Rockchip RT-Thread SPIFlash

文件标识：RK-KF-YF-080

发布版本：V1.4.0

日期：2024-07-09

文件密级：□绝密 □秘密 □内部资料 ■公开

瑞芯微电子股份有限公司

Rockchip Electronics Co., Ltd.

地址： 福建省福州市铜盘路软件园A区18号

网址： www.rock-chips.com

客户服务电话： +86-4007-700-590

客户服务传真： +86-591-83951833

客户服务邮箱： fae@rock-chips.com

前言

概述

本文主要描述了 ROCKCHIP RT-Thread SPI Flash 的原理和使用方法。

产品版本

芯片名称	内核版本
所有使用 RK RT-Thread SDK 的芯片产品	RT-Thread

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	林鼎强	2020-02-21	初始版本
V1.0.1	陈谋春	2020-03-05	修改无序列表的缩进
V1.1.0	林鼎强	2024-01-18	增加芯片 FSPI/SFC 关键特性、软件框架介绍
V1.2.0	林鼎强	2024-03-26	增加 RK2118 支持、SPI Nand 支持说明
V1.3.0	林鼎强	2024-06-03	优化 SPI Nand 方案
V.1.4.0	林鼎强	2024-07-09	增加 SPI Nand 只读块设备支持

目录

1. 简介

1.1 支持器件

支持 **SPI Nor**：

SPI Nor 具有封装小、读速率相较其他小容量非易失存储更快、引脚少和协议简单等优势，市面通用的 SPI Nor 支持1线、2线和4线传输，且 SPI Nor 具有不易位翻转、byte 寻址和读操作无等待延时（发送 cmd 和 address 下一拍就传输数据）的特点，所以支持使用 XIP技术 （Excute In Place）。

RK RTOS SPI flash 框架提供通用的 SPI Nor 接口和自动化的 XIP 方案。

支持 **SPI Nand**：
SPI Nand 与 SPI Nor 在 RK 平台上使用相同的控制器，RK SDK 在 RT-Thread 也有做针对性的支持，但由于 SPI Nand 的使用依赖 FTL 算法或支持 FTL 算法的文件系统，FTL 算法对于 ram/code size 的消耗较大，稳定性一般，而且 SPI Nand 对于掉电时供电的稳定性要求较高，对于不带电池的 MCU 产品不友好，存在较高的异常掉电风险，所以不建议使用 SPI Nand 存储，仅建议在少量场景下作为只读固件的扩展
存储来使用。

1.2 主控控制器

RK 平台 SPI Flash 可选用的控制器包括 FSPI、SFC、SPI 三种方案。

FSPI (Flexible Serial Peripheral Interface) 是一个灵活的串行传输控制器，有以下主要特性：

- 支持 SPI Nor、SPI Nand、SPI 协议的 Psram 和 SRAM
- 支持 Standard SPI（单线）、Dual SPI、Quad SPI，部分版本支持 Octal SPI
- 支持 SDR（单沿传输），部分版本支持 DTR （双沿传输）
- XIP 技术
- DMA 传输（内置 DMA）

SFC (Serial Flash Controller) 是串行传输控制器，有以下主要特性：

- 支持 SPI Nor、SPI Nand、SPI 协议的 Psram 和 SRAM
- 支持 Standard SPI（单线）、Dual SPI、Quad SPI
- 支持 SDR（单沿传输）
- DMA 传输（内置 DMA）

SPI （ Serial Peripheral Interface ）为通用的 串行传输控制器，有以下主要特性：

- 支持 SPI Nor、SPI Nand、SPI 协议的 Psram
- 支持 Standard SPI（单线）
- 支持 SDR（单沿传输）
- DMA 传输（外部 DMA）

1.3 Nor Flash XIP 技术

XIP（eXecute In Place），即芯片内执行，指 CPU 直接通过映射地址的 memory 空间取指运行，即应用程序可以直接在 flash 闪存内运行，不必再把代码读到系统 RAM 中，所以片内执行代码的运行地址为相应的映射地址。由于 SPI Nor XIP 仅支持读，所以只能将代码段和只读信息置于 SPI Nor 中。

FSPI 除支持 CPU XIP 访问 SPI flash，还支持如DSP 等其他模块以相近方式获取 flash 数据，如同访问一片“只读的 sram”空间，详细 FSPI 信息参考 TRM 中 FSPI 章节。

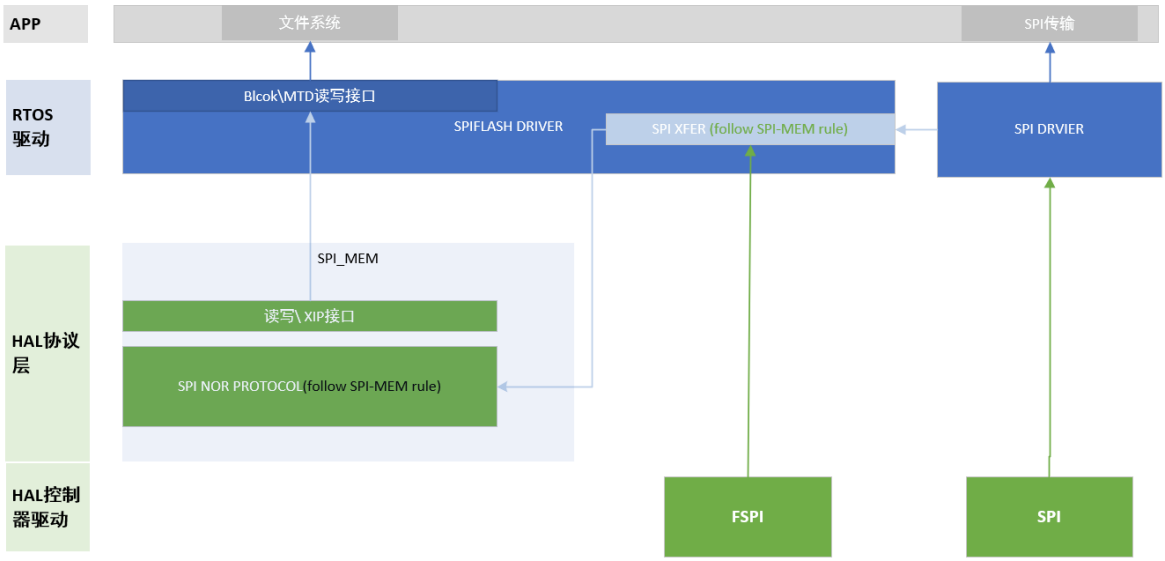
XIP 方案下的 SPI Nor 读函数接口实现方式：

- 支持：
  - XIP 读，访问 XIP 映射空间获取数据，无配置寄存器冗余动作，且支持预取，效率高，整体速率快
  - CPU 读，配置控制器寄存器发起传输，CPU 搬移 fifo 数据
- 读接口默认为 “XIP 读” 实现
- 支持配置 “XIP 读” 实现的起始区间 RT_SNOR_XIP_DATA_BEGIN，例如定义 327680，即：
  - [0: 327680) 空间 "CPU 读" 实现
  - [327680, 尾部] 空间 "XIP 读" 实现

1.4 SPI Flash 驱动框架

考虑到要适配 FSPI、SFC、SPI 两种控制器，所以抽象出控制器层，从而将整个驱动框架分为四个层次，以 SPI Nor 为例：

- MTD 框架层
- RTOS Driver 层，完成以下逻辑:
  - RTOS 设备框架注册
  - 注册控制器及操作接口到 HAL_SNOR 协议层
  - 封装读写擦除接口给用户
- 基于 SPI Nor 传输协议的 HAL_SNOR 协议层
- 控制器层



基于 **FSPI** 控制器的 **RT-Thread** 实现：

- OS 驱动层：drv_snor.c 实现:
  - 基于 FSPI HAL层读写接口封装 SPI_Xfer，并注册 FSPI host 及 SPI_Xfer 至 HAL_SNOR 协议层
  - 封装 HAL_SNOR 协议层提供的读写擦除接口
  - 注册 OS 设备驱动到 MTD 框架层
- 协议层：HAL 开发包中的 hal_snor.c 实现 SPI Nor flash 的协议层
- 控制器层：HAL 开发包中的 hal_fspi.c 实现 FSPI 控制器驱动代码

基于 **SPI** 控制器的 **RT-Thread** 实现：

- OS 驱动层：drv_snor.c 实现:
  - 基于 SPI OS driver 读写接口封装 SPI_Xfer，并注册 SPI host 和 SPI_Xfer 至 HAL_SNOR 协议层；
  - 封装 HAL_SNOR 协议层提供的读写擦除接口
  - 注册 OS 设备驱动到 MTD 框架层
- 协议层：HAL 开发包中的 hal_snor.c 实现 SPI Nor flash 的协议层
- 控制器层：HAL 开发包中的 hal_spi.c 实现 SPI 控制器 low layer 驱动代码，SpiDevice.c 代码实现 RTOS SPI DRIVER 的设备注册和接口封装

注意：

1. 以上实现一一对应附图，可结合阅读
2. 由于 RK SPI DMA 传输相关代码在 OS Driver 层，且 SPI 控制器除了应用在 SPI Nor 上，还支持较多其他器件，存在硬件资源边界保护，所以在 SPI Flash 框架中的 SPI 控制器不应直接套接 HAL 层 hal_spi.c 驱动，而应使用 OS Driver 中的 SPI 接口。

1.5 芯片 FSPI 模块关键特性

仅统计使用开源存储的 SOC 芯片：

SOC	FSPI/SFC IO max rate	FSPI/SFC data io lines	Support XIP
RK2108/Pisces	150MHz	x1\x2\x4 SDR	Y
RK2206	150MHz	x1\x2\x4 SDR	Y
RK2118	SDR：133MHz DTR：80MHz	x1\x2\x4 SDR x4 DDR with/without DQS x8 DDR with/without DQS	Y

说明：

- AP 芯片支持情况参考 《Rockchip_Developer_Guide_Linux_Flash_Open_Source_Solution_CN.md》 文档
- 通常称 x1为 Standar SPI、x2 为 Dual SPI、x4 为 Quad SPI、x8 为 Octal SPI
- Quad SPI 颗粒默认配置为 SDR mode、Octal SPI 颗粒默认配置为 DDR with DQS

2. SPI Nor

2.1 配置

2.1.1 框架简介

简称	主要支持的颗粒类型	主控驱动	**flash** 框架
SPI Nor	SPI Nor	hal: lib/hal/src/hal_fspi.c RT-Thread: bsp/rockchip/commob/drv_snor.c	hal：lib/hal/src/hal_snor.c

说明：

- RT-Thread HAL 源码根目录在 `bsp/rockchip/common/hal`
- 软件已支持的 SPI Nor 颗粒可以查询源码 `lib/hal/src/hal_snor.c`，其余颗粒支持在同级目录下
- 可以通过 flash id 或 flash 丝印来匹配源码
- 软件通过 flash id 识别特定颗粒，软件上默认配置为该控制器最高支持的传输线宽，如需降低线宽，需配置驱动宏开关

2.1.2 驱动配置

在进行该项配置前，需明确硬件上所选用的 SPI Flash 相应的主控类型，以选择相应方案;

基础配置

Menuconfig 配置入口：

```
RT-Thread rockchip common drivers  --->
    [*] Enable ROCKCHIP SPI NOR Flash
    (80000000) Reset the speed of SPI Nor flash in Hz
    [ ]   Set SPI Host DUAL IO Lines /* 如果 FSPI 主控仅预留 IO0~1,应使用 Dual mode
*/
            Choose SPI Nor Flash Adapter (Attach FSPI controller to SNOR) --->
```

配置说明：

```
RT_USING_MTD_NOR=y
RT_USING_SNOR=y
RT_SNOR_SPEED=80000000 /* IO 接口速率 */
# RT_SNOR_DUAL_IO=n /* 默认不配置，Quad SPI 限制为 Duad SPI 使用 */
# RT_SNOR_XIP_DATA_BEGIN=0 /* 默认不配置，XIP 读接口实现起始地址，详细参考 “Nor Flash
XIP 技术”说明 */
RT_USING_SNOR_FSPI_HOST=y /* FSPI 控制器方案 */
# RT_USING_SNOR_SFC_HOST=y /* SFC 控制器方案 */
# RT_USING_SNOR_SPI_HOST=y /* SPI 控制器方案 */
# RT_SNOR_SPI_DEVICE_NAME="spi2_0" /* SPI 控制器方案，指定目标控制器 */
RT_USING_DFS_MNTTABLE=y                 # 启动 DFS mnt table 自动挂载支持
RT_USING_DFS_ELMFAT=y                   # 启用 elm FatFs
RT_DFS_ELM_MAX_SECTOR_SIZE=4096         # 设置 elm FatFs sector size，设定固定为 4KB
```

2.1.3 分区表配置

参考 rockchip 目录下的《Rockchip_Developer_Guide_RT-Thread_CN.md》，"分区表配置" 章节。

须知：

- SPI Nor 默认仅注册 FLAG [8, 9] : property 为 1、2 或者 3 的分区为 RT-Thread 设备，例如 root 分区通常 Flag=0x305，flash 驱动运行后注册为 root 块设备

2.1.4 分区信息生成、解析和注册分区设备

RK RT-Thread SDK 打包后的固件会在 flash 前 2KB 位置生成 RK_PARTITION 分区表，可以通过修改相应的 setting.ini 来调整分区信息。

随固件下载到 SPI Nor 的分区表将在 SPI Nor 初始化成功后进行探测解析并注册（代码在 drv_snor.c 中的 snor_partition_init 函数中实现）。

可以通过以下命令查看相应分区是否注册成功：

```
msh />list_device
device           type         ref count
-------- -------------------- ----------
root     Block Device         1            /* 分区名 root，分区类型 block 设备，Nor
flash 支持 setting.ini 修改设定为 MTD 设备 */
snor     MTD Device           0            /* SPI Nor 根存储设备，分区读写最终接入到该
节点完成读写擦除 */
```

2.1.5 elm-fat 文件系统及分区挂载文件系统

RT-Thread elm-fat 文件支持

```
RT-Thread Components --->
    Device virtual file system  --->
    [*] Using device virtual file system
    [*]   Using mount table for file system /* 实现相应注册分区表，可实现分区上电自动挂
载 */
    [*]   Enable elm-chan fatfs /* fat 文件系统 */
        elm-chan's FatFs, Generic FAT Filesystem Module  --->
        (4096) Maximum sector size to be handled. /*对于 SPI Nor 产品必须修改为 4096
*/
```

如开启分区自动挂载文件系统宏 `CONFIG_RT_USING_DFS_MNTTABLE` ，可在 mnt.c 中添加相应分区注册信息，例如“root”分区到“/”目录：

```
const struct dfs_mount_tbl mount_table[] =
{
    {"root", "/", "elm", 0, 0},
    {0}
};
```

如希望自行设计文件系统挂载流程，也可以通过以下代码实现文件系统挂载：

```
dfs_mount("root", "/", "elm", 0, 0)
```

分区挂载命令：

```
mount root / elm
```

elm-fat 镜像制作：

建议参考 `./bsp/rockchip/tools/mkroot.sh` 脚本实现，关键命令 `mkfs.fat`

2.2 XIP 实现方案须知

前面已经介绍 SPI Nor 支持 XIP 功能，如果选用 FSPI 主控实现的 SPI Nor 方案，会自动开启 XIP 功能，以下介绍产品应用中涉及到 XIP 的一些须知。

2.2.1 添加 XIP 支持

当选用 FSPI 实现的 SPI Flash 方案，并按照 “驱动配置” 章节中关于 FSPI 配置方法去配置，SPI Flash 将默认配置使用 XIP 功能，如要关闭该功能，应则关闭配置 “Enable FSPI XIP”。

2.2.2 XIP 使用过程的开关

RK SPI Flash 框架会在需求的场景自动开关 XIP 功能，客户无需调用开关接口，详细如下:

XIP Off

由于 SPI Nor 擦除/写耗时长的特点， SPI Nor 不支持 XIP 下的擦除/写，所以当 SPI Nor flash 有擦除和写请求的时候，如文件系统写请求， 软件会调用 XIP suspend 接口切换 SPI nor 主控 FSPI 到 normal mode，期间将无法使用 XIP 功能，完整的 suspend XIP 切换流程如下：

1. 通知所有受 XIP disable 影响的 master 模块挂起 XIP 操作
2. 关闭全局中断，避免产生中断导致 CPU 执行放在 SPI Nor 中的 XIP 的代码
3. 关闭 XIP

XIP On

当 SPI Nor flash 擦除/写完成且 FSPI idle 后 SPI Flash 驱动将重新恢复 XIP 功能，也就是在没有擦除写操作的情况下， FSPI 一直将使能 XIP 功能，完整的 XIP resume 流程如下：

1. 开启XIP
2. 使能全局终端
3. 通知所有 XIP suspend 设备恢复使用 XIP

以上所述操作入口如下：

```
static rt_base_t snor_xip_suspend(void)
static void snor_xip_resume(rt_base_t level)
```

总结

- 如果 SPI Nor 颗粒由 FSPI 主控驱动，默认使能 XIP 功能
- SPI Nor 主控 FSPI 在 idle 状态默认开启 XIP 功能，此时支持 XIP 通路的设备都可访问 XIP memory 映射地址获取 SPI Nor 上的数据
- SPI Nor 主控 FSPI 在进行擦除和写行为时关闭 XIP 功能
- SPI Nor 主控 FSPI 开关 XIP 过程，需通知相应设备停止/恢复 XIP 访问，具体按照上文 “XIP 关” 所述操作
- XIP 关，同时会关闭全局中断

2.3 函数接口调用范例

2.3.1 SNOR MTD 字节设备

参考 snor_test.c 源码中的 _at_snor_test 测试代码，主要涉及：

```
snor_device = (struct rt_mtd_nor_device *)rt_device_find("snor");
rt_mtd_nor_erase_block(snor_device, test_lba * block_size, block_size);
rt_mtd_nor_write(snor_device, test_lba * block_size, (const rt_uint8_t
*)pwrite32, block_size);
rt_mtd_nor_read(snor_device, test_lba * block_size, (rt_uint8_t *)pread32,
block_size);
```

说明：

- snor 为字节设备
- rt_mtd_nor_erase_block：单位为 Bytes，最小对齐为 4096Bytes，即 Flash sector size
- rt_mtd_nor_write/read：单位为 Bytes，最小对齐为 1 Byte
- RK partition 分区表单位为 sector（512B），读写分区表信息请注意单位换算

2.3.2 分区 MTD 字节设备

如“分区信息生成、解析和注册分区设备”章节所述，SPI Nor 驱动解析分区表，注册特定分区为分区设备，如果通过 setting.ini 配置为 MTD partition，则默认注册为 MTD 字节设备，接口同 SNOR MTD 字节设备，主要为：

```
snor_device = (struct rt_mtd_nor_device *)rt_device_find("xxxx");
rt_mtd_nor_erase_block(snor_device, test_lba * block_size, block_size);
rt_mtd_nor_write(snor_device, test_lba * block_size, (const rt_uint8_t
*)pwrite32, block_size);
rt_mtd_nor_read(snor_device, test_lba * block_size, (rt_uint8_t *)pread32,
block_size);
```

2.3.3 分区块设备

如“分区信息生成、解析和注册分区设备”章节所述，SPI Nor 驱动解析分区表，注册特定分区为分区设备，如果通过 setting.ini 配置为 Block partition，则默认注册为块设备，接口参考 RT-Thread 标准的 read/write 方式，sector 单位为 4KB。

2.4 测试驱动

建议 SPI Flash 开发流程中引入以下测试流程，做简单的读写测试判断。

2.4.1 驱动配置

```
    RT-Thread bsp test case  --->
        RT-Thread Common Test case  --->
            [*] Enable BSP Common TEST
            [*]   Enable BSP Common SNOR TEST (NEW)
```

如配置成功，在 msh 中会有 snor_test 的 命令选项。

2.4.2 测试命令

输入命令 snor_test 可以获取详细的说明，以下命令单位皆为 byte。

```
1. snor_test write offset size
2. snor_test read offset size loop
3. snor_test erase offset
4. snor_test stress offset size loop
5. snor_test io_stress offset size loop width rw
6. snor_test pm_test
like:
        snor_test write 2097152 4096
        snor_test read 2097152 4096 2000
        snor_test erase 2097152
        snor_test stress 2097152 2097152 5000
        snor_test io_stress 2097152 2097152 5000 4
        snor_test pm_test
```

说明：

- offset：flash 内的偏移地址，非 XIP 地址空间，仅支持 10 进制输入，不支持 16 进制，单位字节
- stress：数据类似为递增序列
- io_stress
  - width：数据类型为支持各种 io 状态的序列，其中 width 4 为 Quad Flash 专用序列，width 8 为 Octal Flash 专用序列
  - rw：1-只写（包括擦除），2-只读，3-读写

2.4.3 测试命令实例

2.4.3.1 读速率

```
msh >snor_test read 2097152 4096 2000
[SNOR TEST] read:  30219d00 + 0x0:0x00000200,0x0000000f,0x000000ff,0x00000000,
[SNOR TEST] read:  30219d00 + 0x10:0x000000ff,0x000000cc,0x000000c3,0x000000cc,
[SNOR TEST] read:  30219d00 + 0x20:0x000000c3,0x0000003c,0x000000cc,0x000000ff,
[SNOR TEST] read:  30219d00 + 0x30:0x000000fe,0x000000ff,0x000000fe,0x000000ef,
snor read speed: 143719 KB/s
```

2.4.3.2 Quad Flash 信号测试

写信号测试：

```
snor_test io_stress 2097152 2097152 5000 4 1
```

读信号测试：

```
snor_test io_stress 2097152 2097152 5000 4 2
```

2.4.3.3 Octal Flash 信号测试

写信号测试：

```
snor_test io_stress 2097152 2097152 5000 8 1
```

读信号测试：

```
snor_test io_stress 2097152 2097152 5000 8 2
```

2.5 常见问题

- 如何判断 **SPI flash** 已经挂载成功？

  通过 list_device 查看是否有 snor 设备节点。

- **Init adapte error ret= -19** 是什么报错？

  “Init adapte error ret= -19” 为存储驱动的初始化函数 snor_adapt 的返回结果打印，相应的错误码解释如下：

  a. -1: SPI Nor ID 为 0xff，可能为 SPI Nor 没有焊接良好，请检查一下电路和信号；

  b. -19: 该颗粒不在支持列表，请联系 RK 工程师，添加相应颗粒支持；

  c. -22: SPI 控制器没有找到。

- 常开 **XIP** 功能是否有多余功耗？

  FSPI 有 time out 机制，主控 idle 一定时长后，自动释放 cs，SPI Nor 将进入 idle 的低功耗状态，所以不会有较大功耗差异。

- 如果多个 **master** 同时通过 **XIP** 访问 **SPI Nor**，是否会有冲突？

  多个 master 同时访问 XIP，总线会做仲裁，串行完成传输，请求会挂在总线上。

- 对于频繁读写文件系统的产品，是否应该使用 **XIP** 功能？

  对于频繁写文件系统的产品，建议将代码段至于 sram 或 psram 中，然后关闭 FSPI XIP 功能，否则将影响 RTOS 系统实时性，同样，使用 XIP 运行 code 段的方案，应减少 SPI Nor 的擦除和写。

- 使能 **XIP** 功能后，文件系统读，是否关 **XIP** ？

  使能 XIP 功能后，文件系统读，驱动实现直接通过 XIP 访问，所以不关 XIP。

- 如果 **SPI Nor** 上没有分区表，还能挂载分区吗？

  必须固化 flash 分区到 flash 前端，RK 目前 SPI Nor flash 存储驱动仅做固化在 flash 中的分区解析挂载方案。

3. SPI Nand

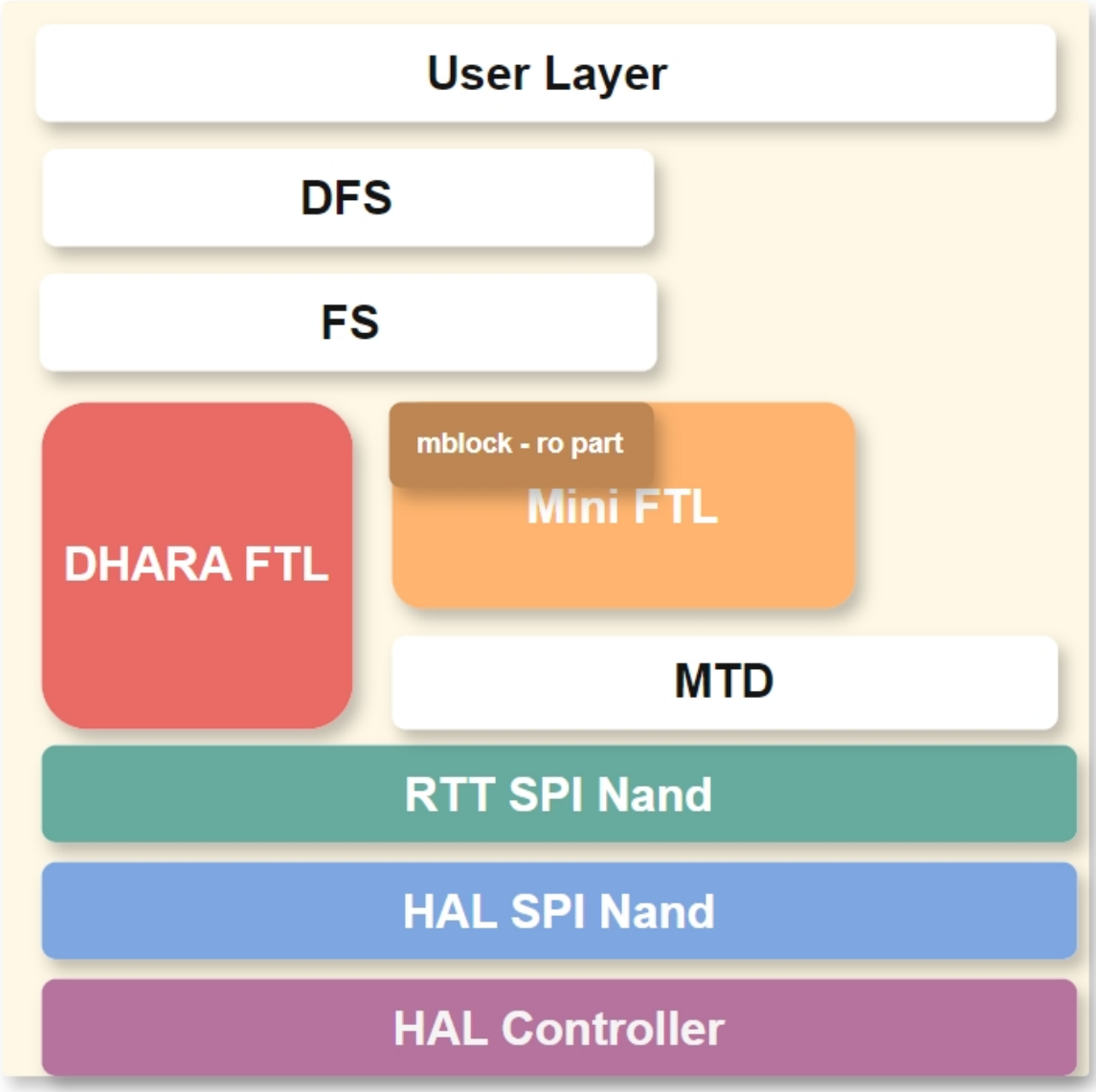
3.1 须知

3.1.1 RK SPI Nand 存储方案简介

主要支持文件系统	支持的烧录方式
FATFS	USB 升级、烧录器升级、OTA 升级

3.2 配置

3.2.1 框架简介



简称	主要支持的颗粒类型	驱动
SPI Nand	SPI Nand	DHARA FTL: bsp/rockchip/common/dhara/ RTT Mini FTL: bsp/rockchip/common/mini_ftl.c RTT SPI Nand: bsp/rockchip/common/drv_spinand.c HAL SPI Nand: lib/hal/src/hal_spinand.c HAL Controller: lib/hal/src/hal_fspi.c

说明：

- RT-Thread HAL 源码根目录在 `bsp/rockchip/common/hal`
- 软件已支持的 SPI Nand 颗粒可以查询源码 `lib/hal/src/hal_spinand.c`，其余颗粒支持在同级目录下
- 可以通过 flash id 或 flash 丝印来匹配源码
- 软件通过 flash id 识别特定颗粒，软件上默认配置为该控制器最高支持的传输线宽，如需降低线宽，需配置驱动宏开关

3.2.2 驱动配置

在进行该项配置前，需明确硬件上所选用的 SPI Flash 相应的主控类型，以选择相应方案;

基础配置

Menuconfig 配置入口：

```
RT-Thread rockchip common drivers  --->
    [*] Enable ROCKCHIP SPI Nand Flash
    (80000000) Reset the speed of SPI Nand flash in Hz
    (y)   Support mini ftl
          Choose SPI Nor Flash Adapter (Attach FSPI controller to SNOR) --->
            Attach FSPI controller to SPINAND
    [ ]     extend fspi cs1 as SPINAND device (NEW)
```

配置说明：

```
RT_USING_SNOR=n                         # 关闭 SPI Nor 驱动，通常外设仅有一种 SPI
Flash
RT_USING_MTD_NAND=y
RT_USING_SPINAND=y
RT_SPINAND_SPEED=80000000               # IO 接口速率
RT_USING_SPINAND_FSPI_HOST=y            # FSPI 控制器方案
RT_USING_FTL=y                          # 启用 SPI Nand FTL 存储管理算法
RT_USING_DHARA=y                        # 启用 SPI Nand DHARA FTL 存储管理算法（与
Mini FTL 兼容）
RT_USING_DFS_MNTTABLE=y                 # 启动 DFS mnt table 自动挂载支持
RT_USING_DFS_ELMFAT=y                   # 启用 elm FatFs
RT_DFS_ELM_MAX_SECTOR_SIZE=2048         # 设置 elm FatFs sector size，设定为 flash
pagesize，通常为 2KB/4KB，参考颗粒手册
```

说明：

- 除了以上配置，还应确认是否已经关闭 XIP 支持，rtconfig.py 文件 “XIP = 'N'”

3.2.3 FTL 算法

3.2.3.1 DHARA FTL 算法

Dhara（音同：Dæ:ra） 是一个开源 FTL 算法，用于在资源受限的系统中管理 NAND 闪存。它提供了一个可变的块接口，具有标准的读写操作。它具有以下附加功能：

- 磨损平衡：任何两个块的擦除计数最多相差 1
- 可以牺牲冗余空间调整 GC 比例来优化 GC 速率
- 数据完整性：写（和修剪）逻辑扇区的操作是原子的。如果断电，状态将回滚到上次同步点。同步点以固定的时间间隔出现，但也可以在需要时主动同步
- 实时性能：所有操作（包括启动）在芯片大小为n的情况下，最坏情况下的时间复杂度为O(log n)，如果坏块均匀分布。

启用 DHARA FTL 算法后，SPI Nand 注册的分区块设备读写接口为 DHARA 读写接口，sector size 为 pagesize，通常为 2KB/4KB。

该算法限制：

- 默认仅支持 Nand 全空间作为 FTL 算法管理空间

当前应用方向：

- 单 FSPI 控制器，SPI Nand 为主存储存放固件镜像及用户数据
- 单 FSPI 控制器，CS0 SPI Nor 为主存储存放固件镜像，关闭 XIP 使用，CSN1 SPI Nand 存放大量用户数据
- 多 FSPI 控制器，FSPI0 SPI Nor 为主存储存放固件镜像，支持 FSPI1 外挂一颗 SPI Nand 存放大量用户数据

3.2.3.2 Mini FTL 驱动

Mini FTL，实际上并非标准 FTL 层，提供的接口仅实现最基础的坏块管理，主要用以分区固件升级支持，注册分区表中命名为 "root" 的分区为同名块设备，用以挂载只读文件系统。

3.2.4 分区表配置

3.2.4.1 分区规划

- 要求分区起始和大小对齐 256KB，兼容 128KB/256KB flash block
- 要求预留分区内坏块替换空间，除分区表分区，其余小分区应预留 1~2 个 flash blocks 空间，大分区超过 8MB 应预留 1% 冗余空间
- 建议预留 2MB 尾部空间，兼容 128KB/256KB flash block 坏块表空间 4 blocks、备份分区表空间 1 good block
- 建议 OTA 的分区规划为 A/B 分区

3.2.4.2 RK Partition 分区扩展

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。

RK SPI Nand 方案默认注册最后一个分区为 DHARA FTL 管理分区，该分区通常为 root 用户分区，用以挂载文件系统。

3.2.4.3 GPT 分区扩展

由于 RK SDK 支持 GPT 分区扩展，所以同样支持 GPT 分区扩展，且默认注册最后一个分区为 DHARA FTL 管理分区，该分区通常为 root 用户分区，用以挂载文件系统。

3.2.4.4 自定义 ROOT 分区扩展

当没有使用分区表分区扩展，可以考虑实现 drv_spinand.c 源码中的以下 ROOT 单分区定义：

```
RT_ROOT_PART_OFFSET
RT_ROOT_PART_SIZE
```

说明：

- 单位为 sectors，512B/sector
- 可以考虑参考 RK3308 RT-Thread build.sh 实现与编译脚本绑定的定义方案

3.2.5 分区信息解析和注册分区设备

SPI Nand 初始化成功后进行解析分区表结构体并注册最后一个分区。

可以通过以下命令查看相应分区是否注册成功：

```
msh />list_device
device           type         ref count
-------- -------------------- ----------
userdata Block Device         0           /* DHARA FTL 实现，分区类型 block 设备，
通常为可读写 FAT 文件系统 */
root     Block Device         1           /* Mini FTL 实现，分区类型 block 设备，通
常为只读 FAT 文件系统 */
spinand  MTD Device           0           /* SPI Nand MTD 设备，分区读写最终接入到该
节点完成读写擦除 */
```

3.3 PC 工具烧录

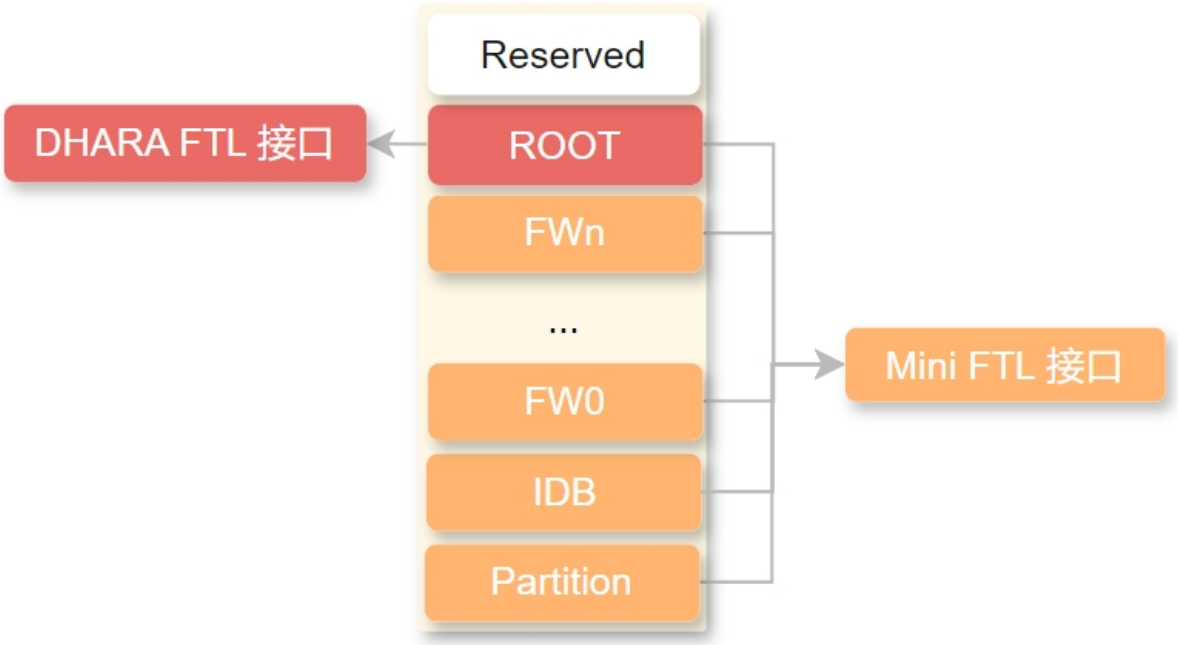
使用 SocToolKit 工具，支持 Windows 和 Linux 版本，并有匹配的量产工具。

特殊逻辑处理：

- 建议支持 OTA 的分区规划 A/B 分区

- 不论是分立镜像还是打包镜像，最终 SPI Nand 升级原理为“按照分区分立烧录对应分区镜像”
    - RK Partition 分区扩展，目前支持分立镜像烧录，暂不支持 Firmware.img 升级方案
    - GPT 分区扩展，支持分立镜像烧录、update.img 镜像烧录
    - 其他分区扩展，仅支持分立镜像烧录

3.4 函数接口调用范例



3.4.1 MTD 字节设备 - 基础接口

参考 spinand_test.c 源码中的 spinand_test 测试代码，主要涉及：

```
mtd_dev = (struct rt_mtd_nor_device *)rt_device_find("spinand0");
rt_mtd_nand_check_block(mtd_dev, block);
rt_mtd_nand_erase_block(mtd_dev, block);
rt_mtd_nand_write(mtd_dev, page_addr + i, (const rt_uint8_t *)buffer, mtd_dev-
>page_size, RT_NULL, 0);
rt_mtd_nand_read(mtd_dev, page_addr + i, (rt_uint8_t *)buffer, mtd_dev-
>page_size, RT_NULL, 0);
```

说明：

- spinand0 为字节设备
- rt_mtd_nand_check_block：确认该 block 是否为坏块，true 坏块，false 好块
- rt_mtd_nor_erase_block：单位为 block
- rt_mtd_nor_write/read：单位为 bytes，最小对齐为 pagesize，通常为 2KB/4KB，详细参考 flash 手册
- RK partition 分区表单位为 sector（512B），读写分区表信息请注意单位换算

3.4.2 Mini FTL 接口 - OTA 接口/文件系统

OTA 接口

参考 spinand_test.c 源码中的 mini_ftl_test_one_flash_block 测试代码，主要涉及：

```
rt_uint32_t block_size = mtd_dev->pages_per_block * mtd_dev->page_size;
mtd_dev = (struct rt_mtd_nor_device *)rt_device_find("spinand0");
mini_ftl_erase(mtd_dev, addr, block_size);
mini_ftl_write(mtd_dev, buffer, addr, block_size);
mini_ftl_read(mtd_dev, buffer, addr, block_size);
```

说明：

- mini_ftl_erase：单位为 byte，最小对齐为 blocksize
- mini_ftl_write：单位为 byte，最小对齐为 pagesize，写数据前要求先调用擦除接口
- mini_ftl_read：单位为 byte，最小对齐为 pagesize
- 建议 OTA 升级时：
    - buffer 对齐 block_size，一次调用完成 erase/write/read，擦写回读校验
    - 升级 elm FatFs 时应先擦除整个分区
    - RK Partition 分区表信息参考源码中 rk_partition_list 接口
    - GPT 分区表信息参考源码中 dfs_part_list 接口，暂不支持 name info

文件系统接口

参考 “分区表配置” 后，默认识别分区表中 root 命名的分区后注册 “root” block 设备 ，使用 Mini FTL 接口，用于挂载只读文件系统。仅支持镜像烧录，不支持在线格式化。

3.4.3 DHARA FTL 接口 - 文件系统接口

参考 “分区表配置” 后，默认识别分区表最后一个分区后注册该分区同名的 block 设备，DHARA FTL 接口，用于挂载可读写文件系统。支持镜像烧录/在线格式化。

3.5 文件系统

3.5.1 elm FatFs 支持

RK SPI Nand 方案提供：

- 基于 Mini FTL 接口实现的 block 设备，只读，支持挂载 FAT 文件系统，烧录镜像为 FAT 镜像固件，不支持在线低格
- 基于 DHARA FTL 接口实现的 block 设备，可读写，支持挂载 FAT 文件系统，基于 FAT 镜像进一步封装为 DHARA 镜像固件，烧录镜像为 DHARA 镜像固件，支持在线低格

3.5.1.1 分区挂载

如开启分区自动挂载文件系统宏 `CONFIG_RT_USING_DFS_MNTTABLE`，可在 mnt.c 中添加相应分区注册信息，例如“root”分区到“/”目录：

```
const struct dfs_mount_tbl mount_table[] =
{
    {"root", "/", "elm", 0, 0},
    {0}
};
```

如希望自行设计文件系统挂载流程，也可以通过以下代码实现文件系统挂载：

```
dfs_mount("root", "/", "elm", 0, 0)
```

分区挂载命令：

```
mount root / elm
```

3.5.1.2 分区在线低格

```
mkfs -t elm userdata
```

说明：

- root 分区为只读分区，仅支持挂载，不支持在线低格

3.5.1.3 elm-fat 镜像离线预制作

elm-fat 镜像制作：

建议参考 `./bsp/rockchip/tools/mkroot.sh` 脚本实现，关键命令 `mkfs.fat`，其中 `-S` 参数修改为 page_size 对齐，通常为 2048/4096，以 SPI Nand 颗粒手册为准。

3.5.1.4 DHARA 镜像离线预制作

DHARA 工具：

```
Usage: ./dhara_utils <file_path> <flash_page_size> <num_blocks>
```

命令实例：离线制作 FAT16 的 DHARA 镜像，假定分区大小 504 blocks，SPI Nand page size 2KB：

```
mkfs.vfat -F 16 -s 1 -S 2048 fat.img
./rkbin/dhara_utils fat.img 2 504
```

说明：

- 输出文件：fat.img.dhara

- 分区大小参考 “分区规划” 说明

3.6 测试驱动

3.6.1 测试驱动基础

```
    RT-Thread bsp test case  --->
        RT-Thread Common Test case  --->
            [*] Enable BSP Common TEST
            [*]   Enable BSP Common SPINAND TEST (NEW)
```

如配置成功，在 msh 中会有 spinand_test 的 命令选项：

```
msh />spinand_test
1. spinand_test write <page_addr> <page_num>
2. spinand_test read <page_addr> <page_nun>
3. spinand_test erase <blk_addr>
4. spinand_test erase_all <blk_addr>
5. spinand_test stress <loop>
6. spinand_test bbt
7. spinand_test markbad <blk_addr>
8. spinand_test mini_ftl_test <byte_offset>
9. spinand_test part_list
like:
        spinand_test write 1024 1
        spinand_test read 1024 1
        spinand_test erase 16
        spinand_test erase_all 512
        spinand_test stress 5000
        spinand_test bbt
        spinand_test mini_ftl_test 67108864
        spinand_test part_list
```

3.6.2 flash_stress_test 读写压测

测试目标 测试 SPI Nand 在读写压力测试下的稳定性

测试设备 需要准备 10 台目标设备

测试方法

假定测试区域为：flash block offset 512 blocks，大小 256 flash blocks：

1. 设备启动后执行 `spinand_test stress 5000 512 256`，并确保设备在测试过程中不会掉电
2. 记录测试过程中的日志
3. 根据测试结果进行判断：如果设备在经过 24 小时后仍正常工作，则测试通过；否则测试失败

3.6.3 power_lost_test 异常掉电

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

测试设备 需要准备 10 台目标设备

测试方法

1. 构建文件拷贝场景
2. 设置掉电器以在供电 2 秒后断电 3 秒，即每组 5 秒，进行 8 小时的测试，即异常掉电约 5760 次，并确保每次掉电后设备能在 5 秒内完成上电并运行文本拷贝动作
3. 记录测试过程中的日志
4. 根据测试结果进行判断，如果设备在经过 24 小时后仍正常工作，则测试通过；否则测试失败

参考源码

参考 “SPI Nand 异常掉电参考源码”。

3.7 常见问题

3.7.1 RK2118 启用 SPI Nand 系统运行卡死

现象：

```
FW data_offset:2048
FW data_size:  195584
FW load_addr:  0x1060800
copy 1090000 20a200 400
hash_256 fail! ret:-1
MemBoot fali!

SoftReset...
```

尝试关闭 XIP 支持：

```
diff --git a/bsp/rockchip/rk2118/rtconfig.py b/bsp/rockchip/rk2118/rtconfig.py
index 2b6e696e2b..32b5117513 100644
--- a/bsp/rockchip/rk2118/rtconfig.py
+++ b/bsp/rockchip/rk2118/rtconfig.py
@@ -29,8 +29,8 @@ if os.getenv('RTT_EXEC_PATH'):
 #BUILD = 'debug'
 BUILD = 'release'

-XIP = 'Y'
-#XIP = 'N'
+#XIP = 'Y'
+XIP = 'N'
 if os.getenv('RTT_BUILD_XIP'):
     XIP = os.getenv('RTT_BUILD_XIP').upper()
```

3.7.2 SPI Nand 不支持 RK Firmware.img 打包镜像升级方案

RK Firmware.img 打包镜像有以下特点：

- 除了最后一个分区支持配置为空镜像，其余分区内有效镜像以外的空间填充 0 数据
- 在有效镜像不变的情况下，随着分区的增大，填充数据的增加，Firmware.img 冗余区域占比增幅明显
- 升级时 Firmware.img 携带冗余数据一次性升级

SPI Nand 器件有以下特性：

- 出厂设备允许存在一定比例的原厂坏块，所以分区内要求留有一定空余空间用以坏块替换

所以，由于以上特点，Firmware.img 打包填充 one firmware 的固件包与 SPI Nand 特性冲突，无法使用 one firmware 的升级方式，即便升级包为 one firmware，最终升级时依旧要求拆包、按照分区仅升级有效镜像。同理烧录器烧录也是不支持单镜像烧录。

3.7.3 SPI Nand 异常掉电问题

掉电关键关联指标

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

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

典型异常

Flash 异常掉电主要会造成以下两种情况：

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

保护机制

正在写入的数据不完整：

- 文件系统有掉电修复机制保证文件系统不受不完整数据影响，可能会丢数据

不稳定低电平导致的随机异常：

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

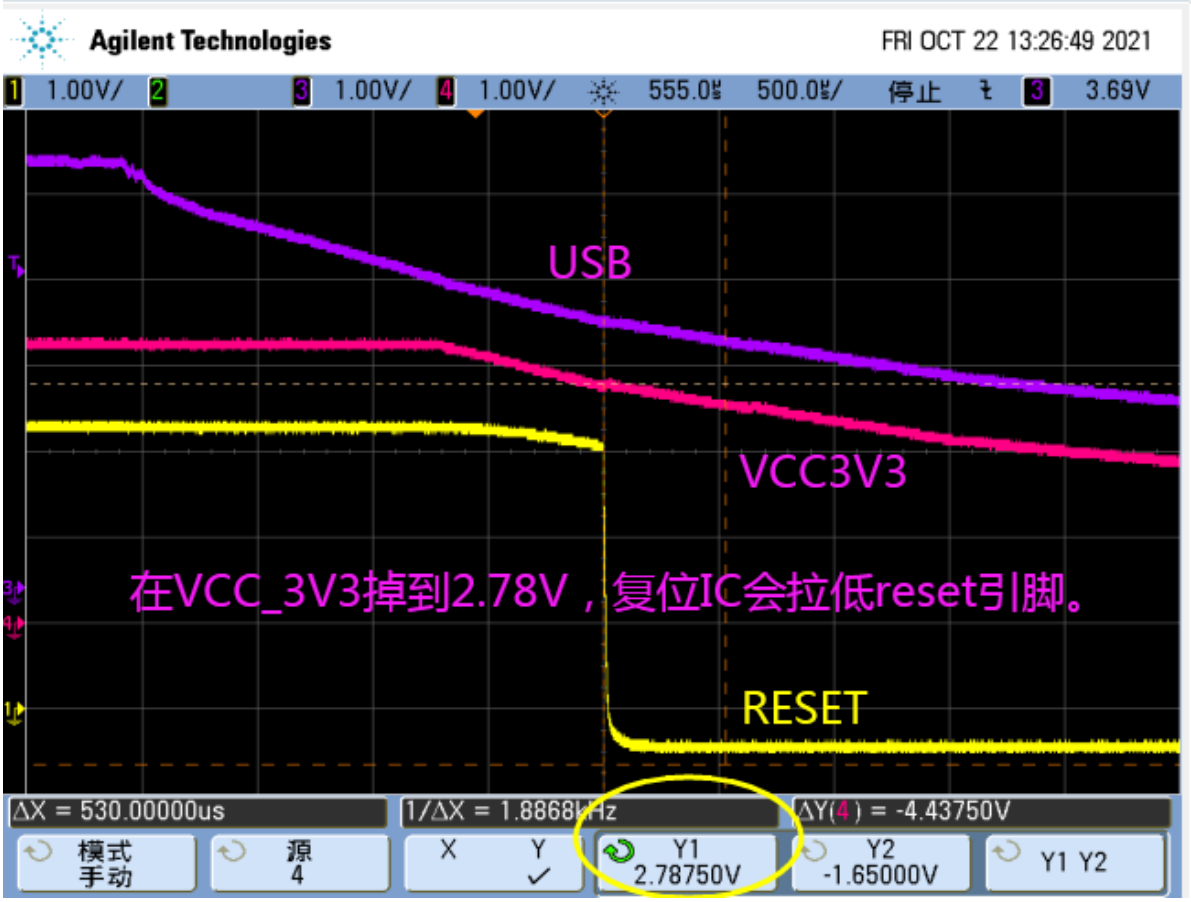
如何避免异常掉电问题

- 建议 Nand 产品设计为带电池或常供电方案，以避免异常掉电行为
- 如无法避免异常掉电行为，建议优化硬件设计，保护 Nand 下电过程的工作电压符合：
    - PMIC/Reset IC/RK NPOR 方案检测 VCC IO 电平，监测掉电行为，触发 CPU 复位，避免低电环境下发起 Nand 擦写命令

- 参考 RK demo 板添加一定的电容，让 Nand 供电 VCCIO flash 维持正常工作电平，时长通常为 tBERS，即块擦除时长
- 减少 flash 读写过程中的掉电行为
- 关键数据做备份和恢复的机制：
  - 固件双备份
  - 关键数据存放在只读文件系统中
  - 可读写分区如果出现掉电异常，建议有恢复机制，保证设备能够正常运行，或者实现 OTA

如何确定异常掉电时序是否合理

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



VCC3V3/RESETn 信号时序测量要求

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

3.7.4 SPI Nand 异常掉电参考源码

```
#include <dfs_file.h>

static void readwrite_test(void *param)
{
    int fd, size, len = 2048, loops = 0;
    char *pwrite, *pread;

    pwrite = rt_malloc(len);
    if (!pwrite) {
        rt_kprintf("malloc pwrite failed\n");
        return;
    }
    pread = rt_malloc(len);
    if (!pread) {
        rt_kprintf("malloc pread failed\n");
        rt_free(pwrite);
        return;
    }

    fd = open("/file_stress.txt", O_RDWR | O_CREAT);
    if (fd < 0)
    {
        rt_kprintf("%s open file failed\n", __func__);
        goto err_open;
    }

    while (1) {
        rt_kprintf("file_stress, loops=%d\n", loops);
        rt_memset(pwrite, loops & 0xff, len);

        lseek(fd, 0, SEEK_SET);
        write(fd, pwrite, len);
        lseek(fd, 0, SEEK_SET);
        size = read(fd, pread, len);
        if (size < 0) {
            break;
        }

        if (rt_memcmp(pwrite, pread, len)) {
            rt_kprintf("%s compare failed, %x\n", __func__, pread[0]);
            break;
        }
        loops++;
    }

    close(fd);
err_open:
    rt_free(pread);
    rt_free(pwrite);
}

void file_stress(void)
```

```
{
    rt_thread_t thread;

    thread = rt_thread_create("file_stresss",
                              readwrite_test, NULL,
                              2048,
                              21, 20);
    rt_thread_startup(thread);
}
```