Руководство по установке Linux на Nand, с обработкой битых блоков.

Используемый nand - K9F2G08U0C-SCB (Samsung).

Listing 1: Характеристики nand

```
1 Eresesize(block size): 131072 = 0x00020000
2 Writesize(page size) : 2048   = 0x00000800
```

Битые блоки, бывают двух типов, заводские, и которые появились в процессе работы. Ели дамп блока содержит одни нули (так же и в области oob), это вероятнее всего, что это заводской bad.

С завода, nand выходит с заводскими битыми блоками, но без таблицы битых блоков. Статус блока определяется первым байтом в области oob (spare area). Samsung гарантирует, что либо первая либо вторая страница каждого блока, содержит НЕ FFh по адресу 2048 (бервый байт области oob, ну, размер страницы 2048, начиная с нуля). Система должна уметь разпознать заводские битые блоки, и создать таблицу битых блоков. На рис.(1), предложенная в даташите [**?**]. схема создания таблицы битых блоков.

Это договоренность, которой придерживаются разработчки аппаратуры, и ПО. И на основе этой договоренности, различное ПО, в том числе и u-boot, в востоянии создать таблицу битых блоков, и работать с ней. Ядро, как будет показано ниже, тоже работает с этой таблицей.

Мне досталась nand, с уже созданной талбицей битых блоков. Запишем адреса этих битых блоков.

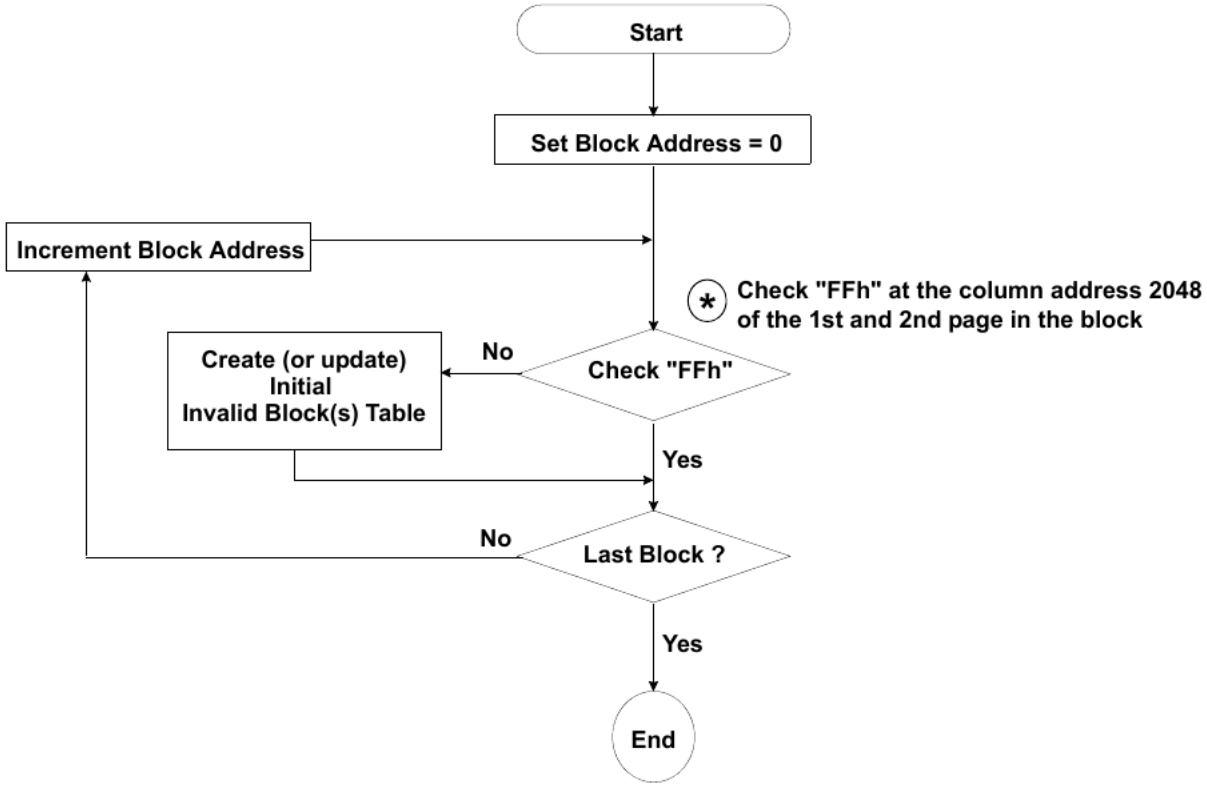

Рис. 1: Flow chart to create initial block table

Listing 2: Доставшиеся мне битые блоки

```
1 U-Boot: nand bad
2
3 Device 0 Bad blocks:
4     0bdc0000
5     0ff80000
6     0ffa0000
7     0ffc0000
8     0ffe0000
```

Блок по адресу 0x0bdc0000 это заводской bad. В этом можно убедится сделав дамп этого блока из u-boot. Этот блок содержит одни нули, что в соответствии с datasheet означает, это это заводской bad. Далее, идут подряд 4-ре блока. Последние два, это сама таблица, первые два зарезервированы u-boot.

Запишем процесс установки Linux из u-boot, без стирания исходной таблицы би-

тых блоков:

Listing 3: Установка Linux из U-Boot без стирания таблицы битых блоков

```
U-Boot: nand erase.chip
U-Boot: mmc rescan
U-Boot: nandecc hw 2
U-Boot: mw.b 0x81000000 0xff 0x20000
U-Boot: fatload mmc 0 0x81000000 MLO
U-Boot: nand write 0x81000000 0x0 0x20000
U-Boot: nand write 0x81000000 0x20000 0x20000
U-Boot: nand write 0x81000000 0x40000 0x20000
U-Boot: nand write 0x81000000 0x60000 0x20000
U-Boot: mw.b 0x81000000 0xff 0x200000
U-Boot: fatload mmc 0 0x81000000 u-boot.img
U-Boot: nand write 0x81000000 0x80000 0x200000
U-Boot: nandecc hw 1
U-Boot: mw.b 0x81000000 0xff 0x500000
U-Boot: fatload mmc 0 0x81000000 uImage
U-Boot: nand write 0x81000000 0x280000 0x500000
U-Boot: mw.b 0x81000000 0xff 0x0f880000
U-Boot: fatload mmc 0 0x81000000 rootfs.jffs2
U-Boot: nand write 0x81000000 0x780000 0x0f880000
```

Теперь сотрем полностью nand, вместе с таблицей битых блоков, и с самими битыми блоками. При этом, мы всегда можем пометить блок как bad, руками из u-boot.

Listing 4: Полное стирание nand(вместе с таблицец bbt)

```
U-Boot: nand scrub 0x0 0x10000000
Erasing at 0xbd60000 -- 74% complete.
NAND 256Mib 3,3V 8-bit: MTD Erase failrue: -5
Erasing at 0xffe0000 --100% complete
Bad block table not found for chip 0
Bad block table not found for chip 0
Bad block table written to 0x00000ffe0000, version 0x01
Bad block table written to 0x00000ffc0000, version 0x01
OK
```

Данная команда, также, утановит почти все битые блоки как хорошие блоки. В результате u-boot стер, и автоматически воссоздал таблицу битых блоков. Посмотрим, что получилось:

Listing 5: Вновь воссозданная таблица bbt

```
U-Boot: nand bad

```

```
3 Device 0 Bad blocks:
4    0bdc0000
5    0ff80000
6    0ffa0000
7    0ffc0000
8    0ffe0000
```

Теперь, сравним набор битых блоков из листинга (2), с набором из последнего листинга. Если мы ничего "не потеряли" то можно установить систему. Если некоторые битые блоки не были вновь помечены u-boot, то пометим их самостоятельно. Пример:

Listing 6: Пример маркирования битого блока

```
1 U-Boot: nand markbad 0x0bda0000
2 Bad block table written to 0x00000ffe0000, version 0x02
3 Bad block table written to 0x00000ffc0000, version 0x02
4 block 0x0bda0000 succesfully marked as bad
5 U-Boot:
6 U-Boot: nand bad
7
8 Device 0 Bad blocks:
9    0bdc0000
10   0ff80000
11   0ffa0000
12   0ffc0000
13   0ffe0000
```

Теперь, можно полностью повторить процесс установки Linux из листинга (3), но уже без первой команды nand erase.chip

А мы тем временем попробуем стереть таблицу из под ядра, используя утилиту flash-erase из набора mtd-utils:

Listing 7: flasherase

```
1 omap2sh: flash_erase -N /dev/mtd4 0x0 0x0
2 nand_erase_nand:
3    attempt to erase bad block at page 0x00017b40
4 nand_erase_nand:
5    attempt to erase bad block at page 0x00017b80
6 complete libmtd:error!:
7    MEMERASE64 ioctl failed for eraseblock 1457
8 complete libmtd:error!:
9    MEMERASE64 ioctl failed for eraseblock 1458
```

```
nand_erase_nand:
   attempt to erase bad block at page 0x0001ff00
nand_erase_nand:
   attempt to erase bad block at page 0x0001ff40
nand_erase_nand:
   attempt to erase bad block at page 0x0001ff80
nand_erase_nand:
   attempt to erase bad block at page 0x0001ffc0
   MEMERASE64 ioctl failed for eraseblock 1984
   MEMERASE64 ioctl failed for eraseblock 1985
   MEMERASE64 ioctl failed for eraseblock 1986
   MEMERASE64 ioctl failed for eraseblock 1987
```

Теперь приведем все адреса к абсолютному смещению в битах:

Listing 8: Приведем адреса

```
page: 0x00017b40  * 0x800 = 0x0bda0000
page: 0x00017b80  * 0x800 = 0x0bdc0000
page: 0x0001ff00  * 0x800 = 0x0ff80000
page: 0x0001ff40  * 0x800 = 0x0ffa0000
page: 0x0001ff80  * 0x800 = 0x0ffc0000
page: 0x0001ffc0  * 0x800 = 0x0ffe0000

eraseblock: 1457 * 64  * 2048 = 0x0b620000
eraseblock: 1458 * 64  * 2048 = 0x0b640000
eraseblock: 1984 * 64  * 2048 =
eraseblock: 1985 * 64  * 2048 =
eraseblock: 1986 * 64  * 2048 =
eraseblock: 1987 * 64  * 2048 =
```

Мы видим, что добавленный нами в листинге (6) битый блок, распознался ядром корректно. Однако, из u-boot видим, что данный блок так же и "выжил", т.е таблица не была перезаписана. Это говорит о неэффективности использования для этих целей данной утилиты, даже не сомтря на задействование опции -N, --noskipbad (don't skip bad blocks). С другой стороны выполнение тут же в u-boot: nand scrub 0x0 0x10000000 возвращает нас к первоначальной таблице.

На основе системных вызовов mtd, мы разработали утилиту, позволяющую протестировать раздел nand, так и какой то конкретный блок. И в случае если стирание блока заканчивается ошибкой, пометить его как битый блок. Но в этом слчае информация на разле/блоке будет уничтожена.

Тут следует заметить что пока, в целях тестирования, утилита повтороно маркирует уже маркированные блоки. Т.е. если u-boot, к примеру, уже добавил 5-блоков

на /dev/mtd4, то после тестирования всего /dev/mtd4, утилита добавит еще 5. Пример.

Listing 9: Протестируем /dev/mtd4

```
1 ./bin/badmark /dev/mtd4
2 Before test:
3 Number of bad blocks: 5
4 Test...
5 After test:
6 Number of bad blocks: 10
```

Т.е мы повтороно промаркировали теже самые битые блоки. Это сделано в целях тестирования. Правильно было бы поставить проверку - не является ли тестируемый блок уже занесенным в таблицу?

Далее, мы занем, что блок 1458 это заводской битый блок, отключенные от шины. Укажем на него утилите.

Listing 10: Указем заводской bad

```
1 ./bin/badmark -b 1458 /dev/mtd4
2 Before test:
3 Number of bad blocks: 10
4 Test...
5 After test:
6 Number of bad blocks: 11
```

А теперь, укажем на хороший блок

Listing 11: На хороший блок

```
1 ./bin/badmark -b 1457 /dev/mtd4
2 Before test:
3 Number of bad blocks: 11
4 Test...
5 After test:
6 Number of bad blocks: 11
```

Как видим, ничего не изменилось. Добавленные блоки будут видны и после перезагрузки из u-boot. Мы можем указать любой блок, и если его стирание завершиться ошибкой, то утилита промаркирует его как bad блок.

Тут, при сканировании всего раздела (markbad /dev/mtd4) мы снова сталкиваемся с проблемой стирания информации. Но зато, мы имеем возможность маркировать отдельный блок.

Тут еще важно то, что для того что бы системный вызов ioctl(fd, MEMSETBADBLOCK, &some_bb) промаркировал блок, необходимо, что бы перед этим системный вызов ioctl(fd, MEMERASE64, &ei64) дал ошибку, иначе блок не маркируется.

Но если мы хотим маркировать хороший блок(в целях тестирования), например блок номер 10, то естественно MEMERASE64 не даст ошибки. В этом случае, мы ищем заводской bad, и указываем его в опции -b, а хороший блок(который мы хотим маркировать) указываем в опции -g. Пример

Listing 12: На тестирование заводской bad, на маркирование рабочий блок."

```
./bin/badmark -g 10 -b 1458 /dev/mtd4
```

В этом случае, на тестирование подается заводской bad, стирание дает ошибку, а следом идет вызов MEMSETBADBLOCK для рабочего блока номер 10, и счетчик битых блоков увеличивается.

Если же вызвать

Listing 13: На тестирование рабочий блок и на маркирование рабочий блок."

```
./bin/badmark -g 10 -b 10 /dev/mtd4
```

т.е на тестирование подать тоже хороший блок, то счетчик битых блоков останется на прежнем значении.

В нашем распоряжении также опция -t, которая применяет системыный вызов ioctl(fd, MEMGETBADBLOCK, &test_ofs), ко всем блокам подряд.

Listing 14: Проверка, bad не bad"

```
./bin/badmark -t /dev/mtd4 > test.txt

cat ./test.txt
...
...
Good block at 0x0013e800

Good block at 0x0013f000

Good block at 0x0013f800

Bad block at 0x00140000

Bad block at 0x00140800

Bad block at 0x00141000
...
...
Bad block at 0x0015e800

Bad block at 0x0015f000

```

```
23 Bad block at 0x0015f800
24
25 Good block at 0x00160000
26
27 Good block at 0x00160800
28
29 Good block at 0x00161000
30 ...
31 ...
```

Т.е мы видим, что только что маркированный 10-й блок, распознался подсистемой mtd как Bad Block. Распознал этот блок как битый так же и u-boot после перезагрузки. Таким образом, теперь мы имеем возможность маркировать битые блоки "не отходя от кассы", под ядром. Так же было проверено еще несколько маркирований, все совпадает(u-boot - ядро).