

思澈方案外发 NAND 烧录

1.nand flash BBM(坏块管理策略)

(1)坏块跳过策略:遇到坏块跳过, 存放进好块里。

(2)坏块替换策略:替换之后, FTL 会将坏块地址重新映射到好块地址,

基于 NAND Flash 来讲, 用 SA 区中的好块替换坏块。SpareArea(SA 区)一般用来标记坏块, 和保存对 main 区数据的 ECC 校验码。是基于 NAND Flash 的概念。

思澈方案保留给 BBM(坏块管理)的大小为 flash 总大小的 1/32,

例:

512Mb flash 烧录包大小为 490Mb, 最后 16Mb 是没有固件的。需要烧录厂按照思澈的坏块管理策略写入相应坏块地址跟映射地址。

512Mb : 0x400 0000

固件烧录地址: 从 0x0 开始到 0x3E0 0000

BBM(坏块管理地址): 0x3E0 0000 到 0x400 0000 结束

```
#define BBM_MAGIC_NUM (0x5366424d)

typedef struct
{
    uint16_t logic_blk; // logic block number
    uint16_t physical_blk; // physical block number
} Sifli_MapTbl;

typedef struct
{
    uint32_t magic; // magic number to decalare bbm talbe, 0x5366424d
    uint32_t version: 31;
    uint32_t tbl_idx: 1; // 0:TBL1, 1:TBL2
    uint16_t bbk_num; // bad block number
    uint16_t free_blk_num; // free block number
    uint16_t free_blk_start; // free block start addr, it will be used for next map.
    uint16_t reserv_blk_start; // start block number for reverved spare
    uint32_t hdr_crc; // crc for this table before this member
    uint32_t tbl_crc; // map table crc
    Sifli_MapTbl stru_tbl[64 - 4]; // max support 64 block backup
} Sifli_NandBBM; //Nand Flash Bad block management
```

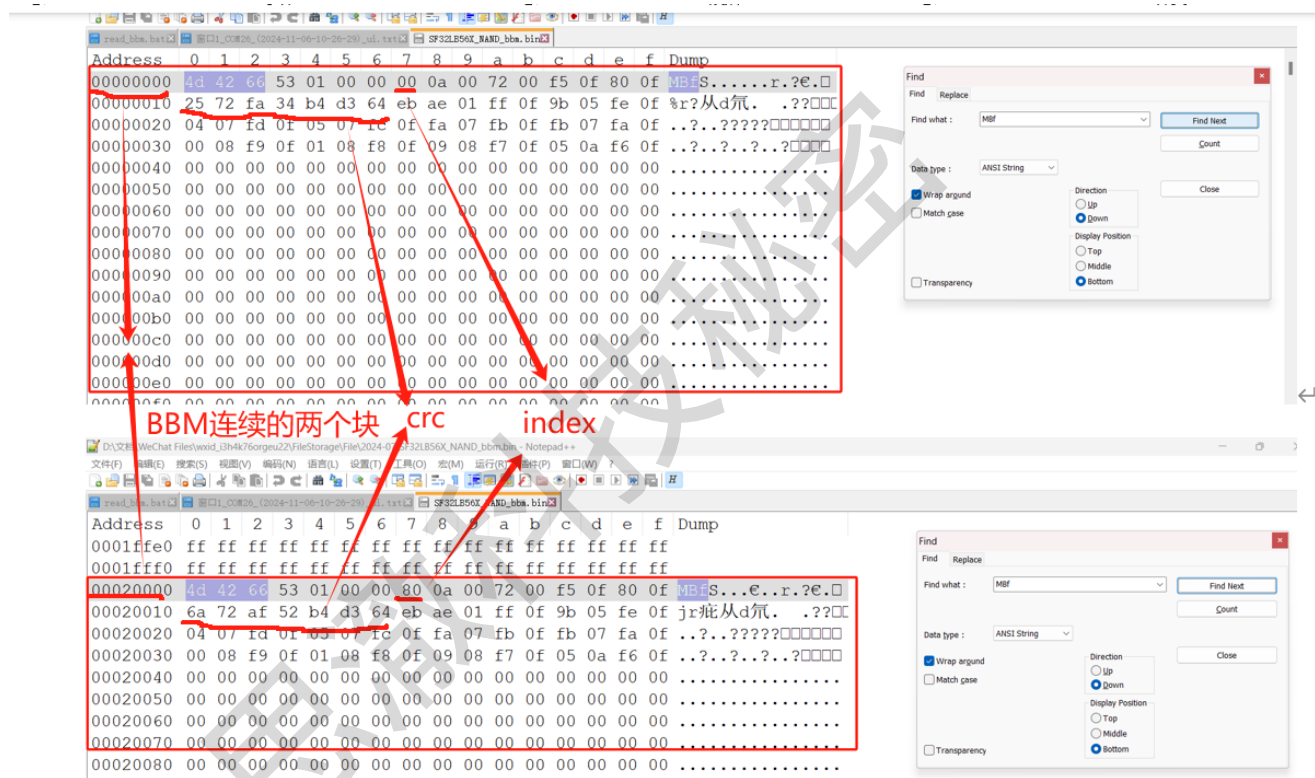
(映射表的数据结构)

2.BBM(坏块管理)规则

2.1.从 0x3e0 0000 开始连续两个块，除了 index 和 CRC 值其他的内容都应该相同

保留区从前往后数，在遇到的第一个 good block 创建映射表格，第二个 good block 创建备份映射表格

如图数据是从 BBM(坏块管理区域)读取的数据



2.2.BBM(坏块管理)数据

1. magic 固定不变，前后两个块都是一样的。

Magic: 坏块管理表的 magic number, 固定为 0x5366424D

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
00000000	4d	42	66	53	01	00	00	00	0a	00	72	00	f5	0f	80	0f	MBfS.....r.?€.
00000010	25	72	fa	34	b4	d3	64	eb	ae	01	ff	0f	9b	05	fe	0f	%r?从d氘. .??□□
00000020	04	07	fd	0f	05	07	fc	0f	fa	07	fb	0f	fb	07	fa	0f	..?..?????□□□□□
00000030	00	08	f9	0f	01	08	f8	0f	09	08	f7	0f	05	0a	f6	0f	..?..?..?..?□□□□
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	
00000080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000a0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

magic固定不变

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
0001fff0	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	
00020000	4d	42	66	53	01	00	00	80	0a	00	72	00	f5	0f	80	0f	MBfS...€..r.?€.
00020010	6a	72	af	52	b4	d3	64	eb	ae	01	ff	0f	9b	05	fe	0f	jr疵从d氘. .??□□
00020020	04	07	fd	0f	05	07	fc	0f	fa	07	fb	0f	fb	07	fa	0f	..?..?????□□□□□
00020030	00	08	f9	0f	01	08	f8	0f	09	08	f7	0f	05	0a	f6	0f	..?..?..?..?□□□□
00020040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00020050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00020060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00020070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00020080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00020090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000200a0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000200b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000200c0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000200d0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000200e0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

2.version 版本号从 1 开始，前后两个块都是一样的，烧录厂烧录默认写

1 就行

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
00000000	4d	42	66	53	01	00	00	00	0a	00	72	00	f5	0f	80	0f	MBfS...€..r.?€.
00000010	25	72	fa	34	b4	d3	64	eb	ae	01	ff	0f	9b	05	fe	0f	%r?从d氘. .??□□
00000020	04	07	fd	0f	05	07	fc	0f	fa	07	fb	0f	fb	07	fa	0f	..?..?????□□□□□
00000030	00	08	f9	0f	01	08	f8	0f	09	08	f7	0f	05	0a	f6	0f	..?..?..?..?□□□□
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	
00000080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000a0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000c0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000d0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000e0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

版本号

Find

Find Replace

Find what : MBf

Data type : ANSI String

☒ Wrap around

☐ Match case

☐ Transparency

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
0001fff0	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	
00020000	4d	42	66	53	01	00	00	80	0a	00	72	00	f5	0f	80	0f	MBfs...€...r.?€.
00020010	6a	72	af	52	b4	d3	64	eb	ae	01	ff	0f	9b	05	fe	0f	jr疵从d氖. .??□□
00020020	04	07	fd	0f	05	07	fc	0f	fa	07	fb	0f	fb	07	fa	0f	..?..?..?..?□□□□□
00020030	00	08	f9	0f	01	08	f8	0f	09	08	f7	0f	05	0a	f6	0f	..?..?..?..?□□□□
00020040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00020050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00020060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00020070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00020080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00020090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000200a0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000200b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000200c0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000200d0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000200e0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

3. index 这里有区别，前面的块区里固定是 0x0.后面一个块区固定为 0x80

read_bbm.bat

窗口1_COW26_(2024-11-06-10-26-29)_ui.txt

SF32LB56X_NAND_bbm.bin

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
00000000	4d	42	66	53	01	00	00	00	0a	00	72	00	f5	0f	80	0f	MBfs...€...r.?€.
00000010	25	72	fa	34	b4	d3	64	eb	ae	01	ff	0f	9b	05	fe	0f	%r?从d氖. .??□□
00000020	04	07	fd	0f	05	07	fc	0f	fa	07	fb	0f	fb	07	fa	0f	..?..?..?..?□□□□□
00000030	00	08	f9	0f	01	08	f8	0f	09	08	f7	0f	05	0a	f6	0f	..?..?..?..?□□□□
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	
00000080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000a0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000c0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000d0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000e0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000f0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00000100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Find

Find Replace

Find what: MBf

Data type: ANSI String

☒ Wrap around

Direction

☐ Up
 ☒ Down

☐ Match case

Display Position

☐ Top
 ☐ Middle
 ☒ Bottom

☐ Transparency

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
0001fff0	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	
00020000	4d	42	66	53	01	00	00	80	0a	00	72	00	f5	0f	80	0f	MBfs...r.?e.
00020010	6a	72	af	52	b4	d3	64	eb	ae	01	ff	0f	9b	05	fe	0f	jr疵从d氖. .??
00020020	04	07	fd	0f	05	07	fc	0f	fa	07	fb	0f	fb	07	fa	0f	..?..?..?..?
00020030	00	08	f9	0f	01	08	f8	0f	09	08	f7	0f	05	0a	f6	0f	..?..?..?..?
00020040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00020050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00020060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00020070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00020080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00020090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000200a0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000200b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000200c0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000200d0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000200e0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000200f0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00020100	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00020110	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Find

Find Replace

Find what : MBF

Data type : ANSI String

☒ Wrap around

☐ Match case

☐ Transparency

4. bbk_num (坏块数量)，烧录时记录的坏块数量。前后两个块都是一样的

Bbk_num: 当前统计到的坏块数量，当未发现坏块时为 0.

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
00000000	4d	42	66	53	01	00	00	0a	00	72	00	f5	0f	80	0f		MBfs.....r.?e.
00000010	25	72	fa	34	b4	d3	64	eb	ae	01	ff	0f	9b	05	fe	0f	%r?从d氖. .??
00000020	04	07	fd	0f	05	07	fc	0f	fa	07	fb	0f	fb	07	fa	0f	..?..?..?..?
00000030	00	08	f9	0f	01	08	f8	0f	09	08	f7	0f	05	0a	f6	0f	..?..?..?..?
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	

nand_bad统计为10

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
00020000	4d	42	66	53	01	00	00	80	0a	00	72	00	f5	0f	80	0f	MBfs...r.?e.
00020010	6a	72	af	52	b4	d3	64	eb	ae	01	ff	0f	9b	05	fe	0f	jr疵从d氖. .??
00020020	04	07	fd	0f	05	07	fc	0f	fa	07	fb	0f	fb	07	fa	0f	..?..?..?..?
00020030	00	08	f9	0f	01	08	f8	0f	09	08	f7	0f	05	0a	f6	0f	..?..?..?..?
00020040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00020050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00020060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00020070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

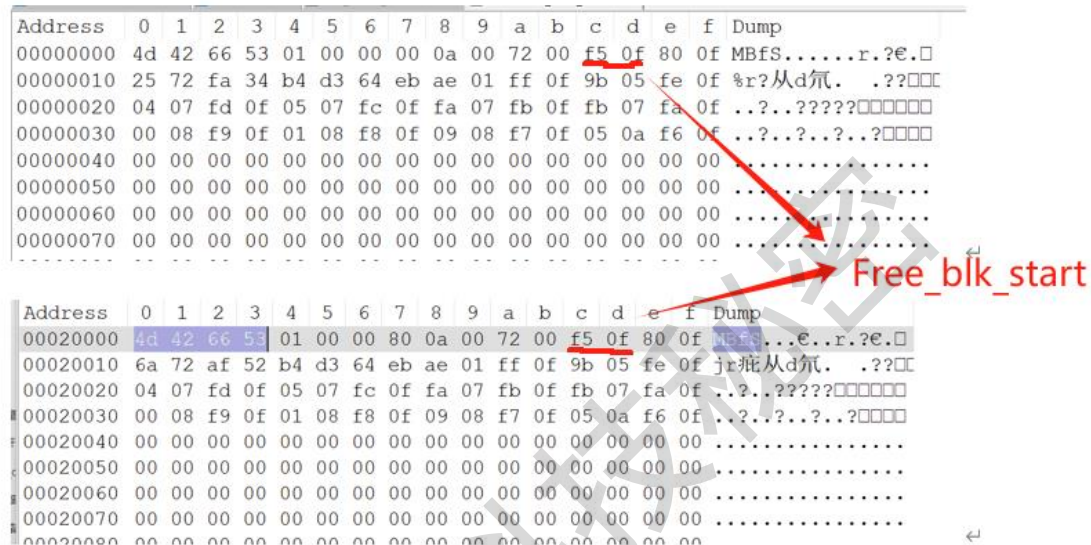
10个bab地址



6. Free_blk_start: 可用来做替换的表的块号，这是个递减的数据，替换块从后往前找（如果未发生过替换，则应该为总块数减 1，比如 1024 个块的颗粒，初始值应该为 1023）

如下图： $0x0ff5(\text{Free_blk_start}) + 1 + 0x0a(\text{bbk_num}) = 4096$ 个块

512M flash 有 4096 个块，128 个块是 1/32.



Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
00000000	4d	42	66	53	01	00	00	00	0a	00	72	00	f5	0f	80	0f	MBfS.....r.?e.□
00000010	25	72	fa	34	b4	d3	64	eb	ae	01	ff	0f	9b	05	fe	0f	%r?从d氖. .??□□
00000020	04	07	fd	0f	05	07	fc	0f	fa	07	fb	0f	fb	07	fa	0f	...???.??□□□□□
00000030	00	08	f9	0f	01	08	f8	0f	09	08	f7	0f	05	0a	f6	0f	...?..?..?□□□□
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	

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
00020000	4d	42	66	53	01	00	00	80	0a	00	72	00	f5	0f	80	0f	MBfS...e..r.?e.□
00020010	6a	72	af	52	b4	d3	64	eb	ae	01	ff	0f	9b	05	fe	0f	%r?从d氖. .??□
00020020	04	07	fd	0f	05	07	fc	0f	fa	07	fb	0f	fb	07	fa	0f	...???.??□□□□□
00020030	00	08	f9	0f	01	08	f8	0f	09	08	f7	0f	05	0a	f6	0f	...?..?..?□□□□
00020040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00020050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00020060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00020070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

7. Reserve_blk_start: 保留块的第一个块号，比如 1024 个块，最后 32 个块保留的话，则此数据为 992，这个数据不会更新。

如下图： $4096 - 128 = 0x0f80(\text{Reserve_blk_start})$

512M flash 有 4096 个块，128 个块是 1/32.

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
00000000	4d	42	66	53	01	00	00	00	0a	00	72	00	f5	0f	80	0f	MBfS.....r.?€.
00000010	25	72	fa	34	b4	d3	64	eb	ae	01	ff	0f	9b	05	fe	0f	%r?从d氘. .??□□
00000020	04	07	fd	0f	05	07	fc	0f	fa	07	fb	0f	fb	07	fa	0f	..?..????□□□□□□
00000030	00	08	f9	0f	01	08	f8	0f	09	08	f7	0f	05	0a	f6	0f	..?..?..?□□□□
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	

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
00020000	4d	42	66	53	01	00	00	80	0a	00	72	00	f5	0f	80	0f	MBfS...e..r.?€.
00020010	6a	72	af	52	b4	d3	64	eb	ae	01	ff	0f	9b	05	fe	0f	jr疵从d氘. .??□□
00020020	04	07	fd	0f	05	07	fc	0f	fa	07	fb	0f	fb	07	fa	0f	..?..????□□□□□□
00020030	00	08	f9	0f	01	08	f8	0f	09	08	f7	0f	05	0a	f6	0f	..?..?..?□□□□
00020040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00020050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00020060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00020070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Reserve_
blk_start

8. Hdr_crc: 保存此字段之前的数据的 CRC 校验值（从 magic 开始的 16 个字节）

算法 api: Static uint32_t bbm_crc_check(const uint8_t *buf, uint32_t size)

代码公式: `bbm_local[0].hdr_crc = bbm_crc_check((const uint8_t *)&bbm_local[0], 16);`

实际值: `const uint8_t *buf = 0x4d426653010000000a007100f50f800f`

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
00000000	4d	42	66	53	01	00	00	0a	00	72	00	f5	0f	80	0f		*buf

uint32_t size = 16

9. Tbl_crc: 后面 stru_tbl 字段的 CRC 校验值（长度为 4 * （保留块数-4））

注意 stru_tbl 有新老驱动区别，老的是 64-4.新的是 128-4

算法 api: Static uint32_t bbm_crc_check(const uint8_t *buf, uint32_t size)

代码公式: `bbm_local[0].tbl_crc = bbm_crc_check((const uint8_t *)&(bbm_local[0].stru_tbl), sizeof(Sifli_MapTbl) * (bkup_blk - 4));`

实际值: `const uint8_t *buf = bbm_local[0].stru_tbl[cnt].logic_blk = bblk; //目前字段的逻辑块编号，对烧录来说就是最后 32/1 开始的这块号。4096-128 = 3968`

`bbm_local[0].stru_tbl[cnt].physical_blk =`

`bbm_local[0].free_blk_start + 1 ; //可用来做替换的表的块号+1`

bkup_blk = //保留的用来做映射的块，老代码最大支持 2Gb 所以是 $2048/32=64$ ，新代码最大支持 4Gb 所有是 $4096/32=128$

uint32_t size = $4 * (128 - 4)$ //新驱动代码最大支持 4G

uint32_t size = $4 * (64 - 4)$ //旧驱动代码最大支持 2G

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
00000000	4d	42	66	53	01	00	00	00	0b	00	71	00	f4	0f	80	0f	MBfS.....q.??e.
00000010	70	67	78	c6	7f	47	66	ef	32	02	ff	0f	33	02	fe	0f	pgx?Gf?. .3.??
00000020	37	02	fd	0f	65	02	fc	0f	bd	02	fb	0f	c5	02	fa	0f	7.?e.?????□□□□□
00000030	cd	02	f9	0f	9d	06	f8	0f	7d	07	f7	0f	b9	07	f6	0f	??..?}.???□□□□□
00000040	77	0f	f5	0f	00	00	00	00	00	00	00	00	00	00	00	00	w.?.....□
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	
00000080	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	从绿色款之后取4* (实际flash的32分
00000090	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	一减4)...
000000a0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	4Gb就是4* ((4096/32) -4)
000000b0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
000000c0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	取得496个字节的数据在代入crc公式
000000d0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	进行计算，最后填入绿框 (Tbl_crc)
000000e0	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

```
#define BBM_MAGIC_NUM (0x5366424d)

typedef struct
{
    uint16_t logic_blk; // logic block number
    uint16_t physical_blk; // physical block number
} Sifli_MapTbl;

typedef struct
{
    uint32_t magic; // magic number to decalare bbm talbe, 0x5366424d
    uint32_t version: 31;
    uint32_t tbl_idx: 1; // 0:TBL1, 1:TBL2
    uint16_t bbk_num; // bad block number
    uint16_t free_blk_num; // free block number
    uint16_t free_blk_start; // free block start addr, it will be used for next map.
    uint16_t reserv_blk_start; // start block number for reverved spare
    uint32_t hdr_crc; // crc for this table before this member
    uint32_t tbl_crc; // map table crc
    Sifli_MapTbl stru_tbl[64 - 4]; // max support 64 block backup
} Sifli_NandBBM; //Nand Flash Bad block management
```

```

1 编辑(E) 选择(S) 查看(V) 转到(G) 运行(R) 终端(T) 帮助(H)
2
3 文件 编辑 视图 窗口 帮助
4
5 文件列表
6 -h
7 刷新
8 文件管理 (Ctrl+Shift+G) - 59 个挂起的更改
9
10 文件中有 27 个结果 - 已禁止排除设置和忽略文件 (启用) - 在编辑器中打开
11
12 C:\sifli_bbm.c sf32b5xx\drivers\hal
13 static Sifli_NandBBM bbm_local[2];
14 static int bbm_map_check(Sifli_NandBBM *bbm, table)
15 void *bbm_local[0], sizeof(Sifli_NandBBM);
16 Sifli_NandBBM *bbmt = (Sifli_NandBBM *)bbm_page_cache;
17 Sifli_NandBBM *bbmt = (Sifli_NandBBM *)bbm_page_cache;
18 *bbm_local[tid], sizeof(Sifli_NandBBM);
19 if (bbm_map_check(Sifli_NandBBM *bbm, page, cache) != 0)
20 Sifli_NandBBM *bbmt = (Sifli_NandBBM *)bbm_page_cache;
21 Sifli_NandBBM *bbmt = (Sifli_NandBBM *)bbm_page_cache;
22 Sifli_NandBBM *tab;
23 tab = (Sifli_NandBBM *)bbm_page_cache;
24 tab = (Sifli_NandBBM *)bbm_page_cache;
25 tab = (Sifli_NandBBM *)bbm_page_cache;
26 uint8_t *tab, 16; //sizeof(Sifli_NandBBM) - sizeof(stru_tbi) - 4*2
27 bbm_local[tid], tab, sizeof(Sifli_NandBBM);
28 void *bbm_local[0], sizeof(Sifli_NandBBM);
29 Sifli_NandBBM *bbmt = (Sifli_NandBBM *)bbm_page_cache;
30 Sifli_NandBBM *bbmt = (Sifli_NandBBM *)bbm_page_cache;
31 *bbm_local[0], 0, sizeof(Sifli_NandBBM) * 2;
32 void *bbm_local[0], sizeof(Sifli_NandBBM);
33 void *bbm_local[1], sizeof(Sifli_NandBBM);
34 Sifli_NandBBM *tbl = &bbm_local[0];
35 C:\sifli_bbm.h sf32b5xx\drivers\include
36 Sifli_NandBBM; //Nand Flash Bad block management
37
38 C:\drv_spi_nand.c sf32b5xx\drivers\thread\bsp\sifli\drivers
39 Sifli_NandBBM *ctx = (Sifli_NandBBM *)bbm_get_context(0);
40 Sifli_NandBBM *ctx = (Sifli_NandBBM *)bbm_get_context(0);
41 Sifli_NandBBM *ctx = (Sifli_NandBBM *)bbm_get_context(0);
42 Sifli_NandBBM *ctx = (Sifli_NandBBM *)bbm_get_context(0);
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
780
781
782
783
784
785
786
787
788
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
810
811
812
813
814
815
816
817
818
819
820
821
822
823
824
825
826
827
828
829
830
831
832
833
834
835
836
837
838
839
840
841
842
843
844
845
846
847
848
849
850
851
852
853
854
855
856
857
858
859
860
861
862
863
864
865
866
867
868
869
870
871
872
873
874
875
876
877
878
879
880
881
882
883
884
885
886
887
888
889
890
891
892
893
894
895
896
897
898
899
900
901
902
903
904
905
906
907
908
909
910
911
912
913
914
915
916
917
918
919
920
921
922
923
924
925
926
927
928
929
930
931
932
933
934
935
936
937
938
939
940
941
942
943
944
945
946
947
948
949
950
951
952
953
954
955
956
957
958
959
960
961
962
963
964
965
966
967
968
969
970
971
972
973
974
975
976
977
978
979
980
981
982
983
984
985
986
987
988
989
990
991
992
993
994
995
996
997
998
999
1000

```

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
00000000	4d	42	66	53	01	00	00	00	0a	00	72	00	f5	0f	80	0f	MBfs.....r.?e.
00000010	25	72	fa	34	b4	d3	64	eb	ae	01	ff	0f	9b	05	fe	0f	%r?从d气. .??
00000020	04	07	fd	0f	05	07	fc	0f	fa	07	fb	0f	fb	07	fa	0f	..?..?????□□□□□□
00000030	00	08	f9	0f	01	08	f8	0f	09	08	f7	0f	05	0a	f6	0f	..?..?..?..?□□□□
00000040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

Address	0	1	2	3	4	5	6	7	8	9	a	b	c	d	e	f	Dump
0001ffe0	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	
0001fff0	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	ff	
00020000	4d	42	66	53	01	00	00	80	0a	00	72	00	f5	0f	80	0f	MBfs...e..r.?e.
00020010	6a	72	af	52	b4	d3	64	eb	ae	01	ff	0f	9b	05	fe	0f	jr症从d气. .??
00020020	04	07	fd	0f	05	07	fc	0f	fa	07	fb	0f	fb	07	fa	0f	..?..?????□□□□□□
00020030	00	08	f9	0f	01	08	f8	0f	09	08	f7	0f	05	0a	f6	0f	..?..?..?..?□□□□
00020040	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00020050	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00020060	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	
00020070	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	00	

10.附 CRC 算法

```

Static uint32_t bbm_crc_check(const uint8_t *buf, uint32_t size)
{
    unsigned int i, crc;
    crc = 0xFFFFFFFF;

    for (i = 0; i < size; i++)
        crc = crc32tab[(crc ^ buf[i]) & 0xff] ^ (crc >> 8);
}

```

```
    return crc ^ 0xFFFFFFFF;
}
```

```
static const unsigned int crc32tab[] =
```

```
{
    0x00000000L, 0x77073096L, 0xee0e612cL, 0x990951baL,
    0x076dc419L, 0x706af48fL, 0xe963a535L, 0x9e6495a3L,
    0x0edb8832L, 0x79dcb8a4L, 0xe0d5e91eL, 0x97d2d988L,
    0x09b64c2bL, 0x7eb17cbdL, 0xe7b82d07L, 0x90bf1d91L,
    0x1db71064L, 0x6ab020f2L, 0xf3b97148L, 0x84be41deL,
    0x1adad47dL, 0x6ddde4ebL, 0xf4d4b551L, 0x83d385c7L,
    0x136c9856L, 0x646ba8c0L, 0xfd62f97aL, 0x8a65c9ecL,
    0x14015c4fL, 0x63066cd9L, 0xfa0f3d63L, 0xd080df5L,
    0x3b6e20c8L, 0x4c69105eL, 0xd56041e4L, 0xa2677172L,
    0x3c03e4d1L, 0x4b04d447L, 0xd20d85fdL, 0xa50ab56bL,
    0x35b5a8faL, 0x42b2986cL, 0xdbbbc9d6L, 0xacbcf940L,
    0x32d86ce3L, 0x45df5c75L, 0xdcd60dcfL, 0xabd13d59L,
    0x26d930acL, 0x51de003aL, 0xc8d75180L, 0xbfd06116L,
    0x21b4f4b5L, 0x56b3c423L, 0xcfba9599L, 0xb8bda50fL,
    0x2802b89eL, 0x5f058808L, 0xc60cd9b2L, 0xb10be924L,
    0x2f6f7c87L, 0x58684c11L, 0xc1611dabL, 0xb6662d3dL,
    0x76dc4190L, 0x01db7106L, 0x98d220bcL, 0xefd5102aL,
    0x71b18589L, 0x06b6b51fL, 0x9fbfe4a5L, 0xe8b8d433L,
    0x7807c9a2L, 0x0f00f934L, 0x9609a88eL, 0xe10e9818L,
    0x7f6a0dbbL, 0x086d3d2dL, 0x91646c97L, 0xe6635c01L,
    0x6b6b51f4L, 0x1c6c6162L, 0x856530d8L, 0xf262004eL,
    0x6c0695edL, 0x1b01a57bL, 0x8208f4c1L, 0xf50fc457L,
    0x65b0d9c6L, 0x12b7e950L, 0x8bbbeb8eL, 0xfcb9887cL,
    0x62dd1ddfl, 0x15da2d49L, 0x8cd37cf3L, 0xfbd44c65L,
    0x4db26158L, 0x3ab551ceL, 0xa3bc0074L, 0xd4bb30e2L,
    0x4adfa541L, 0x3dd895d7L, 0xa4d1c46dL, 0xd3d6f4fbL,
    0x4369e96aL, 0x346ed9fcL, 0xad678846L, 0xda60b8d0L,
    0x44042d73L, 0x33031de5L, 0xaa0a4c5fL, 0xdd0d7cc9L,
    0x5005713cL, 0x270241aaL, 0xbe0b1010L, 0xc90c2086L,
    0x5768b525L, 0x206f85b3L, 0xb966d409L, 0xce61e49fL,
    0x5edef90eL, 0x29d9c998L, 0xb0d09822L, 0xc7d7a8b4L,
    0x59b33d17L, 0x2eb40d81L, 0xb7b5dc3bL, 0xc0ba6cadL,
    0xedb88320L, 0x9abfb3b6L, 0x03b6e20cL, 0x74b1d29aL,
    0xeadd54739L, 0x9dd277afL, 0x04db2615L, 0x73dc1683L,
    0xe3630b12L, 0x94643b84L, 0x0d6d6a3eL, 0x7a6a5aa8L,
    0xe40ecf0bL, 0x9309ff9dL, 0x0a00ae27L, 0x7d079eb1L,
    0xf00f9344L, 0x8708a3d2L, 0x1e01f268L, 0x6906c2feL,
    0xf762575dL, 0x806567cbL, 0x196c3671L, 0x6e6b06e7L,
    0xfed41b76L, 0x89d32be0L, 0x10da7a5aL, 0x67dd4accL,
    0xf9b9df6fL, 0x8ebeeff9L, 0x17b7be43L, 0x60b08ed5L,
    0xd6d6a3e8L, 0xa1d1937eL, 0x38d8c2c4L, 0x4fdff52L,
    0xd1bb67f1L, 0xa6bc5767L, 0x3fb506ddL, 0x48b2364bL,
    0xd80d2bdaL, 0xaf0a1b4cL, 0x36034af6L, 0xa1047a60L,
    0xdf60efc3L, 0xa867df55L, 0x316e8eefL, 0xa4699be79L,
    0xcb61b38cL, 0xbcb6831aL, 0x256fd2a0L, 0x5268e236L,
    0xccc0c7795L, 0xbb0b4703L, 0x220216b9L, 0x5505262fL,
    0xc5ba3bbeL, 0xb2bd0b28L, 0x2bb45a92L, 0x5cb36a04L,
    0xc2d7ffa7L, 0xb5d0cf31L, 0x2cd99e8bL, 0x5bdeae1dL,
    0x9b64c2b0L, 0xec63f226L, 0x756aa39cL, 0x026d930aL,
    0x9c0906a9L, 0xeb0e363fL, 0x72076785L, 0x05005713L,
    0x95bf4a82L, 0xe2b87a14L, 0x7bb12baeL, 0x0cb61b38L,
    0x92d28e9bL, 0xe5d5be0dL, 0x7cdcefb7L, 0x0bdbdf21L,
    0x86d3d2d4L, 0xf1d4e242L, 0x68ddb3f8L, 0x1fda836eL,
    0x81be16cdL, 0xf6b9265bL, 0x6fb077e1L, 0x18b74777L,
    0x88085ae6L, 0xff0f6a70L, 0x66063bcaL, 0x11010b5cL,
    0x8f659effL, 0xf862ae69L, 0x616bffd3L, 0x166ccf45L,
    0xa00ae278L, 0xd70dd2eeL, 0x4e048354L, 0x3903b3c2L,
    0xa7672661L, 0xd06016f7L, 0x4969474dL, 0x3e6e77dbL,
    0xaed16a4aL, 0xd9d65adcL, 0x40df0b66L, 0x37d83bf0L,
    0xa9bcae53L, 0xdebb9ec5L, 0x47b2cf7fL, 0x30b5ffe9L,
    0xbdbdf21cL, 0xcabac28aL, 0x53b39330L, 0x24b4a3a6L,
    0xbad03605L, 0xcdd70693L, 0x54de5729L, 0x23d967bfL,
    0xb3667a2eL, 0xc4614ab8L, 0x5d681b02L, 0x2a6f2b94L,
```

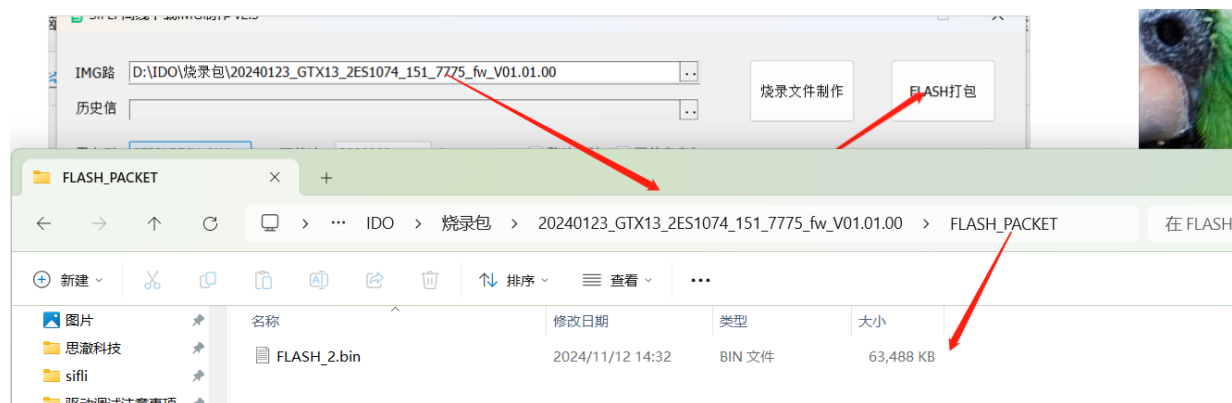
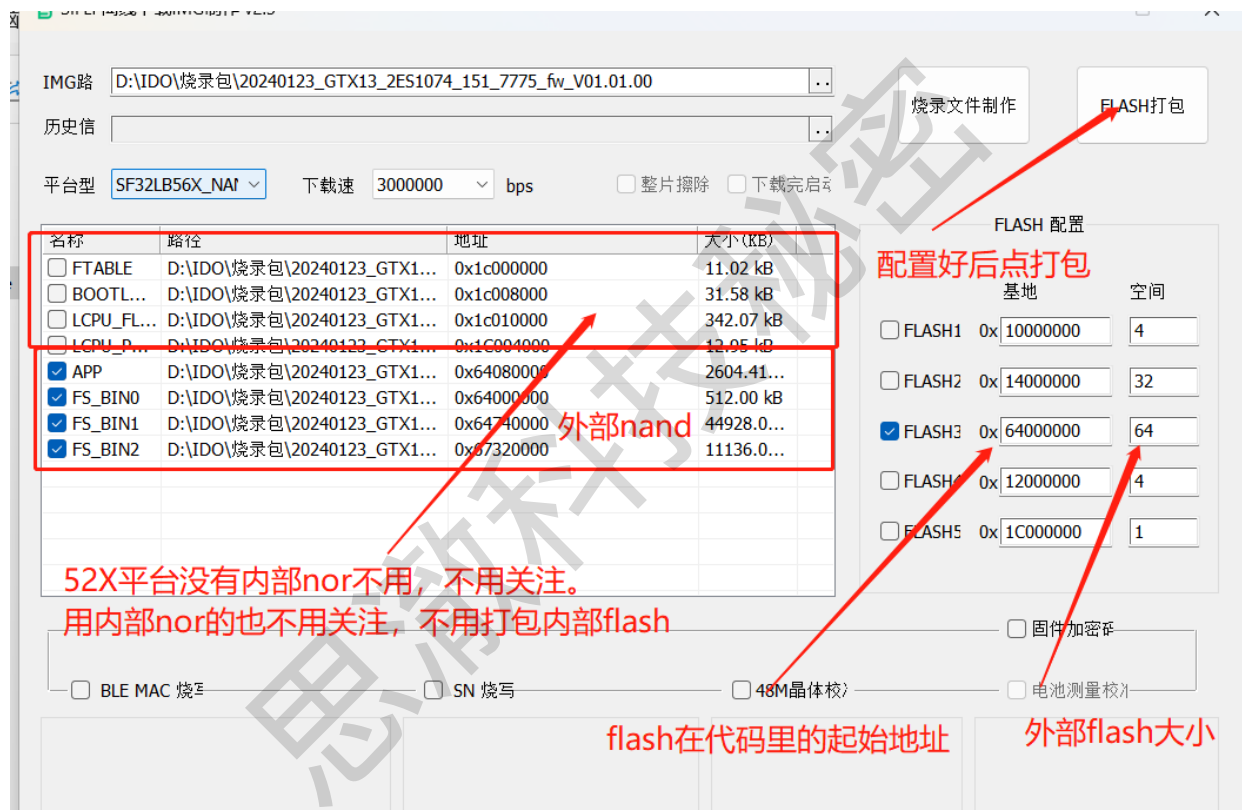
0xb40bbe37L, 0xc30c8ea1L, 0x5a05df1bL, 0x2d02ef8dL
};

3.外部 flash 打包

打包外部 flash 只用关注:

1. 选对 bin 文件
2. flash 起始地址配对
3. flash 大小填写实际大小。注意 nand 跟 nor 都是填写实际大小。nand 打包后 bin 大小比填写的值小 1/32 是对的。nand 有坏块管理

其他参数配置并不影响。

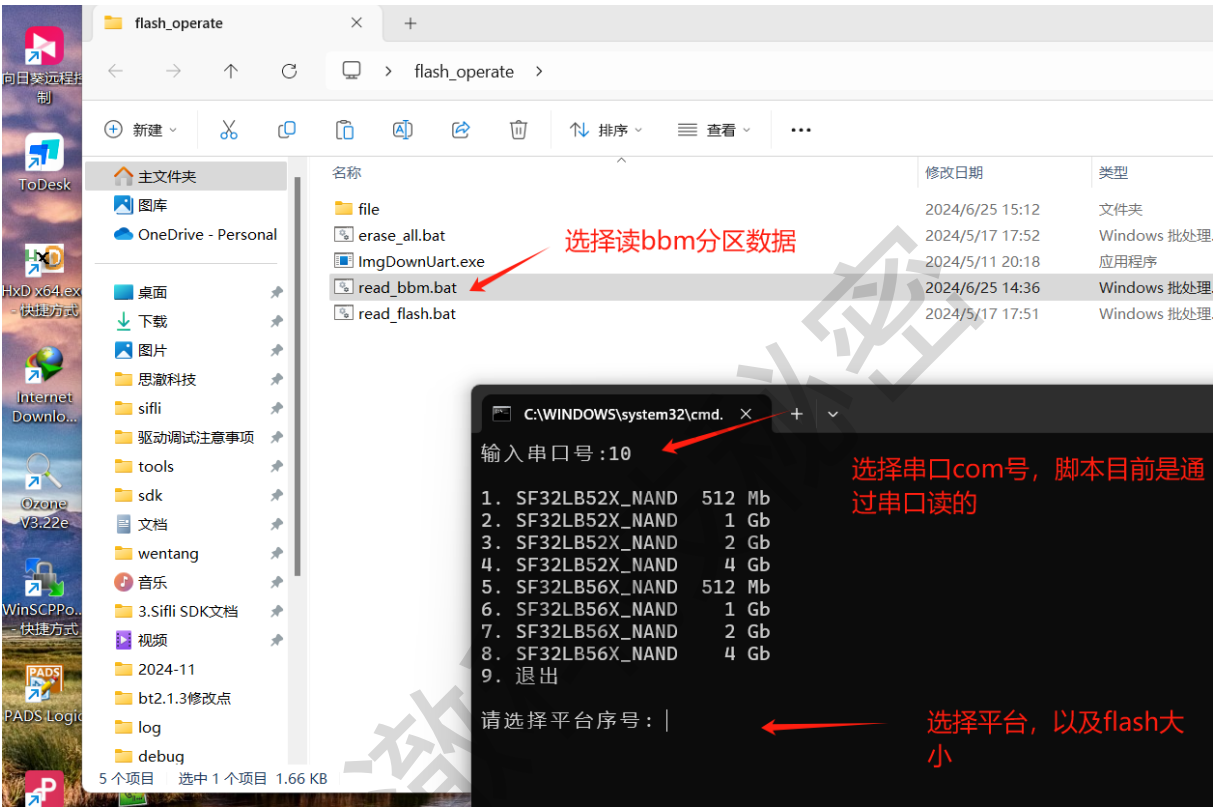


4.nand 读 bbm 分区验证，看是否正常

读出来的数据跟前面 2.2BBM(坏块管理)数据比较。看是否计算的跟烧录的一致。

也可以用要外发烧录的 nand，先用思澈工具烧录。再用脚本读出 bbm 分区数据，可以省掉除 #4 #5 #6 #7 #8 及坏块替换的计算。

同一块 flash 烧录时 flash: #1 #2 #3 是不变的



5. 附件

ImgStamp_v2.3.7zflash_operate.zip