

# 1

## ABOUT SMDK41100 BOARD

### SYSTEM OVERVIEW

SMDK41100 (Samsung MCU Development Kit) for S3C44B0X is a platform that is suitable for code development of SAMSUNG's S3C44B0X 16/32-bit RISC microcontroller (General ARM) for hand-held device and general applications.

S3C44B0X consists of 16-/32-bit RISC (ARM7TDMI) CPU core, 8KB cache, optional internal SRAM, LCD controller (up to 256 color DSTN), 2-ch UART with hand-shake(IRDA1.0,16-byte FIFO), 4-ch DMA, System manager (chip select logic, FP/ EDO/SDRAM controller), 6-ch timers with PWM, 71-bit general purpose I/O ports, RTC, 8-ch 10-bit ADC, IIC-BUS interface, IIS-BUS interface, Sync. SIO interface and PLL for clock.

SMDK41100 (Samsung MCU Development Kit) consist of S3C44B0X, boot EEPROM (Flash ROM), EDO DRAM, SDRAM, LCD connect, two serial communication ports, configuration switches, RTC, JTAG interface and status LEDs.

### SMDK41100 OVERVIEW

The SMDK41100 (Samsung MCU Development Kit) shows the basic system-based hardware design which uses the S3C44B0X. It can evaluate the basic operations of the S3C44B0X and develop codes for it as well.

When the S3C44B0X is contained in the SMDK41100 (Samsung MCU Development Kit), you can use an in-circuit emulator (ICE).

This allows you to test and debug a system design at the processor level. In addition, the S3C44B0X with embeddedICETM capability can be debugged directly using the EmbeddedICE Interface.

The SMDK41100 (Samsung MCU Development Kit) function blocks are shown in Figure 1-1.

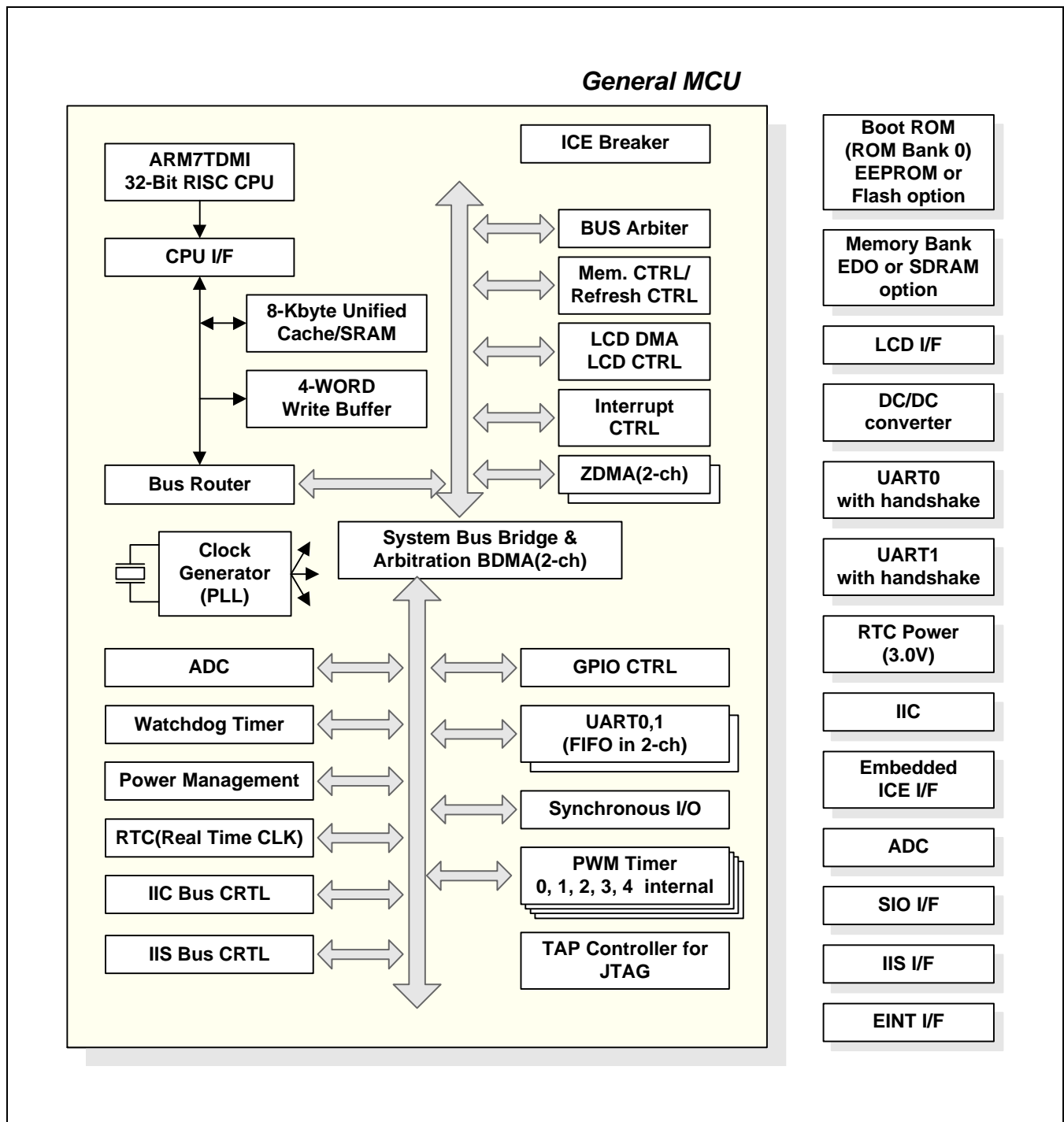


Figure 1-1. SMDK41100 Function Block Diagram

**FEATURES**

- S3C44B0X: 16/32-bit RISC microcontroller
- X-tal operation
- Boot ROM: 512 K bit, 1 M bit, 8 M bit, support half-word size boot ROM
- DRAM: 4 M x 16 EDO DRAM support
- SDRAM: 4 M x 16 SDRAM support
- 8-bit LCD connector
- General I/O: GIP 2-port & GOP 2-port
- Two-channel UART
- EmbeddedICE™ Interface
- RTC X-tal input & power logic
- IIC : KS24C080 support
- ADC I/F
- SIO I/F
- IIS I/F
- EINT I/F

## CIRCUIT DESCRIPTION

SMDK41100 (Samsung MCU Development Kit) consists of logic components, several control/status display block, and a debug interface block. SMDK41100's detailed block diagram, and its components are shown in figure 1-3.

## POWER SUPPLY

SMDK41100 (Samsung MCU Development Kit) is designed to operate at 2.5V for core and 3.3V for I/O. Power to SMDK41100 (Samsung MCU Development Kit) is supplied through a DC jack power adapter which supports the voltage between 5V and 9V and drives the current at least 850 mA .

SMDK41100 has distributed power plane, with power going separately to the MCU and the main power plane. For this reason, power jumpers J1, J2, J3 and JP5 are inserted (see Figure 1-2 ) .

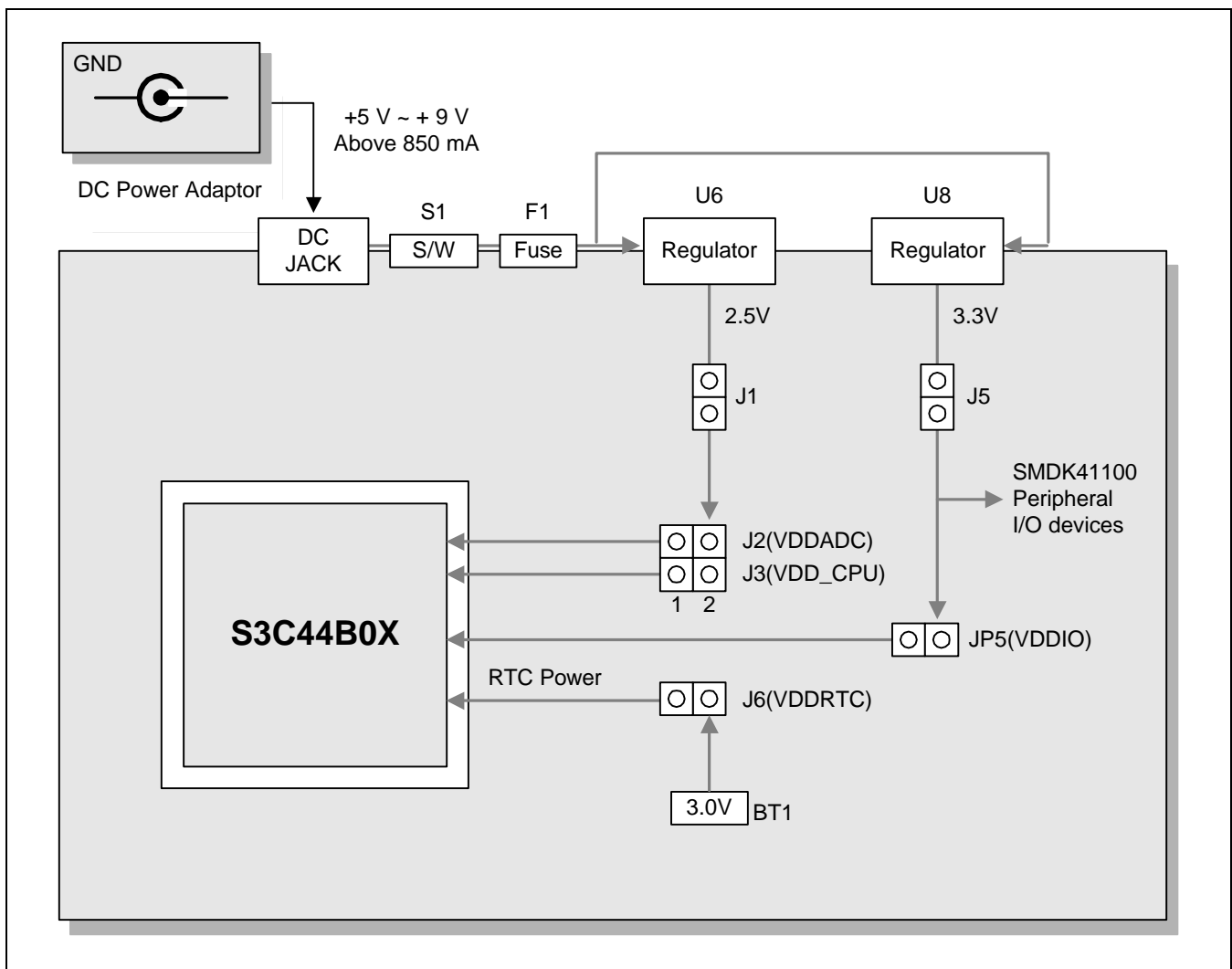


Figure 1-2. SMDK41100 Power Plane

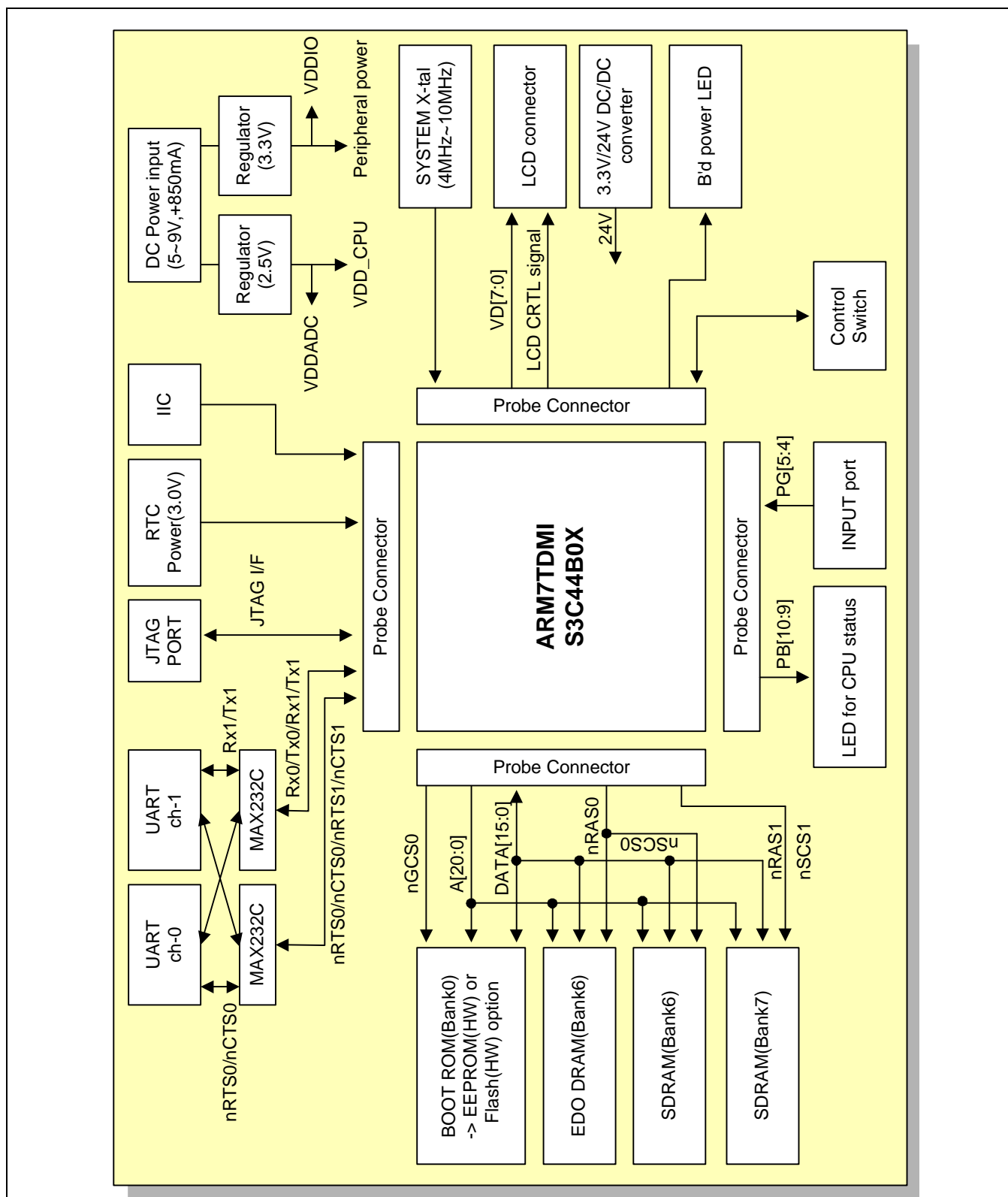


Figure 1-3. Detailed SMDK41100 Board Diagram

## CLOCK SOURCE AND DISTRIBUTION

The Following clock source is supported at SMDK41100 target board.

### System Clock (MCLK)

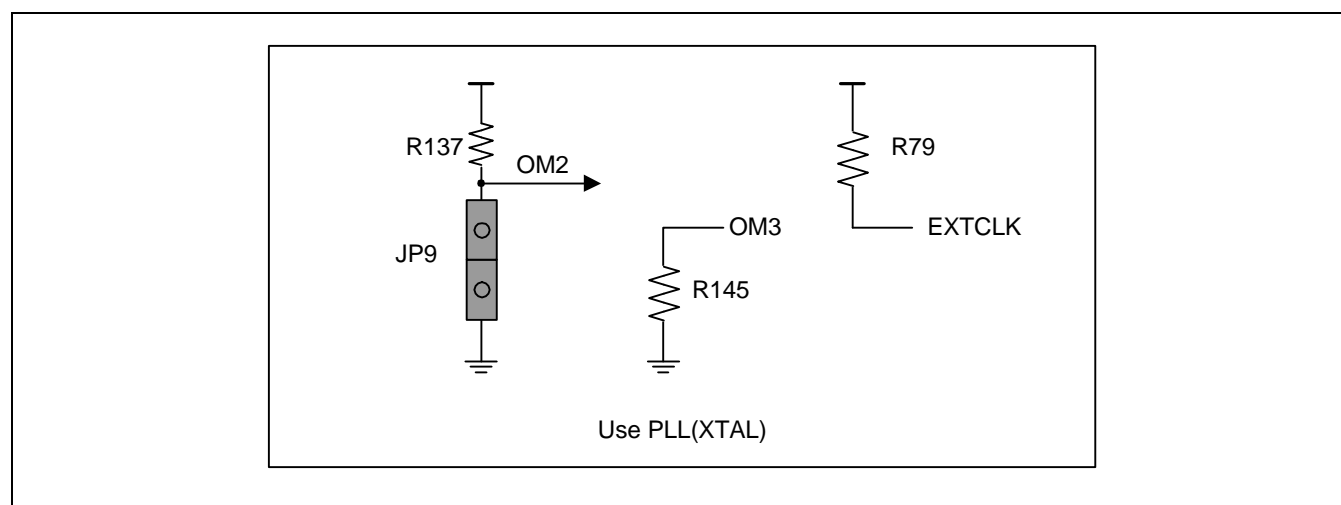
Table 1-1 shows the relationship between the mode control pin(OM2 & 3) combination and the clock operating mode. The OM[3:2] status is latched at rising edge of nRESET. (See, Figure 1-4)

**Table 1-1. System clock configuration**

CLOCK MODE(JUMPER)	Pin Value OM[3:2]	Descriptions	PIN Status
JP9			
Short	00	XTAL0, PLL-on	EXTCLK(#67)=Vdd

#### NOTES:

1. SMDK41100 provides only XTAL. If you'd like to use oscillator as a external clock, you have to modify our board.
2. If the EXTCLK is used, XTAL0 has to be in H level. If XTAL0, EXTAL0 is used, EXTCLK also has to be in H level.
3. Although the PLL starts to operate just after reset, the PLL output is not used as Fout. Until S/W writes valid settings to the PLLCON register, the crystal clock will be used as Fout directly. After S/W writes to the PLLCON, the PLL output is used as Fout.



**Figure 1-4. SMDK41100 Board System Clock (MCLK) Configurations**

## RESET LOGIC

The nRESET(System Reset Signal) must be held to low level at least 5 CLK to recognize the reset signal and it takes 128 CLK between the nRESET and internal nRESET. nRESET and nTRST(JTAG Reset signal) should be merged by jumper option. But, if you want to use circuit emulator (ex, Embedded ICE) for debug without BOOT ROM, you should have the nTRST be pull-up. If not, whenever the ADW (ARM Debug Window) were invoked SW interrupt will be occurred.

## SMDK41100 SYSTEM CONFIGURATIONS

S3C44B0X supports Big-/Little-endian mode and Byte/Half-word/Word access the data bus. But SMDK41100 supports only Half-word data access in BOOT ROM.

### BOOT ROM

You can select EEPROM or Flash memory using jumper option(JP13) for BOOT ROM.

The data bus width of BOOT ROM bank is fixed by Half-word data in SMDK41100.

In EEPROM, you have to open JP13 but in Flash memory you have to remove EEPROM and to close JP13.

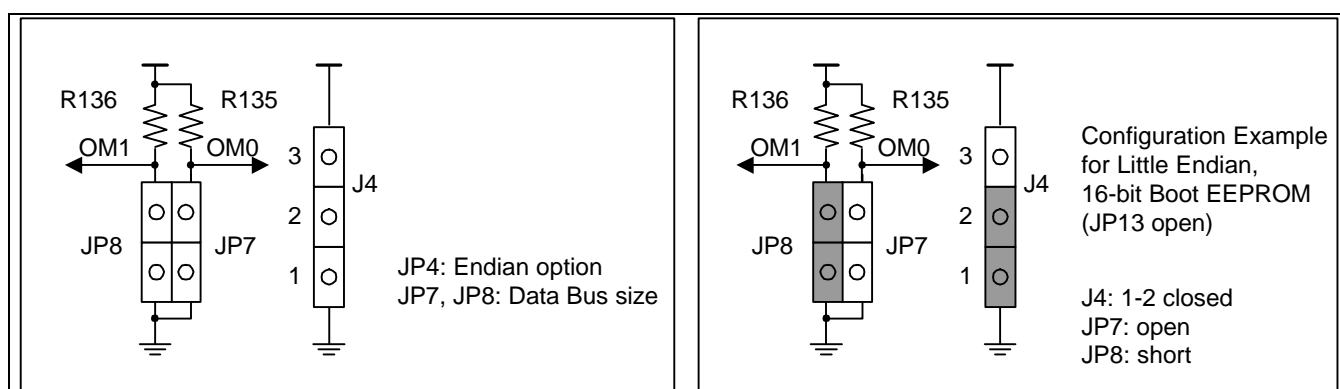


Figure 1-5 SMDK41100 Board Configurations

Table 1-3. ROM Bank0 Data Bus Width

PIN FUNCTIONS	JP8	JP7	PIN value, OM[1:0]	DESCRIPTIONS
ROM Bank0 Data Bus Width Configuration	Short	Open	"01"	16_bit mode

Table 1-4. Endian Mode Configuration

PIN FUNCTIONS	J4(1-2)	J4(2-3)	PIN value	DESCRIPTIONS
Endian Mode Selection	closed	open	"0"	Little endian mode
	open	closed	"1"	Big endian mode.

**DRAM/SDRAM CONFIGURATIONS**

S3C44B0X supports FP DRAM, EDO DRAM and Synchronous DRAM. S3C44B0X's nRAS0 signal simultaneously can be used EDO DRAM and Sync. DRAM on SMDK41100. SDRAM or EDO DRAM can be selected alternatively using SYSCFG register (rBANKCON).

R76/R81 are resistors for selecting memory type of BANK6 and R100/R111 are resistors for selecting the data bus width of BANK6 or BANK7.

So, SMDK41100's DRAM/SDRAM configuration can be summarized in three cases(16bit EDO DRAM mode/16bit SDRAM mode/ 32bit SDRAM mode)

**Table 1-5. DRAM/SDRAM Configuration of BANK6 and BANK7**

Number of case	R76	R81	R100	R111	Memory Mode	
					Descriptions	Block Diagram
1	Closed	Open	Open	Open	16bit EDO DRAM mode	
2	Open	Closed	Open	Closed	16bit SDRAM mode	
3	Open	Closed	Closed	Open	32bit SDRAM mode	

**NOTES:**

1. If 32bit data bus width is used in SMDK41100, optional resistors(Data option:R62-R71, R74, R75, R78, R80, R82-R85, R87, R89, R91, R92, R94, R95, R99, R101, R103, R104, R106, R109, R110, R113-R125, R175, R176, Address option:r1-r40) have to be set properly.
2. When the user set the data option, please consider other options(IIS, LCD, UART0 and UART1)



## General I/O PORTS

S3C44B0X's general I/O ports are used for DEMO B'd key interrupt input and LED status display. The function of control switch and the status of LED can be defined by user software.

**Table 1-6. General I/O Configurations on SMDK41100**

General I/O port number	I/O type	Descriptions
PB[10:9]	Output	LED display
PG[5:4]	Input	Key input pad (External interrupt input pins).

## LCD INTERFACE

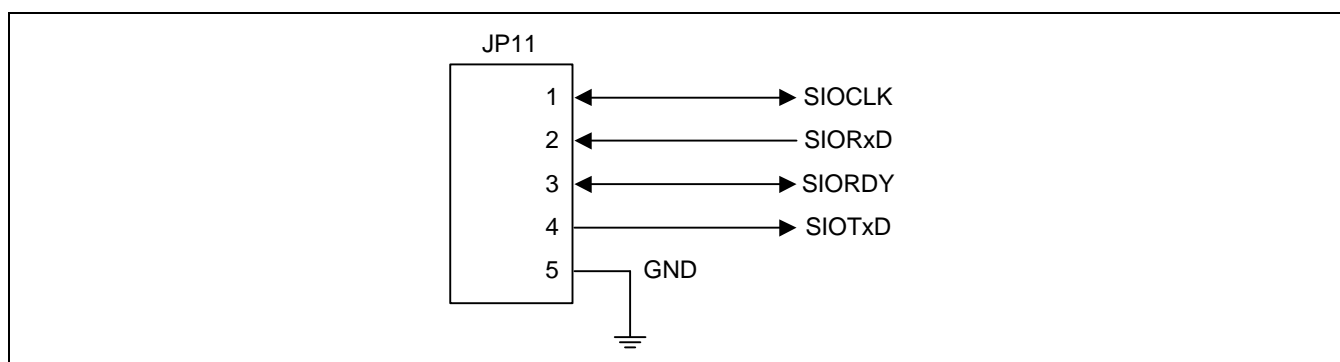
S3C44B0X has LCD controller. The LCD controller has some signal, such as VD[7:0], VM, VFRAME, VCLK. SMDK41100 provides the LCD control signals as follows;

**Table 1-7. LCD Interface on SMDK41100**

# of hole	Descriptions	# of hole	Descriptions	# of hole	Descriptions	# of hole	Descriptions
1	VSS	6	NC	11	VD0	16	VD5
2	VDD	7	VM	12	VD1	17	VD6
3	VFRAME	8	NC	13	VD2	18	VD7
4	VLINE	9	NC	14	VD3	19	NC
5	VCLK	10	NC	15	VD4	20	NC

## SIO INTERFACE

S3C44B0X has SIO(Synchronous I/O). SMDK41100 provides the SIO(JP11) signals as follows;



**Figure 1-6 SIO Interface on SMDK41100**

## IIS Interface

S3C44B0X has IIS(Inter IC Sound).  
SMDK41100 provides the IIS(JP10) signals as follows;

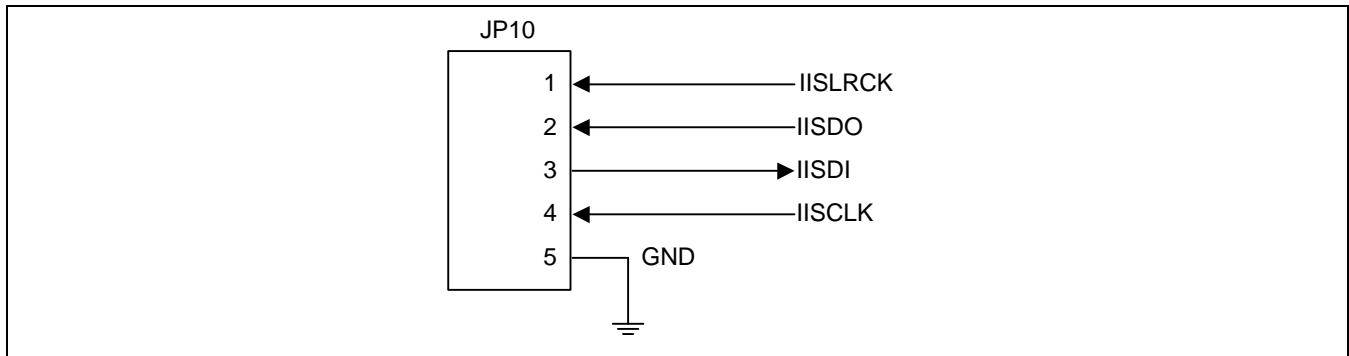


Figure 1-7 IIS Interface on SMDK41100

## ExINT INTERFACE

There are four ExINT(External Interrupt) interface in SMDK41100. SMDK41100 provides the ExINT(JP14) signals as follows;

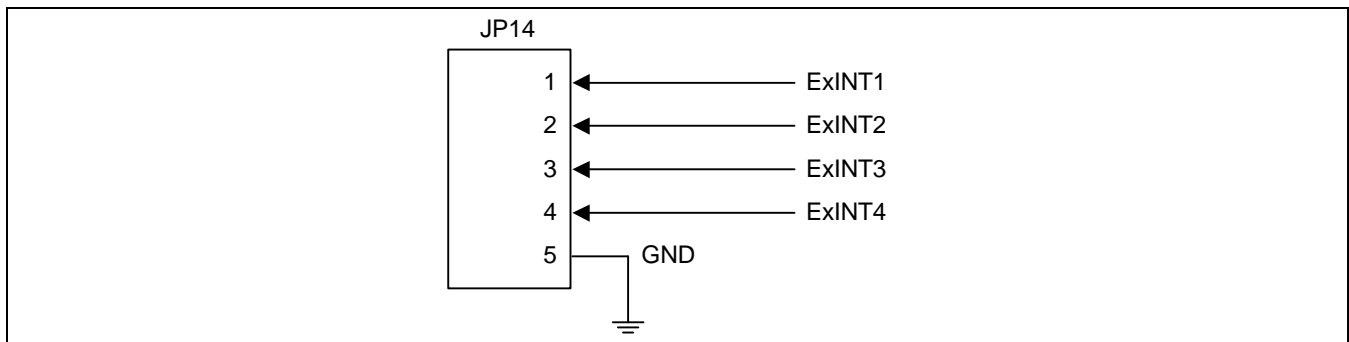


Figure 1-8 ExINT Interface on SMDK41100

## A/D CONVERTER INTERFACE

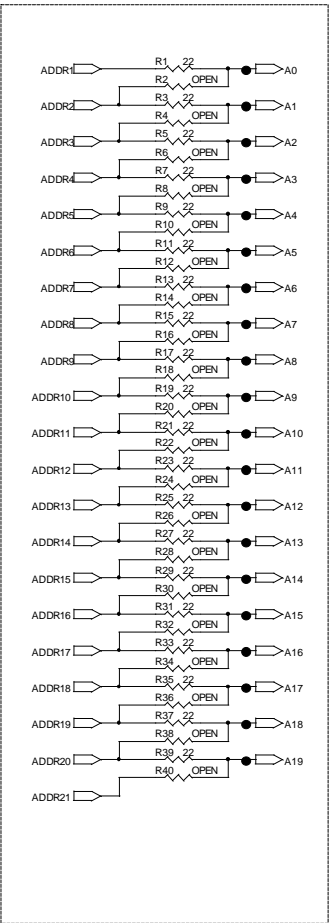
S3C44B0X has ADC(Analog to Digital Converter). The ADC has 8-ch analog input signals.  
SMDK41100 provides the ADC(JP15) signals as follows;

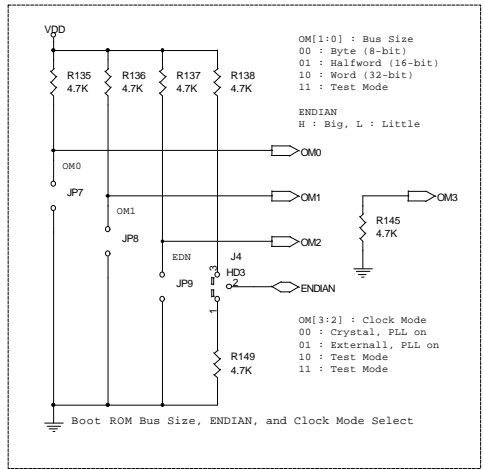
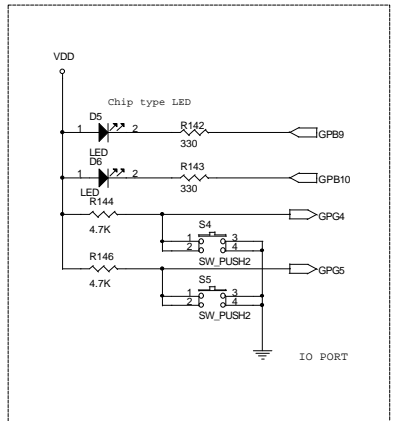
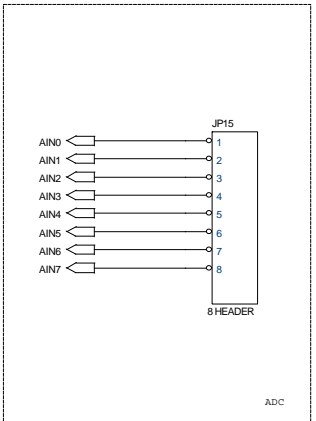
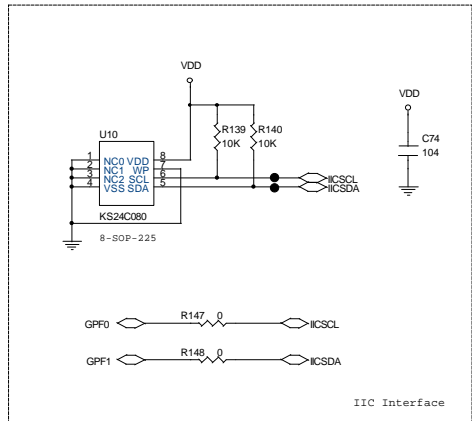
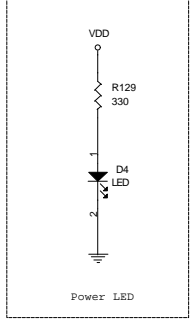
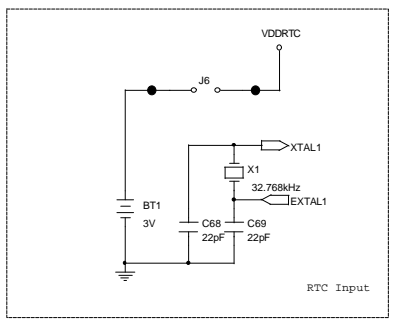
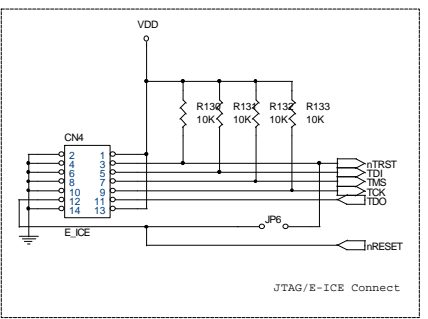
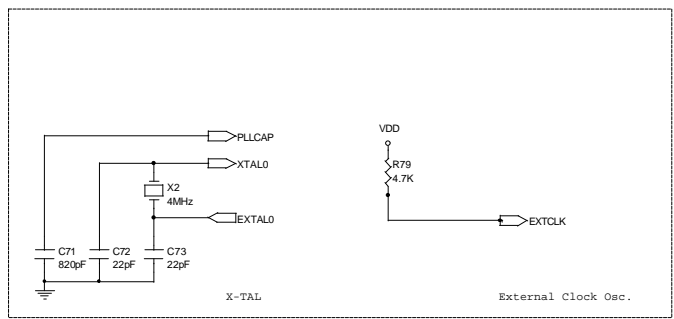
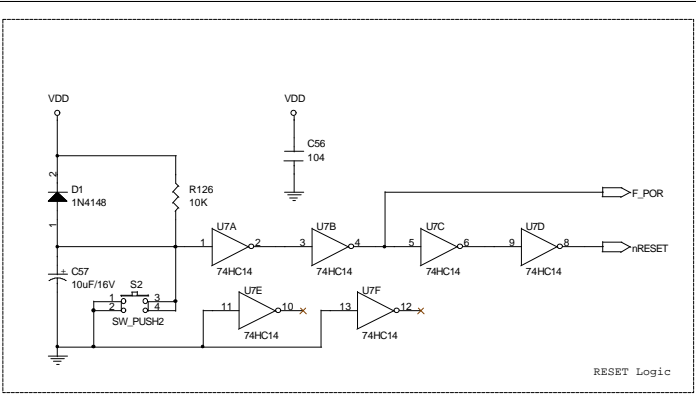
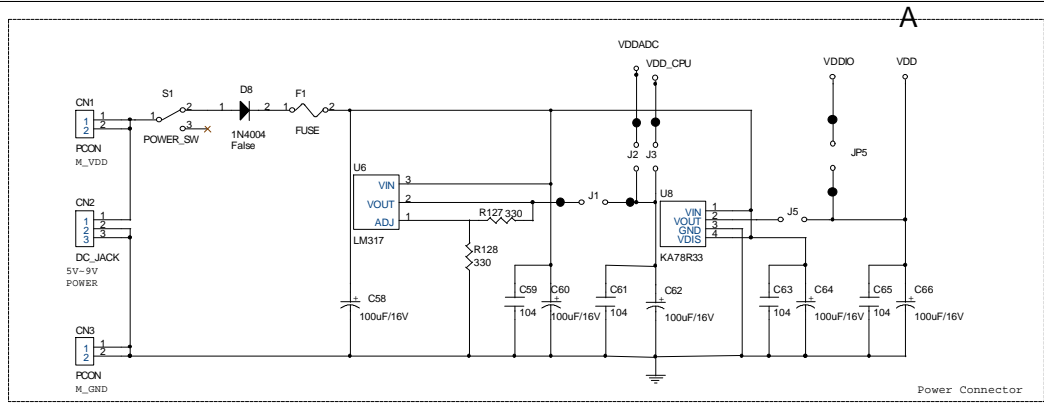
Table 1-8. ADC Interface on SMDK41100

# of hole	Descriptions	# of hole	Descriptions	# of hole	Descriptions	# of hole	Descriptions
1	AIN0	2	AIN1	3	AIN2	4	AIN3
5	AIN4	6	AIN5	7	AIN6	8	AIN7

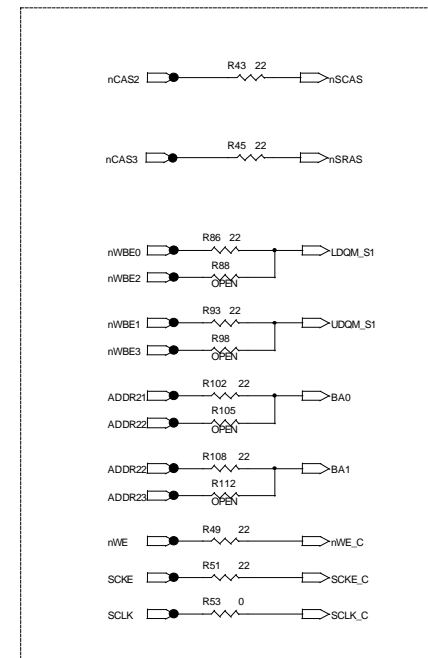
**SMDK41100 REV. 1.0. BOARD SCHEMATICS**

1. General MCU(S3C44B0X) device interface
2. Etc.(Power/JTAG/Clock . . . )
3. SDRAM/DRAM
4. EEPROM/Flash
5. UART
6. LCD



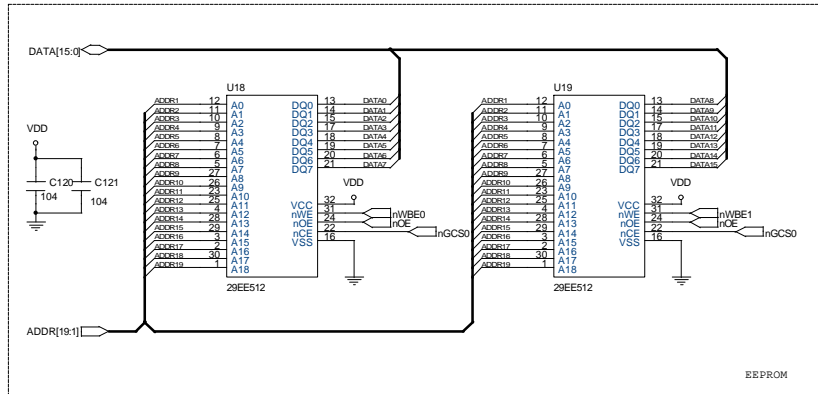


SAMSUNG ELECTRONICS CO.,LTD		
Title	S3C44BOX DEMO BOARD	
Size	Document Number	Rev
C	Etc Circuit	0.0
Date: Thursday, August 24, 2000, Sheet: 3 of 6		

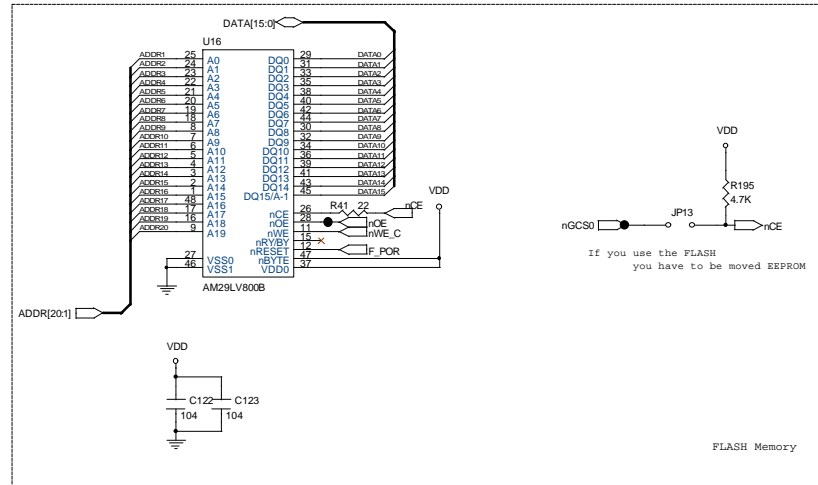


SAMSUNG ELECTRONICS CO.,LTD			
Title S3C4B0X DEMO BOARD			
Size C	Document Number DRAMSDRAM	Rev (	
Date:	Thursday, August 24, 2000	Sheet	2 of 6

A



A



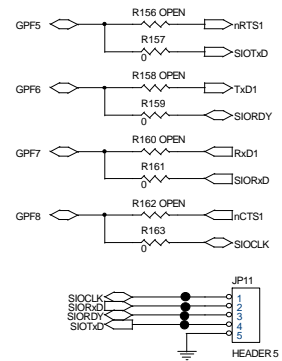
A

Title		
S3C44BOX DEMO BOARD		
Size		
C	Document Number	Rev
	EEPFLASH	0.0
Date: Thursday, August 24, 2000 Sheet 6 of 6		

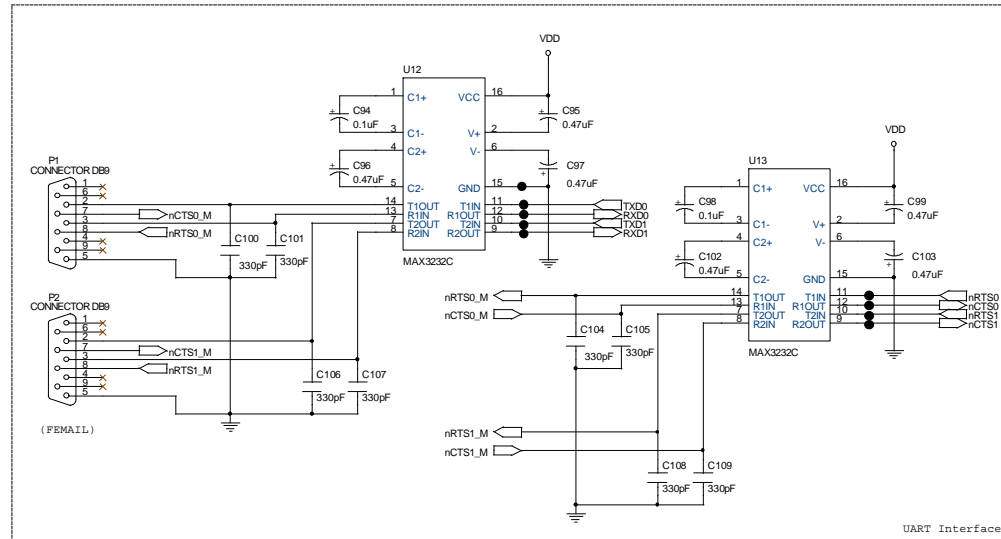
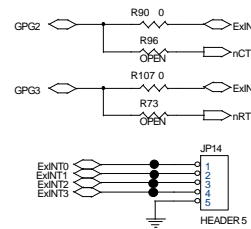
A

A

In 32-bit data mode be supported below,  
 -UART2 channel with RTS/CTS  
 -or UART0's Rx/D/TxD with RTS/CTS & SIO



In 32-bit mode , UART0 Handshake

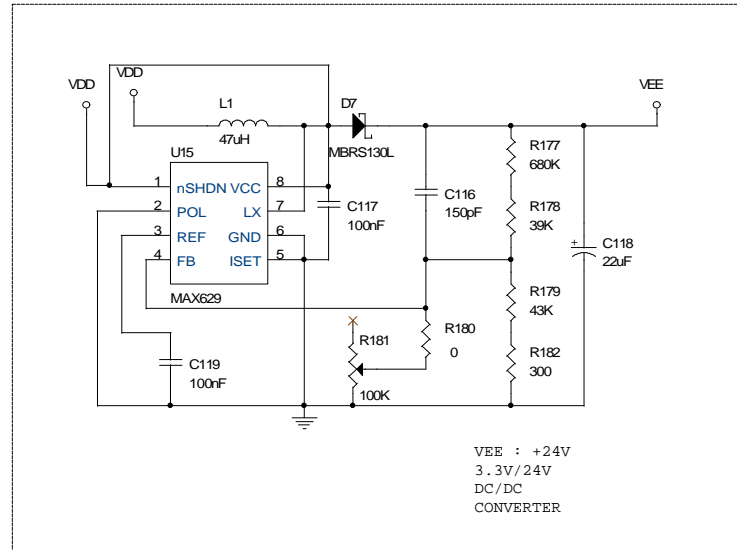
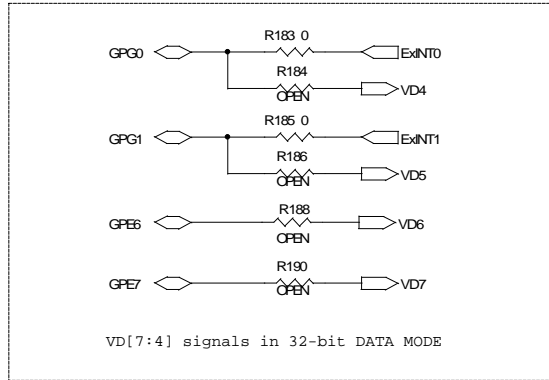
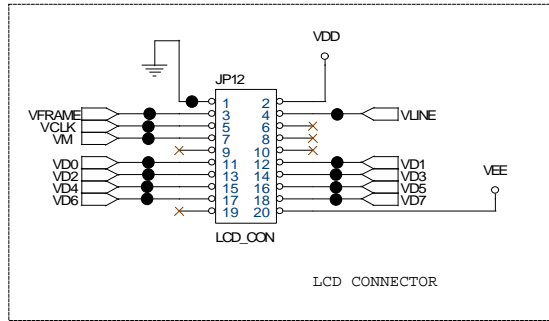


Title		
S3C44BOX DEMO BOARD		
Size	Document Number	Rev
C	UART/SIO/S	0.0
Date: Thursday, August 24, 2000 Sheet 4 of 6		

A



A



Title		
S3C44B0X DEMO BOARD		
Size	Document Number	Rev
B	LCD	0.0
Date:	Thursday, August 24, 2000 Sheet 5 of 6	

A

# 2

## TOOLKIT & DEBUGGING

### SETUP SMDK41100 ENVIRONMENTS

The evaluation environments for SMDK41100 are shown in Figure 2-1. Serial port(UART0) on SMDK41100 have to be connected to COM port of Host PC . This can be used as console for monitoring and debugging SMDK41100.

If you have the emulator like as EmbeddedICE, you can use JTAG port on SMDK41100 as the interface for it.

DC power adapter can be used as input power of the SMDK41100 board. This input power regulated to 2.5V and 3.3V for CPU and peripheral devices on SMDK41100 board.

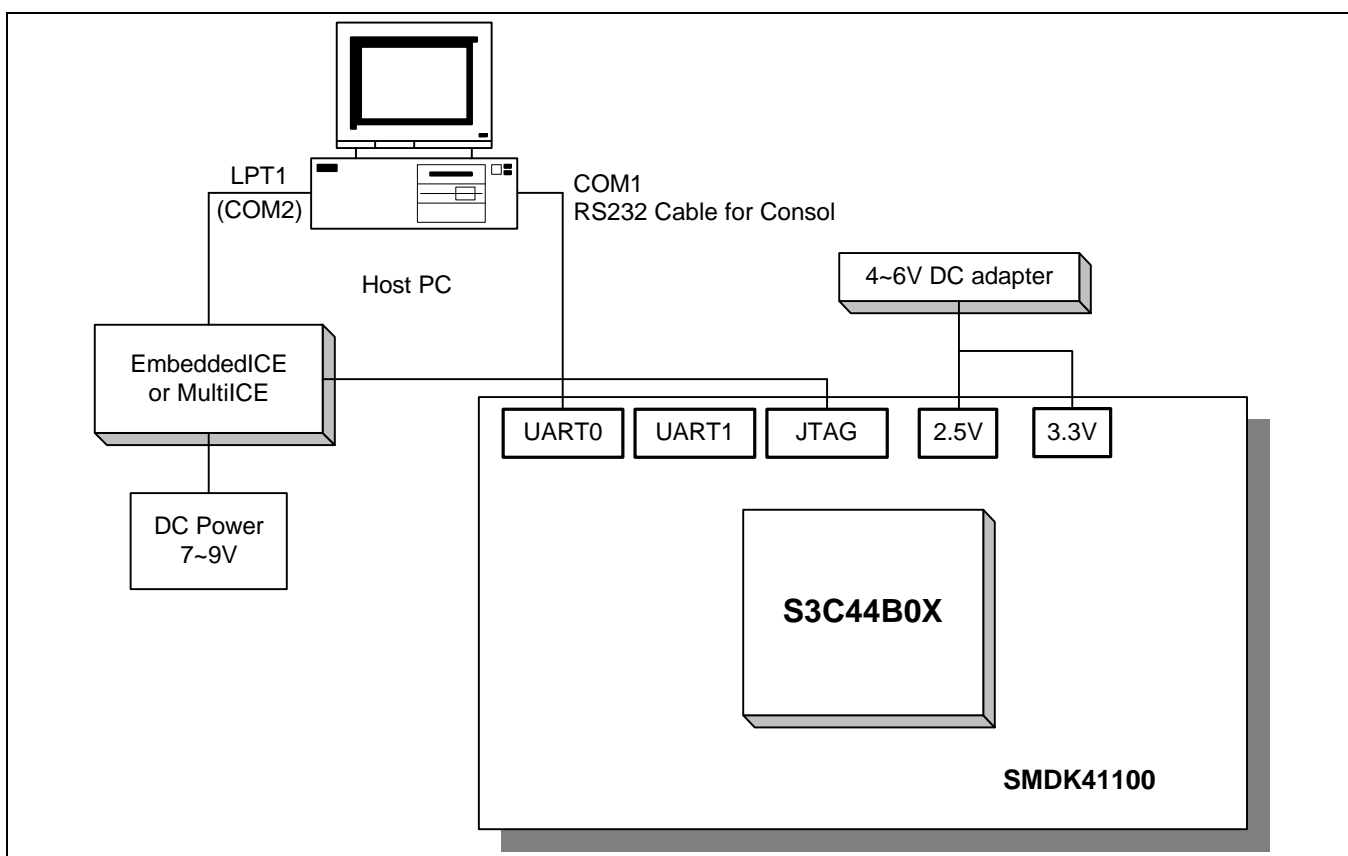
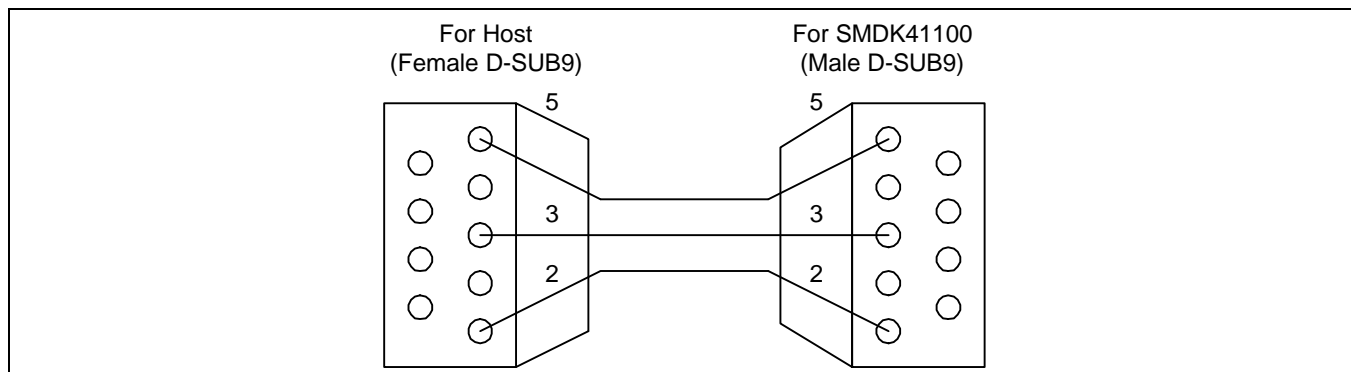


Figure 2-1. Setup Environments for SMDK41100 Board

**RS232C CABLE CONNECTION**

The serial cable is made as in Figure 2-2. The other pins of the cable are not used. Please check the cable's connections again!

The UART0 and PC COM1 port has to be connected through this cable connection.

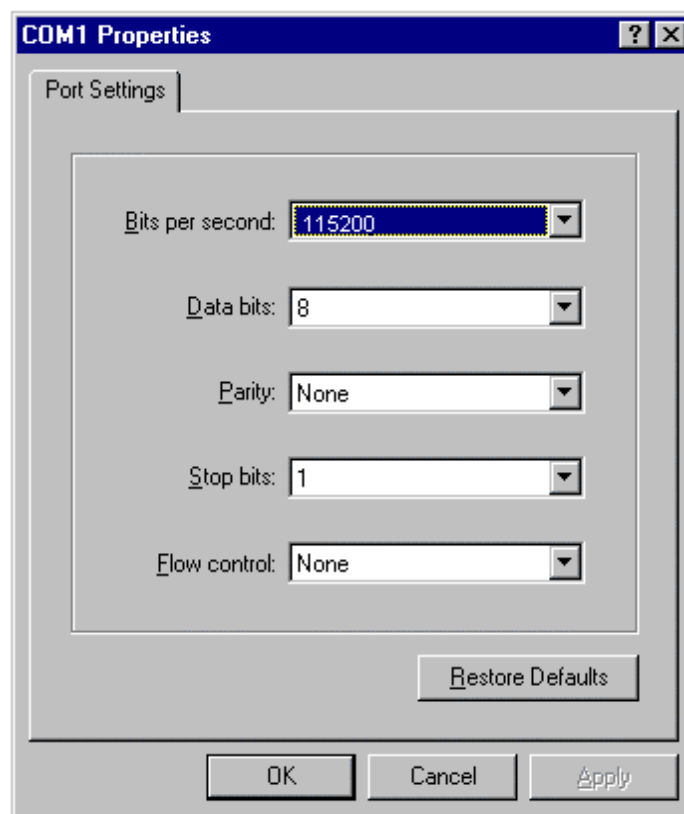


**Figure 2-2. Serial Cable Connections for SMDK41100 Board**

**CONFIGURING THE HYPER TERMINAL (ON WINDOWS 95 OR NT)**

To configure the Hyper Terminal, which is windows utility program for serial communications be given on windows 95 or NT, please following steps:

1. Running the Hyper Terminal utility program.  
Window 95 or Windows NT start tool bar → Program → Accessories → Hyper Terminal Group → Double click Hyperterm.exe → Enter connection name → Select icon → Click OK.
2. Select COM port to communicate with SMDK41100 target board.  
Choose COM1 (or COM2) as the serial communication port → Click OK
3. Set the serial port properties. (See the Figure 2-3.)
  - Bits per second: 115200 bps
  - Data bits: 8 bits
  - Stop bits: 1
  - Flow control: None



**Figure 2-3. Setting Port Properties**

4. Select Properties menu  
File → Properties.
5. Choose Setting Page (Figure 2-4)
6. Click ASCII Setup button
7. Check ASCII Setup menus and set (Figure 2-5)
8. Re-connect Hyper-Terminal to run at new properties
  - Disconnect: Call → Disconnect
  - Connect: Call → Call
9. Power-On Reset or Push the reset button on SMDK41100 board
  - Now, The diagnostic menu is showed on the Hyper-Terminal (Refer to Figure 2-6).

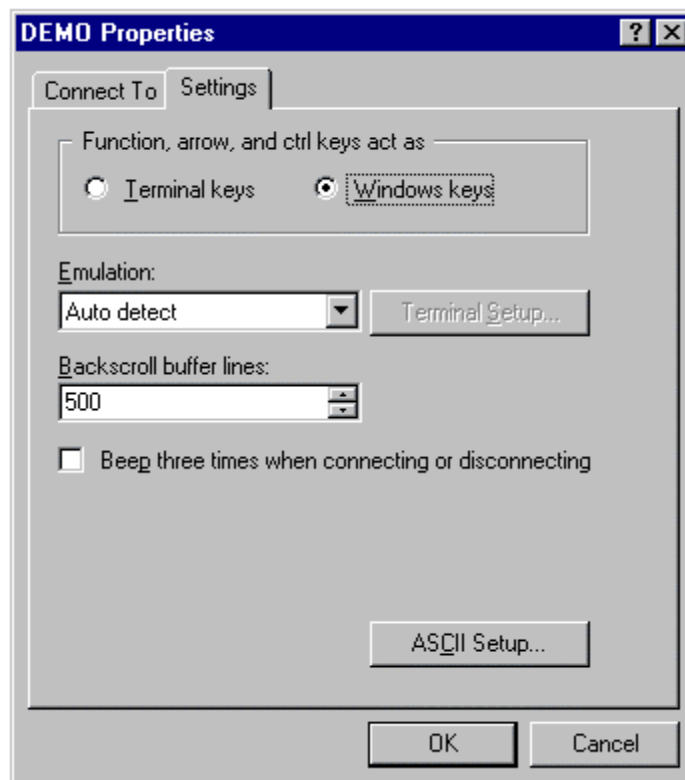


Figure 2-4. Choose Setting Page

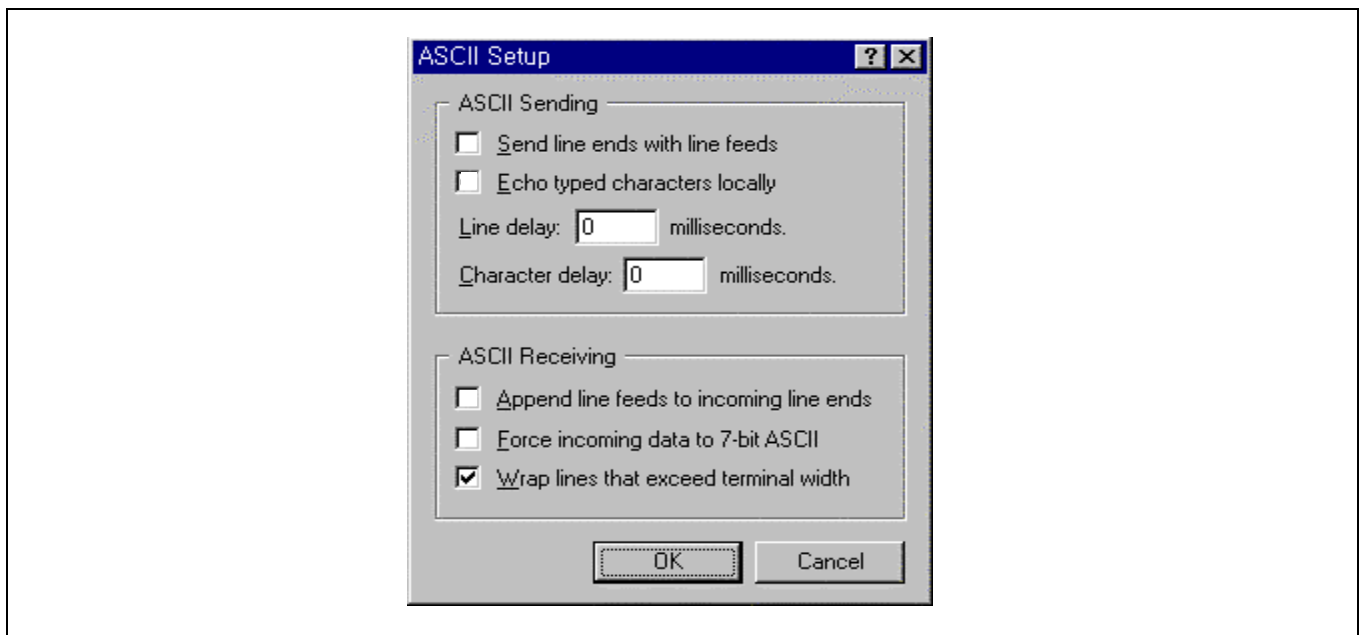


Figure 2-5. ASCII Setup Menu

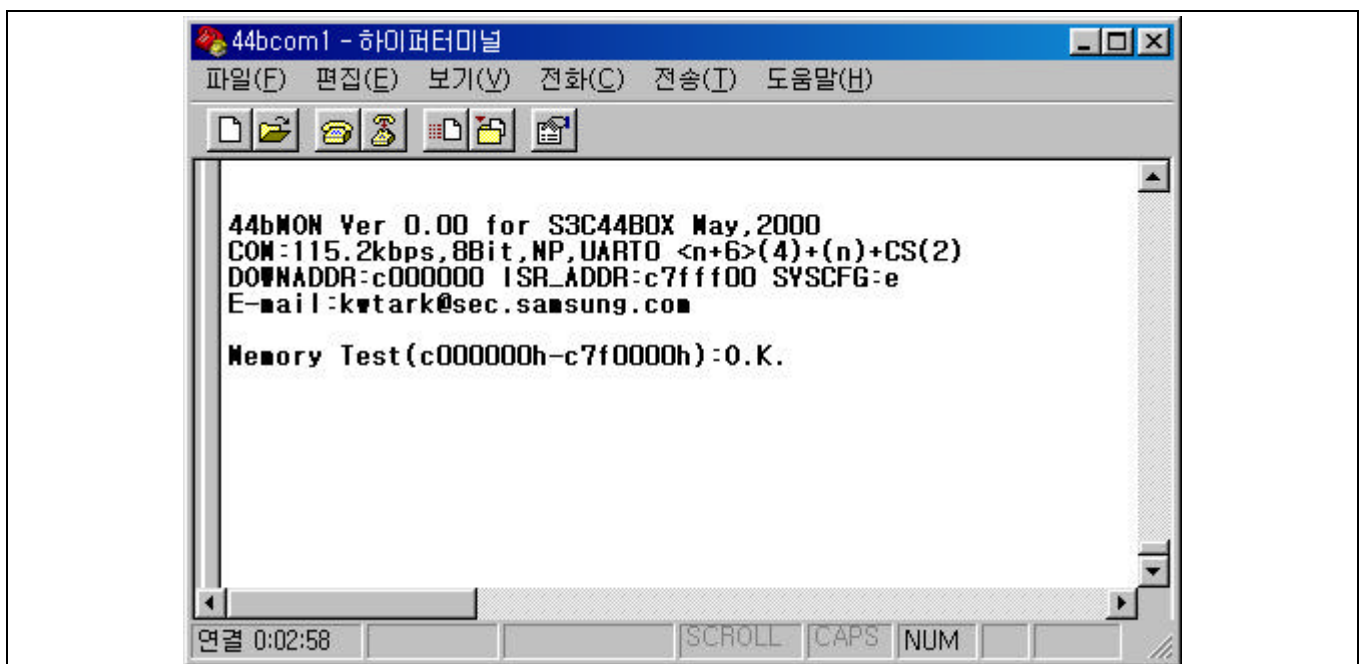


Figure 2-6. SMDK41100 Booting Window After Reset

## INSTALL ARM TOOLKIT

First of all, install ARM toolkit 2.51.

You may install ARM toolkit 2.11a but makefile has to be changed a little because the toolkit 2.11a doesn't support fromelf.exe utility. However, We recommend ARM toolkit 2.51 because it's more enhanced toolkit that removes some bugs in toolkit 2.11a.

Also, the DOS environment variable has to be changed as follows after installation of the toolkit.

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

## HOW TO BUILD EXECUTABLE IMAGE FILE

Execution image file can be built using by ARM Project manager or makefile. First you have to build ELF format image. An ELF format image can be used for ARM debugger directly. The binary file(.bin file) can be extracted from .ELF format image. If you want to use ARM debugger.

First of all, you have to download S3C44B0X evaluation source code and any other utilities from our web site ([www.samsungsemi.com](http://www.samsungsemi.com)). It will be helpful to more easily understand the development environments of S3C44B0X. The distributed evaluation source codes are consist of following directories.

Directory	Description
44BMON	Down loader program through UART0. This will be an example for program running on ROM.
44BTEST	Evaluation source code that test the S3C44B0X function blocks.
ARMutil	Utilities for compile and download above program.

## BUILD 44BMON.BIN

If you made your own board instead of SMDK41100, 44BMON.BIN file has to be made first. To use interrupt service routine in sample codes, the 44BMON.BIN image file has to operate on ROM.

To build 44BMON.BIN, run makefile file in 44BMON/SRC directory using following command. If the 44BMON.BIN operates, you will see the figure 2-6.

```
cd 44BMON/SRC
armmake -a
```

If 44BMON.BIN operates, you can download an application program (ex. 44BTEST.bin) through serial (COM1). If SMDK41100 board doesn't operate, check that the JP6 jumper is short.

**BUILD 44BTEST.ELF**

To build the sample source code, 44BTEST, run makefile file in 44BTEST/SRC directory using following command. cd 44BTEST/SRC  
armmake -a

After build, 44BTEST.ELF and 44BTEST.BIN image files will be seen in 44BTEST/BIN directory. The 44BTEST.ELF file is used for ARM debugger.

The 44BTEST.BIN file is used for serial downloading with 44BMON.



**EXECUTE 44BTEST WITHOUT MDS (EMBEDDED ICE, MULTI ICE OR JEENI )**

First, 44BMON has to operate on ROM. 44BMON will be ready to receive 44BTEST.BIN. 44BMON will be launch 44BTEST.BIN after receiving 44BTEST.BIN.

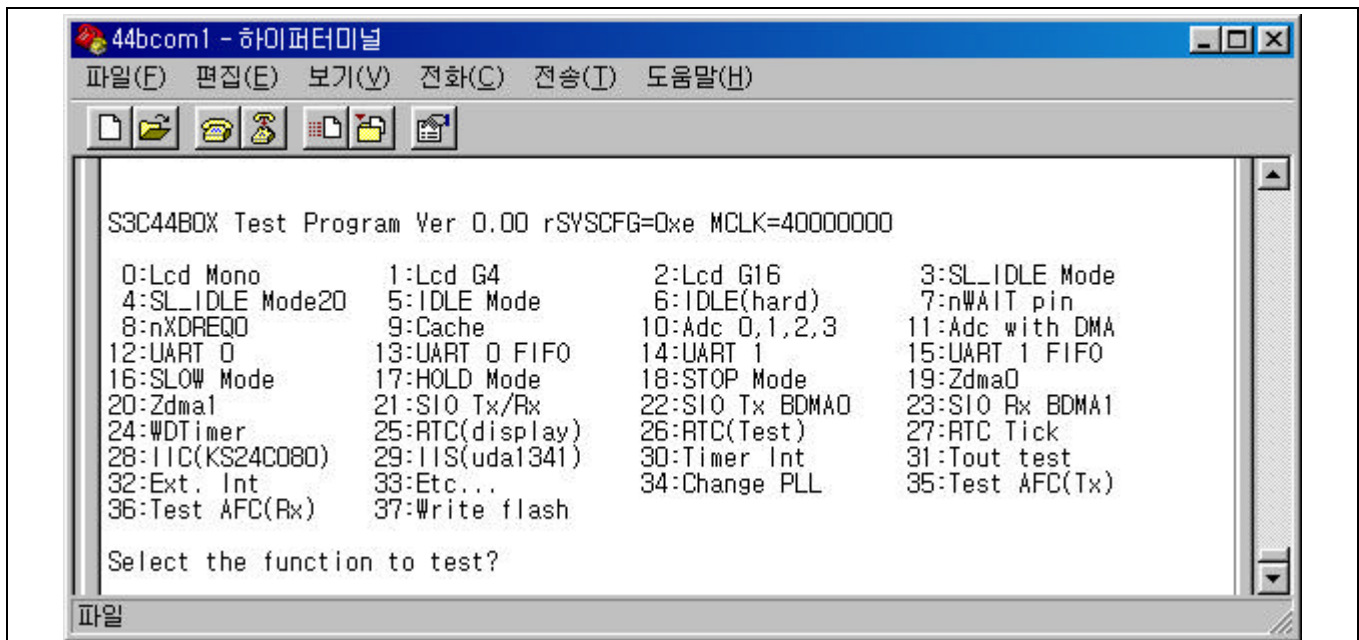
To download 44BTEST.BIN, Type following commands. Wkcom2.exe will transmit file size and checksum together with .BIN file.

```
wkcom2 44BTEST.bin /1 /d:1
```

```
/1: com1 port is used.
```

```
/d:<n> : baud rate=115200 / <n>
```

After download, you can test S3C44B0X functional units and see figure 2-7



**Figure 2-7. 44BTEST Execution After Download Using UART0**

## HOW TO USE ARM DEBUGGER WITH MDS (EMBEDDED ICE, MULTI ICE OR JEENI )

If you have built 44BTEST program without any error, you can find 44BTEST.ELF in 44BTEST/BIN directory. The generated image file will be downloaded to SDRAM/DRAM memory on the SMDK41100 target board by ARM debugger through MDS, such as EmbeddedICE, MultiICE, JEENI and so on. Next, you can start to debug the downloaded image using ADW( ARM Debugger for Window ).

## PREPARING EMBEDDED ICE

1. EmbeddedICE will be connected through JTAG port on the board. Connect all cables properly by each MDS manual. Open JP6 on SMDK41100 board to use JTAG port.

Reset the SMDK41100 board and the EmbeddedICE interface unit.

Reset SMDK41100 board (Press reset switch) → Reset EmbeddedICE unit (Press red switch)

2. Start ARM Debugger using ARM debugger icon

Also, you can start it at DOS command window as you type `adw 44BTEST.elf` in there.

3. When ARM Debugger is started, it will load the image code to Armulator. (The Armulator is software emulator for ARM7T CPU )

If you ever used ARM Debugger as a remote debugger, Remote Debugger warning message dialog box will be displayed. If the remote debugger option is correct, then select Yes, otherwise, select No.

## CONFIGURING ARM DEBUGGER FOR EMBEDDED ICE

In order to access a remote target, you should configure ARM Debugger for Windows (ADW) rather than ARMulator. The EmbeddedICE interface unit must also be configured for the ARM core in the target system

The ARM7TDMI core is contained in the S3C44B0X on the SMDK41100 board. ARM7TDMI macro cell includes the ARM7 core with Thumb, debug extensions, and the EmbeddedICE macro cell.

To configure ARM Debugger using the EmbeddedICE interface, follow the below steps:

1. Select Configure Debugger from the Options menu.  
Options -> Configure Debugger
2. Debugger Configuration dialog box is displayed (Figure 2-8). When you select Target page, there are two Target Environment available as follows.
  - ARMulator: lets you execute the ARM program without any physical ARM hardware by simulating ARM instructions in software.
  - Remote\_A: connects the ARM debugger directly to the target board or to an EmbeddedICE unit attached to the target.
3. Select Remote\_A from target environments, and click Configure.
4. Configure Angel Remote Configuration dialog box(Figure 2-9).
  - Remote Connection: select your host and EmbeddedICE communication port configuration.
  - Select Ports and Serial Line Speed.

5. Select Debugger page from Debugger Configuration dialog box (Figure 2-10 ) and configure it.
  - Endian: little (If the big endian is used, Endian: big has to be selected.)
6. Select Memory Maps page from Debugger Configuration dialog box and configure it.
  - No Map File
7. 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 target. This means that the executable image file is downloaded to the SDRAM/DRAM code area.

This configuration is done once at first. In the next time, this configuration is not needed because the setting was saved.

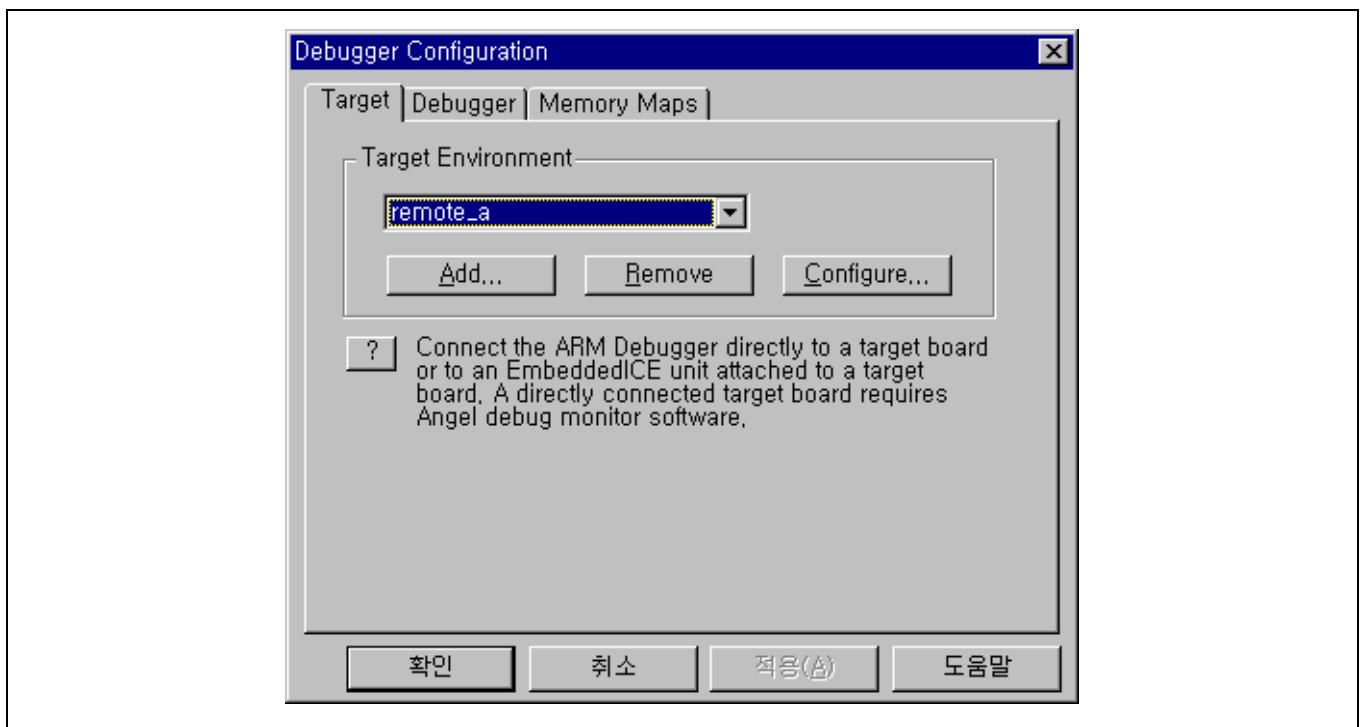


Figure 2-8. Debugger Configuration: Target Page

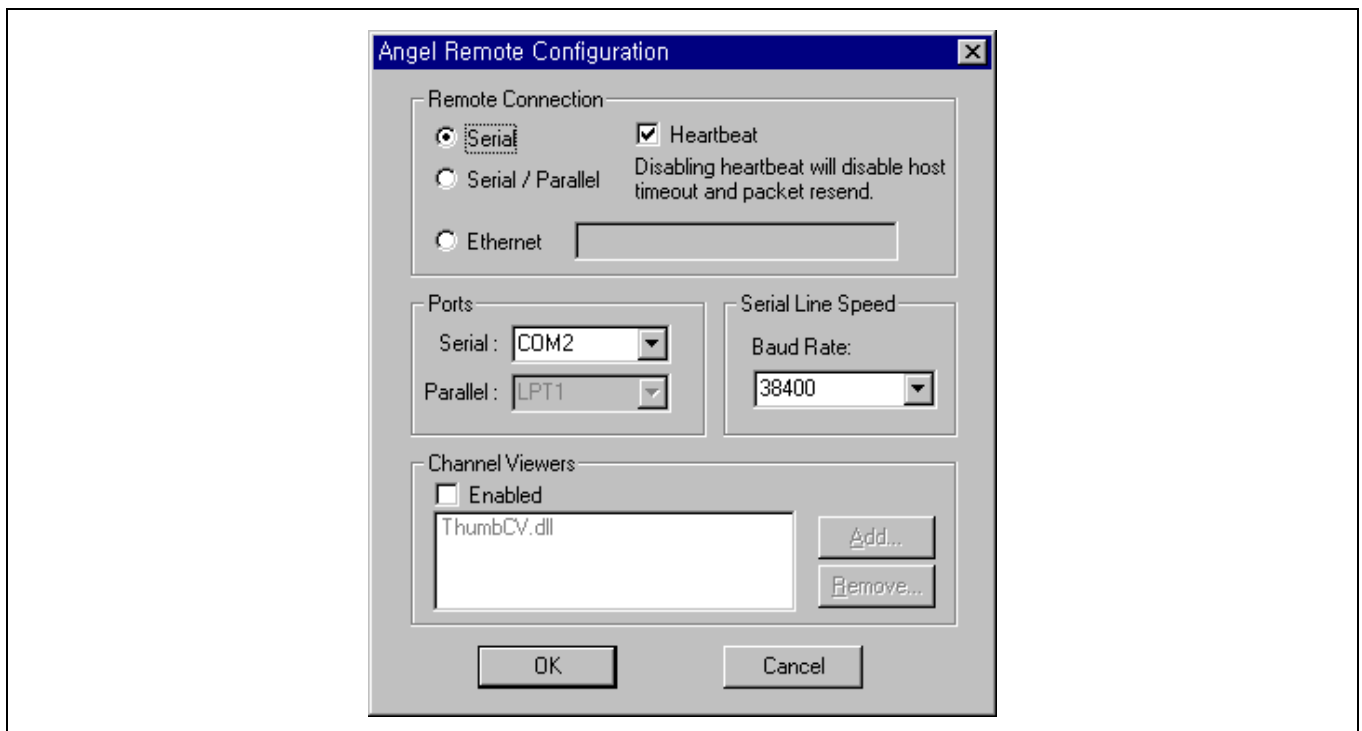


Figure 2-9. Angel Remote Configuration

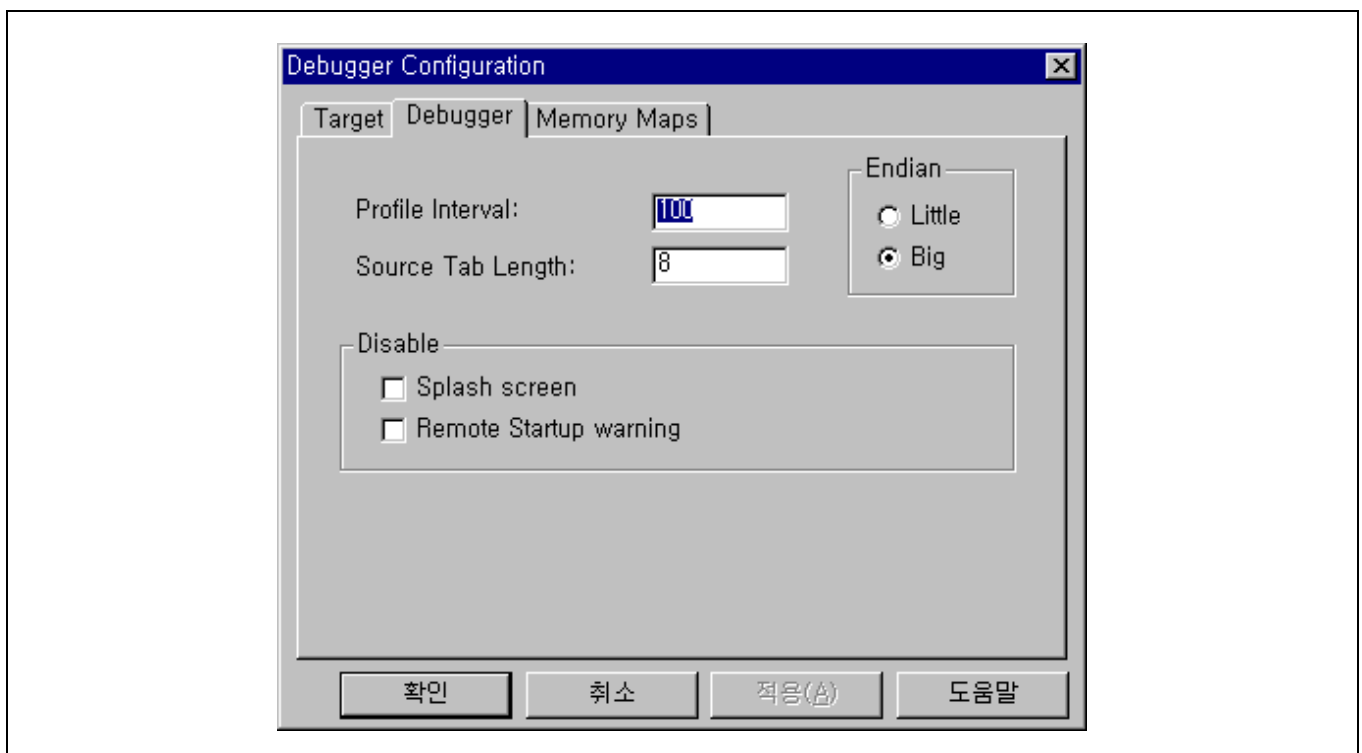


Figure 2-10. Debugger Configuration: Debugger Page

## EXECUTING 44BTEST.ELF USING EMBEDDED ICE

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

```
let $vector_catch=0x00
let $semihosting_enabled=0x00
let psr=%if_SVC32
```

Or, you can initialize these variables as follows:

First, create a text file named "armsd.ini", which includes the commands described above. Then, enter the following command in the Command window (Figure 2-11):

```
obey c:\armutil\armsd.ini
```

For more information about these steps, please refer to "ARM Software Development Toolkit User's Guide".

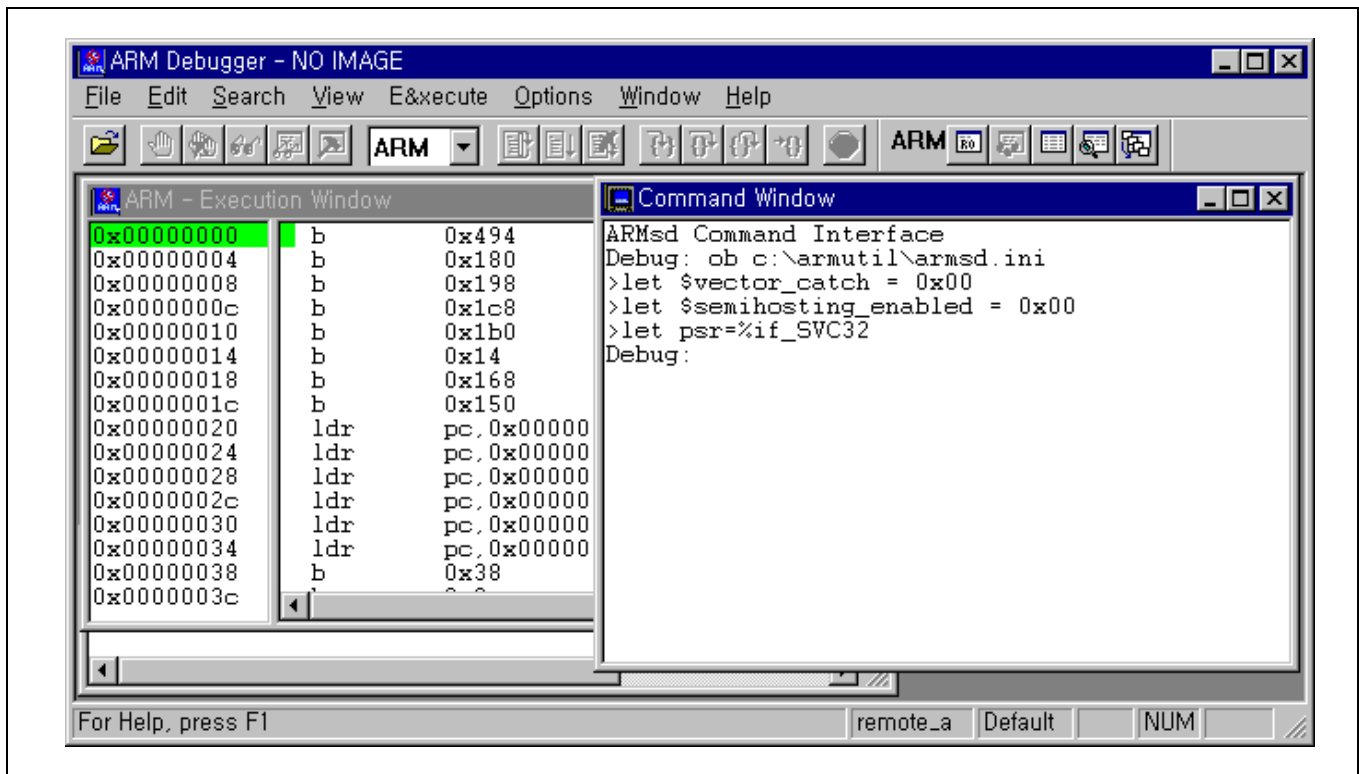


Figure 2-11. ARM Debugger Window(ADW): Command Window

3. Set breakpoint at Main in 44BTEST.c as follows  
break Main
4. Execute the program by clicking Execute menu→Go. The program execution will stop at Main( ).
5. Now, The downloaded image file will be run on SDRAM/DRAM area. 44BTEST program running status can be monitored on the current Hyper-Terminal.

## DEBUGGING DOWNLOAD IMAGE IN ADW

### Stepping Through The 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 A Break Point

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 the 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 A Watch Point

A watch point halts a program when the value of a specified register or a variable changes is set to a specific number.

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 watchpoint. 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 THE CODE INTERLEAVED WITH THE 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, please refer to ARM Software Development Toolkit User's Guide.

## WRITE IMAGE TO FLASH MEMORY(AM29LV800BB)

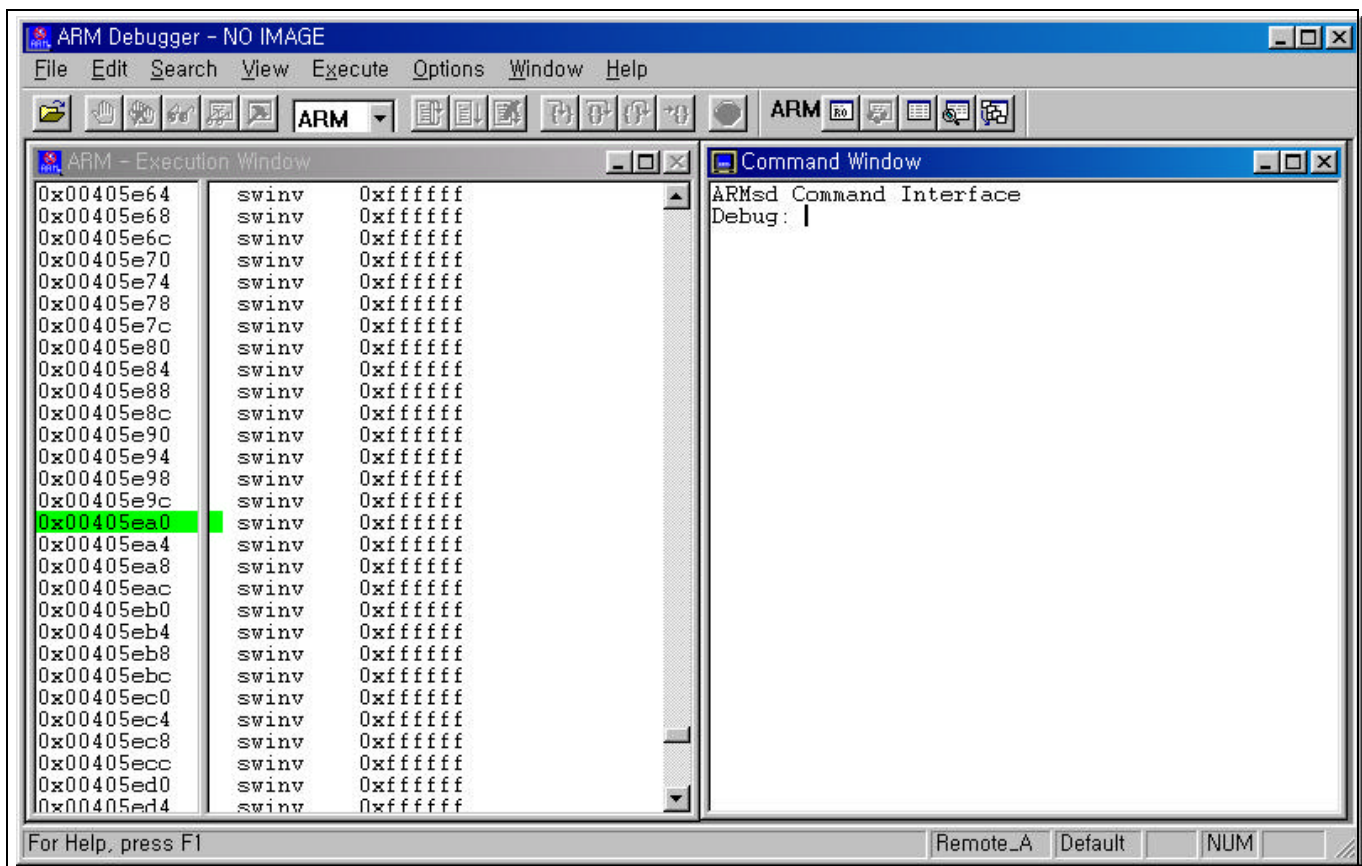
### OPEN THE JP3 IN SMDK41100

The SMDK41100 shares nGCS0 signal for Flash memory(U16) and EEPROMs(U18, U19), so users open the JP3 pins to disconnect the nGCS0 signal to EEPROM.

Note : Flash memory should be unprotected before soldering on SMDK41100.

### RUNNING EMBEDDED ICE

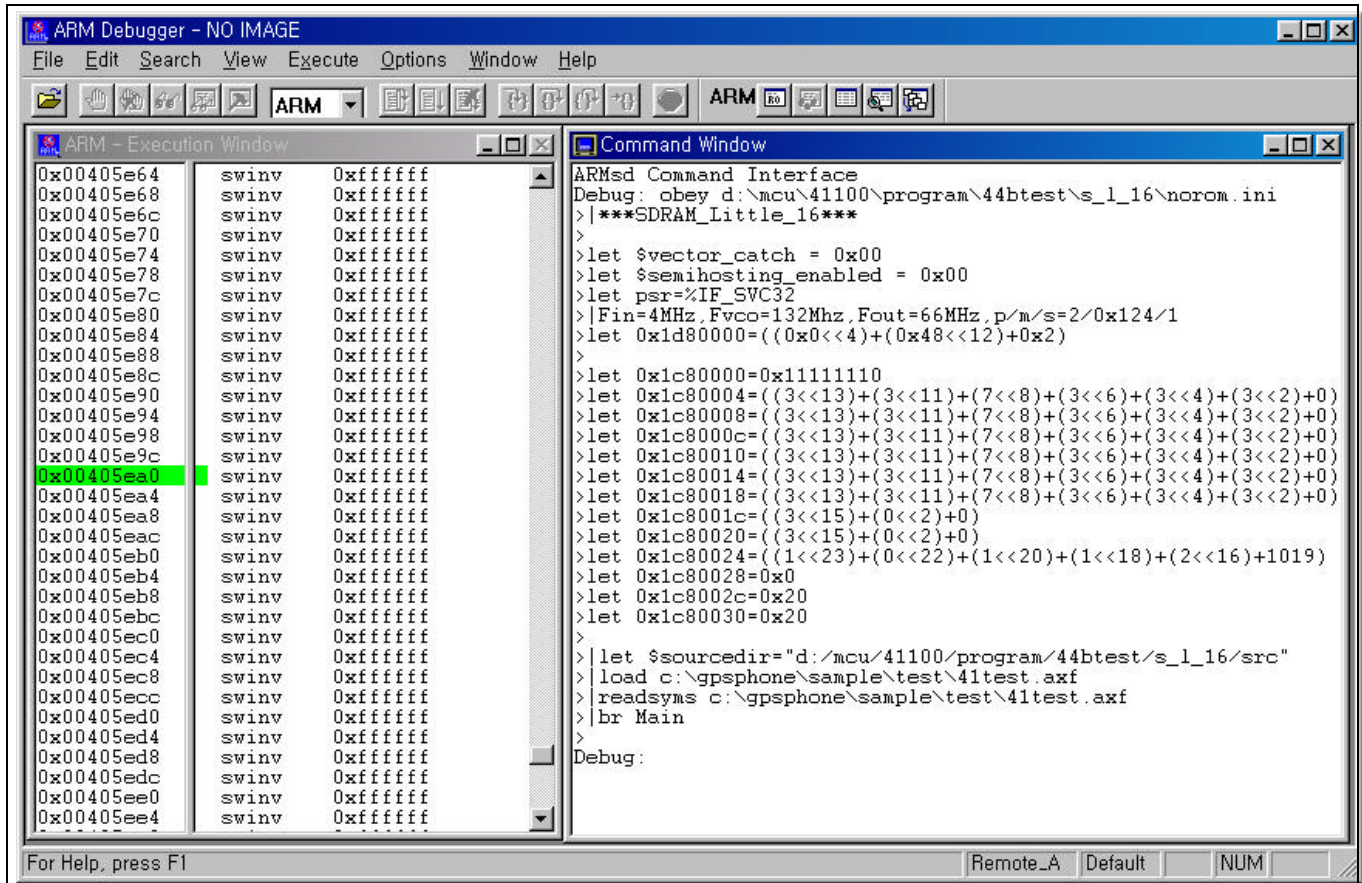
Please refer to page 2-9.





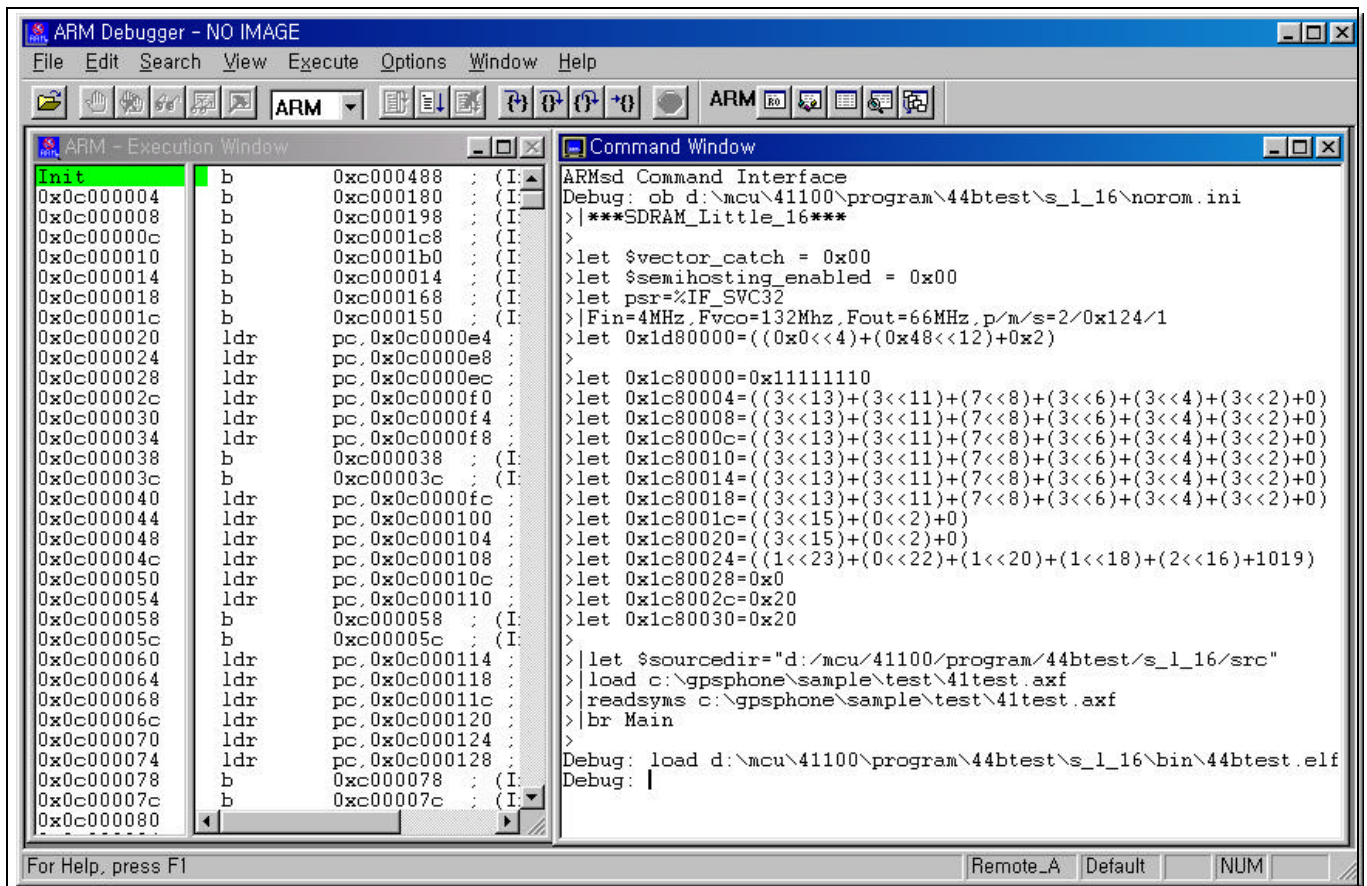
## OBEYING NOROM.INI

Obeying the norom.ini in 44BTEST.zip



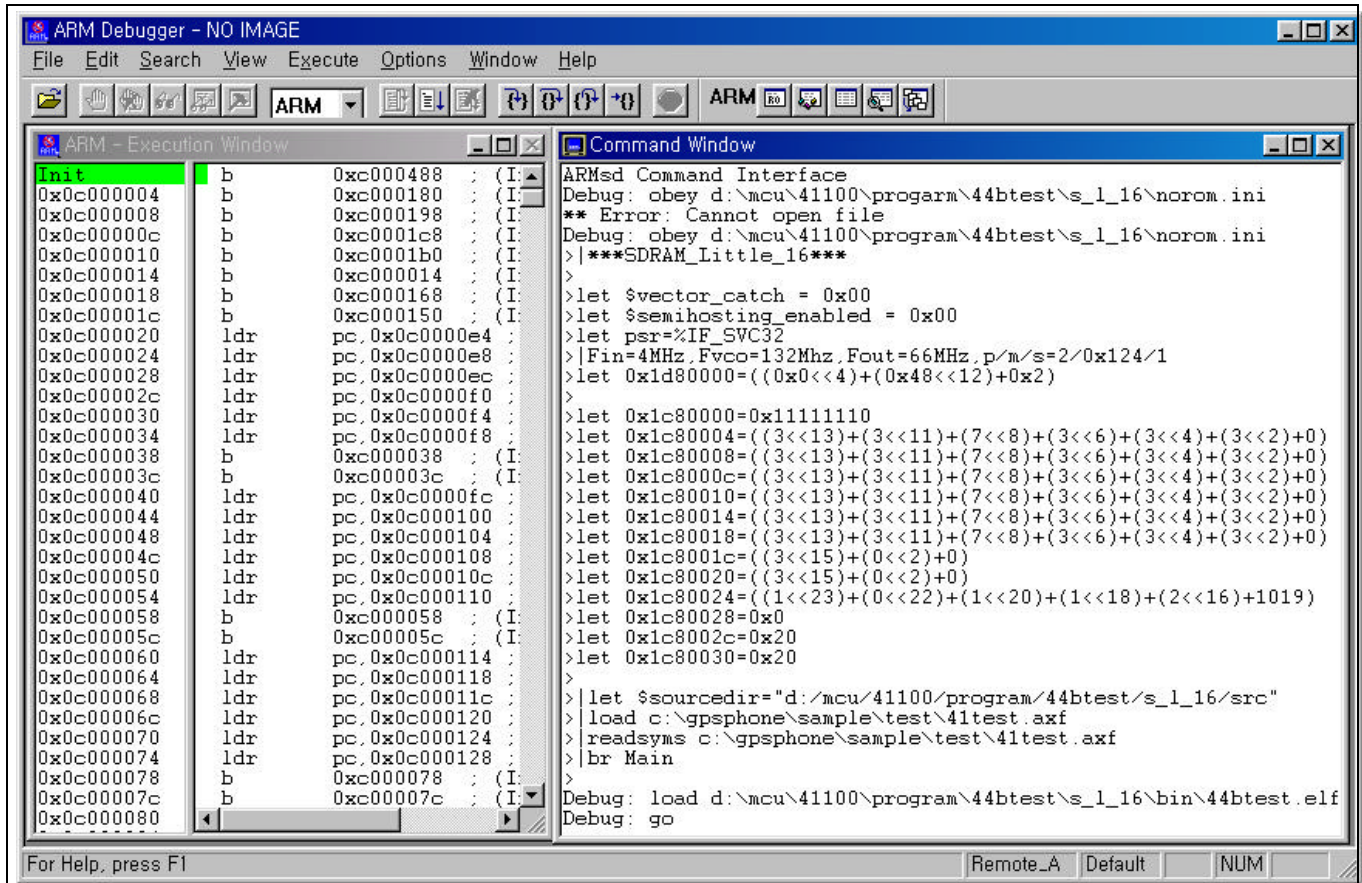
## LOADING 44BTEST.ELF

Load the 44btest.elf file in BIN directory



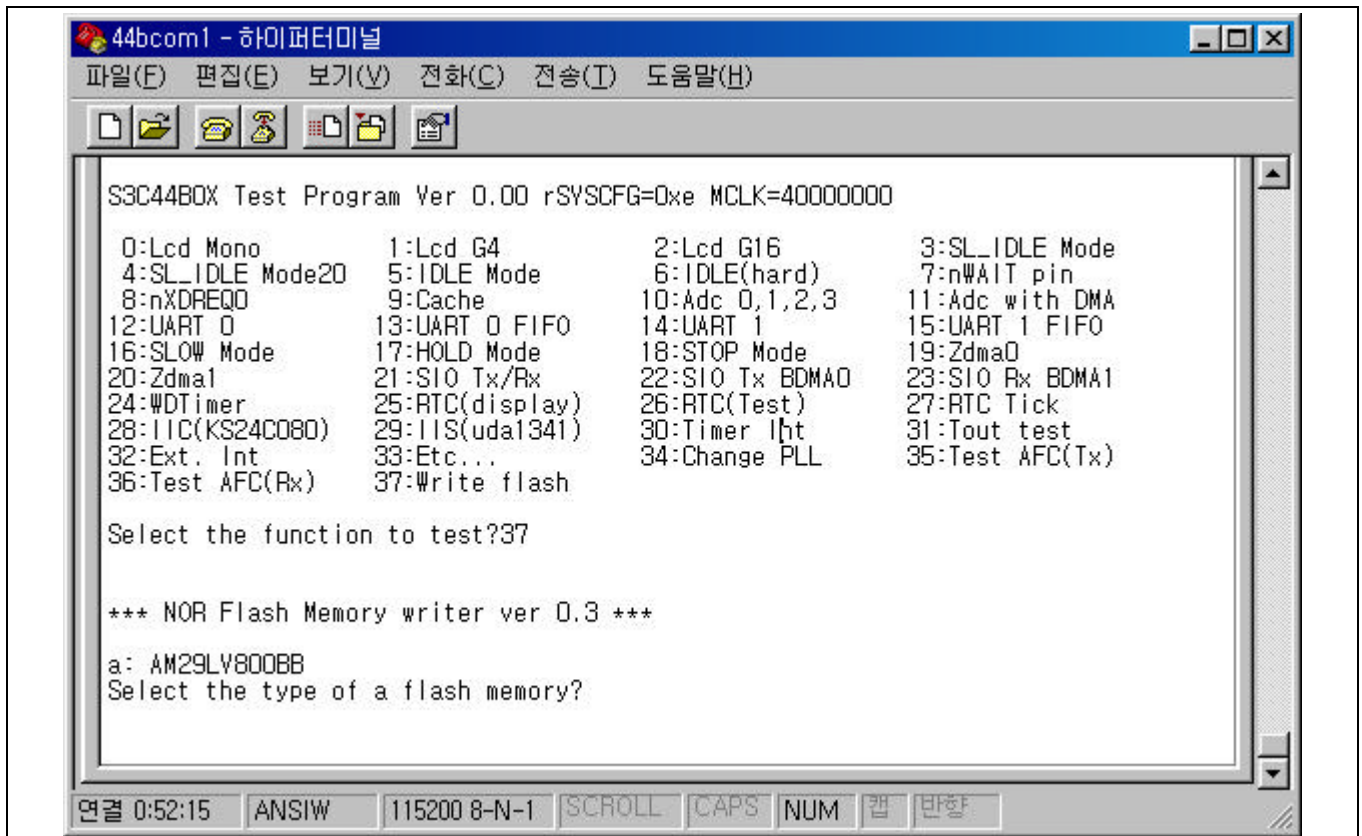
## EXECUTION 44BTEST.ELF

Execute 44btest code with Go command



## EXECUTION WRITE FLASH PROGRAM

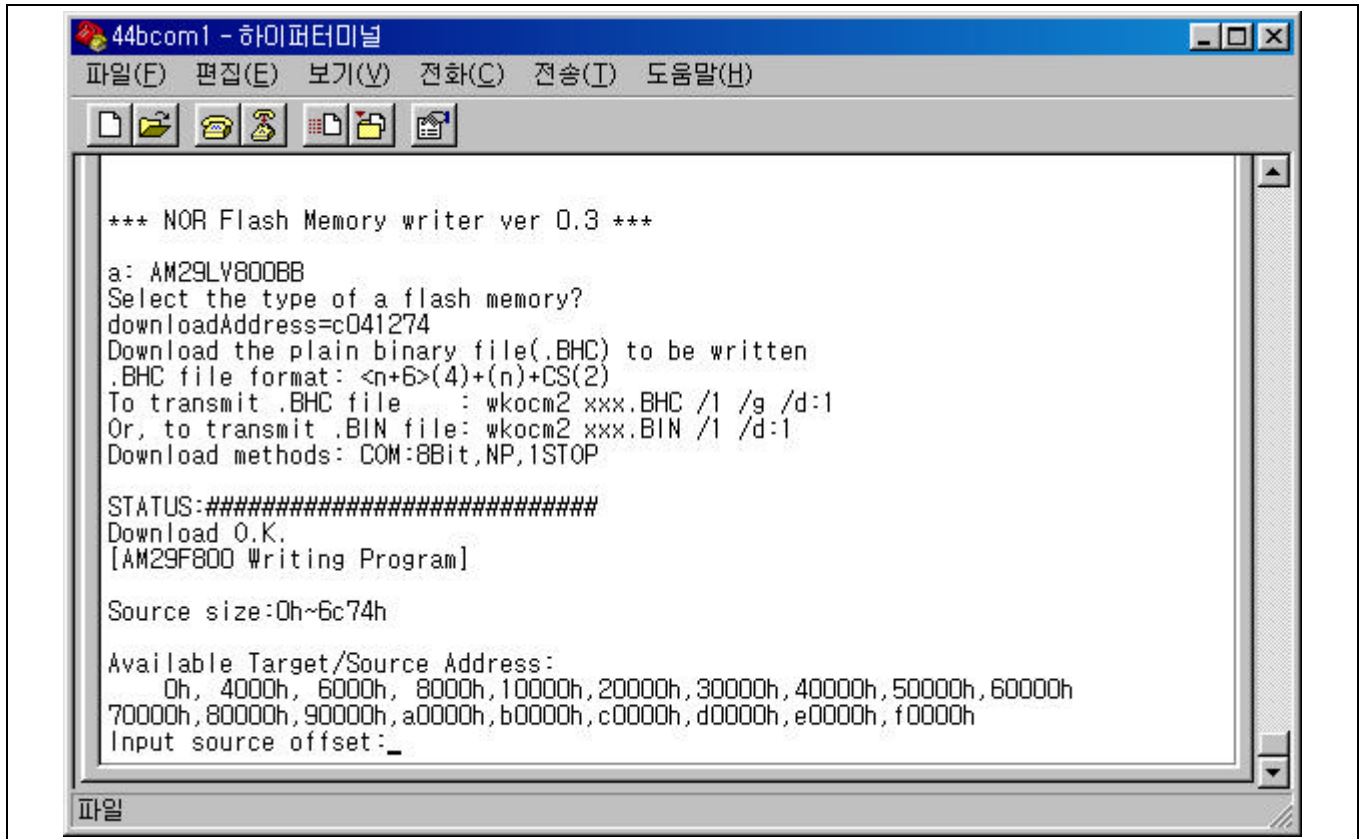
1. Select " 37. Write flash " item in Hyper terminal



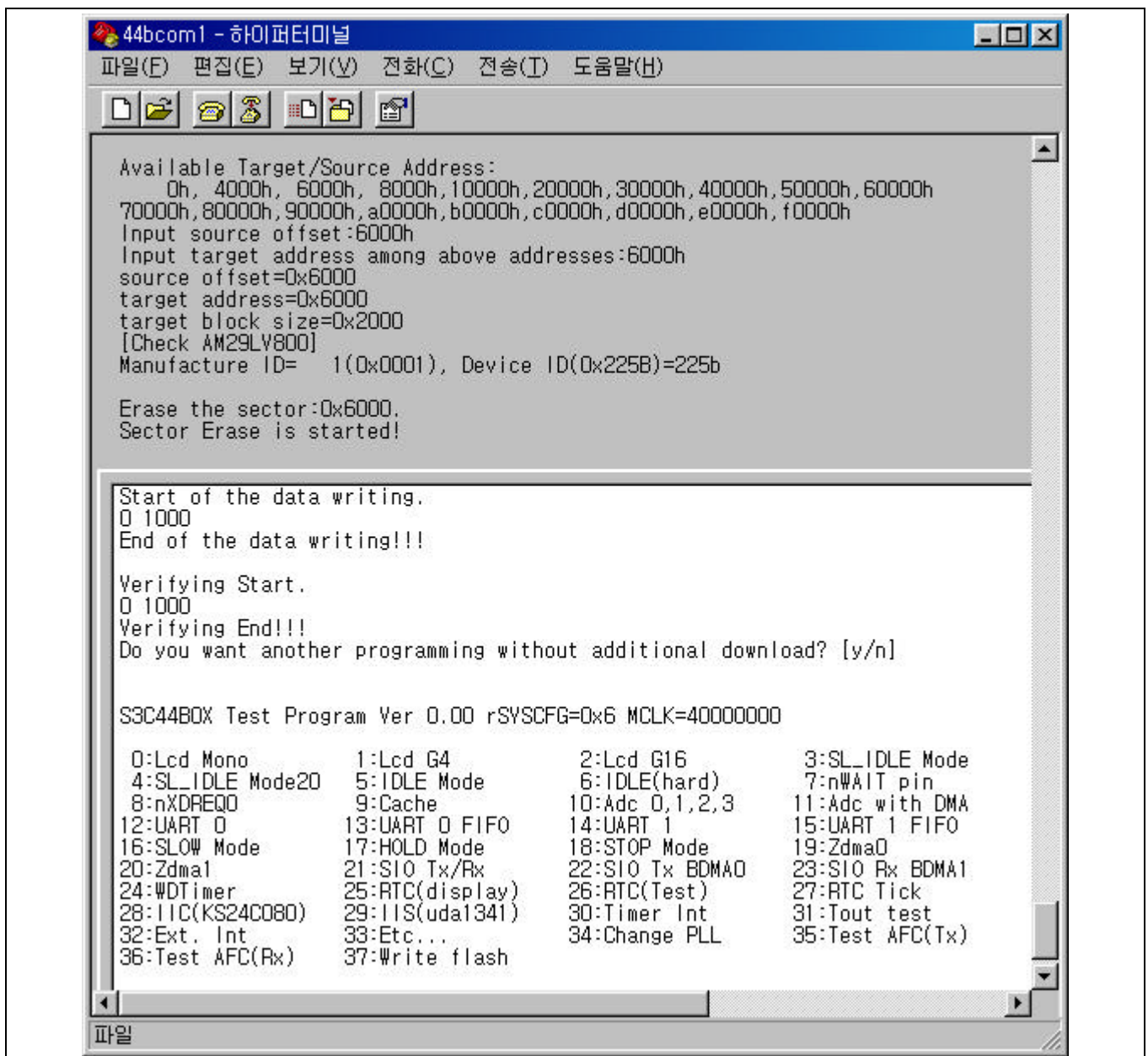
2. Select memory type
  - It is only one type of memory is supported.(AM29LV800BB)



3. Download image file to write to flash memory with wkcom2 utility in DOS command window.



## 4. Write source offset and target offset repeatedly until source size



## 5. Reset SMDK41100

## TRANSLATING MAKEFILE FOR ADS 1.0 FROM SDT 2.50

This document explains about only translating an old makefile for SDT 2.50 to a new makefile for ADS 1.0. If you have used SDT 2.50, it's recommended that you should read the document (ADS, Getting Started, ARM DUI0064A) about the difference between SDT 2.50 and ADS 1.0.

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

1. ARMLINK option
  - first: the path of an object file isn't needed.
2. ARMASM option
  - cpu : should be changed as -cpu ARM7TDMI
  - apcs: should be changed to -apcs /noswst
3. compiler option
  - fc : should be removed.
  - zpz0 : should be removed. This is not needed anymore.
  - apcs : should be changed to -apcs /noswst
  - processor : should be removed.
  - arch : should be removed.
  - cpu : should be added as -cpu ARM7TDMI
4. fromelf.exe
  - nozeropad : should be removed. This is not needed anymore.
  - output : command line style should be changed using -output option as follows;  
fromelf -nodebug -bin -output \$(BIN)\\$(PRJ).bin \$(BIN)\\$(PRJ).elf

### THE OTHER ITEMS CHANGED FOR ADS 1.0

1. ammake.exe
  - The armmake.exe isn't supplied with ADS 1.0. So, 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 library in ADS 1.0 is made for embedded application.
3. There is no tasm.exe. The tasm.exe is merged into armasm.exe

**THE EXAMPLE OF MAKEFILE FOR ADS 1.0**

Pay attention to the bold type font items. The items are different parts with makefile for SDT 2.50.

#### File Definition ####

PRJ = 44btest

INIT= 44binit

AM1 = 44blib\_a

CM1 = 44blib

CM2 = lcd

CM3 = glib

CM4 = lcdlib

CM5 = extdma

CM6 = idle

CM7 = flash

CM8 = am29f800

CM9 = nwait

#### Destination path Definition ####

SRC=.

INC=..\inc

OBJ=..\obj



# 3

## TEST CODE EXAMPLE

### 44BMON & 44BTEST

44BMON is simple serial downloader program. 44BMON receives an application program and execute the program. 44BTEST is the S3C44B0X function test program.

These two programs (44BMON & 44BTEST) are another user's guide. It's very helpful when you make your own application. You can learn the S3C44B0X peripheral usage in this test codes. The memory map of 44BMON and 44BTEST program is in Figure 3-1.

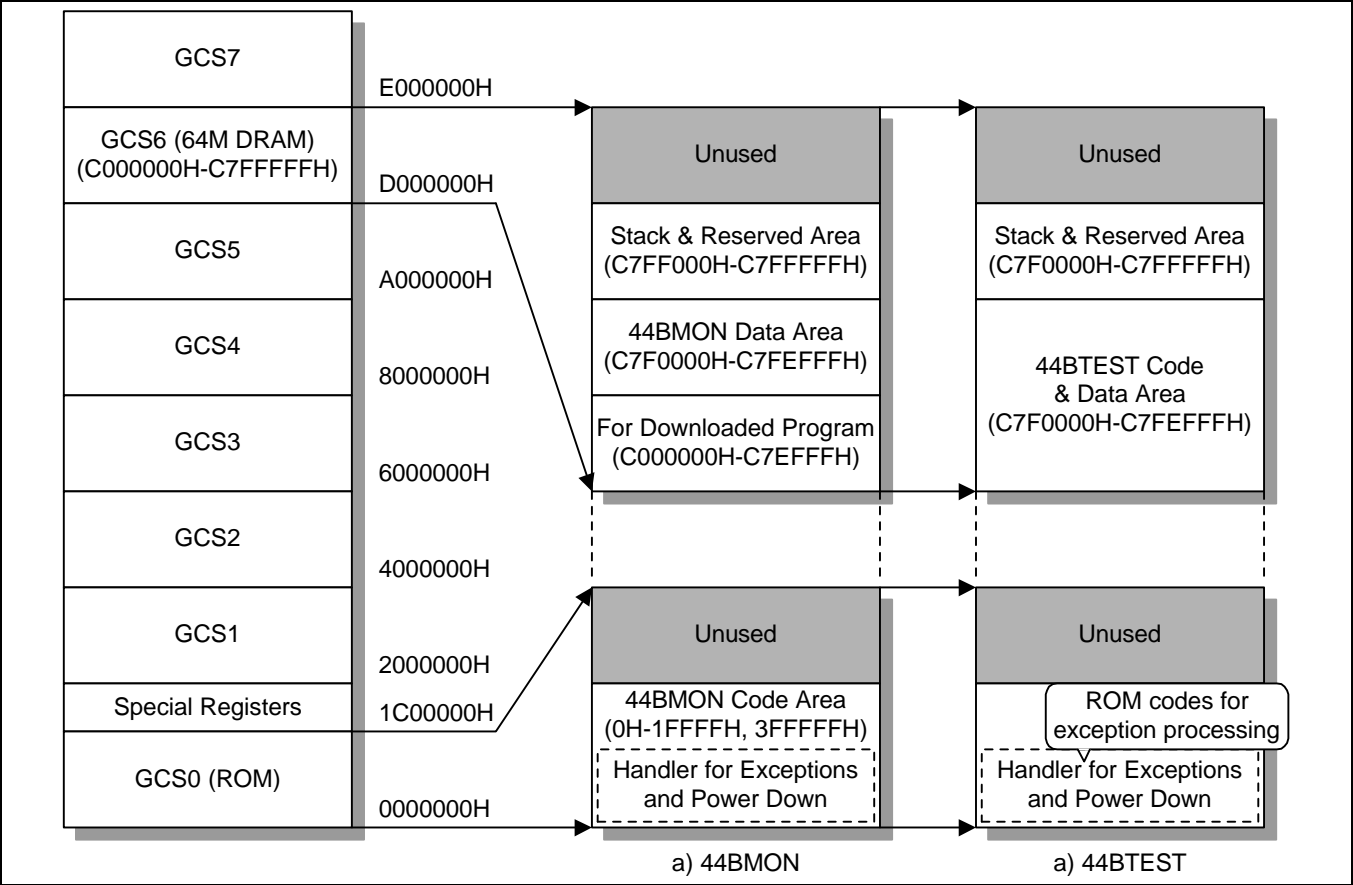


Figure 3-1. Memory Map for 44BMON and 44BTEST

**44BMON FILE DESCRIPTIONS**

Filename	File Descriptions	Page
makefile	Makefile for 44BMON	3-12
option.a	Configuration file for 44binit.s	3-13
memcfg.a	Configuarion file for memory bank control register	3-14
44binit.s	Start-up code	3-16
44blib_a.s	Assembly library functions	3-24
def.h	Header file for data type definitions	3-25
option.h	Header file for board configuration	3-25
44b.h	Header file for S3C44B0X special register definitions	3-26
44blib.h	Header file for 44blib.c	3-33
lcd.h	Header file for 44bmon.c	3-34
44blib.c	C library functions	3-35
44bmon.c	Downloader program main file.	3-42

**NOTE**

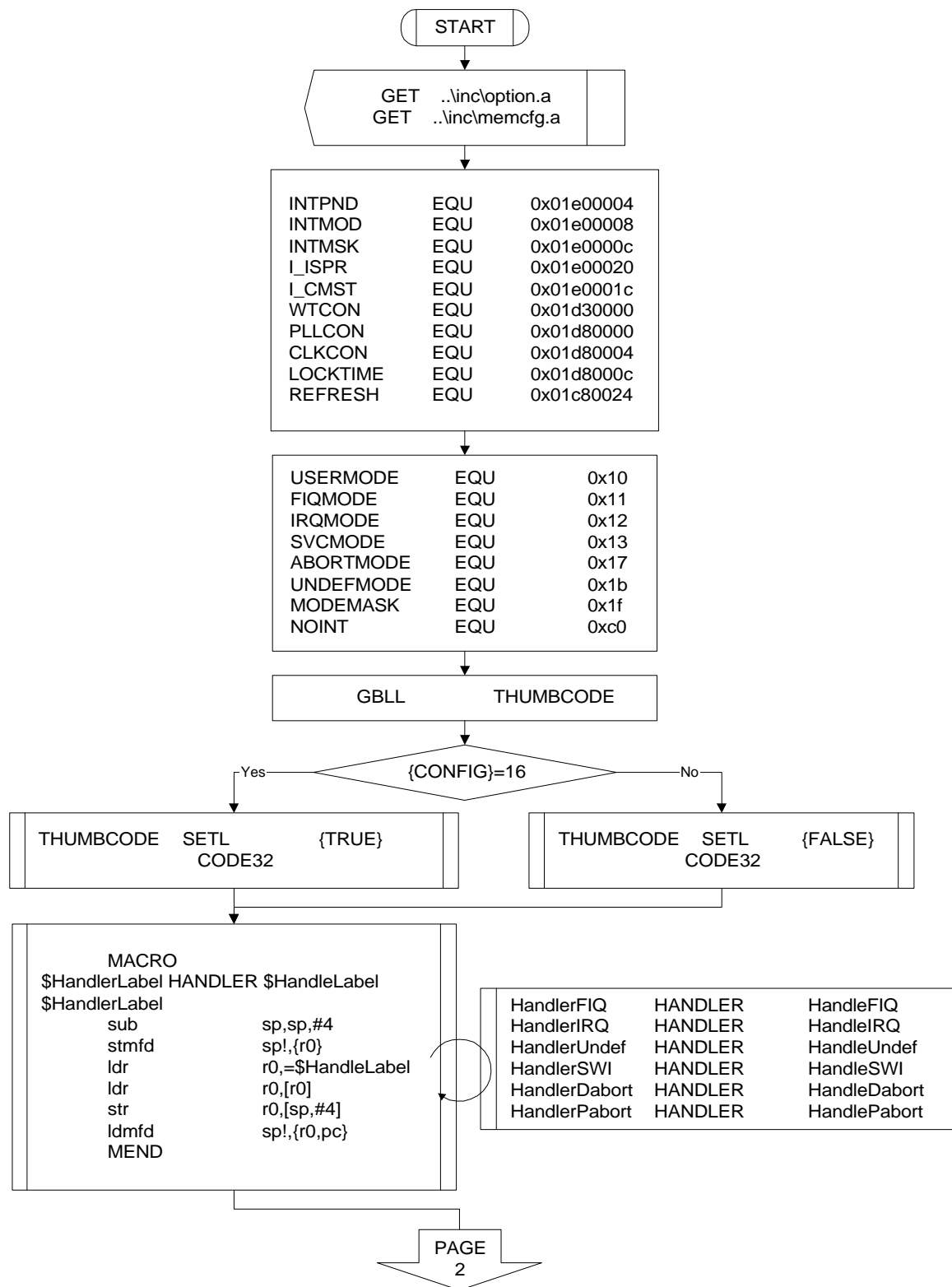
The program source data in this application notes is Sep.29.2000. Please download the latest test code source from our website. (<http://www.samsungsemi.com>)

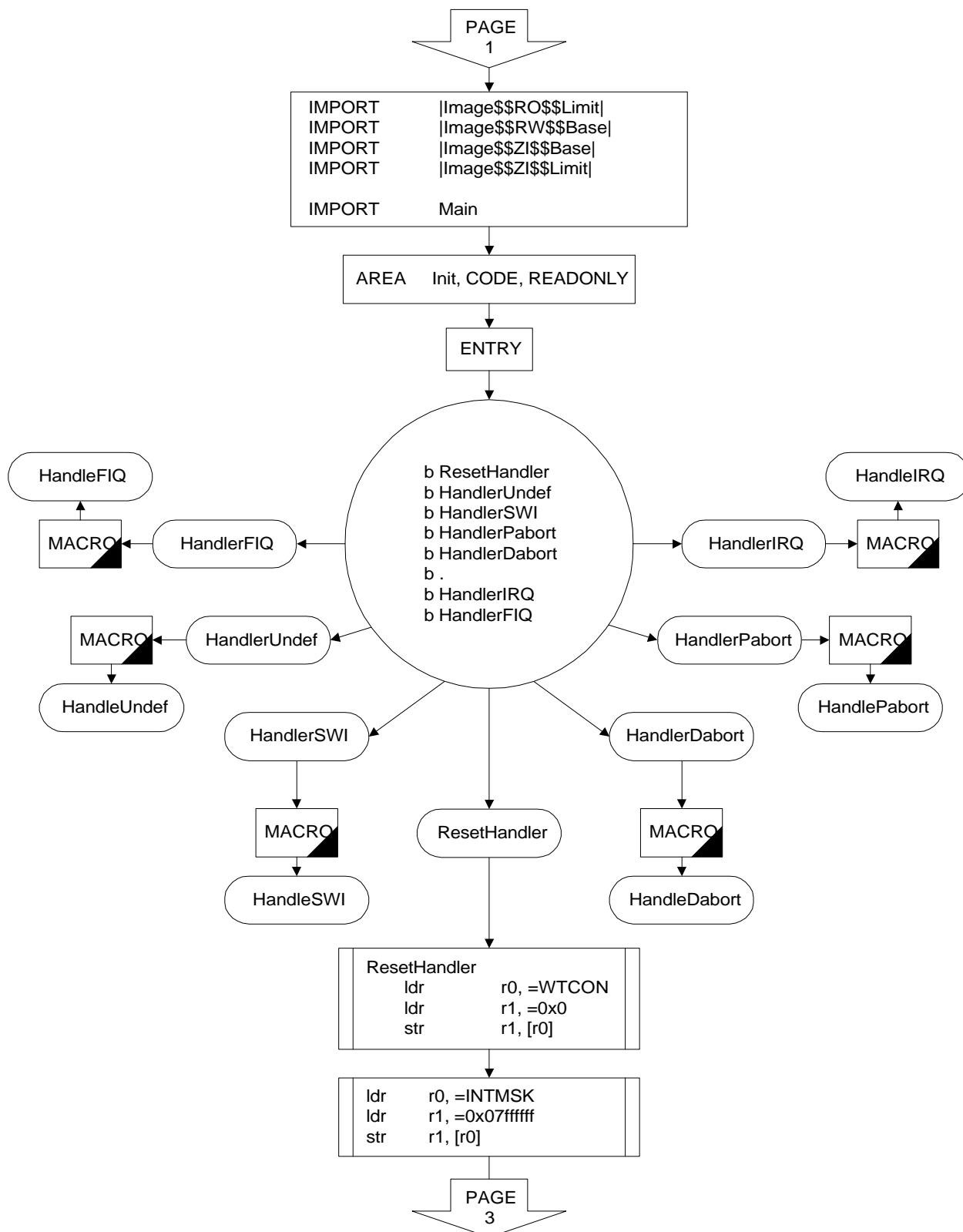
**44BTEST FILE DESCRIPTIONS**

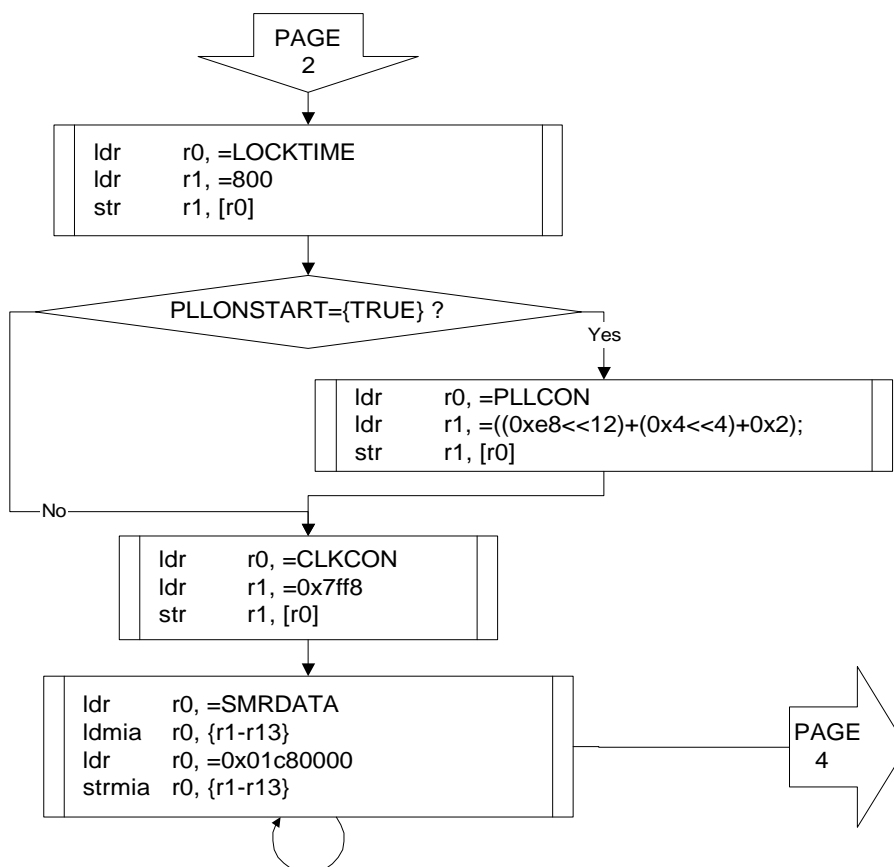
See the page, 3-40. There is 44BTEST file description table.

## FLOW CHART OF 44BINIT

The flow chart of 44binit shows the simple stream of 44binit.s.

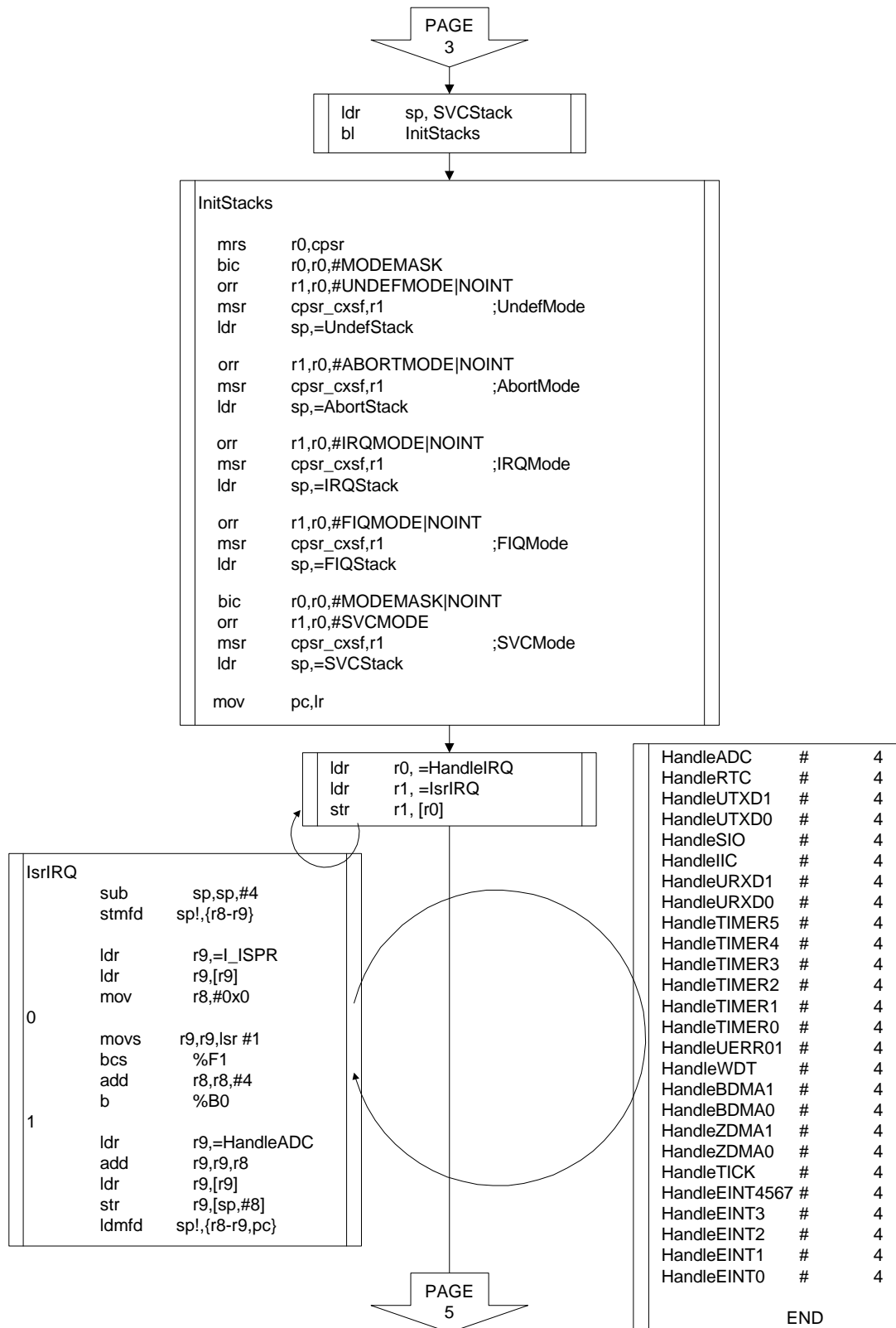


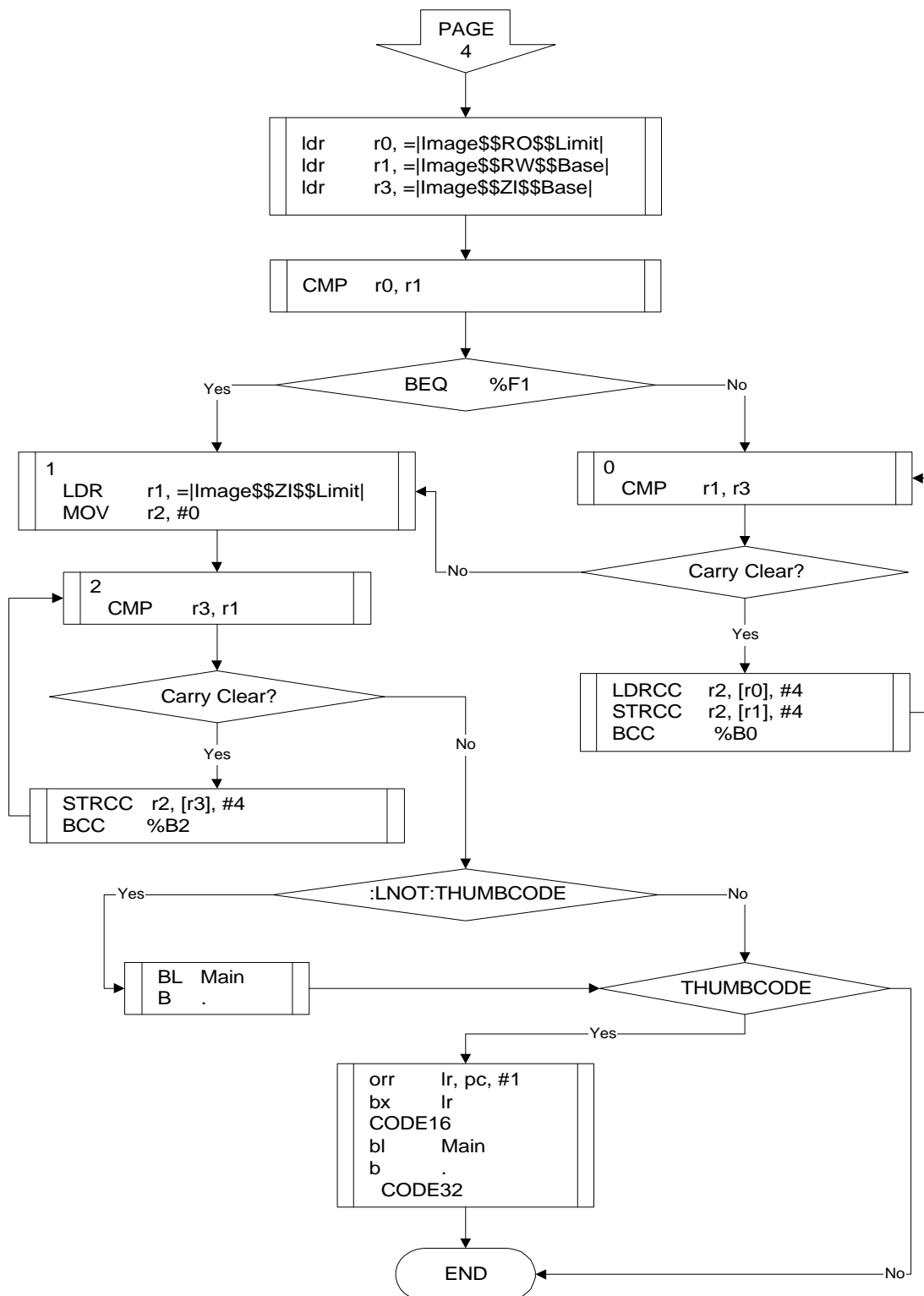




```

SMRDATA      DATA
[ BUSWIDTH=16
  DCD 0x11111110
  |
  DCD 0x22222220
]
DCD ((B0_Tacs<<13)+(B0_Tcos<<11)+(B0_Tacc<<8)+(B0_Tcoh<<6)+(B0_Tah<<4)+(B0_Tacp<<2)+(B0_PMC))
DCD ((B1_Tacs<<13)+(B1_Tcos<<11)+(B1_Tacc<<8)+(B1_Tcoh<<6)+(B1_Tah<<4)+(B1_Tacp<<2)+(B1_PMC))
DCD ((B2_Tacs<<13)+(B2_Tcos<<11)+(B2_Tacc<<8)+(B2_Tcoh<<6)+(B2_Tah<<4)+(B2_Tacp<<2)+(B2_PMC))
DCD ((B3_Tacs<<13)+(B3_Tcos<<11)+(B3_Tacc<<8)+(B3_Tcoh<<6)+(B3_Tah<<4)+(B3_Tacp<<2)+(B3_PMC))
DCD ((B4_Tacs<<13)+(B4_Tcos<<11)+(B4_Tacc<<8)+(B4_Tcoh<<6)+(B4_Tah<<4)+(B4_Tacp<<2)+(B4_PMC))
DCD ((B5_Tacs<<13)+(B5_Tcos<<11)+(B5_Tacc<<8)+(B5_Tcoh<<6)+(B5_Tah<<4)+(B5_Tacp<<2)+(B5_PMC))
[ BDRAMTYPE="DRAM"
  DCD ((B6_MT<<15)+(B6_Trcd<<4)+(B6_Tcas<<3)+(B6_Tcp<<2)+(B6_CAN))
  DCD ((B7_MT<<15)+(B7_Trcd<<4)+(B7_Tcas<<3)+(B7_Tcp<<2)+(B7_CAN))
  |
  DCD ((B6_MT<<15)+(B6_Trcd<<2)+(B6_SCAN))
  DCD ((B7_MT<<15)+(B7_Trcd<<2)+(B7_SCAN))
]
DCD ((REFEN<<23)+(TREFMD<<22)+(Trp<<20)+(Trc<<18)+(Tchr<<16)+REFCNT)
DCD 0x10
DCD 0x20
DCD 0x20
  
```



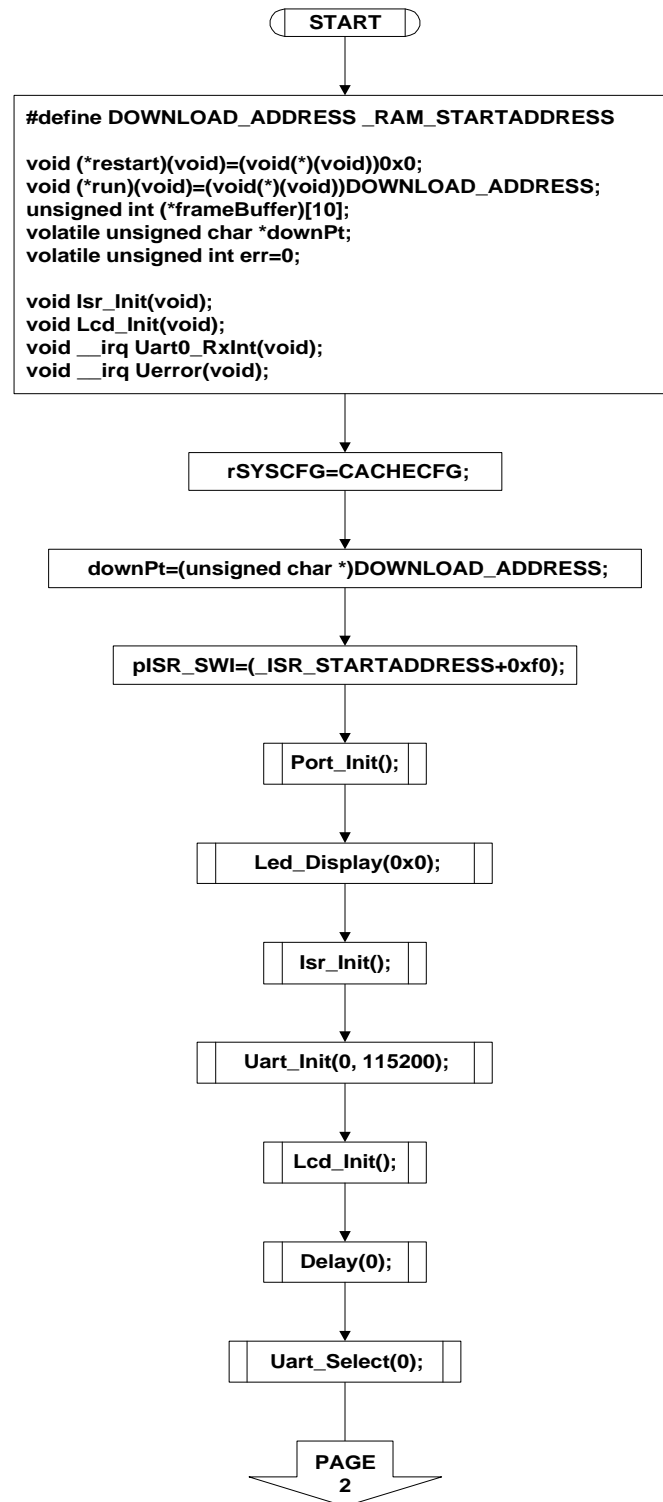


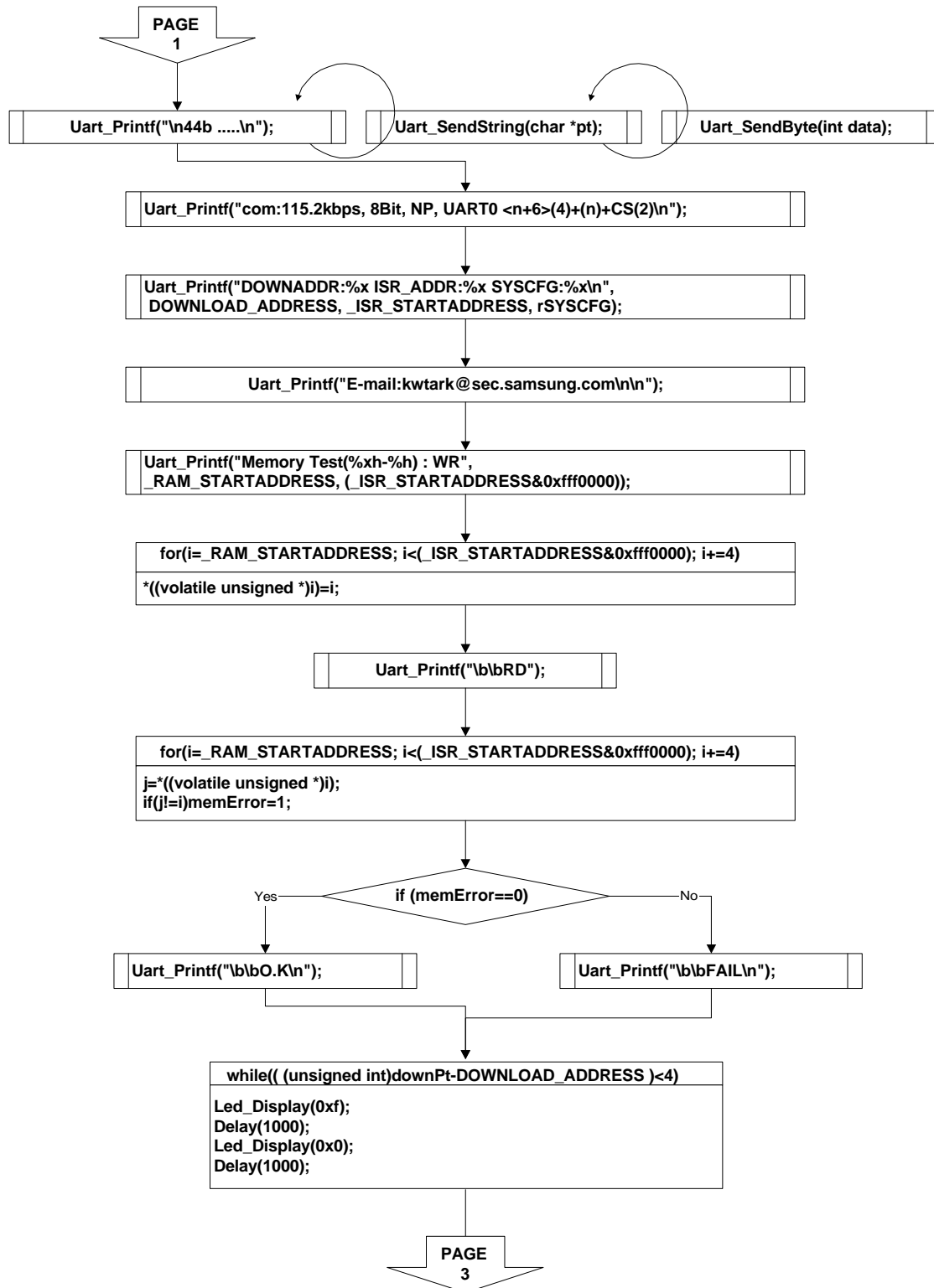
EnterPWDN	
	mov r2, r0
	ldr r0, =REFRESH
	ldr r3, [r0]
	mov r1, r3
	orr r1, r1, #0x400000
	str r1, [r0]
	nop
	nop
	nop
	nop
	nop
	nop
	nop
	ldr r0, =CLKCON
	str r2, [r0]
0	mov r0, #0xff
	subs r0, r0, #1
	bne %B0
	ldr r0, =REFRESH
	str r3, [r0]
	mov pc, lr
LTORG	

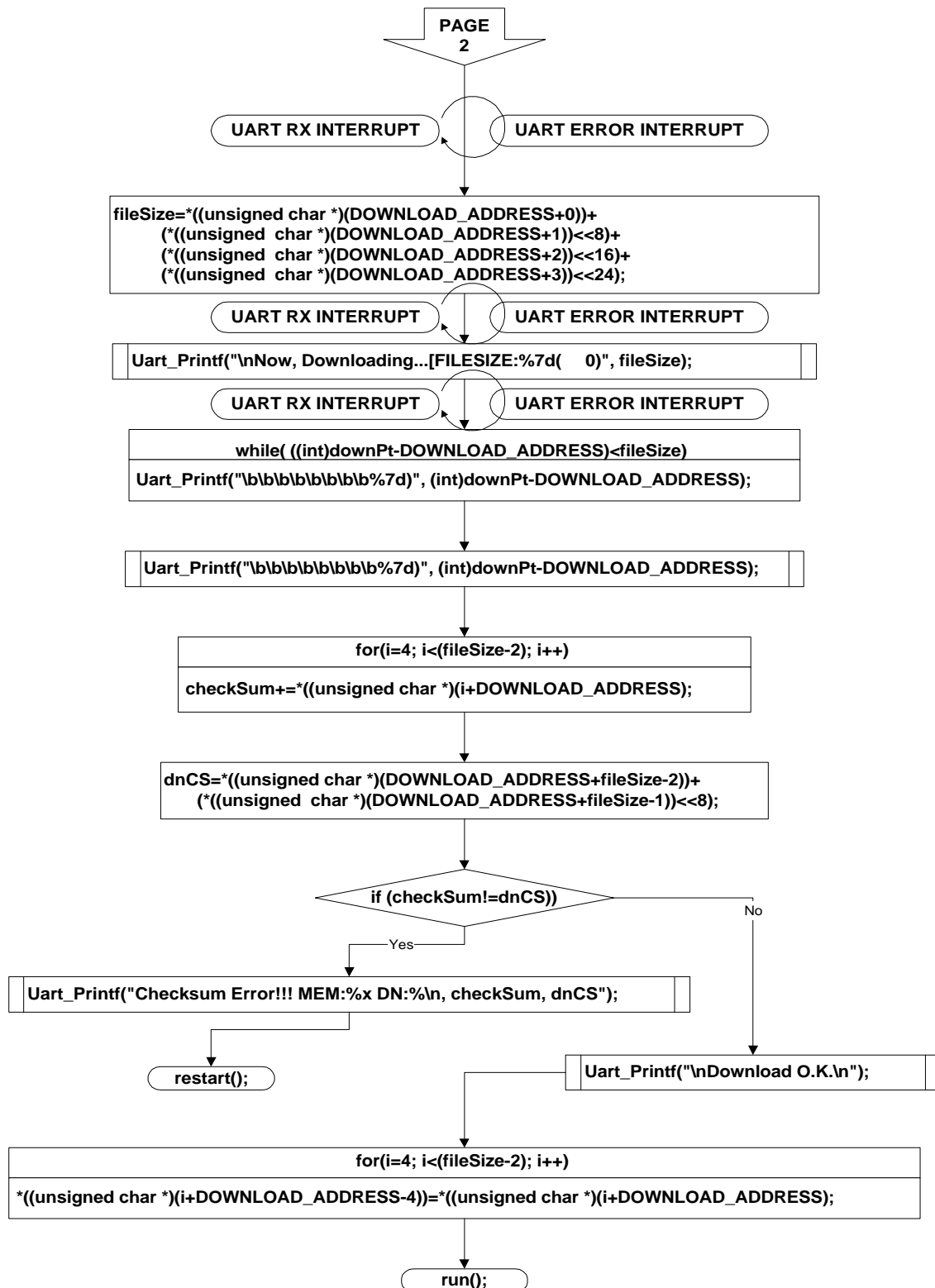


## FLOW CHART OF 44BMON

The flow chart of 44bmon shows the simple stream of 44bmon.c.







**44BMON : Makefile**

```
##### File Definition #####
PRJ = 44bmon
INIT= 44binit
CM1 = 44blib
AM1 = 44blib_a

#### Destination path Definition ####
SRC=.
INC=..\inc
OBJ=..\obj
ERR=..\err
BIN=..\bin

#### ARM tool Definition ####
ARMLINK= armlink
ARMASM = armasm
ARMCC  = armcc

#### Option Definition #####
LFLAGS = -ro-base 0x0 -rw-base 0xc7f0000 -elf -map -xref -list $(BIN)\list.lst
AFLAGS  = -li -apcs 3/32bit/noswst/nofp -cpu ARM7TM
CFLAGS  = -c -g+ -fc -apcs 3/32bit/noswst/nofp -li -processor ARM7TM -arch 4T

#If you doesn't debug,use following CFLAGS for more faster operation.
#CFLAGS = -c -g- -fc -apcs 3/32bit/noswst/nofp -li -processor ARM7TM -arch 4T

#### Object combine Definition ####

OBJS = $(OBJ)\$(PRJ).o      $(OBJ)\$(INIT).o $(OBJ)\$(CM1).o $(OBJ)\$(AM1).o

all: $(BIN)\$(PRJ).elf

$(BIN)\$(PRJ).elf: $(OBJS)
    del $(BIN)\$(PRJ).bin
    del $(BIN)\$(PRJ).elf
    $(ARMLINK) $(LFLAGS) -first $(OBJ)\$(INIT).o(Init) -o $(BIN)\$(PRJ).elf $(OBJS)
    fromelf -nodebug -nozeropad $(BIN)\$(PRJ).elf -bin $(BIN)\$(PRJ).bin
    fromelf $(BIN)\$(PRJ).elf -text/s $(BIN)\syms.sym
    fromelf $(BIN)\$(PRJ).elf -text/c $(BIN)\symc.sym

$(OBJ)\$(PRJ).o : $(SRC)\$(PRJ).c $(INC)\lcd.h $(INC)\44b.h $(INC)\44blib.h makefile
    del $(OBJ)\$(PRJ).o
    del $(ERR)\$(PRJ).err
    $(ARMCC) $(CFLAGS) $(SRC)\$(PRJ).c -o $(OBJ)\$(PRJ).o -Errors $(ERR)\$(PRJ).err

$(OBJ)\$(INIT).o: $(SRC)\$(INIT).s makefile
    del $(OBJ)\$(INIT).o
    del $(ERR)\$(INIT).err
    $(ARMASM) $(AFLAGS) $(SRC)\$(INIT).s -o $(OBJ)\$(INIT).o -Errors $(ERR)\$(INIT).err

$(OBJ)\$(CM1).o: $(SRC)\$(CM1).c $(INC)\44b.h $(INC)\44blib.h makefile
    del $(OBJ)\$(CM1).o
    del $(ERR)\$(CM1).err
    $(ARMCC) $(CFLAGS) $(SRC)\$(CM1).c -o $(OBJ)\$(CM1).o -Errors $(ERR)\$(CM1).err

$(OBJ)\$(AM1).o: $(SRC)\$(AM1).s $(INC)\44b.h $(INC)\44blib.h makefile
    del $(OBJ)\$(AM1).o
    del $(ERR)\$(AM1).err
    $(ARMASM) $(AFLAGS) $(SRC)\$(AM1).s -o $(OBJ)\$(AM1).o -Errors $(ERR)\$(AM1).err
```

**44BMON : option.a**

```

;*****OPTIONS*****
;_RAM_STARTADDRESS EQU    0xc000000
;_ISR_STARTADDRESS EQU    0xc7fff00 ;GCS6:64M DRAM/SDRAM
;_ISR_STARTADDRESS EQU    0xc1fff00 ;GCS6:16M DRAM

;BUSWIDTH; 16,32
                GBLA    BUSWIDTH
BUSWIDTH       SETA     16

; "DRAM", "SDRAM"
                GBLA    BDRAMTYPE
BDRAMTYPE      SETS     "SDRAM"

;This value has to be TRUE on ROM program.
;This value has to be FALSE in RAM program.
                GBLA    PLLONSTART
PLLONSTART     SETL     {TRUE}

                GBLA    PLLCLK
PLLCLK         SETA     60000000

                [        PLLCLK = 40000000
M_DIV EQU      0x48    ;Fin=10MHz Fout=40MHz
P_DIV EQU      0x3
S_DIV EQU      0x2
                ]

                [        PLLCLK = 50000000
M_DIV EQU      0x2a    ;Fin=10MHz Fout=50MHz
P_DIV EQU      0x3
S_DIV EQU      0x1
                ]

                [        PLLCLK = 60000000
M_DIV EQU      0x34    ;Fin=10MHz Fout=60MHz
P_DIV EQU      0x3
S_DIV EQU      0x1
                ]

                [        PLLCLK = 75000000
M_DIV EQU      0x43    ;Fin=10MHz Fout=75MHz
P_DIV EQU      0x3
S_DIV EQU      0x1
                ]
;*****
END

```

**44BMON : memcfg.a**

```
*****MEMORY CONTROL PARAMETERS*****
```

```
;Bank 0 parameter
```

```
B0_Tacs      EQU    0x0    ;0clk
B0_Tcos      EQU    0x0    ;0clk
B0_Tacc      EQU    0x6    ;10clk
B0_Tcoh      EQU    0x0    ;0clk
B0_Tah       EQU    0x0    ;0clk
B0_Tacp      EQU    0x0    ;0clk
B0_PMC       EQU    0x0    ;normal(1data)
```

```
;Bank 1 parameter
```

```
B1_Tacs      EQU    0x3    ;4clk
B1_Tcos      EQU    0x3    ;4clk
B1_Tacc      EQU    0x7    ;14clk
B1_Tcoh      EQU    0x3    ;4clk
B1_Tah       EQU    0x3    ;4clk
B1_Tacp      EQU    0x3    ;6clk
B1_PMC       EQU    0x0    ;normal(1data)
```

```
;Bank 2 parameter
```

```
B2_Tacs      EQU    0x3    ;4clk
B2_Tcos      EQU    0x3    ;4clk
B2_Tacc      EQU    0x7    ;14clk
B2_Tcoh      EQU    0x3    ;4clk
B2_Tah       EQU    0x3    ;4clk
B2_Tacp      EQU    0x3    ;6clk
B2_PMC       EQU    0x0    ;normal(1data)
```

```
;Bank 3 parameter
```

```
B3_Tacs      EQU    0x3    ;4clk
B3_Tcos      EQU    0x3    ;4clk
B3_Tacc      EQU    0x7    ;14clk
B3_Tcoh      EQU    0x3    ;4clk
B3_Tah       EQU    0x3    ;4clk
B3_Tacp      EQU    0x3    ;6clk
B3_PMC       EQU    0x0    ;normal(1data)
```

```
;Bank 4 parameter
```

```
B4_Tacs      EQU    0x3    ;4clk
B4_Tcos      EQU    0x3    ;4clk
B4_Tacc      EQU    0x7    ;14clk
B4_Tcoh      EQU    0x3    ;4clk
B4_Tah       EQU    0x3    ;4clk
B4_Tacp      EQU    0x3    ;6clk
B4_PMC       EQU    0x0    ;normal(1data)
```

```
;Bank 5 parameter
```

```
B5_Tacs      EQU    0x3    ;4clk
B5_Tcos      EQU    0x3    ;4clk
B5_Tacc      EQU    0x7    ;14clk
B5_Tcoh      EQU    0x3    ;4clk
B5_Tah       EQU    0x3    ;4clk
B5_Tacp      EQU    0x3    ;6clk
B5_PMC       EQU    0x0    ;normal(1data)
```

```

;Bank 6(if SROM) parameter
B6_Tacs      EQU    0x3    ;4clk
B6_Tcos      EQU    0x3    ;4clk
B6_Tacc      EQU    0x7    ;14clk
B6_Tcoh      EQU    0x3    ;4clk
B6_Tah       EQU    0x3    ;4clk
B6_Tacp      EQU    0x3    ;6clk
B6_PMC       EQU    0x0    ;normal(1data)

;Bank 7(if SROM) parameter
B7_Tacs      EQU    0x3    ;4clk
B7_Tcos      EQU    0x3    ;4clk
B7_Tacc      EQU    0x7    ;14clk
B7_Tcoh      EQU    0x3    ;4clk
B7_Tah       EQU    0x3    ;4clk
B7_Tacp      EQU    0x3    ;6clk
B7_PMC       EQU    0x0    ;normal(1data)

;Bank 6 parameter
[ BDRAMTYPE="DRAM" ;MT=01(FP DRAM) or 10(EDO DRAM)
B6_MT        EQU    0x2    ;EDO DRAM
B6_Trcd      EQU    0x0    ;1clk
B6_Tcas      EQU    0x0    ;1clk
B6_Tcp       EQU    0x0    ;1clk
B6_CAN       EQU    0x2    ;10bit
| ; "SDRAM"          ;MT=11(SDRAM)
B6_MT        EQU    0x3    ;SDRAM
B6_Trcd      EQU    0x0    ;2clk
B6_SCAN      EQU    0x0    ;8bit
]

;Bank 7 parameter
[ BDRAMTYPE="DRAM" ;MT=01(FP DRAM) or 10(EDO DRAM)
B7_MT        EQU    0x2    ;EDO DRAM
B7_Trcd      EQU    0x0    ;2clk
B7_Tcas      EQU    0x0    ;2clk
B7_Tcp       EQU    0x0    ;2clk
B7_CAN       EQU    0x2    ;10bit
| ; "SDRAM"          ;MT=11(SDRAM)
B7_MT        EQU    0x3    ;SDRAM
B7_Trcd      EQU    0x0    ;2clk
B7_SCAN      EQU    0x0    ;8bit
]

;REFRESH parameter
REFEN        EQU    0x1    ;Refresh enable
TREFMD       EQU    0x0    ;CBR(CAS before RAS)/Auto refresh
Trp          EQU    0x0    ;2clk
Trc          EQU    0x1    ;5clk
Tchr         EQU    0x2    ;3clk
REFCNT       EQU    1113   ;period=15.6us, MCLK=60Mhz
;*****
END

```

**44BMON : 44binit.s**

```

; *****
; * NAME      : 44BINIT.S
; * Version   : 10.April.2000
; * Description:
; *   C start up codes
; *   Configure memory, Initialize ISR ,stacks
; *   Initialize C-variables
; *   Fill zeros into zero-initialized C-variables
; *****

    GET ..\inc\option.a
    GET ..\inc\memcfg.a

;Memory Area
;GCS6 64M 16bit(8MB) DRAM/SDRAM(0xc000000-0xc7ffff)
;APP   RAM=0xc000000~0xc7effff
;44BMON RAM=0xc7f0000-0xc7ffff
;STACK =0xc7ffa00

;Interrupt Control
INTPND EQU 0x01e00004
INTMOD EQU 0x01e00008
INTMSK EQU 0x01e0000c
I_ISPR EQU 0x01e00020
I_CMST EQU 0x01e0001c

;Watchdog timer
WTCN EQU 0x01d30000

;Clock Controller
PLLCON EQU 0x01d80000
CLKCON EQU 0x01d80004
LOCKTIME EQU 0x01d8000c

;Memory Controller
REFRESH EQU 0x01c80024

;Pre-defined constants
USERMODE EQU 0x10
FIQMODE EQU 0x11
IRQMODE EQU 0x12
SVCMODE EQU 0x13
ABORTMODE EQU 0x17
UNDEFMODE EQU 0x1b
MODEMASK EQU 0x1f
NOINT EQU 0xc0

;check if tasm.exe is used.
    GBL THUMBCODE
    [ {CONFIG} = 16
    THUMBCODE SETL {TRUE}
    CODE32
    |
    THUMBCODE SETL {FALSE}
    ]

    [ THUMBCODE
    CODE32 ;for start-up code for Thumb mode
    ]

```



```

MACRO
$HandlerLabel HANDLER $HandleLabel

$HandlerLabel
    sub        sp,sp,#4            ;decrement sp(to store jump address)
    stmfd      sp!,{r0}           ;PUSH the work register to stack(lr does't push because it
return to original address)
    ldr        r0,=$HandlerLabel;load the address of HandleXXX to r0
    ldr        r0,[r0]           ;load the contents(service routine start address) of
HandleXXX
    str        r0,[sp,#4]         ;store the contents(ISR) of HandleXXX to stack
    ldmfd      sp!,{r0,pc}       ;POP the work register and pc(jump to ISR)
MEND

IMPORT |Image$$RO$$Limit| ; End of ROM code (=start of ROM data)
IMPORT |Image$$RW$$Base| ; Base of RAM to initialise
IMPORT |Image$$ZI$$Base| ; Base and limit of area
IMPORT |Image$$ZI$$Limit| ; to zero initialise

IMPORT Main ; The main entry of mon program

AREA Init, CODE, READONLY

ENTRY
b ResetHandler ;for debug
b HandlerUndef ;handlerUndef
b HandlerSWI ;SWI interrupt handler
b HandlerPabort ;handlerPAbort
b HandlerDabort ;handlerDAbort
b . ;handlerReserved
b HandlerIRQ
b HandlerFIQ
;***IMPORTANT NOTE***
;If the H/W vectored interrupt mode is enabled, The above two instructions should
;be changed like below, to work-around with H/W bug of S3C44B0X interrupt controller.
; b HandlerIRQ -> subs pc,lr,#4
; b HandlerFIQ -> subs pc,lr,#4

VECTOR_BRANCH
    ldr pc,=HandlerEINT0 ;mGA H/W interrupt vector table
    ldr pc,=HandlerEINT1 ;
    ldr pc,=HandlerEINT2 ;
    ldr pc,=HandlerEINT3 ;
    ldr pc,=HandlerEINT4567 ;
    ldr pc,=HandlerTICK ;mGA
    b .
    b .
    ldr pc,=HandlerZDMA0 ;mGB
    ldr pc,=HandlerZDMA1 ;
    ldr pc,=HandlerBDMA0 ;
    ldr pc,=HandlerBDMA1 ;
    ldr pc,=HandlerWDT ;
    ldr pc,=HandlerUERR01 ;mGB
    b .
    b .
    ldr pc,=HandlerTIMER0 ;mGC
    ldr pc,=HandlerTIMER1 ;
    ldr pc,=HandlerTIMER2 ;
    ldr pc,=HandlerTIMER3 ;
    ldr pc,=HandlerTIMER4 ;
    ldr pc,=HandlerTIMER5 ;mGC
    b .
    b .

```

```

ldr pc,=HandlerURXD0      ;mGD
ldr pc,=HandlerURXD1      ;
ldr pc,=HandlerIIC        ;
ldr pc,=HandlerSIO        ;
ldr pc,=HandlerUTXD0      ;
ldr pc,=HandlerUTXD1      ;mGD
b .
b .
ldr pc,=HandlerRTC        ;mGKA
b .                        ;
b .                        ;
b .                        ;
b .                        ;
b .                        ;mGKA
b .
b .
ldr pc,=HandlerADC        ;mGKB
b .                        ;
b .                        ;
b .                        ;
b .                        ;
b .                        ;mGKB
b .
b .
;0xe0=EnterPWDN
ldr pc,=EnterPWDN

LTORG

HandlerFIQ                HANDLER HandleFIQ
HandlerIRQ                HANDLER HandleIRQ
HandlerUndef              HANDLER HandleUndef
HandlerSWI                HANDLER HandleSWI
HandlerDabort             HANDLER HandleDabort
HandlerPabort             HANDLER HandlePabort

HandlerADC                HANDLER HandleADC
HandlerRTC                HANDLER HandleRTC
HandlerUTXD1              HANDLER HandleUTXD1
HandlerUTXD0              HANDLER HandleUTXD0
HandlerSIO                HANDLER HandleSIO
HandlerIIC                HANDLER HandleIIC
HandlerURXD1              HANDLER HandleURXD1
HandlerURXD0              HANDLER HandleURXD0
HandlerTIMER5             HANDLER HandleTIMER5
HandlerTIMER4             HANDLER HandleTIMER4
HandlerTIMER3             HANDLER HandleTIMER3
HandlerTIMER2             HANDLER HandleTIMER2
HandlerTIMER1             HANDLER HandleTIMER1
HandlerTIMER0             HANDLER HandleTIMER0
HandlerUERR01             HANDLER HandleUERR01
HandlerWDT                HANDLER HandleWDT
HandlerBDMA1              HANDLER HandleBDMA1
HandlerBDMA0              HANDLER HandleBDMA0
HandlerZDMA1              HANDLER HandleZDMA1
HandlerZDMA0              HANDLER HandleZDMA0
HandlerTICK               HANDLER HandleTICK
HandlerEINT4567           HANDLER HandleEINT4567
HandlerEINT3              HANDLER HandleEINT3
HandlerEINT2              HANDLER HandleEINT2
HandlerEINT1              HANDLER HandleEINT1
HandlerEINT0              HANDLER HandleEINT0

```

;One of the following two routines can be used for non-vectorized interrupt.

IsrIRQ ;using I\_ISPR register.

```
Sub      sp,sp,#4      ;reserved for PC
stmfd    sp!,{r8-r9}
```

;IMPORTANT CAUTION

;if I\_ISPC isn't used properly, I\_ISPR can be 0 in this routine.

```
ldr      r9,=I_ISPR
ldr      r9,[r9]
```

```
cmp      r9, #0x0      ;If the IDLE mode work-around is used,
                        ;r9 may be 0 sometimes.
beq      %F2
```

```
mov      r8,#0x0
```

```
0  movs   r9,r9,lsr #1
    bcs   %F1
    add   r8,r8,#4
    b     %B0
```

```
1  ldr      r9,=HandleADC
    add     r9,r9,r8
    ldr     r9,[r9]
    str     r9,[sp,#8]
    ldmfd   sp!,{r8-r9,pc}
```

```
2  ldmfd   sp!,{r8-r9}
    add     sp,sp,#4
    subs    pc,lr,#4
```

```
;*****
;*      START                                     *
;*****
```

ResetHandler

```
ldr      r0,=WTCN      ;watch dog disable
ldr      r1,=0x0
str      r1,[r0]
```

```
ldr      r0,=INTMSK
ldr      r1,=0x07ffffff ;all interrupt disable
str      r1,[r0]
```

```
;*****
;* Set clock control registers                      *
;*****
ldr      r0,=LOCKTIME
ldr      r1,=0xffff
str      r1,[r0]
```

```
[ PLLONSTART
  ldr     r0,=PLLCON      ;temporary setting of PLL
  ldr     r1,=((M_DIV<<12)+(P_DIV<<4)+S_DIV)      ;Fin=10MHz,Fout=40MHz
  str     r1,[r0]
]
```

```

ldr      r0,=CLKCON
ldr      r1,=0x7ff8          ;All unit block CLK enable
str      r1,[r0]

;*****
;* Set memory control registers      *
;*****
ldr      r0,=SMRDATA
ldmia    r0,{r1-r13}
ldr      r0,=0x01c80000      ;BWSCON Address
stmia    r0,{r1-r13}

;*****
;* Initialize stacks                  *
;*****
ldr      sp, =SVCStack      ;Why?
bl       InitStacks

;*****
;* Setup IRQ handler                  *
;*****
ldr      r0,=HandleIRQ      ;This routine is needed
ldr      r1,=IsrIRQ         ;if there isn't 'subs pc,lr,#4' at 0x18, 0x1c
str      r1,[r0]

;*****
;* Copy and paste RW data/zero initialized data      *
;*****
LDR      r0, =|Image$$RO$$Limit|      ; Get pointer to ROM data
LDR      r1, =|Image$$RW$$Base|      ; and RAM copy
LDR      r3, =|Image$$ZI$$Base|
        ;Zero init base => top of initialised data

CMP      r0, r1              ; Check that they are different
BEQ      %F1

0
CMP      r1, r3              ; Copy init data
LDRCC    r2, [r0], #4        ;--> LDRCC r2, [r0] + ADD r0, r0, #4
STRCC    r2, [r1], #4        ;--> STRCC r2, [r1] + ADD r1, r1, #4
BCC      %B0

1
LDR      r1, =|Image$$ZI$$Limit| ; Top of zero init segment
MOV      r2, #0

2
CMP      r3, r1              ; Zero init
STRCC    r2, [r3], #4
BCC      %B2

[ :LNOT:THUMBCODE
  BL      Main              ;Don't use main() because .....
  B       .
]

[ THUMBCODE                  ;for start-up code for Thumb mode
  orr     lr,pc,#1
  bx      lr
  CODE16
  bl      Main              ;Don't use main() because .....
  b       .
  CODE32
]

```

```

;*****
;*      The function for initializing stack      *
;*****
InitStacks
    ;Don't use DRAM,such as stmfd,ldmfd.....
    ;SVCstack is initialized before
    ;Under toolkit ver 2.50, 'msr cpsr,r1' can be used instead of 'msr cpsr_cxsf,r1'

    mrs        r0,cpsr
    bic        r0,r0,#MODEMASK
    orr        r1,r0,#UNDEFMODE|NOINT
    msr        cpsr_cxsf,r1        ;UndefMode
    ldr        sp,=UndefStack

    orr        r1,r0,#ABORTMODE|NOINT
    msr        cpsr_cxsf,r1        ;AbortMode
    ldr        sp,=AbortStack

    orr        r1,r0,#IRQMODE|NOINT
    msr        cpsr_cxsf,r1        ;IRQMode
    ldr        sp,=IRQStack

    orr        r1,r0,#FIQMODE|NOINT
    msr        cpsr_cxsf,r1        ;FIQMode
    ldr        sp,=FIQStack

    bic        r0,r0,#MODEMASK|NOINT
    orr        r1,r0,#SVCMODE
    msr        cpsr_cxsf,r1        ;SVCMode
    ldr        sp,=SVCStack

    ;USER mode is not initialized.
    mov        pc,lr ;The LR register may be not valid for the mode changes.

;*****
;*      The function for entering power down mode      *
;*****
;void EnterPWDN(int CLKCON);
EnterPWDN
    mov        r2,r0        ;r0=CLKCON
    ldr        r0,=REFRESH
    ldr        r3,[r0]
    mov        r1,r3
    orr        r1,r1,#0x400000    ;self-refresh enable
    str        r1,[r0]

    nop        ;Wait until self-refresh is issued. May not be needed.
    Nop        ;If the other bus master holds the bus, ...
    nop
    nop        ;mov r0, r0
    nop
    nop
    nop

;enter POWERDN mode
    ldr        r0,=CLKCON
    str        r2,[r0]

;wait until enter SL_IDLE,STOP mode and until wake-up
    ldr        r0,=0x10
0    subs     r0,r0,#1
    bne        %B0

```

```
;exit from DRAM/SDRAM self refresh mode.
```

```
    ldr        r0,=REFRESH
    str        r3,[r0]
    mov        pc,lr
```

```
LTORG
```

```
SMRDATA DATA
```

```
;*****
;* Memory configuration has to be optimized for best performance *
;* The following parameter is not optimized. *
;*****
```

```
*** memory access cycle parameter strategy ***
```

```
; 1) Even FP-DRAM, EDO setting has more late fetch point by half-clock
; 2) The memory settings,here, are made the safe parameters even at 66Mhz.
; 3) FP-DRAM Parameters:trCD=3 for tRAC, tcas=2 for pad delay, tcp=2 for bus load.
; 4) DRAM refresh rate is for 40Mhz.
```

```
[ BUSWIDTH=16
  DCD 0x11111110      ;Bank0=OM[1:0], Bank1~Bank7=16bit
| ;BUSWIDTH=32
  DCD 0x22222220      ;Bank0=OM[1:0], Bank1~Bank7=32bit
]
```

```
    DCD
((B0_Tacs<<13)+(B0_Tcos<<11)+(B0_Tacc<<8)+(B0_Tcoh<<6)+(B0_Tah<<4)+(B0_Tacp<<2)+(B0_PMC))
;GCS0
    DCD
((B1_Tacs<<13)+(B1_Tcos<<11)+(B1_Tacc<<8)+(B1_Tcoh<<6)+(B1_Tah<<4)+(B1_Tacp<<2)+(B1_PMC))
;GCS1
    DCD
((B2_Tacs<<13)+(B2_Tcos<<11)+(B2_Tacc<<8)+(B2_Tcoh<<6)+(B2_Tah<<4)+(B2_Tacp<<2)+(B2_PMC))
;GCS2
    DCD
((B3_Tacs<<13)+(B3_Tcos<<11)+(B3_Tacc<<8)+(B3_Tcoh<<6)+(B3_Tah<<4)+(B3_Tacp<<2)+(B3_PMC))
;GCS3
    DCD
((B4_Tacs<<13)+(B4_Tcos<<11)+(B4_Tacc<<8)+(B4_Tcoh<<6)+(B4_Tah<<4)+(B4_Tacp<<2)+(B4_PMC))
;GCS4
    DCD
((B5_Tacs<<13)+(B5_Tcos<<11)+(B5_Tacc<<8)+(B5_Tcoh<<6)+(B5_Tah<<4)+(B5_Tacp<<2)+(B5_PMC))
;GCS5
```

```
[ BDRAMTYPE="DRAM"
  DCD ((B6_MT<<15)+(B6_TrCd<<4)+(B6_Tcas<<3)+(B6_Tcp<<2)+(B6_CAN))      ;GCS6 check
the MT value in parameter.a
  DCD ((B7_MT<<15)+(B7_TrCd<<4)+(B7_Tcas<<3)+(B7_Tcp<<2)+(B7_CAN))      ;GCS7
| ;"SDRAM"
  DCD ((B6_MT<<15)+(B6_TrCd<<2)+(B6_SCAN))          ;GCS6
  DCD ((B7_MT<<15)+(B7_TrCd<<2)+(B7_SCAN))          ;GCS7
]
```

```
    DCD ((REFEN<<23)+(TREFMD<<22)+(Trp<<20)+(Trc<<18)+(Tchr<<16)+REFCNT) ;REFRESH
RFEN=1, TREFMD=0, trp=3clk, trc=5clk, tchr=3clk,count=1019
    DCD 0x10      ;SCLK power mode, BANKSIZE 32M/32M
    DCD 0x20      ;MRSR6 CL=2clk
    DCD 0x20      ;MRSR7
```

```

ALIGN

AREA RamData, DATA, READWRITE

    ^      (_ISR_STARTADDRESS-0x500)

UserStack    #      256      ;c1(c7)ffa00
SVCStack     #      256      ;c1(c7)ffb00
UndefStack   #      256      ;c1(c7)ffc00
AbortStack   #      256      ;c1(c7)ffd00
IRQStack     #      256      ;c1(c7)ffe00
FIQStack     #      0        ;c1(c7)fff00

    ^      _ISR_STARTADDRESS
HandleReset  #      4
HandleUndef  #      4
HandleSWI    #      4
HandlePabort #      4
HandleDabort #      4
HandleReserved #      4
HandleIRQ    #      4
HandleFIQ    #      4

;Don't use the label 'IntVectorTable',
;because armasm.exe cann't recognize this label correctly.
;the value is different with an address you think it may be.
;IntVectorTable
HandleADC    #      4
HandleRTC    #      4
HandleUTXD1  #      4
HandleUTXD0  #      4
HandleSIO    #      4
HandleIIC    #      4
HandleURXD1  #      4
HandleURXD0  #      4
HandleTIMER5 #      4
HandleTIMER4 #      4
HandleTIMER3 #      4
HandleTIMER2 #      4
HandleTIMER1 #      4
HandleTIMER0 #      4
HandleUERR01 #      4
HandleWDT    #      4
HandleBDMA1  #      4
HandleBDMA0  #      4
HandleZDMA1  #      4
HandleZDMA0  #      4
HandleTICK   #      4
HandleEINT4567 #      4
HandleEINT3   #      4
HandleEINT2   #      4
HandleEINT1   #      4
HandleEINT0   #      4      ;0xc1(c7)fff84

END

```

**44BMON : 44blib\_a.s**

```

; *****
; * NAME      : assembly function library      *
; * Version   : 07.JUL.2000                    *
; *****

        AREA |C$$code|, CODE, READONLY

        EXPORT ChangeMemCon

;void ChangeMemCon();
ChangeMemCon
    stmfd    sp!,{r4-r9}        ;Assembler uses the high registers(r4~).

    ldmia    r0,{r1-r9}
    ldr      r0,=0x01c80004      ;BANKCON0 Address
    stmia    r0,{r1-r9}

    ldmfd    sp!,{r4-r9}

    mov      pc,lr

        EXPORT DisableInterrupt
DisableInterrupt
;This function works only if the processor is in previliged mode.

NOINT    EQU        0xc0

    mrs      r0,cpsr
    orr      r0,r0,#NOINT
    msr      cpsr_cxsf,r0

    mov      pc,lr

        EXPORT EnableInterrupt
EnableInterrupt
;This function works only if the processor is in previliged mode.

    mrs      r0,cpsr
    bic      r0,r0,#NOINT
    msr      cpsr_cxsf,r0

    mov      pc,lr

    END

```



**44BMON : def.h**

```
#ifndef __DEF_H__
#define __DEF_H__

#define U32 unsigned int
#define U16 unsigned short
#define S32 int
#define S16 short int
#define U8  unsigned char
#define      S8  char

#endif /*__DEF_H__*/
```

**44BMON : option.h**

```
#ifndef __OPTION_H__
#define __OPTION_H__

// ***** OPTIONS *****
#define MCLK 60000000

#define WRBUFOPT (0x8)    //write_buf_on

#define SYSCFG_0KB (0x0|WRBUFOPT)
#define SYSCFG_4KB (0x2|WRBUFOPT)
#define SYSCFG_8KB (0x6|WRBUFOPT)

#define DRAM      1          //In case DRAM is used
#define SDRAM     2          //In case SDRAM is used
#define BDRAMTYPE SDRAM //used in power.c,44bllib.c

//BUSWIDTH; 16,32
#define BUSWIDTH (16)

#define CACHECFG SYSCFG_8KB

#define _RAM_STARTADDRESS 0xc000000

//#define _ISR_STARTADDRESS 0xc1fff00 //GCS6:16M DRAM
#define _ISR_STARTADDRESS 0xc7fff00 //GCS6:64M DRAM/SDRAM

#endif /*__OPTION_H__*/
```

**44BMON : 44b.h**

```

/*****
 * NAME           : K44b.H
 * Version      : 07.MARCH.2000
 *****/

#ifndef __44B0X_H__
#define __44B0X_H__

#ifdef __cplusplus
extern "C" {
#endif

#include "option.h"

/* System */
#define rSYSCFG          (*(volatile unsigned *)0x1c00000)

/* Cache */
#define rNCACHEB0        (*(volatile unsigned *)0x1c00004)
#define rNCACHEB1        (*(volatile unsigned *)0x1c00008)

/* Bus control */
#define rSBUSCON          (*(volatile unsigned *)0x1c40000)

/* Memory control */
#define rBWSCON           (*(volatile unsigned *)0x1c80000)
#define rBANKCON0         (*(volatile unsigned *)0x1c80004)
#define rBANKCON1         (*(volatile unsigned *)0x1c80008)
#define rBANKCON2         (*(volatile unsigned *)0x1c8000c)
#define rBANKCON3         (*(volatile unsigned *)0x1c80010)
#define rBANKCON4         (*(volatile unsigned *)0x1c80014)
#define rBANKCON5         (*(volatile unsigned *)0x1c80018)
#define rBANKCON6         (*(volatile unsigned *)0x1c8001c)
#define rBANKCON7         (*(volatile unsigned *)0x1c80020)
#define rREFRESH          (*(volatile unsigned *)0x1c80024)
#define rBANKSIZE         (*(volatile unsigned *)0x1c80028)
#define rMRSRB6           (*(volatile unsigned *)0x1c8002c)
#define rMRSRB7           (*(volatile unsigned *)0x1c80030)

/* UART */
#define rULCON0           (*(volatile unsigned *)0x1d00000)
#define rULCON1           (*(volatile unsigned *)0x1d04000)
#define rUCON0            (*(volatile unsigned *)0x1d00004)
#define rUCON1            (*(volatile unsigned *)0x1d04004)
#define rUFCON0           (*(volatile unsigned *)0x1d00008)
#define rUFCON1           (*(volatile unsigned *)0x1d04008)
#define rUMCON0           (*(volatile unsigned *)0x1d0000c)
#define rUMCON1           (*(volatile unsigned *)0x1d0400c)
#define rUTRSTAT0         (*(volatile unsigned *)0x1d00010)
#define rUTRSTAT1         (*(volatile unsigned *)0x1d04010)
#define rUERSTAT0         (*(volatile unsigned *)0x1d00014)
#define rUERSTAT1         (*(volatile unsigned *)0x1d04014)
#define rUFSTAT0          (*(volatile unsigned *)0x1d00018)
#define rUFSTAT1          (*(volatile unsigned *)0x1d04018)
#define rUMSTAT0          (*(volatile unsigned *)0x1d0001c)
#define rUMSTAT1          (*(volatile unsigned *)0x1d0401c)
#define rUBRDIV0          (*(volatile unsigned *)0x1d00028)
#define rUBRDIV1          (*(volatile unsigned *)0x1d04028)

```

```

#ifdef __BIG_ENDIAN
#define rUTXH0      (*(volatile unsigned char *)0x1d00023)
#define rUTXH1      (*(volatile unsigned char *)0x1d04023)
#define rURXH0      (*(volatile unsigned char *)0x1d00027)
#define rURXH1      (*(volatile unsigned char *)0x1d04027)
#define WrUTXH0(ch) (*(volatile unsigned char *)0x1d00023)=(unsigned char)(ch)
#define WrUTXH1(ch) (*(volatile unsigned char *)0x1d04023)=(unsigned char)(ch)
#define RdURXH0()   (*(volatile unsigned char *)0x1d00027)
#define RdURXH1()   (*(volatile unsigned char *)0x1d04027)
#define UTXH0       (0x1d00020+3) //byte_access address by BDMA
#define UTXH1       (0x1d04020+3)
#define URXH0       (0x1d00024+3)
#define URXH1       (0x1d04024+3)

#else //Little Endian
#define rUTXH0      (*(volatile unsigned char *)0x1d00020)
#define rUTXH1      (*(volatile unsigned char *)0x1d04020)
#define rURXH0      (*(volatile unsigned char *)0x1d00024)
#define rURXH1      (*(volatile unsigned char *)0x1d04024)
#define WrUTXH0(ch) (*(volatile unsigned char *)0x1d00020)=(unsigned char)(ch)
#define WrUTXH1(ch) (*(volatile unsigned char *)0x1d04020)=(unsigned char)(ch)
#define RdURXH0()   (*(volatile unsigned char *)0x1d00024)
#define RdURXH1()   (*(volatile unsigned char *)0x1d04024)
#define UTXH0       (0x1d00020) //byte_access address by BDMA
#define UTXH1       (0x1d04020)
#define URXH0       (0x1d00024)
#define URXH1       (0x1d04024)
#endif

/* SIO */
#define rSIOCON      (*(volatile unsigned *)0x1d14000)
#define rSIODAT      (*(volatile unsigned *)0x1d14004)
#define rSBRDR       (*(volatile unsigned *)0x1d14008)
#define rIVTCNT      (*(volatile unsigned *)0x1d1400c)
#define rDCNTZ       (*(volatile unsigned *)0x1d14010)

/* IIS */
#define rIISCON       (*(volatile unsigned *)0x1d18000)
#define rIISMOD       (*(volatile unsigned *)0x1d18004)
#define rIISPSR       (*(volatile unsigned *)0x1d18008)
#define rIISFCN       (*(volatile unsigned *)0x1d1800c)

#ifdef __BIG_ENDIAN
#define rIISFIF       ((volatile unsigned short *)0x1d18012)
#else //Little Endian
#define rIISFIF       ((volatile unsigned short *)0x1d18010)
#endif

/* I/O PORT */
#define rPCONA        (*(volatile unsigned *)0x1d20000)
#define rPDATA        (*(volatile unsigned *)0x1d20004)

#define rPCONB        (*(volatile unsigned *)0x1d20008)
#define rPDATB        (*(volatile unsigned *)0x1d2000c)

#define rPCONC        (*(volatile unsigned *)0x1d20010)
#define rPDATC        (*(volatile unsigned *)0x1d20014)
#define rPUPC         (*(volatile unsigned *)0x1d20018)

#define rPCOND        (*(volatile unsigned *)0x1d2001c)
#define rPDATD        (*(volatile unsigned *)0x1d20020)
#define rPUPD         (*(volatile unsigned *)0x1d20024)

```

```

#define rPCONE      (*(volatile unsigned *)0x1d20028)
#define rPDATE      (*(volatile unsigned *)0x1d2002c)
#define rPUPE       (*(volatile unsigned *)0x1d20030)

#define rPCONF      (*(volatile unsigned *)0x1d20034)
#define rPDATF      (*(volatile unsigned *)0x1d20038)
#define rPUPF       (*(volatile unsigned *)0x1d2003c)

#define rPCONG      (*(volatile unsigned *)0x1d20040)
#define rPDATG      (*(volatile unsigned *)0x1d20044)
#define rPUPG       (*(volatile unsigned *)0x1d20048)

#define rSPUCR      (*(volatile unsigned *)0x1d2004c)
#define rEXTINT      (*(volatile unsigned *)0x1d20050)
#define rEXTINPND    (*(volatile unsigned *)0x1d20054)

/* WATCHDOG */
#define rWTCON      (*(volatile unsigned *)0x1d30000)
#define rWTDAT      (*(volatile unsigned *)0x1d30004)
#define rWTCNT      (*(volatile unsigned *)0x1d30008)

/* ADC */
#define rADCCON      (*(volatile unsigned *)0x1d40000)
#define rADCPSR      (*(volatile unsigned *)0x1d40004)
#define rADCDAT      (*(volatile unsigned *)0x1d40008)

/* Timer */
#define rTCFG0      (*(volatile unsigned *)0x1d50000)
#define rTCFG1      (*(volatile unsigned *)0x1d50004)
#define rTCNT0      (*(volatile unsigned *)0x1d50008)

#define rTCNTB0      (*(volatile unsigned *)0x1d5000c)
#define rTCMPB0      (*(volatile unsigned *)0x1d50010)
#define rTCNT00      (*(volatile unsigned *)0x1d50014)

#define rTCNTB1      (*(volatile unsigned *)0x1d50018)
#define rTCMPB1      (*(volatile unsigned *)0x1d5001c)
#define rTCNT01      (*(volatile unsigned *)0x1d50020)

#define rTCNTB2      (*(volatile unsigned *)0x1d50024)
#define rTCMPB2      (*(volatile unsigned *)0x1d50028)
#define rTCNT02      (*(volatile unsigned *)0x1d5002c)

#define rTCNTB3      (*(volatile unsigned *)0x1d50030)
#define rTCMPB3      (*(volatile unsigned *)0x1d50034)
#define rTCNT03      (*(volatile unsigned *)0x1d50038)

#define rTCNTB4      (*(volatile unsigned *)0x1d5003c)
#define rTCMPB4      (*(volatile unsigned *)0x1d50040)
#define rTCNT04      (*(volatile unsigned *)0x1d50044)

#define rTCNTB5      (*(volatile unsigned *)0x1d50048)
#define rTCNT05      (*(volatile unsigned *)0x1d5004c)

/* IIC */
#define rIICCON      (*(volatile unsigned *)0x1d60000)
#define rIICSTAT      (*(volatile unsigned *)0x1d60004)
#define rIICADD      (*(volatile unsigned *)0x1d60008)
#define rIICDS       (*(volatile unsigned *)0x1d6000c)

```

```

/* RTC */
#ifdef __BIG_ENDIAN
#define rRTCCON      (*(volatile unsigned char *)0x1d70043)
#define rRTCALM      (*(volatile unsigned char *)0x1d70053)
#define rALMSEC      (*(volatile unsigned char *)0x1d70057)
#define rALMMIN      (*(volatile unsigned char *)0x1d7005b)
#define rALMHOUR      (*(volatile unsigned char *)0x1d7005f)
#define rALMDAY      (*(volatile unsigned char *)0x1d70063)
#define rALMMON      (*(volatile unsigned char *)0x1d70067)
#define rALMYEAR      (*(volatile unsigned char *)0x1d7006b)
#define rRTCRST      (*(volatile unsigned char *)0x1d7006f)
#define rBCDSEC      (*(volatile unsigned char *)0x1d70073)
#define rBCDMIN      (*(volatile unsigned char *)0x1d70077)
#define rBCDHOUR      (*(volatile unsigned char *)0x1d7007b)
#define rBCDDAY      (*(volatile unsigned char *)0x1d7007f)
#define rBCDDATE      (*(volatile unsigned char *)0x1d70083)
#define rBCDMON      (*(volatile unsigned char *)0x1d70087)
#define rBCDYEAR      (*(volatile unsigned char *)0x1d7008b)
#define rTICINT      (*(volatile unsigned char *)0x1d7008e)
#else
#define rRTCCON      (*(volatile unsigned char *)0x1d70040)
#define rRTCALM      (*(volatile unsigned char *)0x1d70050)
#define rALMSEC      (*(volatile unsigned char *)0x1d70054)
#define rALMMIN      (*(volatile unsigned char *)0x1d70058)
#define rALMHOUR      (*(volatile unsigned char *)0x1d7005c)
#define rALMDAY      (*(volatile unsigned char *)0x1d70060)
#define rALMMON      (*(volatile unsigned char *)0x1d70064)
#define rALMYEAR      (*(volatile unsigned char *)0x1d70068)
#define rRTCRST      (*(volatile unsigned char *)0x1d7006c)
#define rBCDSEC      (*(volatile unsigned char *)0x1d70070)
#define rBCDMIN      (*(volatile unsigned char *)0x1d70074)
#define rBCDHOUR      (*(volatile unsigned char *)0x1d70078)
#define rBCDDAY      (*(volatile unsigned char *)0x1d7007c)
#define rBCDDATE      (*(volatile unsigned char *)0x1d70080)
#define rBCDMON      (*(volatile unsigned char *)0x1d70084)
#define rBCDYEAR      (*(volatile unsigned char *)0x1d70088)
#define rTICINT      (*(volatile unsigned char *)0x1d7008c)
#endif

/* Clock & Power management */
#define rPLLCON      (*(volatile unsigned *)0x1d80000)
#define rCLKCON      (*(volatile unsigned *)0x1d80004)
#define rCLKSLOW      (*(volatile unsigned *)0x1d80008)
#define rLOCKTIME      (*(volatile unsigned *)0x1d8000c)

/* INTERRUPT */
#define rINTCON      (*(volatile unsigned *)0x1e00000)
#define rINTPND      (*(volatile unsigned *)0x1e00004)
#define rINTMOD      (*(volatile unsigned *)0x1e00008)
#define rINTMSK      (*(volatile unsigned *)0x1e0000c)

#define rI_PSLV      (*(volatile unsigned *)0x1e00010)
#define rI_PMST      (*(volatile unsigned *)0x1e00014)
#define rI_CSLV      (*(volatile unsigned *)0x1e00018)
#define rI_CMST      (*(volatile unsigned *)0x1e0001c)
#define rI_ISPR      (*(volatile unsigned *)0x1e00020)
#define rI_ISPC      (*(volatile unsigned *)0x1e00024)

#define rF_ISPR      (*(volatile unsigned *)0x1e00038)
#define rF_ISPC      (*(volatile unsigned *)0x1e0003c)

```

```

/* LCD */
#define rLCDCON1      (*(volatile unsigned *)0x1f00000)
#define rLCDCON2      (*(volatile unsigned *)0x1f00004)
#define rLCDCON3      (*(volatile unsigned *)0x1f00040)

#define rLCDSADDR1    (*(volatile unsigned *)0x1f00008)
#define rLCDSADDR2    (*(volatile unsigned *)0x1f0000c)
#define rLCDSADDR3    (*(volatile unsigned *)0x1f00010)

#define rREDLUT       (*(volatile unsigned *)0x1f00014)
#define rGREENLUT     (*(volatile unsigned *)0x1f00018)
#define rBLUELUT      (*(volatile unsigned *)0x1f0001c)

#define rDP1_2        (*(volatile unsigned *)0x1f00020)
#define rDP4_7        (*(volatile unsigned *)0x1f00024)
#define rDP3_5        (*(volatile unsigned *)0x1f00028)
#define rDP2_3        (*(volatile unsigned *)0x1f0002c)
#define rDP5_7        (*(volatile unsigned *)0x1f00030)
#define rDP3_4        (*(volatile unsigned *)0x1f00034)
#define rDP4_5        (*(volatile unsigned *)0x1f00038)
#define rDP6_7        (*(volatile unsigned *)0x1f0003c)
#define rDITHMODE     (*(volatile unsigned *)0x1f00044)

/* ZDMA0 */
#define rZDCON0       (*(volatile unsigned *)0x1e80000)
#define rZDISRC0      (*(volatile unsigned *)0x1e80004)
#define rZDIDES0      (*(volatile unsigned *)0x1e80008)
#define rZDICNT0      (*(volatile unsigned *)0x1e8000c)
#define rZDCSRC0      (*(volatile unsigned *)0x1e80010)
#define rZDCDES0      (*(volatile unsigned *)0x1e80014)
#define rZDCCNT0      (*(volatile unsigned *)0x1e80018)

/* ZDMA1 */
#define rZDCON1       (*(volatile unsigned *)0x1e80020)
#define rZDISRC1      (*(volatile unsigned *)0x1e80024)
#define rZDIDES1      (*(volatile unsigned *)0x1e80028)
#define rZDICNT1      (*(volatile unsigned *)0x1e8002c)
#define rZDCSRC1      (*(volatile unsigned *)0x1e80030)
#define rZDCDES1      (*(volatile unsigned *)0x1e80034)
#define rZDCCNT1      (*(volatile unsigned *)0x1e80038)

/* BDMA0 */
#define rBDCON0       (*(volatile unsigned *)0x1f80000)
#define rBDISRC0      (*(volatile unsigned *)0x1f80004)
#define rBDIDES0      (*(volatile unsigned *)0x1f80008)
#define rBDICNT0      (*(volatile unsigned *)0x1f8000c)
#define rBDCSRC0      (*(volatile unsigned *)0x1f80010)
#define rBDCDES0      (*(volatile unsigned *)0x1f80014)
#define rBDCCNT0      (*(volatile unsigned *)0x1f80018)

/* BDMA1 */
#define rBDCON1       (*(volatile unsigned *)0x1f80020)
#define rBDISRC1      (*(volatile unsigned *)0x1f80024)
#define rBDIDES1      (*(volatile unsigned *)0x1f80028)
#define rBDICNT1      (*(volatile unsigned *)0x1f8002c)
#define rBDCSRC1      (*(volatile unsigned *)0x1f80030)
#define rBDCDES1      (*(volatile unsigned *)0x1f80034)
#define rBDCCNT1      (*(volatile unsigned *)0x1f80038)

```

```

/* ISR */
#define pISR_RESET      (*(unsigned *) (_ISR_STARTADDRESS+0x0))
#define pISR_UNDEF      (*(unsigned *) (_ISR_STARTADDRESS+0x4))
#define pISR_SWI        (*(unsigned *) (_ISR_STARTADDRESS+0x8))
#define pISR_PABORT     (*(unsigned *) (_ISR_STARTADDRESS+0xc))
#define pISR_DABORT     (*(unsigned *) (_ISR_STARTADDRESS+0x10))
#define pISR_RESERVED   (*(unsigned *) (_ISR_STARTADDRESS+0x14))
#define pISR_IRQ        (*(unsigned *) (_ISR_STARTADDRESS+0x18))
#define pISR_FIQ        (*(unsigned *) (_ISR_STARTADDRESS+0x1c))

#define pISR_ADC         (*(unsigned *) (_ISR_STARTADDRESS+0x20))
#define pISR_RTC         (*(unsigned *) (_ISR_STARTADDRESS+0x24))
#define pISR_UTXD1       (*(unsigned *) (_ISR_STARTADDRESS+0x28))
#define pISR_UTXD0       (*(unsigned *) (_ISR_STARTADDRESS+0x2c))
#define pISR_SIO         (*(unsigned *) (_ISR_STARTADDRESS+0x30))
#define pISR_IIC         (*(unsigned *) (_ISR_STARTADDRESS+0x34))
#define pISR_URXD1       (*(unsigned *) (_ISR_STARTADDRESS+0x38))
#define pISR_URXD0       (*(unsigned *) (_ISR_STARTADDRESS+0x3c))
#define pISR_TIMER5      (*(unsigned *) (_ISR_STARTADDRESS+0x40))
#define pISR_TIMER4      (*(unsigned *) (_ISR_STARTADDRESS+0x44))
#define pISR_TIMER3      (*(unsigned *) (_ISR_STARTADDRESS+0x48))
#define pISR_TIMER2      (*(unsigned *) (_ISR_STARTADDRESS+0x4c))
#define pISR_TIMER1      (*(unsigned *) (_ISR_STARTADDRESS+0x50))
#define pISR_TIMER0      (*(unsigned *) (_ISR_STARTADDRESS+0x54))
#define pISR_UERR01      (*(unsigned *) (_ISR_STARTADDRESS+0x58))
#define pISR_WDT         (*(unsigned *) (_ISR_STARTADDRESS+0x5c))
#define pISR_BDMA1       (*(unsigned *) (_ISR_STARTADDRESS+0x60))
#define pISR_BDMA0       (*(unsigned *) (_ISR_STARTADDRESS+0x64))
#define pISR_ZDMA1       (*(unsigned *) (_ISR_STARTADDRESS+0x68))
#define pISR_ZDMA0       (*(unsigned *) (_ISR_STARTADDRESS+0x6c))
#define pISR_TICK        (*(unsigned *) (_ISR_STARTADDRESS+0x70))
#define pISR_EINT4567     (*(unsigned *) (_ISR_STARTADDRESS+0x74))
#define pISR_EINT3       (*(unsigned *) (_ISR_STARTADDRESS+0x78))
#define pISR_EINT2       (*(unsigned *) (_ISR_STARTADDRESS+0x7c))
#define pISR_EINT1       (*(unsigned *) (_ISR_STARTADDRESS+0x80))
#define pISR_EINT0       (*(unsigned *) (_ISR_STARTADDRESS+0x84))

/* PENDING BIT */
//CAUTION:You must clear the pending bit as general special register.
//          it's different way with KS32C6x00

#define BIT_ADC          (0x1)
#define BIT_RTC          (0x1<<1)
#define BIT_UTXD1        (0x1<<2)
#define BIT_UTXD0        (0x1<<3)
#define BIT_SIO          (0x1<<4)
#define BIT_IIC          (0x1<<5)
#define BIT_URXD1        (0x1<<6)
#define BIT_URXD0        (0x1<<7)
#define BIT_TIMER5       (0x1<<8)
#define BIT_TIMER4       (0x1<<9)
#define BIT_TIMER3       (0x1<<10)
#define BIT_TIMER2       (0x1<<11)
#define BIT_TIMER1       (0x1<<12)
#define BIT_TIMER0       (0x1<<13)
#define BIT_UERR01       (0x1<<14)
#define BIT_WDT          (0x1<<15)
#define BIT_BDMA1        (0x1<<16)
#define BIT_BDMA0        (0x1<<17)
#define BIT_ZDMA1        (0x1<<18)
#define BIT_ZDMA0        (0x1<<19)
#define BIT_TICK         (0x1<<20)

```

```
#define BIT_EINT4567      (0x1<<21)
#define BIT_EINT3         (0x1<<22)
#define BIT_EINT2         (0x1<<23)
#define BIT_EINT1         (0x1<<24)
#define BIT_EINT0         (0x1<<25)
#define BIT_GLOBAL        (0x1<<26)

#ifdef __cplusplus
}
#endif
#endif /* __44B0X_H__ */
```



**44BMON : 44blib.h**

```

/*****
* NAME      : 44BLIB.H
* Version   : 17.Apr.00
*****/

#ifndef __44blib_h__
#define __44blib_h__

#ifdef __cplusplus
extern "C" {
#endif

#define DebugOut Uart_Printf

#define min(x1,x2) ((x1<x2)? x1:x2)
#define max(x1,x2) ((x1>x2)? x1:x2)

#define ONESEC0 (62500)    //16us resolution, max 1.04 sec
#define ONESEC1 (31250)    //32us resolution, max 2.09 sec
#define ONESEC2 (15625)    //64us resolution, max 4.19 sec
#define ONESEC3 (7812)     //128us resolution, max 8.38 sec
#define ONESEC4 (MCLK/128/(0xff+1)) //60Mhz, 128*4us resolution, max 32.53 sec

#define NULL 0

#define EnterPWRDN(clkcon) ((void (*)(int))0xe0)(clkcon)

/*44blib.c*/
void Delay(int time); //Watchdog Timer is used.

void *malloc(unsigned nbyte);
void free(void *pt);

void Port_Init(void);
void Cache_Flush(void);
void Uart_Select(int ch);
void Uart_TxEmpty(int ch);
void Uart_Init(int mclk,int baud);
char Uart_Getch(void);
char Uart_GetKey(void);
int  Uart_GetIntNum(void);
void Uart_SendByte(int data);
void Uart_Printf(char *fmt,...);
void Uart_SendString(char *pt);

void Timer_Start(int divider); //Watchdog Timer is used.
int  Timer_Stop(void);         //Watchdog Timer is used.

void Led_Display(int data);

void ChangePllValue(int m,int p,int s);

void ChangeMemCon(unsigned *pMemCfg);
void EnableInterrupt(void);
void DisableInterrupt(void);

#ifdef __cplusplus
}
#endif

#endif /*__44blib_h__*/

```

**44BMON : lcd.h**

```

#ifndef __LCD_H__
#define __LCD_H__
#define CLCD_240_320      (1)
#define MLCD_320_240      (2)
#define LCD_TYPE          MLCD_320_240

#define MODE_MONO          (1)
#define MODE_G4            (4)
#define MODE_G16           (16)
#define MODE_COLOR         (256)

#if (LCD_TYPE==MLCD_320_240)
#define SCR_XSIZE          (640)
#define SCR_YSIZE          (480)

#define LCD_XSIZE          (320)
#define LCD_YSIZE          (240)

#elif (LCD_TYPE==CLCD_240_320)
#define SCR_XSIZE          (640)
#define SCR_YSIZE          (480)

#define LCD_XSIZE          (240)
#define LCD_YSIZE          (320)
#endif

#define M5D(n) ((n) & 0x1ffffff)

#define ARRAY_SIZE_MONO    (SCR_XSIZE/8*SCR_YSIZE)
#define ARRAY_SIZE_G4      (SCR_XSIZE/4*SCR_YSIZE)
#define ARRAY_SIZE_G16     (SCR_XSIZE/2*SCR_YSIZE)
#define ARRAY_SIZE_COLOR   (SCR_XSIZE/1*SCR_YSIZE)

#define HOZVAL              (LCD_XSIZE/4-1)
#define HOZVAL_COLOR        (LCD_XSIZE*3/8-1)
#define LINEVAL             (LCD_YSIZE-1)
#define MVAL                (13)
// #define CLKVAL_MONO      (19) //60Mhz, CLKVAL=19 ->78.6Hz
// #define CLKVAL_G4        (19) //60Mhz, CLKVAL=19 ->78.6Hz
// #define CLKVAL_G16       (19) //60Mhz, CLKVAL=19 ->78.6Hz
// #define CLKVAL_COLOR     (19) //60Mhz, CLKVAL=19 ->78.6Hz

#define CLKVAL_MONO         (13) //60Mhz, CLKVAL=19 ->78.6Hz
#define CLKVAL_G4           (13) //60Mhz, CLKVAL=19 ->78.6Hz
#define CLKVAL_G16          (13) //60Mhz, CLKVAL=19 ->78.6Hz
#define CLKVAL_COLOR        (10) //60Mhz, CLKVAL=19 ->78.6Hz

unsigned int (*frameBuffer1)[SCR_XSIZE/32];
unsigned int (*frameBuffer4)[SCR_XSIZE/16];
unsigned int (*frameBuffer16)[SCR_XSIZE/8];
unsigned int (*frameBuffer256)[SCR_XSIZE/4];

#define MVAL_USED 0

#endif /*__LCD_H__*/

```

**44BMON : 44blib.c**

```

/*****
 * NAME      : 44BLIB.C
 * Version   : 17.APR.00
 *****/

// Revision History
// Delay()

#include "..\inc\44b.h"
#include "..\inc\44blib.h"
#include "..\inc\def.h"
#include "..\inc\option.h"

#include <stdarg.h>
#include <string.h>
#include <stdlib.h>
#include <stdio.h>
#include <ctype.h>

#define STACKSIZE    0xa00 //SVC stack size(do not use user stack)
#define HEAPEND      (_ISR_STARTADDRESS-STACKSIZE-0x500) // = 0xc7ff000
                    //SVC Stack Area:0xc(e)7ff000-0xc(e)7ffa00

extern char Image$$RW$$Limit[];

void *mallocPt=Image$$RW$$Limit;

/***** SYSTEM *****/
static int delayLoopCount=400;

void Delay(int time)
// time=0: adjust the Delay function by WatchDog timer.
// time>0: the number of loop time
// 100us resolution.
{
    int i,adjust=0;
    if(time==0)
    {
        time=200;
        adjust=1;
        delayLoopCount=400;
        rWTCON=((MCLK/1000000-1)<<8)|(2<<3); // 1M/64,Watch-dog,nRESET,interrupt disable
        rWTDAT=0xffff;
        rWTCNT=0xffff;
        rWTCON=((MCLK/1000000-1)<<8)|(2<<3)|(1<<5); // 1M/64,Watch-dog
enable,nRESET,interrupt disable
    }
    for(;time>0;time--)
        for(i=0;i<delayLoopCount;i++);
    if(adjust==1)
    {
        rWTCON=((MCLK/1000000-1)<<8)|(2<<3);
        i=0xffff-rWTCNT; // lcount/16us????????
        delayLoopCount=8000000/(i*64); //400*100/(i*64/200)
    }
}

```

```

/***** PORTS *****/

void Port_Init(void)
{
    //SMDK41100 B/D Status
    //LED D5 D6
    // PB9 PB10
    //S/W S4 S5
    // PG5 PG4

    //CAUTION:Follow the configuration order for setting the ports.
    // 1) setting value
    // 2) setting control register
    // 3) configure pull-up resistor.

    //16bit data bus configuration

    //PORT A GROUP
    //ADDR24 ADDR23 ADDR22 ADDR21 ADDR20 ADDR19 ADDR18 ADDR17 ADDR16 ADDR0
    // 1, 1, 1, 1, 1, 1, 1, 1, 1, 1
    rPCONA=0x3ff;

    //PORT B GROUP
    //OUT OUT nGCS2 nGCS1 nWBE3 nWBE2 nSRAS nSCAS SCLK SCKE
    // 0, 0, 1, 1, 1, 1, 1, 1, 1, 1
    rPCONB=0x1ff;

    //PORT C GROUP
    #if (BUSWIDTH==32)
    //D31 D30 D29 D28 D27 D26 D25 D24 D23 D22 D21 D20 D19 D18 D17 D16
    // 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10, 10
    rPCONC=0xaaaaaaaa;
    rPUPC=0xffff;
    #else //BUSWIDTH=16
    //PORT C GROUP
    //All input
    // 0x0
    rPDATC=0x0;
    rPCONC=0x10000000;
    rPUPC=0xc000; //should be enabled
    #endif

    //PORT D GROUP
    //VFRAME VM VLINE VCLK VD3 VD2 VD1 VD0
    // 10,10, 10, 10, 10, 10, 10, 10
    rPCOND=0xaaaa;
    rPUPD=0xff;

    //PORT E GROUP
    //PE0:FOUT, PE1:TxD0, PE2:RxD0, others input
    // 11 10, 10, 0
    // rPCONE=0x28;
    rPCONE=0x2b;
    rPUPE=0x106;

    //PORT F GROUP
    //All input
    // 0x0
    rPDATF=0x0;
    rPCONF=0x2400;
    rPUPF=0x1e3;

```

```

//PORT G GROUP
//All input
// 0x0

rPCONG=0xf00;
rPUPG=0x30;

rSPUCR=0x7; //pull-up disable

rEXTINT=0x22222222; //All EINT[7:0] will be falling edge triggered.
}

/***** UART *****/

static int whichUart=0;

void Uart_Init(int mclk,int baud)
{
    int i;
    if(mclk==0)
        mclk=MCLK;
    rUFCON0=0x0; //FIFO disable
    rUFCON1=0x0;
    rUMCON0=0x0;
    rUMCON1=0x0;
//UART0
    rULCON0=0x3; //Normal,No parity,1 stop,8 bit
//    rULCON0=0x7; //Normal,No parity,2 stop,8 bit
    rUCON0=0x245; //rx=edge,tx=level,disable timeout int.,enable rx error
int.,normal,interrupt or polling
    rUBRDIV0=( (int)(mclk/16./baud + 0.5) -1 );
//UART1
//    rULCON1=0x7; //Normal,No parity,2 stop,8 bit
    rULCON1=0x3;
    rUCON1=0x245;
    rUBRDIV1=( (int)(mclk/16./baud + 0.5) -1 );

    for(i=0;i<100;i++);
}

void Uart_Select(int ch)
{
    whichUart=ch;
}

void Uart_TxEmpty(int ch)
{
    if(ch==0)
        while(!(rUTRSTAT0 & 0x4)); //wait until tx shifter is empty.
    else
        while(!(rUTRSTAT1 & 0x4)); //wait until tx shifter is empty.
}

char Uart_Getch(void)
{
    if(whichUart==0)
    {
        while(!(rUTRSTAT0 & 0x1)); //Receive data read
        return RdURXH0();
    }
}

```

```

else
{
    while(!(rUTRSTAT1 & 0x1)); //Receive data ready
    return rURXH1;
}
}

```

```

char Uart_GetKey(void)
{
    if(whichUart==0)
    {
        if(rUTRSTAT0 & 0x1) //Receive data ready
            return RdURXH0();
        else
            return 0;
    }
    else
    {
        if(rUTRSTAT1 & 0x1) //Receive data ready
            return rURXH1;
        else
            return 0;
    }
}

```

```

void Uart_GetString(char *string)
{
    char *string2=string;
    char c;
    while((c=Uart_Getch())!='\r')
    {
        if(c=='\b')
        {
            if((int)string2 < (int)string )
            {
                Uart_Printf("\b \b");
                string--;
            }
        }
        else
        {
            *string++=c;
            Uart_SendByte(c);
        }
    }
    *string='\0';
    Uart_SendByte('\n');
}

```

```

int Uart_GetIntNum(void)
{
    char str[30];
    char *string=str;
    int base=10;
    int minus=0;
    int lastIndex;
    int result=0;
    int i;

    Uart_GetString(string);

```

```

    if(string[0]=='-')
    {
        minus=1;
        string++;
    }

    if(string[0]=='0' && (string[1]=='x' || string[1]=='X'))
    {
        base=16;
        string+=2;
    }

    lastIndex=strlen(string)-1;
    if( string[lastIndex]=='h' || string[lastIndex]=='H' )
    {
        base=16;
        string[lastIndex]=0;
        lastIndex--;
    }

    if(base==10)
    {
        result=atoi(string);
        result=minus ? (-1*result):result;
    }
    else
    {
        for(i=0;i<=lastIndex;i++)
        {
            if(isalpha(string[i]))
            {
                if(isupper(string[i]))
                    result=(result<<4)+string[i]-'A'+10;
                else
                    result=(result<<4)+string[i]-'a'+10;
            }
            else
            {
                result=(result<<4)+string[i]-'0';
            }
        }
        result=minus ? (-1*result):result;
    }
    return result;
}

void Uart_SendByte(int data)
{
    if(whichUart==0)
    {
        if(data=='\n')
        {
            while(!(rUTRSTAT0 & 0x2));
            Delay(10); //because the slow response of hyper_terminal
            WrUTXH0('\r');
        }
        while(!(rUTRSTAT0 & 0x2)); //Wait until THR is empty.
        Delay(10);
        WrUTXH0(data);
    }
}

```

```

else
{
    if(data=='\n')
    {
        while(!(rUTRSTAT1 & 0x2));
        Delay(10); //because the slow response of hyper_terminal
        rUTXH1='\r';
    }
    while(!(rUTRSTAT1 & 0x2)); //Wait until THR is empty.
    Delay(10);
    rUTXH1=data;
}
}

void Uart_SendString(char *pt)
{
    while(*pt)
        Uart_SendByte(*pt++);
}

//if you don't use vsprintf(), the code size is reduced very much.
void Uart_Printf(char *fmt,...)
{
    va_list ap;
    char string[256];

    va_start(ap,fmt);
    vsprintf(string,fmt,ap);
    Uart_SendString(string);
    va_end(ap);
}

/***** S3C44B0X EV. BOARD LED *****/

void Led_Display(int data)
{
    rPDATB=(rPDATB & 0x1ff) | ((data & 0x3)<<9);
}

/***** Timer *****/

void Timer_Start(int divider) //0:16us,1:32us 2:64us 3:128us
{
    rWTCN=(MCLK/1000000-1)<<8)|(divider<<3);
    rWTDAT=0xffff;
    rWTCNT=0xffff;

    // 1/16/(65+1),nRESET & interrupt disable
    rWTCN=(MCLK/1000000-1)<<8)|(divider<<3)|(1<<5);
}

int Timer_Stop(void)
{
    // int i;
    rWTCN=(MCLK/1000000-1)<<8);
    return (0xffff-rWTCNT);
}

```



```

/***** PLL *****/
void ChangePllValue(int mdiv,int pdiv,int sdiv)
{
    rPLLCON=(mdiv<<12)|(pdiv<<4)|sdiv;
}

/***** General Library *****/

void * malloc(unsigned nbyte)
/*Very simple; Use malloc() & free() like Stack*/
//void *mallocPt=Image$$RW$$Limit;
{
    void *returnPt=mallocPt;

    mallocPt= (int *)mallocPt+nbyte/4+((nbyte%4)>0); //to align 4byte

    if( (int)mallocPt > HEAPEND )
    {
        mallocPt=returnPt;
        return NULL;
    }
    return returnPt;
}

void free(void *pt)
{
    mallocPt=pt;
}

void Cache_Flush(void)
{
    int i,saveSyscfg;

    saveSyscfg=rSYSCFG;

    rSYSCFG=SYSCFG_0KB;
    for(i=0x10004000;i<0x10004800;i+=16)
    {
        *((int *)i)=0x0;
    }
    rSYSCFG=saveSyscfg;
}

```

**44BMON : 44bmon.c**

```

#include <string.h>
#include "..\inc\option.h"
#include "..\inc\def.h"
#include "..\inc\44b.h"
#include "..\inc\44blib.h"
#include "..\inc\lcd.h"

//99.5.14 : Lcd_Init(), non-cacheable region is added.
//99.5.26 : link address: 0x1800000->0x0
//99.6.30(Ver 0.2): supports 64M DRAM(GCS6)
//99.8.10 : filesize counter: 6digit->7digit
//99.8.27 : support Power Down mode
//00.4.20 : modify for S3C44BOX

#define DOWNLOAD_ADDRESS _RAM_STARTADDRESS

void (*restart)(void)=(void (*)(void))0x0;
void (*run)(void)=(void (*)(void))DOWNLOAD_ADDRESS;

unsigned int (*frameBuffer)[10];

volatile unsigned char *downPt;

void Isr_Init(void);
void Lcd_Init(void);
void __irq Uart0_RxInt(void);
void __irq Uerror(void);
volatile unsigned int err=0;

void Main(void)
{
    int i,j;
    int memError=0;
    unsigned int fileSize;
    unsigned short int checkSum=0,dnCS;
    //    volatile unsigned *pt;

    rSYSCFG=CACHECFG;

    downPt=(unsigned char *)DOWNLOAD_ADDRESS;

    pISR_SWI=(_ISR_STARTADDRESS+0xf0);    //for pSOS

    Port_Init();
    Led_Display(0x0);
    Isr_Init();
    Uart_Init(0,115200);

    Lcd_Init();    //to avoid LCD damage.

    Delay(0);
    Uart_Select(0); //Select UART0

    Uart_Printf("\n\n44bMON Ver 0.01 for S3C44B0X May,2000\n");
    Uart_Printf("COM:115.2kbps,8Bit,NP,UART0 <n+6>(4)+(n)+CS(2)\n");
    Uart_Printf("DOWNADDR:%x ISR_ADDR:%x SYSCFG:%x\n",
    DOWNLOAD_ADDRESS,_ISR_STARTADDRESS,rSYSCFG );
    Uart_Printf("E-mail:kwark@sec.samsung.com\n\n");

```

```

    Uart_Printf("Memory Test(%xh-%xh):WR",
        _RAM_STARTADDRESS,(_ISR_STARTADDRESS&0xfff0000));
    for(i=_RAM_STARTADDRESS;i<(_ISR_STARTADDRESS&0xfff0000);i+=4)
    {
        *((volatile unsigned *)i)=i;
    }
    Uart_Printf("\b\bRD");
    for(i=_RAM_STARTADDRESS;i<(_ISR_STARTADDRESS&0xfff0000);i+=4)
    {
        j=*((volatile unsigned *)i);
        if(j!=i)memError=1;
    }
    if(memError==0)
        Uart_Printf("\b\bO.K.\n");
    else
        Uart_Printf("\b\bFAIL\n");

while(((unsigned int)downPt-DOWNLOAD_ADDRESS )<4)
{
    Led_Display(0xf);
    Delay(1000);
    Led_Display(0x0);
    Delay(1000);
}

fileSize=*((unsigned char *)(DOWNLOAD_ADDRESS+0))+
    (*((unsigned char *)(DOWNLOAD_ADDRESS+1))<<8)+
    (*((unsigned char *)(DOWNLOAD_ADDRESS+2))<<16)+
    (*((unsigned char *)(DOWNLOAD_ADDRESS+3))<<24);

Uart_Printf("\nNow, Downloading... [FILESIZE:%7d(      0)",fileSize);

while((((int)downPt-DOWNLOAD_ADDRESS)<fileSize)
{
    Uart_Printf("\b\b\b\b\b\b\b\b\b%7d)",(int)downPt-DOWNLOAD_ADDRESS);
//    Uart_Printf("error=%d\n",err);
}
Uart_Printf("\b\b\b\b\b\b\b\b\b%7d)\n",(int)downPt-DOWNLOAD_ADDRESS);

for(i=4;i<(fileSize-2);i++)
{
    checkSum+=*((unsigned char *)(i+DOWNLOAD_ADDRESS));
}
dnCS=*((unsigned char *)(DOWNLOAD_ADDRESS+fileSize-2))+
    (*( (unsigned char *)(DOWNLOAD_ADDRESS+fileSize-1) )<<8);

if(checkSum!=dnCS)
{
    Uart_Printf("Checksum Error!!! MEM:%x DN:%x\n",checkSum,dnCS);
    restart();
}

Uart_Printf("\nDownload O.K.\n");

for(i=4;i<(fileSize-2);i++) //Move the program from 0x1000004~ to 0x1000000~
{
    *((unsigned char *)(i+DOWNLOAD_ADDRESS-4))
        =*((unsigned char *)(i+DOWNLOAD_ADDRESS));
}

run();
}

```

```

void Isr_Init(void)
{
    rINTCON=0x5;           //Non-vectorred,IRQ enable,FIQ disable
    rINTMOD=0x0;           //All=IRQ mode
    rINTMSK=~( BIT_URXD0 | BIT_GLOBAL); //Default value=0x7ffffff
//    rINTMSK=~( BIT_UERR01 | BIT_URXD0 | BIT_GLOBAL); //Default value=0x7ffffff

    /*pISR_FIQ,pISR_IRQ must be initialized*/
    pISR_URXD0=(unsigned)Uart0_RxInt;
//    pISR_UERR01=(unsigned)Uerror;
}

void __irq Uart0_RxInt(void)
{
    rI_ISPC=BIT_URXD0 ;    //clear pending bits,Default value=0x0000000
    //rI_ISPC;             //is needed only when cache=on & wrbuf=on & BSFRD=0
    *downPt++=RdURXH0();
}

void __irq Uerror(void)
{
    rI_ISPC=BIT_UERR01 ;   //clear pending bits,Default value=0x0000000
    err++;
//    Uart_Printf("UERSTAT=%x\n", rUERSTAT0);
}

void Display_4Gray160x240(void)
{
    int i,j;

    for(j=0;j<100;j++)
        for(i=2;i<10;i++)
        {
            frameBuffer4[j][i]=0x55aa55aa;
        }

    for(j=0;j<100;j++)
    {
        frameBuffer4[j][9]=0x5555ffff;
    }

    for(j=100;j<240;j++)
        for(i=2;i<10;i++)
            frameBuffer4[j][i]=0x0;

    for(i=2;i<10;i++)
        frameBuffer4[100][i]=0xffffffff;
    for(i=2;i<10;i++)
        frameBuffer4[0][i]=0xffffffff;
}

```

```

void Lcd_Init(void)
{
    //320x240 1bit/1pixel LCD
    int i,j;

    if((U32)frameBuffer1==0)
    {
        frameBuffer1=(unsigned int (*)(SCR_XSIZE/32))malloc(ARRAY_SIZE_MONO);
    }

    rLCDCON1=(0)|(1<<5)|(MVAL_USED<<7)|(0x3<<8)|(0x3<<10)|(CLKVAL_MONO<<12);
    // disable, 4B_SINGL_SCAN, WDLY=8clk, WLH=8clk,
    rLCDCON2=(LINEVAL)|(HOZVAL<<10)|(10<<21);
    //LINEBLANK=10 (without any calculation)
    rLCDSADDR1= (0x0<<27) | ( ((U32)frameBuffer1>>22)<<21 ) | M5D((U32)frameBuffer1>>1);
    // monochrome, LCDBANK, LCDBASEU
    rLCDSADDR2= M5D( (((U32)frameBuffer1+(SCR_XSIZE*LCD_YSIZE/8))>>1) ) | (MVAL<<21);
    rLCDSADDR3= (LCD_XSIZE/16) | ( ((SCR_XSIZE-LCD_XSIZE)/16)<<9 );

    rLCDCON1=(1)|(1<<5)|(MVAL_USED<<7)|(0x3<<8)|(0x3<<10)|(CLKVAL_MONO<<12);
    // enable, 4B_SINGL_SCAN, WDLY=8clk, WLH=8clk,

    for(j=0;j<240;j++)
        for(i=0;i<10;i++)
        {
            if((j%16)<8)
                frameBuffer1[j][i]=0xff00ff00;
            else
                frameBuffer1[j][i]=0x00ff00ff;
        }
}

```

## 44BTEST FILE DESCRIPTIONS

Filename	File Descriptons	Page
makefile	Makefile for 44BTEST	3-48
option.a	Configuration file for 44binit.s	3-52
memcfg.a*	Configuration file for memory bank control register	-
44binit.s*	Start-up code	-
44blib_a.s*	Assembly library functions	-
def.h*	Header file for data type definitions	-
option.h	Header file for board configuration	3-53
44b.h*	Header file for S32C44B0X special register definitions	-
44btest.c	Test program main file.	3-54
44blib.h*	Header file for 44blib.c	-
44blib.c*	C library functions	-
adc.c	ADC unit test program	3-57
am29f800.c	AM29LV800BB(NOR flash) program sub-routines	3-60
cache.c	Cache memory test program	3-64
clcd.c	LCD controller test program	3-69
dma.c	DMA unit test program	3-72
eint.c	External interrupt test program	3-76
etc.c	Etc program	3-79
extdma.c	External DMA test program	3-80
flash.c	NOR flash writing program	3-88
glib.c	C library for LCD test program	3-90
idle.c	Power down mode test program	3-95
iic.c	IIC interface test program	3-102
iis.c	IIS interface test program	3-106
lcd.c	Lcd controller test program	3-111
lcdlib.c	C library for LCD test program	3-120
nwait.c	nWAIT test program	3-126
power.c	Power down mode test program	3-129
rtc.c	RTC unit test program	3-132

Filename	File Descriptons	Page
sio.c	SIO unit test program	3-135
stop.c	Power down mode test program	3-139
timer.c	Timer unit test program	3-142
uart.c	UART0,1 test program	3-146

**NOTE:** \* marked file is exactly same with that used in 44BMON.

#### NOTE

The program source date in this application notes is Sep.29.2000. Please download the latest test code source from our website. (<http://www.samsungsemi.com>)

**44BTEST : Makefile**

```
#### File Definition ####
PRJ = 44btest
INIT= 44binit
AM1 = 44blib_a
CM1 = 44blib
CM2 = cache
CM3 = lcd
CM4 = adc
CM5 = dma
CM6 = timer
CM7 = etc
CM8 = uart
CM9 = power
CM10= sio
CM11= rtc
CM12= iic
CM13= clcd
CM14= eint
CM15= nwait
CM16= lcdlib
CM17= idle
CM18= flash
CM19= am29f800
CM20= iis
CM21= stop
CM22= extdma
CM23= glib

#### Destination path Definition ####
SRC=.
INC=..\inc
OBJ=..\obj
ERR=..\err
BIN=..\bin

#### ARM tool Definition ####
ARMLINK= armlink
ARMASM = armasm
ARMCC = armcc

#### Option Definition ####
LFLAGS = -ro-base 0xc000000 -rw-base 0x0c040000 -elf
#LFLAGS = -ro-base 0xc000000 -rw-base 0x0c040000 -elf -map -xref -list $(BIN)\list.lst
#LFLAGS = -ro-base 0x10000 -rw-base 0x0c040000 -elf
AFLAGS = -li -apcs 3/32bit/noswst/nofp -cpu ARM7TM
CFLAGS = -c -g+ -fc -apcs 3/32bit/noswst/nofp -li -processor ARM7TM -arch 4T

#If you doesn't debug,use following CFLAGS for more faster operation.
#CFLAGS = -c -g- -fc -apcs 3/32bit/noswst/nofp -li -processor ARM7TM -arch 4T

#### Object combine Definition ####
OBJS = $(OBJ)\$(PRJ).o $(OBJ)\$(INIT).o $(OBJ)\$(AM1).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
```



```

all: $(BIN)\$(PRJ).elf

$(BIN)\$(PRJ).elf: $(OBJS)
    del $(BIN)\$(PRJ).bin
    del $(BIN)\$(PRJ).elf
    $(ARMLINK) $(LFLAGS) -first $(OBJ)\$(INIT).o (Init) -o $(BIN)\$(PRJ).elf $(OBJS)
    fromelf -nodebug -nozeropad $(BIN)\$(PRJ).elf -bin $(BIN)\$(PRJ).bin
#    fromelf $(BIN)\$(PRJ).elf -text/s $(BIN)\syms.sym
#    fromelf $(BIN)\$(PRJ).elf -text/c $(BIN)\symc.sym

$(OBJ)\$(PRJ).o : $(SRC)\$(PRJ).c $(INC)\44b.h $(INC)\44blib.h $(INC)\option.h makefile
    del $(OBJ)\$(PRJ).o
    del $(ERR)\$(PRJ).err
    $(ARMCC) $(CFLAGS) $(SRC)\$(PRJ).c -o $(OBJ)\$(PRJ).o -Errors $(ERR)\$(PRJ).err

$(OBJ)\$(INIT).o: $(SRC)\$(INIT).s $(INC)\option.a $(INC)\memcfg.a makefile
    del $(OBJ)\$(INIT).o
    del $(ERR)\$(INIT).err
    $(ARMASM) $(AFLAGS) $(SRC)\$(INIT).s -o $(OBJ)\$(INIT).o -Errors $(ERR)\$(INIT).err

$(OBJ)\$(AM1).o: $(SRC)\$(AM1).s makefile
    del $(OBJ)\$(AM1).o
    del $(ERR)\$(AM1).err
    $(ARMASM) $(AFLAGS) $(SRC)\$(AM1).s -o $(OBJ)\$(AM1).o -Errors $(ERR)\$(AM1).err

$(OBJ)\$(CM1).o: $(SRC)\$(CM1).c $(INC)\$(CM1).h $(INC)\44b.h $(INC)\44blib.h\
$(INC)\option.h makefile
    del $(OBJ)\$(CM1).o
    del $(ERR)\$(CM1).err
    $(ARMCC) $(CFLAGS) $(SRC)\$(CM1).c -o $(OBJ)\$(CM1).o -Errors $(ERR)\$(CM1).err

$(OBJ)\$(CM2).o: $(SRC)\$(CM2).c $(INC)\$(CM2).h $(INC)\44b.h $(INC)\44blib.h\
$(INC)\option.h makefile
    del $(OBJ)\$(CM2).o
    del $(ERR)\$(CM2).err
    $(ARMCC) $(CFLAGS) $(SRC)\$(CM2).c -o $(OBJ)\$(CM2).o -Errors $(ERR)\$(CM2).err

$(OBJ)\$(CM3).o: $(SRC)\$(CM3).c $(INC)\$(CM3).h $(INC)\44b.h $(INC)\44blib.h\
$(INC)\option.h makefile
    del $(OBJ)\$(CM3).o
    del $(ERR)\$(CM3).err
    $(ARMCC) $(CFLAGS) $(SRC)\$(CM3).c -o $(OBJ)\$(CM3).o -Errors $(ERR)\$(CM3).err

$(OBJ)\$(CM4).o: $(SRC)\$(CM4).c $(INC)\$(CM4).h $(INC)\44b.h $(INC)\44blib.h\
$(INC)\option.h makefile
    del $(OBJ)\$(CM4).o
    del $(ERR)\$(CM4).err
    $(ARMCC) $(CFLAGS) $(SRC)\$(CM4).c -o $(OBJ)\$(CM4).o -Errors $(ERR)\$(CM4).err

$(OBJ)\$(CM5).o: $(SRC)\$(CM5).c $(INC)\$(CM5).h $(INC)\44b.h $(INC)\44blib.h\
$(INC)\option.h makefile
    del $(OBJ)\$(CM5).o
    del $(ERR)\$(CM5).err
    $(ARMCC) $(CFLAGS) $(SRC)\$(CM5).c -o $(OBJ)\$(CM5).o -Errors $(ERR)\$(CM5).err

$(OBJ)\$(CM6).o: $(SRC)\$(CM6).c $(INC)\$(CM6).h $(INC)\44b.h $(INC)\44blib.h\
$(INC)\option.h makefile
    del $(OBJ)\$(CM6).o
    del $(ERR)\$(CM6).err
    $(ARMCC) $(CFLAGS) $(SRC)\$(CM6).c -o $(OBJ)\$(CM6).o -Errors $(ERR)\$(CM6).err

```

```
$(OBJ)\$(CM7).o: $(SRC)\$(CM7).c $(INC)\$(CM7).h $(INC)\44b.h $(INC)\44bplib.h\
$(INC)\option.h makefile
    del $(OBJ)\$(CM7).o
    del $(ERR)\$(CM7).err
    $(ARMCC) $(CFLAGS) $(SRC)\$(CM7).c -o $(OBJ)\$(CM7).o -Errors $(ERR)\$(CM7).err

$(OBJ)\$(CM8).o: $(SRC)\$(CM8).c $(INC)\$(CM8).h $(INC)\44b.h $(INC)\44bplib.h\
$(INC)\option.h makefile
    del $(OBJ)\$(CM8).o
    del $(ERR)\$(CM8).err
    $(ARMCC) $(CFLAGS) $(SRC)\$(CM8).c -o $(OBJ)\$(CM8).o -Errors $(ERR)\$(CM8).err

$(OBJ)\$(CM9).o: $(SRC)\$(CM9).c $(INC)\$(CM9).h $(INC)\44b.h $(INC)\44bplib.h\
$(INC)\option.h makefile
    del $(OBJ)\$(CM9).o
    del $(ERR)\$(CM9).err
    $(ARMCC) $(CFLAGS) $(SRC)\$(CM9).c -o $(OBJ)\$(CM9).o -Errors $(ERR)\$(CM9).err

$(OBJ)\$(CM10).o: $(SRC)\$(CM10).c $(INC)\$(CM10).h $(INC)\44b.h $(INC)\44bplib.h\
$(INC)\option.h makefile
    del $(OBJ)\$(CM10).o
    del $(ERR)\$(CM10).err
    $(ARMCC) $(CFLAGS) $(SRC)\$(CM10).c -o $(OBJ)\$(CM10).o -Errors $(ERR)\$(CM10).err

$(OBJ)\$(CM11).o: $(SRC)\$(CM11).c $(INC)\$(CM11).h $(INC)\44b.h $(INC)\44bplib.h\
$(INC)\option.h makefile
    del $(OBJ)\$(CM11).o
    del $(ERR)\$(CM11).err
    $(ARMCC) $(CFLAGS) $(SRC)\$(CM11).c -o $(OBJ)\$(CM11).o -Errors $(ERR)\$(CM11).err

$(OBJ)\$(CM12).o: $(SRC)\$(CM12).c $(INC)\$(CM12).h $(INC)\44b.h $(INC)\44bplib.h\
$(INC)\option.h makefile
    del $(OBJ)\$(CM12).o
    del $(ERR)\$(CM12).err
    $(ARMCC) $(CFLAGS) $(SRC)\$(CM12).c -o $(OBJ)\$(CM12).o -Errors $(ERR)\$(CM12).err

$(OBJ)\$(CM13).o: $(SRC)\$(CM13).c $(INC)\$(CM13).h $(INC)\44b.h $(INC)\44bplib.h\
$(INC)\option.h makefile
    del $(OBJ)\$(CM13).o
    del $(ERR)\$(CM13).err
    $(ARMCC) $(CFLAGS) $(SRC)\$(CM13).c -o $(OBJ)\$(CM13).o -Errors $(ERR)\$(CM13).err

$(OBJ)\$(CM14).o: $(SRC)\$(CM14).c $(INC)\$(CM14).h $(INC)\44b.h $(INC)\44bplib.h\
$(INC)\option.h makefile
    del $(OBJ)\$(CM14).o
    del $(ERR)\$(CM14).err
    $(ARMCC) $(CFLAGS) $(SRC)\$(CM14).c -o $(OBJ)\$(CM14).o -Errors $(ERR)\$(CM14).err

$(OBJ)\$(CM15).o: $(SRC)\$(CM15).c $(INC)\$(CM15).h $(INC)\44b.h $(INC)\44bplib.h\
$(INC)\option.h makefile
    del $(OBJ)\$(CM15).o
    del $(ERR)\$(CM15).err
    $(ARMCC) $(CFLAGS) $(SRC)\$(CM15).c -o $(OBJ)\$(CM15).o -Errors $(ERR)\$(CM15).err

$(OBJ)\$(CM16).o: $(SRC)\$(CM16).c $(INC)\$(CM16).h $(INC)\44b.h $(INC)\44bplib.h\
$(INC)\option.h makefile
    del $(OBJ)\$(CM16).o
    del $(ERR)\$(CM16).err
    $(ARMCC) $(CFLAGS) $(SRC)\$(CM16).c -o $(OBJ)\$(CM16).o -Errors $(ERR)\$(CM16).err
```

```
$(OBJ)\$(CM17).o: $(SRC)\$(CM17).c $(INC)\$(CM17).h $(INC)\44b.h $(INC)\44blib.h\
$(INC)\option.h makefile
    del $(OBJ)\$(CM17).o
    del $(ERR)\$(CM17).err
    $(ARMCC) $(CFLAGS) $(SRC)\$(CM17).c -o $(OBJ)\$(CM17).o -Errors $(ERR)\$(CM17).err

$(OBJ)\$(CM18).o: $(SRC)\$(CM18).c $(INC)\$(CM18).h $(INC)\44b.h $(INC)\44blib.h\
$(INC)\option.h makefile
    del $(OBJ)\$(CM18).o
    del $(ERR)\$(CM18).err
    $(ARMCC) $(CFLAGS) $(SRC)\$(CM18).c -o $(OBJ)\$(CM18).o -Errors $(ERR)\$(CM18).err

$(OBJ)\$(CM19).o: $(SRC)\$(CM19).c $(INC)\$(CM19).h $(INC)\44b.h $(INC)\44blib.h\
$(INC)\option.h makefile
    del $(OBJ)\$(CM19).o
    del $(ERR)\$(CM19).err
    $(ARMCC) $(CFLAGS) $(SRC)\$(CM19).c -o $(OBJ)\$(CM19).o -Errors $(ERR)\$(CM19).err

$(OBJ)\$(CM20).o: $(SRC)\$(CM20).c $(INC)\$(CM20).h $(INC)\44b.h $(INC)\44blib.h\
$(INC)\option.h makefile
    del $(OBJ)\$(CM20).o
    del $(ERR)\$(CM20).err
    $(ARMCC) $(CFLAGS) $(SRC)\$(CM20).c -o $(OBJ)\$(CM20).o -Errors $(ERR)\$(CM20).err

$(OBJ)\$(CM21).o: $(SRC)\$(CM21).c $(INC)\$(CM21).h $(INC)\44b.h $(INC)\44blib.h\
$(INC)\option.h makefile
    del $(OBJ)\$(CM21).o
    del $(ERR)\$(CM21).err
    $(ARMCC) $(CFLAGS) $(SRC)\$(CM21).c -o $(OBJ)\$(CM21).o -Errors $(ERR)\$(CM21).err

$(OBJ)\$(CM22).o: $(SRC)\$(CM22).c $(INC)\$(CM22).h $(INC)\44b.h $(INC)\44blib.h\
$(INC)\option.h makefile
    del $(OBJ)\$(CM22).o
    del $(ERR)\$(CM22).err
    $(ARMCC) $(CFLAGS) $(SRC)\$(CM22).c -o $(OBJ)\$(CM22).o -Errors $(ERR)\$(CM22).err

$(OBJ)\$(CM23).o: $(SRC)\$(CM23).c $(INC)\$(CM23).h $(INC)\44b.h $(INC)\44blib.h\
$(INC)\option.h makefile
    del $(OBJ)\$(CM23).o
    del $(ERR)\$(CM23).err
    $(ARMCC) $(CFLAGS) $(SRC)\$(CM23).c -o $(OBJ)\$(CM23).o -Errors $(ERR)\$(CM23).err
```

**44BTEST : option.a**

```

;*****OPTIONS*****
;_RAM_STARTADDRESS EQU 0xc000000
;_ISR_STARTADDRESS EQU 0xc7fff00 ;GCS6:64M DRAM/SDRAM
;_ISR_STARTADDRESS EQU 0xc1fff00 ;GCS6:16M DRAM

;BUSWIDTH; 16,32
                GBLA      BUSWIDTH
BUSWIDTH        SETA      16

; "DRAM", "SDRAM"
                GBLA      BDRAMTYPE
BDRAMTYPE        SETS      "SDRAM"

;This value has to be TRUE on ROM program.
;This value has to be FALSE in RAM program.
                GBLA      PLLONSTART
PLLONSTART        SETL      {TRUE}

                GBLA      PLLCLK
PLLCLK            SETA      60000000

                [
M_DIV EQU        PLLCLK = 40000000
P_DIV EQU        0x48 ;Fin=10MHz Fout=40MHz
S_DIV EQU        0x3
                ]

                [
M_DIV EQU        PLLCLK = 50000000
P_DIV EQU        0x2a ;Fin=10MHz Fout=50MHz
S_DIV EQU        0x3
                ]

                [
M_DIV EQU        PLLCLK = 60000000
P_DIV EQU        0x34 ;Fin=10MHz Fout=60MHz
S_DIV EQU        0x3
;M_DIV EQU        0x34 ;Fin=4MHz Fout=60MHz
;P_DIV EQU        0x0
;S_DIV EQU        0x1
;M_DIV EQU        0x48 ;Fin=3MHz Fout=60MHz
;P_DIV EQU        0x0
;S_DIV EQU        0x1
                ]

                [
M_DIV EQU        PLLCLK = 66000000
P_DIV EQU        0x3a ;Fin=10MHz Fout=75MHz
S_DIV EQU        0x3
                ]

                [
M_DIV EQU        PLLCLK = 75000000
P_DIV EQU        0x43 ;Fin=10MHz Fout=75MHz
S_DIV EQU        0x3
                ]
;*****
END

```

**44BTEST : option.h**

```

#ifndef __OPTION_H__
#define __OPTION_H__

// ***** OPTIONS *****
#define MCLK 60000000

#define WRBUFOPT (0x8)    //write_buf_on

#define SYSCFG_0KB (0x0|WRBUFOPT)
#define SYSCFG_4KB (0x2|WRBUFOPT)
#define SYSCFG_8KB (0x6|WRBUFOPT)

#define DRAM      1        //In case DRAM is used
#define SDRAM     2        //In case SDRAM is used
#define BDRAMTYPE SDRAM //used in power.c,44bplib.c

//BUSWIDTH; 16,32
#define BUSWIDTH (16)

#define CACHECFG SYSCFG_8KB

#define _RAM_STARTADDRESS 0xc000000

//#define _ISR_STARTADDRESS 0xc1fff00    //GCS6:16M DRAM
#define _ISR_STARTADDRESS 0xc7fff00    //GCS6:64M DRAM/SDRAM

// NOTE: rom.mak,option.a have to be changed
#endif /*__OPTION_H__*/

```

**44BTEST : 44btest.c**

```

#include <string.h>
#include "..\inc\option.h"
#include "..\inc\44b.h"
#include "..\inc\44blib.h"
#include "..\inc\def.h"
#include "..\inc\cache.h"
#include "..\inc\lcd.h"
#include "..\inc\lcdlib.h"
#include "..\inc\glib.h"
#include "..\inc\clcd.h"
#include "..\inc\adc.h"
#include "..\inc\uart.h"
#include "..\inc\power.h"
#include "..\inc\dma.h"
#include "..\inc\timer.h"
#include "..\inc\sio.h"
#include "..\inc\rtc.h"
#include "..\inc\etc.h"
#include "..\inc\iic.h"
#include "..\inc\eint.h"
#include "..\inc\flash.h"
#include "..\inc\iis.h"
#include "..\inc\stop.h"
#include "..\inc\idle.h"
#include "..\inc\extdma.h"
#include "..\inc\nwait.h"
#include "..\inc\Am29f800.h"

void AutoTest(void);
void Isr_Init(void);
void HaltUndef(void);
void HaltSwi(void);
void HaltPabort(void);
void HaltDabort(void);

/*****
*           S3C44B0X developer's notes           *
*****/

1. 99.6.6:KIW: For H/W vectored interrupt test, add the codes in Isr_Init()
2. 99.8.27: Stop,SL_IDLE mode is modified.
3. 99.10.13: check KS32C41000 date:A937 or after.
4. 99.10.19: AM29LV800 flash writing routine is added
5. 00.4.25:KWT: Modify for S3C44B0X
*****/

void * function[][2]=
{
  #if (LCD_TYPE==MLCD_320_240)
    (void *)Test_LcdMono,      "Lcd Mono",      ",
    (void *)Test_LcdG4,      "Lcd G4",      ",
    (void *)Test_LcdG16,      "Lcd G16",      ",
  #elif (LCD_TYPE==CLCD_240_320)
    (void *)Test_LcdColor,      "Lcd Color",      ",
  #endif
    (void *)Test_SLIdleMode, "SL_IDLE Mode",      ",
    (void *)Test_SLIdleMode20, "SL_IDLE Mode20",      ",
    (void *)Test_IdleMode,      "IDLE Mode",      ",
    (void *)Test_IdleModeHard,"IDLE(hard)",      ",
    (void *)Test_WaitPin,      "nWAIT pin",      ",

```

```

(void *)Test_Zdma0Xdreq, "nXDREQ0",
(void *)Test_Cache, "Cache",
(void *)Test_Adc, "Adc 0,1,2,3",
(void *)Test_DMA_Adc, "Adc with DMA",
(void *)Test_Uart0, "UART 0",
(void *)Test_Uart0Fifo, "UART 0 FIFO",
(void *)Test_Uart1, "UART 1",
(void *)Test_Uart1Fifo, "UART 1 FIFO",
(void *)Test_SlowMode, "SLOW Mode",
(void *)Test_HoldMode, "HOLD Mode",
(void *)Test_StopMode, "STOP Mode",
(void *)Test_Zdma0, "Zdma0",
(void *)Test_Zdma1, "Zdma1",
(void *)Test_Sio, "SIO Tx/Rx",
(void *)Test_SIOTX_BDMA0, "SIO Tx BDMA0",
(void *)Test_SIORX_BDMA1, "SIO Rx BDMA1",
(void *)Test_WDTimer, "WDTimer",
(void *)Display_Rtc, "RTC(display)",
(void *)Test_Rtc_Alarm, "RTC(Test)",
(void *)Test_Rtc_Tick, "RTC Tick",
(void *)Test_Iic, "IIC(KS24C080)",
(void *)Test_Iis, "IIS(uda1341)",
(void *)Test_TimerInt, "Timer Int",
(void *)Test_Timer, "Tout test",
(void *)Test_Eint, "Ext. Int",
(void *)Etc, "Etc...",
(void *)Test_PLL, "Change PLL",
(void *)Test_UartAFC_Tx, "Test AFC(Tx)",
(void *)Test_UartAFC_Rx, "Test AFC(Rx)",
(void *)ProgramFlash, "Write flash",
0,0
};

void Main(void)
{
    int i;

    rSYSCFG=CACHECFG;
#ifdef (PLLON==1)
    ChangePllValue(PLL_M,PLL_P,PLL_S);
#endif

    //_ctype_init(); //to use ctype.h
    Port_Init();
    Isr_Init();

    Uart_Init(0,115200);

    //rBUSCON=(1<<31)|(3<<6)|(2<<4)|(1<<2)|(0); //nBREQ>BDMA>ZDMA>LCD_DMA

    Uart_Select(0);
    Delay(0); //calibrate Delay()

    while(1)
    {
        i=0;
        Uart_Printf("\n\nS3C44B0X Test Program Ver 0.00 rSYSCFG=0x%x\n\n",rSYSCFG,MCLK);

        while(1)
        {
            //display menu
            Uart_Printf("%2d:%s",i,function[i][1]);

```

```

        i++;
        if((int)(function[i][0])==0)
        {
            Uart_Printf("\n");
            break;
        }
        if((i%4)==0)
            Uart_Printf("\n");
    }

    Uart_Printf("\nSelect the function to test?");

    i=Uart_GetIntNum();
    Uart_Printf("\n");
    if(i>=0 && (i<(sizeof(function)/8)) )
        ( (void (*)(void)) (function[i][0]) )();
    }
}

void Isr_Init(void)
{
    pISR_UNDEF=(unsigned)HaltUndef;
    pISR_SWI  =(unsigned)HaltSwi;
    pISR_PABORT=(unsigned)HaltPabort;
    pISR_DABORT=(unsigned)HaltDabort;

    //rINTCON=0x1;    // Vectored Int. IRQ enable,FIQ disable
    rINTCON=0x5;      // Non-vectored,IRQ enable,FIQ disable

    rINTMOD=0x0;      // All=IRQ mode
    rINTMSK=BIT_GLOBAL; // All interrupt is masked.
}

void HaltUndef(void)
{
    Uart_Printf("Undefined instruction exception!!!\n");
    while(1);
}

void HaltSwi(void)
{
    Uart_Printf("SWI exception!!!\n");
    while(1);
}

void HaltPabort(void)
{
    Uart_Printf("Pabort exception!!!\n");
    while(1);
}

void HaltDabort(void)
{
    Uart_Printf("Dabort exception!!!\n");
    while(1);
}

```



**44BTEST : adc.h**

```

#ifndef __ADC_H__
#define __ADC_H__

void Test_DMA_Adc(void);
void Test_Adc(void);

#endif /*__ADC_H__*/

```

**44BTEST : adc.c**

```

#include <string.h>
#include "..\inc\44b.h"
#include "..\inc\44blib.h"
#include "..\inc\adc.h"

void __irq BDMA1_Done(void);

void Test_Adc(void)
{
    Uart_Printf("The ADC_IN are adjusted to the following values.\n");
    Uart_Printf("Push any key to exit!!!\n");

    rCLKCON=0x7ff8;

    while(Uart_GetKey()==0)
    {
        rADCCON=0x0;          // MCLK/2

        rADCCON=0x1|(0x0<<2);    // AIN0
        while(!(rADCCON & 0x40));
        Uart_Printf("A0=%03xh ",rADCDAT);

        rADCCON=0x1|(0x1<<2);    // AIN1
        while(!(rADCCON & 0x40));
        Uart_Printf("A1=%03xh ",rADCDAT);

        rADCCON=0x1|(0x2<<2);    // AIN2
        while(!(rADCCON & 0x40));
        Uart_Printf("A2=%03xh ",rADCDAT);

        rADCCON=0x1|(0x3<<2);    // AIN3
        while(!(rADCCON & 0x40));
        Uart_Printf("A3=%03xh\n",rADCDAT);

        rADCCON=0x1|(0x4<<2);    // AIN4
        while(!(rADCCON & 0x40));
        Uart_Printf("A4=%03xh ",rADCDAT);

        rADCCON=0x1|(0x5<<2);    // AIN5
        while(!(rADCCON & 0x40));
        Uart_Printf("A5=%03xh ",rADCDAT);

        rADCCON=0x1|(0x6<<2);    // AIN6
        while(!(rADCCON & 0x40));
        Uart_Printf("A6=%03xh ",rADCDAT);
    }
}

```

```

        rADCCON=0x1|(0x7<<2);        // AIN7
        while(!(rADCCON & 0x40));
        Uart_Printf("A3=%03xh\n",rADCDAT);
    }
}

volatile char end_test;

void Test_DMA_Adc(void)
{
    unsigned int *dst, *temp,i;
    end_test=0;

    pISR_BDMA1=(unsigned)BDMA1_Done;
    rINTMSK=~(BIT_GLOBAL|BIT_BDMA1);

    Uart_Printf("Test of the 'Start by read' in ADC block.\n");
    Uart_Printf("Change the value of AIN0\n");

    //    rCLKCON=0x7ff8;
    dst=(unsigned int *)malloc(0x100);
    rNCACHBE0= ((int)dst>>12) + ( (((int)dst>>12) +1)<<16 );

    temp=dst;

    /***ADC init***/
    rADCCON=0x2;        //normal,AIN0,enable start by read
    rADCPSR=0x4;        // MCLK/2

    /*
    while(1)
    {
        if( rADCCON & 0x40 )
        {
            i=rADCDAT;
            Uart_Printf("!!!0x%03x, ",i);
            for(i=0;i<0x10;i++);
        }
    }

    */

    /***BDMA1 init***/
    rBDISRC1=(2<<30)+(3<<28)+0x1d40008;        //word,peri.(ADC)
    rBDIDES1=(2<<30)+(1<<28)+(unsigned)dst;    //IO2M,++
    rBDICNT1=(1<<30)+(1<<26)+(3<<22)+(1<<20)+10*4;//timer,unit,end-
    >interrupt,enable,count=10(word)

    rBDCON1 = 0x0;

    /***Timer0 init***/
    rTCFG0=255;        //prescaler0=255
    rTCFG1=(1<<24)+4;    //Timer0 DMA, div=32
    rTCNTB0=4902;    //(1/(40MHz/255/32))*4902=1.xx s
    rTCON=0xa;        //auto, update

    rTCON=0x9;        //Start

    while(!end_test);

    Uart_Printf("dst=0x%x,temp=0x%x\n",dst,temp);

```

```
    for(i=0;i<10;i++)
        Uart_Printf("d=0x%03x\n",i,*temp++);
    Uart_Printf("dst=0x%x,temp=0x%x\n",dst,temp);

    free(dst);
    rINTMSK=BIT_GLOBAL;
}

void __irq BDMA1_Done(void)
{
    rI_ISPC=BIT_BDMA1; //clear pending bit
    rTCON=0x0;          //Stop
    end_test=1;          //set end flag
}
```

**44BTEST : am29f800.h**

```
#ifndef __AM29F800_h__
#define __AM29F800_h__

void ProgramAM29F800(void);

#endif /*AM29F800*/
```

**44BTEST : am29f800.c**

```
#include "..\inc\44b.h"
#include "..\inc\44blib.h"
#include "..\inc\def.h"

int    AM29F800_ProgFlash(U32 realAddr,U16 data);
void    AM29F800_EraseSector(int targetAddr);
int    AM29F800_CheckId(void);
int    BlankCheck(int targetAddr,int targetSize);
int    _WAIT(void);

static void InputTargetAddr(void);

// Because S3C44B0X is connected to AM29LV800B,
// the addr parameter has to be a WORD address, so called in AMD specification.

#define _WR(addr,data)    *((U16 *)(addr<<1))=(U16)data    //the addr should be shifted
#define _RD(addr)        ( *((U16 *)(addr<<1)) )
#define _RESET()         _WR(0x0,0xf0f0)
#define BADDR2WADDR(addr) (addr>>1)

extern U32 downloadAddress;
extern U32 downloadProgramSize;

U32 srcAddress;
U32 srcOffset;
U32 targetAddress;
U32 targetSize;

void ProgramAM29F800(void)
{
    int i;

    InputTargetAddr();
    srcAddress=downloadAddress+4;

    Uart_Printf("[Check AM29LV800]\n");
    if(!AM29F800_CheckId())
    {
        Uart_Printf("ID Check Error!!!\n");
        return;
    }

    Uart_Printf("\nErase the sector:0x%x.\n",targetAddress);

    AM29F800_EraseSector(targetAddress);
```

```

if(!BlankCheck(targetAddress,targetSize))
{
    Uart_Printf("Blank Check Error!!!\n");
    return;
}

Uart_Printf("\nStart of the data writing.\n");

for(i=0;i<targetSize;i+=2)
{
    AM29F800_ProgFlash( i+targetAddress,* (U16 *) (srcAddress+srcOffset+i) );
    if((i%0x1000)==0)
        Uart_Printf("%x ",i);
}
Uart_Printf("\nEnd of the data writing!!!\n");

_RESET();

Uart_Printf("\nVerifying Start.\n");
for(i=0x0;i<targetSize;i+=2)
{
    if(* (U16 *) (i+targetAddress) ) != * (U16 *) (srcAddress+srcOffset+i) )
    {
        Uart_Printf("%x=verify error\n",i+targetAddress);
        return;
    }
    if((i%0x1000)==0)
        Uart_Printf("%x ",i);
}
Uart_Printf("\nVerifying End!!!\n");

Uart_Printf("Do you want another programming without additional download? [y/n]\n");
if(Uart_Getch()=='y')
    ProgramAM29F800();
}

static void InputTargetAddr(void)
{
    Uart_Printf("[AM29F800 Writing Program]\n");

    Uart_Printf("\nSource size:0h~%xh\n",downloadProgramSize);
    Uart_Printf("\nAvailable Target/Source Address:\n");
    Uart_Printf("    0h, 4000h, 6000h, 8000h,10000h,20000h,30000h,40000h,50000h,60000h\n");
    Uart_Printf("70000h,80000h,90000h,a0000h,b0000h,c0000h,d0000h,e0000h,f0000h\n");

    Uart_Printf("Input source offset:");
    srcOffset=Uart_GetIntNum();
    Uart_Printf("Input target address among above addresses:");
    targetAddress=Uart_GetIntNum();

    if(targetAddress<0x4000)
        targetSize=0x4000;
    else if(targetAddress<0x6000)
        targetSize=0x2000;
    else if(targetAddress<0x8000)
        targetSize=0x2000;
    else if(targetAddress<0x10000)
        targetSize=0x8000;
    else
        targetSize=0x10000;
}

```

```

    Uart_Printf("source offset=0x%x\n",srcOffset);
    Uart_Printf("target address=0x%x\n",targetAddress);
    Uart_Printf("target block size=0x%x\n",targetSize);
}

int AM29F800_CheckId(void)
{
    U16 manId,devId;

    _RESET();

    _WR(0x555,0xaaaa);
    _WR(0x2aa,0x5555);
    _WR(0x555,0x9090);
    manId=_RD(0x0);

    _WR(0x555,0xaaaa);
    _WR(0x2aa,0x5555);
    _WR(0x555,0x9090);
    devId=_RD(0x1);

    _RESET();

    Uart_Printf("Manufacture ID=%4x(0x0001), Device ID(0x225B)=%4x\n",manId,devId);
    if(manId==0x0001 && devId==0x225b)
        return 1;
    else
        return 0;
}

void AM29F800_EraseSector(int targetAddr)
{
    Uart_Printf("Sector Erase is started!\n");

    _RESET();

    _WR(0x555,0xaaaa);
    _WR(0x2aa,0x5555);
    _WR(0x555,0x8080);
    _WR(0x555,0xaaaa);
    _WR(0x2aa,0x5555);
    _WR(BADDR2WADDR(targetAddr),0x3030);
    _WAIT();
    _RESET();
}

int BlankCheck(int targetAddr,int targetSize)
{
    int i,j;
    for(i=0;i<targetSize;i+=2)
    {
        j=((U16 *) (i+targetAddr));
        if( j!=0xffff)
        {
            Uart_Printf("E:%x=%x\n", (i+targetAddr),j);
            return 0;
        }
    }
    return 1;
}

```

```

int _WAIT(void) //Check if the bit6 toggle ends.
{
    volatile U16 flashStatus,old;

    old=((volatile U16 *)0x0);

    while(1)
    {
        flashStatus=((volatile U16 *)0x0);
        if( (old&0x40) == (flashStatus&0x40) )
            break;
        if( flashStatus&0x20 )    //Time_limit_over check(in case normally end with set DQ5)
        {
            //Uart_Printf("[DQ5=1:%x]\n",flashStatus);
            old=((volatile U16 *)0x0);
            flashStatus=((volatile U16 *)0x0);
            if( (old&0x40) == (flashStatus&0x40) )
                return 1;    //No toggle
            else
                return 0;    //toggling
        }
        //Uart_Printf(".");
        old=flashStatus;
    }
    //Uart_Printf("!\n");
    return 1;
}

```

```

int AM29F800_ProgFlash(U32 realAddr,U16 data)
{
    volatile U16 *tempPt;
    int temp,count=0;
    tempPt=(volatile U16 *)realAddr;
    _WR(0x555,0xaaaa);
    _WR(0x2aa,0x5555);
    _WR(0x555,0xa0a0);
    *tempPt=data;

    return _WAIT();
/*
    while(1)
    {
        temp=*tempPt;
        if(temp==data || count==100)break;
        if(temp&0x20)
        {
            count++;
        }
    }
    if(count>0)Uart_Printf("Time out is occurred at %x\n",realAddress);
*/
}

```

**44BTEST : cache.h**

```
#ifndef __CACHE_H__
#define __CACHE_H__

int Test_Cache(void);
void FlushCache(void);

#endif /*__CACHE_H__*/
```

**44BTEST : cache.c**

```
#include <string.h>
#include "..\inc\44b.h"
#include "..\inc\44bllib.h"

void MarchCMinus32(int address,int unit,unsigned int pattern);
void MarchCMinusM_10(int address,int unit,unsigned int pattern,unsigned int mask);
void _MarchSub1(unsigned int *pt,int unit,unsigned int pattern,int incdec);
void _MarchSub1_10(unsigned int *pt,int unit,unsigned int pattern,int incdec,unsigned int
mask);
int marchError;

void FlushCache(void)
{
    int i,saveSyscfg;

    saveSyscfg=rSYSCFG;

    rSYSCFG=SYSCFG_0KB;
    for(i=0x10002000;i<0x10004800;i+=16)
    {
        *((int *)i)=0x0;
    }

    rSYSCFG=saveSyscfg;
}

int Test_Cache(void)
{
    int saveSyscfg;
    Uart_Printf("Cache(Internal RAM) Cell Test by March C-\n");

    saveSyscfg=rSYSCFG;
    marchError=0;

    //00->01->11<->10

    rSYSCFG=SYSCFG_0KB; //WB_off,Cache_off,I_RAM 8KB,stall disable
    //set0=0x10000000~0x100007ff
    //set1=0x10000800~0x10000fff
    //set2=0x10001000~0x100017ff
    //set3=0x10001800~0x10001fff

    MarchCMinus32(0x10000000,511,0x00000000);
    MarchCMinus32(0x10000000,511,0x0f0f0f0f);
    MarchCMinus32(0x10000000,511,0x33333333);
    MarchCMinus32(0x10000000,511,0x55555555);
    MarchCMinus32(0x10000000,511,0xaaaaaaaa);
    Uart_Printf("Set0 is tested!\n");
```



```

MarchCMinus32(0x10000800,511,0x00000000);
MarchCMinus32(0x10000800,511,0x0f0f0f0f);
MarchCMinus32(0x10000800,511,0x33333333);
MarchCMinus32(0x10000800,511,0x55555555);
MarchCMinus32(0x10000800,511,0xaaaaaaaa);
Uart_Printf("Set1 is tested!\n");

MarchCMinus32(0x10001000,511,0x00000000);
MarchCMinus32(0x10001000,511,0x0f0f0f0f);
MarchCMinus32(0x10001000,511,0x33333333);
MarchCMinus32(0x10001000,511,0x55555555);
MarchCMinus32(0x10001000,511,0xaaaaaaaa);
Uart_Printf("Set2 is tested!\n");

MarchCMinus32(0x10001800,511,0x00000000);
MarchCMinus32(0x10001800,511,0x0f0f0f0f);
MarchCMinus32(0x10001800,511,0x33333333);
MarchCMinus32(0x10001800,511,0x55555555);
MarchCMinus32(0x10001800,511,0xaaaaaaaa);
Uart_Printf("Set3 is tested!\n");

MarchCMinusM_10(0x10002000,127,0x00000000,0x1ffff);
MarchCMinusM_10(0x10002000,127,0x0000ffff,0x1ffff);
MarchCMinusM_10(0x10002000,127,0x00ff00ff,0x1ffff);
MarchCMinusM_10(0x10002000,127,0x0f0f0f0f,0x1ffff);
MarchCMinusM_10(0x10002000,127,0x33333333,0x1ffff);
MarchCMinusM_10(0x10002000,127,0x55555555,0x1ffff);
Uart_Printf("TagRAM0 is tested!\n");

MarchCMinusM_10(0x10002800,127,0x00000000,0x1ffff);
MarchCMinusM_10(0x10002800,127,0x0000ffff,0x1ffff);
MarchCMinusM_10(0x10002800,127,0x00ff00ff,0x1ffff);
MarchCMinusM_10(0x10002800,127,0x0f0f0f0f,0x1ffff);
MarchCMinusM_10(0x10002800,127,0x33333333,0x1ffff);
MarchCMinusM_10(0x10002800,127,0x55555555,0x1ffff);
Uart_Printf("TagRAM1 is tested!\n");

MarchCMinusM_10(0x10003000,127,0x00000000,0x1ffff);
MarchCMinusM_10(0x10003000,127,0x0000ffff,0x1ffff);
MarchCMinusM_10(0x10003000,127,0x00ff00ff,0x1ffff);
MarchCMinusM_10(0x10003000,127,0x0f0f0f0f,0x1ffff);
MarchCMinusM_10(0x10003000,127,0x33333333,0x1ffff);
MarchCMinusM_10(0x10003000,127,0x55555555,0x1ffff);
Uart_Printf("TagRAM2 is tested!\n");

MarchCMinusM_10(0x10003800,127,0x00000000,0x1ffff);
MarchCMinusM_10(0x10003800,127,0x0000ffff,0x1ffff);
MarchCMinusM_10(0x10003800,127,0x00ff00ff,0x1ffff);
MarchCMinusM_10(0x10003800,127,0x0f0f0f0f,0x1ffff);
MarchCMinusM_10(0x10003800,127,0x33333333,0x1ffff);
MarchCMinusM_10(0x10003800,127,0x55555555,0x1ffff);
Uart_Printf("TagRAM3 is tested!\n");

MarchCMinusM_10(0x10004000,127,0x00000000,0xf);
MarchCMinusM_10(0x10004000,127,0x0000ffff,0xf);
MarchCMinusM_10(0x10004000,127,0x00ff00ff,0xf);
MarchCMinusM_10(0x10004000,127,0x0f0f0f0f,0xf);
MarchCMinusM_10(0x10004000,127,0x33333333,0xf);
MarchCMinusM_10(0x10004000,127,0x55555555,0xf);
Uart_Printf("LRU is tested!\n");

```

```

Uart_Printf("4KB internal SRAM test.\n");

rSYSCFG=SYSCFG_4KB;
MarchCMinus32(0x10001000,511,0x00000000);
MarchCMinus32(0x10001000,511,0x0f0f0f0f);
MarchCMinus32(0x10001000,511,0x33333333);
MarchCMinus32(0x10001000,511,0x55555555);
MarchCMinus32(0x10001000,511,0xaaaaaaaa);
Uart_Printf("Set2 is tested at 4KB!\n");

MarchCMinus32(0x10001800,511,0x00000000);
MarchCMinus32(0x10001800,511,0x0f0f0f0f);
MarchCMinus32(0x10001800,511,0x33333333);
MarchCMinus32(0x10001800,511,0x55555555);
MarchCMinus32(0x10001800,511,0xaaaaaaaa);
Uart_Printf("Set3 is tested at 4KB!\n");

rSYSCFG=saveSyscfg;      //wr_on,Cache 8KB+I_RAM 0KB,stall disable.

//cache flush is needed....

if(marchError==0)
    return 1;
else
    return 0;
}

void MarchCMinus32(int address,int unit,unsigned int pattern)
{
    int i;
    unsigned int *tempPt;

    tempPt=(unsigned int *)address;

    for(i=0;i<=unit;i++)
    {
        //Uart_Printf("P=%x",pattern);
        *tempPt+=pattern;
        //Uart_Printf("A=%x,P=%x\n",tempPt,*tempPt);
    }

    _MarchSub1((unsigned int *)address,unit,~pattern,1);
    _MarchSub1((unsigned int *)address,unit,pattern,1);
    _MarchSub1((unsigned int *)address+unit,unit,~pattern,-1);
    _MarchSub1((unsigned int *)address+unit,unit,pattern,-1);

    tempPt=(unsigned int *)address;
    for(i=0;i<=unit;i++)
    {
        if(*tempPt!=pattern)
        {
            marchError=1;
            Uart_Printf("ERROR0:%x\n",tempPt);
        }
        tempPt++;
    }
    //Uart_Printf("RP=%x,RA=%x\n",*tempPt,tempPt);
}

```

```

void MarchCMinusM_10(int address,int unit,unsigned int pattern,unsigned int mask)
{
    int i;
    unsigned int *tempPt;

    tempPt=(unsigned int *)address;

    for(i=0;i<=unit;i++)
    {
        //Uart_Printf("P=%x",pattern);
        //    *tempPt++=pattern;
        //Uart_Printf("A=%x,P=%x\n",tempPt,*tempPt);
        *tempPt=pattern;
        tempPt+=0x4;
    }

    _MarchSubl_10((unsigned int *)address,unit,~pattern,4,mask);
    _MarchSubl_10((unsigned int *)address,unit,pattern,4,mask);
    _MarchSubl_10((unsigned int *)address+(unit*4),unit,~pattern,-4,mask);
    _MarchSubl_10((unsigned int *)address+(unit*4),unit,pattern,-4,mask);

    tempPt=(unsigned int *)address;

    for(i=0;i<=unit;i++)
    {
        if( *tempPt!=(pattern&mask) )
        {
            marchError=1;
            Uart_Printf("ERROR0:%x\n",tempPt);
        }
        tempPt+=0x4;
    }
    //Uart_Printf("RP=%x,RA=%x\n",*tempPt,tempPt);
}

```

```

void _MarchSubl(unsigned int *pt,int unit,unsigned int pattern,int incdec)
{
    unsigned int i,rp;

    for(i=0;i<=unit;i++)
    {
        rp=*pt;
        //Uart_Printf("RP=%x",*pt);

        *pt=pattern;
        //Uart_Printf("j=%x,p=%x,pt=%x\n",rp,(~pattern),pt);

        if(rp!=(~pattern))
        {
            marchError=1;
            Uart_Printf("ERROR1:A=%x,RP=%x,WP=%x \n",pt,rp,(~pattern));
        }
        pt+=incdec;
    }
}

```

```
void _MarchSub1_10(unsigned int *pt,int unit,unsigned int pattern,int incdec,unsigned int
mask)
{
    unsigned int i,rp;
    unsigned int *pt2=pt;

    for(i=0;i<=unit;i++)
    {
        rp=*pt;
        //Uart_Printf("RP=%x",*pt);
        *pt=pattern;
        //Uart_Printf("j=%x,p=%x,pt=%x\n",rp,(~pattern),pt);

        if( rp!=(mask&(~pattern)) )
        {
            marchError=1;
            Uart_Printf("ERROR1:A=%x,RP=%x,WP=%x pt2=%x \n",pt,rp,(~pattern),pt2);
        }
        pt+=incdec;
    }
}
```

**44BTEST : clcd.h**

```
#ifndef __CLCD_H__
#define __CLCD_H__

void Clcd_Init(void);
void Test_Clcd(void);

#endif /*__CLCD_H__*/
```

**44BTEST : clcd.c**

```
#include <string.h>
#include "..\inc\44b.h"
#include "..\inc\44blib.h"
#include "..\inc\clcd.h"
#include "..\inc\def.h"

void Lcd_PutPixel(int x,int y,U8 rgb);
void DrawLine(void);

unsigned int (*frameBuffer256)[60]=(unsigned int (*)(60))0x0;

void Test_Clcd(void)
{
    int i,j;

    Uart_Printf("[ (240x3)x320 COLOR STN LCD TEST]\n");

    Uart_Printf("1-1)COLOR BAR TEST\n");

    Uart_Printf("      R:0    ...    7 \n");
    Uart_Printf("G:0  B0:1 B0:1 B0:1 \n");
    Uart_Printf("G:.   2:3  2:3  2:3 \n");
    Uart_Printf("G:.   B0:1 B0:1 B0:1 \n");
    Uart_Printf("G:.   2:3  2:3  2:3 \n");
    Uart_Printf("G:.   B0:1 B0:1 B0:1 \n");
    Uart_Printf("G:7   2:3  2:3  2:3 \n");

    Clcd_Init();

    for(j=0;j<320;j++)
        for(i=0;i<240;i++)
        {
            //RRRGGBB
            Lcd_PutPixel( i,j,(i/30<<5)+(j/40<<2)+((i/15)&0x1)+((j/20&0x1)<<1) );
        }

    Uart_Printf("Press any key!!!\n");

    DrawLine();

    Uart_Getch();
}
```

```

void DrawLine(void)
{
    int x=0,y=0,xd=1,yd=1,count=0;
    while(Uart_GetKey()==0)
    {
        if(x==0)xd=1;
        if(x==240)xd=-1;
        if(y==0)yd=1;
        if(y==319)yd=-1;

        Lcd_PutPixel(x,y,0xff);
        x+=xd;
        y+=yd;
        Delay(100);
        if(count++==2000)break;
    }
}

void Lcd_PutPixel(int x,int y,U8 rgb)
{
    U32 mask[4]={0x00ffffff,0xff00ffff,0xffff00ff,0xffffffff};

    framebuffer256[y][x/4]=(framebuffer256[y][x/4] & mask[x%4]) | ( (U32)rgb<<(3-x%4)*8 );
}

void Clcd_Init(void)
{
    //Now, the dithering function doesn't operate. So, Only 3 levels(100%,50%,0%) are
    supported.

    //(240x3)x320 color STN LCD(8-bit interface)
    //frame rate =75Hz
    //MCLK=40Mhz

    if(frameBuffer256==0)
    {
        framebuffer256=(unsigned int (*)[60])malloc(76800+16); //240x320
        framebuffer256=(unsigned int (*)[60])( ((unsigned)frameBuffer256+15)>>4)<<4 );
        //rNCACHBE0=( ( ( (unsigned)frameBuffer256+76800+16)>>12 )+1 )
        <<16) | ((unsigned)frameBuffer256>>12);
    }

    // 4bit single scan, VM=frame, WDLY=8clk, WLH=8clk CLKVAL=8->85.2Hz
    // 8-bit single scan, VM=each line, WDLY=6clk, WLH=6clk,
    // VCLK=2.3Mhz, CLKVAL=9 ->2.22Mhz
    rLCDCON1=(2<<5) | (1<<7) | (0x3<<8) | (0x3<<10) | ((8)<<12);

    // LIVEVAL=320-1, HOZVAL=(240x3)/8-1=89, LINEBLANK=10, MODESEL=color
    rLCDCON2=(319) | (89<<10) | (10<<19) | (0x3<<30);
    rLCSADDR1= ( ((unsigned int)frameBuffer256>>20)<<16 ) | ( (unsigned
int)frameBuffer256>>4 );
    rLCSADDR2= ( ((unsigned int)frameBuffer256+76800)>>4 )-1 | (35<<16);

    rREDLUT =0xfdb96420;
    rGREENLUT=0xfdb96420;
    rBLUELUT =0xfb40;

```

```
/*
rDP1_2 =0xa5a5;
rDP4_7H=0x00b5a5a5;
rDP4_7L=0xe;
rDP3_5 =0x5a5be;
rDP2_3 =0xd6b;
rDP3_4 =0x7dbe;
rDP5_7H=0x007be5db;
rDP5_7L=0xe;
rDP4_5 =0x7ebdf;
rDP6_7H=0x007fdfbf;
rDP6_7L=0xe;
*/
rLCDCON1=(2<<5)|(1<<7)|(0x2<<8)|(0x2<<10)|((8)<<12)|0x1;
}
```

**44BTEST : dma.h**

```
#ifndef __DMA_H__
#define __DMA_H__

void Test_Zdma0(void);
void Test_Zdma1(void);
//void Test_Zdma0Xdreq(void);

#endif /*__DMA_H__*/
```

**44BTEST : dma.c**

```
/*
*****
CAUTION: DMA operation is being done in cache-on state NOW.
         So, some read value may be not same with real memory value
         because of the cache. Users must fully consider the role of
         cache after DMA has been being operated. It is the best way
         using non-cacheable region in the memory area written by DMA.
*****
*/

#include <string.h>
#include "..\inc\44b.h"
#include "..\inc\44blib.h"
#include "..\inc\def.h"

void Zdma0Int(int srcAddr, int dstAddr, int length, int dw);
void Zdma1Int(int srcAddr, int dstAddr, int length, int dw);
void __irq Zdma0Done(void);
void __irq Zdma1Done(void);

volatile int zdma0Done, zdma1Done;

void Test_Zdma0(void)
{
    unsigned char *src, *dst;
    int i;
    unsigned int memSum;

    rINTMSK=BIT_GLOBAL;
    pISR_ZDMA0=(int)Zdma0Done;

    Uart_Printf("[ZDMA0 MEM2MEM Test]\n");

    dst=(unsigned char *)malloc(0x80000);
    src=(unsigned char *)malloc(0x80000);

    rNCACHE1=( ( ((unsigned)dst+0x100000)>>12) +1 )<<16 | ((unsigned)dst>>12);

    Uart_Printf("dst=%x,src=%x\n", (int)dst, (int)src);

    /* Copy by word */
    memSum=0;
    for(i=0; i<0x80000; i++)
        *(src+i)=0x1;
```



```

    Zdma0Int((int)src,(int)dst,0x80000,2); //word
    for(i=0;i<0x80000;i++)
        memSum+=(dst+i);
    Uart_Printf("memSum=%8x:",memSum);
    if(memSum==0x80000)Uart_Printf("O.K.\n");
    else Uart_Printf("ERROR!!!\n");

/* Copy by half-word */
    memSum=0;
    for(i=0;i<0x80000;i++)
        *(src+i)=2;
    Zdma0Int((int)src,(int)dst,0x80000,1); //half-word
    for(i=0;i<0x80000;i++)
        memSum+=(dst+i);
    Uart_Printf("memSum=%8x:",memSum);
    if(memSum==0x100000)Uart_Printf("O.K.\n");
    else Uart_Printf("ERROR!!!\n");

/* Copy by byte */
    memSum=0;
    for(i=0;i<0x80000;i++)
        *(src+i)=3;
    Zdma0Int((int)src,(int)dst,0x80000,0); //byte
    for(i=0;i<0x80000;i++)
        memSum+=(dst+i);
    Uart_Printf("memSum=%8x:",memSum);
    if(memSum==0x180000)Uart_Printf("O.K.\n");
    else Uart_Printf("ERROR!!!\n");

    free(src);
    free(dst);
}

void Test_Zdmal(void)
{
    unsigned char *src, *dst;
    int i;
    unsigned int memSum;

    rINTMSK=BIT_GLOBAL;
    pISR_ZDMA1=(int)ZdmalDone;

    Uart_Printf("[ZDMA1 MEM2MEM Test]\n");

    dst=(unsigned char *)malloc(0x80000);
    src=(unsigned char *)malloc(0x80000);

    rNCACHBE1=( ( ((unsigned)dst+0x100000)>>12) +1 )<<16 |((unsigned)dst>>12);

/* Copy by word */
    memSum=0;
    for(i=0;i<0x80000;i++)
        *(src+i)=1;
    Zdma1Int((int)src,(int)dst,0x80000,2); //word
    for(i=0;i<0x80000;i++)
        memSum+=(dst+i);
    Uart_Printf("memSum=%8x:",memSum);
    if(memSum==0x80000)Uart_Printf("O.K.\n");
    else Uart_Printf("ERROR!!!\n");

```

```

/* Copy by half-word */
memSum=0;
for(i=0;i<0x80000;i++)
    *(src+i)=2;
Zdma1Int((int)src,(int)dst,0x80000,1); //half-word
for(i=0;i<0x80000;i++)
    memSum+=*(dst+i);
Uart_Printf("memSum=%8x:",memSum);
if(memSum==0x100000)Uart_Printf("O.K.\n");
else Uart_Printf("ERROR!!!\n");

/* Copy by byte */
memSum=0;
for(i=0;i<0x80000;i++)
    *(src+i)=3;
Zdma1Int((int)src,(int)dst,0x80000,0); //byte
for(i=0;i<0x80000;i++)
    memSum+=*(dst+i);
Uart_Printf("memSum=%8x:",memSum);
if(memSum==0x180000)
    Uart_Printf("O.K.\n");
else
    Uart_Printf("ERROR!!!\n");

free(src);
free(dst);
}

void Zdma0Int(int srcAddr,int dstAddr,int length,int dw)
//returns the checksum
{
    int time;
    zdma0Done=0;
    rINTMSK=~(BIT_GLOBAL|BIT_ZDMA0);
    rZDISRC0=srcAddr|(dw<<30)|(1<<28); // inc
    rZDIDES0=dstAddr|( 2<<30)|(1<<28); // inc
    rZDICNT0=length |( 2<<28)|(1<<26)|(3<<22)|(1<<20);
    //whole,unit transfer,int@TC,enable DMA
    rZDCON0=0x1; // start!!!

    Timer_Start(3);//128us resolution
    while(zdma0Done==0);
    time=Timer_Stop();
    Uart_Printf("ZDMA0 %8x->%8x,%x:time=%f\n",srcAddr,dstAddr,length,time*128E-6);
    rINTMSK=BIT_GLOBAL;
}

void Zdma1Int(int srcAddr,int dstAddr,int length,int dw)
//returns the checksum
{
    int time;
    zdma1Done=0;
    rINTMSK=~(BIT_GLOBAL|BIT_ZDMA1);
    rZDISRC1=srcAddr|(dw<<30)|(1<<28); // inc
    rZDIDES1=dstAddr|( 2<<30)|(1<<28); // inc
    rZDICNT1=length |( 2<<28)|(1<<26)|(3<<22)|(1<<20);
    //whole,unit transfer,int@TC,enable DMA
    rZDCON1=0x1; // start!!!

```

```
    Timer_Start(3); //128us resolution
    while(zdmalDone==0);
    time=Timer_Stop();
    Uart_Printf("ZDMA1 %8x->%8x,%x:time=%f\n",srcAddr,dstAddr,length,time*128E-6);
    rINTMSK=BIT_GLOBAL;
}
```

```
void __irq Zdma0Done(void)
{
    rI_ISPC=BIT_ZDMA0;
    zdma0Done=1;
}
```

```
void __irq ZdmalDone(void)
{
    rI_ISPC=BIT_ZDMA1;    //clear pending
    //rI_ISPC;            //is needed only when cache=on & wrbuf=on & BSFRD=0
    zdmalDone=1;
}
```

**44BTEST : eint.h**

```
#ifndef __EINT_H__
#define __EINT_H__

void Test_Eint(void);

#endif /*__EINT_H__*/
```

**44BTEST : eint.c**

```
#include <string.h>
#include "..\inc\44b.h"
#include "..\inc\44blib.h"
#include "..\inc\eint.h"

void __irq Eint4567Isr(void);
void __irq Eint2Isr(void);

volatile char which_int=0;

void Test_Eint(void)
{
    unsigned int save_G, save_PG;
/*test
    save_G=rPCONG;
    save_PG=rPUPG;
    rPCONG=0xffff; //EINT7~0
    rPUPG=0x0;      //pull up enable

    Uart_Printf("1.Read test!!!\nPress EINT2 key and press any key\n");
    rINTMSK=BIT_GLOBAL;
    rI_ISPC=0x3fffffff;
    Uart_Getch();
    Uart_Printf("rINTPND=0x%x\n", rINTPND);
    rI_ISPC=BIT_EINT2;
    Uart_Printf("rINTPND=0x%x\n", rINTPND);
    while(1);
test*/

    Uart_Printf("[External Interrupt Test]\n");

    /***Vectored interrupt test***/
    rINTCON=0x1;
    /***Vectored interrupt test***/

    pISR_EINT4567=(int)Eint4567Isr;
    pISR_EINT2=(int)Eint2Isr;
    //    rINTMSK=~(BIT_GLOBAL|BIT_EINT4567);

    Uart_Printf("...Select the trigger:\n"
        " 1. Falling trigger\n"
        " 2. Rising trigger\n"
        " 3. Both Edge trigger\n"
        " 4. Low level trigger\n"
        " 5. High level trigger\n");
```

```

save_G=rPCONG;
save_PG=rPUPG;
rPCONG=0xffff; //EINT7~0
rPUPG=0x0;      //pull up enable
switch(Uart_Getch())
{
    case '1':
        rEXTINT=0x22222222; //Falling edge mode
        break;
    case '2':
        rEXTINT=0x44444444; //Rising edge mode
        break;
    case '3':
        rEXTINT=0x77777777; //Both edge mode
        break;
    case '4':
        rEXTINT=0x0;      //"0" level mode
        break;
    case '5':
        Uart_Printf("In this board EINT2 and EINT5 are pulled
up\n"); //rEXTINT=0x11111111; //"1" level mode
        which_int=10;
        break;
    default:
        rPCONG=save_G;
        rPUPG=save_PG;
        return;
}

// Uart_Printf("\nDEMO B/D push buttons may have glitch noise problem.\n");
Uart_Printf("Press the EINT buttons!!!\n");
rINTMSK=~(BIT_GLOBAL|BIT_EINT2|BIT_EINT4567);

while(!which_int);

switch(which_int)
{
    case 1:
        Uart_Printf("EINT4 had been occurred...\n");
        break;
    case 2:
        Uart_Printf("EINT5 had been occurred...\n");
        break;
    case 4:
        Uart_Printf("EINT6 had been occurred...\n");
        break;
    case 8:
        Uart_Printf("EINT7 had been occurred...\n");
        break;
    case 9:
        Uart_Printf("EINT2 had been occurred...\n");
    default :
        break;
}

rINTMSK=BIT_GLOBAL;
rPCONG=save_G;
rPUPG=save_PG;
which_int=0;

Uart_Printf("\nrINTCON=0x%x\n",rINTCON);
rINTCON=0x5;
}

```

```
void __irq Eint4567Isr(void)
{
    which_int=rEXTINPND;
    rEXTINPND=0xf;      //clear EXTINPND reg.
    rI_ISPC=BIT_EINT4567; //clear pending_bit
}
```

```
void __irq Eint2Isr(void)
{
    rI_ISPC=BIT_EINT2;   //clear pending_bit
    which_int=9;
}
```

**44BTEST : etc.h**

```
#ifndef __ETC_H__
#define __ETC_H__
void Test_Sram8(void);
void Etc(void);
//void Test_WaitPin(void);
#endif /*__ETC_H__*/
```

**44BTEST : etc.c**

```
#include <string.h>
#include "..\inc\44b.h"
#include "..\inc\44blib.h"
#include "..\inc\etc.h"
#include "..\inc\def.h"

/*
void Test_WaitPin(void)
{
    U32 readVal;
    Uart_Printf("nGCS1 is controlled by nWAIT pin.\n");
    Uart_Printf("nWAIT/PF0 pin is configured as nWAIT pin.\n");
    Uart_Printf("\nPush any key to exit!!!\n");

    // Work-around with nWAIT.
    // 8bit device ->8bit bus width, 16bit access
    // address0 is not connected to the device to cheat the device.
    rBWSCON=rBWSCON & ~(0x7<<4)|(0x5<<4); //nGCS1: nWAIT enable, 8bit
    rBANKCON1=0x200;
    rNCACHBE0=( (0x4000000>>12)<<16 )|(0x2000000>>12);
    while(Uart_GetKey()==0x0)
    {
        readVal=(*(volatile U16 *)0x2000000);
#ifdef __BIG_ENDIAN
        readVal=readVal&0xff;
#else
        readVal=readVal>>8;
#endif
        Uart_Printf("read_data=%x\n",readVal);
    }
}
*/
extern char Image$$RO$$Limit[];
extern char Image$$RO$$Base[];
extern char Image$$RW$$Limit[];
extern char Image$$RW$$Base[];
extern char Image$$ZI$$Limit[];
extern char Image$$ZI$$Base[];

void Etc(void)
{
    Uart_Printf("Image$$RO$$Base=%x\n",Image$$RO$$Base);
    Uart_Printf("Image$$RO$$Limit=%x\n",Image$$RO$$Limit);
    Uart_Printf("Image$$RW$$Base=%x\n",Image$$RW$$Base);
    Uart_Printf("Image$$RW$$Limit=%x\n",Image$$RW$$Limit);
    Uart_Printf("Image$$ZI$$Base=%x\n",Image$$ZI$$Base);
    Uart_Printf("Image$$ZI$$Limit=%x\n",Image$$ZI$$Limit);
}
```

**44BTEST : extdma.h**

```
#ifndef __EXTDMA_H__
#define __EXTDMA_H__

void Test_ZDma0Xdreq(void);

#endif /*__EXTDMA_H__*/
```

**44BTEST : extdma.c**

```
#include <string.h>
#include "..\inc\44b.h"
#include "..\inc\44blib.h"
#include "..\inc\def.h"

void _Zdma0XdreqHandshakeUnit8bit(void);
void _Zdma0XdreqHandshakeUnit16bit(void);
void _Zdma0XdreqHandshakeUnit32bit(void);
void _Zdma0XdreqHandshakeBlock16bit(void);
void _Zdma0XdreqWholeUnit16bit(void);
void _Zdma0XdreqWholeBlock16bit(void);
void _Zdma0XdreqHandshakeOntheflyRd16bit(void);
void _Zdma0XdreqHandshakeOntheflyWr16bit(void);

#define BUS8      (0)
#define BUS16     (1)
#define BUS32     (2)
#define ENWAIT    (1)
#define SRAMBE03  (1)

//PG3 = DMAMODE0
//PG4 = DMAMODE1
//PE4 = nDMASTART
//PE5 = D16_32ENABLE

#define SET_XDREQ16()      rPDATG=(rPDATG&~(3<<3))|(0x1<<3)
#define SET_XDREQ1()      rPDATG=(rPDATG&~(3<<3))|(0x2<<3)
#define START_XDREQ()     {rPDATE=(rPDATE&~(1<<4))|(0x0<<4);} \
                          rPDATE=(rPDATE&~(1<<4))|(0x1<<4);}

//DMAMODE[1:0]:      01b= one time request
//                  10b= 16 time requests

#define B2_Tacs      (0x0) //0clk
#define B2_Tcos      (0x0) //0clk
//#define B2_Tacc      (0x2) //3clk
#define B2_Tacc      (0x6) //10clk
#define B2_Tcoh      (0x0) //0clk
#define B2_Tah       (0x0) //0clk
#define B2_Tacp      (0x0) //2clk
#define B2_PMC       (0x0) //no page mode
```



```

volatile int isZdma0Done;

//Verilog code for data bus pins of the EPM7256A
//assign data[3:0] = (!nGCS && !nOE ) ? wdata:4'bz;
//assign data[23:4]= (!nGCS && !nOE && !nUDE) ? addr[19:0]:20'bz;

void __irq IsrZdma0Done(void)
{
    rI_ISPC=BIT_ZDMA0;
    isZdma0Done=1;
}

void Test_ZDma0Xdreq(void)
{
    U32 savePCONC;
    rINTMSK=BIT_GLOBAL;
    pISR_ZDMA0=(U32)IsrZdma0Done;
    rINTMSK=~(BIT_GLOBAL|BIT_ZDMA0);

    Cache_Flush(); //to clear the cache memory.

    savePCONC=rPCONC;
    rPCONC=0xaaaaaaaa; //DATA[16:31] is selected.

    rPCONF=rPCONF&(~(0xf<<6)|(0xf<<6)); //PF3=nXDACK0,PF4=nXDREQ0
    rPCONE=rPCONE&(~(3)|(3)); //PE0=CLKOUT

    rPDATG=rPDATG&(3<<3);
    rPCONG=rPCONG&~(0xf<<6)|(5<<6); //PG3,4=OUTPUT
    rPDATE=rPDATE&~(1<<4)|(1<<4); //PE4=H,PE5=H
    rPCONE=rPCONE&~(0xf<<8)|(5<<8); //PE4,PE5=OUTPUT

    rPDATE=rPDATE&~(1<<5)|(0<<5); //nUDE=L(PE5=L) to enable EPM7256A d[23:4]

    Uart_Printf("Test Menu for ZDMA0 nXDREQ/nXDACK\n");
    Uart_Printf("1: Handshake/unit/8bit src:GCS2(8) dst:SDRAM(16)\n");
    Uart_Printf("2: Handshake/unit/16bit src:GCS2(16) dst:SDRAM(16)\n");
    Uart_Printf("3: Handshake/unit/32bit src:GCS2(32) dst:SDRAM(16)\n");
    Uart_Printf("4: Handshake/block/16bit src:GCS2(16) dst:SDRAM(16)\n");
    Uart_Printf("5: Whole/unit/16bit src:GCS0(8) dst:SDRAM(16)\n");
    Uart_Printf("6: Whole/block/16bit src:GCS0(8) dst:SDRAM(16)\n");
    Uart_Printf("7: Handshake/of_rd/16bit src:GCS2(16)\n");
    Uart_Printf("8: Handshake/of_wr/16bit dst:GCS2(16)\n");
    Uart_Printf("Select the item?\n");
    switch(Uart_Getch())
    {
    case '1':
        _Zdma0XdreqHandshakeUnit8bit();
        break;
    case '2':
        _Zdma0XdreqHandshakeUnit16bit();
        break;
    case '3':
        _Zdma0XdreqHandshakeUnit32bit();
        break;
    }
}

```

```

    case '4':
        _Zdma0XdreqHandshakeBlock16bit();
        break;
    case '5':
        _Zdma0XdreqWholeUnit16bit();
        break;
    case '6':
        _Zdma0XdreqWholeBlock16bit();
        break;
    case '7':
        _Zdma0XdreqHandshakeOntheflyRd16bit();
        break;
    case '8':
        _Zdma0XdreqHandshakeOntheflyWr16bit();
        break;
    default:
        break;
}
rPCONC=savePCONC;
}

void _Zdma0XdreqHandshakeUnit8bit(void)
{
    int i;
    static U8 bufDst[16];

    rBWSCON=rBWSCON&(~0xf00)|(BUS8<<8);

    rBANKCON2=((B2_Tacs<<13)+(B2_Tcos<<11)+(B2_Tacc<<8)+(B2_Tcoh<<6)+(B2_Tah<<4)+(B2_Tacp<<2)+(
    B2_PMC));

    Uart_Printf("[ZDMA0,HandShake,unit,8bit Test,B2->SDRAM]\n");
    Uart_Printf("NOTE:Ignore bit[3:0]!!\n");

    for(i=0;i<16;i++){bufDst[i]=0;}

    isZdma0Done=0;
    rNCACHBE0=( ( ((unsigned)bufDst+16*1)>>12) +1 )<<16 |((unsigned)bufDst>>12);

    rINTMSK=~(BIT_GLOBAL|BIT_ZDMA0);
    rZDISRC0=(U32)0x4001230|(0<<30)|(1<<28); // byte,inc
    rZDIDES0=(U32)bufDst|((U32)2<<30)|(1<<28); // normal,inc
    rZDICNT0=16|(0x0<<30)|(0x0<<28)|(0x1<<26)|(0x2<<24)|(0x3<<22)|(0x1<<21)|(0x1<<20);
    //nXDREQ0,handshake,unit,terminal_int,auto-reload,enable DMA,
    rZDCON0=0x0; // nXDREQ0 enable,CMD=no_command.

    SET_XDREQ16();
    START_XDREQ();

    while(isZdma0Done==0);

    for(i=0;i<16;i++)Uart_Printf( "%2x,",*((U8 *)0x4001230+i) );
    Uart_Printf("\n");
    for(i=0;i<16;i++)Uart_Printf( "%2x,",bufDst[i]);
    Uart_Printf("\n");

    Cache_Flush();
    rNCACHBE0=0;
}

```

```

void _Zdma0XdreqHandshakeUnit16bit(void)
{
    int i;
    static U16 bufDst[16];
    static U16 bufSrc[16];

    rBWSCON=rBWSCON&(~0xf00)|(BUS16<<8);

    rBANKCON2=((B2_Tacs<<13)+(B2_Tcos<<11)+(B2_Tacc<<8)+(B2_Tcoh<<6)+(B2_Tah<<4)+(B2_Tacp<<2)+(
    B2_PMC));

    Uart_Printf("[ZDMA0,HandShake,unit,16bit Test,SDRAM->SDRAM]\n");

    for(i=0;i<16;i++){bufSrc[i]=i;bufDst[i]=0;}

    isZdma0Done=0;
    rNCACHBE0=(( ((unsigned)bufDst+16*4)>>12) +1 )<<16 |(((unsigned)bufDst>>12);

    rINTMSK=~(BIT_GLOBAL|BIT_ZDMA0);
    rZDISRC0=(U32)bufSrc|(1<<30)|(1<<28); // half-word,inc
    rZDIDES0=(U32)bufDst|((U32)2<<30)|(1<<28); // normal,inc
    rZDICNT0=32|(0x0<<30)|(0x0<<28)|(0x1<<26)|(0x2<<24)|(0x3<<22)|(0x1<<21)|(0x1<<20);
    //nXDREQ0,handshake,unit,terminal_int,auto-reload,enable DMA,
    rZDCON0=0x0; // nXDREQ0 enable,CMD=no_command.

    SET_XDREQ16();
    START_XDREQ();

    while(isZdma0Done==0);

    for(i=0;i<16;i++)Uart_Printf("%x,",*((U16 *)bufSrc+i));
    Uart_Printf("\n");
    for(i=0;i<16;i++)Uart_Printf("%x,",*((U16 *)bufDst+i));
    Uart_Printf("\n");

    Cache_Flush();
    rNCACHBE0=0;
}

void _Zdma0XdreqHandshakeUnit32bit(void)
{
    int i;
    static U32 bufDst[16];

    rBWSCON=rBWSCON&(~0xf00)|(BUS32<<8);

    rBANKCON2=((B2_Tacs<<13)+(B2_Tcos<<11)+(B2_Tacc<<8)+(B2_Tcoh<<6)+(B2_Tah<<4)+(B2_Tacp<<2)+(
    B2_PMC));

    Uart_Printf("[ZDMA0,HandShake,unit,32bit Test,B2->SDRAM]\n");
    Uart_Printf("NOTE:Ignore bit[3:0],bit[31:24]!!\n");

    for(i=0;i<16;i++){bufDst[i]=0;}

    isZdma0Done=0;
    rNCACHBE0=(( ((unsigned)bufDst+16*4)>>12) +1 )<<16 |(((unsigned)bufDst>>12);

    rINTMSK=~(BIT_GLOBAL|BIT_ZDMA0);
    rZDISRC0=(U32)0x4001230|((U32)2<<30)|(1<<28); // word,inc

```

```

rZDIDES0=(U32)bufDst|((U32)2<<30)|(1<<28); // normal,inc
rZDICNT0=64|(0x0<<30)|(0x0<<28)|(0x1<<26)|(0x2<<24)|(0x3<<22)|(0x1<<21)|(0x1<<20);
//nXDREQ0,handshake,unit,terminal_int,auto-reload,enable DMA,
rZDCON0=0x0; // nXDREQ0 enable,CMD=no_command.

SET_XDREQ16();
START_XDREQ();

while(isZdma0Done==0);
for(i=0;i<16;i++)
{
    Uart_Printf("%8x,",*((U32 *)0x4001230+i));
    if(i%8==7)Uart_Printf("\n");
}

for(i=0;i<16;i++)
{
    Uart_Printf("%8x,",bufDst[i]);
    if(i%8==7)Uart_Printf("\n");
}

Cache_Flush();
rNCACHBE0=0;
}

void _Zdma0XdreqHandshakeBlock16bit(void)
{
    int i;
    static U16 bufDst[128];

    rBWSCON=rBWSCON&(~0xf00)|(BUS16<<8);

rBANKCON2=((B2_Tacs<<13)+(B2_Tcos<<11)+(B2_Tacc<<8)+(B2_Tcoh<<6)+(B2_Tah<<4)+(B2_Tacp<<2)+(
B2_PMC));

    Uart_Printf("[ZDMA0,HandShake,block,16bit Test,B2->SDRAM]\n");
    Uart_Printf("NOTE:Ignore bit[3:0]\n");

    for(i=0;i<128;i++){bufDst[i]=0;}

    isZdma0Done=0;
    rNCACHBE0=( ( ((unsigned)bufDst+128*2)>>12) +1 )<<16 |(((unsigned)bufDst>>12);

    rINTMSK=~(BIT_GLOBAL|BIT_ZDMA0);
    rZDISRC0=(U32)0x4001200|((U32)2<<30)|(1<<28); // word,inc
    rZDIDES0=(U32)bufDst|((U32)2<<30)|(1<<28); // normal,inc
    rZDICNT0=256|(0x0<<30)|(0x0<<28)|(0x2<<26)|(0x2<<24)|(0x3<<22)|(0x1<<21)|(0x1<<20);
    //nXDREQ0,handshake,block,terminal_int,auto-reload,enable DMA,
    rZDCON0=0x0; // nXDREQ0 enable,CMD=no_command.

    SET_XDREQ16();
    START_XDREQ();

    while(isZdma0Done==0)
    {
        Uart_Printf("rZDCCNT0=%x\r",rZDCCNT0);
    }

    Uart_Printf("Correct data:200x,202x,204x,....,2fex\n");

```

```

/*
    for(i=0;i<128;i++)
    {
        Uart_Printf("%4x,",*((U16 *)0x4001200+i));
        if(i%16==15)Uart_Printf("\n");
    }
*/

    for(i=0;i<128;i++)
    {
        Uart_Printf("%4x,",bufDst[i]);
        if(i%8==7)Uart_Printf("\n");
    }

    Cache_Flush();
    rNCACHBE0=0;

}

void _Zdma0XdreqWholeUnit16bit(void)
{
    int i;
    static U16 bufDst[16];

    rBWSCON=rBWSCON&(~0xf00)|(BUS16<<8);

    rBANKCON2=((B2_Tacs<<13)+(B2_Tcos<<11)+(B2_Tacc<<8)+(B2_Tcoh<<6)+(B2_Tah<<4)+(B2_Tacp<<2)+(
    B2_PMC));

    Uart_Printf("[ZDMA0,Whole,unit,16bit Test,B2->SDRAM]\n");
    Uart_Printf("NOTE:Ignore bit[3:0]\n");

    for(i=0;i<16;i++){bufDst[i]=0;}

    isZdma0Done=0;
    rNCACHBE0=( ( ((unsigned)bufDst+16*2)>>12) +1 )<<16 |(((unsigned)bufDst>>12);

    rINTMSK=~(BIT_GLOBAL|BIT_ZDMA0);
    rZDISRC0=(U32)0x4001200|(((U32)1<<30)|(1<<28); // half-word,inc
    rZDIDES0=(U32)bufDst|(((U32)2<<30)|(1<<28); // normal,inc
    rZDICNT0=32|(0x0<<30)|(0x2<<28)|(0x1<<26)|(0x2<<24)|(0x3<<22)|(0x1<<21)|(0x1<<20);
    //nXDREQ0,whole,unit,terminal_int,auto-reload,enable DMA,
    rZDCON0=0x0; // nXDREQ0 enable,CMD=no_command.

    SET_XDREQ1();
    START_XDREQ();

    while(isZdma0Done==0);

    Uart_Printf("Correct data:200x,202x,204x,...,21ex\n");

    for(i=0;i<16;i++)Uart_Printf("%x,",*((U16 *)bufDst+i));
    Uart_Printf("\n");

    Cache_Flush();
    rNCACHBE0=0;
}

```

```

void _Zdma0XdreqWholeBlock16bit(void)
{
    int i;
    static U16 bufDst[128];

    rBWSCON=rBWSCON&(~0xf00)|(BUS16<<8);

    rBANKCON2=((B2_Tacs<<13)+(B2_Tcos<<11)+(B2_Tacc<<8)+(B2_Tcoh<<6)+(B2_Tah<<4)+(B2_Tacp<<2)+(
    B2_PMC));

    Uart_Printf("[ZDMA0,whole,block,16bit Test,B2->SDRAM]\n");
    Uart_Printf("NOTE:Ignore bit[3:0]\n");

    for(i=0;i<128;i++){bufDst[i]=0;}

    isZdma0Done=0;
    rNCACHBE0=(( ((unsigned)bufDst+128*2)>>12) +1 )<<16 |(((unsigned)bufDst>>12);

    rINTMSK=~(BIT_GLOBAL|BIT_ZDMA0);
    rZDISRC0=(U32)0x4001200|((U32)2<<30)|(1<<28); // word,inc
    rZDIDES0=(U32)bufDst|((U32)2<<30)|(1<<28); // normal,inc
    rZDICNT0=256|(0x0<<30)|(0x2<<28)|(0x2<<26)|(0x2<<24)|(0x3<<22)|(0x1<<21)|(0x1<<20);
    //nXDREQ0,whole,block,terminal_int,auto-reload,enable DMA,
    rZDCON0=0x0; // nXDREQ0 enable,CMD=no_command.

    SET_XDREQ1();
    START_XDREQ();

    while(isZdma0Done==0)
    {
        Uart_Printf("rZDCCNT0=%x\r",rZDCCNT0);
    }

    Uart_Printf("Correct data:200x,202x,204x,....,2fex\n");

    for(i=0;i<128;i++)
    {
        Uart_Printf("%4x,",bufDst[i]);
        if(i%8==7)Uart_Printf("\n");
    }

    Cache_Flush();
    rNCACHBE0=0;
}

```

```

void _Zdma0XdreqHandshakeOntheflyRd16bit(void)
{
    rBWSCON=rBWSCON&(~0xf00)|(BUS16<<16);

    rBANKCON2=((B2_Tacs<<13)+(B2_Tcos<<11)+(B2_Tacc<<8)+(B2_Tcoh<<6)+(B2_Tah<<4)+(B2_Tacp<<2)+(
    B2_PMC));

    Uart_Printf("[ZDMA0,HandShake,read_OTF,16bit Test,B2(src)]\n");
    Uart_Printf("NOTE:Ignore bit[3:0]\n");
    Uart_Printf("Check the data using a logic analyzer.\n");

    isZdma0Done=0;
    rNCACHBE0=(( ((unsigned)0x6000000)>>12)<<16 |(((unsigned)0x4000000>>12);

```

```

rINTMSK=~(BIT_GLOBAL|BIT_ZDMA0);
rZDISRC0=(U32)0x4001230|((U32)2<<30)|(1<<28); // word,inc
rZDIDES0=0|((U32)2<<30)|(1<<28); // normal,inc
rZDICNT0=32|(0x0<<30)|(0x0<<28)|(0x3<<26)|(0x2<<24)|(0x3<<22)|(0x1<<21)|(0x1<<20);
//nXDREQ0,handshake,read_OTF,terminal_int,auto-reload,enable DMA,
rZDCON0=0x0; // nXDREQ0 enable,CMD=no_command.

SET_XDREQ16();
START_XDREQ();

while(isZdma0Done==0)
{
    Uart_Printf("rZDCCNT0=%x\r",rZDCCNT0);
}

Cache_Flush();
rNCACHBE0=0;
}

void _Zdma0XdreqHandshakeOntheflyWr16bit(void)
{
    rBWSCON=rBWSCON&(~0xf00)|(BUS16<<16);

rBANKCON2=((B2_Tacs<<13)+(B2_Tcos<<11)+(B2_Tacc<<8)+(B2_Tcoh<<6)+(B2_Tah<<4)+(B2_Tacp<<2)+(
B2_PMC));

    Uart_Printf("[ZDMA0,HandShake,write_OTF,16bit Test,B2(dst)]\n");
    Uart_Printf("Check your data using a logic analyzer.\n");

    isZdma0Done=0;
    rNCACHBE0=((((unsigned)0x6000000)>>12)<<16)|(((unsigned)0x4000000)>>12);

    rINTMSK=~(BIT_GLOBAL|BIT_ZDMA0);
    rZDISRC0=0x0|((U32)2<<30)|(1<<28); // word,inc
    rZDIDES0=(U32)0x4001230|((U32)2<<30)|(1<<28); // normal,inc
    rZDICNT0=32|(0x0<<30)|(0x0<<28)|(0x3<<26)|(0x3<<24)|(0x3<<22)|(0x0<<21)|(0x1<<20);
    //nXDREQ0,handshake,write_OTF,terminal_int,auto-reload,enable DMA,
    rZDCON0=0x0; // nXDREQ0 enable,CMD=no_command.

    SET_XDREQ16();
    START_XDREQ();

    while(isZdma0Done==0)
    {
        Uart_Printf("rZDCCNT0=%x\r",rZDCCNT0);
    }

    Cache_Flush();
    rNCACHBE0=0;
}

```

**44BTEST : flash.h**

```
#ifndef __FLASH_H__
#define __FLASH_H__

void ProgramFlash(void);

#endif /*__FLASH_H__*/
```

**44BTEST : flash.c**

```
#include <string.h>
#include "..\inc\option.h"
#include "..\inc\def.h"
#include "..\inc\44b.h"
#include "..\inc\44blib.h"
#include "..\inc\am29f800.h"

int DownloadData(void);
U32 downloadAddress;
U32 downloadProgramSize;

void *flashType[][2]=
{
    (void *)ProgramAM29F800,      "AM29LV800BB",
    0,0
};

void ProgramFlash(void)
{
    int i=0;
    Uart_Printf("\n*** NOR Flash Memory writer ver 0.3 ***\n\n");
    downloadAddress=(unsigned)malloc(0x400000);
    if(downloadAddress==0)return;
    rNCACHBE0=( (0x2000000>>12)<<16 )|(0>>12); //flash area must be non-cachable area.
    rSYSCFG=rSYSCFG&(~0x8); //write buffer has to be off for proper timing.

    while(1)
    {
        //display menu
        Uart_Printf("%c: %s", 'a'+i, flashType[i][1]);
        i++;
        if((int)(flashType[i][0])==0){Uart_Printf("\n");break;}
        if((i%4)==0)Uart_Printf("\n");
    }

    Uart_Printf("Select the type of a flash memory? ");
    i=Uart_Getch()-'a';
    Uart_Printf("\n");
    if( i<0 || (i>=(sizeof(flashType)/8)) )
        return;
    if(!DownloadData())return;

    ( (void (*)(void))(flashType[i][0]) )();

    free((void *)downloadAddress);
    rNCACHBE0=0x0;
}
```



```

int DownloadData(void)
{
    int i,tmp;
    U16 checkSum=0,dnCS;
    U32 fileSize=10;
    U8 *downPt;

    downPt=(U8 *)downloadAddress;

    Uart_Printf("downloadAddress=%x\n",downloadAddress);

    Uart_Printf("Download the plain binary file(.BHC) to be written\n");
    Uart_Printf(".BHC file format: <n+6>(4)+(n)+CS(2)\n");
    Uart_Printf("To transmit .BHC file      : wkocm2 xxx.BHC /1 /g /d:1\n");
    Uart_Printf("Or, to transmit .BIN file: wkocm2 xxx.BIN /1 /d:1\n");
    Uart_Printf("Download methods: COM:8Bit,NP,1STOP\n");

    Uart_Printf("\nSTATUS:");
    rINTMSK=~BIT_GLOBAL;

    tmp=RdURXH0(); //To remove overrun error state.

    i=0;
    while(i<fileSize)
    {
        while(!(rUTRSTAT0&0x1));
        *(downPt+i)=RdURXH0();
        if(i==3)
        {
            fileSize=((U8 *) (downloadAddress+0))+
                ((U8 *) (downloadAddress+1))<<8)+
                ((U8 *) (downloadAddress+2))<<16)+
                ((U8 *) (downloadAddress+3))<<24);
        }

        if((i%1000)==0)WrUTXH0('#');
        i++;
    }

    downloadProgramSize=fileSize-6;

    for(i=4;i<(fileSize-2);i++)
    {
        checkSum+=*((U8 *) (i+downloadAddress));
    }

    dnCS=((U8 *) (downloadAddress+fileSize-2))+
        ((U8 *) (downloadAddress+fileSize-1))<<8);

    if(checkSum!=dnCS)
    {
        Uart_Printf("Checksum Error!!! MEM:%x DN:%x\n",checkSum,dnCS);
        return 0;
    }

    Uart_Printf("\nDownload O.K.\n");
    return 1;
}

```

**44BTEST : glib.h**

```

#ifndef __GLIB_H__
#define __GLIB_H__

void Glib_Init(int depth);

void Glib_Line(int x1,int y1,int x2,int y2,int color);
void Glib_Rectangle(int x1,int y1,int x2,int y2,int color);
void Glib_FilledRectangle(int x1,int y1,int x2,int y2,int color);
void Glib_ClearScr(U8 c);

void _PutPixelMono(U32 x,U32 y,U8 c);
void _PutPixelG4(U32 x,U32 y,U8 c);
void _PutPixelG16(U32 x,U32 y,U8 c);
void _PutPixelColor(U32 x,U32 y,U8 c);

extern void (*PutPixel)(U32,U32,U8);

#endif //__GLIB_H__

```

**44BTEST : glib.c**

```

#include "..\inc\def.h"
#include "..\inc\lcdlib.h"
#include "..\inc\glib.h"
#include "..\inc\lcd.h"

void (*PutPixel)(U32,U32,U8);

void Glib_Init(int depth)
{
    switch(depth)
    {
        case 1:
            PutPixel=_PutPixelMono;
            break;
        case 4:
            PutPixel=_PutPixelG4;
            break;
        case 16:
            PutPixel=_PutPixelG16;
            break;
        case 256:
            PutPixel=_PutPixelColor;
            break;
        default: break;
    }
}

void _PutPixelMono(U32 x,U32 y,U8 c)
{
    if(x<SCR_XSIZE && y<SCR_YSIZE)
        frameBuffer1[(y)][(x)/32]=( frameBuffer1[(y)][(x)/32] & ~(0x80000000>>((x)%32)*1) )
        | ( (c)<< ((32-1-((x)%32))*1) );
}

```

```

void _PutPixelG4(U32 x,U32 y,U8 c)
{
    if(x<SCR_XSIZE && y<SCR_YSIZE)
        frameBuffer4[(y)][(x)/16]=( frameBuffer4[(y)][x/16] & ~(0xc0000000>>((x)%16)*2) )
        | ( (c)<<((16-1-((x)%16))*2) );
}

void _PutPixelG16(U32 x,U32 y,U8 c)
{
    if(x<SCR_XSIZE && y<SCR_YSIZE)
        frameBuffer16[(y)][(x)/8]=( frameBuffer16[(y)][x/8] & ~(0xf0000000>>((x)%8)*4) )
        | ( (c)<<((8-1-((x)%8))*4) );
}

void _PutPixelColor(U32 x,U32 y,U8 c)
{
    if(x<SCR_XSIZE && y<SCR_YSIZE)
        frameBuffer256[(y)][(x)/4]=( frameBuffer256[(y)][x/4] & ~(0xff000000>>((x)%4)*8) )
        | ( (c)<<((4-1-((x)%4))*8) );
}

void Glib_Rectangle(int x1,int y1,int x2,int y2,int color)
{
    Glib_Line(x1,y1,x2,y1,color);
    Glib_Line(x2,y1,x2,y2,color);
    Glib_Line(x1,y2,x2,y2,color);
    Glib_Line(x1,y1,x1,y2,color);
}

void Glib_FilledRectangle(int x1,int y1,int x2,int y2,int color)
{
    int i;

    for(i=y1;i<=y2;i++)
        Glib_Line(x1,i,x2,i,color);
}

// LCD display is flipped vertically
// But, think the algorithm by mathematics point.
//      3I2
//      4 I 1
//      --+--    <-8 octants  mathematical coordinate
//      5 I 8
//      6I7

void Glib_Line(int x1,int y1,int x2,int y2,int color)
{
    int dx,dy,e;
    dx=x2-x1;
    dy=y2-y1;

```

```

if(dx>=0)
{
    if(dy >= 0) // dy>=0
    {
        if(dx>=dy) // 1/8 octant
        {
            e=dy-dx/2;
            while(x1<=x2)
            {
                PutPixel(x1,y1,color);
                if(e>0){y1+=1;e-=dx;}
                x1+=1;
                e+=dy;
            }
        }
        else // 2/8 octant
        {
            e=dx-dy/2;
            while(y1<=y2)
            {
                PutPixel(x1,y1,color);
                if(e>0){x1+=1;e-=dy;}
                y1+=1;
                e+=dx;
            }
        }
    }
    else // dy<0
    {
        dy=-dy; // dy=abs(dy)

        if(dx>=dy) // 8/8 octant
        {
            e=dy-dx/2;
            while(x1<=x2)
            {
                PutPixel(x1,y1,color);
                if(e>0){y1-=1;e-=dx;}
                x1+=1;
                e+=dy;
            }
        }
        else // 7/8 octant
        {
            e=dx-dy/2;
            while(y1>=y2)
            {
                PutPixel(x1,y1,color);
                if(e>0){x1+=1;e-=dy;}
                y1-=1;
                e+=dx;
            }
        }
    }
}
else //dx<0
{
    dx=-dx; //dx=abs(dx)
    if(dy >= 0) // dy>=0
    {
        if(dx>=dy) // 4/8 octant
        {
            e=dy-dx/2;

```

```

        while(x1>=x2)
        {
            PutPixel(x1,y1,color);
            if(e>0){y1+=1;e-=dx;}
            x1-=1;
            e+=dy;
        }
    }
    else // 3/8 octant
    {
        e=dx-dy/2;
        while(y1<=y2)
        {
            PutPixel(x1,y1,color);
            if(e>0){x1-=1;e-=dy;}
            y1+=1;
            e+=dx;
        }
    }
}
else // dy<0
{
    dy=-dy; // dy=abs(dy)

    if(dx>=dy) // 5/8 octant
    {
        e=dy-dx/2;
        while(x1>=x2)
        {
            PutPixel(x1,y1,color);
            if(e>0){y1-=1;e-=dx;}
            x1-=1;
            e+=dy;
        }
    }
    else // 6/8 octant
    {
        e=dx-dy/2;
        while(y1>=y2)
        {
            PutPixel(x1,y1,color);
            if(e>0){x1-=1;e-=dy;}
            y1-=1;
            e+=dx;
        }
    }
}
}
}
}

```

```
void Glib_ClearScr(U8 c)
{
    //Very inefficient function.

    int i,j;

    for(j=0;j<SCR_YSIZE;j++)
        for(i=0;i<SCR_XSIZE;i++)
            PutPixel(i,j,c);
}
```

```
/*
void Lcd_MonoFig(U8 *fig)
{
    int i,j,k;
    int xSize,ySize;
    xSize=((U8 *)fig+0)+*((U8 *)fig+1)*0x100;
    ySize=((U8 *)fig+2)+*((U8 *)fig+3)*0x100;
    Uart_Printf("xsize=%d, ysize=%d\n",xSize,ySize);
    fig+=4;

    xSize=xSize/32;
    for(i=ySize-1;i>=0;i--)
        for(j=0;j<xSize;j++)
        {
            framebuffer1[i][j]=~((*(fig+0)<<24)+(*(fig+1)<<16)+(*(fig+2)<<8)+*(fig+3));
            fig+=4;
        }
}
*/
```

**44BTEST : idle.h**

```

#ifndef __IDLE_H__
#define __IDLE_H__

void Test_SLIdleMode20(void);
void Test_SLIdleMode(void);
void Test_IdleMode(void);
void Test_IdleModeHard(void);

#endif /*__IDLE_H__*/

```

**44BTEST : idle.c**

```

#include "..\inc\44b.h"
#include "..\inc\44blib.h"
#include "..\inc\def.h"
#include "..\inc\idle.h"
#include "..\inc\rtc.h"

void SLIdleMode(void); //160x240

int debug;

int t0cnt,t1cnt;

void __irq Eint45Int(void)
{
    rEXTINPND=0xf;          //clear EXTINPND reg.
    rI_ISPC=BIT_EINT4567;
    Uart_Printf("EINT45 ISR is occurred for wake-up from IDLE mode.\n");
}

void __irq Timer0Int(void)
{
    int i;
    //for(i=0;i<10000;i++);
    rI_ISPC=BIT_TIMER0;
    for(i=0;i<100;i++); //why???
    t0cnt++;
}

void __irq Timer1Int(void)
{
    rI_ISPC=BIT_TIMER1;
    t1cnt++;
}

void __irq AlarmInt(void)
{
    rI_ISPC=BIT_RTC;
    Uart_Printf("ALARM ISR is occurred for wake-up from IDLE mode.\n");
}

```

```

void __irq SlEint45Int(void)
{
    rEXTINPND=0xf;          //clear EXTINPND reg.
    rI_ISPC=BIT_EINT4567;
    debug=1;
    //????
    //why does not this interrupt occur in SL-IDLE mode?
    //????
}

void __irq SlAlarmInt(void)
{
    //If you have to use the internal peripherals,
    //you must configure CLKCON,CLKSLOW,PLL

    rI_ISPC=BIT_RTC;
    debug=2;
}

/*****
 *   IDLE mode test   *
 *****/

void Test_IdleMode(void)
{
    int i;
    int extintMode;
    Uart_Printf("[IDLE Mode Test]\n");
    Uart_Printf("Check the current consumption. Push the buttons to exit IDLE mode.\n");
    Uart_Printf("After 10 seconds, S3C44B0X will wake up by RTC alarm interrupt.\n");
    Uart_Printf("S3C44B0X will also wake up by EINT4/5.\n");
    Uart_Printf("1.L-LEVEL  2.H-LEVEL  3.F-EDGE  4.R-EDGE  5.B-EDGE\n");
    Uart_Printf("Select the external interrupt type. Press the number!!!\n");
    extintMode=Uart_Getch();

    Uart_TxEmpty(0);    //Wait until UART0 Tx buffer empty.

    rPCONG=0xf00;    //PG2=EINT2

    switch(extintMode)
    {
    case '1':
        rEXTINT=0x0; //low
        break;
    case '2':
        rEXTINT=0x11111111; //high
        break;
    case '3':
        rEXTINT=0x22222222; //falling
        break;
    case '4':
        rEXTINT=0x44444444; //rising
        break;
    case '5':
        rEXTINT=0x66666666; //both edge
        break;
    default:
        break;
    }

    Rtc_Init();
}

```



```

rRTCCON = 0x01;    // R/W enable, 1/32768, Normal(merge), No reset

rALMYEAR=TESTYEAR2 ;
rALMMON =TESTMONTH2;
rALMDAY =TESTDAY2  ;
rALMHOUR=TESTHOUR2 ;
rALMMIN =TESTMIN2  ;
rALMSEC =TESTSEC2+9;

rRTCALM=0x7f;

rI_ISPC=BIT_EINT4567|BIT_RTC;    //to clear the previous pending status.
Uart_Printf("rINTPND=%x\n",rINTPND);

pISR_EINT4567=(U32)Eint45Int;
pISR_RTC=(U32)AlarmInt;
rINTMSK=~(BIT_GLOBAL|BIT_EINT4567|BIT_RTC);
for(i=0;i<2;i++);    //wait until the pended interrupt is executed.

rCLKCON=0x7ff8|0x4;    //enter IDLE mode.
//Uart_Getch();
for(i=0;i<10;i++);    //wait until S3C44B0X enters IDLE mode.

// wait EINT[7:0] interrupt or alarm wake-up

rCLKCON=0x7ff8;
//turn-off IDLE bit on rCLKCON to synchronize rCLKCON with the real mode.

Uart_Printf("Return to Normal Mode.\n");

rINTMSK=BIT_GLOBAL;
}

```

```

void Test_IdleModeHard(void)
{
    int i,j;

    Uart_Printf("[IDLE Mode Test with Timer0,1 10000times]\n");
    Uart_Printf("S3C44B0X will also wake up by EINT4/5.\n");
    Uart_TxEmpty(0);    //Wait until UART0 Tx buffer empty.

    rPCONG=0xf00;    //PG2=EINT2
    rEXTINT=rEXTINT&~(7<<8)|(2<<8);    //falling

    pISR_TIMER0=(U32)Timer0Int;
    pISR_TIMER1=(U32)Timer1Int;
    pISR_EINT4567=(U32)Eint45Int;

    rTCFG0=0x00010110;    //PRESC01,23,45= 1
    rTCFG1=0x00000000;    //TIMER0,1,2,3,4,5= 1/2
    rTCNTB0=65535;
    rTCNTB1=2570;
    //rTCON=0xa0a;    //T1=MU,ITV,T0=MU,ITV
    //rTCON=0x909;    //Start T0,T1.
    rTCON=0x00a;
    rTCON=0x009;

    rINTMSK=~(BIT_GLOBAL|BIT_EINT4567|BIT_TIMER0|BIT_TIMER1);
    //rINTMSK=~(BIT_GLOBAL|BIT_EINT2);

```

```

//The two timer will test the IDLE mode hard.

for(i=0;i<10000;i++)
{
    rCLKCON=0x7ff8|0x4; //enter IDLE mode.

    for(j=0;j<10;j++); //wait until KS32C41100 enters IDLE mode.
    //wake up from normal mode

    rCLKCON=0x7ff8;
    //turn-off IDLE bit on rCLKCON to synchronize rCLKCON with the real mode.
    Uart_Printf(".");
}

rTCN=0x0; //timer off
rINTMSK=BIT_GLOBAL;
}

```

```

/*****
 * SL_IDLE mode test *
 *****/

#define MVAL_USED    (0)
#define MVAL          (13)
#define L248          (8)
#define CLKVAL_SL    (38) // 60Mhz, fr=100Hz (CLKVAL=38.6)

```

```

#define M5D(n) ((n) & 0x1fffff)

unsigned int (*_frameBuffer4)[10];

void LcdInit_4Gray160x240(void);
void Display_4Gray160x240(void);

```

```

void Test_SLIdleMode20(void)
{
    int i;
    LcdInit_4Gray160x240();
    Display_4Gray160x240();
    Delay(1000); //wait more than 1 frame time.
    for(i=0;i<20;i++)SLIdleMode();
}

```

```

void Test_SLIdleMode(void)
{
    LcdInit_4Gray160x240();
    Display_4Gray160x240();
    Delay(1000); //wait more than 1 frame time.
    SLIdleMode();
}

```

```

void SLIdleMode(void) //160x240

```

```

{
    int i;
    int saveDramcon;
    debug=0;
    Uart_Printf("[SL_IDLE MODE TEST for 160x240]\n");
    Uart_Printf("After 10 seconds, S3C44B0X will wake up by RTC alarm interrupt.\n");
    Uart_Printf("S3C44B0X can wake up by EINT4/5.\n");
    Uart_TxEmpty(0);

    rPCONG=0xf00;    //PG2=EINT2
    rEXTINT=0x22222222;    //falling

    //pISR_EINT2=(U32)SlEint2Int;
    //pISR_RTC=(U32)SlAlarmInt;
    //rINTMSK=~(BIT_GLOBAL|BIT_EINT2|BIT_RTC);

    //If you want to use interrupt generation,the ISR should be on ROM/SRAM
    //(DRAM can't be allowed because of DRAM self-refresh)

    Rtc_Init();
    rRTCCON = 0x01;    // R/W enable, 1/32768, Normal(merge), No reset

    rALMYEAR=TESTYEAR2 ;
    rALMMON =TESTMONTH2;
    rALMDAY =TESTDAY2 ;
    rALMHOUR=TESTHOUR2 ;
    rALMMIN =TESTMIN2 ;
    rALMSEC =TESTSEC2+9;
    //rALMSEC =TESTSEC2+1;

    rRTCALM=0x7f;    //To test alarm wake-up.

    rADCCON|=0x20;    //ADC power down mode

    //
    // The I/O ports have to be configured properly to reduce STOP mode current.
    //

    //
    //          LINECNT  4n+m th line
    // 1st line 239      4n+1th
    // 2nd line 238      4n+2th
    // 3rd line 237      4n+3th
    // 4th line 236      4n  th
    // ....
    //236th line 4       4n  th
    //237th line 3       4n+1th
    //238th line 2       4n+2th
    //239th line 1       4n+3th
    //240th line 0       4n  th

    while((rLCDCON1>>22)!=9);
    while((rLCDCON1>>22)!=8);
    // To enter self-refresh mode, the VCLK has to be 'L' after 4n-th line is displayed
    // completely,
    // So,the self-refresh command has to be issued at 4n th line.
    // VCLK will be 'L' from 4n+1th line.

    //for(i=0;i<10;i++);
    rLCDCON3=0x1;    //Enter self-refresh mode.

```

```

while((rLCDCON1>>22)!=0);

//Because the SLOW mode is used, the LCDCON1,2 should be changed.
rLCDCON1=(0)|(1<<5)|(MVAL_USED<<7)|(0x0<<8)|(0x0<<10)|(1/*CLKVAL*/<<12);
// disable, 4B_SINGL_SCAN, WDLY=2clk, WLH=2clk
rLCDCON2=(239+L248)|(15/*39*/<<10)|(1<<21); //LINEBLANK=1
rCLKSLOW=2|(1<<4)|(1<<5); //SLOWVAL=2, Fout=Fin/(2x2), PLL off.
//DRAM refresh may not be done because of slow MCLK. but, the abnormal period is very
short.
rLCDCON1=(1)|(1<<5)|(MVAL_USED<<7)|(0x0<<8)|(0x0<<10)|(1<<12);

saveDramcon=rREFRESH;
rREFRESH=(2017)|(0<<22)|(1<<23);
//15,6us@2Mhz, tchr,trc,trp=min, refresh enable.
// DRAM refresh may not be done every 15.6uS
// because of slow MCLK(1Mhz) and long memory access cycle.

EnterPWDN(0x46); //enters SL_IDLE mode. rCLKCON=0x46
//DRAM/SDRAM self-refresh is executed in EnterPWDN()
//NOTE:
//Any special registers setting will not accepted because CPU is not normal mode.

rCLKCON=0x7ff8;

rCLKSLOW=2|(1<<4)|(0<<5); //SLOWVAL=2, Fout=Fin/(2x2), PLL on.
for(i=0;i<2048;i++); //Wait PLL lock time.

while((rLCDCON1>>22)!=1);
while((rLCDCON1>>22)!=0);

//Because the SLOW mode is exited, the LCDCON1,2 should be changed.
rLCDCON1=(0)|(1<<5)|(MVAL_USED<<7)|(0x3<<8)|(0x3<<10)|(CLKVAL_SL<<12);
// disable, 4B_SINGL_SCAN, WDLY=8clk, WLH=8clk,
rLCDCON2=(239+L248)|(39<<10)|(10<<21); //HOZVAL=39, LINEBLANK=1
rCLKSLOW=2; //SLOWVAL=2, Fout=Fp1lo, PLL on.

rLCDCON3=0x0; //LCD self refresh mode off.

rLCDCON1=(1)|(1<<5)|(MVAL_USED<<7)|(0x3<<8)|(0x3<<10)|(CLKVAL_SL<<12);

rREFRESH=saveDramcon;

while((rLCDCON1>>22)!=1);
while((rLCDCON1>>22)!=0); //Display 1 more frame before exiting SL_IDLE mode.
//why? no reason. it may be better.

//while((rLCDCON1>>22)!=9);
//while((rLCDCON1>>22)!=8); //to catch the exit time for SL_IDLE mode.
//rLCDCON3=0x0; //LCD self refresh mode off.

//for(i=0;i<10;i++);

rADCCON&=~(0x20);

rEXTINPND=0xf; //Clear EXTINPND register

Uart_Printf("debug=%d\n", debug);
Uart_Printf("I have exited LCD SELFREF mode.\n");
}

```

```

void LcdInit_4Gray160x240(void)
{
    if((U32)_frameBuffer4==0)
    {
        _frameBuffer4=(unsigned int (*)(10))malloc(10*4*240);
    }

    rLCDCON1=(0)|(1<<5)|(MVAL_USED<<7)|(0x3<<8)|(0x3<<10)|(CLKVAL_SL<<12);
    // disable, 4B_SINGL_SCAN, WDLY=8clk, WLH=8clk, CLKVAL=?
    rLCDCON2=(239+L248)|(39<<10)|(10<<21);
    //LINEBLANK=10 (without any calculation)
    rLCDSADDR1= (0x1<<27) | ( ((U32)_frameBuffer4>>22)<<21 ) |
M5D((U32)_frameBuffer4>>1);
    // 4-gray, LCDBANK, LCDBASEU
    rLCDSADDR2= M5D(((U32)_frameBuffer4+((160/4)*(240+L248)))>>1) | (MVAL<<21);
    rLCDSADDR3= (160/8) | ( 0<<9 );
    //No virtual screen.

    //The following value has to be changed for better display.
    //Select 4 levels among 16 gray levels.

    rBLUELUT=0xfa40;
    rDITHMODE=0x0;
    rDP1_2 =0xa5a5;
    rDP4_7 =0xba5da65;
    rDP3_5 =0xa5a5f;
    rDP2_3 =0xd6b;
    rDP5_7 =0xeb7b5ed;
    rDP3_4 =0x7dbe;
    rDP4_5 =0x7ebdf;
    rDP6_7 =0x7fdfbfe;

    rLCDCON1=(1)|(1<<5)|(MVAL_USED<<7)|(0x3<<8)|(0x3<<10)|(CLKVAL_SL<<12);
    // enable, 4B_SINGL_SCAN, WDLY=8clk, WLH=8clk, CLKVAL=?
}

```

```

void Display_4Gray160x240(void)
{
    int i,j;

    for(j=0;j<100;j++)
        for(i=2;i<10;i++)
        {
            _frameBuffer4[j][i]=0x55aa55aa;
        }

    for(j=0;j<100;j++)
    {
        _frameBuffer4[j][9]=0x5555ffff;
    }

    for(j=100;j<240;j++)
        for(i=2;i<10;i++)
            _frameBuffer4[j][i]=0x0;

    for(i=2;i<10;i++)
        _frameBuffer4[100][i]=0xffffffff;
    for(i=2;i<10;i++)
        _frameBuffer4[0][i]=0xffffffff;
}

```

**44BTEST : iic.h**

```

#ifndef __IIC_H__
#define __IIC_H__

void Test_Iic(void);
void InitI341(void);
void Wr24C040(U32 slvAddr,U32 addr,U8 data);
void Rd24C040(U32 slvAddr,U32 addr,U8 *data);

#endif /*__IIC_H__*/

```

**44BTEST : iic.c**

```

#include <string.h>
#include "..\inc\44b.h"
#include "..\inc\44blib.h"
#include "..\inc\def.h"
#include "..\inc\iic.h"

#define WRDATA          (1)
#define POLLACK         (2)
#define RDDATA          (3)
#define SETRDADDR       (4)

#define IICBUFSIZE 0x20

U8 _iicData[IICBUFSIZE];
volatile int _iicDataCount;
volatile int _iicStatus;
volatile int _iicMode;
int _iicPt;

void __irq IicInt(void);

void Test_Iic(void)
{
    unsigned int i,j,save_F,save_PF;
    static U8 data[256];

    Uart_Printf("[IIC Test using KS24C080]\n");

    save_F=rPCONF;
    save_PF=rPUPF;
    rPCONF |=0xa;    //PF0:IICSCL, PF1:IICSDA
    rPUPF |=0x3;     //pull-up disable

    pISR_IIC=(unsigned)IicInt;
    rINTMSK=~(BIT_GLOBAL|BIT_IIC);

    rIICCON=(1<<7)|(0<<6)|(1<<5)|(0xf);
    //Enable interrupt, IICCLK=MCLK/16, Enable ACK
    //40Mhz/16/(15+1) = 257Khz
    rIICADD=0x10;    // S3C44B0X slave address

    rIICSTAT=0x10;

    Uart_Printf("Write test data into KS24C080\n");
}

```

```

    for(i=0;i<256;i++)
        Wr24C040(0xa0,(U8)i,i);
    for(i=0;i<256;i++)
        data[i]=0;

    Uart_Printf("Read test data from KS24C080\n");
    for(i=0;i<256;i++)
        Rd24C040(0xa0,(U8)i,&(data[i]));

    for(i=0;i<16;i++)
    {
        for(j=0;j<16;j++)
            Uart_Printf("%2x ",data[i*16+j]);
        Uart_Printf("\n");
    }

    rPCONF=save_F;
    rPUPF=save_PF;
}

void Wr24C040(U32 slvAddr,U32 addr,U8 data)
{
    _iicMode=WRDATA;
    _iicPt=0;
    _iicData[0]=(U8)addr;
    _iicData[1]=data;
    _iicDataCount=2;

    rIICDS=slvAddr;//0xa0
    rIICSTAT=0xf0; //MasTx,Start
    //Clearing the pending bit isn't needed because the pending bit has been cleared.
    while(_iicDataCount!=-1);

    _iicMode=POLLACK;

    while(1)
    {
        rIICDS=slvAddr;
        _iicStatus=0x100;
        rIICSTAT=0xf0; //MasTx,Start
        rIICCON=0xaf; //resumes IIC operation.
        while(_iicStatus==0x100);
        if(!(_iicStatus&0x1))
            break; // when ACK is received
    }
    rIICSTAT=0xd0; //stop MasTx condition
    rIICCON=0xaf; //resumes IIC operation.
    Delay(1); //wait until stop condition is in effect.

    //write is completed.
}

void Rd24C040(U32 slvAddr,U32 addr,U8 *data)
{
    _iicMode=SETRDADDR;
    _iicPt=0;
    _iicData[0]=(U8)addr;
    _iicDataCount=1;

    rIICDS=slvAddr;
    rIICSTAT=0xf0; //MasTx,Start

```

```

//Clearing the pending bit isn't needed because the pending bit has been cleared.
while(_iicDataCount!=-1);

_iicMode=RDDATA;
_iicPt=0;
_iicDataCount=1;

rIICDS=slvAddr;
rIICSTAT=0xb0; //MasRx,Start
rIICCON=0xaf; //resumes IIC operation.
while(_iicDataCount!=-1);

*data=_iicData[1];
}

```

```

void __irq IicInt(void)
{
    U32 iicSt,i;
    rI_ISPC=BIT_IIC;

    iicSt=rIICSTAT;
    if(iicSt&0x8){} // when bus arbitration is failed.
    if(iicSt&0x4){} // when a slave address is matched with IICADD
    if(iicSt&0x2){} // when a slave address is 0000000b
    if(iicSt&0x1){} // when ACK isn't received

    switch(_iicMode)
    {
        case POLLACK:
            _iicStatus=iicSt;
            break;

        case RDDATA:
            if((_iicDataCount--)==0)
            {
                _iicData[_iicPt++]=rIICDS;

                rIICSTAT=0x90; //stop MasRx condition
                rIICCON=0xaf; //resumes IIC operation.
                Delay(1); //wait until stop condition is in effect.
                //too long time...
                //The pending bit will not be set after issuing stop condition.
                break;
            }
            _iicData[_iicPt++]=rIICDS;
            //The last data has to be read with no ack.
            if((_iicDataCount)==0)
            {
                rIICCON=0x2f; //resumes IIC operation with NOACK.
            }
            else
            {
                rIICCON=0xaf; //resumes IIC operation with ACK
            }
            break;

        case WRDATA:
            if((_iicDataCount--)==0)
            {
                rIICSTAT=0xd0; //stop MasTx condition
                rIICCON=0xaf; //resumes IIC operation.
                Delay(1); //wait until stop condition is in effect.
                //The pending bit will not be set after issuing stop condition.
                break;
            }
    }
}

```



```
        rIICDS=_iicData[_iicPt++]; // _iicData[0] has dummy.
        for(i=0;i<10;i++);          //for setup time until rising edge of IICSCL
        rIICCON=0xaf;               //resumes IIC operation.
        break;

    case SETRDADDR:
        //Uart_Printf("[S%d]",_iicDataCount);
        if((_iicDataCount--)==0)
        {
            break; //IIC operation is stopped because of IICCON[4]
        }
        rIICDS=_iicData[_iicPt++];
        for(i=0;i<10;i++); //for setup time until rising edge of IICSCL
        rIICCON=0xaf;      //resumes IIC operation.
        break;

    default:
        break;
}
}
```

**44BTEST : iis.h**

```

#ifndef __IIS_H__
#define __IIS_H__

#ifdef __cplusplus
extern "C" {
#endif

//void PlayWav(U32 sample,S16 *lBuf,S16 *rBuf,U32 freq);
void PlayWav(void);
void Init1341(void);
void Test_Iis(void);
void Play1341(void);
void Stop1341(void);

//#define FS2205KHZ
#define FS441KHZ

#ifdef __cplusplus
}
#endif

#endif /*__IIS_H__*/

```

**44BTEST : iis.c**

```

#include "..\inc\44b.h"
#include "..\inc\44blib.h"
#include "..\inc\def.h"
#include "..\inc\iis.h"

void _WrL3Addr(U8 data);
void _WrL3Data(U8 data,int halt);
void __irq BDMA0_Done(void);
void __irq RxInt(void);

#define L3D (0x200)
#define L3M (0x40)
#define L3C (0x80)

#define TESTSIZE 2000000//1000//100
//for test
//static short int Buf[TESTSIZE];
//for down
unsigned char *Buf,*_temp;

volatile unsigned int size=0;

void Test_Iis(void)
{
    unsigned int i,j,save_A,save_C,save_E,save_G,save_PC,save_PE,save_PG;

    ChangePllValue(0x69,0x17,0x0);      //MCLK=45.1584MHz <-- 5.6448MHz*8
    Uart_Init(45200000,115200);
    Uart_Printf("[IIS test]\n");

    save_A=rPCONA;//L3DATA
    save_C=rPCONC;//IIS port

```

```

    save_E=rPCONE;//CODEC clk
    save_G=rPCONG;//L3CLK,L3MOD
    save_PC=rPUPC;
    save_PE=rPUPE;
    save_PG=rPUPG;

#if (BUSWIDTH==32)
    Uart_Printf("IIS test should be configured 16bit data bus\n");
    return;
#else //BUSWIDTH=16
    rPCONC |=0xff;
    rPUPC |= 0xf;
#endif

    rPCONE=(rPCONE&0xffff)+(2<<16); //PE:CODECLK
    pISR_BDMA0=(unsigned)BDMA0_Done;

    rINTMSK=~(BIT_GLOBAL|BIT_BDMA0|BIT_URXD0);

    /***** Tx DATA Initialize *****/
//for test
/*    rNCACHBE0= ((int)Buf>>12) + ( (((int)Buf>>12) +0x100)<<16 );
    Uart_Printf("rNCACHBE0=0x%x\n",rNCACHBE0);
    for(i=0;i<TESTSIZE/2 ;i+=2)
    {
        Buf[i]=i*30;
        Buf[i+1]=i*30;
    }
    for(i=TESTSIZE/2;i<TESTSIZE;i+=2)
    {
        Buf[i]=(TESTSIZE-i)*30;
        Buf[i+1]=(TESTSIZE-i)*30;
    }
*/
////////////////////////////////////
//for down
pISR_URXD0=(unsigned)RxInt;
Buf=(unsigned char *)malloc(TESTSIZE);
rNCACHBE0= ((int)Buf>>12) + ( (((int)Buf>>12) +0x200)<<16 );//non-cachable 2MB
Uart_Printf("rNCACHBE0=0x%x\n",rNCACHBE0);
_temp=Buf;

Uart_Printf("Download the PCM(no ADPCM) wave file by wkcom2(with header)!!\n");

while(((unsigned int)_temp-(unsigned int)Buf)<4)
{
    Led_Display(0xf);
    Delay(1500);
    Led_Display(0x0);
    Delay(1500);
}
size=(Buf) | *(Buf+1)<<8 | *(Buf+2)<<16 | *(Buf+3)<<24;
Uart_Printf("\nNow, Downloading... [FILESIZE:%7d(      0)",size);
while(((unsigned int)_temp-(unsigned int)Buf)<size)
Uart_Printf("\b\b\b\b\b\b\b\b%7d", (unsigned int)_temp-(unsigned int)Buf);
Uart_Printf("\b\b\b\b\b\b\b\b%7d\n", (unsigned int)_temp-(unsigned int)Buf);

rINTMSK |=BIT_URXD0;

size=(Buf+0x2c) | *(Buf+0x2d)<<8 | *(Buf+0x2e)<<16 | *(Buf+0x2f)<<24;
size=(size>>1)<<1;
Uart_Printf("sample size=0x%x\n",size/2);

```

```

    Uart_Printf("\n[ ]Now play the wave file\n");

    //////////////////////////////////////
    Initl341();

    //    for(i=4;i<300;i++)
    //        Uart_Printf("%02x,",*(Buf+i));

    /***** BDMA0 Initialize *****/
    //for down
    rBDISRC0=(1<<30)+(1<<28)+(int)(Buf+0x30); //Half word,inc,Buf
    //    rBDISRC0=(1<<30)+(1<<28)+(int)(Buf+4); //Half word,inc,Buf
    //for test
    //    rBDISRC0=(1<<30)+(1<<28)+(int)(Buf); //Half word,inc,Buf
    rBDIDES0=(1<<30)+(3<<28)+((int)IISFIF); //M2IO,fix,IISFIF
    //for down
    rBDICNT0=(1<<30)+(1<<26)+(3<<22)+(1<<21)+(1<<20)+size;
    //for test
    //    rBDICNT0=(1<<30)+(1<<26)+(3<<22)+(1<<21)+(1<<20)+TESTSIZE*2;
    //iis,reserve,done_int,auto-reload/start,DMA enable,COUNT
    rBDCON0 = 0x0<<2;

    /***** IIS Initialize *****/
    rIISCON=0x22; //Tx DMA enable,Rx idle,prescaler enable
    rIISMOD=0x89; //Master,Tx,L-ch=low,iis,16bit ch.,codeclk=256fs,lrck=32fs
    //    rIISPSR=0x11; //Prescaler_A/B enable, value=1
    rIISPSR=0x33; //Prescaler_A/B enable, value=3
    rIISFCON=0xa00; //Tx/Rx DMA,Tx/Rx FIFO --> start piling....

    Uart_Printf("Push any key to exit!!!\n");
    /***** IIS Tx Start *****/
    rIISCON |=0x1;
    while(!Uart_GetKey());

    /***** IIS Tx Stop *****/
    rIISCON=0x0; //IIS stop
    rBDICNT0=0x0; //BDMA stop
    //for down
    free(Buf);
    Cache_Flush();
    rNCACHBE0=0x0;
    size=0;

    rPCONA=save_A;
    rPCONC=save_C;
    rPCONE=save_E;
    rPCONG=save_G;
    rPUPC=save_PC;
    rPUPE=save_PE;
    rPUPG=save_PG;

    rINTMSK=BIT_GLOBAL;
    ChangePllValue(0x34,0x3,0x1); //Fin=10MHz, Fout=60MHz
    Uart_Init(0,115200);
}

void Initl341(void)
{
    /***** Port Initialize *****/
    rPCONA = 0x1ff; //PA9(out):L3D
    rPCONG = 0x5000; //PG6:L3M, PG7:L3C
    rPUPG |= 0xc0; //disable(pull-up)

```

```

    rPDATG = L3M|L3C;      //L3M=H(start condition)
                          //L3C=H(start condition)

    /***** L3 Interface *****/
    _WrL3Addr(0x14+2);     //status (000101xx+10)
#ifdef FS441KHZ
    _WrL3Data(0x60,0); //0,1,10,000,0 reset,256fs,no DCfilter,iis
#else
    _WrL3Data(0x40,0); //0,1,00,000,0 reset,512fs,no DCfilter,iis
#endif

    _WrL3Addr(0x14+2); //status (000101xx+10)
#ifdef FS441KHZ
    _WrL3Data(0x20,0); //0,0,10,000,0 no reset,256fs,no DCfilter,iis
#else
    _WrL3Data(0x00,0); //0,0,00,000,0 no reset,512fs,no DCfilter,iis
#endif

    _WrL3Addr(0x14+2); //status (000101xx+10)
    _WrL3Data(0x83,0);
    //1,0,0,0,0,0,11 OGS=0,IGS=0,ADC_NI,DAC_NI,sngl speed,AonDon
}

void _WrL3Addr(U8 data)
{
    U32 vPdata = 0x0;      //L3D=L
    U32 vPdatg = 0x0;      //L3M=L(in address mode)/L3C=L
    S32 i,j;

    rPDATG = vPdatg;      //L3M=L
    rPDATG |= L3C;      //L3C=H

    for(j=0;j<4;j++);      //tsu(L3) > 190ns

    //PA9:L3D PG6:L3M PG7:L3C
    for(i=0;i<8;i++)
    {
        if(data&0x1)//if data bit is 'H'
        {
            rPDATG = vPdatg;      //L3C=L
            rPDATA = L3D;      //L3D=H
            for(j=0;j<4;j++);      //tcy(L3) > 500ns
            rPDATG = L3C;      //L3C=H
            rPDATA = L3D;      //L3D=H
            for(j=0;j<4;j++);      //tcy(L3) > 500ns
        }
        else      //if data bit is 'L'
        {
            rPDATG=vPdatg; //L3C=L
            rPDATA=vPdata; //L3D=L
            for(j=0;j<4;j++);      //tcy(L3) > 500ns
            rPDATG=L3C;      //L3C=H
            rPDATA=vPdata; //L3D=L
            for(j=0;j<4;j++);      //tcy(L3) > 500ns
        }
        data >>=1;
    }
    rPDATG=L3C|L3M; //L3M=H,L3C=H
}

```

```

void _WrL3Data(U8 data,int halt)
{
    U32 vPdata = 0x0;    //L3D=L
    U32 vPdatg = 0x0;    //L3M/L3C=L
    S32 i,j;
    if(halt)
    {
        rPDATG=L3C;      //L3C=H(while tstp, L3 interface halt condition)
        for(j=0;j<4;j++); //tstp(L3) > 190ns
    }
    rPDATG=L3C|L3M;      //L3M=H(in data transfer mode)
    for(j=0;j<4;j++);    //tsu(L3)D > 190ns

    //PA9:L3MODE PG6:L3DATA PG7:L3CLOCK
    for(i=0;i<8;i++)
    {
        if(data&0x1)      //if data bit is 'H'
        {
            rPDATG=L3M;    //L3C=L
            rPDATA=L3D;    //L3D=H
            for(j=0;j<4;j++); //tcy(L3) > 500ns
            rPDATG=L3C|L3M; //L3C=H,L3D=H
            rPDATA=L3D;
            for(j=0;j<4;j++); //tcy(L3) > 500ns
        }
        else              //if data bit is 'L'
        {
            rPDATG=L3M;    //L3C=L
            rPDATA=vPdatg; //L3D=L
            for(j=0;j<4;j++); //tcy(L3) > 500ns
            rPDATG=L3C|L3M; //L3C=H
            rPDATA=vPdata; //L3D=L
            for(j=0;j<4;j++); //tcy(L3) > 500ns
        }
        data>>=1;
    }
    rPDATG=L3C|L3M; //L3M=H,L3C=H
}

void __irq BDMA0_Done(void)
{
    rI_ISPC=BIT_BDMA0;    //clear pending bit
    WrUTXH0('.');
}

void __irq RxInt(void)
{
    rI_ISPC=BIT_URXD0 ;    //clear pending bits
    *_temp++=RdURXH0();
}

```

**44BTEST : lcd.h**

```

#ifndef __LCD_H__
#define __LCD_H__

void MoveViewPort(int depth);
void Test_LcdMono(void);
void Test_LcdG4(void);
void Test_LcdG16(void);
void Test_LcdColor(void);

#endif /*__LCD_H__*/

```

**44BTEST : lcd.c**

```

#include <string.h>
#include "..\inc\def.h"
#include "..\inc\44b.h"
#include "..\inc\44blib.h"
#include "..\inc\lcdlib.h"
#include "..\inc\glib.h"
#include "..\inc\lcd.h"

void Test_LcdMono(void);
void Test_LcdG4(void);
void Test_LcdG16(void);
void Test_LcdColor(void);

void Test_LcdMono(void)
{
    int i,j;
    Lcd_Init(MODE_MONO);
    Glib_Init(MODE_MONO);
    Uart_Printf("[Mono(1bit/1pixel) LCD Test]: Press Any Key!\n");

    Glib_ClearScr(0);
    for(j=0;j<LCD_YSIZE;j+=16)
        for(i=0;i<LCD_XSIZE;i+=16)
            Glib_FilledRectangle(i,j,i+15,j+15,((j+i)/16)%2);
    Uart_Printf("Mono test 1. Press any key!\n");
    Uart_Getch();

    Glib_ClearScr(0);
    Glib_FilledRectangle(160,0,319,239,1);
    Uart_Printf("Mono test 2. Press any key!\n");
    Uart_Getch();

    Glib_ClearScr(0);
    Glib_Rectangle(0,0,319,239,1);    // #0
    Glib_Line(0,0,319,239,1);        // 00
    Glib_Line(0,239,319,0,1);

    Glib_Rectangle(0+320,0,319+320,239,1);    // 0#
    Glib_Line(0+320,0,319+320,239,1);        // 00
    Glib_Line(0+320,239,319+320,0,1);
    Glib_FilledRectangle(50+320,50,269+320,189,1);

```

```

Glib_Rectangle(0,0+240,319,239+240,1);    // 00
Glib_Line(0,0+240,319,239+240,1);          // #0
Glib_Line(0,239+240,319,0+240,1);
Glib_FilledRectangle(50,50+240,269,189+240,1);

Glib_Rectangle(0+320,0+240,319+320,239+240,1);    // 00
Glib_Line(0+320,0+240,319+320,239+240,1);          // 0#
Glib_Line(0+320,239+240,319+320,0+240,1);
Glib_Rectangle(50+320,50+240,269+320,189+240,1);

Uart_Printf("Virtual Screen Test(Mono). Press any key[ijkm\\r]!\\n");
MoveViewPort(MODE_MONO);

Lcd_MoveViewPort(0,0,MODE_MONO);
}

```

```

void Test_LcdG4(void)
{
    int i,j,k;

    Lcd_Init(MODE_G4);
    Glib_Init(MODE_G4);

    Uart_Printf("[4gray(2bit/1pixel) LCD Test]: Press Any Key!\\n");

    Glib_ClearScr(0);

    j=0;
    for(i=0;i<320;i+=80)
        Glib_FilledRectangle(0+i,0,79+i,239,j++);
    Uart_Printf("4 gray mode test 1. Press any key!\\n");
    Uart_Getch();

    Glib_ClearScr(0);
    j=0;
    for(i=0;i<320;i+=80)
    {
        Glib_FilledRectangle(0+i,0,79+i,119,j);
        Glib_FilledRectangle(0+i,120,79+i,239,3-j);
        j++;
    }
    Uart_Printf("4 gray mode test 2. Press any key!\\n");
    Uart_Getch();

    Glib_ClearScr(0);
    j=0;
    for(i=0;i<240;i+=60)
    {
        Glib_FilledRectangle(i,i,i+59,i+59,j);
        j++;
    }
    Uart_Printf("4 gray mode test 3. Press any key!\\n");
    Uart_Getch();
}

```



```

Glib_ClearScr(0);
k=0;
for(i=160;i<480;i+=80)
{
    for(j=120;j<360;j+=60)
    {
        Glib_FilledRectangle(i,j,i+79,j+59,k%4);
        k++;
    }
    k+=2;;
}

// #0
// 00
Glib_Rectangle(0,0,319,239,3);
Glib_Line(0,0,319,239,3);
Glib_Line(0,239,319,0,3);

// 0#
// 00
Glib_Rectangle(0+320,0,319+320,239,3);
Glib_Line(0+320,0,319+320,239,3);
Glib_Line(0+320,239,319+320,0,3);

// 00
// #0
Glib_Rectangle(0,0+240,319,239+240,3);
Glib_Line(0,0+240,319,239+240,3);
Glib_Line(0,239+240,319,0+240,3);

// 00
// 0#
Glib_Line(0+320,0+240,319+320,239+240,3);
Glib_Line(0+320,239+240,319+320,0+240,3);
Glib_Rectangle(50+320,50+240,269+320,189+240,3);

Uart_Printf("Virtual Screen Test(4 gray). Press any key[ijkm\\r]!\\n");
MoveViewPort(MODE_G4);

Lcd_MoveViewPort(0,0,MODE_G4);
}

void Test_LcdGl6(void)
{
    int i,j,k;

    Lcd_Init(MODE_G16);
    Glib_Init(MODE_G16);

    Uart_Printf("[16 gray(4bit/1pixel) LCD Test]: Press Any Key!\\n");

    Glib_ClearScr(0);
    j=0;
    for(i=0;i<320;i+=20)
        Glib_FilledRectangle(0+i,0,19+i,239,j++);
    Uart_Printf("16 gray mode test 1. Press any key!\\n");
    Uart_Getch();

    Uart_Printf("16 gray color viewing. Press any key 16 times.\\n");

```

```

j=0;
Glib_ClearScr(0);
for(i=0;i<16;i++)
{
    Glib_FilledRectangle(0,0,319,239,i);
    Uart_Printf("Color num=%d\n",i);
    Uart_Getch();
}

Glib_ClearScr(0);
j=0;
for(i=0;i<320;i+=20)
{
    Glib_FilledRectangle(0+i,0,19+i,119,j);
    Glib_FilledRectangle(0+i,120,19+i,239,15-j);
    j++;
}
Uart_Printf("16 gray mode test 2. Press any key!\n");
Uart_Getch();

Glib_ClearScr(0);
j=0;
for(i=0;i<240;i+=30)
{
    Glib_FilledRectangle(i,i,i+29,i+29,j);
    Glib_FilledRectangle(i+110,i,i+29+110,i+29,j+8);
    j++;
}

Uart_Printf("4 gray mode test 3. Press any key!\n");
Uart_Getch();

Glib_ClearScr(0);

k=0;
for(i=160;i<480;i+=40)
{
    for(j=120;j<360;j+=30)
    {
        Glib_FilledRectangle(i,j,i+39,j+29,k%16);
        k++;
    }
}

// #0
// 00
Glib_Rectangle(0,0,319,239,15);
Glib_Line(0,0,319,239,15);
Glib_Line(0,239,319,0,15);

// 0#
// 00
Glib_Rectangle(0+320,0,319+320,239,15);
Glib_Line(0+320,0,319+320,239,15);
Glib_Line(0+320,239,319+320,0,15);

// 00
// #0
Glib_Rectangle(0,0+240,319,239+240,15);
Glib_Line(0,0+240,319,239+240,15);
Glib_Line(0,239+240,319,0+240,15);

```

```

// 00
// 0#
Glib_Rectangle(0+320,0+240,319+320,239+240,15);
Glib_Line(0+320,0+240,319+320,239+240,15);
Glib_Line(0+320,239+240,319+320,0+240,15);
Glib_Rectangle(50+320,50+240,269+320,189+240,15);

Uart_Printf("Virtual Screen Test(16 gray). Press any key[ijkm\\r]!\n");
MoveViewPort(MODE_G16);

Lcd_MoveViewPort(0,0,MODE_G16);
}

void Test_LcdColor(void)
{
    int i,j,k;

    rPDATE=rPDATE&~(1<<5)|(1<<5);      //GPE5=H
    rPCONE=rPCONE&~(3<<10)|(1<<10);      //GPE5=output
    rPCONC=rPCONC&~(0xff<<8)|(0xff<<8); //GPC[4:7] => VD[7:4]

    Uart_Printf("[ (240x3)x320 COLOR STN LCD TEST]\n");

    Uart_Printf("      R:0      ...      7 \n");
    Uart_Printf("G:0  B0:1  B0:1  B0:1 \n");
    Uart_Printf("G:.   2:3   2:3   2:3 \n");
    Uart_Printf("G:.   B0:1  B0:1  B0:1 \n");
    Uart_Printf("G:.   2:3   2:3   2:3 \n");
    Uart_Printf("G:.   B0:1  B0:1  B0:1 \n");
    Uart_Printf("G:7    2:3   2:3   2:3 \n");

    Lcd_Init(MODE_COLOR);
    Glib_Init(MODE_COLOR);

    Glib_ClearScr(0);
    for(j=0;j<320;j++)
        for(i=0;i<240;i++)
        {
            //RRRGGBB
            PutPixel( i,j,((i/30)<<5)+((j/40)<<2)+((i/15)&0x1)+(((j/20)&0x1)<<1) );
        }
    Uart_Printf("256 color mode test 1. Press any key!\n");
    Uart_Getch();

    Glib_ClearScr(0);
    k=0;
    for(i=160;i<480;i+=20)
    {
        for(j=120;j<360;j+=15)
        {
            Glib_FilledRectangle(i,j,i+19,j+14,k);
            k++;
        }
    }

    // #0
    // 00
    Glib_Rectangle(0,0,319,239,255);
    Glib_Line(0,0,319,239,255);

```

```

Glib_Line(0,239,319,0,255);

// 0#
// 00
Glib_Rectangle(0+320,0,319+320,239,255);
Glib_Line(0+320,0,319+320,239,255);
Glib_Line(0+320,239,319+320,0,255);

// 00
// #0
Glib_Rectangle(0,0+240,319,239+240,255);
Glib_Line(0,0+240,319,239+240,255);
Glib_Line(0,239+240,319,0+240,255);

// 00
// 0#
Glib_Rectangle(0+320,0+240,319+320,239+240,255);
Glib_Line(0+320,0+240,319+320,239+240,255);
Glib_Line(0+320,239+240,319+320,0+240,255);
Glib_Rectangle(50+320,50+240,269+320,189+240,255);

Uart_Printf("Virtual Screen Test(256 color). Press any key[ijkm\\r]!\n");
MoveViewPort(MODE_COLOR);

Lcd_MoveViewPort(0,0,MODE_COLOR);

}

void MoveViewPort(int depth)
{
    int vx=0,vy=0,vd;
    vd=(depth==1)*16+(depth==4)*8+(depth==16)*4+(depth==256)*2;
    while(1)
    {
        switch(Uart_Getch())
        {
            case 'i':
                if(vy>=vd)vy-=vd;
                break;
            case 'j':
                if(vx>=vd)vx-=vd;
                break;
            case 'k':
                if(vx<=SCR_XSIZE-LCD_XSIZE-vd)vx+=vd;
                break;
            case 'm':
                if(vy<=(SCR_YSIZE-LCD_YSIZE-vd))vy+=vd;
                break;
            case '\r':
                return;
            default:
                break;
        }
        Uart_Printf("vx=%3d,vy=%3d\n",vx,vy);
        Lcd_MoveViewPort(vx,vy,depth);
    }
}

```

```

/***** for only test *****/

#define BUFFER0_PREPARED (0)
#define BUFFER1_PREPARED (1)
#define BUFFER0_USED (2)
#define BUFFER1_USED (3)

#define M5D(n) ((n) & 0x1ffff)
#define MVAL (13)
unsigned int (*frameBuffer256_2)[SCR_XSIZE/4];

void MoveViewPort2(int depth)
{
    int vx=0,vy=0,vd;
    U32 addr;
    char key;
    int state=BUFFER0_USED;
    vd=(depth==1)*16+(depth==4)*8+(depth==16)*4+(depth==256)*2;
    while(1)
    {
        while(1)
        {
            key=Uart_GetKey();
            if(key!=0)break;

            if(state==BUFFER1_USED)
            {
                //set the frame buffer as BUFFER1
                state=BUFFER0_PREPARED;
                while((rLCDCON1>>22)==0); // if x>64
                addr=(U32)frameBuffer256_2+(vx/1)+vy*(SCR_XSIZE/1);
                rLCSADDR1= (0x3<<27) | ( (addr>>22)<<21 ) | M5D(addr>>1);
                // 256-color, LCDBANK, LCDBASEU
                rLCSADDR2= M5D(((addr+(SCR_XSIZE*LCD_YSIZE))>>1)) | (MVAL<<21);
            }

            if(state==BUFFER0_USED)
            {
                //set the frame buffer as BUFFER0
                state=BUFFER1_PREPARED;
                while((rLCDCON1>>22)==0); // if x>64
                addr=(U32)frameBuffer256_2+(vx/1)+vy*(SCR_XSIZE/1);
                rLCSADDR1= (0x3<<27) | ( (addr>>22)<<21 ) | M5D(addr>>1);
                // 256-color, LCDBANK, LCDBASEU
                rLCSADDR2= M5D(((addr+(SCR_XSIZE*LCD_YSIZE))>>1)) | (MVAL<<21);
            }
        }

        if((rLCDCON1>>22)==0)
        {
            if(state==BUFFER0_PREPARED)state=BUFFER0_USED;
            if(state==BUFFER1_PREPARED)state=BUFFER1_USED;
        }
    }

    switch(key)
    {
        case 'i':
            if(vy>=vd)vy-=vd;
            break;
        case 'j':
            if(vx>=vd)vx-=vd;
            break;
        case 'k':
            if(vx<=SCR_XSIZE-LCD_XSIZE-vd)vx+=vd;
            break;
    }
}

```

```

        case 'm':
            if(vy<=(SCR_YSIZE-LCD_YSIZE-vd))vy+=vd;
            break;
        case '\r':
            return;
        default:
            break;
    }
    Uart_Printf("vx=%3d,vy=%3d\n",vx,vy);
    Lcd_MoveViewPort(vx,vy,depth);
}

}

void Test_LcdColor2(void)
{
    int i,j,k;

    rPDATE=rPDATE&~(1<<5)|(1<<5);        //GPE5=H
    rPCONE=rPCONE&~(3<<10)|(1<<10);        //GPE5=output
    rPCONC=rPCONC&~(0xff<<8)|(0xff<<8);    //GPC[4:7] => VD[7:4]

    Uart_Printf("[ (240x3)x320 Color STN Virtual Screen & 2 Frame Buffers Test]\n");

    Lcd_Init(MODE_COLOR);
    framebuffer256_2=(unsigned int (*)(SCR_XSIZE/4))malloc(SCR_XSIZE/1*SCR_YSIZE);

    if((U32)frameBuffer256==0x0)return;
    if((U32)frameBuffer256_2==0x0)return;

    Glib_Init(MODE_COLOR);

    Glib_ClearScr(0);

    Glib_ClearScr(0);
    k=0;
    for(i=160;i<480;i+=20)
    {
        for(j=120;j<360;j+=15)
        {
            Glib_FilledRectangle(i,j,i+19,j+14,k);
            k++;
        }
    }

    // #0
    // 00
    Glib_Rectangle(0,0,319,239,255);
    Glib_Line(0,0,319,239,255);
    Glib_Line(0,239,319,0,255);

    // 0#
    // 00
    Glib_Rectangle(0+320,0,319+320,239,255);
    Glib_Line(0+320,0,319+320,239,255);
    Glib_Line(0+320,239,319+320,0,255);

    // 00
    // #0
    Glib_Rectangle(0,0+240,319,239+240,255);
    Glib_Line(0,0+240,319,239+240,255);

```

```
Glib_Line(0,239+240,319,0+240,255);

// 00
// 0#
Glib_Rectangle(0+320,0+240,319+320,239+240,255);
Glib_Line(0+320,0+240,319+320,239+240,255);
Glib_Line(0+320,239+240,319+320,0+240,255);
Glib_Rectangle(50+320,50+240,269+320,189+240,255);

for(i=0;i<720/4;i++)
    for(j=0;j<960;j++)
    {
        framebuffer256_2[j][i]=frameBuffer256[j][i];
    }

Uart_Printf("Virtual Screen Test(256 color). Press any key[ijkm\\r]!\n");
MoveViewPort2(MODE_COLOR);

Lcd_MoveViewPort(0,0,MODE_COLOR);

}
```

**44BTEST : lcdlib.h**

```

#ifndef __LCDLIB_H__
#define __LCDLIB_H__

#include "..\inc\option.h"

void Lcd_Init(int depth);
void Lcd_MoveViewPort(int vx,int vy,int depth);

#define MODE_MONO    (1)
#define MODE_G4      (4)
#define MODE_G16     (16)
#define MODE_COLOR   (256)

#if (LCD_TYPE==MLCD_320_240)
#define SCR_XSIZE    (640)
#define SCR_YSIZE    (480)

#define LCD_XSIZE    (320)
#define LCD_YSIZE    (240)

#elif (LCD_TYPE==CLCD_240_320)
#define SCR_XSIZE    (640)
#define SCR_YSIZE    (480)

#define LCD_XSIZE    (240)
#define LCD_YSIZE    (320)
#endif

extern unsigned int (*frameBuffer1)[SCR_XSIZE/32];
extern unsigned int (*frameBuffer4)[SCR_XSIZE/16];
extern unsigned int (*frameBuffer16)[SCR_XSIZE/8];
extern unsigned int (*frameBuffer256)[SCR_XSIZE/4];

#endif /*__LCDLIB_H__*/

```

**44BTEST : lcdlib.c**

```

#include <string.h>
#include "..\inc\44b.h"
#include "..\inc\44bllib.h"
#include "..\inc\def.h"
#include "..\inc\lcd.h"
#include "..\inc\lcdlib.h"
#include "..\inc\glib.h"

//SCR_XSIZE,SCR_YSIZE,LCD_XSIZE,LCD_YSIZE are defined in lcd.h

#define M5D(n) ((n) & 0x1ffffff)

#define ARRAY_SIZE_MONO    (SCR_XSIZE/8*SCR_YSIZE)
#define ARRAY_SIZE_G4      (SCR_XSIZE/4*SCR_YSIZE)
#define ARRAY_SIZE_G16     (SCR_XSIZE/2*SCR_YSIZE)
#define ARRAY_SIZE_COLOR   (SCR_XSIZE/1*SCR_YSIZE)

#define HOZVAL              (LCD_XSIZE/4-1)

```



```

#define HOZVAL_COLOR          (LCD_XSIZE*3/8-1)
#define LINEVAL               (LCD_YSIZE-1)
#define MVAL                   (13)

#define CLKVAL_MONO           (13) //60Mhz, CLKVAL=19 ->78.6Hz
#define CLKVAL_G4              (9)  //40Mhz, CLKVAL=9  ->110Hz
#define CLKVAL_G16             (10) //40Mhz, CLKVAL=10 ->101Hz
                                //          9  ->112Hz

//#define CLKVAL_COLOR         (7)  //40Mhz
#define CLKVAL_COLOR           (10) //60Mhz

void LcdInit(int color);

unsigned int (*frameBuffer1)[SCR_XSIZE/32];
unsigned int (*frameBuffer4)[SCR_XSIZE/16];
unsigned int (*frameBuffer16)[SCR_XSIZE/8];
unsigned int (*frameBuffer256)[SCR_XSIZE/4];

#define MVAL_USED 0

void Lcd_Init(int depth)
{
    //320x240 1bit/1pixel LCD

    switch(depth)
    {
    case 1:
        if((U32)frameBuffer1==0)
        {
            //The total frame memory should be inside 4MB.
            //For example, if total memory is 8MB, the frame memory
            //should be in 0xc000000~0xc3ffffff or c400000~c7ffffff.
            //But, the following code doesn't meet this condition(4MB)
            //if the code size & location is changed..
            frameBuffer1=(unsigned int (*)(SCR_XSIZE/32))malloc(ARRAY_SIZE_MONO);
        }

        rLDCON1=(0)|(1<<5)|(MVAL_USED<<7)|(0x3<<8)|(0x3<<10)|(CLKVAL_MONO<<12);
            // disable, 4B_SINGL_SCAN, WDLY=8clk, WLH=8clk,
        rLDCON2=(LINEVAL)|(HOZVAL<<10)|(10<<21);
            //LINEBLANK=10 (without any calculation)
        rLCSADDR1= (0x0<<27) | ( ((U32)frameBuffer1>>22)<<21 ) | M5D((U32)frameBuffer1>>1);
            // monochrome, LDCBANK, LCDBASEU
        rLCSADDR2= M5D( (((U32)frameBuffer1+(SCR_XSIZE*LCD_YSIZE/8))>>1) ) | (MVAL<<21);
        rLCSADDR3= (LCD_XSIZE/16) | ( ((SCR_XSIZE-LCD_XSIZE)/16)<<9 );

        rLDCON1=(1)|(1<<5)|(MVAL_USED<<7)|(0x3<<8)|(0x3<<10)|(CLKVAL_MONO<<12);
            // enable, 4B_SINGL_SCAN, WDLY=8clk, WLH=8clk,
        break;

    case 4:
        if((U32)frameBuffer4==0)
        {
            //The total frame memory should be inside 4MB.
            //For example, if total memory is 8MB, the frame memory
            //should be in 0xc000000~0xc3ffffff or c400000~c7ffffff.
            //But, the following code doesn't meet this condition(4MB)
            //if the code size & location is changed..
            frameBuffer4=(unsigned int (*)(SCR_XSIZE/16))malloc(ARRAY_SIZE_G4);
        }
    }
}

```

```

rLCDCON1=(0)|(1<<5)|(MVAL_USED<<7)|(0x3<<8)|(0x3<<10)|(CLKVAL_G4<<12);
// disable, 4B_SINGL_SCAN, WDLY=8clk, WLH=8clk,
rLCDCON2=(LINEVAL)|(HOZVAL<<10)|(10<<21);
//LINEBLANK=10 (without any calculation)
rLCDSADDR1= (0x1<<27) | ( ((U32)frameBuffer4>>22)<<21 ) | M5D((U32)frameBuffer4>>1);
// 4-gray, LCDBANK, LCDBASEU
rLCDSADDR2= M5D(((U32)frameBuffer4+(SCR_XSIZE*LCD_YSIZE/4))>>1)) | (MVAL<<21);
rLCDSADDR3= (LCD_XSIZE/8) | ( ((SCR_XSIZE-LCD_XSIZE)/8)<<9 );

//The following value has to be changed for better display.
//Select 4 levels among 16 gray levels.

rBLUELUT=0xfa40;
rDITHMODE=0x0;
rDP1_2 =0xa5a5;
rDP4_7 =0xba5da65;
rDP3_5 =0xa5a5f;
rDP2_3 =0xd6b;
rDP5_7 =0xeb7b5ed;
rDP3_4 =0x7dbe;
rDP4_5 =0x7ebdf;
rDP6_7 =0x7fdfbfe;

rLCDCON1=(1)|(1<<5)|(MVAL_USED<<7)|(0x3<<8)|(0x3<<10)|(CLKVAL_G4<<12);
// enable, 4B_SINGL_SCAN, WDLY=8clk, WLH=8clk,
break;

case 16:
if((U32)frameBuffer16==0)
{
//The total frame memory should be inside 4MB.
//For example, if total memory is 8MB, the frame memory
//should be in 0xc000000~0xc3ffffff or c400000~c7ffffff.
//But, the following code doesn't meet this condition(4MB)
//if the code size & location is changed..
frameBuffer16=(unsigned int (*)(SCR_XSIZE/8))malloc(ARRAY_SIZE_G16);
}

//The following value has to be changed for better display.
/* rDITHMODE=0x1223a; //philips
rDP1_2 =0x5a5a;
rDP4_7 =0x366cd9b;
rDP3_5 =0xda5a7;
rDP2_3 =0xad7;
rDP5_7 =0xfeda5b7;
rDP3_4 =0xebd7;
rDP4_5 =0xebfd7;
rDP6_7 =0x7efdfbf;*/

//rDITHMODE=0x12210;
/* rDITHMODE=0x0;
rDP1_2 =0xa5a5;
rDP4_7 =0xba5da65;
rDP3_5 =0xa5a5f;
rDP2_3 =0xd6b;
rDP5_7 =0xeb7b5ed;
rDP3_4 =0x7dbe;
rDP4_5 =0x7ebdf;
rDP6_7 =0x7fdfbfe;*/

```

```

rDITHMODE=0x12210;
//rDITHMODE=0x0;
rDP1_2 =0xa5a5;
rDP4_7 =0xba5da65;
rDP3_5 =0xa5a5f;
rDP2_3 =0xd6b;
rDP5_7 =0xeb7b5ed;
rDP3_4 =0x7dbe;
rDP4_5 =0x7ebdf;
rDP6_7 =0x7dfbfe;

rLCDCON1=(0)|(1<<5)|(MVAL_USED<<7)|(0x3<<8)|(0x3<<10)|(CLKVAL_G16<<12);
// disable, 4B_SINGL_SCAN, WDLY=8clk, WLH=8clk,
rLCDCON2=(LINEVAL)|(HOZVAL<<10)|(10<<21);
//LINEBLANK=10 (without any calculation)
rLCSADDR1= (0x2<<27) | ( ((U32)frameBuffer16>>22)<<21 ) |
M5D((U32)frameBuffer16>>1);
// 16-gray, LCD BANK, LCD BASEU
rLCSADDR2= M5D(((U32)frameBuffer16+(SCR_XSIZE*LCD_YSIZE/2))>>1)) | (MVAL<<21);
rLCSADDR3= (LCD_XSIZE/4) | ( ((SCR_XSIZE-LCD_XSIZE)/4)<<9 );
rLCDCON1=(1)|(1<<5)|(MVAL_USED<<7)|(0x3<<8)|(0x3<<10)|(CLKVAL_G16<<12);

// enable, 4B_SINGL_SCAN, WDLY=8clk, WLH=8clk,
break;

case 256:
if((U32)frameBuffer256==0)
{
//The total frame memory should be inside 4MB.
//For example, if total memory is 8MB, the frame memory
//should be in 0xc000000~0xc3ffffff or c400000~c7ffffff.
//But, the following code doesn't meet this condition(4MB)
//if the code size & location is changed..
frameBuffer256=(unsigned int (*)(SCR_XSIZE/4))malloc(ARRAY_SIZE_COLOR);
}

rLCDCON1=(0)|(2<<5)|(MVAL_USED<<7)|(0x3<<8)|(0x3<<10)|(CLKVAL_COLOR<<12);
// disable, 8B_SINGL_SCAN, WDLY=8clk, WLH=8clk,
rLCDCON2=(LINEVAL)|(HOZVAL_COLOR<<10)|(10<<21);
//LINEBLANK=10 (without any calculation)
rLCSADDR1= (0x3<<27) | ( ((U32)frameBuffer256>>22)<<21 ) |
M5D((U32)frameBuffer256>>1);
// 256-color, LCD BANK, LCD BASEU
rLCSADDR2= M5D(((U32)frameBuffer256+(SCR_XSIZE*LCD_YSIZE))>>1)) | (MVAL<<21);
rLCSADDR3= (LCD_XSIZE/2) | ( ((SCR_XSIZE-LCD_XSIZE)/2)<<9 );

//The following value has to be changed for better display.

rREDLUT =0xfdb96420;
rGREENLUT=0xfdb96420;
rBLUELUT =0xfb40;

rDITHMODE=0x0;
rDP1_2 =0xa5a5;
rDP4_7 =0xba5da65;
rDP3_5 =0xa5a5f;
rDP2_3 =0xd6b;
rDP5_7 =0xeb7b5ed;
rDP3_4 =0x7dbe;

```

```

    rDP4_5 = 0x7ebdf;
    rDP6_7 = 0x7fdfbfe;

    rLCDCON1=(1)|(2<<5)|(MVAL_USED<<7)|(0x3<<8)|(0x3<<10)|(CLKVAL_COLOR<<12);
    // enable, 8B_SINGL_SCAN, WDLY=8clk, WLH=8clk,

    break;

default:
    break;
}
}

void Lcd_MoveViewPort(int vx,int vy,int depth)
{
    U32 addr;
    switch(depth)
    {

    case 1:
        // LCDBASEU,LCDBASEL register has to be changed before 12 words before the end of
VLINE.
        // In mono mode, x=320 is 10 words, So, We can't change LCDBASEU,LCDBASEL
        // during LINECNT=1~0 at mono mode.

        //The processor mode should be supervisor mode.
        DisableInterrupt();

        #if (LCD_XSIZE<512)
            while((rLCDCON1>>22)<=1); // if x<512
        #else
            while((rLCDCON1>>22)==0); // if x>512 ((12+4)*32)
        #endif

        addr=(U32)frameBuffer1+(vx/8)+vy*(SCR_XSIZE/8);
        rLCDSADDR1= (0x0<<27) | ( (addr>>22)<<21 ) | M5D(addr>>1);
        // monochrome, LCDBANK, LCDBASEU
        rLCDSADDR2= M5D( ((addr+(SCR_XSIZE*LCD_YSIZE/8))>>1) ) | (MVAL<<21);

        EnableInterrupt();
        break;

    case 4:
        //The processor mode should be supervisor mode.
        DisableInterrupt();

        #if (LCD_XSIZE<256)
            while((rLCDCON1>>22)<=1); // if x<256
        #else
            while((rLCDCON1>>22)==0); // if x>256
        #endif

        addr=(U32)frameBuffer4+(vx/4)+vy*(SCR_XSIZE/4);
        rLCDSADDR1= (0x1<<27) | ( (addr>>22)<<21 ) | M5D(addr>>1);
        // 4-gray, LCDBANK, LCDBASEU
        rLCDSADDR2= M5D( ((addr+(SCR_XSIZE*LCD_YSIZE/4))>>1) ) | (MVAL<<21);

        EnableInterrupt();
        break;
    }
}

```

```

case 16:
    //The processor mode should be supervisor mode.
    DisableInterrupt();

    #if (LCD_XSIZE<128)
        while((rLCDCON1>>22)<=1); // if x<128
    #else
        while((rLCDCON1>>22)==0); // if x>128
    #endif
    addr=(U32)frameBuffer16+(vx/2)+vy*(SCR_XSIZE/2);
    rLCDSADDR1= (0x2<<27) | ( (addr>>22)<<21 ) | M5D(addr>>1);
    // 16-gray, LCDBANK, LCDBASEU
    rLCDSADDR2= M5D(((addr+(SCR_XSIZE*LCD_YSIZE/2))>>1)) | (MVAL<<21);

    EnableInterrupt();
    break;

case 256:
    //The processor mode should be supervisor mode.
    DisableInterrupt();

    #if (LCD_XSIZE<64)
        while((rLCDCON1>>22)<=1); // if x<64
    #else
        while((rLCDCON1>>22)==0); // if x>64
    #endif
    addr=(U32)frameBuffer256+(vx/1)+vy*(SCR_XSIZE/1);
    rLCDSADDR1= (0x3<<27) | ( (addr>>22)<<21 ) | M5D(addr>>1);
    // 256-color, LCDBANK, LCDBASEU
    rLCDSADDR2= M5D(((addr+(SCR_XSIZE*LCD_YSIZE))>>1)) | (MVAL<<21);

    EnableInterrupt();
    break;
}
}

```

**44BTEST : nwait.h**

```
#ifndef __NWAIT_H__
#define __NWAIT_H__

void Test_WaitPin(void);

#endif /*__NWAIT_H__*/
```

**44BTEST : nwait.c**

```
#include <string.h>
#include "..\inc\def.h"
#include "..\inc\44b.h"
#include "..\inc\44blib.h"
#include "..\inc\idle.h"

void WaitRd8(void);
void WaitWr8(void);
void WaitRd32(void);
void WaitWr32(void);
void WaitRd16(void);
void WaitWr16(void);

#define BUS8                (0)
#define BUS16               (1)
#define BUS32               (2)
#define ENWAIT              (1)
#define SRAMBE03            (1)

#define B2_Tacs              (0x0) //0clk
#define B2_Tcos              (0x0) //0clk
#define B2_Tacc              (0x2) //3clk
#define B2_Tcoh              (0x0) //0clk
#define B2_Tah               (0x0) //0clk
#define B2_Tacp              (0x0) //2clk
#define B2_PMC               (0x0) //no page mode

// Work-around with nWAIT.

void Test_WaitPin(void)
{
    rPCONF=rPCONF&(~(3<<4))|(2<<4);    //nWAIT pin is selected.
    rPUPF=0x0;

    rPCONE=rPCONE&(~(3))|(3);           //PE0=CLKOUT

    //WaitRd32();
    //WaitWr32();
    //WaitRd16();
    //WaitWr16();
    WaitRd8();
    //WaitWr8();
}
```

```

void WaitRd8(void)
{
    U8 readVal;
    int i;

    Uart_Printf("nGCS2,8-bit,read,nWAIT Test.\n");
    Uart_Printf("nWAIT/PF2 pin is configured as nWAIT pin.\n");

    rNCACHBE0=( (0x6000000>>12)<<16 )|(0x4000000>>12);
    rBWSCON=rBWSCON & ~(0xf<<8)|(BUS8<<8)|(ENWAIT<<10); //nGCS2

rBANKCON2=((B2_Tacs<<13)+(B2_Tcos<<11)+(B2_Tacc<<8)+(B2_Tcoh<<6)+(B2_Tah<<4)+(B2_Tacp<<2)+(
B2_PMC));

    for(i=0;i<2;i++); //wait until the bank configuration is in effect.

    readVal=(*(volatile U8 *)0x4000000);
    Uart_Printf("read_data=%x\n",readVal);
}

void WaitWr8(void)
{
    U8 writeVal=0xaa;
    int i;

    Uart_Printf("nGCS2,8-bit,write,nWAIT Test.\n");
    Uart_Printf("nWAIT/PF2 pin is configured as nWAIT pin.\n");

    rBWSCON=rBWSCON & ~(0xf<<8)|(BUS8<<8)|(ENWAIT<<10); //nGCS2

rBANKCON2=((B2_Tacs<<13)+(B2_Tcos<<11)+(B2_Tacc<<8)+(B2_Tcoh<<6)+(B2_Tah<<4)+(B2_Tacp<<2)+(
B2_PMC));
    rNCACHBE0=( (0x6000000>>12)<<16 )|(0x4000000>>12);

    for(i=0;i<2;i++); //wait until the bank configuration is in effect.

    *((volatile U8 *)0x4000000)=writeVal;
}

void WaitRd16(void)
{
    U16 readVal;
    int i;

    Uart_Printf("nGCS2,16-bit,read,nWAIT Test.\n");
    Uart_Printf("nWAIT/PF2 pin is configured as nWAIT pin.\n");

    rBWSCON=rBWSCON & ~(0xf<<8)|(BUS16<<8)|(ENWAIT<<10); //nGCS2

rBANKCON2=((B2_Tacs<<13)+(B2_Tcos<<11)+(B2_Tacc<<8)+(B2_Tcoh<<6)+(B2_Tah<<4)+(B2_Tacp<<2)+(
B2_PMC));
    rNCACHBE0=( (0x6000000>>12)<<16 )|(0x4000000>>12);

    for(i=0;i<2;i++); //wait until the bank configuration is in effect.

    readVal=(*(volatile U16 *)0x4000000);
    Uart_Printf("read_data=%x\n",readVal);
}

```

```

void WaitWr16(void)
{
    U32 writeVal=0xaaaa;
    int i;

    Uart_Printf("nGCS2,16-bit,write,nWAIT Test.\n");
    Uart_Printf("nWAIT/PF2 pin is configured as nWAIT pin.\n");

    rBWSCON=rBWSCON & ~(0xf<<8)|(BUS16<<8)|(ENWAIT<<10); //nGCS2

    rBANKCON2=((B2_Tacs<<13)+(B2_Tcos<<11)+(B2_Tacc<<8)+(B2_Tcoh<<6)+(B2_Tah<<4)+(B2_Tacp<<2)+(
    B2_PMC));
    rNCACHBE0=( (0x6000000>>12)<<16 )|(0x4000000>>12);

    for(i=0;i<2;i++); //wait until the bank configuration is in effect.

    *((volatile U16 *)0x4000000)=writeVal;
}

void WaitRd32(void)
{
    U32 readVal;
    int i;

    Uart_Printf("nGCS2,32-bit,read,nWAIT Test.\n");
    Uart_Printf("nWAIT/PF2 pin is configured as nWAIT pin.\n");

    rBWSCON=rBWSCON & ~(0xf<<8)|(BUS32<<8)|(ENWAIT<<10); //nGCS2

    rBANKCON2=((B2_Tacs<<13)+(B2_Tcos<<11)+(B2_Tacc<<8)+(B2_Tcoh<<6)+(B2_Tah<<4)+(B2_Tacp<<2)+(
    B2_PMC));
    rNCACHBE0=( (0x6000000>>12)<<16 )|(0x4000000>>12);

    for(i=0;i<2;i++); //wait until the bank configuration is in effect.

    readVal=*((volatile U32 *)0x4000000);
    Uart_Printf("read_data=%x\n",readVal);
}

void WaitWr32(void)
{
    U32 writeVal=0xaaaa;
    int i;

    Uart_Printf("nGCS2,32-bit,write,nWAIT Test.\n");
    Uart_Printf("nWAIT/PF2 pin is configured as nWAIT pin.\n");

    rBWSCON=rBWSCON & ~(0xf<<8)|(BUS32<<8)|(ENWAIT<<10); //nGCS2

    rBANKCON2=((B2_Tacs<<13)+(B2_Tcos<<11)+(B2_Tacc<<8)+(B2_Tcoh<<6)+(B2_Tah<<4)+(B2_Tacp<<2)+(
    B2_PMC));
    rNCACHBE0=( (0x6000000>>12)<<16 )|(0x4000000>>12);

    for(i=0;i<2;i++); //wait until the bank configuration is in effect.

    *((volatile U32 *)0x4000000)=writeVal;
}

```



**44BTEST : power.h**

```

#ifndef __POWER_H__
#define __POWER_H__

void Test_SlowMode(void);
void Test_HoldMode(void);
void _Test_StopMode(void);
void Test_SLIdleMode(void);
//void Test_IdleMode(void);
void Test_PLL(void);

#endif /*__POWER_H__*/

```

**44BTEST : power.c**

```

#include <math.h>
#include "..\inc\44b.h"
#include "..\inc\44blib.h"
#include "..\inc\def.h"
#include "..\inc\power.h"
#include "..\inc\rtc.h"

#define FIN 4000000

static void __irq SLWEint45Int(void);
//void Display_4GrayMonoImage(void);
char slw_exit=0;
void Test_SlowMode(void)
{
    int i;
    unsigned int save_MCON[9];
    unsigned int save_PCONE;

    //the value of memory control register in slow clock(1MHz)
    #if (BDRAMTYPE==DRAM)
        unsigned int MemCon[9]={0,0,0,0,0,0,0x10002,0x10002,0x800000+2033};
    #else //BDRAMTYPE==SDRAM
        unsigned int MemCon[9]={0,0,0,0,0,0,0x18000,0x18000,0x800000+2033};
    #endif

    rEXTINT=0x22222222; //falling edge
    pISR_EINT4567=(U32)SLWEint45Int;
    rINTMSK=~(BIT_GLOBAL|BIT_RTC|BIT_EINT4567);

    save_PCONE=rPCONE;
    rPCONE |= 0x3;//for monitoring FOUT
    Uart_Printf("rPCONE=0x%x\n",rPCONE);

    for(i=0;i<9;i++)
        save_MCON[i]=((unsigned int *)0x01c80004)[i];// *(unsigned int *) (0x01c80004+4*i);

    Uart_Printf("[POWER DOWN MODE TEST]\n");
    Uart_Printf("1)Entering SLOW mode.\n");
    Uart_Printf(" LEDs are flickered by 20ms period at %dMhz.\n",MCLK/1000000);
    Uart_Printf(" But, The frequency,in 1Mhz, is about %dms.\n",MCLK*20/1000000);
    Uart_Printf(" Press EINT 4/5 key to exit SLOW mode\n");
    Uart_TxEmpty(0); //To avoid being crushed the character

```

```

rLOCKTIME=0x190;//0x7d0;      //count=t_lock*Fin=2000 (t_lock=200us, Fin=10MHz)

rCLKSLOW=5|(1<<4)|(1<<5); //PLL off,SLOW mode,SLVAL=5 Fout=Fin/(2*SLVAL)=1MHz

ChangeMemCon(MemCon);//change memory control register(60MHz-->1MHz)

while(!slw_exit)
//now!  slow mode:1Mhz, PLL off
{
    Led_Display(0xf);
    Delay(100);
    Led_Display(0x0);
    Delay(100);
}

rINTMSK=BIT_GLOBAL;
for(i=0;i<9;i++)
    MemCon[i]=save_MCON[i];

ChangeMemCon(MemCon);//change memory control register(1MHz-->66MHz)

//test
//    rCLKSLOW=2;  //PLL on & Slow off
//    Uart_Printf("BYE!!\n");
//test

rCLKSLOW=2|(1<<4)|(0<<5);      //PLL on

for(i=0;i<2048;i++); //wait during PLL lock-time

rCLKSLOW=2;                //exit SLOW mode(disable slow_bit)

rPCONE=save_PCONE;
slw_exit=0;
}

void __irq SLWEint45Int(void)
{
    rEXTINPND=0xf;          //clear EXTINPND reg.
    rI_ISPC=BIT_EINT4567; //clear pending_bit
    slw_exit=1;
}

void Test_PLL(void)
{
    int i,P_div, M_div, S_val, S_div, mck;

    Uart_Printf("[Running change test of M/P/S value]\n");

    Uart_Printf("Input M vlaue\n");
    M_div=Uart_GetIntNum();
    Uart_Printf("Input P vlaue\n");
    P_div=Uart_GetIntNum();
    Uart_Printf("Input S vlaue\n");
    S_div=Uart_GetIntNum();
    S_val=pow(2,S_div);
    mck=( (M_div+8)*FIN )/( (P_div+2)*S_val );

```

```

Uart_Printf("MCLK=%d,M=0x%x,P=0x%x,S=0x%x\n",mck,M_div,P_div,S_div);
Uart_Printf("Now change PLL value\n");
Uart_TxEmpty(0);

ChangePllValue(M_div,P_div,S_div);
Uart_Init(mck,115200);
Uart_Printf("...I'm running in changed MCLK...\nPress any key to return!\n");

while(!Uart_GetKey())
{
    Led_Display(0xf);
    Delay(1000);
    Led_Display(0x0);
    Delay(1000);
}

//    ChangePllValue(0x48,0x3,0x2);//Fin=10MHz, Fout=40MHz
ChangePllValue(0x34,0x0,0x1);//Fin=10MHz, Fout=40MHz
Uart_Init(0,115200);
Uart_Printf("Returned original clock\n");
}

void Test_HoldMode(void)
{
    Uart_Printf("[HOLD Mode(Normal mode with some stopped blocks)]\n");
    Uart_Printf("IIS,IIC,ADC,RTC,UART1,SIO,ZDMA,Timer,LCD are stopped step by step.\n");
    Uart_Printf("Check the current consumption. Type any key to exit.\n");
    //Except GPIO,BDMA,UART0
    Uart_Printf("IIS off.\n");
    rCLKCON=0x3ff8;
    Uart_Getch();
    Uart_Printf("IIC off.\n");
    rCLKCON=0x1ff8;
    Uart_Getch();
    Uart_Printf("ADC off.\n");
    rADCCON|=0x20;
    rCLKCON=0x0ff8;
    Uart_Getch();
    Uart_Printf("RTC_control off.\n");
    rCLKCON=0x07f8;
    Uart_Getch();
    Uart_Printf("UART1 off.\n");
    rCLKCON=0x05f8;
    Uart_Getch();
    Uart_Printf("SIO off.\n");
    rCLKCON=0x05d8;
    Uart_Getch();
    Uart_Printf("ZDMA01 off.\n");
    rCLKCON=0x05c8;
    Uart_Getch();
    Uart_Printf("TIMER0123 off.\n");
    rCLKCON=0x05c0;
    Uart_Getch();
    Uart_Printf("LCD off.\n");
    rCLKCON=0x0580;
    Uart_Getch();

    Uart_Printf("Return to Normal Mode.\n");
    rCLKCON=0x7ff8; //IIS,IIC,ADC,RTC,UART,BRDMA,SIO,ZDMA,Timer,LCD
}

```

**44BTEST : rtc.h**

```

#ifndef __RTC_H__
#define __RTC_H__

int Test_Rtc_Alarm(void);
void Test_Rtc_Tick(void);
void Display_Rtc(void);

void Rtc_Init(void);

#define TESTYEAR      (0x99)
#define TESTMONTH     (0x12)
#define TESTDAY       (0x31)
#define TESTDATE      (0x06) // SUN:1 MON:2 TUE:3 WED:4 THU:5 FRI:6 SAT:7
#define TESTHOUR      (0x23)
#define TESTMIN        (0x59)
#define TESTSEC        (0x59)

#define TESTYEAR2     (0x00)
#define TESTMONTH2    (0x01)
#define TESTDAY2      (0x01)
#define TESTDATE2     (0x07) // SUN:1 MON:2 TUE:3 WED:4 THU:5 FRI:6 SAT:7
#define TESTHOUR2     (0x00)
#define TESTMIN2      (0x00)
#define TESTSEC2      (0x00)

#endif //__RTC_H__

```

**44BTEST : rtc.c**

```

#include "..\inc\44b.h"
#include "..\inc\44blib.h"
#include "..\inc\rtc.h"

void __irq Rtc_Tick(void);

char *date[8] = {"", "SUN", "MON", "TUE", "WED", "THU", "FRI", "SAT"};

volatile unsigned int sec_tick;

void Display_Rtc(void)
{
    int year;
    int month, day, weekday, hour, min, sec;

    rRTCCON = 0x01; // R/W enable, 1/32768, Normal(merge), No reset
    // Uart_Printf("This test should be excuted once RTC test(Alarm) for RTC
    initialization\n");

    while(1)
    {
        if(rBCDYEAR == 0x99)
            year = 0x1999;
        else
            year = 0x2000 + rBCDYEAR;
        month=rBCDMON;
        day=rBCDDAY;
    }
}

```

```

        weekday=rBCDDATE;
        hour=rBCDHOURL;
        min=rBCDMIN;
        sec=rBCDSEC;
        if(sec!=0)
            break;
    }
    Uart_Printf("%4x,%2x,%2x,%s,%2x:%2x:%2x\n",year,month,day,date[weekday],hour,min,sec);
    rRTCCON = 0x00;    // R/W disable(for power consumption), 1/32768, Normal(merge), No
reset
}

volatile int isRtcInt;

void __irq Rtc_Int(void)
{
    rI_ISPC=BIT_RTC;
    //rI_ISPC; //is needed only when cache=on & wrbuf=on & BSFRD=0
    Uart_Printf("RTC Alarm Interrupt O.K.\n");
    isRtcInt=1;
}

int Test_Rtc_Alarm(void)
{
    Uart_Printf("[RTC Alarm Test for S3C44B0X]\n");

    Rtc_Init();

    //    rRTCCON = 0x01; // R/W enable, 1/32768, Normal(merge), No reset
    rALMYEAR=TESTYEAR2 ;
    rALMMON =TESTMONTH2;
    rALMDAY =TESTDAY2   ;
    rALMHOUR=TESTHOUR2  ;
    rALMMIN =TESTMIN2   ;
    rALMSEC =TESTSEC2+9;
    isRtcInt=0;
    pISR_RTC=(unsigned int)Rtc_Int;
    rRTCALM=0x7f;
    rRTCCON=0x0;
    rINTMSK=~(BIT_GLOBAL|BIT_RTC);

    while(isRtcInt==0);

    rINTMSK=BIT_GLOBAL;
    rRTCCON = 0x0;    // R/W disable(for power consumption), 1/32768, Normal(merge), No
reset
    return 1;
}

void Rtc_Init(void)
{
    rRTCCON = 0x01; // R/W enable, 1/32768, Normal(merge), No reset
    rBCDYEAR = TESTYEAR;
    rBCDMON  = TESTMONTH;
    rBCDDAY  = TESTDAY;    // SUN:1 MON:2 TUE:3 WED:4 THU:5 FRI:6 SAT:7
    rBCDDATE = TESTDATE;
    rBCDHOURL = TESTHOUR;
    rBCDMIN   = TESTMIN;
    rBCDSEC   = TESTSEC;
}

```

```
void Test_Rtc_Tick(void)
{
    Uart_Printf("[RTC Tick interrupt test for S3C44B0X]\n");
    Uart_Printf("Typing any key to exit!!!\n");
    pISR_TICK=(unsigned)Rtc_Tick;

    sec_tick=1;
    rINTMSK=~(BIT_GLOBAL|BIT_TICK);
    rRTCCON=0x0;          //R/W disable(for power consumption), 1/32768, Normal(merge), No
reset
    rTICINT = 127+(1<<7); //START

    Uart_Getch();
    rINTMSK |= (BIT_GLOBAL | BIT_TICK);

    rRTCCON=0x0;        //END
}

void __irq Rtc_Tick(void)
{
    rI_ISPC=BIT_TICK;
    Uart_Printf("\b\b\b\b\b\b\b\b%03d sec",sec_tick++);
}
```

**44BTEST : sio.h**

```

#ifndef __SIO_H__
#define __SIO_H__

void Test_Sio(void);
void Test_SIO_TX_BDMA0(void);
void Test_SIO_RX_BDMA1(void);

#endif /*__SIO_H__*/

```

**44BTEST : sio.c**

```

#include <string.h>
#include "..\inc\44b.h"
#include "..\inc\44blib.h"
#include "..\inc\sio.h"
#include "..\inc\def.h"

volatile char *sioTxStr,*sioRxStr;
volatile int endSioTx;

void __irq Sio_Int(void)
{
    rI_ISPC=BIT_SIO;
    //rI_ISPC; //is needed only when cache=on & wrbuf=on & BSFRD=0

    *sioRxStr++=rSIODAT;

    if(*sioTxStr!='\0')
    {
        rSIODAT=*sioTxStr++;
        rSIOCON|=(1<<3);
    }
    else
        endSioTx=1;
}

void Test_Sio(void)
{
    unsigned int save_F,save_PF;
    char *txStr,*rxStr;

    Uart_Printf("[SIO Tx/Rx Test]\n");
    Uart_Printf("Connect SIO_TXD into SIO_RXD.\n");

    pISR_SIO=(unsigned)Sio_Int;
    rINTMSK=~(BIT_GLOBAL|BIT_SIO);
    endSioTx=0;
    sioTxStr="ABCDEFGHIJKLMNOPQRSTUVWXYZ";
    sioRxStr=(char *)malloc(strlen((char *)sioTxStr)+1);
    txStr=(char *)sioTxStr;
    rxStr=(char *)sioRxStr;

    save_F=rPCONF;
    save_PF=rPUPF;
    //interrupt,autorun,rising edge,tx&rx,MSB first,internal clk
    rPCONF=(rPCONF&0x3ff)+(3<<19)+(3<<16)+(3<<13)+(3<<10);

```

```

rPUPF |=0x1e0;

rSBRDR=0x1; //if MCLK=40Mhz,SIOCK=10Mhz
rIVTCNT=0x1; //if MCLK=40Mhz,Interval=200ns(5Mhz)

rSIOCON=1|(0<<2)|(1<<4)|(1<<5)|(0<<6);

rSIODAT='A';sioTxStr++;

rSIOCON|=(1<<3);

while(endSioTx==0);

*sioRxStr='\0';

Uart_Printf("Tx Strings:%s\n",txStr);
Uart_Printf("Rx Strings:%s :",rxStr);
if(strcmp(rxStr,txStr)==0)Uart_Printf("O.K.\n");
else Uart_Printf("ERROR!!!\n");

rINTMSK=BIT_GLOBAL;
rPCONF=save_F;
rPUPF=save_PF;
free(rxStr);
}

static U8 tx_buffer[100];

volatile int tx_bdma0Done;

void __irq Bdma0Tx_Int(void)
{
    rDCNTZ=0x3;
    tx_bdma0Done=1;

    rI_ISPC=BIT_BDMA0;    //clear pending
}

void Test_SIOTX_BDMA0(void)
{
    int i;
    U32 save_F;
    save_F=rPCONF;

    rPCONF=(rPCONF&0x3fff)+(3<<19)+(3<<16)+(3<<13)+(3<<10);

    Uart_Printf("[SIO BDMA Tx test]\n");
    Uart_Printf("This test should be configured two boards.\n");
    Uart_Printf("Start Rx first, then press any key.\n");

    rNCACHBE0= ((int)tx_buffer>>12) + ( (((int)tx_buffer>>12) +1)<<16 );
    for(i=0; i<100; i++)
        tx_buffer[i]=i;
    //    tx_buffer[0]=0xff;

    tx_bdma0Done=0;

```



```

pISR_BDMA0 = (unsigned)Bdma0Tx_Int;
rINTMSK=~(BIT_GLOBAL|BIT_BDMA0);

rSBRDR = (U32)(((MCLK/2)/500000)-1);          //rSBRDR[11:0] = ((MCLK/2)/BRG) - 1
rIVTCNT = 0xff;
rDCNTZ = 0x3;
rDCNTZ = 0x0;    // init to zero
rSIOCON = 0x0|(0<<2)|(0<<4)|(1<<5)|(0<<6)|(0<<7); //dma block

rBDISRC0 = (0<<30)|(1<<28)|(U32)tx_buffer;      // SRCset = byte | fixed | src_addr
rBDIDES0 = (1<<30)|(3<<28)|(U32)0x1d14004;      // DESset = M2IO | fixed |
des_addr(little)
rBDICNT0 = (3<<30)|(1<<26)|(3<<22)|(0<<21)|(1<<20)|(U32)100; // SIO | int.cnt | ALOff
| Enable | cnt.value
// rBDICNT0 = (3<<30)|(1<<26)|(3<<22)|(0<<21)|(1<<20)|(U32)1; // SIO | int.cnt | ALOff
| Enable | cnt.value
rBDCON0 = (0x0<<2);

Uart_Getch();

rSIOCON = 2 | (0<<2) | (0<<4) | (1<<5) | (0<<6) | (0<<7); //dma unblock

while(tx_bdma0Done==0);

// rSIOCON = 0x0|(0<<2)|(0<<4)|(1<<5)|(0<<6)|(0<<7); //dma block
// rDCNTZ = 0x3;

Uart_Printf("\nBDMA transfer end\n");
for(i=0; i<100; i++)
    Uart_Printf("0x%02x",tx_buffer[i]);
// Uart_Printf("0x%x",tx_buffer[0]);
Cache_Flush();
rNCACHBE0=0x0;

// rPCONF=save_F;
}

static U8 rx_buffer[100];
volatile int rx_bdma1Done;

void __irq Bdma1Rx_Int(void)
{
    rDCNTZ=0x3;
    rx_bdma1Done=1;
    rI_ISPC=BIT_BDMA1;    //clear pending
}

void Test_SIORX_BDMA1(void)
{
    int i,j,error=0;
    U32 save_F;
    Uart_Printf("[SIO DMA Rx Test]\n");
    save_F=rPCONF;

    rNCACHBE0= ((int)rx_buffer>>12) + ( (((int)rx_buffer>>12) +1)<<16 );

    for(i=0;i<100;i++)
        rx_buffer[i]=0x0;
    // rx_buffer[0]=0x0;

    rPCONF=(rPCONF&0x3ff)+(3<<19)+(3<<16)+(3<<13)+(3<<10); //SIO port

```

```

rx_bdma1Done=0;

pISR_BDMA1 = (unsigned)Bdma1Rx_Int;
rINTMSK=~(BIT_GLOBAL|BIT_BDMA1);

rIVTCNT = 0xff;
rDCNTZ = 0x3;
rDCNTZ = 0x0; // init to zero
rSIOCON = 0x3 | (0<<2) | (0<<4) | (0<<5) | (0<<6) | (1<<7);
//BDMA1,autorun,rising edge,tx&rx,MSB first,internal clk
rBDISRC1 = (0<<30)|(3<<28)|(U32)0x1d14004; // SRCset = byte | fixed | SIODAT
rBDIDES1 = (2<<30)|(1<<28)|(U32)rx_buffer; // DESset = IO2M | incre. | des_addr
rBDICNT1 = (3<<30)|(1<<26)|(3<<22)|(0<<21)|(1<<20)|(U32)100; // SIO | int.cnt | ALoff
| Enable | cnt.value
// rBDICNT1 = (3<<30)|(1<<26)|(3<<22)|(0<<21)|(1<<20)|(U32)1;//00; // SIO | int.cnt |
ALoff | Enable | cnt.value
rBDCON1 = (0x0<<2);

rSIOCON = 0x3 | (0<<2) | (1<<3) | (0<<4) | (0<<5) | (0<<6) | (1<<7);//must set the
start bit

while(rx_bdma1Done==0);
// rSIOCON = 0x0 | (0<<2) | (0<<4) | (0<<5) | (0<<6) | (1<<7);
// rDCNTZ = 0x3;

for(i=0;i<100;i++)
    Uart_Printf("0x%02x",rx_buffer[i]);
// Uart_Printf("0x%x",rx_buffer[0]);

Cache_Flush();
rNCACHBE0=0x0;
Uart_Printf("\nBDMA receive end");

// rPCONF=save_F;
}

```

**44BTEST : stop.h**

```
#ifndef __STOP_H__
#define __STOP_H__

void Test_StopMode(void);

#endif /*__STOP_H__*/
```

**44BTEST : stop.c**

```
#include <math.h>
#include "..\inc\44b.h"
#include "..\inc\44blib.h"
#include "..\inc\def.h"
#include "..\inc\rtc.h"

void stopPort(void);
void ReturnPort(void);
void __irq STEINT45(void);
void __irq STAlarm(void);

char wake_int=0;

// Users should not use interrupt in stop mode..
void Test_StopMode(void)
{
    char *date[8] = {"", "SUN", "MON", "TUE", "WED", "THU", "FRI", "SAT"};
    unsigned int i, j, portStatus[20];
    unsigned int *memdata, *temp;
    unsigned int save_ref;

    memdata = (unsigned int *)malloc(0x100000);
    temp=memdata;

    pISR_EINT4567=(unsigned int)STEINT45;
    pISR_RTC=(U32)STAlarm;
    for(i=0;i<0x40000;i++)
        *(temp+i)=i;

    rLOCKTIME=0xffff;//400us @10MHz

    // Save the port configurations
    for(i=0;i<20;i++)
        portStatus[i]=*( (volatile U32 *) (0x1d20000+(i<2)) );

    Uart_Printf("[STOP Mode Test]\n");
    Uart_Printf("After 5 seconds, S3C44B0X will wake up by RTC alarm interrupt.\n");
    Uart_Printf("S3C44B0X will also wake up by EINT4 and EINT5.\n\n");
    Uart_TxEmpty(0); //Wait until UART0 Tx buffer empty.

    Rtc_Init();

    rALMYEAR=TESTYEAR2 ;
    rALMMON =TESTMONTH2;
    rALMDAY =TESTDAY2 ;
    rALMHOUR=TESTHOUR2 ;
    rALMMIN =TESTMIN2 ;
    rALMSEC =TESTSEC2+9;
```

```

//test rtc current
    rRTCALM=0x7f; //Start RTC alarm
//    Display_Rtc();
//    Uart_TxEmpty(0);
//test

//    rINTMSK=BIT_GLOBAL;//Before entering Stop mode interrupts must be masked!!

    stopPort();
//test slow->stop
//    rCLKSLOW=2|(1<<4)|(1<<5); //SLOWVAL=2,Fout=Fin/(2x2),PLL off.
//test
    rLCDCON1 &= 0xffffffff; //Before entering the stop mode LCD must be off

    EnterPWDN(0x1); //Entering Power down mode.

    rCLKCON=0x7ff8; //Thaw mode -> normal mode.

//test slow->stop
//    rCLKSLOW=2|(1<<4)|(0<<5); //SLOWVAL=2,Fout=Fin/(2x2),PLL on.
//    for(i=0;i<2048;i++); //Wait PLL lock time.
//    rCLKSLOW=2; //SLOWVAL=2,Fout=Fpllo,PLL on.
//test

    rLCDCON1 |= 0x1;
    for(i=0;i<20;i++) // Retrun original port configurations
        *( (volatile U32 *) (0x1d20000+(i<<2)) )=portStatus[i];

//test rtc current
    rRTCCON = 0x01; //R/W enable to write rRTCALM
//    Uart_Printf("%4x,%2x,%2x,%s,%2x:%2x:%2x\n",0x2000 +
rBCDYEAR,rBCDMON,rBCDDAY,date[rBCDDATE],rBCDHOURL,rBCDMIN,rBCDSEC);
//Display_Rtc();
//test

    rRTCALM = 0x0; //Stop RTC alarm
    rRTCCON = 0x0; //R/W disable :Stop RTC

    rINTMSK = ~(BIT_GLOBAL | BIT_EINT4567);

    switch(wake_int)
    {
        case 1:
            Uart_Printf("\nS3C44B0X is waked by EINT4\n");
            break;
        case 2:
            Uart_Printf("\nS3C44B0X is waked by EINT5\n");
            break;
        case 0:
            Uart_Printf("\nS3C44B0X is waked by Alarm\n");
            break;
        default :
            Uart_Printf("\nCheck int!!!\n");
            break;
    }

    Uart_Printf("Return to Normal Mode.\n");

    Uart_Printf("Self-refresh data verifing...\n");

```

```

    for(i=0;i<0x40000;i++)
    {
        j=*(temp+i);
        if(j!=i)
        {
            Uart_Printf("Memory test fail after Self-refresh\n");
            break;
        }
    }
    if(j+1==i)
        Uart_Printf("Stop mode and self-refresh test O.K.\n");
    free(memdata);
    wake_int=0;
    rINTMSK = BIT_GLOBAL | BIT_EINT4567;
}

void __irq STEINT45(void)
{
    wake_int=rEXTINPND;
    rEXTINPND=0xf; //clear EXTINPND reg.
    rI_ISPC=BIT_EINT4567;
}

void __irq STAlarm(void)
{
    rI_ISPC=BIT_RTC;
    wake_int=0;
}

void stopPort(void)
{
    // The I/O ports have to be configured
    // properly to reduce STOP mode current.
    // pullup + output=high
    rPCONA=0x3ff; //ROM addr:16,17,18,19 SDRAM bank addr:21,22

    rPCONB=0x7ff; //SDRAM:SCKE,SCLK,nSCAS,nSRAS

    rPDATC=0x0;
    rPCONC=0x10000000; //all output
    rPUPC=0xc000;

    rPDATD=0x0;
    rPCOND=0x5555; //all output

    rPDATE=0x0;
    rPCONE=0x4; //all output
    rPUPE=0x106;

    rPDATF=0x0;
    rPCONF=0x2400; //all output
    rPUPF=0x1e3;

    rPCONG=0xf00;
    rPUPG=0x30;

    rSPUCR=0x0;
    rEXTINT=0x22222222; //falling edge

    rRTCCON=0x0; //RTC R/W disable for power consumption
    rADCCON |=0x20; //ADC sleep mode
}

```

**44BTEST : timer.h**

```

#ifndef __TIMER_H__
#define __TIMER_H__

void Test_Timer(void);
void Test_WDTimer(void);
void Test_TimerInt(void);

#endif /*__TIMER_H__*/

```

**44BTEST : timer.c**

```

#include <string.h>
#include "..\inc\44b.h"
#include "..\inc\44blib.h"
#include "..\inc\timer.h"

void __irq Timer0Done(void);
void __irq Timer1Done(void);
void __irq Timer2Done(void);
void __irq Timer3Done(void);
void __irq Timer4Done(void);
void __irq Timer5Done(void);
void __irq Wdt_Int(void);

/*****
 * PWM Timer TOUT0/1/2/3/4 test *
 *****/
void Test_Timer(void)
{
    int save_E, save_PE;
    save_E=rPCONE;
    save_PE=rPUPE;
    rPCONE=0xaa8; //Tout0/1/2/3/4, RxD0, TxD0
    rPUPE |=0xf8;

    Uart_Printf("[Timer 0,1,2,3,4,5 Test]\n");
    Uart_Printf("Check PWM Output\n");
    Uart_Printf("MCLK=%dHz TIMER CLK=%dHz\n",MCLK,MCLK/4);
    Uart_Printf("TCNTBn=1000(PWM frequency=%dHz)\n",MCLK/4000);
    Uart_Printf("To exit PWM test mode, Press any key!!!\n");

    // T0/1/2/3/4
    rTCFG0=0x10101; // Prescaler0/1/2=1, Deadzone=1
    rTCFG1=0x0; // Interrupt, Devider=1/2
    // Timer clock = (MCLK/1)/2

    rTCNTB0=1000;
    rTCNTB1=1000;
    rTCNTB2=1000; // Total cont=1000
    rTCNTB3=1000;
    rTCNTB4=1000;

    rTCMPB0=1000-700;
    rTCMPB1=1000-700;
    rTCMPB2=1000-700; // L_count:700, H_count:300
    rTCMPB3=1000-700;
    rTCMPB4=1000-700;

```

```

    rTCON=0xaaaa0a; //auto reload, inverter off, manual update, dead zone off
    rTCON=0x999909; //start PWM operation

    Uart_Getch();

    rTCON=0x0;          //Stop timer

    rPCONE=save_E;
    rPUPE=save_PE;
}

volatile int isWdtInt;

/*****
 *      Watch-dog timer test
 *****/
void Test_WDTimer(void)
{
    Uart_Printf("[WatchDog Timer Test]\n");

    rINTMSK=~(BIT_GLOBAL|BIT_WDT);
    pISR_WDT=(unsigned)Wdt_Int;
    isWdtInt=0;

    rWTCN=(MCLK/1000000-1)<<8|(3<<3)|(1<<2); // t_watchdog = 1/66/128, interrupt
enable
    rWTDAT=8448/4;
    rWTCNT=8448/4;
    rWTCN=rWTCN|(1<<5); // 1/40/128,interrupt

    while(isWdtInt!=10);

    rWTCN=(MCLK/1000000-1)<<8|(3<<3)|(1); // 1/66/128, reset enable
    Uart_Printf("\nI will restart after 2 sec!!!\n");
    rWTCNT=8448*2;
    rWTCN=rWTCN|(1<<5); // 1/40/128,interrupt
    while(1);
    rINTMSK=BIT_GLOBAL;
}

void __irq Wdt_Int(void)
{
    rI_ISPC=BIT_WDT;
    Uart_Printf("%d ",++isWdtInt);
}

volatile int variable0,variable1,variable2,variable3,variable4,variable5;

/*****
 *      PWM Timer Interrupt0/1/2/3/4/5 test
 *****/
void Test_TimerInt(void)
{
    variable0=0;variable1=0;variable2=0;variable3=0;variable4=0;variable5=0;

    rINTMSK=~(BIT_GLOBAL|BIT_TIMER0|BIT_TIMER1|BIT_TIMER2|BIT_TIMER3|BIT_TIMER4|BIT_TIMER5);
    pISR_TIMER0=(int)Timer0Done;
    pISR_TIMER1=(int)Timer1Done;
    pISR_TIMER2=(int)Timer2Done;

```

```

pISR_TIMER3=(int)Timer3Done;
pISR_TIMER4=(int)Timer4Done;
pISR_TIMER5=(int)Timer5Done;

Uart_Printf("[Timer0/1/2/3/4/5 Interrupt Test]\n");

rTCFG0=0xf0f0f; //dead zone=0,pre2=0xf,pre1=0xf,pre0=0xf
// rTCFG1=0x401234; //all
interrupt,mux5=EXTCLK,mux2=1/2,mux3=1/4,mux2=1/8,mux1=1/16,mux0=1/32
rTCFG1=0x01234; //all interrupt,mux5=1/2,mux2=1/2,mux3=1/4,mux2=1/8,mux1=1/16,mux0=1/32

rTCNTB0=0xffff; //(1/(66MHz/15/32))*0xffff=0.48s
rTCNTB1=0xffff; //(1/(66MHz/15/16))*0xffff=0.24s
rTCNTB2=0xffff; //(1/(66MHz/15/8 ))*0xffff=0.12s
rTCNTB3=0xffff; //(1/(66MHz/15/4 ))*0xffff=0.06s
rTCNTB4=0xffff; //(1/(66MHz/15/2 ))*0xffff=0.03s
rTCNTB5=0xffff; //(1/(66MHz/15/2 ))*0xffff=0.03s
// rTCNTB5=0xffff; //(1/(1MHz))*0xffff=0.06s, EXTCLK

rTCON=0x2222202; //update T5/T4/T3/T2/T1/T0
rTCON=0x5999901; //T5/T4/T3/T2/T1=auto reload,T0=one shot,all_start

while(variable0==0);
    Delay(1); //To compensate timer error(<1 tick period)
rTCON=0x0; //all_stop

if(variable5==16 && variable4==16 && variable3==8 && variable2==4 && variable1==2 &&
variable0==1)
// if(variable5==8 && variable4==16 && variable3==8 && variable2==4 && variable1==2 &&
variable0==1)
    Uart_Printf("Timer0/1/2/3/4/5 interrupt Test --> OK\n");
else
    Uart_Printf("Timer0/1/2/3/4/5 interrupt Test --> Fail\n");
    Uart_Printf("Timer0-%d(1),Timer1-%d(2),Timer2-%d(4),Timer3-%d(8),Timer4-%d(16),Timer5-
%d(16)\n",
// Uart_Printf("Timer0-%d(1),Timer1-%d(2),Timer2-%d(4),Timer3-%d(8),Timer4-
%d(16),Timer5-%d(8)\n",
        variable0,variable1,variable2,variable3,variable4,variable5);

rINTMSK=BIT_GLOBAL;
}

void __irq Timer0Done(void)
{
    rI_ISPC=BIT_TIMER0;
    variable0++;
}

void __irq Timer1Done(void)
{
    rI_ISPC=BIT_TIMER1;
    variable1++;
}

void __irq Timer2Done(void)
{
    rI_ISPC=BIT_TIMER2;
    variable2++;
}

```



```
void __irq Timer3Done(void)
{
    rI_ISPC=BIT_TIMER3;
    variable3++;
}

void __irq Timer4Done(void)
{
    rI_ISPC=BIT_TIMER4;
    variable4++;
}

void __irq Timer5Done(void)
{
    rI_ISPC=BIT_TIMER5;
    variable5++;
}
```

**44BTEST : uart.h**

```
#ifndef __UART_H__
#define __UART_H__

void Test_Uart0(void);
void Test_Uart0Fifo(void);
void Test_Uart1(void);
void Test_Uart1Fifo(void);
void Test_Uart0Range(void);
void Test_Uart1Max(void);
void Test_Uart1Status(void);
void Test_UartAFC_Rx(void);
void Test_UartAFC_Tx(void);
void Test_BDMA(void);
#endif /*__UART_H__*/
```

**44BTEST : uart.c**

```
#include <string.h>
#include <stdlib.h>
#include "..\inc\44b.h"
#include "..\inc\44blib.h"
#include "..\inc\def.h"
#include "..\inc\rtc.h"

#define KEY_BUFLLEN 100
#define AFC_BUFLLEN 0x100

char Uart_IntGetkey(void);
void Uart_Port(void);
void Return_Port(void);

void __irq Uart0_TxFifoInt(void);
void __irq Uart0_RxFifoInt(void);
void __irq Uart0_RxFifoErrorInt(void);

void __irq Uart0_RxInt(void);
void __irq Uart0_TxInt(void);

void __irq Uart1_TxFifoInt(void);
void __irq Uart1_RxFifoInt(void);
void __irq Uart1_RxFifoErrorInt(void);

void __irq Uart1_RxInt(void);
void __irq Uart1_TxInt(void);

void __irq U1AFC_TxInt(void);
void __irq U0AFC_RxInt(void);
void __irq U0AFC_RxErrorInt(void);

void __irq Exint2(void);
void __irq Test_Done(void);
void __irq Error(void);

static unsigned char keyBuf[KEY_BUFLLEN];
volatile static int keyBufRdPt=0;
volatile static int keyBufWrPt=0;
volatile char out=1;
```

```

volatile static char *uart0TxStr;
volatile static char *uart1TxStr;

volatile U32 save_UC,save_UE,save_UF,save_UPC,save_UPE,save_UPF;

void Uart_Port(void)
{
    save_UC=rPCONC;//for nRTS0,nCTS0
    save_UE=rPCONE;//for TxD0,RxD0
    save_UF=rPCONF;//for nRTS1,TxD1,RxD1,nCTS1
    save_UPC=rPUPC;
    save_UPE=rPUPE;
    save_UPF=rPUPF;

    rPCONC |=0xf0000000;
    rPUPC |=0xc000;
    rPCONE=(rPCONE &0x3ffeb)|0x28;
    rPUPE |=0x6;
    rPCONF=(rPCONF &0x3ff)+0x124800;
    rPUPF |=0x1e0;
}

void Return_Port(void)
{
    rPCONC=save_UC;
    rPCONE=save_UE;
    rPCONF=save_UF;
    rPUPC=save_UPC;
    rPUPE=save_UPE;
    rPUPF=save_UPF;
}

char Uart_IntGetkey(void)
{
    if(keyBufRdPt==KEY_BUFLEN)
        keyBufRdPt=0;

    while(keyBufWrPt==keyBufRdPt); //until FIFO is triggered
    return keyBuf[keyBufRdPt++];
}

/////////UART 0 TEST/////////

void Test_Uart0Fifo(void)
{
    int key;

    Uart_Port();
    keyBufRdPt=keyBufWrPt=0;
    pISR_UTXD0=(unsigned)Uart0_TxFifoInt;
    pISR_URXD0=(unsigned)Uart0_RxFifoInt;
    pISR_UERR01=(unsigned)Uart0_RxFifoErrorInt;

    /***** UART0 Tx FIFO test with interrupt *****/
    Uart_Printf("[Uart channel 0 tx FIFO Interrupt Test]\n");
    Uart_TxEmpty(0); //wait until tx shifter is empty.

    uart0TxStr="UART0 Tx FIFO interrupt test is good!!!!\r\n";
    rUFCON0=(2<<6)|(1<<4)|(6)|1;
    //FIFO trigger:tx/rx:8byte,tx/rx_fifo reset(will be cleared),FIFO enable.
    rUCON0 = 0x244; //tx:levl,rx:edge,error int,normal*2,interrupt(Start)

```

```

rINTMSK=~(BIT_GLOBAL|BIT_UTXD0);

Delay(500);
/***** UART0 Tx FIFO test with BDMA0 *****/
Uart_Init(0,115200);
Uart_Printf("\n[Uart0 FIFO Tx Test by BDMA0]\n");
uart0TxStr="UART0 Tx FIFO Test by BDMA0 is good!!!\r\n";
Uart_TxEmpty(0);

rUCON0=0x48;    //tx:BDMA0 rx:disable

rBDICNT0=0;
rBDCON0 =0x0;
rBDISRC0=(unsigned int)uart0TxStr|(0<<30)|(1<<28); // byte,inc
rBDIDES0=UTXH0 |(1<<30)|(3<<28); //L/B endian,M2IO,fix
rBDICNT0=strlen((char *)uart0TxStr)|(2<<30)|(1<<26)|(1<<20); //UART0,start,polling

while(!((rBDCON0&0x30)==0x20));
Uart_TxEmpty(0);

/***** UART0 Rx FIFO test with interrupt *****/
rUCON0=0x245|0x80;    //tx:level,rx:edge,tx/rx:int,rcv_time_out enabled,error int
enable

Uart_Printf("\n[Uart channel 0 FIFO Rx Interrupt Test]:Type any key!!!\n");
Uart_Printf("You have to see the typed character. To quit, press Enter key.\n");

rINTMSK=~(BIT_GLOBAL|BIT_URXD0|BIT_UERR01);

while( (rUFSTAT0&0xf) >0 )
    key=RdURXH0(); //To clear the Rx FIFO
rUERSTAT0;        //To clear the error state

while((key=Uart_IntGetkey())!='\r')
    Uart_SendByte(key);

rUFCON0=(2<<6)|(1<<4)|(6)|0;
//FIFO trigger:tx/rx:8byte, txrx_fifo reset(will be cleared), FIFO disable.

rINTMSK=~BIT_GLOBAL;
rUCON0=0x45;    //rcv_time_out disabled
Uart_Printf("\n");
Return_Port();
}

void __irq Uart0_TxFifoInt(void)
{
    while( !(rUFSTAT0 & 0x200) && (*uart0TxStr != '\0') )    //until tx fifo full or end of
string
    {
        rUTXH0=*uart0TxStr++;
        //for(i=0;i<700;i++); //to avoid overwriting FIFO
    }

    rI_ISPC=BIT_UTXD0;
    if(*uart0TxStr == '\0')
        SET_INTMSK_IRQ(BIT_UTXD0); //work around for ES2
}

void __irq Uart0_RxFifoInt(void)
{

```

```

    rI_ISPC=BIT_URXD0;
// if(rUFSTAT0==0)
//     Uart_Printf("time out\n");
while( (rUFSTAT0&0xf) >0 )    //until FIFO is empty
{
    keyBuf[keyBufWrPt++]=rURXH0;//rx buffer->keyBuf[]
    if(keyBufWrPt==KEY_BUFLEN)
        keyBufWrPt=0;
}
}

void __irq Uart0_RxFifoErrorInt(void)
{
    rI_ISPC=BIT_UERR01;

    Uart_Printf("UERSTAT0 = 0x%x\n", rUERSTAT0 & 0xf);
    while( (rUFSTAT0&0xf) >0 )
    {
        keyBuf[keyBufWrPt++]=rURXH0;
        if(keyBufWrPt==KEY_BUFLEN)
            keyBufWrPt=0;
    }
}

void Test_Uart0(void)
{
    int key;
    Uart_Port();
    keyBufRdPt=keyBufWrPt=0;
    pISR_UTXD0=(unsigned)Uart0_TxInt;
    pISR_URXD0=(unsigned)Uart0_RxInt;

    /***** UART0 Tx test with interrupt *****/
    Uart_Printf("[Uart channel 0 tx Interrupt Test]\n");
    Uart_TxEmpty(0); //wait until tx shifter is empty.

    uart0TxStr="UART0 Tx interrupt test is good!!!!\r\n";

    rUCON0 = 0x244; //tx:level,rx:edge,error int,normal*2,interrupt(Start)
    rINTMSK=~(BIT_GLOBAL|BIT_UTXD0);
//     rUCON0 &= 0x3f3;
//     rUCON0 |= 0x4; //needed to set the UTXD0 pending bit.

    Delay(3000);
//     rINTMSK =(BIT_GLOBAL|BIT_UTXD0);
//     rINTMSK=~BIT_GLOBAL;
//     Led_Display(0);
//     Led_Display(0xf);
//     return;

    /***** UART0 Tx test with BDMA0 *****/
    rUCON0 = 0x245;//workaround

    Uart_Printf("\n[Uart0 Tx Test by BDMA0]\n");
    uart0TxStr="UART0 Tx Test by BDMA0 is good!!!!\r\n";
    Uart_TxEmpty(0);

    rUCON0=0x48;    //tx:BDMA0 rx:disable

    rBDICNT0=0x0;
    rBDCON0 =0x0;

```

```

rBDISRC0=(unsigned int)uart0TxStr|(0<<30)|(1<<28); // byte,inc
rBDIDES0=UTXH0 |(1<<30)|(3<<28); //L/B endian,M2IO,fix
rBDCNT0=strlen((char *)uart0TxStr)|(2<<30)|(1<<26)|(1<<20); //UART0,

while(!((rBDCON0&0x30)==0x20));
Uart_TxEmpty(0);

/***** UART0 Rx test with interrupt *****/
rUCON0=0x45; //tx:int rx:int

Uart_Printf("\n[Uart channel 0 Rx Interrupt Test]:Type any key!!!\n");
Uart_Printf("You will see the typed character. To quit, press Enter key.\n");
Uart_TxEmpty(0);

rINTMSK=~(BIT_GLOBAL|BIT_URXD0);

keyBufRdPt=keyBufWrPt=0;
while((key=Uart_IntGetkey())!='\r')
    Uart_SendByte(key);

rINTMSK=~BIT_GLOBAL;
Uart_Printf("\n");
Return_Port();
}

void __irq Uart0_RxInt(void)
{
    rI_ISPC=BIT_URXD0;

    keyBuf[keyBufWrPt++]=RdURXH0();
    if(keyBufWrPt==KEY_BUFLen)
        keyBufWrPt=0;
}

//unsigned int txcount=0;
void __irq Uart0_TxInt(void)
{
    // rI_ISPC=BIT_UTXD0; //This cann't make the pending bit be cleared in the level trigger
    mode.

    if(*uart0TxStr != '\0')
    {
        WrUTXH0(*uart0TxStr++);
        rI_ISPC=BIT_UTXD0; //Because UART operates using level trigger mode.
    }
    else
    {
        rUCON0 &= 0x3f3; //workaround
        rI_ISPC=BIT_UTXD0; //This cann't make the pending bit be cleared in the level
        trigger mode.
        rINTMSK|=BIT_UTXD0;
    }
}

/////////UART 1 TEST/////////

void Test_Uart1Fifo(void)
{
    int key;
    Uart_Port();
    keyBufRdPt=keyBufWrPt=0;
    pISR_UTXD1=(unsigned)Uart1_TxFifoInt;

```

```

pISR_URXD1=(unsigned)Uart1_RxFifoInt;
pISR_UERR01=(unsigned)Uart1_RxFifoErrorInt;

/***** UART1 Tx FIFO test with interrupt *****/
Uart_Printf("[Uart channel 1 tx FIFO Interrupt Test]\n");
Uart_Printf("Plug the serial cable into ch1 connector!!! \n");
Uart_Printf("Then, press any key through UART ch1.\n");
Uart_Select(1);
Uart_Getch();

uart1TxStr="UART1 Tx FIFO interrupt test is good!!\r\n";
rUFCON1=(2<<6)|(1<<4)|(6)|1;
//FIFO trigger:tx/rx:8byte,tx/rx_fifo reset(will be cleared),FIFO enable.
rUCON1 = 0x244; //rx:edge,tx:level,error int,normal*2,interrupt(Start)
rINTMSK=~(BIT_GLOBAL|BIT_UTXD1);

Delay(500);
/***** UART1 Tx FIFO test with BDMA1 *****/
Uart_Init(0,115200);
Uart_Printf("\n[Uart1 FIFO Tx Test by BDMA1]\n");
uart1TxStr="UART1 Tx Test by BDMA0 is good!!!!\r\n";
Uart_TxEmpty(1);

rUCON1=0x4c; //tx:BDMA1 rx:disable

rBDICNT1=0;
rBDCON1 =0x0;
rBDISRC1=(unsigned int)uart1TxStr|(0<<30)|(1<<28); // byte,inc
rBDIDES1=UTXH1 |(1<<30)|(3<<28); //L/B endian,M2IO,fix
rBDICNT1=strlen((char *)uart1TxStr)|(2<<30)|(1<<26)|(1<<20); //UART1,start,polling

while((rBDCON1&0x30)==0x10);
Uart_TxEmpty(1);

/***** UART1 Rx FIFO test with interrupt *****/
rUCON1=0x245|0x80; //rx:edge,tx:level,tx/rx:int,rcv_time_out enabled,error int
enable

Uart_Printf("\n[Uart channel 1 FIFO Rx Interrupt Test]:Type any key!!!\n");
Uart_Printf("You have to see the typed character. To quit, press Enter key.\n");

rINTMSK=~(BIT_GLOBAL|BIT_URXD1|BIT_UERR01);
keyBufRdPt=keyBufWrPt=0;
while( (rUFSTAT1&0xf) >0 )
    key=RdURXH1(); //To clear the Rx FIFO
rUERSTAT1; //To clear the error state

while((key=Uart_IntGetkey())!='\r')
    Uart_SendByte(key);

rUFCON1=(2<<6)|(1<<4)|(6)|0;
//FIFO trigger:tx/rx:8byte,txrx_fifo reset(will be cleared), FIFO disable.

rINTMSK=~BIT_GLOBAL;
rUCON1=0x45; //rcv_time_out disabled
Uart_Printf("\n");

Uart_Printf("Plug the serial cable into ch0 as before this test!!!\n");
Uart_Printf("Then, press any key through UART ch 0.\n");
Uart_Select(0);
Uart_Getch();
Return_Port();
}

```

```

void __irq Uart1_TxFifoInt(void)
{
    int i;
    while( !(rUFSTAT1 == 16) && (*uart1TxStr != '\0') )    //until tx fifo full or end of
string
    {
        rUTXH1=*uart1TxStr++;
        for(i=0;i<700;i++); //to avoid overwriting FIFO
    }

    rI_ISPC=BIT_UTXD1;
    if(*uart1TxStr == '\0')
        SET_INTMSK_IRQ(BIT_UTXD1); //work around for ES2
}

void __irq Uart1_RxFifoInt(void)
{
    rI_ISPC=BIT_URXD1;
    // if(rUFSTAT1==0)
    //     Uart_Printf("time out\n");
    while( (rUFSTAT1&0xf) >0 )    //until FIFO is empty
    {
        keyBuf[keyBufWrPt++]=rURXH1; //rx buffer->keyBuf[]
        if(keyBufWrPt==KEY_BUFLLEN)
            keyBufWrPt=1;
    }
}

void __irq Uart1_RxFifoErrorInt(void)
{
    rI_ISPC=BIT_UERR01;

    switch(rUERSTAT1) //to clear and check the status of register bits
    {
        case '1':
            Uart_Printf("Overrun error\n");
            break;
        case '2':
            Uart_Printf("Parity error\n");
            break;
        case '4':
            Uart_Printf("Frame error\n");
            break;
        case '8':
            Uart_Printf("Breake detect\n");
            break;
        default :
            break;
    }
    while( (rUFSTAT1&0xf) >0 )
    {
        keyBuf[keyBufWrPt++]=rURXH1;
        if(keyBufWrPt==KEY_BUFLLEN)
            keyBufWrPt=0;
    }
}

void Test_Uart1(void)

```



```

{
    int key;
    Uart_Port();
    keyBufRdPt=keyBufWrPt=0;
    pISR_UTXD1=(unsigned)Uart1_TxInt;
    pISR_URXD1=(unsigned)Uart1_RxInt;

    /***** UART1 Tx test with interrupt *****/
    Uart_Printf("[Uart channel 1 tx Interrupt Test]\n");
    Uart_Printf("Plug the serial cable into ch1 connector!!! \n");
    Uart_Printf("Then, press any key through UART ch1.\n");
    Uart_Select(1);
    Uart_Getch();

    uart1TxStr="UART1 Tx interrupt test is good!!!!\r\n";
    rINTMSK=~(BIT_GLOBAL|BIT_UTXD1);
// rUCON1 &= 0x3f3;
// rUCON1 |= 0x4; //needed to set the UTXD0 pending bit.
    rUCON1 = 0x244; //rx:edge,tx:level,error int,normal*2,interrupt(Start)
    Delay(3000);

    /***** UART1 Tx test with BDMA1 *****/
    rUCON1 = 0x245;

    Uart_Printf("\n[Uart1 Tx Test by BDMA1]\n");
    uart1TxStr="UART1 Tx Test by BDMA1 is good!!!!\r\n";
    Uart_TxEmpty(1);

    rUCON1=0x4c;    //tx:BDMA0 rx:disable

    rBDICNT1=0x0;
    rBDCON1 =0x0;
    rBDISRC1=(unsigned int)uart1TxStr|(0<<30)|(1<<28); // byte,inc
    rBDIDES1=UTXH1 |(1<<30)|(3<<28); //L/B endian,M2IO,fix
    rBDICNT1=strlen((char *)uart1TxStr)|(2<<30)|(1<<26)|(1<<20); //UART1,

    while(!((rBDCON1&0x30)==0x20));
    Uart_TxEmpty(1);

    /***** UART1 Rx test with interrupt *****/
    rUCON1=0x45;    //tx:int rx:int
    Uart_Printf("\n[Uart channel 1 Rx Interrupt Test]:Type any key!!!\n");
    Uart_Printf("You have to see the typed character. To quit, press Enter key.\n");

    rINTMSK=~(BIT_GLOBAL|BIT_URXD1);

    keyBufRdPt=keyBufWrPt=0;
    while((key=Uart_IntGetkey())!='\r')
        Uart_SendByte(key);

    rINTMSK=~BIT_GLOBAL;
    Uart_Printf("\n");

    Uart_Printf("Plug the serial cable into ch0 as before this test!!!\n");
    Uart_Printf("Then, press any key through UART ch 0.\n");
    Uart_Select(0);
    Uart_Getch();
    Return_Port();
}

void __irq Uart1_RxInt(void)
{

```

```

    rI_ISPC=BIT_URXD1;

    keyBuf[keyBufWrPt++]=RdURXH1();
    if(keyBufWrPt==KEY_BUFLen)
        keyBufWrPt=0;
}

void __irq Uart1_TxInt(void)
{
    //    rI_ISPC=BIT_UTXD1;

    if(*uart1TxStr != '\0')
    {
        WrUTXH1(*uart1TxStr++);
        rI_ISPC=BIT_UTXD1;
    }
    else
    {
        rUCON1 &= 0x3f3; //workaround
        rI_ISPC=BIT_UTXD1;
        rINTMSK|=BIT_UTXD1;
    }
}

/////////Auto Flow Control TEST(Tx)/////////

volatile unsigned char * volatile tx0,* tx1,*tx2,tx_end=0;
volatile int i;
volatile int tx_cnt=0;
void Test_UartAFC_Tx(void)
{
    Uart_Port();
    tx_cnt=0;

    tx0=(unsigned char *)malloc(AFC_BUFLen);
    tx1=tx0;
    tx2=tx0;
    Uart_Printf("!!!tx0=0x%x\n",tx0);

    for(i=0;i<AFC_BUFLen;i++)
        *tx1++=i;

    pISR_UTXD1=(unsigned)U1AFC_TxInt;

    //    Led_Display(0xf);    //LED off

    Uart_Printf("[UART AFC Tx Test]\n");
    Uart_Printf("This test should be configured two boards.\n");
    Uart_Printf("Connect twitsted(rx/tx, nCTS/nRTS) cable com2 to other board's com1
port.\n");
    Uart_Printf("Start Rx first and press any key and,.\n");
    Uart_TxEmpty(0);
    Uart_Getch();

    rUCON1=(1<<9)|(0<<8)|(0<<7)|(0<<6)|(0<<5)|(0<<4)|(1<<2)|(1);
    rUFCON1=0x0;    //FIFO disable
    rUMCON1=0x10;    //Uart1 AFC enable

    Uart_Printf("Now... Tx with AFC\n");
    Uart_TxEmpty(0);
    rINTMSK=~(BIT_GLOBAL|BIT_UTXD1);

```

```

    while(!tx_end);

    Led_Display(0x0);        //LED on
    Uart_Printf("\nEnd Tx, transfer data count=%d\n",tx_cnt);

    Return_Port();
    free((void *)tx0);
    tx_end=0;
    tx_cnt=0;
}

void __irq U1AFC_TxInt(void)
{
    if(tx_cnt < (AFC_BUFLen))
    {
        Uart_Printf("%d,",*tx2);
        WrUTXH1(*tx2++);
        rI_ISPC=BIT_UTXD1;
        tx_cnt++;
    }
    else
    {
        Uart_TxEmpty(1);
        rUCON1 &= 0x3f3;//workaround
        rI_ISPC=BIT_UTXD1;
        rINTMSK|=BIT_UTXD1;
        tx_end=1;
    }
}

/////////Auto Flow Control TEST(Rx with FIFO)/////////

volatile unsigned char *rx0,*rx1,*rx2,rx_end=0;
volatile int rx_cnt=0;

void Test_UartAFC_Rx(void)
{
    unsigned int i;
    unsigned int err_cnt=0;

    Uart_Port();

    Uart_Printf("[UART AFC Rx Test]\n");
    Uart_Printf("This test should be configured two boards.\n");
    Uart_Printf("Connect twisted(rx/tx, nCTS/nRTS) cable com1 to other board's com2
port.\n");
    Uart_Printf("Then, change connected cable between host and com1, host to com2.\n");
    Uart_Printf("Press any key to com2 and start Rx first.\n");
    Uart_TxEmpty(0);

    Uart_Select(1);
    Uart_Getch();

    rx0=(unsigned char *)malloc(AFC_BUFLen);
    rx1=rx0;
    rx2=rx0;

    pISR_URXD0=(unsigned)U0AFC_RxInt;
    pISR_UERR01=(unsigned)U0AFC_RxErrorInt;

    Led_Display(0xf);        //LED off

```

```

rUMCON0=0x10;    //Uart0 AFC enable
rUCON0=(1<<9)|(0<<8)|(1<<7)|(1<<6)|(0<<5)|(0<<4)|(1<<2)|(1);
rUFCON0=(2<<6)|(1<<4)|(1<<2)|(1<<1)|(1);

Uart_Printf("Now... Rx with AFC\n");
Uart_TxEmpty(1);

rINTMSK=~(BIT_GLOBAL|BIT_URXD0|BIT_UERR01);

while(!rx_end);
Delay(1000);

rINTMSK=BIT_GLOBAL;

Uart_Printf("\nEnd Rx, receive data count=%d\n",rx_cnt);
Led_Display(0x0);    //LED on
Uart_Printf("Now, change connected cable between host and com2, host to com1.\n");
Uart_Printf("Then, press any key.\n");
Uart_Init(0,115200);
Uart_Select(0);
Uart_Getch();

for(i=0;i<AFC_BUFLLEN;i++)
{
    if(i-(*rx1++))
    {
        Uart_Printf("i=%d\n",i);
        err_cnt++;
    }
}
if(err_cnt)
    Uart_Printf("AFC test fail!! Error count=%d\n",err_cnt);
else
    Uart_Printf("AFC test is good!!\n");

Return_Port();
free((void *)rx0);
rx_end=0;
rx_cnt=0;
}

void __irq U0AFC_RxInt(void)
{
    rI_ISPC=BIT_URXD0;

    while( (rUFSTAT0 & 0x100) || ((rUFSTAT0 & 0xf) >0) )
    {
        Delay(1000);
        *rx2++=rURXH0;
        Uart_Printf("%d,",*(rx2-1));
        rx_cnt++;
    }
    if(rx_cnt == (AFC_BUFLLEN))
        rx_end=1;
}

void __irq U0AFC_RxErrorInt(void)
{
    rI_ISPC=BIT_UERR01;

```

```

switch(rUERSTAT0)//to clear and check the status of register bits
{
    case 1:
        WrUTXH1('!');
        break;
    case 2:
        WrUTXH1('#');
        break;
    case 4:
        WrUTXH1('$');
        break;
    case 8:
        WrUTXH1('@');
        break;
    default :
        WrUTXH1('*');
        break;
}

char _done=0, error=0;
void Test_BDMA(void)
{
    char *_buf,i;
    char *_temp2;
    int *_temp;

    _buf=(char *)malloc(100);
    _temp=(int *)malloc(1);
    _temp2=_buf;

    rINTMSK=~(BIT_GLOBAL|BIT_BDMA0|BIT_UERR01);
    pISR_BDMA0=(unsigned)Test_Done;
    pISR_UERR01=(unsigned)Error;
    Uart_Init(0,115200);
    Uart_Printf("[Read BDCON0 register in Rxing...]\n");
    Uart_TxEmpty(0);

    rBDISRC0=(0<<30)+(3<<28)+(int)URXH0;    //byte,inc,Rx-buf
    rBDIDES0=(2<<30)+(1<<28)+(int)_buf;    //M2IO,fix,IISFIF
    rBDICNT0=(2<<30)+(1<<26)+(3<<22)+(1<<21)+(1<<20)+700;
    //Uart0,reserve,done_int,auto-reload/start,DMA enable,COUNT
    rBDCON0 = 0x0<<2;

    rUCON0=0x2c6;    //tx:polling rx:BDMA0

    while(!_done)
    {
        *_temp=rBDCON0;
        if((rBDCON0 & 0xf))
        {
            Uart_Printf("!!Error0x%x!!",rBDCON0);
            break;
        }
    }
    Uart_Printf("!END!\n");

    if(error)
        Uart_Printf("[rUERSTAT=0x%x]\n",rUERSTAT0);
}

```