SSD Firmware Implementation Project
- Lab. #5-

Sang-Phil Lim (Isfeel0204@gmail.com)
SKKU VLDB Lab.

2011-05-12
## Lab. Time Schedule

<table>
<thead>
<tr>
<th>Lab.</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>#1</td>
<td>FTL Simulator Development Guide</td>
</tr>
<tr>
<td>#2</td>
<td>FTL Simulation Guide</td>
</tr>
<tr>
<td>#3</td>
<td>Project 1 Presentation</td>
</tr>
<tr>
<td>#4</td>
<td>Jasmine OpenSSD platform tutorial #1</td>
</tr>
<tr>
<td><strong>#5</strong></td>
<td><strong>Jasmine OpenSSD platform tutorial #2</strong></td>
</tr>
<tr>
<td>#6</td>
<td>FTL Porting Guide</td>
</tr>
<tr>
<td>#7</td>
<td>Firmware Debugging Guide</td>
</tr>
<tr>
<td>#8</td>
<td>SSD Performance Evaluation Guide</td>
</tr>
<tr>
<td>#9</td>
<td>Project 2 Presentation</td>
</tr>
</tbody>
</table>
Jasmine SSD Firmware

- **Layers**
  - HIL (Host Interface Layer), FTL, FIL (Flash Interface Layer)

- **Sort of FTLs**
  - **Tutorial FTL**
    - Page-mapping FTL, but no garbage collection
  - **Greedy FTL**
    - Page-mapping FTL with simple garbage collection
    - Support POR (*to appear*)
  - **Dummy FTL**
    - Not a real FTL (*Not access NAND flash at all*)
    - For measuring SATA and DRAM speed
Dummy FTL

• Overview
  – Literally, Dummy FTL is not a real FTL
  – Not access NAND flash at all
  – To measure only SATA & memory speed
DRAM ECC & Memory Utility

• ECC (Error Correcting Code) engine for DRAM
  – In storage device, ECC is a mandatory to support high reliability for memory data
  – Four byte ECC parity is added to every 128 bytes of data

• Memory Utility
  – Data transfer between DRAM and SRAM
    • Also SRAM-to-SRAM, DRAM-to-DRAM
  – Recommendation
    • Access (modify) DRAM data via Memory Utility hardware
Memory Utility (contd.)

• Internal operation (mem_copy)
SATA Host Buffer

- Buffer management and flow control is done by hardware (BM)
  - SATA write pointer does not run ahead of BM write limit
  - SATA read pointer does not run ahead of BM read limit
void ftl_read(UINT32 const lba, UINT32 const total_sectors)
{
    UINT32 num_sectors_to_read;
    UINT32 lpage_addr = lba / SECTORS_PER_PAGE;    // logical page address
    UINT32 sect_offset = lba % SECTORS_PER_PAGE;    // sector offset within the page
    UINT32 sectors_remain = total_sectors;

    while (sectors_remain != 0)    { // one page per iteration
        if (sect_offset + sectors_remain < SECTORS_PER_PAGE)
            num_sectors_to_read = sectors_remain;
        else
            num_sectors_to_read = SECTORS_PER_PAGE - sect_offset;
        // read data from nand
        UINT32 next_read_buf_id = (g_ftl_read_buf_id + 1) % NUM_RD BUFFERS;

        // wait if the read buffer is full (slow host)
        while (next_read_buf_id == GETREG(SATA_RBUF_PTR));

        SETREG(BM_STACK_RDSET, next_read_buf_id); // change bm_read_limit
        SETREG(BM_STACK_RESET, 0x02);             // change bm_read_limit

        g_ftl_read_buf_id = next_read_buf_id; // increase FTL buffer pointer

        sect_offset = 0;
        sectors_remain -= num_sectors_to_read;
        lpage_addr++;
    }
}
Dummy FTL Internal: Read Operation

Read Buffer is full...

Can’t use this buffer frame for next read op. (slow SATA)

Now Increase FTL buffer ptr.

Increase BM read limit

Wait for SATA ← DRAM

Complete send data to host

SATA_RBUF_PTR

BM_STACK_RDSET

g_ftl_read_buf_id
void ftl_write(UINT32 const lba, UINT32 const total_sectors) {
    UINT32 num_sectors_to_write;
    UINT32 sect_offset = lba % SECTORS_PER_PAGE;
    UINT32 remain_sectors = total_sectors;
    while (remain_sectors != 0) {
        if (sect_offset + remain_sectors >= SECTORS_PER_PAGE) {
            num_sectors_to_write = SECTORS_PER_PAGE - sect_offset;
        } else {
            num_sectors_to_write = remain_sectors;
        }
        // bm_write_limit should not outpace SATA_WBUF_PTR
        while (g_ftl_write_buf_id == GETREG(SATA_WBUF_PTR));
        g_ftl_write_buf_id = (g_ftl_write_buf_id + 1) % NUM_WR_BUFFERS;
        SETREG(BM_STACK_WRSET, g_ftl_write_buf_id); // change bm_write_limit
        SETREG(BM_STACK_RESET, 0x01); // change bm_write_limit
        sect_offset = 0;
        remain_sectors -= num_sectors_to_write;
    }
}
Dummy FTL Internal: Write Operation

- **SATA_WBUF_PTR**: Increase FTL buffer pointer.
- **BM_STACK_WRSET**: Increase BM write limit.
- **g_ftl_write_buf_id**: Complete buffering data from host.
- **Wait for SATA → DRAM**: Start buffering data from SATA.
- **Increase BM write limit**: Increase buffer pointer when buffer is full.

**Event Q**: W,4,4

**SATA Write Buffer**: SATA → DRAM
NAND Flash Configuration

- Four channels
- Eight banks in each channel
NAND Flash Configuration (contd.)

- Bank
  - 16 Bit IO bus
  - High/Low NAND chips
- Virtual block/page
  - In 2-plane mode,
    Physical page size x 4
    Physical block size x 4
NAND Flash Configuration (contd.)

• The Banks share the same IO bus
• However, cell operation can occur in parallel
• Barefoot has only four R/B signal inputs (each channel) from banks, max 4 way interleaving is possible
NAND Flash Configuration (contd.)

- Interleaved cell operation (2-plane mode)
  - 33 44 55 66 ........ 77 88 99 AA ........
Ref) Two-plane Page Program

- Timing diagram

80h: data input command of 1st plane
11h: data-loading of 1st plane
$T_{DBSY}$: short period of time (dummy busy)
81h: data input command of 2nd plane
10h: actual page program
70h: read status command
$T_{PROG}$: cell operation time

Reference: Samsung MLC NAND data sheet
NAND Flash Controller

Flash Command Port
- CMD
- BANK
- OPTION
- DMA_ADDR
- DMA_CNT
- COLUMN
- ROW_0_L
- ROW_0_H
- ...
- ROW_31_L
- ROW_31_H
- DST_COL
- DST_ROW
- ISSUE

Waiting Room
- CMD
- BANK
- OPTION
- ...

Bank Status Ports
- CMD
- OPTION
- ...

Issue → Accept → Abort

c.f.)
Auto-select command?

WR_BANK register (read only)

How can we figure out whether WR is empty?

WR_STAT register (read only)
NAND Flash Controller (contd.)

```c
void flash_issue_cmd(UINT32 const bank, UINT32 const sync)
{
    UINT32 rbank = REAL_BANK(bank);
    SETREG(FCP_BANK, rbank);

    // You should not issue a new command when Waiting Room is not empty.
    while ((GETREG(WR_STAT) & 0x00000001) != 0);

    // If you write any value to FCP_ISSUE,
    // FCP register contents are copied to Waiting Room.
    SETREG(FCP_ISSUE, NULL);

    if (sync == RETURN_ON_ISSUE)
        return;

    // wait until the new command is accepted by the target bank
    while ((GETREG(WR_STAT) & 0x00000001) != 0);

    if (sync == RETURN_ON_ACCEPT)
        return;

    // wait until the target bank finishes the command
    while (_BSP_FSM(rbank) != BANK_IDLE);
}
```
Any Questions?