Cosmos OpenSSD Platform Tutorial

Cosmos OpenSSD

ENC Lab. @ Hanyang University

HANYANG UNIVERSITY
ENC Lab
Embedded Network Computing Lab.

What is the OpenSSD Project

- Open-source SSD platform for research and education on the SSD technology since 2011
  - Jasmine OpenSSD
  - Cosmos OpenSSD
- New "OpenSSD platform" for developing SSD firmware, controller hardware, and host software
- Contribution
  - Indilinx (Merged to OCZ in 2012)
  - HYU (Hanyang University), Korea
  - SKKU (Sungkyunkwan University), Korea

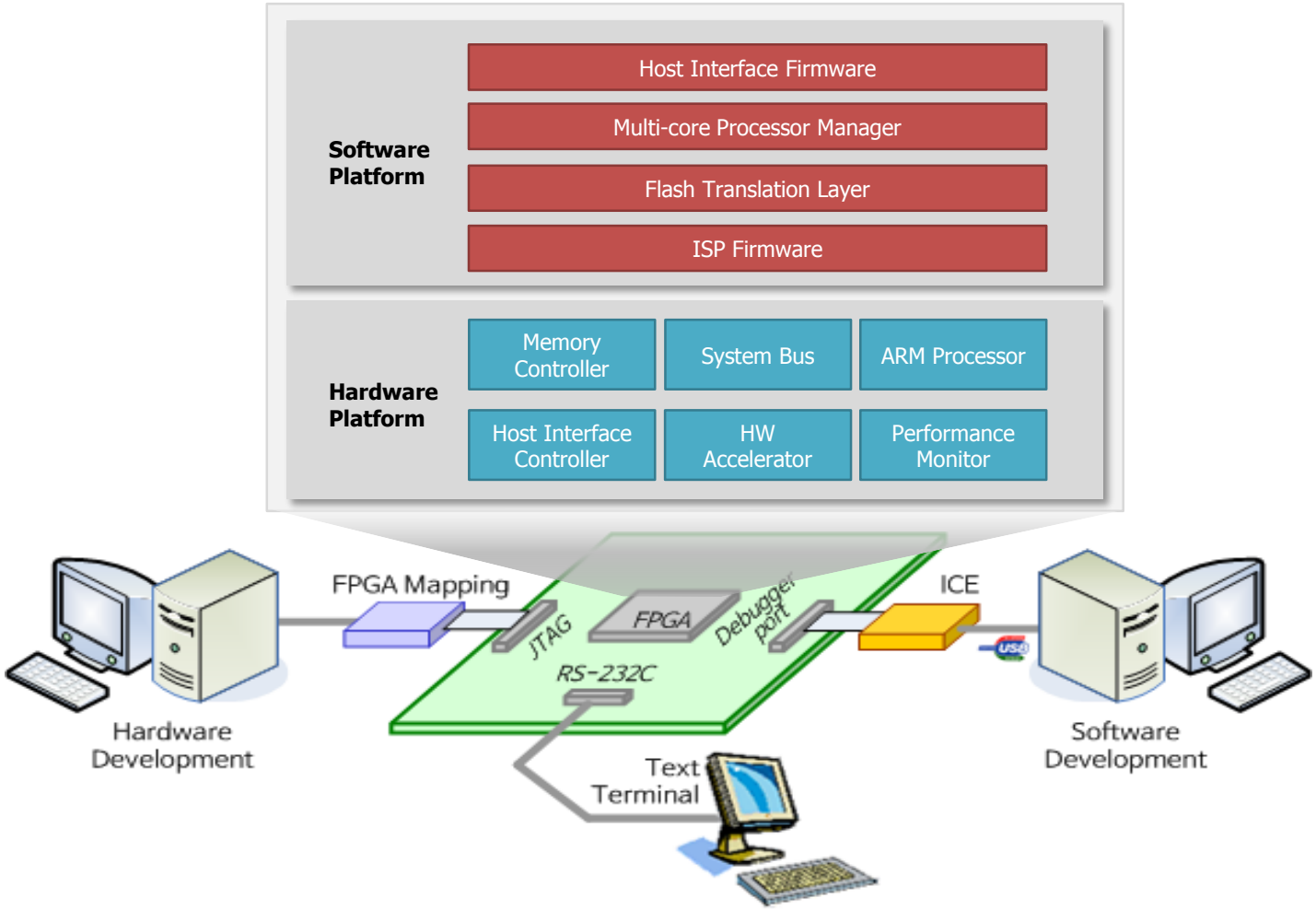
Why OpenSSD

- Solve your problem in a real system
- Share your solution with people in society
- Design your own SSD controller, if possible
- Contribute to "open" community
- Use it as a PC disk
- Play for fun

OpenSSD Platform Development Co-design

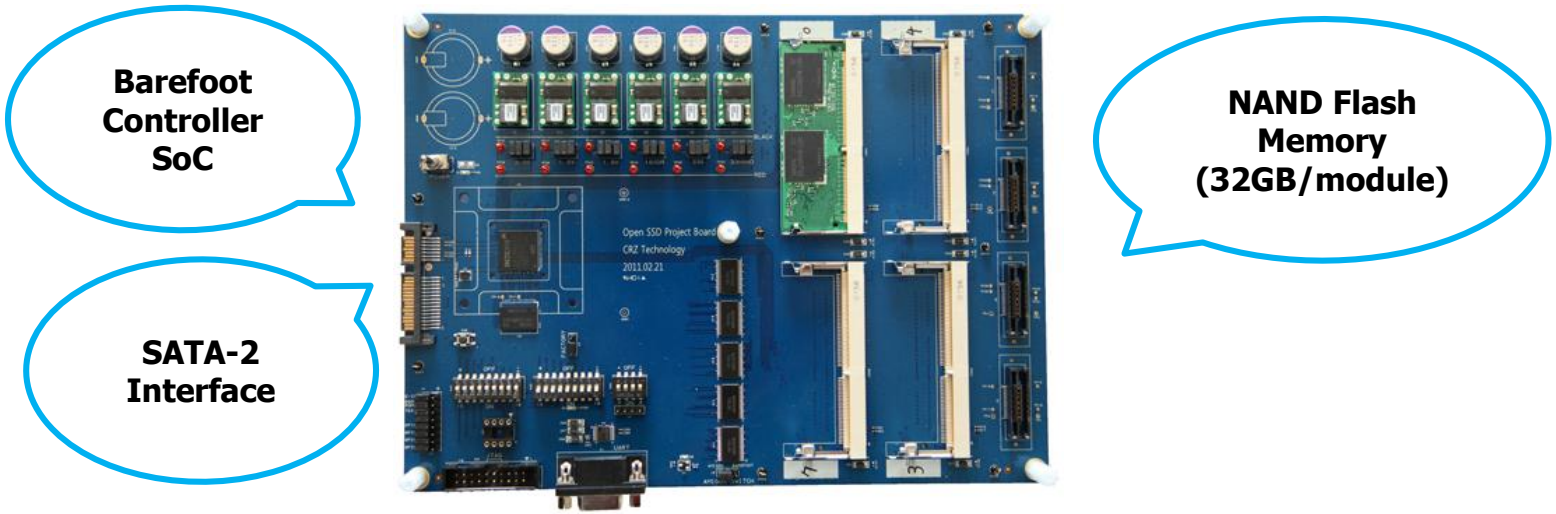
Integrated verification of hardware and software IP,

e.g. ISP hardware accelerator design, verification, and software development



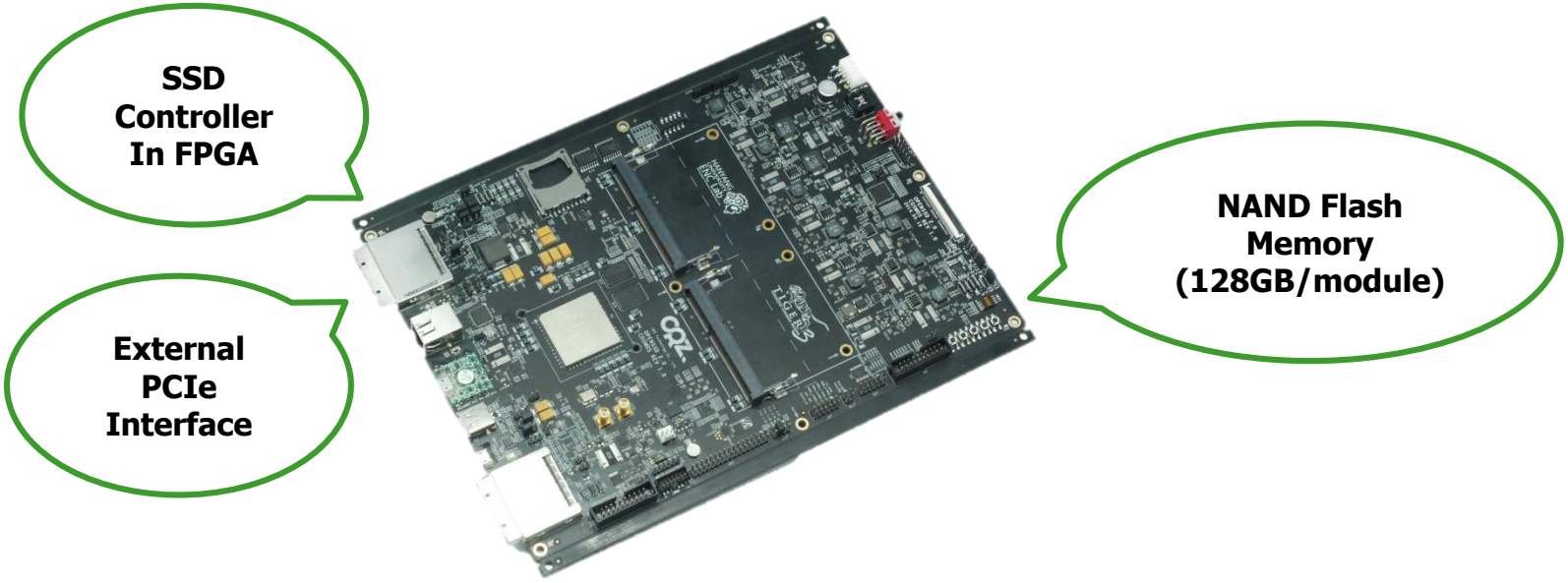
OpenSSD Project History [1/2]

- Jasmine OpenSSD (2011)
  - SSD controller: Indilinx Barefoot (SoC w/SATA-2)
  - Firmware: SKKU VLDB Lab
  - Users from 10+ countries
  - 10+ papers published



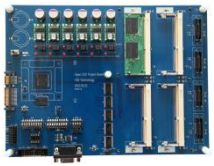
OpenSSD Project History [2/2]

- Cosmos OpenSSD (2014)
  - SSD controller: HYU Tiger3 (FPGA w/PCIe Gen2)
  - Firmware: HYU ENC Lab
  - Users from ?? countries (at least one)
  - ?? papers to be published (at least three)



Comparison among the Platforms

	Jasmine OpenSSD	Cosmos Prototype (Tiger2)	Cosmos OpenSSD (Tiger3)
SSD Controller	Indilinx Barefoot (SoC)	HYU Tiger2 (FPGA)	HYU Tiger3 (FPGA)
Year	2011	2012	2014
Host Interface	SATA-2	PCIe Gen1.1 (AHCI Subset)	PCIe Gen2 (AHCI Subset)
Storage Capacity	128 GB	512 GB	256 GB
NAND Data Interface	Asynchronous	Asynchronous	Synchronous
DRAM Capacity	64 MB	512 MB	1 GB



OpenSSD Project Homepage

http://www.openssd-project.org

Cosmos OpenSSD Platform Tutorial

Cosmos OpenSSD Platform Board

ENC Lab. @ Hanyang University

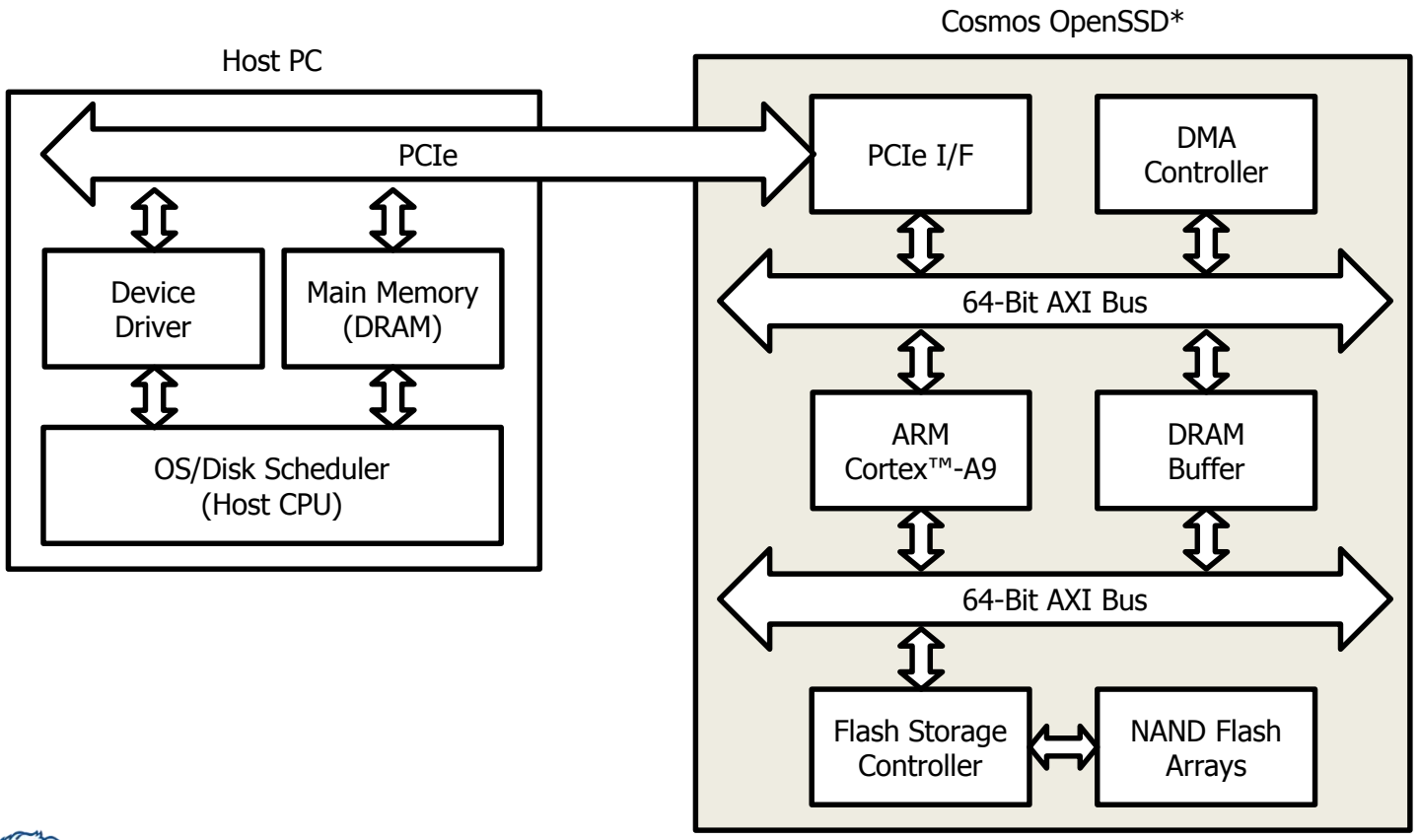
HANYANG UNIVERSITY
ENC Lab
Embedded Network Computing Lab.

Document Revision History

- v1.0.0
  - Initial release
  - Contents are based on platform board_Rev 1.1

Cosmos OpenSSD Platform in a Nutshell

- Cosmos OpenSSD is a storage device whose hardware and software can be freely modified
  - Use of external PCIe as a host interface



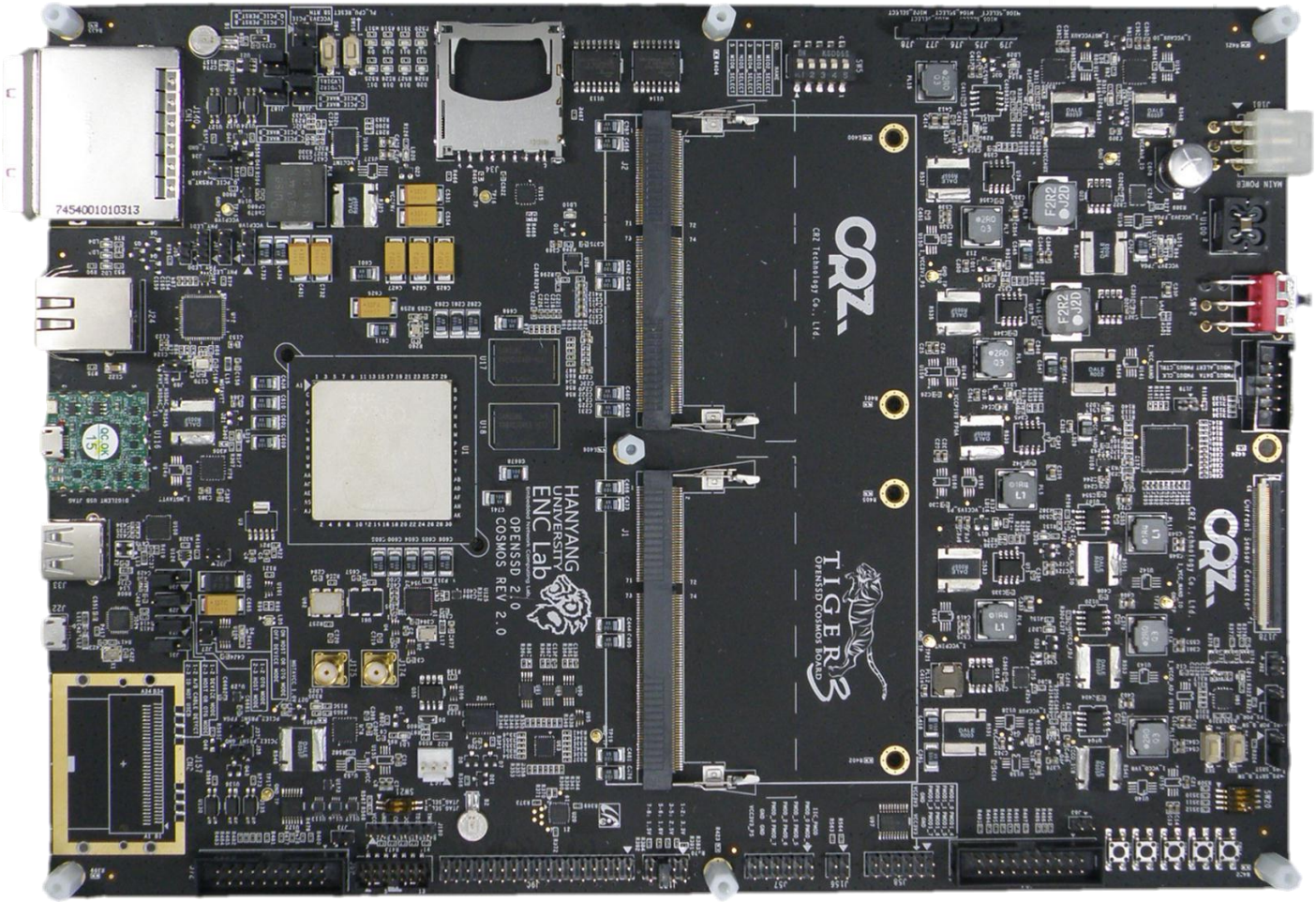
*also called 'storage device' or 'device'

INDEX

Platform Board Overview

HANYANG UNIVERSITY
ENC Lab
Embedded Network Computing Lab.

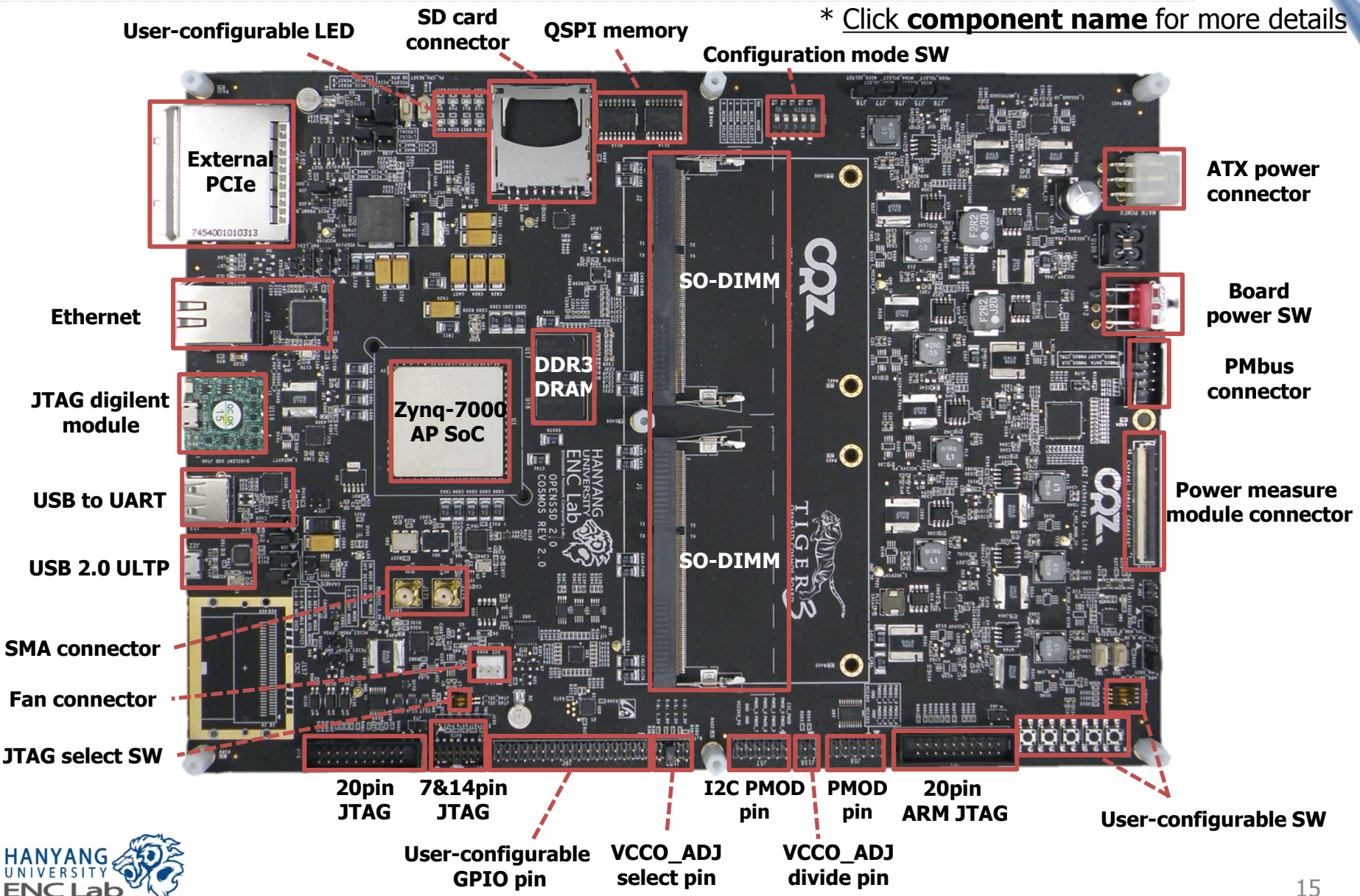
Cosmos OpenSSD Platform Board (Front-side)



HANYANG UNIVERSITY ENC Lab
Embedded Network Computing Lab.

Components of Cosmos OpenSSD Platform Board

* Click **component name** for more details



Primary Details

FPGA		Xilinx Zynq-7000 AP SoC (XC7Z045-FFG900-3)
Logic cells		350K (~ 5.2M ASIC gates)
CPU	**Type**	Dual-Core ARM Cortex™- A9
	Clock frequency	667 MHz
Storage	**Total capacity**	128 GB / DIMM
	Organization	4channel-4way / DIMM
DRAM	**Device interface**	DDR3 (533 MHz)
	Total capacity	1 GB
Bus	**System**	AXI-Lite (bus width: 32 bits)
	Storage data	AXI (bus width: 64 bits, burst length: 16)
SRAM		256 KB (FPGA internal)
Power measurement		Flash module and board power measurement (need external ADC module)

Power Configurations

Power-on Sequence

1. Install a PCIe adapter on a host PC (running the Linux operating system)
2. Attach the platform board to the PCIe adapter using a PCIe cable
3. Power on the platform board before you turn on the host PC
4. Then, start to boot the host PC

   Note: host PC should be powered on after FPGA configuration is completed

Power Source

- Single-source of power to the platform board
  - Part number: J181

- Cosmos power connector similar to the regular ATX power connector

  Note: Difference in pin assignment between two connectors

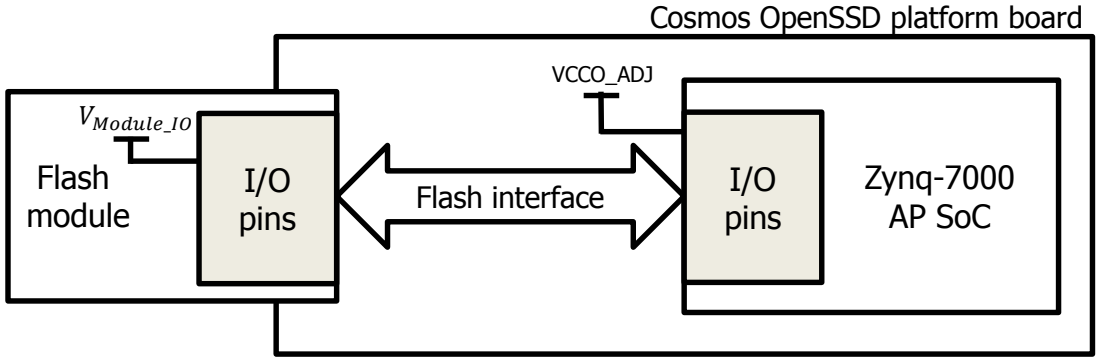
Connector	ATX 6pin					
	1	2	3	4	5	6
Platform board ATX	12V	12V	NC	NC	GND	GND
PC ATX	GND	GND	GND	12V	12V	12V

- Caution
  - Do not plug PC ATX power 6-pin connector to J181

Voltage Selection

- Used to select a flash interface voltage (VCCO_ADJ and V_{Module_IO})
  - VCCO_ADJ: supply voltage for FPGA I/O pins connected with flash module
  - V_{Module_IO}: supply voltage for I/O pins of flash module

    Note: V_{Module_IO} is dependent on flash module



VCCO_ADJ select pin (J177) setting

Section	J177			
	1-2	3-4	5-6	7-8
VCCO_ADJ voltage	3.3V	2.5V	1.8V (default)	1.5V

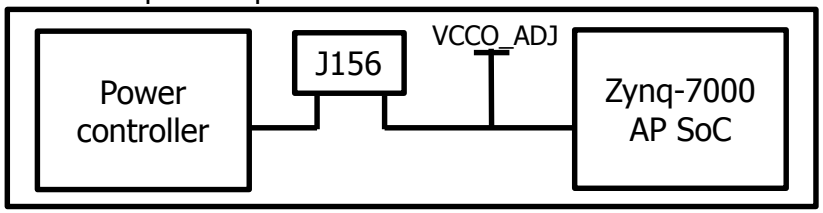
- Caution
  - Do not remove the shunt away

Power Management [1/2]

- Power-on sequence must comply with the requirement of Zynq-7000 AP SoC
  - Refer to Xilinx data sheet (DS191)
  - The sequence is adjusted by the power controller

- Power controller (U102)
  - Manage power-on sequence
    - Power-on sequence of power supplies is pre-set by manufacturer
    - If you want to change the power-on sequence, use "Fusion Digital Power Designer"
  - Check the voltage of each power supply
    - Input voltage of power controller should be less than 2.5V
    - When VCCO_ADJ is 3.3V, user should reduce input voltage of power controller

Cosmos OpenSSD platform board



VCCO_ADJ divide pin (J156) setting

Position	Description
1-2	Reduce 3.3V to 2V
3-4	NC

  - Part number : UCD90120ARGC (Texas Instruments)

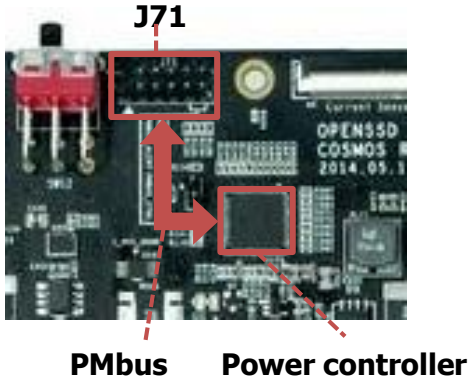
Power Management [2/2]

- Power Management (PM) bus
  - Open standard digital power management protocol
  - Use pre-defined command language and physical interface

- PMbus connector (J71)

Pin #	Net name	Pin #	Net name
7	PMBUS_CTRL	8	PMBUS_ALERT
9	PMBUS_CLK	10	PMBUS_DATA

* Pin 1-6: uncoupled

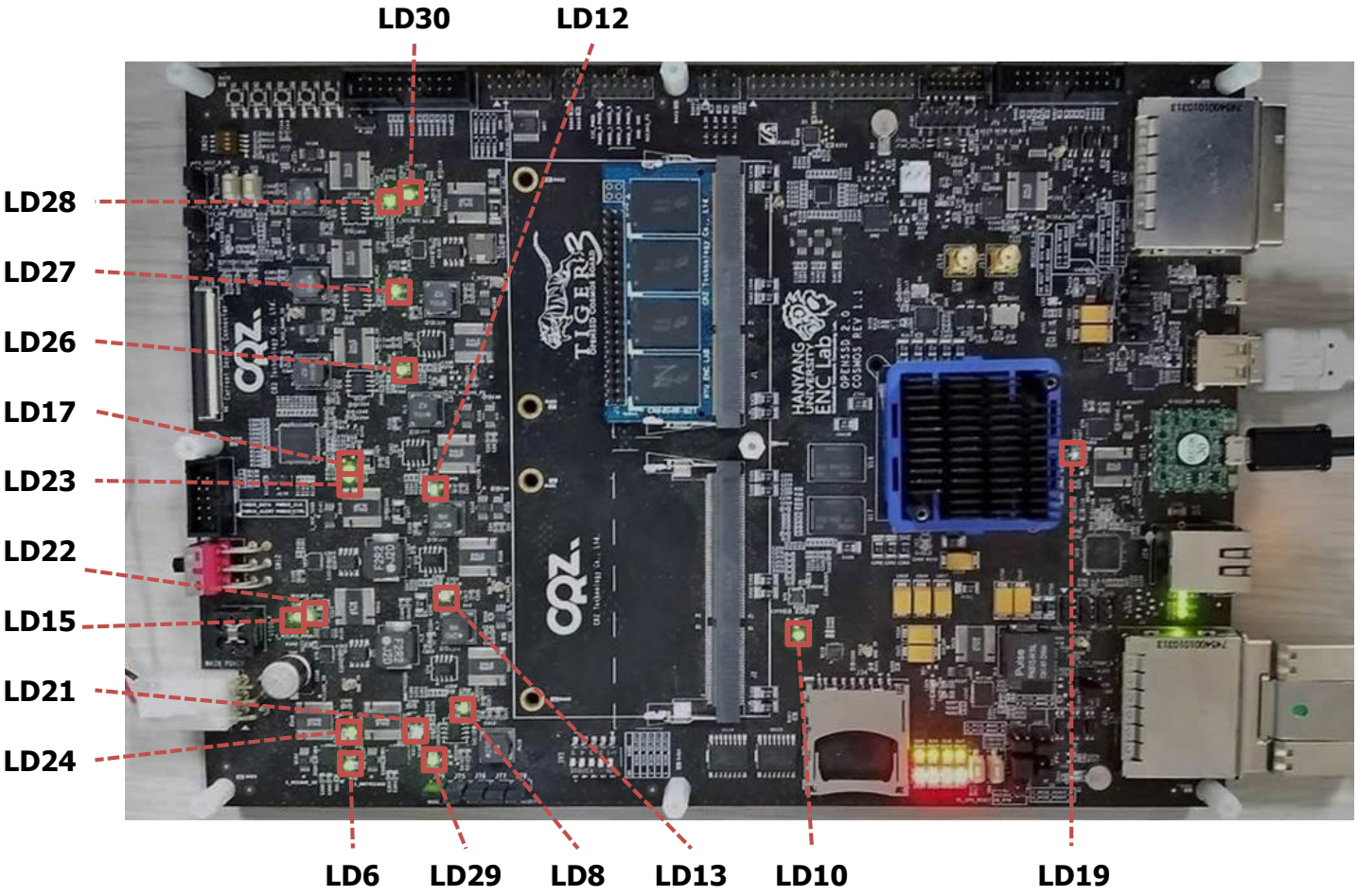


Power Related LEDs

	LED	Net Name	Description
Zynq PS power supplies	LD8	VCC3V3_PS	Supply voltage for PS peripheral devices
	LD10	PS_DDR_LINEAR_PG	Check whether VTT(for DDR) is stable
	LD12	VCC1V5_PS	Supply voltage for FPGA pins connected with DRAM
	LD13	VCCP1V8_FPGA	PS auxiliary, PS PLL supply voltage
GTX Transceiver power supplies	LD19	MGTAVTT	Analog supply voltage for the GTX transceivers termination circuits
	LD21	MGTVCCAUX	Auxiliary analog Quad PLL (QPLL) voltage supply for the GTX transceivers
Zynq PL power supplies	LD22	VCC3V3_FPGA	Supply voltage for configuration bank
	LD24	VCCAUX_IO	Auxiliary supply voltage
	LD27	VCCO_ADJ	Supply voltage for FPGA pins connected with flash module
	LD28	VCCO_1V8	PL supply voltage for HP I/O banks
	LD30	VCCAUX	PL auxiliary supply voltage
Flash power supplies	LD23	VCC_NAND	Supply voltage for flash module
	LD26	VCC_NAND_IO	Supply voltage for flash package IO
Miscellany	LD6	LINEAR_POWER_GOOD	Check whether power supplies are stable
	LD15	VCC12_P	Main supply voltage of Cosmos OpenSSD platform board
	LD17	CTRL1_PWRGOOD	Check whether power controller is operating well
	LD29	VCC5V0	Supply voltage for peripheral devices

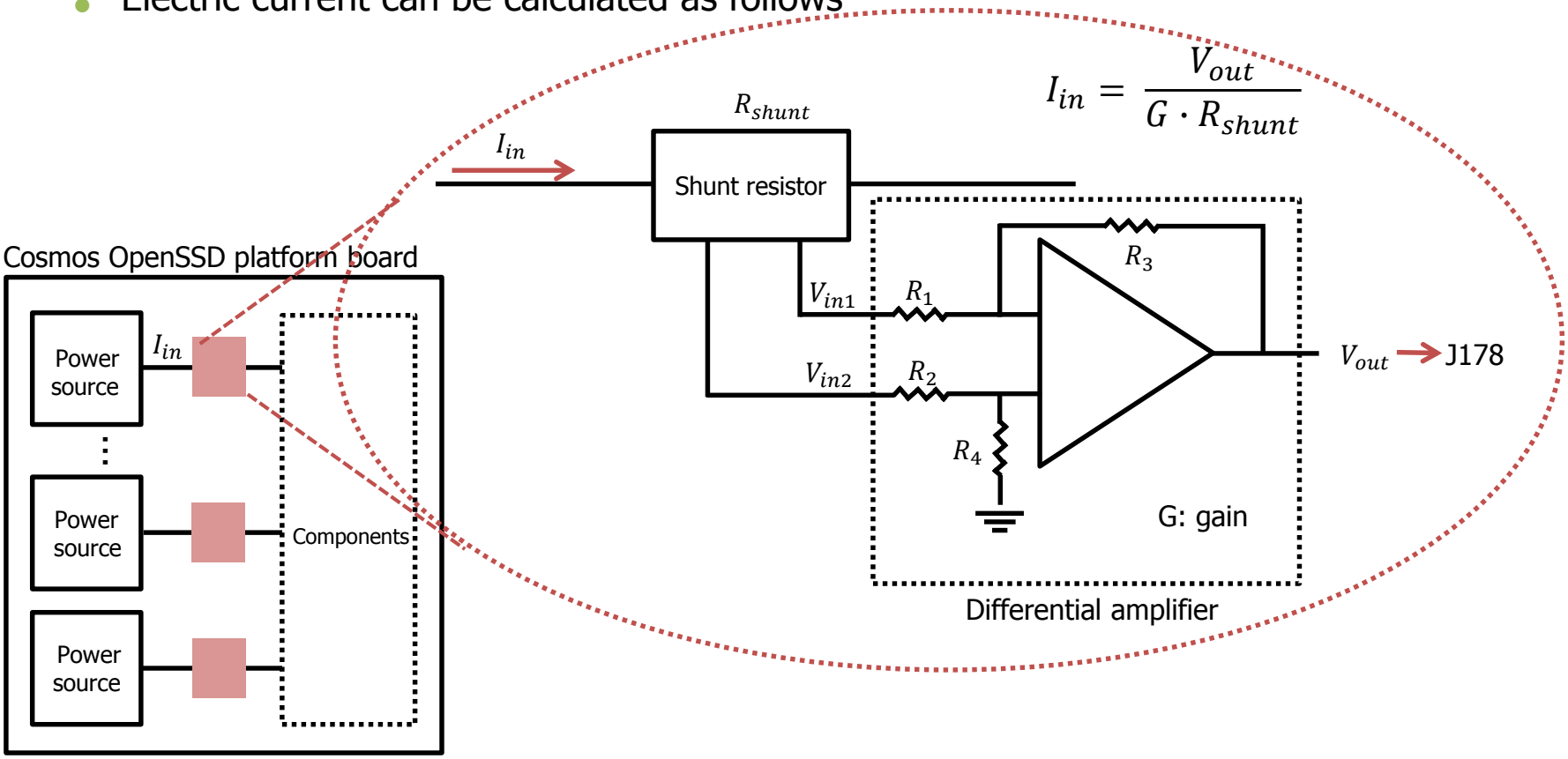
When the Platform is Powered up

- All power related LEDs should be turned on



Power Consumption Measurement

- User can check the power of various components
  - The voltage can be monitored via power measure module connector (J178)
  - Electric current can be calculated as follows

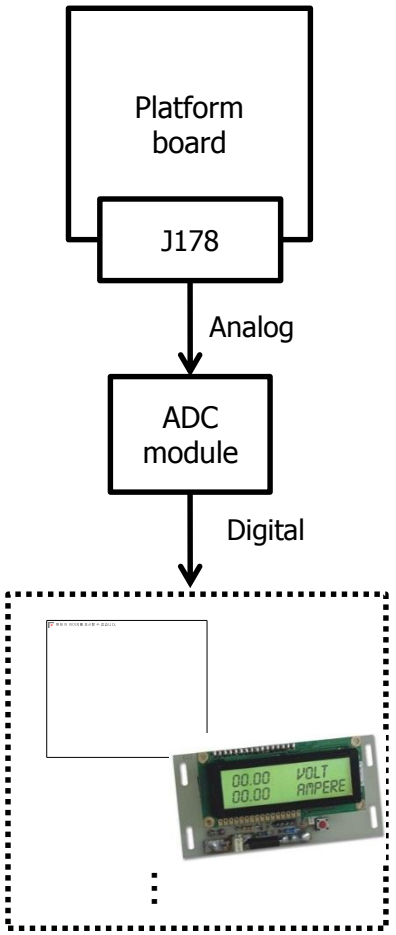


Power Measure Module Connector (J178)

- Additional ADC module can be used to measure power consumption
- Part number: 05002HR-60J02 (Yeonho)

Pin #	Net name	Pin #	Net name
2	I_VCCAUX_IO (G: 333)	36	MGTAVTT
4	I_MGTAVCC (G: 333)	38	MGTVCCAUX
6	I_MGTAVTT (G: 333)	39	VCC_NAND
8	I_MGTVCCAUX (G: 333)	40	VCC_NAND_IO
10	I_VCC_NAND (G: 100)	42	VCCO_ADJ
12	I_VCC_NAND_IO (G: 333)	43	VCC3V3_FPGA
14	I_VCCO_ADJ (G: 333)	44	VCCO_1V8
16	I_VCC3V3_FPGA (G: 100)	46	VCCAUX
18	I_VCCO_1V8 (G: 333)	47	VCC1V5_PS
20	I_VCCAUX (G: 100)	48	VCCINT
22	I_VCC1V5_PS (G: 333)	50	VCCPINT
24	I_VCCINT (G: 50)	51	VCCP1V8_FPGA
26	I_VCCPINT (G: 333)	52	VCC3V3_PS
28	I_VCCP1V8_FPGA (G: 333)	54	VCC3V3
30	I_VCC3V3_PS	56	VCC5V0
34	VCCAUX_IO	58	VCC12_P
35	MGTAVCC	The other	GND

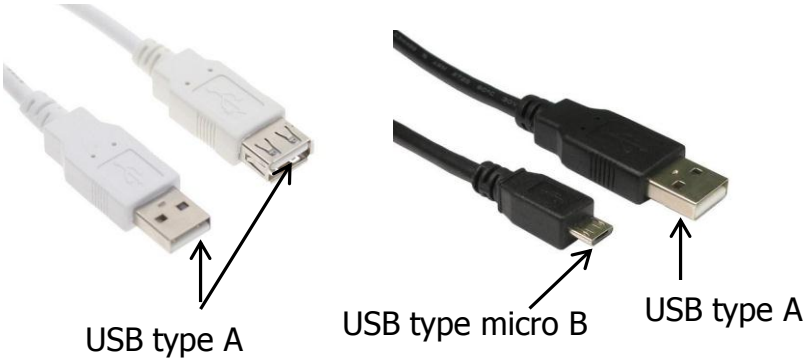
Notes
1. "I_" means the voltage to be used for calculating electric current
2. R_{shunt}: 5mΩ



Platform Board Setup

Hardware Preparation

- Hardware components
  - Cosmos OpenSSD platform board
  - External PCIe adapter
  - External PCIe cable
  - JTAG cable
    - USB type A to USB type micro B cable
    - Emulator, JTAG N pin cable (N: 7, 14, 20)
  - USB type A to USB type A cable for UART



Software Preparation

- Software tools
  - Xilinx ISE design suite 14.7 system edition
    - WebPACK is not allowed to use
  - UART terminal emulator software
    - Xilinx Software Development Kit(SDK) includes UART terminal
    - You can use separate software such as Xshell
- Software files: download from OpenSSD project homepage
  - EDK project file
  - Sources (C, RTL) / MCS file
  - PCIe device driver
    - Linux: available
    - Windows: soon to be released

FPGA Initialization Steps

Step 1. Insert flash module into SO-DIMM

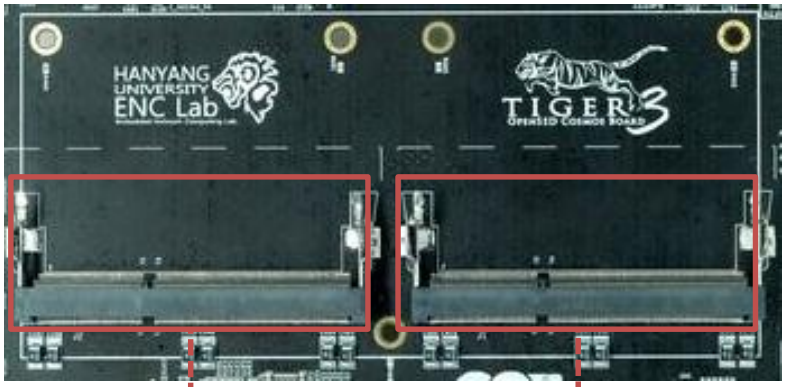
Step 2. Set FPGA configuration mode

Step 3. Connect board with PC

Step 4. Check status related LEDs

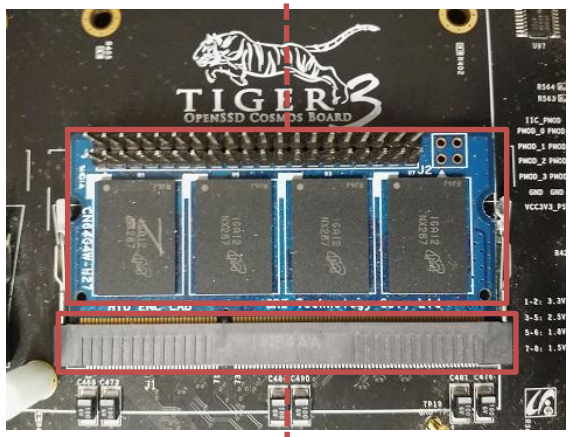
Step 1. Insert Flash Module into SO-DIMM

- The released version of RTL code only supports J1
  - You can not use both yet



SO-DIMM (J2) SO-DIMM (J1)

Flash module

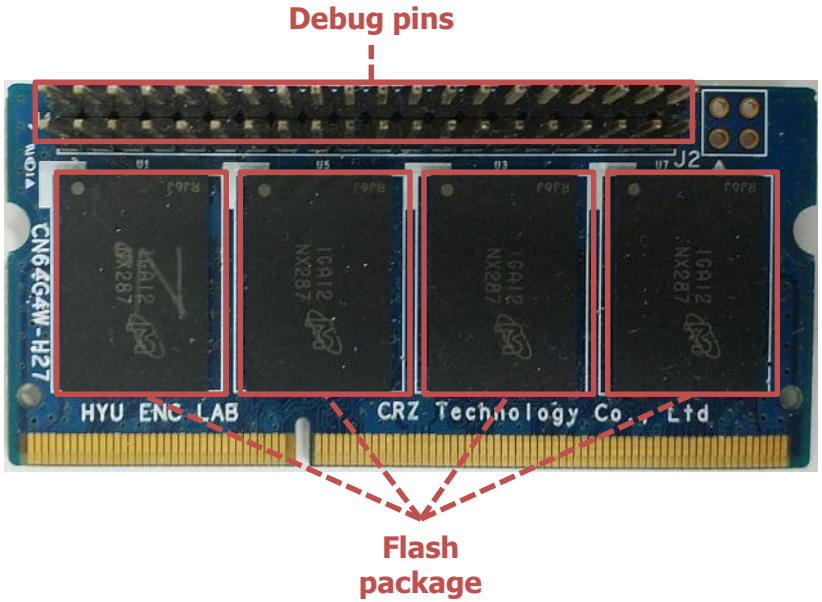


SO-DIMM (J1)

- Caution
  - SO-DIMM has its own pin assignment (different from SDRAM)
  - You should not insert any SDRAM module into this slot

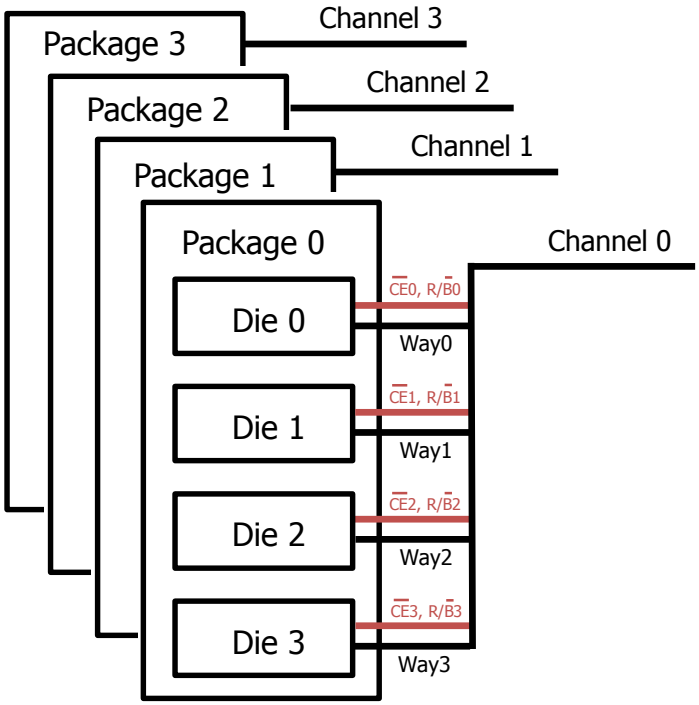
Flash Module

- Consist of 4 flash packages and debug pins
  - A flash package
    - Capacity: 32 GB
    - Page size: 8640 Bytes (spare area: 448 Bytes)
    - Part number: MT29F256G08CMCABH2 (Micron)
  - Debug pins
    - User can debug I/O signals



Channel Organization of Flash Module

- Module configuration
  - 4 channels/module and 4 ways/channel
- Shared signals within a channel (a package)
  - Dies in the same package share the I/O channel
  - Dies in the same package share command signals except Chip Enable ($\overline{CE}$)
  - Each die has own Ready/Busy (R/$\overline{B}$) signal



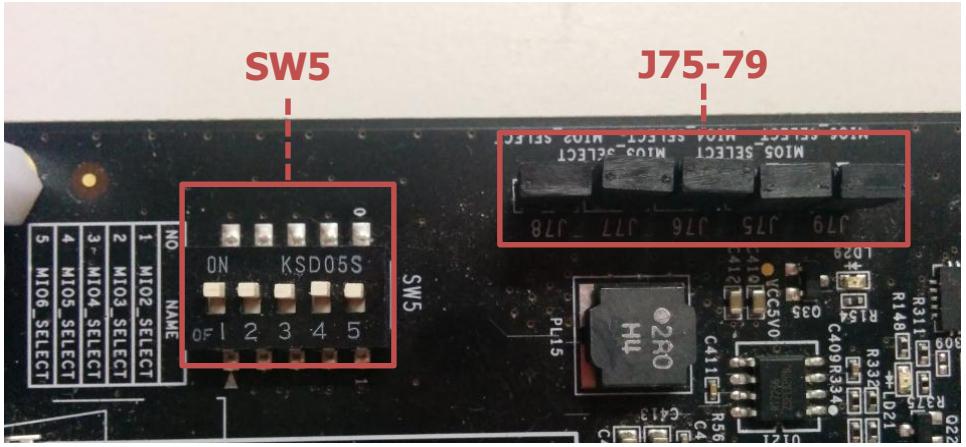
Debug Pins of Flash Module

- Debug signal assignment for channel 0

Pin #	Net name	Pin #	Net name
1	CHANNEL0_CE0#	2	CHANNEL0_IO0
3	CHANNEL0_CE1#	4	CHANNEL0_IO1
5	CHANNEL0_CE2#	6	CHANNEL0_IO2
7	CHANNEL0_CE3#	8	CHANNEL0_IO3
9	GND	10	CHANNEL0_IO4
11	GND	12	CHANNEL0_IO5
13	GND	14	CHANNEL0_IO6
15	GND	16	CHANNEL0_IO7
17	CHANNEL0_R/B0#	18	CHANNEL0_W/R#
19	CHANNEL0_R/B1#	20	CHANNEL0_CLE
21	CHANNEL0_R/B2#	22	CHANNEL0_ALE
23	CHANNEL0_R/B3#	24	CHANNEL0_WP
25	GND	26	GND
27	GND	28	CHANNEL0_CLK
29	GND	30	GND
31	GND	32	CHANNEL0_DQS
33	GND	34	GND
35	GND	36	GND
37	GND	38	GND
39	GND	40	GND

Step 2. Set FPGA Configuration Mode

- User can select how to configure the FPGA via the switch SW5



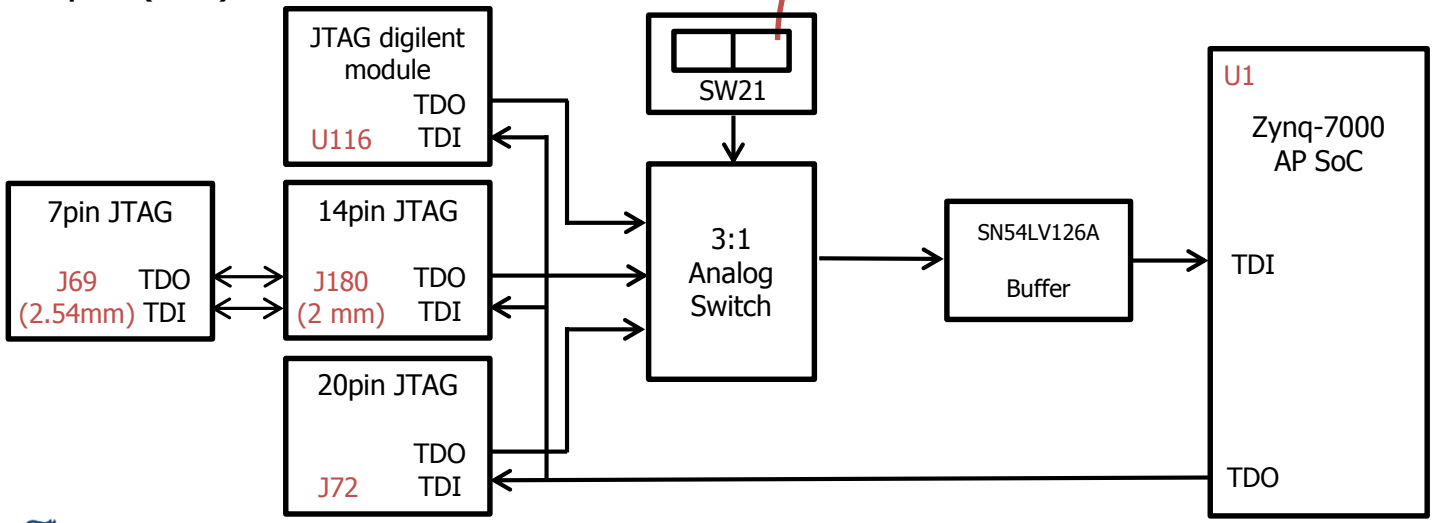
Mode	SW5					J75-79
	1	2	3	4	5	
JTAG	On	On	On	On	On	On
QSPI			On	Off		
SD card			Off	Off		

JTAG Configuration Mode

- Download the configuration files to the platform board
  - Configuration files are generated by Xilinx ISE design suite
  - ELF file: software configuration file
  - BIT file: hardware programing file
  - BMM file: block memory map file
- Use may use one of three JTAG I/Fs
  - Digilent module (U116)
  - 14pin (J180)
  - 20pin (J72)

JTAG select SW (SW21) setting

Type	SW21	
	1	2
Digilent module	Off	On
14pin	Off	Off
20 pin	On	Off

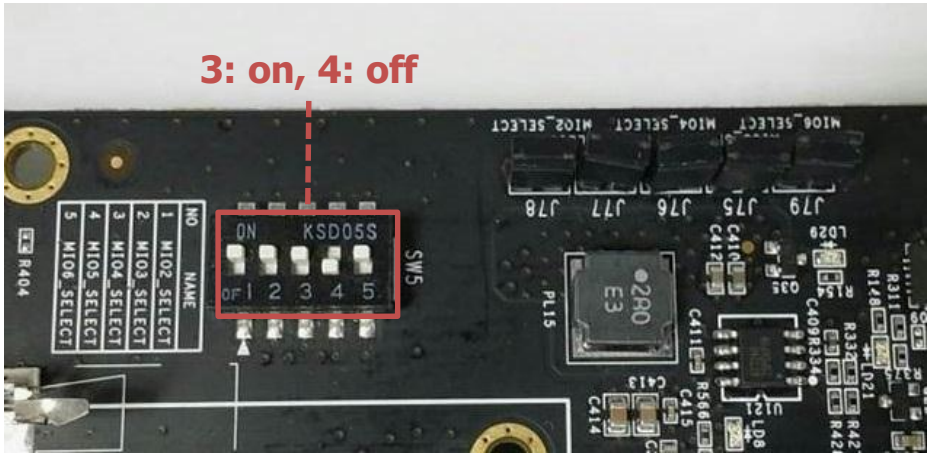


JTAG connection diagram

QSPI Configuration Mode

- Before booting the board, download MCS file to QSPI memory
  - MCS file: PROM file format including configuration data
  - QSPI memory (U113,114)
    - Nonvolatile
    - Part number: S25FL128SAGMFIR01(Spansion)

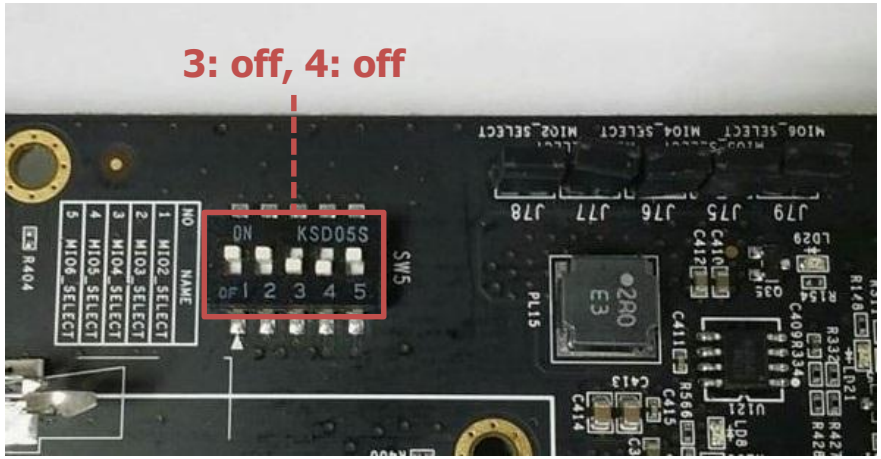
- When booting the board, the downloaded MCS file is used to initialize FPGA



SD Card Configuration Mode

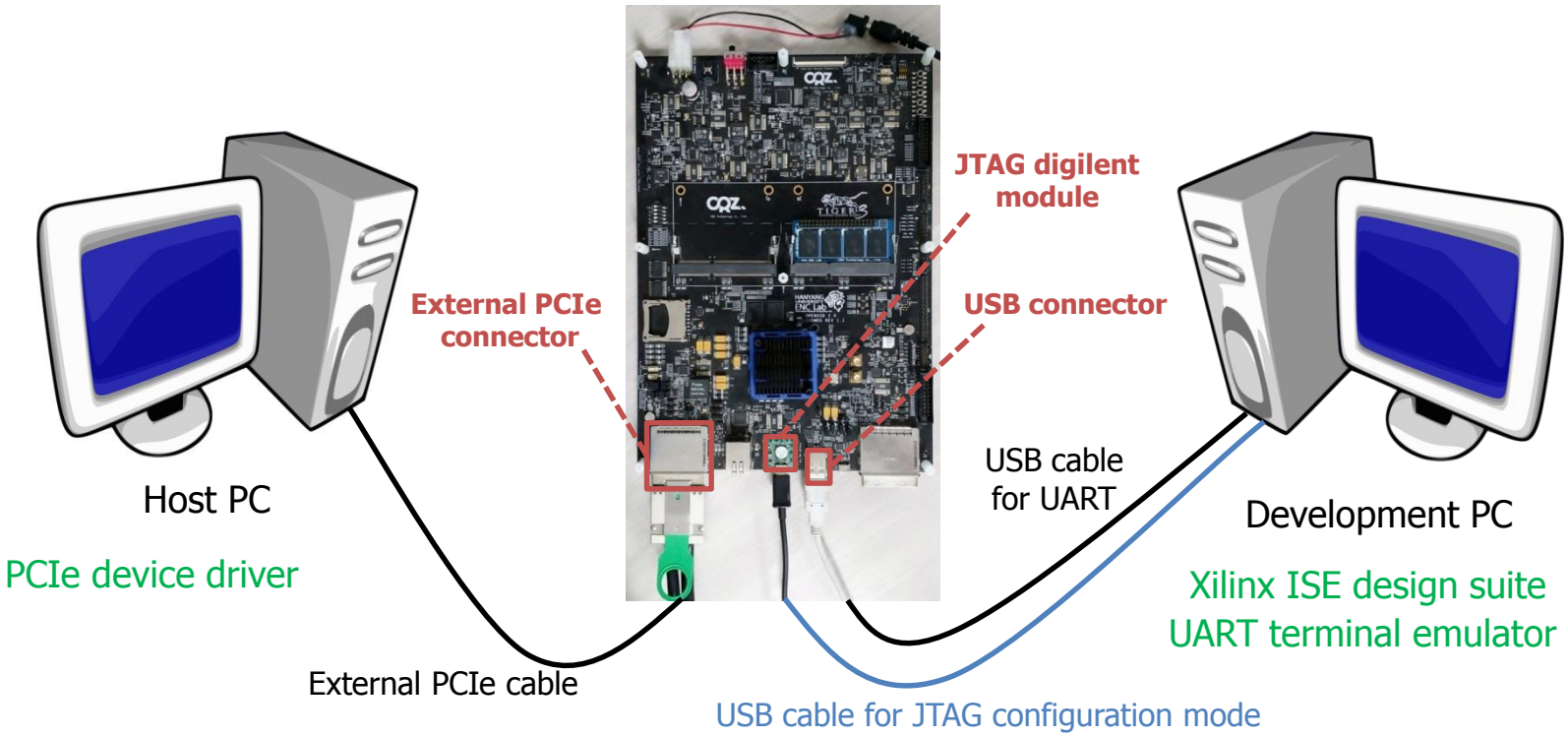
- Before booting the board, download MCS file to SD card
  - MCS file: PROM file format including configuration data
  - SD card connector (J34): 67840-8001(Molex)

- When booting the board, the downloaded MCS file is used to initialize FPGA



Step3. Connect Board with PC

- Software preparation
  - Xilinx ISE design suite 14.7 system edition is installed on development PC
  - UART terminal emulator software is installed on development PC
  - PCIe device driver is installed on host PC



External PCIe

- External PCIe adapter card
  - Installed on host PC
  - Provide a high-performance and low latency solution for expanding PCIe
  - Part number: OSS-PCIe-HIB25-x8 (One Stop Systems)
  - Interface: PCIe x8 Gen 2.0
- External PCIe cable
  - Part number: 74546-0801 (Molex)
- External PCIe connector (CN1) in platform board
  - Up to 8 lanes
  - 2.5 GT/s for a Gen1, 5.0 GT/s for a Gen2
  - Connected with high data rate serial transceiver in FPGA
  - Part number: 75586-0007 (Molex)

External PCIe adapter

External PCIe cable

PCIe Slot in Host PC

- Make sure how many PCIe lanes are supported in your pc before enabling them
- The released PCIe interface uses only 1 lane
- Example
  - You may use either 2-, 4- or 8-lane PCIe interface as long as your host PC supports

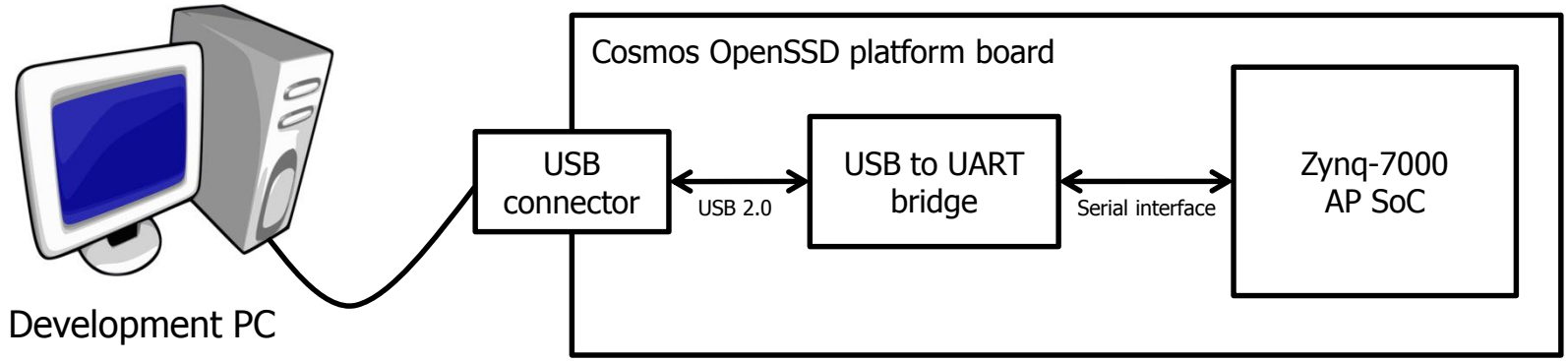
- 2 x PCI Express 3.0 x16 Slots (PCIE2/PCIE4: single at x16 (PCIE2); dual at x8 (PCIE2) / x8 (PCIE4))
- 1 x PCI Express 2.0 x16 slot (PCIE5: x2 mode)
- 2 x PCI Express 2.0 x1 Slots

Supports 2- and 8-lane PCIe interface

PCIe Slot Specification of ASR*** Z97 extreme6 motherboard

Development Monitor

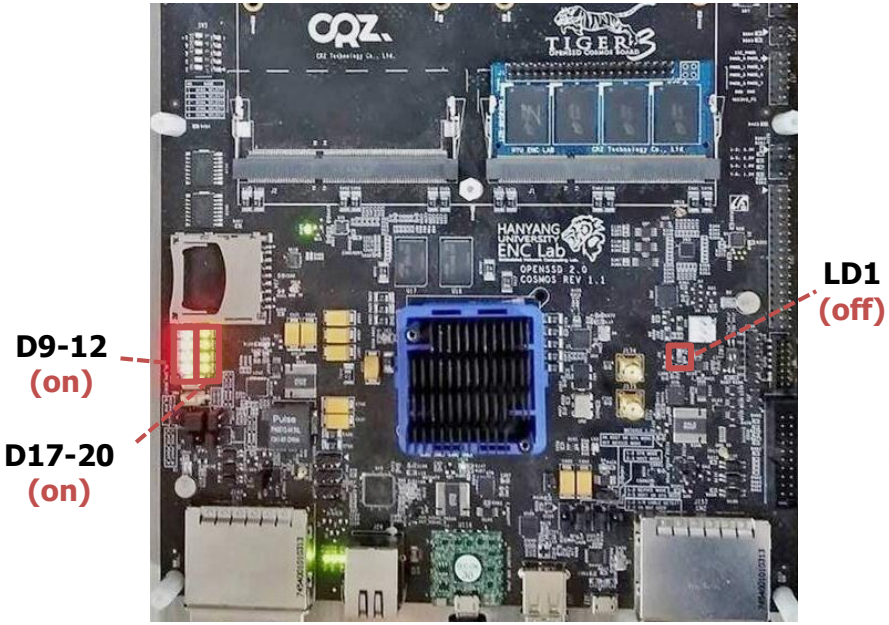
- The development PC is connected via the USB connector(J33)
  - USB type A
  - USB 2.0
- The platform board has a USB-to-UART bridge(U108) on itself
  - But, powered by the development PC
  - Needs USB to UART bridge driver (supplied by the device vendor)
  - Part number: CP2103GM (Silicon Labs)



Step 4. Check Status Related LEDs

LED	Description	Color	Before programing FPGA	After programing FPGA
LD1	FPGA initialization indicator	Green	Off	On
D9-12	User-configurable LEDs	Red	On	Off
D17-20		Green		

Note: FPGA has a pin called PUDC_B which chooses either a pull-up mode or a pull-down mode on power-up.
PUDC_B is pre-set to pull-up mode



Before initializing FPGA

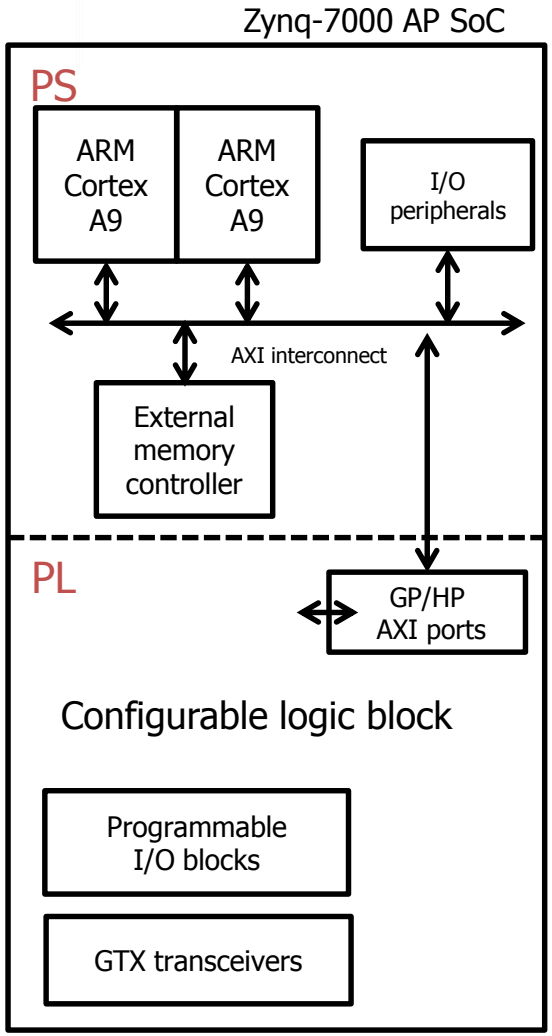
After initializing FPGA

HANYANG UNIVERSITY ENC Lab
Embedded Network Computing Lab.

Components Description

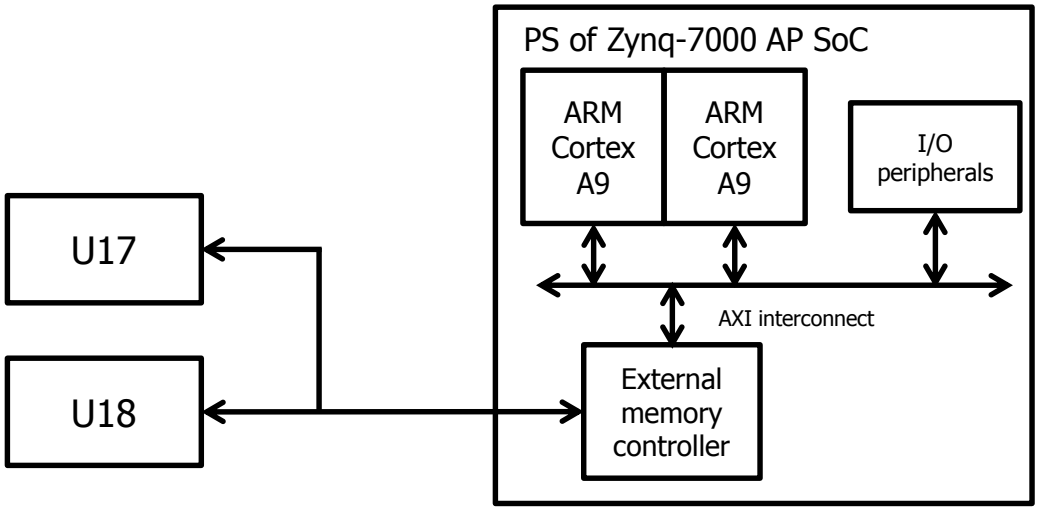
Zynq-7000 AP SoC (U1)

- Zynq-7000 FPGA has two regions: PS and PL
- Operation
  - Storage controller is synthesized in PL
  - Firmware is running on the Cortex-A0 in PS
  - Other I/O peripheral interfaces
    - UART, I2C, SPI, ethernet and USB
- Organization
  - Processing System (PS)
    - Dual-core ARM Cortex-A9
    - External memory interfaces
    - I/O peripherals and interfaces
    - ARM AMBA AXI based interconnect
  - Programmable Logic (PL)
    - Configurable logic block
    - Programmable I/O blocks
    - GTX transceivers: high data rate serial transceivers
- Part number: XC7Z045-FFG900-3



DDR3 DRAM (U17,18)

- Component Use
  - Data and instruction space for storage controller, firmware and host interface
- DRAM Feature
  - Capacity: 1GB
  - Interface data rate: DDR3 1066
  - Data width: 32bit
- Part number: K4B2G1646C-HCK0 (Samsung)



Ethernet

- Component Use
  - Ethernet can be used for physical layer interface of a storage network
  - Zynq-7000 AP SoC supports 10/100/1000 Mb/s ethernet MAC
- Organization
  - Ethernet connector (J24): R18101DBG_LC (M-TEK)
  - Ethernet Transceiver (U19)
    - Part number: 88E1116R (Marvell)
    - Jumper setting

Jumper	Description	Default	Selects
J85	Setting PHY CONFIG3 pin	1-2	1-2: CONFIG3 =0 (1K pull-down to GND) 2-3: CONFIG3 =1 (1K pull-up to 1.8V)
J86	Connect CONFIG3 to LED	Open	1-2: LD2 2-3: LD3
J87	Setting PHY CONFIG2 pin	Open	1-2: LD2 2-3: CONFIG2 =1
J88	Setting PHY CONFIG2 pin	1-2	1-2: CONFIG2 =0 (1K pull-down to GND)

- The current design release has no specific use of this component
  - You may use this upon your application

USB 2.0 ULTP

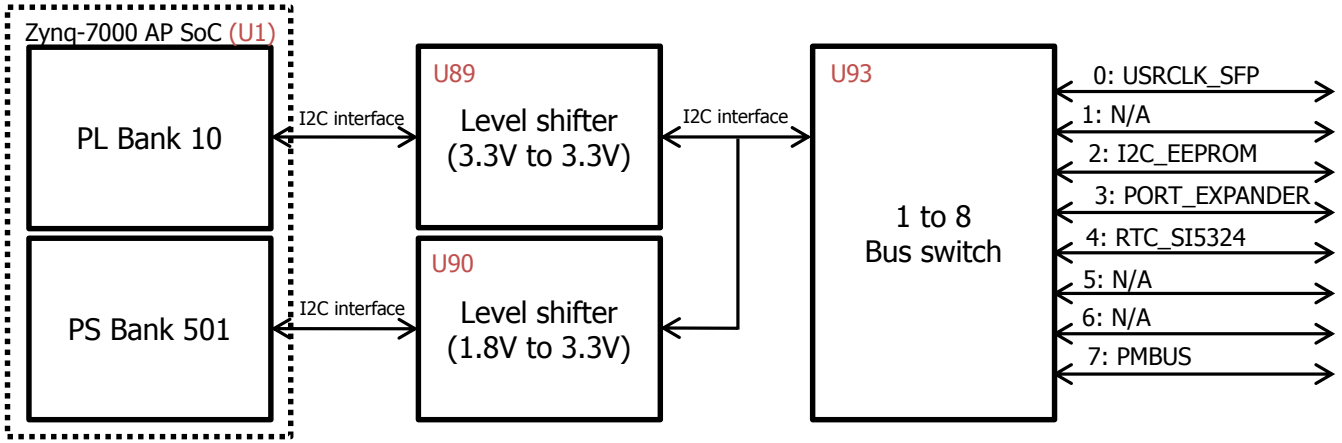
- Component Use
  - USB interface can be used for host interface
  - Zynq-7000 AP SoC supports USB2.0 PHY interface
- Organization
  - USB connector (J22): ZX62D_AB_5P8 (HIROSE)
  - USB 2.0 transceiver (U58)
    - Part number: USB3320_QFN32 (SMSC)

Header	Function	Shunt Position
J32	USB PHY reset	Shunt on: USB PHY reset Shunt off: USB PHY normal operation
J27	Host/OTG or device	Shunt on: Host or OTG mode Shunt off: Device mode
J28	RVBUS select	1–2: Device mode (10 KΩ) 2–3: Host or OTG mode (1 KΩ)
J29	CVBUS select	1-2: 1 μF to GND 2-3: 120 μF to GND
J31	Cable ID select	1-2: A/B cable detect 2-3: ID not used
J30	USB Type-A	1-2: Shield connected to GND 2-3: Shield floating

- The current design release has no specific use of this component
  - You may use this upon your application

Bus Switch for I2C

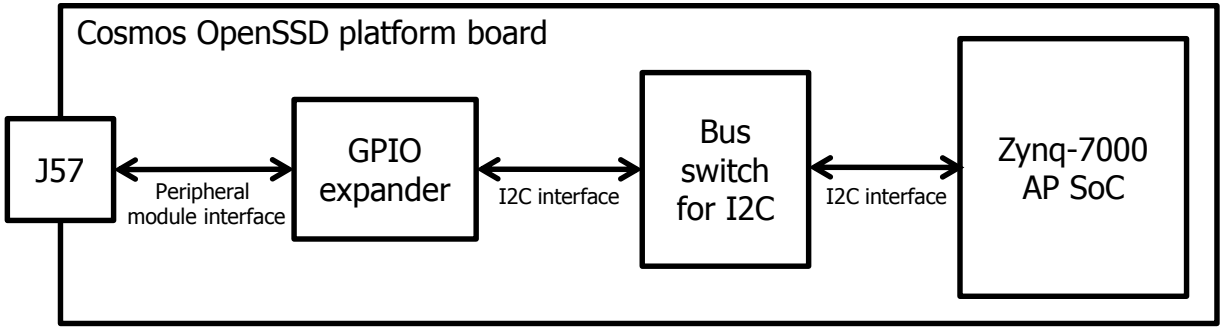
- Supports multiple I2C connections
  - User can select a device by sending I2C address ahead to bus switch



I2C Bus	Switch Position	I2C Address	Device	Description
PCA9648 bus switch	NA	0b1110100	PCA9548ARGER (U93)	8 channel I2C switch
USRCLK_SFP	0	0b1011101	Si570 (U62)	LVDS I2C programmable oscillator
I2C_EEPROM	2	0b1010100	M24C08 (U91)	EEPROM
PORT_EXPANDER	3	0b1000000	PCA9555BS_HVQFN24(U20)	General Purpose parallel Input/Output (GPIO) expansion for I2C interface
RTC_SI5324	4	0b1010001	RTC8564JE (U92)	Real time clock module
		0b1101000	SI5324 (U54)	Jitter attenuator
PMBUS	7	0b1100101	UCD90120A (U102)	Power controller

PMOD Pins

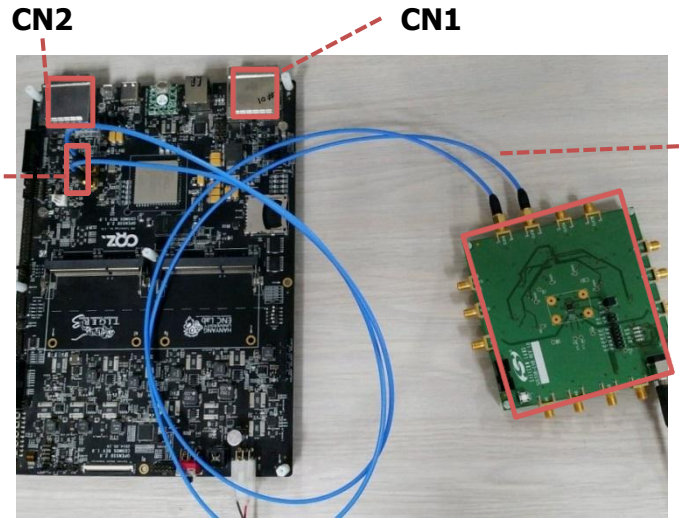
- Component Use
  - Supports Peripheral MODule (PMOD™) interface
  - PMODs™ are small I/O interface boards that offer a way to extend the capabilities of FPGA
- Organization
  - I2C PMOD pin (J57)
    - J57 is connected with GPIO expander
    - GPIO expander (U20) connects I2C interface and peripheral module interface



  - PMOD pin (J58)
    - Connected with FPGA IO pin through level shifter (3.3V to 1.8V)
    - Operating voltage of J58: 3.3V
    - Operating voltage of FPGA IO pin:1.8V

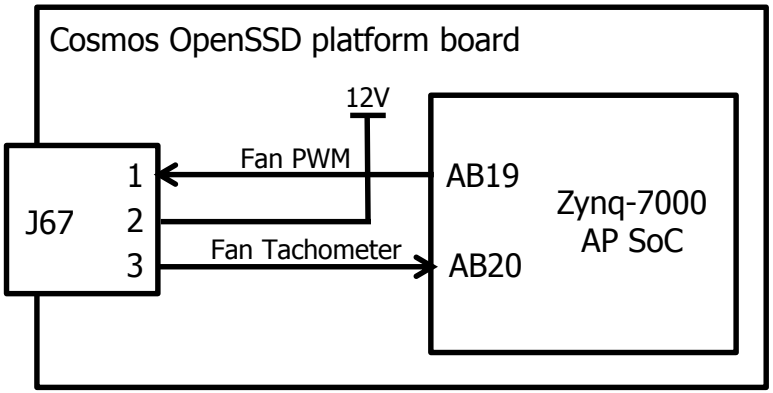
SMA Connector (J174,175)

- SubMiniature version A (SMA)
  - Connector interface for coaxial cable with a screw type coupling mechanism
  - Designed for use from DC to 18 GHz
- Component Use
  - When using a spare connector (CN2), a reference clock signal may be provided via the SMA connector
    - Input clock is used as reference clock for GTX transceivers connected with CN2
    - External clock generator can be used to generate a clock signal
    - Input clock should meet reference clock characteristics
      - Refer to Xilinx data sheet (DS191)



Fan Connector (J67)

- Component Use
  - To dissipate the heat generated by the FPGA
- Organization
  - Pulse-Width-Modulated(PWM) signal
    - Control fan speed
    - Need additional logic in FPGA to control PWM
  - Tachometer signal
    - Monitor fan speed



HANYANG UNIVERSITY ENC Lab Embedded Network Computing Lab.

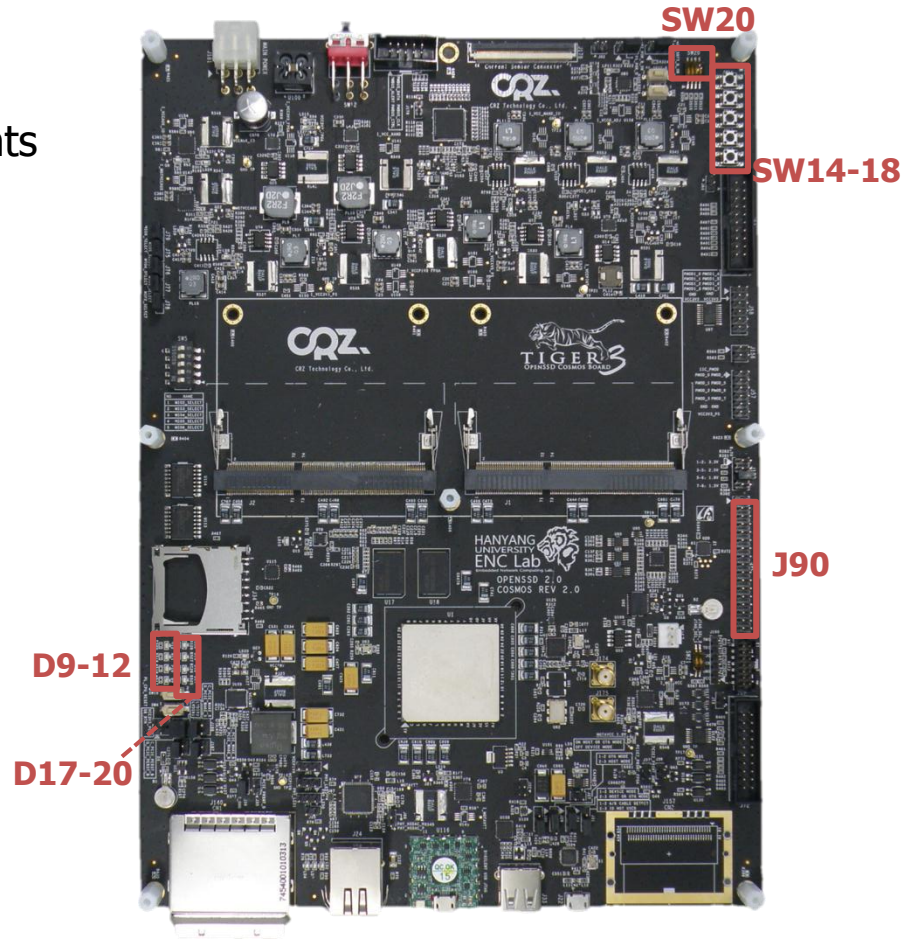
Back

ARM JTAG

- Component Use
  - To aid the debug process of firmware operation by monitoring ARM CPU
- Organization
  - Standard JTAG interface
    - Zynq-7000 AP SoC provides ARM CPU debug access via a standard JTAG (IEEE 1149.1) debug interface
    - Refer to Xilinx user guide (chapter 27 of UG585)
    - User can use U116, J180 orJ72 for standard JTAG debug interface
  - ARM JTAG interface
    - User can use 20pin ARM JTAG header (J84) for ARM debugger such as RealView ICE

Other Debugging Aids

- Use
  - You may pull out the signals to inspect out of the FPGA
  - Check pin assignment in guide documents (click)
- Organization
  - User-configurable GPIO pin (J90)
    - Connected with GPIO pin of FPGA
    - Total of 34 pins provided
  - User-configurable switches
    - Five push buttons: SW14-18
    - Four slide switches: SW20
  - User-configurable LEDs
    - Four red LEDs: D9-12
    - Four green LEDs: D17-20



Control and Configuration Switches

Switch	Name	Description
SW2	PS_POR_B	The master reset of the entire chip. It must be held low through PS power-up.
SW3	PS_SRST_B	Reset all of the functional logic.
SW4	Program_B push button	Clear the programmable logic configuration.
SW5	Configuration mode SW	Select configuration mode
SW12	Power SW	Main power on/off slide switch
SW21	JTAG select SW	Selct JTAG type

Status LEDs

LED	Net Name	LED Color	Description
LD11	POR	Green	Processor system operating
LD1	DONE	Green	FPGA bit file download is complete
LD2	PHY_LED0	Green	Ethernet PHY LED0
LD3	PHY_LED1	Green	Ethernet PHY LED1
LD4	PHY_LED2	Green	Ethernet PHY LED2
D4	U106_FLG	Red	USB 2.0 MOSFET power switch fault

Authors

Name	E-mail	Contribution
Youngnam Kim	ynkim@enc.hanyang.ac.kr	2013-01 ~ 2014-12
Jaewook Kwak	jwkwak@enc.hanyang.ac.kr	2015-01 ~ Now

Cosmos OpenSSD Platform Tutorial

Flash Storage Controller

ENC Lab. @ Hanyang University

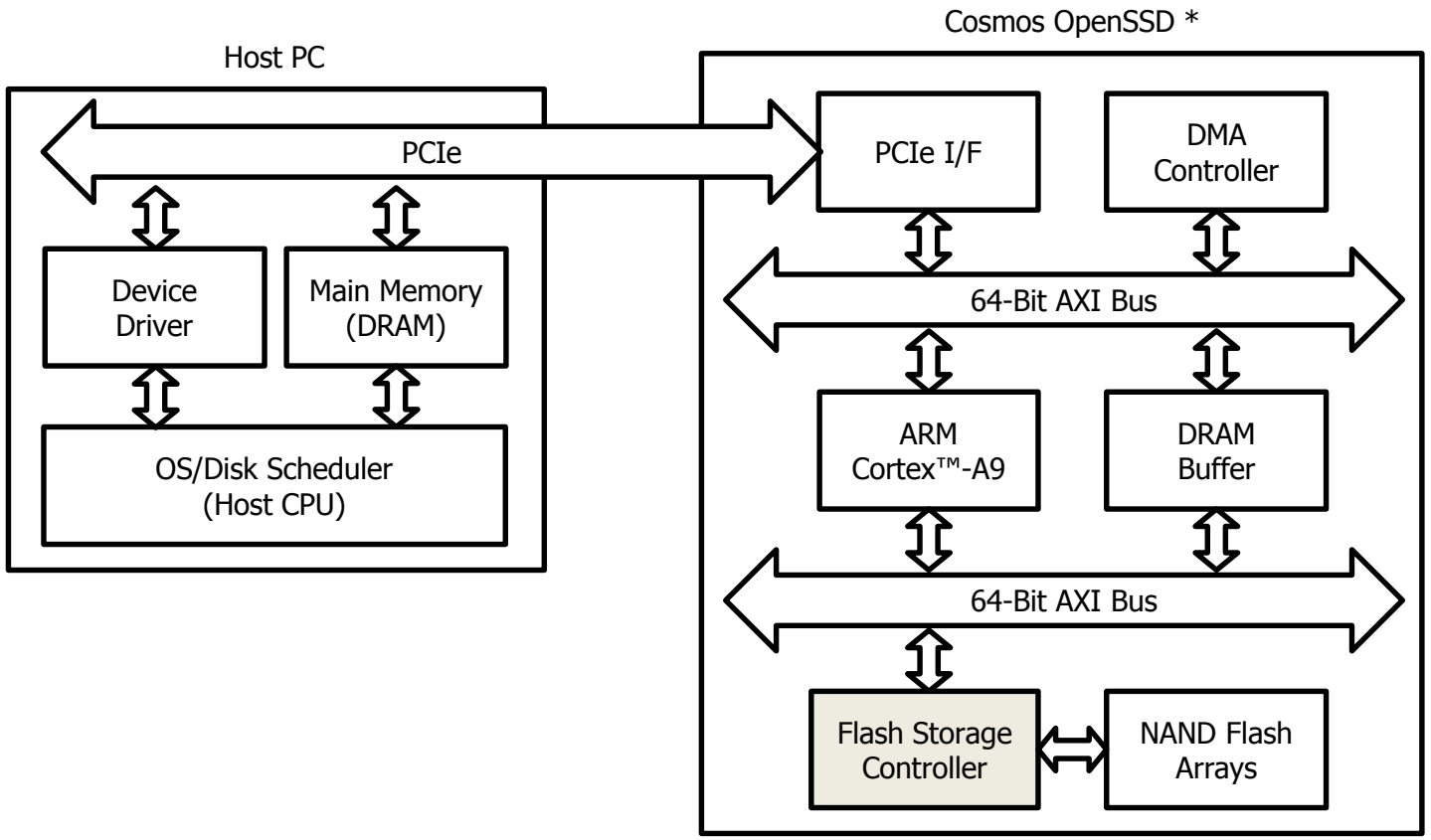
HANYANG UNIVERSITY
ENC Lab
Embedded Network Computing Lab.

Document Revision History

- v1.0.0
  - Initial release
  - This document is based on the FSC_v1.0.0

Section Scope

- Design of flash storage controller in Cosmos OpenSSD



*also called 'storage device' or 'device'

INDEX

Storage System Overview

Storage System Overview

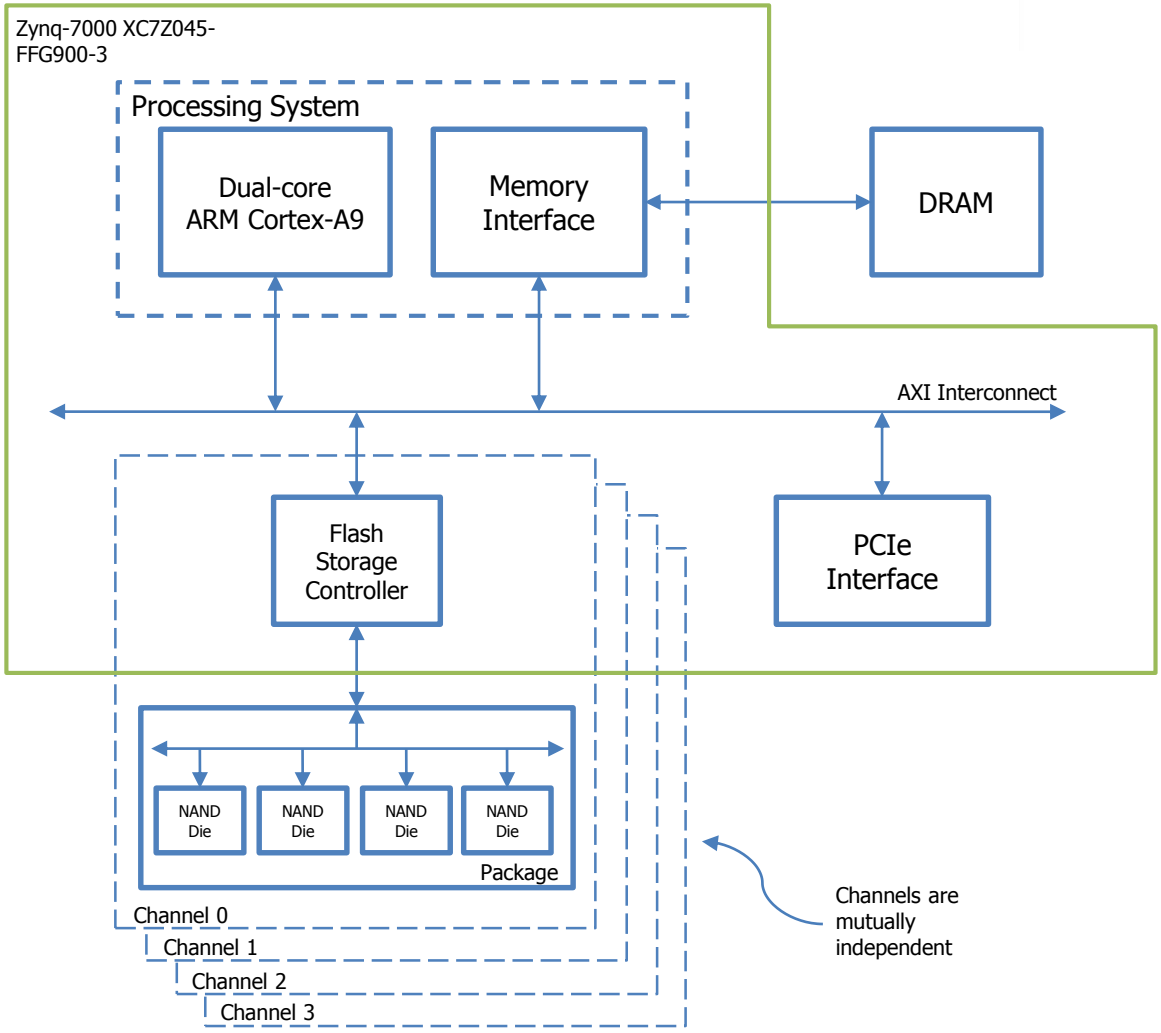


Figure 1. Cosmos OpenSSD storage system overview

Zynq-7000 Architecture

- Xilinx's embedded SoC
- Two regions
  - Processing System (PS)
    - Hardwired components
    - Executes the firmware program
  - Programmable Logic (PL)
    - Programmable components (FPGA region)
    - Flash storage controller (FSC) and PCIe engine are in here
- Benefits of Using Zynq
  - CPU is more faster than soft core (such as MicroBlaze)
  - No need to worry about organizing hardware memory controller, and some other peripherals (such as UART)
  - Xilinx supports BSP (Board Support Package)

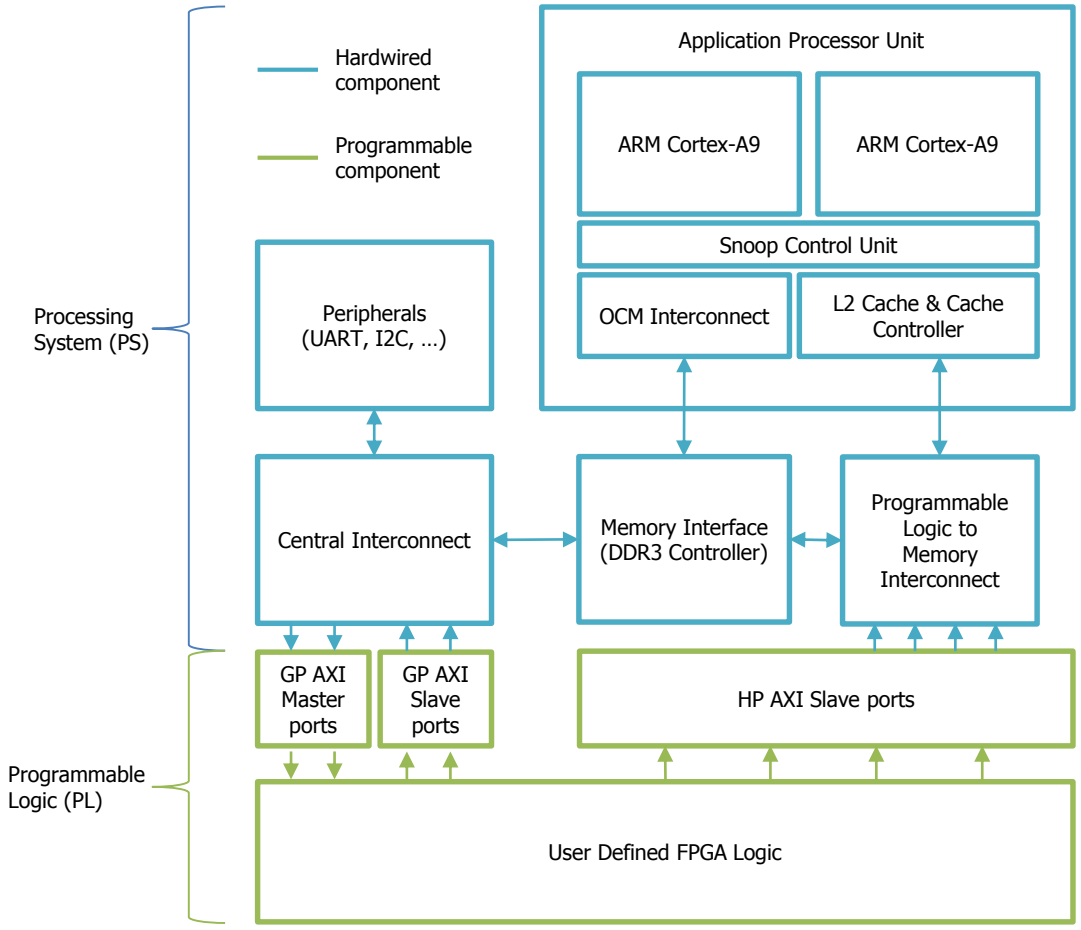


Figure 2. Zynq-7000 architecture overview

System Bus Structure

- General Purpose (GP) AXI4 Lite bus
  - 32-bit data path
  - Used for control
  - Operates @ 100MHz
- High Performance (HP) AXI4 bus
  - 64-bit data path
  - Used for DMA operations between flash storage controller/PCIe interface and DRAM
  - Operates @ 100 MHz

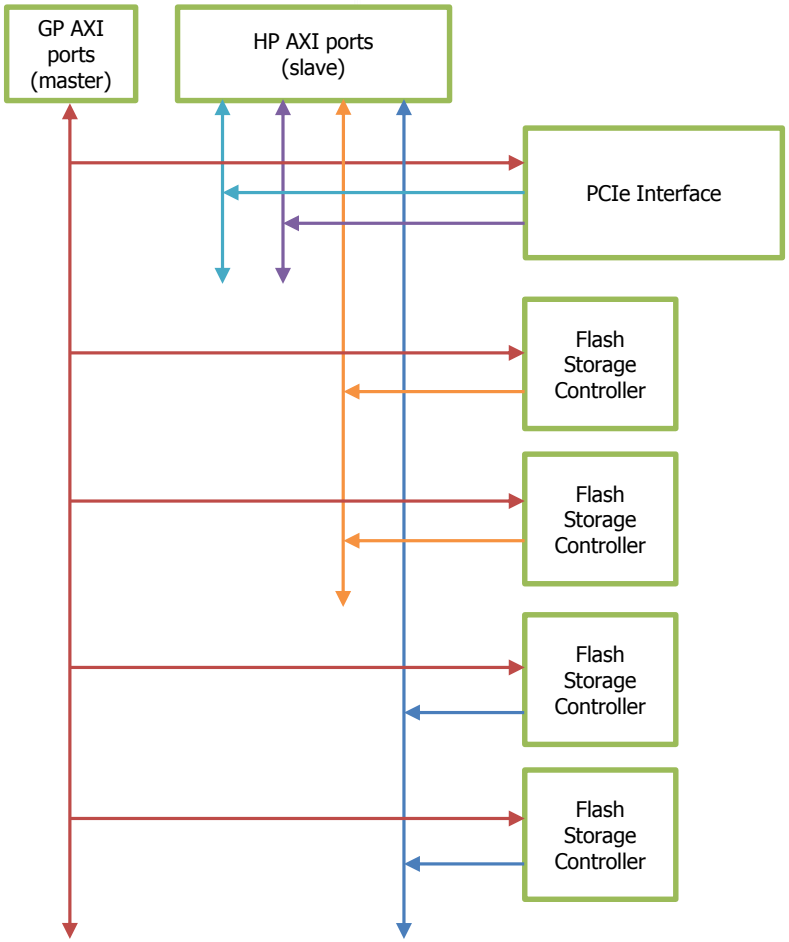


Figure 3. Bus structure of Cosmos OpenSSD

Flash Bus Structure

- Flash channel
  - One flash chip package constitutes one data channel
  - Operates @ 50MHz, DDR (100MB/s data rate)
- Flash way
  - Four dies in a flash chip package share the channel
  - Each die has its own chip enable (CE) and ready/busy (R/B) signal pin

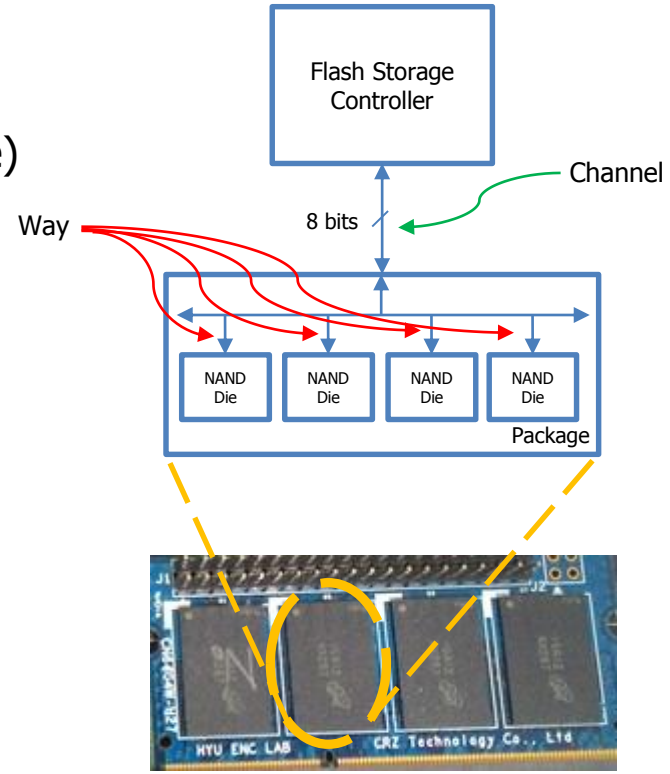


Figure 4. Flash bus structure of Cosmos OpenSSD

Address Map of the Device

Address Range	Name	Description
0x00000000~0x3FFFFFFF	DRAM	Main memory of the device Includes control registers, queues, and data
0x40200000~0x4020FFFF	DMA Controller	DMA controller register file
0x65C00000~0x65C0FFFF	PCIe Interface	PCIe interface register file
0x65C20000~0x65C2FFFF	AXI Aperture	AXI aperture mapped to DRAM in the host PC through PCIe interface
0x66E00000~0x66E0FFFF	Flash Storage Controller Channel 3	Channel 3 flash storage controller register file
0x66E20000~0x66E2FFFF	Flash Storage Controller Channel 2	Channel 2 flash storage controller register file
0x66E40000~0x66E4FFFF	Flash Storage Controller Channel 1	Channel 1 flash storage controller register file
0x66E60000~0x66E6FFFF	Flash Storage Controller Channel 0	Channel 0 flash storage controller register file
0x7A000000~0x7A00FFFF	PCIe Interface initialization checker	PCIe interface initialization checker register file

Flash Storage Controller

HANYANG UNIVERSITY
ENC Lab
Embedded Network Computing Lab.

FSC Feature

- Primitive NAND flash storage controller
  - No luxury features (such as compression, de-duplication, etc.)
- ONFI 2.2 compliant
  - Only synchronous mode is supported
- ECC support
  - BCH error correction code
    - 32bits correctable per 2KB with 60B parity
- Way interleaving
  - Maximum 8-way (die) interleaving may be supported
  - Default interleaving level is 4-way

Supported Operations

- Flash storage controller (FSC) provides low level operations to FTL
  - Reset
    - Reset both flash device and controller
  - Mode change
    - Switch asynchronous mode to synchronous mode
  - Read way status
    - Read last-checked status of the way (die)
  - Read page
  - Program (write) page
  - Erase block

- FSC utilizes 6 flash memory operations for implementing low level operations
  - RESET
  - READ STATUS
  - READ PAGE
  - PROGRAM PAGE
  - ERASE BLOCK
  - SET FEATURES
    - Used to switch asynchronous mode to synchronous mode

Materials

- Sources are available at
  - http://166.104.36.35:10080/Cosmos_OpenSSD/sources/RTL/flash_storage_controller/

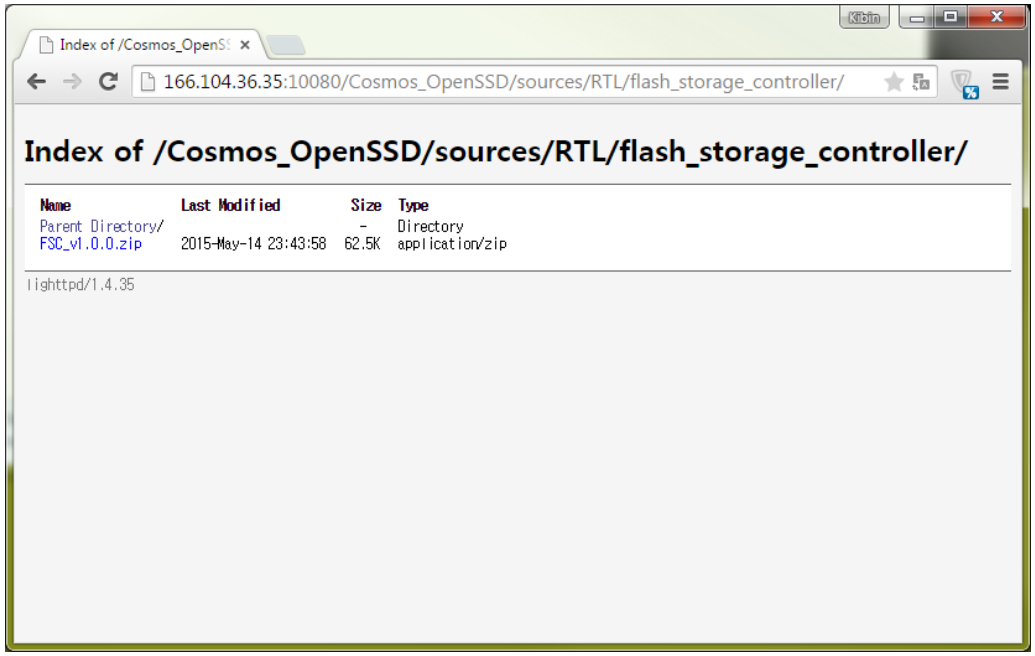


Figure 5. Source distribution website

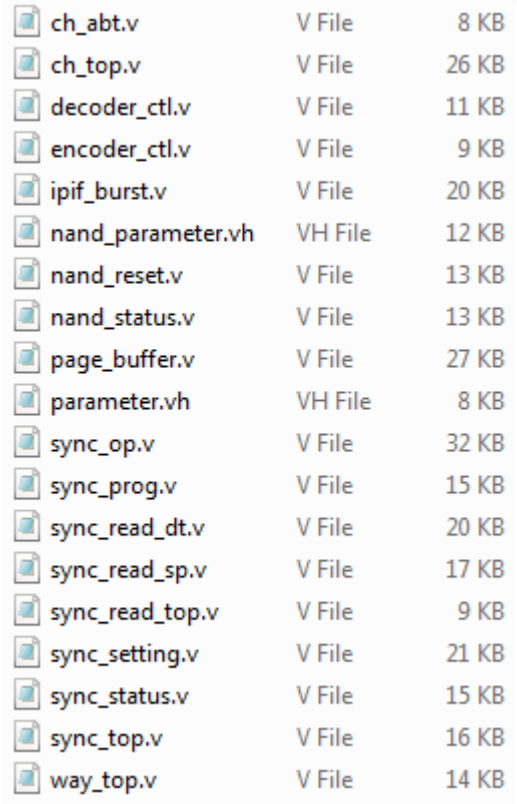


Figure 6. Source files tree

Register map

- Way (die) controller is visible to FTL directly
  - Flash storage controller (channel controller) is invisible to FTL
  - FTL sends commands to way controllers directly
- Register functions on reading and writing are different
  - Though two registers have the same offset, their functions might be different based on the operation (read, write)

Table 1. Register map on reading

Offset	Description
0x40	read status of way 3
0x44	read last issued command from way 3
0x48	read last given DMA address from way 3
0x4C	read last given flash row address from way 3
0x50	read status of way 2
0x54	read last issued command from way 2
0x58	read last given DMA address from way 2
0x5C	read last given flash row address from way 2
0x60	read status of way 1
0x64	read last issued command from way 1
0x68	read last given DMA address from way 1
0x6C	read last given flash row address from way 1
0x70	read status of way 0
0x74	read last issued command from way 0
0x78	read last given DMA address from way 0
0x7C	read last given flash row address from way 0

Table 2. Register map on writing

Offset	Description
0x40	issue command to way 3
0x44	should be 0 (default is 0)
0x48	give DMA address to way 3
0x4C	give flash row address to way 3
0x50	issue command to way 2
0x54	should be 0 (default is 0)
0x58	give DMA address to way 2
0x5C	give flash row address to way 2
0x60	issue command to way 1
0x64	should be 0 (default is 0)
0x68	give DMA address to way 1
0x6C	give flash row address to way 1
0x70	issue command to way 0
0x74	should be 0 (default is 0)
0x78	give DMA address to way 0
0x7C	give flash row address to way 0

FSC Internal Organization

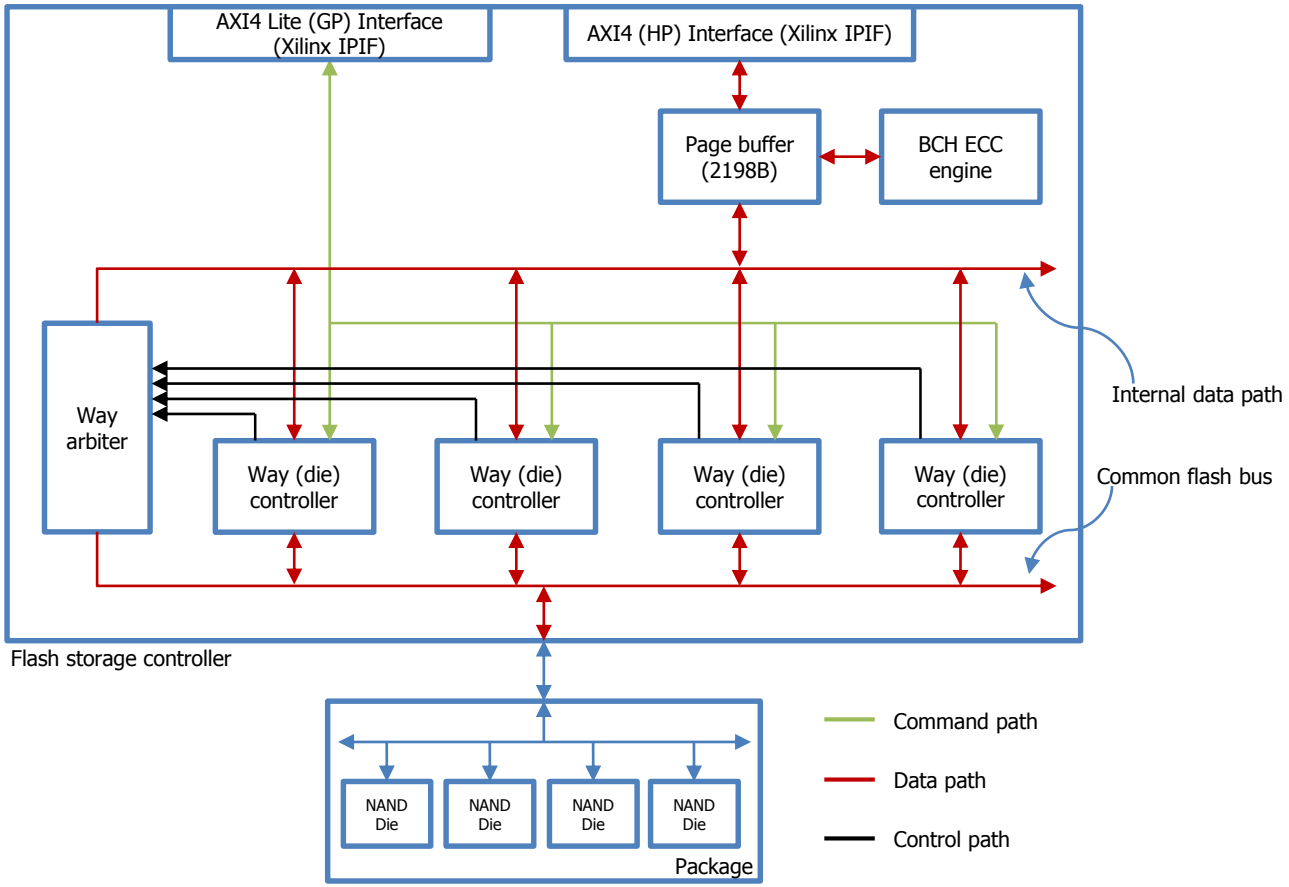


Figure 7. FSC architecture overview

Way Arbiter (1/2)

- Way arbiter is responsible for two different buses: common flash bus and internal data bus
  - Used when two or more way controllers ask for permission to use the common flash bus and access page buffer
- Permission is granted in a round-robin manner
  - Fairness is not guaranteed

```
lastGrantedWay = 0
while (true)
    if (requestArrived)
        for (i = 0 to numberOfWays)
            nextCandidate = (lastGrantedWay + i) % numberOfWays
            if (hasRequested(nextCandidate))
                grantPermission(nextCandidate)
```

Code 1. Software pseudo code equivalent to hardware RTL code

Source file: ch_abt.v

Way Arbiter (2/2)

- *Permission granted* means that the way controller can do the following operations
  - Access page buffer (through data path)
  - Use flash bus

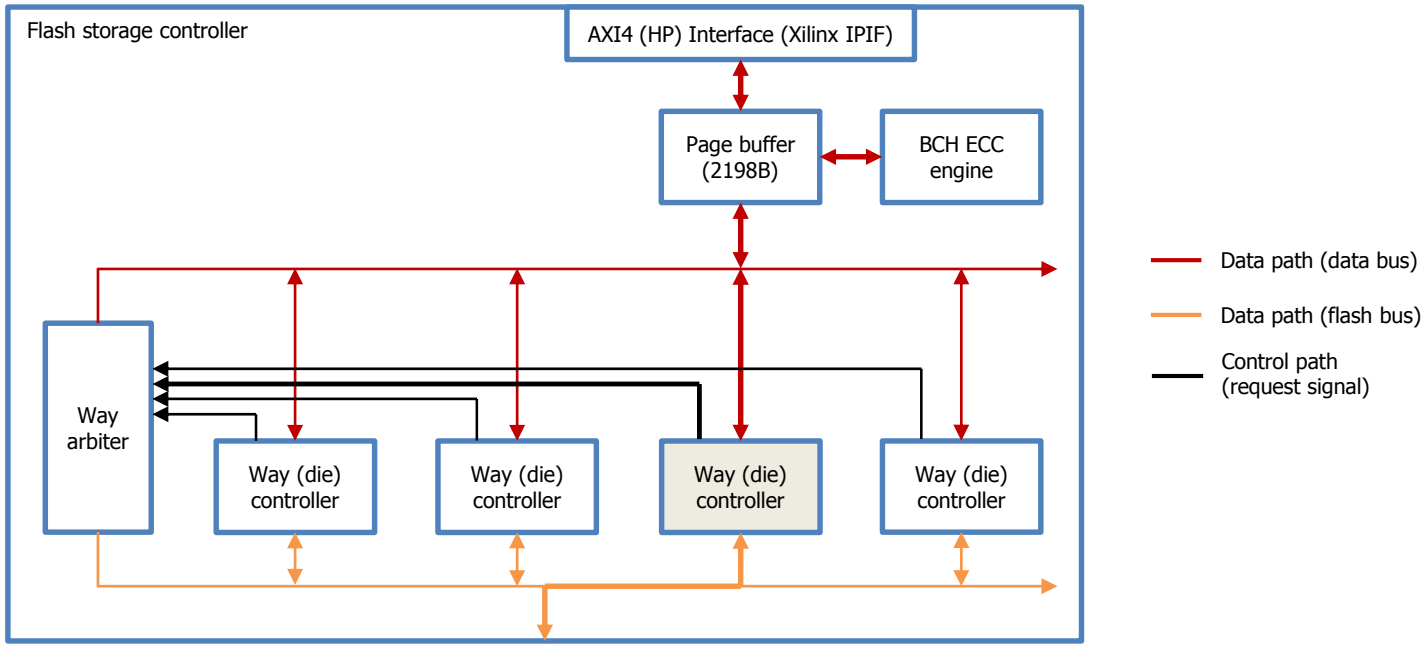


Figure 8. Connection between way arbiter and other module

Source file: ch_abt.v

Way Arbitration Example

- Scenario
  - Way controller 0, 1, 2, 3 have program, read, read, program command respectively
  - Way controllers ask permission to way arbiter simultaneously at T_0

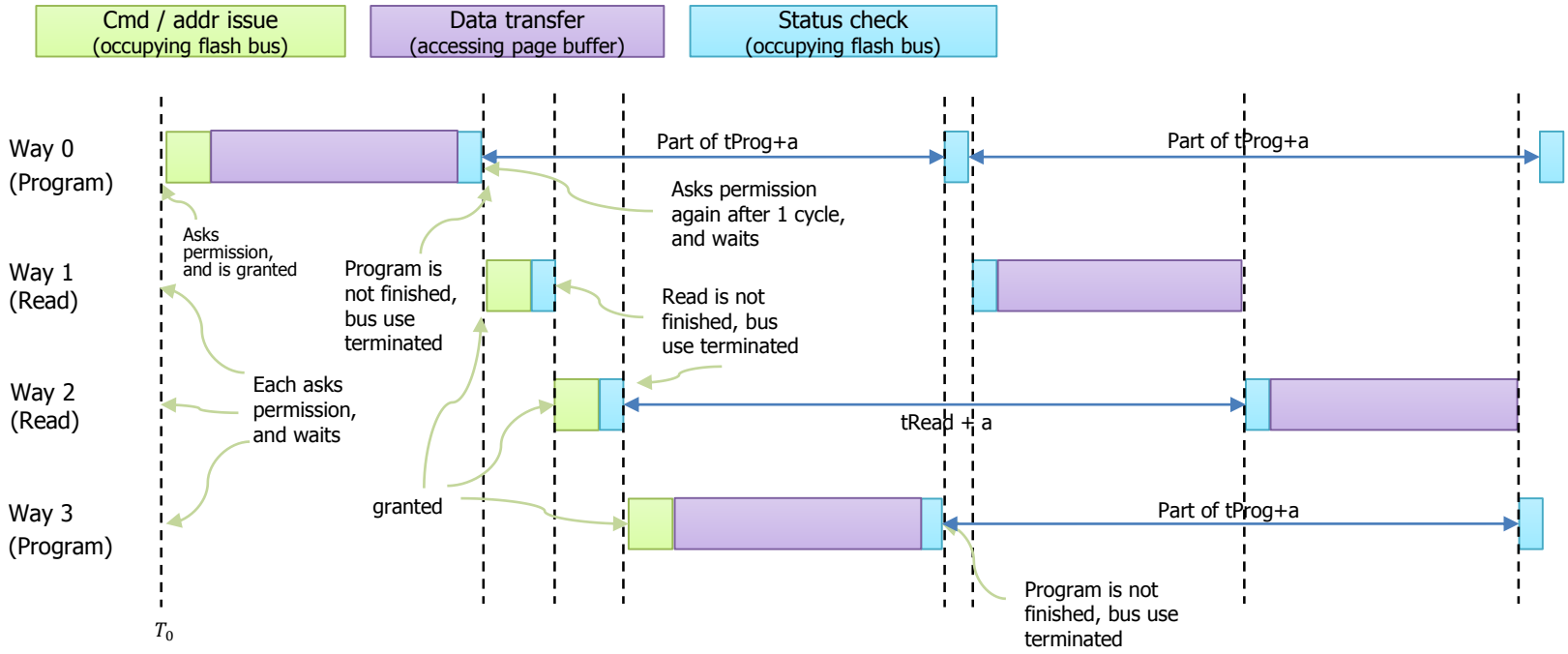


Figure 9. Sequence example

Buffer Controller

- Buffer controller controls data bus between page buffer and BCH ECC encoder/decoder
- Buffer size is 2198B
  - Data 2048B + 60B ECC parity + reserved (90B)
    - Reserved space was intended to support more powerful ECC (currently not used)
  - Page size of flash device is 8192B, thus, data transfer occurs 4 times

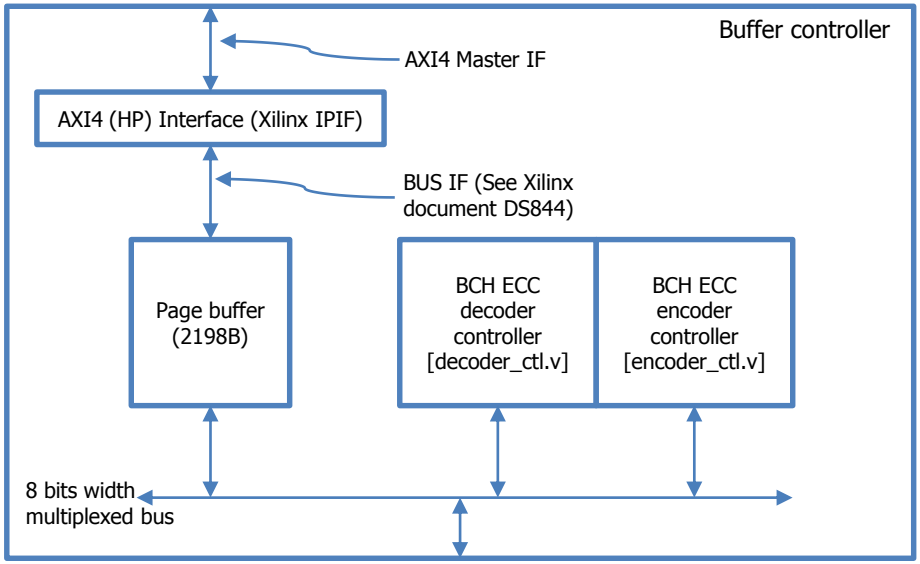


Figure 10. Buffer controller internal

Source file: page_buffer.v

Data Flow at Buffer Controller (Write)

- Data are sequentially transferred
  1. Data move from DRAM to page buffer
  2. Data are transferred to ECC encoder
  3. ECC encoder calculates parity and transfers data and parity to buffer
  4. Data are transferred to way controller

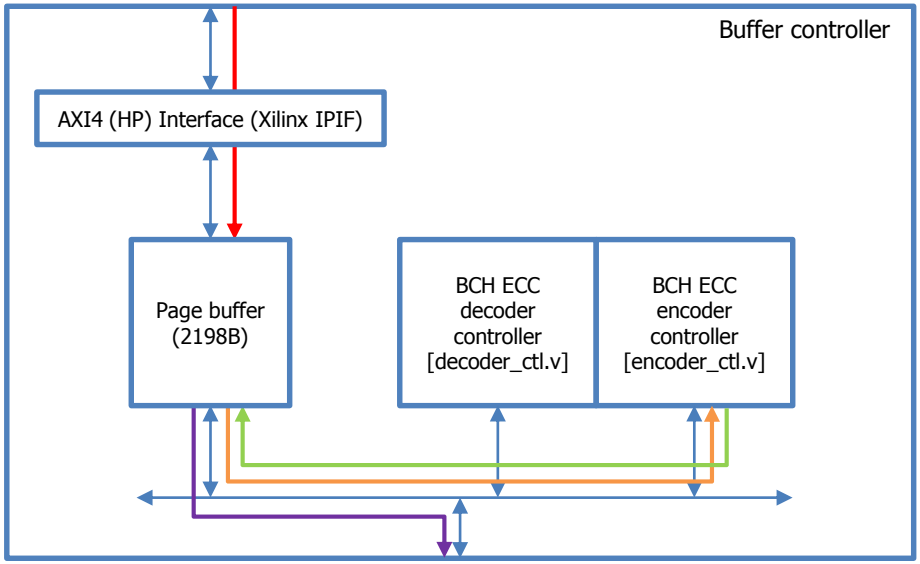


Figure 11. Buffer controller internal with data flow

Source file: page_buffer.v

Data Flow at Buffer Controller (Read)

- Data are sequentially transferred
  1. Data arrive from way controller
  2. Data are transferred to ECC decoder
  3. If there are some errors in data, ECC decoder corrects data and transfers data to buffer
  4. Data are transferred to DRAM, through Xilinx IPIF and AXI

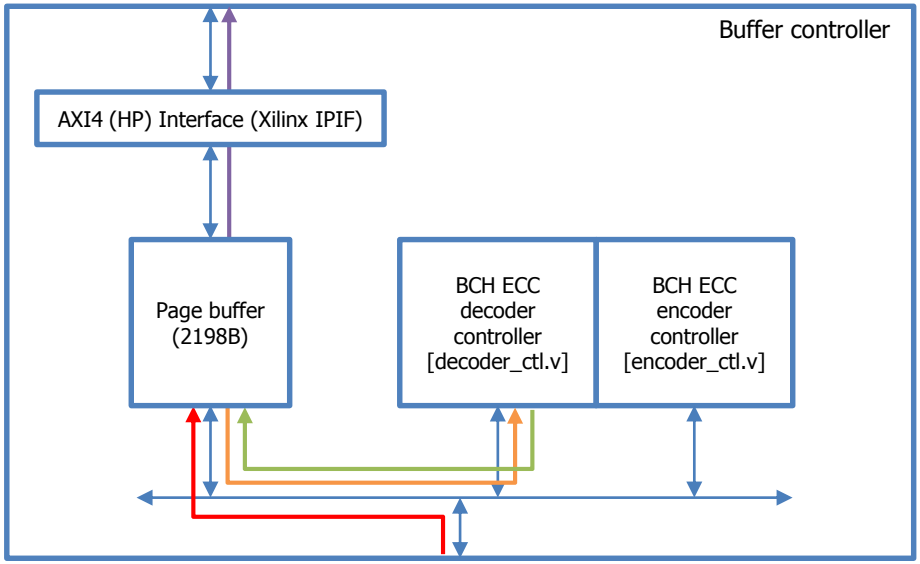


Figure 12. Buffer controller internal with data flow

Source file: page_buffer.v

Logical Page Structure

- Physical page size of flash device
  - 8KB+448B (8640B)
- There are four chunks
  - Each chunk consists of a pair of data and parity
- Reserved 40B area is intended to store FTL metadata (currently not used)

Figure 13. Page structure

Way Controller

- After the command is written to register file, the flash controller operation sequence is triggered

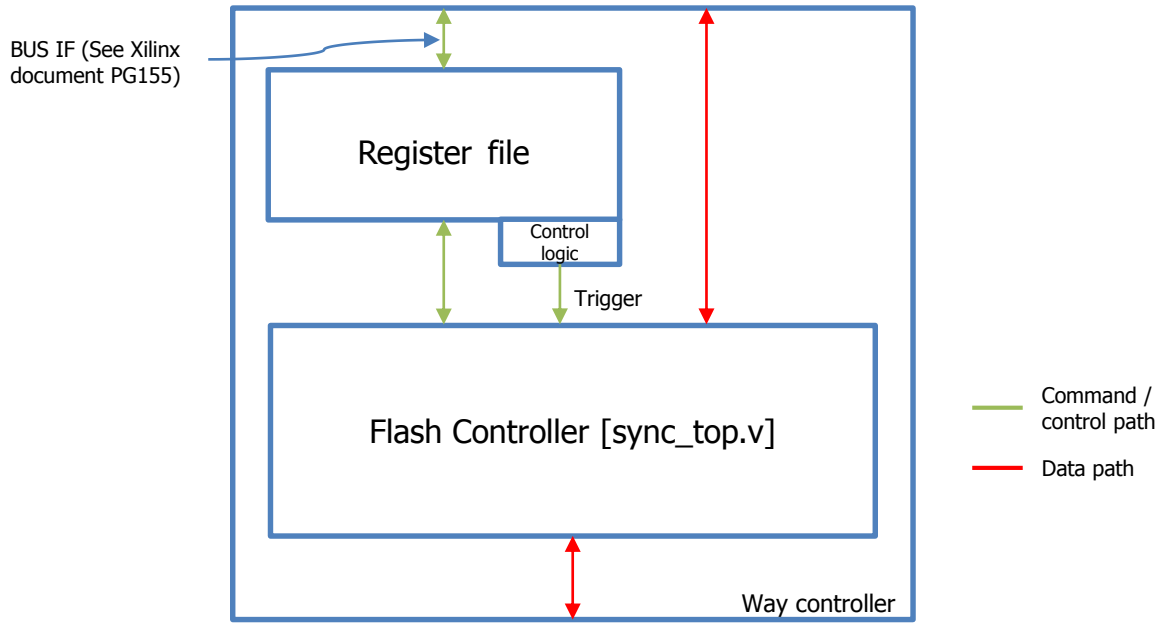


Figure 14. Way controller internal

Source file: way_top.v

Flash Controller (1/2)

- Reset, mode change commands are directly executed at their dedicated modules
  - Reset module and mode change module are responsible for the control

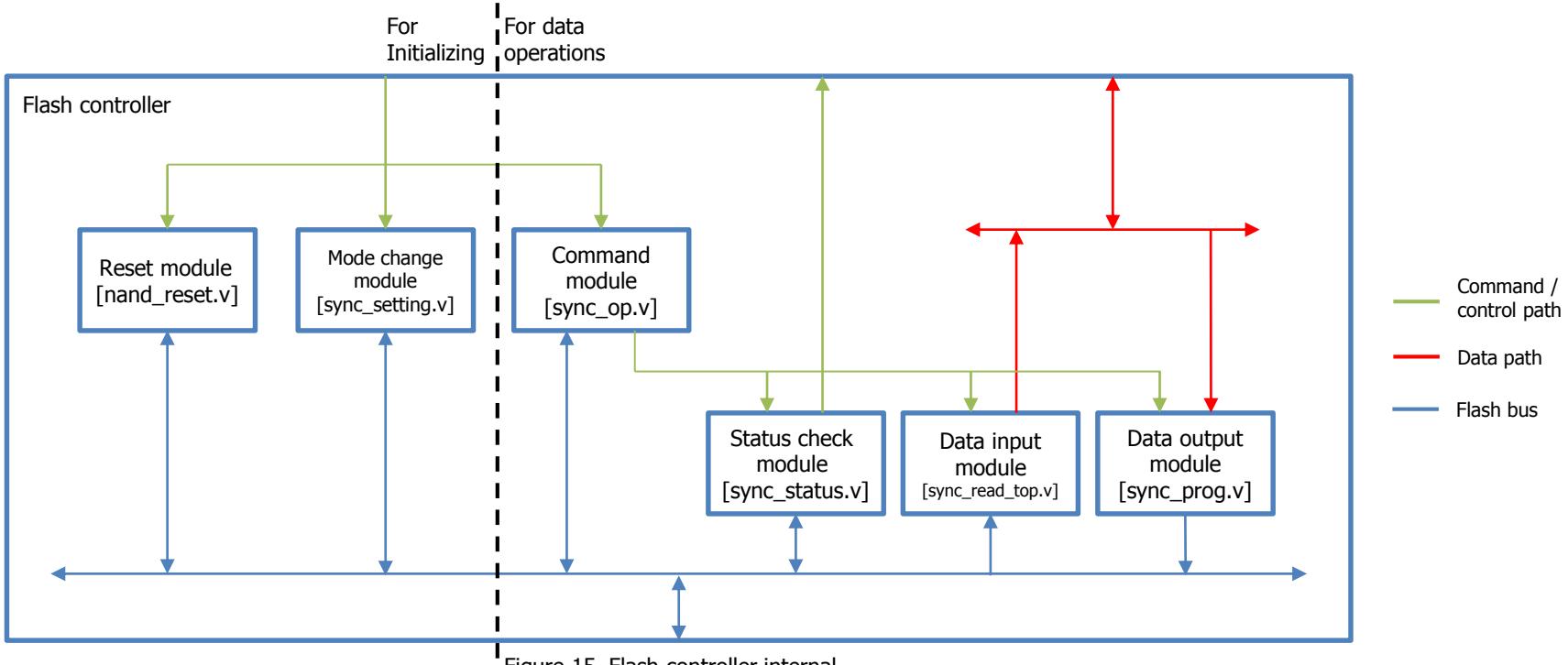


Figure 15. Flash controller internal

Source file: sync_top.v

Flash Controller (2/2)

- Page read, page program, block erase commands are executed in 2-3 stages
  1. Command module sends op code and row/column address to flash device
  2. Command module triggers data input operation(for page read) or data output operation (for page program)
  3. Command module triggers status check operation
     - Status check module returns the status of the way (die) to the command module, and writes it to register file

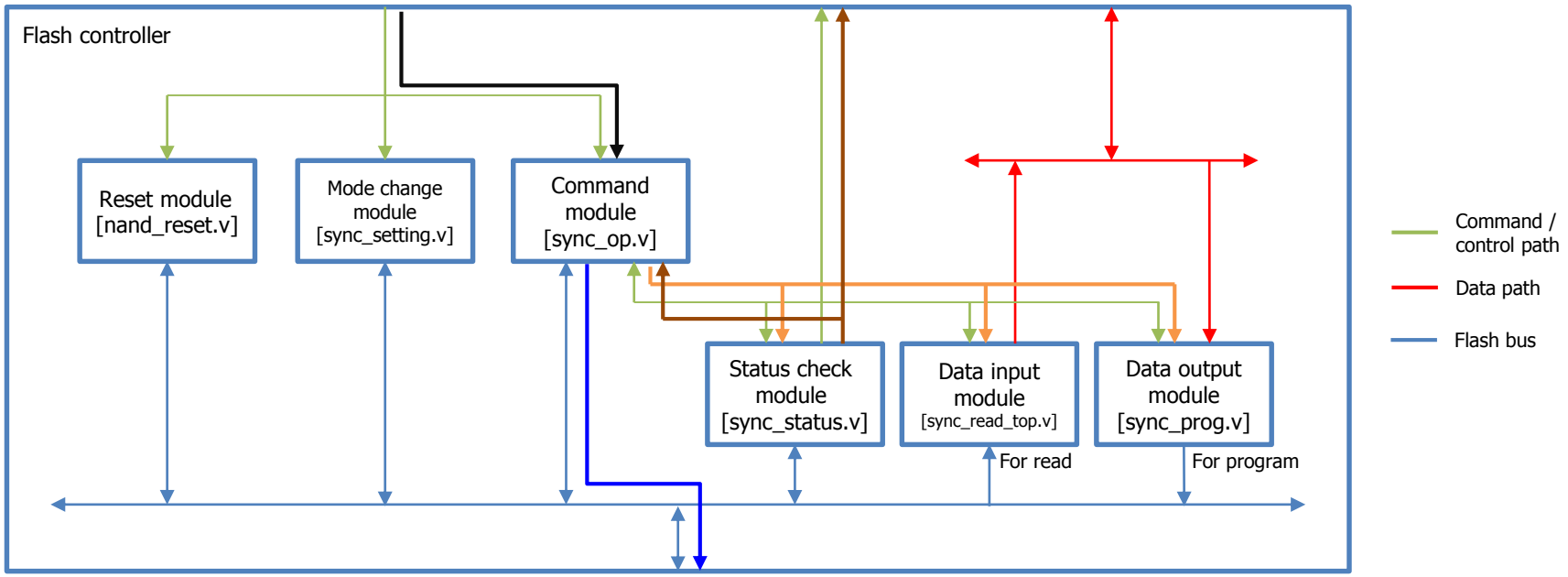


Figure 16. Flash controller internal

Source file: sync_top.v

Low Level Operation

Low Level Operation

- The operation of flash storage controller is directed by writing/reading relevant control registers
- The low level driver included in FTL includes a sequence of read and write operations to registers to run a flash storage controller

Initialization Sequence

- To use a flash storage controller, all way controllers should complete reset and mode change commands

Reset phase

Mode change phase

```
for (targetWay = 0 to numberOfWays)
    // issue reset command and wait
    *((Int32*)(NSC_BASEADDR + ((7 - targetWay) << 4) + 0x00) = SSD_CMD_RESET
    while (*((Int32*)(NSC_BASEADDR + ((7 - targetWay) << 4) + 0x00) & 0x20202020 == 0)
        wait

    // issue mode change command and wait
    *((Int32*)(NSC_BASEADDR + ((7 - targetWay) << 4) + 0x00) = SSD_CMD_MODE_CHANGE
    while (*((Int32*)(NSC_BASEADDR + ((7 - targetWay) << 4) + 0x00) & 0x20202020 == 0)
        wait
```

Code 2. Reset sequence pseudo code

Command Sequences

- Program page/read page
  1. Write control information to DRAM buffer address/flash row address registers
  2. Write program/read operation code to command register
- Reset/mode change
  1. Write reset/mode change operation code to command register
- User can check the completion status of a command
  - Read a way status register

Table 3. Status register (see flash datasheet to get more details)

Bit	31 – 8	7	6	5	4	3	2	1	0
Item	Unused (repetition of bit 7-0)	Protected (0) Writable (1)	Dcache Busy (0) Ready (1)	**Module Busy (0) Ready (1)**	R/B signal	0	0	**Previous operation has failed**	**Most recent operation has failed**

Command Sequence Example

- Scenario
  - User wants to program a page
- Steps
  1. Move data in DRAM buffer
  2. Write control information to DRAM buffer address and flash row address registers
  3. Write program operation code to command register

```
*((Int32*)(NSC_BASEADDR + ((7 - targetWay) << 4) + 0x08) = DRAM_BUF_ADDR
*((Int32*)(NSC_BASEADDR + ((7 - targetWay) << 4) + 0x0C) = FLASH_ROW_ADDR
*((Int32*)(NSC_BASEADDR + ((7 - targetWay) << 4) + 0x00) = SSD_CMD_PROG
```

Code 3. Issuing program page command pseudo code

  4. Check completion status of the command

```
while (*((Int32*)(NSC_BASEADDR + ((7 - targetWay) << 4) + 0x00) & 0x20202020 == 0)
```

Code 4. Polling status register pseudo code

  5. Check success of the completed command

```
if (*((Int32*)(NSC_BASEADDR + ((7 - targetWay) << 4) + 0x00) & 0x03030303 == 0)
    reportSuccess()
else
    reportFailure()
```

Code 5. Detect failure pseudo code

Configuration of the number of ways

- Default configuration supports 4-way flash memories (4 dies) in a module
- You need to modify ch_top.v file to enable/disable other way controllers
  - Ex) 2-way configuration

```
assign o_m_ch_cmplt = &{w_m_ch_cmplt[0], w_m_ch_cmplt[1]};
```

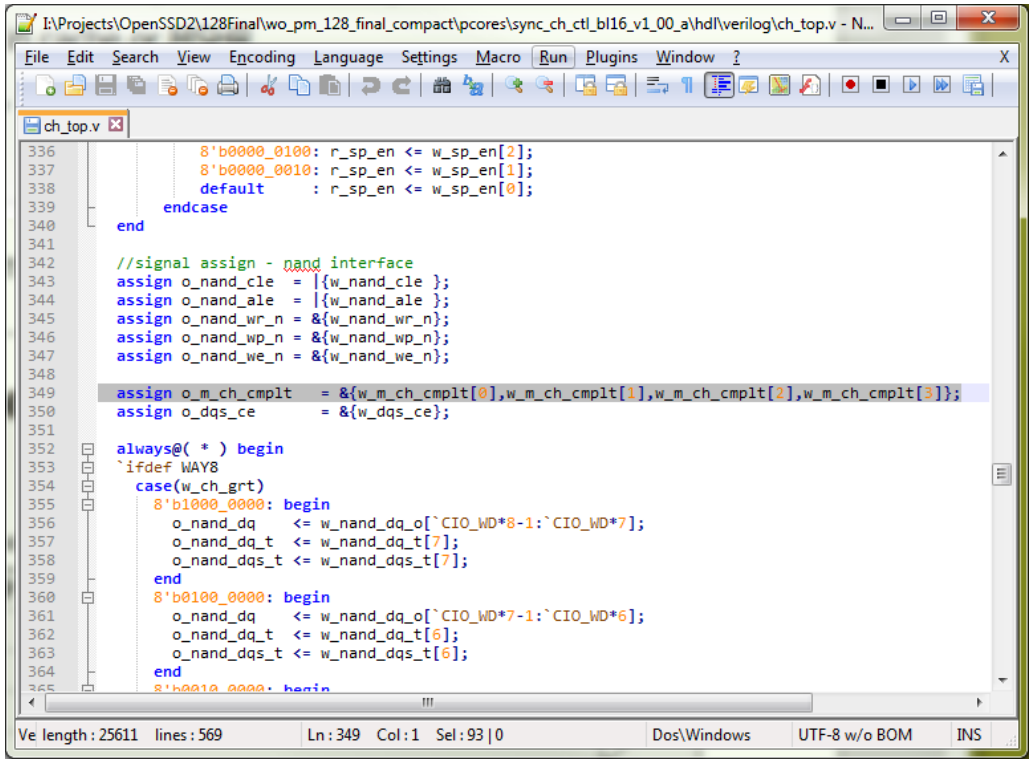


Figure 17. way configuration

Note

- The development environment was Xilinx ISE design suite 14.7 system edition
  - The main IC XC7Z045 (Zynq-7045) requires system edition license
- Design of this version (v1.0.0) has some known issues
  - Clock domain synchronization issue at system AXI bus
  - Fragile NAND PHY module
  - Refer to the related article in Cosmos OpenSSD forum for further details

Reference

- Xilinx documents
  - LogiCORE AXI4 Lite IPIF
    http://www.xilinx.com/support/documentation/ip_documentation/axi_lite_ipif/v2_0/pg155-axi-lite-ipif.pdf
    - Xilinx bus IF used in storage controller (channel controller)
  - LogiCORE AXI4 Master Burst IPIF
    http://www.xilinx.com/support/documentation/ip_documentation/axi_master_burst/v1_00_a/ds844_axi_master_burst.pdf
    - Xilinx bus IF used in buffer controller
  - Platform Specification Format Reference Manual
    http://www.xilinx.com/support/documentation/sw_manuals/edk10_psf_rm.pdf

Authors

Name	E-mail	Contribution
Jaehyeong Jeong	jhjeong@enc.hanyang.ac.kr	2013-08 ~ 2014-01
Taeyeong Huh	tyhuh@enc.hanyang.ac.kr	2014-01 ~ 2015-01
Kibin Park	kbpark@enc.hanyang.ac.kr	2015-02 ~ Now

Cosmos OpenSSD Platform Tutorial

HW ECC Engine for Flash Storage Controller

ENC Lab. @ Hanyang University

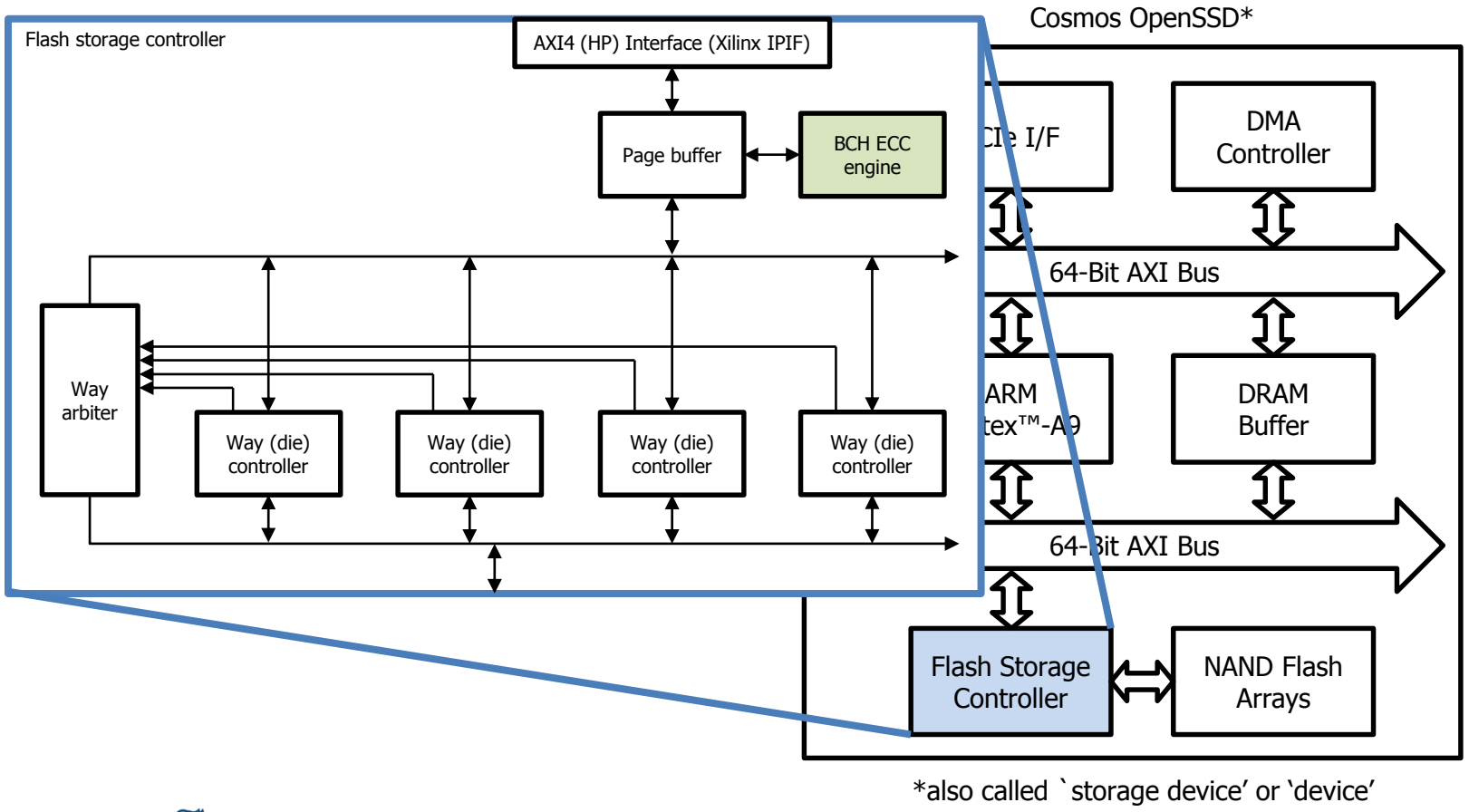
HANYANG UNIVERSITY
ENC Lab
Embedded Network Computing Lab.

Document Revision History

- v1.0.0
  - Initial release
  - Contents are based on the design release of ECC_engine_v1.0.0

Section Scope

- ECC engine is implemented inside storage controller



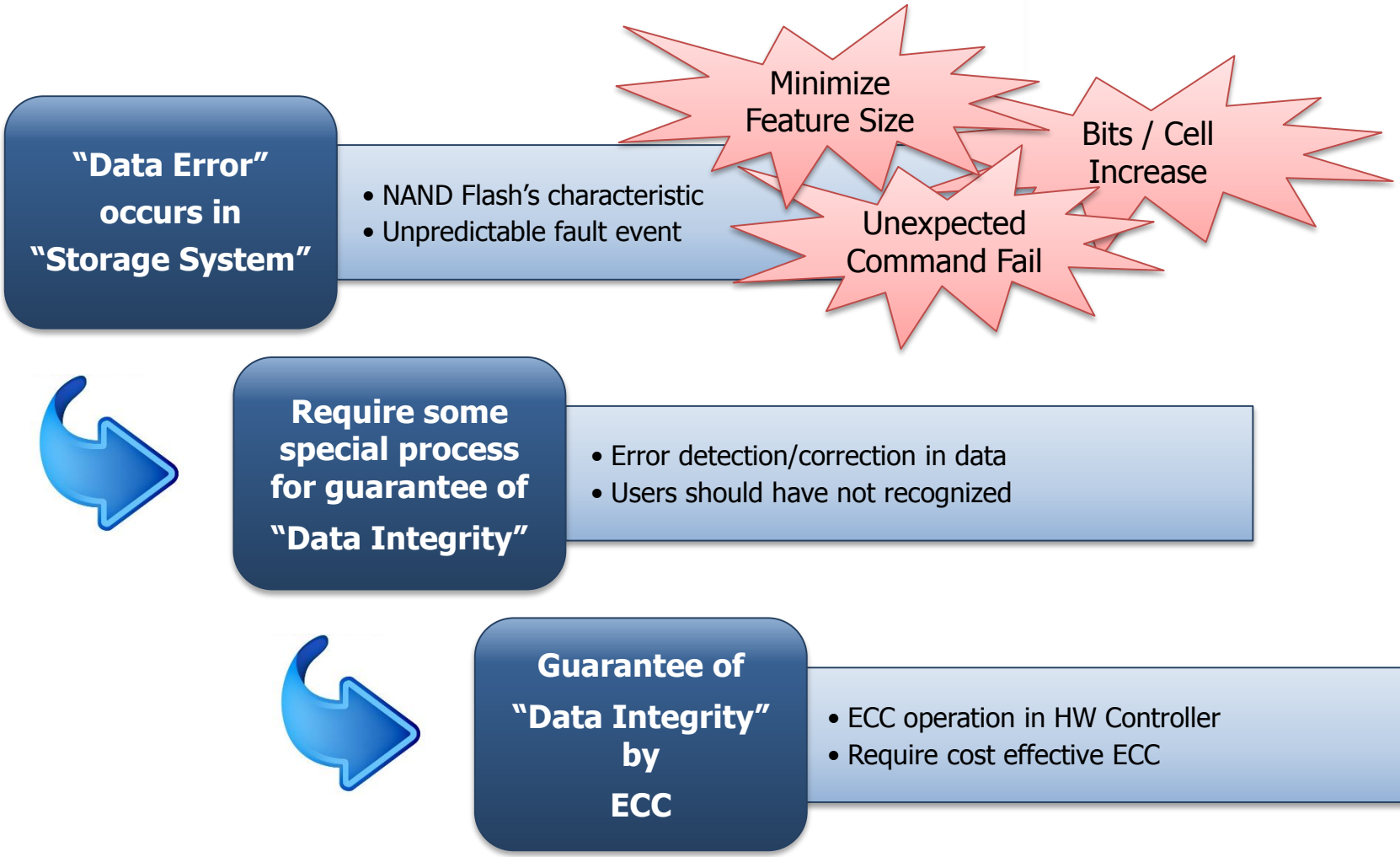
*also called `storage device' or 'device'

INDEX

HANYANG UNIVERSITY
ENC Lab
Embedded Network Computing Lab.

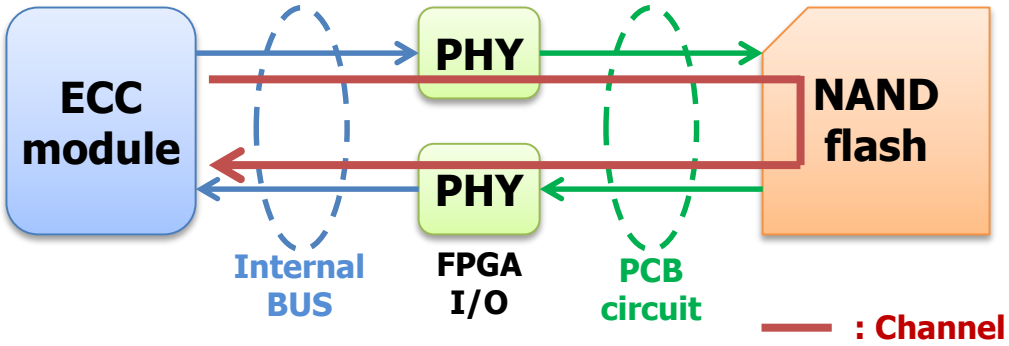
ECC Introduction

Necessity of Error Correction Code (ECC)



Concept of Channel

- Channel
  - The medium that transmits the information signal in time and space
    - Transmission : passing information to a different location from the current location (space)
    - Storage : passing information from current to the future (time)
- Channel coding
  - Providing reliability to meet the required level
- Channel in storage system



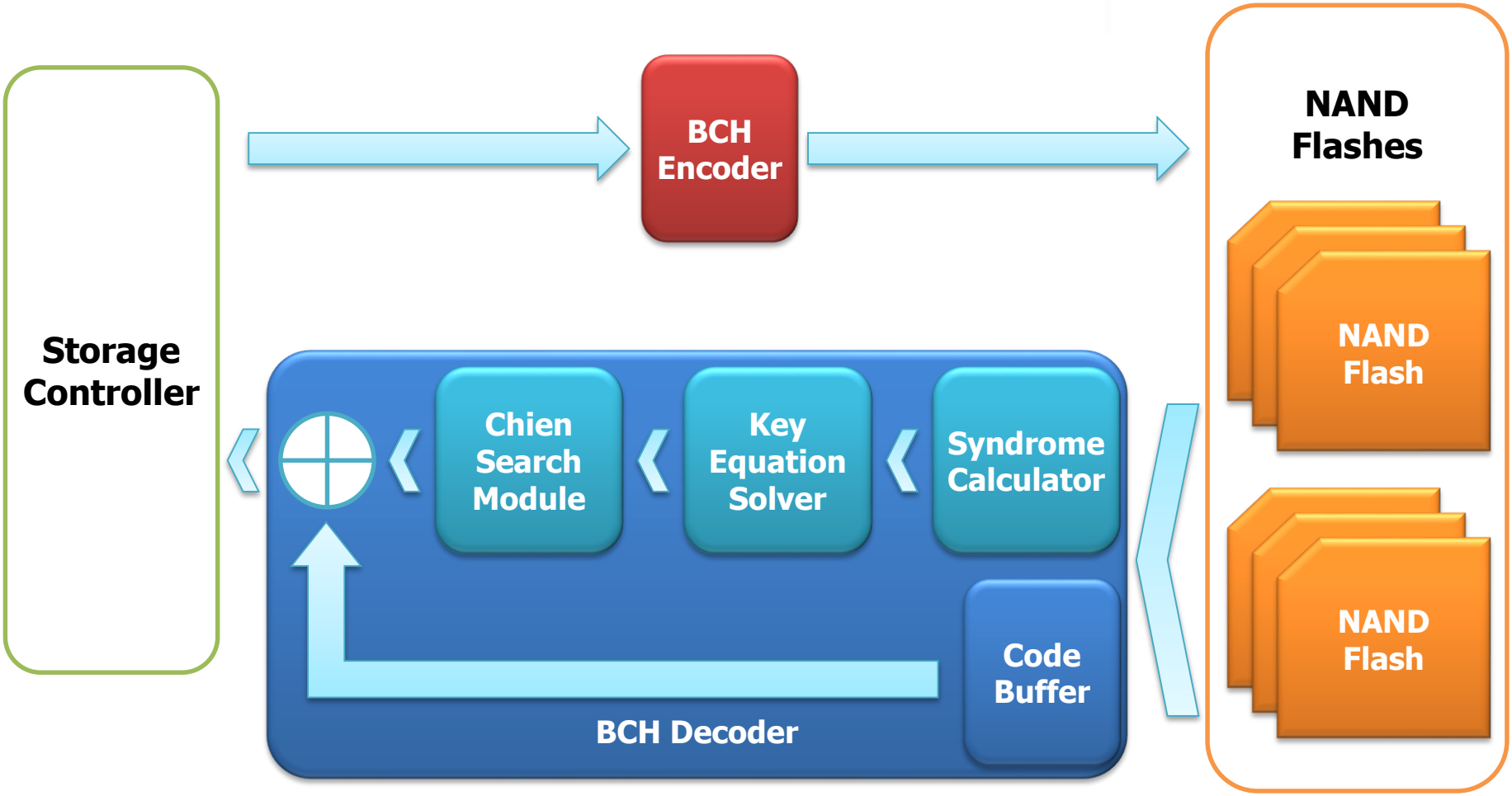
Types of Channel Coding

- Error Detection Code (EDC)
  - Parity check
  - CRC (Cyclic Redundancy Check)
- Error Correction Code (ECC)
  - Block code
    - Hamming code
    - BCH code
    - Reed-Solomon code
  - Convolution code
    - Turbo code
    - Low-Density Parity Check Code (LDPC)

Characteristics of BCH Code

- Good for correcting random errors
  - NAND flash may have random errors
  - Note that Reed-Solomon code is suited for sequential error correction (optical disk, tape media, etc.)
- Separate parity code
  - Appended to data to user data
  - Bit error in data and parity can be detected/corrected
- Easy implementation
  - Easier to understand than Low-Density Parity Check Code (LDPC) algorithm

Flow of BCH code

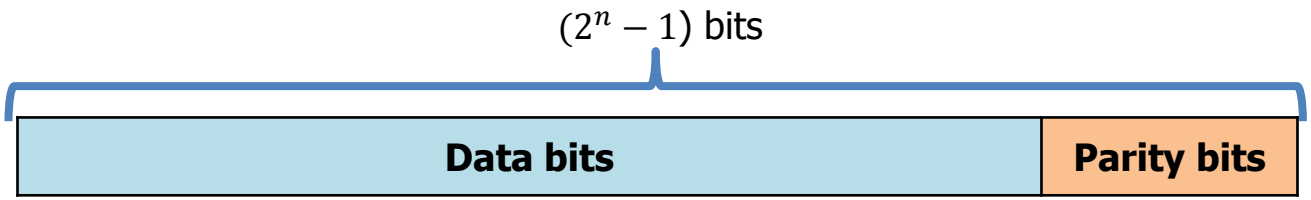


Mathematical Backgrounds

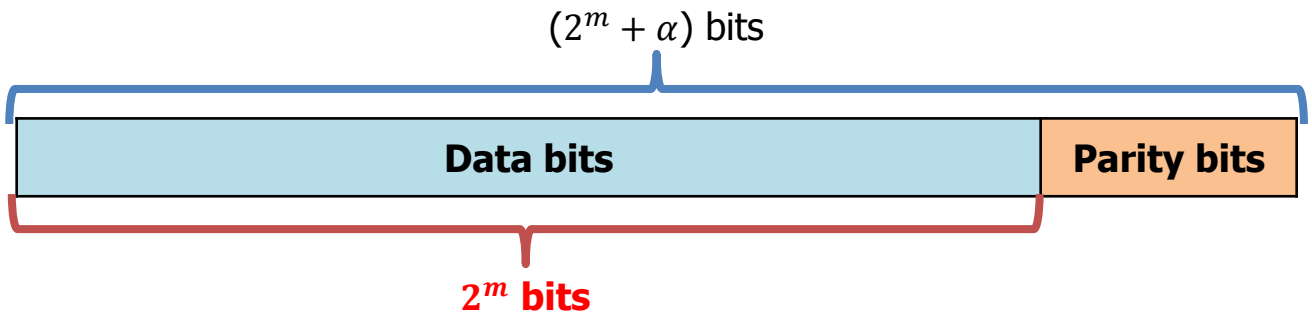
- Galois field
  - Primitive element: α
  - All the elements of the Galois field are a form of power α
- Primitive polynomial
  - Equation with a solution α^i
  - Define the elements of Galois field
- Minimal polynomial
  - Generated by cyclotomic coset matrix
  - Polynomial to obtain the Generator polynomial ($g(x)$)

Shortened BCH Code

- Normal BCH code
  - Code length = data + parity = $2^n - 1$ bits



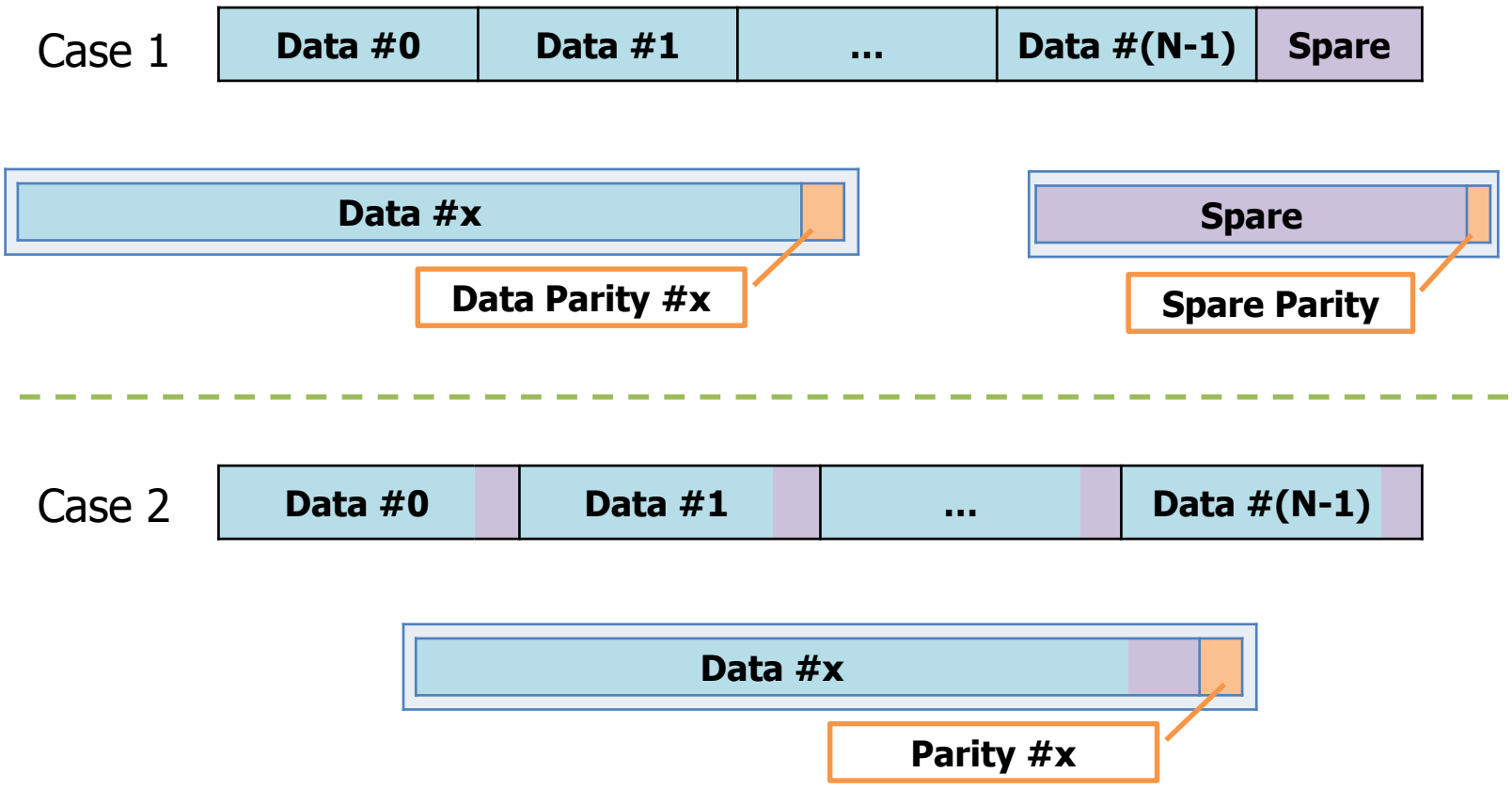
- Shortened BCH code
  - Code length = data (2^m) + parity (α) = $(2^m + \alpha)$ bits $< 2^{m+1} - 1$



Page Structure/BCH Code Design

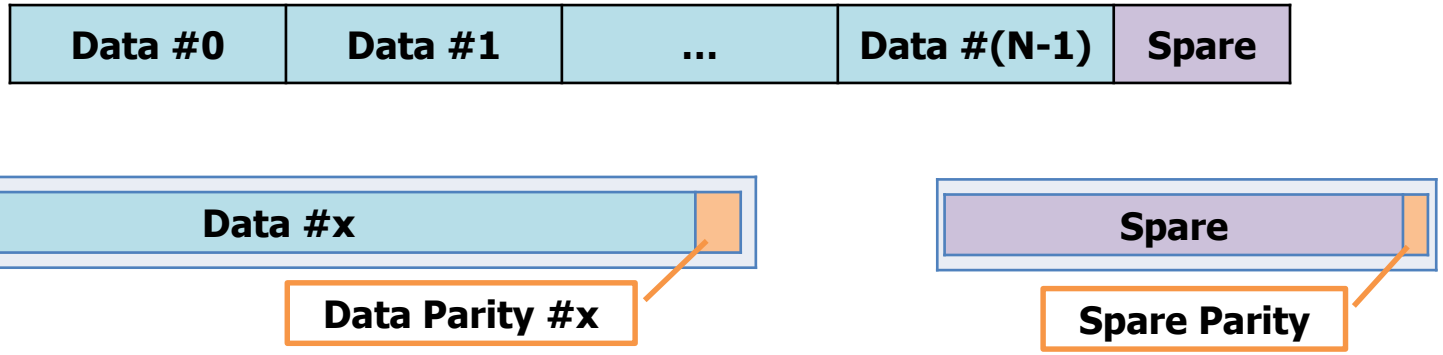
Page Structure Design

- Page structure can be selected by the designer



BCH Code Design

- Design factors
  - NAND flash characteristics
    - Page size: data/spare area size
    - Physical characteristic: ECC requirement data from manufacturer
  - System requirement
    - Spare capacity requirement
  - Hardware cost
    - Chunk size
    - Error correction capability



BCH Engine Specification

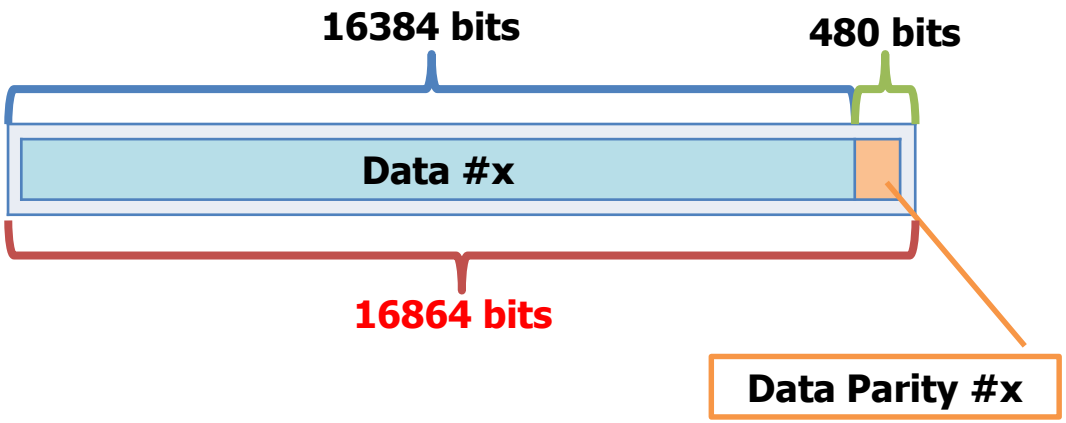
HANYANG UNIVERSITY
ENC Lab
Embedded Network Computing Lab.

BCH Code Specification

- BCH code design with 8KB page NAND flash

Table 1. Specification of BCH code (n, k, t)

Codeword length, n	2 KB + 60 B = 16864 bits
Message length, k	2 KB = 16384 bits
Parity length, (n-k)	60 B = 480 bits
Correction capability, t	32 bits per chunk
Galois field size	GF(2^15)



BCH Engine Specification (1/2)

Table 2. Parallel level of BCH engine

Encoder	Input	8 bits per cycle
	Output	
Decoder	Input	
	Output	

Table 3. Main method of BCH engine

Encoder	-	Modified LFSR (mLFSR: LFSR + shifter)
Decoder	SC	LFSR, evaluation matrix
	KES	Binary inversion-less BM (iBM.b)
	CS	Evaluation matrix

BCH Engine Specification (2/2)

Table 4. Hardware cost of BCH engine

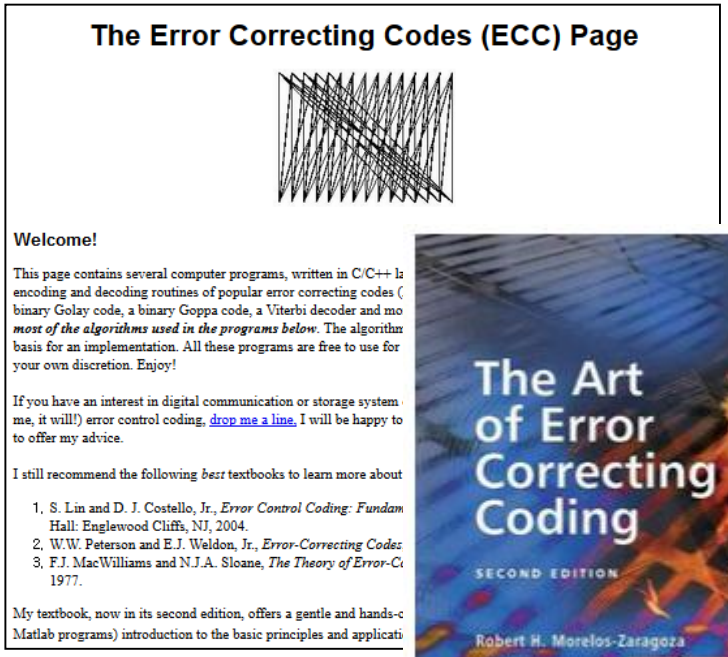
			LUT	Registers	Maximum frequency (MHz)
Xilinx Zynq 7045 FFG900-3			218,600	437,200	-
Encoder	-	-	1,152 (0%)	511 (0%)	399.488
Decoder	Total	-	21,495 (9%)	8,973 (2%)	337.277
	-	SC	2,297 (1%)	1,238 (0%)	355.492
		KES	**14,936 (6%)**	**6,665 (1%)**	**341.250**
		CS	4,335 (1%)	1,094 (0%)	337.701

* Synthesis results with Xilinx ISE 14.7

SW Implementation

Reference Source Code for BCH Code

- Reference C source code
  - http://www.eccpage.com/, 5. BCH codes, bch3.c
- Functions in the reference code
  - read_p();
  - generate_gf();
  - gen_poly();
  - encode_bch();
  - decode_bch();
- Usage
  - Generate BCH code parameter
  - Generate HDL code
  - Generate golden vector and formula for verification



Source Code Modification

- Function decomposition
  - Decomposing a monolithic function into small sub-functions for step-by-step hardware design
- Add text file output function
  - print_specification(): output parameter, information and etc.
  - data_gen(), err_gen(): generate golden vector and temporal formula
- Generate HDL code
  - ~code_gen(): generate HDL(verilog) code
- iBM algorithm
  - The reference code uses Berlekamp-Massey (BM) algorithm
  - Inversion-less BM (iBM) algorithm is added

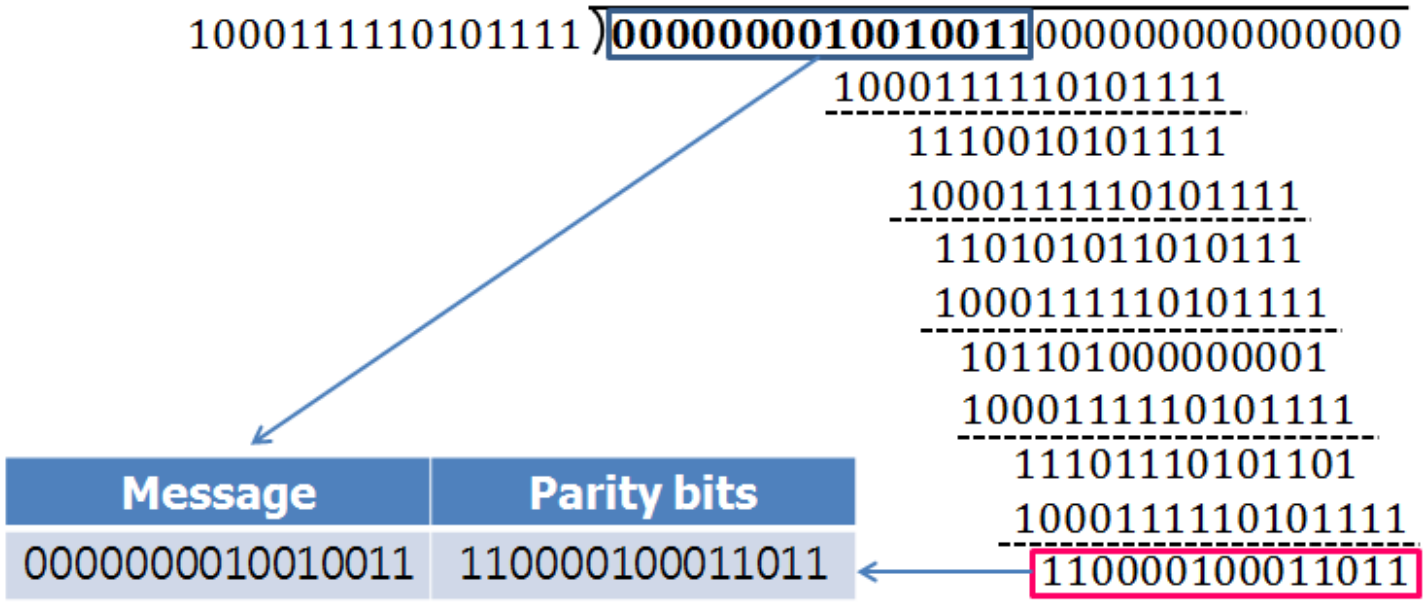
HW Implementation I: Encoder

Glance at Encoder Operation

- $f(x)\,x^n = g(x)Q(x) + r(x) \Rightarrow f(x)x^n + r(x) \equiv 0\ (mod\ g(x))$

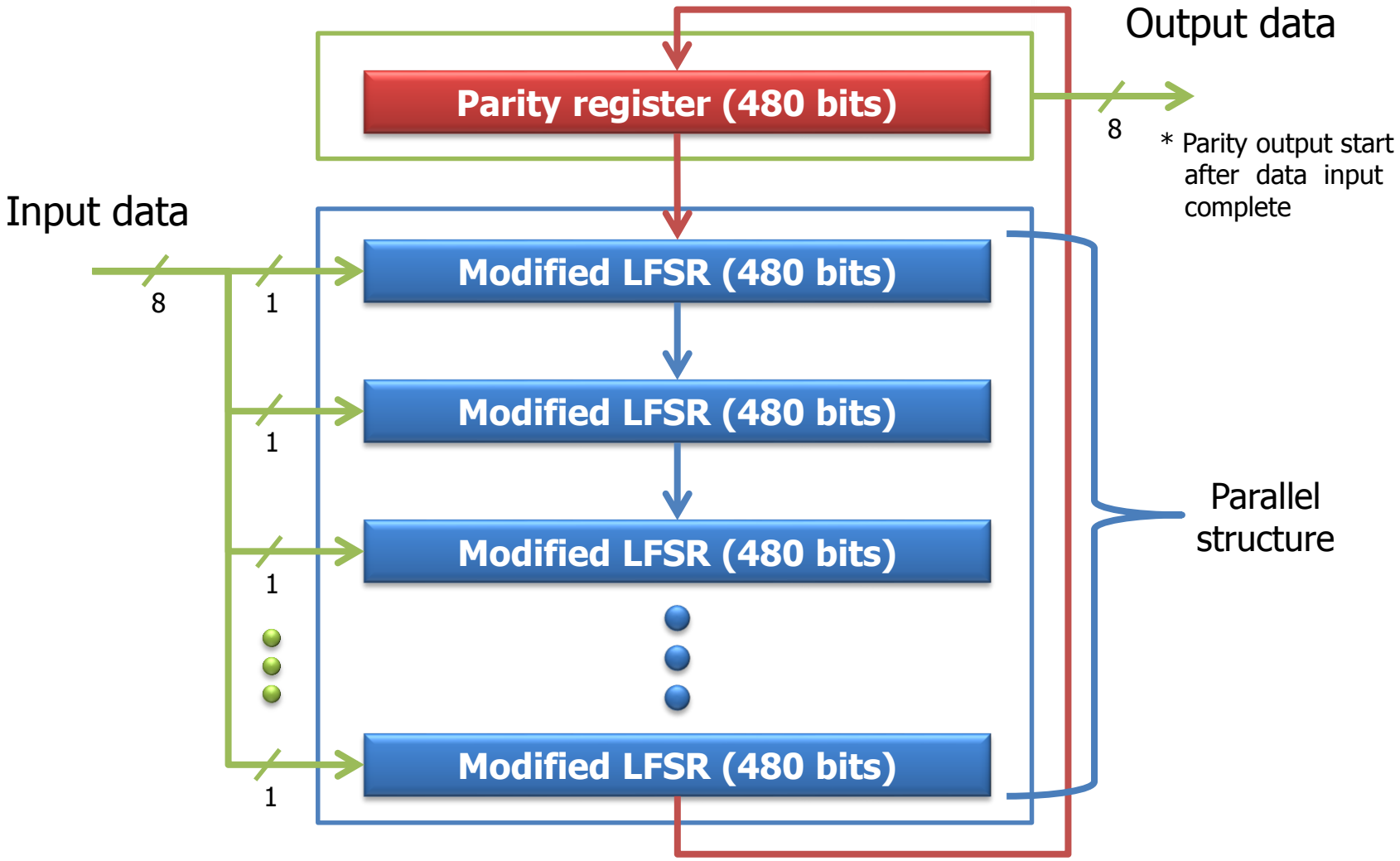
EX) C(31, 16, t=3) =>

code word = 31bits, message = 16bits, correction capability = 3 bits



Source file: serial_m_lfs_XOR.v

Encoder Block Diagram



LFSR: Linear Feedback Shift Register

* 2KB chunk, 32bit error correction, 8bit parallel level

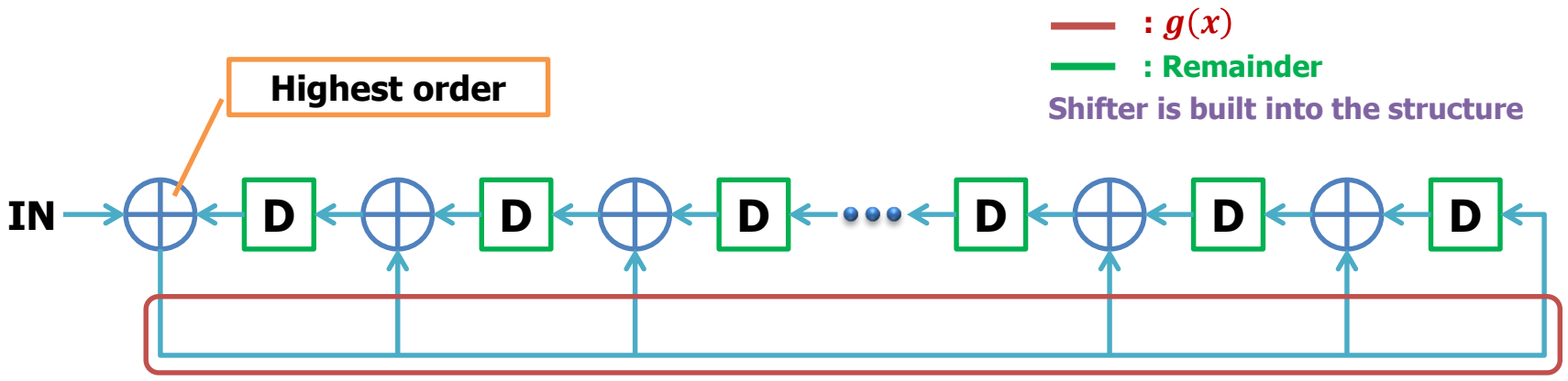
Source file: encoder.v, parallel_m_lfs_XOR.v

Modified Linear Feedback Shift Register (mLFSR)

- Shift and divide
  - mLFSR shifts original data by a predetermined number of bits and divides by $g(x)$
  - mLFSR operates in series but it can be composed in parallel

- $f(x)x^n = g(x)Q(x) + r(x)$ → remainder

  Shift: x^n — divide by $g(x)$: $g(x)$



Source file: serial_m_lfs_XOR.v

HW Implementation II: Decoder

Polynomial Arithmetic

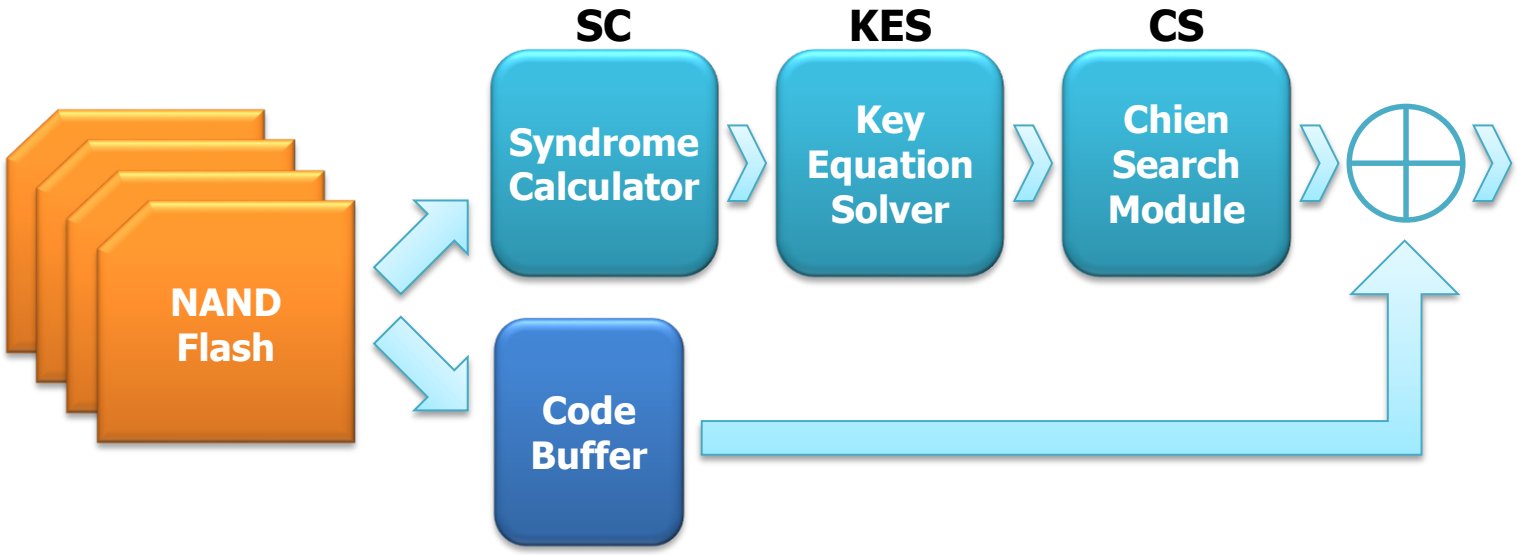
- 4 basic operations
  - Addition/Subtraction
    - Bit-wise XOR
      example)

$$f(x) = x^7 + x^4 + x^3 + x^2 + x$$
$$g(x) = x^6 + x^5 + x^4 + x^3 + x$$
$$f(x) + g(x) = x^7 + x^6 + x^5 + x^2$$

  - Multiplication/Division
    - Use XOR instead of ADD
- Widely used in BCH implementation

Decoding Flow

- BCH decoding flow



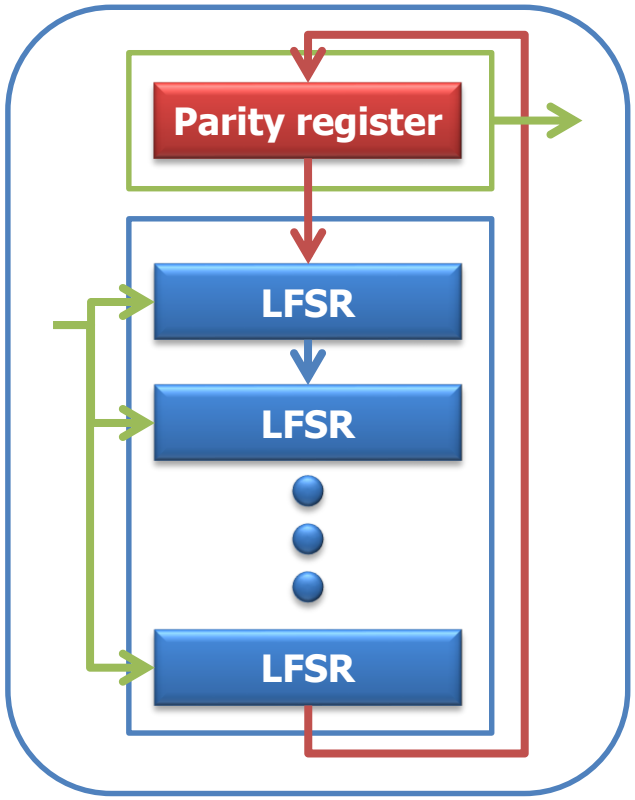
- Syndrome Calculator (SC): check error in received code by generating syndromes
- Key Equation Solver (KES): create an error locator polynomial to locate bit error
- Chien Search module (CS): solve the error locator polynomial, and find out error position

SC Implementation Methods

- Method 1. Substituting α^i to read data $R(x)$
  - $R(\alpha^i) = S_i$
- Method 2. Divide the read data by the $M_i(x)$, substituting the α^i to the rest
  - $R(\alpha^i) = \{M_1(\alpha^i)M_3(\alpha^i) \dots M_i(\alpha^i) \dots M_{2t}(\alpha^i)\}\, Q(\alpha^i) + e(\alpha^i)$
    $= M_i(\alpha^i)Q'(\alpha^i) + e'(\alpha^i) = 0 + S_i = S_i$
- Method 3. Divide the read data in the $g(x)$, substituting the α^i
  - $R(\alpha^i) = g(\alpha^i)Q(\alpha^i) + e(\alpha^i) = 0 + S_i = S_i$

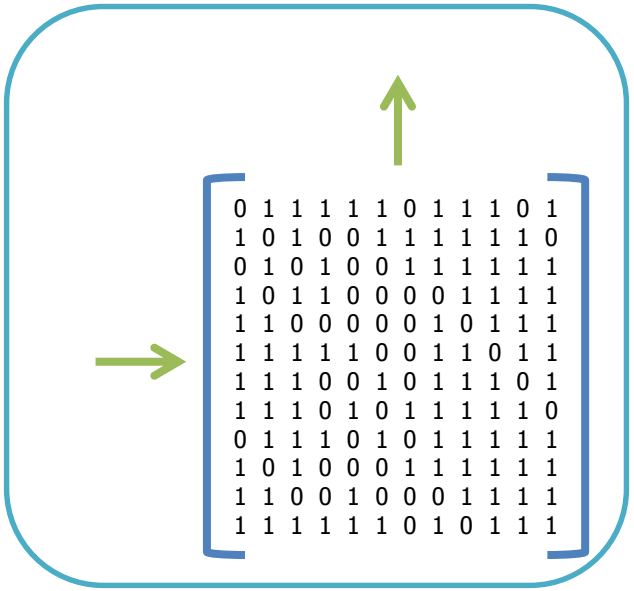
* The released version is implemented using the second method

SC Block Elements



Divide module $\boldsymbol{i}$

$$f(x) = M_i(x)Q(x) + r(x)$$



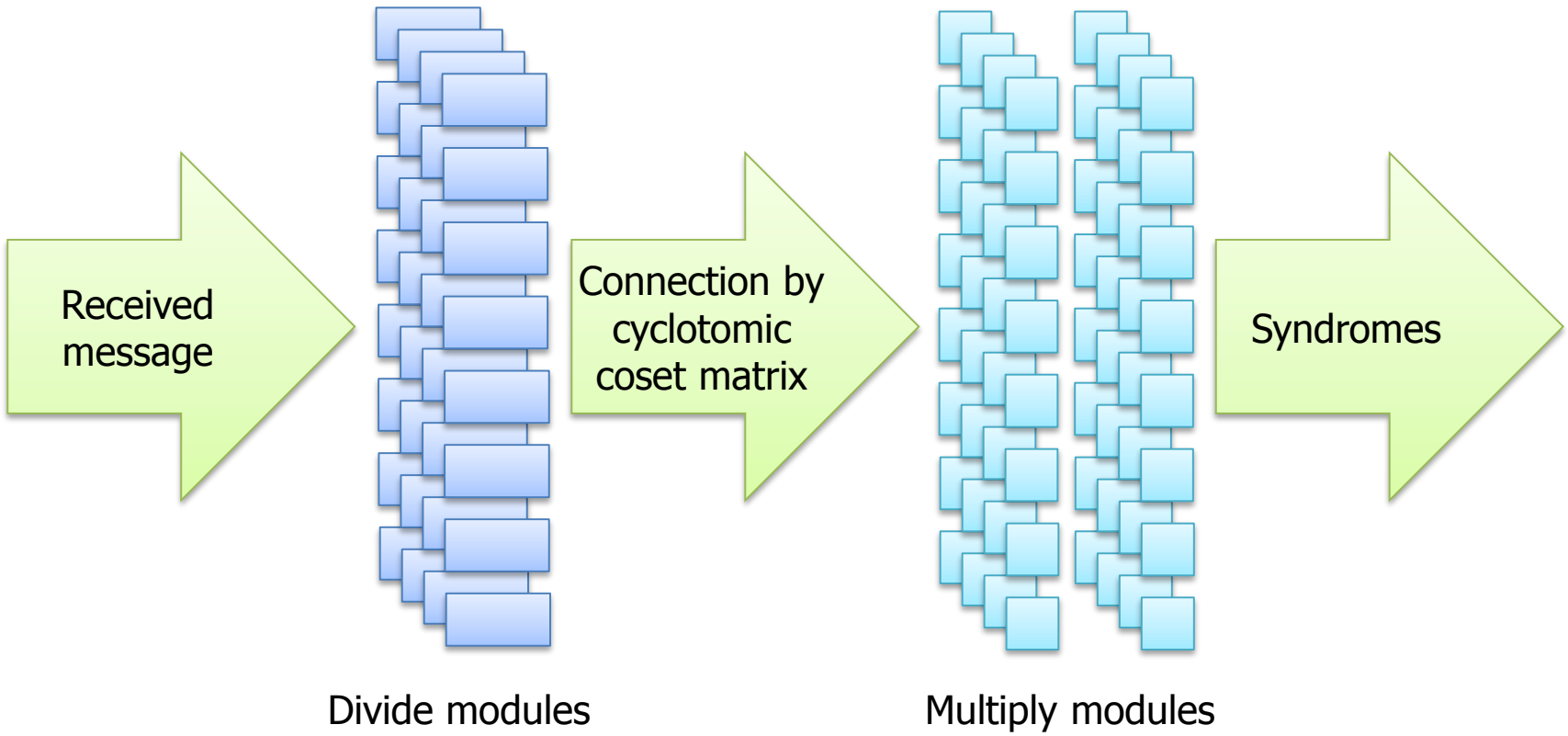
Multiply module $\boldsymbol{i}$

Substituting α^i the $r(x)$,
Deriving syndromes

Source file: sc_dividers_p_lfs_XOR.v, sc_evaluation_matrices.v

SC Block Diagram

- Operation flow of SC



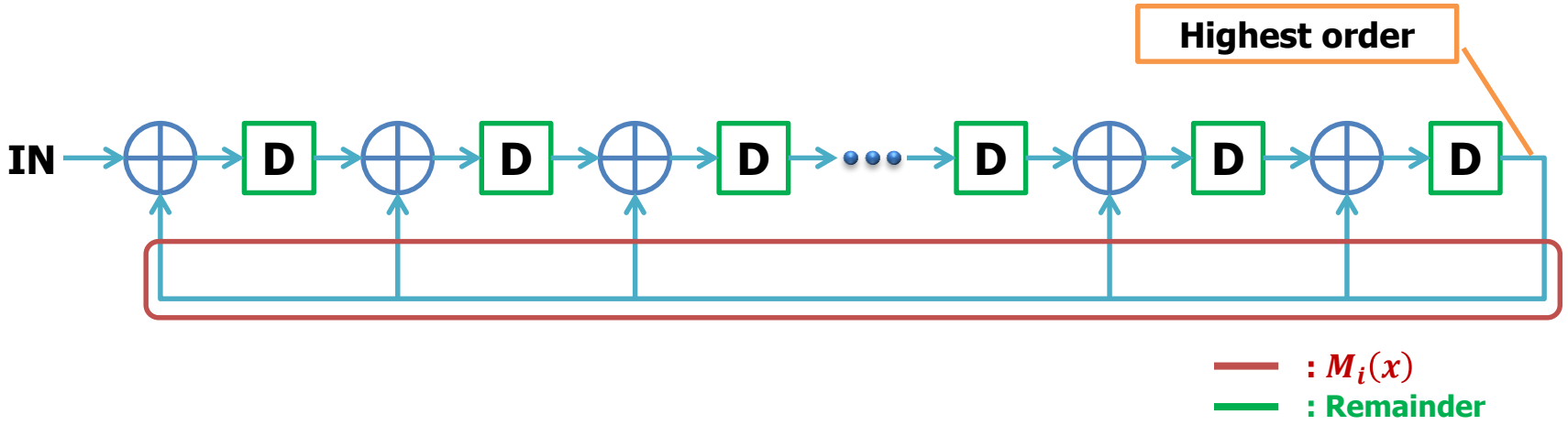
Source file: syndrome_calculator.v

SC Linear Feedback Shift Register (LFSR)

- Divide
  - LFSR divides received data by $g(x)$
  - LFSR operates in series but it can be composed in parallel

- $R(x) = M_i(x)Q(x) + r'(x)$ → remainder

  divide by $M_i(x)$



SC Evaluation Matrix (1/2)

- Substituting a specific element in the formula represented by polynomial form
  - Performing **Multiplication** and **Addition** together

For example, assuming $r(x) = r_0 + r_1 x + r_2 x^2 + r_3 x^3$

$$S_3(x) = r_0 + r_1(\alpha^3) + r_2(\alpha^3)^2 + r_3(\alpha^3)^3 = r_0 + r_1\alpha^3 + r_2\alpha^6 + r_3\alpha^9$$
$$= r_0 + r_1 a^3 + r_2(\alpha^3 + \alpha^2) + r_3(\alpha^3 + \alpha)$$
$$= r_0 + r_3\alpha + r_2\alpha^2 + (r_1 + r_2 + r_3)\alpha^3$$

Therefore, $s_0 = r_0,\ s_1 = r_3,\ s_2 = r_2,\ s_3 = (r_1 + r_2 + r_3)$

In matrix form…

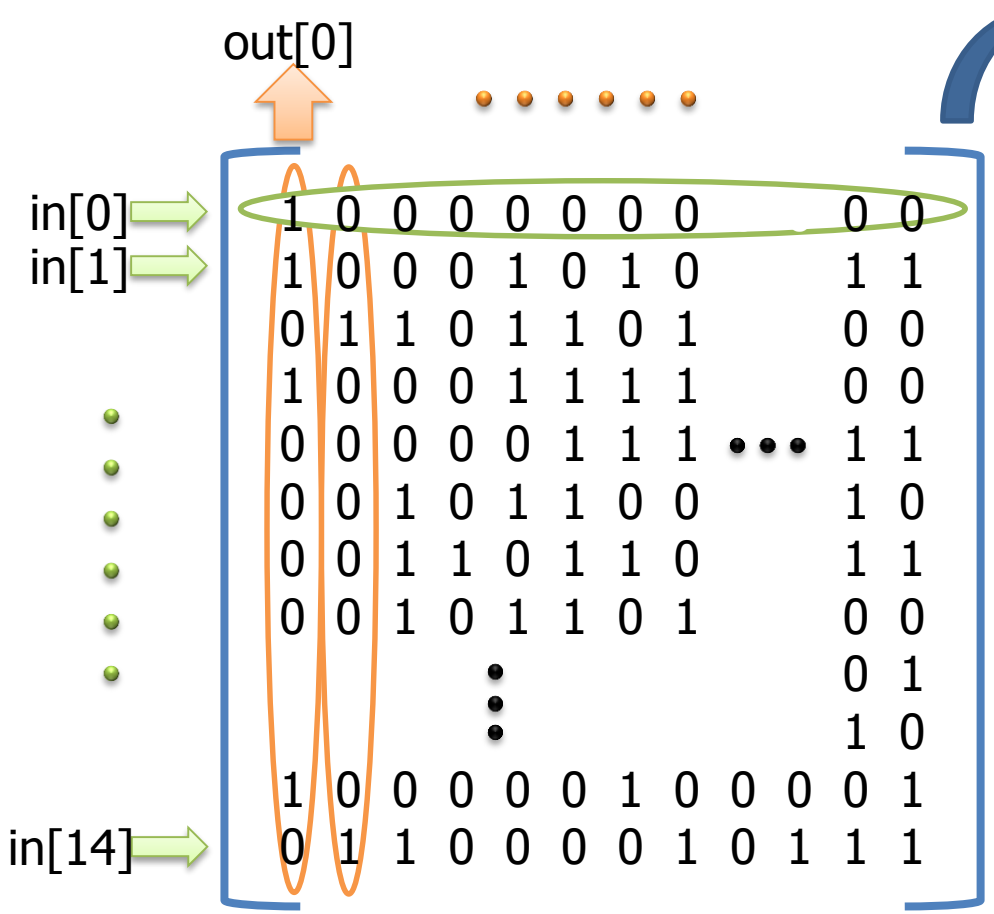
$$(s_0, s_1, s_2, s_3) = (r_0, r_1, r_2, r_3) \cdot \begin{pmatrix} 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 1 \\ 0 & 1 & 0 & 1 \end{pmatrix}$$

→ Evaluation matrix

Source file: sc_evaluation_matrices.v

SC Evaluation Matrix (2/2)

- Matrix programming example




```
module sc_evaluation_matrix_???(in, out);

…

input wire [14:0] in;
    output wire [14:0] out;

    assign out[0] = in[0] ^ in[1] ^ in[3]
                        ^ in[13];

    assign out[1] = in[2] ^ in[14];

…

endmodule
```

Source file: sc_evaluation_matrices.v

KES Algorithms

- Error location polynomial
  - $\Lambda(x) = 1 + \lambda_1 x + \lambda_2 x^2 + \cdots + \lambda_v x^v$
  - $\Lambda(x)$ is the equation that shows the error positions
  - Error location polynomial coefficient is determined by the S_i
- Berlekamp-Massey (BM) algorithm
  - Find the shortest linear feedback shift register (LFSR) for a given binary output sequence
  - Representative KES algorithm
  - High HW complexity
  - iBM, RiBM, SiBM, TSiBM: derivatives of BM algorithm to reduce HW cost or latency

KES BM Algorithm

- Inversion-less Berlekamp-Massey algorithm
  - Remove inversion operator from normal BM algorithm to reduce HW cost
  - Released version is implemented in this algorithm

* iBM algorithm

(1) $$d = \sum_{j=0}^{t-1} \nu_j S_{2i+1-j}$$

(2) $$\nu^{(2i+2)}(z) = \delta^{(2i-2)}\nu^{(2i)}(z) + d^{(2i)}k^{(2i)}(z)\cdot z$$

(3) $$k^{(2i+2)}(z) = \begin{cases} z^2 k^{(2i)}(z) & \text{if } d^{(2i)} = 0 \text{ or if } \deg \nu^{(2i)}(z) > i \\ z\nu^{(2i)}(z) & \text{if } d^{(2i)} \neq 0 \text{ or if } \deg \nu^{(2i)}(z) \leq i \end{cases}$$

(4) $$\delta^{(2i)} = \begin{cases} \delta^{(2i-2)} & \text{if } d^{(2i)} = 0 \text{ or if } \deg \nu^{(2i)}(z) > i \\ d^{(2i)} & \text{if } d^{(2i)} \neq 0 \text{ or if } \deg \nu^{(2i)}(z) \leq i \end{cases}$$

Source file: key_equation_solver.v

KES Finite Field Multiplier (1/3)

- Finite field multiplier (FFM)
  - Fulfill multiply operation with given finite field ($GF(2^m)$)
    - $in\ GF(2^5), \alpha^{16} \otimes \alpha^{28} = \alpha^{44} = \alpha^{44-31} = \alpha^{13}$
  - Cf.) add/sub operations are achieved through FFA (Finite field adder) that is the same as XOR

- FFM is one of the key components that affect implementation cost of KES
  - Each module has high complexity
  - KES uses large amounts of FFMs

KES Finite Field Multiplier (2/3)

- FFM operation example

A0 * B0

5 bits FFM…

$A_0 = a_0 + a_1x + a_2x^2 + a_3x^3 + a_4x^4$

$B_0 = b_0 + b_1x + b_2x^2 + b_3x^3 + b_4x^4$

$A_0 * B_0$

$= a_0b_0 + (a_0b_1 + a_1b_0)x + (a_0b_2 + a_1b_1 + a_2b_0)x^2 + (a_0b_3 + a_1b_2 + a_2b_1 + a_3b_0)x^3 + \cdots$

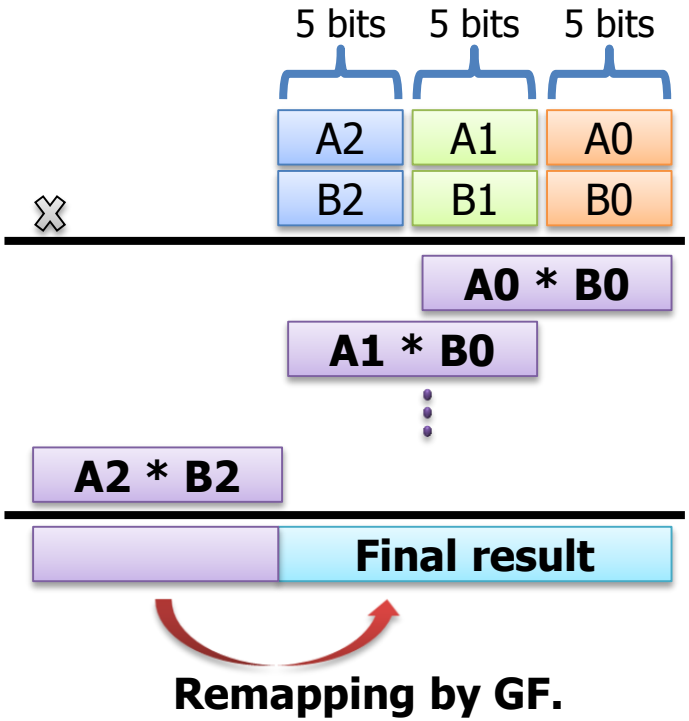
$+ (a_3b_4 + a_4b_3)x^7 + a_4b_4x^8$

```
assign o_r[8] = (i_a[4]&i_b[4]);
assign o_r[7] = (i_a[3]&i_b[4]) ^ (i_a[4]&i_b[3]);
assign o_r[6] = (i_a[2]&i_b[4]) ^ (i_a[3]&i_b[3]) ^ (i_a[4]&i_b[2]);
assign o_r[5] = (i_a[1]&i_b[4]) ^ (i_a[2]&i_b[3]) ^ (i_a[3]&i_b[2]) ^ (i_a[4]&i_b[1]);
assign o_r[4] = (i_a[0]&i_b[4]) ^ (i_a[1]&i_b[3]) ^ (i_a[2]&i_b[2]) ^ (i_a[3]&i_b[1]) ^ (i_a[4]&i_b[0]);
assign o_r[3] = (i_a[0]&i_b[3]) ^ (i_a[1]&i_b[2]) ^ (i_a[2]&i_b[1]) ^ (i_a[3]&i_b[0]);
assign o_r[2] = (i_a[0]&i_b[2]) ^ (i_a[1]&i_b[1]) ^ (i_a[2]&i_b[0]);
assign o_r[1] = (i_a[0]&i_b[1]) ^ (i_a[1]&i_b[0]);
assign o_r[0] = (i_a[0]&i_b[0]);
```


Source file: partial_FFM_gate.v

KES Finite Field Multiplier (3/3)

- Operation process of FFM



Remapping by GF.

- Divide and use small unit for reducing HW cost

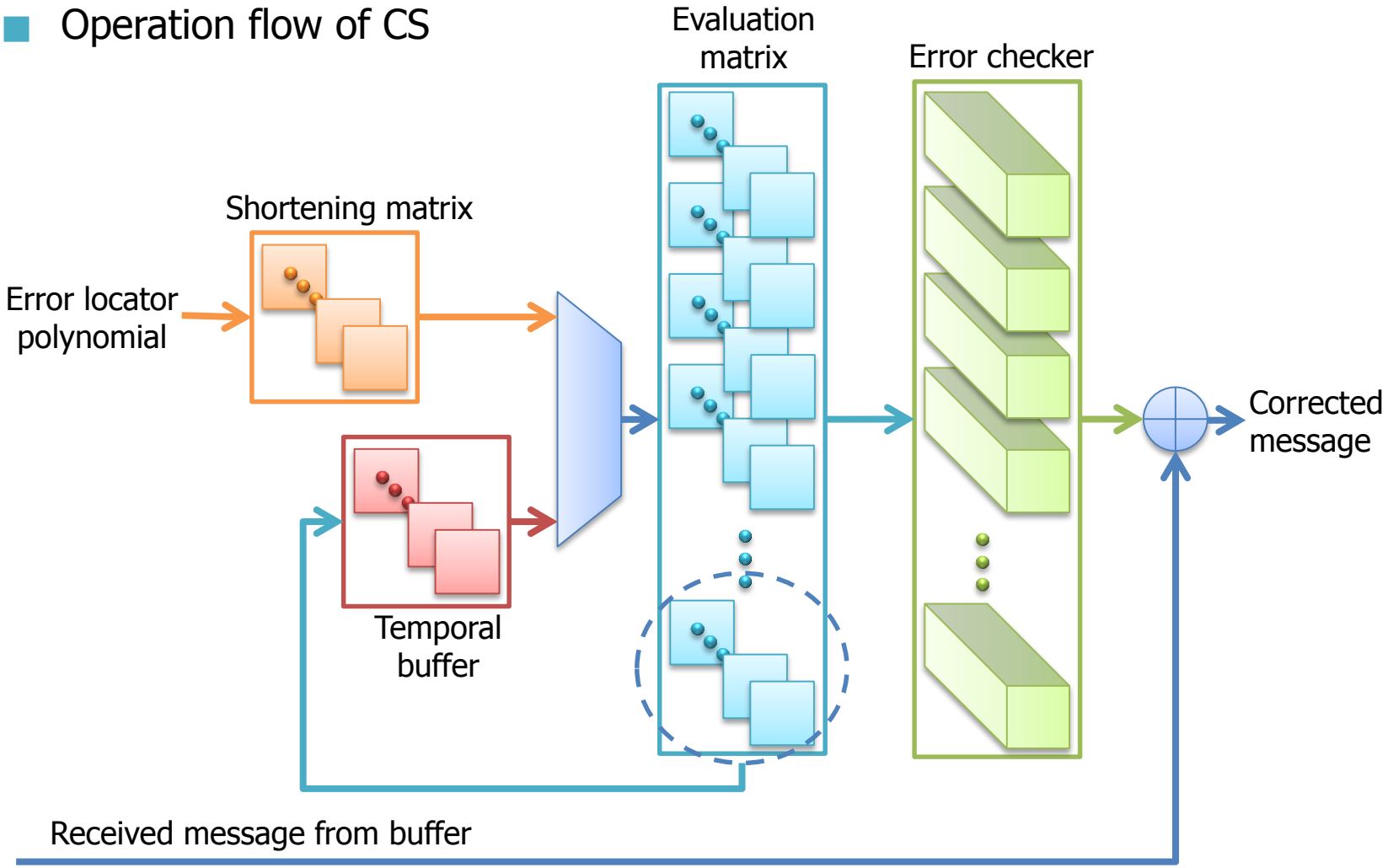
- Remapping by GF

Elements of $GF(2^{15})$		
Power Form	n-Tuple Form	Polynomial Form
0	00000_00000_00000	0
1	00000_00000_00001	1
α	00000_00000_00010	α
…	…	…
α^{13}	01000_00000_00000	α^{13}
α^{14}	10000_00000_00000	α^{14}
α^{15}	00000_00000_00011	$\alpha + 1$
α^{16}	00000_00000_00110	$\alpha^2 + \alpha$
α^{17}	00000_00000_01100	$\alpha^3 + \alpha^2$
α^{18}	00000_00000_11000	$\alpha^4 + \alpha^3$
α^{19}	00000_00001_10000	$\alpha^5 + \alpha^4$
…	…	…
α^{28}	11000_00000_00000	$\alpha^{14} + \alpha^{13}$

Source file: parallel_FFM_gate.v

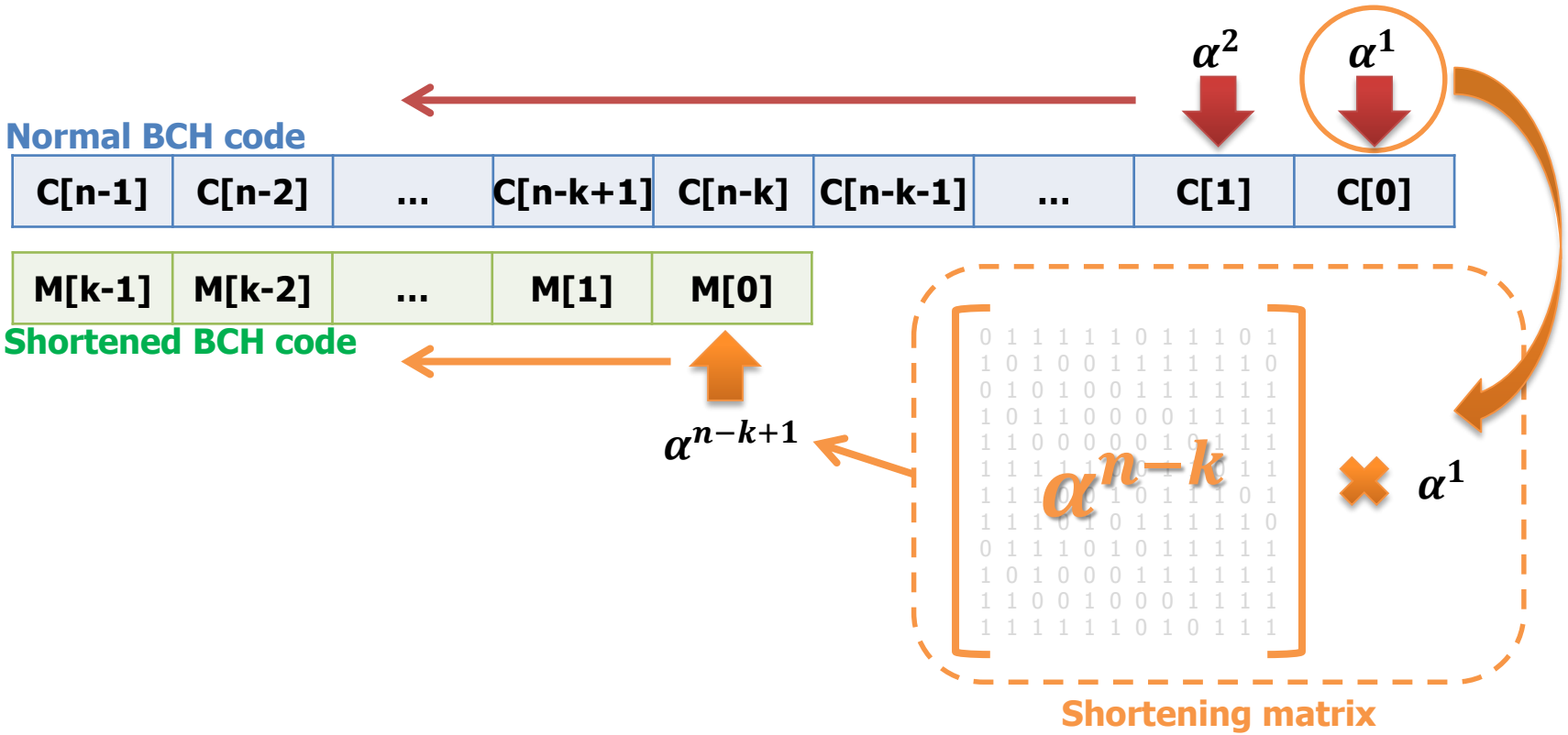
CS Block Diagram

- Operation flow of CS



CS Shortening Matrix

- Eliminate unnecessary cycles
  - Normal BCH code: substitute α^1 to α^n
  - Shortened BCH code: substitute α^{n-k+1} to α^n



Source file: cs_evaluation_matrices.v

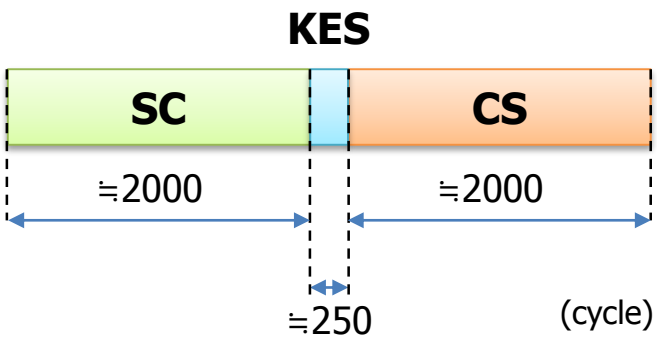
Future Extension

Counting Error bits

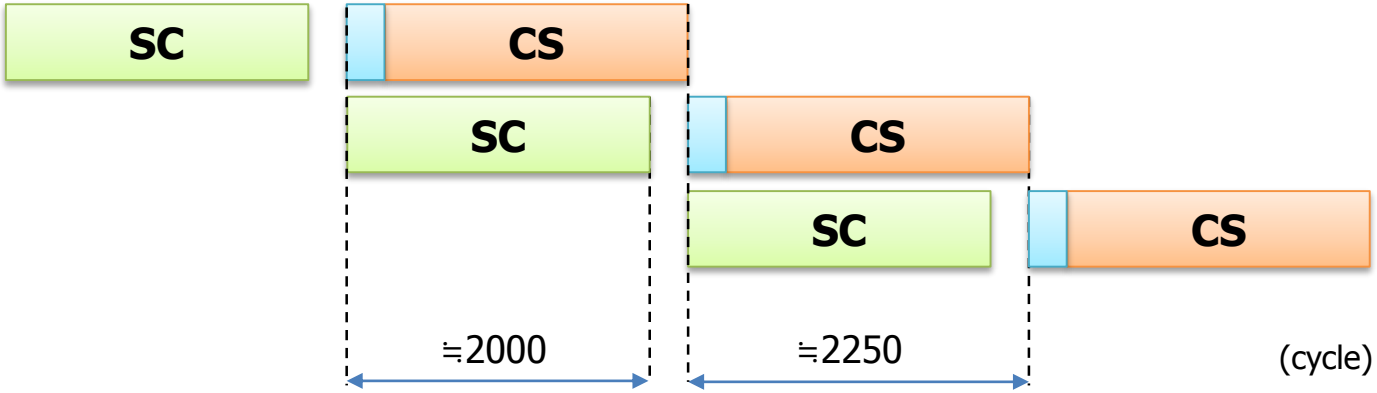
- Purpose
  - Estimating lifetime residual of NAND flash memory
  - Used for dynamic bad block management
- Requirement
  - Counting error bits in target chunk
  - (optional) Adding all the error counts in a page
- Method
  - Readout the MAXIMUM order of the error locator polynomial
- Affected module
  - KES or CS

Latency Reduction by a Pipeline

- Serial scheme
  - Total cycle ≒ 4250



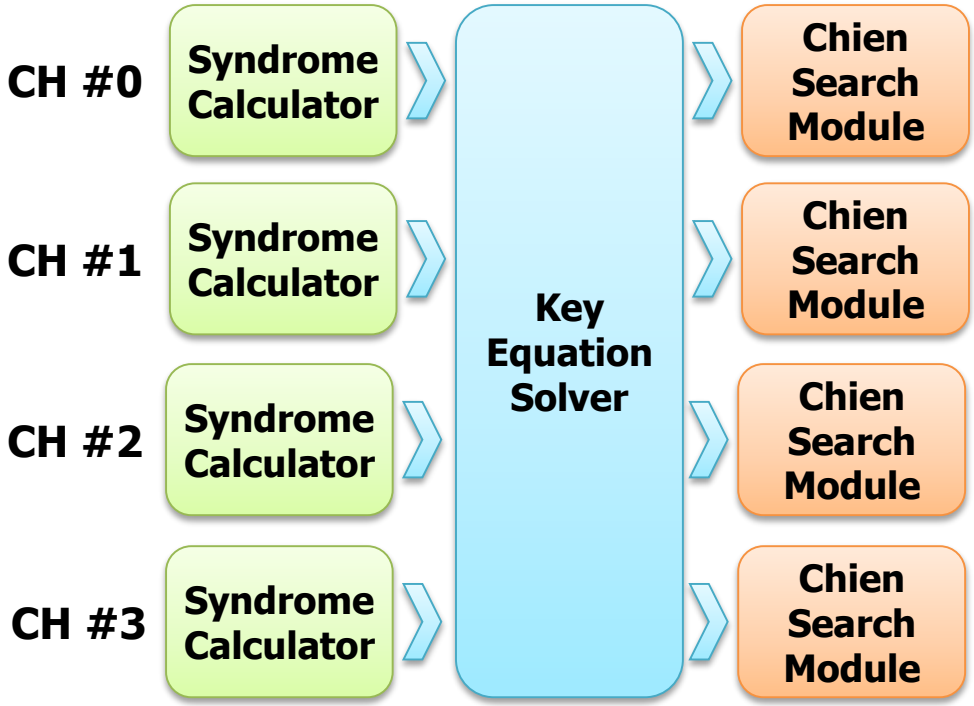
- Pipeline scheme
  - Total cycle ≒ 2250



* 2KB chunk, 32bit error correction, 8bit parallel level

Shared KES

- Key Equation Solver (KES)
  - Highest HW cost (≒ 70% of decoder cost)
  - Lowest latency (≒ 6% of decoder cycles)



* 2KB chunk, 32bit error correction, 8bit parallel level

KES is very good target for sharing

References

- Books
  - R. Micheloni, A. Marelli and R. Ravasio, *Error Correction Codes for Non-Volatile Memories,* Springer Netherlands, 2008.
  - S. Wicker, *Error Control Coding for Digital Communication and Storage*, Prentice-Hall, Englewood Cliffs, NJ, 1995.
  - Wallace, H. (2001) Error detection and correction using the BCH code. Retrieved January 3, 2014 from http://www.aqdi.com/bch.pdf
  - More…
- Papers
  - D. V. Sarwate and N. R. Shanbhag, "High-speed architectures for Reed-Solomon decoders," IEEE Trans, on Very Large Scale Integration, vol. 9. no. 5, pp. 641-655, Oct. 2001. - iBM, RiBM
  - More…

Authors

Name	E-mail	Contribution
Ilyong Jung	iyjung@enc.hanyang.ac.kr	2014-04 ~ Now
Jinwoo Jeong	jwjeong@enc.hanyang.ac.kr	2015-03 ~ Now

Cosmos OpenSSD Platform Tutorial

Host Interface

ENC Lab. @ Hanyang University

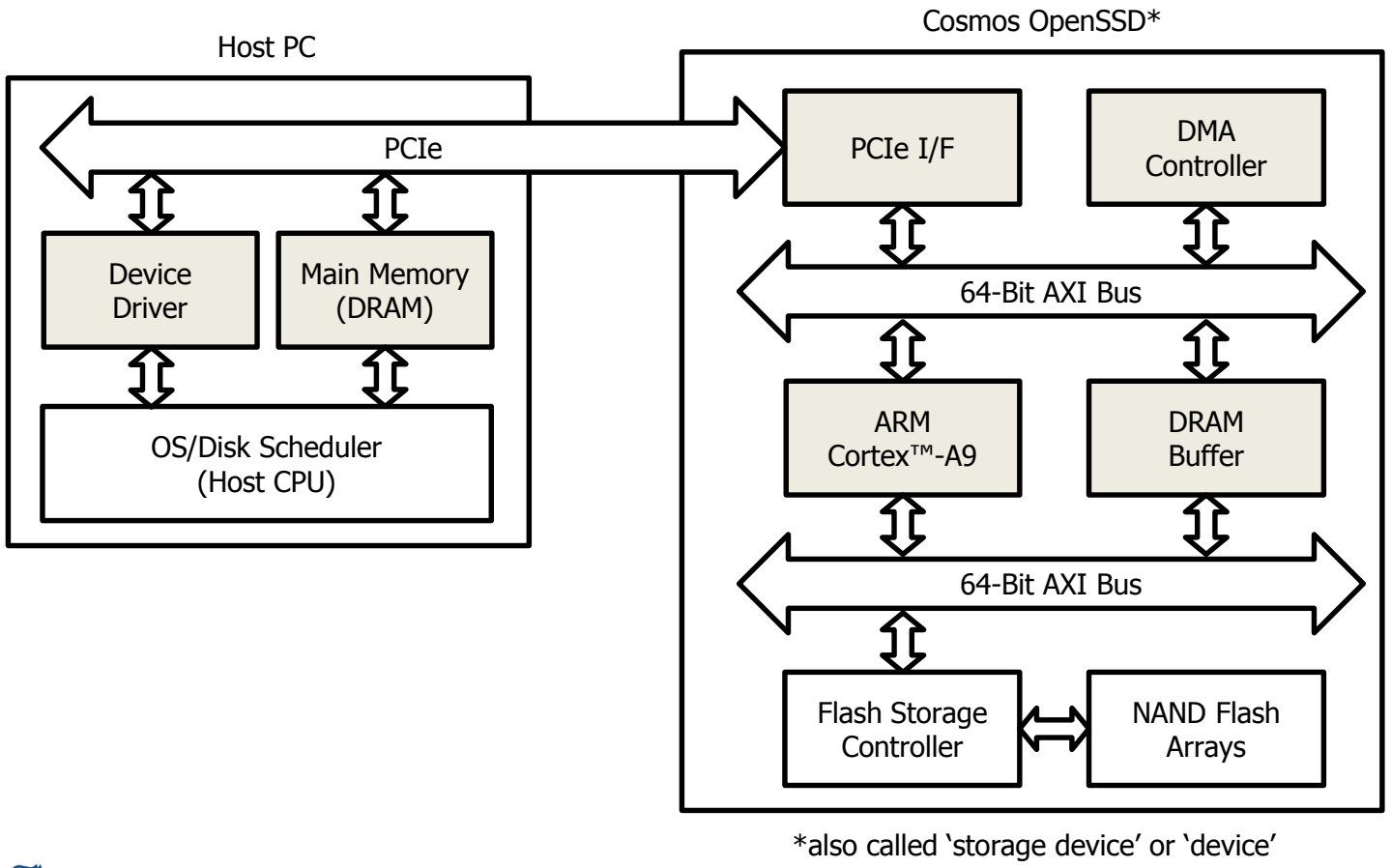
HANYANG UNIVERSITY
ENC Lab
Embedded Network Computing Lab.

Document Revision History

- v1.0.0
  - Initial release
  - Contents are based on the device_drvier_v1.1.0-Linux

Section Scope

- Includes driver, firmware, PCIe block and DMA controller



*also called 'storage device' or 'device'

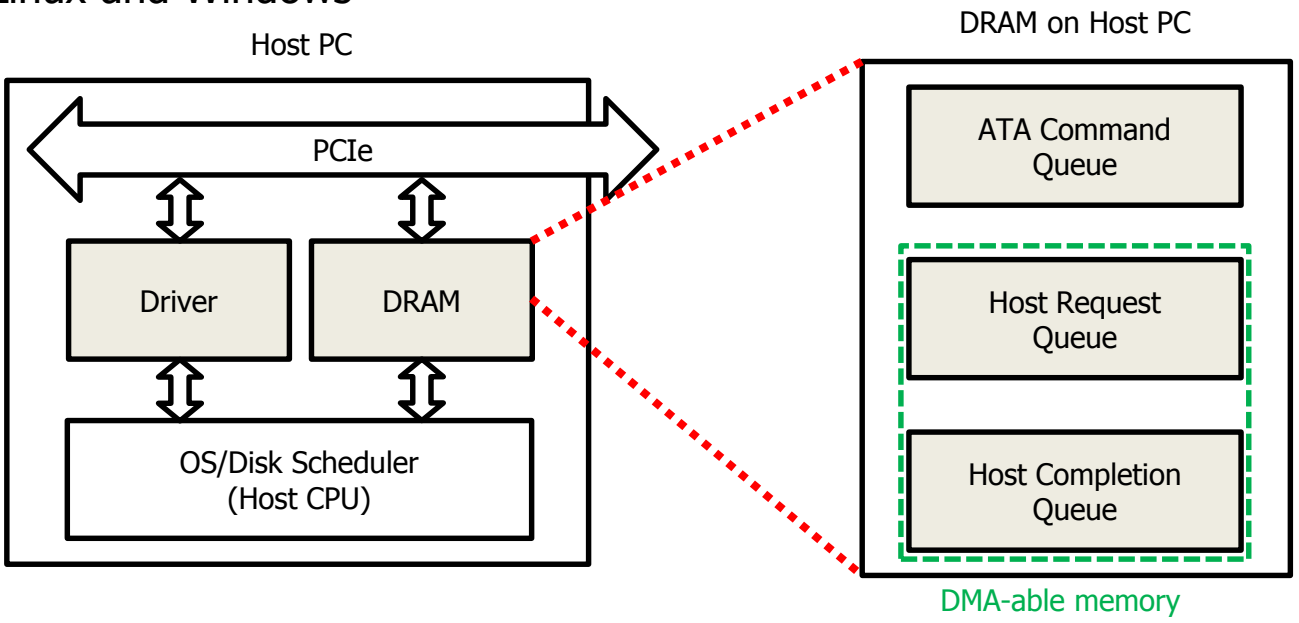
INDEX

HANYANG UNIVERSITY
ENC Lab
Embedded Network Computing Lab.

Host Interface Overview

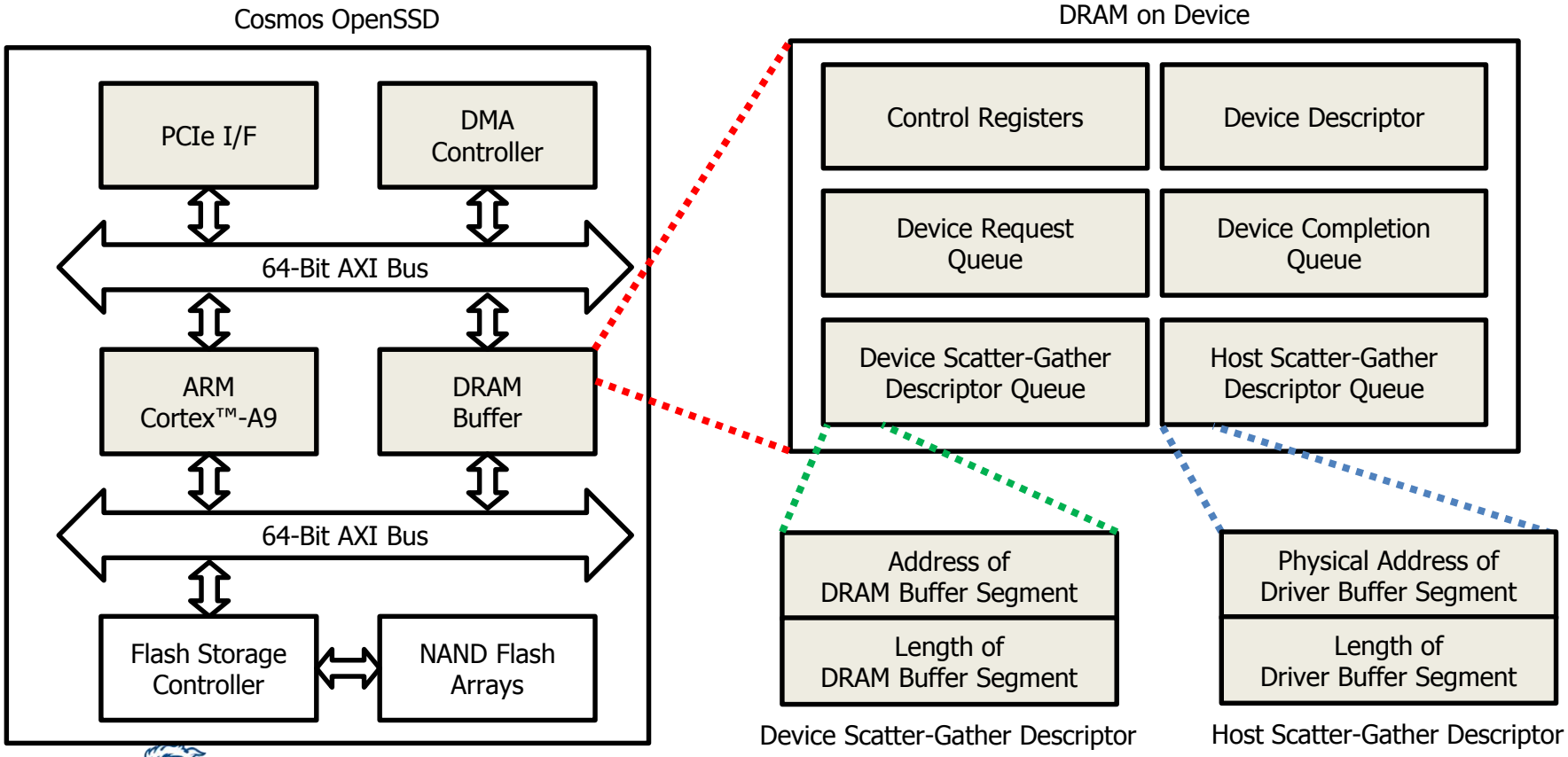
Host PC Driver

- Role
  - Makes the device act as disk drive on the host PC
- Function
  - Allocates special queues dedicated to DMA in host PC's DRAM
    - Queue sizes are controlled by the driver
- Support
  - Supports a subset of the ATA command set
  - Supports Linux and Windows



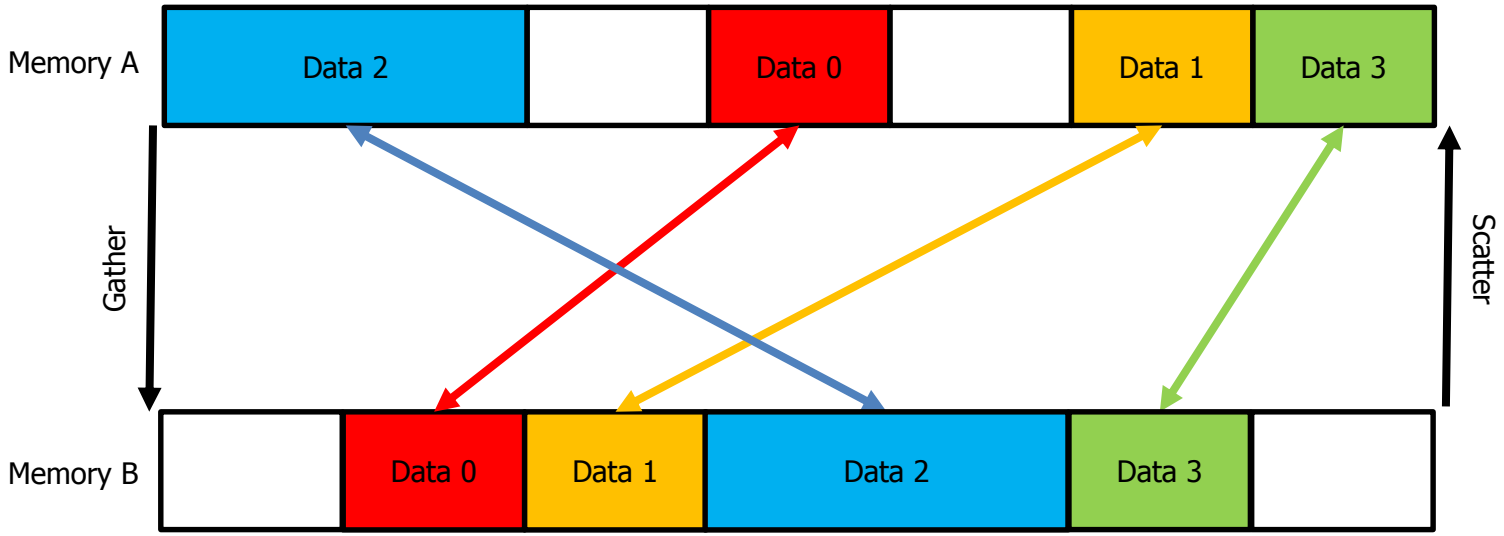
PCIe Firmware

- Runs on the core CPU on the device
- Supports a subset of the ATA command set
- Controls a DMA controller



Scatter-Gather I/O Mechanism

- Also known as vectored I/O



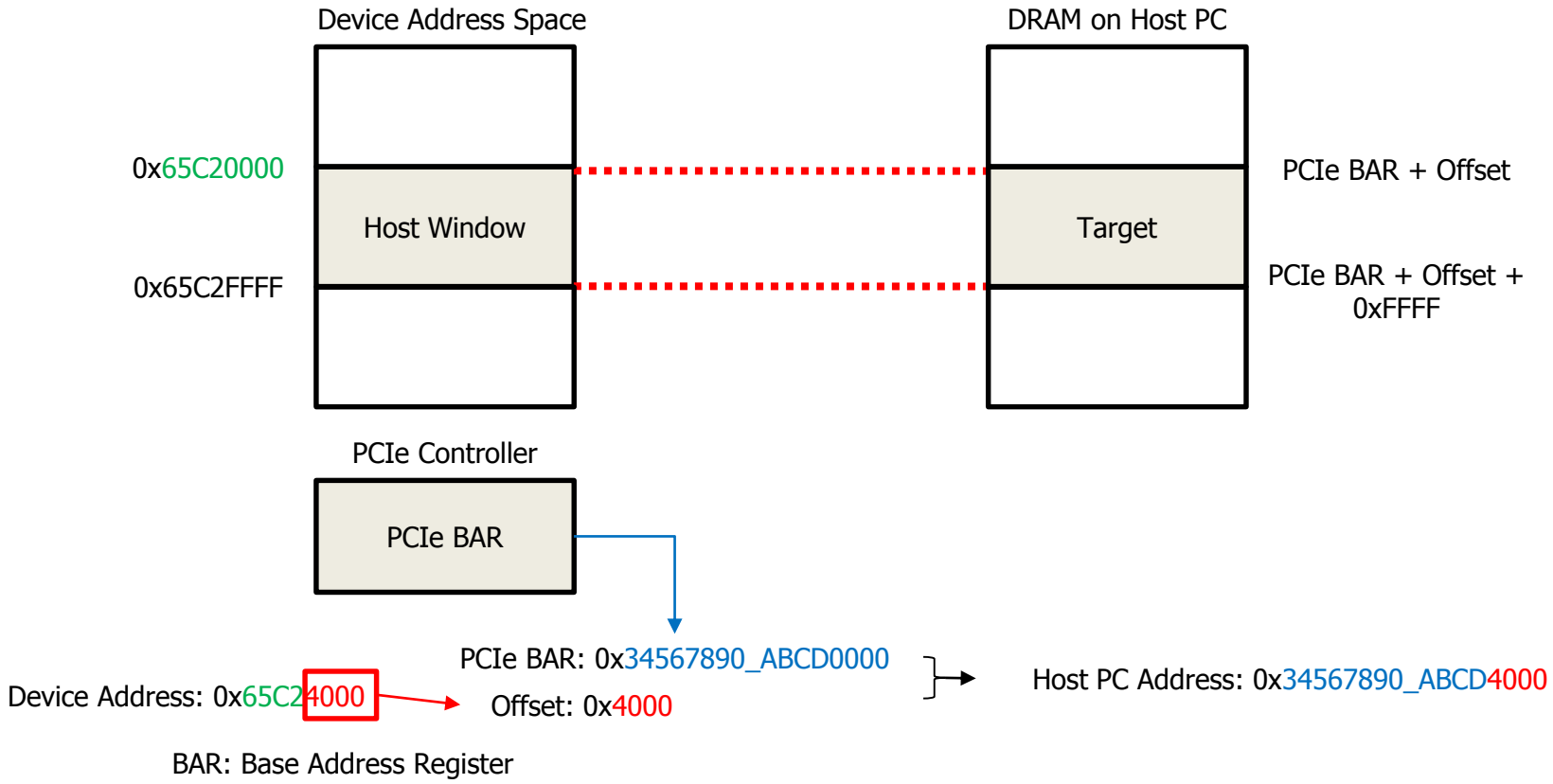
Address Map of the Device

Address Range	Name	Description
0x00000000~0x3FFFFFFF	DRAM	Main memory of the device, includes control registers, queues, and data
0x40200000~0x4020FFFF	DMA Controller	DMA controller register file
0x65C00000~0x65C0FFFF	PCIe Interface	PCIe interface register file
0x65C20000~0x65C2FFFF*	Host Window	Host window mapped to DRAM in the host PC through PCIe interface
0x66E00000~0x66E0FFFF	Flash Storage Controller Channel 3	Channel 3 flash storage controller register file
0x66E20000~0x66E2FFFF	Flash Storage Controller Channel 2	Channel 2 flash storage controller register file
0x66E40000~0x66E4FFFF	Flash Storage Controller Channel 1	Channel 1 flash storage controller register file
0x66E60000~0x66E6FFFF	Flash Storage Controller Channel 0	Channel 0 flash storage controller register file
0x7A000000~0x7A00FFFF	PCIe Interface initialization checker	PCIe interface initialization checker register file

*Configurable by C_AXIBAR_0 & C_AXIBAR_HIGHADDR_0 in PCIe IP config

Host Window Mechanism

- Address mapping between host DRAM and host window
  - Direction: device to host only



DRAM Address Map on The Device

Address Range	Size	Name	Description
0x01100000- 0x011FFFFF**	1MB**	DRQ	Device request queue
0x01200000- 0x012FFFFF**	1MB**	DCQ	Device completion queue
0x01300000- 0x013FFFFF**	1MB**	HSGQ	Host scatter-gather descriptor queue
0x01400000- 0x014FFFFF**	1MB**	DSGQ	Device scatter-gather descriptor queue
0x02000000- 0x020001FF**	512Byte	DD	Device descriptor for Windows
0x08000000- 0x080FFFFF*	1MB*	DCR	Device control register file
0x10000000- 0x3FFFFFFF**	768MB**	DRB	Dram buffer

*Configurable by C_PCIEBAR2AXIBAR_0 & C_PCIEBAR_LEN_0 in PCIe IP config

**Configurable at line 72 ~ 84 in host_controller.h (firmware)

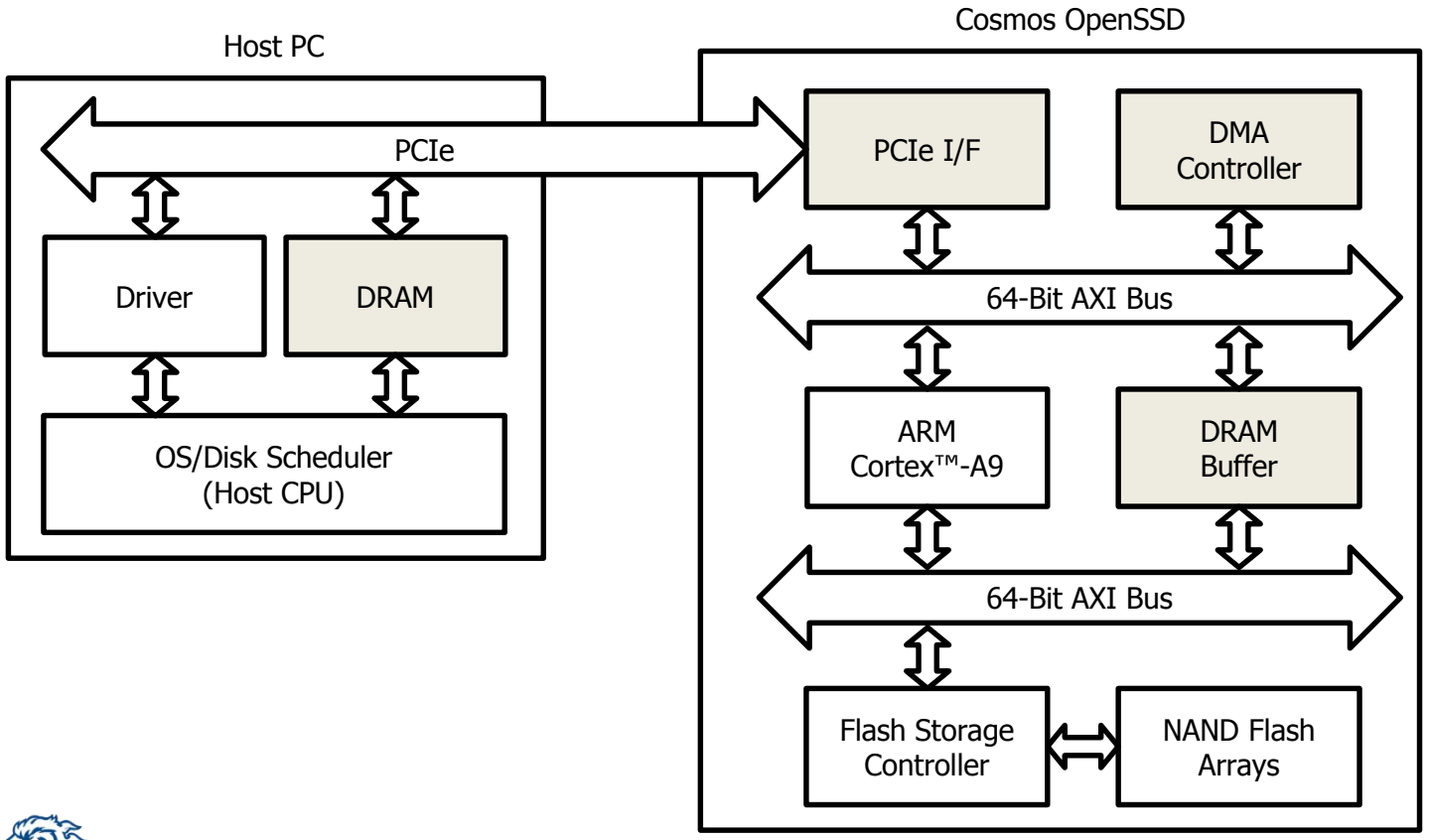
Device Control Registers in Device DRAM

Address	Name	Description	Driver	Firmware
0x08000000	RS	Request start (Request queue tail)	W	R
0x08000004	HRQU	Upper part of base address of request queue in DRAM on host PC	W	R
0x08000008	HRQL	Lower part of base address of request queue in DRAM on host PC	W	R
0x0800000C	HCQU	Upper part of base address of completion queue in DRAM on host PC	W	R
0x08000010	HCQL	Lower part of base address of completion queue in DRAM on host PC	W	R
0x08000014	SDC	Shutdown command	W	R
0x08000018	SCI	Sector count information	R	W

All control registers are configurable at line 106 ~ 115 in host_controller.h (firmware)
& line 135 ~ 143 in enc_pcie.h (driver)

PCIe Interface and DMA Controller

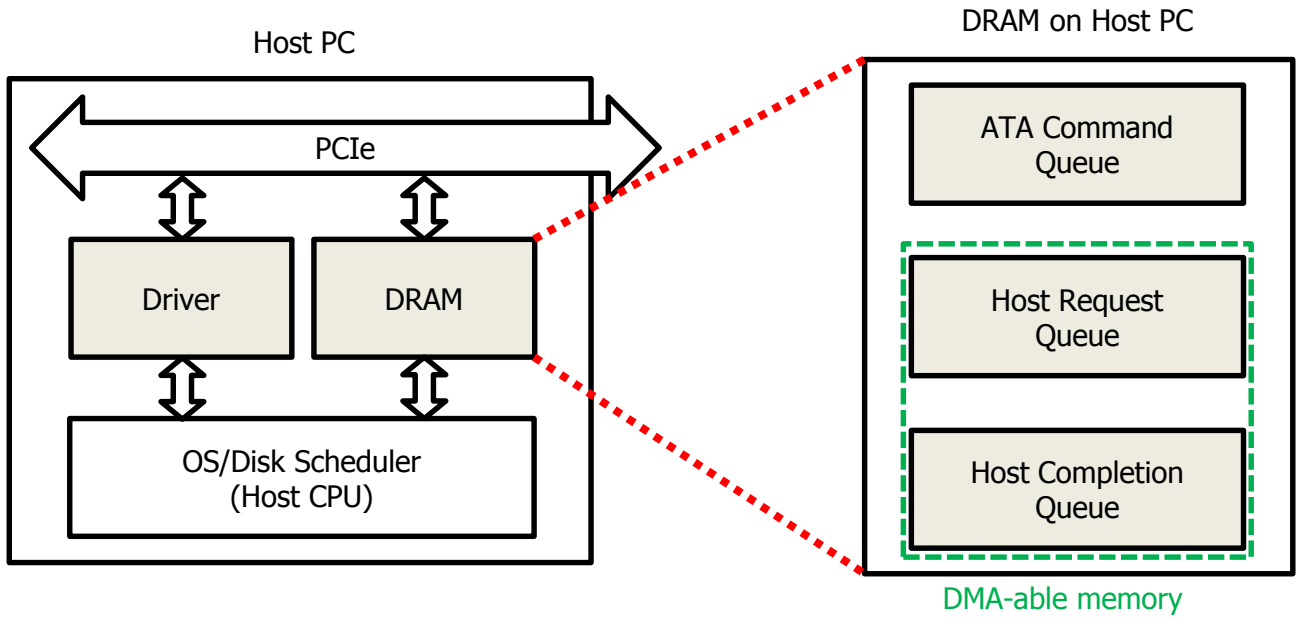
- Using Xilinx AXI mapped PCIe IP and Central DMA IP
- Data transfer between DRAM on host PC and DRAM on device through PCIe interface



Operation Flow

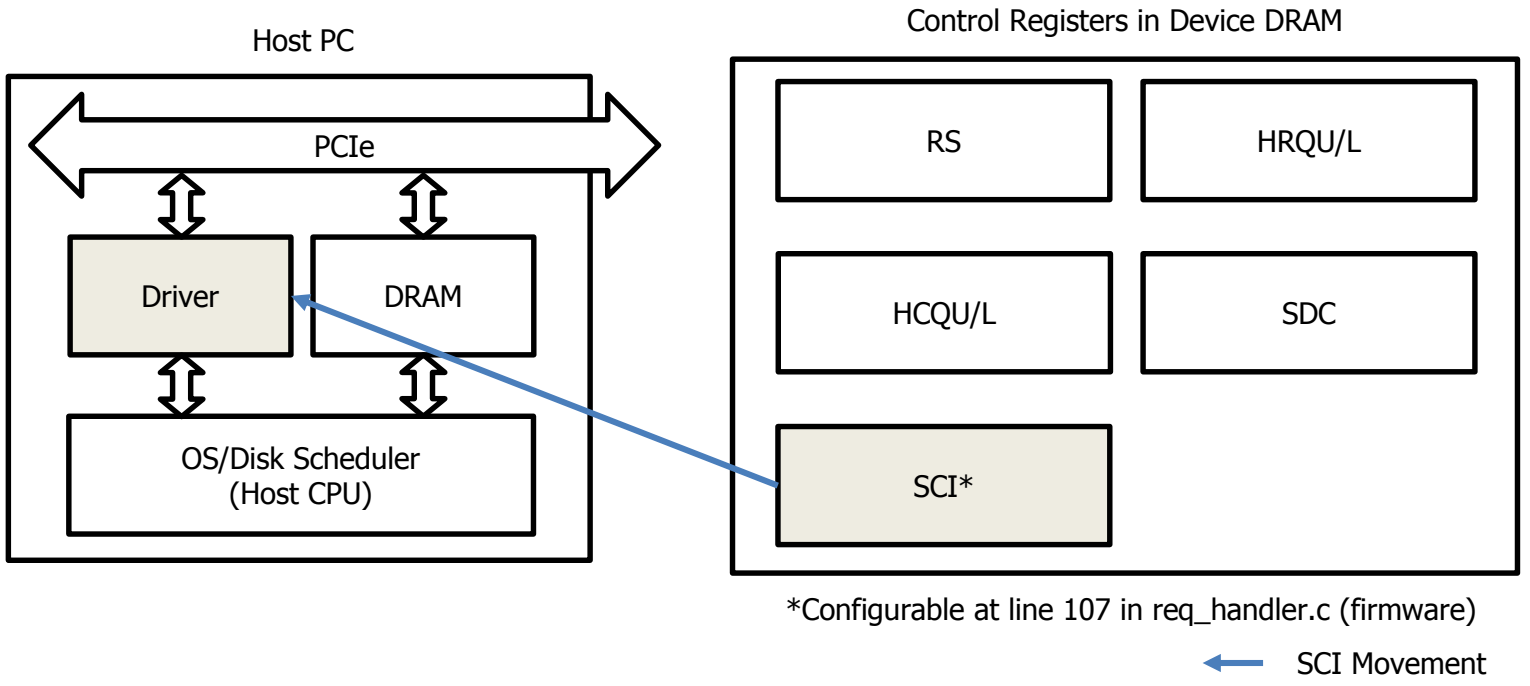
Driver Initialization–Memory Allocation

- Driver allocates memory for
  - ATA command queue
  - Host request queue
  - Host completion queue



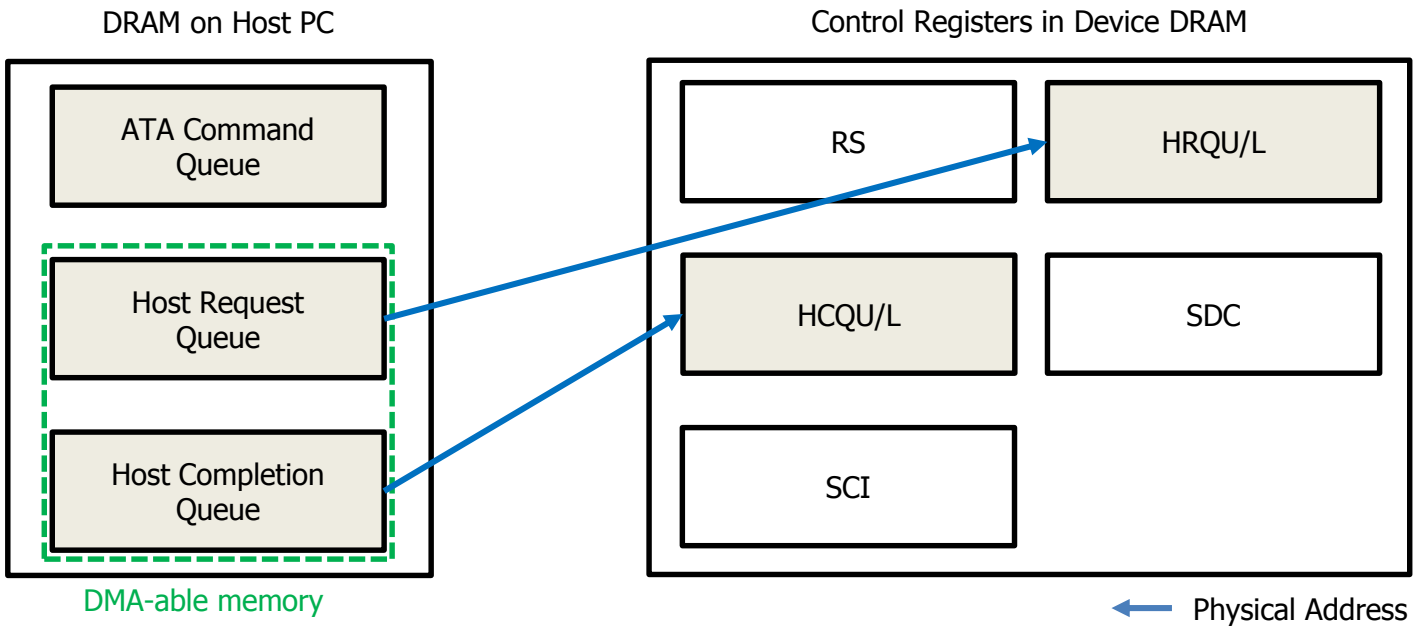
Driver Initialization–Storage Capacity Setting

- Driver reads Sector Count Information (SCI) register from the device
  - Initial SCI value is written by PCIe firmware on boot time
- Driver sets storage capacity of OS kernel using the SCI value read



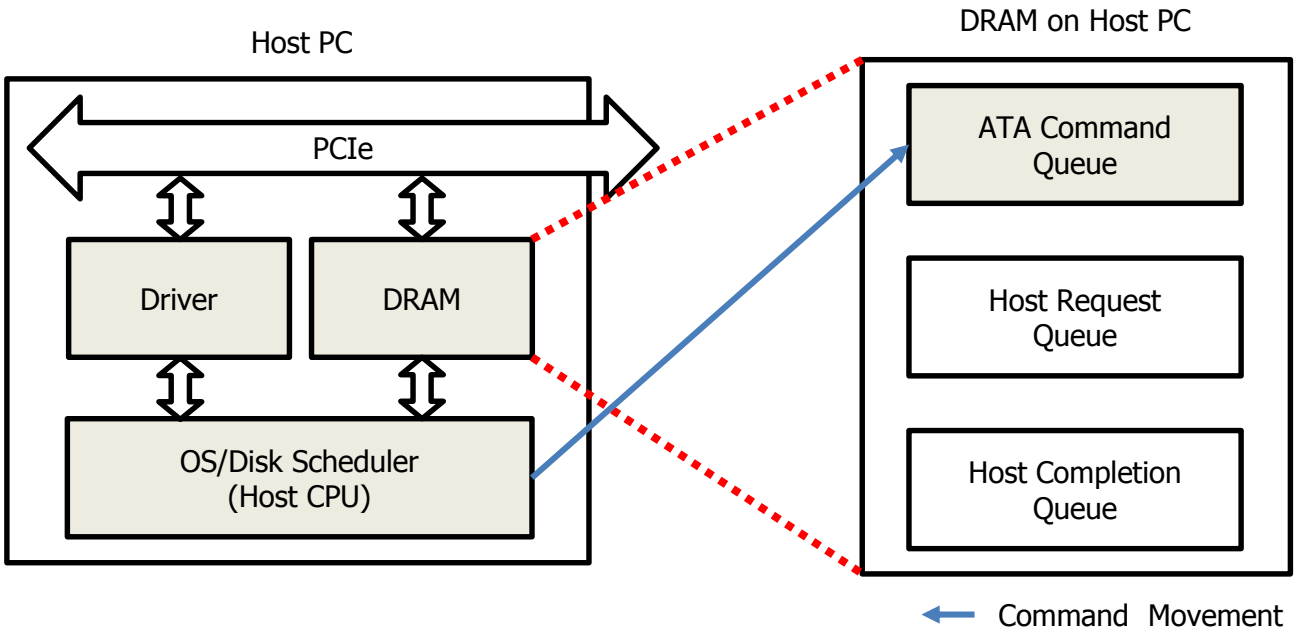
Device Initialization

- Device initialization is triggered by host driver initialization
- Driver writes the physical addresses of request queue and completion queue on host PC to control registers of the device



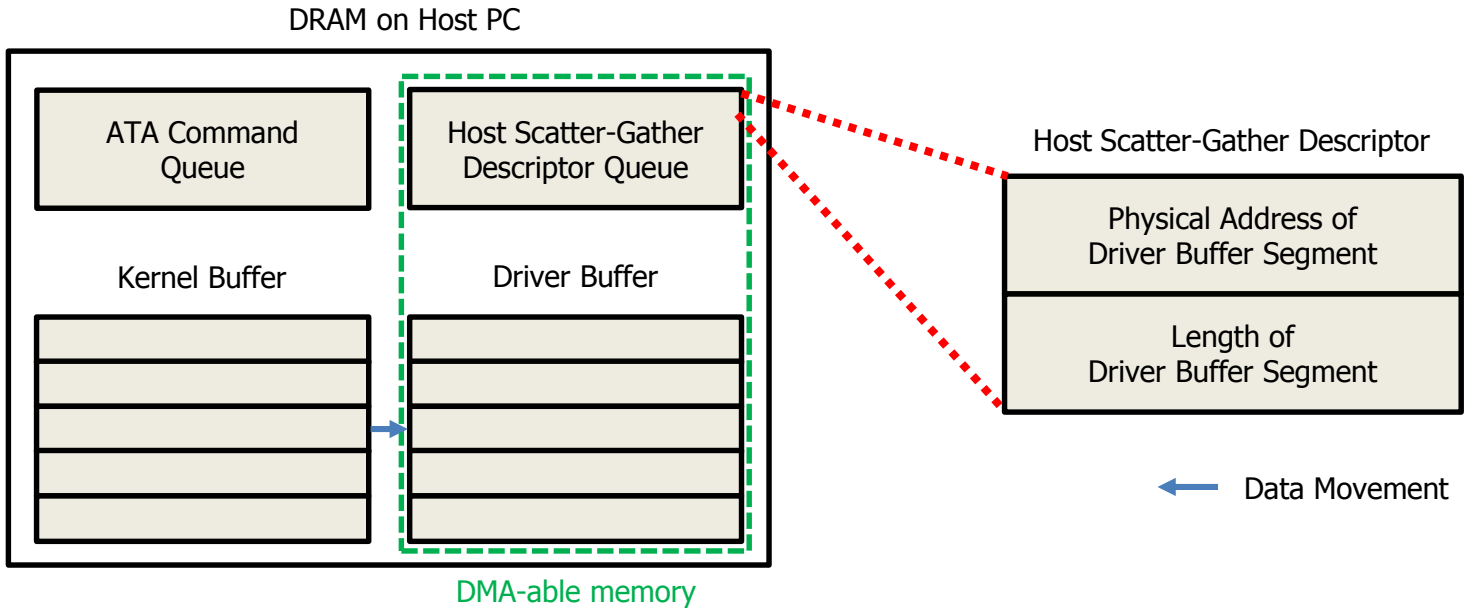
ATA Command Queuing

- If the ATA command queue of driver is not full, the scheduler enqueues ATA commands to the command queue
- Otherwise, driver returns busy state to the scheduler



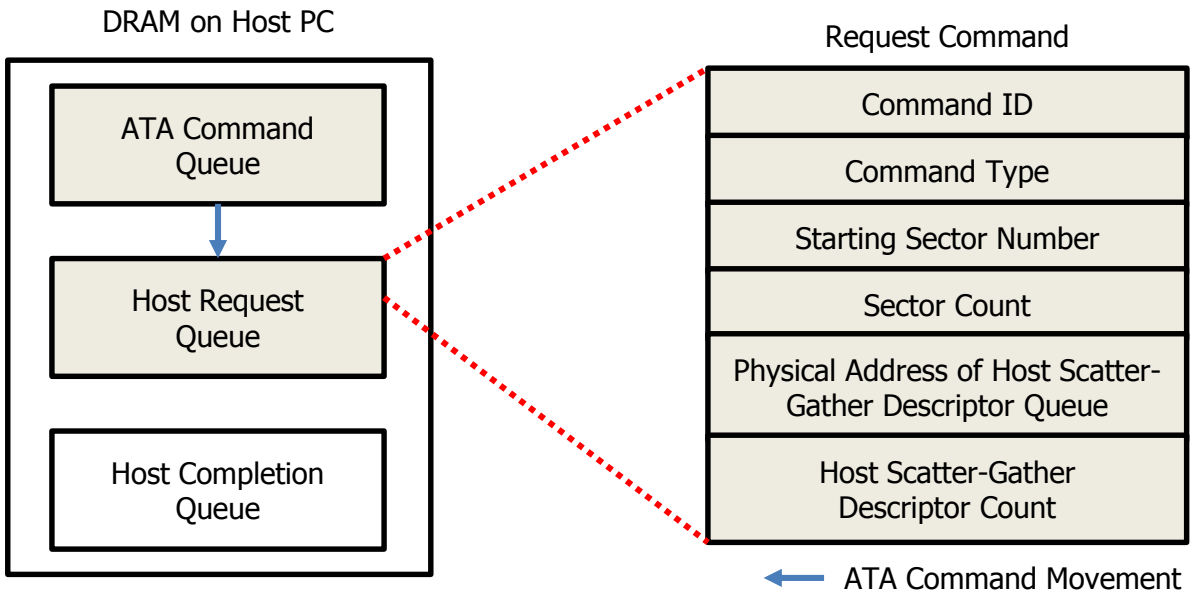
Copy Data in Host System

- If ATA command requires data movement,
  - Driver allocates driver buffer (DMA-able) using the sector count
  - Driver prepares host scatter-gather descriptors and enqueues them to the queue
  - If it is a write command, driver copies data from kernel buffer to driver buffer



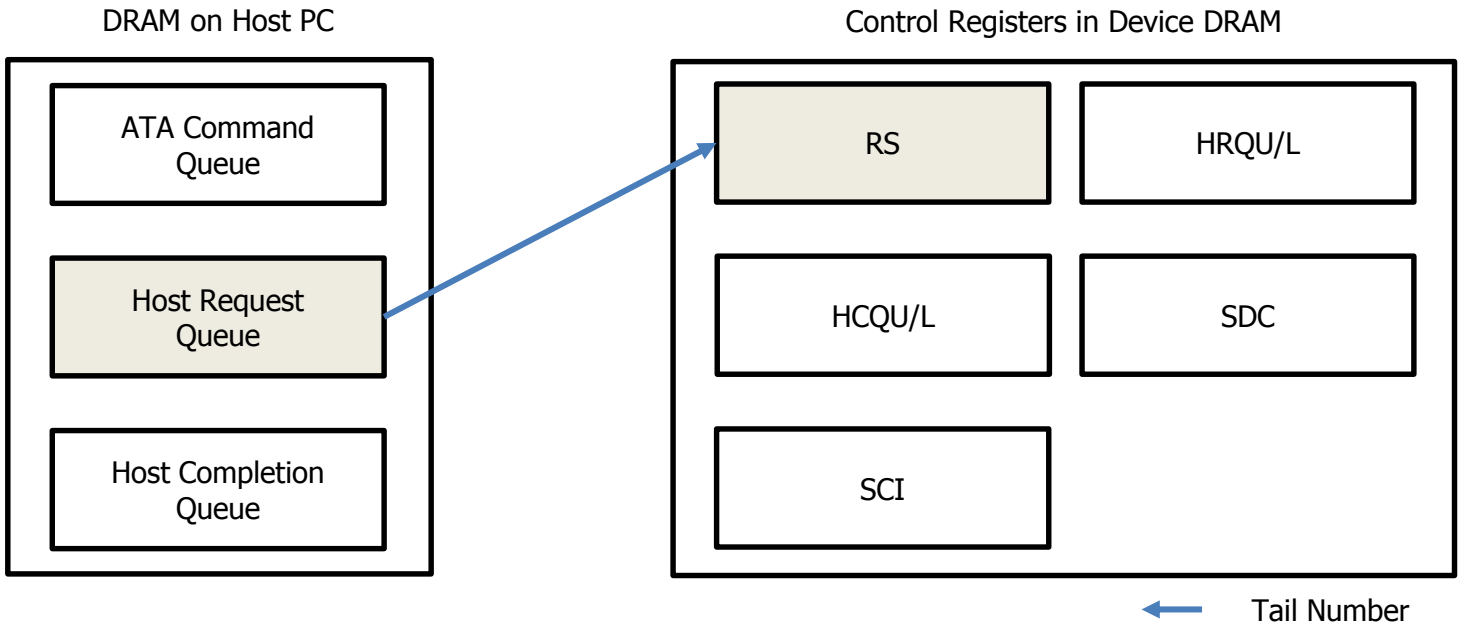
Setup Request Command

- Driver makes a request command by fetching an ATA command from the queue
  - Driver enqueues the request command to the host request queue
- The request command includes command ID, command type, starting sector number, sector count and host scatter region descriptor



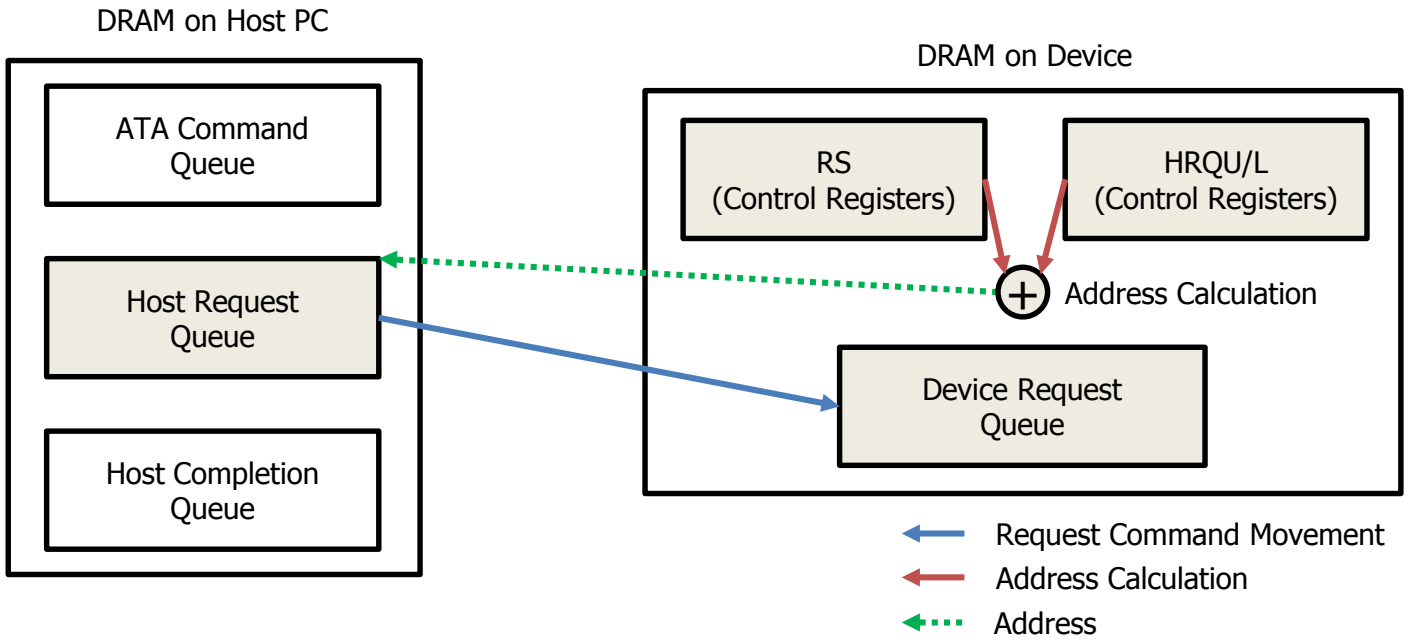
Command Issue

- After the setup of a request command is completed, driver updates request queue tail number to the request start register of PCIe firmware



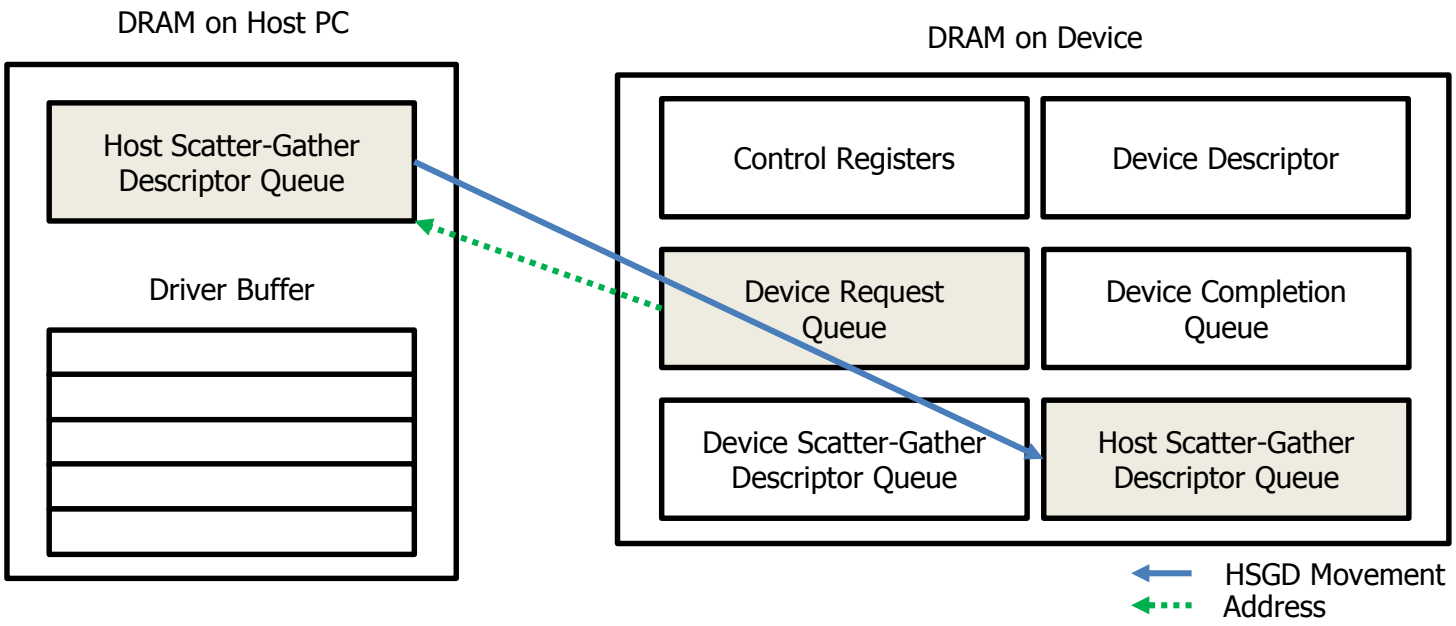
Request Command DMA

- PCIe firmware transfers a request command from host request queue to device request queue using HRQU/L



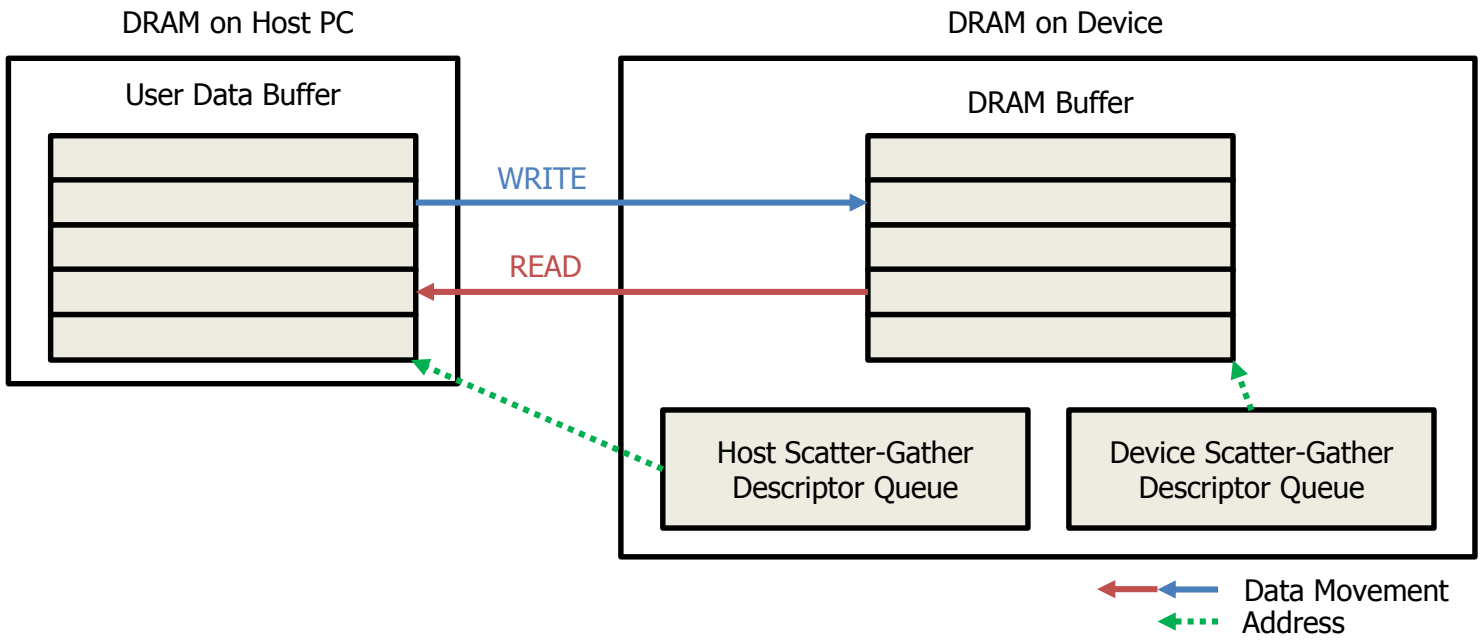
Scatter-Gather Descriptor DMA

- PCIe firmware transfers a scatter-gather descriptor from host to device
  - Using physical address of the host scatter-gather descriptor queue in request command



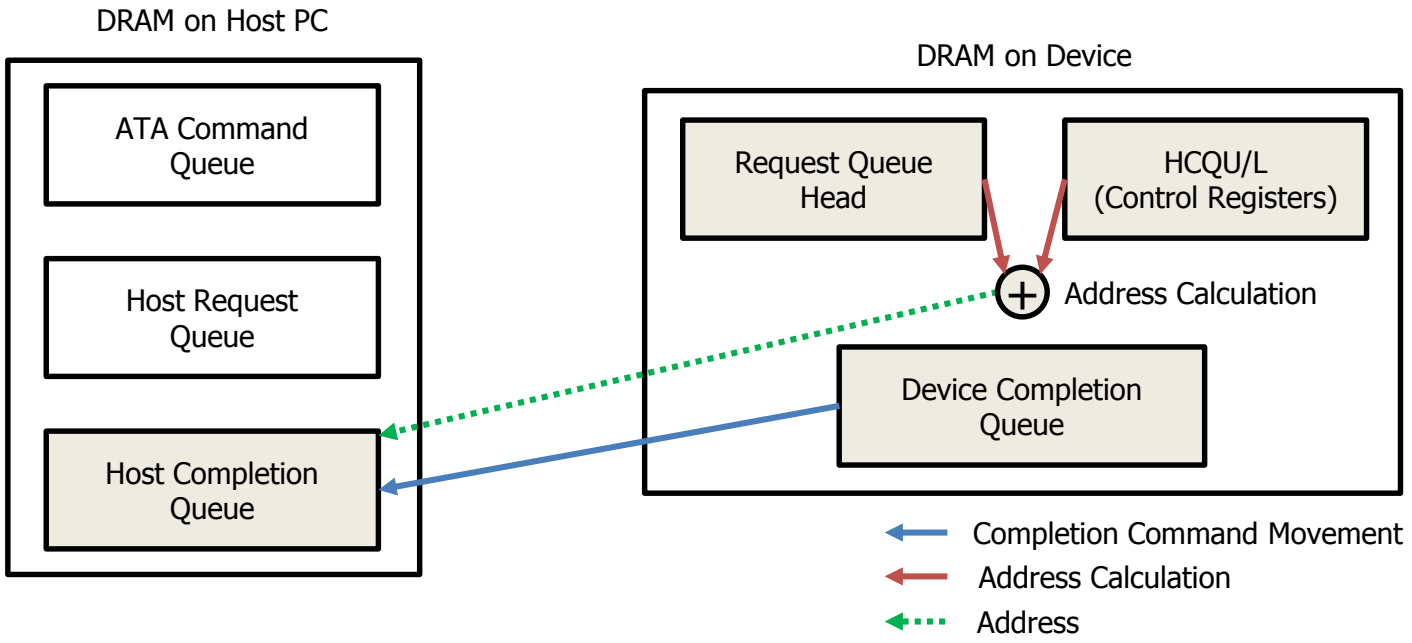
Data DMA

- When I/O command is
  - READ, PCIe firmware transfers data from device to host
  - WRITE, PCIe firmware transfers data from host to device
  - Using host scatter-gather descriptor and device scatter-gather descriptor
    - FTL manages device scatter-gather descriptor



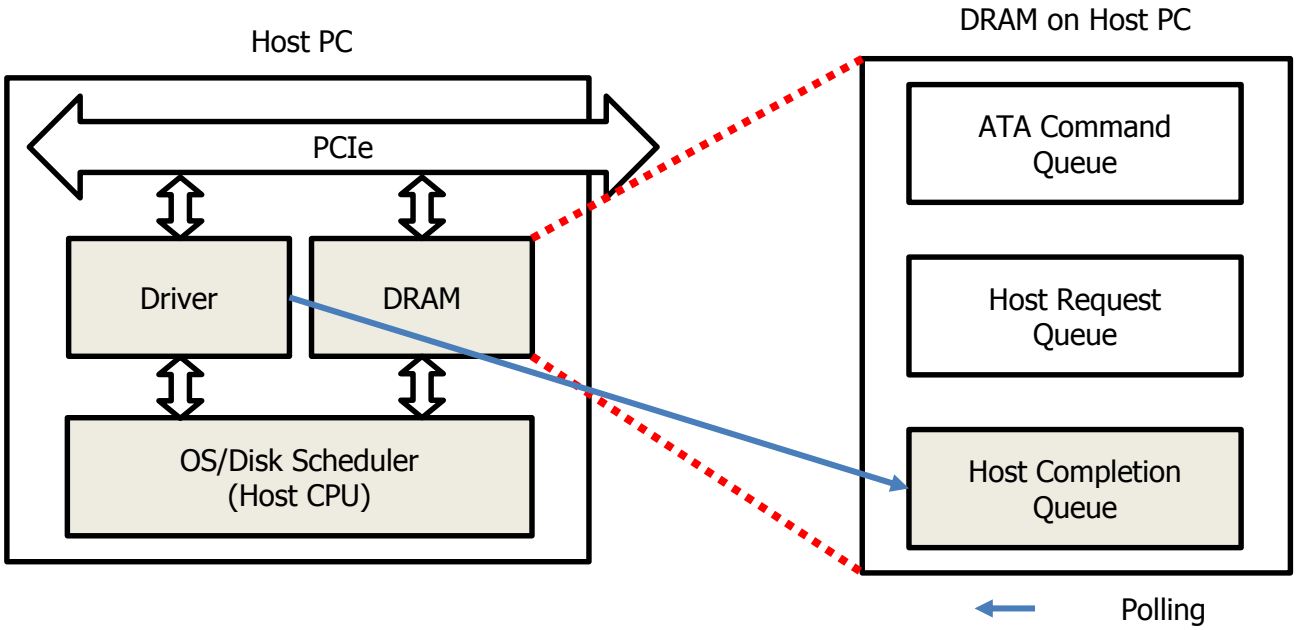
Completion Command DMA

- After a request command is completes, PCIe firmware transfers completion command from device completion queue to host completion queue
- Completion command includes command status and error status



ATA Command Completion

- Driver polls host completion queue to check whether ATA command is finished
- When driver senses a new completion command, driver sends command status and error status to scheduler



API to FTL on Device

- host_controller.h

#define DISK_BUFFER_BASE_ADDR 0x1000_0000

- Start address of DRAM buffer (part of device DRAM)
- The DRAM buffer ends at DRAM0_HIGH_ADDR (0x3FFF_FFFF)

struct REQUEST_IO

- u32 cmd: command type of the request from the host PC
- u32 currentSect: base sector number of the request
- u32 reqSect: size of the request in sector unit

void DmaDeviceToHost(P_HOST_CMD hostCmd, u32 deviceAddr, u32 reqSize, u32 scatterLength)

- Transfer data on DRAM buffer to host
- reqSize sectors data on deviceAddr is read by the host

void DmaHostToDevice(P_HOST_CMD hostCmd, u32 deviceAddr, u32 reqSize, u32 scatterLength)

- Transfer data from host to buffer
- reqSize sectors data from the host is written on deviceAddr

Authors

Name	E-mail	Contribution
Sangjin Lee	sjlee@enc.hanyang.ac.kr	2014-04 ~ Now

Cosmos OpenSSD Platform Tutorial

Greedy Flash Translation Layer

ENC Lab. @ Hanyang University

HANYANG UNIVERSITY
ENC Lab
Embedded Network Computing Lab.

Document Revision History

- v1.0.0
  - Initial release
  - Contents are based on greedy_FTL_v1.1.0

Section Scope

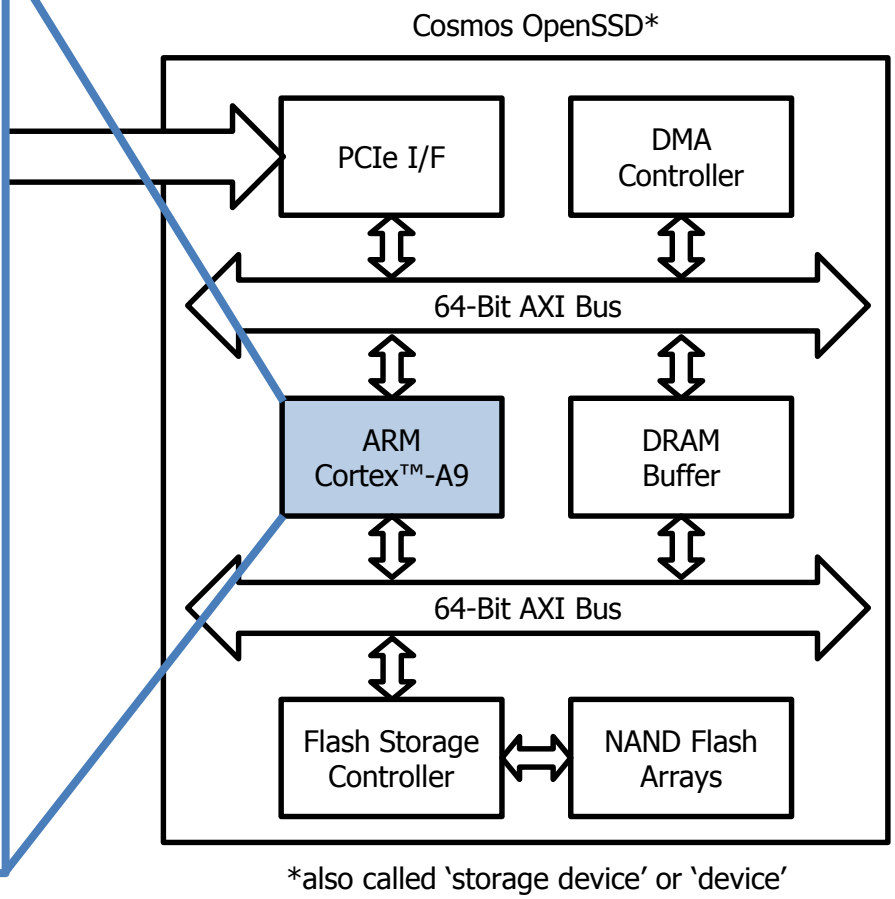
- FTL runs on ARM Cortex™-A9 core

```
InitFlash();
InitFTL();
InitStorageSize();

while(true)
{
        GetRequest();

        if(request.command == read)
        {
                PageMapRead();
                DmaDeviceToHost();
        }
        else if(request.command == write)
        {
                PageMapPreread();
                DmaHostToDevice();

                if(no free page)
                        GreedyGc();
                PageMapWrite();
        }
}
```



*also called 'storage device' or 'device'

INDEX

FTL Overview

FTL Feature

Pure page-level mapping (8 KB page)

- Static mapping
- Channel/way interleaving

Greedy garbage collection

- On-demand garbage collection
- Greedy selection of GC victims

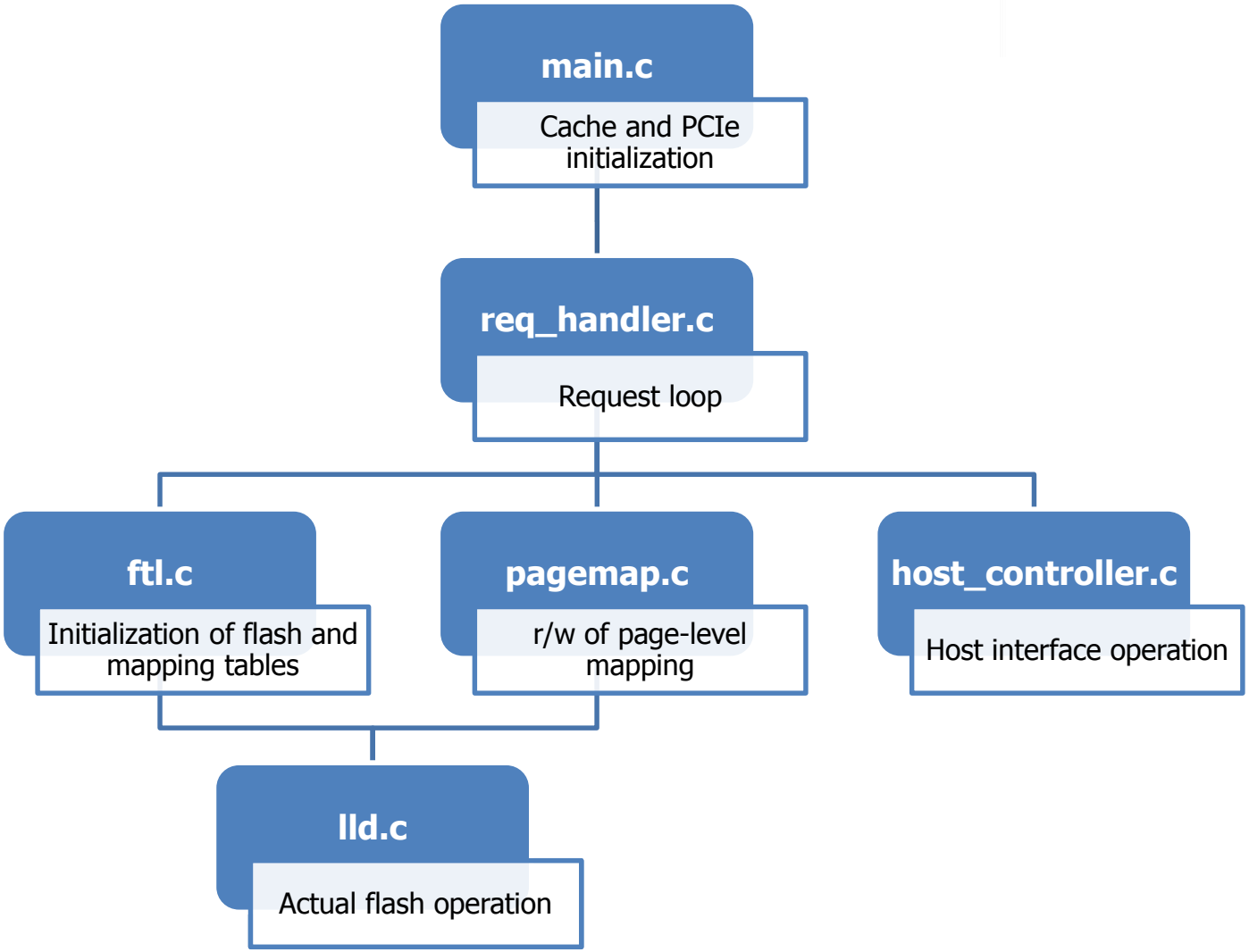
Static bad block management

- Factory manufacturing bad block management
- Initial bad block checking at booting

Single plane operation

- Reset, mode change, status read
- Erase, read, program

Source Hierarchy



Low Level Driver APIs to FTL

Functions in lld.c

int SsdReset(u32 chNo, u32 wayNo)
▶ Reset flash devices ▶ Reset is required as first command after power-on
int SsdModeChange(u32 chNo, u32 wayNo)
▶ Change to synchronous flash ▶ Change the mode after resetting the flash
int SsdErase(u32 chNo, u32 wayNo, u32 blockNo)
▶ Erase a block of flash
int SsdRead(u32 chNo, u32 wayNo, u32 rowAddr, u32 dstAddr)
▶ Read a page of flash ▶ Load page-data of rowAddr on dstAddr
int SsdProgram(u32 chNo, u32 wayNo, u32 rowAddr, u32 srcAddr)
▶ Program a page of flash ▶ Program rowAddr with page-data on srcAddr
void WaitWayFree(u32 ch, u32 way)
▶ Wait until the issued command is completed ▶ This function should be preceded to erase, read and program operations

FTL Operation Flow

Overall Sequence

```
InitFlash();
InitFTL();
InitStorageSize();
```

Flash storage and mapping table initialization

```
while(true)
{
        GetRequest();
```

Request is ready?

```
        if(request.command == read)
        {
                PageMapRead();
                DmaDeviceToHost();
        }
        else if(request.command == write)
        {
                PageMapPreread();
                DmaHostToDevice();

                if(no free page)
                        GreedyGc();
                PageMapWrite();
        }
}
```

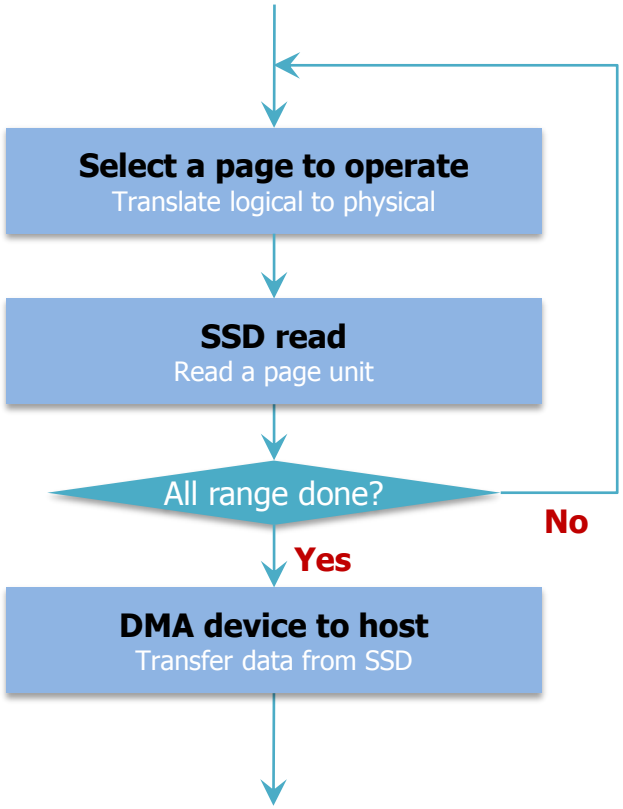
Command parsing: read

Command parsing: write

HANYANG UNIVERSITY ENC Lab Embedded Network Computing Lab.

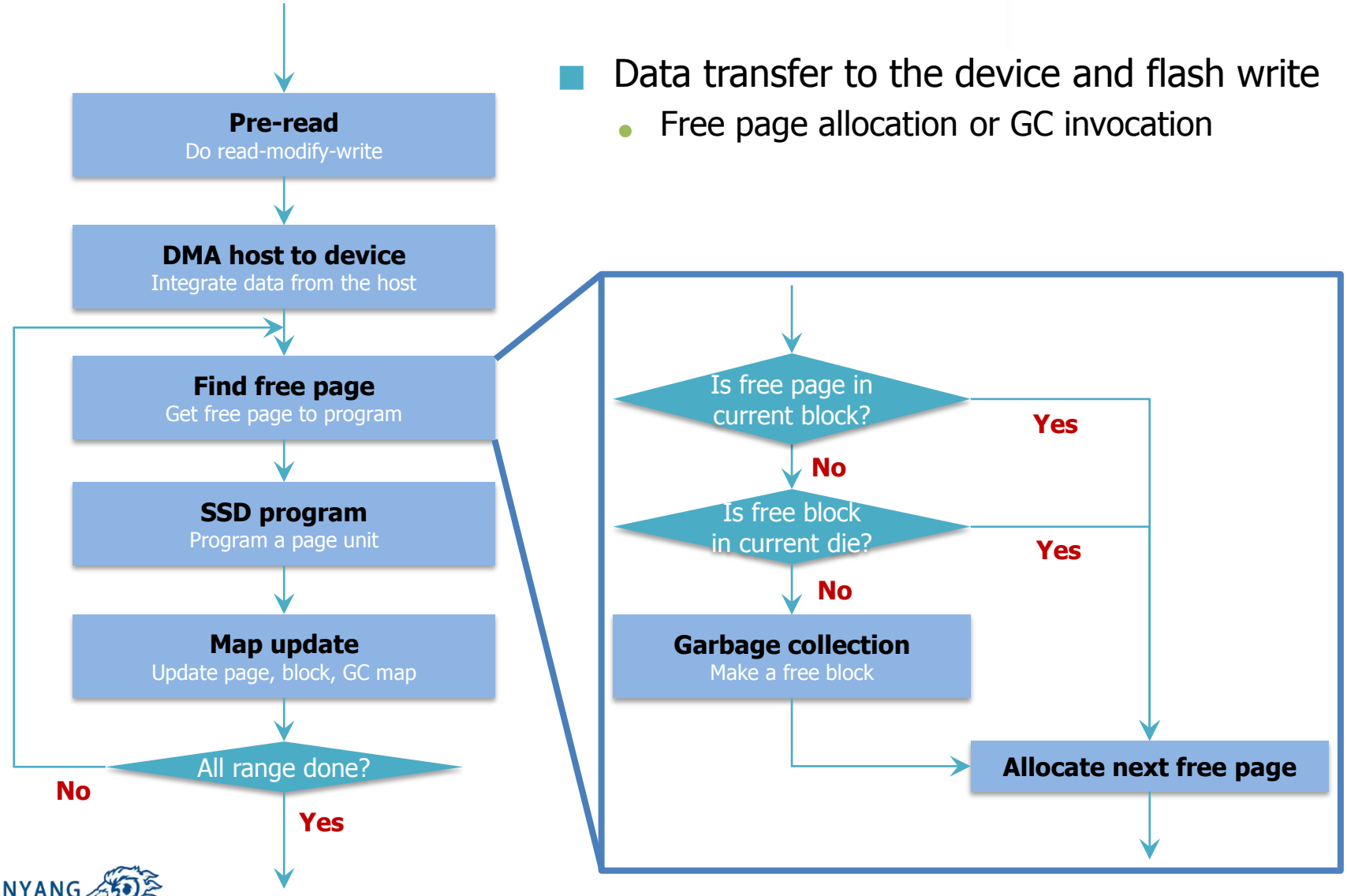
Read Sequence

- Flash read and data transfer to the host



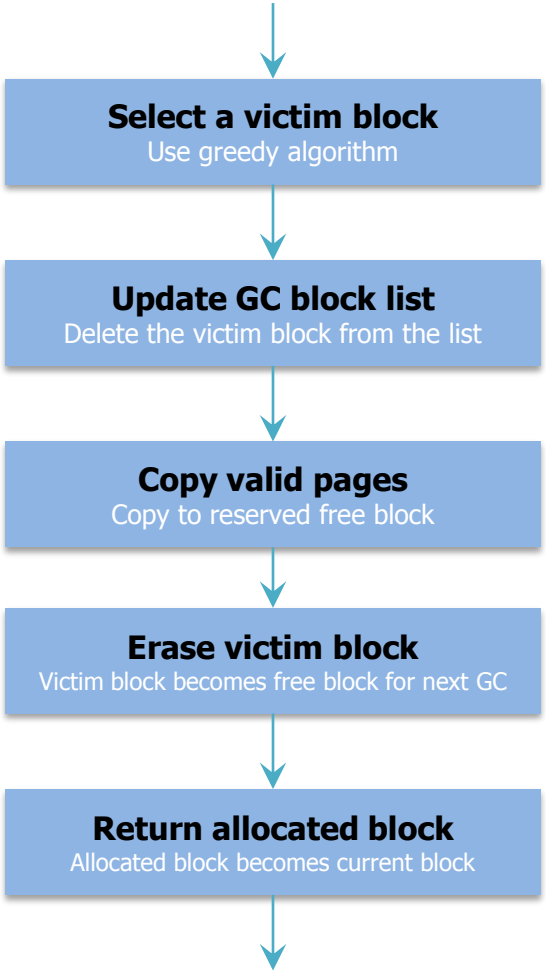
Write Sequence

- Data transfer to the device and flash write
  - Free page allocation or GC invocation



GC Sequence

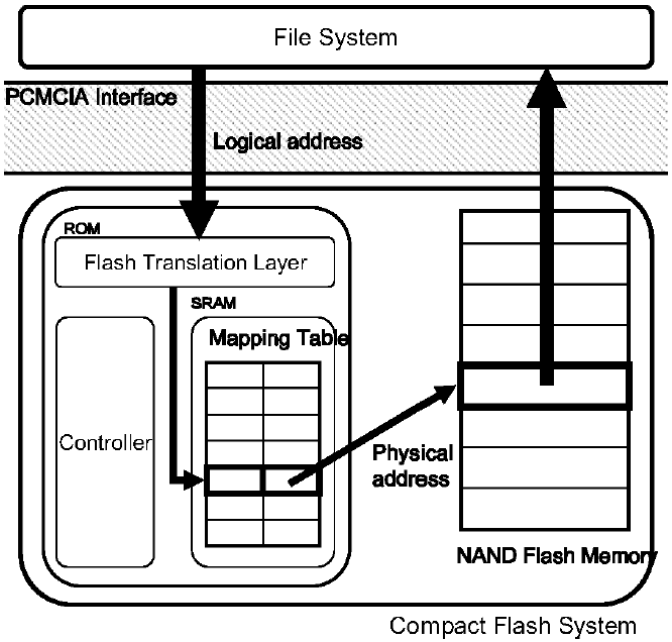
- Victim selection and valid page migration



Page-level Mapping

Concept of Address Mapping

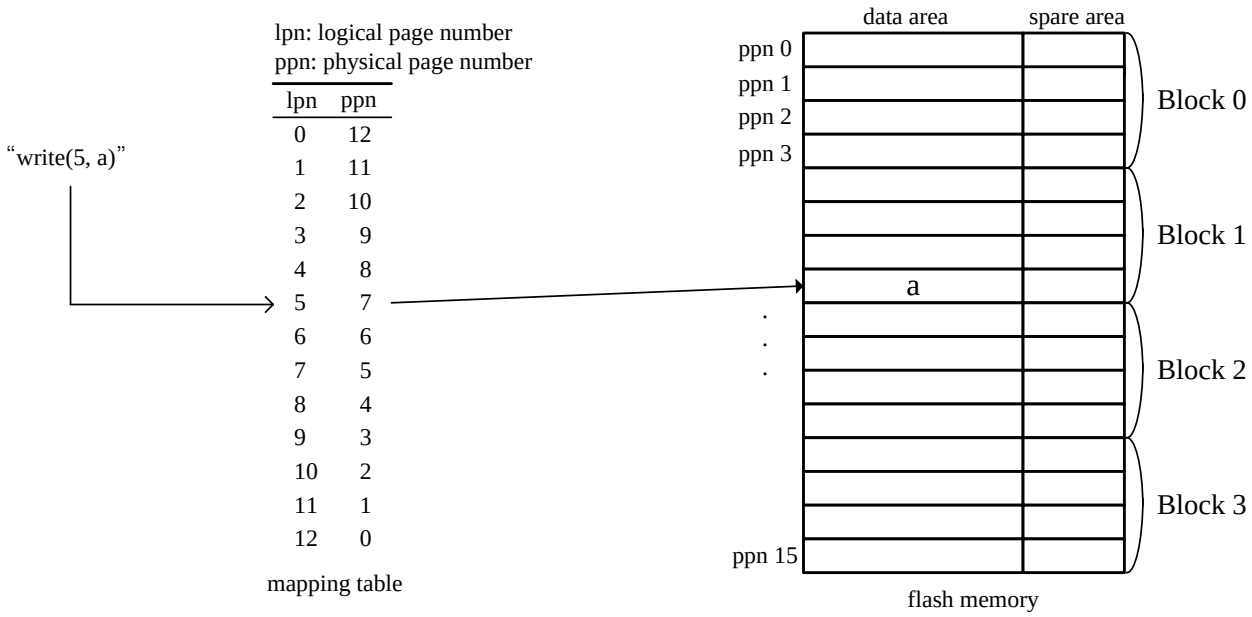
- Definition
  - A software layer that emulates the functionality of an HDD while hiding the peculiarities of flash memory (Logical to physical address mapping)
- Purposes of address mapping
  - To overcome erase-before-write limitation
  - To overcome limited program/erase cycles
    - Redirect each write request from the host to an empty area that has been already erased



*reference: A Log Buffer-Based Flash Translation Layer Using Fully-Associative Sector Translation

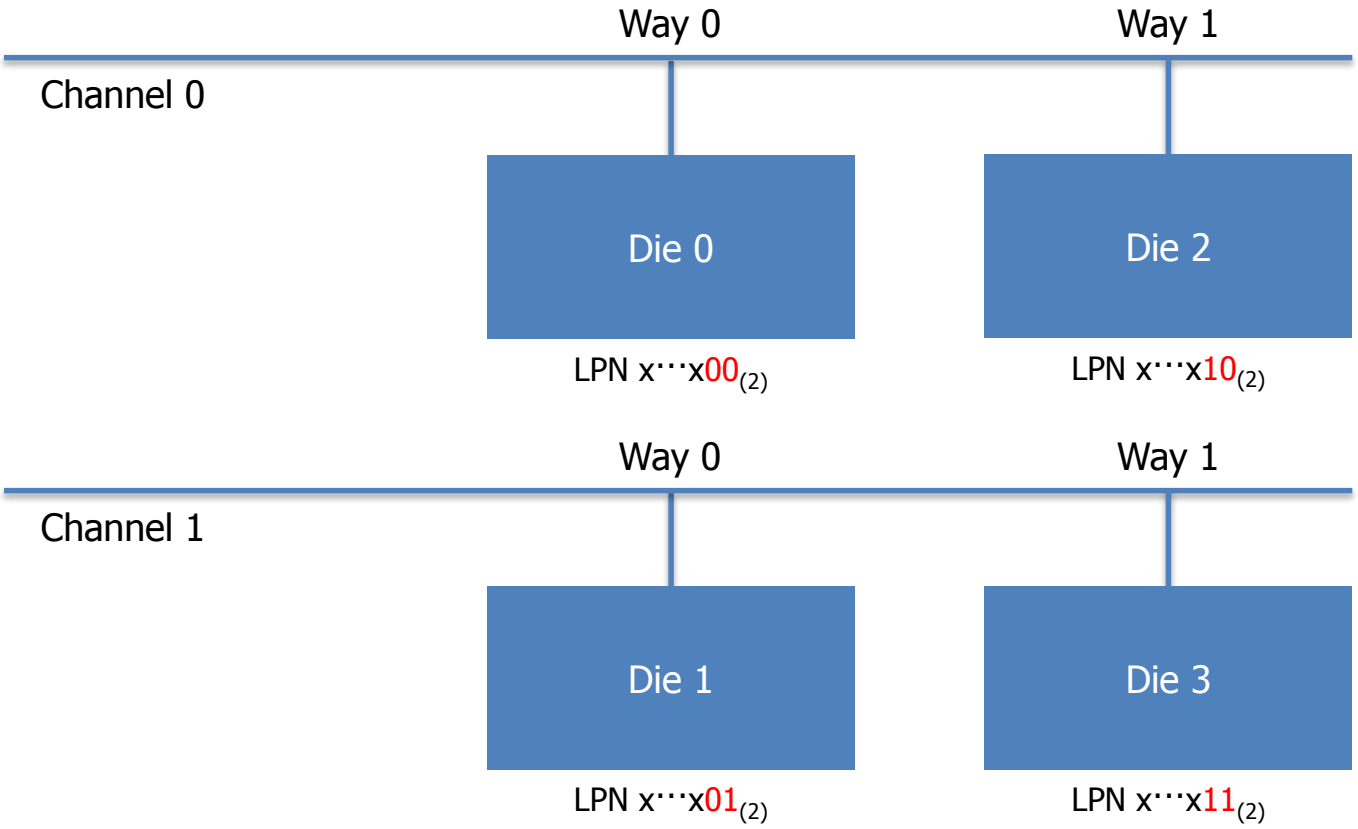
Page-level Mapping

- Main Idea
  - Every logical page is mapped to a corresponding physical page
- Advantage
  - Better performance over random write than block-level mapping
- Disadvantage
  - Huge amount of memory space requirement for the mapping table



Static Mapping

- Mapping tables are managed within a die



Each LPN is deterministically mapped to specific die (2-channel, 2-way)

LPN: Logical Page Number

FTL Metadata

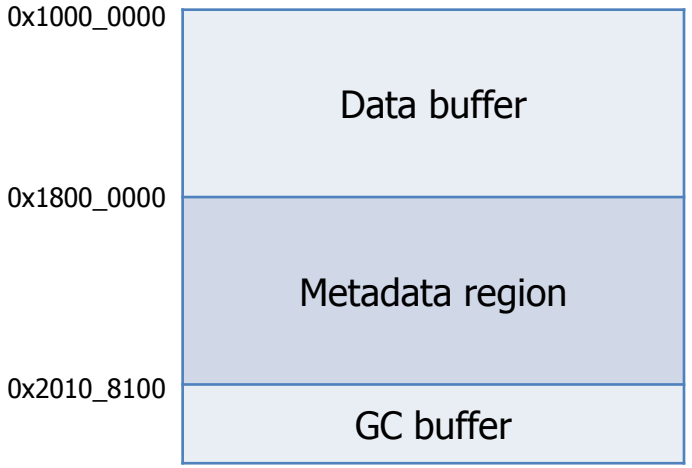
Item	Contents
Page map table	• **Physical Page Number (PPN)**: PPN to which a logical page is mapped • **Valid**: Flag for whether a physical page is valid or invalid • **Logical Page Number (LPN)**: LPN of a physical page
Block table	• **Bad**: Flag for bad block management • **Free**: Flag for a block that is erased and available for programming • **Erase count**: The number of erase operations done on a block • **Invalid page count**: The number of invalid pages in a block • **Current page**: Free page allocated for programming in a block • **Previous/next block**: Pointer to previous/next block used in GC block list
Die table	• **Current block**: Open block allocated for programming in a die • **Free block**: Reserved free block for garbage collection in a die
GC table	• **Head**: The head of GC block list • **Tail**: The tail of GC block list

*Current release doesn't flush the FTL metadata to NAND flash memories on shutdown

Source file: pagemap.h

DRAM Buffer Usage

- DRAM buffer is divided into three sections
  - Data buffer
    - Data for I/O are stored temporarily
  - Metadata region
    - Page map table
    - Block table
    - Die table
    - GC table
  - GC buffer
    - Valid pages of GC are stored temporarily



Three sections of DRAM buffer

Source file: host_controller.h, pagemap.h

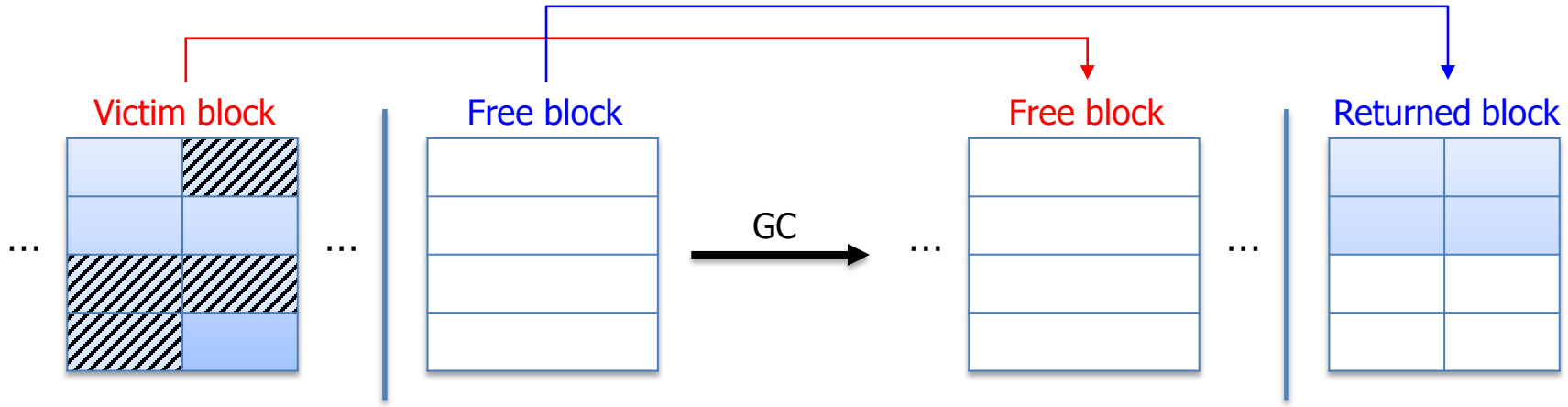
Greedy Garbage Collection

Concept of Garbage Collection

- Why is garbage collection needed
  - To reclaim new free blocks for future write requests
    - Invalid data occupy storage space before GC
- What is garbage collection
  - Copies the valid data into a new free block and erases the original invalid data
  - Basic operations involved in GC are the following
    - 1. The victim blocks meeting the conditions are selected for erasure
    - 2. The valid physical pages are copied into a free block
    - 3. The selected physical blocks are erased
- What is important in GC
  - Victim block selection
    - GC time depends on the status of victim block

On-demand GC

- GC Trigger
  - Each GC is triggered independently of other dies
  - GC is triggered when there is no free user block of each die
- Blocks in GC
  - One block per die is overprovisioned
  - Single victim block is a target of GC

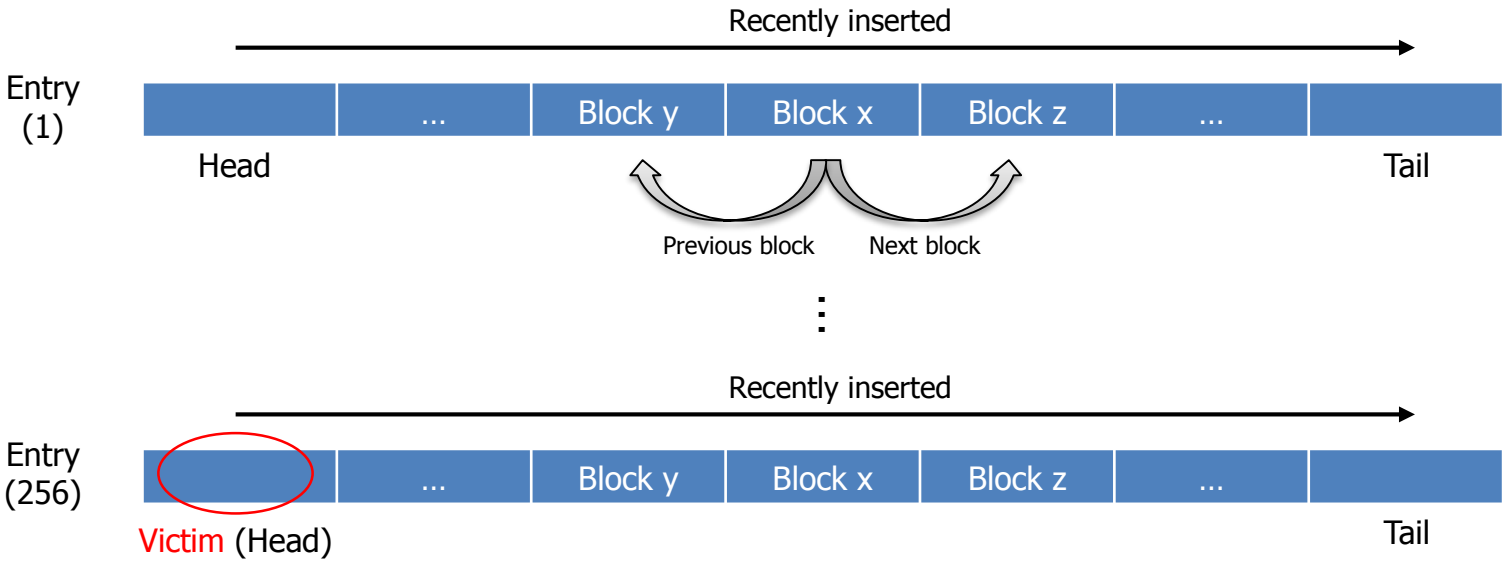


Valid pages in victim block are copied to free block and the role of two blocks are swapped

Source file: pagemap.c

Victim Block Selection

- Greedy algorithm
  - Victim block is selected in such a way to minimize GC cost
  - The more the number of invalid pages, the less the cost of GC
- GC block list
  - Victim search time is minimized

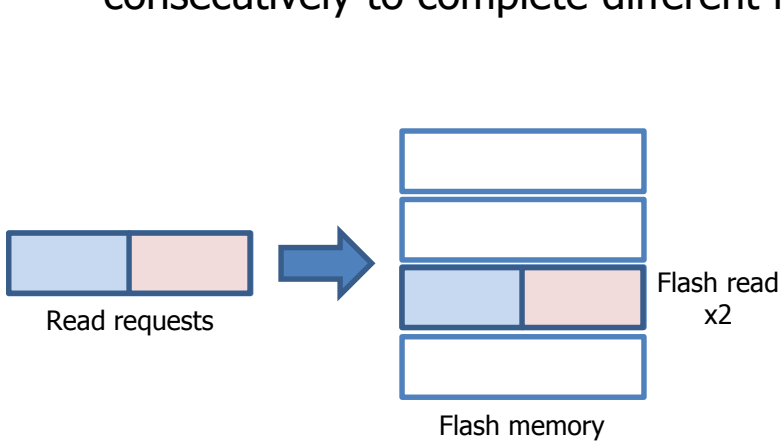


Example of GC block list when the number of pages per block is 256

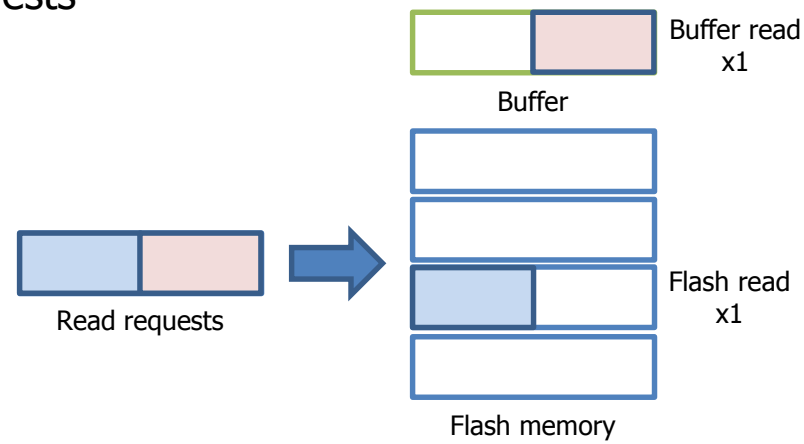
Source file: pagemap.c

Known Issues

- No FTL metadata flush
  - Reliability of spare area in a page is not guaranteed by ECC
  - LPN cannot be stored in the spare area of corresponding PPN
- Long formatting time
  - Requests from format have spatial locality within a page
  - Without buffer management, low level operation is done at the same page consecutively to complete different requests



Two read requests w/o buffer management



Two read requests w/ buffer management

Authors

Name	E-mail	Contribution
Gyeongyong Lee	gylee@enc.hanyang.ac.kr	2014-04 ~ Now
Jaewook Kwak	jwkwak@enc.hanyang.ac.kr	2015-05 ~ Now

Thank you