UBIFS User Guide

Issue 03

Date 2018-11-30

HiSilicon (Shanghai) Technologies Co., Ltd.

Address: New R&D Center, 49 Wuhe Road, Bantian,
Longgang District,
Shenzhen 518129 P. R. China

Website: http://www.hisilicon.com/en/

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.

Related Version

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

Product Name	Version
Hi3519	V100
Hi3519	V101
Hi3516C	V300
Hi3516D	V200
Hi3516E	V100
Hi3516E	V200
Hi3516E	V300
Hi3518E	V300
Hi3559	V100
Hi3556	V100
Hi3516A	V200
Hi3536C	V100
Hi3559A	V100ES
Hi3559A	V100

Product Name	Version
Hi3559C	V100
Hi3536D	V100
Hi3531D	V100
Hi3521D	V100
Hi3520D	V400
Hi3521A	V100
Hi3531A	V100
Hi3518E	V200
Hi3518E	V201
Hi3516C	V200
Hi3519A	V100
Hi3556A	V100
Hi3516C	V500
Hi3516D	V300
Hi3516A	V300
Hi3559	V200
Hi3556	V200

Intended Audience

This document is intended for technical support engineers.

Change History

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

Issue 03 (2018-11-30)

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

The descriptions in the Hi3516E V200, Hi3516E V300, and Hi3518E V300 are added.

Issue 02 (2018-05-20)

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

Sections 2.2, 2.4, and 2.5 are modified.

Issue 01 (2018-01-16)

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

The descriptions in the Hi3521D V100, Hi3531D V100, Hi3559A V100, and Hi3559C V100 are added.

Issue 00B08 (2017-11-20)

This issue is the eighth draft release, which incorporates the following changes:

Chapter 2 UBIFS Application Samples

Section 2.2 and 2.4 are updated.

Section 2.5 is added.

Issue 00B07 (2017-08-15)

This issue is the seventh draft release, which incorporates the following changes:

The description of the Hi3536D V100 is added.

Issue 00B06 (2017-05-27)

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

The description of the Hi3559A V100ES is added.

Issue 00B05 (2017-04-10)

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

The description of the Hi3536C V100 is added.

Issue 00B04 (2017-03-27)

This issue is the fourth draft release, which incorporates the following changes:

Chapter 2 UBIFS Application Samples

Section 2.2 is modified.

Chapter 3 Appendix

Descriptions in section 3.2 are added.

Issue 00B03 (2017-02-25)

This issue is the third draft release, which incorporates the following changes:

The description of the Hi3556 V100 is added.

Section 2.1 is modified.

Issue 00B02 (2016-07-30)

The contents related to the Hi3516C V300 and Hi3559 V100 are added.

Issue 00B01 (2015-11-04)

This issue is the first draft release.

Contents

1 UBI Configuration in the Kernel

Table 1-1 describes the kernel versions of the chips.

Table 1-1 Kernel versions of the chips

Chip	Kernel Version
Hi3519 V100, Hi3519 V101, Hi3516A V200, Hi3516C V300, Hi3516E V100, Hi3559 V100, Hi3556 V100, Hi3559A V100ES, Hi3521D V100, Hi3531D V100, Hi3520D V400, Hi3536C V100	Linux 3.18.y
Hi3536D V100, Hi3521A V100, Hi3531A V100, Hi3518E V200, Hi3518E V201, Hi3516C V200, Hi3559A V100, Hi3559C V100, Hi3519A V100, Hi3556A V100, Hi3516C V500, Hi3516D V300, Hi3516A V300, Hi3559 V200, Hi3556 V200, Hi3516E V200, Hi3516E V300, Hi3518E V300, Hi3516D V200	Linux 4.9.y

CAUTION

This document uses the Linux kernel 3.18.y on the board as an example. 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 steps:

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  --->
   <*> Enable UBI - Unsorted block images  --->
```

```
-*- Enable UBI - Unsorted block images
(4096) UBI wear-leveling threshold
(20)  Maximum expected bad eraseblock count per 1024 eraseblocks
[ ]   UBI Fastmap (Experimental feature)
<*>   MTD devices emulation driver (gluebi)
[ ]   Read-only block devices on top of UBI volumes
```

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

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

Figure 1-2 Enabling the UBIFS

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

```
<*>   UBIFS file system support
[ ]     Advanced compression options (NEW)
```

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.

----End

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.

2 UBIFS Application Samples

2.1 Mounting an Empty UBIFS Volume

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

Figure 2-1 Partition information

```
# cat /proc/mtd
dev:    size   erasesize  name
mtd0: 01000000 00020000 "hinand"
mtd1: 00400000 00020000 "kernel"
mtd2: 02000000 00020000 "rootfs"
mtd3: 03200000 00020000 "ubi"
```

To map the mtd3 partition to a UBI volume and use the mtd3 partition as a UBI partition, perform the following steps:

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

```
# ubiformat /dev/mtd3
```

NOTE

- After the OSDRV compilation is complete, the generated UBI tools are stored in the **osdrv/pub/bin/ board_xxx /** directory. The naming of the **board_xxx** directory is related to the selected tool chains and products in compiling.
- The UBI tools need to be downloaded to the board. The executable permission can be added by running the **chmod +x '*Name of a UBI tool*'** command.
- 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.
- At least 44 blocks are required for MTD partitions.

Step 2 Bind the UBI to the MTD partition. You can bind the UBI to the mtd3 partition by running the following command:

```
# ubiattach /dev/ubi_ctrl -m 3
```


The parameter **-m 3** indicates the mtd3 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 3
UBI: attaching mtd3 to ubi0
UBI: scanning is finished
UBI: attached mtd3 (name "UBIFS01", size 50 MiB) to ubi0
UBI: PEB size: 131072 bytes (128 KiB), LEB size: 126976 bytes
UBI: min./max. I/O unit sizes: 2048/2048, sub-page size 2048
UBI: VID header offset: 2048 (aligned 2048), data offset: 4096
UBI: good PEBs: 400, bad PEBs: 0, corrupted PEBs: 0
UBI: user volume: 0, internal volumes: 1, max. volumes count: 128
UBI: max/mean erase counter: 1/1, WL threshold: 4096, image
sequence number: 728242785
UBI: available PEBs: 376, total reserved PEBs: 24, PEBs reserved
for bad PEB handling: 20
UBI: background thread "ubi_bgt0d" started, PID 101
UBI: attaching mtd3 to ubi0
UBI: scanning is finished
UBI: attached mtd3 (name "UBIFS01", size 50 MiB) to ubi0
UBI: PEB size: 131072 bytes (128 KiB), LEB size: 126976 bytes
UBI: min./max. I/O unit sizes: 2048/2048, sub-page size 2048
UBI: VID header offset: 2048 (aligned 2048), data offset: 4096
UBI: good PEBs: 400, bad PEBs: 0, corrupted PEBs: 0
UBI: user volume: 0, internal volumes: 1, max. volumes count: 128
UBI: max/mean erase counter: 1/1, WL threshold: 4096, image
sequence number: 728242785
UBI: available PEBs: 376, total reserved PEBs: 24, PEBs reserved
for bad PEB handling: 20

UBI: background thread "ubi_bgt0d" started, PID 101
```

The characters "UBI bgt0d" in the last line indicate that the ubi0 device is successfully created. If you run **ls /dev/ubi*** to view all the devices, one more device (**/dev/ubi0**) 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/ubi0 -N ubifs -s  SIZE
```

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

Table 2-1 Parameter meanings (1)

Parameter	Description
/dev/ubi0	UBI device (ubi1) that is created in step 3
-N ubifs	Name of the created UBI volume (**ubifs**)
-s SIZE	Size of the created partition

NOTE

- The value of ***-s SIZE*** must be less than the size of the space provided by the ***/dev/ubi0*** device.
- You can view the size of available LEBs by running the **ubinfo** command. As shown in the red lines in Figure 2-3, when the size of the space provided by the UBI device is 50 MiB, the size of available space is 45.5 MiB. Therefore, ensure that the size of the created volume is less than the size of available LEBs.

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

```
# ubinfo /dev/ubi0
ubi0
Volumes count:                       0
Logical eraseblock size:             126976 bytes, 124.0 KiB
Total amount of logical eraseblocks:    400 (50790400 bytes, 48.4 MiB)
Amount of available logical eraseblocks: 376 (47742976 bytes, 45.5
MiB)
Maximum count of volumes             128
Count of bad physical eraseblocks:      0
Count of reserved physical eraseblocks:  20
Current maximum erase counter value:    1
Minimum input/output unit size:         2048 bytes
Character device major/minor:           253:0
```

If you run **ls /dev/ubi*** to view all the devices, one more device (**/dev/ubi0_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/ubi0_0 /mnt/
# mount -t ubifs ubi0:ubifs /mnt/
```


The parameter **/dev/ubi0_0** indicates that the ubi1_0 volume is mounted. You can also use the **ubi0:ubifs** parameter. In some kernel versions, the **/dev/ubi0_0** parameter is not supported and only the **ubi0:ubifs** parameter is available. The characters "ubifs" in the **ubi0: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/ubi0_0 /mnt/
UBIFS: default file-system created
UBIFS: background thread "ubifs_bgt0_0" started, PID 107
UBIFS: mounted UBI device 0, volume 0, name "ubifs"
UBIFS: LEB size: 126976 bytes (124 KiB), min./max. I/O unit sizes:
2048 bytes/2048 bytes
UBIFS: FS size: 46473216 bytes (44 MiB, 366 LEBs), journal size
2285568 bytes (2 MiB, 18 LEBs)
UBIFS: reserved for root: 2195044 bytes (2143 KiB)
UBIFS: media format: w4/r0 (latest is w4/r0), UUID 80EC88B4-1AF1-4193-
AC8F-5506B1A21742, small LPT model
```

Figure 2-5 shows the partition information.

Figure 2-5 Partition information

```
# df -h
Filesystem              Size      Used Available Use% Mounted on
/dev/root              32.0M     14.8M     17.2M  46% /
devtmpfs               28.9M      4.0K     28.9M   0% /dev
/dev/ubi1_0            40.3M     20.0K     38.2M   0% /mnt
```

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 -F -d rootfs_uclibc -m 2KiB -o rootfs.ubiimg -e 126976 -c
```



```
256 -v
```

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

Table 2-2 Parameter meanings (2)

Parameter	Description
-F	white-space-fixup enable. This function is used when the image is burnt under the U-Boot.
-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 calculated as 256 multiplied by LEB.
-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/mtd3
mtd3
Name:                           ubi
Type:                           nand
Eraseblock size:                131072 bytes, 128.0 KiB
Amount of eraseblocks:          400 (52428800 bytes, 50.0 MiB)
Minimum input/output unit size: 2048 bytes
Sub-page size:                  2048 bytes
OOB size:                       60 bytes
Character device major/minor:   90:6
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:    400 (50790400 bytes, 48.4
MiB)
Amount of available logical eraseblocks: 0 (0 bytes)
Maximum count of volumes             128
Count of bad physical eraseblocks:      0
Count of reserved physical eraseblocks:  20
Current maximum erase counter value:    2
Minimum input/output unit size:         2048 bytes
Character device major/minor:           253:0
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. 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 (not U-Boot 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/ubi0_0  rootfs.ubiimg
```

The parameter **/dev/ubi0_0** indicates the UBI volume to be upgraded and the volume must be created in advance. The contents of the UBI volume do not need to be erased before upgrade.

You can run **ubiupdatevol/dev/ubi0_0 –t** to erase the volume.

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

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

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

```
setenv bootargs 'mem=128M console=ttyAMA0,115200 ubi.mtd=3
root=ubi0:ubifs rootfstype=ubifs rw
mtdparts=hinand:1M(boot),4M(kernel),32M(yaffs2),50M(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.

Parameter	Description
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/ubi0_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_hi35xx_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*.ubiimg	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.

Parameter	Description
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.ubifs -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 ubifs.ubifs	Name of the converted UBI (**rootfs.ubifs**). 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.ubifs -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_hi35xx_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_xxx_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:

```
# nand erase [offset] [len]
# mw.b [ddr_addr] 0xff [len]
# tftp [ddr_addr] rootfs.ubifs
# nand write [ddr_addr] [flash_start_addr] [ubi_len]
```


NOTE

- **offset** indicates the start address of flash operations. For example, nand erase 0x800000 0x720000 indicates that the flash memory erasure starts from 8 MB and the length of data to be erased is 7296 KB.
- **len** indicates the length of the partition of the UBIFS root file system.
- **ddr_addr** is the memory address, an available DDR address should be selected for operation. Otherwise, the system may suspend. For details about the DDR address, see the burning method in the *Hi35xx U-Boot Porting Development Guide*.

----End

2.5 Creating UBIs in One-click Mode by Using the mkubiimg.sh Script

For the complicated procedure of creating UBI s and root UBIFS images in section 2.1 to 2.4, the **mkubiimg.sh** script is provided in the **osdrv/tools/pc/ubi_sh/** directory and specially used for creating UBIs and root UBIFS images that are used by NAND flashes without sub pages. Run the following command:

```
$./mkubiimg.sh  Chip Pagesize  Blocksize  Dir  Size  Tool_Path  Res
```

```
Parameter Description:
Chip          Chip name, for example, hi35xx
Pagesize      NAND page size, 2k/4k/8k
Blocksize     NAND block size. 128k/256k/1M
Dir           Root directory being created as UBIFS images
Size          Size of the partition of the root UBIFS
Tool_Path     Directory for placing mkfs.ubifs and ubinize tools
              that used for creating UBIs, generally is
              osdrv/pub/bin/pc/
Res           Whether to reserve UBIs and the ubicfg
              configuration file (1: Yes 0: No (by default))
```

NOTE

For example, run the following command: ./mkubiimg.sh hi35xx 2k 128k osdrv/pub/rootfs 50M osdrv/pub/bin/pc 1

The following three files are generated:

- **rootfs_hi35xx_2k_128k_50M.ubiimg**: The image cannot be directly burnt into the MTD partition. You can run the **ubiupdate** command to upgrade the empty UBIFS in the kernel. For details, see section 2.3 "Upgrading the Empty UBIFS Volume."
- **rootfs_hi35xx_2k_128k_50M.ubicfg**: A configuration file is required during UBI format conversion. This configuration file mainly specifies the volume name, volume type, extension attribute, and other information about the root file system.
- **rootfs_hi35xx_2k_128k_50M.ubifs**: The image can be directly burned into the NAND flash and is the final image after format conversion of the .ubiimg file.

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

- 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**.

- Why does the UBI volume fail to be mounted after restart?

  When the kernel version is 3.0 or later, the **-F** parameter needs to be added to the command for creating the UBI of the root UBIFS to rectify the bug. 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.

- After the UBI partition is written into raw data, why an error checking and correction (ECC) error occurs when the attach command is run after restart?

  UBIFS has strict requirements on the format of data in partitions. If the partition is written into raw data, the data format is damaged. To run **attach**, run **ubiformat** or **flash_erase** to format the partition. Erase the UBI partition before any UBI operations if the UBI partition is written into raw data.

