UBI 文件系统使用指南

文档版本　02

发布日期　2016-01-04

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

linux-2.6.27 后，内核加入了一种新型的 FLASH 文件系统 UBI (Unsorted Block Images)。主要针对 FLASH 的特有属性，通过软件的方式来实现日志管理、坏块管理、损益均衡等技术。

本文主要介绍如何在内核中配置使用 UBI 文件系统以及制作对应的 UBI 文件系统根文件系统镜像。同时还介绍了如何转换镜像格式以便于在 u-boot 上进行烧录。

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，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
Hi3516A	V100
Hi3516D	V100
Hi3518E	V200
Hi3518E	V201
Hi3516C	V200

读者对象

本文档（本指南）主要适用于技术支持工程师。

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。

文档版本 02 (2016-01-04)

第 2 次正式发布

产品版本中添加 Hi3518EV20X 及 Hi3516CV200。

文档版本 01 (2015-02-09)

第 1 次正式发布。

目 录

1 内核中使能 UBI

当前单板上使用的内核版本是 linux 3.4.35，使用 UBIFS 文件系统,可以按以下配置。

1.1 内核配置 UBI 选项

步骤 1. 使能 UBI 设备驱动

```
Device Drivers  --->
<*> Memory Technology Device (MTD) support  --->
UBI - Unsorted block images  --->
<*> Enable UBI
```

使能 UBI 设备驱动后默认选项如下：

```
--- Enable UBI - Unsorted block images
(4096) UBI wear-leveling threshold
(1)    Percentage of reserved eraseblocks for bad eraseblocks handling
< >    MTD devices emulation driver (gluebi)
[ ]    UBI debugging
```

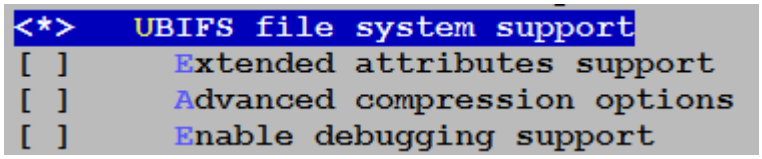
说明

必须先使能 UBI 设备驱动，才能找到 UBIFS 文件系统选项。

步骤 2. 使能 UBIFS 文件系统

```
File systems  --->
    -*- Miscellaneous filesystems  --->
       <*>  UBIFS file system support
```

使能 UBIFS 文件系统后默认选项如下：



说明

所有配置按以上图中所示，其它 UBI/UBIFS 配置选项使用系统默认值，不要随意选择配置，如果选择不慎, UBI 文件系统可能无法正常工作。

----结束

1.2 UBI 设备驱动配置选项说明

- UBI wear-leveling threshold

  UBI 系统记录每个擦除块发生擦除操作的次数。此选项表示所有擦除操作次数中，最小值和最大值之间允许的最大间隔。此值默认为 4096，对于寿命比较短的 MLC 器件，此值应该配置相对小一点，比如 256。

- MTD devices emulation driver (gluebi)

  模拟 MTD 驱动，选择此选项，当创建一个卷时，UBI 将同时模拟一个 MTD 设备。这个功能提供了一个接口,供其它文件系统使用 UBI。

- UBI debugging

  UBIFS 调试功能，是 UBIFS 的作者测试文件系统所使用，通常用户用不到。

  配置完成后正常编译内核，会在 /dev/ 目录下生成 ubi_ctrl 文件，用来将 ubi 绑定到 mtd 上，生成 ubi 设备。第一个绑定的设备为 ubi0，第二个绑定的为 ubi2，以此类推，设备 ID 仅与绑定先后顺序有关。

2 UBIFS 应用样例

2.1 mount 一个空 UBIFS 文件系统

单板当前有 3 个分区，分区的情况如下图。

```
# cat /proc/mtd
dev:    size   erasesize  name
mtd0: 01000000 00040000 "hi_sfc"
mtd1: 00a00000 00020000 "boot"
mtd2: 03200000 00020000 "nand0"
```

通过以下几步，就可以把 mtd2 分区映射成 ubi 卷，做为 ubi 分区使用。

步骤 1. 格式化 ubi 分区

使用以下命令格式化 ubi 分区。

```
# ubiformat /dev/mtd2
```

说明

不推荐擦除(如:用命令 flash_eraseall)分区，擦除分区后，可以正常 mount 到 ubifs。 但是擦除分区操作，将使 UBI 系统丢失记录的每个擦除块的擦除次数。

步骤 2. 绑定 UBI 到 MTD 分区

绑定 ubi 到 mtd2 分区，使用以下命令。

```
# ubiattach /dev/ubi_ctrl -m 2
```

参数”-m 2”表示使用 mtd2 分区。只有绑定了 ubi 到 mtd 分区以后，才能在 /dev/ 下找到 ubi 设备“ubi0”，如果曾经创建过 ubi 卷，那么绑定以后才能在 /dev/ 下找到并且访问 ubi 卷 “ubi0_0”。

命令执行成功，显示信息如下图。

```
# ubiattach /dev/ubi_ctrl -m 2
UBI: attaching mtd2 to ubi1
UBI: physical eraseblock size:   131072 bytes (128 KiB)
UBI: logical eraseblock size:    126976 bytes
UBI: smallest flash I/O unit:    2048
UBI: VID header offset:          2048 (aligned 2048)
UBI: data offset:               4096
UBI: attached mtd2 to ubi1
UBI: MTD device name:            "nand0"
UBI: MTD device size:            50 MiB
UBI: number of good PEBs:        393
UBI: number of bad PEBs:         7
UBI: max. allowed volumes:       128
UBI: wear-leveling threshold:    4096
UBI: number of internal volumes: 1
UBI: number of user volumes:     0
UBI: available PEBs:             386
UBI: total number of reserved PEBs: 7
UBI: number of PEBs reserved for bad PEB handling: 3
UBI: max/mean erase counter: 5/2
UBI: image sequence number: 0
UBI: background thread "ubi_bgt1d" started, PID 358
UBI device number 1, total 393 LEBs (49901568 bytes, 47.6 MiB),
available 386 LEBs (49012736 bytes, 46.7 MiB), LEB size 126976 bytes
(124.0 KiB)
```

最后一行打印“UBI device number 1”表示成功创建设备 ubi1，查看所有设备“ls /dev/ubi*”，将发现多一个设备“/dev/ubi1”。

步骤 3. 创建 UBI 卷

UBI 卷可以理解为对 UBI 设备的进行分区。创建 ubi 卷命令如下：

```
# ubimkvol /dev/ubi1 -N ubifs -m
```

参数“/dev/ubi1”是上一步骤创建的 ubi 设备。

参数“-N ubifs”表示创建的卷名为“ubifs”。

参数“-m”表示使用全部空间，也就是“/dev/ubi1”设备能提供的空间，即 UBI1 绑定的 MTD2 大小，在这里是 50M。

创建成功，将显示如下信息：

```
# ubimkvol /dev/ubi1 -N ubifs -m
Set volume size to 49012736
Volume ID 0, size 386 LEBs (49012736 bytes, 46.7 MiB), LEB size 126976
bytes (124.0 KiB), dynamic, name "ubifs", alignment 1
```

打印“Volume ID 0”表示成功创建了 ubi 卷 0，查看所有设备 “ls /dev/ubi*”，将发现多一个设备“/dev/ubi1_0”。

说明

卷只用创建一次. 创建后，卷信息将被记录在 UBI 设备上，下一次启动，不用再次创建卷。删除卷，用命令 “ubirmvol”. 如果使用此命令删除卷，卷上所有数据，也将被删除。

步骤 4. 挂载空 UBIFS 文件系统

此时就可以将创建的卷挂载到指定的目录上去了，命令如下：

```
# mount -t ubifs /dev/ubi1_0 /mnt/
```

或者

```
# mount -t ubifs ubi1:ubifs /mnt/
```

参数“/dev/ubi1_0”表示 mount 到卷 “ubi1_0”，也可以使用参数“ubi1:ubifs”。某些版本的内核，不支持“/dev/ubi1_0”形式的参数，只能使用“ubi1:ubifs”形式的参数. “ubi1:ubifs”中的“ubifs”表示卷的名称，在创建 ubi 卷时设置。

Mount 成功，将显示如下信息：

```
# mount -t ubifs /dev/ubi1_0 mtd
UBIFS: default file-system created
UBIFS: mounted UBI device 1, volume 0, name "ubifs"
UBIFS: file system size:   47742976 bytes (46624 KiB, 45 MiB, 376
LEBs)
UBIFS: journal size:      2412544 bytes (2356 KiB, 2 MiB, 19 LEBs)
UBIFS: media format:      w4/r0 (latest is w4/r0)
UBIFS: default compressor: lzo
UBIFS: reserved for root:  2255018 bytes (2202 KiB)
```

查看分区信息,将显示如下内容：

```
# df -h
Filesystem              Size     Used Available Use% Mounted on
ubi0:rootfs            37.4M      2.5M     34.9M   7% /
tmpfs                  37.2M     4.0K     37.2M   0% /dev
/dev/ubi1_0            41.4M     24.0K     39.3M   0% /root/mtd
```

UBIFS 文件系统显示的分区大小、剩余空间并不准确。因为 UBIFS 文件保存的是文件压缩后的内容，压缩比率与文件内容相关。可能剩余空间显示只有 2M，但是可以将一个 4M 的文件完整保存。

----结束

2.2 制作 UBIFS 根文件系统 UBI 镜像

制作 ubifs 文件系统镜像，需要使用 mtd-utils 工具，命令如下：

```
mkfs.ubifs -d rootfs_uclibc -m 2KiB -o rootfs.ubiimg -e 126976 -c 256
–F -v
```

参数“-d rootfs_uclibc”表示将要被制作为 UBIFS 镜像的根目录为“rootfs_uclibc”，这个参数也可以写为 “-r rootfs_uclibc”。

参数“-m 2KiB”表示最小读写单元是 2KiB，这个参数也可以写为“-m 2048”。这里使用的 NAND 芯片页大小为 2KiB。最小读写单元是指 FLASH 器件一次读写操作，最小操作的字节数，对 NAND 器件，是页大小，如 2K/4K/8K；对于 NOR 器件，是 1 个字节。

参数“-o rootfs.ubiimg”表示制作出来的镜像名称为“rootfs.ubiimg”。

参数 “-e 126976”表示逻辑擦除块大小。

最小读写单元和逻辑擦除块大小可以通过读 MTD 和 UBI 系统信息获得，也可以通过计算获得。读 MTD 信息命令以及显示内容如下：

```
# ./mtdinfo /dev/mtd4
mtd4
Name:                          reserve
Type:                          nand
Eraseblock size:               131072 bytes, 128.0 KiB
Amount of eraseblocks:         1024 (134217728 bytes, 128.0 MiB)
Minimum input/output unit size: 2048 bytes
Sub-page size:                 2048 bytes
OOB size:                      60 bytes
Character device major/minor:  90:8
Bad blocks are allowed:        true
Device is writable:            true
```

读 UBI 信息命令（此命令需要先绑定 UBI 见 2.1.3）以及显示内容如下：

```
# ubinfo /dev/ubi0
ubi0
Volumes count:                       1
Logical eraseblock size:             126976 bytes, 124.0 KiB
Total amount of logical eraseblocks:    398 (50536448 bytes, 48.2
MiB)
Amount of available logical eraseblocks: 0 (0 bytes)
Maximum count of volumes             128
Count of bad physical eraseblocks:      2
Count of reserved physical eraseblocks:  3
Current maximum erase counter value:    2
Minimum input/output unit size:         2048 bytes
Character device major/minor:           252:0
Present volumes:                     0
```

参数“-c 256”表示此文件系统最多使用“256”个逻辑擦除块。计算“256 * 2KiB”得到此文件系统的最大可使用空间。

参数“-F”使能"white-space-fixup"，如果是通过 u-boot 烧写需要使能此功能。

参数“-v”显示制作 UBIFS 过程中的详细信息。

以上标记为红色的，表示此芯片的逻辑擦除块大小。

逻辑擦除块大小也可以通过计算得到，计算方法如下表：

FLASH 种类	逻辑擦除块(LEB)大小
NOR	LEB = blocksize – 128

FLASH 种类	逻辑擦除块(LEB)大小
NAND 无子页	LEB = blocksize – pagesize * 2
NAND 有子页	LEB = blocksize – pagesize * 1
说明: Blocksize: flash 物理擦除块大小; Pagesize: flash 读写页大小;	

下图为制作成功后, “mkfs.ubifs”的详细打印。

```
# mkfs.ubifs -d rootfs_uclibc -m 2KiB -o rootfs. ubiimg -e 126976 -c
256 -v
mkfs.ubifs
        root:        rootfs_uclibc/
        min_io_size:  2048
        leb_size:    126976
        max_leb_cnt:  256
        output:      rootfs.ubiimg
        jrn_size:    3936256
        reserved:    0
        compr:       lzo
        keyhash:     r5
        fanout:      8
        orph_lebs:   1
        space_fixup:  0
        super lebs:   1
        master lebs:  2
        log_lebs:    4
        lpt_lebs:    2
        orph_lebs:   1
        main_lebs:   45
        gc lebs:     1
        index lebs:   1
        leb_cnt:     55
```

这里需要注意, 制作成功的 UBIFS 根文件系统镜像为 UBI 镜像，可以在内核下对空 UBIFS 文件系统进行升级（update）操作，详见 2.3 空 UBIFS 文件系统升级为根文件系统。

该镜像不能直接烧录到 MTD 分区上使用，但是可以通过格式转换,转换成能直接烧录 MTD 分区上的格式。详见 2.4 UBI 镜像的转换格式和烧录。

说明

做 UBI 镜像需要专用工具和内核版本必须搭配使用，当前单板内核版本: linux 3.4.35, UBI 工具版本: zlib-1.2.5, lzo-2.03, ubi-utils-1.5.0. UBI 工具版本号和单板内核版本如果不配套,制作的镜像无法在单板上 mount.

2.3 空 UBIFS 文件系统升级为根文件系统

在内核(区别 u-boot.bin)下建立好 UBI 卷后，可以使用应用程序，直接对卷进行升级。升级步骤如下：

步骤 1. 创建建立 UBI 卷

详见 2.1.3 创建 UBI 卷

步骤 2. 制作 UBIFS 根文件系统 UBI 镜像

详见 2.2 制作 UBIFS 根文件系统 UBI 镜像。

步骤 3. tftp 下载根文件系统 UBI 镜像到内核

```
# tftp -g -r rootfs.ubiimg 10.67.209.140
```

步骤 4. 在内核下升级 UBIFS 文件系统

使用以下命令：

```
# ubiupdatevol /dev/ubi1_0  rootfs.ubiimg
```

参数“/dev/ubi1_0”表示需要升级的卷，这个卷需要预先创建，升级前，卷上的内容可以不用擦除。

清除卷内容，使用命令“ubiupdatevol /dev/ubi1_0 -t” 。

步骤 5. 设置 u-boot.bin 启动参数

UBIFS 下 u-boot.bin 的启动参数 bootargs 配置形式如下图：

```
setenv bootargs 'mem=64M console=ttyAMA0,115200 ubi.mtd=3
root=ubi0:ubifs rootfstype=ubifs rw
mtdparts=hinand:1M(boot),3M(kernel),60M(yaffs2),64M(ubi),-
(reserve)'
```

参数“ubi.mtd=3”表示 UBI 绑定到“/dev/mtd3”分区。

参数“root=ubi0:ubifs”中“ubi0”表示使用 UBI 绑定后的 UBI 分区，其中“ubifs”为创建 UBI 卷时定义的卷名，某些内核版本不识别“root=/dev/ubi1_0”形式的参数。

参数“rootfstype=ubifs”表示使用 ubifs 文件.

----结束

2.4 UBI 镜像的转换格式和烧录

步骤 1. 制作 UBIFS 根文件系统 UBI 镜像

详见 2.2 制作 UBIFS 根文件系统 UBI 镜像。

步骤 2. 制作 UBI 镜像转换配置文件

UBI 镜像转换格式时，需要一个配置文件 ubi.cfg 作为第（3）步的输入。内容如下图所示：

```
[ubifs-volumn]
mode=ubi
image=./rootfs_hi3516a_2k_128k_32M.ubiimg
vol_id=0
vol_type=dynamic
vol_alignment=1
vol_name=ubifs
vol_flags=autoresize
```

参数“mode=ubi”是强制参数，当前不能输入别的值，保留为以后扩展功能；

参数“image=./rootfs*.img”表示卷对应的 UBIFS 文件系统镜像文件名称，此文件即 2.2 制作 UBIFS 根文件系统 UBI 镜像制作的镜像文件。

参数“vol_id=0”表示卷的 ID 号, UBI 镜像可能包含多个卷，这个用来区别不同的卷。

参数“vol_type=dynamic”表示当前卷类型是可读写的。如果此文件为只读,对应的参数应该为“vol_type=static”；

参数“vol_name=ubifs”表示卷的名称, UBIFS 做根文件系统时，将用到卷名称。

参数“vol_flags=autoresize”表示卷大小是可以动态扩展。

步骤 3. 转换 UBI 镜像格式

使用以下命令：

```
ubinize -o rootfs.img -m 2KiB -p 128KiB ubi.cfg -v
```

参数“-o rootfs.img”表示输出的 UBI 镜像转换后的名称为“rootfs.img”,输入的 UBI 镜像文件名由 ubi.cfg 配置文件输入。

参数“-m 2KiB”表示最小读写单元是“2KiB” 。

参数“-p 128KiB”表示 flash 的擦除块大小. 注意这是物理擦除大小，不是逻辑擦除块大小。

参数“ubi.cfg”是一个配置文件，第（2）步已经详细讲解过此文件。

参数“-v”显示制件过程的详细信息。

下图为制作成功后的详细打印。

```
$ ./ubinize -o rootfs.img -m 2KiB -p 128KiB ubi.cfg -v
ubinize: LEB size:               126976
ubinize: PEB size:               131072
ubinize: min. I/O size:           2048
ubinize: sub-page size:           2048
ubinize: VID offset:             2048
ubinize: data offset:             4096
ubinize: UBI image sequence number: 2067745235
ubinize: loaded the ini-file "ubi.cfg"
ubinize: count of sections: 1
ubinize: parsing section "ubifs-volumn"
ubinize: mode=ubi, keep parsing
ubinize: volume type: dynamic
ubinize: volume ID: 0
ubinize: volume size was not specified in section "ubifs-volumn",
assume minimum to fit image
"./rootfs_hi3516a_2k_128k_32M.ubiimg"6983680 bytes (6.7 MiB)
ubinize: volume name: ubifs
ubinize: volume alignment: 1
ubinize: autoresize flags found
ubinize: adding volume 0
ubinize: writing volume 0
ubinize: image file: ./rootfs_hi3516a_2k_128k_32M.ubiimg
ubinize: writing layout volume
ubinize: done
```

步骤 4. U-BOOT 下 ubi 镜像转换文件的烧写

U-BOOT 下，烧写 UBI 镜像转换文件和烧写内核的方法一样．命令如下图所示：

```
hisilicon # nand erase 0x4000000 0x720000
hisilicon # tftp 0x82000000 rootfs.img
hisilicon # nand write 0x82000000 0x4000000 0x720000
```

说明

写数据到 NAND 时，长度一定要是 UBI 镜像转换文件本身大小长度，多输或少输，都将造成文件系统错误。

----结束

3 附录

3.1 UBI 和 MTD 相关的接口和命令

(1) cat /proc/mtd	可以看到当前系统的各个 mtd 情况。
(2) mtdinfo /dev/mtd3	查看 MTD 分区信息。
(3) ubinfo –a	显示当前所有 UBI 分区信息
(4) ls /dev/ubi*	查看 UBI 设备节点和 UBI 卷

3.2 UBI 常见问题

a. 空 FLASH 运行 UBI 前，必须要先用 erase 命令擦除么?

没有必要,UBI 文件系统会记录每个块的使用次数,如果 ERASE,将会把这些使用次数也擦除掉。这将破坏 UBI 的读写均衡特性。使用 ubiformat 程序，能擦除 flash，并且保存每个块的读写次数。

b. 为什么第一次能 mount 上 ubifs，重启以后会挂载失败?

在内核版本在 3.0 以上，需要在制作 UBIFS 根文件系统 UBI 镜像的时候，增加“-F”参数，这个参数会设置修正空白区域的标志，在第一次 mount 的时候对空白区域进行修正，以便后续能正常使用。