Hi3518A/Hi3518C/Hi3518E/Hi3516C Linux Development Environment

User Guide

Issue **01**

Date **2014-02-26**

HiSilicon Technologies Co., Ltd.

Address: Huawei Industrial Base
Bantian, Longgang
Shenzhen 518129
People's Republic of China

Website: http://www.hisilicon.com

Email: support@hisilicon.com

About This Document

Purpose

This document describes the Linux development environment of the Hi3518A/ Hi3518C/ Hi3518E/Hi3516C. This document also explains how to set up the Linux and network development environments, burn the Linux kernel, and root file system, and start Linux-based applications.

After reading this document, customers will clearly understand the Linux development environment.

Related Version

The following table lists the product version related to this document.

Product Name	Version
Hi3518A	V100
Hi3518C	V100
Hi3518E	V100
Hi3516C	V100

Intended Audience

This document is intended for:

- Technical support personnel
- Software development engineers

Change History

Changes between document issues are cumulative. Therefore, the latest document issue contains all changes made in previous issues.

Issue 01 (2014-02-26)

This issue is the first official release, which incorporates the following changes:

Hi3518E information is added.

Issue 00B30 (2012-12-26)

This issue is the third draft release, which incorporates the following changes:

Hi3516C information is added.

Chapter 2 U-boot

Section 2.3 "Compiling the U-boot" is deleted.

Chapter 3 Linux Kernel

In section 3.2, the descriptions of the **CROSS_COMPILE** parameter and two tool chains are added to the **Note** field.

Issue 00B20 (2012-11-25)

This issue is the second draft release, which incorporates the following changes:

Chapter 3 Linux Kernel

In sections 3.2 and 3.3, descriptions are updated.

Issue 00B10 (2012-08-30)

This issue is the first draft release.

Contents

About This Document i

1 Development Environment 1

1.1 Embedded Development Environment 1
1.2 Overview of the Linux Development Environment 2
1.3 Setting Up the Linux Development Environment 3
1.3.1 Installing an OS on the Linux Server 3
1.3.2 Installing the Cross Compiler 3
1.3.3 Installing the SDK 3

2 U-boot 5

2.1 Introduction to the U-boot 5
2.2 Starting the U-boot 5
2.3 Burning the U-boot 6
2.4 Common U-boot Commands 6
2.5 U-boot Environment Variables 12

3 Linux Kernel 15

3.1 Kernel Source Codes 15
3.2 Configuring the Kernel 15
3.3 Compiling the Kernel and Generating the Kernel Image uImage 16

4 Root File System 17

4.1 Introduction to the Root File System 17
4.2 Creating a Root File System by Using the BusyBox 18
4.2.1 Obtaining the Source Code of the BusyBox 18
4.2.2 Configuring the BusyBox 18
4.2.3 Compiling and Installing the BusyBox 19
4.2.4 Creating a Root File System 19
4.3 Introduction to File Systems 20
4.3.1 SquashFS 20
4.3.2 JFFS2 21
4.3.3 NFS 22
4.3.4 YAFFS2 22

5 Burning the Kernel and Root File System 24

5.1 Address Space of the Memory 24
5.2 Burning the Kernel and Root File System Over the ETH Port 24
5.2.1 Setting Parameters and Setting Up the TFTP Service 24
5.2.2 Downloading a Kernel 25
5.2.3 Downloading a Root File System 26
5.3 Burning the Kernel and Root File System Over the Serial Port 27
5.3.1 Connecting Devices 27
5.3.2 Downloading a Kernel 28
5.3.3 Downloading a Root File System 29
6 Starting the Linux Kernel 30
6.1 Setting Boot Parameters and the Auto-Boot Mode 30
7 Introduction to Application Development 32
7.1 Compiling Code 32
7.2 Running Applications 32

Figures

Figure 1-1 Development in embedded mode 1
Figure 1-2 Setting up the Linux development environment 2
Figure 4-1 Structure of the root file system's top directory 17
Figure 5-1 Serial port settings 28
Figure 5-2 Send File dialog box 28

Tables

Table 1-1 Software running in the Linux development environment 2
Table 2-1 Common U-boot commands 6
Table 2-2 NAND flash commands 9
Table 2-3 SPI flash commands 11
Table 2-4 Common U-boot environment variables 12
Table 4-1 Some directories that can be ignored 18
Table 4-2 JFFS2 parameters 21

1 Development Environment

NOTE

The principles and operations of the Linux development environments are almost the same for the Hi3518A, Hi3518C, Hi3518E and Hi3516C. The following uses the Hi3518A as an example to describe chip operations.

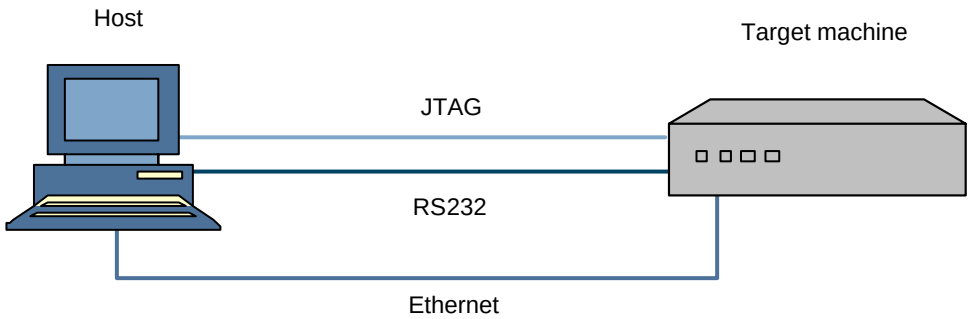
1.1 Embedded Development Environment

The development and debugging tool cannot be run on the embedded board, because the resources of the embedded board are limited. The embedded board is generally developed in cross compilation mode, that is, in host+target machine (evaluation board) mode. The host and target machine are typically connected over the serial port. However, they can also be connected through the network port or Joint Test Action Group (JTAG) interface, as shown in Figure 1-1.

The processors of the host and the target machine are different. A cross compilation environment must be built on the host for the target machine. After a program is processed through compilation, connection, and location, an executable file is created. When the executable file is burnt to the target machine by some means, the program can then run on it.

After the target machine's bootloader is started, operational information about the target machine is transmitted to the host and displayed through the serial port or the network port and displayed. You can control the target machine by entering commands on the host's console.

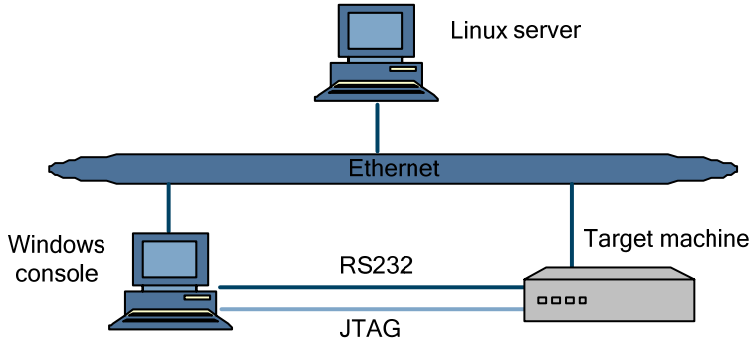
Figure 1-1 Development in embedded mode



1.2 Overview of the Linux Development Environment

The Linux development environment consists of a Linux server, a Windows console, and a demo board (target machine) on the same network, as shown in Figure 1-2.

Figure 1-2 Setting up the Linux development environment



After a cross compilation environment is set up on the Linux server, and the Windows console is connected to the demo board through the serial port or network interface, the developer can develop programs on the Windows console or on the Linux server through remote login. Table 1-1 describes the software running in the Linux development environment.

NOTE

Although a Windows console exists in the development environment, many operations can be completed on the Linux server, such as replacing the HyperTerminal with Minicom. Therefore, you can adjust the development environment to your personal preferences.

Table 1-1 Software running in the Linux development environment

Software		Description
Windows console	Operating system (OS)	Windows 98, Windows me, Windows 2000, or Windows XP.
	Application software	Putty, HyperTerminal, Trivial File Transfer Protocol (TFTP) server, and ARM development suite (ADS)/RealView Debugger.
Linux server	OS	Redhat and Debian are the recommended Linux distributions, however, there are no special requirements. The kernel version 2.6.18 or later is supported. Additionally, full installation is recommended.
	Application software	Network file system (NFS), telnetd, Samba, and VIM, and ARM cross compilation environment (Gcc 4.4). Other application software varies according to the actual development requirements. The required software is pre-installed by default. You need only configure the software before using it.
Target machine	Boot program	Fastboot.

Software		Description
	OS	HiSilicon Linux (HiLinux for short). The HiLinux kernel is developed based on the standard Linux kernel V3.0.y, and the root file system is developed based on the BusyBox V1.16.1.
	Application software	Supports common Linux commands, such as **telnetd** and **gdb server**.
	Program development library	uClibc-0.9.32.1 and glibc-2.11.1.

1.3 Setting Up the Linux Development Environment

1.3.1 Installing an OS on the Linux Server

It is recommended that you install the latest stable release of an OS on the Linux server (such as the RedHat Fedora Core series, SUCE10, or Ubuntu9) to ensure easily available technical support. The following lists some recommended OS versions:

- Later versions of RedHat, such as the RedHat Fedora Core series, Redhat Enterprise Linux, and Red Hat 3.4.4-2.
- Earlier versions of RedHat, such as RedHat 9.0.

The stable releases of Debian are also commonly used. The advantage of Debian is that several types of installation packages are available and can be easily updated online.

1.3.2 Installing the Cross Compiler

CAUTION

A cross compiler obtained from other sources (such as Internet) may not be compatible with the existing kernel, and may result in unexpected problems in the development process.

ARM cross compilers obtained from other sources (such as Internet) can be used, however you must know how to install and use the cross compilation environment. It is strongly recommended to decompress the tool chain in the software development kit (SDK), and run **cross.install** as the **sudo** or **root** user to install the cross compiler.

1.3.3 Installing the SDK

NOTE

Names of SDKs for the Hi3518C, Hi3518E and Hi3516C are Hi3518_SDK_V*XX*.tgz, and *XX* indicates the version number.

The Hi3518A SDK is a software development package based on the REFB. It contains all the tools and source code used in Linux-related development. Therefore, the Hi3518A SDK acts as the basic software platform for chip development.

To install the Hi3518A SDK on a Linux server, perform the following steps:

Step 1 Copy the **Hi3518_SDK_VXX.tgz** package (XX indicates the version number) to the Linux server.

Step 2 Decompress the preceding package by running **tar –zxf Hi3518_SDK_VXX.tgz**.

If no message is displayed during decompression, wait until the command is executed.

Step 3 Open the **Hi3518_SDK_VXX** folder, and run **./ sdk.unpack**.

If you do not have the **root** permission, the system asks you to enter the password of the **root** user or **sudo** user.

----End

2 U-boot

2.1 Introduction to the U-boot

The universal Bootloader (U-boot) is developed based on the U-boot-2010.06.

The Bootloader is a small program that runs before the startup of the kernel. The Bootloader provides the following functions:

- Initializes hardware devices
- Sets up memory mapping.
- Sets up software and hardware environments in which the OS kernel is called.

The U-boot acts as a Bootloader and a burner. When the U-boot is running, you can download the Linux kernel or application programs to the common memory or flash memory over serial ports or Ethernet (ETH) ports.

NOTE

Unless otherwise specified, the U-boot in this document can be used when the system boots from either NAND flash or serial peripheral interface (SPI) flash. In both booting modes, the processes of starting, compiling, and burning the U-boot are the same except that the related commands and environment variables may differ. For details, see section 2.4 "Common U-boot Commands" and section 2.5 "U-boot Environment Variables."

2.2 Starting the U-boot

After the Hi3518A demo board is powered on, a prompt is displayed on the console. The standard inputs and outputs of the Hi3518A demo board are redirected to UART0. UART0 connects to the host. If the host is the Windows console, the serial port terminal is used; if the host is the Linux server, Minicom is used. The settings for connecting to UART0 are as follows:

- Baud rate: 115200
- Data bit: 8
- Parity check: N/A
- Stop bit: 1
- Flow control: N/A

After the system is powered on, the following message is displayed on the console, indicating that the U-boot has started successfully:

```
U-Boot 2010.06 (Mar 31 2011 - 16:27:04)


DRAM:  1 GiB
NAND:  Special NAND ID table Version 1.21
Nand ID: 0xAD 0xDC 0x80 0x95 0xAD 0xDC 0x80 0x95
Nand(Hardware): Block:128K Page:2K Ecc:1bit Chip:512M OOB:64Byte
512 MiB
In:    serial
Out:   serial
Err:   serial
hisilicon #
```

2.3 Burning the U-boot

For details about how to Compile and burn the U-boot, see the *Hi3518A/Hi3518C/Hi3518E/Hi3516C U-boot Porting Application Notes.*

2.4 Common U-boot Commands

Table 2-1 shows common commands of the U-boot.

NOTE

The U-boot supports auto-completion. When you begin typing a command, pressing **Tab** will complete the command or list all probable command options.

Table 2-1 Common U-boot commands

Command	Description
?	Displays a list of all commands or lists the help information about a specified command. Syntax: `? [command ...]` Description: The preceding command is used to list the help information of a command. If only **?** is executed, all commands and descriptions are listed.
help	The **help** command is the same as the **?** command.
printenv	Displays environment variables. Syntax: `printenv [name ...]` Description: The preceding command is used to display environment variables. If only **printenv** is executed, all variables are displayed.
setenv	Sets or deletes variables. Syntax: `setenv name [ value ]` Description: In the preceding command, **name** is the name of a U-boot

Command	Description
	environment variable or a user-defined variable. When **value** is left blank, delete the variable **name**. Otherwise, set **name** to **value**.
saveenv	Saves variables. Syntax: `saveenv` Description: The preceding command is used to save variables and values in the flash memory.
ping	Checks the network status of the destination host or the current machine. Syntax: `ping <ipaddr>` Description: In the preceding command, **ipaddr** is the IP address of the target host. When the network runs properly, the message "host <ipaddr> is alive" is displayed. Otherwise, the message "ping failed; host <ipaddr> is not alive" is displayed.
loadb	Downloads binary files complying with the Kermit protocol over the serial port. Syntax: `loadb [ addr ] [ baud ]` Description: In the preceding command, **addr** is the address for saving files, and **baud** is the download rate of the serial port. After running the command, choose **Transmit** > **Send Files** on the menu of the HyperTerminal, and select **Kermit** in the displayed window. Example: `loadb 0x82000000 115200` Note: Running the **loadb** command downloads files to the double-date rate (DDR) instead of the flash memory.

tftp	Downloads files from the TFTP server to the DDR.
	Syntax: `tftp addr file`
	Description: The downloaded file is saved in the DDR at the address **addr**.
	Note: Before running the **tftp** command, you must configure the network and run the **setenv** command to set the **ipaddr**, **netmask**, and **serverip** parameters.
	For example:
	`hisilicon > setenv ipaddr 192.168.1.1 /*Set the IP address.*/`
	`hisilicon > setenv netmask 255.255.254.0 /*Set the subnet mask.*/`
	`hisilicon > setenv serverip 192.168.1.254 /*Set the IP address of the server.*/`
	`hisilicon > tftp 0x82000000 uImage`
	Description: You need to write the **uImage** file from the TFTP server (the environment variable serverip is the IP address of the server) to the address 0x82000000 of the DDR.

cp	Copies data from the DDR. Syntax: `cp [.b, .w, .l] source target count` Description: The preceding command is used to copy data of size **count** from a source address to a target address. The size of the copied data varies

Command	Description
	according to command options. The details are as follows: Run the **cp.b** command to copy 1 x count bytes. Run the **cp.w** command to copy 2 x count bytes. Run the **cp.l** command to copy 4 x count bytes. The **cp** command is equivalent to the **cp.l command.** Description: **source** and **target** define the address scope of the DDR synchronous dynamic random access memory (SDRAM).
go	Goes to a specified address and executes the codes. Syntax: `go addr [ arg ...]` Description: The preceding command is used to execute the binary code stored in **addr** and transfer the **arg** parameter.
bootm	Sets the running environment and executes the binary codes. Syntax: `bootm [ addr [ arg … ] ]` Description: The preceding command is used to execute the code stored at the **addr** address. The binary code must be processed by running the **mkimage** command.
md	Displays the contents of the DDR. Syntax: `md [.b, .w, .l] address` Description: The preceding command is used to display the contents stored at a specified address of the DDR. The details are as follows: Run the **md.b** command to display one byte. Run the **md.w** command to display two bytes. Run the **md.l** command to display four bytes. The **md** command is equivalent to the **md.1** command.
mm	Modifies the contents of the DDR. The address is incremented automatically. Syntax: `mm [.b, .w, .l] address` Description: The preceding command is used to modify the contents stored at a specified address of the DDR. Run the **mm.b** command to modify one byte once. Run the **mm.w** command to modify two bytes once. Run the **mm.l** command to modify four bytes once. The **mm** command is equivalent to the **mm.1** command.
nm	Modifies the contents stored at a specified address of the DDR. The address is not incremented automatically. Syntax: `nm [.b, .w, .l] address` Description: The preceding command is used to modify the contents stored at a specified address of the DDR. Run the **nm.b** command to modify one byte once. Run the **nm.w** command to modify two bytes once. Run the **nm.l** command to modify four bytes once.

Command	Description
	The **nm** command is equivalent to the **nm.1** command.
mw	Fills in the DDR. Syntax: `mw [.b, .w, .l] address value [ count ]` Description: Set the DDR size starting from **address** with the size of **count** to **value**. Run the **mw.b** command to fill in 1 x count byte of the DDR. Run the **mw.w** command to fill in 2 x count bytes of the DDR. Run the **mw.l** command to fill in 4 x count bytes of the DDR. The **mw** command is equivalent to the **mw.1** command. Example: **`mw.b 0x82000000 FF 10000`** Description: The preceding command is used to fill in the DDR starting from 0x81000000 with the size of 0x10000 bytes to **0xFF**.
cmp	Compares two DDR areas. Syntax: `cmp [.b, .w, .l] addr1 addr2 count` Description: The preceding command is used to compare the contents of two DDR areas whose address are **addr1** and **addr2** respectively. The size of the compared areas is **count**. Run the **cmp.b** command to compare 1 x count bytes. Run the **cmp.w** command to compare 2 x count bytes. Run the **cmp.l** command to compare 4 x count bytes. The **cmp** command is equivalent to the **cmp.1** command.

NOTE

Each command must be entered in one line.

Table 2-2 describes the NAND flash commands.

Table 2-2 NAND flash commands

Command	Description
? nand	Displays the help information about all NAND commands. Syntax: `? nand` Description: The preceding command is used to display all help information pertaining to NAND commands.
nand info	Displays the information about all the NAND devices. Syntax: `nand info` Example: `nand info` Description: The following message is displayed, indicating that a NAND device is detected. Its capacity size is 128 MB, and sector size is 128 KB. Device 0: NAND 128MiB 3,3V 8-bit, sector size 128 KiB

Command	Description
Nand device	Displays the information about a specific NAND device. Syntax: `nand device [device number]` Example: **`nand device 0`** Description: The following message indicates the information about device 0: Device 0: NAND 128MiB 3,3V 8-bit... is now current

Nand write	Writes data to the NAND flash.
	Syntax: `nand write mem_addr start_offset count`
	Example:
	`tftp 0x82000000 u-boot.bin <Download the` **`u-boot.bin`** `file to the DDR>`
	`nand write 0x82000000 0x0 0x100000 <Write the contents in the DDR starting from 0x82000000 to a NAND flash with the offset address 0. The content size is 0x100000.>`
	Note:
	This write operation is implemented through DDR transition. To be specific, before running this command, you must run the **tftp** command in Table 2-1 to download contents to the DDR, and write the contents to the NAND flash.

Nand write.yaffs	Downloads the YAFFS2 file system to the NAND flash. Syntax: `nand write.yaffs mem_addr start_offset fs_size` Example: **`tftp 0x82000000 rootfs-FULL_REL-Flash.yaffs2`** nand write.yaffs 0x82000000 0x300000 0x702600 <This parameter indicates the actual file length of the image of the YAFFS2 file system. Note that the length is a hexadecimal value.> **Note:** After **tftp 82000000 rootfs-FULL_REL-Flash.yaffs2** file is downloaded, the message "Bytes transferred = 7809760 (702600 hex)" is displayed. The **fs_size** parameter must be the actual size of the YAFFS2 file system; otherwise, an error occurs. The definitions of other parameters are the same as that of the **nand write** command.
Nand read	Reads data from the NAND flash to the DDR. Syntax: `nand read mem_addr start_offset count` Example: `nand read 0x82000000 0x100000 0x130000.` Description: The contents from the offset address 0x100000 are read to the DDR starting from 0x82000000. The content size is 0x130000.

Command	Description
Nand erase	Erases the NAND flash. Syntax 1: `nand erase start_offset count` Example: `nand erase 0 100000` Description: The preceding command is used to erase the contents (with the size of 0x100000) starting from the offset address 0x0. Syntax 2: `nand erase start_offset` Example: `nand erase 0x0` Description: The preceding command is used to erase all the spaces starting from 0x0.
Nand bad	Detects the bad sectors of the NAND flash. Syntax: `nand bad` Example: **`nand bad`** Description: The preceding command is used to display the information about the bad sectors detected in the NAND flash.
Nand dump	Displays the data information about the NAND flash. Syntax: `nand dump start_offset` Example: **`nand dump 0x0`** Description: The preceding command is used to display the 2048-byte data information and 64-byte out-of-band (OOB) information starting from the offset address 0.

NOTE

The DDR operation commands (such as **md**, **mw**, and **mm**) and the **go** command do not apply to the NAND flash, because the NAND flash does not support execute in place (XIP).

Table 2-3 describes the SPI flash commands.

Table 2-3 SPI flash commands

Command	Description
? sf	Displays the help information about all sf commands. Syntax: `? sf` Description: The preceding command is used to list the help information about all sf commands.
sf probe	Initializes an SPI flash after the U-boot runs. Note: Before using the SPI flash, you must run this command. Syntax: `sf probe 0` Example: `sf probe 0` Description: The following message is displayed, indicating that an SPI flash has been detected. Its capacity size is 16 MB, and sector size is 256 KB. Output:

Command	Description
	Spi(cs1) ID: 0x20 0x20 0x18 0x00 0x00 0x00 Spi(cs1): Block:256KB Chip:16 MB (Name: M25P128) 16384 KiB hi_sfc at 0:0 is now current device

sf write	Write data to the SPI flash.
	Syntax: `sf write mem_addr start_offset count`
	Example:
	`tftp 0x82000000 u-boot.bin <Download the u-boot.bin file to the DDR>`
	`sf write 0x82000000 0x0 0x100000 <Write the contents starting from 0x82000000 to the SPI flash at the offset address 0. The content size 0x100000.>`
	Note:
	This write operation is implemented through DDR transition. To be specific, before running this command, you must run the **tftp** command in Table 2-1 to download contents to the DDR, and write the contents to the SPI flash.

sf read	Reads data from the SPI flash to the DDR. Syntax: `sf read mem_addr start_offset count` Example: **`sf read 0x82000000 0x100000 0x130000`** Description: The preceding command is used to read the contents from the offset address 0x100000 to the DDR starting from 0x82000000. The content size is 0x130000.
sf erase	Erases the SPI flash. Syntax: `sf erase start_offset count` Example: `SF erase 0 100000` Description: The preceding command is used to erase the contents (with the size of 0x100000) starting from the offset address 0x0.

NOTE

The DDR operation commands (such as **md**, **mw**, and **mm**) and the **go** command do not apply to the SPI flash, because the SPI flash does not support XIP.

2.5 U-boot Environment Variables

You can set U-boot environment variables by running the **setenv** command. Table 2-4 describes the common environment variables and their formats.

Table 2-4 Common U-boot environment variables

Environment Variable	Description
ipaddr	Sets the IP address of the board.

Environment Variable	Description
	Synax: `setenv gatewayip xxx.xxx.xxx.xxx` Example: **`setenv ipaddr 192.168.0.100`** Description: Sets the IP address of the board to **192.168.0.100**.
serverip	Sets the IP address of a TFTP server. Synax: `setenv gatewayip xxx.xxx.xxx.xxx` Example: **`setenv serverip 192.168.0.10`** Description: The preceding command is used to set the IP address of the TFTP server to **192.168.0.10**.
netmask	Sets the subnet mask. Synax: `setenv netmask xxx.xxx.xxx.xxx` Example: `setenv netmask 255.255.255.0` Description: Sets the subnet mask to **255.255.255.0**.
gatewayip	Sets the gateway. Synax: `setenv gatewayip xxx.xxx.xxx.xxx` Example: **`setenv gatewayip 192.168.0.1`** Description: Sets the gateway to **192.168.0.1**.
bootargs	Sets the boot parameters for OS startup. Synax: `setenv bootargs arg1=value1 arg2=value2 … argn=valuen` Example 1: **`setenv bootargs 'mem=64M console=ttyAMA0,115200 root=/dev/mtdblock2 rootfstype=yaffs2 mtdparts=hinand:16M(boot),32M(rootfs),32M(test)'`** Example 2: **`setenv bootargs 'mem=64M console=ttyAMA0,115200 root=/dev/mtdblock1 rootfstype=jffs2 mtdparts=hi_sfc:5M(boot),9M(rootfs),2M(test)'`** Description: Sets the transfer parameters, including the DDR size and the information about the root file system device.
bootcmd	Sets the automatic start mode of the U-boot and the commands to be executed. The boot delay time varies according to the **bootdelay** variable. For details, see the description of the **bootdelay** variable. If the **bootdelay** variable is not set, the delay time is 2s by default. Synax: **`setenv bootcmd cmd1; cmd2; …; cmdn`** Example 1: **`setenv bootcmd 'nand read 0x82000000 100000 400000;bootm 0x82000000'`** Description: After startup, the U-boot automatically reads 0x400000 data from the NAND flash with the offset address 0x82000000, and run the codes stored at address 0x82000000. Example 2: **`setenv bootcmd 'sf probe 0;sf read 0x82000000 0x100000 0x400000;bootm 0x82000000'`** Description: After startup, the U-boot automatically reads 0x400000 data from the SPI flash with the offset address 0x82000000, and run

Environment Variable	Description
	the codes stored at address 0x82000000. Note: Multiple parameters must be separated by semicolons, and the whole parameter string must be surrounded with single quotation marks.
bootdelay	Sets the delay time of auto-boot. The unit is the second. The variable is valid only after the **bootcmd** variable is set. The variable value must be an integer and not smaller than –1. If the value is set to **–1**, the auto-boot function is disabled. Synax: `setenv bootdelay value` Example 1: **`setenv bootdelay 4`** Description: The preceding command is used to set the delay time during automatic booting to **4**s. Example 2: **`setenv bootdelay -1`** Description: The auto-boot function is disabled. Tip: You can press any key to switch to the command-line mode during the delay period. Note: Do not set the delay time to **0** during the development and debugging phases. If you set the delay time to 0, you can press **CTRL**+**C** to interrupt programs immediately when the system boots, and force the system to enter the command-line mode.
mdio_intf	Sets the management data input/output (MDIO) interface mode of the ETH. The Hi3518A supports the media independent interface (MII) and reduced gigabit media independent interface (RGMII) modes. Syntax: `setenv mdio_intf mode` Example 1: **`setenv mdio_intf rgmii`** Description: The preceding command is used to set the MDIO mode of the ETH to RGMII mode. Example 2: **`setenv mdio_intf mii`** Description: The preceding command is used to set the MDIO mode of the ETH to MII mode. Note: The MDIO interface mode depends on the board hardware configurations.
phyaddr	Sets the physical layer (PHY) address of the ETH. Synax: `setenv phyaddr value` Example: **`setenv phyaddr 1`** Description: The preceding command is used to set the PHY address of the ETH to 1. Note: The PHY address depends on the board hardware configurations.

NOTE

Each command must be entered in one line.

3 Linux Kernel

3.1 Kernel Source Codes

After the Hi3518A SDK is installed, the kernel source code is saved in **osdrv/kernel** folder of the SDK. You can open the folder to perform related operations. In addition, another package of the kernel source code is saved in the **packge** folder under the SDK directory. You can also decompress the package, and perform related operations.

3.2 Configuring the Kernel

CAUTION

If you are not familiar with the kernel and Hi3518A platform, do not change the default configuration. However, you can add modules as required.

To configure the kernel, perform the following steps:

Step 1 Copy the **config** file manually:

```
hisilicon$cd osdrv/kernel/linux-3.0.y
hisilicon$cp arch/arm/configs/hi3518a_full_defconfig .config
```

Step 2 Configure the kernel by running the **make menuconfig** command.

```
hisilicon$make ARCH=arm CROSS_COMPILE=arm-hisiv100nptl-linux- menuconfig
```

Step 3 Select modules as required.

Step 4 Save the settings and exit.

----End

NOTE

- Use the **hi3518c_full_defconfig** file to compile the Hi3518C kernel, the **hi3518e_full_defconfig** file to compile the Hi3518E kernel and the **hi3516c_full_defconfig** file to compile the Hi3516C kernel.
- This document describes how to set **CROSS_COMPILE** by using the arm-hisiv100nptl-linux- tool chain (uClibc) as an example:

  arm-hisiv100nptl-linux- tool chain (uClibc): CROSS_COMPILE=arm-hisiv100nptl-linux-

  If you want to use the arm-hisiv200-linux- tool chain (glibc), set **CROSS_COMPILE** as follows: CROSS_COMPILE=arm-hisiv200-linux-
- The Uclibc tool chain supports both full version and tailored version, whereas the Glibc tool chain supports only the full version.

  Full version: hi3518a_full_defconfig

  Tailored version: hi3518a_mini_defconfig

  This document uses the full version hi3518a_full_defconfig as an example. To use the tailored version, change the parameter.

3.3 Compiling the Kernel and Generating the Kernel Image uImage

After settings are saved, run **make ARCH=arm CROSS_COMPILE=arm-hisiv100nptl-linux- uImage** to compile the kernel and generate the kernel image. This may take several minutes.

NOTE

If an error occurs during compilation, run the following commands in sequence:

- make ARCH=arm CROSS_COMPILE=arm-hisiXXX-linux- clean
- make ARCH=arm CROSS_COMPILE=arm-hisiXXX-linux- menuconfig,
- make ARCH=arm CROSS_COMPILE=arm-hisiv100nptl-linux- uImage.

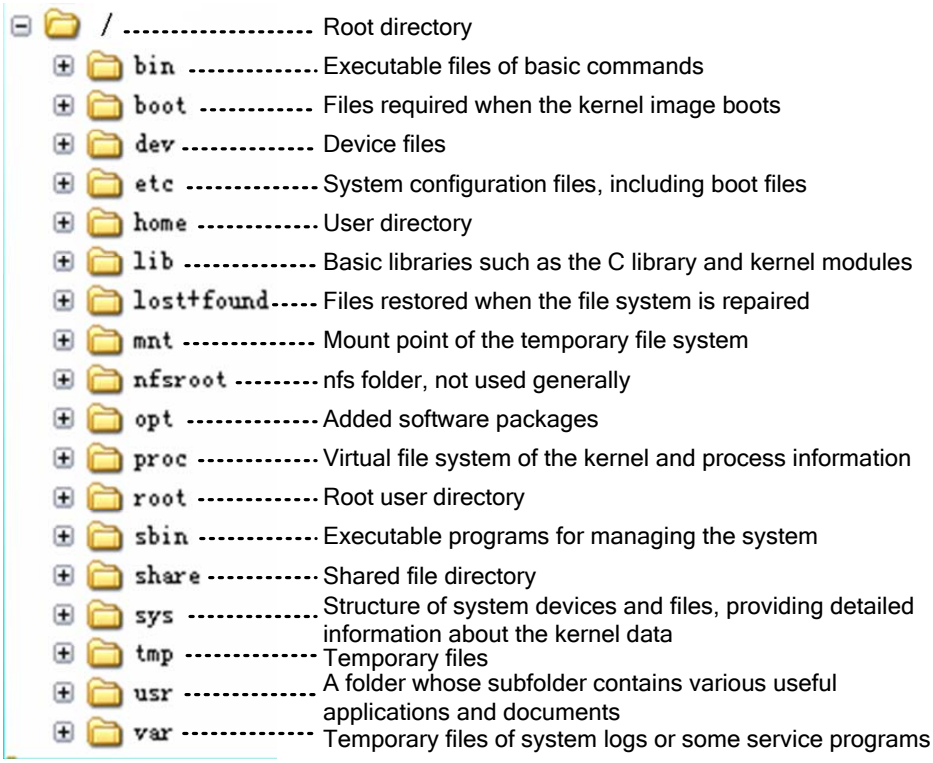
4 Root File System

4.1 Introduction to the Root File System

The top layer of the Linux directory structure is the root directory called "/". After loading the Linux kernel, the system mounts a device to the root directory. The file system of the device is called the root file system. All the mount points of system commands, system configuration, and other file systems are all located in the root file system.

The root file system is typically stored in the common memory, flash memory, or network-based file system. All the applications, libraries, and other services required by the embedded system are stored in the root file system. Figure 4-1 shows the structure of the top directory of the root file system.

Figure 4-1 Structure of the root file system's top directory



A common Linux root file system consists of all the directories in the root file system's top directory structure. In an embedded system, the root file system needs to be simplified. Table 4-1 describes some directories that can be ignored.

Table 4-1 Some directories that can be ignored

Directory	Description
/home, /mnt, /opt, and /root	Directories that can be expanded by multiple users.
/var and /tmp	The **/var** directory stores system logs or temporary service program files. The **/tmp** directory stores temporary user files.
/boot	**The /boot** directory stores kernel images. During startup, the PC loads the kernel from the **/boot** directory. For an embedded system, the kernel images are stored in the flash memory or on the network server instead of the root file system to conserve space. Therefore, this directory can be ignored.

NOTE
Empty directories do not increase the size of a file system. If there is no specific reason, it is recommended to retain these directories.

4.2 Creating a Root File System by Using the BusyBox

Before creating a root file system by using the BusyBox, you need to obtain the BusyBox source code, and then configure, compile, and install the BusyBox.

4.2.1 Obtaining the Source Code of the BusyBox

After the SDK is installed successfully, the complete source code of the BusyBox is saved in **packge** folder. You can also download the source code of the BusyBox from http://www.busybox.net.

4.2.2 Configuring the BusyBox

To configure the BusyBox, you need to go to the directory of the BusyBox, and then run the following command:

```
hisilicon$ make ARCH=arm CROSS_COMPILE=arm-hisiv100nptl-linux- menuconfig
```

The configuration GUI of the BusyBox is the same as that of the kernel. By using the simple and intuitive configuration options, you can configure the BusyBox as required. Pay attention to the following two options that are displayed after you choose **BusyBox Settings > Build Options**:

```
[*]Build BusyBox as a static binary (no shared libs)
(arm-hisiv100nptl-linux-) Cross Compiler prefix
```

Note the following points:

- The first option is used to determine whether to compile the BusyBox as an executable file with a static link. If the option is selected, the compiled BusyBox has a static link. In this case, the BusyBox does not depend on the dynamic library and has a large size. If the option is deselected, the compiled BusyBox has a dynamic link. In this case, the BusyBox has a small size but requires the support of the dynamic library.
- The second option is used to select a cross compiler recommended in the SDK. After configuration, you need to save the settings and close the BusyBox.

For details about the options of the BusyBox, see the *BusyBox Configuration Help*.

4.2.3 Compiling and Installing the BusyBox

To compile and install the BusyBox, run the following commands:

```
hisilicon$ make ARCH=arm CROSS_COMPILE=arm-hisiv100nptl-linux-
hisilicon$ make ARCH=arm CROSS_COMPILE=arm-hisiv100nptl-linux- install
```

If the BusyBox is compiled and installed successfully, the following directories and files are generated in the **_install** directory of the BusyBox:

```
drwxr-xr-x  2 linux  linux  4096 2005-04-22 11:01 bin
lrwxrwxrwx  1 linux  linux  11 2005-04-22 11:01 linuxrc->bin/busybox
drwxr-xr-x  2 linux  linux  4096 2005-04-22 11:01 sbin
drwxr-xr-x  4 linux  linux  4096 2005-04-22 11:01 usr
```

4.2.4 Creating a Root File System

After the SDK is installed successfully, the created root file system **rootfs.tgz** is saved in the **osdrv/rootfs_scripts** directory.

If necessary, you can create a root file system based on the BusyBox.

To create a root file system, perform the following steps:

Step 1 Run the following commands:

```
hisilicon$mkdir rootfs
hisilicon$cd rootfs
hisilicon$cp –R package/osdrv/ busybox/busybox-1.16.1/_intsall/* .
hisilicon$mkdir etc dev lib tmp var mnt home proc
```

Step 2 Provide required files in the **etc**, **lib**, and **dev** directories.

1. For the files in the **etc** directory, see the files in **/etc** of the system. The main files include **inittab**, **fstab**, and **init.d/rcS**. You are recommended to copy these files from the **examples** directory of the BusyBox, and then modify them as required.
2. You can copy device files from the system by running the **cp –R file** command or generate required device files by running the **mknod** command in the **dev** directory.
3. The **lib** directory is used to store the library files required by applications. You need to copy related library files based on applications.

----End

After the preceding steps are completed, a root file system is generated.

NOTE

If you have no special requirements, the configured root file system in the SDK can be used directly. If you want to add applications developed by yourself, you only need to copy the applications and related library files to the corresponding directories of the root file system.

4.3 Introduction to File Systems

In the embedded system, the common file systems include the SquashFS, journaling flash file system v2 (JFFS2), NFS, and YAFFS2. These file systems have the following features:

- SquashFS and JFFS2 have good spatial features; therefore, they apply to embedded applications.
- SquashFS is a read-only file system.
- JFFS2 is a readable/writable file system.
- NFS is suitable for the commissioning phase at the initial stage of development.
- YAFFS2 applies only to the NAND flash.

4.3.1 SquashFS

SquashFS is a read-only Linux system that applies to flash memories. SquashFS provides a high compression rate. The data, nodes, and directories are compressed. It is used when the storage medium is limited.

SquashFS provides the 32-bit UIS or GIDS and file creation time, and supports a maximum of 4 GB capacity. It is easy to use and responds rapidly.

SquashFS is developed by improving Cramfs. Compared with Cramfs, SquashFS provides the following features:

- SquashFS provides a higher compression rate.
- SquashFS responds more rapidly.
- The maximum file size supported is 16 MB for Cramfs and is 4 GB for SquashFS.
- The capacity of Cramfs is limited to 256 MB (the actual capacity is slightly greater than 256 MB), whereas the capacity of SquashFS is 4 GB.

To use SquashFS, add **make ARCH=arm CROSS_COMPILE=arm-hisiv100nptl-linux-menuconfig** to the kernel, and select the following options:

```
File systems  --->
    [*] Miscellaneous filesystems  --->
        <*>  SquashFS 4.0 - Squashed file system support
```

mksquashfs is a tool for generating the SquashFS image. To be specific, mksquashfs processes a created root file system to generate the SquashFS image by running **./mksquashfs ./rootfs ./rootfs.squashfs.img -b 256K**.

where

- **rootfs** is a created root file system.
- **rootfs.squashfs.img** is the generated SquashFS image.
- **-b 256K** indicates that the specified block size of SquashFS is 256 KB.

4.3.2 JFFS2

JFFS2 is on the successor of the JFFS file system created by David Woodhouse of RedHat. JFFS2 is the actual file system used in original flash chips of embedded mini-devices. As a readable/writable file system with structured logs, JFFS2 has the following advantages and disadvantages:

- Advantages: The stored files are compressed. The most important feature is that the system is readable and writable.
- Disadvantages: When being mounted, the entire JFFS2 needs to be scanned. Therefore, when the JFFS2 partition is expanded, the mounting time also increases. Flash memory space may be wasted if JFFS2 format is used. The main causes of this are excessive use of log files and reclamation of useless storage units of the system. The size of wasted space is equal to the size of several data segments. Another disadvantage is that the running speed of JFFS2 decreases rapidly when the memory is full or nearly full due to trash collection.

To load JFFS2, perform the following steps:

Step 1 Scan the entire chip, check log nodes, and load all the log nodes to the buffer.

Step 2 Collate all the log nodes to collect effective nodes and generate a file directory.

Step 3 Search the file system for invalid nodes and then delete them.

Step 4 Collate the information in the memory and release the invalid nodes that are loaded to the buffer.

---End

The preceding features show that system reliability is improved at the expense of system speed. Additionally, flash chips with large capacity, the loading process is slower.

To enable kernel support for JFFS2, you must select the **JFFS2** option when compiling the kernel (the released kernel of HiSilicon supports JFFS2 by default). The process is as follows: After running the **make ARCH=arm CROSS_COMPILE=arm-hisiv100nptl-linux-menuconfig** command, choose **File systems**, select **miscellaneous filesystems**, and then select **Journaling Flash File System v2 (JFFS2) support** (the option is selected in the SDK kernel by default).

To create a JFFS2 file system, run the following command:

```
hisilicon$ mkfs.jffs2 -d ./rootfs -l -e 0x20000 -o  jffs2-root.img
```

You can download the mkfs.jffs2 tool from the Internet or obtain it from the SDK. **rootfs** is a created root file system. Table 4-2 describes the JFFS2 parameters.

Table 4-2 JFFS2 parameters

Parameter	Description
d	Specifies the root file system.
l	Indicates the little-endian mode.
e	Specifies the buffer size of the flash memory.
o	Exports images.

4.3.3 NFS

Before using CRAMFS and JFFS2, you need to burn the image of the root file system to the flash memory. The system loads the image from the flash memory during booting. At the initial stage of system development and porting, applications need to be added or modified frequently. Each time an application is modified, the image needs to be burnt again. This wastes time and affects the lifecycle of the flash memory.

As a distributed file system, NFS is used for sharing files and printers. NFS allows you to share files by invoking the file systems or devices mounted remotely. The usage of the invoked file systems is the same as that of the local file system. NFS adopts the client-server model. In this model, the server exports the directories that need to be shared, and the client can mount the directories and access files in the directories through the network.

When NFS is used as the mounted file system, the kernel mounts a directory exported by the NFS server as its root directory according to the preset kernel command parameters. During this process, no operation needs to be performed on the flash memory. Applications are modified on the Linux server. Therefore, NFS is suitable for the debugging at the initial stage of development.

The method of configuring the NFS file system on the Linux server is as follows: Edit the **exports** configuration file in /etc, add directories and parameters, and then run the **/etc/init.d/nfs start** command to start NFS.

The preceding operations must be performed by a super user and the exported directory must be an absolute path. If the NFS service is started, you only need to restart the NFS service after configuring files, that is, run **/etc/init.d/ nfs restart**.

After configuring the NFS root file system on the Linux server, mount the NFS on the board. The detailed commands are as follows:

```
ifconfig eth0 hw ether 00:10:85:18:01:84            /*Configure the MAC
address.*/
ifconfig eth0 10.85.180.184 netmask 255.255.254.0   /*Configure the IP address
and subnet mask.*/
route add default gw 10.85.180.1                     /*Configure the default
gateway.*/
mount -t nfs -o nolock 10.85.180.133:/home/c54122/glibc-nfs /mnt
                                        /*Mount the NFS folder to the mnt folder
of JFFS2. */
```

4.3.4 YAFFS2

YAFFS2 is an embedded file system designed for the NAND flash. As a file system with structured logs, YAFFS2 provides the loss balance and power failure protection, which ensures the consistency and integrity of the file system in case of power failure.

The advantages and disadvantages of YAFFS2 are as follows:

- Advantages
  - Designed for NAND flash and provides optimized software structure and fast running speed.

  - Stores the file organization information by using the spare area of the hardware. Only the organization information is scanned in the case of system startup. In this way, the system starts fast.
  - Adopts the multi-policy trash recycle algorithm. Therefore, YAFFS2 improves the efficiency and fairness of trash recycle for the loss balance.

- Disadvantages

  The stored files are not compressed. Even when the contents are the same, a YAFFS2 image is greater than a JFFS2 image.

In the SDK, YAFFS2 is provided as a module. To generate the YAFFS2 module, you need to add the path of the related kernel code to the **Makefile** in the YAFFS2 code package, and then perform compilation.

Both the YAFFS2 image and CRAMFS image can be generated by using tools. To generate a YAFFS2 image, run the following command:

```
hisilicon$ mkyaffs2image  ./rootfs./yaffs2-root.img pagesize ecctype
```

Where, **rootfs** is a created root file system, **yaffs2-root.img** is a generated YAFFS2 image, **pagesize** is the page size of the NAND flash welded on the board, and ecctype is the error checking and correcting (ECC) type of the NAND flash.

5 Burning the Kernel and Root File System

5.1 Address Space of the Memory

On the Hi3518A demo board, there are DDR, SPI flash, and NAND flash. The address space of the DDR starts from 0x80000000; the address space of the SPI flash starts from 0x58000000; the address space of the NAND flash start from 0x50000000. The address allocation of flash memories is different. The following is an example using the SPI flash. The size of address space varies according to boards. You can query the actual size by reading the board hardware manuals.

The address space allocation for a 16 MB SPI flash is as follows:

- 0x0–0x100000 are reserved for storing the U-boot.
- 0x100000–0x500000 are reserved for storing the kernel
- 0x500000–0xE00000 are reserved for storing the file system.
- Other spaces are reserved or used for other purposes.

5.2 Burning the Kernel and Root File System Over the ETH Port

Before burning the kernel and root file system over the ETH port, you need to set parameters, set up the TFTP service, and download the kernel and root file system.

5.2.1 Setting Parameters and Setting Up the TFTP Service

You need to connect the ETH port of the Hi3518A demo board by using a network cable, and set related parameters of the U-boot. The U-boot supports the TFTP protocol only. The commands for setting parameters are as follows:

```
hisilicon#setenv serverip 10.85.180.211    /*Set the IP address of the server
as required.*/
hisilicon#setenv ipaddr 10.85.180.130      /*Set the IP address of the
Hi3518A demo board.*/
hisilicon#setenv netmask 255.255.254.0     /*Set the subnet mask.*/
hisilicon#setenv gatewayip 10.85.180.1     /*Set the gateway.*/
hisilicon#saveenv
```

```
hisilicon# ping 10.85.77.69           /*Check whether the network is
normal.*/
```

The U-boot cannot be used to receive broadcasting packets. The U-boot allows you to transmit ping packages and receive corresponding response packages. Therefore, you can check whether the network is normal by pinging the host on the board. After executing the **hisilicon# ping 10.85.77.69** command, if the message "host 10.85.77.69 is alive" displayed, the network is normal; if the message "ping failed; host 10.85.77.69 is not alive" is displayed, the network is abnormal, and you need to check network settings.

In addition, you need to set up the TFTP service on the Windows console or the Linux server. The TFTP service on the Windows console is recommended for its ease of operation.

5.2.2 Downloading a Kernel

This section describes how to upload and download the kernel on the Hi3518A demo board.

To write the kernel to the SPI flash, run the following commands:

hisilicon#sf probe 0 /*Initialize the SPI flash.*/

```
hisilicon#sf erase 0x100000 0x300000 /*Before writing to the SPI flash, you
must erase the SPI flash manually. Otherwise, a data error occurs.*/
hisilicon#tftp 0x82000000 uImage /* Download the uImage file from the TFTP
server to address 0x82000000.*/
```

The following information is displayed on the HyperTerminal if the **kernel-hi3516v100_release.img** file is downloaded successfully:

```
MAC:   00-10-85-18-01-30
TFTP from server 10.85.180.211; our IP address is 10.85.180.130
Download Filename ' uImage'.
Download to address: 0x82000000
Downloading: %T%T%#     [ Connected ]
################################
        0.988 MB download ok.
Bytes transferred = 1012660 (f73b4 hex)
hisilicon#sf write 0x82000000 0x100000 0x300000  /*Write the data from the
address 0x82000000 to the address 0x100000 of the SPI flash. The data length
is 0x300000.*/
```

```
To write the kernel to the NAND flash, run the following commands:
hisilicon#nand erase 0x100000 0xf00000 /*Before writing to the NAND flash,
you must erase the NAND flash manually. Otherwise, a data error occurs.*/
hisilicon#tftp 0x82000000 uImage /*Download the uImage file from the TFTP
server to address 0x82000000. */
hisilicon#nand write 0x82000000 0x100000 0xf00000  /*Write the data from the
address 0x82000000 to the address 0x100000 of the NAND flash. The data length
is 0xF00000.*/
```

5.2.3 Downloading a Root File System

The supported file systems include the SPI flash file system and NAND flash file system.

The commands for downloading a root file system are as follows. The following is an example using the SPI flash:

```
hisilicon#sf probe 0  /*Initialize the SPI flash.*/
hisilicon#sf erase 0x500000 0x900000 /*Erase the file system partition of the SPI flash. For details about the flash partition, see the settings of the boot parameters in section 6.1 "Setting Boot Parameters and the Auto-Boot Mode."*/
hisilicon#tftp 0x82000000 rootfs-FULL_REL.jffs2  /*Download the rootfs-FULL_REL.jffs2 file to the address 0x82000000.*/
```

The following information is displayed on the HyperTerminal if the **rootfs-FULL_REL.jffs2** file is downloaded successfully:

```
MAC:   00-10-85-18-01-30
TFTP from server 10.85.180.211; our IP address is 10.85.180.130
Download Filename 'rootfs-FULL_REL.jffs2'.
Download to address: 0x80800000
Downloading: %# [ Connected ]
#################################       [ 1.000 MB]
#################################       [ 2.000 MB]
#################################       [ 3.000 MB]
#################################       [ 4.000 MB]
#################################       [ 5.000 MB]
#################################       [ 6.000 MB]
###################
        6.591 MB download ok.
Bytes transferred = 6897136 (693df0 hex)
hisilicon#sf write 0x82000000 0x500000 0x900000
```

Writing data to the SPI flash is slow. Therefore, downloading a large file may take a long time. If the prompt "hisilicon#" is displayed, the downloading is complete.

The commands for downloading a root file system are as follows. The following is an example using the NAND flash:

```
hisilicon#nand erase 0x1000000 0x2000000 /*Erase the file system partition of the NAND flash. For details about the flash partition, see the settings of the boot parameters in section 6.1 "Setting Boot Parameters and the Auto-Boot Mode."*/
hisilicon # tftp 82000000 2k1b.yaffs2  /*Download the 2k1b.yaffs2 file to the address 0x82000000.*/
```

The following information is displayed on the HyperTerminal if the **2k1b.yaffs2** file is downloaded successfully:

```
miiphy_register: non unique device name '0:01'
MAC:   00-00-23-34-45-66
TFTP from server 10.85.181.114; our IP address is 10.85.180.193
Download Filename '2k1b.yaffs2'.
Download to address: 0x82000000
Downloading: ################################################
```

```
done
Bytes transferred = 9928512 (977f40 hex)
hisilicon # nand write.yaffs 82000000 1000000 977F40

NAND write: device 0 offset 0x1000000, size 0x977f40
pure data length is 9627648, len_incl_bad is 9699328

 9928512 bytes written: OK
```

Note: 977F40 is actual length of the image of the YAFFS2 file system, and is a hexadecimal value. If the prompt "hisilicon#" is displayed again, the download is complete.

5.3 Burning the Kernel and Root File System Over the Serial Port

Before burning the kernel and root file system over the serial port, you need to connect the Windows console to the Hi3518A demo board over the serial port, and download the kernel and root file system.

5.3.1 Connecting Devices

To connect devices, perform the following steps:

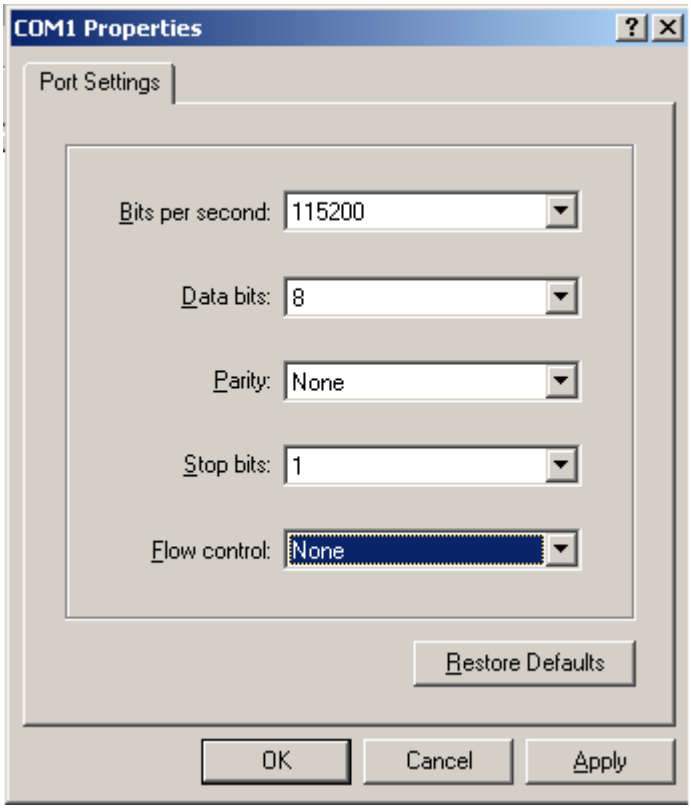
Step 1 Connect COM1 of the Windows console to the COM port of the Hi3518A demo board by using a serial cable (DB9 interface). Note that the serial port COM1 is taken as an example. Other serial ports are also allowed.

Step 2 Start the HyperTerminal installed on the Windows console, and set the parameters of COM1, as shown in Figure 5-1.

Step 3 Start the Hi3518A demo board. If the U-boot command-line interface is displayed, the system runs properly. Then you can download kernels and root file systems or perform other operations.

---End

Figure 5-1 Serial port settings



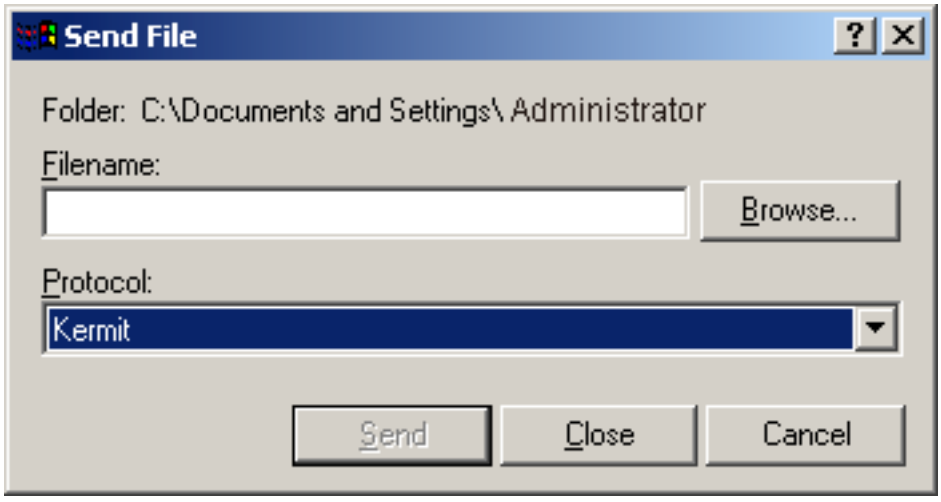
5.3.2 Downloading a Kernel

To download a kernel, perform the following steps:

Step 1 Enter **loadb 0x82000000** (the storage address of the kernel) at the U-boot command prompt of the HyperTerminal

Step 2 Choose Transmit > Send File. The Send File dialog box shown in Figure 5-2 is displayed.

Figure 5-2 Send File dialog box



Step 3 Click **Browse** to select a kernel file.

Step 4 Choose **Kermit** from the **Protocol** drop-down list.

Step 5 Click **Send**.

----End

After the downloading is complete, run the **sf** command of the U-boot to copy the kernel from the DDR to the SPI flash as follows:

```
hisilicon#sf probe 0   /*Initialize the SPI flash.*/
hisilicon#sf erase 0x100000 0x300000          /*Erase the flash memory.*/
hisilicon#sf write 0x82000000 0x100000 0x300000   /*Write the kernel to the
SPI flash at the offset address 0x100000.*/
```

5.3.3 Downloading a Root File System

To download a root file system, perform the following steps:

Step 1 Enter **loadb 0x82000000** (storage address of the image of the root file system) at the U-boot command prompt of the HyperTerminal.

Step 2 Choose **Transmit > Send File**. In the displayed window, select **Kermit** from the **Protocol** drop-down list, and select the image of a root file system in **Filename**. The image can be **rootfs-FULL_REL.jffs2** or **yaffs2-root.img** that is created in Chapter 4 "Root File System."

Step 3 Wait till the downloading is complete, and run the **sf** command of the U-boot to copy the file from the DDR to the flash memory.

The following is an example using the SPI flash:

```
hisilicon#sf probe 0   /*Initialize the SPI flash.*/
hisilicon#sf erase 0x500000 0x900000          /*Erase the SPI flash.*/
hisilicon#sf write 0x82000000 0x500000 0x900000   /*Write the file system
to the SPI flash at the offset address 0x500000.*/
```

---End

NOTE

Using serial ports for downloading is slow yet simple. Therefore, downloading files over serial ports is suitable for small files. Using ETH ports for downloading is fast, which improves efficiency. You are advised to download files over ETH ports.

6 Starting the Linux Kernel

6.1 Setting Boot Parameters and the Auto-Boot Mode

When the kernel is booted from the U-boot, you need to set the parameters of the kernel, including the DDR size and device mounted to the root file system. The parameter settings vary according to file systems. The following setting is only for reference:

The parameters are described as follows:

- mem: Sets the memory size of the OS. mem=64M indicates that the memory size of the OS is 64 MB.
- console: Sets the console. The format is console = ttyAMA0,115200. It indicates that the console uses serial port 0, and the baud rate is 115200.
- root: Sets the device mounted to the file system. root= /dev/mtdblock1 indicates that the file system is mounted from partition 1 of the flash memory. The partition is numbered from 0.
- rootfstype: Sets the type of the mounted root file system.
- mtdparts: Describes the flash partition. The format is mtdparts=hi_sfc:5M(boot),9M(rootfs). It indicates that there are two partitions. Partition **0** is 5 MB and is used for starting the kernel. Partition 1 is 9 MB and is used for the file system.

CAUTION

Each of the following commands must be entered in one line.

JFFS2

If the root file system is JFFS2, the configuration is as follows:

```
hisilicon# setenv bootargs 'mem=64M console=ttyAMA0,115200
root=/dev/mtdblock1 rootfstype=jffs2 mtdparts=hi_sfc:5M(boot),9M(rootfs)'
setenv bootcmd 'sf probe 0;sf read 0x82000000 0x100000 0x400000;bootm
0x82000000'  /*Set the auto-boot parameters.*/
hisilicon#setenv bootdelay 2              /*Set the boot delay to 2s.*/
```

```
hisilicon#saveenv
```

YAFFS2

If the root file system is YAFFS2, the configuration is as follows:

```
hisilicon# setenv bootargs 'mem=64M console=ttyAMA0,115200
root=/dev/mtdblock1 rootfstype=yaffs2
mtdparts=hinand:16M(boot),32M(rootfs),32M(test)'
setenv bootcmd 'nand read 0x82000000 100000 400000;bootm 0x82000000'  /*Set
the auto-boot parameters.*/
hisilicon#setenv bootdelay 2              /*Set the boot delay to 2s.*/
hisilicon#saveenv
```

To reset the board, run the following command in the U-boot command line:

```
hisilicon#reset /*Reset the board to start Linux automatically.*/
```

SquashFS

If the root file system is SquashFS, the configuration is as follows:

```
hisilicon# setenv bootargs 'mem=64M console=ttyAMA0,115200
root=/dev/mtdblock1 rootfstype=squashfs
mtdparts=hi_sfc:5M(boot),9M(rootfs)'
setenv bootcmd 'sf probe 0;sf read 0x82000000 0x100000 0x400000;bootm
0x82000000'  /*Set the auto-boot parameters.*/
hisilicon#setenv bootdelay 2              /*Set the boot delay to 2s.*/
hisilicon#saveenv
```

7 Introduction to Application Development

7.1 Compiling Code

You can choose a code compilation tool as desired. In general, the Source Insight is used in Windows, and Vim+ctags+cscope is used in Linux.

7.2 Running Applications

Before running compiled applications, you need to add them to the target machine and perform the following operations:

- Add the applications and required library files (if any) to the directories of the root file system of the target machine. Generally, applications are placed in the **/bin** directory, library files are placed in the **/lib** directory, and configuration files are placed in the **/etc** directory.
- Creates a root file system containing new applications.

NOTE

Before running applications, you need to write and read the file system. You are recommended to use the YAFFS2 or JFFS2 file system.

It is recommended to use the NFS in the debugging phase so that you do not need to re-create the root file system and burn files. You need to set and start the NFS service (for details, see section 4.3.3 "NFS"), and then mount the NFS directory to the directory of YAFFS2 or JFFS2 by running the following command:

```
mount -t nfs -o nolock serverip:path /mnt
```

The **serverip** parameter indicates the IP address of the server where the NFS directory exists. The **path** parameter indicates the path of the NFS directory on the server. After mounting, applications only need be copied to the NFS system directory before running them on the target machine.

To create SquashFS, YAFFS2 or JFFS2, you need to create corresponding file system (see section 4.3 "Introduction to File Systems"), burn the root file system to the specified address in the flash memory (see Chapter 5 "Burning the Kernel and Root File System"), and set the related boot parameters. In this case, new applications can run properly after Linux starts.

NOTE

To enable new applications to run automatically when the system starts, edit the **/etc/init.d/rcS** file, and then add the paths of the applications to be started.

A Acronyms and Abbreviations

A

ADS	ARM development suite
ARM	advanced RISC machine

D

DMS	digital media solution

E

ECC	error checking and correcting
ETH	Ethernet
ELF	executable and linkable format

F

FC2	Fedora Core 2.0

G

GCC	GNU compiler collection
GDB	GNU debugger
GNU	GNU's not Unix
GUI	graphical user interface

H

HD	high-definition
HILinux	HiSilicon Linux

I

IP	Internet Protocol

J

JFFS2	journaling flash file system V2
JTAG	joint test action group

N

NFS	network file system

O

OS	operating system

P

PC	personal computer

R

REFB	reference board

S

SDRAM	synchronous dynamic random access memory
SDK	software development kit
SPI	synchronous peripheral interface
SquashFS	Squash Files system
SSH	secure shell

T

TFTP	Trivial File Transfer Protocol

Y

YAFFS2	yet another flash file system v2