Flash Open Source Solution 开发指南

文件标识：RK-KF-YF-314

发布版本：V3.8.0

日期：2024-03-04

文件密级：□绝密 □秘密 □内部资料 ■公开

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd.

地址： 福建省福州市铜盘路软件园A区18号

网址： www.rock-chips.com

客户服务电话： +86-4007-700-590

客户服务传真： +86-591-83951833

客户服务邮箱： fae@rock-chips.com

前言

概述

在内核版本为 4.4 及更旧的 SDK 中，RK 小容量存储的驱动代码为自开发的代码，考虑到近年来开源社区对于并口 Nand、SPI Nand 和 SPI Nor 的支持逐步完善，且 UBIFS 的维护也较为稳定，所以内核版本 4.4 后的 SDK，逐步替换存储支持为开源方案，内核版本为 5.10 后的 SDK 已转为全开源支持。

产品版本

芯片名称	内核版本
所有支持 FSPI（旧称 SFC）的芯片	Linux 4.4、Linux4.19、Linux 5.10、Linux6.1

读者对象

本文档（本指南）主要适用于以下工程师：

技术支持工程师

软件开发工程师

修订记录

版本号	作者	修改日期	修改说明
V1.0.0	HKH	2019-06-20	初始版本
V1.0.1	HKH	2019-11-11	增加sd卡升级说明
V1.0.2	Ruby Zhang	2020-07-08	格式修订
V1.1.0	Jair Wu	2020-07-10	新增u-boot编译说明
V2.0.0	Jon Lin	2020-10-19	完善驱动配置等详细信息
V2.0.1	Jon Lin	2020-11-27	增加 UBIFS 多卷支持、增减 ubiattach 参数说明
V2.1.0	Jon Lin	2021-01-27	添加更多 UBIFS 支持说明
V2.1.1	CWW	2021-02-22	格式修订
V2.2.0	Jon Lin	2021-04-13	u-boot 支持 UBIFS，更改使用 programmer_image_tool 制作烧录器镜像
V2.3.0	Jon Lin	2021-07-06	增加 IDB Layout 说明、优化 vendor 分区说明、增加 ECC 相关说明
V2.4.0	Jon Lin	2021-10-29	增加 Linux 5.10 支持、增加更多文档说明
V2.5.0	Jon Lin	2022-02-28	优化 SLC Nand OTA 方案
V3.0.0	Jon Lin	2022-03-06	去除 RK3308 补丁支持、增加 SPI Nor 支持、新增 GPT 分区扩展概念
V3.1.0	Jon Lin	2022-07-28	优化文档结构、添加 SPI Nor shell 命令 OTA 介绍
V3.2.0	Jon Lin	2023-06-05	完善驱动配置信息
V3.3.0	Jon Lin	2023-08-15	增加 SPI Nor ENV 方案烧录器烧录章节、增加测试项目
V3.4.0	Jon Lin	2023-12-04	添加SPI Flash内核速度计算和Nand产品寿命问题解释
V3.5.0	Jon Lin	2024-01-08	添加异常掉电问题说明

版本号	作者	修改日期	修改说明
V3.6.0	Jon Lin	2024-01-10	增加信号测试指导及更多常见问题
V3.7.0	Jon Lin	2024-01-18	增加芯片及对应软件方案关键信息、
V3.8.0	Jon Lin	2024-03-03	增加 RK3576、内核 Linux6.1 版本支持

目录

1. 开源方案须知

1.1 开源方案特性

确认以下开源实现方案特性:

简称	主要支持的颗粒类型	注册设备类型	主要支持文件系统	支持的烧录方式
SLC Nand 开源方案（并口 Nand）	SLC Nand	mtd 设备、ubiblock	SquashFS、UBIFS	USB 升级、SD 卡升级、烧录器升级
SPI Nand 开源方案	SPI Nand	mtd 设备、ubiblock	SquashFS、UBIFS	USB 升级、SD 卡升级、烧录器升级
SPI Nor 开源方案	SPI Nor	mtd 设备、mtd block 设备	SquashFS、JFFS2	USB 升级、SD 卡升级、烧录器升级

主要确认：

- 选用的颗粒类型
- 文件系统是否符合需求

1.2 Nand Flash 颗粒选型须知

由于 UBIFS 依赖 Nand 设备的具体规格来进行制作，所以制作出来的固件无法兼容不同 page size 和 block size 的颗粒。

- Nand 物料选型规格要一致
- 该文档中 SLC Nand 指并口 nand

1.3 芯片对应分区扩展支持

如 “分区配置” 章节中所述，RK SDK 为 MTD 分区做了一些特殊扩展，请根据芯片确认扩展支持情况：

芯片	GPT 分区扩展	ENV 分区扩展
RK3308	Y	N
RV1126 & RV1109	Y	N
RK3568 & RK3566	Y	N
RK3588 & RK3588s	Y（default）	Y（option）
RV1106/RV1103	N	Y
RK3528	Y	N
RK3562	Y	N
RK3576	Y	N

1.4 芯片 FSPI/SFC 关键特性

仅统计使用开源存储的 SOC 芯片：

SOC	FSPI/SFC IO rate	FSPI/SFC data io lines
RK3308	<= 48MHz（推荐）	x1\x4
RV1126 & RV1109	<= 125MHz	x1\x4
RK3568/RK3566	<= 150MHz，固定频点 OSC\50\75\100\125\150 MHz	x1\x4
RK3588/RK3588s	<= 150MHz	x1\x4
RV1106/RV1103	<= 150MHz，支持 125MHz 和 150MHz，但由于分频限制，二者之间频点不支持	x1\x4
RK3528	<= 150MHz	x1\x4
RK3562	<= 150MHz	x1\x4
RK3576	<= 133MHz，由于分频限制，最高到 125MHz	x1\x4

说明：

- IP 支持 x4，同样兼容 x2，只是开源框架不支持 dual spi
- IP 虽然支持较高频率，但 SDK 默认配置通常为较低频率， 推荐 80MHz，可以根据项目和外接的颗粒实际支持速率做调整
- Quad SPI Nand 通常速率上限为 133MHz、Quad SPI Nor 通常速率上限为 133MHz（最高有 166MHz）
- Octal SPI Flash 通常速率上限为 200MHz

1.5 芯片分区扩展及 IDB 规划

1.5.1 芯片分区扩展方案

芯片	GPT 分区扩展	ENV 分区扩展
RK3308	默认方案（兼容方案1）	N
RV1126/RV1109	默认方案（兼容方案2）	N
RK3568/RK3566	默认方案	N
RK3588/RK3588s	默认方案	N
RV1106/RV1103	N	默认方案
RK3528	默认方案	N
RK3562	默认方案	N
RK3576	默认方案	N

说明：

- 兼容方案的修改点详细查看 “GPT 分区扩展方案 IDB 规划” 描述

1.5.2 ENV 分区扩展方案 IDB 规划

存储类型	idb 起始地址	多备份说明
SLC Nand	256KB	要求镜像双备份
SPI Nand	256KB	要求镜像多备份
SPI Nor	64KB	要求镜像双备份

1.5.3 GPT 分区扩展方案 IDB 规划

存储类型	idb 起始地址	多备份说明
SLC Nand	flash block size（128KB 或 256KB）	要求工具支持双备份，大小对齐 flash block size
SPI Nand	flash block size（128KB 或 256KB）	要求工具支持双备份，大小对齐 flash block size
SPI Nor	64KB	要求工具支持双备份，大小对齐 64KB

说明：

- PC 升级工具支持 IDB 双备份烧录

- 兼容方案1：RK3308 方案 SPI Nor idb 仅支持单备份、SPI Nand block 1~8 为 idb 多备份区域，最多存储 5 份
- 兼容方案2：RV1126/RV1109 方案 SPI Nor idb 起始地址 32KB，仅支持单备份

2. 存储软件驱动配置

2.1 SPL & U-Boot 基础配置

defconfig 配置：

```
// MTD 驱动支持
CONFIG_MTD=y
CONFIG_CMD_MTD_BLK=y
CONFIG_SPL_MTD_SUPPORT=y
CONFIG_MTD_BLK=y
CONFIG_MTD_DEVICE=y

// spi nand 驱动支持
CONFIG_MTD_SPI_NAND=y
CONFIG_ROCKCHIP_SFC=y
CONFIG_SPL_SPI_FLASH_SUPPORT=y
CONFIG_SPL_SPI_SUPPORT=y

// nand 驱动支持
CONFIG_NAND=y
CONFIG_CMD_NAND=y
CONFIG_NAND_ROCKCHIP=y /* NandC v6 可根据 TRM NANDC->NANDC_NANDC_VER 寄存器确认，
0x00000801 */
//CONFIG_NAND_ROCKCHIP_V9=y  /* NandC v9 可根据 TRM NANDC->NANDC_NANDC_VER 寄存器
确认，0x56393030，例如：RK3326/PX30 为此版本 */
CONFIG_SPL_NAND_SUPPORT=y
CONFIG_SYS_NAND_U_BOOT_LOCATIONS=y

// spi nor 驱动支持
CONFIG_CMD_SF=y
CONFIG_CMD_SPI=y
CONFIG_SPI_FLASH=y
CONFIG_SF_DEFAULT_MODE=0x1
CONFIG_SF_DEFAULT_SPEED=50000000
CONFIG_SPI_FLASH_GIGADEVICE=y
CONFIG_SPI_FLASH_MACRONIX=y
CONFIG_SPI_FLASH_WINBOND=y
CONFIG_SPI_FLASH_MTD=y
CONFIG_ROCKCHIP_SFC=y
CONFIG_SPL_SPI_SUPPORT=y
CONFIG_SPL_MTD_SUPPORT=y
CONFIG_SPL_SPI_FLASH_SUPPORT=y
```

去除 rkflash/rknand宏配置：

```
CONFIG_RKFLASH=n
CONFIG_RKNAND=n
```

驱动文件：

```
./drivers/mtd/nand/raw          //SLC Nand 主控驱动及协议层
./drivers/mtd/nand/spi          //SPI Nand 协议层
./drivers/spi/rockchip_sfc.c    //SPI Flash 主控驱动
./drivers/mtd/spi               //SPI Nor 协议层
```

说明：

- SPI Nor 颗粒，软件已支持的 SPI Nand 颗粒可以查询源码 `drivers/mtd/spi/spi-nor-ids.c`
- 以 Winbound SPI Nand 颗粒为例，软件已支持的 SPI Nand 颗粒可以查询源码 `drivers/mtd/nand/spi/winbond.c`，其余颗粒支持在同级目录下
- 可以通过 flash id 或 flash 丝印来匹配源码

2.2 Kernel 基础配置

2.2.1 SLC Nand

框架简介：

简称	主要支持的颗粒类型	主控驱动	flash 框架
SLC Nand 开源方案	SLC Nand	drivers/mtd/nand/raw	drivers/mtd/nand/raw

defconfig 配置：

```
CONFIG_RK_NANDC_NAND=n    /* 不兼容 */
CONFIG_MTD_NAND_ROCKCHIP_V6=y /* NandC v6 可根据 TRM NANDC->NANDC_NANDC_VER 寄存器确认，0x00000801 */
# CONFIG_MTD_NAND_ROCKCHIP_V9=y /* NandC v9 可根据 TRM NANDC->NANDC_NANDC_VER 寄存器确认，0x56393030 */
CONFIG_MTD_CMDLINE_PARTS=y
```

2.2.2 SPI Nor\Nand

2.2.2.1 Linux 4.19 及旧版本

框架简介：

简称	主要支持的颗粒类型	主控驱动	flash 框架
SPI Nand 开源方案	SPI Nand	drivers/rkflash	drivers/rkflash
SPI Nor 开源方案	SPI Nor	drivers/rkflash	drivers/rkflash

defconfig 配置：

```
CONFIG_RK_FLASH=y

CONFIG_RK_SFC_NAND=y      /* SPI Nand flash */
CONFIG_RK_SFC_NAND_MTD=y /* SPI Nand flash 及其分区注册为 mtd 设备，不选中则注册为普通 block 设备（rkflash0pn） */
CONFIG_RK_SFC_NOR=y       /* SPI Nor flash */
CONFIG_RK_SFC_NOR_MTD=y  /* SPI Nor flash 及其分区注册为 mtd 设备，不选中则注册为普通 block 设备（rkflash0pn） */
CONFIG_MTD_CMDLINE_PARTS=y
```

dts 配置：

```
&sfc {
    assigned-clocks = <&cru SCLK_SFC>; # 选择 dtsi sfc 设备节点中的 clk_sfc 对应时钟源修改 IO 速率
    assigned-clock-rates = <100000000>;
    status = "okay";
};
```

2.2.2.2 Linux 5.10 及更高版本

框架简介

简称	主要支持的颗粒类型	主控驱动	flash 框架
SPI Nand 开源方案	SPI Nand	drivers/spi/spi-rockchip-sfc.c	drivers/mtd/nand/spi
SPI Nor 开源方案	SPI Nor	drivers/spi/spi-rockchip-sfc.c	drivers/mtd/spi-nor

说明：

- 以 Winbound SPI Nor 颗粒为例，软件已支持的 SPI Nor 颗粒可以查询源码 `drivers/mtd/spi-nor/winbond.c`，其余颗粒支持在同级目录下
- 以 Winbound SPI Nand 颗粒为例，软件已支持的 SPI Nand 颗粒可以查询源码 `drivers/mtd/nand/spi/winbond.c`，其余颗粒支持在同级目录下
- 可以通过 flash id 或 flash 丝印来匹配源码
- 软件通过 flash id 识别特定颗粒，如果颗粒在源码中的配置及控制器节点在 dts 中的 bus-width 配置都设置为 quad/octal，则驱动就会选中 quad/octal 传输方案

defconfig 配置：

```
CONFIG_SPI_ROCKCHIP_SFC=y
CONFIG_MTD_SPI_NAND=y
CONFIG_MTD_SPI_NOR=y
```

dts 配置：

```
&sfc {
    sfc-cs-gpios = <&gpio4 RK_PB0 GPIO_ACTIVE_LOW>, <&gpio4 RK_PA5
GPIO_ACTIVE_LOW>; # 扩展 gpio 为 CSN
    status = "okay";

    flash@0 {
        compatible = "spi-nand";     # compatible = "jedec,spi-nor"; for spinor
        reg = <0>;
        spi-max-frequency = <75000000>; # 修改 IO 速率
        spi-rx-bus-width = <4>;          # 阅读 "芯片 FSPI/SFC 关键特性" 确认支持的线
宽
        spi-tx-bus-width = <1>;          # 阅读 "芯片 FSPI/SFC 关键特性" 确认支持的线
宽，x4 软件自动降级为兼容性更好的 x1 传输
    };
};
```

2.3 分区配置

2.3.1 MTD 分区基础

flash 开源方案注册存储设备为 MTD 设备，仅支持解析规范的 mtd partition 分区表信息，通常开源社区是通过在内核 dts cmdline 定义 mtdparts 分区表。

2.3.2 GPT 分区扩展

RK 部分 SDK 方案支持在 u-boot 中解析 GPT 并生成 MTD mtdparts 所需的信息，最终通过 cmdline 传递给内核，可以通过查阅 “芯片对应分区扩展支持” 章节确认对应芯片平台，u-boot 内实现 GPT 转 mtdparts 的代码在 `u-boot/drivers/mtd_blk.c`。

GPT 分区表也是通过 parameter 文件配置，结构和 MTD Partition 类似，差异的地方有四个：

1. 设置 TYPE 为 GPT
2. 没有定义 parameter 分区（如果定义，也不会使用）
3. 最后一个分区需要增加关键字”grow“
4. MTD 方案无需指定 rootfs 的 uuid

```
FIRMWARE_VER:8.1
MACHINE_MODEL:RK3326
MACHINE_ID:007
MANUFACTURER: RK3326
MAGIC: 0x5041524B
ATAG: 0x00200800
MACHINE: 3326
CHECK_MASK: 0x80
PWR_HLD: 0,0,A,0,1
TYPE: GPT
CMDLINE:mtdparts=rk29xxnand:0x00002000@0x00004000 (uboot),0x00002000@0x00006000
(trust),0x00002000@0x00008000 (misc),0x00008000@0x0000a000
(resource),0x00010000@0x00012000 (kernel),0x00010000@0x00022000
(boot),0x00020000@0x00032000 (recovery),0x00038000@0x00052000
(backup),0x00002000@0x0008a000 (security),0x000c0000@0x0008c000
(cache),0x00300000@0x0014c000 (system),0x00008000@0x0044c000
(metadata),0x000c0000@0x00454000 (vendor),0x00040000@0x00514000
(oem),0x00004400@0x00554000 (frp),-@0x00554400 (userdata:grow)
```

GPT 分区表升级流程：

1. 工具读取 parameter 里面的分区定义
2. 从 loader 处获取存储设备的容量
3. 修改最后一个分区大小并创建 gpt 分区表文件
4. 烧写分区表到存储设备的 0 地址和 - 33（末尾）地址
5. parameter 文件本身不会被烧写到存储设备中。

2.3.3 ENV 分区扩展

RK 部分 SDK 方案支持设定 ENV 分区，支持 u-boot 标准的 ENV 环境变量，最终通过 cmdline 传递给内核，可以通过查阅 “芯片对应分区扩展支持” 章节确认对应芯片平台。

制作 ENV 镜像是通过 u-boot 目录下的 `./tools/mkenvimage` 工具完成，配置文件自定义，如定义 env.txt：

```
mtdparts=spi-
nand0:256K(env),256K@256K(idblock),4M(uboot),8M(boot),72M(rootfs),32M(userdata)
,-(media)
sys_bootargs=ubi.mtd=4 root=ubi0:rootfs rootfstype=ubifs
```

说明：

- SPI Nand 制作命令参考：./tools/mkenvimage -s 0x40000 -p 0x0 -o env.img env_.txt
- SPI Nor 制作命令参考：./tools/mkenvimage -s 0x10000 -p 0x0 -o env.img env.txt
- sys_bootargs 以实际 SDK 配置为准，参考开发手册做相应修改

2.3.4 Nand 分区规划须知

须知：

- SLC Nand 及 SPI Nand 开源方案每个分区应预留出 3~4 个 flash block size 的冗余空间，以便遇到坏块时，有冗余空间可替换，尤其注意 u-boot 分区是否做到了空间预留，如果分区大小超过 10MB，建议增大冗余空间，计算方式 4 + 分区所占 block 数 * 1%，例如：flash block size 128KB，oem 空间大小 16MB，占 128 flash block，可以考虑填值 5
- 分区起始地址应做 flash block size 对齐
- SLC Nand 及 SPI Nand 开源方案预留最后 4 个 flash block 给全局坏块表，所以最后一个用户分区不应包括该区域
    - 如果最后一个分区包含这些块，将会包含 4 个人为标记的坏块，等于分区有效空间固定减小 4 flash block，如果最后一个镜像远小于分区大小，就不会造成实际问题，但依旧不建议包含坏块表区域

2.4 Vendor Storage

Vendor Storage 是设计来存放一些非安全小数据，比如 SN、MAC 等。

配置

定义 "vnvm" 命令的分区：

- 建议 vnvm 分区规划在 IDB 分区后，uboot 分区前，size 建议为 4 flash block，SPI Nor 256KB，SPI Nand 1MB（兼容 128KB/256KB）
- 如 parameter.txt 添加 SPI Nor vnvm 项，`0x00000200@0x00000800(vnvm)`，SPI Nand vnvm 项 `0x00000800@0x00001800(vnvm)`
- 如 env.txt 添加 SPI Nor vnvm 项，`256K@1M(vnvm)`，SPI Nand vnvm 项 `1M@3M(vnvm)`

u-boot defconfig 配置:

CONFIG_MTD_BLK = y 时默认开启，关键代码 `arch/arm/mach-rockchip/vendor.c`，接口 `./arch/arm/include/asm/arch-rockchip/vendor.h`

kernel defconfig 配置：

宏开关 `CONFIG_ROCKCHIP_MTD_VENDOR_STORAGE=y` ，关键代码 `drivers/soc/rockchip/mtd_vendor_storage.c`，接口 `./include/linux/soc/rockchip/rk_vendor_storage.h`

用户工具

支持 PC 工具、量产工具和内核用户层接口，详细参考 《Rockchip_Application_Notes_Storage_CN.pdf》文档。

3. PC 工具烧录

3.1 PC 工具烧录 —— GPT 分区扩展

工具版本要求：

- AndroidTools 工具版本必须在 V2.7.8 或者之上。
- upgrade_tools 应在 V1.5.6 或者之上

特殊逻辑处理：

- Nand 设备 PC 工具烧录会自动拷贝多份 IDB 固件在 block 1~ block 7，对应如下：
  - page size 2KB flash 前1MB 为 gpt 分区和 IDB 空间
  - page size 4KB flash 前2MB 为 gpt 分区和 IDB 空间
- Nor 设备 PC 工具烧录会做 IDB 固件双备份，对应如下：
  - 64KB 处存放第一份
  - 第一份后紧接着存放第二份 IDB，size 对齐为 64KB
- 开发过程中，如果 gpt 分区做了调整，升级固件请转到 maskrom mode 下升级，否则可能导致 UBIFS 挂载异常
- 烧写工具支持 UBIFS 镜像烧写，识别到固件为 UBIFS，分区，再烧写该分区

3.2 PC 工具烧录 —— ENV 分区扩展

使用 SocToolKit 工具，支持 Windows 和 Linux 版本，并有匹配的量产工具。

4. OTA

4.1 OTA 须知

- 在线升级时明确先卸载目标分区的 UBIFS 文件系统，然后再升级
- MTD 方案，且使用 parameter 分区信息的设备，如果产品需要升级 gpt 分区表、idb 镜像，parameter.txt 文件应添加 gpt/idb 分区：
  - block size 128KB Nand：0x00000100@0x00000000(gpt),0x00000700@0x00000100(idb)
  - block size 256KB Nand：0x00000200@0x00000000(gpt),0x00000e00@0x00000200(idb)
  - SPI Nor：0x00000080@0x00000000(gpt),0x00000780@0x00000080(idb)

4.2 镜像来源

OTA 仅支持预制作镜像，镜像制作参考“烧录器烧录” “制作镜像” 相关章节。

4.3 命令行在线升级 MTD 分区

首先要确定，如果 MTD 分区内的镜像使用 UBIFS 文件系统，要参考 “Shell 命令升级 UBIFS 镜像分区” 章节，所以该 MTD 分区主要针对 IDB、u-boot、kernel 等只读没有文件系统的固件分区。

4.3.1 SLC Nand u-boot

nand info：

```
nand info
```

nand erase：

```
nand erase off size
```

- off：block size 对齐，单位为 byte，仅支持十六进制，不计入 oob size，例如 block 128KB data size + 8KB oob size 计算值为 128KB
- size：block size 对齐，单位为 byte，仅支持十六进制，不计入 oob size，例如 block 128KB data size + 8KB oob size 计算值为 128KB

nand write：

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

- addr：memory 地址，仅支持十六进制
- off|partition：page size 对齐，单位为 byte，仅支持十六进制，不计入 oob size，例如 block 128KB data size + 8KB oob size 计算值为 128KB
- count：单位为 page，仅支持十六进制，计入 oob size，例如 page 为 0x800 Byte data size + 0x80 Byte oob size 则 count 计算公式为 file_size / 0x880
- 默认带写回读 verify

nand read：

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

- addr：memory 地址，仅支持十六进制
- off|partition：page size 对齐，单位为 byte，仅支持十六进制，不计入 oob size，例如 block 128KB data size + 8KB oob size 计算值为 128KB
- count：单位为 page，仅支持十六进制，计入 oob size，例如 page 为 0x800 Byte data size + 0x80 Byte oob size 则 count 计算公式为 file_size / 0x880

针对一个分区的升级，建议操作顺序：

```
tftp 0x4000000 rootfs.img
nand erase 0x600000 0x200000                          /* 升级分区时先擦除整个分区 */
nand write.raw 0x4000000 0x600000 0x1000
```

4.3.2 SLC Nand kernel

flash_erase：

```
flash_erase       /* 例如: flash_erase /dev/mtd1 0 0 */
```

nanddump：

```
nanddump -o -n --bb=skipbad /dev/mtd3 -f /userdata/boot_read.img
```

- --bb=skipbad：跳过坏块
- -o：读出 oob 数据
- -n：读数据不带 ecc

nandwrite：

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

- -o：输入文件带 oob 数据
- -n：写数据不带 ecc

针对一个分区的升级，建议操作顺序：

```
flash_erase /dev/mtd4 0 0                          /* 升级分区时先擦除整个分区 */
nandwrite -o -n -p /dev/mtd4 /userdata/boot.img
sync
nanddump -o -n --bb=skipbad /dev/mtd4 -f /userdata/boot_read.img
md5sum /userdata/boot_read.img ...                 /* 建议添加校验 */
```

4.3.3 SPI Nand u-boot

SPI Nand 无法支持 nand cmd 命令，可选用 cmd/mtd.c 接口，依旧能跳过坏块。

mtd erase:

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

- name：spi-nand0 for SPI Nand mtd device
- off：page size 对齐，单位为 byte，仅支持十六进制
- size: block size 对齐，单位为 byte，仅支持十六进制

mtd write:

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

- name：spi-nand0 for SPI Nand mtd device
- addr：memory 地址，仅支持十六进制
- off：page size 对齐，单位为 byte，仅支持十六进制
- size: page size对齐，单位为 byte，仅支持十六进制

mtd read:

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

- name：spi-nand0 for SPI Nand mtd device
- addr：memory 地址，仅支持十六进制
- off：page size 对齐，单位为 byte，仅支持十六进制
- size: page size 对齐，单位为 byte，仅支持十六进制

针对一个分区的升级，建议操作顺序：

```
tftp 0x4000000 rootfs.img
mtd erase spi-nand0 0x600000 0x200000                    /* 升级分区时先擦除整个
分区 */
mtd write spi-nand0 0x4000000 0x600000 0x200000
```

4.3.4 SPI Nand kernel

flash_erase：

```
flash_erase       /* 例如：flash_erase /dev/mtd1 0 0 */
```

nanddump：

```
nanddump --bb=skipbad /dev/mtd3 -f /userdata/boot_read.img
```

- --bb=skipbad：跳过坏块

nandwrite：

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

针对一个分区的升级，建议操作顺序：

```
flash_erase /dev/mtd4 0 0                       /* 升级分区时先擦除整个分区 */
nandwrite -p /dev/mtd4 /userdata/boot.img
sync
nanddump --bb=skipbad /dev/mtd3 /userdata/boot_read.img
md5sum /userdata/boot_read.img ...              /* 建议添加校验 */
```

4.3.5 SPI Nor u-boot

建议选用 mtd 接口，cmd/sf.c 可作为替代方案。
mtd erase:

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

- name：nor0 for SPI Nor mtd device
- off：sector size 对齐， 4KB per sector，单位为 byte，仅支持十六进制
- size：sector size 对齐，4KB per sector，单位为 byte，仅支持十六进制

mtd write:

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

- name：nor0 for SPI Nor mtd device
- addr：memory 地址，仅支持十六进制
- off：最小支持对齐到 byte，单位为 byte，仅支持十六进制
- size：最小支持对齐到 byte，单位为 byte，仅支持十六进制

mtd read:

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

- name：nor0 for SPI Nor mtd device
- addr：memory 地址，仅支持十六进制

- off：最小支持对齐到 byte，单位为 byte，仅支持十六进制
- size：最小支持对齐到 byte，单位为 byte，仅支持十六进制

针对一个分区的升级，建议操作顺序：

```
tftp 0x4000000 rootfs.img
mtd erase nor0 0x600000 0x200000                                    /* 升级分区时先擦除整个分区
*/
mtd write nor0 0x4000000 0x600000 0x200000
```

4.3.6 SPI Nor kernel

使用 mtd_debug 接口

4.4 命令行在线升级 UBIFS 镜像分区

参考 "UBIFS 文件系统" -> "UBIFS OTA" 章节。

4.5 函数接口升级 MTD 分区

首先要确定，如果 MTD 分区内的镜像使用 UBIFS 文件系统，要参考 “Shell 命令升级 UBIFS 镜像分区” 章节，所以该 MTD 分区主要针对 IDB、u-boot、kernel 等只读没有文件系统的固件分区。

u-boot

- 建议参考 drivers/mtd/nand/nand_util.c，使用有坏块识别的读写擦除接口。
- 对于数据量较少的一次完整写行为（建议每次上电写数据量少于 2KB），可以考虑使用 RK SDK 中 mtd 转 block 设备相应的接口，源码 drivers/mtd/mtd_blk.c，该 block 抽象接口有以下特点：
    - 无论单次写请求的数据量多大，都会擦除数据对应的 flash block，所以对于零碎且频繁的写行为如果调用该接口将会影响 flash 的寿命。

kernel

建议参考 mtd-utils 开源 mtd 源码中的 ./miscutils/nandwrite.c ./miscutils/flash_eraseall.c，使用有坏块识别的读写擦除接口。

user

原则上依旧是参考 ./miscutils/nandwrite.c ./miscutils/flash_eraseall.c，结合 mtd 设备阶段支持的系列 ioctrl 命令，完成有坏块识别的读写擦除应用代码。

mtd device 支持的 ioctrl 选项在 include/uapi/mtd/mtd-abi.h 中。

5. 文件系统

5.1 UBIFS 文件系统

5.1.1 简介

UBIFS 是 Unsorted Block Image File System 的简称，常应用于 raw nand 上的文件系统支持，作为 jffs2 的后继文件系统之一。UBIFS 通过 UBIFS 子系统处理与 MTD 设备之间动作。

5.1.2 配置

内核支持：

```
CONFIG_MTD_UBI=y
CONFIG_UBIFS_FS=y
CONFIG_UBIFS_FS_ADVANCED_COMPR=y
CONFIG_UBIFS_FS_LZO=y /* 建议选用 lzo 压缩 */
```

5.1.3 镜像制作及挂载

5.1.3.1 离线预制作镜像

命令详细说明

```
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, 待制作的文件系统目录
-m, --min-io-size=SIZE     minimum I/O unit size，最小输入输出大小，NAND FLASH 的最小读写单元，一般为 page size，有 4096 或 2048
-e, --leb-size=SIZE        logical erase block size 逻辑可擦出块大小，为 block size-2x (page size)，如 block_size 256KB page_size 2KB 应设置 -e 258048，如 block_size 128KB page_size 2KB 应设置 -e 126976
-c, --max-leb-cnt=COUNT    maximum logical erase block count 最大逻辑可擦出块数目，autoresize 时文件系统的的上限
-o, --output=FILE          output to FILE 输出文件名
-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),如果是通过 u-boot 烧写需要使能此功能。
-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
```

流程

1. 制作 UBIFS 镜像（通常只需配置以下参数）

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

2. 制作为 flash 烧写格式

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

- -p：block size。
- -m：NAND FLASH 的最小读写单元，一般为 page size
- -o：输出的 ubi.img 文件

其中，ubinize.cfg 配置：

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

- mode=ubi： 是强制参数，当前不能输入别的值，保留为以后扩展功能
- image=out/rootfs.ubifs：此文件为源文件
- vol_id=0：表示卷的 ID，UBI 镜像可能包含多个卷，这个用来区别不同的卷
- vol_type=dynamic：表示当前卷类型是可读写的，只读为 static
- vol_name=ubifs：卷的名称
- vol_flags=autosize：表示卷的大小是可扩展的

实例

page size 2KB，page per block 64，即 block size 128KB，分区 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，即 block size 256KB，分区 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，即 block size 256KB，分区 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
```

多卷镜像实例

以 page size 2KB，page per block 64，即 block size 128KB，分区大小 8MB oem 和 分区大小 8MB userdata 复合的多卷分区为例：

```
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
```

设置 ubinize_oem_userdata.cfg 如下：

```
[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
```

挂载分区：

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

注意：

- 多卷中的独立分区无法单独简单升级，即 flash 读写接口直接升级多卷中特定卷，所以多卷内的分区 OTA 需求和频率应该相近

5.1.3.2 内核分区在线低格

```
umount userdata/
ubidetach -m 4
ubiformat -y /dev/mtd4
ubiattach /dev/ubi_ctrl -m 4 -d 4
ubimkvol /dev/ubi4 -N userdata -m                                    # -N 指定卷名，-m 将分区设备 autorisize 可动态调整到最大
mount -t ubifs /dev/ubi4_0 /userdata
```

5.1.3.3 UBIFS 分区命令挂载

将 MTD 设备连接到 UBI 设备：

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

- -m：指定 mtd 分区序号
- -d：绑定后的 ubi 设备编号，建议与 mtd 分区序号一致
- -b, --max-beb-per1024：每1024个eraseblock预期的最大坏块数，注意：
    1. 不带参数，默认为 20
    2. 分区镜像预制作：分区冗余 flash block < --max-beb-per1024 实际值 < --max-beb-per1024 设定值，即实际值可能比设定值小
    3. 命令制作空分区为 UBI 镜像：--max-beb-per1024 实际值等于设定值
    4. SDK 默认值可设定为 10（可能旧版本 SDK 该值未设定）
    5. 如需优化空间，请灵活设定该值：4 + 分区所占 block 数 * 1%，例如：flash block size 128KB，oem 空间大小 16MB，占 128 flash block，可以考虑填值 5

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

5.1.3.4 UBIFS 分区自动挂载

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

5.1.3.5 UBI 镜像分区损耗

UBI 镜像挂载文件系统后有效空间小于分区大小，主要存在 UBIFS 冗余信息和坏块替换所需的预留块的损耗。

精确计算

```
UBI overhead = (B + 4) * SP + 0 * (P - B - 4) /* 该空间用户无法获取 */
P - MTD 设备上物理除块的总数
SP - 物理擦除块大小,通常为 128KB 或 256KB
SL - 逻辑擦除块，即 mkfs 时 -e 参数值,通常为 block_size - 2 * page_size
B - 为坏块替换预留的 flash blocks,与 ubiattach- b参数相关
O - 与以字节为单位存储 EC 和 VID 文件头有关的开销, i.e. 0 = SP - SL
```

通用案例1

flash block size 128KB, page size 2KB，128 MB size, ubiattach -b 预留默认 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
```

以对应分区为 32MB 为例，即 P = 256，那么最终损耗为 UBI overhead = 2976KB + 4KB * 256 = 4000KB

通用案例2

flash block size 128KB, page size 2KB，256 MB size, ubiattach -b 预留默认 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
```

以对应分区为 32MB 为例，即 P = 256，那么最终损耗为 UBI overhead = 5456KB + 4KB * 256 = 6456KB

详细参考：Flash space overhead 章节 http://www.linux-mtd.infradead.org/doc/ubi.html#L_overhead

5.1.4 UBI Block 支持 SquashFS

内核配置

```
+CONFIG_MTD_UBI_BLOCK=y
```

dts 指定 rootfs

```
dts 中 bootargs 参数指定了 cmdline 中相关 ubi 的参数。

- ubi.mtd=4                 : 选择 MTD 设备（从0开始）
- ubi.block=0,rootfs        : "rootfs" 是 vol_name (参考 ubinize.cfg), block=0 指定了ubi block的序号
- root=/dev/ubiblock0_0     : 指定 rootfs 分区名；由 ubi block 驱动根据 ubi.block 参数生成的块设备
- rootfstype=squashfs       : 指定文件系统类型
```

实例参考：

```
ubi.mtd=3 ubi.block=0,rootfs root=/dev/ubiblock0_0 rootfstype=squashfs
```

注意：

1. 如果 mtd 分区由 u-boot 解析 GPT 获取，则与 parameter.txt 中的分区一一对应，从 mtd0 或者 mtdblock0 开始计数；
2. mtd 为 char 设备，mtdblock 为 block 设备

制作 SquashFS UBI volume

Buildroot 默认会打包 SquashFS image。如需要另行打包，可使用mksquashfs命令，例如：

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

使用ubinize工具将 SquashFS 镜像打包成ubi image。

首先生成ubinize.cfg文件：

```
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/r
ootfs.squashfs
EOF
```

其中:

- vol_type 为 static 指定为只读
- image 指定 SquashFS image 的路径

然后使用ubinize打包image：

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

-p：指定flash的物理擦除块大小

-m：指定flash的最小输入输出单元,当为nor flash时，此值应指定为1，当为nand flash时此值应指定为页面大小

-s：指定子页大小,当为nor flash时，此值应指定为1，当为nand flash时需指定此值为nand flash的子页大小

参数

输出 rootfs.ubi 文件即升级所用文件。

注意：

- Nand 产品使用开源方案应将 SquashFS 挂载在 UBI block 上而非 mtdblock，因为 mtdblock 没有加入坏块探测，所以无法跳过坏块.

手动挂载 UBI block 参考

```
ubiattach /dev/ubi_ctrl -m 4 -d 4    /* 先挂载 UBI 设备 */
ubiblock -c /dev/ubi4_0              /* UBI 设备上扩展 UBI block 支持 */
mount -t squashfs /dev/ubiblock4_0 /oem
```

5.1.5 镜像空间大小优化

通过以上描述可知，主要通过以下三点来优化镜像可用空间：

1. 选择合适的 --max-beb-per1024 参数，参考 “命令制作空分区 UBI 镜像及镜像挂载” 章节的 “-b 参数详述” 第 5 点
2. 使用 UBI 多卷技术来共享部分 UBIFS 冗余开销，参考 “镜像制作” 中多卷制作说明
3. 使用 UBI block 支持下的 SquashFS，参考 “UBI Block 支持 SquashFS” 章节

UBIFS 最小分区:

```
Minimum block num = 4（固定预留） + B + 17  /* B - 为坏块替换预留的 flash blocks,与 ubiattach - b 参数相关，UBIFS_MIN_LEB_CNT 等于 17 */
```

可通过 ubiattach 时打印 log 来判断，例如：

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

如果分区 available PEBs + total reserved PEBs < Minimum block num，则挂载时会报错：

```
mount: mounting /dev/ubi4_0 on userdata failed: Invalid argument
```

5.1.6 UBIFS OTA

升级使用 UBIFS 的分区应使用 ubiupdatevol 工具，参考，命令：

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

注意：

- rootfs.ubifs 为 mkfs.ubifs 命令所制作的镜像，非 ubinize 制作的最终烧录镜像

5.1.7 U-Boot 下支持 UBIFS

u-boot 下 UBIFS 仅支持文件读操作，不支持文件写或删除操作。

SLC Nand 补丁参考

参考 RK3308 支持，补丁如下：

```
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),
0x3000000@0xc00000(rootfs),0x1200000@0x4000000(oem),0x9f60000@0x6000000(userdat
a)"
+
 #ifdef CONFIG_ARM64
 #define ENV_MEM_LAYOUT_SETTINGS \
        "scriptaddr=0x00500000\0" \
```

SPI Nand 补丁参考

参考 RK3568 支持，补丁如下：

```
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)
,0x4400000@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 开启宏配置

```
#define CONFIG_CMD_UBI = y
```

挂载命令

以 rootfs 分区为例，详细参考 doc/README.ubi。

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

说明

- 如 “GPT 分区扩展” 章节所述，uboot cmdline 传递 mtdparts 信息到内核，但 MTDPARTS_DEFAULT 定义例外，所以内核 cmdline 应添加 mtdparts 定义以实现内核 mtd 分区定义

5.2 JFFS2 文件系统支持

5.2.1 简介

JFFS2 的全名为 Journalling Flash FileSystem Version 2（闪存日志型文件系统第 2 版），其功能就是管理在 MTD 设备上实现的日志型文件系统。与其他的存储设备存储方案相比，JFFS2 并不准备提供让传统文件系统也可以使用此类设备的转换层。它只会直接在 MTD 设备上实现日志结构的文件系统。JFFS2 会在安装的时候，扫描 MTD 设备的日志内容，并在 RAM 中重新建立文件系统结构本身。

5.2.2 配置

内核配置：

```
CONFIG_JFFS2_FS=y
```

5.2.3 镜像制作

参考以下是实例：

```
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
```

说明：

- --pad：定为与分区大小一致
- -e：erase size 设置为 64KB，内核版本 5.10 及更高版本，应关闭 4KB 擦除支持，MTD_SPI_NOR_USE_4K_SECTORS = n
- -s：默认设置为 4KB

6. 烧录器烧录

6.1 SPI Nand 镜像烧录 —— GPT 分区扩展

6.1.1 SPI Nand 制作烧录镜像

输入文件：SDK 输出的用于 PC 工具烧录的镜像

```
[/IMAGES] tree
.
├── parameter.txt
├── MiniLoaderAll.bin
├── uboot.img
├── boot.img
├── rootfs.img
├── oem.img
└── update.img                    // 用以制作烧录镜像
```

制作镜像

工具 programmer_image_tool 在 SDK rkbin/ 目录下，命令说明如下：

```
./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
```

例如： rv1126 block size 128KB page size 2KB flash:

256KB 4KB * 64

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

说明：

- 4KB page size programmer_image_tool 添加 -2 参数

输出文件：用于烧录器烧的镜像

```
out
├── boot.img
├── gpt.img
├── idblock.img             // 自行根据需求制作 idblock_mutli_copies.img 的多备份镜像
├── misc.img
├── oem.img
├── recovery.img
├── rootfs.img
├── trust.img
├── uboot.img
└── userdata.img
```

注释：

- IDB 多备份命令参考，通常双备份即可：

```
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 烧录器烧录

烧录地址

假定 block size 为 128KB 的 flash，PC 烧录工具及相应烧录器镜像烧录信息对比如下：

烧录器镜像源文件：SDK 默认输出镜像	PC 烧录工具扇区地址	烧录器镜像	烧录器块起始地址	结束地址	固件大小	备注
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

表格注释：

1. gpt.img 固定烧录在 block 0，backup gpt 由软件在 u-boot 下自动修复生成；
2. idblock_mutli_copies.img 固定烧录在 block1，要求结束地址为 block 7，镜像要求小于分区大小（预留一个块做坏块替换）；
3. 除 gpt.img 和 idblocks.img 由特定的烧录地址要求，其他固件按照 parameter.txt 中的地址烧录，sector 单位为 512B/s，所以烧录器块地址 = sectors * 512B / block_size，简化换算：
    - 128KB block size：sectors / 0x100；
    - 256KB block size：sectors / 0x200。
    - 除了 gpt.img，其余固件均应比分区小 1~2 个 block size，以便冗余块替换分区内可能存在的坏块；
4. 尾部预留 4 flash block size 给坏块表空间，所以用户分区不应达到该区间，可以考虑定义 reverved 分区以避免用户使用或将来误用。

注意事项

1. 由于 RK spinand 主控 FSPI（旧称SFC）没有集成 ECC 模块，所以需要依赖颗粒自身的 ECC 功能，因此烧录器烧录时所用镜像不带 oob 数据，oob 空间由烧录器自行填充全 FF，并在烧录器工具界面中使能 spinand device 端的 ECC 功能；
2. 非空片烧录，烧录器应先擦除所有 flash 好块再烧录镜像；
3. 烧录器要开启烧录校验功能。

6.2 SLC Nand 镜像烧录 —— GPT 分区扩展

6.2.1 SLC Nand 制作烧录镜像

输入文件：SDK 输出的用于 PC 工具烧录的镜像

```
[/IMAGES] tree
.
├── parameter.txt
├── MiniLoaderAll.bin
├── uboot.img
├── boot.img
├── rootfs.img
├── oem.img
└── update.img                    // 用以制作烧录镜像
```

制作镜像

例如： 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.
```

说明：

- RK3326/PX30/RK3568 programmer_image_tool 添加 -l 参数来修改链表链接方式
- RV1126/RK3326/PX30/RK3308 使用 4KB page size 颗粒时 programmer_image_tool 添加 -2 参数
- 制作出来的镜像对齐为 " block size with oob"，详细参考 ”Nand Flash 信息“ 章节说明

输出文件：用于烧录器烧的镜像

```
out
├── boot.img
├── gpt.img
├── idblock.img             // 自行根据需求制作 idblock_mutli_copies.img 的多备份镜像
├── misc.img
├── oem.img
├── recovery.img
├── rootfs.img
├── trust.img
├── uboot.img
└── userdata.img
```

注释：

- IDB 多备份命令参考，通常备份即可：

```
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 烧录器烧录

烧录地址

假定 block size 为 128KB 的 flash，PC 烧录工具及相应烧录器镜像烧录信息对比如下：

烧录器镜像源文件：SDK 默认输出镜像	PC 烧录工具扇区地址	烧录器镜像	烧录器块起始地址	结束地址	固件大小	备注
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

表格注释：

1. gpt.img 固定烧录在 block 0；
2. idblock_mutli_copies.img 固定烧录在 block1，要求结束地址为 block 7，镜像要求小于 7 个 blocks（预留一个块做坏块替换）；
3. 除 gpt.img 和 idblocks.img 由特定的烧录地址要求，其他固件按照 parameter.txt 中的地址烧录，sector 单位为 512B/s，所以烧录器块地址 = sectors * 512B / block_size，简化换算：
    - 128KB block size：sectors / 0x100；

- 256KB block size：sectors / 0x200。
- 除了 gpt.img，其余固件均应比分区小 1~2 个 block size，以便冗余块替换分区内可能存在的坏块；

4. 尾部预留 4 flash block size 给坏块表空间，所以用户分区不应达到该区间，可以考虑定义 reverved 分区以避免用户使用或将来误用。

注意事项

1. 所有镜像带 oob 数据；
2. 非空片烧录，烧录器应先擦除所有 flash 好块再烧录镜像；
3. 烧录器要开启烧录校验功能。

6.3 SPI Nor 镜像烧录 —— GPT 分区扩展

6.3.1 SPI Nor 制作烧录镜像

输入文件：SDK 输出的用于 PC 工具烧录的镜像

```
[/IMAGES] tree
.
├── parameter.txt
├── MiniLoaderAll.bin
├── uboot.img
├── boot.img
├── rootfs.img
├── oem.img
└── update.img                    // 用以制作烧录镜像
```

制作镜像

例如：

```
./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.
```

输出文件：用于烧录器烧的镜像

```
tree
.
└── out_image.img
```

6.3.2 SPI Nor 烧录器烧录

programmer_image_tool 输出的镜像烧录到 SPI Nor 0 地址。

6.4 SPI Nand 镜像烧录 —— ENV 分区扩展

6.4.1 SPI Nand 制作烧录镜像

输入文件：SDK 输出镜像

```
[/IMAGES] tree
.
├── boot.img
├── download.bin
├── env.img
├── idblock.img
├── oem_2KB_128KB_48MB.ubi
├── oem_2KB_256KB_48MB.ubi
├── oem_4KB_256KB_48MB.ubi
├── oem.img
├── rootfs_2KB_128KB_32MB.ubi
├── rootfs_2KB_256KB_32MB.ubi
├── rootfs_4KB_256KB_32MB.ubi
├── rootfs.img
├── sd_update.txt
├── tftp_update.txt
├── uboot.img
├── update.img
├── userdata_2KB_128KB_32MB.ubi
├── userdata_2KB_256KB_32MB.ubi
├── userdata_4KB_256KB_32MB.ubi
└── userdata.img
```

说明：

- SDK 输出镜像既是 PC 上位机升级镜像，同样也是烧录器烧录所用镜像
- 默认 idblock.img 仅支持 2KB page size spinand，4KB page spinand idblock.img 需重新制作，输出文件为 idblock.img，命令如下：

```
./rkbin/tools/programmer_image_tool  -i download.bin -b 256 -p 4 -2 -t spinand
-o ./
```

6.4.2 SPI Nand 烧录器烧录

烧录地址

假定 block size 为 128KB 的 flash，PC 烧录工具及相应烧录器镜像烧录信息对比如下：

烧录器镜像源文件：SDK 默认输出镜像	PC 烧录工具扇区地址	烧录器镜像	烧录器块起始地址	结束地址	固件大小	备注
env.img	0	env.img	0x0	0x1	0x1	Note 1
idblock.img	0x40000	idblock_mutli_copies.img	0x1	0x3	0x3	Note 2
uboot.img	0x80000	uboot.img	0x4	0x7	0x4	**Note 3**
boot.img	0xC0000	boot.img	0x8	0x47	0x40	
...	...	...	...			
xxx.img	0x58C0000	xxx.img	0x2c6	0x3fc	0x136	Note 4

表格注释：

1. env.img 固定烧录在 block 0；
2. idblock_mutli_copies.img 固定烧录在 block1，要求结束地址为 block 3，镜像要求小于分区大小（预留一个块做坏块替换）；
3. 除 gpt.img 和 idblocks.img 由特定的烧录地址要求，其他固件按照 env.img 中的地址烧录，单位为 Bytes，所以烧录器块地址 = address / block_size，简化换算：
   - 128KB block size：address / 0x20000；
   - 256KB block size：address / 0x40000。
   - 除了 env.img，其余固件均应比分区小 1~2 个 block size，以便冗余块替换分区内可能存在的坏块；
4. 尾部预留 4 flash block size 给坏块表空间，所以用户分区不应达到该区间，可以考虑定义 reverved 分区以避免用户使用或将来误用。

注意事项

1. 由于 RK spinand 主控 FSPI（旧称SFC）没有集成 ECC 模块，所以需要依赖颗粒自身的 ECC 功能，因此烧录器烧录时所用镜像不带 oob 数据，oob 空间由烧录器自行填充全 FF，并在烧录器工具界面中使能 spinand device 端的 ECC 功能；
2. 非空片烧录，烧录器应先擦除所有 flash 好块再烧录镜像；
3. 烧录器要开启烧录校验功能。

6.5 SPI Nor 镜像烧录 —— ENV 分区扩展

6.5.1 SPI Nor 制作烧录镜像

输入文件：SDK 输出的用于 PC 工具烧录的镜像

```
[/IMAGES] tree
.
├── env.img
├── idblock.img
├── uboot.img
├── boot.img
├── rootfs.img
├── oem.img
└── update.img                    // 用以制作烧录镜像
```

制作镜像

例如：

```
./tools/programmer_image_tool -i update.img  -t SPINOR -o ./out
input firmware is 'update.img'
flash type is 'SPINOR'
output directory is './out'
start to write partitions...env
preparing env at 0x00000000
writing env...OK
preparing idblock at 0x00000200
writing idblock...OK
preparing uboot at 0x00000400
writing uboot...OK
preparing boot at 0x00000600
writing boot...OK
preparing rootfs at 0x00004600
writing rootfs...OK
preparing oem at 0x00014600
writing oem...OK
preparing userdata at 0x0002c600
writing userdata...OK
creating programming image ok.
```

输出文件：用于烧录器烧的镜像

```
tree
.
└── out_image.img
```

6.5.2 SPI Nor 烧录器烧录

programmer_image_tool 输出的镜像烧录到 SPI Nor 0 地址。

7. 测试项目

7.1 Nand Flash 产品测试项目

适用于 SLC PP Nand 及 SPI Nand 产品。

7.1.1 flash_stress_test 读写压测

测试目标 测试 SPI Nand 在读写压力测试下的稳定性。相关测试通常已经集成在 SDK buildroot 中，只需添加 rockchip_test 项即可。

测试设备 需要准备 10 台目标设备，其中 userdata 分区预留约 30MB 的空间。

测试方法

1. 在 SDK buildroot 中选择 `./rockchip_test/rockchip_test.sh` 的 "3 flash_stress_test" 选项，并确保设备在测试过程中不会掉电。
2. 记录测试过程中的日志。
3. 使用上位机按下 `ctrl + c` 来停止测试。
4. 根据测试结果进行判断：如果设备在经过 24 小时后仍正常工作，则测试通过；否则测试失败。建议使用关键字搜索 "ubi" 和 "error"，以查看是否有明显的报错信息。

7.1.2 power_lost_test 异常掉电

测试目标 Nand 产品在运行过程中往往会遇到异常掉电的情况，特别是对于不带电池的产品来说，异常掉电的次数较多。因此建议进行针对性的健壮性测试。

相关测试通常已经集成在 SDK buildroot 中，只需添加 rockchip_test 项即可。

测试设备 需要准备 10 台目标设备，其中 userdata 分区预留约 30MB 的空间。

测试方法

1. 在 SDK buildroot 中选择 `./rockchip_test/rockchip_test.sh` 的 "18 nand power lost test" 选项，设备重新上电后将进入持续测试状态，并记录串口日志。
2. 设置掉电器以在供电 14 秒后断电 3 秒，即每组 17 秒，进行 24 小时的测试，即异常掉电约 5000 次，并确保每次掉电后设备能在 17 秒内完成上电并运行测试脚本。
3. 记录测试过程中的日志。
4. 使用上位机输入以下命令来停止测试：`echo off > /data/cfg/rockchip_test/reboot_cnt`
5. 根据测试结果进行判断，如果设备在经过 24 小时后仍正常工作，则测试通过；否则测试失败。建议使用关键字搜索 "ubi" 和 "error"，以查看是否有明显的报错信息。

8. 常见问题

8.1 Nand Flash 信息

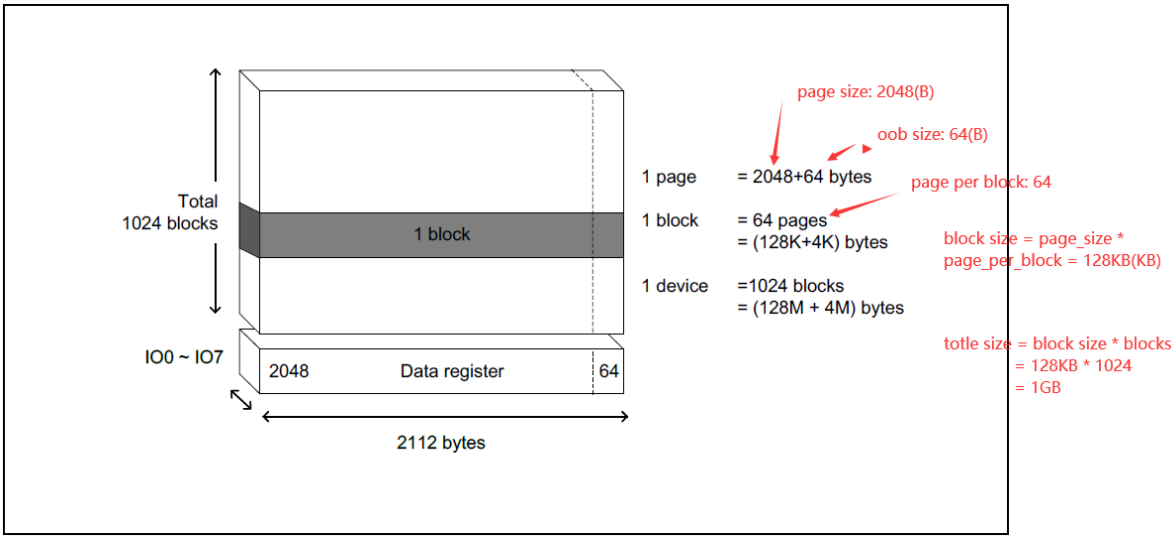
在制作 UBIFS 镜像、IDB 镜像（Pre loader）时需要通过存储颗粒手册来确认 Nand flash 相关信息，并根据选择镜像制作参数，主要包括以下：

- page size，SLC Nand 通常为 2KB 或 4KB page size；
- oob size，SLC Nand 通常为 64B，128B 或 256B。
- page per block，SLC Nand 通常为 64 或 128；
- block size = page_size * page_per_block，SLC Nand 通常为 128KB，256KB;
- block size with oob = (page_size + oob_size) * page_per_block;

在以下操作时，默认通常为 2KB page size，128KB block size 配置，如用其他颗粒，应做相应调整：

- 制作烧录器镜像
- 制作使用 UBIFS 文件系统的镜像；

可以通过通过查看颗粒手册来确认 flash 信息。



8.2 IDB Layout

IDB 为 ddr.bin 和 spl.bin 的打包固件，掩膜代码后的第一级固件，其布局如下：

- PC 工具升级存放在 flash block 1~7，多备份填充；
- 烧录器烧录建议：参考 ”烧录器烧录章节“ 中的 ”烧录地址“ 说明。

8.3 SPI Flash 内核速率测算

mtd_debug 测试命令

选择空闲的 mtd 分区测试，测试前要求该分区未挂载文件系统避免对测试造成干扰，使用 mtd_debug 测试命令进行测速：

```
#测试准备
dd if=/dev/random of=/tmp/test4M bs=1M count=4
mtd_debug read /dev/mtd6 0 0x400000 /tmp/test_backup
sync

#写速率测试
```

```
mtd_debug erase /dev/mtd6 0 0x400000 && sync
time mtd_debug write /dev/mtd6 0 0x400000 /tmp/test4M && time sync

#读测试
time mtd_debug read /dev/mtd6 0 0x400000 /tmp/test4M && time sync

#擦除测试
time mtd_debug erase /dev/mtd6 0 0x400000 && time sync

#测试结束
time mtd_debug write /dev/mtd6 0 0x400000 /tmp/test_backup && time sync
```

SPI Nand 颗粒标称速率计算

以 W25N01KVxxIR SPI Nand 颗粒为例，通过阅读手册 “AC Electrical Characteristics” 章节，得到以下信息：

- block size 128KB，page size 2KB
- 擦除时间：tBE typ. 2ms
- 编程时间：tPP typ. 250us
- 读取时间（ECC enable）：tRD typ. 45us

以 typical 值进行理论计算：

- 编程速率需要计入 io 速率，以 io 100MHz single line 为例，读取 tIO 160us，编程速率 page_size / (tPP + tIO)，约为 5MB/s
  - 内核默认仅支持 single line 编程
- 读取速率需要计入 io 速率，以 io 100MHz quad line 为例，读取 tIO 40us，读取速率 page_size / (tRD + tIO) ，约为 23MB/s
- 擦除速率无需计入 io 延时，所以擦除速率 block_size / tBE，约为 64MB/s

SPI Nor 颗粒标称速率计算

以 XT25F128B SPI Nor 颗粒为例，通过阅读手册 “AC Electrical Characteristics” 章节，得到以下信息：

- block size 64KB，page size 256B
- 擦除时间：tBE typ. 200ms
- 编程时间：tPP typ. 300us
- 读取时间（ECC enable）：接近 IO 速率，tIO

以 typical 值进行理论计算：

- 编程速率需要计入 io 速率，以 io 100MHz single line 为例，读取 tIO 160us，编程速率 page_size / (tPP + tIO)，约为 557KB/s
  - 内核默认仅支持 single line 编程
- 读取速率仅为 io 速率，以 io 100MHz quad line 为例，读取速率约为 50MB/s
- 擦除速率无需计入 io 延时，所以擦除速率 block_size / tBE，约为 320KB/s

8.4 Nand 产品寿命

Nand 标称的擦写寿命，简称 P/E cycles。通常颗粒手册标称的 P/E cycles 为 100K 次。

颗粒产品寿命预估应添加冗余

原厂标称的颗粒 P/E cycles 通常有约束使用条件，常见的是：

- 同一个数据块应避免频繁擦写，否则影响寿命
- 部分颗粒对数据均衡也有一定要求，否则影响电荷平衡

所以实际可以做一些冗余，计入颗粒实际寿命、不同原厂颗粒差异等因素带来的差异，例如以 50% 即 50K 次读写擦除去预估。

颗粒最大数据写入量理论计算

假定分区假如 100M，颗粒计入冗余寿命后支持 50K * 100M 的读写量，但是这部分只是裸数据读写寿命，实际还受到文件系统的策略影响。对于 UBIFS 来看，文件系统冗余通常在 20% 左右，相当于 128KB flash block，大概有 25KB 左右要用来存放 UBIFS 算法信息，所以 50K x 100M x 80% 存放 UBIFS 有效数据，可读写数据量为 4T 数据量。

如果考虑到 UBIFS 上支持的压缩文件系统，4T 还能放大一定量，比如压缩率 50%，那就相当于 8T 用户数据，以上计算仅考虑主要影响因素推算，与实际可能有所出入，请自行进一步推敲。实际计算结果还会因为使用的文件系统、文件系统使用的压缩方案、颗粒的实际寿命而产生相应变化。

颗粒产品寿命实际测试 - UBIFS 文件系统

设备启动过程打印分区 P/E cycles 信息：

```
ubi0: max/mean erase counter: 164/10, WL threshold: 4096, image sequence
number: 207598880
```

说明：

- 其中 max 值即为分区内数据块最大擦除量
- 烤机一定时长后，确认 max 值变化，从而估算产品寿命，例如 24h max 值增加 100 次，则对于 50K 计入冗余的 P/E 值而言，寿命为 500 天

颗粒产品寿命实际测试 - 无文件系统

测试补丁，以 userdata 分区 100MB 编程数据水线为例：

```
diff --git a/drivers/mtd/mtdcore.c b/drivers/mtd/mtdcore.c
index a0b1a7814e2e..7074dc1a267c 100644
--- a/drivers/mtd/mtdcore.c
+++ b/drivers/mtd/mtdcore.c
@@ -1069,6 +1069,7 @@ int mtd_read(struct mtd_info *mtd, loff_t from, size_t
len, size_t *retlen,
 }
 EXPORT_SYMBOL_GPL(mtd_read);

+static size_t total_size;
 int mtd_write(struct mtd_info *mtd, loff_t to, size_t len, size_t *retlen,
           const u_char *buf)
 {
@@ -1082,6 +1083,13 @@ int mtd_write(struct mtd_info *mtd, loff_t to, size_t
len, size_t *retlen,
         return 0;
     ledtrig_mtd_activity();

+    if (!strncmp(mtd->name, "userdata", 8)) {
+        total_size += len;
+        if (total_size == 0x6400000) {
+            pr_err("%s %s 100MB data\n", __func__, mtd->name);
+        }
+    }
```

```
+
    if (!mtd->_write) {
        struct mtd_oob_ops ops = {
            .len = len,
```

操作方式：

- 添加补丁，可以调整 total_size 结束时长，默认 100MB，默认记录 userdata 分区，请根据实际调整
- 更新补丁支持的内核
- 上电后，开始烤机，直到出现 “100MB data”打印，烤机时长

计算寿命：

- 自行计算每小时写入数据量，单位 MB/小时
- Nand 按照 spec P/E cycle，比如 100K（通常），分区 50MB，标称就是 5T 数据量，所以产品寿命预计为“总数据量 / 每小时写入数据量”

其他说明：

- flash P/E 寿命为 flash 寿命的主要指标，其他指标相对影响较低

8.5 Nand 异常掉电问题

掉电关键关联指标

设备异常掉电主要指未经过系统标准反初始化过程的下电行为，可能会导致 Nand 颗粒工作在不稳定电平下。

受影响的主要为 Nand flash 擦除和编程动作，通常擦除时长达到数个毫秒，编程时长达到数百微妙，3V3 颗粒通常工作电平在 2.7V 及以上，实际以手册为准。

典型异常

Flash 异常掉电主要会造成以下两种情况：

- 正在写入的数据不完整
- 掉电过程，flash 正在处理命令，如果工作在不稳定电压，可能会造成随机异常

保护机制

正在写入的数据不完整：

- 文件系统有掉电修复机制保证文件系统不受不完整数据影响，可能会丢数据

不稳定低电平导致的随机异常：

- PMIC 方案：PMIC 检测 VCC IO，当电平低于一定值时，通常为 2.91V 时，触发 CPU 复位信号，Host 不再发起 Nand 擦写命令，flash 外围电路进入放电过程，维持 1ms 左右的工作电平保证 Nand 最后一笔数据操作正常
- Reset IC 方案：原理同 PMIC 方案，通常选择检测 VCCIO 低于阈值电平 2.93V 时触发 CPU 复位信号
- RK NPOR 方案：原理同 PMIC 方案，NPOR 逻辑检测 VCCIO 低于阈值电平时触发 CPU 复位信号

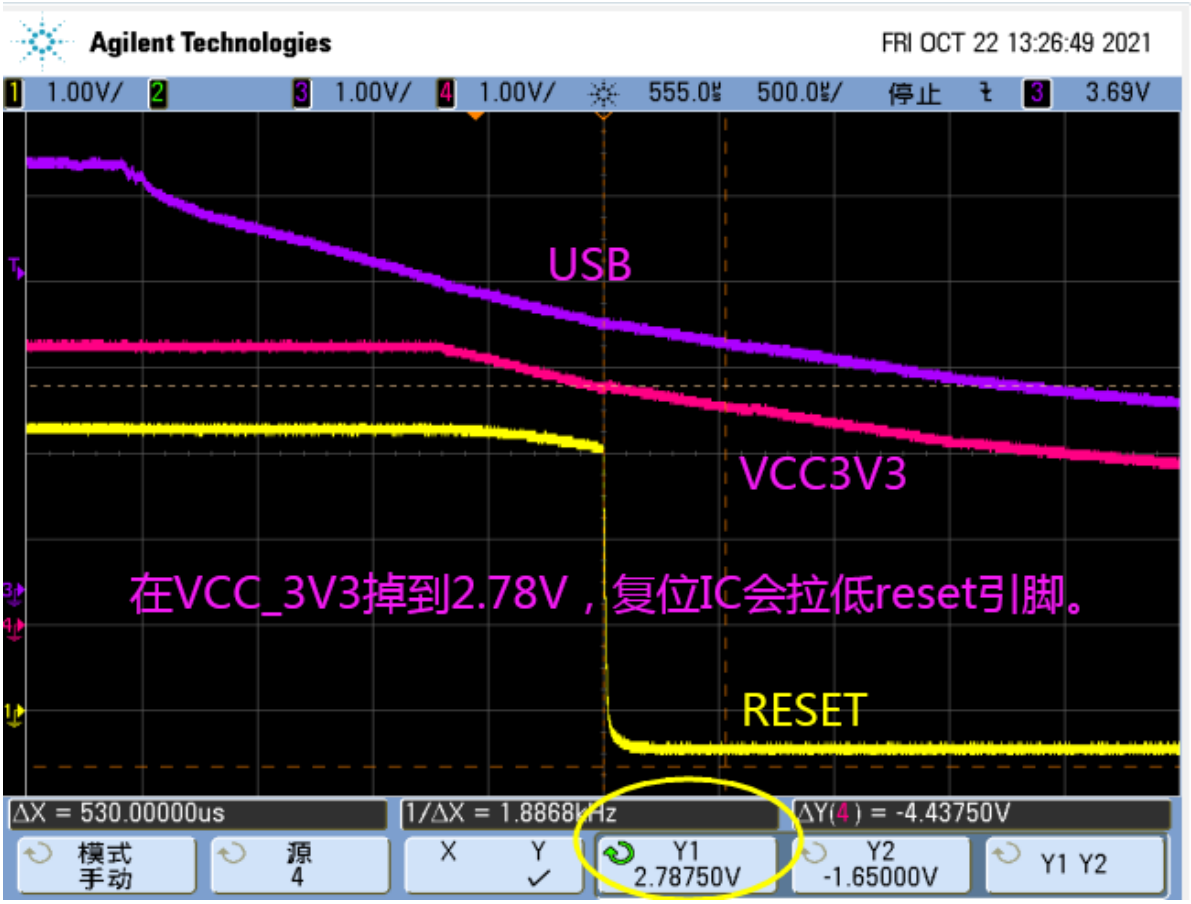
如何避免异常掉电问题

- 建议 Nand 产品设计为带电池或常供电方案，以避免异常掉电行为
- 如无法避免异常掉电行为，建议优化硬件设计，保护 Nand 下电过程的工作电压符合：

- PMIC/Reset IC/RK NPOR 方案检测 VCC IO 电平，监测掉电行为，触发 CPU 复位，避免低电环境下发起 Nand 擦写命令
- 参考 RK demo 板添加一定的电容，让 Nand 供电 VCCIO flash 维持正常工作电平，时长通常为 tBERS，即块擦除时长
- 减少 flash 读写过程中的掉电行为
- 关键数据做备份和恢复的机制：
  - 固件双备份
  - 关键数据存放在只读文件系统中
  - 可读写分区如果出现掉电异常，建议有恢复机制，保证设备能够正常运行，或者实现 OTA

如何确定异常掉电时序是否合理

在实际产品、实际业务环境下，搭建 “power_lost_test” 测试环境，然后抓取掉电时的 VCC3V3/RESETn 信号的时序：



VCC3V3/RESETn 信号时序测量要求

- 示波器界面提供以下信息：
  - RESETn 触发点或 RESET IC 触发点（Y1 轴）
  - VCC3V3 掉电到 2V7（Y2 轴）
  - 确认 ΔY1Y2 轴时间
- 多次测量，挑选 ΔY1Y2 值最小场景分析
- RV1106 等 RKNPOR 方案测量掉电时序时使用 GPIO 替代对齐 RESETn 信号，RKNPOR RESETn 信号为芯片内部信号，无法直接测量，所以建议可以考虑将某个 GPIO 置高，然后外部加下拉，CPU 复位触发的时候，对应 GPIO 失效并被外部下拉拉低。电平翻转的这个点，可以近似认为是 CPU 复位的触发点

8.6 Flash 内核信号测试脚本

开源 MTD 框架有成熟 MTD_TESTS 框架，源码路径 drivers/mtd/tests。

spinand 设备补丁

为避免测试写信号时过多的读状态位行为引入，增加延时跳过轮询时间。

```
diff --git a/drivers/mtd/nand/spi/core.c b/drivers/mtd/nand/spi/core.c
index f0ba92a09ae5..3f1d09200f74 100644
--- a/drivers/mtd/nand/spi/core.c
+++ b/drivers/mtd/nand/spi/core.c
@@ -573,6 +573,7 @@ static int spinand_write_page(struct spinand_device
*spinand,
        if (ret)
                return ret;

+       mdelay(1);
        ret = spinand_wait(spinand, &status);
        if (!ret && (status & STATUS_PROG_FAILED))
                ret = -EIO;
```

宏开关

CONFIG_MTD_TESTS=m

输出 ko 文件

mtd_readtest.ko、mtd_torturetest.ko

推送读写压测用 ko 文件

```
adb push mtd_readtest.ko mtd_torturetest.ko /data
```

读写命令

```
insmod /data/mtd_readtest.ko dev=0 cycles_count=10                    # 读测试，可
以通过修改 cycles_count 来调整测试时长
rmmod mtd_readtest                                                    # 读测试清
除，如需重新测试，重新 insmod


insmod /data/mtd_torturetest.ko dev=0 check=0 cycles_count=10 random_pattern=1
     # 写测试，可以通过修改 cycles_count 来调整测试时长
rmmod mtd_torturetest                                                 # 写测试清
除，如需重新测试，重新 insmod
```

说明：

- 读测试建议选择容量大、有有效镜像的分区测试，这样数据线易于呈现高低电平变化
- 写测试建议选择空闲分区，避免写数据对系统运行产生影响

8.7 Flash 物料更换支持列表上的颗粒后升级或启动异常

通常新物料需同步更新存储驱动，向 RK 业务申请 Redmine 权限并同步 存储补丁 。

8.8 Flash 概率性启动异常或读文件系统校验出错

从以下方向做一些信号调整，如拷机正常，建议测量信号并针对 flash 信号质量做进一步优化调整：

- 降低频率测试，修改方式参考 “Kernel 基础配置” 章节
- 改为单线传输：

mtd 方案单线传输补丁：

dts spiflash 设备节点修改 spi-rx-bus-width 为 1，即单线写单线读。

rkflash 方案 SPI Nand 单线传输补丁：

```
diff --git a/drivers/rkflash/sfc_nand.c b/drivers/rkflash/sfc_nand.c
index 4accf3d791c6..98b1cfecc821 100644
--- a/drivers/rkflash/sfc_nand.c
+++ b/drivers/rkflash/sfc_nand.c
@@ -1179,6 +1179,8 @@ u32 sfc_nand_init(void)
                return -ENOMEM;
        }

+       p_nand_info->feature &= ~(FEA_4BIT_READ | FEA_4BIT_PROG); // 单线写、单线读
+       // p_nand_info->feature &= (~FEA_4BIT_PROG); // 单线写、四线读
        if (p_nand_info->feature & FEA_4BIT_READ) {
                if ((p_nand_info->has_qe_bits && sfc_nand_enable_QE() == SFC_OK) ||
                    !p_nand_info->has_qe_bits) {
```

rkflash 方案 SPI Nor 单线传输补丁：

```
diff --git a/drivers/rkflash/sfc_nor.c b/drivers/rkflash/sfc_nor.c
index c38801b4fb0d..4badff4249ac 100644
--- a/drivers/rkflash/sfc_nor.c
+++ b/drivers/rkflash/sfc_nor.c
@@ -678,7 +678,9 @@ static int snor_parse_flash_table(struct SFNOR_DEV *p_dev,
                        p_dev->write_status = snor_write_status1;
                else if (i == 2)
                        p_dev->write_status = snor_write_status2;

+               g_spi_flash_info->feature &= ~(FEA_4BIT_READ | FEA_4BIT_PROG); // 单线写、单线读
+               // g_spi_flash_info->feature &= ~(FEA_4BIT_PROG); // 单线写、四线读
                if (g_spi_flash_info->feature & FEA_4BIT_READ) {
                        ret = SFC_OK;
                        if (g_spi_flash_info->QE_bits)
```

8.9 UBIFS 挂载时带 ro 属性并非真实只读文件系统

UBIFS 挂载为 ro，文件系统后台依旧有存储算法涉及 Nand 擦写行为，建议选择 SquashFS 并挂载在 UBI Block 设备下实现真正的只读文件系统。

8.10 其他问题排查

如有其他问题，建议阅读 《Rockchip_Developer_FAQ_Storage_CN.pdf》 文档，Redmine 路径：

Rokchip_Developer_FAQ_Storage_CN.pdf

如对 flash 有一些基础性的学习需求，建议阅读《Rockchip_Application_Notes_Storage_CN.pdf》 文档 “Flash 简介” “颗粒验证” 章节，Redmine 路径：

Rockchip_Application_Notes_Storage_CN

如有双存储驱动开发需求，建议阅读 《Rockchip_Developer_Guide_Dual_Storage_CN.pdf》 文档，Redmine 路径：

Rockchip_Developer_Guide_Dual_Storage_CN.pdf

9. 附录参考

[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