Dummy FTL

Yeongjae Woo (yjwoo@csl.skku.edu)
Computer Systems Laboratory
Sungkyunkwan University
http://csl.skku.edu

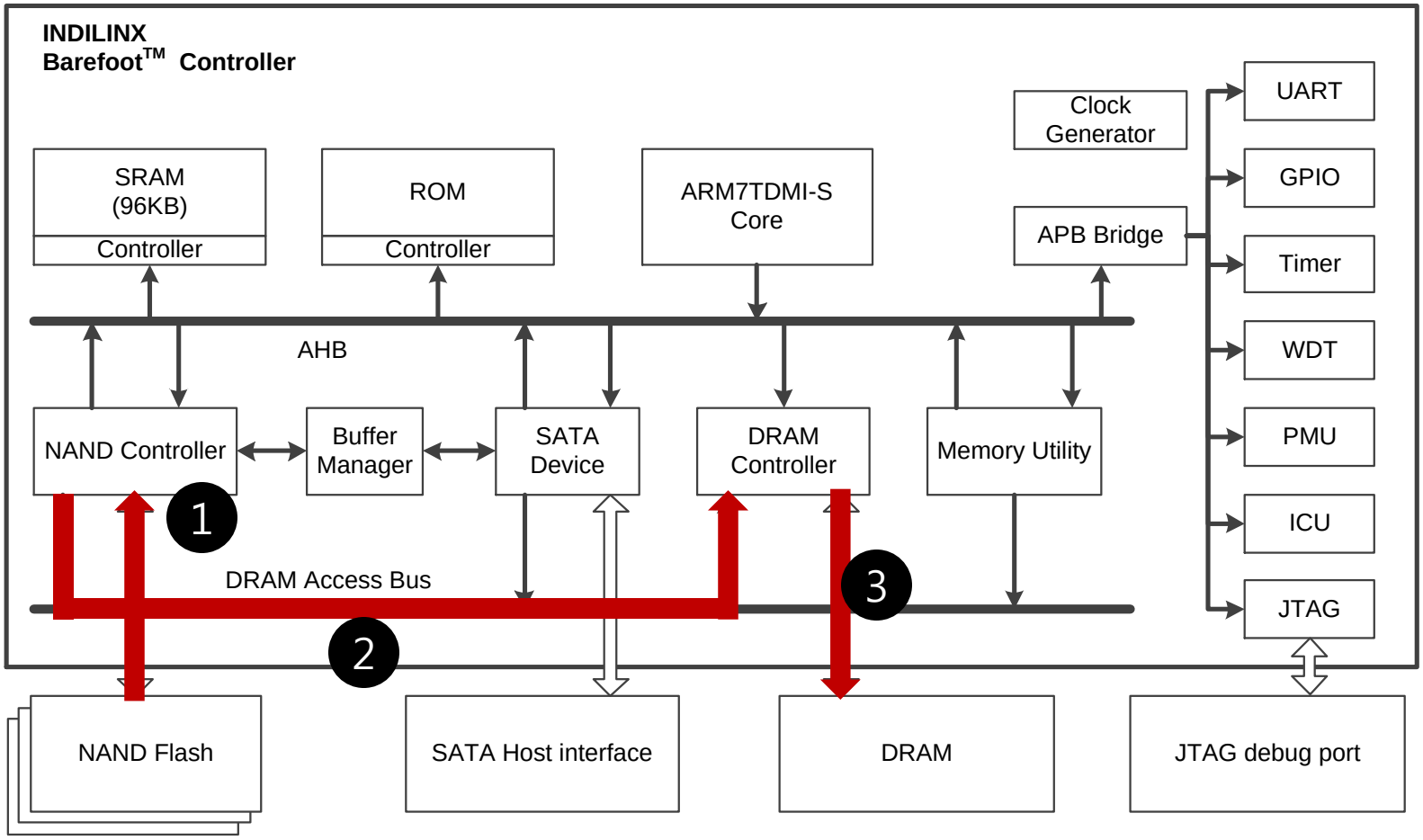
Contents

- Read/Write Command
- SATA Controller
- Buffer Manager
- Dummy FTL
- Appendix. Iometer
- Appendix. Memory Map

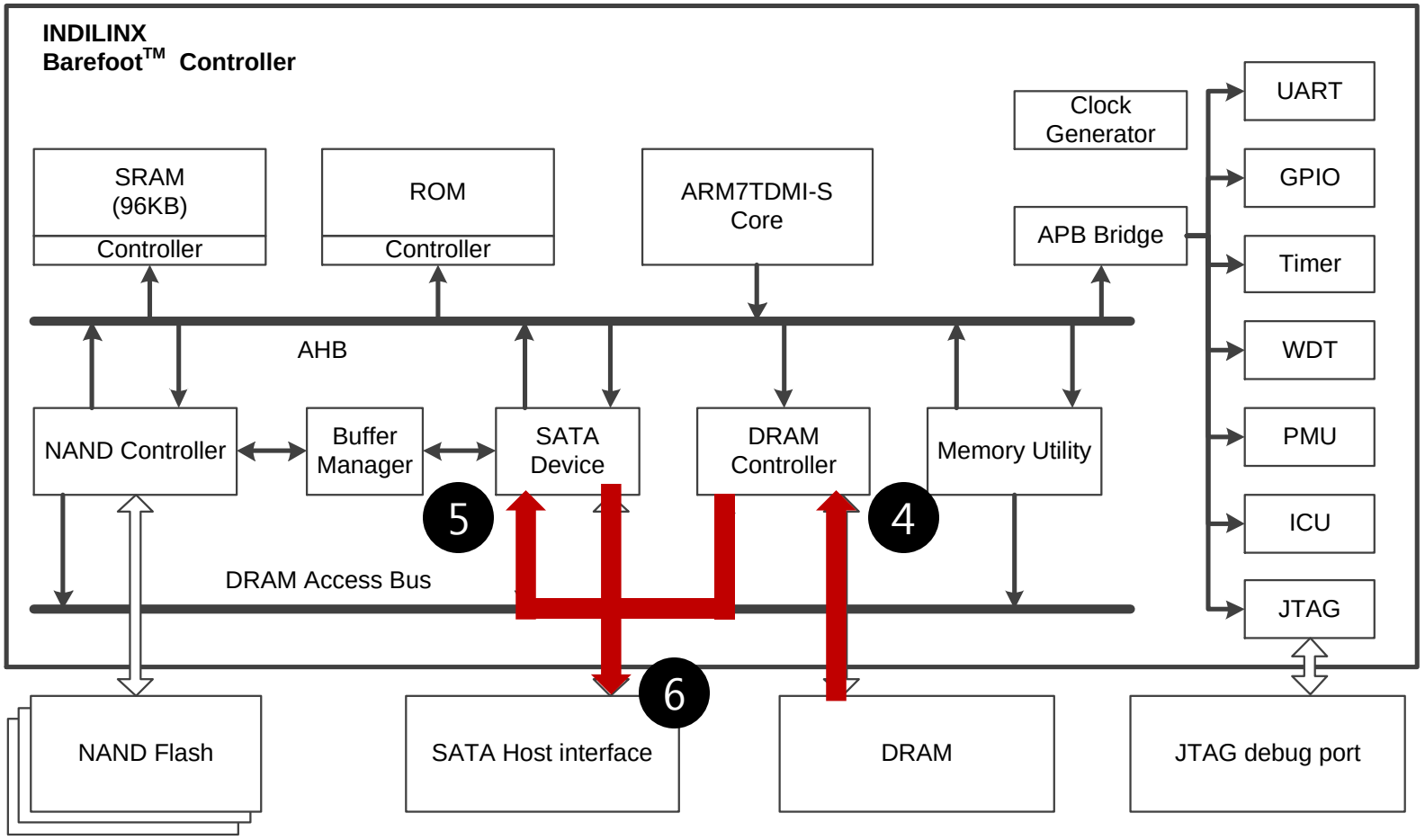
Schedule

Date	Title
3/15 (Th)	Intro. to the Jasmine OpenSSD Platform
3/22 (Th)	Dummy FTL
3/29 (Th)	Tutorial FTL
4/5 (Th)	Greedy FTL
4/12 (Th)	Reliability Issues
5/3 (Th)	Project #1 Presentation
5/14 (M)	Project #2 Proposal (1)
5/17 (Th)	Project #2 Proposal (2)
6/7 (Th)	Project #2 Progress Report
6/25 (M)	Project #2 Presentation

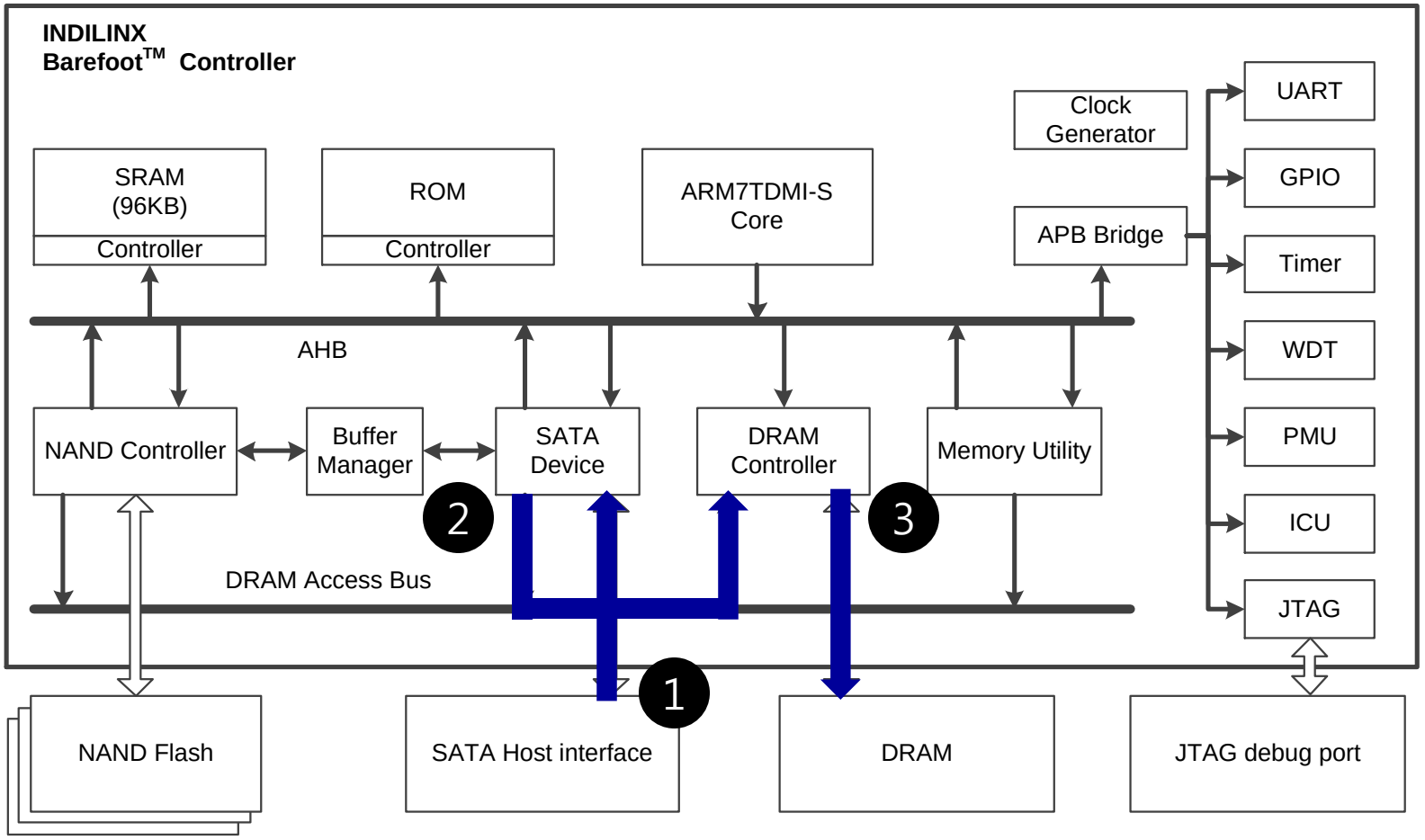
Read Command



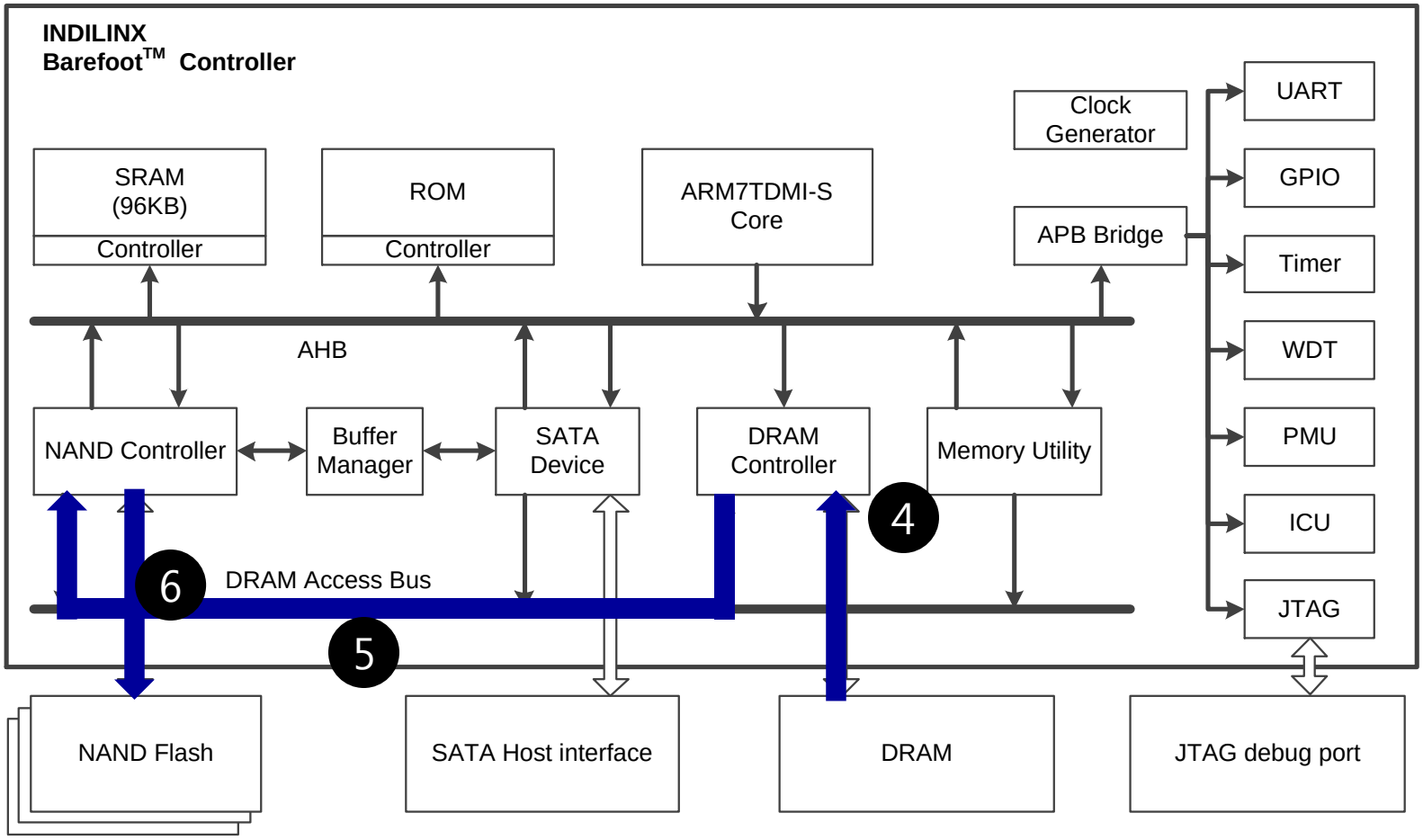
Read Command



Write Command



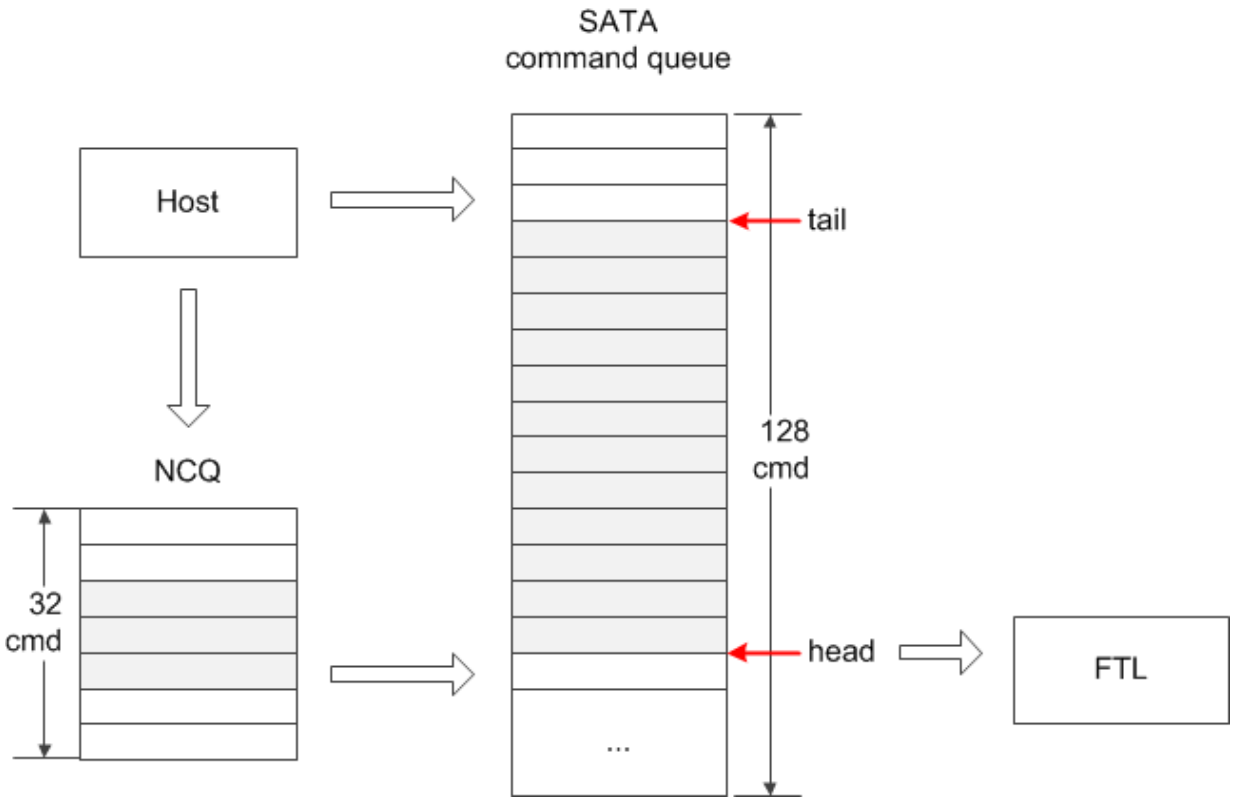
Write Command



SATA Controller

- SATA Event Queue
  - 128 slots for SATA commands
  - An entry is inserted by ISR upon command reception
  - An entry is removed by FTL top level loop and processed
- NCQ
  - 32 slots for NCQ commands
  - Currently, NCQ is disabled in Jasmine firmware

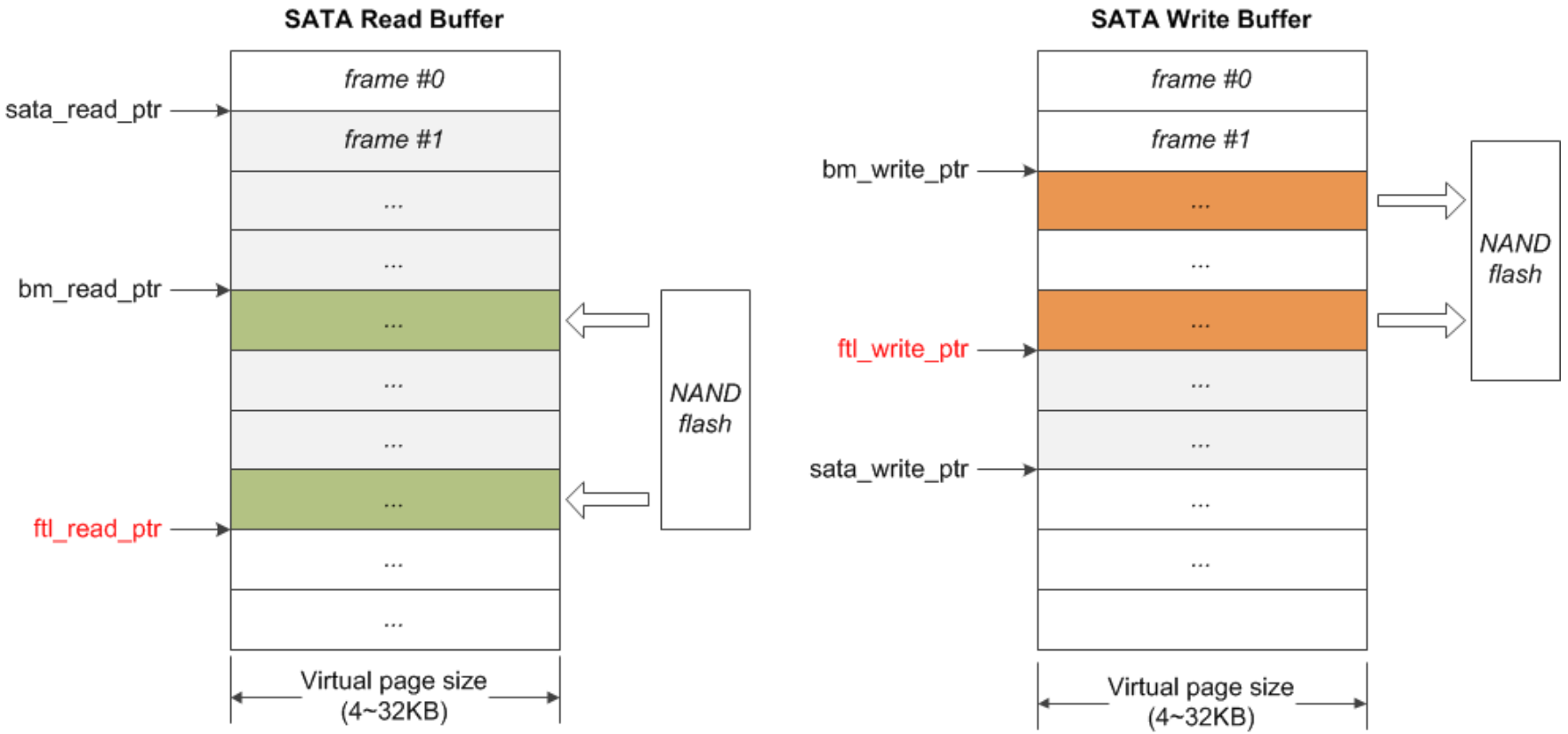
SATA Controller



Buffer Manager

- SATA data is buffered in DRAM
- Separate read and write buffer space
- Buffer space consists of multiple buffers
  - 4 ~ 32 KB per buffer
  - Must be identical to virtual flash page size

Buffer Manager



Buffer Manager

- ftl_read_ptr
  - Buffer ID maintained by firmware
  - Firmware reads data from NAND to ftl_read_ptr
- sata_read_ptr
  - Buffer ID which SATA read transfer is being done
- bm_read_ptr
  - Buffer ID which NAND read transfer is being done
  - sata_read_ptr does not run ahead of bm_read_ptr

Buffer Manager

- ftl_write_ptr
  - Buffer ID maintained by firmware
  - Firmware writes data from ftl_write_ptr to NAND
- sata_write_ptr
  - Buffer ID which SATA write transfer is being done
- bm_write_ptr
  - Buffer ID which NAND write transfer is being done
  - sata_write_ptr does not run ahead of bm_write_ptr

Dummy FTL

- ./ftl_dummy
  - ftl.c, ftl.h

- Literally, Dummy FTL is not a real FTL
  - Not access NAND flash at all
  - Write data to DRAM buffer
  - Read data from DRAM buffer

Start-up

- ./target_spw/init_gnu.s
  - Call init_jasmine()
  - Call Main()
- init_jasmine()
  - Initialize H/W configuration
- Main()
  - FTL top level loop

Start-up

- ./sata/sata_main.c

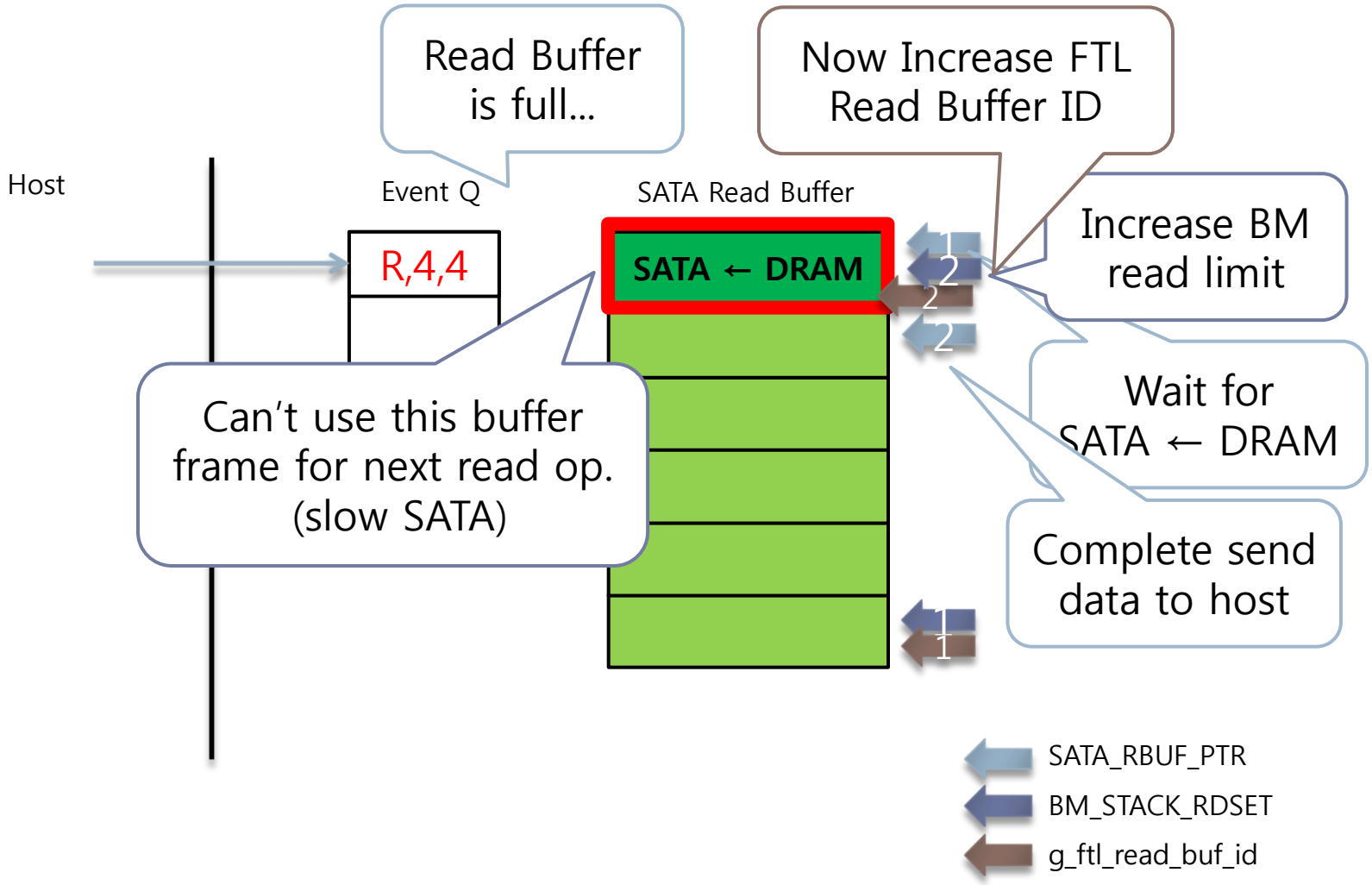
```
void Main(void)
{
    while (1)
    {
        if (eventq_get_count())
        {
            CMD_T cmd;

            eventq_get(&cmd);

            if (cmd.cmd_type == READ)
            {
                ftl_read(cmd.lba, cmd.sector_count);
            }
            else
            {
                ftl_write(cmd.lba, cmd.sector_count);
            }
        }
        else if (g_sata_context.slow_cmd.status == SLOW_CMD_STATUS_PENDING)
        {
            void (*ata_function)(UINT32 lba, UINT32 sector_count);

            slow_cmd_t* slow_cmd = &g_sata_context.slow_cmd;
            slow_cmd->status = SLOW_CMD_STATUS_BUSY;

            ata_function = search_ata_function(slow_cmd->code);
            ata_function(slow_cmd->lba, slow_cmd->sector_count);

            slow_cmd->status = SLOW_CMD_STATUS_NONE;
        }
        else
        {
            // idle time operations
        }
    } ? end while 1 ?
} ? end Main ?
```

Read Operation

- ./ftl_dummy/ftl.c

```
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;
        }

        UINT32 next_read_buf_id = (g_ftl_read_buf_id + 1) % NUM_RD_BUFFERS;

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

        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;

        sect_offset = 0;
        sectors_remain -= num_sectors_to_read;
        lpage_addr++;
    } ? end while sectors_remain! =0 ?
} ? end ftl_read ?
```

Read Operation

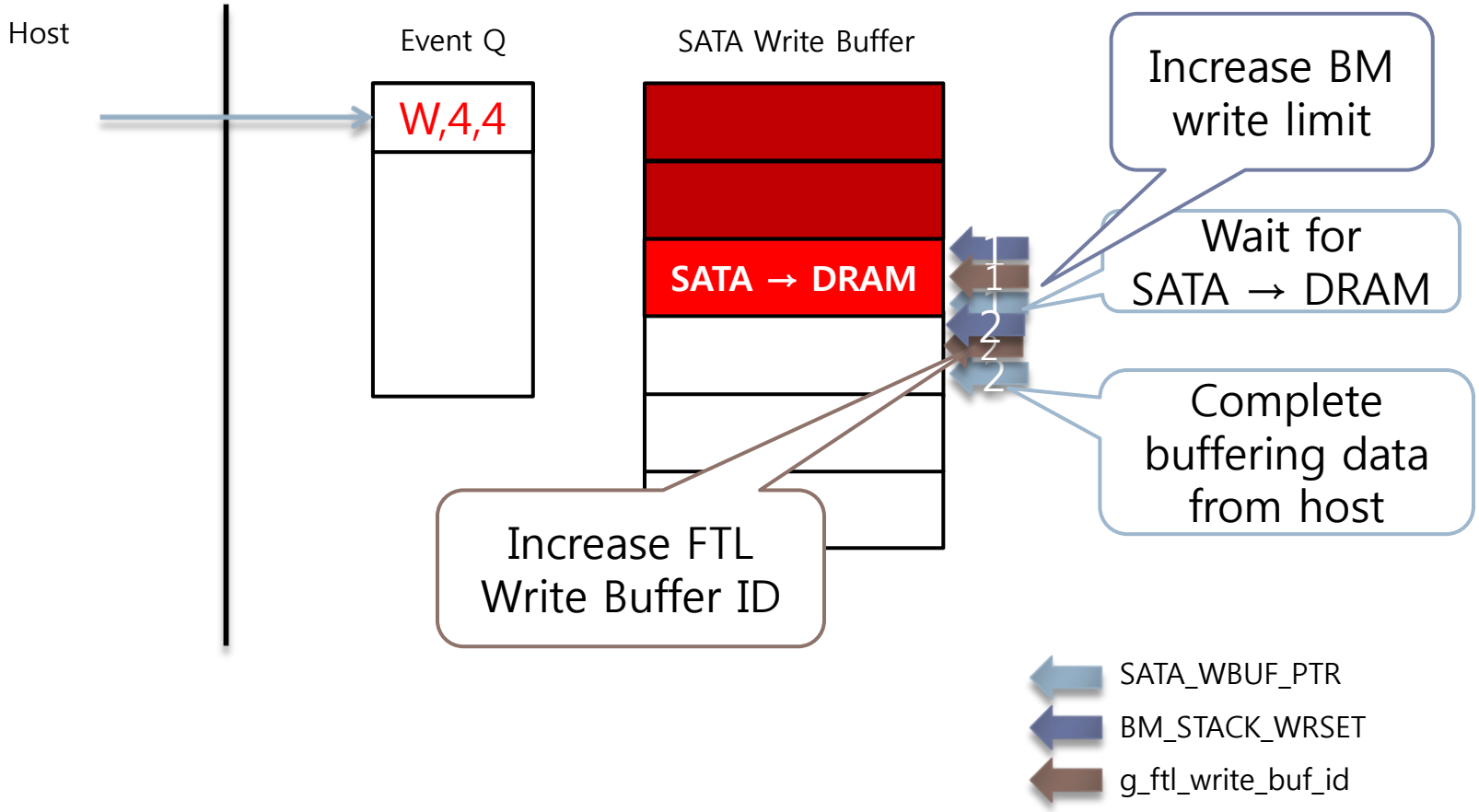


Write Operation

- ./ftl_dummy/ftl.c

```
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;
        }

        while (g_ftl_write_buf_id == GETREG(SATA_WBUF_PTR));     // bm_write_limit should not outpace SATA_WBUF_PTR

        g_ftl_write_buf_id = (g_ftl_write_buf_id + 1) % NUM_WR_BUFFERS;     // Circular buffer

        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;
    } ? end while remain_sectors! =0 ?
} ? end ftl_write ?
```

Write Operation



Any Questions?

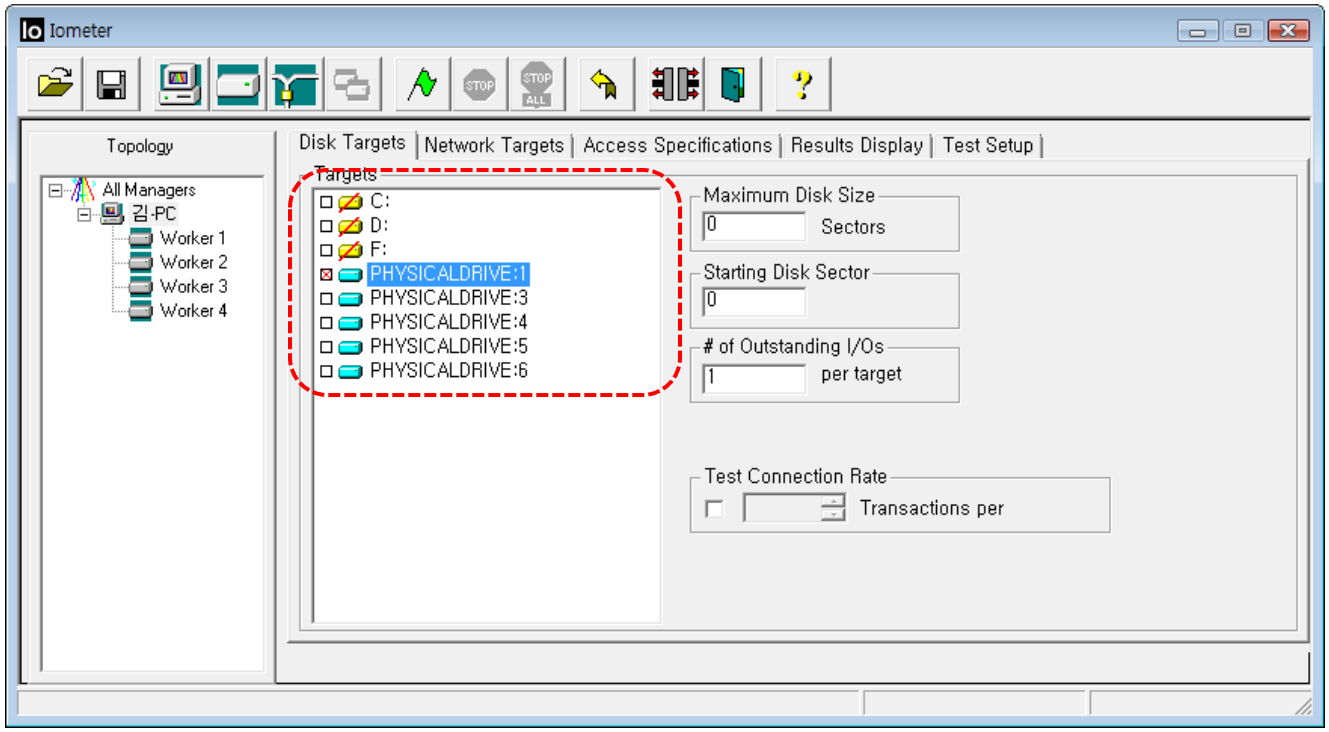
Iometer

- Performance measurement tool for storage system
  - Perform I/Os accessing a file or block device
  - http://www.iometer.org

- Performance is measured in
  - IOPS (IOs Per Second)
  - MB/s (Mega Bytes Per Second)
  - Response time

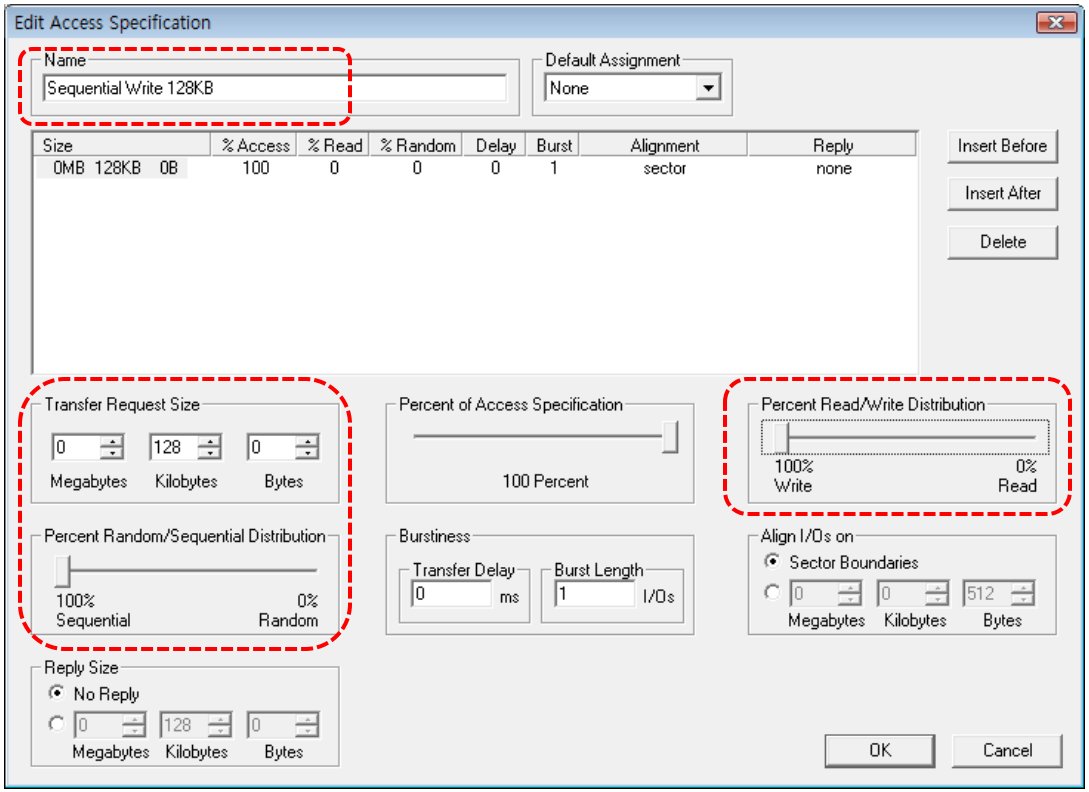
Example

- Install & Run Iometer
- Select disk target



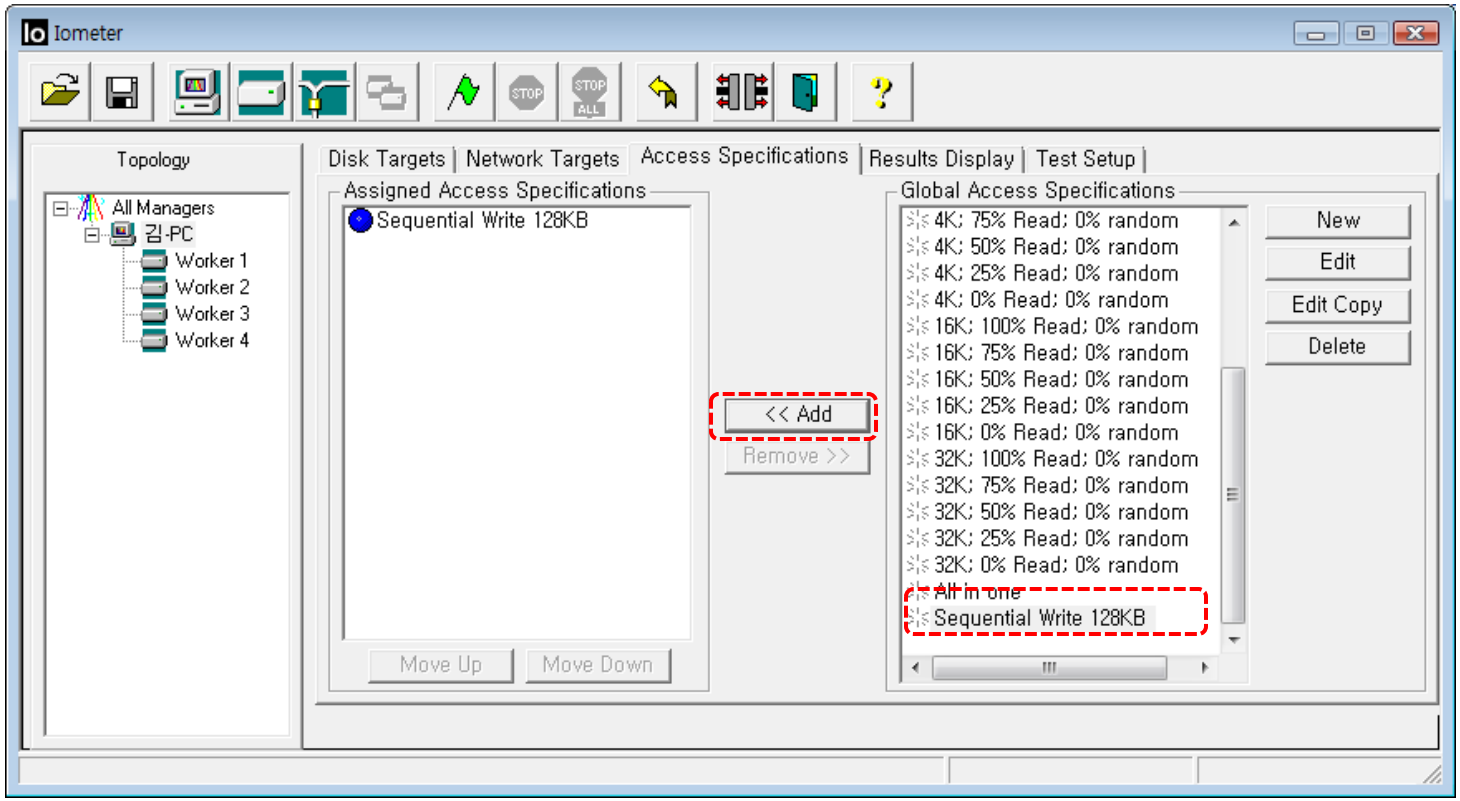
Example

- Make new access specification
  - Access Specifications -> New



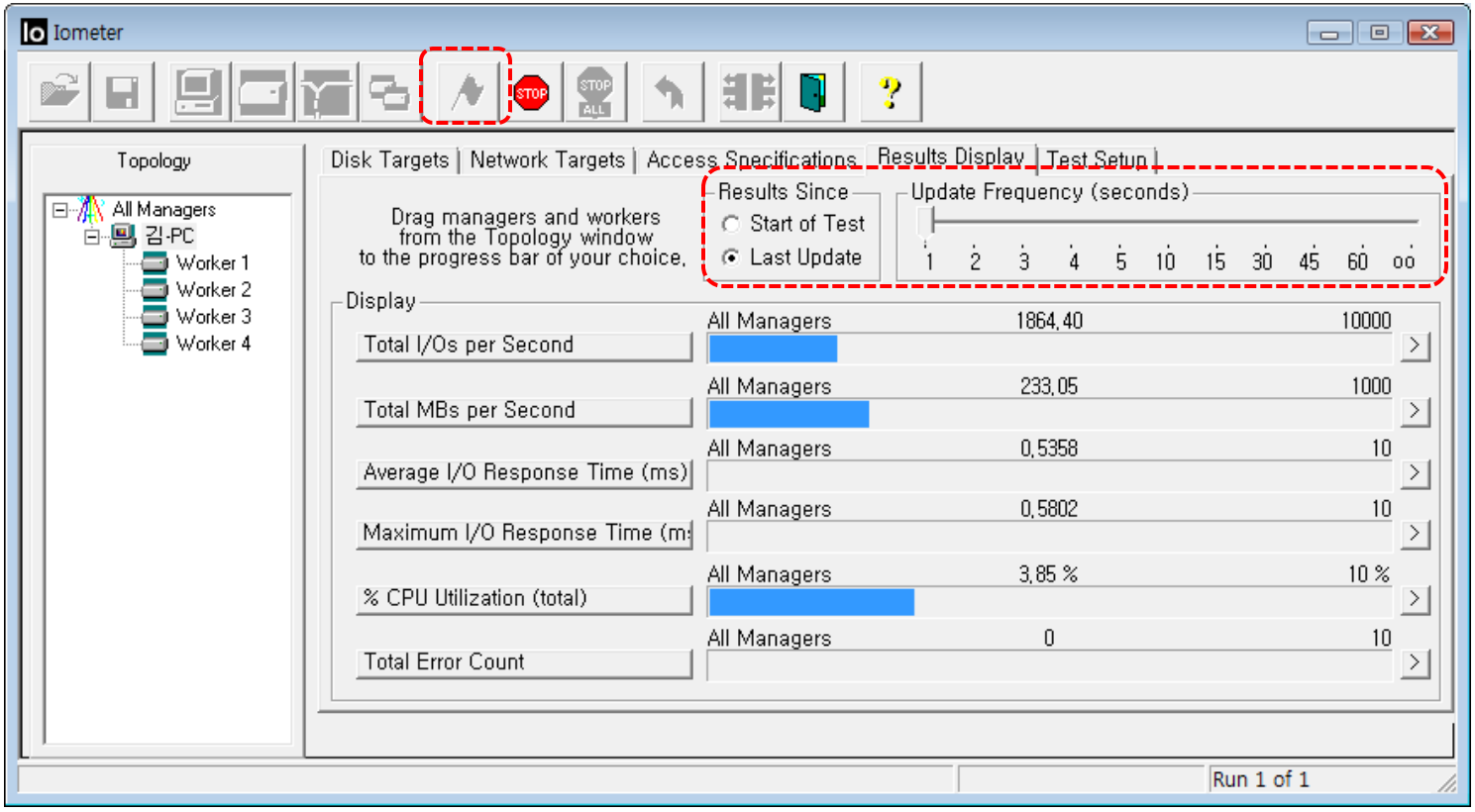
Example

- Assign access specification
  - Select access specification and “Add”

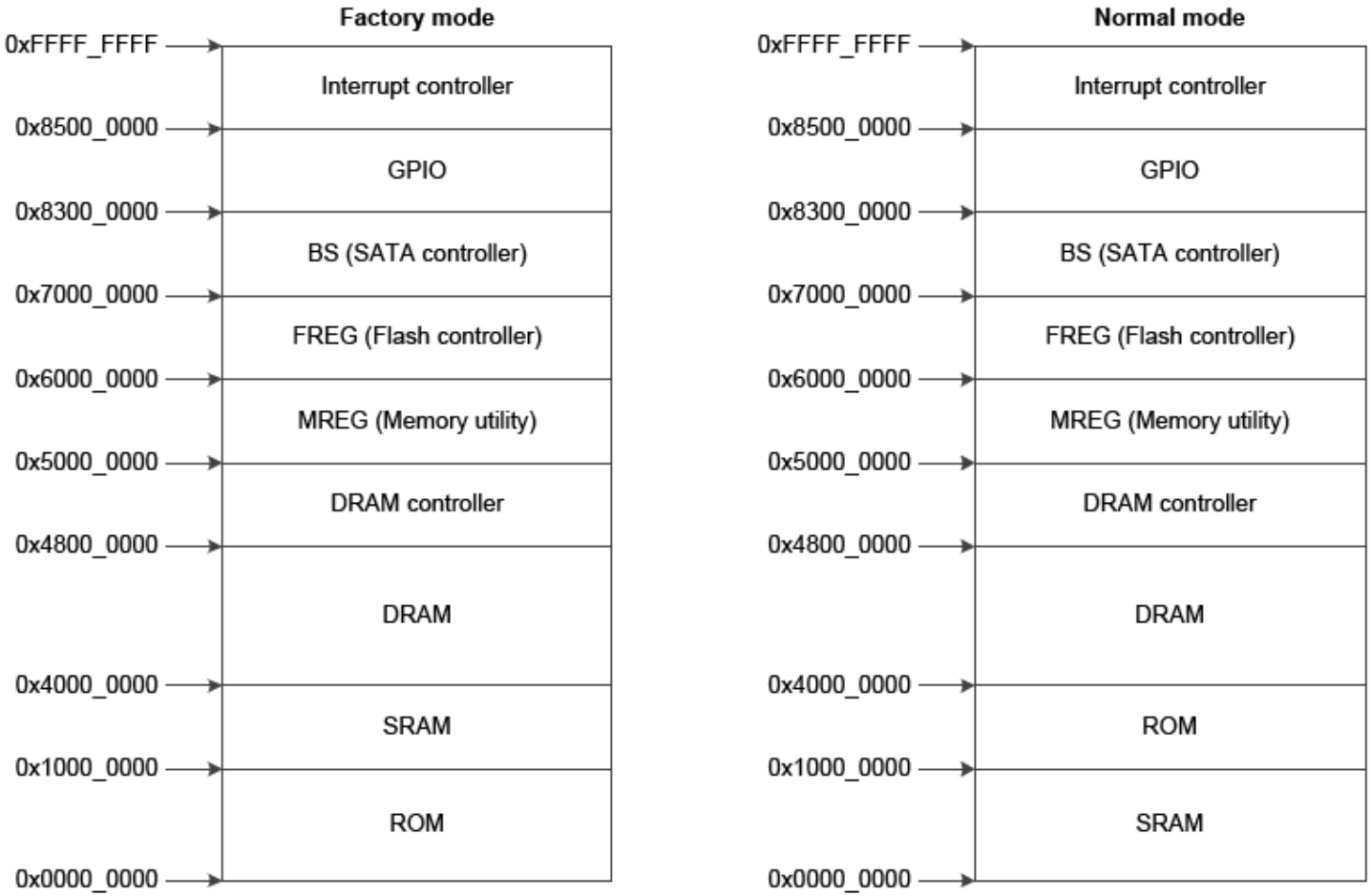


Example

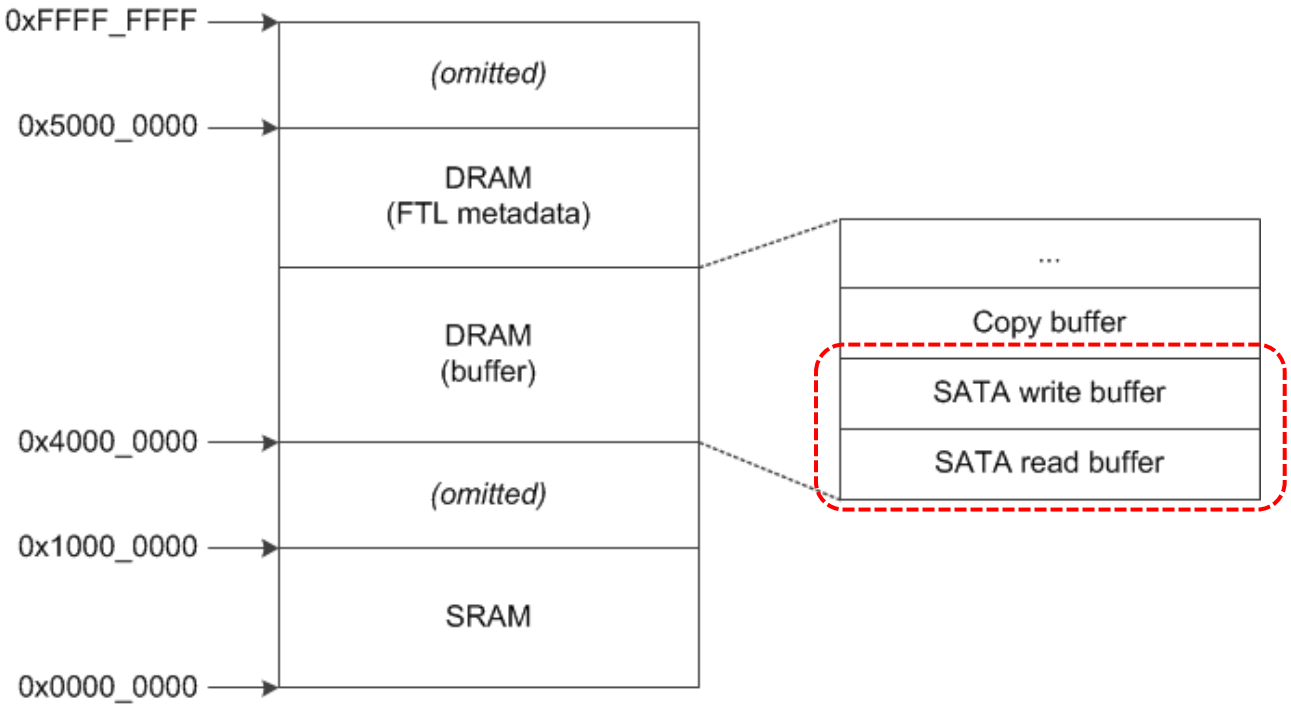
- Start Tests
  - Results Display -> Click 'Flag Icon'



Memory Map


Factory mode
0xFFFF_FFFF
Interrupt controller
0x8500_0000
GPIO
0x8300_0000
BS (SATA controller)
0x7000_0000
FREG (Flash controller)
0x6000_0000
MREG (Memory utility)
0x5000_0000
DRAM controller
0x4800_0000
DRAM
0x4000_0000
SRAM
0x1000_0000
ROM
0x0000_0000
Normal mode
0xFFFF_FFFF
Interrupt controller
0x8500_0000
GPIO
0x8300_0000
BS (SATA controller)
0x7000_0000
FREG (Flash controller)
0x6000_0000
MREG (Memory utility)
0x5000_0000
DRAM controller
0x4800_0000
DRAM
0x4000_0000
ROM
0x1000_0000
SRAM
0x0000_0000

Memory Map


0xFFFF_FFFF
(omitted)
0x5000_0000
DRAM
(FTL metadata)
DRAM
(buffer)
0x4000_0000
(omitted)
0x1000_0000
SRAM
0x0000_0000
...
Copy buffer
SATA write buffer
SATA read buffer