

Dummy FTL & RAM FTL

Prof. Dongkun Shin (dongkun@skku.edu)

TA – Junho Lee (crow6316@skku.edu)

TA – Somm Kim (sommkim@skku.edu)

Embedded Software Laboratory

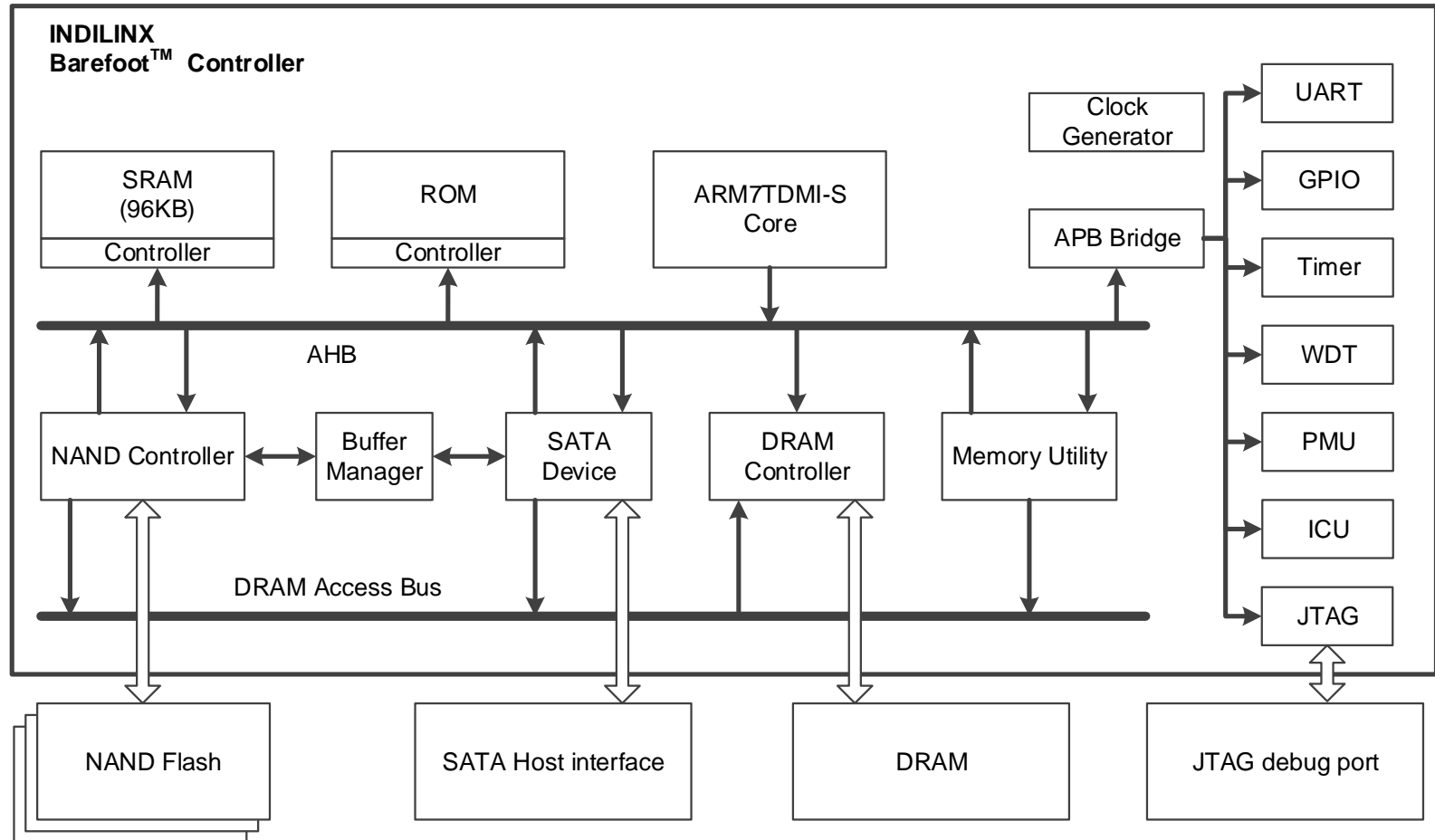
Sungkyunkwan University

<http://nyx.skku.ac.kr>

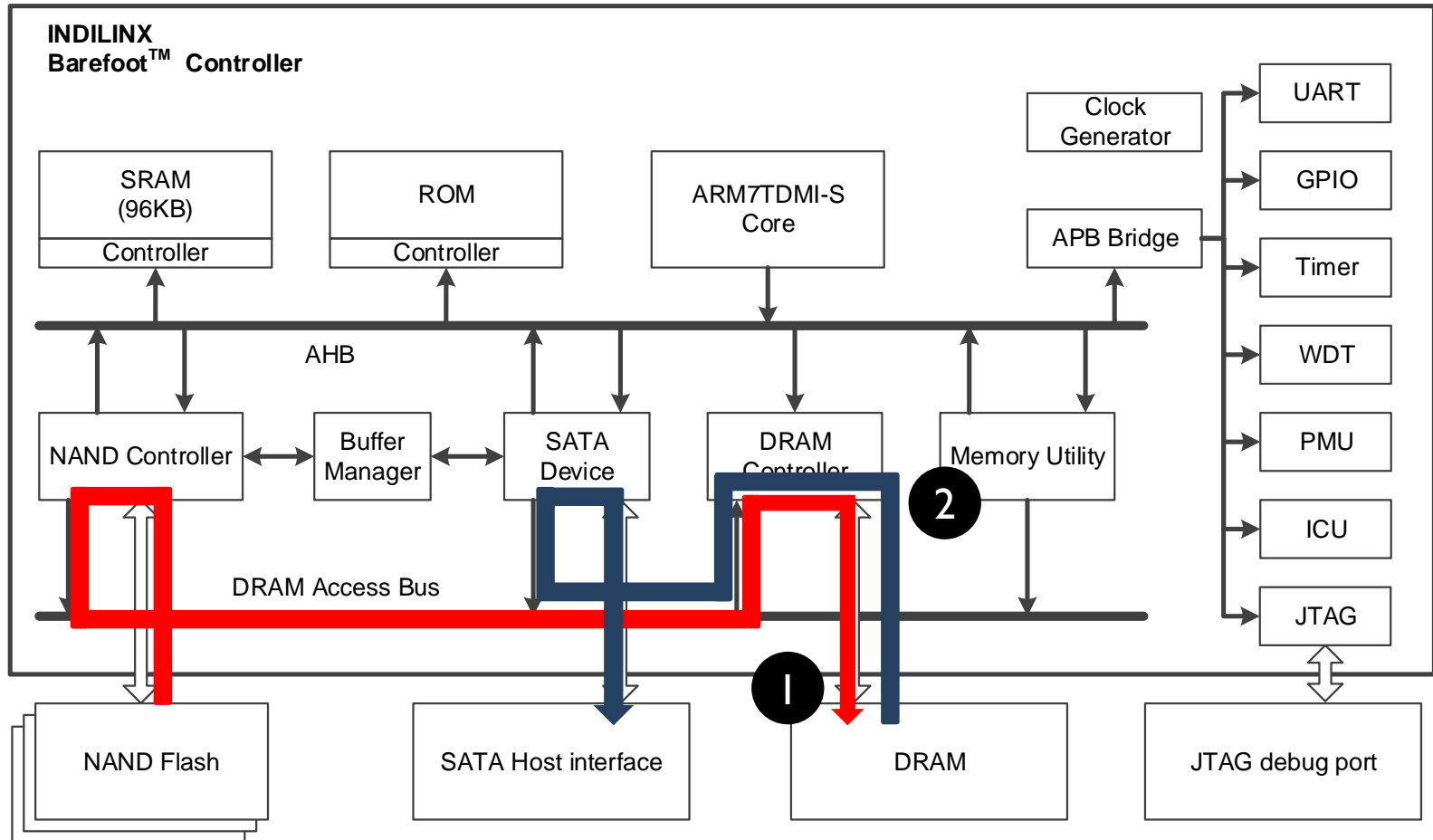
Contents

- Review of Jasmine operation
- Deep dive to the Jasmine & codes
 - Read / write command flow
 - Host interface layer
 - SATA controller
 - Buffer manager
 - Flash translation layer
- Intro. to Dummy FTL
 - Request handling in Dummy FTL (code-level)
- Measuring performance with IOmeter
- RAM FTL

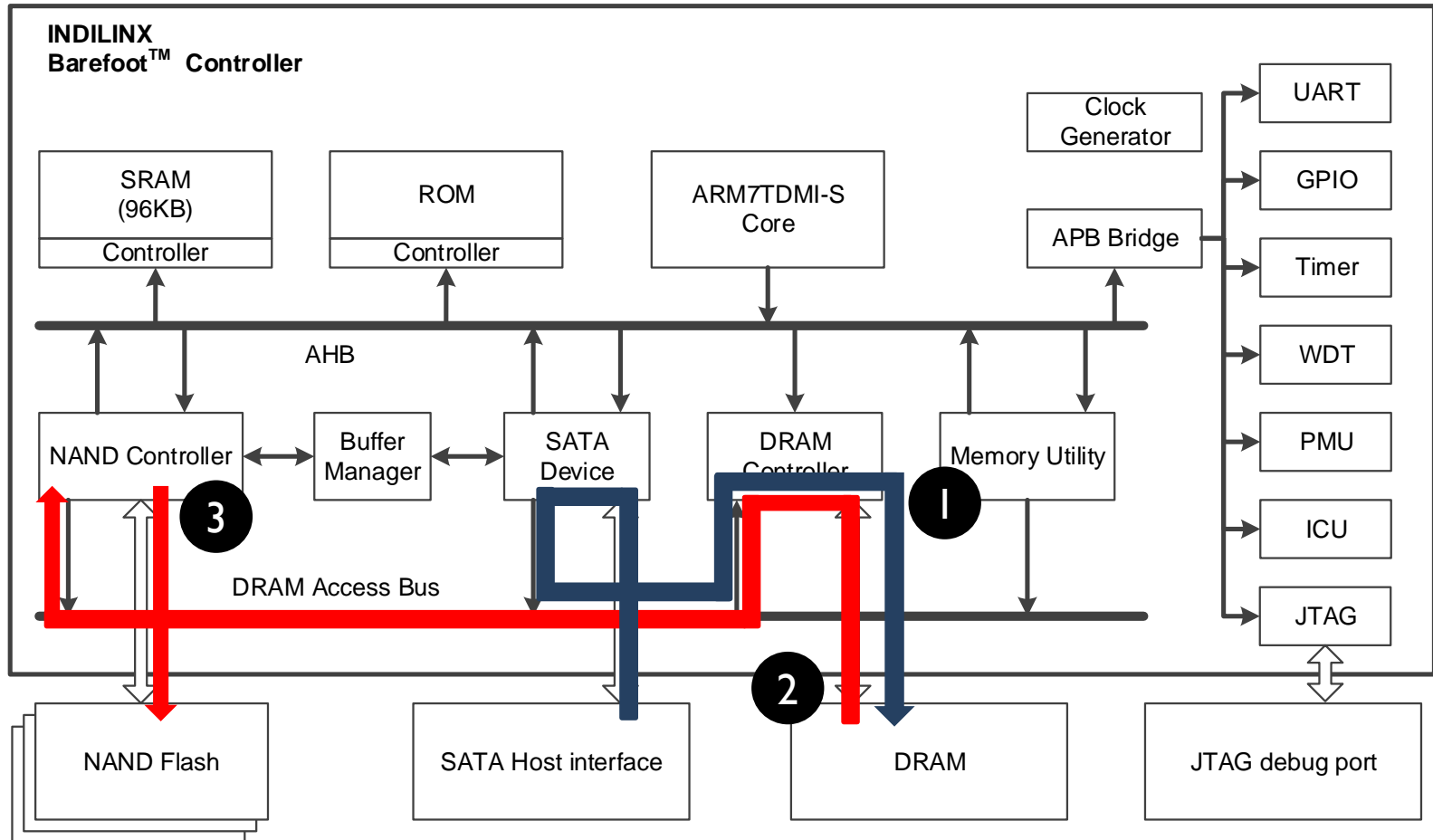
Hardware Architecture



Read Command Flow

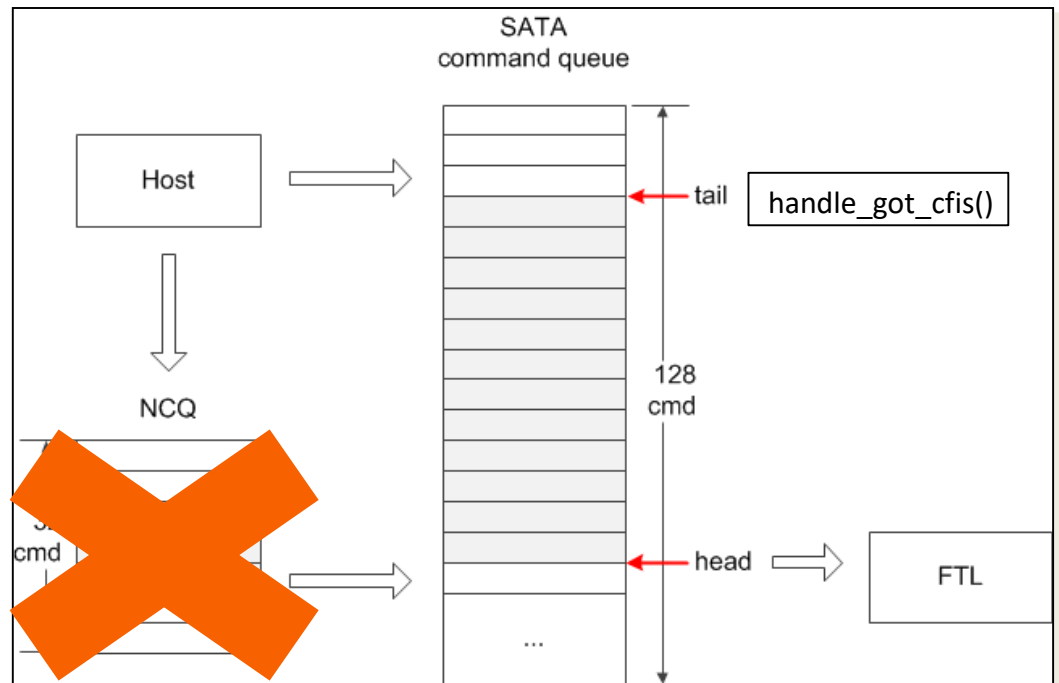


Write Command Flow



SATA Controller

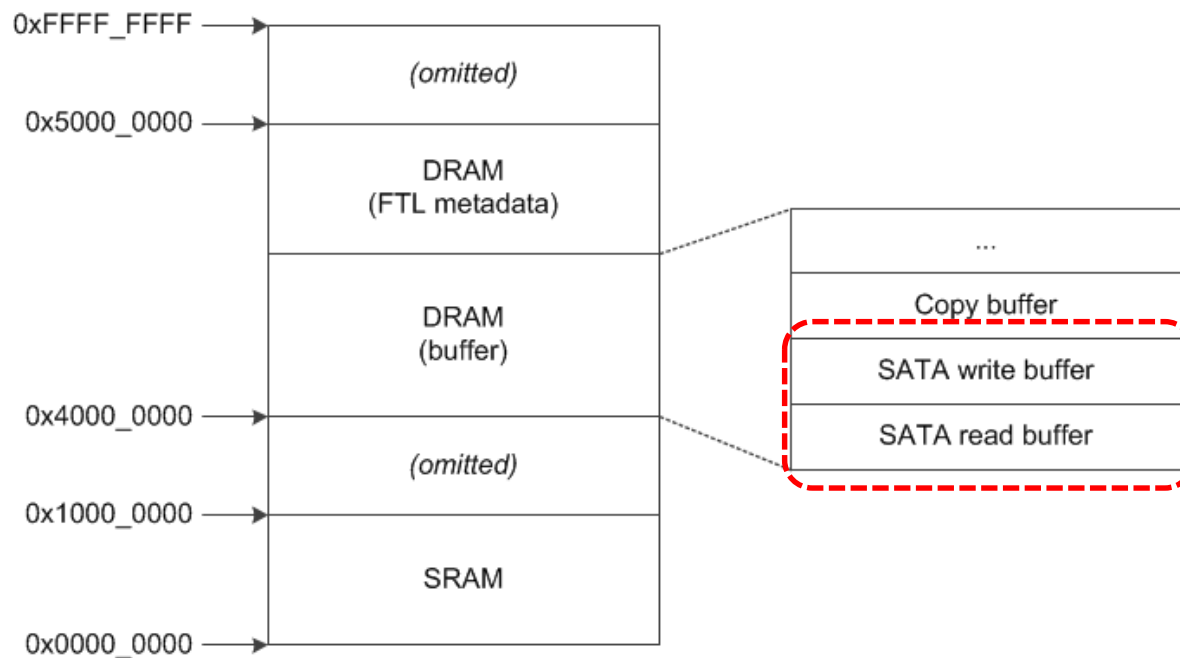
- SATA event queue
 - 128 slots for SATA command
 - Inserted by ISR
 - Deleted by FTL
- Queue policy
 - FIFO
 - Read latency suffers
 - Read first
 - RAW hazard
 - History log by H/W



Disabled in Jasmine

Buffer Manager

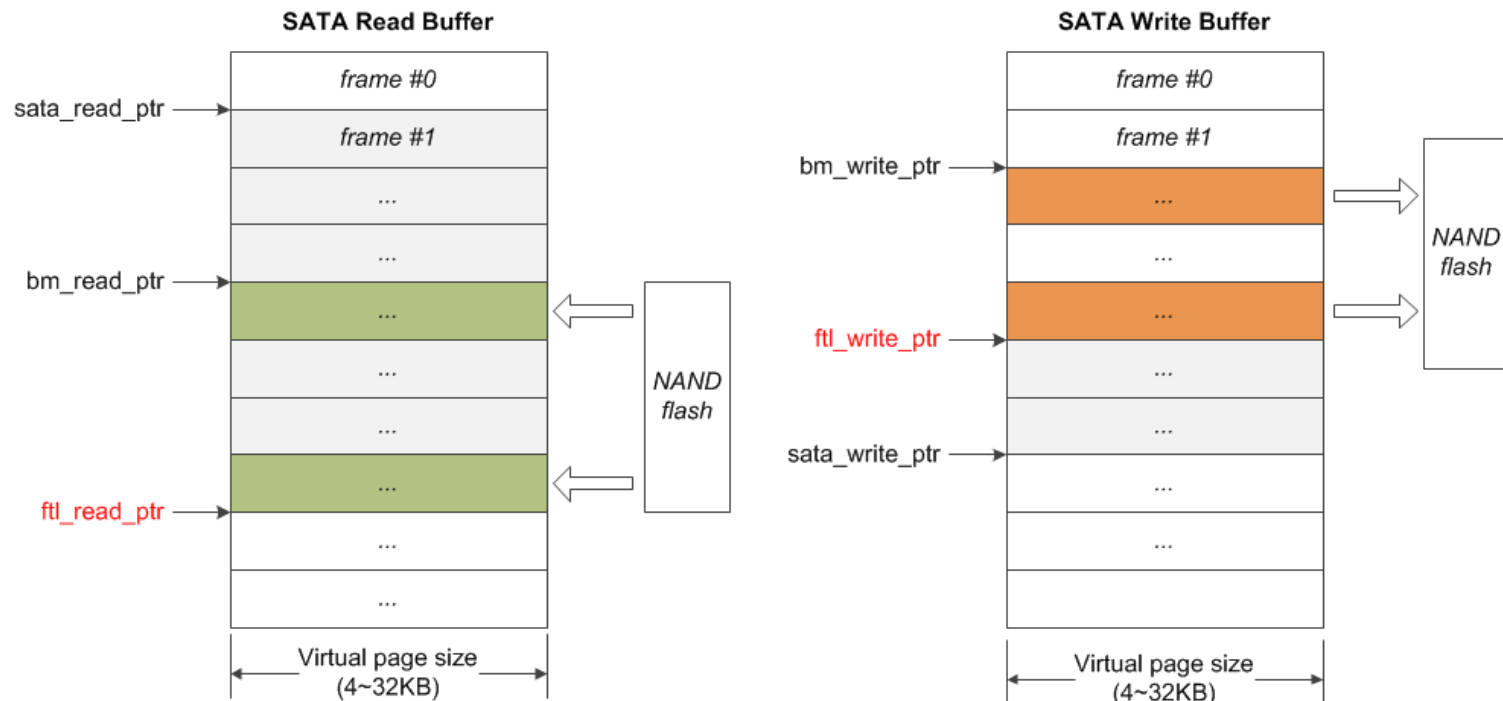
- SATA data is buffered in DRAM
- Memory map of Jasmine board



Buffer Manager (cont'd)

- `ftl_read_ptr`, `ftl_write_ptr`
 - Transfer data from / to NAND
- `sata_read_ptr` / `sata_write_ptr`
 - Transfer data to / from host

Q. Why the order of `bm_ptr` is different in read and write?



Triggering & Initializing FTL

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

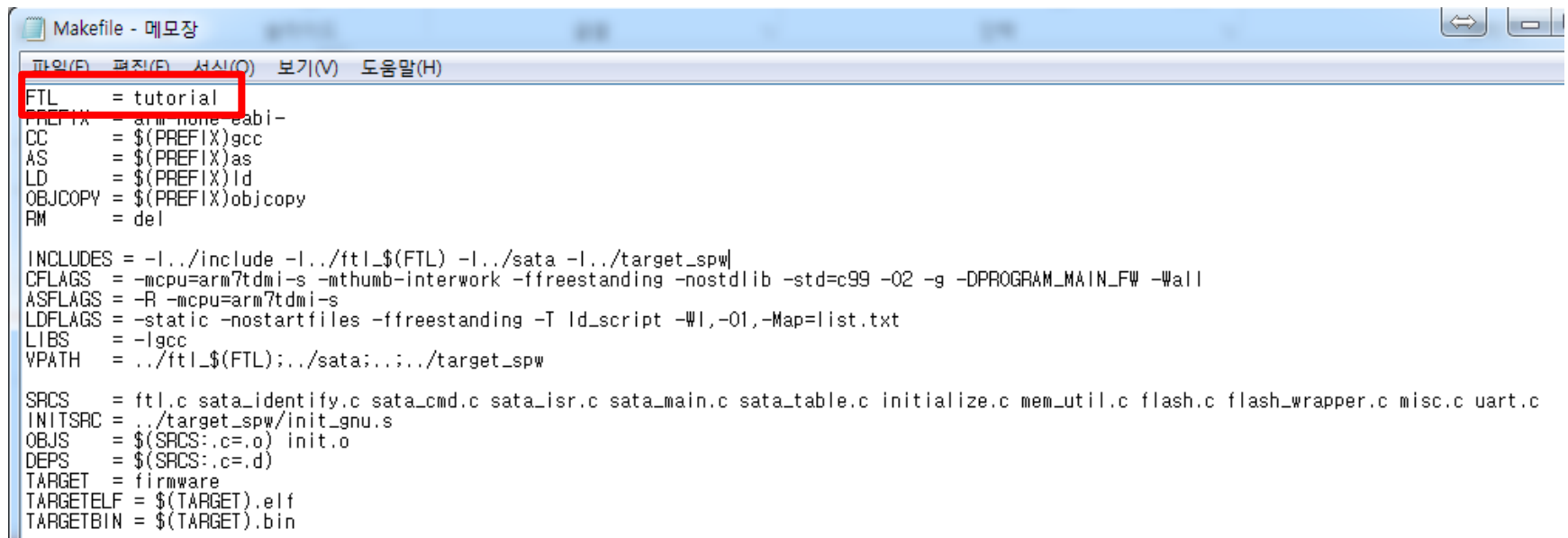
            slow_cmd->status = SLOW_CMD_STATUS_NONE;
        }
        else
        {
            // idle time operations
        }
    }
}
} ? end while 1 ?
} ? end Main ?
```

Dummy FTL

- `./ftl_dummy`
 - `ftl.c, ftl.h`
- Dummy FTL is not a real FTL
 - No access to NAND flash
 - Neither stores nor retrieves any data
- Why Dummy FTL?
 - To simply measure the SATA & DRAM speed

How to Enable Dummy FTL

- `./build_gnu/Makefile`



The screenshot shows a window titled "Makefile - 메모장" (Makefile - Notepad). The menu bar includes "파일(F)", "편집(E)", "서식(O)", "보기(V)", and "도움말(H)". The main text area contains the following Makefile content:

```
FTL = tutorial
PREFIX = arm-none-eabi-
CC = $(PREFIX)gcc
AS = $(PREFIX)as
LD = $(PREFIX)ld
OBJCOPY = $(PREFIX)objcopy
RM = del

INCLUDES = -I../include -I../ftl_$(FTL) -I../sata -I../target_spw
CFLAGS = -mcpu=arm7tdmi-s -mthumb-interwork -ffreestanding -nostdlib -std=c99 -O2 -g -DPROGRAM_MAIN_FW -Wall
ASFLAGS = -R -mcpu=arm7tdmi-s
LDFLAGS = -static -nostartfiles -ffreestanding -T ld_script -Wl,-O1,-Map=list.txt
LIBS = -lgcc
VPATH = ../ftl_$(FTL);../sata;../target_spw

SRCS = ftl.c sata_identify.c sata_cmd.c sata_isr.c sata_main.c sata_table.c initialize.c mem_util.c flash.c flash_wrapper.c misc.c uart.c
INITSRC = ../target_spw/init_gnu.s
OBJS = $(SRCS:.c=.o) init.o
DEPS = $(SRCS:.c=.d)
TARGET = firmware
TARGETELF = $(TARGET).elf
TARGETBIN = $(TARGET).bin
```

Dummy FTL: Read Handling

- ./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 ?
```

Dummy FTL: Write Handling

- ./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 ?
```

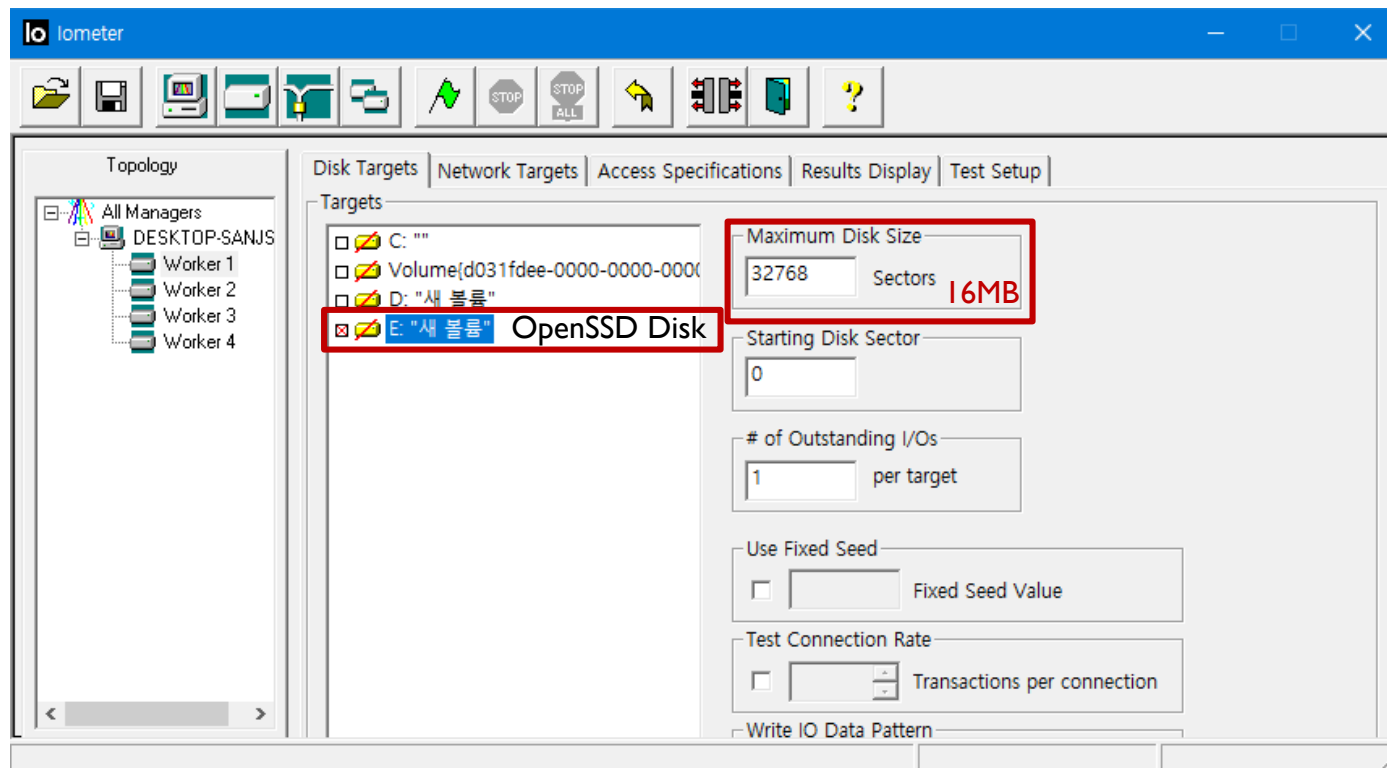
Measuring performance with IOmeter

Iometer

- Performance measurement tool for storage
 - <http://www.iometer.org>
- Performance factors
 - IOPS (IOs Per Second)
 - Bandwidth (MB/s)
 - Response time (also known as latency)

IOmeter

- Select disk target



IOmeter

- Make new access specification
 - Access Specifications -> New

Edit Access Specification

Name: Sequential Write 16KB

Default Assignment: None

Size	% Access	% Read	% Random	Delay	Burst	Alignment	Reply
0 MiB 16 KiB 0 B	100	0	0	0	1	0 MiB 16 KiB ...	none

Transfer Request Size: 0 Megabytes, 16 Kilobytes, 0 Bytes

Percent of Access Specification: 100 Percent

Percent Read/Write Distribution: 100% Write, 0% Read

Percent Random/Sequential Distribution: 100% Sequential, 0% Random

Burstiness: Transfer Delay 0 ms, Burst Length 1 I/Os

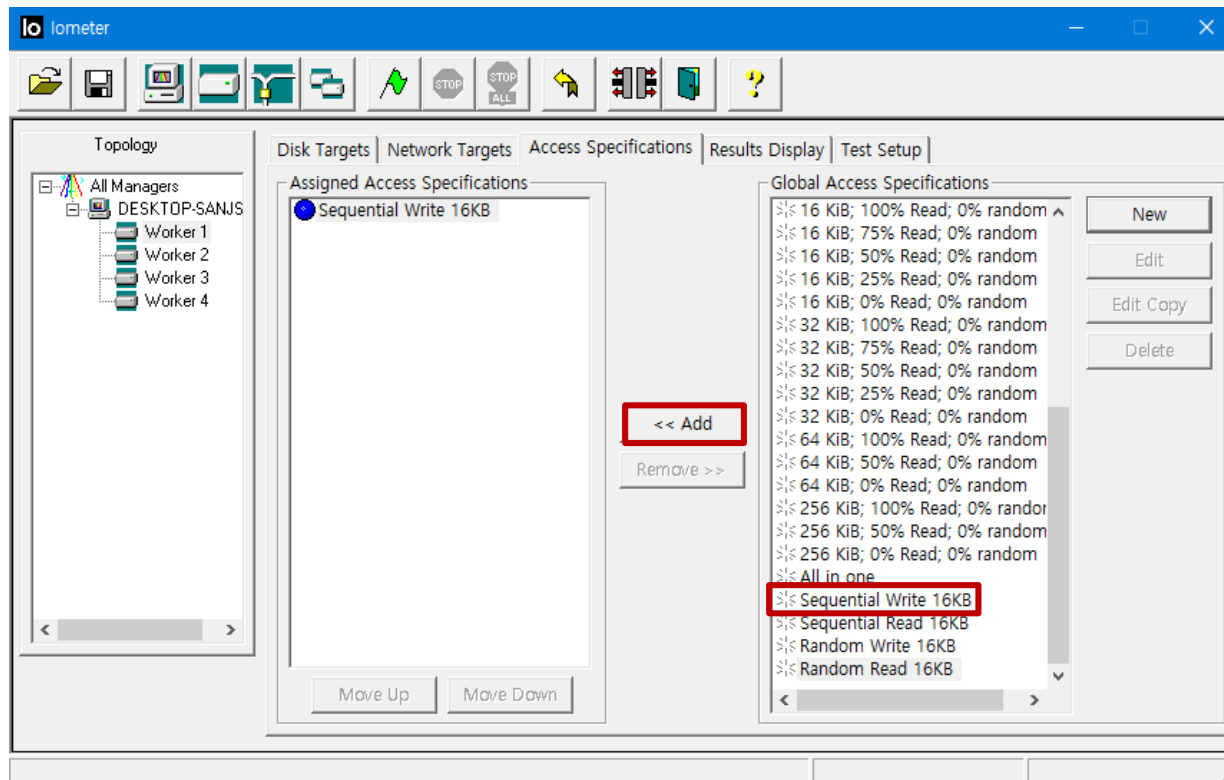
Align I/Os on: Request Size Boundaries, Sector Boundaries

Reply Size: No Reply, 0 Megabytes, 16 Kilobytes, 0 Bytes

OK Cancel

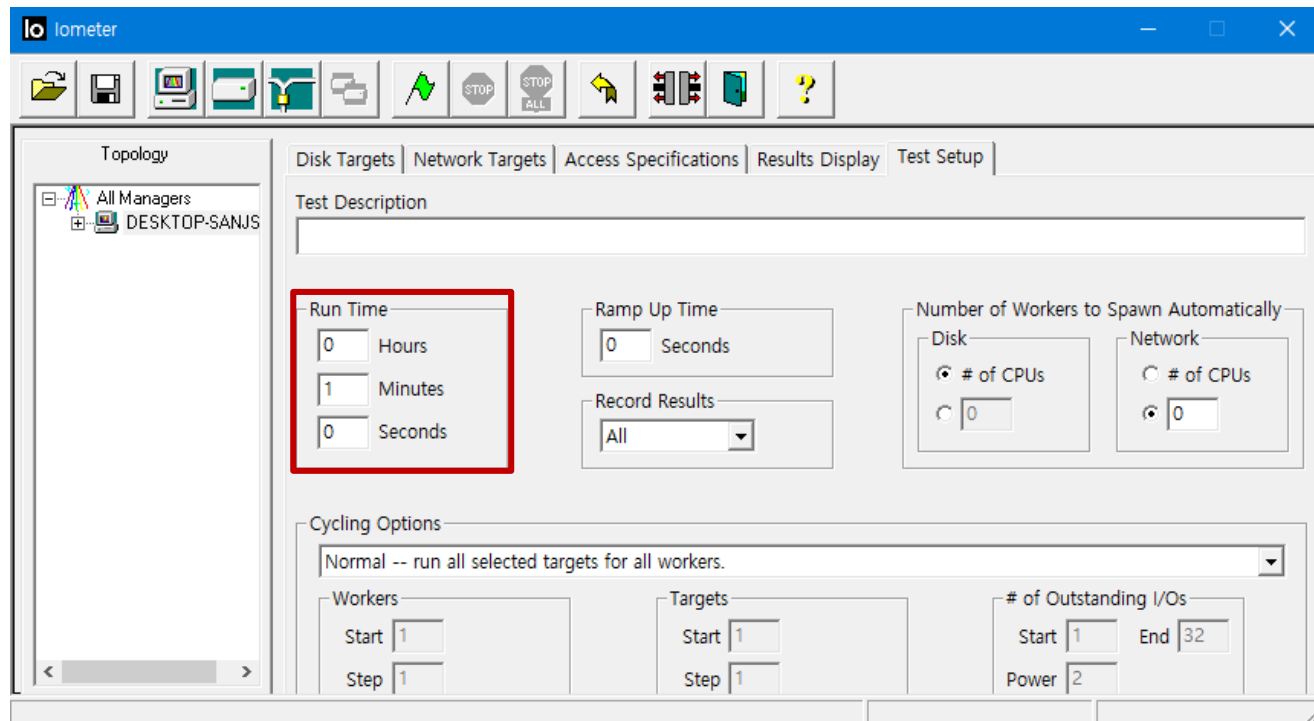
IOmeter

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



IOmeter

- Assign Run Time



IOmeter

- Start Tests
 - Result Display -> Click “Flag Icon”

The screenshot shows the IOmeter software interface. The window title is "io iometer". The toolbar contains several icons, with a green flag icon highlighted by a red box. The main window is divided into several sections:

- Topology:** A tree view showing "All Managers" and "DESKTOP-SANJS" with four workers (Worker 1, Worker 2, Worker 3, Worker 4).
- Results Display:** The active tab, showing performance metrics. A red box highlights the "Update Frequency (seconds)" dropdown menu, which is set to "1".
- Display:** A table of performance metrics with progress bars and target values.

Metric	Value	Target
Total I/Os per Second	9544.88	10000
Total MBs per Second (Decimal)	156.38 MBPS (149.14 MiBPS)	1000
Average I/O Response Time (ms)	0.1044	1
Maximum I/O Response Time (ms)	5.8530	10
% CPU Utilization (total)	12.59 %	100 %
Total Error Count	0	0

Test Completed Successfully

RAM FTL (Jasmine)

Lab 3 : RAM FTL

- Develop a 32MB disk with dummy FTL (OpenSSD-1.1.0\ftl_dummy)
 - It can be used like a normal disk.
 - Modify `ftl_read()`, `ftl_write()` functions to read/write data using Jasmine DRAM.
 - Hint: See the '#define' of `ftl.h`

ftl_read(), ftl_write()

- **ftl_read(*lba*, *total_sectors*)**
 - Description
 - Read the data corresponding to 'lba' and 'total_sectors' from the DRAM.
 - Hint: Set the offset of the `g_ftl_read_buf_id` pointer and DRAM address
- **ftl_write(*lba*, *total_sectors*)**
 - Description
 - Write the data corresponding to 'lba' and 'total_sectors' to the DRAM.
 - Hint: Set the offset of the `g_ftl_write_buf_id` pointer and DRAM address
- **Note: Use the memory utility function to transfer data between DRAM and SRAM.**

Miscellaneous

- Recommended environment :Windows
- Team Project
- You should submit a report
 - Use an IOmeter to measure performance
 - Sequential Write 16KB, Sequential Read 16KB, Random Write 16KB, Random Read 16KB
 - Capture and describe the performance comparison of Dummy FTL and RAM FTL (Direct connection to SATA, Not through a USB)
- Submit to the icampus
 - Due: 10/7(Mon.) 23:59:59
 - File to submit: ftl.c, ftl.h, jasmine.h, report.pdf
 - File name: team_ \$NUMBER.zip (ex. team_01.zip)
- Late penalty : -20% per day (Up to 3 days)

Any Questions?