

# Nand Flash Open Source Solution Develop Guide

---

ID: RK-KF-YF-314

Release Version: V2.0.1

Release Date: 2020-11-27

Security Level: ☐Top-Secret ☐Secret ☐Internal ☒Public

## DISCLAIMER

THIS DOCUMENT IS PROVIDED “AS IS”. ROCKCHIP ELECTRONICS CO., LTD.(“ROCKCHIP”)DOES NOT PROVIDE ANY WARRANTY OF ANY KIND, EXPRESSED, IMPLIED OR OTHERWISE, WITH RESPECT TO THE ACCURACY, RELIABILITY, COMPLETENESS, MERCHANTABILITY, FITNESS FOR ANY PARTICULAR PURPOSE OR NON-INFRINGEMENT OF ANY REPRESENTATION, INFORMATION AND CONTENT IN THIS DOCUMENT. THIS DOCUMENT IS FOR REFERENCE ONLY. THIS DOCUMENT MAY BE UPDATED OR CHANGED WITHOUT ANY NOTICE AT ANY TIME DUE TO THE UPGRADES OF THE PRODUCT OR ANY OTHER REASONS.

## Trademark Statement

"Rockchip", "瑞芯微", "瑞芯" shall be Rockchip's registered trademarks and owned by Rockchip. All the other trademarks or registered trademarks mentioned in this document shall be owned by their respective owners.

**All rights reserved. ©2020. Rockchip Electronics Co., Ltd.**

Beyond the scope of fair use, neither any entity nor individual shall extract, copy, or distribute this document in any form in whole or in part without the written approval of Rockchip.

Rockchip Electronics Co., Ltd.

No.18 Building, A District, No.89, software Boulevard Fuzhou, Fujian, PRC

Website: [www.rock-chips.com](http://www.rock-chips.com)

Customer service Tel: +86-4007-700-590

Customer service Fax: +86-591-83951833

Customer service e-Mail: [fae@rock-chips.com](mailto:fae@rock-chips.com)

## Preface

### Overview

Rockchip SDKs use closed-source miniloader to load trust and u-boot by default. All memories (eMMC NAND or NOR Flash) are accessed through block interface. For developers who want to access NAND or NOR Flash through MTD interface, Rockchip provides open source SPL to load trust and u-boot, and access NAND or NOR Flash through MTD interface in u-boot.

### Product Version

Chipset	Kernel version
RK3308	Linux 4.4, Linux4.19
RK1126 & RK1109	Linux 4.19

### Intended Audience

This document (this guide) is mainly intended for:

Technical support engineers

Software development engineers

### Revision History

Version	Author	Date	Change Description
V1.0.0	HKH	2019-06-20	Initial version
V1.0.1	HKH	2019-11-11	Add SD card upgrade introduction
V1.0.2	Ruby Zhang	2020-07-08	Update the format of the document
V1.1.0	Jair Wu	2020-07-10	Add u-boot compile introduction
V2.0.0	Jon Lin	2020-10-19	Improve the driver configuration and other details
V2.0.1	Jon Lin	2020-11-27	Add UBIFS multi volume support, increase or decrease ubiattach parameter description

## Contents

### Nand Flash Open Source Solution Develop Guide

1. Notes on Open Source Solution
  - 1.1 Feature
  - 1.2 Nand Flash Infomation
  - 1.3 Other Notes
2. Build Configuration Changes
  - 2.1 SPL & U-Boot
    - 2.1.1 Configurations
    - 2.1.2 Compile Instrodution
  - 2.2 Kernel
    - 2.2.1 SLC Nand Open Source Solution
    - 2.2.2 SPI Nand Open Source Solution
  - 2.3 MTD Partition Table
    - 2.3.1 Parse GPT And Generate MTD Partition In RK SDK
    - 2.3.2 Dts Adds Rootfs Mount Information
    - 2.3.3 Vendor Storage
  - 2.4 Buildroot
  - 2.5 Building Script Patch
    - 2.5.1 RK3308 Patches
  - 2.6 Partition Table
  - 2.7 SD Booting Upgrade
  - 2.8 Enable Key Log
3. OTA
  - 3.1 Upgrade MTD Partitions By Shell Command
  - 3.2 Upgrade UBIFS Image By Shell Command
  - 3.3 Upgrade MTD Partitions By APIs
4. UBIFS Filesystem
  - 4.1 Instruction
  - 4.2 Configuration
  - 4.3 Making Images
  - 4.4 Mount UBIFS
  - 4.5 Support SquashFS In UBI Block
  - 4.6 UBIFS OTA
5. PC Tools For Downloading
6. Flash Programmer
  - 6.1 SPI Nand Flash Programmer
    - 6.1.1 Make SPI Nand Images
    - 6.1.2 SPI Nand Flash Programmer Operation
  - 6.2 SLC Nand Flash Programmer
    - 6.2.1 Make SLC Nand Images
    - 6.2.2 SLC Nand Flash Programmer Operation
7. Reference documents

# 1. Notes on Open Source Solution

---

## 1.1 Feature

Confirm these following feature:

Note.	Support Device Type	Register Device Type	Filesystem	Download methods
SLC Nand open source solution (Parallel Nand)	SLC Nand	mtd、ubiblock	SquashFS、UBIFS	USB download、SD card download、Flash Programmer
SPI Nand open source solution	SPI Nand	mtd、ubiblock	SquashFS、UBIFS	USB download、SD card download、Flash Programmer

Main points:

- The device type to choose
- Does the filesystem meet the requirements

## 1.2 Nand Flash Infomation

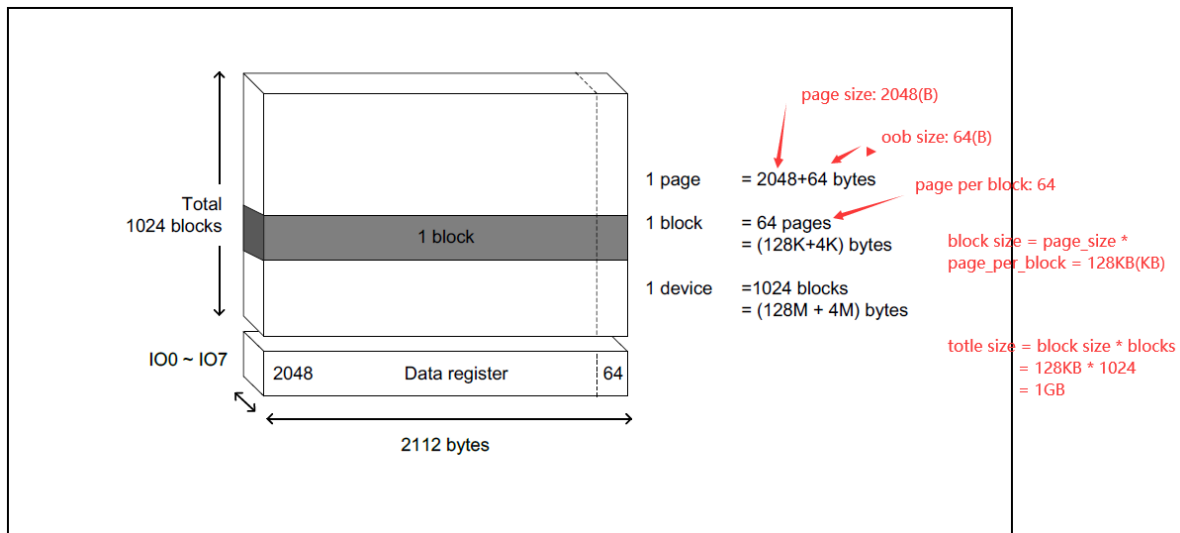
Making UBIFS images、IDB images(Pre loader) should depend on the concrete Nand flash infomation, including follow information:

- page size, SLC Nand is usually in 2KB or 4KB;
- page per block, SLC Nand is usually in 64 or 128;
- block size = page\_size \* page\_per\_block, SLC Nand is usually 128KB or 256KB;
- oob size, SLC Nand is usually in 64B, 128B or 256B.

The default configuration is mostly base on 2KB page size and 128 block size flash, change it if need when you perform the following process:

- Making Programmer images
- Making UBIFS filesystem images

The Nand flash information is mostly like this:



### 1.3 Other Notes

- UBIFS images is not compatible with two different Nand flash with different page size or block size, Therefore, the selection of flash devices should be careful.
- SLC NAND in the document refers to the parallel port NAND

## 2. Build Configuration Changes

### 2.1 SPL & U-Boot

Note.	Support Device Type	Driver Codes	Flash Framework	Device Register Type	Filesystem	Download methods
SLC Nand open source solution	SLC Nand	drivers/mtd/nand/raw	drivers/mtd/nand/raw	mtd	UBIFS	USB download、SD card download、Flash Programmer
SPI Nand open source solution	SPI Nand	drivers/spi	drivers/mtd/nand/spi	mtd	UBIFS	USB download、SD card download、Flash Programmer\

## 2.1.1 Configurations

The defconfig is configured as follows:

Add:

```
1 // MTD support
2 CONFIG_MTD=y
3 CONFIG_CMD_MTD_BLK=y
4 CONFIG_SPL_MTD_SUPPORT=y
5 CONFIG_MTD_BLK=y
6 CONFIG_MTD_DEVICE=y
7
8 // spi nand support
9 CONFIG_NAND=y
10 CONFIG_MTD_SPI_NAND=y
11 CONFIG_ROCKCHIP_SFC=y
12 CONFIG_SPL_SPI_FLASH_SUPPORT=y
13 CONFIG_SPL_SPI_SUPPORT=y
14
15 // nand support
16 CONFIG_NAND=y
17 CONFIG_CMD_NAND=y
18 CONFIG_NAND_ROCKCHIP=y
19 CONFIG_SPL_NAND_SUPPORT=y
20 CONFIG_SYS_NAND_U_BOOT_LOCATIONS=y
21 CONFIG_SYS_NAND_U_BOOT_OFFSETS=0x8000
22 CONFIG_SYS_NAND_U_BOOT_OFFSETS_REDUND=0x10000
23
24 // rkfw firmware uboot trust address (Needless in fit format)
25 CONFIG_RKFW_TRUST_SECTOR=0X3000    #The flashing address in memory is in
    sectors, 1 sector=512 Bytes, which is the start address of the trust in
    paramter.txt
26 CONFIG_RKFW_U_BOOT_SECTOR=0X2000    #The flashing address in memory is in
    sectors, 1 sector=512 Bytes, which is the starting address of u-boot in
    paramter.txt
```

Remove:

```
1 CONFIG_RKFLASH=y
2 CONFIG_RKNANDC_NAND=y
3 CONFIG_RKSFC_NAND=y
4 CONFIG_RKSFC_NOR=y
```

## 2.1.2 Compile Instrodution

64-bits:

Excute `./make.sh uboot` in root path of SDK or excute `./make.sh rk3308` in [ROOT PATH]/u-boot.

32-bits:

64-bits loader is needed in 32-bit u-boot compilation, so compile 64-bits u-boot firstly, backup spl/u-boot-spl.bin, replace spl/u-boot-spl.bin after compiling 32-bits u-boot, and then pack loader, commands are as follows:

```
1 | cd u-boot
2 | ./make.sh rk3308
3 | cp spl/u-boot-spl.bin ..
4 | ./make.sh rk3308-aarch32
5 | cp ../u-boot-spl.bin spl/
6 | ./make.sh spl-s ../rkbin/RKBOOT/RK3308MINIALL_WO_FTL.ini
```

These commads will generate rk3308\_loader\_wo\_ftl\_v\*.bin which you can flash into board as final loader.

2.2 Kernel

Note.	Support Device Type	Driver Codes	Flash Framework	Device Register Type	Filesystem	Download methods
SLC Nand open source solution	SLC Nand	drivers/mtd/nand/raw	drivers/mtd/nand/raw	mtd	UBIFS	USB download、SD card download、Flash Programmer
SPI Nand open source solution	SPI Nand	drivers/rkflash	drivers/rkflash	mtd	UBIFS	USB download、SD card download、Flash Programmer

2.2.1 SLC Nand Open Source Solution

Configuration:

```
1 | CONFIG_RK_FLASH=n /* It's not compatible */
2 | CONFIG_MTD_NAND_ROCKCHIP_V6=y /* NandC v6 is depending on TRM NANDC->NANDC_NANDC_VER register, 0x00000801 */
3 | # CONFIG_MTD_NAND_ROCKCHIP_V9=y /* NandC v9 is depending on TRM NANDC->NANDC_NANDC_VER register, 0x56393030, Currently only RK3326 is relevant */
4 | CONFIG_MTD_CMDLINE_PARTS=y
```

Sources Codes:

```
1 | ./drivers/mtd/nand/raw/
```

Reserved 4 blocks for bad block table, so the firmware should not overlay the area and it's better to reserved 1MB, consult to "MTD Partition Table" chapter for detail.

## 2.2.2 SPI Nand Open Source Solution

Configuration:

```
1 CONFIG_RK_FLASH=y
2
3 CONFIG_RK_SFC_NAND=y      /* SPI Nand flash */
4 CONFIG_RK_SFC_NAND_MTD=y /* SPI Nand flash and partitions is register as mtd
   device, otherwise block devices(rkflash0pn) */
5 CONFIG_MTD_CMDLINE_PARTS=y
```

Sources Codes:

```
1 | ./drivers/rkflash
```

SPI Nand Bad blocks Management Strategy:

The last 4 blocks are reserved for Nand bad block table, it's better to reserve 1MB.

Sources Codes:

```
1 | ./drivers/rkflash/sfc_nand_mtd_bbt.c
```

Reserved 4 blocks for bad block table, so the firmware should not overlay the area and it's better to reserved 1MB, consult to "MTD Partition Table" chapter for detail.

## 2.3 MTD Partition Table

### 2.3.1 Parse GPT And Generate MTD Partition In RK SDK

RK SDK support parsing GPT and generate mtdparts to kernel.

Notes:

- Each partition of SLC NAND and SPI NAND open source solution images should reserve 2 ~ 3 redundant flash block sizes, so that when bad blocks are encountered, there is redundant space to replace, especially for uboot partition.
- Partition should start from address which is flash block size aligned
- The last 4 blocks are reserved for Nand bad block table, so the firmware should not overlay the area, there are two specific situations:

parameter.txt use the "grow" flag for the last partition in the code: the size of the last partition will be automatically adjusted in the code, and the SDK is the scheme by default;

parameter.txt The last partition in does not use the "grow" flag or does not use the GPT scheme: the last user partition should not be defined to the 1MB space at the end of the flash.

- The partition table should be GPT table, that is, configure the following fields in parameter.txt file:

```
1 | TYPE: GPT
```



## 2.3.2 Dts Adds Rootfs Mount Information

Select and add the appropriate code in dts bootargs:

```
1 | ubi.mtd=4 root=ubi0:rootfs rootfstype=ubifs
2 | ubi.mtd=3 ubi.block=0,rootfs root=/dev/ubiblock0_0 rootfstype=squashfs /*
   | Mount SquashFS on UBI block */
```

Note:

1. MTD partitions is correspond with partitions parameter.txt, counting from mtd0 or mtblock0;
2. mtd is char device, mtblock is block device

## 2.3.3 Vendor Storage

Add a partition "vnvm" in parameter.txt to enable vendor storage. The size of the partition needs to be an integer multiple of the NAND flash block size, it is recommended to configure 1MB.

## 2.4 Buildroot

Takes ubifs as an rootfs example, refer to the configuration below, please refer to ubifs for detailed parameters configurations:

```
+-----^(-)-----
[ ] tar the root filesystem
[*] ubi image containing an ubifs root filesystem
(0x20000) physical eraseblock size PEB:Physical logical block size
(2048) sub-page size Page size
[ ] Use custom config file
(-v) Additional ubinize options Print compilation information
-*- ubifs root filesystem
(0x1f000) logical eraseblock size LEB:Logical erase block size
(0x800) minimum I/O unit size Page size
(488) maximum logical eraseblock count Number of logical erase blocks
      ubifs runtime compression (lzo) --->
      Compression method (no compression) --->
(-F -v) Additional mkfs.ubifs options -F can be flashed, -v prints compilation
[ ] yaffs2 root filesystem information
+-----
```

When finishing the configuration, use "make savedefconfig" to save Buildroot configuration.

## 2.5 Building Script Patch

Add spl building to build.sh. The generated spl file is located at "u-boot/spl/u-boot-spl.bin", but parts of SOC should patch following patches.

### 2.5.1 RK3308 Patches

#### Building Pre loader

The SDK generate Pre loader from the Miniloader project with FTL version which it not support this open source case, so you have to choose one of following patch:

Closed source without FTL support Miniloader.bin :

```
1 diff --git a/common/build.sh b/common/build.sh
2 index 671decd..c4fe085 100755
3 --- a/common/build.sh
4 +++ b/common/build.sh
5 @@ -46,7 +46,7 @@ function build_uboot(){
6     if [ -f u-boot/*_loader*.bin ]; then
7         rm u-boot/*_loader*.bin
8     fi
9     - cd u-boot && ./make.sh $RK_UBOOT_DEFCONFIG && cd -
10    + cd u-boot && ./make.sh $RK_UBOOT_DEFCONFIG && ./make.sh spl-s
11    + ./rkbin/RKBOOT/RK3308MINIALL_WO_FTL.ini && cd -
12    if [ $? -eq 0 ]; then
13        echo "====Build uboot ok!===="
14    else
```

Open source SPL:

consult to "Compile Introduction" chapter.

### Modify SDK Filesystem Configuration

Modify the following fields in BoardConfig.mk:

```
1 export RK_ROOTFS_TYPE=ubi
2 export RK_OEM_FS_TYPE=ubi
3 export RK_USERDATA_FS_TYPE=ubi
```

Package tools modification (tools directory), oem and userdata are not packaged as follows:

```
1 diff --git a/linux/Linux_Pack_Firmware/rockdev/rk3308-package-file
2 b/linux/Linux_Pack_Firmware/rockdev/rk3308-package-file
3 index 92c0259..260e2fe 100755
4 --- a/linux/Linux_Pack_Firmware/rockdev/rk3308-package-file
5 +++ b/linux/Linux_Pack_Firmware/rockdev/rk3308-package-file
6 @@ -9,8 +9,8 @@ uboot          Image/uboot.img
7 boot          Image/boot.img
8 rootfs        Image/rootfs.img
9 recovery       Image/recovery.img
10 -oem           Image/oem.img
11 -userdata:grow Image/userdata.img
12 +#oem          Image/oem.img
13 +#userdata:grow Image/userdata.img
```

## 2.6 Partition Table

The partition table should be GPT table, that is, configure the following fields in parameter.txt file:

```
1 TYPE: GPT
```

## 2.7 SD Booting Upgrade

The SPL solution supports SD card upgrade solution. If you need this function, the following configuration should be turned on:

U-Boot directory:

```
1 diff --git a/arch/arm/dts/rk3308-evb.dts b/arch/arm/dts/rk3308-evb.dts
2 index 3178d45..68853d6 100644
3 --- a/arch/arm/dts/rk3308-evb.dts
4 +++ b/arch/arm/dts/rk3308-evb.dts
5 @@ -330,7 +330,7 @@
6         sd-uhs-sdr25;
7         sd-uhs-sdr50;
8         sd-uhs-sdr104;
9 -     status = "disabled";
10 +     status = "okay";
11 };
12
13 &u2phy {
```

```
1 -CONFIG_OF_SPL_REMOVE_PROPS="pinctrl-names clock-names interrupt-
parent assigned-clocks assigned-clock-rates assigned-clock-parents"
2 +CONFIG_OF_SPL_REMOVE_PROPS=""
3 +CONFIG_SPL_PINCTRL_GENERIC=y
4 +CONFIG_SPL_PINCTRL=y
```

kernel directory:

```
1 diff --git a/arch/arm64/boot/dts/rockchip/rk3308.dtsi
b/arch/arm64/boot/dts/rockchip/rk3308.dtsi
2 index 8a98886..970fb69 100644
3 --- a/arch/arm64/boot/dts/rockchip/rk3308.dtsi
4 +++ b/arch/arm64/boot/dts/rockchip/rk3308.dtsi
5 @@ -1166,6 +1166,8 @@
6         nandc_id = <0>;
7         clocks = <&cru SCLK_NANDC>, <&cru HCLK_NANDC>;
8         clock-names = "clk_nandc", "hclk_nandc";
9 +     pinctrl-names = "default";
10 +     pinctrl-0 = <&flash_csn0 &flash_rdy &flash_ale &flash_cle
&flash_wrn &flash_rdn &flash_bus8>;
11         status = "disabled";
12     };
```

```
1 diff --git a/drivers/mtd/nand/rockchip_nand_v6.c
b/drivers/mtd/nand/rockchip_nand_v6.c
2 index 5a74427..31208ba 100644
3 --- a/drivers/mtd/nand/rockchip_nand_v6.c
4 +++ b/drivers/mtd/nand/rockchip_nand_v6.c
5 @@ -20,6 +20,7 @@
6 #include <linux/gpio.h>
7 #include <linux/interrupt.h>
8 #include <linux/iopoll.h>
9 +#include <asm/io.h>
```

```

10
11 #define          NANDC_V6_NUM_BANKS          4
12 #define          NANDC_V6_DEF_TIMEOUT        20000
13 @@ -689,6 +690,7 @@ static int rk_nandc_probe(struct platform_device *pdev)
14     int irq;
15     int ret;
16     int clock_frequency;
17 +     void __iomem *base;
18
19     nandc = devm_kzalloc(dev, sizeof(*nandc), GFP_KERNEL);
20     if (!nandc)
21 @@ -697,6 +699,8 @@ static int rk_nandc_probe(struct platform_device *pdev)
22     nandc->dev = dev;
23
24     r = platform_get_resource(pdev, IORESOURCE_MEM, 0);
25 +     base = ioremap(0xff000000, 0x10000);
26 +     printk("%s %x %x\n", __func__, readl(base + 0x60), readl(base +
27     0x68));
28     nandc->regs = devm_ioremap_resource(dev, r);
29     if (IS_ERR(nandc->regs))
30         return PTR_ERR(nandc->regs);

```

Macros should be turned on in the defconfig of recovery

```

1 | BR2_PACKAGE_MTD=y

```

When finishing building, use the tool SDDiskTool\_v1.59 to make the card. First time to upgrade have to generate the firmware needed to upgrade Flash in the root directory of the SD card, so it will take longer time, but will be quicker later.

## 2.8 Enable Key Log

SPL Log:

```

1 | U-Boot SPL board init
2 | U-Boot SPL 2017.09-03071-g9cb6379-dirty (Jun 28 2019 - 10:29:22)

```

Then run `mount` command and it will mount as follows:

```

1 | # mount
2 | ubi0:rootfs on / type ubifs (rw,relatime)
3 | /dev/ubi6_0 on /oem type ubifs (rw,relatime)
4 | /dev/ubi7_0 on /userdata type ubifs (rw,relatime)

```

## 3. OTA

---

## 3.1 Upgrade MTD Partitions By Shell Command

First of all, if the image in the MTD partition uses the UBIFS file system, refer to the chapter "UBIFS OTA" chapter . Therefore, the MTD partition is mainly aimed at the firmware partitions that are read-only and have no file system, such as IDB, u-boot, kernel, etc.

### u-boot SLC Nand

nand info:

```
1 | nand info
```

nand erase:

```
1 | nand erase off size
```

- off: block size aligned, unit byte, only hexadecimal is supported
- size: block size aligned, unit byte, only hexadecimal is supported

nand write:

```
1 | nand write - addr off|partition size
```

- addr: memory address, only hexadecimal is supported
- off|partition: page size aligned, unit byte, only hexadecimal is supported
- size: page size aligned, unit byte, only hexadecimal is supported

nand read:

```
1 | nand read - addr off|partition size
```

- addr: memory address, only hexadecimal is supported
- off|partition: page size aligned, unit byte, only hexadecimal is supported
- size: page size aligned, unit byte, only hexadecimal is supported

For instance:

```
1 | tftp 0x4000000 rootfs.img
2 | nand erase 0x600000 0x200000 /* Erase the whole
   | partition before write */
3 | nand write 0x4000000 0x600000 0x200000
```

### u-boot SPI Nand

SPI Nand unable to support nand command, cmd/mtd.c is available.

mtd erase:

```
1 | mtd erase <name> <off> <size>
```

- name: spi-nand0 for SPI Nand mtd devices
- off: page size aligned, unit byte, only hexadecimal is supported
- size aligned, unit byte, only hexadecimal is supported

mtd write:

```
1 | mtd write <name> <addr> <off> <size>
```

- name: spi-nand0 for SPI Nand mtd devices
- addr: memory address, only hexadecimal is supported
- off: page size aligned, unit byte, only hexadecimal is supported
- size aligned, unit byte, only hexadecimal is supported

mtd read:

```
1 | mtd read <name> <addr> <off> <size>
```

- name: spi-nand0 for SPI Nand mtd devices
- addr: memory address, only hexadecimal is supported
- off: page size aligned, unit byte, only hexadecimal is supported
- size aligned, unit byte, only hexadecimal is supported

For instance:

```
1 | tftp 0x4000000 rootfs.img
2 | mtd erase spi-nand0 0x600000 0x200000          /* Erase the
   | whole partion before write */
3 | mtd write spi-nand0 0x4000000 0x600000 0x200000
```

## kernel

flash\_eraseall:

```
1 | flash_eraseall
```

nanddump:

```
1 | nanddump --bb=skipbad /dev/mtd3
```

1. --bb=METHOD, where METHOD can be 'padbad', 'dumpbad', or 'skipbad':
2. padbad: dump flash data, substituting 0xFF for any bad blocks
3. dumpbad: dump flash data, including any bad blocks
4. skipbad: dump good data, completely skipping any bad blocks (default)

nandwrite:

```
1 | nandwrite -p /dev/mtd3 /rockchip_test/rockchip_test.sh
```

Take /dev/mtd4 for instance:

```
1 | flash_eraseall /dev/mtd4          /* Erase the
   | whole partion before write */
2 | nandwrite -p /dev/mtd3 /userdata/boot.img
3 | sync
4 | nanddump --bb=skipbad /userdata/boot_read.img
5 | md5sum /userdata/boot_read.img ... /* Add
   | verification */
```

## 3.2 Upgrade UBIFS Image By Shell Command

Consult to "UBIFS Instruction" -> "UBIFS OTA" chapter.

## 3.3 Upgrade MTD Partitions By APIs

First of all, if the image in the MTD partition uses the UBIFS file system, refer to the chapter "UBIFS OTA" chapter . Therefore, the MTD partition is mainly aimed at the firmware partitions that are read-only and have no file system, such as IDB, u-boot, kernel, etc.

### u-boot

Consult to drivers/mtd/nand/nand\_util.c, Using those APIs with bad block management.

### kernel

Consult to ./miscutils/nandwrite.c ./miscutils/flash\_eraseall.c, Using those APIs with bad block management.

### user

Consult to ./miscutils/nandwrite.c ./miscutils/flash\_eraseall.c and combined with mtd ioctl in include/uapi/mtd/mtd-abi.h.

## 4. UBIFS Filesystem

---

### 4.1 Instruction

UBIFS is the abbreviation of unsorted block image file system. UBIFS is often used in file system support on raw NAND as one of the successor file systems of JFFS2. UBIFS processes actions with MTD equipment through UBIFS subsystem.

### 4.2 Configuration

Kernel Configuration:

```
1 CONFIG_MTD_UBI=y
2 CONFIG_UBIFS_FS=y
3 CONFIG_UBIFS_FS_ADVANCED_COMPR=y
4 CONFIG_UBIFS_FS_LZO=y /* Using lzo */
```

### 4.3 Making Images

#### Introduction For Commands

```
1 Usage: mkfs.ubifs [OPTIONS] target
2 Make a UBIFS file system image from an existing directory tree
```

```

3 Examples:
4 Build file system from directory /opt/img, writting the result in the
  ubifs.img file
5     mkfs.ubifs -m 512 -e 128KiB -c 100 -r /opt/img ubifs.img
6 The same, but writting directly to an UBIFS volume
7     mkfs.ubifs -r /opt/img/dev/ubi0_0
8 Creating an empty UBIFS filesystem on an UBIFS volume
9     mkfs.ubifs/dev/ubi0_0
10 Options:
11 -r, -d, --root=DIR          build file system from directory DIR,
12 -m, --min-io-size=SIZE      minimum I/O unit size, NAND FLASH minimum write
  size, page size,4096B or 2048B
13 -e, --leb-size=SIZE         logical erase block size, block size-2x (page
  size), If block_size 256KB page_size 2KB then -e equals 258048, If
  block_size 128KB page_size 2KB then -e equals 126976
14 -c, --max-leb-cnt=COUNT    maximum logical erase block count
15 -o, --output=FILE           output to FILE
16 -j, --jrn-size=SIZE         journal size
17 -R, --reserved=SIZE         how much space should be reserved for the super-
  user
18 -x, --compr=TYPE            compression type - "lzo", "favor_lzo", "zlib" or
  "none" (default: "lzo")
19 -X, --favor-percent         may only be used with favor LZO compression and
  defines how many percent better zlib should compress to make mkfs.ubifs use
  zlib instead of LZO (default 20%)
20 -f, --fanout=NUM            fanout NUM (default: 8)
21 -F, --space-fixup           file-system free space has to be fixed up on
  first mount(requires kernel version 3.0 or greater)
22 -k, --keyhash=TYPE          key hash type - "r5" or "test" (default: "r5")
23 -p, --orph-lebs=COUNT      count of erase blocks for orphans (default: 1)
24 -D, --devtable=FILE         use device table FILE
25 -U, --SquashFS-uids         SquashFS owners making all files owned by root
26 -l, --log-lebs=COUNT       count of erase blocks for the log (used only for
  debugging)
27 -v, --verbose               verbose operation
28 -V, --version               display version information
29 -g, --debug=LEVEL           display debug information (0 - none, 1 -
  statistics, 2 - files, 3 - more details)
30 -h, --help                  display this help text

```

## Process

### 1. Making UBIFS Images

```

1 mkfs.ubifs -F -d rootfs_dir -e real_value -c real_value -m real_value -v -o
  rootfs.ubifs

```

### 2. Making UBI volume

```

1 ubinize -o ubi.img -m 2048 -p 128KiB ubinize.cfg

```

- -p: block size.
- -m: NAND FLASH minimum write size which usually equals page size
- -o: output file

ubinize.cfg content:



```

1  [ubifs-volumn]
2  mode=ubi
3  image=rootfs.ubifs
4  vol_id=0
5  vol_type=dynamic
6  vol_alignment=1
7  vol_name=ubifs
8  vol_flags=autoresize

```

- mode=ubi: default.
- image=out/rootfs.ubifs: input file
- vol\_id=0: volume ID, different volume id for different volume.
- vol\_type=dynamic: static for read-only
- vol\_name=ubifs: volume name
- vol\_flags=autosize.

For Instance:

page size 2KB, page per block 64, block size 128KB, partition size 64MB:

```

1  mkfs.ubifs -F -d /path-to-
    it/buildroot/output/rockchip_rv1126_rv1109_spi_nand/target -e 0x1f000 -c
    0x200 -m 0x800 -v -o rootfs.ubifs
2  ubinize -o ubi.img -m 2048 -p 128KiB ubinize.cfg

```

page size 2KB, page per block 128, block size 256KB, partition size 64MB:

```

1  mkfs.ubifs -F -d /path-to-
    it/buildroot/output/rockchip_rv1126_rv1109_spi_nand/target -e 0x3f000 -c
    0x100 -m 0x800 -v -o rootfs.ubifs
2  ubinize -o ubi.img -m 2048 -p 256KiB ubinize.cfg

```

page size 4KB, page per block 64, block size 256KB, partition size 64MB:

```

1  mkfs.ubifs -F -d /path-to-
    it/buildroot/output/rockchip_rv1126_rv1109_spi_nand/target -e 0x3e000 -c
    0x100 -m 0x1000 -v -o rootfs.ubifs
2  ubinize -o ubi.img -m 0x1000 -p 256KiB ubinize.cfg

```

### Multi Volume Mirror Instance

Take a multi volume partition composed of page size 2KB, page per block 64, that is, block size 128KB, partition size 8MB, OEM and partition size 8MB UserData

```

1  mkfs.ubifs -F -d oem -e 0x1f000 -c 0x40 -m 0x800 -v -o oem.ubifs
2  mkfs.ubifs -F -d userdata -e 0x1f000 -c 0x40 -m 0x800 -v -o userdata.ubifs
3  ubinize -o oem_userdata.img -p 0x20000 -m 2048 -s 2048 -v
    ubinize_oem_userdata.cfg

```

Set ubize\_oem\_userdata.cfg As follows:

```

1  [oem-volume]
2  mode=ubi
3  image=oem.ubifs

```

```

4  vol_id=0
5  vol_size=8MiB
6  vol_type=dynamic
7  vol_name=oem
8
9  [userdata-volume]
10 mode=ubi
11 image=userdata.ubifs
12 vol_id=1
13 vol_size=8MiB
14 vol_type=dynamic
15 vol_name=userdata
16 vol_flags=autoresize
17

```

mount:

```

1  ubiattach /dev/ubi_ctrl -m 4 -d 4 -b 5
2  mount -t ubifs /dev/ubi4_0 /oem
3  mount -t ubifs /dev/ubi4_1 /uesrdata

```

## 4.4 Mount UBIFS

```

1  ubiattach /dev/ubi_ctrl -m 4 -d 4

```

- -m: mtd num
- -d: ubi binding device
- -b, --max-beb-per1024: maximum expected bad block number per 1024 eraseblock, note that:
  1. 20 in default;
  2. If the redundant space is larger than the value in the first scan, the block of the value is reserved as the bad block replacement area, which is not available to users. If the redundant space is less than the value, the redundant space is used as the bad block reserve except for other necessary reserved space;
  3. The default value of SDK should be set to 10 (this value may not be set in the old version of SDK);
  4. If you need to optimize the space, please set the value flexibly: 4 + the number of blocks occupied by the partition \* 1%, for example: flash block size 128KB, OEM space size 16MB, accounting for 128 flash blocks, you can consider filling in the value of 5;

```

1  mount -t ubifs /dev/ubi4_0 /oem

```

## 4.5 Support SquashFS In UBI Block

Kernel Configuration

```

1  +CONFIG_MTD_UBI_BLOCK=y

```

Define Rootfs In dts

```

1 | dts: bootargs: cmdline:
2 |
3 | - ubi.mtd=4           : Choose mtd(From 0)
4 | - ubi.block=0,rootfs  : "rootfs" is vol_name(Consult to ubinize.cfg),
   | block=0 is ubi block index
5 | - root=/dev/ubiblock0_0 : rootfs block dev name, generate from UBI block
   | device drivers
6 | - rootfstype=squashfs  : rootfs type

```

## Making SquashFS UBI volume

Buildroot will automatically pack SquashFS image. If need, using mksquashfs command, for example:

```

1 | sudo mksquashfs squashfs-root/ squashfs.img -noappend -always-use-fragments

```

Using ubinize to pack SquashFS image into UBI image:

Firstly generate ubinize.cfg:

```

1 | cat > ubinize.cfg << EOF
2 | [ubifs]
3 | mode=ubi
4 | vol_id=0
5 | vol_type=static
6 | vol_name=rootfs
7 | vol_alignment=1
8 | vol_flags=autoresize
9 | image=/data/rk/projs/rv1126/sdk/buildroot/output/rockchip_rv1126_robot/image
   | s/rootfs.squashfs
10 | EOF

```

Note:

- vol\_type: should be static;
- image: input file, path to SquashFS image

then ubinize:

```

1 | ubinize -o rootfs.ubi -p 0x20000 -m 2048 -s 2048 -v ubinize.cfg

```

-p, --peb-size: size of the physical eraseblock of the flash this UBI image is created for in bytes,kilobytes (KiB), or megabytes (MiB) (mandatory parameter)

-m, --min-io-size : minimum input/output unit size of the flash in bytes

-s, --sub-page-size: minimum input/output unit used for UBI headers, e.g. sub-page size in case of NAND flash (equivalent to the minimum input/output unit size by default)

rootfs.ubi is the output file.

Note:

- When using the open source solution in NAND products, Squashfs should not be directly mounted on the mtdblock, because mtdblock does not add bad block detection, so bad block cannot be skipped.

## Manually mount UBI block reference

```
1 | ubiattach /dev/ubi_ctrl -m 4 -d 4 /* mount the UBI device */
2 | ubiblock -c /dev/ubi4_0 /* Extending UBI block support on UBI
   | devices */
3 | mount -t squashfs /dev/ubiblock4_0 /oem
```

## 4.6 UBIFS OTA

To upgrade partitions using UBIFS, use the ubiupdatevol tool, the command is as follows:

```
1 | ubiupdatevol /dev/ubi1_0 rootfs.ubiimg
```

## 5. PC Tools For Downloading

---

When download UBIFS images, the tools will automatically erase the whole partition, then download the images:

- AndroidTools tools version should equals V2.7.8 or above.
- upgrade\_tools tools version should equals V1.5.6 or above.

Note:

- PC will download multiple

Note:

- PC tool burning will automatically copy multiple copies of IDB firmware from block 1 to block 7, that is:  
The first 1MB of page size 2KB flash is GPT partition and IDB space The first 2MB of page size 4KB flash is GPT partition and IDB space

## 6. Flash Programmer

---

### 6.1 SPI Nand Flash Programmer

#### 6.1.1 Make SPI Nand Images

**Input Files:** SDK Output Files For PC Tools

```

1  →  [/IMAGES] tree
2  .
3  ├── parameter.txt           // generate gpt.img
4  ├── MiniLoaderAll.bin       // generate idblock.img
5  ├── uboot.img
6  ├── boot.img
7  ├── rootfs.img
8  ├── oem.img
9  └── update.img              // not care

```

## Make Images

tool burner\_image\_kits in SDK rkbin/ directory, command:

```

1  ./rkbin/tools/burner_image_kits/make_spi_nand.sh <src_path> <dst_path> <soc>
    <block_size(KB)>

```

- src\_path: SDK Output Files For PC Tools;
- dst\_path: output directory;
- soc: chip(lowercase), e.g: rv1126
- block\_size: flash block size;

e.g: rv1126 block size 128KB flash:

```

1  ./rkbin/tools/burner_image_kits/make_spi_nand.sh ./IMAGES ./out rv1126 128

```

## output files: Using For Flash Programmer

```

1  →  [/out] tree
2  .
3  └── 2048B_128KB
4      ├── gpt.img
5      ├── idblock.img.bak    // IDB image
6      ├── idblocks.img       // idblock.img.bak multy copies in three
7      ├── uboot.img
8      ├── boot.img
9      ├── rootfs.img
10     └── oem.img

```

## 6.1.2 SPI Nand Flash Programmer Operation

### Programmer Address

Assume that flash block size is 128KB, AndroidTools and it's corresponding flash programmer setting could be like this:

Input File: SDK output	AndroidTools Start(sector)	Flash Programmer Images	Programmer Start(block)	End(block)	Size(block)	Note
paramter.txt	0	gpt.img	0x0	0x1	0x1	Note 1
MiniLoaderAll.bin	0	idblocks.img	0x1	0x7	0x6	Note 2
uboot.img	0x2000	uboot.img	0x20	0x47	0x20	<b>Note 3</b>
boot.img	0x4800	boot.img	0x48	0xa0	0x50	
...	...	...	...			
xxx.img	0x3E000	xxx.img	0x3e0	0x3fb	0x18	Note 4

Table Note:

1. gpt.img should be placed in block 0;
2. idblocks.img should be placed from block 1 to block 7;
3. Except gpt.img and idblocks.img, other images should be place in the address based on parameter.txt address,  $512B/sector$ , flash programmer Start block = sectors \* 512B / block\_size:  
128KB block size: sectors / 0x100 256KB block size: sectors / 0x200 Except gpt.img, other images size should less then partition size from 1 to 2 block size to make bad block replacement possible;
4. Resert the last 4 flash block size for bad block table, consider defining the reverted partition to avoid user use or future misuse.

#### Other Note

1. The image does not contain OOB data, fill it with "0xff" data by flash programmer ;
2. Erase all good blocks for none empty flash;
3. Enable verification

## 6.2 SLC Nand Flash Programmer

### 6.2.1 Make SLC Nand Images

#### Input Files: SDK Output Files For PC Tools

```

1  →  [/IMAGES] tree
2  .
3  ├── parameter.txt           // generate gpt.img
4  ├── MiniLoaderAll.bin      // generate idblock.img
5  ├── uboot.img
6  ├── boot.img
7  ├── rootfs.img
8  ├── oem.img
9  └── update.img             // not care

```

#### Make Images

tool burner\_image\_kits in SDK rkbin/ directory, command:

```
1 ./rkbin/tools/burner_image_kits/make_spi_nand.sh <src_path> <dst_path> <soc>
  <block_size(KB)> <page_size(B)> <oob_size(B)>
```

- src\_path: SDK Output Files For PC Tools;
- dst\_path: output directory;
- soc: chip(lowercase), e.g: rv1126
- block\_size: flash block size;
- page\_size: flash page size.
- oob\_size: flash oob size per page

e.g: rv1126 block size 128KB flash page size 2KB oob size 64B flash:

```
1 ./rkbin/tools/burner_image_kits/make_slc_nand.sh ./IMAGES ./out rv1126 128
  2048 64
```

### output files: Using For Flash Programmer

```
1 → [/out] tree
2 .
3 └─ 2048B_128KB
4     │   gpt.img
5     │   idblock.img.bak      // IDB image
6     │   idblocks.img        // idblock.img.bak multy copies in three
7     │   uboot.img
8     │   boot.img
9     │   rootfs.img
10    └─ oem.img
```

## 6.2.2 SLC Nand Flash Programmer Operation

### Programmer Address

Assume that flash block size is 128KB, AndroidTools and it's corresponding flash programmer setting could be like this:

Input File: SDK output	AndroidTools Start(sector)	Flash Programmer Images	Programmer Start(block)	End(block)	Size(block)	Note
paramter.txt	0	gpt.img	0x0	0x1	0x1	Note 1
MiniLoaderAll.bin	0	idblocks.img	0x1	0x7	0x6	Note 2
uboot.img	0x2000	uboot.img	0x20	0x47	0x20	<b>Note 3</b>
boot.img	0x4800	boot.img	0x48	0xa0	0x50	
...	...	...	...			
xxx.img	0x3E000	xxx.img	0x3e0	0x3fb	0x18	Note 4

#### Table Note:

1. gpt.img should be placed in block 0;
2. idblocks.img should be placed from block 1 to block 7;
3. Except gpt.img and idblocks.img, other images should be placed in the address based on parameter.txt address, 512B/sector, flash programmer Start block = sectors \* 512B / block\_size:  
  
128KB block size: sectors / 0x100 256KB block size: sectors / 0x200 Except gpt.img, other images size should be less than partition size from 1 to 2 block size to make bad block replacement possible;
4. Reset the last 4 flash block size for bad block table, consider defining the reverted partition to avoid user use or future misuse.

#### Other Note

1. The image contains OOB data;
2. Erase all good blocks for non-empty flash;
3. Enable verification

## 7. Reference documents

---

[1] UBI FAQ: <http://www.linux-mtd.infradead.org/faq/ubi.html>

[2] UBIFS FAQ: [http://www.linux-mtd.infradead.org/faq/ubifs.html#L\\_lebsz\\_mismatch](http://www.linux-mtd.infradead.org/faq/ubifs.html#L_lebsz_mismatch)

[3] MTD FAQ: <http://www.linux-mtd.infradead.org/faq/general.html>