HiLoader

Development Guide

Issue	**07**
Date	**2016-03-31**

Trademarks and Permissions

Notice

The purchased products, services and features are stipulated by the contract made between HiSilicon and the customer. All or part of the products, services and features described in this document may not be within the purchase scope or the usage scope. Unless otherwise specified in the contract, all statements, information, and recommendations in this document are provided "AS IS" without warranties, guarantees or representations of any kind, either express or implied.

The information in this document is subject to change without notice. Every effort has been made in the preparation of this document to ensure accuracy of the contents, but all statements, information, and recommendations in this document do not constitute a warranty of any kind, express or implied.

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 architecture of the loader of high-definition (HD) chips and loader-based service development.

Related Versions

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

Product Name	Version
Hi3716C	V2*XX*
Hi3719C	V1*XX*
Hi3719M	V1*XX*
Hi3718C	V2*XX*
Hi3718M	V3*XX*
Hi3716M	V4*XX*
Hi3796C	V1*XX*
Hi3798C	V1*XX*
Hi3796M	V1*XX*
Hi3798M	V1*XX*
Hi3798C	V2*XX*

NOTE

XX indicates the chip series number.

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 07 (2016-03-31)

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

The loader is reconstructed.

Issue 06 (2015-12-09)

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

Hi3798M V100, Hi3796M V100, Hi3798C V100, Hi3796C V100, and Hi3798C V200 are supported.

Issue 05 (2015-07-20)

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

Chapter 1 Overview

Section 1.6 is modified.

Chapter 2 Loader Development

Section 2.6.4 is added.

Issue 04 (2015-05-26)

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

Chapter 4 is added.

Issue 03 (2015-01-26)

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

Chapter 2 Loader Development

Section 2.2.1 is modified.

Issue 02 (2014-08-30)

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

Chapter 1 Overview

Section 1.6 is modified.

Chapter 2 Loader Development

Section 2.6.2 is modified.

Issue 01 (2013-12-25)

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

Chapter 2 Loader Development

Section 2.3 is modified.

Modifications are made to support Hi3716M V400.

Issue 00B01 (2013-08-30)

This issue is the first draft release.

Contents

Figures

Tables

1 Overview

1.1 Introduction

This section describes the important concepts, overall workflow, code structure, and supported upgrade modes and protocols of the loader.

1.2 Important Concepts

- Loader upgrade program

  The loader upgrade program is the main part of the loader, which is used to download data, parse protocols, and store data to the memory. The loader is divided into apploader and bootloader based on the running environment of the loader upgrade program.

- Apploader

  The loader upgrade program runs in the kernel environment. After being tailored, the kernel and rootfs as well as the loader upgrade program and its dependent drivers are compiled as the initramfs image, which is separately stored in the loader partition.

- Bootloader

  The loader upgrade program runs in the boot environment. The boot and loader upgrade program as well as its dependent drivers are compiled as the bootloader image, which is stored in the boot partition.

- Large system

  Default system images (including the images of the kernel and file system) generated in the SDK are the large system, which is relative to the small system such as the apploader.

1.3 Code Structure

The loader consists of the upgrade detection program, loader upgrade program, and loader APIs. Table 1-1 shows the functions of the three parts and the directories for storing the bootloader and apploader.

Table 1-1 Loader

Item	Function	Bootloader Directory	Apploader Directory
Upgrade detection program	Detects whether upgrade is required, and starts the loader upgrade program if required.	source/boot/product/ loader/schedule/	source/boot/product/ loader/schedule/
Loader upgrade program	• Downloads upgrade data, parses data based on protocols, and stores parsed data to the storage device. • Sets the loader parameters by using the KEYLED and IR. • Displays the upgrade progress information, error information, and configuration information on the output device (OSD).	source/boot/product/ loader/app/	source/component/ loader/app/
Loader APIs	Allows applications on Linux to read or write to upgrade parameters.	source/component/ loader/api/	source/component/ loader/api/

1.4 Overall Workflow

The only difference between the bootloader workflow and apploader workflow is the way that the loader is started. The bootloader directly calls the entrance API of the loader upgrade program, whereas the apploader reads the initramfs image to the DDR from the loader partition, starts the initramfs, and then runs the loader upgrade program application.

Figure 1-1 shows the workflow of the loader upgrade solution.

Figure 1-1 Workflow of the loader upgrade solution

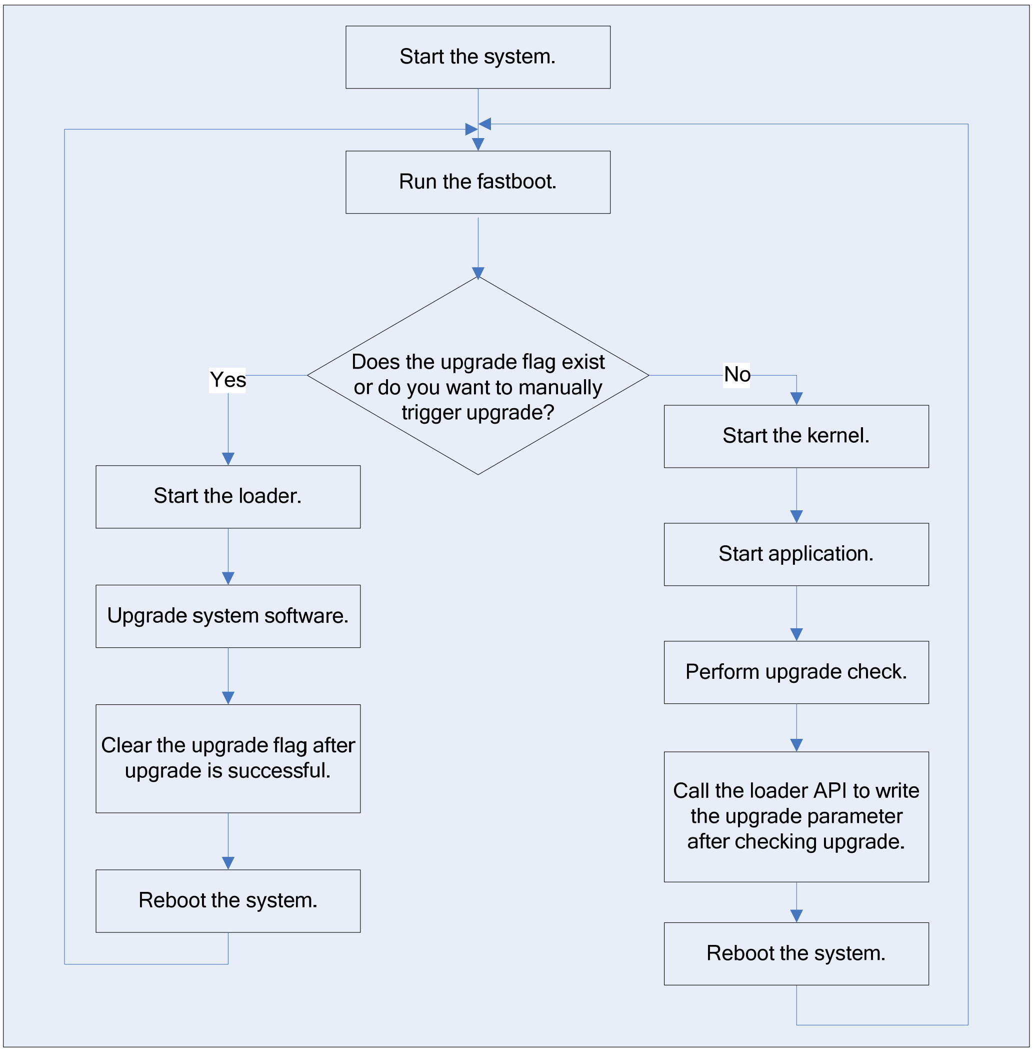
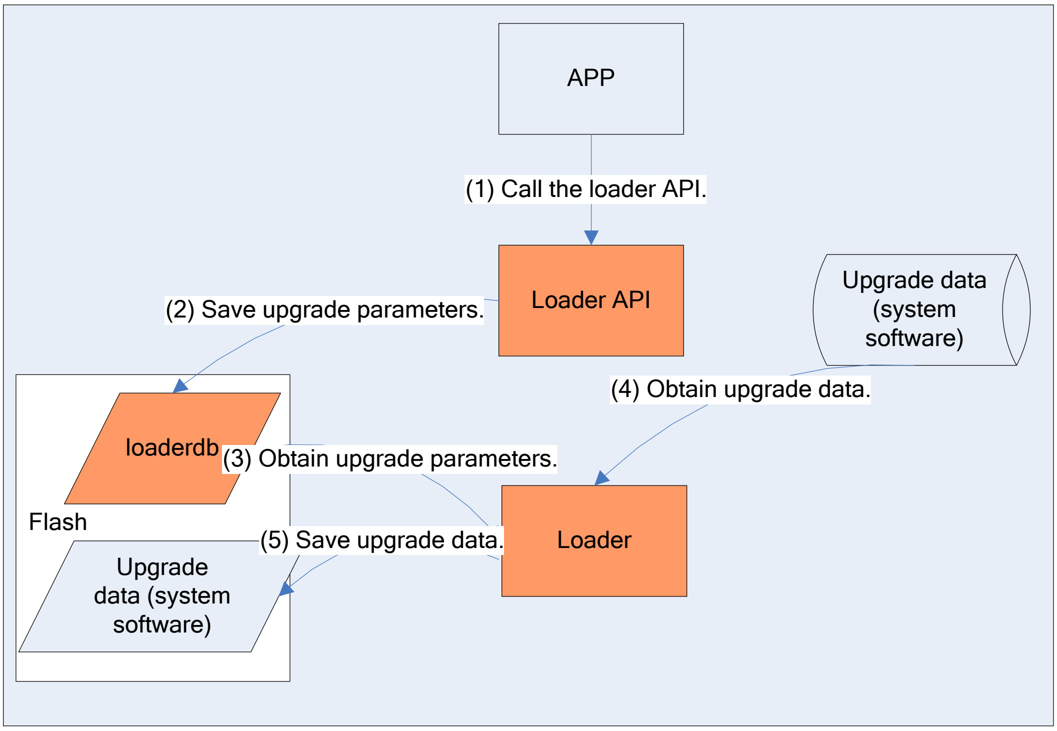

Start the system.
Run the fastboot.
Does the upgrade flag exist or do you want to manually trigger upgrade?
Yes
No
Start the loader.
Upgrade system software.
Clear the upgrade flag after upgrade is successful.
Reboot the system.
Start the kernel.
Start application.
Perform upgrade check.
Call the loader API to write the upgrade parameter after checking upgrade.
Reboot the system.

Figure 1-2 shows the simple data flowchart.

Figure 1-2 Data flowchart


APP
(1) Call the loader API.
Loader API
(2) Save upgrade parameters.
Upgrade data (system software)
(4) Obtain upgrade data.
loaderdb
(3) Obtain upgrade parameters.
Flash
(5) Save upgrade data.
Loader
Upgrade data (system software)

1.5 Supported Protocols and Upgrade Modes

The bootloader supports the following protocols:

- System Software Updates (SSU) protocol

  A system software upgrade service protocol defined based on the European digital video broadcast (DVB) standard. This protocol applies to OTA upgrade.
- HISI OTA protocol

  A protocol defined by HiSilicon. This protocol applies to OTA upgrade.
- HISI FILE protocol

  A protocol defined by HiSilicon. This protocol is also called USB/IP protocol and applies to USB and IP upgrade.

The apploader supports the OTA, USB, and IP upgrade, and the bootloader supports the OTA and USB upgrade. Table 1-2 describes features of each upgrade mode.

Table 1-2 Upgrade mode features

Upgrade Mode	Triggering Mode	Features
OTA	Triggered by the application/Triggered manually by pressing the KEYLED key	Supports the following protocols: • HISI OTA • SSU
USB	Triggered by the application/Triggered manually by pressing the KEYLED key	• Supports the HISI FILE protocol. • Supports the upgrade by using the USB device that runs the FAT32/NTFS/EXT2/EXT3/EXT4 file system. • Allows the application to trigger the upgrade by using the upgrade file of the specified name in a specified directory. • Supports upgrade triggered by pressing the KEYLED key to run the upgrade file **usb_update.bin** in the root directory.
IP	Triggered by the application	Supports the following protocols: • TFTP • FTP • HTTP

1.6 System Partitions

The deviceinfo partition stores private STB configuration information, such as information about the vendor and hardware version. The loaderdb partition stores parameter configuration information required for the loader upgrade. You can reserve a loaderdbbak partition to back up the loaderdb partition in case that the upgrade parameters in the loaderdb partition are damaged. The softwareinfo partition stores information of the software version, including software version, partition versions, and anti-rollback versions.

The names of the deviceinfo, loaderdb, softwareinfo, and loaderdbbak partitions are fixed, and the size of each partition is a flash block. You can change the name or size of these partitions only in the corresponding code.

Table 1-3 shows the system partitions of the bootloader solution. You can define system partitions as required.

Table 1-3 System partitions of the bootloader solution

bootloader	bootargs	deviceinfo	softwareinfo	loaderdb	loaderdbbak	baseparam	logo	kernel	rootfs

The apploader requires a loader partition. If the loader needs to be upgraded, a backup partition LoaderBak needs to be reserved to restore the loader partition if the loader partition is damaged during the upgrade process.

Table 1-4 shows the system partitions of the apploader solution. You can define system partitions as required.

Table 1-4 System partitions of the apploader solution

boot	bootargs	device info	software info	loaderdb	loaderdb bak	loader	loader bak	baseparam	logo	kernel	rootfs

2 Loader Development

2.1 Overview

This section describes how to compile, run, and debug the loader program.

2.2 Configuring and Compiling the Loader Program

2.2.1 Bootloader

To configure and compile the bootloader, perform the following steps:

Step 1 Replace **SDK/cfg.mak** with the corresponding configuration file in **SDK/configs** based on the target board type.

Step 2 Run **make menuconfig** to enter the configuration interface, and select the following configuration options:

```
Base  --->
[*] HiLoader Support  --->
   --- HiLoader Support
       Support Loader Type (BootLoader)  --->
Uboot  --->
[*] Build Compressed Fastboot Image
 -*- Support Usb Drivers
 -*- Support FAT filesystem
Build Product Code in Fastboot  --->
   [*]  BootLoader Config  --->
       --- BootLoader Config
          [*]  USB Upgrade Support (NEW)
             Protocol Type (Hisi File Protocol)  --->
          [*]  OTA Upgrade Support (NEW)
              Tuner Type (Cable)  --->
              Protocol Type (Hisi OTA Protocol)  --->
```

Step 3 Run **make distclean && make build**. The following files are generated:

- Bootloader image (**SDK/pub/chip type/image/fastboot-burn.bin**)
- Linux loader libraries (**SDK/source/component/loader/api/libhiloader.a** and **SDK/source/component/loader/api/libhiloader.so**)
- Linux loader sample (**SDK/sample/loader/sample_loader**)

----End

2.2.2 Apploader

To configure and compile the apploader, perform the following steps:

Step 1 Replace **SDK/cfg.mak** with the corresponding configuration file in **SDK/configs** based on the target board type.

Step 2 Run **make menuconfig** to enter the configuration interface, and select the following configuration options:

```
Base  --->
    [*] HiLoader Support  --->
          --- HiLoader Support
              Support Loader Type (AppLoader)  --->
```

Step 3 Run **make distclean && make build**. The following files are generated:

- Boot image (**SDK/pub/chip type/image/fastboot-burn.bin**)
- Linux loader libraries (**SDK/source/component/loader/api/libhiloader.a** and **SDK/source/component/loader/api/libhiloader.so**)
- Linux loader sample (**SDK/sample/loader/sample_loader**)

Step 4 Replace **SDK/cfg.mak** with the private apploader configuration file **cfg.mak**, run **make menuconfig** to enter the configuration interface, and select the following configuration options:

```
Component  --->
[*] AppLoader Config  --->
    --- AppLoader Config
       OSD Language Type (English)  --->
       [*]  USB Upgrade Support (NEW)
          Protocol Type (Hisi File Protocol)  --->
       [*]  IP Upgrade Support (NEW)
          Protocol Type (Hisi File Protocol)  --->
       [*]  OTA Upgrade Support (NEW)
          Tuner Type (Cable)  --->
       (*) Tuner Port Index (0,3) (NEW)
          Protocol Type (Hisi OTA Protocol)  --->
```

Step 5 Run **make distclean && make build**. The apploader image (**SDK/pub/chip type/image/apploader.bin**) is generated.

----End

CAUTION

To support the apploader, you need to use the **fastboot-burn.bin** file compiled in step 3, but not the one compiled by using the apploader configuration file.

2.3 Using the HiLoader_Tool

For details about how to use the package tool, see the *HiLoader User Guide***.**

2.4 Calling Loader APIs

2.4.1 Overview

The loader APIs provide applications with APIs for triggering upgrade, setting and querying upgrade parameters, and querying the software version and device information. These APIs are independent of each other and can be called in any sequence. This section describes how to use APIs to configure upgrade parameters and trigger upgrading by using instances.

2.4.2 Triggering USB Upgrade

The following is the reference code:

```
HI_VOID USB_Upgrade(HI_VOID)
{
    HI_CHAR *pcFileName = "/usb/test/usb_update.bin";
    HI_LOADER_PARAM_S stLoaderParam;

    memset(&stLoaderParam, 0x00, sizeof(stLoaderParam));
    HI_LOADER_GetParameter(&stLoaderParam);

    stLoaderParam.enUpgradeType = HI_LOADER_UPGRADE_TYPE_USB;
    stLoaderParam.enUpgradeMode = HI_LOADER_UPGRADE_MODE_BASIC;
    strncpy((char*)stLoaderParam.unParam.stUSBParam.as8FileName,
            pcFileName, HI_LOADER_FILENAME_LEN - 1);

    HI_LOADER_SetParameter(&stLoaderParam);
    return;
}
```

2.4.3 Triggering OTA Upgrade Through Satellite Signals

The following is the reference code:

```
HI_VOID OTA_Satelite_Upgrade(HI_VOID)
{
    HI_LOADER_PARAM_S stLoaderParam;
    HI_LOADER_PARAM_OTA_S *pstOTAParam = HI_NULL_PTR;
    HI_LOADER_PARAM_SAT_S *pstSatParam = HI_NULL_PTR;

    memset(&stLoaderParam, 0x00, sizeof(stLoaderParam));
    HI_LOADER_GetParameter(&stLoaderParam);

    HI_LOADER_GetParameter(&stLoaderParam);
    stLoaderParam.enUpgradeType = HI_LOADER_UPGRADE_TYPE_OTA;
    stLoaderParam.enUpgradeMode = HI_LOADER_UPGRADE_MODE_BASIC;

    pstOTAParam = &stLoaderParam.unParam.stOTAParam;
    pstOTAParam->enSigType = HI_UNF_TUNER_SIG_TYPE_SAT;
    pstOTAParam->u32TunerID = 0; /**< not enabled */
    pstOTAParam->u32Pid = 7000;

    /** Tuner Parameter Config */
    pstSatParam = &pstOTAParam->unParam.stSat;
    pstSatParam->u32IsiID = 0; /**< not enabled */
    pstSatParam->stConnectParam.u32Freq = 3840000;
    pstSatParam->stConnectParam.u32SymbolRate = 27500000;
    pstSatParam->stConnectParam.enPolar = HI_UNF_TUNER_FE_POLARIZATION_H;
    pstSatParam->stConnectParam.u32ScrambleValue = 0;
    pstSatParam->enLNBPower = HI_UNF_TUNER_FE_LNB_POWER_OFF;
    pstOTAParam->unParam.stSat.stLNBConfig.u32LowLO = 5105;
    pstOTAParam->unParam.stSat.stLNBConfig.u32HighLO = 5105;

    HI_LOADER_SetParameter(&stLoaderParam);
    return;
}
```

2.4.4 Triggering OTA Upgrade Through Cable Signals

The following is the reference code:

```
HI_VOID OTA_Calbe_Upgrade(HI_VOID)
{
    HI_LOADER_PARAM_S stLoaderParam;
    HI_LOADER_PARAM_OTA_S *pstOTAParam = HI_NULL_PTR;
    HI_LOADER_PARAM_CAB_S *pstCabParam = HI_NULL_PTR;

    memset(&stLoaderParam, 0x00, sizeof(stLoaderParam));
    HI_LOADER_GetParameter(&stLoaderParam);
    stLoaderParam.enUpgradeType = HI_LOADER_UPGRADE_TYPE_OTA;
    stLoaderParam.enUpgradeMode = HI_LOADER_UPGRADE_MODE_BASIC;

    /** Tuner Parameter Config */
    pstOTAParam = &stLoaderParam.unParam.stOTAParam;
    pstOTAParam->enSigType = HI_UNF_TUNER_SIG_TYPE_CAB;
    pstOTAParam->u32TunerID = 0; /**< not enabled */
    pstOTAParam->u32Pid = 7000;

    pstCabParam = &pstOTAParam->unParam.stCab;
    pstCabParam->stConnectParam.u32Freq = 443000;
    pstCabParam->stConnectParam.u32SymbolRate = 6875000;
    pstCabParam->stConnectParam.enModType = HI_UNF_MOD_TYPE_QAM_64;
    pstCabParam->stConnectParam.bReverse = HI_FALSE;

    HI_LOADER_SetParameter(&stLoaderParam);
    return;
}
```

2.4.5 Triggering IP Upgrade

```
HI_VOID OTA_Terestrial_Upgrade(HI_VOID)
{
    HI_LOADER_PARAM_S stLoaderParam;
    HI_LOADER_PARAM_OTA_S *pstOTAParam = HI_NULL_PTR;
    HI_LOADER_PARAM_TER_S *pstTerParam = HI_NULL_PTR;

    memset(&stLoaderParam, 0, sizeof(stLoaderParam));
    HI_LOADER_GetParameter(&stLoaderParam);

    stLoaderParam.enUpgradeType = HI_LOADER_UPGRADE_TYPE_OTA;
    stLoaderParam.enUpgradeMode = HI_LOADER_UPGRADE_MODE_BASIC;

    /** Tuner Parameter Config */
    pstOTAParam = &stLoaderParam.unParam.stOTAParam;
    pstOTAParam->enSigType = HI_UNF_TUNER_SIG_TYPE_DVB_T;
    pstOTAParam->u32TunerID = 0; /**< not enabled */
    pstOTAParam->u32Pid = 7000;

    pstTerParam = &pstOTAParam->unParam.stTer;
    pstTerParam->u32PLPId = 0; /**< not enabled */
    pstTerParam->stConnectParam.u32Freq = 443000;
    pstTerParam->stConnectParam.u32BandWidth = 8000;
    pstTerParam->stConnectParam.enModType = HI_UNF_MOD_TYPE_QAM_64;
    pstTerParam->stConnectParam.bReverse = HI_FALSE;
    pstTerParam->stConnectParam.enChannelMode
       = HI_UNF_TUNER_TER_MODE_BASE;
    pstTerParam->stConnectParam.enDVBTPrio
       = HI_UNF_TUNER_TS_PRIORITY_NONE;

    HI_LOADER_SetParameter(&stLoaderParam);
    return;
}
```

2.4.6 Triggering IP Upgrade

```
HI_VOID IP_Upgrade(HI_VOID)
{
    HI_LOADER_PARAM_S stLoaderParam;

    memset(&stLoaderParam, 0x00, sizeof(stLoaderParam));
    HI_LOADER_GetParameter(&stLoaderParam);

    stLoaderParam.enUpgradeType = HI_LOADER_UPGRADE_TYPE_IP;
    stLoaderParam.enUpgradeMode = HI_LOADER_UPGRADE_MODE_BASIC;

    /** IP Parameter Config */
    stLoaderParam.unParam.stIPParam.enIPCfgType = HI_LOADER_IPCFG_STATIC;
    stLoaderParam.unParam.stIPParam.enProtType = HI_LOADER_IPPROT_HTTP;
    stLoaderParam.unParam.stIPParam.ipServer = inet_addr("10.67.217.28");
    stLoaderParam.unParam.stIPParam.ipServerPort = 8080;
    strncpy((char*)stLoaderParam.unParam.stIPParam.as8FileName,
          "ip_update.bin", HI_LOADER_FILENAME_LEN - 1);

    HI_LOADER_SetParameter(&stLoaderParam);
    return;
}
```

2.5 Application Scenarios

After the loader program is compiled, burn the loader image (bootloader or apploader) to the corresponding flash partition, and then run the upgrade program.

2.5.1 OTA Upgrade Triggered by the Application

Scenario

The application writes the OTA upgrade parameters (including the frequency parameter and PID) and OTA upgrade type flag to the loaderdb partition and then restarts the system to implement the OTA upgrade.

Procedure

Perform the following steps:

Step 1 Compile the loader image and burn it to the corresponding partition.

Step 2 Package the upgrade images as a TS upgrade file by using the HiLoader_Tool.

Step 3 Configure the front end upgrade server and play the TS upgrade file.

Step 4 Run the loader sample or an application to trigger OTA upgrade.

Step 5 Restart the system. Then the system automatically starts the loader program for upgrade.

Step 6 Wait until the upgrade is complete.

----End

Notes

None

Sample

For details about the reference code, see sections 2.4.3 "Triggering OTA Upgrade Through Satellite Signals" and 2.4.4 "Triggering OTA Upgrade Through Cable Signals."

2.5.2 Manually Triggering the OTA Upgrade

Scenario

Press the forcible upgrade key combination (for example, **Menu+OK**) on the front panel when the board restarts to enter the loader forcible upgrade mode. If the USB device that stores the upgrade file is properly identified by the loader system, the USB upgrade is forcibly implemented. Otherwise, the OTA upgrade is implemented, and you can forcibly enter the OTA upgrade mode without connecting the USB device or storing valid upgrade files in the USB device.

Procedure

Perform the following steps:

Step 1 Compile the loader image and burn it to the corresponding partition.

Step 2 Package the upgrade images as a TS upgrade file by using the HiLoader_Tool.

Step 3 Configure the front end upgrade server and play the TS upgrade file.

Step 4 Ensure that no valid USB upgrade file is in the USB device, or remove the USB device.

Step 5 Restart the device and press the forcible upgrade key combination on the front panel. If the USB upgrade file fails to be detected, enter the OTA upgrade mode.

Step 6 Set upgrade parameters based on the message displayed in the OSD window. Press **Start** for upgrade.

----End

Notes

None

Sample

None

2.5.3 USB Upgrade Triggered by the Application

Scenario

The application writes the USB upgrade type flag and the directory (including the file name) for storing the USB upgrade file to the loaderdb partition and then restarts the system to implement the USB upgrade.

Procedure

Perform the following steps:

Step 1 Compile the loader image and burn it to the corresponding partition.

Step 2 Package the upgrade images as a USB upgrade file (for example, **usb_update.bin**) by using the HiLoader_Tool.

Step 3 Copy **usb_update.bin** to a specific directory such as **/usb** of the USB device, and then insert the USB device into the USB port of the board. This directory must be the one configured when the conditions for triggering USB upgrade are set.

Step 4 Run the loader sample or an application to trigger USB upgrade, and set the USB upgrade path to the one (for example **/usb/usb_update.bin**) where the USB upgrade file is located.

Step 5 Restart the system. Then the system automatically starts the loader program for upgrade.

Step 6 Wait until the upgrade is complete.

----End

Notes

You can specify the upgrade file name and directory for the USB upgrade triggered by the application.

Sample

For details about the reference code, see section 2.4.3 "Triggering OTA Upgrade Through Satellite Signals."

2.5.4 Manually Triggering the USB Upgrade

Scenario

Press the forcible upgrade key combination (for example, **Menu+OK**) on the front panel when the board restarts to enter the loader forcible upgrade mode. If the USB device that stores the upgrade file is properly identified by the loader system, the USB upgrade is forcibly implemented. Otherwise, the OTA upgrade is implemented, and you can forcibly enter the USB upgrade mode by connecting the USB device that has a valid upgrade file to the board before performing forcible upgrade.

Procedure

Perform the following steps:

Step 1 Compile the loader image and burn it to the corresponding partition.

Step 2 Package the upgrade images as a USB upgrade file (for example, **usb_update.bin**) by using the HiLoader_Tool.

Step 3 Copy **usb_update.bin** to the root directory of the USB device.

Step 4 Restart the device and press the forcible upgrade key combination on the front panel. Then the system starts the loader program for upgrade.

Step 5 Detect the USB upgrade file automatically. If the upgrade file is detected, enter the USB forcible upgrade mode.

----End

Notes

N/A.

2.5.5 IP Upgrade Triggered by the Application

Scenario

The application writes the IP upgrade parameters (including the transmission protocol, server IP address, upgrade file name, local IP address, and gateway) to the loaderdb partition and then restarts the system to implement IP upgrade.

Procedure

Perform the following steps:

Step 1 Compile the loader image and burn it to the corresponding partition.

Step 2 Package the upgrade images as a USB upgrade file by using the HiLoader_Tool.

Step 3 Configure the front end upgrade server and copy the packaged USB upgrade file to the upgrade server.

Step 4 Run the loader sample to run the application to configure IP upgrade parameters.

Step 5 Restart the system. Then the system automatically starts the loader program for upgrade.

----End

Notes

Only the apploader supports the IP upgrade.

Sample

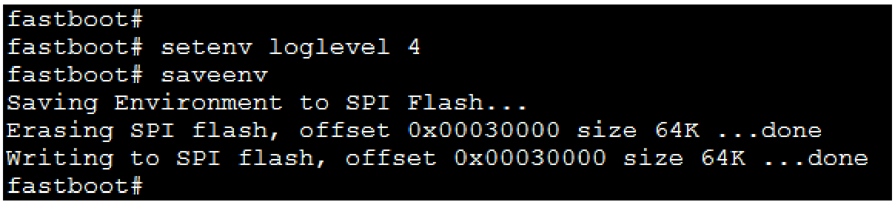
For details about the reference code, see section 2.4.5 "Triggering IP Upgrade."

2.6 Debugging the Loader Program

2.6.1 Debugging Logs

The bootloader solution allows you to change the value of **loglevel** in the command-line interface (CLI) of the fastboot to specify the log information to be displayed. **loglevel** ranges from 0 to 4. If **loglevel** is **0**, no log information is displayed; if **loglevel** is **4**, all log information is displayed. See Figure 2-1.

Figure 2-1 Setting loglevel



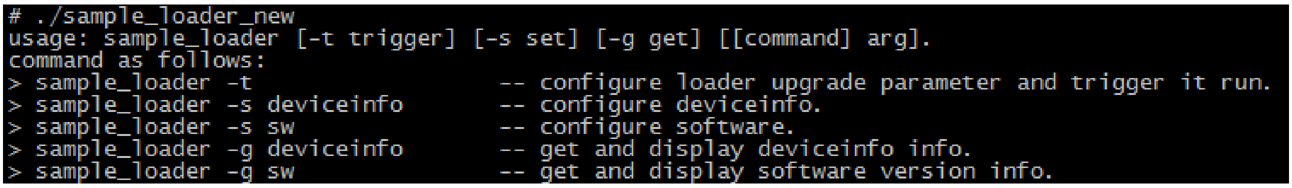
```
fastboot#
fastboot# setenv loglevel 4
fastboot# saveenv
Saving Environment to SPI Flash...
Erasing SPI flash, offset 0x00030000 size 64K ...done
Writing to SPI flash, offset 0x00030000 size 64K ...done
fastboot#
```

In the apploader solution, you can set **loglevel** to **4** to output debugging logs.

2.6.2 Triggering Upgrade by Simulating the Application

The loader program needs to be debugged during loader-based development. If the system does not integrate applications, you can run the loader sample to debug the loader program. See Figure 2-2.

Figure 2-2 Loader sample



```
# ./sample_loader_new
usage: sample_loader [-t trigger] [-s set] [-g get] [[command] arg].
command as follows:
> sample_loader -t              -- configure loader upgrade parameter and trigger it run.
> sample_loader -s deviceinfo   -- configure deviceinfo.
> sample_loader -s sw           -- configure software.
> sample_loader -g deviceinfo   -- get and display deviceinfo info.
> sample_loader -g sw           -- get and display software version info.
```

For details about how to use the sample_loader, see **Readme** in **SDK/sample/loader**.

2.6.3 Forcible Upgrade

Press **Menu+OK** on the front panel when the board starts to enter the loader forcible upgrade mode. If the USB device that stores the upgrade file is properly identified by the loader system, the USB upgrade is forcibly implemented. Otherwise, the OTA upgrade is used, and you can set OTA parameters by using the IR or front panel keys to implement the OTA upgrade.

2.6.4 Compiling and Debugging libhiloader.so and the Loader Program Quickly

In the preceding sections, the SDK for compiling the kernel and file system is distinguished from that for compiling **apploader.bin**. The previous one is called a common SDK, and the latter one is called a loader SDK.

To compile and debug **libhiloader.so** quickly, perform the following steps in the root directory of the common SDK:

Step 1 Clean the compilation path of the loader library by running the following command:

```
make component_clean mod=loader
```

Step 2 Run the following command to generate **libhiloader.a** and **libhiloader.so** in **pub/lib/static** and **pub/lib/share** respectively:

```
make component_install mod=loader
```

----End

To compile and debug the loader program quickly, perform the following steps in the root directory of the loader SDK:

Step 1 Clean the loader compilation path by running the following command:

```
make component_clean mod=loader
```

Step 2 Run the following command to obtain the compiled loader program in the **source/component/loader/app/release** directory:

```
make component_install mod=loader
```

To quickly generate **apploader.bin**, run the following command in the root directory of the SDK:

```
make loader_rebuild
```

The **apploader.bin** image is generated under the **pub/image** directory.

Step 3 Enter the **apploader.bin** small system by using sample_loader, press **CTRL+C** to enter the command-line interface, and replace the original loader program in the **/home** directory with the newly compiled loader program by mounting the program to the NFS file system or running the **tftp** command. Then you can debug the loader program by following instructions in section 2.6.1 "Debugging Logs" and using other methods for debugging Linux applications.

----End

3 Porting the Loader

3.1 Overview

This section describes how to port and develop the loader based on the HiSilicon loader solution to meet customer requirements.

3.2 Configuring the Tuner

Tuner-related attributes vary according to boards. For details about how to configure the tuner, see **download_ota.c**.

3.3 Configuring the Remote Control

Major APIs are as follows:

- uiIRInit ()

  Initializes the IR device.
- uiIRDeInit()

  Deinitializes the IR device.
- uiIRGetValue ()

  Obtains the input key code of the IR device. This key code is corresponding to UI_KEYVALUE_E which can be modified to switch between various IR devices.

3.4 Configuring the KEYLED

Major APIs are as follows:

- uiKEYLEDInit ()

  Initializes the KEYLED device. The KEYLED model can be configured by modifying the macro definition UI_D_KEYLED_TYPE.
- uiKEYLEDDeInit ()

  Deinitializes the KEYLED device.

- uiKEYLEDKeyConvert()

  Converts the KEYLED key code into IR key code. The KEYLED key code is visible internally. You can disclose the KEYLED key code to the upper-layer applications by extending the UI_KEYVALUE_E enumeration.

- uiKEYLEDGetValue ()

  Obtains the input key code of the IR device.

- uiKEYLEDDisplay()

  Displays character strings on the KEYLED device. Character conversion codes for various device models are configured by using the s_au8DigitalCode array.

3.5 Checking the Target System Version

loaderCheckVersionMatch is used to compare the current system version with the upgrade stream system version. If the upgrade is allowed only when certain conditions are met, the implementation of loaderCheckVersionMatch needs to be modified. Then upgrade is performed only when HI_SUCCESS is returned. In the bootloader solution, the current system version and upgrade stream system version are not compared. That is, the system is upgraded when an upgrade file is detected.

3.6 Processing After the Upgrade Is Complete

After the upgrade is complete (whether the upgrade is successful or fails), call loaderUpgradeDone to determine whether the system automatically restarts or waits for users to turn off the power supply to restart the system. In the bootloader solution, the system automatically starts.

3.7 Switching the GUI Language

AppLoader switches the language of the GUI by running the **make menuconfig** command.

```
Component  --->
[*] AppLoader Config  --->
    --- AppLoader Config
       OSD Language Type (English)  --->
```

In the bootloader solution, the GUI language is English. If you want to switch the GUI language, extend character output APIs.

3.8 Changing the Mode of Manually Triggering Upgrade

Modify the implementation of Loader_CheckManuForceUpgrade to change the mode of manually triggering upgrade.

3.9 Adding Download Modes

A new download mode can be added by implementing the following functions. *XXX* in the following function names indicates the name of the new download mode. The following functions are called by the protocol type-related functions. For details about how to call the following functions, see the protocol types supported by the loader.

- DOWNLOAD_*XXX*_Init

  Initializes the download mode and performs the operations such as allocating the memory and connecting the board to the upgrade data source (front end or server).
- DOWNLOAD_*XXX*_getdata

  Obtains data from the upgrade data source.
- DOWNLOAD_*XXX*_DeInit

  Deinitializes the download mode and performs the operations such as releasing the memory and disconnecting the board from the upgrade data source (front end or server).
- DOWNLOAD_USB_GetFileSize

  Obtains the size of the upgrade file.

3.10 Adding Protocol Types

A new protocol type can be added by implementing the following functions. *XXX* in the following function names indicates the name of the new protocol type. The following functions are called by the main process of the loader. The adaptation code of the new protocol needs to be added to **protocol.c**.

- PROT_XXX_Init

  Initializes the system and performs the operations such as allocating the memory and calling the LOADER_DOWNLOAD_*XXX*_Init to connect the system to the upgrade data source.
- PROT_XXX_DeInit

  Deinitializes the system and performs the operations such as releasing the memory and calling the PROT_*XXX*_DeInit to disconnect the system from the upgrade data source.
- PROT_XXX_GetVersionInfo

  Obtains the version information about the upgrade data from the data source. In this case, PROT_*XXX*_getdata is called.
- PROT_XXX_GetPartitionInfo

  Obtains the information about the image partition that stored the upgrade data from the data source. In this case, PROT_*XXX*_getdata is called.
- PROT_XXX_Process

  Obtains the data of the upgrade image from the data source. In this case, PROT_*XXX*_getdata is called.

3.11 Developing the GUI

For details, see the codes in the **ui** directory in the loader APP program.

- ui_gfx.c

Packs graphics components.

- ui_display.c

  Displays the parameter configuration and the initialization and deinitialization of the unit.

- ui_window.c

  Manages components.

- ui_win_main.c

  Manages upgrade progress.

- ui_win_msgbox.c

  Manages the message box.

- ui_win_setting.c

  Manually upgrades the configuration GUI.

4 Loader Upgrade Protocols

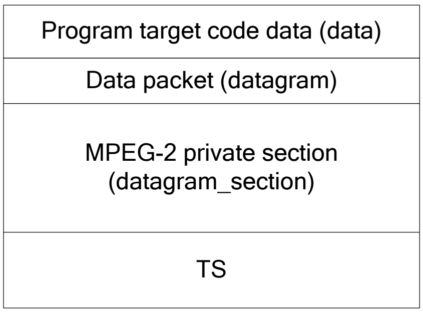
4.1 Overview

This chapter introduces the OTA upgrade stream transfer protocols (HISI OTA and SSU OTA) and USB upgrade file protocol. The IP upgrade file protocol is the same as the USB upgrade file protocol.

4.1.1 TS Protocol Stack

Figure 4-1 shows the protocol stack for the encapsulation protocol of the TS upgrade data.

Figure 4-1 Protocol stack



The target data (data) is segmented into multiple blocks with the appropriate size, each block is packaged into a data packet (datagram), and then the data packets are combined as private sections that comply with the MPEG-2 standard to form the TS.

4.1.2 TS Structure

The loader packages and encapsulates the upgrade data to obtain the TS that complies with the MPEG-2 standard. For details about the TS format, see the ISO/IEC 13818-1 protocol.

4.2 HISI OTA Upgrade Stream Transfer Protocol

4.2.1 Syntax of the HISI MPEG-2 Private Section

The HISI OTA upgrade protocol uses the private section defined by MPEG-2 as the data carrier. Table 4-1 describes the syntax of the MPEG-2 private section. For details, see the ISO/IEC 13818-1 protocol.

Table 4-1 HISI private section syntax

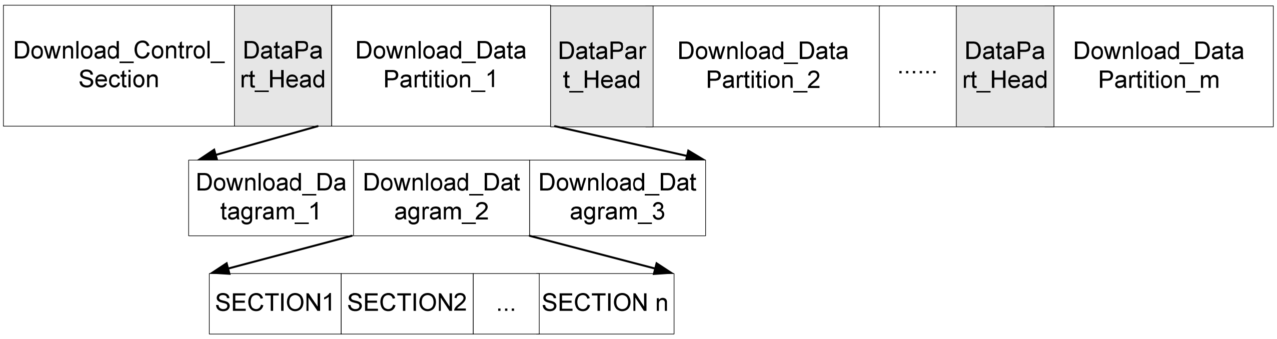
Syntax	Number of Bits	Identifier
private_section()	-	-
{	-	-
table_id	8	uimsbf
section_syntax_indicator	1	bslbf
reserved	3	bslbf
private_section_length	12	uimsbf
if(section_syntax_indicator =='0'){	-	-
for (i = 0; i < N; i++){	-	-
private_data_byte	8	uimsbf
}	-	-
}		
else{	-	-
table_id_extension	16	uimsbf
reserved	2	bslbf
version_number	5	uimsbf
current_next_indicator	1	bslbf
section_number	8	uimsbf
last_section_number	8	uimsbf
for(i=0; i < N; i++){	-	-
private_data_byte	8	bslbf
}	-	-
CRC32	32	rpchof
}		
}	-	-

Note that the **section_syntax_indicator** field is **1**, indicating that there are fields such as **table_id_extension** after the **private_section_length** field, and private data is after the **last_section_number** field.

4.2.2 Syntax of the HISI Downloaded Data Stream Section

Figure 4-2 shows the definition of the HISI downloaded data section.

Figure 4-2 HISI downloaded data stream format



Each partition corresponds to a partition or an application program in the flash memory. Each partition consists of several datagram packets, and contains at most 512 MB data. The HiLoader package tool has split the package, and therefore each partition image can be 4 GB at the maximum. Each datagram packet contains at most 8 KB valid data. It consists of at most eight sections, each of which contains at most 1024 bytes.

The format of the upgrade data package is described as follows:

- One partition corresponds to one software program to be upgraded, one file, or one flash partition.
- A partition consists of several datagram packets.
- A datagram packet consists of one to eight datagram sections (currently the HiLoader supports only one datagram section in a datagram packet, and therefore the upgrade file is 64 MB at the maximum). There are at most 2^{16} **extension_table_id**. Each **extension_table_id** can store one section, and therefore at most 64 MB (64 MB x 8 theoretically) data can be received. Due to this restriction, the HiLoader package tool has split the package internally. Therefore, image size of a single partition can be at most 4 GB when the upgrade package is created.

Table 4-2 describes the data stream segments.

Table 4-2 HISI data stream segments

Table ID	Section Number	Tbl_Ext_ID	Meaning/Usage
0xFE	0x00	0x00	Download_Control_Section
Table ID specified in the download	0x01	0x00	Partition_Control(partition 1)
		0x01	Datagram 1
		0x02	Datagram 2

Table ID	Section Number	Tbl_Ext_ID	Meaning/Usage
control section 0x00–0xFD		0x03	Datagram 3
		...	Datagram ...
		N	Datagram n
	0x02	0x00	Partition_Control (partition 2)
		0x01	Datagram 1
		0x02	Datagram 2
		0x03	Datagram 3
		...	Datagram ...
		N	Datagram n
	y	0x00	Partition_Control (partition y)
		0x01	Datagram 1
		0x02	Datagram 2
		...	Datagram ...
		N	Datagram n

The following sections are in the same TS and have the same PID:

- **table_id=0xF**E, **section_number=0x00**, and **table_id_extension=0x0000** determine that the section is a download control section.
- **download_table_id=0xXX**, **section_number=(1….n)partition_number**, and **table_id_extension=0x0000** determine that the section is the packet header section of a partition data area.
- **download_table_id=0xXX**, **section_number=(1…..n)partition_number**, **table_id_extension=(1….n)datagram_number**, and **last_section_number≤8** indicate the number of datagram sections in each datagram data packet, and **Datagram_current_section_number** in the section indicates the current section in the datagram data packet.

4.2.3 Syntax of Download_Control_Section

Table 4-3 describes the syntax of Download_Control_Section.

Table 4-3 Download_Control_Section syntax

Syntax	Number of Bits	Identifier
Download_Control_Section()	-	-
{	-	-
table_id = 0xfe	8	uimsbf

Syntax	Number of Bits	Identifier
section_syntax_indicator=1	1	bslbf
reserved	3	bslbf
section_length	12	uimsbf
table_id_extension = 0x0000	16	uimsbf
software_version	8	uimsbf
section_number = 0	8	uimsbf
last_section_number	8	uimsbf
for(i=0; i < –N; i++){	-	-
Download_info() }	-	-
}	-	-
CRC32	32	rpchof
}	-	-

The fields in Download_Control_Section are described as follows:

- **table_id**: It is set to **0xFE**.
- **section_syntax_indicator**: It should be set to **1**.
- **section_length**: length (number of bytes) of the section (including the CRC) after this field
- **table_id_extension**: It is set to **0**.
- **software_version**: software version to which the data packet or control information belongs
- **section_number**: It is set to **0**.
- **last_section_number**: serial number of the last valid section

4.2.4 Syntax of Download_Info

Table 4-4 describes the syntax of Download_Info.

Table 4-4 Download_Info syntax

Syntax	Number of Bits	Identifier
Download_Info()	-	-
{	-	-
download_Info_tag = 0xea	8	uimsbf
download_Info_len	8	uimsbf
for (i = 0; i < N; i++)	-	-

Syntax	Number of Bits	Identifier
{	-	-
STB_manufacturerID	32	uimsbf
hardware_version	32	uimsbf
software_version	32	uimsbf
download_table_id	8	uimsbf
key_contorl	8	uimsbf
serial_number_start	32	uimsbf
serial_number_end	32	uimsbf
download_date	32	uimsbf
if (key_control&1) {	-	-
app_version	32	uimsbf
}	-	-
if (key_control&2) {	-	-
kernel_version	32	uimsbf
}	-	-
if (key_control& 4) {	-	-
CA_version	32	uimsbf
}	-	-
if (key_control&8) {	-	-
bootloader_version	32	uimsbf
}	-	-
if (key_control&16) {	-	-
apploader_version	32	uimsbf
}	-	-
if (key_control&32) {	-	-
logo_version	32	uimsbf
}	-	-
hardware_string_len	8	uimsbf
for(i = 0; i< hardware_string_len; i++){;	-	-
hardware_string_char	8	uimsbf
}	-	-

Syntax	Number of Bits	Identifier
}	-	-
download_PartInfo_tag = 0xeb	8	uimsbf
download_PartInfo_len	8	uimsbf
for (i = 0; i < N; i++)	-	-
download_data_totalsize	32	uimsbf
partition_count	8	uimsbf
reserved	8	uimsbf
part_description_length	16	uimsbf
for(i = 0; i < N; i++)	-	-
download_mode	8	uimsbf
download_mode_data_len	8	uimsbf
If(download_mode == 0) {//By address		uimsbf
download_addr	64	uimsbf
download_size	32	uimsbf
download_crc32	32	rpchof
}	-	-
else (download_mode ==1) {//By file name	-	-
downloadstring_length; i++){	8	uimsbf
for(i = 0; i < downloadstring_length)	-	-
download_string_char	8	uimsbf
}	-	-
download_size	32	uimsbf
download_crc32	32	rpchof
}	-	-
}	-	-
}	-	-
download_description_length	8	bslbf
for(i=0; i < download_description_length;i++){	-	-

Syntax	Number of Bits	Identifier
download_description_char	8	uimsbf
}	-	-
reserved_tag	8	uimsbf
reserved_data_length	16(<=512)	uimsbf
for (i = 0; i < reserved_data_length;i++)	-	-
{ reserved_data}	8	-
}	-	–
CRC_32	32	rpchof
}	-	-

The fields in Download-Info are described as follows:

- **download_table_id**: table ID of the download sequence (0x00–0xFD)
- **download_Info_tag**: descriptor of the download information. The value is **0xEA**.
- **download_Info_len**: length of the subsequent download information
- **key_control**: control over whether the webbrowser, kernel, and CA descriptions exist using bit validity
- **STB_manufacturerID**: manufacturer ID
- **hardware_version**: hardware version to which the downloaded software applies
- **software_version**: version of the downloaded software
- **app_version**: version of the downloaded application
- **kernel_version**: version of the downloaded kernel
- **CA_version**: version of the downloaded CA
- **bootloader_version**: BootLoader version
- **apploader _version**: Loader version
- **logo_version**: logo version
- **hardware_string_len**: length of the hardware version description (using character strings)
- **hardware_string**: character string for describing the hardware version
- **download_date**: date (year/month/day) of downloading the software in compressed BCD code format
- **serial_number_start**: start serial number of the STBs for which the software needs to be upgraded
- **serial_number_end**: end serial number of the STBs for which the software needs to be upgraded
- **download_PartInfo_tag**: descriptor of the download partition information. The value is **0xEB**.
- **download_PartInfo_len**: length of the subsequent download partition information
- **partition_count**: number of software programs that need to download data

- **download_type**: control over whether to upgrade by flash address or file
  - 0: upgrade by flash address
  - 1: upgrade by file name (this function is not enabled currently)
- **download_type_data_len**: length of the description information about the upgrade mode (by address or file)
- **download_addr**: address for the upgrade by flash address.
- **download_size**: size of the software to be upgraded
- **download_crc32**: CRC value of the partition software to be upgraded, which is used for checking the integrity of partition data
- **downloadstring_length**: length of the file name for the upgrade by file.
- **downloadstring_char**: description of the file for the upgrade by file.
- **download_description_length**: length of subsequent **download_description**
- **download_description_char**: download description
- **reserved_tag**: tag for reserved data
  - 0: without extended data
  - 1: with extended data
- **reserved_data_length**: length of reserved data
- **reserved_data_byte**: see Table 4-5.
- **CRC32**: CRC value of **Download_Info**

Table 4-5 reserved_data

Syntax	Number of Bits	Identifier
CRC32	32	rpchof
magic_num	32	Uimsbf

- **CRC32**: CRC value obtained after CRC check on all upgrade streams, which is used for checking the integrity of the entire upgrade data packet
- **magic_num**: random number magic number of the data packet, which differs every time and is used to ensure the uniqueness of the upgrade

4.2.5 Syntax of Partition_Control_Section

Table 4-6 describes the syntax of Partition_Control_Section.

Table 4-6 Partition_Control_Section syntax

Syntax	Number of Bits	Identifier
Partition_Control_Section ()	-	-
{	-	-
table_id	8	uimsbf
section_syntax_indicator = 1	1	bslbf

Syntax	Number of Bits	Identifier
'0'	1	bslbf
reserved	2	bslbf
section_length	12	uimsbf
table_id_extension = 0x0000	16	uimsbf
software_version	8	uimsbf
section_number = partition_number	8	uimsbf
last_section_number	8	uimsbf
for (i = 0; i < N; i++){	-	-
Partition_Control()	8	uimsbf
}	-	–
CRC32	32	rpchof
}	-	-

The fields in Partition_Control_Section are described as follows:

- **table_id**: download **table_id** specified in the Download_Control_Section, 8-bit width
- **section_syntax_indicator**: It is set to **1**.
- **section_length**: length (number of bytes) of the section (including the CRC) after this field
- **table_extension_id**: It is set to **0**.
- **software_version**: software version to which the data packet or control information belongs
- **section_number**: section serial number, which is the same as the partition number
- **last_section_number**: serial number of the last valid section

4.2.6 Syntax of Partition_Control

Table 4-7 describes the syntax of Partition_Control.

Table 4-7 Partition_Control syntax

Syntax	Number of Bits	Identifier
Partition_Control()	-	-
{	-	-
part_head_tag = 0xec	8	uimsbf
part_head_data_len	8	uimsbf
download_type	8	uimsbf

Syntax	Number of Bits	Identifier
reserved	8	uimsbf
part_datagram_number	16	uimsbf
part_total_size	32	uimsbf
part_ori_size	32	uimsbf
part_old_ver_start	32	uimsbf
part_old_ver_end	32	uimsbf
part_new_ver	32	uimsbf
if (download_type == 0) {	-	-
start_addr	64	uimsbf
}	-	-
else(download_type ==1) {	-	-
start_string_len	8	uimsbf
for(i = 0; i < start_string_len;i++) {	-	-
start_string	8	uimsbf
}	-	-
}	-	-
downloadstring_length	16	uimsbf
for(i = 0; i < downloadstring_length; i++) {	-	-
downloadstringchar	8	uimsbf
}	-	-
reserved_tag	8	uimsbf
reserved_data_length	16(<=512)	uimsbf
for(i = 0; i < reserved_data_length; i++) {	-	-
reserved_data_byte	8	uimsbf
}	-	-
}	-	-

The fields in Partition_Control are described as follows:

- **part_head_tag**: tag of the Partition_Control section
- **part_head_data_len**: valid data length of Partition_Control, that is, data length from this field to Partition_Control
- **download_type**: upgrade type

  - 0: upgrade by flash address
  - 1: upgrade by file name (not supported currently)
  - 2−15: reserved
- **part_datagram_number**: number of datagram packets in the upgrade file of the partition to be upgraded
- **part_total_size**: size of the software to be upgraded
- **part_ori_size**: original software size
- **part_old_ver_start**: start version number of the original partition software
- **part_old_ver_end**: end version number of the original partition software
- **part_new_ver**: new partition version number
- **start_addr**: address for the software to be upgraded in the flash memory
- **downloadstring_char**: description of the software to be upgraded
- **downloadstring_length**: length of the download description information
- **reserved_data_byte**: tag of the reserved field
  - 0: without extended data
  - 1: with extended data
- **reserved_data_length**: length of extended data
- **reserved_data**: data content. See Table 4-8.

Table 4-8 reserved_data

Syntax	Number of Bits	Identifier
Flash_type	32	Uimsbf
Flash_index	32	Uimsbf
End_addr	64	Uimsbf

- **Flash_type**: type of the target component. Currently the HD chip supports the following flash memories:
  - 0: partitions of the SPI flash for the STB
  - 1: partitions of the NAND flash for the STB
  - 2: partitions of the eMMC flash for the STB
- **Flash_index**: type of the data image to be downloaded. It indicates the CS of the target component type in the earlier protocols.
  - 00000: common file system image (none, UBI, or EXT3/4)
  - 10000: Yaffs file system image
- **End_addr**: end address for the upgrade partition

4.2.7 Syntax of Datagram_Section

Table 4-9 describes the syntax of Datagram_Section.

Table 4-9 Datagram_Section syntax

Syntax	Number of Bits	Identifier
Datagram_Section ()	-	-
{	-	-
table_id	8	uimsbf
section_syntax_indicator = 1	1	bslbf
'0'	1	bslbf
reserved	2	bslbf
section_length	12	uimsbf
table_id_extension = Datagram_number	16	uimsbf
software_version	8	uimsbf
section_number = partition_number	8	uimsbf
last_section_number	8	uimsbf
for (i = 0; i < N; i++){	-	-
{Datagram()}	-	-
}	-	–
CRC32	32	rpchof
}	-	-

The fields in Datagram_Section are described as follows:

- **table_id**: download **table_id** specified in Download_Control_Section
- **section_syntax_indicator**: section syntax indicator. It should be set to **1**.
- **section_length**: length (number of bytes) of the section (including the CRC) after this field
- **table_extension_id**: extension ID (equal to the datagram number), 16-bit width
- **software_version**: software version to which the data packet or control information belongs
- **section_number**: same as the partition number (not the current section number)
- **last_section_number**: number of the last datagram section

4.2.8 Syntax of Datagram

Table 4-10 describes the syntax of Datagram.

Table 4-10 Datagram syntax

Syntax	Number of Bits	Identifier
Datagram ()	-	-
{	-	-
magic_num	32	uimsbf
reserved_data_length	16	uimsbf
for (i = 0; i < reserved_data_length;i++){	-	-
reserved_data_byte	8	-
}	-	-
Datagram_current_section_number	8	uimsbf
data_length	16	uimsbf
for (i=0; i < data_length;i++){	-	-
data_byte	8	uimsbf
}	-	–
CRC32	32	rpchof
}	-	-

The fields in the Datagram are described as follows:

- **magic_num**: magic number of the data packet. All data packets in the same packaged TS have the same magic number, but different TSs have different magic numbers. When the upgrade data is received, the magic number can be used as the key word for filtering to ensure that the received upgrade data comes from the same upgrade stream.
- **reserved_data_length**: length of reserved data
- **reserved_data_byte**: reserved data
- **Datagram_current_section_number**: current section number in the datagram. The value ranges from 1 to 8. To ensure compatibility with the earlier versions, when the MSB is **1**, multiple sections are supported;
otherwise, **Datagram_current_section_number** is **0**, indicating that a datagram contains only one section. You can check whether the reception of a datagram is complete based on **Datagram_current_section_number** and **last_section_number**. If multiple sections are supported, the maximum partition size is 512 MB; otherwise, the maximum partition size is 64 MB. Due to this restriction, the HiLoader package tool has split the package internally. Therefore, the image size of a single partition can be at most 4 GB when the upgrade package is created.
- **data_length**: data length
- **Data_byte**: upgrade data
- **CRC32**: CRC value of valid payload data

4.3 SSU OTA Upgrade Stream Transfer Protocol

Data required for the SSU upgrade is carried in the DSM-CC sections that contain the Download_Server_Initiate (DSI), DownloadInfo_Indication (DII), and Download_DataBlock (DDB) data respectively. DSI and DII carry upgrade control information. To be specific, DSI carries the upgrade stream group information and version information, and DII carries the upgrade partition (module) information.

4.3.1 Syntax of the SSU MPEG-2 Private Section

The SSU OTA upgrade protocol uses the private section defined by MPEG-2 as the data carrier. Table 4-11 describes the syntax of the MPEG-2 private section. For details, see the ISO/IEC 13818-1 protocol.

Table 4-11 SSU private section (syntax of the DSM-CC_Section)

Syntax	Number of Bits	Identifier	Remarks
private_section()			
{			
table_id	8	uimsbf	
section_syntax_indicator	1	bslbf	
private_indicator	1	bslbf	
reserved	2	bslbf	
dsmcc_section_length	12	uimsbf	
table_id_extension	16	uimsbf	
reserved	2	bslbf	
version_number	5	uimsbf	
current_next_indicator	1	bslbf	
section_number	8	uimsbf	
last_section_number	8	uimsbf	
if(table_id==0x3A){			
LLCSNAP()			
}			
else if(table_id==0x3B){			
userNetworkMessage()			DSI or DII
}			
else if(table_id==0x3C){			
downloadDataMessage()			DDB

Syntax	Number of Bits	Identifier	Remarks
}			
else if(table_id==0x3D){			
DSMCC_descriptor_list()			
}			
else if(table_id==0x3E){			
for(i=0;i<N;i++){			
private_data_byte			
}			
}			
if(section_syntax_indicator==“0”){			
checksum	32		
}			
else {			
CRC32	32		
}			
}			

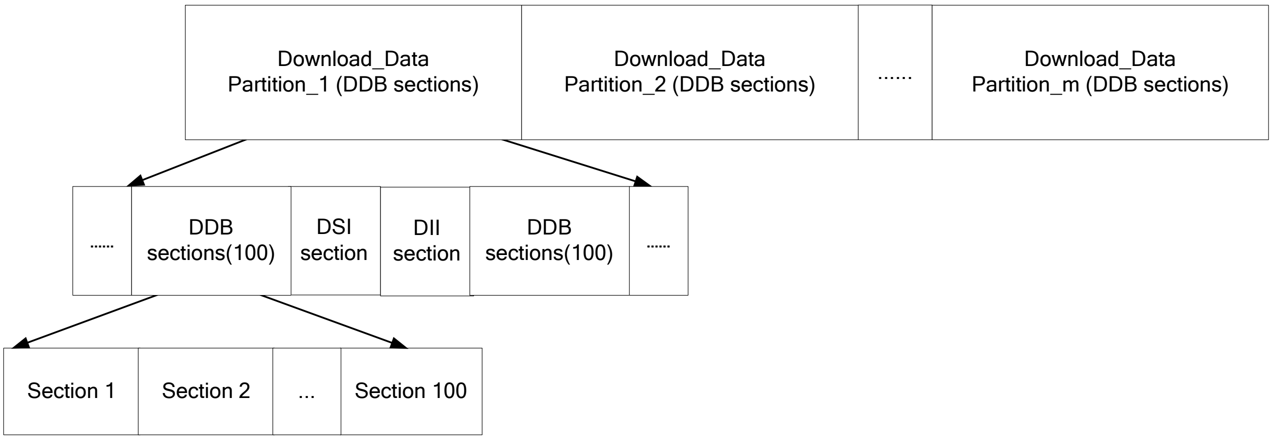
Table 4-12 Table_Id, Table_id_extension, and messageId in sections that carry DSI, DII, and DDB data

DSI/DII/DDB	Table ID	Table ID Extension	Message ID
DownloadServerInitiate	0x3B	Lower two bytes of Transaction_id	0x1006
DownloadInfoIndication	0x3B	Lower two bytes of Transaction_id	0x1002
DownloadDataBlock	0x3C	moduleId	0x1003

4.3.2 SSU Syntax of the Download Data Stream Section

Figure 4-3 shows the definitions of private data sections in the SSU protocol.

Figure 4-3 SSU upgrade data stream format


Download_Data Partition_1 (DDB sections)
Download_Data Partition_2 (DDB sections)
......
Download_Data Partition_m (DDB sections)
......
DDB sections(100)
DSI section
DII section
DDB sections(100)
......
Section 1
Section 2
...
Section 100

Each partition corresponds to a partition in the flash memory and consists of several Download_DataBlock sections. The Download_Server_Initiate and DownloadInfo_Indication control information is inserted for every 100 sections. Due to this restriction, the HiLoader package tool has split the package internally. Therefore, the image size of a single partition can be at most 4 GB when the upgrade package is created.

Table 4-13 describes the data stream sections.

Table 4-13 SSU data stream sections

Table ID	Part Number	Section Number	Meaning/Usage
0x3C	1	0	Download_DataBlock_Section
		...	Download_DataBlock_Section
		99	Download_DataBlock_Section
0x3B		100	Download_Server_Initiate
0x3B		101	DownloadInfo_Indication
0x3C		102	Download_DataBlock_Section
		...	Download_DataBlock_Section
		201	Download_DataBlock_Section
0x3C	2	100	Download_DataBlock_Section
			Download_DataBlock_Section
			Download_DataBlock_Section
0x3B		302	Download_Server_Initiate
0x3B		303	DownloadInfo_Indication
0x3C		100 sections	Download_DataBlock_Section
			Download_DataBlock_Section

Table ID	Part Number	Section Number	Meaning/Usage
			Download_DataBlock_Section
...	...	...	...
0x3C	N	100 sections	Download_DataBlock_Section
			Download_DataBlock_Section
			Download_DataBlock_Section
0x3B		xxx	Download_Server_Initiate
0x3B		xxx	DownloadInfo_Indication
0x3C		100 sections	Download_DataBlock_Section
			Download_DataBlock_Section
			Download_DataBlock_Section

The format of the upgrade data package is described as follows:

- One partition corresponds to one flash area.
- A partition consists of several DDB section packets.
- A Download_Server_Initiate section and a DownloadInfo_Indication section are inserted for every 100 DDB sections in a partition.

The HiLoader package tool has split the package internally. Therefore, the image size of a single partition can be at most 4 GB when the upgrade package is created.

4.3.3 Syntax of Download_Server_Initiate

Table 4-14 describes the syntax of Download_Server_Initiate.

Table 4-14 Download_Server_Initiate syntax

Syntax	Number of Bytes	Remarks
DownloadServerInitiate(){	-	-
dsmccMessageHeader()		For details about the syntax, see the description of **dsmccMessageHeader**.
serverId	20	It is filled with **0xFF**.
compatibilityDescriptor()	2	It contains only the **compatibilityDescriptorLength** field and is **0x0000**.
privateDataLength	2	-
for(i=0;i<privateDataLength; i++){	-	-

Syntax	Number of Bytes	Remarks
privateDataByte	1	It is filled with **GroupInfoIndication**.
}	-	-
}	-	-

Table 4-15 dsmccMessageHeader

Syntax	Number of Bytes	Remarks
dsmccMessageHeader(){	-	-
protocolDiscriminator	1	**0x11**, indicating DSM-CC
dsmccType	1	**0x03**, indicating that this message is a U-N download message
messageId	2	-
transactionId	4	DSI: The lower two bytes change between 0x0000 and 0x0001, and the upper two bytes indicate the version. DII: The value ranges from 0x0002 to 0xFFFF and is the same as **groupId** of the DSI.
reserved	1	-
adapationLength	1	-
messageLength	2	Length of all data after this field, including **dsmccAdapationHeader**
if(adatationLength>0){	-	-
dsmccAdapationHeader()	-	-
}	-	-
}	-	-

Table 4-16 GroupInfoIndication syntax

Syntax	Number of Bytes	Remarks
GroupInfoIndication() {	-	-
NumberOfGroups	2	Number of updates

Syntax	Number of Bytes	Remarks
for(i=0;i<numberOfGroups;i++) {	-	-
GroupId	4	The value ranges from 1 to **NumberOfGroups** and is the same as **transactionId** of the DII.
GroupSize	8	Size of the upgrade data in the group
GroupCompatibility	-	Equal to **CompatibilityDescriptor** of DSM-CC
GroupInfoLength	2	0x0000
for(i=0;i<N;i++) {	-	-
GroupInfoByte	1	This field is not defined in the SSU.
}	-	-
PrivateDataLength	2	0x0000
for(i=0;i<N;I++) {	-	-
PrivateDataByte	1	This field is not defined in the SSU.
}	-	-
}	-	-
}	-	-

Table 4-17 CompatibilityDescriptor

Syntax	Number of Bytes	Remarks
compatibilityDescriptor(){		
compatibilityDescriptorLength	2	
DescriptorCount	2	The value is **2**, including the hardware and software versions.
for(i=0;i<descriptorCount;i++){		Each **For** field contains 11 bytes.
descriptorType	1	See the **descriptorType** coding.
descriptorLength	1	

specifierType	1	0x01(IEEE OUI)
specifierData	3	
model	2	
version	2	
subDescriptorCount	1	The value is **0**, excluding **subdescriptor**.
for(i=0;i<subDescriptorCount;i++){		
subDescriptor()		
}		
}		
}		

When **CompatibilityDescriptor** appears in the DSI, **model** and **version** are used to identify the upgrade stream to be used in multiple upgrade streams of a vendor.

Table 4-18 descriptorType coding

Descriptor Type	Description
0x00	Pad descriptor
0x01	System hardware descriptor
0x02	System software descriptor
0x03 to 0x3F	ISO/IEC 13818-6 [1] reserved
0x40 to 0x7F	DVB reserved for future use
0x80 to 0xFF	User defined

4.3.4 Syntax of DownloadInfo_Indication

Table 4-19 describes the syntax of DownloadInfo_Indication.

Table 4-19 DownloadInfo_Indication syntax

Syntax	Number of Bytes	Remarks
DownloadInfoIndication(){	-	-
dsmccMessageHeader()	-	-
downloadId	4	Equal to **transactionId** in dsmccMessageHeader()

Syntax	Number of Bytes	Remarks
blockSize	2	Size (in byte) of each block in the DownloadDataBlock. The size of the last block of each module can be smaller than **blockSize**.
windowSize	1	-
ackPeriod	1	-
tCDownloadWindow	4	-
tCDownloadscenario	4	-
CompatibilityDescriptor()	2	It contains only the length field.
numberOfModules	2	
for(i=0;i< numberOfModules;i++){	-	-
moduleId	2	Bits 15–8 are equal to the lowermost byte of **groupId**, and bits 7–0 indicate **moduleId** (a module is a partition).
moduleSize	4	-
moduleVersion	1	It is related to the lowermost byte of **transactionId** in the DSI.
moduleInfoLength	1	-
for(i=0;i< N;i++){		module_extend_info
moduleInfoByte	1	-
}	-	-
}	-	-
privateDataLength	2	-
for(i=0; i< privateDataLength;i++){	-	-
privateDataByte	1	-
}	-	-
}	-	-

For each module, the SSU defines only the **moduleId** and **moduleSize** attributes. An additional attribute **module_extend_info** is defined to specify the position in the flash memory to which the partition data is written.

Table 4-20 module_extend_info

Syntax	Number of Bits	Identifier
module_extend_info(){	-	-
Flash_startaddr	64	Uimsbf
Flash _endaddr	64	Uimsbf
Flash_type	32	Uimsbf
Flash_index	32	Uimsbf
CRC32	32	rpchof
}	-	-

- **Flash_startaddr**: start address for the upgrade partition
- **Flash_endaddr**: end address for the upgrade partition
- **Flash_type**: type of the target component. Currently the HD chip supports the following flash memories:
  - 0: partitions of the SPI flash for the STB
  - 1: partitions of the NAND flash for the STB
  - 2: partitions of the eMMC flash for the STB
- **Flash_index**: type of the data image to be downloaded. It indicates the CS of the target component type in the earlier protocols.
  - 00000: common file system image (none, UBI, or EXT3/4)
  - 10000: Yaffs file system image
- **CRC32**: CRC value of valid payload data

4.3.5 Syntax of Download_DataBlock

Table 4-21 describes the syntax of Download_DataBlock.

Table 4-21 Download_DataBlock syntax

Syntax	Number of Bytes	Remarks
DownloadDataBlock(){	-	-
dsmccDownloadDataHeader()	-	-
moduleId	2	-
moduleVersion	1	-
reserved	1	-
blockNumber	2	-
for(i=0;i<N;i++){	-	-

Syntax	Number of Bytes	Remarks
blockDataByte	1	
}	-	-
}	-	-

Table 4-22 dsmccDownloadDataHeader

Syntax	Number of Bytes	Remarks
dsmccMessageHeader(){	-	-
protocolDiscriminator	1	**0x11**, indicating DSM-CC
dsmccType	1	**0x03**, indicating that this message is a U-N download message
messageId	2	-
downloadId	4	-
reserved	1	-
adapationLength	1	-
messageLength	2	-
If(adatationLength>0){	-	-
dsmccAdapationHeader()	-	-
}	-	-
}	-	-

4.4 USB/IP Upgrade Data Format

The USB upgrade file and IP upgrade file comply with the same protocol. See Table 4-23.

Table 4-23 Syntax of the USB/IP upgrade file

Syntax	Number of Bits	Identifier
File_header{	-	-
Magic_number = 0x4C4F4144	32	Uimsbf
Header_crc	32	rpchof

Syntax	Number of Bits	Identifier
Header_length	32	Uimsbf
File_length	32	Uimsbf
Manufactur_number	16	Uimsbf
for(i=0; i< Manufactur_number; i++){	-	-
manufacture_id	32	Uimsbf
Hardware_version	32	Uimsbf
Hardware_sub_version	32	Uimsbf
Software_version	32	Uimsbf
Serial_number_start	32	Uimsbf
Serial_number_end	32	Uimsbf
Download_type	32	Uimsbf
reserved	32	Uimsbf
Flash_map_number	16	Uimsbf
for(i=0; i< Flash_map_number; i++){	-	-
image_length	32	Uimsbf
image_offset	32	Uimsbf
partition _startaddr	64	Uimsbf
partition_endaddr	64	Uimsbf
Flash_type	32	Uimsbf
Flash_index	32	Uimsbf
}	-	-
for(i=0; i< Flash_map_number; i++){	-	-
Image_length	32	Uimsbf
Image_crc	32	rpchof
for(i=0;i< image_length;i++){	-	-
{ image _byte}	8	Uimsbf
}	-	-

Syntax	Number of Bits	Identifier
}	-	-
}	-	-

The fields in the file header are described as follows:

- **Magic_number**: magic number of the file. It should be **0x4C4F4144"LOAD"**.
- **Header_crc**: CRC code in the file header
- **Header_length**: length of the file header
- **File_length**: length of the upgrade image
- **Manufactur_number**: number of STB manufacturers
- **manufactur_id**: ID of the STB manufacturer
- **Hardware_version**: hardware version
- **Software_version**: software version
- **serial_number_start**: start serial number of the STBs for which the software needs to be upgraded
- **serial_number_end**: end serial number of the STBs for which the software needs to be upgraded
- **Download_type**: upgrade type control code, for specifying the software upgrade type. This field allows you to upgrade the software flexibly and control the upgrade risk. The supported upgrade types are as follows:
  - 0: forcible upgrade
  - 1: basic upgrade
  - 2: upgrade by serial number. See Table 4-24.
- **reserved**: reserved field
- **Flash_map_number**: number of upgrade files
- **image_offset**: offset address for the image upgrade data in the upgrade file
- **image_length**: upgrade data length
- **Image_crc**: CRC value of the upgrade data
- **partition_startaddr**: start address for the target flash partition of the upgrade
- **partition_endaddr**: end address for the target flash partition of the upgrade
- **Flash_type**: flash memory type
  - 0: SPI flash
  - 1: NAND flash
  - 2: eMMC flash
- **Flash_index**: type of the data image to be downloaded. It indicates the CS of the target component type in the earlier protocols.
  - 00000: common file system image (none, UBI, or EXT3/4)
  - 10000: Yaffs file system image

Table 4-24 Download_type upgrade type control code

Upgrade Type	Code Value	Remarks
Forcible upgrade	0x00	Forcible upgrade is implemented when the hardware version and software type are specified and the current software version is not the one in the upgrade stream (the user is not notified). The start serial number and end serial number for the upgrade are 0.
Basic upgrade	0x01	Basic upgrade is implemented when the hardware version and software type are specified and the current software version is earlier than the one in the upgrade stream (the user is notified). The start serial number and end serial number for the upgrade are 0.
Upgrade by batch	0x02	Upgrade by batch is implemented when the hardware version and software type are specified, the batch is within the batch range, and the software version is earlier than the one in the upgrade stream.
Upgrade by serial number	0x03	Upgrade by serial number is implemented when the hardware version and software type are specified, the serial number is within the range, and the software version is earlier than the one in the upgrade stream.
-	0x04–0xFF	Reserved