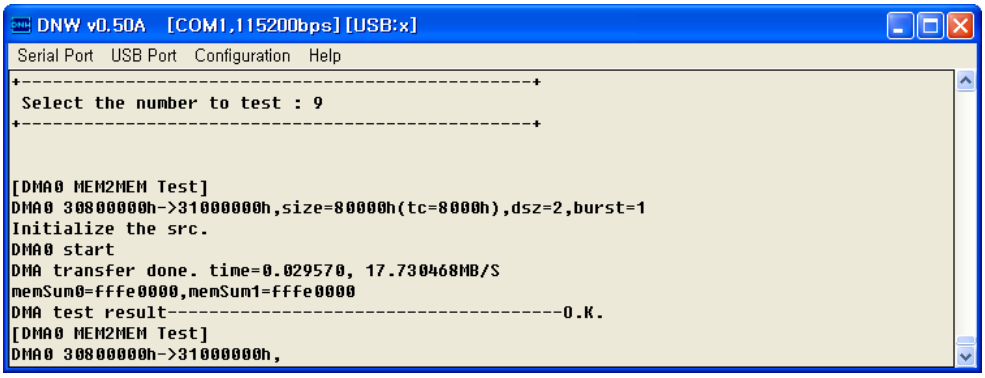
[그림 4.14]

J3 스위치를 1-2로 설정하게 되면 SOP(K9F5608U0B) NAND flash를 제어할 수 있게 되며,

J3 스위치를 2-3로 설정하게 되면 SMC(K9S1208V0M) NAND flash를 제어할 수 있게 된다.

반드시, MBA2440 test 프로그램을 동작 시키기 전에 원하는 NANDflash를 제어할 수 있도록 J3을 설정하여야 한다. 동작 중에 J3 스위치의 전환은 프로그램에 적용되지 않음을 유의한다.

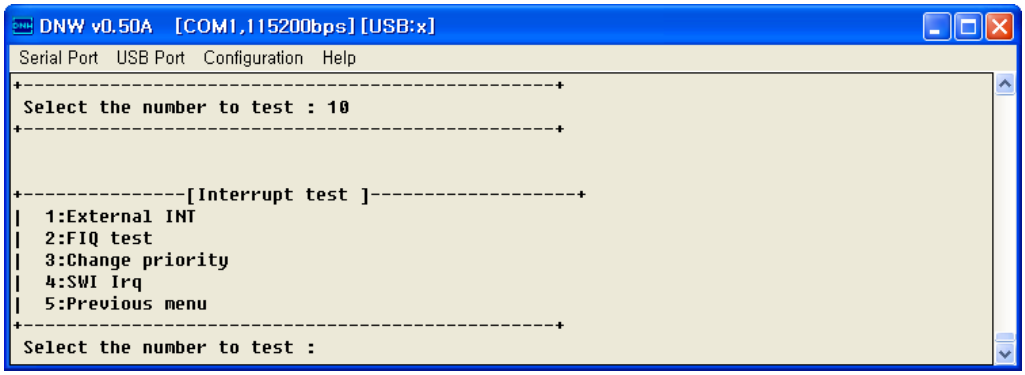
4.3.9 DMA



[그림 4.15]

메뉴 선택과 동시에 DMA를 테스트 한다.

4.3.10 Interrupt

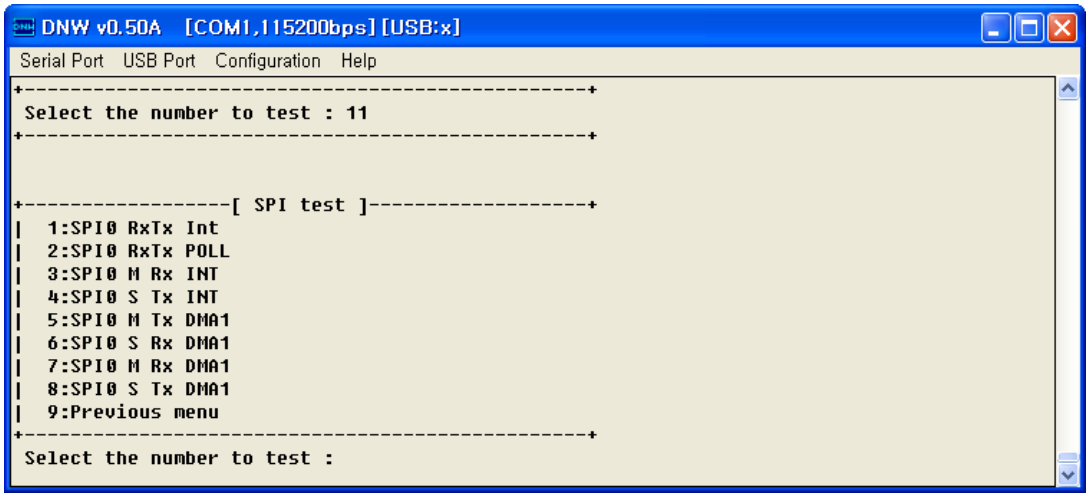


[그림 4.16]

1번 메뉴는 MBA2440 보드에 있는 스위치(KEY1 ~ 4)를 external interrupt로 설정하여 버튼을 누르면 메시지를 출력한다.

각 메뉴를 선택한 후, 출력되는 메시지를 따라 테스트 해 본다.

4.3.11 SPI



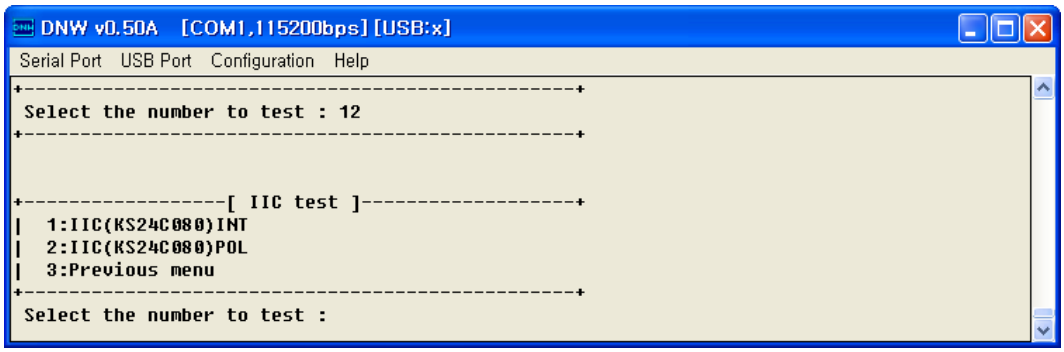
[그림 4.17]

SPI 테스트이다. 이 메뉴에서는 SPI0만을 테스트한다.

1, 2번 테스트는 MBA2440 보드의 SPI 5핀 커넥터에서 2번 핀과 3번 핀을 서로 연결한 후에 테스트한다.

3~8은 2대의 MBA2440 보드에서 SPI 핀 중 한 보드는 SPIMOSI0핀을, 다른 한 보드는 SPIMISO0 핀을 서로 연결하여 사용하여야 한다. M은 master 모드로 동작함을 의미하며, S는 slave 모드로 동작함을 의미한다.

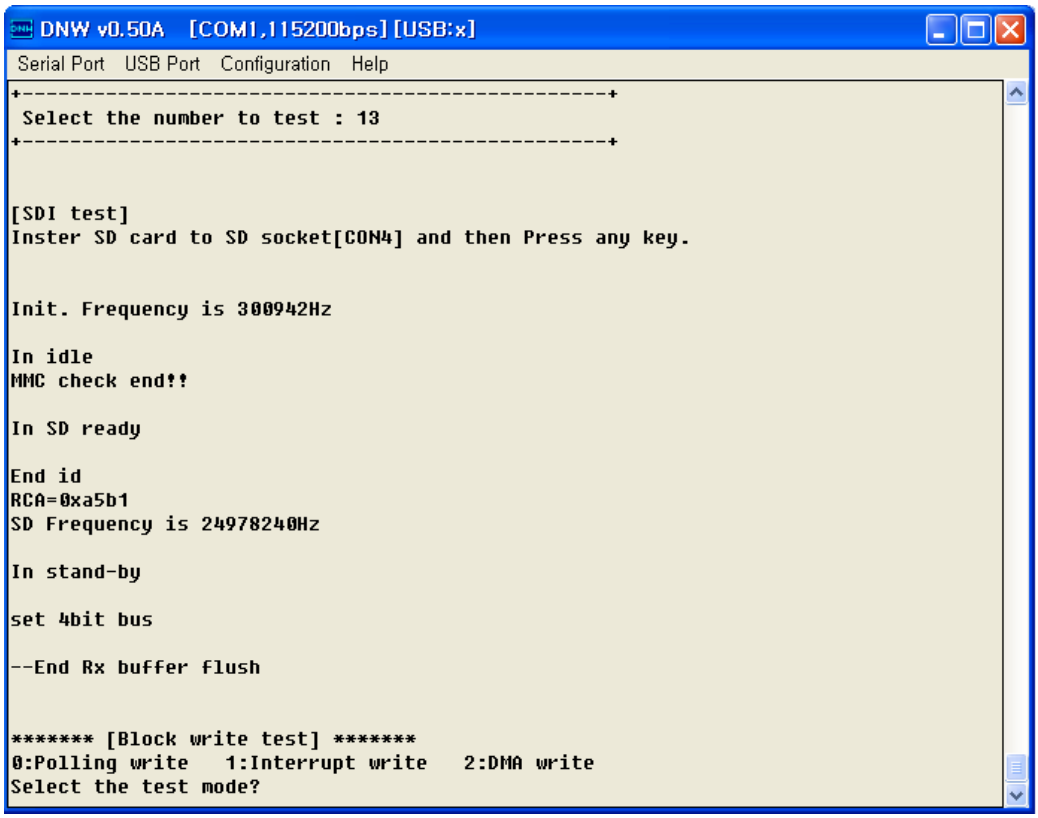
4.3.12 IIC



[그림 4.18]

MBA2440보드에 있는 KS24C080C EEPROM에 IIC를 통해 데이터 쓰기를 테스트 한다.

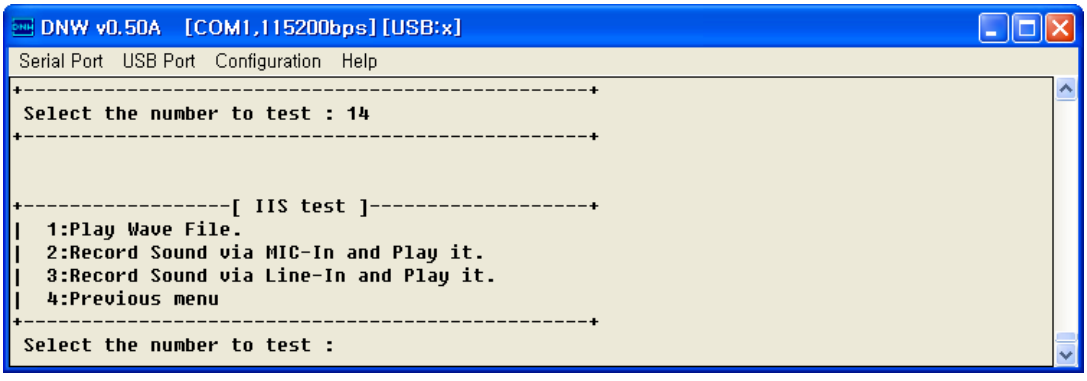
4.3.13 SDIO



[그림 4.19]

먼저 SD 메모리 카드를 삽입한 후, 13번 메뉴를 선택하면 block write test와 block read test를 한다. 각 메뉴를 선택한 후, 출력되는 메시지를 따라 테스트 해 본다.

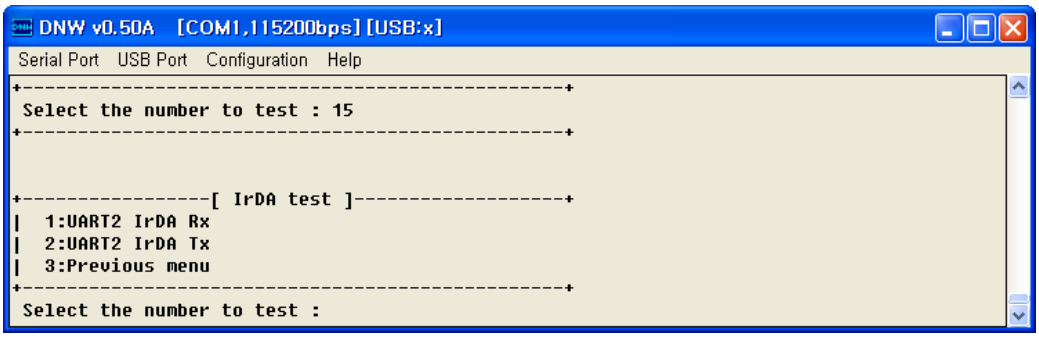
4.3.14 IIS



[그림 4.20]

UDA1341을 테스트 하는 프로그램이다. 각 메뉴를 선택한 후, 출력되는 메시지를 따라 테스트 해 본다.

4.3.15 IrDA test



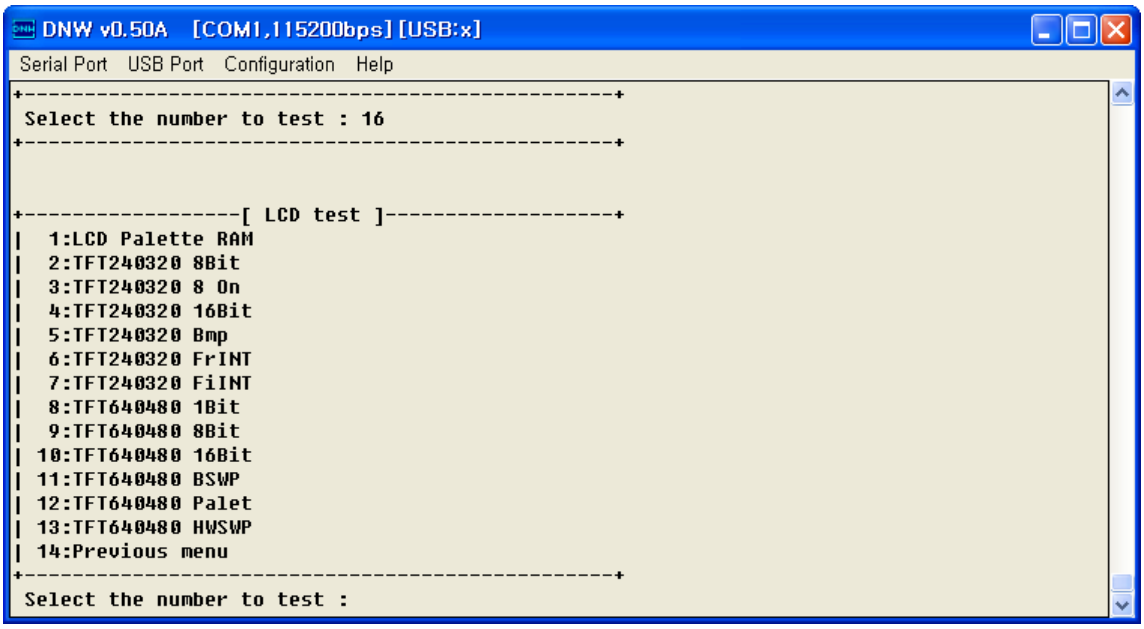
[그림 4.21]

IrDA 테스트이다. 2대의 MBA2440 보드가 필요하다. 하나는 RX로 다른 하나는 TX로 사용한다.

항상 RX를 먼저 구동시키고 나서, TX를 구동한다.

두 대의 보드를 구동할 때, IrDA 포트가 서로 정확하게 마주 보도록 위치를 잡아야 한다. 위치가 잘못 잡혀 있거나 거리가 너무 멀면 제대로 송수신 되지 않음을 유의한다.

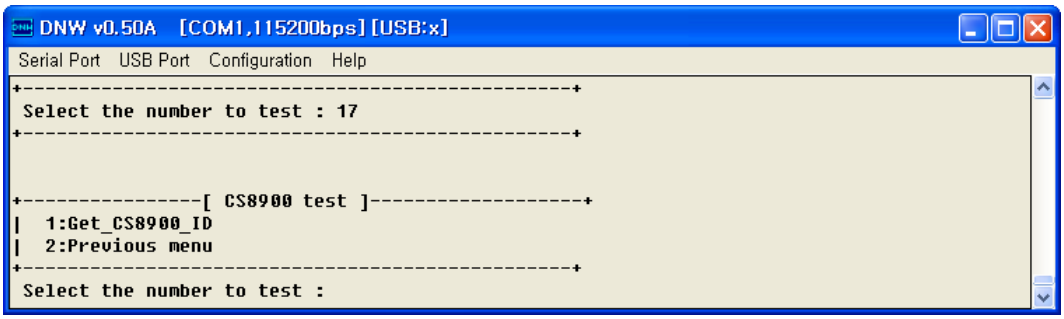
4.3.16 TFT-LCD



[그림 4.22]

240x320 TFT-LCD와 640x480 TFT-LCD를 테스트한다. 각 메뉴를 선택한 후, 출력되는 메시지를 따라 테스트 해 본다.

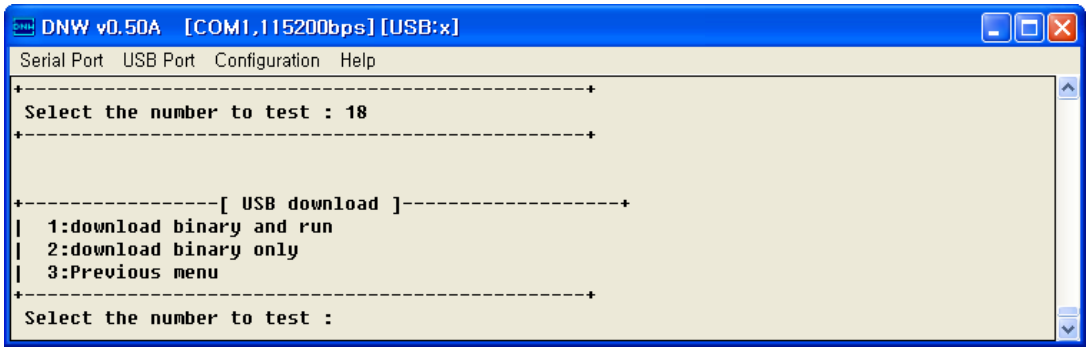
4.3.17 CS8900_Ethernet



[그림 4.23]

CS8900 이더넷 테스트이다. 이 메뉴에서는 단지 CS8900 ID를 읽어봄으로써, 칩의 동작 여부를 확인한다. 실제 CS8900의 송수신은 MBA2440용 u-boot 를 통해 확인할 수 있다.

4.3.18 USB download



[그림 4.24]

USB를 통해 데이터를 SDRAM에 다운로드 하기 위한 메뉴이다.

1번은 실행 바이너리 파일을 다운로드하고 바로 실행 시킨다.

2번은 단지 파일을 다운로드하기만 한다. 2번을 이용해 다운 받은 파일을 NOR flash나 NAND flash에 write할 수 있다.

Test program을 boot flash의 0x0 번지에 적재하고 부팅한 경우, 메모리에 다운로드 할 때 **맨 하위 32KB영역과 맨 상위에서 32KB 영역은 사용하지 않는다. 사용하게 되면 MBA2440 test 프로그램이 동작하지 않게 된다.**

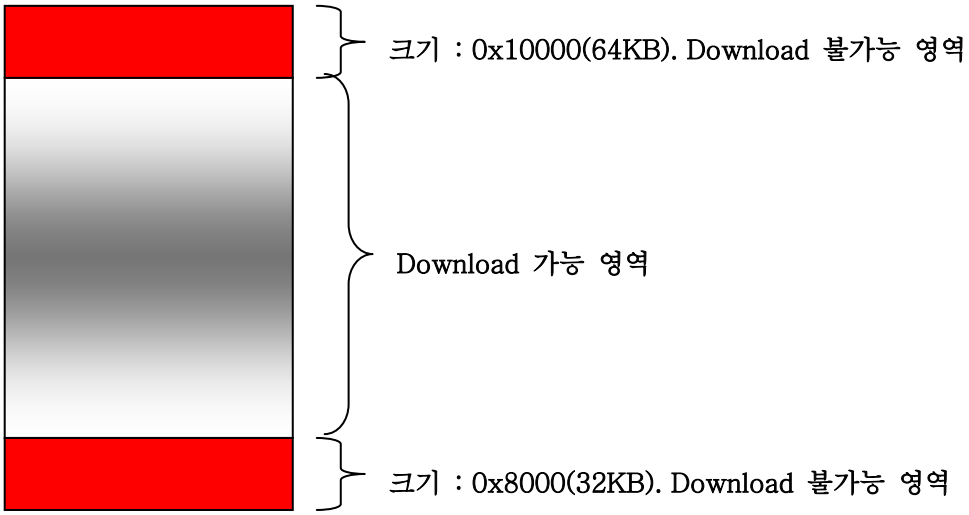
만약, SDRAM이 64MB 크기라면, 0x30000000 ~ 0x30007fff 영역과 0x33ff8000 ~ 0x33ffffff 영역에는 다운로드 하지 않도록 유의한다.

Test program을 SDRAM의 0x30000000 번지에 적재하고 부팅한 경우, 메모리에 다운로드 할 때 **맨 하위 1MB영역과 맨 상위에서 32KB 영역은 사용하지 않는다. 사용하게 되면 MBA2440 test 프로그램이 동작하지 않게 된다.**

만약, SDRAM이 64MB 크기라면, 0x30000000 ~ 0x300fffff 영역과 0x33ff8000 ~ 0x33ffffff 영역에

는 다운로드 하지 않도록 유의한다.

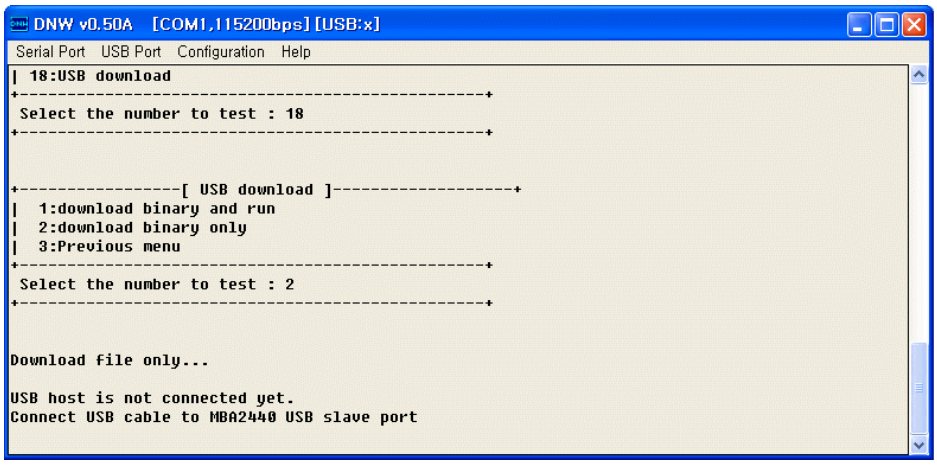
Test program을 boot flash의 0x0 번지에 적재하고 부팅한 경우에, 다운로드 영역은 [그림 4.25]와 같다. [그림 4.25]은 MBA2440 test 프로그램의 소스를 컴파일 할 때, RO_BASE를 0x0으로 설정하고 컴파일 했을 경우이다. 즉, MBA2440 test 프로그램 코드가 AMD flash상에서 동작하고 있을 때이다.



[그림 4.25]

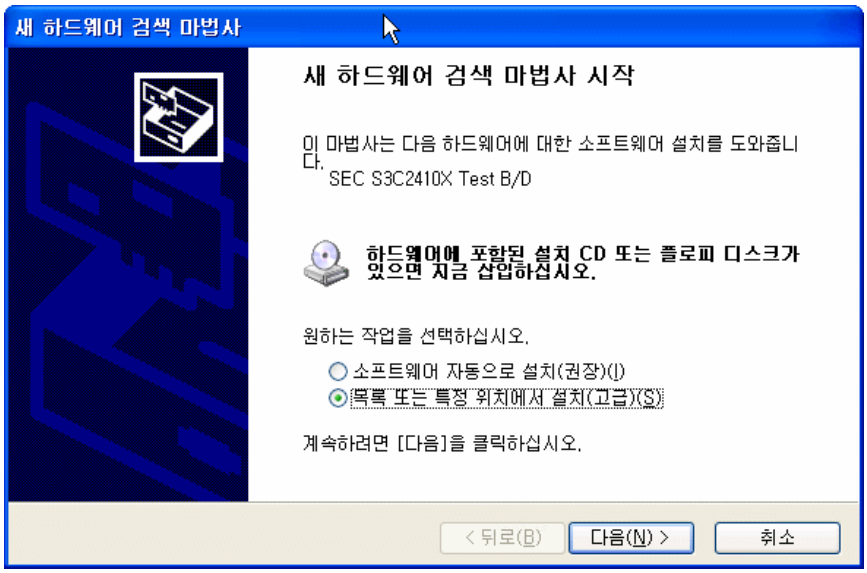
맨 처음 보드를 구매한 후에, 이 메뉴를 테스트한다면, Windows OS에 USB driver를 설치해 주어야 한다.

메뉴의 2번을 선택한다.



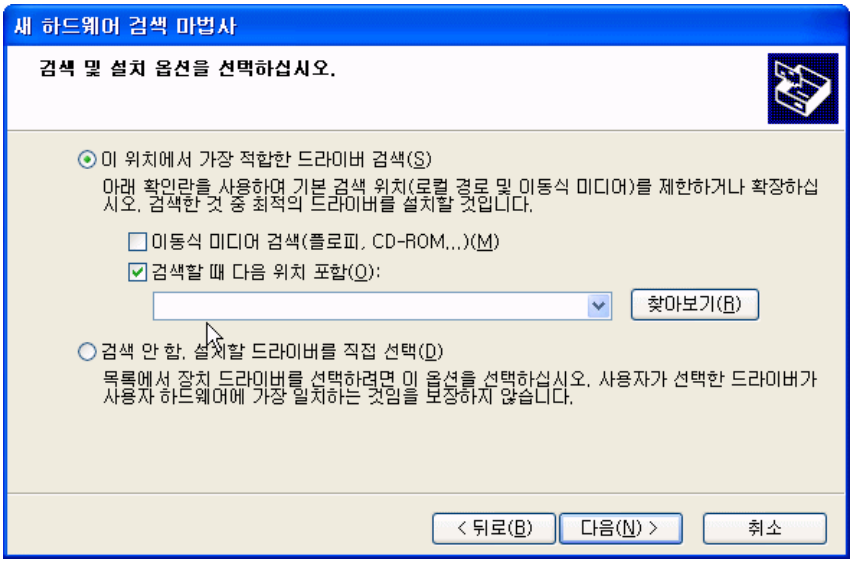
[그림 4.26]

PC와 MBA2440 간에 USB 케이블을 연결하면, 잠시 후, PC에서 새 하드웨어 발견 메시지가 보이고 새 하드웨어 검색 마법사 창이 뜬다.



[그림 4.27]

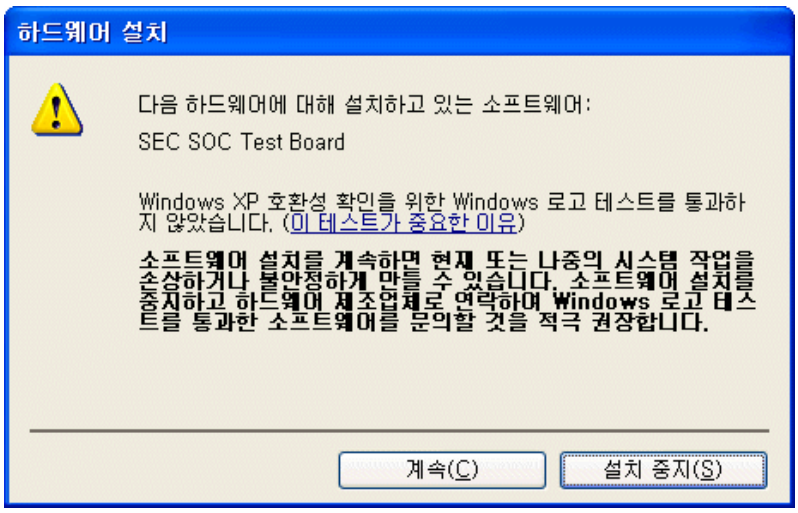
[그림 4.27]와 같이 선택한 후, 다음 버튼을 누른다.



[그림 4.28]

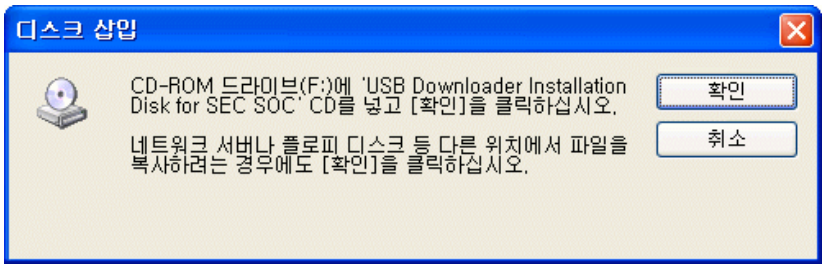
[그림 4.28]에서 찾아보기를 누르고 제공한 CD의 utility/USB_driver 디렉토리를 선택한 후, 다음 버튼을 누른다.

하드웨어를 검색하게 된다. 중간에 [그림 4.29]과 같은 창이 뜨는데 계속 버튼을 누른다.



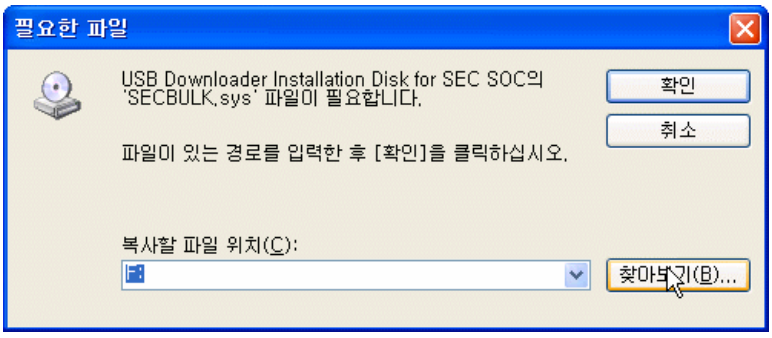
[그림 4.29]

[그림 4.30]과 같은 창이 뜬다면, 확인 버튼을 누른다



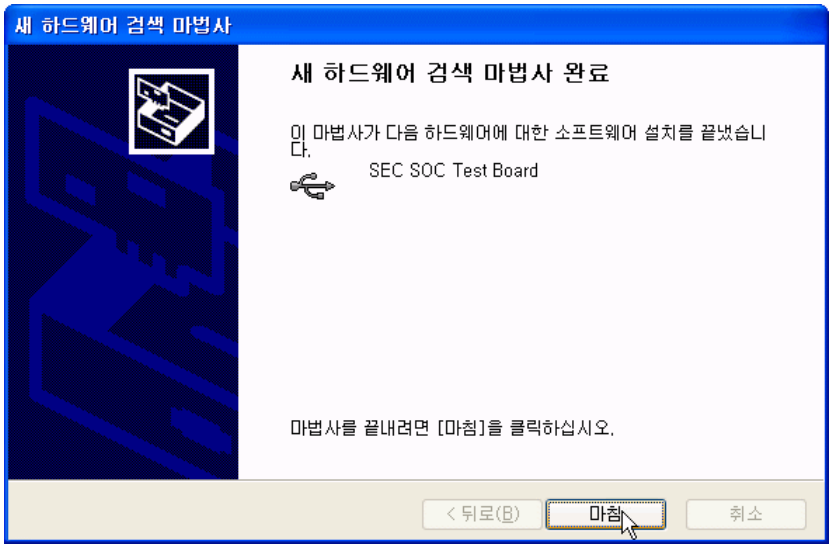
[그림 4.30]

[그림 4.31]과 같은 창이 뜨면 찾아보기에서 제공한 CD의 utility/USB_driver 디렉토리에 있는 secbulk.sys 파일을 찾아 선택한다.



[그림 4.31]

[그림 4.32]과 같이 마법사 완료창이 뜨면 마침 버튼을 눌러 종료한다.



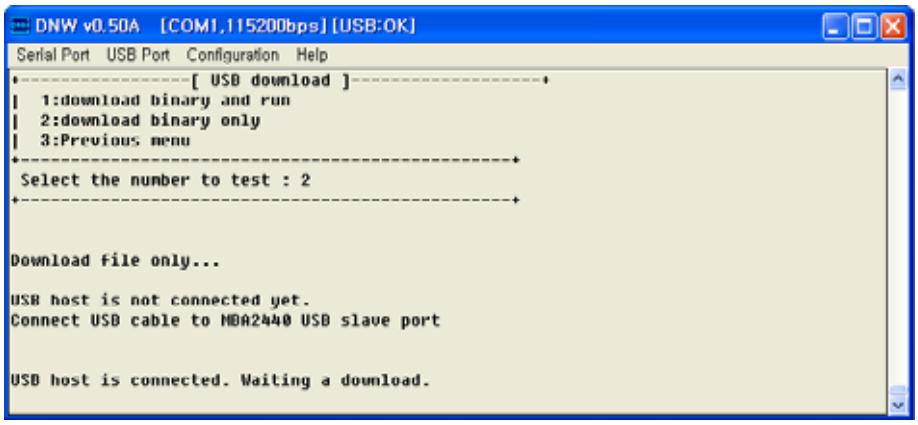
[그림 4.32]

이제 파일을 다운로드해 보자.

원하는 메뉴를 선택한다.

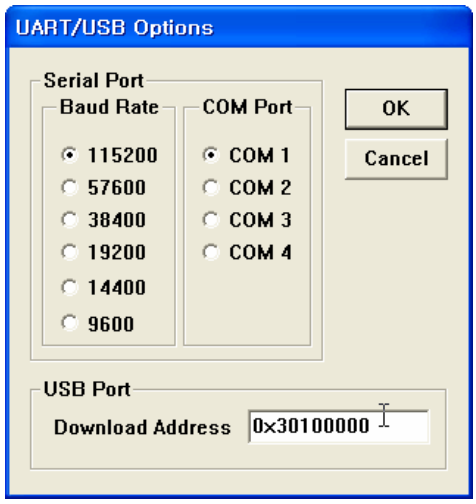
만약, USB SLAVE port에 USB 케이블이 연결되어 있지 않으면 케이블을 연결하라는 메시지가 출력될 것이다.

케이블이 연결되어 있다면, download를 받기 위해 대기한다.



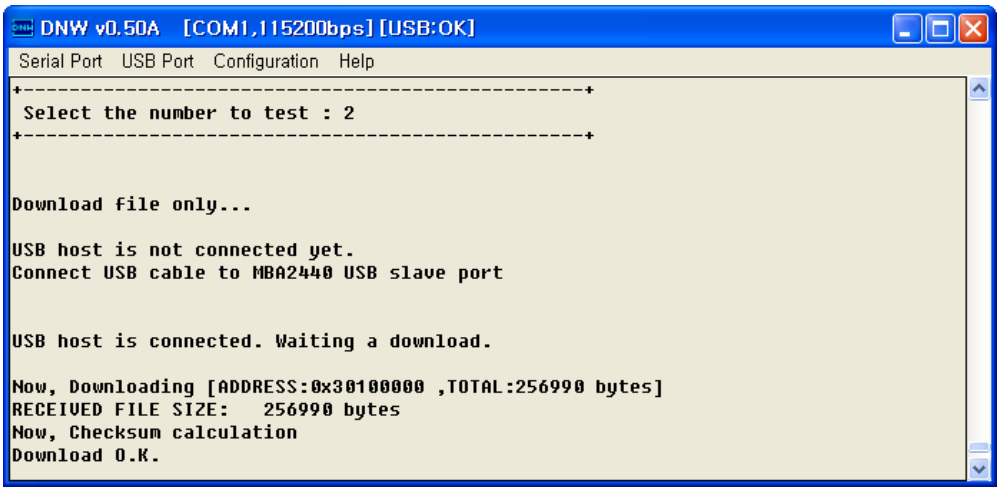
[그림 4.33]

DNW의 Configuration->Option에서 다운로드하고자 하는 주소를 설정해 준다. 이 주소는 MBA2440의 메모리 영역의 주소를 말하는 것이며, 앞서 설명한 바와 같이 **맨 하위 32KB 영역과 맨 상위에서 32KB 영역의 주소는 설정하지 않는다.** Download Address에 주소를 설정한 후 OK버튼을 누른다.



[그림 4.34]

DNW의 USB Port->Transmit에서 보내고자 하는 파일을 선택하면 파일이 전송된다.



[그림 4.35]

5 U-Boot

MBA2440을 위해 제공되는 부트로더는 u-boot, eboot, nboot가 있다.

- **u-boot 부트로더**
  : 리눅스를 위한 부트로더. Open source인 u-boot를 MBA2440에 맞게 수정한 부트로더
  MBA2440에서 가장 핵심이 되는 부트로더이다. 이것은 임베디드 리눅스 부팅뿐만 아니라, WinCE 이미지를 부팅할 때에도 사용할 수 있다.
- **eboot 부트로더:** 7.2 Eboot boot loader 참조
  : MBA2440용 WinCE BSP에 포함되어 있는 부트로더로서 WinCE 개발시 사용되는 부트로더
- **nboot 부트로더:** 7.3 Nboot boot loader 참조
  : boot loader라기 보다 단순히 NAND flash를 boot flash로 설정한 경우에 WinCE 이미지를 메모리에 올리고 부팅시키기 위해 만든 firmware

이 장에서는 MBA2440에서 가장 중요한 부트로더인 U-boot에 대해서 다룬다.

먼저, 부트로더가 무엇인지에 대해서 알아보고, MBA2440의 핵심 부트로더인 u-boot에 대하여 알아본다.

5.1 부트로더란?

부트로더란 일반 데스크탑 PC로 비유하여 설명하자면, BIOS와 같은 역할을 하는 firmware이다.

부트로더는 전원이 켜질 때, 제일 먼저 동작하는 프로그램으로서 MBA2440의 MCU와 그 외에 여러 가지 device들(메모리 등등)을 초기화 하고 OS가 동작할 수 있는 기본적인 환경을 구성해 주는 역할을 담당한다.

이 외에도, OS를 통해 제어하기 이전에 하드웨어의 동작 여부를 테스트할 수 있는 명령어를 개발자가 직접 구현하여 하드웨어를 검증할 수도 있다.

MBA2440에서는 u-boot 1.1.1 버전을 기본으로 하여 MBA2440에 맞게 수정한 부트로더를 제공하고 있다.

u-boot의 모든 개발은 리눅스 PC(Redhat 9.0)상에서 이루어졌으며, 이 장에서의 설명도 리눅스 PC의 개발환경을 기본으로 설명할 것이다.

5.2 MBA2440용 U-boot

MBA2440에 맞게 수정된 소스인 **mba2440-uboot1.1.1.tar.bz2** 파일을 CD에 linux/u-boot 디렉토리에서 볼 수 있다.

mba2440-uboot1.1.1에서는 가장 기본적이며 중요한 다음과 같은 기능들을 가지고 있다.

- memory 읽기, 쓰기, 지우기 기능

MBA2440에서 제공되는 메모리인 SDRAM, NAND flash, NOR flash를 제어할 수 있는 명령어들이 있다.

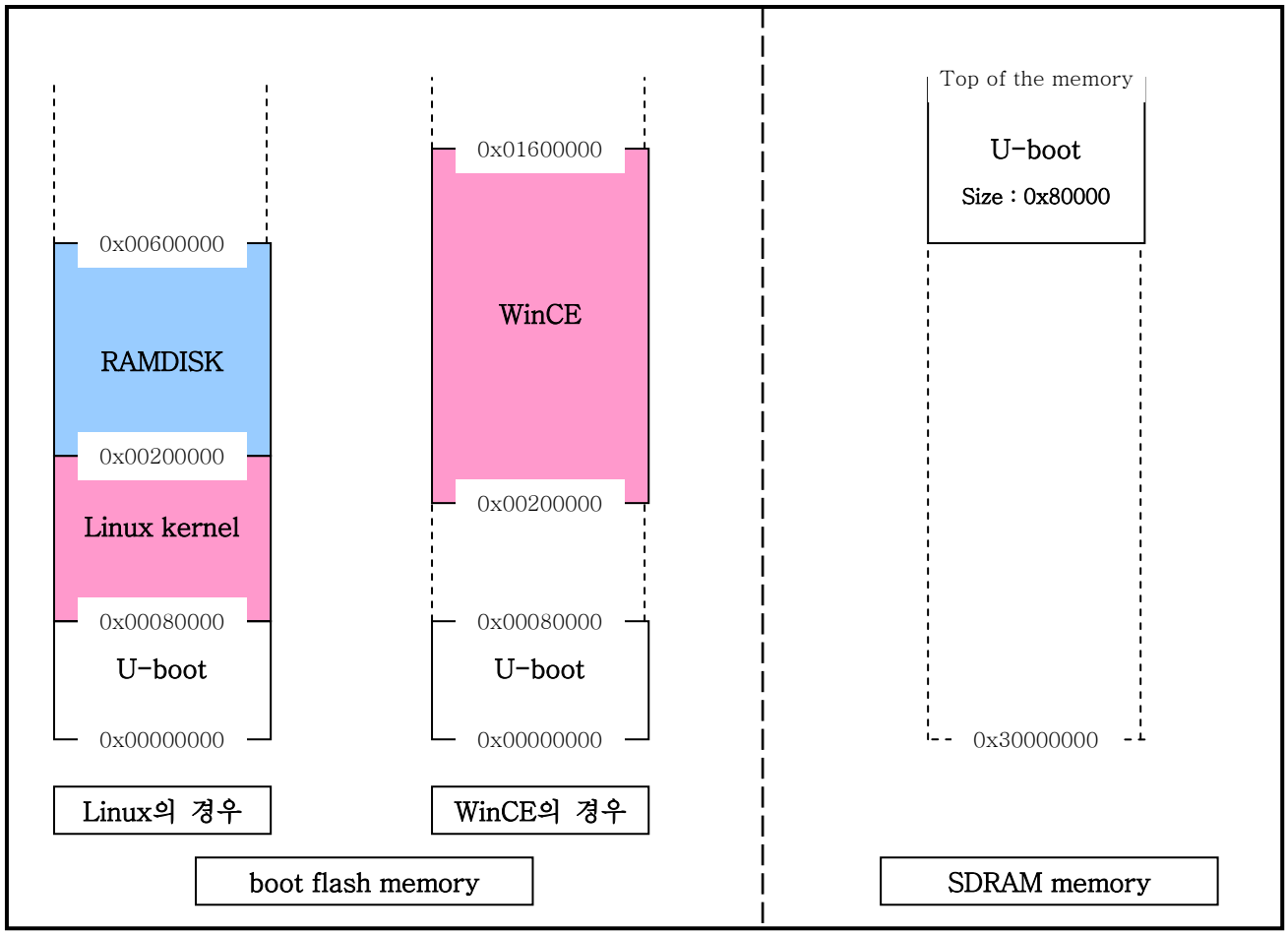
- network 통신 기능

MBA2440에서는 ping 명령을 통해 host PC와의 연결 여부를 확인하고, tftp로 시스템 메모리 가용 범위 안에서 파일을 수신 받을 수 있다.

이 외에도 여러 가지 부수적인 명령어들이 제공되고 있다.

5.2.1 U-Boot memory map

U-boot는 보드가 부팅된 후 수행되면서 [그림 5.1]과 같은 메모리맵을 구성하게 된다.



[그림 5.1]

[그림 5.1]은 U-boot가 boot flash(NAND flash memory인 경우)에 대해서 인식하고 있는 메모리 맵

과 SDRAM에 대해서 인식하고 있는 메모리 맵이다.

(Boot flash가 AMD flash인 경우에는 flash 메모리 크기가 1MB 밖에 안되기 때문에 OS 이미지나 파일 시스템 이미지는 적재할 수 없을을 유의한다.)

리눅스의 경우, linux kernel은 0x80000 ~ 0x17ffff 영역(1.5MB 크기) 안에 있어야 "bootrom kernel" 명령이 제대로 동작하게 된다.

마찬가지로, **fliesystem(램디스크 이미지)은 0x200000 ~ 0x5fffff 영역(4MB 크기) 안에 있어야 "bootrom kernel" 명령이 제대로 수행**된다.

WinCE의 경우, WinCE이미지가 0x200000 ~ 0x15ffffff 영역(20MB 크기) 안에 존재해야 "bootrom wince" 명령이 제대로 동작할 수 있다.

u-boot는 처음 동작하자 마자 SDRAM 메모리(시스템 메모리)의 최상위 번지에서 0x80000(512KB 크기) 만큼의 공간에 자신의 실행 이미지를 복사 한 후, 복사한 그 위치에서 하드웨어 초기화등의 나머지 작업들을 수행하게 된다.

5.2.2 U-Boot 명령어들

이 장에서는 u-boot 명령어들에 대해서 알아본다. U-boot-1.1.1 버전의 소스를 수정하여 개발하였으며, 몇몇 명령어들은 MBA2440에 맞게 수정되었다.

이 장에서는 모든 U-boot 명령어들을 소개하지 않으며, 소개하는 명령들에 대해서도 자세하게 다루지는 않는다. 명령어에 대해서는 u-boot 소스를 분석하기 바란다.

5.2.2.1 U-boot Information command

1) bdinfo : Board Info 구조체를 보여준다. bd 구조체는 u-boot에서 Target Board에 관한 하드웨어, 메모리 구성 등의 정보를 가지고 있다.

```
MBA2440 #
MBA2440 # bdinfo
arch_number = 0x000000C1
env_t       = 0x00000000
boot_params = 0x30000100
DRAM bank   = 0x00000000
-> start    = 0x30000000
-> size     = 0x04000000
ethaddr     = 08:00:3E:26:0A:5B
ip_addr     = 192.168.100.5
baudrate    = 115200 bps
MBA2440 #
[영어][완성][두벌식]
```

[그림 5.2]

2) coninfo : 콘솔의 정보를 보여준다.

```
MBA2440 # coninfo
List of available devices:
serial   80000003 SIO stdin stdout stderr
MBA2440 #

[영어][완성][두벌식]
```

[그림 5.3]

3) help : 전체 명령어를 볼 수 있다.

```
MBA2440 # help
?       - alias for 'help'
autolinux - get default 'linux' image to 0x00020000
autoscr - run script from memory
base    - print or set address offset
bdinfo  - print Board Info structure
boot    - boot default, i.e., run 'bootcmd'
bootd   - boot default, i.e., run 'bootcmd'
bootelf - Boot from an ELF image in memory
bootm   - boot application image from memory
bootp   - boot image via network using BootP/TFTP protocol
bootrom   - just typing bootrom and enter
bootvx  - Boot vxWorks from an ELF image
cmp     - memory compare
coninfo - print console devices and informations
cp      - memory copy
crc32   - checksum calculation
date    - get/set/reset date & time
dcache  - enable or disable data cache
echo    - echo args to console
flerase   - erase NOR FLASH memory
flinfo  - print NOR FLASH memory information
flread   - read NOR FLASH memory
flwrite   - write NOR FLASH memory
go      - start application at address 'addr'
help    - print online help
icache  - enable or disable instruction cache
iminfo  - print header information for application image
imls    - list all images found in flash
itest    - return true/false on integer compare
loadb   - load binary file over serial line (kermit mode)
loadm  - load a image from flash rom to memory
loads   - load S-Record file over serial line
loop    - infinite loop on address range
md      - memory display
me      - memory erase
mm      - memory modify (auto-incrementing)
mtest   - simple RAM test
mw      - memory write (fill)
nandb   - MBA2410 NAND Flash Bad block check program
nande   - MBA2410 NAND Flash Erase program : erase blocks
nandr   - MBA2410 NAND Flash Read program : read and copy to memory
nandw   - MBA2410 NAND Flash Write program : write from memory to NAND flash
nfs    - boot image via network using NFS protocol
nm      - memory modify (constant address)
ping    - send ICMP ECHO_REQUEST to network host
printenv- print environment variables
protect - enable or disable OR FLASH write protection
rarpboot- boot image via network using RARP/TFTP protocol
reset   - Perform RESET of the CPU
run     - run commands in an environment variable
saveenv - save environment variables to persistent storage
setenv  - set environment variables
sleep   - delay execution for some time
tftpboot- boot image via network using TFTP protocol
version - print monitor version
MBA2440 #

CTRL-A Z for help |115200 8N1 | NOR | Minicom 2.00.0 | VT102 |      Offline
[영어][완성][두벌식]
```

[그림 5.4]

5.2.2.2 Memory command

1) base : Base Address로부터 Offset을 설정한다. base가 0x10000이면 실지 SDRAM주소는 0x30010000번지가 시작번지다.

```
MBA2440 # help base
base
    - print address offset for memory commands
base off
    - set address offset for memory commands to 'off'

MBA2440 # base
Base Address: 0x00000000
MBA2440 # 
[영어][완성][두벌식]
```

[그림 5.5]

2) crc32 : address부터 count까지의 체크섬(checksum)을 계산한다.

```
MBA2440 #
MBA2440 #
MBA2440 # help crc32
crc32 address count [addr]
    - compute CRC32 checksum [save at addr]

MBA2440 # crc32 0x31000000 0x100
CRC32 for 31000000 ... 310000ff ==> 0d968558
MBA2440 # 
[영어][완성][두벌식]
```

[그림 5.6]

3) mtest : start에서 end 주소까지 read/write 테스트를 한다. 주의할 점은 RAM영역만 가능하기 때문에 NOR, NAND는 지원하지 않는다.

```
MBA2440 #
MBA2440 #
MBA2440 #
MBA2440 # help mtest
mtest [start [end [pattern]]]
    - simple RAM read/write test

MBA2440 # mtest 0x30100000 0x30200000
Pattern FFFFFFDD  Writing...  
[영어][완성][두벌식]
```

[그림 5.7]

4) cp : 메모리의 source 주소에서 target 주소로 count만큼을 복사한다.

옵션 : .b(8 bit) .w(16 bit) .l(32 bit) 단위를 지원

```
MBA2440 #
MBA2440 #
MBA2440 # help cp
cp [.b, .w, .l] source target count
    - copy memory

MBA2440 # cp 0x30100000 0x30200000 0x1000
Copy from 0x30100000 to 0x30200000... done
MBA2440 # 
[영어][완성][두벌식]
```

[그림 5.8]

5) md : 지정된 메모리 주소부터 objects 수 만큼의 메모리 값을 출력한다. 여기서 objects의 수는 4바이트 크기이다.

옵션 : .b(8 bit) .w(16 bit) .l(32 bit) 단위를 지원

예) md.b 0x30100000 0x1000

=> 이것은 0x30100000 부터 0x1000 사이즈 만큼 byte 단위로 출력하라는 뜻

```
MBA2440 #
MBA2440 # help md
md [.b, .w, .l] address [# of objects]
    - memory display

MBA2440 # md 0x30002000 0x20
30002000: 6c656420 5b207961 35322d31 203a5d35     delay [1-255]:
30002010: 00000000 4e524157 3a474e49 74655320    ....WARNING: Set
30002020: 39385343 414d3030 64644143 73736572    CS8900MACAddress
30002030: 6e49203a 696c6176 414d2064 64612043    : Invalid MAC ad
30002040: 73657264 0a0d2e73 00000000 4f464e49    dress.......INFO
30002050: 414d203a 64612043 73657264 65732073    : MAC address se
30002060: 6f742074 7825203a 3a78253a 253a7825    t to: %x:%x:%x:%
30002070: 78253a78 0d78253a 0000000a 6e450a0d    x:%x:%x.......En
MBA2440 #
[영어][완성][두벌식]
```

[그림 5.9]

7) me : U-boot 탑재 영역을 제외한 특정 메모리 또는 모든 메모리 영역을 지운다. 메모리 영역 테스트와 관련된 작업시 유용하다.

예) me all

=> U-boot가 실행중인 영역을 제외한 모든 영역을 지운다.

```
MBA2440 # help me
me me[.b, .w, .l] start_add  size  - start_addr and size is hex value
me all  - erase all memory except the area of u-boot running code

MBA2440 # me 0x30000000 0x1000
erasing memory .. start address: 0x30000000, size: 0x1000(4096 bytes)
.
erase completed..

MBA2440 #
[영어][완성][두벌식]
```

[그림 5.10]

```
MBA2440 #
MBA2440 #
MBA2440 #
MBA2440 #
MBA2440 # me all
erasing memory .. start address: 0x30000000, size: 0x3f00000(66060288 bytes)
.....................................................................................
erase completed..

MBA2440 #
[영어][완성][두벌식]
```

[그림 5.11]

5.2.2.3 NAND flash command

1) nandb : 현재 사용하고 있는 NAND 플래시 메모리의 블록 상태를 체크하는 명령

nandb all : NAND 플래시 메모리의 모든 블록을 체크

```
MBA2440 # help nandb
nandb  - MBA2410 NAND Flash Bad block check program
    nandb startblock(0~4095) endblock(0~4095)
    nande blocknumber(0~4095)  - check one block you select
    nande all                  - check all blocks

MBA2440 # nandb all
check all blocks [0 ~ 4095]
Display block status :
All block is OK..


MBA2440 #
[영어][완성][두벌식]
```

[그림 5.12]

nandb [block 번호] : NAND 플래시 메모리의 해당 블록 하나를 체크

```
MBA2440 #
MBA2440 # nandb 300
check block number [300]
Display block status :
block[ 300] is OK..


MBA2440 # ▌
[영어][완성][두벌식]
```

[그림 5.13]

nandb [block 시작번호] [block 끝번호] : 시작번호 블록에서 끝번호 블록까지 체크

```


MBA2440 # nandb 340 399
erase block [340 ~ 399]
Display block status :block[ 340] ~ block[ 399] OK..


MBA2440 # ▌
[영어][완성][두벌식]
```

[그림 5.14]

2) nande : 현재 사용하고 있는 NAND SMC 플래시 메모리의 블록 상태를 지우는 명령

nande all : NAND 플래시 메모리의 모든 영역을 지우는 명령

```
MBA2440 # nande all
erase all blocks [0 ~ 4095]
Erasing block..\
Complete erasing block


MBA2440 # ▌
[영어][완성][두벌식]
```

[그림 5.15]

nande [block 번호] : NAND 플래시 메모리의 해당 블록을 지우는 명령

```
MBA2440 #
MBA2440 # nande b 0
erase block [0 ~ 0]
Erasing block..-
Complete erasing block


MBA2440 # ▌
[영어][완성][두벌식]
```

[그림 5.16]

nande [block 시작번호] [block 끝번호] : NAND 플래시 메모리의 지정된 시작번호 블록에서 끝번호 블록까지 지우는 명령

```
MBA2440 #
MBA2440 # nande 10 200
erase block [10 ~ 200]
Erasing block../
Complete erasing block


MBA2440 # ▌
[영어][완성][두벌식]
```

[그림 5.17]

3) nandr : 현재 사용하고 있는 NAND 플래시 메모리의 블록을 메모리로 읽어오는 명령

nandr p [mem_addr] [block 번호] [page 번호] : NAND 플래시 메모리에서 해당 블록 내에

있는 해당 page를 읽어서 mem_addr에 저장 하는 명령

```
MBA2440 #
MBA2440 #
MBA2440 #
MBA2440 #
MBA2440 # nandr p 0x30000000  1 10
read page[block:1, page:10].. copy to memory address 0x30000000, size is 0x200
MBA2440 #
[영어][완성][두벌식]
```

[그림 5.18]

nandr b [mem_addr] [block 번호] : NAND 플래시 메모리에서 해당 블록을 읽어서 mem_addr에 저장하는 명령

```
MBA2440 #
MBA2440 #
MBA2440 # nandr b 0x31000000 2
read block[2].. copy to memory address 0x31000000, size is 0x4000
\
Reading block is Completed

MBA2440 #
[영어][완성][두벌식]
```

[그림 5.19]

nandr t [mem_addr] [nand flash의 시작주소] [크기] : NAND 플래시 메모리에서 읽고자 하는 영역의 시작주소를 기점으로 지정된 크기만큼 읽어서 mem_addr에 저장하는 명령. 단 시작주소나 크기는 최소 page 단위이어야 한다.

```
MBA2440 #
MBA2440 #
MBA2440 #
MBA2440 # nandr t 0x30100000 0x400 0x200
Read start point : block[0] page[2] size = 0x200
Reading data ...\
Read compelete(0x200): from 0x400(0) NAND flash to 0x30100000 SDRAM
MBA2440 #
[영어][완성][두벌식]
```

[그림 5.20]

4) **nandw** : 현재 사용하고 있는 NAND 플래시 메모리의 블록을 메모리로 쓰는 명령

nandw p [mem_addr] [block 번호] [page 번호] : 메모리 상에 있는 mem_addr 주소를 시작주소로 page 크기 만큼을 읽어서 NAND 플래시 해당 블록의 해당 페이지에 쓰는 명령

```
MBA2440 #
MBA2440 #
MBA2440 #
MBA2440 #
MBA2440 #
MBA2440 # nandw p 0x30000000 1 10
write page[block:1, page:10].. read from memory address 0x30000000, size is 0x200
MBA2440 #
[영어][완성][두벌식]
```

[그림 5.21]

nandw b [mem_addr] [block 번호] : 메모리 상에 있는 mem_addr 주소를 시작주소로 한 블

록 크기를 읽어서 NAND 플래시 해당 블록에 쓰는 명령

```
MBA2440 #
MBA2440 #
MBA2440 # nandw b 0x30100000 4
write block[4].. read from memory address 0x30100000, size is 0x4000
-
Writing block is Completed

MBA2440 # ▮
[영어][완성][두벌식]
```

[그림 5.22]

nandw t [mem_addr] [nand flash의 시작주소] [크기] : 메모리 상에 있는 mem_addr 주소를 시작주소로 지정된 크기만큼 읽어서 NAND 플래시의 시작주소부터 크기만큼 쓴다. 단, 시작주소나 크기는 최소 page 단위이어야 한다.

```
MBA2440 #
MBA2440 #
MBA2440 #
MBA2440 # nandw t 0x30000000 0x0 0x20000
Write start point : block[0] page[0] size = 0x20000
Writing data .../
Write compelete(0x20000): from 0x30000000 SDRAM to 0x0(0) NAND flash
MBA2440 # ▮
[영어][완성][두벌식]
```

[그림 5.23]

5.2.2.4 NOR flash command

1) flinfo : NOR flash memory에 대한 정보를 나타낸다.

flinfo [bank number] : u-boot 부팅시 NOR flash information에서 표시되는 bank 번호를 입력하면 해당 NOR flash의 정보를 출력한다.
Bank 번호를 입력하지 않으면 연결된 모든 NOR flash의 정보를 출력한다.

```
MBA2440 #
MBA2440 # flinfo

Bank # 0: AMD AM29LV800BB
  Size: 0x100000 Bytes, 19 Sectors
  Sector Start Addresses:
00000000(RO)  00004000(RO)  00006000(RO)  00008000(RO)
00010000(RO)  00020000      00030000      00040000
00050000      00060000      00070000      00080000
00090000      000A0000      000B0000      000C0000
000D0000      000E0000      000F0000(RO)

Bank # 1: INTEL 28F128J3A
  Size: 0x2000000 Bytes, 128 Sectors
  Sector Start Addresses:
08000000      08040000      08080000      080C0000
08100000      08140000      08180000      081C0000
08200000      08240000      08280000      082C0000
08300000      08340000      08380000      083C0000
08400000      08440000      08480000      084C0000
08500000      08540000      08580000      085C0000
08600000      08640000      08680000      086C0000
08700000      08740000      08780000      087C0000
08800000      08840000      08880000      088C0000
08900000      08940000      08980000      089C0000
08A00000      08A40000      08A80000      08AC0000
08B00000      08B40000      08B80000      08BC0000
08C00000      08C40000      08C80000      08CC0000
08D00000      08D40000      08D80000      08DC0000
08E00000      08E40000      08E80000      08EC0000
08F00000      08F40000      08F80000      08FC0000
09000000      09040000      09080000      090C0000
09100000      09140000      09180000      091C0000
09200000      09240000      09280000      092C0000
09300000      09340000      09380000      093C0000
09400000      09440000      09480000      094C0000
09500000      09540000      09580000      095C0000
09600000      09640000      09680000      096C0000
09700000      09740000      09780000      097C0000
09800000      09840000      09880000      098C0000
09900000      09940000      09980000      099C0000
09A00000      09A40000      09A80000      09AC0000
09B00000      09B40000      09B80000      09BC0000
09C00000      09C40000      09C80000      09CC0000
09D00000      09D40000      09D80000      09DC0000
09E00000      09E40000      09E80000      09EC0000
09F00000      09F40000      09F80000      09FC0000
MBA2440 #
CTRL-A Z for help |115200 8N1 | NOR | Minicom 2.00.0 | VT102 |      Offline
[영어][완성][두벌식]
```

[그림 5.24]

2) flread : NOR flash에서 데이터를 읽어서 메모리에 복사한다.

flread [NOR flash 주소] [메모리 주소] [size] : NOR flash 메모리의 주소를 시작점으로 size 만큼 읽어서 메모리 주소에 복사한다.

NOR flash의 메모리 주소는 flinfo를 통해 알 수 있다.

```
MBA2440 #
MBA2440 #
MBA2440 #
MBA2440 #
MBA2440 #
MBA2440 #
MBA2440 #
MBA2440 # flread 0x8000000 0x30000000 0x20000
Read data from INTEL flash
read from 0x08000000 and copy to 0x30000000 ... \
Complete read data!! size : 0x20000


MBA2440 #
[영어][완성][두벌식]
```

[그림 5.25]

2) flwrite : NOR flash에 데이터를 write한다. write하기 전에 항상 먼저 해당 블록을 flerase 명령을 통해 지운다.

flwrite [메모리 주소] [NOR flash 주소] [size] : 메모리 주소를 시작점으로 size 만큼을 읽어

서 NOR flash 메모리의 주소에 write한다. NOR flash의 메모리 주소는 flinfo를 통해 알 수 있다.
데이터를 write하기 전에 항상 먼저 해당 블록을 flerase 명령을 통해 지운다.

```
MBA2440 #
MBA2440 #
MBA2440 #
MBA2440 #
MBA2440 # flerase 1:0-4
Erase Flash Sectors 0 - 4 in Bank # 1
Erasing INTEL flash now...


Erasing sector   0 ... complete
Erasing sector   1 ... complete
Erasing sector   2 ... complete
Erasing sector   3 ... complete
Erasing sector   4 ... complete
MBA2440 #
MBA2440 # flwrite 0x30000000 0x8000000 0x20000
Write data to INTEL flash
read from 0x30000000 and write to 0x08000000 ... \
Complete write data!! size : 0x20000


MBA2440 # ▉
[영어][완성][두벌식]
```

[그림 5.26]

3) flerase : NOR flash 메모리를 블록 단위로 삭제한다.

flerase bank all : u-boot 부팅시 인식된 모든 NOR flash 메모리 전체를 지운다.

flerase bank [bank 번호] : bank 번호에 해당하는 NOR flash 메모리 전체를 지운다.

flerase bank [bank 번호]:[시작 블록]-[끝 블록] : bank 번호에 해당하는 NOR flash에서 시작 블록 에서 끝 블록 안에 해당 하는 블록만 지운다.

예) flerase 1:0-9 => bank 1에 해당하는 NOR flash의 0번 블록 ~ 9번 블록을 지운다.

```
MBA2440 # flerase 1:0-9
Erase Flash Sectors 0 - 9 in Bank # 1
Erasing INTEL flash now...


Erasing sector   0 ... complete
Erasing sector   1 ... complete
Erasing sector   2 ... complete
Erasing sector   3 ... complete
Erasing sector   4 ... complete
Erasing sector   5 ... complete
Erasing sector   6 ... complete
Erasing sector   7 ... complete
Erasing sector   8 ... complete
Erasing sector   9 ... complete
MBA2440 # ▉

[영어][완성][두벌식]
```

[그림 5.27]

4) protect : NOR flash 메모리의 블록을 lock 또는 unlock한다. 블록이 lock으로 설정되면 write가 안 된다.

protect [on/off] all : on은 블록을 lock 시킨다. off는 블록을 unlock 시킨다.
all은 u-boot 부팅시 인식된 모든 NOR flash의 모든 블록을 lock 또는 unlock 시킨다.

protect [on/off] bank [bank 번호] : on은 블록을 lock 시킨다. off는 블록을 unlock 시킨다.
Bank 번호에 해당하는 NOR flash의 모든 블록을 lock 또는 unlock 시킨다.

protect [on/off] [bank 번호]:[시작 블록]-[끝 블록] : on은 블록을 lock 시킨다. off는 블록을 unlock 시킨다.
bank 번호에 해당하는 NOR flash에서 시작 블록에서 끝 블록 안에 해당 하는 블록만 lcok 또는 unlock 시킨다.

```
MBA2440 #
MBA2440 # flinfo 0

Bank # 0: AMD AM29LV800BB
  Size: 0x100000 Bytes, 19 Sectors
  Sector Start Addresses:
00000000(RO) 00004000(RO) 00006000(RO) 00008000(RO)
00010000(RO) 00020000     00030000     00040000
00050000     00060000     00070000     00080000
00090000     000A0000     000B0000     000C0000
000D0000     000E0000     000F0000(RO)
MBA2440 #
MBA2440 # protect off bank 0
Un-Protect Flash Bank # 0
MBA2440 #
MBA2440 # flinfo 0

Bank # 0: AMD AM29LV800BB
  Size: 0x100000 Bytes, 19 Sectors
  Sector Start Addresses:
00000000     00004000     00006000     00008000
00010000     00020000     00030000     00040000
00050000     00060000     00070000     00080000
00090000     000A0000     000B0000     000C0000
000D0000     000E0000     000F0000
MBA2440 #
 CTRL-A Z for help |115200 8N1 | NOR | Minicom 2.00.0 | VT102 |      Offline
[영어][완성][두벌식]
```

[그림 5.28]

5.2.2.5 Execute command

1) go : Serial Port나 Network을 통해 다운받은 어플리케이션의 시작 주소를 지정해 줌으로써 실행한다. U-Boot에서는 console I/O나 malloc(), free()등의 기본적인 시스템 함수와 하드웨어 함수를 미리 작성해두면 어플리케이션에서 사용할 수 있다.

```
MBA2440 #
MBA2440 # go 30f00000
## Starting application at 0x30F00000 ...
Uncompressing Linux..................................................... done, booting the.
Linux version 2.4.20_elfin-d1.5 (root@localhost.localdomain) (gcc version 2.95.3 20010315 (rT
CPU: ARM/CIRRUS Arm920Tid(wb) revision 0
Machine: MBA2440 AIJI System Co.,LTD1
Warning: bad configuration page, trying to continue
On node 0 totalpages: 8192
zone(0): 8192 pages.
zone(1): 0 pages.
zone(2): 0 pages.
Kernel command line: root=/dev/ram0 rw console=ttyS0
Console: colour dummy device 80x30
Calibrating delay loop... 199.47 BogoMIPS
Use CONFIG_INSTANT_ON_LPJ=997376 for Instant On.
Memory: 32MB = 32MB total
Memory: 26432KB available (1457K code, 305K data, 80K init)
Dentry cache hash table entries: 4096 (order: 3, 32768 bytes)
Inode cache hash table entries: 2048 (order: 2, 16384 bytes)
Mount-cache hash table entries: 512 (order: 0, 4096 bytes)
Buffer-cache hash table entries: 1024 (order: 0, 4096 bytes)
Page-cache hash table entries: 8192 (order: 3, 32768 bytes)
POSIX conformance testing by UNIFIX
Linux NET4.0 for Linux 2.4
Based upon Swansea University Computer Society NET3.039
Initializing RT netlink socket
CPU clock = 399.651840 Mhz, HCLK = 99.912960 Mhz, PCLK = 49.956480 Mhz
PWM-Timers Management Module Loaded.
Disabling the Out Of Memory Killer
[영어][완성][두벌식]
```

[그림 5.29]

2) bootrom : 이 명령은 NAND flash에 리눅스 커널 이미지와 램디스크 이미지 또는 WinCE 이미지가 있을 경우에 사용하는 명령이다. NAND flash memory에서 이미지를 읽어온 후, 해당 커널을 실행시킨다.

옵션 : bootrom linux => 리눅스를 부팅시킨다.
bootrom wince => WinCE를 부팅시킨다.

5.2.2.6 Download command

1) tftp : TFTP Client로 TFTP서버로부터 파일을 다운로드 받아 0x30008000 과 같은 Load Address에 저장한다. Load Address는 tftp에서 바로 지정해주거나 setenv(bootfile, loadaddr)에서 설정 가능하다. 다음으로 주어지는 파라미터가 다운 받을 이미지 이름으로 그림의 예제에서는 zImage가 된다. 이 다운 받을 이미지는 TFTP 서버의 root 디렉토리에 존재하는 파일이어야 한다.

```
MBA2440 # tftp 30f00000 zImage.bin
TFTP from server 192.168.100.2; our IP address is 192.168.100.5
Filename 'zImage.bin'.
Load address: 0x30f00000
Loading:-
done
Bytes transferred = 765536 (bae60 hex)
MBA2440 # ▌

[영어][완성][두벌식]
```

[그림 5.30]

주의 사항) 다운받을 파일은 대 소문자를 명확하게 써 넣어야 한다. 만약 틀린 경우 파일을 제대로 다운 받지 못하는 현상이 생긴다.

5.2.2.7 Environment variable command

1) 환경 변수 리스트

이 리스트는 u-boot가 동작하는 동안 u-boot에 영향을 미치는 환경 변수들이다.

autoload – "no"라고 설정되어 있으면 rarpboot, tftpboot, bootp는 환경변수를 따르지 않고 BOOTP/DHCP서버의 설정을 받는다.
autostart – "yes"로 설정되어 있으면 rarpboot, bootp,dhcp,tftpboot를 자동으로 시작하여 이미지를 다운로드 받고 실행한다.(bootm의 내부적으로 사용됨)
baudrate – Console baudrate를 설정
bootargs – Linux kernel의 부트 변수를 저장
bootcmd – U-Boot 부팅시에 자동으로 실행되는 명령어를 저장
bootdelay – U-Boot 자동 부팅시에 프롬프트 대기시간을 설정
bootfile – TFTP로 다운로드 받은 파일명
ethaddr – MAC Address
ipaddr – MBA2440의 IP Address
loadaddr – TFTP로 다운로드 받을 주소
loads_echo – loads로 다운받는 데이터를 화면으로 echo시킴
pram – Protect RAM
serverip – TFTP Server IP
silent – 부팅시에 화면에 아무것도 출력하지 않음 CONFIG_SILENT_CONSOLE이 선언되어 있어야 한다.
verify – bootm에 정의된 이미지파일 헤더의 Checksum을 계산
bootfile – TFTP/BOOTP/DHCP에 다운로드 받을 파일명
dnsip – DNS IP
gatewayip – Gateway IP
hostname – Hostname
netmask – Netmask
filesize – TFTP/BOOTP/DHCP에서 다운로드 받을 파일 사이즈

2) printenv : 환경변수는 부팅시 실행되는 스크립트, Delay등을 설정하고 네트웍에 관련된 IP Address, Netmask와 TFTP Server IP등이 있다. printenv에서는 각각의 파라메터와 저장 가능한 Environment 크기와 사용량을 출력한다.

```
MBA2440 #
MBA2440 #
MBA2440 #
MBA2440 # printenv
bootcmd=nandr t 30200000 200000 1400000; go wince
bootdelay=3
baudrate=115200
ethaddr=08:00:3e:26:0a:5b
ipaddr=192.168.100.5
serverip=192.168.100.2
netmask=255.255.255.0
stdin=serial
stdout=serial
stderr=serial

Environment size: 211/65532 bytes
MBA2440 #
[영어][완성][두벌식]
```

[그림 5.31]

3) setenv : IP Address, MAC Address, Netmask, Gateway IP를 설정

```
MBA2440 #
MBA2440 #
MBA2440 # setenv ipaddr 10.0.0.1
MBA2440 # printenv
bootcmd=nandr t 30200000 200000 1400000; go wince
bootdelay=3
baudrate=115200
ethaddr=08:00:3e:26:0a:5b
serverip=192.168.100.2
netmask=255.255.255.0
stdin=serial
stdout=serial
stderr=serial
ipaddr=10.0.0.1

Environment size: 206/65532 bytes
MBA2440 #
[영어][완성][두벌식]
```

[그림 5.32]

5.2.2.8 The rest command

1) date : 시스템 날짜를 출력한다.

```
MBA2440 #
MBA2440 # help date
date [MMDDhhmm[[CC]YY][.ss]]
date reset
  - without arguments: print date & time
  - with numeric argument: set the system date & time
  - with 'reset' argument: reset the RTC

MBA2440 # date
Date: 2003-01-01 (Monday)    Time: 13:42:00
MBA2440 # date 061013452005
Date: 2005-06-10 (Friday)    Time: 13:45:00
MBA2440 # date
Date: 2005-06-10 (Friday)    Time: 13:45:05
MBA2440 # date
Date: 2005-06-10 (Friday)    Time: 13:45:10
MBA2440 #
[영어][완성][두벌식]
```

[그림 5.33]

2) echo : 메시지 출력

```
MBA2440 #
MBA2440 #
MBA2440 #
MBA2440 #
MBA2440 #
MBA2440 #
MBA2440 # help echo
echo [args..]
    - echo args to console; \c suppresses newline

MBA2440 # echo MBA2440 reference board
MBA2440 reference board
MBA2440 #
[영어][완성][두벌식]
```

[그림 5.34]

3) sleep : 정해준 초 만큼 sleep

```
MBA2440 #
MBA2440 #
MBA2440 #
MBA2440 # help sleep
sleep N
    - delay execution for N seconds (N is _decimal_ !!!)

MBA2440 # sleep 4
MBA2440 #
[영어][완성][두벌식]
```

[그림 5.35]

4) version : U-Boot 버전을 출력

```
MBA2440 #
MBA2440 # version

U-Boot 1.1.1 (Jun 29 2005 - 22:27:49)
MBA2440 #
[영어][완성][두벌식]
```

[그림 5.36]

5) ? - help와 동일한 명령

5.3 부트로더 컴파일

CD에서 제공하고 있는 부트로더 소스의 이름은 mba2440-uboot1.1.1.tar.bz2 이다.
이 소스를 컴파일 하여 부트로더 이미지를 만드는 방법에 대해 알아 보자.

5.3.1 소스의 압축 풀기

제공한 소스를 임의의 디렉토리 안에 넣어두고, 압축을 해제한다.
여기에서는 리눅스 PC의 "/uboot" 라는 디렉토리를 생성하고 이 디렉토리 안에서 작업을 할 것이다

```
[root@localhost]#  mkdir  /uboot
[root@localhost]#  cd  /uboot
[root@localhost]#  cp  /mnt/cdrom/bootloader/mba2440-uboot1.1.1.tar.bz2  ./
[root@localhost]#  tar  cjf  mba2440-uboot1.1.1.tar.bz2
[root@localhost]#  cd  mba2440-uboot1.1.1
```

5.3.2 U-boot configuration

u-boot는 어떤 플래시 메모리를 boot flash로 사용하느냐에 따라, 시스템 메모리의 크기가 얼마이냐에 따라 컴파일시 조정해야 하는 부분이 있다.

5.3.2.1 Boot flash 설정

u-boot는 NOR flash를 boot flash로 사용하느냐 NAND flash를 boot flash로 사용하느냐에 따라 설정 헤더파일을 수정해 주어야 한다.

이 헤더 파일은 mba2440-uboot1.1.1/include/configs/mba2440.h 이며 수정해야 할 매크로는 다음과 같다.

CONFIG_INIT_CRITICAL

이 매크로는 **boot flash로부터 부팅하려 할 때에는 반드시 "1"로 설정**되어 있어야 한다.
U-boot를 수정하거나 테스트하려 할 때에는 컴파일해서 이미지를 올리는 횟수가 자주 있기 때문에 그 때마다 boot flash에 올리는 것이 번거롭다. 이럴 경우에는 컴파일 된 이미지를 메모리에 바로 적재하게 되는데 이 때에는 값을 "0"으로 설정해 주어야 한다.

CONFIG_NAND_BOOT

이 매크로는 boot flash가 NOR냐 NAND냐를 구분하기 위한 매크로이다.
NOR 플래시[AM29LV800(기본) 또는 28F128J3C(옵션)]를 boot flash로 사용하려 할 경우에는 이 매크로를 "0"으로 설정한 후 컴파일 후, 생성된 이미지를 NOR 플래시의 0번지에 올리면 된다.
NAND 플래시[K9F5608(기본) 또는 SMC(옵션)]를 boot flash로 사용하려 할 경우에는 이 매크로를 "1"으로 설정한 후 컴파일 후, 생성된 이미지를 NAND 플래시의 0번지에 올리면 된다.

5.3.2.2 시스템 메모리 크기 설정

MBA2440은 기본적으로 64MB의 SDRAM을 시스템 메모리로 사용하나, 128MB로 확장할 수 있다. 이러한 특성을 u-boot에 적용하여 시스템 메모리에 따라 컴파일을 다시 해 주어야 부트로더 상에서 메모리를 제대로 사용할 수 있다.

이를 위해서 마찬가지로 mba2440-uboot1.1.1/include/configs/mba2440.h 의 다음 매크로를 수정한다.

CONFIG_MBA2440_MEM_SIZE

64MB일 경우에는 "64"를 128MB일 경우에는 "128"을 설정해 준다.

또 한가지, 메모리상에서 부트로더의 위치를 잡아 주는 매크로를 수정해 주어야 한다.

이 파일은 mba2440-uboot1.1.1/board/mba2440/config.mk 이며, 다음 매크로를 수정한다.

TEXT_BASE

64MB인 경우에는 "0x33F80000"으로, 128MB인 경우에는 "0x37F80000"으로 설정해 준다.

이로써, 기본적인 모든 설정은 끝났다.

5.3.2.3 OS 커널 이미지 자동 부팅

u-boot는 부팅과 동시에 일정 시간을 기다린 후, 원하는 WinCE 또는 리눅스 OS 커널을 부팅시킬 수 있다.

이를 위해서 마찬가지로 mba2440-uboot1.1.1/include/configs/mba2440.h 의 다음 매크로를 수정한다.

CONFIG_BOOTDELAY

이 매크로는 자동 부팅으로 설정했을 경우 기다리는 시간을 초 단위로 설정한다.

CONFIG_BOOTCOMMAND

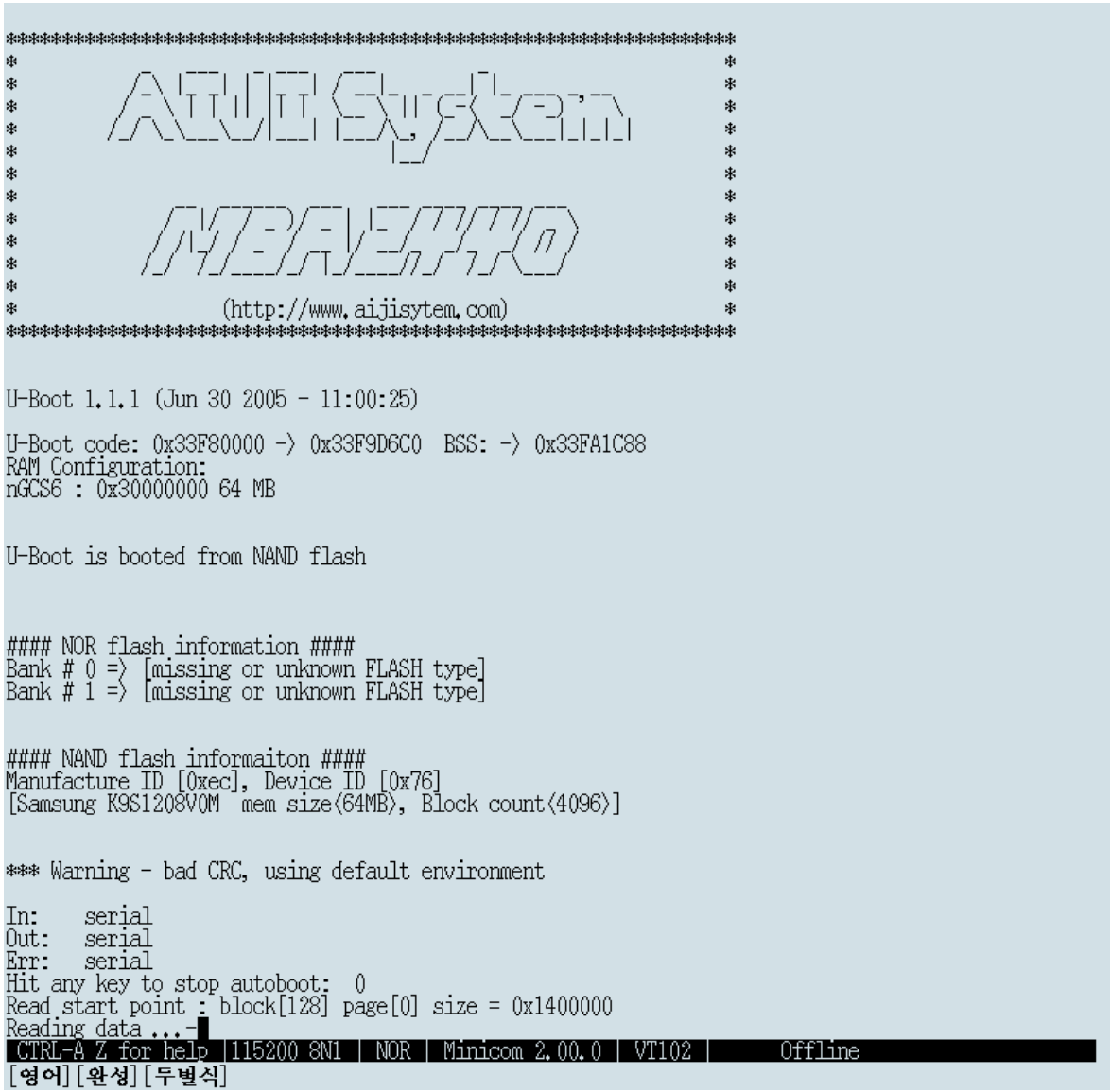
이 매크로가 일정 시간이 지난 후, 수행해야 할 명령어를 나타내는 매크로이다.

명령어는 ""안에 적어 넣으면 된다. 수행해야 할 명령이 여러 개일 경우에는 하나의 명령어가 끝나는 부분에 ;를 입력한다.

이 매크로가 주석처리 되면 자동 부팅 기능이 수행되지 않는다.

"nandr t 30200000 200000 1400000; go wince"

위 명령은 NAND flash에 있는 WinCE 이미지를 메모리로 복사한 후, WinCE를 부팅하라는 명령이다.



[그림 5.37]

[그림 5.37]는 전원이 켜진 후, 자동으로 WinCE를 부팅하는 모습이다.

5.3.3 U-boot 컴파일 하기

필요한 설정이 모두 끝나면 다음과 같은 명령을 차례로 수행한다.

물론, **이 명령어들은 mba2440-uboot1.1.1 디렉토리 상에서 수행되어야 한다.**

```
[root@localhost]#  pwd
/uboot/mba2440-uboot1.1.1
[root@localhost]#  make clobber
[root@localhost]#  make mba2440_config
[root@localhost]#  make all
```

make clobber

이 명령은 기존의 컴파일 되어 있는 모든 오브젝트 파일등을 제거해 준다.

make mba2440_config

이 명령은 앞서 설정한 것들을 적용시킨다.

make all

이 명령은 실제 컴파일을 수행한다.

위 세 명령을 다음과 같이 함으로써, 한 번에 수행 시킬 수 있다.

```
[root@localhost]#  make clobber;  make mba2440_config;  make all
```

정상적으로 컴파일 되었다면, 아래와 같이 **u-boot.bin** 파일이 생성됨을 확인할 수 있다.

```
[root@localhost]#  ls
CHANGELOG       System.map      drivers         lib_i386            mips_config.mk  u-boot
CHANGELOG.log   arm_config.mk   dtt             lib_m68k            mkconfig        u-boot.bin
COPYING         board           examples        lib_microblaze      net             u-boot.map
CREDITS         common          fs              lib_mips            nios_config.mk  u-boot.srec
MAINTAINERS     config.mk       i386_config.mk  lib_nios            post
MAKEALL         cpu             include         lib_ppc             ppc_config.mk
Makefile        disk            lib_arm         m68k_config.mk      rtc
README          doc             lib_generic     microblaze_config.mk  tools
```

u-boot.bin

이 파일은 컴파일 후, 생성되는 최종 바이너리 파일로서 이 파일을 부트롬의 0번지에 fusing하면 된다.

u-boot

이 파일은 elf 포맷의 u-boot 이미지 이다

u-boot.map

이 파일은 컴파일된 모든 함수들에 대한 메모리상의 위치 정보를 나타낸다.

6 Linux Kernel

임베디드 시스템을 위한 OS는 많이 존재하고 있다. MBA2440 보드는 크게 세 가지의 임베디드 시스템용 OS를 지원하고 있다.

1. 임베디드 리눅스
2. WinCE
3. iRTOS

이 장에서는 MBA2440 보드를 지원하는 임베디드 리눅스에 대하여 알아본다.

이 매뉴얼에서 다루고자 하는 리눅스의 커널 버전은 2.4.20 버전이며, 이 버전의 리눅스 소스를 기반으로 하여 MBA2440을 지원하도록 수정하였다.

이 장에서는 앞서 설명한 **3.3 임베디드 리눅스 개발 환경에서 설명한 내용대로 리눅스 PC의 개발 환경을 제대로 구축하였다는 가정하에서 설명**할 것이다.
이 장을 따라 하기에 앞서 반드시 3.3 임베디드 리눅스 개발 환경을 읽고 환경을 구축해야 한다.

먼저, MBA2440용 리눅스 소스를 컴파일 하는 방법에 대해서 알아보고, 다음으로는 리눅스를 여러 파일 시스템을 기반으로 하여 구동하는 방법을 알아본다.

6.1 Kernel compile

제공한 CD에 리눅스 커널 압축 소스가 있다. linux/kernel 디렉토리 안에 mba2440-linux2.4.20-v1.0.tar.bz2 이라는 이름의 압축 파일이 커널 소스이다.

먼저, **리눅스가 설치된 PC를 켜고 리눅스로 login할 때는 root 계정으로 login**하자. 이것은 리눅스 개발 과정에서 root 계정이 아니어서 생기는 문제들을 피하기 위한 것이다.

6.1.1 리눅스 커널의 압축 해제

커널 소스의 압축을 풀고 컴파일 하기 위해서 작업할 디렉토리를 하나 만들자.

/root/ 디렉토리 밑에 mba2440 이라는 디렉토리를 하나 만든다.

```
[root@localhost root]#  mkdir  /root/mba2440
```

다음으로는 제공한 CD를 리눅스 PC의 CD-ROM에 삽입하고 마운트하여 리눅스 커널 소스를 복사해 온다.

```
[root@localhost root]#  mount  /mnt/cdrom
[root@localhost root]#  cp  -a  /mnt/cdrom/linux/kernel/mba2440-linux2.4.20-v1.0.tar.bz2  ./mba2440/
[root@localhost root]#  cd  /root/mba2440
[root@localhost root]#  ls
mba2440-linux2.4.20-v1.0.tar.bz2
```

커널 소스의 압축을 해제한다.

```
[root@localhost root]#  tar  xjf  mba2440-linux2.4.20-v1.0.tar.bz2
[root@localhost root]#  ls
mba2440-linux2.4.20-v1.0   mba2440-linux2.4.20-v1.0.tar.bz2
```

압축 해제가 된 후에 mba2440-linux2.4.20-v1.0 이라는 디렉토리가 생겼음을 확인할 수 있다. 이 디렉토리로 이동하자.

```
[root@localhost root]#  cd  mba2440-linux2.4.20-v1.0
```

```
[root@localhost mba2440-linux2.4.20-v1.0]# ls
COPYING     Documentation  README          System.map  fs       ipc      mm       vmlinux
CREDITS     MAINTAINERS    REPORTING-BUGS  arch        include  kernel   net
ChangeLog   Makefile       Rules.make      drivers     init     lib      scripts
[root@localhost mba2440-linux2.4.20-v1.0]#
[영어][완성][두벌식]
```

[그림 6.1]

6.1.2 리눅스 커널의 설정

제공한 리눅스 소스는 이미 컴파일 되어 있는 소스 이기 때문에 설정을 다시 할 필요는 없으나 알아

둘 필요가 있다.

6.1.2.1 설정 파일 불러오기

각각의 하드웨어 플랫폼에 맞는 리눅스 설정에 대한 내용은 하나의 파일로 저장되어 있다. 이 파일들은 소스의 arch/arm/def-configs/ 디렉토리에 존재한다.

MBA2440을 위한 설정 파일도 이 디렉토리 안에 존재하며

- mba2440ramdisk : ramdisk를 root 파일 시스템으로 선택하기 위한 설정 파일
- mba2440cramfs : cramfs를 root 파일 시스템으로 선택하기 위한 설정 파일

두 개의 파일이 존재한다.

제공한 소스는 기본적으로 mba2440ramdisk 설정 파일의 내용대로 설정되어 있다.(64MB 메모리에 3.5"(240x320) TFT-LCD에서 동작하도록 설정)

자세한 사항은 6.2장 File System에서 다루기로 한다.

"make menuconfig" 명령을 통해 리눅스 커널에 대한 설정을 한다.

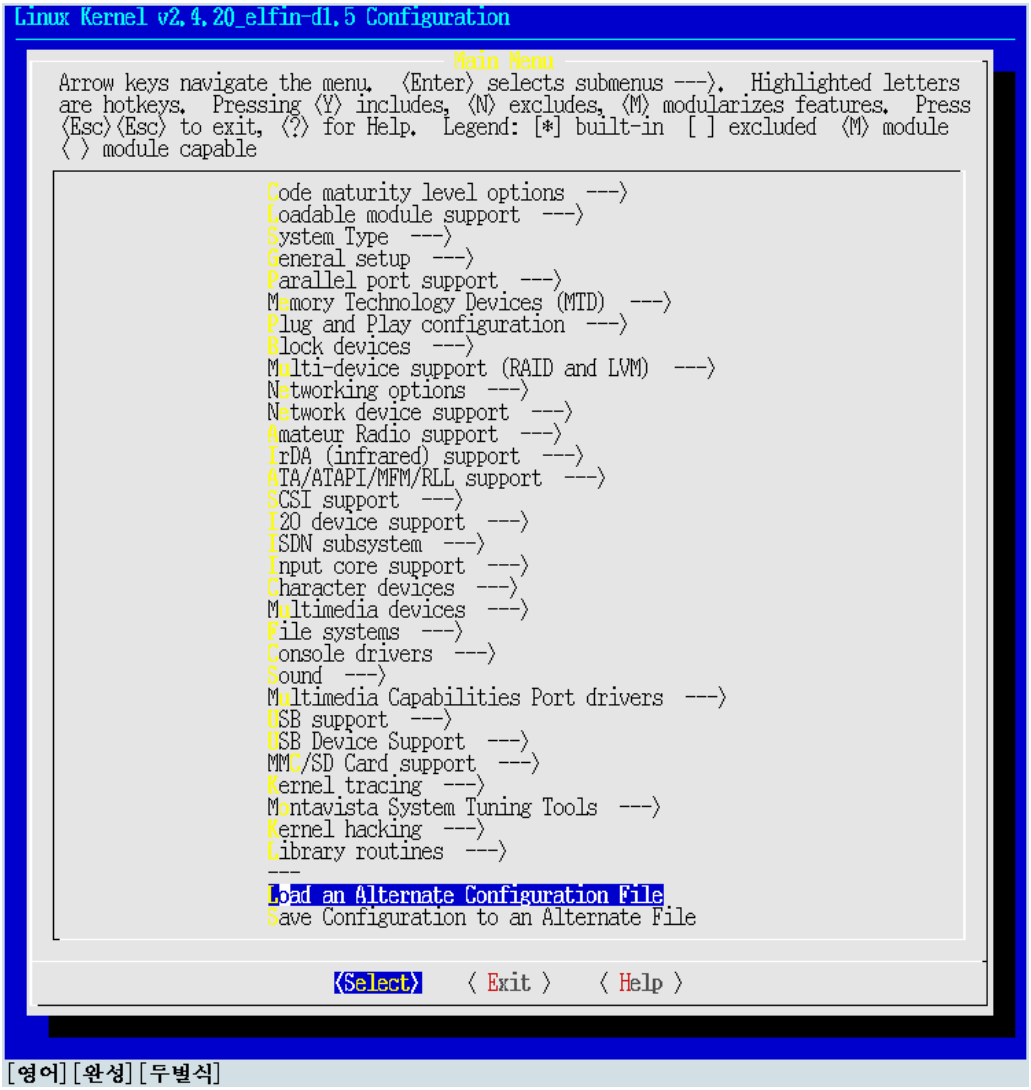
이 명령을 실행하면 텍스트 그래픽 모드의 main menu 설정 창이 생기며, 이 main menu설정 창에서 자신이 원하는 부분을 커널에 삽입하기도 하고 빼기도 한다.

이 main menu설정 창에서 자신이 원하는 메뉴를 화살표로 설정한 후, 엔터키나 스페이스 바를 누르면 선택한 메뉴의 서브 메뉴로 들어가게 된다.

메뉴를 빠져 나올 때에는 tab 키를 이용해 설정 창 제일 하단에 있는 <Exit>를 선택한 후, 마찬가지로 엔터키나 스페이스 바를 누르면 된다.

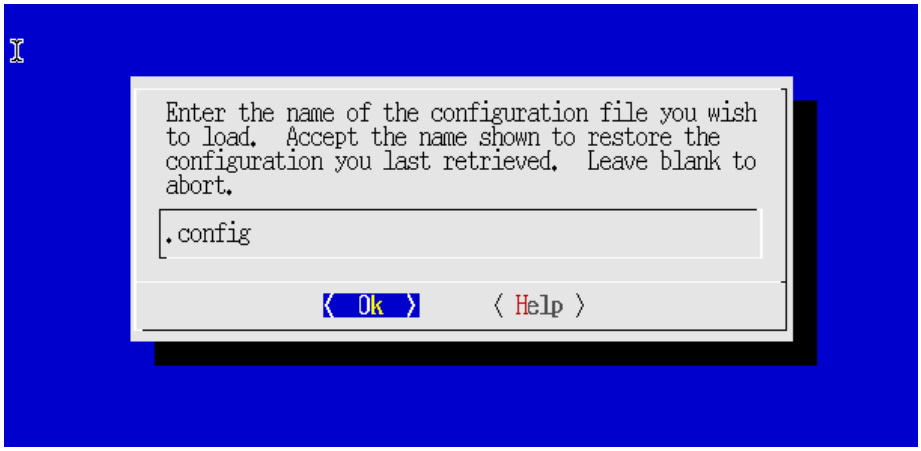
tab키로 <help>를 선택하면 방향키로 선택한 메뉴에 대한 설명을 볼 수 있다.

```
[root@localhost root]#  make  menuconfig
```



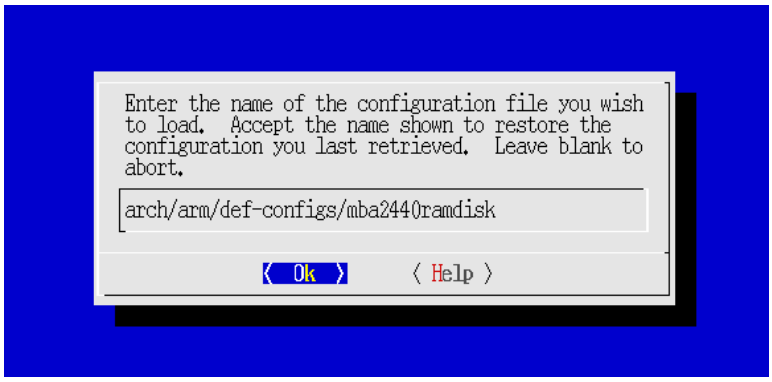
[그림 6.2] main menu 설정 창

[그림 6.2]에서 보는 바와 같이 Load an Alternate Configuration File을 선택하고, 엔터키나 스페이스 바를 누르면, [그림 6.3]와 같은 모습을 볼 수 있다.



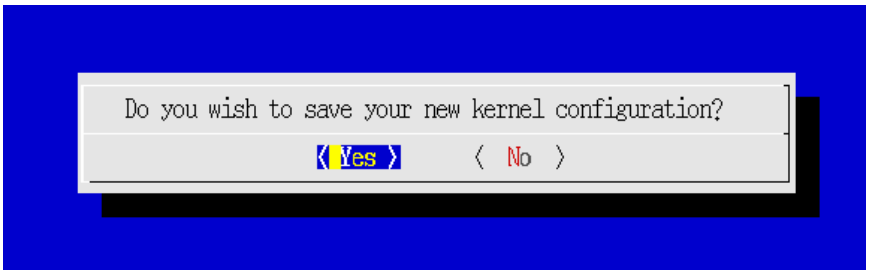
[그림 6.3]

".config"를 백스페이스 키로 지우고 "arch/arm/def-configs/mba2440ramdisk"를 입력한 후 엔터키를 누르면 mba2440ramdisk 의 설정 내용이 리눅스 커널에 적용된다.



[그림 6.4]

main menu 설정창에서 <Exit>를 설정하고 빠져 나오면 [그림6.5]처럼 설정 내용을 저장할지를 묻는데 이 때, <Yes>를 선택한다.



[그림 6.5]

저장하고 빠져 나온 후, 6.1.3 커널 컴파일 명령 수행을 참조하여 커널을 컴파일 한다.

6.1.2.2 메모리 크기 설정

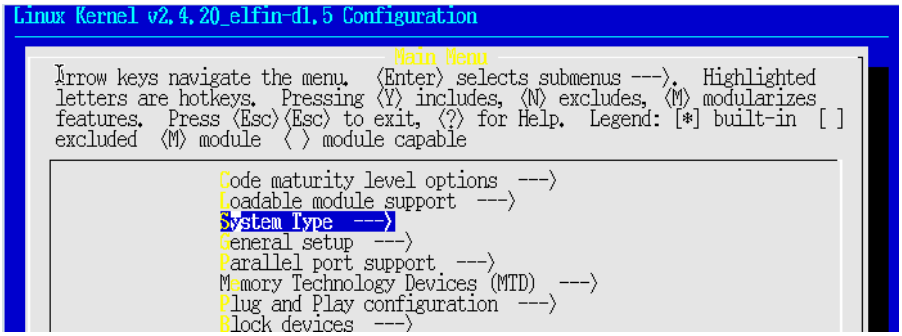
CD에서 제공하는 mba2440-linux2.4.20-v1.0.tar.bz2 리눅스 커널 소스는 기본적으로 64MB SDRAM을 장착한 MBA2440 보드에 맞춰져 있다.

128MB가 장착된 MBA2440 보드를 위해서는 컴파일 설정을 변경해 주어야 한다.

make menuconfig 명령을 수행한다.

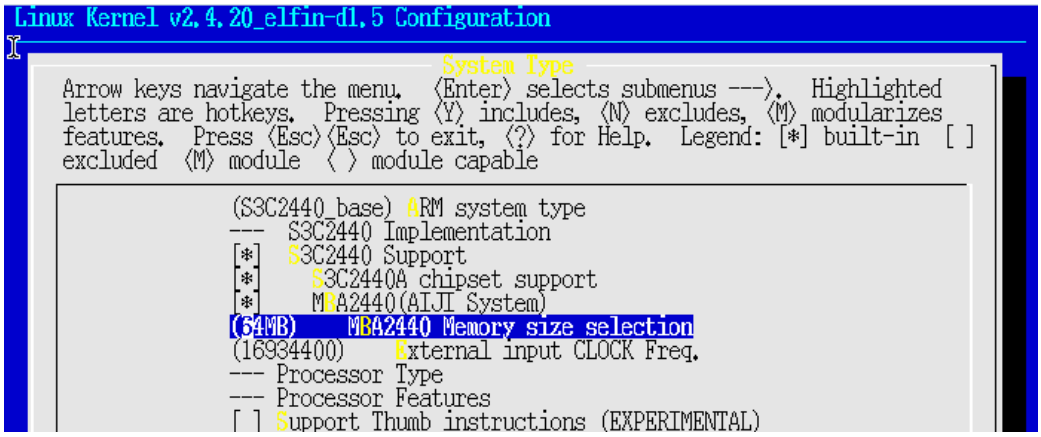
```
[root@localhost root]#  make  menuconfig
```

System Type을 선택하고 엔터키를 누른다.



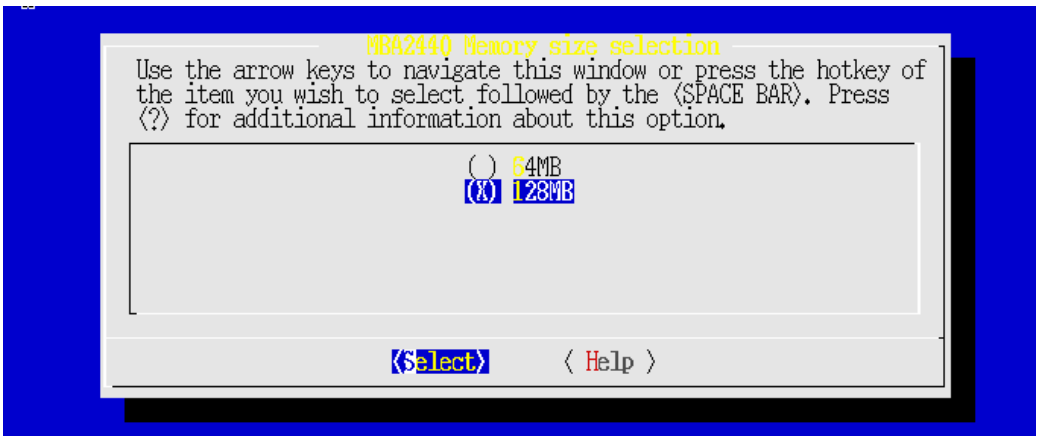
[그림 6.6]

[그림 6.7]과 같이 MBA2440 Memory size selection을 선택하고 엔터키를 누른다.



[그림 6.7]

[그림 6.8]과 같이 128MB를 선택하고 스페이스 바를 누른다.



[그림 6.8]

[그림 6.9]같이 저장할지를 묻는 메뉴가 나올 때 까지 Exit로 빠져 나온다.

[그림 6.9]

저장하고 빠져 나온 후, 6.1.3 커널 컴파일 명령 수행을 참조하여 커널을 컴파일 한다.

6.1.2.3 TFT-LCD 설정

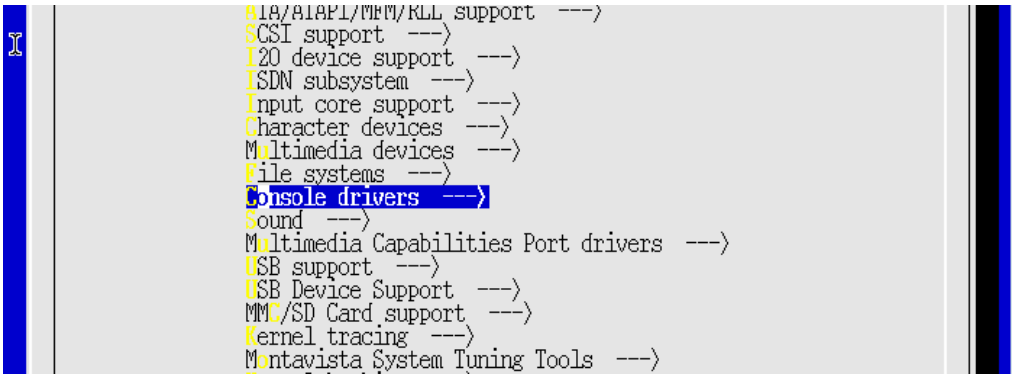
CD에서 제공하는 mba2440-linux2.4.20-v1.0.tar.bz2 리눅스 커널 소스는 기본적으로 240x320 3.5" TFT-LCD에 맞게 동작하도록 되어 있다.

640x480 6.4" TFT-LCD에 맞게 동작하도록 하기 위해서는 커널 설정을 변경해 주어야 한다.

make menuconfig 명령을 수행한다.

```
[root@localhost root]#  make  menuconfig
```

Console drivers를 선택한다.



[그림 6.10]

Frame-buffer support를 선택한다.

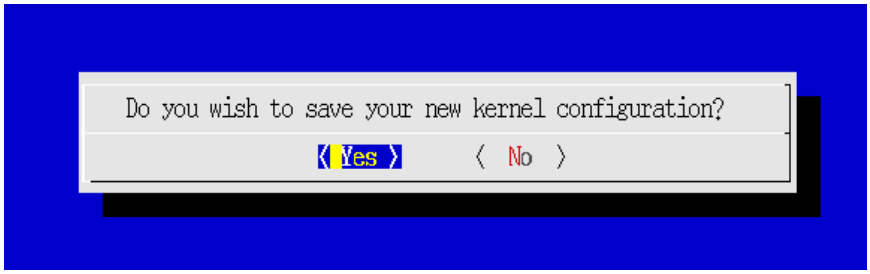
[그림 6.11]

[그림 6.12] 240x320 TFT-LCD support에 * 표가 되어 있을 것이다. 240x320 TFT-LCD support를 선택하고 스페이스 바를 눌러 *를 없앤다.

640x480 TFT-LCD support를 선택하고 스페이스 바를 두번 눌러 * 표를 체크한다.

[그림 6.12]

[그림 6.13]같이 저장할지를 묻는 메뉴가 나올 때 까지 Exit로 빠져 나온다.



[그림 6.13]

저장하고 빠져 나온 후, 6.1.3 커널 컴파일 명령 수행을 참조하여 커널을 컴파일 한다.

6.1.3 커널 컴파일 명령 수행

커널 환경 설정이 모두 끝났다면, 컴파일 하기 전에 이전에 이미 생성되어 있는 오브젝트 파일이나, 이미지들을 지우는 청소 과정을 거쳐야 한다.

```
[root@localhost root]#  make  clean
```

“make clean” 명령은 이전에 생성되었던 오브젝트 파일, 커널 이미지, 임시파일 등을 삭제한다.

다음으로는, 컴파일 하기 전에 각 소스들간의 의존성 관계를 체크하는 명령을 수행한다.

```
[root@localhost root]#  make  dep
```

마지막으로 커널을 컴파일 하여 이미지를 생성하는 명령을 수행한다.

```
[root@localhost root]#  make  zImage
```

커널 컴파일이 완료되었다면 arch/arm/boot/ 디렉토리에 있는 파일들을 열람해 보자. 이 디렉토리 안에 있는 zImage 라는 파일이 바로 커널 이미지이다.

이 이미지를 MBA2440의 메모리에 적재하면 리눅스로 부팅할 수 있다.

```
[root@localhost root]#  ls  arch/arm/boot/
Makefile  bootp  compressed  endianswap.c  install.sh  zImage
[root@localhost root]#
```

리눅스 커널을 MBA2440에 다운로드하여 부팅 시키는 방법에 대해서는 2.4 리눅스 실행하기를 참조하기 바란다.

6.2 File System

리눅스는 부팅할 때, 어떤 파일 시스템을 root 파일 시스템으로 선택하여 부팅할 지를 결정할 수 있다.

이 장에서는 root 파일 시스템으로 사용하기 위한 파일 시스템인

- **RAMDISK 파일 시스템**
- **CRAMFS 파일 시스템**

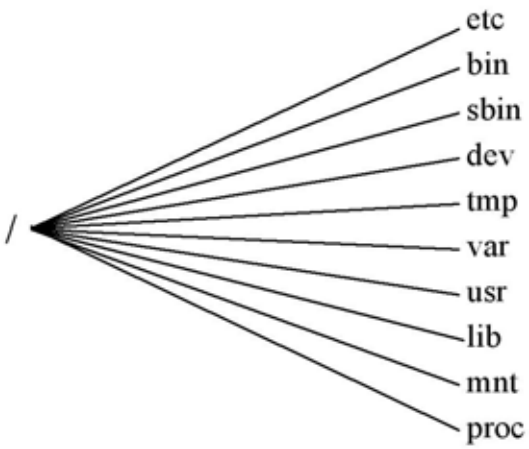
에 대해 알아본다.

또 한, **JFFS2 파일 시스템**을 이용하여 flash 메모리를 데이터 저장용 비휘발성 저장 장치로 마운트하여 사용하는 방법에 대해서 알아본다.

6.2.1 Root File System

Root 파일 시스템(리눅스에서 최상위 디렉토리 "/")은 커널이 동작하기 위한 시스템 메모리 공간과 library, utility등이 존재하는 공간으로서 리눅스에서 반드시 필요한 부분이다.

Root 파일 시스템은 [그림 6.14]과 같이 구성되어 있으며, 데스크 탑 PC에서 동작하는 root 파일 시스템과 구조상 별 차이가 없다.



[그림 6.14]

다음 표는 루트 파일 시스템에 대한 간단한 설명이다.

[표 6.1] 루트 파일 시스템에 대한 설명

etc	각 machine에 따른 설정 파일들을 포함하고 있다. /etc/hosts /etc/passwd 등이 시스템 관리에 필요한 파일들이 존재한다.
bin	사용자 명령어들이 존재한다.

sbin	각 machine들이 부팅할 때 필요한 파일들이 존재한다. Boot 이미지가 부팅할 때 Ramdisk file system에서 읽어 들여 수행하게 된다.
dev	Tape 드라이브, 프린터, 디스크, 디스크 파티션, 램 디스크, 터미널과 같은 디바이스에 대한 노드들이 존재한다.
tmp	시스템에서 사용하는 임시파일이 존재한다.
var	Log, lock, spool, mail 등 애플리케이션에 대한 변수 등이 쓰인다.
usr	실행 가능한 파일이나 공유 가능한 ASCII 문서를 포함하고 있으면 바뀌지 않는 파일들이 존재하다. 또한 다른 원격 서버에 의해서도 공유되어 쓸 수 있다.
lib	해당 시스템에 필요한 라이브러리 파일들이 존재한다.
mnt	외장 디바이스의 마운트를 위한 공간
proc	가상 파일시스템으로 일시적을 시스템의 동적인 정보를 제공

6.2.2 RAMDISK File System

임베디드 시스템의 경우 하드 디스크를 갖지 않는 것이 보통이고, 플래시 메모리도 읽기는 빠르지만 쓰기가 느린 저장 장치이므로 일반적으로 시스템의 이미지를 압축 저장하는 용도로 사용된다.

커널이 부팅되기 시작하면 root 파일 시스템이 필요하다.

이 장에서는 root 파일 시스템으로 RAMDISK 파일 시스템에 대해 알아본다.

MBA2440 보드에 적재될 이미지 중 커널을 제외한 루트 파일 시스템, 사용자 응용 프로그램 등은 램 디스크 이미지로 구성되어 리눅스로 부팅하기 전에 메모리에 적재되어 있다가 리눅스가 부팅한 후, root파일 시스템의 역할을 하게 된다.

램 디스크 이미지는 일반적으로 임베디드 리눅스 공급자에 의해 구성된 것이 제공되고 사용자는 일반적으로 개발한 응용 프로그램을 램 디스크 이미지에 추가하는 방식을 사용한다.

RAMDISK를 root 파일 시스템으로 하여 리눅스를 부팅하는 방법은 2.4 리눅스 실행하기를 참조하라.

6.2.2.1 램디스크 파일 시스템

다음으로는, 램디스크에 대해서 알아보자.

램디스크(ramdisk) 파일 시스템은 RAM의 일부를 디스크 파일 시스템처럼 사용하게 하여 커널에게 읽고 쓸 수 있는 빠른 파일 시스템을 제공한다.

제공한 CD에 linux/filesystem/ramdisk 디렉토리에서 **mba2440-ram.gz** 램디스크 이미지를 얻을 수 있다.

이 램디스크를 root 파일 시스템으로 사용하기 위해서는 앞서 설명한 것처럼 리눅스 커널 컴파일 설정에서 램디스크를 root 파일 시스템으로 사용하도록 설정해 주어야 한다.

이를 위한 설정 파일이 arhc/arm/def-configs/ 디렉토리에 있는 **mba2440ramdisk** 파일이다. 6.1.2.1

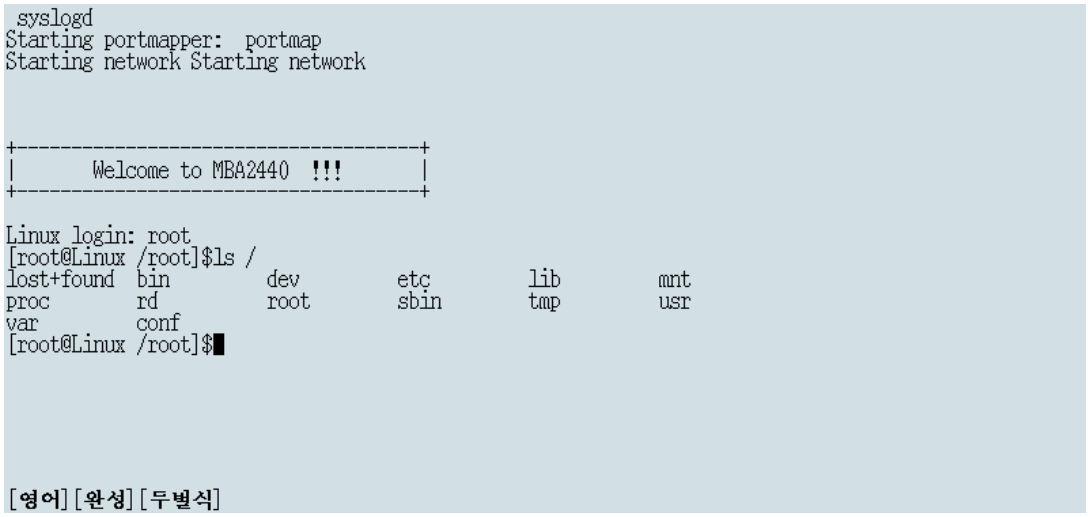
설정 파일 불러오기에서 설정 파일 선택시 이 파일을 선택하여 설정하면 된다.

컴파일 과정은 6.1 커널 컴파일 명령 수행을 참조하라.

램디스크 이미지는 gzip format으로 압축된 파일이며, 이 이미지는 메모리의 0x30800000 번지에 위치해 있어야 한다. 이 램디스크는 16MB의 크기로 마운트되며 MBA2440을 리눅스로 부팅한 후 console 창에서 df 명령을 통해서 이를 확인 할 수 있다.

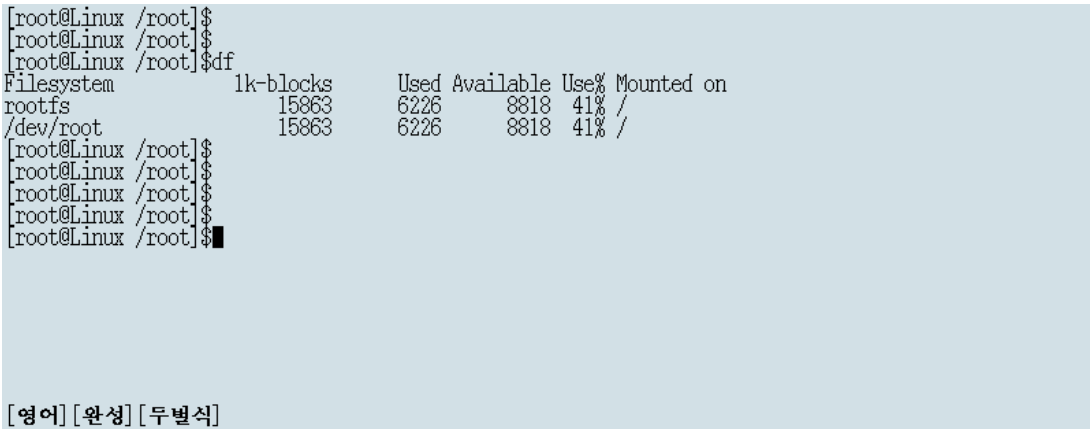
리눅스 커널은 부팅과정에서 0x30800000 번지에 있는 압축된 램디스크 이미지를 찾아내어 압축을 푼 후, 이를 root 파일 시스템으로 마운트하게 된다.

[그림 6.15]에서 MBA2440상에서 리눅스가 램디스크를 root 파일 시스템으로 마운트하여 부팅한 후, serial console(minicom이나 DNW등...)에서 마운트된 root 파일 시스템의 모든 디렉토리를 확인할 수 있다.



[그림 6.15]

[그림 6.16]은 mount된 램디스크를 “df” 명령을 통해 확인한 모습이다.



[그림 6.16]

rootfs는 root 파일 시스템을 나타내는 것이다.

dev/root는 실제로 램디스크를 block device driver로 인식하여 mount 하였음을 나타낸다.

6.2.2.2 램디스크의 수정

개발자의 편의에 따라 ramdisk 내부에 파일을 추가 하거나 삭제할 수 있다.
이 방법을 통해 사용자가 개발한 어플리케이션 실행 파일을 ramdisk 이미지에 넣은 후, 실행 시킬 수 있다.

램디스크를 수정해 보자.
이 장에서는 다음과 같은 순서를 따라 해 보면서 램디스크 수정 방법에 대해 배울 것이다.

1. **램디스크 파일 시스템의 /root 디렉토리에 view.txt 라는 파일을 생성한다.**
2. **생성한 view.txt 파일에 "Hello, MBA2440"이라는 텍스트를 저장한다.**
3. **램디스크를 압축한 후, u-boot의 tftp 명령을 이용해 메모리에 적재한다.**
4. **리눅스 커널 이미지를 MBA2440에 tftp 명령을 통해 적재한다.**
5. **리눅스로 부팅한 후 /root 디렉토리 밑에 view.txt 파일이 있음을 확인한다.**

램디스크를 수정하는 것이므로 기존의 램디스크 이미지(mba2440-ram.gz)가 있어야 한다.
/root/ramdisk 라는 디렉토리를 만들고 CD에서 제공한 램디스크 이미지(mba2440-ram.gz)를 복사한다.
디렉토리 생성

```
[root@localhost root]#  mkdir  /root/ramdisk
```

제공한 CD를 리눅스 PC의 CD-ROM에 넣은 후, 램디스크 이미지를 복사한다

```
[root@localhost root]#  mount  /mnt/cdrom
[root@localhost root]#  cp  /mnt/cdrom/filesystem/ramdisk/mba2440-ram.gz  /root/ramdisk/
[root@localhost root]#  cd  /root/ramdisk
[root@localhost root]#  ls
mba2440-ram.gz
[root@localhost root]#
```

압축된 램디스크 이미지를 마운트하여 파일 시스템으로 인식하기 위해 임의의 디렉토리를 생성한다
디렉토리명은 ram_tmp 이다.

```
[root@localhost root]#  mkdir  ram_tmp
[root@localhost root]#  ls -al
합계 1852
drwxr-xr-x    3 root     root         4096  6월  27 20:39 .
drwxr-x---   31 root     root         4096  6월  27 20:39 ..
-rw-r--r--    1 root     root      1878054  6월  27 20:39 mba2440-ram.gz
drwxr-xr-x    2 root     root         4096  6월  27 20:39 ram_tmp
```

램디스크 이미지의 압축을 푼다.

```
[root@localhost root]#  gzip  -d  mba2440-ram.gz
```

```
[root@localhost root]#  ls -al
합계 16416
drwxr-xr-x    3 root     root         4096  6월 27 20:44 .
drwxr-x---   31 root     root         4096  6월 27 20:44 ..
-rw-r--r--    1 root     root     16777216  6월 27 20:39 mba2440-ram
drwxr-xr-x    2 root     root         4096  6월 27 20:39  ram_tmp
```

압축이 풀린 이미지 mba2440-ram을 보면 사이즈가 16M로 늘어난 것을 볼 수 있다.

압축이 풀린 램디스크 이미지를 리눅스의 루프 디바이스와 연결 시킨다

```
[root@localhost root]#  losetup  /dev/loop0  mba2440-ram
```

연결된 루프 디바이스를 ram_tmp 디렉토리에 마운트한다.

```
[root@localhost root]#  mount  /dev/loop0  ram_tmp
[root@localhost root]#  cd  ram_tmp
[root@localhost root]#  ls
bin  conf  dev  etc  lib  lost+found  mnt  proc  rd  root  sbin  tmp  usr  var
[root@localhost root]#
```

이제 root 디렉토리 내부에 view.txt 파일을 생성한 후, “Hello, MBA2440”을 쓴 후, 저장하자

```
[root@localhost root]#  cd  root
[root@localhost root]#  vi  view.txt
```

vi 에디터로 파일에 Hello, MBA2440을 쓰고 저장한 후, 빠져 나온다.

ram_tmp 디렉토리를 언마운트한다.

```
[root@localhost root]#  cd  ../../
[root@localhost root]#  ls
mba2440-ram  ram_tmp
[root@localhost root]#  umount  ram_tmp
```

루프 디바이스의 연결을 끊는다

```
[root@localhost root]#  losetup  -d  /dev/loop0
```

수정된 램디스크 이미지를 gzip으로 압축한다.

```
[root@localhost root]#  gzip  mba2440-ram
[root@localhost root]#  ls -al
합계 1852
drwxr-xr-x    3 root     root         4096  6월 27 20:53 .
drwxr-x---   31 root     root         4096  6월 27 20:53 ..
```

```
-rw-r--r--    1 root     root       1878985  6월 27 20:39 mba2440-ram.gz
drwxr-xr-x    2 root     root          4096  6월 27 20:39 ram_tmp
```

이제 압축된 램디스크 이미지와 CD에서 제공한 커널 이미지를 u-boot의 tftp 명령을 이용하여 다운로드한 후 리눅스를 부팅해 보자.(2.4.1 TFTP로 리눅스 실행하기 참조)

부팅이 완료된 후, root 계정으로 로그인한다.

“ls” 명령을 통해 view.txt 파일이 있음을 볼 수 있으며, “cat” 명령을 통해 view.txt 파일의 내용을 확인할 수 있다. [그림 6.17]

```
eth0: using half-duplex 10Base-T (RJ-45)
IP-Config: Incomplete network configuration information.
NET4: Unix domain sockets 1.0/SMP for Linux NET4.0.
NetWinder Floating Point Emulator V0.95 (c) 1998-1999 Rebel.com
RAMDISK: Compressed image found at block 0
Freeing initrd memory: 4096K
EXT2-fs warning: checktime reached, running e2fsck is recommended
VFS: Mounted root (ext2 filesystem).
Freeing init memory: 76K
INIT: versio~ 2.74 booting
eth0: using half-duplex 10Base-T (RJ-45)
INIT: Enteriog runlevel: 3
Starting system logger:  syslogd
Starting portmapper:  portmap
Starting network Starting network



+--------------------------------------+
|       Welcome to MBA2440  !!!        |
+--------------------------------------+

Linux login: root
[root@Linux /root]$
[root@Linux /root]$
[root@Linux /root]$ls -al
drwxr-x---    2 root     root        1024 Jun 27  2005 .
drwxr-xr-x   16 root     root        1024 Jun 27  2005 ..
-rw-r-----    1 root     root         208 Mar 15  2002 .profile
-rw-r--r--    1 root     root          15 Jun 27  2005 view.txt
[root@Linux /root]$cat view.txt
Hello, MBA2440
[root@Linux /root]$
[root@Linux /root]$
[root@Linux /root]$
[root@Linux /root]$
[root@Linux /root]$
[root@Linux /root]$
[root@Linux /root]$
 CTRL-A Z for help |115200 8N1 | NOR | Minicom 2.00.0 | VT102 |      Offline
[영어][완성][두벌식]
```

[그림 6.17]

6.2.3 CRAMFS

이 장에서는 CRAMFS의 기본 개념을 알아보고, CRAMFS를 root 파일 시스템으로 사용하여 리눅스를 부팅하는 방법에 대해 알아본다.

6.2.3.1 CRAMFS

CRAMFS는 리눅스 2.4 버전에서 새롭게 선보인 파일 시스템이다. 이 파일시스템은 작은 크기의 읽

기 전용 파일 시스템으로 설계되었다.

CRAMFS의 최대 장점은 모든 파일들이 압축된 형태로 저장되며 실행 중에 그 파일들이 압축 해제된다는 점이다. 이처럼 작은 메모리 공간을 사용하기 때문에 임베디드 시스템의 파일 시스템으로서의 사용이 용이하다.

RAMDISK 파일 시스템의 경우에는 일정 크기의 SDRAM을 파일 시스템으로 사용하기 때문에 시스템 메모리로 사용될 공간이 그만큼 줄어든다. CRAMFS를 사용하게 되면 RAMDISK에서 사용하게 되는 메모리 공간을 사용하지 않게 되므로 그 만큼의 공간을 시스템 메모리 공간으로 사용할 수 있게 된다.

반면에, 파일들이 압축되어 있기 때문에 응용 프로그램이 동작하기 위해선 RAM에 응용 프로그램을 복사해야 하는 번거로움이 있고 이로 인해 더 많은 용량의 RAM을 필요로 할 수 있다.

Read Only 파일 시스템이기 때문에 쓰기는 전혀 불가능하다.

큰 용량의 파일 시스템(256MB이상)은 구현할 수 없다.

CRAMFS를 사용하기 위해서는 MBA2440용 리눅스 소스를 CRAMFS를 사용할 수 있도록 컴파일 설정을 한 후, 컴파일 하여 생성된 커널 이미지를 사용해야 한다.

이를 위한 커널 설정 파일은 MBA2440용 리눅스 커널 소스의 arch/arm/def-configs/ 디렉토리 안에 있는 **mba2440cramfs** 파일이다. 6.1.2.1 설정 파일 불러오기에서 설정 파일 선택시 **mba2440cramfs** 파일을 선택하여 설정하면 된다.

컴파일 과정은 6.1.3 커널 컴파일 명령 수행 을 참조하라.

제공한 CD에 linux/image/3.5 디렉토리에서 240x320 3.5” TFT-LCD가 장착된 MBA2440보드의 CRAMFS용 커널 이미지를 얻을 수 있다. **zImage-cramfs3.5.bin**

Linux/image/filesystem/cramfs 디렉토리에서 CRAMFS 파일 시스템 이미지도 얻을 수 있다.

root-mba2440.cramfs

6.2.3.2 CRAMFS로 리눅스 부팅하기

CD에 있는 CRAMFS용 리눅스 커널 이미지와 CRAMFS 이미지를 이용하여 리눅스를 부팅시켜 보자.

이 작업은 MBA2440 보드의 콘솔 상에서 u-boot를 이용하는 작업이다.

CRAMFS 이미지는 NAND flash 메모리상에 적재되어 있어야 한다.

먼저, CRAMFS 이미지를 TFTP로 다운로드 받은 후, NAND flash에 적재한다.

CRAMFS용 리눅스 커널 이미지와 CRAMFS 이미지를 TFTP 서버의 root 디렉토리에 복사해야 함을 잊지 말자.

```
MBA2440 #  tftp  30000000  root-mba2440.cramfs
TFTP from server 192.168.100.2; our IP address is 192.168.100.3
Filename 'root-mba2440.cramfs'.
Load address: 0x30000000
Loading:-
```

```
done
Bytes transferred = 1269760 (136000 hex)
```

NAND flash의 file system영역을 지운 후, 그 자리에 write한다. File system 영역에 대해서는 6.2.4 JFFS2 File System에서 다룬다.

```
MBA2440 #  nande  128  383
erase block [128 ~ 383]
Erasing block ../
Complete erasing block



MBA2440 #  nandw  t  30000000  200000  140000
Write start point : block[128] page[0] size = 0x140000
Writing data .../
Write compelete(0x140000): from 0x30000000 SDRAM to 0x200000(128) NAND flash
```

CRAMFS용 리눅스 커널 이미지를 다운로드한 후, 부팅한다.

```
MBA2440 #  tftp  30c00000  zImage-cramfs3.5.bin
TFTP from server 192.168.100.2; our IP address is 192.168.100.3
Filename 'mba2440/zImage-cramfs.bin'.
Load address: 0x30c00000
Loading:/
done
Bytes transferred = 746012 (b621c hex)
MBA2440 #  go  30c00000
Uncompressing Linux..... done, booting the kern.
Linux version 2.4.20_elfin-d1.5 (root@localhost.localdomain) (gcc version 2.95.3 20010315 (releaT
CPU: ARM/CIRRUS Arm920Tid(wb) revision 0
.....
.....(생략)
.....
+--------------------------------------+
|         Welcome to MBA2440   !!!                |
+--------------------------------------+

MBA2440(S3C2440) Linux (experimental)
Kernel 2.4.20_elfin-d1.5 on   armv4l
```

```
console=/dev/console
init started:  BusyBox v0.60.3 (2002.05.13-08:36+0000) multi-call binary
Starting pid 30, console /dev/console: '/etc/init.d/rcS'
eth0: using half-duplex 10Base-T (RJ-45)
Waiting for enter to start '/bin/sh' (pid 38, terminal /dev/console)


Please press Enter to activate this console.
```

엔터키를 치면 프롬프트가 뜨는 것을 확인할 수 있다.

[그림 6.18]

6.2.3.3 CRAMFS 수정하기

CRAMFS 파일 시스템 이미지는 RAMDISK 파일 시스템과는 달리 이미지 자체가 Read Only 이미지이기 때문에 임의의 디렉토리에 mount 하더라도 읽기만 가능할 뿐 쓰기는 불가능하다.

그렇다면, 과연 어떻게 CRAMFS를 수정할 것인가…??

대답은 간단하다. CRAMFS 이미지를 압축해제하는 utilily(cramfsck)를 사용하여 임의의 디렉토리에 풀어 놓은 다음, 수정을 가하고 그 임의의 디렉토리를 CRAMFS 이미지로 만들어 주는 utility(mkcramfs)를 사용하면 된다.

이 CRAMFS용 utility들은 제공한 CD에 linux/filesystem/cramfs 디렉토리에 있다.

이 작업은 리눅스 데스크 탑 PC상에서 이루어 지며, 작업 디렉토리는 /root/cramfs 라는 디렉토리를 하나 만들고 이 디렉토리 안에서 작업할 것이다.

먼저, CRAMFS 이미지를 임의의 디렉토리를 만들고 이 안에 utiliy, CRAMFS 이미지등 모든 것을 복사하자.

CD를 리눅스 PC의 CD-ROM에 삽입한다.

```
[root@localhost root]#  mkdir  /root/cramfs
[root@localhost root]#  cd  cramfs
[root@localhost root]#  mount  /mnt/cdrom
[root@localhost root]#  cp  /mnt/cdrom/filesystem/cramfs/*  ./
[root@localhost root]#  ls
cramfsck  mkcramfs  root-mba2440.cramfs
```

CRAMFS를 마운트 하기 위한 임의의 디렉토리를 만든다. 디렉토리명은 tmp로 하자.

```
[root@localhost root]#  mkdir  tmp
[root@localhost root]#  ls
cramfsck  mkcramfs  root-mba2440.cramfs  tmp
[root@localhost root]#  mount  -t  cramfs  -o  loop  root-mba2440.cramfs  tmp
[root@localhost root]#  ls  tmp
bin  dev  etc  lib  linuxrc  mnt  proc  sbin  tmp  usr  var
```

임의의 파일을 하나 만들어 보자

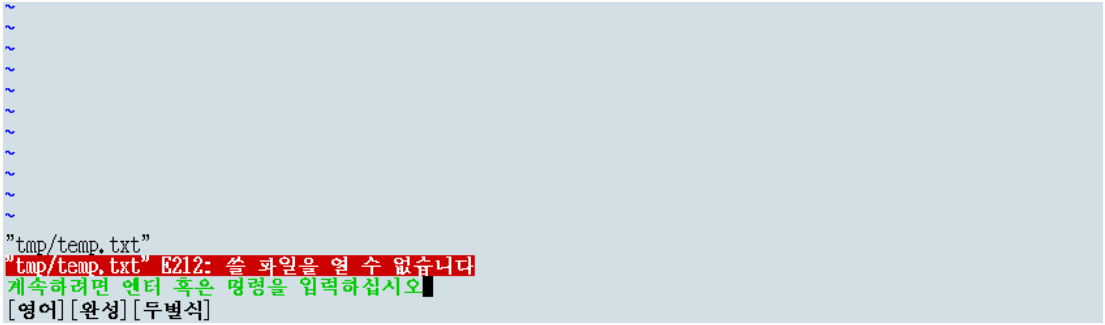
```
[root@localhost root]#  vi  tmp/temp.txt
```

파일 내용 안에 아무 글이나 적고 wq 명령으로 저장하고 빠져 나와보자…

제대로 되는가???

[그림 6.19]과 같은 모습을 볼 수 있을 것이다. 물론, 제대로 변경되지 않는다. 오히려, 이미지가 손상되어 버린다.

이유는, 앞서 설명한 대로 CRAMFS 파일 시스템 이미지는 read-only 이기 때문이다.



[그림 6.19]

이제부터는, CRAMFS 파일 시스템 이미지를 수정해 보자.

CRAMFS 파일 시스템 이미지가 손상되었을 것이므로, 다시 CD에서 제대로 된 CRAMFS 파일 시스템 이미지를 복사해 온다.

물론, 이전에 마운트 되어 있는 tmp 디렉토리를 unmount 시킨다.

```
[root@localhost root]#  umount  tmp
[root@localhost root]#  rm  -f  root-mba2440.cramfs
[root@localhost root]#  cp  /mnt/cdrom/filesystem/cramfs/root-mba2440.cramfs  ./
```

CRAMFS 파일 시스템 이미지를 tmp 디렉토리에 extract한다. 이 때, 사용되는 utility가 **cramfsck** 이다. 이 utility은 인터넷 상에서 얻을 수 있다.

```
[root@localhost root]#  ./cramfsck  -h
usage: ./cramfsck [-hv] [-x dir] file
 -h          print this help
 -x dir      extract into dir
 -v          be more verbose
 file        file to test
[root@localhost root]#  ./cramfsck  -x  tmp  root-mba2440.cramfs
[root@localhost root]#  ls  tmp
bin  dev  etc  lib  linuxrc  mnt  proc  sbin  tmp  usr  var
```

linuxrc 파일을 열고 다음과 같이 수정해 보자

```
[root@localhost root]#  vi  tmp/temp.txt
```

temp.txt 파일

```
temp data ..
```

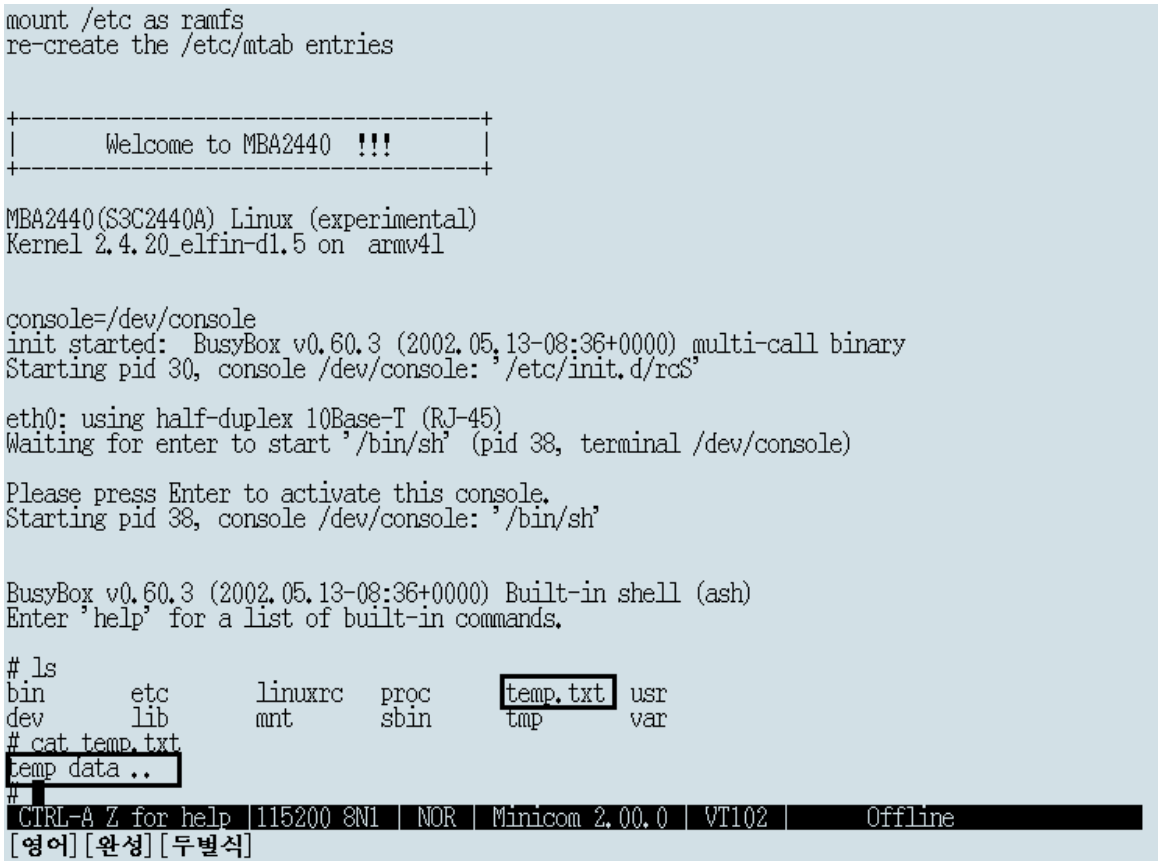
문장을 쓴 후에 저장하고 빠져 나온다.

이제 CRAMFS 파일 시스템 이미지를 만들어 보자. 이 때, 사용되는 utility가 **mkcramfs** 이다. 이 utility은 인터넷 상에서 얻을 수 있다.

```
[root@localhost root]#  ./mkcramfs  tmp  modified.cramfs
.....
-48.18% (-2168 bytes)    lnx_init
-53.81% (-31596 bytes)   pump
 38.46% (+10 bytes)      pump.sh
150.00% (+12 bytes)      var
Everything: 1240 kilobytes
Super block: 76 bytes
CRC: 112c8b5a
```

```
[root@localhost cramfs]#  ls
cramfsck   mkcramfs   modified.cramfs   root-mba2440.cramfs   tmp
```

새로 만들어진 modified.cramfs를 TFTP로 다운로드 받은 후 실행 시켜보자. 바로 앞 장 6.2.3.2 CRAMFS로 리눅스 부팅하기를 참조하라.



[그림 6.20]

[그림 6.20] 에서 보는 바와 같이 수정했던 내용이 반영되어 출력되는 것을 확인할 수 있다.

6.2.4 JFFS2 File System

MBA2440용 임베디드 리눅스 커널은 MTD 드라이버를 사용한다. 이 드라이버를 이용하여 NAND flash를 저장 장치로 사용할 수 있다. 이 때, 이 NAND flash에 적용하는 파일 시스템이 JFFS2 이다.

JFFS2(Journaling Flash File System 2)은 갑작스럽게 전원이 꺼질 경우에도 안전하며, 다음에 다시 부팅할 때, 파일 시스템을 체크하지 않아도 되는 장점이 있다.

Flash memory는 수만번 정도의 쓰기 동작을 견딜 수 있을 정도의 수명을 가지고 있다. JFFS2는 특별히 flash memory를 사용하기 위해 설계되었다. Flash memory의 수명 단축을 막기 위해 wear leveling이라는 기능을 제공하는데, 이 기능은 데이터를 저장할 때, 플래시의 모든 영역이 골고루 사용되도록 함으로써 flash 메모리의 수명을 획기적으로 향상 시킨다.

JFFS2는 이전 버전인 JFFS와는 달리 파일을 압축시켜 저장하기 때문에 효율적이다.

JFFS2는 journaling과 garbage collection system에 의해 약간의 영역이 소비된다.

MBA2440용 리눅스 커널은 커널 부팅시 NAND flash가 연결되어 있는 상태라면 자동으로 NAND flash(SOP, SMC 모두를 인식한다)를 JFFS2 파일 시스템으로 partitioning하여 인식할 수 있다.

다음의 두 그림은 커널 부팅시 NAND flash를 인식하여 partitioning한 내용을 보여준다.

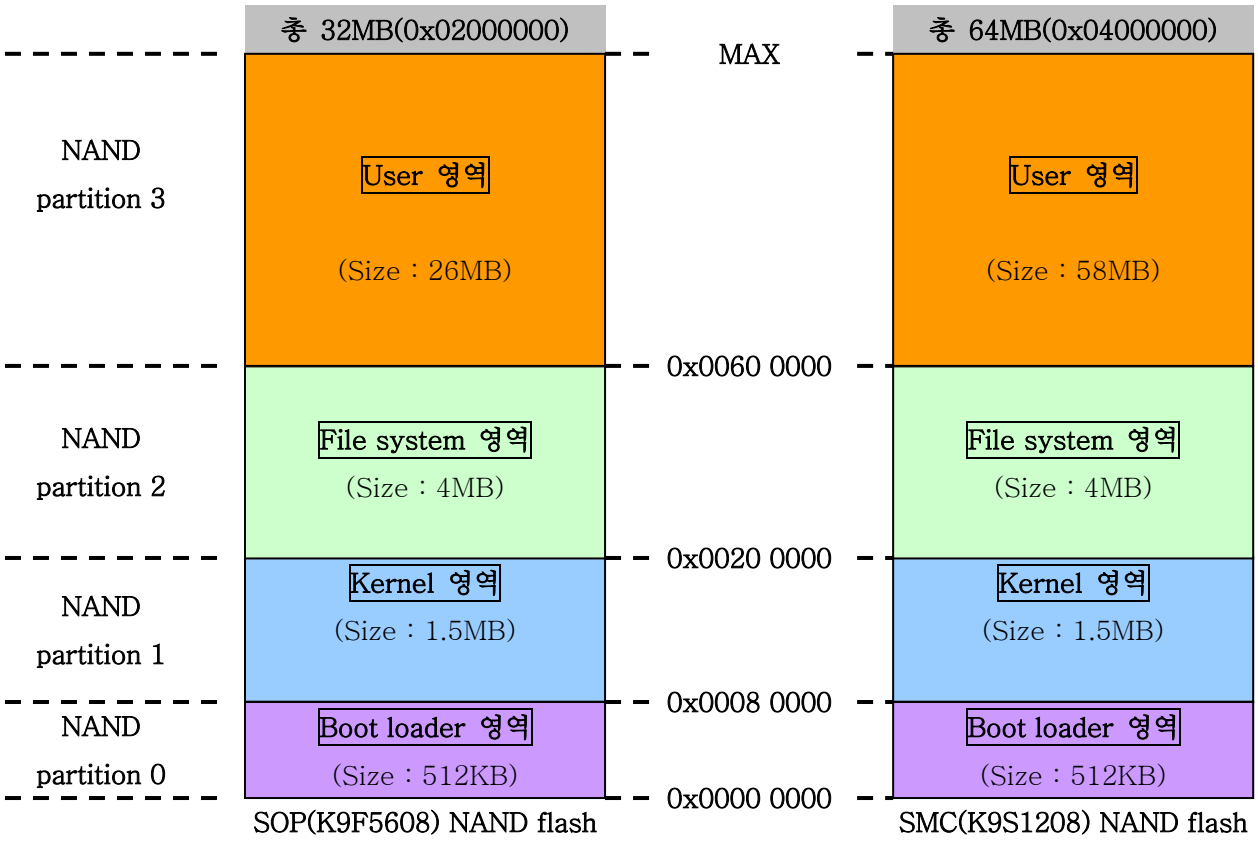
```
elfin_init: Initialized IIC on S3CX, 5kHz clock
iic_elfin_init: initialized iic-bus at 0xf4000000.
ELFIN UDA1341 audio driver initialized
NAND device: Manufacture ID: 0xec, Chip ID: 0x75 (Samsung NAND 32MB 3,3V)
Creating 4 MTD partitions on "NAND 32MB 3,3V":
0x00000000-0x00080000 : "NAND partition 0 : Bootloader"
0x00080000-0x00200000 : "NAND partition 1 : Kernel"
0x00200000-0x00600000 : "NAND partition 2 : File System"
0x00600000-0x02000000 : "NAND partition 3 : User"
usb.c: registered new driver hub
S3C2440 USBD Controller Core Initialized
USB Function Ethernet Driver Interface
NET4: Linux TCP/IP 1.0 for NET4.0
IP Protocols: ICMP, UDP, TCP, IGMP
IP: routing cache hash table of 512 buckets, 4Kbytes
TCP: Hash tables configured (established 2048 bind 2048)
eth0: using half-duplex 10Base-T (RJ-45)
IP-Config: Incomplete network configuration information.
NET4: Unix domain sockets 1.0/SMP for Linux NET4.0.
```

[그림 6.21] SOP(K9F5608) NAND flash partitioning 결과

```
elfin_init: Initialized IIC on S3CX, 5kHz clock
iic_elfin_init: initialized iic-bus at 0xf4000000.
ELFIN UDA1341 audio driver initialized
NAND device: Manufacture ID: 0xec, Chip ID: 0x76 (Samsung NAND 64MB 3,3V)
Creating 4 MTD partitions on "NAND 64MB 3,3V":
0x00000000-0x00080000 : "NAND partition 0 : Bootloader"
0x00080000-0x00200000 : "NAND partition 1 : Kernel"
0x00200000-0x00600000 : "NAND partition 2 : File System"
0x00600000-0x04000000 : "NAND partition 3 : User"
usb.c: registered new driver hub
S3C2440 USBD Controller Core Initialized
USB Function Ethernet Driver Interface
NET4: Linux TCP/IP 1.0 for NET4.0
IP Protocols: ICMP, UDP, TCP, IGMP
IP: routing cache hash table of 512 buckets, 4Kbytes
TCP: Hash tables configured (established 2048 bind 2048)
eth0: using half-duplex 10Base-T (RJ-45)
IP-Config: Incomplete network configuration information.
NET4: Unix domain sockets 1.0/SMP for Linux NET4.0.
```

[그림 6.22] SMC(K9S1208) NAND flash partitioning 결과

6.2.4.1 JFFS2 파일 시스템 Memory Map



[그림 6.23] JFFS2 파일 시스템의 memory map

6.2.3.2 CRAMFS로 리눅스 부팅하기에서 TFTP를 통해 CRAMFS 이미지를 다운로드 한 후, NAND flash에 write하는 영역이 바로 File system 영역(NAND partition 2)이다.

CRAMFS를 root 파일 시스템으로 하여 리눅스로 부팅한 경우에는 File system 영역을 절대 건드려서는 안된다.

6.2.4.2 Partition mount

커널이 부팅할 때, User Area 영역을 자동으로 마운트한다.

커널이 부팅한 후에 개발자의 의도에 따라 [그림 6.15]에서 보는 바와 같이 임의의 partition 영역을 마운트해서 쓸 수 있다.

그러나 주의할 점이 있다.

- Boot loader영역은 부트로더 이미지의 손상을 막기 위해 기본적으로 마운트되지 않도록 설정되어 있다. Boot loader영역은 마운트 되지 않음을 유의한다.
- CRAMFS를 root 파일 시스템으로 하여 부팅된 경우에는, File system 영역을 건드리지 않도록 한다. 리눅스 커널이 동작 중에 root 파일 시스템이 손상되면 동작하지 않게 됨을 유의한다.
- RAMDISK를 root 파일 시스템으로 하여 부팅한 경우에는 Boot loader영역을 제외한 모든 부분을 mount하여 사용할 수 있다.

이 장에서는 User Area 영역을 /mnt/mtd3 디렉토리에 마운트하여 사용해 본다.
CRAMFS를 root 파일 시스템으로 사용하든, RAMDISK를 root 파일 시스템으로 사용하든 상관없다.

동작 방법은 다음과 같다.
MBA2440용 임베디드 리눅스 커널이 부팅이 완료된 후,

1. User Area영역을 erase한다.
2. User Area 영역을 /mnt/mtd3 디렉토리에 마운트한다.
3. /mnt/mtd3에 임의의 파일을 복사한다.
4. 보드의 전원을 끈 후, 다시 리눅스를 부팅하고 마운트하여 복사한 파일의 존재 여부를 확인한다.

User Area 영역을 erase한다. erase하기 위해서 eraseall 이라는 utility를 사용한다. 이는 파일 시스템 안에 미리 넣어 두었다.

```
[root@Linux /root]$  eraseall  /dev/mtd/3
Erased 26624 Kibyte @ 0 -- 100% complete.
```

User Area 영역을 /mnt/mtd3 디렉토리에 마운트한다.

```
[root@Linux /root]$  mount  -t  jffs2  /dev/mtdblock/3  /mnt/mtd3/
```

임의의 파일을 복사한다. /etc/ 디렉토리에 있는 services 파일을 복사해 보자.

```
[root@Linux /root]$  cd  /mnt/mtd3/
[root@Linux /root]$  ls
[root@Linux /mtd3]$  cp  /etc/services  ./
[root@Linux /mtd3]$  ls  -al
-rw-r--r--     1 root      root         10788 Jul 31 00:01 services
```

services 파일이 복사되었고, 10778 바이트 크기임을 확인하였다.
이제 보드의 전원을 끄고, 잠시 기다린 후, 전원을 다시 켠다.
리눅스를 부팅한 후, 마운트해 본다.

```
[root@Linux /root]$  mount  -t  jffs2  /dev/mtdblock/3  /mnt/mtd3/
[root@Linux /root]$  cd  /mnt/mtd3/
[root@Linux /mtd3]$  ls  -al
-rw-r--r--     1 root      root         10788 Jul 31 00:01 services
```

파일이 존재함을 확인할 수 있다.

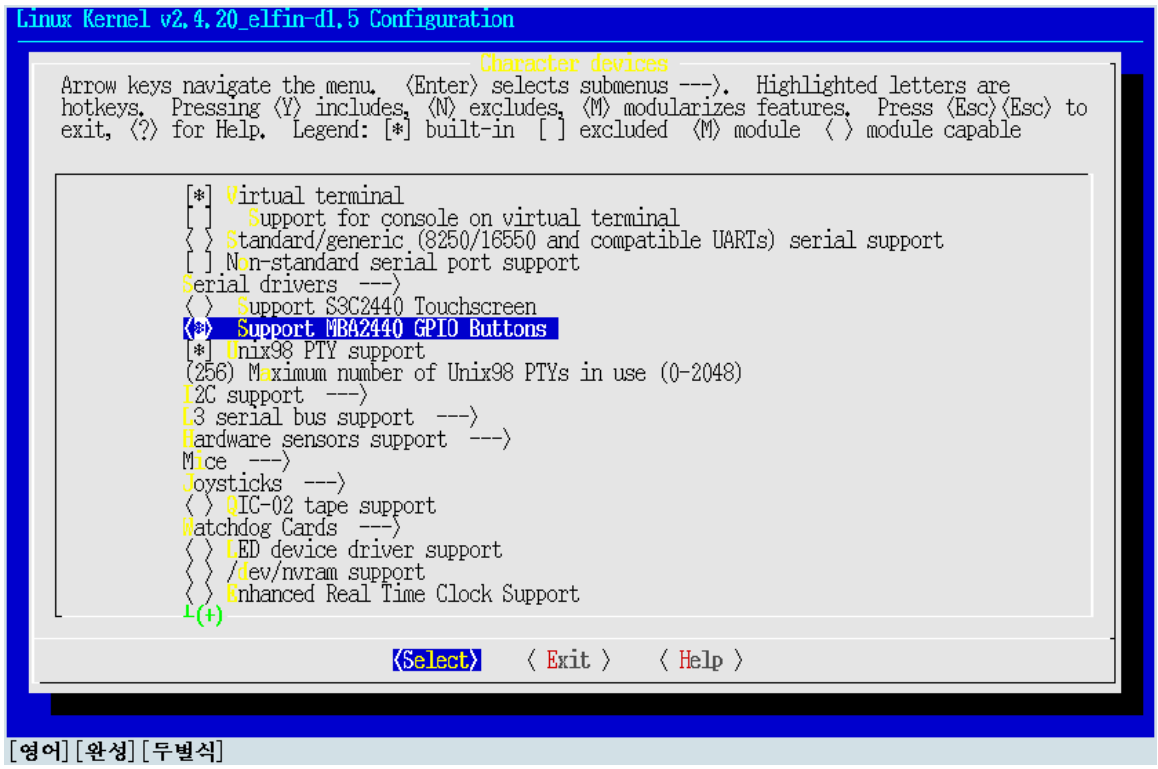
6.3 Device Driver

MBA2440용 임베디드 리눅스 커널에서는 external interrupt KEY1 ~ KEY4 버튼을 위한 character GPIO button device driver와 RTC(Real Time Clock) device driver를 제공한다.

6.3장에서는 KEY1 ~ KEY4 버튼을 위한 GPIO button device driver만을 소개한다.
RTC device driver와 어플리케이션은 6.4 Application 에서 다룬다.

6.3장에서는 RAMDISK 파일 시스템을 root 파일 시스템으로 설정하고 리눅스 커널 소스를 컴파일 할 것이다.

MBA2440용 리눅스 커널을 제공한 CD에서 리눅스 PC의 /root/mba2440 디렉토리에 복사하고 압축을 해제한다. (6.1 Kernel compile을 참조하라)
make menuconfig 명령을 통해 커널 컴파일 설정을 한다.
main menu에서 Character devices ---> 를 선택한다.



[그림 6.24]

[그림 6.24]에서 보는 바와 같이 스페이스 바를 눌러서 *표가 되게 선택한다. 스페이스 바를 누를 때마다 < > 내부의 표시가 변한다.

설정 내용을 저장하고 빠져 나온 후, 커널을 컴파일 한다.

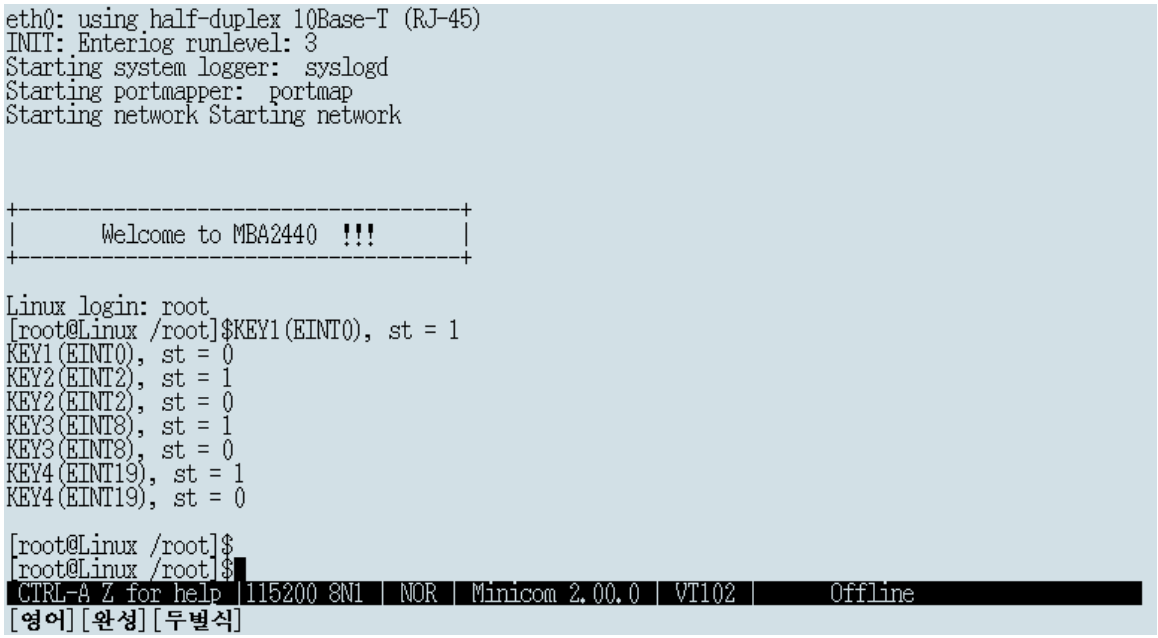
```
[root@localhost root]#  make  zImage
scripts/split-include include/linux/autoconf.h include/config
arm-linux-gcc  -D__KERNEL__  -I/root/mba2440/mba2440-linux2.4.20-v1.0/include  -Wall  -Wstrict-
```

```
prototypes -Wno-trigraphs -Os -fno-strict-aliasing -fno-common -ggdb -Uarm -fno-common -pipe -mapcs -mno-sched-prolog -g -mapcs-32 -D__LINUX_ARM_ARCH__=4 -march=armv4 -mtune=arm9tdmi -mshort-load-bytes -msoft-float -DKBUILD_BASENAME=main -c -o init/main.o init/main.c
.....
```

컴파일이 완료되면 arch/arm/boot/ 디렉토리에 있는 zImage 커널 이미지를 u-boot를 통해 MBA2440에 올리고 커널을 부팅한다.(2.4.1 TFTP로 리눅스 부팅하기 참조)

부팅이 완료된 후, KEY1 ~ KEY4 버튼 중 아무거나 눌러보자. 화면에 문장이 출력됨을 확인할 수 있다.

KEY1 ~ KEY4는 external interrupt로 설정되어 있으며, both edge detect로 설정되어 있어서 누를 때와 버튼을 뗄 때 두번 인터럽트가 걸린다.



[그림 6.25]

GPIO button device driver 소스는

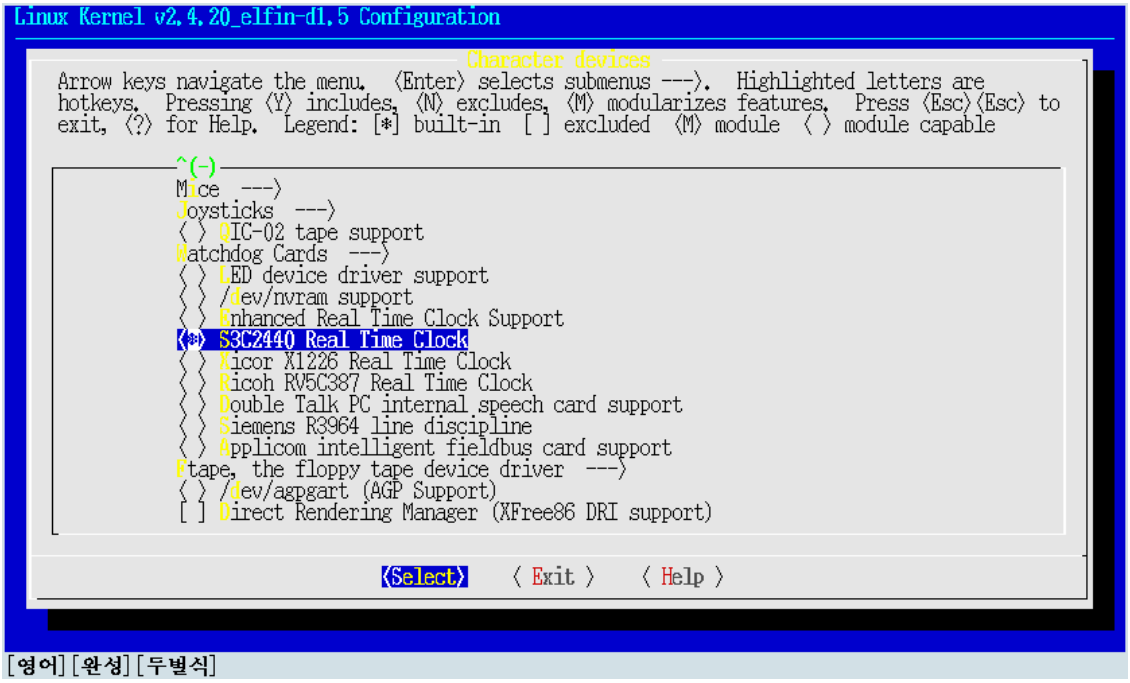
MBA2440용 임베디드 리눅스 커널 소스의 **drivers/char/** 디렉토리에 있는 **mba2440_gpio_button.c** 파일 이다.

소스는 직접 분석하기 바란다.

6.4 Application

이 장에서는 RTC device driver를 RTC device driver를 이용하는 application을 실행 시켜본다.
제공한 CD에 linux/application 디렉토리에 RTC test application 실행 파일과 소스가 있다.

먼저, 디바이스 드라이버를 컴파일 하자.
make menuconfig 명령을 통해 커널 컴파일 설정을 한다.
main menu에서 Character devices ---> 를 선택한다.



[그림 6.26]

[그림 6.26]에서 보는 바와 같이 * 표시가 되도록 설정한 후, 설정을 저장하고 빠져 나온다.

커널 소스를 컴파일 한다.

```
[root@localhost root]#  make  zImage
scripts/split-include include/linux/autoconf.h include/config
arm-linux-gcc -D__KERNEL__ -I/root/mba2440/mba2440-linux2.4.20-v1.0/include -Wall -Wstrict-prototypes -Wno-trigraphs -Os -fno-strict-aliasing -fno-common -ggdb -Uarm -fno-common -pipe -mapcs -mno-sched-prolog -g -mapcs-32 -D__LINUX_ARM_ARCH__=4 -march=armv4 -mtune=arm9tdmi -mshort-load-bytes -msoft-float  -DKBUILD_BASENAME=main -c -o init/main.o init/main.c
.....
```

이로써, RTC 디바이스 드라이버는 커널 이미지에 포함되었다.

제공한 CD에서 RTC를 사용하기 위한 어플리케이션 실행 파일인 **rtc-app** 파일을 /nfsroot 디렉토리에 복사하자.

```
[root@localhost root]#  cp  /mnt/cdrom/app/rtc/rtc-app  /nfsroot
[root@localhost root]#  ls  /nfsroot
ram2440.gz  root-mba2440.cramfs  rtc-app  zImage-cramfs.bin  zImage.bin
```

arch/arm/boot/ 디렉토리에 있는 zImage 커널 이미지를 u-boot를 통해 MBA2440에 올리고 커널을 부팅한다.(2.4.1 TFTP로 리눅스 부팅하기 참조)

부팅한 후에 NFS를 이용하여 /mnt/nfs를 리눅스 PC의 /nfsroot 디렉토리와 마운트한다.

마운트가 완료된 후, rtc-app 파일을 /root 디렉토리에 복사한다.

[그림 6.27]는 위 과정을 실행한 화면이다.

```
eth0: using half-duplex 10Base-T (RJ-45)
INIT: Entering runlevel: 3
Starting system logger:  syslogd
Starting portmapper:  portmap
Starting network Starting network



+--------------------------------------+
|       Welcome to MBA2440  !!!        |
+--------------------------------------+

Linux login: root
[root@Linux /root]$mount -t nfs 192.168.100.2:/nfsroot /mnt/nfs/
nfs warning: mount version older than kernel
[root@Linux /root]$ls /mnt/nfs/
zImage.bin              ram2440.gz              zImage-cramfs.bin
root-mba2440.cramfs     rtc-app                 s3c2440-rtc.o
mba2440_gpio_button.o
[root@Linux /root]$
[root@Linux /root]$cp /mnt/nfs/rtc-app .
[root@Linux /root]$ls
rtc-app
[root@Linux /root]$
 CTRL-A Z for help |115200 8N1 | NOR | Minicom 2.00.0 | VT102 |      Offline
[영어][완성][두벌식]
```

[그림 6.27]

rtc-app 어플리케이션을 실행하기 전에 /dev/ 디렉토리에 RTC 디바이스 드라이버를 위한 디바이스 파일을 만들어야 한다. 이것은 RTC어플리케이션과 RTC 디바이스 드라이버를 연결시켜 주는 역할을 한다.

mknod 명령을 이용한다. [그림 6.28]과 같이 명령을 수행한다.

```
[root@Linux /root]$ mknod /dev/rtc c 10 135

[root@Linux /root]$ ls /dev/rtc
/dev/rtc
[root@Linux /root]$ ls -al /dev/rtc
crw-r--r--    1 root     root      10, 135 Dec  1 00:36 /dev/rtc
[root@Linux /root]$




[영어][완성][두벌식]
```

[그림 6.28]

[그림 6.29]과 같이 실행해 본다.

```
[root@Linux /root]$./rtc-app
program start


2094년 24월 45일 31시 6분 45초


Set time you want..
enter year : 2005

enter month : 6

enter day : 20

enter hour : 14

enter minute: 24

enter second : 20

2005, 6, 20, 14, 24, 20


2005년 6월 20일 14시 24분 20초


program exit
[root@Linux /root]$
 CTRL-A Z for help |115200 8N1 | NOR | Minicom 2.00.0 | VT102 |      Offline
[영어][완성][두벌식]
```

[그림 6.29]

약간의 시간이 흐른 뒤 [그림 6.30]과 같이 실행해 본다.

```
[root@Linux /root]$
[root@Linux /root]$
[root@Linux /root]$
[root@Linux /root]$./rtc-app display
2005년 6월 20일 14시 25분 45초
[root@Linux /root]$




[영어][완성][두벌식]
```

[그림 6.30]

설정한 시간을 기준으로 초가 변했음을 확인할 수 있다.

7 WinCE

이 장에서는 CD에서 제공하고 있는 MBA2440용 WinCE 5.0 BSP(Board Support Package)를 WinCE 5.0 Platform Builder를 통해 컴파일 하여 이미지를 생성하는 방법과 WinCE 개발 과정에서 뿐만 아니라 WinCE 부팅을 위한 eboot에 대해서 알아본다.

이 장의 내용을 수행하기 전에 반드시 WinCE5.0의 Platform Builder가 설치되어 있어야 한다.

7.1 MBA2440용 WinCE 5.0 BSP

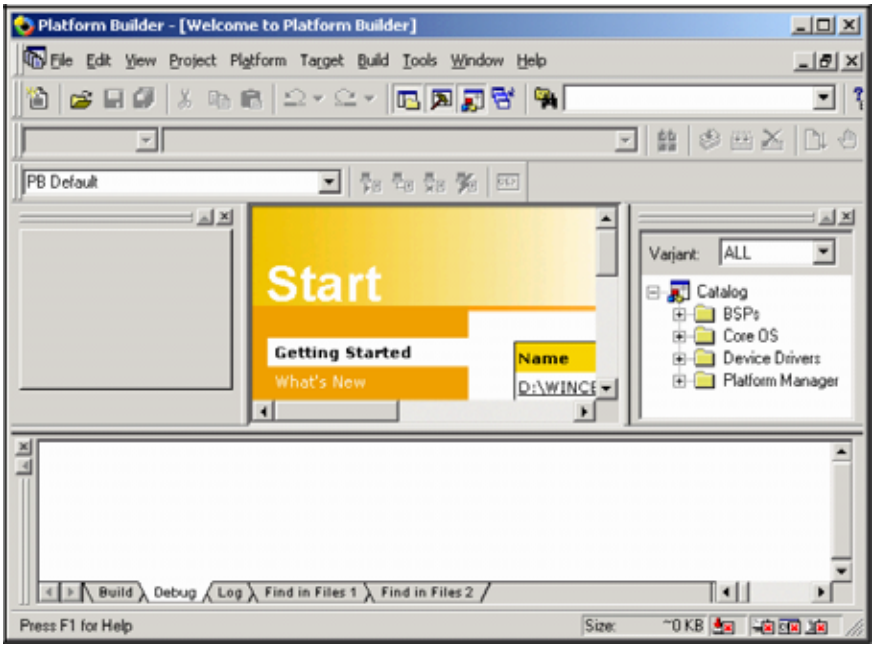
제공한 CD의 WinCE5.0 디렉토리에서 **MBA2440_ARM4.zip**이라는 WinCE5.0용 BSP를 얻을 수 있다.

MBA2440은 3.5"(240x320)TFT-LCD와 6.4"(640x480)TFT-LCD을 탑재할 수 있도록 설계되어 있다. MBA2440용 WinCE 5.0 BSP도 각각의 TFT-LCD에 따라 소스를 수정하여 사용할 수 있다. **CD에서 제공한 WinCE5.0 BSP는 기본적으로 3.5" TFT-LCD에서 동작하게 되어 있으며 영문을 지원하도록 설정** 되어 있다.

7.1.1 BSP Build

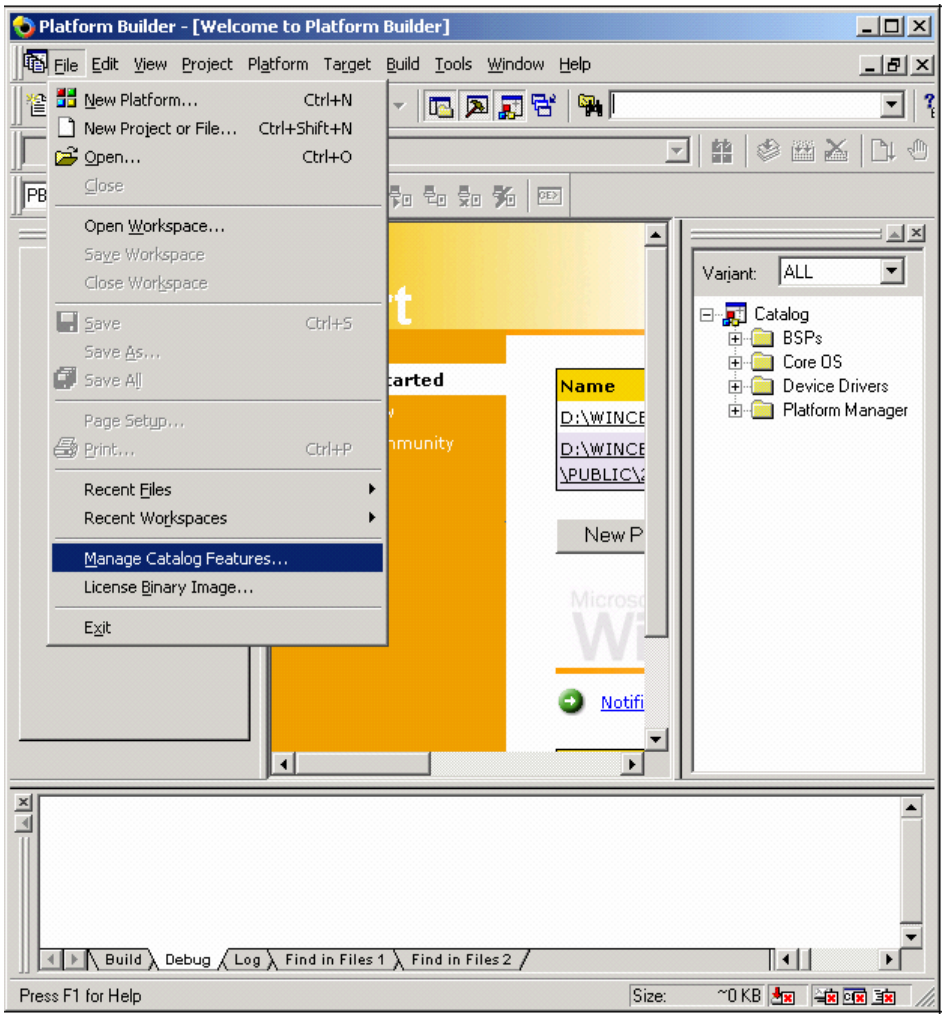
이제부터 BSP를 컴파일해 보자.

1. 위에서 언급한 CD안에 WinCE5.0 디렉토리에 있는 **MBA2440_ARM4_v1.0.zip** 파일을 복사하여 압축을 해제하면 MBA2440_ARM4_v1.0 이라는 디렉토리가 생긴다.
   이 디렉토리의 이름을 **MBA2440_ARM4로 수정**하고 **C:/WINCE5.0/PLATFORM/에 그대로 복사**한다.
2. **WinCE를 실행** 시킨다.

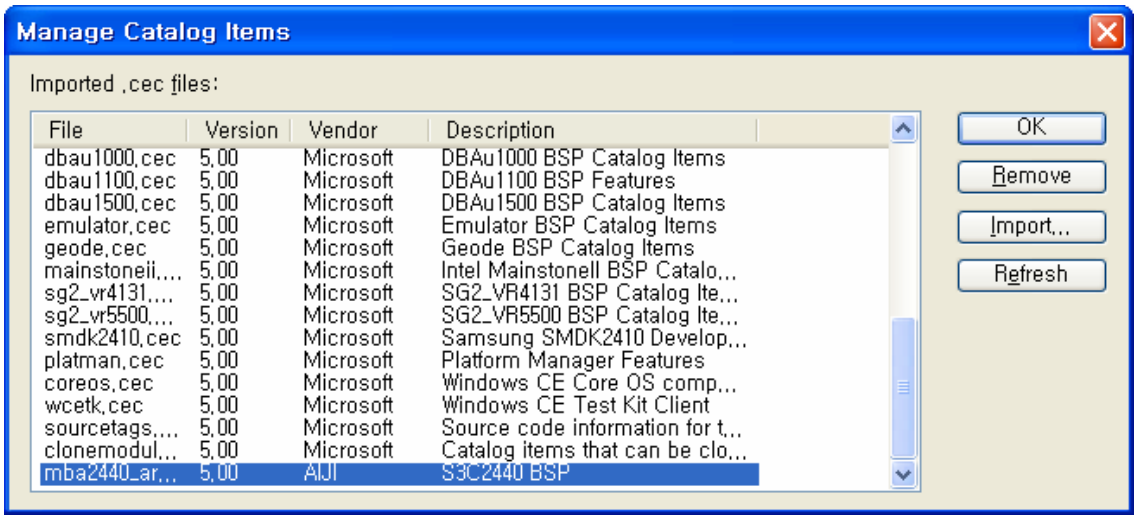


[그림 7.1] WinCE5.0을 실행한 모습

3. **Files->Manage Catalog Features...를 실행**하고 Manage Catalog Features 창에서 복사한 BSP 디렉토리에 있는 **MBA2440_ARM4.cec 파일을 선택하고 import 버튼을 클릭**한다.

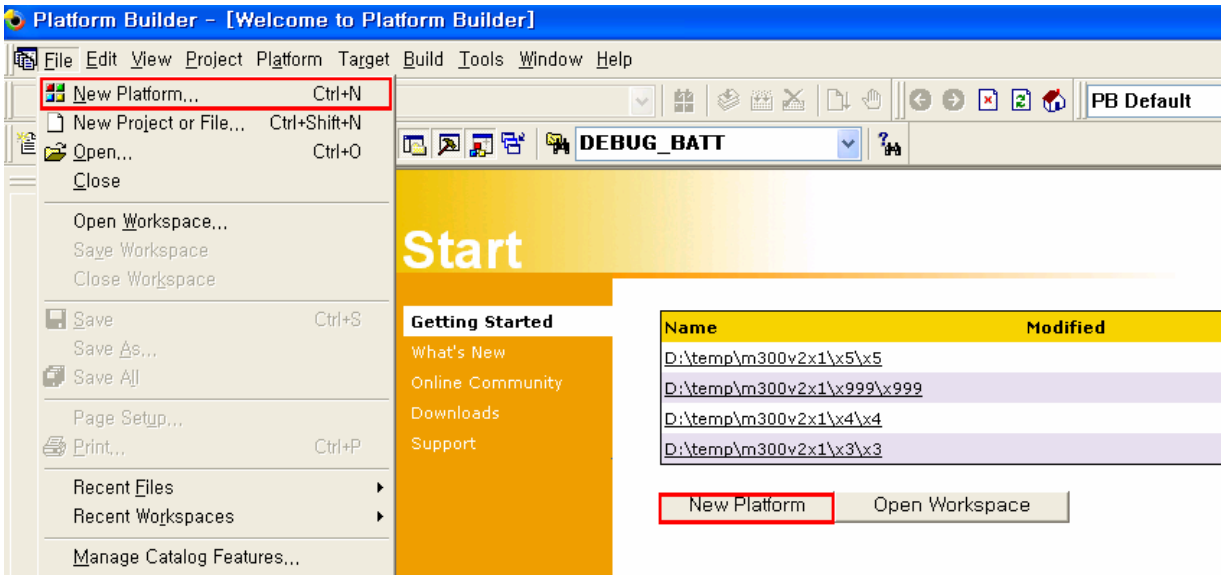


[그림 7.2]



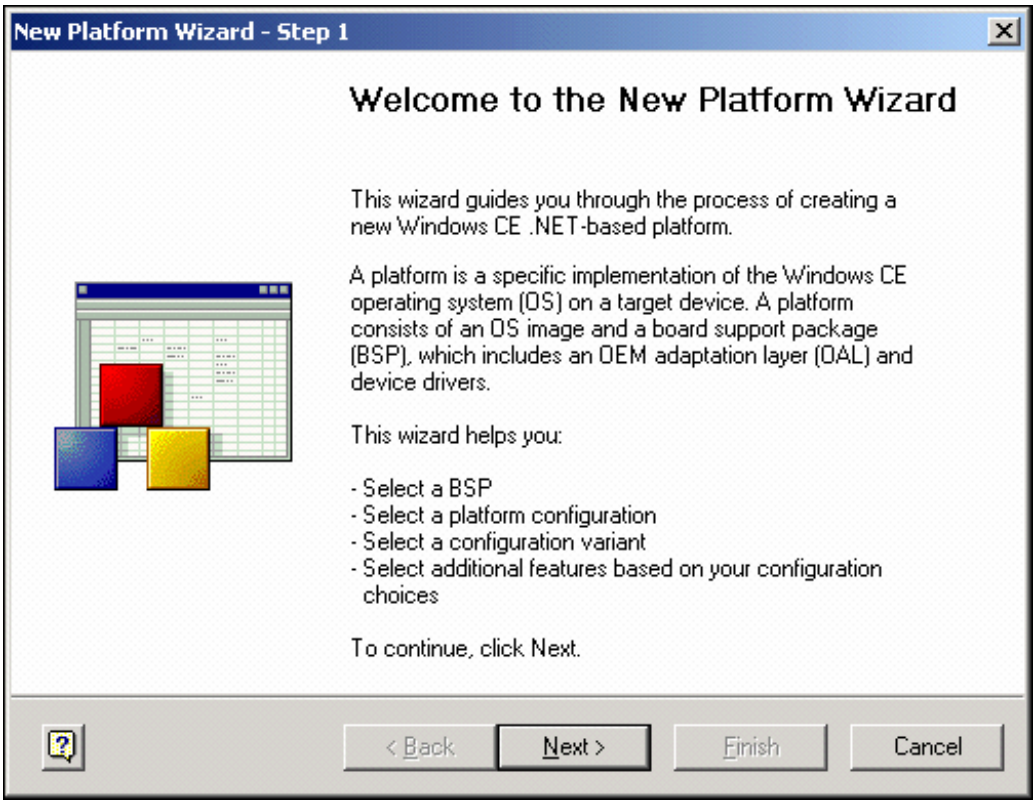
[그림 7.3] MBA2440_ARM4.cec file Import

4. Files->New Platform... 을 선택하거나 platform builder 중앙에 있는 New Platform 버튼을 누른다.



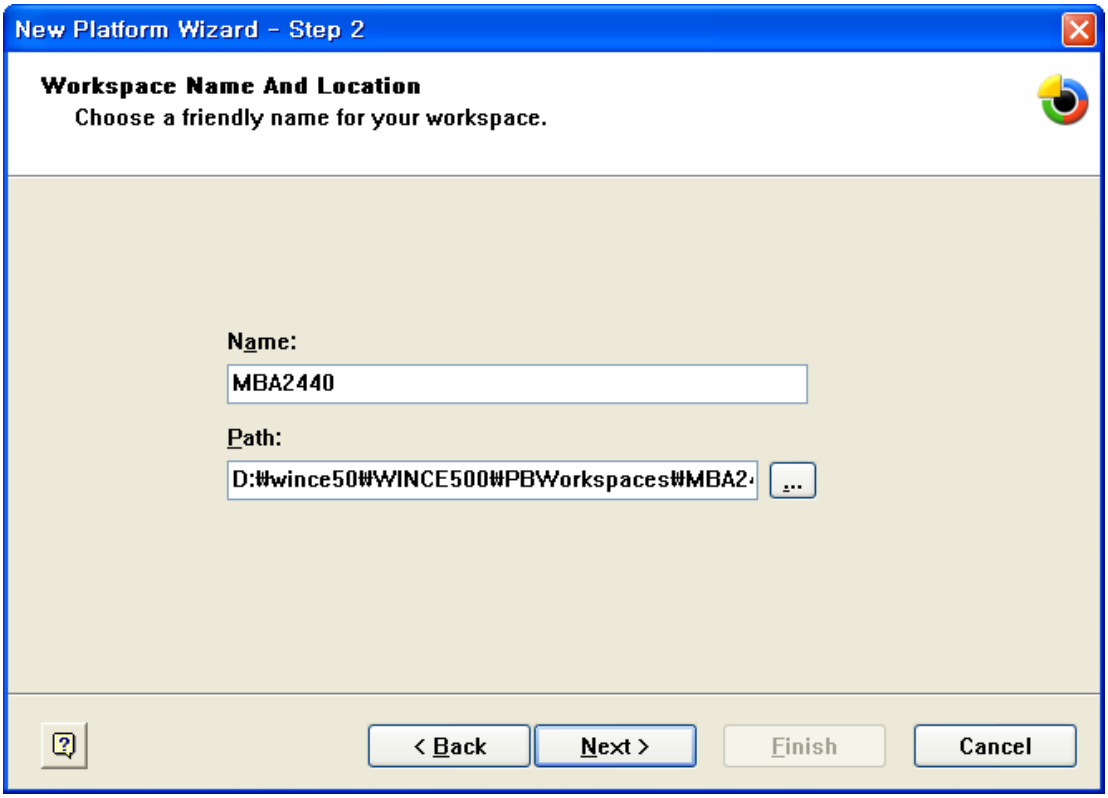
[그림 7.4]

5. Step 1 창에서는 Next 버튼을 누른다.



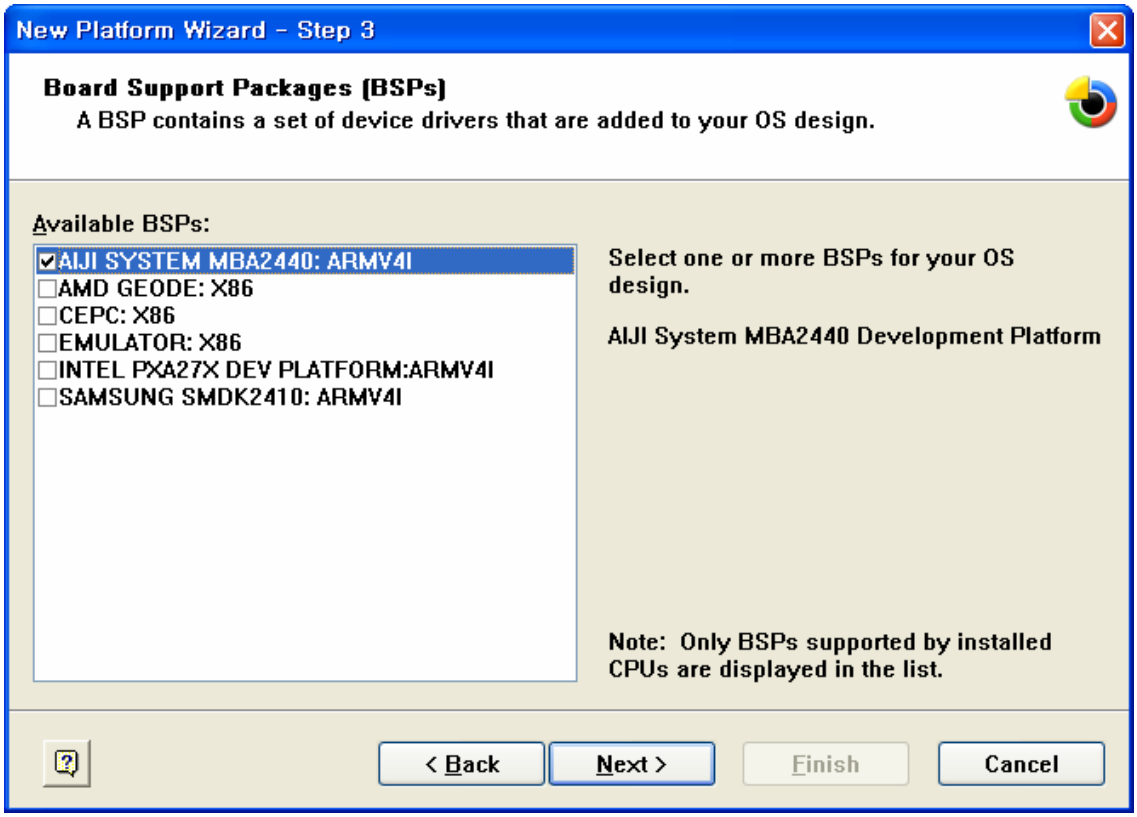
[그림 7.5]

6. Step 2 창에서는 자신이 원하는 Workspace 이름을 설정한다.



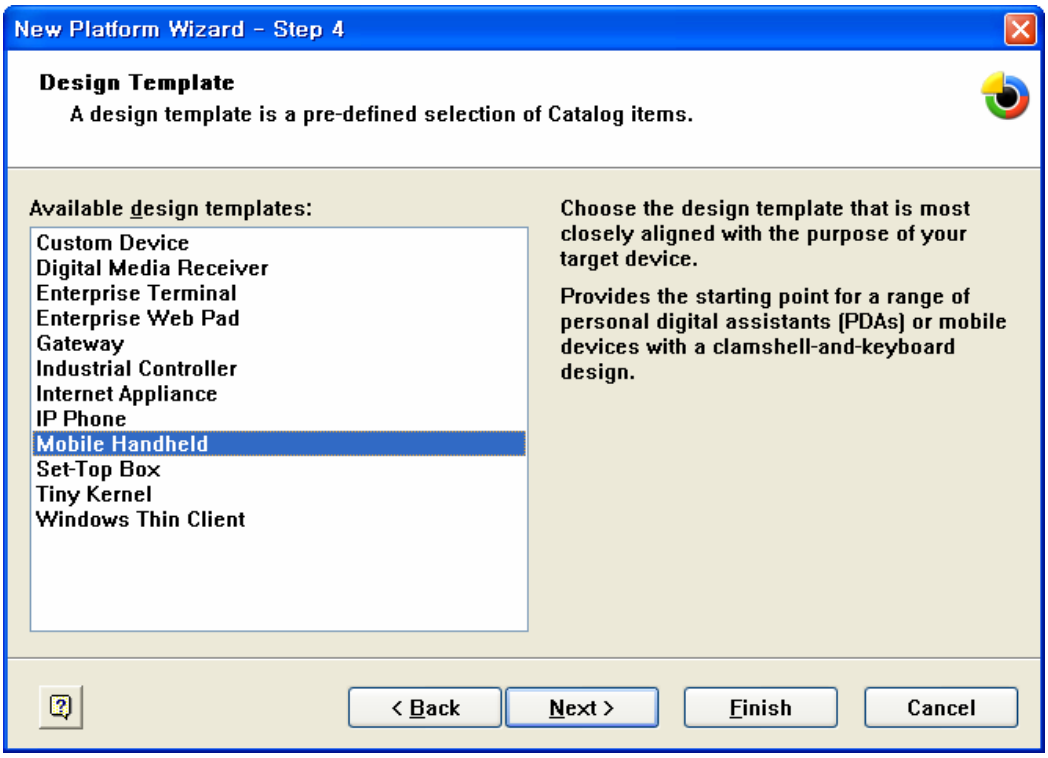
[그림 7.6]

7. Step 3 창에서는 AIJI SYSTEM MBA2440: ARMV4I를 선택하고 Next 버튼 누른다.



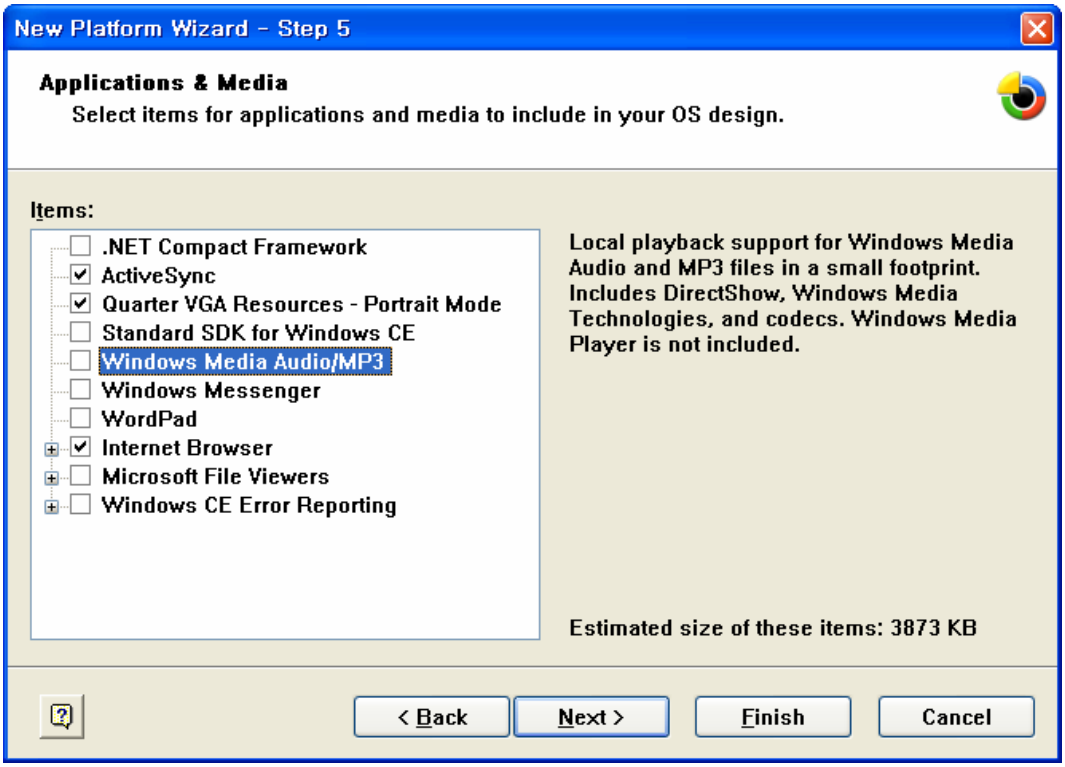
[그림 7.7]

8. Step 4에서는 Mobile Handheld를 선택하고, Platform name에는 자신이 원하는 Platform 명을 입력한다. 이 문서에서는 mba2440_test 라고 입력한다.



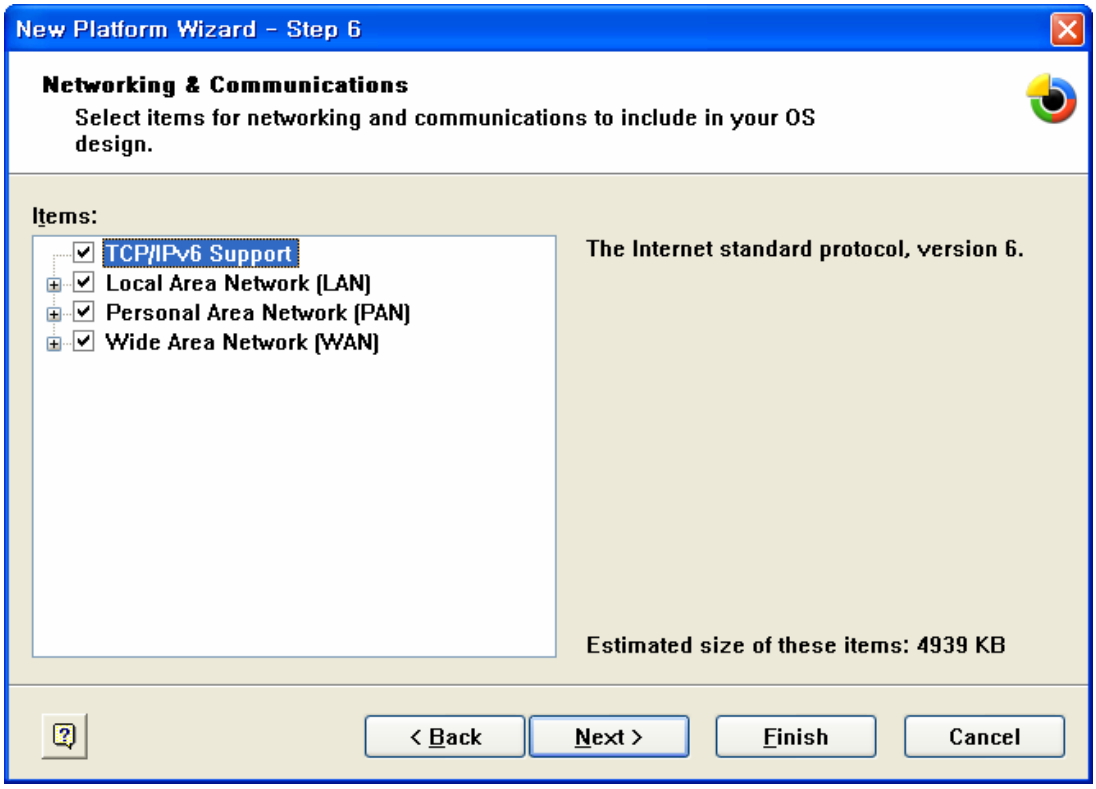
[그림 7.8]

9. Step 5에서는 [그림 7.9]에서 보는 바와 같이 check한다.



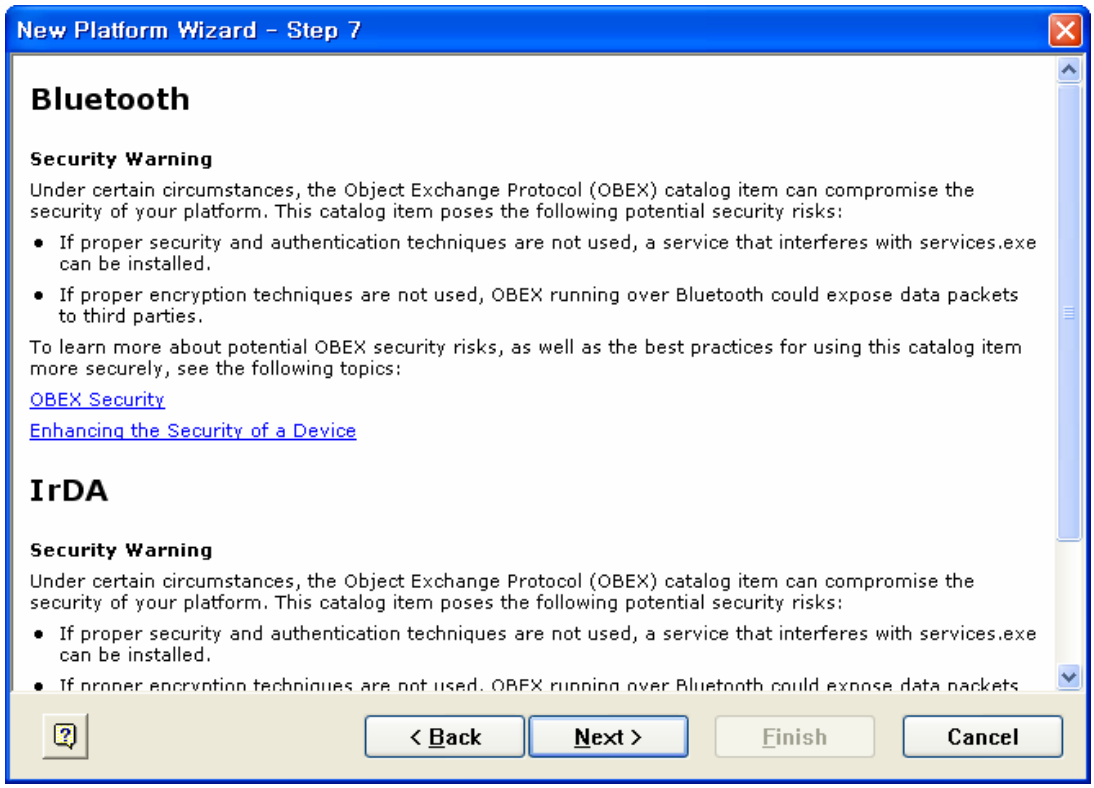
[그림 7.9]

10. Step 6에서는 수정할 필요 없이 그냥 Next 버튼을 누른다.



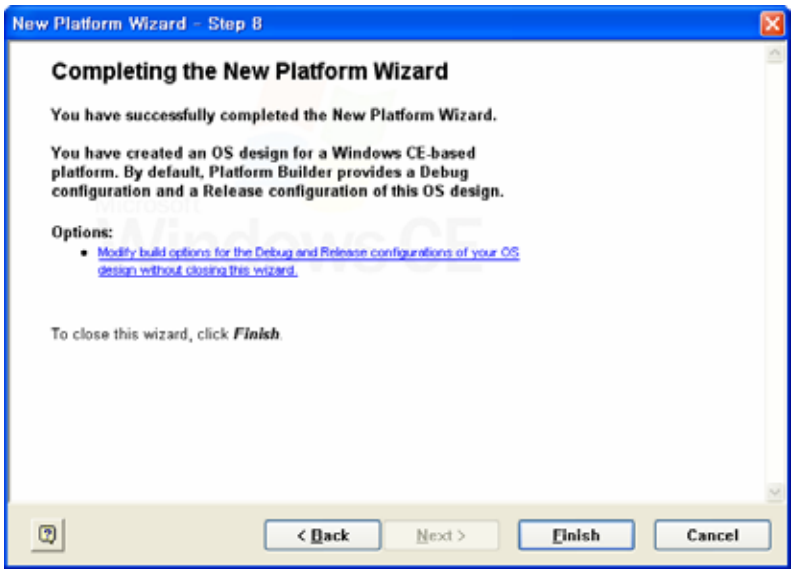
[그림 7.10]

11. Step 7에서도 그냥 Next 버튼을 누른다.



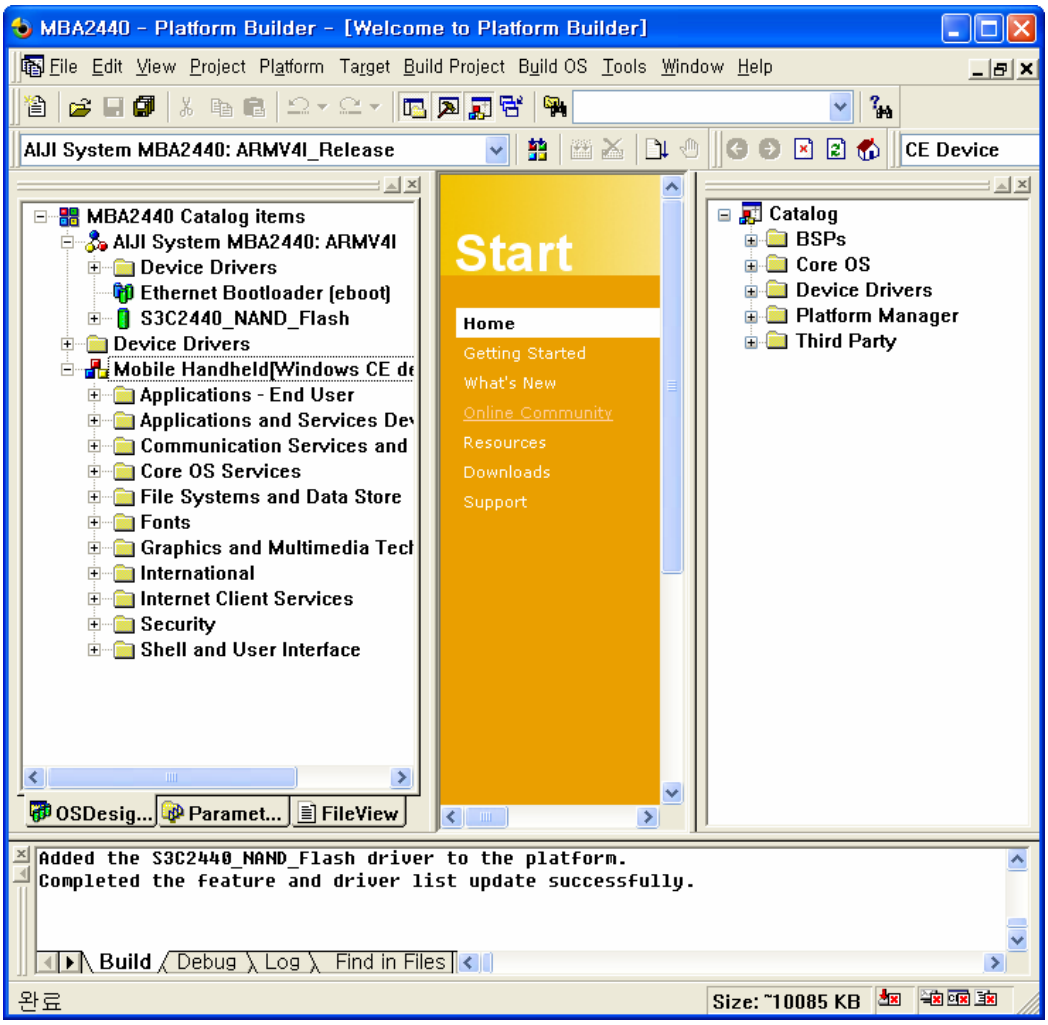
[그림 7.11]

12. Step 8에서 Finish 버튼을 눌러 New Platform Wizard를 마친다.



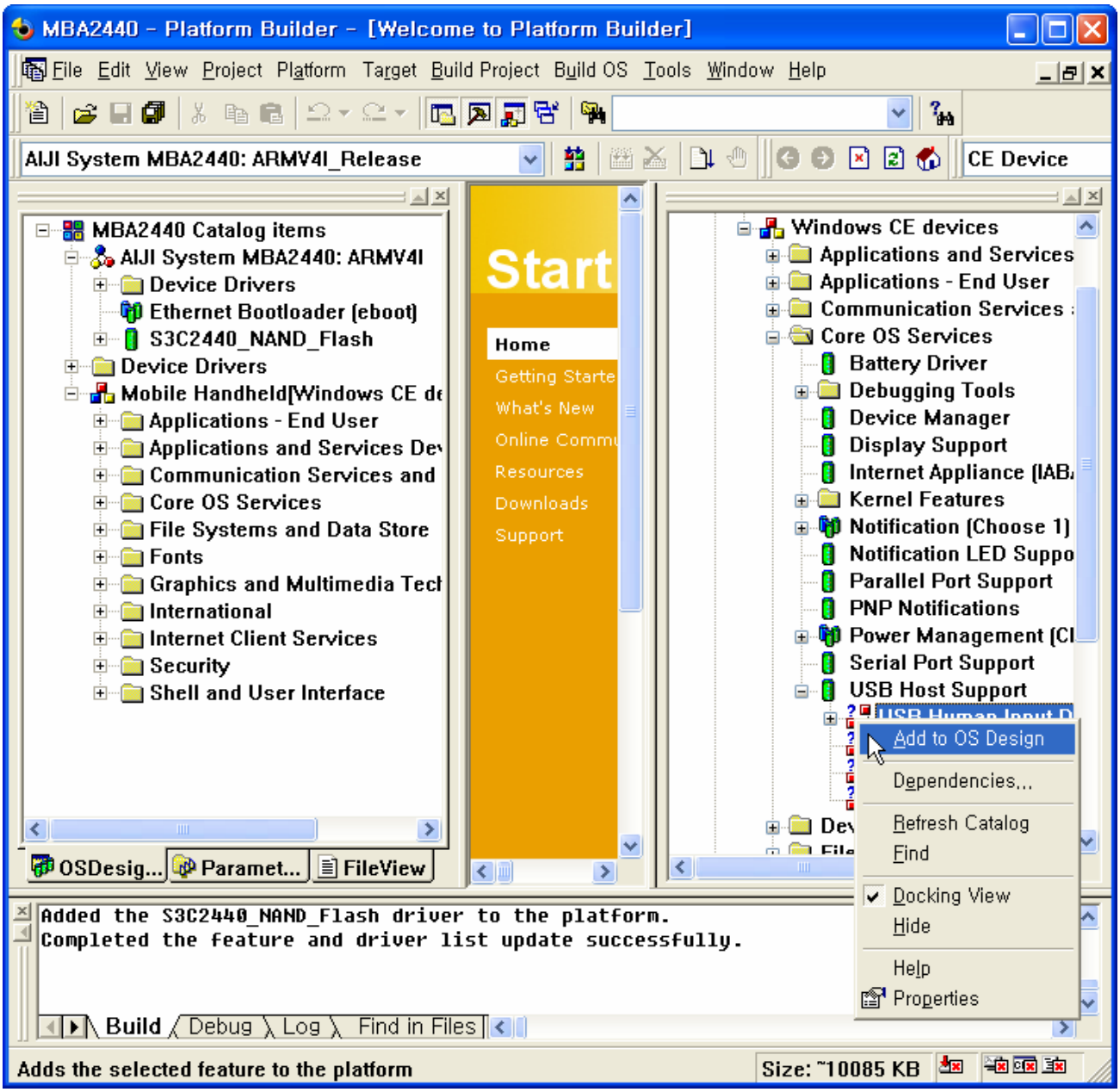
[그림 7.12]

13. New Platform Wizard를 마치고 나면 [그림 7.13]과 같은 화면이 뜬다.



[그림 7.13]

14. 우측에 있는 **카달로그 창에서 디바이스 드라이버와 Core OS에서 필요한 부분을 새로 생성한 platform에 추가**한다.



[그림 7.14]

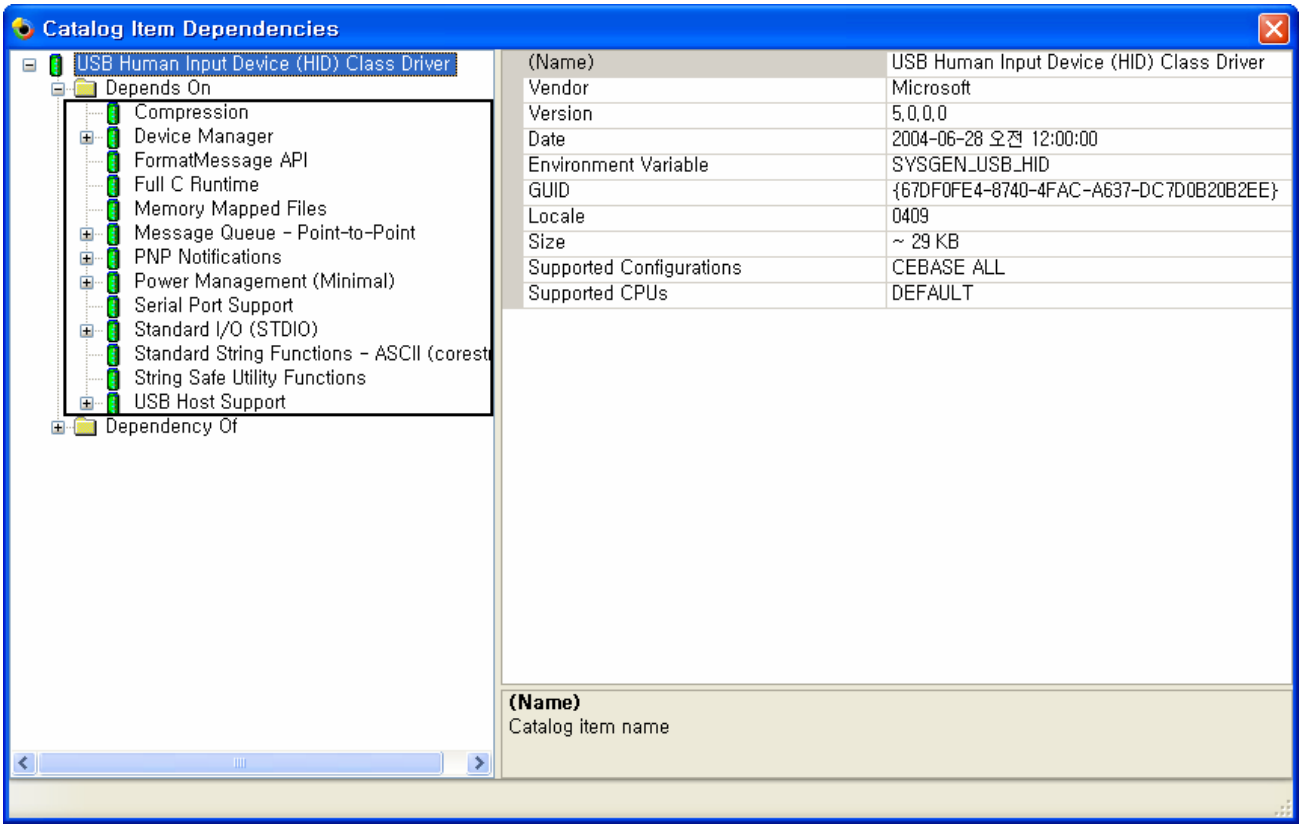
OS에 새로 추가하는 기능들은 개발자가 자신이 필요한 것들을 직접 추가할 수 있다.

이 부분에서 개발자가 무엇을 추가하느냐에 따라 WinCE 에서 제공하는 기능들이 달라지게 된다. CD에 제공하는 이미지는 가능한 모든 기능을 추가한 이미지이다. WinCE 이미지를 새로 만드는 과정에서는 개발자가 직접 일일이 찾아서 추가해 주어야 한다.

추가하고자 하는 기능에 마우스를 갖다 대고 마우스 오른쪽 버튼을 클릭하면 [그림 7.14]와 같이 팝업창이 뜨게 되며, Add to OS Design을 누르면 OS에 포함된다.

추가하고자 하는 기능에 대한 Dependency는 팝업창에서 Dependencies를 누르면 Item Dependencies 창이 뜨는데 이 창에서 확인할 수 있다.

[그림 7.15]에서 왼쪽의 Depends On 영역에 있는 것들을 찾아서 추가해 주면 된다.



[그림 7.15] Item Dependencies 창

WinCE 5.0 platform builder의 오른쪽 창에서 다음의 사항들을 추가해 준다.

► Catalog → Core OS → Windows CE devices → Core OS Services → USB Host Support에서
USB Human Input Device (HID) Class Driver를 추가

► Catalog → Core OS → Windows CE devices → Core OS Services → USB Host Support에서
USB Storage Class Driver를 추가

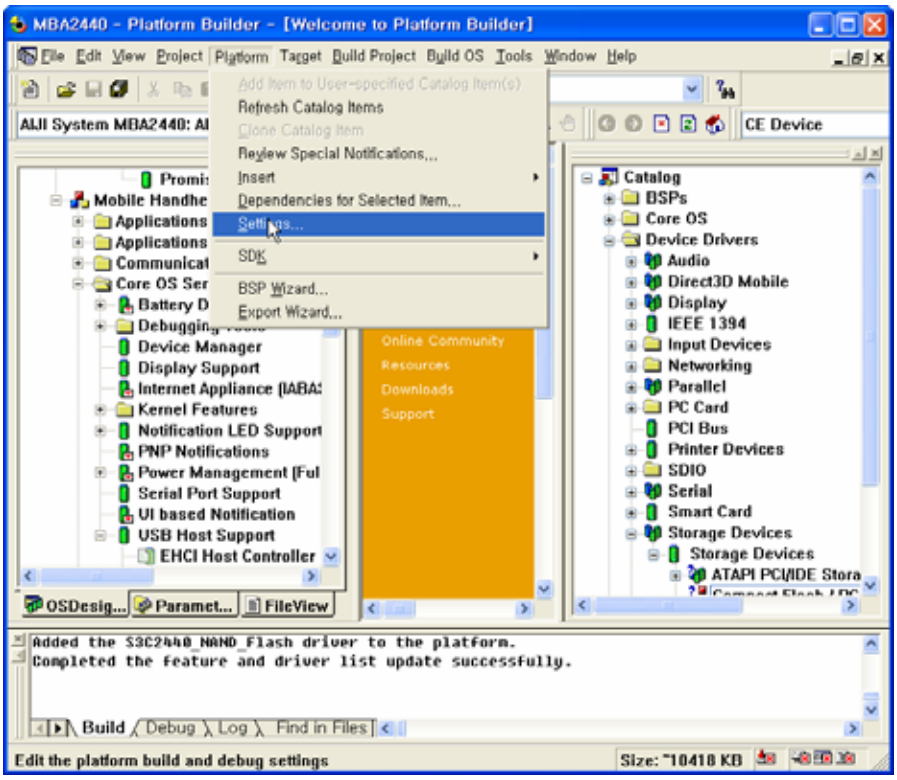
► Catalog → Core OS → Windows CE devices → Core OS Services → USB Host Support → USB Human Input Device(HID) Class Driver에서
USB HID Keyboard and Mouse를 추가

► Catalog → Core OS → Windows CE devices → File Systems and Data Store → Storage Manager 에서
FAT File System을 추가

► Catalog → Device Drivrers → Storage Devices → Storage Devices → ATAPI PCI/IDE Storage Block Driver 에서
ATAPI PCI/IDE Storage Block Driver를 추가

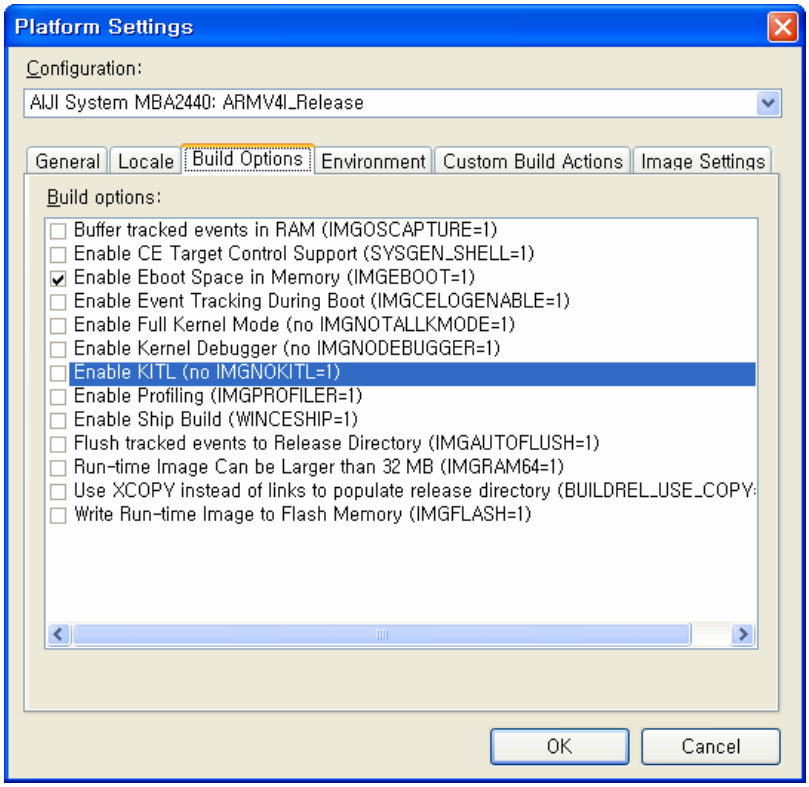
► Catalog → Device Drivrers → Storage Devices → Storage Devices 에서
Compact Flash/PC Card Storage[ATADISK]를 추가

15. Platform->Settings...를 선택한다.



[그림 7.16]

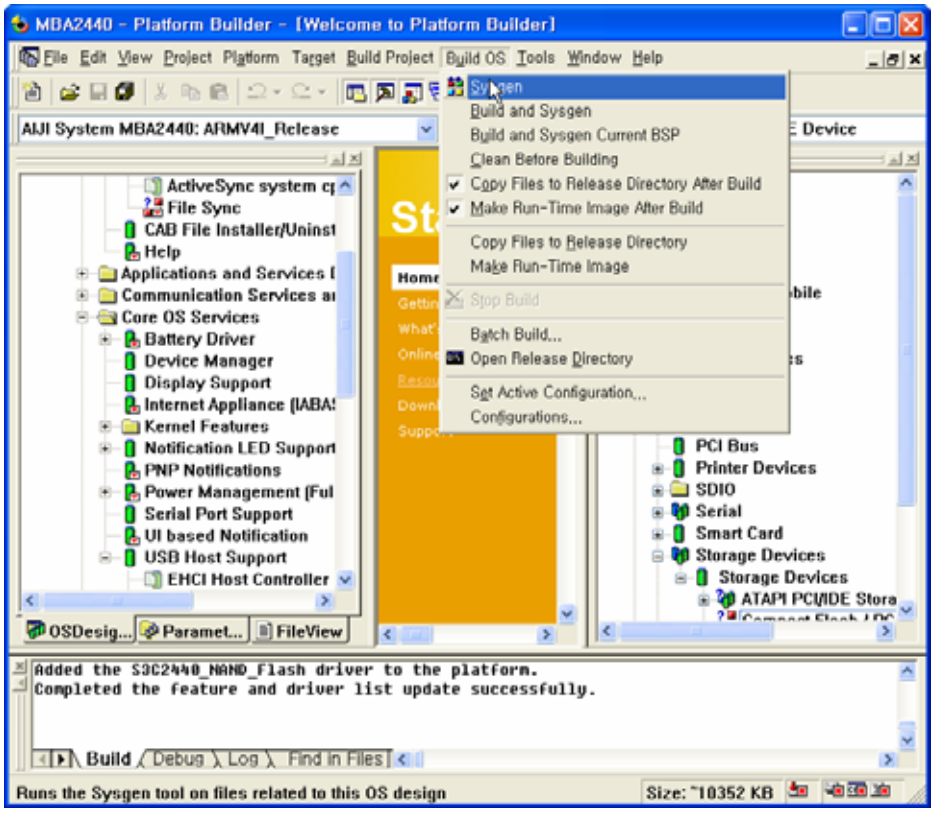
16. Platform settings 창의 Build Option탭에서 Enable Eboot Space in Memory만 선택한다.



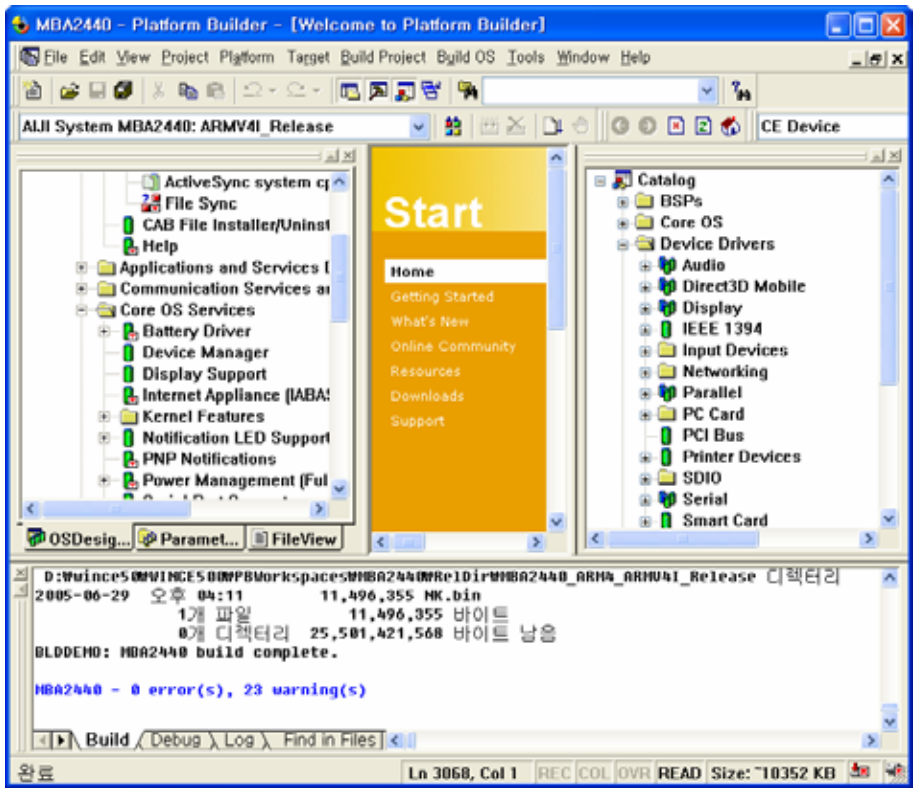
[그림 7.17]

17. 이제 컴파일을 한다. Build OS -> Sysgen을 선택하면 컴파일이 시작된다. 컴파일은 PC의 성능과

추가한 option에 따라 수분에서 수십 분이 소요된다.



[그림 7.18]



[그림 7.19] build가 완료된 모습

[그림 7.19]은 컴파일이 완료된 후의 모습이다.

컴파일이 완료되면

WINCE500/PBWorkspaces/MBA2440/RelDir/MBA2440_ARM4_ARMV4I_Release 디렉토리에서 NK.nb0 파일과 NK.bin 파일이 생성되어 있음을 확인할 수 있다.

u-boot나 NAND boot loader를 이용하여 NK.nb0 WinCE 이미지를 MBA2440에 적재하여 WinCE로 부팅할 수 있다. (2.3장 WinCE 부팅하기 참조)

7.1.2 6.4" TFT-LCD 지원

CD에서 제공하는 WinCE BSP는 기본적으로 3.5" TFT-LCD를 지원한다. 그러나, BSP를 수정하면 6.4" TFT-LCD를 지원할 수 있다.

이 장에서는 BSP에서 수정해야 할 부분에 대해서 알아본다.

1. s2440.h 에서 LCD_TYPE 매크로를 수정

BSP의 inc 디렉토리에 s2440.h 라는 헤더 파일이 있다.

이 헤더 파일에서 LCD_TYPE 매크로를 수정한다.

수정 전…

```
#define LCD_TYPE        TFT240_320
//#define LCD_TYPE        TFT640_480
```

수정 후…

```
//#define LCD_TYPE        TFT240_320
#define LCD_TYPE        TFT640_480
```

2. platform.reg에서 CalibrationData 수정

BSP의 FILES 디렉토리에 platform.reg 파일에서 CalibrationData 부분을 수정한다.

수정 전…

```
; for 640x480 display
;       "CalibrationData"="538,696 176,81 181,1293 874,1275 880,81"
; for 240x320 display
      "CalibrationData"="500.664 166,140 120,1172 888,1160 876,152"
```

수정 후…

```
; for 640x480 display
      "CalibrationData"="538,696 176,81 181,1293 874,1275 880,81"
; for 240x320 display
;       "CalibrationData"="500.664 166,140 120,1172 888,1160 876,152"
```

수정 완료 후, 재컴파일 한다.

7.1.3 SDRAM 메모리 설정

CD에서 제공하는 WinCE 5.0 BSP는 SDRAM 64MB MBA2440 보드에 맞춰져 있다. 128MB SDRAM으로 확장된 MBA2440 보드에서 WinCE를 동작시키기 위해서는 BSP를 128MB SDRAM에 맞게 수정해야 한다.

1. map.a 파일 수정

BSP의 KERNEL/HAL/ARM 디렉토리에 map.a 파일에서 다음과 같이 수정한다.

수정전…

```
DCD   0x8C000000,   0x30000000,   64    ; 64 MB DRAM BANK 0, 1
```

수정후…

```
DCD   0x8C000000,   0x30000000,   128    ; 128 MB DRAM BANK 0, 1
```

수정 완료 후, 재컴파일 한다.

7.1.4 한글 지원

CD에서 제공하는 WinCE 5.0 BSP는 기본적으로 영문만을 지원한다. 한글을 지원하기 위해서는 몇가지 BSP 수정 사항과 platform builder에서 build option을 수정해 주어야 한다.

한글 폰트 등이 WinCE 이미지 안에 포함되면서 이미지 크기가 상당히 커지기 때문에 컴파일 된 후 생성되는 이미지 크기를 조정해 주는 작업이 필요하다.

영문만 지원될 경우에는 WinCE 이미지 크기가 0x1400000(20MB 크기)이지만, 한글을 지원하도록 할 경우에는 0x2000000(32MB) 크기로 이미지 크기를 재조정해야 한다.

이를 위해 다음과 같이 수정한다.

1. config.bib 파일 수정

BSP의 FILES 디렉토리에 config.bib 파일에서 다음과 같이 수정한다.

수정 전…

```
NK              8C200000    01400000    RAMIMAGE
RAM             8D600000    01800000    RAM
……
……
ROMSIZE=01400000
```

수정 후

```
NK              8C200000    02000000    RAMIMAGE
RAM             8E200000    01800000    RAM
……
```

```
……
ROMSIZE=02000000
```

2. loader.h 파일 수정

BSP의 inc 디렉토리에 loader.h 파일에서 다음과 같이 수정한다.

수정 전…

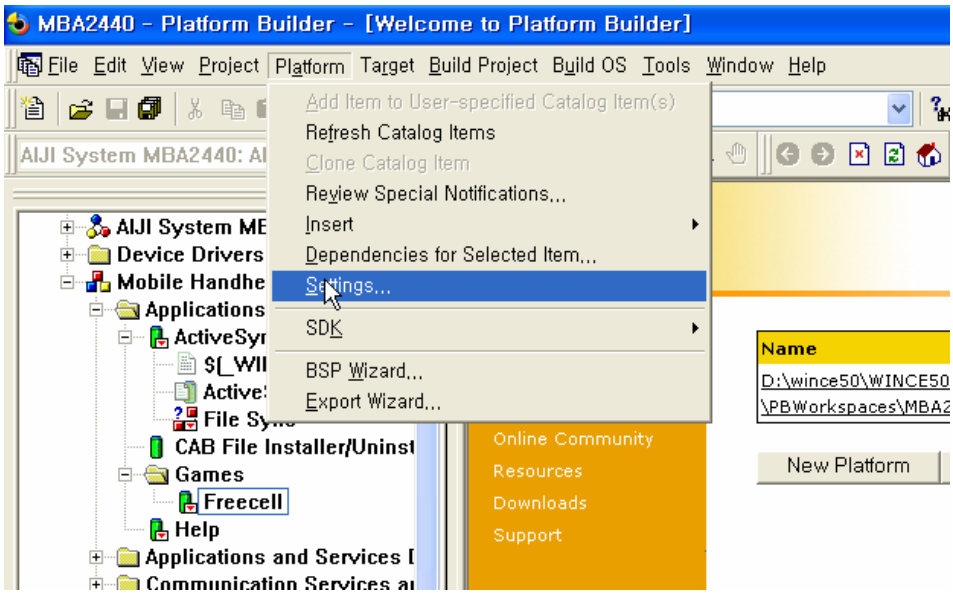
```
#define OS_RAM_IMAGE_SIZE        0x01400000
……
……
#define FLASH_CACHE_SIZE         0x01400000UL
```

수정 후…

```
#define OS_RAM_IMAGE_SIZE        0x02000000
……
……
#define FLASH_CACHE_SIZE         0x02000000UL
```

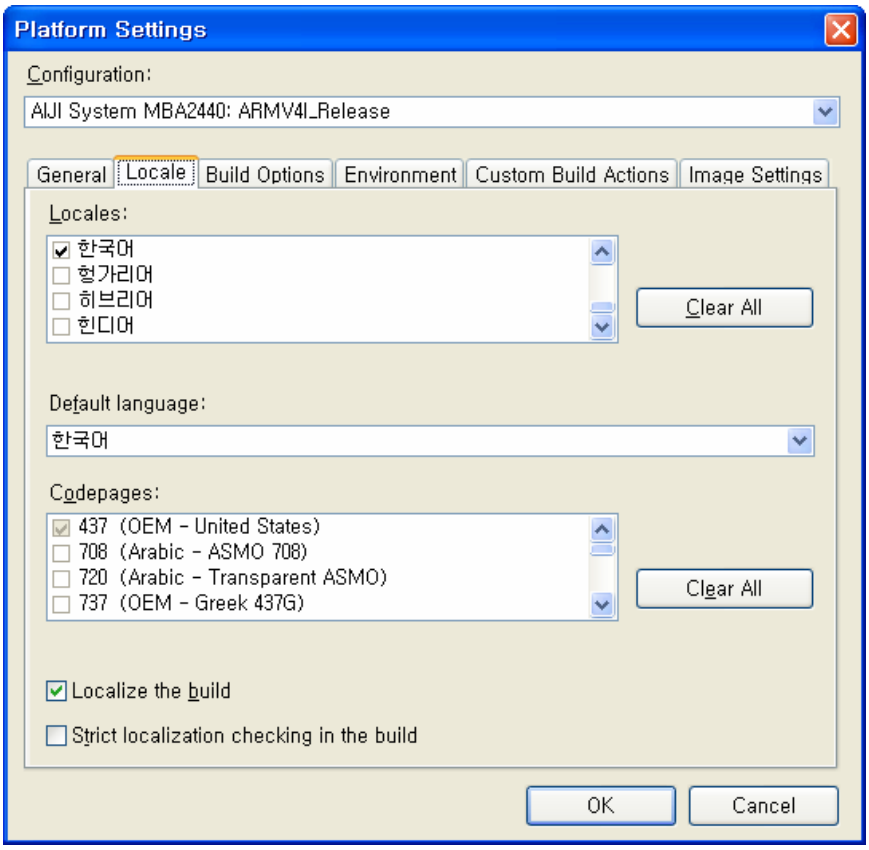
3. platform → settings 에서 locale 수정

WinCE platform builder에서 platform → settings를 선택하고 클릭한다.



[그림 7.20]

Platform settings 창에서 Locales는 한국어 체크, Default Language를 선택한다.



[그림 7.21]

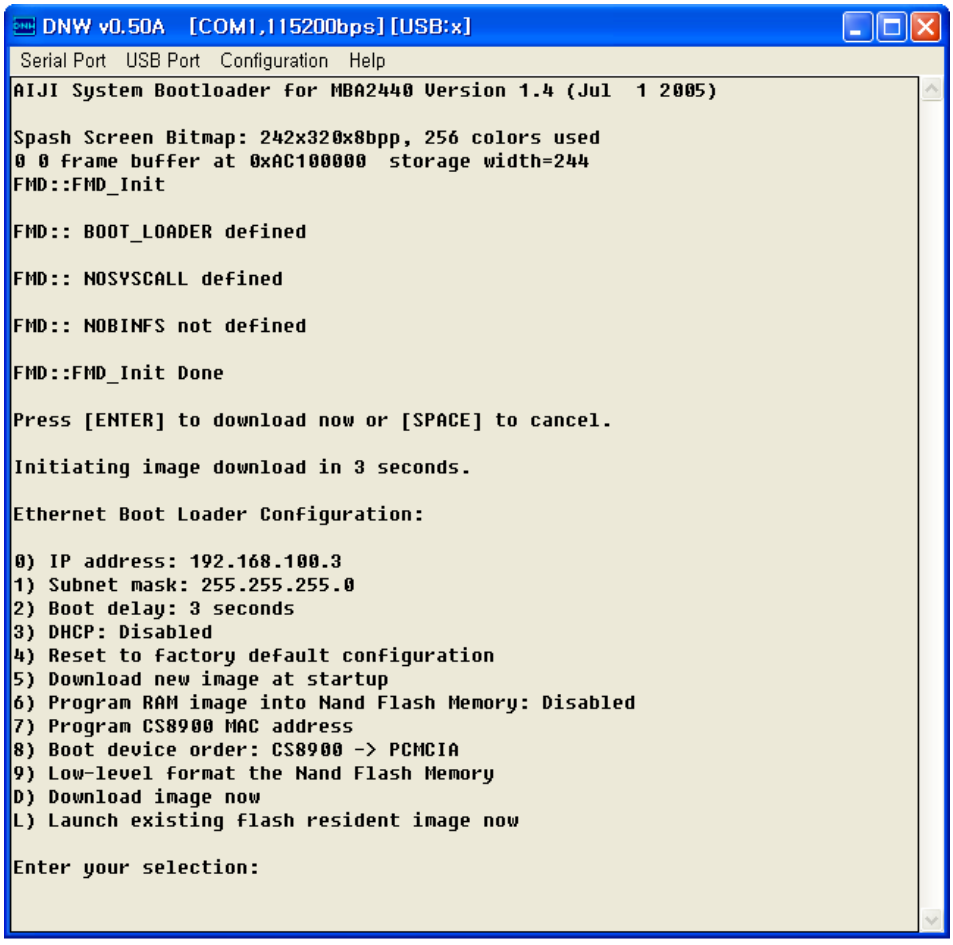
OK 버튼을 누른 후, 재컴파일 한다.

컴파일 완료된 후, WinCE 이미지를 MBA2440에 올리고 WinCE를 실행 시킨다.

7.2 Eboot boot loader

Eboot는 BSP 컴파일시 같이 컴파일 되어 생성된다. 해당 소스는 BSP 소스내의 eboot 디렉토리이다. 이 장에서는 eboot를 이용하여 보드에 WinCE 이미지를 올리고 부팅하는 방법에 대해 알아본다.

1. 먼저, WinCE 5.0이 설치되어 있는 PC와 MBA2440가 **LAN 케이블을 통해 연결되어 있어야 하며 네트워크 설정도 되어 있어야 한다.**
   데스트탑 PC의 IP주소는 192.168.100.1이고, MBA2440의 IP주소는 192.168.100.3으로 설정하자.(3.1 개발 환경 구축 예를 참조)
2. MBA2440 BSP**를 컴파일** 한다.(7.1.1 BSP build 참조)
   컴파일이 완료되면,
   WINCE50/PBworkspaces/MBA2440/RelDir/MBA2440_ARM4_ARMV4I_Release에
   EBOOT.nb0 파일과 EBOOT.bin 파일이 생성되었음을 확인할 수 있다.
   (제공한 CD의 WinCE5.0 디렉토리에 EBOOT.nb0 이미지를 사용해도 된다.)
3. EBOOT.nb0**를 AMD flash에 write**하자. (3.5.2 Boot flash에 이미지 쓰기 참조)
4. AMD flash에 write한 후, AMD flash를 boot flash로 설정하여 부팅하면 eboot가 부팅된다.
   부팅하자 마자 스페이스 바를 누르면 eboot 메뉴가 뜬다.



[그림 7.22]

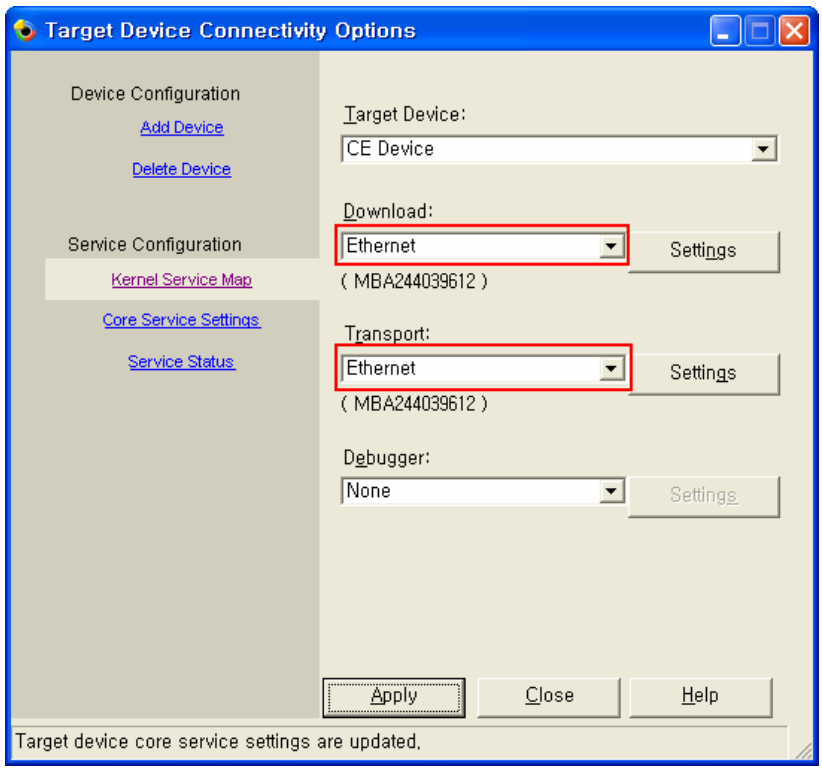
0번 메뉴에서 MBA2440의 IP address를 설정해 준다. : 192.168.100.3

1번 메뉴에서 Subnet mask를 설정한다. : 255.255.255.0

필요에 따라 7번 메뉴에서 임의의 MAC address를 설정한다 : 12:34:56:78:9a:bc

5. MBA2440과 PC간에 LAN 케이블이 제대로 연결되어 있는지 반드시 확인한다.

6. WinCE 5.0 platform builder의 Target->Connectivity Options를 선택하고 Target Device Connectivity Options창에서 [그림 7.23]와 같이 설정하고 Apply 버튼을 누른 후 Close 버튼을 눌러 종료 한다.



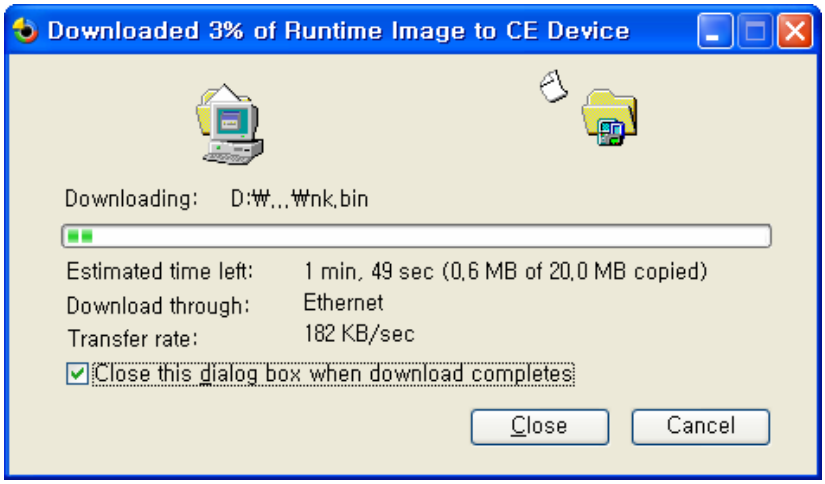
[그림 7.23]

7. Eboot의 메뉴에서 D) Download image now를 누른다. [그림 7.24]와 같이 연결된 device(MBA2440)을 찾는다.

```
Enter your selection: d
CS8900: MAC Address: 12:34:56:78:9A:BC

INFO: Probe: CS8900 is detected.

INFO: Init: CS8900_Init OK.

System ready!
Preparing for download...
INFO: Using static IP address 192.168.100.3.

INFO: Using subnet mask 255.255.255.0.

INFO: Using device name: 'MBA244039612'
+EbootSendBootmeAndWaitForTftp
Sent BOOTME to 255.255.255.255
Sent BOOTME to 255.255.255.255
Sent BOOTME to 255.255.255.255
```

[그림 7.24]

8. WinCE 5.0 platform builder의 Target->Attach Device를 누른다.



[그림 7.25]

[그림 7.25]과 같이 MBA2440에 nk.bin 파일을 다운로드 하게 된다.
다운로드가 완료되면 WinCE가 부팅된다.

7.3 Nboot boot loader

이 장에서 설명하고자 하는 nboot 부트로더는 eboot나 u-boot와 같이 하드웨어를 초기화 하는 기능을 담당하지는 않는다. 다만, MBA2440상에서 NAND flash를 boot flash로 설정하여 WinCE 이미지를 부팅하고자 할 때, NAND flash의 block 0에 적재되어 보드 부팅 후, WinCE 이미지를 메모리에 복사하고 이 WinCE 이미지를 실행 시키는데 사용되는 firmware이다. eboot나 u-boot 보다 덩치가 매우 작다.

Nboot의 실제 동작과정은 매우 단순하다.

Nboot 이미지는 4KB이하이다. MBA2440에서 NAND flash로 부팅하게 되면, S3C2440은 먼저 NAND flash의 맨 처음 4KB의 데이터를 버퍼로 읽어와서 이를 수행하게 된다.

이 버퍼는 4KB 크기의 SRAM 버퍼로서 S3C2440 내부에 존재하며 이를 **stepping stone 버퍼**라 한다. 이 때문에 **nboot는 크기가 4KB 이하이며 NAND flash의 0번 블록에 저장되는 것**이다.

WinCE 이미지는 NAND flash의 1번 블록부터 저장되어 있으며, nboot는 stepping stone 버퍼에 적재된 후, 동작하기 시작하면 1번 블록부터 WinCE 이미지 크기만큼을 NAND flash에서 읽어서 이것을 SDRAM 시스템 메모리의 0x30200000 번지에 복사하게 된다.

복사가 완료되면, 제어권을 WinCE 이미지에게 넘겨 주게 되고, 이 이후부터는 WinCE 이미지가 동작하게 되는 것이다.

Nboot 소스는 제공한 CD의 firmwares/nboot 디렉토리에 **MBA2440_nand_loader.zip** 이라는 압축파일이다.
Main함수는 2440loader.c 라는 파일 안에 있으며 ADS1.2로 개발 되었다.

2440loader.c 에서 참고해야 할 매크로들이 있다.

```
#define BOOT_IMAGE_SIZE                    (0x1400000)
    ▪ ▪ ▪
#define  BOOT_START_ADDR_OFFSET            (0x200000)
    ▪ ▪ ▪
#define DOWNLOAD_ADDRESS   (_RAM_STARTADDRESS+ BOOT_START_ADDR_OFFSET)
```

위 매크로는 nboot의 핵심이다.

BOOT_IMAGE_SIZE

: 이 매크로는 WinCE 이미지의 크기를 나타낸다. CD에서 제공하는 WinCE 이미지의 크기는 20MB이기 때문에 여기서는 0x14000000으로 설정되어 있다. 만약, WinCE 이미지를 새로 build 할 때, 이미지의 크기가 변경되었다면 nboot에서도 이를 수정해 주어야 WinCE 이미지를 제대로

메모리에 적재할 수 있게 된다.

DOWNLOAD_ADDRESS

: 이 매크로는 WinCE 이미지를 NAND flash에서 복사한 후 시스템 메모리(SDRAM)의 어느 위치에 적재할 지를 결정해 주는 매크로이다.

SDRAM의 **실제 물리 주소는 0x30000000(_RAM_STARTADDRESS)**이고, **WinCE 이미지의 start offset은 0x200000(BOOT_START_ADDR_OFFSET)**이므로, 결과적으로 **DOWNLOAD_ADDRESS의 값은 0x30200000 번지**가 된다.

이 매크로는 후에 개발자의 편의에 따라 수정되어 사용될 수 있다.

WinCE는 메모리의 0x30200000 번지에 WinCE 이미지가 존재해야만 부팅이 된다. 이것은, WinCE bsp에 설정되어 있기 때문이다.

자세한 부분은 소스를 직접 분석해 보기 바란다.

7.3.1 Nboot로 WinCE 실행하기

u-boot 상에서 tftp로 Nboot 이미지와 WinCE 이미지를 메모리에 적재한 후 이를 다시 NAND flash memory에 적재하고 해당 NAND flash를 boot flash로 설정하여 전원을 켬과 동시에 WinCE를 실행해 보자.

Nboot는 u-boot 와는 달리 NAND flash로 부팅하게 될 경우 제일 먼저 실행되어 WinCE 이미지를 메모리에 적재하고 이 이미지에게 제어권을 넘겨 주는 역할만을 하는 부트로더이다.

동작 방법은 크게 다음과 같이 구분된다.

1. Nboot 이미지와 WinCE 이미지를 tftp 명령을 통해 메모리에 다운로드한다.
2. 다운로드한 이미지들을 NAND flash에 write한다.
3. write한 NAND flash를 boot flash로 설정한 후 부팅한다.

u-boot가 NOR flash에서 동작하든지 NAND flash에서 동작하든지 J3을 통해 어떤 NAND flash에 write할지를 결정해야 한다.

J3이 1-2로 설정되면 SOP(K9F5608)에, 2-3으로 설정되면 SMC(K9S1208)에 이미지들을 write하게 된다.

u-boot 상에서 NAND 부트로더 이미지와 WinCE이미지를 다운로드 한다.

Nboot 이미지는 nboot.bin이며, WinCE 이미지는 NK.nb0라 하자.

Nboot 이미지를 tftp 명령을 통해 다운로드 한다.

```
MBA2440 #  tftp  30000000  nboot.bin
TFTP from server 192.168.100.2; our IP address is 192.168.100.3
Filename 'nboot.bin'.
```

```
Load address: 0x30000000
Loading:-
done
Bytes transferred = 4024 (fb8 hex)
```

WinCE 이미지를 tftp 명령을 통해 다운로드 한다.

```
MBA2440 #  tftp  30004000  NK.nb0
TFTP from server 192.168.100.2; our IP address is 192.168.100.3
Filename 'NK.nb0'.
Load address: 0x30004000
Loading:/
done
Bytes transferred = 20971520 (1400000 hex)
```

다음으로 이 이미지를 NAND flash에 write한다.

```
MBA2440 #  nande  all
erase all blocks [0 ~ 2047]
Erasing block ..-
Complete erasing block
MBA2440 #  nandw  t  30000000  0  1500000
Write start point : block[0] page[0] size = 0x1500000
Writing data .../
Write compelete(0x1500000): from 0x30000000 SDRAM to 0x0(0) NAND flash
MBA2440 #
```

보드의 전원을 끈 후에, write했던 NAND flash를 boot flash로 설정한 후, 전원을 켠다.

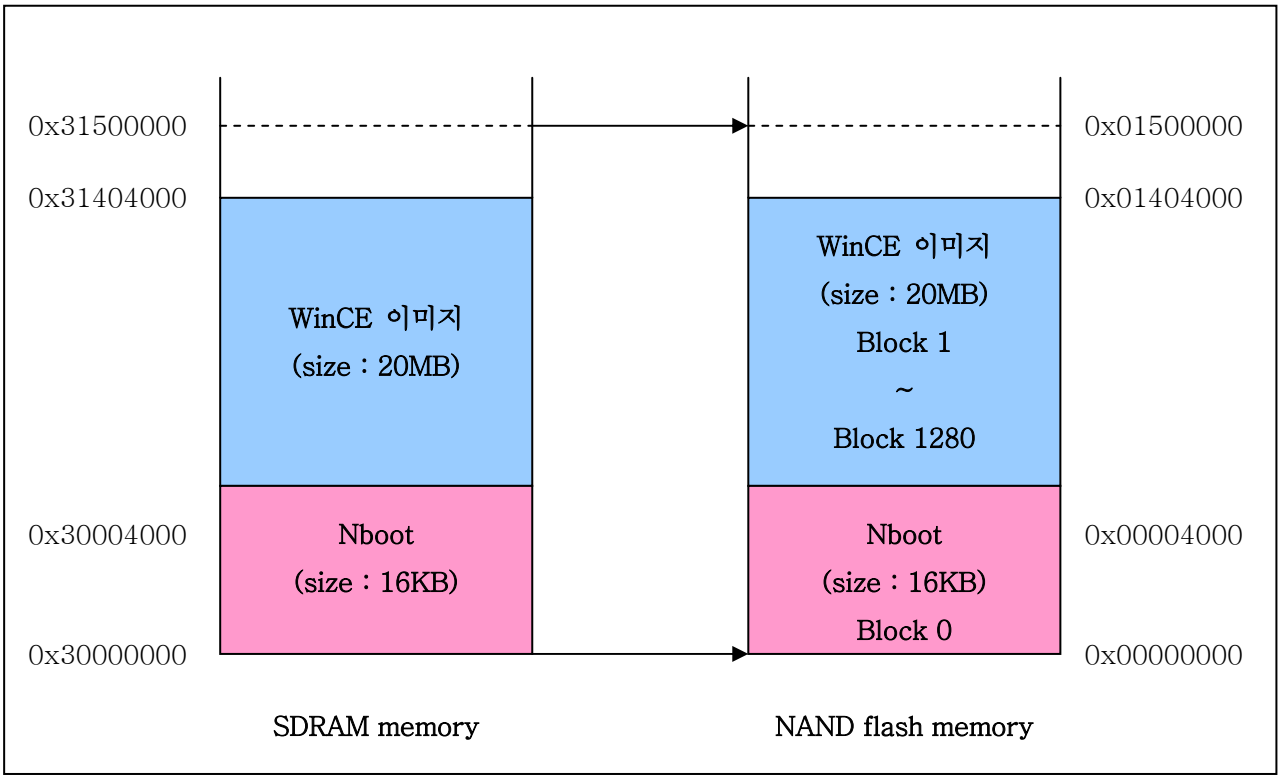
```
Image loading Start
Image loading End
Boot time=nTCNT*82uS. nTCNT=0x6784.
I/O Strength Min



CE Kernel for ARM (Thumb Enabled) Built on Jun  1 2004 at 16:38:37
ProcessorType=0920  Revision=0
sp_abt=ffff5000 sp_irq=ffff2800 sp_undef=ffffc800 OEMAddressTable = 8c201480

Windows CE Firmware Init
INFO: Initializing system interrupts...
INFO: Initializing system clock(s)...
```

```
INFO: Initializing driver globals area...
INFO: Initializing driver globals area...-> ok!
INFO: Initializing Display grid -> ok!
### SDMMC config current rGPGCON: ff95ffba
OEMInit Done...
Sp=ffffc7cc
OEMSetRealTime: Year: 2003, Month: 1, Day: 1, Hour: 12, Minute: 0, second: 0 rcnr=1h
OEMSetRealTime(register): Year: 3, Month: 1, Day: 1, Hour: 12, Minute: 0, second: 0 rcnr=1h
FMD::FMD_Init
.....
```

Image loading Start와 **Image loading End** 메시지를 통해서 NK.nb0 WinCE이미지를 메모리로 복사했음을 확인할 수 있다.

CE Kernel for ARM (Thumb Enabled) Built on Jun 1 2004 at 16:38:37 메시지를 통해 WinCE 이미지가 부팅되고 있음을 확인할 수 있다.

다음은 위 과정을 쉽게 이해할 수 있는 NAND flash와 메모리 간의 memory map이다.



[그림 7.26] NAND flash와 메모리 간의 memory map

8 iRTOS

8.1 iRTOS란…

*i*RTOS™는 "*i*nternet-speeding *RTOS*"라는 캐치프레이즈 하에서 2003년 충남대학교 컴퓨터공학과 이철훈교수(시스템소프트웨어) 연구실과 아이지시스템의 산학 공동개발을 통해 독자적으로 개발한 순수 국산 실시간 운영체제로서 1996년부터 수년간에 걸친 개발 과정을 통하여 안정적인 Kernel을 구현하였다.

iRTOS™는 작은 메모리 용량을 요구하는 임베디드 시스템의 특성을 고려하여 iRTOS™ 커널의 각 기능별로 모듈화 하였고 이는 응용프로그램의 요구사항에 맞춰 선택적으로 사용할 수 있게 되어 이미지 크기를 최적화 할 수 있다.

iRTOS™는 국내 제품인 멀티미디어용 SoC에 탑재됨으로써 안정성을 검증 받았다. 해외 유명 RTOS와 비교해 약 50%의 작은 이미지 사이즈(최대 21KByte)로 인해 제품 가격과 사이즈를 줄이는데 매우 효과적이다. 다양한 플랫폼(ARM계열, CalmRISC계열, MPC계열, Intel계열)을 지원하면서 안정성, 이식성의 효과를 제공한다.

iRTOS™커널의 일반적인 특징을 살펴보면 다음과 같다.

- **임베디드 시스템을 위한 우선순위 기반 멀티태스킹 선점형 실시간 운영체제**
- **구성요소**
  - Scheduling
    - 우선순위 기반 선점형 스케줄링
    - 동일한 우선순위에 대해 RR(Round-Robin) / FIFO(First-In First-Out) 지원
    - "Task Scheduling 방법" 특허
  - Dynamic Memory Management
    - Heap storage manager (Variable-Size)
    - Memory Pool(Fixed-Size)
  - ITC (Inter Task Communication)
    - 메시지 큐, 메시지 메일박스, 메시지 포트, 태스크 포트
  - 태스크 간 동기화
    - 세마포, 이벤트 플래그, 시그널
  - 효율적인 인터럽트 처리
    - LISR (Low-Level ISR)
    - HISR (High-Level ISR)
  - Software Timer
- Module화
  - *i*RTOS™는 각 기능별 모듈화로, 사용자의 요구사항에 맞춰 최적화 할 수 있습니다.
- Source Code
  - BSP(Board Support Package), 스케줄러, 예외 처리

: Assembly language

- 그 외 나머지

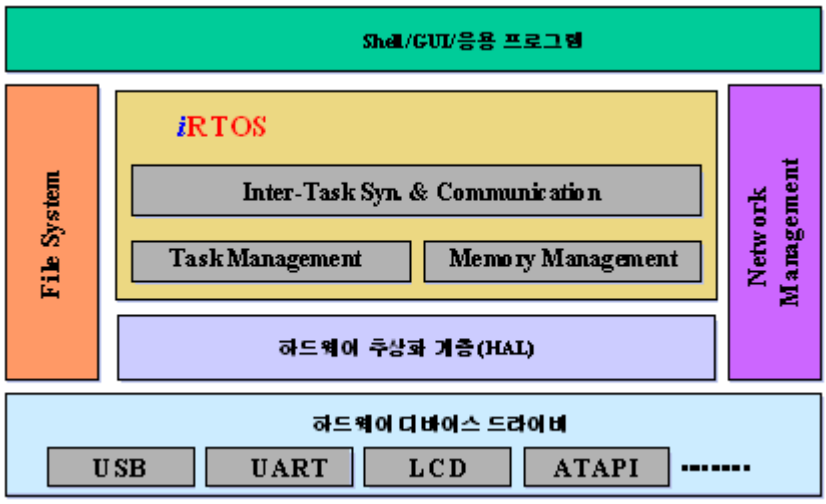
: ANSI C

• 지원 Processor

- ARM 계열 (S3C2400, S3C2800, S3C2440, S5H5002)
- CalmRISC 계열 (CalmRISC 16, CalmRISC 32)
- Motorola 계열 (MPC 750)
- Intel 계열 (80x86)

iRTOS™은 임베디드 시스템을 위한 멀티태스킹 환경의 우선순위 기반 선점형 실시간 운영체제이다. 태스크(Task)는 쓰레드(Thread) 기반으로 동작되는 구조로, 공통의 작업 영역을 자유롭게 접근할 수 있다. iRTOS™커널의 소스 코드는 플랫폼에 의존적인 하드웨어 초기화, 시스템 성능향상을 위한 스케줄러 부분이 어셈블리 언어로 작성되어 있고, 그 외 나머지 소스 코드는 ANSI C기반으로 작성되어 있다. 또한 iRTOS™은 태스크간의 통신을 위해서 메시지 큐, 메시지 메일박스, 메시지 포트, 태스크 포트와 같은 기능을 제공하고, 태스크간의 동기화를 위해 세마포, 이벤트 플래그, 시그널 등을 제공하고 있다.

[그림 8-1]은 iRTOS의 구성을 도식화 한 것이다.



[그림 8-1] iRTOS™의 전체구성

8.1.1 Kernel

iRTOS™ 커널의 기본 구성 요소는 다음과 같다.

❒ 태스크 관리(Task Management)
❒ 인터럽트 처리(Interrupt Handling)
❒ 동적 메모리 관리(Storage Allocation)
❒ 태스크간 통신(Inter Task Communication) 도구

❒ 태스크간 동기화(Inter Task Synchronization) 도구

❒ 타이머(Timer) 등

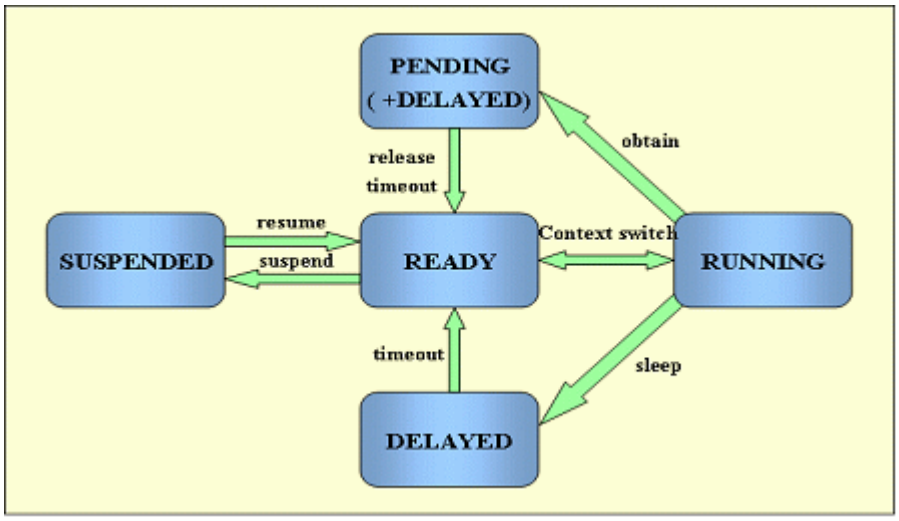
8.1.1.1 태스크 관리

임베디드 시스템의 응용프로그램은 여러 기능들이 동시에 수행되고, 논리적으로 나누어졌기 때문에 기능상 매우 밀접한 관계를 가진다. 이처럼 순차적으로 프로그램하기 어렵기 때문에 여러 개의 태스크가 동시에 수행될 수 있도록 하는 방식인 멀티태스킹이 도입되었다. iRTOS™ 커널은 이러한 기본적인 멀티태스킹 환경을 지원한다.

❍ iRTOS™에서의 태스크 상태

- SUSPEND : 태스크가 생성된 초기상태 또는 어떤 이벤트가 발생하기를 기다리는 상태.
- READY : 실행되기를 기다리는 상태.
- RUNNING : 태스크가 CPU를 점유하여 실행중인 상태.
- DELAYED : 일정 시간 만큼 스케줄링 대상에서 제외되는 상태.
- PENDING : 태스크의 수행 중 특정 자원을 얻기 위해서 대기하는 상태.
- PENDING+ DELAYED : 특정 자원을 획득하기 위해 대기할 때 타임아웃을 설정한 상태.

[그림 8-2]는 태스크의 상태 변화를 나타낸 것이다.



[그림 8-2] iRTOS™에서 태스크의 상태 변화

❍ iRTOS™의 스케줄러

iRTOS™는 우선순위 기반의 선점형 스케줄링을 하고, 동일한 우선순위에 대해서는 기본적으로 라운드 로빈을 스케줄링하며, 필요에 따라 FIFO(First-In First-Out) 스케줄링도 지원한다. iRTOS™에서 각각의 태스크의 우선순위는 0부터 255까지 256단계의 우선순위(Priority)를 가지며, 0이 가장 높은 우선순위를 가진다.

8.1.1.2 인터럽트 처리

인터럽트는 시스템의 내·외부에서 발생하는 이벤트로서, 인터럽트가 발생하면 태스크의 수행을 중지하고 인터럽트 서비스 루틴(ISR-Interrupt Service Routine)을 수행하게 된다. 인터럽트 처리는 실시간 운영체제의 특성상 빠른 응답시간을 요구한다. iRTOS™은 인터럽트에 대한 빠른 응답 시간을 융통성 있게 보장하기 위해 LISR(Low-Level ISR)과 HISR(High-Level ISR)로 구분하여 설계되었다.

[표 1]은 LISR과 HISR의 특징에 대하여 설명해 놓은 것이다.

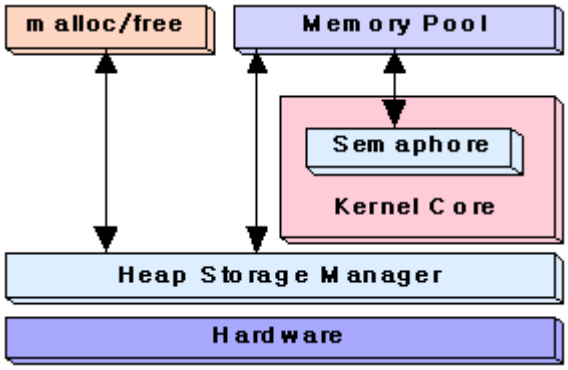
LISR(Low-level ISR)	HISR(High-level ISR)
• 빠른 응답시간이 필요한 인터럽트 • 기존의 인터럽트 처리와 동일 • 태스크 스택 사용 • 태스크의 상태를 변경시키는 시스템 콜 사용 불가	• LISR 에 비해 시간적 여유를 갖는 인터럽트 • 자신의 스택(HISR 스택) 사용 • 3 단계(0~2)의 우선순위 부여 • 태스크의 스케줄링에 앞서 HISR 스케줄링 • 태스크의 스케줄링 기법과 동일

[표 8-1] LISR 과 HISR 의 특징

8.1.1.3 동적 메모리 관리

iRTOS™는 동적 메모리 관리를 위해 두 단계의 메모리 관리 기법을 제공한다.
하위 단계는 가변 크기의 메모리를 할당 및 해제할 수 있는 힙 스토리지 매니저(Heap- Storage Manager)이고, 상위 단계는 고정 크기의 메모리를 할당 및 해제할 수 있는 메모리 풀(Memory Pool)이다.

[그림 8-3]은 동적 메모리 관리 체계를 보여준다.



[그림 8-3] iRTOS™의 동적 메모리 관리 체계

❍ 힙 스토리지 매니저(Heap Storage Manager)

힙 스토리지 매니저는 임의의 크기 메모리를 할당 받을 수 있어 유연성이 높다. 하지만 적당한 크기의 메모리 블록(Memory Block)을 찾는데 O(n)의 시간이 걸리면서 시간 결정성을 저해하고, 외부

단편화 현상이 나타난다.

❍ 메모리 풀(Memory Pool)

메모리 풀은 고정 크기의 메모리를 할당하는 것으로, 메모리의 사이즈가 동일한 곳이나 메모리의 할당이 다소 빈번하게 발생하는 응용프로그램에 적당하다. 메모리 할당 시 유연성이 떨어지지만, 할당되고 해제 된 영역을 그대로 사용할 수 있어 시간 결정성 및 외부 단편화 현상을 줄일 수 있다.

응용프로그램은 두 가지 관리 기법 모두를 병행하여 사용할 수 있으며, 또한 각각의 장단점을 고려하여 적당한 관리 기법을 선택할 수도 있다.

8.1.1.4 태스크간 통신 도구

iRTOS™는 여러 개의 독립적인 태스크가 동시에 실행될 수 있는 멀티 태스킹 환경을 제공한다. 멀티 태스킹 환경에서 태스크 간에 정보를 주고받기 위해서는 태스크간의 통신 도구가 필요하다.

❍ 메시지 메일박스(Message Mailbox)

iRTOS™에서 메시지 메일박스는 한번에 하나의 메시지만을 보낼 수 있는 구조로, 간단한 메시지를 주고받을 때 편리하게 사용할 수 있다.

❍ 메시지 큐(Message Queue)

메시지 큐는 메시지 메일박스와 달리 여러 개의 메시지를 전달할 때 사용된다. iRTOS™는 전달되는 메시지의 크기의 가변성에 따라 가변길이 메시지 큐와 고정길이 메시지 큐를 제공하며, 메시지 복사에 의한 전달 방법을 사용한다.

❍ 메시지 포트(Message Port)

iRTOS™는 포인터 자체를 전달하는 메시지 큐로 메시지 포트(Message Port)를 제공한다. 메시지 복사에 의한 전달은 포인터를 전달하는 방법에 비해 데이터 보안에 유리하지만, 복사에 따른 오버헤드가 뒤따른다. TCP/IP 등에서와 같이 태스크간 메시지 전송에 있어 메시지 복사에 의한 방법보다는 포인터 전달 방법이 훨씬 유리하다.

❍ 태스크 포트(Task Port)

iRTOS™는 메시지 큐, 메시지 메일박스 메시지 포트와는 달리 태스크와 태스크간의 일대일 통신을 위해 태스크 포트를 제공한다.

❍세마포(Semaphore)

iRTOS™에서 세마포는 공유자원에 접근할 때 상호 배타적인 접근을 가능하게 한다. 즉 다른 태스크의 간섭 없이 세마포를 획득한 태스크만이 공유자원을 사용한다.

8.1.1.5 태스크간 동기화

태스크간의 동기화란 태스크간의 순서를 조정하는 것을 말하며, iRTOS™에서 제공하는 태스크간의 동기화 방법에는 세마포, 이벤트 플래그, 시그널이 있다.

❍ 세마포(Semaphore)

iRTOS™의 세마포는 상호 배타적인 접근뿐 아니라 태스크간 동기화하는 방법으로도 사용할 수 있다.

❍ 이벤트 플래그(Event Flags)

이벤트 플래그는 세마포 개념을 확장한 개념으로 여러 개의 이벤트에 대한 동기화가 필요할 때 사용된다. iRTOS™의 이벤트 플래그는 32bit의 Flag 변수로 각각의 비트들은 하나의 개별적인 이벤트를 나타내며, 이벤트 처리에 있어서 OR와 AND연산이 가능하다.

❍ 시그널(Signal)

iRTOS™의 시그널은 특정 태스크에 소프트웨어 인터럽트를 거는 기법으로, 이벤트 플래그와 마찬가지로 32bit의 Signal 변수로 구성되어 있다. 하지만 시그널은 이벤트 플래그와는 달리 태스크마다 하나씩 존재하며, 시그널 처리를 위해서 스택(Stack)을 새로 구성하여 등록된 시그널 핸들러를 동작시킨다.

8.1.1.6 타이머

iRTOS™는 소프트웨어 타이머를 제공하여, 주기적으로 수행될 일들이 있을 경우 사용할 수 있다. 타이머에서 사용되는 시간의 단위는 틱(Tick)이고, 이 틱은 하드웨어 타이머 인터럽트가 발생하는 주기를 말한다.

8.1.1.7 동적 로더

로더는 프로그램을 한꺼번에 적재하는 것이 아니라 실행시 필요한 일부분만을 차례로 적재하는 방식이다.

8.2 iRTOS 부팅하기

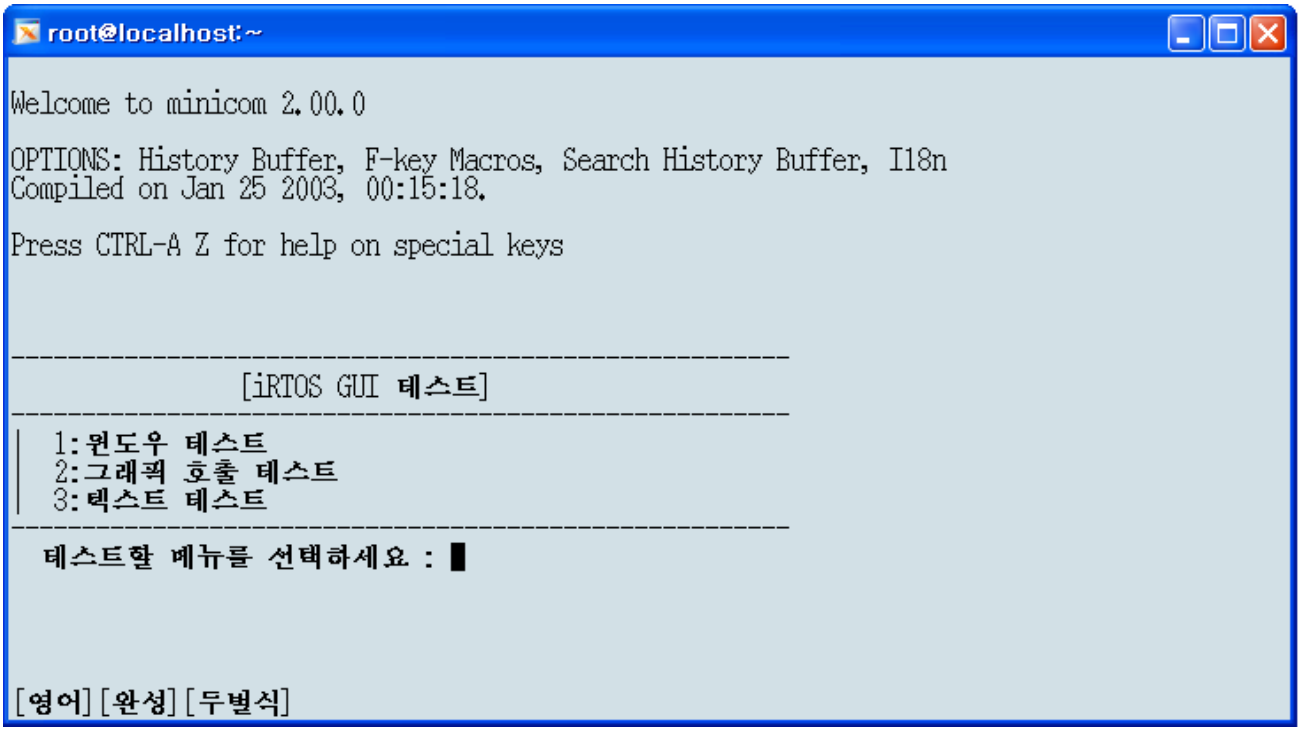
iRTOS는 이미지만을 제공하고 있다.

제공한 CD에 iRTOS 디렉토리에 **iRTOS_kor.bin**(메뉴가 한글버전)과 **iRTOS_eng.bin**(메뉴가 영문버전) 파일이 있다. **iRTOS_kor.bin** 파일을 AMD flash의 0번지에 write한다. Write하는 방법은 3.5.2 Boot flash에 이미지 쓰기 **또는** 5.2.2.4 NOR flash command를 참조한다.

시리얼 케이블을 MBA2440과 PC에 연결하고 DNW나 minicom이나 하이퍼 터미널을 콘솔로 사용한다. (2.1.3 하이퍼터미널 설정, 3.3.3 minicom 설정 참조)

(**주의 : DNW에서 사용시 글자가 깨지는 경우가 생기면 다른 콘솔 에뮬레이터를 이용할 것**)

AMD flash를 boot flash로 설정한 후, 전원을 켜면 [그림 8.4]과 같은 메뉴 화면을 볼 수 있다.



[그림 8.4]

iRTOS를 이용하여 LCD에 GUI를 display하기 위한 메뉴들이다.

직접 메뉴들을 선택하고 테스트 해 보기 바란다.