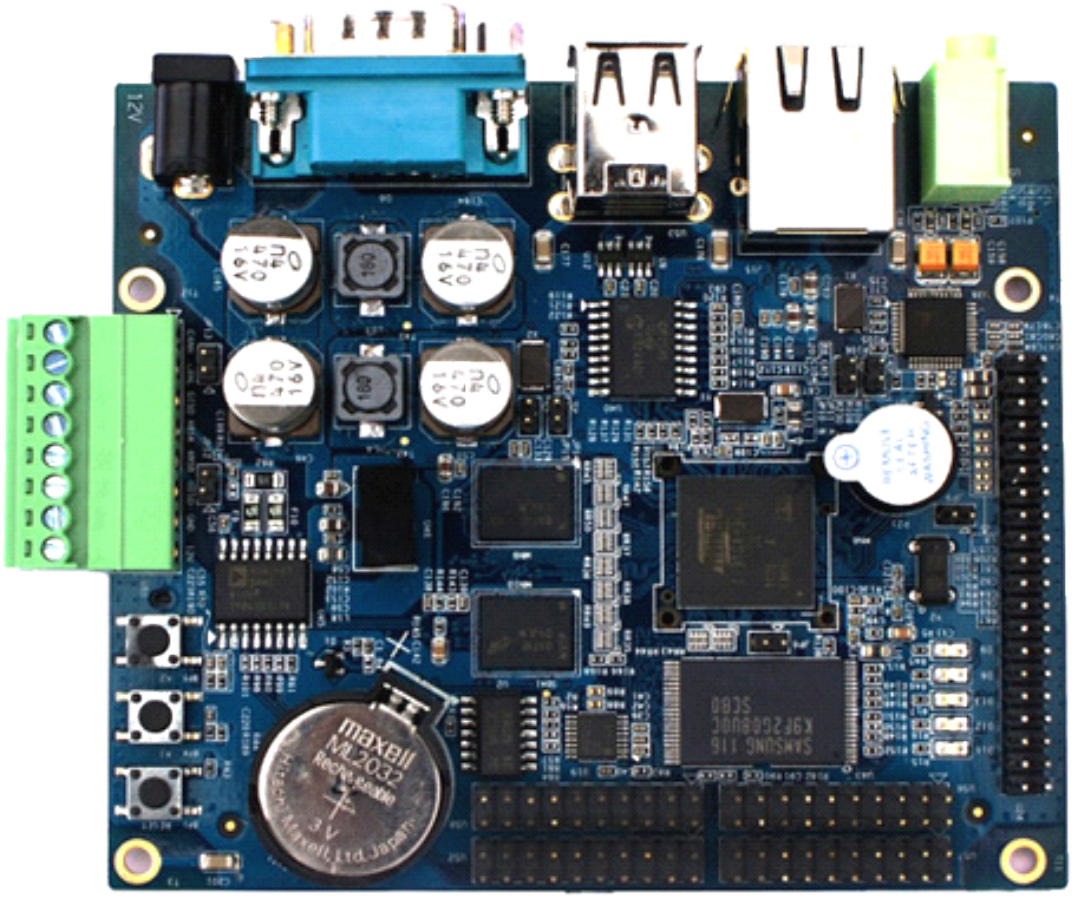
SBC6845
Single Board Computer



User Manual

(For Linux System)

Version 1.1 – Jan.11.2013

Copyright Statement:

- SBC6845 and its related intellectual property are owned by Shenzhen Embest Technology Co., Ltd.
- Shenzhen Embest Technology has the copyright of this document and reserves all rights. Any part of the document should not be modified, distributed or duplicated in any approach and form with the written permission issued by Embest Technology Co., Ltd.

Disclaimer:

- Shenzhen Embest Technology does not take warranty of any kind, either expressed or implied, as to the program source code, software and documents in the CD/DVD-ROMs provided along with the products, and including, but not limited to, warranties of fitness for a particular purpose; The entire risk as to the quality or performance of the program is with the user of products; Should the program prove defective, the user of products assumes the cost of all necessary servicing, repair or correction.

Revision History:

Version	Date	Note
1.0	2012-11-16	Original Version
1.1	2013-01-11	First Revision

Table of Contents

Chapter 1 Product Overview

1.1 Introduction

SBC6845 is an ARM embedded single board computer designed by Shenzhen Embest Technology Co., Ltd. It is a small-sized board based on Atmel's industrial microprocessor AT91SAM9G45 and featured with 128MB DDR2 SDRAM, 256MB NAND Flash and 4MB Data Flash, as well as abundant interfaces including 5 RS232 serial interfaces (COM2 is RS485), a CAN interface, an Ethernet interface, a high-speed USB host interface, a SD/MMC card interface and an audio output interface.

SBC6845 is designed to satisfy the different requirements of various fields such as industrial control, intelligent instrumentation, data acquisition and analysis, medical products and network equipments.

1.2 Packing List

- A SBC6845 board
- A serial cable
- A network cable
- A USB cable
- A 12V power adapter
- A LCD touch-screen (optional item, available in 4.3-inch 480*272 or 7-inch 800*480)
- A CD-ROM

1.3 Product Features

- Dimensions: 106.5mm x 94mm (6-layer PCB)
- Operation temperature: -40 ~ +85℃

- Operating Humidity: 0% ~ 90%
- Power Supply: 12V/1.25A
- Processor: Atmel AT91SAM9G45
- On-Board Memories:
  - 64MB*2 DDR2 SDRAM
  - 256MB NAND Flash
  - 4MB Data Flash
- Audio/Video Output Interfaces:
  - A 3.5mm jack
  - A 2*20-pin DIP interface for LCD touch-screen
  - A buzzer
- Data Transfer Interfaces:
  - 5 serial interfaces
  - A CAN 2.0 interface (isolated)
  - 2 USB 2.0 interfaces
  - A 10/100Mb Ethernet interface
  - A SD card slot (hot plugging)
  - Pins for SPI, I2C, PWM, ADC, Keypad and GPIO

1.4 Components on SBC6845

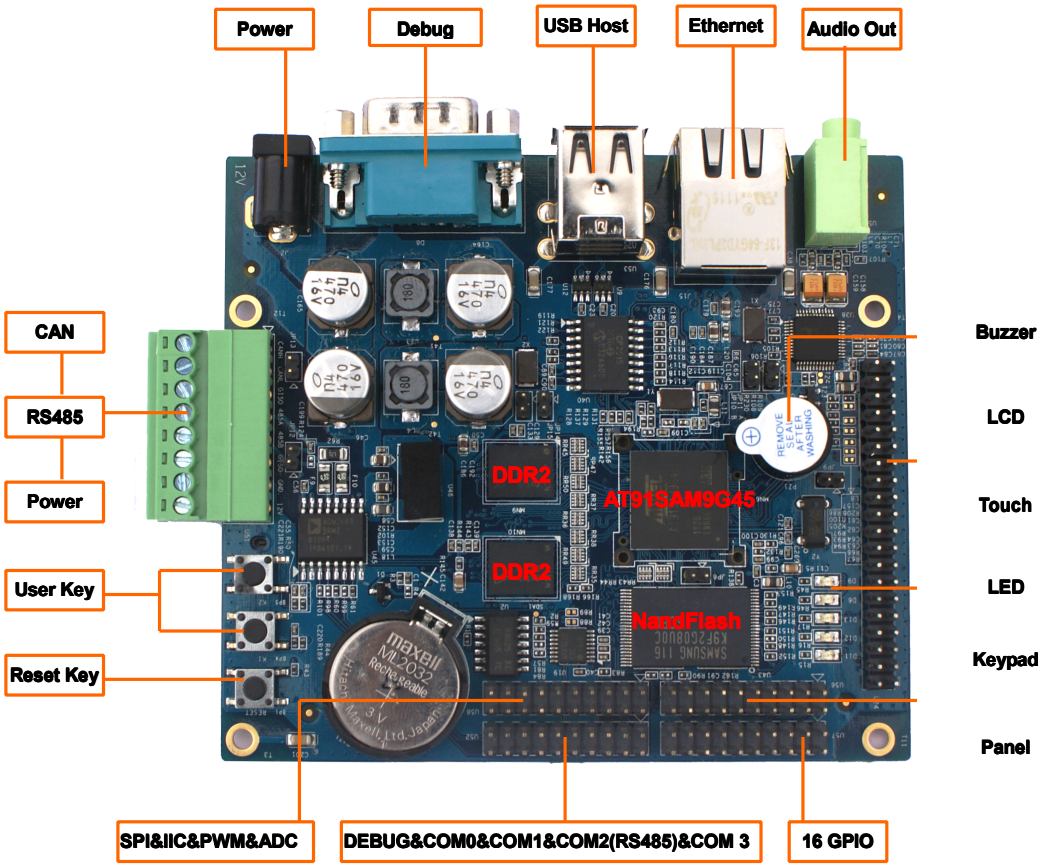


Figure 1-1 Components on SBC6845

1.5 Hardware Dimensions

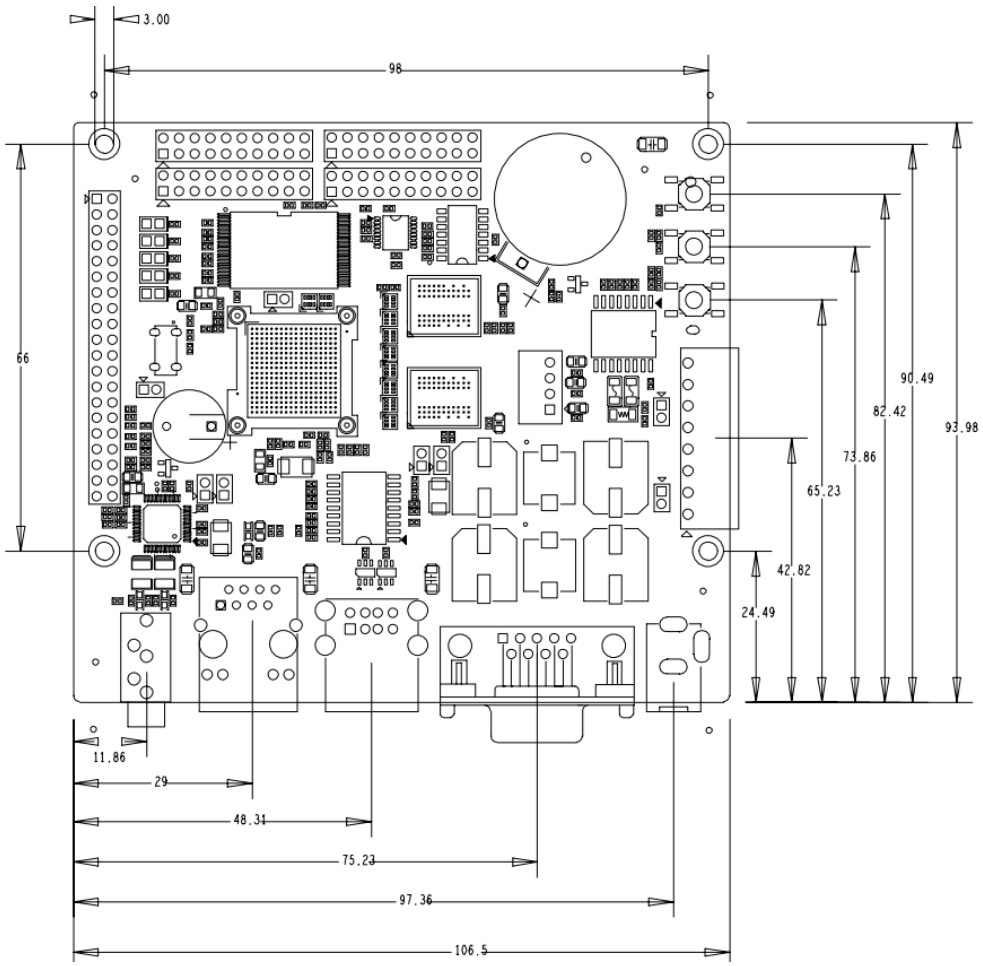


Figure 1-2 Hardware Dimensions

1.6 BSP Package in CD-ROM

The CD-ROM provided along with SBC6845 contains a BSP package that helps development the Linux system. Users can make a custom system running on SBC6845 platform by utilizing the package. The table shown below lists out the contents of the BSP attached with the corresponding descriptions.

Table 1-1 BSP Package

Categories	Names	Descriptions
BIOS	Bootstrap	DATA FLASH
	u-boot	DATA FLASH

Categories	Names	Descriptions
		Supports downloading kernel and file system through SAM-BA and network
Kernel	Linux-2.6.30	ROM/CRAM/EXT2/EXT3/FAT/NFS/JFFS2/YAFFS2 file systems
Device Drivers	Serial	Drivers for 4 serial interfaces and a debug interface built in CPU
	RTC	AT91SAM9G45 extension RTC driver
	NET	10/100M Ethernet driver
	Flash	NAND FLASH and DATA FLASH drivers
	LCD	LCD driver supporting 480x272, 640x480, 800x480 and 800x600
	Touch Screen	Driver for touch-screen controller built in CPU
	USB Host	USB Host driver
	Watchdog	Built-in watchdog driver
	SD Card	SD card driver
	CAN 总线	Driver for an extension CAN bus
	RS-485	Driver for an extension RS-485 bus
	LED	Driver for 4 LED indicators
	BEEP	Buzzer driver
	Sound	AC97 Codec audio driver
	Button	Driver for 2 custom keys
	Keypad	Driver for 6x6 matrix keyboard
	GPIO	Driver for 16 GPIO
	ADC	Driver for 10bit ADC
Root Filesystem	YAFFS2	Readable and writable filesystem
Graphic Interface	Qt	Version 4.4.2

Chapter 2 System Boot-up and Testing

2.1 System Boot-up

SBC6845 can support boot-up from Data Flash only. A default Linux system image has been burned in Data Flash when it is manufactured. If you need to update the image, please refer to the contents of Chapter 5 Updating Linux System.

Before you power up the board, you need to set the video output resolution of the board by applying a specific jumper configuration on SBC6845. The table shown below lists different configurations of the jumpers.

Table 2-1 Jumper Configurations

JP11	JP10	Sizes of LCD	Resolutions of LCD
0	0	4.3 inches	480x272
0	1	5.6 inches	640x480
1	0	7.0 inches	800x480
1	1	10.4 inches	800x600

Additionally, you need to set up a HyperTerminal on your PC according to the serial interface configuration window shown below.

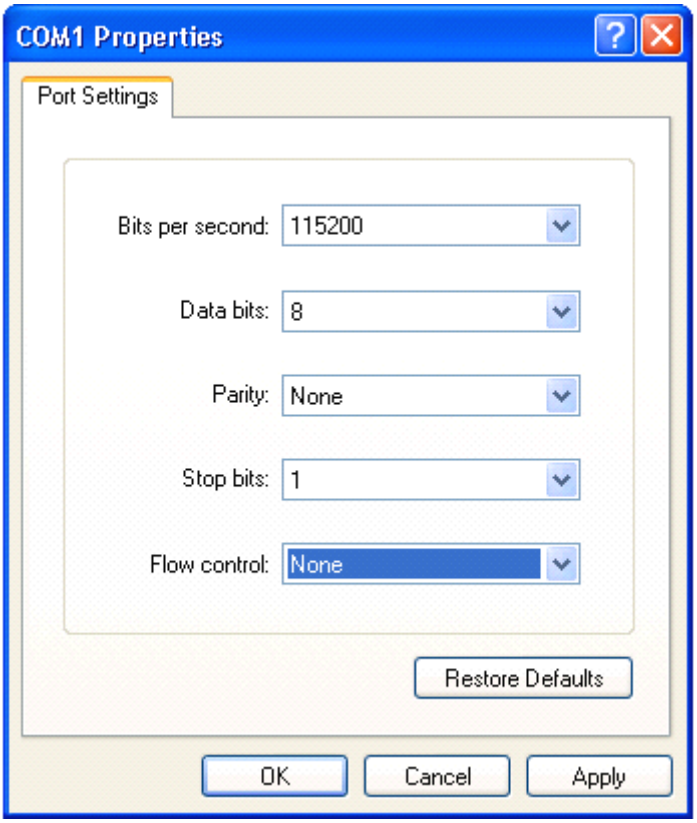
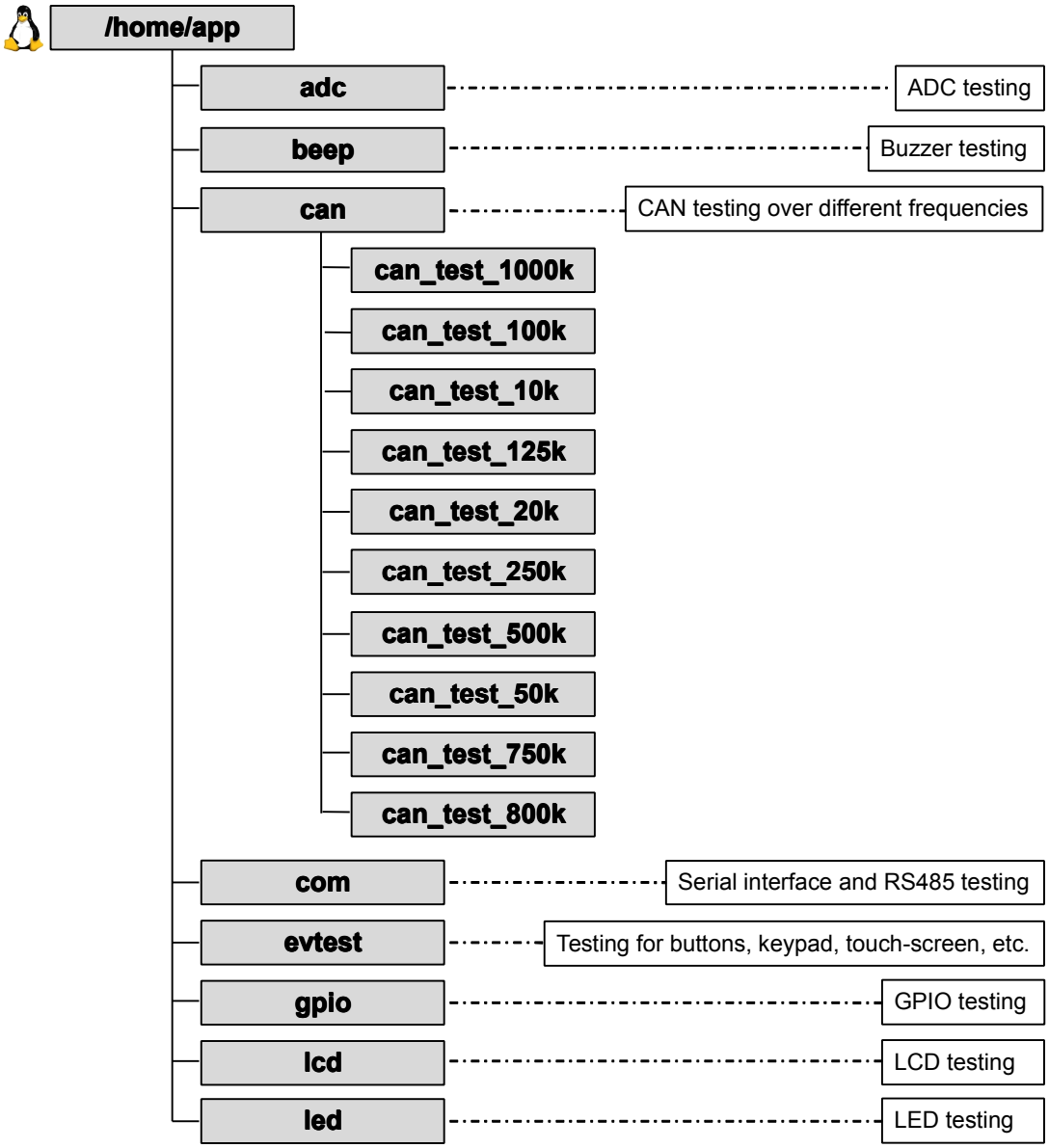


Figure 2-2 Setting up HyperTerminal

Now you can connect the power supply to SBC6845 and see the boot-up information displayed in HyperTerminal.

2.2 Example Applications

There are lots of example applications can be found under /home/app of the default Linux system on SBC6845. Users can test or demonstrate the functions of board by taking advantage of these programs. The diagram shown below can help you find them more easily.


/home/app
adc
ADC testing
beep
Buzzer testing
can
CAN testing over different frequencies
can_test_1000k
can_test_100k
can_test_10k
can_test_125k
can_test_20k
can_test_250k
can_test_500k
can_test_50k
can_test_750k
can_test_800k
com
Serial interface and RS485 testing
evtest
Testing for buttons, keypad, touch-screen, etc.
gpio
GPIO testing
lcd
LCD testing
led
LED testing

2.3 Introduction to API

Before you get started with testing, it's necessary to learn something about the API functions used by example applications. If you'd like to look into the details of how the programs are working, please view the source code under \Linux 2.6 Kit\01 SourceCode\application\ in the SBC6845's CD-ROM.

The tables listed below are the API functions called by some of the example applications we provided in the CD-ROM, as well as the descriptions attached to each function.

Table 2-2 LED API Function

API Function for LED	
LED_API int led_ctrl (char *name, int onoff);	
Source	ledlib.h
Function	Turn on/off LED indicators
Parameters	name (LED name, e.g. D6, D9 or D13)
	on/off (0 for off, 1 for on)
Returned Value	0 for success, otherwise failure
Example	led_ctrl ("D9", 1);

Table 2-3 Buzzer API Function

API Function for Buzzer	
BEEP_API int beep_ctrl (char *name, int onoff);	
Source	beeplib.h
Function	control buzzer to make sound or keep silent
Parameters	name (buzzer name, normally there is only one buzzer which is called "beep")
	on/off (0 for off, 1 for on)
Returned Value	0 for success, otherwise failure
Example	beep_ctrl ("beep", 1); beep_ctrl ("beep", 0);

Table 2-4 Serial Interface API Function

API Function for Serial Interface	
int OpenDev(char *Dev);	
Source	com_example.c
Function	Enable serial devices and acquire descriptors
Parameter	dev (character string of serial devices, e.g. “/dev/ttySAC0”)
Returned Value	Value more than 0 is a serial file descriptor, less than 0 stands for failure
void set_speed(int fd, int speed);	
Source	com_example.c
Function	Configure bitrate of serial interfaces
Parameter	fd (serial file descriptor) speed (bitrate, e.g. 15200)
Returned Value	None
int set_Parity (int fd,int databits,int stopbits,int parity,int flowctrl);	
Source	com_example.c
Function	Configure serial interface data bits, stop bits, parity check and data flow control
Parameters	fd (serial file descriptor) databits (length of data bits) stopbits (length of stop bits) parity (check type, N for no check, O for odd check, E for even check) flowctrl (switch of hardware data follow control, 1 for enable, 0 for disable)
Returned Value	0 for success, otherwise failure
size_t read(int fd, const void *buf, size_t nbytes);	
Source	unistd.h
Function	Called by system to acquire data received on serial interface
Parameters	fd (serial file descriptor) Buf (pointer to the received data) nbytes (data length about to be read, Byte)
Returned Value	Value less than 0 stands for error, more than 0 stands for received data length (Byte)
size_t write(int fd, const void *buf, size_t nbytes);	
Source	unistd.h
Function	Called by system to send data through serial interfaces
Parameters	fd (serial file descriptor) buf (pointer to the data about to be sent) nbytes (length of data about to be sent, Byte)

API Function for Serial Interface	
int OpenDev(char *Dev);	
Returned Value	Value less than 0 stands for error, more than 0 stands for data length being sent (Byte)
int close(int fd);	
Source	unistd.h
Function	Called by system to disable serial interface
Parameters	fd (serial file descriptor)
Returned Value	0 for success, less than 0 stands for error

Table 2-5 GPIO API Function

API Function for GPIO	
GPIO_API int init_gpio (void);	
Source	gpiolib.h
Function	Initializing GPIO
Parameters	None
Returned Value	0 for success, otherwise failure
GPIO_API int remove_gpio (void);	
Source	gpiolib.h
Function	Release gpio
Parameters	None
Returned Value	0 for success, otherwise failure
GPIO_API int set_gpio_output (char *pin, int value);	
Source	gpiolib.h
Function	Provide a logic level output on GPIO interfaces
Parameters	pin (GPIO pin name, e.g. “pb29” value (0 for low level, 1 for high level)
Returned Value	0 for success, otherwise failure
GPIO_API int get_gpio_input (char *pin);	
Source	unistd.h
Function	Acquire the logic level input on GPIO interfaces
Parameters	pin (GPIO pin name, e.g. “pb29”
Returned Value	0 or 1 for input level, other values for input failure

Table 2-6 PWM API Function

API Function for PWM	
PWM_API int pwm_ctrl (char *name, int ratio);	
Source	pwmlib.h
Function	Control the duty cycle of PWM output
Parameters	name (pin name, e.g. “PD26”) ratio (integer between 0～255, duty cycle =ratio / 256)
Returned Value	0 for success, otherwise error

2.4 Starting Testing

This section will show you how to use the example applications installed by default in the system to accomplish the testing of SBC6845’s functions. After you finish the configurations mentioned in **2.1 System Boot-up**, you can run the applications as shown in the following contents.

2.4.2 Touch-Screen Testing

1） Execute the following instruction to run touch-screen calibration program;

[root@SBC6845:/]# ts_calibrate

Press the symbol “+” shown on touch-screen for 5 times to accomplish screen calibration.

2） Execute the following instruction to test touch-screen;

[root@SBC6845:/]# ts_test

Select **Drag** or **Draw** to test dragging and drawing functions.

Press **Ctrl+C** on your PC’s keyboard to exit the example application.

2.4.3 LCD Color Testing

By executing the following instruction, LCD will display each of three-primary color respectively and then show them all together on the screen.

[root@SBC6845:/]# /home/app/lcd

2.4.4 LCD Backlight Testing

1） Execute the following instruction to adjust backlight brightness; the brightness value should be an integer between 1~10;

[root@SBC6845:/]# bl_adjust SET 5

2） Execute the following instruction to turn off backlight;

[root@SBC6845:/]# bl_adjust OFF

3） Execute the following instruction to turn on backlight;

[root@SBC6845:/]# bl_adjust ON

2.4.5 Ethernet Testing

1） Execute the following instruction to set an IP address for SBC6845;

[root@SBC6845:/]# ifconfig eth0 192.192.192.200

2） Execute the following instruction to test network communication;

[root@SBC6845:/]# ping 192.192.192.105

Note:

The IP in the above instructions are just examples. Please set IP address according to your network configurations and ensure that SBC6845 and your PC are set in the same network segment.

The table shown below is the testing information in HyperTerminal window.

Table 2-7 Ethernet Testing

PING 192.192.192.105 (192.192.192.105): 56 data bytes
64 bytes from 192.192.192.105: icmp_seq=0 ttl=64 time=0.5 ms
64 bytes from 192.192.192.105: icmp_seq=1 ttl=64 time=0.3 ms
64 bytes from 192.192.192.105: icmp_seq=2 ttl=64 time=0.3 ms
64 bytes from 192.192.192.105: icmp_seq=3 ttl=64 time=0.3 ms
64 bytes from 192.192.192.105: icmp_seq=4 ttl=64 time=0.3 ms
64 bytes from 192.192.192.105: icmp_seq=5 ttl=64 time=0.3 ms
64 bytes from 192.192.192.105: icmp_seq=6 ttl=64 time=0.3 ms

```
--- 192.192.192.105 ping statistics ---
7 packets transmitted, 7 packets received, 0% packet loss
round-trip min/avg/max = 0.3/0.3/0.5 ms
~ $
```

Press **Ctrl+C** on your PC's keyboard to exit Ethernet testing.

2.4.6 Serial Interface Testing

SBC6845 has 5 serial interfaces - ttySAC0, ttySAC1, ttySAC2, ttySAC3 and ttySAC6. ttySAC2 is defined as RS-485 and ttySAC6 is used for debugging. Please execute the following instruction to test serial interfaces.

[root@SBC6845:/home/app]# ./com -d /dev/ttySAC1 -s 1234567890 -b 115200 –f

Table 2-8 Command Parameters

Parameters	Descriptions
-d	Serial device node used to specify serial interfaces
-s	Character string to be sent
-b	Bitrate
-f	Enable hardware data flow control

Note:

- ttySAC1 requires hardware data flow control.
- The internal pull-up resistor of serial interface is enabled by default by the driver in order to ensure the quality and reliability of communication.
- Please be aware of the SPACEs in the instruction; Missing of any space would cause failure in executing instructions.

2.4.7 CAN Bus Testing

SBC6845 integrates only one CAN bus, and therefore it needs to work with another SBC6845 or a CAN-enabled device to test the data transfer over CAN bus. The figure shown below illustrates the connections between SBC6845 and a CAN-enabled device. **Please be aware that jumper JP13 is shorted.**

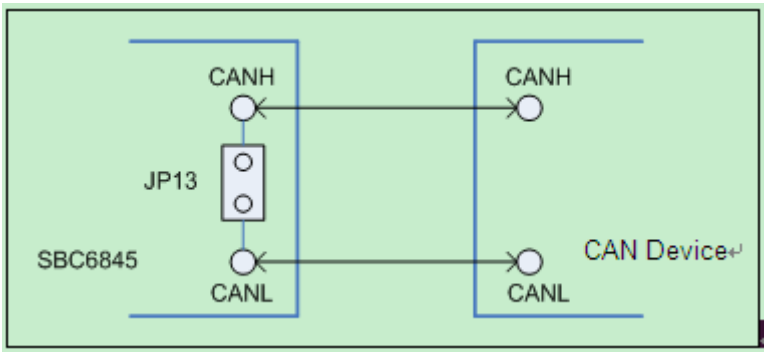


Figure 2-3 CAN Bus Connections

Please execute the following instructions;

```
[root@SBC6845:/]# cd /home/app/can/

[root@SBC6845:/home/app/can]# ./can_test_125k /dev/can0
```

The HypterTerminal will display the testing information as shown in the following table.

Table 2-9 CAN Testing Information

```
/dev/can0 Work Rate: 125000bps
/dev/can0 work on normal mode.
Press "\q" to quit!
Input the message and press Enter to send!
The frame id=1...2031, and eid=id*2+1
CanTest> TestData
CanTest>
Header: id=1 srr=0 ide=0 eid=0 rtr=0 rb1=0 rb0=0 dlc=8
RECV: "TestData"
```

Press **Ctrl+C** on your PC's keyboard to exit testing.

2.4.8 RS485 Bus Testing

RS485 interface is associated to the device /dev/ttySAC2. Similar to CAN bus testing, two RS485-enabled devices are required to accomplish the testing as shown below.

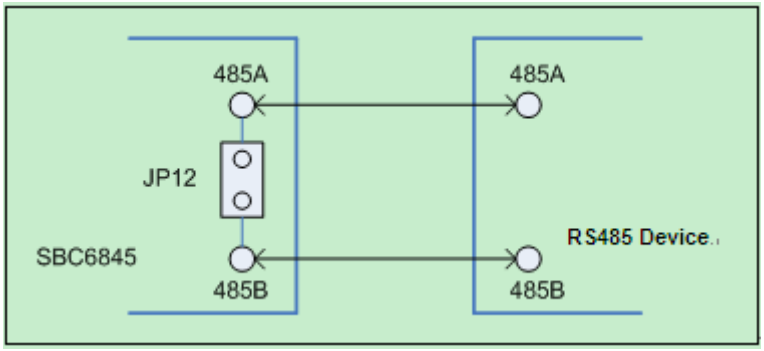


Figure 2-4 RS485 Connections

Note:

📖 In order to eliminate signal reflection occurs in long-distance transmission, the terminal resistor should be enabled by shorting the jumper JP12 on SBC6845. Short-distance (less than 300 meters) transmission can work well without terminal resistor.

2.4.9 USB Testing

SBC6845 has a USB host interface. Please insert a flash disk and see the information in HyperTerminal window as shown below.

Table 2-10 USB Connection Information

```
usb 1-1: USB disconnect, address 2
usb 1-1: new full speed USB device using at91_ohci and address 3
usb 1-1: configuration #1 chosen from 1 choice
scsi2 : SCSI emulation for USB Mass Storage devices
scsi 2:0:0:0: Direct-Access     Generic  USB  SD Reader    0.00 PQ: 0 ANSI: 2
sd 2:0:0:0: [sda] 7744512 512-byte hardware sectors (3965 MB)
sd 2:0:0:0: [sda] Write Protect is off
sd 2:0:0:0: [sda] Assuming drive cache: write through
sd 2:0:0:0: [sda] 7744512 512-byte hardware sectors (3965 MB)
sd 2:0:0:0: [sda] Write Protect is off
sd 2:0:0:0: [sda] Assuming drive cache: write through
 sda: sda1
sd 2:0:0:0: [sda] Attached SCSI removable disk
sd 2:0:0:0: Attached scsi generic sg1 type 0
```

The above information shows that the flash disk has been defined as sda1 device.

Please execute the following instructions to conduct testing.

1） Mount the flash disk to the directory /mnt, and specify VFAT as the disk format;

[root@SBC6845:/]# mount -t vfat /dev/sda1 /mnt/

2） View the contents of the flash disk;

[root@SBC6845:/]# cd /mnt/

[root@SBC6845:/mnt]# ls

3） Unmount the flash disk;

[root@SBC6845:/mnt]# cd /

[root@SBC6845:/]# umount /mnt/

Note:

- The udev rule of the Linux system that contained in the CD-ROM can automatically mount portable devices to the directory /media. However, manual mounting is still recommended in case the system fails to mount devices automatically.
- Please be aware of the SPACEs in the instruction; Missing of any space would cause failure in executing instructions.

2.4.10 RTC Testing

The RTC mentioned here refers to an individual clock chip working outside of the CPU to store and update system clock. Please follow the steps listed below to test the external RTC.

1） View current system clock;

[root@ SBC6845:/]# date

The system clock is shown below;

```
Thu May 27 11:48:02 UTC 2010
```

2） Set the system clock to 16:43, 29th Nov, 2012;

[root@ SBC6845:/]# date -s 112916432012

The new system clock is shown below;

```
Thu Nov 29 16:43:00 UTC 2012
```

3） Write the system clock into RTC;

[root@ SBC6845:/]# hwclock -w

4） View the clock of RTC;

[root@ SBC6845:/]# hwclock -r

The RTC clock is show below;

```
Thu Nov 29 16:43:07 2012  0.000000 seconds
```

5） Update the system clock with the information saved in RTC and view the system clock again;

[root@ SBC6845:/]# hwclock -s

[root@ SBC6845:/]# date

The updated system clock is shown below;

```
Thu Nov 29 16:43:45 UTC 2012
```

2.4.11 SD Card Testing

After inserting a SD card into the SD slot on SBC6845, HyperTerminal will display the following information;

Table 2-11 SD Card Recognized

```
[root@SBC6845:/]# mmc1: new SD card at address 0002
mmcblk0: mmc1:0002 N/A     489 MB
mmcblk0: p1
```

The above information shows that the SD card has been defined as mmcblk0p1 device. Please follow the steps listed below to conduct testing.

1） Mount SD card to the directory /mnt, and specify VFAT as the SD card format;

[root@SBC6845:/]# mount -t vfat /dev/mmcblk0p1 /mnt/

2） View the contents of SD card;

[root@SBC6845:/]# cd /mnt/

[root@SBC6845:/mnt]# ls

3） Unmount SD card;

root@SBC6845:/mnt]# cd /

[root@SBC6845:/]# umount /mnt/

Note:

- The udev rule of the Linux system that contained in the CD-ROM can automatically mount portable devices to the directory /media. However, manual mounting is still recommended in case the system fails to mount devices automatically.
- Please be aware of the SPACEs in the instruction; Missing of any space would cause failure in executing instructions.

2.4.12 LED Testing

SBC6845 has 4 LED indicators, among them the D12 defined as system state indictor. The following testing is intended for the three LEDs except D12.

1） Test LEDs with application;

[root@SBC6845:/]# /home/app/led

LED D9/D6/D13 will be turned on in sequence repeatedly.

2） Turn off a LED;

[root@SBC6845:/]# echo '0' >/sys/class/leds/d6/brightness

LED D6 will be turned off.

3） Turn on a LED;

[root@SBC6845:/]# echo '1' >/sys/class/leds/d6/brightness

LED D6 will be turned on.

2.4.13 Buzzer Testing

1） Test buzzer with application;

[root@SBC6845:/]# /home/app/beep

The buzzer will emit a single beep.

2） Emit a constant beep;

```
[root@SBC6845:/]# echo '1' >/sys/class/leds/beep/brightness
```

3） Stop beeping;

```
[root@SBC6845:/]# echo '0' >/sys/class/leds/beep/brightness
```

2.4.14 GPIO Testing

To test GPIO interfaces on SBC6845, you need to use jumper caps to short the pins of U57 connector on the board according to the following table and figure. (Please refer to **1.4 Components on SBC6845** to locate the position of U57 connector which is marked as “16 GPIO”)

Table 2-12 GPIO Pin Pairs

Pin Pairs to be Shorted	Pin Pairs to be Shorted
PA22 ----- PB29	PA28 ----- PD12
PA24 ----- PB30	PA29 ----- PD13
PA26 ----- PB23	PA30 ----- PD14
PA27 ----- PB24	PA31 ----- PD28

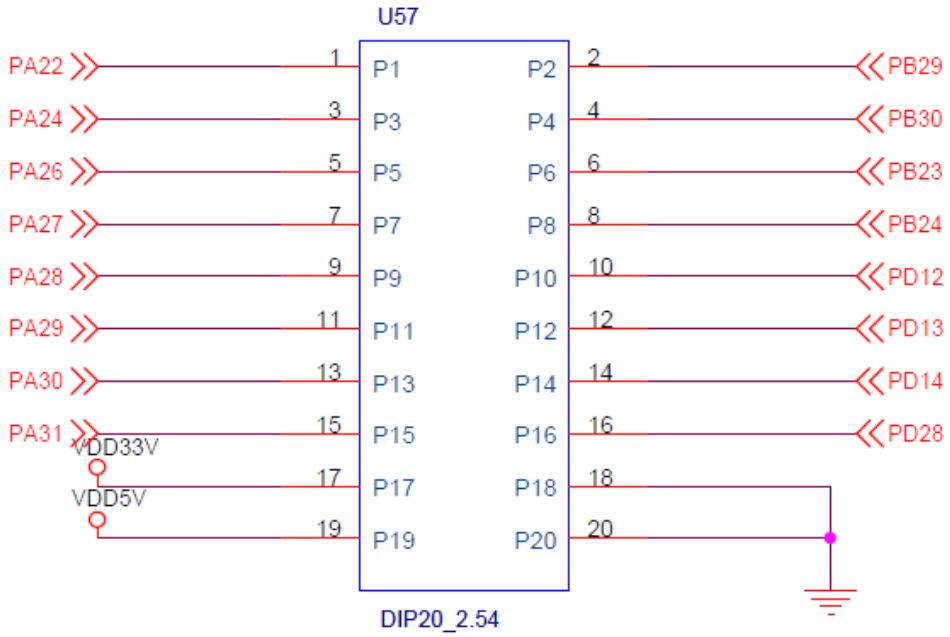


Figure 2-5 U57 Pin Definition

And then execute the following instruction to conduct testing.

[root@SBC6845:/]# /home/app/gpio

If testing successes, you will see the information in HyperTerminal window shown as below.

Table 2-13 GPIO Testing Successes

```
GPIO Test : OK
```

If testing fails, the input/output result on pin pairs will be shown in HyperTerminal window as the following information;

Table 2-14 GPIO Testing Fails

```
PA22..PA31: 0 1 0 0 0 0 0 0
PB29..PD28: 1 1 0 0 0 0 0 0
------------------------------
PA22..PA31: 0 1 0 0 0 0 0 0
PB29..PD28: 1 1 0 0 0 0 0 0
GPIO Test : FAIL
```

2.4.15 ADC Testing

The ADC input pin, which is named as PD27, can be found on U58 connector as shown below. (Please refer to **1.4 Components on SBC6845** to locate the position of U58 connector which is marked as “SPI&IIC&PWM&ADC”)

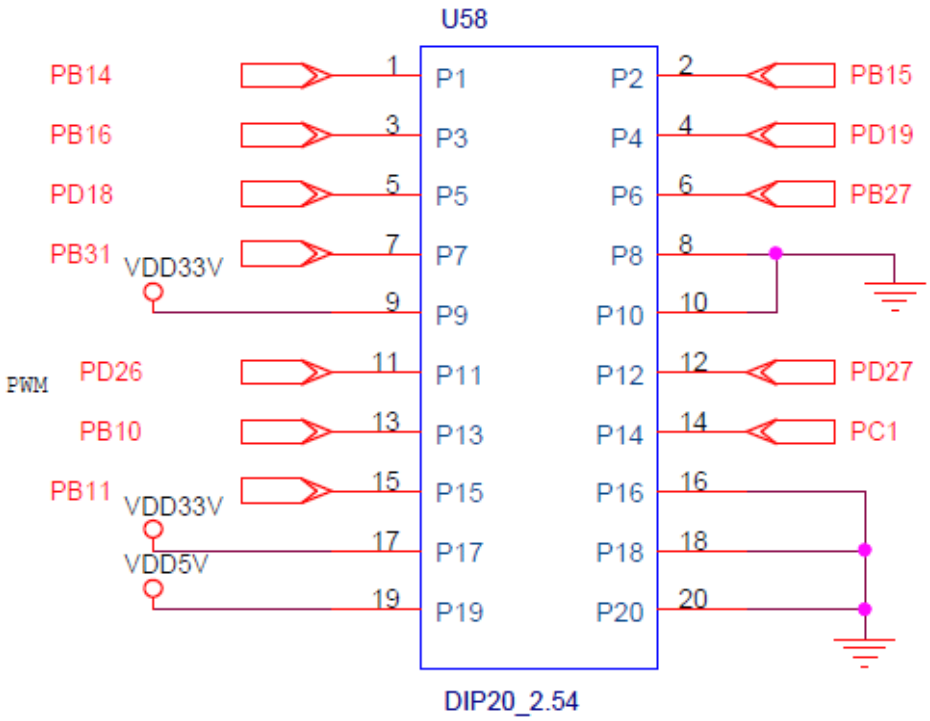


Figure 2-6 U58 Pin definition

Firstly put an analog voltage between 0~3.3V on PD27, and then execute the following instruction.

```
[root@SBC6845:/]# /home/app/adc
```

A series of integer values within 0~1023 (10bit) that are sampled by ADC will be displayed in HyperTerminal window.

Note:

🕮 An analog voltage higher than 3.3V may damage the board.

2.4.16 Button Testing

This testing is intended for the button K1 and K2, which are marked as “User Key” in the figure shown in **1.4 Components on SBC6845**). Please execute the following instruction to conduct testing.

```
[root@SBC6845:/]# /home/app/evtest /dev/event0
```

Information of HyperTerminal window is shown below;

Table 2-15 Running Button Testing Application

```
Input device ID: bus 0x19 vendor 0x1 product 0x1 version 0x100
Input device name: "gpio-keys"
Supported events:
  Event type 0 (Sync)
  Event type 1 (Key)
    Event code 158 (Back)
    Event code 279 (TaskBtn)
Testing ... (interrupt to exit)
```

Press K1 and K2 to view the information in HyperTerminal window as shown below;

Table 2-16 Testing Buttons

```
Input device ID: bus 0x19 vendor 0x1 product 0x1 version 0x100
Input device name: "gpio-keys"
Supported events:
  Event type 0 (Sync)
  Event type 1 (Key)
    Event code 158 (Back)
    Event code 279 (TaskBtn)
Testing ... (interrupt to exit)
```

Press **Ctrl+C** on your PC's keyboard to exit button testing.

Note:

Please be aware of the SPACEs in the instruction; Missing of any space would cause failure in executing instructions.

2.4.17 Keypad Testing

Please connect a 6X6 matrix keypad to U56 interface on SBC6845 and then execute the following instruction to conduct testing. (Please refer to **1.4 Components on SBC6845** to locate the position of U56 connector which is marked as "Keypad/Panel")

```
[root@SBC6845:/]# /home/app/evtest /dev/event2
```

event2 is a device code registered for keypad module. Press any key on the keypad to view the resulted event information in HyperTerminal window.

Press **Ctrl+C** on your PC's keyboard to exit keypad testing.

2.4.18 Screen Capture Testing

Execute the following instruction to capture the screen of LCD and save it as a JPG image.

```
[root@SBC6845:/]# fbcat /dev/fb0 Figure.jpg
```

The image is saved under the root directory of the system.

2.4.19 Audio Testing

By default, the system has a built-in open-source audio player "madplay" which support the playback of MP3 and WAV files. Please insert an earphone into the 3.5 mm audio output interface on SBC6845 and then execute the following instruction to conduct testing.

```
[root@SBC6845:/]# madplay /home/mp3/music.mp3
```

If you hear music, the audio playback function works properly. You can view the help information about the player by entering an instruction "madplay –h".

Note:

🕮 Please be aware of the SPACEs in the instruction; Missing of any space would cause failure in executing instructions.

2.4.20 Telnet Login Testing

Please use a network cable to connect SBC6845 to your LAN and then follow the steps listed below to complete testing.

1） Select **Start > Run** on your PC's desktop to open command line window as shown below;

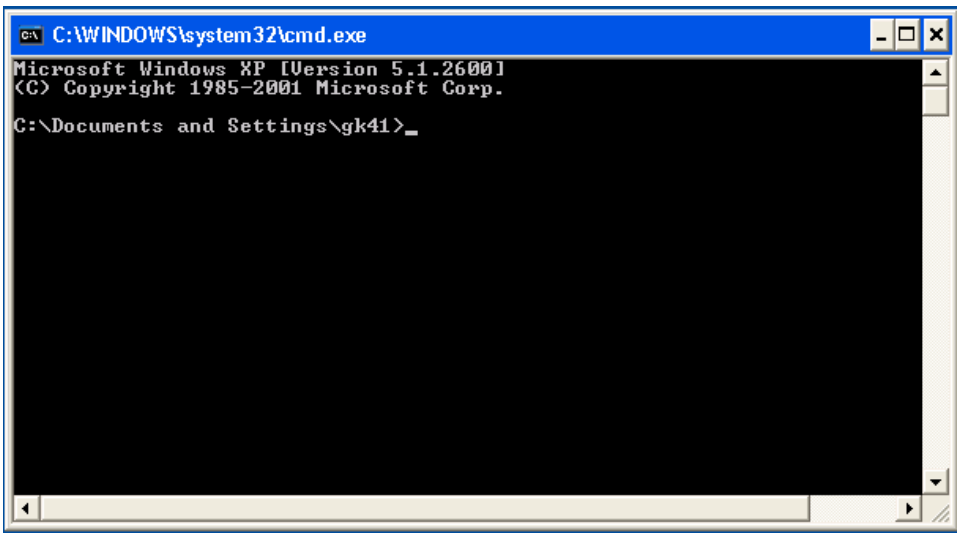


Figure 2-7 Open Command Line Window

2） Enter "ping 192.192.192.211 (the default IP address of SBC6845) as shown below to test network connection;

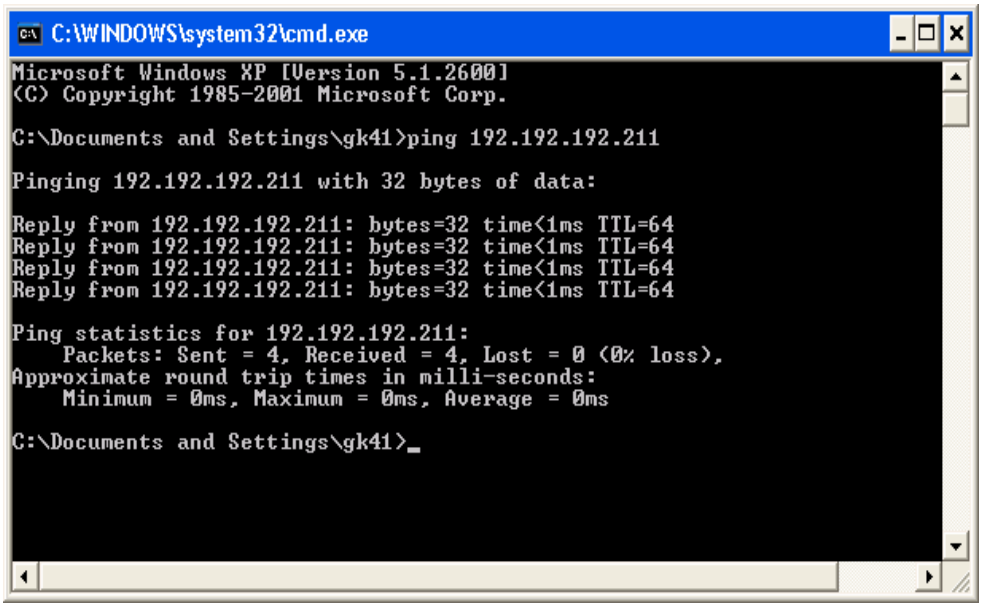


Figure 2-8 Testing Network Connection

3） Execute command **telnet** to initiate a session with SBC6845 as shown below;

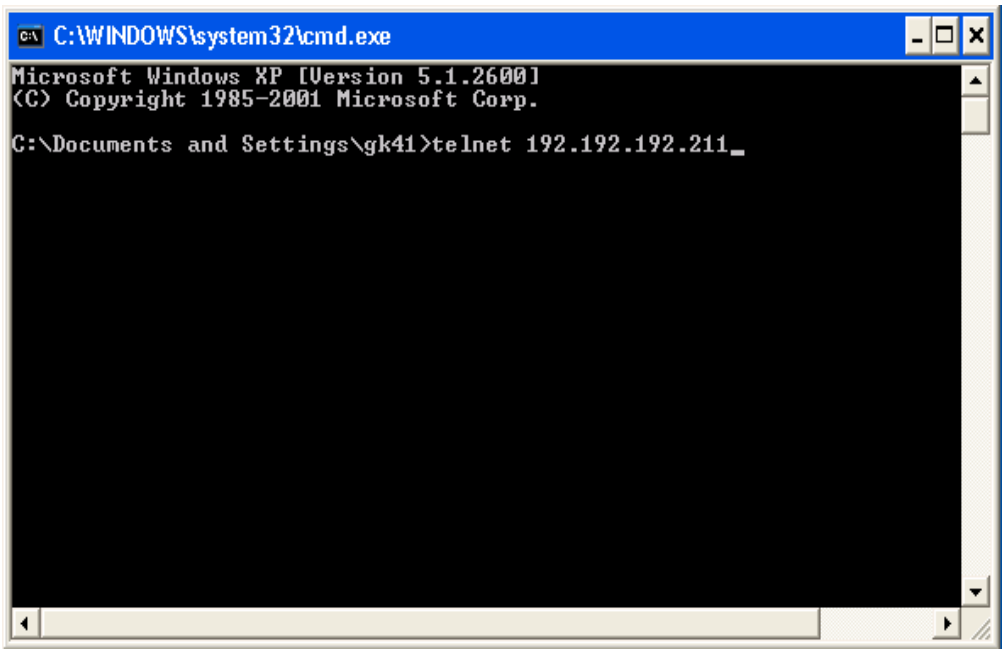


Figure 2-9 Initiating Telnet Session

4） Enter the default telnet login name **root** and leave the password blank, and then press Enter key on keyboard;

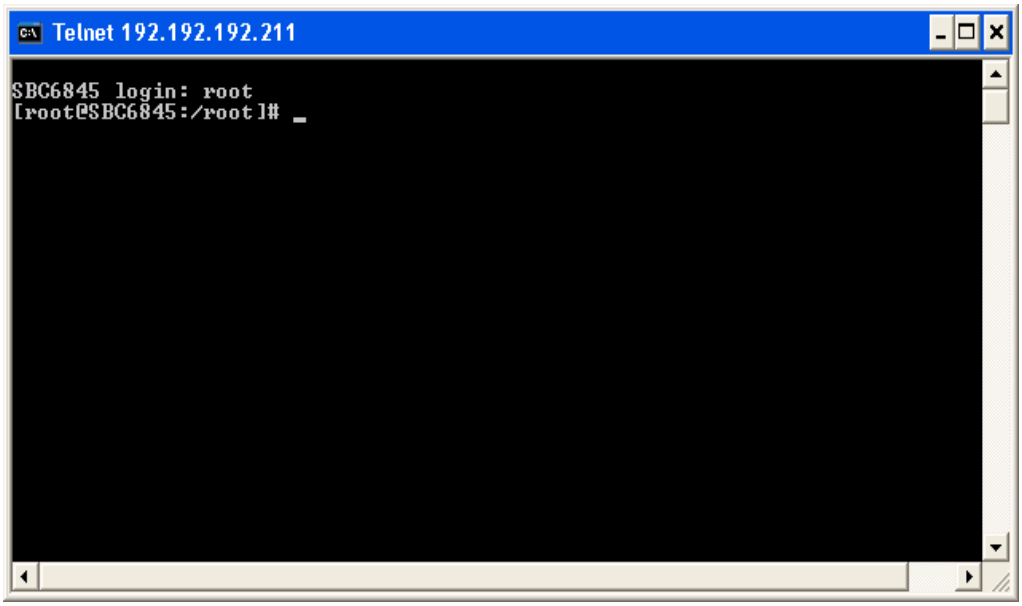


Figure 2-10 Logging in telnet

Now you have logged in telnet session.

5） Execute command **exit** to end telnet session;

Note:

- By default, telnet service is disabled under Window 7. Please select Control Panel > Programs > Programs and Features > Turn Windows features on or off, and then check “Telnet Client” to enable it.
- The default IP address of SBC6845 is 192.192.192.211. Please ensure that the board and your PC are set in the same network segment.

2.4.21 Mounting NFS Network Filesystem

By mounting NFS network filesystem, users can access share directory remotely under Linux environment. Please following the steps listed below to test NFS network filesystem.

1） Log in Linux system on your PC as **root** ;

2） Add the line shown below to the end of the file “exports” that is located under /etc/ and then save the change;

/home/nfs *(rw,sync,no_root_squash)

/home/nfs is the share directory of NFS server. All the clients could be mounted to this directory. **no_root_squash** allows users who access this directory work as root.

3） Execute the following instruction to start NFS server;

```
[root@SBC6845:/]# /etc/init.d/nfs-kernel-server start
```

4） Check if NFS server has been started successfully;

```
[root@SBC6845:/]# mount -o nolock localhost:/home/nfs /tmp
```

The server is working properly If the system doesn’t report any error, as well as the contents listed out by using the instruction “ls /tmp” is consistent with that under the share directory of the NFS server,

5） Power up SBC6845, connect a network cable to it, configure IP of the board, and use the command PING to ensure a proper communication between it

and Linux system on PC.

6） Execute the following instruction to mount share directory /home/nfs to /mnt;

[root@SBC6845:/]# mount -o nolock 192.192.192.105:/home/nfs /mnt

Now the contents under the share directory can be viewed by accessing /mnt.

Note:

📖 SBC6845 has the write privilege over the share directory, which means any change will be effective immediately.

2.4.22 Transferring Files by Using SecureCRT

The following contents will lead you through the testing of data transfer on serial interfaces by utilizing a Windows-based software SecureCRT.

1） Open **SecureSRT** window as shown below;

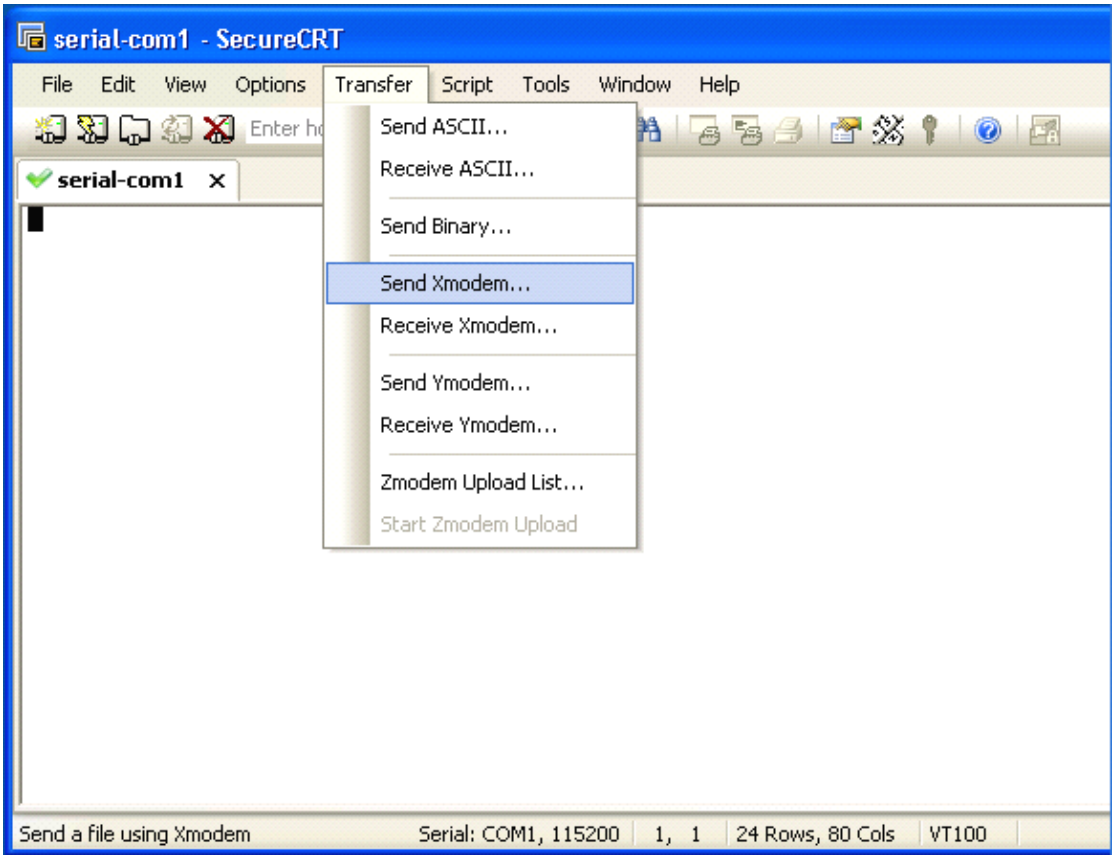


Figure 2-11 SecurCRT Window

Execute the following instructions in the window;

[root@SBC6845:/]# cd /tmp

[root@SBC6845:/tmp]# rx recvfile

2） Select **Transfer > Send Xmodem** on the menu bar to open the window as shown below;

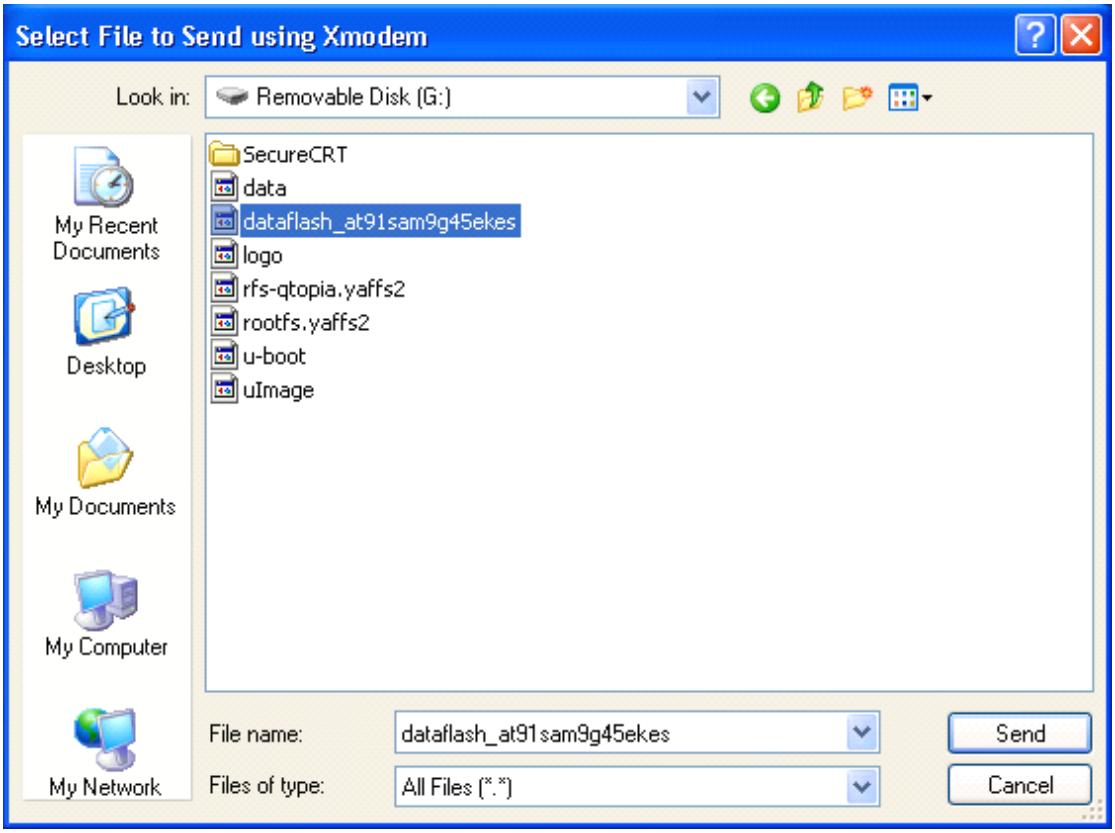


Figure 2-12 Select a File

Select a file and click **Send**; HyperTerminal window will display the following information;

Table 2-17 Sending File

```
Starting xmodem transfer. Press Ctrl+C to cancel.
Transferring    dataflash_at91sam9g45ekes.bin...
  100%          4 KB      0 KB/s 00:00:05          0 errors
[root@SBC6845:/tmp]#
```

The above information shows that the file has been transmitted successfully.

Please press **Ctrl+C** to exit.

Note:

🕮 It is recommended to select a small file due to the relatively slow speed of serial interfaces.

2.4.23 Transferring Files over Network

The following contents will lead you through the testing of big file transmission over network based on TFTP protocol.

1） Run tftpd.exe on your PC and put the file you about to send under HOME directory, e.g. G:\data.bin;

2） Execute the following instruction in HyperTerminal window to download data.bin;

```
[root@SBC6845:/tmp]# tftp -g 192.192.192.71 -r data.bin
```

3） Execute the following instruction to view the downloaded file;

```
[root@SBC6845:/tmp]# ls -l
```

The HyperTerminal will display the following information;

Table 2-18 Viewing Downloaded Files

-rw-r--r-- 1 root root 4420 Jan 1 00:44 data.bin

The above information indicates a successful downloading of the file.

4） Execute the following instruction to rename the downloaded file as data_send.bin.

```
[root@SBC6845:/tmp]# mv data.bin data_send.bin
```

5） Upload the file to HOME directory on PC;

```
[root@SBC6845:/tmp]# tftp -p 192.192.192.71 -l data_send.bin
```

6） Enter the share directory to view the uploaded file as shown below;

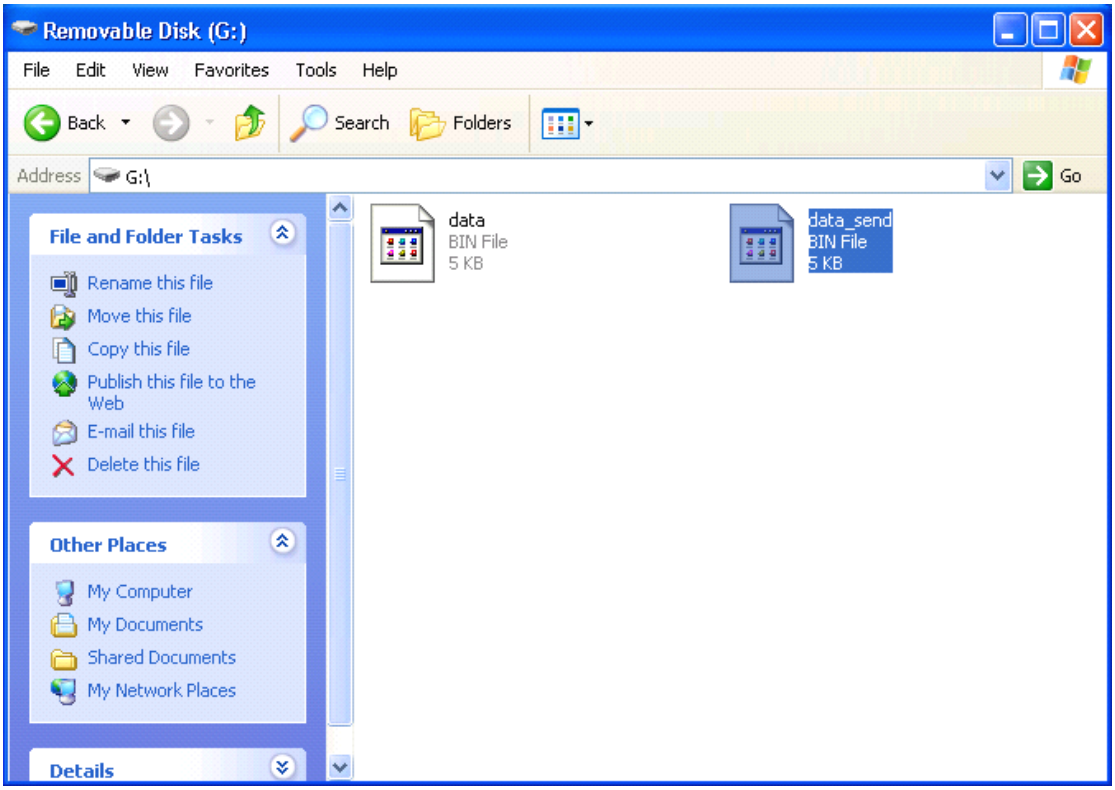


Figure 2-13 Viewing the Uploaded Files

The above figure indicates a successful uploading.

2.4.24 Linux QT Demonstration

When the system is under shell interactive mode, you can start Qtopia application by entering the command “qpe”. Please follow the steps listed below;

1） Firstly execute the following instruction to perform touch-screen calibration;

```
[root@SBC6845:/]# ts_calibrate
```

Please follow the instructions appear on the screen to calibrate it.

2） And then execute “qpe” to start Qtopia application; (the filesystem has to have a QT installed)

```
[root@SBC6845:/]# qpe
```

QT interface and system information are shown below;

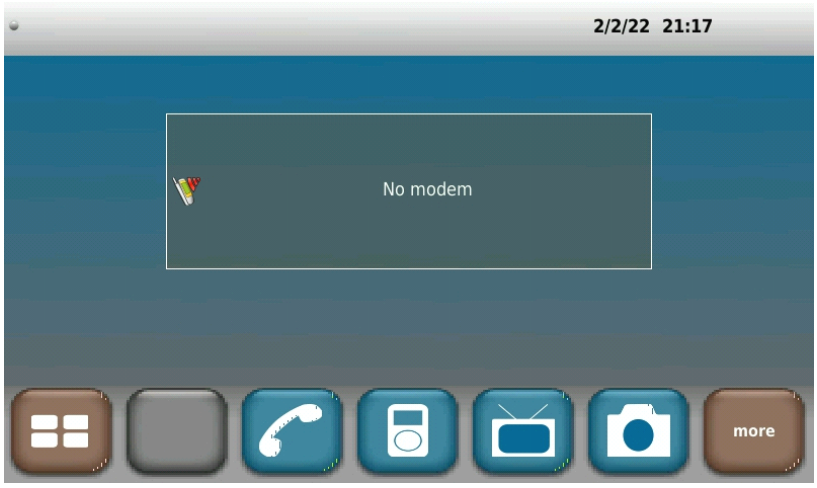


Figure 2-14 QT Interface

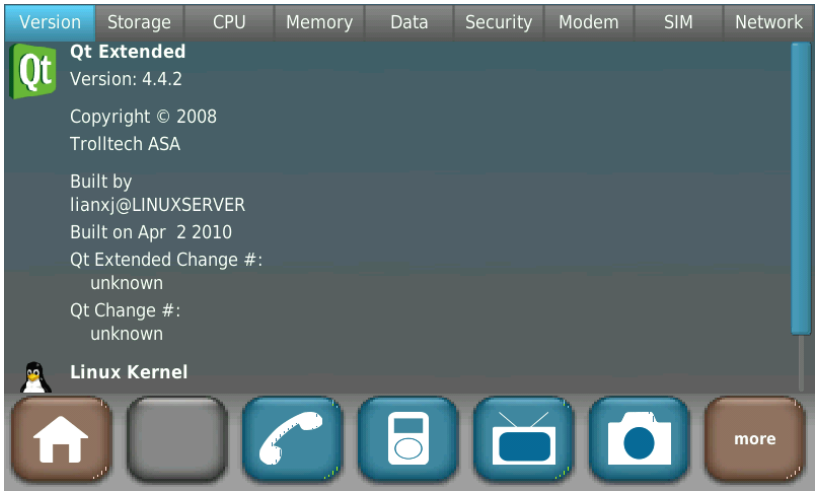


Figure 2-15 System Information

Chapter 3 Development Environment and System Compilation

Before getting started with the development on SBC6845, an ARM Linux cross development environment is required. This chapter will take Ubuntu as the example operating system to show you how to build a cross development environment and accomplish system compilation.

3.1 Building a Cross Compilation Environment

The CD-ROM provided along with the product contains a cross compilation tool arm-2007q1-10-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2 under \02 Linux2.6 Kit\02 Tools\. Please install it step by step as shown below.

1） Put the CD-ROM in your drive. Ubuntu will mount the CD to **/media/cdrom** by default. Please execute the following instructions to install the cross compilation tool;

mkdir /usr/local/arm

tar –jxvf arm-2007q1-10-arm-none-linux-gnueabi-i686-pc-linux-gnu.tar.bz2 –C /usr/local/arm

2） Execute the instruction below to add an environment variable which specifies the path to the cross compilation tool in the system.

export PATH=/usr/local/arm/arm-2007q1/bin/:$PATH

And then check if the installation successes by executing the following instruction;

arm-none-linux-gnueabi-gcc –v

The information in HyperTerminal is shown below;

Table 3-1 Checking Installation

```
Using built-in specs.
Target: arm-none-linux-gnueabi
...
gcc version 4.2.0 20070413 (prerelease) (CodeSourcery Sourcery G++ Lite 2007q1-10)
```

If the version number within the last line is correct, the cross compilation environment has been built successfully.

Note:

📖 The instruction adding environment variables can be put into the file .bashrc under user directory to allow the system load the variable automatically each time when it boots up.

3.2 System Compilation

The compilation of operating system can be accomplished in 5 steps – uncompressing files, making U-boot, making kernel and making filesystem image. This section will introduce these steps in detail.

1.1.1 Uncompressing Files

The system source code can be found under \02 Linux 2.6 Kit\01 SourceCode\. Please execute the following instructions to uncompress it under Linux system.

Table 1-1 Uncompressing files

```
root@LINUXSERVER:~# mkdir embest
root@LINUXSERVER:~# cd embest/
root@LINUXSERVER:~/embest#
cp /media/cdrom/02\ Linux\ 2.6\ Kit/01\ SourceCode/bootloader/Bootstrap-v1.14.tar.bz2 ./
root@LINUXSERVER:~/embest#
cp /media/cdrom/02\ Linux\ 2.6\ Kit/01\ SourceCode/bootloader/u-boot-1.3.4.tar.bz2 ./
root@LINUXSERVER:~/embest#
cp /media/02\ Linux\ 2.6\ Kit/01\ SourceCode/kernel /linux-2.6.30.tar.bz2 ./
root@LINUXSERVER:~/embest#
    cp /media/cdrom/02\ Linux\ 2.6\ Kit/01\ SourceCode/rfs/rfs-sbc6845.tar.bz2 ./
root@LINUXSERVER:~/embest#
```

```
    cp /media/cdrom/02\ Linux\ 2.6\ Kit/02\ Tools/mkyaffs2image /usr/local/bin/
root@LINUXSERVER:~/embest#
    cp /media/cdrom/02\ Linux\ 2.6\ Kit/02\ Tools/mkimage /usr/local/bin/
root@LINUXSERVER:~/embest#
chmod 755 /usr/local/bin/mkyaffs2image /usr/local/bin/mkimage
root@LINUXSERVER:~/embest# tar jxvf Bootstrap-v1.14.tar.bz2
root@LINUXSERVER:~/embest# tar jxvf u-boot-1.3.4.tar.bz2
root@LINUXSERVER:~/embest# tar jxvf linux-2.6.30.tar.bz2
root@LINUXSERVER:~/embest# mkdir rfs-qtopia; tar jxvf rfs-sbc6845.tar.bz2 -C rfs-qtopia
```

When all the instructions are executed, four directories - linux-2.6.30, u-boot-1.3.4, Bootstrap-v1.14 and rfs-qtopia are generated under current directory.

1.1.2 Making Bootstrap

SBC6845 support boot-up from DATAFLASH. Please execute the following instructions;

```
root@LINUXSERVER:~/embest# cd Bootstrap-v1.14
root@LINUXSERVER:~/embest/Bootstrap-v1.14# cd board/at91sam9g45ekes/dataflash
root@LINUXSERVER:~/embest/Bootstrap-v1.14/board/at91sam9g45ekes/dataflash # make
```

A bootstrap file dataflash_at91sam9g45ekes.bin has been generated under current directory.

1.1.3 Making U-boot

Please execute the following instructions;

```
root@LINUXSERVER:~/embest/u-boot-1.3.4# make at91sam9g45ekes_dataflash_config
root@LINUXSERVER:~/embest/u-boot-1.3.4# make
```

A file U-boot.bin has been generated under current directory.

1.1.4 Making Kernel

Please execute the following instructions;

```
root@LINUXSERVER:~/embest/linux-2.6.30# make sbc6845_defconfig
root@LINUXSERVER:~/embest/linux-2.6.30# make menuconfig
```

```
root@LINUXSERVER:~/embest/linux-2.6.30# make uImage
```

A kernel file named uImage has been generated under /arch/arm/boot/.

Note:

📖 If errors occur when executing **make menuconfig**, the most likely cause is the lack of **ncurse** library in your PC's Linux system. Downloading and then installing the library could solve the issue.

1.1.5 Making Filesystem Image

Use the tool mkyaffs2image under the directory \02 Linux 2.6 Kit\02 Tools\ of the CD-ROM to make a filesystem image by executing the following instruction (suitable for Ubuntu system only).

```
root@LINUXSERVER:~/embest# mkyaffs2image rfs-qtopia/ rootfs.yaffs2
```

Chapter 2 System Customization

In order to satisfy different requirements of customers, designers commonly need to make some customized modification based on the configuration of default Linux kernel. This chapter will introduce the process of system customization by showing a number of examples.

2.1 Kernel Customization

Please execute the following instructions to enter the configuration menu and select the drivers you need according to the entries shown in Table 4-1.

```
root@LINUXSERVER:~/embest/linux-2.6.30# make sbc6845_defconfig
root@LINUXSERVER:~/embest/linux-2.6.30# make menuconfig
```

Table 2-1 Kernel Customization

Serial Interface Driver	**Device drivers > Character devices > Serial drivers > AT91 / AT32 on-chip serial port support**
Button Driver	**Device drivers > Input device support > Keyboards > GPIO Buttons**
Matrix Keypad Driver	**Device drivers > Input device support > Keyboards > GPIO Keypad**
GPIO Driver	**Device drivers > GPIO Support > /sys/class/gpio/... (sysfs interface)**
LED Driver	**Device drivers > LED Support > LED Class Support > LED Support for GPIO connected LEDs**
SD/MMC Driver	**Device drivers > MMC/SD/SDIO card support > MMC block device driver > Atmel SD/MMC Driver (Atmel Multimedia Card Interface support)**
USB Driver	**Device drivers > USB support > Support for Host-side USB > EHCI HCD (USB 2.0) support > OHCI HCD support > USB Mass Storage supportHCD support > USB Mass Storage support**
RTC Driver	**Device drivers > Real Time Clock > EPSON RX-8025SA**

Watchdog Driver	**Device drivers > Watchdog Timer Support > AT91SAM9 watchdog**
CAN Driver	**Device drivers > CAN support > CAN support > MCP2515 CAN Controller**
MACB Driver	**Device drivers > Network device support > Ethernet(10 or 100Mbit) > Atmel MACB support**
Graphics Driver	**Device drivers > Graphics support > Support for frame buffer devices > AT91/AT32 LCD Controller support**
Touch-Screen Driver	**Input device support > Touchscreens > Atmel Touchscreen Interface**

Save the changes and execute the instruction below to compile the customized kernel;

root@LINUXSERVER:~/embest/linux-2.6.24# **make uImage**

2.2 Filesystem Customization

Table 2-2 Filesystem Customization

Configuration Information	Path	Note
Driver Modules	/lib/modules/2.6.30/	Driver modules are save here (ko)
Driver Module Mounting	/etc/init.d/S50modules	
Network Address	/etc/network/interfaces.eth0	
Command Line Prompt Name	/etc/hostname	
User Program Auto Running	/etc/init.d/S60evnset	Add it to the end of file
Environment Variables	/etc/profile	
Touch-Screen Coordinate Files	/etc/pointercal	
udev Rules	/etc/udev	
LCD Backlight Brightness	/etc/bl_adjust.conf	
User Testing Applications	/home/app	

2.3 Simple Driver Modules in Kernel

Drivers are running under kernel mode and can drive hardware directly. They provide a series of interfaces to be called by applications so as to control devices. The table shown below is an example of driver modules that are simple but including most of interfaces.

Table 2-3 Drivers

```
/* File: device_drv.c */
#include <linux/kernel.h>
#include <linux/module.h>
#include <linux/init.h>
#include <linux/input.h>
#include <linux/miscdevice.h>
#include <asm/io.h>
#include <asm/uaccess.h>                    /* common head files used by driver */

#define DEVICE_NAME "demo"                  /* device names that generate nodes /dev/demo */ after
mounting successfully

static int result = 0;

static int device_open(struct inode *inode, struct file *file)          /* implement open operation */
{
    result = 0;                             /* initiate result */
    return 0;
}
static ssize_t device_read(struct file *filp, char *buffer, size_t count, loff_t *ppos)    /*    implement    read
operation */
{
    int ret = copy_to_user (buffer, (char *)&result, sizeof(result));   /* copy the value of result to buffer */
    if (ret < 0) {
        printk (KERN_ERR "%s: copy_to_user error\n", DEVICE_NAME);
        return -1;
    }
    return sizeof(result);                  /* return the valid length of buffer, i.e. the storage
length of result */
}
static ssize_t device_write(struct file *filp, const char *buffer, size_t count, loff_t *ppos) /* write operation */
{
    int ret = copy_from_user ((char *)&result, buffer, sizeof(result));
/* copy the received data in buffer to result */
    if (ret < 0) {
        printk (KERN_ERR "%s: copy_from_user error\n", DEVICE_NAME);
        return -1;
    }
    return sizeof(result);
}
static int device_release(struct inode *inode, struct file *filp)       /* close will trigger the function */
{
    return 0;
```

```
}
static struct file_operations device_fops =          /* register interface function for file operation */
{
    .owner      = THIS_MODULE,
    .open       = device_open,
    .read       = device_read,
    .write      = device_write,
    .release    = device_release,
};
static struct miscdevice device_miscdev =                    /* register misc device information */
{
    .minor     = MISC_DYNAMIC_MINOR,
    .name      = DEVICE_NAME,
    .fops      = &device_fops,
};
static int __init device_init(void)                  /* insmod operation will trigger the function */
{
    int ret;

    ret = misc_register(&device_miscdev);                /* register device */
    if (ret) {
        printk(KERN_ERR "cannot register miscdev on minor=%d (%d)\n", MISC_DYNAMIC_MINOR, ret);
        goto out;
    }

    printk(KERN_INFO DEVICE_NAME " initialized!\n");
    return 0;

out:
    return ret;
}

static void __exit device_exit(void)                 /* rmmod operation will trigger the function */
{
        misc_deregister(&device_miscdev);
        printk(KERN_INFO DEVICE_NAME " removed!\n");
}

module_init(device_init);
module_exit(device_exit);

MODULE_LICENSE("GPL");                                       /* protocol used by driver modules */
MODULE_DESCRIPTION("Linux Driver Demo");             /* driver module description */
```

1.1.1 Using Makefile to Associate Drivers with Kernel

Driver files have to be associated with kernel by a Makefile before they can be compiled and loaded. The following table shows the contents of Makefile.

Table 1-1 Contents of Makefile

```
# File: Makefile
ifneq ($(KERNELRELEASE),)
    obj-m := device_drv.o          # driver file with extension name .o other than .c; by default .c files will be searched and compiled automatilly
else
        KERNELDIR ?= ~/embest/linux-2.6.30                    # specify the path of kernel source code

        PWD := $(shell pwd)
all:
        $(MAKE) -C $(KERNELDIR) M=$(PWD) modules
clean:
        rm -rf *.o *~ core .depend .*.cmd *.ko *.mod.c .tmp_versions Module.symvers modules.order device_drv.ko
endif
```

1.1.2 Compiling and Downloading Drivers

Before you start to compile drivers with the command “make”, kernel source code should be compiled first. After compiling successfully, you can download the generated file device_drv.ko to the board. The following table contains the required instructions and corresponding information.

Table 1-2 Compiling and Downloading Drivers

```
[root@SBC6845:/]# insmod device_drv.ko
demo initialized!
[root@SBC6845:/]# ls /dev/demo
/dev/demo
[root@SBC6845:/]# rmmod device_drv.ko
demo removed!
```

1.2 Brief Introduction to Applications

The previous example shows the executing process of drivers. You might notice that there are only two functions – device_init and device_exit have been called, while others remain unused in the above process. The interfaces in structure device_fops intended for application layer. The table shown below will give you an example of the basic structure of a Linux application.

Table 1-3 Example Application

```
/* File: demo.c */
#include <stdio.h>
#include <fcntl.h>
#include <string.h>                                    /* head file being called */

#define dev    "/dev/demo"                          /* demo file node */

int main (void)
{
      int fd;
      int err = 0;
      int value;

      fd = open (dev, O_RDWR);                     /* open file node, readable and writable */
      if (fd < 0) {
            fprintf (stderr, "open fail\n");
            err = 1;
            goto out;
      }

      if (read (fd, &value, sizeof(value)) < 0) {              /* read function that calls driver; the read value t is
 save in value */
            fprintf (stderr, "read error\n");
            err = 1;
            goto out;
      }
      printf ("read before write, value=%X\n", value);    /*     print read value before writing */

      int writeValue = 0x5E7F;
      if (write (fd, &writeValue, sizeof(writeValue)) < 0) {       /* writing 0x5E7F to driver module by calling
write function */
```

```
            fprintf (stderr, "write error\n");
            err = 1;
            goto out;
        }
        if (read (fd, &value, sizeof(value)) < 0) {                    /* read again after writing */
            fprintf (stderr, "read error\n");
            err = 1;
            goto out;
        }
        printf ("read after write, value=0x%X\n", value);          /* print read value after writing */

out:
        if (fd > 0) close (fd);
        return err;
}
```

1.1.1 Compiling and Running Applications

Execute the instruction below to compile application.

```
# arm-none-linux-gnueabi-gcc demo.c -o demo
```

The generated executable file named demo is the application we need to download to the board and run it.

Table 1-1 Compiling and Running Applications

```
[root@SBC6845:/]# insmod device_drv.ko
demo initialized!
[root@SBC6845:/]# ./demo
read before write, value=0
read after write, value=0x5E7F
```

1.1.2 Common Functions

The following three functions are commonly used by driver layer to control GPIO.

```
int at91_set_gpio_input(unsigned pin, int use_pullup)                    /* set GPIO as input */

int at91_get_gpio_value(unsigned pin)                                    /* acquire GPIO input
value */
```

```
int at91_set_gpio_output(unsigned pin, int value)                    /* set GPIO as output */
```

Adding the above GPIO code to the appropriate location in drivers can easily implement LED control.

```
at91_set_gpio_input (AT91_PIN_PC16, 0);                    /* set PC16 as input, pull-up
disabled*/
at91_get_gpio_value (AT91_PIN_PC16);                    /* read the input value on
PC16 */
at91_set_gpio_output(AT91_PIN_PC16, 1);                    /* set PC16 to provide
high-level output */
```

1.2 Linux Multi-Thread Programming

The threads here refer to the multiple tasks created in user space. These tasks share resources of the same process – low-weight process. It consumes much less cost than common process and features fast context switch.

Since the resources are shared by processes, it is necessary to adopt synchronizing measures in order to avoid competition when accessing resources.

Table 1-2 Multi-thread Programming Example

```
/* File: pthread.c */
#include <stdio.h>
#include <unistd.h>
#include <pthread.h>

void read_func(void);
void write_func(void);



int buffer_has_item = 0;                    /* share resources */

pthread_mutex_t mutex;                    /* mutex lock */

int main(void)
{
    pthread_t reader, writer;                    /* define process ID */
    pthread_mutex_init(&mutex, NULL);                    /* initiate mutex lock */
```

```
    pthread_create(&reader, NULL, (void*)&read_func, NULL);   /* create process */
    pthread_create(&writer, NULL, (void*)&write_func, NULL);

    pthread_join(reader, NULL);                               /* wait for end of process */
    pthread_join(writer, NULL);

    return 0;
}

void write_func(void)
{
    while (1) {
        pthread_mutex_lock(&mutex);                           /* enable lock, other processes will
be locked/
        if (buffer_has_item == 0) {
            printf("create a new item\n");
            buffer_has_item = 1;
        }
        pthread_mutex_unlock(&mutex);                         /* disable lock, other process will be
unlocked */
    }
}

void read_func(void)
{
    while (1) {
        pthread_mutex_lock(&mutex);
        if (buffer_has_item == 1) {
            printf ("destroy item\n");
            buffer_has_item = 0;
        }
        pthread_mutex_unlock(&mutex);
    }
}
```

Please execute the instruction below to implement compilation.

 # arm-none-linux-gnueabi-gcc pthread.c -o pthread_demo -lpthread

1.3 Linux Network Programming

Linux network programming generally can be implemented based on UDP and TCP protocols. UDP is a connectionless transport protocol that provides simple, unreliable and message-oriented services; TCP is a reliable, connection-oriented and byte-stream-based transport protocol. The following examples are a simple TCP server and a client.

- **server**: monitors the connection initiated by client and sends character string to client when a connection is created.
- **client**: Initiates a connection to server, receives and prints information sent from server.

Table 1-3 TCP Server Example

```
/* File: server.c */
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <sys/types.h>
#include <netinet/in.h>
#include <sys/socket.h>
#include <sys/wait.h>

#define MYPORT 3490 /* the port users will be connecting to */
#define BACKLOG 10    /* how many pending connections queue will hold */

main()
{
      int sockfd, new_fd;              /* listen on sock_fd, new connection on new_fd */
      struct sockaddr_in my_addr; /* local address information */
      struct sockaddr_in their_addr; /* connector's address information */
      int sin_size;

      if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
            perror (" socket ") ;
            exit(1) ;
      }
```

```
    my_addr.sin_family = AF_INET;
    my_addr.sin_port = htons(MYPORT);
    my_addr. sin_addr.s_addr = INADDR_ANY; /* auto-fill with local IP */

    if (bind(sockfd, (struct sockaddr *)&my_addr, sizeof(struct sockaddr)) == -1) {
        perror (" bind ") ;
        exit(1) ;
    }

    if (listen(sockfd, BACKLOG) == -1) {
        perror (" listen ") ;
        exit(1) ;
    }

    while(1) {              /* main accept() loop */
        sin_size = sizeof(struct sockaddr_in);
        if ((new_fd = accept(sockfd, (struct sockaddr *)&their_addr,
        &sin_size)) == -1) {
            perror (" accept ") ;
            continue ;
        }
        printf("server: got connection from %s\n", inet_ntoa(their_addr.sin_addr));
        if (!fork()) { /* this is the child process */
        if (send(new_fd, "Hello, world!\n", 14, 0) == -1)
            perror( " send " ) ;
            close( new_fd ) ;
            exit ( 0 ) ;
        }
        close(new_fd);
        while(waitpid(-1,NULL,WNOHANG) > 0); /* clean up child processes */
    }
}
```

Table 1-4 TCP Client Example

```
/* File: client.c */
#include <stdio.h>
#include <stdlib.h>
#include <errno.h>
#include <string.h>
#include <netdb.h>
#include <sys/types.h>
#include <netinet/in.h>
```

```
#include <sys/socket.h>

#define PORT 3490            /* the port client will be connecting to */
#define MAXDATASIZE 100 /* max number of bytes we can get at once */

int main(int argc, char *argv[])
{
    int sockfd, numbytes;
    char buf[MAXDATASIZE] ;
    struct hostent *he;
    struct sockaddr_in their_addr; /* connector's address information */

    if (argc != 2) {
        fprintf(stderr,"usage: client hostname\n");
        exit (1) ;
    }

    if ((he=gethostbyname(argv[1])) == NULL) { /* get the host info */
        herror(" gethostbyname ") ;
        exit (1);
    }

    if ((sockfd = socket(AF_INET, SOCK_STREAM, 0)) == -1) {
        perror( " socket ");
        exit (1);
    }

    their_addr.sin_family = AF_INET;
    their_addr.sin_port = htons(PORT);
    their_addr.sin_addr = *((struct in_addr *)he->h_addr); //inet_addr

    if (connect(sockfd, (struct sockaddr *)&their_addr, sizeof(struct sockaddr)) == -1)
{
        perror(" connect ");
        exit (1);
    }

    if ((numbytes=recv(sockfd, buf, MAXDATASIZE, 0)) == -1) {
        perror (" recv ");
        exit (1);
    }
    buf[numbytes] = '\0';
    printf("Received: %s",buf);
```

```
    close(sockfd) ;
    return 0;
}
```

1.1.1 Compiling Server and Client

If you have two SBC6845, you only need to execute two instructions below to implement compilation;

 # arm-none-linux-gnueabi-gcc client.c -o client

 # arm-none-linux-gnueabi-gcc server.c -o server

And then run the server and client on two SBC6845 respectively to conduct testing.

If you only have one SBC6845, firstly you need to compile a server program working on PC's Linux system by executing the instruction below;

 # gcc server.c -o server

Secondly execute the following instruction under Linux system on PC to run server;

 # ./server

Thirdly download client to SBC6845 and run it as shown in the table below;

Table 1-1 Run Client

```
[root@SBC6845:/]# chmod 755 client
[root@SBC6845:/]# ./client 192.192.192.105          # server IP
Received: Hello, world!
[root@SBC6845:/]# ./client 192.192.192.105
Received: Hello, world!
[root@SBC6845:/]# ./client 192.192.192.105
Received: Hello, world!
```

The information on server is shown below;

Table 1-2 Information on Server

```
server: got connection from 192.192.192.211
server: got connection from 192.192.192.211
server: got connection from 192.192.192.211
```

Chapter 2 Updating Linux System

SBC6845 has a DATAFLASH and a NANDFLASH on board. But Linux system can only support boot-up from DATAFLASH currently. This chapter will introduce in detail how to update Linux system in DATAFLASH.

2.1 Image Mapping and Burning

The following two figures illustrate how the images are distributed in DATAFLASH and NANDFLASH.

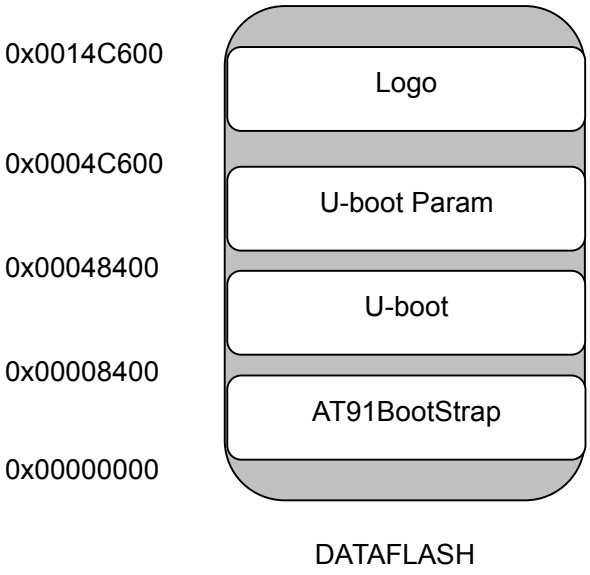


Figure 2-1 DATAFLASH

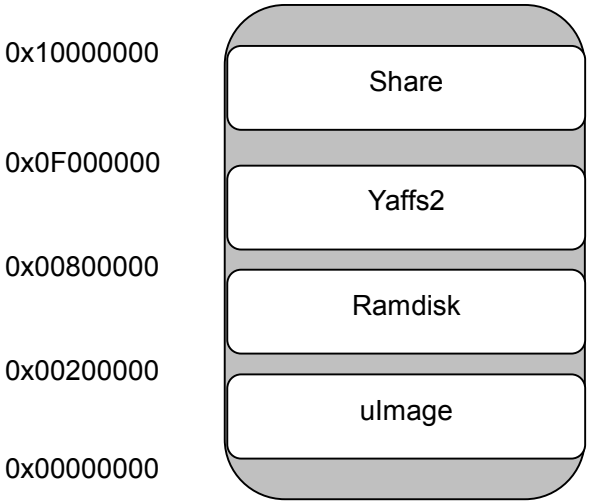


Figure 2-2 NANDFLASH

Please install a software tool first named **SAM-BA** on your PC by executing **Install AT91-ISP v1.13.exe** (or higher version) saved under **\02 Linux2.6 Kit\02 Tools** in the CD-ROM. SAM-BA is a Windows-based tool that can write flash memories on single board computer.

Figure 2-3 SAM-BA V2.9 Shortcut on Desktop

2.2 Burning System Image Manually

Please follow the instructions below to burn the system image on the board step by step.

1.1.1 Preparations

1） Connect the serial and USB interfaces on SBC6845 to your PC respectively; Please refer to the relevant contents of Hardware Manual to view the detailed information on connections;

2） Open a HyperTerminal on your PC, and configure it as 115200 bitrate, 8 bit

databit, None Parity, 1 stop bit, and None data flow control.

3） Connect power supply to SBC6845 and launch SAM-BA v2.9 to open the window as shown below;

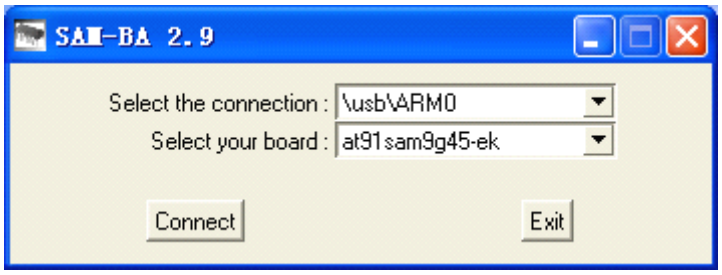


Figure 1-1 SAM-BA Initial Window

If the USB connection between the board and your PC is working properly, an option **\usb\ARM0** can be seen in **Select the connection** drop-down menu. Select **AT91sam9g45-ek** in **Select your board** drop-down menu and then click **Connect**;

Note:

Please ensure there is no any boot code in DATAFLASH or DATAFLASH is disabled (by disconnecting JP14 on the board) before you power up the board, so that SAM-BA can work properly.

4） Click the **Dataflash** tab in SAM-BA main window as shown below, and select **Enable Dataflash (SPI0 CS0)** in **Scripts** drop-down menu (if the JP14 on the board is disconnected, please short it to enable DATAFLASH), and then click **Execute** on the right to start enabling process. The information box at the bottom of the window will display the details of the process as the figure shown below;

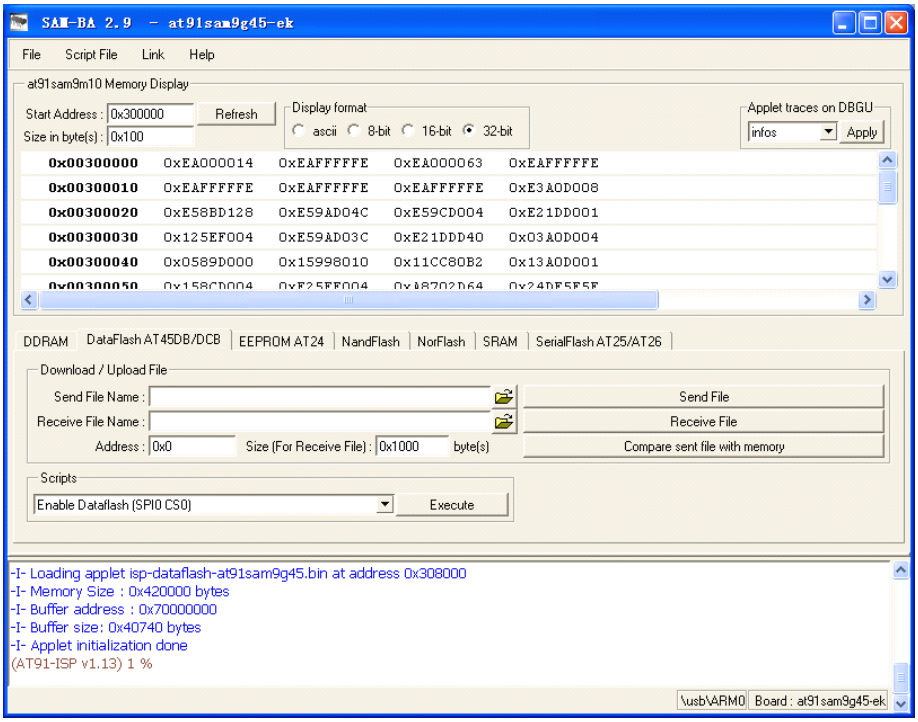


Figure 1-2 Enable DATAFLASH

5） Select **Erase All** in **Scripts** drop-down menu and then click **Execute** to erase all the contents in DATAFLASH as shown in the figure below;

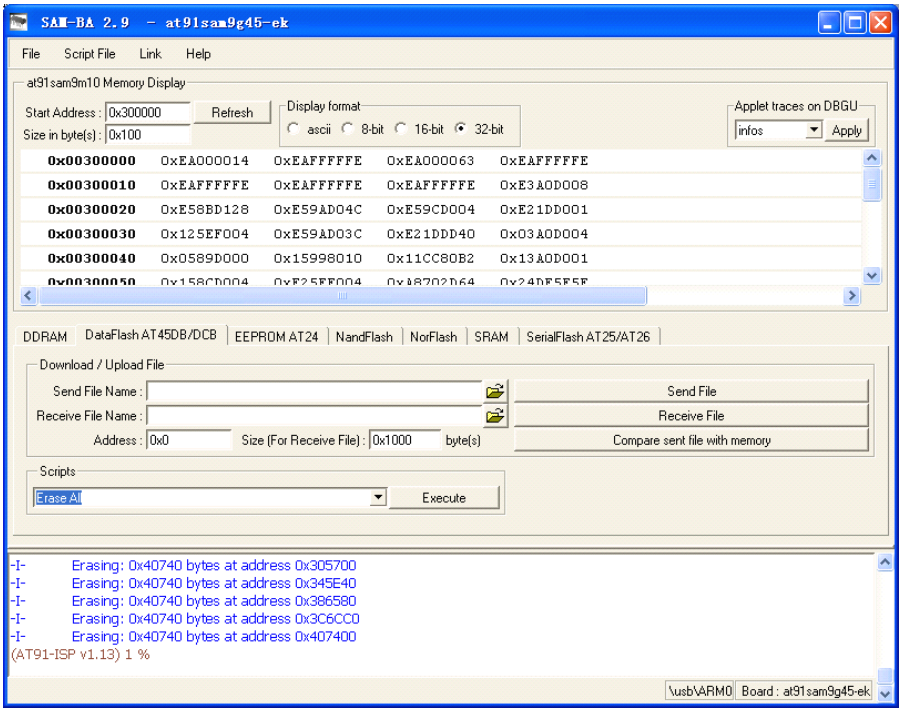


Figure 1-3 Erase DATAFLASH

1.1.2 Burning Bootstrap File

1） Select **Send Boot File** in **Scripts** drop-down menu of SAM-BA's main window and then click **Execute** to open the following window;

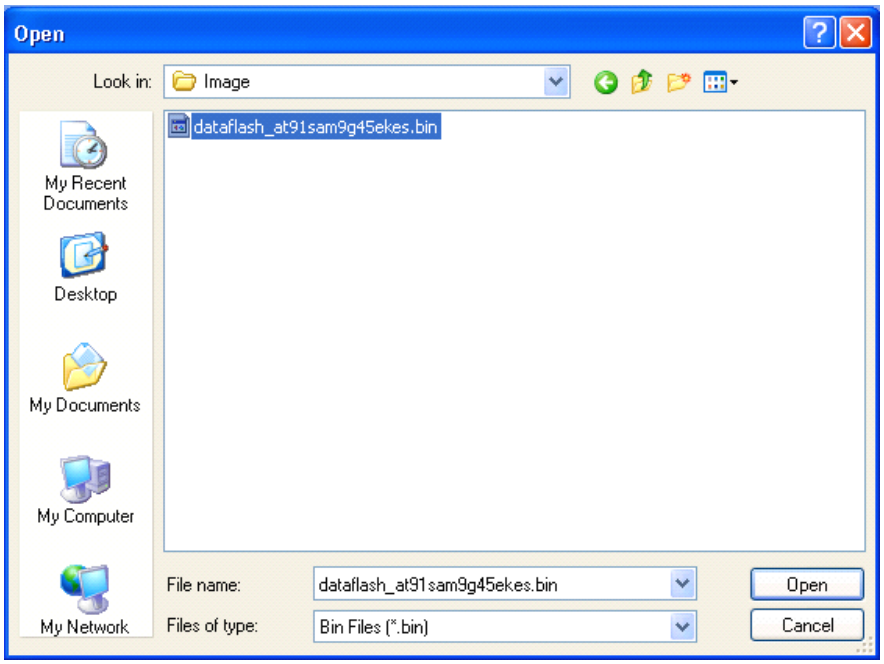


Figure 1-4 Select the File to be Burned

Select **dataflash_at91sam9g45ekes.bin** in the above window and click **Open**;

2） The information box at the bottom of SAM-BA's main interface will show the downloading process as you can see below;

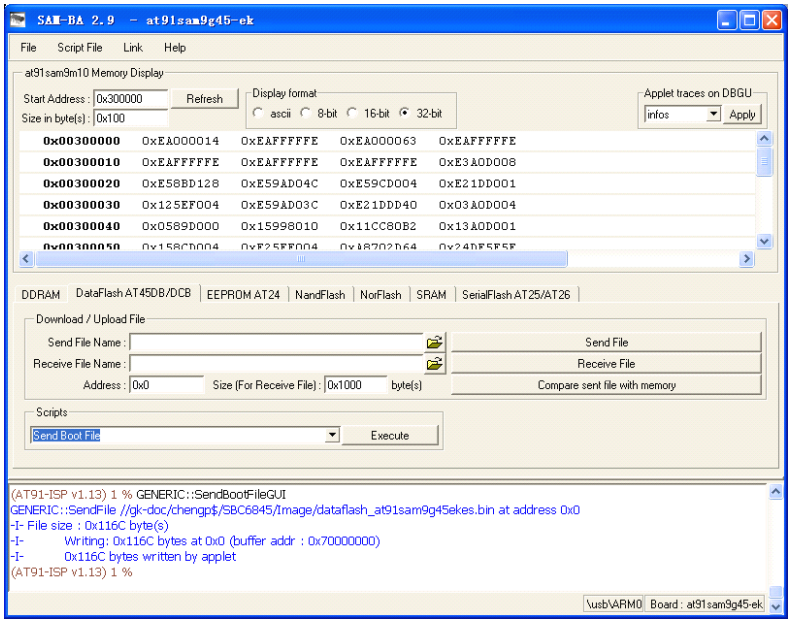


Figure 1-5 Burn dataflash_at91sam9g45ekes.bin

1.1.3 Burning U-boot File

1） Enter an address **0x8400** in **Address** text box of SAM-BA's main window and click  on the right of **Send File Name** text box to open the following window;

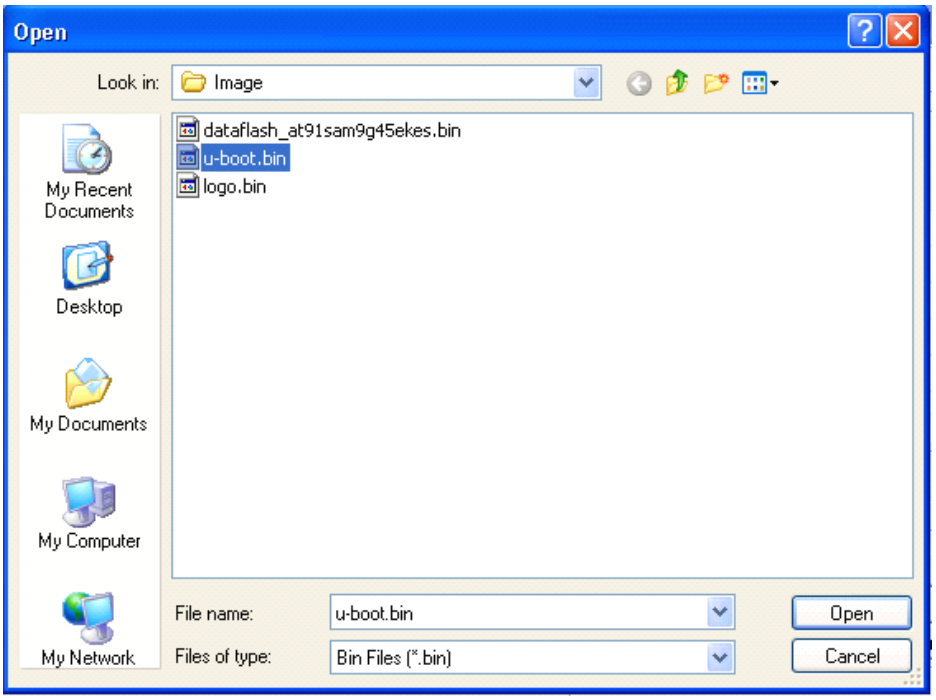


Figure 1-6 Select u-boot.bin

Select **u-boot.bin** and click **Open**;

2） Click **Send File** in SAM-BA main window to download u-boot.bin to the DATAFLASH on the board;

1.1.4 Burning Logo File

1） Enter an address **0x4C600** in the **Address** text box of SAM-BA main window and click  on the right of **Send File Name** text box to open the following window;

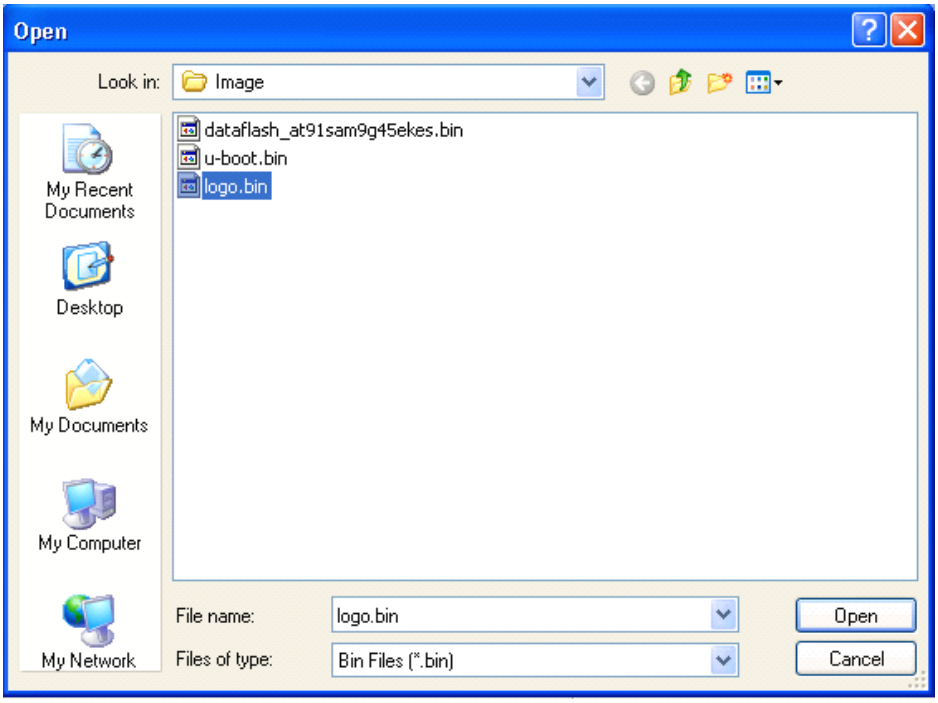


Figure 1-7 Select logo.bin

Select **logo.bin** and click **Open**;

2） Click **Send File** in SAM-BA main window to download **logo.bin** to the DATAFLASH on the board;

1.1.5 Burning uImage File

1） Click **NandFlash** tab and select **Enable NandFlash** in **Script** drop-down menu, and then click **Execute** to enable NANDFLASH as shown below;

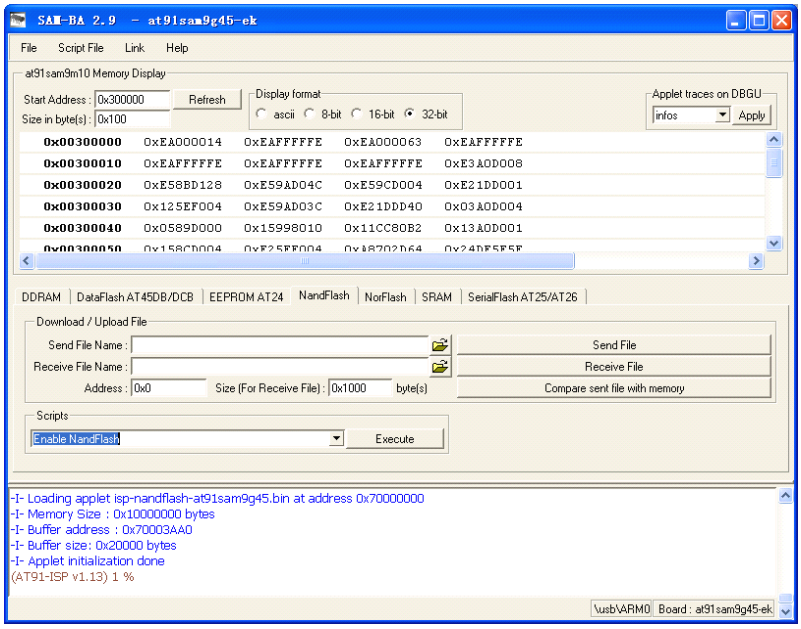


Figure 1-8 Enable NANDFLASH

Note:

Please ensure the JP8 on the board is shorted.

2） Enter an address **0x0** in **Address** text box and click on the right of **Send File Name** text box to open the following window;

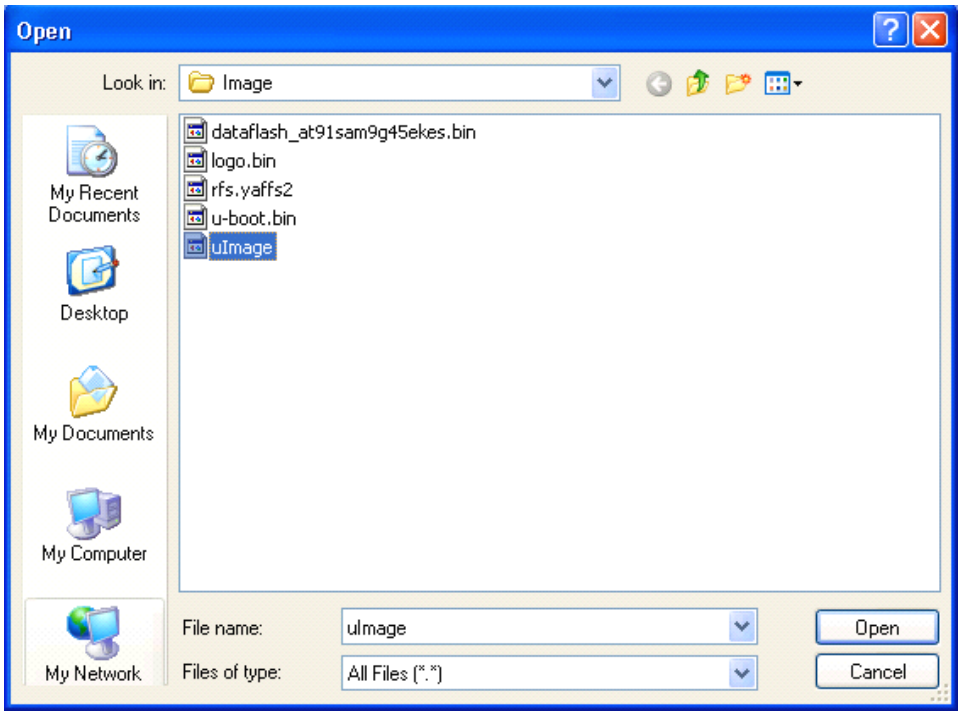


Figure 1-9 Select uImage

Select uImage and click **Open**;

3） Click **Send File** in SAM-BA main window to download **uImage** to the NANDFLASH on the board;

1.1.6 Burning Linux System

1） Connect SBC6845 to your PC with a network cable;

2） Copy **tftpd.exe** located under **\02 Linux2.6 Kit\02 Tools** of the CD-ROM to your PC as shown below;

Figure 1-10 Tftpd Software

3） Double-click tftpd.exe to open TFTP server as shown below;

Figure 1-11 Tftpd Software Window

4） Select **Tftpd > Configure** in the menu bar as shown below;

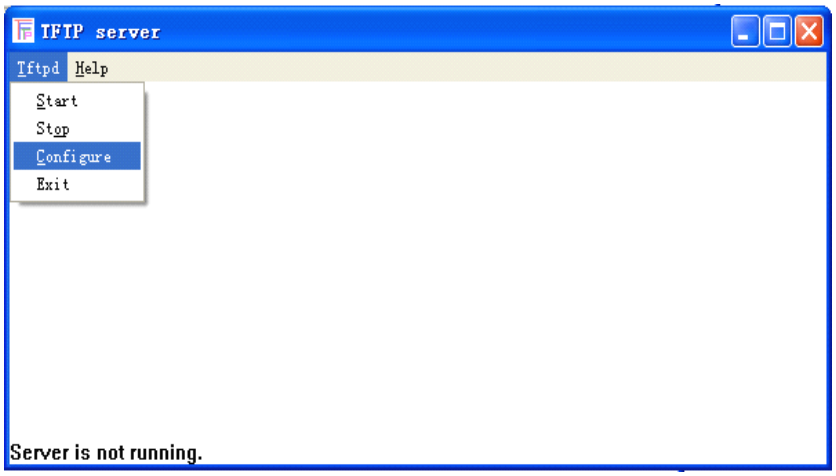


Figure 1-12 Select Configure

5） Click **Browse** in the pop-up window shown below to specify the path of uImage.bin and click **OK**;

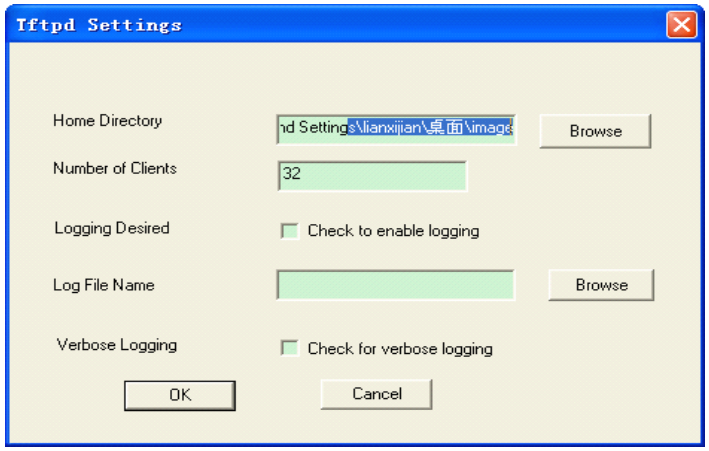


Figure 1-13 Configuration Window

6） Select **Tftpd > Start** in the menu bar of TFT server window to start the server as shown below;

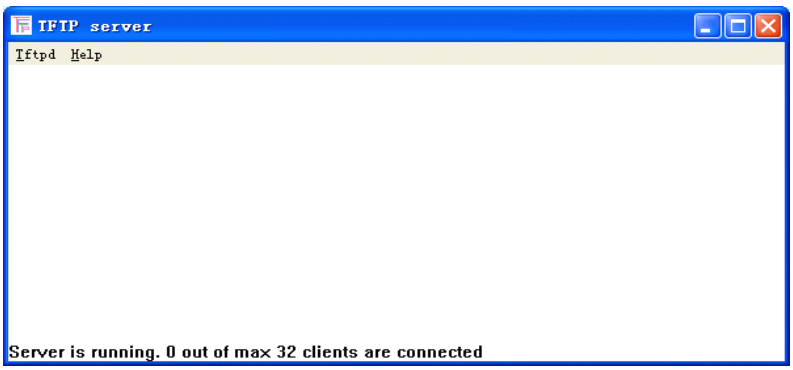


Figure 1-14 Start Server

7） Reboot SBC6845 and press **Space** key on your keyboard when a countdown information is shown in HyperTerminal to enter U-boot command mode as shown below;

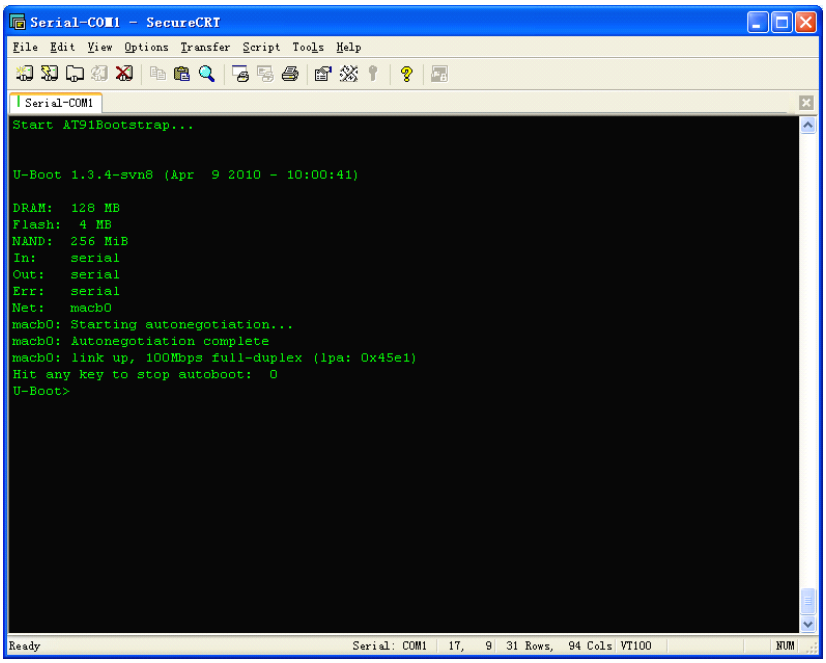


Figure 1-15 U-boot Mode

8） Execute the following instructions to set the IP addresses of TFTP server and the board;

U-Boot> **setenv serverip 192.192.192.71** **#server IP**

U-Boot> **setenv ipaddr 192.192.192.200** **#board IP**

9） Execute the instruction below to erase the memory space at the address 0x800000 in NANDFLASH;

U-Boot> **nand erase 0x800000**

10） Execute the instruction below to scrub the blocks in NANDFLASH;

U-Boot> **nand scrub 0x800000**

(When a prompt message is displayed in U-boot window, type **Y** to confirm and press **Enter** key)

11） Execute the instruction below to download the system file to RAM on the board (it may take a while due to the relatively large size of the file);

U-Boot> **tftp 0x70000000 rootfs.yaffs2**

12） Execute the instruction below to copy the file in RAM to NANDFLASH (it may take a while due to the relatively large size of the file);

U-Boot> **nand write.yaffs 0x70000000 0x800000 $(filesize)**

Now all the images have been burned to the board. The system will be working after rebooting SBC6845.

1.2 Burning System Image Automatically

The operations to realize automatic burning of system images are much easier that manual steps. You only need to copy the relevant Linux images including boot.bin and uboot.bin to the root directory of a SD card, and insert it into the SD slot on the board and then power it up. The system will automatically implement image burning to DATAFLASH and NANDFLASH. After burning process is complete, you just need to reboot SBC6845 to finish the whole work. (Images are saved under \01 Module\SBC6845 Update System Through SD\SD Image_Linux\ of the CD-ROM)

The table shown blow contains of the images required.

Table 1-1 Images Required

Categories	Names	Ways to Make Images
Tool Images	boot.bin	By using tools
	u-boot.bin	By using tools

Categories	Names	Ways to Make Images
System Images	strap.bin	System image, by renaming dataflash_at91sam9g45ekes.bin
	u-boot.bin	System image, u-boot.bin
	logo.bin	System image
	uImage	System image, uImage
	rootfs.bin	System image, by renaming yaffs2 filesystem

Technical Support and Warranty

Technical Support

Embest Technology provides its product with one-year free technical support including:

- Providing software and hardware resources related to the embedded products of Embest Technology;
- Helping customers properly compile and run the source code provided by Embest Technology;
- Providing technical support service if the embedded hardware products do not function properly under the circumstances that customers operate according to the instructions in the documents provided by Embest Technology;
- Helping customers troubleshoot the products.

The following conditions will not be covered by our technical support service. We will take appropriate measures accordingly:

- Customers encounter issues related to software or hardware during their development process;
- Customers encounter issues caused by any unauthorized alter to the embedded operating system;
- Customers encounter issues related to their own applications;
- Customers encounter issues caused by any unauthorized alter to the source code provided by Embest Technology;

Warranty Conditions

1） 12-month free warranty on the PCB under normal conditions of use since

the sales of the product;

2） The following conditions are not covered by free services; Embest Technology will charge accordingly:

A. Customers fail to provide valid purchase vouchers or the product identification tag is damaged, unreadable, altered or inconsistent with the products.

B. Products are damaged caused by operations inconsistent with the user manual;

C. Products are damaged in appearance or function caused by natural disasters (flood, fire, earthquake, lightning strike or typhoon) or natural aging of components or other force majeure;

D. Products are damaged in appearance or function caused by power failure, external forces, water, animals or foreign materials;

E. Products malfunction caused by disassembly or alter of components by customers or, products disassembled or repaired by persons or organizations unauthorized by Embest Technology, or altered in factory specifications, or configured or expanded with the components that are not provided or recognized by Embest Technology and the resulted damage in appearance or function;

F. Product failures caused by the software or system installed by customers or inappropriate settings of software or computer viruses;

G. Products purchased from unauthorized sales;

H. Warranty (including verbal and written) that is not made by Embest Technology and not included in the scope of our warranty should be fulfilled by the party who committed. Embest Technology has no any responsibility;

3） Within the period of warranty, the freight for sending products from customers to Embest Technology should be paid by customers; the freight from Embest to customers should be paid by us. The freight in any direction occurs after warranty period should be paid by customers.

4） Please contact technical support if there is any repair request.

注意：

Embest Technology will not take any responsibility on the products sent back without the permission of the company.

Contact Information

Hotline: +86-755-25635626-872/875

Fax: +86-755-25635626-666

Pre-sales: sales@embedinfo.com

After-sales: support@embedinfo.com

Website: http://www.armkits.com or http://www.embest-tech.com

Address: Tower B 4/F, Shanshui Building, Nanshan Yungu Innovation Industry Park, Liuxian Ave. No. 1183, Taoyuan St., Nanshan District, Shenzhen, China (518055)