Hi3516C V500/Hi3516D V300/Hi3516A V300 Development Environment

User Guide

Issue 00B03

Date 2019-01-15

HiSilicon (Shanghai) Technologies Co., Ltd.

Address: New R&D Center, 49 Wuhe Road, Bantian,
Longgang District,
Shenzhen 518129 P. R. China

Website: http://www.hisilicon.com/en/

Email: support@hisilicon.com

About This Document

Purpose

This document describes the Linux and Huawei LiteOS development environment of the Hi3516C V500. This document also explains how to set up the Linux and network development environments, burn the U-Boot, Linux kernel, and root file system, and start Linux-based applications, as well as the configuration and compilation of Huawei LiteOS.

After reading this document, customers will clearly understand the development environment of Linux and Huawei LiteOS.

NOTE

- This document uses Hi3516C V500 as an example. In the subsequent description, if Hi3516D V300/Hi3516A V300 is used, replace Hi3516C V500 with Hi3516D V300/Hi3516A V300, and replace hi3516cv500 with hi3516dv300/hi3516av300.
- The current version supports only Linux.

Related Versions

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

Product Name	Version
Hi3516C	V500
Hi3516D	V300
Hi3516A	V300

Intended Audience

This document is intended for:

- Technical support engineers
- Software development engineers

Change History

Changes between document issues are cumulative. The latest document issue contains all changes made in previous issues.

Issue 00B03 (2019-01-15)

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

The Note description in "About This Document" is modified.

In section 9.1, Figure 9-1 is modified.

Issue 00B02 (2018-10-15)

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

Section 1.3 is modified.

The Note parts in chapter 5, chapter 8, chapter 9, and chapter 10 are added.

Issue 00B01 (2018-08-17)

This issue is the first draft release.

Contents

Figures

Tables

1 Development Environment

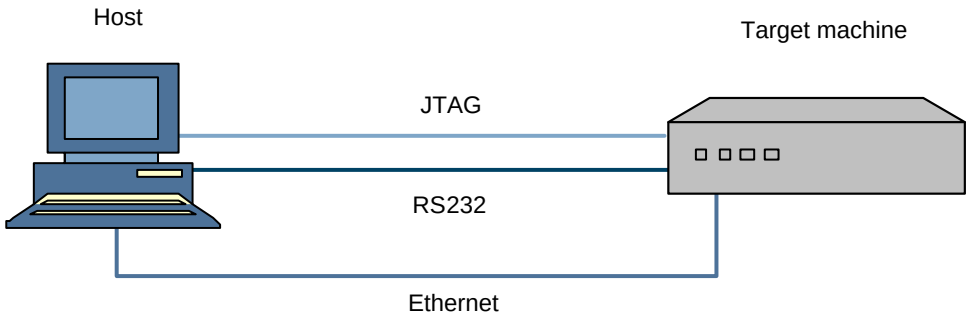
1.1 Embedded Development Environment

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 typically developed in cross compilation mode, that is, in host+target machine (evaluation board) mode. The host and target machine are typically connected through 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.

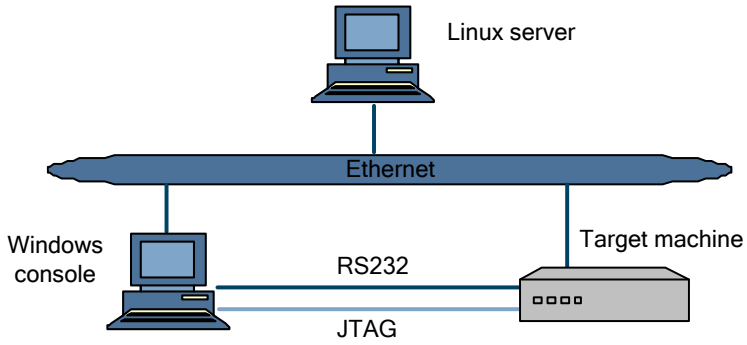
Figure 1-1 Development in embedded mode



1.2 Development Environment

The Hi3516C V500 development environment consists of a Linux server, a Windows console, and a Hi3516CV500DMEB (target machine) on the same network, as shown in Figure 1-2.

Figure 1-2 Setting up the Hi3516C V500 development environment



After a cross compilation environment is set up on the Linux server, and the Windows console is connected to the Hi3516C V500 reference board (REFB) 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 Hi3516C V500 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 Hi3516C V500 development environment

Software		Description
Windows console	Operating system (OS)	Windows XP, Windows 7, or Windows 10
	Application software	Putty, HyperTerminal, Trivial File Transfer Protocol (TFTP) server, and DS-5.
Linux server	OS	Ubuntu, Red Hat, and Debian are supported, and there are no special requirements. The kernel 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 6.3.0). 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.
Hi3516C V500	Boot program	U-Boot

Software		Description
	OS	HiSilicon Linux, Huawei LiteOS. The Linux kernel is developed based on the standard Linux kernel V4.9.y, and the root file system is developed based on the BusyBox V1.26.2.
	Application software	Supports common Linux commands, such as **telnetd** and **gdb server**.
	Program development library	Supports uClibc-0.9.33.2 and glibc-2.24.

1.3 Setting Up the Development Environment

You are advised to use the 64-bit Linux server. This SDK may have unknown incompatibility issues if the 32-bit Linux server, Linux server of an earlier version, or less popular Linux server is used.

The recommended hardware configurations are as follows:

- CPU Intel(R) Xeon(R) CPU E5-2450 0 @ 2.10 GHz or better CPU
- DDR: ≥ 16 GB
- Hard disk ≥ 600 GB
- Gigabit Ethernet
- OS: Ubuntu 16.04 64-bit

1.3.1 Linux Server Version Used by the SDK

In the SDK, the Linux system version in the following example is used for compilation by default.

```
Linux version 4.4.0-104-generic (buildd@lgw01-amd64-022) (gcc version 5.4.0
20160609 (Ubuntu 5.4.0-6ubuntu1~16.04.5) ) #127-Ubuntu SMP Mon Dec 11 12:16:42
UTC 2017
```

NOTE

The procedure of installing the Ubuntu system is not described here.

1.3.2 Establishing the Network Environment

You need to configure the network and install network components such as the NFS, Samba, and SSH by yourself.

1.3.3 Installing the Software Package

After the operating system is installed and the network environment is configured, you can install the related software package by performing the following steps:

Step 1 Use bash by default.

Run the **sudo dpkg-reconfigure dash** command and select **no**.

Step 2 Install the software package.

Run the following command: **sudo apt-get install make libc6:i386 lib32z1 lib32stdc++6 zlib1g-dev libncurses5-dev ncurses-term libncursesw5-dev g++ u-boot-tools:i386 texinfo texlive gawk libssl-dev openssl bc**

Step 3 Create the **/etc/ld.so.preload** file and run the **echo "" > /etc/ld.so.preload** command to solve the problem that some third-party libraries fail to be compiled on the 64-bit Linux server.

----End

1.3.4 Installing the Cross Compiler

NOTICE

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

The SDK provides the compilation toolchains **arm-himix100-linux-** and **arm-himix200-linux-**. **arm-himix100-linux-** indicates the uClibc toolchain on a 32-bit OS, while **arm-himix200-linux-** indicates the glibc toolchain on a 32-bit OS. In this document, **arm-himix*XXX*-linux** is used to represent the two tool chains.

To install the cross compiler, perform the following steps:

Step 1 Decompress the toolchain by running the following command:

tar –xvf arm-himixXXX-linux.tgz

Step 2 Install the toolchain by running **sudo ./arm-himixXXX-linux.install**.

Install other toolchains according to the preceding steps.

----End

1.3.5 Installing the SDK

For details, see the *Description of the Installation and Upgrade of the Hi3516C V500 SDK*.

2 U-Boot

For details, see the *Hi3516C V500/Hi3516D V300/Hi3516A V300 U-boot Porting Development Guide*.

3 Linux Kernel

3.1 Kernel Source Codes

After the Hi3516C V500 SDK is installed, the kernel source code is saved in the **osdrv** folder of the SDK. You can open the folder to perform related operations.

3.2 Configuring the Kernel

NOTICE

If you are not familiar with the kernel and Hi3516C V500 platform, do not change the default configuration. However, you can add modules as required.

To configure the kernel, run the following commands:

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

```
cp arch/arm/configs/hi3516cv500_xxx_defconfig .config
```

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

```
make ARCH=arm CROSS_COMPILE=arm-himixXXX-linux- menuconfig
```

Step 3 Select modules as required.

Step 4 Save the settings and exit.

----End

3.3 Compiling the Kernel and Generating the Kernel Image

After settings are saved, run **make ARCH=arm CROSS_COMPILE=arm-himixXXX-linux-uImage-j 20** to compile the kernel and generate the kernel image uImage. This may take several minutes.

NOTE

If errors occur during kernel compilation, execute commands in the following sequence:

- **make ARCH=arm CROSS_COMPILE=arm-himixXXX-linux- clean**
- **make ARCH=arm CROSS_COMPILE=arm-himixXXX-linux- menuconfig**
- **make ARCH=arm CROSS_COMPILE=arm-himixXXX-linux- uImage.**

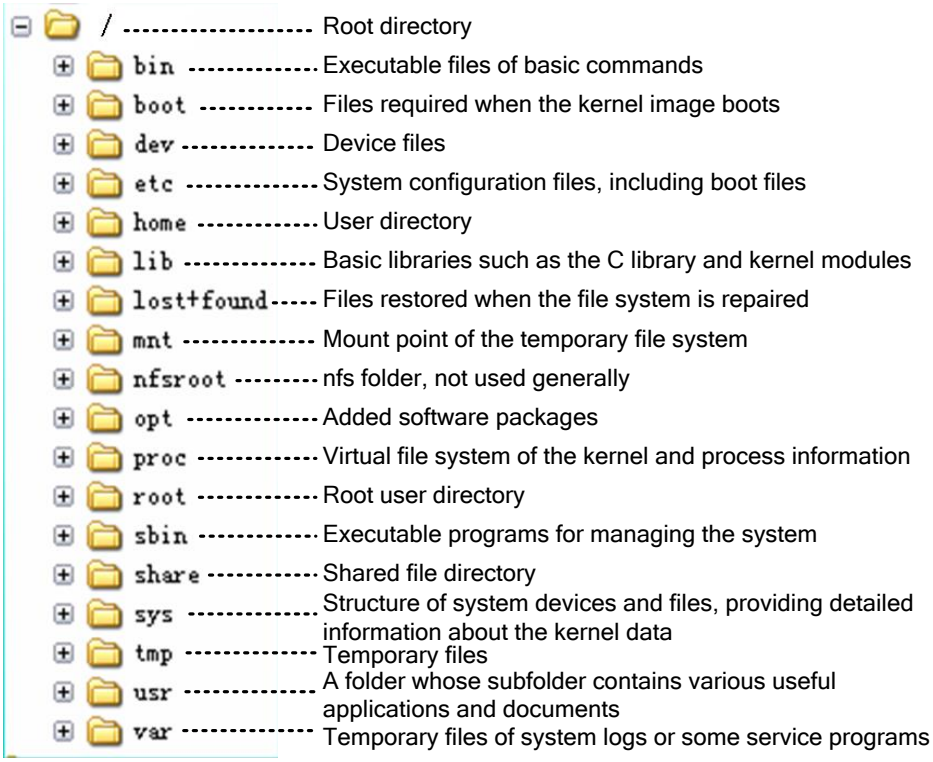
4 Root File System

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, you are advised 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 **osdrv** 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 commands:

- cp osdrv/busybox/busybox-1.26.2/config_XXX_softfp_neon osdrv/busybox/busybox-1.26.2/.config //Specifies the configuration file.

  config_XXX_softfp_neon can be the following:

  - **config_arm_himix100_a7_softfp_neon** corresponds to the multi-core toolchain **arm-himix100-linux-** of the 32-bit operating system.
  - **config_arm_himix200_a7_softfp_neon** corresponds to the multi-core toolchain **arm-himix200-linux-** of the 32-bit operating system.
- make 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 one option that is displayed after you choose **BusyBox Settings > Build Options**:

```
[*] Build with Large File Support (for accessing files > 2 GB)
(arm-himix100-linux) Cross Compiler prefix
()  Path to sysroot
( -mcpu=cortex-a7 -mfloat-abi=softfp -mfpu=neon-vfpv4
-fno-aggressive-loop-optimizations) Additional CFLAGS
()  Additional LDFLAGS
()  Additional LDLIBS
```

NOTE

The first 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:

```
make
make install
```

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

```
drwxrwxr-x  2 xxx XXX 4096 Feb 13 11:41 bin
lrwxrwxrwx  1 xxx XXX   11 Feb 13 11:41 linuxrc -> bin/busybox
drwxrwxr-x  2 xxx XXX 4096 Feb 13 11:41 sbin
drwxrwxr-x  4 xxx XXX 4096 Feb 13 11:41 usr
```

NOTE

xxx indicates the user. *XXX* indicates the group.

4.2.4 Creating a Root File System

After the SDK is installed successfully, the created root file system is saved in the **osdrv/pub** 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:

```
mkdir rootbox
cd rootbox
cp -R packet/os /busybox-1.26.2/_intsall/.
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.

NOTICE

- To facilitate debugging, the root password is not set in the released version by default.
- To ensure system security, customers can set the root password by themselves.

4.3 Introduction to File Systems

In the embedded system, the common file systems include the compressed RAM file system (CRAMFS), journaling flash file system v2 (jffs2), NFS, initrd, YAFFS2, ext4, SquashFS, and Unsorted Block Image File System (UBIFS). These file systems have the following features:

- CRAMFS and JFFS2 have good spatial features; therefore, they are applicable to embedded applications.
- CRAMFS and SquashFS are read-only file systems, which are supported only by the serial peripheral interface (SPI) NOR flash currently.
- SquashFS provides the highest compression rate.
- JFFS2 is a readable/writable file system.
- NFS is suitable for the commissioning phase at the initial stage of development.
- YAFFS2 is applicable only to the NAND flash.
- initrd uses the read-only CRAMFS.
- EXT4 applies to the eMMC.

4.3.1 CRAMFS

CRAMFS is a new file system that was developed based on Linux kernel V2.4 and later. CRAMFS is easy to use, easy to load, and has a high running speed.

The advantages and disadvantages of CRAMFS are as follows:

- Advantages: CRAMFS stores file data in compression mode. When CRAMFS runs, the data is decompressed. This mode can save the storage space in flash memory.

- Disadvantages: CRAMFS cannot directly run on flash memory, because it stores compressed files. When CRAMFS runs, the data needs to be decompressed and then copied to the memory, which reduces read efficiency. Also, CRAMFS is read-only.

For Linux running on the board to support CRAMFS, you must add the **cramfs** option when compiling the kernel. The process is as follows: After running **make menuconfig**, choose **File systems**, select **Miscellaneous filesystems,** and then select **Compressed ROM file system support(cramfs) (OBSOLETE)** (the option is selected in the SDK kernel by default).

The mkfs.cramfs tool is used to create the CRAMFS image. To be specific, the CRAMFS image is generated after you process a created file system by using mkfs.cramfs. This procedure is similar to the procedure for creating an ISO file image using a CD-ROM. The related command is as follows:

```
mkfs.cramfs  ./ rootbox ./cramfs-root.img
```

Where, **rootbox** is a created root file system, and **cramfs-root.img** is the generated CRAMFS image.

4.3.2 JFFS2

JFFS2 is on the successor of the JFFS file system created by David Woodhouse of Red Hat. 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 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:

```
./mkfs.jffs2 -d ./ rootbox -l -e 0x20000 -o  jffs2-root.img
```

You can download the mkfs.jffs2 tool from the Internet or obtain it from the SDK. **rootbox** 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 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:

For the SPI NAND flash memory:

```
./mkyaffs2image100  ./ rootbox yaffs2-root.img [pagesize] [ecctype]
```

For the parallel NAND flash memory:

```
./mkyaffs2image610  ./rootbox yaffs2-root.img [pagesize] [ecctype]
```

Where, **rootbox** 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.

4.3.4 initrd

The initrd is equivalent to the store media and supports the formats such as ext2 and cramfs. Therefore, the kernel must support both initrd and CRAMFS. The following configurations must be complete to enable the initrd to work normally. The following configurations are complete in the SDK by default. If you change the configurations by mistake, do as follows:

- Choose **Device Drivers** > **Block devices**, and select **RAM disk support**.
- Go to **General setup**, and select **Initial RAM filesystem and RAM disk (initramfs/initrd) support**.
- Choose **File systems**, and select **Miscellaneous filesystems** and **Compressed ROM file system support (cramfs) (OBSOLETE)**.

To create an initrd image, perform the following steps:

Step 1 Create a CRAMFS image. For details, see section 4.3.1 "CRAMFS."

Step 2 Create an initrd image based on the created CRAMFS image by running the **mkimage -A arm -T ramdisk -C none -a 0 -e 0 -n cramfs-initrd -d ./cramfs-image cramfs-initrd** command.

----End

4.3.5 Squashfs

Squashfs is a read-only file system based on the Linux kernel and features high compression rate.

Squashfs has the following features:

- Compresses data, inode, and directories.
- Retains 32-bit UID/GIDS and file creation time.
- Supports a maximum of 4 GB file system.
- Detects and deletes duplicate files.

To use squashfs, perform the following steps:

Step 1 Create a kernel image supporting squashfs by going to the **linux-4.9.y** directory and running the following commands:

```
cp arch/arm/configs/hi3516cv500_xxx_defconfig .config
make ARCH=arm CROSS_COMPILE=arm-himixXXX-linux- menuconfig (Save the configurations and exit.)
make ARCH=arm CROSS_COMPILE=arm-himixXXX-linux- uImage
```

Step 2 Create the image of the squashfs file system by using mksquashfs stored in **SDK/package/osdrv/tools/pc**. To use mksquashfs, run the following command:

```
./mksquashfs rootfs ./rootfs.squashfs.img -b 64K -comp xz
```

where **rootfs** is the created root file system; **rootfs.squashfs.img** is the generated image of the squashfs file system; **-b 64K** indicates that the block size of the squashfs file system is 64 KB (which determines the actual block size of the SPI flash); **–comp xz** indicates that the algorithm for compressing the file system is XZ. You need to modify the parameters as required.

----End

4.3.6 EXT4

EXT4 is an efficient, excellent, reliable, and unique file system. Compared with EXT3, EXT4 is optimized in the data structure of the file system.

To use EXT4, perform the following steps:

Step 1 Go to the **linux-4.9.y** directory and run the following commands to create a kernel image supporting EXT4:

```
cp arch/arm/configs/hi3516cv500_xxx_defconfig .config
make ARCH=arm CROSS_COMPILE=arm-himixXXX-linux- menuconfig (Save the
configurations and exit.)
make ARCH=arm CROSS_COMPILE=arm-himixXXX-linux- uImage
```

Step 2 Create the image of the EXT4 file system by running the following command to use make_ext4fs stored in **osdrv/tools/pc**:

```
./make_ext4fs -l 96M -s rootfs.ext4.img rootfs
```

where **-l 96M** indicates that the block size of the EXT4 file system used to configure the eMMC in U-Boot is 96 MB; **-s** indicates that the algorithm for compressing the file system is gzip; **rootfs.ext4.img** is the generated image of the EXT4 file system; **rootfs** is the created root file system. You need to modify the parameters as required.

----End

5 Multi-core Loading Startup

NOTE
This chapter is for reference only in the Linux + Huawei LiteOS dual-system solution.

The system startup steps are as follows:

Step 1 When the system is powered on, A7_cpu0 is started first and runs the U-Boot.

Step 2 Run the **go_cpu1 0X80200000** command in the U-Boot to start the A7 Huawei LiteOS.

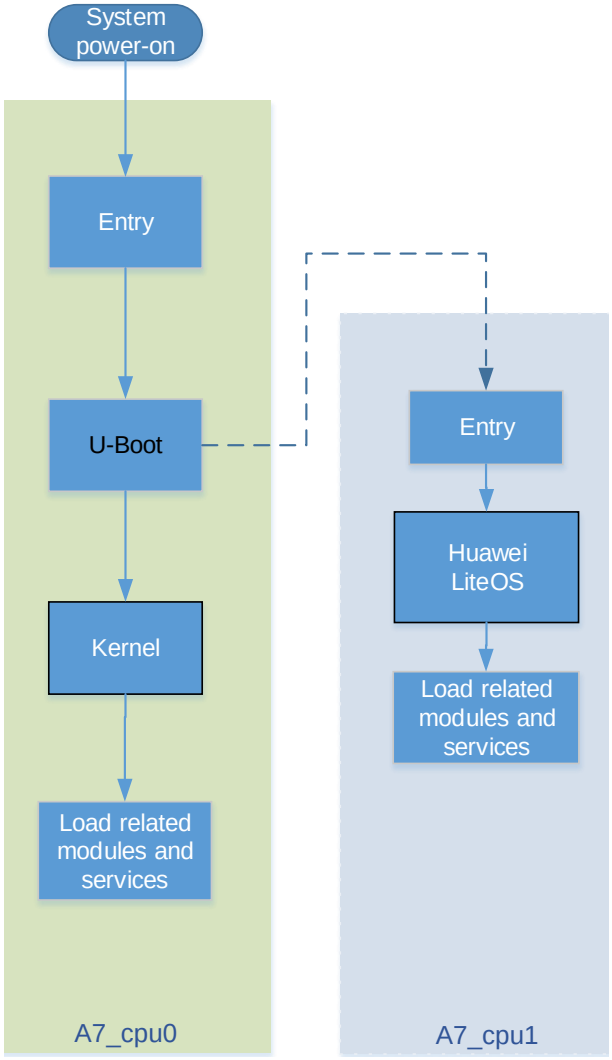
Step 3 Run the **bootm 0xxxxxxxxx** command in the U-Boot to load the kernel image of the A7 Linux and run this kernel.

Figure 5-1 shows the system startup workflow.

NOTICE

When the Linux operating system is started, the image that is downloaded through TFTP or read from the storage medium cannot be located within the memory range of Huawei LiteOS.

Figure 5-1 System startup workflow



----End

6 Huawei LiteOS

6.1 Configuring the Huawei LiteOS

For Huawei LiteOS development, the top priority of the compilation is to install the compilation toolchain **arm-himix100-linux**. For details about the installation method, see section 1.3.4 "Installing the Cross Compiler."

Step 3 Switch the current directory to the Huawei LiteOS directory.

cd platform/liteos/

Step 4 Run **make menuconfig**.

Save the settings and exit.

----End

6.2 Compiling the Huawei LiteOS

After saving the settings, compile the OS library file by entering the **make** command.

7 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.

To create CRAMFS, 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, 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.

8 IPCM

NOTE
This chapter is for reference only in the IPCM solution.

8.1 Introduction

The inter-processor communication module (IPCM) implements the communication between the A7 CUP0 and A7 CUP1.

8.2 IPCM Source Code

The IPCM source code is stored in the **osdrv/components/ipcm** directory. The user can directly enter this directory to perform related operations.

The following shows the structure of the top-level directory for the IPCM source code.

```
├── arch
│   └── hi3516cv500
│       └── configs
├── class
│   ├── cdev
│   ├── net
│   ├── sharefs
│   └── virt-tty
├── include
├── message
├── sysdeps
│   ├── bare
│   ├── linux
```

```
│   └── liteos
└── test
└── readme_cn.txt
└── readme_en.txt
└── Makefile
└── do_make_module
└── makeprepare.sh
```

The preceding directories or files are described as follows:

- **arch**: chip platform related directories or files
- **class ipcm**: package of the functional components used, including **cdev** (character device), **sharefs** (shared file system), and **virt-tty** (virtual serial port terminal)
- **include**: header files
- **message**: code of the communication message layer
- **sysdeps**: system-dependent content, including Linux, Huawei LiteOS, and bare (non-operating system)
- **test**: test cases and samples
- **readme**: usage description

8.3 Node Allocation

The IPCM allocates a node to each core and numbers the node, which is used to send messages during connection setup to discover its peer end. Table 8-1 shows the default allocation mode of the Hi3516C V500.

Table 8-1 Allocated nodes

Node ID	Core	OS
0	A7 CUP0	Linux
1	A7 CUP1	Huawei LiteOS

8.4 IPCM Usage Description

Run the following command in the top-level directory of the IPCM.

```
make PLATFORM=hi3516cv500 CFG=hi3516cv500_xxx_yyyyyy_config
```

Where, **hi3516cv500_xxx_yyyyyy_config** is a configuration file stored in the **arch/hi3516cv500/configs** directory.

After the compilation is completed, the target file is generated in the **out/node** directory. The Linux uses the .ko file and Huawei LiteOS uses the library file.

For details about the configuration of the IPCM and the operation of the device nodes, see the **readme.txt** in the top-level directory of the IPCM.

IPCM compilation depends on OSDRV compilation. When the OSDRV compilation is complete, perform the following operations in the IPCM directory:

- For the Linux, run the following command:

  make PLATFORM=hi3516cv500 CFG=hi3516cv500_a7_linux_config all

  The following data is generated in the **out/node_0** directory:

  node_0
  ├── hi_ipcm.ko
  ├── hi_virt-tty.ko
  ├── libsharefs.a
  ├── libsharefs.so
  ├── sharefs
  └── virt-tty

- For the A7 Huawei LiteOS, run the following command:

  make PLATFORM=hi3516cv500 CFG=hi3516cv500_a7_liteos_config all

  The following data is generated in the **out/node_1** directory:

  node_1
  ├── libipcm.a
  ├── libsharefs.a
  └── libvirt-tty.a

8.5 Virt-tty Virtual Serial Port Terminal

The Hi3516C V500 has multiple operating systems deployed. Developers need to debug each system and check displayed information. Configuring a physical serial port for each system will increase trace routing and costs of hardware boards. To address this issue, the IPCM provides a virtual terminal virt-tty for debugging each system.

The virt-tty code is stored in the **osdrv/components/ipcm/class/virt-tty** directory. After the virt-tty is configured, its target file will be compiled during IPCM compilation. The virt-tty uses port 5 on the IPCM.

The typical application scenario of Virt-tty in Hi3516C V500 is as follows: The A7 Linux functions as the server, and the A7 Huawei LiteOS or DSP functions as the Client. The topology is shown in Figure 8-1. Perform the following steps:

Step 1 Huawei LiteOS links to the **libipcm.a** and **libvirt-tty.a** libraries. And it performs initialization in **app_init**.

_ipcm_vdd_init();

virt_tty_dev_init();

Step 2 See chapter 5 "Multi-core Loading Startup" to start multiple systems.

Step 3 Load the **hi_ipcm.ko** and **hi_virt-tty.ko** file for the A7 Linux system. The virt-tty file is an application.

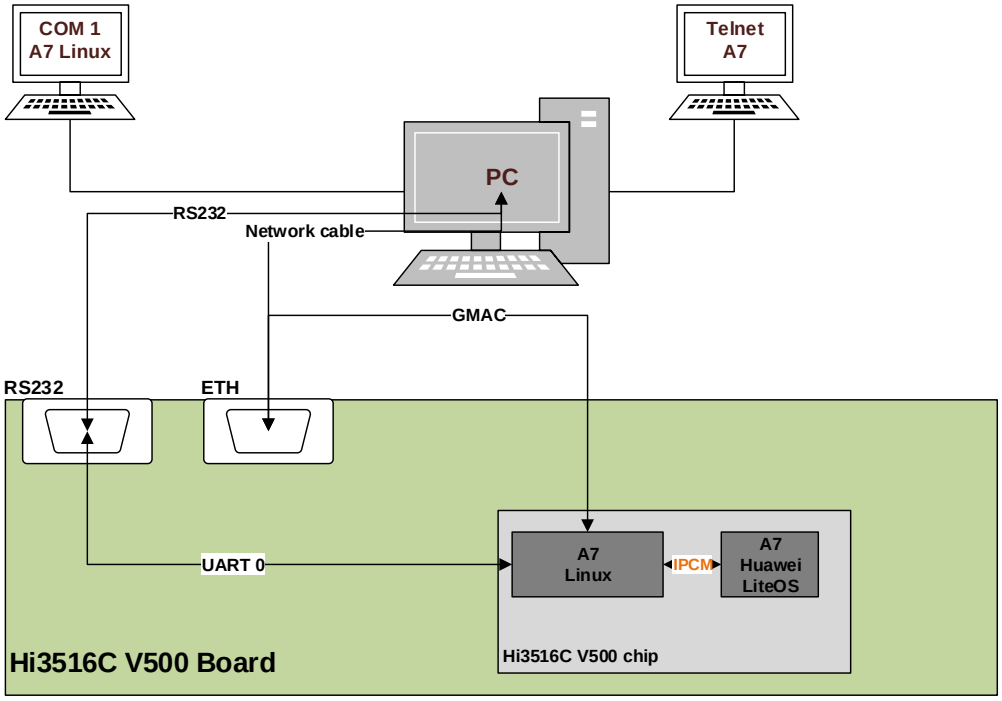
Step 4 Configure a network to connect the A7 Linux system to the PC and enable the telnetd service.

Step 5 Establish Telnet connection to the A7 Linux system on the PC terminal. Run the following command in the Telnet window:

virt-tty a7

The debugging console of A7 Huawei LiteOS is displayed.

Figure 8-1 Hi3516C V500 virt-tty topology



8.6 ShareFS Function

ShareFS allows the A7 Huawei LiteOS to access directories on the A7 Linux. The source code directory is stored in the **osdrv/components/ipcm/class/sharefs** directory. ShareFS uses port 6 on the IPCM.

ShareFS uses the server/client model. The server provides directories to be accessed, receives and executes the file access commands sent by a client, and returns the result to the client. On the client, you can access desired files or directories by calling functions, such as open, read, write, close, cd, ls, and stat. These operations are equivalent to accessing the corresponding directory on the server.

The server is specified as the access directory of ShareFS. It can also be used as the mount point of a USB flash drive, an SD card, or an NFS. In this way, the client can access media such as the USB flash drive, SD card, and NFS.

Perform the following steps:

Step 1 Link the A7 Huawei LiteOS to the **libipcm.a** and **libsharefs.a** libraries. Perform the initialization in app_init.

```
_ipcm_vdd_init();

sharefs_client_init("/sharefs");
```

"/sharefs" is a user-defined input parameter, which indicates the directory on the server where the client is to access.

Step 2 Load the **hi_ipcm.ko** file for the A7 core and run the **sharefs &** command to configure ShareFS as a background program. You can also link the **libsharefs.a** or **libsharefs.so** library to the program and execute the function **sharefs_server_init ()**.

Step 3 Run either of the following commands to configure the A7 Huawei LiteOS to access the **/sharefs** directory on the A7 Linux:

- cd /sharefs
- ls /sharefs

NOTICE

ShareFS access directory (specified by ***"/sharefs"***) to which the client is to access must exist on the server and can be accessed by ShareFS on the server. Otherwise, the client will fail to access this directory.

----End

9 Memory Allocation

NOTE
This chapter is for reference only in the Linux + Huawei LiteOS dual-system solution.

9.1 Memory Allocation

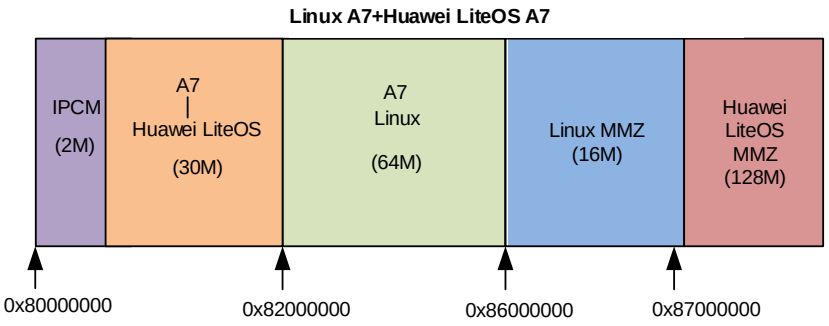
The DDR SDRAM address space of Hi3516C V500 starts from 0x80000000.

The Hi3516C V500 DDR SDRAM can be divided into the following parts:

- Shared memory for inter-core communication
- A7 Linux system memory

Table 9-1 shows the default reference solution to memory allocation provided by the SDK.

Table 9-1 Default memory allocation for dual systems



10 Interrupt Allocation

NOTE
This chapter is for reference only in the Linux + Huawei LiteOS dual-system solution.

10.1 Interrupt Configuration and Allocation

For details about how to allocate interrupt sources, see section 3.4.2 "Interrupt Sources" in the *Hi3516C V500 Professional Smart IP Camera SoC Data Sheet*.

In the Linux+Huawei LiteOS dual-system architecture, the implementation of interrupt allocation in the Huawei LiteOS system is described in the following: **liteos/platform/bsp/board/hi3516cv500/include/irq_map.h**.

Acronyms and Abbreviations

A

ARM	advanced RISC machine

C

CRAMFS	compressed RAM file system

D

DMS	digital media solution

E

ELF	executable and linkable format

G

GCC	GNU compiler collection
GNU	GNU's not Unix

I

IP	Internet Protocol

J

JFFS2	journaling flash file system V2
JTAG	joint test action group

P

PC	personal computer

S

SDRAM	synchronous dynamic random access memory
SDK	software development kit

Y

YAFFS2	yet another flash file system v2