Linux SPINOR
开发指南

版本号: 1.4
发布日期: 2024.10.23

版本历史

版本号	日期	制/修订人	内容描述
1.0	2021.12.21	AWA1669	初始版本
1.1	2022.02.22	AWA1669	增加 uboot shell 使用
1.2	2022.11.10	AWA1669	增加 linux5.10 配置说明
1.3	2024.08.14	AWA2155	增加 uboot2023 配置说明
1.4	2024.10.23	AWA1669	增加存储布局说明，新增了开发功能、日志分析、调试方法和 FAQ 章节

目　录

插　图

1 引言

1.1 编写目的

此文档描述 Sunxi NOR 模块的使用方法，为相关人员调试提供指导

1.2 适用范围

boot0: 适用于 brandy-2.0

u-boot: 适用于 u-boot-2018/u-boot-2023

kernel: 适用于 linux-4.9/linux-5.4/linux-5.4-ansc/linux-5.10/linux-6.6 内核

1.3 相关人员

BSP 的开发人员、测试人员

2 模块介绍

2.1 模块功能介绍

NOR Flash 和普通的内存比较像的一点是他们都可以支持随机访问，这使它也具有支持 XIP（eXecute In Place）的特性，可以像普通 ROM 一样执行程序。这点让它成为 BIOS 等开机就要执行的代码的绝佳载体。

NOR Flash 根据与 Host 端接口的不同，可以分为 Parallel NOR Flash 和 Serial NOR Flash 两类。

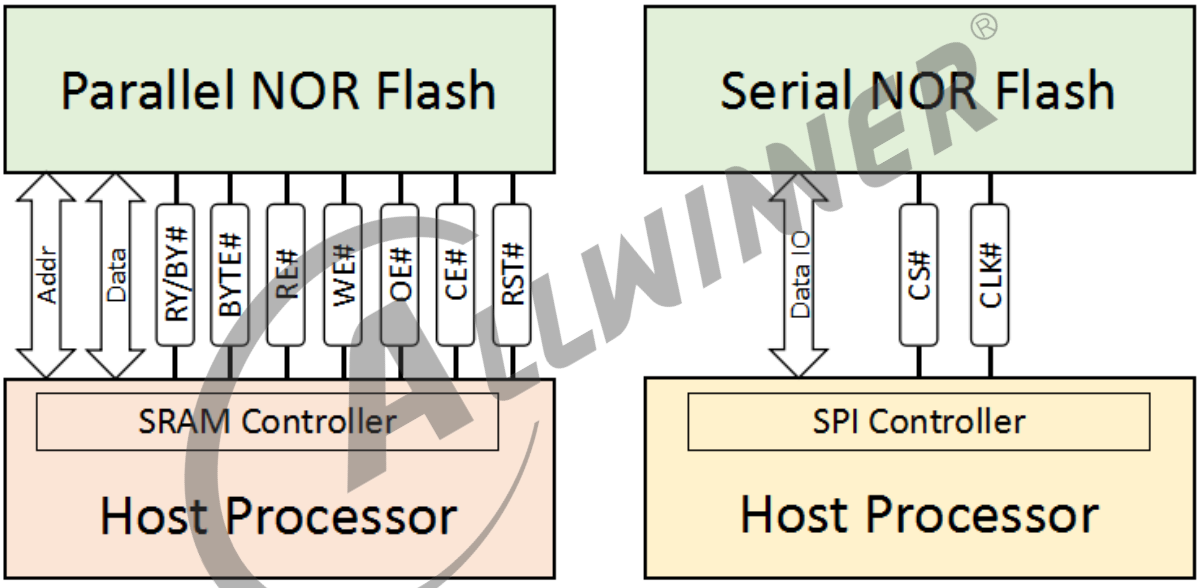


图 2-1: NOR 接口类型

Parallel NOR Flash 可以接入到 Host 的控制器上，所存储的内容可以直接映射到 CPU 地址空间，不需要拷贝到 RAM 中即可被 CPU 访问。NOR Flash 在 BIOS 中最早就是这种接口，叫做 FWH（Firmware HUB），由于其接是并行接口，速度缓慢，现在基本已经被淘汰。Serial NOR Flash 的成本比 Parallel NOR Flash 低，主要通过 SPI 接口与 Host 也就是 PCH 相连。

Linux 中 SPINOR 体系结构如下图所示：

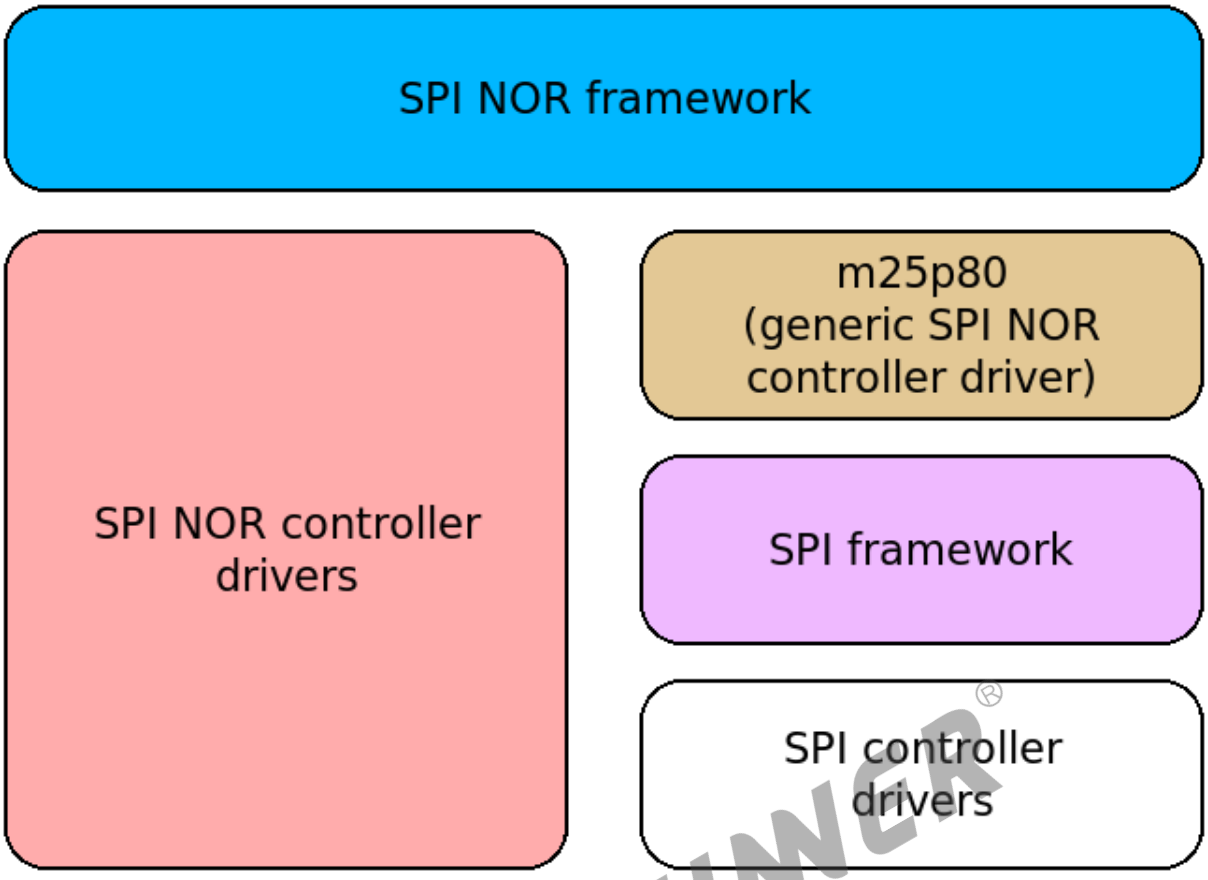


图 2-2: SPINOR 软件框架

SPI NOR Framework：这层主要是处理不同厂家的 NOR 物理特色差异，初始化 SPINOR 的工作状态，如工作线宽（1 线、2 线、4 线、8 线）、有效地址位（16M 以上的 NOR 需要使用 4 地址模式），为上层 MTD 提供读写擦接口。

对应代码目录：drivers/mtd/spi-nor/spi-nor.c

M25P80（generic SPI NOR controller driver）：这层主要对 SPI NOR Framework 层传下来的数据封装成 msg，传递给 SPI framework 层。

对应代码目录：drivers/mtd/devices/m25p80.c

注：linux4.9 后将 m25p80 整合到 spi-nor.c 中

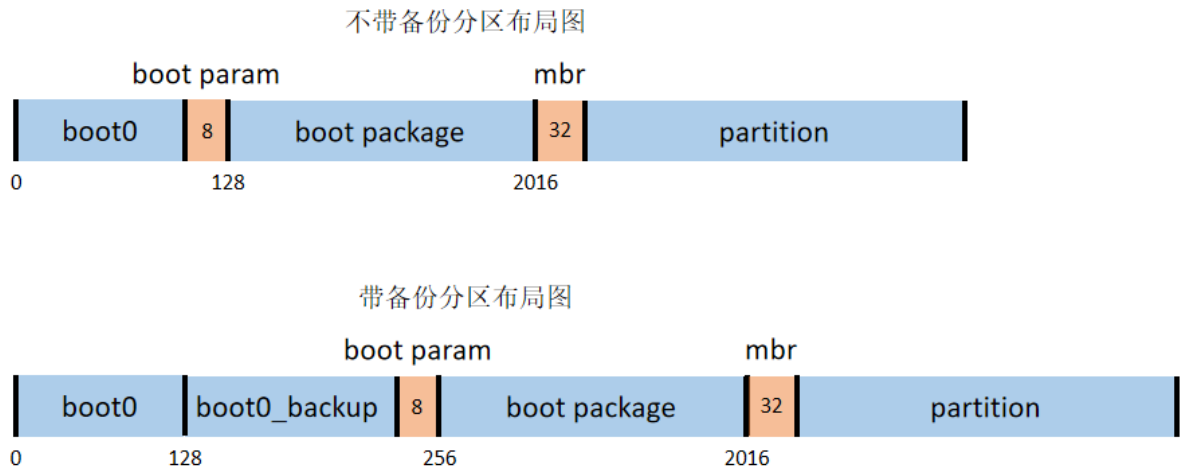
SPI Framework：这层主要是将 msg 加入 ctl 的工作队列中，启动内核线程队列，处理队列中的 msg。

对应代码目录：drivers/spi/spi.c

SPI controller driver：这层初始化 SPI 控制器频率、时钟模式、cs 有效电平、大小端等配置，同时处理上层传下来的 msg，通过 CPU/DMA 方式传输数据到 FIFO，再传输给外设 SPINOR。

对应代码目录：drivers/spi/spi-sunxi.c

2.1.1 存储布局



注:
1. sunxi 平台PhoenixSuit工具还没支持烧备份boot0功能
2. 烧录器方案要用备份功能，需要修改
brandy/brandy-2.0/spl/board/sun8iw21p1/spinor.mk的

CFG_SPINOR_UBOOT_OFFSET = 256

图 2-3: SPINOR 布局

2.2 相关术语介绍

术语	解释说明
Sunxi	指 Allwinner 的一系列 SOC 硬件平台
SPI	Serial Peripheral Interface，同步串行外设接口
NOR Flash	NOR Flash 是一种非易失闪存技术，是 Intel 在 1988 年创建
MTD	MTD(memory technology device 内存技术设备) 是用于访问 memory 设备（ROM、flash）的 Linux 的子系统

2.3 模块配置介绍

2.3.1 longan 的配置和打包

```
./build.sh config
All available platform:
  0. android
  1. linux
Choice [linux]: 1
...                //配置根据需求选择
All available flash:   //flash类型，只区分nor和非nor方案，Android方案无此选项，默认非nor
  0. default
  1. nor
Choice [default]: 1
```

1. 打包普通固件

```
#./build.sh clean
#./build.sh
#./build.sh pack
```

2. 打包卡打印固件

```
#./build.sh clean
#./build.sh
#./build.sh pack_debug
```

在配置的过程中会把平台目录下的 BoardConfig.mk 的信息拷贝到.buildconfig 中。

2.3.2 sys_config 配置

SPINOR 的 boot0 启动阶段，部分参数是从 boot0 头部获取的，而这些参数是我们在打包固件时，通过工具 update_boot0 将 sys_config.fex 中 [spinor_para]，更新到 boot0 头部的，sys_config.fex 的 [spinor_para] 配置参数如下：

```
[spinor_para]
;readcmd         =0x6b
;read_mode       =4
;write_mode      =4
;flash_size      =16
;delay_cycle     =1
;frequency       =100000000
;erase_size      =64
;lock_flag       =0
;sample_delay    =0
;sample_mode     =2

spi_sclk        = port:PC00<4><0><2><default>
spi_cs          = port:PC01<4><1><2><default>
```

```
spi0_mosi       = port:PC02<4><0><2><default>
spi0_miso       = port:PC03<4><0><2><default>
spi0_wp         = port:PC04<4><0><2><default>
spi0_hold       = port:PC05<4><0><2><default>
```

其中：

readcmd：boot0 用于读取数据的命令，不填默认用 uboot 传递过来的 readcmd
read_mode、write_mode：boot0 的工作线宽（1、2、4），不填默认更加 readcmd 决定线宽
flash_size：flash 的大小
delay_cycle：boot0 的采样延时配置，大于 60MHZ 配置为 1，小于 24MHZ 配置为 2，大于 24MHZ 小于 60HZ 配置为 3
frequency：boot0 的 SPI 工作频率，不填使用默认值 50M
erase_size：boot0 的擦除单位
lock_flag：锁功能是否打开
sample_delay：boot0 的细调采样的采样延时，uboot、kernel 也会用到，默认不填等于 0xaaaaffff
sample_mode：boot0 的细调采样的采样模式，uboot、kernel 也会用到，默认不填等于 0xaaaaffff
spi_sclk、spi_cs、spi0_mosi、spi0_miso、spi0_wp 和 **spi0_hold** 用于配置相应的 GPIO。

2.3.3 UBOOT 配置

2.3.3.1 uboot-2018

- 进入 Device Drivers

```
Device Drivers ---->
  [*]SPI Suppport ---->
  [*]Sunxi flash support ---->
```

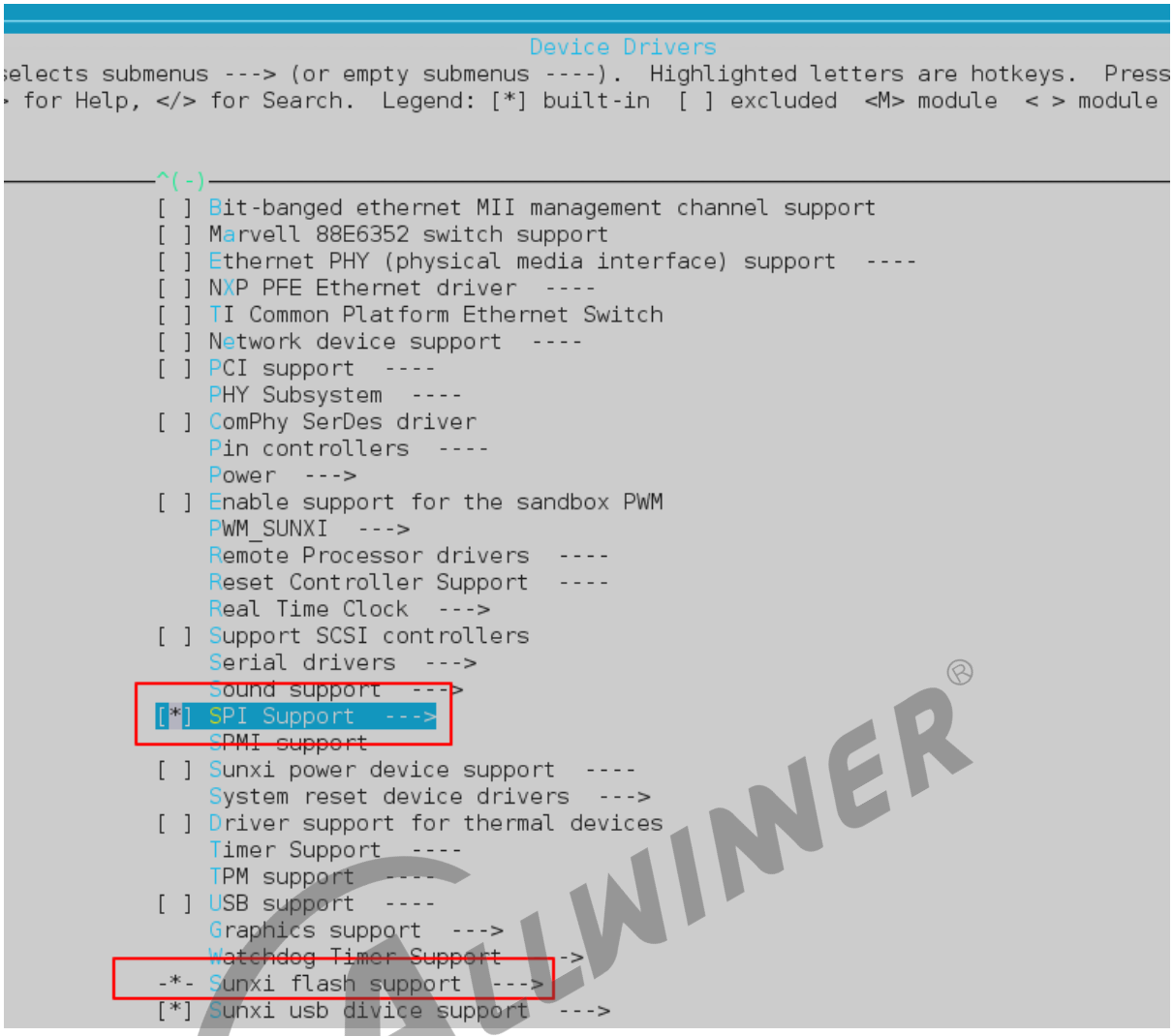


图 2-4: uboot_menuconfig1

- 进入 SPI Support

```
Device Drivers ---->
  [*]SPI Suppport ---->
    [*]Sunxi SPI driver
```

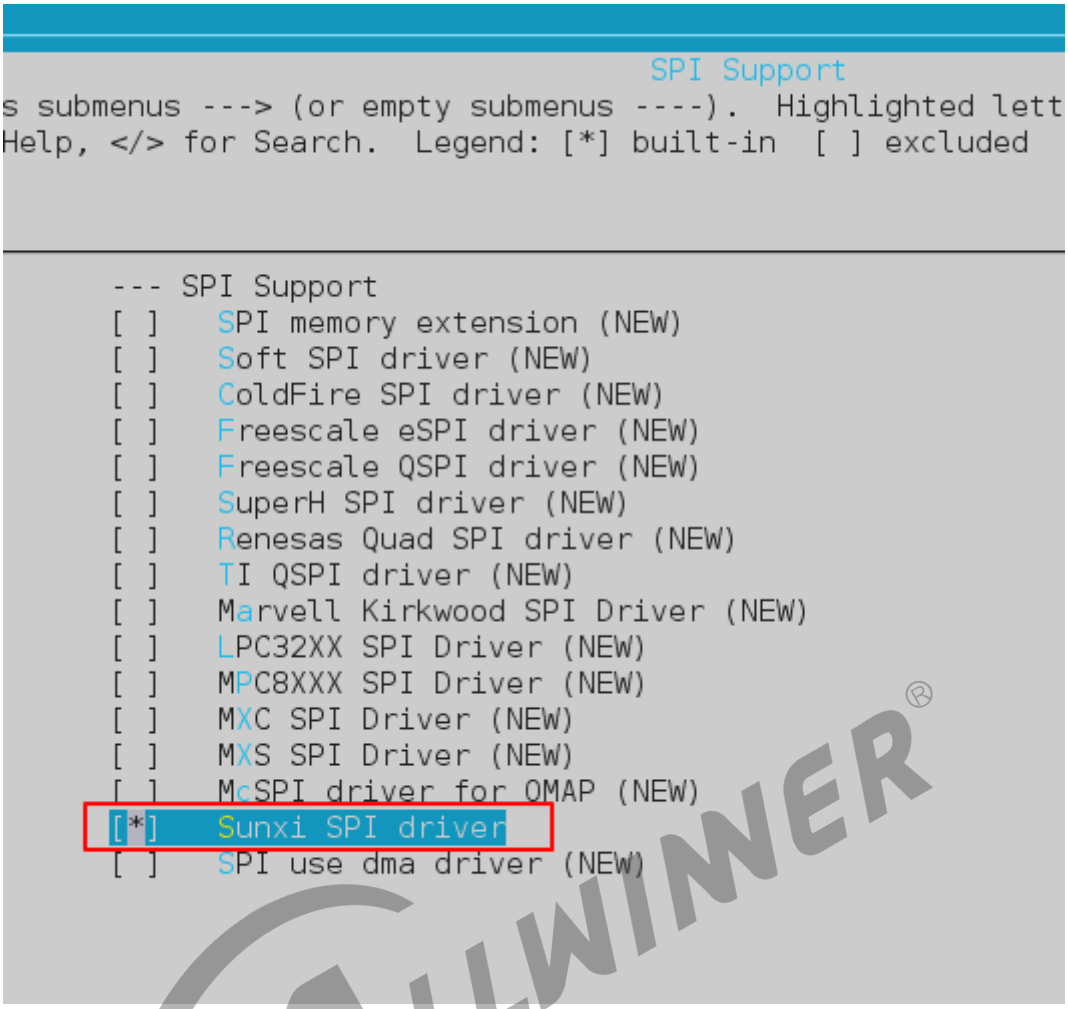


图 2-5: uboot_menuconfig2

- 进入 sunxi_flash_support

```
Device Drivers ---->
  [*]Sunxi flash support ---->
    [*]Support sunxi spinor devices
```

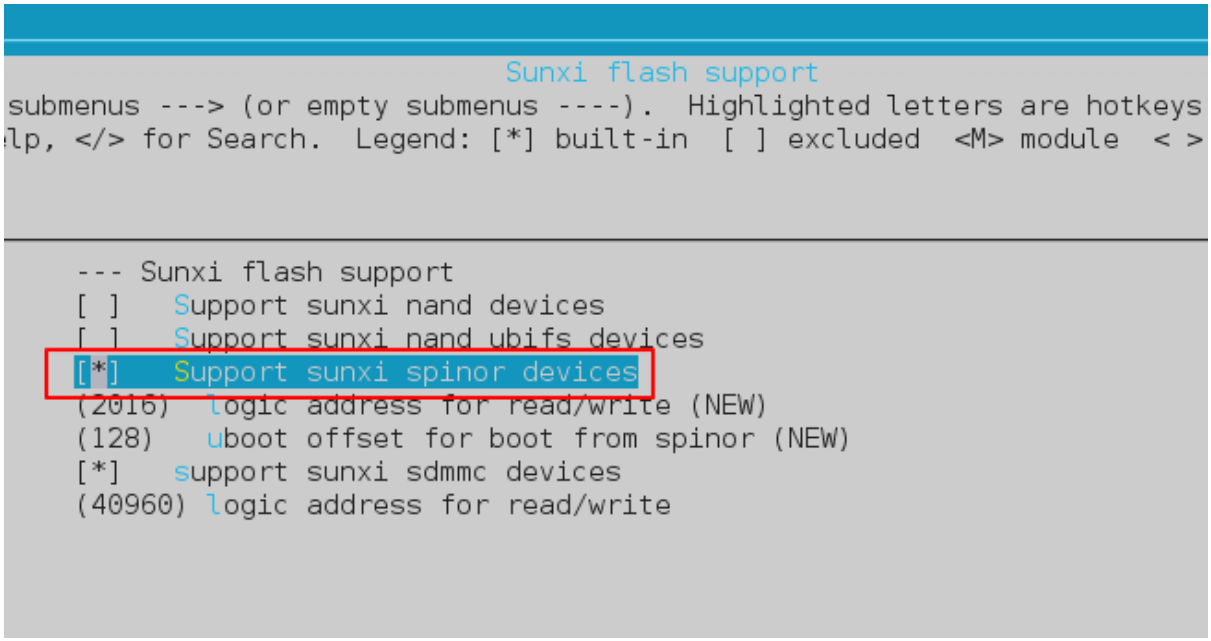


图 2-6: uboot_menuconfig3

2.3.3.2 uboot-2023

首先打开 spi 驱动

```
Allwinner uboot BSP ---> Device Drivers --->
[*] SUNXI SPI driver
```

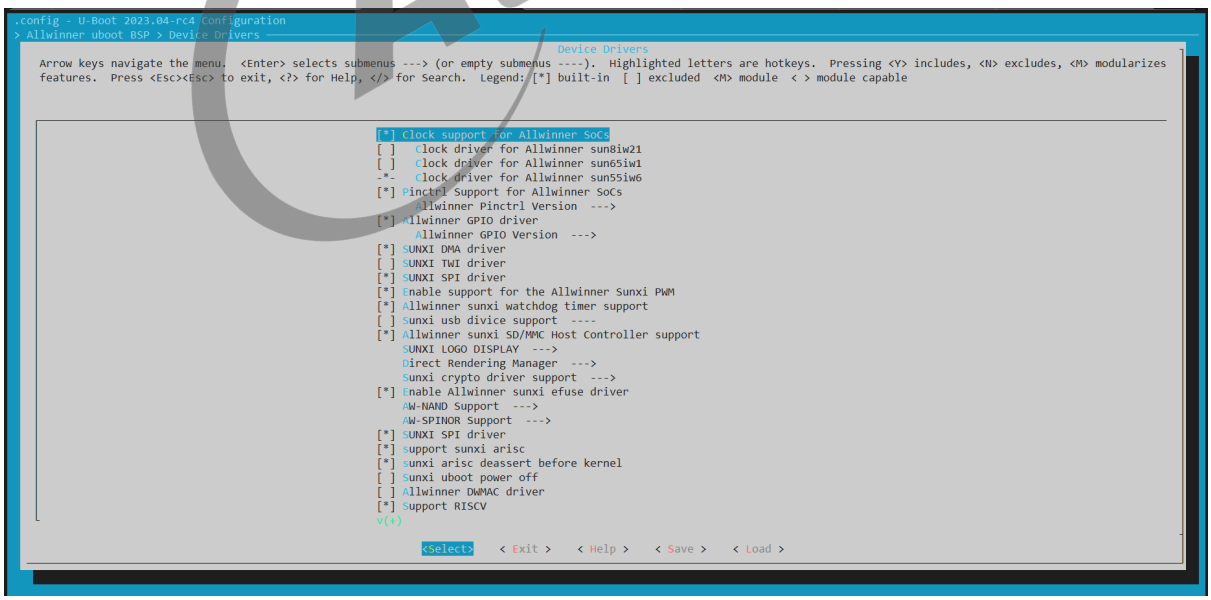


图 2-7

接着打开 uboot-bsp 仓库中的 spinor 配置

```
Allwinner uboot BSP ---> Device Drivers ---> AW-SPINOR Support  --->
[*] Support sunxi spinor devices
```

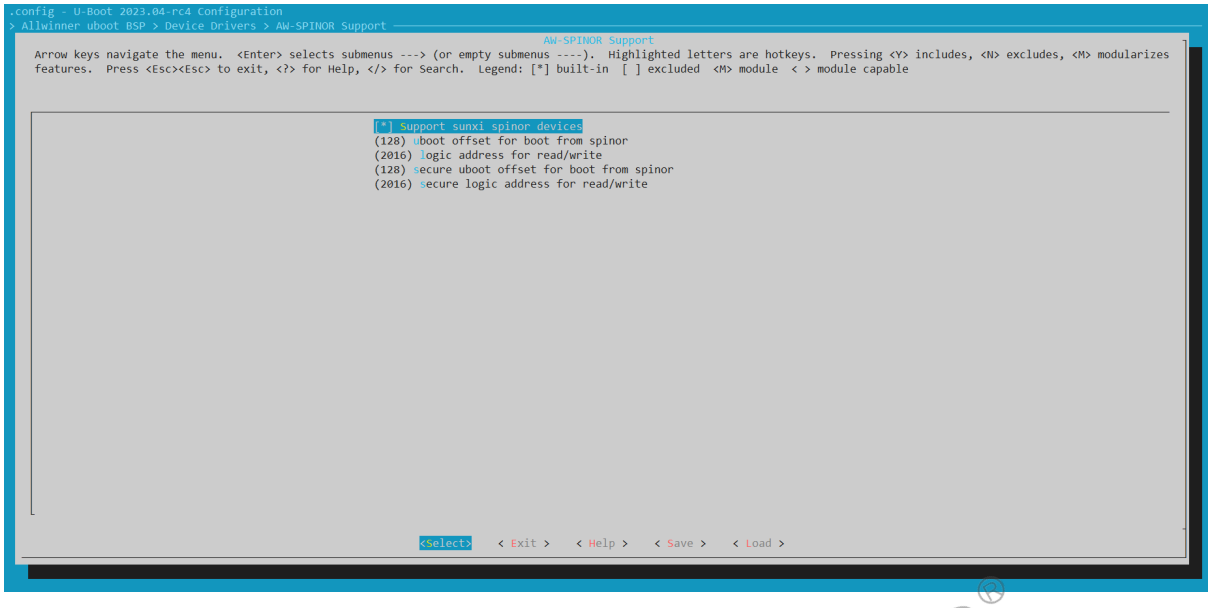


图 2-8

打开支持不同 flah 厂商的 defconfig

```
Device Drivers ---> MTD Support ---> SPI Flash Support --->
[*] Atmel SPI flash support
[*] EON SPI flash support
[*] GigaDevice SPI flash support                    [*] ISSI SPI flash support
                    [*] Macronix SPI flash support                    [*] Spansion
    SPI flash support
[*] STMicro SPI flash support
[*] SST SPI flash support                           [*] Winbond SPI flash support
[*] XMC SPI flash support
[*] XTX SPI flash support
```

2.3.4 KERNEL 配置

2.3.4.1 SPINOR-驱动配置

2.3.4.1.1 linux4.9/5.4

```
#cd kernel/liunx-4.9
#make ARCH=arm menuconfig
```

- 进入 Device Drivers

```
Device Drivers ---->
  <*>Memory Technology Device (MTD) support ---->
  [*]SPI support ---->
```

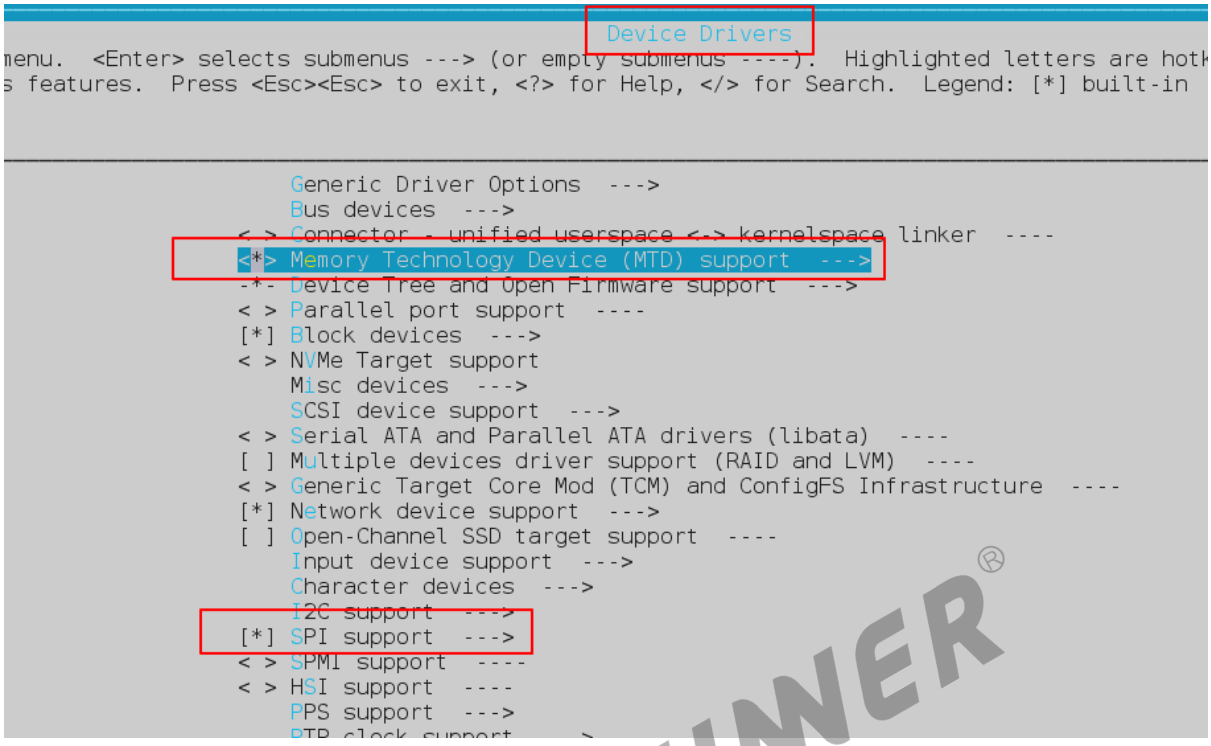


图 2-9: kernel_menuconfig1

- 进入 Menory Technology Device(MTD) support

```
Device Drivers ---->
  <*>Memory Technology Device (MTD) support ---->
    <*>SUNXI partitioning support
    <*>Direct char device access to MTD devices
    <*>Caching block device access to MTD devices
    Self-contained MTD device drivers ---->
    SPI-NOR device support ---->
```

图 2-10: kernel_menuconfig2

- 进入 Self-contained MTD device drivers（5.4 内核不需要选择此项）

```
Device Drivers ---->
  <*>Memory Technology Device (MTD) support ---->
    Self-contained MTD device drivers ---->
      <*>Support most SPI Flash chips (AT16DF, M25P.....)
```

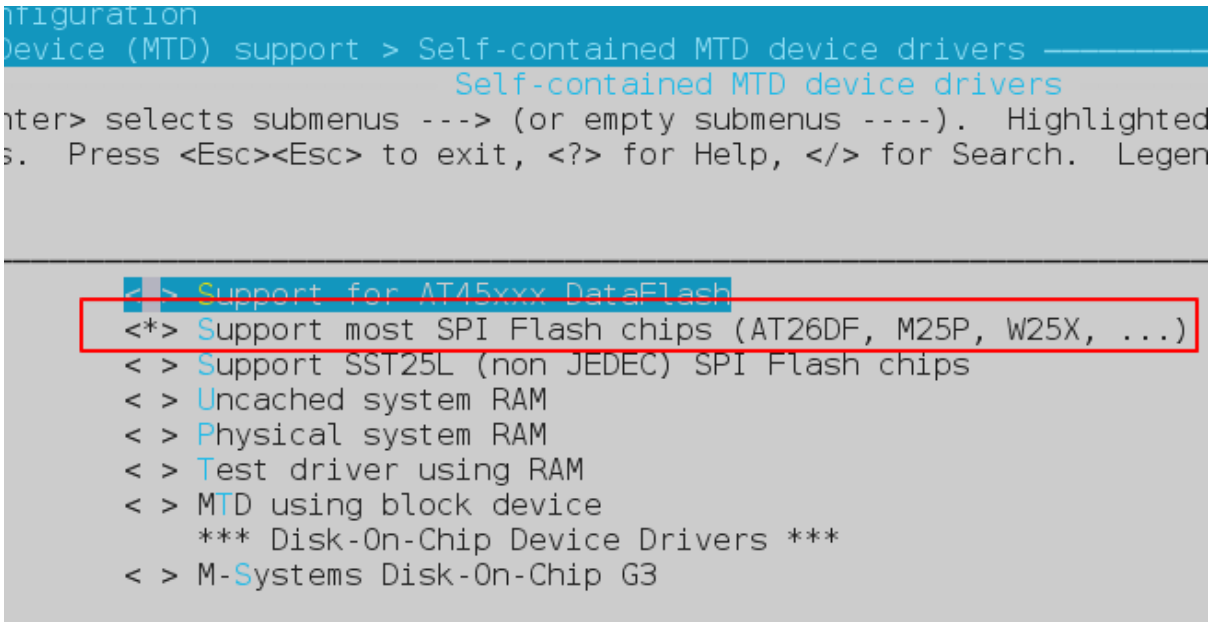


图 2-11: kernel_menuconfig3

Boot opttions ---->

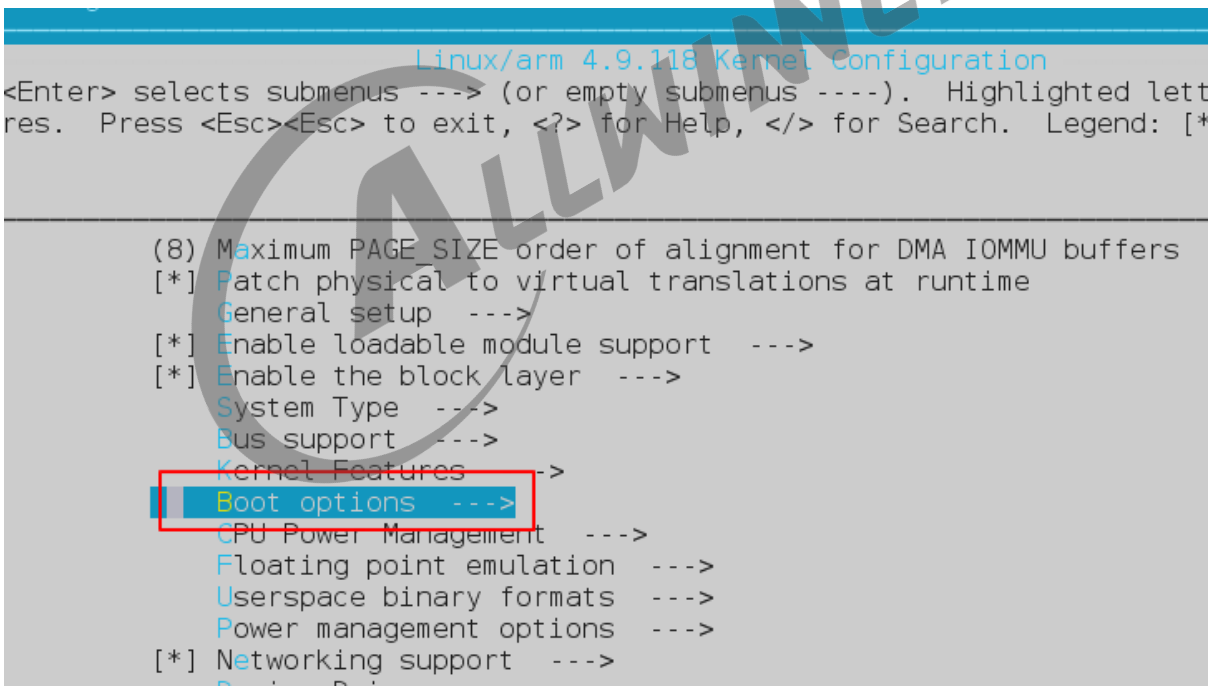


图 2-12: kernel_menuconfig5

- 进入 Boot options

```
Boot opttions ---->
  Kernel command line type ---->
```

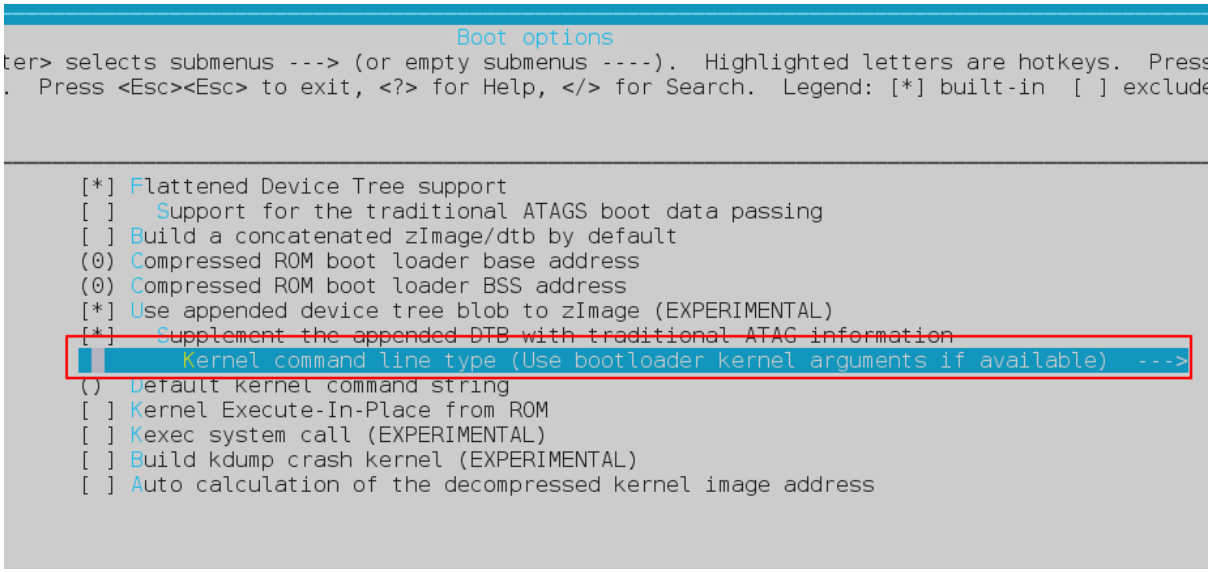


图 2-13: kernel_menuconfig6

- 进入 kernel command line type

```
Boot opttions ---->
  Kernel command line type ---->
    (X)Use bootloade kernel arguments if available
```

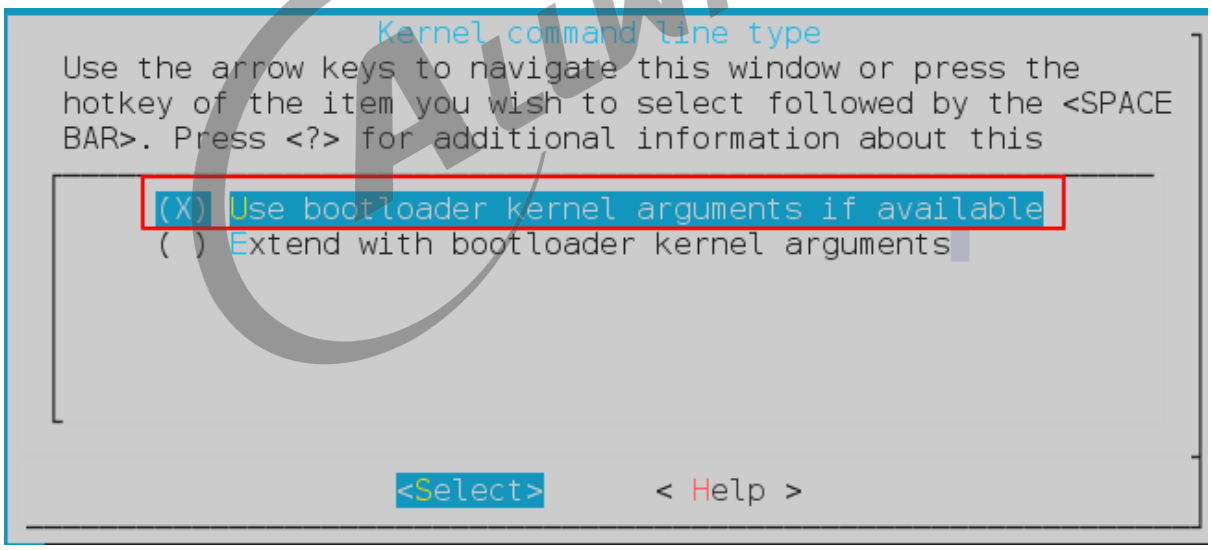


图 2-14: kernel_menuconfig7

2.3.4.1.2 Linux5.4-ansc/5.10/5.15/6.6

```
Allwinner BSP --->Device Drivers --->Memory Technology Device(MTD) support
```

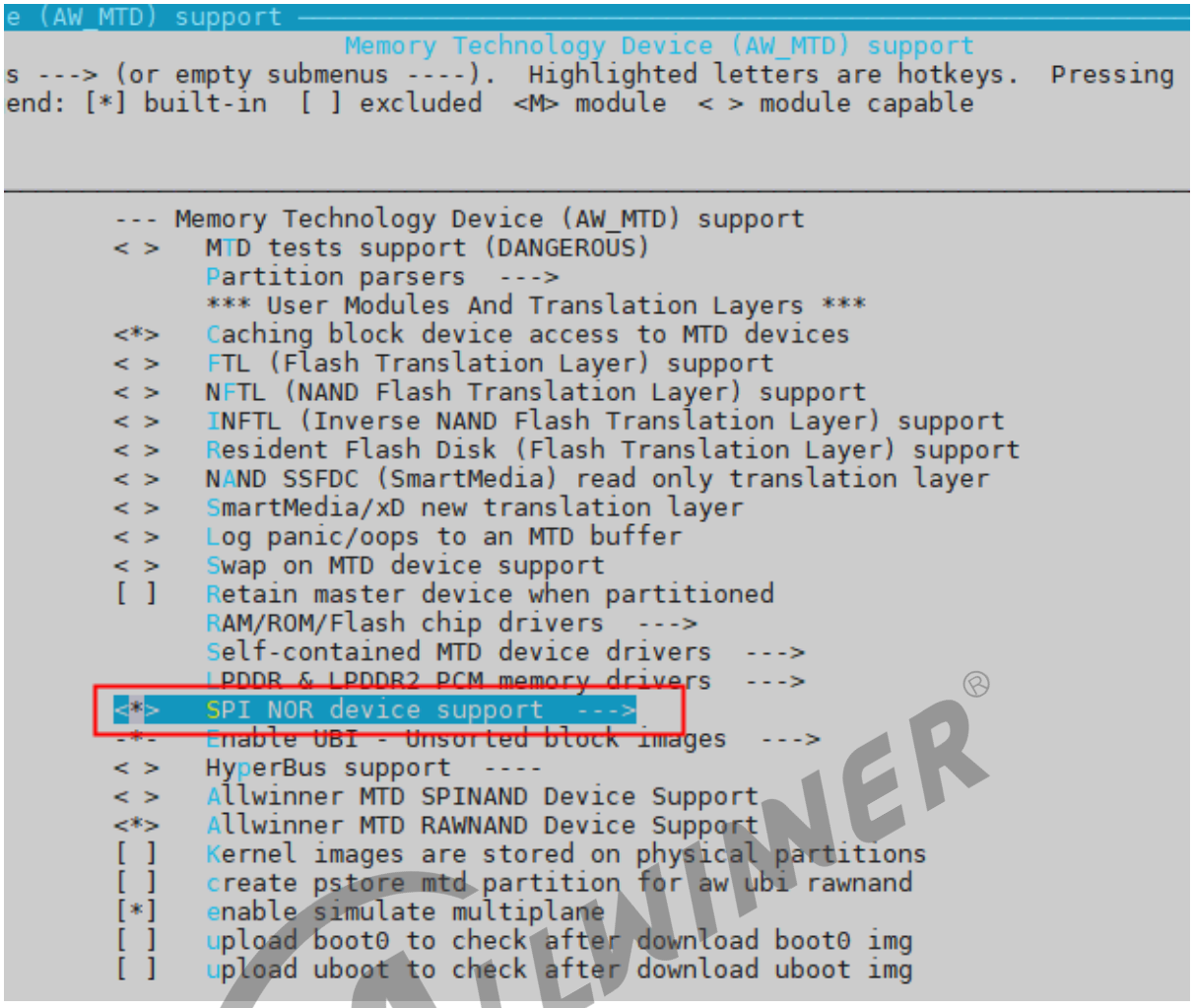


图 2-15: spinor-config

Allwinner BSP --->Device Drivers --->SPI Drivers

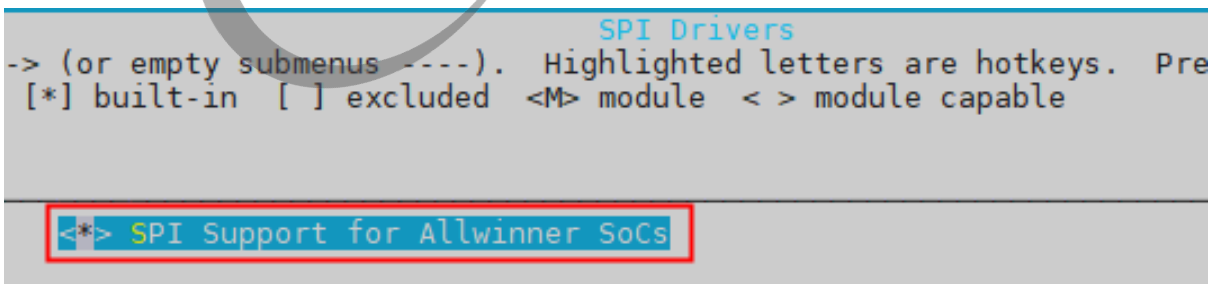


图 2-16: menuconfig-spi

Allwinner BSP --->Device Drivers --->DMA Drivers

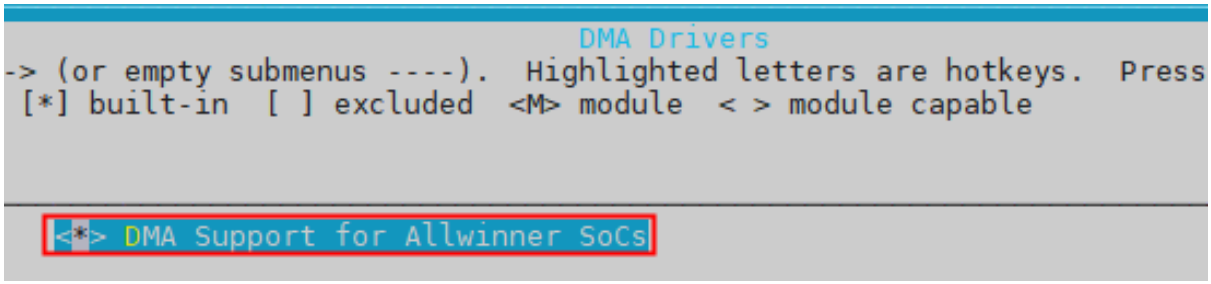


图 2-17: menuconfig-dma

2.3.4.2 文件系统配置

- 进入 File systems

```
File system ---->
  [*]Miscellaneous filesystems ---->
```

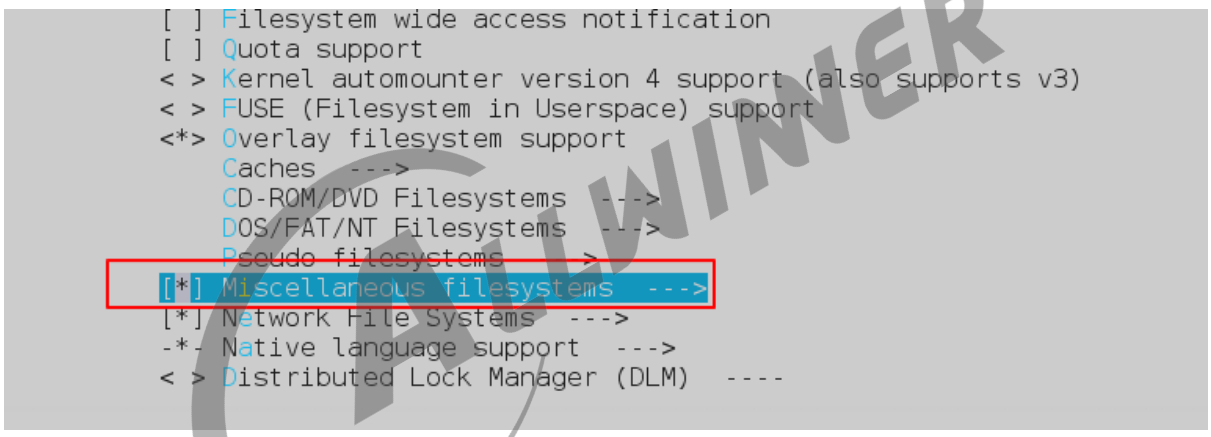


图 2-18: kernel_menuconfig8

- 进入 Miscellaneous filesystems
- Incluede support for ZLIB compressed file systems (NEW)
- Incluede support for LZ4 compressed file systems (NEW)
- Incluede support for LZO compressed file systems (NEW)
- Incluede support for XZ compressed file systems (NEW)

```
File system ---->
  [*]Miscellaneous filesystems ---->
    [*]Incluede support for XZ compressed file systems (NEW)(压缩方式选择如下)
```

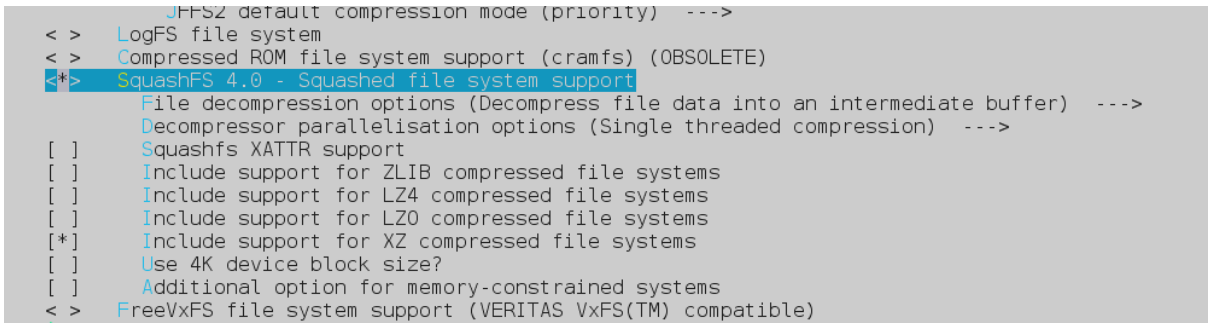


图 2-19: kernel_menuconfig9

以上的压缩方式（ZLIB/LZ4/LZO/XZ）具体选择哪一种需要根据 longan/build/mkcmd.sh 中如下代码使用的压缩方式而定，如下代码使用的是 gzip 压缩方式，则内核 File systems 中配置需选择 LZO 压缩方式，若使用的是 xz, 则需选择 XZ 压缩方式。

```
${ROOTFS} ${LICHEE_PLAT_OUT}/rootfs.squashfs -root-owned -no-progress -comp gzip -noappend
```

2.4 源码目录介绍

2.4.1 UBOOT 源码目录

```
\u-boot-2018\drivers
 ├────sunxi_flash    ---sunxi_flash的初始化/退出/读/写/擦除等flash接口
 ├──mmc          ---mmc接口代码
 ├──nand         ---nand接口代码
 ├──spinor       ---spi nor接口代码
 ├──sunxi_flash.c    ---sunxi_flash操作接口
 └────其他
 ├──── spi        --sunxi_spi的接口代码
 ├──sunxi_spi.c     ---具体代码的实现
 ├────mtd
 ├──spi
 ├──sf_probe.c      ---nand接口代码
 ├──spinor       ---spi nor接口代码
 ├──sunxi_flash.c    ---sunxi_flash操作接口
 └──── makefile     ---编译文件
```

2.4.2 KERNEL 源码目录

2.4.2.1 Linux4.9/5.4

```
\longan\kernel\linux-4.9\drivers\
├──── mtd
├──spi-nor
├──spi-nor.c    ---spi nor驱动代码
└────其他
├──── spi       --spi的接口代码
└──── makefile   ---编译文件
```

2.4.2.2 Linux5.10/5.15/6.6

```
bsp/drivers/mtd/spi-nor/
├──── atmel.c
├──── catalyst.c
├──── controllers
│    ├──── aspeed-smc.c
│    ├──── hisi-sfc.c
│    ├──── intel-spi.c
│    ├──── intel-spi.h
│    ├──── intel-spi-pci.c
│    ├──── intel-spi-platform.c
│    ├──── Kconfig
│    ├──── Makefile
│    └──── nxp-spifi.c
├──── core.c
├──── core.h
├──── eon.c
├──── esmt.c
├──── everspin.c
├──── fujitsu.c
├──── gigadevice.c
├──── intel.c
├──── issi.c
├──── Kconfig
├──── macronix.c
├──── Makefile
├──── micron-st.c
├──── sfdp.c
├──── sfdp.h
├──── spansion.c
├──── sst.c
├──── winbond.c
├──── xilinx.c
└──── xmc.c
```

3 接口描述

3.1 驱动物理层接口

3.1.1 spi_nor_erase

```
static int spi_nor_erase(struct mtd_info *mtd, struct erase_info *instr)
```

description： mtd erase interface

@mtd： MTD device structure

@instr： erase operation descrition structure

return： success return 0，fail return fail code

3.1.2 spi_nor_read

```
static int spi_nor_read(struct mtd_info *mtd, loff_t from, size_t len,
                        size_t *retlen, u_char *buf)
```

description： mtd read interface

@mtd： MTD device structure

@from： offset to read from MTD device

@len： data len

@retlen： had read data len

@buf： data buffer

return： success return max_bitflips，fail return fail code

3.1.3 spi_nor_write

```
static int spi_nor_write(struct mtd_info *mtd, loff_t to, size_t len,
                         size_t *retlen, const u_char *buf)
```

description： mtd write data interface

@to： offset to MTD device

@len： want write data len

@retlen： return the writen len

@buf： data buffer

return： success return 0, fail return code fail

3.1.4 spi_nor_lock

```
static int spi_nor_lock(struct mtd_info *mtd, loff_t ofs, uint64_t len)
```

description： check block is badblock or not

@mtd： MTD device structure

@ofs: offset the mtd device start (align to simu block size)

@len： The length of the operating

return： success return 0, fail return code fail

3.1.5 spi_nor_unlock

```
static int spi_nor_unlock(struct mtd_info *mtd, loff_t ofs, uint64_t len)
```

description： check block is badblock or not

@mtd： MTD device structure

@ofs: offset the mtd device start (align to simu block size)

@len： The length of the operating

return： success return 0, fail return code fail

3.1.6 spi_nor_is_locked

```
static int spi_nor_is_locked(struct mtd_info *mtd, loff_t ofs, uint64_t len)
```

description： check block is badblock or not

@mtd： MTD device structure

@ofs: offset the mtd device start (align to simu block size)

@len： The length of the operating

return： Is lock return 1, else return 0

3.1.7 spi_nor_has_lock_erase

```
static int spi_nor_has_lock_erase(struct mtd_info *mtd, struct erase_info *instr)
```

description： mtd has lock erase interface，First unlock to operate space, after the completion of the flash lock up

@mtd： MTD device structure

@instr： erase operation descrition structure

return： success return 0，fail return fail code

3.1.8 spi_nor_has_lock_write

```
static int spi_nor_has_lock_write(struct mtd_info *mtd, loff_t to, size_t len,
                                  size_t *retlen, const u_char *buf)
```

description： mtd has lock write data interface，First unlock to operate space, after the completion of the flash lock up

@to： offset to MTD device

@len： want write data len

@retlen： return the writen len

@buf： data buffer

return： success return 0, fail return code fail

3.2 Uboot 应用接口

3.2.1 sunxi_flash_spinor_probe

```
static int sunxi_flash_spinor_probe(void)
```

description：SPINOR initialization，Set the storage type。

return：zero on success, else a negative error code.

3.2.2 sunxi_flash_spinor_init

```
static int sunxi_flash_spinor_init(int boot_mode, int res)
```

description：SPINOR initialization。

@boot_mode：Working mode

@res：The default is 0

return：zero on success, else a negative error code.

3.2.3 sunxi_flash_spinor_exit

```
int sunxi_flash_spinor_exit(void)
```

description：Release registration is a resource for applications.

return：zero on success, else a negative error code.

3.2.4 sunxi_flash_spinor_write

```
static int sunxi_flash_spinor_write(uint start_block, uint nblock, void *buffer)
```

description：mtd write data interface.

@start_block：want write start sector

@nblock：want write sectorcount

@buffer：data buffer

return：zero on success, else a negative error code.

3.2.5 sunxi_flash_spinor_write

```
static int sunxi_flash_spinor_write(uint start_block, uint nblock, void *buffer)
```

description： mtd readdata interface.

@start_block： want read start sector

@nblock： want read sector count

@buffer： data buffer

return： zero on success, else a negative error code.

3.2.6 sunxi_flash_spinor_erase

```
static int sunxi_flash_spinor_erase(int erase, void *mbr_buffer)
```

description： erase boot || partition data.

@erase： erase flag

@buffer： The default is NULL

return： zero on success, else a negative error code.

3.2.7 sunxi_flash_spinor_force_erase

```
int sunxi_flash_spinor_force_erase(void)
```

description： erase boot & partition data.

return： zero on success, else a negative error code.

3.2.8 sunxi_flash_spinor_flush

```
int sunxi_flash_spinor_flush(void)
```

description： Flush physical cache data to flash.

return： zero on success, else a negative error code.

3.2.9 sunxi_flash_spinor_download_spl

```
static int sunxi_flash_spinor_download_spl(unsigned char *buf, int len, unsigned int ext)
```

description： write boot0.

@buf： boot0 data buffer

@len： boot0 data len

@ext： storage type

return： zero on success, else a negative error code.

3.2.10 sunxi_flash_spinor_download_toc

```
static int sunxi_flash_spinor_download_toc(unsigned char *buf, int len,  unsigned int ext)
```

description： write uboot.

@buf： uboot data buffer

@len： uboot data len

@ext： storage type

return： zero on success, else a negative error code.

4 功能开发

4.1 功能概述

主要介绍 boot、内核态、用户态访问 flash 时的流程说明。

4.2 开发流程

4.2.1 BOOT0 读取数据

```
/* 头文件依赖 */
#include <arch/spinor.h>

int spinor_read(uint start, uint sector_cnt, void *buffer)
```

参数说明：

start：起始扇区，一个扇区等于 512byte

V853 快起方案我们约定好数据偏移 112 扇区（这个需要注意，要求 boot0 size 要小于 112 个扇区大小，即 56k）

sector_cnt：要读取的扇区数

V853 快起方案预留 4 个扇区给 ISP

buffer：存放数据的缓存

4.2.2 uboot shell 使用

mem_addr：内存地址，0x40000000 之后可以随便选取如：0x45000000，0x46000000

part_name：分区文件名，boot-resource、env、boot、rootfs

size：可以省略，默认读取整个分区文件

1. sunxi_flash read [size] 读取 flash 中的分区文件到内存中

例：使用 sunxi_flash read 命令将 boot 分区读入到 0x49000000 中，然后使用 md 命令读取 0x49000000 中的内容。

```
=> sunxi_flash read  0x49000000 boot
partinfo: name boot, start 0x2620, size 0x3c80
=> md 0x49000000
49000000: 52444e41 2144494f 003b52b0 40008000    ANDROID!.R;....@
49000010: 003cfac7 41000000 00000000 40f00000    ..<....A.......@
49000020: 40000100 00000800 00000000 00000000    ...@............
49000030: 386e7573 72615f69 0000006d 00000000    sun8i_arm.......
49000040: 00000000 00000000 00000000 00000000    ................
49000050: 00000000 00000000 00000000 00000000    ................
```

图 4-1: sunxi flash read

验证方法：

1. 0x49000000 读入前与读入后数据有没有发生变化
2. 在 **out/pack_out** 目录下找到对应的分区文件，使用 **hexdump -Cv boot.fex -n 500** 命令输出分区文件的数据，对比一致即读入成功。

```
guanyanfei@AwExdroid100:~/workspace/longanV853/out/pack_out$ hexdump -Cv boot.fex -n 500
00000000  41 4e 44 52 4f 49 44 21  b0 52 3b 00 00 80 00 40  |ANDROID!.R;....@|
00000010  c7 fa 3c 00 00 00 00 41  00 00 00 00 00 00 f0 40  |..<....A.......@|
00000020  00 01 00 40 00 08 00 00  00 00 00 00 00 00 00 00  |...@............|
00000030  73 75 6e 38 69 5f 61 72  6d 00 00 00 00 00 00 00  |sun8i_arm.......|
00000040  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000050  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000060  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
00000070  00 00 00 00 00 00 00 00  00 00 00 00 00 00 00 00  |................|
```

图 4-2: hexdump

2. sunxi_flash write [size] 将内存中的数据，写入到分区中

例：

1）使用 mm 命令修改内存内容

```
=> mm 0x44000000          修改内存中数据
44000000: fedcba98 ? 123
44000004: fedcba99 ? 456
44000008: fedcba9a ? 789
4400000c: fedcba9b ? ?    ? 退出编辑
=> md 0x44000000          查看内存
44000000: 00000123 00000456 00000789 fedcba9b    #...V...........  修改后数据
44000010: fedcba9c fedcba9d fedcba9e fedcba9f    ................
44000020: fedcbaa0 fedcbaa1 fedcbaa2 fedcbaa3    ................
44000030: fedcbaa4 fedcbaa5 fedcbaa6 fedcbaa7    ................
```

图 4-3: mm - md

2）使用 sunxi_flash write 0x44000000 env 将内存中的数据写入 env 分区

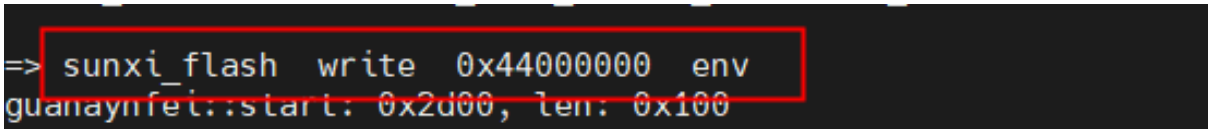


图 4-4: sunxi flash write

3）重新将 env 分区读入内存中，对比一致表示写入成功

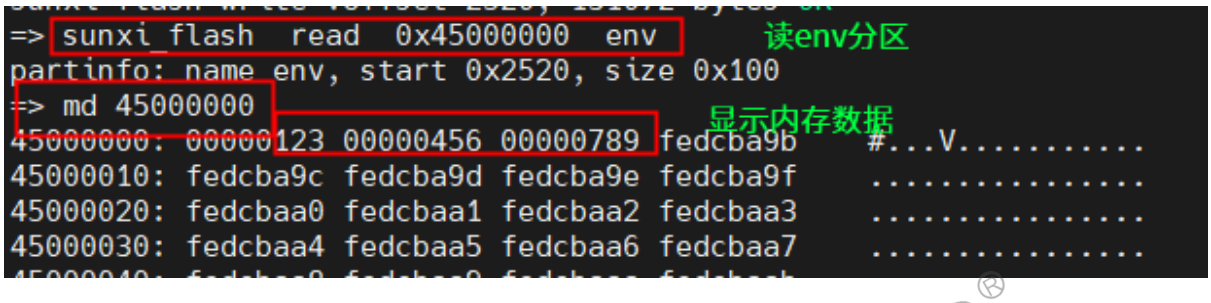


图 4-5: sunxi flash read2

4.2.3 内核态访问 flash

代码示例

```
/* 头文件依赖 */
#include <linux/fs.h>
#include <linux/uaccess.h>

char part_name = "/dev/mtd0";
struct file *fp;
mm_segment_t fs;

fp = filp_open(part_name, 0_RDONLT, 0444);
if (IS_ERR(fp)) {
   printk("open %s error\n", part_name);
   return -1;
}

fs =get_fs();
set_fs(KERNEL_DS);

ret = vfs_read(fp, buf, len, pos);
ret = vfs_write(fp, buf, len, pos);

filp_close(fp, NULL);
set_fs(fs);
```

下面注意说明一下 vfs_write 接口

```
vfs_write(struct file *file, const char __user *buf, size_t count, loff_t *pos)
```

file：传入 filp_open 的返回值

buf：要写入的数据 buf

count：要写入的数据大小，byte 为单位

pos：要写入的数据偏移，byte 为单位

4.2.4 用户态访问 flash

代码示例

```
#include <stdio.h>
#include <fcntl.h>
#include <unistd.h>
#include <string.h>
#include <sys/ioctl.h>
#include <linux/mtd/mtd-user.h>

#define DEVICE "/dev/mtdblock0" // 替换为实际的 mtdblock 设备节点
#define BUFFER_SIZE 4096

int main() {
    int fd;
    char buffer[BUFFER_SIZE];

    // 打开 mtdblock 设备
    fd = open(DEVICE, O_RDWR);
    if (fd < 0) {
        perror("Failed to open mtdblock device");
        return 1;
    }

    // 读取数据
    ssize_t bytes_read = read(fd, buffer, BUFFER_SIZE);
    if (bytes_read < 0) {
        perror("Failed to read from mtdblock device");
        close(fd);
        return 1;
    }
    printf("Read %zd bytes from %s\n", bytes_read, DEVICE);

    // 打印读取的数据
    for (ssize_t i = 0; i < bytes_read; i++) {
        printf("%02x ", (unsigned char)buffer[i]);
    }
    printf("\n");

    // 写入数据
    const char *data_to_write = "Hello MTD!";
    ssize_t bytes_written = write(fd, data_to_write, strlen(data_to_write));
    if (bytes_written < 0) {
        perror("Failed to write to mtdblock device");
        close(fd);
        return 1;
    }
    printf("Wrote %zd bytes to %s\n", bytes_written, DEVICE);
```

```
    // 关闭设备
    close(fd);
    return 0;
}
```

参数介绍

- **DEVICE**: 设备节点的路径，例如 /dev/mtdblock0。根据系统中的实际设备进行修改。
- **open()**: 用于打开设备文件。参数 O_RDWR 表示以读写方式打开设备。
- **read(fd, buffer, BUFFER_SIZE)**:
- **fd**: open() 返回的文件描述符。
- **buffer**: 存储读取数据的缓冲区。
- **BUFFER_SIZE**: 要读取的字节数。
- **write(fd, data_to_write, strlen(data_to_write))**:
- **fd**: open() 返回的文件描述符。
- **data_to_write**: 要写入的数据。
- **strlen(data_to_write)**: 要写入的字节数。

4.3 注意事项

1. **设备节点**: 确保使用正确的设备节点路径，设备节点可能因系统配置而异。
2. **权限**: 确保你有足够的权限访问 mtdblock 设备，通常需要 root 权限。
3. **数据对齐**: MTD 设备的读写通常需要特定的对齐，具体取决于硬件特性。
4. **数据持久性**: 在写入数据之前，了解设备的擦除块和页面大小，以避免损坏数据。
5. **错误处理**: 在实际应用中，建议在每个系统调用后添加错误处理代码，以处理可能的失败情况。

5 日志分析

```
sunxi:sunxi_spif-44f00000.spif:[INFO]: sample_mode:0 sample_delay:16
sunxi:sunxi_spif-44f00000.spif:[INFO]: [spi-flash0] mclk 24000000
sunxi:sunxi_spif-44f00000.spif:[INFO]: [spi-flash0] working clk 100000000
sunxi_spif 44f00000.spif: xt25p1288 (16384 Kbytes) IO DTR
8 sunxipart partitions found on MTD device spif
Creating 8 MTD partitions on "spif":
0x000000000000-0x000000100000 : "uboot"
0x000000100000-0x000000110000 : "boot-resource"
0x000000110000-0x000000130000 : "env"
0x000000130000-0x000000150000 : "env-redund"
0x000000150000-0x000000360000 : "boot"
0x000000360000-0x000000680000 : "rootfs"
0x000000680000-0x000000700000 : "rootfs_data"
0x000000700000-0x000001000000 : "UDISK"
sunxi:sunxi_spif-44f00000.spif:[INFO]: probe succeed (Version 1.0.2)
```

SPINOR 驱动成功加载时，会有以上日志，下面介绍一些关键信息：

sample_mode:0 sample_delay:16：代码 spiflash 控制器的采样模式，由于电路的传输延时和 nor 设备接受命令后的响应延时，我们在读取数据时，发送完成命令后，需要 delay 一段时间，再发送 clock 进行数据读出，确保数据的准确性，启动对应的采样模式（sample_mode）和采样延时（sample_delay），要烧写时 try 出来，然后启动更新到设备树的。

working clk 100000000：spiflash 的工作时钟是 100MHz

xt25p1288 (16384 Kbytes) IO DTR：型号是 xt25p1288，大小 16M，支持 IO、DTR 功能

8 sunxipart partitions：一个 8 个 mtd 分区，对应的起始地址、结束地址和分区名字

```
  起始地址      结束地址      分区名
0x000000000000-0x000000100000 : "uboot"
0x000000100000-0x000000110000 : "boot-resource"
0x000000110000-0x000000130000 : "env"
0x000000130000-0x000000150000 : "env-redund"
0x000000150000-0x000000360000 : "boot"
0x000000360000-0x000000680000 : "rootfs"
0x000000680000-0x000000700000 : "rootfs_data"
0x000000700000-0x000001000000 : "UDISK"
```

6 调试方法

6.1 调试节点

查看 spi flash 控制器相关信息

```
cat /sys/devices/platform/soc@2002000/44f00000.spif/info
IP Version = V10002
pdev->id  = 0
pdev->name = 44f00000.spif
pdev->num_resources = 2
sspi->base_addr = 0xa0478000
sspi->irq = 143
sspi->data->bus_num  = 0
sspi->data->chip_select = 0
sspi->data->min_speed_hz  = 25000000
sspi->data->max_speed_hz  = 100000000
sspi->sample_mode  = 0x0
sspi->sample_delay = 0x18
```

IP Version：控制器版本

sspi->data->chip_select：片选

sspi->data->min_speed_hz：最小工作频率，未升频前的初始频率

sspi->data->max_speed_hz：最大工作频率

查看 spi flash 控制器寄存器信息

```
cat /sys/devices/platform/soc@2002000/44f00000.spif/dump
[  242.135910] spif->base_addr = 0x(ptrval), the SPIF control register:
[  242.135910] [VER] 0x00 = 0x00010002, [GC]  0x04 = 0x00000101, [GCA] 0x08 = 0x00000000
[  242.135910] [TCR] 0x0c = 0x00100058, [TDS] 0x10 = 0x00002018, [INT] 0x14 = 0x00000000
[  242.135910] [STA] 0x18 = 0x00000010, [CSD] 0x1c = 0x00050606, [PHC] 0x20 = 0x10000000
[  242.135910] [TCF] 0x24 = 0x00000000, [TCS] 0x28 = 0x06000000, [TNM] 0x2c = 0x10000000
[  242.135910] [PSR] 0x30 = 0x00000000, [PSA] 0x34 = 0x00000000, [PEA] 0x38 = 0x00000000
[  242.135910] [PMA] 0x3c = 0x00000000, [DMA] 0x40 = 0x00000200, [DSC] 0x44 = 0x20d92800
[  242.135910] [DFT] 0x48 = 0x00000600, [CFT] 0x4c = 0x64106410, [CFS] 0x50 = 0x00000000
[  242.135910] [BAT] 0x54 = 0x000000a0, [BAC] 0x58 = 0x00000000, [TB]  0x5c = 0x00000000
[  242.135910] [RB]  0x60 = 0x00000000
```

查看 spinor flash 相关工作信息

```
cat /sys/devices/platform/soc@2002000/44f00000.spif/spinor-info
flash_size:16384 Kbytes
page_size:256 Byte
addr_width:3
flags:0x0
read_opcode:0xed
```

```
read_proto:0x1010404
read_dummy:7
program_opcode:0x32
write_proto:0x10104
erase_opcode:0xd8
hwcaps mask:0x305d9
```

flash_size：flash size

page_size：page size

addr_width：地址位数

read_opcode：读命令

read_proto：读模式

read_dummy：dummy 数

program_opcode：写命令

write_proto：写模式

erase_opcode：擦除命令

6.2 驱动 debug 信息

驱动路径：bsp/drivers/mtd/spi-nor-xxx/controllers/sunxi-spif.c

打开 debug 打印，SPIF_DEBUG 宏改为 1

```
#define SPIF_DEBUG 1
```

SPIF_DEBUG 为 1 后，设置节点，打印每次传输对应的描述符和寄存器信息

```
echo 1 > /sys/module/sunxi_spif/parameters/spif_debug_mask
```

7 FAQ

7.1 硬件排查

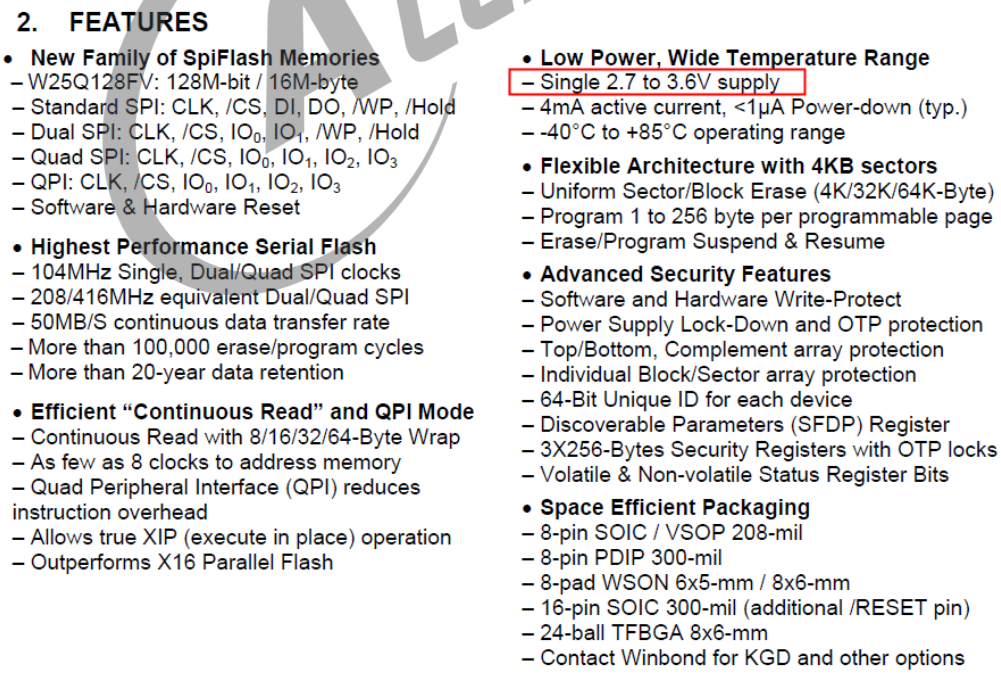
7.1.1 NOR 的 PCB 检查

1、规划性能板的时候，如果要引入 Socket 的，要尽量让 Socket 和贴片靠近，这样信号会好。

2、要检查所有信号管脚的电压（VCCQ）是否为 3.3V；目前 NOR 的样片只有 3.3V 的信号电压，但控制器端的 GPIO 电压是可选择的，需要找准 IO 电压来源。

7.1.2 检查供电电源

供电要求，一般都在 Flash data sheet 最前面的 Features 章节，如下截图中的红框，这里只列举几种，请根据实际情况参考对应 Flash 物料的 data sheet 举一反三。



2. FEATURES

- **New Family of SpiFlash Memories**
  - W25Q128FV: 128M-bit / 16M-byte
  - Standard SPI: CLK, /CS, DI, DO, /WP, /Hold
  - Dual SPI: CLK, /CS, IO_0, IO_1, /WP, /Hold
  - Quad SPI: CLK, /CS, IO_0, IO_1, IO_2, IO_3
  - QPI: CLK, /CS, IO_0, IO_1, IO_2, IO_3
  - Software & Hardware Reset
- **Highest Performance Serial Flash**
  - 104MHz Single, Dual/Quad SPI clocks
  - 208/416MHz equivalent Dual/Quad SPI
  - 50MB/S continuous data transfer rate
  - More than 100,000 erase/program cycles
  - More than 20-year data retention
- **Efficient "Continuous Read" and QPI Mode**
  - Continuous Read with 8/16/32/64-Byte Wrap
  - As few as 8 clocks to address memory
  - Quad Peripheral Interface (QPI) reduces instruction overhead
  - Allows true XIP (execute in place) operation
  - Outperforms X16 Parallel Flash
- **Low Power, Wide Temperature Range**
  - Single 2.7 to 3.6V supply
  - 4mA active current, <1μA Power-down (typ.)
  - -40°C to +85°C operating range
- **Flexible Architecture with 4KB sectors**
  - Uniform Sector/Block Erase (4K/32K/64K-Byte)
  - Program 1 to 256 byte per programmable page
  - Erase/Program Suspend & Resume
- **Advanced Security Features**
  - Software and Hardware Write-Protect
  - Power Supply Lock-Down and OTP protection
  - Top/Bottom, Complement array protection
  - Individual Block/Sector array protection
  - 64-Bit Unique ID for each device
  - Discoverable Parameters (SFDP) Register
  - 3X256-Bytes Security Registers with OTP locks
  - Volatile & Non-volatile Status Register Bits
- **Space Efficient Packaging**
  - 8-pin SOIC / VSOP 208-mil
  - 8-pin PDIP 300-mil
  - 8-pad WSON 6x5-mm / 8x6-mm
  - 16-pin SOIC 300-mil (additional /RESET pin)
  - 24-ball TFBGA 8x6-mm
  - Contact Winbond for KGD and other options

图 7-1: features

FEATURES

- Single power supply operation
  - Full voltage range: 2.7-3.6 volt
- Serial Interface Architecture
  - SPI Compatible: Mode 0 and Mode 3
- 64 M-bit Serial Flash
  - 64 M-bit / 8,192 KByte /32,768 pages
  - 256 bytes per programmable page
- Standard, Dual or Quad SPI
  - Standard SPI: CLK, CS#, DI, DO, WP#, HOLD#/RESET#
  - Dual SPI: CLK, CS#, DQ_0, DQ_1, WP#, HOLD#/RESET#
  - Quad SPI: CLK, CS#, DQ_0, DQ_1, DQ_2, DQ_3
  - Configurable dummy cycle number
- High performance
  - Normal read
    - 83MHz
  - Fast read
    - Standard SPI: 104MHz with 1 dummy bytes
    - Dual SPI: 104MHz with 1 dummy bytes
    - Quad SPI: 104MHz with 3 dummy bytes
- Write Suspend and Write Resume
- Low power consumption
  - 5 mA typical active current
- Software and Hardware Write Protection:
  - Write Protect all or portion of memory via software
  - Enable/Disable protection with WP# pin
- Software and Hardware Reset
- High performance program/erase speed
  - Page program time: 0.5ms typical
  - Sector erase time: 40ms typical
  - Half Block erase time 200ms typical
  - Block erase time 300ms typical
  - Chip erase time: 32 Seconds typical
- Volatile Status Register Bits.
- Lockable 512 byte OTP security sector
- Read Unique ID Number
- Minimum 100K endurance cycle
- Data retention time 20 years
- Package Options
  - 8 pins SOP 200mil body width
  - 8 contact VDFN(5x6mm)
  - 8 pins PDIP
  - 16 pins SOP 300mil body width
  - 24 balls TFBGA (6x8mm)
  - All Pb-free packages are compliant RoHS, Halogen-Free and REACH.

图 7-2: features2

7.1.3 检查硬件板

(1) 检查电路是否虚焊。

(2) 检查电路是否与原理图一致。

(3) 检查设备端 (flash) 是否有问题（更换设备）。

常见的 SPI 接口 NOR 封装类型有 8-PIN 的 SOIC、8-PIN 的 LAND 以及 16-PIN 的 SOIC，如下图所示

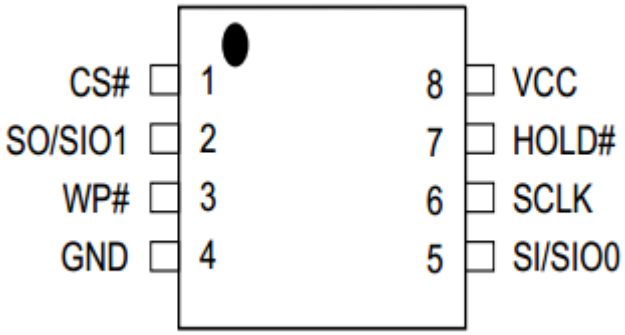


图 7-3: flash 封装类型 _1

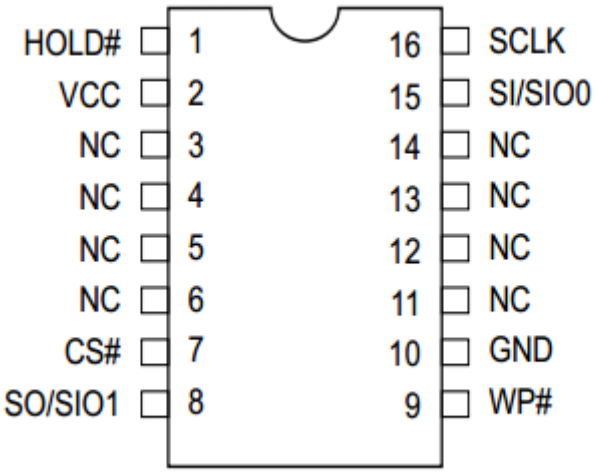


图 7-4: flash 封装类型 _2

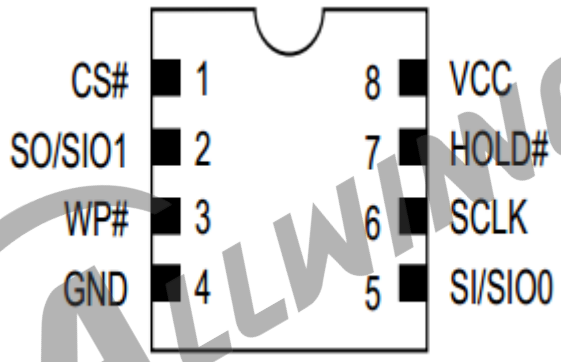


图 7-5: flash 封装类型 _3

不论哪一类型封装，主控端控制的有效的信号线是 CS#、SO/SIO1、SI/SIO0、SCLK，在我们常用的接口电路中，**CS#、WP# 和 HOLD# 都是上拉**。检查 SPI 接口的模块时，需要确定主控端是否已经预留对应的 IO 口，并且确保 IO 已经和模块连接上。

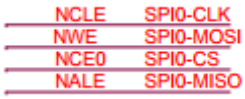


图 7-6: 原理图

图 7-7: 原理图

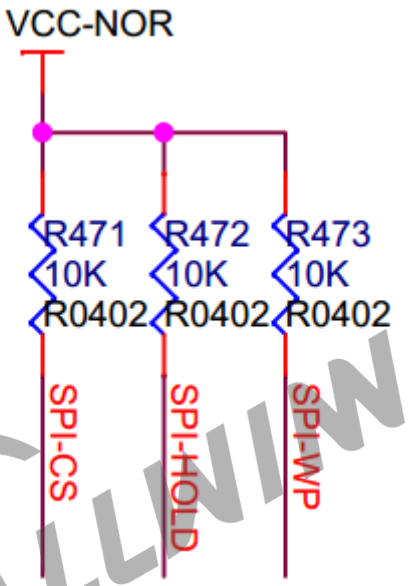


图 7-8: 原理图

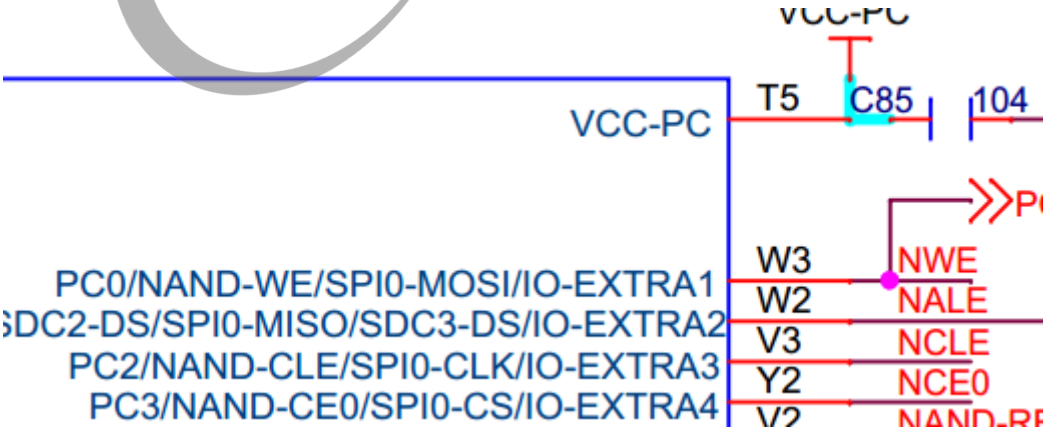


图 7-9: 原理图

7.1.4 对比实验

(1) 不同硬件板，同一设备，做对比实验；

(2) 同一硬件板，不同设备，做对比实验；

(3) 对比实验结果分析；

a. 怀疑整体：

机器是一个整体，其他部件有问题，也是可能导致出错的。比如说 DRAM 出错，这一点在量产的时候尤为明显。

b. 怀疑设备：

这个模块，事实上是一个主从关系，有问题，设备端的几率也是十分大。毕竟设备端千差百异。另外，当报大规模烧录不良的时候，要对设备端进行分析。

排查方法：

(1)（良品）设备端差异，换不同设备如果问题就不出现，设备端问题几率比较大。

(2)（黑片）对于怀疑是黑片、次品，可先用工具对样品做全盘读写测试，有必要的时候，可以做 rwcheck 工具进行读写压力测试。

7.2 常见问题及排查方法

7.2.1 打包报错

现象：

```
packing for linux
normal
ERROR: dl file rootfs_nor.fex size too large
ERROR: filename = rootfs_nor.fex
ERROR: dl_file_size = 17352 sector
ERROR: part_size = 15104 sector
ERROR: update mbr file fail
ERROR: update_mbr failed
yuxianyang@AExdroid02:~/longan$ ./build.sh clean
ACTION List: mkclean;========
Execute command: mkclean
INFO: clean kernel ...
INFO: Prepare toolchain ...
Cleaning modules ...
[GPU]: No GPU type is configured in .config.
```

图 7-10: 分区不够大

解决方法：

修改分区表

接将 rootfs_nor.fex 分区大小调大，比 17352 大就可以，但是大小要保证为 128 的倍数（需要按 block 对齐，nor block 64k，即 128 个扇区）。

路径：平台目录下 device/config/chips/xxx/configs/xxx/linux/sys_partition_nor.fex

7.2.2 读 ID 失败

现象：

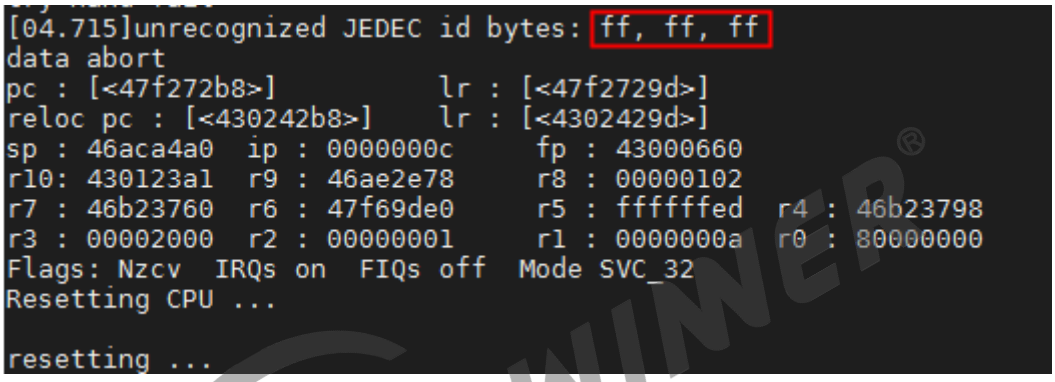


图 7-11: 硬件异常

解决方法：

通常读出为全 FF 一般是硬件问题，先进行 NOR 原理图检查，参考检查硬件板章节。

7.2.3 烧写正常，启动找不到正常 env

现象

```
Loading Environment from SUNXI_FLASH... *** Warning - bad CRC, using default environment

*** Warning - no device, using default environment
```

问题描述：烧写正常，启动找不到正常 env

问题根因及解决方法：

- **排查方向 1**

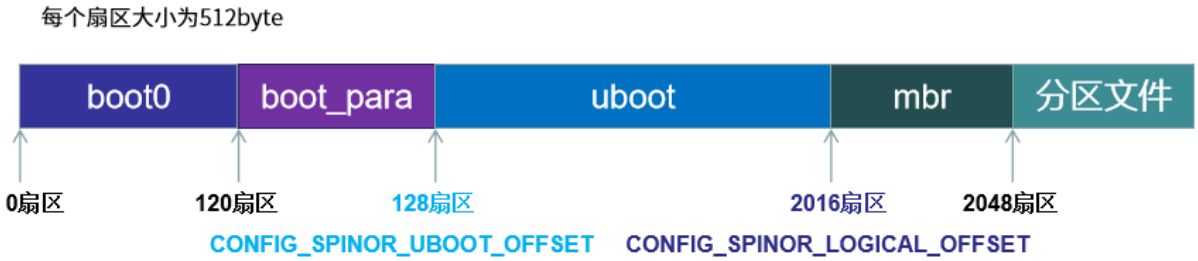


图 7-12: spinor 布局 1

烧写时，由于 uboot size 大于 CONFIG_SPINOR_LOGICAL_OFFSET - CONFIG_SPINOR_UBOOT_OFFSET，分区表被破坏

请检查u-boot-2018/configs/sunxiwxp1_nor_defconfig宏 CONFIG_SPINOR_LOGICAL_OFFSET、CONFIG_SPINOR_UBOOT_OFFSET 和out/pack_out/boot_package_nor.cfg 大小

“‘shell CONFIG_SPINOR_LOGICAL_OFFSET - CONFIG_SPINOR_UBOOT_OFFSET = 1888 扇区 = 944KB

:$ ls out/pack_out/boot_package_nor.cfg -alh -rwxrwxr-x 1 xxx xxx 356 Mar 8 15:21 out/pack_out/boot_package_nor.cfg “‘

- **排查方向 2**

若使用备份 env 功能，需要将下面配置同时配置上，若不使用备份 env 功能，需要将下面配置都删除，不然就会出现读 env 失败问题

打开备份 env 功能配置如下： device/config/chips/xxx/configs/default/BoardConfig（_nor）.mk

LICHEE_REDUNDANT_ENV_SIZE:=0x20000

u-boot-2018/configs/sunxiwxp1_defconfig

# Environment CONFIG_SUNXI_REDUNDAND_ENVIRONMENT=y CONFIG_SYS_REDUNDAND_ENVIRONMENT=y
CONFIG_SUNXI_ENV_REDUNDAND_PARTITION="env-redund" CONFIG_ENV_SIZE=0x20000

若使用备份 env 功能，上述配置需要同时配置上

7.2.4 mtd 设备不能 open

解决方法：通过 menuconfig 打开 CONFIG_MTD_CHAR 配置即可

7.2.5 环境变量参数更新失败

对于 uboot 2014 的代码，spinor 会有 cache 数据缓存层，而在更新 env 参数时，没有进行刷 cache，导致环境变量参数更新失败问题。

修改补丁如下：http://gerrit.allwinnertech.com:8081/c/lichee/brandy/+/96663

路径：**u-boot-2014.07/common/env_sunxi_flash.c**

```
File
@@                                  +10↑ - Show 106 common lines - +10↓
107 »       start = sunxi_partition_get_offset_byname(CONFIG_SUNXI_ENV_PARTITION);
108 »       if(!start){
109 »       »       printf("fail to find part named %s\n", CONFIG_SUNXI_ENV_PARTITION);
110 »       »       return -1;
111 »       }
112
113 »       ret = env_export(env_new);
114 »       if(ret)
115 »       »       goto fini;
116
117 »       ret = sunxi_flash_write(start, CONFIG_ENV_SIZE/512, env_new);
118 »       sunxi_flash_flush();
119 »       return ret;
120
121 fini:
122 »       return ret;
123 }
124
125 static void flash_env_relocate_spec(int workmode)
126 {
127 #if !defined(ENV_IS_EMBEDDED)
128 »       char buf[CONFIG_ENV_SIZE];
129 »       u32 start;
130
@@                                  +10↑ - Show 43 common lines - +10↓
```

图 7-13: flush_env_patch

在更新 env 写完数据时，调用 flush 接口

```
sunxi_flash_flush();
```

7.2.6 jffs2 文件系统异常

现象：挂载异常

```
[   1.141435] jffs2: jffs2_scan_eraseblock(): Magic bitmask 0x1985 not found at 0x00000f70: 0x0144 instead
[   1.150994] jffs2: jffs2_scan_eraseblock(): Magic bitmask 0x1985 not found at 0x00000f74: 0x912a instead
[   1.160547] jffs2: jffs2_scan_eraseblock(): Magic bitmask 0x1985 not found at 0x00000f78: 0x0002 instead
[   1.170127] jffs2: jffs2_scan_eraseblock(): Magic bitmask 0x1985 not found at 0x00000f7c: 0x000d instead
[   1.180689] jffs2: jffs2_scan_eraseblock(): Magic bitmask 0x1985 not found at 0x00000f80: 0x81a4 instead
[   1.190183] jffs2: jffs2_scan_eraseblock(): Magic bitmask 0x1985 not found at 0x00000f84: 0x03e8 instead
[   1.199668] jffs2: jffs2_scan_eraseblock(): Magic bitmask 0x1985 not found at 0x00000f88: 0x11d8 instead
[   1.209151] jffs2: jffs2_scan_eraseblock(): Magic bitmask 0x1985 not found at 0x00000f8c: 0xdec2 instead
[   1.218634] jffs2: jffs2_scan_eraseblock(): Magic bitmask 0x1985 not found at 0x00000f90: 0xdec2 instead
[   1.228102] jffs2: jffs2_scan_eraseblock(): Magic bitmask 0x1985 not found at 0x00000f94: 0xdec2 instead
```

根本原因：

jffs2 制作文件系统工具是按照 64K 制作的镜像，内核按照 4K 访问并产生了 4K 的读写数据，之后内核升级又改为 64K 访问该分区，导致之前 4K 生成的数据无法正常识别，文件系统挂载报错，挂载为只读。

解决方法：

1. 如果文件系统数据不重要，丢弃文件 jffs2 系统旧数据，重新烧写 jffs2 文件系统分区。
2. 如果想保留之前文件系统原数据，则内核需要 OTA 将擦除单位改为旧的配置，以 4K 进行擦写数据。

7.2.7 数据丢失

7.2.7.1 有掉电场景

背景：

- **检查掉电时序**

在部分情况，由于掉电比较慢，可能会出现在电压不稳定的情况下还在进行工作（如检查供电电源，检查供电电源低于 2.7V），这样会导致传输数据异常，如果的写擦命令的地址位数据异常，就会出现误擦，丢数据情况。

解决方案：

1. 检查 RTC spec，是否有 RTC 电压检查寄存器

11:4	R/W	0x2	VCCIO_DET_SPARE VCCIO detect spare. bit[7:5]: reserved bit[4]: bypass debounce 电路 0: bypass 1: not bypass bit[3]: 使能控制 0: disable vccio detecter 1: force detecter output bit[2:0]: 档位调节 000: 选择检测阈值为 2.5V 001: 选择检测阈值为 2.6V 010: 选择检测阈值为 2.7V 011: 选择检测阈值为 2.8V 100: 选择检测阈值为 2.9V 101: 选择检测阈值为 3V 110~111: N/A

图 7-14: RTC 寄存器

如果有，打上下列补丁（**注：**补丁是针对平台的，请根据相应平台进行修改）

连接：http://gerrit.allwinnertech.com:8081/c/lichee/brandy-2.0/u-boot-2018/+/174167

arch/arm/include/asm/arch-sunxi/clock_sun8iw20.h

```
+10↓                                                    +10↑ - Show 334 common lines - +10↓
@@
#define USBEHCI0_RST_BIT 20
#define USBEHCI0_GATIING_BIT 4
#define USBPHY0_RST_BIT 30
#define USBPHY0_SCLK_GATING_BIT 29

#define USBEHCI1_RST_BIT 21
#define USBEHCI1_GATIING_BIT 5
#define USBPHY1_RST_BIT 30
#define USBPHY1_SCLK_GATING_BIT 29

#define FORCE_DETECTER_OUTPUT»  »       (1 << 7)
#define VCCIO_THRESHOLD_VOLTAGE_2_5»    (0 << 4)
#define VCCIO_THRESHOLD_VOLTAGE_2_6»    (1 << 4)
#define VCCIO_THRESHOLD_VOLTAGE_2_7»    (2 << 4)
#define VCCIO_THRESHOLD_VOLTAGE_2_8»    (3 << 4)
#define VCCIO_THRESHOLD_VOLTAGE_2_9»    (4 << 4)
#define VCCIO_THRESHOLD_VOLTAGE_3_0»    (5 << 4)
#define VCCIO_DET_BYPASS_EN»    »       (1 << 0)
void rtc_set_vccio_det_spare(void);
#endif /* _SUNXI_CLOCK_SUN8IW20_H */
```

图 7-15: 补丁 1

arch/arm/mach-sunxi/clock_sun8iw20.c

```
@@                                          Show 4 common lines
 * SPDX-License-Identifier:»    GPL-2.0+
*/

#include <common.h>
#include <asm/io.h>
#include <asm/arch/cpu.h>
#include <asm/arch/clock.h>
#include <asm/arch/timer.h>
#include <asm/arch/prcm.h>

void rtc_set_vccio_det_spare(void)
{
»       u32 val = 0;
»       val = readl(SUNXI_RTC_BASE + 0x1f4);
»       val &= ~(0xff << 4);
»       val |= (VCCIO_THRESHOLD_VOLTAGE_2_9 | FORCE_DETECTER_OUTPUT);
»       val &= ~VCCIO_DET_BYPASS_EN;
»       writel(val, SUNXI_RTC_BASE + 0x1f4);
}

void clock_init_uart(void)
{
»       struct sunxi_ccm_reg *const ccm =
»       »       (struct sunxi_ccm_reg *)SUNXI_CCM_BASE;

»       /* uart clock source is apb2 */
»       writel(APB2_CLK_SRC_OSC24M|
»              APB2_CLK_RATE_N_1|
»              APB2_CLK_RATE_M(1),
@@                                  +10↑ - Show 385 common lines - +10↓
```

图 7-16: 补丁 2

2. 如果没有 RTC 功能，可以使用 lock 功能来规避，误擦写问题。

SPINOR flash 支持两种 lock 功能，分别如下：

block 保护

7.1.17 W25Q128FV Individual Block Memory Protection (WPS=1)

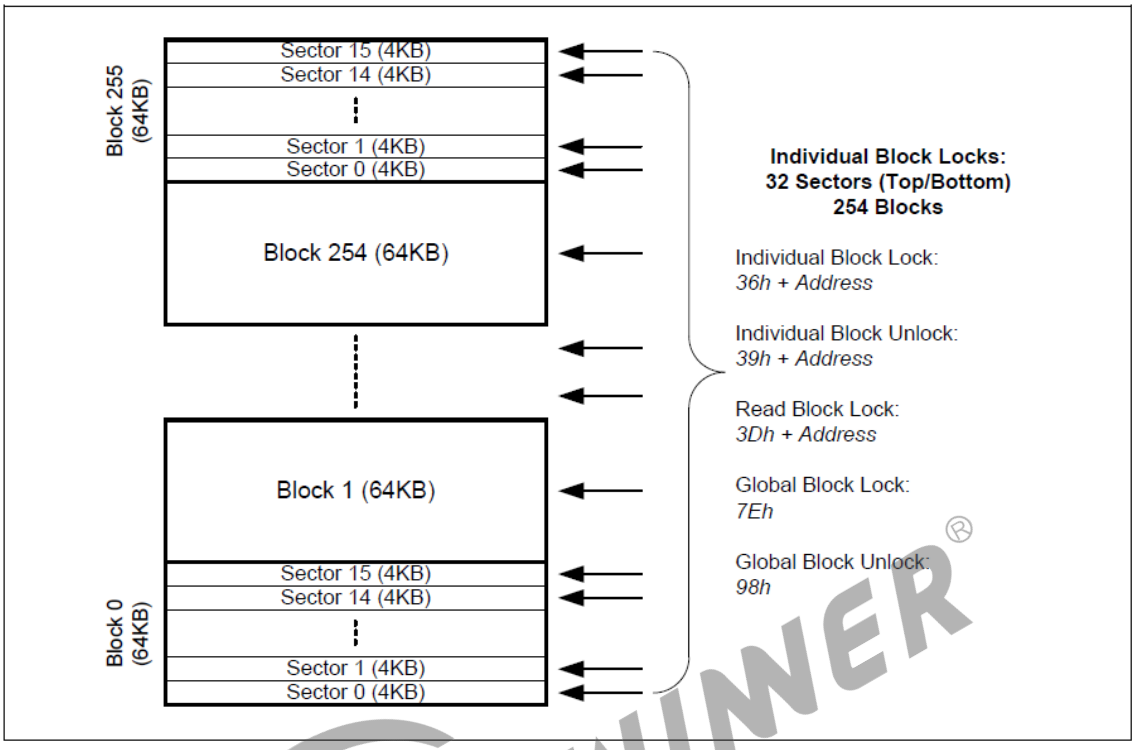


Figure 4d. Individual Block/Sector Locks

图 7-17: block-lock

区域保护

Table 4. GD25Q128E Protected area size (CMP=0)

Status Register Content					Memory Content			
BP4	BP3	BP2	BP1	BP0	Blocks	Addresses	Density	Portion
X	X	0	0	0	NONE	NONE	NONE	NONE
0	0	0	0	1	252 to 255	FC0000H-FFFFFFH	256KB	Upper 1/64
0	0	0	1	0	248 to 255	F80000H-FFFFFFH	512KB	Upper 1/32
0	0	0	1	1	240 to 255	F00000H-FFFFFFH	1MB	Upper 1/16
0	0	1	0	0	224 to 255	E00000H-FFFFFFH	2MB	Upper 1/8
0	0	1	0	1	192 to 255	C00000H-FFFFFFH	4MB	Upper 1/4
0	0	1	1	0	128 to 255	800000H-FFFFFFH	8MB	Upper 1/2
0	1	0	0	1	0 to 3	000000H-03FFFFH	256KB	Lower 1/64
0	1	0	1	0	0 to 7	000000H-07FFFFH	512KB	Lower 1/32
0	1	0	1	1	0 to 15	000000H-0FFFFFH	1MB	Lower 1/16
0	1	1	0	0	0 to 31	000000H-1FFFFFH	2MB	Lower 1/8
0	1	1	0	1	0 to 63	000000H-3FFFFFH	4MB	Lower 1/4
0	1	1	1	0	0 to 127	000000H-7FFFFFH	8MB	Lower 1/2
X	X	1	1	1	0 to 255	000000H-FFFFFFH	16MB	ALL
1	0	0	0	1	255	FFF000H-FFFFFFH	4KB	Top Block
1	0	0	1	0	255	FFE000H-FFFFFFH	8KB	Top Block
1	0	0	1	1	255	FFC000H-FFFFFFH	16KB	Top Block
1	0	1	0	X	255	FF8000H-FFFFFFH	32KB	Top Block
1	0	1	1	0	255	FF8000H-FFFFFFH	32KB	Top Block
1	1	0	0	1	0	000000H-000FFFH	4KB	Bottom Block
1	1	0	1	0	0	000000H-001FFFH	8KB	Bottom Block

图 7-18: area-lock

如果 flash 支持 **block 保护**，需要在 ID 表配置下列参数，和增加设备树配置。

```
#define SPI_NOR_HAS_LOCK    BIT(8)  /* Flash supports lock/unlock via SR */
#define SPI_NOR_INDIVIDUAL_LOCK BIT(16) /* individual block/sector lock mode */
#define SPI_NOR_HAS_LOCK_HANDLE BIT(17) /* OP/ERASE for lock operation */
```

例：

```
INFO("w25q128jv", 0xef7018, 0, 64 * 1024, 256,
        SECT_4K | SPI_NOR_DUAL_READ | SPI_NOR_QUAD_READ |
        SPI_NOR_HAS_LOCK | SPI_NOR_INDIVIDUAL_LOCK | SPI_NOR_HAS_LOCK_HANDLE)
```

dts：

```
spi_board0 {
    individual_lock;
  };
```

如果 flash 支持**区域保护**，需要在 ID 表配置下列参数

```
#define SPI_NOR_HAS_LOCK    BIT(8)  /* Flash supports lock/unlock via SR */
#define SPI_NOR_HAS_LOCK_HANDLE BIT(17) /* OP/ERASE for lock operation */
```

例：

```
INFO("gd25lq128", 0xc86018, 0, 64 * 1024, 256,
        SECT_4K | SPI_NOR_DUAL_READ | SPI_NOR_QUAD_READ |
        SPI_NOR_HAS_LOCK | SPI_NOR_HAS_LOCK_HANDLE)
```

如果出现下列打印，说明 lock 配置有问题，再检查一些配置。

```
individual_lock fail

not support individual global unlock nor

not support individual global lock nor
```

7.2.7.2 无掉电场景

无掉电场景发生的数据丢失问题，首先排查是否存在误擦写的应用场景。曾经遇到过客户在 uboot 增加擦动作，误擦了 data 分区，导致文件缺失。

7.2.8 3byte、4byte 地址码

问题描述：

A40I 平台，sys_config.fex 中默认 nor flash 的大小配置为 16MB，在 uboot 模式下，输入 reset 命令，系统可以正常重启。把 sys_config.fe 中 nor flash 大小设置成 32MB，在 Uboot 下，输入 reset 命令，板子会调整到烧录模式。

问题原因：

在 2014 旧版本的 u-boot 版本中 nor falsh 容量大小是在 sun*iw_defconfig 文件中定义的，当设置的 nor flash 参数大于 16M，并且 bootloader 阶段没有支持 4 byte 地址码的转换策略，将会出现参数设置错误进入 FEL 烧写模式。

```
[norflash]
size                = 32
```

图 7-19: Flash size

注：

3 byte address mode：3 字节地址码，寻址范围 0x000000 ~ 0xFFFFFF，最大支持 16M 大小 flash。

4 byte address mode：4 字节地址码，寻址范围 0x00000000 ~ 0xFFFFFFFF，最大支持 4G 大小 flash。

解决方法：

u-boot 2014 bootloader 支持 4 线地址策略。

补丁：http://gerrit.allwinnertech.com:8081/c/lichee/brandy/+/6212

路径：**u-boot-2014.07/drivers/spinor/sunxi_spinor.c**

```
ile
@@                                  +10↑ - Show 21 common lines - +10↓
 * MA 02111-1307 USA
 */
#include <common.h>
#include <malloc.h>
#include <spi.h>
#include <asm/arch/spi.h>
#include <sunxi_mbr.h>
#include <private_boot0.h>
#include <asm/arch/spinor.h>
#include <sunxi_board.h>
#include <fdt_support.h>


static int   spinor_flash_inited = 0;
uint  total_write_bytes;
static char *spinor_store_buffer;
//static char  spinor_mbr[SUNXI_MBR_SIZE];
static char *spinor_write_cache;
static int   spinor_cache_block = -1;
static int   spinor_4bytes_addr_mode = 0;

#define ENABLE_4BYTES»  1
#define DISABLE_4BYTES  0
#define  SYSTEM_PAGE_SIZE        (512)
#define  SPINOR_PAGE_SIZE        (256)
#define  NPAGE_IN_1SYSPAGE       (SYSTEM_PAGE_SIZE/SPINOR_PAGE_SIZE)
#define  SPINOR_BLOCK_BYTES      (64 * 1024)
#define  SPINOR_BLOCK_SECTORS    (SPINOR_BLOCK_BYTES/512)

extern uint sunxi_sprite_generate_checksum(void *buffer, uint length, uint src_sum);
extern int sunxi_sprite_verify_checksum(void *buffer, uint length, uint src_sum);
static void spinor_enter_4bytes_addr(int);
static void spinor_config_addr_mode(u8 *sdata, uint page_addr, uint *num, u8 cmd);
```

图 7-20: 4addr_patch1

```
*/
static int __spinor_pp(uint page_addr, void *buf, uint len)
{
	u8   sdata[300];
	u8   status = 0;
	uint i = 0;
	int  ret = -1;
	uint txnum = 0;
	uint rxnum = 0;

	if (len > 256)
	{
		return -1;
	}

	ret = __spinor_wren();
	if (ret < 0)
	{
		return -1;
	}

	memset((void *)sdata, 0xff, sizeof(sdata)/sizeof(sdata[0]));
	spinor_config_addr_mode(sdata,page_addr,&txnum,SPINOR_PP);
	memcpy((void *)(sdata+txnum), buf, len);
	txnum += len;

```

图 7-21: 4addr_patch2

```
static int __spinor_sector_read(uint start, uint sector_cnt, void *buf)
{
	uint page_addr;
	uint rbyte_cnt;
	u8   sdata[5] = {0};
	int ret = 0;
	uint tmp_cnt, tmp_offset = 0;
	void  *tmp_buf;
	uint txnum = 0;
	uint rxnum = 0;


	while (sector_cnt)
	{
		if (sector_cnt > 127)
		{
			tmp_cnt = 127;
		}
		else
		{
			tmp_cnt = sector_cnt;
		}

		page_addr = (start + tmp_offset) * SYSTEM_PAGE_SIZE;
		rbyte_cnt = tmp_cnt * SYSTEM_PAGE_SIZE;
		spinor_config_addr_mode(sdata,page_addr,&txnum,SPINOR_READ);
```

图 7-22: 4addr_patch3

```
465 int spinor_init(int stage)
466 {
467 »       int spi_size = 0;
468
469 »       spi_size = spinor_size();
470 »       if(spinor_flash_inited)
471 »       {
472 »       »       puts("sunxi spinor is already inited\n");
473                 return 0;
474 »       }
475 »       else
476 »       {
477 »       »       puts("sunxi spinor is initing...");
478 »       »       if(spic_init(0))
479 »       »       {
 @@                                      Show 2 common lines
482 »       »       »       return -1;
483 »       »       }
484 »       »       else
485 »       »       {
486 »       »       »       puts("OK\n");
487 »       »       }
488
489 »       }
490 »       spinor_flash_inited ++;
491
492 »       if(spi_size > 16*1024*1024/512)
493 »       {
494 »       »       spinor_4bytes_addr_mode = 1;
495 »       »       spinor_enter_4bytes_addr(ENABLE_4BYTES);
496 »       }
497
```

图 7-23: 4addr_patch4

```
784 int spinor_size(void)
785 {
786 »       int size = 0;
787 »       int ret = -1;
788 »       int nodeoffset = 0;
789
790 »       nodeoffset =  fdt_path_offset(working_fdt,"/soc/spi_board0");
791 »       if(nodeoffset > 0)
792 »       {
793 »       »       ret = fdt_getprop_u32(working_fdt, nodeoffset,"sflash_size",(uint32_t*)&size);
794 »       }
795 »       if (ret < 0 )
796 »       {
797 »       »       size = 8*1024*1024/512;
798 »       »       printf("get flash_size warning\n");
799 »       }
800 »       else
801 »       {
802 »       »       size =  size*1024*1024/512;
803 »       }
804 »       printf("flash size =0x%x sectors\n",size);
805
806 »       return size;
807 }
808 /*
809 ********************************************************************************************************
    ********
810 *
811 *                                           function
812 *
813 *    name          :
814 *
815 *    parmeters     :
816 *
 @@                                 +10↑ - Show 127 common lines - +10↓
944 //»   for(i=0;i<total_write_bytes;i++)
945 //»   {
946 //»   »       if(buffer[i] != spinor_store_buffer[i])
947 //»   »       {
948 //»   »       »       printf("compare spinor read and write error\n");
949 //»   »       »       printf("offset %d\n", i);
950 //
951 //»   »       »       return -1;
952 //»   »       }
953 //»   }
954 »       spinor_enter_4bytes_addr(DISABLE_4BYTES);
955 »       printf("spinor download data ok\n");
956
957 »       return 0;
958 }
959 /*
```

图 7-24: 4addr_patch5

```
1099 static void spinor_enter_4bytes_addr(int enable)
1100 {
1101 »       int command = 0;
1102 »       if(spinor_4bytes_addr_mode == 1 && enable == 1)
1103 »       »       command = 0xB7;
1104 »       else if(spinor_4bytes_addr_mode == 1 && enable == 0)
1105 »       »       command = 0xE9;
1106 »       else
1107 »       »       return ;
1108
1109 »       spic_config_dual_mode(0, 0, 0, 1);
1110
1111 »       spic_rw(1, (void*)&command, 0, 0);
1112 »       return;
1113 }
1114
1115 static void spinor_config_addr_mode(u8 *sdata, uint page_addr, uint *num, u8 cmd)
1116 {
1117 »       if ((sdata == NULL)||(num == NULL))
1118 »       {
1119 »       »       return;
1120 »       }
1121
1122 »       if(spinor_4bytes_addr_mode == 0)
1123 »       {
1124 »       »       *num = 4;
1125 »       »       sdata[0]  =  cmd;
1126 »       »       sdata[1]  = (page_addr >> 16) & 0xff;
1127 »       »       sdata[2]  = (page_addr >> 8 ) & 0xff;
1128 »       »       sdata[3]  =  page_addr        & 0xff;
1129 »       }
1130 »       else if(spinor_4bytes_addr_mode == 1)
1131 »       {
1132 »       »       *num = 5;
1133 »       »       sdata[0]  =  cmd;
1134 »       »       sdata[1]  = (page_addr >> 24) & 0xff;
1135 »       »       sdata[2]  = (page_addr >> 16) & 0xff;
1136 »       »       sdata[3]  = (page_addr >> 8 ) & 0xff;
1137 »       »       sdata[4]  =  page_addr        & 0xff;
1138 »       }
1139
1140 }
```

图 7-25: 4addr_patch6

7.2.9 自定义 nor flash 烧写布局

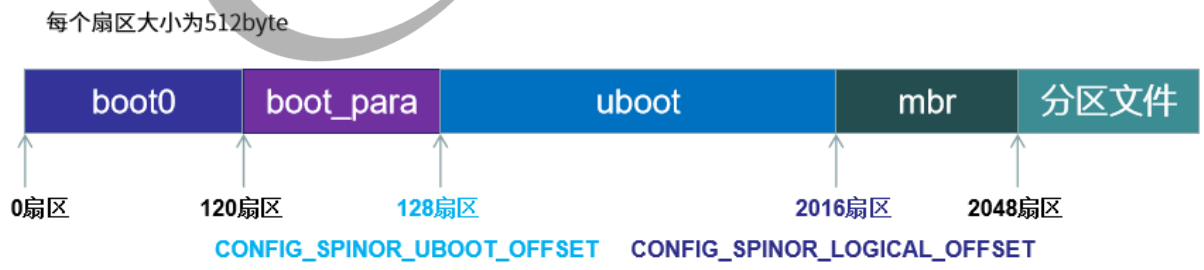


图 7-26: spinor 布局 1

spinor flash 布局划分如上图所示，但实际烧写的各个镜像文件一般小于划分区域，如果想尽用 spinor flash 可以根据镜像文件的实际大下重新划分烧写布局。

1. 查看镜像文件大小

path: out/pack_out

命令：ls -alh

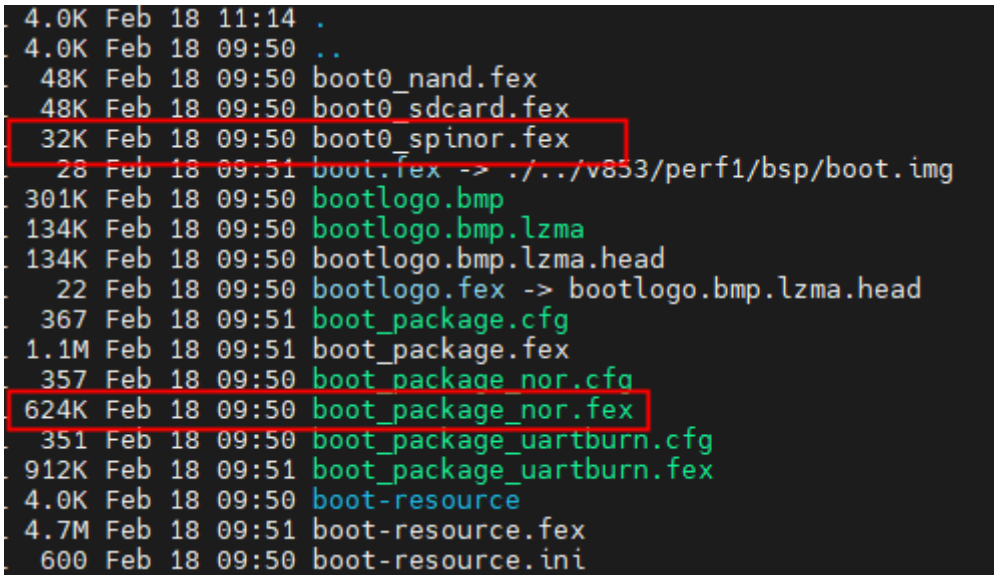


图 7-27: 镜像文件大小

boot0 实际大小 32K，64 扇区

boot_package(uboot) 实际大小 624K，1248 扇区

2. 计算偏移地址

boot0 大小 64 扇区，boot_para 8 扇区，**u-boot 起始扇区** = 64 + 8 = 72，uboot 的起始扇区可以设置为 72 + 一定预留（可不预留，或预留几个分区）

boot_package(uboot) 大小为 1248 扇区，**MBR 起始扇区** = 64 + 8 +1248 = 1320，MBR 起始扇区可以设置为 1320 + 一定预留（可不预留，或预留几个分区）

分区表中也可以根据分区文件的实际大小，设置分区大小，patch：device/config/chips/v853/configs/default/**sys_partition_nor.fex**

3. 参数设置方法

以 sun8iw21p1 为例，需要在 spinor.mk、sun8iw21p1_nor_defconfig、sun8iw21p1_defconfig 将 uboot 偏移、MBR 偏移设置为上面计算的起始扇区数

1）boot0 参数修改路径：/brandy/brandy-2.0/spl/board/sun8iw21p1/**spinor.mk**

```
10 endif
11
12 MODULE=spinor
13 CFG_SUNXI_SPINOR =y
14 CFG_SUNXI_SPI =y
15 CFG_SUNXI_DMA =y
16 CFG_SPI_USE_DMA =y
17 CFG_SPINOR_UBOOT_OFFSET=128
```

图 7-28: spinor.mk

2）u-boot 参数修改路径：/brandy/brandy-2.0/u-boot-2018/configs/**sun8iw21p1_nor_defconfig、sun8iw21p1_defconfig**

```
44 CONFIG_SF_DEFAULT_SPEED=50000000
45 # BIT(12) BIT(13) (SPI_RX_DUAL|SPI_RX_QUAD)
46 CONFIG_SF_DEFAULT_MODE=0x3000
47 CONFIG_SPINOR_UBOOT_OFFSET=128
48 CONFIG_SPINOR_LOGICAL_OFFSET=2016
49
50
```

图 7-29: sun8iw21p1_nor_defconfig

3）内核参数修改路径：linux4.9/drivers/mtd/sunxipart.c 中的 MBR_OFFSET 宏

MBR_OFFSET 宏修改为 ((**MBR 起始扇区** + 32) / 2 - 16) * 1024

若通过 cmdlink 传参，需要修改 linux4.9/arch/arm/boot/dts/sun8iw21p1.dtsi 中 bootargs 中的 mbr_offset 值为 ((起始扇区 + 32) / 2 - 16) * 1024

著作权声明

商标声明

免责声明