Linux SLCNAND
开发指南

版本号: 1.2
发布日期: 2024.03.08

版本历史

版本号	日期	制/修订人	内容描述
1.0	2022.11.10	AW1669	初始版本
1.1	2023.03.22	AW1669	增加 a523 平台介绍
1.2	2024.03.08	AW2155	更新内核及设备树配置方法

目　录

插　图

1 概述

1.1 编写目的

介绍 Sunxi SLCNand mtd/ubi 驱动设计, 方便相关驱动和应用开发人员

1.2 适用范围

本设计适用于 UBI 方案 SLCNAND 平台

1.3 相关人员

Nand 模块开发人员，及应用开发人员等

2 术语、缩略语及概念

MTD：（Memory Technology device）是用于访问存储设备的 linux 子系统。本模块是 MTD 子系统的 flash 驱动部分

UBI：UBI 子系统是基于 MTD 子系统的，在 MTD 上实现 nand 特性的管理逻辑，向上屏蔽 nand 的特性

坏块 (Bad Block)：制作工艺和 nand 本身的物理性质导致在出厂和正常使用过程中都会产生坏块

3 流程设计

3.1 体系结构

NAND MTD/UBI 驱动主要包括 5 大组件，如下图：

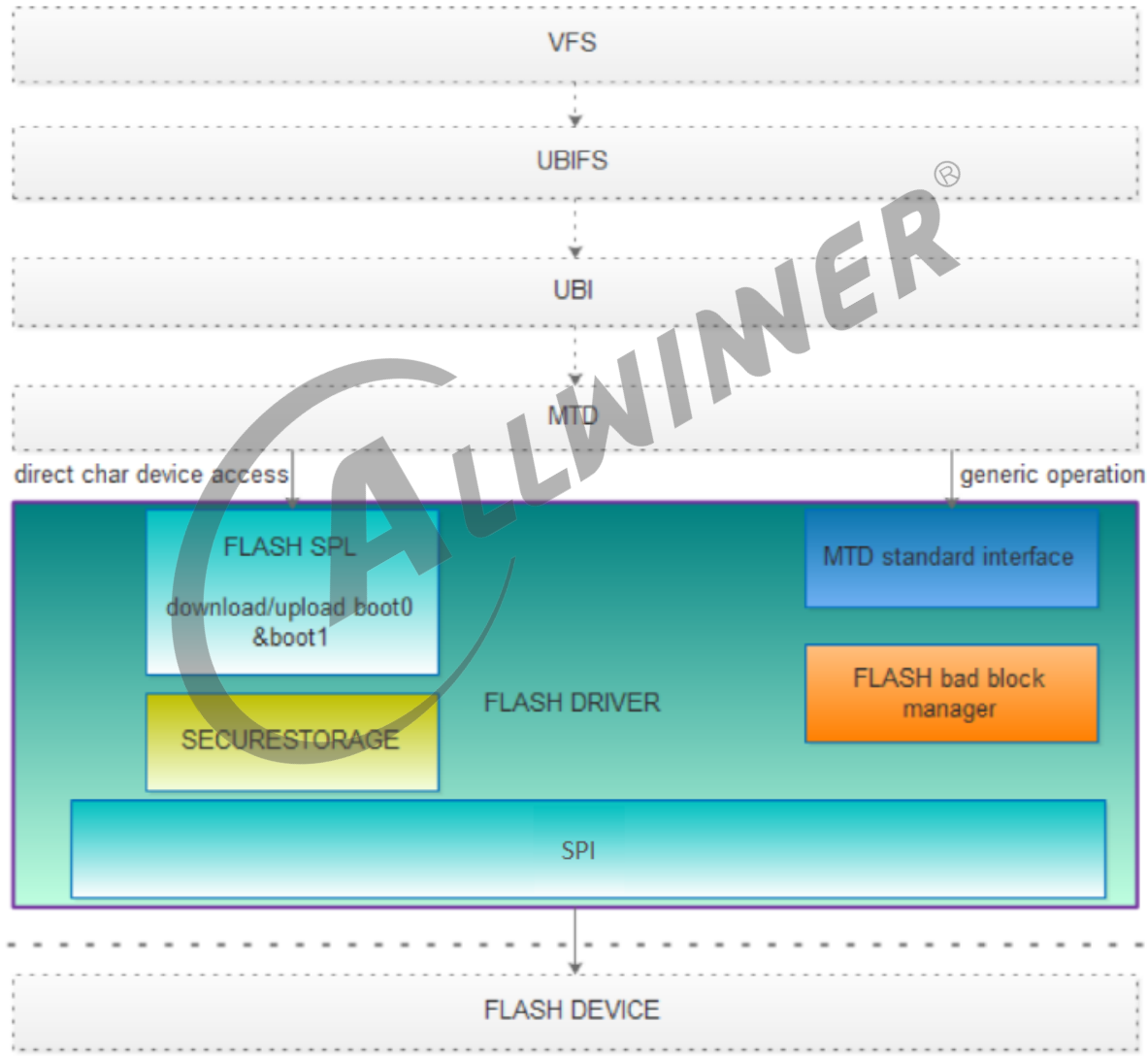

图 3-1: UBI 架构

说明：

MTD standard interface: 对接 MTD 层通用读写接口

FLASH bad block manager: 驱动层对 flash 坏块的管理

FLASH SPL: 主要是实现读写 boot0、boot1，可用于 ioctl 对 boot0、boot1 的升级

SECURESTORAGE：主要是给上层提供私有数据的管理

SPI：HOST 端控制器层的实现

3.2 源码结构

3.2.1 内核版本 ≤Linux5.4

kernel 源码目录：linux-5.4/drivers/mtd/awnand/rawnand

```
├── Kconfig
├── Makefile
├── rawnand
│   ├── aw_rawnand_base.c
│   ├── aw_rawnand_bbt.c
│   ├── aw_rawnand_ids.c
│   ├── aw_rawnand_nfc.c
│   ├── aw_rawnand_nfc.h
│   ├── aw_rawnand_securestorage.c
│   ├── aw_rawnand_spl.c
│   ├── Kconfig
│   └── Makefile
```

头文件在：linux-5.4/include/linux/mtd/

```
include/linux/mtd/
├── aw-rawnand.h
```

3.2.2 内核版本 ≥Linux5.10

kernel 源码目录：bsp/drivers/mtd/awnand

```
├── Kconfig
├── Makefile
├── rawnand
│   ├── aw_rawnand_base.c
│   ├── aw_rawnand_bbt.c
│   ├── aw_rawnand_ids.c
│   ├── aw_rawnand_nfc.c
│   ├── aw_rawnand_nfc.h
│   ├── aw_rawnand_nfc_v0.c
│   ├── aw_rawnand_securestorage.c
│   ├── aw_rawnand_spl.c
│   ├── bootloader_update_demo.c
│   ├── Kconfig
│   └── Makefile
```

头文件在：bsp/include/linux/mtd/

```
├── aw-rawnand.h
```

3.3 关键数据定义

3.3.1 flash 设备信息数据结构

```
struct aw_nand_flash_dev {
    char *name;
    union {
        struct {
            uint8_t mfr_id;
            uint8_t dev_id;
        };
        uint8_t id[RAWNAND_MAX_ID_LEN];
    };
    int id_len;
    unsigned int dies_per_chip;
    /*main data size, eg. Page Size:(2K+64)byte ==> pagesize=2K byte,
     * sparesize=64byte*/
    unsigned int pagesize;
    unsigned int sparesize;
    unsigned int pages_per_blk;
    unsigned int blks_per_die;
    unsigned int access_freq;
    enum error_management badblock_flag_pos;
    unsigned int pe_cycles;
    unsigned int options;
};
```

说明：

- name：flash 的物料名字
- id：flash 的 id 码
- dies_per_chip：每 chip 的 die 个数
- pagesize：一个页大小
- sparesize：spare 区大小
- pages_per_blk：每 block 有多少个 page
- access_freq：工作频率
- badblock_flag_pos：坏块标志存放在每个 block 的那个 page 中

  1. PST_FIRST_PAGE
  2. PST_FIRST_TWO_PAGES
  3. PST_LAST_PAGE
  4. PST_LAST_TWO_PAGES
  5. PST_FIRST_AND_LAST_PAGES
  6. PST_FIRST_TWO_AND_LAST_PAGES

- pe_cycles：flash 支持擦除次数
- options：支持的操作

1. RAWNAND_ITF_SDR
2. RAWNAND_ITF_ONFI_DDR
3. RAWNAND_ITF_ONFI_DDR2
4. RAWNAND_ITF_TOGGLE_DDR
5. RAWNAND_ITF_TOGGLE_DDR2
6. RAWNAND_TOGGLE_SUPPORT_ONLY // TOGGLE only suppor
7. RAWNAND_ONFI_TIMING_MODE // ONFI timing mode, used in both asynchronous and synchronous mode
8. RAWNAND_ONFI_FEATURE_EXT_PARAM_PAGE //ONFI features
9. RAWNAND_MULTI_WRITE // Chip allow multi writes (80h – 11h ~ 80h – 10h)
10. RAWNAND_MULTI_READ // Chip allow multi reads
11. RAWNAND_MULTI_ERASE // Chip allow multi erase（60h-60h-d0h）
12. RAWNAND_MULTI_ONFI_ERASE // Chip allow onfi multi erase（60h-d1h – 60h- d0h）
13. RAWNAND_JEDEC_MULTI_WRITE // Chip allow multi writes（80h – 11h ~ 81h – 10h）
14. RAWNAND_ROW_ADDR_2 // Device needs 2rd row address cycle
15. RAWNAND_TOGGLE_DDR_TO_SDR // Default Toggle DDR1.0 , SDR need to set
16. RAWNAND_NFC_RANDOM //Open nfc randomizer

例子（TC58NVG1S3HTA00）：

```
{
    .name = "TC58NVG1S3HTA00",
    .id = {0x98, 0xda, 0x90, 0x15, 0x76},
    .id_len = 5,
    .dies_per_chip = 1,
    .pagesize = SZ_2K,
    .sparesize = 128,
    .pages_per_blk = 64,
    .blks_per_die = 2048,
    .access_freq = 40,
    .badblock_flag_pos = PST_FIRST_PAGE,
    .pe_cycles = PE_CYCLES_100K,
    .options = RAWNAND_ITF_SDR | RAWNAND_NFC_RANDOM | RAWNAND_JEDEC_MULTI_WRITE |
      RAWNAND_MULTI_ERASE,
},
```

说明

详细的 flash 参数配置方法参考《NAND 物料 _ 调试指南》

3.3.2 flash chip 数据结构

```
struct aw_nand_chip {
    struct mutex lock;
    /*************************
     *  mtd layer
     *-----------------------
     *  simu chip
     *-----------------------
     *    chip
     *| --blkn----|--blkn+1--|
     *|  (planeA) |  (planeB)|
     * ************************/
    struct mtd_info mtd;
#define SLC_NAND    (0)
#define MLC_NAND    (1)
    int type;

    uint8_t id[RAWNAND_MAX_ID_LEN];
    unsigned int dies;
#define MAX_DIES (2U)
    uint64_t diesize[MAX_DIES];
    int chips;
    uint64_t chipsize;
    uint64_t simu_chipsize;
    int chip_shift;
    int simu_chip_shift;
    int chip_pages;
    /*simulation is for multi, see line@48 rawnand  multiplane layout.*/
    int simu_chip_pages;
    int chip_pages_mask;
    int simu_chip_pages_mask;

    /*main data size*/
    int pagesize;
    int simu_pagesize;
    /*main data size shift*/
    unsigned int pagesize_shift;
    unsigned int simu_pagesize_shift;
    int pagesize_mask;
    int simu_pagesize_mask;
    /*main data size + spare data size*/
    int real_pagesize;
    unsigned int erasesize;
    unsigned int simu_erasesize;
    unsigned int erase_shift;
    unsigned int simu_erase_shift;
    unsigned int erasesize_mask;
    unsigned int simu_erasesize_mask;
    unsigned int pages_per_blk_shift;
    unsigned int simu_pages_per_blk_shift;
    unsigned int pages_per_blk_mask;
    unsigned int simu_pages_per_blk_mask;
    int avalid_sparesize;
    int ecc_mode;
    int random;
    int row_cycles;
    enum error_management badblock_mark_pos;
    unsigned int pe_cycles;
```

```
    unsigned int options;
    int clk_rate;

    int operate_boot0;
    int boot0_ecc_mode;
    int uboot_end;

    struct select_chip selected_chip;
    struct ce_info ceinfo[MAX_CHIPS];

    struct aw_nand_chip_cache simu_chip_buffer;

    struct rawnand_data_interface data_interface;
#define BBT_B_INVALID  (2)
#define BBT_B_BAD  (1)
#define BBT_B_GOOD  (0)
    uint8_t *bbt;
    /*mark whether the corresponding bbt bit is updated*/
    uint8_t *bbtd;

    uint8_t bitflips;

    void (*select_chip)(struct mtd_info *mtd, int chip);
    bool  (*dev_ready_wait)(struct mtd_info *mtd);
    int  (*dev_status)(struct mtd_info *mtd);


    int (*block_bad)(struct mtd_info *mtd, int block);
    int (*simu_block_bad)(struct mtd_info *mtd, int block);
    int (*block_markbad)(struct mtd_info *mtd, int block);
    int (*simu_block_markbad)(struct mtd_info *mtd, int block);
    /*scan device to update bbt*/
    int (*scan_bbt)(struct mtd_info *mtd);

    int (*erase)(struct mtd_info *mtd, int page);
    int (*multi_erase)(struct mtd_info *mtd, int page);

    int (*write_page)(struct mtd_info *mtd, struct aw_nand_chip *chip,
      uint8_t *mdata, int mlen, uint8_t *sdata, int slen, int page);
    int (*multi_write_page)(struct mtd_info *mtd, struct aw_nand_chip *chip,
      uint8_t *mdata, int mlen, uint8_t *sdata, int slen, int page);
    int (*cache_write_page)(struct mtd_info *mtd, struct aw_nand_chip *chip,
      uint8_t *mdata, int mlen, uint8_t *sdata, int slen, int page);
    int (*read_page)(struct mtd_info *mtd, struct aw_nand_chip *chip,
      uint8_t *mdata, int mlen, uint8_t *sdata, int slen, int page);
    int (*multi_read_page)(struct mtd_info *mtd, struct aw_nand_chip *chip,
      uint8_t *mdata, int mlen, uint8_t *sdata, int slen, int page);
    int (*read_page_spare)(struct mtd_info *mtd, struct aw_nand_chip *chip,
      uint8_t *sdata, int slen, int page);

    int (*write_boot0_page)(struct mtd_info *mtd, struct aw_nand_chip *chip,
      uint8_t *mdata, int mlen, uint8_t *sdata, int slen, int page);
    int (*read_boot0_page)(struct mtd_info *mtd, struct aw_nand_chip *chip,
      uint8_t *mdata, int mlen, uint8_t *sdata, int slen, int page);

    int (*setup_read_retry)(struct mtd_info *mtd, struct aw_nand_chip *chip);
    int (*setup_data_interface)(struct mtd_info *mtd, struct aw_nand_chip *chip,
        int chipnr, const struct rawnand_data_interface *conf);
```

```
    struct aw_nand_flash_dev *dev;

    void *priv;
    struct list_head node;
};
```

此结构定义了 flash chip 层的物理模型数据结构以及 chip 层对 flash 的操作接口。

- type：raw nand 的类型（SLC_NAND、MLC_NAND）
- pagesize：页大小
- simu_pagesize：super page 大小
- int real_pagesize：main data size + spare data size
- erasesize：擦除大小
- simu_erasesize：super block 擦除大小
- pages_per_blk_shift：block 转 page 数的移位次数
- ecc_mode：ecc 模式

1. BCH_16
2. BCH_24
3. define BCH_28
4. define BCH_32
5. define BCH_40
6. define BCH_48
7. define BCH_56
8. define BCH_60
9. define BCH_64 (8)

- random：随机化功能
- options：支持的操作

函数指针对应**物理层接口**说明

3.3.3 ubi_ec_hdr

```
struct ubi_ec_hdr {
    __be32  magic;
    __u8    version;
    __u8    padding1[3];
    __be64  ec; /* Warning: the current limit is 31-bit anyway! */
    __be32  vid_hdr_offset;
    __be32  data_offset;
    __be32  image_seq;
```

```
    __u8   padding2[32];
    __be32 hdr_crc;
} __packed;
```

@magic: erase counter header magic number (%UBI_EC_HDR_MAGIC)

@version: version of UBI implementation which is supposed to accept this UBI image

@padding1: reserved for future, zeroes

@ec: the erase counter

@vid_hdr_offset: where the VID header starts

@data_offset: where the user data start

@image_seq: image sequence number

@padding2: reserved for future, zeroes

@hdr_crc: erase counter header CRC checksum

EC: Erase Count，记录块的擦除次数，在 ubiattach 的时候指定一个 mtd，如果 PEB 上没有 EC，则用平均的 EC 值，写入 EC 值只有在擦除的时候才会增加 1

3.3.4 ubi_vid_hdr

```
struct ubi_vid_hdr {
    __be32 magic;
    __u8   version;
    __u8   vol_type;
    __u8   copy_flag;
    __u8   compat;
    __be32 vol_id;
    __be32 lnum;
    __u8   padding1[4];
    __be32 data_size;
    __be32 used_ebs;
    __be32 data_pad;
    __be32 data_crc;
    __u8   padding2[4];
    __be64 sqnum;
    __u8   padding3[12];
    __be32 hdr_crc;
} __packed;
```

@magic: volume identifier header magic number (%UBI_VID_HDR_MAGIC)

@version: UBI implementation version which is supposed to accept this UBI image (%UBI_VERSION)

@vol_type: volume type (%UBI_VID_DYNAMIC or %UBI_VID_STATIC)

@copy_flag: if this logical eraseblock was copied from another physical eraseblock (for wear-leveling reasons)

@compat: compatibility of this volume(%0, %UBI_COMPAT_DELETE, %UBI_COMPAT_IGNORE, %UBI_COMPAT_PRESERVE, or %UBI_COMPAT_REJECT)

@vol_id: ID of this volume

@lnum: logical eraseblock number

@padding1: reserved for future, zeroes

@data_size: how many bytes of data this logical eraseblock contains

@used_ebs: total number of used logical eraseblocks in this volume

@data_pad: how many bytes at the end of this physical eraseblock are not used

@data_crc: CRC checksum of the data stored in this logical eraseblock

@padding2: reserved for future, zeroes

@sqnum: sequence number

@padding3: reserved for future, zeroes

@hdr_crc: volume identifier header CRC checksum

参数说明

@sqnum 是创建此 VID 头时的全局序列计数器的值。每次 UBI 写一个新的 VID 头到 flash 时，全局序列计数器都会增加，比如当它将一个逻辑的 eraseblock 映射到一个新的物理的 eraseblock 时。全局序列计数器是一个无符号 64 位整数，我们假设它永远不会溢出。@sqnum(序列号) 用于区分新旧版本的逻辑擦除块。

有两种情况，可能有多个物理 eraseblock 对应同一个逻辑 eraseblock，即在卷标识头中有相同的 **@vol_id** 和 **@lnum** 值。假设我们有一个逻辑的擦除块 L，它被映射到物理的擦除块 P。

1. 因为 UBI 可以异步擦除物理上的擦除块，所以可能出现以下情况:L 被异步擦除，所以 P 被安排擦除，然后 L 被写入，即。映射到另一个物理的擦除块 P1，所以 P1 被写入，然后不干净的重启发生。结果-有两个物理的 eraseblock P 和 P1 对应同一个逻辑的 eraseblock L。但是 P1 的序列号更大，所以 UBI 在连接 flash 时选择 P1。
2. UBI 不时地将逻辑擦除块移动到其他物理擦除块，以达到损耗均衡的目的。例如，如果 UBI 将 L 从 P 移动到 P1，在 P 被物理擦除之前会发生不干净的重启，有两个物理擦除块 P 和 P1 对应于 L, UBI 必须在 flash 连接时选择其中一个。**@sqnum** 字段表示哪个 PEB 是原始的 (显然 P 的 **@sqnum** 更低) 和副本。但是选择具有更高序列号的物理擦除块是不够的，因为不干净的重新引导可能发生在复制过程的中间，因此 P 中的数据被损坏（P->P1 没复制完）。仅仅选择序号较低的物理擦除块是不够的，因为那里的数据可能很旧 (考虑在复制之后向 P1 添加更多数据

的情况)。此外，不干净的重启可能发生在擦除 P 刚刚开始的时候，所以它会导致不稳定的 P，“大部分” 是 OK 的，但仍然有不稳定的情况。

UBI 使用 **@copy_flag** 字段表示这个逻辑擦除块是一个副本。UBI 还计算数据的 CRC，当数据被移动时，并将其存储在副本 (P1) 的 **@data_crc** 字段。因此，当 UBI 需要从两个 (P 或 P1) 中选择一个物理擦除块时，会检查新块 (P1) 的 **@copy_flag**。如果它被清除，情况就简单了，新的就会被选中。如果设置了该值，则检查副本 (P1) 的数据 CRC。如果 CRC 校验和是正确的，这个物理擦除块被选中 (P1)。否则，将选择较老的 P。

如果是静态卷，**@data_crc** 字段包含逻辑擦除块内容的 CRC 校验和。对于动态卷，它不包含 CRC 校验和规则。唯一的例外情况是，当物理擦除块的数据被磨损均衡子系统移动时，磨损均衡子系统计算数据 CRC，并将其存储在 **@data_crc** 字段中。

@used_ebs 字段仅用于静态卷，它表示该卷的数据需要多少个擦除块。对于动态卷，这个字段不被使用并且总是包含 0。

@data_pad 在创建卷时使用对齐参数计算。因此，@data_pad 字段有效地减少了该卷的逻辑擦除块的大小。当一个人在 UBI 卷上使用面向块的软件 (比如，cramfs) 时，这是非常方便的。

LEB 与 PEB

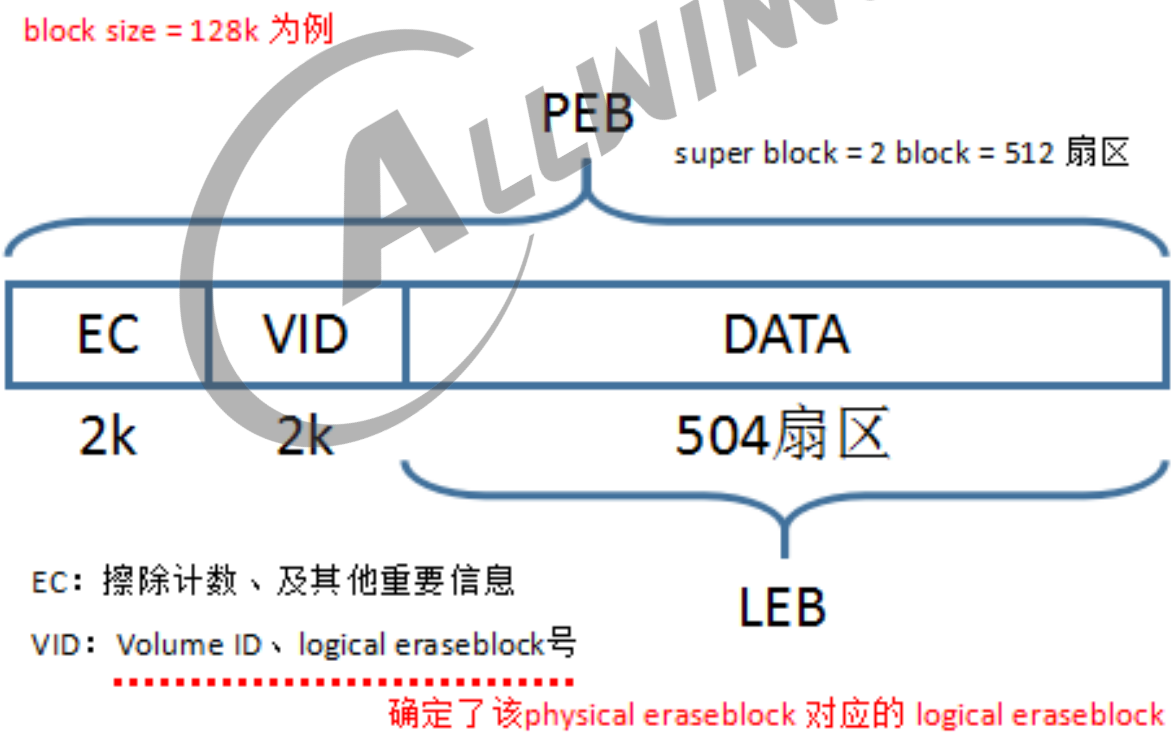

图 3-2: PEB-LEB

3.4 关键接口说明

3.4.1 MTD 层接口

3.4.1.1 aw_rawnand_mtd_erase

```
static int aw_rawnand_mtd_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

3.4.1.2 aw_rawnand_mtd_read

```
static int aw_rawnand_mtd_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

3.4.1.3 aw_rawnand_mtd_read_oob

```
static int aw_rawnand_mtd_read_oob(struct mtd_info *mtd, loff_t from, struct mtd_oob_ops *ops)
```

description： mtd read data with oob

@mtd： MTD device structure

@ops： oob eperation descrition structure

return： success return max_bitflips，fail return fail code

3.4.1.4 aw_rawnand_mtd_write

```
static int aw_rawnand_mtd_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

3.4.1.5 aw_rawnand_mtd_write_oob

```
static int aw_rawnand_mtd_write_oob(struct mtd_info *mtd, loff_t to, struct mtd_oob_ops *ops)
```

description： write data with oob

@mtd： MTD device structure

@to： offset to MTD device

@ops： oob operation descrition structure

return： success return 0, fail return code fail

3.4.1.6 aw_rawnand_mtd_block_isbad

```
static int aw_rawnand_mtd_block_isbad(struct mtd_info *mtd, loff_t ofs)
```

description： check block is badblock or not

@mtd： MTD device structure

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

return： true if the block is bad, or false if the block is good

3.4.1.7 aw_rawnand_mtd_block_markbad

```
static int aw_rawnand_mtd_block_markbad(struct mtd_info *mtd, loff_t ofs)
```

description：mark block at the given offset as bad block

@mtd：MTD device structure

@ofs： offset the mtd device start

return：success to mark return 0, or fail return fail code.

3.4.2 物理层接口

3.4.2.1 aw_rawnand_chip_read_page

```
int aw_rawnand_chip_read_page(struct mtd_info *mtd, struct aw_nand_chip *chip,
    uint8_t *mdata, int mlen, uint8_t *sdata, int slen, int page)
```

description：Read physics on a page

@mtd：MTD device structure

@chip：See 3.3.2

@mdata：要读出数据缓存

@mlen：要读出数据长度

@sdata：要读出 spare 区数据缓存

@slen：要读出 pare 区数据长度

@page：要读取的目标 page

return：zero on success, else a negative error code

3.4.2.2 aw_rawnand_chip_write_page

```
int aw_rawnand_chip_write_page(struct mtd_info *mtd, struct aw_nand_chip *chip,
    uint8_t *mdata, int mlen, uint8_t *sdata, int slen, int page)
```

description：Write physics on a page

@mtd：MTD device structure

@chip：See 3.3.2

@mdata：要读出数据缓存

@mlen：要读出数据长度

@sdata：要读出 spare 区数据缓存

@slen：要读出 pare 区数据长度

@page：要读取的目标 page

return：zero on success, else a negative error code

3.4.2.3 aw_rawnand_chip_erase

```
int aw_rawnand_chip_erase(struct mtd_info *mtd, int page)
```

description：Erase physics on a block

@mtd：MTD device structure

@page：擦除 page 对应的 block

return：zero on success, else a negative error code

3.4.2.4 aw_rawnand_chip_block_bad

```
int aw_rawnand_chip_block_bad(struct mtd_info *mtd, int block)
```

description：aw_rawnand_chip_simu_block_bad - read bad block marker from the chip

@mtd: MTD device structure

@block: simu block offset from device start simu block

return： BBT_B_BAD return 1，BBT_B_GOOD return 0

3.4.2.5 aw_rawnand_chip_block_markbad

```
int aw_rawnand_chip_block_markbad(struct mtd_info *mtd, int block)
```

description：aw_rawnand_chip_block_markbad - mark a block bad in mark pos and update bbt&bbtd

@mtd: MTD device structure

@block: block offset from device start

return： zero on success, else a negative error code

3.4.3 Uboot 应用接口

3.4.3.1 sunxi_flash_nand_probe

```
static int sunxi_flash_nand_probe(void)
```

description： MTD layer and SPINAND || RAWNAND initialization，Set the storage type。

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

3.4.3.2 sunxi_flash_nand_init

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

description： MTD layer and SPINAND || RAWNAND initialization。

boot_mode： Working mode

res： The default is 0

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

3.4.3.3 sunxi_flash_nand_exit

```
int spinand_mtd_exit(void)
```

description： Release registration is a resource for applications.

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

3.4.3.4 sunxi_flash_nand_write

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

description： mtd write data interface.

start_block： want write start block

nblock： want write block count

buffer： data buffer

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

3.4.3.5 sunxi_flash_nand_read

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

description： mtd readdata interface.

start_block： want read start block

nblock： want read block count

buffer： data buffer

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

3.4.3.6 sunxi_flash_nand_erase

```
static int sunxi_flash_nand_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.

3.4.3.7 sunxi_flash_nand_force_erase

```
int spinand_mtd_force_erase(void)
```

description： erase boot & partition data.

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

3.4.3.8 sunxi_flash_nand_flush

```
int ubi_nand_flush(void)
```

description： Flush physical cache data to flash.

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

3.4.3.9 sunxi_flash_nand_download_spl

```
static int sunxi_flash_nand_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.

3.4.3.10 sunxi_flash_nand_download_toc

```
static int sunxi_flash_nand_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 模块配置

4.1 uboot 模块配置

```
Device Drivers-->Sunxi flash support-->
[*]Support sunxi nand devices
[*]Support sunxi nand ubifs devices
[*]Support COMM NAND V1 interface
```

如下图：

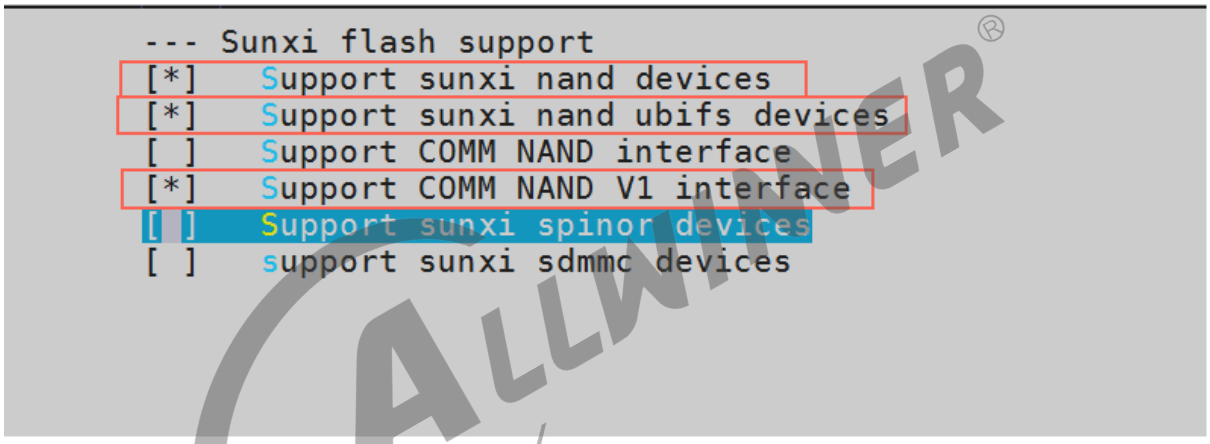

图 4-1: u-boot-spinand-menuconfig

4.2 kernel 模块配置

4.2.1 内核版本 ≤Linux5.4

```
Device Drivers->Memory Technology Device(MTD) support-->sunxi-nand
```

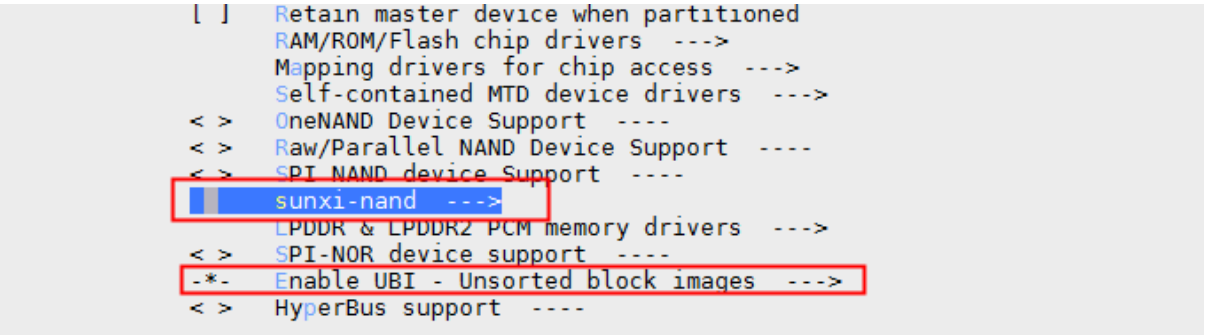

图 4-2: UBI

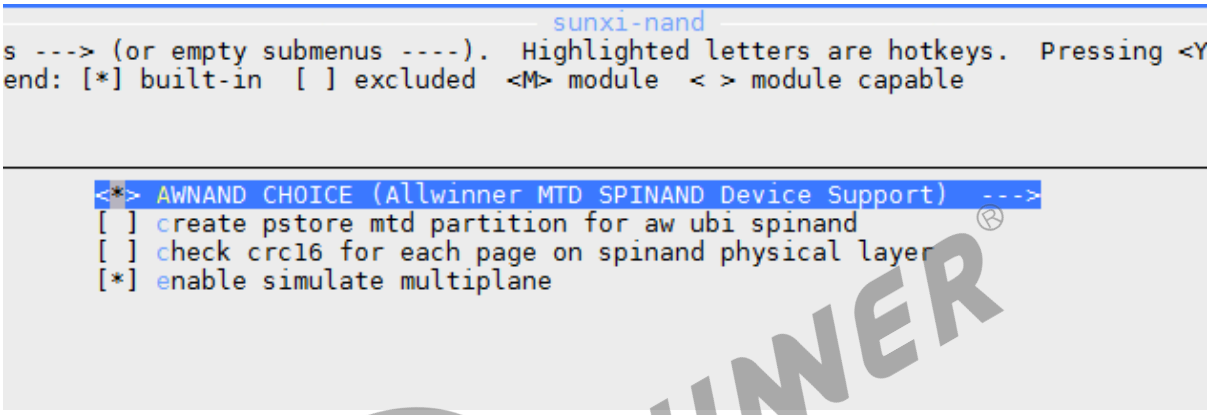

图 4-3: ker_nand-cfg

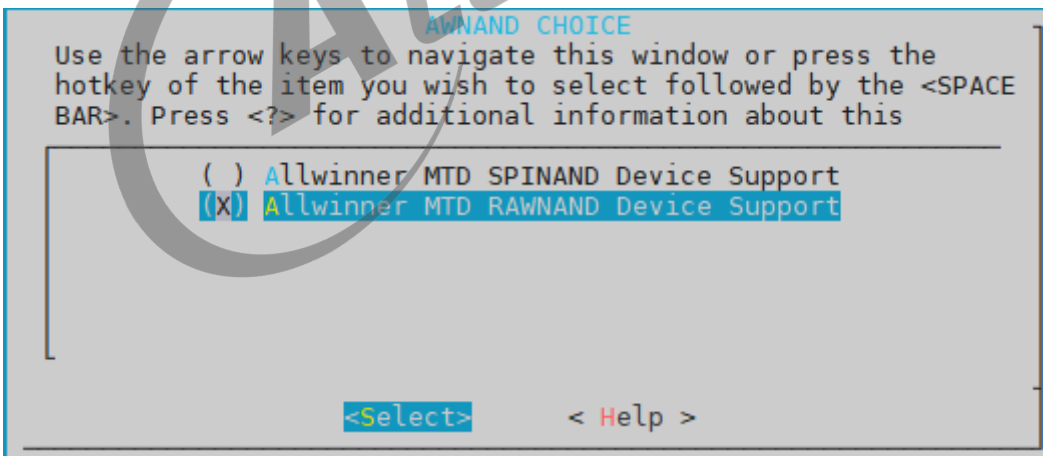

图 4-4: ker_spinand

File systems-->Miscellaneous filesystems-->

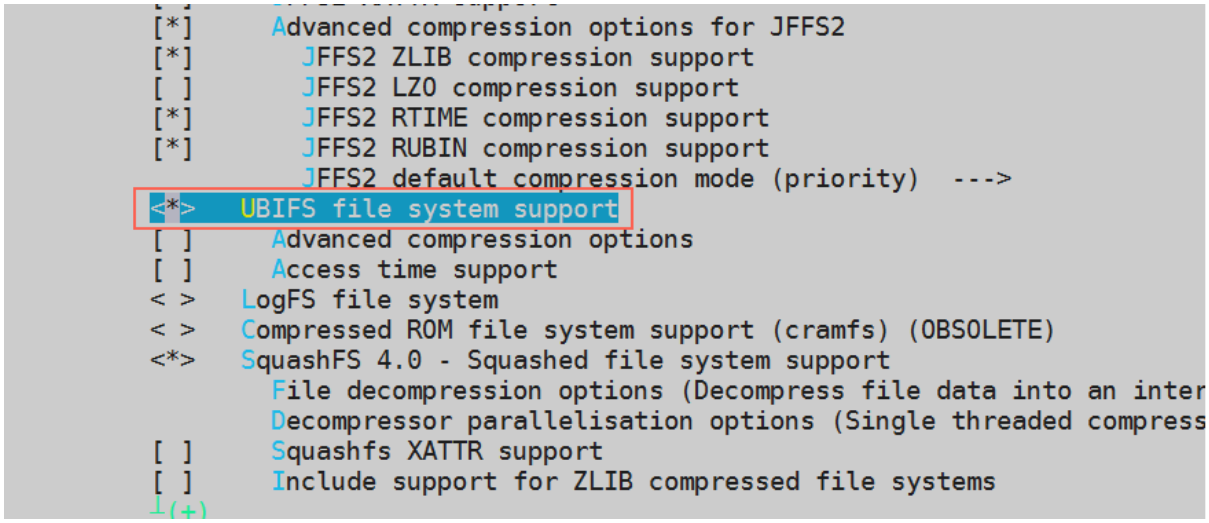

图 4-5: menuconfig_spinand_ubifs

4.2.2 内核版本 ≥Linux5.10

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

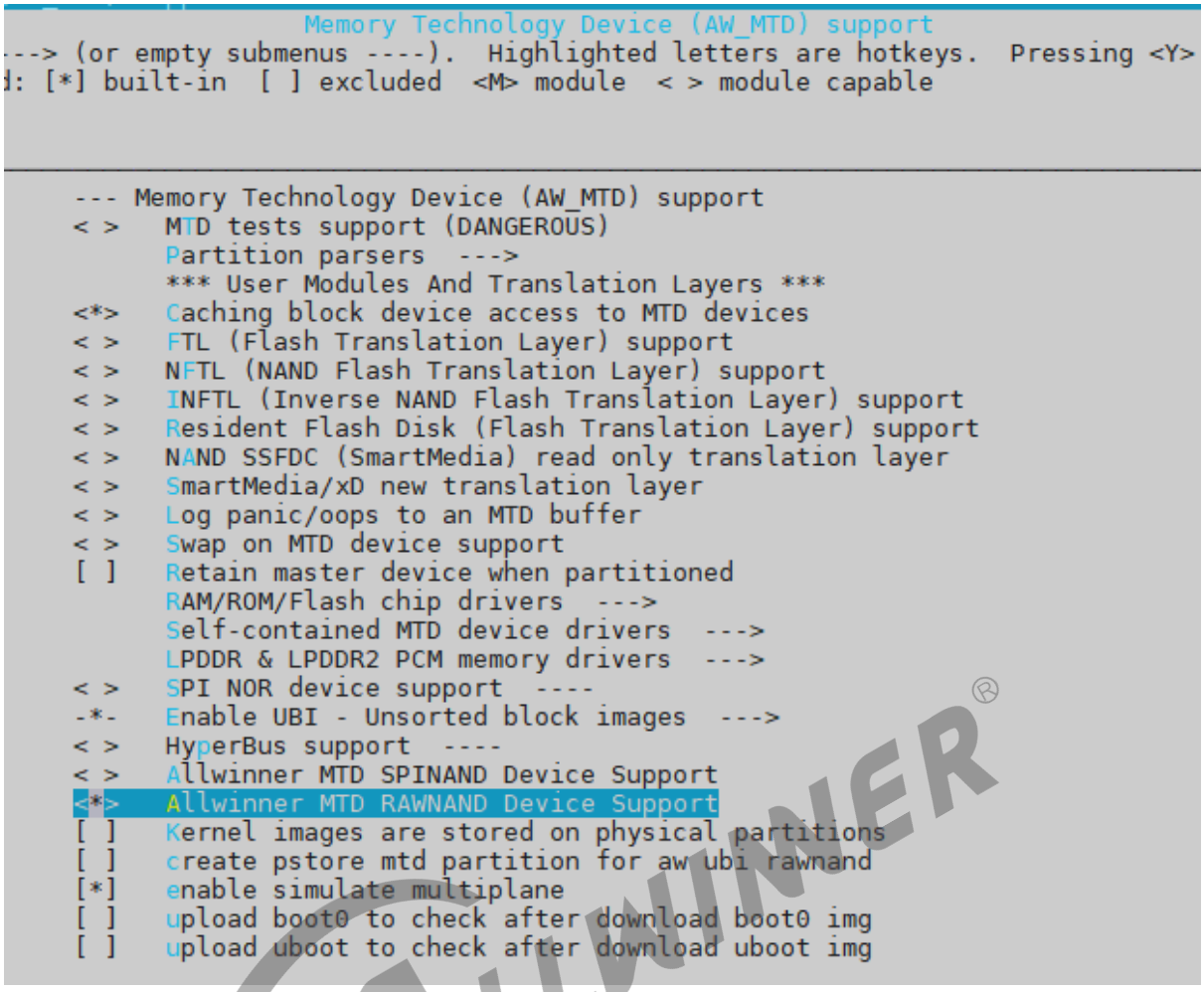

图 4-6: rawnand-config

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

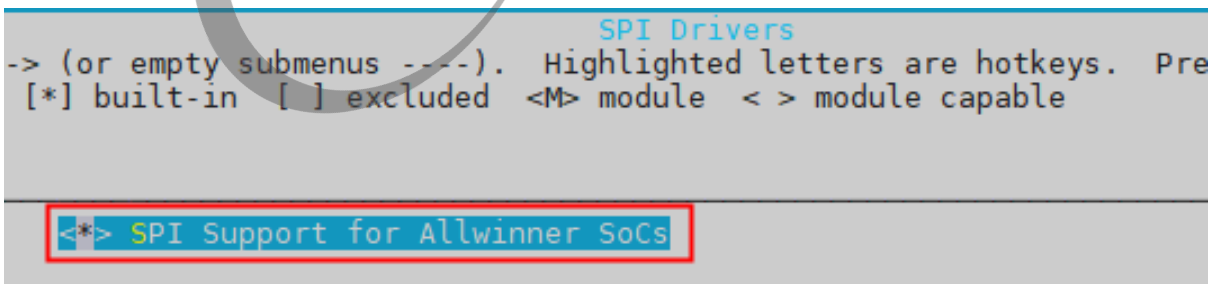

图 4-7: linxu5.10-menuconfig-spi

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

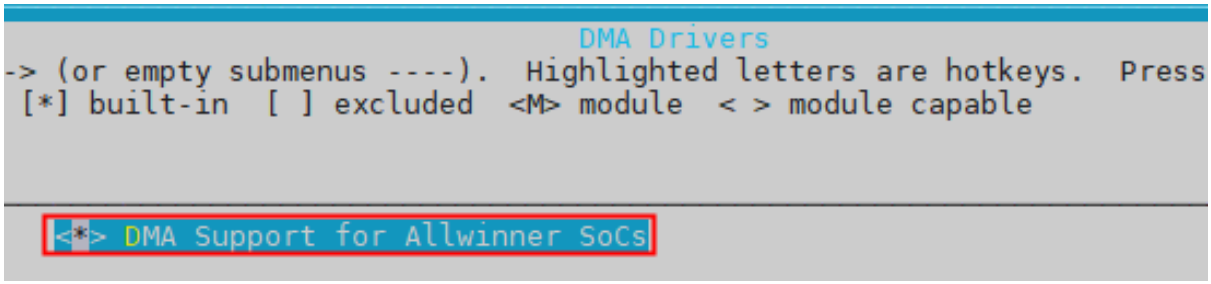

图 4-8: linxu5.10-menuconfig-dma

File systems-->Miscellaneous filesystems-->

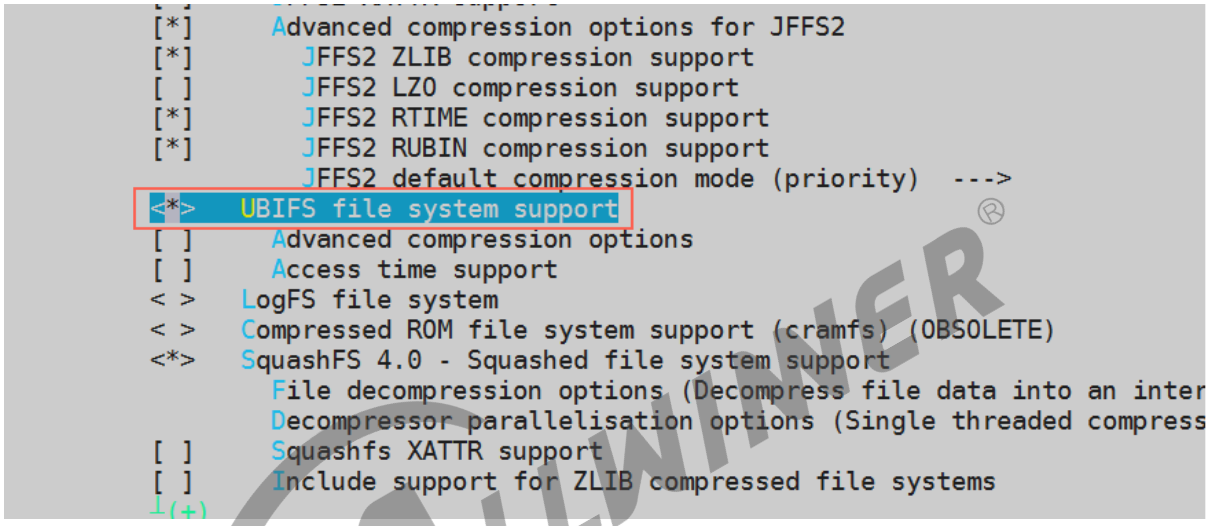

图 4-9: menuconfig_spinand_ubifs

4.3 env.cfg

在 env.cfg 中添加修改下值，setargs_nand_ubi 先 copy 一份 setargs_nand 再添加对应变量

路径：device/config/chips/平台（v833）/configs/default/env.cfg

```
nand_root=ubi0_4
mtd_name=sys
rootfstype=ubifs,rw
setargs_nand_ubi=setenv bootargs ubi.mtd=${mtd_name}
                                 rootfstype=${rootfstype}
```

图 4-10: build-mkcmd

4.4 Soc 级设备树配置

Soc 级设备树保存的是该类芯片所有平台的模块配置，不同内核版本 Soc 级设备树所在路径不同，但对于每一个 nand 控制器来说，在设备树中配置参数相似，平台设备树文件路径为：

- 内核版本 ≤Linux-5.4

kernel/linux-x.x/arch/armxx/boot/dts/sunxi/CHIP.dtsi(CHIP为研发代号，如sun50iw10p1等)。

- 内核版本 ≥Linux5.10

bsp/configs/内核版本/CHIP.dtsi(CHIP为研发代号，如sun55iw3p1等)：

```
nand0:nand0@04011000 {
        compatible = "allwinner,sun55iw3-nand";
        device_type = "nand0";
        reg = <0x0 0x04011000 0x0 0x1000>;/* nand0 */
        interrupts = <GIC_SPI 38 IRQ_TYPE_LEVEL_HIGH>;
        clocks = <&ccu CLK_PLL_PERI1_400M>,
          <&ccu CLK_NAND0_CLK0>,
           <&ccu CLK_NAND0_CLK1>,
          <&ccu CLK_NAND0>,
           <&ccu CLK_NAND_MBUS_GATE>;
        clock-names = "pll_periph","mclk","ecc", "bus", "mbus";
        resets = <&ccu RST_BUS_NAND>;
        reset-names = "rst";
        nand0_regulator1 = "none";
        nand0_regulator2 = "none";
        nand0_cache_level = <0x55aaaa55>;
        nand0_flush_cache_num = <0x55aaaa55>;
        nand0_capacity_level = <0x55aaaa55>;
        nand0_id_number_ctl = <0x55aaaa55>;
        nand0_print_level = <0x55aaaa55>;
        nand0_p0 = <0x55aaaa55>;
        nand0_p1 = <0x55aaaa55>;
        nand0_p2 = <0x55aaaa55>;
        nand0_p3 = <0x55aaaa55>;
        chip_code = "sun55iw3";
        boot_crc = "okay";
        status = "okay";
    };
```

4.5 board 级设备树配置

配置文件路径为：/device/config/chips/{IC}/configs/{BOARD}/board.dts, 用于保存每一个板级平台设备差异化的信息的补充。里面的配置信息会覆盖上面 Soc 级默认配置信息。

4.5.1 引脚 PINMUX 配置

```
nand0_pins_default: nand0@0 {
    pins = "PC0", "PC1", "PC2", "PC5",
        "PC8", "PC9", "PC10", "PC11",
        "PC12", "PC13", "PC14", "PC15",
        "PC16";
    function = "nand0";
    drive-strength = <30>; //IO驱动能力
};

nand0_pins_rb: nand0@1 {
    pins = "PC4", "PC6", "PC3", "PC7";
    function = "nand0";
    drive-strength = <30>;
    bias-pull-up;   /* only RB&CE should be pulled up */
};

nand0_pins_sleep: nand0@2 {
    pins = "PC0", "PC1", "PC2", "PC3",
        "PC4", "PC5", "PC6", "PC7",
        "PC8", "PC9", "PC10", "PC11",
        "PC12", "PC13", "PC14", "PC15",
        "PC16";
    function = "io_disabled";
    drive-strength = <10>;
};
```

4.5.2 nand 设备节点配置

```
&nand0 {
    compatible = "allwinner,sun55iw3-nand";
    device_type = "nand0";
    pinctrl-names = "default", "sleep";
    pinctrl-0 = <&nand0_pins_default &nand0_pins_rb>;
    pinctrl-1 = <&nand0_pins_sleep>;
    nand0_regulator1 = "vcc-nand";
    nand0_regulator2 = "none";
    chip_code = "sun55iw3";
    status = "okay";
};
```

设备树配置参数说明：

配置项	功能
pinctrl-0	默认的引脚配置状态，详见与 CHIP.dtsi 同级目录下的 CHIP-pinctrl.dtsi，用户可通过 pinctrl 中修改引脚驱动能力，上下拉，一般 nand CE、RB 引脚默认上拉
pinctrl-1	休眠时的引脚状态，一般不需要用户修改
nand0_regulator1	VCC-NAND 电源配置，根据实际硬件原理图修改，可配置在对应的 board.dts 中

配置项	功能
nand0_regulator2	VCCQ 电源配置，根据实际硬件原理图修改，可配置在对应的 board.dts 中
nand0_cache_level	默认 0x55aaaa55，用于调整算法 cache 数量，不建议用户修改
nand0_capacity_level	默认 0x55aaaa55，每个分区的保留比例是十分之一；改为 1：1. 每个分区的保留比例由十分之一改为十三分之一。2. 重负载下随机写速度会降低。不建议用户修改
nand0_flush_cache_num	默认 0x55aaaa55，不建议用户修改
nand0_id_number_ctl	默认 0x55aaaa55，修改 two plane，interleave、dual channel 配置，不建议用户修改
nand0_print_level	默认 0x55aaaa55，修改算法打印等级，不建议用户修改
nand0_px	配合 nand0_id_number_ctl 一起使用，修改 flash 频率，two plane，interleave、dual channel 配置，不建议用户修改
chip_code	绑定平台，不建议用户修改
boot_crc	“disabled” 关闭启动检测逻辑页 crc 功能，默认打开

5 使用案例

在 ubi 卷上模拟 mtdblock 设备，挂载块设备文件系统

1. 在 sys_partition*.fex 中添加分区（大小要求对齐到 504 扇区）；
2. 在内核配置中打开 CONFIG_MTD_BLOCK、CONFIG_MTD_UBI_GLUEBI；
3. 编译、打包、烧录固件；
4. 对应的块设备为/dev/mtdblock* ，具体序号可以从后往前对应 sys_partition*.fex 文件中的分区
5. 如果 sys_partition*.fex 中没有指定 downloadfile，挂载前需要格式化：mkfs.vfat /dev/mtdblock12
6. 挂载分区：mkdir /mnt/test1 & mount -t vfat /dev//dev/mtd12 /mnt/test1；

6 常见问题排查

参考《NAND 量产问题 _ 排查指南》和《NAND 硬件 _ 排查指南 v0.4》

著作权声明

商标声明

免责声明