Tutorial FTL

Prof. Dongkun Shin (dongkun@skku.edu)
TA – Junho Lee (crow6316@skku.edu)
TA – Somm Kim (sommkim@skku.edu)
Embedded Software Laboratory
Sungkyunkwan University
http://nyx.skku.ac.kr

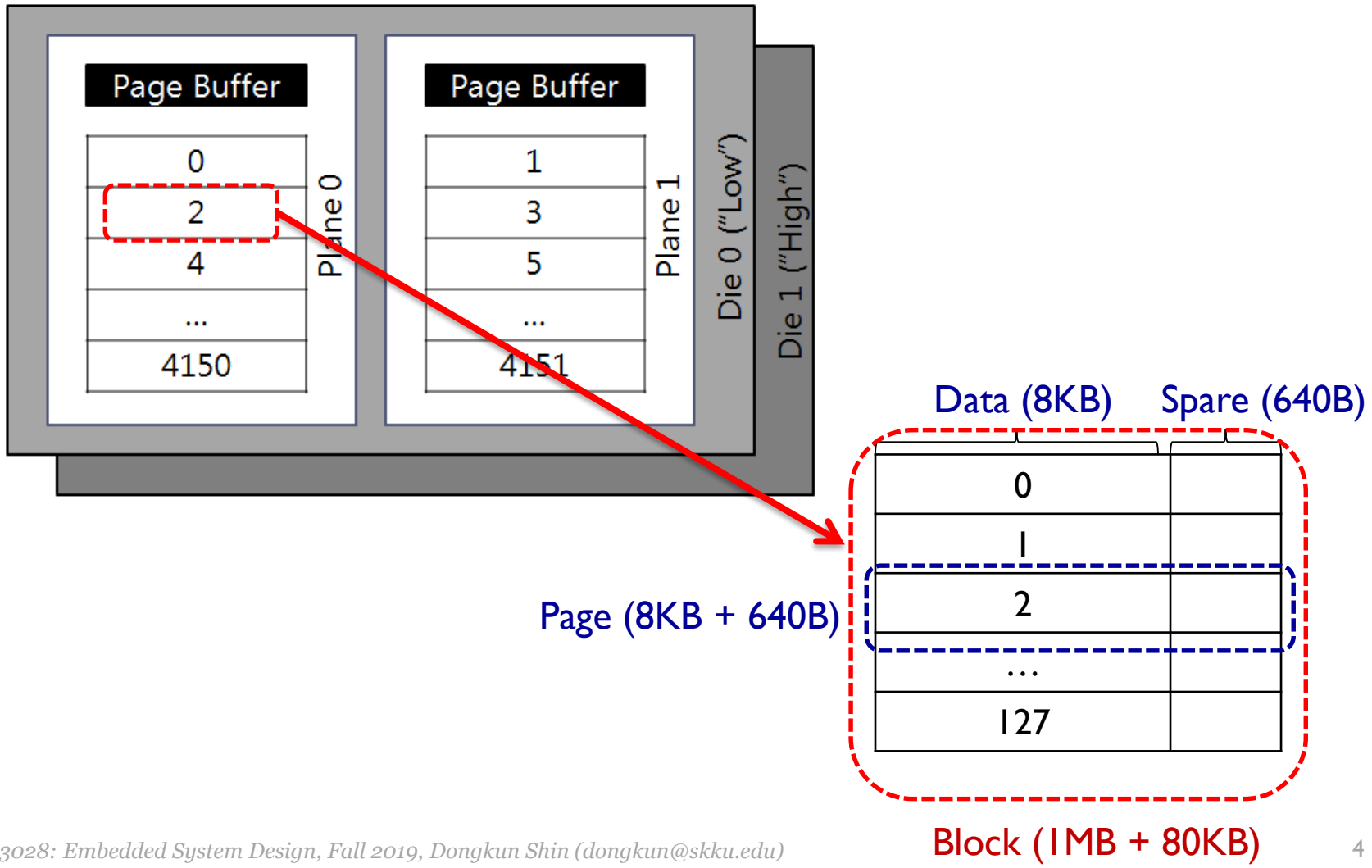
Contents

- Deep dive to the Jasmine
  - NAND flash memory
    - NAND flash memory configuration
    - NAND flash memory layout
  - Flash memory abstraction in Jasmine
  - Flash interface layer in Jasmine
    - NAND flash controller
- Intro. to Tutorial FTL
  - Read / Write in Tutorial FTL
- Page mapping FTL simulator

NAND Flash Memory

- K9LCG08U1M (Dual die)
  - Samsung 35 nm 2-bit MLC flash
  - Configuration
    - 16 sectors per page (8 KB + 640 B)
    - 128 pages per block (1 MB + 80 KB)
    - 4096 + 56 blocks per die
  - Average performance
    - Page read : 250 us
    - Page program : 1.3 ms
    - Block erase : 1.5 ms

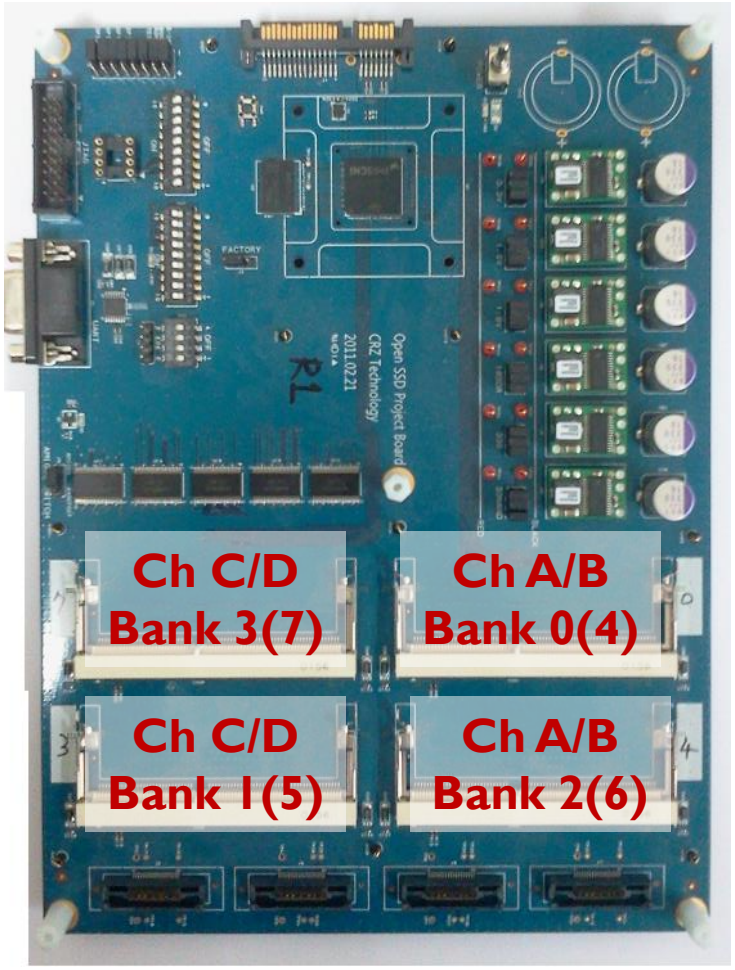
NAND Flash Memory(cont'd)

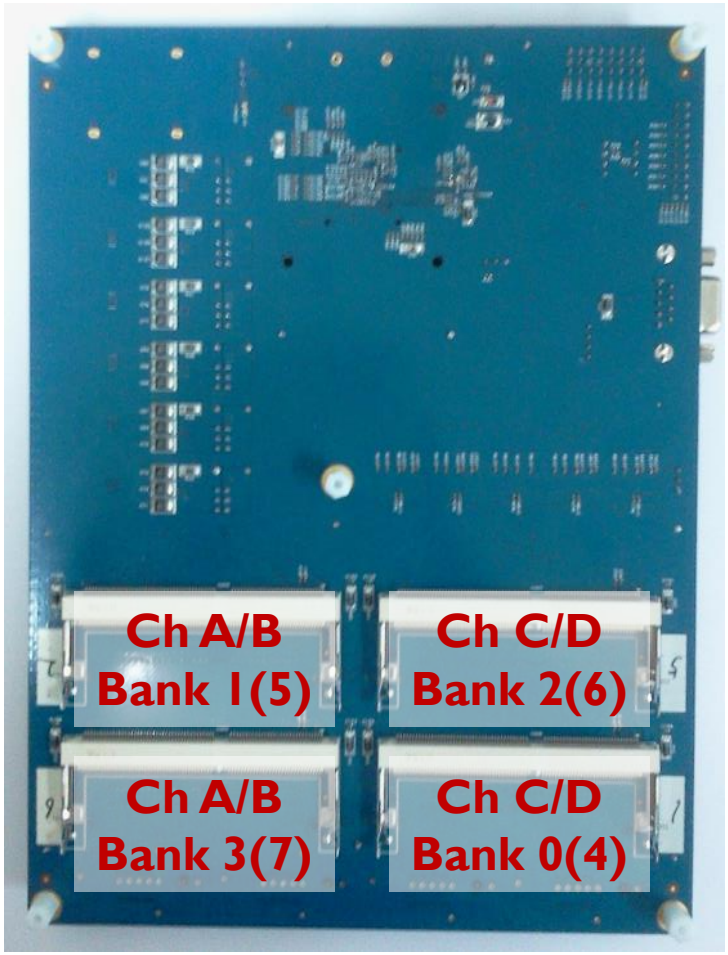


NAND Flash Operations

- Page read
  - Cell -> Page Buffer -> RAM
- Page program
  - RAM - > Page Buffer -> Cell
- Page copy-back
  - Cell (src) -> Page Buffer -> Cell (dst)

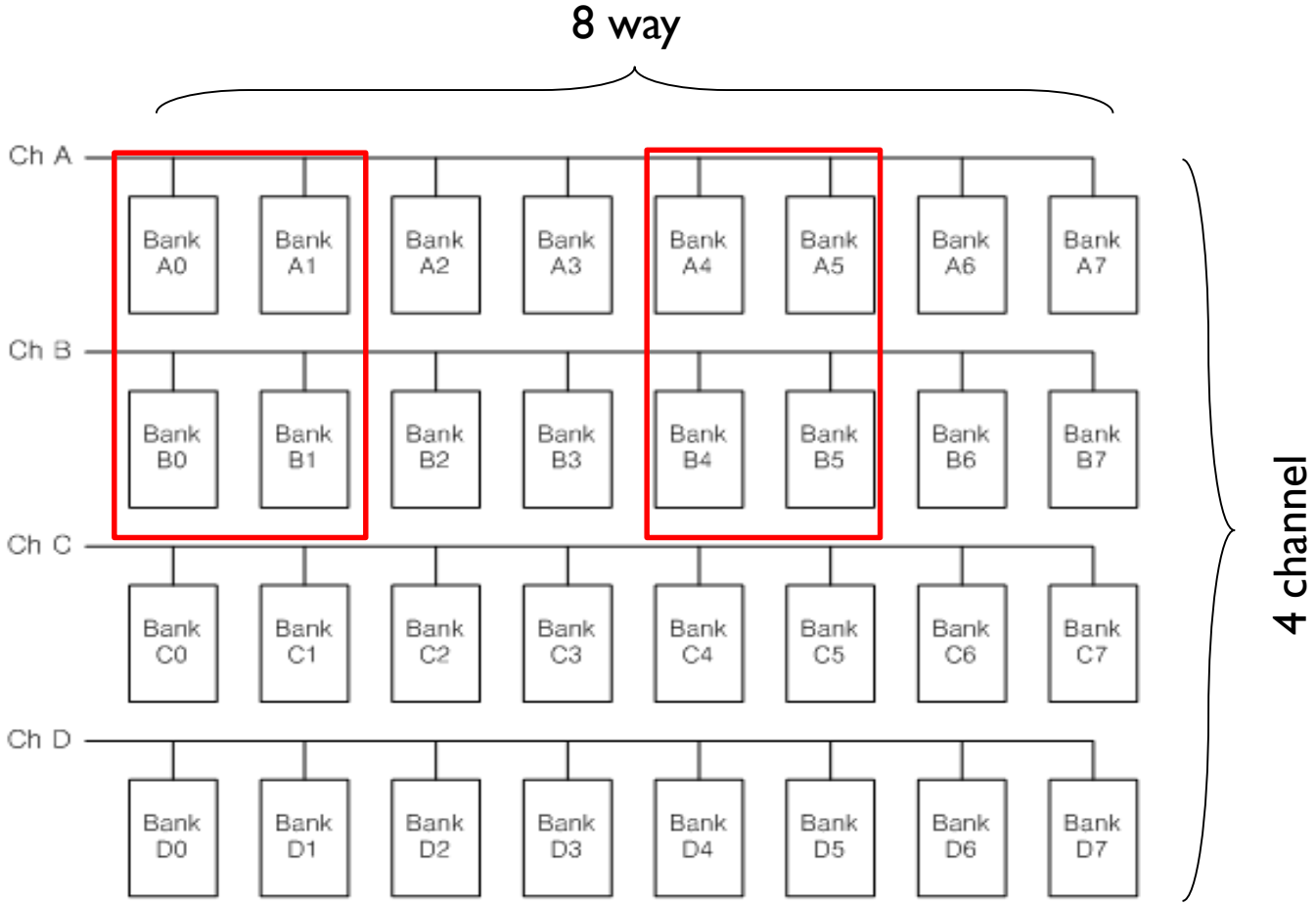
NAND Flash Physical Layout





NAND Flash Logical Layout

- Four channels + eight banks in each channel

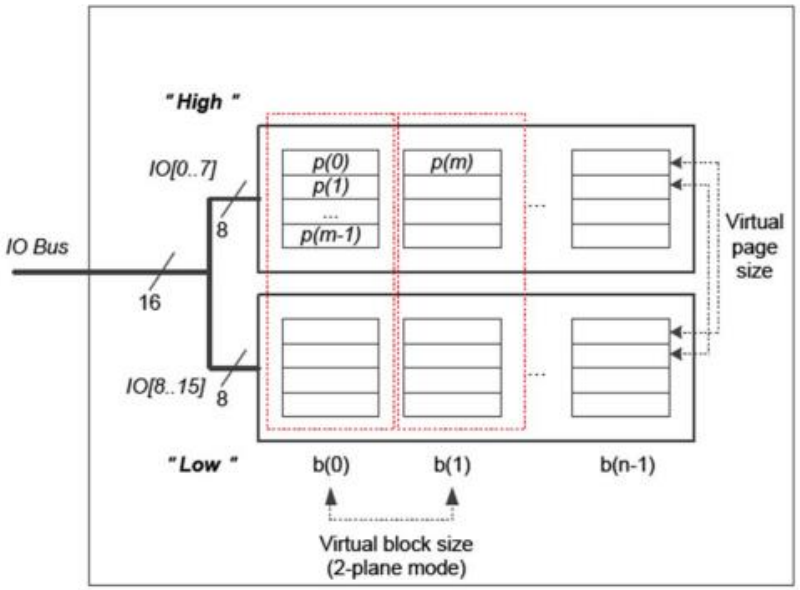


NAND Flash Logical Layout(cont'd)

- Channel
  - Banks in same channel share I/O bus
    - Only single I/O operation per a channel
- Bank
  - 16 bit I/O bus
  - High / Low die (dual die)
  - Each bank can perform cell operations in parallel
    - Internal operations not using I/O bus can run parallel
  - Barefoot has only 4 R/B signal inputs per channel
    - (0,4) (1,5) (2,6) (3,7) chip binding in a same channel
    - Maximum 4 way interleaving

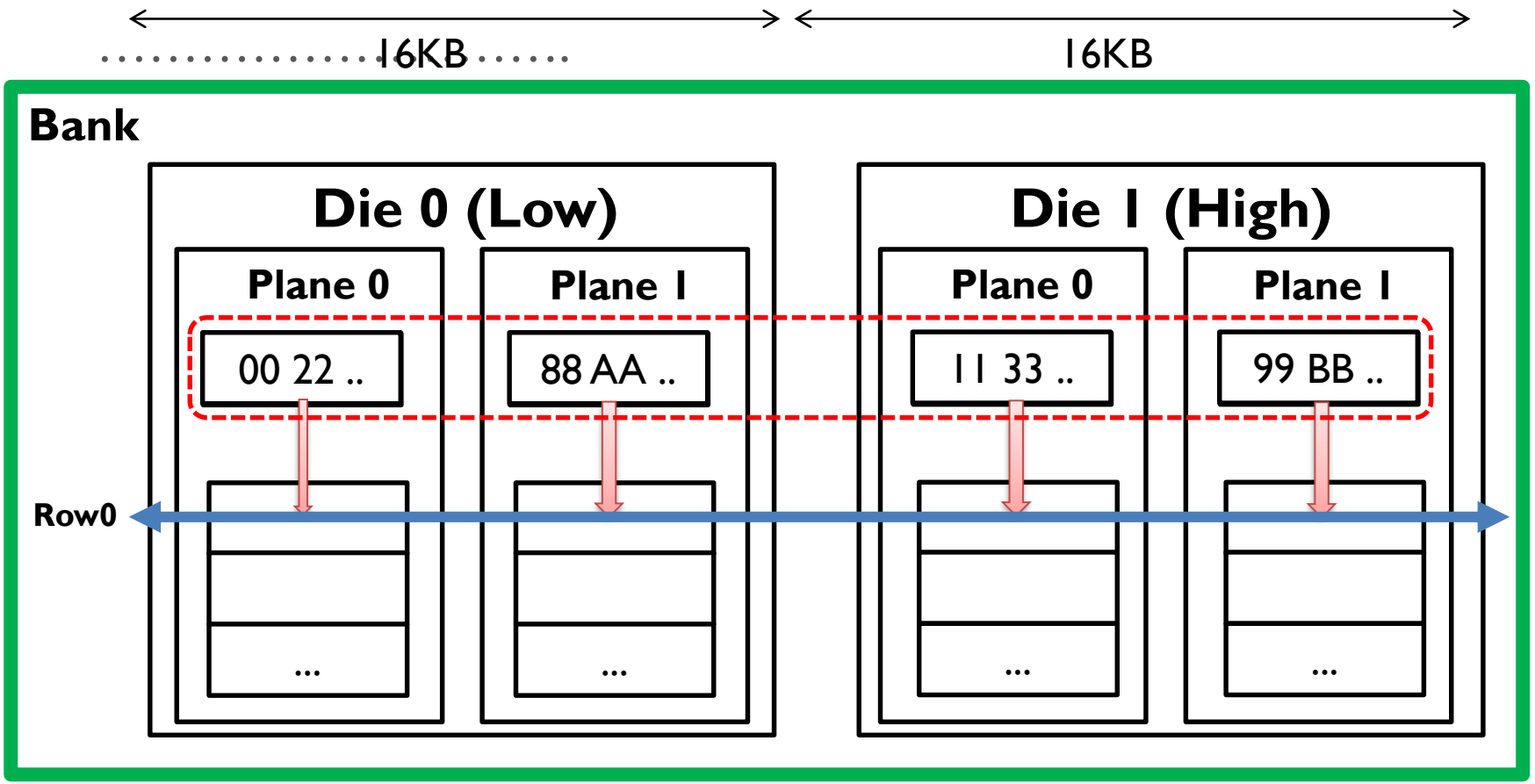
Flash Memory Abstraction

- Bank interleaved operation (dual die)
  - Example
    - DRAM buffer data = 01 23 45 67 89 AB CD EF 01 … (512 bytes)
    - 'Low' chip <- 01 45 89 CD 01 … (256 bytes)
    - 'High' chip <- 23 67 AB EF … (256 bytes)



Flash Memory Abstraction(cont'd)

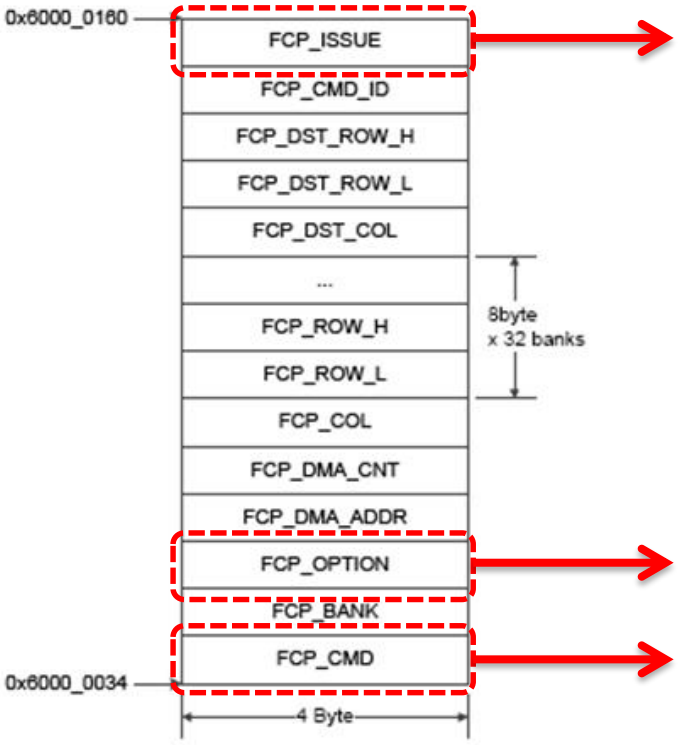
- 2-plane mode
  - 00 11 22 33 ..... 88 99 AA BB



NAND Flash Controller

- To issue NAND flash operation
  - include/flash.h

Registers	Offset
FCP_ISSUE	87
FCP_CMD_ID	86
FCP_DST_ROW_H	85
FCP_DST_ROW_L	84
FCP_DST_COL	83
FCP_ROW31_H	82
FCP_ROW31_L	81
...	...
FCP_ROW0_H	20
FCP_ROW0_L	19
FCP_COL	18
FCP_DMA_CNT	17
FCP_DMA_ADDR	16
FCP_OPTION	15
FCP_BANK	14
FCP_CMD(FCP_BASE)	13
WR_BANK	12
WR_STAT	11

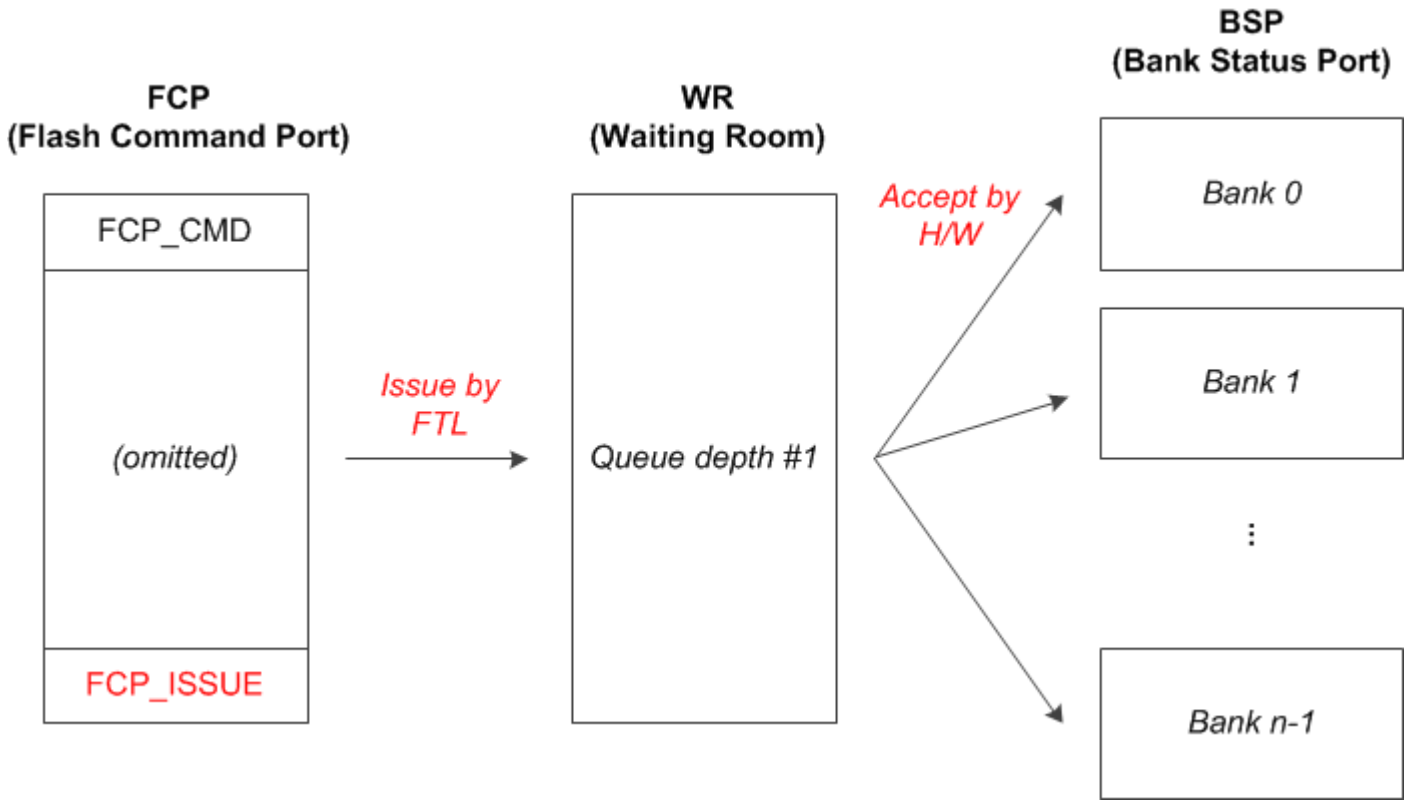


FO_P
FO_E
FO_B_SATA_W
FO_B_SATA_R

FC_COL_ROW_READ_OUT
FC_COL_ROW_IN_PROG
FC_COPYBACK

NAND Flash Controller (cont'd)

- To issue NAND flash operation
  - include/flash.h



Dive to the code: FCP Setting

```
void nand_page_ptread_to_host(UINT32 const bank, UINT32 const vblock,
UINT32 const page_num, UINT32 const sect_offset, UINT32 const num_sectors)
{
…
    row = (vblock * PAGES_PER_BLK) + page_num;

    SETREG(FCP_CMD, FC_COL_ROW_READ_OUT);
    SETREG(FCP_DMA_ADDR, RD_BUF_PTR(g_ftl_read_buf_id));
    SETREG(FCP_DMA_CNT, num_sectors * BYTES_PER_SECTOR);

    SETREG(FCP_COL, sect_offset);
…
    SETREG(FCP_OPTION, FO_P | FO_E | FO_B_SATA_R);
…
    SETREG(FCP_ROW_L(bank), row);
    SETREG(FCP_ROW_H(bank), row);

    g_ftl_read_buf_id = (g_ftl_read_buf_id + 1) % NUM_RD_BUFFERS;
…
    flash_issue_cmd(bank, RETURN_ON_ISSUE);
}
```

Dive to the code: FCP Issue

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

        SETREG(FCP_BANK, rbank);

        while ((GETREG(WR_STAT) & 0x00000001) != 0);

        SETREG(FCP_ISSUE, NULL);

        if (sync == RETURN_ON_ISSUE)
                return;

        while ((GETREG(WR_STAT) & 0x00000001) != 0);

        if (sync == RETURN_ON_ACCEPT)
                return;

        while (_BSP_FSM(rbank) != BANK_IDLE);
}
```

Tutorial FTL

Tutorial FTL: Overview

- ./ftl_tutorial
  - ftl.c, ftl.h
- Page Mapping FTL
  - Write data from DRAM to NAND
  - Read data from NAND to DRAM
  - No garbage collection

Tutorial FTL: Page Mapping Table

- LPN to PPN map
  - LPN (Logical Page Number)
    = LBA (from host) / Sectors per Page
  - PPN (Physical Page Number)

Index (=LPN)	0	1	2	3	…	209715
Value (=PPN)	100	256	0	INVALID	…	20000

```
static UINT32 get_physical_address(UINT32 const lpage_addr)
{
    // Page mapping table entry size is 4 byte.
    return read_dram_32(PAGE_MAP_ADDR + lpage_addr * sizeof(UINT32));
}

static void update_physical_address(UINT32 const lpage_addr, UINT32 const new_bank, UINT32 const new_row)
{
    write_dram_32(PAGE_MAP_ADDR + lpage_addr * sizeof(UINT32), new_bank * PAGES_PER_BANK + new_row);
}
```

Tutorial FTL

- Replace some part of tutorial ftl with flash wrapper functions in target_spw/flash_wrapper.c as possible as you can.

- Analyze the firmware code.

Page Mapping FTL Simulator

Lab 4 : Page Mapping FTL Simulator

- Develop a Page Mapping FTL simulator
  - Simulate the operations of page mapping FTL
    - Manage mapping with L2P table
  - Assumption
    - Page size: 32B(data area) + 4B(spare area)
    - Uniform size of write/read request (request size: 32B)
    - Initial state of flash blocks: empty
  - Misc.
    - Write in order starting with block #0, page #0
    - Page stripping by bank
    - Victim Selection policy: Greedy policy & Cost-Benefit policy
    - GC triggering condition: when # of free blocks becomes one
      - Use 'N_GC_BLOCKS' define

Victim Selection policy

- Greedy policy
  - Selects a block with the largest amount of invalid page
  - A block with the minimum utilization u

$$u = \frac{Number\ of\ valid\ pages\ in\ a\ block}{Number\ of\ Pages\ in\ a\ block}$$

- Cost-Benefit policy
  - Selects a block with the maximum

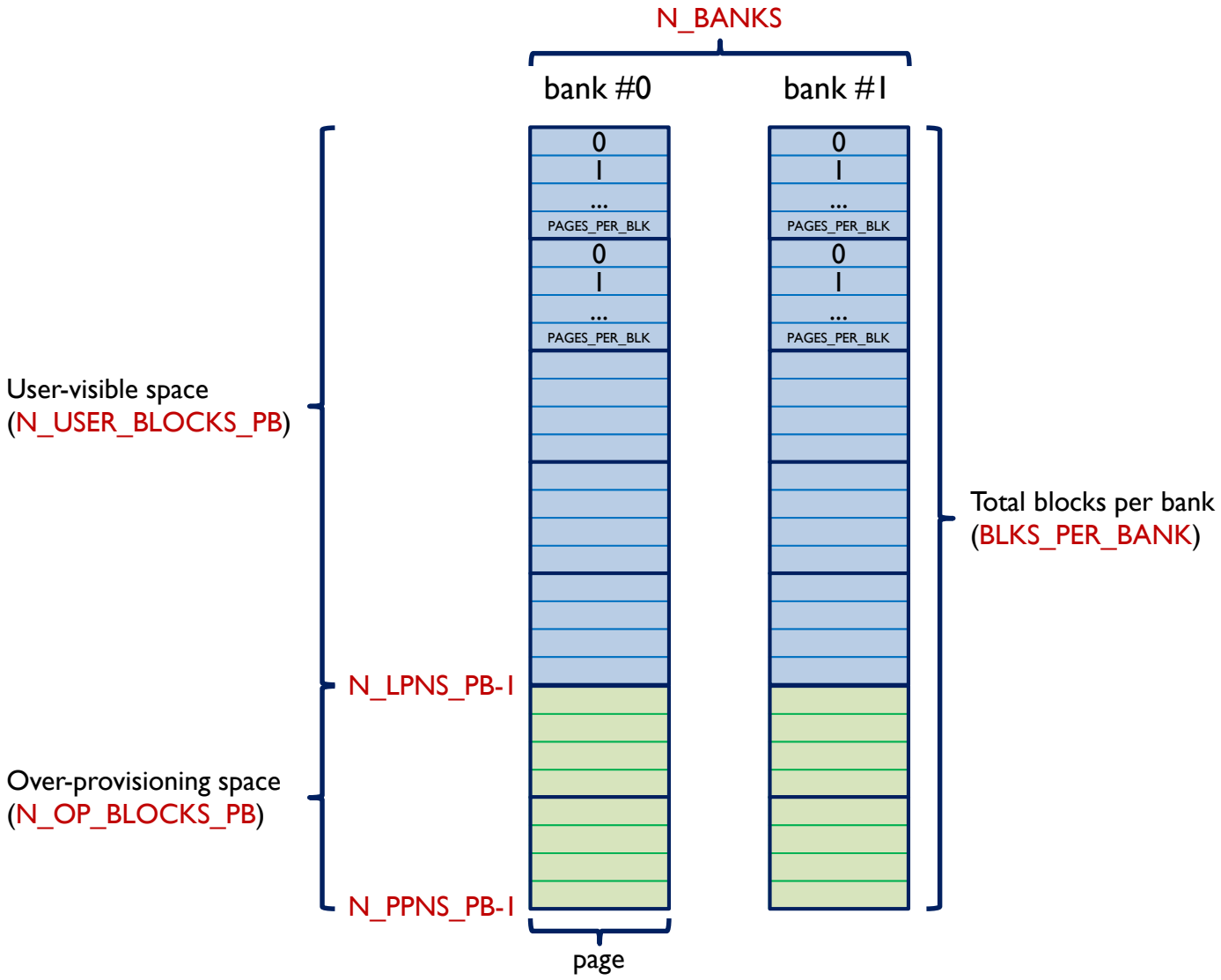
$$\frac{Benefit}{Cost} = \frac{(1-u)}{2u} \times age$$

  - u: utilization
  - age: the time since the last page invalidation

Data Structures

- Per-bank information
  - L2P table
  - Store the LPN into the spare area of written page
  - Free block info & Used block info
  - Per-block information
    - page validity status
    - # of valid pages

Configurations



ftl.c

- ftl_open()
  - Initialize NAND
  - Initialize data structure
- ftl_read(u32 lpn, u32 *read_buffer)
  - Get the PPN through the L2P table
  - Read it from NAND
- ftl_write(u32 lpn, u32 *write_buffer)
  - Check GC
  - If old data already exist, invalid old page
  - Write it to NAND
  - Set the PPN into the L2P table
- garbage_collection(u32 bank)
  - Select the victim block
    - Note: Block currently performing write should not be selected as victim block
  - Copy valid page from victim block to free block
  - Erase victim block

- You can modify only ftl.c and nand.c.

sample output (Random Workload)

```
Bank: 2
Blocks / Bank: 32 blocks
Pages / Block: 32 pages
OP ratio: 7%
Physical Blocks: 64
User Blocks: 56
OP Blocks: 8
PPNs: 2048
LPNs: 1792
Workload: Random
FTL: Greedy policy

[Run 1] host 1792, valid page copy 0, GC# 0, WAF 1.00
[Run 2] host 3584, valid page copy 1297, GC# 91, WAF 1.36
[Run 3] host 5376, valid page copy 4595, GC# 250, WAF 1.85
[Run 4] host 7168, valid page copy 9533, GC# 460, WAF 2.33
[Run 5] host 8960, valid page copy 15672, GC# 708, WAF 2.75
[Run 6] host 10752, valid page copy 22268, GC# 970, WAF 3.07
[Run 7] host 12544, valid page copy 29082, GC# 1239, WAF 3.32
[Run 8] host 14336, valid page copy 35836, GC# 1506, WAF 3.50
[Run 9] host 16128, valid page copy 42494, GC# 1770, WAF 3.63
[Run 10] host 17920, valid page copy 49498, GC# 2045, WAF 3.76
```

```
[Run 190] host 340480, valid page copy 1283419, GC# 50685, WAF 4.77
[Run 191] host 342272, valid page copy 1290265, GC# 50955, WAF 4.77
[Run 192] host 344064, valid page copy 1297212, GC# 51228, WAF 4.77
[Run 193] host 345856, valid page copy 1304088, GC# 51499, WAF 4.77
[Run 194] host 347648, valid page copy 1310773, GC# 51764, WAF 4.77
[Run 195] host 349440, valid page copy 1317691, GC# 52036, WAF 4.77
[Run 196] host 351232, valid page copy 1324412, GC# 52302, WAF 4.77
[Run 197] host 353024, valid page copy 1331351, GC# 52575, WAF 4.77
[Run 198] host 354816, valid page copy 1338205, GC# 52845, WAF 4.77
[Run 199] host 356608, valid page copy 1345084, GC# 53116, WAF 4.77
[Run 200] host 358400, valid page copy 1352027, GC# 53389, WAF 4.77

Results ------
Host writes: 358400
GC writes: 1352027
Number of GCs: 53389
Valid pages per GC: 25.32 pages
WAF: 4.77
```

```
Bank: 2
Blocks / Bank: 32 blocks
Pages / Block: 32 pages
OP ratio: 7%
Physical Blocks: 64
User Blocks: 56
OP Blocks: 8
PPNs: 2048
LPNs: 1792
Workload: Random
FTL: Cost-Benefit policy

[Run 1] host 1792, valid page copy 0, GC# 0, WAF 1.00
[Run 2] host 3584, valid page copy 1431, GC# 95, WAF 1.40
[Run 3] host 5376, valid page copy 5144, GC# 267, WAF 1.96
[Run 4] host 7168, valid page copy 10713, GC# 497, WAF 2.49
[Run 5] host 8960, valid page copy 17723, GC# 772, WAF 2.98
[Run 6] host 10752, valid page copy 25084, GC# 1058, WAF 3.33
[Run 7] host 12544, valid page copy 32764, GC# 1354, WAF 3.61
[Run 8] host 14336, valid page copy 40598, GC# 1655, WAF 3.83
[Run 9] host 16128, valid page copy 48633, GC# 1962, WAF 4.02
[Run 10] host 17920, valid page copy 56637, GC# 2268, WAF 4.16
```

```
[Run 190] host 340480, valid page copy 1478841, GC# 56792, WAF 5.34
[Run 191] host 342272, valid page copy 1486840, GC# 57098, WAF 5.34
[Run 192] host 344064, valid page copy 1494584, GC# 57396, WAF 5.34
[Run 193] host 345856, valid page copy 1502491, GC# 57699, WAF 5.34
[Run 194] host 347648, valid page copy 1510619, GC# 58009, WAF 5.35
[Run 195] host 349440, valid page copy 1518495, GC# 58311, WAF 5.35
[Run 196] host 351232, valid page copy 1526621, GC# 58621, WAF 5.35
[Run 197] host 353024, valid page copy 1534711, GC# 58930, WAF 5.35
[Run 198] host 354816, valid page copy 1542587, GC# 59232, WAF 5.35
[Run 199] host 356608, valid page copy 1550461, GC# 59534, WAF 5.35
[Run 200] host 358400, valid page copy 1558299, GC# 59835, WAF 5.35

Results ------
Host writes: 358400
GC writes: 1558299
Number of GCs: 59835
Valid pages per GC: 26.04 pages
WAF: 5.35
```

sample output (Hot/Cold Workload)

```
Bank: 2
Blocks / Bank: 32 blocks
Pages / Block: 32 pages
OP ratio: 7%
Physical Blocks: 64
User Blocks: 56
OP Blocks: 8
PPNs: 2048
LPNs: 1792
Workload: Hot 95 / Cold 5
FTL: Greedy policy

[Run 1] host 1792, valid page copy 0, GC# 0, WAF 1.00
[Run 2] host 3584, valid page copy 54, GC# 53, WAF 1.02
[Run 3] host 5376, valid page copy 200, GC# 113, WAF 1.04
[Run 4] host 7168, valid page copy 410, GC# 176, WAF 1.06
[Run 5] host 8960, valid page copy 713, GC# 241, WAF 1.08
[Run 6] host 10752, valid page copy 1124, GC# 310, WAF 1.10
[Run 7] host 12544, valid page copy 1629, GC# 382, WAF 1.13
[Run 8] host 14336, valid page copy 2230, GC# 457, WAF 1.16
[Run 9] host 16128, valid page copy 2899, GC# 533, WAF 1.18
[Run 10] host 17920, valid page copy 3691, GC# 614, WAF 1.21
```

```
Bank: 2
Blocks / Bank: 32 blocks
Pages / Block: 32 pages
OP ratio: 7%
Physical Blocks: 64
User Blocks: 56
OP Blocks: 8
PPNs: 2048
LPNs: 1792
Workload: Hot 95 / Cold 5
FTL: Cost-Benefit policy

[Run 1] host 1792, valid page copy 0, GC# 0, WAF 1.00
[Run 2] host 3584, valid page copy 66, GC# 53, WAF 1.02
[Run 3] host 5376, valid page copy 204, GC# 113, WAF 1.04
[Run 4] host 7168, valid page copy 427, GC# 176, WAF 1.06
[Run 5] host 8960, valid page copy 739, GC# 242, WAF 1.08
[Run 6] host 10752, valid page copy 1171, GC# 311, WAF 1.11
[Run 7] host 12544, valid page copy 1680, GC# 383, WAF 1.13
[Run 8] host 14336, valid page copy 2305, GC# 459, WAF 1.16
[Run 9] host 16128, valid page copy 3017, GC# 537, WAF 1.19
[Run 10] host 17920, valid page copy 3816, GC# 618, WAF 1.21
```

```
[Run 190] host 340480, valid page copy 1558367, GC# 59277, WAF 5.58
[Run 191] host 342272, valid page copy 1569180, GC# 59671, WAF 5.58
[Run 192] host 344064, valid page copy 1580283, GC# 60074, WAF 5.59
[Run 193] host 345856, valid page copy 1591134, GC# 60469, WAF 5.60
[Run 194] host 347648, valid page copy 1601948, GC# 60863, WAF 5.61
[Run 195] host 349440, valid page copy 1612412, GC# 61246, WAF 5.61
[Run 196] host 351232, valid page copy 1623101, GC# 61636, WAF 5.62
[Run 197] host 353024, valid page copy 1633693, GC# 62023, WAF 5.63
[Run 198] host 354816, valid page copy 1644893, GC# 62429, WAF 5.64
[Run 199] host 356608, valid page copy 1655515, GC# 62817, WAF 5.64
[Run 200] host 358400, valid page copy 1666714, GC# 63223, WAF 5.65

Results ------
Host writes: 358400
GC writes: 1666714
Number of GCs: 63223
Valid pages per GC: 26.36 pages
WAF: 5.65
```

```
[Run 190] host 340480, valid page copy 1536702, GC# 58600, WAF 5.51
[Run 191] host 342272, valid page copy 1547194, GC# 58984, WAF 5.52
[Run 192] host 344064, valid page copy 1557821, GC# 59372, WAF 5.53
[Run 193] host 345856, valid page copy 1568253, GC# 59754, WAF 5.53
[Run 194] host 347648, valid page copy 1578748, GC# 60138, WAF 5.54
[Run 195] host 349440, valid page copy 1589307, GC# 60524, WAF 5.55
[Run 196] host 351232, valid page copy 1599962, GC# 60913, WAF 5.56
[Run 197] host 353024, valid page copy 1610235, GC# 61290, WAF 5.56
[Run 198] host 354816, valid page copy 1620574, GC# 61669, WAF 5.57
[Run 199] host 356608, valid page copy 1631326, GC# 62061, WAF 5.57
[Run 200] host 358400, valid page copy 1641948, GC# 62449, WAF 5.58

Results ------
Host writes: 358400
GC writes: 1641948
Number of GCs: 62449
Valid pages per GC: 26.29 pages
WAF: 5.58
```

Miscellaneous

- Recommended environment : Linux (Ubuntu is ok!)
  - You can do it in Windows, but be sure that your work also runs in Linux (I'll score all the works only in Ubuntu 16.04)
- Personal Project
- You should submit a report
  - Describe your code
  - Capture and analyze the WAF of Greedy policy and Cost-Benefit policy
    - 4 cases (Random/Hot Cold Workload & Greedy/Cost-Benefit policy)
    - Use COST_BENEFIT & HOT_COLD defines in ftl.h
    - Note: WAF may vary depending on your implementation. But, if WAF is significantly low, your score will be degraded.
  - Analyze OP ratio vs. WAF
- Submit to the icampus
  - Due: 10/16(Wed.) 23:59:59
  - File to submit: ftl_sim.c, ftl.h, ftl.c, nand.h, nand.c, Makefile, report.pdf
  - File name: $(STUDENT_ID).tar.gz
- Late penalty : -20% per day (Up to 3 days)

Any Questions?