Developing FTLs on the Jasmine OpenSSD Platform

Sang-Phil Lim (lsfeel0204@gmail.com)
SKKU VLDB Lab.

2011-05-11
Outline

- Getting Started
- Building & Installing Firmware
- Greedy FTL Implementation & Development Guide
- Debugging Guide
Getting Started
Development Setup: Hardware

- Hardware Requirement
  - For debugging (optional)
Development Setup: Hardware
Development Setup: Software

• Jasmine Firmware
  – Latest version: v1.0.3
  – Available from
• RVDS(RealView Development Suite) 3.0 ↑ or Code Sourcery G++ Lite Edition for ARM (free)
• MS Visual Studio Express Free Edition 2010 (free)
Development Setup: Software

• **Installing toolchain**
  – RVDS or Code Sourcery G++ Lite Edition for ARM
    • To build the firmware binary file (`firmware.bin`)
  – MS Visual Studio Express Free Edition 2010
    • To build the firmware installer (`install.exe`)

• **Serial communication**
  – Hyperterminal (BAUD_115200/8/N/1/X)
  – Configure on-board switches (SW 2,3,4) (please refer to Jasmine board schematics)
Building & Installing Firmware
Compile & Build Firmware

• Setting compile options (`./include/jasmine.h`)

```c
OPTION_2_PLANE
OPTION_ENABLE_ASSERT
OPTION_FTL_TEST
OPTION_UART_DEBUG
OPTION_SLOW_SATA
OPTION_SUPPORT_NCQ
OPTION_REduced_CAPACITY
```
Compile & Build Firmware

• Build the firmware

  > cd ./build_gnu
  > build.bat

• Compile the installer
  – Open ./installer/installer.sln & Build
  – Move ./installer/install.exe to ./build_gnu
Install Firmware to the Jasmine Board

- Booting the Jasmine board as ‘Factory mode’

- Install firmware

```
> ./build_gnu/install.exe
```
Install Firmware to the Jasmine Board

- Install for the first time
  1 – 2 – 6 – 3

- Reinstall
  1 – 2 – 3
Run Firmware

• Booting the Jasmine board as ‘Normal mode’
  – Unplug SATA cable
    • Jasmine would be busy doing internal low-level format
  – Plug SATA cable when LED at D4 position is lit

• Now Jasmine is ready to process SATA commands

• Try to send IO requests to Jasmine board! 😊
Greedy FTL Implementation & FTL Development Guide
# FTL Implementation

<table>
<thead>
<tr>
<th>Source file</th>
<th>Function</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>./installer/installer.c</td>
<td>ftl_install_mapping_table</td>
<td>FTL 초기 메타데이터를 NAND 플래시에 기록하는 연산 수행. 필웨어 설치 시 함께 호출됨</td>
</tr>
</tbody>
</table>
| ./ftl_[scheme]/ftl.c     | ftl_open | FTL 초기화 과정 수행  
- NAND 플래시로부터 메타데이터 로드 및 초기화  
- VBLK #0 에 포맷 마크가 기록되어 있지 않을 경우, format 함수 호출  |
|                         | format | VBLK #0 와 FTL 메타 영역을 제외한 나머지 블록들을 삭제  
- 포맷 마크 기록  |
|                         | ftl_read | 사용자 데이터 읽기 처리  |
|                         | ftl_write | 사용자 데이터 쓰기 처리  |
|                         | ftl_flush | SATA idle/standby time 에 주기적으로 메타데이터를 flush 하는 연산 수행  |
Sample FTLs on the Jasmine Platform

- **Tutorial FTL** (developed by INDILINX)
  - Page-mapping FTL, but no garbage collection

- **Greedy FTL** (developed by SKKU VLDB Lab.)
  - Page-mapping FTL with simple garbage collection
  - Support Power-Off Recovery *(to appear)*

- **Dummy FTL**
  - Not a real FTL *(Not access NAND flash at all)*
  - For measuring SATA and DRAM speed
void ftl_open(void)
{
    // check DRAM footprint
    sanity_check();
    // build bitmap of bad blocks
    build_bad_blk_list();

    if (check_format_mark() == FALSE) {
        format();
    }
    // load FTL metadata
    else {
        // load FTL metadata in SRAM/DRAM from nand
        load_metadata();
    }
    // init FTL & SATA buffer pointer
    g_ftl_read_buf_id = 0;
    g_ftl_write_buf_id = 0;
}
Greedy FTL: FTL Metadata

• **DRAM metadata**
  - Page-level mapping table (**PAGE_MAP**)
    - For Logical-to-Physical page address mapping
  - Bad block bitmap table (**BAD_BLK_BMP**)
  - Block valid count information (**VCOUNT**)
    - For victim block selection

• **SRAM metadata**
  - Misc. information
    - Page index pointer of meta & user area
    - Remain free block count
    - Etc.
Guide #1 - Memory Utility

- Do not update DRAM data by ARM
Guide #2 – DRAM Buffer Management

• SATA read/write buffer
  – Buffering the data to send to host or the data to program onto NAND
  – Circular FIFO
  – Controlled by SATA/FTL/BM buffer pointers
  – Why do we need a BM buffer pointer?
    • I/O Consistency issue
      – bandwidth gap between SATA and NAND
      – “SATA could send the wrong data to host”
      – “NAND could program the wrong host data”
Guide #2 – DRAM Buffer Management (contd.)

• What if BM buffer pointer is not used?

Still reading data from NAND!
Thus, SATA should not send it to host now.
Guide #2 – DRAM Buffer Management (contd.)

• What if BM buffer pointer is not used?

Still programming to NAND!
Thus, the data should not be overwritten now
Guide #2 – DRAM Buffer Management (contd.)

- SATA read/write buffer
Greedy FTL: Read operation

```c
void ftl_read(UINT32 const lba, UINT32 const num_sectors)
{
    lpn = lba / SECTORS_PER_PAGE;
    sect_offset = lba % SECTORS_PER_PAGE;
    remain_sects = num_sectors;

    while (remain_sects != 0) {
        if ((sect_offset + remain_sects) < SECTORS_PER_PAGE) {
            num_sectors_to_read = remain_sects;
        } else {
            num_sectors_to_read = SECTORS_PER_PAGE - sect_offset;
        }

        bank = get_num_bank(lpn); // virtual page-level striping
        vpn = get_vpn(lpn); // address mapping
    }
...
```
...  
// old data was already written  
if (vpn != NULL) {  
    // send read request to nand flash  
nand_page_ptread_to_host(bank, 
        vpn / PAGES_PER_BLK, 
        vpn % PAGES_PER_BLK, 
        sect_offset, 
        num_sectors_to_read); 
}  
// The host is requesting to read a logical page that has never been written to.  
else {  
    // change Buffer Manager read limit pointer  
    // change FTL SATA read pointer  
}  
sect_offset = 0;  
remain_sects -= num_sectors_to_read;  
lpn++;  
} // end of ftl_read function
Greedy FTL: Read operation (contd.)

- Host
- Event Q
- SATA Read Buffer
- DRAM
- PAGE_MAP
- NAND Flash

1. Increase FTL buffer ptr.
2. Transfer user data to DRAM
Greedy FTL: Read operation (contd.)

- Event Q
- SATA Read Buffer
- DRAM PAGE_MAP
- NAND Flash

<table>
<thead>
<tr>
<th>LPN</th>
<th>VPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

- Host
- Send to host
- Increase BM buffer ptr. (by H/W)
- Increase SATA buffer ptr. (by H/W)

- complete loading user data

- SATA_RBUF_PTR
- BM_STACK_RDSET
- RD_BUF_PTR
Greedy FTL : Write Operation

```c
static void write_page(UINT32 const lpn, UINT32 const sect_offset, UINT32 const num_sectors)
{
    bank       = get_num_bank(lpn);
    page_offset = sect_offset;
    column_cnt  = num_sectors;
    old_vpn     = get_vpn(lpn);                      // address mapping
    new_vpn     = assign_new_write_vpn(bank);        // get free vpage

    // if old data already exist,
    if (old_vpn != NULL) {
        // read `left hole sectors'  [left*][new data][right]
        if (page_offset != 0)
            nand_page_ptread(..., RETURN_ON_ISSUE);
        // read `right hole sectors'  [left][new data][right*]
        if ((page_offset + column_cnt) < SECTORS_PER_PAGE)
            nand_page_ptread(..., RETURN_ON_ISSUE);
        // invalid old page (decrease vcount)
        set_vcount(bank, vblock, get_vcount(bank, vblock) - 1);
    }
    ...
```
Greedy FTL : Write Operation (contd.)

```c
... 
vblock = new_vpn / PAGES_PER_BLK;
page_num = new_vpn % PAGES_PER_BLK;

// write new data (RETURN_ON_ISSUE)
nand_page_ptprogram_from_host(bank,
    vblock,
    page_num,
    page_offset,
    column_cnt);

/* update metadata */
set_lpn(bank, page_num, lpn); // maintain lpn list for GC
set_vpn(lpn, new_vpn);  // address mapping table
// increase block valid page count
set_vcount(bank, vblock, get_vcount(bank, vblock) + 1);

} // end of write_page function (caller: ftl_write)
```
Greedy FTL: Write Operation (contd.)

- SATA Write Buffer
- DRAM → NAND
- Event Q: W,4,4
- PAGE_MAP:
<table>
<thead>
<tr>
<th>LPN</th>
<th>VPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>200</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>
- Find free page & Address mapping
- SATA_WBUF_PTR
- BM_STACK_WRSET
- WR_BUF_PTR
Greedy FTL: Write Operation (contd.)

Event Q

SATA Write Buffer

DRAM → NAND

Complete buffering host data

Increase FTL buffer ptr.

Incrase SATA buffer ptr. to buffer next host data (by H/W)

NAND Flash

DRAM

PAGE_MAP

<table>
<thead>
<tr>
<th>LPN</th>
<th>VPN</th>
</tr>
</thead>
<tbody>
<tr>
<td>0</td>
<td>200</td>
</tr>
<tr>
<td>1</td>
<td></td>
</tr>
</tbody>
</table>

Programming host data to NAND flash
Greedy FTL: Write Operation (contd.)

- Event Q:
  - W,4,4
  - W,8,4
  - W,51,4

- SATA Write Buffer
- DRAM → NAND
- SATA → DRAM

- DRAM
- PAGE_MAP
  - LPN: 0, 1, 2
  - VPN: 200

- NAND Flash

- Increase BM buffer pointer (by H/W)

- Complete Programming!
Guide #3 – Flash Command

• To send I/O command to NAND flash
  ① Programming FCP command manually
    • See tutorial FTL code (.ftl_tutorial/ftl.c)
  ② Use LLD interface (.target_spw/flash_wrapper.c)
    • Easiest way to implement an FTL
    • If you want to make a high performance FTL, the first option is recommended
Greedy FTL: NAND Configuration

Meta area

VBLOCK #0 #1 #2~#31 #n-1

Scan list
Firmware image

VCOUNT + Misc. metadata
PAGE_MAP

Data area

Host data

VPAGE #0

Host data

VPAGE #m-1

LPN list

To determine valid page
Guide #4 - Restriction of Accessing NAND

- No support for accessing ‘spare area’

![Diagram showing physical page with data area and spare area](image)

- Data area (16 Sectors, 8192KB)
- Spare area (448B)

- Sector #1, ECC, Sector #2, ECC, ..., ECC, Sector #n
Greedy FTL: Garbage Collection

- If free blocks run out, a victim block is chosen based on the ‘greedy policy’
  - i.e., select the block containing min. valid pages
Greedy FTL : POR(Power-Off Recovery)

- **Metadata logging**
  - Flush all metadata instantly (by `ftl_flush`)
  - @ SATA ready/idle/standby time

<table>
<thead>
<tr>
<th>Block #0</th>
<th>Scan list</th>
<th>VCOUNT + Misc. metadata</th>
<th>PAGE_MAP</th>
<th>User space</th>
</tr>
</thead>
<tbody>
<tr>
<td>Firmware image</td>
<td>...</td>
<td>VCOUNT + Misc. metadata</td>
<td>PAGE_MAP</td>
<td>User space</td>
</tr>
</tbody>
</table>
Guide #5 – BSP interrupt

• Check **BSP_INTR** register (**ftl_isr**)  

<table>
<thead>
<tr>
<th>FIRQ_CORRECTED</th>
<th>FIRQ_CRC_FAIL</th>
<th>FIRQ_MISMATCH</th>
</tr>
</thead>
<tbody>
<tr>
<td>FIRQ_BADBLK_L</td>
<td>FIRQ_BADBLK_H</td>
<td>FIRQ_ALL_FF</td>
</tr>
<tr>
<td>FIRQ_ECC_FAIL</td>
<td>FIRQ_DATA_CORRUPT</td>
<td></td>
</tr>
</tbody>
</table>

• BSP contains the flash command which was last issued for debugging
Debugging Guide
Ways to Debug Your Code

① Serial communication (UART interface)
② ICE + RVD (JTAG interface)
③ On-board indicator (LED on Position D4)
UART Debugging

- Memory/Register dump message
- You also can measure I/O response time using Timer function (./target_spw/misc.c)
Debugging using ICE + RVD

• Line-by-line debugging

```c
    g_barrier = 0;
    while (g_barrier == 0);
```

To stop the firmware execution

• Check controller registers

Last issued FC (BSP)
Jasmine Technical Document

• You can download them from the OpenSSD Project homepage
  – http://www.openssd-project.org

• Technical Reference Manual
  – Jasmine Board Schematics
  – INDILINX Barefoot™ SSD Controller Specification
  – Jasmine Firmware Architecture
  – Jasmine Firmware Software Specification

• FTL Developer’s Guide
  – FTL Porting Guide
  – Compile, Build & Install Firmware
  – Debugging Tips
Participate in Our OpenSSD Activities!

The OpenSSD Project

The OpenSSD Project is an initiative to promote research and education on the recent SSD (Solid State Drive) technology by providing easy access to OpenSSD platforms on which open source SSD firmware can be developed. Currently, we offer an OpenSSD platform based on the commercially successful Barefoot™ controller from Indilinx Co., Ltd. This site is also intended to be a forum to share various simulators, tools, and workload generators and traces related to SSDs, among researchers in academia and industry.

OpenSSD Platforms

Indilinx Jasmine Platform

The Indilinx Jasmine Platform is the Indilinx’s reference implementation of SSD based on the Barefoot™ controller. The Indilinx’s Barefoot™ controller is an ARM-based SATA controller used in numerous high-performance SSDs such as Corsair Memory’s Extreme/Nova, Crucial Technology’s M225, G.Skill’s Falcon, A-RAM’s Pro series, OCZ’s Vertex/Vertex Turbo/Agility/Solid II, Patriot Memory’s Torqx/Koi, RunCore’s IV, SuperTalent’s Ultradrive ME/GX, etc. For more information on the Indilinx Jasmine Platform, please visit the following pages:

- Jasmine Platform Overview
- Jasmine Platform FAQs
- Jasmine Platform Technical Resources
- Download Jasmine Firmware
Thank you!