UBIFS User Guide

Issue	**02**
Date	**2016-01-04**

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

For Linux 2.6.27 and later versions, a new flash file system unsorted block image (UBI) is added to the kernel. Aiming at the distinctive attributes of the flash, the UBI implements technologies including log management, bad block management as well as profit and loss balancing by using software.

This document describes how to configure and use the UBI file system in the kernel, how to create the root file system image of the UBI file system, and how to convert the image format so that the UBI can be burnt under the U-boot.

NOTE

- Unless otherwise specified, the contents of the Hi3516A also apply to the Hi3516D
- Unless otherwise specified, the contents of Hi3518E V200 also apply to Hi3518E V201 and Hi3516C V200.

Related Versions

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

Product Name	Version
Hi3516A	V100
Hi3516D	V100
Hi3518E	V200
Hi3518E	V201
Hi3516C	V200

Intended Audience

This document is intended for field application engineers.

Change History

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

Issue 02 (2016-01-04)

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

The products Hi3518E V20X and Hi3516C V200 are added.

Issue 01 (2015-02-09)

This issue is the first official release.

Contents

1 UBI Configuration in the Kernel

The version of the kernel used on the current board is Linux kernel 3.4.35. The following sections describe how to configure the unsorted block image file system (UBIFS).

1.1 Configuring UBI Configuration Options in the Kernel

Perform the following step:

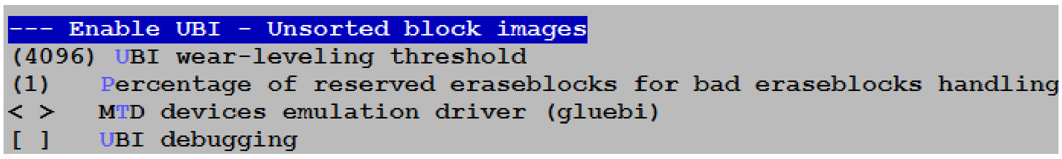
Step 1 Enable the UBI drivers, as shown in Figure 1-1.

Figure 1-1 Enabling the UBI drivers

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

Figure 1-2 shows the default configuration options after the UBI drivers are enabled.

Figure 1-2 Default configuration options after the UBI drivers are enabled



```
--- 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
```

Note that the UBIFS configuration options are displayed only after the UBI drivers are enabled.

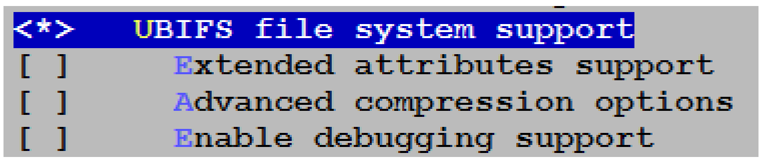
Step 2 Enable the UBIFS, as shown in Figure 1-3.

Figure 1-3 Enabling the UBIFS

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

Figure 1-4 shows the default configuration options after the UBIFS is enabled.

Figure 1-4 Default configuration options after the UBIFS is enabled



```
<*>    UBIFS file system support
[ ]      Extended attributes support
[ ]      Advanced compression options
[ ]      Enable debugging support
```

NOTE

The UBI configuration options must be configured by following the preceding figures. For other UBI/UBIFS configuration options, the default values are used. You are advised not to change the values of these configuration options; otherwise, the UBIFS may fail to work properly.

1.2 Configuration Options of UBI Drivers

Note the following configuration options of UBI drivers:

- UBI wear-leveling threshold

  The UBIFS records the number of erase operations for each erase block. This configuration option indicates the maximum difference between the minimum number of erase operations and the maximum number of erase operations. The default value is 4096. For the multi-level cell (MLC) components with short life cycles, a smaller value such as 256 is recommended.
- MTD devices emulation driver (gluebi)

  Analog memory technology device (MTD) driver. If this option is selected, the UBI emulates an MTD when a UBI volume is created. This provides an interface for other file systems to use the UBI.
- UBI debugging

  UBIFS debugging function. This configuration option is used by the creator of the UBIFS for testing the file systems and is not used by users generally.

  After all configuration options are configured, compiling the kernel generates the **ubi_ctrl** file in the **/dev/** directory. **ubi_ctrl** is used to bind the UBI to the MTD and generate the UBI device. The first bound device is ubi0, the second bound device is ubi2, and so on. The device ID depends only on the binding sequence.

2 UBIFS Application Samples

2.1 Mounting an Empty UBIFS Volume

The board is divided into three partitions. Figure 2-1 shows the partition information.

Figure 2-1 Partition information

```
# cat /proc/mtd
dev:    size   erasesize  name
mtd0: 01000000 00040000 "hi_sfc"
mtd1: 00a00000 00020000 "boot"
mtd2: 03200000 00020000 "nand0"
mtd3: 03200000 00020000 "nand1"
mtd4: 01200000 00020000 "nand2"
mtd4: 01200000 00020000 "nand2"
```

To map the mtd2 partition to a UBI volume and use the mtd2 partition as a UBI partition, perform the following steps:

Step 1 Format the UBI partition by running the following command:

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

NOTE

If the partition is erased, the **mount** command can be executed properly. However, you are advised not to erase the partition by running the **flash_eraseall** command because the number of erase operations for each erase block recorded by the UBIFS will be lost if the partition is erased.

Step 2 Bind the UBI to the MTD partition. You can bind the UBI to the mtd2 partition by running the following command:

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

The parameter **-m 2** indicates the mtd2 partition. The UBI device ubi0 can be found in **/dev/** only after the UBI is bound to the MTD partition. If a UBI volume is created, the UBI volume ubi0_0 can be found in **/dev/** and accessed only after the UBI is bound to the MTD partition.

Figure 2-2 shows the information displayed after the command is successfully executed.

Figure 2-2 Information displayed after the command is successfully executed

```
# 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)
```

The characters "UBI device number 1" in the last line indicate that the ubi1 device is successfully created. If you run **ls /dev/ubi*** to view all the devices, one more device (**/dev/ubi1**) is found in the displayed information.

Step 3 Create a UBI volume (a partition of the UBI device) by running the following command:

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

Table 2-1 describes the meanings of the parameters in the preceding command.

Table 2-1 Parameter meanings (1)

Parameter	Description
/dev/ubi1	UBI device (ubi1) that is created in step 3
-N ubifs	Name of the created UBI volume (**ubifs**)
-m	UBI volume size. It is set to the maximum size of the memory space provided by the ubi1 device, that is, the size of the mtd2 partition (50 MB).

Figure 2-3 shows the information displayed after a UBI volume is successfully created.

Figure 2-3 Information displayed after a UBI volume is successfully created

```
# 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
```

The characters "Volume ID 0" in the displayed information indicate that UBI volume 0 is successfully created. If you run **ls /dev/ubi*** to view all the devices, one more device (**/dev/ubi1_0**) is found in the displayed information.

NOTE

Once a UBI volume is created, information about the UBI volume is recorded in the UBI device. The UBI volume does not need to be created again during next startup. You can run **ubirmvol** to delete the UBI volume. If a UBI volume is deleted by using this command, all the data in the UBI volume is deleted.

Step 4 Mount the created empty UBIFS volume to the specified directory by running either of the following commands:

```
# mount –t ubifs /dev/ubi1_0 /mnt/
# mount –t ubifs ubil:ubifs /mnt/
```

The parameter **/dev/ubi1_0** indicates that the ubi1_0 volume is mounted. You can also use the **ubi1:ubifs** parameter. In some kernel versions, the **/dev/ubi1_0** parameter is not supported and only the **ubi1:ubifs** parameter is available. The characters "ubifs" in the **ubi1:ubifs** parameter indicate the name of the UBI volume, which is configured when the UBI volume is created.

Figure 2-4 shows the information displayed if the UBI volume is successfully mounted.

Figure 2-4 Information displayed if the UBI volume is successfully mounted

```
# 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)
```

Figure 2-5 shows the partition information.

Figure 2-5 Partition information

```
# 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
```

The partition sizes and available space size of the UBIFS are not accurate because the UBIFS stores compressed files and the compression ratio is related to the file contents. The available space size may be only 2 MB, but a 4 MB file can be completely stored in the UBIFS after compression.

----End

2.2 Creating the UBI of the Root UBIFS

You can run the following command to create the UBIFS image by using the mtd-utils tool:

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

Table 2-2 describes the meanings of the parameters in the preceding command.

Table 2-2 Parameter meanings (2)

Parameter	Description
-d rootfs_uclibc	Root directory used to create the UBIFS image (**rootfs_uclibc**). This parameter can be replaced by **-r rootfs_uclibc**.
-m 2KiB	Minimum read/write unit (2 KiB). This parameter can be replaced by **-m 2048**. The page size of the NAND flash is 2 KB. The minimum read/write unit indicates the minimum number of bytes read and written by the flash memory at a time. For the NAND flash, the minimum read/write unit is the page size, such as 2 KB, 4 KB, or 8 KB. For the NOR flash, the minimum read/write unit is 1 byte.
-o rootfs.ubiimg	Name of the created image (**rootfs.ubiimg**)
-e 126976	Size of the logical erase block (LEB)
-c 256	Maximum number of LEBs (256) for the file system. The maximum available space of the file system is 512 KiB (256 x 2 KiB).
-F	white-space-fixup enable. This function is used when the image is burnt under the U-boot.

Parameter	Description
-v	Detailed information about UBIFS image creation

The minimum read/write unit and LEB size can be obtained by reading the MTD and UBI system information or by calculation.

Figure 2-6 shows the information displayed after the command for reading the MTD information is executed.

Figure 2-6 Information displayed after the command for reading the MTD information is executed

```
# ./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
```

Before running the command for reading the UBI information, ensure that the UBI is bound to the MTD partition. For details, see step 2 in section 2.1 "Mounting an Empty UBIFS Volume." Figure 2-7 shows the information displayed after the command for reading the UBI information is executed.

Figure 2-7 Information displayed after the command for reading the UBI information is executed

```
# 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
```

The red characters in Figure 2-7 indicate the LEB size.

The LEB size can be calculated. Table 2-3 describes the calculation methods.

Table 2-3 Methods for calculating the LEB size

Flash Type	LEB Size
NOR flash	LEB size = blocksize − 128
NAND flash (without sub pages)	LEB size = blocksize – pagesize x 2
NAND flash (with sub pages)	LEB size = blocksize – pagesize x 1

NOTE

- **blocksize** indicates the size of the physical erase block (PEB) of the flash memory.
- **pagesize** indicates the size of the read/write page of the flash memory.

You can run **mkfs.ubifs** to view the detailed information after a UBIFS image is successfully created, as shown in Figure 2-8.

Figure 2-8 Detailed information after a UBIFS image is successfully created

```
# 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
```

Note that the created image of the root UBIFS is a UBI. The UBIFS can be upgraded to the root file system. For details, see section 2.3 "Upgrading the Empty UBIFS Volume."

This image can be burnt directly to the MTD partition only after format conversion. For details, see section 2.4 "Converting the UBI Format and Burning the UBI."

NOTE

The versions of the tools required for creating the UBI must be consistent with the kernel version. The kernel version of the board is Linux kernel 3.4.35, and the versions of the UBI tools are zlib-1.2.5, lzo-2.03, and ubi-utils-1.5.0. If the tool versions and the kernel version are not consistent, the created image cannot be mounted to the board.

2.3 Upgrading the Empty UBIFS Volume

After the UBI volume is created in kernel mode, perform the following steps to upgrade the UBI volume:

Step 1 Create a UBI volume. For details, see step 2 in section 2.1 "Mounting an Empty UBIFS Volume."

Step 2 Create the UBI of the UBIFS. For details, see section 2.2 "Creating the UBI of the Root UBIFS."

Step 3 Download the UBI of the root file system to the kernel over Trivial File Transfer Protocol (TFTP) by running the following command:

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

Step 4 Upgrade the UBIFS volume in kernel mode by running the following command:

```
# ubiupdatevol /dev/ubil_0  rootfs.ubiimg
```

The parameter **/dev/ubi1_0** indicates the UBI volume to be upgraded and the volume must be created in advance. The contents of the UBI volume does not need to be erased before upgrade.

You can run **ubiupdatevol /dev/ubi1_0 –t** to erase the volume.

Step 5 Configure the startup parameter of **u-boot.bin**.

Figure 2-9 shows the configuration of the **bootargs** parameter (startup parameter of **u-boot.bin**) in the UBIFS.

Figure 2-9 Configuration of the **bootargs** parameter in the UBIFS

```
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)'
```

Table 2-4 describes the meanings of the parameters in the preceding configuration.

Table 2-4 Parameter meanings (3)

Parameter	Description
ubi.mtd=3	This parameter indicates that the UBI is bound to the **/dev/mtd3** partition.
root=ubi0:ubifs	In the parameter **root=ubi0:ubifs**, **ubi0** indicates the partition bound to the UBI, **ubifs** indicates the name of the UBI volume specified during creation. In some kernel versions, parameters in the root=/dev/ubi1_0 format cannot be identified.
rootfstype=ubifs	This parameter indicates that the UBIFS is used.

----End

2.4 Converting the UBI Format and Burning the UBI

Perform the following steps:

Step 1 Create the UBI of the UBIFS. For details, see section 2.2 "Creating the UBI of the Root UBIFS."

Step 2 Create the configuration file for converting the UBI format.

During UBI format conversion, the configuration file **ubi.cfg** must be created and will be used in step 3. Figure 2-10 shows the contents of the **ubi.cfg** file.

Figure 2-10 Contents of the ubi.cfg file

```
[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
```

Table 2-5 describes the meanings of the parameters in the preceding command.

Table 2-5 Parameter meanings (4)

Parameter	Description
mode=ubi	Mandatory parameter. Currently this parameter can be set only to **ubi** and is reserved for future extensions.
image=./rootfs*.img	Name of the UBIFS image corresponding to the UBI volume, that is, the image created in section 2.2 "Creating the UBI of the Root UBIFS."
vol_id=0	Volume ID. The UBI image may contain multiple volumes, which are distinguished by the volume ID.
vol_type=dynamic	Volume type, indicating that the current volume has read and write properties. If this volume is read-only, the corresponding parameter is **vol_type=static**.
vol_name=ubifs	Volume name. The volume name is used when the UBIFS is used as the root file system.
vol_flags=autoresize	Volume size, indicating that the volume size can be dynamically extended

Step 3 Convert the UBI format by running the following command:

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

Table 2-6 describes the meanings of the parameters in the preceding command.

Table 2-6 Parameter meanings (5)

Parameter	Description
-o rootfs.img	Name of the converted UBI (**rootfs.img**). The name of the input UBI is specified by the **ubi.cfg** configuration file.
-m 2KiB	Minimum read/write unit (2 KiB)
-p 128KiB	PEB size of the flash memory
ubi.cfg	Configuration file
-v	Detailed information about UBI format conversion

Figure 2-11 shows the detailed information after the UBI format is successfully converted.

Figure 2-11 Detailed information after the UBI format is successfully converted

```
$ ./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
```

Step 4 Burn the converted UBI under the U-boot.

The method of burning the UBI under the U-boot is the same as that of burning the UBI under the kernel. You can burn the UBI by running the following commands:

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

NOTE

When data is written to the NAND flash, the data length must be equal to the size of the converted UBI. Otherwise, a file system error occurs.

----End

3 Appendix

3.1 Commands Related to the UBI and MTD

Table 3-1 describes the meanings of the commands related to the UBI and MTD.

Table 3-1 Command meanings

Command	Description
cat /proc/mtd	Views the information about each MTD in the current system.
mtdinfo /dev/mtd3	Views the MTD partition information.
ubinfo –a	Displays the partition information about all the UBI partitions.
ls /dev/ubi*	Views the UBI device nodes and UBI volumes.

3.2 UBI FAQs

3.2.1 Must the Empty Flash Memory Be Formatted by Running the Erase Command Before the Flash Memory Runs the UBI?

You are advised not to format the flash memory by running the erase command. The UBIFS records the number of times that each block is used. If the flash memory is formatted by running the erase command, the number is also erased, which affects read/write leveling of the UBIFS. You can erase the flash memory while maintaining the number of times that each block is read or written by running **ubiformat**.

3.2.2 Why Does the UBI Volume Fail to Be Mounted After Restart?

When the kernel version is 3.0 or later, to rectify the bug, the **-F** parameter needs to be added to the command for creating the UBI of the root UBIFS. This parameter is used to configure the flag for rectifying the blank regions. When the UBI is mounted for the first time, the blank regions are rectified so that they can be used later.