Application Note

08 10, 2010

NAND Flash Bad Block Management
---For Linux BSP

ABSTRACT:

This doc describe how to handle nand flash bad block with linux software

KEYWORDS: NAND BAD BLOCK BBI BBT

APPROVED:

AUTHOR	COMMENTS
Jason Liu	Initial draft

NAND Flash Bad Block Management
---For Linux BSP

1. What is bad block?

Nand flash will have some invalid blocks which is what we called bad blocks. This invalid bad block can't be used to store data because it's not stable. Software should avoid using these invalid blocks by means of checking the bad block first. If it's bad, skip it and not use it.

2. How to check bad block?

Nand flash manufactory will mark the bad block with one flag in the spare area of nand flash out of factory. This bad block is what we called initial bad block. This flag is what we called bad block indication (BBI). The location of BBI is defined by the NAND flash manufacturer. Usually, the BBI will locate in the spare area as the following table shown.

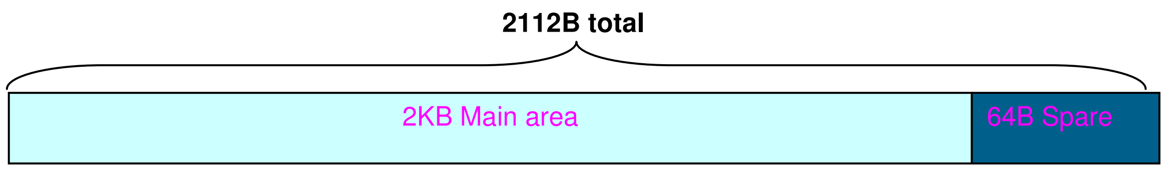
Take 8bit NAND as example:

Nand Flash	SLC(small page)	SLC(large page)	MLC
BBI offset in page	5th byte [1]	1th byte [1]	1th byte [1]
Page offset in block	first page	first page	First or first 2 pages or Last page or last 2 pages [2]

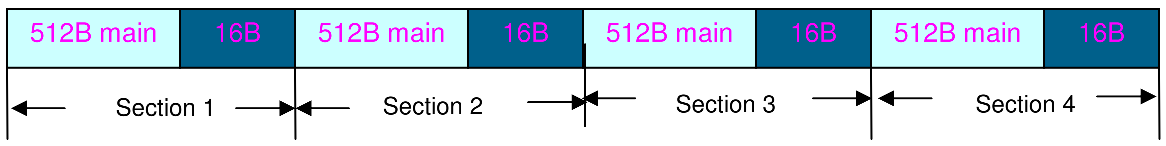
Note1: 0xFF means good block, others means bad block
Note2: The page which contains the bad block marker differs between each vendors. Refer to the NAND spec for detailed information about which pages contain the bad block marker.

3. Incompatibility

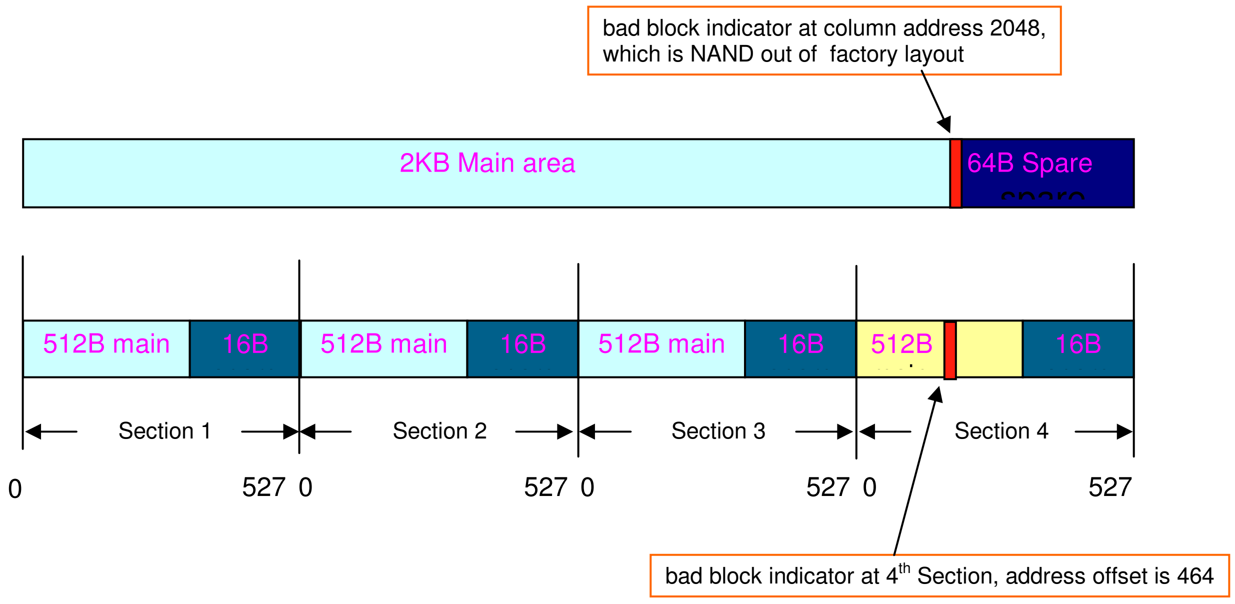
Nand flash has the initial data lout which is main + spare area when it come out of factory. Take 2KB + 64B MLC nand flash as example, the layout of nand flash is:



But the FSL IMX NFC layout is as the followings,



So, the BBI of NAND flash out of factory is located in the main area of sections 4, as the following shows,


bad block indicator at column address 2048, which is NAND out of factory layout
2KB Main area
64B Spare
512B main
16B
512B main
16B
512B main
16B
512B
16B
Section 1
Section 2
Section 3
Section 4
0
527
0
527
0
527
0
527
bad block indicator at 4th Section, address offset is 464

Here is the summery of the incompatibility between the NAND flash layout and the FSL NFC data layout:

- BBI of NAND flash out of factory is located in the data area of the last section of NFC
- BBI byte of NFC layout is not correct with large page NAND flash, it only compatible with small page NAND flash

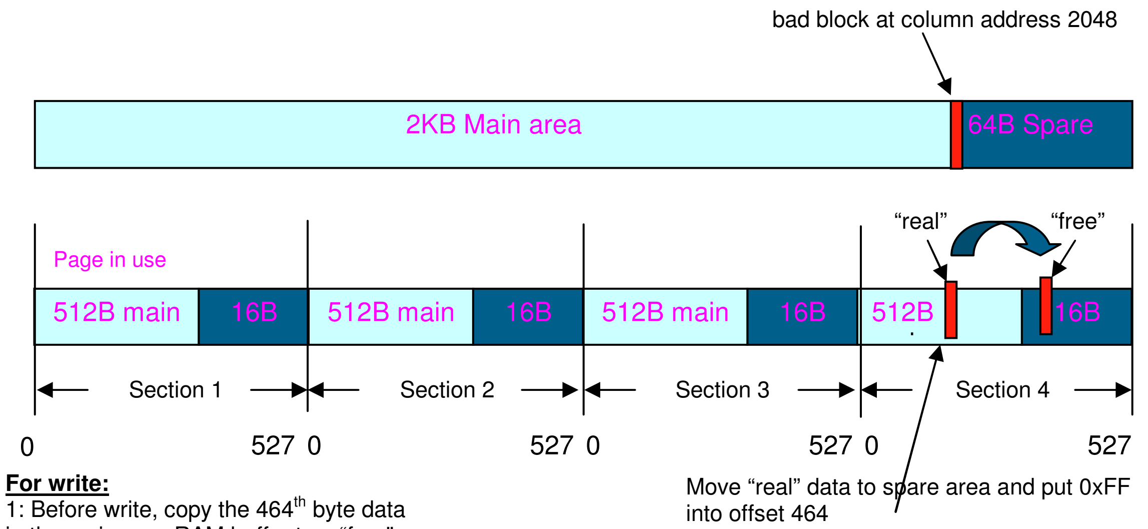
In order to solve the incompatibility above, we need take one solution to preserve the BBI out of factory not be overriden by user data, the solution is called BBI swap.

4. BBI swap solution internals

From the above graph, the user data will overwrite with BBI out of factory during the use of NAND flash and make the BBI lost. So, we need figure out one solution to record the BBI of factory when the NAND flash is used for the first time with FSL NAND flash driver. The solution is as followings:

Solution: Take 2K + 64B MLC NAND as example

1) Swap the byte in main area which denotes the BBI with one byte in the spare area during program, and let the BBI byte is 0xFF. The user data is now swapped to the spare area.
2) Swap back the user data with the spare area which is used to store the user data during read.
3) the software always check the swapped byte of spare area to get the BBI information and build the bad block table for use.



For write:
1: Before write, copy the 464th byte data in the main area RAM buffer to a "free" spare-area RAM buffer.
2. Write 0xFF to the 464th byte in the main area RAM.
3. Start "write" operation.

For read:
After reading data out of NAND flash, copy the data from the "free" byte in the spare-area RAM to the 464th byte in the main area RAM buffer to "recover"

For bad block detection:
After reading data out of NAND flash, check the "free" byte

Free byte offset:
The second byte of last spare section

5. Why Linux BSP need BBT solution?

BBT means bad block table which will be stored onto NAND flash.Without BBT, NAND driver will scan all the blocks on NAND flash to get the bad block information . But with BBT, NAND driver can just fetch the bad block information from bad blcok table which has been stored onto NAND. We need BBT solution due to the following reasons:

- ➢ If run-time error occurs during write/erase, how to mark that block as bad again? We may can't write to that block now
- ➢ Performance – without BBTgiven large size of NAND, the scanning time will become pretty significant.

6. Why we have BBT solution, We still Need BBI swap?

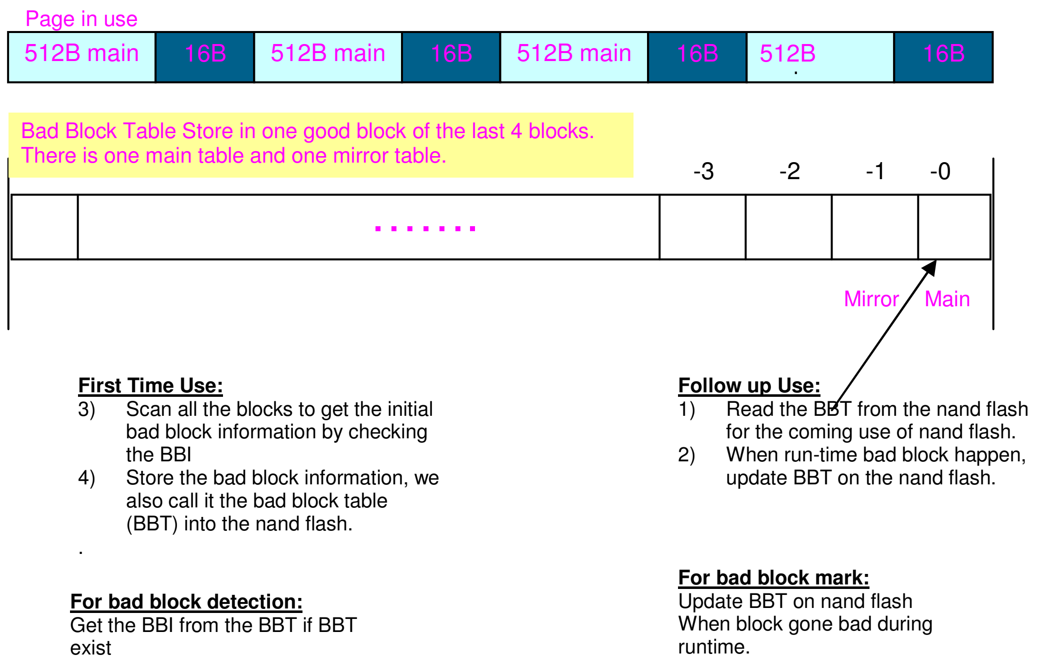
Preserve the bad block indicator for NAND out of factory by using BBI swap will make it possible for NAND driver to reconstruct the accurate BBT talbe once the bad block table is been erased by other tools or bad block table gone corrupt or the bad blcok table not exist.

7. NAND bad block management of Linux BSP

The following solution is the NAND bad block management applied into the Linux BSP:

Solution:(BBI swap + BBT)

1) Scan all the blocks to get the initial bad block information by checking the BBI for a fresh NAND. BBI swap will be used to get the real BBI information from the NAND out of factory.
2) Store the bad block information into the NAND flash . Thus all the bad block information can be fetched from NAND during the following use. The bad block information store on the NAND is what we called the BBT(Bad Block Table).
3) When run-time bad block happen, update BBT on the NAND flash.



Bad Block Table Store in one good block of the last 4 blocks.
There is one main table and one mirror table.

First Time Use:

3) Scan all the blocks to get the initial bad block information by checking the BBI
4) Store the bad block information, we also call it the bad block table (BBT) into the nand flash.

Follow up Use:

1) Read the BBT from the nand flash for the coming use of nand flash.
2) When run-time bad block happen, update BBT on the nand flash.

For bad block detection:
Get the BBI from the BBT if BBT exist

For bad block mark:
Update BBT on nand flash When block gone bad during runtime.

8. NAND driver upgrade notes

We add the BBI swap solution in the Linux BSP 10.05 release which will cause some incompatible issue when we try to upgrade Linux BSP release to 10.05 or higher version.

Issues:

The NAND data which written by old NAND driver will become corrupt when using the new NAND driver. This is due to that one byte of user data written by old driver is swapped to spare area and is replaced with 0xFF by new driver.

What need to do:

If the NAND flash contains the valid user data, please do the following,

1. Use old NAND driver to back up all the user files.
2. Use old NAND driver to erase all the NAND partitions
3. Use new NAND driver to copy back all the user files.

Note: If don't want to do back/restore, just take step 2

As for the BBT table which created by old NAND driver. Currently, the BBT can't be erased by NAND driver, which means the BBT table, will remain there. The new driver can recognize the BBT table and re-use it. But due to the BI swap, the bad block information appears on the BBI offset will be swapped with 0xFF, which lead to any bad block appears on this range will be taken as good block. This is very very rarely case and if it's unfortunately happen, the only we can do is to erase the bad block table with uboot force erase function.

Recommendation:

Highly recommend using the BBI swap + BBT solutions for NAND flash bad block management during mass production.

9.NAND driver internals

Please refer to drivers/mtd/nand/mxc_nd2.c & drivers/mtd/nand/mxc_nd2.h file under Linux BSP source code package.