Evaluation of Flash Filesystems Update

SFTL and BENAND

2012/9/20
Toshiba Corporation
UWATOKO Katsuki

Copyright 2012, Toshiba Corporation.
Contents

- SFTL (Simple Flash Translation Layer) NAND Driver
- BENAND (Built-In ECC NAND)
- fastmap of UBI
Background

- **Boot Sequence of our target.**
  - load the boot image from NAND to RAM
  - mount the image on RAM with SQUASHFS
    - The reasons are:
      - it is faster boot than direct mount in our target.
      - it reduces the number of NAND read times and make it predictable for reliability.
  - mount some parts of the boot image directly.

![Diagram of NAND and boot image areas]

- **The Target in this presentation**
  - Boot Image area almost read-only
  - read/write data area
  - loaded to RAM and mounted with SquashFS
  - mounted directly
  - mounted with ubi/ubifs

---

*TOSHIBA*  
Leading Innovation
Background

- Requirements of NAND Driver for our target
  - Quick initialization for fast boot
  - Bad Block Management
  - wear-leveling and scrubbing
    - but GOOD wear-leveling is not necessary, because it is read-only except for update and the frequency of read is low.
  - Block Device interface for a direct mount

- Other Drivers and status (when we started to develop SFTL)
  - mtdblock
    - has no bad block management.
  - sm_ftl/ssfdc
    - These are for SmartMedia™. Therefore there are some limitations of Media Size, Zone wear-leveling etc.
  - ubi + (ubifs or gluebi or ubiblk)
    - The initialization time of ubi was not match for our target because it was slow. But now it provides “fastmap” which reduces the time. We are considering using this.
Features of SFTL

- SFTL (Simple Flash Translation Layer)
- Block Device Interface (using MTD block dev in MTD)
- Provides wear-leveling and Scrubbing
  - no static wear-leveling
- One erase block size cache
  - Sequential access is fast, even if a size of request is not Block size.
- uses 6 bytes in OOB for a logical address, status, version.
- Erases a erase block just before a write
  - for maintenance
  - easy to analyze after boot issue.
- We sent the patch to Linux MTD ML at Dec. 2012.
  The maintainer suggested that:
    Using OOB for status/data, not only for bad block status, is a bad idea nowadays. we have to re-design this.
Software Structure

Targets for a performance measurement in this presentation

- **Linux Kernel**
  - SFTL
  - ubiblk
  - gluebi
  - UBI (Wear-Leveling)

- **Block Device IF**
  - MTD block dev
  - UBIIFS

- **MTD (Memory Technology Device)**

- **HW**
  - NAND Controller in SoC

- **NAND Chip**
BENAND (Built-In ECC NAND)

Raw NAND Chip

- SFTL/UBI (Wear-leveling)
- MTD (Memory Technology Device)
- Read/Write/Erase
- ECC Control
- SW ECC
- SoC NANDC Driver
- ECC Cntrl

Common NAND Interface

BENAND™

- Bad Block Management
- Wear leveling
- Logical-Physical address change
- Garbage collection
- ECC

Common NAND Interface

HW Block

SW Block

MTD (Memory Technology Device)

- BENAND ECC
- SW ECC

SoC NANDC Driver

ECC Cntrl
BENAND (Built-In ECC NAND)

- Easy to port BENAND support to MTD NAND
  - add ECC layout for BENAND
  - add a check routine of ECC status after read.

- We have a plan to send the patch to Linux MTD ML.

```
drivers/mtd/nand/nand_base.c | 73 ++++++++++++++++++++++++++++++++++++++++++++++++++++++++++
include/linux/mtd/nand.h     |  3 +
2 files changed, 75 insertions(+), 1 deletion(-)
```
Environment

- **CPU**: Cortex-A9 528Mhz, SMP (3 core)
- **NAND Controller**: w/ HW ECC, w/o DMA
- **Kernel**: Linux 3.0.32-ltsi
  - ubiblk v0.9 9/26/2011
  - fastmap RFCv5 5/17/2012
  - sftp 12/14/2011
- **mtd-utils**: 1.3.1
- **gcc**: 4.5.1
- **NAND**

<table>
<thead>
<tr>
<th></th>
<th>BENAND (TC58BVG2S0FTA00)</th>
<th>NORMAL NAND (TC58NVG1S3ETA00)</th>
</tr>
</thead>
<tbody>
<tr>
<td>Media Size</td>
<td>512MB</td>
<td>256MB</td>
</tr>
<tr>
<td>Block Size</td>
<td>256KB</td>
<td>128KB</td>
</tr>
<tr>
<td>Page Size</td>
<td>4096B + 128B</td>
<td>2048B + 64B</td>
</tr>
</tbody>
</table>
Basic NAND Performance (Erase/Read/Write)

The erase/write performance of BENAND is better than NORMAL. The result is considered that it is because of the difference of BLOCK size.

Erase (flash_erase command)

<table>
<thead>
<tr>
<th></th>
<th>BENAND (256MB)</th>
<th>NORMAL NAND (256MB)</th>
</tr>
</thead>
<tbody>
<tr>
<td>epmty</td>
<td>2.51</td>
<td>3.96</td>
</tr>
<tr>
<td>full</td>
<td>2.52</td>
<td>5.34</td>
</tr>
</tbody>
</table>

Read (dd from /dev/mtd to /dev/null)

<table>
<thead>
<tr>
<th></th>
<th>BENAND bs=256k</th>
<th>NORMAL NAND bs=128k</th>
</tr>
</thead>
<tbody>
<tr>
<td>MB/sec</td>
<td>7.51</td>
<td>7.54</td>
</tr>
</tbody>
</table>

Write (dd from /dev/zero to /dev/mtd)

<table>
<thead>
<tr>
<th></th>
<th>BENAND bs=256k</th>
<th>NORMAL NAND bs=128k</th>
</tr>
</thead>
<tbody>
<tr>
<td>MB/sec</td>
<td>3.15</td>
<td>2.73</td>
</tr>
</tbody>
</table>
Performance on our target’s boot sequence

The following graph is the result of our target’s boot sequence example.

- 128MB MTD Partition.
- read 64MB compressed file (squashfs image) to RAM.
options: unit size 4KB (-b), one thread (-t), writing synchronously (-S) drop cache between each tests (modified the source) ubifs mount with compr="none" option to avoid an influence of contents.

The performance of sequential read/write of SFTL are good. But the performance random is not good, especially random read. Because SFTL has just one erase block buffer for read/write.
These are just for reference data, because these depend on parameters of reserved block number for bad block, etc. But in the case of a small partition, especially using a large page NAND, we have to be careful the size.
ubiattach time - fastmap vs !fastmap -

fastmap depends on the size of partition
Conclusion

- **SFTL**
  - SFTL has good performance to our target so far. But we have to reconsider the implement to merge Linux main line.

- **BENAND**
  - It’s easy to port BENAND to MTD NAND.
  - The performance is almost same as NORMAL NAND. In this presentation, the result is assumed to depend on the ERASE Block size, not BENAND.

- **UBI fastmap**
  - Ubiattach time is very fast. We are considering to adopt this, and keep on evaluating it.