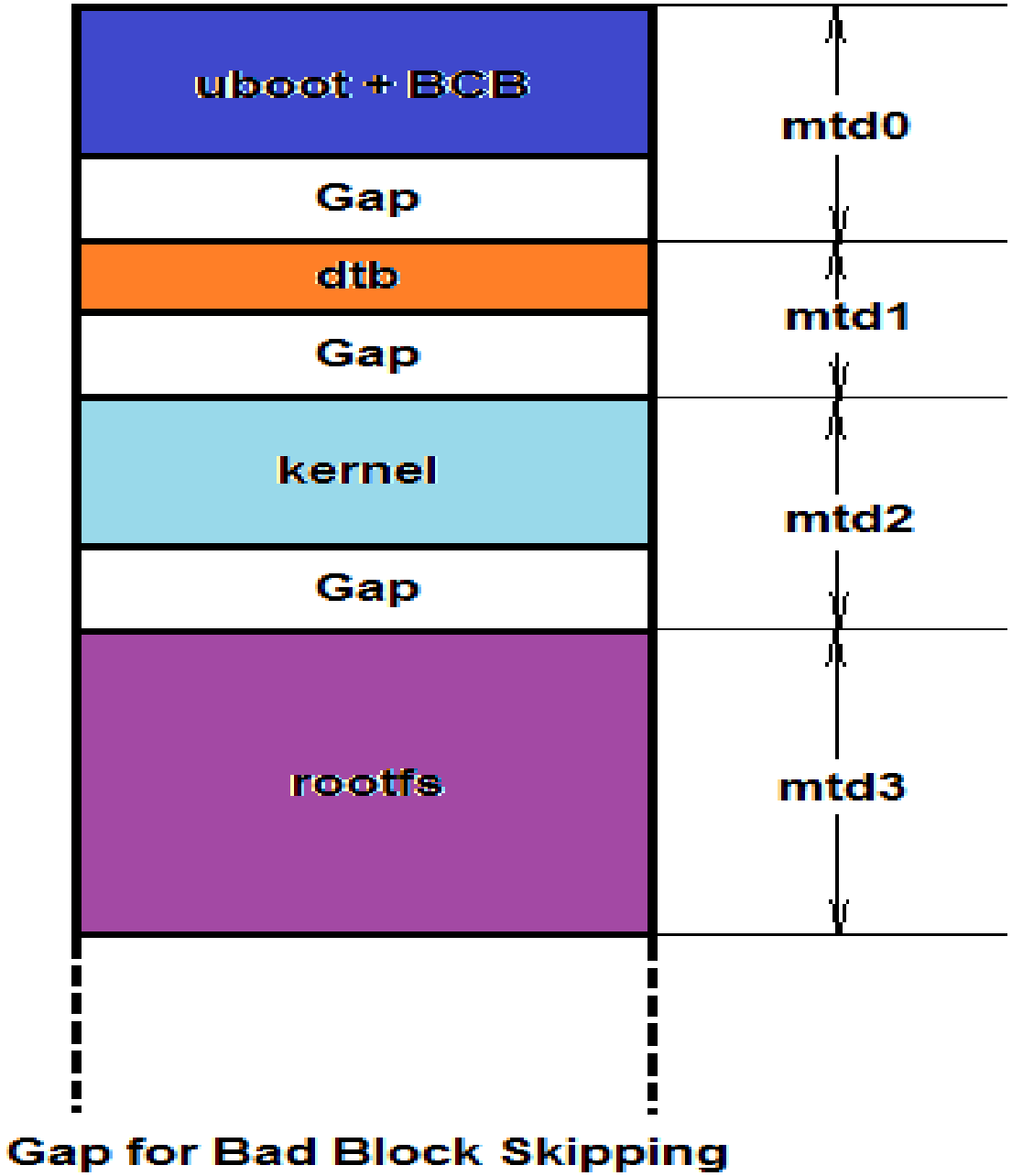
Mass Production with NAND Programmer

Biyong SUN

11, JULY 2017

NAND Layout


uboot + BCB
Gap
dtb
Gap
kernel
Gap
rootfs
mtd0
mtd1
mtd2
mtd3
Gap for Bad Block Skipping

Method

1. Use no bad block NAND flash as master chip
2. Dump each mtd with image size
3. Burning each mtd mirror image from the mtd start address by programmer

Question:

1. How to handle different DBBT(Discovered Bad Block Table) in BCB(Boot Control Blocks) for each single chip
2. Why need multiple mirror images from each mtd part instead of whole NAND flash dump.

Handle DBBT(Discovered Bad Block Table)

Issue:

- kobs-ng is a tool to create the DBBT at runtime
- Each one of single NAND has different DBBT
- NAND mirror images could not use a particular DBBT from a master chip.

Solution:

- If the DBBTSearchAreaStartAddress in FCB(Firmware Configuration Block) is “0”, ROM code will start search Bad Block Table instead of reading DBBT created by kobs-ng
- Modify the kobs-ng to fix the DBBTSearchAreaStartAddress “0”.

Flash Control Block Structure

Firmware2_startingPage	108	4
PagesInFirmware1	112	4
PagesInFirmware2	116	4
DBBTSearchAreaStartAddress	120	4
BadBlockMarkerByte	124	4

```
struct fcb_block {
    FCB_ROM_NAND_Timing_t  m_NANDTiming;          //!< Optimum timing parameters for Tas, Tds,
    uint32_t               m_u32PageDataSize;     //!< 2048 for 2K pages, 4096 for 4K pages.
    uint32_t               m_u32TotalPageSize;    //!< 2112 for 2K pages, 4314 for 4K pages.
    uint32_t               m_u32SectorsPerBlock;  //!< Number of 2K sections per block.
    uint32_t               m_u32NumberOfNANDs;    //!< Total Number of NANDs - not used by ROM.

    uint32_t               m_u32DBBTSearchAreaStartAddress;//!< Page address where dbbt search area begins
    uint32_t               m_u32BadBlockMarkerByte;
                                  //!< bw swapped with metadata[0] to complete page data.
```

kobs-ng Modify Example

Different version of kobs-ng could be a tiny difference of the modification for “**m_u32DBBTSearchAreaStartAddress**”

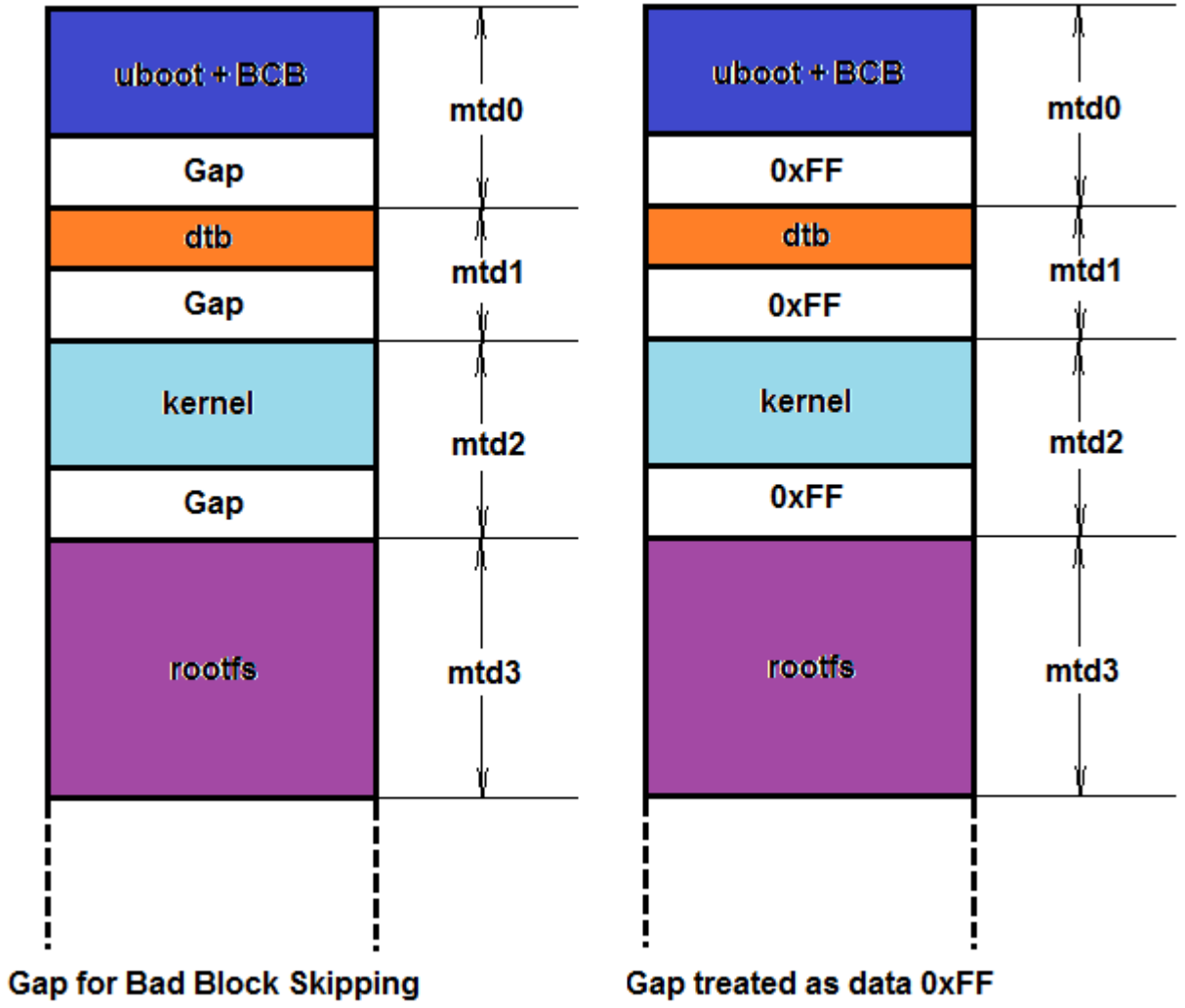
src/mtd.c

```
static int fill_fcb(struct mtd_data *md, FILE *fp)
{
.....
.....
.....
                b->m_u32PagesInFirmware1      = boot_stream_size_in_pages;
                b->m_u32PagesInFirmware2      = boot_stream_size_in_pages;

                //b->m_u32DBBTSearchAreaStartAddress = cfg->search_area_size_in_pages;
                b->m_u32DBBTSearchAreaStartAddress = 0;
                b->m_u32BadBlockMarkerByte     = geo->block_mark_byte_offset;
                b->m_u32BadBlockMarkerStartBit = geo->block_mark_bit_offset;


int v2_rom_mtd_init(struct mtd_data *md, FILE *fp)
{
.....
.....
.....

                fcb->FCB_Block.m_u32PagesInFirmware1       = boot_stream_size_in_pages;
                fcb->FCB_Block.m_u32PagesInFirmware2       = boot_stream_size_in_pages;
                //fcb->FCB_Block.m_u32DBBTSearchAreaStartAddress = search_area_size_in_pages;
                fcb->FCB_Block.m_u32DBBTSearchAreaStartAddress = 0;
```

Why need multiple mtd mirror images

The gap between will be treated as data when dumping the entire flash.

In this case, it will be a problem for bad block skipping.

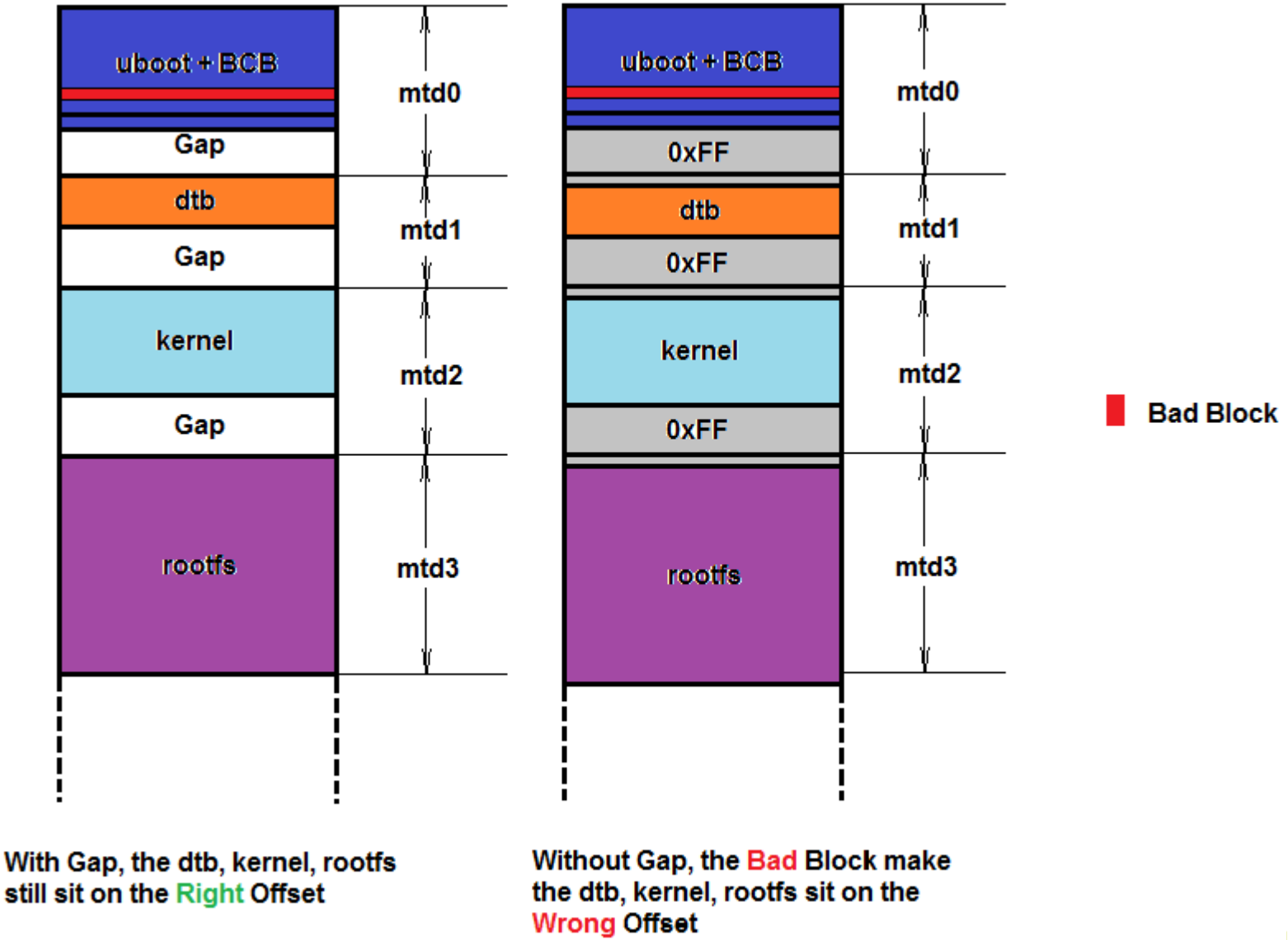


Why need multiple mtd mirror images(Cont.)

The Gap between each mtd gives a chance to keep each image still sit on the right offset.

But if use the whole flash mirror, there is no gap for bad block skipping. Each offset after bad block will be moved.

In this case, the uboot got the wrong dtb and kernel with the wrong offset



How do dump images

- NAND programmer to read/dump

Benefit

- No Algorithm source code needed
- Customer no need to spend money for buying a customized programming software from programmer vendor

SECURE CONNECTIONS
FOR A SMARTER WORLD