Rockchip Application Notes Storage

文件标识：RK-SM-YF-017

发布版本：V2.1.0

日期：2024-01-15

文件密级：□绝密 □秘密 □内部资料 ■公开

免责声明

本文档按"现状"提供，瑞芯微电子股份有限公司（"本公司"，下同）不对本文档的任何陈述、信息和内容的准确性、可靠性、完整性、适销性、特定目的性和非侵权性提供任何明示或暗示的声明或保证。本文档仅作为使用指导的参考。

由于产品版本升级或其他原因，本文档将可能在未经任何通知的情况下，不定期进行更新或修改。

商标声明

"Rockchip"、"瑞芯微"、"瑞芯"均为本公司的注册商标，归本公司所有。

本文档可能提及的其他所有注册商标或商标，由其各自拥有者所有。

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

本文主要指导读者了解启动流程，对存储进行配置和调试。

本文档不尽详实内容还可以参考下列文档：

序号	文档名称	内容概要
1	《Rockchip_Introduction_Partition》	介绍分区配置
2	《Rockchip-Developer-Guide-UBoot-nextdev-CN》	uboot 开发文档
3	《RK Vendor Storage Application Note》	Vendor Stroage 应用文档
4	《Rockchip量产烧录指南_v1.2》	量产烧录指南
5	《Rockchip_Developer_Guide_Linux_Flash_Open_Source_Solution_CN》	Flash 开源存储方案开发文档
6	《Rockchip_Developer_Guide_Dual_Storage_CN》	双存储开发文档

各芯片 feature 支持状态

芯片名称	内核版本
所有产品	--

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	赵仪峰、林鼎强	2018-08-25	初始版本
V1.1.0	赵仪峰	2021-03-02	增加RK3566和RK3568
V1.2.0	林鼎强	2021-10-29	增加 IDB 说明，移除开源存储驱动支持说明
V2.0.0	林鼎强	2024-01-08	增加命名规则及 Flash 简介
V2.1.0	林鼎强	2024-01-14	增加关于 flash 简介的内容

目录

1. 命名规则

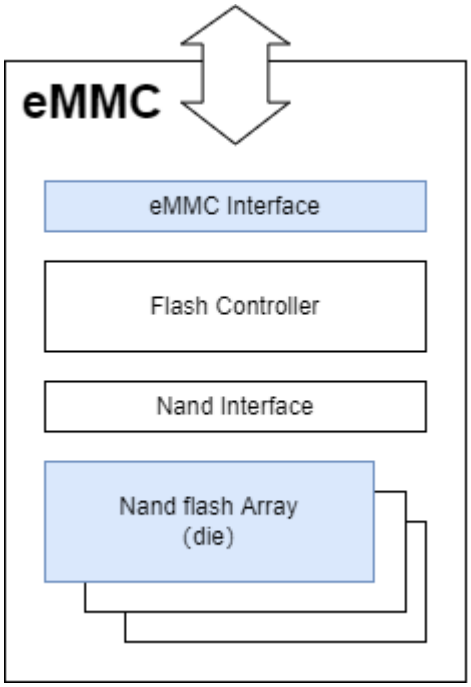
命名	简介
SPI Nand	SPI 协议 Nand，多为 SLC Nand
SPI Nor	SPI 协议 Nor
PP Nand	Parallel peripherals Nand，并口 Nand，SLC\MLC\TLC Nand
Flash	SPI Nand、SPI Nor、PP Nand 统称
Nand	SPI Nand 和 PP Nand 等 Nand 颗粒统称
Octal SPI DTR Nor flash	Octal SPI Nor flash 带双采样
OCTA flash	Octal SPI Flash，包括 Octal SPI Nor、Octal SPI Nand

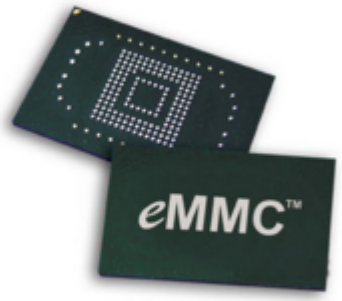
2. Flash 简介

2.1 Flash 存储类型

广义上的 flash 指所有基于 Nand flash 和 Nor flash 技术的非易失存储，例如我们 RK 设备上常用的 EMMC 和 SPI flash ，抑或是这几年逐渐进入视野的 PCIe SSD、SATA SSD 以及 UFS ，大家可能都比较好奇这几个存储之间的差异是什么，下面做一个简单的介绍。

存储颗粒主要涉及两个内容，颗粒原片和接口控制器，也就是我们常说的 die 和 controller。



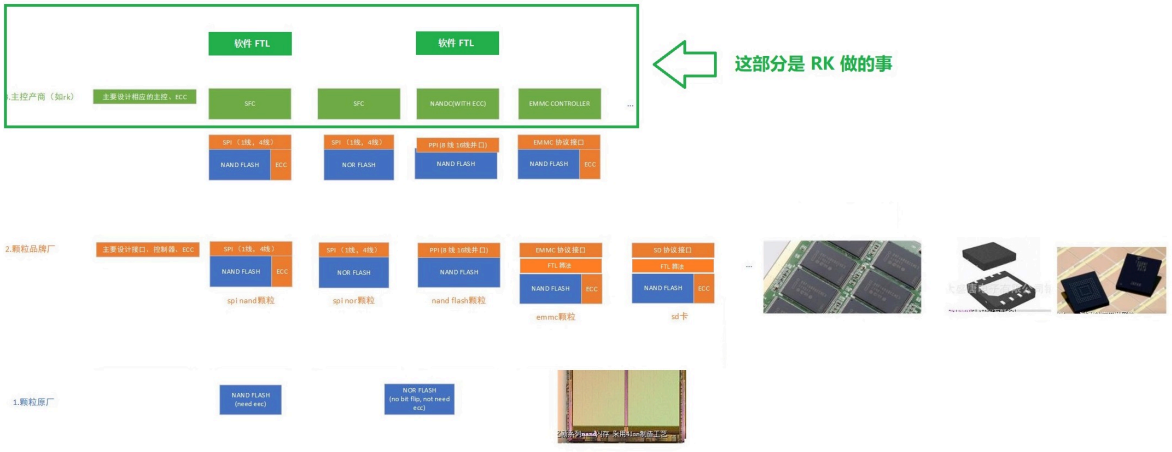


Flash 原片是最终贮存用户数据的载体，如截图里的 Nand flash Array。原厂生产不同容量大小的原片然后搭配不同的控制器组成特定的存储器件，用于各个行业。如 Nand flash die 加上 EMMC 接口封装成的 EMMC 颗粒，Nor flash die 加上 SPI 接口封装成 SPI flash 颗粒，当然这些存储颗粒通常也和我们的芯片一样内部也有运行固件，也就是俗称的 Firmware 固件，以下是部分存储颗粒的相应说明：

器件	器件介绍	Host 控制器	Host 端特殊驱动
SPI Nor	Nor + SPI 接口	FSPI/SPI 控制器	较为简单，支持磨损均衡、异位更新的文件系统即可
PP Nand	Nand + 并口	NandC 控制器	FTL 算法或带 FTL 的文件系统
SPI Nand	Nand + SPI 接口	FSPI/SPI 控制器	FTL 算法或带 FTL 的文件系统
eMMC	Nand + eMMC 接口 + FTL 算法	SDMMC、SDHCI 控制器	MMC 协议框架
SD 卡	Nand + SDIO 接口+ FTL 算法	SDMMC 控制器	MMC 协议框架
SATA SSD	Nand + Sata 接口+ FTL 算法	Sata 控制器	SATA 协议框架
NVMe SSD	Nand + PCIe 接口+ FTL 算法	PCIe 控制器	NVMe 协议框架
UFS	Nand + UFS 接口+ FTL 算法	UFS 控制器	UFS 协议框架
...			

说明：

- flash 器件基本上是伴随着更高速的接口控制器、内部并行复杂度更高、容量更大的方向发展去迭代
- RK 产品除了支持过以上常见存储外，还支持过 SD Nand（贴片封装）
- Nand flash 组成的器件都需要存储 FTL 算法对 Nand 进行管理，原因 "Nand 基础原理" 章节有所介绍
- PP Nand/SPI Nand 由于器件自身未集成 FTL 算法，所以 Host 端软件需要集成 FTL 算法或带 FTL 算法的文件系统
- 颗粒原厂、颗粒品牌商、RK 主控结合说明：



2.2 Flash 选型

SLC Nand、SPI Nand、SPI Nor

以 GigaDevice 选型参考认识 Flash 物料选型的范围，GD 颗粒选型参考链接

2.3 Flash 简单对比

RK 几种主存简单对比：

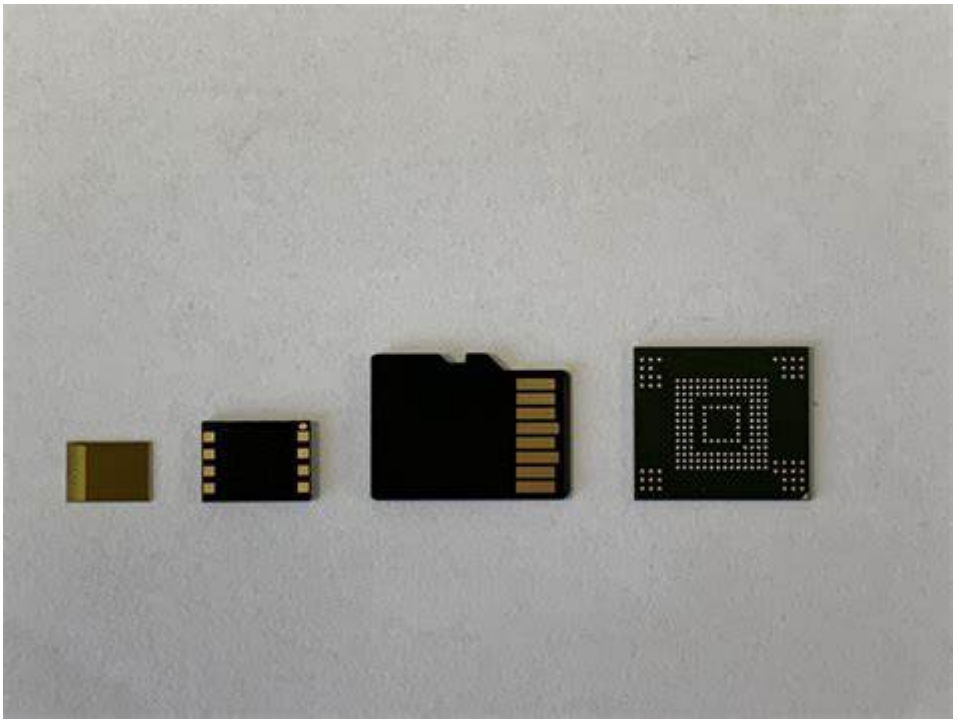
存储类型	稳定性	单体价格	常见尺寸及封装	IO 速率	常见容量
EMMC	Good	High	13x11.5 153 FBGA	200M DDR (HS400)	>= 1GB
SLC Nand	Nornal	Nornal	20x12 TSOP48	30M SDR	128MB~512MB
SPI Nand	Nornal	Low	8x6 WSON	133M SDR 80M DDR	64MB~512MB
SPI Nor	Good	Low	5x4 SOP8/8x6 WSON	166M SDR 104M DDR	<= 128MB
Octal SPI Nand	Nornal	High	8x6 24-BALL TFBGA	120M DDR	128MB~512MB
Octal SPI Nor	Good	High	8x6 24-BALL TFBGA	200M DDR	<= 128MB

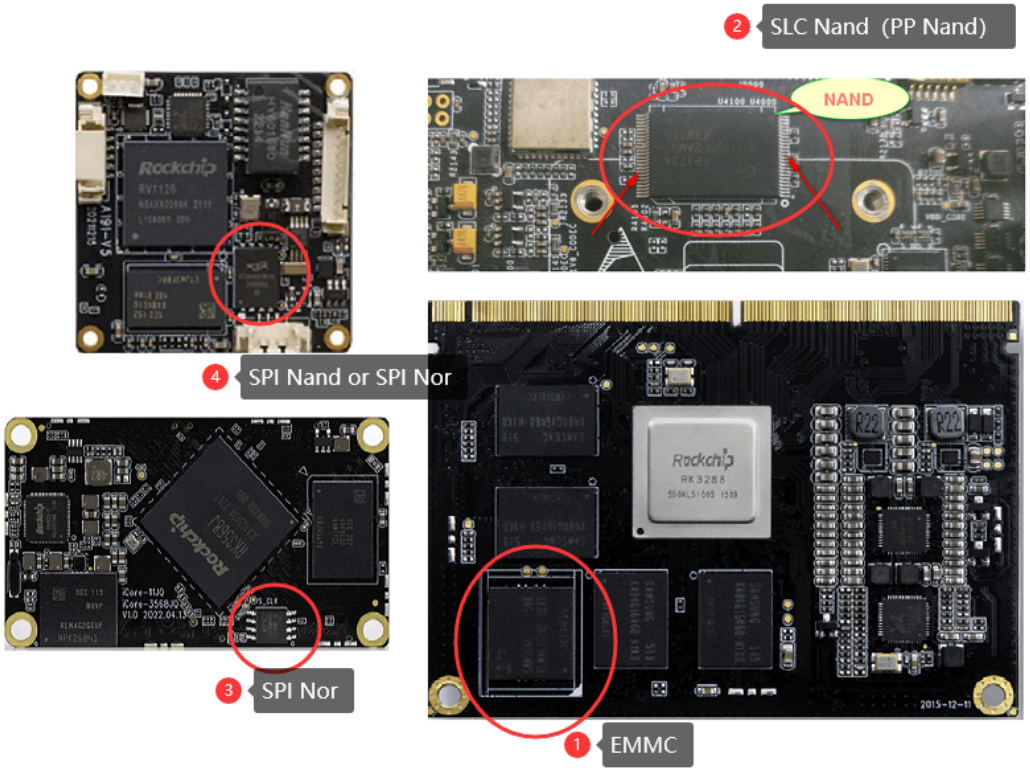
说明：

- EMMC 有容量大、稳定性较高、读写速率快等优点，但单体价格较贵，封装较大，初始化时间开销较大（通常达到百毫秒级别）等缺点
- SPI Nand 有封装小、单片价格低、容量相较于 Nor flash 来的大等优化，但由于 Nand 涉及 ECC、存储算法管理等限制，所以读写速率及稳定性较为一般
- SPI Nor 有封装小、稳定性高、单片价格低、软件初始化时间开销小的优点，尤其是 SPI Nor 在小固件场景下加载速率不俗，但连续数据传输速率不如 EMMC，且当容量上升后单体价格较贵
- 由于 MLC/TLC PP Nand 对 FTL 算法的依赖、NandC 控制器占芯片面积较大、以及引脚多等缺点，RK 近年来已经淘汰 PP Nand 接口，所以小容量存储主要指 SPI Flash，也因此 MLC/TLC PP Nand 并未列入以上表格
- 价格仅基于单片计算价格，每 MB 所折算的价格随容量增大而降低

2.4 Flash 常见封装

几种封装对比：





SLC Nand/SPI flash 常见封装：

封装介绍

2.5 Flash 价格

存储类型	容量	价格（美金）
SPINand	1Gbits QPI Comsummer	0.53
SPINand	2Gbits QPI Comsummer	0.75
SPINand	4Gbits QPI Comsummer	1.7
SPINand（Continuous read mode 物料单一）	相较于同容量颗粒	+30% 左右
PP	1Gbits QPI Comsummer	0.6
PP	2Gbits QPI Comsummer	1.7
PP	4Gbits QPI Comsummer	3.8
Nor	32Mbits QPI Comsummer	0.12
Nor	64Mbits QPI Comsummer	0.17
Nor	128Mbits QPI Comsummer	0.28
Nor	256Mbits QPI Comsummer	0.9
Nor	512Mbits QPI Comsummer	2.1
Nor	256Mbits OPI Auto	2.1
Nor	512Mbits OPI Auto	3.6

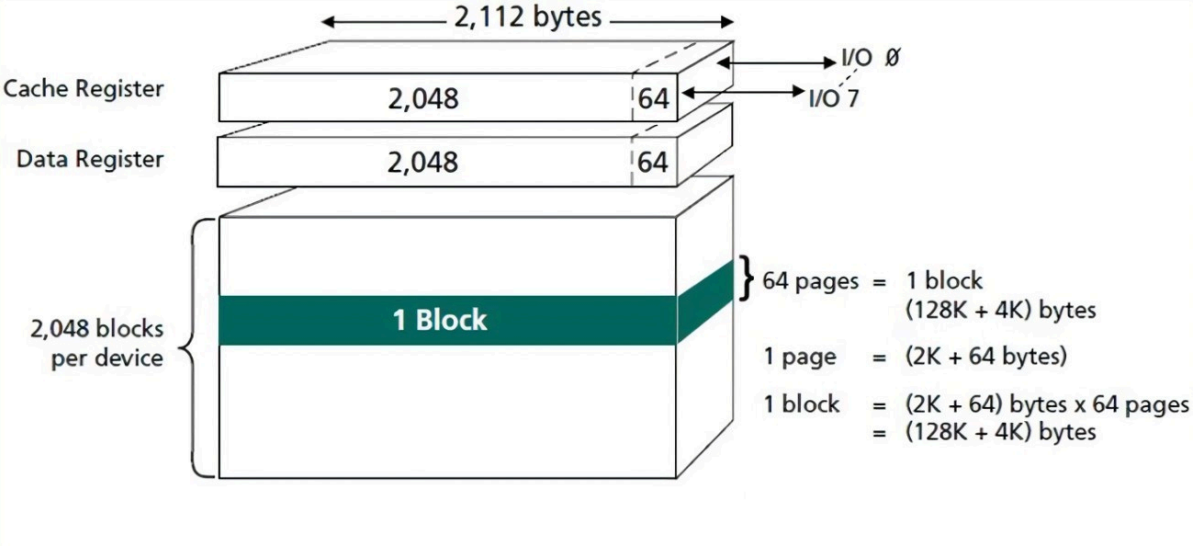
说明：

- 以上价格为特定厂家的市面报价，非客户对接加价
- 以上价格主要用于反映不同存储器件之间的价格差异以及同种存储器件的价格随容量增大的趋势情况

2.6 Nand 基本原理

Nand flash 阵列上的物理结构

以 SLC Nand 常见结构为例：

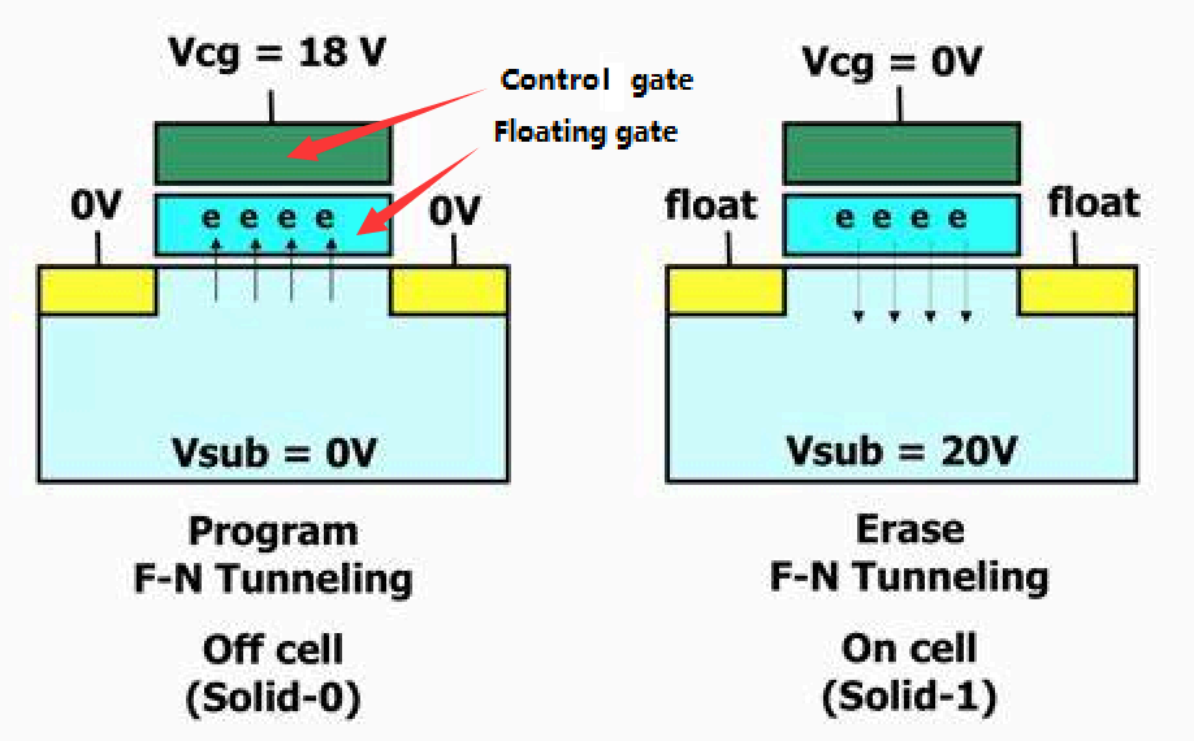


说明：

- SLC Nand 是由多个 flash block 组成，通常达到 1024、2048 或 4096 blocks，以上截图为 2048 blocks
- 块（block ）是 Nand flash 最小擦除单位，SLC Nand block size 通常为 128KB 或 256KB，擦除后由逻辑 0 变为 1
- 一个 block 通常由 64 pages 组成，有一些颗粒多达 128 pages
- 页（page） 是 Nand flash 编程和读的基础单位，SLC Nand page size 通常为 2KB 或 4KB，编程后由逻辑 1 变为 0

逻辑上的 0 或 1 实际上是每个 Bit cell 物理单元上的电平值

以 floating gate 实现的 SLC Nand 为例：

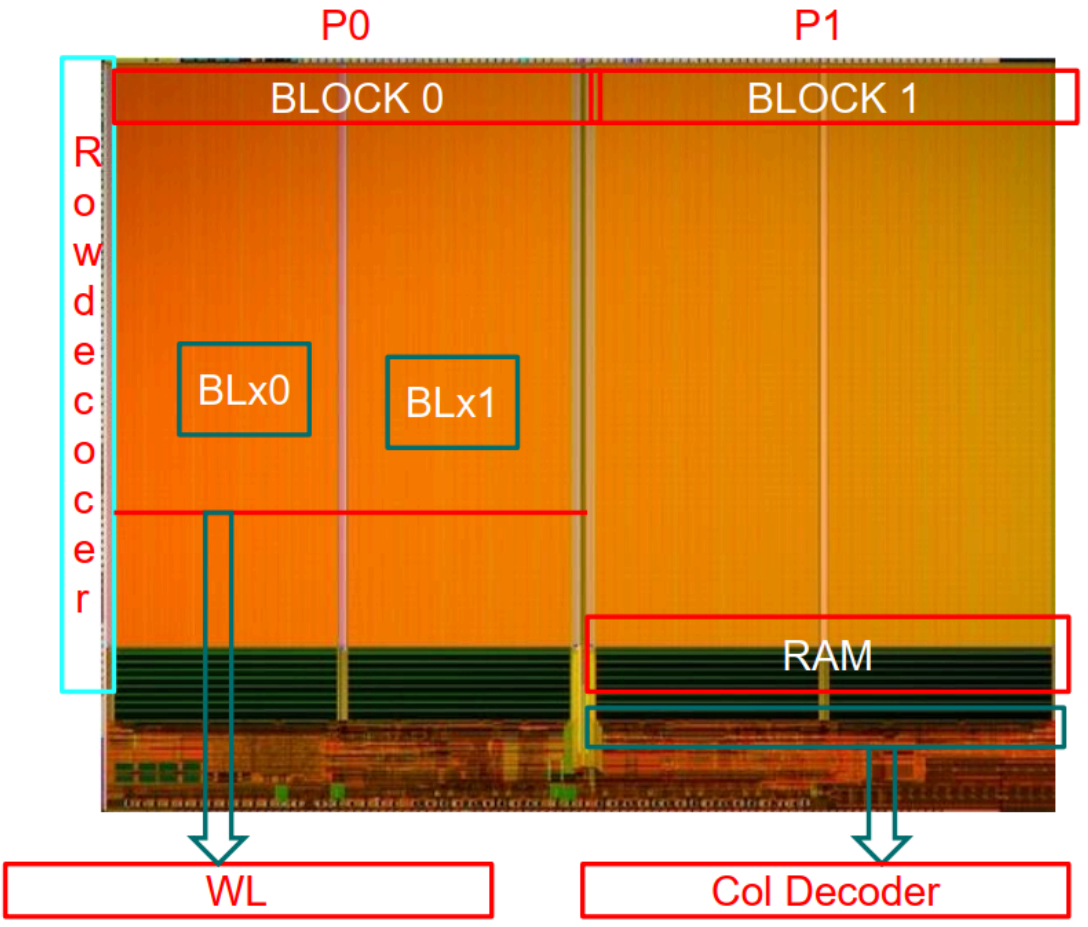


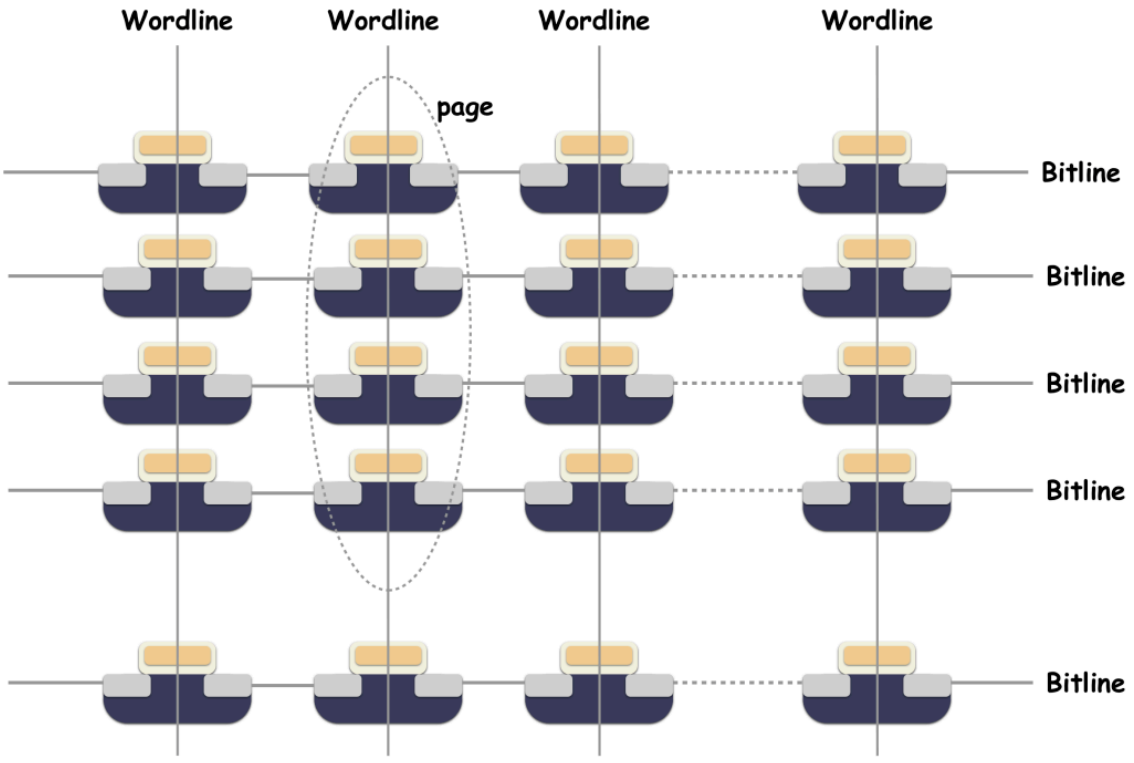
Flash 真正存储数据的单元是内部的浮栅（Floating gate），通过控制门（Control gate） 和衬底施加电压控制浮栅是冲入电荷还是释放电荷，而数据上逻辑 0 或 1 由存储的电荷的电压是否超过一个特定的阈值 Vth 来表示：

- 对于 NAND Flash 的写入，就是通过加高压控制门对浮栅充电，超过阈值 Vth，就表示 0
    - Data Retention 问题：电荷有加高压进入浮栅，在没有擦除行为时，电荷存储在浮栅中，但由于浮栅和衬底存在电场，电荷会随着时间逐渐流失
    - P/E cycle 问题：随着擦写次数增加，也就是 Program/Erase Cycle（P/E Cycle）增加，浮栅和衬底之间的氧化物层老化，同样会影响浮栅储存电荷的能力
- 对于 NAND Flash 的擦除，就是通过加高压衬底对浮栅放电，低于阀值 Vth，就表示 1

Wordline 和 Bitline 的设计

为了高效地完成成千上万 Nand flash cell 的充放电动作，设计者实现了 Bitline/Wordline 等辅助电路结构，例如多个 cell 串行，多组 cell 组成 Nand string，一组 Nand string 的两端是 Bit Line 和 Source line，每个 cell 控制门由一根 Wordline 连接，同一根 Wordline 上控制的 cell 为逻辑上的页（Page），整体组成一个 block，block 里所有的 cell 又是共用衬底，所以最小擦除为块。





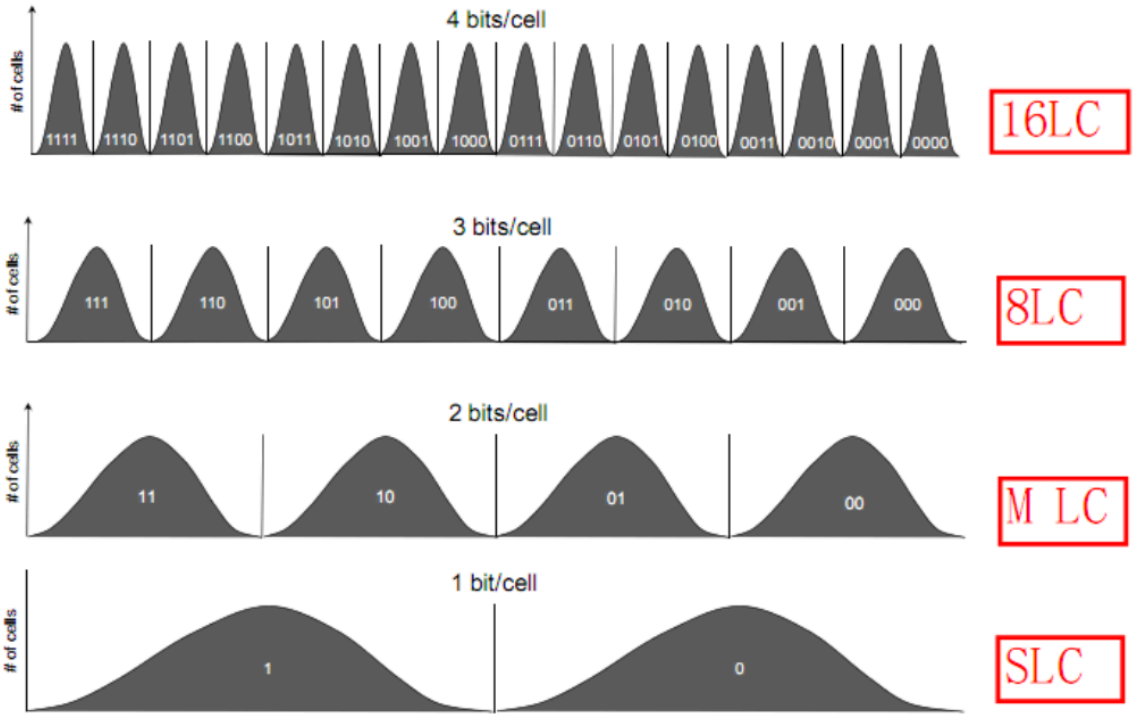
Bitline 图片来源

关于 SLC\MLC\TLC\QLC 的基础原理：

如上可以通过 Bitine 来控制充放电行为，决定 bit cell 的电平状态，实际上还可以通过控制充电量的多少，实现单个 bit cell 的不同电平表现：

- SLC、Single Level cell，单个 bit cell 代笔 1 bits 数据，即仅有 0/1 两种电平逻辑
- MLC，支持 2 bits 数据，有 0~3 的 4 级电平状态

- TLC，支持 3bits 数据，有 0~7 的 8 级电平状态
- QLC，支持 4bits 数据，有 0~15 的 16 级电平状态



—— 图片来源于网络

Bit cell 电荷存储能力失效出现位翻转（bit flip）

由于 Nand 特定的物理结构，导致 Nand 颗粒存在充放电过程电压不稳定、擦写磨损导致物理结构老化影响储电能力的可能，从而导致逻辑电平发生变化，也就是出现位翻转（bit flip），在用户层面上可能就会出现数据出错、固件异常、文件系统乱飞等表现，所以 Nand flash 产品都会引入不同能力的 ECC 纠错算法支持，以延长颗粒寿命、增强 Nand 产品的健壮性。

Nand flash 产品特点

由于 Nand 的物理实现原理决定着 Nand flash 产品有以下特点：

- 异处更新(out-of-place Update)。Nand Flash 编程操作只能把存储单元从1变为0，所以在重新编程之前需要进行擦除操作，而且编程以页为单位，擦除以块为单位(一个块包括多个页)，如果使用同处更新(in-place update)，就是把同逻辑地址重复更新到同样的位置上，那么每一次更新，都需要先进行一次擦除操作。由于擦除操作耗费时间和对Flash有损伤，所以一般 FTL 使用异处更新，把更新的数据映射到一个新的位置上。
- P/E次数有限制。之前有提到，Nand Flash每个块是有擦除次数限制的，在擦除一定次数后，这个块会变得不稳定，编程进去的数据容易出错，甚至会擦除失败。通常旧制程的 SLC Nand 擦写次数在 100K 次，现在原厂为了降成本推出的新制程的颗粒擦写寿命可能有所下降。
- 性能更好。和传统机械硬盘不同，Flash存储是没有机械设备的，比如说不需要寻道，对所有的地址访问开销都一样，特别是在随机读性能上，SSD远远好于传统机械硬盘。按这个道理，Flash设备随机访问和顺序访问的速度是一样，但现实上，Flash支持Cache操作，在顺序访问中可以提前把下一个页的数据读取放到内部寄存器中，可以更快响应读请求。所以在顺序访问上速度要比随机访问要快的。
- 读、写速度不一致。如之前文章描述的，把电子从浮动门中吸进去（写操作）比检测浮动门电场状态（读操作）要耗时。所以FTL在管理时，尽量减少写和擦除的操作。

2.7 Nor 基础原理

原理与 Nand 相近，主要有以下差异：

- bit cell 实现原理相近，但每个 cell 两端直接接入 Bit Line 和 Source line，更稳定，所以所以通常 Nor 不需要 ECC 机制来防止物理上的老化（然而实际上依旧存在极小概率翻转异常），但 cell 周围的电路结构更复杂，相同容量上 die 的成本更高
- Block/page 和 Nand 大小不一
    - 通常支持广义上的擦除块 Block/Sector 两种格式，分别为 64KB 和 4KB
    - 通常支持基础编程（写）单位 page 为 256B

2.8 Nand 存储 ECC 依赖

RK 集成情况：

颗粒类型	主控	选择所依赖ECC
EMMC	无需 ECC	EMMC 颗粒
SPI Nand	不带 ECC	SPI Nand 颗粒
SLC Nand	Nand V6 使用 16bits ECC	NandC 控制器
MLC TLC Nand	Nand V9 使用 16bits 及更高位 ECC	NandC 控制器

说明：

- SLC Nand 部分颗粒自带 ECC，其余 PP Nand 都不集成 ECC，依赖控制器自带的 ECC

2.9 Nand 原厂坏块

Nand Flash 由于其物理特性的原因，工艺上是允许存在一定比例的原厂坏块，出厂时会在 flash 的特定区域置上原厂坏块标记，通常是在 spare first byte (或称为 oob 区域)，通常原厂坏块标记是不可破坏的。

- W25N01GV SPI Nand 除外

2.10 Nand 寿命及 ECC 报错

Nand flash 有耐久度，通常是用 P/E（擦除/写）来表征，P/E 达到一定数量级后会出现颗粒失效的情形，实测个别品质差的颗粒可能在 30K 次左右就会出现异常，这点如果与原厂标称不符，会建议客户联系原厂定位。

接近 Nand flash 寿命时候，通常会有如下 warning：

- 读数据位翻转达到需要 refresh 数据的情形
    - 数据有效，使用正常

达到或超过 Nand flash 时通常会出现如下异常：

- 读数据位翻转达到需要 refresh 数据的情形
  - 数据有效，使用正常
- 读数据报 ECC fail
  - 数据无效，底层会有 retry 机制，所以可能会造成线程占用 cpu 率较高
  - 有部分数据丢失，进而导致上层异常
- 擦除/写 fail
  - 标记为坏块，不影响使用，数据不会丢失

2.11 Nand 存储 FTL 算法关键技术说明

- 地址映射管理。闪存设备对外是一个黑盒子，里面集成了Nand Flash和FTL等，上层应用使用逻辑地址来访问，FTL把逻辑地址映射到不同物理地址上，管理着每个逻辑地址最新的数据存放的物理位置
- 垃圾回收。随着数据的写入，闪存设备上有些块的部分数据已经无效了，需要把有效的数据从块上搬走，然后擦除用来接收新的数据。
- 磨损均衡和坏块管理。因为每个块的P/E次数是有限的，某些块可能被重复使用而损坏了，而有些块数据很少被访问，所以一直没有进行操作过。为了避免这种情况，FTL加入磨损均衡的功能，大致是通过控制垃圾回收和空块池的管理，从而平衡每个块的使用次数，最理想是所有块一起达到磨损阈值。由于Flash本身就存在部分坏块，在使用的过程中部分块会变坏，所以FTL在管理的时候需要避开这些无用块，把使用后变得不稳定块上的数据及时拷贝到稳定位置。

2.12 Nand 存储方案变迁（包括 PP Nand 和 SPI Nand）

- 早期仅支持 RK 闭源 FTL 方案 rkflash
  - 主要包括 RK3326\RK3308\RV1108
- 开始有客户有 UBIFS、MTD 及烧录器烧录等需求，需要用到 MTD 存储驱动框架
  - RK3308 芯片开始，由于该方案从驱动、到算法、到文件系统都为开源代码，所以通常内部区别闭源方案，称为 MTD 开源方案
  - RK3308 提供底层驱动，上层衔接由客户（开发能力较强）自行完成
  - RK3308 产品同事曾提供 PP SLC Nand MTD 开源方案的 SDK 配置
- 全面转到 MTD 开源方案，并提供更详细的支持和指导：
  - RV1126\RK3568 及之后芯片

2.13 Flash Host 端控制器

2.13.1 SFC 控制器

串行闪存控制器（SFC）用于控制芯片系统与串行nor/nand闪存设备之间的数据传输。

The SFC supports the following features:

- 支持 SPI Nor、SPI Nand

- 支持 SPI Nor 1线、2线和4线传输
- DMA 传输

2.13.2 FSPI 控制器

FSPI (Flexible Serial Peripheral Interface) 是一个灵活的串行传输控制器，SFC 的新设计，考虑到支持的器件有所变化，所以更新命名为 FSPI，有以下主要特性：

- 支持 SPI Nor、SPI Nand、SPI 协议的 Psram 和 SRAM
- 支持 SPI Nor 1线、2线和4线传输，version 8 及以后支持 8 线 DDR 传输
- XIP 技术
- DMA 传输

2.13.3 NandC 控制器

NandC 使用来完成 Nand flash 和主芯片之间的数据传输的主控，支持数据直接通过 AHB 总线的 master 传输。为了针对不同应用场景，RK 目前主要有两版的 NandC，富集成的 NandC V9 和 简化后占芯片面积较小的 NandC V6。

一个芯片一般根据市场地位选择 NandC 版本，NandC V6 仅支持 SLC Nand 通常放在小容量存储方案的产品中（通常 SLC Nand 小于 512MB），NandC V9 可以支持MLC、TLC，所以可以应用在大容量存储的产品中。

2.13.4 通用 SPI 接口

FSPI 为专用 SPI Flash 接口，但 RK SOC 通常还有多个通用的 SPI 接口，该接口同样支持外挂 SPI Flash 器件，但通常该接口器件无法作为 bootdev。

2.14 SPI Flash 输出延时统计

汇总：

	1.8 V	**3.3 V**	**1.65-3.6**
FORESEE		< 8 ns	
BIWIN		< 9 ns	
Dosilicon	<10 ns	<8 ns	
ESMT		<8 ns	
Toshiba		<6 ns	
WINBONG		<7 ns	
MXIC		<8 ns	
MXIC (SPI NOR)			<12 ns(30 pf)<10 ns(15 pf)

说明：

- 具体颗粒以手册为准，表格仅供参考

3. 颗粒验证

3.1 SLC Nand/SPI Nand/SPI Nor 验证内容简述

验证说明：

小容量颗粒自身较为稳定，且与主控制器之间的兼容性通常较优，所以大部分仅作功能性验证，其可靠性和稳定性主要依赖原厂自身进行的大量测试及其报告。

1.功能性验证

2.基本产品生命周期可靠性验证

3.软件兼容性、稳定性验证

存储类型	功能性验证	颗粒可靠性验证	软件兼容性、稳定性验证
SPI Nor	Y	N*1	N
SPI Nand	Y	Y	TBD*2
SLC Nand	Y	N*3	TBD*2

注释说明：

1. SPI Nor 只做功能性验证，主要是考虑到其相对稳定、兼容性较好且测试要达到较大数量级才能测试到边界量，可靠性和稳定性验证由颗粒原厂自身保证
2. TBD：只对兼容性改动较大的存储颗粒、存储驱动版本更新较大的部分做相应测试，通常不做该项测试
3. SLC Nand 的 ECC part 主要由 RK Nand 主控提供，可纠错达到 16bits/ 1KB，所以冗余较大，稳定性较高，不针对其可靠性做进一步验证，可靠性和稳定性验证由颗粒原厂自身保证

验证方法：

1.功能性验证

- 主控兼容，系统能正常boot起来，基础功能正常，flash 10 loop 压力测试 pass

2.颗粒可靠性验证

Retention测试：

- 对flash进行一定比例的P/E cycle处理后，进行烤机，模拟产品寿命

3.软件兼容性、稳定性的异常掉电验证(建议为实际产品)

异常掉电测试

- 1分钟掉电1次
- 上电进入到linux系统
- 上电期间不断进行dd命令读写，不做比对，主要测试ftl数据搬移数据良好
- 持续7天（达到大部分产品基本需求，约10K读写，1万次左右掉电）

3.2 RK Flash 送样要求简述

3.2.1 验证相关须知

通用说明：

1. 小容量存储包括 SLC PP Nand、SPI Nand、SPI Flash
2. 优先推送主要在售颗粒
3. 该渠道为验证驱动，暂无业务相关建议可回复
4. 通常小容量存储支持列表更新周期为 1~2 个月，实际颗粒验证时间不定，但会在更新支持列表前完成验证
5. 至少寄送 10 pcs，并提供对应的颗粒手册
6. 优先验证客户亟需的颗粒

SPI Nand 特殊说明：

1. 颗粒需自带 ECC 模块，否则不支持，RK 主控没有集成 ECC 模块
2. Flash 尾部没有连续坏块，否则将破坏尾部坏块表信息，如无法解决则无法支持
3. 无需 plane select bits 即可片选 odd plane block，部分颗粒为 2 plane 结构，需要传输地址上置上 plane slect bits 才能选中 odd plane 数据，该颗粒 RK 平台存在兼容问题，驱动需做兼容性处理，不建议使用

SPI Nor 特殊说明：

1. SPI Nor 颗粒要求 tRST 少于 200us

3.2.2 验证流程

- 内部会做功能测试和压力测试以验证颗粒与主控的兼容性，但颗粒自身的稳定性和压测表现以原厂自身标定为准
- 仅在经典平台上验证，RK 小容量存储主控 IP 为兼容 IP，所以仅验证 RK3568 但能兼容大部分有相应控制器件的 SOC，例如 RK3568 验证 OK 的 SPI Nand，对于 RV1126 RK3308 都是兼容的

3.2.3 验证邮寄地址

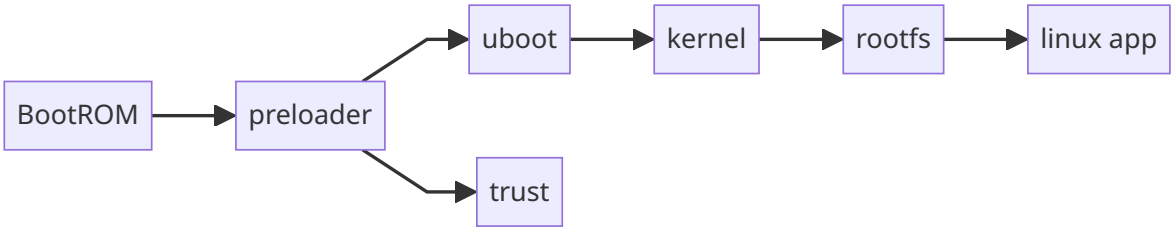
RK Redmine 问题平台上提出申请，由存储模块软件工程师提供地址。

3.2.4 客户补丁推送

- 由于 flash 新物料，尤其是 SPI Flash 物料，完成验证后通常要有源码补丁更新，如果客户有驱动需求，请在 RK Redmine 问题反馈平台提出，届时会在线提供补丁包

4. 设备启动流程

启动流程是指系统上电到系统启动完成的一个软件流程，下面是 linux 系统启动流程：



4.1 RK SOC BOOTROM Boot 支持状态

芯片名称	Emmc Boot	Nand Boot	SPI NAND Boot	SD Boot	SPI NOR Boot
RV1108	Y	Y	Y	Y	Y
RV1126/RV1109	Y	Y	Y	Y	Y
RK2108	Y	N	N	N	Y
RK2206	Y	N	N	Y	Y
RK3036	Y	Y	Y	Y	Y
RK3126C	Y	Y	Y*1	Y	Y
RK3128	Y	Y	Y	Y	Y
RK3229	Y	Y	Y*1	Y	Y
RK3288	Y	Y	Y	Y	Y
RK3308	Y	Y	Y	Y	Y
RK3326/PX30	Y	Y	Y*1	Y	Y
RK3328	Y	N	Y*1	Y	Y
RK3368/PX5	Y	Y	Y*1	Y	Y*1
RK3399	Y	N	Y*1	Y	Y
RK3568/RK3566	Y	Y	Y	Y	Y
RK3588	Y	N	Y	Y	Y
RV1106/RV1103	Y	N	Y	Y	Y
RK3528	Y	N	Y	Y	Y
RK3562	Y	N	Y	Y	Y

*1： 芯片硬件支持，SDK release 开发包未做支持。

4.2 RK SOC 存储接口规格

AP	NANDC	SPI0	SPI1	SPI2	SFC	SD	SDIO	EMMC	USB0	USB1
RK3188	60bits MLC SLC	Boot		-	-	SD 3.0	SDIO 3.0	SD50 DDR50	2.0 Host	2.0 OTG
RK3128	60bits MLC SLC		-	-	Boot	SD 3.0	SDIO 3.0	SD50 DDR50	2.0 Host	2.0 OTG
RK3126	60bits MLC SLC		-	-	Boot	SD 3.0	SDIO 3.0	SD50 DDR50	2.0 Host	2.0 OTG
RK3036	60bits MLC SLC		-	-	Boot	SD 3.0	SDIO 3.0	SD50 DDR50	2.0 Host	2.0 OTG
RK3288	60bits MLC SLC			Boot	-	SD 3.0	SDIO 3.0	HS200	2.0 Host	2.0 OTG
RK3399	-	-	Boot	-	-	SD 3.0	SDIO 3.0	HS400 HS200	3.0 OTG TYPEC	3.0 OTG TYPEC
RK3368	60bits MLC SLC				Boot	SD 3.0	SDIO 3.0	HS200	2.0 Host	2.0 OTG
RK3228 RK3229	60bits MLC SLC	Boot	-	-	-	SD 3.0	SDIO 3.0	HS200	2.0 Host	2.0 OTG
RK3328	-			Boot		SD 3.0	SDIO 3.0	HS200	2.0 Host	3.0 OTG
RK3228H	-			Boot		SD 3.0	SDIO 3.0	HS200	2.0 Host	3.0 OTG
RK3128X	60bits MLC SLC	Boot	-	-	-	SD 3.0	SDIO 3.0	SD50 DDR50	2.0 Host	2.0 OTG
RV1107 RV1108	16bits SLC				Boot	SD 3.0	SDIO 3.0	HS200	-	2.0 OTG
RV1109 RV1126	16bits SLC				Boot	SD 3.0	SDIO 3.0	HS200	-	2.0 OTG
RK3308	16bits SLC				Boot	SD 3.0	SDIO 3.0	HS200		
RK3326	70bits TLC MLC SLC				Boot	SD 3.0	SDIO 3.0	HS200		
RKPX3	60bits MLC SLC	Boot		-	-	SD 3.0	SDIO 3.0	SD50 DDR50	2.0 Host	2.0 OTG
RKPX3SE	60bits MLC SLC				Boot	SD 3.0	SDIO 3.0	SD50 DDR50		
RKPX5	60bits MLC SLC				Boot	SD 3.0	SDIO 3.0	HS200	2.0 Host	2.0 OTG
RKPX30	70bits TLC MLC SLC				Boot	SD 3.0	SDIO 3.0	HS200		
RK1608	-	Boot	-	-	-	-	-	-		
RK1808	-				Boot	-	-	HS200		

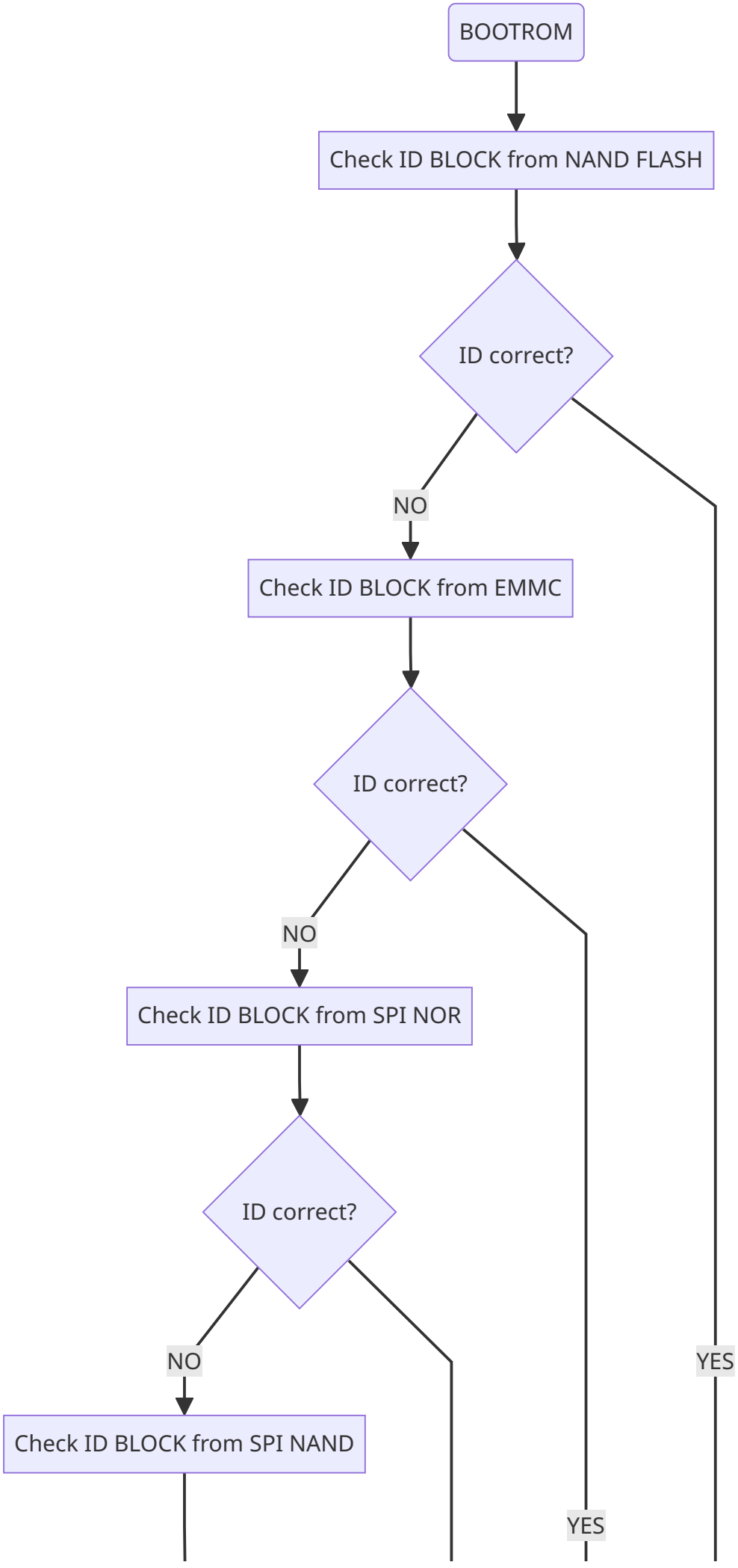
AP	NANDC	SPI0	SPI1	SPI2	SFC	SD	SDIO	EMMC	USB0	USB1
RK3568 RK3566	70bits TLC MLC SLC				Boot	SD 3.0	SDIO 3.0	HS200	2.0 Host	3.0 OTG
RK3588	-	-	-	-	Boot	SD 3.0	SDIO 3.0	HS400	2.0 Host	3.0 OTG
RV1106/RV1103	-	-	-	-	Boot	SD 3.0	-	HS50		
RK3528	-	-	-	-	Boot	SD 3.0	SDIO 3.0	HS400	2.0 Host	3.0 OTG
RK3562	-	-	-	-	Boot	SD 3.0	SDIO 3.0	HS400	2.0 Host	3.0 OTG

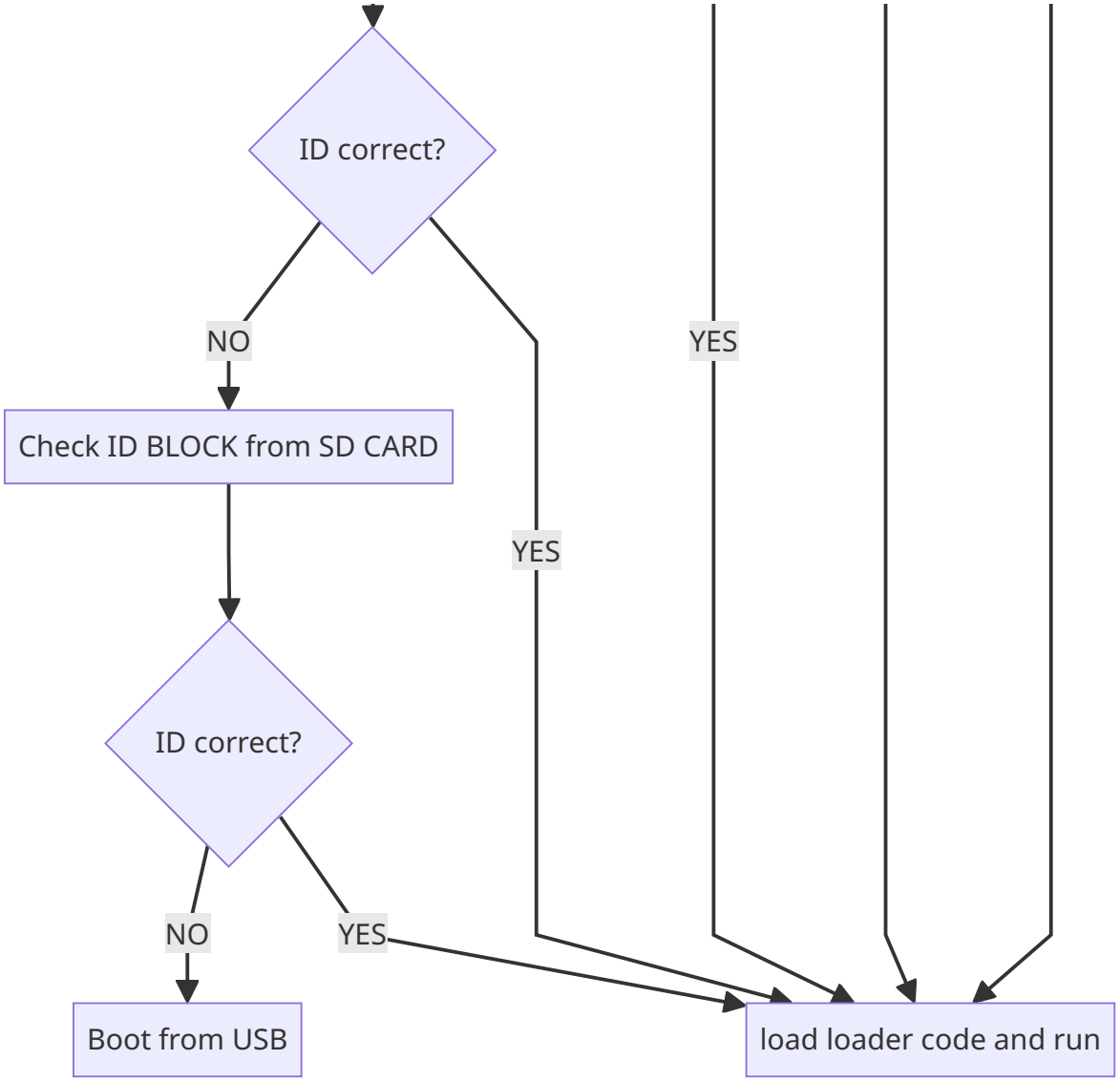
备注：表格中SPIx 和 SFC 有 Boot 标识的表示 BOOTROM 可以从对应的 SPI/SFC 接口启动。

4.3 BOOTROM 流程

RK 芯片上电后最新执行代码是集成在芯片内部不可更改的掩膜代码，也就是 BOOTROM 代码，AP 和 MCU 都有集成，系统上电时先运行 BOOTROM 代码，然后 BOOTROM 代码会探测外设存储器并加载其中的 Loader 代码。

不同芯片，BOOTROM 探测外设存储器的顺序不同。下图是 BOOTROM 启动流程图一个例子：


BOOTROM
Check ID BLOCK from NAND FLASH
ID correct?
NO
YES
Check ID BLOCK from EMMC
ID correct?
NO
YES
Check ID BLOCK from SPI NOR
ID correct?
NO
YES
Check ID BLOCK from SPI NAND



说明：

- 部分芯片设置支持 ADC Key 不同输入电平来指定 BOOTROM 探测的存储器件，如果该存储探测失败，直接进入 maskrom mode
- 探测存储器件通常是通过探测器件 ID 来确认是否由外挂设备
- 如果所有器件都未探测到有效固件，设备进入 maskrom mode，等待特定的接口如 usb/uart 进行固件下载，不是所有芯片都支持 usb/uart 接口升级

各芯片 BOOTROM 启动顺序

AP	No.1	No.2	No.3	No.4	No.5	No.6
RK3188	SD0	NAND	SPI NOR(SPI0)	SPI NAND(SPI0)	EMMC	USB
RK3128	NAND	EMMC	SPI NOR(SFC)	SPI NAND(SFC)	SD0	USB
RK3126(B)	NAND	EMMC	SPI NOR(SFC)	SPI NAND(SFC)	SD0	USB
RK3036	NAND	EMMC	SPI NOR(SFC)	SPI NAND(SFC)	SD0	USB
RK3288	NAND	EMMC	SPI NOR(SPI2)	SPI NAND(SPI2)	SD0	USB
RK3399	SPI NOR(SPI2)	SPI NAND(SPI2)	EMMC	SD0	USB	--
RK3368	NAND	EMMC	SPI NOR(SFC)	SPI NAND(SFC)	SD0	USB
RK3228/9	NAND	EMMC	SPI NOR(SPI2)	SPI NAND(SPI2)	SD0	USB
RK3328	EMMC	SPI NOR(SPI2)	SPI NAND(SPI2)	SD0	USB	--
RK3228H	EMMC	SPI NOR(SPI2)	SPI NAND(SPI2)	SD0	USB	--
RK3128X/H	NAND	EMMC	SPI NOR(SPI2)	SPI NAND(SPI2)	SD0	USB
RV1107/8	NAND	EMMC	SPI NOR(SFC)	SPI NAND(SFC)	SD0	USB
RV1109	SPI NOR(SFC)	SPI NAND(SFC)	NAND	EMMC	SD0	USB
RV1126	SPI NOR(SFC)	SPI NAND(SFC)	NAND	EMMC	SD0	USB
RK3308	NAND	EMMC	SPI NOR(SFC)	SPI NAND(SFC)	SD0	USB
RK3326	NAND	EMMC	SPI NOR(SFC)	SPI NAND(SFC)	SD0	USB
RKPX3	SD0	NAND	SPI NOR(SPI0)	SPI NAND(SPI0)	EMMC	USB
RKPX3SE	NAND	EMMC	SPI NOR(SFC)	SPI NAND(SFC)	SD0	USB
RKPX5	NAND	EMMC	SPI NOR(SFC)	SPI NAND(SFC)	SD0	USB
RKPX30	NAND	EMMC	SPI NOR(SFC)	SPI NAND(SFC)	SD0	USB
RK1608	SPI SLAVE	SPI NOR(SPI2)	SPI NAND(SPI2)	--	--	--
RK1808	SPI SLAVE	SPI NOR(SFC)	SPI NAND(SFC)	EMMC	USB	--

AP	No.1	No.2	No.3	No.4	No.5	No.6
RK3399PRO	SPI NOR(SPI2)	SPI NAND(SPI2)	EMMC	SD0	USB	--
RK3568	SPI NOR(SFC)	SPI NAND(SFC)	NAND	EMMC	SD0	USB
RK3566	SPI NOR(SFC)	SPI NAND(SFC)	NAND	EMMC	SD0	USB
RK3588	SPI NOR(SFC)	SPI NAND(SFC)	EMMC--	SD0	USB	
RV1106/RV1103	SPI NOR(SFC)	SPI NAND(SFC)	EMMC--	SD0	USB/UART	
RK3528	SPI NOR(SFC)	SPI NAND(SFC)	EMMC--	SD0	USB	
RK3562	SPI NOR(SFC)	SPI NAND(SFC)	EMMC--	SD0	USB	

4.4 Pre Loader 流程

RK SDK 编译工程输出的镜像通常是可以看到一个 MiniloaderAll.bin 的镜像，这个镜像是实际上是有两个主要功能：

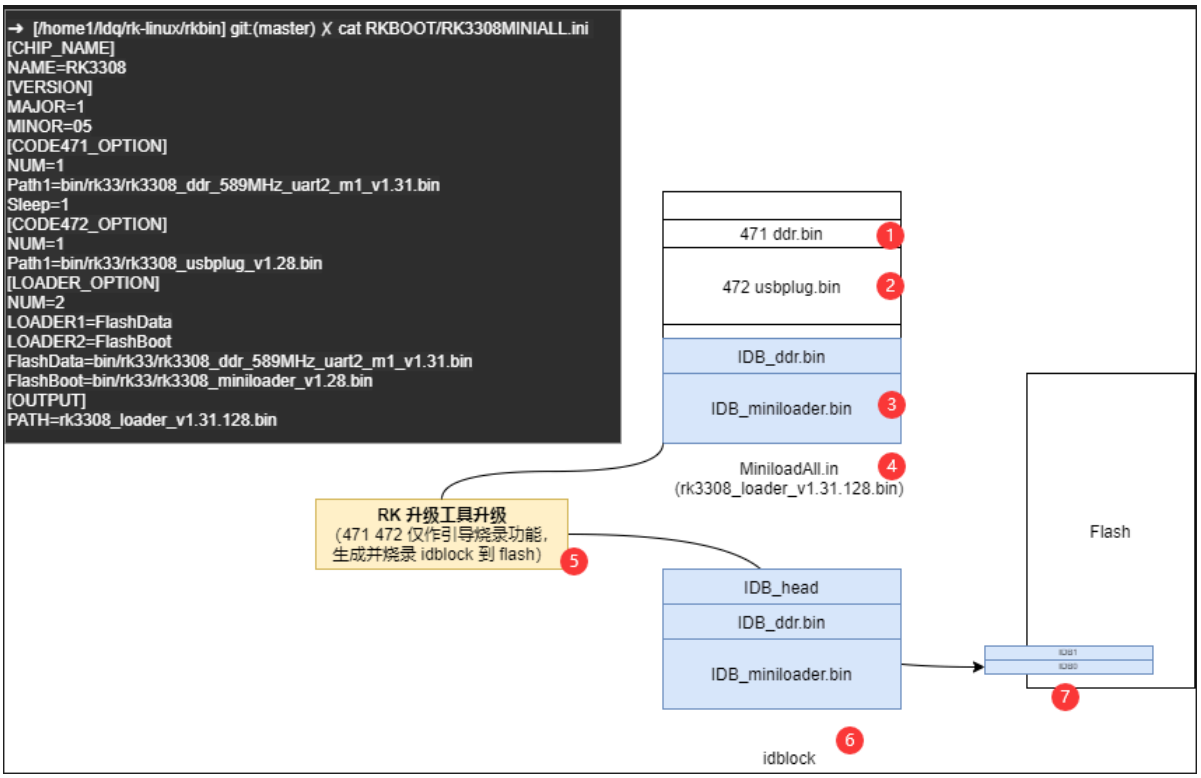
- 引导烧录
- 烧录 idb 镜像

idb 镜像是 MiniadloerAll.bin 解包出来的有效镜像，又称为 idblock，最终烧录到 flash 中，通常 idb 镜像由 ddr.bin 和 Pre Loader 镜像打包而成，部分芯片支持打包更多的功能镜像。Pre Loader 目前主要有 3 种： miniloader（非开源）， uboot spl 和 loader。

4.4.1 Miniloader

简介

miniloader 固件是 RK 非开源的 preLoader 固件，通常打包与 ddr.bin usbplug.bin 打包为 loader.bin，结构如下：



注释：

1. ddr.bin：ddr 初始化固件，简称 471
2. usbplug.bin：引导烧录所使用的运行固件，简称 472
3. miniloader.bin：闭源 pre-loader 固件
4. MiniloaderAll.bin：为 SDK 统一命名文件，实际为对应芯片打包固件的 loader.bin
5. RK 升级工具升级过程会从 loader.bin 中提取并升级 idblock.bin，并作多备份处理
6. idblock， 固件为 ddr.bin + pre-loader 的打包固件，闭源版本 pre-loader 就是指 miniloader.bin

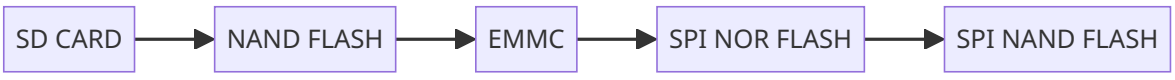
rkbin 仓库打包生成 Miniloader

以 rk3308 为例，进入 SDK 中 rkbin 目录，最终生成 rk3308_loader_v1.xx.1xx.bin：

```
./tools/boot_merger ./RKBOOT/RK3308MINIALL.ini .
./tools/boot_merger ./RKBOOT/RK3308MINIALL_WO_FTL.ini . /* 尾缀为 _WO_FTL 文件、为开源存储方案选用的 Pre Loader，文件系统选择ubifs或者jaffs2 */
./tools/boot_merger ./RKBOOT/RK3326MINIALL_SLC.ini . /* 尾缀为 _SLC 文件、为小容量存储（SLC Nand、SPI Nand、SPI Nor）方案专用的 Pre Loader，内带rk ftl算法，不支持ubifs */
```

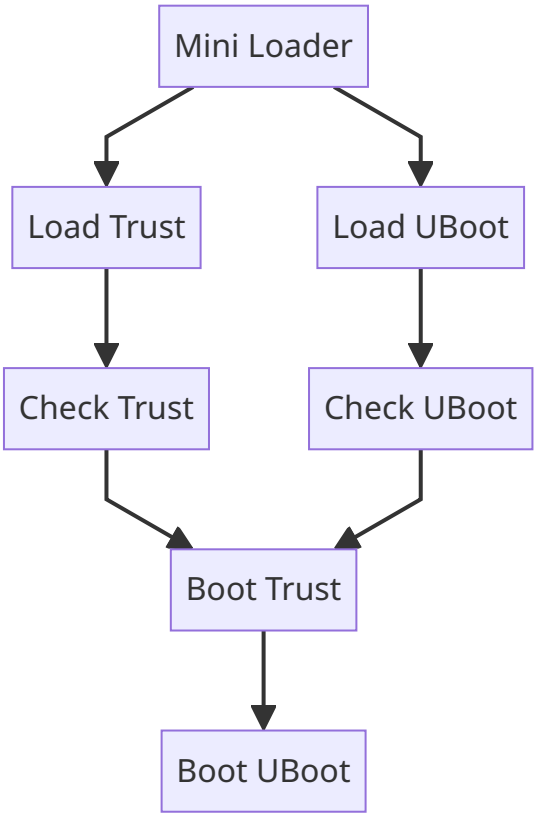
存储探测顺序

如同 BootRom，为兼容不同存储类型，Preloader 阶段也会去探测不同存储外设：



由于代码没有开源，用户不能自行修改启动顺序。

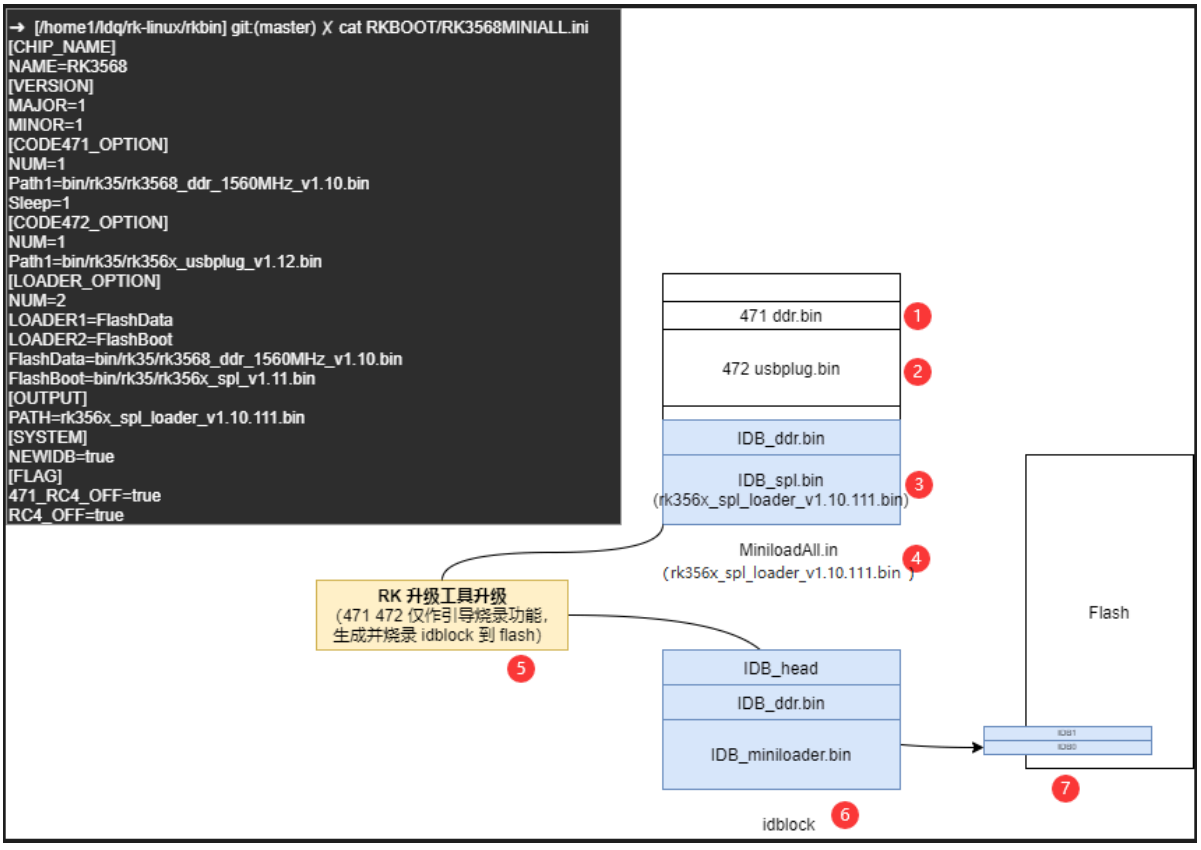
启动流程



4.4.2 u-boot spl

芯片支持情况参考文档《Rockchip-Developer-Guide-UBoot-nextdev-CN》，支持 NAND 和 SPI NAND 时不带 FTL 算法，只用开源 NAND 驱动，建议使用 UBIFS 文件系统。

简介



注释：

1. ddr.bin：ddr 初始化固件，简称 471
2. usbplug.bin：引导烧录所使用的运行固件，简称 472
3. spl.bin：u-boot 源码下编译出来的 spl 固件，输出文件在 uboot 目录下的 spl/u-boot-spl.bin
4. MiniloaderAll.bin：为 SDK 统一命名文件，实际为对应芯片打包固件的 spl_loader.bin
5. RK 升级工具升级过程会从 loader.bin 中提取并升级 idblock.bin，并作多备份处理
6. idblock， 固件为 ddr.bin + pre-loader 的打包固件，spl 版本 pre-loader 就是指 u-boot-spl.bin，RK SDK 会定期编译 u-boot-spl.bin 贮存在 rkbin 对应目录，命名为 rkxxxx_spl_vx.xx.bin
7. PC 升级工具 SLC Nand/SPI Flash idblock 镜像会做双倍份，其他存储器件 idblock 镜像五备份

结合 BOOTROOM 的流程，理解 "加载 xxx.bin 运行 xxx 功能"行为：

- 烧录流程：
  - BOOTROM 加载 ddr.bin，初始化 ddr
  - BOOTROM 加载 usbplug.bin，通过 usb plug 固件（烧录）
- 启动流程：
  - BOOTROM 加载 ddr.bin，初始化 ddr
  - BOOTROM 加载 spl.bin，执行 Loader 功能，Load 后级固件
  - 部分芯片还有在 idblock 中打包 mcu.bin、PCIe.bin 也是类似行为和目标，pre-load the pre-loader

spl 存储探测顺序

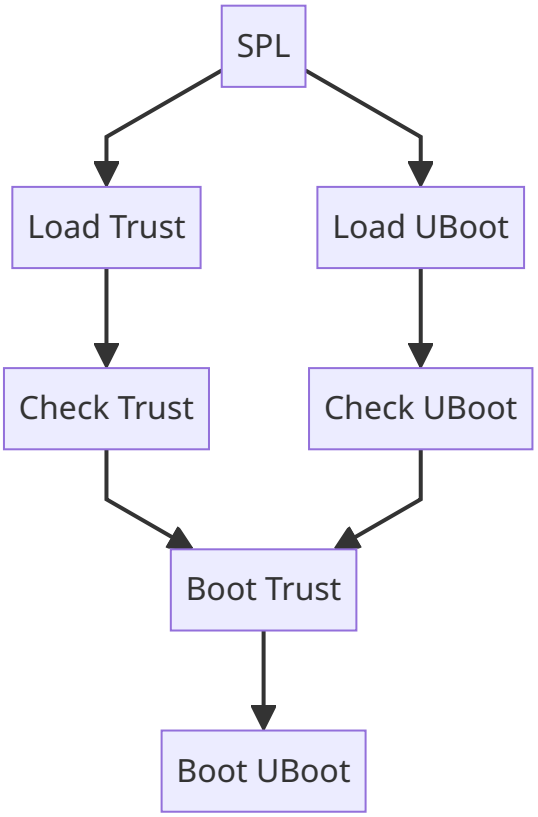
说明：

- spl 支持开启 SD 卡启动功能，探测到 SD 卡及 SD 卡内的有效固件，加载 SD 卡中的后期固件，完成 SD 启动
- 支持 atags 的 spl 固件优先检测使用 BOOTROM 探测成功的存储器件启动，atags 功能详细参考 u-boot 开发手册

spl 存储相关 log

```
U-Boot SPL 2017.09-gcc781e0266-230509-dirty #ldq (Nov 24 2023 - 00:15:39)
unknown raw ID 0 0 0
unrecognized JEDEC id bytes: 00, 00, 00
Trying to boot from MMC2
MMC: no card present
mmc_init: -123, time 0
spl: mmc init failed with error: -123
Trying to boot from MMC1                # bootdev 探测结果：MMC2(SD Card)、
MMC1(EMMC)、MTD0(SLC Nand)、MTD1(SPI Nand)、MTD2(SPI Nor)
No misc partition
Trying fit image at 0x4000 sector
```

启动流程



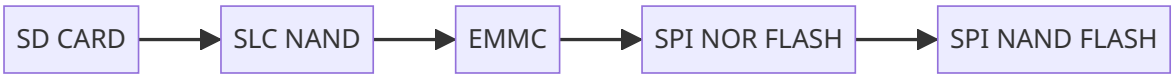
4.4.3 loader

支持 RV1107、RV1108、RK3036、RK3128 和 RK3229 等平台，一般用于支持小容量存储，不使用 uboot，直接引导 kernel。

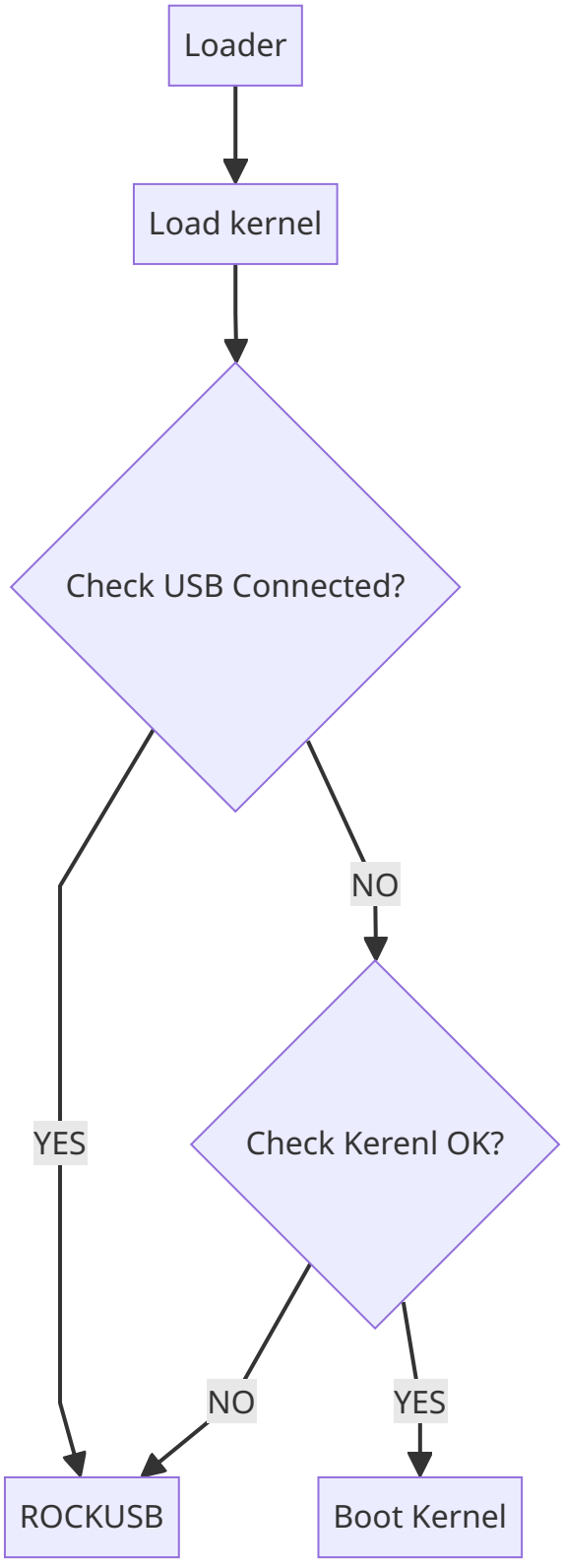
芯片	SD CARD	SLC NAND	EMMC	SPI NOR	SPI NAND
RV1107/8	支持	支持	支持	支持	支持
RK3036	支持	支持*1	支持*1	支持	支持
RK3128	支持	支持*1	支持*1	支持	支持
RK3229	支持	支持*1	支持*1	支持	不支持

*1 RK3036、RK3128 和 RK3229 使用 SLC NAND 和 EMMC 的项目，一般直接用 miniloader。

存储探测顺序



启动流程


Loader
Load kernel
Check USB Connected?
NO
YES
Check Kerenl OK?
NO
YES
ROCKUSB
Boot Kernel

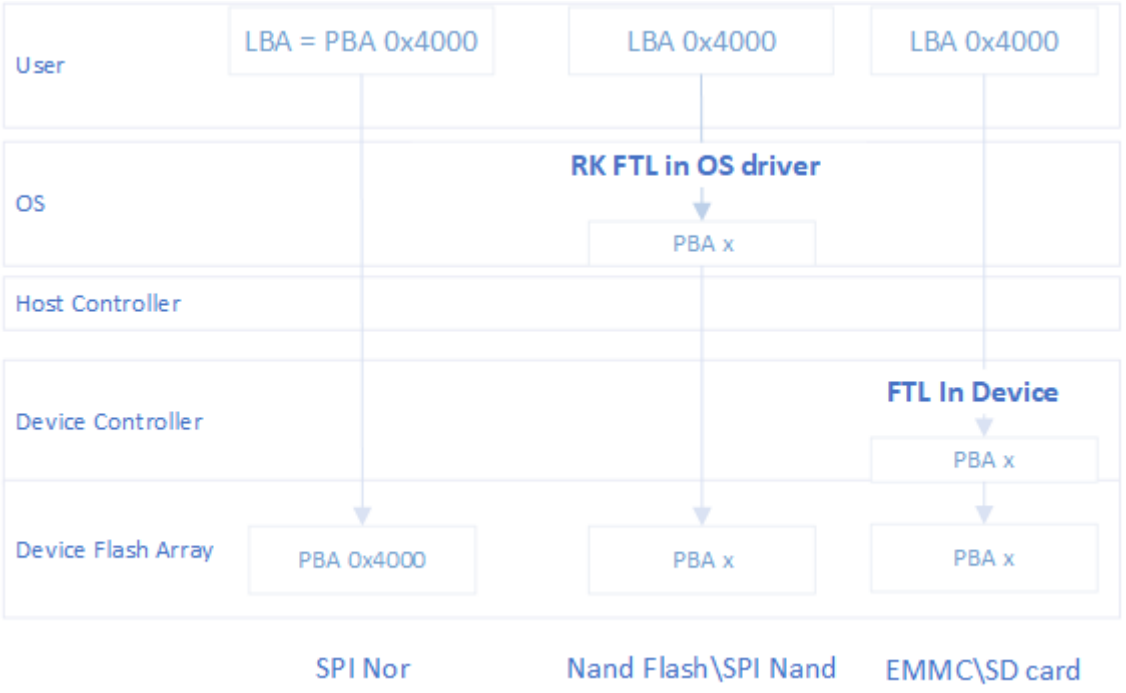
5. 分区及数据存储

5.1 数据存储

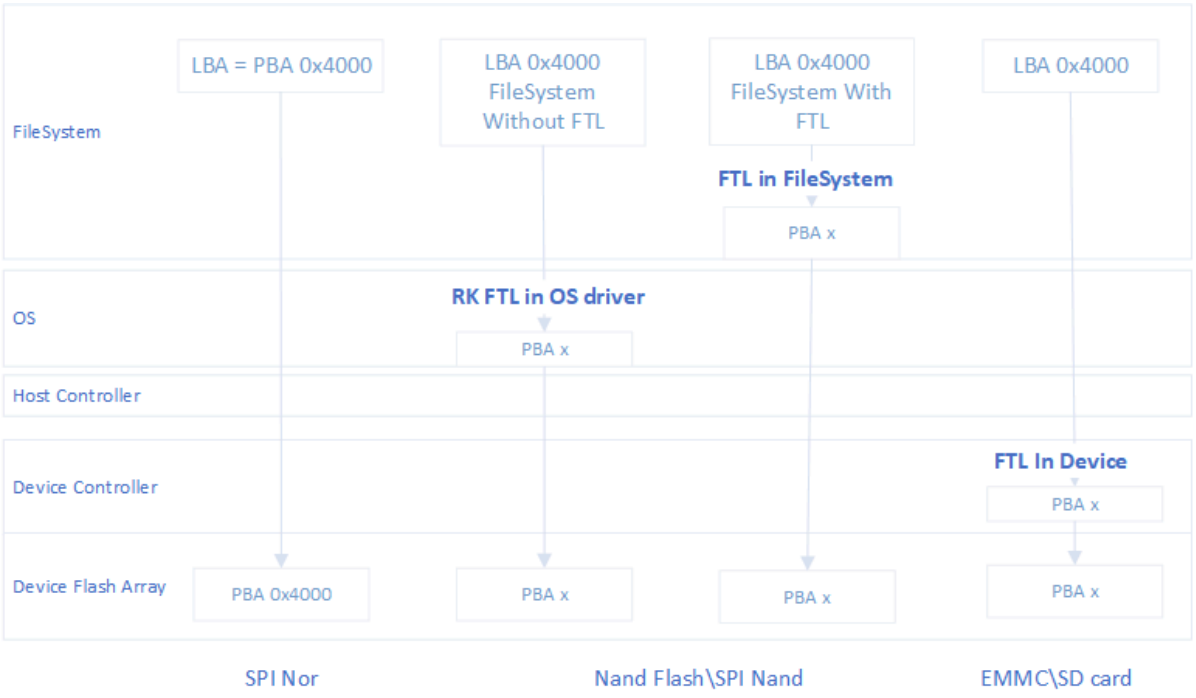
5.1.1 地址转换简介

如果对存储有所了解，那么应该会知道存储颗粒大多不是平坦映射，而是由用户逻辑扇区地址（lba）转换到 Flash 物理扇区地址（pba），这种映射过程即 FTL （Flash translation layer）,FTL 需要综合数据磨损、坏块管理、垃圾回收等需求进行地址转换。不论是否有 FTL，用户和文件仅需关心逻辑地址即可，而地址转换细节由软件完成。

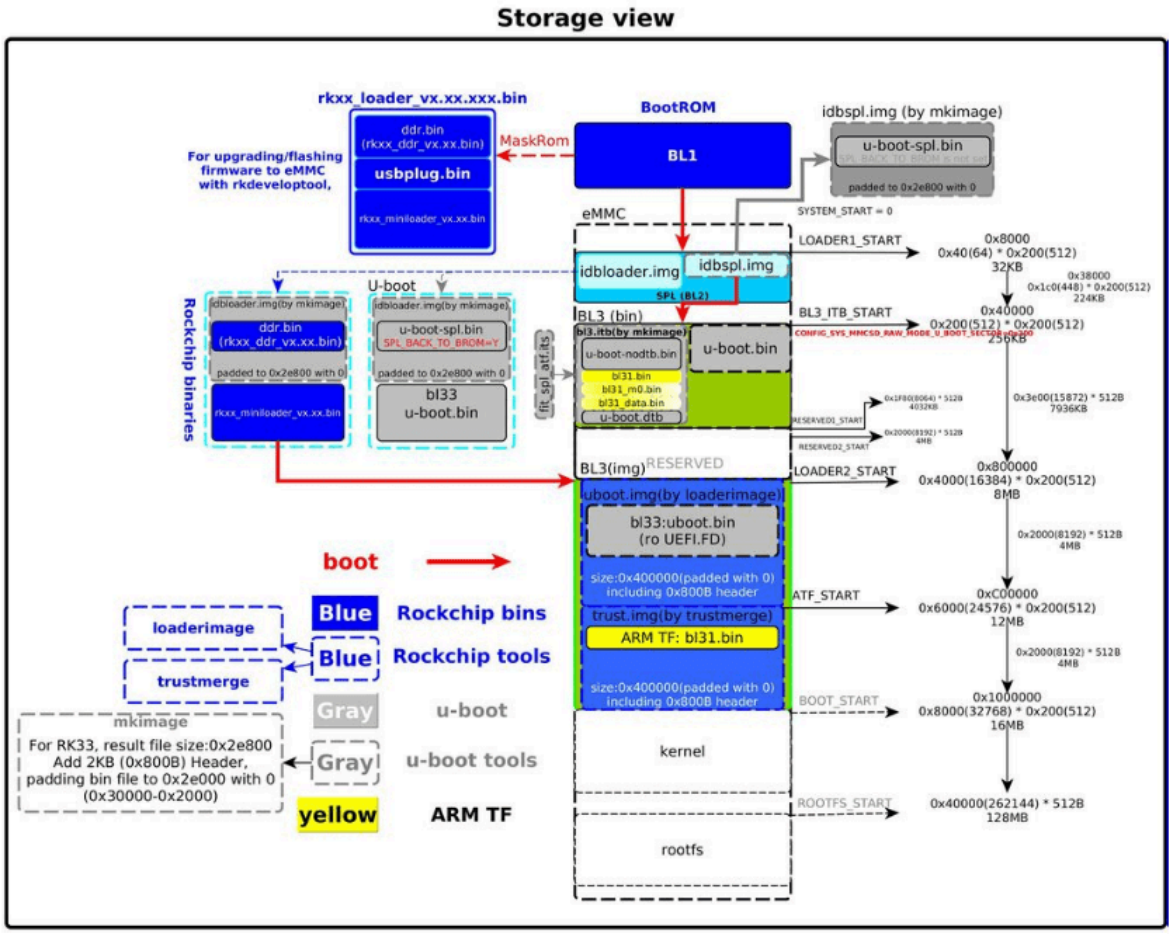
假定用户需要访问地址 0x4000 扇区，地址转换关系如下：



假定文件系统内接口访问地址 0x4000 扇区，地址转换关系如下：



5.1.2 分区及数据逻辑地址存储



5.2 分区表分区

RK 存储方案中一共有 3 种分区表可固化到存储分区中：MTD Partition，GPT 和 RK partition。

详细的信息可以参考文档《Rockchip_Introduction_Partition》。

分区	说明	适用平台	限制
MTD Partition	parameter 文件中定义，通过 cmdline 传递，uboot-next 分支开始不再提供支持	所有 AP*1	需要独立分区存放
GPT	EFI 通用分区表，uboot-next 分支支持	所有 AP*2	占用资源多一点
RK partition	参考 GPT 设计，主要用于小容量存储，节省资源	RV1107/8, MCU	RK 自定义，不通用

*1 使用 uboot-next 分支的平台不再支持 MTD partition，如果需要使用，需要自己适配。

*2 使用 uboot-next 分支的平台默认都是使用 GPT 做分区表，如果需要使用其他分区表，需要自己适配。

5.2.1 MTD Partition

参考 《Rockchip_Developer_Guide_Linux_Flash_Open_Source_Solution_CN.md》 文档。

5.2.2 GPT

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. 需要指定 rootfs 的 uuid，不同SDK可能设定值不同，需要和DTS里面定义的rootfs uuid匹配。

```
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 /* GPT 分区 */
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),0x00000400@0x00554000 (frp),-@0x00554400 (userdata:grow)
uuid:rootfs=614e0000-0000-4b53-8000-1d28000054a9
```

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

   注： 1. parameter 文件本身不会被烧写到存储设备中。

5.2.3 RK partition

RK 自定义的一种分区表，结构和 GPT 类似，占用资源少，初始化更快，主要用在 RV1107/8 平台和 MCU 平台。

下面为 Linux\Android 产品分区定义文件模板：

```
#Flag 目前只有两个值，1 为分区需要下载，0 为不需要下载
#type 目前有 5 种值，0x1=Vendor 分区 0x2=IDBlock 分区 0x4=Kernel 分区 0x8=boot 分区
0x80000000 = 普通分区
#PartSize 和 PartOffset 字段的值都是以扇区为单位
[System]
FwVersion=16.12.23
# 如果 Nano=1，则生成 nano 的 idblock
Nano=
# 如果 BLANK_GAP=1，则生成的 idblock 按每 2k 数据间隔 2k 空白保存
BLANK_GAP=1
#FILL_BYTE 表示分区尾部空白用什么数据填充，默认为 0
FILL_BYTE=
[IDBlock]
Flag=1
DDR_Bin=rk3399_DDR_800MHz_v1.17.bin
Loader_Bin=rk3399_miniloader_spi_nor_v1.14.bin
```

```
PartOffset=0x40
PartSize=0x780
[UserPart1]
Name=trust
Type=0x10
Flag=1
File=trust_1MB.img
PartOffset=0x800
PartSize=0x800
[UserPart2]
Name=uboot
Type=0x20
Flag=1
File=uboot_1MB.img
PartOffset=0x1000
PartSize=0x800
```

下面是 RTOS 产品分区定义文件模板，其中 Flag 标志中的 bits [8,10] 规范暂时仅在 RTOS 产品有效：

```
#Flag:
#  bits filed:
#  [0]    : skip                  : 0 - disabled (default), 1 - enable
#  [2]    : no partition size     : 0 - diabled (default), 1 - enable
#  [8, 9] : property              : 0 - do not register (default), 1 - read only,
2 - write only, 3 - rw
#  [10]   : register type         : 0 - block partition (default), 1 - MTD
partition
#type can suppot 32 partiton types,0x0:undefined 0x1:Vendor 0x2:IDBlock
,bit3:bit31 are available
#PartSize and PartOffset unit by sector
#Gpt_Enable 1:compact gpt,0:normal gpt
#Backup_Partition_Enable 0:no backup,1:backup
#Loader_Encrypt 0:no encrypt,1:rc4
#nano 1:generate idblock in nano format
[System]
FwVersion=1.0
Gpt_Enable=
Backup_Partition_Enable=
Nano=
Loader_Encrypt=
Chip=
Model=
[UserPart1]
Name=IDBlock
Type=0x2
PartOffset=0x80
PartSize=0x80
Flag=
File=../../Image/rk2108_loader.bin,../../Image/Boot2_Fake.bin
[UserPart2]
Name=rtthread
Type=0x8
PartOffset=0x100
PartSize=0xa00
```

```
Flag=
File=../../Image/rtthread.img
[UserPart3]
Name=root
Type=
PartOffset=0x1100
PartSize=0x6f00
Flag=0x305
File=../../Image/root.img
```

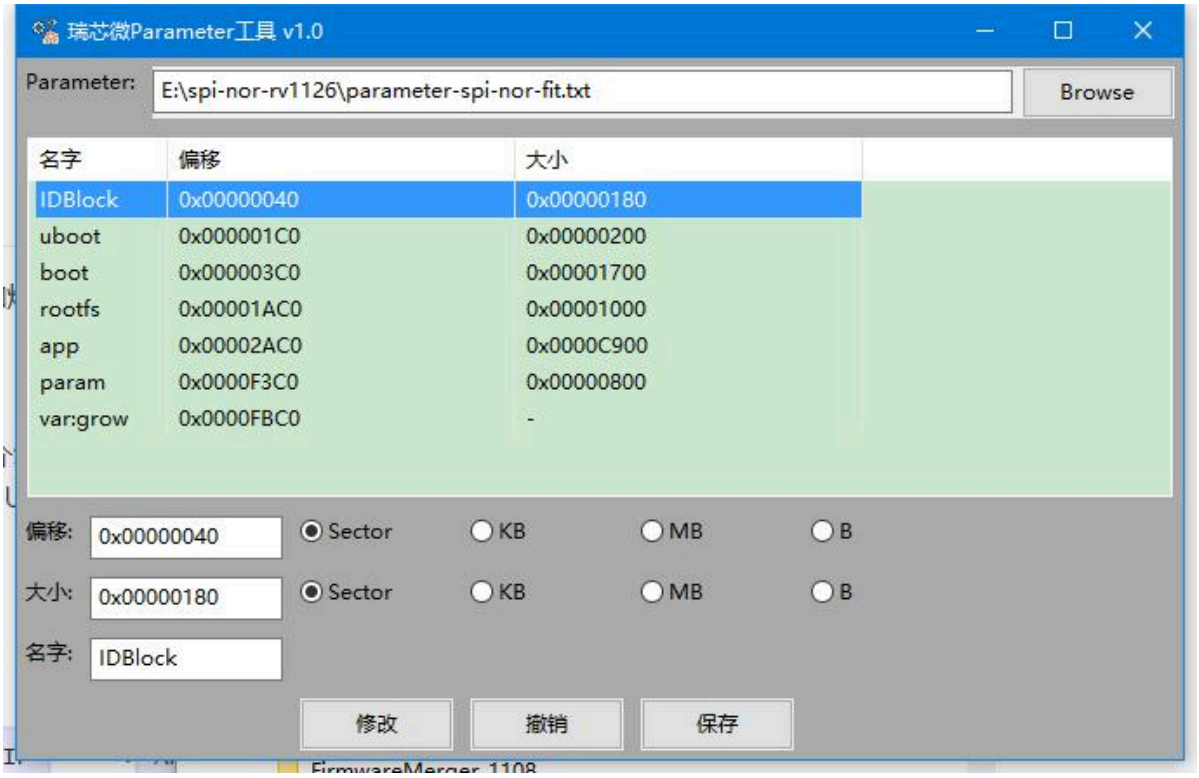
5.2.4 ENV 分区

ENV（Environment-Variables）是 U-Boot 支持的一种全局数据管理和传递方式，原理是构建一张 HASH 映射表，把用户的数据以"键值-数据"作为表项进行管理。

RK 部分芯片平台使用将 ENV 信息贮存在 flash 中，定义为 ENV 分区，同时将 cmdlines 中的 mtdparts 分区表信息生成并记录在 ENV 信息表中，并从 SPL 或 U-Boot 传递到内核。

5.3 parameter 分区表修改工具

分区表修改工具可以用于修改parameter定义的分区，一个分区大小被修改时，其后的分区偏移都会匹配修改。



5.4 分区写保护设置

5.4.1 块设备分区写保护设置

Linux Kernel下EMMC和SD CARD是块设备，NAND FLASH使用rknand 或者rkflash驱动时也是块设备，可以通过下面命令配置分区的读写属性。

示例1. 设置system分区为只读：

```
./busybox blockdev --setro /dev/block/by-name/system
```

示例2. 设置system分区为可读写：

```
./busybox blockdev --setrw /dev/block/by-name/system
```

注意：分区配置最好在分区mount之前，不然分区mount为可写，在配置分区属性为只读，文件系统会报错。

5.4.2 MTD设备分区写保护设置

mtd一般通过cmdline定义分区，可以在分区名后加上字符'ro'来设定这个分区为只读。可以在uboot传递cmdline给kernel的时候修改mtdparts来实现特定分区写保护。

示例：修改分区表，设置boot分区为只读：

```
mtdparts=rk29xxnand:0x00002000@0x00004000(uboot),0x00004000@0x00006000(boot)ro,..
.
```

6. 固件烧录

目前量产烧录固件主要有 3 种方式： USB 升级、SD 卡升级和烧录器烧录。

6.1 USB 升级

USB 升级目前有两种协议：rockusb 和 fastboot。本文档只介绍 rockusb 升级方式，如果需要用 fastboot 升级方式，可以参考 uboot 开发文档《Rockchip-Developer-Guide-UBoot-nextdev-CN》。

6.1.1 流程框图

AP rockusb: maskrom rockusb, miniloader rockusb and uboot rockusb。

NVM： SPI NOR， SPI NAND， SLC NAND， EMMC，M/TLC NAND。

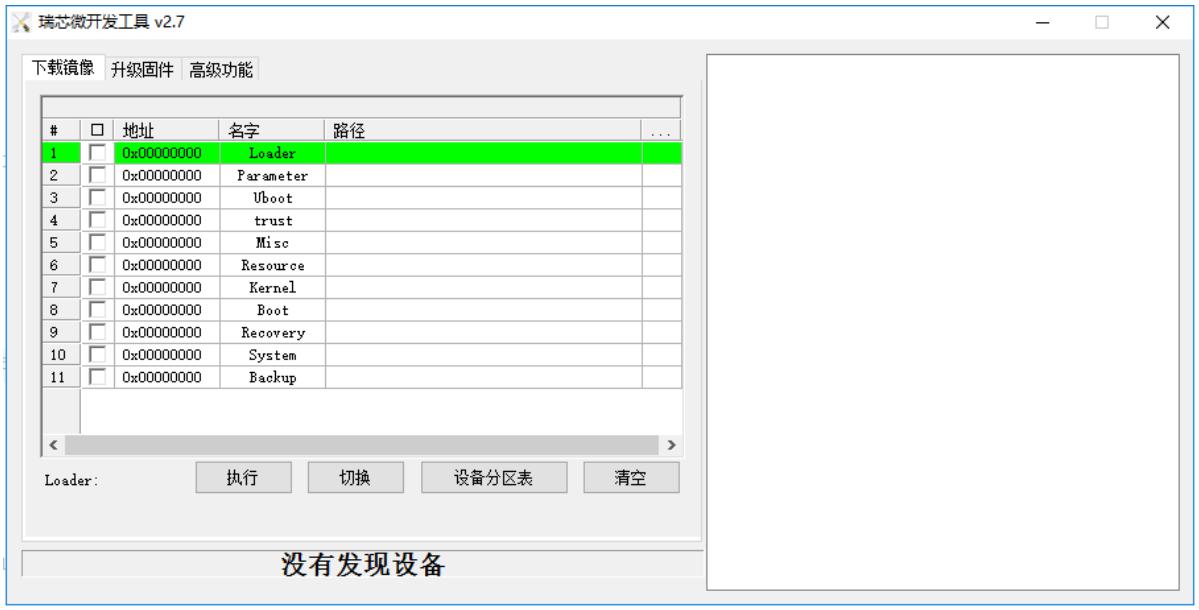
6.1.2 WIN 开发工具 RKDevTool

GPT/RK Partition 方案：

AP SDK 发布的时候会提供配置好的开发工具，用于开发时烧录完整固件或更新部分分区的数据。工具附带功能比较多，详细功能介绍参考工具自带的文档，这里介绍几个比较实用的功能：

1. 读取设备分区表：在 loader 升级模式，点击按钮 “设备分区表”，可以读取设备的分区表
2. 切换到 loader 升级模式：在 MSC 或者 MTP 模式下，可以点击 “切换” 按钮切换到 loader 升级模式
3. 从 loader 切换到 maskrom 升级模式： 在高级功能里面点击 “进入 maskrom” 按钮可以从 loader 升级模式切换到 maskrom 升级模式
4. 重启设备：在 loader mode 或者 maskrom mode 下，可以点击高级功能 “重启设备”

工具界面：



高级功能：

1. maskrom升级模式，需要选择loader文件下载到DDR里面运行
2. update.img 固件解包
3. 支持脚本运行
4. 读取FLASH ID
5. 读取FLASH信息
6. 读取芯片信息
7. 读取loader支持扩展功能
8. 测试测试是否ready
9. 重启设备
10. 重启进去maskrom升级模式，一般从loader升级模式切换到maskrom升级模式
11. 覆盖写数据，清空序列号，可能会破坏固件
12. 导出loader头部IDB结构
13. 根据16和17定义的起始地址和扇区数，擦除扇区，需要对齐到4MB，不然可能会多擦除或者少擦
14. 导出loader运行的串口信息，保存在工具的output目录

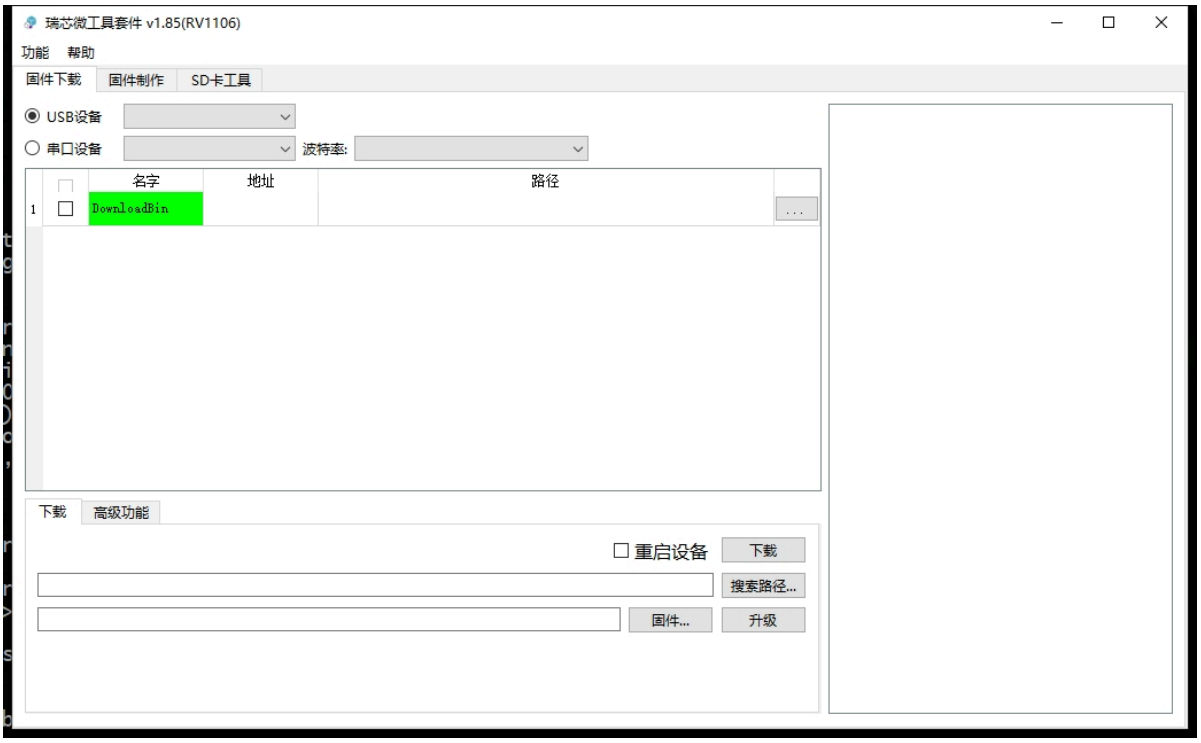
15. 根据16和17定义的起始地址和扇区数，导出固件镜像，保存在工具的output目录
16. 定义起始扇区
17. 定义操作操作的扇区数
18. 工具日志

6.1.3 WIN 开发工具 SocToolKit

ENV 方案：

RK 部分 AP 平台支持开源的 ENV 分区信息，支持分区表、bootargs 等信息记录在 ENV 分区表内，并通过 cmdlines 的方式传递到内核，该方案有特定的镜像打包方案和升级工具。

工具界面：



6.1.4 LINUX 开发工具 upgrade_tool

Linux 工具与安卓工具类似，都有相近的功能。

工具界面：

```
[/home/ldq] upgrade_tool -h
--------------------Tool Usage--------------------
Help:               H
Quit:               Q
Version:            V
Clear Screen:       CS
------------------Upgrade Command------------------
ChooseDevice:       CD
ListDevice:         LD
SwitchDevice:       SD
UpgradeFirmware:    UF <Firmware> [-noreset]
UpgradeLoader:      UL <Loader> [-noreset]
DownloadImage:      DI <-p|-b|-k|-s|-r|-m|-u|-t|-re image>
DownloadBoot:       DB <Loader>
EraseFlash:         EF <Loader|firmware> [DirectLBA]
PartitionList:      PL
----------------Professional Command----------------
TestDevice:         TD
ResetDevice:        RD [subcode]
ResetPipe:          RP [pipe]
ReadCapability:     RCB
ReadFlashID:        RID
ReadFlashInfo:      RFI
ReadChipInfo:       RCI
ReadSector:         RS  <BeginSec> <SectorLen> [-decode] [File]
WriteSector:        WS  <BeginSec> <File>
ReadLBA:            RL  <BeginSec> <SectorLen> [File]
WriteLBA:           WL  <BeginSec> <File>
EraseBlock:         EB <CS> <BeginBlock> <BlokcLen> [--Force]
----------------------------------------------------
```

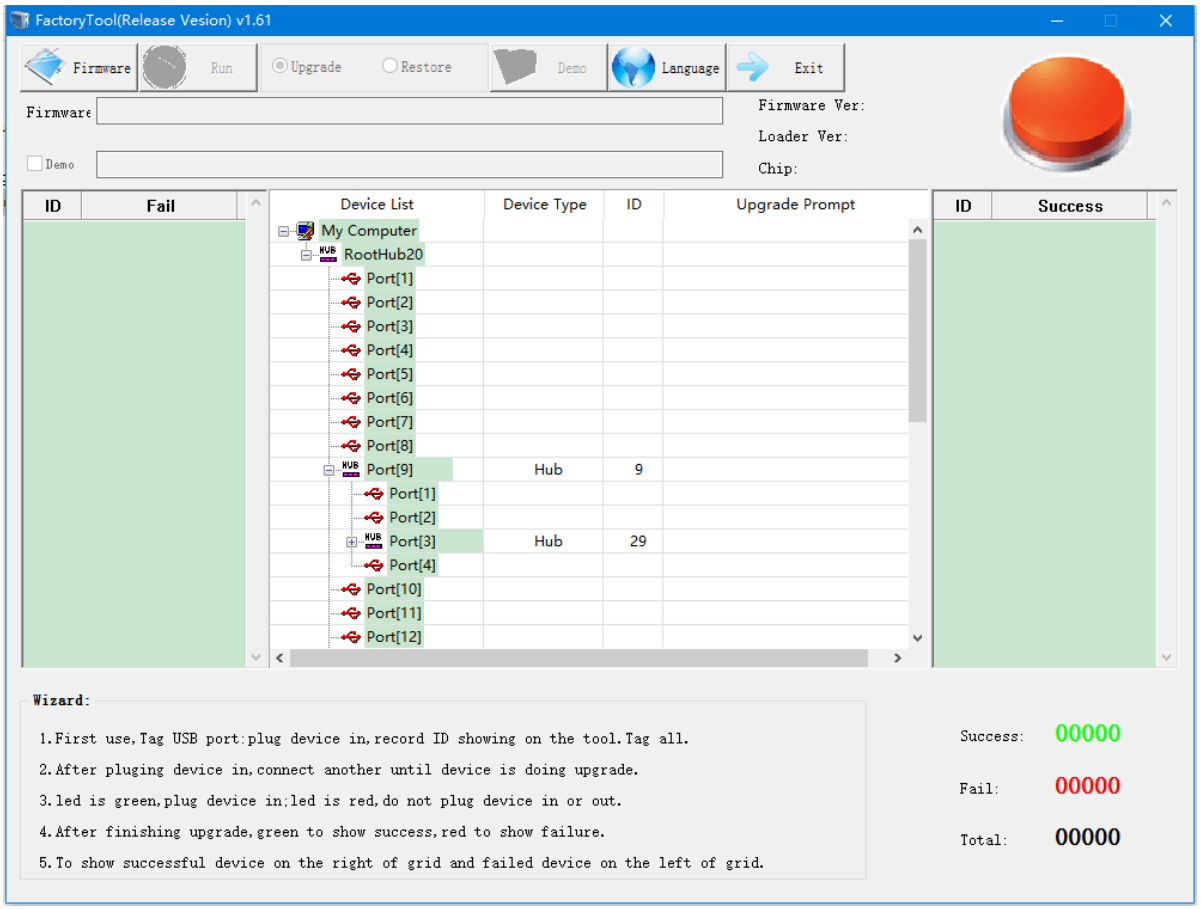
6.1.5 LINUX 开发工具 SocToolKit

Linux 工具与安卓工具类似，都有相近的功能。

6.1.6 量产工具

量产工具支持一拖多异步烧录固件，工具运行升级功能后，每接上一台设备，工具就会开始升级固件，多台机器之间独立的。

工具界面：



工具目录下有config.ini配置文件，每个选项都有详细注释，这里列举几个常用的配置：

1、FW_BURN_EFUSE 烧录固件的同时烧录efuse，启用secure boot。

AP用OTP，或者PCB没有预留EFUSE电源控制电路时不能开启这个功能。

2、NOTRESET_AFTER_UPGRADE 升级后不重启机器

有些产品第一次开机要求不能断电，需要设置升级固件后不重启。

3、FORCE_DATA_BAND 修改USB单包传输数据大小，烧写SPI NOR时如果出现usb超时出错，可以改小这个值。

4、SN_DLL_ON 开启升级固件过程同时烧写SN的功能

5、RB_CHECK_OFF固件升级是否需要回读调用

6.2 SD卡升级

使用 SD_Firmware_Tool 工具把update.img固件烧录到sd卡里面，把制作好的升级用SD（TF）卡插到机器的SD卡口，上电就会从sd卡启动到recovery并升级固件到机器内部存储中。
工具界面：

工具功能说明：

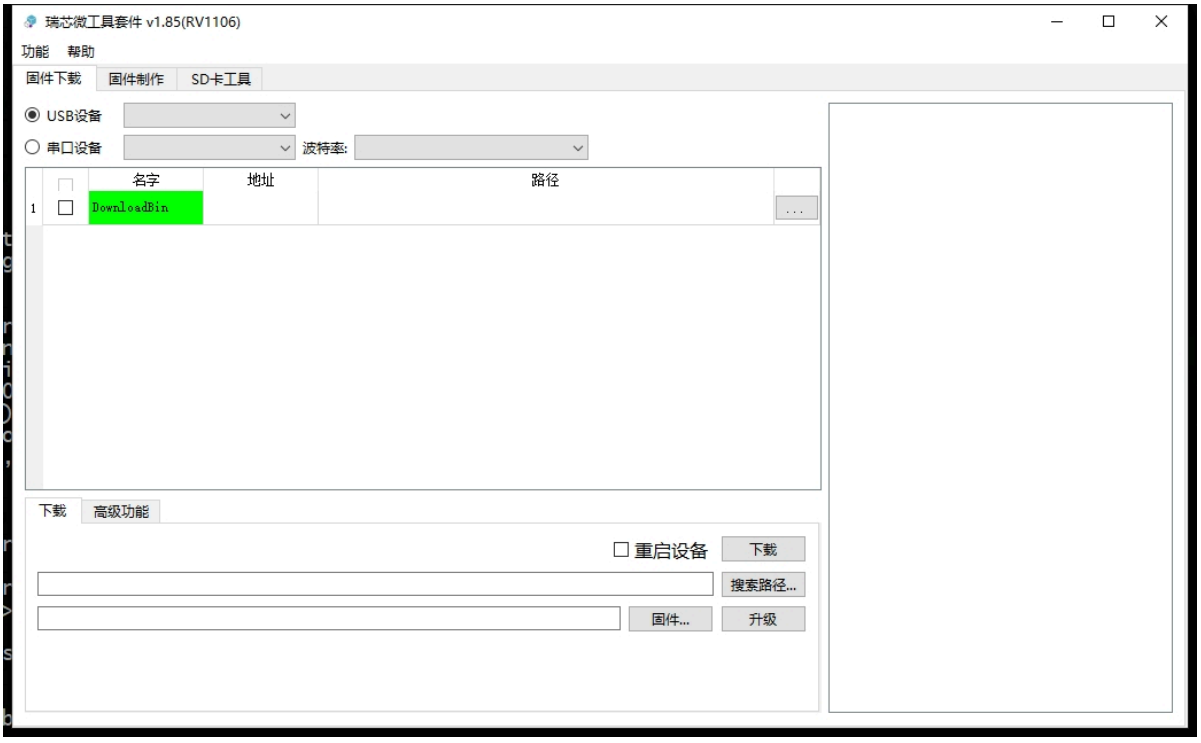
1、PCBA测试，勾选这个功能会先进行PCBA测试后再升级固件。

2、SD启动，制作启动卡，完整固件都存在在SD卡里面。

3、恢复磁盘，删除启动卡的启动代码，恢复位普通sd卡。

6.3 UART 升级

特定的芯片支持通过 UART 接口升级镜像，同样支持 linux、windows 和量产工具。



6.4 EMMC 镜像烧录

用SpiImageTools把update.img转成烧录器用镜像。

工具界面：

工具配置说明：

1、空白填充： EMMC选择 0x0

2、SPI FLASH：不要勾选

3、数据区预留：需要勾选

如果使用GPT分区的固件，制作镜像时parameter需要配置 DISKSIZE参数 ，具体参考文档《Rockchip量产烧录指南_v1.2》。

录器配置说明：

1、把data.bin烧录到EMMC的user分区

2、如果是RK3188/RKPX3,还需要把boot0.bin烧录到EMMC的boot1和boot2分区

3、烧录器配置全0的数据跳过不烧录

4、CSD值全部用默认值，不能修改

5、EXT CSD配置：

没有列出的项全部使用默认值，不能修改。
RK3188/RKPX3:
EXT_CSD[167] = 0x1f （如果EMMC颗粒支持，需要配置）
EXT_CSD[162] = 0x1 (启用 reset pin功能)
EXT_CSD[177] = 0x0 (默认值)
EXT_CSD[178] = 0x0 (默认值)
EXT_CSD[179] = 0x8 (0x8,从 boot1 启动)

其他AP：
EXT_CSD[167] = 0x1f （如果EMMC颗粒持，需要配置）
EXT_CSD[162] = 0x0 (默认值)
EXT_CSD[177] = 0x0 (默认值)
EXT_CSD[178] = 0x0 (默认值)
EXT_CSD[179] = 0x0 (默认值)

6.5 SLC Nand 镜像烧录

参考《Rockchip_Developer_Guide_Linux_Nand_Flash_Open_Source_Solution_CN.pdf》对应章节。

6.6 SPI Nand 镜像烧录

参考《Rockchip_Developer_Guide_Linux_Nand_Flash_Open_Source_Solution_CN.pdf》对应章节。

6.7 SPI Nor 镜像烧录

参考《Rockchip_Developer_Guide_Linux_Nand_Flash_Open_Source_Solution_CN.pdf》对应章节。

7. 存储软件驱动配置

RK 主要提供以下存储方案：

简称	主要支持的颗粒类型	主要支持文件系统	支的烧录方式
eMMC 方案	eMMC	FAT、EXT、SquashFS	USB 升级、SD 卡升级
rknand 方案	MLC、TLC Nand	FAT、EXT、SquashFS	USB 升级、SD 卡升级
rkflash 方案	SLC Nand、SPI Nand	FAT、EXT、SquashFS	USB 升级、SD 卡升级
rkflash 方案（SPI Nor 支持）	SPI Nor	SquashFS、JFFS2	USB 升级、SD 卡升级、烧录器升级
SLC Nand 开源方案	SLC Nand	UBIFS	USB 升级、SD 卡升级、烧录器升级
SPI Nand 开源方案	SPI Nand	UBIFS	USB 升级、SD 卡升级、烧录器升级
SPI Nor 开源方案	SPI Nor	SquashFS、JFFS2	USB 升级、SD 卡升级、烧录器升级

7.1 u-boot

详细参考《Rockchip-Developer-Guide-UBoot-nextdev-CN》 CH05 - 驱动模块 Storage 章节。

7.2 kernel

由于内核 4.4 及旧版本内核对于开源 SPI Flash 的支持不完善，所以内核中关于 flash 的开源方案与 uboot 下的实现方式有所不同：

简称	主要支持的颗粒类型	主控驱动	flash 框架	注册设备类型	主要支持文件系统	支持的烧录方式
rknand 方案	MLC TLC Nand	drivers/rkand	drivers/rkand	block 设备	FAT、EXT、SquashFS	USB 升级、SD 卡升级
rkflash 方案	SLC Nand、SPI Nand	drivers/rkflash	drivers/rkflash	block 设备	FAT、EXT、SquashFS	USB 升级、SD 卡升级
rkflash 方案（SPI Nor 支持）	SPI Nor	drivers/rkflash	drivers/rkflash	block 或 mtd 设备	SquashFS、JFFS2	USB 升级、SD 卡升级、烧录器升级
SLC Nand 开源方案	SLC Nand	drivers/mtd/nand/raw	drivers/mtd/nand/raw	mtd	UBIFS	USB 升级、SD 卡升级、烧录器升级
SPI Nand 开源方案	SPI Nand	drivers/rkflash	drivers/rkflash	mtd	UBIFS	USB 升级、SD 卡升级、烧录器升级

简称	主要支持的颗粒类型	主控驱动	flash 框架	注册设备类型	主要支持文件系统	支持的烧录方式
SPI Nor 开源方案	SPI Nor	drivers/rkflash	drivers/rkflash	mtd 或 mtd block 设备	SquashFS、JFFS2	USB 升级、SD 卡升级、烧录器升级

7.2.1 MLC Nand、TLC Nand rknand 方案

配置：

```
CONFIG_RK_NAND=y
```

驱动文件：

```
./drivers/rk_nand/
```

7.2.2 SLC Nand、SPI Nand 及 SPI Nor rkflash 方案

参考 《Rockchip_Developer_Guide_Linux_Flash_Open_Source_Solution_CN.md》 文档。

7.2.3 SLC Nand、SPI Nand 及 SPI Nor MTD 开源方案

参考 《Rockchip_Developer_Guide_Linux_Flash_Open_Source_Solution_CN.md》 文档。

7.3 各阶段存储 iomux/clk 配置情况及扫描次序

阶段	存储器件	是否配置 iomux	是否配置 clock	驱动配置
maskrom	扫描 nor、spinand、emmc、sdcard	配置（仅配置探测成功的器件）	配置	
spl	优先级1（支持 atags 方案）：maskrom 探测成功设备 优先级2：扫描 nor、spinand、emmc、sdcard	1.不配置（默认） 2.补充配置需要的 iommux	配置（驱动里配置）	dts/defconfig 根据具体 sdk 配置
uboot	优先级1（默认未开启）：CONFIG_ROCKCHIP_BOOTDEV 指定目标存储 优先级2（支持 atags 方案）：maskrom 探测成功设备 优先级3：扫描 nor、spinand、emmc、sdcard	不配置	配置（驱动里配置）	dts/defconfig 根据具体 sdk 配置
kernel	扫描 nor、spinand、emmc、sdcard			dts/defconfig 根据具体 sdk 配置

7.4 双存储方案扩展

参考 《Rockchip_Developer_Guide_Dual_Storage_CN.md》 文档。

8. 开源方案 OTA

参考 《Rockchip_Developer_Guide_Linux_Flash_Open_Source_Solution_CN.md》 文档。

9. 文件系统支持

9.1 UBIFS 文件系统

参考 《Rockchip_Developer_Guide_Linux_Flash_Open_Source_Solution_CN.md》 文档。

9.2 JFFS2 文件系统支持

参考 《Rockchip_Developer_Guide_Linux_Flash_Open_Source_Solution_CN.md》 文档。

10. Vendor Storage 使用说明

Vendor Storage 是设计来存放一些非安全小数据，比如 SN、MAC 等，详细参考文档：

- EMMC：《RK Vendor Storage Application Note》
- rkflash 方案的 flash 支持：《RK Vendor Storage Application Note》
- MTD 方案的 flash 支持：
  《Rockchip_Developer_Guide_Linux_Flash_Open_Source_Solution_CN.pdf》

10.1 Vendor Storage ID

Vendor Storage 是通过 ID（16bits）访问数据，不需要关心数据具体存放在分区的哪个位置，可以简单认为 ID 就是索引或者文件名。ID 0 -31 保留为通用 SDK 功能使用，客户自定义存储时请使用 32-65535.

下表为具体的 ID 功能定义：

ID	Function
0	reserved
1	SN
2	WIFI MAC
3	LAN MAC
4	BT MAC
5	HDCP 1.4 HDMI
6	HDCP 1.4 DP
7	HDCP 2.X
8	DRM KEY
9	PLAYREADY Cert
10	ATTENTION KEY
11	PLAYREADY ROOT KEY 0
12	PLAYREADY ROOT KEY 1
13	SENSOR CALIBRATION
14	RK reserved for future use
15	IMEI
16	LAN_RGMII_DL
17 – 31	RK reserved for future use
32 - 65535	Vendor use

10.2 Vendor Storage API

10.2.1 Uboot API

```
int vendor_storage_init (void)
    function: Initialize vendor storage
    input: none
    return: 0, Initialize success
            other, Initialize fail
```

```
int vendor_storage_read (u32 id, void *pbuf, u32 size)
    function: read vendor storage by id
    input: id, item id; pbuf, data buffer; size, number byte to read.
    return: -1, read fail.
            other: number byte have read.
```

```
int rk_vendor_write (u32 id, void *pbuf, u32 size)
    function: write vendor storage by id
    input: id, item id; pbuf: data buffer; size: number bytes to write.
    return: 0: write success
            other : write fail
```

10.2.2 kernel API

Source code : kernel/drivers/soc/rockchip/rk_vendor_storage.c

Include header：include/linux/soc/rockchip/rk_vendor_storage.h

```
int vendor_storage_init (void)
    function: Initialize vendor storage
    input: none
    return:  0, Initialize success
            other, Initialize fail
```

```
int vendor_storage_read (u32 id, void *pbuf, u32 size)
    function: read vendor storage by id
    input: id, item id; pbuf, data buffer; size, number byte to read.
    return: -1, read fail.
            other: number byte have read.
```

```
int rk_vendor_write (u32 id, void *pbuf, u32 size)
    function: write vendor storage by id
    input: id, item id; pbuf: data buffer; size: number bytes to write.
    return: 0: write success
            other : write fail
```

10.2.3 User API

用户应用是通过 IOCTL 接口访问 vendor storage，下面是读写的参考代码。

```
#include <fcntl.h>
#include <sys/ioctl.h>

#define VENDOR_REQ_TAG  0x56524551
#define VENDOR_READ_IO  _IOW ('v', 0x01, unsigned int)
#define VENDOR_WRITE_IO _IOW ('v', 0x02, unsigned int)
#define VENDOR_SN_ID        1
#define VENDOR_WIFI_MAC_ID  2
#define VENDOR_LAN_MAC_ID   3
```

```
#define VENDOR_BLUETOOTH_ID 4

struct rk_vendor_req {
    u32 tag;
    u16 id;
    u16 len;
    u8 data [1];
};

static void print_hex_data (uint8 *s, uint32 *buf, uint32 len)
{
    uint32 i, j, count;

    ERROR ("% s", s);
    for (i = 0; i < len; i += 4)
        ERROR ("% x % x % x % x", buf [i], buf [i + 1], buf [i + 2], buf [i +
3]);
}

int vendor_storage_read_test (void)
{
    u32 i;
    int ret, sys_fd;
    u8 p_buf [2048];/* malloc req buffer or used extern buffer */
    struct rk_vendor_req *req;

    req = (struct rk_vendor_req *) p_buf;
    sys_fd = open ("/dev/vendor_storage", O_RDWR, 0);
    if (sys_fd < 0){
        ERROR ("vendor_storage open fail\n");
        return -1;
    }

    req->tag = VENDOR_REQ_TAG;
    req->id = VENDOR_SN_ID;
    req->len = 512; /* max read length to read*/
    ret = ioctl (sys_fd, VENDOR_READ_IO, req);
    print_hex_data ("vendor read:", (uint32*) req, req->len + 8);
/* return req->len is the real data length stored in the NV-storage */
    if (ret){
        ERROR ("vendor read error\n");
        return -1;
    }

    return 0;
}

int vendor_storage_write_test (void)
{
    uint32 i;
    int ret, sys_fd;
    uint8 p_buf [2048]; /* malloc req buffer or used extern buffer */
    struct rk_vendor_req *req;

    req = (struct rk_vendor_req *) p_buf;
    sys_fd = open ("/dev/vendor_storage", O_RDWR, 0);
```

```
    if (sys_fd < 0){
        ERROR ("vendor_storage open fail\n");
        return -1;
    }

    req->tag = VENDOR_REQ_TAG;
    req->id = VENDOR_SN_ID;
    req->len = 32; /* data len */
    for (i = 0; i < 32; i++)
        req->data [i] = i;
    print_hex_data ("vendor write:", (uint32*) req, req->len + 8);
    ret = ioctl (sys_fd, VENDOR_WRITE_IO, req);
    if (ret){
        ERROR ("vendor write error\n");
        return -1;
    }

    return 0;
}
```

10.2.4 PC Tool API

PC 工具有提供参考工程源码，由 C++ 开发，这里列出读写的两个 API 接口。

```
int RK_ReadProvisioningData (int id, (PBYTE) pbuf, int size)
    function: read vendor storage by id
    input: id, item id; pbuf, data buffer; size, number byte to read.
    return: 0, read data okay.
            other: read fail.
```

```
int  RK_WriteProvisioningData (int id, (PBYTE) pbuf, int size)
    function: write vendor storage by id
    input: id, item id; pbuf: data buffer; size: number bytes to write.
    return:  0: write success
            other : write fail
```

10.3 使用注意事项

10.3.1 VENDOR 分区单个 item 最大支持数据量

Nand 和 EMMC Vendor 分区共 64KB，Nor 为 4KB，存放在 vendor 结构体里：

```
struct vendor_info {
    struct vendor_hdr *hdr; //32byte
    struct vendor_item *item; //8byte * item
    u8 *data; //size = sum (item 1, item 2, ... item n)
    u32 *hash;
    u32 *version2;
};
```

所以如果只写 1 个 item：

1. Nand 和 EMMC 单个 item data size 可达 64 * 1024 - 32 - 8 -4 -4 = 65488 bytes
2. Nor 单个 item data size 可达 4 * 1024 - 32 -8 -4 -4= 4048 bytes

10.3.2 VENDOR 数据双备份支持

VENDOR 数据默认支持双备份写入，所以：

- 如果写入第一份时掉电，则回退用旧的数据（不可避免）
- 如果写入第二份时掉电，则使用第一份

11. 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