U-boot Automatic Upgrade

Design Guide

Issue 08

Date 2016-08-31

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 analyzes the automatic upgrade requirements for the SD card and USB flash drive under the U-boot and application scenarios and provides upgrade principles. This document also describes the upgrade design methodology and implementation to provide guidance for function development.

NOTE

Unless otherwise specified, this document applies to the Hi3516A and Hi3516D. Unless otherwise specified, this document applies to the Hi3521A and Hi3520D V300. Unless otherwise specified, this document applies to the Hi3518E V200, Hi3518E V201, and Hi3516C V200.

Related VersionsThe following table lists the product versions related to this document.

Product Name	Version	Remarks
Hi3516C	V100	None
Hi3518	V100	The Hi3518 indicates the Hi3518A or Hi3518C.
Hi3518E	V200	The Hi3518E indicates the Hi3518E V200, Hi3518E V201 or Hi3516C V200.
Hi3520A	V100	None
Hi3521	V100	None
Hi3531	V100	None
Hi3531A	V100	The Hi3531A does not support the image upgrade through the SD card.
Hi3520D	V100	The Hi3520D does not support the image upgrade through the SD card.
Hi3515A	V100	The Hi3515A does not support the image upgrade through the SD card.
Hi3535	V100	The Hi3535 does not support the image upgrade through the SD card.
Hi3516A	V100	None

Product Name	Version	Remarks
Hi3516D	V100	None
Hi3536	V100	The Hi3536 does not support the image upgrade through the SD card.
Hi3521A	V100	The Hi3521A does not support the image upgrade through the SD card.
Hi3520D	V300	The Hi3520D V300 does not support the image upgrade through the SD card.
Hi3519	V100	-
Hi3519	V101	-
Hi3516C	V300	-

Intended Audience

This document is intended for:

- Technical support personnel
- Software development engineers

Symbol Conventions

The symbols that may be found in this document are defined as follows.

Symbol	Description
DANGER	Alerts you to a high risk hazard that could, if not avoided, result in serious injury or death.
WARNING	Alerts you to a medium or low risk hazard that could, if not avoided, result in moderate or minor injury.
CAUTION	Alerts you to a potentially hazardous situation that could, if not avoided, result in equipment damage, data loss, performance deterioration, or unanticipated results.
TIP	Provides a tip that may help you solve a problem or save time.
NOTE	Provides additional information to emphasize or supplement important points in the main text.

Change History

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

Issue 08 (2016-08-31)

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

The description related to Hi3516C V300 is added.

Issue 07 (2016-07-10)

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

The description related to Hi3519 V101 is added.

Issue 06 (2015-10-31)

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

The description related to Hi3519 V100 is added.

Issue 05 (2015-09-01)

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

The description related to the Hi3531A is added.

Issue 04 (2015-08-05)

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

The descriptions related to the Hi3518E V200, Hi3518E V201, and Hi3516C V200 are added.

Issue 03 (2015-06-26)

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

The descriptions related to the Hi3521A and Hi3520D V300 are added.

Issue 02 (2015-04-01)

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

The descriptions related to the Hi3536 are added.

Issue 01 (2014-12-30)

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

The descriptions related to the Hi3516D are added.

Issue 00B03 (2014-10-19)

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

The descriptions related to the Hi3516A are added.

Issue 00B02 (2013-11-18)

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

The descriptions related to the Hi3535 are added.

Issue 00B01 (2013-05-09)

This issue is the first draft release.

Contents

About This Document i

1 Design Analysis 1

1.1 Requirement Analysis 1

1.2 Upgrade Process 1

2 Implementation 3

2.1 Overview 3

2.1.1 Compilation Switches 3

2.1.2 Module Structure 5

2.2 Implementation Details 6

2.2.1 Supporting FAT File Systems 6

2.2.2 Porting the MMC Driver 6

2.2.3 Enabling the USB OHCI 7

2.2.4 Verifying the Ported MMC Driver by Manually Upgrading Images 8

2.2.5 Manually Upgrading and Checking the Ported USB OHCI Driver 10

2.2.6 Adding the Automatic Upgrade Entry Function 12

3 Design Details 13

3.1 Design Details About the Automatic Upgrade Solution 13

3.1.1 Data Description 13

3.1.2 Functions 16

Figures

Figure 1-1 Upgrade process by using an SD card or USB flash drive 2

Figure 3-1 Workflow of int do_auto_update(void) 18

1 Design Analysis

1.1 Requirement Analysis

The previous upgrade solutions only allow you to upgrade images by using upper-layer applications, which brings inconveniences due to low reliability and flexibility. Currently, a more reliable and flexible solution is provided to facilitate the upgrade.

You can obtain images from HiSilicon, and then securely and rapidly upgrade the U-boot, kernel, or rootfs by using an SD card or USB flash drive. The upgrade steps include (taking the USB flash drive as an example):

Step 1 Copying images to a USB flash drive.

Step 2 Connecting the USB flash drive to the board.

Step 3 Restarting the board.

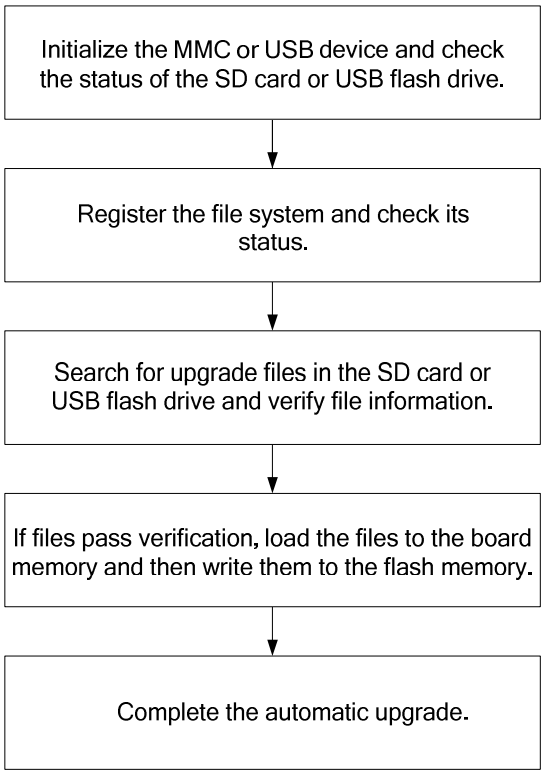
Step 4 Waiting until the images are upgraded.

----End

1.2 Upgrade Process

Figure 1-1 shows the upgrade process by using an SD card or USB flash drive.

Figure 1-1 Upgrade process by using an SD card or USB flash drive



NOTE

When an SD card and USB flash drive are inserted, if there is no valid upgrade package in the SD card, the system automatically upgrades images by using the USB flash drive; if there is a valid upgrade package in the SD card, the system upgrades images by using the upgrade package without scanning the USB flash drive. For details, see section 3.1.2.2 "Implementation of Internal Functions."

2 Implementation

NOTE

- This section uses the Hi3536 as an example. This section uses the Hi3536 as an example; however, the SD card upgrade is described by taking the Hi3516A as an example. The actual code path varies according to the chip model.
- The Hi3520D, Hi3515A, Hi3536, Hi3521A, Hi3520D V300 and Hi3535 support the image upgrade only through the USB flash drive.

2.1 Overview

2.1.1 Compilation Switches

The compilation switches for automatic upgrade are in the chip compilation file. You can enable the corresponding macro definition switches as required.

- Configuration file path
  - include/configs/hi3516c.h [Hi3516C]
  - include/configs/hi3518a.h [Hi3518A]
  - include/configs/hi3518c.h [Hi3518C]
  - include/configs/hi3518ev200.h [Hi3518E V200]
  - include/configs/hi3518ev201.h [Hi3518E V201]
  - include/configs/hi3516cv200.h [Hi3516C V200]
  - include/configs/ hi3520d.h [Hi3520D/Hi3515A]
  - include/configs/ godcare.h [Hi3520A]
  - include/configs/ godarm.h [Hi3521]
  - include/configs/ godnet.h [Hi3531]
  - include/configs/ hi3535.h [Hi3535]
  - include/configs/ hi3516a.h and include/configs/hi3516a_spinand.h [Hi3516A]
  - include/configs/ hi3536.h and include/configs/hi3536_spinand.h [Hi336]
  - include/configs/ hi3521a.h [Hi3521A]
  - include/configs/ hi3531a.h [Hi3531A]
  - include/configs/hi3519.h and include/configs/hi3519_nand.h [Hi3519 V100]
  - include/configs/hi3519v101.h and include/configs/hi3519v101_nand.h [Hi3519 V101]

- include/configs/hi3516cv300.h [Hi3516C V300]

The following describes switch functions by using the Hi3516A as an example.

```
/*-----------------------------------------------------------------
* sdcard/usb storage system update
* -----------------------------------------------------------------*/
#define CONFIG_AUTO_UPDATE                  1     //Main switch for
controlling the upgrade
#ifdef CONFIG_AUTO_UPDATE
       #define CONFIG_AUTO_SD_UPDATE         1     //Switch for
controlling the upgrade images by using an SD card
       #define CONFIG_AUTO_USB_UPDATE        1     //Switch for
controlling the upgrade by using a USB flash drive
#endif
```

//The following four rows indicate the switches for controlling the compilation of FAT file system, which apply to the upgrade by using the SD card and USB flash drive:

```
#define __LITTLE_ENDIAN                     1
#define CONFIG_DOS_PARTITION                1

#define CONFIG_CMD_FAT                      1

/*-----------------------------------------------------------------
* sdcard
* -----------------------------------------------------------------*/
#ifdef CONFIG_AUTO_SD_UPDATE               //  Switch for controlling
the compilation of the SD card driver, which is controlled by the main
switch
       #define CONFIG_HIMCI_HI3516a
       #define REG_BASE_MCI                0x206e0000
       #define CONFIG_HIMCI_V100
       #define CONFIG_GENERIC_MMC
       #define CONFIG_MMC                  1
       #define CONFIG_CMD_MMC
#endif
/*-----------------------------------------------------------------
* usb
*
-----------------------------------------------------------------*/
//The USB driver is enabled by default. CONFIG_AUTO_USB_UPDATE controls
the upgrade by using a USB flash drive.
#define CONFIG_USB_OHCI                     1
#define CONFIG_CMD_USB                      1
#define CONFIG_USB_STORAGE                  1
```

```
#define CONFIG_USB_XHCI                         1
#define CONFIG_SYS_USB_XHCI_MAX_ROOT_PORTS      2
#define CONFIG_LEGACY_USB_INIT_SEQ
```

The preceding macro definitions include:

- Main switch for controlling the upgrade by using an SD card or USB flash drive
- SD card driver
- USB open host controller interface (OHCI) driver

Commenting out or enabling the **CONFIG_AUTO_UPDATE** macro definition determines whether to support the automatic upgrade function.

When **CONFIG_AUTO_UPDATE** is enabled, commenting out or retaining the following two definitions determines whether to support the upgrade by using an SD card or USB flash drive:

```
#define CONFIG_AUTO_SD_UPDATE          1
#define CONFIG_AUTO_USB_UPDATE         1
```

The upgrades by using an SD card and USB flash drive are supported by default. For details about the upgrade policy when an SD card and USB flash drive are simultaneously inserted, see section 3.1 "Design Details About the Automatic Upgrade Solution."

2.1.2 Module Structure

The following describes the code paths of submodules of the module to be upgraded by using the Hi3536 as an example. the SD card upgrade is described by taking the Hi3516A as an example.

For other chips, modify the related code.

The contents in black indicate the files to be modified, and the contents in red indicate the files to be added.

```
----Compilation
Makefile
----Configuration
board/hi3536/board.c       [Program entry]
include/configs/hi3536.h or include/configs/hi3536_spinand.h [Compilation
switch]
-----SD driver
drivers/mmc/himciv100_3516a.c
drivers/mmc/himciv100.c
drivers/mmc/himciv100.h
drivers/mmc/mmc.c
-----USB driver
drivers/usb/host/hiusb/hiusb-3536.c
drivers/usb/host/hiusb/hiusb-ohci.c
drivers/usb/host/hiusb/hiusb-ohci.h
drivers/usb/host/hiusb/hiusb-xhci-3536.c
drivers/usb/host/hiusb/xhci.c
drivers/usb/host/hiusb/xhci.h
```

```
drivers/usb/host/hiusb/xhci-mem.c
drivers/usb/host/hiusb/xhci-ring.c
drivers/usb/host/hiusb/Makefile
----Upgrade main control process
product/hiupdate/Makefile
product/hiupdate/auto_update.c
product/hiupdate/mmc_init.c
product/hiupdate/usb_init.c
```

2.2 Implementation Details

2.2.1 Supporting FAT File Systems

To support FAT file systems, add a compilation switch (for details, see the following contents in red) to the chip configuration file:

```
/*-----------------------------------------------------------------------
 * sdcard/usb storage system update
 *
-----------------------------------------------------------------------*/
#define CONFIG_AUTO_UPDATE                  1
#ifdef CONFIG_AUTO_UPDATE
       #define CONFIG_AUTO_SD_UPDATE         1
       #define CONFIG_AUTO_USB_UPDATE        1
#endif
#define __LITTLE_ENDIAN                     1
#define CONFIG_DOS_PARTITION                1
#define CONFIG_CMD_FAT                      1
```

2.2.2 Porting the MMC Driver

2.2.2.1 Compilation Switch

To add a compilation switch for the MMC driver in the chip configuration file, modify the codes as follows:

```
/*-----------------------------------------------------------------------
* sdcard
* -----------------------------------------------------------------------*/
#ifdef CONFIG_AUTO_SD_UPDATE
       #define CONFIG_HIMCI_HI3516a
       #define REG_BASE_MCI                0x206e0000
       #define CONFIG_HIMCI_V100
       #define CONFIG_GENERIC_MMC
       #define CONFIG_MMC                  1
       #define CONFIG_CMD_MMC
```

```
#endif
```

NOTE

The MMC driver compilation switch **CONFIG_AUTO_SD_UPDATE** is controlled by the upgrade main switch **CONFIG_AUTO_UPDATE**.

2.2.2.2 Porting the Code

The contents in black indicate the files to be modified, and the contents in red indicate the files to be added.

```
The following are the code paths for the Hi3516A MMC driver:
drivers/mmc/himciv100.c
drivers/mmc/himciv100.h
drivers/mmc/mmc.c
drivers/mmc/himciv100_3516a.c
```

The following are the code paths for the MMC drivers of other chips:

```
drivers/mmc/himciv100_3518.c       [Hi3518A and Hi3518E]
drivers/mmc/himciv100_godnet.c     [Hi3531]
drivers/mmc/himciv100_godarm.c     [Hi3521]
drivers/mmc/himciv200.c
drivers/mmc/himciv200.h
drivers/mmc/himciv200_hi3518ev200.c    [Hi3518E V200]
drivers/mmc/himciv200_hi3518ev201.c    [Hi3518E V201]
drivers/mmc/himciv200_hi3516cv200.c    [Hi3516C V200]
drivers/mmc/himciv200_hi3519.c          [Hi3519 V100]
drivers/mmc/himciv200_hi3519v101.c     [Hi3519 V101]
drivers/mmc/himciv200_hi3516cv300.c    [Hi3516C V300]
```

NOTE

Because the Hi3520D, Hi3515A, Hi3536, Hi3535 and Hi3531A do not support the SD card, they do not support the upgrade by using the SD card.

2.2.3 Enabling the USB OHCI

2.2.3.1 Compilation Switch

To add a compilation switch for the USB OHCI driver in the chip configuration file, modify the codes as follows:

```
/*-----------------------------------------------------------------------
* usb
* -----------------------------------------------------------------------*/
#define CONFIG_USB_OHCI                     1
#define CONFIG_CMD_USB                      1
#define CONFIG_USB_STORAGE                   1
#define CONFIG_LEGACY_USB_INIT_SEQ
#define CONFIG_USB_XHCI                        1
#define CONFIG_SYS_USB_XHCI_MAX_ROOT_PORTS     2
```

NOTE

The USB OHCI driver is not controlled by the upgrade main switch **CONFIG_AUTO_UPDATE**, because the driver is compiled into the U-boot by default.

2.2.3.2 Porting the Code

The content in red indicates the file to be added.

The following are the code paths for the Hi3536 USB OHCI and USB XHCI driver:

```
drivers/usb/host/hiusb/Makefile
drivers/usb/host/hiusb/hiusb-ohci.c
drivers/usb/host/hiusb/hiusb-ohci.h
drivers/usb/host/hiusb/hiusb-3536.c
drivers/usb/host/hiusb/hiusb-xhci-3536.c
drivers/usb/host/hiusb/xhci.c
drivers/usb/host/hiusb/xhci.h
drivers/usb/host/hiusb/xhci-mem.c
drivers/usb/host/hiusb/xhci-ring.c
```

The following are the code paths for the USB OHCI drivers of other chips:

```
drivers/usb/host/hiusb/hiusb-3520d.c       [Hi3520A, Hi3520D, and Hi3515A]
drivers/usb/host/hiusb/hiusb-godarm.c      [Hi3521]
drivers/usb/host/hiusb/hiusb-godnet.c      [Hi3531]
drivers/usb/host/hiusb/hiusb-3518.c        [Hi3518A and Hi3518C]
drivers/usb/host/hiusb/hiusb-3535.c        [Hi3535]
drivers/usb/host/hiusb/hiusb-3516a.c       [Hi3536]
drivers/usb/host/hiusb/hiusb-3521a.c       [Hi3521A]
drivers/usb/host/hiusb/hiusb-3518ev200.c   [Hi3518E V200]
drivers/usb/host/hiusb/hiusb-3518ev201.c   [Hi3518E V201]
drivers/usb/host/hiusb/hiusb-3516cv200.c   [Hi3516C V200]
drivers/usb/host/hiusb/hiusb-3531a.c       [Hi3531A]
drivers/usb/host/hiusb/hiusb-3519.c        [Hi3519 V100, Hi3519 V101]
drivers/usb/host/hiusb/hiusb-3516cv300.c   [Hi3516C V300]
```

2.2.4 Verifying the Ported MMC Driver by Manually Upgrading Images

After the operations described in sections 2.2.1 and 2.2.2 are complete, the U-boot generated during compilation supports the **mmcinfo** and **mmc** commands for operating the SD card. After the operations described in sections 2.2.1 and 2.2.3 are complete, you can upgrade the U-boot image by using a USB flash drive to verify the ported USB OHCI driver.

2.2.4.1 Manually Upgrading the Kernel Image by Using an SD Card

This section uses the kernel upgrade as an example to describe how to upgrade images by using an SD card. To upgrade other images such as the image of the U-boot or rootfs, download corresponding upgrade packages to the memory and then burn the packages to the flash memory. To upgrade the kernel by using an SD card, perform the following steps:

Step 1 Burn the U-boot with the ported MMC driver or start the U-boot from the board memory.

Step 2 Insert the SD card that is formatted in FAT32 format and stores upgrade image. For details about how to create upgrade packages, see the *U-boot Automatic Upgrade and Porting User Guide*.

Step 3 Enter the **mmcinfo** command under the U-boot to obtain SD card information and initialize the SD card.

Step 4 Enter **fatls mmc 0** to view the upgrade image in the first partition of the SD card.

Step 5 Enter **fatload mmc 0 0x82000000 kernel** to download the kernel image to 0x82000000 of the board memory.

Step 6 Run **sf probe 0;sf erase 100000 400000;sf write 0x82000000 100000 400000;** to burn the kernel image to a position in the SPI flash or run **bootm 0x82000000** to start the kernel image.

NOTE

The burning position is user-defined. The U-boot or rootfs image is burnt in a similar way.

Step 7 Set and save environment variables based on the burning position.

Step 8 Start the system and view the compilation time displayed when the kernel starts to check whether the kernel image is successfully upgraded.

For other images to be upgraded, view the related compilation time displayed when the images start.

----End

2.2.4.2 Detailed Operations

The following describes detailed operations (the contents in red indicate the commands that need to be executed under the U-boot):

- Start the U-boot with the ported MMC driver.
- Insert the SD card in FAT format that stores upgrade packages.
- Enter **mmcinfo**.

  If the MMC driver is successfully ported, the information similar to the following is displayed:

```
hisilicon # mmcinfo
Device: HIMCI_V100
Manufacturer ID: 3
OEM: 5344
Name: SD01G
Tran Speed: 25000000
Rd Block Len: 512
SD version 2.0
High Capacity: No
Capacity: 1015808000
Bus Width: 4-bit
```

- Enter **fatls mmc 0** to view the upgrade images in the SD card.

```
hisilicon # fatls mmc 0
2902412   kernel
220236   u-boot
3949568   rootfs
```

- Enter **fatload mmc 0 0x82000000 kernel** to download the kernel image to the memory.

```
hisilicon # fatload mmc 0 0x82000000 kernel
reading kernel
2902412 bytes read
```

- Enter **bootm 0x82000000** to start the new kernel from the memory.

```
hisilicon # bootm 0x82000000
## Booting kernel from Legacy Image at 82000000 …
   Image Name:   Linux-3.4.35
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    2902348 Bytes = 2.8 MiB
   Load Address: 80008000
   Entry Point: 80008000
   Loading Kernel Image ... OK
OK
Starting kernel ...
```

If the content similar to the preceding information is displayed, correct data is read from the SD card, and the MMC driver is successfully ported.

2.2.5 Manually Upgrading and Checking the Ported USB OHCI Driver

2.2.5.1 Manually Upgrading the Kernel Image by Using a USB Flash Drive (Taking the Hi3536 as an Example)

To upgrade the kernel by using a USB flash driver, perform the following steps:

Step 1 Burn the U-boot with the ported USB OHCI driver or start the U-boot from the board memory.

Step 2 Insert the USB flash drive that is formatted in FAT32 format and store upgrade image. For details about how to create upgrade packages, see the *U-boot Automatic Upgrade and Porting User Guide*.

Step 3 Enter **usb start** to initialize the USB flash drive.

Step 4 Enter **fatls usb 0** to view the upgrade images in the USB flash drive.

Step 5 Enter **fatload mmc 0 0x42000000 kernel** to download the kernel image to the memory.

Step 6 Run **sf probe 0;sf erase 100000 600000;sf write 0x42000000 100000 600000;** to burn the kernel image to a position in the SPI flash or run **bootm 0x42000000** to start the kernel image.

NOTE

The burning position is user-defined. The U-boot or rootfs image is burnt in a similar way.

Step 7 Set and save environment variables based on the burning position.

Step 8 Start the system and view the compilation time displayed when the kernel starts to check whether the kernel image is successfully upgraded.

For other images to be upgraded, view the related compilation time displayed when the images start.

----End

2.2.5.2 Detailed Operations (Taking the Hi3536 as an Example)

The following describes detailed operations (the contents in red indicate the commands that need to be executed under the U-boot):

- Start the U-boot with the ported USB OHCI driver.
- Insert the USB flash drive in FAT format that stores upgrade packages.
- Enter **USB start**.

  If the USB OHCI driver is successfully ported, the information similar to the following is displayed:

```
hisilicon # usb start
    (Re)start USB...
USB:    scanning bus for devices... 2 USB Device(s) found
scanning bus for storage devices... usb_stor_get_info->1406,blksz:512
1 Storage Device(s) found
```

- Enter **fatls usb 0** to view the upgrade images in the USB flash drive.

```
hisilicon # fatls usb 0
2902412   kernel
220236   u-boot
3949568   rootfs
3 file(s), 0 dir(s)
```

- Enter **fatload usb 0 0x42000000 kernel** to download the kernel image to the board memory.

```
hisilicon # fatload usb 0 0x42000000 kernel
reading kernel
2902412 bytes read
```

- Enter **bootm 0x42000000** to start the new kernel from the board memory.

```
hisilicon # bootm 0x42000000
## Booting kernel from Legacy Image at 42000000 ...
   Image Name:   Linux-3.10.0_hi3536
   Image Type:   ARM Linux Kernel Image (uncompressed)
   Data Size:    3941184 Bytes = 3.8 MiB
   Load Address: 40008000
   Entry Point:  40008000
   Loading Kernel Image ... OK
OK
Starting kernel ...
```

If the content similar to the preceding information is displayed, correct data is read from the USB flash drive, and the USB OHCI driver is successfully ported.

2.2.6 Adding the Automatic Upgrade Entry Function

The following describes how to add the automatic upgrade entry function by taking the Hi3536 as an example. If you want to add the function for other chips, replace the corresponding paths.

Add the automatic upgrade entry function to the **board.c** file in **board/hi3536/**. The contents in red need to be modified. For details, see chapter 3 "Design Details."

```
int misc_init_r(void)
{
#ifdef CONFIG_RANDOM_ETHADDR
       random_init_r();
#endif
       setenv("verify", "n");
#ifdef CONFIG_AUTO_UPDATE
       extern int do_auto_update(void);
#ifdef CFG_MMU_HANDLEOK
       dcache_stop();
#endif
       do_auto_update();    //The automatic upgrade entry function is called
after the U-boot starts.
#ifdef CFG_MMU_HANDLEOK
       dcache_start();
#endif
#endif /* CONFIG_AUTO_UPDATE */
       return 0;
}
```

3 Design Details

3.1 Design Details About the Automatic Upgrade Solution

3.1.1 Data Description

3.1.1.1 Related Data

Because only basic driver functions are ported, the following instances involve only the boot, kernel, and rootfs.

1. Familiarize yourself with the following definitions in the **image.h** file in **include/** before learning about implementation of the automatic upgrade function:

```
/*
* Image types
*
* "Standalone Programs" are directly runnable in the environment
*   provided by U-Boot; it is expected that (if they behave
*   well) you can continue to work in U-Boot after return from
*   the Standalone Program.
* "OS Kernel Images" are usually images of some Embedded OS which
*   will take over control completely. Usually these programs
*   will install their own set of exception handlers, device
*   drivers, set up the MMU. - this means, that you cannot
*   expect to re-enter U-Boot except by resetting the CPU.
* "RAMDisk Images" are more or less just data blocks, and their
*   parameters (address, size) are passed to an OS kernel that is
*   being started.
* "Multi-File Images" contain several images, typically an OS
*   (Linux) kernel image and one or more data images like
*   RAMDisks. This construct is useful for instance when you want
*   to boot over the network using BOOTP, where the boot
*   server provides just a single image file, but you want to get
*   for instance an OS kernel and a RAMDisk image.
*
```

```
*   "Multi-File Images" start with a list of image sizes, each
*   image size (in bytes) specified by an "uint32_t" in network
*   byte order. This list is terminated by an "(uint32_t)0".
*   Immediately after the terminating 0 follow the images, one by
*   one, all aligned on "uint32_t" boundaries (size rounded up to
*   a multiple of 4 bytes - except for the last file).
*
* "Firmware Images" are binary images containing firmware (like
*   U-Boot or FPGA images) which usually will be programmed to
*   flash memory.
*
* "Script files" are command sequences that will be executed by
*   U-Boot's command interpreter; this feature is especially
*   useful when you configure U-Boot to use a real shell (hush)
*   as command interpreter (=> Shell Scripts).
*/
#define IH_TYPE_INVALID0   /* Invalid image    */
#define IH_TYPE_STANDALONE 1   /* Standalone program      */
#define IH_TYPE_KERNEL 2   /* OS kernel image     */
#define IH_TYPE_RAMDISK3   /* RAMDisk image       */
#define IH_TYPE_MULTI  4   /* Multi-file image*/
#define IH_TYPE_FIRMWARE   5   /* Firmware image   */
#define IH_TYPE_SCRIPT 6   /* Script file  */
#define IH_TYPE_FILESYSTEM 7   /* File system image (any type)*/
#define IH_TYPE_FLATDT 8   /* Binary flat device tree blob*/
```

The **T** parameter in the **mkimage** command specifies the image type. Table 3-1 describes the mapping between image types and the **mkimage** command.

Table 3-1 Mapping between image types and the mkimage command

Image Type	mkimage Command
u-boot.bin	**mkimage –T firmware** (corresponding to IH_TYPE_FIRMWARE)
kernel	**mkimage –T kernel** (corresponding to IH_TYPE_KERNEL)
rootfs	**mkimage –T filesystem** (corresponding to IH_TYPE_FILESYSTEM)

2. The following code defines the indexes of the images to be updated and the total number of images:

```
/*Index of each file in the following arrays */
#define IDX_FIRMWARE 0
#define IDX_KERNEL 1
#define IDX_ROOTFS 2
/*Maximum number of files which could interest us */
```

```
#define AU_MAXFILES 3
```

3. The following shows the names of the images to be updated:

```
#define AU_FIRMWARE"u-boot"
#define AU_KERNEL      "kernel"
#define AU_ROOTFS      "rootfs"
/*Pointers to file names */
char *aufile[AU_MAXFILES] = {
AU_FIRMWARE,
AU_KERNEL,
AU_ROOTFS
};
```

Note that the preceding file names are consistent with the names of files stored in the USB flash drive. After the files in the USB flash drive are enumerated, the automatic upgrade program searches for images by file name.

4. Storage positions of images in the flash memory:

```
struct flash_layout
{
long start;
long end;
};
/*Layout of the FLASH. ST = start address, ND = end address. */
#define AU_FL_FIRMWARE_ST  0x0
#define AU_FL_FIRMWARE_ND  0x7FFFF
#define AU_FL_KERNEL_ST0x100000
#define AU_FL_KERNEL_ND0x5FFFFF
#define AU_FL_ROOTFS_ST0x600000
#define AU_FL_ROOTFS_ND0xbFFFFF

/*Sizes of flash areas for each file */
long ausize[AU_MAXFILES] = {
(AU_FL_FIRMWARE_ND + 1) - AU_FL_FIRMWARE_ST,
(AU_FL_KERNEL_ND + 1) - AU_FL_KERNEL_ST,
(AU_FL_ROOTFS_ND + 1) - AU_FL_ROOTFS_ST,
};

/*Array of flash areas start and end addresses */
struct flash_layout aufl_layout[AU_MAXFILES] = {
{ AU_FL_FIRMWARE_ST,   AU_FL_FIRMWARE_ND, },
{ AU_FL_KERNEL_ST,  AU_FL_KERNEL_ND,   },
{ AU_FL_ROOTFS_ST,  AU_FL_ROOTFS_ND,   },
};
```

5. Start address for the required memory. The start address is required for writing the upgraded images in the USB flash drive to the board memory.

```
/*Where to load files into memory */
#define LOAD_ADDR ((unsigned char *)0x82000000)
/*The app is the largest image */
#define MAX_LOADSZ ausize[IDX_ROOTFS]
```

To write upgrade images in the USB flash drive to the flash memory, perform the following steps:

Step 1 Load the upgrade images in the USB flash drive to the board memory using the file system.

Step 2 Write the images from the memory to the SPI flash.

----End

3.1.1.2 Check Information

The first several bytes of each image to be upgraded must contain the information such as the magic number, cyclic redundancy check (CRC), file type, and file size.

```
#define IH_MAGIC   0x27051956 /*Image magic number*/
#define IH_NMLEN   32 /* Image name length*/
typedef struct image_header {
uint32_t   ih_magic;   /* Image header magic number   */
uint32_t   ih_hcrc;    /* Image header CRC checksum   */
uint32_t   ih_time;    /* Image creation timestamp */
uint32_t    ih_size;     /* Image data size    */
uint32_t    ih_load;   /* Data load address    */
uint32_t    ih_ep;   /* Entry point address    */
uint32_t    ih_dcrc;       /* Image data CRC checksum */
uint8_t ih_os;  /* Operating system*/
uint8_t ih_arch;     /* CPU architecture       */
uint8_t ih_type;    /* Image type   */
uint8_t ih_comp;       /* Compression type     */
uint8_t ih_name[IH_NMLEN];  /* Image name   */
} image_header_t;
```

The information is written by running the **mkimage** command. If the information does not exist or some items mismatch, images are not updated.

3.1.2 Functions

3.1.2.1 Implementation of External Functions

The following are the functions related to the FAT file system:

```
extern int fat_register_device(block_dev_desc_t *, int);
extern int file_fat_detectfs(void)
extern long file_fat_read(const char *, void *, unsigned long);
```

The preceding functions are used for registering FAT devices and reading files by file name.

```
extern block_dev_desc_t *get_dev (char*, int);
```

The function is used for operating the block device. The first partition is selected by default in the automatic upgrade process.

3.1.2.2 Implementation of Internal Functions

The following four major functions are implemented during the automatic upgrade progress. Each function performs CRC. If CRC for any function fails, an error is returned, and the upgrade is aborted.

- int au_check_cksum_valid(int idx, long nbytes)

  The function is used to obtain the size of the file to be upgraded and compare the file size with that in the **image_header.h** file. If the sizes mismatch, an error is returned, and the upgrade is aborted.

- int au_check_header_valid(int idx, long nbytes)
  - This function is used to check the image_header of an image. If the image does not pass the check, the upgrade is aborted.
  - If the image passes the check, the function can be used to check the image version by comparing the presentation time stamps (ih_time) of the new and old images. If the creation time of the image is earlier than the time recorded, the image is not downloaded to the flash memory. The image that contains required information and passes the check is written to the flash memory. Then, the PTS of the image is saved in the U-boot environment variables for the next comparison.

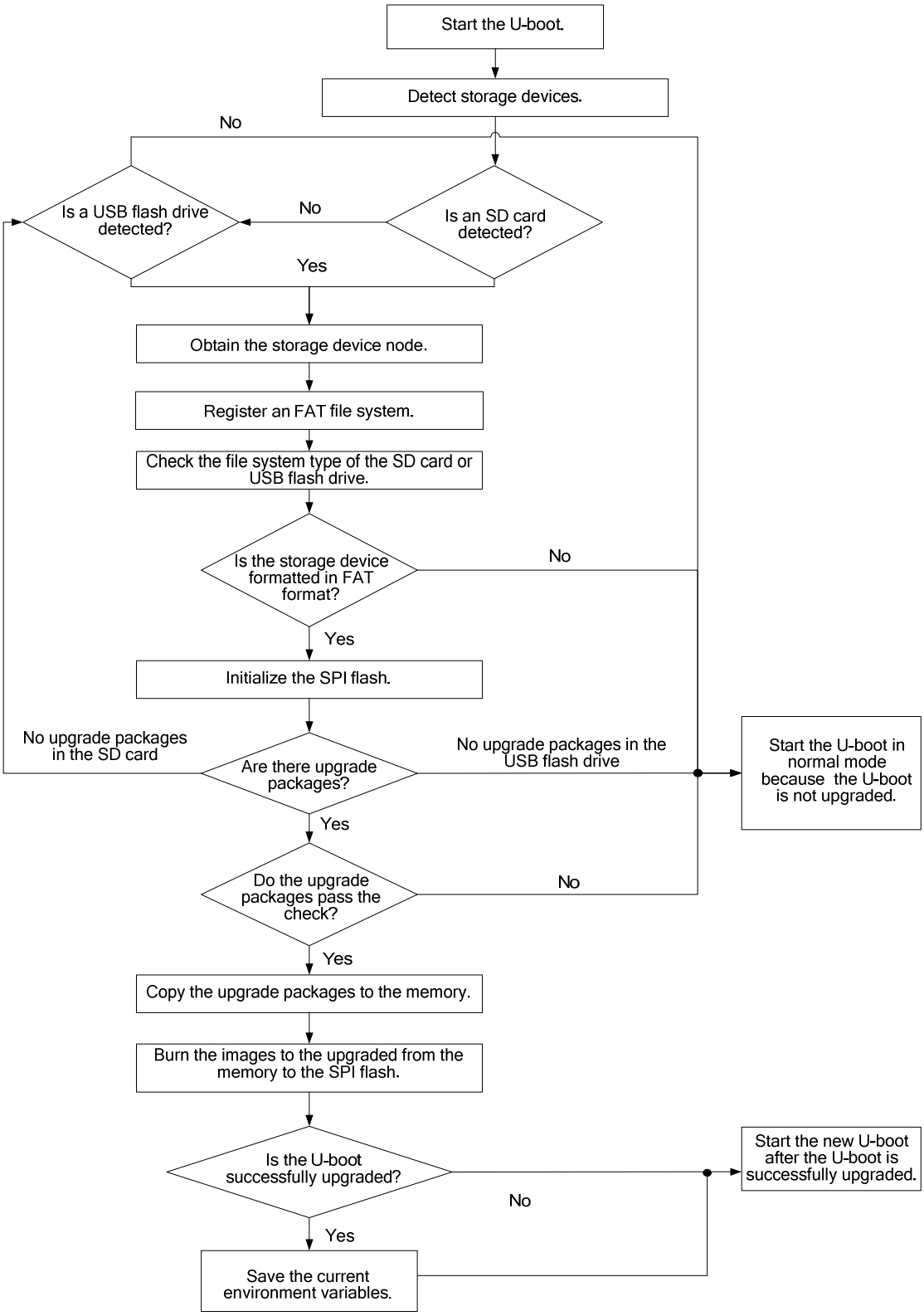
- int au_do_update(int idx, long sz)

  This function is used to write data to the flash memory. After an image to be upgraded passes all checks, the image is written to the flash memory using this function.

- int do_auto_update(void)

  This function is the main function of the automatic upgrade process, and is called by external files. Figure 3-1 shows the workflow of int do_auto_update(void).

Figure 3-1 Workflow of int do_auto_update(void)



For details about implementation, see **product/hiupdate/auto_update.c** under the U-boot.