Flash Open Source Solution Develop Guide

ID: RK-KF-YF-314

Release Version: V3.0.0

Release Date: 2022-03-06

Security Level: □Top-Secret □Secret □Internal ■Public

DISCLAIMER

Trademark Statement

Rockchip Electronics Co., Ltd.

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

Website: 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

Preface

Overview

In the SDK with kernel version 4.4 and older, the driver code of RK small capacity storage is self-developed code. Considering that the support of the open source community for parallel port Nand, SPI Nand and SPI Nor has been gradually improved in recent years, and the maintenance of UBIFS is relatively stable, the SDK with kernel version 4.4 will gradually replace the storage support with the open source solution, The SDK with kernel version 5.10 has been converted to full open source support.

Product Version

Chipset	Kernel version
RK3308	Linux 4.4, Linux4.19
RK3568 & RK3566	Linux 4.19
RV1126 & RV1109	Linux 4.19
All SOC support FSPI(SFC)	Linux 5.10

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
V2.1.0	Jon Lin	2021-01-27	Add more UBIFS support
V2.1.1	CWW	2021-02-22	Update the format of the document
V2.2.0	Jon Lin	2021-04-13	Supports u-boot UBIFS and change to use programmer_image_tool
V2.3.0	Jon Lin	2021-07-06	Add IDB layout description, optimize vendor partition description and ECC related description
V2.4.0	Jon Lin	2021-10-29	Support Linux 5.10、Add more details
V2.5.0	Jon Lin	2022-02-28	Upgrade the OTA strategy of SLC Nand
V3.0.0	Jon Lin	2022-03-06	Remove RK3308 patches, Support SPI Nor, Add GPT partition extension concept

Contents

1. Notes on Open Source Solution

1.1 Open Source Solution 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
SPI Nor open source solution	SPI Nor	mtd、mtd block	SquashFS、JFFS2	USB download、SD card download、Flash Programmer

Main points：

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

1.2 Notes on NAND flash selection

Because UBIFS relies on the specific specifications of Nand equipment for production, the produced UBI image cannot be compatible with Nand of different page size and block size.

- Nand material selection specifications shall be consistent.
- SLC Nand in the document refers to the parallel port NAND

1.3 SOC Corresponding Partition Extension

As described in the "Partition Configuration" chapter, the RK SDK has made some special extensions for MTD partitions. Please confirm the extension support according to the chip:

SOC	GPT Partition Extension
RK3308	Y
RV1126 & RV1109	Y
RK3568 & RK3566	Y
RK3588 & RK3588s	Y

2. Build Configuration Changes

2.1 SPL & U-Boot Basic Configuration

The framework:

Note.	Support Device Type	Driver Codes	Flash Framework
SLC Nand open source solution	SLC Nand	drivers/mtd/nand/raw	drivers/mtd/nand/raw
SPI Nand open source solution	SPI Nand	drivers/spi/rockchip_sfc.c	drivers/mtd/nand/spi
SPI Nor open source solution	SPI Nor	drivers/spi/rockchip_sfc.c	drivers/mtd/spi

Defconfig configuration:

Add

```
// MTD support
CONFIG_MTD=y
CONFIG_CMD_MTD_BLK=y
CONFIG_SPL_MTD_SUPPORT=y
CONFIG_MTD_BLK=y
CONFIG_MTD_DEVICE=y

// spi nand support
CONFIG_NAND=y
CONFIG_MTD_SPI_NAND=y
CONFIG_ROCKCHIP_SFC=y
CONFIG_SPL_SPI_FLASH_SUPPORT=y
CONFIG_SPL_SPI_SUPPORT=y

// nand support
CONFIG_NAND=y
CONFIG_CMD_NAND=y
CONFIG_NAND_ROCKCHIP=y
CONFIG_SPL_NAND_SUPPORT=y
CONFIG_SYS_NAND_U_BOOT_LOCATIONS=y
```

```
CONFIG_SYS_NAND_U_BOOT_OFFS=0x8000
CONFIG_SYS_NAND_U_BOOT_OFFS_REDUND=0x10000

// rkfw firmware uboot trust address（Needless in fit format）
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
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

```
CONFIG_RKFLASH=y
CONFIG_RKNANDC_NAND=y
CONFIG_RKSFC_NAND=y
CONFIG_RKSFC_NOR=y
```

2.2 Kernel Basic Configuration

2.2.1 SLC Nand Open Source Solution

The framwork:

Note.	Support Device Type	Driver Codes	Flash Framework
SLC Nand open source solution	SLC Nand	drivers/mtd/nand/raw	drivers/mtd/nand/raw

Configuration：

```
CONFIG_RK_NANDC_NAND=n   /* It's not compatible */
CONFIG_MTD_NAND_ROCKCHIP_V6=y /* NandC v6 is depending on TRM NANDC-
>NANDC_NANDC_VER register, 0x00000801 */
# CONFIG_MTD_NAND_ROCKCHIP_V9=y /* NandC v9  is depending on TRM NANDC-
>NANDC_NANDC_VER register, 0x56393030 */
CONFIG_MTD_CMDLINE_PARTS=y
```

2.2.2 SPI Nand Open Source Solution

2.2.2.1 Linux 4.19 and former version

The framework:

Note.	Support Device Type	Driver Codes	Flash Framework
SPI Nand open source solution	SPI Nand	drivers/rkflash	drivers/rkflash
SPI Nor open source solution	SPI Nor	drivers/rkflash	drivers/rkflash

Defconfig configuration：

```
CONFIG_RK_FLASH=y

CONFIG_RK_SFC_NAND=y     /* SPI Nand flash */
CONFIG_RK_SFC_NAND_MTD=y /* SPI Nand flash and partitions is register as mtd
device,  otherwise block devices(rkflash0pn)  */
CONFIG_RK_SFC_NOR=y      /* SPI Nor flash */
CONFIG_RK_SFC_NOR_MTD=y  /* SPI Nor flash and partitions is register as mtd
device,  otherwise block devices(rkflash0pn) */
CONFIG_MTD_CMDLINE_PARTS=y
```

dts configuration：

```
&sfc {
    status = "okay";

    flash@0 {
        compatible = "spi-nand";    # compatible = "jedec,spi-nor"; for spinor
        reg = <0>;
        spi-max-frequency = <75000000>;
        spi-rx-bus-width = <4>;
        spi-tx-bus-width = <1>;
    };
};
```

2.2.2.2 Linux 5.10

Note.	Support Device Type	Driver Codes	Flash Framework
SPI Nand open source solution	SPI Nand	drivers/spi/spi-rockchip-sfc.c	drivers/mtd/nand/spi
SPI Nor open source solution	SPI Nor	drivers/spi/spi-rockchip-sfc.c	drivers/mtd/spi-nor

Defconfig configuration：

```
CONFIG_SPI_ROCKCHIP_SFC=y
```

dts :

```
&sfc {
    status = "okay";

    flash@0 {
        compatible = "spi-nand";    # compatible = "jedec,spi-nor"; for spinor
        reg = <0>;
        spi-max-frequency = <75000000>;
        spi-rx-bus-width = <4>;
        spi-tx-bus-width = <1>;
    };
};
```

2.3 Partition Configuration

2.3.1 MTD Partition Basics

Open source solution flash devices is register as MTD devices, and only support mtd partition.

Notes for Nand Partition Definition：

- 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 u-boot 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

2.3.2 GPT Partition Extension

Some SDK schemes of RK support parsing GPT in u-boot and generating the information required by MTD mtdparts, which is finally transmitted to the kernel through cmdline. You can confirm the corresponding chip platform by referring to the chapter "SOC Corresponding Partition Extension".

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

  ```
  TYPE: GPT
  ```

2.4 Vendor Storage

u-boot defconfig configuration:

It is selected by CONFIG_MTD_BLK = y。

u-boot source code：

```
arch/arm/mach-rockchip/vendor.c
```

kernel defconfig configuration：

```
CONFIG_ROCKCHIP_MTD_VENDOR_STORAGE=y
```

kernel source code：

```
drivers/soc/rockchip/mtd_vendor_storage.c
```

partition configuration:

Add “vnvm” partition.

Note For Partition Definition:

- vnvm is recommended follow with IDB partition, and make 4 flash blocks size.

3. OTA

3.1 OTA input source files

It's recommended to use the images which are made by programmer tools for OTA. Refer to chapter "Flash Programmer" for making programmer images.

3.2 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：

```
nand info
```

nand erase：

```
nand erase off size
```

- off：block size aligned, unit byte, only hexadecimal is supported, not included in OOB size
- size：block size aligned, unit byte, only hexadecimal is supported, not included in OOB size

nand write：

```
nand write.raw[.noverify] - addr off|partition [count]
```

- addr：memory address, only hexadecimal is supported, not included in OOB size
- off|partition：page size aligned, unit page, only hexadecimal is supported
- count: unit byte, only hexadecimal is supported, included in OOB size

nand read：

```
nand read.raw - addr off|partition [count]
```

- addr：memory address, only hexadecimal is supported, not included in OOB size
- off|partition：page size aligned, unit page, only hexadecimal is supported
- count: unit byte, only hexadecimal is supported, included in OOB size

For instance:

```
tftp 0x4000000 rootfs.img
nand erase 0x600000 0x200000                          /* Erase the whole partion
before write */
nand write.raw 0x4000000 0x600000 0x1000
```

kernel SLC Nand

flash_erase：

```
flash_erase       /* for example: flash_erase /dev/mtd1 0 0 */
```

nanddump：

```
nanddump -o -n -p --bb=skipbad /dev/mtd3
```

- --bb=skipbad：Skip bad block
- -o: read data with oob
- -n: read data without ecc

nandwrite：

```
nandwrite -o -n -p /dev/mtd3 /rockchip_test/rockchip_test.sh
```

- -o: input file with oob
- -n: write without ecc

For instance:

```
flash_erase /dev/mtd4 0 0                       /* Erase the whole partion before
write */
nandwrite -o -n -p /dev/mtd3 /userdata/boot.img
sync
nanddump -o -n -p --bb=skipbad /userdata/boot_read.img
md5sum /userdata/boot_read.img ...
```

u-boot SPI Nand

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

mtd erase:

```
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:

```
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:

```
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:

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

kernel

flash_eraseall：

```
flash_erase       /* for example: flash_erase /dev/mtd1 0 0 */
```

nanddump：

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

- --bb=skipbad：Skip bad block

nandwrite：

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

Take /dev/mtd4 for instance:

```
flash_erase /dev/mtd4 0 0                                    /* Erase the whole
partion before write */
nandwrite -p /dev/mtd3 /userdata/boot.img
sync
nanddump -p --bb=skipbad /userdata/boot_read.img
md5sum /userdata/boot_read.img ...                            /* Add
verification */
```

3.3 Upgrade UBIFS Image By Shell Command

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

3.4 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.
- For a complete write with less data (it is recommended to write less than 2KB data on each power on), you can consider using the corresponding interface of MTD to block device in RK SDK, source code drivers/mtd/mtd_blk.c, the block abstract interface has the following characteristics:

  Regardless of the amount of data in a single write request, the flash block corresponding to the data will be erased. Therefore, for fragmented and frequent write behavior, calling this interface will affect the life of flash.

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 conbined with mtd ioctrl in include/uapi/mtd/mtd-abi.h.

4. Filesystem

4.1 UBIFS Filesystem

4.1.1 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.

4.1.2 Configuration

Kernel Configuration：

```
CONFIG_MTD_UBI=y
CONFIG_UBIFS_FS=y
CONFIG_UBIFS_FS_ADVANCED_COMPR=y
CONFIG_UBIFS_FS_LZO=y /* Using lzo */
```

4.1.3 Image Making And Mounting

4.1.3.1 Image Making

Introduction For Commands

```
Usage: mkfs.ubifs [OPTIONS] target
Make a UBIFS file system image from an existing directory tree
Examples:
Build file system from directory /opt/img, writting the result in the ubifs.img
file
        mkfs.ubifs -m 512 -e 128KiB -c 100 -r /opt/img ubifs.img
The same, but writting directly to an UBIFS volume
        mkfs.ubifs -r /opt/img/dev/ubi0_0
Creating an empty UBIFS filesystem on an UBIFS volume
        mkfs.ubifs/dev/ubi0_0
Options:
-r, -d, --root=DIR         build file system from directory DIR,
-m, --min-io-size=SIZE     minimum I/O unit size, NAND FLASH minimum write size,
page size,4096B or 2048B
-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
-c, --max-leb-cnt=COUNT    maximum logical erase block count
-o, --output=FILE          output to FILE
-j, --jrn-size=SIZE        journal size
-R, --reserved=SIZE        how much space should be reserved for the super-user
-x, --compr=TYPE           compression type - "lzo", "favor_lzo", "zlib" or
"none" (default: "lzo")
-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%)
-f, --fanout=NUM           fanout NUM (default: 8)
-F, --space-fixup          file-system free space has to be fixed up on first
mount(requires kernel version 3.0 or greater)
-k, --keyhash=TYPE         key hash type - "r5" or "test" (default: "r5")
-p, --orph-lebs=COUNT      count of erase blocks for orphans (default: 1)
-D, --devtable=FILE        use device table FILE
-U, --SquashFS-uids        SquashFS owners making all files owned by root
-l, --log-lebs=COUNT       count of erase blocks for the log (used only for
debugging)
-v, --verbose              verbose operation
-V, --version              display version information
```

```
-g, --debug=LEVEL          display debug information (0 - none, 1 - statistics, 2
- files, 3 - more details)
-h, --help                 display this help text
```

Process

1. Making UBIFS Images

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

2. Making UBI volume

```
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：

```
[ubifs-volumn]
mode=ubi
image=rootfs.ubifs
vol_id=0
vol_type=dynamic
vol_alignment=1
vol_name=ubifs
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：

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

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

```
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
ubinize -o ubi.img -m 2048 -p 256KiB ubinize.cfg
```

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

```
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
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

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

Set ubize_ oem_ userdata.cfg As follows:

```
[oem-volume]
mode=ubi
image=oem.ubifs
vol_id=0
vol_size=8MiB
vol_type=dynamic
vol_name=oem

[userdata-volume]
mode=ubi
image=userdata.ubifs
vol_id=1
vol_size=8MiB
vol_type=dynamic
vol_name=userdata
vol_flags=autoresize
```

mount：

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

4.1.3.2 UBIFS Image Making Of Empty Partition

```
ubiformat -y /dev/mtd4
ubiattach /dev/ubi_ctrl -m 4 -d 4
ubimkvol /dev/ubi4 -N userdata -m /* -N specifies the volume name, - M
dynamically adjusts the partition device autorisize to the maximum */
```

4.1.3.3 UBIFS dts Root Mount

```
ubi.mtd=4 root=ubi0:rootfs rootfstype=ubifs
```

4.1.3.4 UBIFS Partition Command Mount

```
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. Partition image pre production: partition redundancy flash block < --max-beb-per1024 actual value < --max-beb-per1024 set value, that is, the actual value may be smaller than the set value
    3. Command to make empty partition as UBI image: - - max-beb-per1024, the actual value is equal to the set value
    4. The default value of SDK can be set to 10 (this value may not be set in the old version of SDK)
    5. 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;

```
mount -t ubifs /dev/ubi4_0 /oem
```

4.1.3.5 UBI Image Partition Overhead

After the UBI image is mounted on the file system, the effective space is less than the partition size. There are mainly UBIFS redundant information and the loss of reserved blocks for bad block replacement.

Accurate calculation

```
UBI overhead = (B + 4) * SP + 0 * (P - B - 4) /* the space cannot be obtained by
users */

P - The total number of physical blocks removed on the MTD device
SP - physical erase block size, typically 128KB or 256Kb
SL - logical erase block, i.e. mkfs - e parameter value, usually block_ size - 2
* page_ size
B - Flash blocks reserved for bad block replacement, related to the ubiattach - b
parameter
O - overhead associated with storing EC and vid file headers in bytes, i.e. 0 =
SP - sl
```

General case 1

Flash block size 128KB, page size 2KB, 128 MB size, ubiattach - b is reserved by default of 20;

```
SP = block size = 128KB
SL = 128kb - 2 * 2KB = 124KB
B = --max-beb-per1024 * n_ 1024 = 20 * 1 = 20
O = 128KB -124KB = 4KB

UBI overhead = (20 + 4) * 128KB + 4KB * (P - 20 - 4) = 2976KB + 4KB * P
```

If the corresponding partition is 32MB, that is, P = 256, then the final UBI overhead = 2976kb + 4KB * 256 = 4000kb

General case 2

Flash block size 128KB, page size 2KB, 256 MB size, ubiattach - B reserved default 20;

```
SP = block size = 128KB
SL = 128kb - 2 * 2KB = 124KB
B = --max-beb-per1024 * n_ 1024 = 20 * 2 = 40
O = 128KB -124KB = 4KB

UBI overhead = (40 + 4) * 128KB + 4KB * (P - 40 - 4) = 5456KB + 4KB * P
```

If the corresponding partition is 32MB, that is, P = 256, then the final UBI overhead = 5456kb + 4KB * 256 = 6456kb

Detailed reference: flash space overhead chapter http://www.linux-mtd.infradead.org/doc/ubi.html#L_overhead

4.1.4 Support SquashFS In UBI Block

Kernel Configuration

```
+CONFIG_MTD_UBI_BLOCK=y
```

Define Rootfs In dts

```
dts: bootargs: cmdline:

- ubi.mtd=4              : Choose mtd(From 0)
- ubi.block=0,rootfs     : "rootfs" is vol_name(Consult to ubinize.cfg), block=0
is ubi block index
- root=/dev/ubiblock0_0  : rootfs block dev name, generate from UBI block device
drivers
- rootfstype=squashfs    : rootfs type
```

Making SquashFS UBI volume

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

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

Using ubinize to pack SquashFS image into UBI image:

Firstly generate ubinize.cfg：

```
cat > ubinize.cfg << EOF
[ubifs]
mode=ubi
vol_id=0
vol_type=static
vol_name=rootfs
vol_alignment=1
vol_flags=autoresize
image=/data/rk/projs/rv1126/sdk/buildroot/output/rockchip_rv1126_robot/images/roo
tfs.squashfs
EOF
```

Note:

- vol_type: should be static;
- image: input file, path to SquashFS image

then ubinize：

```
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

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

4.1.5 Optimization Of UBIFS Space Size

As can be seen from the above description, the mirror free space can be optimized through the following three points:

1. Select the appropriate -- max-beb-per1024 parameter, and refer to point 5 of the "- b parameter details" section of "Image making of empty partition"
2. Use UBI multi volume technology to share part of UBIFS redundant overhead. Refer to the description of multi volume production in "Image making"
3. Use the SquashFS supported by UBI block. Refer to the chapter "Support SquashFS In UBI Block"

UBIFS minimum partition:

```
Minimum block num = 4 (fixed reservation) + B + 17 / * B - Flash blocks reserved
for bad block replacement, related to ubiattach - b parameter*/
```

It can be judged by printing log when ubiattach, for example:

```
ubi4: available PEBs: 7, total reserved PEBs: 24, PEBs reserved for bad PEB
handling: 20 /* B = 20 */
```

If the partition available PEBS + total reserved PEBS < minimum block num, an error will be reported when mounting:

```
mount: mounting /dev/ubi4_ 0 on userdata failed: Invalid argument
```

4.1.6 UBIFS OTA

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

```
ubiupdatevol /dev/ubi1_0 rootfs.ubifs
```

Note：

- rootfs.ubifs is made by mkfs.ubifs tool

4.1.7 Support U-Boot UBIFS

Under uboot, UBIFS only supports read operation, no write/erase operation.

SLC Nand Patches

Refer to RK3308 support, and the patch is as follows:

```
CONFIG_CMD_UBI=y

diff --git a/include/configs/rk3308_common.h b/include/configs/rk3308_common.h
index 1c2b9e4461..bc861acde8 100644
--- a/include/configs/rk3308_common.h
+++ b/include/configs/rk3308_common.h
@@ -57,6 +57,9 @@
 #define CONFIG_USB_FUNCTION_MASS_STORAGE
 #define CONFIG_ROCKUSB_G_DNL_PID        0x330d
+#define MTDIDS_DEFAULT "nand0=rk-nand"
+#define MTDPARTS_DEFAULT "mtdparts=rk-
nand:0x100000@0x400000(uboot),0x100000@0x500000(trust),0x600000@0x600000(boot),0x
3000000@0xc00000(rootfs),0x1200000@0x4000000(oem),0x9f60000@0x6000000(userdata)"
+
 #ifdef CONFIG_ARM64
 #define ENV_MEM_LAYOUT_SETTINGS \
        "scriptaddr=0x00500000\0" \
```

SPI Nand Patches

Refer to RK3568 support, and the patch is as follows:

```
diff --git a/include/configs/rk3568_common.h b/include/configs/rk3568_common.h
index cce44b52a8..eb93455c62 100644
--- a/include/configs/rk3568_common.h
+++ b/include/configs/rk3568_common.h
@@ -92,6 +92,9 @@
        "run distro_bootcmd;"
 #endif
+#define MTDIDS_DEFAULT "spi-nand0=spi-nand0"
+#define MTDPARTS_DEFAULT "mtdparts=spi-
nand0:0x100000@0x400000(vnvm),0x440000@0x500000(uboot),0x1600000@0xa00000(boot),0
x4400000@0x2000000(rootfs),0x1b60000@0x6400000(userdata)"
+
 /* rockchip ohci host driver */
 #define CONFIG_USB_OHCI_NEW
 #define CONFIG_SYS_USB_OHCI_MAX_ROOT_PORTS     1
(END)
```

menuconfig Enable Configuration

```
#define CONFIG_CMD_UBI = y
```

Mounting

Take rootfs partition as an example, refer to doc for doc/README.ubi.

```
mtdpart
ubi part rootfs
ubifsmount ubi0:rootfs
ubifsls
```

4.2 JFFS2 Filesystem

4.2.1 Introduction

The full name of JFFS2 is journaling flash file system version 2. Its function is to manage the journaling file system implemented on MTD devices. Compared with other storage device storage schemes, JFFS2 is not prepared to provide a conversion layer that allows traditional file systems to use such devices. It will only implement the file system with log structure directly on the MTD device. JFFS2 will scan the log contents of MTD device during installation and re-establish the file system structure itself in RAM.

4.2.2 Configuration

Kernel Configuration：

```
CONFIG_JFFS2_FS=y
```

4.2.3 JFFS2 Image Making

For example：

```
mkfs.jffs2 -r data/-o data.jffs2 -e 0x10000 --pad=0x400000 -s 0x1000 -n
Options:
 --pad [=SIZE]            Pad output to SIZE bytes with 0xFF. If SIZE is
                          not specified, the output is padded to the end of
                          the final erase block
  -r, -d, --root=DIR      Build file system from directory DIR (default: cwd)
  -s, --pagesize=SIZE     Use page size (max data node size) SIZE.
                          Set according to target system's memory management
                          page size (default: 4KiB)
  -e, --eraseblock=SIZE   Use erase block size SIZE (default: 64KiB)
  -n, --no-cleanmarkers   Don't add a cleanmarker to every eraseblock
```

Note：

- --pad: format with partition size
- -e：erase size setting as 64KB，4KB for Linux 5.10
- -s：4KB in default

5. PC Tools For Downloading

5.1 PC Tools For Downloading - GPT Partition Extension

Tools version requirements:

- AndroidTools tools version should equals V2.7.8 or above。
- upgrade_tools tools version should equals V1.5.6 or above。

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

- The PC tool burning of nor device will make double backup of IDB firmware, which corresponds to the following:

  64KB offset for the first copy

  The second IDB is followed by the first IDB, aligned size 64KB

- During the development process, if the GPT partition is adjusted and the firmware is upgraded, please go to the maskrom mode to upgrade, otherwise UBIFS may be mounted abnormally
- When download UBIFS images, the tools will automatically erase the whole partition, then download the images

6. Flash Programmer

6.1 SPI Nand Flash Programmer - GPT Partition Extension

6.1.1 Make SPI Nand Images

Input Files： SDK Output Files For PC Tools

```
[/IMAGES] tree
.
├── parameter.txt
├── MiniLoaderAll.bin
├── uboot.img
├── boot.img
├── rootfs.img
├── oem.img
└── update.img                  // For make programmer imagers
```

Make Images

tool programmer_image_tool is in SDK rkbin/tools/，command：

```
./tools/programmer_image_tool  --help
NAME
        programmer_image_tool - creating image for programming on flash
SYNOPSIS
        programmer_image_tool [-iotbpsvh]
DESCRIPTION
        This tool aims to convert firmware into image for programming
        From now on,it can support slc nand(rk)|spi nand|nor|emmc.
OPTIONS:
          -i    input firmware
          -o    output directory
          -t    storage type,range in[SLC|SPINAND|SPINOR|EMMC]
          -b    block size,unit KB
          -p    page size,unit KB
          -s    oob size,unit B
          -2    2k data in one page
          -l    using page linked l
```

Assume that: rv1126 block size 128KB page size 2KB flash:

```
./tools/programmer_image_tool  -i update.img -b 128 -p 2 -t spinand -o out
input firmware is 'update.img'
block size is '128'
page size is '2'
flash type is 'spinand'
output directory is 'out'
writing idblock...
start to write partitions...gpt=1
preparing gpt saving at out/gpt.img
writing gpt...OK
preparing trust saving at out/trust.img
writing trust...OK
preparing uboot saving at out/uboot.img
writing uboot...OK
```

```
preparing boot saving at out/boot.img
writing boot...OK
preparing rootfs saving at out/rootfs.img
writing rootfs...OK
preparing recovery saving at out/recovery.img
writing recovery...OK
preparing oem saving at out/oem.img
writing oem...OK
preparing userdata:grow saving at out/userdata.img
writing userdata:grow...OK
preparing misc saving at out/misc.img
writing misc...OK
creating programming image ok.
```

Note:

- 4KB page size programmer_image_tool add -2 parameter

output files: Using For Flash Programmer

```
out
├── boot.img
├── gpt.img
├── idblock.img            //Making multiple backup images of
idblock_mutli_copies.img on demand
├── misc.img
├── oem.img
├── recovery.img
├── rootfs.img
├── trust.img
├── uboot.img
└── userdata.img
```

Note：

- IDB multi backup command reference, usually double backup can be:

```
cat out/idblock.img > out/idblock_mutli_copies.img   // 1 copy
cat out/idblock.img >> out/idblock_mutli_copies.img  // 2 copies
```

6.1.2 SPI Nand Flash Programmer OperationProgrammer 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	idblock_mutli_copies.img	0x1	0x7	0x6	Note 2
uboot.img	0x2000	uboot.img	0x20	0x47	0x20	**Note 3**
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. idblock_mutli_copies.img should be placed from block 1 to block 7, the image size is limit to 7 blocks；
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. Because the RK spinand controller FSPI (formerly known as SFC) does not integrate the ECC module, it needs to rely on the ECC function of the device itself. Therefore, the image used by the programmer does not contain oob data, the oob space is filled by the programmer itself, and the ECC function of the spinand device is enabled in the tool interface of the programmer;
2. Erase all good blocks for none empty flash;
3. Enable verification.

6.2 SLC Nand Flash Programmer - GPT Partition Extension

6.2.1 Make SLC Nand Images

Input Files： SDK Output Files For PC Tools

```
[/IMAGES] tree
.
├── parameter.txt
├── MiniLoaderAll.bin
├── uboot.img
├── boot.img
├── rootfs.img
├── oem.img
└── update.img                  // For make programmer imagers
```

Make Images

tool programmer_image_tool is in SDK rkbin/tools/，command：

```
./tools/programmer_image_tool  --help
NAME
        programmer_image_tool - creating image for programming on flash
SYNOPSIS
        programmer_image_tool [-iotbpsvh]
DESCRIPTION
        This tool aims to convert firmware into image for programming
        From now on,it can support slc nand(rk)|spi nand|nor|emmc.
OPTIONS:
          -i    input firmware
          -o    output directory
          -t    storage type,range in[SLC|SPINAND|SPINOR|EMMC]
          -b    block size,unit KB
          -p    page size,unit KB
          -s    oob size,unit B
          -2    2k data in one page
          -l    using page linked l
```

Assume that： rv1126 block size 128KB page size 2KB oob size 128 flash:

```
./tools/programmer_image_tool -i update.img -b 128 -p 2 -s 128 -t slc -o out
input firmware is 'update.img'
block size is '128'
page size is '2'
oob size is '128'
flash type is 'slc'
2k data page on.
output directory is 'out'
writing idblock...
start to write partitions...gpt=1
preparing gpt saving at out/gpt.img
writing gpt...OK
preparing trust saving at out/trust.img
writing trust...OK
preparing uboot saving at out/uboot.img
writing uboot...OK
preparing boot saving at out/boot.img
writing boot...OK
preparing rootfs saving at out/rootfs.img
writing rootfs...OK
preparing recovery saving at out/recovery.img
writing recovery...OK
preparing oem saving at out/oem.img
writing oem...OK
preparing userdata:grow saving at out/userdata:grow.img
writing userdata:grow...OK
preparing misc saving at out/misc.img
writing misc...OK
creating programming image ok.
```

Note:

- Add '-l' tag for RK3326/PX30/RK3568

- Using 4KB page size flash in RV1126/RK3326/PX30/RK3308, add -2 to programmer_image_tool

output files: Using For Flash Programmer

```
out
├── boot.img
├── gpt.img
├── idblock.img            //Making multiple backup images of
idblock_mutli_copies.img on demand
├── misc.img
├── oem.img
├── recovery.img
├── rootfs.img
├── trust.img
├── uboot.img
└── userdata.img
```

Note：

- IDB multi backup command reference, usually double backup can be:

```
cat out/idblock.img > out/idblock_mutli_copies.img   // 1 copy
cat out/idblock.img >> out/idblock_mutli_copies.img  // 2 copies
```

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	idblock_mutli_copies.img	0x1	0x7	0x6	Note 2
uboot.img	0x2000	uboot.img	0x20	0x47	0x20	**Note 3**
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. idblock_mutli_copies.img should be placed from block 1 to block 7, the image size is limit to 7 blocks；
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 contain OOB data；
2. Erase all good blocks for none empty flash；
3. Enable verification

6.3 SPI Nor Flash Programmer - GPT Partition Extension

6.3.1 Make SPI Nor Images

Input Files：SDK Output Files For PC Tools

```
[/IMAGES] tree
.
├── parameter.txt
├── MiniLoaderAll.bin
├── uboot.img
├── boot.img
├── rootfs.img
├── oem.img
└── update.img                  // For make programmer imagers
```

Make Images

tool programmer_image_tool is in SDK rkbin/tools/，command：

```
./tools/programmer_image_tool  --help
NAME
        programmer_image_tool - creating image for programming on flash
SYNOPSIS
        programmer_image_tool [-iotbpsvh]
DESCRIPTION
        This tool aims to convert firmware into image for programming
        From now on,it can support slc nand(rk)|spi nand|nor|emmc.
OPTIONS:
          -i    input firmware
          -o    output directory
          -t    storage type,range in[SLC|SPINAND|SPINOR|EMMC]
          -b    block size,unit KB
          -p    page size,unit KB
          -s    oob size,unit B
          -2    2k data in one page
          -l    using page linked l
```

For example：

```
./tools/programmer_image_tool -i update.img  -t SPINOR -o ./out
input firmware is 'update.img'
flash type is 'SPINOR'
output directory is './out'
writing idblock...
start to write partitions...gpt=1
preparing gpt at 0x00000000
writing gpt...OK
preparing trust at 0x00002800
writing trust...OK
preparing uboot at 0x00002000
writing uboot...OK
preparing boot at 0x00009800
writing boot...OK
preparing rootfs at 0x0000c800
writing rootfs...OK
preparing recovery at 0x00003800
writing recovery...OK
preparing misc at 0x00003000
writing misc...OK
creating programming image ok.
```

output files: Using For Flash Programmer

```
tree
.
└── out_image.img
```

6.3.2 SPI Nor Flash Programmer Operation

The image output from programmer_image_tool is burned to SPI Nor 0 address.

7. FAQ

7.1 Nand Flash Infomation

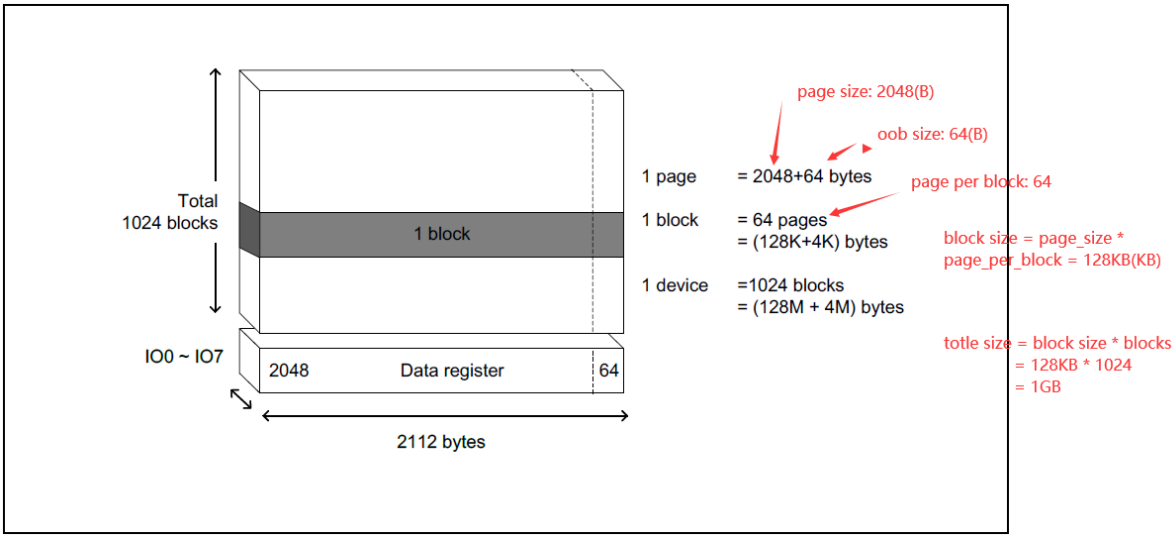
Making UBIFS images、IDB images(Pre loader) should depend on the concrete Nand flash infomation, including follow information:

- 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:



7.2 IDB Layout

IDB is the packaging firmware of ddr.bin and spl.bin, and the first level firmware after maskrom. Its layout is as follows:

- The upgrade of PC tools is stored in flash block 1 ~ 7 and filled with multiple backups;
- Suggestion of programmer image layout: refer to the description of "Programmer address" in "Flash Programmer" charpter.

8. 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

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