```
·····10·····20·····30·····40·····50·····60·····70·····80·····90·····100·····110·····120·····130·····140·····150
```

bbt坏块管理

日月 发表于 - 2010-3-2 9:59:00

2

推荐

前面看到在nand_scan（）函数的最后将会跳至scan_bbt（）函数，这个函数在nand_scan里面有定义：

```
2415 if (!this->scan_bbt)
2416 this->scan_bbt = nand_default_bbt;
```

nand_default_bbt（）位于Nand_bbt.c文件中。

```
1047 /**
     * nand_default_bbt - [NAND Interface] Select a default bad block table for the device
     * @mtd: MTD device structure
     *
     * This selects the default bad block table
     * support for the device and calls the nand_scan_bbt
    **/
    int nand_default_bbt (struct mtd_info *mtd)
    {
     struct nand_chip *this = mtd->priv;
```

这个函数的作用是建立默认的坏块表。

```
1059 /* Default for AG-AND. We must use a flash based
   * bad block table as the devices have factory marked
   * _good_ blocks. Erasing those blocks leads to loss
   * of the good / bad information, so we _must_ store
* this information in a good / bad table during
* startup
   */
   if (this->options & NAND_IS_AND) {
   /* Use the default pattern deors */
   if (!this->bbt_td) {
    this->bbt_td = &bbt_main_descr;
    this->bbt_md = &bbt_mirror_descr;
   }
    this->options |= NAND_USE_FLASH_BBT;
    return nand_scan_bbt (mtd, &agand_flashbased);
   }
```

如果Flash的类型是AG-AND（这种Flash类型比较特殊，既不是MLC又不是SLC，因此不去深究了，而且好像瑞萨要把它淘汰掉），需要使用默认的模式描述符，最后再进入nand_scan_bbt（）函数。

```
1078 /* Is a flash based bad block table requested ? */
   if (this->options & NAND_USE_FLASH_BBT) {
   /* Use the default pattern deors */
   if (!this->bbt_td) {
    this->bbt_td = &bbt_main_descr;
    this->bbt_md = &bbt_mirror_descr;
   }
   if (!this->badblock_pattern) {
    this->badblock_pattern = (mtd->oobblock > 512) ?
     &largepage_flashbased : &smallpage_flashbased;
   }
   } else {
   this->bbt_td = NULL;
   this->bbt_md = NULL;
   if (!this->badblock_pattern) {
    this->badblock_pattern = (mtd->oobblock > 512) ?
     &largepage_memorybased : &smallpage_memorybased;
   }
   }

   return nand_scan_bbt (mtd, this->badblock_pattern);
```

如果Flash芯片需要使用坏块表，对于1208芯片来说是使用smallpage_memorybased。

```
985   static struct nand_bbt_descr smallpage_memorybased = {
   .options = NAND_BBT_SCAN2NDPAGE,
   .offs = 5,
   .len = 1,
   .pattern = scan_ff_pattern
   };
```

暂时没看到如何使用这些赋值，先放着。后面检测坏块时用得着。

```
1099 return nand_scan_bbt (mtd, this->badblock_pattern);
```

最后将badblock_pattern作为参数，调用nand_can_bbt函数。

```
844   /**
845
* nand_scan_bbt - [NAND Interface] scan, find, read and maybe create bad block table(s
)
846    * @mtd: MTD device structure
847    * @bd:   deor for the good/bad block search pattern
848    *
849    * The checks, if a bad block table(s) is/are already
850    * available. If not it scans the device for manufacturer
851    * marked good / bad blocks and writes the bad block table(s) to
852    * the selected place.
853    *
854    * The bad block table memory is allocated here. It must be freed
855    * by calling the nand_free_bbt .
856    *
857    */
858    int nand_scan_bbt (struct mtd_info *mtd, struct nand_bbt_descr *bd)
859    {
```

检测、寻找、读取甚至建立坏块表。函数检测是否已经存在一张坏块表，否则建立一张。坏块表的内存分配也在这个函数中。

```
860 struct nand_chip *this = mtd->priv;
861 int len, res = 0;
862 uint8_t *buf;
863 struct nand_bbt_descr *td = this->bbt_td;
864 struct nand_bbt_descr *md = this->bbt_md;
865 len = mtd->size >> (this->bbt_erase_shift + 2);
866 /* Allocate memory (2bit per block) */
867 this->bbt = kmalloc (len, GFP_KERNEL);
868 if (!this->bbt) {
869    printk (KERN_ERR "nand_scan_bbt: Out of memory/n");
870    return -ENOMEM;
871 }
872 /* Clear the memory bad block table */
873 memset (this->bbt, 0x00, len);
```

一些赋值、变量声明、内存分配，每个block分配2bit的空间。1208有4096个block，应该分配4096*2bit的空间。

```
877 /* If no primary table decriptor is given, scan the device
878 * to build a memory based bad block table
879 */
880 if (!td) {
881    if ((res = nand_memory_bbt(mtd, bd))) {
882        printk (KERN_ERR "nand_bbt: Can't scan flash and build the RAM-based BBT/n");
883     kfree (this->bbt);
884     this->bbt = NULL;
885    }
886    return res;
887 }
```

如果没有提供ptd，就扫描设备并建立一张。这里调用了nand_memory_bbt（）这个内联函数。

```
653 /**
654    * nand_memory_bbt - [GENERIC] create a memory based bad block table
655    * @mtd: MTD device structure
656    * @bd:   deor for the good/bad block search pattern
657    *
658    * The creates a memory based bbt by scanning the device
659    * for manufacturer / software marked good / bad blocks
660    */
661
662  static inline int nand_memory_bbt (struct mtd_info *mtd, struct nand_bbt_descr *bd)
663
664    {
665    struct nand_chip *this = mtd->priv;
666    bd->options &= ~NAND_BBT_SCANEMPTY;
667    return create_bbt (mtd, this->data_buf, bd, -1);
668    }
```

函数的作用是建立一张基于memory的坏块表。

将操作符的NAND_BBT_SCANEMPTY清除，并继续调用creat_bbt（）函数。

```
671 /**
```

```
129.     * create_bbt - [GENERIC] Create a bad block table by scanning the device
130.     * @mtd: MTD device structure
131.     * @buf: temporary buffer
132.     * @bd:   deor for the good/bad block search pattern
133.     * @chip: create the table for a specific chip, -1 read all chips.
134.     *   Applies only if NAND_BBT_PERCHIP option is set
135.     *
136.     * Create a bad block table by scanning the device
137.     * for the given good/bad block identify pattern
138.     */
139.
      static int create_bbt (struct mtd_info *mtd, uint8_t *buf, struct nand_bbt_descr *bd
    , int chip)
140.      {
```

真正的建立坏块表函数。chip参数是-1表示读取所有的芯片。

```
142. 284 struct nand_chip *this = mtd->priv;
143. int i, j, numblocks, len, scanlen;
144. int startblock;
145. loff_t from;
146. size_t readlen, ooblen;
147. printk (KERN_INFO "Scanning device for bad blocks/n");
```

一些变量声明，开机时那句话就是在这儿打印出来的。

```
149. 292 if (bd->options & NAND_BBT_SCANALLPAGES)
150. len = 1 << (this->bbt_erase_shift - this->page_shift);
151. else {
152.    if (bd->options & NAND_BBT_SCAN2NDPAGE)
153.     len = 2;
154.    else
155.     len = 1;
156. }
```

在前面我们定义了smallpage_memorybased这个结构体，现在里面NAND_BBT_SCANALLPAGES的终于用上了，对于1208芯片来说，len=2。

```
158. 304 if (!(bd->options & NAND_BBT_SCANEMPTY)) {
159.    /* We need only read few bytes from the OOB area */
160.    scanlen = ooblen = 0;
161.    readlen = bd->len;
162. } else {
163.    /* Full page content should be read */
164.    scanlen = mtd->oobblock + mtd->oobsize;
165.    readlen = len * mtd->oobblock;
166.    ooblen = len * mtd->oobsize;
167. }
```

前面已经将NAND_BBT_SCANEMPTY清除了，这里肯定执行else的内容。需要将一页内容都读取出来。

```
169. 316 if (chip == -1) {
170.    /* Note that numblocks is 2 * (real numblocks) here, see i+=2 below as it
171.    * makes shifting and masking less painful */
172.    numblocks = mtd->size >> (this->bbt_erase_shift - 1);
173.    startblock = 0;
174.    from = 0;
175. } else {
176.    if (chip >= this->numchips) {
177.     printk (KERN_WARNING "create_bbt(): chipnr (%d) > available chips (%d)/n",
178.      chip + 1, this->numchips);
179.     return -EINVAL;
180.    }
181.    numblocks = this->chipsize >> (this->bbt_erase_shift - 1);
182.    startblock = chip * numblocks;
183.    numblocks += startblock;
184.    from = startblock << (this->bbt_erase_shift - 1);
185. }
```

前面提到chip为-1，实际上我们只有一颗芯片，numblocks这儿是4096*2。

```
187. 335 for (i = startblock; i < numblocks;) {
188.    int ret;
189.    if (bd->options & NAND_BBT_SCANEMPTY)
190.     if ((ret = nand_read_raw (mtd, buf, from, readlen, ooblen)))
191.      return ret;
192.    for (j = 0; j < len; j++) {
193.     if (!(bd->options & NAND_BBT_SCANEMPTY)) {
194.      size_t retlen;
195.      /* Read the full oob until read_oob is fixed to
```

```
     * handle single byte reads for 16 bit buswidth */
     ret = mtd->read_oob(mtd, from + j * mtd->oobblock,
        mtd->oobsize, &retlen, buf);
     if (ret)
      return ret;
     if (check_short_pattern (buf, bd)) {
       this->bbt[i >> 3] |= 0x03 << (i & 0x6);
      printk (KERN_WARNING "Bad eraseblock %d at 0x%08x/n",
       i >> 1, (unsigned int) from);
        break;
     }
    } else {
     if (check_pattern (&buf[j * scanlen], scanlen, mtd->oobblock, bd)) {
         this->bbt[i >> 3] |= 0x03 << (i & 0x6);
      printk (KERN_WARNING "Bad eraseblock %d at 0x%08x/n",
       i >> 1, (unsigned int) from);
         break;
     }
    }
   }
   i += 2;
   from += (1 << this->bbt_erase_shift);
}
return 0;
```

检测这4096个block，刚开始的nand_read_raw肯定不会执行。len是2，在j循环要循环2次。

每次循环真正要做的事情是下面的内容：

```
ret = mtd->read_oob(mtd, from + j * mtd->oobblock, mtd->oobsize, &retlen, buf);
```

read_oob（）函数在nand_scan（）里被指向nand_read_oob（），这个函数在Nand_base.c文件中，看来得回Nand_base.c看看了。

```
1397 /**
   * nand_read_oob - [MTD Interface] NAND read out-of-band
   * @mtd: MTD device structure
   * @from: offset to read from
   * @len: number of bytes to read
   * @retlen: pointer to variable to store the number of read bytes
   * @buf: the databuffer to put data
   *
   * NAND read out-of-band data from the spare area
   */
static int nand_read_oob (struct mtd_info *mtd, loff_t from, size_t len, size_t * retl
en, u_char * buf)
    {
```

才发现oob全称是out-of-band，from是偏移量，len是读取的长度，retlen是存储指针。

```
1409 int i, col, page, chipnr;
struct nand_chip *this = mtd->priv;
int blockcheck = (1 << (this->phys_erase_shift - this->page_shift)) - 1;
DEBUG (MTD_DEBUG_LEVEL3, "nand_read_oob: from = 0x%08x, len = %i/n", (unsigned int) fr
om, (int) len);
/* Shift to get page */
page = (int)(from >> this->page_shift);
chipnr = (int)(from >> this->chip_shift);
/* Mask to get column */
col = from & (mtd->oobsize - 1);
/* Initialize return length value */
*retlen = 0;
```

一些初始化，blockcheck对于1208应该是（1<<(0xe-0x9)-1）=31。然后通过偏移量计算出要读取oob区的page，chipnr和col。

```
1425 /* Do not allow reads past end of device */
if ((from + len) > mtd->size) {
   DEBUG (MTD_DEBUG_LEVEL0, "nand_read_oob: Attempt read beyond end of device/n");
   *retlen = 0;
   return -EINVAL;
}
/* Grab the lock and see if the device is available */
nand_get_device (this, mtd , FL_READING);
/* Select the NAND device */
this->select_chip(mtd, chipnr);
/* Send the read command */
this->cmdfunc (mtd, NAND_CMD_READOOB, col, page & this->pagemask);
```

不允许非法的读取，获取芯片控制权，发送读取OOB命令，这儿会调用具体硬件驱动中相关的Nand控制函数。

```
1442 /*
* Read the data, if we read more than one page
* oob data, let the device transfer the data !
*/
i = 0;
while (i < len) {
   int thislen = mtd->oobsize - col;
   thislen = min_t(int, thislen, len);
   this->read_buf(mtd, &buf[i], thislen);
   i += thislen;
   /* Read more ? */
   if (i < len) {
    page++;
    col = 0;
    /* Check, if we cross a chip boundary */
    if (!(page & this->pagemask)) {
     chipnr++;
     this->select_chip(mtd, -1);
     this->select_chip(mtd, chipnr);
    }
    /* Apply delay or wait for ready/busy pin
    * Do this before the AUTOINCR check, so no problems
    * arise if a chip which does auto increment
    * is marked as NOAUTOINCR by the board driver.
    */
    if (!this->dev_ready)
     udelay (this->chip_delay);
    else
     nand_wait_ready(mtd);
    /* Check, if the chip supports auto page increment
    * or if we have hit a block boundary.
    */
    if (!NAND_CANAUTOINCR(this) || !(page & blockcheck)) {
     /* For subsequent page reads set offset to 0 */
           this->cmdfunc (mtd, NAND_CMD_READOOB, 0x0, page & this->pagemask);
    }
   }
}
/* Deselect and wake up anyone waiting on the device */
nand_release_device(mtd);
/* Return happy */
*retlen = len;
return 0;
```

开始读取数据，while循环只要获取到oob区大小的数据即可。注意，read_buf才是最底层的读写Nand的函数，在我们的驱动中根据参数可以实现读取528byte全部内容，或者16byte的oob区。

如果一次没读完，就要继续再读，根据我们实际使用经验好像没出现过这种问题。

最后Return Happy~回到Nand_bbt.c的creat_bbt（）函数，348行，好像都快忘记我们还没出creat_bbt（）函数呢，我再把他贴一遍吧：

```
346   /* Read the full oob until read_oob is fixed to
   * handle single byte reads for 16 bit buswidth */
   ret = mtd->read_oob(mtd, from + j * mtd->oobblock,
        mtd->oobsize, &retlen, buf);
     if (ret)
      return ret;
     if (check_short_pattern (buf, bd)) {
       this->bbt[i >> 3] |= 0x03 << (i & 0x6);
      printk (KERN_WARNING "Bad eraseblock %d at 0x%08x/n",
       i >> 1, (unsigned int) from);
        break;
      }
     } else {
      if (check_pattern (&buf[j * scanlen], scanlen, mtd->oobblock, bd)) {
          this->bbt[i >> 3] |= 0x03 << (i & 0x6);
       printk (KERN_WARNING "Bad eraseblock %d at 0x%08x/n",
        i >> 1, (unsigned int) from);
          break;
       }
     }
    }
    i += 2;
```

```
330.    from += (1 << this->bbt_erase_shift);
331.  }
332.  return 0;
333.      }
```

334. 刚刚如果不是Ruturn Happy，下面的352行就会返回错误了。接着会调用check_short_pattern（）这个函数。

```
335.  113 /**
336.     * check_short_pattern - [GENERIC] check if a pattern is in the buffer
337.     * @buf: the buffer to search
338.     * @td:   search pattern deor
339.     *
340.     * Check for a pattern at the given place. Used to search bad block
341.     * tables and good / bad block identifiers. Same as check_pattern, but
342.     * no optional empty check
343.     *
344.      */
345.     static int check_short_pattern (uint8_t *buf, struct nand_bbt_descr *td)
346.  {
347.  int i;
348.  uint8_t *p = buf;
349.  /* Compare the pattern */
350.  for (i = 0; i < td->len; i++) {
351.     if (p[td->offs + i] != td->pattern[i])
352.      return -1;
353.  }
354.  return 0;
355.  }
```

356. 检查读到的oob区是不是坏块就靠这个函数了。前面放了好久的struct nand_bbt_descr smallpage_memorybased终于用上了，挨个对比，有一个不一样直接返回-1，坏块就这样产生了。下面会将坏块的位置打印出来，并且将坏块记录在bbt表里面，在nand_scan_bbt（）函数的开始我们就为bbt申请了空间。

```
357.  this->bbt[i >> 3] |= 0x03 << (i & 0x6);
```

358. 为啥要右移3bit呢？首先i要右移1bit，因为前面乘以了2。由于没个block占用2bit的空间，一个char变量8bit，所以还再要右移2bit吧。

359. 下面的check_pattern（）函数调用不到的。

360. 依次检测完所有block，creat_bbt（）函数也顺利返回。

361. 这样nand_memory_bbt（）函数也正确返回。

362. 接着是nand_scan_bbt（）同样顺利结束。

363. 最后nand_default_bbt（）完成。

364. 整个nand_scan（）的工作终于完成咯，好长。