# Rockchip Application Notes Storage

ID: RK-SM-YF-017

Release Version: V2.1.0

Release Date: 2024-01-15

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

**DISCLAIMER**



**Trademark Statement**



Rockchip Electronics Co., Ltd.

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

Website: www.rock-chips.com

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

Customer service Fax: +86-591-83951833

Customer service e-Mail: fae@rock-chips.com

**Preface**

**Overview**

This article mainly guides readers to understand the boot process and configure and debug storage.

For more detailed content, please refer to the following documents:

| No. | Document Name | Content Overview |
| --- | --- | --- |
| 1 | "Rockchip_Introduction_Partition" | Partition configuration introduction |
| 2 | "Rockchip-Developer-Guide-UBoot-nextdev-CN" | Uboot development documentation |
| 3 | "RK Vendor Storage Application Note" | Vendor Stroage application note |
| 4 | "Rockchip Mass Production Burning Guide_v1.2" | Mass production burning guide |
| 5 | "Rockchip_Developer_Guide_Linux_Flash_Open_Source_Solution_CN" | Flash open source storage solution development document |
| 6 | "Rockchip_Developer_Guide_Dual_Storage_CN" | Dual storage development document |

**Product Version**

| Chipset | Kernel Version |
| --- | --- |
| ALL SOC | \ |

**Intended Audience**

This document (this guide) is mainly intended for:

Technical support engineers

Software development engineers

**Revision History**

| Version | Author | Date | Change Description |
| --- | --- | --- | --- |
| V2.0.0 | Jon Lin | 2024-01-08 | Initial version |
| V2.1.0 | Jon Lin | 2024-01-15 | Add content to the introduction chapter of Flash |

**Contents**

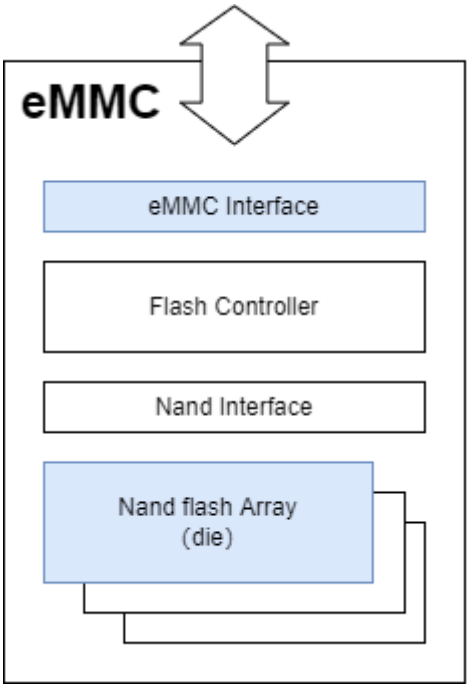
# 1. Naming Conventions

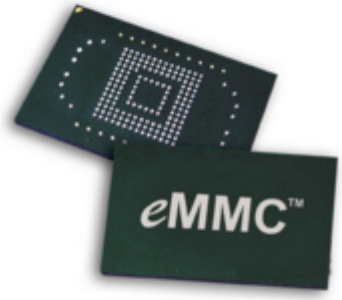
| Naming | Introduction |
| --- | --- |
| SPI Nand | SPI protocol Nand, mostly SLC Nand |
| SPI Nor | SPI protocol Nor |
| PP Nand | Parallel peripherals Nand, parallel Nand, SLC\MLC\TLC Nand |
| Flash | SPI Nand、SPI Nor、PP Nand collectively referred to as |
| Nand | SPI Nand and PP Nand and other Nand particles collectively referred to as |
| Octal SPI DTR Nor flash | Octal SPI Nor flash with dual sampling |
| OCTA flash | Octal SPI Flash, including Octal SPI Nor、Octal SPI Nand |

# 2. Flash Introduction

## 2.1 Flash Storage Types

Flash memory, in a broad sense, refers to all non-volatile storage technologies based on Nand flash and Nor flash. For example, the commonly used EMMC and SPI flash on our RK devices, or the PCIe SSD, SATA SSD, and UFS that have gradually entered our field of vision in recent years. You may be curious about the differences between these storages. Here is a brief introduction.



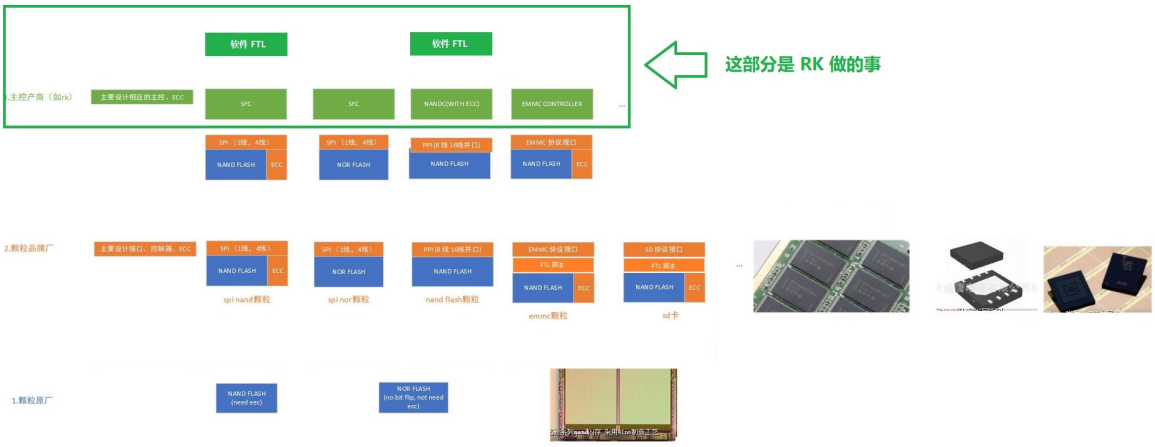

Flash die are the ultimate carrier for storing user data, such as the Nand Flash Array shown in the screenshot. The devices manufacturers produce chips of different capacity sizes and then combine them with different controllers to form specific storage devices for various industries. For example, EMMC devices are composed of Nand flash die encapsulated EMMC interfaces, and SPI Nor devices are composed of Nor flash die encapsulated SPI interfaces. Of course, these storage particles usually have firmware running inside, also known as Firmware firmware, just like our chips. The following are the corresponding explanations for some storage particles:

| Device | Device Introduction | Host Controller | Special Drivers on Host Side |
| --- | --- | --- | --- |
| SPI Nor | Nor + SPI Interface | FSPI/SPI Controller | A relatively simple file system that supports wear leveling and out-of-place updates is sufficient. |
| PP Nand | Nand + Parallel Port | NandC Controller | FTL algorithm or file system with FTL |
| SPI Nand | Nand + SPI Interface | FSPI/SPI Controller | FTL algorithm or file system with FTL |
| eMMC | Nand + eMMC Interface + FTL algorithm | SDMMC/SDHCI Controller | MMC protocol framework |
| SD Card | Nand + SDIO Interface+ FTL algorithm | SDMMC Controller | MMC protocol framework |
| SATA SSD | Nand + Sata Interface+ FTL algorithm | Sata Controller | SATA protocol framework |
| NVMe SSD | Nand + PCIe Interface+ FTL algorithm | PCIe Controller | NVMe protocol framework |
| UFS | Nand + UFS Interface+ FTL algorithm | UFS Controller | UFS protocol framework |
| ... | | | |

Note:

- Flash devices are essentially iterating in the direction of faster interface controllers, higher internal parallel complexity, and larger capacities.

- In addition to supporting the above common storage, RK products also support SD Nand (surface-mount packaging).
- The devices composed of Nand flash need to store the FTL algorithm to manage Nand, and the reason is explained in the "Basic Principles of Nand" chapter
- PP Nand/SPI Nand, as the device itself does not integrate FTL algorithm, the host software should include FTL algorithm or file system with FTL algorithm support
- Explanation of combination of die manufacturer, brand manufacturer, and RK main control:




- Due to the dependence of MLC/TLC Nand on FTL algorithm, the impact of NandC controller on SOC area, and the disadvantages of multiple pins of PP Nand, RK has gradually eliminated the PP Nand interface in recent years. Therefore, the current Flash products of RK mainly include SPI Flash products.

## 2.2 Flash Selection

**SLC Nand, SPI Nand, SPI Nor**

Refer to the GigaDevice selection reference to understand the range of Flash material selection. [GD Chip Selection Reference Link]

## 2.3 Simple Comparison of Flash

| torage Type | Stability | Price | Package | Speed | Selectable Capacity |
|---|---|---|---|---|---|
| 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 | 166M SDR 104M DDR | <= 128MB |
| Octal SPI | | | 8x6 24-BALL | 120M | |

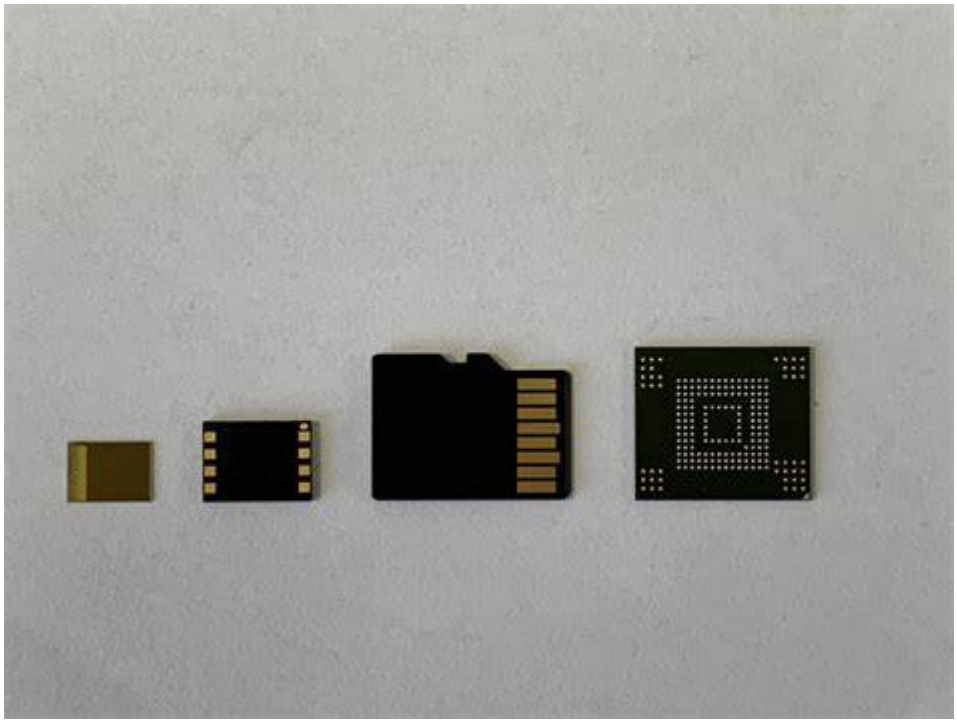
| | | | | | |
|---|---|---|---|---|---|
| Nand<br>**torage Type** | Normal<br>**Stability** | High<br>**Price** | TFBGA<br>**Package** | DDR<br>**Speed** | 128MB~512MB<br>**Selectable Capacity** |
| Octal SPI Nor | Good | High | 8x6 24-BALL TFBGA | 200M DDR | <= 128MB |

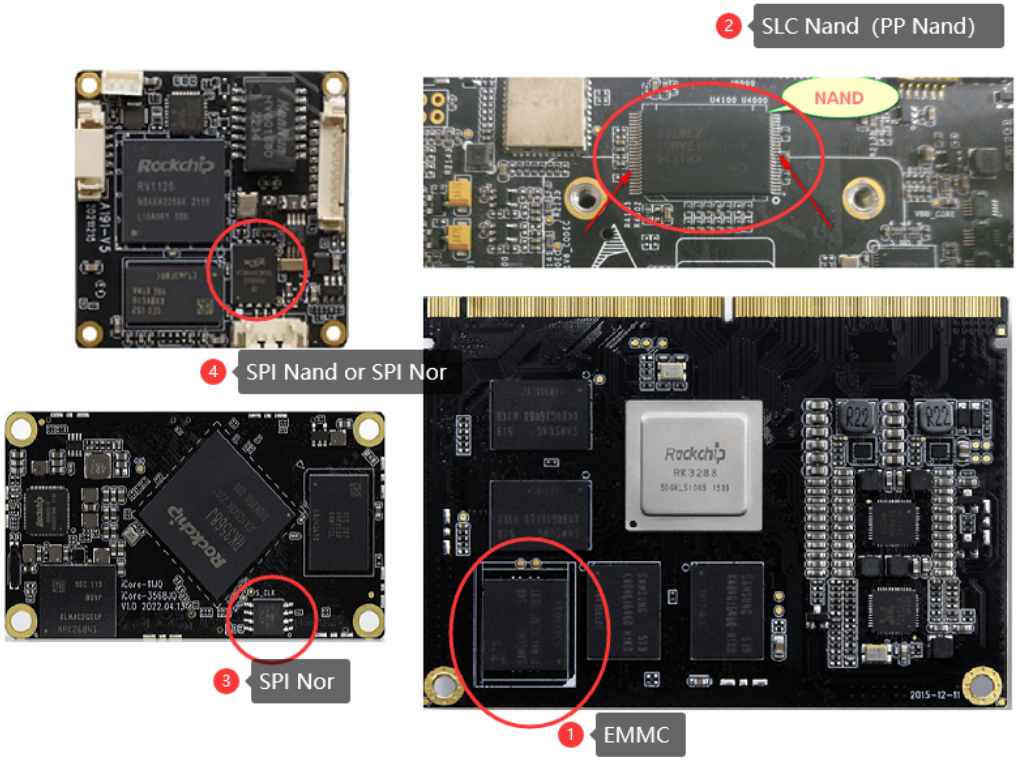
Note:

- EMMC has advantages such as large capacity, high stability, and fast read and write rates, but it has disadvantages such as high unit price, large packaging, and high initialization time overhead (usually reaching the level of one hundred milliseconds)
- SPI Nand has optimizations such as small packaging, low single-chip price, and larger capacity compared to Norflash. However, due to limitations such as ECC and storage algorithm management, the read and write speed and stability of Nand are relatively average
- SPI Nor has the advantages of small package size, high stability, low single-chip price, and low software initialization time overhead. Especially in small firmware scenarios, SPI Nor has good loading speed, but its continuous data transmission speed is not as fast as EMMC, and when the capacity increases, the unit price is relatively expensive
- Due to the dependence of MLC/TLC PP Nand on FTL algorithm, the large chip area occupied by NandC controllers, and the large number of pins, RK has phased out the PP Nand interface in recent years. Therefore, small capacity storage mainly refers to SPI Flash, and MLC/TLC PP Nand is not included in the above table
- The price is based on the calculation of a single chip, and the price per MB decreases as the capacity increases.

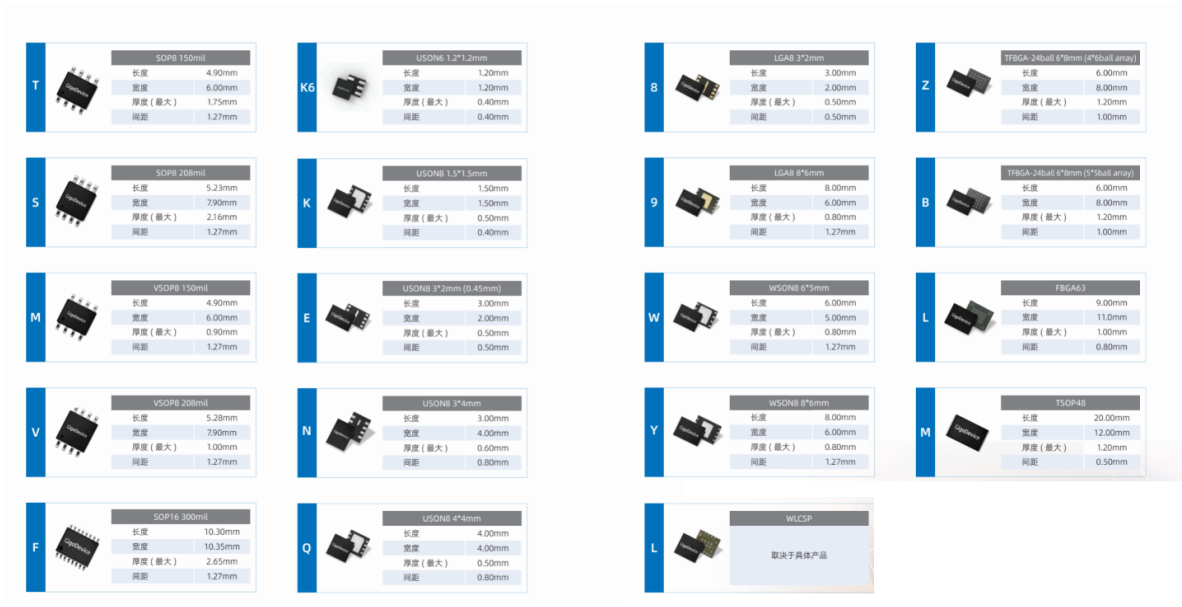
## 2.4 Common Flash Packages

Comparison of several packaging options:






SLC Nand/SPI flash：




Introduction to Packaging

## 2.5 Flash Prices

| Storage Type | Capacity | Price (USD) |
|---|---|---|
| SPINand | 1Gbits QPI Comsummer | 0.53 |
| SPINand | 2Gbits QPI Comsummer | 0.75 |
| SPINand | 4Gbits QPI Comsummer | 1.7 |
| SPINand (Continuous read mode, single material) | compare to the same capacity devices | +30% |
| 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 |

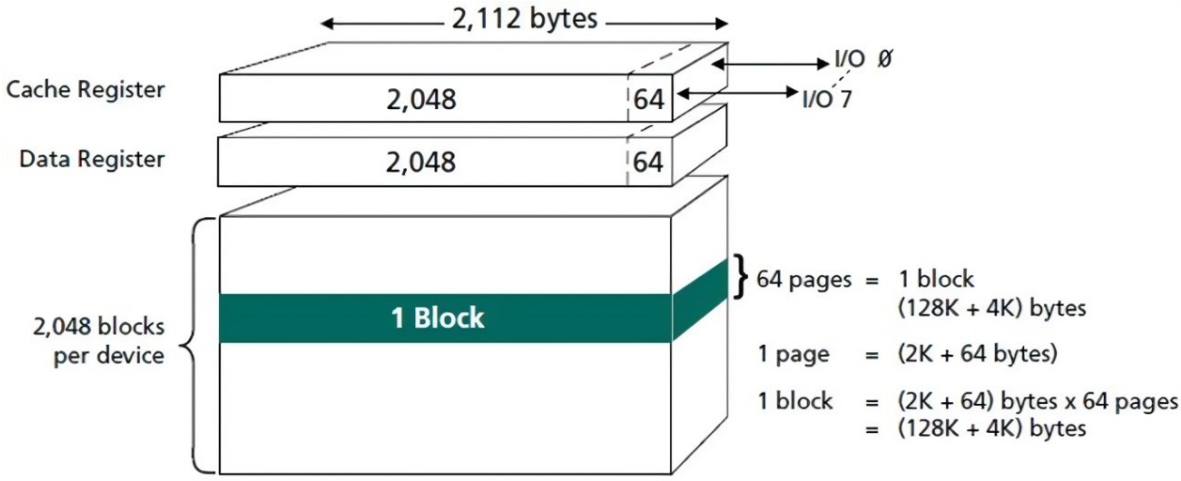
Note:

- The above prices are market quotes from specific manufacturers and do not include any markup for customer communication.
- The above prices mainly reflect the price differences between different storage devices and the trend of prices increasing with capacity for the same type of storage device.

# 2.6 Basic Principles of Nand

**Nand Physical Structure**

Taking the common structure of SLC Nand as an example:

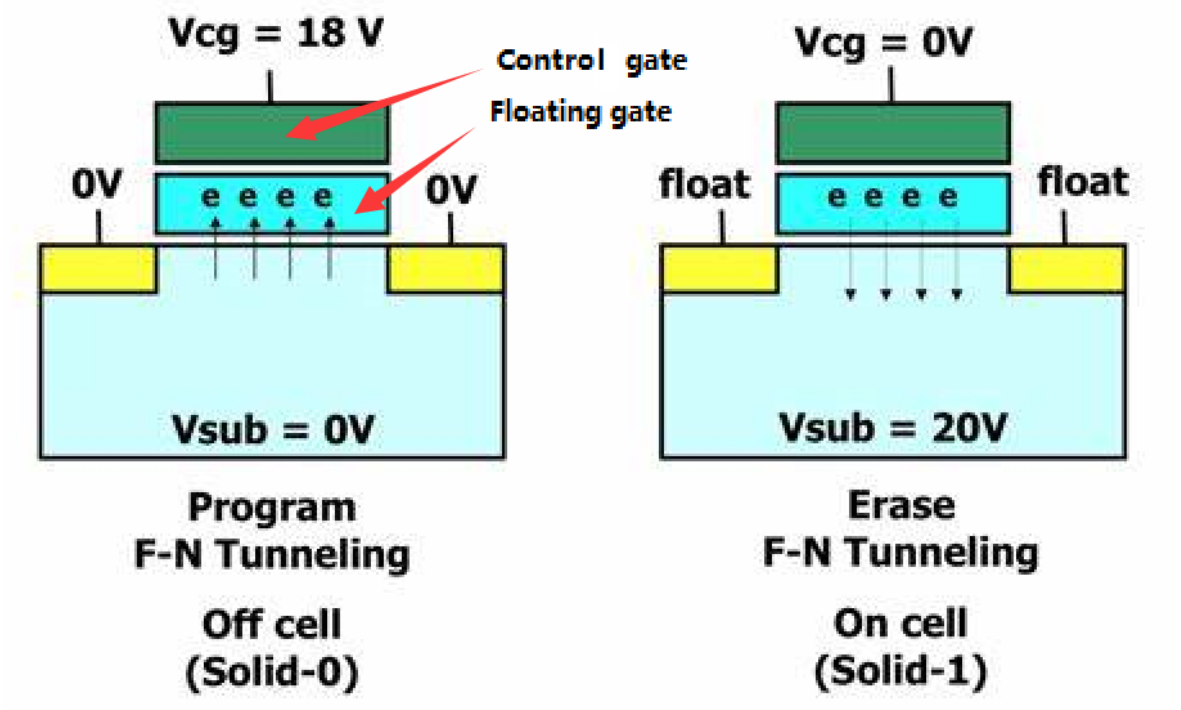



Note：

- SLC Nand is composed of multiple flash blocks, typically reaching 1024, 2048, or 4096 blocks. The screenshot above shows 2048 blocks
- Block is the smallest erasing unit of Nand flash, and the SLC Nand block size is usually 128KB or 256KB, which changes from logical 0 to 1 after erasing
- A block typically consists of 64 pages, with some particles reaching up to 128 pages
- The page is the basic unit for Nand flash programming and reading, and the SLC Nand page size is usually 2KB or 4KB, which changes from logic 1 to 0 after programming

**Logically, 0 or 1 is actually the level value on each physical unit of the Bit cell**

Take the SLC NAND implemented by floating gate as an example:

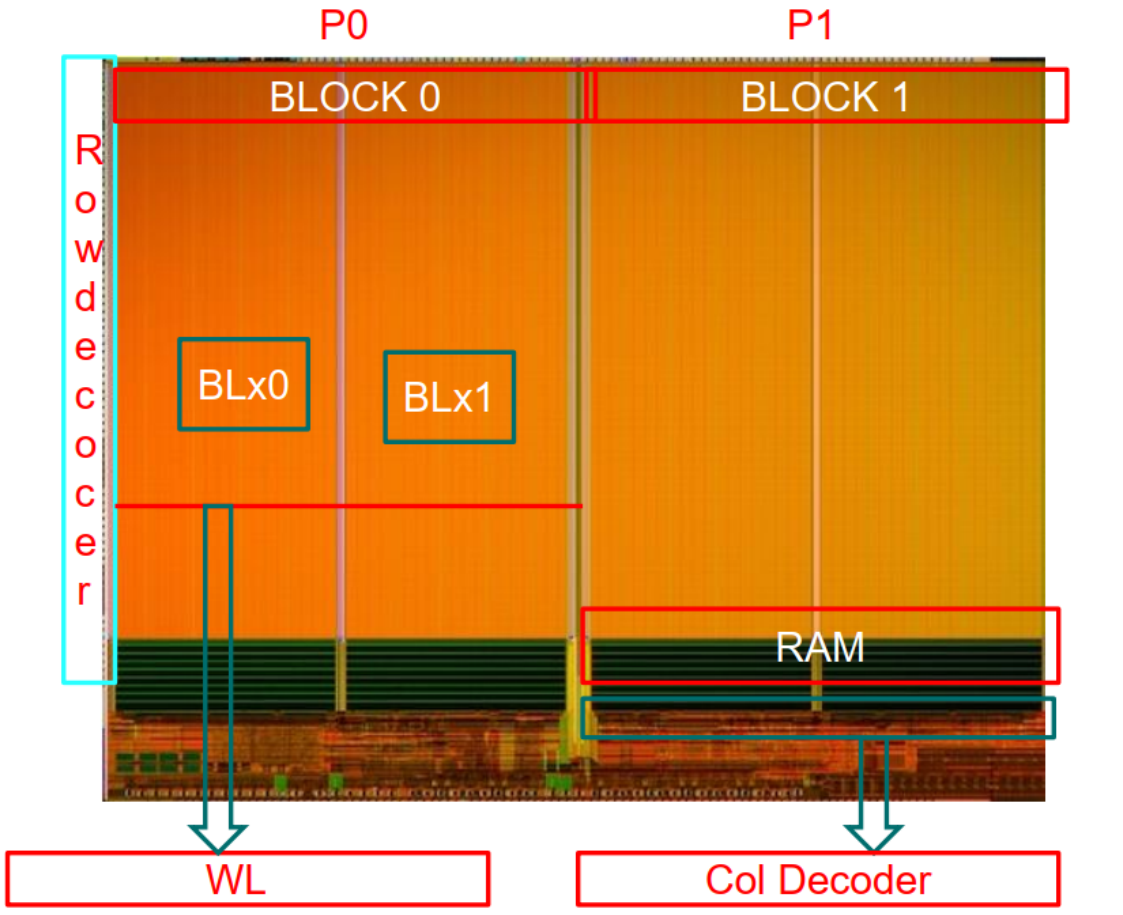


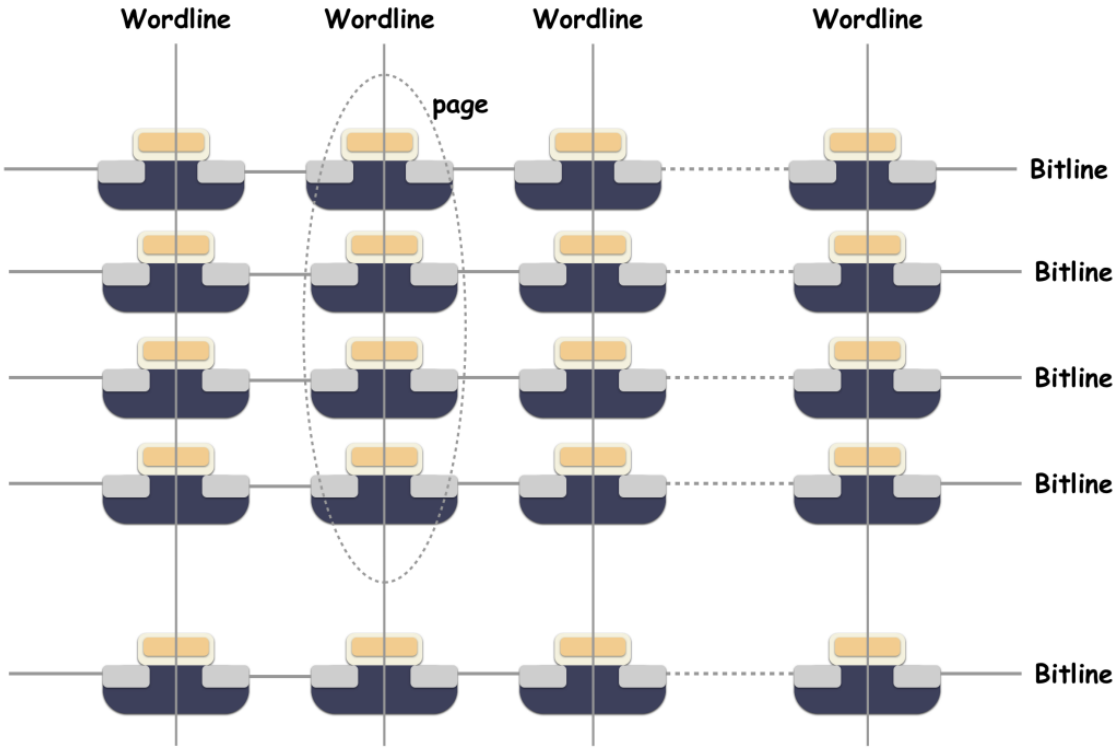
The actual unit that stores data in Flash is the internal floating gate (Floating gate). By controlling the voltage applied to the control gate and the substrate, it determines whether to charge or discharge the floating gate. The logical state of 0 or 1 on the data is represented by whether the stored charge exceeds a specific threshold Vth:

- For writing to NAND Flash, it involves charging the floating gate by applying high voltage to the control gate. When the voltage exceeds the threshold Vth, it represents a 0.
    - Data Retention issue: When charged at high voltage, the charge remains stored in the floating gate without any erasing action. However, due to the electric field between the floating gate and the substrate, the charge gradually leaks over time.
    - P/E cycle issue: As the number of program/erase cycles (P/E Cycles) increases, the oxide layer between the floating gate and the substrate ages, which also affects the ability of the floating gate to store charge.
- For erasing in NAND Flash, it involves discharging the floating gate by applying high voltage to the substrate. When the voltage is below the threshold Vth, it represents a 1.

**Design of Wordline and Bitline**

In order to efficiently complete the charging and discharging actions of thousands of Nand flash cells, the designer has implemented auxiliary circuit structures such as Bitline/Wireline. For example, multiple cells are connected in series, and multiple sets of cells form a Nand string. The ends of a Nand string are called Bit Line and Source line. Each cell's control gate is connected by a single Wordline. Cells controlled by the same Wordline constitute a logical page (Page). Together, they form a block. All cells within a block share the same substrate, so the minimum erase unit is a block.



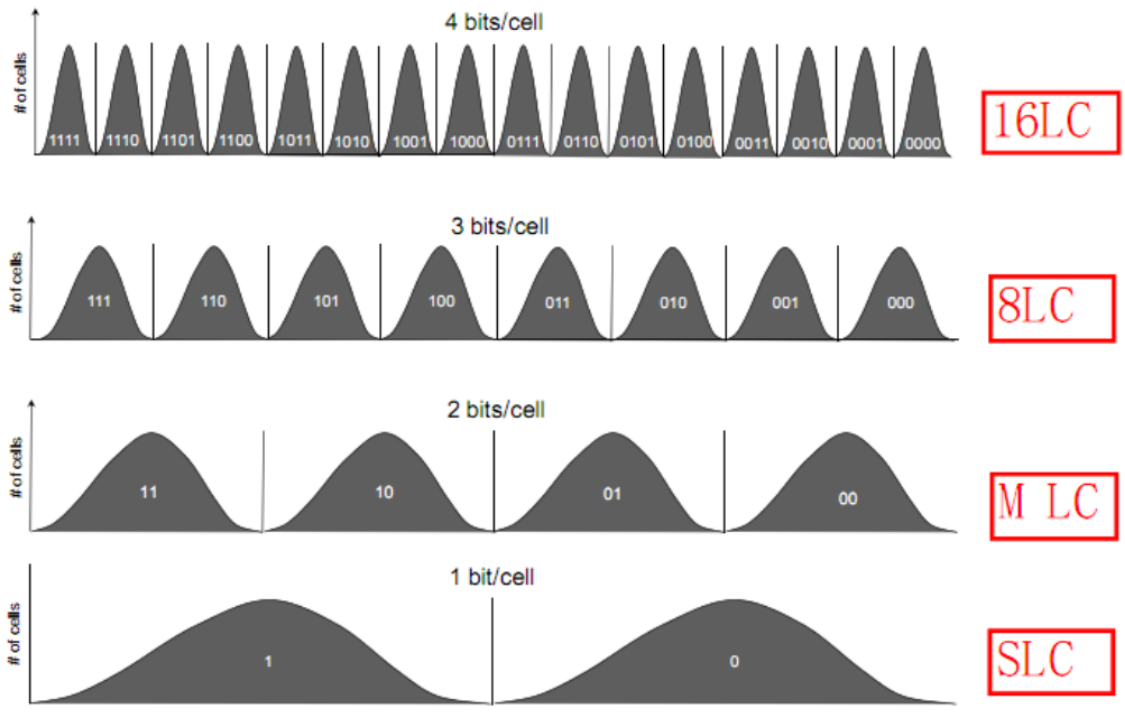



Bitline 图片来源

**Abort SLC\MLC\TLC\QLC：**

As mentioned above, Bitine can be used to control the charging and discharging behavior, determining the level state of the bit cell. In fact, by controlling the amount of charging, different level performances of a single bit cell can be achieved:

- SLC, Single Level cell, represents 1 bits data with only 0/1 two levels of logic.
- MLC, supports 2 bits data with 4 levels of states from 0 to 3.
- TLC, supports 3 bits data with 8 levels of states from 0 to 7.
- QLC, supports 4 bits data with 16 levels of states from 0 to 15.




—— Image sourced from the internet

**Bit cell charge storage capacity failure with bit flip**

Due to the specific physical structure of Nand, there is a possibility that the charging and discharging process of Nand particles may be unstable, and the physical structure may age due to friction and wear, which may affect the energy storage capacity. This can lead to changes in logical levels, such as bit flip. At the user level, data errors, firmware anomalies, and file system crashes may occur. Therefore, Nand flash products will introduce ECC error correction algorithms with different capabilities to support, in order to extend the lifespan of particles and enhance the robustness of Nand products.

**Characteristics of Nand Products**

The physical implementation principle of Nand determines that Nand products have the following characteristics:

- Out-of-place Update. Nand Flash programming can only change the storage unit from 1 to 0, so an erase operation is needed before reprogramming. Moreover, programming is done in pages while erasing is done in blocks (a block includes multiple pages). If in-place update is used, it means repeatedly updating the same logical address to the same location. In this case, every update requires an erase operation first. Since erasing operations consume time and damage Flash, FTL generally uses out-of-place update to map updated data to a new location.
- Limited P/E cycles. As mentioned earlier, each block of Nand Flash has a limit on the number of erase cycles. After a certain number of erase cycles, this block becomes unstable and the data programmed into it may be prone to errors or even fail to erase.
- Better performance. Unlike traditional mechanical hard drives, Flash storage does not have mechanical devices, such as seeking. The access overhead for all addresses is the same, especially in random read performance, where SSD is much better than traditional mechanical hard drives. According to this reasoning, the speed of Flash device random access and sequential access should be the same. However, in reality, Flash supports Cache operations, which can preemptively read the next page's data into internal registers during

## 2.7 Basic Principles of Nor

The principle is similar to Nand, with the following main differences:

- The implementation principle of bit cell is similar, but each cell is directly connected to the Bit Line and Source line at both ends, making it more stable. Therefore, Nor usually does not require ECC mechanism to prevent physical aging (however, there is still a tiny probability of flipping anomalies). However, the circuit structure around the cell is more complex, resulting in higher cost per die for the same capacity.
- Block/page size is different from Nand.
    - Generally supports two formats of erasable blocks - Block/Sector in a broad sense, which are 64KB and 4KB respectively.
    - Generally supports basic programming (writing) unit page as 256B.

## 2.8 Nand Storage ECC Dependency

| Chip Type | Main Control\ECC Support | ECC Selection Dependent |
| --- | --- | --- |
| EMMC | No Need | EMMC devices |
| SPI Nand | Without ECC | SPI Nand devices |
| SLC Nand | Nand V6 16 bits per 1KB | NandC Control |
| MLC TLC Nand | Nand V9 16 bits and more | NandC Control |

Note：

- Some particles in SLC Nand come with ECC, while the rest of PP Nand do not integrate ECC and rely on the ECC provided by the controller itself

## 2.9 Original Bad Blocks in Nand

Due to physical characteristics, a certain proportion of original bad blocks are allowed in the manufacturing process of Nand Flash. At the time of shipment, the manufacturer will set the original bad block mark in a specific area of the flash, usually in the spare first byte (or referred to as the OOB area). Typically, the original bad block mark cannot be destroyed.

- W25N01GV SPI Nand is an exception.

## 2.10 Lifespan and ECC Errors in Nand

Nand flash has durability, which is typically characterized by P/E (erase/write). After reaching a certain number of P/E cycles, failure may occur. In actual tests, abnormalities may occur in some particles even at 30K times (the actual value may differ from the manufacturer's claim, contact the manufacturer for details).

When approaching the lifespan of Nand flash, there are usually the following warnings:

- The read data bit flip reaches a situation that requires refreshing the data.
    - The data is valid and can be used normally.

When reaching or exceeding the lifespan of Nand flash, the following abnormalities may occur:

- The read data bit flip reaches a situation that requires refreshing the data.
    - The data is valid and can be used normally.
- Reading data reports ECC fail.
    - The data is invalid, and there is a retry mechanism at the bottom, so it may cause high CPU usage of threads.
    - Some data is lost, which may lead to upper layer exceptions.
- Erase/write fail.
    - Marked as a bad block, does not affect use, and data will not be lost.

## 2.11 Technical Key Points of Nand FTL

- Address mapping management. Flash memory is a black box to the outside world, integrating Nand Flash and FTL, etc. Upper-layer applications use logical addresses for access. FTL maps logical addresses to

different physical addresses, managing the physical location where the latest data of each logical address is stored.

- Garbage collection. As data is written, some parts of the data in some blocks of the flash memory have become invalid, and it is necessary to move the valid data from the block and then erase it to receive new data.
- Wear leveling and bad block management. Because the P/E cycle of each block is limited, some blocks may be damaged due to repeated use, while some blocks are rarely accessed and have not been operated. To avoid this situation, FTL adds wear leveling functionality, which is generally achieved by controlling garbage collection and empty block pool management, thereby balancing the usage frequency of each block. The ideal situation is that all blocks reach the wear threshold together. Since there are some bad blocks in Flash itself, some blocks will become unstable during use. Therefore, when managing, FTL needs to avoid these useless blocks and copy the data on the unstable blocks that become unstable in time to stable locations.

## 2.12 Evolution of RK Nand Storage Solutions (including PP Nand and SPI Nand)

- Early only supported closed-source FTL solutions from RK
  - Mainly including RK3326\RK3308\RV1108
- Customers began to have UBIFS, MTD, and burner burning requirements, requiring the use of MTD storage drive framework
  - Starting from RK3308 chip, because this solution is open source from driver to algorithm to file system, it is usually called MTD open source solution internally compared with closed-source solutions
  - RK3308 provides low-level drivers, and the upper connection is completed by customers (with strong development capabilities) themselves
  - RK3308 products have provided SDK configuration for PP SLC Nand MTD open source solution
- Completely switched to MTD open source solution and provided more detailed support and guidance:
  - RV1126\RK3568 and subsequent chips

## 2.13 Flash Host Controller

### 2.13.1 SFC Controller

The Serial Flash Controller (SFC) is used to control the data transfer between the chip system and serial nor/nand flash memory devices.

The SFC supports the following features:

- Supports SPI Nor, SPI Nand
- Supports SPI Nor 1-wire, 2-wire, and 4-wire transmission
- DMA transmission

### 2.13.2 FSPI Controller

FSPI (Flexible Serial Peripheral Interface) is a flexible serial transmission controller. It is a new design of SFC, considering changes in supported devices, so it is renamed as FSPI. It has the following main features:

- Supports SPI Nor, SPI Nand, PSRAM, and SRAM under SPI protocol
- Supports SPI Nor 1-wire, 2-wire, and 4-wire transmission， Version 8 and later support 8-line DDR transmission
- XIP technology
- DMA transmission

### 2.13.3 NandC Controller

NandC is the main controller used to complete the data transfer between Nand flash and the main chip. It supports direct data transmission through the AHB bus master. To adapt to different application scenarios, RK currently has two versions of NandC: the highly integrated NandC V9 and the simplified NandC V6 with smaller chip area.

Generally, a chip selects the NandC version based on its market position. NandC V6 only supports SLC Nand and is usually placed in products with small storage capacity (usually SLC Nand is less than 512MB). NandC V9 can support MLC and TLC, so it can be applied in products with large storage capacity.

### 2.13.4 General SPI Interface

While FSPI is a dedicated SPI Flash interface, RK SOC usually has multiple general SPI interfaces. This interface also supports external SPI Flash devices, but generally these interface devices cannot be used as bootdev.

## 2.14 SPI Flash Output Latency Statistics

Summary:

| | **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) |

Notes:

- The specific particles are subject to the manual, and the table is for reference only.

# 3. Particle Verification

## 3.1 Overview of SLC Nand/SPI Nand/SPI Nor Verification Content

**Verification Explanation:**

Small-capacity particles are generally more stable and have better compatibility with the main controller. Therefore, most of them only undergo functional verification, and their reliability and stability mainly depend on the extensive tests and reports conducted by the original manufacturer.

1. Functional Verification
2. Basic Product Lifecycle Reliability Verification
3. Software Compatibility and Stability Verification

| Storage Type | Functional Verification | Particle Reliability Verification | Software Compatibility and Stability Verification |
|---|---|---|---|
| SPI Nor | Y | N*1 | N |
| SPI Nand | Y | Y | TBD*2 |
| SLC Nand | Y | N*3 | TBD*2 |

Notes:

1. SPI Nor only undergoes functional verification, mainly considering its relative stability, good compatibility, and the fact that reliability and stability verification need to reach a large number of tests to test the boundary conditions. The reliability and stability verification are guaranteed by the particle's original manufacturer.
2. TBD: Only the storage particles with significant compatibility changes and the storage driver version updates are subject to corresponding tests. Generally, this test is not performed.
3. The ECC part of SLC Nand is mainly provided by the RK Nand main control, which can correct errors up to 16 bits/1KB, resulting in high redundancy and stability. No further verification of its reliability is required, and the reliability and stability verification are guaranteed by the particle's original manufacturer.

**Verification Methods:**

1. Functional Verification

- Main control compatibility: The system can boot up normally, basic functions are normal, and flash 10 loop stress testing passes.

2. Particle Reliability Verification

Retention Test:

- After a certain proportion of P/E cycle processing on the flash, bake it to simulate the product lifespan.

3. Software Compatibility and Stability Abnormal Power-off Verification (recommended for actual products)

Abnormal Power-off Test:

- Power off once every minute for 1 minute.

- Boot into the Linux system during power-on.
- Continuously perform dd commands during power-on without comparison, mainly testing FTL data migration performance.
- Continue for 7 days (meeting the basic requirements of most products, approximately 10K reads and writes, around 10,000 power-offs).

## 3.2 RK Flash Sample Submission Requirements

### 3.2.1 Verification-related Information

General Information:

1. Small-capacity storage includes SLC PP Nand, SPI Nand, and SPI Flash.
2. Prioritize submitting the main selling particles.
3. This channel is for verification purposes only; no business-related suggestions can be provided at this time.
4. The support list for small-capacity storage is typically updated every 1-2 months, but actual particle verification times may vary and will be completed before the support list is updated.
5. At least 10 pieces should be submitted, along with the corresponding particle manual.
6. Prioritize verifying particles that are urgently needed by customers.

Special Notes for SPI Nand:

1. Particles must have an ECC module; otherwise, they are not supported as the RK main control does not have an integrated ECC module.
2. There should be no continuous bad blocks at the end of the Flash; otherwise, it will destroy the information in the bad block table at the end, and if it cannot be resolved, it cannot be supported.
3. Some particles have a 2-plane structure and require plane select bits to be set on the address to select odd plane data. However, there is compatibility issues with these particles on the RK platform, and the driver needs to handle compatibility; therefore, it is not recommended to use them.

Special Notes for SPI Nor:

1. SPI Nor particles require a tRST of less than 200us.

### 3.2.2 Verification Process

- The internal team will perform functional testing and stress testing to verify the compatibility between the particles and the main control. However, the stability of the particles and their performance during stress tests should be determined by the original manufacturer's specifications.
- Verification will only be conducted on classic platforms. The small-capacity storage main control IP of RK is a compatible IP, so only RK3568 will be verified, but it can be compatible with most SOCs that have corresponding control devices. For example, if SPI Nand is verified as OK on RK3568, it will also be compatible with RV1126 and RK3308.

### 3.2.3 Verification Mailing Address

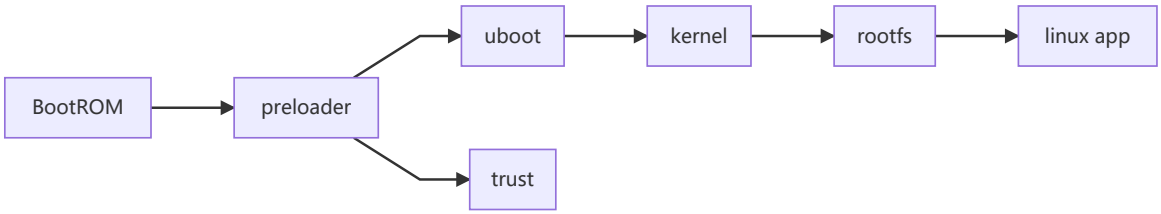
Submit a request on the RK Redmine issue tracking platform, and the storage module software engineer will provide the address.

### 3.2.4 Customer Patches Distribution

- Due to new flash materials, especially SPI Flash materials, source code patches are usually required after verification. If customers have driver requirements, please submit a request on the RK Redmine issue feedback platform, and patch packages will be provided online at that time.

# 4. Device Bootup Process

The bootup process refers to the software flow from system power-on to system startup completion. The following is the Linux system bootup process:




## 4.1 RK SOC BOOTROM Boot Support Status

| SOC | 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 | Y | 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 |
| RK3228 | Y | Y | Y*1 | Y | Y |
| RK3288 | Y | Y | Y | Y | Y |

| SOC | Emmc Boot | Nand Boot | SPI NAND Boot | SD Boot | SPI NOR Boot |
|---|---|---|---|---|---|
| RK3308 | Y | Y | Y | Y | Y |
| RK3326/PX30 | Y | Y | Y*1 | Y | Y |
| RK3328 | N | Y | Y*1 | Y | Y |
| RK3368/PX5 | Y | Y | Y*1 | Y | Y*1 |
| RK3399 | N | Y | Y | 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: The chip hardware supports it, but the SDK release development package does not support it.

## 4.2 RK SOC Storage Interface Specifications

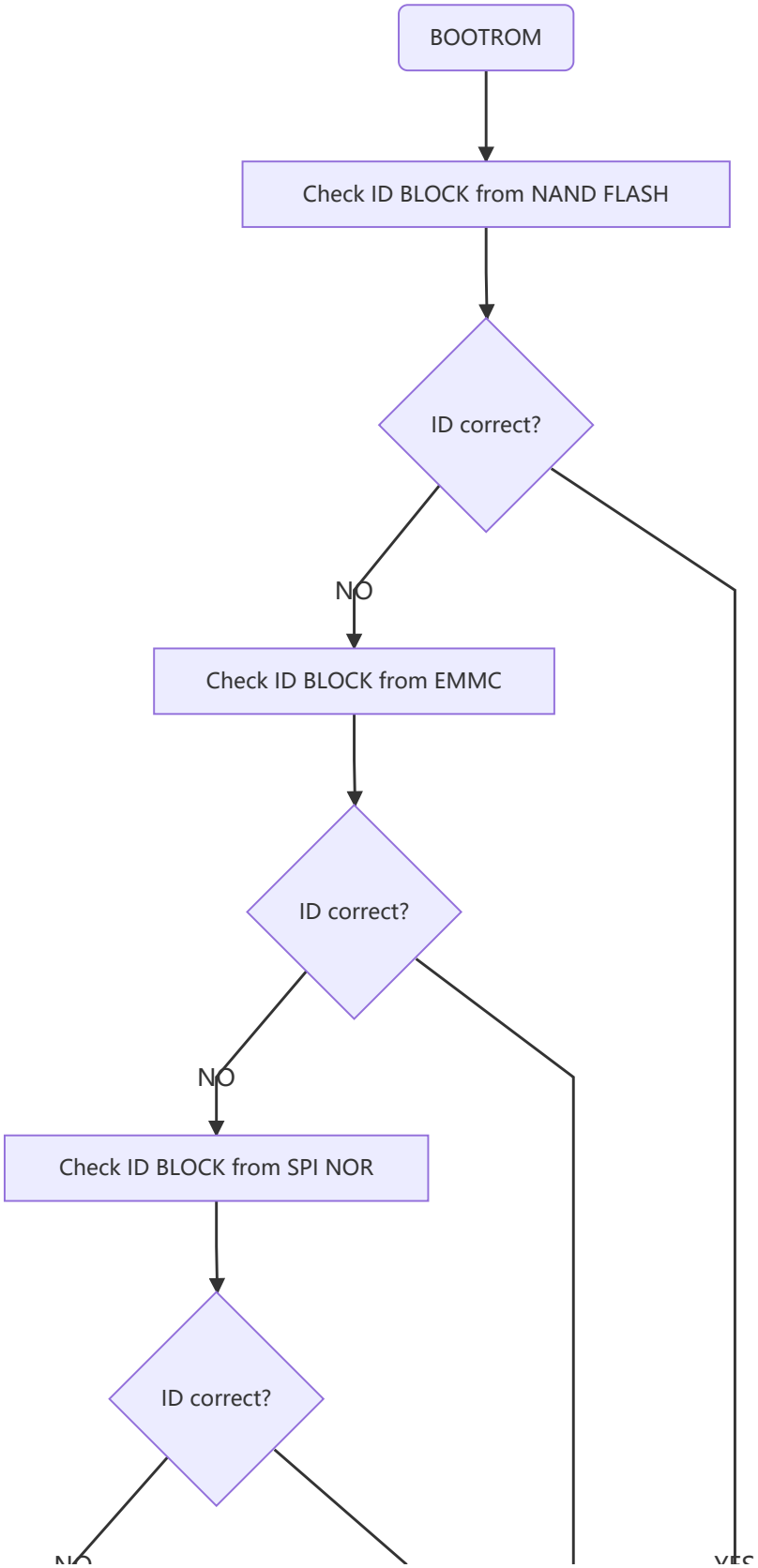
| AP | NANDC | SPI0 | SPI1 | SPI2 | SFC | SD | SDIO | EMMC | USB0 | USB1 |
|---|---|---|---|---|---|---|---|---|---|---|
| RK3188 | 60bits MLC SLC | Boot | | - | - | SD 3.0 | SDIO 3.0 | SD50 DDR50 | 2.0 Host | 2.0 OTG |
| RK3128 | 60bits MLC SLC | | - | - | Boot | SD 3.0 | SDIO 3.0 | SD50 DDR50 | 2.0 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 | | |
| RK3568 RK3566 | 70bits TLC MLC SLC | | | | Boot | SD 3.0 | SDIO 3.0 | HS200 | 2.0 HOST | 3.0 OTG(RK3568 Only) |
| 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 |

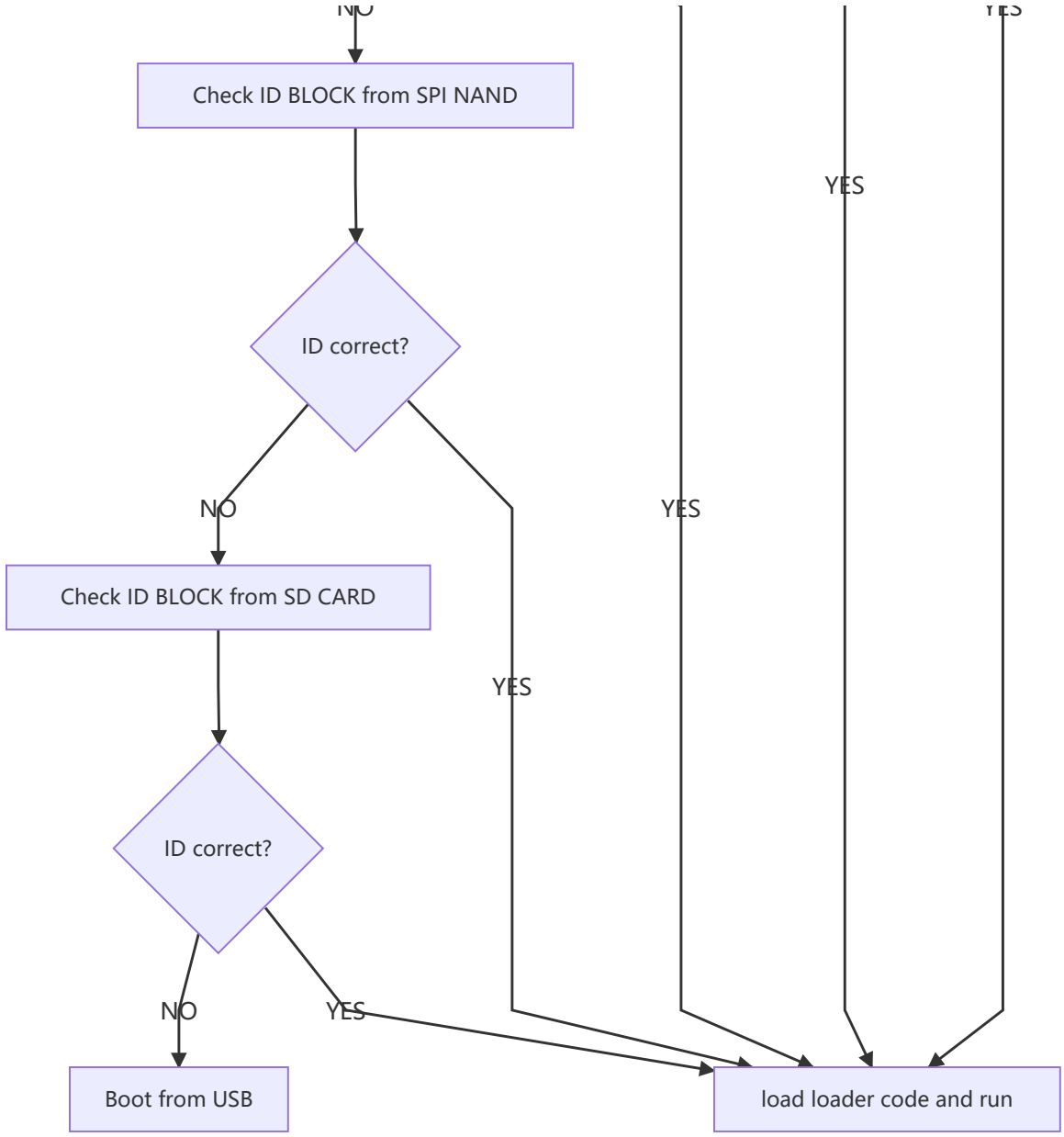
*1: The chip hardware supports it, but the SDK release development package does not support it.

## 4.3 BOOTROM Process

Both the AP and MCU have an integrated BOOTROM. When the system is powered on, the BOOTROM code will be executed first, and then the BOOTROM code will detect peripheral memory and load the Loader code.

The order in which different chips' BOOTROMs detect peripheral memory may vary. The following diagram is an example of the BOOTROM startup process:






Note：

- Some chips have a setting that supports using different input levels for the ADC Key to specify the storage device for BOOTROM probing
- Detecting memory devices is usually confirmed by the detector device ID to determine if they are from external devices
- If no valid firmware is detected in all devices, the device entersIf no valid firmware is detected in all devices, the device enters maskrom mode and waits for a specific interface such as USB/UART to download the firmware. However, not all chips support USB/UART interface upgrades.

**Boot ROM Boot Sequence for Each Chip**

| 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 Process

The image generated by compiling the RK SDK project usually includes a file named MiniloaderAll.bin, which actually has two main functions:

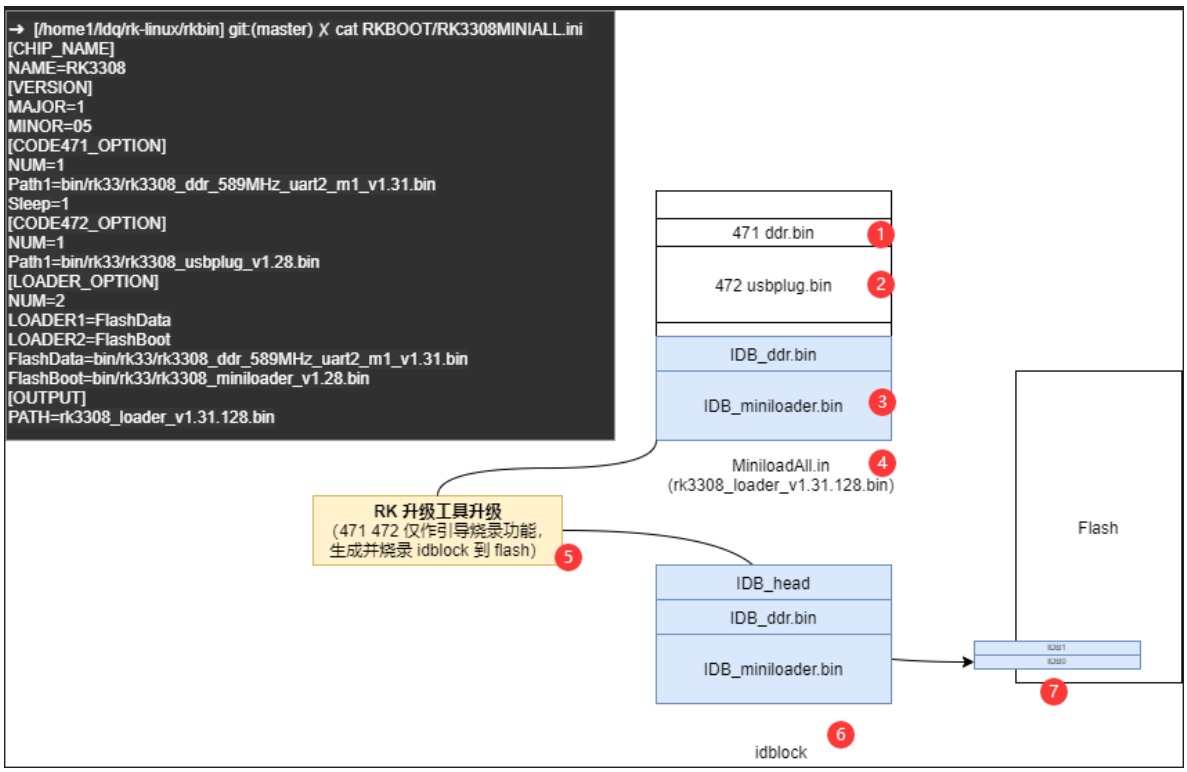
- Boot burning
- Burning idb image

The idb image is an effective image extracted from MiniloaderAll.bin, also known as idblock, and is finally burned into the flash memory. Typically, the idb image is packaged by ddr.bin and Pre Loader image, and some chips support packaging more functional images. Currently, there are three main types of Pre Loader: miniloader (non-open source), uboot spl, and loader.

## 4.4.1 Miniloader

**Introduction**

The miniloader firmware is the non-open source preLoader firmware of RK, which is usually packaged with ddr.bin usbplug.bin as loader.bin, with the following structure:




Comments:

1. ddr.bin: DDR initialization firmware, abbreviated as 471
2. usbplug.bin: Boot firmware for burning, abbreviated as 472
3. miniloader.bin: Closed-source pre-loader firmware
4. MiniloaderAll.bin: Unified naming file for SDK, actually the loader.bin for the corresponding chip packaging firmware
5. During the upgrade process of RK upgrade tool, it will extract and upgrade idblock.bin from loader.bin and make multiple backups
6. idblock, the firmware is the packed firmware of ddr.bin + pre-loader, and the closed-source version of pre-loader refers to miniloader.bin

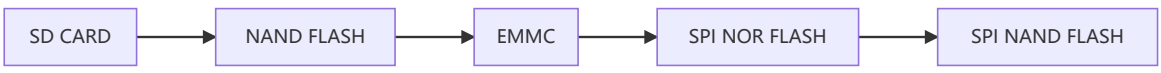
**rkbin repository packs and generates Miniloader**

Taking rk3308 as an example, enter the sdk directory in the rkbin directory of the SDK, and finally generate rk3308_loader_v1.xx.1xx.bin:

```
./tools/boot_merger ./RKBOOT/RK3308MINIALL.ini .
./tools/boot_merger ./RKBOOT/RK3308MINIALL_WO_FTL.ini . /* Files with suffix
_WO_FTL are Pre Loaders selected for open source storage solutions, and the file
system chooses ubifs or jaffs2 */
./tools/boot_merger ./RKBOOT/RK3326MINIALL_SLC.ini . /* Files with suffix _SLC
are Pre Loaders dedicated to small capacity storage (SLC Nand, SPI Nand, SPI
Nor) solutions, with built-in rk ftl algorithm, not supporting ubifs */
```

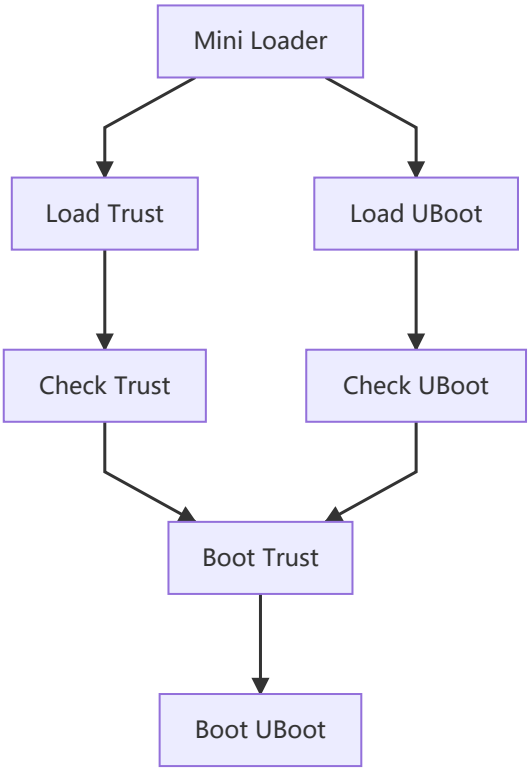
**Pre Loader Process**

Similar to BootRom, in order to be compatible with different storage types, the Preloader stage will also detect different storage peripherals during the boot process:




Due to the code not being open source, users cannot modify the startup sequence on their own.

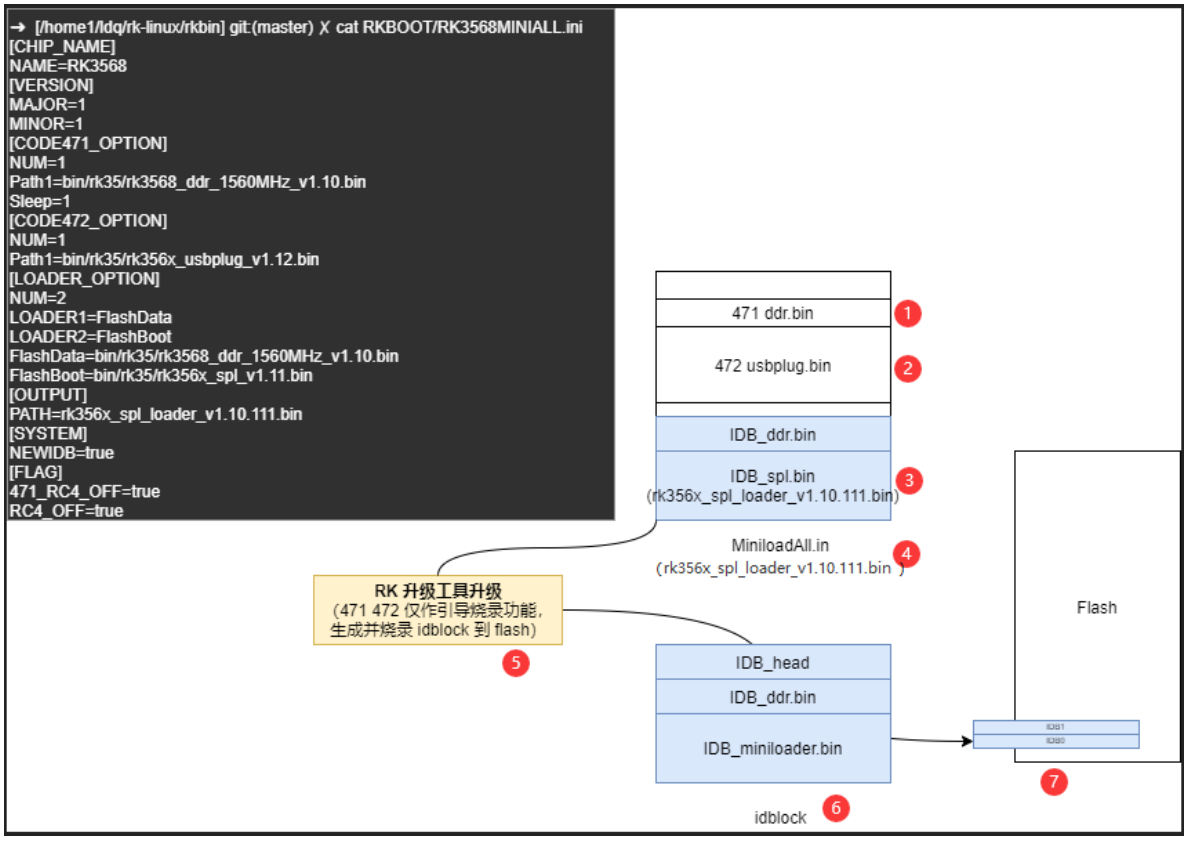
**Boot Flow**




## 4.4.2 u-boot spl

For chip support information, please refer to the document "Rockchip-Developer-Guide-UBoot-nextdev-CN". When supporting NAND and SPI NAND without FTL algorithm, only use the open source NAND driver and it is recommended to use the UBIFS file system.

**Introduction**



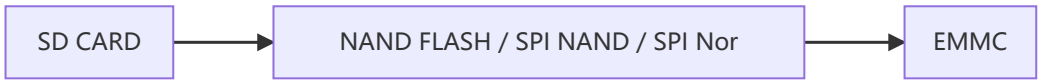
Comments:

1. ddr.bin: DDR initialization firmware, abbreviated as 471
2. usbplug.bin: Boot firmware used for burning, abbreviated as 472
3. spl.bin: u-boot source code compiled spl firmware, output file in the uboot directory of spl/u-boot-spl.bin
4. MiniloaderAll.bin: Unified naming file for the SDK, actually the spl_loader.bin corresponding to the chip packaging firmware
5. During the RK upgrade tool upgrade process, idblock.bin will be extracted from loader.bin for upgrading and multiple backups will be made.
6. idblock, the firmware is the packed firmware of ddr.bin + pre-loader, and the spl version pre-loader refers to u-boot-spl.bin, RK SDK will regularly compile u-boot-spl.bin and store it in the corresponding directory of rkbin, named as rkxxxx_spl_vx.xx.bin.
7. The PC upgrade tool SLC Nand/SPI Flash idblock image will be duplicated, while the other storage device idblock images will be backed up five times

Based on the process of BOOTROOM, understand the behavior of "loading xxx. bin and running xxx function":

- Burning process:
  - BOOTROM loads ddr.bin and initializes ddr
  - BOOTROM loads usbplug.bin, using the USB plug firmware (burning)
- Startup process:
  - BOOTROM loads ddr.bin and initializes ddr
  - BOOTROM loads spl.bin, executes the Loader function, and loads the subsequent firmware
  - Some chips also package mcu.bin and PCIe.bin in the idblock, which have similar behavior and goals, pre-loading the pre-loader

**spl Storage Probe Order**



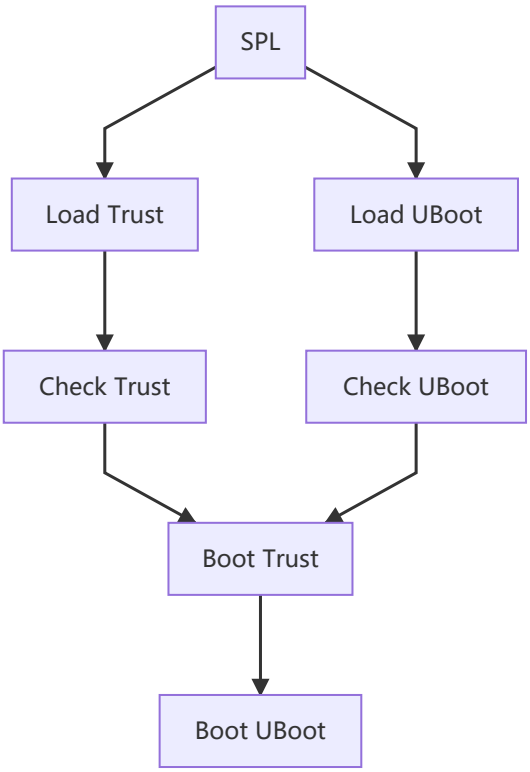

Explanation:

- The spl supports enabling the SD card boot function, detecting the presence of an SD card and a valid firmware within it, loading subsequent firmware from the SD card, and completing SD card boot.
- The spl firmware that supports atags prioritizes detecting and using the storage device successfully detected by BOOTROM for booting. For detailed information on the atags feature, refer to the u-boot development manual.

**spl log**

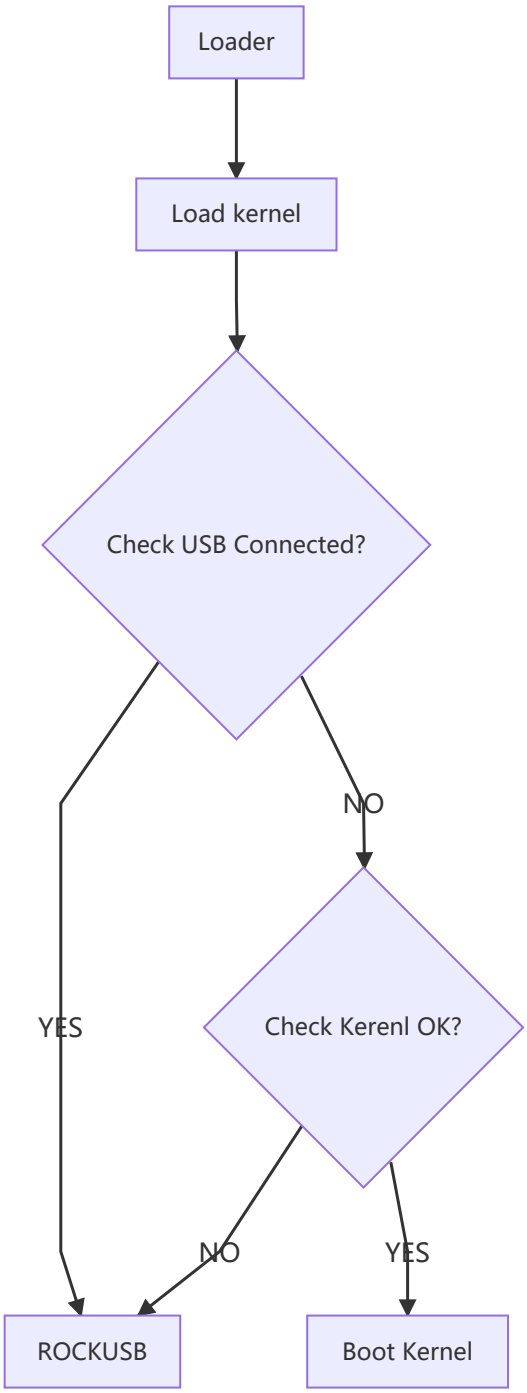
```
U-Boot SPL 2017.09-gcc781e0266-230509-dirty #ldq (Nov 24 2023 - 00:15:39)
unknown raw ID 0 0 0
unrecognized JEDEC id bytes: 00, 00, 00
Trying to boot from MMC2
MMC: no card present
mmc_init: -123, time 0
spl: mmc init failed with error: -123
Trying to boot from MMC1                # bootdev detection: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

**Boot Flow**




**Storage Probe Order**


**Boot Process**




### 4.4.3 loader

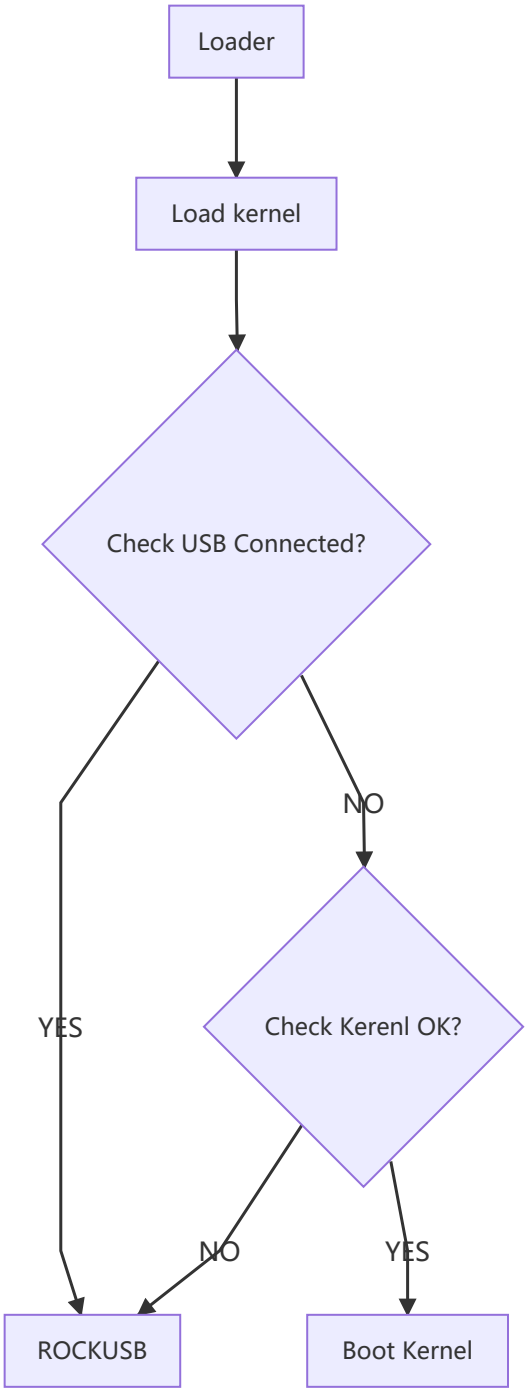
Supports platforms such as RV1107, RV1108, RK3036, RK3128, and RK3229. It is generally used to support small capacity storage without using uboot and directly boot the kernel.

| Chip | SD CARD | SLC NAND | EMMC | SPI NOR | SPI NAND |
|---|---|---|---|---|---|
| RV1107/8 | Supported | Supported | Supported | Supported | Supported |
| RK3036 | Supported | Supported*1 | Supported*1 | Supported | Supported |
| RK3128 | Supported | Supported*1 | Supported*1 | Supported | Supported |
| RK3229 | Supported | Supported*1 | Supported*1 | Supported | Not Supported |

*1 RK3036, RK3128, and RK3229 projects using SLC NAND and EMMC usually use miniloader directly.

**Storage Probing Order**


**Boot Process**



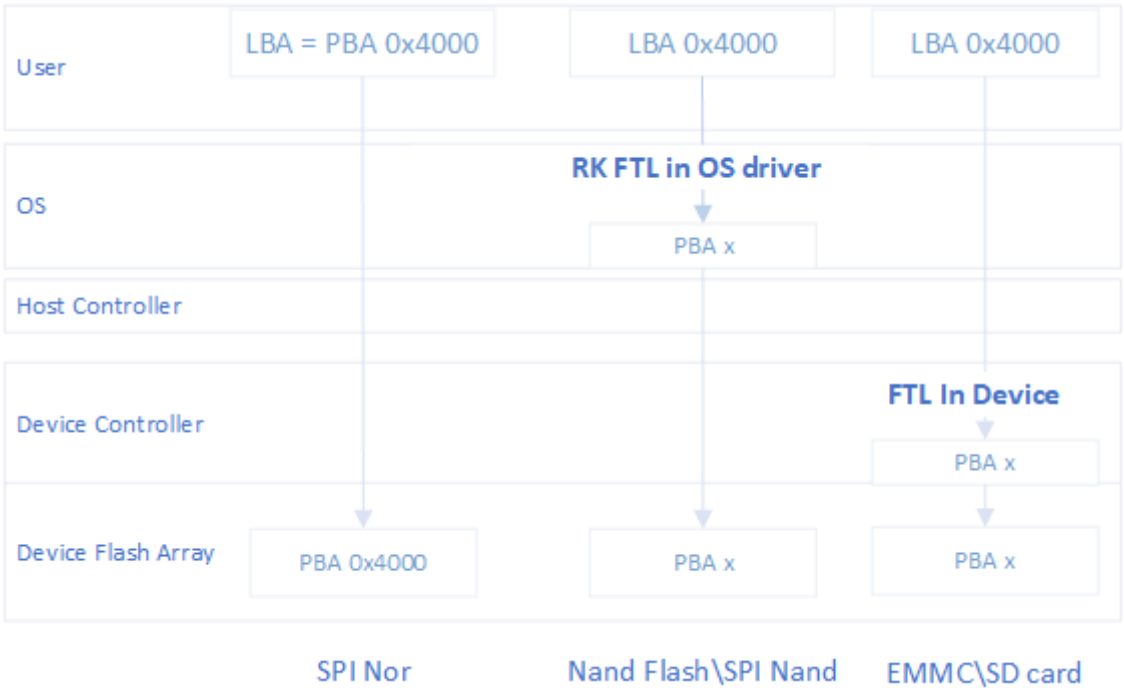
# 5. Partition and Data Storage

## 5.1 Data Storage

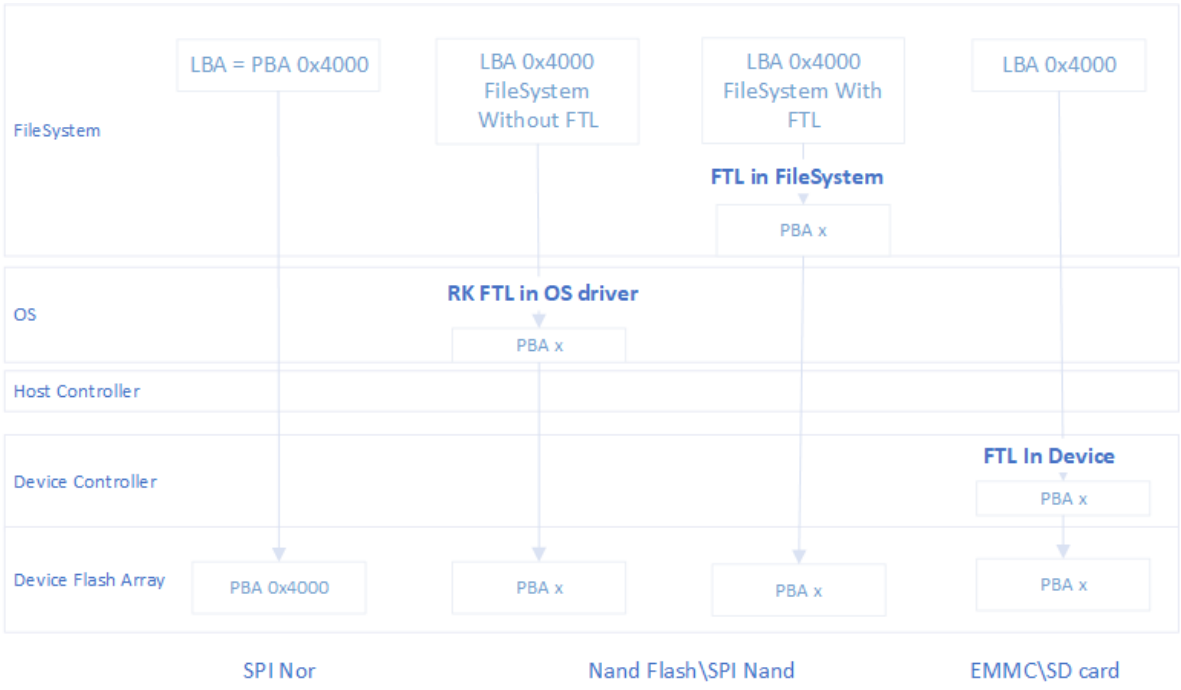
### 5.1.1 Introduction to Address Conversion

If you have knowledge of storage, you should know that most storage chips are not flat mapped, but are converted from user logical sector address (lba) to Flash physical sector address (pba) by the user. This mapping process is called FTL (Flash translation layer). FTL needs to integrate data wear, bad block management, garbage collection and other requirements for address conversion. Whether or not there is an FTL, users and files only need to care about logical addresses, and the details of address conversion are completed by software.

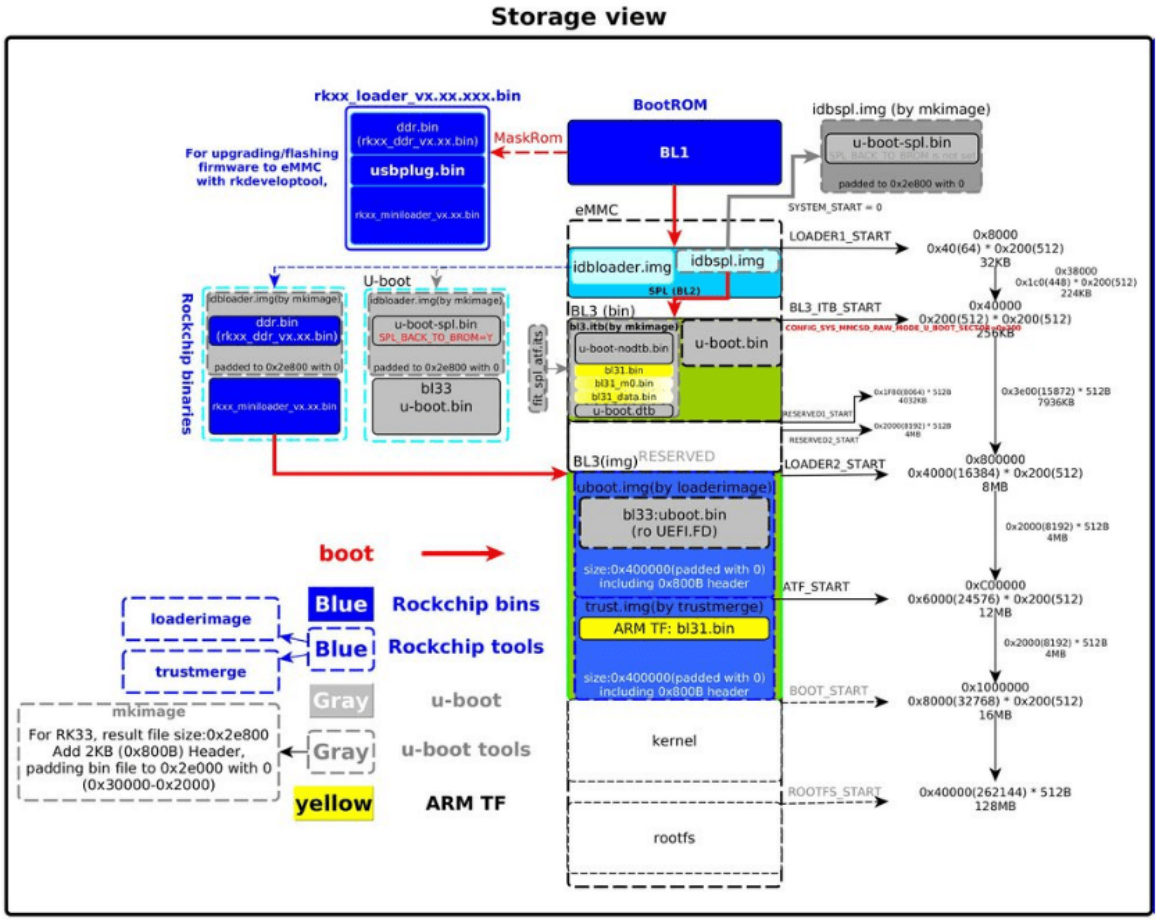
Assuming a user needs to access the address 0x4000 sector, the address conversion relationship is as follows:




Assuming a file system interface accesses the address 0x4000 sector, the address conversion relationship is as follows:




### 5.1.2 Partition and Data Logical Address Storage




## 5.2 Partition Table Partition

In the RK storage solution, there are three types of partition tables that can be solidified into the storage partition: MTD Partition, GPT, and RK partition.

For more detailed information, please refer to the document "Rockchip_Introduction_Partition".

| Partition | Description | Applicable Platforms | Restrictions |
| --- | --- | --- | --- |
| MTD Partition | Defined in the parameter file and passed through cmdline, no longer supported by the uboot-next branch | All AP*1 | Needs to be stored in a separate partition |
| GPT | EFI general partition table, supported by the uboot-next branch | All AP*2 | Uses a little more resources |
| RK partition | Designed with reference to GPT, mainly used for small capacity storage to save resources | RV1107/8, MCU | RK custom, not universal |

*1 Platforms using the uboot-next branch no longer support MTD partition. If needed, it must be adapted by yourself.

*2 Platforms using the uboot-next branch default to using GPT as the partition table. If another partition table is needed, it must be adapted by yourself.

### 5.2.1 MTD Partition

Refer to the "Rockchip_Developer_Guide_Linux_Flash_Open_Source_Solution_EN.md" document.

### 5.2.2 GPT

GPT partition table is also configured through a parameter file, and its structure is similar to MTD Partition. The four differences are:

1. Set TYPE to GPT.
2. No definition of parameter partition (if defined, it will not be used).
3. The last partition needs to add the keyword "grow".
4. Need to specify the rootfs uuid, which may be different for different SDKs and needs to match the rootfs uuid defined in DTS.

```
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 partition */
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 partition table upgrade process:

1. Tool reads partition definition in parameter file
2. Get storage device capacity from loader
3. Modify last partition size and create gpt partition table file
4. Burn partition table to storage device's 0 address and - 33 (end) address

   Note: 1. The parameter file itself will not be burned to the storage device.

### 5.2.3 RK partition

RK is a customized partition table with a structure similar to GPT, which occupy less resources and initialize faster. It is mainly used on RV1107/8 platforms and MCU platforms.

Below is the template for Linux\Android product partition definition files:

```
#Flag currently only has two values, 1 for partition that needs to be
downloaded, 0 for no need to download
#type currently has 5 values, 0x1=Vendor partition, 0x2=IDBlock partition,
0x4=Kernel partition, 0x8=boot partition, 0x80000000 = ordinary partition
#PartSize and PartOffset fields' values are in sectors
[System]
FwVersion=16.12.23
# If Nano=1, then generate an idblock in nano format
Nano=
# If BLANK_GAP=1, then the generated idblock will be saved with a blank of 2k
data interval every 2k data interval
BLANK_GAP=1
#FILL_BYTE indicates what data is used to fill the blank at the end of the
partition, default is 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
```

Below is the template for RTOS product partition definition files, where the bits [8,10] flag in the Flag field are only valid for RTOS products:

```
#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 support 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
```

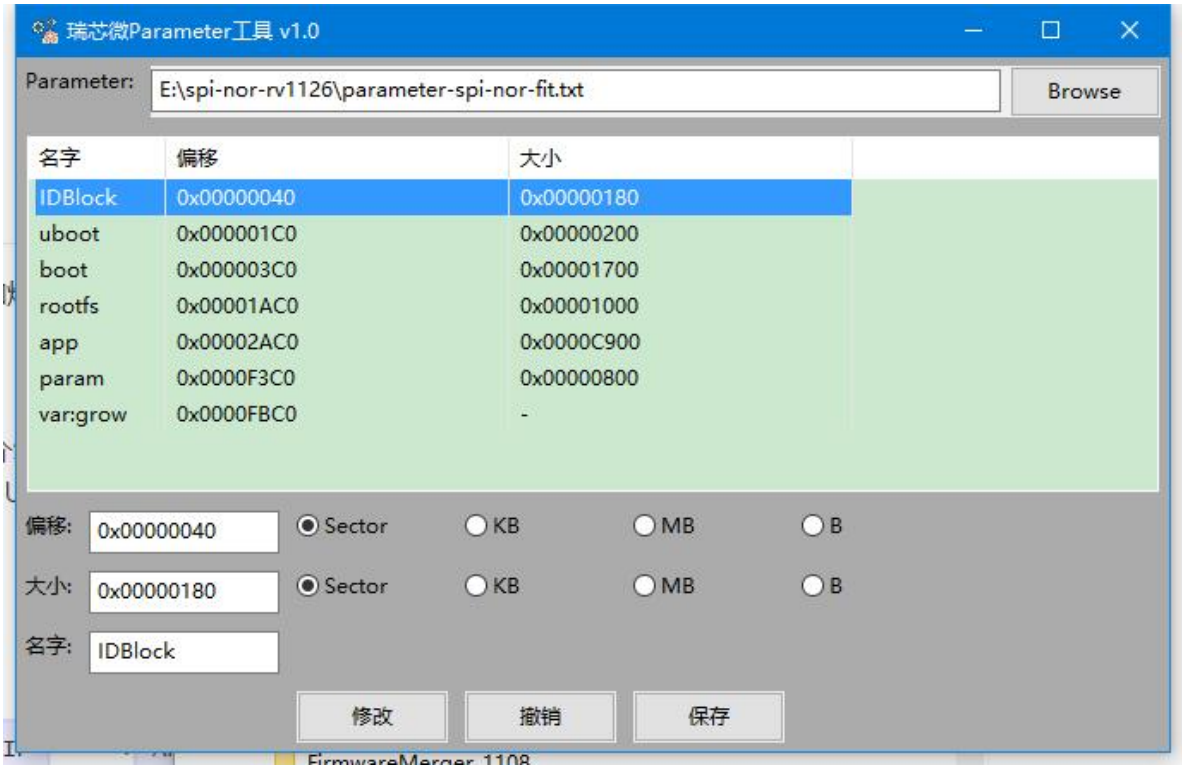
### 5.2.4 ENV Partition

ENV (Environment-Variables) is a global data management and transfer method supported by U-Boot. The principle is to build a HASH mapping table, managing user data as "key-value" entries.

Some RK chip platforms use the ENV information stored in flash, defining it as an ENV partition. At the same time, the mtdparts partition table information in cmdlines is generated and recorded in the ENV information table, and passed from SPL or U-Boot to the kernel.

## 5.3 Partition Table Modification Tool

The Partition Table Modification Tool can be used to modify partitions defined by the parameter, and when a partition size is modified, the offset of subsequent partitions will match the modification.




## 5.4 Partition Write Protection Settings

### 5.4.1 Block Device Partition Write Protection Settings

In the Linux Kernel, EMMC and SD CARD are block devices, and when using the rknand or rkflash driver for NAND FLASH, they are also block devices. You can configure the read-write attribute of partitions through the following commands.

Example 1: Set the system partition to read-only:

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

Example 2: Set the system partition to read-write:

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

Note: It is recommended that partition configuration is done before partition mounting, otherwise if partition mounting is set to read-write and the partition property is configured to read-only, the file system will report an error.

### 5.4.2 MTD Device Partition Write Protection Settings

MTD is generally defined by cmdline to partition, and you can set this partition as read-only by adding the character 'ro' after the partition name. You can modify mtdparts when U-Boot passes cmdline to the kernel to achieve specific partition write protection.

Example: Modify the partition table and set the boot partition as read-only:

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

# 6. Firmware Burning

Currently, there are three main ways to burn firmware in mass production: USB upgrade, SD card upgrade, and programmer burning.

## 6.1 USB Upgrade

There are currently two protocols for USB upgrade: rockusb and fastboot. This document only introduces the rockusb upgrade method. If you need to use the fastboot upgrade method, you can refer to the U-Boot development document "Rockchip-Developer-Guide-UBoot-nextdev-CN".

### 6.1.1 Flowchart


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

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

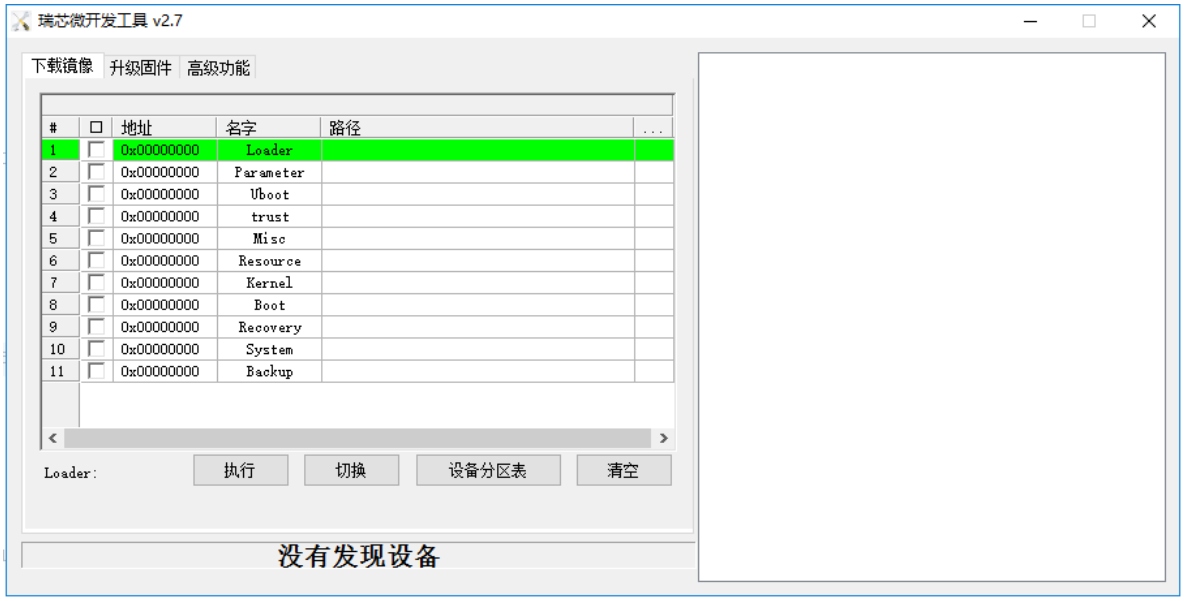
### 6.1.2 WIN Development Tool RKDevTool

GPT/RK Partition Scheme:

When the AP SDK is released, a configured development tool will be provided for burning the complete firmware or updating data of some partitions during development. The tool comes with a variety of features, and detailed feature descriptions can be found in the documentation that comes with the tool. Here are several practical functions:

1. Read device partition table: In the loader upgrade mode, clicking the "Device Partition Table" button allows you to read the device's partition table.
2. Switch to loader upgrade mode: In MSC or MTP mode, you can click the "Switch" button to switch to loader upgrade mode.
3. Switch from loader to maskrom upgrade mode: In the advanced functions, clicking the "Enter maskrom" button allows you to switch from loader upgrade mode to maskrom upgrade mode.
4. Restart the device: In loader mode or maskrom mode, you can click the "Restart Device" function in the advanced functions.

Tool Interface:




Advanced Functions:


1. For maskrom upgrade mode, you need to select the loader file to download and run in DDR.
2. Unpack the update.img firmware.
3. Support script running.
4. Read FLASH ID.
5. Read FLASH information.
6. Read chip information.
7. Read loader support extended functions.
8. Test if it's ready.
9. Restart the device.
10. Restart into maskrom upgrade mode, usually switching from loader upgrade mode to maskrom upgrade mode.
11. Overwrite and write data, clearing the serial number, which may damage the firmware.
12. Export the header IDB structure of the loader.
13. Erase sectors based on the starting address and sector number defined in 16 and 17, which needs to be aligned to 4MB, otherwise it may erase more or less than expected.
14. Export the serial port information for the loader running, saved in the output directory of the tool.
15. Export the firmware image based on the starting address and sector number defined in 16 and 17, saved in the output directory of the tool.
16. Define the starting sector.

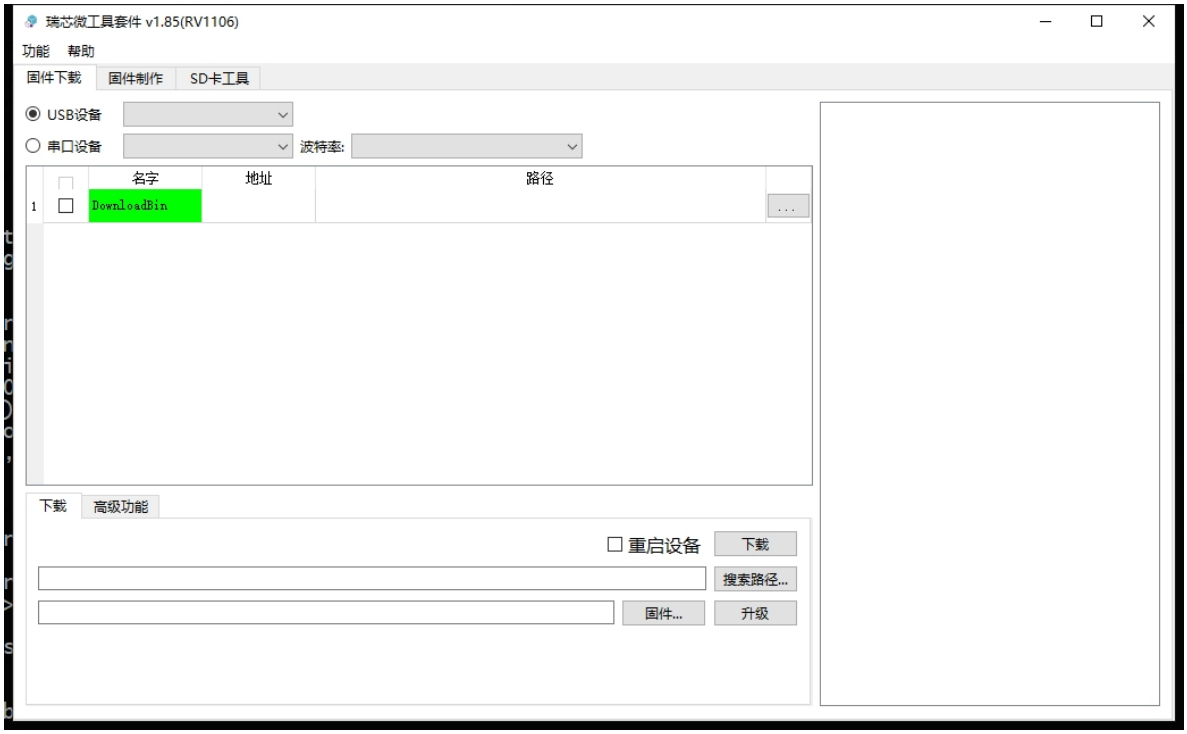
17. Define the number of sectors to operate on.
18. Tool log

### 6.1.3 WIN Development Tool SocToolKit

ENV Scheme:

Some AP platforms of RK support the open-source ENV partition information, which supports recording partition tables, bootargs, and other information in the ENV partition table and passing them to the kernel through the cmdlines method. This scheme has a specific image packaging solution and upgrade tool.

Tool Interface:




### 6.1.4 Linux Development Tool upgrade_tool

The Linux tool is similar to the Android tool and has similar functions.

Tool Interface:

```
[/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]
------------------------------------------------
```

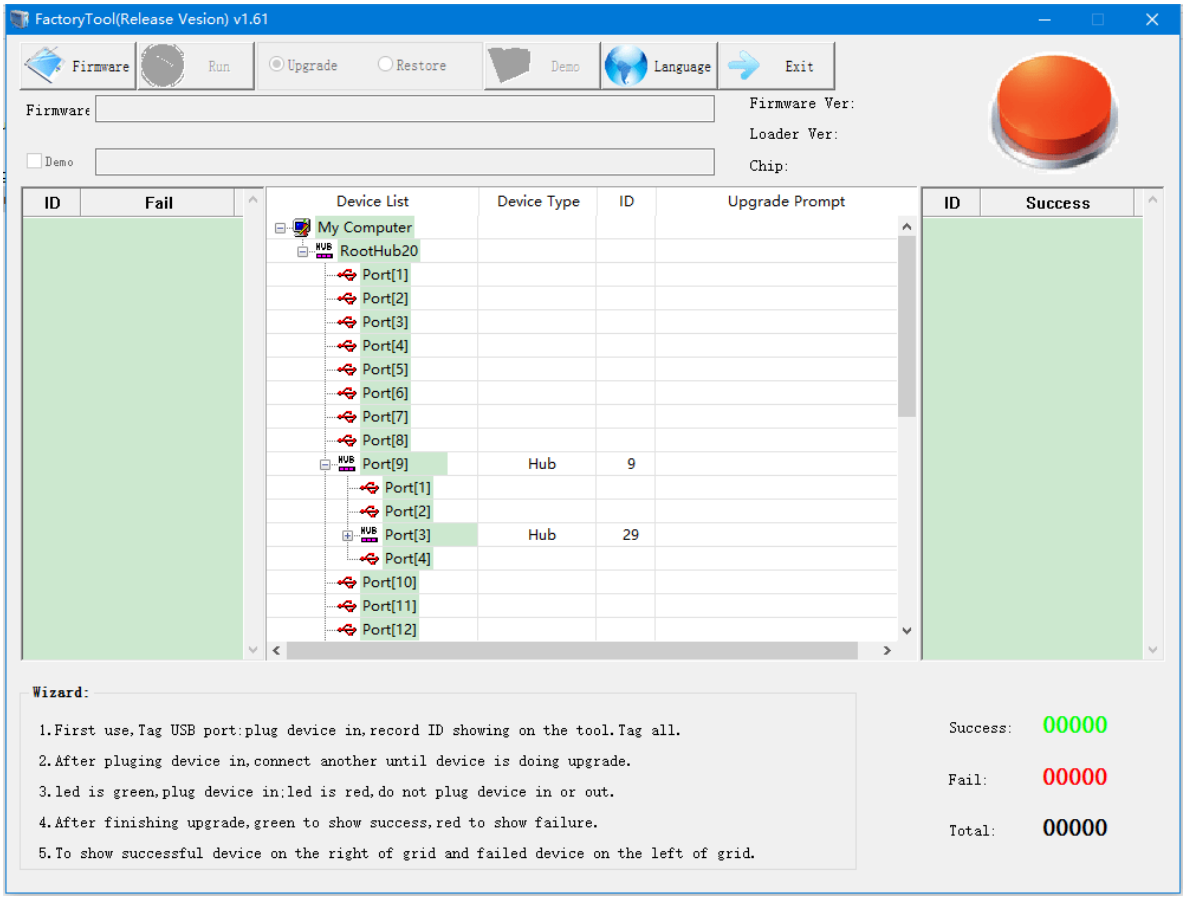
### 6.1.5 Linux Development Tool SocToolKit

The Linux tool is similar to the Android tool and has similar functions.

### 6.1.6 Mass Production Tool

The mass production tool supports one-click multi-async firmware burning. After running the upgrade function, each time a device is connected, the tool will start upgrading the firmware independently for multiple machines.

Tool Interface:




There is a config.ini configuration file in the tool directory, with detailed comments for each option. Here are some commonly used configurations:

1. FW_BURN_EFUSE Burns the firmware while burning the efuse, enabling secure boot.
   AP uses OTP, or if the PCB does not reserve an EFUSE power control circuit, this feature cannot be turned on.
2. NOTRESET_AFTER_UPGRADE Does not restart the machine after upgrading.
   Some products require that the first boot cannot be interrupted, so it needs to be set up not to restart after upgrading the firmware.
3. FORCE_DATA_BAND Modifies the USB single packet transmission data size. If there is a usb timeout error when burning SPI NOR, you can reduce this value.
4. SN_DLL_ON Turns on the function of burning SN during the firmware upgrade process.
5. RB_CHECK_OFF Whether the firmware upgrade needs to read back and call

## 6.2 SD Card Upgrade

Use the SD_Firmware_Tool to burn the update.img firmware into the SD card, and insert the prepared upgrade SD (TF) card into the machine's SD card slot. When powered on, it will start from the SD card to recovery and upgrade the firmware to the internal storage of the machine.

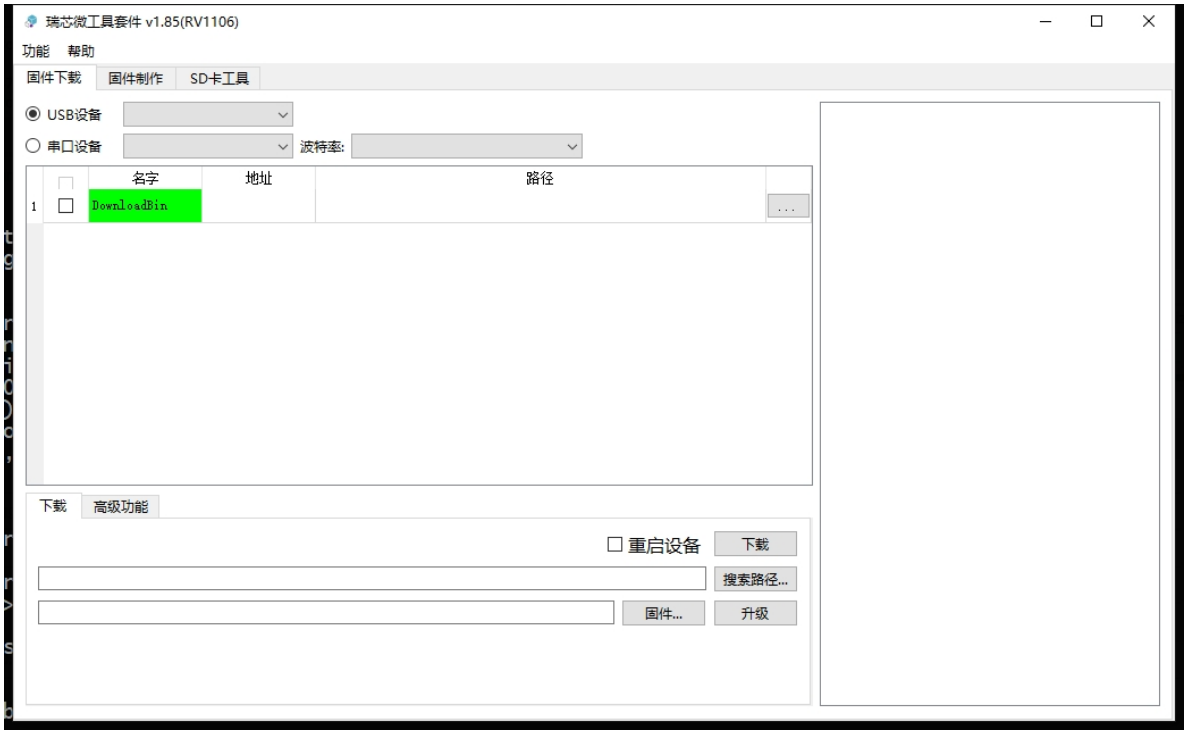
Tool interface:


Tool function description:

1. PCBA test: Check this option to perform PCBA testing before upgrading the firmware.
2. SD boot: Create a boot card, with the complete firmware stored in the SD card.
3. Recover disk: Delete the boot code from the boot card and restore it to a normal SD card.

## 6.3 UART Upgrade

Specific chips support upgrading images through UART interfaces, as well as Linux, Windows, and production tools.



## 6.4 EMMC Image Burning

Use SpiImageTools to convert update.img into an image for the programmer.

Tool interface:


Tool configuration instructions:

1. Blank fill: EMMC selects 0x0
2. SPI FLASH: Do not check
3. Data area reservation: Need to check

If using GPT partitioned firmware, when creating the image, the parameter needs to be configured with the DISKSIZE parameter, refer to the document "Rockchip Mass Production Burning Guide_v1.2" for details.

Programmer configuration instructions:

1. Burn data.bin to the user partition of EMMC
2. If it is RK3188/RKPX3, also burn boot0.bin to the boot1 and boot2 partitions of EMMC
3. Programmer configuration skips all data that is set to 0 and does not burn it
4. CSD values are all used with default values and cannot be modified
5. EXT CSD configuration:
    - For items not listed, use default values and cannot be modified.
    - For RK3188/RKPX3:
      EXT_CSD[167] = 0x1f (if EMMC chip supports, need to configure)
      EXT_CSD[162] = 0x1 (enable reset pin function)
      EXT_CSD[177] = 0x0 (default value)
      EXT_CSD[178] = 0x0 (default value)
      EXT_CSD[179] = 0x8 (0x8, boot from boot1)
    - For other APs:
      EXT_CSD[167] = 0x1f (if EMMC chip supports, need to configure)
      EXT_CSD[162] = 0x0 (default value)
      EXT_CSD[177] = 0x0 (default value)
      EXT_CSD[178] = 0x0 (default value)
      EXT_CSD[179] = 0x0 (default value)

## 6.5 SLC Nand Image Burning

Refer to the corresponding chapter of
"Rockchip_Developer_Guide_Linux_Nand_Flash_Open_Source_Solution_CN.pdf".

## 6.6 SPI Nand Image Burning

Refer to the corresponding chapter of
"Rockchip_Developer_Guide_Linux_Nand_Flash_Open_Source_Solution_CN.pdf".

## 6.7 SPI Nor Image Burning

Refer to the corresponding chapter of
"Rockchip_Developer_Guide_Linux_Nand_Flash_Open_Source_Solution_CN.pdf".

# 7. Storage Software Driver Configuration

RK mainly provides the following storage solutions:

| Abbreviation | Mainly supported flash types | Mainly supported file systems | Supported burning methods |
| --- | --- | --- | --- |
| eMMC scheme | eMMC | FAT、EXT、SquashFS | USB upgrade、SD card upgrade |
| rknand scheme | MLC、TLC Nand | FAT、EXT、SquashFS | USB upgrade、SD card upgrade |
| rkflash scheme | SLC Nand、SPI Nand | FAT、EXT、SquashFS | USB upgrade、SD card upgrade |
| rkflash scheme (SPI Nor support) | SPI Nor | SquashFS、JFFS2 | USB upgrade、SD card upgrade、Burner upgrade |
| SLC Nand open source scheme | SLC Nand | UBIFS | USB upgrade、SD card upgrade、Burner upgrade |
| SPI Nand open source scheme | SPI Nand | UBIFS | USB upgrade、SD card upgrade、Burner upgrade |
| SPI Nor open source scheme | SPI Nor | SquashFS、JFFS2 | USB upgrade、SD card upgrade、Burner upgrade |

## 7.1 u-boot

For detailed information, please refer to Chapter CH05 - Storage Driver Module of "Rockchip-Developer-Guide-UBoot-nextdev-CN".

## 7.2 kernel

Due to the incomplete support for open source SPI Flash in kernel 4.4 and older versions, the open source flash solution in the kernel is different from the implementation under uboot:

| Abbreviation | Mainly supported flash types | Main control driver | Flash framework | Registered device types | Mainly supported file systems | Supported burning methods |
|---|---|---|---|---|---|---|
| rknand scheme | MLC TLC Nand | drivers/rkand | drivers/rkand | block device | FAT、EXT、SquashFS | USB upgrade、SD card upgrade |
| rkflash scheme | SLC Nand、SPI Nand | drivers/rkflash | drivers/rkflash | block device | FAT、EXT、SquashFS | USB upgrade、SD card upgrade |
| rkflash scheme (SPI Nor support) | SPI Nor | drivers/rkflash | drivers/rkflash | block or mtd device | SquashFS、JFFS2 | USB upgrade、SD card upgrade、Burner upgrade |
| SLC Nand open source scheme | SLC Nand | drivers/mtd/nand/raw | drivers/mtd/nand/raw | mtd | UBIFS | USB upgrade、SD card upgrade、Burner upgrade |
| SPI Nand open source scheme | SPI Nand | drivers/rkflash | drivers/rkflash | mtd | UBIFS | USB upgrade、SD card upgrade、Burner upgrade |
| SPI Nor open source scheme | SPI Nor | drivers/rkflash | drivers/rkflash | mtd or mtd block device | SquashFS、JFFS2 | USB upgrade、SD card upgrade、Burner upgrade |

### 7.2.1 MLC Nand、TLC Nand rknand scheme

Configuration:

```
CONFIG_RK_NAND=y
```

Driver files:

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

### 7.2.2 SLC Nand、SPI Nand 及 SPI Nor rkflash scheme

Refer to the "Rockchip_Developer_Guide_Linux_Flash_Open_Source_Solution_EN.md" document.

### 7.2.3 SLC Nand、SPI Nand 及 SPI Nor MTD open source scheme

Refer to the "Rockchip_Developer_Guide_Linux_Flash_Open_Source_Solution_EN.md" document.

## 7.3 Configuration of iomux/clk for storage devices at different stages and scanning order

| Stage | Storage Devices | Configuration of iomux | Configuration of clock | Driver Configuration |
|---|---|---|---|---|
| maskrom | Scan nor、spinand、EMMC、sdcard | Configured (only for devices detected successfully) | Configured | |
| spl | Priority1 (supports atags scheme): maskrom detected devices<br>Priority2: Scan nor、spinand、EMMC、sdcard | 1. Not configured (default)<br>2. Supplementary configuration needed for iommux | Configured (driver-configured) | dts/defconfig configured based on specific sdk |
| uboot | Priority1 (default not enabled): CONFIG_ROCKCHIP_BOOTDEV specifies target storage<br>Priority2 (supports atags scheme): maskrom detected devices<br>Priority3: Scan nor、spinand、EMMC、sdcard | Not configured | Configured (driver-configured) | dts/defconfig configured based on specific sdk |
| kernel | Scan nor、spinand、EMMC、sdcard | | | dts/defconfig configured based on specific sdk |

## 7.4 Expansion of dual storage solution

Refer to the "Rockchip_Developer_Guide_Dual_Storage_CN.md" document.

# 8. Open source OTA solution

Refer to the "Rockchip_Developer_Guide_Linux_Flash_Open_Source_Solution_EN.md" document.

# 9. File system support

## 9.1 UBIFS file system

Refer to the "Rockchip_Developer_Guide_Linux_Flash_Open_Source_Solution_EN.md" document.

## 9.2 JFFS2 file system support

Refer to the "Rockchip_Developer_Guide_Linux_Flash_Open_Source_Solution_EN.md" document.

# 10. Vendor Storage usage instructions

Vendor Storage is designed to store some non-secure small data, such as SN、MAC, etc. For detailed information, refer to the following documents:

- EMMC: "RK Vendor Storage Application Note"
- Flash support for the rkflash solution: "RK Vendor Storage Application Note"
- Flash support for the MTD solution: "Rockchip_Developer_Guide_Linux_Flash_Open_Source_Solution_EN.pdf"

## 10.1 Vendor Storage ID

Vendor Storage is accessed by ID (16 bits), and it does not need to be concerned about where the data is stored in the partition. It can be simply considered that the ID is an index or a filename. IDs 0-31 are reserved for general SDK functions, and customers should use 32-65535 when customizing storage.

The following table defines the function of each ID:

| 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

User applications access vendor storage via IOCTL interface, here are reference code for reading and writing.

```
#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 tool has provided reference source code developed by C++, here are two API interfaces for reading and writing.

```
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 Usage Notes

### 10.3.1 Maximum Data Size for a Single Vendor Partition Item

The Nand and EMMC Vendor partitions combined have a total of 64KB, while the Nor partition has a size of 4KB. These are stored in the vendor structure:

```
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;
};
```

So if you only write one item:

1. The data size for a single Nand and EMMC item is 65488 bytes (64 * 1024 - 32 - 8 -4 -4)
2. The data size for a single Nor item is 4048 bytes (4 * 1024 - 32 -8 -4 -4)

### 10.3.2 Dual Backup Support for VENDOR Data

VENDOR data is backed up by default in two copies, so:

- If power is lost while writing the first copy, the old data will be used (unavoidable)
- If power is lost while writing the second copy, the first copy will be used

# 11. Appendix References

[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