Description of the Installation and Upgrade of the Hi3516C V500/Hi3516D V300/Hi3516A V300 SDK

Issue 01

Date 2019-09-12

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 installation and upgrade of the Hi3516C V500/Hi3516D V300/Hi3516A V300 software development kit (SDK) to facilitate the build of the SDK operating environment on the Hi3516C V500/Hi3516D V300/Hi3516A V300 demo board. The following uses Hi3516C V500 as an example. Unless otherwise specified, the descriptions of Hi3516C V500 also apply to Hi3516D V300/Hi3516A V300.

Related Versions

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

Product	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 earlier issues.

Issue 01 (2019-09-12)

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

Section 2.1.2 is modified.

Issue 00B03 (2019-03-12)

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

The contents related to the Hi3516A V300 are added.

Issue 00B02 (2018-09-30)

This issue is the second draft release.

Step 3 in section 2.1.2 is modified.

Step 2 in section 4.4 is modified.

Section 5.1 and section 5.2 are modified.

Issue 00B01 (2018-09-06)

This issue is the first draft release.

Contents

1 Initial Installation of SDK

If you have installed the software development kit (SDK), see chapter 2 "Installing and Upgrading the Development Environment for the Hi3516C V500 Demo Board."

1.1 Locating the Hi3516C V500 SDK

In the **Hi3516C V500***/01.software/board** directory, you can find the **Hi3516C V500_SDK_Vx.x.x.x.tgz** file, which is the Hi3516C V500 SDK.

1.2 Decompressing the SDK

On a Linux server or a PC running a mainstream Linux operating system (OS), run the **tar -zxf Hi3516CV500_SDK_V*x.x.x.x*.tgz** command to decompress the SDK. Then, you can get the **Hi3516CV500_SDK_V*x.x.x.x*** folder.

1.3 Unpacking the SDK

Return to the **Hi3516CV500_SDK_V*x.x.x.x*** folder, and run **./sdk.unpack** (as the **root** or **sudo** user) to expand the compressed things of the SDK by following the instructions.

To copy the SDK to another location on Windows, run **./sdk.cleanup** first to fold the SDK, copy the SDK to a new directory, and then unfold the SDK.

1.4 Installing the Cross Compiler on a Linux server

Download the toolchain file from the directory where the SDK **Hi3516C V500R001C01SPCxxx.rar** is located.

NOTE

To install the cross compiler, the sudo or root permission is required.

1. Install the himix200 cross compiler.

Run **tar –xzf arm-himix200-linux.tgz**, run **chmod +x arm-himix200-linux.install**, and run **./arm-himix200-linux.install**.

2. Run **source /etc/profile** or log in to the server again for the environment variables to take effect. The environment variables are configured through the scripts that are used to install the cross compiler.

1.5 Compiling OSDRV

See the *readme.txt* in the **osdrv** directory.

1.6 Structure of the SDK Directory

The structure of the **Hi3516C V500_SDK_V*x.x.x.x*** directory is as follows:

```
|-- smp                          #smp directory
    |--a7_linux
        |-- drv                  # drv  directory
        |    |-- extdrv          # Source code of board-level peripheral drivers
        |    |-- interdrv        # Source code of the MIPI, cipher, and other drivers
        |-- mpp                  # Directory that stores the single-core media processing platform (MPP)
        |    |-- component       # mpp components
        |    |    |-- isp        # ISP components
        |    |-- init            # Source code for the initialization of the kernel module
        |    |-- obj             # Object file of the kernel module
        |    |-- include         # External header files
        |    |-- ko              # .ko drivers of the kernel
        |    |-- lib             # User-mode libraries
        |    |-- sample          #Source code sample
        |    |-- tools           # Tools used for media processing
        |    |-- cfg.mak         # Configuration files of the MPP
        |    |-- Makefile.param  # Global compilation option of the MPP
        |    |-- Makefile.linux.param  # Linux compilation option of the MPP
        |-- osal                 # Directory that stores the header files and source files of the OS adaptation layer
        |    |-- include         # Directory that stores the header files of the OS adaptation layer
```

```
|       |-- linux                 # Directory that stores the source files of the OS
adaptation layer
|-- osdrv                         # Directory that stores the OS and related drivers
    |-- component                 # Component source code
    |-- opensource                # Open source code
    |    |-- busybox              # BusyBox source code
    |    |-- kernel               # Source code of the Linux kernel
    |    |-- uboot                # U-Boot source code
    |-- platform                  # Platform files
    |-- pub                       # Compiled images, tools, and drivers
    |-- tools                     # Source code of tools
    |-- readme_cn.txt             # OSDRV application notes (Chinese version)
    |-- readme_en.txt             # OSDRV application notes (English version)
    |-- ......                    #
    |-- Makefile                  # osdrv Makefile
|-- package                       # Directory that stores the compressed SDK packages
    |-- drv.tgz                   # Driver compressed package
    |-- mpp_smp_linux.tgz         # Compressed package of the MPP software
    |-- osal.tgz                  # Compressed package of the source code for the OS
adaptation layer
    |-- osdrv.tgz                 # Source code package of the Linux kernel/U-
Boot/rootfs/tools
|-- scripts                       # Directory that stores the shell scripts
|-- sdk.cleanup                   # Cleanup script of the SDK
|-- sdk.unpack                    # Unpack script of the SDK
```

2 Installing and Upgrading the Development Environment for the Hi3516C V500 Demo Board

This chapter uses the Hi3516C V500 demo board as an example to describe how to burn the U-Boot, kernel, and file system.

2.1 Burning U-Boot, Kernel, and File Systems

2.1.1 Preparations

First of all, read *Hi3516C V500 Demo Board User Guide* to learn the hardware functions, structures, interfaces, and other information about the Hi3516C V500 demo board.

1. If the board lacks U-Boot, you need to use HiTool to burn U-Boot to it. HiTool is stored in **Hi3516C V500*****/01.software/pc/HiTool**. For details about how to use the HiTool, see *HiBurn Tool User Guide* in **ReleaseDoc\zh\01.software\pc\HiTool**.
2. If the board has U-Boot installed, you can perform the following steps to burn the U-Boot, kernel, and rootfs to the flash memory over the network port.

All burning operations in this chapter are performed over the serial port, and the content is burnt to the SPI NOR flash.

2.1.2 Operation Procedure

Step 1 Configure the TFTP server.

You can copy the files in **package/smp_image_uclibc_xxx** or **image_uclibc_xxx** to the directory of any TFTP server.

Step 2 Configure the parameters.

After the board is powered on, press any key to enter U-Boot. Set **serverip** (IP address of the TFTP server), **ipaddr** (IP address of the board), and **ethaddr** (MAC address of the board) by running the following commands:

setenv serverip xx.xx.xx.xx

setenv ipaddr xx.xx.xx.xx

setenv ethaddr xx:xx:xx:xx:xx:xx

setenv netmask xx.xx.xx.xx

setenv gatewayip xx.xx.xx.xx

Ping **serverip** to ensure that the network connection is normal.

Step 3 Burn the images of the SMP version to the SPI NOR flash.

Address space:

1 MB	4 MB	11 MB
boot	kernel	rootfs

The following illustration is based on this address space allocation. Or you can make adjustments according to the actual situation.

1. Burn U-Boot to the flash memory.

   mw.b 82000000 0xff 80000

   tftp 82000000 u-boot-hi3516cv500.bin

   sf probe 0;sf erase 0 80000;sf write 82000000 0 80000

2. Burn the kernel to the flash memory.

   mw.b 82000000 0xff 400000

   tftp 82000000 uImage_hi3516cv500_smp

   sf probe 0;sf erase 100000 400000;sf write 82000000 100000 400000

3. Burn the file system to the flash memory.

   mw.b 82000000 0xff b00000

   tftp 82000000 rootfs_hi3516cv500_64k. jffs2

   sf probe 0;sf erase 500000 b00000;sf write 82000000 500000 b00000

4. Set the bootargs. (Note that the file system is read-only by default for the linux-4.9.y kernel. Therefore, the **rw** option needs to be added to bootargs so that the file system can be read and written.)

   setenv bootargs 'mem=64M console=ttyAMA0,115200 root=/dev/mtdblock2 rootfstype=jffs2 rw mtdparts=hi_sfc:1M(boot),4M(kernel),11M(rootfs)'

   setenv bootcmd 'sf probe 0;sf read 0x82000000 0x100000 0x400000;bootm 0x82000000'

   saveenv

5. Restart the system.

   reset

Step 4 Burn the images of the SMP version to the SPI NAND flash.

A 64 MB SPI NAND flash memory is used as an example. Sub-steps 4 and 5 are for reference only when the yaff2 file system is burnt. Sub-steps 6 and 7 are for reference only when the UBI file system is burnt.

1. Address space:

1 MB	4 MB	32 MB
boot	kernel	rootfs

   The following illustration is based on this address space allocation. Or you can make adjustments according to the actual situation.

2. Burn the U-Boot image.

   mw.b 82000000 0xff 80000

   tftp 82000000 u-boot-hi3516cv500.bin

   nand erase 0x0 0x80000

   nand write 0x82000000 0x0 0x80000

3. Burn the kernel image.

   mw.b 82000000 0xff 400000

   tftp 82000000 uImage_hi3516cv500_smp

   nand erase 0x100000 0x400000

   nand write 0x82000000 0x100000 0x400000

4. Burn the file system to the flash memory.

   mw.b 82000000 0xff 1000000

   tftp 82000000 rootfs_hi3516cv500_4k_24bit.yaffs2

   nand erase 0x500000 0x2000000

   nand write.yaffs 0x82000000 0x500000 0xd06398 #Note: **d06398** is the actual size of the rootfs file in hexadecimal.

5. Set the bootargs. (Note that the file system is read-only by default for the linux-4.9.y kernel. Therefore, the **rw** option needs to be added to bootargs so that the file system can be read and written.)

   setenv bootargs 'mem=64M console=ttyAMA0,115200 root=/dev/mtdblock2 rootfstype=yaffs2 rw mtdparts=hinand:1M(boot),4M(kernel),32M(rootfs)'

   setenv bootcmd 'nand read 0x82000000 0x100000 0x400000;bootm 0x82000000'

   saveenv

6. Burn the UBIFS file system.

   mw.b 0x82000000 0xff 0x3200000

   tftp 0x82000000 rootfs_hi3516cv500_4k_256k_50M.ubifs

   nand erase 0x500000 0x3200000

   nand write 0x82000000 0x500000 0x3200000

7. Set the bootargs. (Note that the file system is read-only by default for the linux-4.9.y kernel. Therefore, the **rw** option needs to be added to bootargs so that the file system can be read and written.)

   setenv bootargs 'mem=64M console=ttyAMA0,115200 ubi.mtd=2 root=ubi0:ubifs rootfstype=ubifs rw mtdparts=hinand:1M(boot),4M(kernel),50M(rootfs.ubifs)'

   setenv bootcmd 'nand read 0x82000000 0x100000 0x400000;bootm 0x82000000'

saveenv

8. Restart the system.

   reset

Step 5 Burn the image files to the eMMC.

The following takes a 1024 MB eMMC as an example.

1. Address space:

1 MB	4 MB	96 MB	923 MB
boot	kernel	rootfs	other

   The following illustration is based on this address space allocation. Or you can make adjustments according to the actual situation.

2. Burn the U-Boot image.

   mw.b 0x82000000 0xff 0x80000

   tftp 0x82000000 u-boot-hi3516cv500.bin

   mmc write 0x0 0x82000000 0x0 0x400

3. Burn the kernel image.

   mw.b 0x82000000 0xff 0x400000

   tftp 0x82000000 uImage_hi3516cv500_smp

   mmc write 0 0x82000000 0x800 0x2000

4. Burn the file system to the flash memory.

   mw.b 0x82000000 0xff 0x6000000

   tftp 0x82000000 rootfs_hi3516cv500_96M.ext4

   mmc write.ext4sp 0 0x82000000 0x2800 0x30000

5. Set the bootargs. (Note that the file system is read-only by default for the linux-4.9.y kernel. Therefore, the **rw** option needs to be added to bootargs so that the file system can be read and written.)

   setenv bootargs 'mem=64M console=ttyAMA0,115200 root=/dev/mmcblk0p3 rw rootfstype=ext4 rootwait blkdevparts=mmcblk0:1M(boot),4M(kernel),96M(rootfs)'

   setenv bootcmd 'mmc read 0x0 0x82000000 0x800 0x2000;bootm 0x82000000'

   saveenv

6. Restart the system.

   reset

----End

3 Preparing the Development Environment

3.1 Pin Multiplexing

None

3.2 Serial Port Connection

Connect to the CPU over the serial port on the demo board.

3.3 NFS Environment

Connect to the Network File System (NFS) over the network port on the demo board.

4 Using the SDK and Demo Board for Development

4.1 Starting the Network in Linux

Step 1 Set the network.

ifconfig eth0 hw ether xx:xx:xx:xx:xx:xx;

ifconfig eth0 xx.xx.xx.xx netmask xx.xx.xx.xx;

route add default gw xx.xx.xx.xx

Step 2 Ping another equipment. If no exception occurs, the network connection is normal.

----End

4.2 Using the NFS

Step 1 Use the NFS as the development environment (recommended). In this way, you do not need to create and burn the root file system again.

Step 2 Mount the NFS by running the following command:

mount -t nfs -o nolock -o tcp -o rsize=32768,wsize=32768 xx.xx.xx.xx:/your-nfs-path /mnt

Step 3 Access the files in the server under **/mnt** for development.

----End

4.3 Enabling the Telnet Service

When the network connection is normal, run the **telnetd&** command to enable the Telnet service of the board, and connect to the board over Telnet.

4.4 Running the MPP Services

Step 1 On the serial port, go to th**e mpp/ko** directory and load the driver. For example,

cd mpp/ko

./ load3516cv500 -i -sensor0 imx327

Step 2 Go to the sample directories, run the sample programs. (The samples must have been compiled successfully on the server.)

cd mpp/sample /vio/smp

./sample_vio 0

----End

5 Allocating and Using the Address Space

5.1 DDR Memory Management

- For the SMP version, the DDR memory consists of the following parts:
  - OS memory, which is managed by the Linux OS
  - Media memory zone (MMZ), which is managed by the osal module and is used by the media service independently
- For the AMP version, the DDR memory consists of the following parts:
  - IPCM memory, which is managed by the Inter-Processor Communications Module (IPCM)
  - Linux OS memory, which is managed by the Linux OS
  - Linux MMZ, which is managed by the osal module in Linux and is used by the media service independently
  - LiteOS memory, which is managed by Huawei LiteOS
  - LiteOS MMZ, which is managed by the osal module in Huawei LiteOS and is used by the media service independently
- For the SMP version, the start address of the OS memory is **0x80000000**. The memory size can be changed by setting bootargs. For example, the parameter **setenv bootargs 'mem=64M ... '** described in 2.1.2 Step 3 indicates that the OS memory allocated to the Linux OS is 64 MB. You can change the OS memory as required. The MMZ is managed by the osal kernel module (**smp/a7_linux/mpp/ko/hi_osal.ko**). When loading the osal module, you can change **mmz_start** and **mmz_size** in the load script to set the start address and size of the MMZ.
- For DDR configuration of the AMP version, see section 5.2 "DDR Memory Management on the Demo Board."
- Note that the divided memories of any area cannot overlap with each other.

5.2 DDR Memory Management on the Demo Board

Take a DDR SDRAM of 256 MB an example. The following shows the memory management obtained based on this document and the default SDK configurations:

SMP Version

DDR:

```
|---------------|------------------------|   0x80000000     # Memory managed by OS
|   64 MB   |   Linux OS             |
|---------------|------------------------|   0x84000000     # Memory managed by MMZ
|   192 MB |       MMZ              |
|---------------|------------------------|   0x90000000
```

Note:

- Set the size of the memory managed by the OS by running **"setenv bootargs 'mem=64M ..."** when configuring bootargs.
  - For Hi3516C V500, the recommended size is 64 MB.
  - For Hi3516D V300/Hi3516A V300, the recommended size is 128 MB.

  You can change the size as required.
- By default, in the **load3516cv500** script (**mpp/ko**), the memory managed by the MMZ starts from the address **0x84000000** and the size is 192 MB. You can modify the settings based on the actual situation.
- By default, in the **load3516dv300/load3516av300** script (**mpp/ko**), the memory managed by the MMZ starts from the address **0x88000000** and the size is 384 MB. You can modify the settings based on the actual situation.
- The address spaces of memory areas for any purpose cannot overlap with each other.
- If required, you can modify the **load3516cv500**, **load3516dv300**, or **load3516av300** script to divide the MMZ. For example,
  insmod hi_osal.ko mmz_allocator=hisi mmz=anonymous,0,0x84000000,32M:jpeg,0,0x86000000,2M

AMP Version

DDR:

```
|---------------|------------------------|   0x80000000     # Memory managed by IPCM.
|   2 MB        |   IPCM                 |
|---------------|------------------------|   0x80200000     # Memory managed by Liteos OS.
|   30 MB     |   Liteos OS            |
|---------------|------------------------|   0x82000000     # Memory managed by Linux OS.
|   96 MB       |   Linux OS             |
|---------------|------------------------|   0x88000000     # Memory managed by Liteos MMZ.
|   336 MB      |       MMZ              |
|---------------|------------------------|   0x9d000000     # Memory managed by Linux MMZ.
|   48 MB     |       MMZ              |
|---------------|------------------------|   0xa0000000
```

Note:

- When configuring bootargs for the OS memory in Linux, set "**setenv bootargs 'mem=96M …**" The MMZ is configured through the **load3516cv500**, **load3516dv300,** or **load3516av300**, script in the **mpp/ko** directory. The configuration method is the same as that of the SMP version.
- The OS memory and MMZ in Huawei LiteOS are configured in the **osdrv/liteos/platform/bsp/board/hi3516cv500/include/board.h** file. The details are as follows:
  - **DDR_MEM_ADDR**: start address of the physical memory. The default value is **0x80000000**.
  - **DDR_MEM_SIZE**: total size of the physical memory. Set this parameter based on the actual situation.
  - **SYS_MEM_BASE**: start address of the OS memory in Huawei LiteOS. The default value is **0x80000000** plus the memory managed by the IPCM.
  - **SYS_MEM_SIZE_DEFAULT**: size of the OS memory in Huawei LiteOS
  - **MMZ_MEM_BASE**: start address of the MMZ in Huawei LiteOS
  - **MMZ_MEM_LEN**: size of the MMZ in Huawei LiteOS

  The MMZ of Huawei LiteOS must be also configured in the **sdk_int.c** file in **mpp/out/amp/a7_liteos/init** as follows:

  static unsigned long long mmz_start = 0x88000000;

  static unsigned int mmz_size = 336; //M Byte

  The preceding variables correspond to the start address and size of the MMZ, respectively. Ensure that the values are the same as those in the **board.h** file.
- The address spaces of memory areas for any purpose cannot overlap with each other.