Linux MTD下获取Nand flash各个参数的过程的详细解析

版本：v1.1

Crifan Li

摘要

本文主要介绍了Nand Flash的一些背景知识和术语，然后详尽分析了Linux的MTD中的nand_base.c中的nand_get_flash_type函数每一步骤的功能。

本文提供多种格式供：

在线阅读	HTML [1]	HTMLs [2]	PDF [3]	CHM [4]	TXT [5]	RTF [6]	WEBHELP [7]
下载（7zip压缩包）	HTML [8]	HTMLs [9]	PDF [10]	CHM [11]	TXT [12]	RTF [13]	WEBHELP [14]

HTML版本的在线地址为：

http://www.crifan.com/files/doc/docbook/nand_get_type/release/html/nand_get_type.html

有任何意见，建议，提交bug等，都欢迎去讨论组发帖讨论：

http://www.crifan.com/bbs/categories/nand_get_type/

修订历史		
修订 1.0	2009-07-28	crl
1. 详细解析了Linux MTD下获取Nand flash各个参数的具体过程		
修订 1.1	2012-08-09	crl
1. 通过Docbook发布		

[1] http://www.crifan.com/files/doc/docbook/nand_get_type/release/html/nand_get_type.html
[2] http://www.crifan.com/files/doc/docbook/nand_get_type/release/htmls/index.html
[3] http://www.crifan.com/files/doc/docbook/nand_get_type/release/pdf/nand_get_type.pdf
[4] http://www.crifan.com/files/doc/docbook/nand_get_type/release/chm/nand_get_type.chm
[5] http://www.crifan.com/files/doc/docbook/nand_get_type/release/txt/nand_get_type.txt
[6] http://www.crifan.com/files/doc/docbook/nand_get_type/release/rtf/nand_get_type.rtf
[7] http://www.crifan.com/files/doc/docbook/nand_get_type/release/webhelp/index.html
[8] http://www.crifan.com/files/doc/docbook/nand_get_type/release/html/nand_get_type.html.7z
[9] http://www.crifan.com/files/doc/docbook/nand_get_type/release/htmls/index.html.7z
[10] http://www.crifan.com/files/doc/docbook/nand_get_type/release/pdf/nand_get_type.pdf.7z
[11] http://www.crifan.com/files/doc/docbook/nand_get_type/release/chm/nand_get_type.chm.7z
[12] http://www.crifan.com/files/doc/docbook/nand_get_type/release/txt/nand_get_type.txt.7z
[13] http://www.crifan.com/files/doc/docbook/nand_get_type/release/rtf/nand_get_type.rtf.7z
[14] http://www.crifan.com/files/doc/docbook/nand_get_type/release/webhelp/nand_get_type.webhelp.7z

Linux MTD下获取Nand flash各个参数的过程的详细解析:

Crifan Li

版本：v1.1

出版日期 2012-08-09

[15] http://www.crifan.com/files/doc/docbook/soft_dev_basic/release/html/soft_dev_basic.html#cc_by_nc

目录

插图清单

第 1 章 看此文之前，一些有必要先解释的术语

下面是Linux MTD中，获取nand flash型号,各个参数，以及硬件特性的函数，其实也就是nand_get_flash_type，下面对其详细解析：

1.1. Program(编程)

此处的编程，不是写软件，写代码，而是对于硬件来说的，可以理解为对硬件编程，只不过其工具是硬件内部的逻辑，而不是你用的软件。对Nand Flash的编程，本质上就是实现写操作，将数据写到Nand Flash里面去，所以对于nand flash，可以简单的理解为 program编程 = write写（数据）。

1.2. Datasheet(数据手册)

这个词，本来没啥好说的，接触多了，自然就知道了。但是对于和我类似，最开始接触的时候，就是没搞懂这个词的具体含义。其中文翻译，一般称作，数据手册，意思就是，一个关于描述硬件各个硬件特性，参数以及/或者如何操作，如何使用的文档。

1.3. Erasesize / Writesize

这个是Linux MTD中，关于块大小和页大小的别名，第一次见到的时候，把我搞糊涂了，后来才慢慢明白的。因为，nand 操作的写基本单位页，所以，writesize，对应的就是pagesize，页大小。而擦除操作的基本单位是blocksize，块大小，所以也叫它erasesize。在此简单提一下这几个名词，方便和我遇到类似问题的朋友。

1.4. Spare Area / Redundant Area / OOB

nand flash中每一页对应一块区域，用于存放校验的ECC数据和其他一些信息，比如上层文件系统放的和自己文件系统相关的数据。这个区域，在Linux MTD相关系统中，被称作oob（out of band），可以翻译为带外，也就是nand flash的一个页，可以称作一个band，band之外，对应的就是指那个多出来的，特殊的区域了。而nand flash的datasheet中，一般成为spare area，可译为空闲区域，另外，在ID的含义解释中也叫做redundant area，可译为冗余区域，归根结底，都是一个含义。不要被搞糊涂了就好。

1.5. Page Register(页寄存器)

nand flash硬件中的一块地方，名字叫做register，实际就是一个数据缓存，一个buffer，用于存放那些从flash读出来或者将要写入到flash中的。其实叫做页缓存，更合适，更容易明白其含义。此页寄存器的大小 = 页大小+ oob 大小，即pagesize+oob，对于常见的页是2KB的，此页寄存器就是2KB+64=2112字节。

1.6. Chip和Plane

对于chip，其实任何某个型号的flash，都可以称其是一个chip，但是实际上，此处的chip，是针对内部来说的，也就是某型号的flash，内部有几个chip，比如下面会举例说到的，三星的2GB的K9WAG08U1A芯片（可以理解为外部芯片/型号）内部装了2个单片是1GB的K9K8G08U0A，此时就称 K9WAG08U1A内部有2个chip，而有些单个的chip，内部又包含多个plane，比如上面的K9K8G08U0A内部包含4个单片是2Gb的Plane。只有搞清楚了此处的chip和plane的关系，才能明白后面提到的多页（Multi Plane / Multi Page）编程和交互（interleave）编程的含义。

第 2 章 代码详细解析

详细代码可以在这里找到：linux/drivers/mtd/nand/nand_base.c

2.1. 解析函数nand_get_flash_type

```
2407/*
2408 * Get the flash and manufacturer id and lookup if the type is supported
2409 */
2410static struct nand_flash_dev *nand_get_flash_type(struct mtd_info *mtd,
2411                                                  struct nand_chip *chip,
2412                                                  int busw, int *maf_id)
2413{
2414        struct nand_flash_dev *type = NULL;
2415        int i, dev_id, maf_idx;
2416        int tmp_id, tmp_manf;
2417
2418        /* Select the device */
2419        chip->select_chip(mtd, 0);❶
2420
2421        /*
2422         * Reset the chip, required by some chips (e.g. Micron MT29FxGxxxxx)
2423         * after power-up
2424         */
2425        chip->cmdfunc(mtd, NAND_CMD_RESET, -1, -1);
2426
2427        /* Send the command for reading device ID */
2428        chip->cmdfunc(mtd, NAND_CMD_READID❷, 0x00, -1);
2429
2430        /* Read manufacturer and device IDs */❸
2431        *maf_id = chip->read_byte(mtd);
2432        dev_id = chip->read_byte(mtd);
2433
2434        /* Try again to make sure, as some systems the bus-hold or other
2435         * interface concerns can cause random data which looks like a
2436         * possibly credible NAND flash to appear. If the two results do
2437         * not match, ignore the device completely.
2438         */
2439
2440        chip->cmdfunc(mtd, NAND_CMD_READID❹, 0x00, -1);
2441
2442        /* Read manufacturer and device IDs */
2443
2444        tmp_manf = chip->read_byte(mtd);
2445        tmp_id = chip->read_byte(mtd);
2446
2447        if (tmp_manf != *maf_id || tmp_id != dev_id) {
2448                printk(KERN_INFO "%s: second ID read did not match "
2449                        "%02x,%02x against %02x,%02x\n", __func__,
2450                        *maf_id, dev_id, tmp_manf, tmp_id);
2451                return ERR_PTR(-ENODEV);
2452        }
2453
2454        /* Lookup the flash id */
2455        for (i = 0; nand_flash_ids❺[i].name != NULL; i++) {
2456                if (dev_id == nand_flash_ids[i].id) {❻
```

```
2457                    type =  &nand_flash_ids[i];
2458                    break;
2459            }
2460    }
2461
2462    if (!type)
2463            return ERR_PTR(-ENODEV);
2464
2465    if (!mtd->name)
2466            mtd->name = type->name;
2467
2468    chip->chipsize = (uint64_t)type->chipsize << 20❼;
2469
2470    /* Newer devices have all the information in additional id bytes */
2471    if (!type->pagesize) {
2472            int extid;
2473            /* The 3rd id byte holds MLC / multichip data */
2474            chip->cellinfo❽ = chip->read_byte(mtd);
2475            /* The 4th id byte is the important one */❾
2476            extid = chip->read_byte(mtd);
2477            /* Calc pagesize */❿
2478            mtd->writesize = 1024 << (extid & 0x3);
2479            extid >>= 2;
2480            /* Calc oobsize */⓫
2481            mtd->oobsize = (8 << (extid & 0x01)) * (mtd->writesize >> 9);
2482            extid >>= 2;
2483            /* Calc blocksize. Blocksize is multiples of 64KiB */⓬
2484            mtd->erasesize = (64 * 1024) << (extid & 0x03);
2485            extid >>= 2;
2486            /* Get buswidth information */
2487            busw = (extid & 0x01) ? NAND_BUSWIDTH_16⓭ : 0;
2488
2489    } else {
2490            /*
2491             * Old devices have chip data hardcoded in the device id table ⓮
2492             */
2493            mtd->erasesize = type->erasesize;
2494            mtd->writesize = type->pagesize;
2495            mtd->oobsize = mtd->writesize / 32;
2496            busw = type->options & NAND_BUSWIDTH_16;
2497    }
2498
2499    /* Try to identify manufacturer */⓯
2500    for (maf_idx = 0; nand_manuf_ids[maf_idx].id != 0x0; maf_idx++) {
2501            if (nand_manuf_ids[maf_idx].id == *maf_id)
2502                    break;
2503    }
2504
2505    /*
2506     * Check, if buswidth is correct. Hardware drivers should set⓰
2507     * chip correct !
2508     */
2509    if (busw != (chip->options & NAND_BUSWIDTH_16)) {
2510            printk(KERN_INFO "NAND device: Manufacturer ID:"
2511                    " 0x%02x, Chip ID: 0x%02x (%s %s)\n", *maf_id,
2512                    dev_id, nand_manuf_ids[maf_idx].name, mtd->name);
2513            printk(KERN_WARNING "NAND bus width %d instead %d bit\n",
2514                    (chip->options & NAND_BUSWIDTH_16) ? 16 : 8,
```

```
2515                busw ? 16 : 8);
2516        return ERR_PTR(-EINVAL);
2517    }
2518
2519    /* Calculate the address shift from the page size */
2520    chip->page_shift⓱ = ffs(mtd->writesize) - 1;
2521    /* Convert chipsize to number of pages per chip -1. */
2522    chip->pagemask⓲ = (chip->chipsize >> chip->page_shift) - 1;
2523
2524    chip->bbt_erase_shift = chip->phys_erase_shift =
2525        ffs(mtd->erasesize) - 1;
2526    if (chip->chipsize & 0xffffffff)
2527        chip->chip_shift = ⓳ffs((unsigned)chip->chipsize) - 1;
2528    else
2529        chip->chip_shift = ffs((unsigned)(chip->chipsize >> 32)) + 32 - 1;
2530
2531    /* Set the bad block position */⓴
2532    chip->badblockpos = mtd->writesize > 512 ?
2533        NAND_LARGE_BADBLOCK_POS : NAND_SMALL_BADBLOCK_POS;
2534
2535    /* Get chip options, preserve non chip based options */㉑
2536    chip->options &= ~NAND_CHIPOPTIONS_MSK;
2537    chip->options |= type->options & NAND_CHIPOPTIONS_MSK;
2538
2539    /*
2540     * Set chip as a default. Board drivers can override it, if necessary
2541     */
2542    chip->options |= NAND_NO_AUTOINCR;㉒
2543
2544    /* Check if chip is a not a samsung device. Do not clear the
2545     * options for chips which are not having an extended id.
2546     */
2547    if (*maf_id != NAND_MFR_SAMSUNG && !type->pagesize)
2548        chip->options &= ~NAND_SAMSUNG_LP_OPTIONS;㉓
2549
2550    /* Check for AND chips with 4 page planes */
2551    if (chip->options & NAND_4PAGE_ARRAY)㉔
2552        chip->erase_cmd = multi_erase_cmd;
2553    else
2554        chip->erase_cmd = single_erase_cmd;
2555
2556    /* Do not replace user supplied command function ! */
2557    if (mtd->writesize > 512 && chip->cmdfunc == nand_command)
2558        chip->cmdfunc = nand_command_lp;㉕
2559
2560    printk㉖(KERN_INFO "NAND device: Manufacturer ID:"
2561        " 0x%02x, Chip ID: 0x%02x (%s %s)\n", *maf_id, dev_id,
2562        nand_manuf_ids[maf_idx].name, type->name);
2563
2564    return type;㉗
2565}
```

❶ 选中芯片，才能对其操作。
❷ 发送ReadID的命令：0x90，去取得芯片的ID信息
❸ 根据datasheet中的定义，第一个字节，简称byte1，是生产厂商的信息，不同的厂商，对应不同的数字。而byte2是芯片类型，不同的nand flash芯片，对应不同的设备ID，也就是一个字节的数字。

关于读取出来的ID的具体含义，可以参考三星K9K8G08U0A的datasheet中解释：

图 2.1. Nand Flash读取出来的各个ID的含义

ID Definition Table

90 ID : Access command = 90H

	Description
1st Byte	Maker Code
2nd Byte	Device Code
3rd Byte	Internal Chip Number, Cell Type, Number of Simultaneously Programmed Pages, Etc
4th Byte	Page Size, Block Size,Redundant Area Size, Organization, Serial Access Minimum
5th Byte	Plane Number, Plane Size

❹ 再次发送ReadID命令，其目的，上面注释代码中说了，有些特殊的系统中，第一次读取的信息，看起来是很正常，但是实际是错的，所以这里读两次，正常的设备，肯定都会一样的，如果两次不一样，那么说明设备有问题，也就直接函数返回了。

❺ 下面根据读取出来的flash ID，也就是具体flash芯片，或叫做设备ID，不同的数值，对应不同的容量和物理参数的flash。

其中，nand_flash_ids是个预先定义好的数组，其定义在：drivers\mtd\nand\nand_ids.c中，此处简要摘录如下：

```
/*
* Chip ID list
*
* Name. ID code, pagesize, chipsize in MegaByte, eraseblock size, options
*
* Pagesize; 0, 256, 512
* 0 get this information from the extended chip ID
+ 256 256 Byte page size
* 512 512 Byte page size
*/
struct nand_flash_dev❶ nand_flash_ids[] = {
......
    /* 4 Gigabit */
    {"NAND 512MiB 1,8V 8-bit", 0xAC, 0, 512, 0, LP_OPTIONS},
    {"NAND 512MiB 3,3V 8-bit", 0xDC, 0, 512, 0, LP_OPTIONS},
    {"NAND 512MiB 1,8V 16-bit", 0xBC, 0, 512, 0, LP_OPTIONS16},
    {"NAND 512MiB 3,3V 16-bit", 0xCC, 0, 512, 0, LP_OPTIONS16},

    /* 8 Gigabit */
    {"NAND 1GiB 1,8V 8-bit", 0xA3, 0, 1024, 0, LP_OPTIONS},
    {"NAND 1GiB 3,3V 8-bit", 0xD3, 0, 1024, 0, LP_OPTIONS},
    {"NAND 1GiB 1,8V 16-bit", 0xB3, 0, 1024, 0, LP_OPTIONS16},
    {"NAND 1GiB 3,3V 16-bit", 0xC3, 0, 1024, 0, LP_OPTIONS16},
......
}
```

❶ 而结构体nand_flash_dev的定义如下：include\linux\mtd\nand.h

```
/**
 * struct nand_flash_dev - NAND Flash Device ID Structure
 * @name: Identify the device type
 * @id:  device ID code
```

```
 * @pagesize: Pagesize in bytes. Either 256 or 512 or 0
 *  If the pagesize is 0, then the real pagesize
 *  and the eraseize are determined from the
 *  extended id bytes in the chip
 * @erasesize: Size of an erase block in the flash device.
 * @chipsize: Total chipsize in Mega Bytes
 * @options: Bitfield to store chip relevant options
 */
struct nand_flash_dev {
    char *name;
    int id;
    unsigned long pagesize;
    unsigned long chipsize;
    unsigned long erasesize;
    unsigned long options;
};
```

在结构体数组nand_flash_ids[]中，预先定义了，目前所支持的很多类型Nand Flash的具体物理参数，主要是上面结构体中的页大小pagesize，芯片大小chipsize，块大小erasesize，而id变量表示此类型的芯片，用哪个数字来表示。

❻ 此处通过刚读取到的设备ID，去和预先定义好的那个结构体数组nand_flash_ids[]中的每一个ID去比较，如果相等，那么说明支持此款nand falsh，而其他的信息，就可以直接从后面几项中直接获得了。

当pagesize为0的时候

如果pagesize是0，那么说明关于pagesize和其他一些信息，要通过读取额外的ID来获得，这也就是待会下面要详细解释的。

而对于旧的一些nand flash，在表项中其pagesize不是0，就可以直接可以从上面的预定义的表里面获得了。

比如，对于常见的三星的型号为K9K8G08U0A的nand flash，其设备号是0xD3，找到匹配的表项就是：

```
{"NAND 1GiB 3,3V 8-bit", 0xD3, 0, 1024, 0, LP_OPTIONS},
```

因此也就知道，其容量是1024MB，设备相关物理特性是1GiB 3,3V 8-bit了。

而关于pagesize和块大小erasesize此处都是0，就只能另外从后面读取的ID中获得了。

❼ 此处由于上面表中的chipsize是MB = 2^{10}Bytes为单位的，所以要左移20位，换算成byte单位

❽ 解释下面代码第三个字节之前，要先把图标帖出来，才更容易看得懂具体的解释：

图 2.2. Nand Flash 第三个ID的具体含义

3rd ID Data

	Description	I/O7	I/O6	I/O5 I/O4	I/O3 I/O2	I/O1 I/O0
Internal Chip Number	1 2 4 8					0 0 0 1 1 0 1 1
Cell Type	2 Level Cell 4 Level Cell 8 Level Cell 16 Level Cell				0 0 0 1 1 0 1 1	
Number of Simultaneously Programmed Pages	1 2 4 8			0 0 0 1 1 0 1 1		
Interleave Program Between multiple chips	Not Support Support		0 1			
Cache Program	Not Support Support	0 1				

由表中定义可以看出：

1. Internal Chip Number
   意思是，内部芯片有几颗。

   有些型号的Nand Flash，为了实现更高的容量，在芯片内部封装了多个芯片。

   比如三星的K9WAG08U1A容量是2GB，内部是装了2个单片是1GB的K9K8G08U0A，对应地，里面要包含2个片选CE1和CE2（均是低电平有效），而4GB的K9NBG08U5A包含了4片的K9K8G08U0A。

2. Cell Type：SLC / MLC
   bit2&bit3表示的是芯片的类型，是SLC还是某种MLC：

   - Bit2,bit3=0x00 ： SLC，简单说就是内部单个存储单元，存储一位的数据，所能表示的数值只有0，1，也就需要两种不同的电压来表示，所以叫做2 Level的Cell。

   - Bit2,bit3=0x01/0x10/0x11 ： 4 /8/16 Level Cell，都叫做MLC，其含义是内部单个存储单元设计成可以表示多个，即4/8/16个不同的电压，对应地，可以表示2，3，4位的数据。这类的MLC的nand flash，由于单个存储单元，要存储更多的数据，所以内部结构更复杂，读取和写入数据的逻辑更复杂，相对数据出错的几率也比SLC要大。

     所以，一般MLC的使用，都需要检错和纠错能力更强的硬件或软件算法，以保证数据的正确性。

     软件实现此类的多位数据的检错和纠错的效率相对较低，一般是硬件本身就已经提供此功能。

     对应的其为硬件ECC，也就是Linux内核MTD中的HW_ECC。

     其他关于SLC/MLC的更详细解释，感兴趣的可以去看另一个帖子：[【详解】如何编写Linux下Nand Flash驱动][1]

3. Number of Simultaneously Programmed Pages
   可以对几个页同时编程/写。

   此功能简单的说就是，一次性地写多个页的数据到对应的不同的页。

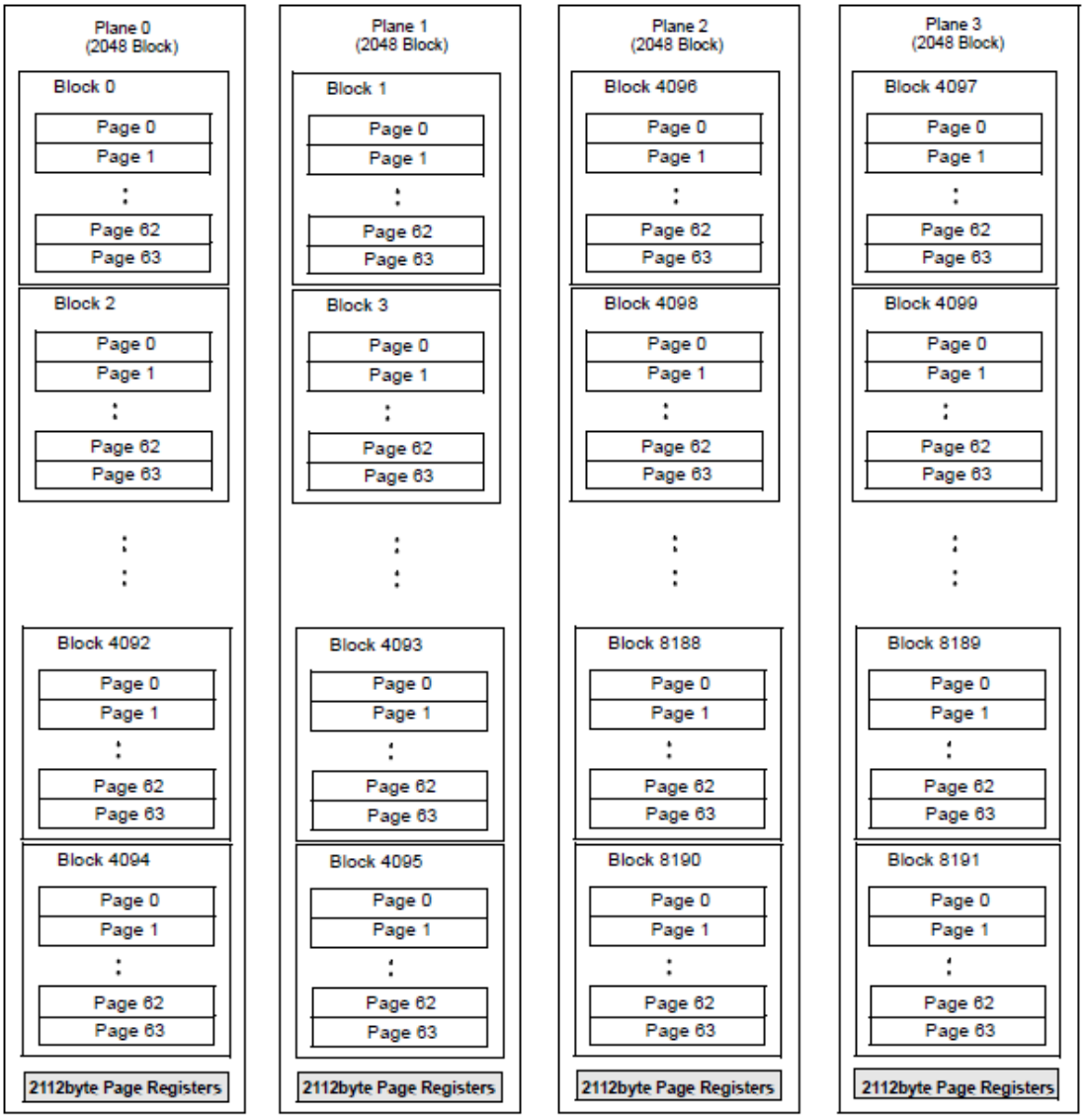
[1] http://www.crifan.com/files/doc/docbook/linux_nand_driver/release/html/linux_nand_driver.html

对应支持此操作的，硬件上必须要有多个plane，而每个plane，都有一个自己的页寄存器。

比如K9K8G08U0A有4个plane，分别叫做，plane0，plane1，plane2，plane3。

它们共分成2组，plane0和plane1，plane2和plane3。如图：

图 2.3. Nand Flash中多页编程对应的多个Plane的组织架构



在多页编程时候，只能对某一组中的两个plane操作，不允许类似于plane0和plane2或plane3一起去做多页编程。

以plane0和plane1为例，在实现具体的编程动作之前，将你要写入的2个页的数据，分别写入plane0和plane1中的页寄存器，然后才能发命令，去实现具体的编程操作。

正是因为多页编程需要底层的多plane支持，底层实现的时候，是同时对多个plane编程，所以，也被叫做Multi Plane Program

4. Interleave Program Between Multiple chips
交错，从字面意思就可以看出，此操作涉及对象就不止一个。

交错编程，就是对多个chip，交错地进行编程，先对一个编程，充分利用第一个编程过程中需要等待的时间，转去操作另一个，以此实现总体效率的提高。

如果支持Interleave Program的话，那么前面的chip number必然大于1。

5. Cache Program

- Cache读
  在开始了一次cache读之后，在你把数据读出去的这段时间，nand flash会自动地把下一页的数据读取出来放到页寄存器。

- Cache写
  在你写入数据的时候，对应的内存中的数据，不是直接写到页寄存器中，而是到了cache buffer中

  然后再发cache 写的命令，此时，数据才从cache buffer中，转递到页寄存器中，然后把数据一点点编程到nand flash

  此时，你可以去利用页编程的时间，去准备下一次的数据，然后依此地写入下一个页。

Cache读或写，是充分利用了读一页数据出来，或者将一页数据写到flash里面去的时间，去准备新的一页的数据，这样就可以实现连续的读或写，大大提高读写效率。

❾ 读取4th ID

4th ID的含义，如图：

图 2.4. Nand Flash 第4个ID的具体含义

4th ID Data

	Description	I/O7	I/O6	I/O5 I/O4	I/O3	I/O2	I/O1 I/O0
Page Size (w/o redundant area)	1KB 2KB 4KB 8KB						0 0 0 1 1 0 1 1
Block Size (w/o redundant area)	64KB 128KB 256KB 512KB			0 0 0 1 1 0 1 1			
Redundant Area Size (byte/512byte)	8 16					0 1	
Organization	x8 x16		0 1				
Serial Access Minimum	50ns/30ns 25ns Reserved Reserved	0 1 0 1			0 0 1 1		

❿ Page Size，如图 2.4 “Nand Flash 第4个ID的具体含义” 所示，页大小，是bit0和bit1组合起来所表示的。

extid & 0x3，就是取得bit0和bit1的值，而左移1024位，是因为上面表中的单位是KB = 2^{10}=1024。

此处关于1024 << (extid & 0x3)的含义

我之前也是看了很长时间，都没看懂，后来才看懂具体的意思的。

1024 << (extid & 0x3) 其实就是

= 1024 × (1<< (extid & 0x3))

= 前面的1024

= 之前的单位：KB

而后面的写法，即2的extid & 0x3的次方，比如，如果extid & 0x3是3，那么，1<<(extid & 0x3)就是1<<3=8，对应的上面的8KB。

⓫ Redundant Area Size(byte/512byte)

前面介绍过了，此处的oob，就是datasheet中的redundant area size就是linux中的oob大小。

上面表中的意思是，512个byte，对应8还是16个字节的redundant area。

之所以是512字节对应多少个，是因为以前的nand flash页大小是512（除了最早的好像是256之外），所以估计是硬件设计就这样设计了。512个字节对应多少个冗余的数据用作oob，而后来的页大小，对应的是512的整数倍，比如2K，4K等

所以，此处可以按照每个512对应几个字节的oob，然后再算页大小是512的多少倍，即：

此处的extid & 0x01算出来的值，对应上面的8或16，而mtd->writesize >> 9，其实就是

mtd->writesize /512，到此，才算清楚，为何此处oob是这么算的。

⓬ Block Size

具体算法很清楚，算出是64KB的多少倍，得出总大小。

此处之所以是64KB为基础，是因为已知最小的blocksize，就是64KB的。

⓭ Organization

X8/X16，表示的是，硬件I/O位宽（Bus Width）是8位的还是16位的。

目前大多数，都是X8的。

⓮ 旧的nand flash的一些参数，是知道设备ID后，可以直接从表中读取出来的。

⓯ 根据读取出来的生长厂商的ID，去和表中对应项匹配，找到是哪家的nand flash芯片。

其中，nand_manuf_ids和上面nand_flash_ids类似，也是个预先定义好的数组，其定义和nand_flash_ids同文件drivers\mtd\nand\nand_ids.c中：

```
/*
* Manufacturer ID list
*/
struct nand_manufacturers nand_manuf_ids[] = {
 {NAND_MFR_TOSHIBA, "Toshiba"},
 {NAND_MFR_SAMSUNG❶, "Samsung"},
 {NAND_MFR_FUJITSU, "Fujitsu"},
 {NAND_MFR_NATIONAL, "National"},
 {NAND_MFR_RENESAS, "Renesas"},
 {NAND_MFR_STMICRO, "ST Micro"},
 {NAND_MFR_HYNIX, "Hynix"},
 {NAND_MFR_MICRON, "Micron"},
 {NAND_MFR_AMD, "AMD"},
 {0x0, "Unknown"}
};
```

❶ 对应的各个厂家的宏和结构体的定义是在：include\linux\mtd\nand.h中：

```
/*
 * NAND Flash Manufacturer ID Codes
 */
```

```
#define NAND_MFR_TOSHIBA 0x98
#define NAND_MFR_SAMSUNG 0xec❶
#define NAND_MFR_FUJITSU 0x04
#define NAND_MFR_NATIONAL 0x8f
#define NAND_MFR_RENESAS 0x07
#define NAND_MFR_STMICRO 0x20
#define NAND_MFR_HYNIX  0xad
#define NAND_MFR_MICRON  0x2c
#define NAND_MFR_AMD  0x01

/**
 * struct nand_manufacturers - NAND Flash Manufacturer ID Structure
 * @name: Manufacturer name
 * @id:  manufacturer ID code of device.
*/
struct nand_manufacturers {
 int id;
 char * name;
};
```

❶ 我们最常读到的生产厂家的id：0xEC，就是对应这里的Samsung，表示此款nand flash是三星家的。

⑯ 检测你的驱动中的关于位宽的定义，是否和硬件所一致。

⑰ 此处计算的pagesize，blocksize等的shift，是为了后期的对这些值的除操作更加高效而做的。

对于代码中的除操作，如果直接只是/pagesize，则没有直接算出其是2的多少次方，然后用位移操作，更加高效。因此，此处直接计算好是多少个shift，以后的除于pagesize，blocksize，就可以直接用对应的位移操作了。

⑱ 算出mask，为了后期保证传入的地址不越界，所以会对其mask一下。

⑲ 这段，貌似是新的kernel里面新加的，而且把chip->chipsize定义换成uint64_t了，支持超过4GB大小的nand flash了

否则，如果你正好是4GB，对于旧的代码chip->chipsize是uint32_t类型的，那么正好就变成0了。

此处之所以去chip->chipsize & 0xffffffff判断是超过4GB，看起来，估计是ffs函数最多支持32位，所以，没法计算超过此大小的ffs。

ffs简介

简单说一下，ffs是计算一个数值的第一个被置位的位置，全称好像是 find first set bit

其简单解释如下：

ffs

查找第一个已设置的位

函数原型：int ffs (int x)

x为要搜索的字

刚百度了一下，好像还有个对应的函数：ffz，找到第一个0的位置，估计就是find first zero bit了。呵呵。

而且，这些，好像是Linux提供的基本函数库里面的，自己之前都没怎么听说过呢。汗一个先。。。

⑳ 设置坏块的标记位置。

关于nand flash的small block和large block，据我了解，好像就是对应的small pagesize和large pagesize，而此处的大小，是针对于旧的nand flash，其页大小pagesize是512

所以，Small block就是页大小是512B的nand flash，而larger block就是新的，页大小大于512B的，比如2KB，4KB等的nand flash。

下面的宏定义在include\linux\mtd\nand.h中：

```
/*
* Constants for oob configuration
*/
#define NAND_SMALL_BADBLOCK_POS  5
#define NAND_LARGE_BADBLOCK_POS  0
```

约定俗成的，small block的nand ，坏块标记在byte5，而large block的nand flash在byte0。

关于坏块标记，实际情况更复杂些：

Nand Flash坏块标记位置

对于2K页的nand flash，标记位置都是页内oob开始处，都是非0xFF表示坏块，

但是，对于是第几页，不同nand flash就有不同的规定了：

有些nand flash，是标记在坏块的第一个页（或者是第二个页，这点是考虑到，万一第一个页是坏的，所以才做此规定的。一般都是在第一个页处做标记）

比如三星的多数SLC，Hynix等

另一些，是在一个块内的最后一页或倒数第二页做此标记，比如samsung MLC，Numonyx等。

所以，真正比较完整的检查坏块的做法，至少要检测块内第一，第二，倒数第一，倒数第二页，是否是0xFF，才能比较全面的判断是否是坏块的。

21. 获得上面nand id表中的默认设置的那些option：LP_OPTIONS（如果是X16则是，LP_OPTIONS16）。
22. 自动增加页数？？？没太搞懂啥意思，估计是cache program/read相关的吧，目前据我了解的，好像页只有cache program/read，能和auto increment pages有点关系。
23. 如果有extentID且不是三星的nand flash，则清除掉上面我们默认设置的那些参数：LP_OPTIONS。
24. 一种特殊的nand flash，AND chip。所以，也要赋值给特殊的擦除函数。

    具体关于此类nand flash的介绍，感兴趣的自己参考上面drivers\mtd\nand\nand_ids.c中nand_flash_ids数组中的解释。
25. 如果nand flash的页大小是大于512B，也就是2KB，4KB等新的，被称作large block或largepage的nand flash，此处的lp，应该也就是large page的缩写。

    此类的nand flash比旧的，在发送地址的时候，多一个地址周期。具体参考datasheet。
26. 终于检测完所有需要的信息了。最后打印出nand flash的相关信息。
27. 活干完了，就可以return回家了，呵呵。