HiLoader

开发指南

文档版本　06

发布日期　2016-03-25

商标声明

 、HISILICON、海思和其他海思商标均为深圳市海思半导体有限公司的商标。

本文档提及的其他所有商标或注册商标，由各自的所有人拥有。

注意

您购买的产品、服务或特性等应受海思公司商业合同和条款的约束，本文档中描述的全部或部分产品、服务或特性可能不在您的购买或使用范围之内。除非合同另有约定，海思公司对本文档内容不做任何明示或默示的声明或保证。

由于产品版本升级或其他原因，本文档内容会不定期进行更新。除非另有约定，本文档仅作为使用指导，本文档中的所有陈述、信息和建议不构成任何明示或暗示的担保。

深圳市海思半导体有限公司

地址：　深圳市龙岗区坂田华为基地华为总部　邮编：518129

网址：　http://www.hisilicon.com

客户服务邮箱：　support@hisilicon.com

前 言

概述

本文档主要介绍高清芯片的 Loader 整体结构、业务开发。

产品版本

与本文档相对应的产品版本如下。

产品名称	产品版本
Hi3716C	V2XX
Hi3719C	V1XX
Hi3719M	V1XX
Hi3718C	V1XX
Hi3718M	V1XX
Hi3716M	V4XX
HI3796C	V100
Hi3796M	V1XX
Hi3798C	V1XX/V2XX
HI3798M	V1XX

读者对象

本文档（本指南）主要适用于以下工程师：

- 技术支持工程师
- 软件开发工程师

作者信息

章节号	章节名称	作者信息
全文	全文	L64565/G00277009

修订记录

修订记录累积了每次文档更新的说明。最新版本的文档包含以前所有文档版本的更新内容。

修订日期	版本	修订说明
2013-07-31	00B01	第一次临时版本发布。
2013-12-25	01	修改 2.3 章节描述；增加 Hi3716MV400 芯片。
2014-07-25	02	修改 1.6、2.6.2 章节。
2015-01-26	03	修改 2.2.1 章节步骤 2 的配置项。
2015-05-08	04	新增第 4 章，loader 升级协议(HISI、SSU、USB/IP)，仅用于 2015-05-08 以后发布的 SDK 版本。
2015-12-09	05	新增 HI3796M/HI3798M 及 HI3796C/HI3798C 芯片支持
2016-03-25	06	Loader 重构。

目 录

插图目录

1 方案概述

1.1 概述

本章主要描述 Loader 相关的重要概念、总体流程、代码结构、支持的升级方式、支持的协议类型。

1.2 重要概念

Loader 升级程序

负责处理数据下载、协议解析和数据存储的一段程序，是 Loader 的主体。根据 Loader 升级程序的运行环境，Loader 分为 AppLoader 和 BootLoader 两种。

AppLoader

Loader 升级程序运行于 kernel 环境，kerner 和 rootfs 经过裁剪后，与 Loader 升级程序及其依赖的驱动组成 initramfs 镜像，initramfs 镜像单独存储于 loader 分区。

BootLoader

Loader 升级程序运行于 boot 环境，boot、Loader 升级程序及其依赖的驱动被编译成 BootLoader 镜像，BootLoader 镜像存储于 boot 分区。

大系统

SDK 默认生成的系统镜像（包括内核、文件系统等）一般称之为大系统，这是相对 AppLoader 来说的，后者往往被称之为小系统。

1.3 代码构成

Loader 由升级检测程序、Loader 升级程序和 Loader API 三部分构成。这三部分的功能、BootLoader 代码路径以及 AppLoader 代码路径如表 1-1 所示。

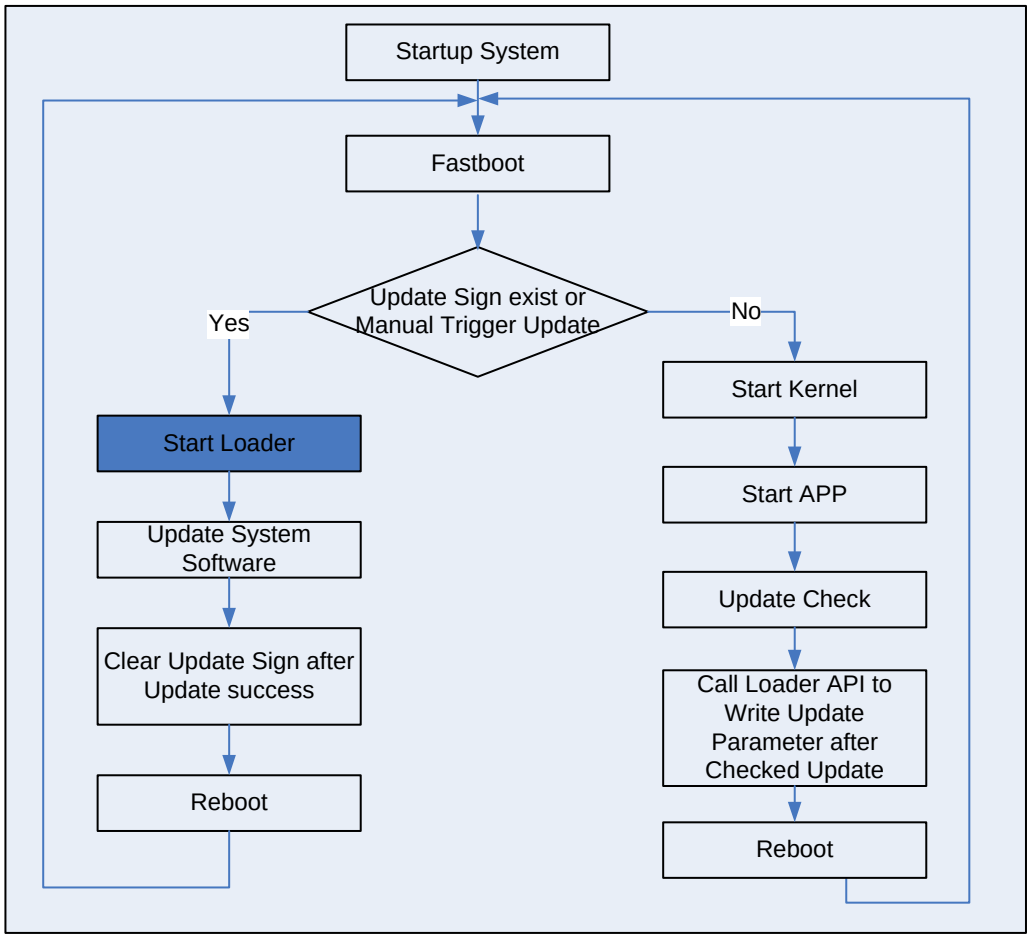
表1-1 Loader 构成

项目	功能	BootLoader 路径	AppLoader 路径
升级检测程序	检测是否有升级需求，如有则启动 Loader 升级程序	source/boot/product/loader/schedule/	source/boot/product/loader/schedule/
Loader 升级程序	1、 负责下载升级数据，按协议解析数据并将解析后的数据存储到存储设备 2、 通过 KEYLED、IR 实现和 loader 的配置交互 3、 在输出设备上(OSD)显示升级的进度信息、错误信息、配置信息等	source/boot/product/loader/app/	source/component/loader/app/
Loader API	提供 linux 环境下供 APP 调用的读写升级参数的接口	source/component/loader/api/	source/component/loader/api/

1.4 总体流程

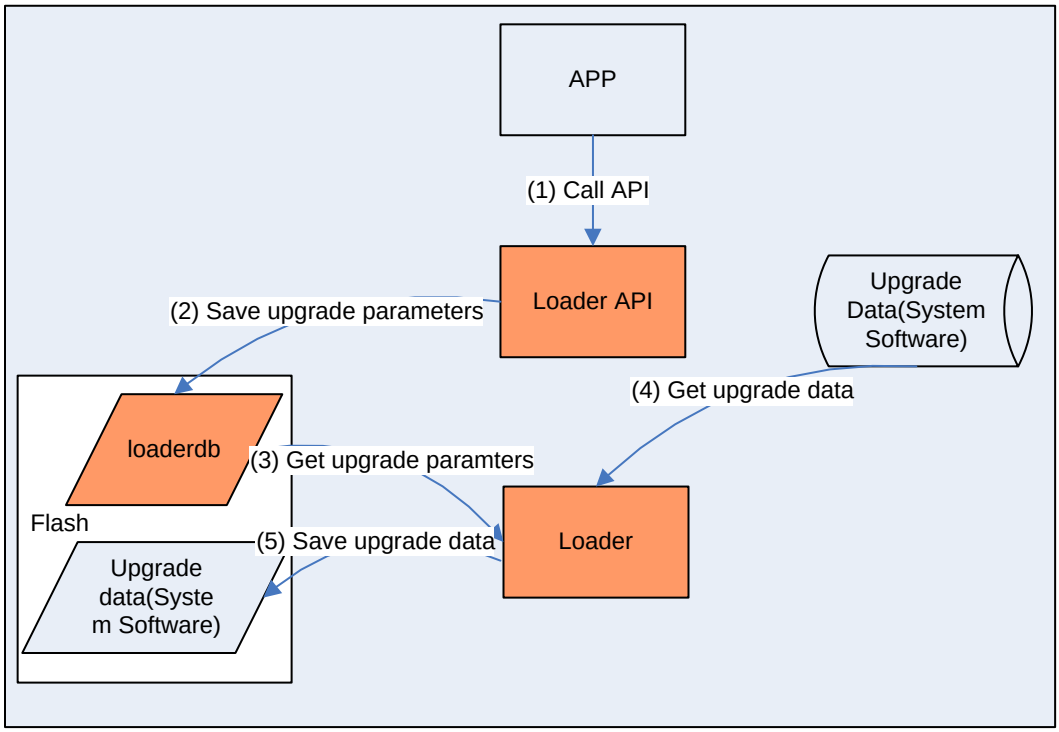
不管是 BootLoader 还是 AppLoader，总体流程如图 1-1 所示，两者之间的差异仅在于途中 Start Loader 的处理方式：BootLoader 直接调用 Loader 升级程序的入口函数；AppLoaLoader 首先从 loader 分区读取 initramfs 镜像到 DDR，然后启动 initramfs，最后执行 Loader 升级程序的 APP。

图1-1 Loader 升级方案的总体流程



简单的数据流如图 1-2 所示。

图1-2 数据流图



1.5 协议类型和升级方式

Loader 支持以下几种协议。

- SSU 协议

  全称 System Software Updates ， 是欧洲数字视频广播(DVB)标准制定的适用于 OTA 升级的系统软件升级服务技术协议。

- HISI OTA 协议

  海思制定的适用于 OTA 升级的协议。

- HISI FILE 协议

  海思制定的适用用于 USB 和 IP 升级的协议，亦称 USB/IP 协议。

AppLoader 支持 OTA、USB 和 IP 升级，BootLoader 支持 OTA、USB 升级。各升级方式的特点如表 1-2 所示。

表1-2 升级方式特点

升级方式	触发方式	特点
OTA	APP 触发//前面板手动触发	1 支持 HISI OTA 协议 2 支持 SSU 协议

升级方式	触发方式	特点
USB	APP 触发/前面板手动触发	1 支持 HISI FILE 协议 2 支持 FAT32/NTFS/EXT2/EXT3/EXT4 文件系统格式的 USB 升级 3 支持 APP 触发升级指定路径指定文件名的升级文件 4 支持前面板手动触发升级根目录下文件名为 usb_update.bin 的升级文件
IP	APP 触发	1 支持 TFTP 传输协议 2 支持 FTP 传输协议 3 支持 HTTP 传输协议

1.6 系统分区

deviceinfo 分区用以保存 stb 私有的配置信息，如厂商信息，硬件版本信息等。loaderdb 分区保存的是 Loader 升级必需的参数配置信息，为防止升级参数被异常破坏导致无法进行升级操作，（可选）增加 loaderdbbak 分区对 loaderdb 分区进行备份。softwareinfo 分区用于存储软件版本信息，包括软件版本号，各分区版本号，防回滚版本号。

deviceinfo、loaderdb、loaderdbbak、softwareinfo 等分区的命名固定，大小约束为一个 flash block，不允许修改；如修改，必须改变相关代码。

BootLoader 方案的系统分区划分例举如图 1-3 所示， 用户可根据自身需要划分分区。

图1-3 BootLoader 方案的系统分区

bootloader	bootargs	deviceinfo	softwareinfo	loaderdb	loaderdbbak	baseparam	logo	kernel	rootfs

AppLoader 本身需要一个 loader 分区，如需升级 Loader 自身，（可选）增加 loaderbak 分区对 loader 分区进行备份，以防升级过程中 loader 分区被破坏时可以用 loaderbak 分区恢复 loader 分区。

ApptLoader 方案的系统分区划分例举如图 1-4 所示， 用户可根据自身需要划分分区。

图1-4 AppLoader 方案系统分区

boot	bootargs	deviceinfo	softwareinfo	loaderdb	loaderdbbak	loader	loaderbak	baseparam	logo	kernel	rootfs

2 Loader 开发指引

2.1 概述

本章描述如何编译、执行及调试 Loader 程序。

2.2 Loader 的配置和编译

2.2.1 BootLoader

步骤 1 根据目标单板类型选择 SDK/configs 目录下的对应配置文件替换 SDK/cfg.mak。

步骤 2 执行 make menuconfig 命令进入配置界面，选中如下描述配置项：

```
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)  --->
```

执行 make distclean && make build 命令，构建成功结束后得到：

- BootLoader 镜像（SDK/pub/芯片类型/image/fastboot-burn.bin）

- Linux 下的 Loader 库（SDK/source/component/loader/api/libhiloader.a 和 SDK/source/component/loader/api/libhiloader.so）
- Linux 下的 Loader Sample（SDK/ sample/loader/sample_loader）

----结束

2.2.2 AppLoader

步骤 1 根据目标单板类型选择 SDK/configs 目录下的对应配置文件替换 SDK/cfg.mak。

步骤 2 执行 make menuconfig 命令进入配置界面，选中如下描述配置项：

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

步骤 3 执行 make distclean && make build 命令，构建成功结束后得到

- Boot 镜像（SDK/pub/芯片类型/image/fastboot-burn.bin）
- Linux 下的 Loader 库（SDK/source/component/loader/api/libhiloader.a 和 SDK/source/component/loader/api/libhiloader.so）
- Linux 下的 Loader Sample（SDK/ sample/loader/sample_loader）等。

步骤 4 使用 AppLoader 私有 cfg.mak 替换 SDK/cfg.mak，执行 make menuconfig 命令进入配置界面，选中如下描述配置项：

```
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)  --->
```

步骤 5 执行 make distclean && make build 命令，构建成功结束后得到

AppLoader 镜像（SDK/pub/芯片类型/image/ apploader.bin）

----结束

警告

如需支持 apploader,boot 要用步骤 3 编译出来的 fastboot-burn.bin,不可用 apploader 配置文件编译出来的 fastboot-burn.bin

2.3 Loader 打包工具的使用

关于如何使用打包工具的详细信息请参考《HiLoader 工具使用指南》文档。

2.4 Loader API 接口

2.4.1 概述

Loader API 提供了一套供 APP 触发升级和查询升级参数的接口。APP 调用这些接口可以设置和查询升级参数、查询软件版本信息、查询设备信息。API 相互独立，不依赖其他接口，不存在调用的先后顺序。本节举例说明如何使用 API 接口配置升级参数触发升级。

2.4.2 触发 USB 升级

```
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 触发卫星信号 OTA 升级

```
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; /**< 未启用 */
    pstOTAParam->u32Pid = 7000;

    /** Tuner Parameter Config */
    pstSatParam = &pstOTAParam->unParam.stSat;
    pstSatParam->u32IsiID = 0; /**< 未启用 */
    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 触发有线信号 OTA 升级

```
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; /**< 未启用 */
    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 触发地面信号 OTA 升级

```
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; /**< 未启用 */
    pstOTAParam->u32Pid = 7000;

    pstTerParam = &pstOTAParam->unParam.stTer;
    pstTerParam->u32PLPId = 0; /**< 未启用 */
    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 触发 IP 升级

```
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 应用场景

Loader 编译完毕后，将 Loader 镜像（BootLoader 或 AppLoader 镜像，取决于使用 BootLoader 还是 AppLoader）烧写到对应的 Flash 分区上，就可以运行升级程序了。

2.5.1 APP 触发 OTA 升级

场景说明

APP 将 OTA 升级参数（包括频点参数、PID 等）以及 OTA 升级类型标志写入 loaderdb 分区，重启系统进行 OTA 升级。

操作步骤

步骤 1 编译 Loader 镜像并烧入设备对应分区。

步骤 2 使用 HiLoader 工具将升级镜像打包成升级 TS 流。

步骤 3 配置前端升级服务器，并播放升级 TS 流。

步骤 4 运行 Loader sample 或运行 APP 触发 OTA 升级。

步骤 5 重启系统，系统自动启动 Loader 程序进入升级过程，等待升级结束。

----结束

注意事项

无。

示例

APP 触发 OTA 升级代码参考“2.4.3 触发卫星信号 OTA 升级”、“2.4.4 触发有线信号 OTA 升级”章节。

2.5.2 手动触发 OTA 升级

场景说明

设备重启时通过前面板输入强制升级组合键（例：Menu+OK）进入 loader 强制升级模式。如果包含有升级文件的 USB 设备被 loader 系统正确识别，则强制进入 USB 升级模式；否则进入 OTA 升级模式。用户可以采用不接入 USB 设备或 USB 设备中不存放合法升级文件的方式强制进入 OTA 升级模式。

操作步骤

步骤 1 编译 Loader 镜像并烧入设备对应分区。

步骤 2 使用 HiLoader 工具将升级镜像打包成升级 TS 流。

步骤 3 配置前端升级服务器，并播放升级 TS 流。

步骤 4 确保 USB 设备中没有合法的 USB 升级文件，或直接移除 USB 设备。

步骤 5 重启设备，按下前面板强制升级组合键，USB 升级文件检测失败，进入 OTA 升级模式。

步骤 6 根据 OSD 的窗口提示配置升级参数，按确认键执行升级。

----结束

注意事项

NA

实例

NA

2.5.3 APP 触发 USB 升级

场景说明

APP 将 USB 升级的文件路径（含文件名）以及 USB 升级类型标志写入 loaderdb 分区，重启系统进行 USB 升级。

操作步骤

步骤 1 编译 Loader 镜像并烧入设备对应分区。

步骤 2 使用 HiLoader 工具将升级镜像打包成 USB 升级文件，假设文件名称是 usb_update.bin。

步骤 3 将 usb_update.bin 拷贝到 USB 设备的特定目录下，此目录必须跟触发 USB 升级时配置的目录一致。假设拷贝到/usb 目录，将 USB 设备插入到 USB 接口。

步骤 4 运行 Loader sample 或运行 APP 触发 USB 升级，配置 USB 升级路径为 USB 升级文件所在路径。根据步骤 3 的假设，配置 USB 升级文件名为/usb/usb_update.bin。

步骤 5 重启系统，系统自动启动 Loader 程序进入升级过程，等待升级结束。

----结束

注意事项

APP 触发的 USB 升级可以指定升级文件的名称以及文件路径。

示例

APP 触发 USB 升级的代码参考“2.4.3 触发卫星信号 OTA 升级”章节。

2.5.4 手动触发 USB 升级

场景说明

设备重启时通过前面板输入强制升级组合键（例：Menu+OK）进入 loader 强制升级模式。如果包含有升级文件的 USB 设备被 loader 系统正确识别，则强制进入 USB 升级模式；否则进入 OTA 升级模式。用户在执行强制升级前将包含有合法升级文件的 USB 设备接入设备，可以强制进入 USB 升级模式。

操作步骤

步骤 1 编译 Loader 镜像并烧入设备对应分区。

步骤 2 使用 HiLoader 工具将升级镜像打包成 USB 升级文件，并命名为 usb_update.bin。

步骤 3 将打包出来的 USB 升级文件拷入 USB 设备的根目录。

步骤 4 重启设备，同时按下前面板强制升级组合键，系统将启动 Loader 程序进入升级过程。

步骤 5 设备自动检测 USB 升级文件，如果升级文件被识别，进入 USB 强制升级模式。

----结束

注意事项

NA

2.5.5 APP 触发 IP 升级

场景说明

APP 将 IP 升级参数（包括传输协议、服务器 ip、升级文件名、本地 ip 及网关等）写入 loaderdb 分区，重启系统进行 IP 升级。

操作步骤

步骤 1 编译 Loader 镜像并烧入设备对应分区。

步骤 2 使用 HiLoader 工具将升级镜像打包成 USB 升级文件。

步骤 3 配置前端升级服务器，并将打包出来的 USB 升级文件拷入升级服务器。

步骤 4 运行 Loader sample 或运行 APP 配置 IP 升级参数。

步骤 5 重启系统，系统自动启动 Loader 程序进入升级过程，等待升级结束。

----结束

注意事项

只有 AppLoader 支持 IP 升级。

示例

APP 触发 IP 升级的代码参考“2.4.6 触发 IP 升级”章节。

2.6 Loader 程序的调试

2.6.1 调试日志

在 BootLoader 方案中，可在 Fastboot 的命令模式下更改 loglevel(范围:0-4)的值来调整 log 的信息，loglevel 为 0 的时候关闭 log 信息，loglevel 为 4 的时候打开所有的 log 信息，如图 2-1 所示。

图2-1 设置 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#
```

在 AppLoader 方案中，可以使用 loglevel=4 ./loader 的方式运行 loader，输出调试日志。

2.6.2 模拟 APP 触发升级

Loader 开发过程中，常常需要进行调试，在系统还没有集成 APP 的情况下，可运行 Loader sample 触发升级来调试 Loader 程序，如图 2-2 所示。

图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.
```

sample_loader 的具体使用请参考 SDK/sample/loader 下的 readme 文档描述。

2.6.3 强制升级

单板重启时可通过前面板输入 Menu+Ok 组合键进入 loader 强制升级模式。如果包含有升级文件的 USB 设备被 loader 系统正确识别，则强制进行 USB 升级；否则进入 OTA 升级模式，用户可通过红外或前面板按键输入 OTA 参数并进行升级。

3 Loader 移植指引

3.1 概述

不同的客户对 Loader 有不同的需求，本章描述如何在现有 Loader 方案的基础上移植和开发（定制）客户自己的 Loader。

3.2 如何配置 Tuner

不同的单板 Tuner 硬件相关的属性不同，请参考 download_ota.c 对 Tuner 进行配置。

3.3 如何配置遥控器

主要接口介绍

- uiIRInit ()

  IR 设备初始化。

- uiIRDeInit()

  IR 设备去初始化。

- uiIRGetValue ()

  获取 IR 设备输入键码，该键码与 UI_KEYVALUE_E 对应，通过修改 UI_KEYVALUE_E 切换不同 IR 设备。

3.4 如何配置 KEYLED

主要接口介绍

- uiKEYLEDInit ()

  KEYLED 设备初始化，通过修改 UI_D_KEYLED_TYPE 宏定义配置 KEYLED 型号。

- uiKEYLEDDeInit ()

  KEYLED 设备去初始化。

- uiKEYLEDKeyConvert()

  将 KEYLED 键码转化为 IR 键码， KEYLED 键码对外不可见，用户可以通过扩展 UI_KEYVALUE_E 枚举的方式将 KEYLED 键码开放给上层应用。

- uiKEYLEDGetValue ()

  获取 IR 设备输入键码。

- uiKEYLEDDisplay()

  在 KEYLED 设备上显示这符串，通过 s_au8DigitalCode 数组配置不同型号设备的字符转换码。

3.5 如何控制升级正确的版本

loaderCheckVersionMatch 是当前系统版本与升级版本的比较函数。如需要满足一定的条件才能升级，则修改此接口的实现，返回 HI_SUCCESS 则升级，否则不升级。本方案中没有对当前系统版本与升级版本做比较，检测到升级版本就进行升级。

3.6 升级结束后如何处理

修改接口 loaderUpgradeDone 实现对升级结束（失败或成功）后的处理，是自动重启系统还是等待用户掉电重启。本方案中自动重启。

3.7 如何实现界面语言的切换

AppLoader 通过 make menuconfig 命令来切换界面语言。

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

BootLoader 方案中实现英文显示，如需其他语言显示，需要扩展字符输出函数以支持其他语言的显示。

3.8 如何修改手动触发升级的方式

修改接口 Loader_CheckManuForceUpgrade 来修改手动触发升级的方式。

3.9 如何添加新的下载方式

实现如下函数，就可以添加一种新的下载方式(XXX 为下载方式名称)。这些函数被协议类型的相关函数调用，可参考 Loader 已支持的协议类型中的方式调用这些函数。

- DOWNLOAD_XXX_Init
  下载方式初始化，执行如内存分配、建立与升级数据源端(前端或服务器)的连接等操作。
- DOWNLOAD_XXX_getdata
  从升级数据源端获取数据。
- DOWNLOAD_XXX_DeInit
  下载方式去初始化，执行如释放内存、销毁与升级数据源端(前端或服务器)的连接等操作。
- DOWNLOAD_USB_GetFileSize
  获取升级文件大小。

3.10 如何添加新的协议类型

实现如下函数，就可以添加一种新的协议类型(XXX 为协议类型名称)。这些函数被 Loader 的主控流程调用，在 protocol.c 中增加新协议的适配代码。

- PROT_XXX_Init
  协议初始化，可执行内存分配等操作，同时调用下载方式初始化接口建立系统与升级数据源端的连接。
- PROT_XXX_DeInit
  协议去初始化，可执行内存释放等操作，同时调用下载方式去初始化接口销毁系统与升级数据源端的连接。
- PROT_XXX_GetVersionInfo
  从数据源端获取升级数据的版本信息，会调用到下载方式的获取数据的接口。
- PROT_XXX_GetPartitionInfo
  从数据源端获取升级数据的镜像分区信息，会调用到下载方式的获取数据的接口。
- PROT_XXX_Process
  从数据源端获取升级镜像的数据，会调用到下载方式的获取数据的接口。

3.11 如何开发界面

请参考 Loader APP 程序 ui 目录代码。

- ui_gfx.c 图形组件封装
- ui_display.c 显示参数配置及显示单元的初始化与去初始
- ui_window.c 窗口管理组件
- ui_win_main.c 升级进度窗口管理
- ui_win_msgbox.c 提示信息窗口
- ui_win_setting.c 手动升级配置界面

4 Loader 升级协议

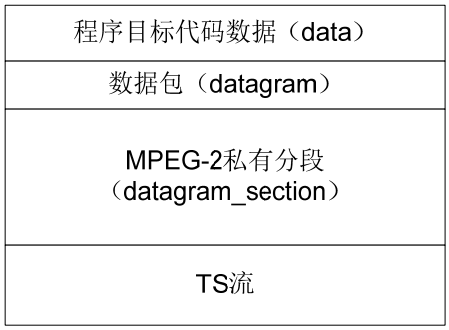
4.1 概述

本章节将绍 OTA 升级流传输协议(包括 HISI OTA 及 SSU OTA 两种协议)与 U 盘升级文件协议。IP 升级文件与 U 盘升级文件协议一样。

4.1.1 TS 流协议栈

TS 流升级数据的封装协议。协议栈如图 4-1 所示。

图4-1 协议栈



通过将目标数据（data）分割成大小合适的多个小块，将每个小块打包成一个数据包（datagram），再将数据包组合成符合 MPEG-2 标准的私有分段，最后形成了 TS 流。

4.1.2 TS 流结构

Loader 通过将升级数据打包封装，最终形成符合 MPEG-2 标准的 TS（Transport Stream）流，TS 流格式详细请参见“*ISO/IEC 13818-1*”。

4.2 HISI OTA 升级流传输协议

4.2.1 HISI MPEG-2 私有分段结构

HISI OTA 升级协议利用 MPEG-2 定义的私有分段（private section）作为数据的载体。采用的 MPEG-2 私有分段的结构如表 4-1 所示。详细描述请参见“*ISO/IEC 13818-1 协议*”。

表4-1 HISI 私有分段

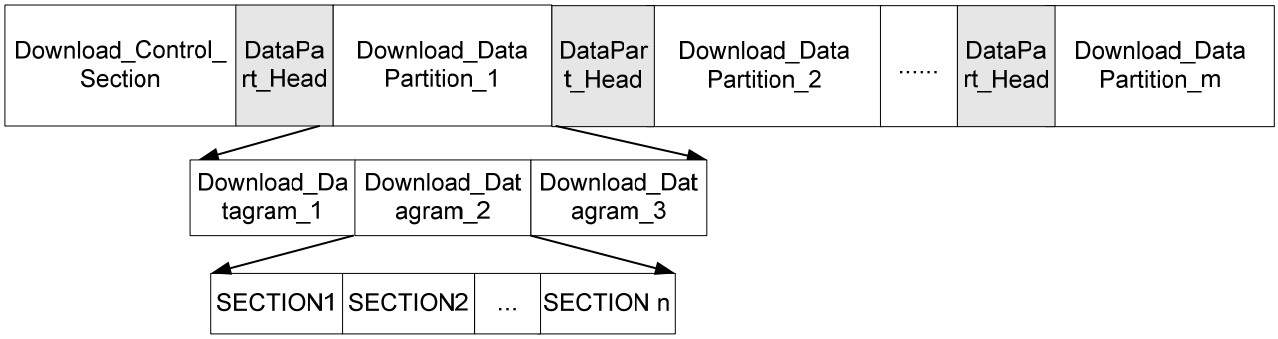
Syntax	No.of bits	Identifier
private_section()	-	-
{	-	-
table_id	8	uimsbf
section_syntax_indicator	1	bslbf
reserved	3	bslbf
private_section_length	12	uimsbf
if (section_syntax_indicator == ’0’){	-	-
for (i = 0; i < N; i++){	-	-
private_data_byte	8	uimsbf
}	-	-
}	-	-
else{	-	-
table_id_extension	16	uimsbf
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
}	-	-
}	-	-

注：section_syntax_indicator 字段为 1，表明在 private_section_length 字段之后有 table_id_extension 等字段，私有数据跟在 last_section_number 字段之后。

4.2.2 HISI 下载数据流分段结构图

HISI 的下载数据分段定义如图 4-2 所示。

图4-2 HISI 升级数据流的格式



其中：每一个 Partition 对应于 Flash 的一个分区，或对应 Flash 上某一个应用程序。每一个 Partition 由若干个 datagram 包构成，每个 Partition 最多包含 512MB 数据，Hiloader 打包工具做了内部拆包每个 Partition 最大支持 4G。规定每一个 datagram 最多包含 8KB 有效数据，每个 datagram 包最多包含 8 个 SECTION，每个 SECTION 最多包含 1024 个字节数据。

升级数据包格式说明如下：

- 1 个 partition 对应某 1 个待更新的软件或应用程序甚至 1 个文件，或者 1 个 Flash 的区域
- 1 个 partition 由若干个 datagram 包组成
- 1 个 datagram 包由 1 个到 8 个不等 datagram section 组成（目前 HiLoader 只支持 1 个 datagram 包含 1 个 datagram section，因此升级文件最大是 64MB）。最多有 2^16 个 extension_table_id，一个 extension_table_id 能接收 1 个 section，所以当前最大能接收 64MB，理论上 64*8MB。针对该限制，HiLoader 打包工具已经做了内部拆包，因此可以做打升级包时单分区的镜像大小可最大支持 4G

数据流分段如表 4-2 所示。

表4-2 HISI 数据流分段

Table_id	Section_number	Tbl_Ext_ID	Meaning/Usage
0xFE	0x00	0x00	Download_Control_Section
Table id specified in the download control section	0x01	0x00	Partition_Control(partition 1)
		0x01	Datagram 1
		0x02	Datagram 2
		0x03	Datagram 3

Table_id	Section_number	Tbl_Ext_ID	Meaning/Usage
0x00～0xFD		….	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

以下的 SECTION 在同一个 TS 流中，为相同的 PID：

- table_id=0xFE，section_number=0x00，table_id_extension=0x0000 这 3 个值决定该 section 为 Download_Control_Section
- download_table_id=0xXX，section_number=（1…n）partition_number，其中 table_id_extension=0x0000 决定该 section 为每个 partition 数据区的包头 SECTION
- download_table_id=0xXX，section_number=（1…..n）partition_number，table_id_extension=（1…n）datagram_number，last_section_number（值小于等于 8）表示每个 datagram 数据包有 section 的个数，在 SECTION 的内容中 Datagram_current_section_number 表示 datagram 数据包中的当前 SECTION

4.2.3 Download_Control_Section 结构

Download_Control_Section 结构如表 4-3 所示。

表4-3 Download_Control_Section

Syntax	No.of bits	Identifier
Download_Control_Section()	-	-
{	-	-
table_id = 0xfe	8	uimsbf
section_syntax_indicator = 1	1	bslbf

Syntax	No.of bits	Identifier
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
}	-	-

Download_Control_Section 各字段含义如下：

- table_id：置为 0xFE
- section_syntax_indicator：置为为 1
- section_length：自该字段之后的 section 的长度（字节数）包括 CRC 字段
- table_id_extension：置为 0
- software_version：标识数据包或者控制信息所属的软件版本号
- section_number：置为 0
- last_section_number：表示最后一个有效的 section 序号

4.2.4 Download_Info 结构

Download_Info 结构如表 4-4 所示。

表4-4 Download_Info 结构

Syntax	No.of bits	Identifier
Download_Info()	-	-
{	-	-
download_Info_tag = 0xea	8	uimsbf
download_Info_len	8	uimsbf
for (i = 0; i < N; i++)	-	-
{	-	-

Syntax	No.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	No.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_descritption_length	16	uimsbf
for (i = 0; i < N; i++){	-	-
download_mode	8	uimsbf
download_mode_data_len	8	uimsbf
if (download_mode == 0) { // 按地址		uimsbf
download_addr	64	uimsbf
download_size	32	uimsbf
download_crc32	32	rpchof
}	-	-
else (download_mode == 1) {//按文件名	-	-
downloadstring_length	8	uimsbf
for (i = 0; i < downloadstring_length; i++){	-	-
downloadstring_char	8	uimsbf
}	-	-
download_size	32	uimsbf
download_crc32	32	rpchof
}	-	-
}	-	-
}	-	-
download_description_length	8	bslbf
for (i = 0; i < download_description_length; i++){	-	-
download_description_char	8	uimsbf
}	-	-
reserverd_tag	8	uimsbf

Syntax	No.of bits	Identifier
reserverd_data_length	16(<=512)	uimsbf
for (i = 0; i < reserved_data_length;i++){	-	-
reserved_data_byte	8	-
}	-	-
CRC32	32	rpchof
}	-	-

Download_Info 各字段含义如下：

- download_table_id：指示下载序列的 table ID（0x00～0xFD）
- download_Info_tag=0xEA：表示下载信息的描述符
- download_Info_len：表示后续下载信息的数据长度
- key_control：通过位有效来控制“webbrowser、kernel、CA”等描述是否存在
- STB_manufacturerID：表示厂家代号
- hardware_version：表示下载软件适用的硬件版本号
- software_version：表示下载软件的版本号
- app_version：表示下载应用的版本号
- kernel_version：表示下载内核的版本号
- CA_version：表示下载 CA 的版本号
- bootloader_version：表示 BootLoader 的版本号
- apploader_version：表示 Loader 的版本号
- logo_version：表示 Logo 的版本号
- hardware_string_len：表示用字符标识硬件版本的描述长度
- hardware_string：硬件版本的描述字符串
- download_date：表示下载软件的日期（年/月/日），压缩 BCD 码格式
- serial_number_start：表示需要更新软件的机顶盒的起始序列号
- serial_number_end：表示需要更新软件的机顶盒的终止序列号
- download_PartInfo_tag=0xEB：表示下载分区信息的描述符
- download_PartInfo_len：表示后续下载分区信息的数据长度
- partition_count：指定要下载数据的软件数目
- download_type：指示按 Flash 的某个地址升级还是按文件升级
  - 0：按 Flash 地址进行升级。
  - 1：按文件名进行升级，目前还未启用此功能
- download_type_data_len：指示后续按某个类型升级的描述信息的长度
- download_addr：指示按 Flash 的某个地址升级的地址值

- download_size：指示待升级的某个软件的大小
- download_crc32：指示待升级的分区软件的 CRC 值，用来校验分区数据的完整性
- downloadstring_length：指示按文件类型的升级的文件名的长度
- downloadstring_char：按文件类型升级的文件名描述
- download_description_length：指定后续 download_description 的长度
- download_description_char：用于对下载进行文字描述
- reserved_tag：表示保留数据的标识
  - 0：无扩展数据
  - 1：有扩展数据
- reserved_data_length：表示后续保留数据的长度
- reserved_data_byte：数据内容，目前扩展如表 4-5 所示
- CRC32：Download_Info 的 CRC 校验值

表4-5 reserved_data 数据目前扩展内容

Syntax	No.of bits	Identifier
CRC32	32	rpchof
magic_num	32	Uimsbf

- CRC32：对所有升级流进行 CRC 校验，得出来统一 CRC 值，用来校验整个升级数据包的完整性
- magic_num：数据包的随机数魔术字，每次打包生成的魔术字都不一样，用来保证此次升级的唯一性

4.2.5 Partition_Control_Section 结构

Partition_Control_Section 结构如表 4-6 所示。

表4-6 Partition_Control_Section

Syntax	No.of bits	Identifier
Partiton_Control_Section ()	-	-
{	-	-
table_id	8	uimsbf
section_syntax_indicator = 1	1	bslbf
‘0’	1	bslbf
reserved	2	bslbf
section_length	12	uimsbf

Syntax	No.of bits	Identifier
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()	-	uimsbf
}	-	–
CRC32	32	rpchof
}	-	-

Partition_Control_Section 各字段含义如下：

- table_id：其值为 Download control section 中指定的 download table_id。
- sectition_syntax_indicator：置为 1
- section_length：指示自该字段之后的 SECTION 的长度（字节数）包括 CRC。
- table_extension_id：置为 0
- software_version：标识数据包或者控制信息所属的软件版本号
- section_number：表示该 SECTION 序号，应与 Partition 序号一致
- last_section_number：表示最后一个有效的 SECTION 序号

4.2.6 Partition_Control 结构

Partition_Control 结构如表 4-7 所示。

表4-7 Partition_Control 结构

Syntax	No.of bits	Identifier
Partiton_Control()	-	-
{	-	-
part_head_tag = 0xec	8	uimsbf
part_head_data_len	8	uimsbf
download_type	8	uimsbf
reserved	8	uimsbf
part_datagram_number	16	uimsbf
part_total_size	32	uimsbf

Syntax	No.of bits	Identifier
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_char	8	uimsbf
}	-	-
}	-	-
downloadstring_length	16	uimsbf
for (i = 0; i < downloadstring_length; i++){	-	-
downloadstring_char	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
}	-	-
}	-	-

Partition_Control 各字段含义如下：

- part_head_tag：Partition_Control Section 标识
- part_head_data_len：Partition Control 有效数据长度，后续数据的长度
- download_type：指示要更新的形式
  - 0：按 Flash 的某一个地址更新
  - 1：按文件名更新，暂不支持
  - 2～15：保留
- part_datagram_number：待更新分区的升级文件包含 datagram 包的个数

- part_total_size：待更新的软件大小
- part_ori_size：原来的软件大小
- part_old_ver_start：原来分区软件的版本号起始值
- part_old_ver_end：原来分区软件的版本号结束值
- part_new_ver：新的分区版本号
- start_addr：待更新的软件的 flash 的地址
- downloadstring_char ：待更新的软件名描述
- downloadstring_length：下载描述信息的长度
- reserved_tag：保留字段标记，为 0 时无扩展数据．为 1 时有扩展数据
- reserved_data_length：扩展数据长度
- reserved_data_byte：数据内容，目前扩展如表 4-8 所示

表4-8 reserved_data 数据目前扩展内容。

Syntax	No.of bits	Identifier
Flash_type	32	Uimsbf
Flash_index	32	Uimsbf
End_addr	64	Uimsbf

- Flash_type：用来表示目标器件的类型。目前高清支持 3 种类型 Flash，分别为：
  - 0：升级机顶盒 SPI Flash 上的分区
  - 1：升级机顶盒 NAND Flash 上的分区
  - 2：升级机顶盒 EMMC Flash 上的分区
- Flash_index：早期协议示目标器件类型的片选，目前表示要下载数据文件镜像类型：
  - 00000：普通文件系统镜像文件(none、ubi、ext3/4)
  - 10000：yaffs 文件系统镜像文件
- End_addr：升级分区结束地址

4.2.7 Datagram_Section 结构

Datagram_Section 结构如表 4-9 所示。

表4-9 Datagram section

Syntax	No.of bits	Identifier
Datagram_Section ()	-	-
{	-	-
table_id	8	uimsbf

Syntax	No.of bits	Identifier
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
}	-	-

Datagram section 各字段含义如下：

- table_id：其值为 Download control section 中指定的 download table_id。
- section_syntax_indicator： section 语法标识，应置为 1
- section_length：指示自该字段之后的 section 的长度（字节数），包括 CRC
- table_extension_id：表扩展 id，应等于 Data gram number
- software_version：标识数据包或者控制信息所属的软件版本号
- section_number： Partition 序号一致，不能表示当前的段序号
- last_section_number： data gram 的最后一个分段号码

4.2.8 Datagram 结构

Datagram 结构如表 4-10 所示。

表4-10 Datagram 结构

Syntax	No.of bits	Identifier
Datagram ()	-	-
{	-	-
magic_num	32	uimsbf
reserverd_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
}	-	-

Datagram 各字段含义如下：

- magic_num：数据包的魔术字。每个打包 TS 流中所有数据包具有相同的魔术字，但不同 TS 流中魔术字取值不同，在收升级数据时，把魔术字作为过滤关键字，能确保收到的升级数据都来自同一个升级流
- reserved_data_length：表示后续保留数据的长度
- reserved_data_byte：表示保留数据
- Datagram_current_section_number：表示当前 Datagram 中分段的编号，从 1 开始，最大值为 8，为了与老版本打包兼容，最高位为 1 时，表示启用多分段支持，否则 Datagram_current_section_number 为 0，表示一个 Datagram 仅包含一个 section。可根据 Datagram_current_section_number 与 last_section_number 来判断一个 Datagram 是否收完。启用多分段支持后，一个分区最大支持 512MB 大小，未启用之前仅支持 64MB 大小。目前，针对该限制，HiLoader 打包工具已经做了内部拆包，因此可以做打升级包时单分区的镜像大小可最大支持 4G。
- data_length：数据的长度
- data_byte：升级数据
- CRC32：有效载荷数据的 CRC 校验值

4.3 SSU OTA 升级流传输协议

SSU 升级需要的数据由分别包含 Download_Server_Initiate(DSI)、DownloadInfo_Indication(DII)和 Download_DataBlock(DDB)数据的 DSM-CC_Section 携带，其中 DSI 和 DII 携带升级控制信息，包括 DSI 携带升级流分组信息和升级流版本信息，DII 携带升级分区(module)信息。

4.3.1 SSU MPEG-2 私有分段结构

SSU OTA 升级协议利用 MPEG-2 定义的私有分段（private section）作为数据的载体。采用的 MPEG-2 私有分段的结构如表 4-11 所示。详细描述请参见“*ISO/IEC 13818-1 协议*”。

表4-11 SSU 私有分段（DSM-CC_Section 语法结构）

Syntax	No.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
}	-	-	-
else if (table_id == 0x3D){	-	-	-
DSMCC_descriptor_list()	-	-	-
}	-	-	-
else if (table_id == 0x3E){	-	-	-

Syntax	No.of bits	Identifier	remarks
for (i = 0;i < N; i++){	-	-	-
private_data_byte	-	-	-
}	-	-	-
}	-	-	-
if (section_syntax_indicator == “0”){	-	-	-
checksum	32	-	-
}	-	-	-
else {	-	-	-
CRC32	32	-	-
}	-	-	-
}	-	-	-

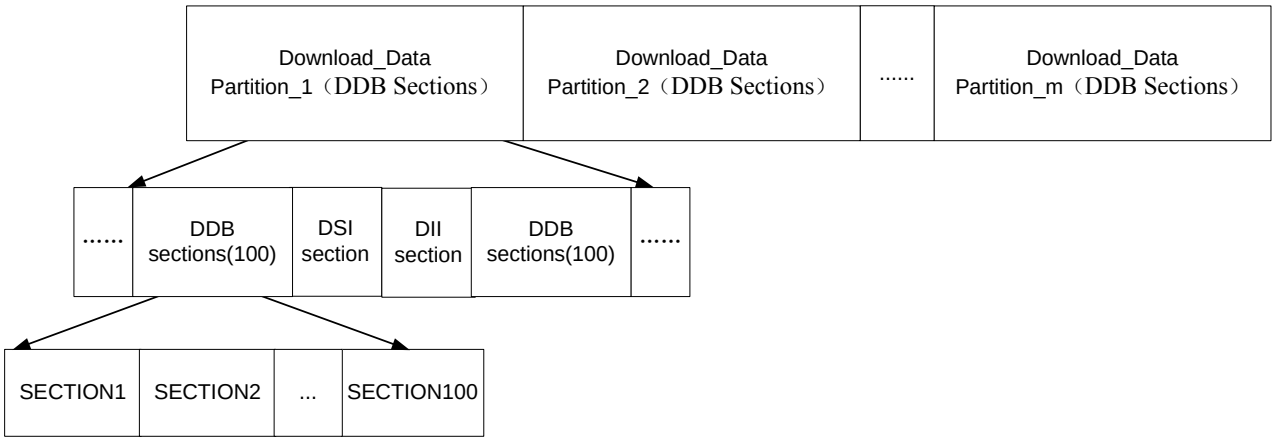
表4-12 Table_Id、Table_id_extension、messageId 在携带 DSI、DII、DDB 结构的 Section 中的分配情况:

	Table_id	Table_id_extension	messageId
DownloadServerInitiate	0x3B	Transaction_id 低 2 字节	0x1006
DownloadInfoIndication	0x3B	Transaction_id 低 2 字节	0x1002
DownloadDataBlock	0x3C	moduleId	0x1003

4.3.2 SSU 下载数据流分段结构图

SSU 协议私有数据分段定义如图 4-2 所示。

图4-3 SSU 升级数据流的格式



其中：每一个 Partition 对应于 Flash 的一个分区，每一个 Partition 由若干个的 Downloade_DataBlock 类型的 section 组成，而每隔 100 个 section 会插入一个 Download_Server_Initiate 及一个 DownloadInfo_Indication 类型的控制信息， Hiloader 打包工具做了内部拆包每个 Partition 最大支持 4G 大小的镜像。

数据流分段如表 4-2 所示。

表4-13 SSU 数据流分段

Table_id	Part Num	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 sections	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 Num	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

升级数据包格式说明如下：

- 1 个 partition 对应 1 个 Flash 的区域
- 1 个 partition 由若干个 DDB sections 包组成
- 1 个 partition 内每隔 100 个 section 会插入 Download_Server_Initiate 及 DownloadInfo_Indication 类型的控制信息 section
- HiLoader 打包工具已经做了内部拆包，因此可以做打升级包时单分区的镜像大小可最大支持 4G

4.3.3 Download_Server_Initiate 结构

表4-14 Download_Server_Initiate 语法结构

Syntax	Num.of Bytes	remarks
DownloadServerInitiate(){	-	-
dsmccMessageHeader()	-	语法结构见 dsmccMessageHeader
serverId	20	全部用 0xFF 填充
compatibilityDescriptor()	2	仅含有 compatibilityDescriptorLength 字段，且为 0x0000
privateDataLength	2	-
for (i = 0; i < privateDataLength; i++){	-	-
privateData_Byte	1	用数据结构 GroupInfoIndication 填充

Syntax	Num.of Bytes	remarks
}	-	-
}	-	-

表4-15 dsmccMessageHeader

Syntax	Num.of Bytes	remarks
dsmccMessageHeader(){	-	-
protocolDiscriminator	1	0x11，标明是 DSM-CC
dsmccType	1	0x03，表明该消息是一条 U-N 下载消息
messageId	2	-
transactionId	4	DSI：低 2 字节在 0x0000 和 0x0001 之间变化，高 2 字节标志版本 DII：取值范围 0x0002~0xFFFF，与 DSI 中的 groupId 相同
reserved	1	-
adapationLength	1	-
messageLength	2	字段之后所有数据的长度，包括 dsmccAdapationHeader
if (adatationLength > 0){	-	-
dsmccAdapationHeader()	-	-
}	-	-
}	-	-

表4-16 GroupInfoIndication structure

Syntax	Num. of Bytes	remarks
GroupInfoIndication() {	-	-
NumberOfGroups	2	number of updates
for (i = 0;i < numberOfGroups; i++) {	-	-

Syntax	Num. of Bytes	remarks
GroupId	4	取值范围:1~ NumberOfGroups,与 DII 的 transactionId 相等
GroupSize	4	本分组升级数据的累积大小
GroupCompatibility	-	等同 DSM-CC 的 CompatibilityDescriptor
GroupInfoLength	2	0x0000
for (i = 0; i < N;i ++) {	-	-
GroupInfoByte	1	SSU 没有定义此字段
}	-	-
PrivateDataLength	2	0x0000
for (i = 0; i < N; i++) {	-	-
PrivateDataByte	1	SSU 没有定义此字段
}	-	-
}	-	-
}	-	-

表4-17 CompatibilityDescriptor

Syntax	Num. of Bytes	remarks
compatibilityDescriptor(){	-	-
compatibilityDescriptorLength	2	-
DescriptorCount	2	取值 2，包含硬件、软件版本
for (i = 0; i < descriptorCount; i++){		每个 for 里面包含 11 字节
descriptorType	1	见 descriptorType coding
descriptorLength	1	-
specifierType	1	0x01(IEEE OUI)
specifierData	3	-

Syntax	Num. of Bytes	remarks
model	2	-
version	2	-
subDescriptorCount	1	取值 0,不包含 subdescriptor
for (i = 0; i < subDescriptorCount; i++){	-	-
subDescriptor()	-	-
}	-	-
}	-	-
}	-	-

注：CompatibilityDescriptor 在 DSI 中出现时，model 和 version 用来在一个产商的多组升级流中识别本机要升级的升级流。

表4-18 descriptorType coding

descriptorType	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 DownloadInfo_Indication 结构

表4-19 DownloadInfo_Indication 语法结构:

Syntax	Num. of Bytes	Remarks
DownloadInfoIndication(){	-	-
dsmccMessageHeader()	-	-
downloadId	4	等于 dsmccMessageHeader() 中的 transactionId

Syntax	Num. of Bytes	Remarks
blockSize	2	DownloadDataBlock 消息中传输的每个块的字节长度，而每个模块的最后一个块的长度可以小于 blockSize
windowSize	1	-
ackPeriod	1	-
tCDownloadWindow	4	-
tCDownloadscenario	4	-
CompatibilityDescriptor()	2	仅包含长度字段
numberOfModules	2	-
for (i = 0;i < numberOfModules; i++){	-	-
moduleId	2	Bit 15-8 等于 groupId 的最低一个字节;bit 7-0 表示 moduleId(一个 module 相当于一个分区)
moduleSize	4	-
moduleVersion	1	与 DSI 中的 transactionId 的最低一个字节相关
moduleInfoLength	1	-
for (i = 0;i < N; i++){	-	module_extend_info
moduleInfoByte	1	-
}	-	-
}	-	-
privateDataLength	2	-
for (i = 0; i < privateDataLength; i++){	-	-
privateDataByte	1	-
}	-	-
}	-	-

对于每个 module，SSU 中只定义了 moudleId 和 moduleSize 两个属性，额外定义 module_extend_info 描述 module 属性，用来说明这个分区数据写到 Flash 的位置。

表4-20 module_extend_info 描述

Syntax	No.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：升级分区起始地址
- Flash_endaddr：升级分区结束地址
- Flash_type：用来表示目标器件的类型。目前高清支持 3 种类型 Flash，分别为：
  - 0：升级机顶盒 SPI Flash 上的分区
  - 1：升级机顶盒 NAND Flash 上的分区
  - 2：升级机顶盒 EMMC Flash 上的分区
- Flash_index：早期协议示目标器件类型的片选，目前表示要下载数据文件镜像类型：
  - 00000：普通文件系统镜像文件(none、ubi、ext3/4)
  - 10000：yaffs 文件系统镜像文件
- CRC32：有效载荷数据的 CRC 校验值

4.3.5 Download_DataBlock 结构

表4-21 Download_DataBlock 语法结构

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

Syntax	Num. of Bytes	remarks
blockDataByte	1	
}	-	-
}	-	-

表4-22 dsmccDownloadDataHeader

Syntax	Num.of Bytes	remarks
dsmccMessageHeader(){	-	-
protocolDiscriminator	1	0x11，标明是 DSM-CC
dsmccType	1	0x03，表明该消息是一条 U-N 下载消息
messageId	2	-
downloadId	4	-
reserved	1	-
adapationLength	1	-
messageLength	2	-
if (adatationLength > 0){	-	-
dsmccAdapationHeader()	-	-
}	-	-
}	-	-

4.4 U 盘/IP 升级的数据格式

USB 升级文件与 IP 升级文件遵循相同的协议，描述内容如表 4-23 所示。

表4-23 USB 升级文件与 IP 升级文件描述内容

Syntax	No. of bits	Identifier
File_header{	-	-
Magic_number = 0x4C4F4144	32	Uimsbf
Header_crc	32	rpchof
Header_length	32	Uimsbf

Syntax	No. of bits	Identifier
File_length	32	Uimsbf
Manufactur_number	16	Uimsbf
for (i = 0; i < Manufactur_number; i++){	-	-
manufacture_id	32	Uimsbf
Hardware_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	No. of bits	Identifier
}	-	-

File_header 各字段含义如下：

- Magic_number： file 的 magic nubmer，应该为 0x4C4F4144“LOAD”
- Header_crc：文件头的 CRC 校验码
- Header_length：文件头长度
- File_length：全升级镜像总长度
- Manufactur_number：机顶盒厂商数量
- manufacture_id：机顶盒厂商 id 号
- Hardware_version：硬件版本号
- Software_version：软件版本号
- serial_number_start：表示需要进行软件更新的机顶盒的起始序列号
- serial_number_end：表示需要进行软件更新的机顶盒的终止序列号
- Download_type：升级类型控制码，用于控制软件升级的类型，通过这个字段，可以灵活多变的进行软件升级，并且可以控制升级的风险。32 位宽度，升级类型：
  - 0：强制升级
  - 1：基本升级
  - 2：按序列号升级。如表 4-24 所示
- reserved：预留字段
- Flash_map_number：升级文件的个数
- image_offset：该镜像升级数据整个升级文件中的偏移地址
- image_length：升级数据长度
- Image_crc：升级数据 CRC 校验值
- partition _startaddr ：升级目标 Flash 分区起始地址
- partition_endaddr：升级目标 Flash 分区结束地址
- Flash_type：flash 类型
  - 0：SPI Flash
  - 1：NAND Flash
  - 2：eMMC Flash
- Flash_index：前期表示目标器件类型的片选，表示要下载数据文件镜像类型：
  - 00000：普通文件系统镜像文件(none、ubi、ext3/4)
  - 10000：yaffs 文件系统镜像文件

表4-24 Download_type 升级类型控制码

升级类型	CODE 值	备注
强制升级	0x00	在指定硬件版本、软件类型条件下，软件版本不等于当前流中软件版本的用户升级（不提示用户选择），升级起始序列号、终止序列号全部为 0。
基本升级	0x01	在指定硬件版本、软件类型条件下，所有低于当前播发软件版本的用户升级（提示用户选择），升级起始序列号、终止序列号全部为 0。
按批次升级	0x02	在指定硬件版本、软件类型条件下，在批次范围中并且软件版本低于当前播发软件版本的用户升级。
按序列号升级	0x03	在指定硬件版本、软件类型条件下，在序列号范围中，并且软件版本低于当前播发软件版本的用户升级。
-	0x04～0xFF	保留。