Reliability Issues

Jinyong Ha (jinyongha@csl.skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu
Contents

- DRAM Controller
- Power-Off Recovery
- Bad Block
# Schedule

<table>
<thead>
<tr>
<th>Date</th>
<th>Title</th>
</tr>
</thead>
<tbody>
<tr>
<td>3/11 (Mon)</td>
<td>Intro. to the Jasmine OpenSSD Platform</td>
</tr>
<tr>
<td>3/18 (Mon)</td>
<td>Dummy FTL</td>
</tr>
<tr>
<td>3/25 (Mon)</td>
<td>Tutorial FTL</td>
</tr>
<tr>
<td>4/1 (Mon)</td>
<td>Greedy FTL</td>
</tr>
<tr>
<td>4/8 (Mon)</td>
<td>Reliability Issues</td>
</tr>
<tr>
<td>4/15 (Mon)</td>
<td>Project #1</td>
</tr>
<tr>
<td>4/29 (Mon)</td>
<td>Project #1 Q&amp;A, Kernel-Based FTL</td>
</tr>
<tr>
<td>5/5 (Mon)</td>
<td>Project #2 Suggestions</td>
</tr>
<tr>
<td>5/27 (Mon)</td>
<td>Project #2 Progress Report</td>
</tr>
<tr>
<td>6/10 (Mon)</td>
<td>Project #2 Presentation</td>
</tr>
</tbody>
</table>
DRAM Controller

- DRAM ECC
  - Check and correct DRAM bit errors
  - 128 Byte + 4 Byte ECC Parity

- Memory Utility
  - Support data transmission from/to DRAM
    - SRAM to DRAM, DRAM to SRAM, DRAM to DRAM
  - ./include/mem_util.h
DRAM Controller

- Write 8 Byte data from SRAM to DRAM
Power-Off Recovery

- Power-off leads the data loss
  - Userdata that resides in write buffer
  - Metadata that resides in SRAM and DRAM
    - Page-level mapping table

- Recover mapping table on next power-on
  - On program operation, store LPN into spare area
  - By scanning spare area, mapping table can be reconstructed
Power-Off Recovery

- Store and recover metadata

* **Power off**
  - SRAM
  - DRAM

* **Power on**
  - SRAM
  - DRAM

Flash

Meta block

Store and recover metadata
Bad Block

- Bad block cannot be used
  - Data in bad block is crashed
  - Any operation cannot be operated
Bad Block

- Initial bad block
  - Bad block caused by problem in manufacturing process

- Runtime bad block
  - Bad block caused by wearing out
Bad Block

- **Bad block detecting**
  - Initial bad block
    - Scan block #0
  - Runtime bad block
    - Invoke interrupt service

- **Bad block handling**
  - Avoid using bad block
  - Remapping to reserved block
Bad Block

- `ftl_isr()`

```c
// BSP interrupt service routine
void ftl_isr(void)
{
    UINT32 bank;
    UINT32 bsp_intr_flag;

    uart_print("BSP interrupt occurred...");
    // interrupt pending clear (ICU)
    SETREG(APB_INT_STS, INTR_FLASH);

    for (bank = 0; bank < NUM_BANKS; bank++) {
        while (BSP_PSM(bank) != BANK_IDLE);
        // get interrupt flag from BSP
        bsp_intr_flag = BSP_INTR(bank);

        if (bsp_intr_flag == 0) {
            continue;
        }
    }

    UINT32 fc = GETREG(BSP_CMD(bank));
    // BSP clear
    CLR_BSP_INTR(bank, bsp_intr_flag);

    // interrupt handling
    if (bsp_intr_flag & IRQ_DATA_CORRUPT) {
        uart_printf("BSP interrupt at bank: 0x%x", bank);
        uart_printf("IRQ_DATA_CORRUPT occurred...");
    }
    if (bsp_intr_flag & (IRQ_BADBLK_H | IRQ_BADBLK_L)) {
        uart_printf("BSP interrupt at bank: 0x%x", bank);
        if (fc -- FC_COL_ROW_IN_PROG || fc == FC_IN_PROG || fc == FC_PROG) {
            uart_printf("Find runtime bad block when block program...");
        } else {
            uart_printf("Find runtime bad block when block erase... block #: %d", GETREG(BSP_ROW_H(bank)) / PAGES_PER_BLK);
            ASSERT(fc == FC_ERASE);
        }
    }
}
```
Any Questions?