



SPINAND UBI 离线烧录 开发指南

版本号: 2.2
发布日期: 2024.10.28

版本历史

版本号	日期	制/修订人	内容描述
2.0	2021.04.08	AWA1669	建立初始版本
2.1	2022.03.21	AWA1543	SPINAND UBI 离线烧录器
2.2	2024.10.28	AWA1669	根据是否为 MULTIPLANE 方案，说明 LEB 的计算方式



目 录

1 概述	1
1.1 编写目的	1
1.2 适用范围	1
1.3 相关人员	1
2 名词解释	2
3 总体数据布局	3
4 toc0 or boot0	6
4.1 input file	6
4.2 flow	6
4.3 normal boot0	7
4.4 secure boot0	8
4.5 filling storage_data	9
4.6 update checksum	11
4.7 burn boot0	11
5 toc1 or uboot	12
6 secure storage block	13
7 计算逻辑区域 LEB 总数	14
8 动态调整 sunxi_mbr 卷	15
9 根据 sunxi_mbr 动态生成 ubi layout volume	16
10 烧写逻辑卷	18
10.1 ubi_ec_hdr	18
10.2 ubi_vid_hdr	19
11 数据对齐	21

插 图

图 3-1	ubi_scheme_p_i_l	3
图 3-2	ubi_scheme_p_i_l	4
图 4-1	boot0_head	6
图 4-2	boot_head	7
图 4-3	storage_data	9



1 概述

1.1 编写目的

介绍 Sunxi SPINand 烧写时的数据布局

1.2 适用范围

本设计适用于所有 SPINAND-UBI 方案平台

1.3 相关人员

制定烧录器客户与烧录器厂商参考



2 名词解释

词	义
UBI	unsorted block image
PEB	physical erase block
LEB	logical erase block

PEB 和 logical block 关系

1 PEB = 1 logical block

1 logical block = 2 physical blocks



3 总体数据布局

- ubi 方案 FLASH 上的数据布局

image item	download file	填充方式	地址属性	备份	备注
secure boot0	boot0 header	动态填充	固定物理地址 block0-block3	多备份 各个备份按 block 对齐（如果 boot0 超过 1 个 block, 单个备份起始 block 地址为偶数）若写单个备份过程中遇到坏块，则中止当前备份写过程，写下一备份即可	如果是非安全方案，那么此处为 Normal boot0
	toc0.fex	静态			
normal boot0	boot0 header	动态填充	固定物理地址 block4-block7	多备份 各个备份按 block 对齐，若写单个备份过程中遇到坏块，则跳过该坏块，写入下一好块，直到将当前备份完整写入	请找我司 boot 小组同事提供支持安全方案烧录器的 normal boot0fex 文件
	boot0_nand.fex	静态			
uboot	boot_package.fex	静态	固定物理地址 block8-block31	多备份 各个备份按 block 对齐，若写单个备份过程中遇到坏块，则跳过该坏块，写入下一好块，直到将当前备份完整写入	1. 备份个数与 uboot 大小有关 2. 如果是安全方案，uboot 需要在第一次启动时把 normal boot0 替换为 secure boot0
	phyinfo_bufile	对于 ubi 方案无，请忽略			
secure storage block	无	对于 ubi 方案无，请忽略	block32-block39, 2 个物理 block, 地址动态计算，但最大 block 号为 39		uboot 后 2 个好的物理 block, 不要求连续

图 3-1: ubi_scheme_p_i_l

mbr 卷 (volume)	sunxi_mbr .fex	静态	逻辑地址 ,从逻辑块 20 (物理 块 40) 开 始映射	无	不用转换为 GPT 格式, 但 UDISK 的 size 需要动态计 算
ubi layout volume	无	动态生成		2 个备份	ubi 内部私有卷, 需由 sunxi_mbr 转化
env 卷	env.fex	静态		无	
env- redund 卷	env.fex	静态		无	
bootA 卷	boot.fex	静态		无	
rootfsA 卷	rootfs.fe x	静态		无	
bootB 卷	boot.fex	静态		无	
rootfsB 卷	rootfs.fe x	静态		无	
UDISK 卷	data_ubif s.fex	静态		无	

Table. ubi nand overview

图 3-2: ubi_scheme_p_i_l

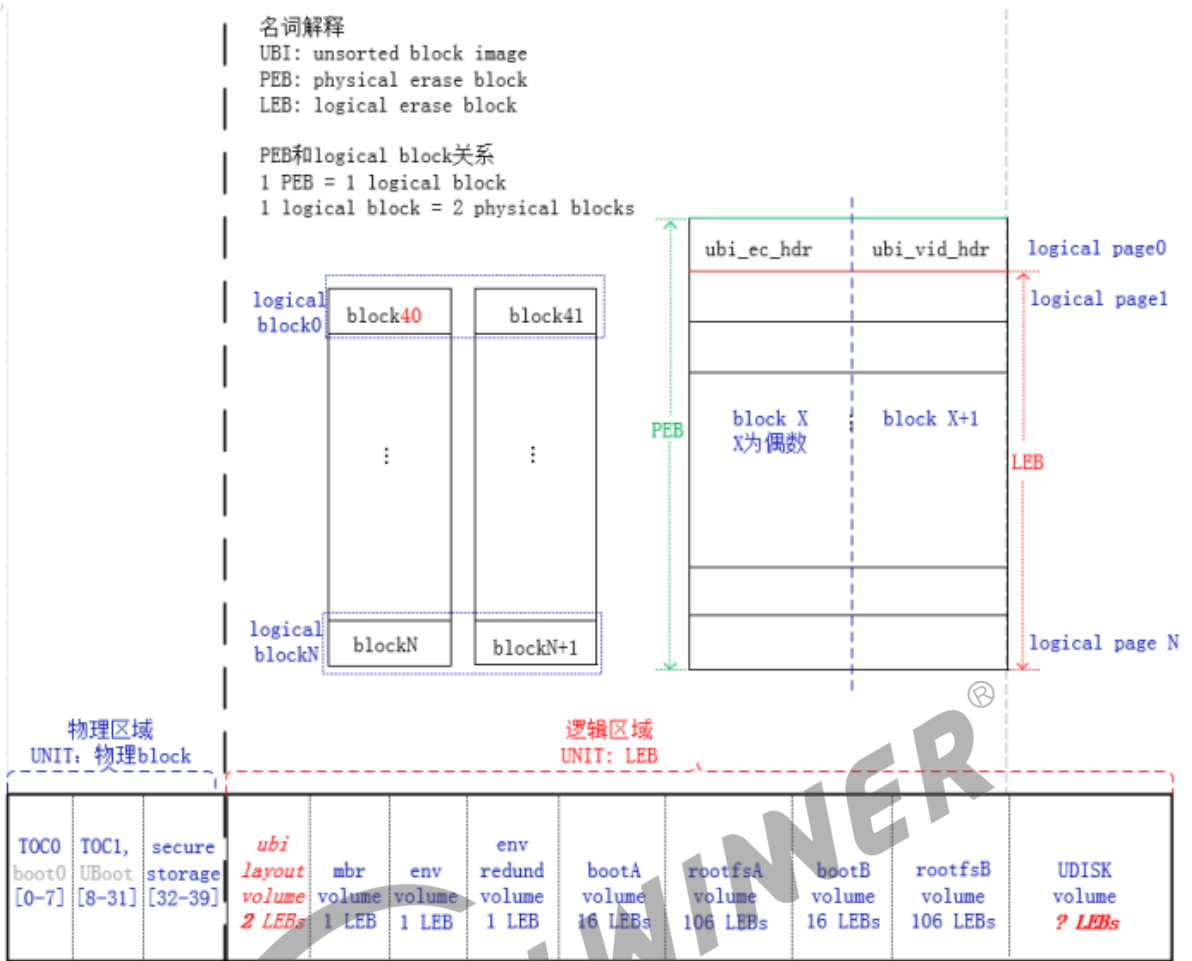


Figure. ubi nand布局

sys_partition.fex 文件中的各个分区大小会按照 LEB 大小对齐，sunxi_mbr 分区概念与 UBI 卷 (volume) 概念相同

需要修改原镜像文件：物理区 TOC0 合逻辑区 sunxi_mbr.fex

需要动态生成文件：逻辑区 ubi layout volume

注意：

1. 各分区镜像以实际应用为准
2. logical page0 = logical block 的两个 page0

4 toc0 or boot0

4.1 input file

boot0_nand.fex（非安） or toc0.fex（安全）

4.2 flow

- 验证 checksum 是否准确
- 填充 storage_data
- 重新生成 checksum 并更新 boot_file_head_t 中的 check_sum

```
typedef struct boot0_file_head_t
{
    boot_file_head_t boot_head;
    boot0_private_head_t prvt_head;
    char hash[64];
} boot0_file_head_t;
```

图 4-1: boot0_head

```

#define BOOT0_MAGIC          "eGON.BT0"
#define SYS_PARA_LOG         0x4d415244

/*****
/*          file head of Boot          */
*****/
typedef struct _Boot_file_head
{
    _u32 jump_instruction; /* one intruction jumping to real code */
    _u8  magic[MAGIC_SIZE]; /* ="eGON.BT0" */
    _u32 check_sum;        /* generated by PC */
    _u32 length;           /* generated by PC */
    _u32 pub_head_size;    /* the size of boot_file_head_t */
    _u8  pub_head_vsn[4];  /* the version of boot_file_head_t */
    _u32 ret_addr;        /* the return value */
    _u32 run_addr;        /* run addr */
    _u32 boot_cpu;        /* eGON version */
    _u8  platform[8];     /* platform information */
}boot_file_head_t;

```

图 4-2: boot_head

参考文件

include/private_boot0.h

sprite/sprite_download.c

参考函数

download_normal_boot0

download_secure_boot0

4.3 normal boot0

normal boot0 存放于 block4-7

参考 function: download_normal_boot0

```

typedef struct _boot0_file_head_t
{
    boot_file_head_t boot_head;
    boot0_private_head_t prvt_head;
    char hash[64];
    __u8 reserved[8];
    union {

```

```

#ifdef CFG_SUNXI_SELECT_DRAM_PARA
    boot_extend_head_t  extd_head;
#endif
    fes_aide_info_t fes1_res_addr;

    } fes_union_addr;
}boot0_file_head_t;
/*****
/*          file head of Boot0          */
/*****
typedef struct _boot0_private_head_t
{
>-----__u32          prvt_head_size;
>-----/*debug_mode = 0 : do not print any message,debug_mode = 1 ,print debug message*/
>-----__u8          debug_mode;
>-----/*0:axp, 1: no axp */
>-----__u8          power_mode;
>-----__u8          reserve[2];
>-----/*DRAM patameters for initialising dram. Original values is arbitrary*/
>-----unsigned int   dram_para[32];
>-----/*uart: num & uart pin*/
>-----__s32>----->----->----->-----uart_port;
>-----normal_gpio_cfg   uart_ctrl[2];
>-----/* jtag: 1 : enable, 0 : disable */
>-----__s32          enable_jtag;
    normal_gpio_cfg>-----   jtag_gpio[5];
>-----/* nand/mmc pin*/
    normal_gpio_cfg   storage_gpio[32];
>-----/*reserve data*/
    char               storage_data[512 - sizeof(normal_gpio_cfg) * 32];
}boot0_private_head_t;

```

4.4 secure boot0

secure boot0 存放于 boot0-block3

```

typedef struct sbrom_toc0_config
{
    unsigned char   config_vsn[4];
    unsigned int    dram_para[32]; // dram参数
    int             uart_port;     // UART控制器编号
    normal_gpio_cfg uart_ctrl[2];  // UART控制器GPIO
    int             enable_jtag;   // JTAG使能
    normal_gpio_cfg jtag_gpio[5];  // JTAG控制器GPIO
    normal_gpio_cfg storage_gpio[50]; // 存储设备 GPIO信息
                                     // 0-23放nand, 24-31存放卡0, 32-39放卡2
                                     // 40-49存放spi
    char            storage_data[384]; // 0-159,存储nand信息; 160-255,存放卡信息
    unsigned int    secure_dram_mbytes; //
    unsigned int    drm_start_mbytes; //
    unsigned int    drm_size_mbytes; //
    unsigned int    boot_cpu; //
    special_gpio_cfg a15_power_gpio; //the gpio config is to a15 extern power enable gpio
    unsigned int    next_exe_pa;
    unsigned int    secure_without_OS; //secure boot without semelis
    unsigned char   debug_mode; //1:turn on printf; 0 :turn off printf
    unsigned char   power_mode; // 0:axp, 1: dummy pmu */

```

```

unsigned char  rotpk_flag;
unsigned char  reserver[1];
unsigned int    card_work_mode;
unsigned int    res[2];      // 总共1024字节
}
sbrom_toc0_config_t;
    
```

4.5 filling storage_data

```

typedef struct
{
    __u8  ChipCnt;           // the count of the total nand flash chips are currently connecting on the CE pin
    __u8  ConnectMode;      // the rb connect mode
    __u8  BankCntPerChip;   // the count of the banks in one nand chip, multiple banks can support inter-leave
    __u8  DieCntPerChip;    // the count of the dies in one nand chip, block management is based on die
    __u8  PlaneCntPerDie;   // the count of planes in one die, multiple planes can support multi-plane operation
    __u8  SectorCntPerPage; // the count of sectors in one single physic page, one sector is 0.5k
    __u16 ChipConnectInfo;  // chip connect information, bit == 1 means there is a chip connecting on the CE pin
    __u32 PageCntPerPhyBlk; // the count of physic pages in one physic block
    __u32 BlkCntPerDie;     // the count of the physic blocks in one die, include valid block and invalid block
    __u32 OperationOpt;    // the mask of the operation types which current nand flash can support support
    __u32 FrequencePar;     // the parameter of the hardware access clock, based on 'MHz'
    __u32 SpiMode;         // spi nand mode, 0: mode 0, 3: mode 3
    __u8  NandChipId[8];    // the nand chip id of current connecting nand chip
    __u32 pagewithbadflag;  // bad block flag was written at the first byte of spare area of this page
    __u32 MultiPlaneBlockOffset; // the value of the block number offset between the two plane block
    __u32 MaxEraseTimes;    // the max erase times of a physic block
    __u32 MaxEccBits;       // the max ecc bits that nand support
    __u32 EccLimitBits;    // the ecc limit flag for the nand
    __u32 uboot_start_block;
    __u32 uboot_next_block;
    __u32 logic_start_block;
    __u32 nand_specialinfo_page;
    __u32 nand_specialinfo_offset;
    __u32 physic_block_reserved;
    __u32 Reserved[4];
} boot_spinand_para_t;
} // end [anorboot_spinand_para_t]
    
```

图 4-3: storage_data

下表中红色字体不能配置错，大部分值直接参考 drivers/mtd/awnand/spinand/physic/id.c

attribute name	type	value	comment
ChipCnt	unsigned char	1	
ConnectMode	unsigned char	1	忽略，可以不用理解
BankCntPerChip	unsigned char	1	忽略，可以不用理解
DieCntPerChip	unsigned char	1	
PlaneCntPerDie	unsigned char	2	忽略，可以不用理解
SectorCntPerPage	unsigned char	4	以具体物料为准，常见为 4
ChipConnectInfo	unsigned short	1	忽略，可以不用理解
PageCntPerPhyBlk	unsigned int	64	以具体物料为准，常见为 64
BlkCntPerDie	unsigned int	1024	以具体物料为准，常见为 1024，也可能为 512 或 2048
OperationOpt	unsigned int	0x?	参考 id.c 各个物料配置
FrequencePar	unsigned int	100	忽略，可以不用理解
SpiMode	unsigned int	0	忽略，可以不用理解

attribute name	type	value	comment
NandChipId[8]	unsigned char	0x?	参考 id.c
pagewithbadflag	unsigned int	0	忽略, 可以不用理解
MultiPlaneBlockOffset	unsigned int	1	忽略, 可以不用理解
MaxEraseTimes	unsigned int		忽略, 可以不用理解
EccLimitBits	unsigned int		忽略, 可以不用理解
uboot_start_block	unsigned int	8	
uboot_next_block	unsigned int	40	
logic_start_block	unsigned int	40	忽略, 可以不用理解
nand_specialinfo_page	unsigned int	0	忽略, 可以不用理解
nand_specialinfo_offset	unsigned int	0	忽略, 可以不用理解
physic_block_reserved	unsigned int	0	忽略, 可以不用理解
Reserved[4]	unsigned int	0	忽略, 可以不用理解

以 GigaDevice GD5F1GQ4UBYIG spinand 为例, 其大部分信息直接来自 id.c [®]

```
{
    .Model      = "GD5F1GQ4UBYIG",
    .NandID     = {0xc8, 0xd1, 0xff, 0xff, 0xff, 0xff, 0xff, 0xff},
    .DieCntPerChip = 1,
    .SectCntPerPage = 4,
    .PageCntPerBlk = 64,
    .BlkCntPerDie = 1024,
    .OobSizePerPage = 64,
    .OperationOpt = SPINAND_QUAD_READ | SPINAND_QUAD_PROGRAM |
        SPINAND_DUAL_READ,
    .MaxEraseTimes = 50000,
    .EccFlag      = HAS_EXT_ECC_SE01,
    .EccType      = BIT4_LIMIT5_TO_7_ERR8_LIMIT_12,
    .EccProtectedType = SIZE16_OFF4_LEN8_OFF4,
    .BadBlockFlag = BAD_BLK_FLAG_FRIST_1_PAGE,
},
```

参考文件:

include/linux/mtd/aw-spinand.h /定义 id.c 中 id 表的数据结构/

drivers/mtd/awnand/spinand/sunxi-spinand.h /定义 boot_spinand_para_t 填充的数据结构/

drivers/mtd/awnand/spinand/sunxi-driver.c /填充函数参考/

drivers/mtd/awnand/spinand/physic/id.c /不同物料的信息配置 (id 表配置) /

参考函数:

ubi_nand_get_flash_info->spinand_mtd_get_flash_info

4.6 update checksum

参考文件：

```
sprite/sprite_download.c
```

```
sprite/sprite_verify.c
```

```
board/sunxi/board_common.c
```

参考函数流程：

```
download_normal_boot0/download_secure_boot0 -> sunxi_sprite_generate_checksum -> sunxi_generate_checksum
```

4.7 burn boot0

• 参考文件：

```
drivers/mtd/awnand/spinand/sunxi-driver.c
```

参考函数流程：

```
spinand_mtd_download_boot0()
```

注意事项：

如果是安全方案，存放 boot0 的 blocks 中一半存放 secure boot0，一半存放 normal boot0, 参考 UBI 方案分区表信息以及第 2 章节说明

各个备份按 block 对齐（如果 boot0 超过 1 个 block, 单个备份起始 block 地址为偶数），若写单个备份过程中遇到坏块，则中止当前备份写过程，写下一备份即可

boot0 的镜像文件已经包含了 boot0 header，不需额外分配组织 boot0 header 格式，只需更新 boot0 header 中的 storage_data 部分，其他属性（比如 dram_para）不需更新。更新后，需重新生成 boot0 header 中的校验和 check_sum

5 toc1 or uboot

区域：block8-block31

直接烧写 toc1 镜像

参考文件：

```
sprite/sprite_download.c
```

```
drivers/sunxi_flash/nand.c
```

```
drivers/sunxi_flash/nand_for_ubi.c
```

```
drivers/mtd/awnand/spinand/sunxi-driver.c
```

参考函数：

```
sunxi_sprite_download_uboot->sunxi_sprite_download_toc->sunxi_flash_nand_download_toc->  
>  
ubi_nand_download_uboot->spinand_mtd_download_uboot
```

6 secure storage block

区域：block32-block39

烧录器不用处理



7 计算逻辑区域 LEB 总数

用户可见 LEB 数 = 总物理块数 - 8 (boot0) - 24 (boot1) - 8 (secure storage) - 20 * 总物理块数/1024 - 4, 规则如下:

1. 减去物理区域块数
2. 减去坏块处理预留数 (每 1024 物理块最多 20 个物理块, 即 10 个逻辑块)
3. 减去 4 (2 个用于 ubi layout volume, 1 个用于 LEB 原子写, 1 个用于磨损均衡处理)

推算方式可以参考 u-boot-2018/cmd/ubi_simu.c 的 ubi_sim_part 和 ubi_simu_create_vol 函数。

正常情况下, ubi 方案 sys_partition.fex 中各个分区的大小会按照 LEB 大小对齐。

假如一款 flash 有 1024 个 block, 每个 block 有 64 个 page, 每个 page 有 2KB, 则逻辑块大小为 $64 * 2K = 128K$, (如果是 MULTIPLANE 方案 $64 * 2K * 2 = 256K$), 那么 PEB 大小是 128K (如果是 MULTIPLANE 方案 = 256K), LEB 大小为 124K (如果是 MULTIPLANE 方案 = 252K), PEB 中的首逻辑页固定用于存放 ubi_ec_hdr 和 ubi_vid_hdr。

由于预先不知道物料的容量信息及预留块信息, 因此 sys_partition.fex (sunxi_mbr.fex) 中最后一个分区的 size 信息默认先填 0, 待 NAND 驱动初始化完成后才知道用户可见 LEB 数有多少个, 此时需要根据信息改写 sunxi_mbr.fex 中最后一个分区的 size。

8 动态调整 sunxi_mbr 卷

sunxi_mbr.fex 共 64k, 共 4 个备份, 每个备份 16K

1. 计算 mbr 卷最后分区 size, 单位: 扇区 (512 字节), 计算规则如下:

根据第 5 章节计算出的用户可见 leb 数转化出总的扇区数 total_sector, 依次减去分区表中各个分区占用的扇区数

2. 回填 sunxi_mbr.fex 最后一个分区 size

3. 重新计算并回填 sunxi_mbr 的 crc32

4. 改写其余 3 个备份

sunxi_mbr_t 结构体: u-boot-2018/include/sunxi_mbr.h, 结构体各个成员均使用小端存储。

```
typedef struct sunxi_mbr
{
    unsigned int crc32;
    unsigned int version;
    unsigned char magic[8];
    unsigned int copy;
    unsigned int index;
    unsigned int PartCount;
    unsigned int stamp[1];
    sunxi_partition array[SUNXI_MBR_MAX_PART_COUNT];
    unsigned int lockflag;
    unsigned char res[SUNXI_MBR_RESERVED];
}attribute ((packed)) sunxi_mbr_t;
```

重新计算并回填 sunxi_mbr crc32 的代码请参考 u-boot-2018/drivers/mtd/aw-spinand/sunxi-ubi.c 的 adjust_sunxi_mbr 函数。

9 根据 sunxi_mbr 动态生成 ubi layout volume

ubi layout volume 可以理解为 UBI 模块内部用的分区信息文件，sunxi_mbr 分区是用于全志烧写 framework 的分区信息文件。二者记录的分区信息本质上是一样的，因此烧写时，可以由 sunxi_mbr 卷转化成 ubi layout volume。

ubi layout volume 由 128 个 struct ubi_vtbl_record (u-boot-2018/drivers/mtd/ubi/ubi-media.h) 组成，结构体各个成员使用大端表示。

```
struct ubi_vtbl_record {
    __be32 reserved_pebs;

    __be32 alignment;

    __be32 data_pad;

    __u8 vol_type;

    __u8 upd_marker;

    __be16 name_len;
    char name[UBI_VOL_NAME_MAX+1];

    __u8 flags;

    __u8 padding[23];

    __be32 crc;
} __packed;
```

attribute name	type	value	comment
reserved_pebs	__be32		卷大小/LEB size, 对于 ubi layout volume, 固定为 2
alignment	__be32	1	
data_pad	__be32	0	
vol_type	__u8	1	动态卷：1，静态卷：2，当前方案均是动态卷
upd_marker	__u8	0	
name_len	__be16		卷名长度
name[128]	char		
flags	__u8		分区内最后一个卷 udisk, flags 为 UBI_VTBL_AUTORESIZE_FLG
padding[23]	__u8	0	

attribute name	type	value	comment
crc	__be32	crc32_le	

ubi layout volume 的内容填充及烧写方法请参考 u-boot-2018/cmd/ubi_simu.c 的 ubi_simu_create_vol 和 wr_vol_table 函数

注意 ubi 中 crc32_le 算法与 sunxi_mbr 的 crc32 算法不一样。

ubi 中 crc32_le 参考 crc32_le.c 用法

sunxi_mbr 中 crc32 参考 crc32.c 用法



10 烧写逻辑卷

PEB = ubi_ec_hdr + ubi_vid_hdr + LEB

其中 ubi_ec_hdr 和 ubi_vid_hdr 存放于 PEB 的首逻辑页（logical page0）。

ubi_ec_hdr 存放于 0 字节偏移处，大小与物理页 size 对齐

ubi_vid_hdr 存放于 1 个物理页 size 偏移处，大小也与物理页 size 对齐

10.1 ubi_ec_hdr

ubi_ec_hdr：主要用于存储 PEB 的擦除次数信息，需动态生成 crc32_le 校验值。

struct ubi_ec_hdr 位于 u-boot-2018/drivers/mtd/ubi/ubi-media.h，结构体各个成员使用大端表示。

```
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;
```

attribute name	type	value	comment
magic	__be32	0x55424923	UBI#
version	__u8	1	
padding1[3]	__u8	0	
ec	__be64	1	
vid_hdr_offset	__be32	physical page size	2048
data_offset	__be32	logical page size	4096
image_seq	__be32	0	
padding2[32]	__u8	0	
hdr_crc	__be32		crc32_le

ubi_ec_hdr 的填充方法请参考 u-boot-2018/cmd/ubi_simu.c 的 fill_ec_hdr 函数。

10.2 ubi_vid_hdr

ubi_vid_hdr: 存放 PEB 和 LEB&Volume 映射信息，需动态生成 crc32_le 校验值

struct ubi_vid_hdr 位于 u-boot-2018/drivers/mtd/ubi/ubi-media.h，结构体各个成员使用大端表示。

```
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;

```

attribute name	type	value	comment
magic	__be32	0x55424921	UBI!
version	__u8	1	
vol_type	__u8	1	
copy_flag	__u8	0	
compat	__u8		默认为 0, layout volume 固定为 5
vol_id	__be32		volume id, 从 0 开始编号, layout vol 固定为 0x7ffefff
lnum	__be32		volume 内 LEB NO., 从 0 开始编号 [®]
padding1[4]	__u8	0	
data_size	__be32	0	
used_ebs	__be32	0	
data_pad	__be32	0	
data_crc	__be32	0	
padding2[4]	__u8	0	
sqnum	__be64		LEB 全局 sequence NO., 记录 LEB 的写顺序, 从 0 开始递增
padding3[12]	__u8	0	
hdr_crc	__be32		crc_le

ubi_vid_hdr 的填充方法请参考 u-boot-2018/cmd/ubi_simu.c 的 fill_vid_hdr 函数。

11 数据对齐

有数据对齐需求时，不能填充 0xff 数据，可选择填充全 0






著作权声明

版权所有 ©2024 珠海全志科技股份有限公司。保留一切权利。

本档及内容受著作权法保护，其著作权由珠海全志科技股份有限公司（“全志”）拥有并保留一切权利。

本档是全志的原创作品和版权财产，未经全志书面许可，任何单位和个人不得擅自摘抄、复制、修改、发表或传播本档内容的部分或全部，且不得以任何形式传播。

商标声明

、 **全志科技** （不完全列举）均为珠海全志科技股份有限公司的商标或者注册商标。在本档描述的产品中出现的其它商标，产品名称，和服务名称，均由其各自所有人拥有。

免责声明

您购买的产品、服务或特性应受您与珠海全志科技股份有限公司（“全志”）之间签署的商业合同和条款的约束。本档中描述的全部或部分产品、服务或特性可能不在您所购买或使用的范围内。使用前请认真阅读合同条款和相关说明，并严格遵循本档的使用说明。您将自行承担任何不当使用行为（包括但不限于如超压，超频，超温使用）造成的不利后果，全志概不负责。

本档作为使用指导仅供参考。由于产品版本升级或其他原因，本档内容有可能修改，如有变更，恕不另行通知。全志尽全力在本档中提供准确的信息，但并不确保内容完全没有错误，因使用本档而发生损害（包括但不限于间接的、偶然的、特殊的损失）或发生侵犯第三方权利事件，全志概不负责。本档中的所有陈述、信息和建议并不构成任何明示或暗示的保证或承诺。

本档未以明示或暗示或其他方式授予全志的任何专利或知识产权。在您实施方案或使用产品的过程中，可能需要获得第三方的权利许可。请您自行向第三方权利人获取相关的许可。全志不承担也不代为支付任何关于获取第三方许可的许可费或版税（专利税）。全志不对您所使用的第三方许可技术做出任何保证、赔偿或承担其他义务。