Flow :

* nand\_read\_skip\_bad
* check\_skip\_len
* nand\_block\_isbad
* nand\_block\_checkbad
* BBT scanned?
* If no
* -nand\_default\_bbt
* use data in the flash to create bbt?
* nand\_scan\_bbt
* 使用指定的page搜尋bbt或是掃過所有page的oob尋找bbt mark
* -search\_read\_bbts () -> serch bbt()->scan\_read\_raw() -> check\_pattern()
* -check\_create()->creat\_bbt()->scan\_block\_fast ->check\_short\_pattern()
* -write\_bbt()->nand\_erase\_nand() -> scan\_write\_bbt-> nand -> write\_oob()

->nand\_fill\_oob()

* mark\_bbt\_region()
* (Chip->bbt)
* -nand\_block\_bad()-> read\_oob()
* nand\_isbad\_bbt()

int nand\_read\_skip\_bad(nand\_info\_t \*nand, loff\_t offset, size\_t \*length, u\_char \*buffer)

{

need\_skip = check\_skip\_len(nand, offset, \*length);

if (need\_skip < 0) {

printf ("Attempt to read outside the flash area\n");

\*length = 0;

檢查壞塊，往後尋找好塊直到滿足寫入長度，返回需要跳過的長度

return -EINVAL;

}

if (!need\_skip) {

rval = nand\_read (nand, offset, length, buffer);

if (!rval || rval == -EUCLEAN)

如果好塊滿足資料長度，則執行動作，無錯誤時返回。

return 0;

\*length = 0;

printf ("NAND read from offset %llx failed %d\n",

offset, rval);

return rval;

}

while (left\_to\_read > 0) {

size\_t block\_offset = offset & (nand->erasesize - 1);

size\_t read\_length;

WATCHDOG\_RESET ();

if (nand\_block\_isbad (nand, offset & ~(nand->erasesize - 1))) {

printf ("Skipping bad block 0x%08llx\n",

offset & ~(nand->erasesize - 1));

offset += nand->erasesize - block\_offset;

continue;

}

如果當前塊是壞塊，offset跳到下一個塊的第0個offset，重複直到好塊

if (left\_to\_read < (nand->erasesize - block\_offset))

read\_length = left\_to\_read;

else

read\_length = nand->erasesize - block\_offset;

計算長度，單次最大讀寫長度不可以超過blocksize

rval = nand\_read (nand, offset, &read\_length, p\_buffer);

}

這個function的功能為在記憶體建立BBT table

int nand\_scan\_bbt(struct mtd\_info \*mtd, struct nand\_bbt\_descr \*bd)

{

struct nand\_chip \*this = mtd->priv;

int len, res = 0; uint8\_t \*buf;

struct nand\_bbt\_descr \*td = this->bbt\_td;//main bbt

struct nand\_bbt\_descr \*md = this->bbt\_md;//mirror bbt

len = mtd->size >> (this->bbt\_erase\_shift + 2);

需要的記憶體長度等於卡容量除以BLOCK SIZE，每個BLOCK用2個bit表示，

一個byte可以存4個BLOCK，所以再除4()

this->bbt = kzalloc(len, GFP\_KERNEL);

if (!this->bbt) {

printk(KERN\_ERR "nand\_scan\_bbt: Out of memory\n");

return -ENOMEM;

}

if (!td) {

if ((res = nand\_memory\_bbt(mtd, bd))) {

printk(KERN\_ERR "nand\_bbt: Can't scan flash and build the RAM-based

BBT\n");

kfree(this->bbt);

this->bbt = NULL;

}

return res;

}

//td = this->bbt\_td = &bbt\_main\_descr，td已賦值

如果td(Primary table decriptor)沒有賦值,則掃描設備建立一個memory base 的

BBT

len = 4320;

buf = vmalloc(len);

if (td->options & NAND\_BBT\_ABSPAGE) {

res = read\_abs\_bbts(mtd, buf, td, md);

從指定的page讀取BBT

} else {

搜索存在OOB區域的BBT標記來確認BBT在哪個BLOCK

res = search\_read\_bbts(mtd, buf, td, md);

}

if (res)

res = check\_create(mtd, buf, bd);

不管有沒有找到BBT, res都會強制返回1以進入check\_create函數, 掃描找到的BBT

在Memory建立bbt表,或是沒有找到BBT,必須掃描整個flash建立新的BBT都在

Check\_create函數裡實現

mark\_bbt\_region(mtd, td);

if (md)

mark\_bbt\_region(mtd, md);

設立標記避免BBT區域被erase或寫入

vfree(buf);

return res;

}

int nand\_default\_bbt(struct mtd\_info \*mtd)

{

struct nand\_chip \*this = mtd->priv;

if (this->options & NAND\_USE\_FLASH\_BBT) {

if (!this->bbt\_td) {

this->bbt\_td = &bbt\_main\_descr;

this->bbt\_md = &bbt\_mirror\_descr;

} 主BBT描述表, 定義BBT的offset, 用多少bit表示一個block的狀態

,BBT版本, 判定為BBT的標記... 等

if (!this->badblock\_pattern) {

this->badblock\_pattern = (mtd->writesize > 512) ?

&largepage\_flashbased : &smallpage\_flashbased;

}bad block標記描述表,定義bad block標記的位置,pattern,長度

}else {

this->bbt\_td = NULL;

this->bbt\_md = NULL;

if (!this->badblock\_pattern) {

printf("memorybased \n");

this->badblock\_pattern = (mtd->writesize > 512) ?

&largepage\_memorybased : &smallpage\_memorybased;

}

}

return nand\_scan\_bbt(mtd, this->badblock\_pattern);

}

static int check\_create(struct mtd\_info \*mtd, uint8\_t \*buf, struct nand\_bbt\_descr \*bd)

{

int i, chips, writeops, chipsel, res;

struct nand\_chip \*this = mtd->priv;

struct nand\_bbt\_descr \*td = this->bbt\_td;

struct nand\_bbt\_descr \*md = this->bbt\_md;

struct nand\_bbt\_descr \*rd, \*rd2;

/\* Do we have a bbt per chip ? \*/

if (td->options & NAND\_BBT\_PERCHIP)

chips = this->numchips;

else

chips = 1;

for (i = 0; i < chips; i++) {

writeops = 0;

rd = NULL;

rd2 = NULL;

/\* Per chip or per device ? \*/

chipsel = (td->options & NAND\_BBT\_PERCHIP) ? i : -1;

/\* Mirrored table avilable ? \*/

//td = primary bbt, md = mirror bbt

//td->pages[i] == -1 mean not found bbt

if (md) {

if (td->pages[i] == -1 && md->pages[i] == -1) {

writeops = 0x03;

goto create;

}

if (td->pages[i] == -1) {

rd = md;

td->version[i] = md->version[i];

writeops = 1;

goto writecheck;

}

if (md->pages[i] == -1) {

rd = td;

md->version[i] = td->version[i];

writeops = 2;

goto writecheck;

}

if (td->version[i] == md->version[i]) {

rd = td;

if (!(td->options & NAND\_BBT\_VERSION))

rd2 = md;

goto writecheck;

}

if (((int8\_t) (td->version[i] - md->version[i])) > 0) {

rd = td;

md->version[i] = td->version[i];

writeops = 2;

} else {

rd = md;

td->version[i] = md->version[i];

writeops = 1;

}

goto writecheck;

} else {

if (td->pages[i] == -1) {

writeops = 0x01;

goto create;

}

rd = td;

goto writecheck;

}

create:

/\* Create the bad block table by scanning the device ? \*/

if (!(td->options & NAND\_BBT\_CREATE)){

printf ("Create the bad block table by scanning the device\n");

continue;

}

/\* Create the table in memory by scanning the chip(s) \*/

create\_bbt(mtd, buf, bd, chipsel);

td->version[i] = 1;

if (md)

md->version[i] = 1;

writecheck:

/\* read back first ? \*/

if (rd){

printf ("read back first 1\n");

read\_abs\_bbt(mtd, buf, rd, chipsel);

}

/\* If they weren't versioned, read both. \*/

if (rd2){

printf ("read back first 2\n");

read\_abs\_bbt(mtd, buf, rd2, chipsel);

}

/\* Write the bad block table to the device ? \*/

if ((writeops & 0x01) && (td->options & NAND\_BBT\_WRITE)) {

printf ("Write the bad block table to the device \n");

res = write\_bbt(mtd, buf, td, md, chipsel);

if (res < 0)

return res;

}

/\* Write the mirror bad block table to the device ? \*/

if ((writeops & 0x02) && md && (md->options & NAND\_BBT\_WRITE)) {

printf ("Write the mirror bad block table to the device \n");

res = write\_bbt(mtd, buf, md, td, chipsel);

if (res < 0)

return res;

}

}

return 0;

}

Largepage\_flashbase 下BBT 所在page的oob結構

ff ff ff ff ff ff ff ff 42 62 74 30 01 ff ff ff

byte [0] = bad block mark 🡺 ff

byte [11:8] = {‘’B, ’b’, ’t’, ’0’} 🡺 42 62 74 30

byte [12] = BBT version 🡺 01