

# 1

## ABOUT SMDK2440 BOARD

### SYSTEM OVERVIEW

SMDK2440 (S3C2440 Development Kit) for S3C2440X is a platform that is suitable for code development of SAMSUNG's S3C2440X 16/32-bit RISC microcontroller (ARM920T) for hand-held devices and general applications.

The S3C2440X consists of 16-/32-bit RISC (ARM920T) CPU core, separate 16KB instruction and 16KB data cache, MMU to handle virtual memory management, LCD controller (STN & TFT), NAND flash boot loader, System Manager (chip select logic and SDRAM controller), 3-ch UART, 4-ch DMA, 4-ch Timers with PWM, I/O ports, RTC, 8-ch 10-bit ADC and touch screen interface, IIC-BUS interface, IIS-BUS interface, USB host, USB device, SD host & multimedia card interface, Camera Interface 2-ch SPI and PLL for clock generation.

The SMDK2440 consists of S3C2440X, boot EEPROM (flash ROM), SDRAM, LCD interface, two serial communication ports, configuration switches, JTAG interface and status LEDs.

### SMDK2440 OVERVIEW

The SMDK2440 (S3C2440 Development Kit) shows the basic system-based hardware design which uses the S3C2440X. It can evaluate the basic operations of the S3C2440X and develop codes for it as well.

When the S3C2440X is contained in the SMDK2440, you can use an in-circuit emulator (MULTI-ICE/OPENice32-A900).

This allows you to test and debug a system design at the processor level. In addition, the S3C2440X with MULTI-ICE/OPENice32-A900 capability can be debugged directly using the MULTI-ICE/OPENice32-A900 interface.

Figure 1-1 shows SMDK2440 function blocks.



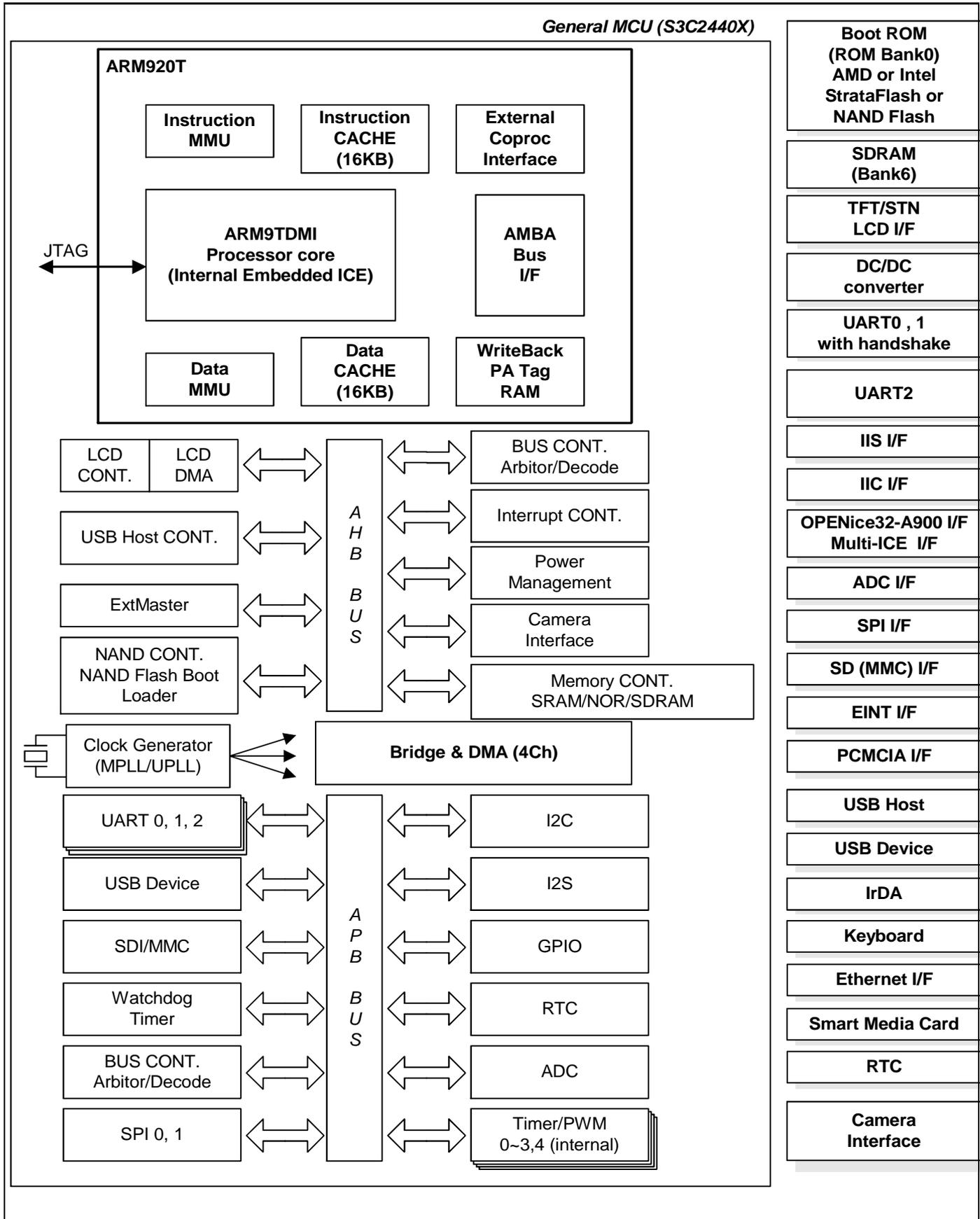


Figure 1-1. SMDK2440 Function Block Diagram

## FEATURES

- S3C2440X: 16/32-bit RISC microcontroller
- X-tal operation or oscillator
- Boot ROM: AMD 8M bit 1EA (support half-word size boot ROM)  
Intel StrataFlash 16M-byte x 2 ( word: 16M-byte x 2 EA): Unload (Option)  
SAMSUNG NAND flash 64M-byte 1EA (smart media card),  
SAMSUNG NAND flash 64M-byte 1EA (sop type)
- SDRAM: 64M-byte (32M-byte x 2)
- SRAM: 256K x 16 – Unload(Option)
- TFT/STN LCD and touch panel interface
- Three-channel UART (including IrDA)
- One Host Type USB port & Selectable Device and Host Type USH port
- SD host (MMC) interface
- Smart media card
- JTAG port (MULTI-ICE/OPENice32-A900 interface)
- RTC X-tal input logic
- IIC with KS24C080
- ADC interface
- SPI interface
- IIS interface (sound CODEC audio input/output)
- EINT interface
- GPIO Switch Interface
- IrDA interface
- Ethernet interface
- PCMCIA interface
- Extension connector 34P \* 3 EA
- LED display (debugging)
- CAMERA Interface



## CIRCUIT DESCRIPTION

The SMDK2440 is designed to test S3C2440X and develop software while hardware is being developed. Figure 1-3 shows SMDK2440's block diagram.

### POWER SUPPLY

SMDK2440 is operated by 1.2V for ARM core, 2.5V/3.3V for Memory and 3.3V for I/O pad and several peripherals. SMDK2440 is supplied by 9V/2A DC Adaptor Power.

The SMDK2440 has distributed power plane, with power going separately to the MCU and the main power plane. For this reason, power jumpers including J4-C~J15-C on the CPU board, J11-B, J12-B and J601-B on the base board are inserted.

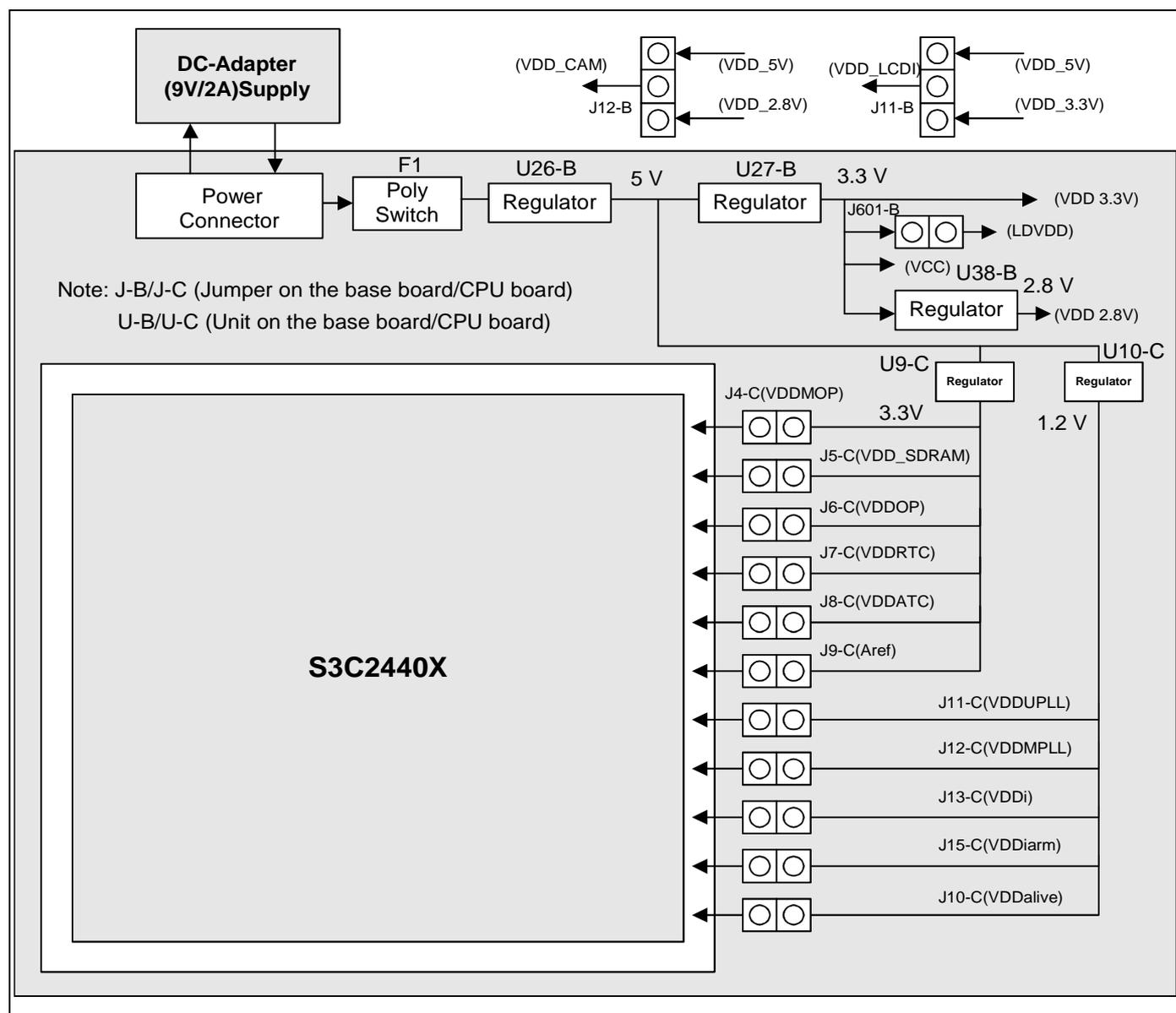


Figure 1-2. SMDK2440 Power Plane

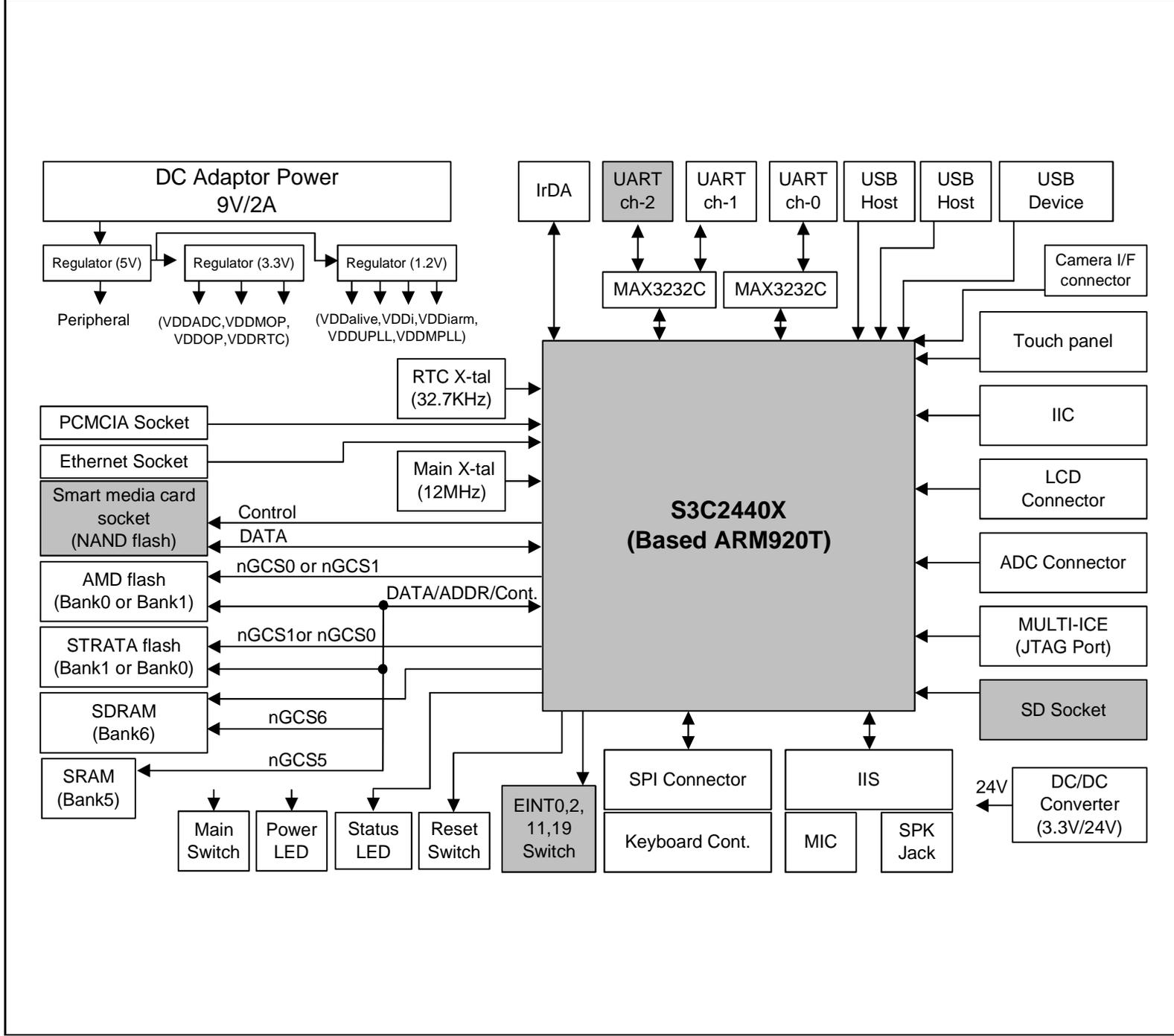


Figure 1-3. Detailed SMDK2440 Board Diagram

Preliminary product information describes products that are in development, for which full characterization data and associated errata are not yet available. Specifications and information herein are subject to change without notice.

## SMDK2440 SYSTEM CONFIGURATIONS

### CLOCK SOURCE

EXTCLK or X-TAL can be selected for the system clock of S3C2440X and USB by the suitable setting of OM values.

**Table 1-1. System Clock (MPLL) & USB Clock (UPLL)**

PIN FUNCTIONS	OM[3:2]		DESCRIPTIONS
Clock source selection	0	0	MPLL: XTAL, UPLL: XTAL
	0	1	MPLL: XTAL UPLL: EXTCLK
	1	0	MPLL: EXTCLK, UPLL: XTAL
	1	1	MPLL: EXTCLK, UPLL: EXTCLK

### RTC Clock

32.768KHz, X-tal is available in SMDK2440 as the RTC clock source.

#### NOTES:

- Although the MPLL starts just after a reset, the MPLL output (Mpll) is not used as the system clock until the software writes valid settings to the MPLLCON register. Before this valid setting, the clock from external crystal or EXTCLK source will be used as the system clock directly. Even if the user wants to maintain the default value of the MPLLCON register, the user should write the same value into the MPLLCON register.
- OM[3:2] is used to determine test mode when OM[1:0] is 11.

### RESET LOGIC

The nRESET (system reset signal) must be held to low level at least 4 CLKs to recognize the reset signal and it takes 128 CLKs between the nRESET and internal nRESET. nRESET and nTRST (JTAG reset signal) are connected through jumper J35-C on the CPU board.

## BOOT ROM (BANK0)

The data bus width of BANK0 can be configured in byte, half-word or word in S3C2440X.

In the case of SMDK2440, half-word data bus width (AMD flash memory), half-word or word data bus width (STRATA flash memory), and byte or half-word data bus width (Samsung NAND flash memory) access can be selected by the suitable jumper setting.

AMD flash or STRATA flash memory can be selected by using jumper (J3-B & J4-B) option for boot ROM. In the SMDK2440, the data bus width of AMD flash memory is fixed by half-word data width (16-bit) and STRATA flash memory can use word (32-bit).

But AMD flash and STRATA flash cannot be selected for BANK0 or BANK1 at the same time. Data bus width of BANK0 should be set by memory type of BANK0. It is set by OM[1:0](J2-B & J1-B).

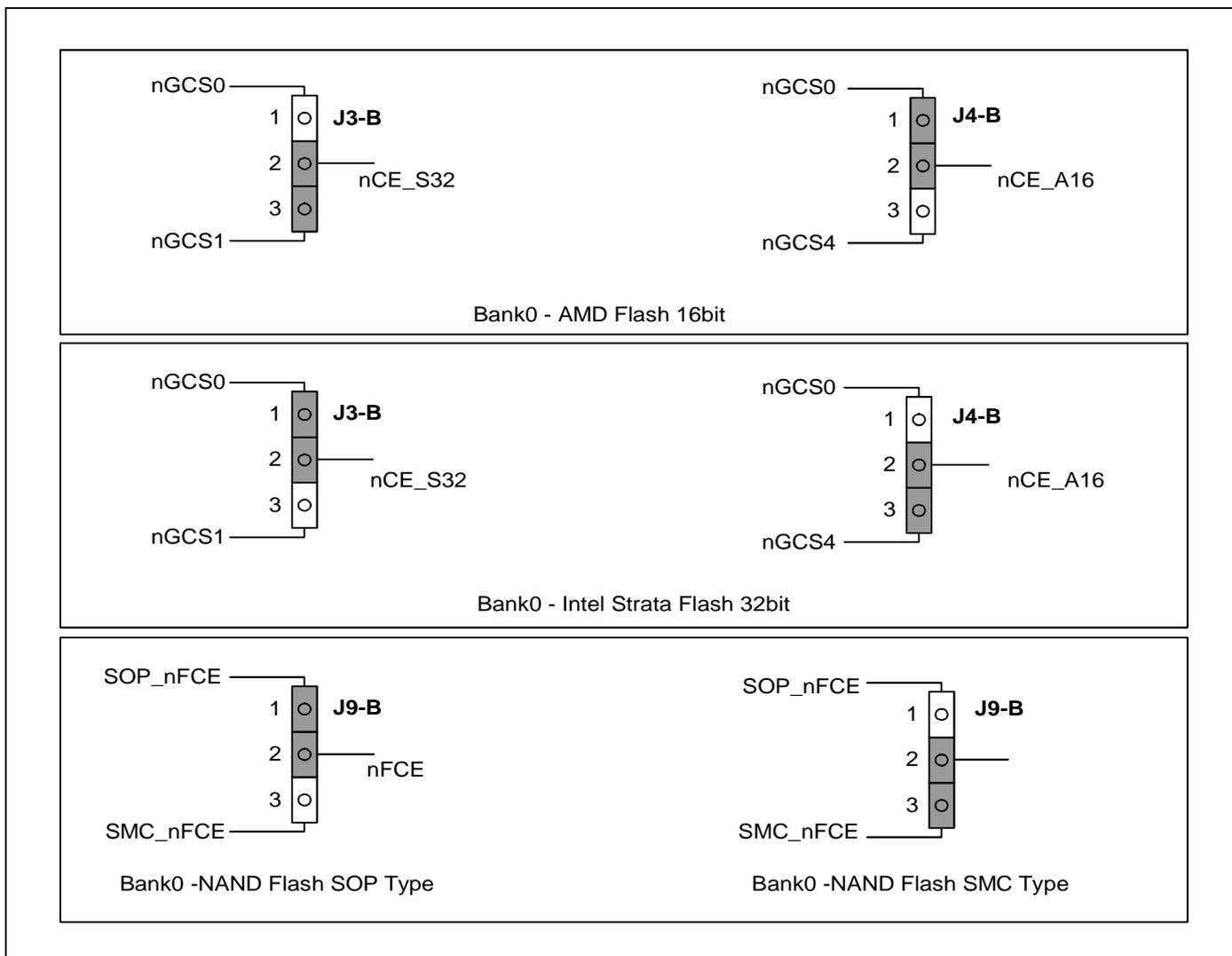


Figure 1-4. SMDK2440 Board Memory Configurations

Table 1-2. Memory Type and Data Bus Width

Pin Functions	J3-B		J4-B		Descriptions
	1-2	2-3	1-2	2-3	
BANK0/1/4 memory type selection and data bus width configuration	Open	Short	Short	Open	AMD flash memory : BANK0 STRATA flash memory : BANK1 Data bus width of BANK0 : Half-word Data bus width of BANK1 : Word
	Short	Open	Open	Short	AMD flash memory : BANK4 STRATA flash memory : BANK0 Data bus width of BANK0 : Word Data bus width of BANK4 : Half-word

Table 1-3. Boot Memory Type and Bus width configuration OM[1:0]

Pin Functions	J1-B [OM0]	J2-B[OM1]	Descriptions
Boot memory type and	2-3(L)	2-3(L)	NAND Boot
bus width configuration	2-3(L)	1-2(H)	Word (32-bit)
	1-2(H)	2-3(L)	Half Word (16-bit)
	1-2(H)	1-2(H)	Test Mode

## NAND FLASH CONFIGURATION

Table 1-4. NAND Flash Type Selection

OM[1:0] = NAND Boot Setting ( L, L )			
J5-B( NCON 0 )		2-3(L)	1-2(H)
		Normal NAND	Advanced NAND
J6-B( PAGE )	2-3(L)	256	1024
	1-2(H)	512	2048
J7-B( ADDR )	2-3(L)	3 Cycle	4 Cycle
	1-2(H)	4 Cycle	5 Cycle
J8-B ( WIDTH )	2-3(L)	8-bit Bus Width	
	1-2(H)	16-bit Bus Width	
J9-B ( NAND Select)	2-3(L)	Use SMC NAND	
	1-2(H)	Use SOP NAND	

### NOTE:

- Jumpers on the base board: J1-B, J2-B, J3-B, ...

## GENERAL I/O PORTS

The S3C2440X's general I/O ports are used for SMDK2440 key interrupt input, normal input and LED status display. The function of control switch and the status of LED can be defined by user software.

**Table 1-5. General I/O Configurations on SMDK2440**

General I/O Port Number	I/O Type	Descriptions
GPF[7:4]	Output	LED display
GPF0, GPF2, GPG3 & GPG11	Input	Key input pad (external interrupt input pins). (EINT0, 2, 11 & 19)

## U4 (EPM7032) XDMA CHANNEL SELECTION

**Table 1-6. U4-C XDMA Channel Selection**

Pin Functions	J1-C	J2-C	Descriptions
XDMA channel selection	(1-2)	(1-2)	nXDREQ0, nXDACK0
	(2-3)	(2-3)	nXDREQ1, nXDACK1

### NOTE:

- Jumpers on the CPU board: J1-C, J2-C, J3-C, ...



## LCD INTERFACE

TFT/STN LCD controllers are equipped in the S3C2440X. TFT/STN LCD, touch panel and LCD backlight driver are supported in the SMDK2440.

### NOTES:

It is supported 2-type SEC TFT LCD panel(SAMSUNG 3.5" Portrait/256 Color/Reflective a TFT LCD)

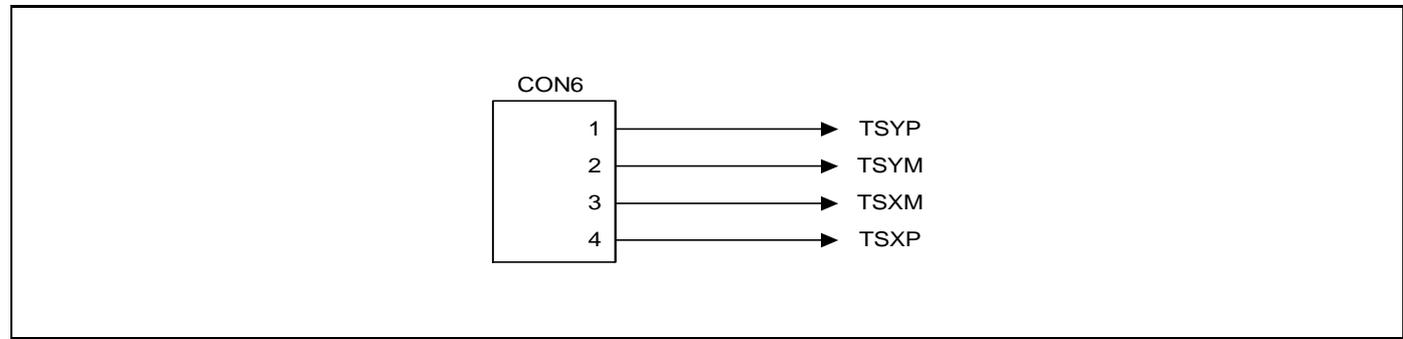
LTS350Q1-PD1 Panel with touch panel and front light unit

LTS350Q1-PD2 Panel only

LTS350Q1-PE1 Panel with touch panel and front light unit

LTS350Q1-PE2 Panel only

## TOUCH SCREEN



**Figure 1-5. Touch Panel Film Connector on SMDK2440**

## SPI Connector

Figure 1-6 shows the way SMDK2440 provides SPI (CON15) signals.

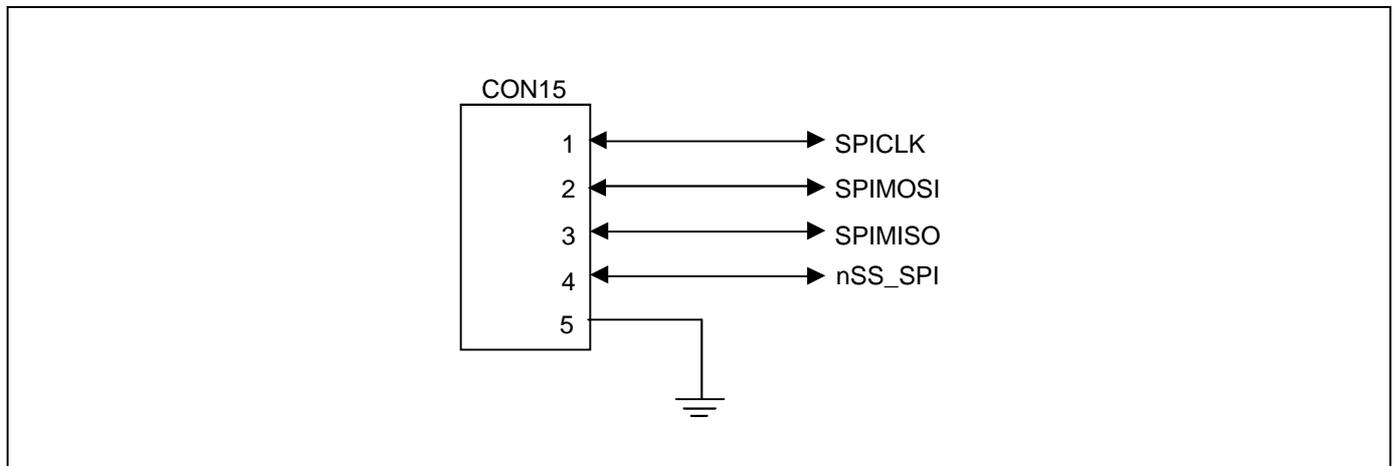


Figure 1-6. SPI Connector on SMDK2440

## A/D CONVERTER INTERFACE

The S3C2440X has Analog to Digital Converter (ADC). The ADC has 8-ch analog input signals. The SMDK2440 provides the ADC (CON8) signals as follows:

Table 1-7. ADC Interface on SMDK2440

# of pin	Descriptions						
1	AIN0	4	AIN3	7	TSXM	10	GND
2	AIN1	5	TSYM	8	TSXP		
3	AIN2	6	TSYP	9	EINT20		

## SD HOST (MMC) INTERFACE

SD(MMC) is provided by the S3C2440X and SD card socket (CON13) is supported in the SMDK2440

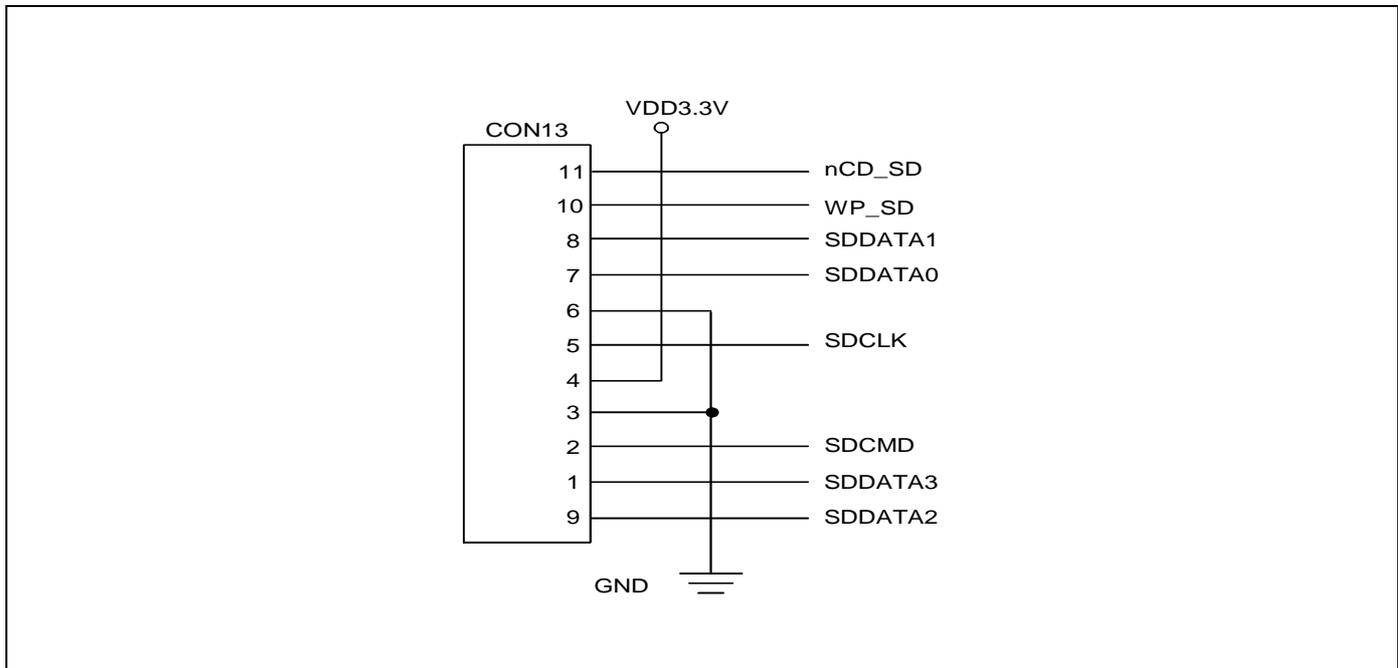


Figure 1-7. SD Card Socket on SMDK2440

## IIC INTERFACE

Serial EEPROM S524C80D80 (KS24C080) access function is provided by SMDK2440 and there is also IIC interface between S3C2440X and camera module through U37-B(CBTD3306) buffer.

## USB INTERFACE

Dual USB Connector(CON3) for Two USB port A-Type and one USB port B-type(CON5) are supported by the SMDK2440.

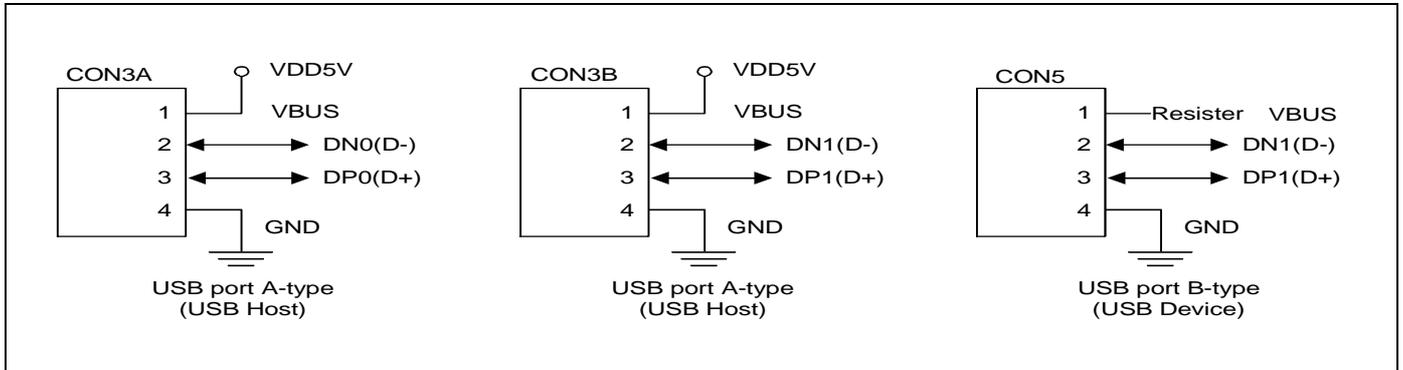


Figure 1-8. USB Ports on SMDK2440

You Can be select the USB port 1 (DN1, DP1) by Jumper (J14-C, J16-C)

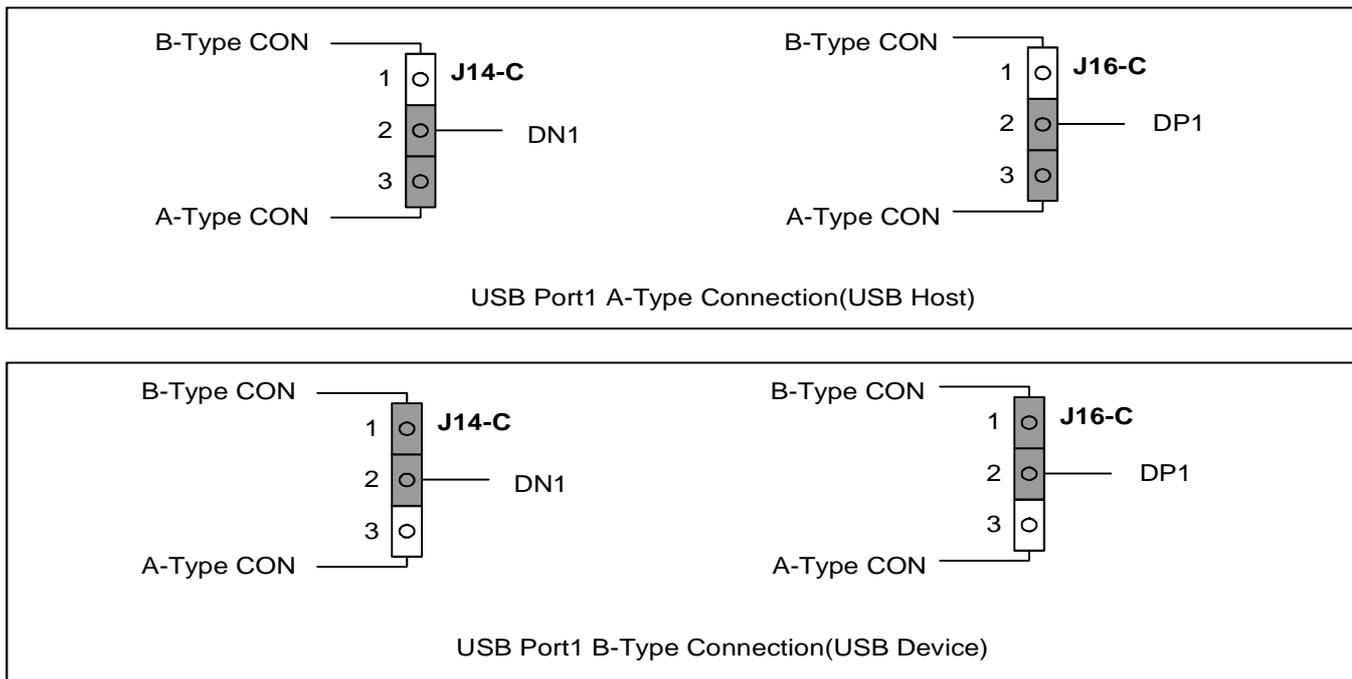


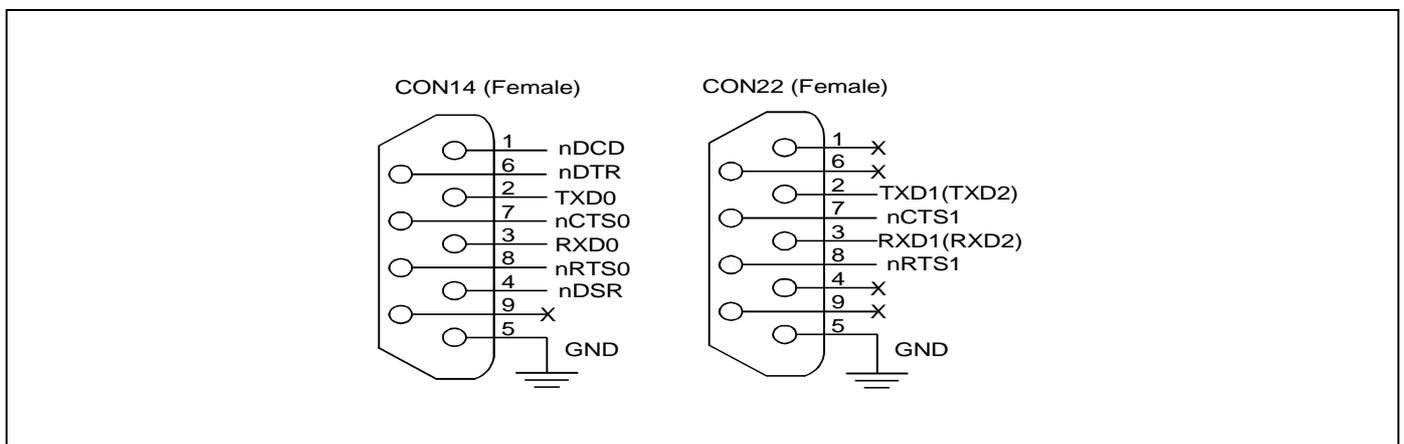
Figure 1-9. USB Port1 Selection

## UART INTERFACE

The S3C2440X UART unit provides three independent asynchronous serial I/O (SIO) ports including IrDA. In SMDK2440 board, a user can change the ports connected to connectors by setting related jumpers.

**Table 1-8. UART Configurations**

Pin Functions	J16-B, J18-B	J17-B, J19-B	Descriptions
UART configurations	(2-3)	(1-2)	CON14: UART0, CON22: UART1
	(1-2)	(2-3)	CON14: UART0, CON22: UART2



**Figure 1-10. UART Ports on SMDK2440**

## IrDA INTERFACE

IrDA is supported by SMDK2440 and J17-B and J19-B should be set to UART2 (RXD2 and TXD2) for IrDA.



Figure 1-11. SMDK2440 Board IrDA Configurations

Table 1-9. IrDA Configurations

Pin Functions	J17-B, J19-B	Descriptions
UART2	(2-3)	Set UART mode
IrDA	(1-2)	Set IrDA mode

## EXTENSION CONNECTOR INTERFACE

Table 1-10. Extension Connector (CON10, CON11 &amp; CON12) on SMDK2440

# of pin	Descriptions						
1	GND	10	DATA8	19	DATA17	28	DATA26
2	DATA0	11	DATA9	20	DATA18	29	DATA27
3	DATA1	12	DATA10	21	DATA19	30	DATA28
4	DATA2	13	DATA11	22	DATA20	31	DATA29
5	DATA3	14	DATA12	23	DATA21	32	DATA30
6	DATA4	15	DATA13	24	DATA22	33	DATA31
7	DATA5	16	DATA14	25	DATA23	34	–
8	DATA6	17	DATA15	26	DATA24	–	–
9	DATA7	18	DATA16	27	DATA25	–	–

# of pin	Descriptions						
1	GND	10	A8	19	A17	28	nWBE0
2	A0	11	A9	20	A18	29	nWBE1
3	A1	12	A10	21	A19	30	nWBE2
4	A2	13	A11	22	A20	31	nWBE3
5	A3	14	A12	23	A21	32	nWE
6	A4	15	A13	24	A22	33	nOE
7	A5	16	A14	25	A23	34	–
8	A6	17	A15	26	A24	–	–
9	A7	18	A16	27	nWAIT	–	–

# of pin	Descriptions						
1	nGCS2	10	GPG7	19	nXDACK1	28	GND
2	nGCS1	11	GPG2	20	nXDREQ0	29	GND
3	nGCS4	12	GPG8	21	GPG5	30	nRESET
4	nGCS3	13	GPG3	22	nXDREQ1	31	VDD5V
5	nGCS7	14	GPG9	23	VDD1.8V	32	VDD3.3V
6	nGCS5	15	GPG4	24	GPG12	33	CLKOUT1
7	GPG0	16	GPG10	25	GND	34	GND
8	GPG6	17	nXDACK0	26	CLKOUT0	–	–
9	GPG1	18	GPG11	27	VDD3.3V	–	–





## NOTES



# 2 TOOLKIT AND DEBUGGING

## SMDK2440 ENVIRONMENT SETUP

The evaluation environments for the SMDK2440 are shown in Figure 2-1. The serial port (UART1) on the SMDK2440 has to be connected to COM port of the host PC. This can be used as a console for monitoring and debugging the SMDK2440. And the USB device on the SMDK2440 should be connected to the USB host of the host PC for downloading test images.

If you have an emulator such as MULTI-ICE and OPENice32-A900, you can use JTAG port on the SMDK2440 to interface the emulator.

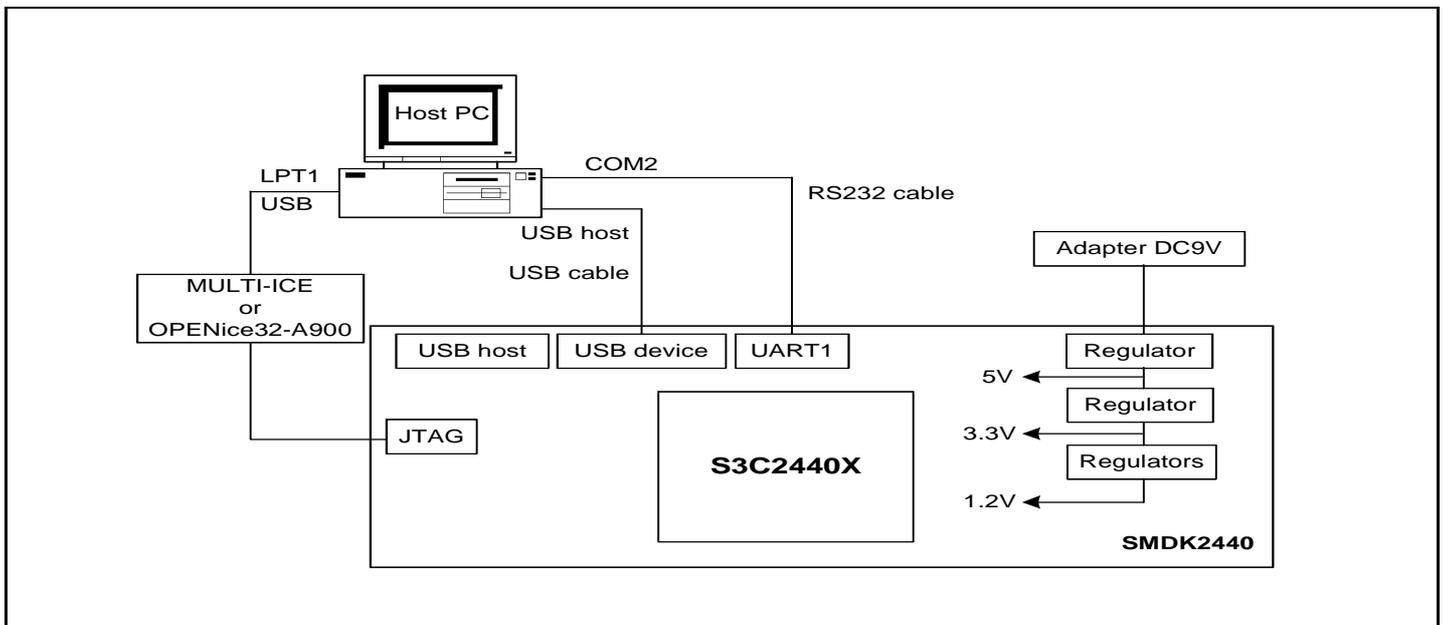


Figure 2-1. Setup Environment for SMDK2440 Board

## RS232C CABLE CONNECTION

The serial cable is made as in Figure 2-2. The pins numbered only 2, 3, and 5 are used; make sure to check the cable's connections to prevent other pins from being used.

The UART1(CON22) and PC COM1 or COM2 port has to be connected through this cable connection.

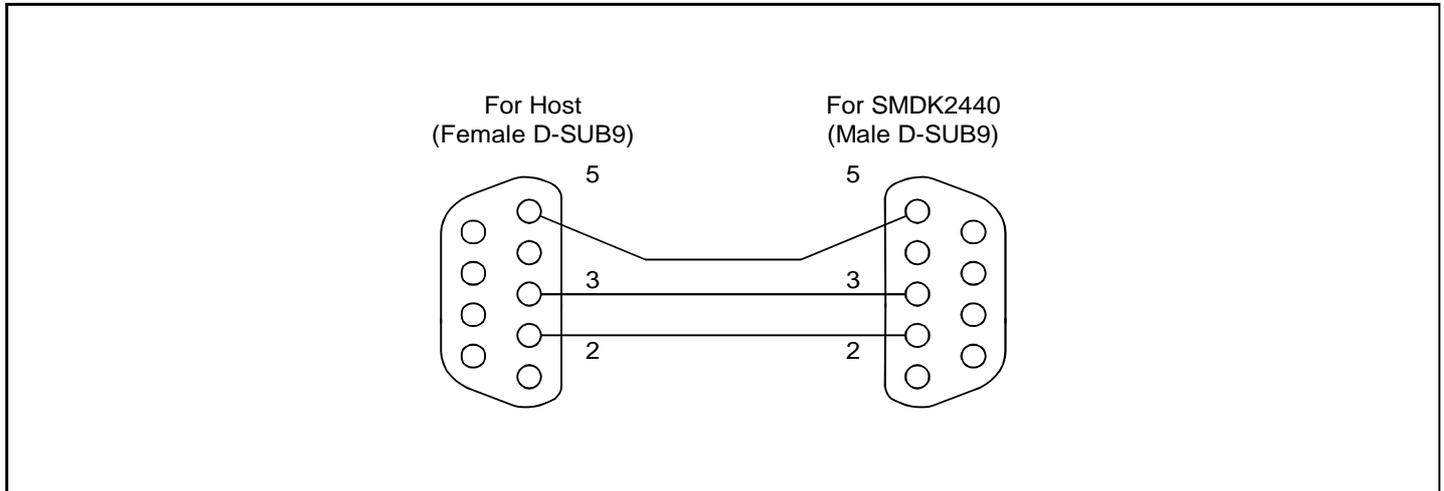


Figure 2-2. Serial Cable Connections for SMDK2440 Board

## USB DOWNLOADER INSTALLATION (ON WINDOWS 98, ME, 2000 OR NT)

To install the USB downloader, follow the steps:

1. Program the u2440mon.bin into the flash memory of SMDK2440X board.
2. Configure Boot Jumper Setting (J1-B ~ J9-B).
3. Turn on the SMDK2440.
4. If you installed the device driver for SMDK2400X/SMDK2440X before, overwrite new 'secbulk.sys' at C:\WINDOWS\SYSTEM32\DRIVERS. In this case, the step 6 will be skipped.
5. Connect the SMDK2440X board with the PC (See Figure 2-3).
6. When the USB device driver installation window appears, install the USB device driver (secbulk.inf).  
Note: 'secbulk.inf' and 'secbulk.sys' should be in the same directory (See Figure 2-4).
7. Run 'dnw.exe'.
8. Turn the SMDK2440 off and then on.
9. The message ([USB:OK]) in the window title bar indicates that the installation is successfully completed.

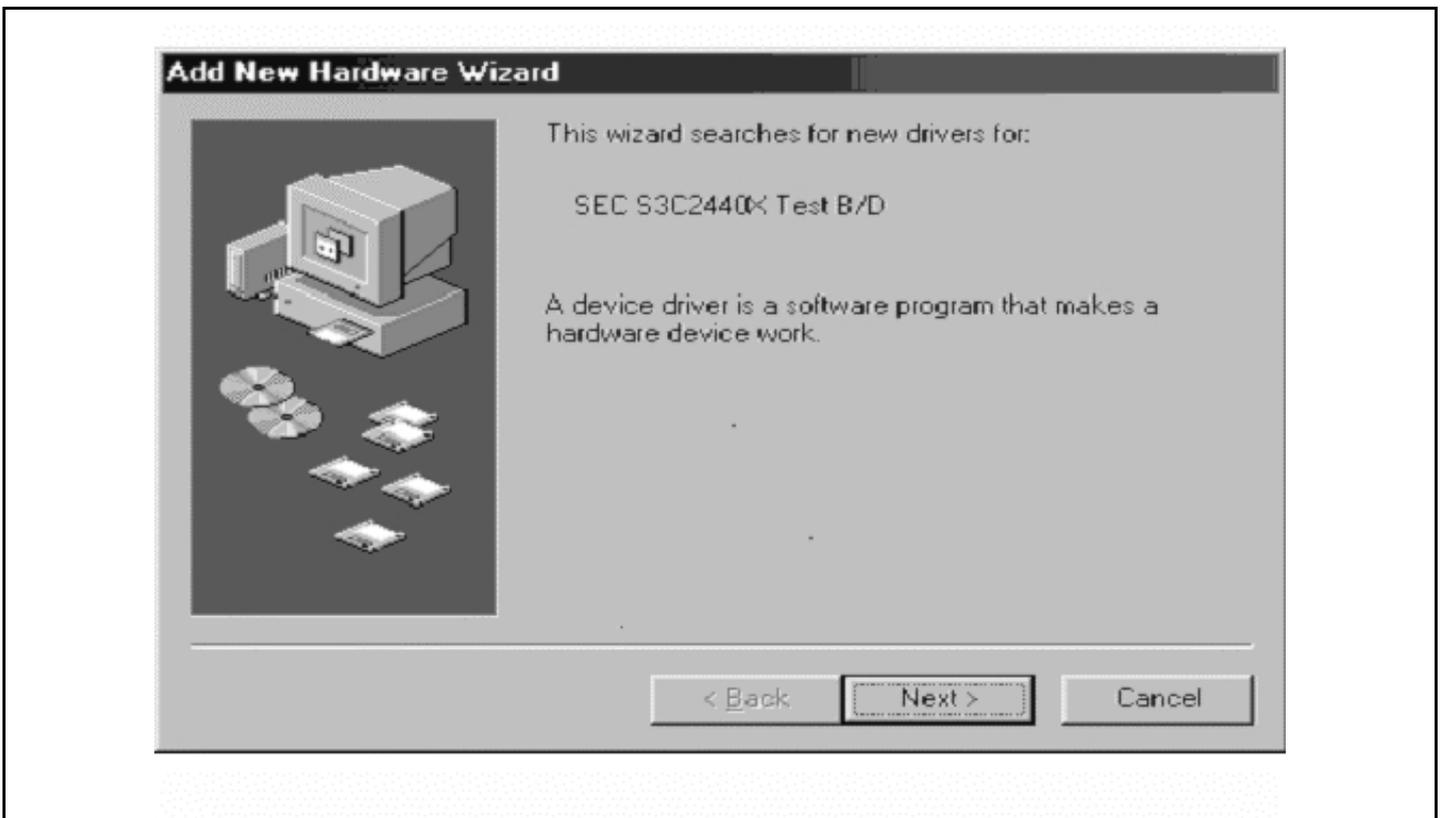


Figure 2-3. Add New Hardware Wizard (Window98)

### NOTES:

1. If you have installed the device driver before, replace the old 'secbulk.sys' in C:\WINDOWS\SYSTEM32\DRIVERS with the new 'secbulk.sys'.
2. The maximum speed of the 'secbulk.sys' with SMDK2440 will be about 980KB/S.
3. 'dnw.exe': PC USB/serial downloader program.
4. 'secbulk.inf' and 'secbulk.sys': PC USB driver.
5. 'u2440mon.bin': S3C2440X USB downloader firmware.

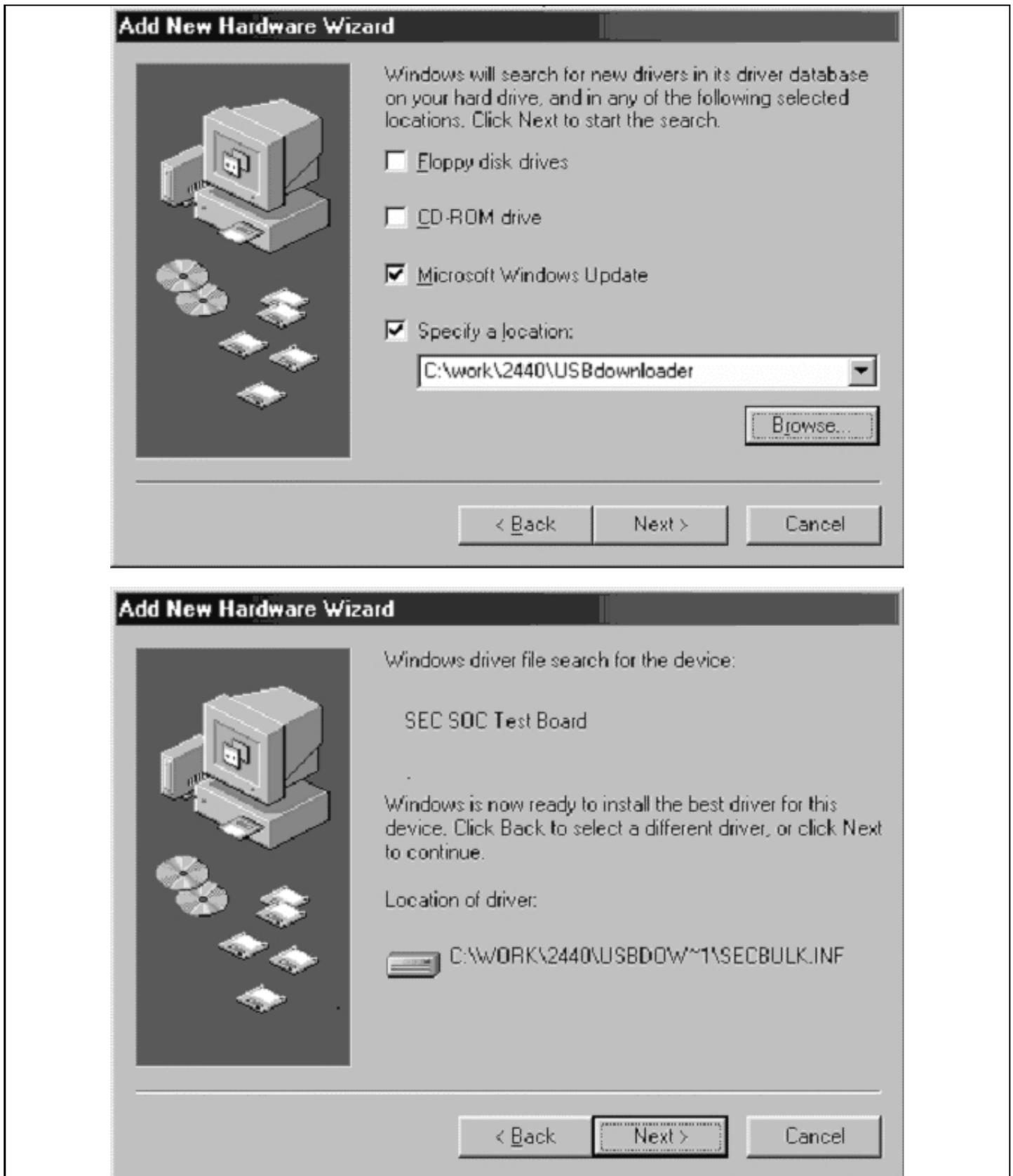


Figure 2-4. USB Device Driver Installation



**Figure 2-4. USB Device Driver Installation (Continued)**

### CONFIGURING DNW

To configure the DNW, which works as USB and serial download utility, follow the steps:

1. Run the DNW.
2. Select Options from the Configuration menu (See Figure 2-5).  
Configuration → Options
3. Select Baud rate for serial communication.  
Serial communication properties of the DNW are as follows:  
Data bits:8-bit / Stop bits:1 / No flow control
4. Select a COM port of the host PC to communicate with the SMDK2440.
5. Set USB download address.
6. Click the OK button.

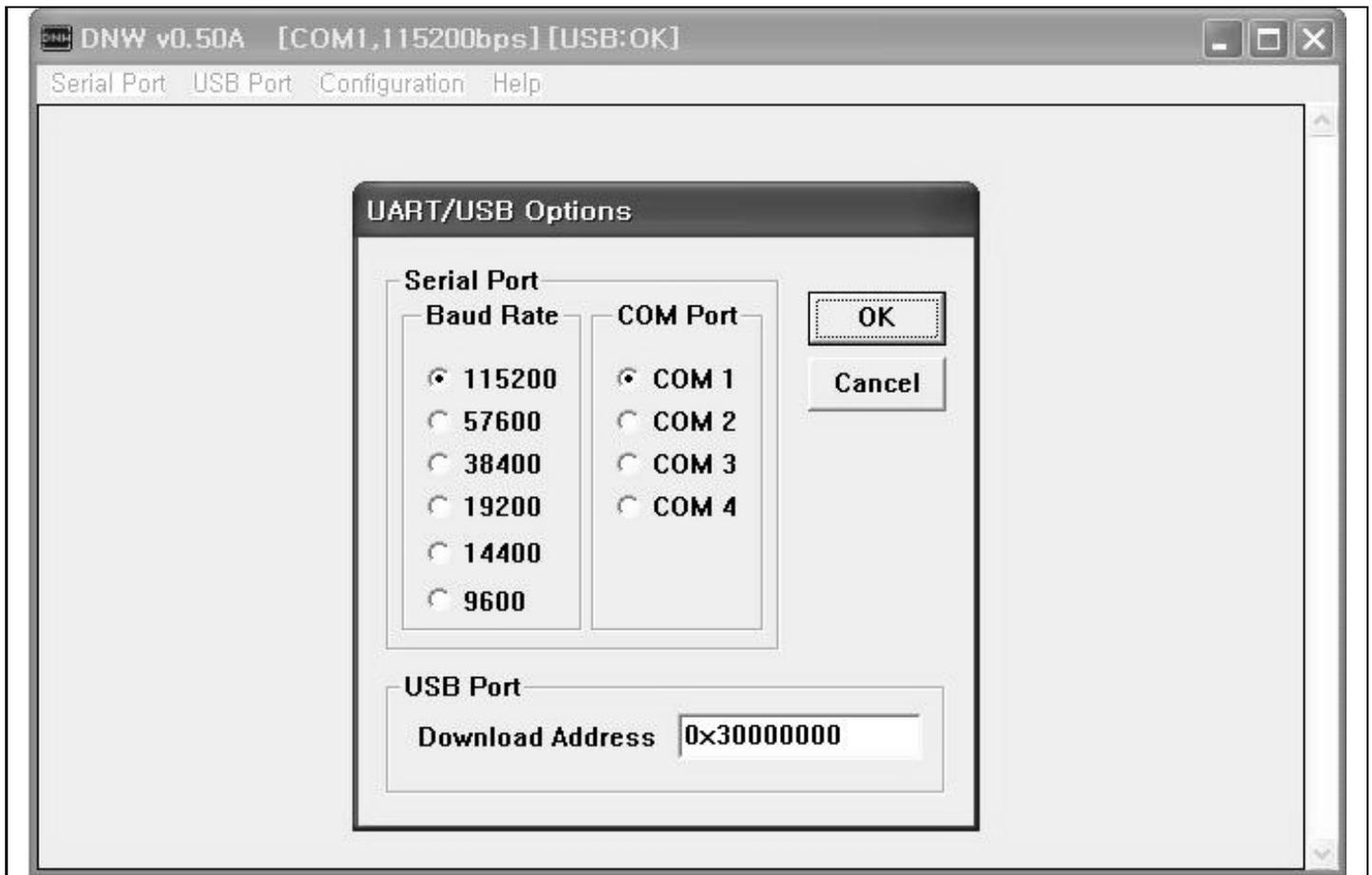


Figure 2-5. Setting UART/USB Options

## CONNECT HOST PC AND SMDK2440 WITH DNW

After setting UART/USB options, users can activate UART and USB communication.

1. Select Connect from the Serial Port menu.  
Serial Port → Connect
2. Power on the SMDK2440 (See Figure 2-6).

```

DNW v0.50A [COM1,115200bps] [USB:OK]
Serial Port  USB Port  Configuration  Help
DIUN_UPLL0
MPLL0a1 [M:58h,P:1h,S:1h]
CLKDIUN:3h

+-----+
| S3C2440X USB Downloader ver R0.01 2003 May |
+-----+
FCLK=192MHz,DMA mode
USB: IN_ENDPOINT:1 OUT_ENDPOINT:3
FORMAT: <ADDR(DATA):4>+<SIZE(n+10):4>+<DATA:n>+<CS:2>
NOTE: 1. Power off/on or press the reset button for 1 sec
       in order to get a valid USB device address.
       2. For additional menu, Press any key.

USB host is connected. Waiting a download.
  
```

Figure 2-6. Power On Screen

## INSTALL ARM TOOLKIT

First of all, install ARM toolkit 2.51, ADS (ARM Developer Suite) 1.0.1 or ADS1.1.

If you installed ARM toolkit 2.11a, then Makefile has to be changed a little. The toolkit 2.11a cannot support fromelf.exe utility. We recommend ARM Developer Suite 1.0.1, which is used in our development environment. We also recommend ADS 1.1.

The DOS environment variable has to be changed as follows after the installation of ARM toolkit 2.51.

```
SET ARMLIB=C:\ARM251\LIB\embedded
SET ARMINC=C:\ARM251\INCLUDE
```

## HOW TO BUILD EXECUTABLE IMAGE FILE

Executable image file can be built by using the ARM Project manager or makefile. First, you have to build ELF format image (\*.ELF or \*.AXF). An ELF format image can be used for the ARM debugger directly. The binary file (.bin file) can be extracted from ELF format image.

First of all, you have to download S3C2440X evaluation source code and any other utilities from our web site ([www.samsungsemi.com](http://www.samsungsemi.com)). They are helpful for you to understand the development environments of S3C2440X in an easier way. The distributed evaluation source code consists of following directories.

Directory	Description
BMP	Graphic header file converted from BMP file
obj	Object files
err	Error files

## BUILDING 2440TEST.AXF (OR 2440TEST.ELF)

To build the sample source code, 2440TEST, run Makefile using the following commands.

```
cd 2440Test
armmake -a
```

or

```
cd 2440Test
make clean
make
```

After the procedure, 2440TEST.AXF (or 2440TEST.ELF) and 2440TEST.BIN image files will be seen in 2440TEST directory. The 2440TEST.AXF (or 2440TEST.ELF) file is used for ARM debugger.

The 2440TEST.BIN file is used for downloading through USB.

**EXECUTING 2440TEST WITHOUT ARM MULTI-ICE OR OPENICE32-A900**

First, U2440MON has to operate on ROM. U2440MON will be ready to receive 2440TEST.BIN. U2440MON will launch 2440TEST.BIN after receiving 2440TEST.BIN.

To download 2440TEST.BIN through USB, after connecting the host PC and the SMDK2440 with the DNW follow the steps below:

1. Select Transmit from the USB Port menu.  
USB Port → Transmit
2. Select 2440TEST.BIN (See Figure 2-7).



```

DNW v0.50A [COM1,115200bps] [USB:OK]
Serial Port USB Port Configuration Help

+-----+
| S3C2440X USB Downloader ver R0.01 2003 May |
+-----+

FCLK=192MHz,DMA mode
USB: IN_ENDPOINT:1 OUT_ENDPOINT:3
FORMAT: <ADDR(DATA):4>+<SIZE(n+10):4>+<DATA:n>+<CS:2>
NOTE: 1. Power off/on or press the reset button for 1 sec
       in order to get a valid USB device address.
       2. For additional menu, Press any key.

USB host is not connected yet.
USB host is connected. Waiting a download.

Now, Downloading [ADDRESS:30000000h,TOTAL:390118]
RECEIVED FILE SIZE: 390118(1016.1KB/S,0.4S)
Now, Checksum calculation
Download O.K.

[SMDK2440 Board Test Program Ver 0.0]

[Fclk:Hclk:Pclk]=[203.2:101.6:50.8]Mhz
[Uclk=48.0Mhz]

 0:User Test      1:Manual Reg Set  2:PCMCIA test    3:Stepping stone
 4:nWAIT test    5:Nand test      6:Program Flash  7:DMA test
 8:FIQ test      9:EINT test     10:Cpu speed test 11:Power/Clk test
12:Lcd test     13:Camera test  14:SPI Test      15:IIC Test
16:RTC Test     17:IrDA Test    18:SD test       19:ADC test
20:ADC TS test  21:Timer test

Select the function to test :

```

Figure 2-7. 2440TEST Execution After its Downloading through USB

## HOW TO USE ARM DEBUGGER WITH ARM MULTI-ICE

If you have built 2440TEST program without any error, you can find 2440TEST.AXF in 2440TEST directory. The generated image file will be downloaded to SDRAM memory on the SMDK2440 by ARM debugger through MDS like a MULTI-ICE. Next, you can start to debug the downloaded image using the ADW (ARM Debugger for Windows).

## PREPARING AND CONFIGURING ARM MULTI-ICE

1. MULTI-ICE will be connected through JTAG port on the board. Connect all cables properly following its manual.
2. Start the ARM MULTI-ICE Server (Double click the MULTI-ICE Server icon).
3. Select Load Configuration from File menu and load 2440.CFG (See Figure 2-8).  
File → Load Configuration
4. Contents of 2440.CFG are as follows:  

```
[TITLE]
S3C2440/S3C2440 TAP Configuration

[TAP 0]
ARM920T

[Timing]
Adaptive=OFF
```
5. Select Start-up Options from Settings menu.  
Settings → Start-up Options
6. Start-up Options dialog box is displayed (See Figure 2-9).  
Now you can select Load Configuration from Start-up Configuration and browse 2440.CFG
7. Start ARM Debugger using ARM debugger icon  
Also, you can start the debugger at DOS command window by typing `adw 2440TEST.AXF`.  
If you use ARM MULTI-ICE for the first time, you have to add Multi-ICE.DLL to ADW.
8. When ARM Debugger is started, it will load the image code to the Armulator (The Armulator is software emulator for ARM920T CPU).

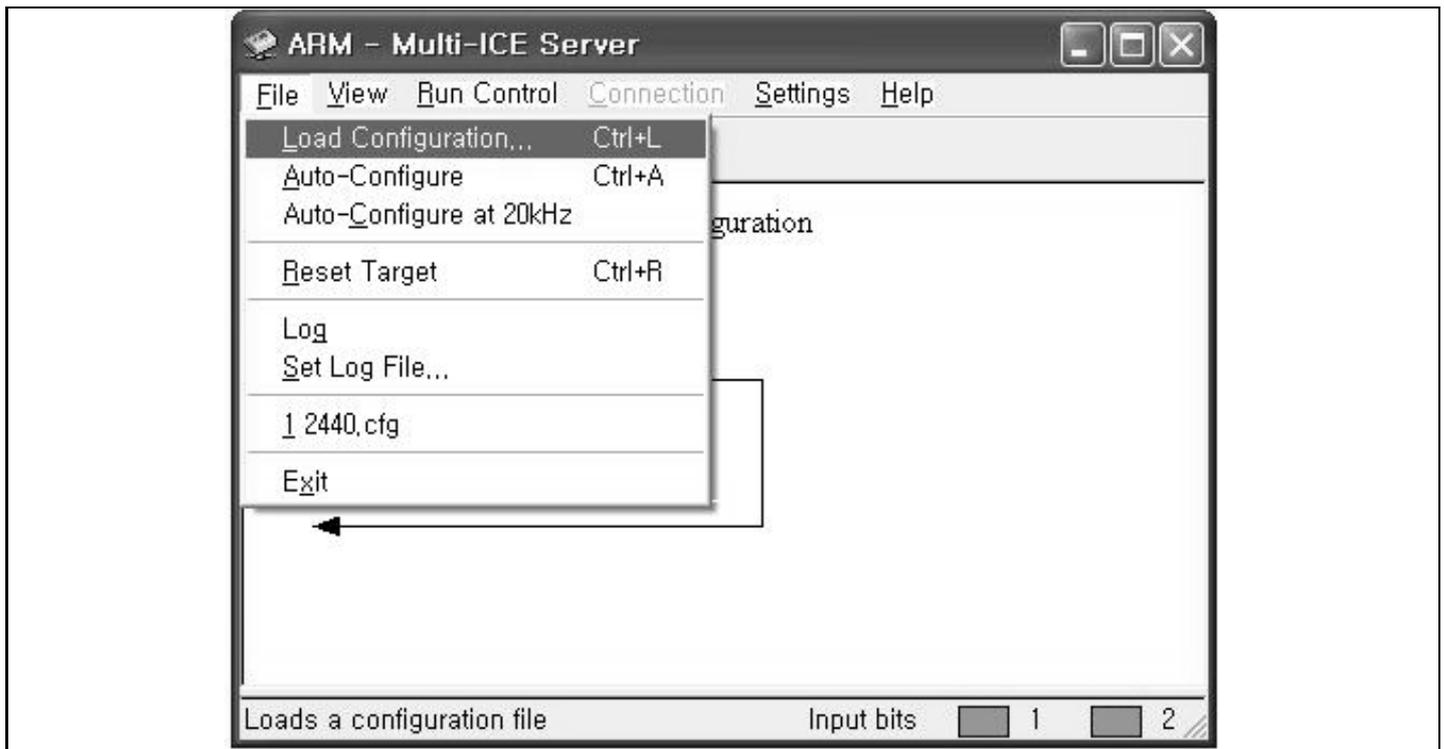


Figure 2-8. Load Configuration

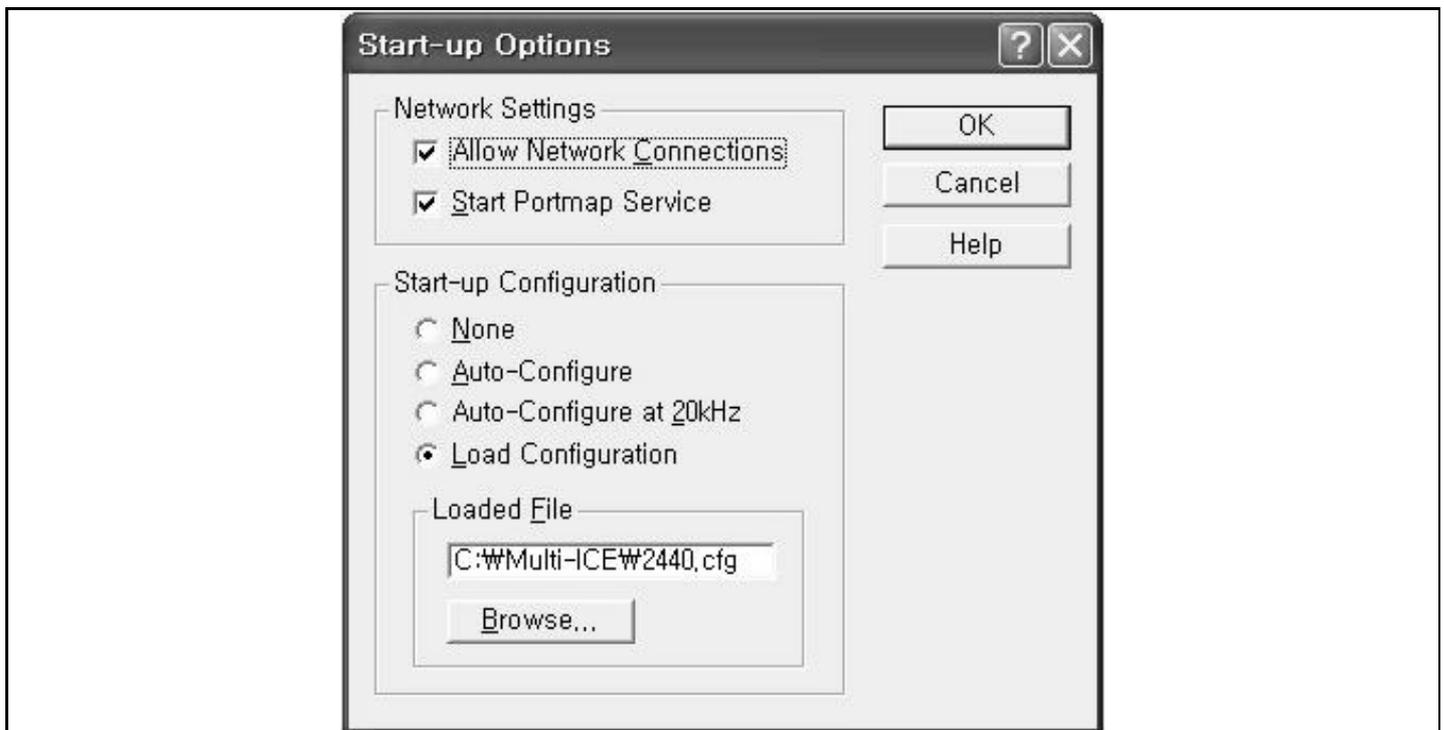


Figure 2-9. Start-up Configuration

## CONFIGURING ARM DEBUGGER FOR ARM MULTI-ICE

In order to access a remote target, you should configure ARM Debugger for Windows (ADW) or ARM Extended Debugger (AxD). There are two kinds of ADW: one for Software Development Toolkit (SDT) and the other for ARM Developer Suit (ADS). These two kinds of ADW are basically same except some trivial differences. The below explanation will be described with AXD of ARM Developer Suit (ADS).

The MULTI-ICE interface unit must also be configured for the ARM core in the target system. The ARM920T core is contained in the S3C2440X on the SMDK2440 board.

To configure AXD Debugger using the MULTI-ICE interface, follow the steps:

1. Select Configure Debugger from the Options menu.  
Options -> Configure Target
2. Debugger Configuration dialog box is displayed (See Figure 2-10).  
If there is no Multi-ICE in the target environment, then you have to select Add button and Multi-ICE.DLL.
  - ARMulator: lets you execute the ARM program without any physical ARM hardware by simulating ARM instructions in software.
  - Multi-ICE: connects the AXD Debugger directly to the target board or to a MULTI-ICE unit attached to the target.
3. Select Multi-ICE from Target environment, and click the Configure button.
4. Configure ARM Multi-ICE dialog box (See Figure 2-11).
  - Connect page: select your host and MULTI-ICE communication target configuration.
  - Processor Settings page: set the cache clean code address.
5. Select Advanced from Debugger Configuration dialog box (See Figure 2-12) and configure it.
  - Endian: little (If the big endian is used, Endian: big has to be selected.)
6. If you click the OK button on Debugger Configuration dialog box, the debugger will be restarted. The restarting dialog box is displayed and numbers are rapidly changing, indicating that it is reading and writing to the target. This means that the executable image file is downloaded to the SDRAM code area.

This configuration is initially done and the setting is saved, which relieves the user of repeating another configuration next time.

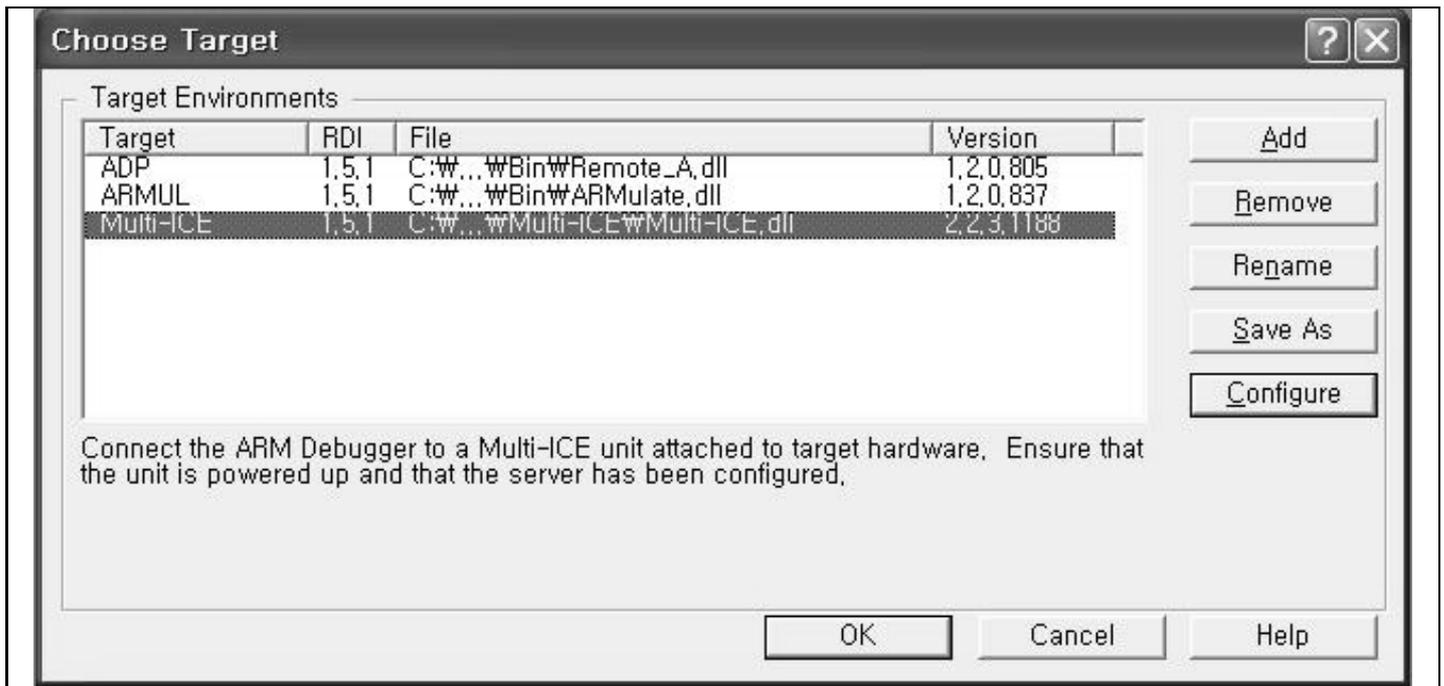


Figure 2-10. Debugger Configuration: Target Page

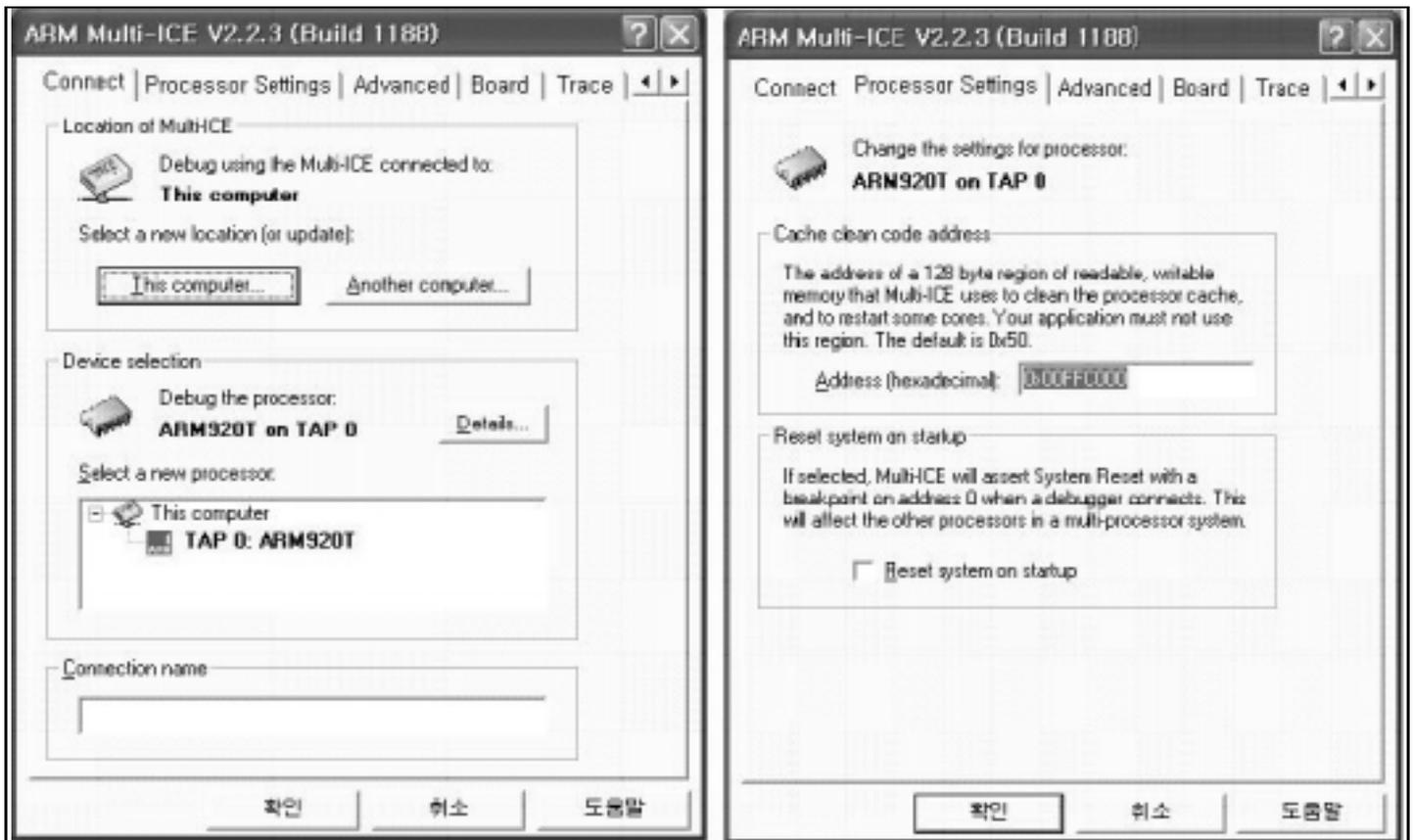


Figure 2-11. ARM Multi-ICE: Connect Page and Processor Settings Page

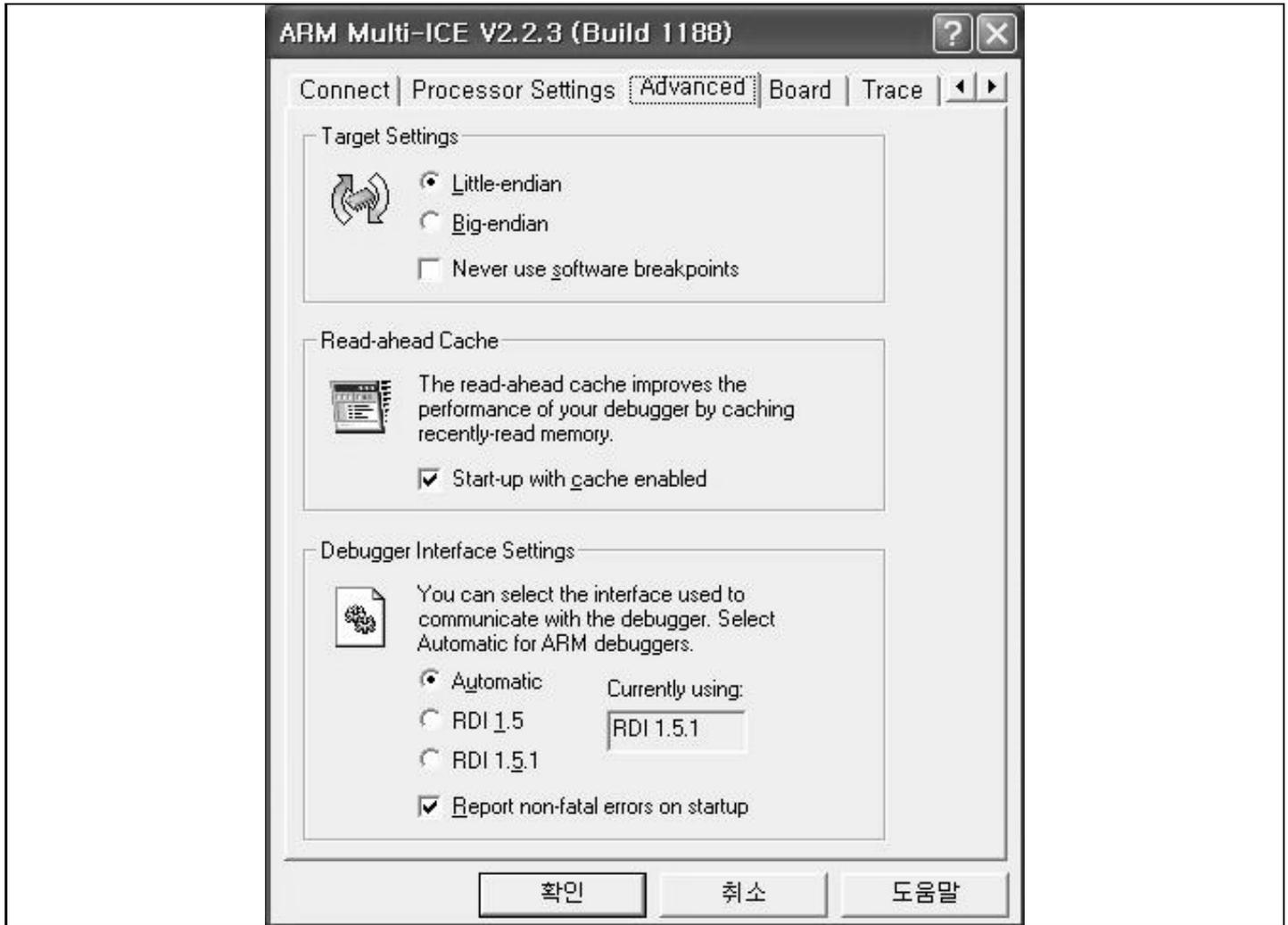


Figure 2-12. Debugger Configuration: Advanced Page

## EXECUTING 2440TEST.AXF USING ARM MULTI-ICE

1. Initialize internal variables of the debugger. After a downloading, several windows are displayed, such as Execution window, Console window, and Command window. In Command window, you should initialize the internal variables of the debugger, "\$semihosting\_enabled" and "\$vector\_catch", by entering the following command:

```
swat $vector_catch 0x00
swat $semihosting_enabled 0x00 ;To use all H/W break points
swat psr %IFt_SVC ;To disable all interrupts
com swat psr %IF_SVC32
```

Or, you can initialize these variables as follows:

First, create a text file named "2440norom.ini", which includes the commands described above. Then, enter the following command in the Command window (See Figure 2-13):

```
obey C:\WORK\2440\2440norom\2440norom.ini
```

For more information about these steps, refer to the reference document released by ARM.

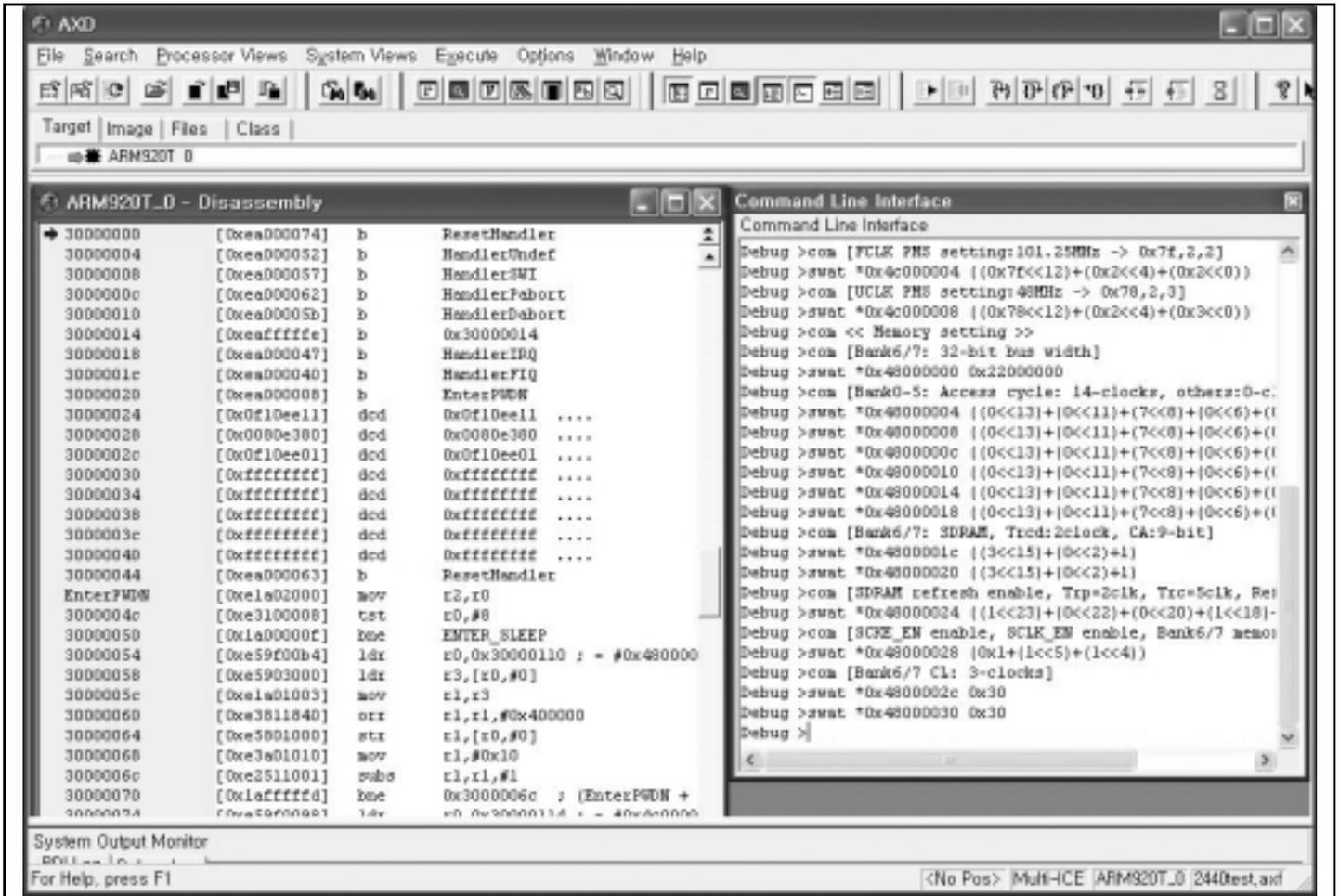


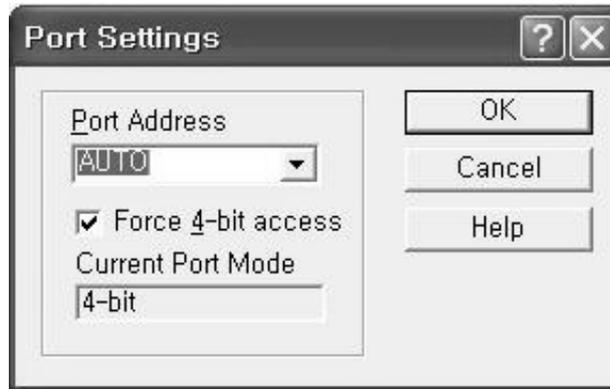
Figure 2-13. ARM Extended Debugger Window (AxD): Command Window

2. Set breakpoint at Main in 2440TEST.c as follows:  
break Main
3. Execute the program by clicking Execute menu→Go. The program execution will stop at Main( ).
4. Now, the downloaded image file will run on SDRAM area. 2440TEST program running status can be monitored on the DNW.

## MULTI-ICE CHECKPOINTS

### 1. Error messages

Refer to Error Messages of the Multi-ICE user's guide. If you cannot solve the problem by using the instructions in the user's guide, then apply the 'Force 4-bit access' option.



### 2. Multi-ICE current consumption problem

Multi-ICE draws the Multi-ICE operating current from a target board. The current is about 130mA at 3.3V. If the target board cannot supply the 130mA, an external power supply must be used for supplying the current to Multi-ICE.

### 3. nTRST, TMS, TCK and TDI pin connections

TMS, TCK and TDI pin must be pulled-up with 10K registers. If the Multi-ICE is not used when development is completed, nTRST must be 'L' level at least during the reset.

## HOW TO USE ARM DEBUGGER WITH OPENICE32-A900

Followings explain how to download the compiled image to SDRAM memory on the SMDK2440 by ARM debugger through OPENice32-A900, an emulator for ARM processor.

Note:

If you need technical support of OPENice32-A900, please contact AIJI System (<http://www.aijisystem.com>, [openice@aijisystem.com](mailto:openice@aijisystem.com))

## CONFIGURING ARM DEBUGGER FOR OPENice32-A900

To debug the target board with OPENice32-A900, you should configure ARM Debugger for Windows (ADW) or ARM Extended Debugger (AxD). As MULTI-ICE, OPENice32-A900 should be connected through JTAG port on the board and switched on.

To configure ARM Debugger for OPENice32-A900 interface, follow the steps:

1. Select Configure Debugger from the Options menu.

Options -> Configure Target

2. Debugger Configuration dialog box is displayed (See Figure 2-14). If there is no OPENice32-A900 in the target environment box, then you have to click on the Add button and select OPENice32-A900.DLL.
  - ARMulator: lets you execute the ARM program without any physical emulator by simulating ARM instructions in software.
  - OPENice32-A900: connects the ARM debugger to OPENice32-A900 attached to the target board.
3. Select OPENice32-A900 from the Target environment box, and click the Configure button.

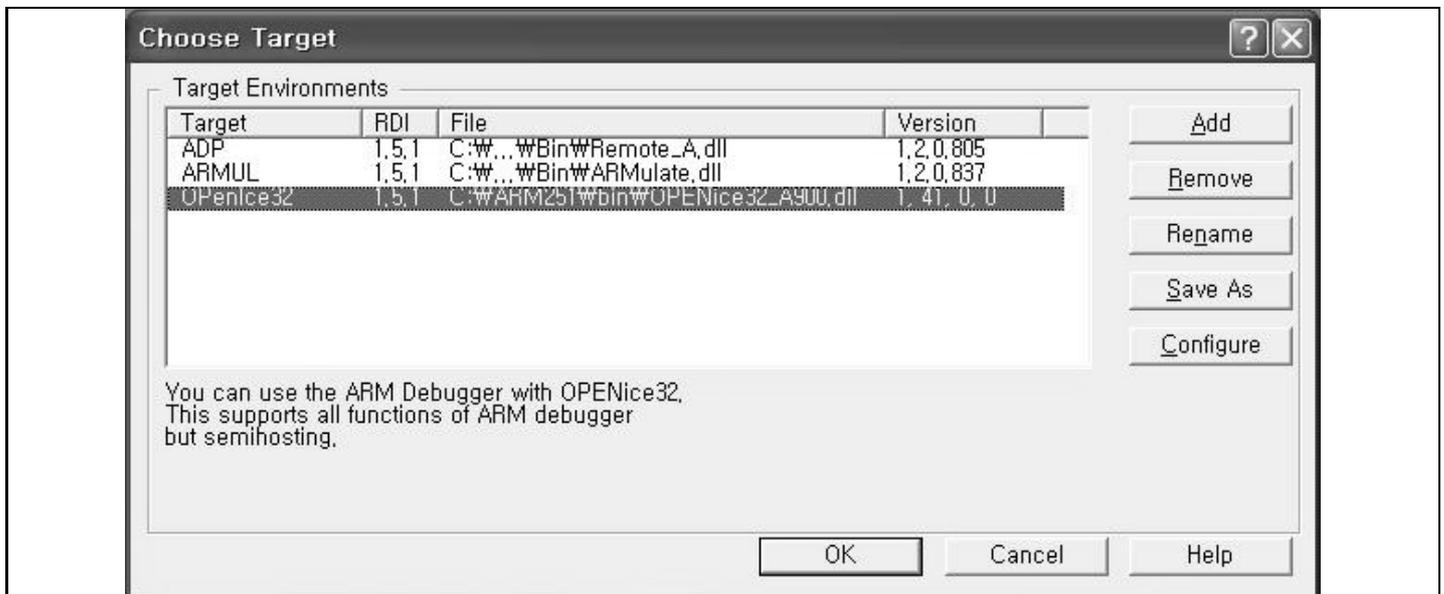


Figure 2-14. Debugger Configuration: Target Page

4. Configure OPENice32-A900 dialog box (See Figure 2-15, 2-16).
- Remote page: select the connection to OPENICE32-A900.



Figure 2-15. OPENice32-A900 Configuration: Communication Setting Page

- Debugger page : set the Endian and decide where initializes S3C2440X without any boot ROM.  
Endian : little (If the big endian is used, Endian: big has to be selected.)  
It should be matched with the option that you set in the compiler.  
To init SMU : If you want to initialize S3C2440X on the board without any boot ROM, check it and set SMU in the SMU page, (Don't check it in this application.)  
Flash download : If image file will be downloaded to a flash device, check it and set options in the Flash config page. (Don't check it in this application.)



Figure 2-16. OPENice32-A900 Configuration: Endian and SMU Setting Page

- SMU page: set SMU of S3C2440X. Select SMDK2440 or S3C2440X from the device name and modify the values. (No need to set it in this application). SMU of S3C2440 is the same as S3C2410. So, if you cannot find S3C2440 from the device name, then you may select SMU of S3C2410 instead of S3C2440.

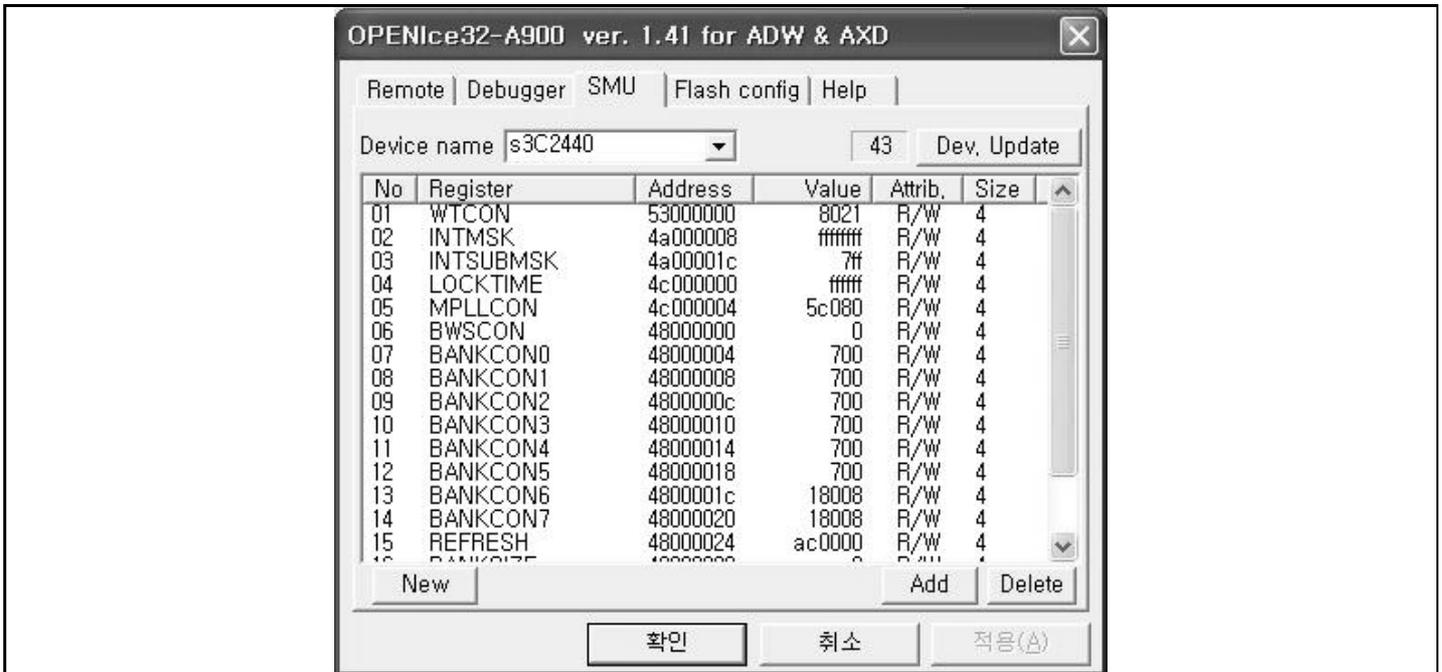


Figure 2-17. OPENIce32-A900 Configuration: SMU Setting Page

- Flash Config page: set options for Flash download. (No need to set it in this application).

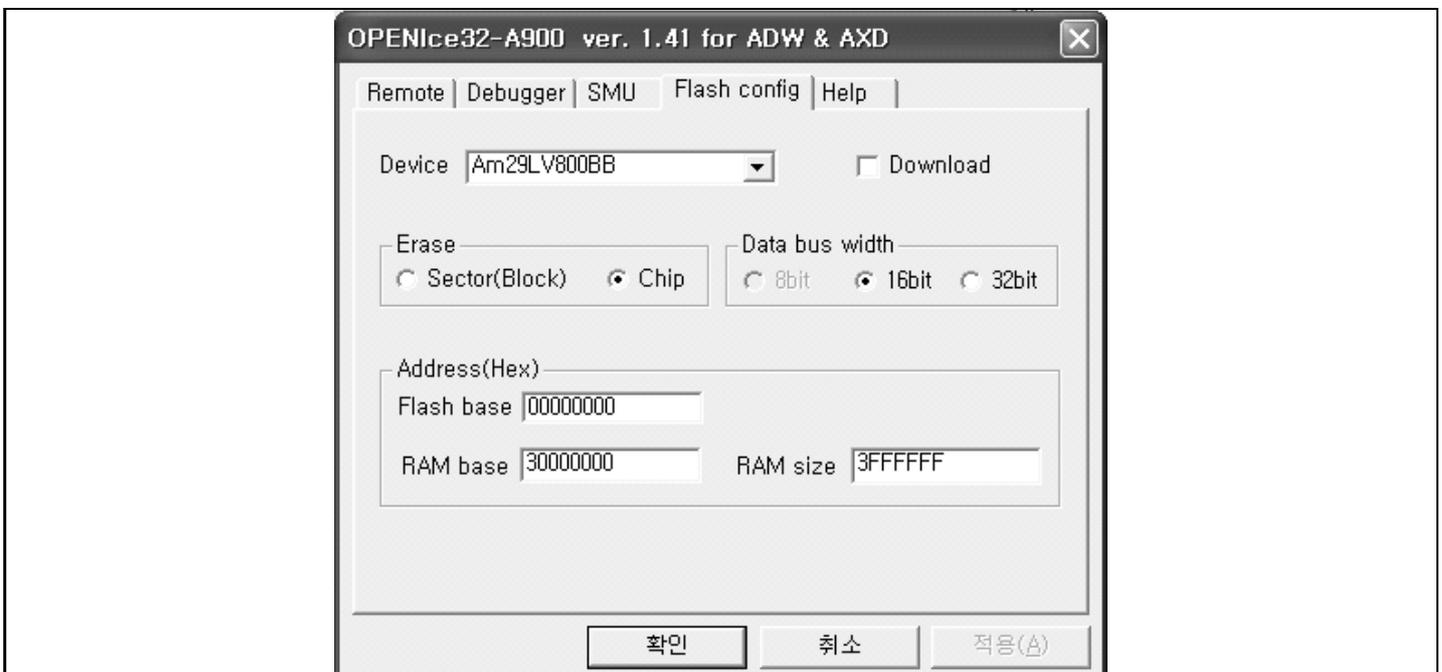
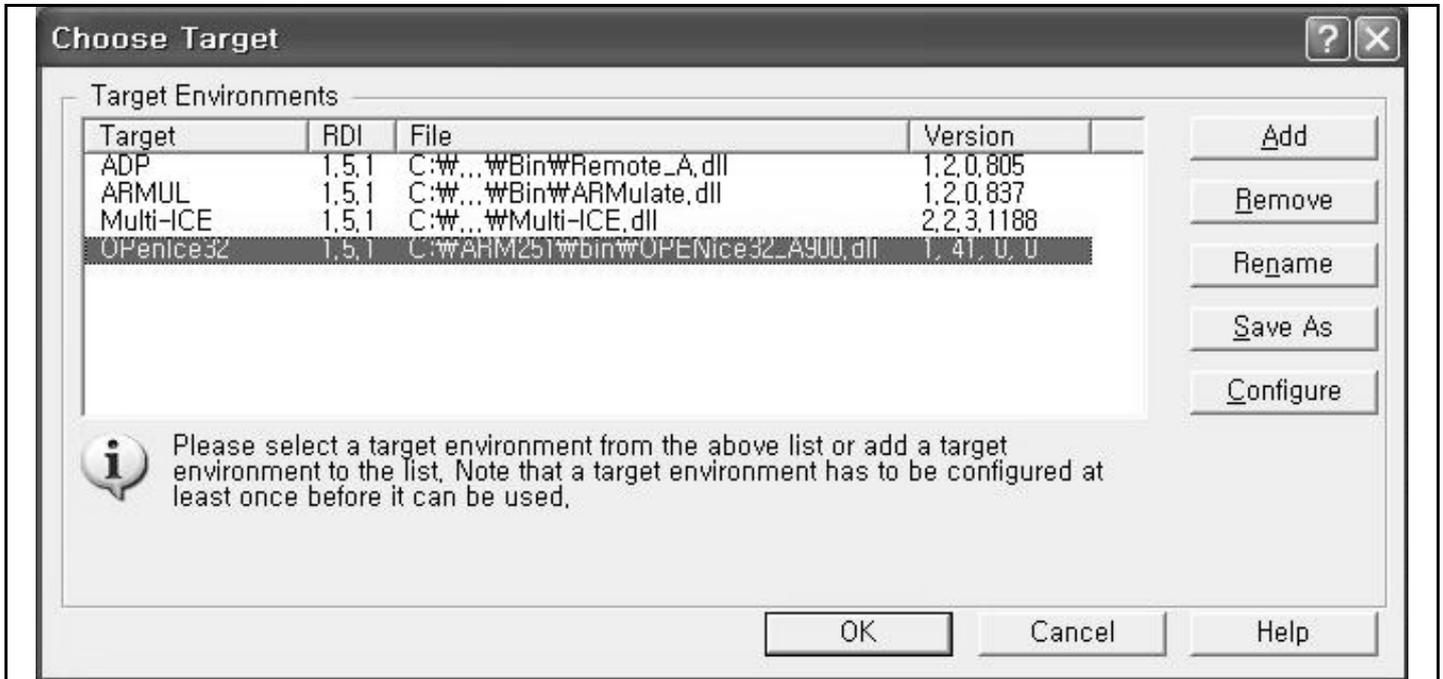


Figure 2-18. OPENIce32-A900 Configuration: Flash Configuration Setting Page

5. If you click the OK button on Choose Target dialog box (See Figure 2-19), the debugger will be restarted. The restarting dialog box is displayed and numbers are rapidly changing, indicating that it is reading and writing to the target. This means that the executable image file is downloaded to the SDRAM code area.



**Figure 2-19. Debugger Configuration: Choose Target**

This configuration is initially done and the setting is saved, which relieves the user of repeating another configuration next time.

## EXECUTING 2440TEST.AXF USING OPENICE32-A900

1. Select Load Image from the File menu and select the compiled image (2440TEST.AXF). Then it will be downloaded to the SDRAM on the board.
2. Execute the program by select Go from the Execute menu.
3. Now, the downloaded image file will run on SDRAM area. 2440TEST program running status can be monitored on the AXD.

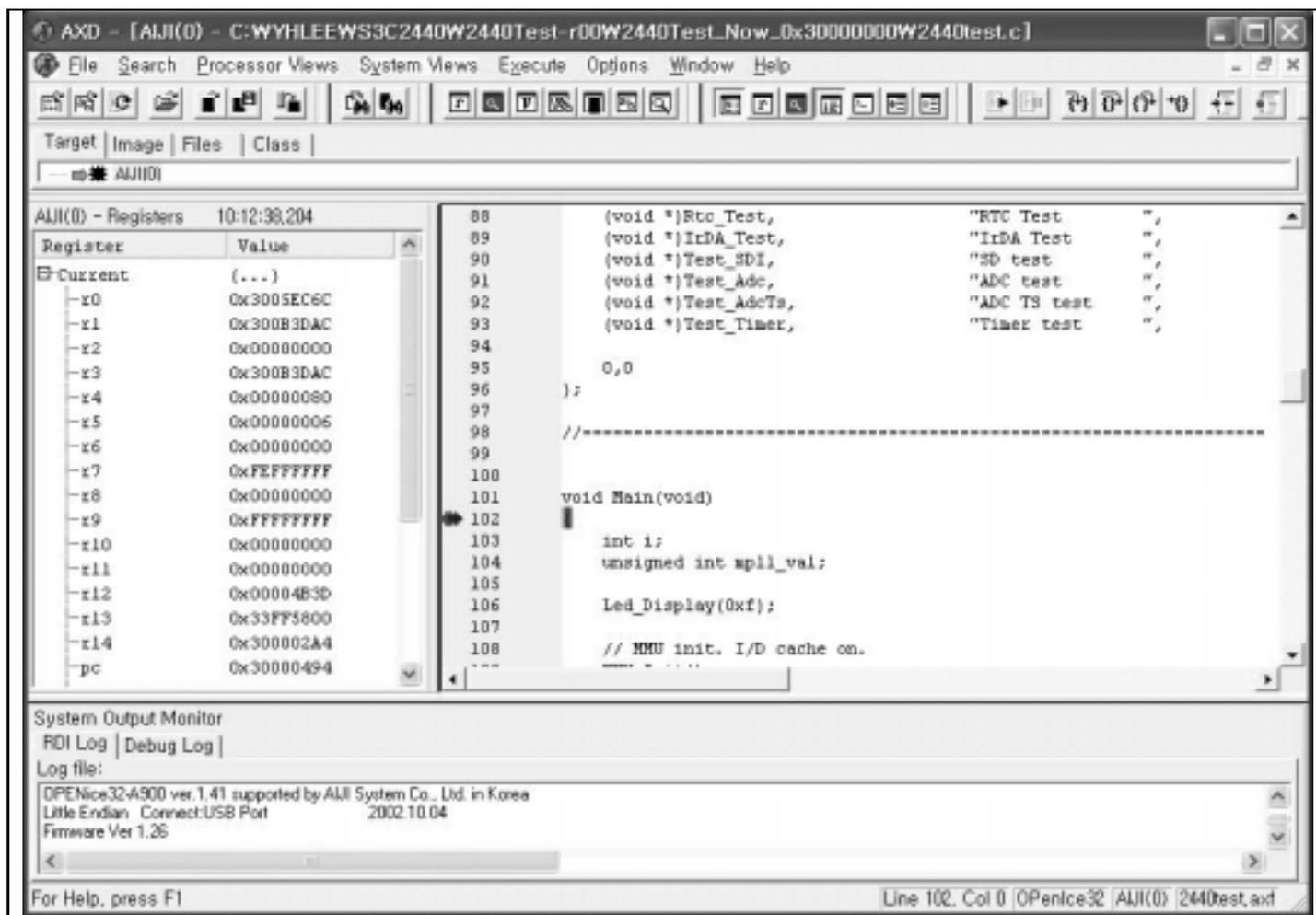


Figure 2-20. ARM Extended Debugger (AxD): After Downloading

## DEBUGGING DOWNLOADED IMAGE IN ADW OR AXD

### Stepping Through Program

To step through the program execution flow, you can select one of the following three options:

- Step: advances the program to the next line of code that is displayed in the execution window.
- Step Into: advances the program to the next line of code that follows all function calls. If the code is in a called function, the function source is displayed in the Execution window and the current code.
- Step Out: advances the program from the current function to the point from which it was called immediately after the function call. The appropriate line of code is displayed in the Execution window.

### Setting Breakpoint

A breakpoint is the point you set in the program code where the ARM debugger will halt the program operation. When you set a breakpoint, it appears as a red marker on the left side of the window.

To set a simple breakpoint on a line of code, follow these steps:

1. Double-click the line where you want to place a break, or choose Toggle Breakpoint from the Execute menu. The Set or Edit Breakpoint dialog box is displayed.
2. Set the count to the required value or expression (The program stops only when this expression is correct).

To set a breakpoint on a line of code within a particular program function:

1. Display a list of function names by selecting Function Names from View menu.
2. Double-click the function name you want to open. A new source window is displayed containing the function source.
3. Double-click the line where the breakpoint is to be placed, or choose Toggle Breakpoint from the Execute menu. The Set or Edit Breakpoint dialog box appears.
4. Set the count to the required value or expression (The program stops only when this expression is correct).

### Setting Watch Point

A watch point halts a program when a specified register or a variable, which is set to a specific number, is about to be changed.

To set a watch point, follow these steps:

1. Display a list of registers, variables, and memory locations you want to watch by selecting the Registers, Variables, and Memory options from the View menu.
2. Click the register, variable, or memory area in which you want to set the watch point. Then, choose Set or Edit Watchpoint from the Execute menu.
3. Enter a Target Value in the Set or Edit Watchpoint dialog box. Program operation will stop when the variable reaches the specified target value.

## VIEWING VARIABLES, REGISTERS, AND MEMORY

You can view and edit the value of variables, registers, and memory by choosing the related heading from the View menu:

- Variables: for global and local variables.
- Registers: for the current mode and for each of the six register view modes.
- Memory: for the memory area defined by the address you enter.

## DISPLAYING CODE INTERLEAVED WITH DISASSEMBLY

If you want to display the source code interleaved with disassembly, choose Toggle Interleaving on the Options menu. This command toggles between Displaying Source Only and Displaying Source Interleaved with Disassembly. When the source code is shown interleaved with disassembly, machine instructions appear in a lighter gray color.

For additional information about ARM Debugger, refer to the reference document released by ARM.

## SWITCHING DEVELOPMENT TOOLKIT

USB boot code (U2440mon.c) and test code (2440test.c) can be executed in the SDT or ADS by changing the option in OPTION.H and Makefile. In other words, boot and test code can be translated from ADS to SDT and vice versa by changing option in the following table.

**Table 2-1. Toolkit Switching Options**

	ADS	SDT
Makefile	fromelf -nodebug -bin -output \$(PRJ).bin \$(PRJ).elf	fromelf -nodebug -nozeropad \$(PRJ).elf -bin \$(PRJ).bin
OPTION.H	#define ADS10 TRUE	#define ADS10 FALSE

### TRANSLATING CODE FROM ADS INTO SDT

U2440MON and 2440TEST codes were optimized for ADS 1.0. In other words, these codes were compiled and linked by the ADS 1.0. So, these codes should be modified to work on the SDT.

If you want to compile our codes with the SDT, then you have to change the definition of ADS1.0 in OPTION.H from 'TRUE' to 'FALSE' and the option in makefile from 'fromelf -nodebug -bin -output \$(PRJ).bin \$(PRJ).elf' to 'fromelf -nodebug -nozeropad \$(PRJ).elf -bin \$(PRJ).bin' option.

### TRANSLATING CODE FROM SDT INTO ADS

First function `__rt_lib_init()`; is applied to the main code. And then old Makefile for SDT is changed to a new one for ADS.

If you have used SDT 2.50, it is recommended that you should read related documents (ADS, Getting Started, and ARM DUI0064A) about the difference between SDT 2.50 and ADS 1.0.

### REMOVED OR CHANGED ITEMS FROM MAKEFILE FOR SDT 2.50

- ARMLINK option
  - first: the path of an object file is not needed.
- ARMASM option
  - cpu: should be changed as -cpu ARM920T
  - apcs: should be changed to -apcs /noswst
- Compiler option
  - fc : should be removed.
  - zpz0 : should be removed. This is not needed any more.
  - apcs : should be changed to -apcs /noswst
  - processor : should be removed.
  - arch : should be removed.
  - cpu : should be added as -cpu ARM920T
- fromelf.exe
  - nozeropad: should be removed. This is not needed any more.
  - output : command line style should be changed using -output option as follows:  
fromelf -nodebug -bin -output \$(BIN)\\$(PRJ).bin \$(BIN)\\$(PRJ).AXF

**OTHER ITEMS SHOULD BE CHANGED FOR ADS 1.0**

## 1. ammake.exe

The armmake.exe is not supplied with ADS 1.0. So, you have to use your own Make utility. (nmake.exe, make.exe, pmake.exe, or armmake.exe in SDT 2.50).

## 2. Embedded library

There is no separate embedded library in ADS 1.0. All the library in ADS 1.0 is made for embedded applications.

But, the library must be initialized using `__rt_lib_init()` function. If you do not use `__rt_lib_init()`, the C library does not work well.

## 3. There is no tasm.exe. The tasm.exe is merged into armasm.exe.

**EXAMPLE OF MAKEFILE FOR ADS 1.0**

This is a sample makefile on ADS 1.0.

**##### File Definition #####**

```
PRJ = 2440test
INIT= 2440init
AM1 = 2440slib
AM2 = 2440swis
CM1 = 2440lib
CM2 = mmu
CM3 = 2440iis
CM4 = timer
CM5 = 2440RTC
CM6 = 2440IIC
```

```
.
.
.
```

```
CM38 = spi
CM39 = strata32
```

**##### Destination path Definition #####**

```
OBJ=. \obj
ERR=. \err
```

**##### ARM tool Definition #####**

```
ARMLINK = armlink
ARMASM = armasm
ARMCC = armcc
```

**##### Option Definition #####**

```
LFLAGS = -ro-base 0x30000000 -elf -map -xref \
-list list.txt -first $(INIT).o \init)
AFLAGS = -li -apcs /noswst -cpu ARM920T
CFLAGS = -c -g+ -li -apcs /noswst -cpu ARM920T
```

**##### Object combine Definition #####**

```
OBJS = $(OBJ)\$(INIT).o $(OBJ)\$(AM1).o $(OBJ)\$(AM2).o $(OBJ)\$(PRJ).o \
$(OBJ)\$(CM1).o $(OBJ)\$(CM2).o $(OBJ)\$(CM3).o $(OBJ)\$(CM4).o \
$(OBJ)\$(CM5).o $(OBJ)\$(CM6).o $(OBJ)\$(CM7).o $(OBJ)\$(CM8).o \
$(OBJ)\$(CM9).o $(OBJ)\$(CM10).o $(OBJ)\$(CM11).o $(OBJ)\$(CM12).o \
$(OBJ)\$(CM13).o $(OBJ)\$(CM14).o $(OBJ)\$(CM15).o $(OBJ)\$(CM16).o \
$(OBJ)\$(CM17).o $(OBJ)\$(CM18).o $(OBJ)\$(CM19).o $(OBJ)\$(CM20).o \
$(OBJ)\$(CM21).o $(OBJ)\$(CM22).o $(OBJ)\$(CM23).o $(OBJ)\$(CM24).o \
$(OBJ)\$(CM25).o $(OBJ)\$(CM26).o $(OBJ)\$(CM27).o $(OBJ)\$(CM28).o \
$(OBJ)\$(CM29).o $(OBJ)\$(CM30).o $(OBJ)\$(CM31).o $(OBJ)\$(CM32).o \
$(OBJ)\$(CM33).o $(OBJ)\$(CM34).o $(OBJ)\$(CM35).o $(OBJ)\$(CM36).o \
$(OBJ)\$(CM37).o $(OBJ)\$(CM38).o $(OBJ)\$(CM39).o
```



all: \$(PRJ).axf

clean:

del \$(OBJ)\\*.o

\$(PRJ).axf: \$(OBJS)

del \$(PRJ).bin

del \$(PRJ).axf

\$(ARMLINK) \$(LFLAGS) -o \$(PRJ).axf \$(OBJS)

**fromelf -nodebug -bin -output \$(PRJ).bin \$(PRJ).axf**

**#For SDT2.5 fromelf -nodebug -nozeropad \$(PRJ).elf -bin \$(PRJ).bin**

**#For ADS1.0 fromelf -nodebug -bin -output \$(PRJ).bin \$(PRJ).elf**

\$(OBJ)\\$(PRJ).o: \$(PRJ).c 2440addr.h 2440lib.h makefile

del \$(OBJ)\\$(PRJ).o

del \$(ERR)\\$(PRJ).err

\$(ARMCC) \$(CFLAGS) \$(PRJ).c -o \$(OBJ)\\$(PRJ).o -Errors \$(ERR)\\$(PRJ).err

.  
.
   
.
   
.
   
.

\$(OBJ)\\$(CM27).o: \$(CM27).c 2440addr.h 2440lib.h makefile

del \$(OBJ)\\$(CM27).o

del \$(ERR)\\$(CM27).err

\$(ARMCC) \$(CFLAGS) \$(CM27).c -o \$(OBJ)\\$(CM27).o -Errors \$(ERR)\\$(CM27).err

\$(OBJ)\\$(CM28).o: \$(CM28).c 2440addr.h 2440lib.h makefile

del \$(OBJ)\\$(CM28).o

del \$(ERR)\\$(CM28).err

\$(ARMCC) \$(CFLAGS) \$(CM28).c -o \$(OBJ)\\$(CM28).o -Errors \$(ERR)\\$(CM28).err

\$(OBJ)\\$(CM29).o: \$(CM29).c 2440addr.h 2440lib.h makefile

del \$(OBJ)\\$(CM29).o

del \$(ERR)\\$(CM29).err

\$(ARMCC) \$(CFLAGS) \$(CM29).c -o \$(OBJ)\\$(CM29).o -Errors \$(ERR)\\$(CM29).err

.  
.
   
.
   
.
   
.

# 3

## PROGRAMMING FLASH MEMORIES

### PROGRAMMING NAND FLASH MEMORY

The SMDK2440 supports NAND flash control interface. There are two methods to write images to NAND flash memory:

- Write image files to NAND flash memory with write-program.
- Write image files to NAND flash memory with JTAG interface.

## NAND FLASH WRITE WITH WRITE-PROGRAM

The target image must be downloaded in SDRAM before executing write-program.

To download and write a target image from the host to SDRAM through USB interface, follow the steps:

1. Run the DNW utility program (See Figure 3-1).



Figure 3-1. DNW Window to Download

2. Select Serial Port on the system menu of DNW and click Connect to open the serial port (See Figure 3-2, 3).



Figure 3-2. DNW Window (to Connect Serial Port)

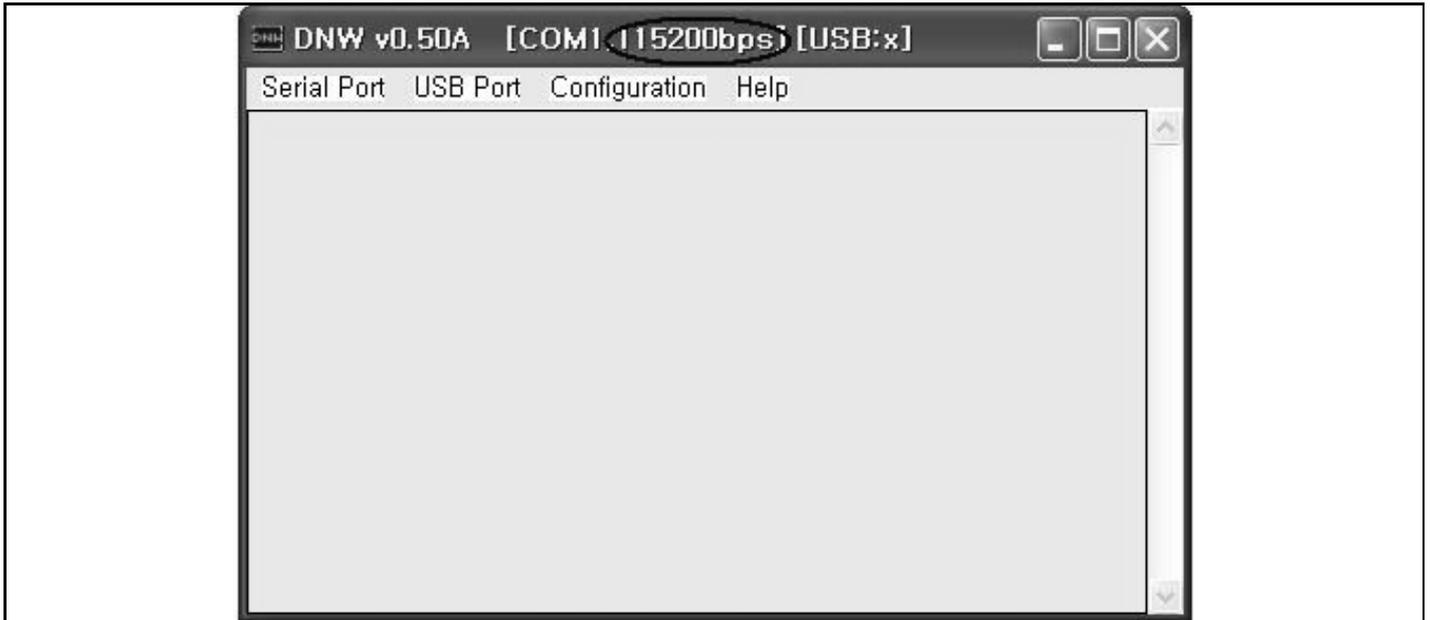


Figure 3-3. DNW Window (after Open Baud-rate is Printed on Title Bar)

3. Connect the serial and USB cable from the host PC to SMDK2440 system and turn on the power of SMDK2440 board (See Figure 3-4).

**NOTES:**

1. Jumper J1-B, J2-B, J3-B, J4-B must be 'H', 'L', 'L' and 'H' for AMD NOR Booting.
2. SMDK2440 must run monitor program that is provided by SAMSUNG.

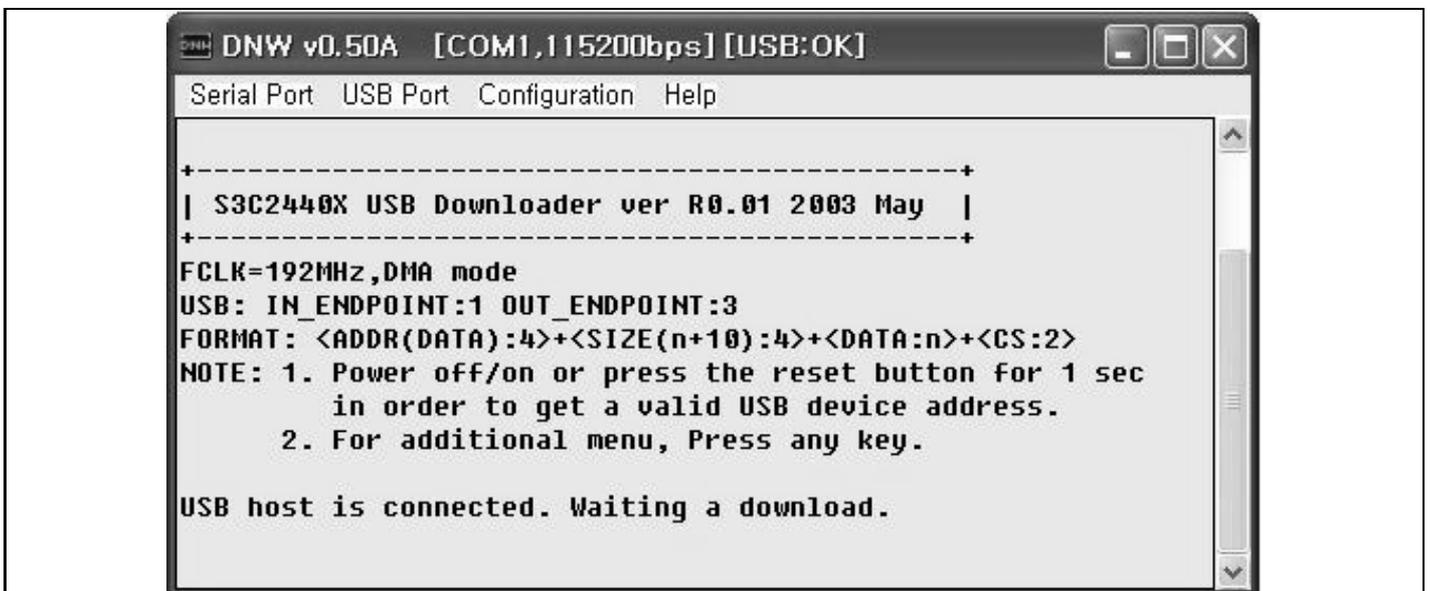


Figure 3-4. DNW Window (after Turning on the SDMK2440)

4. To see the additional menu, press any key on the DNW window (See Figure 3-5).

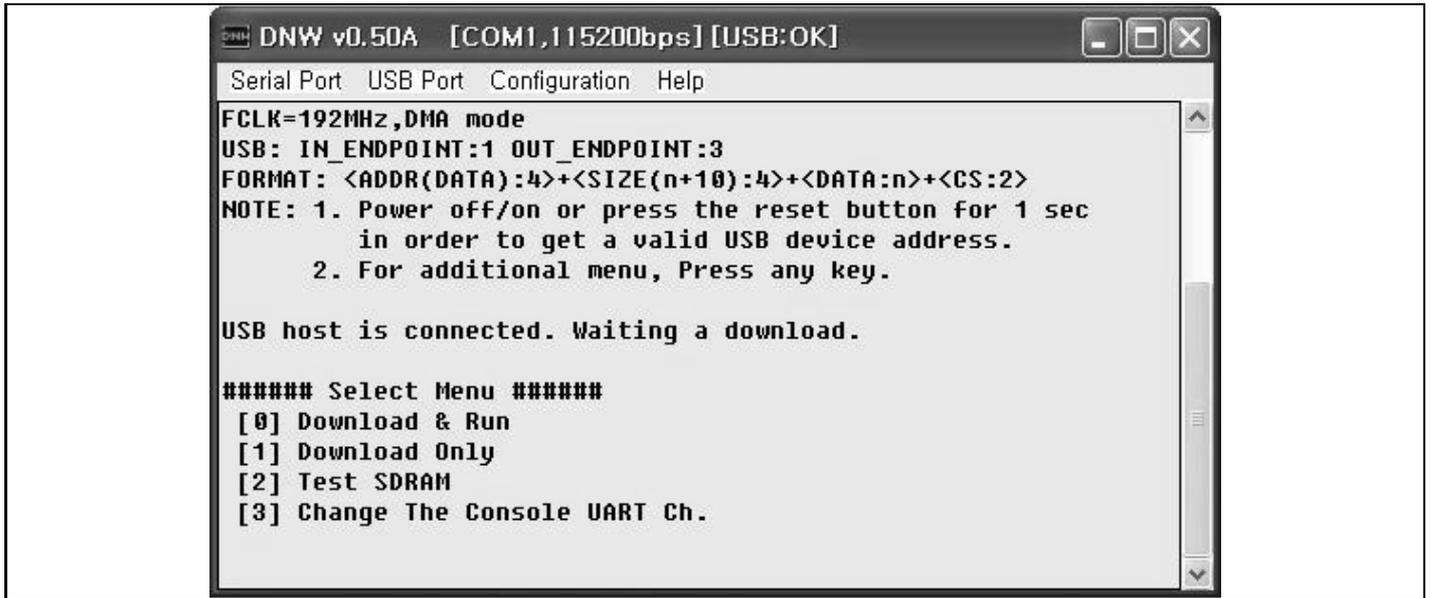


Figure 3-5. DNW Window to Download

5. For downloading a target image, select Download Only item on the DNW window (See Figure 3-6).  
 6. Write the address to download and press enter key (See Figure 3-6).

**NOTE:** The target image must be located on 0x30100000 address.

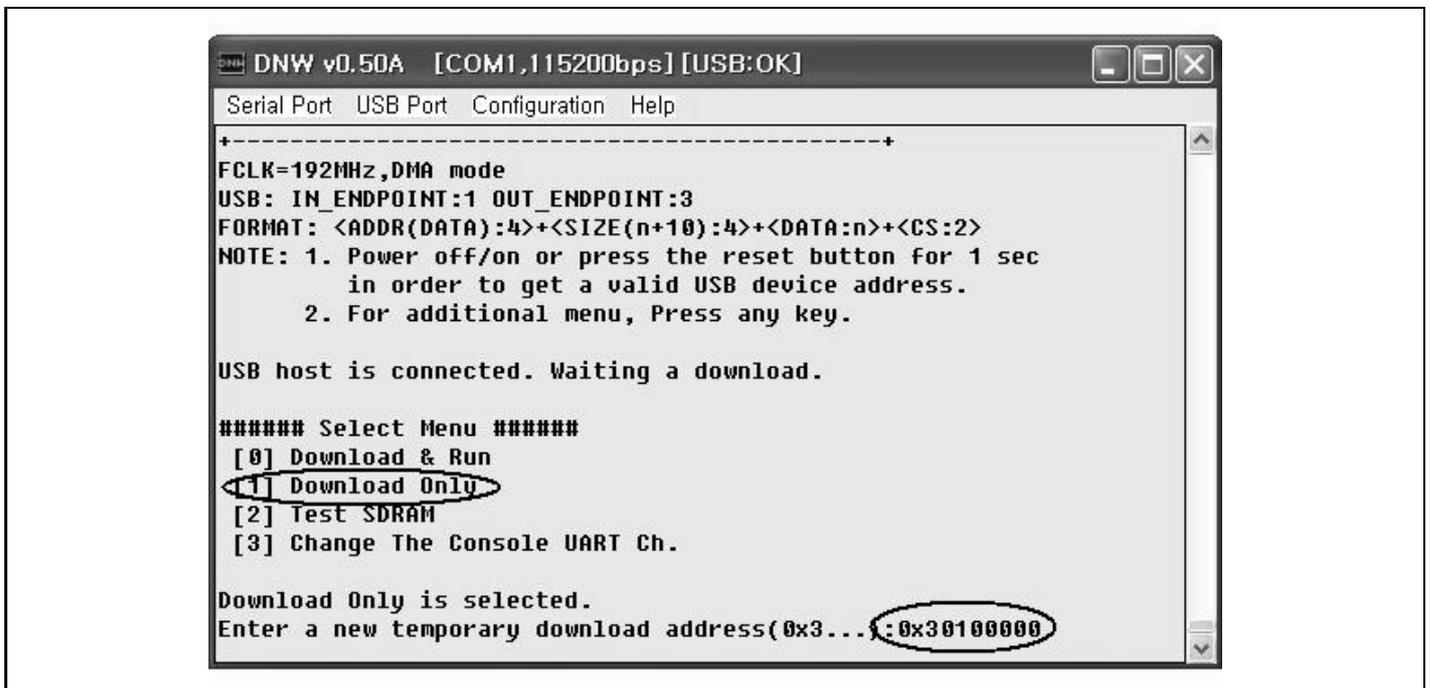


Figure 3-6. DNW Window (to Select Target Image)

7. Select USB Port on the system menu of the DNW and click Transmit to download a target image (See Figure 3-7).

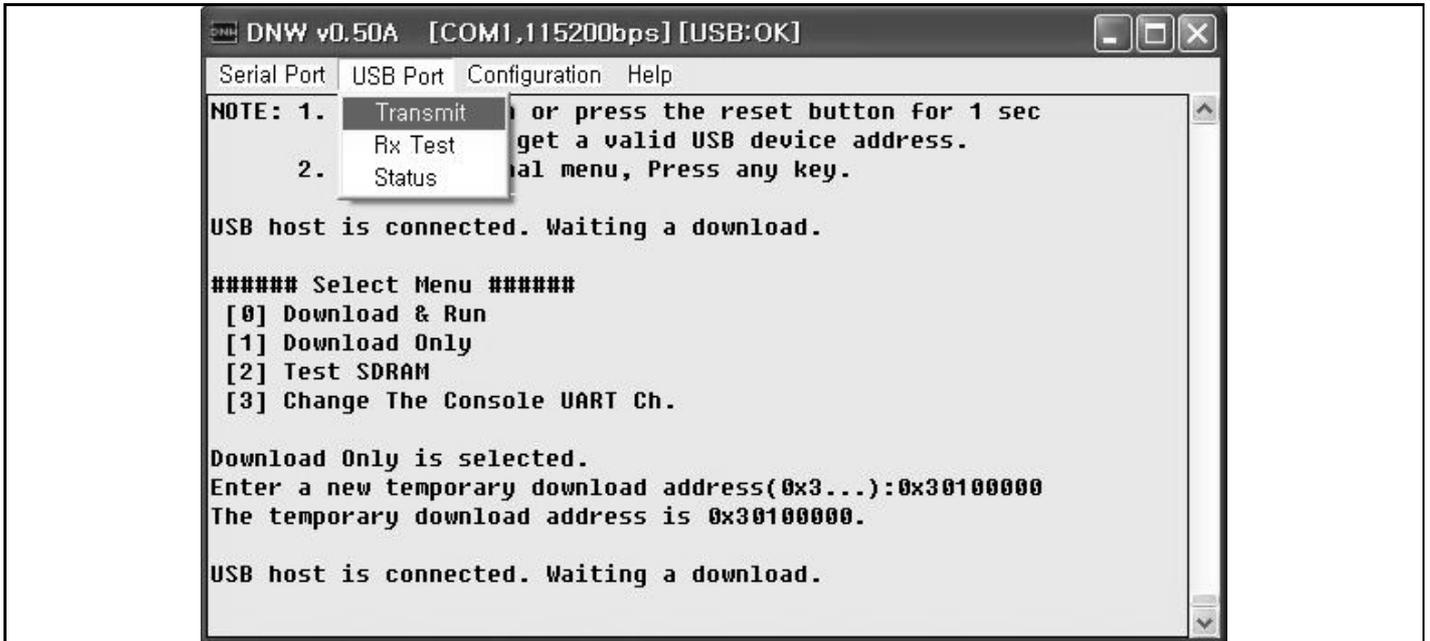


Figure 3-7. DNW Window (for USB Downloading)

8. Select a target image on file open dialog box (See Figure 3-8).



Figure 3-8. DNW Window (File Open Dialog Box)

9. For downloading 2440test program, select Download & Run item on the DNW window and download 2440test program like step 7 (See Figure 3-9).

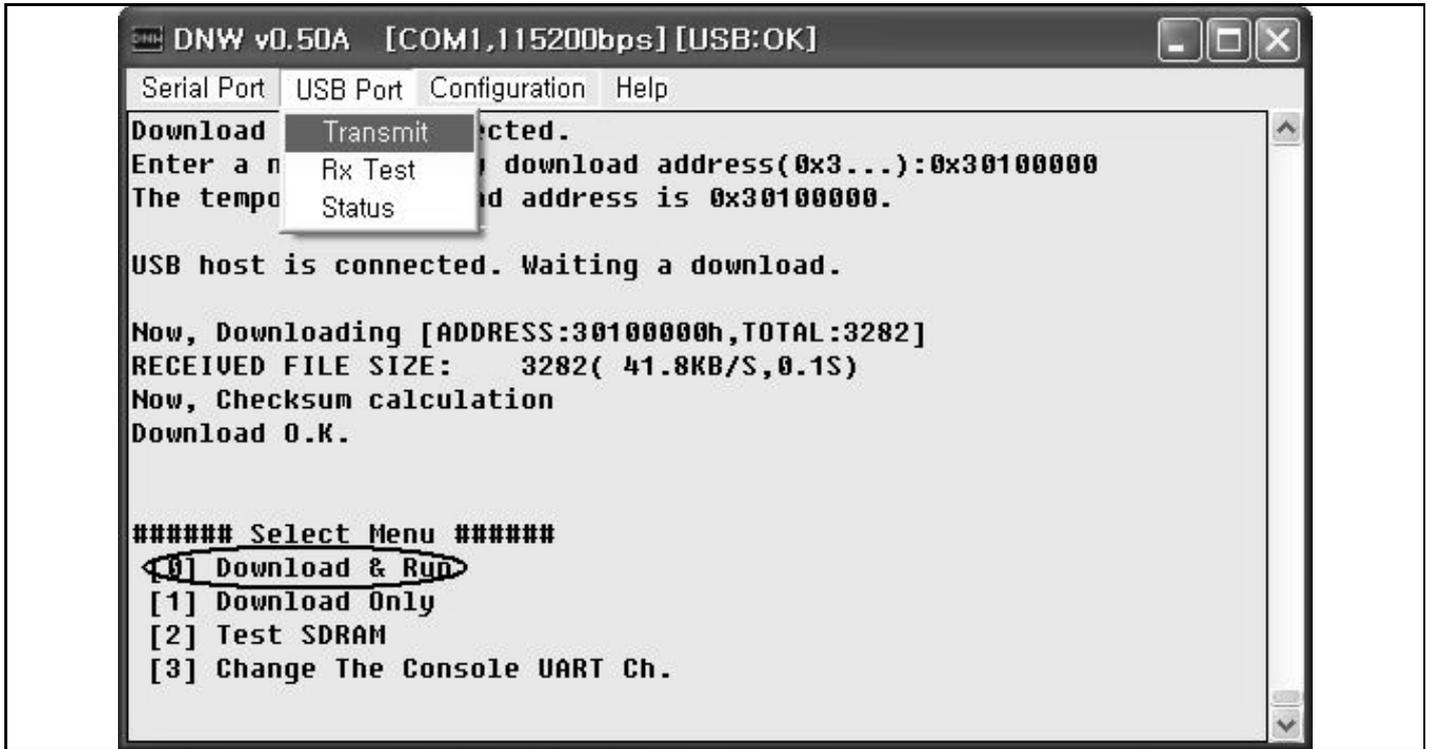


Figure 3-9. DNW Window (File Open Dialog Box)

```

DNW v0.50A [COM1,115200bps] [USB:OK]
Serial Port USB Port Configuration Help

[SMDK2440 Board Test Program Ver 0.0]

[Fclk:Hclk/Pclk]=[400.0:133.3:66.7]Mhz
[Uclk=48.0Mhz]

0:User Test      1:Manual Reg Set  2:PCMCIA test    3:Stepping stone
4:Nand test      5:Program Flash  6:DMA test       7:Interrupt test
8:Power/Clk test 9:Lcd test       10:Camera test   11:SPI Test
12:IIC Test      13:RTC Test      14:IrDA Test     15:SD test
16:ADC test      17:ADC TS test   18:Timer test    19:IIS test
20:Uart Test

Select the function to test : 4

Nand test
Select Nand flash type, Normal(1)/Advanced(2) : 1

K9S1208 Nand flash test start.

0:Read ID        1:Nand reset      2:Block erase     3:Page read
4:Page write     5:Nand R/W test   6:Check Badblock 7:Nand Block lock
8:Soft Unlock    9:K9S1208 Program

Select(-1 to exit): 9

[SMC(K9S1208U0M) NAND Flash writing program]
The program buffer: 0x30100000~0x31ffffff

Source size:0h~0h

Available target block number: 0~4095
Input target block number:

```

Figure 3-10. DNW Window (2440test Program)

10. Write '5' and press enter key on the DNW window (See Figure 3-10).
11. Select Nand flash type (See Figure 3-10).
12. Write '4' and press enter key on the DNW window (See Figure 3-10).
13. Write the target block number that is the start block to write and press enter key on the DNW window (See Figure 3-11).
14. Write total byte size of the target image, it should be aligned 0x4000 (one block size) bytes (See Figure 3-11).

NOTE: 2440test program supports normal NAND flash type (K9S1208: SmartMedia card, SAMSUNG) and advanced NAND flash type (K9K2G16: SAMSUNG). To write another type of device, it is required to modify the source codes NAND.C(normal K9S1208) or K9K2G16.C(advanced K9K2G16).

```

DNW v0.50A [COM1,115200bps] [USB:OK]
Serial Port  USB Port  Configuration  Help

0:Read ID      1:Nand reset   2:Block erase   3:Page read
4:Page write   5:Nand R/W test 6:Check Badblock 7:Nand Block lock
8:Soft Unlock  9:K9S1208 Program

Select(-1 to exit): 9

[SMC(K9S1208U0M) NAND Flash writing program]
The program buffer: 0x30100000~0x31ffffff

Source size:0h~0h

Available target block number: 0~4095
Input target block number: 0
Input program file size(bytes): 0x4000
File:16384[1-block,0-page,0-bytes].

0:Read ID      1:Nand reset   2:Block erase   3:Page read
4:Page write   5:Nand R/W test 6:Check Badblock 7:Nand Block lock
8:Soft Unlock  9:K9S1208 Program

Select(-1 to exit): 3
SMC(K9S1208U0M) NAND Page Read.
Block # to read: 0
Page # to read: 0
Read OK.
Read data(0-block,0-page)

 0: 74 00 00 ea 52 00 00 ea 57 00 00 ea 62 00 00 ea
10: 5b 00 00 ea fe ff ff ea 47 00 00 ea 40 00 00 ea
20: 08 00 00 ea 11 ee 10 0f 80 e3 80 00 01 ee 10 0f
30: ff ff
40: ff ff ff ff 63 00 00 ea 00 20 a0 e1 08 00 10 e3
50: 0f 00 00 1a b4 00 9f e5 00 30 90 e5 03 10 a0 e1
60: 40 18 81 e3 00 10 80 e5 10 10 a0 e3 01 10 51 e2

```

Figure 3-11. DNW Window (NAND Flash Write Program)

15. To check the contents of NAND flash, select Page read function (See Figure 3-11).

## AUTO BOOTING THROUGH NAND FLASH

The S3C2440 supports auto booting operation with NAND flash memory.

Before power on the SMDK2440 system, it must have SmartMedia card with boot-loader and OS image.

### NOTES:

1. Jumper J1-B, J2-B, J3-B, J4-B must be 'L', 'L', 'L' and 'L' for NAND Booting.
2. 2440test program and boot-loader, which is supplied by SAMSUNG, support normal NAND flash type (K9S120: SmartMedia card, SAMSUNG) and advanced NAND flash type (K9K2G16: SAMSUNG). To write another type of device, it is required to modify the source codes NAND.C(normal K9S1208) or K9K2G16.C(advanced K9K2G16).

## BOOTING NAND FLASH

To make NAND flash memory for auto booting, follow the steps:

1. Program the boot-loader image to the block 0 of NAND flash memory.
2. Program the OS image to the other blocks of NAND flash memory. The OS image must be located block 1 to the rest blocks.



## NAND FLASH ECC (ERROR CHECKING AND CORRECTION)

The S3C2440X supports ECC algorithm, which is based on XOR calculation, for error checking and correction.

### 1. Example of one byte ECC (find error bit)

Old

D7	D6	D5	D4	D3	D2	D1	D0
1	0	0	0	0	0	0	0

Old ECC code

P3	NP3	P2	NP2	P1	NP1
1	0	1	0	1	0

New

D7	D6	D5	D4	D3	D2	D1	D0
1	1	0	0	0	0	0	0

New ECC code

P4	NP3	P2	NP2	P1	NP1
0	0	0	0	1	1

Old ECC ^ New ECC

S3	NS3	S2	NS2	S1	NS1
1	0	1	0	0	1

$$P3 = [7] \wedge [6] \wedge [5] \wedge [4]$$

$$P2 = [7] \wedge [6] \wedge [3] \wedge [2]$$

$$P1 = [7] \wedge [5] \wedge [3] \wedge [1]$$

$$NP3 = [3] \wedge [2] \wedge [1] \wedge [0]$$

$$NP2 = [5] \wedge [4] \wedge [1] \wedge [0]$$

$$NP1 = [6] \wedge [4] \wedge [2] \wedge [0]$$

	P3	NP3	P2	NP2	P1	NP1
Old	1	0	1	0	1	0
New	0	0	0	0	1	1
(Old ECC) ^ (New ECC)	1	0	1	0	0	1
Error code	1		1		0	
Result	Bit 6					

#### NOTES:

- [n] means bit [n] (or Dn).
- The '10b' value of 'Old ECC ^ New ECC' corresponds to '1b' Error code and '01b' corresponds to '0b'.

## 2. Example of n byte ECC

	D7	D6	D5	D4	D3	D2	D1	D0
0th byte	[7]0	[6]0	[5]0	[4]0	[3]0	[2]0	[1]0	[0]0
1st byte	[7]1	[6]1	[5]1	[4]1	[3]1	[2]1	[1]1	[0]1
2nd byte	[7]2	[6]2	[5]2	[4]2	[3]2	[2]2	[1]2	[0]2
3rd byte	[7]3	[6]3	[5]3	[4]3	[3]3	[2]3	[1]3	[0]3
...	...							
509th byte	[7]509	[6]509	[5]509	[4]509	[3]509	[2]509	[1]509	[0]509
510th byte	[7]510	[6]510	[5]510	[4]510	[3]510	[2]510	[1]510	[0]510
511th byte	[7]511	[6]511	[5]511	[4]511	[3]511	[2]511	[1]511	[0]511

## — Column parity (CPn)

$CP3 =$ $[7]0^{\wedge} [7]1^{\wedge} [7]2^{\wedge} [7]3^{\wedge} \dots [7]509^{\wedge} [7]510^{\wedge} [7]511^{\wedge}$ $[6]0^{\wedge} [6]1^{\wedge} [6]2^{\wedge} [6]3^{\wedge} \dots [6]509^{\wedge} [6]510^{\wedge} [6]511^{\wedge}$ $[5]0^{\wedge} [5]1^{\wedge} [5]2^{\wedge} [5]3^{\wedge} \dots [5]509^{\wedge} [5]510^{\wedge} [5]511^{\wedge}$ $[4]0^{\wedge} [4]1^{\wedge} [4]2^{\wedge} [4]3^{\wedge} \dots [4]509^{\wedge} [4]510^{\wedge} [4]511^{\wedge}$	$NCP3 =$ $[3]0^{\wedge} [3]1^{\wedge} [3]2^{\wedge} [3]3^{\wedge} \dots [3]509^{\wedge} [3]510^{\wedge} [3]511^{\wedge}$ $[2]0^{\wedge} [2]1^{\wedge} [2]2^{\wedge} [2]3^{\wedge} \dots [2]509^{\wedge} [2]510^{\wedge} [2]511^{\wedge}$ $[1]0^{\wedge} [1]1^{\wedge} [1]2^{\wedge} [1]3^{\wedge} \dots [1]509^{\wedge} [1]510^{\wedge} [1]511^{\wedge}$ $[0]0^{\wedge} [0]1^{\wedge} [0]2^{\wedge} [0]3^{\wedge} \dots [0]509^{\wedge} [0]510^{\wedge} [0]511^{\wedge}$
---	--

$CP2 =$ $[7]0^{\wedge} [7]1^{\wedge} [7]2^{\wedge} [7]3^{\wedge} \dots [7]509^{\wedge} [7]510^{\wedge} [7]511^{\wedge}$ $[6]0^{\wedge} [6]1^{\wedge} [6]2^{\wedge} [6]3^{\wedge} \dots [6]509^{\wedge} [6]510^{\wedge} [6]511^{\wedge}$ $[3]0^{\wedge} [3]1^{\wedge} [3]2^{\wedge} [3]3^{\wedge} \dots [3]509^{\wedge} [3]510^{\wedge} [3]511^{\wedge}$ $[2]0^{\wedge} [2]1^{\wedge} [2]2^{\wedge} [2]3^{\wedge} \dots [2]509^{\wedge} [2]510^{\wedge} [2]511^{\wedge}$	$NCP2 =$ $[5]0^{\wedge} [5]1^{\wedge} [5]2^{\wedge} [5]3^{\wedge} \dots [5]509^{\wedge} [5]510^{\wedge} [5]511^{\wedge}$ $[4]0^{\wedge} [4]1^{\wedge} [4]2^{\wedge} [4]3^{\wedge} \dots [4]509^{\wedge} [4]510^{\wedge} [4]511^{\wedge}$ $[1]0^{\wedge} [1]1^{\wedge} [1]2^{\wedge} [1]3^{\wedge} \dots [1]509^{\wedge} [1]510^{\wedge} [1]511^{\wedge}$ $[0]0^{\wedge} [0]1^{\wedge} [0]2^{\wedge} [0]3^{\wedge} \dots [0]509^{\wedge} [0]510^{\wedge} [0]511^{\wedge}$
---	--

$CP1 =$ $[7]0^{\wedge} [7]1^{\wedge} [7]2^{\wedge} [7]3^{\wedge} \dots [7]509^{\wedge} [7]510^{\wedge} [7]511^{\wedge}$ $[5]0^{\wedge} [5]1^{\wedge} [5]2^{\wedge} [5]3^{\wedge} \dots [5]509^{\wedge} [5]510^{\wedge} [5]511^{\wedge}$ $[3]0^{\wedge} [3]1^{\wedge} [3]2^{\wedge} [3]3^{\wedge} \dots [3]509^{\wedge} [3]510^{\wedge} [3]511^{\wedge}$ $[1]0^{\wedge} [1]1^{\wedge} [1]2^{\wedge} [1]3^{\wedge} \dots [1]509^{\wedge} [1]510^{\wedge} [1]511^{\wedge}$	$NCP1 =$ $[6]0^{\wedge} [6]1^{\wedge} [6]2^{\wedge} [6]3^{\wedge} \dots [6]509^{\wedge} [6]510^{\wedge} [6]511^{\wedge}$ $[4]0^{\wedge} [4]1^{\wedge} [4]2^{\wedge} [4]3^{\wedge} \dots [4]509^{\wedge} [4]510^{\wedge} [4]511^{\wedge}$ $[2]0^{\wedge} [2]1^{\wedge} [2]2^{\wedge} [2]3^{\wedge} \dots [2]509^{\wedge} [2]510^{\wedge} [2]511^{\wedge}$ $[0]0^{\wedge} [0]1^{\wedge} [0]2^{\wedge} [0]3^{\wedge} \dots [0]509^{\wedge} [0]510^{\wedge} [0]511^{\wedge}$
---	--

## — Row parity (RPn)

<p>RP9 =</p> <p>[7]511<sup>Λ</sup>[6]511<sup>Λ</sup>[5]511<sup>Λ</sup>[4]511<sup>Λ</sup>[3]511<sup>Λ</sup>[2]511<sup>Λ</sup>[1]511<sup>Λ</sup>[0]511<sup>Λ</sup>...</p> <p><sup>Λ</sup>[7]384<sup>Λ</sup>[6]384<sup>Λ</sup>[5]384<sup>Λ</sup>[4]384<sup>Λ</sup>[3]384<sup>Λ</sup>[2]384<sup>Λ</sup>[1]384<sup>Λ</sup>[0]384<sup>Λ</sup></p> <p><sup>Λ</sup>[7]383<sup>Λ</sup>[6]383<sup>Λ</sup>[5]383<sup>Λ</sup>[4]383<sup>Λ</sup>[3]383<sup>Λ</sup>[2]383<sup>Λ</sup>[1]383<sup>Λ</sup>[0]383<sup>Λ</sup>...</p> <p><sup>Λ</sup>[7]256<sup>Λ</sup>[6]256<sup>Λ</sup>[5]256<sup>Λ</sup>[4]256<sup>Λ</sup>[3]256<sup>Λ</sup>[2]256<sup>Λ</sup>[1]256<sup>Λ</sup>[0]256<sup>Λ</sup></p>	<p>NRP9 =</p> <p>[7]255<sup>Λ</sup>[6]255<sup>Λ</sup>[5]255<sup>Λ</sup>[4]255<sup>Λ</sup>[3]255<sup>Λ</sup>[2]255<sup>Λ</sup>[1]255<sup>Λ</sup>[0]255<sup>Λ</sup>...</p> <p><sup>Λ</sup>[7]128<sup>Λ</sup>[6]128<sup>Λ</sup>[5]128<sup>Λ</sup>[4]128<sup>Λ</sup>[3]128<sup>Λ</sup>[2]128<sup>Λ</sup>[1]128<sup>Λ</sup>[0]128<sup>Λ</sup></p> <p><sup>Λ</sup>[7]127<sup>Λ</sup>[6]127<sup>Λ</sup>[5]127<sup>Λ</sup>[4]127<sup>Λ</sup>[3]127<sup>Λ</sup>[2]127<sup>Λ</sup>[1]127<sup>Λ</sup>[0]127<sup>Λ</sup>...</p> <p><sup>Λ</sup>[7]0<sup>Λ</sup>[6]0<sup>Λ</sup>[5]0<sup>Λ</sup>[4]0<sup>Λ</sup>[3]0<sup>Λ</sup>[2]0<sup>Λ</sup>[1]0<sup>Λ</sup>[0]0</p>
--	---

<p>RP8 =</p> <p>[7]511<sup>Λ</sup>[6]511<sup>Λ</sup>[5]511<sup>Λ</sup>[4]511<sup>Λ</sup>[3]511<sup>Λ</sup>[2]511<sup>Λ</sup>[1]511<sup>Λ</sup>[0]511<sup>Λ</sup>...</p> <p><sup>Λ</sup>[7]384<sup>Λ</sup>[6]384<sup>Λ</sup>[5]384<sup>Λ</sup>[4]384<sup>Λ</sup>[3]384<sup>Λ</sup>[2]384<sup>Λ</sup>[1]384<sup>Λ</sup>[0]384<sup>Λ</sup></p> <p>[7]255<sup>Λ</sup>[6]255<sup>Λ</sup>[5]255<sup>Λ</sup>[4]255<sup>Λ</sup>[3]255<sup>Λ</sup>[2]255<sup>Λ</sup>[1]255<sup>Λ</sup>[0]255<sup>Λ</sup>...</p> <p><sup>Λ</sup>[7]128<sup>Λ</sup>[6]128<sup>Λ</sup>[5]128<sup>Λ</sup>[4]128<sup>Λ</sup>[3]128<sup>Λ</sup>[2]128<sup>Λ</sup>[1]128<sup>Λ</sup>[0]128<sup>Λ</sup></p>	<p>NRP8 =</p> <p>[7]383<sup>Λ</sup>[6]383<sup>Λ</sup>[5]383<sup>Λ</sup>[4]383<sup>Λ</sup>[3]383<sup>Λ</sup>[2]383<sup>Λ</sup>[1]383<sup>Λ</sup>[0]383<sup>Λ</sup>...</p> <p><sup>Λ</sup>[7]256<sup>Λ</sup>[6]256<sup>Λ</sup>[5]256<sup>Λ</sup>[4]256<sup>Λ</sup>[3]256<sup>Λ</sup>[2]256<sup>Λ</sup>[1]256<sup>Λ</sup>[0]256<sup>Λ</sup></p> <p><sup>Λ</sup>[7]127<sup>Λ</sup>[6]127<sup>Λ</sup>[5]127<sup>Λ</sup>[4]127<sup>Λ</sup>[3]127<sup>Λ</sup>[2]127<sup>Λ</sup>[1]127<sup>Λ</sup>[0]127<sup>Λ</sup>...</p> <p><sup>Λ</sup>[7]0<sup>Λ</sup>[6]0<sup>Λ</sup>[5]0<sup>Λ</sup>[4]0<sup>Λ</sup>[3]0<sup>Λ</sup>[2]0<sup>Λ</sup>[1]0<sup>Λ</sup>[0]0</p>
--	---

...

<p>RP1 =</p> <p>[7]511<sup>Λ</sup>[6]511<sup>Λ</sup>[5]511<sup>Λ</sup>[4]511<sup>Λ</sup>[3]511<sup>Λ</sup>[2]511<sup>Λ</sup>[1]511<sup>Λ</sup>[0]511<sup>Λ</sup></p> <p><sup>Λ</sup>[7]509<sup>Λ</sup>[6]509<sup>Λ</sup>[5]509<sup>Λ</sup>[4]509<sup>Λ</sup>[3]509<sup>Λ</sup>[2]509<sup>Λ</sup>[1]509<sup>Λ</sup>[0]509<sup>Λ</sup></p> <p><sup>Λ</sup>[7]507<sup>Λ</sup>[6]507<sup>Λ</sup>[5]507<sup>Λ</sup>[4]507<sup>Λ</sup>[3]507<sup>Λ</sup>[2]507<sup>Λ</sup>[1]507<sup>Λ</sup>[0]507<sup>Λ</sup>...</p> <p><sup>Λ</sup>[7]1<sup>Λ</sup>[6]1<sup>Λ</sup>[5]1<sup>Λ</sup>[4]1<sup>Λ</sup>[3]1<sup>Λ</sup>[2]1<sup>Λ</sup>[1]1<sup>Λ</sup>[0]1</p>	<p>NRP1 =</p> <p><sup>Λ</sup>[7]510<sup>Λ</sup>[6]510<sup>Λ</sup>[5]510<sup>Λ</sup>[4]510<sup>Λ</sup>[3]510<sup>Λ</sup>[2]510<sup>Λ</sup>[1]510<sup>Λ</sup>[0]510<sup>Λ</sup></p> <p><sup>Λ</sup>[7]508<sup>Λ</sup>[6]508<sup>Λ</sup>[5]508<sup>Λ</sup>[4]508<sup>Λ</sup>[3]508<sup>Λ</sup>[2]508<sup>Λ</sup>[1]508<sup>Λ</sup>[0]508<sup>Λ</sup></p> <p><sup>Λ</sup>[7]506<sup>Λ</sup>[6]506<sup>Λ</sup>[5]506<sup>Λ</sup>[4]506<sup>Λ</sup>[3]506<sup>Λ</sup>[2]506<sup>Λ</sup>[1]506<sup>Λ</sup>[0]506<sup>Λ</sup>...</p> <p><sup>Λ</sup>[7]0<sup>Λ</sup>[6]0<sup>Λ</sup>[5]0<sup>Λ</sup>[4]0<sup>Λ</sup>[3]0<sup>Λ</sup>[2]0<sup>Λ</sup>[1]0<sup>Λ</sup>[0]0</p>
---	--

## 3. Example of 4 bytes ECC (find 1bit error in the 4 bytes)

— Old data

	D7	D6	D5	D4	D3	D2	D1	D0
0th byte	0	0	0	0	0	0	0	0
1st byte	0	0	0	0	0	0	1	0
2nd byte	0	0	0	0	0	0	0	0
3rd byte	0	0	0	0	0	0	0	0

— New data

	D7	D6	D5	D4	D3	D2	D1	D0
0th byte	0	0	0	0	0	0	0	0
1st byte	0	0	1	0	0	0	1	0
2nd byte	0	0	0	0	0	0	0	0
3rd byte	0	0	0	0	0	0	0	0

— Column parity

	CP3	NCP3	CP2	NCP2	CP1	NCP1
Old data	0	1	0	1	1	0
New data	1	1	0	1	0	0
(Old ECC) ^ (New ECC)	1	0	0	0	1	0
Error code	1		0		1	
Result	5th bit					

— Row parity

	RP3	NRP3	RP2	NRP2
Old data	0	1	1	0
New data	0	0	0	0
(Old ECC) ^ (New ECC)	0	1	1	0
Error code	0		1	
Result	1st byte			

— Result: The 5th bit in the 1st byte is in error.



## PROGRAMMING NOR FLASH MEMORY

The SMDK2440 supports NOR flash control interface. There are two types of NOR flash memories in SMDK2440: AMD and Intel STRATA flash memory. The actual methods:

- Write image files to AMD flash memory with UART.
- Write image files to AMD flash memory with Multi-ICE.
- Write image files to AMD flash memory with OPENice32-A900
- Write image files to Intel STRATA flash memory with UART.
- Write image files to Intel STRATA flash memory with Multi-ICE.
- Write image files to Intel STRATA flash memory with OPENice32-A900

## WRITING IMAGE FILES TO AMD FLASH MEMORY WITH UART

1. Connect MULTI-ICE and execute "2440norom.ini" file.

```

ARM920T-D - Disassembly
30090080 [0xe2511001] sub    r1,r1,#1
30090084 [0x1afffffd] bne    0x30090080 ; (ENTER_PWDW + 0x30)
30090088 [0xe59f0080] ldr    r0,0x30090110 ; = #0x48000024
3009008c [0xe59d0080] str    r3,[r0,#0]
30090090 [0xe1adfd0e] mov    pc,r14
ENTER_SLE [0xe59f0074] ldr    r0,0x30090110 ; = #0x48000024
30090098 [0xe59d1000] ldr    r1,[r0,#0]
3009009c [0xe911840] orr    r1,r1,#0x400000
300900a0 [0xe58d1000] str    r1,[r0,#0]
300900a4 [0xe3a01010] mov    r1,#0x10
300900a8 [0xe2511001] sub    r1,r1,#1
300900ac [0x1afffffd] bne    0x300900a8 ; (ENTER_SLEEP + 0x14)
300900b0 [0xe59f1060] ldr    r1,0x30090118 ; = #0x56000080
300900b4 [0xe5910000] ldr    r0,[r1,#0]
300900b8 [0xe3000e0] orr    r0,r0,#0xe000
300900bc [0xe5810000] str    r0,[r1,#0]
300900c0 [0xe59f004c] ldr    r0,0x30090114 ; = #0x4c00000c
300900c4 [0xe58d2000] str    r2,[r0,#0]
300900c8 [0xeafffffe] b     0x300900c8 ; (ENTER_SLEEP + 0x34)
WAKEUP_SL [0xe59f1044] ldr    r1,0x30090118 ; = #0x56000080
300900d0 [0xe5910000] ldr    r0,[r1,#0]
300900d4 [0xe3c00e0] bic    r0,r0,#0xe000
300900d8 [0xe5810000] str    r0,[r1,#0]
300900dc [0xe59f0038] ldr    r0,0x3009011c ; = #0x30000400
300900e0 [0xe3a01448] mov    r1,#0x48000000
300900e4 [0xe2802034] add    r2,r0,#0x34
300900e8 [0xe490304] ldr    r3,[r0],#4
300900ec [0xe4813004] str    r3,[r1],#4
300900f0 [0xe1520000] cmp    r2,r0
300900f4 [0x1afffffb] bne    0x300900e8 ; (WAKEUP_SLEEP + 0x1)
300900f8 [0xe3a01f40] mov    r1,#0x100
300900fc [0xe2511001] sub    r1,r1,#1
30090100 [0x1afffffd] bne    0x300900fc ; (WAKEUP_SLEEP + 0x3)
30090104 [0xe59f1014] ldr    r1,0x30090120 ; = #0x56000088
30090108 [0xe5910000] ldr    r0,[r1,#0]

```

```

Command Line Interface
Command Line Interface
Debug > obey C:\WORK\2440\2440norom\2440norom.ini
Debug > com =====
Debug > com File name: 2440norom.ini
Debug > com 2003. 5. xx 1st draft.
Debug > com =====
Debug > com For S3C2440X
Debug > com SDRAM_Little_32, 64MB
Debug > com FCLK:101.25MHz UPIL:40MHz
Debug > com SDRAM refresh: 64ns(8Kcycle) -> 7.8us
Debug > swat {vector_catch 0x00
An expression could not be parsed or evaluated in the given context
Debug > swat {semhosting_enabled 0x00
An expression could not be parsed or evaluated in the given context
Debug > swat par %IFT_SVC
Invalid expression
Debug > com swat par %IFT_SVC32
Debug > com [disable Match-Dog reset]
Debug > swat *0x53000000 0
Debug > com << Clock setting >>
Debug > com [PLL lock time setting maximum]
Debug > swat *0x4c000000 ((0xffff<<12)+(0xffff<<0))
Debug > com FCLK:HCLK:FCLK=1:2:2.
Debug > swat *0x4c000014 ((0<<2)+(1<<1)+(0))
Debug > com [FCLK FMS setting:101.25MHz -> 0x7E,2,2]
Debug > swat *0x4c000004 ((0x7f<<12)+(0x2<<4)+(0x2<<0))
Debug > com [UCLK FMS setting:40MHz -> 0x78,2,3]
Debug > swat *0x4c000008 ((0x78<<12)+(0x2<<4)+(0x3<<0))
Debug > com << Memory setting >>
Debug > com [Bank6/7: 32-bit bus width]
Debug > swat *0x48000000 0x22000000
Debug > com [Bank0-5: Access cycle: 14-clocks, others:0-clock]
Debug > swat *0x48000004 ((0<<13)+{0<<11}+{7<<8}+{0<<6}+{0<<4}+{0<<2})
Debug > swat *0x48000008 ((0<<13)+{0<<11}+{7<<8}+{0<<6}+{0<<4}+{0<<2})
Debug > swat *0x4800000c ((0<<13)+{0<<11}+{7<<8}+{0<<6}+{0<<4}+{0<<2})

```

2. Load the image file (2440TEST.axf) to execute.

The screenshot displays the ARM920T\_0 - Disassembly window with the following assembly code:

```

30000000 [0xe0000074] b ResetHandler
30000004 [0xe0000052] b HandlerUndef
30000008 [0xe0000057] b HandlerSWI
3000000c [0xe0000062] b HandlerPabort
30000010 [0xe000005b] b HandlerDabort
30000014 [0xeaffffff] b 0x30000014
30000018 [0xe0000047] b HandlerIRQ
3000001c [0xe0000040] b HandlerFIQ
30000020 [0xe0000008] b EnterPWIN
30000024 [0x0f10e011] dcd 0x0f10e011 ....
30000028 [0x0000e380] dcd 0x0000e380 ....
3000002c [0x0f10e011] dcd 0x0f10e011 ....
30000030 [0xffffffff] dcd 0xffffffff ....
30000034 [0xffffffff] dcd 0xffffffff ....
30000038 [0xffffffff] dcd 0xffffffff ....
3000003c [0xffffffff] dcd 0xffffffff ....
30000040 [0xffffffff] dcd 0xffffffff ....
30000044 [0xe0000063] b ResetHandler
EnterPWIN[0xe1a02000] mov r2,r0
3000004c [0xe3100008] tst r0,#8
30000050 [0xe1a0000f] bne ENTER_SLEEP
30000054 [0xe59f00b4] ldr r0,0x30000110 ; = #0x40000024
30000058 [0xe5903000] ldr r3,[r0,#0]
3000005c [0xe1a01003] mov r1,r3
30000060 [0xe3811840] str r1,r1,#0x400000
30000064 [0xe5801000] str r1,[r0,#0]
30000068 [0xe3a01010] mov r1,#0x10
3000006c [0xe2511001] subs r1,r1,#1
30000070 [0xe1affffd] bne 0x3000006c ; (EnterPWIN + 0x24)
30000074 [0xe59f0098] ldr r0,0x30000114 ; = #0x4000000c
30000078 [0xe5802000] str r2,[r0,#0]
3000007c [0xe3a01020] mov r1,#0x20
30000080 [0xe2511001] subs r1,r1,#1
30000084 [0xe1affffd] bne 0x30000080 ; (EnterPWIN + 0x38)
30000088 [0xe59f0090] ldr r0,0x30000110 ; = #0x40000024
3000008c [0xe5803000] str r3,[r0,#0]
30000090 [0xe1a0f00e] mov pc,r14
  
```

The Command Line Interface window shows the following commands and their outputs:

```

Command Line Interface
An expression could not be parsed or evaluated in the given context
Debug >swat par \IFt_SVC
[Invalid expression]
Debug >com swat par \IF_SVC32
Debug >com [disable Watch-Dog reset]
Debug >swat *0x53000000 0
Debug >com << Clock setting >>
Debug >com [PLL lock time setting maximum]
Debug >swat *0x4c000000 ((0xffc12)+(0xfffc0))
Debug >com FCLK:UCLK:FCLK=1:2:2.
Debug >swat *0x4c000014 ((0c2)+(1cc1)+(0))
Debug >com [FCLK FMS setting:101.25MHz -> 0x7f,2,2]
Debug >swat *0x4c000004 ((0x7fcc12)+(0x2cc4)+(0x2cc0))
Debug >com [UCLK FMS setting:40MHz -> 0x78,2,3]
Debug >swat *0x4c000008 ((0x78c12)+(0x2cc4)+(0x3cc0))
Debug >com << Memory setting >>
Debug >com [Bank6/7: 32-bit bus width]
Debug >swat *0x48000000 0x22000000
Debug >com [Bank0-5: Access cycle: 14-clocks, others:0-clock]
Debug >swat *0x48000004 ((0c13)+(0c11)+(7cc8)+(0cc6)+(0cc4)+(0cc2)
Debug >swat *0x48000008 ((0c13)+(0c11)+(7cc8)+(0cc6)+(0cc4)+(0cc2)
Debug >swat *0x4800000c ((0c13)+(0c11)+(7cc8)+(0cc6)+(0cc4)+(0cc2)
Debug >swat *0x48000010 ((0c13)+(0c11)+(7cc8)+(0cc6)+(0cc4)+(0cc2)
Debug >swat *0x48000014 ((0c13)+(0c11)+(7cc8)+(0cc6)+(0cc4)+(0cc2)
Debug >swat *0x48000018 ((0c13)+(0c11)+(7cc8)+(0cc6)+(0cc4)+(0cc2)
Debug >com [Bank6/7: SIGAM, Tred:2clock, CA:9-bit]
Debug >swat *0x4800001c ((3c15)+(0c2)+1)
Debug >swat *0x48000020 ((3c15)+(0c2)+1)
Debug >com [SDRAM refresh enable, Trp=2clk, Trc=5clk, Refresh:1654]
Debug >swat *0x48000024 ((1c23)+(0c22)+(0c20)+(1c18)+1654)
Debug >com [SCE_EN enable, SCLK_EN enable, Bank6/7 memory map: 64KB]
Debug >swat *0x48000028 (0x1)+(1c5)+(1c4)
Debug >com [Bank6/7 Cl: 3-clocks]
Debug >swat *0x4800002c 0x30
Debug >swat *0x48000030 0x30
Debug >
  
```

## 3. Execute 2440TEST code with Go command.

```

123 // Hidden
124 #define NPFL_SEL    (0)
125 #define PCLK_SEL    (2)
126
127
128
129
130 //-----
131
132
133 void Main(void)
134 {
135     int i;
136     unsigned int apll_val;
137
138     Led_Display(0xf);
139
140     // HWF init. I/D cache on.
141     HWI_Init();
142
143 #if ADS10
144     __rt_isb_init(); //for ADS 1.0
145 #endif
146
147     // Clock setting
148     ChangeClockDivider(12,12); // 1:2:4
149     ChangeNF11Value(246,13,0); // 203.2MHz
150     //ChangeNF11Value(88,1,1) // 192MHz
151     ChangeUP11Value(0x3B,2,2); // 48MHz
152
153     // Clock calculation only for display information
154     Calc_Clock(0);
155     UPDATE_REFRESH(10clk);
156
157     // GPIO port init.
158     Port_Init();
159     // ISR init

```

```

Command Line Interface
Command Line Interface
An expression could not be parsed or evaluated in the given o
Debug >swat par %IFt_SVC
Invalid expression
Debug >com swat par %IF_SVC32
Debug >com [disable Watch-Dog reset]
Debug >swat *0x5300000 0
Debug >com << Clock setting >>
Debug >com [PLL lock time setting maximum]
Debug >swat *0x4c00000 (0xffff<<12)+(0xffff<<0)
Debug >com FCLK:HCLK:PCLK=1:2:2.
Debug >swat *0x4c00014 (0<<2)+(1<<1)+(0)
Debug >com [PCLK FMS setting:101.25MHz -> 0x7f,2,2]
Debug >swat *0x4c00004 (0x7f<<12)+(0x2<<4)+(0x2<<0)
Debug >com [UCLK FMS setting:48MHz -> 0x78,2,3]
Debug >swat *0x4c00008 (0x78<<12)+(0x2<<4)+(0x3<<0)
Debug >com << Memory setting >>
Debug >com [Bank6/7: 32-bit bus width]
Debug >swat *0x4800000 0x2200000
Debug >com [Bank0-5: Access cycles: 14-clocks, others:0-clock]
Debug >swat *0x4800004 (0<<13)+(0<<11)+(7<<8)+(0<<6)+(0<<4)-
Debug >swat *0x4800008 (0<<13)+(0<<11)+(7<<8)+(0<<6)+(0<<4)-
Debug >swat *0x480000c (0<<13)+(0<<11)+(7<<8)+(0<<6)+(0<<4)-
Debug >swat *0x4800010 (0<<13)+(0<<11)+(7<<8)+(0<<6)+(0<<4)-
Debug >swat *0x4800014 (0<<13)+(0<<11)+(7<<8)+(0<<6)+(0<<4)-
Debug >swat *0x4800018 (0<<13)+(0<<11)+(7<<8)+(0<<6)+(0<<4)-
Debug >com [Bank6/7: SDRAM, Trd:2clock, CA:9-bit]
Debug >swat *0x480001c (3<<15)+(0<<2)+1
Debug >swat *0x4800020 (3<<15)+(0<<2)+1
Debug >com [SDRAM refresh enable, Trp=2clk, Trc=5clk, Refresh
Debug >swat *0x4800024 ((1<<23)+(0<<22)+(0<<20)+(1<<18))+1654
Debug >com [SRCK_EN enable, SCLK_EN enable, Bank6/7 memory ma
Debug >swat *0x4800028 (0x1+(1<<5)+(1<<4))
Debug >com [Bank6/7 CL: 3-clocks]
Debug >swat *0x480002c 0x30
Debug >swat *0x4800030 0x30
Debug >

```

For Help, press F1 Running Image [|||||] Line 134, Col 0 Multi-ICE [ARM920T.0 2440test.scf]

4. Select "6:Program Flash" on the DNW.

**NOTE:** If you want to download 2440TEST.bin without MULTI-ICE, then skip the 1, 2 & 3 steps above and download 2440TEST.bin using the DNW (See EXECUTE 2440TEST WITHOUT MULTI-ICE).

After download 2440TEST.bin with the DNW, then you can also see the figure below.

```

DNW v0.50A [COM1,115200bps] [USB:OK]
Serial Port USB Port Configuration Help

[SMDK2440 Board Test Program Ver 0.0]

[Fclk:Hclk/Pclk]=[203.2:101.6:50.8]Mhz
[Uclk=48.0Mhz]

0:User Test      1:Manual Reg Set  2:PCMCIA test    3:Stepping stone
4:nWAIT test    5:Nand test       6:Program Flash  7:DMA test
8:Interrupt test 9:Cpu speed test 10:Power/Clk test 11:Lcd test
12:Camera test  13:SPI Test       14:IIC Test      15:RTC Test
16:IrDA Test    17:SD test        18:ADC test      19:ADC TS test
20:Timer test   21:IIS test       22:Uart Test     23:Clkdiv_Test

Select the function to test : 6

[ NOR Flash Memory Writer Ver 0.1 ]

The program buffer : 0x31000000 ~ 0x33ff0000
a : AM29LV800BB x1    b : 28F128J3A(16MB) x2
Select the type of a flash memory ? |
  
```

5. Select the type of memory as AM29LV800BB x1 (AMD Flash) by typing 'a'.
  6. Select whether you download through UART or MULTI-ICE.
- Type 'y' then you can download target files through UART. See the figure below.

```

DNW v0.50A [COM1,115200bps] [USB:OK]
Serial Port  USB Port  Configuration  Help
[SMDK2440 Board Test Program Ver 0.0]

[Fclk:Hclk/Pclk]=[203.2:101.6:50.8]Mhz
[Uclk=48.0Mhz]

 0:User Test          1:Manual Reg Set    2:PCMCIA test       3:Stepping stone
 4:nWAIT test        5:Nand test         6:Program Flash     7:DMA test
 8:Interrupt test    9:Cpu speed test   10:Power/C1k test   11:Lcd test
12:Camera test      13:SPI Test         14:IIC Test          15:RTC Test
16:IrDA Test        17:SD test          18:ADC test          19:ADC TS test
20:Timer test       21:IIS test         22:Uart Test         23:Clkdiv_Test

Select the function to test : 6

[ NOR Flash Memory Writer Ver 0.1 ]

The program buffer : 0x31000000 ~ 0x33ff0000
a : AM29LV800BB x1   b : 28F128J3A(16MB) x2
Select the type of a flash memory ? a
Do you want to download through UART0 from 0x31000000? [y/n] (y)

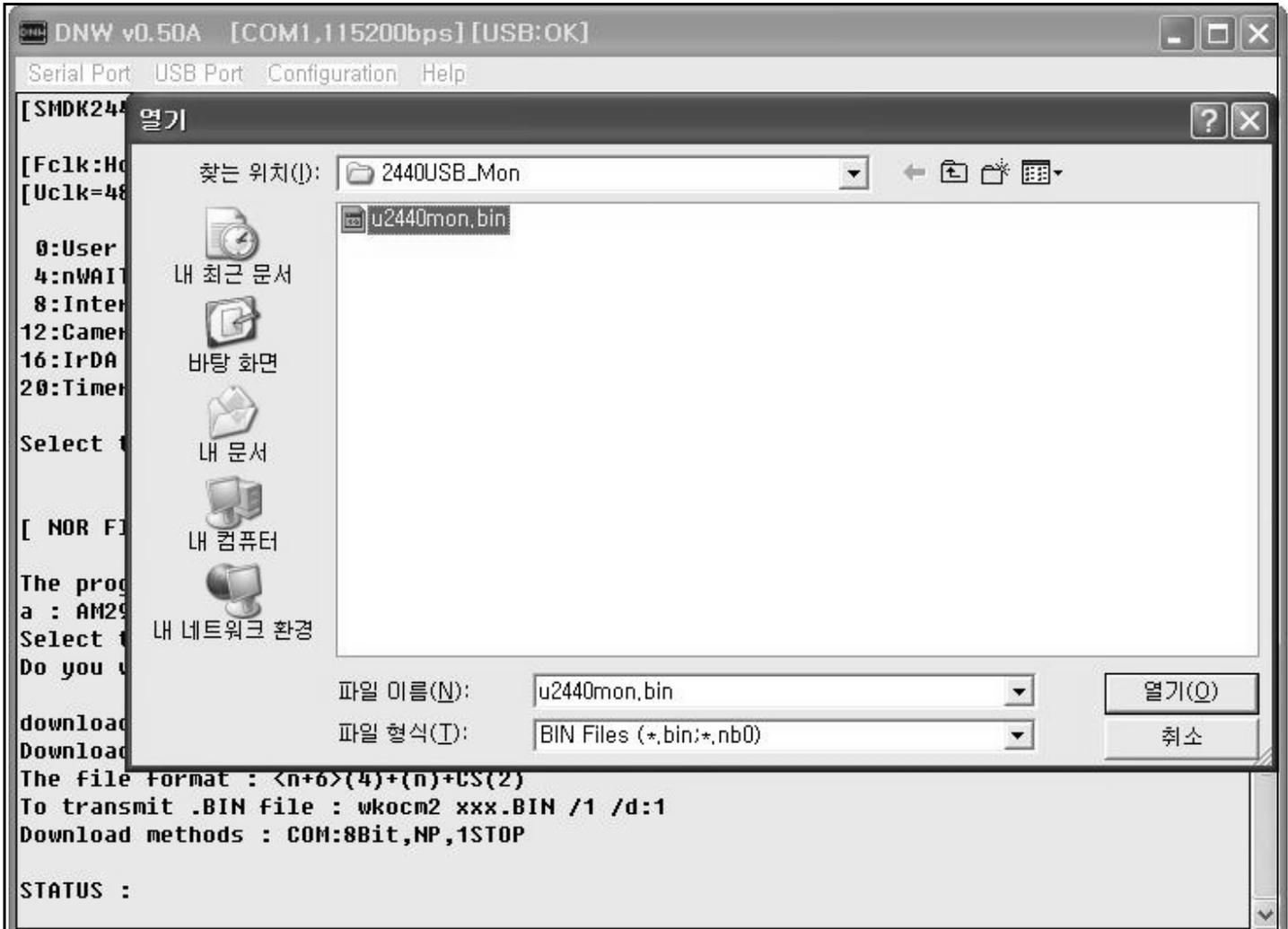
downloadAddress = 31000000
Download the plain binary file(.BHC) to be written
The file format : <n+6>(4)+(n)+CS(2)
To transmit .BIN file : wkocm2 xxx.BIN /1 /d:1
Download methods : COM:8Bit,NP,1STOP

STATUS :

```

7. Download target files with the DNW by selecting Transmit menu from Serial Port.

— Serial Port → Transmit



— Select and Download a target file.

```

DNW v0.50A [COM1,115200bps] [USB:OK]
Serial Port USB Port Configuration Help
Select the function to test : 6

[ NOR Flash Memory Writer Ver 0.1 ]

The program buffer : 0x31000000 ~ 0x33ff0000
a : AM29LV800BB x1      b : 28F128J3A(16MB) x2
Select the type of a flash memory ? a
Do you want to download through UART0 from 0x31000000? [y/n] : y

downloadAddress = 31000000
Download the plain binary file(.BHC) to be written
The file format : <n+6>(4)+(n)+CS(2)
To transmit .BIN file : wkocm2 xxx.BIN /1 /d:1
Download methods : COM:8Bit,NP,1STOP

STATUS : #####
Download O.K.
[AM29F800 Writing Program]

CAUTION: Check AM29LV800 BYTE#(47) pin is connected to VDD.

Source size:0h~8a54h

Available Target/Source Address:
    0x0, 0x4000, 0x6000, 0x8000,0x10000,0x20000,0x30000,0x40000,
    0x50000,0x60000,0x70000,0x80000,0x90000,0xa0000,0xb0000,0xc0000,
    0xd0000,0xe0000,0xf0000
Input source offset[0x0]:

```

8. Write input source offset and target offset and type 'y' repeatedly until the source size is reached. The example below shows how to setting source offset and target offset. The size of target file is 0x8a54 bytes.
  - Write source offset '0x0' and write target address '0x0', and then type 'y'.

```

DNW v0.50A [COM1,115200bps] [USB:OK]
Serial Port USB Port Configuration Help

Source size:0h~8a54h
Available Target/Source Address:
 0x0, 0x4000, 0x6000, 0x8000, 0x10000, 0x20000, 0x30000, 0x40000,
 0x50000, 0x60000, 0x70000, 0x80000, 0x90000, 0xa0000, 0xb0000, 0xc0000,
 0xd0000, 0xe0000, 0xf0000
Input source offset[0x0] 0x0
Input target address among above addresses[0x0] 0x0
source offset=0x0
target address=0x0
target block size=0x4000
[Check AH29LV800]
Manufacture ID= 1(0x0001), Device ID(0x225B)=225b

Erase the sector:0x0.
Sector Erase is started!

Start of the data writing.
0 1000 2000 3000
End of the data writing!!!

Verifying Start.
0 1000 2000 3000
Verifying End!!!
Do you want another programming without additional download? [y/n]
[AH29F800 Writing Program]

CAUTION: Check AH29LV800 BYTE0(A7) pin is connected to VDD.

Source size:0h~8a54h
Available Target/Source Address:
 0x0, 0x4000, 0x6000, 0x8000, 0x10000, 0x20000, 0x30000, 0x40000,
 0x50000, 0x60000, 0x70000, 0x80000, 0x90000, 0xa0000, 0xb0000, 0xc0000,
 0xd0000, 0xe0000, 0xf0000
Input source offset[0x4000] 0x4000
Input target address among above addresses[0x4000] 0x4000
source offset=0x4000
target address=0x4000
target block size=0x2000
[Check AH29LV800]
Manufacture ID= 1(0x0001), Device ID(0x225B)=225b

```

— Write source offset '0x4000' and write target address '0x4000', and then type 'y'.

- Write source offset '0x6000' and write target address '0x6000', and then type 'y'.

```

DNW v0.50A [COM1,115200bps][USB:OK]
Serial Port USB Port Configuration Help
Verifying Start.
0 1000
Verifying End!!!
Do you want another programming without additional download? [y/n]
[AM29F800 Writing Program]

CAUTION: Check AM29L0800 BYTE#(A7) pin is connected to VDD.

Source size:0h~8a54h

Available Target/Source Address:
0x0, 0x4000, 0x6000, 0x8000, 0x10000, 0x20000, 0x30000, 0x40000,
0x50000, 0x60000, 0x70000, 0x80000, 0x90000, 0xa0000, 0xb0000, 0xc0000,
0xd0000, 0xe0000, 0xf0000
Input source offset[0x6000]:0x6000
Input target address among above addresses[0x6000]:0x6000
source offset=0x6000
target address=0x6000
target block size=0x2000
[Check AM29L0800]
Manufacture ID= 1(0x0001), Device ID(0x2258)=225b

Erase the sector:0x6000.
Sector Erase is started!

Start of the data writing.
0 1000
End of the data writing!!!

Verifying Start.
0 1000
Verifying End!!!
Do you want another programming without additional download? [y/n]
[AM29F800 Writing Program]

CAUTION: Check AM29L0800 BYTE#(A7) pin is connected to VDD.

Source size:0h~8a54h

Available Target/Source Address:
0x0, 0x4000, 0x6000, 0x8000, 0x10000, 0x20000, 0x30000, 0x40000,
0x50000, 0x60000, 0x70000, 0x80000, 0x90000, 0xa0000, 0xb0000, 0xc0000,
0xd0000, 0xe0000, 0xf0000
Input source offset[0x8000]:

```

— Write source offset '0x8000' and write target address '0x8000', and then type 'n'.

```

DNW v0.50A [COM1,115200bps][USB:OK]
Serial Port USB Port Configuration Help
End of the data writing!!!
Verifying Start.
0 1000
Verifying End!!!
Do you want another programming without additional download? [y/n]
[AM29F800 Writing Program]
CAUTION: Check AM29LV800 BYTE#(47) pin is connected to UDD.
Source size:0h~8a54h
Available Target/Source Address:
0x0, 0x4000, 0x6000, 0x8000, 0x10000, 0x20000, 0x30000, 0x40000,
0x50000, 0x60000, 0x70000, 0x80000, 0x90000, 0xa0000, 0xb0000, 0xc0000,
0xd0000, 0xe0000, 0xf0000
Input source offset[0x8000] 0x8000
Input target address among above addresses[0x8000] 0x8000
Source offset=0x8000
target address=0x8000
target block size=0x8000
[Check AM29LV800]
Manufacture ID= 1(0x0001), Device ID(0x2258)=225b

Erase the sector:0x8000.
Sector Erase is started!

Start of the data writing.
0 1000 2000 3000 4000 5000 6000 7000
End of the data writing!!!

Verifying Start.
0 1000 2000 3000 4000 5000 6000 7000
Verifying End!!!
Do you want another programming without additional download? [y/n]

0:User Test      1:Manual Reg Set  2:PCMCIA test   3:Stepping stone
4:nWAIT test    5:Mand test      6:Progran Flash 7:DMA test
8:Interrupt test 9:Cpu speed test 10:Power/Clk test 11:Lcd test
12:Camera test  13:SPI Test      14:IIC Test     15:RTC Test
16:IrDA Test    17:SD test       18:ADC test     19:ADC TS test
20:Tiner test   21:IIS test      22:Uart Test    23:Clkdiv_Test

Select the function to test :

```

9. Turn the SMDK2440 off and then on.

## WRITING IMAGE FILES TO AMD FLASH MEMORY WITH MULTI-ICE

1. Connect MULTI-ICE and execute "2440norom.ini" file.

```

ARM200T_0 - Disassembly
30000080 [0xe2511001] sub    r1,r1,#1
30000084 [0x1afffffd] bne    0x30000080 ; (ENTER_PMIW + 0x30)
30000088 [0xe59f0080] ldr    r0,0x30000110 ; = #0x48000024
3000008c [0xe5803000] str    r3,[r0,#0]
30000090 [0xe1a0f00e] mov    pc,r14
ENTER_SLEEP [0xe59f0074] ldr    r0,0x30000110 ; = #0x48000024
30000098 [0xe59d1000] ldr    r1,[r0,#0]
3000009c [0xe3811840] orr    r1,r1,#0x400000
300000a0 [0xe58d1000] str    r1,[r0,#0]
300000a4 [0xe3a01010] mov    r1,#0x10
300000a8 [0xe2511001] sub    r1,r1,#1
300000ac [0x1afffffd] bne    0x300000a8 ; (ENTER_SLEEP + 0x14)
300000b0 [0xe59f1060] ldr    r1,0x30000118 ; = #0x56000080
300000b4 [0xe5910000] ldr    r0,[r1,#0]
300000b8 [0xe3000e00] orr    r0,r0,#0xe000
300000bc [0xe5810000] str    r0,[r1,#0]
300000c0 [0xe59f004c] ldr    r0,0x30000114 ; = #0x4c00000c
300000c4 [0xe58d2000] str    r2,[r0,#0]
300000c8 [0xeafffffc] b     0x300000c8 ; (ENTER_SLEEP + 0x34)
WAKEUP_SLEEP [0xe59f1044] ldr    r1,0x30000118 ; = #0x56000080
300000d0 [0xe5910000] ldr    r0,[r1,#0]
300000d4 [0xe3c00e00] bic    r0,r0,#0xe000
300000d8 [0xe5810000] str    r0,[r1,#0]
300000dc [0xe59f0038] ldr    r0,0x3000011c ; = #0x30000408
300000e0 [0xe3a01448] mov    r1,#0x48000000
300000e4 [0xe2802034] add    r2,r0,#0x34
300000e8 [0xe4903004] ldr    r3,[r0],#4
300000ec [0xe4813004] str    r3,[r1],#4
300000f0 [0xe1520000] cmp    r2,r0
300000f4 [0x1afffffb] bne    0x300000e8 ; (WAKEUP_SLEEP + 0x1)
300000f8 [0xe3a01e40] mov    r1,#0x100
300000fc [0xe2511001] sub    r1,r1,#1
30000100 [0x1afffffd] bne    0x300000fc ; (WAKEUP_SLEEP + 0x3)
30000104 [0xe59f1014] ldr    r1,0x30000120 ; = #0x56000088
30000108 [0xe5910000] ldr    r0,[r1,#0]

```

```

Command Line Interface
Command Line interface
Debug > obey C:\WORK\2440\2440norom\2440norom.ini
Debug > com =====
Debug > com filename: 2440norom.ini
Debug > com 2003. 5. xx 1st draft.
Debug > com =====
Debug > com For S3C2440X
Debug > com SDRAM_Little_32, 64MB
Debug > com FCLK:101.25MHz UPIL:48MHz
Debug > com SDRAM refresh: 64ns(8Kcycle) -> 7.8us
Debug > swat $vector_catch 0x00
An expression could not be parsed or evaluated in the given context
Debug > swat $seashosting_enabled 0x00
An expression could not be parsed or evaluated in the given context
Debug > swat ptr %IFt_3WC
Invalid expression
Debug > com swat ptr %IF_3WC32
Debug > com [disable Watch-Dog reset]
Debug > swat *0x53000000 0
Debug > com << Clock setting >>
Debug > com [PLL lock time setting maximum]
Debug > swat *0x4c000000 ((0xffff<<12)+0xffff<<0))
Debug > com FCLK:HCLK:FCLK-1:2:2.
Debug > swat *0x4c000014 ((0<<2)+(1<<1)+0))
Debug > com [FCLK FMS setting:101.25MHz -> 0x7f,2,2]
Debug > swat *0x4c000004 ((0x7f<<12)+(0x2<<4)+(0x2<<0))
Debug > com [UCLK FMS setting:48MHz -> 0x78,2,3]
Debug > swat *0x4c000008 ((0x78<<12)+(0x2<<4)+(0x3<<0))
Debug > com << Memory setting >>
Debug > com [Bank6/7: 32-bit bus width]
Debug > swat *0x48000000 0x22000000
Debug > com [Bank0-5: Access cycle: 14-clocks, others:0-clock]
Debug > swat *0x48000004 ((0<<13)+(0<<11)+(7<<8)+(0<<6)+(0<<4)+(0<<2))
Debug > swat *0x48000008 ((0<<13)+(0<<11)+(7<<8)+(0<<6)+(0<<4)+(0<<2))
Debug > swat *0x4800000c ((0<<13)+(0<<11)+(7<<8)+(0<<6)+(0<<4)+(0<<2))

```

2. Load the image file (2440TEST.axf) to execute.

The screenshot displays the ARM920T\_0 - Disassembly window with the following assembly code:

```

30000000 [0xe0000074] b      ResetHandler
30000004 [0xe0000052] b      HandlerUndef
30000008 [0xe0000057] b      HandlerSWI
3000000c [0xe0000062] b      HandlerPabort
30000010 [0xe000005b] b      HandlerDabort
30000014 [0xeaffffff] b      0x30000014
30000018 [0xe0000047] b      HandlerIRQ
3000001c [0xe0000040] b      HandlerFIQ
30000020 [0xe0000008] b      EnterPWIN
30000024 [0x0f10e011] dcd    0x0f10e011 ....
30000028 [0x0080e380] dcd    0x0080e380 ....
3000002c [0x0f10e011] dcd    0x0f10e011 ....
30000030 [0xffffffff] dcd    0xffffffff ....
30000034 [0xffffffff] dcd    0xffffffff ....
30000038 [0xffffffff] dcd    0xffffffff ....
3000003c [0xffffffff] dcd    0xffffffff ....
30000040 [0xffffffff] dcd    0xffffffff ....
30000044 [0xe0000063] b      ResetHandler
EnterPWIN[0xe1a02000] mov    r2,r0
3000004c [0xe3100008] tst    r0,#8
30000050 [0xe1a0000f] bne   ENTER_SLEEP
30000054 [0xe59f00b4] ldr    r0,0x30000110 ; = #0x40000024
30000058 [0xe5903000] ldr    r3,[r0,#0]
3000005c [0xe1a01003] mov    r1,r3
30000060 [0xe3811840] str    r1,r1,#0x400000
30000064 [0xe5801000] str    r1,[r0,#0]
30000068 [0xe3a01010] mov    r1,#0x10
3000006c [0xe2511001] subs  r1,r1,#1
30000070 [0xe1affffd] bne   0x3000006c ; (EnterPWIN + 0x24)
30000074 [0xe59f0098] ldr    r0,0x30000114 ; = #0x4000000c
30000078 [0xe5802000] str    r2,[r0,#0]
3000007c [0xe3a01020] mov    r1,#0x20
30000080 [0xe2511001] subs  r1,r1,#1
30000084 [0xe1affffd] bne   0x30000080 ; (EnterPWIN + 0x38)
30000088 [0xe59f0090] ldr    r0,0x30000110 ; = #0x40000024
3000008c [0xe5803000] str    r3,[r0,#0]
30000090 [0xe1a0f00e] mov    pc,r14

```

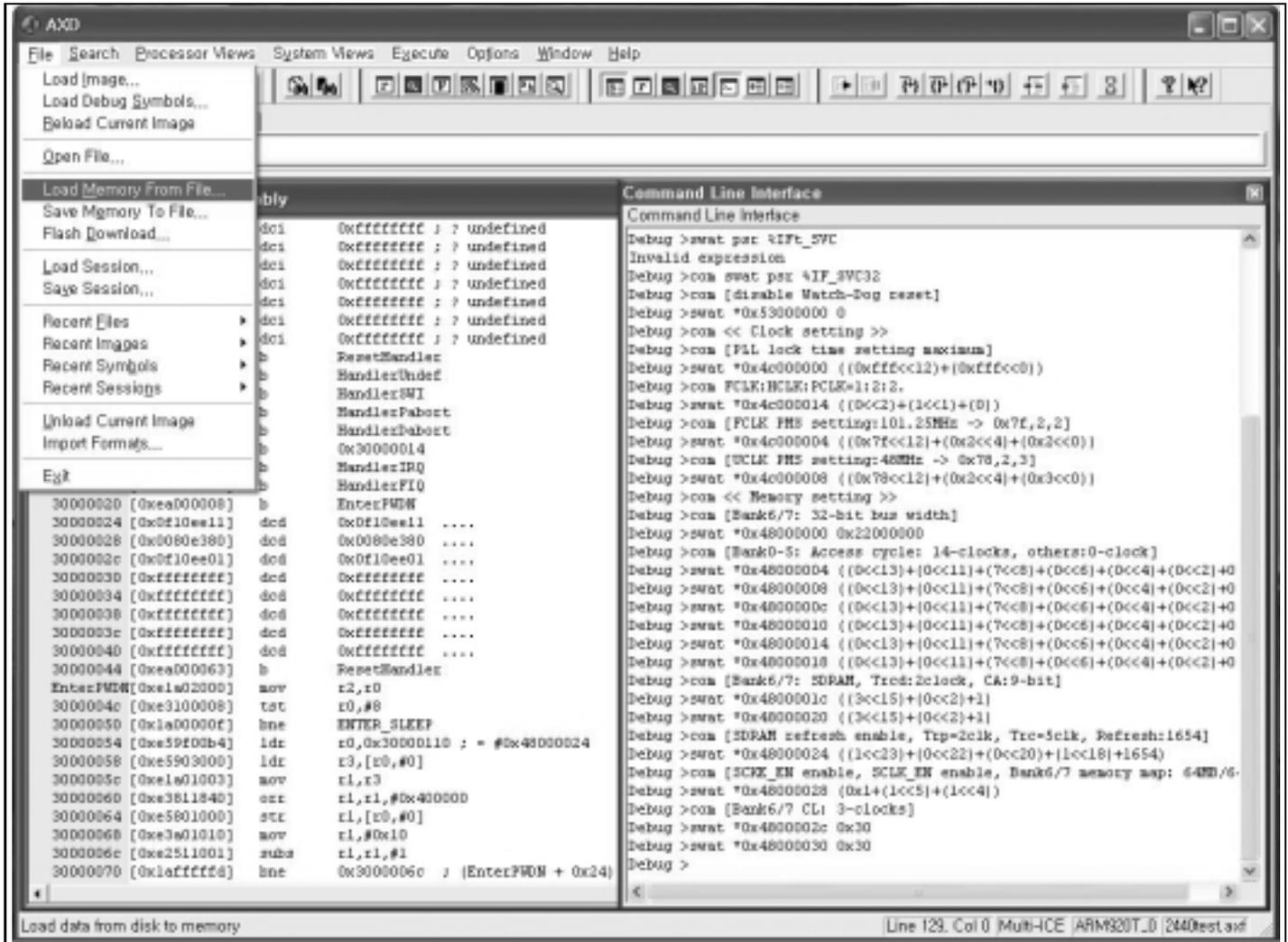
The Command Line Interface window shows the following commands and their outputs:

```

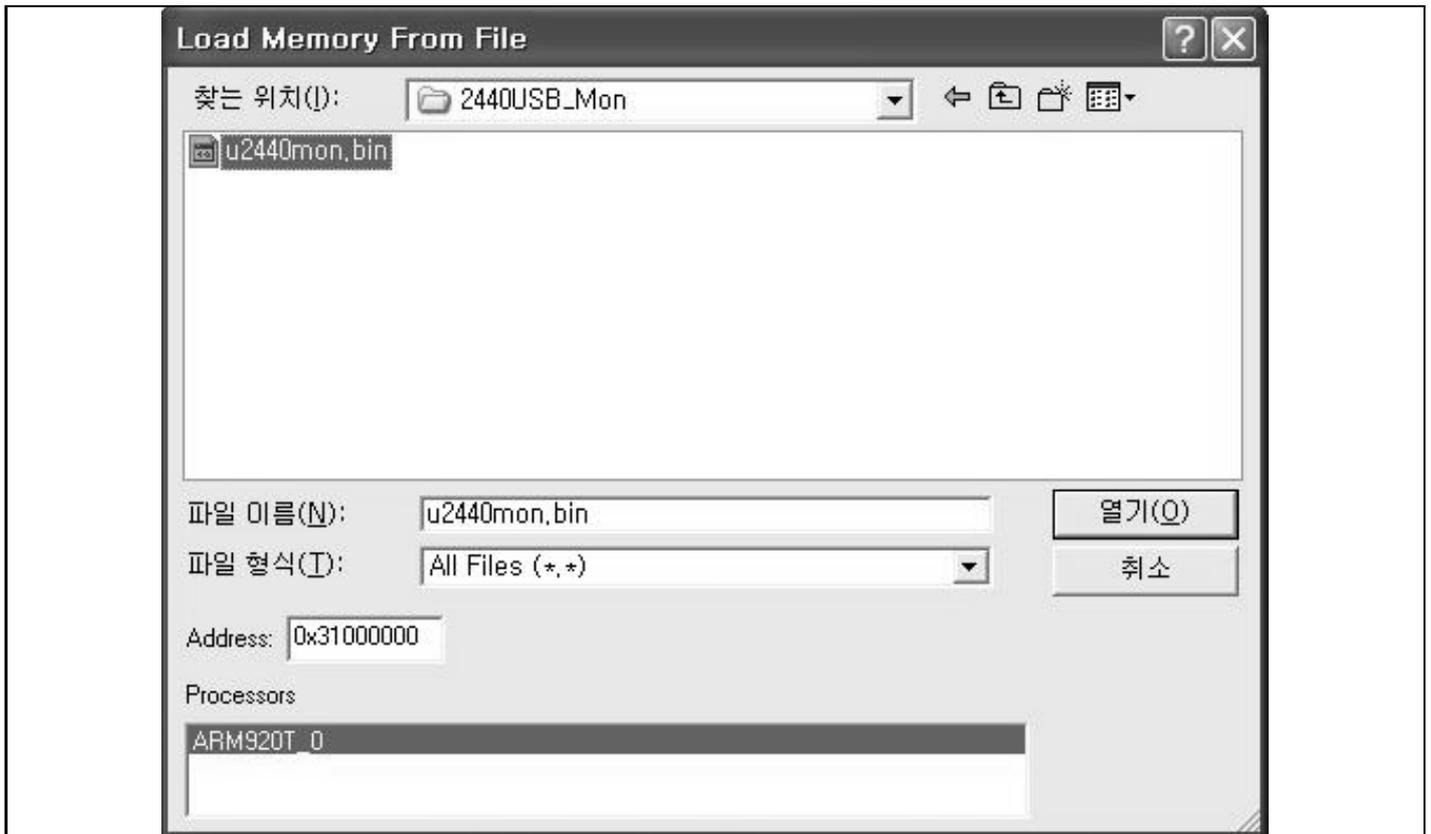
Command Line Interface
An expression could not be parsed or evaluated in the given context
Debug >swat par vif_SVC
[Invalid expression]
Debug >com swat par vif_SVC32
Debug >com [disable Watch-Dog reset]
Debug >swat *0x53000000 0
Debug >com << Clock setting >>
Debug >com [PLL lock time setting maximum]
Debug >swat *0x4c000000 ((0xffc12)+(0xffc0))
Debug >com FCLK:MCLK:FCLK=1:2:2.
Debug >swat *0x4c000014 ((0c2)+(1cc1)+(0))
Debug >com [FCLK PMS setting:101.25MHz -> 0x7f,2,2]
Debug >swat *0x4c000004 ((0x7fcc12)+(0x2cc4)+(0x2cc0))
Debug >com [UCLK PMS setting:40MHz -> 0x78,2,3]
Debug >swat *0x4c000008 ((0x78c12)+(0x2cc4)+(0x3cc0))
Debug >com << Memory setting >>
Debug >com [Bank6/7: 32-bit bus width]
Debug >swat *0x48000000 0x22000000
Debug >com [Bank0-5: Access cycle: 14-clocks, others:0-clock]
Debug >swat *0x48000004 ((0c13)+(0c11)+(7cc8)+(0cc6)+(0cc4)+(0cc2)
Debug >swat *0x48000008 ((0c13)+(0c11)+(7cc8)+(0cc6)+(0cc4)+(0cc2)
Debug >swat *0x4800000c ((0c13)+(0c11)+(7cc8)+(0cc6)+(0cc4)+(0cc2)
Debug >swat *0x48000010 ((0c13)+(0c11)+(7cc8)+(0cc6)+(0cc4)+(0cc2)
Debug >swat *0x48000014 ((0c13)+(0c11)+(7cc8)+(0cc6)+(0cc4)+(0cc2)
Debug >swat *0x48000018 ((0c13)+(0c11)+(7cc8)+(0cc6)+(0cc4)+(0cc2)
Debug >com [Bank6/7: SIGAM, Tred:2clock, CA:9-bit]
Debug >swat *0x4800001c ((3c15)+(0c2)+1)
Debug >swat *0x48000020 ((3c15)+(0c2)+1)
Debug >com [SDRAM refresh enable, Trp=2clk, Trc=5clk, Refresh:1654]
Debug >swat *0x48000024 ((1c23)+(0c22)+(0c20)+(1c18)+1654)
Debug >com [SCEN enable, SCLK_EN enable, Bank6/7 memory map: 64KB]
Debug >swat *0x48000028 (0x1+(1c5)+(1c4))
Debug >com [Bank6/7 Cl: 3-clocks]
Debug >swat *0x4800002c 0x30
Debug >swat *0x48000030 0x30
Debug >

```

## 3. Select Load Memory From File... on the file menu of AXD.



- Get a target file to 0x31000000 in SMDK2440 Board.



## 4. Execute 2440TEST.axf file with GO command.

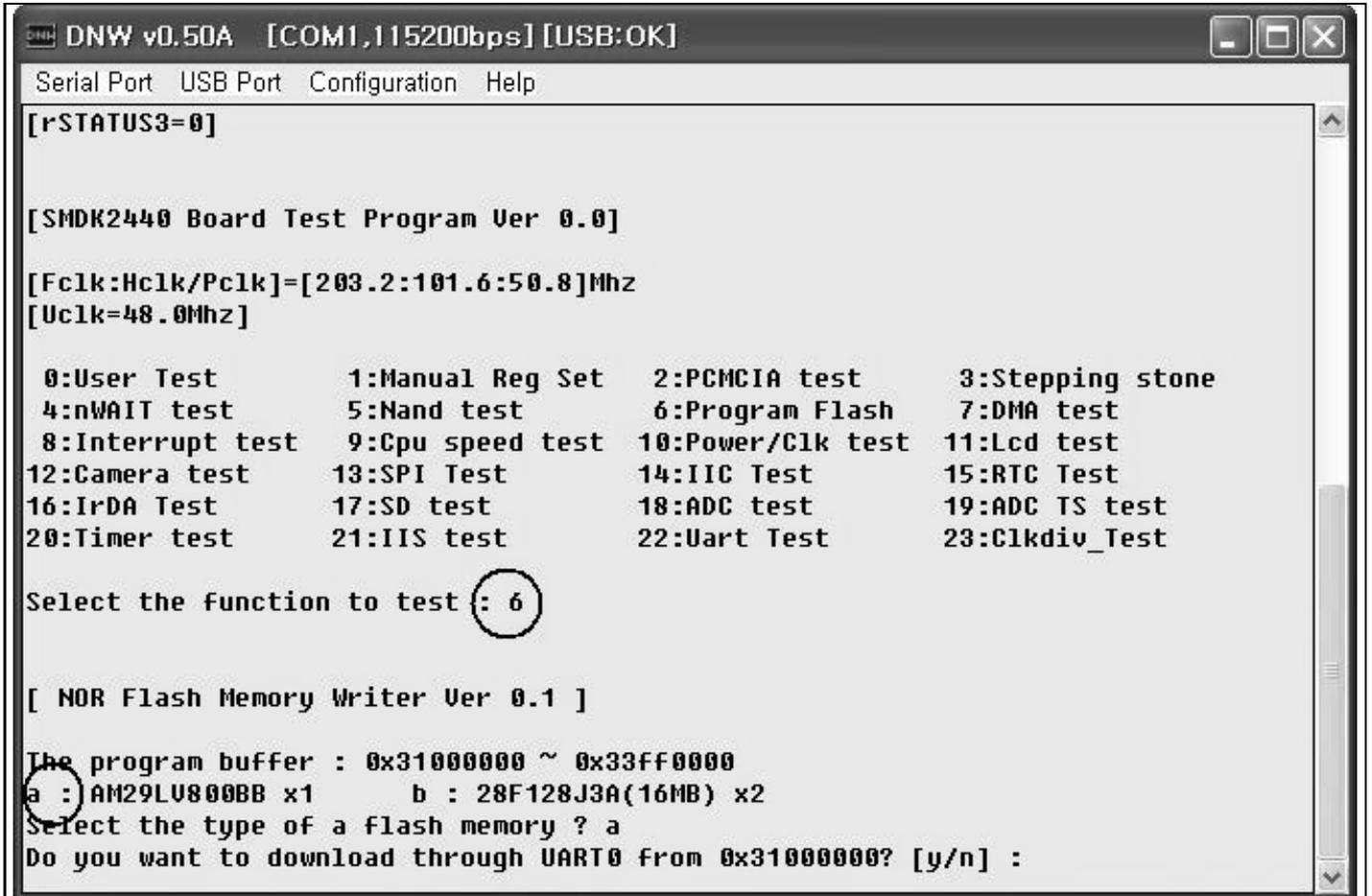
The screenshot displays the AXD debugger interface. The left pane shows the disassembly of the ARM920T\_D target, with the 'Main' function selected. The right pane shows the Command Line Interface (CLI) with various debug commands being entered to configure the system and execute the 2440TEST.axf file.

```

ARM920T_D - Disassembly
3000548 [0xe3e0000] mov     r0,#0
300054c [0xe3a0144a] mov     r1,#0x4a000000
3000550 [0xe5810008] str     r0,[r1,#8]
3000554 [0xe1e00aa0] mov     r0,r0,ldr #21
3000558 [0xe3a0144a] mov     r1,#0x4a000000
300055c [0xe581001c] str     r0,[r1,#0x1c]
3000560 [0xe1a0e00e] mov     pc,r14
Main [0xe2d4070] * staff
3000568 [0xe24d020] sub     r13,r13,#0x28
300056c [0xe3a0000f] mov     r0,#0xf
3000570 [0xeb000354] bl     Led_Display
3000574 [0xeb00052a] bl     MMU_Init
3000578 [0xeb0096e0] bl     __rt_lib_init
300057c [0xe3a0100c] mov     r1,#0xc
3000580 [0xe3a0000c] mov     r0,#0xc
3000584 [0xeb000376] bl     ChangeClockDivider
3000588 [0xe3a02000] mov     r2,#0
300058c [0xe3a01004] mov     r1,#0xd
3000590 [0xe3a000e6] mov     r0,#0xf6
3000594 [0xeb00036c] bl     ChangeMPLLValue
3000598 [0xe3a02002] mov     r2,#2
300059c [0xe3a01002] mov     r1,#2
30005a0 [0xe3a00038] mov     r0,#0x38
30005a4 [0xeb000396] bl     ChangeUPLLValue
30005a8 [0xe3a00000] mov     r0,#0
30005ac [0xeb001ef1] bl     Calc_Clock
30005b0 [0xe59f0268] ldr     r0,0x30000820 ; = #0x30007090
30005b4 [0xe5900000] ldr     r0,[r0,#0]
30005b8 [0xeb00af0b] bl     _ffits
30005bc [0xe1a05000] mov     r5,r0
30005c0 [0xeb00edf9] bl     _f34
30005c4 [0xe5840010] str     r0,[r13,#0x10]
30005c8 [0xe5841014] str     r1,[r13,#0x14]
30005cc [0xe28f0f94] add     r0,pc,#0x250 ; #0x30000824
30005c0 [0xe890000c] ldmia  r0,[r2,r3]
30005d4 [0xe5940010] ldr     r0,[r13,#0x10]
Command Line Interface
Debug >swat par \VFT_3VC
Invalid expression
Debug >com swat par \VIF_3VC32
Debug >com [disable Watch-Dog reset]
Debug >swat *0x53000000 0
Debug >com << Clock setting >>
Debug >com [PLL lock time setting maximum]
Debug >swat *0x4c000000 [(0xff<<12)+(0xfff<<0)]
Debug >com FCLK:MCLK:FCLK=1:2:2.
Debug >swat *0x4c000014 [(0xc2)+(1cc1)+(0)]
Debug >com [FCLK FMS setting:101.25MHz -> 0x7f,2,2]
Debug >swat *0x4c000004 [(0x7f<<12)+(0x2cc4)+(0x2cc0)]
Debug >com [UCLK FMS setting:40MHz -> 0x70,2,3]
Debug >swat *0x4c000008 [(0x70<<12)+(0x2cc4)+(0x3cc0)]
Debug >com << Memory setting >>
Debug >com [Bank6/7: 32-bit bus width]
Debug >swat *0x48000000 0x22000000
Debug >com [Bank0-5: Access cycle: 14-clocks, others:0-clock]
Debug >swat *0x48000004 [(0<<13)+(0<<11)+(7<<8)+(0<<6)+(0c4)+(0c2)+
Debug >swat *0x48000008 [(0<<13)+(0<<11)+(7<<8)+(0<<6)+(0c4)+(0c2)+
Debug >swat *0x4800000c [(0<<13)+(0<<11)+(7<<8)+(0<<6)+(0c4)+(0c2)+
Debug >swat *0x48000010 [(0<<13)+(0<<11)+(7<<8)+(0<<6)+(0c4)+(0c2)+
Debug >swat *0x48000014 [(0<<13)+(0<<11)+(7<<8)+(0<<6)+(0c4)+(0c2)+
Debug >swat *0x48000018 [(0<<13)+(0<<11)+(7<<8)+(0<<6)+(0c4)+(0c2)+
Debug >com [Bank6/7: SIGRAM, Tred:2clock, CA:3-bit]
Debug >swat *0x4800001c [(3<<15)+(0<<2)+1]
Debug >swat *0x48000020 [(3<<15)+(0<<2)+1]
Debug >com [SDRAM refresh enable, Trp=2clk, Txe=5clk, Refresh:1654]
Debug >swat *0x48000024 [(1<<23)+(0<<22)+(0c20)+(1<<18)+1654]
Debug >com [SCE_EN enable, SCLK_EN enable, Bank6/7 memory map: 64MB]
Debug >swat *0x48000028 0x1+(1<<5)+(1<<4)
Debug >com [Bank6/7 Cl: 3-clocks]
Debug >swat *0x4800002c 0x30
Debug >swat *0x48000030 0x30
Debug >
  
```

5. Select "6:Program Flash" on the DNW.

**NOTE:** If you want to download 2440TEST.bin without MULTI-ICE, then skip the 1, 2 & 3 steps above and download 2440TEST.bin using the DNW (See EXECUTE 2440TEST WITHOUT MULTI-ICE).  
After downloading 2440TEST.bin with the DNW, then you can also see the figure below.



```

DNW v0.50A [COM1,115200bps] [USB:OK]
Serial Port  USB Port  Configuration  Help
[rSTATUS3=0]

[SMDK2440 Board Test Program Ver 0.0]

[Fclk:Hclk/Pclk]=[203.2:101.6:50.8]Mhz
[Uclk=48.0Mhz]

 0:User Test      1:Manual Reg Set  2:PCMCIA test    3:Stepping stone
 4:nWAIT test    5:Nand test      6:Program Flash  7:DMA test
 8:Interrupt test 9:Cpu speed test 10:Power/Clk test 11:Lcd test
12:Camera test   13:SPI Test      14:IIC Test      15:RTC Test
16:IrDA Test     17:SD test       18:ADC test      19:ADC TS test
20:Timer test    21:IIS test      22:Uart Test     23:Clkdiv_Test

Select the function to test : 6

[ NOR Flash Memory Writer Ver 0.1 ]

The program buffer : 0x31000000 ~ 0x33FF0000
a : AM29LV800BB x1    b : 28F128J3A(16MB) x2
Select the type of a flash memory ? a
Do you want to download through UART0 from 0x31000000? [y/n] :

```

6. Select the type of memory as AM29LV800BB (AMD) by typing 'a'.

7. Select whether you download through UART0 or MULTI-ICE.  
 — Type 'n' then you can see the figure below in the DNW.

```

DNW v0.50A [COM1,115200bps] [USB:OK]
Serial Port  USB Port  Configuration  Help
12:Camera test    13:SPI Test    14:IIC Test    15:RTC Test
16:IrDA Test     17:SD test    18:ADC test    19:ADC TS test
20:Timer test    21:IIS test    22:Uart Test   23:Clkdiv_Test

Select the function to test : 6

[ NOR Flash Memory Writer Ver 0.1 ]

The program buffer : 0x31000000 ~ 0x33ff0000
a : AM29LV800BB x1    b : 28F128J3A(16MB) x2
Select the type of a flash memory ? a
Do you want to download through UART0 from 0x31000000? [y/n] : n
[AM29F800 Writing Program]

CAUTION: Check AM29LV800 BYTE#(47) pin is connected to VDD.

Source size:0h~0h

Available Target/Source Address:
    0x0, 0x4000, 0x6000, 0x8000,0x10000,0x20000,0x30000,0x40000,
    0x50000,0x60000,0x70000,0x80000,0x90000,0xa0000,0xb0000,0xc0000,
    0xd0000,0xe0000,0xf0000
Input source offset[0x0]:
  
```

8. Write input source offset and target offset and type 'y' repeatedly until the source size is reached.  
See Writing the image file to AMD Flash memory with UART.

```

DNW v0.50A [COM1,115200bps] [USB:OK]
Serial Port USB Port Configuration Help

Verifying Start.
0 1000
Verifying End!!!
Do you want another programming without additional download? [y/n]
[AH29F800 Writing Program]

CAUTION: Check AH29LU800 BYTE#(47) pin is connected to U00.

Source size:0h~0h

Available Target/Source Address:
0x0, 0x4000, 0x6000, 0x8000, 0x10000, 0x20000, 0x30000, 0x40000,
0x50000, 0x60000, 0x70000, 0x80000, 0x90000, 0xa0000, 0xb0000, 0xc0000,
0xd0000, 0xe0000, 0xf0000
Input source offset[0x8000]:0x8000
Input target address among address[0x8000]:0x8000
source offset=0x8000
target address=0x8000
target block size=0x8000
The data must be downloaded using ICE from 31000000
[Check AH29LU800]
Manufacture ID= 1(0x0001), Device ID(0x225B)=225b

Erase the sector:0x8000.
Sector Erase is started!

Start of the data writing.
0 1000 2000 3000 4000 5000 6000 7000
End of the data writing!!!

Verifying Start.
0 1000 2000 3000 4000 5000 6000 7000
Verifying End!!!
Do you want another programming without additional download? [y/n]

0:User Test      1:Manual Reg Set  2:PCMCIA test    3:Stepping stone
4:UART test     5:NAND test       6:Program Flash  7:DMA test
8:Interrupt test 9:CPU speed test 10:Power/Clk test 11:LCD test
12:Camera test  13:SPI Test       14:IIC Test      15:RTC Test
16:IrDA Test    17:SD test        18:ADC test      19:ADC IS test
20:Timer test   21:IIS test       22:UART Test     23:Clkdiv_Test

Select the function to test :

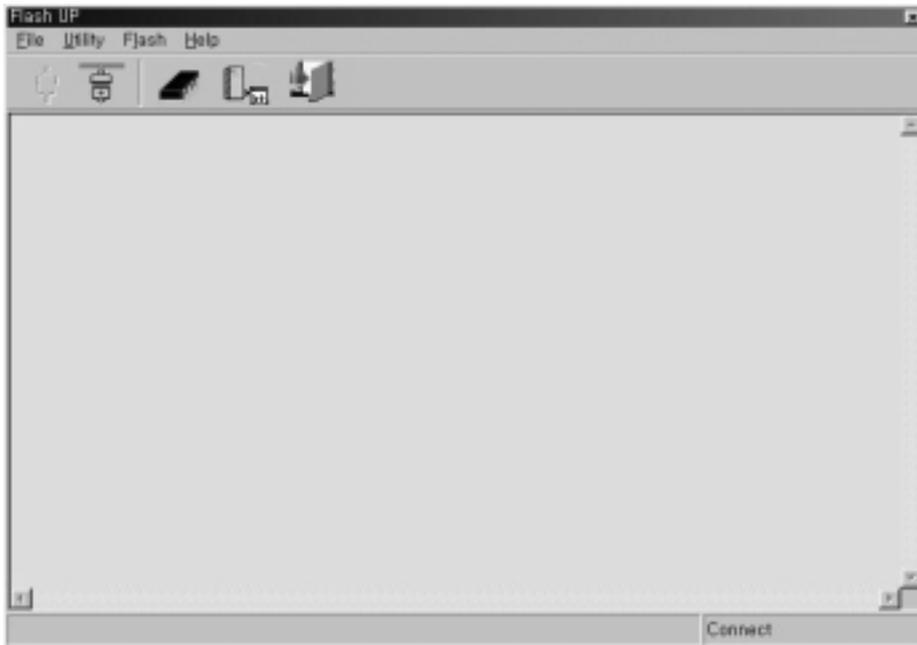
```

9. Turn off and on the SMDK2440.

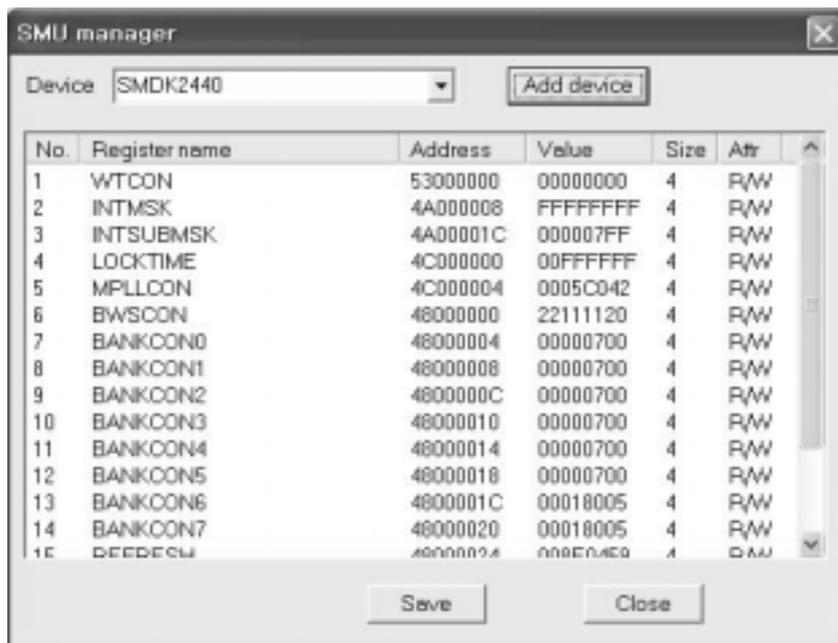
## WRITING IMAGE FILES TO AMD FLASH MEMORY WITH OPENICE32-A900

OPENice32-A900 can write image to AMD Flash memory as Multi-ICE. However, OPENice32-A900 provide a Flash Write Program that is easy to use and don't require ARM SDT/ADS debugger nor DNW. For more information on the program, refer to OPENice32-A900 manual or contact AIJI System ([www.aijisystem.com](http://www.aijisystem.com)).

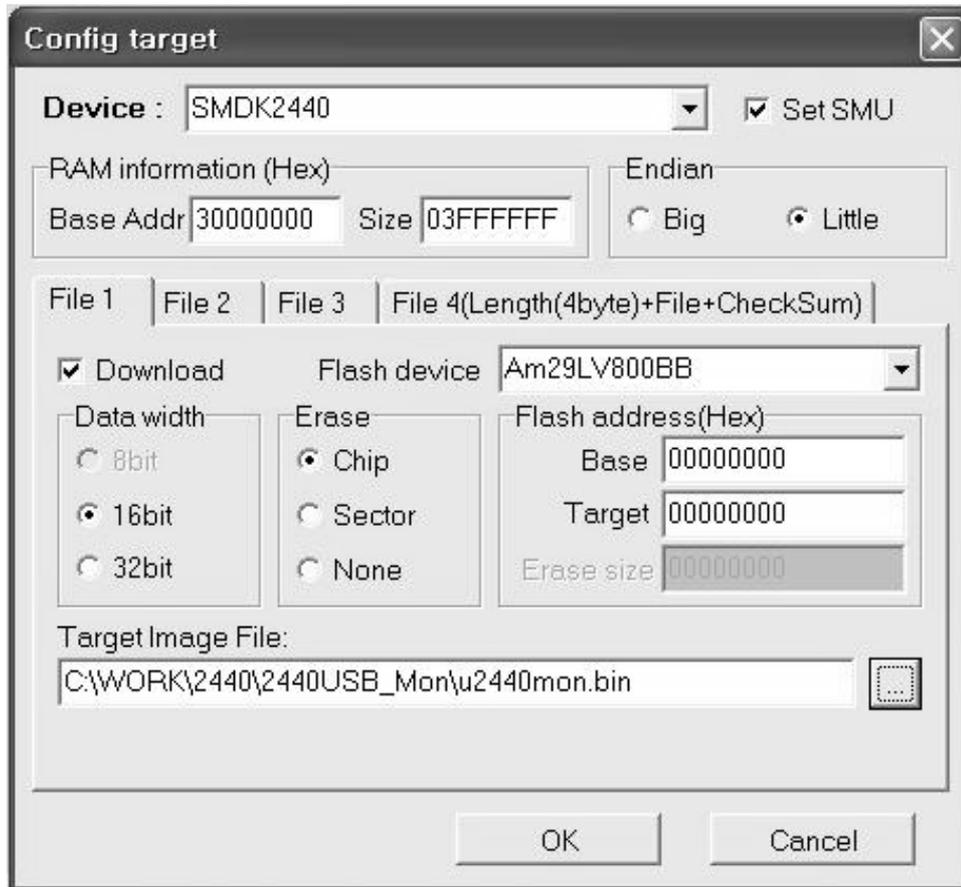
1. Connect OPENice32-A900 to PC through USB and to SMDK2440 board with 20pin Cable.
2. Run the Flash Write program and select Connect MDS from the File menu.



3. Select SMU Manger from the utility menu and choose a device file, SMDK2440. It is used to initialize the system registers in case of there is no boot ROM. If you can't find the file, download the device file SMDK2410 instead of SMDK2440. After that, edit each value if necessary.

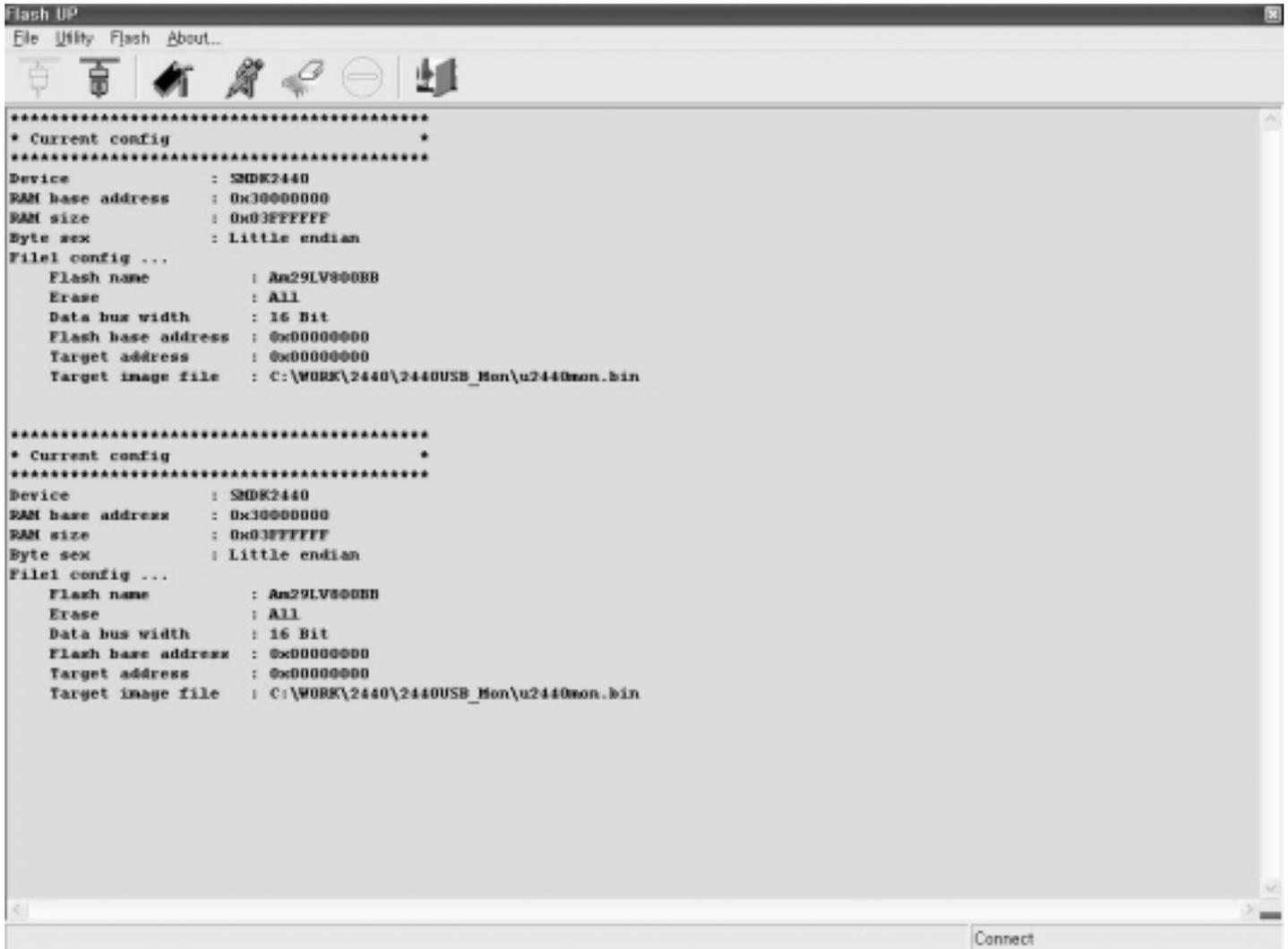


4. Select Config.. from the Flash menu and Set the write options as followings



- Device: SMDK2440
- Set SMU: Checked
- RAM Information:           Base Address:30000000           Size: 3FFFFFF
- Endian: Little
- File 1 page
- Download: checked
- Flash Device Name:AM29LV800BB
- Erase:Chip
- Data Bus width: 16bit
- Flash Address:           Base Address: 0           Target Address:0
- Target Image File: u2440mon.bin

5. Click OK. Then the current configuration is displayed in the window.



6. Select Write from the Flash Menu. Then it starts to erase the specified area of AMD Flash and write the image to the Flash memory. It takes about 10 second.

```

Flash UP
File  Utility  Flash  About...

*****
* Current config
*****
Device       : S3C2440
RAM base address : 0x30000000
RAM size     : 0x03FFFFFF
Byte sex     : Little endian
File config ...
  Flash name   : Am29LV800BB
  Erase       : All
  Data bus width : 16 Bit
  Flash base address : 0x00000000
  Target address : 0x00000000
  Target image file : C:\WORK\2440\2440USB_Mon\u2440mon.bin

FileName: u2440mon.bin
FileSize: 35412
Chip erase start ...
Erase complete!!!
u2440mon.bin @downloading ...
u2440mon.bin programming ...
u2440mon.bin program complete !!!

Connect
  
```

## WRITING IMAGE FILES TO INTEL STRATA FLASH MEMORY WITH UART

1. Connect MULTI-ICE and execute "2440norom.ini" file.

The screenshot displays the AXD (ARM920T\_D) software interface. The left pane shows the assembly code for the ARM920T\_D target, and the right pane shows the Command Line Interface (CLI) with a series of debug commands being executed.

**Assembly Code (Left Pane):**

```

30000074 [0xe59f0098] ldr r0,0x30000114 ; = #0x4c00000c
30000078 [0xe5802009] str r2,[r0,#0]
3000007c [0xe3a01020] mov r1,#0x20
30000080 [0xe2511001] subs r1,r1,#1
30000084 [0x1afffffd] bne 0x30000080 ; (ENTER_PMDN + 0x38)
30000088 [0xe59f0089] ldr r0,0x30000110 ; = #0x48000024
3000008c [0xe5803009] str r3,[r0,#0]
30000090 [0xe1a0f00c] mov pc,r14
ENTER_SLEEP [0xe59f0074] ldr r0,0x30000110 ; = #0x48000024
30000098 [0xe59f0009] ldr r1,[r0,#0]
3000009c [0xe3811840] orr r1,r1,#0x400000
300000a0 [0xe5801009] str r1,[r0,#0]
300000a4 [0xe3a01010] mov r1,#0x10
300000a8 [0xe2511001] subs r1,r1,#1
300000ac [0x1afffffd] bne 0x300000a8 ; (ENTER_SLEEP + 0x14)
300000b0 [0xe59f1069] ldr r1,0x30000118 ; = #0x56000080
300000b4 [0xe59f0009] ldr r0,[r1,#0]
300000b8 [0xe380d8e0] orr r0,r0,#0xe0000
300000bc [0xe5810009] str r0,[r1,#0]
300000c0 [0xe59f004c] ldr r0,0x30000114 ; = #0x4c00000c
300000c4 [0xe5802009] str r2,[r0,#0]
300000c8 [0xeafffffd] b 0x300000c8 ; (ENTER_SLEEP + 0x34)
WAKEUP_SLEEP [0xe59f1044] ldr r1,0x30000118 ; = #0x56000080
300000d0 [0xe59f1009] ldr r0,[r1,#0]
300000d4 [0xe3c0dae0] bic r0,r0,#0xe0000
300000d8 [0xe5810009] str r0,[r1,#0]
300000dc [0xe59f0038] ldr r0,0x3000011c ; = #0x30000408
300000e0 [0xe3a01448] mov r1,#0x48000000
300000e4 [0xe2802034] add r2,r0,#0x34
300000e8 [0xe4903004] ldr r3,[r0],#4
300000ec [0xe4813004] str r3,[r1],#4
300000f0 [0xe1520009] cap r2,r0
300000f4 [0x1afffffd] bne 0x300000e8 ; (WAKEUP_SLEEP + 0x1c)
300000f8 [0xe3a01f40] mov r1,#0x100
300000fc [0xe2511001] subs r1,r1,#1
30000100 [0x1afffffd] bne 0x300000fc ; (WAKEUP_SLEEP + 0x30)

```

**Command Line Interface (Right Pane):**

```

Debug > obey C:\WORK\2440\2440norom\2440norom.ini
Debug > com *****
Debug > com Filename: 2440norom.ini
Debug > com 2003. 5. xx 1st draft.
Debug > com *****
Debug > com For 83C2440X
Debug > com SDRAM Little 32, 64MB
Debug > com FCLK:101.25MHz UPLL:48MHz
Debug > com SDRAM refresh: 64ms(8Kcycle) -> 7.8us
Debug > swat %vector_catch 0x00
An expression could not be parsed or evaluated in the given context
Debug > swat %seahosting_enabled 0x00
An expression could not be parsed or evaluated in the given context
Debug > swat par %IFT_SVC
Invalid expression
Debug > com swat par %IF_SVC32
Debug > com [disable Watch-Dog reset]
Debug > swat *0x53000000 0
Debug > com << Clock setting >>
Debug > com [PLL lock time setting maximum]
Debug > swat *0x4c000000 [(0xff2<<12)+(0xfffc<<0)]
Debug > com FCLK:NCLK:FCLK=1:2:2.
Debug > swat *0x4c000014 [(0xc2)+(1<<1)+(0)]
Debug > com [FCLK FMS settings:101.25MHz -> 0x7f,2,2]
Debug > swat *0x4c000004 [(0x7f<<12)+(0x2<<4)+(0x2<<0)]
Debug > com [UCLK FMS settings:48MHz -> 0x78,2,3]
Debug > swat *0x4c000008 [(0x78<<12)+(0x2<<4)+(0x3<<0)]
Debug > com << Memory setting >>
Debug > com [Bank6/7: 32-bit bus width]
Debug > swat *0x48000000 0x22000000
Debug > com [Bank0-5: Access cycle: 14-clocks, others:0-clock]
Debug > swat *0x48000004 [(0<<13)+(0<<11)+(7<<8)+(0<<6)+(0<<4)+(0<<2)+
Debug > swat *0x48000008 [(0<<13)+(0<<11)+(7<<8)+(0<<6)+(0<<4)+(0<<2)+
Debug > swat *0x4800000c [(0<<13)+(0<<11)+(7<<8)+(0<<6)+(0<<4)+(0<<2)+
Debug > swat *0x48000010 [(0<<13)+(0<<11)+(7<<8)+(0<<6)+(0<<4)+(0<<2)+

```

For Help, press F1 Line 129, Col 0 Multi-ICE | ARM920T\_D | 2440norom.ini

## 2. Load the image file (2440TEST.axf) to execute.

The screenshot displays the AXD debugger interface. The main window shows the disassembly of the ARM920T.D image, with instructions such as HandlerSMI, HandlerFabort, and EnterPMDN. The Command Line Interface (CLI) on the right shows a series of debug commands used to configure the system, including setting the PLL lock time, disabling the Watch-Dog, and configuring memory settings for Bank6/7.

```

ARM920T.D - Disassembly
30000008 [0xea000057] b HandlerSMI
3000000c [0xea000062] b HandlerFabort
30000010 [0xea00005b] b HandlerFabort
30000014 [0xea00005e] b 0x30000014
30000018 [0xea000047] b HandlerIRQ
3000001c [0xea000040] b HandlerPIO
30000020 [0xea000008] b EnterPMDN
30000024 [0x0f10e11] dcd 0x0f10e11 ....
30000028 [0x000e300] dcd 0x000e300 ....
3000002c [0x0f10e01] dcd 0x0f10e01 ....
30000030 [0xffffffff] dcd 0xffffffff ....
30000034 [0xffffffff] dcd 0xffffffff ....
30000038 [0xffffffff] dcd 0xffffffff ....
3000003c [0xffffffff] dcd 0xffffffff ....
30000040 [0xffffffff] dcd 0xffffffff ....
30000044 [0xea000063] b ResetHandler
EnterPMDN[0xe1e02000] mov r2,r0
3000004c [0xe3100008] tat r0,#0
30000050 [0x1a00000f] bne ENTER_SLEEP
30000054 [0xe59f00b4] ldr r0,0x30000110 ; = #0x48000024
30000058 [0xe5903000] ldr r3,[r0,#0]
3000005c [0xe1a01003] mov r1,r3
30000060 [0xe3811840] cxt r1,r1,#0x400000
30000064 [0xe5801000] str r1,[r0,#0]
30000068 [0xe3a01010] mov r1,#0x10
3000006c [0xe2511001] subs r1,r1,#1
30000070 [0x1afffff4] bne 0x3000006c ; (EnterPMDN + 0x24)
30000074 [0xe59f0098] ldr r0,0x30000114 ; = #0x4c00000c
30000078 [0xe5803000] str r2,[r0,#0]
3000007c [0xe3a01020] mov r1,#0x20
30000080 [0xe2511001] subs r1,r1,#1
30000084 [0x1afffff4] bne 0x30000080 ; (EnterPMDN + 0x38)
30000088 [0xe59f0080] ldr r0,0x30000110 ; = #0x48000024
3000008c [0xe5803000] str r3,[r0,#0]
30000090 [0xe1e0c00e] mov pc,r14
ENTER_SLEEP[0xe59f0074] ldr r0,0x30000110 ; = #0x48000024

Command Line Interface
Command Line Interface
Debug >swat par 4IFt_SVC
Invalid expression
Debug >com swat par 4IFt_SVC32
Debug >com [disable Watch-Dog reset]
Debug >swat *0x53000000 0
Debug >com << Clock setting >>
Debug >com [PLL lock time setting maximum]
Debug >swat *0x4c000000 {(0xff<<12)+(0xff<<0)}
Debug >com FCLK:HCLK:PCLK=1:2:2.
Debug >swat *0x4c000014 {(0<<2)+(1<<1)+(0)}
Debug >com [FCLK FHS settings:101.25MHz -> 0x7f,2,2]
Debug >swat *0x4c000004 {(0x7f<<12)+(0x2<<4)+(0x2<<0)}
Debug >com [UCLK FHS setting:48MHz -> 0x78,2,3]
Debug >swat *0x4c000008 {(0x78<<12)+(0x2<<4)+(0x3<<0)}
Debug >com << Memory setting >>
Debug >com [Bank6/7: 32-bit bus width]
Debug >swat *0x48000000 0x22000000
Debug >com [Bank0-5: Access cycle: 14-clocks, others:0-clock]
Debug >swat *0x48000004 {(0<<13)+(0<<11)+(7<<8)+(0<<6)+(0<<4)+(0<<2)+
Debug >swat *0x48000008 {(0<<13)+(0<<11)+(7<<8)+(0<<6)+(0<<4)+(0<<2)+
Debug >swat *0x4800000c {(0<<13)+(0<<11)+(7<<8)+(0<<6)+(0<<4)+(0<<2)+
Debug >swat *0x48000010 {(0<<13)+(0<<11)+(7<<8)+(0<<6)+(0<<4)+(0<<2)+
Debug >swat *0x48000014 {(0<<13)+(0<<11)+(7<<8)+(0<<6)+(0<<4)+(0<<2)+
Debug >swat *0x48000018 {(0<<13)+(0<<11)+(7<<8)+(0<<6)+(0<<4)+(0<<2)+
Debug >com [Bank6/7: SDRAM, Trcd:2clock, CA:9-bit]
Debug >swat *0x4800001c {(3<<15)+(0<<2)+1}
Debug >swat *0x48000020 {(3<<15)+(0<<2)+1}
Debug >com [SDRAM refresh enable, Trp=2clk, Trc=5clk, Refresh:1654]
Debug >swat *0x48000024 {(1<<23)+(0<<22)+(0<<20)+(1<<18)+1654}
Debug >com [SCLF_EN enable, SCLF_EN enable, Bank6/7 memory map: 64MB]
Debug >swat *0x48000028 {0x1+(1<<5)+(1<<4)}
Debug >com [Bank6/7 CL: 3-clocks]
Debug >swat *0x4800002c 0x30
Debug >swat *0x48000030 0x30
Debug >
  
```

## 3. Execute 2440TEST code with Go command.

```

123 // Hidden
124 #define MPLL_SEL (0)
125 #define FCLK_SEL (2)
126
127
128
129
130 //-----
131
132
133 void Main(void)
134 {
135     int i;
136     unsigned int mpll_val;
137
138     led_Display(0xf);
139
140     // MMU init. I/D cache on.
141     MMU_Init();
142
143 #if ADS10
144     __rt_lib_init(); //for ADS 1.0
145 #endif
146
147     // Clock setting
148     ChangeClockDividez(12,12); // 1:2:4
149     ChangeMPLLValue(246,13,0); // 203.2MHz
150     //ChangeMPLLValue(88,1,1); // 192MHz
151     ChangeUPllValue(0x38,2,2); // 468MHz
152
153     // Clock calculation only for display information
154     Calc_Clock(0);
155     UPDATE_REFRESH(Hz1k);
156
157     // GPIO port init.
158     Port_Init();
  
```

```

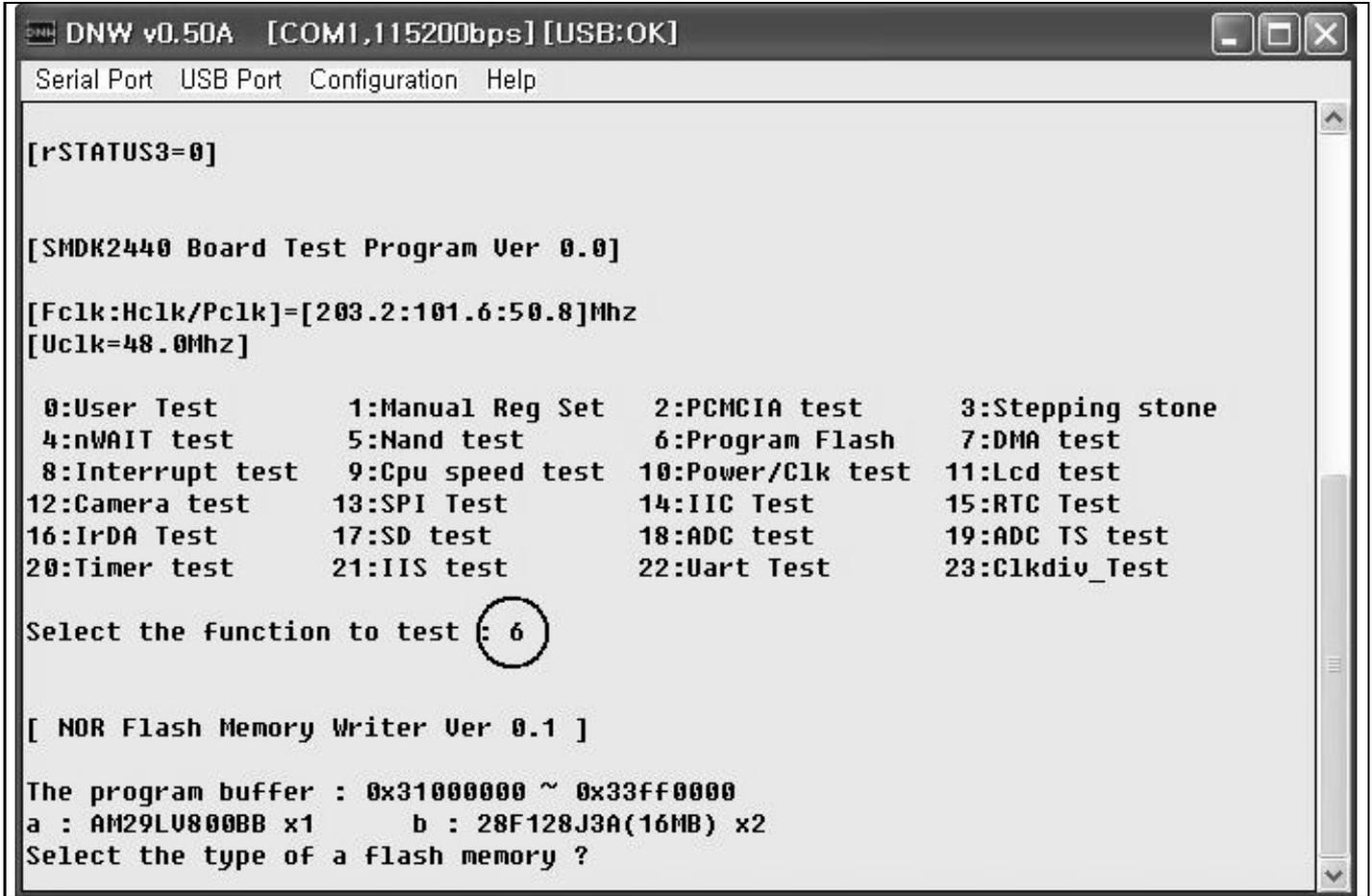
Command Line Interface
Command Line Interface
Debug >swat par %IFt_3VC
Invalid expression
Debug >com swat par %IF_SVC32
Debug >com [disable Watch-Dog reset]
Debug >swat *0x5300000 0
Debug >com << Clock setting >>
Debug >com [PLL lock time setting maximum]
Debug >swat *0x4c000000 ((0xffff<<12)+(0xffff<<0))
Debug >com FCLK:HCLK:FCLK=1:2:2.
Debug >swat *0x4c000014 ((0<<2)+(1<<1)+(0))
Debug >com [FCLK PMS setting:101.25MHz -> 0x7f,2,2]
Debug >swat *0x4c000004 ((0x7f<<12)+(0x2<<4)+(0x2<<0))
Debug >com [UCLK PMS setting:48MHz -> 0x78,2,3]
Debug >swat *0x4c000008 ((0x78<<12)+(0x2<<4)+(0x3<<0))
Debug >com << Memory setting >>
Debug >com [Bank6/7: 32-bit bus width]
Debug >swat *0x48000000 0x22000000
Debug >com [Bank0-5: Access cycle: 14-clocks, others:0-clock]
Debug >swat *0x48000004 ((0<<13)+(0<<11)+(7<<8)+(0<<6)+(0<<4)+(0<<2)+
Debug >swat *0x48000008 ((0<<13)+(0<<11)+(7<<8)+(0<<6)+(0<<4)+(0<<2)+
Debug >swat *0x4800000c ((0<<13)+(0<<11)+(7<<8)+(0<<6)+(0<<4)+(0<<2)+
Debug >swat *0x48000010 ((0<<13)+(0<<11)+(7<<8)+(0<<6)+(0<<4)+(0<<2)+
Debug >swat *0x48000014 ((0<<13)+(0<<11)+(7<<8)+(0<<6)+(0<<4)+(0<<2)+
Debug >swat *0x48000018 ((0<<13)+(0<<11)+(7<<8)+(0<<6)+(0<<4)+(0<<2)+
Debug >com [Bank6/7: 3DRAM, Trcd=2clock, Csr=9-bit]
Debug >swat *0x4800001c ((3<<15)+(0<<2)+1)
Debug >swat *0x48000020 ((3<<15)+(0<<2)+1)
Debug >com [3DRAM refresh enable, Trp=2clk, Trc=5clk, Refresh:1654]
Debug >swat *0x48000024 ((1<<23)+(0<<22)+(0<<20)+(1<<18)+1654)
Debug >com [SCKE_EN enable, SCLK_EN enable, Bank6/7 memory map: 64MB/]
Debug >swat *0x48000028 (0x1+(1<<5)+(1<<4))
Debug >com [Bank6/7 Cl: 3-clocks]
Debug >swat *0x4800002c 0x30
Debug >swat *0x48000030 0x30
Debug >
  
```

For Help, press F1 Running Image Line 134, Col 0 MultiICE ARM920T\_0 2440test.ad

4. Select "6:Program Flash" on the DNW.

**NOTE:** If you want to download 2440TEST.bin without MULTI-ICE, then skip the 1, 2 & 3 steps above and download 2440TEST.bin using the DNW (See EXECUTE 2440TEST WITHOUT MULTI-ICE).

After downloading 2440TEST.bin with the DNW, then you can also see the figure below.



```

DNW v0.50A [COM1,115200bps] [USB:OK]
Serial Port  USB Port  Configuration  Help

[rSTATUS3=0]

[SMDK2440 Board Test Program Ver 0.0]

[Fclk:Hclk/Pclk]=[203.2:101.6:50.8]Mhz
[Uclk=48.0Mhz]

 0:User Test          1:Manual Reg Set    2:PCMCIA test       3:Stepping stone
 4:nWAIT test        5:Nand test         6:Program Flash     7:DMA test
 8:Interrupt test    9:Cpu speed test   10:Power/Clk test  11:Lcd test
12:Camera test      13:SPI Test        14:IIC Test         15:RTC Test
16:IrDA Test        17:SD test         18:ADC test         19:ADC TS test
20:Timer test       21:IIS test        22:Uart Test        23:Clkdiv_Test

Select the function to test : 6

[ NOR Flash Memory Writer Ver 0.1 ]

The program buffer : 0x31000000 ~ 0x33ff0000
a : AM29LU800BB x1    b : 28F128J3A(16MB) x2
Select the type of a flash memory ?

```

5. Select the type of memory as 28F128J3A (INTEL STRATA flash) by typing 'b'.

6. Select whether you download through UART0 or MULTI-ICE.

— Type 'y' then you can download a target file through UART. See the figure below.

```

DNW v0.50A [COM1,115200bps] [USB:OK]
Serial Port  USB Port  Configuration  Help
0:User Test      1:Manual Reg Set  2:PCMCIA test    3:Stepping stone
4:nWAIT test     5:Nand test       6:Program Flash  7:DMA test
8:Interrupt test 9:Cpu speed test 10:Power/Clk test 11:Lcd test
12:Camera test   13:SPI Test       14:IIC Test      15:RTC Test
16:IrDA Test     17:SD test        18:ADC test      19:ADC TS test
20:Timer test    21:IIS test       22:Uart Test     23:Clkdiv_Test

Select the function to test : 6

[ NOR Flash Memory Writer Ver 0.1 ]

The program buffer : 0x31000000 ~ 0x33ff0000
a : AM29LV800BB x1      b : 28F128J3A(16MB) x2
Select the type of a flash memory ? (b)
Do you want to download through UART0 from 0x31000000? [y/n] : y

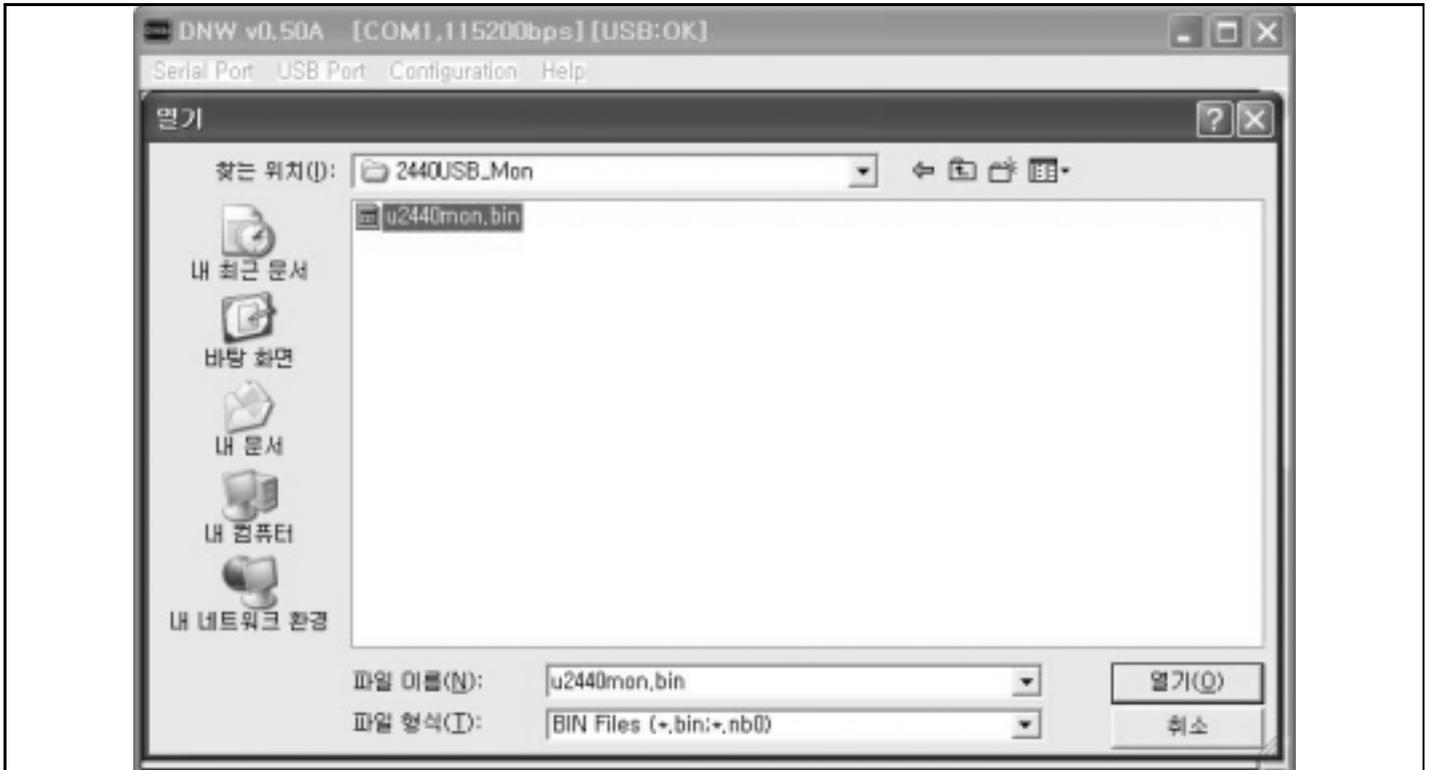
downloadAddress = 31000000
Download the plain binary file(.BHC) to be written
The file format : <n+6>(4)+(n)+CS(2)
To transmit .BIN file : wkocm2 xxx.BIN /1 /d:1
Download methods : COM:8Bit,NP,1STOP

STATUS :

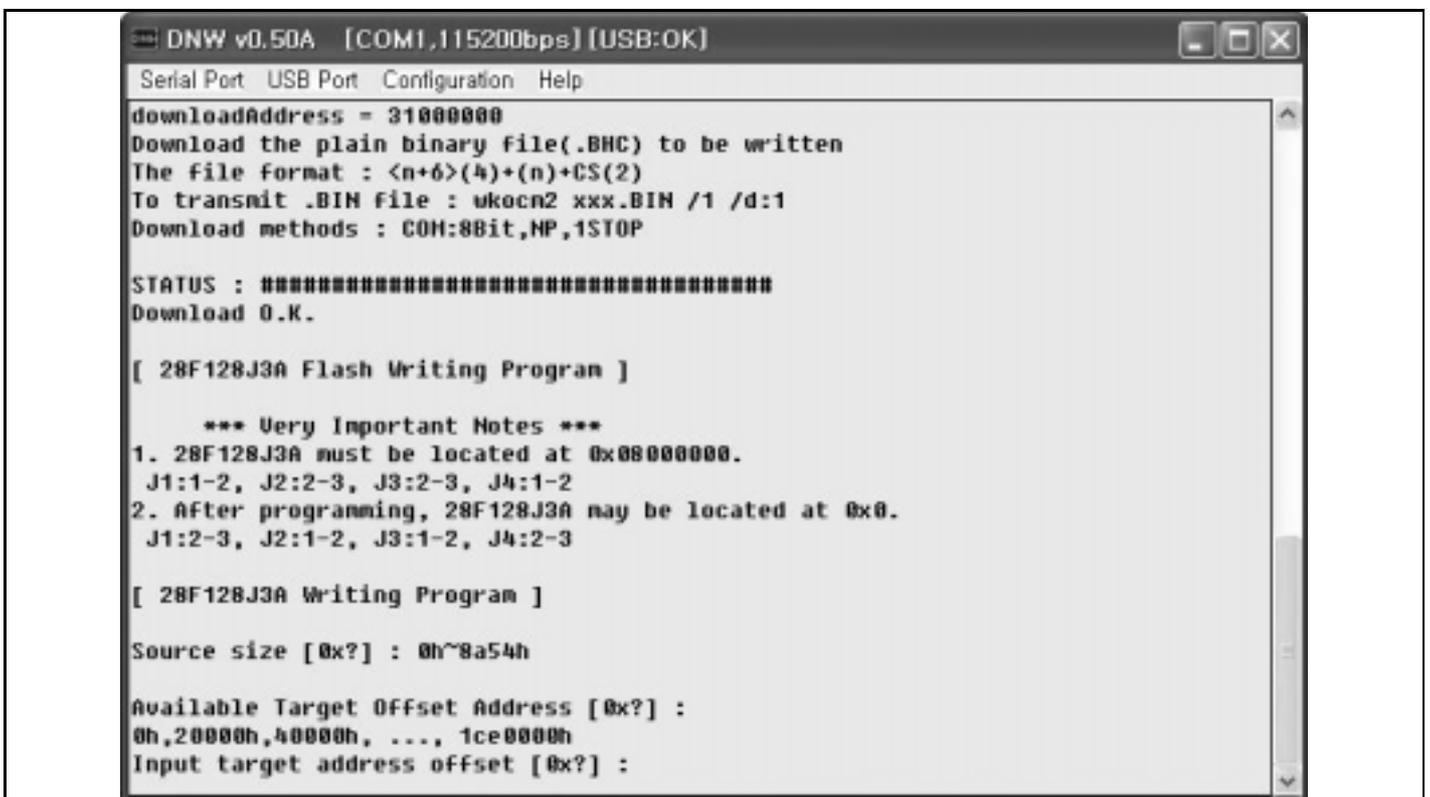
```

7. Download a target file with the DNW by selecting Transmit menu from Serial Port.

— Serial Port → Transmit



— Select and Download a target file.



8. Write input target-offset address.

```

DNW v0.50A [COM1,115200bps] [USB-OK]
Serial Port USB Port Configuration Help
Download the plain binary file(.BMC) to be written
The file format : <n>6>(A)*(n)+CS(2)
To transmit .BIN file : wkocn2 xxx.BIN /1 /d:1
Download methods : COM:8Bit,HP,1STOP

STATUS : #####
Download O.K.

[ 28F128J3A Flash Writing Program ]

*** Very Important Notes ***
1. 28F128J3A must be located at 0x00000000.
   J1:1-2, J2:2-3, J3:2-3, J4:1-2
2. After programming, 28F128J3A may be located at 0x0.
   J1:2-3, J2:1-2, J3:1-2, J4:2-3

[ 28F128J3A Writing Program ]

Source size [0x?]: 0h~8a54h

Available Target Offset Address [0x?]:
0h,20000h,40000h, ..., 1ce0000h
Input target address offset [0x?]: 0x0
Source base address(0x31000000) = 0x31000000
Target base address(0x08000000) = 0x08000000
Target offset (0x0) = 0x0
Target size (0x20000*n) = 0x8a54

Erase the sector : 0x00000000.
Block_00000000 Erase O.K.

Start of the data writing...

End of the data writing
Verifying Start...
Verifying End!!!

0:User Test      1:Manual Reg Set  2:PCMCIA test    3:Stepping stone
4:nWMT test     5:Hand test      6:Program Flash  7:DMA test
8:Interrupt test 9:Cpu speed test 10:Power/CLK test 11:Lcd test
12:Camera test  13:SPI Test      14:IIC Test      15:RTC Test
16:IrDA Test    17:UART Test     18:SD test       19:ADC test
20:ADC TS test  21:Timer test    22:IIS test      23:Clkdiv_Test

Select the function to test :

```

9. Turn the SMDK2440 off and again on.

## WRITING IMAGE FILES TO INTEL STRATA FLASH MEMORY WITH MULTI-ICE

1. Connect MULTI-ICE and execute "norom.ini" file.

The screenshot shows the ARM Debugger interface with two main windows: 'ARM - Execution Window' and 'Command Window'.

**ARM - Execution Window:** Displays assembly code with addresses and instructions. The current instruction at address 0x000004f4 is highlighted.

```

0x00000490 bl 0xa2c
0x00000494 bl 0x34c
0x00000498 mov r0,#0
0x0000049c bl 0x864
0x000004a0 mov r0,#0
0x000004a4 bl 0xc08
0x000004a8 add r0,pc,#0x200 ; #0x6b0
0x000004ac bl 0xdc4
0x000004b0 mov r1,#0x32
0x000004b4 add r0,pc,#0x220 ; #0x6dc
0x000004b8 bl 0xdc4
0x000004bc ldr r2,0x00000718 ; = #0x33ffff00
0x000004c0 mov r1,#0x30000000
0x000004c4 add r0,pc,#0x250 ; #0x71c
0x000004c8 bl 0xdc4
0x000004cc ldr r2,0x00000738 ; = #0x33ff0000
0x000004d0 mov r1,#0x30000000
0x000004d4 add r0,pc,#0x260 ; #0x73c
0x000004d8 bl 0xdc4
0x000004dc mov r4,#0x30000000
0x000004e0 adds r12,r4,#0xcd000000
0x000004e4 subcss r12,r12,#0xff0000
0x000004e8 bcs 0x500
0x000004ec b 0x4f8
0x000004f0 add r4,r4,#0x100
0x000004f4 b 0x4e0
0x000004f8 str r4,[r4,#0]
0x000004fc b 0x4f0
0x00000500 add r0,pc,#0x250 ; #0x758
0x00000504 bl 0xdc4
0x00000508 mov r4,#0x30000000
0x0000050c adds r12,r4,#0xcd000000
  
```

**Command Window:** Shows the ARMsd Command Interface and the execution of the 'norom.ini' file. The output includes memory addresses and values.

```

ARMsd Command Interface
Debug: ob d:\2410\norom.ini
>|* S3C2410
>|* SDRAM_Little_32
>|* 64MB
>
>let $vector_catch = 0x00
>let $semihosting_enabled = 0x00
>let psr=%IFt_SVC
>|let psr=%IF_SVC32
>
>|disable wdt
>|let 0x53000000=0
>
>|pllset
>let 0x4c000004=((0x70<<12)+(0x4<<4)+0x2)
>let 0x4c000008=((0x58<<12)+(0x4<<4)+0x2)
>
>|memset
>let 0x48000000=0x22000000
>let 0x48000004=((0<<13)+(0<<11)+(7<<8)+(0<<6)+(0<<4)+(0<<2)+0)
>let 0x48000008=((0<<13)+(0<<11)+(7<<8)+(0<<6)+(0<<4)+(0<<2)+0)
>let 0x4800000c=((0<<13)+(0<<11)+(7<<8)+(0<<6)+(0<<4)+(0<<2)+0)
>let 0x48000010=((0<<13)+(0<<11)+(7<<8)+(0<<6)+(0<<4)+(0<<2)+0)
>let 0x48000014=((0<<13)+(0<<11)+(7<<8)+(0<<6)+(0<<4)+(0<<2)+0)
>let 0x48000018=((0<<13)+(0<<11)+(7<<8)+(0<<6)+(0<<4)+(0<<2)+0)
>let 0x4800001c=((3<<15)+(0<<2)+1)
>let 0x48000020=((3<<15)+(0<<2)+1)
>let 0x48000024=((1<<23)+(0<<22)+(0<<20)+(1<<18)+(2<<16)+1113)
>let 0x48000028=0x32
>let 0x4800002c=0x20
Debug: |
  
```

The status bar at the bottom shows the file path: C:\Program Files\ARM\Multi-ICE\Multi-ICE.dll, and the target device: localhost: TAP 0, ARM920T.

2. Load the image file (2440TEST.axf) to execute.

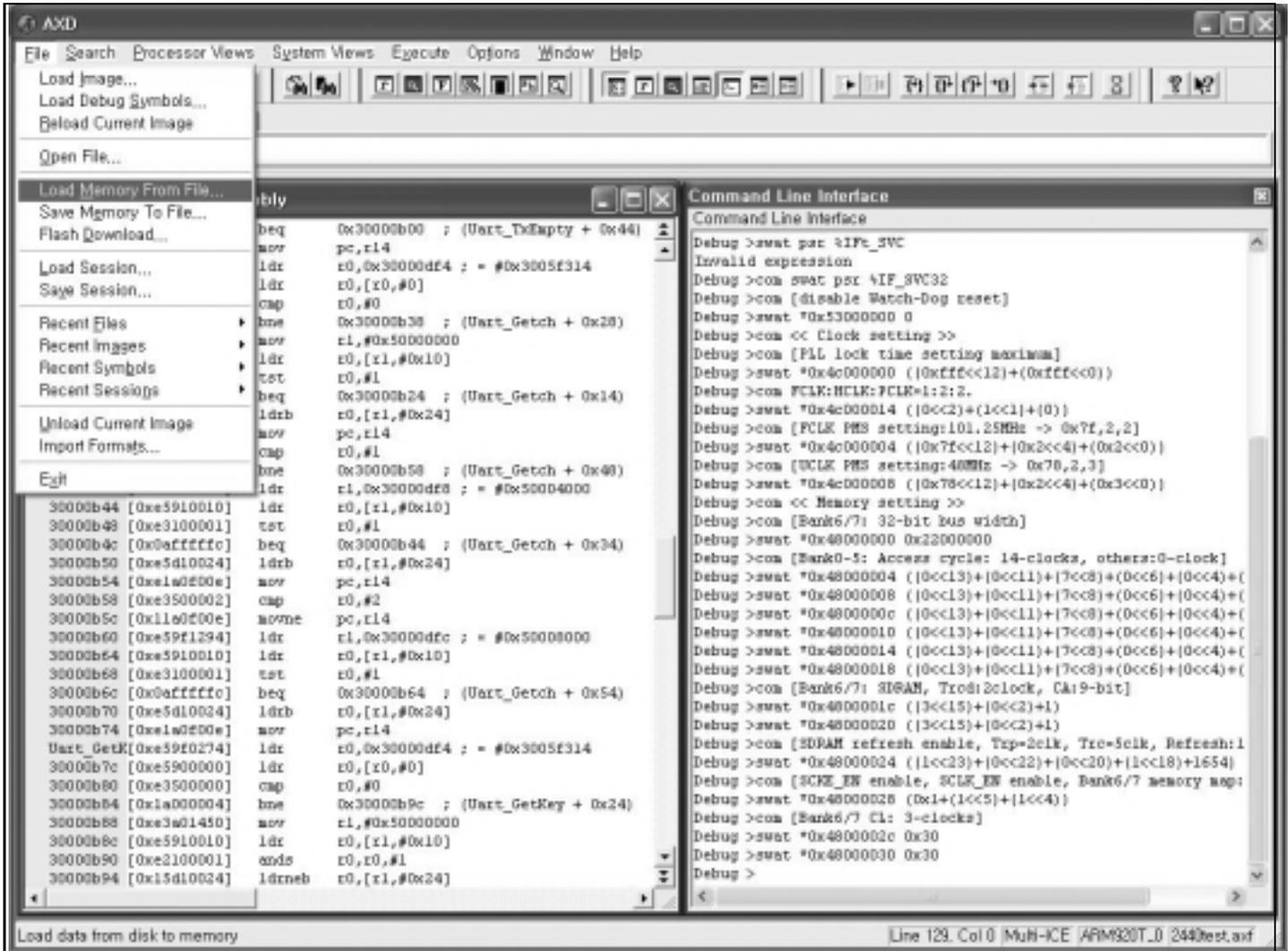
The screenshot shows the ARM920T\_D Disassembly window on the left and the Command Line Interface window on the right. The disassembly window shows instructions for the ARM920T\_D target, including reset handlers and memory settings. The Command Line Interface window shows a series of debug commands being executed, such as loading the image file, setting the filename, and configuring the system clock and memory settings.

```

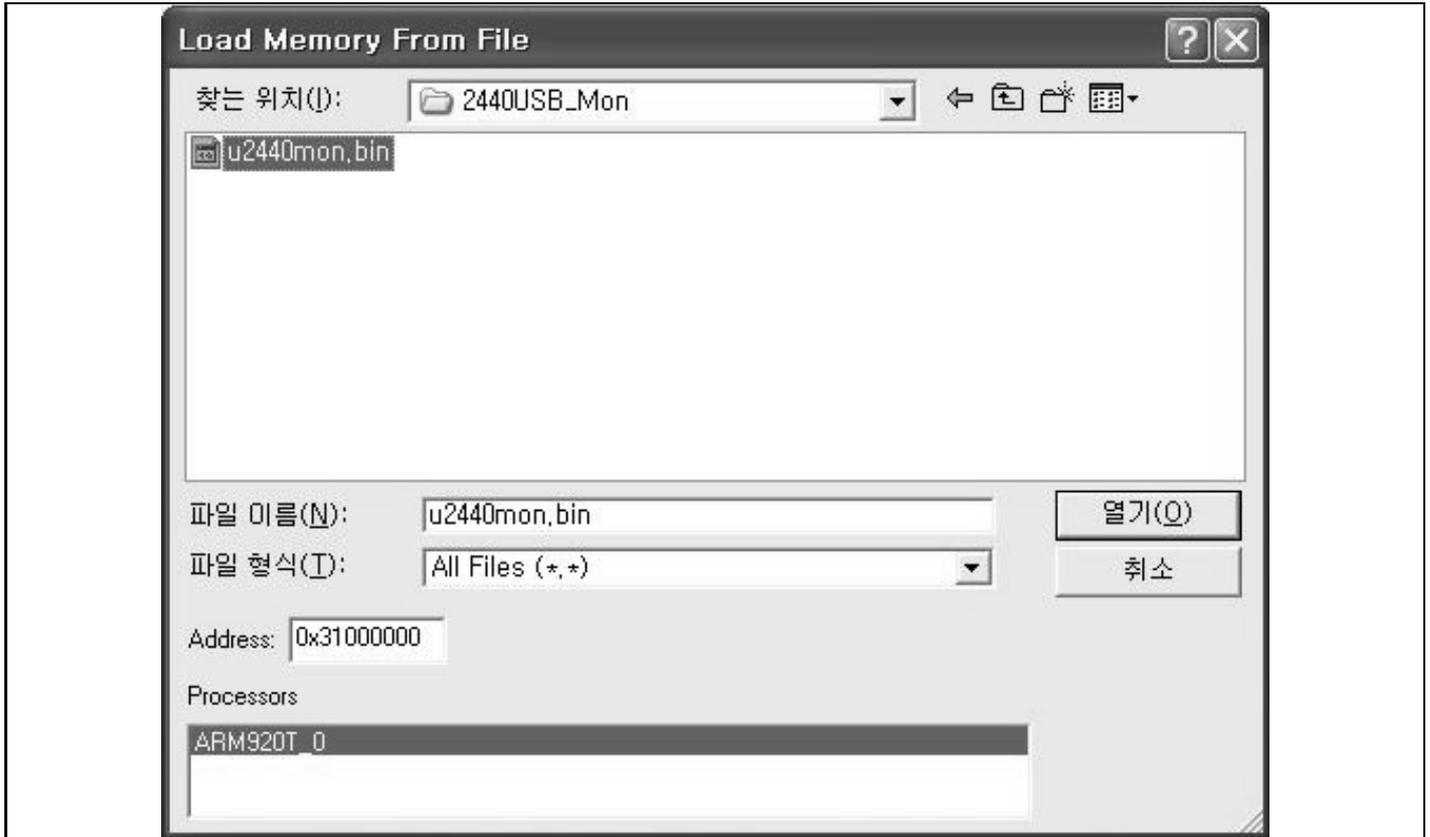
ARM920T_D - Disassembly
2ffffe0 [0xffffffff] dci 0xffffffff ; ? undefined
2ffffe4 [0xffffffff] dci 0xffffffff ; ? undefined
2ffffe8 [0xffffffff] dci 0xffffffff ; ? undefined
2ffffec [0xffffffff] dci 0xffffffff ; ? undefined
2fffff0 [0xffffffff] dci 0xffffffff ; ? undefined
2fffff4 [0xffffffff] dci 0xffffffff ; ? undefined
2fffff8 [0xffffffff] dci 0xffffffff ; ? undefined
2fffffc [0xffffffff] dci 0xffffffff ; ? undefined
3000000 [0xea000070] b ResetHandler
3000004 [0xea0004e] b HandlerUndef
3000008 [0xea00053] b HandlerSWI
300000c [0xea0005e] b HandlerPabort
3000010 [0xea00057] b HandlerDabort
3000014 [0xeaffffff] b 0x3000014
3000018 [0xea00043] b HandlerIRQ
300001c [0xea0003e] b HandlerFIQ
3000020 [0xea00008] b EnterPMM
3000024 [0x0f10ee11] dcd 0x0f10ee11 ....
3000028 [0x0080e380] dcd 0x0080e380 ....
300002c [0x0f10ee01] dcd 0x0f10ee01 ....
3000030 [0xffffffff] dcd 0xffffffff ....
3000034 [0xffffffff] dcd 0xffffffff ....
3000038 [0xffffffff] dcd 0xffffffff ....
300003c [0xffffffff] dcd 0xffffffff ....
3000040 [0xffffffff] dcd 0xffffffff ....
3000044 [0xea0005f] b ResetHandler
EnterPMM[0xe1a02000] mov r2,r0
300004c [0xe3100008] tst r0,#8
3000050 [0x1a00000f] bne ENTER_SLEEP
3000054 [0xe59200a4] ldr r0,0x30000100 ; = #0x48000024
3000058 [0xe5903000] ldr r3,[r0,#0]
300005c [0xe1a01003] mov r1,r3
3000060 [0xe3811840] orr r1,r1,#0x400000
3000064 [0xe5801000] str r1,[r0,#0]
3000068 [0xe3a01010] mov r1,#0x10
300006c [0xe2511001] suba r1,r1,#1

Command Line Interface
Debug > obey C:\WORK\2440\2440norom\2440norom.ini
Debug > ccm *****
Debug > ccm Filename: 2440norom.ini
Debug > ccm 2003. 5. xx 1st draft.
Debug > ccm *****
Debug > ccm For S3C2440X
Debug > ccm SRAM_Little_32, 64MB
Debug > ccm PCLK:101.25MHz UPLL:48MHz
Debug > ccm SRAM refresh: 64ns[8Kcycle] -> 7.8us
Debug > swat 4vector_catch 0x00
An expression could not be parsed or evaluated in the given context
Debug > swat $semihosting_enabled 0x00
An expression could not be parsed or evaluated in the given context
Debug > swat pcr 4IF_SVC
Invalid expression
Debug > ccm swat pcr 4IF_SVC32
Debug > ccm [disable Watch-Dog reset]
Debug > swat *0x53000000 0
Debug > ccm << Clock setting >>
Debug > ccm [PLL lock time setting maximum]
Debug > swat *0x4c000000 |(0xffff<12)+(0xffff<0})
Debug > ccm PCLK:HCLK:PCLK=1:2:2.
Debug > swat *0x4c000014 |(0<<2)+(1<<1)+(0)
Debug > ccm [PCLK MHS setting:101.25MHz -> 0x7f,2,2]
Debug > swat *0x4c000004 |(0x7f<<12)+(0x2<<4)+(0x2<<0})
Debug > ccm [UCLK MHS setting:48MHz -> 0x78,2,3]
Debug > swat *0x4c000008 |(0x78<<12)+(0x2<<4)+(0x3<<0})
Debug > ccm << Memory setting >>
Debug > ccm [Bank6,7: 32-bit bus width]
Debug > swat *0x48000000 0x22000000
Debug > ccm [Bank0-5: Access cycle: 14-clocks, others:0-clock]
Debug > swat *0x48000004 |(0<<13)+(0<<11)+(7<<8)+(0<<6)+(0<<4)+(0<<2)+0)
Debug > swat *0x48000008 |(0<<13)+(0<<11)+(7<<8)+(0<<6)+(0<<4)+(0<<2)+0)
Debug > swat *0x4800000c |(0<<13)+(0<<11)+(7<<8)+(0<<6)+(0<<4)+(0<<2)+0)
Debug > swat *0x48000010 |(0<<13)+(0<<11)+(7<<8)+(0<<6)+(0<<4)+(0<<2)+0)
  
```

## 3. Select Load Memory From File... on the file menu of AXD.



- Get the target file to 0x31000000 in SMDK2440 Board.



## 4. Execute 2440TEST.axf file with GO command.

The screenshot shows the ARM920T\_D Disassembly window on the left and the Command Line Interface window on the right. The disassembly window displays assembly code for the ARM920T\_D target, including instructions like `cap r0,#0`, `bne 0x3000b38 ; (Uart_Getch + 0x28)`, `ldr r0,[r1,#0x10]`, `tst r0,#1`, `beq 0x3000b34 ; (Uart_Getch + 0x14)`, `ldrb r0,[r1,#0x24]`, `mov pc,r14`, `cap r0,#1`, `bne 0x3000b58 ; (Uart_Getch + 0x48)`, `ldr r1,0x3000dfe8 ; = #0x50004000`, `ldr r0,[r1,#0x10]`, `tst r0,#1`, `beq 0x3000b44 ; (Uart_Getch + 0x34)`, `ldrb r0,[r1,#0x24]`, `mov pc,r14`, `cap r0,#2`, `movme pc,r14`, `ldr r1,0x3000dfe8 ; = #0x50008000`, `ldr r0,[r1,#0x10]`, `tst r0,#1`, `beq 0x3000b64 ; (Uart_Getch + 0x54)`, `ldrb r0,[r1,#0x24]`, `mov pc,r14`, `ldr r0,0x3000dfe4 ; = #0x3005e314`, `ldr r0,[r0,#0]`, `cap r0,#0`, `bne 0x3000b9c ; (Uart_GetKey + 0x24)`, `mov r1,#0x50000000`, `ldr r0,[r1,#0x10]`, `ands r0,r0,#1`, `ldrneb r0,[r1,#0x24]`, `mov pc,r14`, `cap r0,#1`, `bne 0x3000bb8 ; (Uart_GetKey + 0x40)`, and `ldr r1,0x3000dfe8 ; = #0x50004000`.

The Command Line Interface window shows the following commands and their outputs:

```

Invalid expression
Debug >com swat par 4IF_SVC32
Debug >com [disable Watch-Dog reset]
Debug >swat *0x53000000 0
Debug >com << Clock setting >>
Debug >com [PLL lock time setting maximum]
Debug >swat *0x4c000000 ((0xffff<12)+(0xffff<0))
Debug >com [CLK:NCLK:PCLK=1:2:2]
Debug >swat *0x4c000014 ((0<2)+(1<1)+(0))
Debug >com [CLK FHS settings:101.25MHz -> 0x7f,2,2]
Debug >swat *0x4c000004 ((0x7f<12)+(0x2<4)+(0x2<0))
Debug >com [UCLK FHS settings:48MHz -> 0x78,2,3]
Debug >swat *0x4c000008 ((0x78<12)+(0x2<4)+(0x3<0))
Debug >com << Memory setting >>
Debug >com [Bank6/7: 32-bit bus width]
Debug >swat *0x48000000 0x22000000
Debug >com [Bank0-5: Access cycle: 14-clocks, others:0-clock]
Debug >swat *0x48000004 ((0<13)+(0<11)+(7<8)+(0<6)+(0<4)+(
Debug >swat *0x48000008 ((0<13)+(0<11)+(7<8)+(0<6)+(0<4)+(
Debug >swat *0x4800000c ((0<13)+(0<11)+(7<8)+(0<6)+(0<4)+(
Debug >swat *0x48000010 ((0<13)+(0<11)+(7<8)+(0<6)+(0<4)+(
Debug >swat *0x48000014 ((0<13)+(0<11)+(7<8)+(0<6)+(0<4)+(
Debug >swat *0x48000018 ((0<13)+(0<11)+(7<8)+(0<6)+(0<4)+(
Debug >com [Bank6/7: SDRAM, Trd=2clock, CA:9-bit]
Debug >swat *0x4800001c ((3<15)+(0<2)+1)
Debug >swat *0x48000020 ((3<15)+(0<2)+1)
Debug >com [SDRAM refresh enable, Trp=2clk, Trc=5clk, Rerfresh:1]
Debug >swat *0x48000024 ((1<23)+(0<22)+(0<20)+(1<18)+1654)
Debug >com [SCYX_EN enable, SCLK_EN enable, Bank6/7 memory map]
Debug >swat *0x48000028 (0x1+(1<5)+(1<4))
Debug >com [Bank6/7 CL: 3-clocks]
Debug >swat *0x4800002c 0x30
Debug >swat *0x48000030 0x30
Debug >go
Debug >

```

5. Select " 6:Program Flash " on the DNW.

**NOTE:** If you want to download 2440TEST.bin without MULTI-ICE, then skip the 1, 2 & 3 steps above and download 2440TEST.bin using the DNW (See EXECUTE 2440TEST WITHOUT MULTI-ICE).  
After downloading 2440TEST.bin with the DNW, then you can also see the figure below.

```

DNW v0.50A [COM1,115200bps] [USB:OK]
Serial Port  USB Port  Configuration  Help
[rSTATUS3=0]

[SMDK2440 Board Test Program Ver 0.0]
[Fclk:Hclk/Pclk]=[203.2:101.6:50.8]Mhz
[Uclk=48.0Mhz]

 0:User Test          1:Manual Reg Set    2:PCMCIA test       3:Stepping stone
 4:nWAIT test        5:Nand test         6:Program Flash     7:DMA test
 8:Interrupt test    9:Cpu speed test   10:Power/Clk test   11:Lcd test
12:Camera test      13:SPI Test         14:IIC Test          15:RTC Test
16:IrDA Test        17:UART Test        18:SD test           19:ADC test
20:ADC TS test      21:Timer test       22:IIS test          23:Clkdiv_Test

Select the function to test : 6

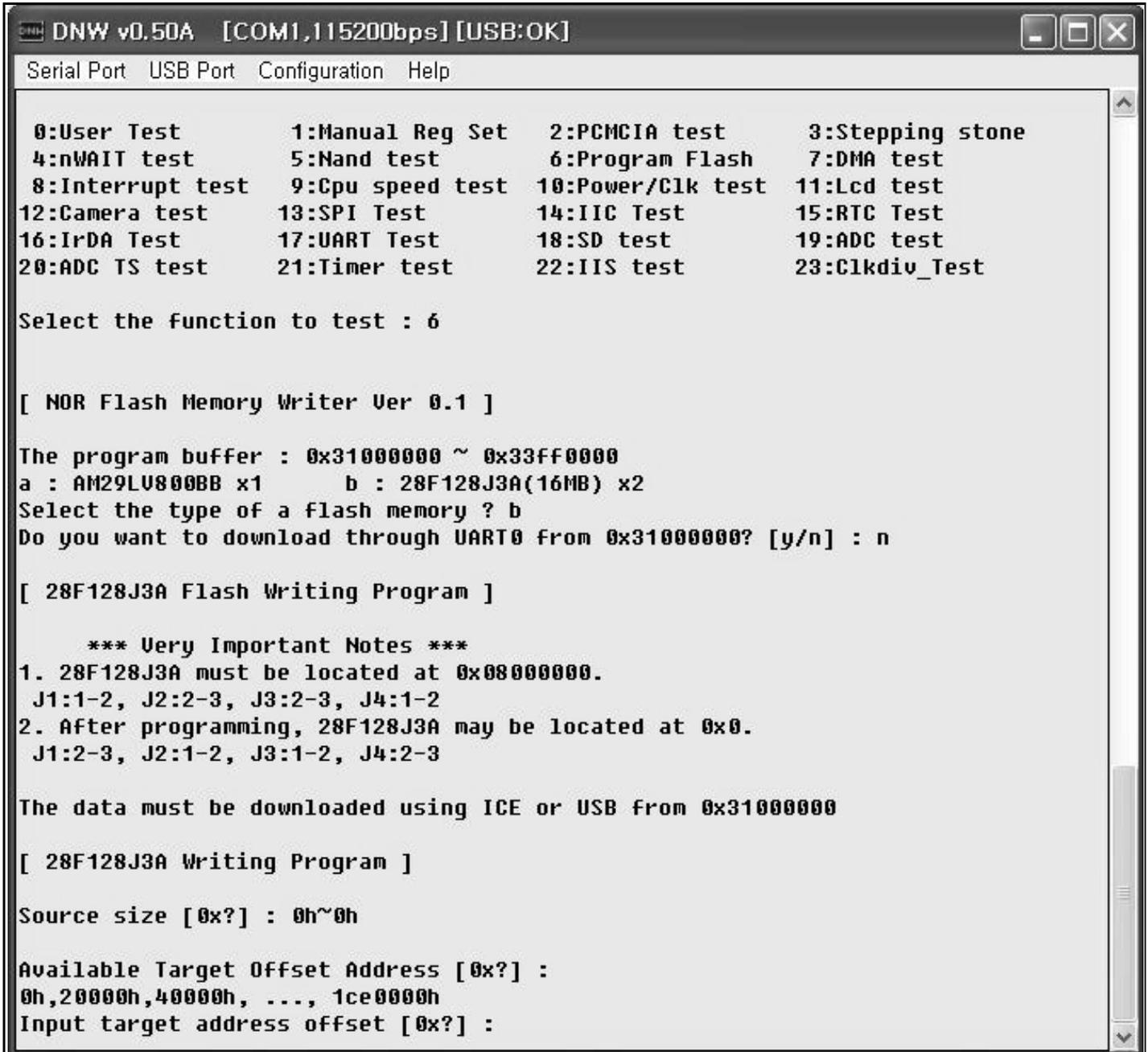
[ NOR Flash Memory Writer Ver 0.1 ]

The program buffer : 0x31000000 ~ 0x33ff0000
a : AM29LV800BB x1   b : 28F128J3A(16MB) x2
Select the type of a flash memory ? b
Do you want to download through UART0 from 0x31000000? [y/n] : n

```

6. Select the type of memory as 28F128J3A (INTEL STRATA flash) by typing 'b'.

7. Select whether you download through UART0 or MULTI-ICE.  
 — Type 'n' then you can see the figure below in the DNW.



```

DNW v0.50A [COM1,115200bps] [USB:OK]
Serial Port  USB Port  Configuration  Help

0:User Test      1:Manual Reg Set  2:PCMCIA test    3:Stepping stone
4:nWAIT test    5:Nand test       6:Program Flash  7:DMA test
8:Interrupt test 9:Cpu speed test 10:Power/Clk test 11:Lcd test
12:Camera test  13:SPI Test       14:IIC Test      15:RTC Test
16:IrDA Test    17:UART Test     18:SD test       19:ADC test
20:ADC TS test  21:Timer test    22:IIS test      23:Clkdiv_Test

Select the function to test : 6

[ NOR Flash Memory Writer Ver 0.1 ]

The program buffer : 0x31000000 ~ 0x33ff0000
a : AM29LV800BB x1      b : 28F128J3A(16MB) x2
Select the type of a flash memory ? b
Do you want to download through UART0 from 0x31000000? [y/n] : n

[ 28F128J3A Flash Writing Program ]

*** Very Important Notes ***
1. 28F128J3A must be located at 0x08000000.
   J1:1-2, J2:2-3, J3:2-3, J4:1-2
2. After programming, 28F128J3A may be located at 0x0.
   J1:2-3, J2:1-2, J3:1-2, J4:2-3

The data must be downloaded using ICE or USB from 0x31000000

[ 28F128J3A Writing Program ]

Source size [0x?] : 0h~0h

Available Target Offset Address [0x?] :
0h,20000h,40000h, ..., 1ce0000h
Input target address offset [0x?] :

```

8. Write input target address offset and size of the target file in hexadecimal.

```

DNW v0.50A [COM1,115200bps][USB:OK]
Serial Port  USB Port  Configuration  Help
The program buffer : 0x31000000 ~ 0x33FF0000
a : 0x29L0800BB x1      b : 28F128J3A(16MB) x2
Select the type of a flash memory ? b
Do you want to download through UART0 from 0x31000000? [y/n] : n

[ 28F128J3A Flash Writing Program ]

*** Very Important Notes ***
1. 28F128J3A must be located at 0x08000000.
   J1:1-2, J2:2-3, J3:2-3, J4:1-2
2. After programming, 28F128J3A may be located at 0x0.
   J1:2-3, J2:1-2, J3:1-2, J4:2-3

The data must be downloaded using ICE or USB from 0x31000000

[ 28F128J3A Writing Program ]

Source size [0x?] : 0h~0h

Available Target Offset Address [0x?] :
0h,20000h,40000h, ..., 1c0000h
Input target address offset [0x?] : 0x0
Input target size [0x?] : 0x10000
Source base address(0x31000000) = 0x31000000
Target base address(0x08000000) = 0x08000000
Target offset      (0x0)      = 0x0
Target size       (0x20000*n) = 0x10000

Erase the sector : 0x08000000.
Block_8000000 Erase O.K.

Start of the data writing...
[1]
End of the data writing
Verifying Start...
Verifying End!!!
0:User Test      1:Manual Reg Set  2:PGMClk test   3:Stepping stone
4:WAIT test     5:Nand test      6:Program Flash 7:DMA test
8:Interrupt test 9:Cpu speed test 10:Power/Clk test 11:Lcd test
12:Camera test  13:SPI Test     14:IIC Test     15:RTC Test
16:IrDA Test    17:UART Test    18:SD test      19:ADC test
20:ADC TS test  21:Timer test   22:IIS test     23:Clkdiv_Test

Select the function to test :

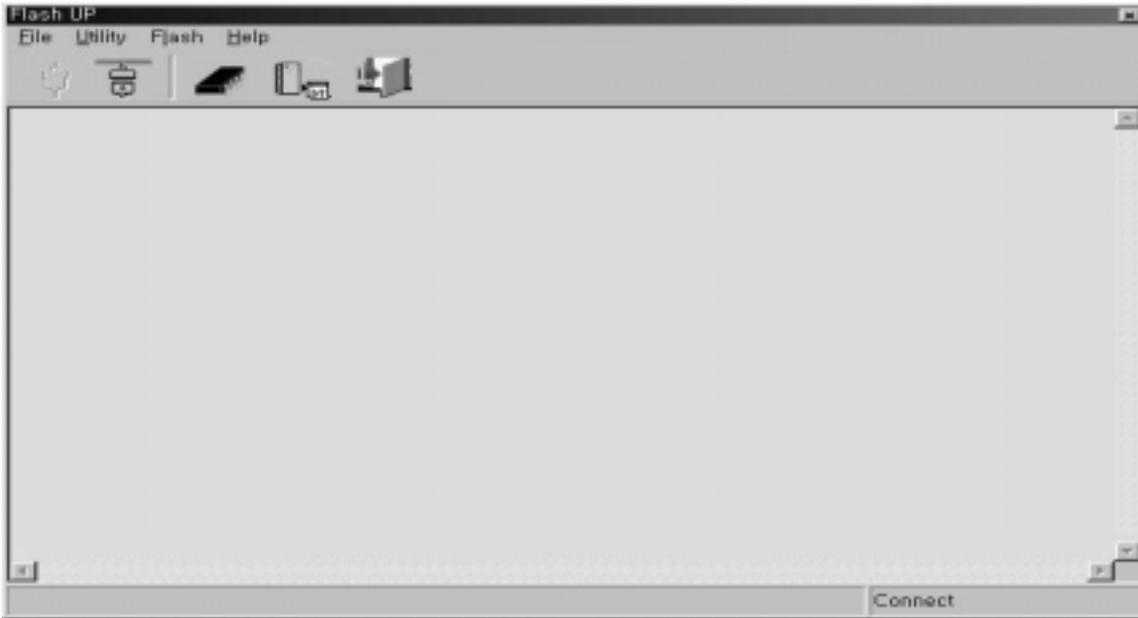
```

9. Turn the SMDK2440 off and again on.

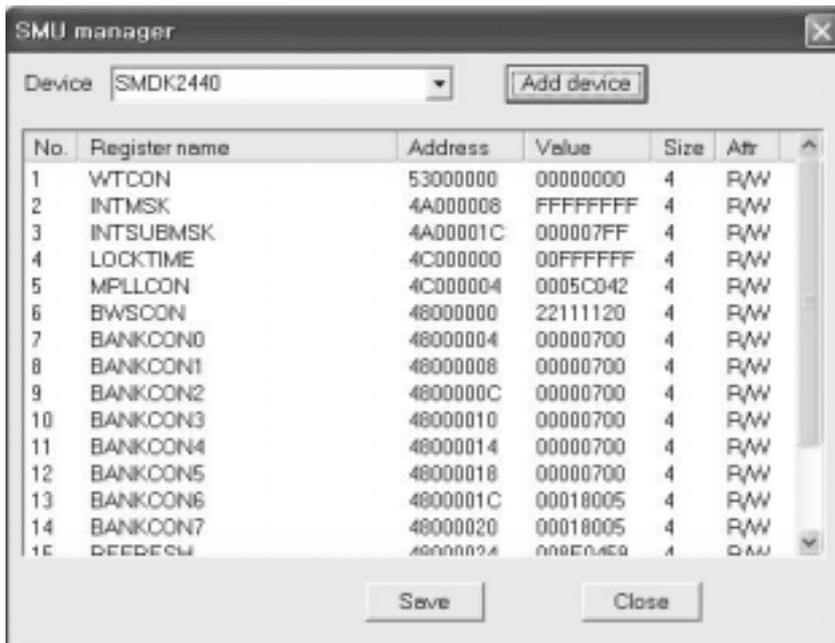
## WRITING IMAGE FILES TO INTEL STRATA FLASH MEMORY WITH OPENICE32-A900

OPENice32-A900 can write image to Intel Strata Flash memory as Multi-ICE. However, OPENice32-A900 provide a Flash Write Program that is easy to use and don't require ARM SDT/ADS debugger nor DNW. For more information on the program, refer to OPENice32-A900 manual or contact AIJI System ([www.aijisystem.com](http://www.aijisystem.com)).

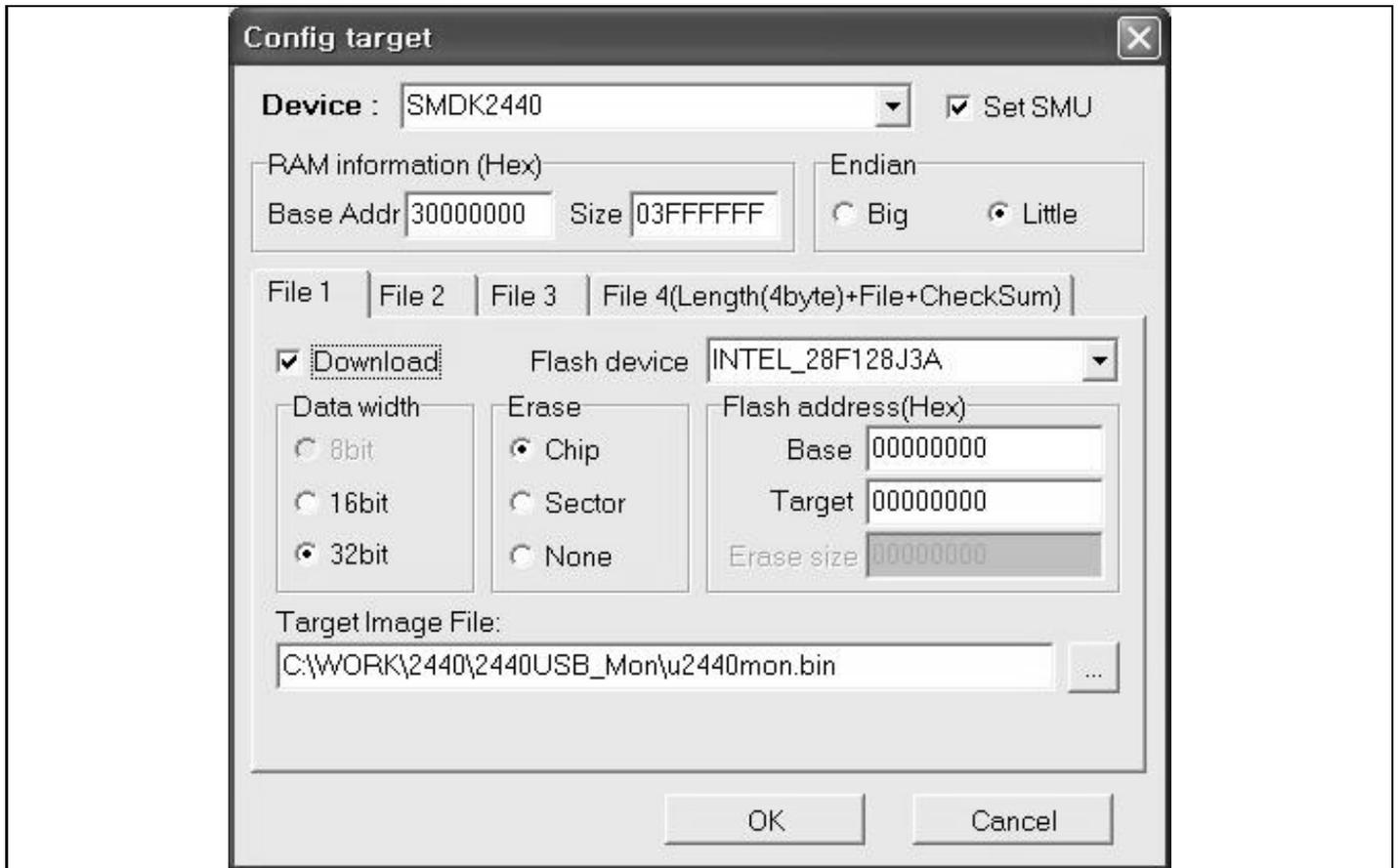
1. Connect OPENice32-A900 to PC through USB and to SMDK2440 board with 20pin Cable.
2. Set the Jumper J1, J2, J3 and J4 as followings and switch on the board  
J1: 2-3 (short)      J2: 1-2 (short)      J3: 1-2 (short)      J4: 2-3 (short)
3. Run the Flash Write program and select Connect MDS from the File menu.



4. Select SMU Manger from the utility menu and choose a device file, SMDK2440. It is used to initialize the system registers in case of there is no boot ROM. If you can't find the file, download the device file SMDK2410 instead of SMDK2440. After that, edit each value if necessary.



5. Select Config.. from the Flash menu and Set the write options as followings



- Device: SMDK2440
- Set SMU: Checked
- RAM Information:           Base Address:30000000           Size: 3FFFFFF
- Endian: Little
- File 1 page
- Download: checked
- Flash Device Name: INTEL\_28F128J3A
- Erase:Chip
- Data Bus width: 32bit
- Flash Address:           Base Address: 0           Target Address:0
- Target Image File: u2440mon.bin

6. Click OK. Then the current configuration is displayed in the window.



7. Select Write from the Flash Menu. Then it starts to erase the specified area of Intel Strata Flash and write the image to the Flash memory. It takes about 10 second.



7. Select Write from the Flash Menu. Then it starts to erase the specified area of Intel Strata Flash and write the image to the Flash memory. It takes about 10 second.



# 4 SYSTEM DESIGN

## OVERVIEW

The S3C2440X, SAMSUNG's 16/32-bit RISC microcontroller is cost-effective and high performance microcontroller solution for hand-held devices and general applications. The S3C2440X has the following integrated on-chip functions:

- 1.2V int., 2.5V/3.3V memory, 3.3V external I/O microprocessor with 16KB I-Cache/16KB D-Cache/MMU
- External memory controller (SDRAM control and Chip select logic)
- LCD controller (up to 4K color STN and 256K color TFT) with 1-ch LCD-dedicated DMA
- 4-ch DMAs with external request pins
- 3-ch UART (with IrDA1.0, 64-Byte Tx FIFO, and 64-Byte Rx FIFO) / 2-ch SPI
- 1-ch multi-master IIC-BUS/1-ch IIS-BUS controller
- SD Host interface version 1.0 & Multi-Media Card Protocol version 2.11 compatible
- 2-port USB host /1-port USB device (ver 1.1)
- 4-ch PWM timers & 1-ch internal timer
- Watch Dog Timer
- 130 general purpose I/O ports / 58 interrupt sources
- Power control: Normal, Slow, Idle and Power-off mode
- 8-ch 10-bit ADC and Touch screen interface
- RTC with calendar function
- On-chip clock generator with PLL



## APPLICABLE SYSTEM WITH S3C2440X

The S3C2440X, SAMSUNG's 16/32-bit RISC microcontroller offers various functions and high efficiencies. In addition to the high performance, the S3C2440X offers low current consumption, ensuring low costs. The followings are sample applications that can be designed with the S3C2440X:

- GPS
- Personal Data Assistance (PDA)
- Fish Finder
- Portable Game Machine
- Fingerprint Identification System
- Car Navigation System
- Smart Phone
- Mobile Information Terminal (MIT)
- Web Screen Phone
- Web Pad

## MEMORY INTERFACE DESIGN

### BOOT ROM DESIGN

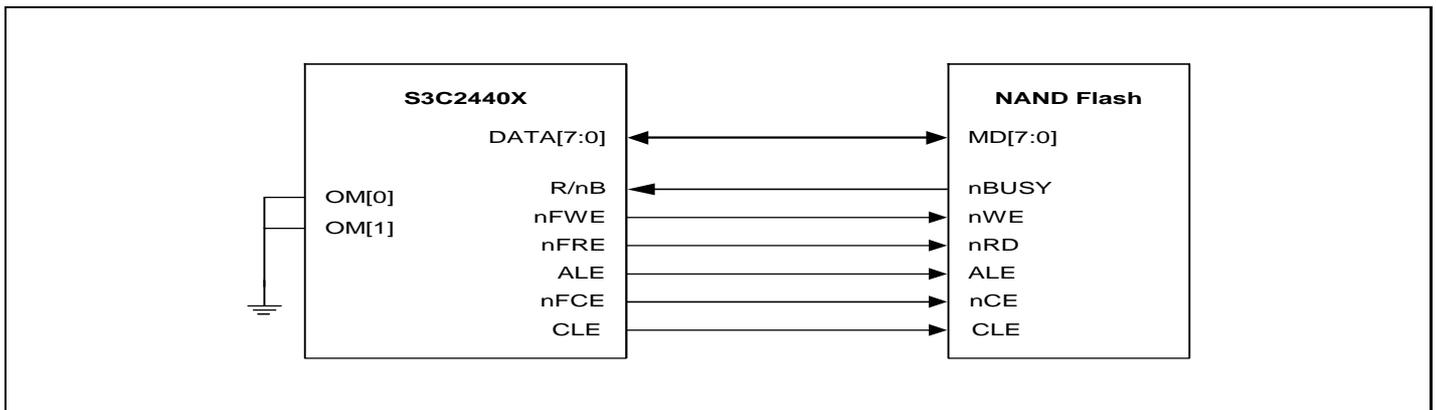
After the system reset, the S3C2440X accesses 0x00000000 address, configuring some system variables. Therefore, this special code (boot ROM image) should be located on the address 0x00000000. Bus width of boot ROM can be selected by setting OM[1:0] pins.

**Table 4-1. Data Bus Width for ROM Bank 0**

OM[1:0]	Data Bus Width
00	NAND boot
01	16-bit (half-word)
10	32-bit (word)
11	Test mode

### NAND BOOT DESIGN

Figure 4-1 shows a design with NAND boot.



**Figure 4-1. NAND Boot Design**

### MAKING NAND BOOT IMAGE

When making a NAND boot loader image, you can use the binary file that is made from compiling and linking.

## HALFWORD BOOT ROM DESIGN WITH BYTE EEPROM/FLASH

Figure 4-2 shows a design with half-word boot ROM with byte EEPROM/Flash.

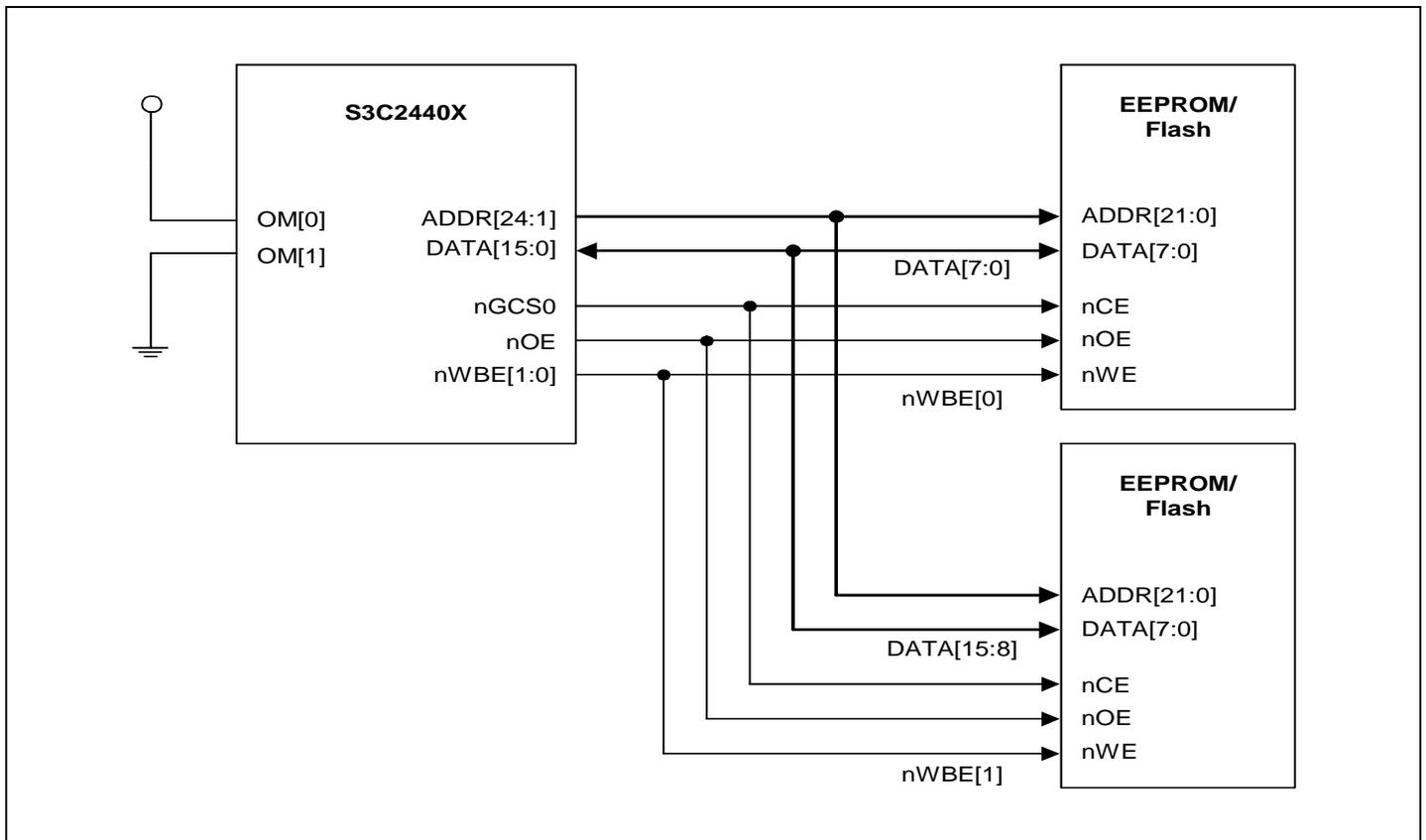


Figure 4-2. Half-word Boot ROM Design with Byte EEPROM/Flash

## MAKING HALFWORD ROM IMAGE WITH BYTE EEPROM/FLASH

When make half-word ROM image, you can split two image files, EVEN and ODD.

Table 4-2. Relationship ROM Image and Endian

	Big Endian	Little Endian
DATA[7:0]	Odd	Even
DATA[15:8]	Even	Odd

## HALFWORD BOOT ROM DESIGN WITH HALFWORD EEPROM/FLASH

Figure 4-3 shows a design with half-word boot ROM with byte EEPROM/Flash.

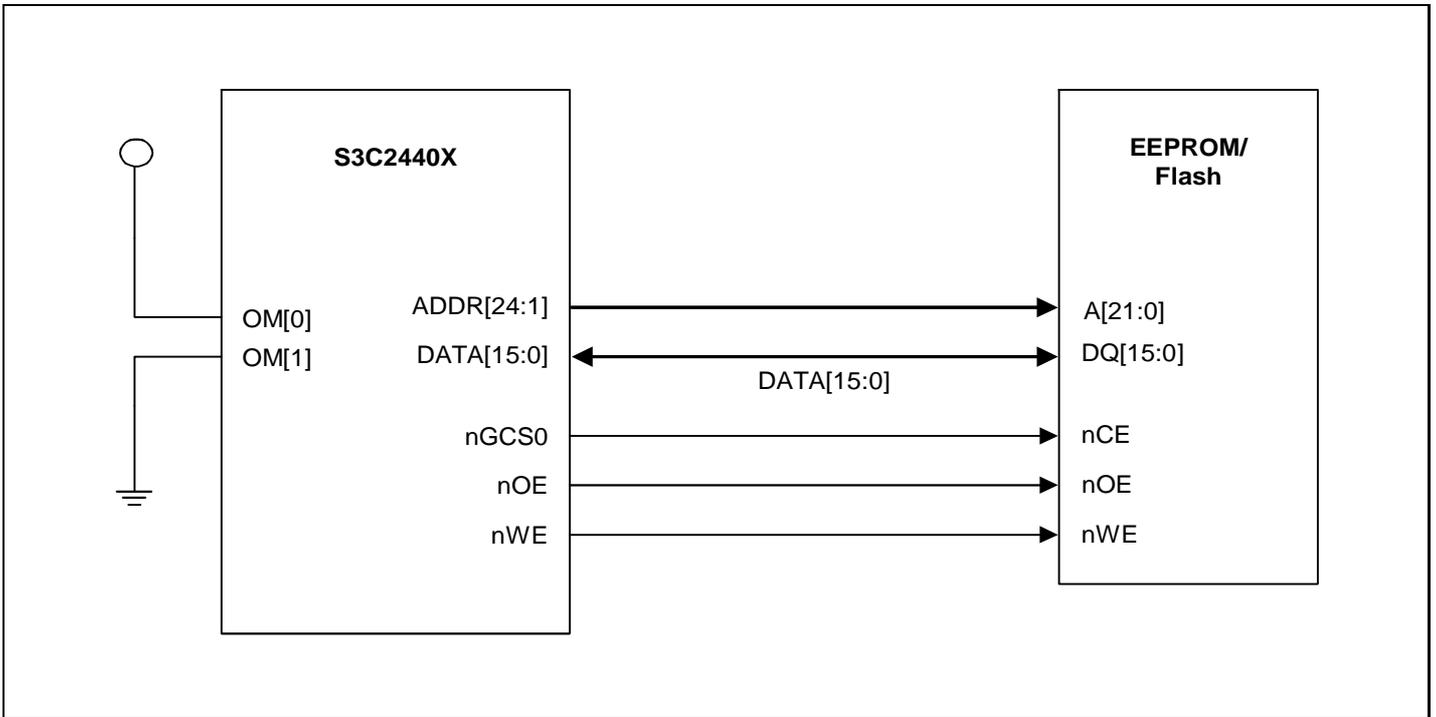


Figure 4-3. The Halfword Boot ROM Design with Halfword EEPROM/Flash

WORD BOOT ROM DESIGN WITH BYTE EEPROM/FLASH

Figure 4-4 shows a design with word boot ROM with byte EEPROM/Flash.

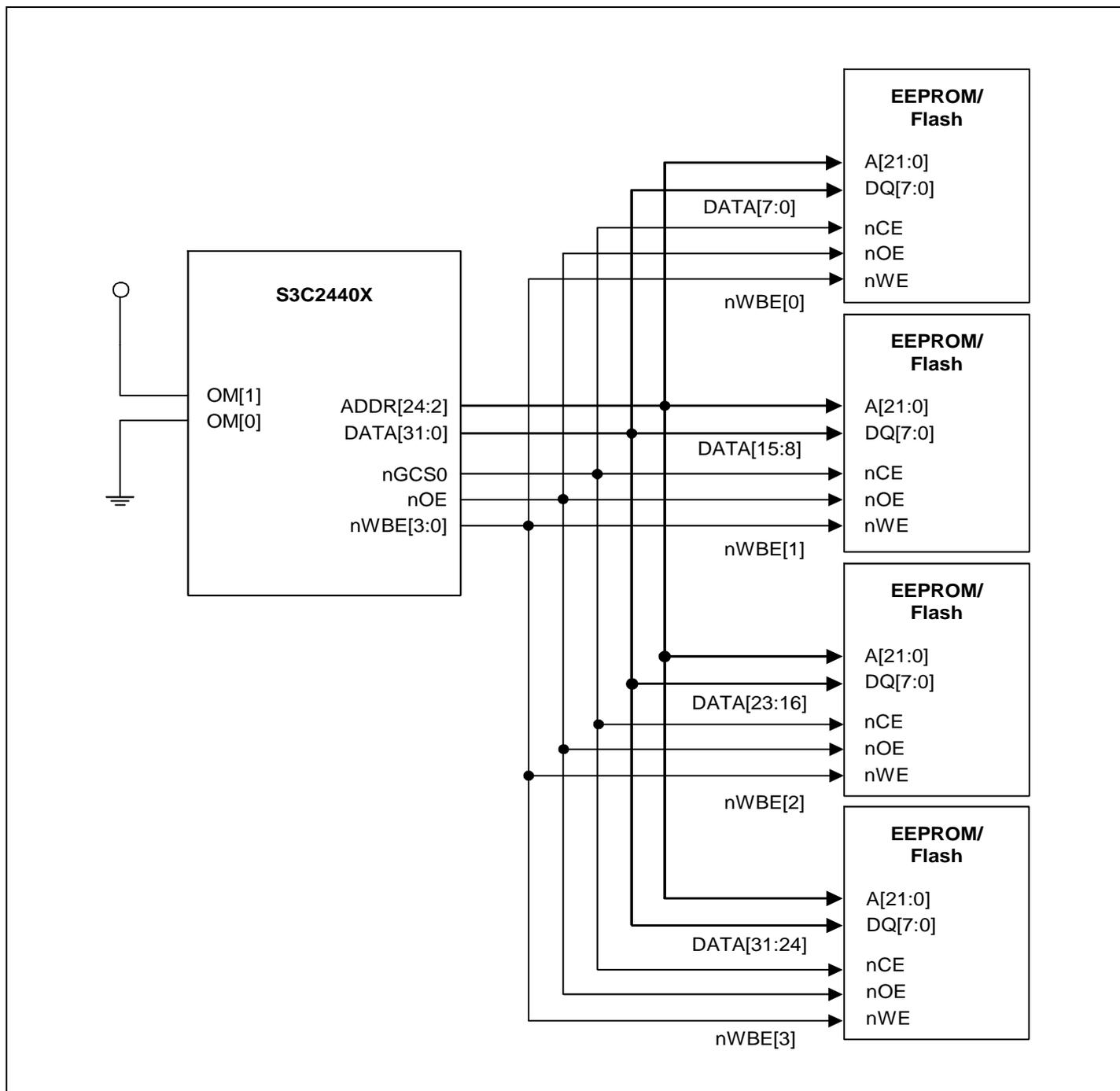


Figure 4-4. The Word Boot ROM Design with Byte EEPROM/Flash

## MAKING WORD ROM IMAGE WITH BYTE EEPROM/FLASH

When you make a word ROM image, you can split it into four image files.

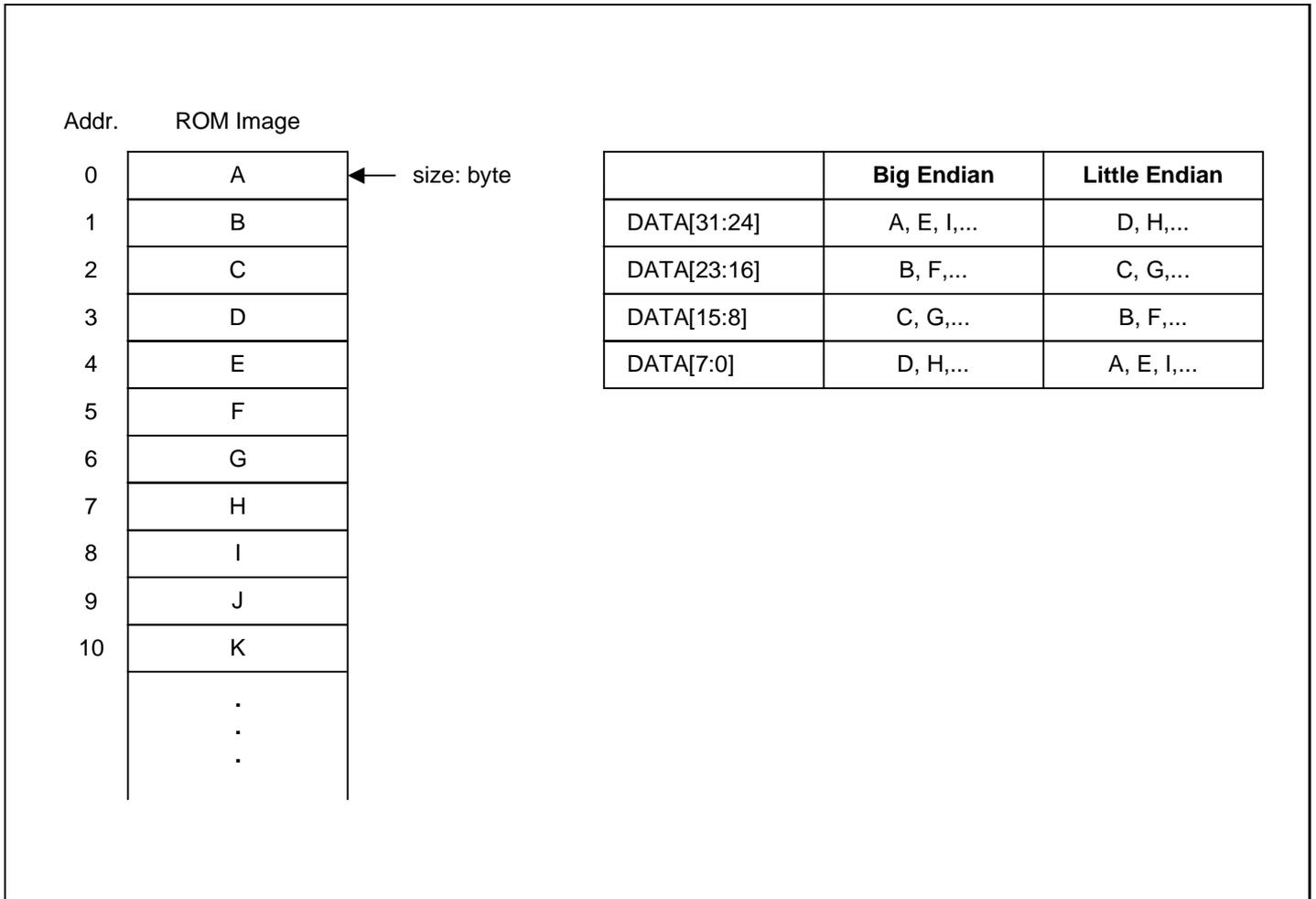


Figure 4-5. Relationship of ROM Image and Endian

## MEMORY BANK DESIGN AND CONTROL

The S3C2440X has six ROM/SRAM banks (including BANK0 for boot ROM) and two ROM/SRAM/SDRAM banks. The system manager on the S3C2440X can control access time, data bus width for each bank by S/W. The access time of ROM/SRAM banks and SDRAM banks is controlled by BANKCON0~5 and BANKCON6~7 control register on the system manager. The data bus width for each ROM/SRAM banks is controlled by BWSCON control register.

The ROM bank0 is used for boot ROM bank, therefore data bus width of bank0 is controlled by H/W. OM[1:0] is used for this purpose.

The control of BWSCON, BANKCON0-7, REFRESH, BANKSIZE, and MRSRB6/7 is performed during the system reset. A sample code for special register configuration is described below.

### Sample code for special register configuration

```

;Set memory control registers
LDR  r0,=SMRDATA
LDR  r1,=BWSCON    ;BWSCON Address
ADD  r2, r0, #52   ;End address of SMRDATA
0
LDR  r3, [r0], #4
STR  r3, [r1], #4
CMP  r2, r0
BNE  %B0
.
.
.
.
.
.
SMRDATA
DCD  0x22111120    ;BWSCON
DCD  0x00000700    ;GCS0
DCD  0x00000700    ;GCS1
DCD  0x00000700    ;GCS2
DCD  0x00000700    ;GCS3
DCD  0x00000700    ;GCS4
DCD  0x00000700    ;GCS5
DCD  0x00018005    ;GCS6 SDRAM(Trcd=3,SCAN=9)
DCD  0x00018005    ;GCS7 SDRAM(Trcd=3,SCAN=9)
DCD  0x008e0000+1113;Refresh(REFEN=1,TREFMD=0,Trp=2 clk,
;      Trc=7 clk, Tchr=3 clk,Ref CNT)
DCD  0x32          ;Bank size, 128MB/128MB
DCD  0x30          ;MRSR 6(CL=3 clk)
DCD  0x30          ;MRSR 7(CL=3 clk)

```

## ROM/SRAM BANK DESIGN

The ROM/SRAM banks 1-7 can have a variety of width of data bus, and the bus width is controlled by S/W. A sample design for ROM/SRAM bank 1-7 is shown in Figure 4-6, Figure 4-7, Figure 4-8 and Figure 4-9.

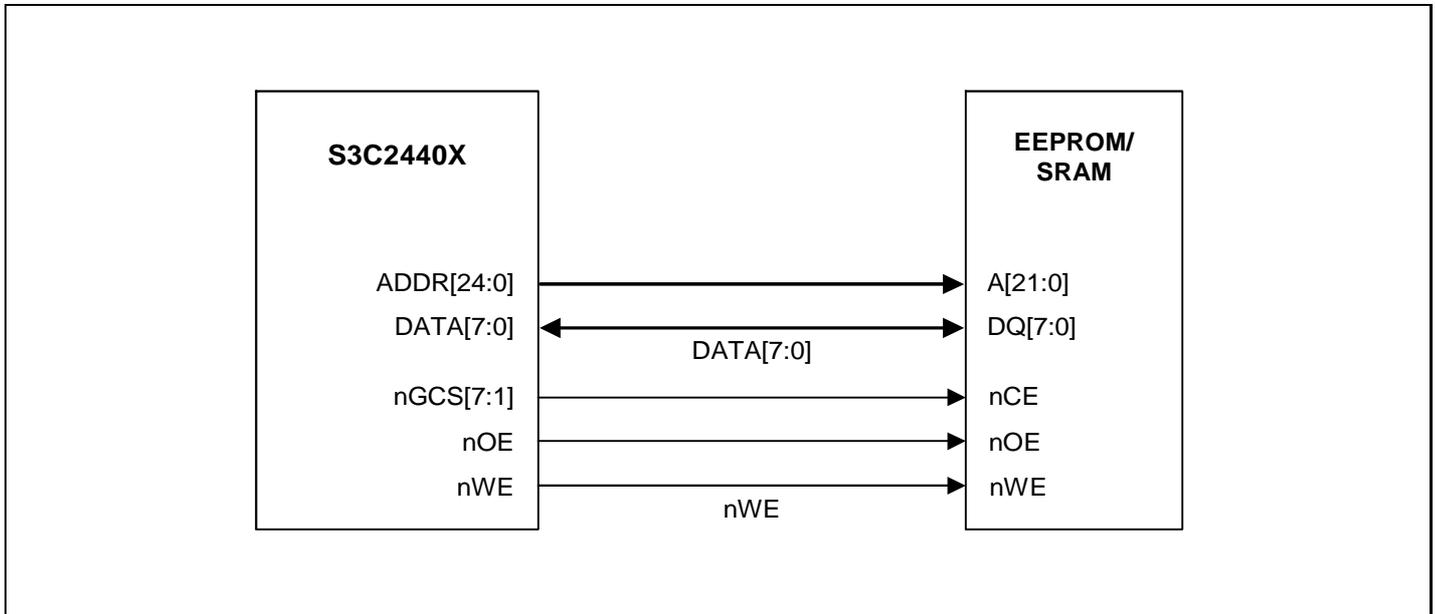


Figure 4-6. One-byte EEPROM/ SRAM Bank Design

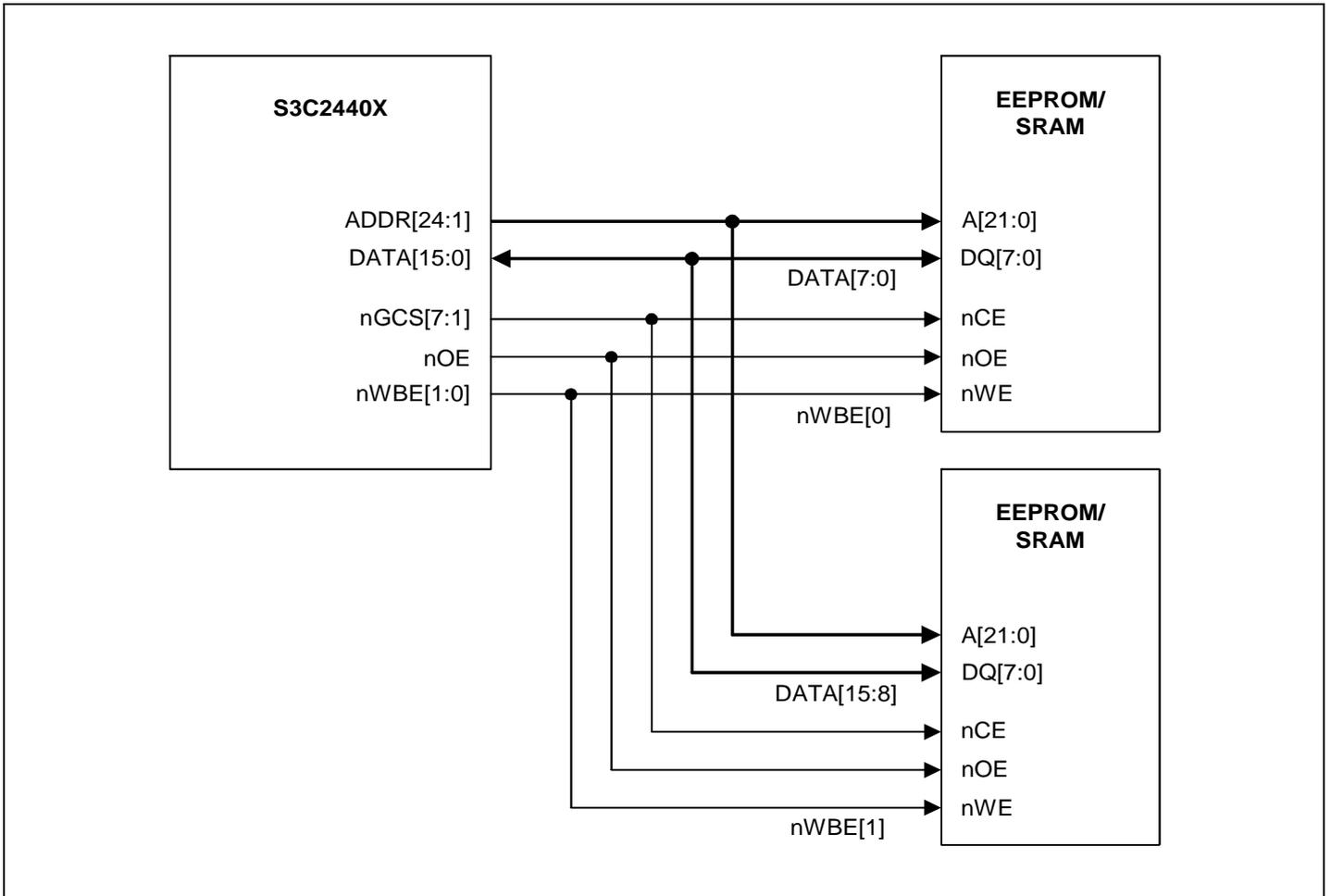


Figure 4-7. Halfword EEPROM/SRAM Bank Design

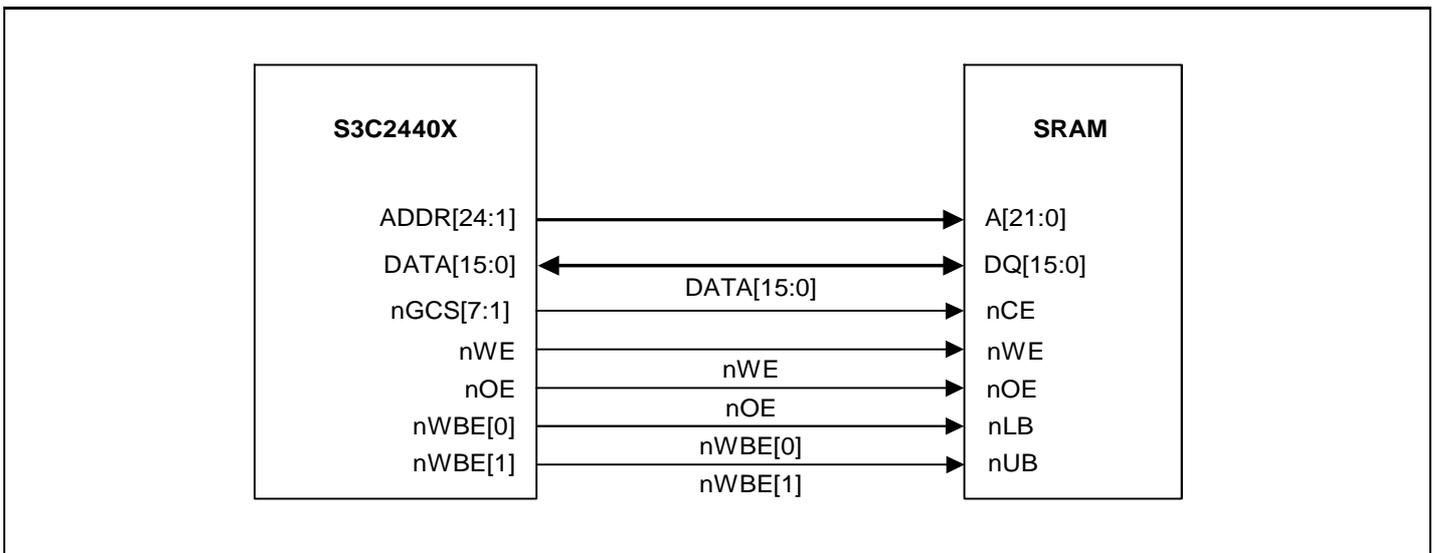


Figure 4-8. Halfword SRAM Bank Design with Halfword SRAM

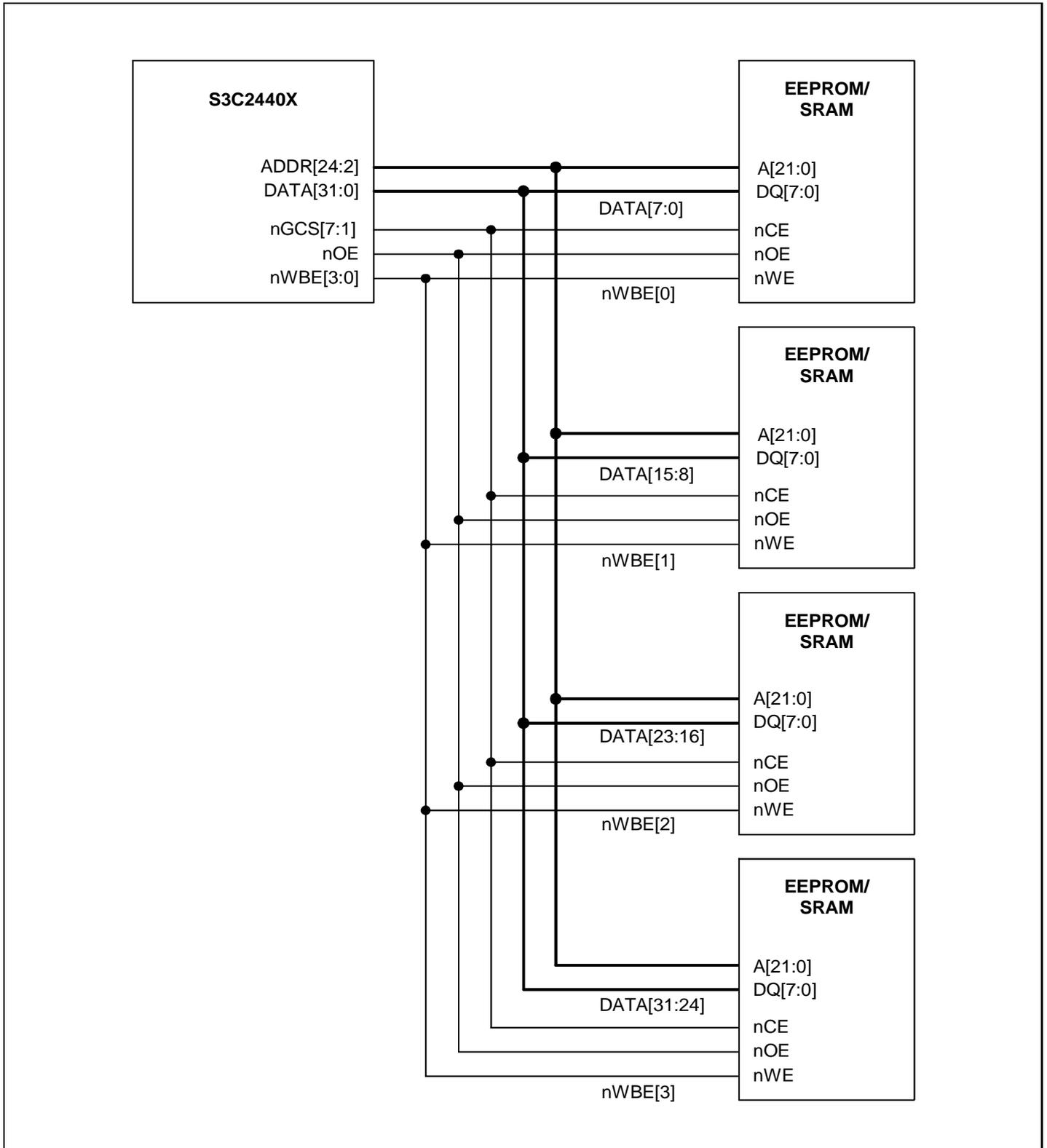


Figure 4-9. Word EEPROM/SRAM Bank Design

## SDRAM BANK DESIGN FOR S3C2440X

Table 4-3. SDRAM Bank Address configuration

Bank Size	Bus Width	Base Component	Memory Configuration	Bank Address
2MByte	x8	16Mbit	(1M x 8 x 2Bank) x 1	A20
	x16		(512K x 16 x 2B) x 1	
4MB	x8	16Mb	(2M x 4 x 2B) x 2	A21
	x16		(1M x 8 x 2B) x 2	
	x32		(512K x 16 x 2B) x 2	
8MB	x16	16Mb	(2M x 4 x 2B) x 4	A22
	x32		(1M x 8x 2B) x 4	
	x8	64Mb	(4M x 8 x 2B) x 1	A[22:21]
	x8		(2M x 8 x 4B) x 1	
	x16		(2M x 16 x 2B) x 1	
	x16		(1M x 16 x 4B) x 1	
	x32		(512K x 32 x 4B) x 1	
16MB	x32	16Mb	(2M x 4 x 2B) x 8	A23
	x8	64Mb	(8M x 4 x 2B) x 2	
	x8		(4M x 4 x 4B) x 2	A[23:22]
	x16		(4M x 8 x 2B) x 2	A23
	x16		(2M x 8 x 4B) x 2	A[23:22]
	x32		(2M x 16 x 2B) x 2	A23
	x32		(1M x 16 x 4B) x 2	A[23:22]
	x8		128Mb	
	x16	(2M x 16 x 4B) x 1		
32MB	x16	64Mb	(8M x 4 x 2B) x 4	A24
	x16		(4M x 4 x 4B) x 4	A[24:23]
	x32		(4M x 8 x 2B) x 4	A24
	x32		(2M x 8 x 4B) x 4	A[24:23]
	x16	128Mb	(4M x 8 x 4B) x 2	
	x32		(2M x 16 x 4B) x 2	
	x8	256Mb	(8M x 8 x 4B) x 1	
	x16		(4M x 16 x 4B) x 1	

Table 4-3. SDRAM Bank Address configuration (Continued)

Bank Size	Bus Width	Base Component	Memory Configuration	Bank Address
64MB	x32	128Mb	(4M x 8 x 4B) x 4	A[25:24]
	x16	256Mb	(8M x 8 x 4B) x 2	
	x32		(4M x 16 x 4B) x 2	
	x8	512Mb	(16M x 8 x 4B) x 1	
128MB	x32	256Mbit	(8M x 8 x 4Bank) x 4	A[26:25]
	x8	512Mb	(32M x 4 x 4B) x 2	
	x16		(16M x 8 x 4B) x 2	

The required SDRAM interface pin is  $\overline{\text{CKE}}$ ,  $\text{SCLK}$ ,  $\overline{\text{nSCS}}[1:0]$ ,  $\overline{\text{nSCAS}}$ ,  $\overline{\text{nSRAS}}$ ,  $\text{DQM}[3:0]$  and  $\text{ADDR}[12]/\overline{\text{AP}}$ . The sample design with SDRAM is shown in Figure 4-10 and Figure 4-11.

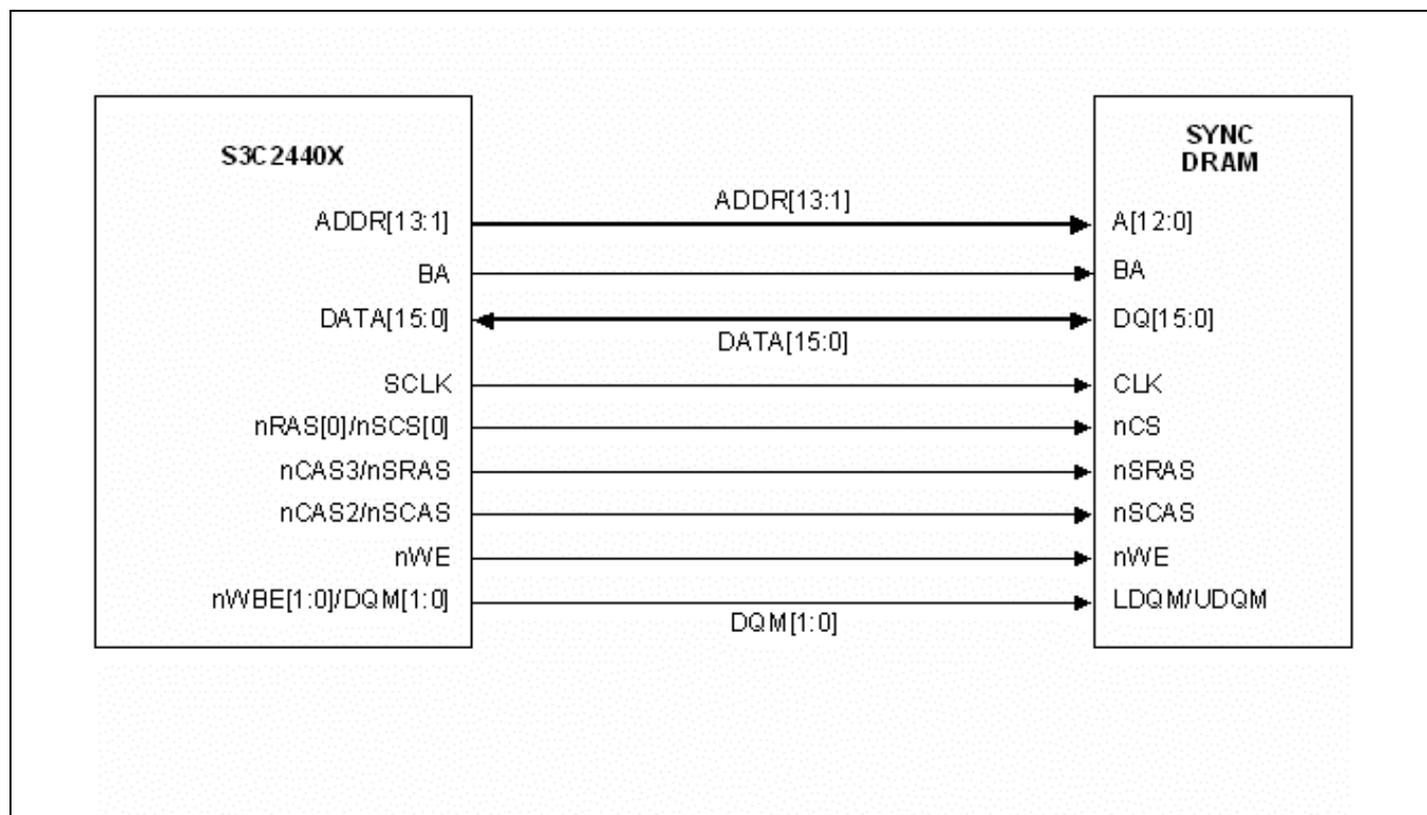


Figure 4-10. Halfword SDRAM Design with Halfword Component

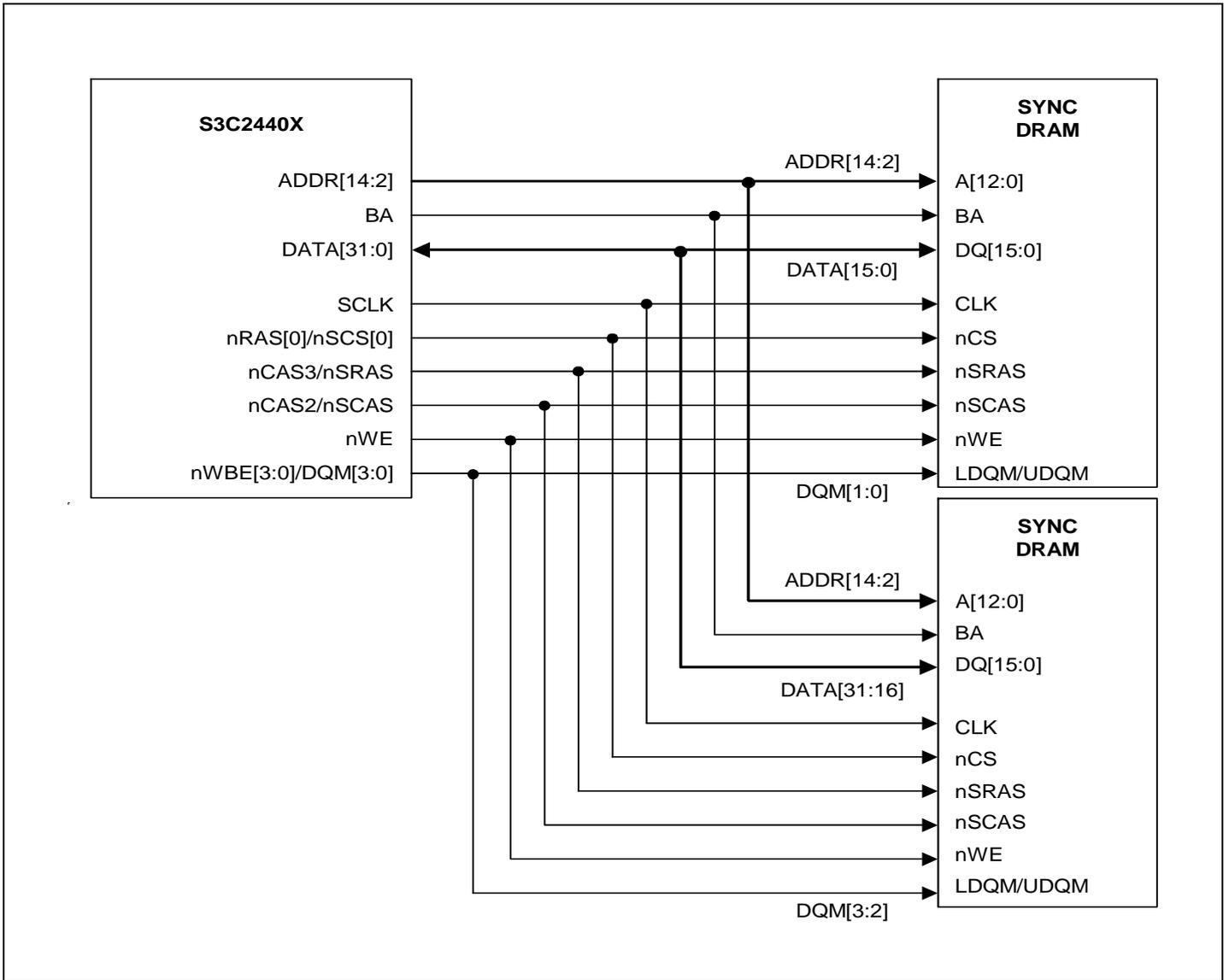


Figure 4-11. Word SDRAM Design with Half-word Component

## GPC CARD (GPCMCIA) INTERFACE APPLICATION USING CL-PD6710 (CIRRUS LOGIC)

The GPC card (GPCMCIA card) can be interfaced with S3C2440X using following components:

- CL-PD6710 from Cirrus logic
- TPS2211 from Texas Instruments

We tested the GPC card interface by accessing the card information structure (CIS) in the modem card as Figure 4-12, using following test code.

File Name	File Descriptions
pd6710.h	CL-PD6710 register definitions
pd6710.c	CL-PD6710 GPC Card program

```

DNW v0.49 [COM1,115200bps][USB:x]
Serial Port USB Port Configuration Help
[PD6710 test for reading pc_card CIS]
Insert PC card!!!
PC card interrupt is occurred.
PC card interrupt is occurred.
Card is inserted.
3.3V card is detected.
PC card interrupt is occurred.
[Card Information Structure]
cisEnd=0^a6
 1, 4,df,4a, 1,ff,1c, 4, 2,d9, 1,ff,18, 2,df, 1,//...J.....
20, 4, 7,c0, 0, 0,15,20, 4, 1,53,41,4d,53,55,4e,// ..... ..SAMSUN
47,20,20,20,20,20,20, 0,53,43,46,43,2d,56,45,52,//G      .SCFC-VER
31,2e,30,20,20, 0, 0,ff,21, 2, 4, 1,22, 2, 1, 1,//1.0 ...!"...
22, 3, 2, c, f,1a, 5, 1, 3, 0, 2, f,1b, 8,c0,c0,/".....
a1, 1,55, 8, 0,20,1b, 6, 0, 1,21,b5,1e,4d,1b, a,//..U.. ....!"..M..
c1,41,99, 1,55,64,f0,ff,ff,20,1b, 6, 1, 1,21,b5,//.A..Ud... ....!"..
1e,4d,1b, f,c2,41,99, 1,55,ea,61,f0, 1, 7,f6, 3,//.M...A..U.a.....
 1,ee,20,1b, 6, 2, 1,21,b5,1e,4d,1b, f,c3,41,99,//.. ....!"..M...A.
 1,55,ea,61,70, 1, 7,76, 3, 1,ee,20,1b, 6, 3, 1,//.U.ap..v... ....
21,b5,1e,4d,14, 0,ff,

```

Figure 4-12. GPC Card CIS Access Example on S3C2440X

### 10BASE-T ETHERNET CONTROLLER (CS8900A) INTERFACE

The 10BASE-T Ethernet can be supported on S3C2440X using following components:

- CS-8900A from Cirrus logic
- XFMRS XF10B11A-COMB1-2S is Ethernet RJ45 with transformer.

### AUDIO CODEC (UDA1341TS) CONNECTION WITH S3C2440X

The S3C2440X IIS interface example circuit is as follows:

- UDA1341TS from Philips Semiconductors.
- The L3 interface of Philips (L3MODE, L3CLOCK and L3DATA) is realized by general I/O port.
- Refer to the sample code of audio application which plays GPCM file.

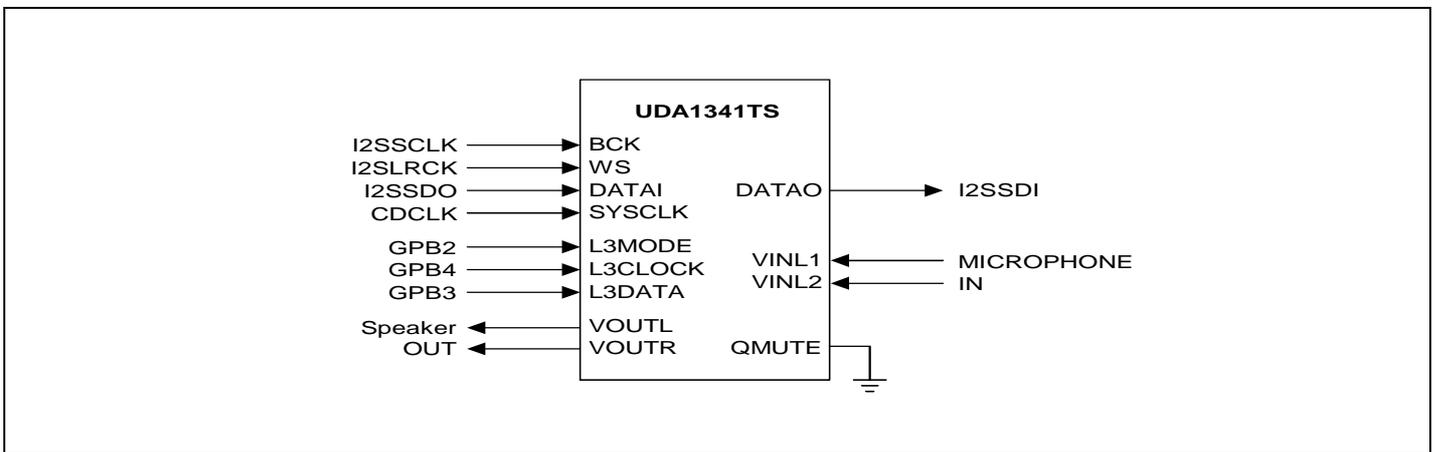


Figure 4-13. UDA1341TS Connection with S3C2440X

## LCD CONNECTION WITH S3C2440X

The S3C2440X LCD interface example circuit is as follows:

- UG-32F04 (320x240 mono STN LCD) from SAMSUNG DISPLAY DEVICES CO., LTD. (refer to Figure 4-14)
  - TL497CAN can be used to make VEE (-25V).
- UG-24U03A (320x240 mono STN LCD) from SAMSUNG DISPLAY DEVICES CO., LTD. (refer to Figure 4-15)
  - VEE is generated by the circuit on LCD module.
  - VL is 2.4V typically.
  - DISPON H: display on, L: display off
  - nEL\_ON H: EL off L: EL on
- KHS038AA1AA-G24 (256 color STN LCD) from KYOCERA Co. (refer to Figure 4-16)
  - DISP signal can be made using I/O port, or power control circuit or nRESET circuit.
  - V1-V5 can be made using the power circuit recommended by the LCD specification.
- LTS350Q1-PE1 (256K color TFT LCD) from SAMSUNG ELECTRONICS CO., LTD. (refer to Figure 4-17)
  - VDD\_LCDI is typically 3.3V.
- LP104V2-W (262,144 color TFT LCD, 10.4") from LG Philips (refer to Figure 4-18)
  - VDD\_LCDI is typically 3.3V.
- V16C6448AB (640x480 TFT LCD) from PRIMEVIEW (refer to Figure 4-19)
  - VDD\_LCDI, VD and control signal are typically 5.V.

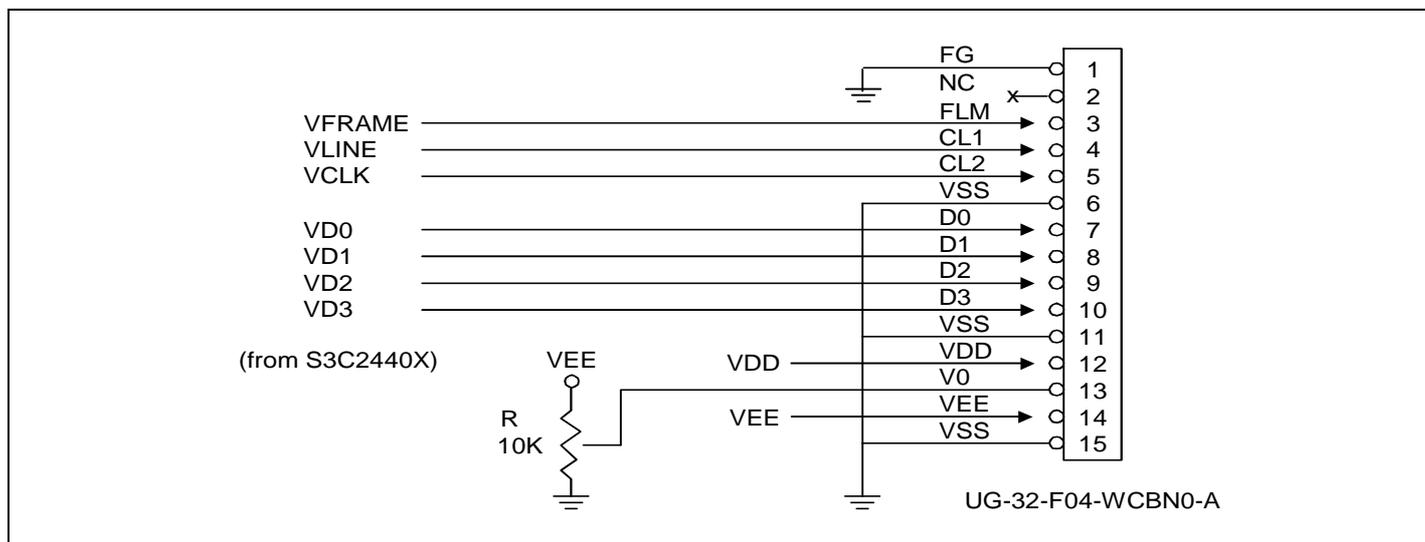


Figure 4-14. UG-32F04 Connection with S3C2440X (320x240 Mono STN LCD)

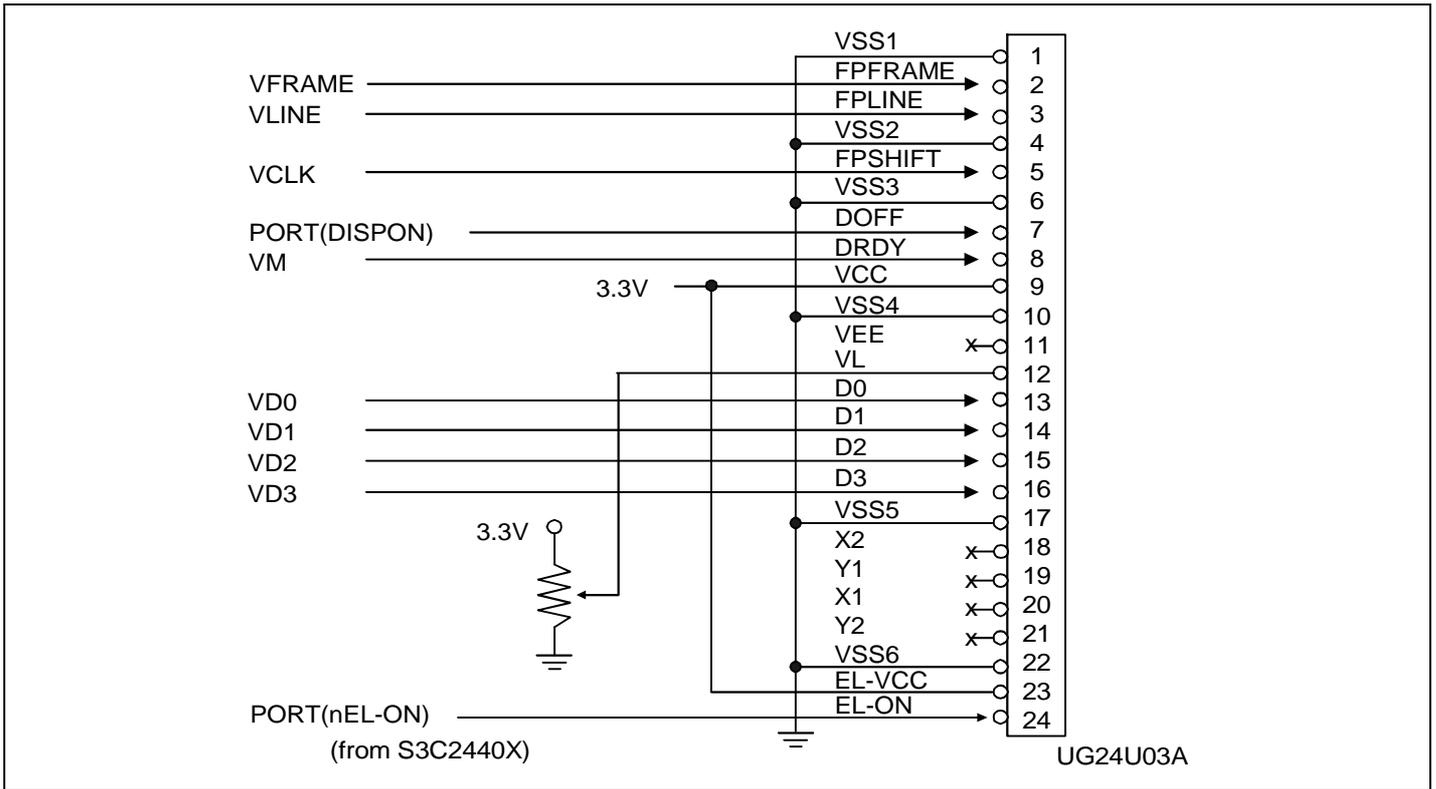


Figure 4-15. UG24U03A Connection with S3C2440X (320x240 Mono STN LCD)

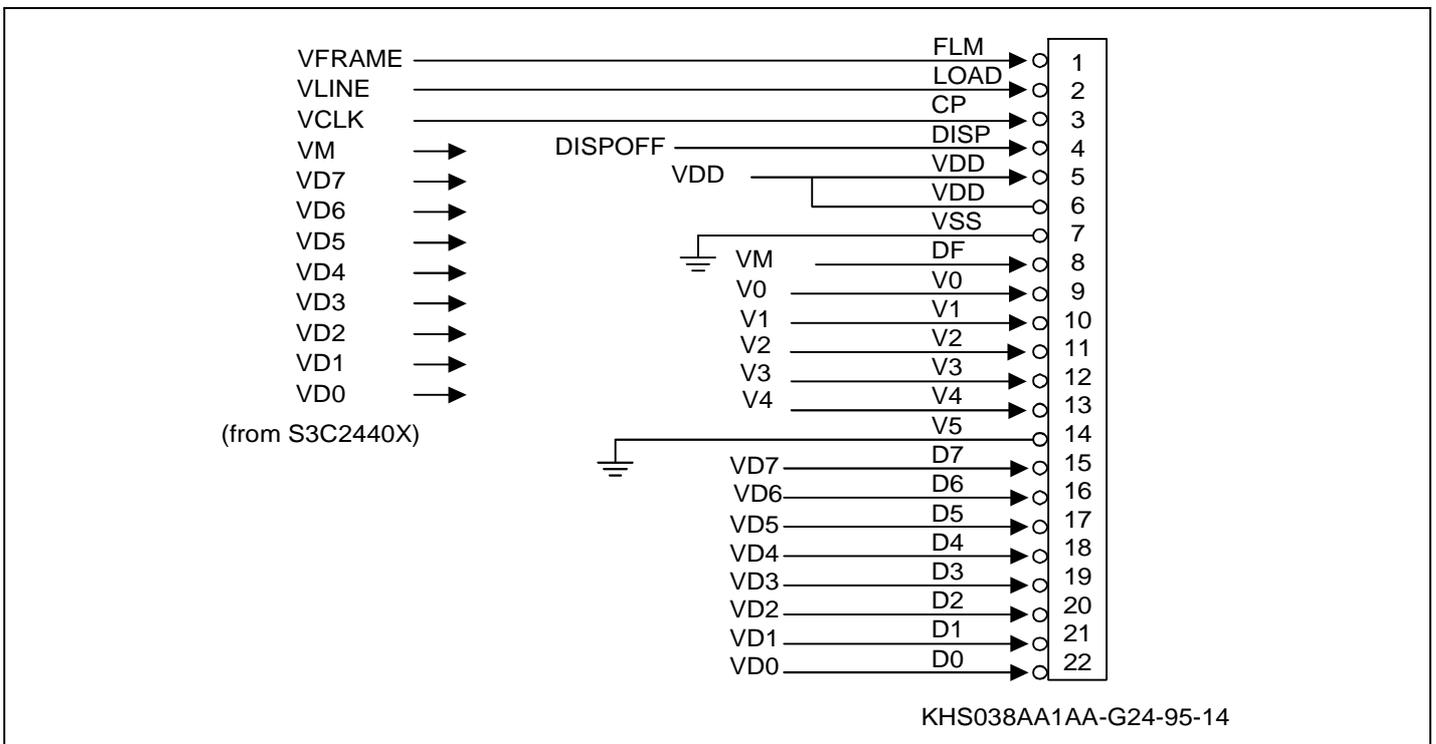


Figure 4-16. KHS038AA1AA-G24 Connection with S3C2440X (256 Color STN LCD)

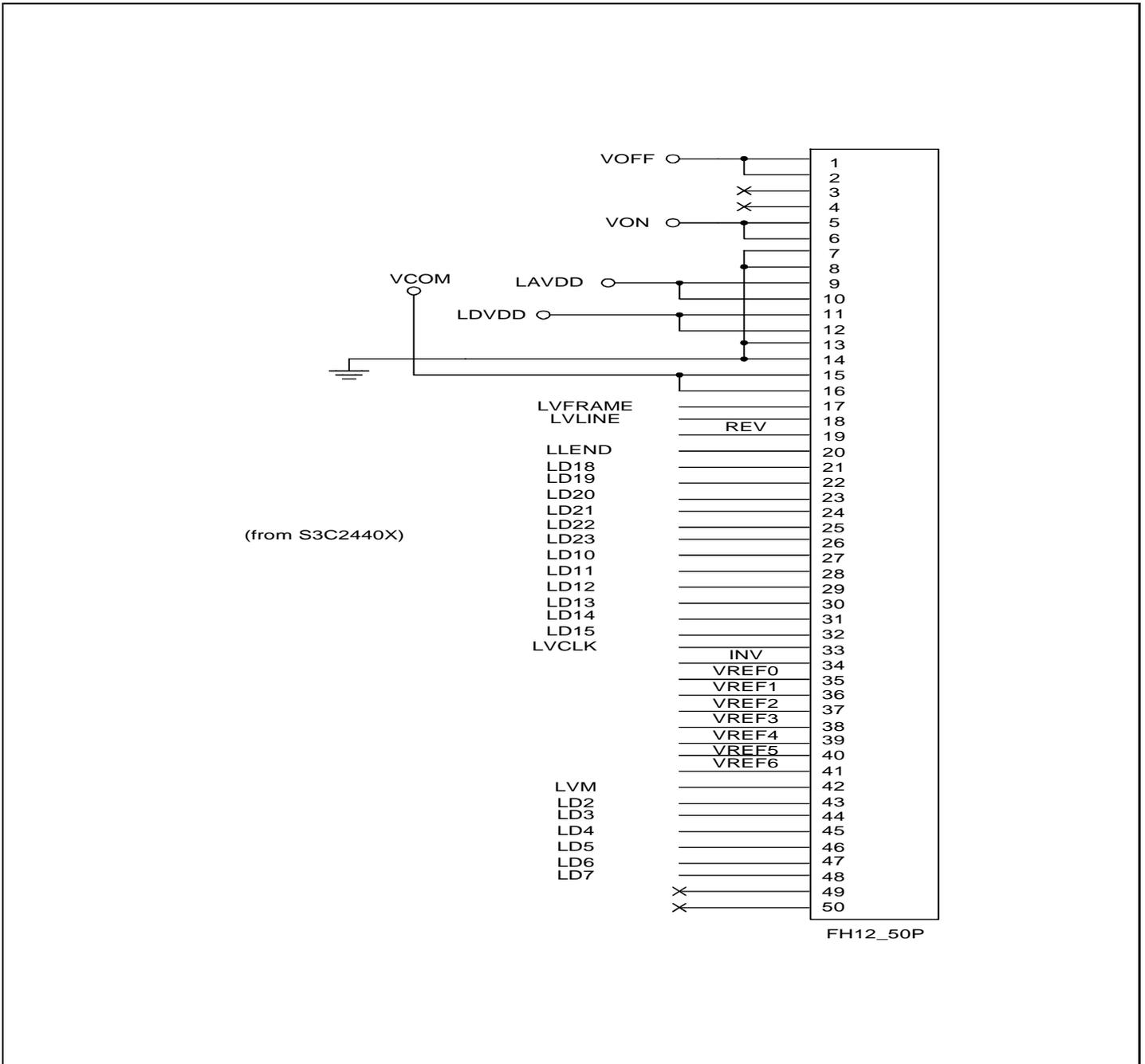


Figure 4-17. LTS350Q1-PE1 Connection with S3C2440X (Samsung 3.5" Transflective TFT LCD)



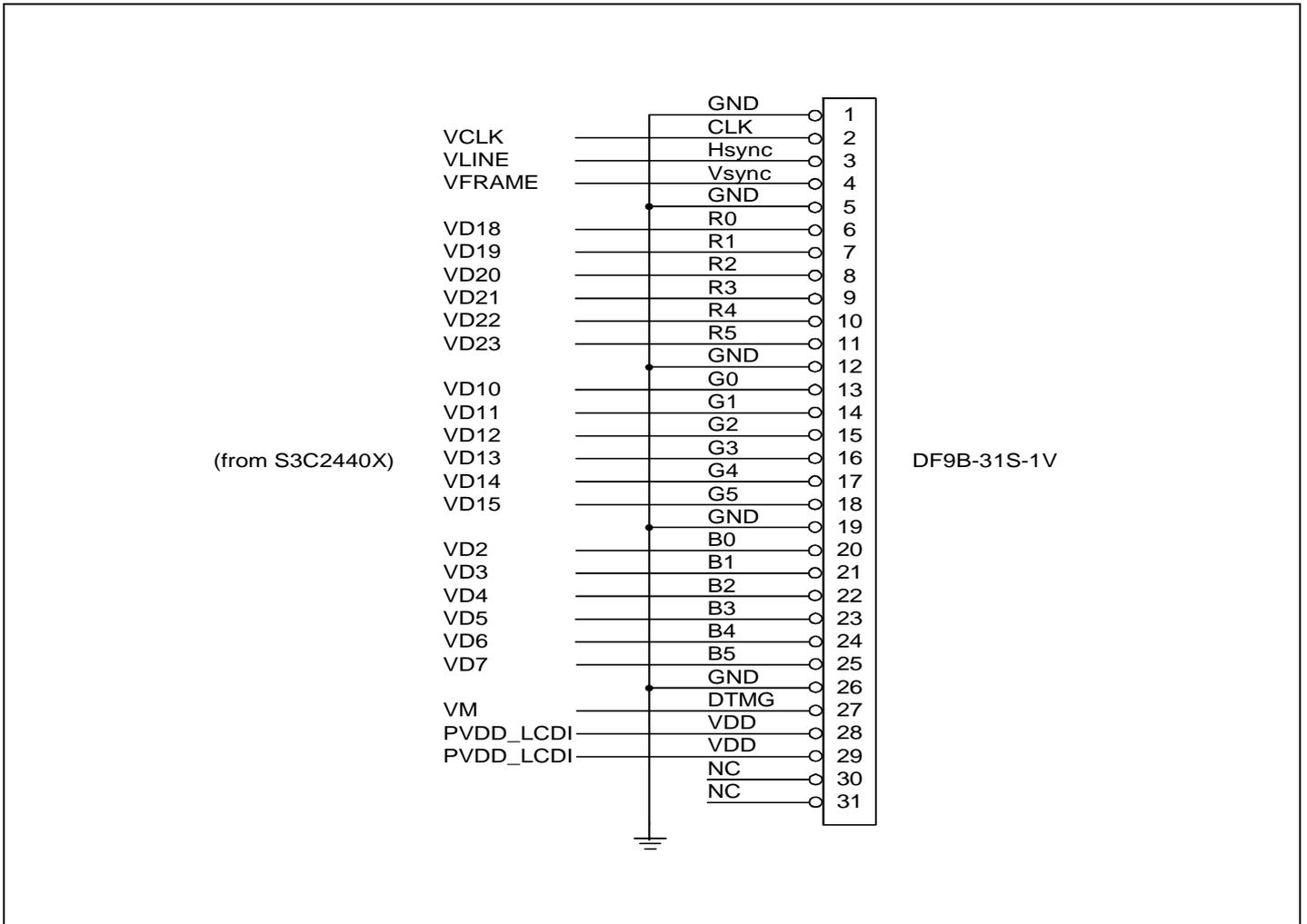


Figure 4-18. LP104V2-W Connection with S3C2440X (LG Philips 10.4" TFT LCD)

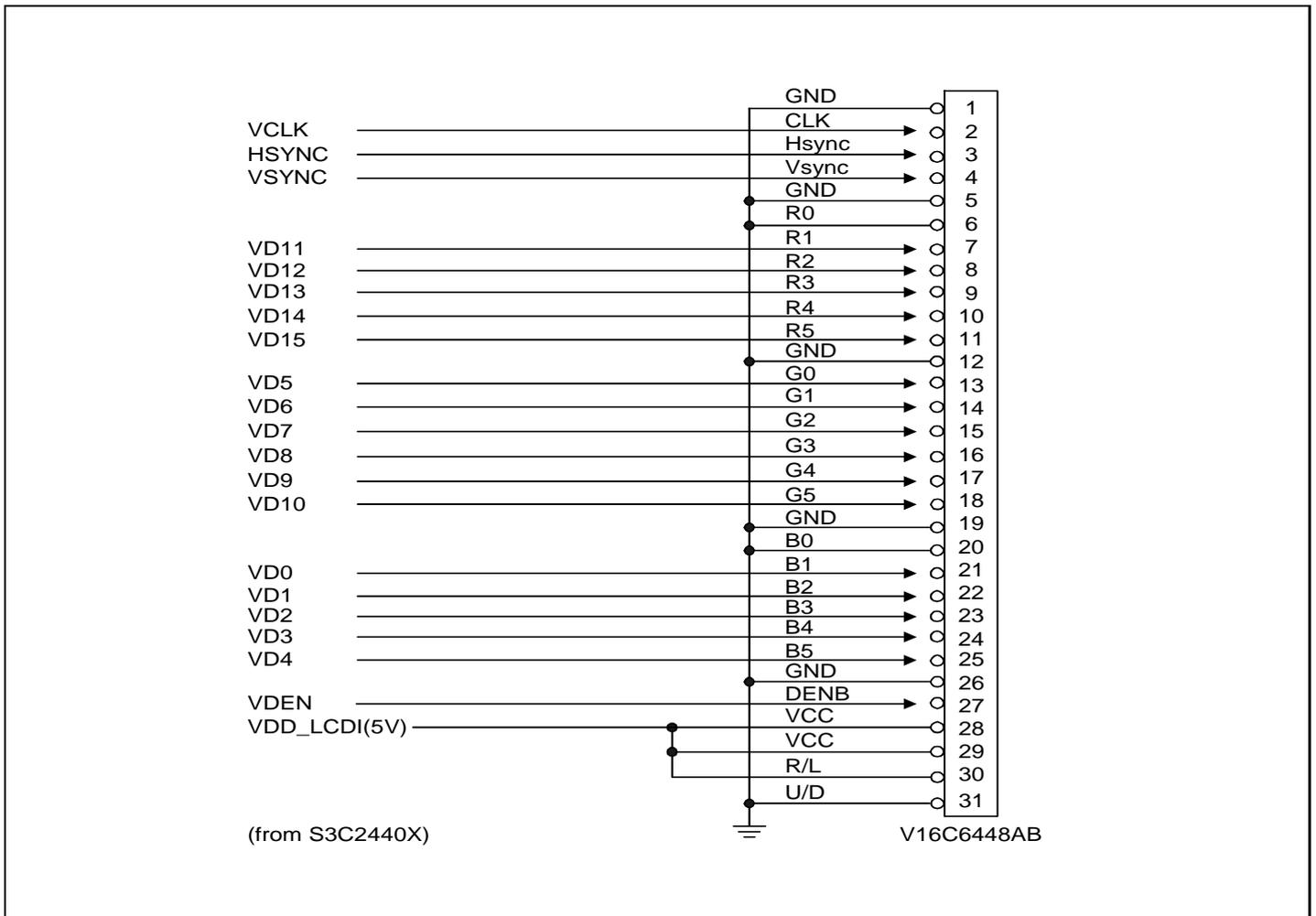


Figure 4-19. V16C6448AB Connection with S3C2440X (TFT LCD)

### TOUCH SCREEN PANEL (TSP) INTERFACE CIRCUIT WSSITH S3C2440X

Typically, the TSP consists of two plane resistors (x-axis plane resistor and y-axis plane resistor). In this reason, TSP has four terminals. To read X coordinate, Q1 and Q2 are turned on while Q3 and Q4 is turned off. Therefore, X coordinate value can be read out from AIN7 by ADC. To read Y coordinates, Q3 and Q4 is turned on while Q1 and Q2 is turned off. So, ADC can read out Y coordinate value from AIN5. The INT\_TC can be used to check whether the TSP is touched.

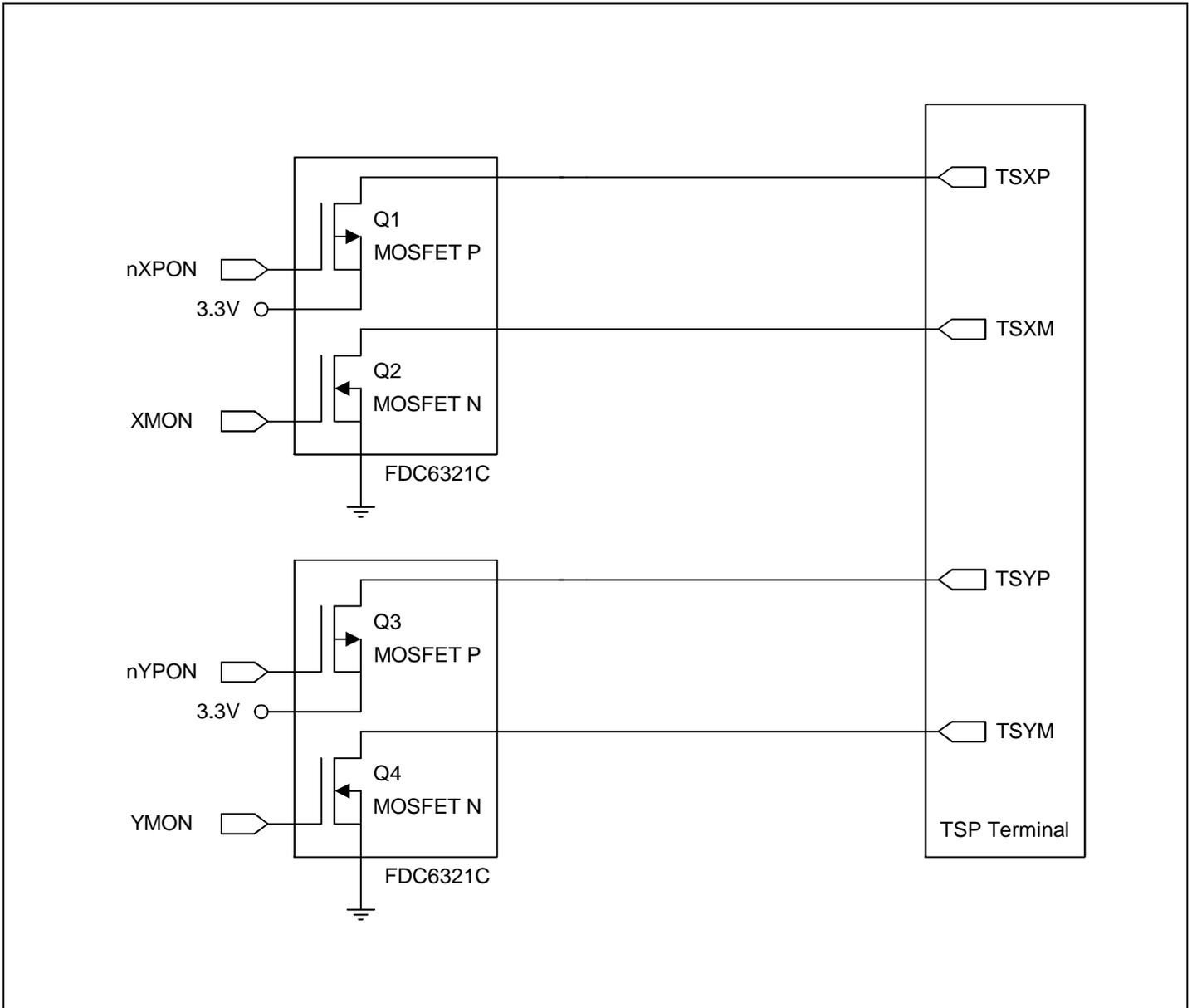


Figure 4-20. TSP Interface Circuit with S3C2440X

## SYSTEM DESIGN WITH DEBUGGER SUPPORT

### MULTI-ICE

The S3C2440X has an Embedded ICE logic that provides debug solution from ARM. MULTI-ICE enables you to debug software running on the S3C2440X. Embedded ICE logic is accessed through the Test Access Port (TAP) controller on the S3C2440X using the JTAG interface.

#### JTAG port for Embedded ICE Interface

When you build a system with the S3C2440X Embedded ICE interface, you should design a JTAG port for MULTI-ICE interface. Usually, the interface connector is a 20-way box header, and this plug is connected to the Embedded ICE logic interface module using 20-way IDC socket.

The JTAG port signals, nTRST, TDI, TMS and TCK have to be connected to pulled-up register (10K ohm) externally.

The pin configuration and a sample design are described in Figure 4-21 and Figure 4-22, respectively.

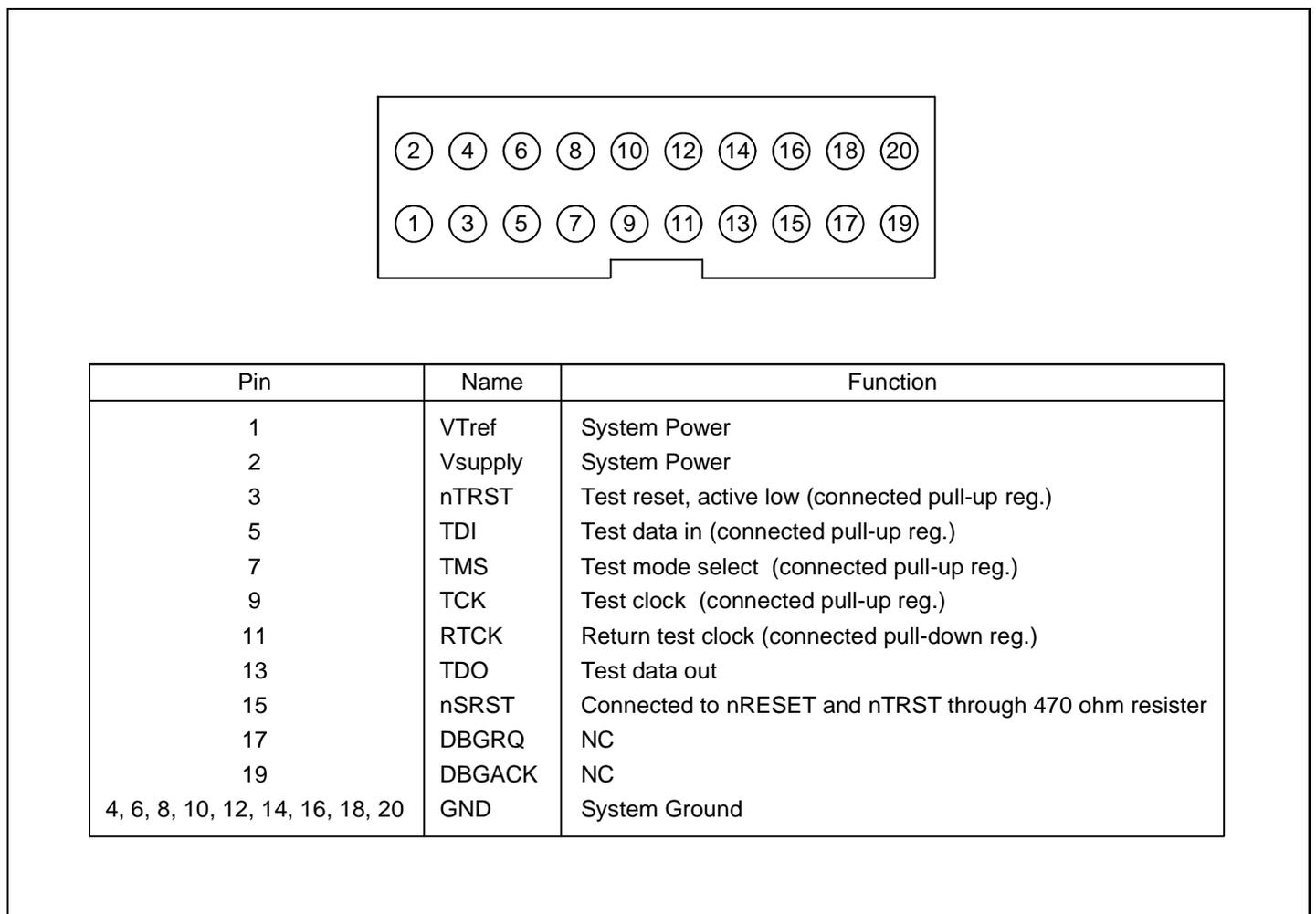


Figure 4-21. MULTI-ICE Interface of JTAG Connector

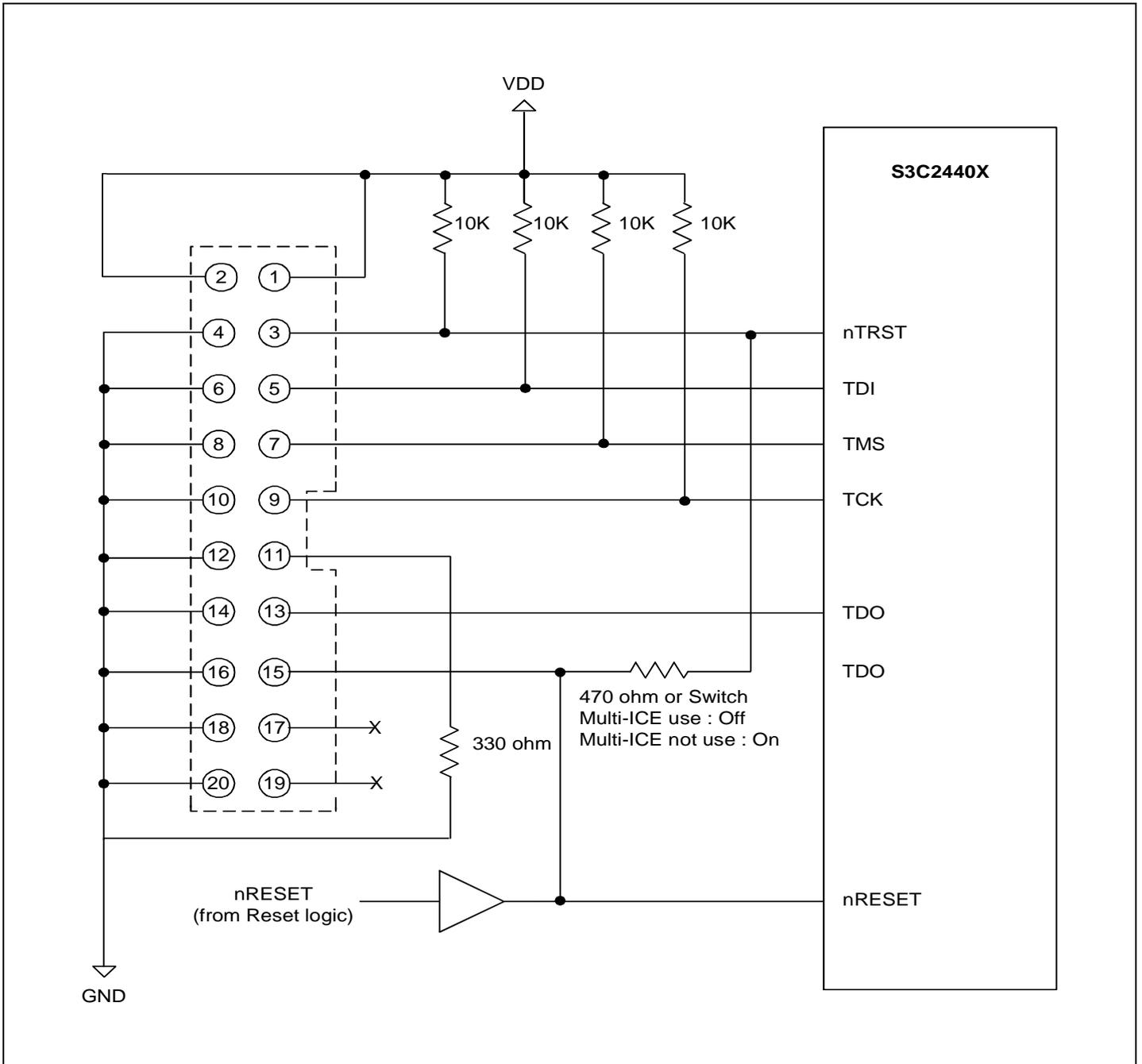


Figure 4-22. MULTI-ICE Interface Design Example

**CHECK ITEMS FOR SYSTEM DESIGN WITH S3C2440X**

When you design a system with the S3C2440X, you should check a number of items to build a good system. The check items are described below.

- The OM[3:0] pin has to be configured.
- If EXTCLK pin is used for MPLL and UPLL, XTIpII has to be connected to VDD. If XTIpII pin is used for MPLL and UPLL, EXTCLK has to be connected to VDD.
- If an input pin is unused, connect the pin to VDD or GND. If the pin is floated, S3C2440X may not operate.



## NOTES

Find price and stock options from leading distributors for s3c2440x on Findchips.com:

<https://findchips.com/search/s3c2440x>