U-boot 下自动升级

功能设计说明书

文档版本　08

发布日期　2016-08-31

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电气生产中心　邮编：518129

网址：　http://www.hisilicon.com

客户服务电话：　+86-755-28788858

客户服务传真：　+86-755-28357515

客户服务邮箱：　support@hisilicon.com

前 言

概述

本文档主要是 U-boot 下 SD 卡/ U 盘自动升级功能的需求、应用场景分析及其升级原理等。并详细描述了 U-boot 下 SD 卡/ U 盘升级功能的设计思路和实现细节，为后续工作相关功能开发提供参考。

说明

本文未做特殊说明，Hi3516D 与 Hi3516A 完全一致

本文未做特殊说明，Hi3520DV300 与 Hi3521A，Hi3518EV201、Hi3516CV200 与 Hi3518EV200 完全一致

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	说明
Hi3516C	V100	-
Hi3518	V100	包含 Hi3518A 和 Hi3518C
Hi3518E	V200	包含 Hi3518EV200、Hi3518EV201 和 Hi3516CV200
Hi3520A	V100	-
Hi3521	V100	-
Hi3531	V100	-
Hi3531A	V100	Hi3531A 不支持 SD 卡升级
Hi3520D	V100	Hi3520D 不支持 SD 卡升级
Hi3515A	V100	Hi3515A 不支持 SD 卡升级
Hi3535	V100	Hi3535 不支持 SD 卡升级
Hi3516A	V100	-
Hi3516D	V100	-

产品名称	产品版本	说明
Hi3536	V100	Hi3536 不支持 SD 卡升级
Hi3521A	V100	Hi3521A 不支持 SD 卡升级
Hi3520D	V300	Hi3520DV300 不支持 SD 卡升级
Hi3519	V100	-
Hi3519	V101	-
Hi3516C	V300	-

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

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
2016-08-31	08	添加 Hi3516CV300 的相关内容。
2016-07-10	07	添加 Hi3519V101 的相关内容。
2015-10-31	06	添加 Hi3519V100 的相关内容。
2015-09-01	05	添加 Hi3531A 的相关内容。
2015-08-05	04	添加 Hi3518EV200/Hi3518EV201/Hi3516CV200 的相关内容。
2015-06-27	03	添加 Hi3521A/Hi3520DV300 的相关内容。
2015-04-01	02	添加 Hi3536 的相关内容。
2014-12-30	01	添加 Hi3516D 的相关内容。
2014-10-19	00B03	补充 Hi3516A 的相关描述。
2013-11-18	00B02	补充 Hi3535 的相关描述。
2013-04-08	00B01	第 1 次临时版本发布。

目 录

1 设计分析 3
1.1 需求分析 3
1.2 流程设计 3
2 实现过程 3
2.1 整体结构 3
2.1.1 编译开关 3
2.1.2 模块结构 3
2.2 具体实现 3
2.2.1 FAT 文件系统支持 3
2.2.2 MMC 驱动移植 3
2.2.3 使能 USB OHCI 3
2.2.4 手动升级验证移植的 MMC 驱动 3
2.2.5 手动升级验证移植的 USB OHCI 驱动 3
2.2.6 增加自动升级入口 3
3 详细设计 3
3.1 Auto Update 详细设计 3
3.1.1 数据描述 3
3.1.2 函数描述 3

插图目录

图 1-1 U-boot 下 SD 卡/U 盘升级简要流程图 3
图 3-1 函数运行过程图..... 3

1 设计分析

1.1 需求分析

以前的升级方案是客户通过上层应用实现，该方案可靠性差、灵活性不高，我们现在提供一种更加可靠灵活的升级方案。

在产品应用中经常有升级 uboot、kernel 和 rootfs 的需要，只需按照以下步骤就可完成升级。

步骤 1. 将提供的镜像文件拷贝到自备的 SD 卡或 U 盘。

步骤 2. 插上 SD 卡或 U 盘。

步骤 3. 重启设备。

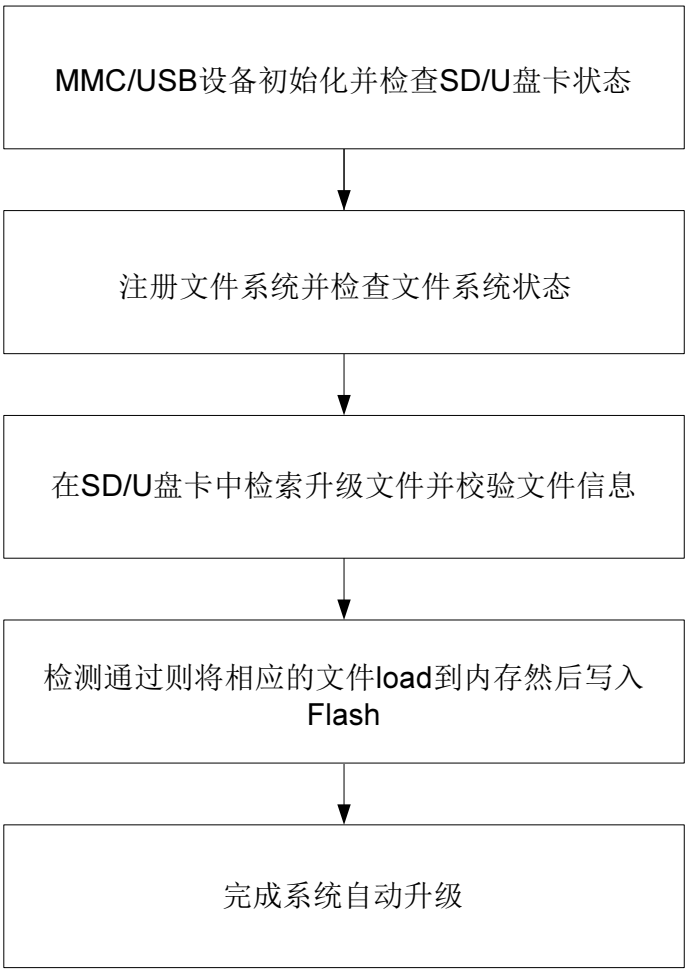
步骤 4. 等待升级完成。

----结束

1.2 流程设计

升级流程如图 1-1 所示。

图1-1 U-boot 下 SD 卡/U 盘升级简要流程图



说明

当同时插入 SD 卡和 U 盘时，若 SD 卡中没有有效升级包，则系统自动转入 U 盘升级，若 SD 卡中含有有效升级包直接使用 SD 卡中的升级包升级系统，不再扫描 U 盘中的内容。具体实现见 3.1.2.2 “内部函数实现”。

2 实现过程

说明

- 以下的实现均以 Hi3536 为例进行说明，其中 SD 卡升级以 Hi3516A 为例进行说明，具体的代码路径需要根据实际使用芯片修改。
- Hi3520D/Hi3515A/Hi3535/Hi3536/Hi3521A/Hi3520DV300 平台仅支持 U 盘升级，不支持 SD 卡升级。

2.1 整体结构

2.1.1 编译开关

自动升级的编译开关在芯片配置文件中，用户可根据需要打开相应的宏定义开关。

- 配置文件路径
  - include/configs/hi3516c.h 【hi3516c】
  - include/configs/hi3518a.h 【hi3518a】
  - include/configs/hi3518c.h 【hi3518c】
  - include/configs/hi3518ev200.h【hi3518ev200】
  - include/configs/hi3518ev201.h【hi3518ev201】
  - include/configs/hi3516cv200.h【hi3516cv200】
  - include/configs/ hi3520d.h 【hi3520d/hi3515a】
  - include/configs/ godcare.h 【hi3520a】
  - include/configs/ godarm.h 【hi3521】
  - include/configs/ godnet.h 【hi3531】
  - include/configs/ hi3535.h 【hi3535】
  - include/configs/ hi3516a.h 和 include/configs/hi3516a_spinand.h 【hi3516a】
  - include/configs/ hi3536.h 和 include/configs/hi3536_spinand.h 【hi3536】
  - include/configs/ hi3521a.h 【hi3521A】
  - include/configs/ hi3531a.h 【hi3531A】
  - include/configs/hi3519.h 和 include/configs/hi3519_nand.h 【hi3519v100】

- include/configs/hi3519v101.h 和 include/configs/hi3519v101_nand. 【hi3519v101】
- include/configs/hi3516cv300.h 【hi3516cv300】

下面以 Hi3516A 代码为例，说明各部分的相关作用

```
/*-----------------------------------------------------------------
 * sdcard/usb storage system update
 * ----------------------------------------------------------------*/
#define CONFIG_AUTO_UPDATE                    1     //升级功能总开关
#ifdef CONFIG_AUTO_UPDATE
       #define CONFIG_AUTO_SD_UPDATE          1     //SD卡升级开关
       #define CONFIG_AUTO_USB_UPDATE         1     //U盘升级开关
#endif

// 以下四行是SD卡/U盘升级的公共部分包含FAT文件系统编译开关
#define __LITTLE_ENDIAN                       1
#define CONFIG_DOS_PARTITION                  1

#define CONFIG_CMD_FAT                        1
/*-----------------------------------------------------------------
 * sdcard
 * ----------------------------------------------------------------*/
#ifdef CONFIG_AUTO_SD_UPDATE                //  SD卡驱动编译开关，受总开关控
制
       #define CONFIG_HIMCI_HI3516a
       #define REG_BASE_MCI                  0x206e0000
       #define CONFIG_HIMCI_V100
       #define CONFIG_GENERIC_MMC
       #define CONFIG_MMC                    1
       #define CONFIG_CMD_MMC
#endif
/*-----------------------------------------------------------
 * usb
 * ----------------------------------------------------------------*/
//USB 驱动默认打开，上述的 CONFIG_AUTO_USB_UPDATE 开关在升级流程上起控制作用
#define CONFIG_USB_OHCI                       1
#define CONFIG_CMD_USB                        1
#define CONFIG_USB_STORAGE                    1
#define CONFIG_USB_XHCI                         1
#define CONFIG_SYS_USB_XHCI_MAX_ROOT_PORTS      2
#define CONFIG_LEGACY_USB_INIT_SEQ
```

以上宏定义包含三部分：

- 升级功能总开关及 SD 卡/U 盘升级的公共部分
- SD 卡驱动
- USB OHCI 驱动

  注释或者打开 CONFIG_AUTO_UPDATE 宏定义直接控制是否支持 SD 卡/U 盘自动升级功能。

  在打开升级功能总开关的前提下，以下两行的注释或者保留可以进一步控制是否支持 SD 卡升级或者 U 盘升级

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

默认 SD 卡/U 盘同时支持，两者共存时的升级策略将在 3.1 “Auto Update 详细设计”中进一步说明。

2.1.2 模块结构

下面以 Hi3536 代码为例，SD 卡升级以 Hi3516A 代码为例，说明升级模块中各个子模块的代码路径，其他芯片平台仅需将 Hi3536 或 Hi3516A 的相关代码改为相应平台的代码即可。

- 黑色部分为需要修改的文件
- 红色部分为新增文件

```
----编译
Makefile
---- 配置
board/hi3536/board.c          【程序入口】
include/configs/hi3536.h或者include/configs/hi3536_spinand.h    【编译开关】
----- SD驱动
drivers/mmc/himciv100_3516a.c
drivers/mmc/himciv100.c
drivers/mmc/himciv100.h
drivers/mmc/mmc.c
-----USB驱动
drivers/usb/host/hiusb/hiusb-3536.c
drivers/usb/host/hiusb/hiusb-ohci.c
drivers/usb/host/hiusb/hiusb-ohci.h
drivers/usb/host/hiusb/hiusb-xhci-3536.c
drivers/usb/host/hiusb/xhci.c
drivers/usb/host/hiusb/xhci.h
drivers/usb/host/hiusb/xhci-mem.c
drivers/usb/host/hiusb/xhci-ring.c
drivers/usb/host/hiusb/Makefile
---- update主控流程
product/hiupdate/Makefile
product/hiupdate/auto_update.c
```

product/hiupdate/mmc_init.c
product/hiupdate/usb_init.c

2.2 具体实现

2.2.1 FAT 文件系统支持

在芯片配置文件中增加 FAT 相关的编译开关即可，红色部分为相关修改

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

2.2.2 MMC 驱动移植

2.2.2.1 编译开关

在芯片配置文件中增加 MMC 驱动编译开关，相关修改如下：

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

说明
MMC 驱动编译开关 CONFIG_AUTO_SD_UPDATE 受控于升级功能总开关 CONFIG_AUTO_UPDATE

2.2.2.2 代码移植

下面以 Hi3516A 代码为例，说明 MMC 驱动模块相关的代码路径

- 黑色部分为修改的文件
- 红色部分为新增文件

```
drivers/mmc/himciv100.c
drivers/mmc/himciv100.h
drivers/mmc/mmc.c
drivers/mmc/himciv100_3516a.c
```

其他平台对应代码：

```
drivers/mmc/himciv100_3518.c        【hi3518a、3518e】
drivers/mmc/himciv100_godnet.c      【hi3531】
drivers/mmc/himciv100_godarm.c      【hi3521】
drivers/mmc/himciv200.c
drivers/mmc/himciv200.h
drivers/mmc/himciv200_hi3518ev200.c     【hi3518ev200】
drivers/mmc/himciv200_hi3518ev201.c     【hi3518ev201】
drivers/mmc/himciv200_hi3516cv200.c     【hi3516cv200】
drivers/mmc/himciv200_hi3519.c          【hi3519v100】
drivers/mmc/himciv200_hi3519v101.c      【hi3519v101】
drivers/mmc/himciv200_hi3516cv300.c     【hi3516cv300】
```

说明

Hi3520D、Hi3515A、Hi3535、Hi3536、Hi3531A 不支持 SD 卡，故不支持从 SD 卡升级功能。

2.2.3 使能 USB OHCI

2.2.3.1 编译开关

在芯片配置文件中增加 USB OHCI 驱动编译开关，相关修改如下：

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

#define CONFIG_SYS_USB_XHCI_MAX_ROOT_PORTS      2
```

说明

USB OHCI 驱动默认直接编译到 U-boot 中，不受升级总开关 CONFIG_AUTO_UPDATE 控制。

2.2.3.2 代码移植

下面以 Hi3536 代码为例，说明 USB OHCI 和 USB XHCI 驱动模块相关的代码路径

红色部分为新增文件：

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

其他平台对应代码：

```
drivers/usb/host/hiusb/hiusb-3520d.c        【Hi3520a、Hi3520d、Hi3515a】
drivers/usb/host/hiusb/hiusb-godarm.c       【Hi3521】
drivers/usb/host/hiusb/hiusb-godnet.c       【Hi3531】
drivers/usb/host/hiusb/hiusb-3518.c         【Hi3518a、Hi3518c】
drivers/usb/host/hiusb/hiusb-3535.c         【Hi3535】
drivers/usb/host/hiusb/hiusb-3516a.c        【Hi3516A】
drivers/usb/host/hiusb/hiusb-3521a.c        【Hi3521A】
drivers/usb/host/hiusb/hiusb-3518ev200.c    【Hi3518ev200】
drivers/usb/host/hiusb/hiusb-3518ev201.c    【Hi3518ev201】
drivers/usb/host/hiusb/hiusb-3516cv200.c    【Hi3516cv200】
drivers/usb/host/hiusb/hiusb-3531a.c        【Hi3531a】
drivers/usb/host/hiusb/hiusb-3519.c         【Hi3519v100、Hi3519V101】
drivers/usb/host/hiusb/hiusb-3516cv300.c    【hi3516cv300】
```

2.2.4 手动升级验证移植的 MMC 驱动

完成“2.2.1 FAT 文件系统支持”和“2.2.2 MMC 驱动移植”后，编译生成的 U-boot 支持 mmcinfo mmc 两个操作 SD 卡的相关命令。同理完成“2.2.1 FAT 文件系统支持”和“2.2.3 使能 USB OHCI”后可以进行从 U 盘升级以验证所移植的 USB OHCI 驱动是否正确。

2.2.4.1 从 SD 卡中手动升级系统步骤

下面以升级内核为例，列举从 SD 卡手动升级系统流程，其他模块升级只需要下载相应的升级包到内存，烧写到 Flash 上的相应的位置即可。

步骤 1. 烧写移植完 MMC 驱动的 uboot 版本或者直接从内存中启动此 uboot。

步骤 2. 插入格式化成 FAT32 并存放有升级镜像的 SD 卡，升级包制作参见《U-boot 下 U 盘 SD 卡自动升级使用手册和移植说明.doc》。

步骤 3. 在 U-boot 下键入 mmcinfo 命令，获取 SD 卡信息，初始化 SD 卡。

步骤 4. 键入 fatls mmc 0 命令，查看 SD 卡中第一个分区的升级镜像。

步骤 5. 键入 fatload mmc 0 0x82000000 kernel 将 kernel 镜像下载到内存 0x82000000 处。

步骤 6. sf probe 0; sf erase 100000 400000; sf write 0x82000000 100000 400000;将内核烧写到 SPI【烧写位置可以自行决定】或者直接 bootm 0x82000000 启动新内核。

步骤 7. 同样的方法烧写 u-boot 和 rootfs。

步骤 8. 根据烧写的位置，设置环境变量，保存。

步骤 9. 启动系统，查看内核或者其他升级模块启动时打印的编译时间可验证相应模块是否升级成功。

----结束

2.2.4.2 详细操作

- 启动移植好 MMC 驱动的 U-boot。
- 插上存放有升级包的 FAT 格式 SD 卡。
- 键入 mmcinfo 命令。

  若 MMC 驱动移植成功可以看到类似如下的打印信息：

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

- 键入 **fatls mmc 0** 命令，查看 SD 卡中升级镜像。

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

- 键入 **fatload mmc 0 0x82000000 kernel**，将 kernel 镜像下载到内存。

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

- 键入 **bootm 0x82000000** 直接从内存启动新内核。

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

若出现类似以上启动内核的打印，则说明从 SD 卡中读取的数据正确，MMC 驱动移植成功。

2.2.5 手动升级验证移植的 USB OHCI 驱动

2.2.5.1 从 U 盘中手动升级系统步骤（以 Hi3536 为例）

步骤 1. 烧写移植好 usb 驱动的 uboot 版本或者直接从内存启动此 uboot。

步骤 2. 插入格式化成 FAT32 存放有升级镜像的 U 盘，升级包制作参见《U-boot 下 U 盘 SD 卡自动升级使用手册和移植说明.doc》。

步骤 3. 键入 **usb start** 命令，初始化 USB 存储设备。

步骤 4. 键入 **fatls usb 0** 命令，查看 U 盘中的升级镜像。

步骤 5. 键入 **fatload mmc 0 0x42000000 kernel**，将 kernel 镜像下载到内存。

步骤 6. sf probe 0;sf erase 100000 600000;sf write 0x42000000 100000 600000;将内核烧写到 SPI（烧写位置可以自行决定）或者直接 bootm 0x42000000 启动新内核。

步骤 7. 同样的方法烧写 u-boot 和 rootfs。

步骤 8. 根据烧写的位置，设置环境变量，保存。

步骤 9. 启动系统，查看内核或者其他升级模块启动时打印的编译时间可验证相应模块是否升级成功。

----结束

2.2.5.2 详细操作（以 Hi3536 为例）

- 启动移植好 USB OHCI 驱动的 U-boot。
- 插上存放有升级包的 FAT 格式 U 盘。
- 键入 usb start 命令。

  若 USB OHCI 驱动移植成功可以看到类似如下的打印信息：

```
hisilicon # usb start
    (Re)start USB...
```

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

- 键入 fatls usb 0 命令，查看 SD 卡中升级镜像。

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

- 键入 **fatload usb 0 0x42000000 kernel**，将 kernel 镜像下载到内存。

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

- 键入 **bootm 0x42000000** 直接从内存启动新内核。

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

若出现类似以上启动内核的打印，则说明从 SD 卡中读取的数据正确，MMC 驱动移植成功。

2.2.6 增加自动升级入口

以下内容以 Hi3536 为样例，增加自动升级入口函数调用。其他芯片平台只需要将 Hi3536 替换成相应路径即可。

在文件 board/hi3536/board.c 中增加自动升级入口，红色为修改部分，自动升级详细设计请参见“3 详细设计。”

```
int misc_init_r(void)
{
#ifdef CONFIG_RANDOM_ETHADDR
       random_init_r();
#endif
       setenv("verify", "n");
#ifdef CONFIG_AUTO_UPDATE
       extern int do_auto_update(void);
#ifdef CFG_MMU_HANDLEOK
       dcache_stop();
```

```
#endif
        do_auto_update();    //U-boot启动后自动升级入口函数调用
#ifdef CFG_MMU_HANDLEOK
        dcache_start();
#endif
#endif /* CONFIG_AUTO_UPDATE */
        return 0;
}
```

3 详细设计

3.1 Auto Update 详细设计

3.1.1 数据描述

3.1.1.1 简单数据描述

驱动部分只做了基本功能的移植，所以下面实例中所涉及的文件只包括 boot、kernel 和 rootfs。

a. 在开始介绍 auto update 的实现之前需要了解 include/image.h 中的如下定义：

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

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

各种 Image Type 在制作镜像时通过 mkimage 工具的’-T’参数去指定，常用的几个径向的对应关系分别为：

类别	对应关系
u-boot.bin	IH_TYPE_FIRMWARE（对应 mkimage –T firmware）
kernel	IH_TYPE_KERNEL（对应 mkimage –T kernel）
rootfs	IH_TYPE_FILESYSTEM（对应 mkimage –T filesystem）

b. 需要更新的镜像的索引关系及总数通过如下代码定义。

```
/* index of each file in the following arrays */
#define IDX_FIRMWARE 0
#define IDX_KERNEL 1
#define IDX_ROOTFS 2
/* max. number of files which could interest us */
#define AU_MAXFILES 3
```

c. 需要更新的镜像的名称及其所应关系。

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

需要注意的是这里的文件名与 U 盘上存储的文件名一一对应，设备枚举出来之后 auto update 程序会通过文件名去检索相关的镜像。

d. 各个镜像在 Flash 上的存储位置。

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

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

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

e. 要用到的内存的起始地址

```
/* where to load files into memory */
#define LOAD_ADDR ((unsigned char *)0x82000000)
```

```
/* the app is the largest image */
#define MAX_LOADSZ ausize[IDX_ROOTFS]
```

需要说明的是 U 盘上的镜像更新到 flash 上的步骤：

步骤 1. 把 SD 卡/U 盘上的镜像通过文件系统 load 到内存上。

步骤 2. 从内存写到 SPI flash 上。

----结束

因此需要这个内存起始地址。

3.1.1.2 校验信息

每个升级所用的镜像的开头几个字节，都必须包含如下信息：（包括魔术字、crc 校验、文件类型、文件大小等等）

```
#define IH_MAGIC    0x27051956 /*Image Magic Number*/
#define IH_NMLEN    32 /* Image Name Length */
typedef struct image_header {
uint32_t    ih_magic;   /* Image Header Magic Number    */
uint32_t    ih_hcrc;    /* Image Header CRC Checksum    */
uint32_t    ih_time;    /* Image Creation Timestamp */
uint32_t     ih_size;     /* Image Data Size    */
uint32_t     ih_load;   /* Data Load Address    */
uint32_t     ih_ep;   /* Entry Point Address    */
uint32_t     ih_dcrc;       /* Image Data CRC Checksum */
uint8_t ih_os;  /* Operating System */
uint8_t ih_arch;     /* CPU architecture       */
uint8_t ih_type;    /* Image Type   */
uint8_t ih_comp;        /* Compression Type     */
uint8_t ih_name[IH_NMLEN];  /* Image Name   */
} image_header_t;
```

如果没有这些信息，或者某个信息不匹配，将不会更新该镜像。这些信息都是通过前面提到过的 mkimage 工具写入的。

3.1.2 函数描述

3.1.2.1 所用到的外部函数实现

FAT 文件系统相关函数

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

实现了注册 FAT 设备，通过文件名读取相应的文件等接口。

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

块设备接口，在 auto update 中默认选择的是第一个分区。

3.1.2.2 内部函数实现

在 auto update 中实现的函数主要有 4 个：（其中每一部分都会进行各自的 crc 校验，如果有一处校验不过，就会返回错误并中止升级。）

- int au_check_cksum_valid(int idx, long nbytes)

  获取当前文件的总大小，把它与 image_header 中记录的总大小比较，如果不匹配则返回错 误，并中止升级。

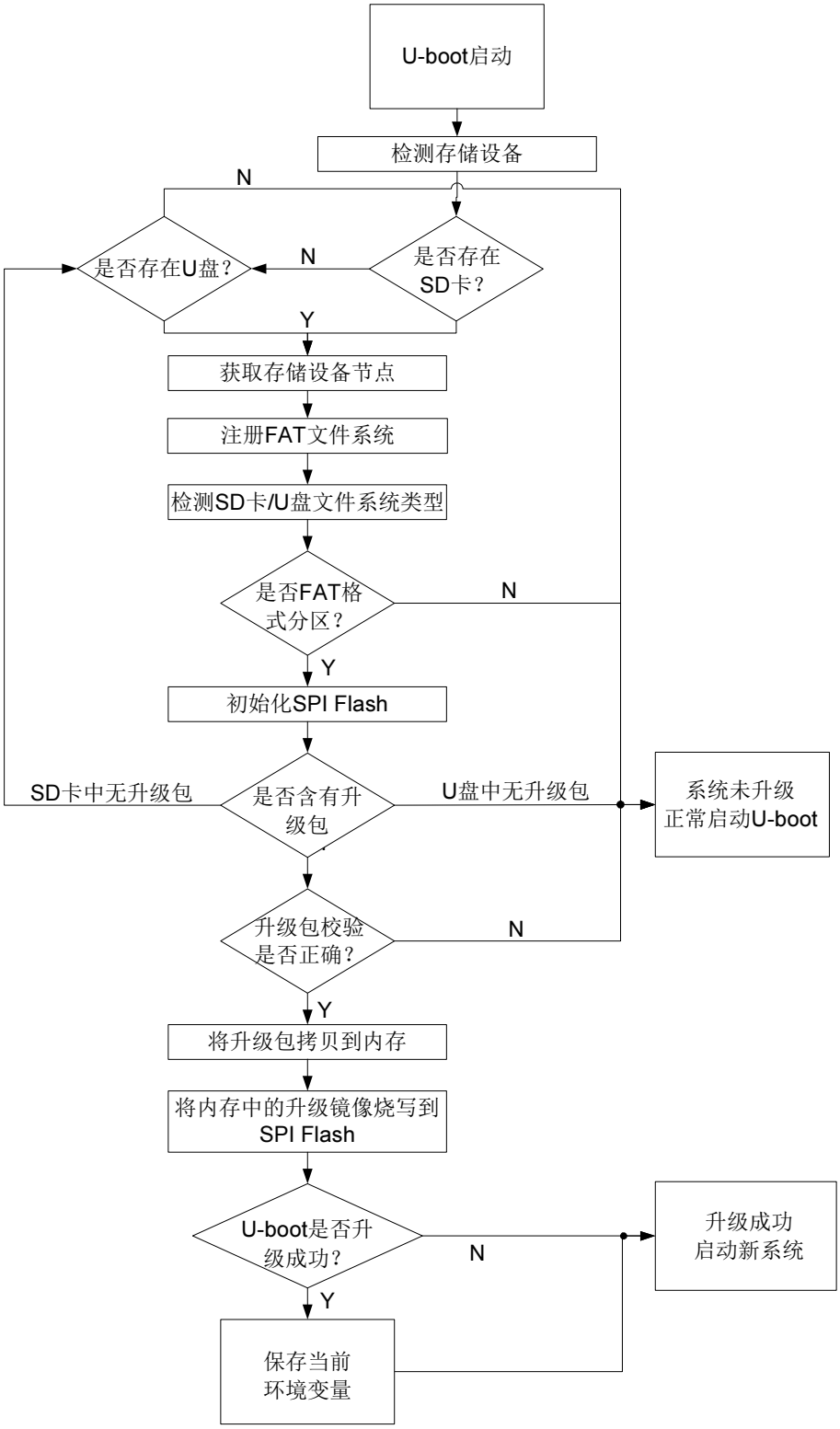
- int au_check_header_valid(int idx, long nbytes)
  - 这个函数的主要功能就是校验镜像文件的 image_header 信息，如果校验不通过则不会进行升级。
  - 如果校验通过，这个函数还有一个重要的功能就是对比 ih_time 这个时间戳信息，用它来判断镜像文件的版本。如果镜像文件的生成时间小于已记录的时间之前，那么该镜像就不会被 down 到 flash 上。当满足信息的镜像更新到 flash 上之后就会在 boot 的环境变量中保存当前镜像的时间戳信息用于下一次比较。
- int au_do_update(int idx, long sz)

  这个函数的主要功能就是写 flash，当所有的校验通过之后，就会通过这个函数把镜像写到 flash 上。

- int do_auto_update(void)

  提供给外部的函数接口，是 auto update 的主函数。函数运行过程如图 3-1 所示。

图3-1 函数运行过程图



以上函数的具体实现可以参考 U-boot 下 product/hiupdate/auto_update.c