Cosmos+ OpenSSD Tutorial 2017

Prof. Yong Ho Song
Department of Electronic Engineering, Hanyang University

OpenSSD Introduction

OpenSSD Motivation

Need a SSD platform

- to develop a new firmware algorithm

- to explore hardware architecture and organization

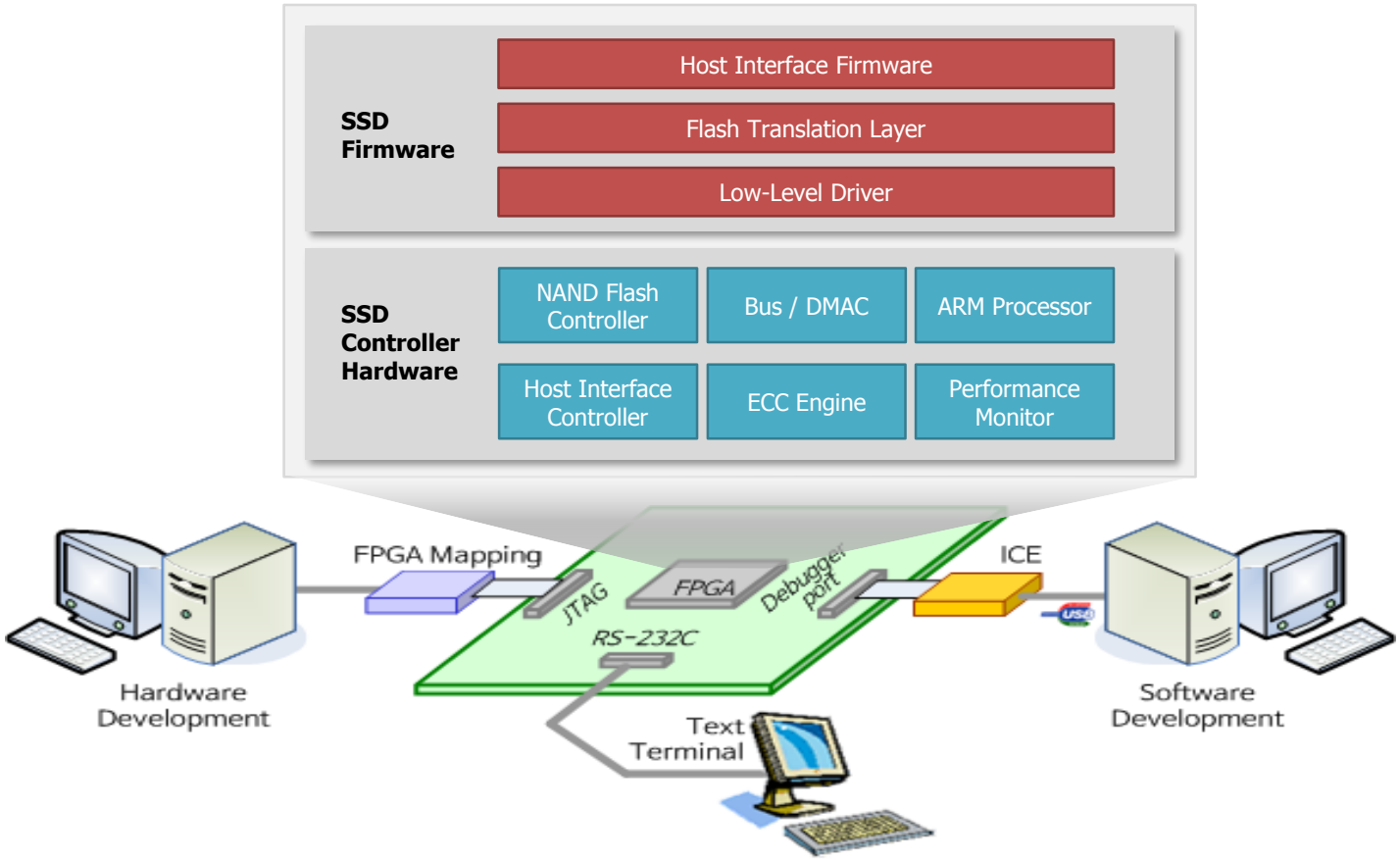
Use a commercial product as a platform?

- little information on HW/SW

- no way to change controller SoC

What's the OpenSSD Project?

Open source SSD design used for research and education



OpenSSD Project History

- **Open-source SSD platforms**
  - Jasmine OpenSSD (2011)
  - Cosmos OpenSSD (2014)
  - Cosmos+ OpenSSD (2016)

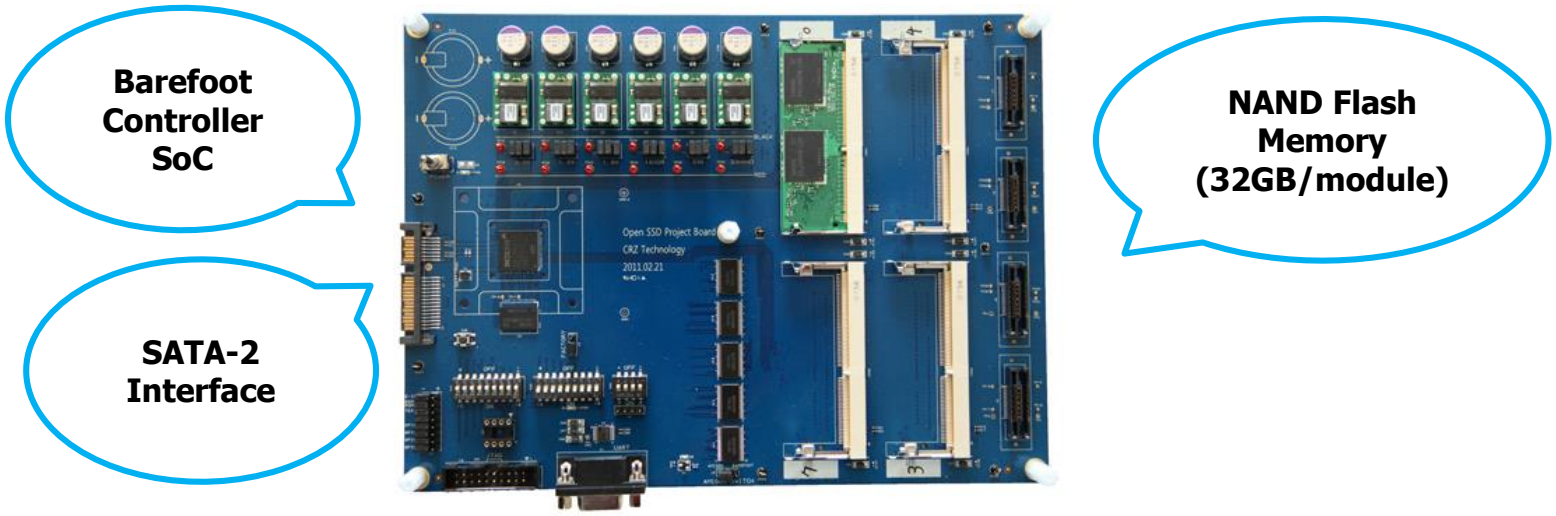
- **Cosmos/Cosmos+ OpenSSD: FPGA-based platform**
  - Could modify SSD controller and firmware
  - Could add new hardware and software functionality

Why OpenSSD

- **Realistic research platform**
  - Solve your problem in a real system running host applications
  - Design your own SSD controller (hardware and firmware), if possible
- **Information exchange**
  - Share your solution with people in society
- **Community contribution**
  - Open your own solution to public
- **Expensive custom-made storage system**
  - Unique
- **Play for fun**

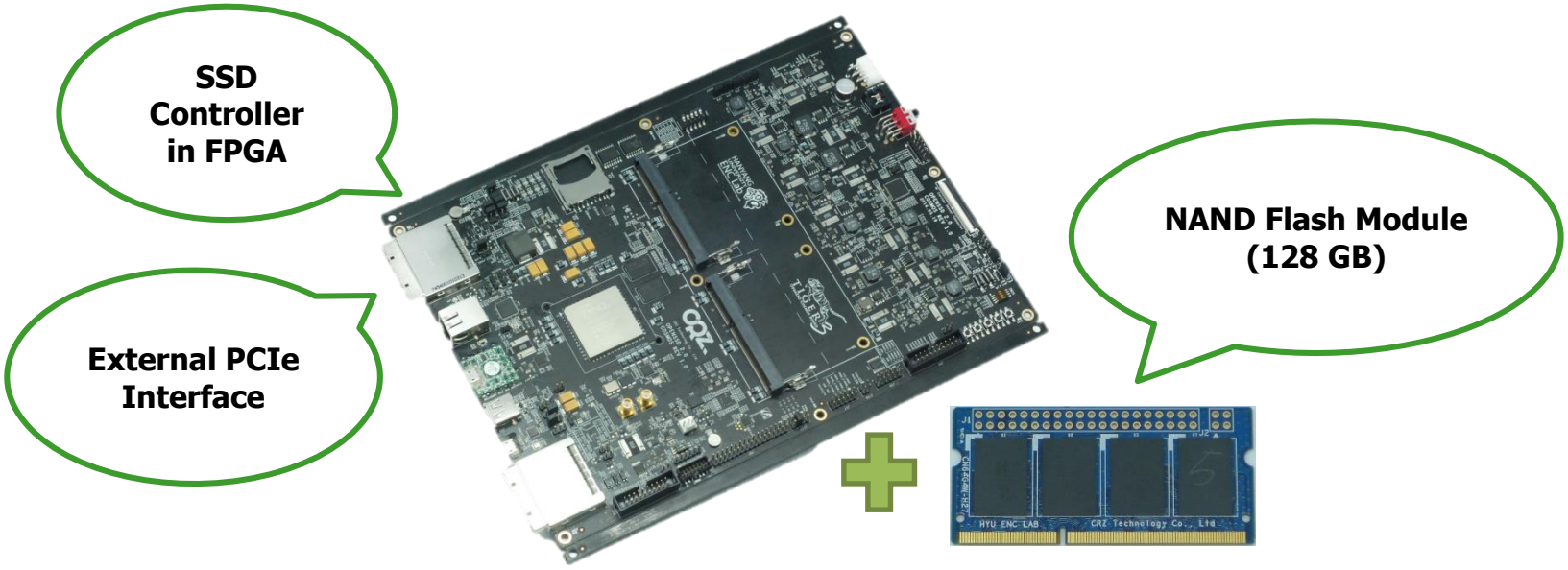
1st OpenSSD (Indilinx)

- **Jasmine OpenSSD (2011)**
  - SSD controller: Indilinx Barefoot (SoC w/SATA2)
  - Firmware: SKKU VLDB Lab
  - Users from 10+ countries



2nd OpenSSD (Hanyang University)

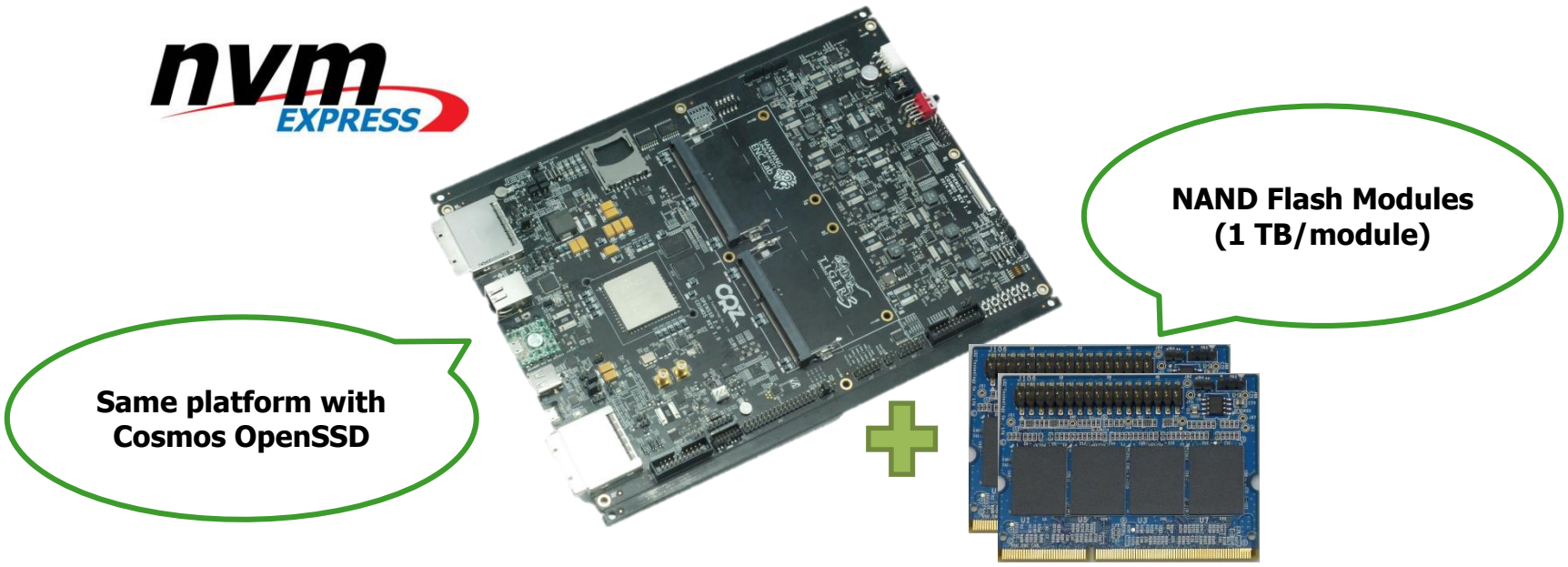
- **Cosmos OpenSSD (2014)**
  - SSD controller: HYU Tiger 3 (FPGA w/PCIe Gen2)
  - Firmware: HYU ENC Lab
  - Users from 5 countries (mostly in USA)



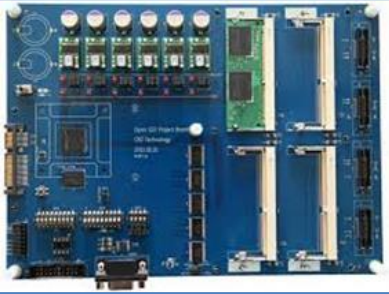
HANYANG UNIVERSITY ENC Lab Embedded Network Computing Lab.

3rd OpenSSD (Hanyang University)

- **Cosmos+ OpenSSD (2016)**
  - SSD controller: HYU Tiger 4 (FPGA w/NVMe over PCIe Gen2)
  - Same main board with different memory modules
  - Firmware: HYU ENC Lab
  - Users from ?? countries



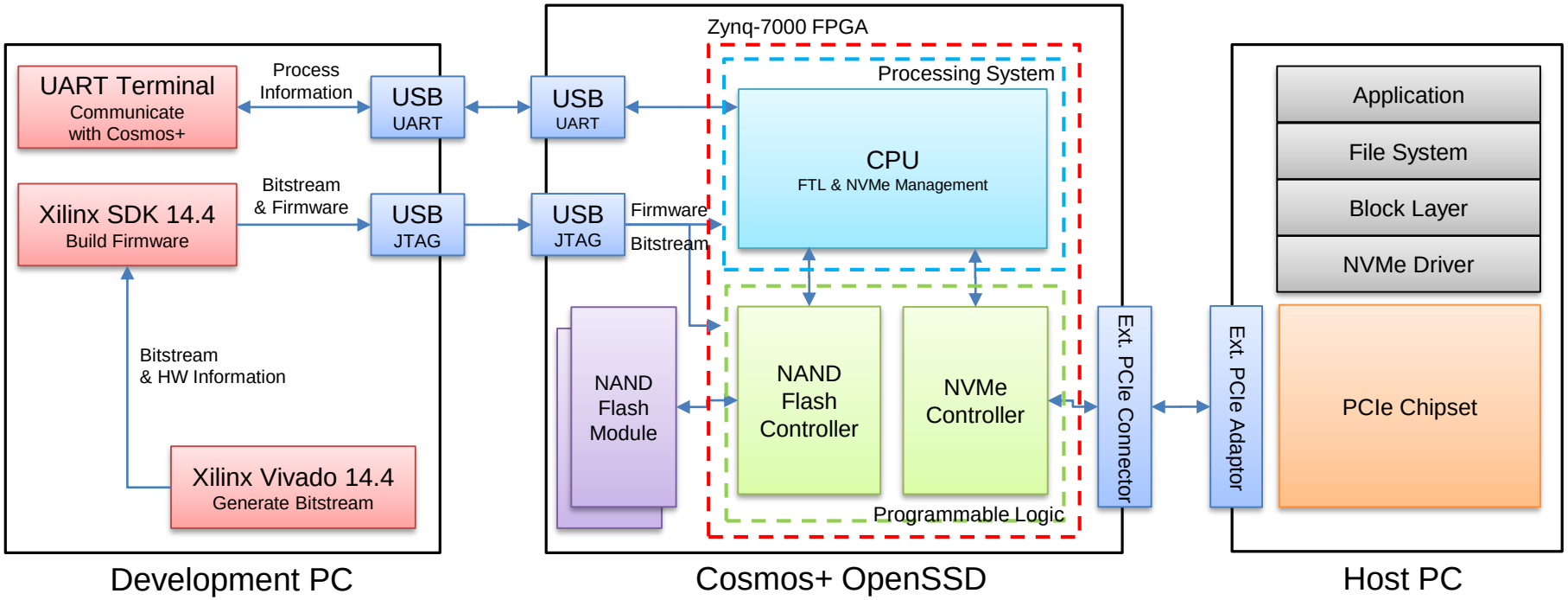
Platform Comparison

	Jasmine OpenSSD	Cosmos OpenSSD	Cosmos+ OpenSSD
Released in	2011	2014	2016
Main Board			
SSD Controller	Indilinx Barefoot (SoC)	HYU Tiger3 (FPGA)	HYU Tiger4 (FPGA)
Host Interface	SATA2	PCIe Gen2 4-lane (AHCI)	PCIe Gen2 8-lane (NVMe)
Maximum Capacity	128 GB (32 GB/module)	256 GB (128 GB/module)	2 TB (1 TB/module)
NAND Data Interface	SDR (Asynchronous)	NVDDR (Synchronous)	NVDDR2 (Toggle)
ECC Type and Strength	BCH, 16 bits/512 B	BCH, 32 bits/2 KB	BCH, 26 bits/512 B

OpenSSD Project Homepage

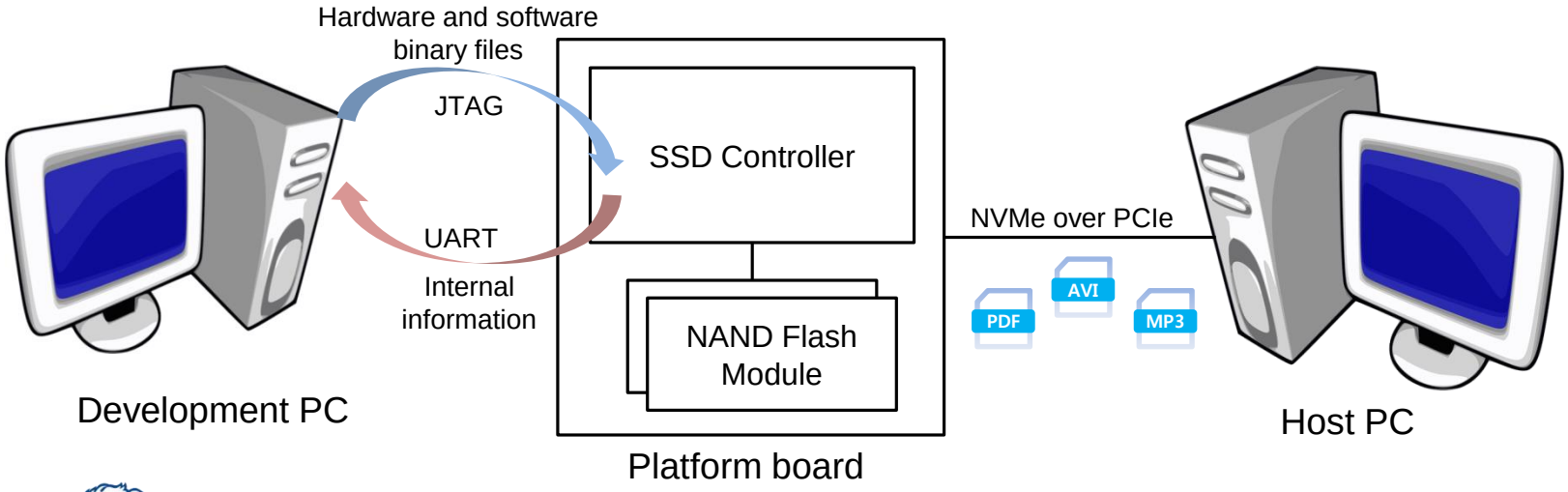
http://www.openssd-project.org

Cosmos+ OpenSSD Overview



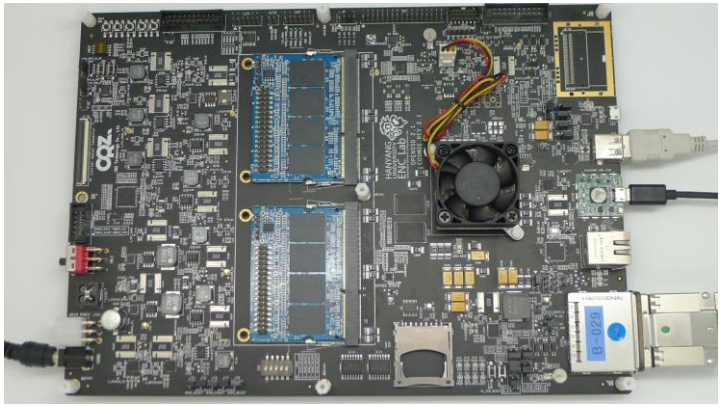
Cosmos+ OpenSSD Environment

- **1 Development PC**
  - Downloading hardware/software design (JTAG)
  - Monitoring Cosmos+ OpenSSD internals (UART)
- **1 Host PC**
  - Executing applications such as a benchmark (PCIe)
- **1 Platform board with 1+ NAND flash modules installed**
  - Working as a storage device to the host PC

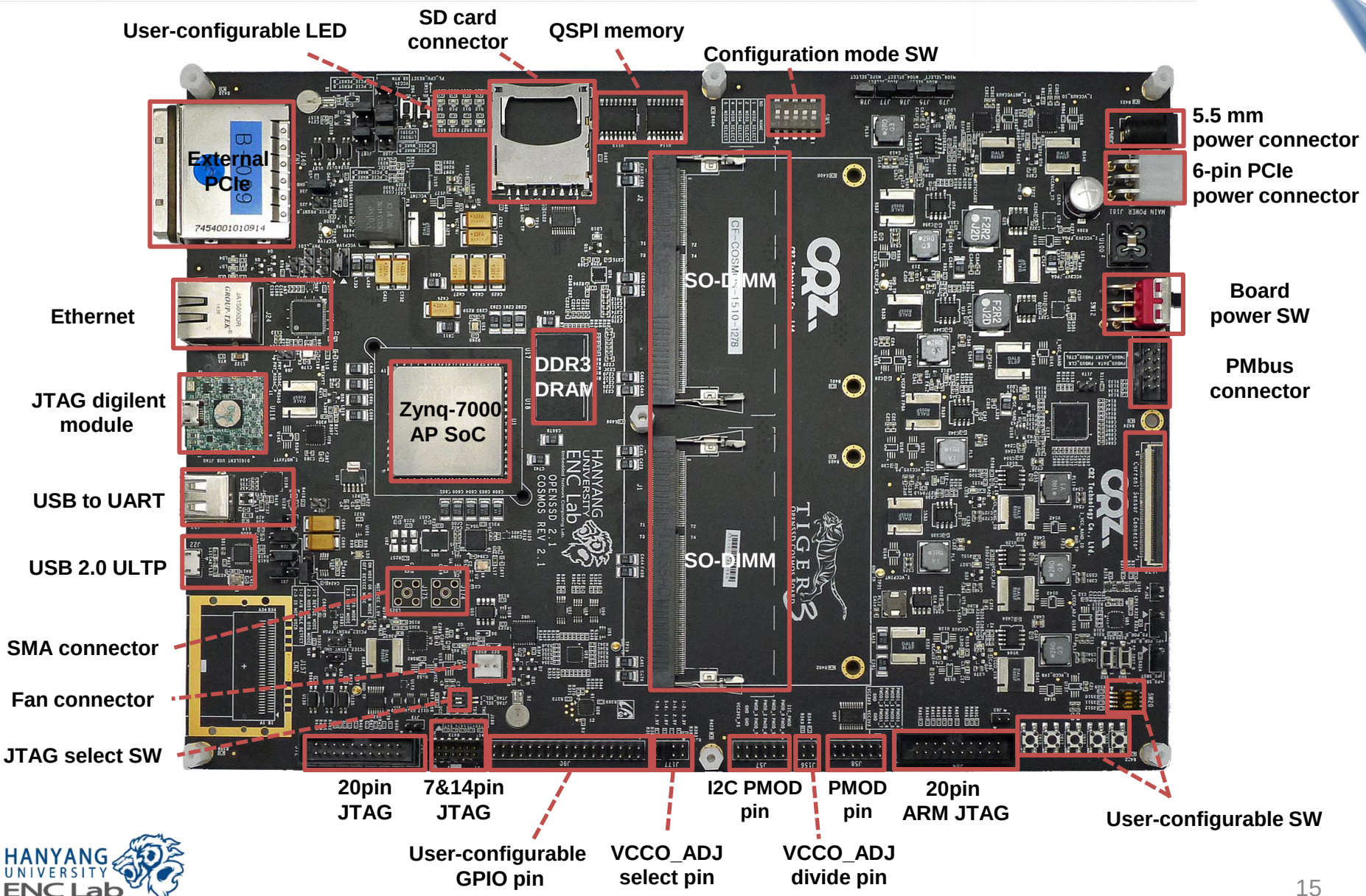


Hardware Components

- **Cosmos+ OpenSSD platform board**
  - Consists of a Zynq FPGA and other peripherals
- **NAND flash modules**
  - Configured as multi-channel and multi-way flash array
  - Inserted into Cosmos+ OpenSSD platform board
- **External PCIe adapter and cable**
  - Connected with host PC
- **USB cables for JTAG and UART**
  - Connected with development PC
- **Power cable and adapter**
  - 12V supply voltage



Cosmos+ OpenSSD Platform Board

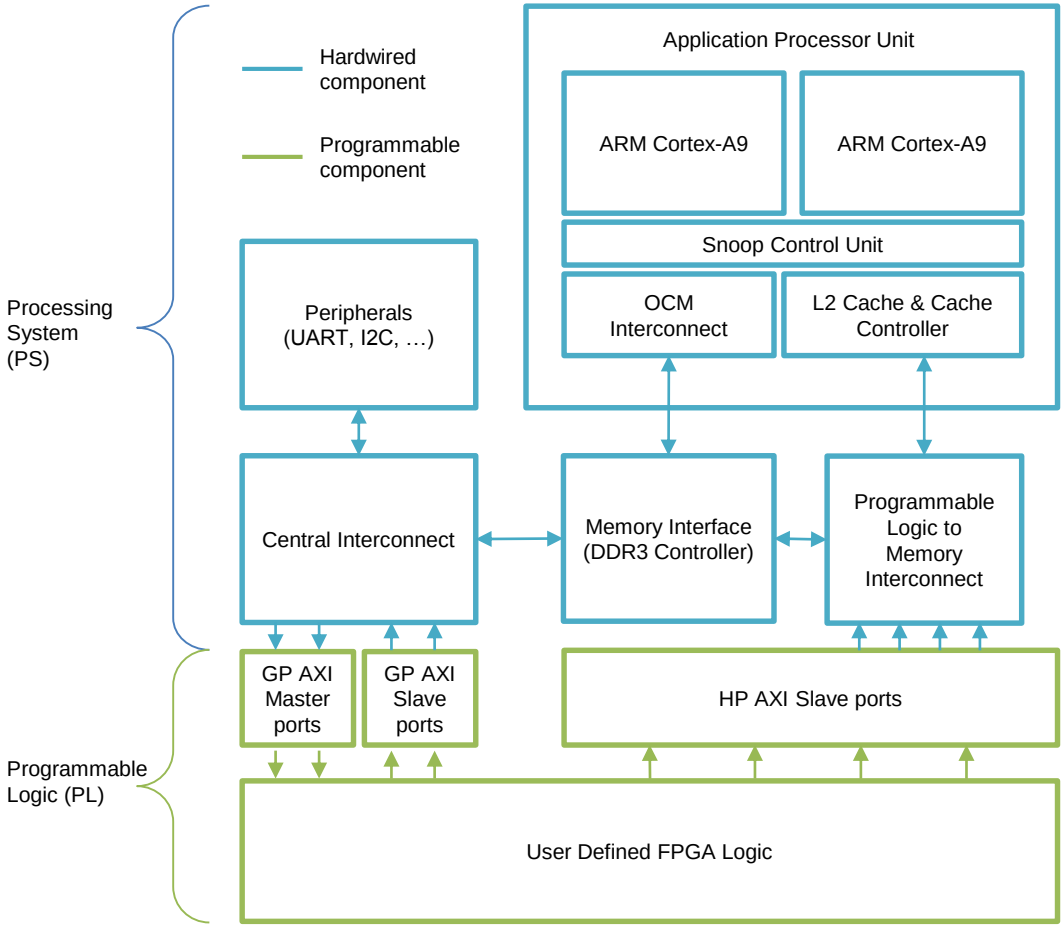


Primary Details

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	Up to 1000 MHz
Storage	**Total capacity**	Up to 2 TB (MLC)
	Organization	Up to 8-channel 8-way
DRAM	**Device interface**	DDR3 1066
	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)

Zynq-7000 FPGA Architecture

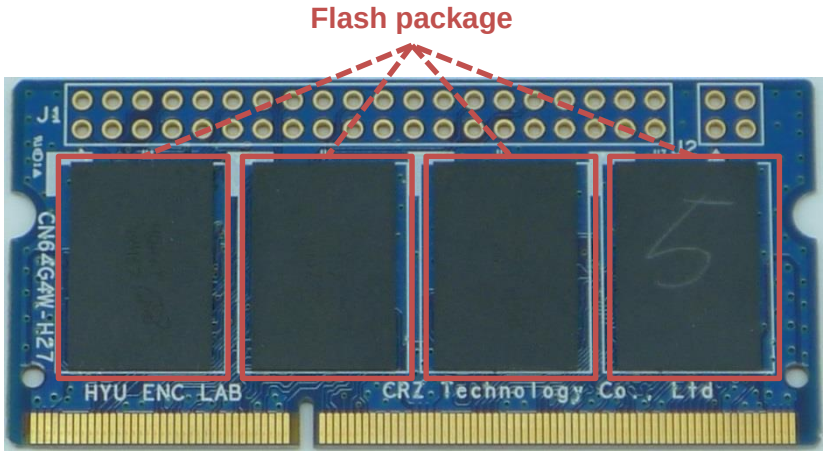
- **Xilinx's embedded SoC**
- **Two regions**
  - Processing System (PS)
    - Hardwired components
    - Executes the firmware program
  - Programmable Logic (PL)
    - Programmable components (FPGA region)
    - NAND flash controller (NFC) and NVMe controller reside in PL
- **Benefits of Using Zynq**
  - 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)



Zynq-7000 architecture overview

Cosmos OpenSSD NAND Module

- **Each module has 4 flash packages**
  - One flash package
    - Capacity: 32 GB
    - Page size: 8640 Bytes (spare area: 448 Bytes)
  - Synchronous NAND
- **Used with Tiger3 Controller**

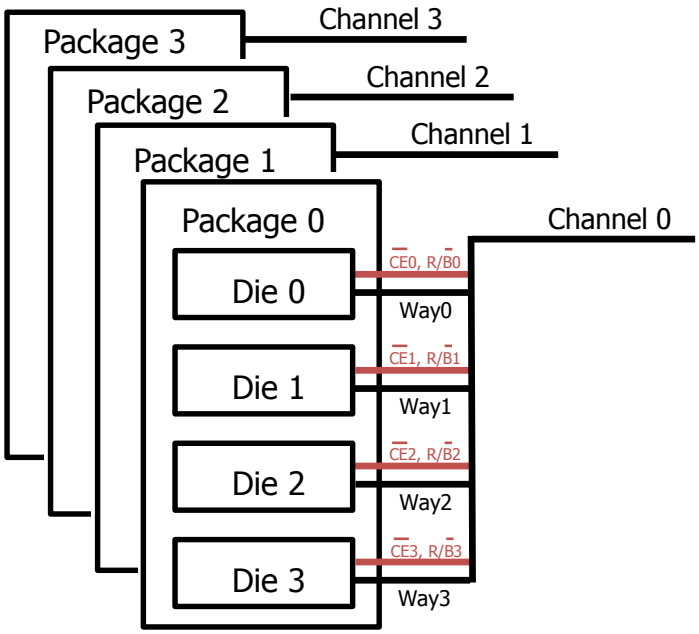


Front side

Rear side

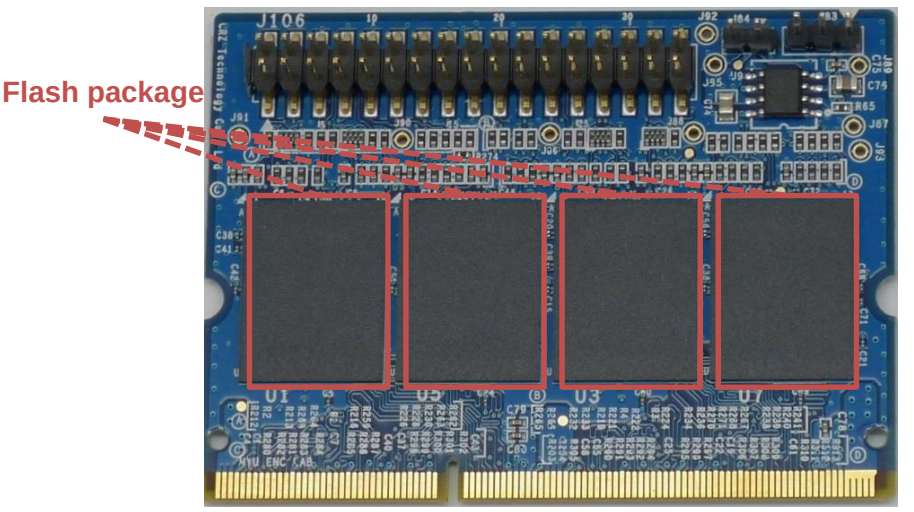
Logical Organization of Flash module

- **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 (CE)
  - Each die has own Ready/Busy (R/B) signal

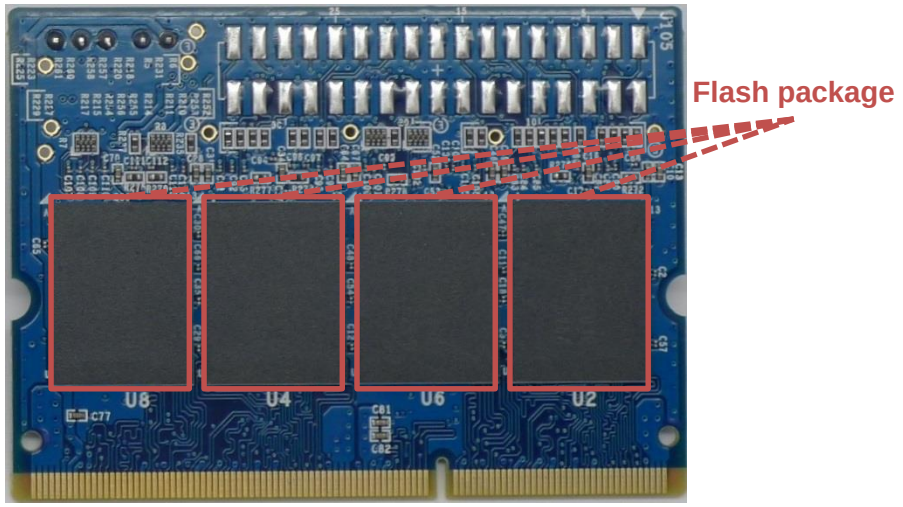


Cosmos+ OpenSSD NAND Module

- **Each module has 8 flash packages**
  - One flash package
    - Capacity: 128 GB
    - Page size: 18048 Bytes (spare area: 1664 Bytes)
  - Toggle NAND
- **Used with Tiger4 Controller**



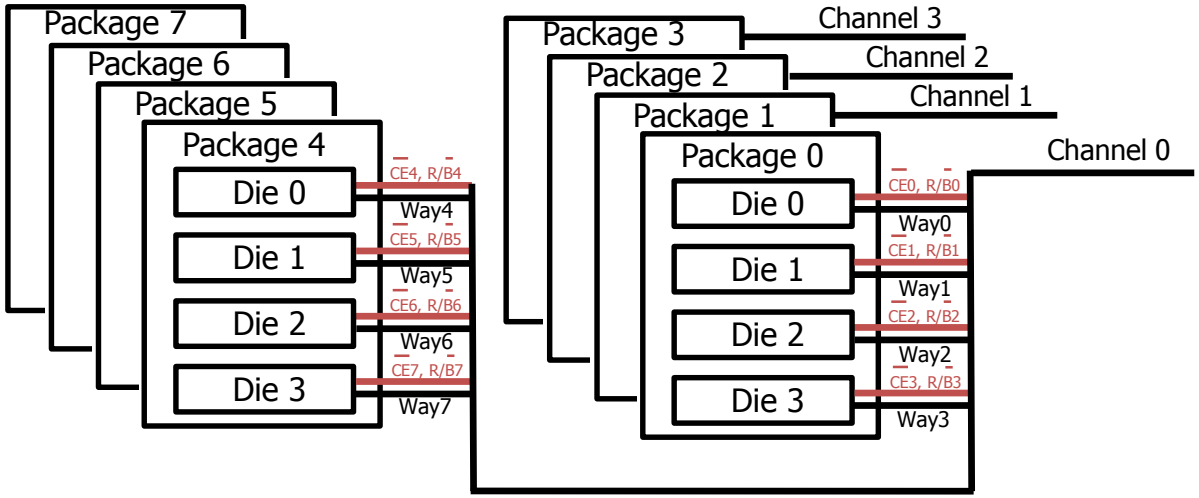
Front side



Rear side

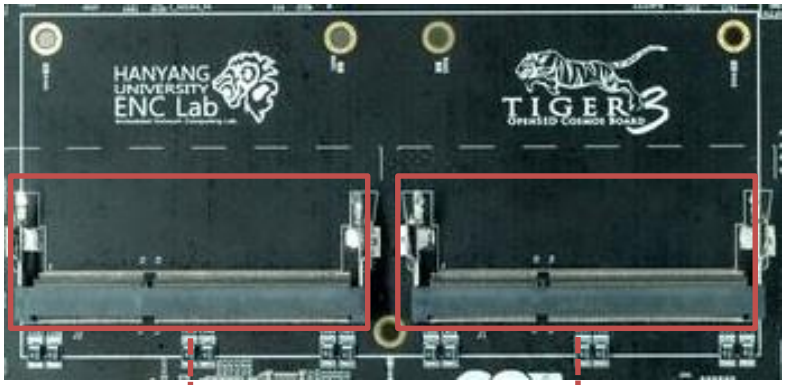
Logical Organization of NAND Flash Module

- **Module configuration**
  - 4-channels/module and 8-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 (CE)
  - Each die has own Ready/Busy (R/B) signal



NAND Module Setup

- **Cosmos OpenSSD**
  - Supports only one flash module slot (J1)
- **Cosmos+ OpenSSD**
  - Supports both flash module slots (J1, J2)



- **Caution**
  - Cosmos/Cosmos+ OpenSSD flash module slots have custom pin maps
  - You should not insert any SDRAM module into this slot

External PCIe

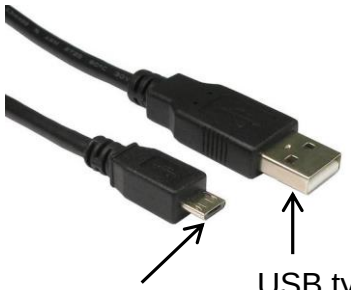
- **Expand PCIe Slot of host PC to connect external device**
- **Adapter card**
  - Installed on host PC
  - Provide a high-performance and low latency solution for expanding PCIe
- **External PCIe cable (8-lane)**
- **External PCIe connector (8-lane) on platform board**
  - 2.5 GT/s for a Gen1, 5.0 GT/s for a Gen2
  - Connected with high data rate serial transceiver in FPGA

External PCIe adapter

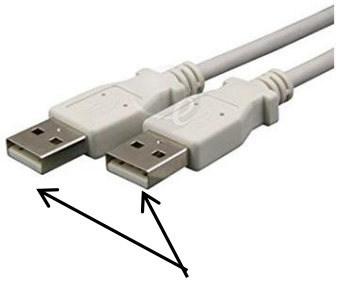
External PCIe cable

Connection with Development PC

- **JTAG cable**
  - Used for downloading hardware and software binary files
  - Available cable types
    - USB type A to USB type micro B cable
    - Emulator, JTAG N pin cable (N: 7, 14, 20)
- **UART cable**
  - Used for monitoring internal processes of Cosmos+ OpenSSD
  - USB type A to USB type A cable



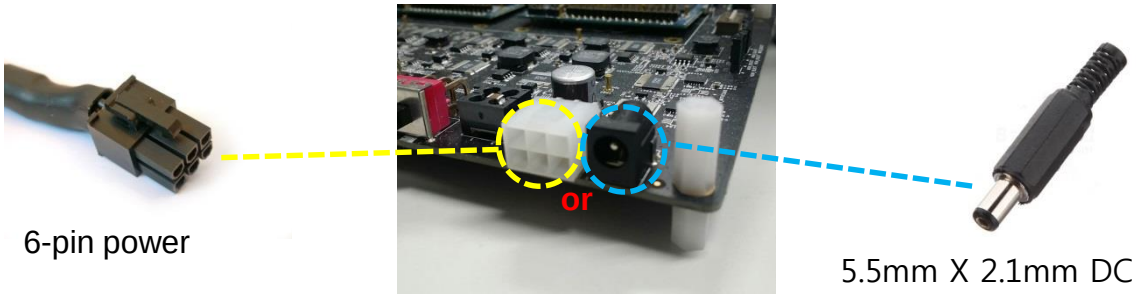
Available JTAG cables



UART cable

Power Connection

- **Single-source of power to the platform board**
  - 6-pin power connector (J181) or 5.5mm X 2.1mm DC power plug (J182)



- **The 6-pin connector looks similar to the regular PC 6-pin PCIe connector**

  Note: Difference in pin assignment between two connectors

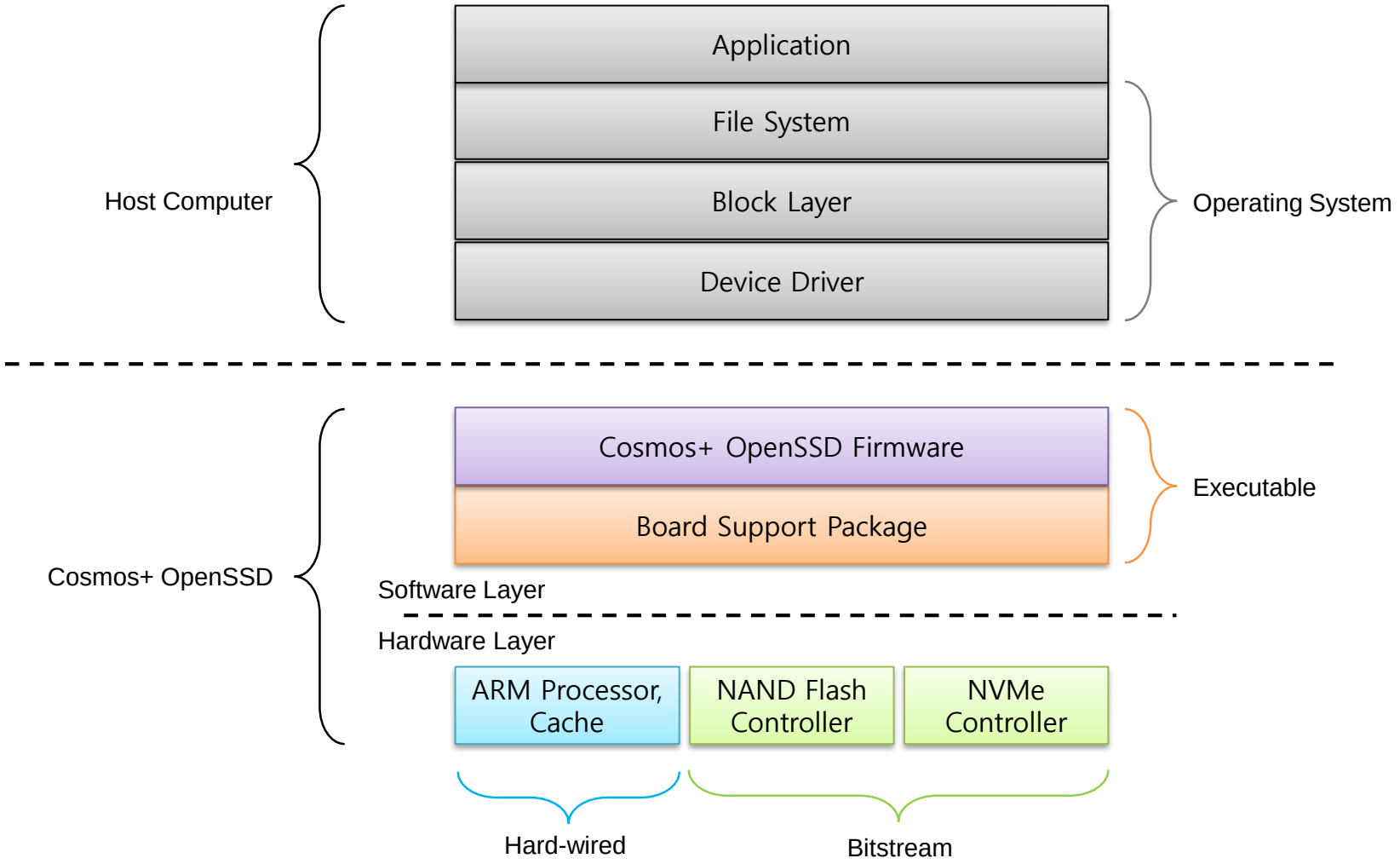
Connector	Pin map					
	1	2	3	4	5	6
Platform board 6-pin power	12V	12V	NC	NC	GND	GND
PC 6-pin PCIe power	GND	GND	GND	12V	12V	12V

- **Caution**
  - Do not plug PC 6-pin PCIe power cable in platform board 6-pin power connector (J181)

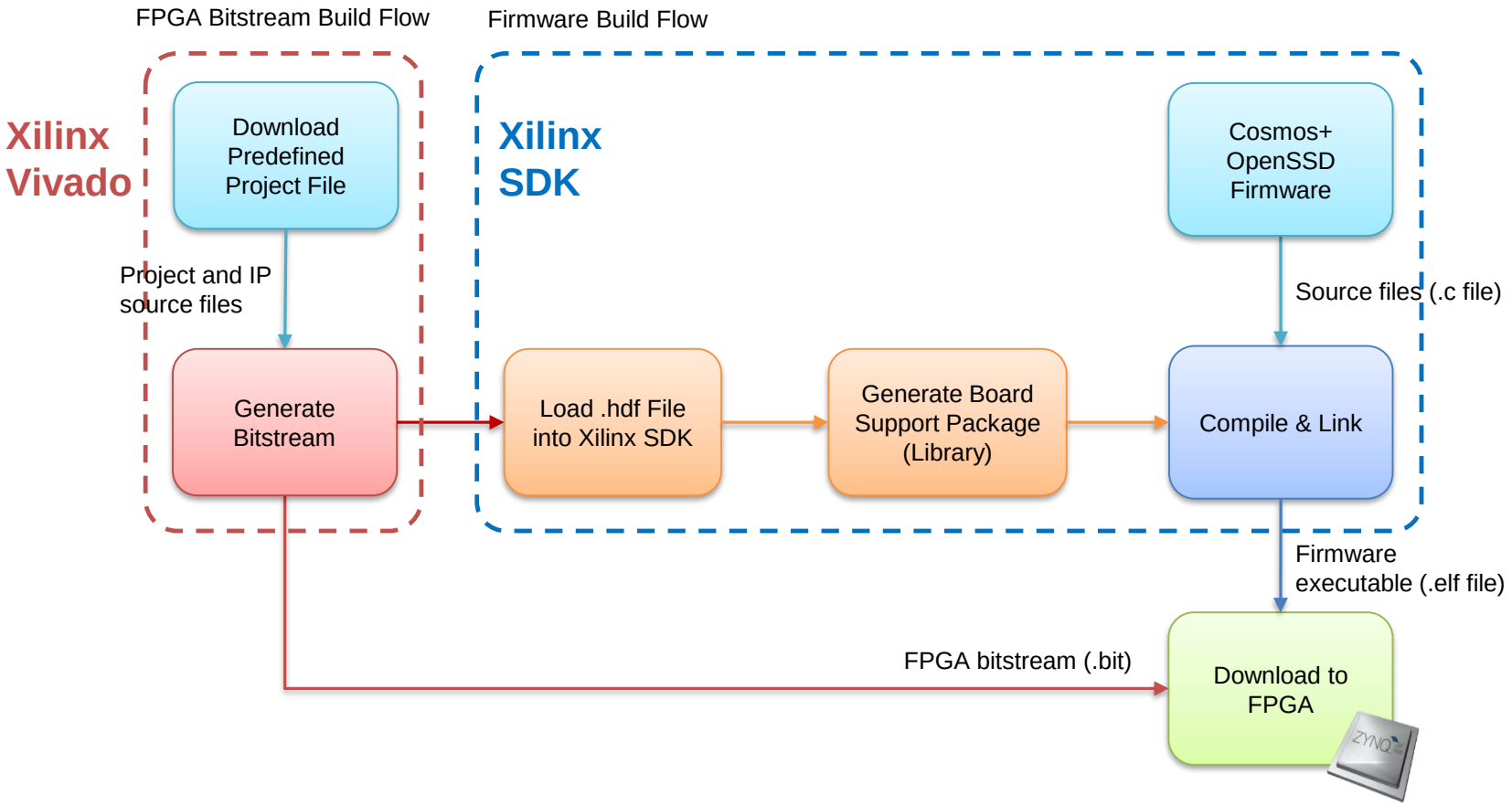
Development Software Components

- **Xilinx Vivado**
  - Generates a FPGA bitstream
  - Exports the generated FPGA bitstream to Xilinx SDK
- **Xilinx SDK**
  - Builds a SSD controller firmware
  - Downloads a FPGA bitstream and a firmware to the Zynq FPGA
- **FPGA bitstream**
  - Used to configure the programmable logic side of Zynq FPGA
- **Firmware**
  - Manages the NAND flash array
  - Handles NVMe commands

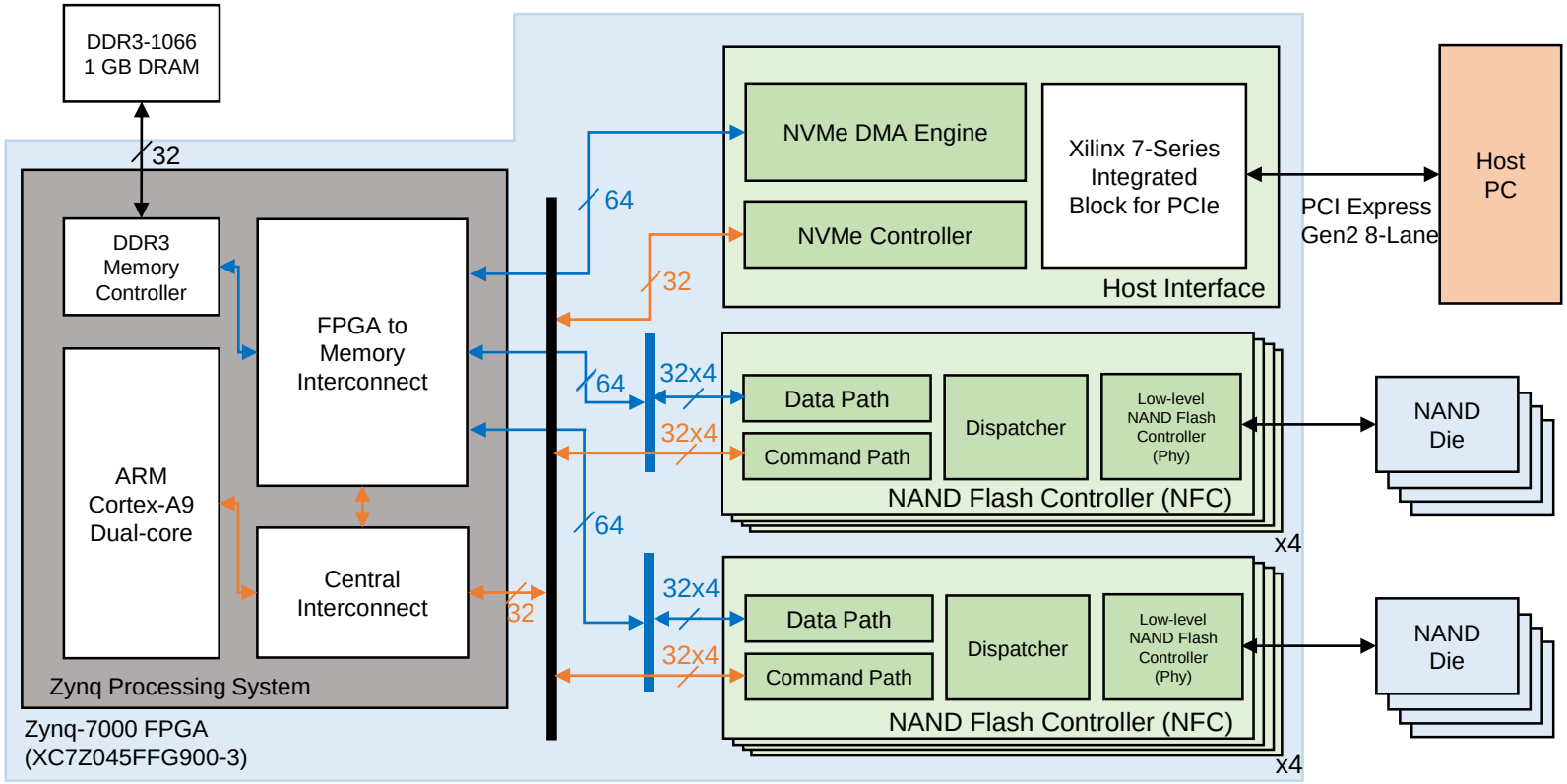
Host System Software Architecture



Software Porting Flow

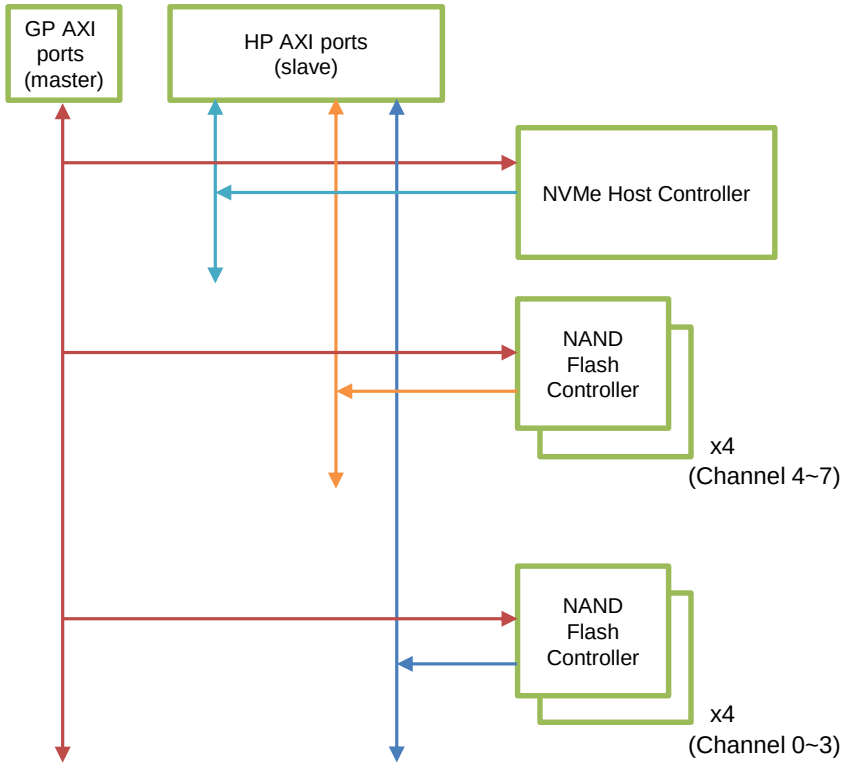


Cosmos+ OpenSSD Internal System Overview

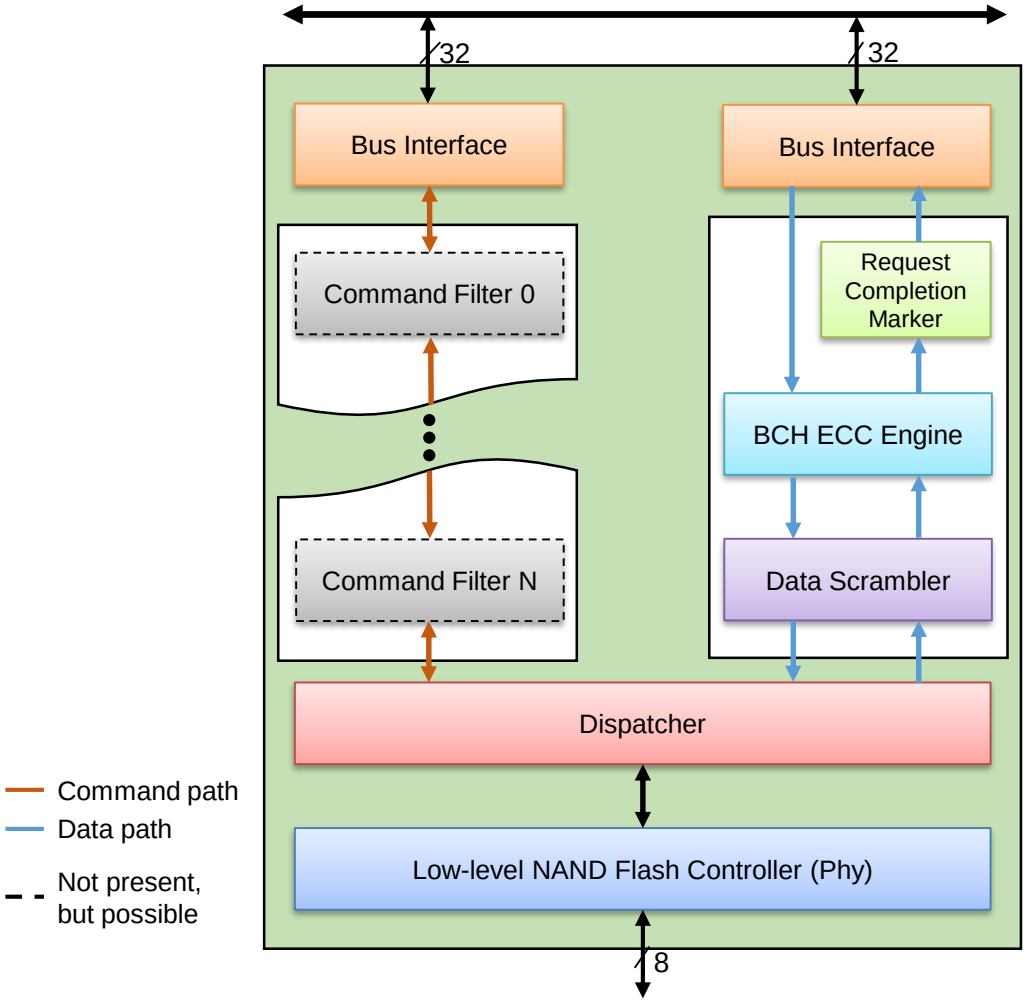


System Bus Structure

- **General Purpose (GP) AXI4 Lite bus**
  - 32bits interface
  - Used for control
  - Operates @ 100MHz
- **High Performance (HP) AXI4 bus**
  - 64bits interface
  - Used for Direct Memory Access (DMA)
  - Operates @ 250 MHz

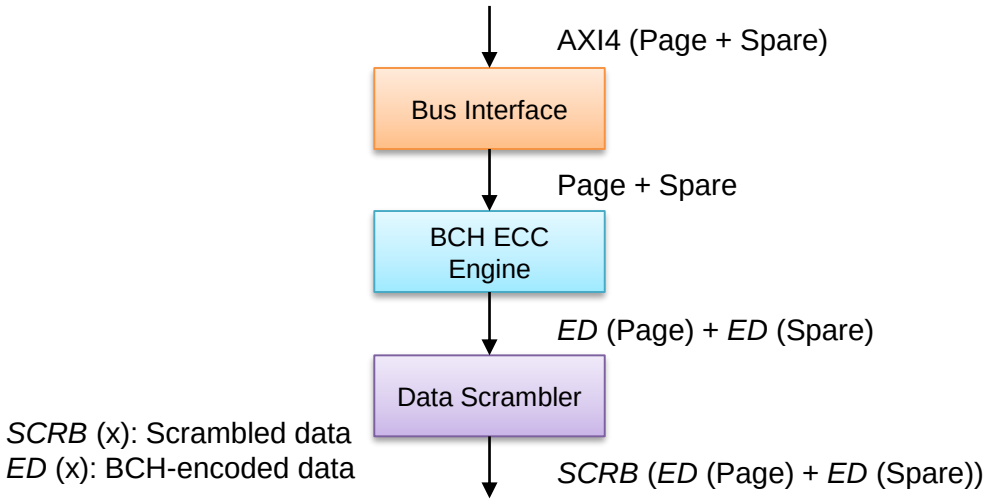


NAND Flash Controller Overview



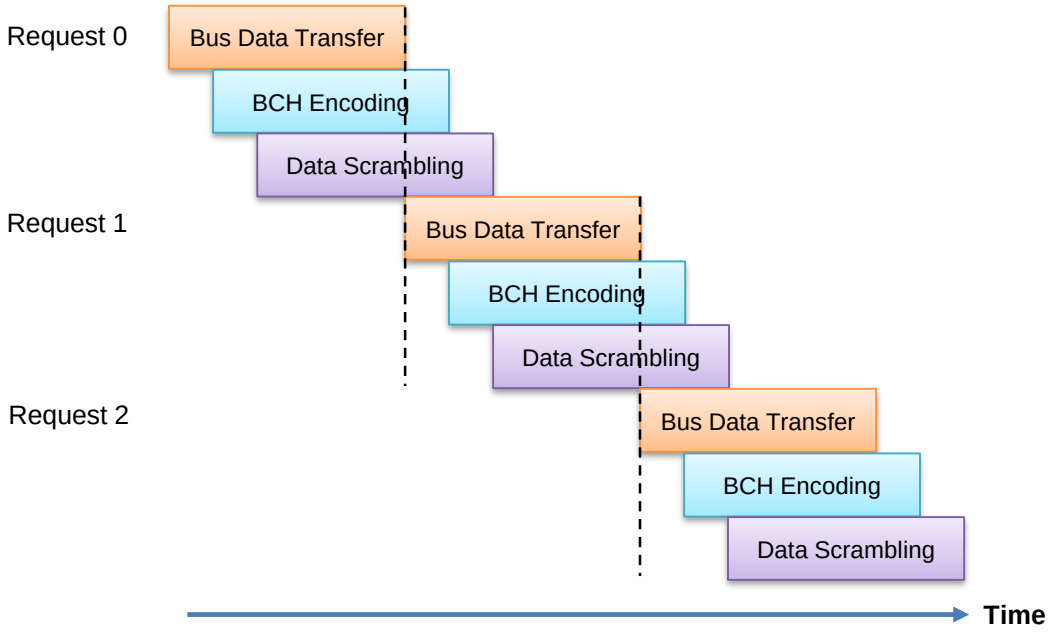
Layered and Modular NFC Design

- **Commands and data streams are encapsulated or decapsulated throughout modules in a layer**
- **Users can insert or remove modules more easily**



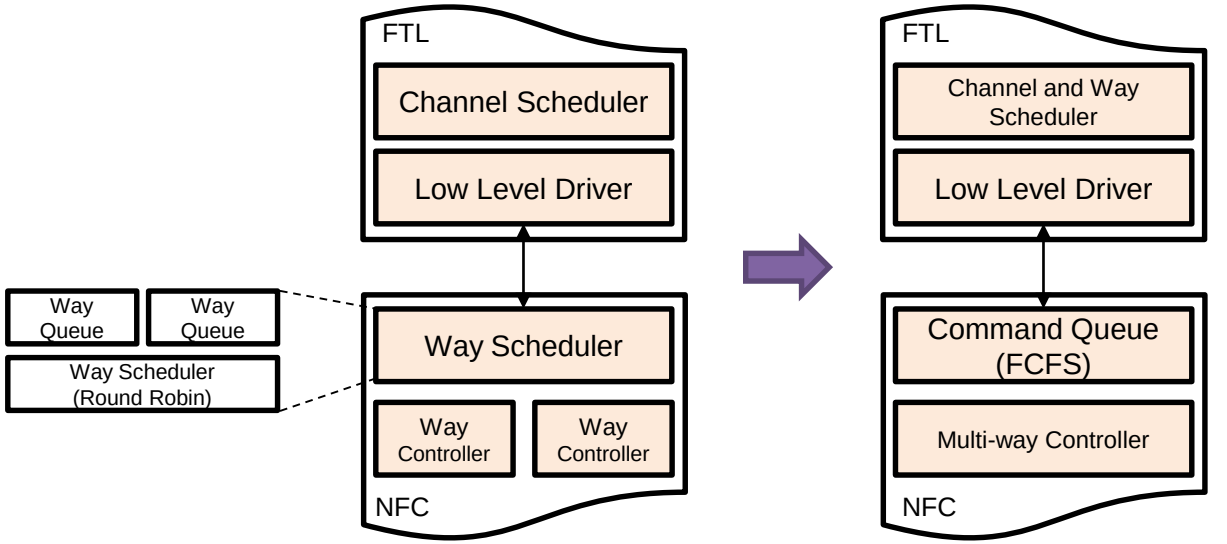
Pipelined NFC Operation

- Data transfers throughout a layer from DRAM to NAND flash or from NAND flash to DRAM are all pipelined
- Page buffer is not required in channel controller



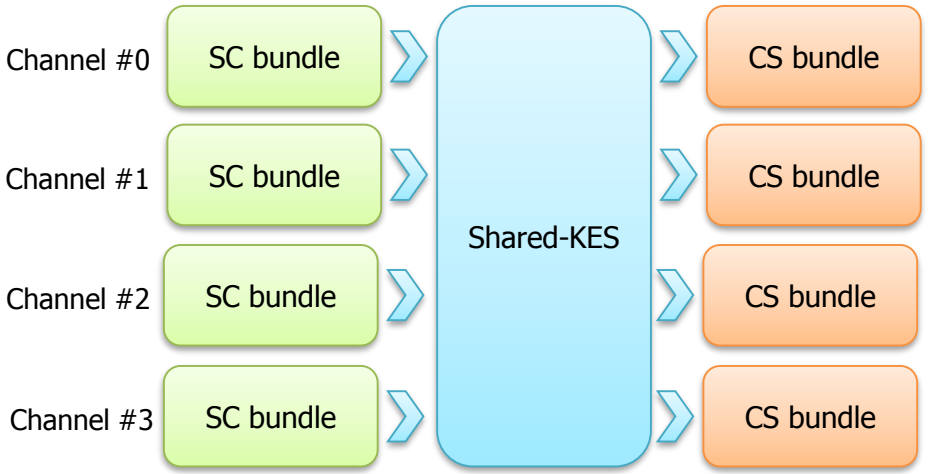
Software-controlled NAND Flash Scheduler

- Hardware-level way scheduler of NFC in Cosmos OpenSSD is removed
- FTL is now responsible for channel and way scheduling
- This enables more flexible scheduling policy



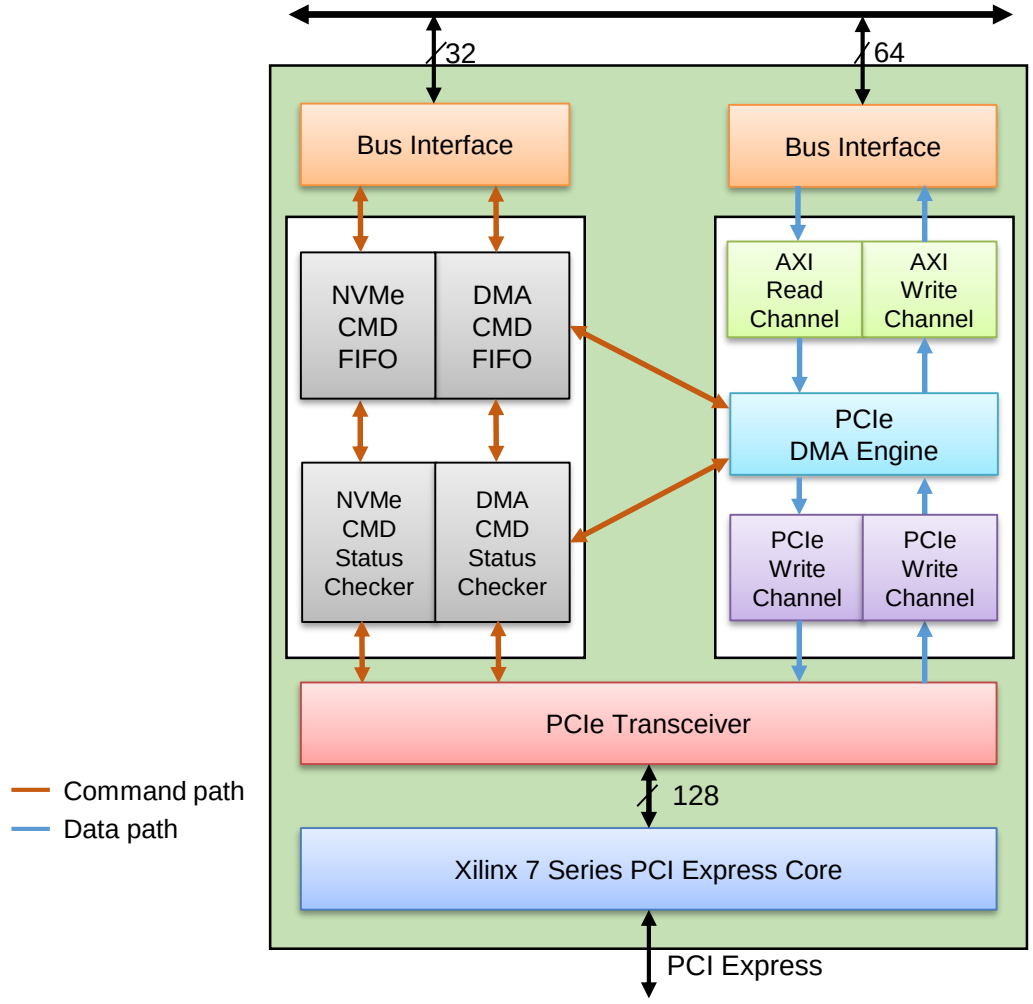
BCH ECC Engine with Shared-KES

- **Key equation solver (KES) used more (≥50 %) of logic cells than syndrome calculator and chien searcher**
- **Shared-KES saves 40 % of logic cells used in a BCH ECC decoder**
- **Short BCH code parallelization is applied for high utilization of hardware resources**



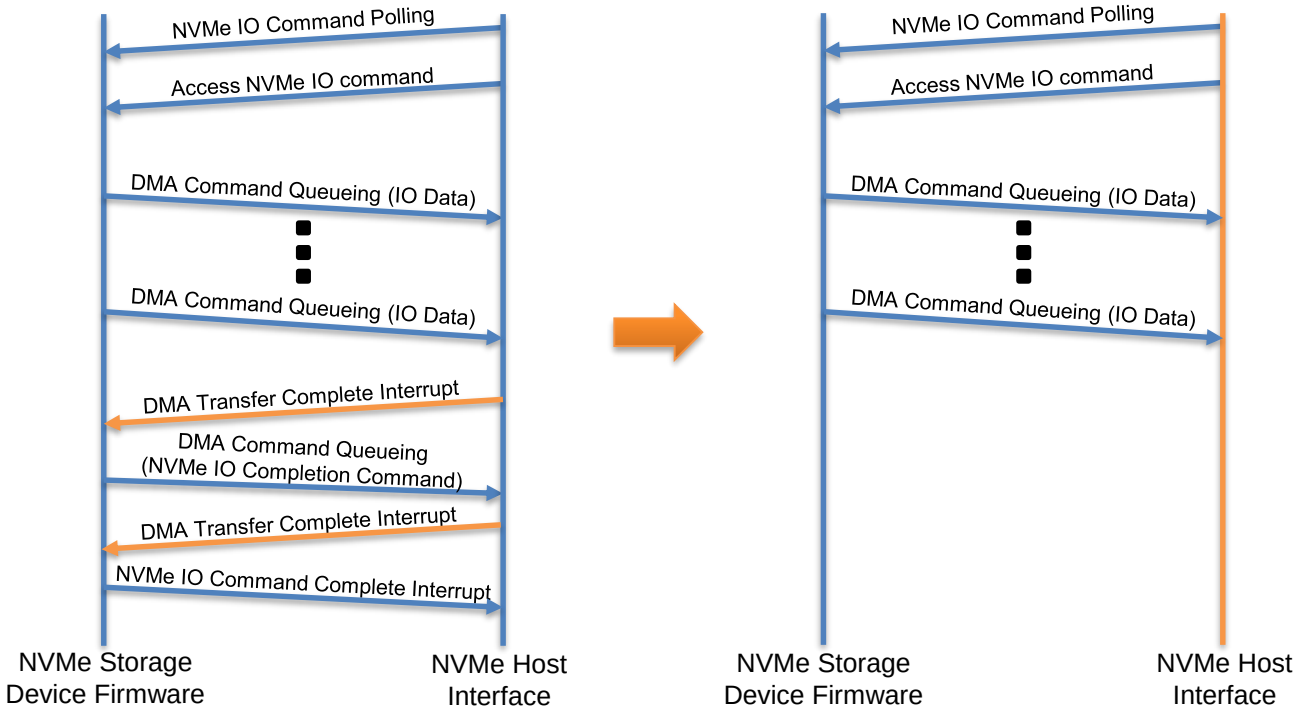
* (256 B x 2) bundle, (13 x 2) bit error correction, (8 bit x 2) parallel level

NVMe Host Interface Overview



Automated NVMe IO Command Completion

- The NVMe host interface completes NVMe IO commands automatically
- The FTL does not need to be involved in the completion process



NVMe Host Interface Specification

NVMe specification 1.1/1.2 compliant

- Up to 8 IO submission/completion queues - 256 entries each
- 512B and 4KB sector size
- Physical region page (PRP) data transfer mechanism
- Native device driver for Windows 8/8.1 and Linux kernel>=3.3
- OpenFabrics Alliance (OFA) NVMe driver for Windows 7 and later

NVMe Interface Performance (DRAM Disk)

Workload	Read	Write
Random 4KB	300K IOPS	300K IOPS
128KB	1.7 GB/s	1.7 GB/s

Firmware FTL Features

LRU data buffer management

- Data transfer between host system and NAND flash memory via data buffer
- Eviction of LRU buffer entry

Pure page-level mapping (16 KB page)

- Static mapping
- Channel/way interleaving

Greedy garbage collection

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

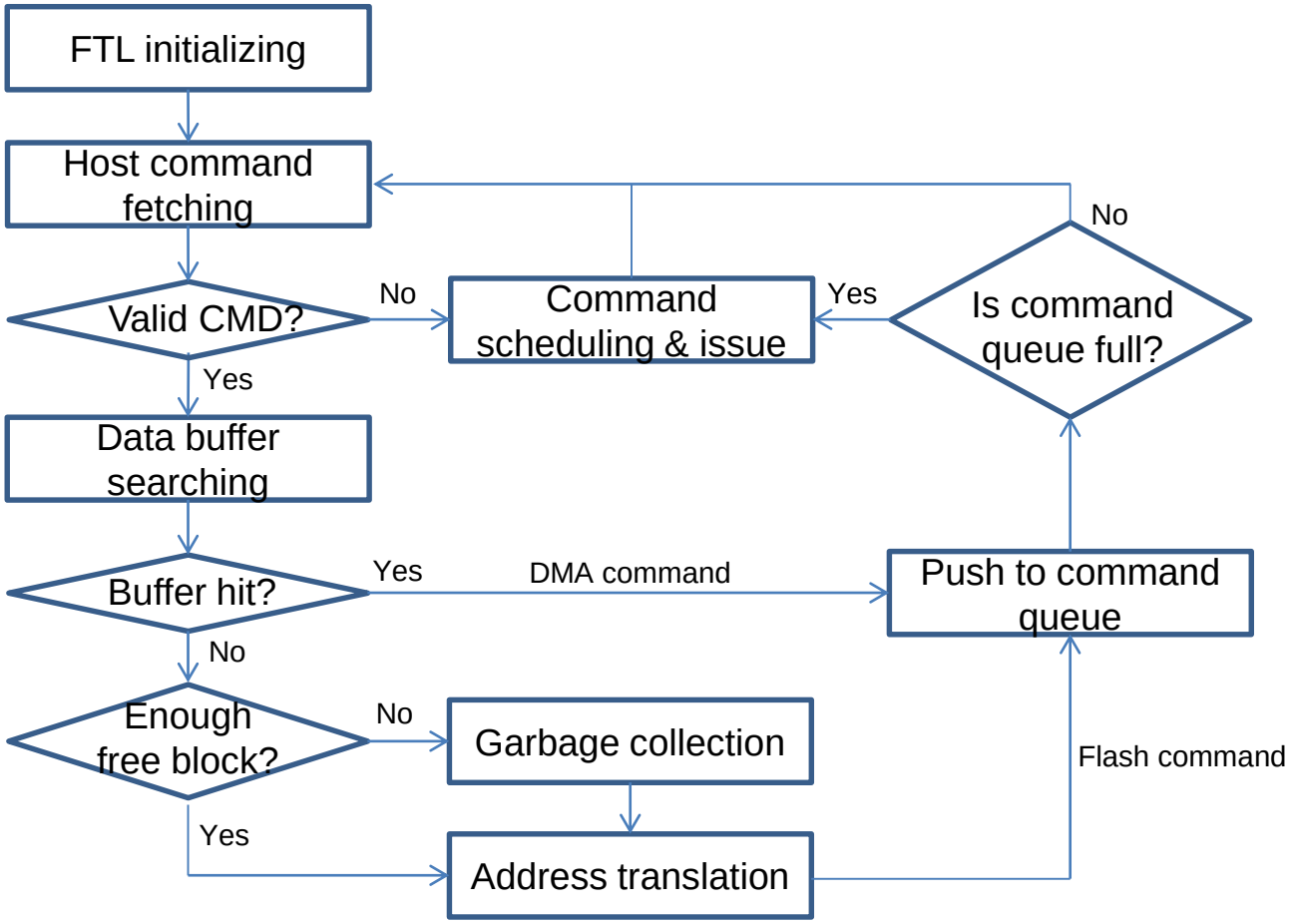
Command Set

- Single plane flash commands
- DMA commands for data transfer between host system and SSD

Priority-based scheduling

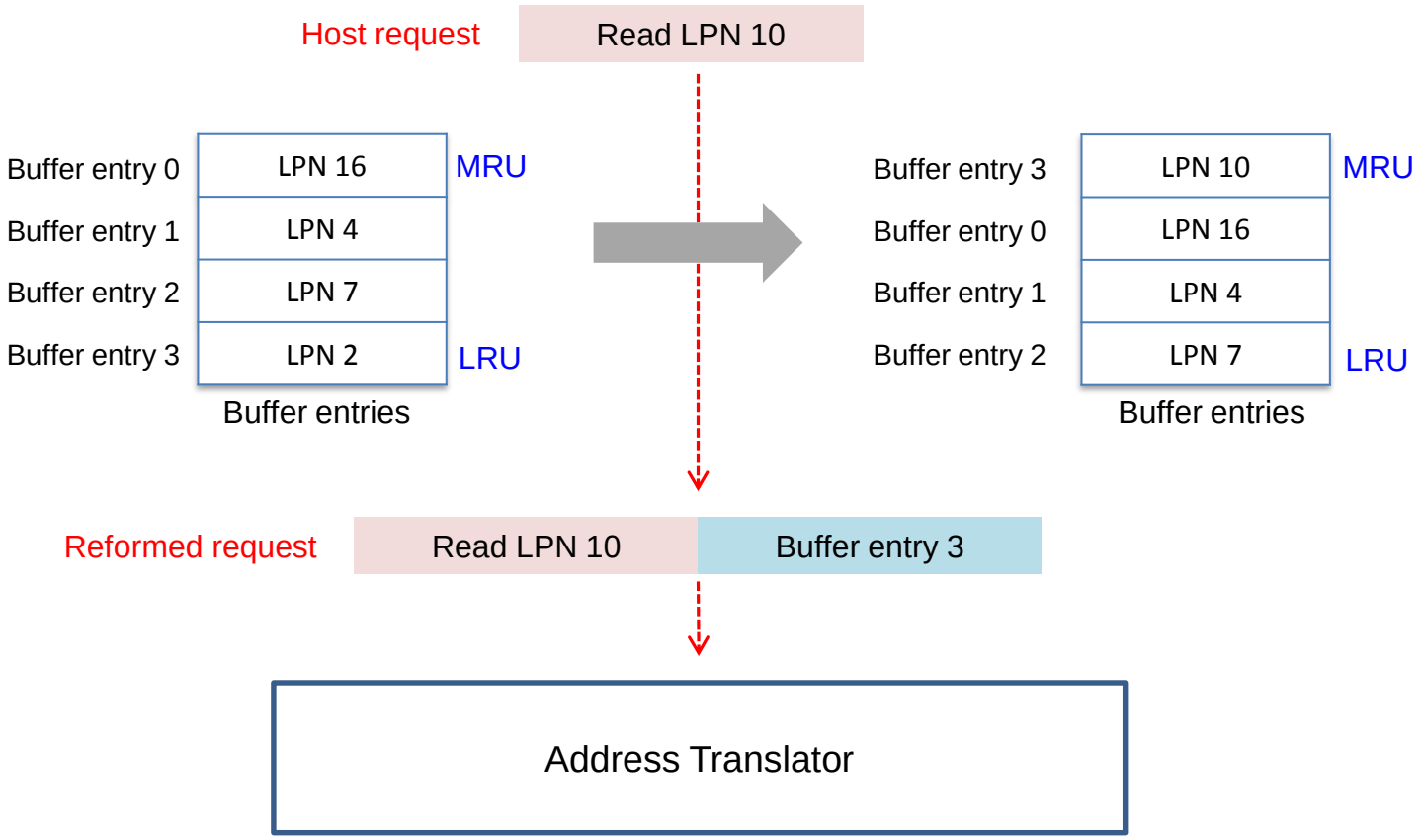
- Predetermined priority between DMA commands and flash commands
- Out of order execution between commands accessing different flash dies

Firmware Overall Sequence



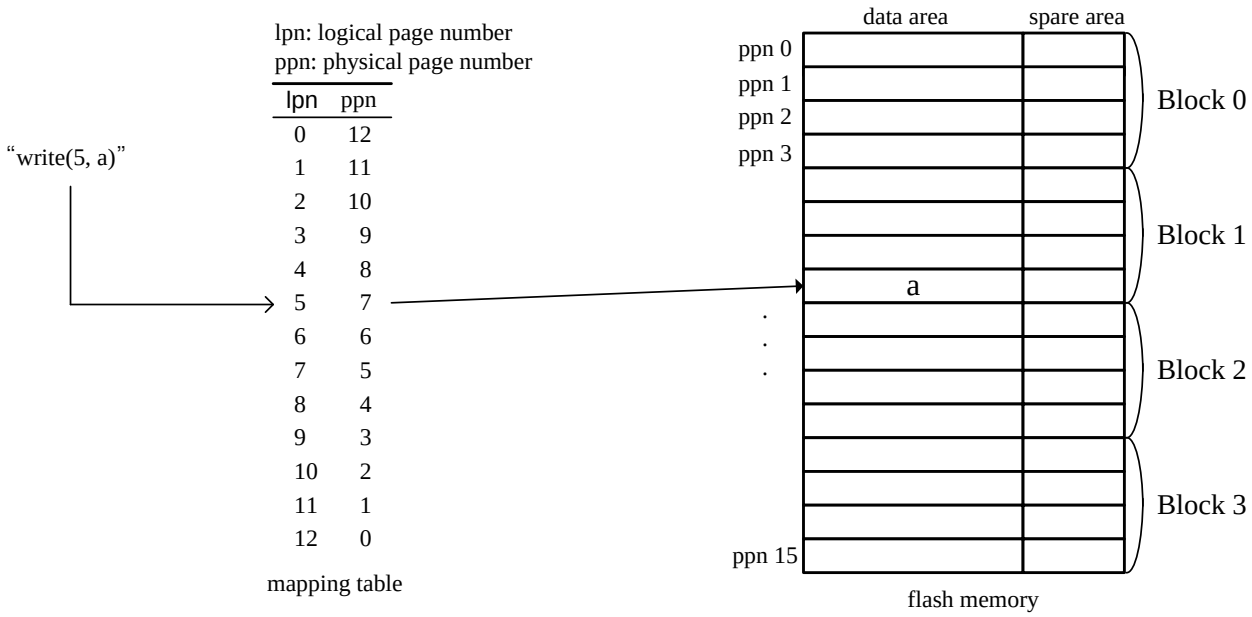
LRU Data Buffer Management

- **Buffer entry eviction**
  - LRU buffer entry is evicted to allocate a buffer entry for a new request



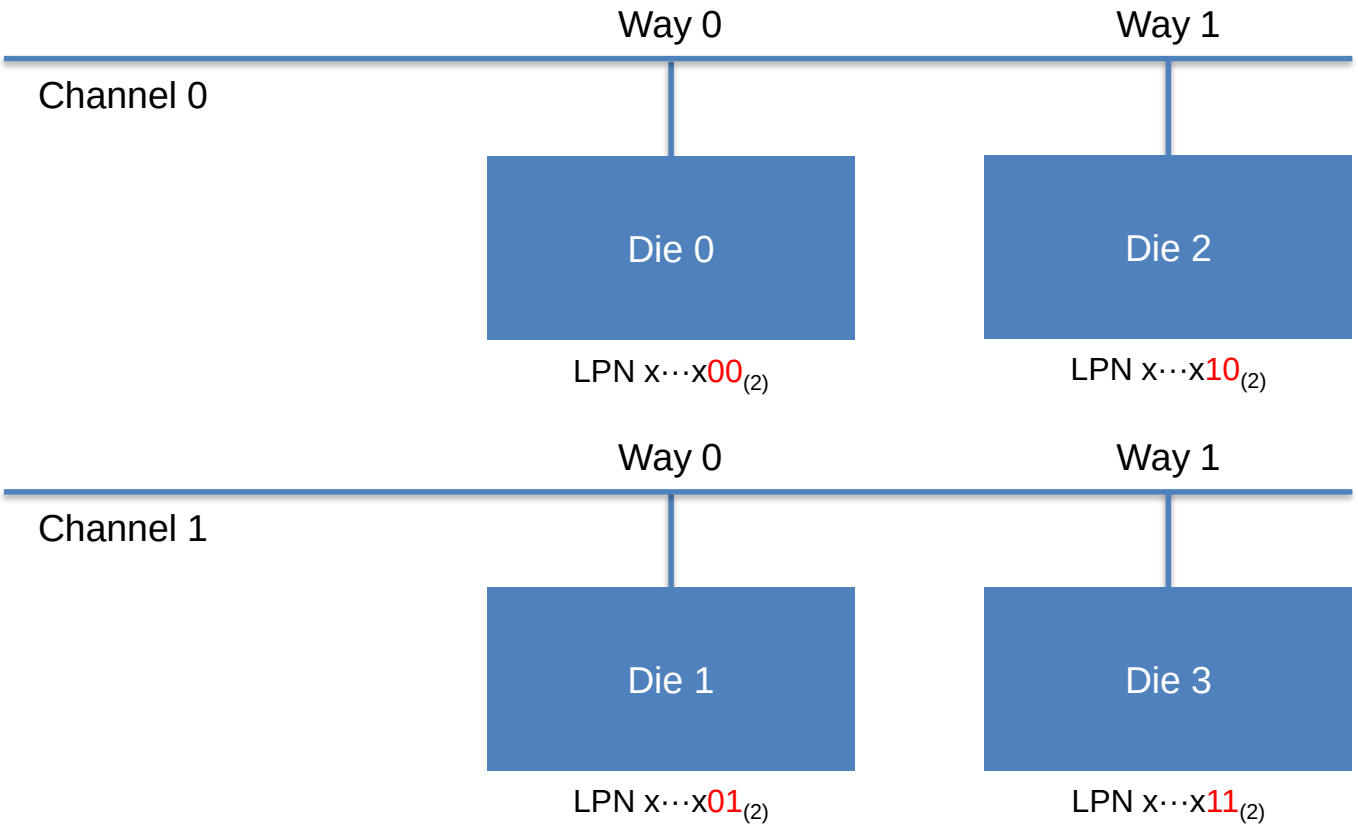
Page-level Mapping

- **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**
  - Simple channel/way interleaving for sequential logical access



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

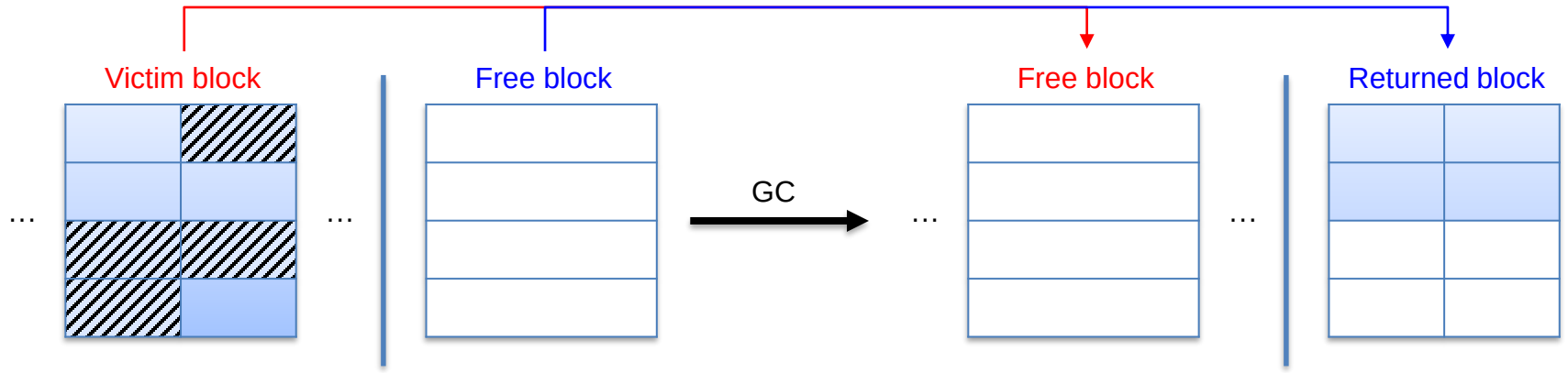
LPN: Logical Page Number

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

Firmware Command Set

Commands for NVMe DMA engine

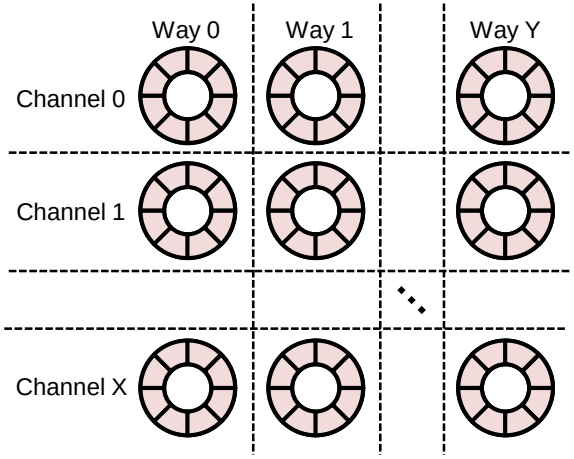
LLSCommand_RxDMA
▶ Transfer data from host system to data buffer
LLSCommand_TxDMA
▶ Transfer data from data buffer to host system

Commands for NAND flash controller

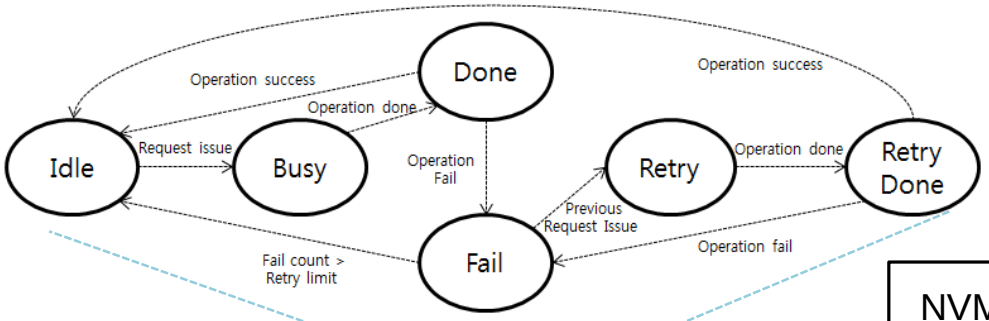
V2FCommand_ReadPageTrigger
▶ Read data of a flash page ▶ Store data to register of the flash die
V2FCommand_ReadPageTransfer
▶ Transfer data from a flash die to data buffer ▶ Inform bit error information to FTL
V2FCommand_ProgramPage
▶ Transfer data from data buffer to a flash die ▶ Program data to a flash page
V2FCommand_BlockErase
▶ Erase a flash block
V2FCommand_StatusCheck
▶ Check a previous command execution result

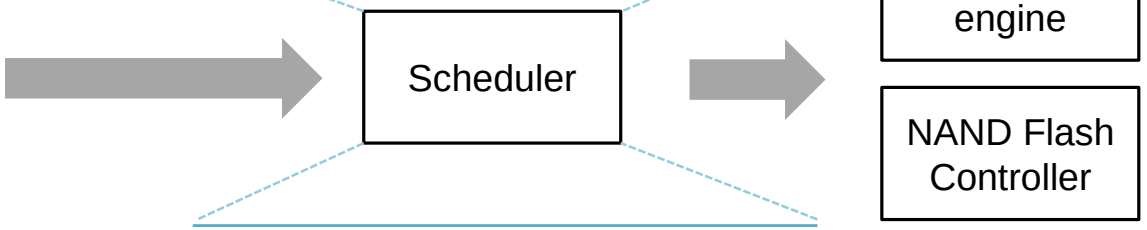
Priority-based Scheduling

- **Waiting commands are issued by scheduler**
  - Scheduler checks the state of flash memory controller and host interface controller
  - Priority of flash commands enhance multi channel, way parallelism



Command queues





Command	Priority
LLSCommand_RxDMA	0
LLSCommand_TxDMA	0
V2FCommand_StatusCheck	1
V2FCommand_ReadPageTrigger	2
V2FCommand_BlockErase	3
V2FCommand_ProgramPage	4
V2FCommand_ReadPageTransfer	5

Known Restrictions (1/3)

- **Firmware**
  - Supports
    - Buffer management (LRU)
    - Static page mapping
    - Garbage collection (On-demand)
  - Not supports
    - Meta flush
    - Wear leveling
  - Notice
    - I / O performance can be degraded when performing garbage collection
    - The number of usable blocks is limited when the MLC NAND array is used in the 8-channel 8-way structure
    - The latest firmware in SLC mode accesses only LSB pages of MLC NAND
    - Accessing to MSB pages may cause data errors not able to be corrected by ECC

MSB Page Data Error Issue

- The bit error rate increases if MSB pages of NAND flash are accessed
- Increased bit errors might not be corrected by BCH error correction engine in the current version of NAND flash controller
- For now, the firmware runs in SLC mode in order to reduce the error rate due to this reason

SLC Mode of Firmware

- **Currently, MLC to SLC mode transition command of NAND flash is not supported**
- **Accessing only LSB pages achieves similar characteristics to real SLC NAND flash**

Paired page address	
LSB pages	MSB pages
00h	02h
01h	04h
⋮	
FDh	FFh

Known Restrictions (2/3)

- **PCIe-NVMe**
  - Supports
    - Up to PCIe Gen2.0 x8 lanes
    - Mendatary NVMe commands
    - PRP data transfer mechanism and out-of-order data transfer in PRP list
    - 1 namespce (can be extended by updating firmware)
    - Up to 8 NVMe IO submission queues and 8 NVMe IO completion queues with 256 depths
    - Up to 256 depths internal NVMe command table
    - MSI interrupt with 8 interrupt vectors
    - x86/x64 Ubuntu 14.04 and Windows 8.1
  - Not supports
    - 4 byte addressing yet (on debugging)
    - Optional NVMe commands (can be supported by updating firmware)
    - SGL data transfer mechanism
    - Power management (can be supported by updating firmware)
    - MSI-X interrupt
    - Virtualization and sharing features

Known Restrictions (3/3)

- **NAND flash controller**
  - Supports
    - Channel can be configured up to 8
    - Maximum bandwidth of NAND flash bus 200 MT
  - Not supports
    - Additional advanced commands are not supported (e.g. multi-plane operation)

Get Started with Cosmos+ OpenSSD

Overall Steps

- **Preparing development environment**
  - Host computer
  - Platform board
  - Development tools
- **Building materials**
  - FPGA bitstream
  - Firmware
- **Operating Cosmos+ OpenSSD**
  - Bitstream and firmware download to the FPGA
  - Host computer boot and SSD recognition check
  - SSD format
  - SSD performance evaluation and analysis

Tested Host PC Mainboard Compositions

Mainboard	BIOS Ver.	Result	Comment
Asrock Z77 Extream 6	P2.40	Working	
ASUS H87-Pro	0806x64	Working	
Gigabyte H97-Gaming 3	F5	Working	
Gigabyte Z97X-UD5H	F8	Working	
	F10c	Not working	4-byte addressing problems in Cosmos+ PCIe DMA engine

Tested Host PC Operating System

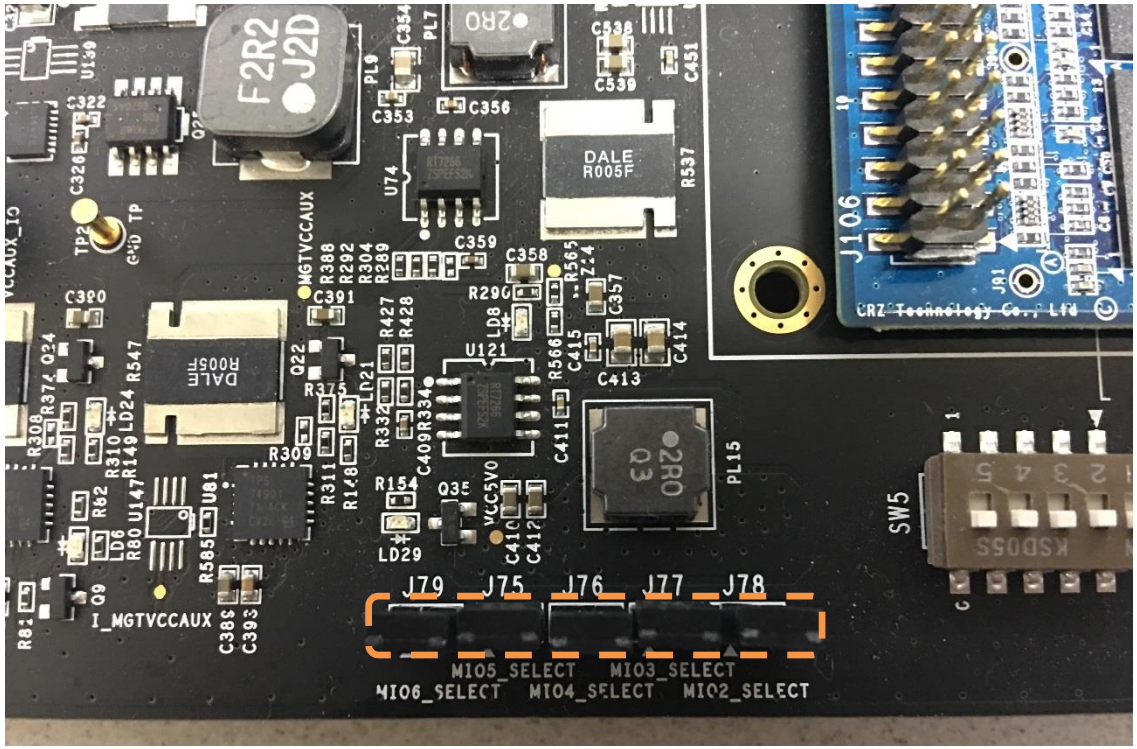
OS	x86/x64	Result	Comment
Windows 7	x64	Working	with OFA driver
Windows 8.1	x64	Working	
Windows 10	x64	Not working	4-byte addressing problems in C osmos+ PCIe DMA engine
Ubuntu 14.04 LTS or above	x64	Working	Kernel version 3.13 or above

Preparing the Platform Board

- Check jumper pins of the platform board
- Insert NAND flash module(s)
- Connect the external PCIe cable
- Connect the USB cable for jtag
- Connect the USB cable for UART
- Connect the power cable

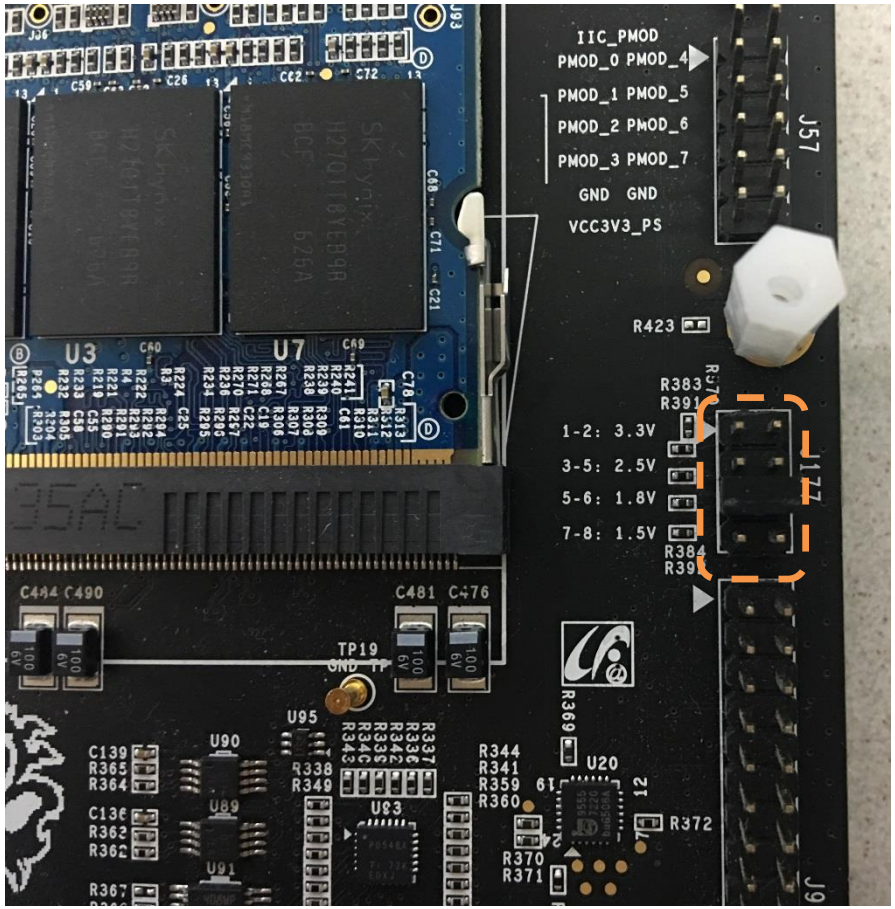
Check Platform Board Jumper Pins (1 / 5)

- **Make sure that jumper pins on board are set as default below**



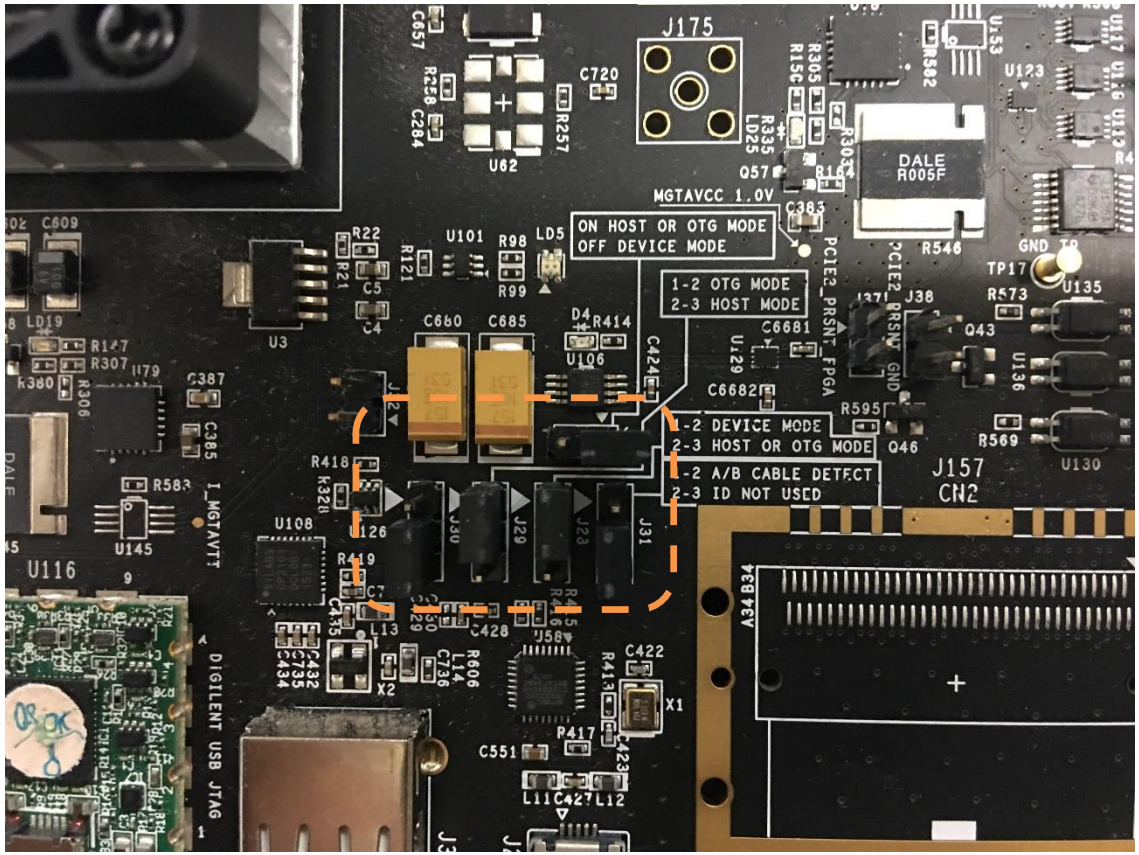
Check Platform Board Jumper Pins (2 / 5)

- **Make sure that jumper pins on board are set as default below**



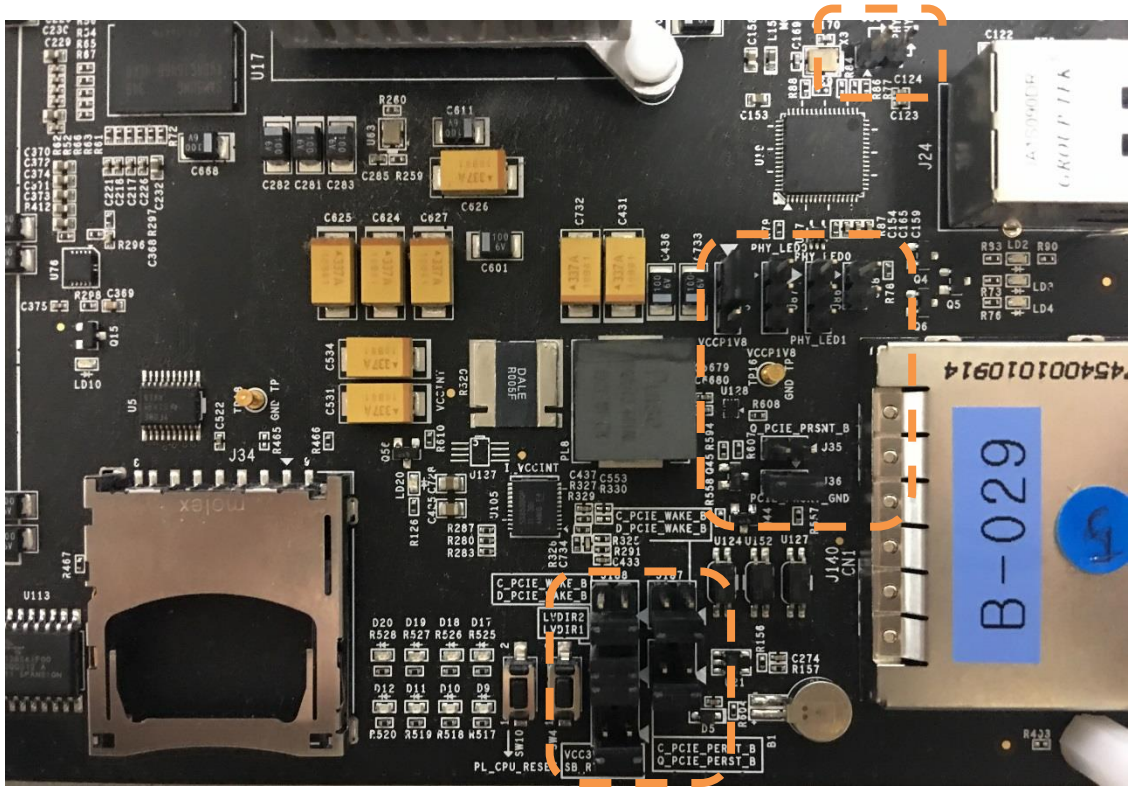
Check Platform Board Jumper Pins (3 / 5)

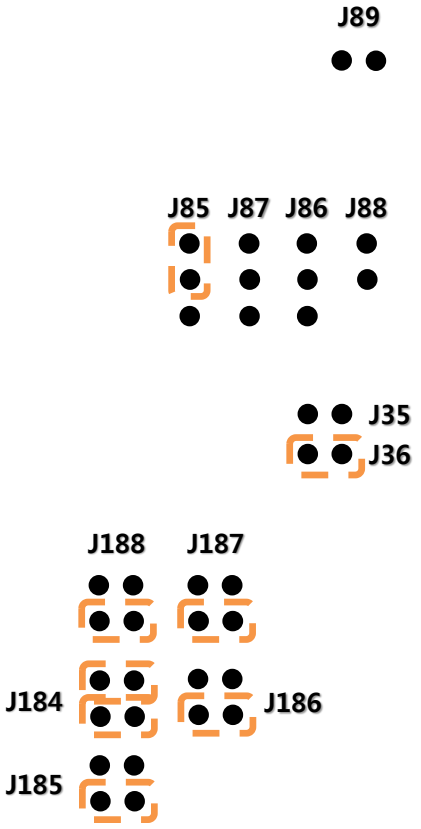
- **Make sure that jumper pins on board are set as default below**



Check Platform Board Jumper Pins (4 / 5)

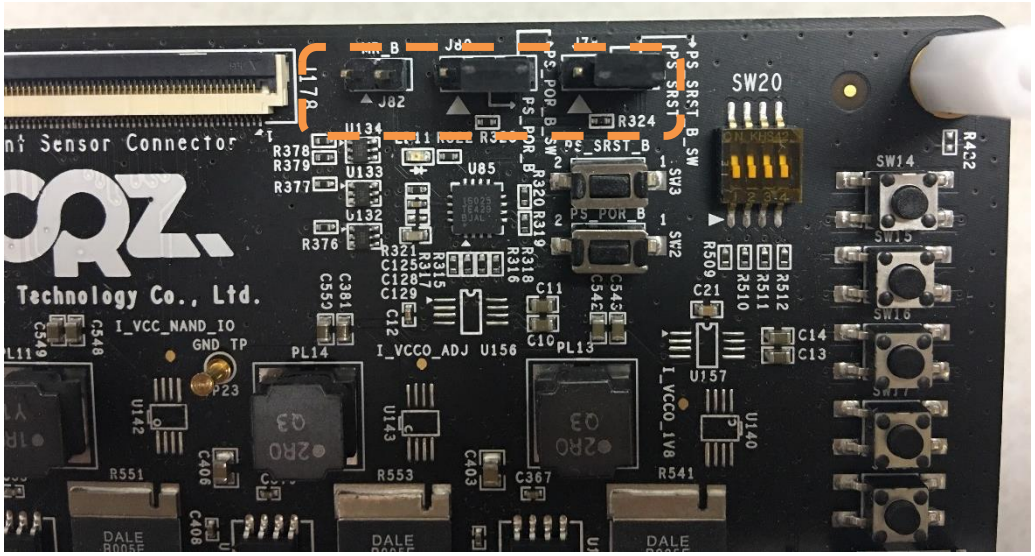
- **Make sure that jumper pins on board are set as default below**





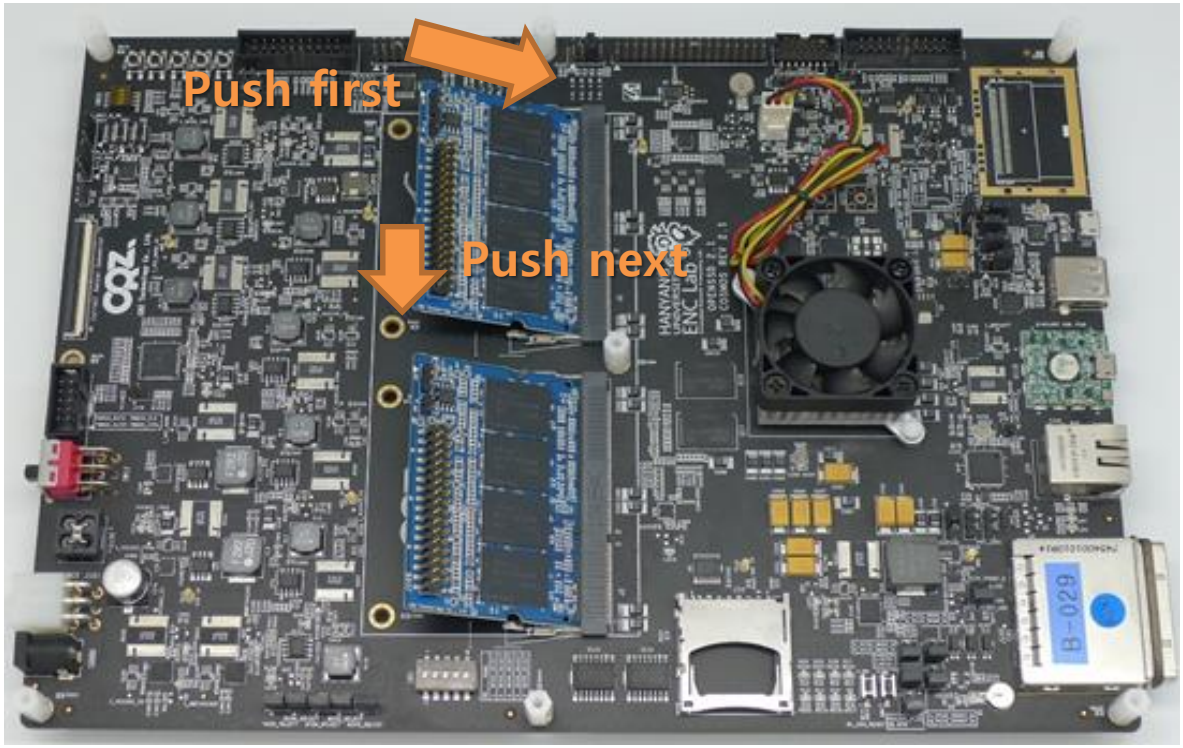
Check Platform Board Jumper Pins (5 / 5)

- **Make sure that jumper pins on board are set as default below**



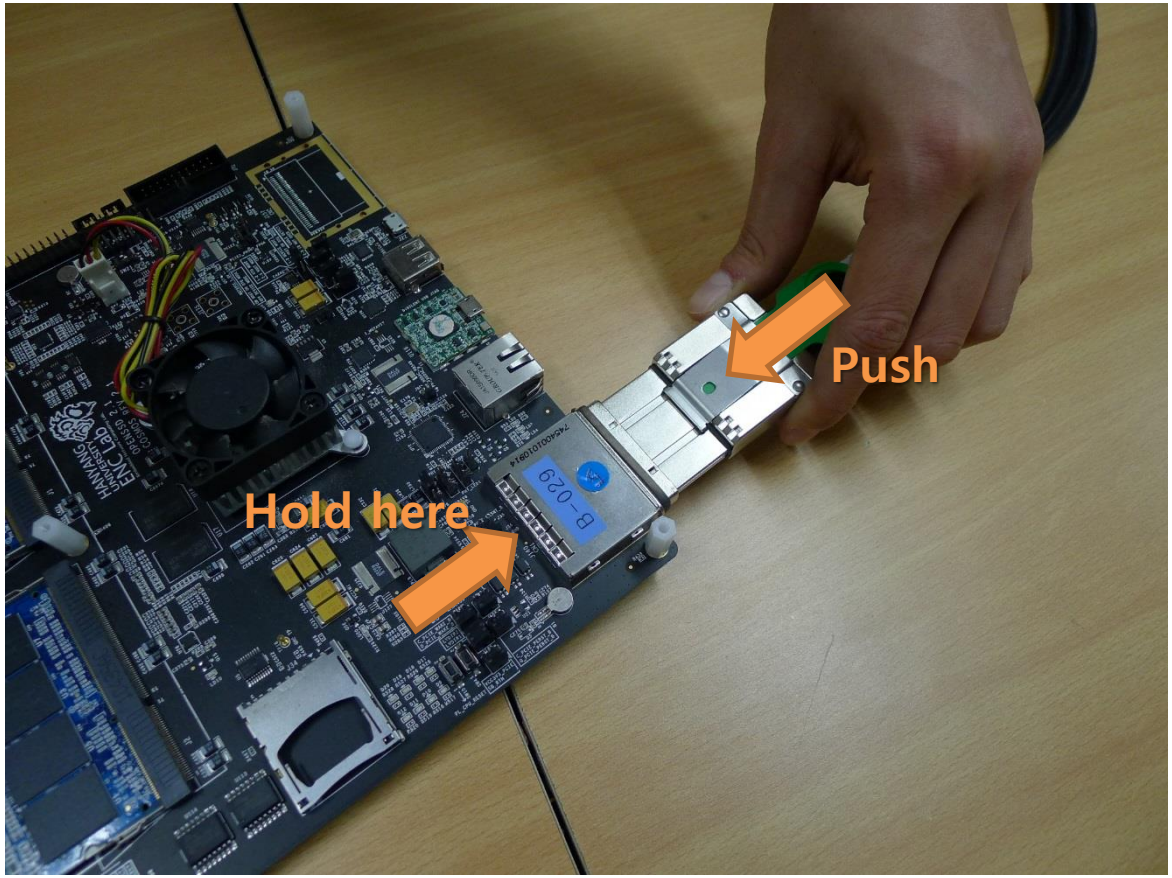
Insert NAND Flash Modules

- **A single NAND flash module can support up to 4-channel configuration**
  - For prebuild 3.0.0, two NAND flash modules are required
  - For predefined project 1.0.0, one NAND flash module is required



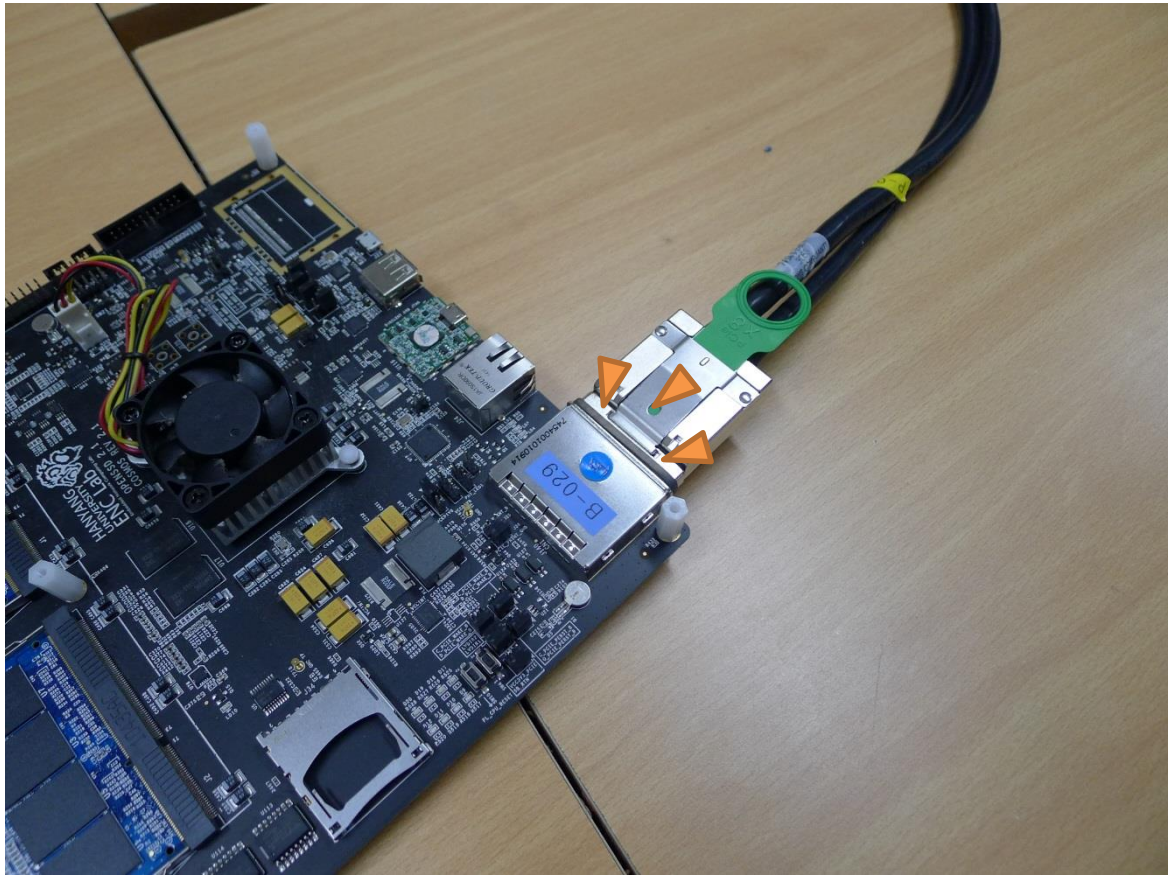
Connect External PCIe Cable (1 / 2)

- **Hold external PCIe connector and push the cable in it**



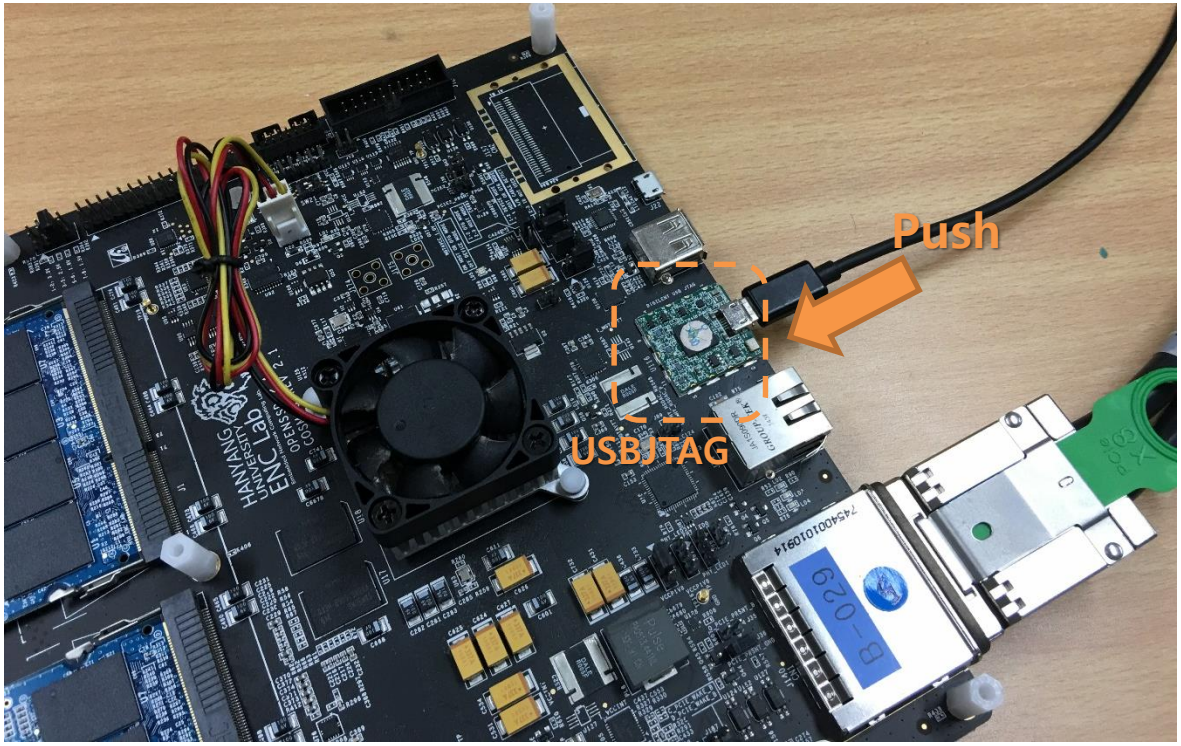
Connect External PCIe Cable (2 / 2)

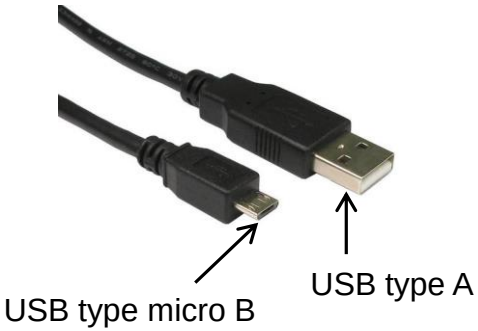
- **Make sure that the cable is fixed tightly**



Connect USBJTAG Cable

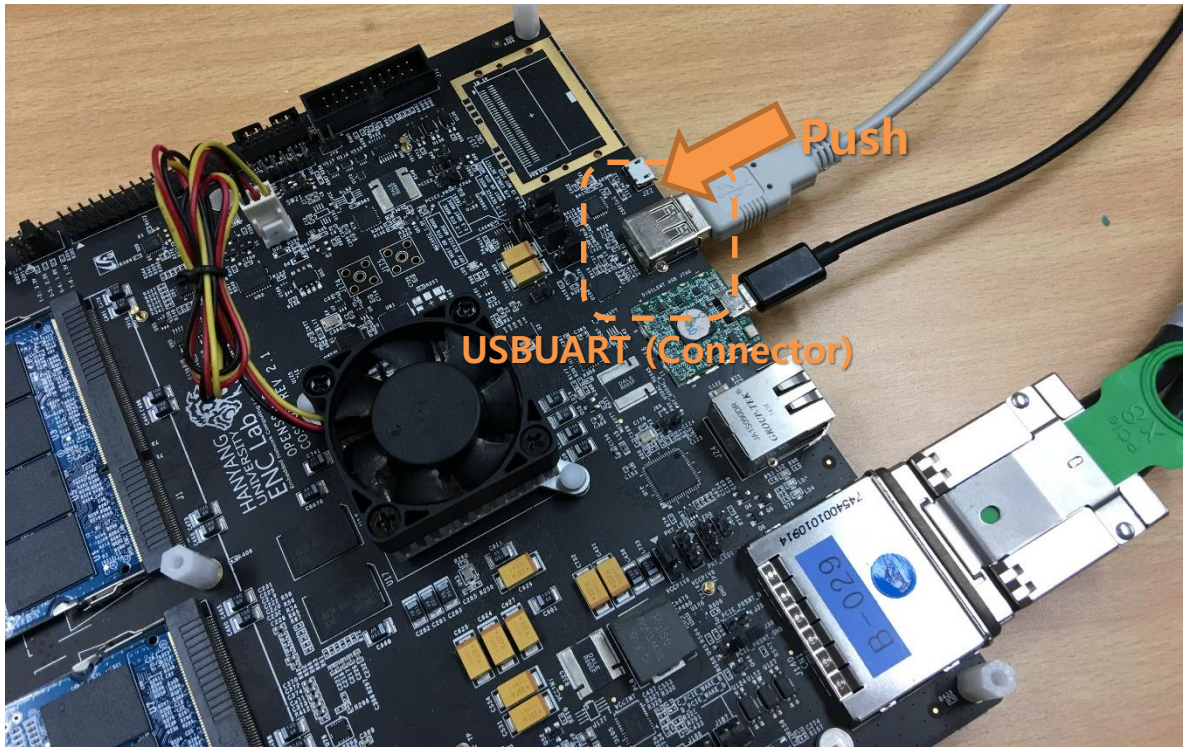
- **USBJTAG requires a micro-USB type B (male) to USB type A (male) cable**

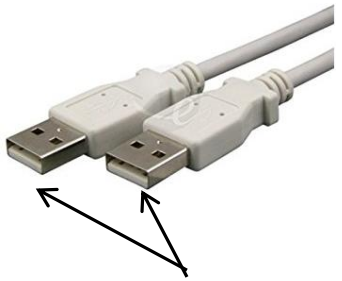




Connect USBUART Serial Communication Cable

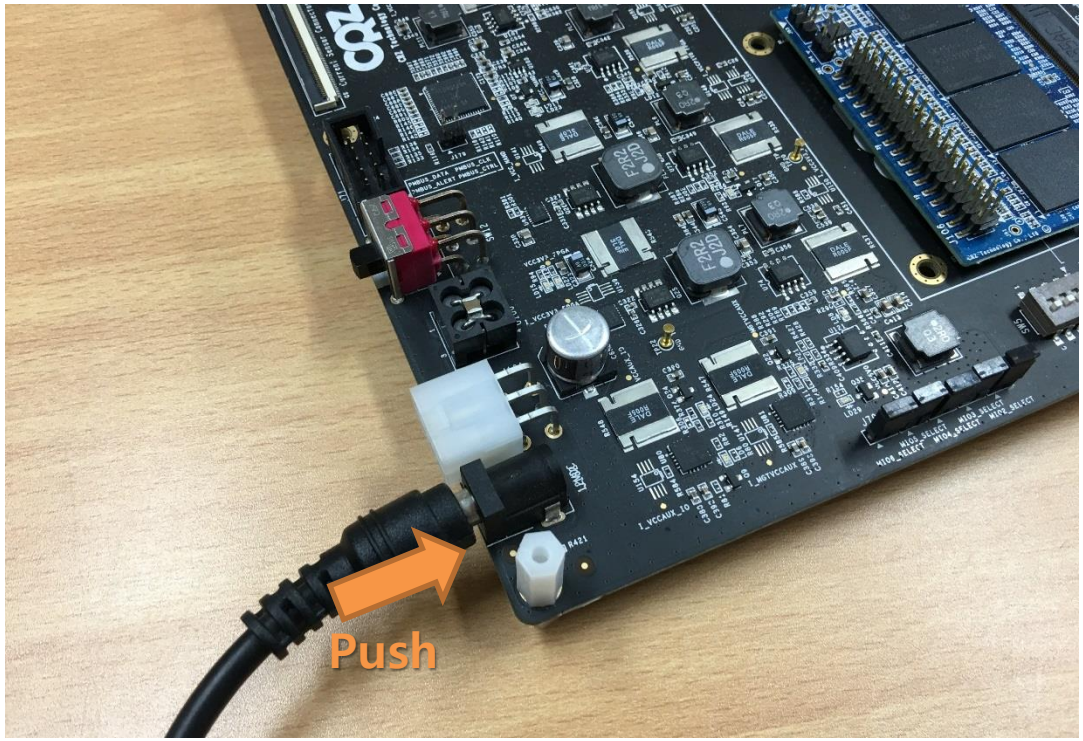
- **USBUART requires a USB type A (male) to USB type A (male) cable**





Connect Power Cable

- **Connect the power cable to the 5.5 mm power connector**

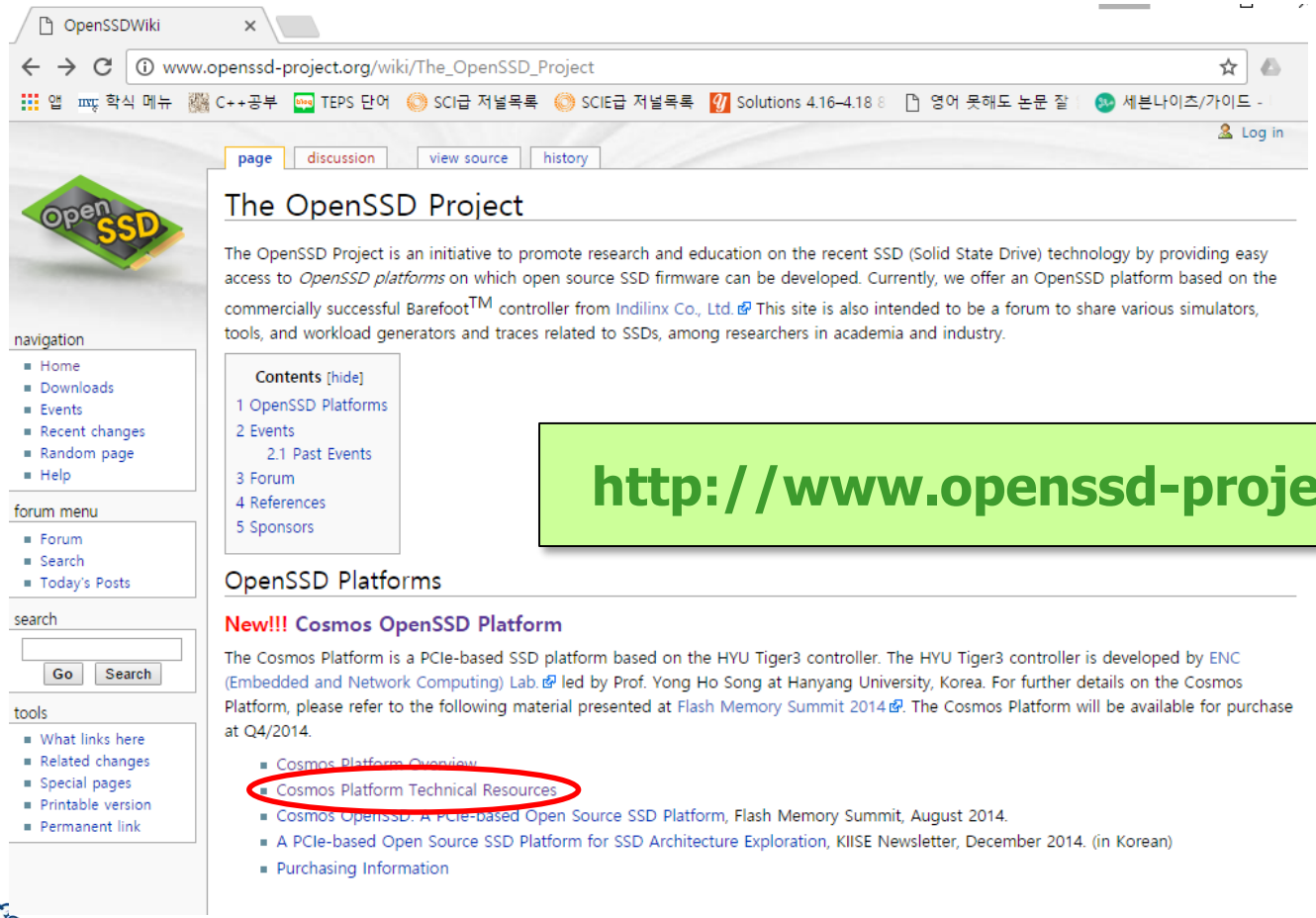


Preparing Software for Development PC

- **Download materials**
  - Prebuilt FPGA bitstream
  - Pre-defined Vivado project for manual FPGA bitstream generation
  - Firmware source code
- **Install Xilinx Vivado Design Suite: System Edition 2014.4**
  - Xilinx Vivado 2014.4
  - Xilinx SDK 2014.4

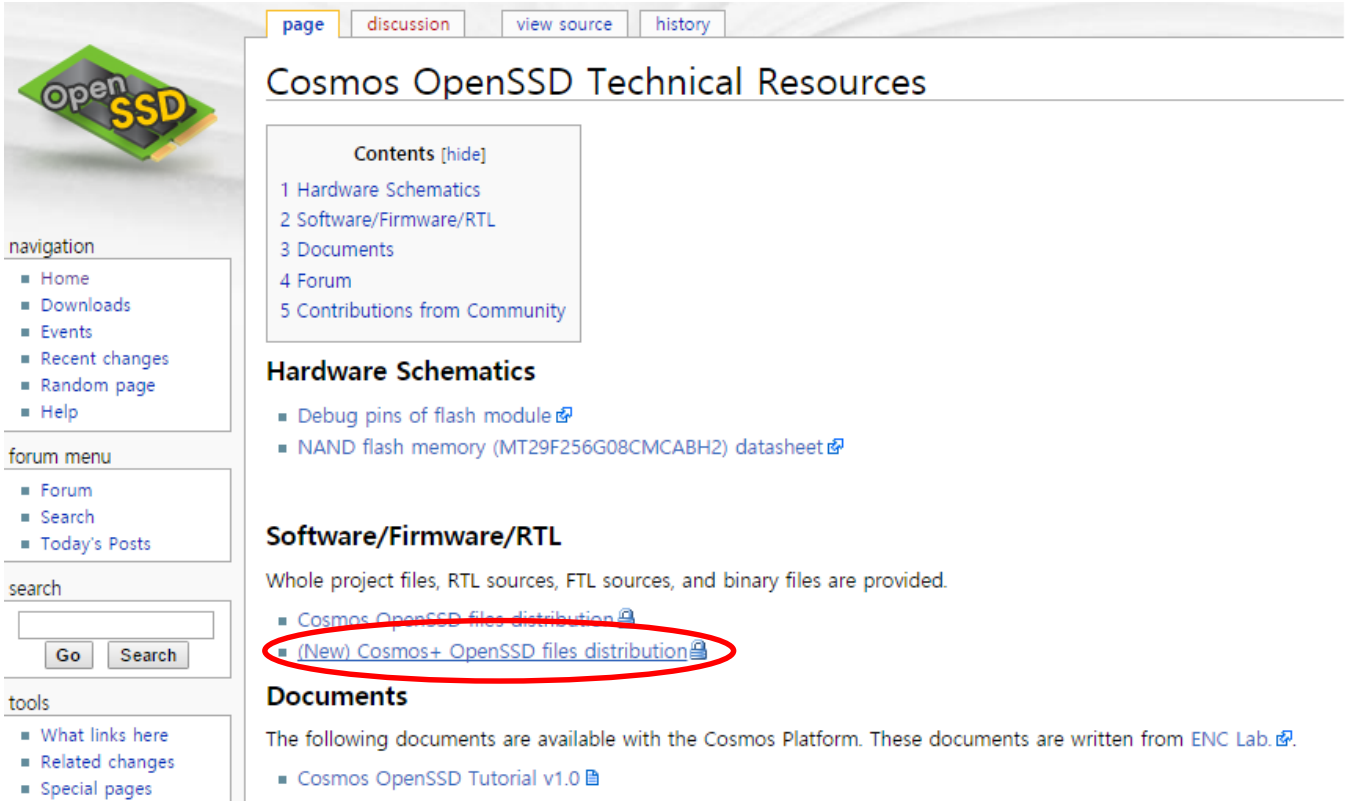
Download Materials (1 / 3)

- **Go to the OpenSSD project wiki, and click “Cosmos Platform Technical Resources”**



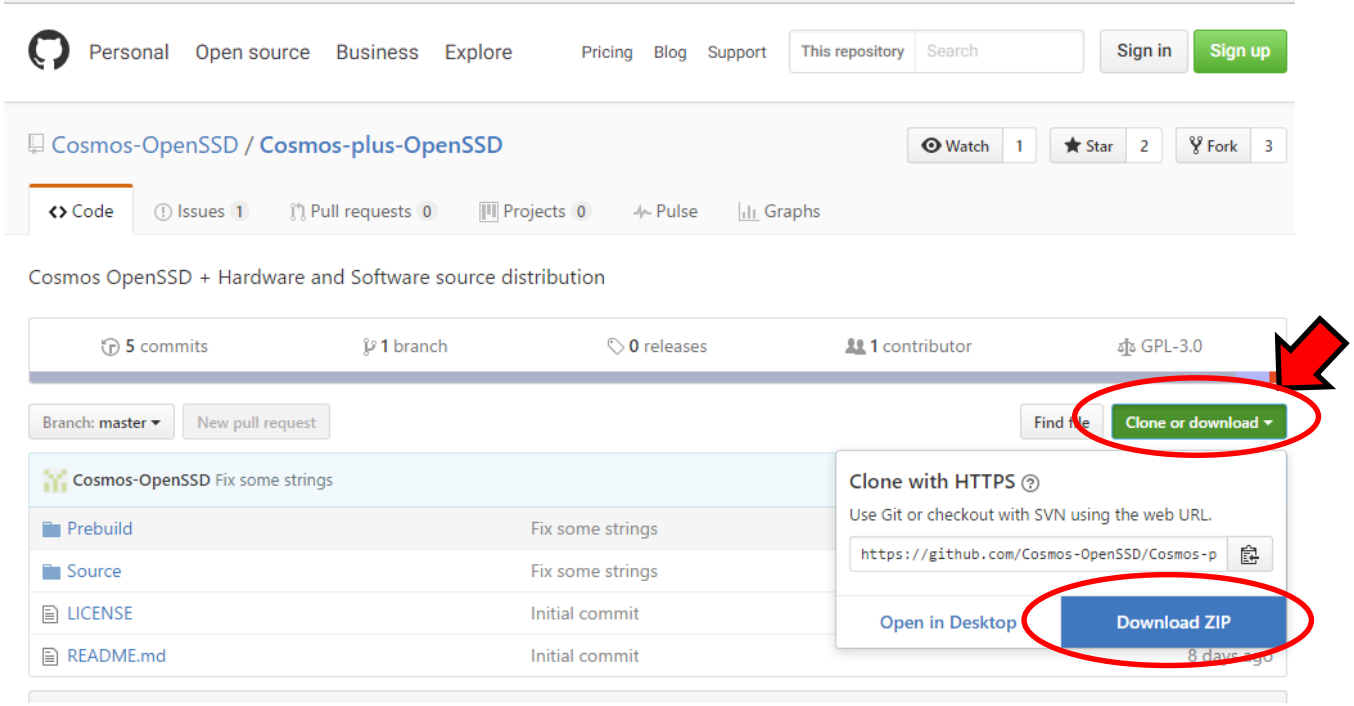
Download Materials (2 / 3)

- **Click “(New) Cosmos+ OpenSSD files distribution”**



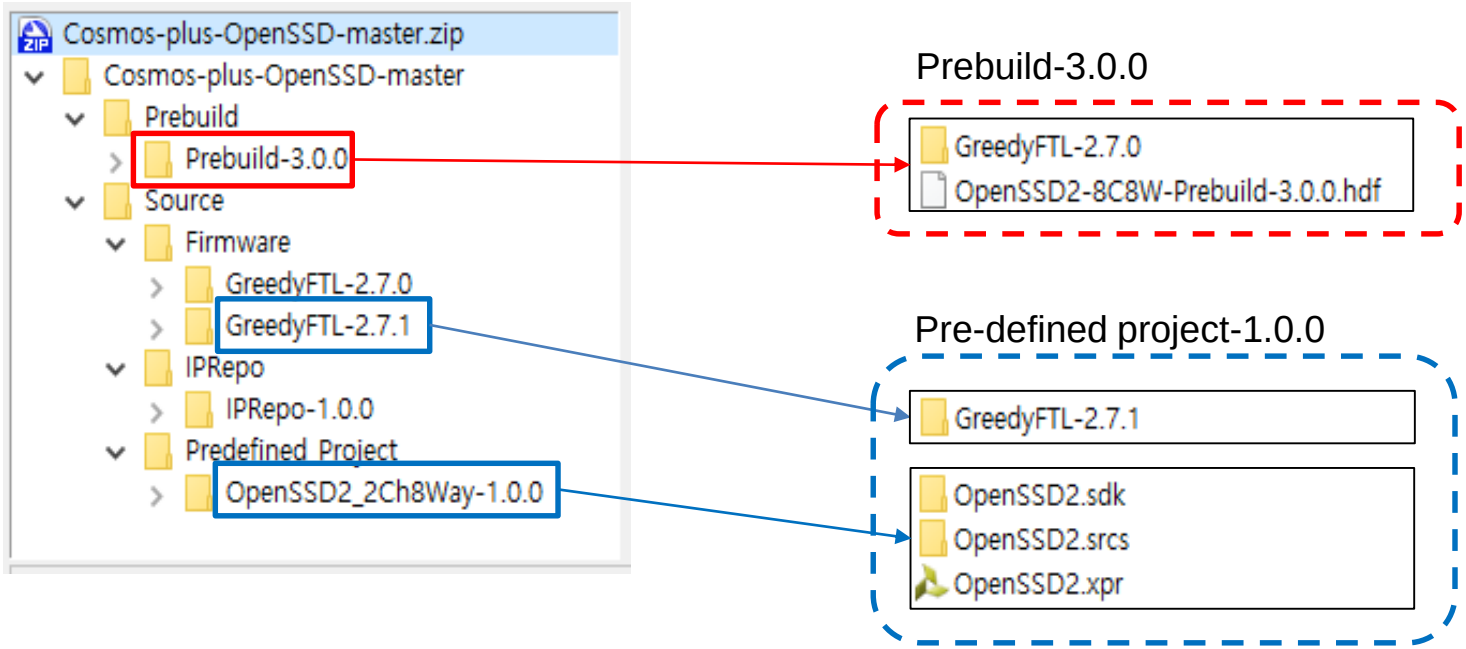
Download Materials (3 / 3)

- **Click “Clone or download” -> “Download ZIP”**



Directory Tree of Downloaded Materials

- **Materials include a prebuilt bitstream, a pre-defined project, and a firmware source code**



Type of Bitstream and Firmware

Bitstream Type	Ver.	Channel	Way	Bits / cell	Capacity
Prebuild	3.0.0	8	8	SLC / MLC	1 TB / 2 TB
Predefined	1.0.0	2	8	SLC / MLC	256 GB / 512 GB

Firmware Type	Ver.	Channel	Way	Bits / cell	Capacity
GreedyFTL	2.5.0	8	8	SLC	1 TB
	2.6.0				
	2.7.0				
GreedyFTL	2.7.1	2	8	SLC	256 GB

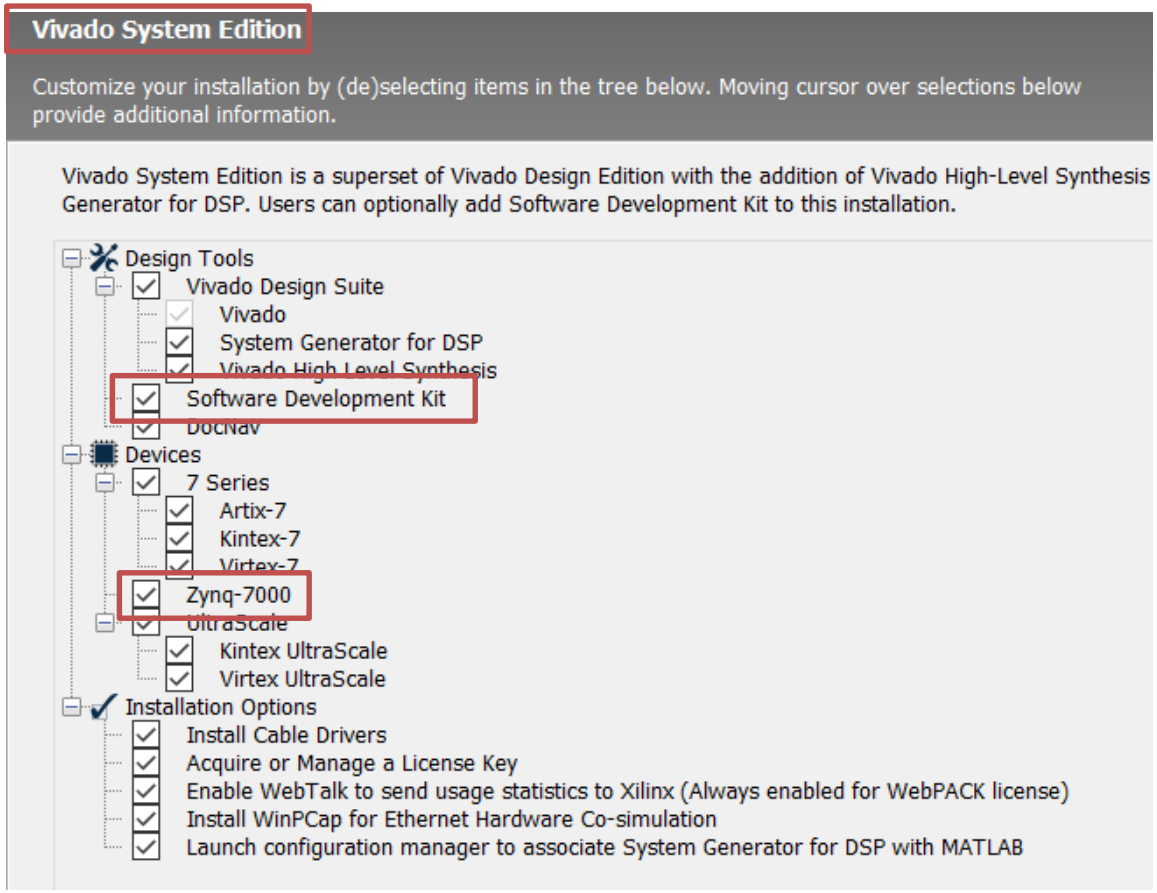
Remarks on the Type of Bitstream

- **Prebuild type**
  - A prebuilt bitstream is included, so you can skip bitstream generation steps
  - Prebuild type is distributed as a hardware description file (.hdf) which consists of a FPGA bitstream, bitstream information, and an initialization code for CPU in Zynq FPGA

- **Pre-defined type**
  - bitstream is not included, so you should follow bitstream generation steps
  - Pre-defined type is distributed as a vivado project file with register transfer level (RTL) source codes of intellectual properties (IPs) such as NVMe controller

Install Xilinx Vivado Design Suite

- **Make sure that Vivado is system edition and that “Software Development Kit” and “Zynq-7000” are checked**

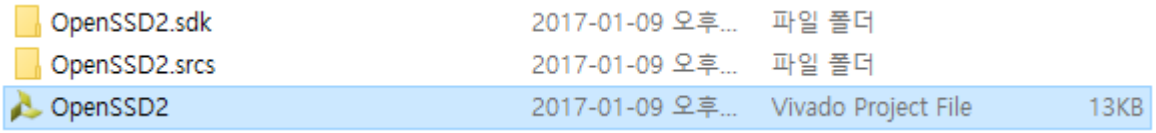


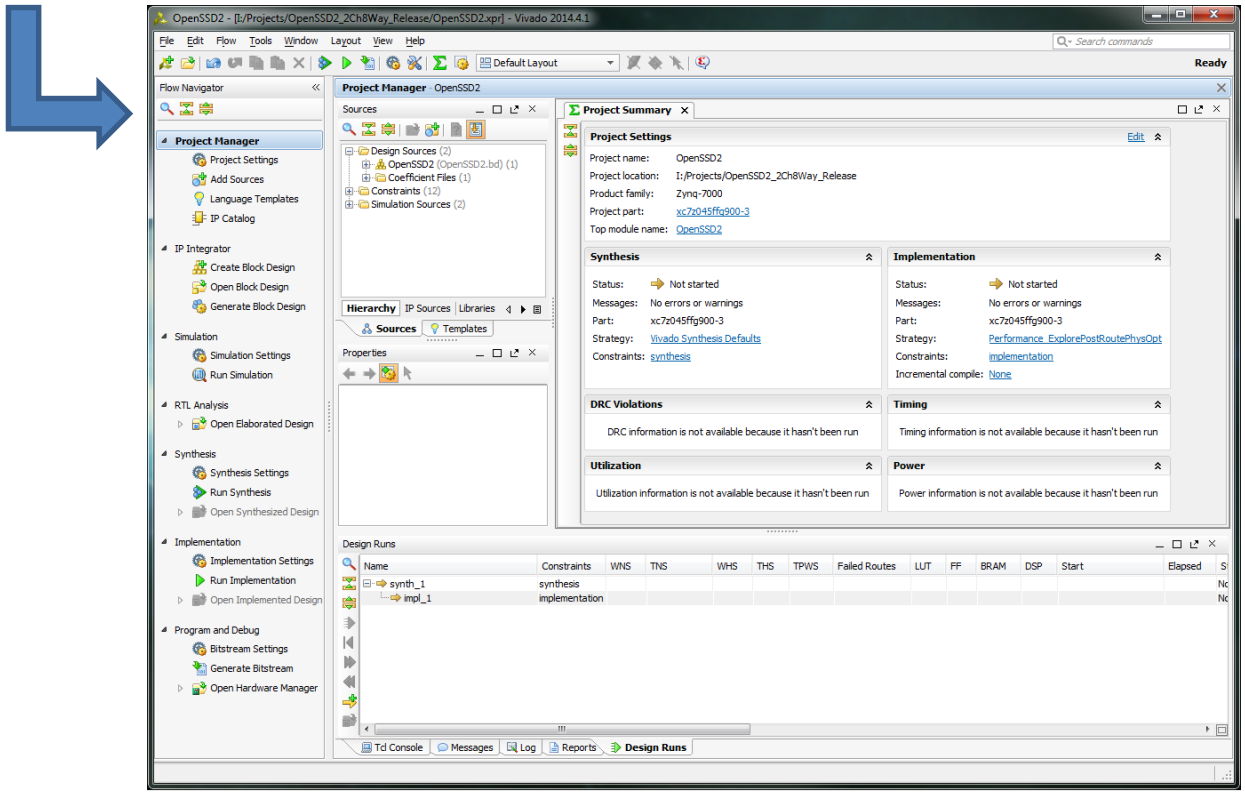
Generating FPGA Bitstream for Pre-defined Project

1. **Run synthesis**
2. **Run implementation**
3. **Generate bitstream**
4. **Export hardware**

Launch Xilinx Vivado

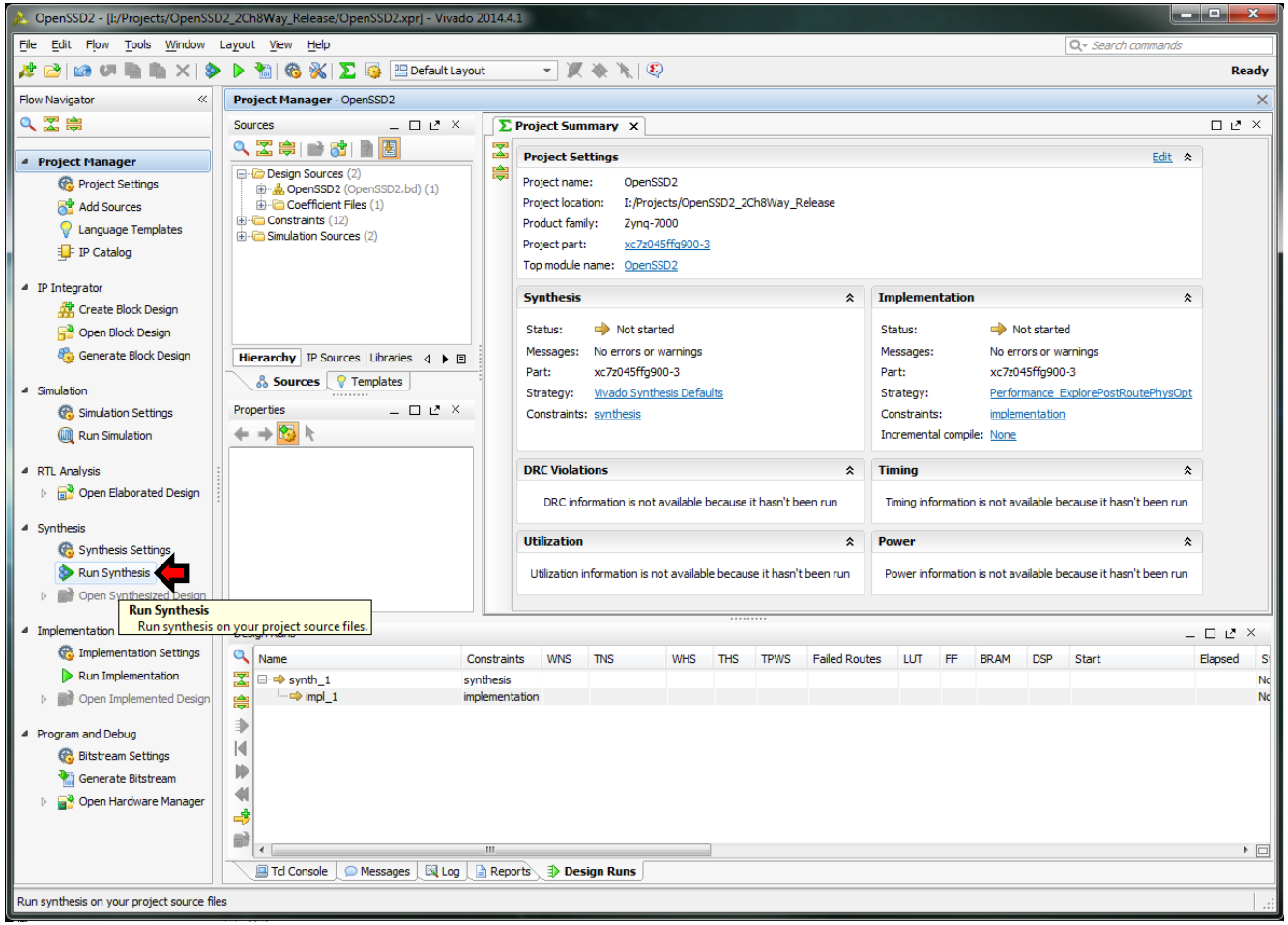
- **Open the predefined project included in “OpenSSD2_2Ch8Way-1.0.0”**





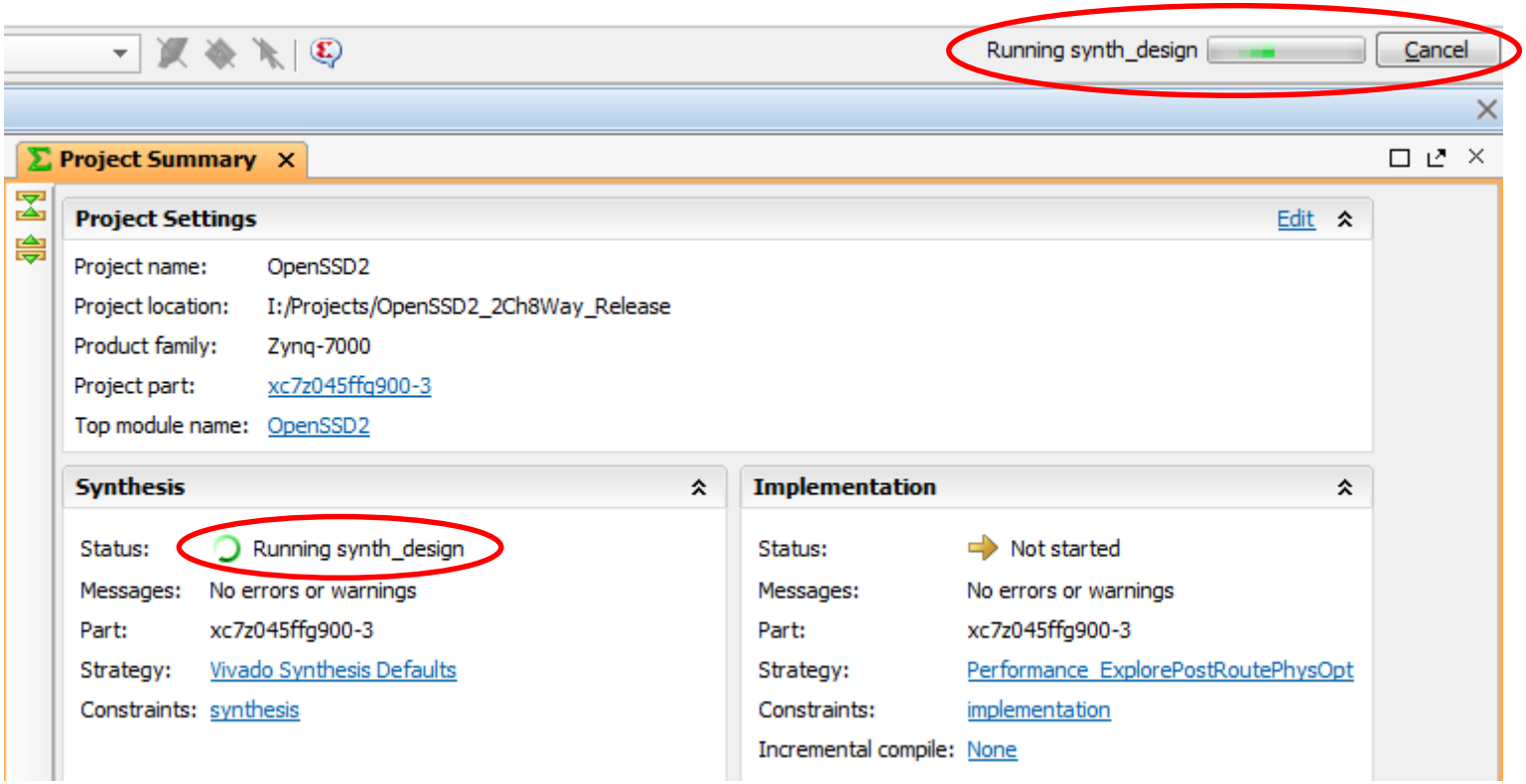
Run Synthesis (1 / 2)

- **Click “Run Synthesis”**



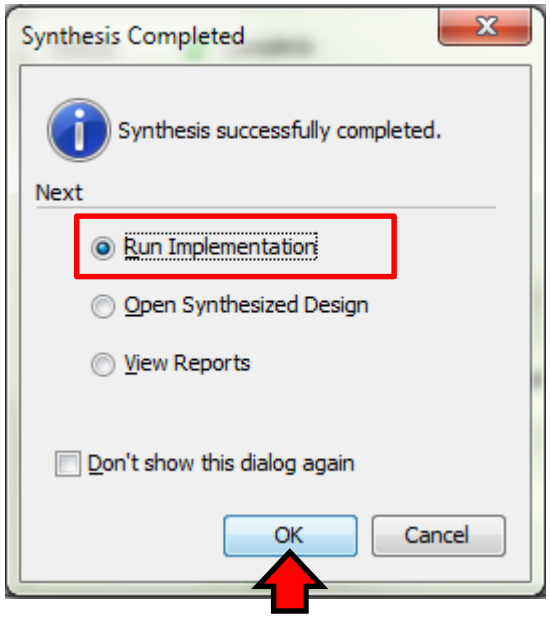
Run Synthesis (2 / 2)

- **Synthesis is running…**



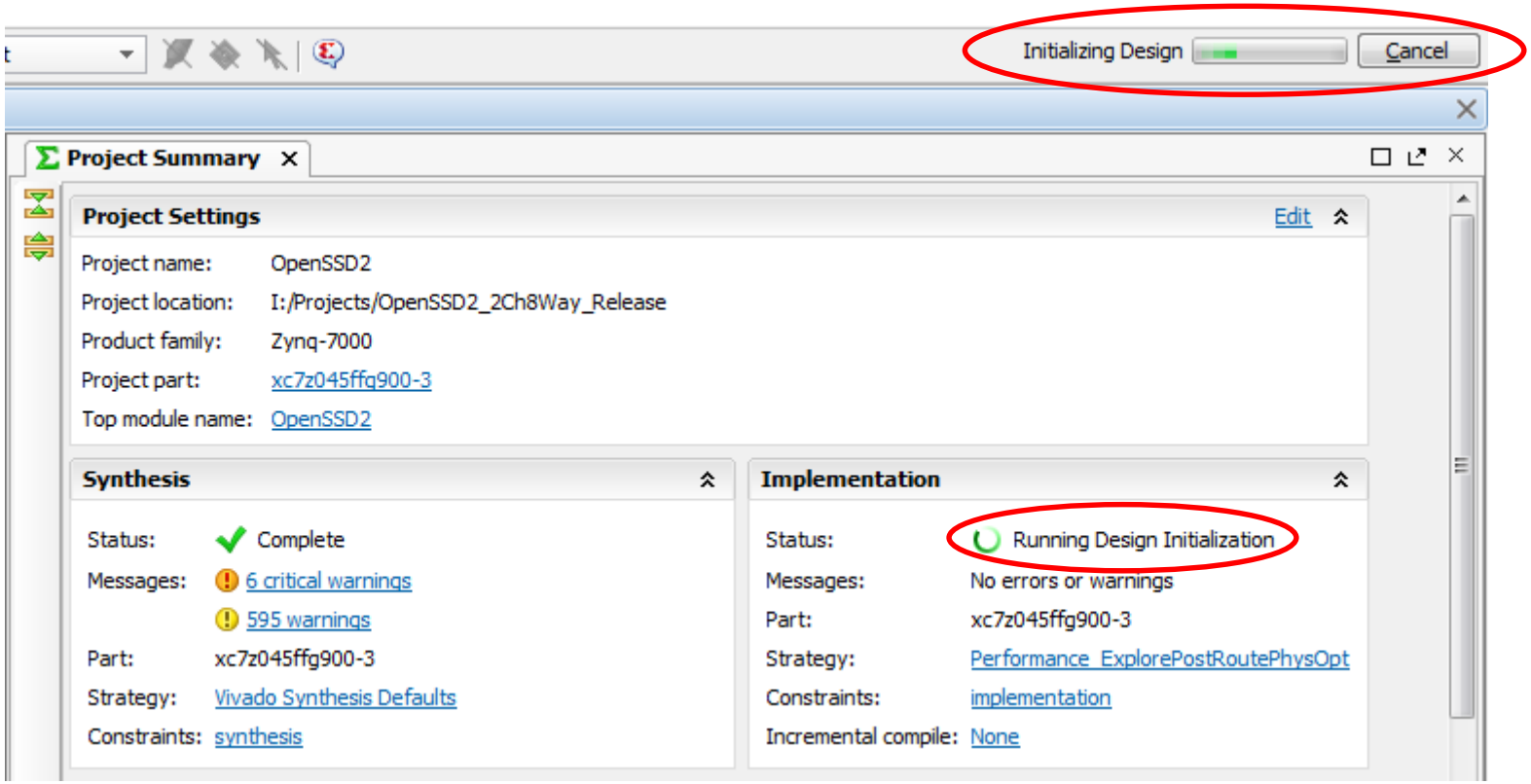
Synthesis Complete

- **Select “Run Implementation” and click OK**
  - If you want to see the synthesized results, choose “Open Synthesized Design” or “View Reports”



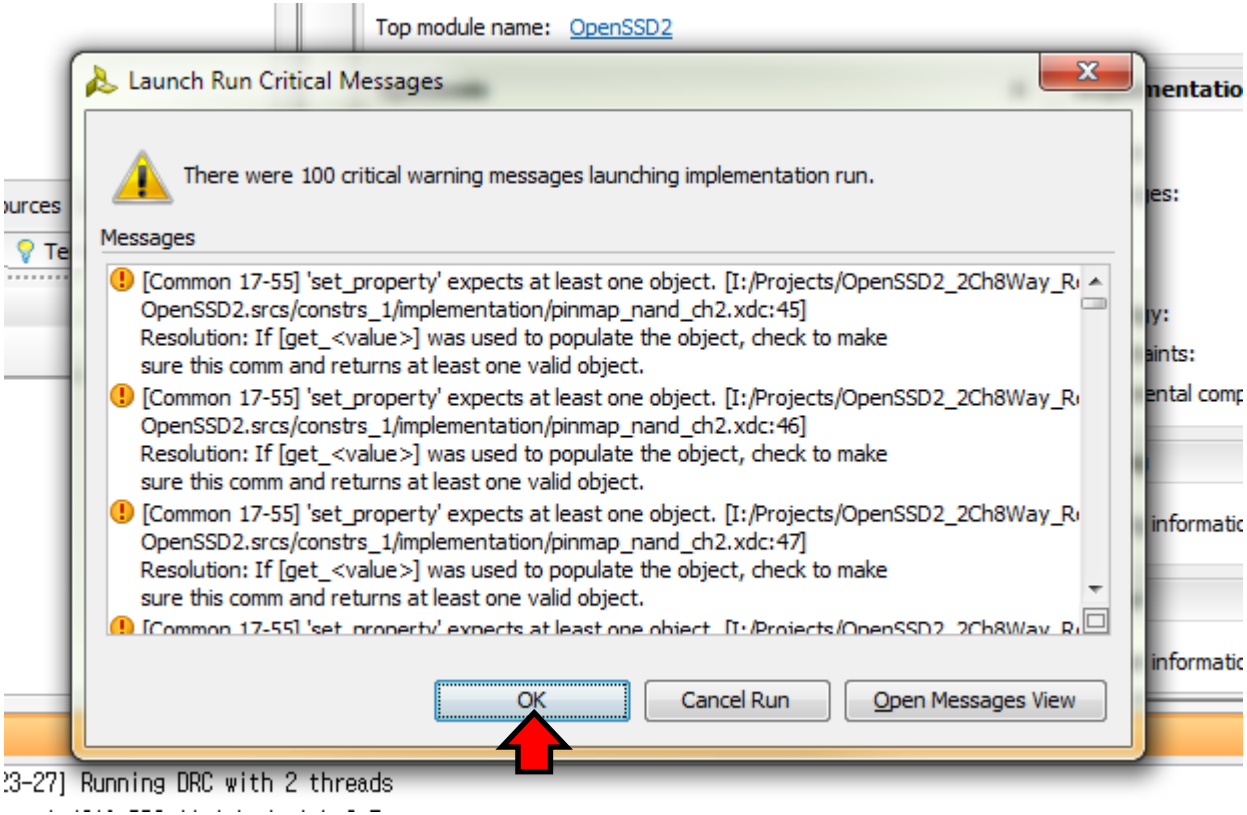
Run Implementation

- **Implementation is running…**



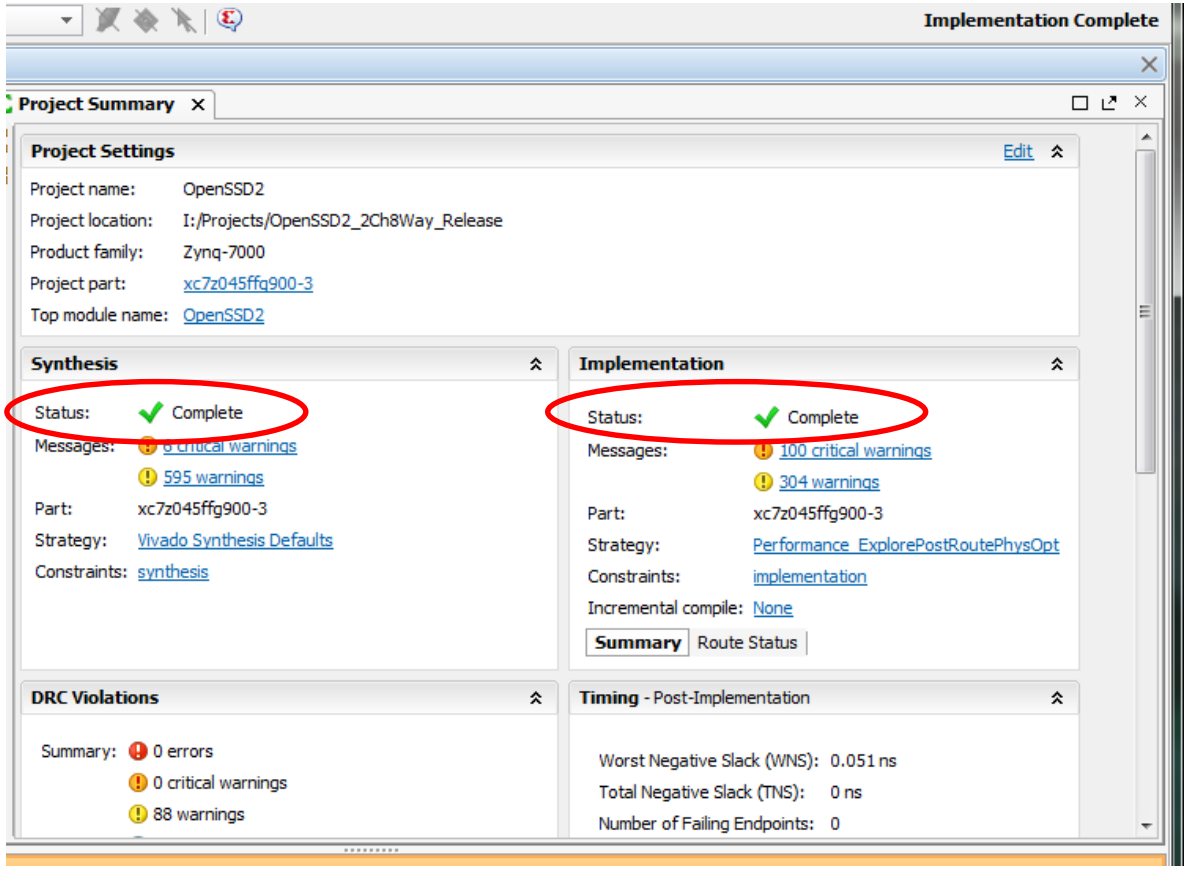
Warning Message

- **The following critical messages appear when implementation is running, but you can ignore it**



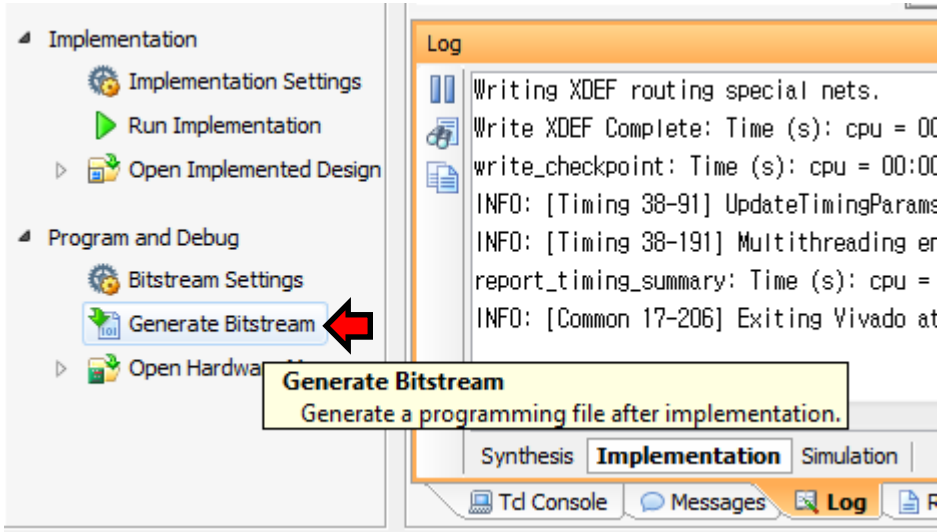
Implementation Complete

- **Check the status of synthesis and implementation**



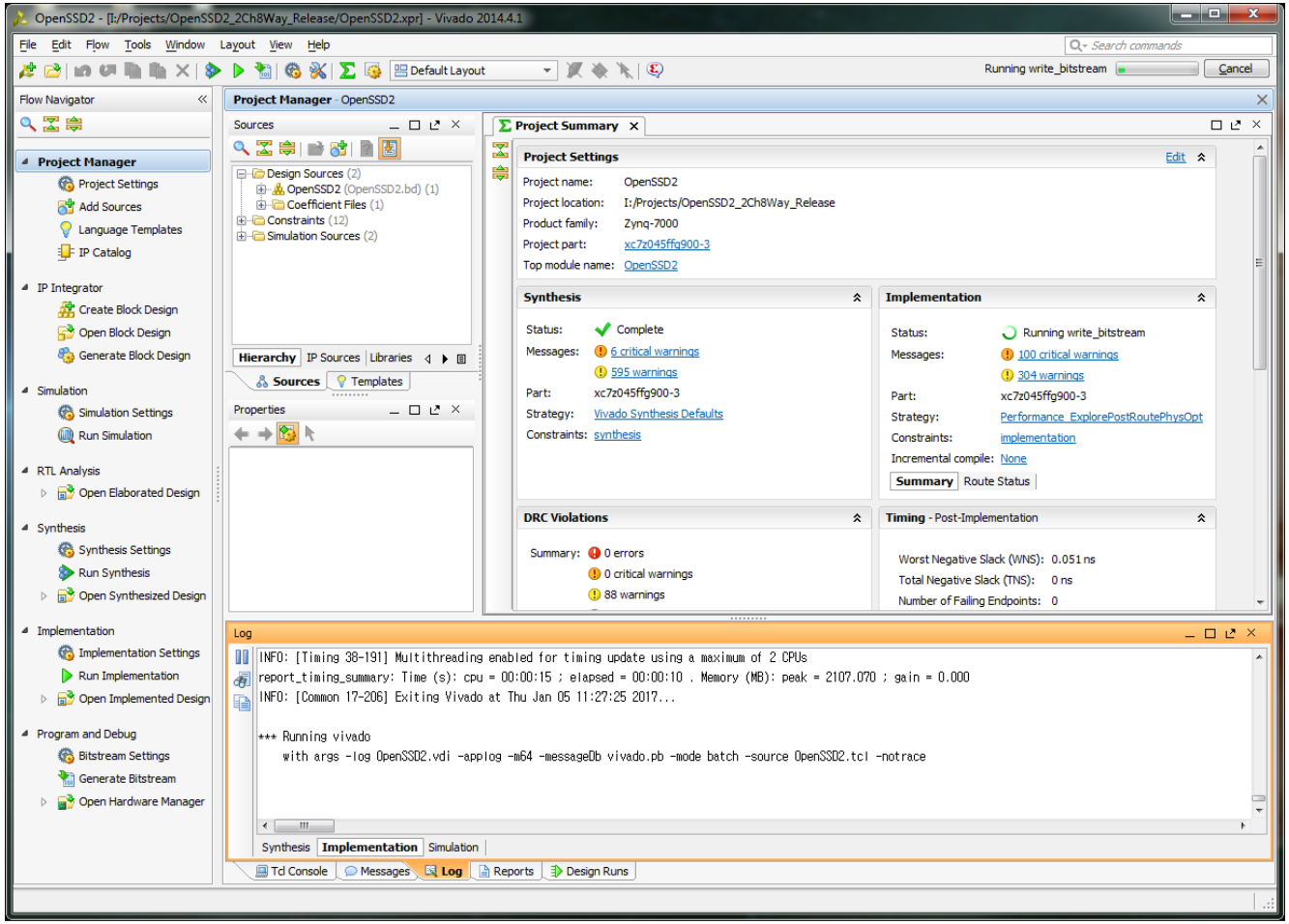
Generate Bitstream (1 / 2)

- **Click “Generate Bitstream”**



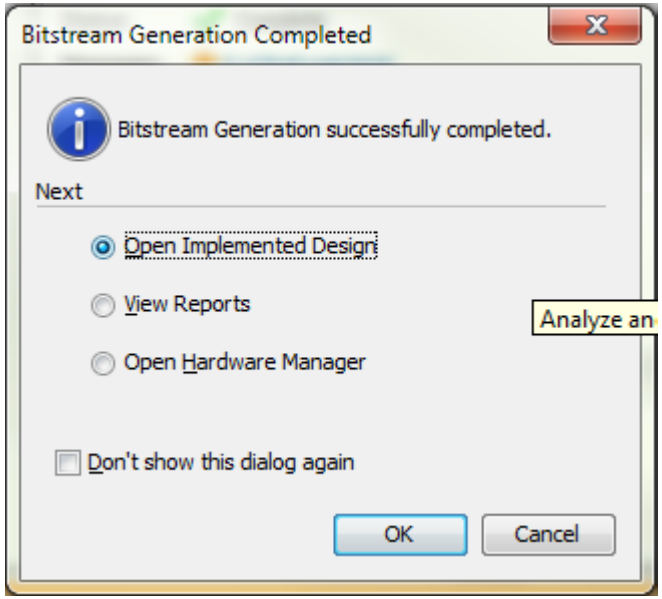
Generate Bitstream (2 / 2)

- **Generate bitstream is running…**



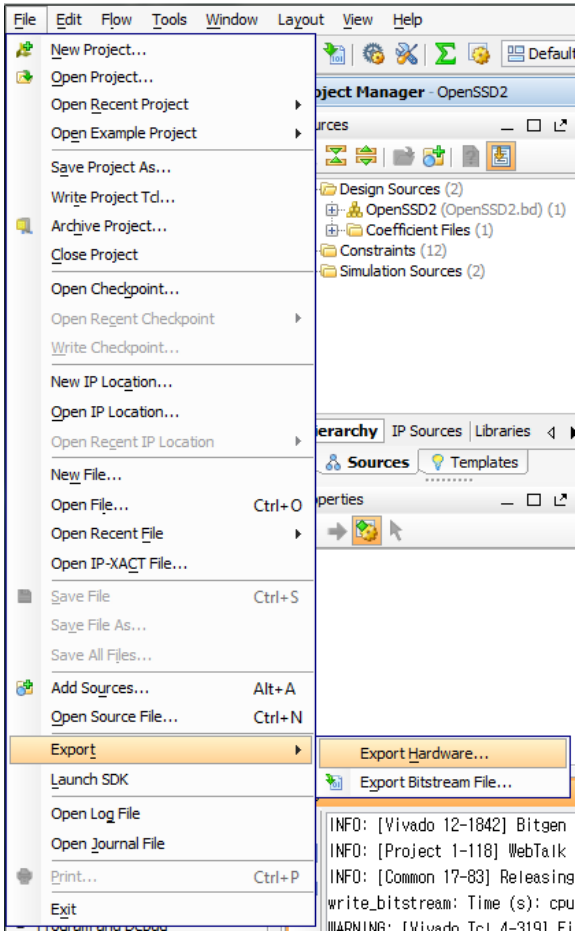
Bitstream Generation Complete

- **If you want to see the implemented design, select open implemented design and click the OK button**



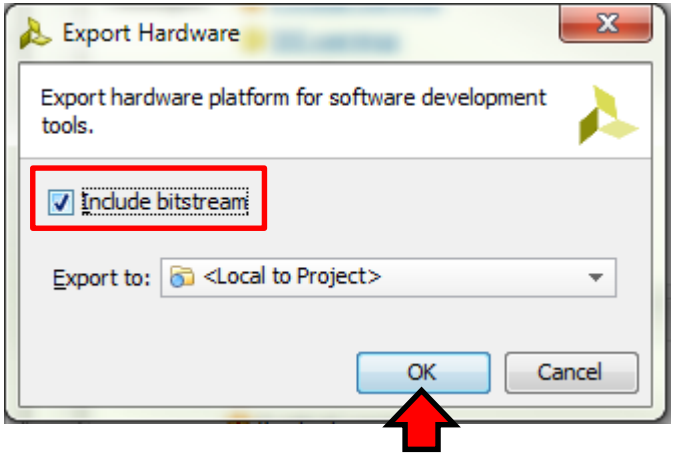
Export hardware (1 / 2)

- Go to File -> Export and click “Export Hardware”



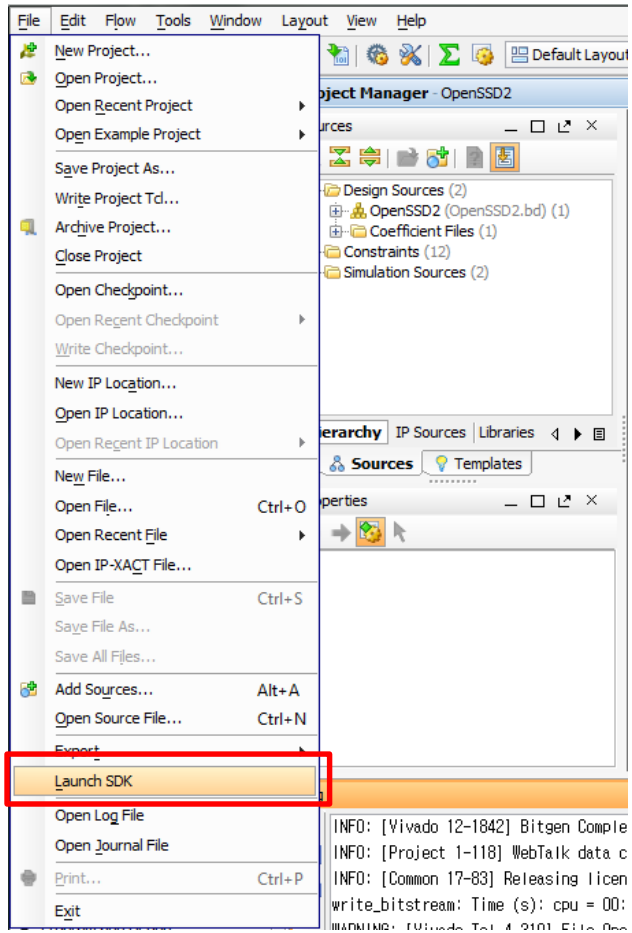
Export hardware (2 / 2)

- **Select the “Include bitstream” and click OK**



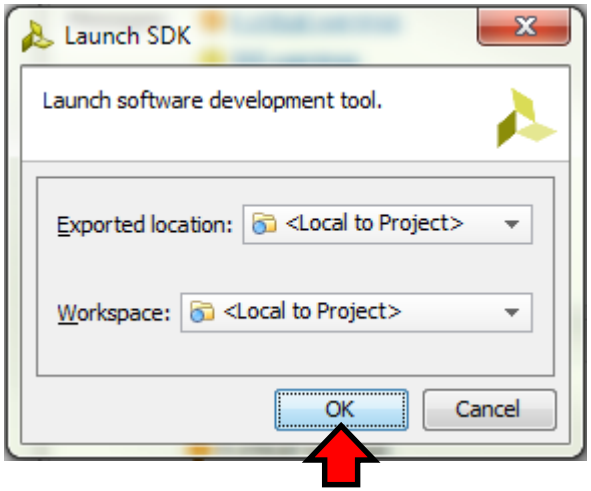
Launch SDK (1 / 4)

- **Go to File -> Launch SDK**



Launch SDK (2 / 4)

- **Click the OK button**

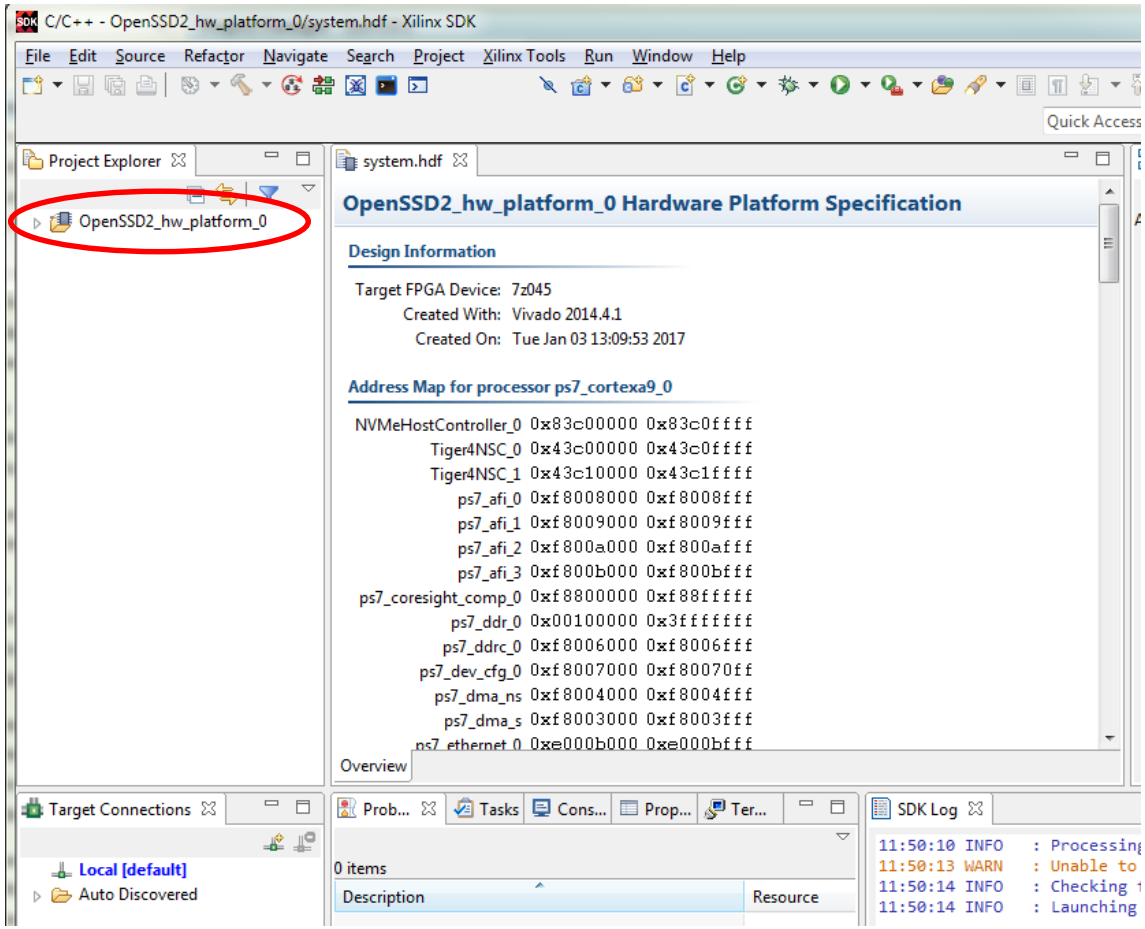


Launch SDK (3 / 4)

- **Then, SDK is launched**

Launch SDK (4 / 4)

- As shown below, exported hardware platform is set as target hardware

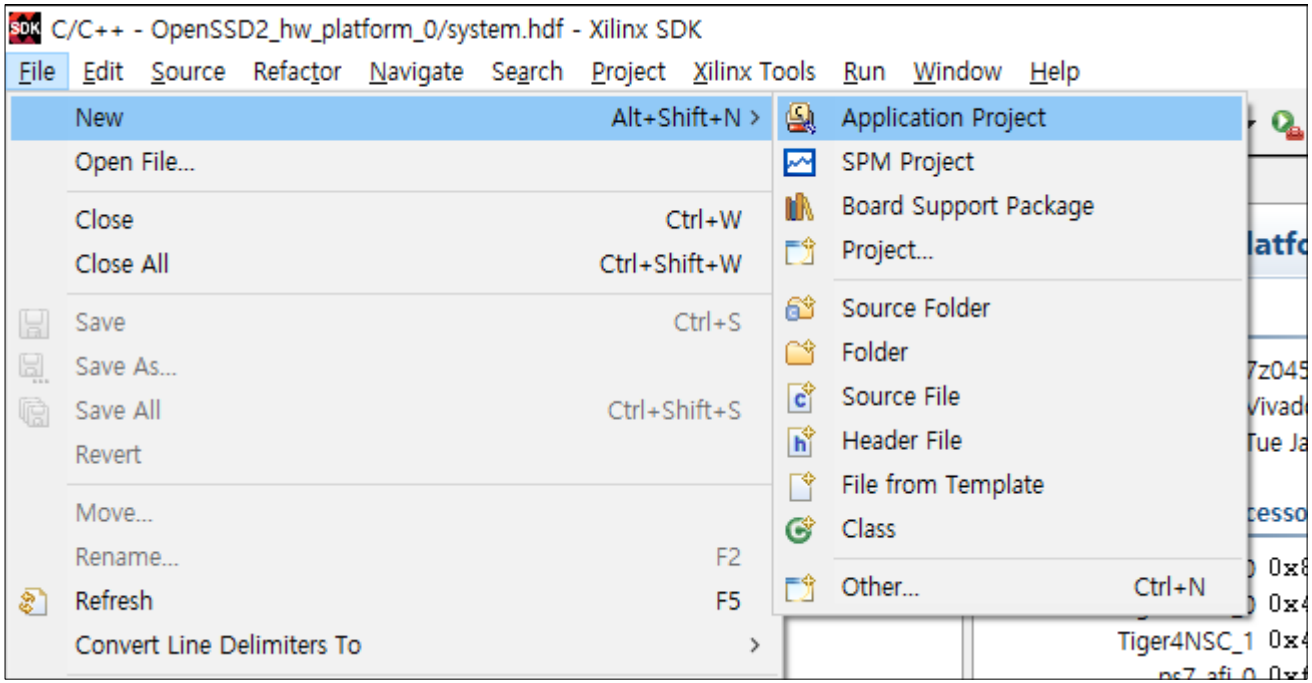


Building Firmware for Pre-defined Project

1. **Create a new application project**
2. **Add source codes**
3. **Build firmware source codes**

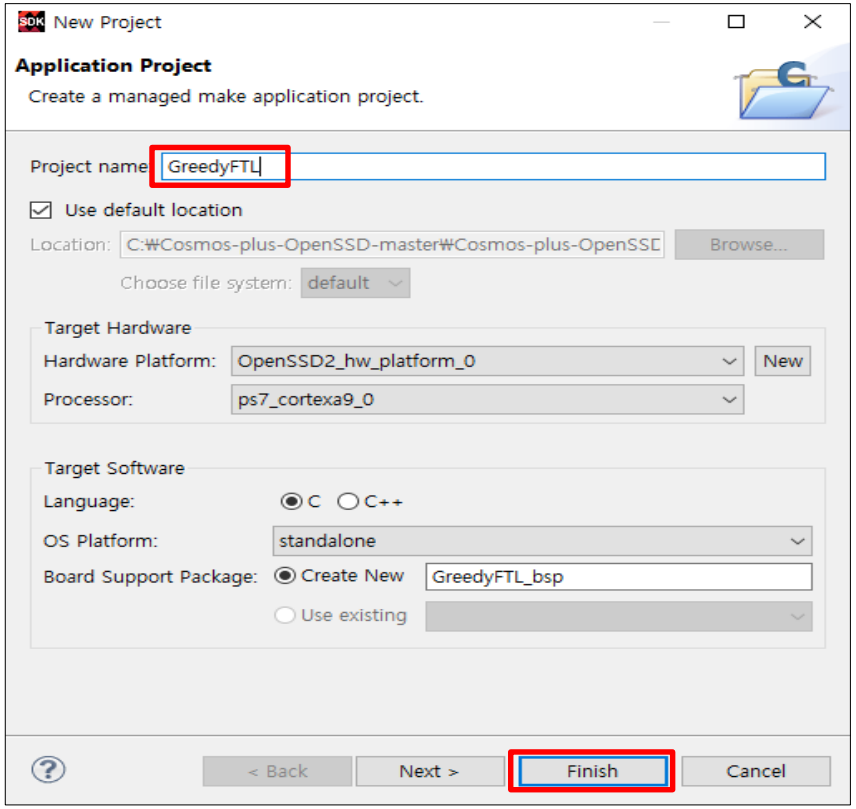
Create a New Application Project

- **Go to File -> New -> Application Project**



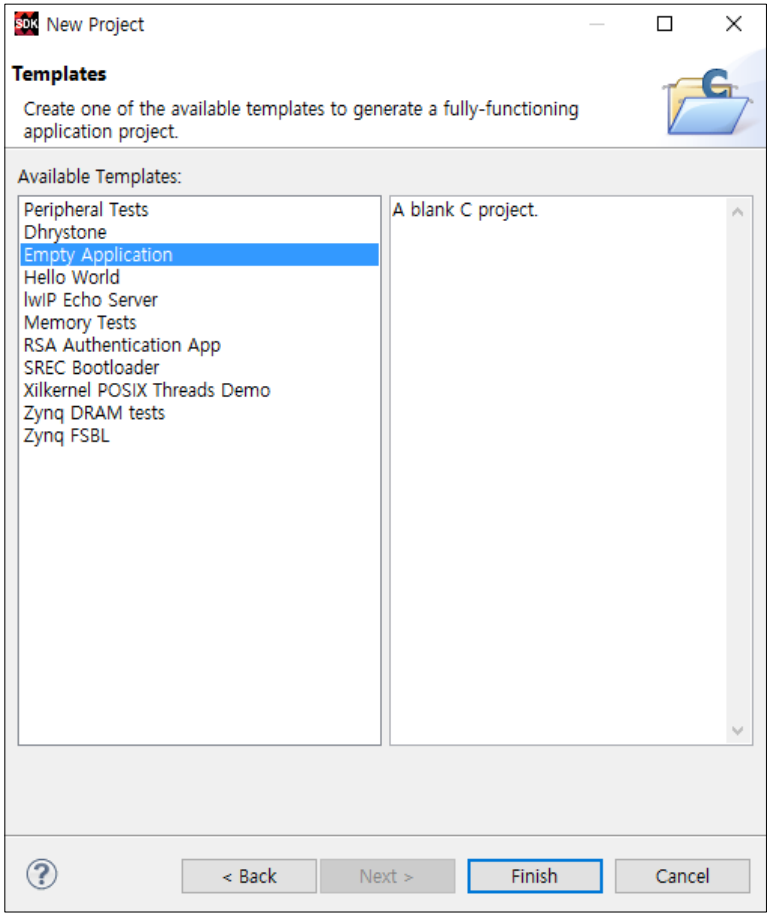
Specify the Project Name

- **Fill in the project name and click “Next”**



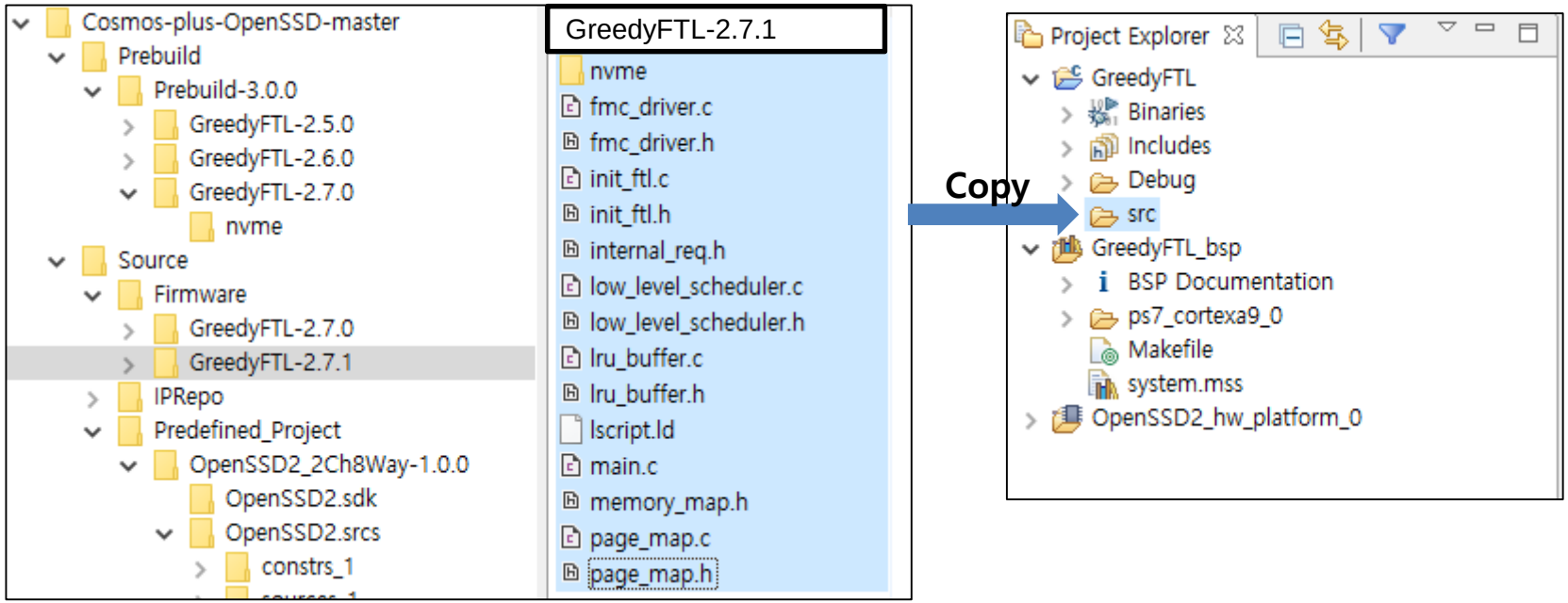
Select a Project Template

- **Select an empty application and finish this template wizard**



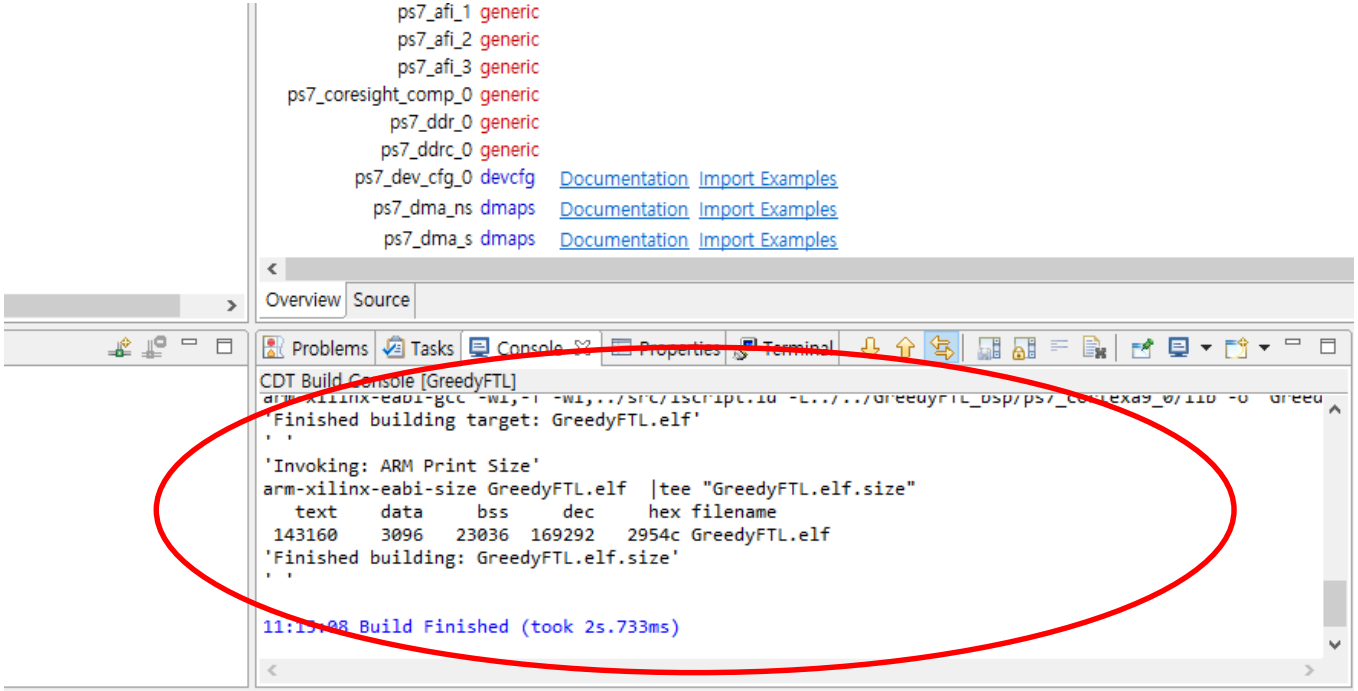
Add Source Code

- **Copy GreedyFTL source files to “src” folder in project explorer**



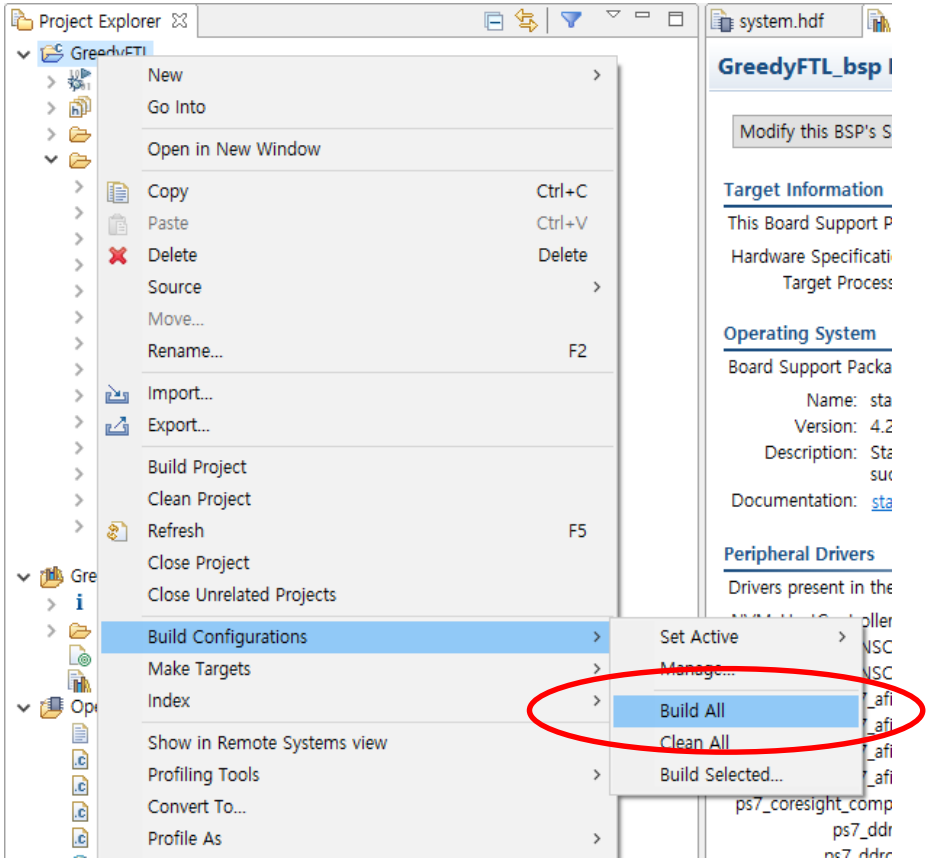
Build Firmware (1 / 2)

- **If everything goes well, the automatic build process should finish successfully**



Build Firmware (2 / 2)

- **Click “Build All” to make both debug and release executables**

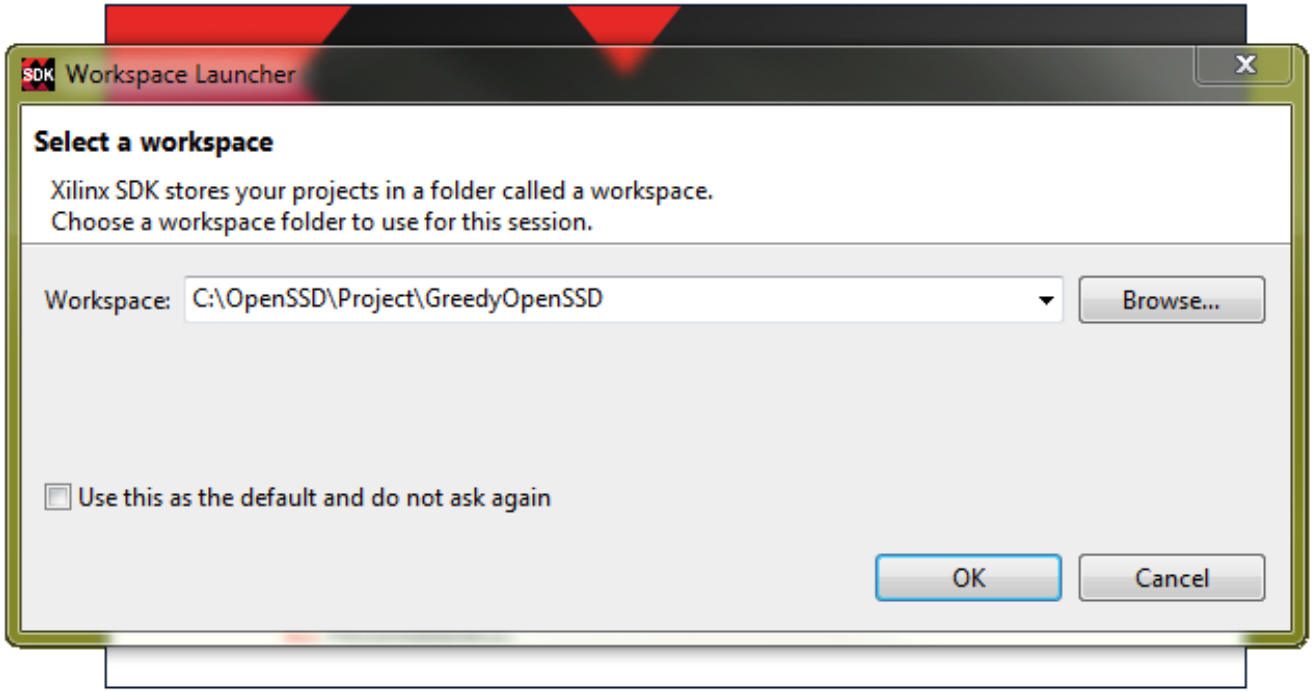


Building Firmware for Prebuild Bitstream

1. **Create a workspace directory and a new application project**
2. **Set a hardware platform**
3. **Add source codes**
4. **Build firmware source codes**

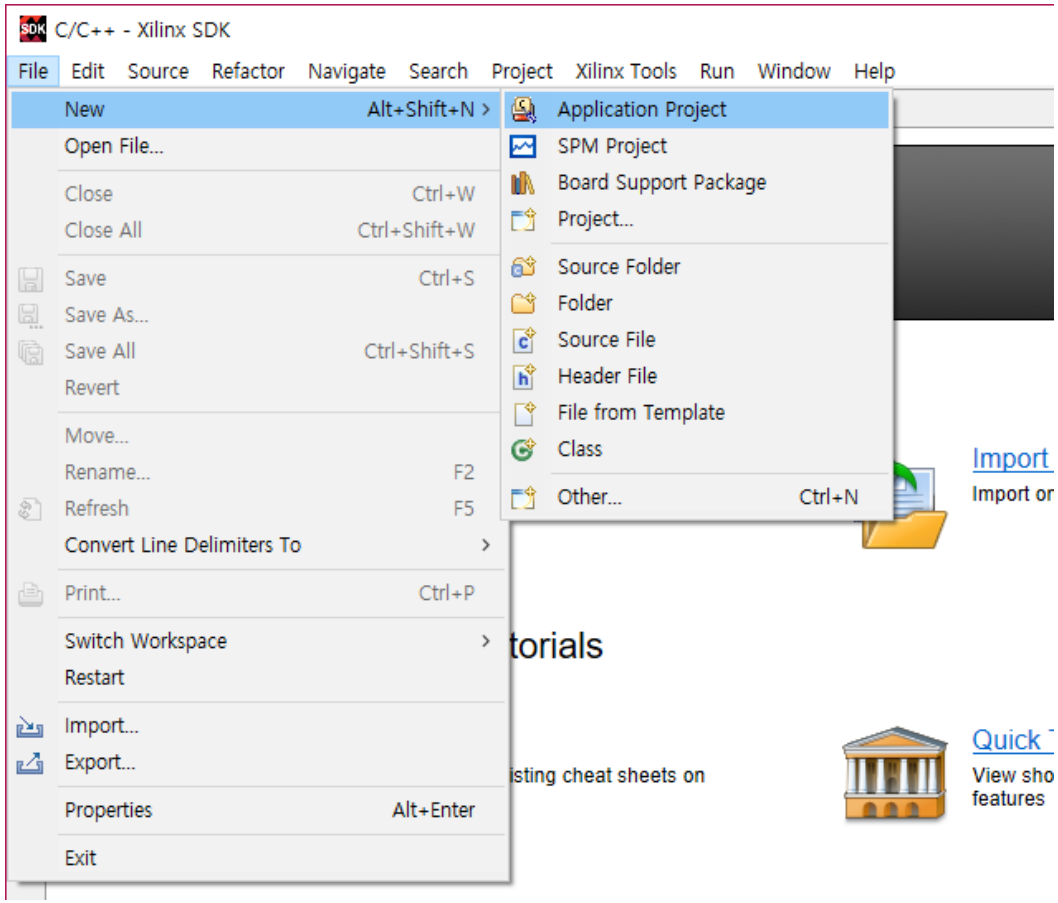
Create a Workspace Directory

- **Launch Xilinx SDK and designate the workspace**



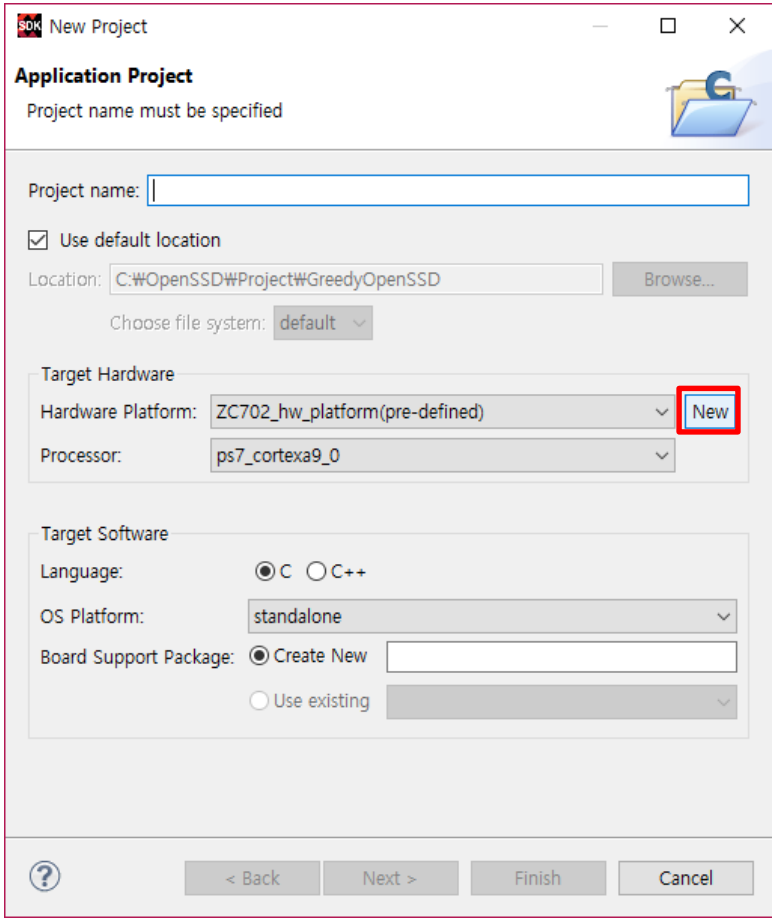
Create a New Application Project

- **Go to File -> New -> Application Project**



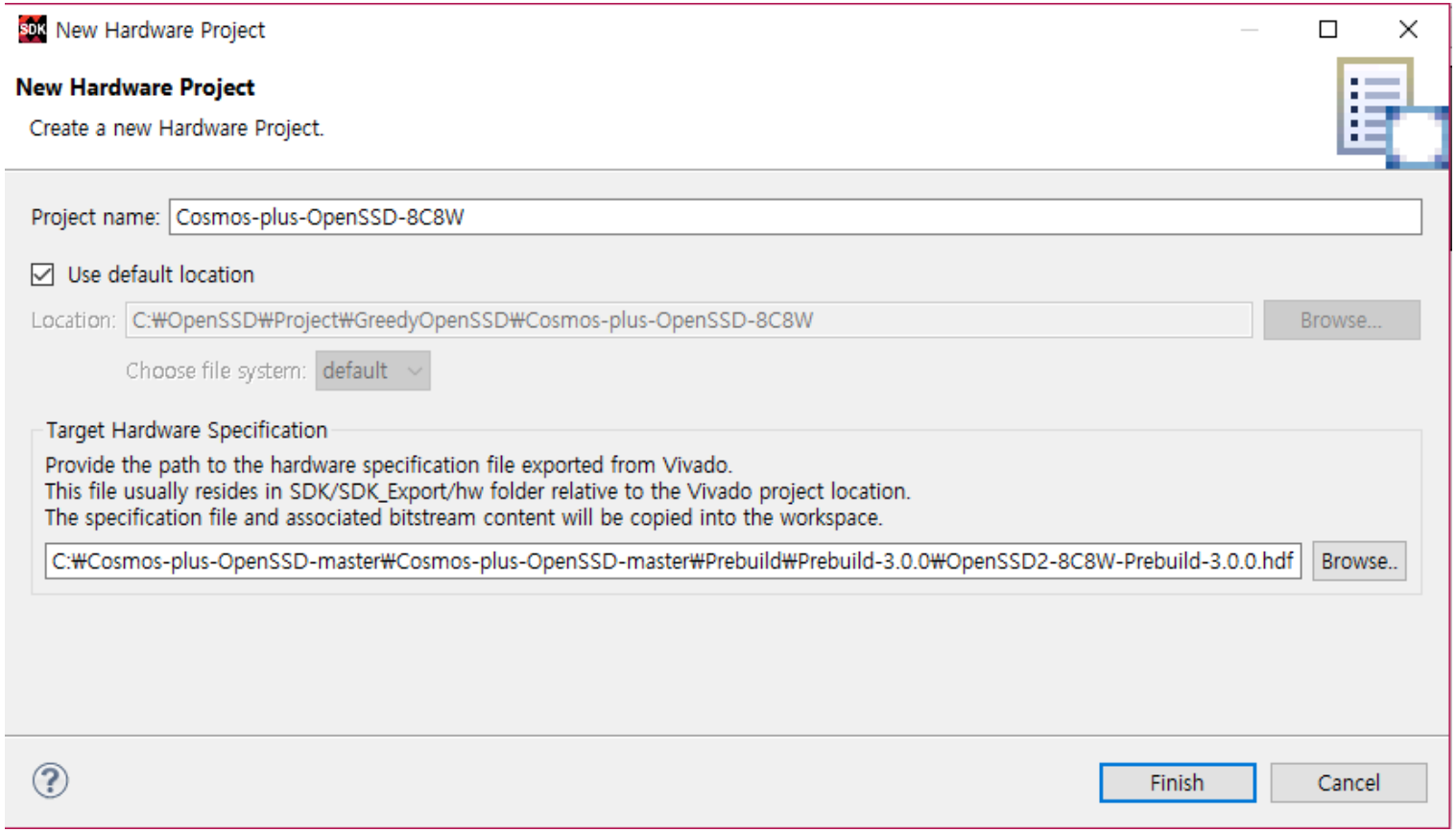
Import the Prebuild Bitstream from HDF (1 / 2)

- **Press “New” to register the hardware description file (HDF)**



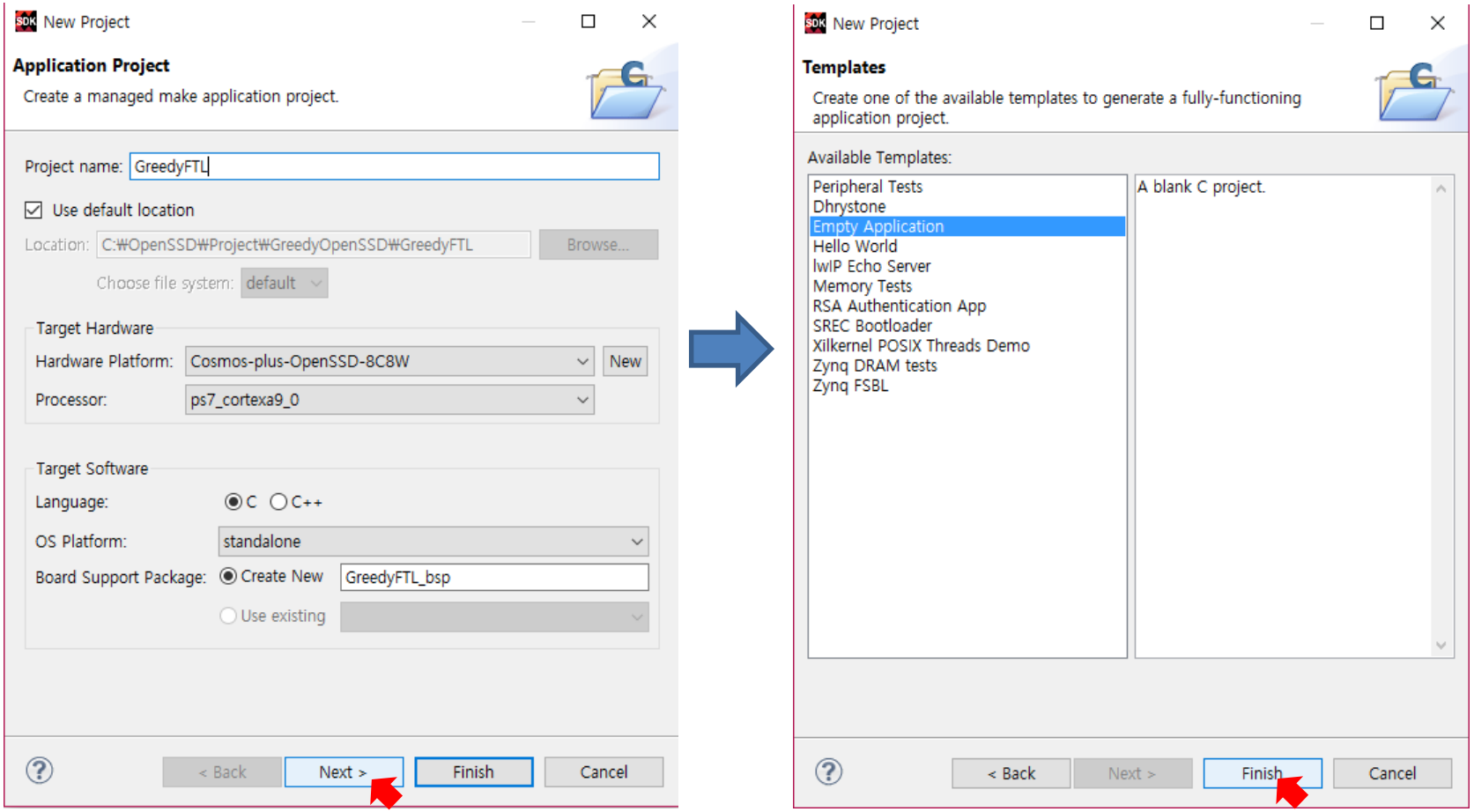
Import the Prebuild Bitstream from HDF (2 / 2)

- **Name the hardware project and specify the path of the HDF**



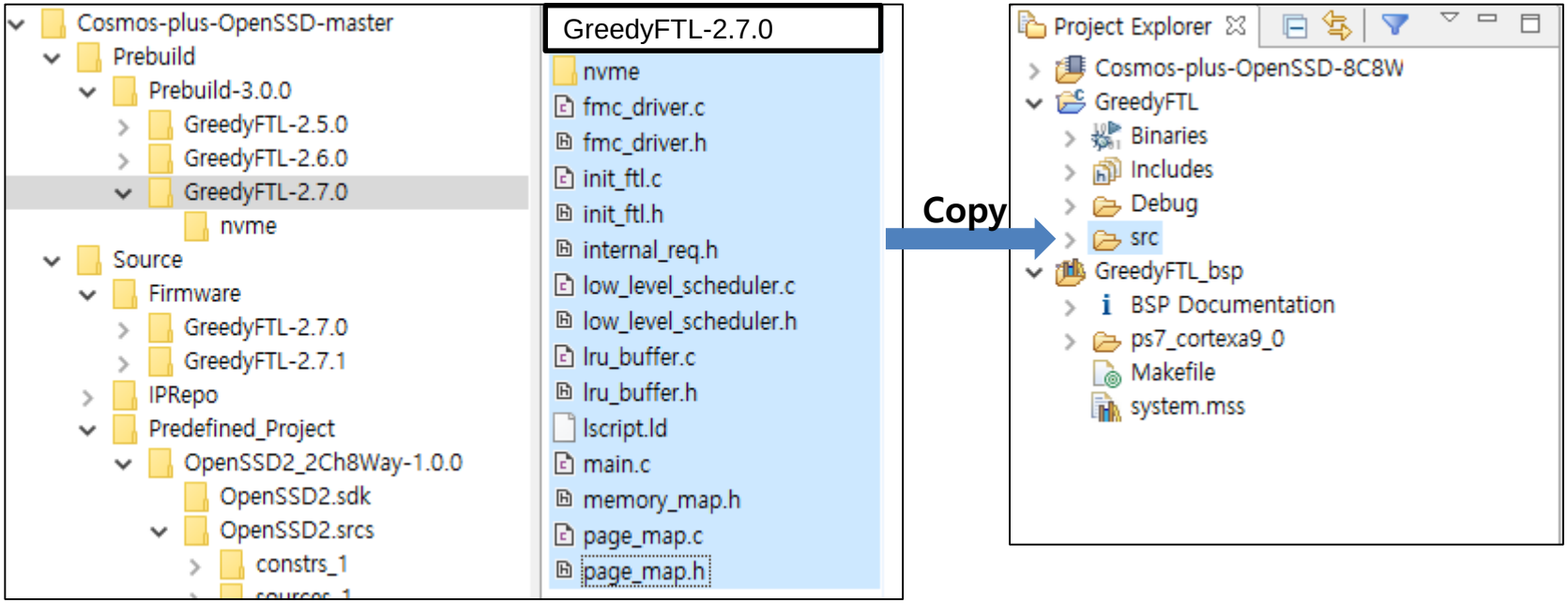
Finish the Project Wizard

- **Name the application project and finish this project wizard**



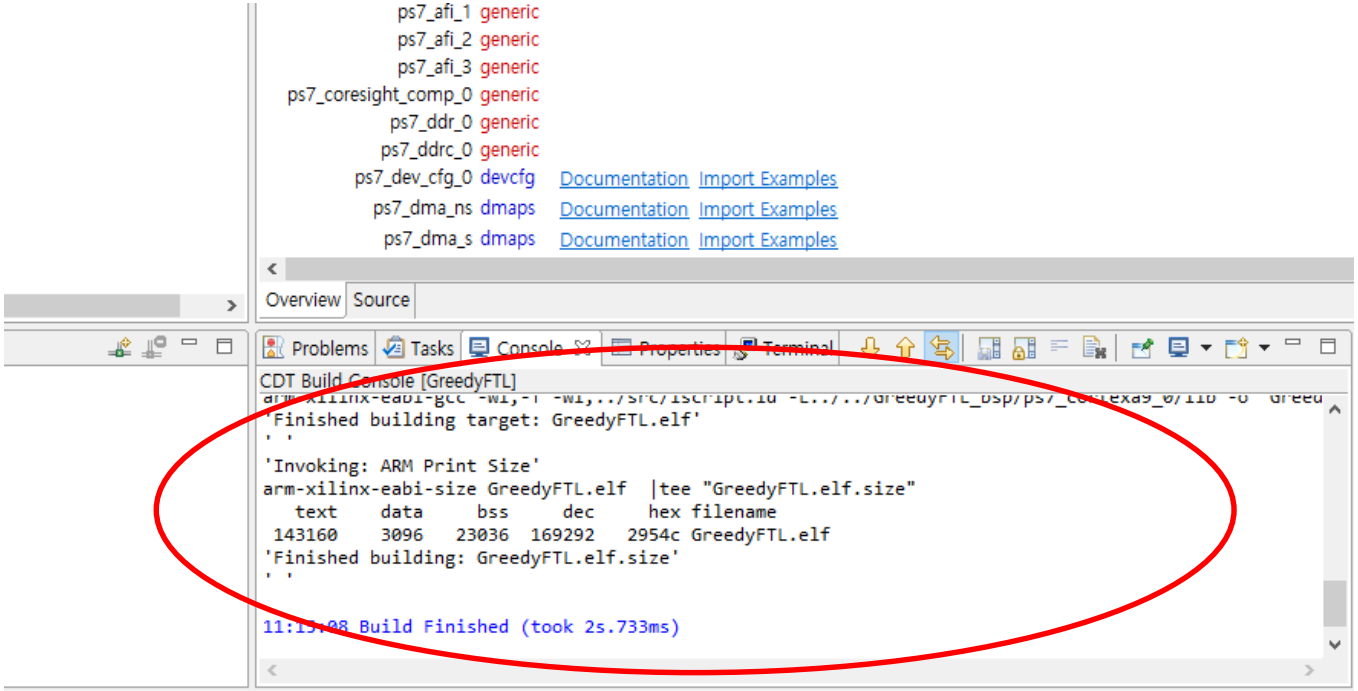
Add Source Code

- **Copy GreedyFTL source files to “src” folder in project explorer**



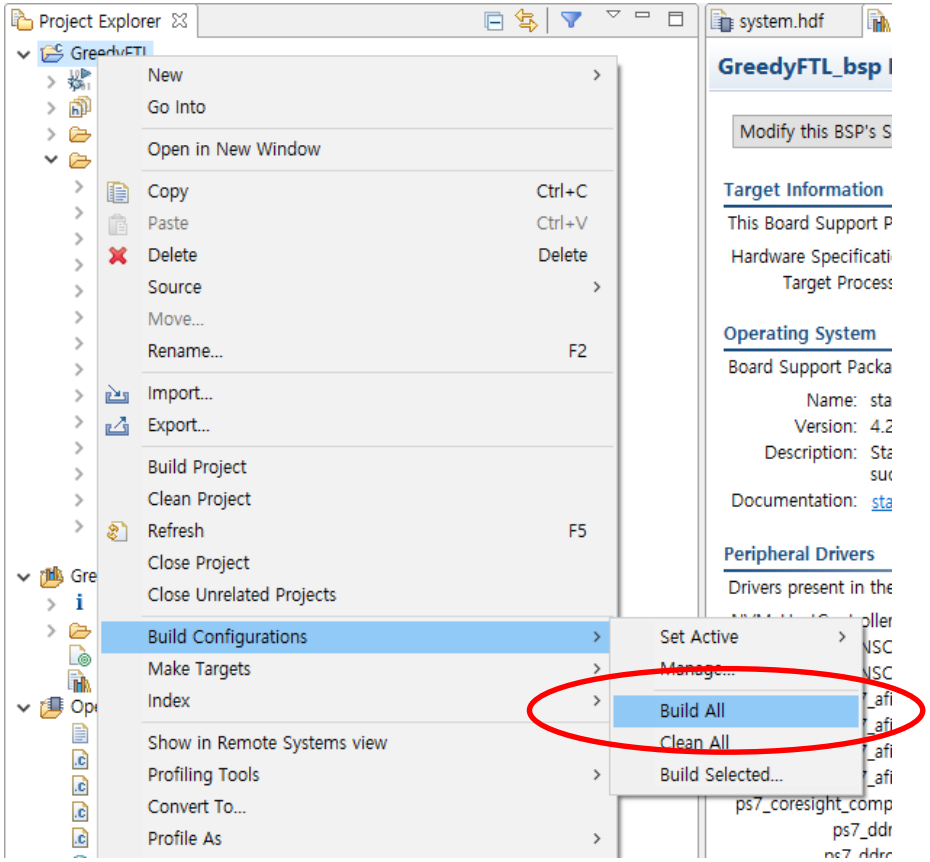
Build Firmware (1 / 2)

- **If everything goes well, the automatic build process should finish successfully**



Build Firmware (2 / 2)

- **Click “Build All” to make both debug and release executables**

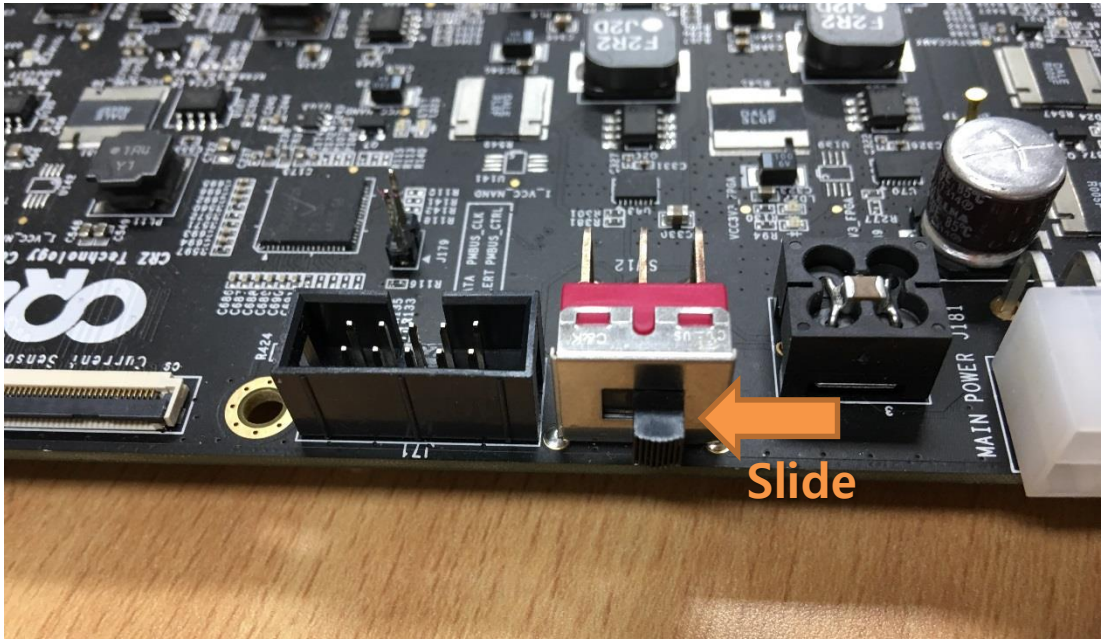


Preparing for Operating Cosmos+ OpenSSD

1. **Power on the platform board**
2. **Configure UART**
3. **Program FPGA**
4. **Execute firmware**

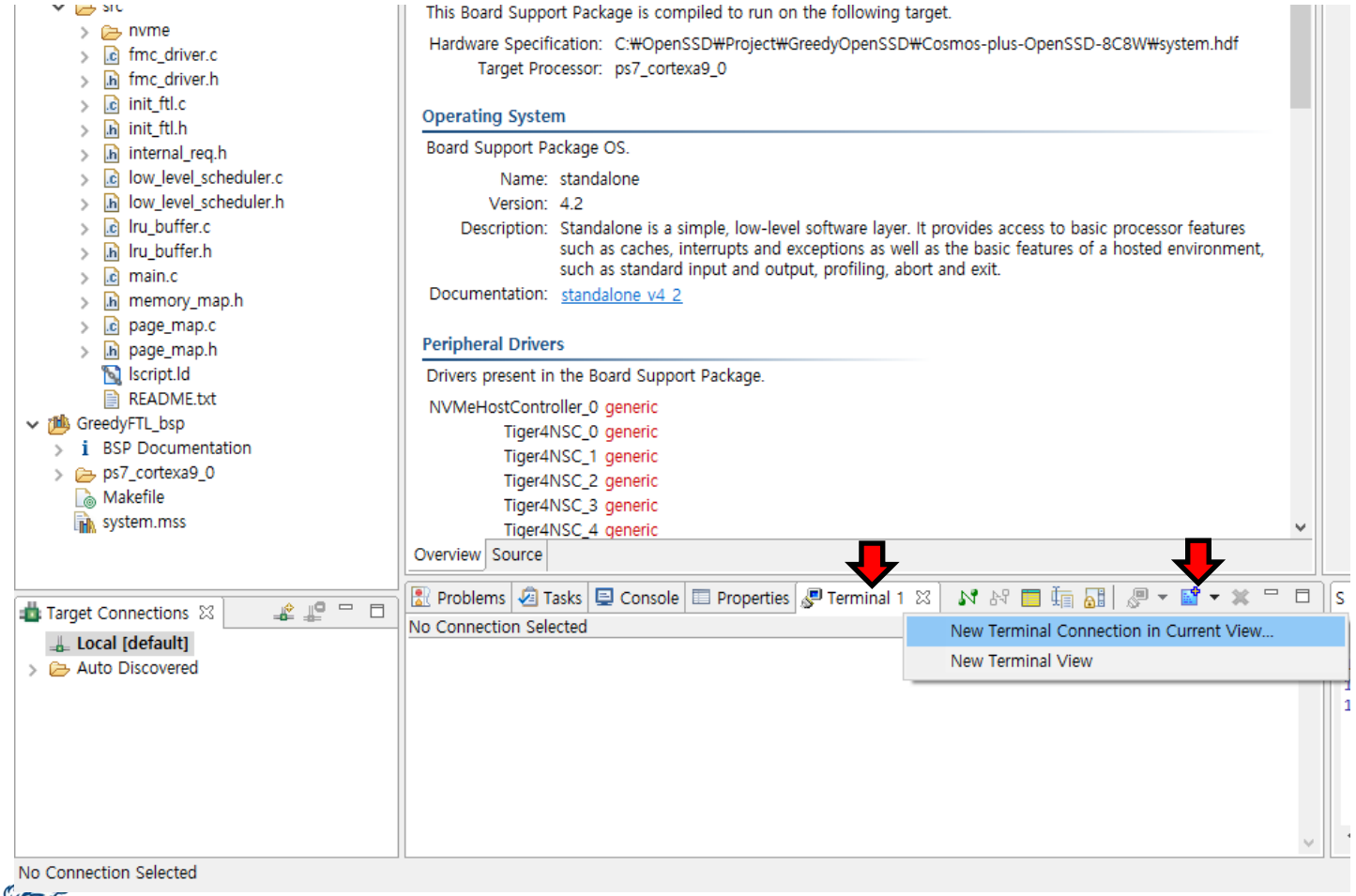
Power on the Platform Board

- **Before you power on the board, make sure that your host computer is powered off**



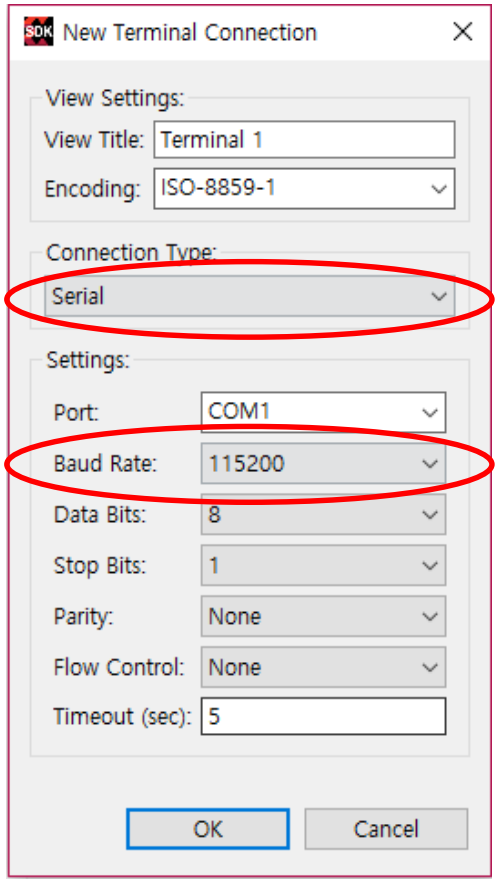
Configure UART

- **In SDK, go to Terminal -> New Terminal Connection as shown below**



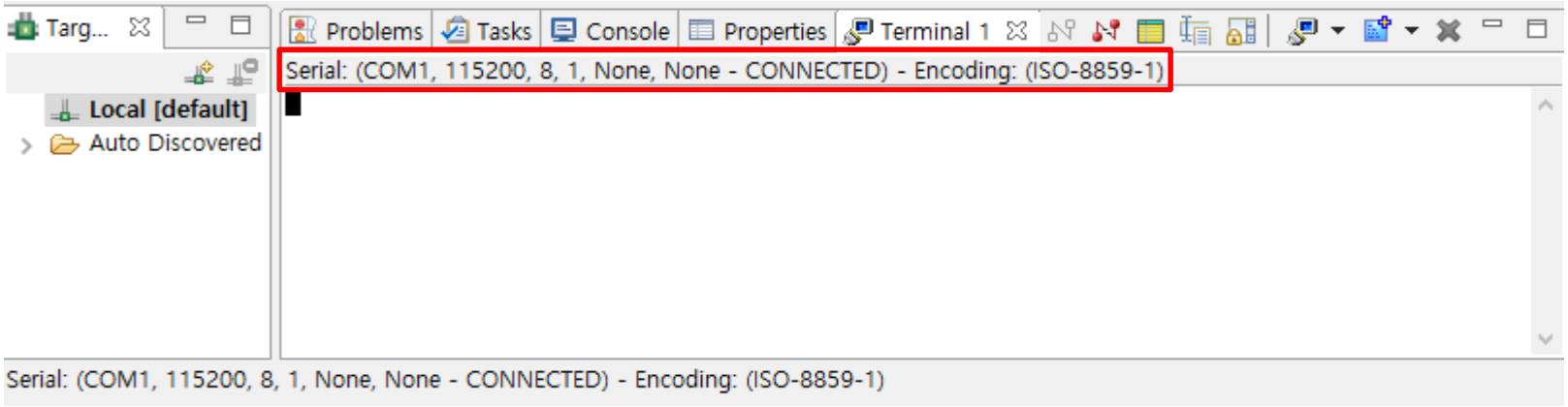
Configure UART

- **Set “Connection Type” and “Baud Rate” to serial and 115200, respectively**



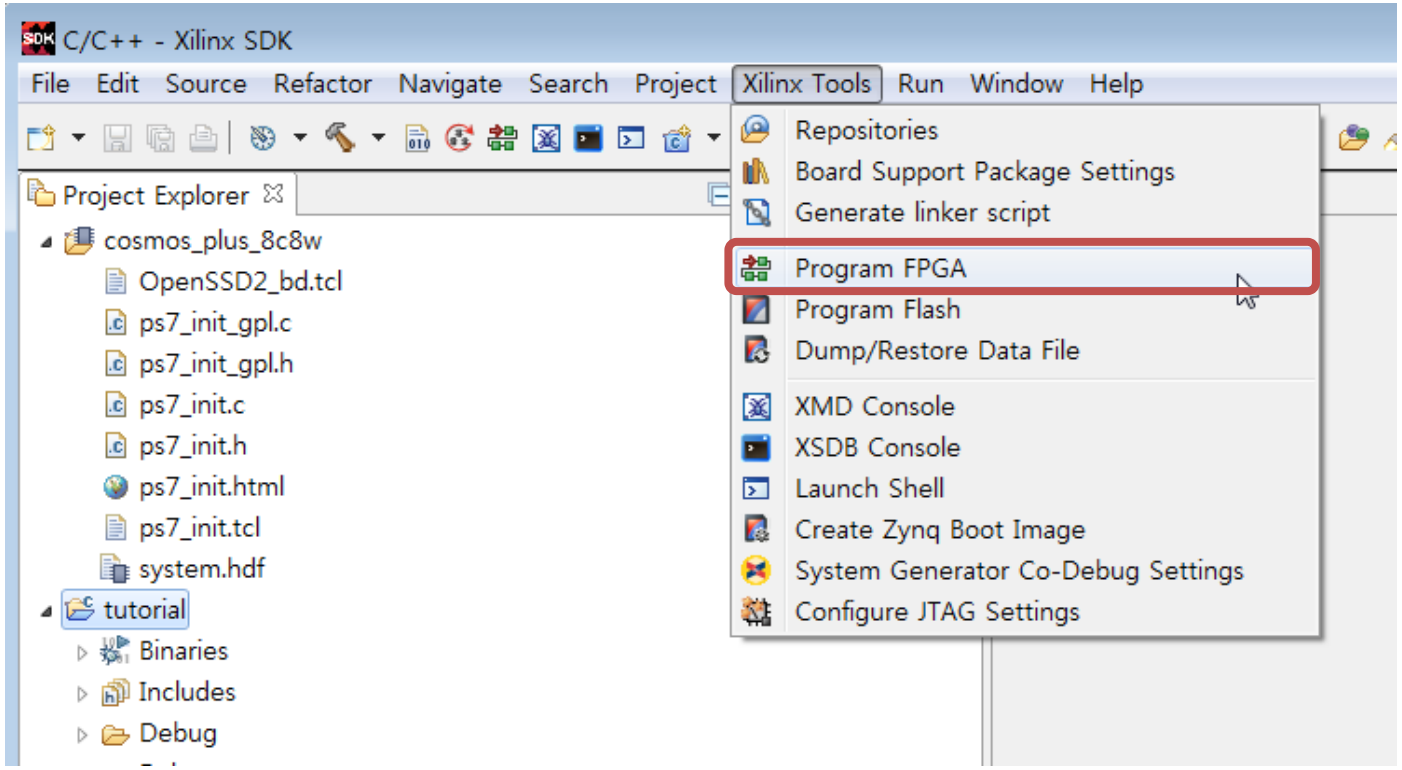
Configure UART

- **If then, UART is connected as shown below**



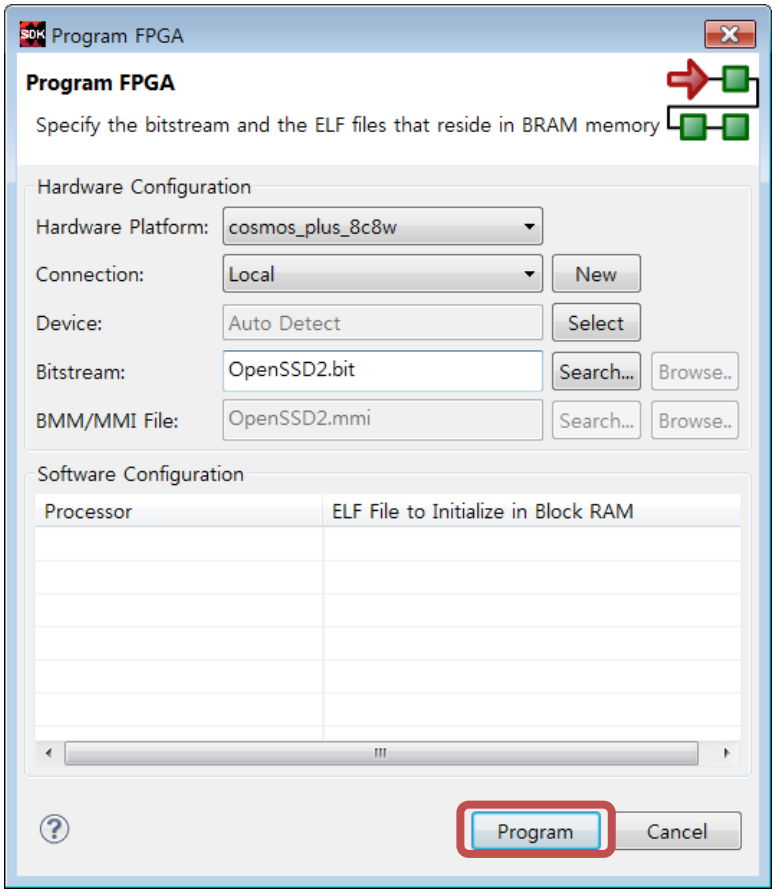
Program FPGA (1 / 4)

- **Click “Xilinx Tools” -> click “Program FPGA”**



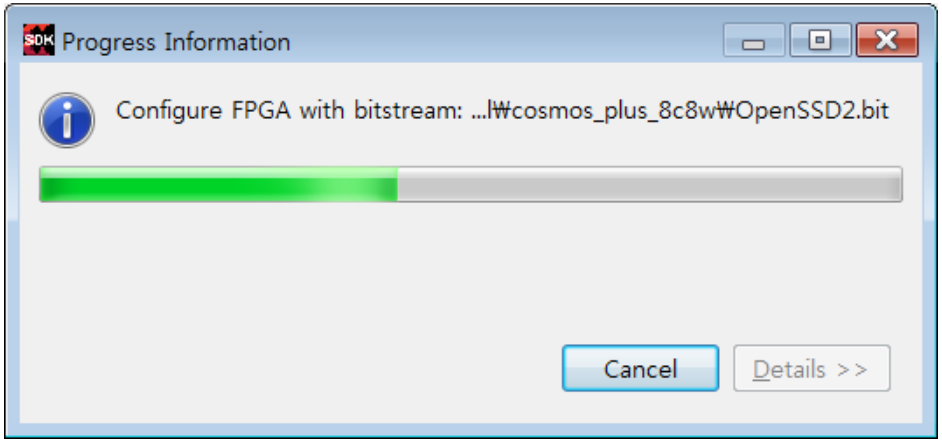
Program FPGA (2 / 4)

- **Click “Program” to program FPGA**



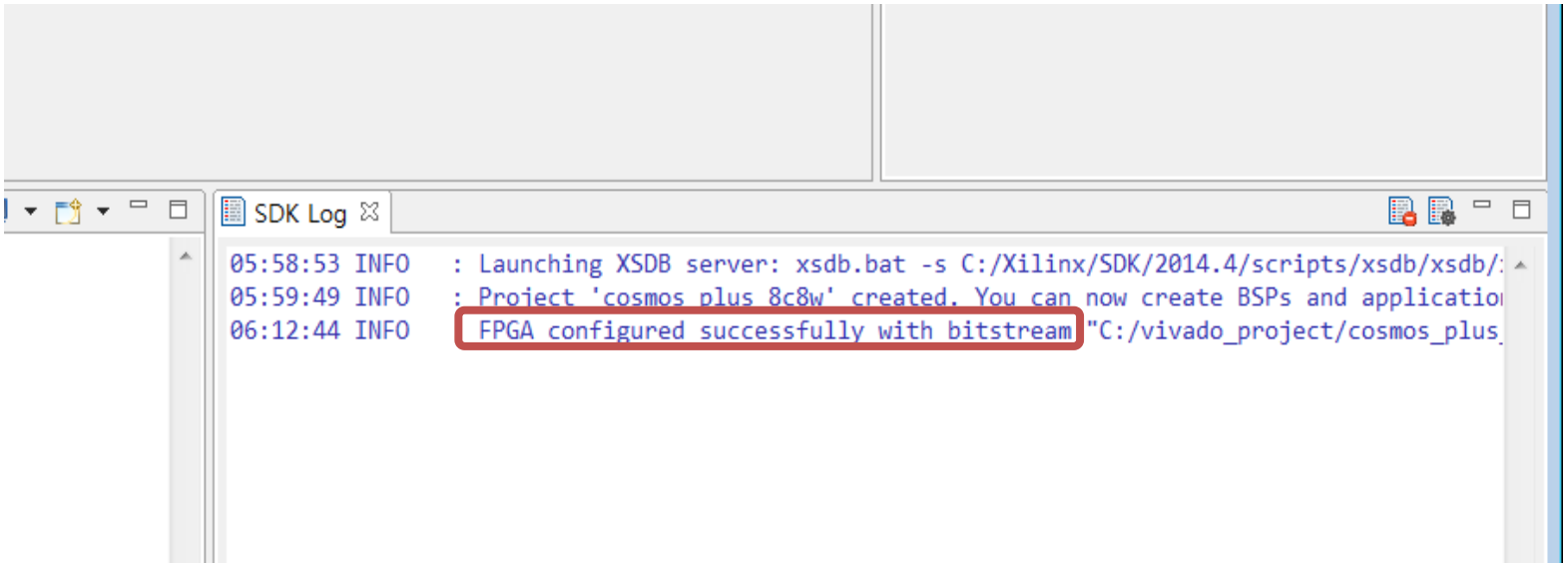
Program FPGA (3 / 4)

- **Hang on a second**



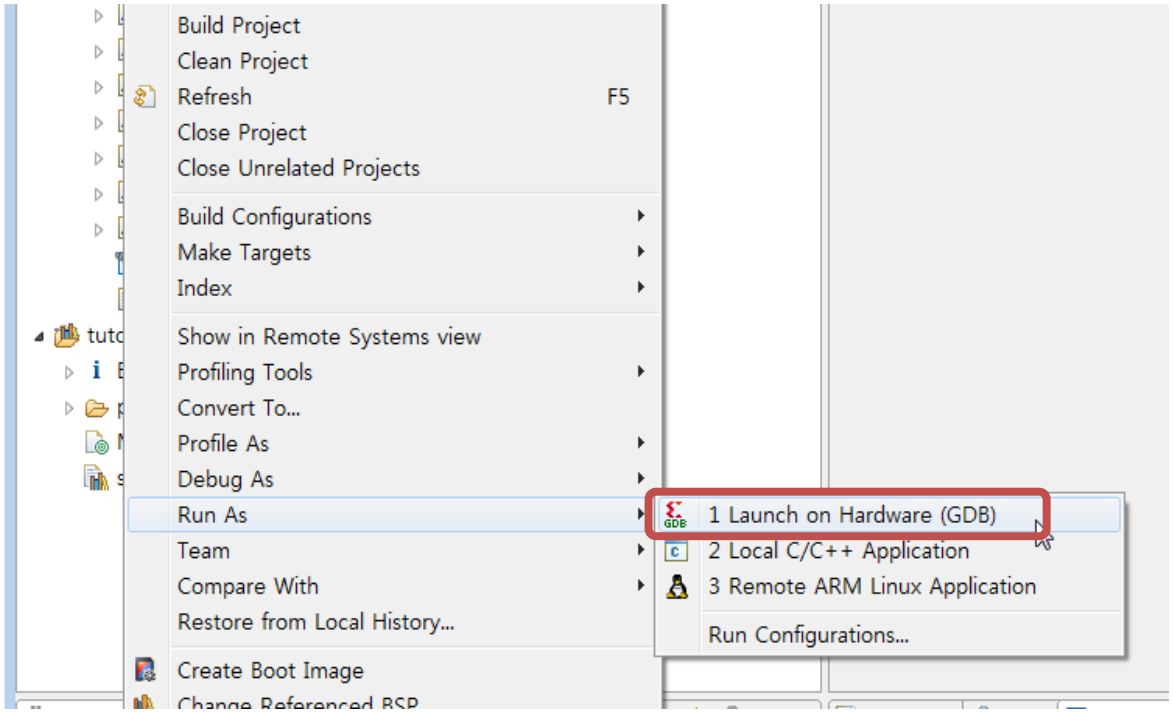
Program FPGA (4 / 4)

- **Check FPGA programming done successfully**



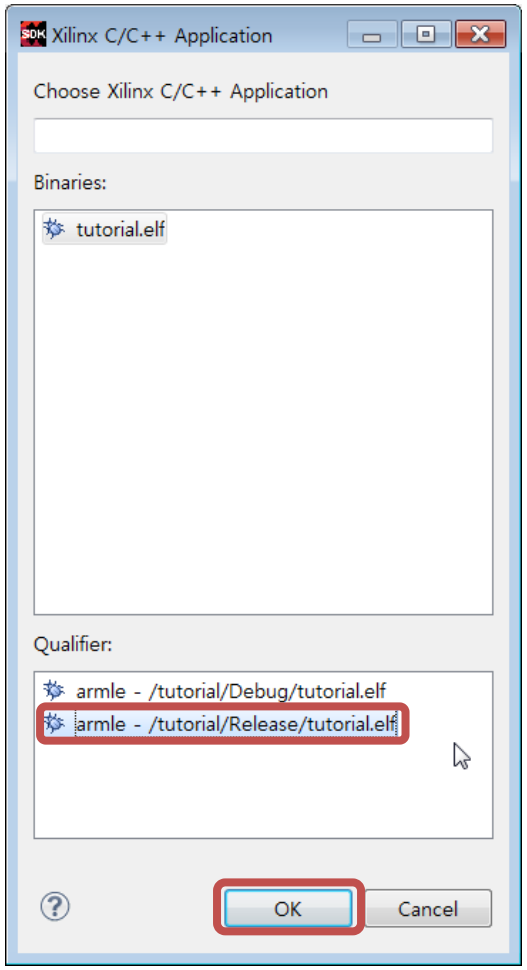
Execute Firmware (1 / 3)

- **Right click on the application project -> “Run As” -> click “1 Launch on Hardware (GDB)”**



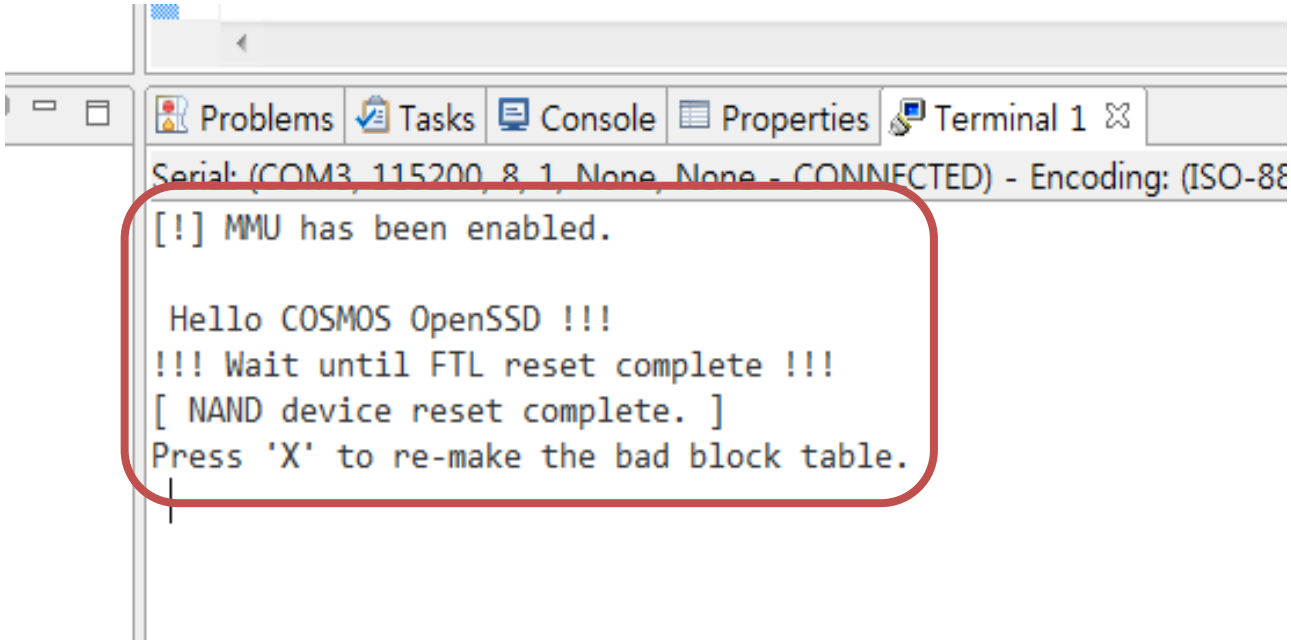
Execute Firmware (2 / 3)

- **Click the firmware to execute -> click “OK” -> wait UART message**



Execute Firmware (3 / 3)

- **Press ‘n’ to maintain the bad block table**



Bad Block Management (1 / 2)

- **Choose whether remake the bad block table in FTL initialization step**
  - If you want to remake the bad block table, press “X” on UART terminal
    - Bad block table format of greedy FTL v2.7.0 is different from the previous versions
    - Damaged bad block table can be recovered

```
[!] MMU has been enabled.
 Hello COSMOS OpenSSD !!!
!!! Wait until FTL reset complete !!!
[ NAND device reset complete. ]
Press 'X' to re-make the bad block table.
```

“X” erases all blocks including a metadata block

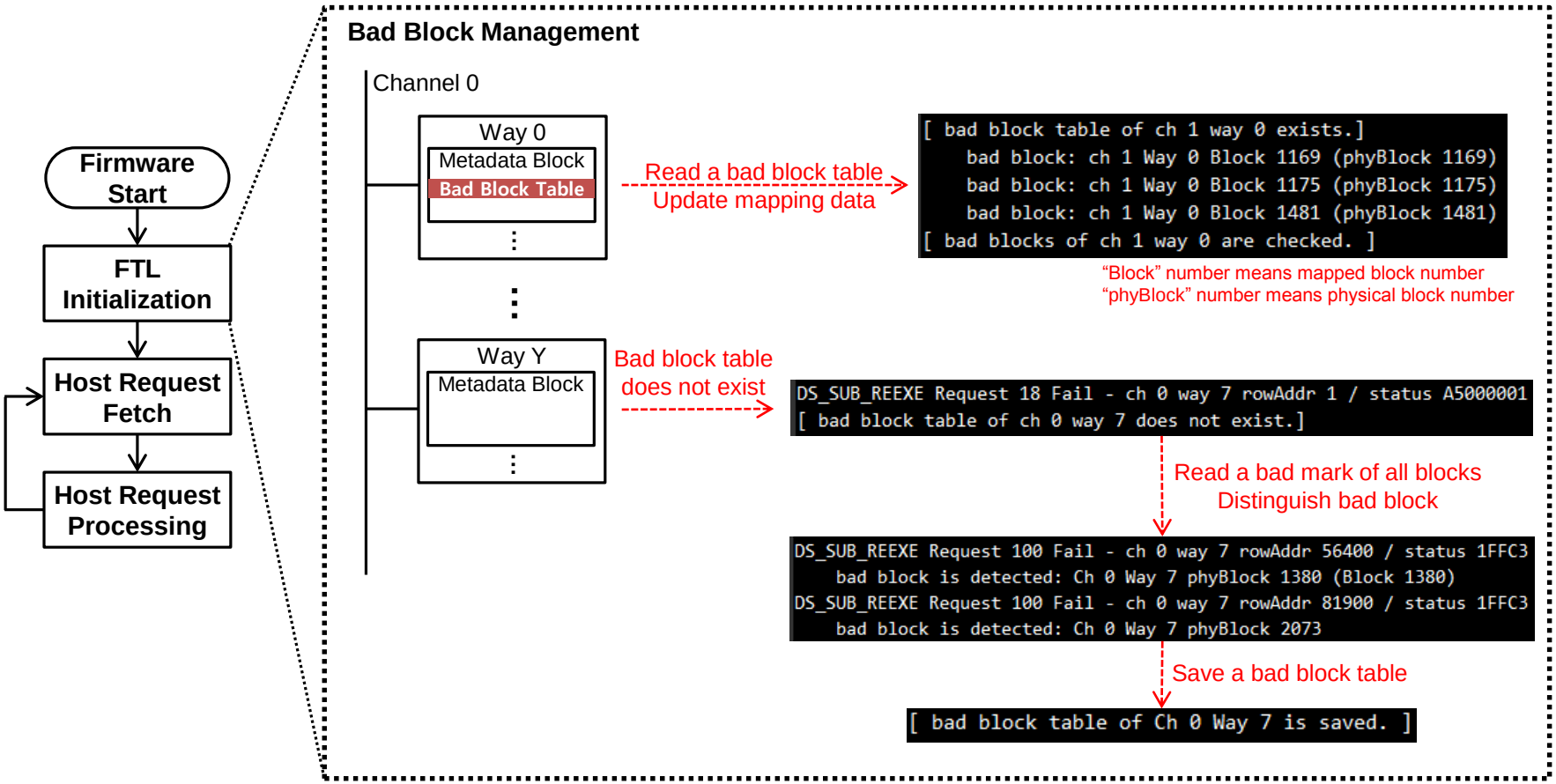
```
DS_SUB_REEXE Request 18 Fail - ch 0 way 7 rowAddr 1 / status A5000001
[ bad block table of ch 0 way 7 does not exist.]
```

Others maintain the bad block table

```
[ bad block table of ch 1 way 0 exists.]
    bad block: ch 1 Way 0 Block 1169 (phyBlock 1169)
    bad block: ch 1 Way 0 Block 1175 (phyBlock 1175)
    bad block: ch 1 Way 0 Block 1481 (phyBlock 1481)
[ bad blocks of ch 1 way 0 are checked. ]
```

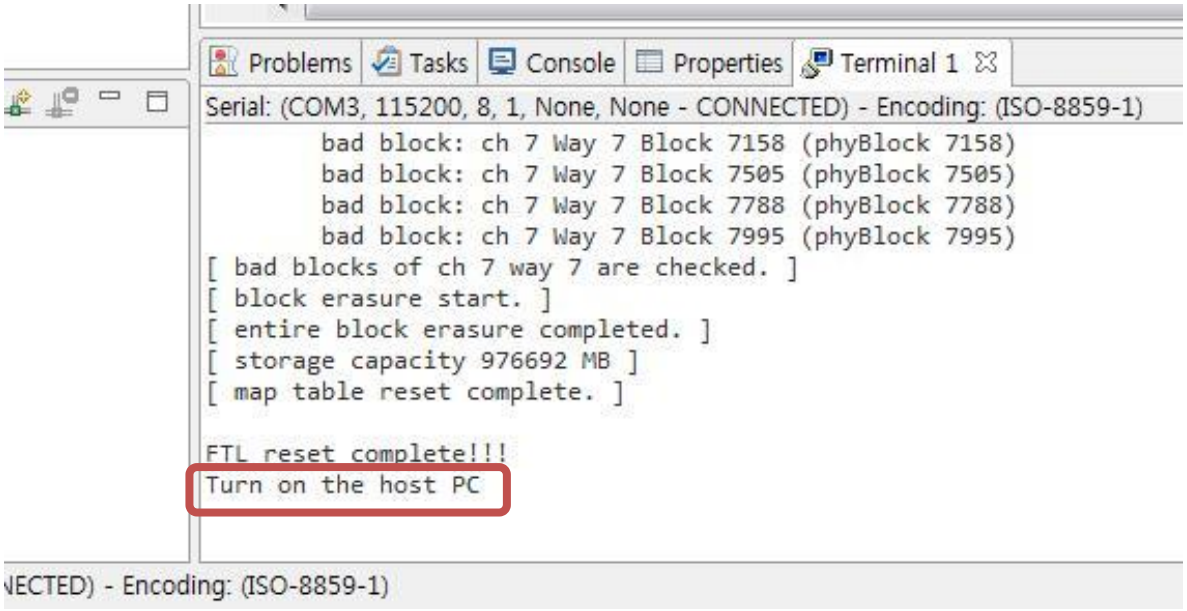
Bad Block Management (2 / 2)

- **Bad blocks are detected in FTL initialization step**



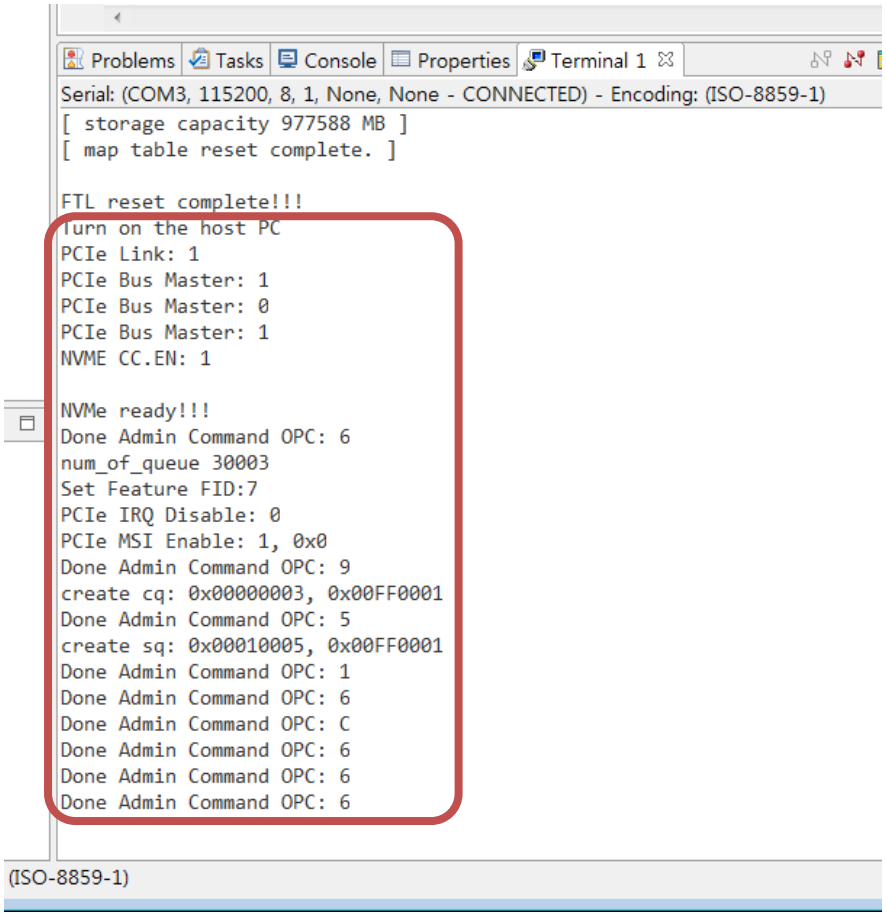
Turn on the Host PC

- **Turn on the host PC when the firmware reset is done**



UART Messages While Host Computer is Booting up

- **NVMe SSD initialization steps are on going**

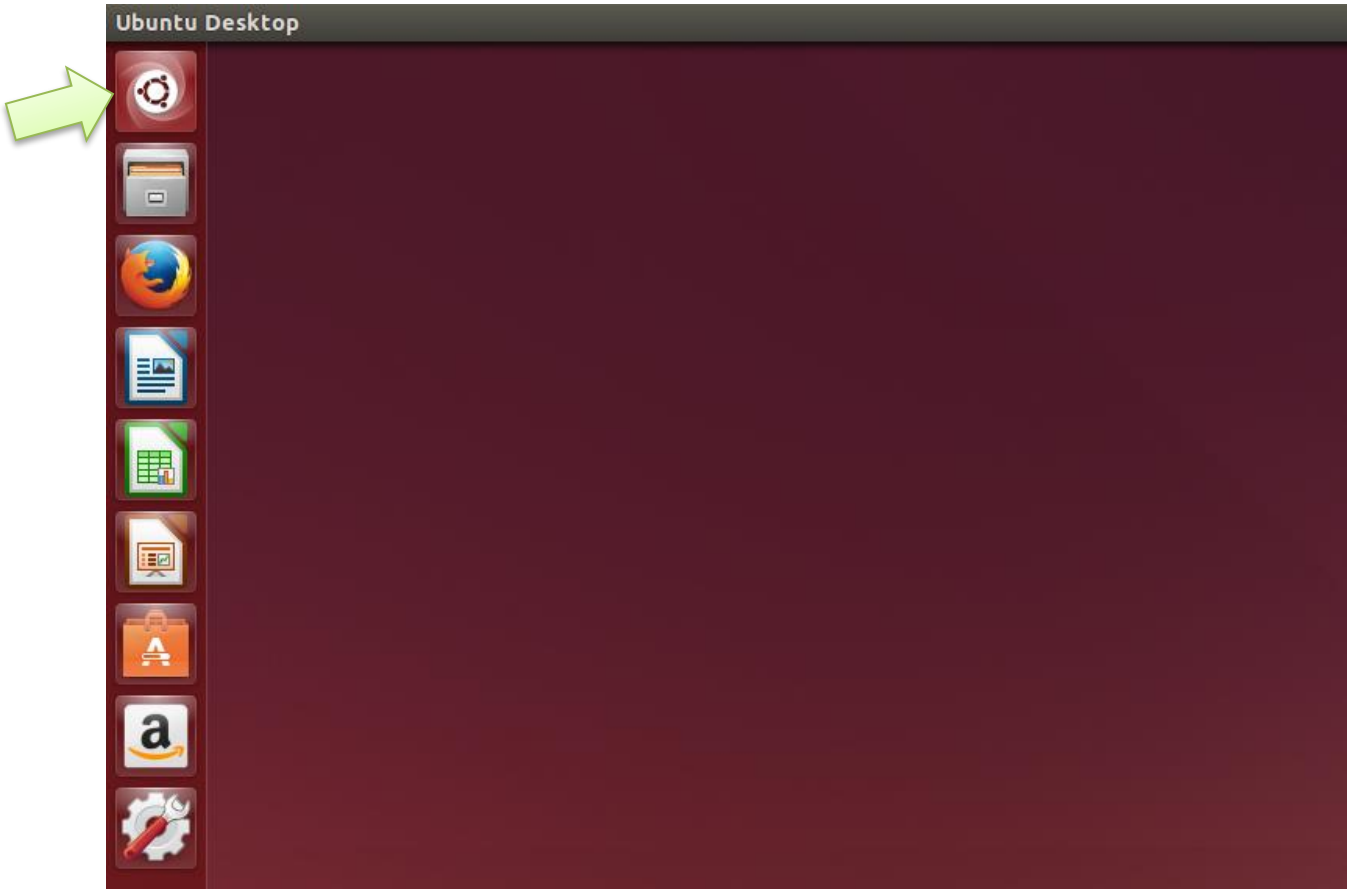


Operating Cosmos+ OpenSSD (Linux)

1. **Check device recognition**
2. **Create a partition**
3. **Check the created partition**
4. **Format the partition**
5. **Create a mount point**
6. **Mount the partition**
7. **Check the mounted partition**

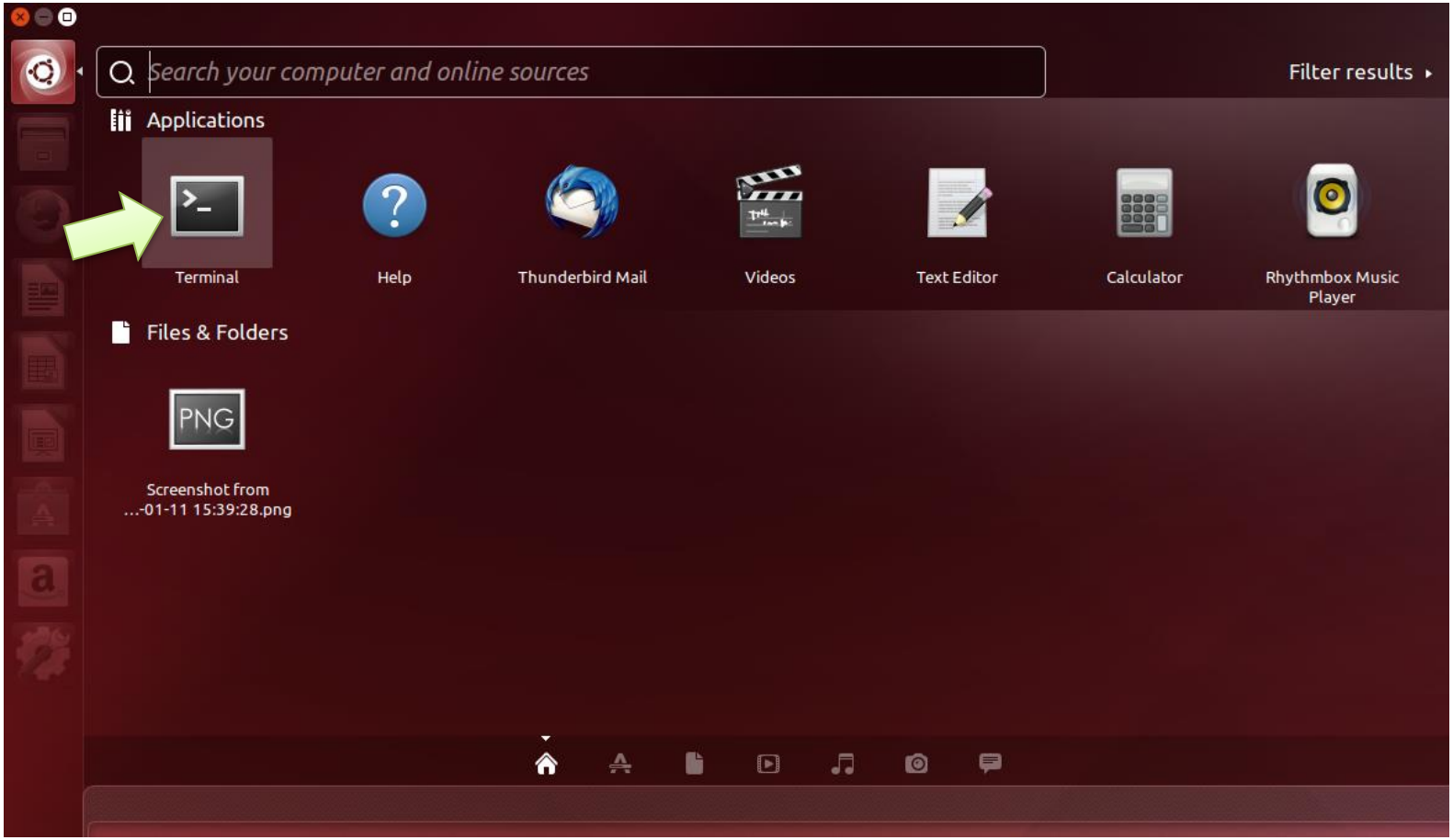
Open a Terminal (1 / 2)

- **Click the pointed icon**



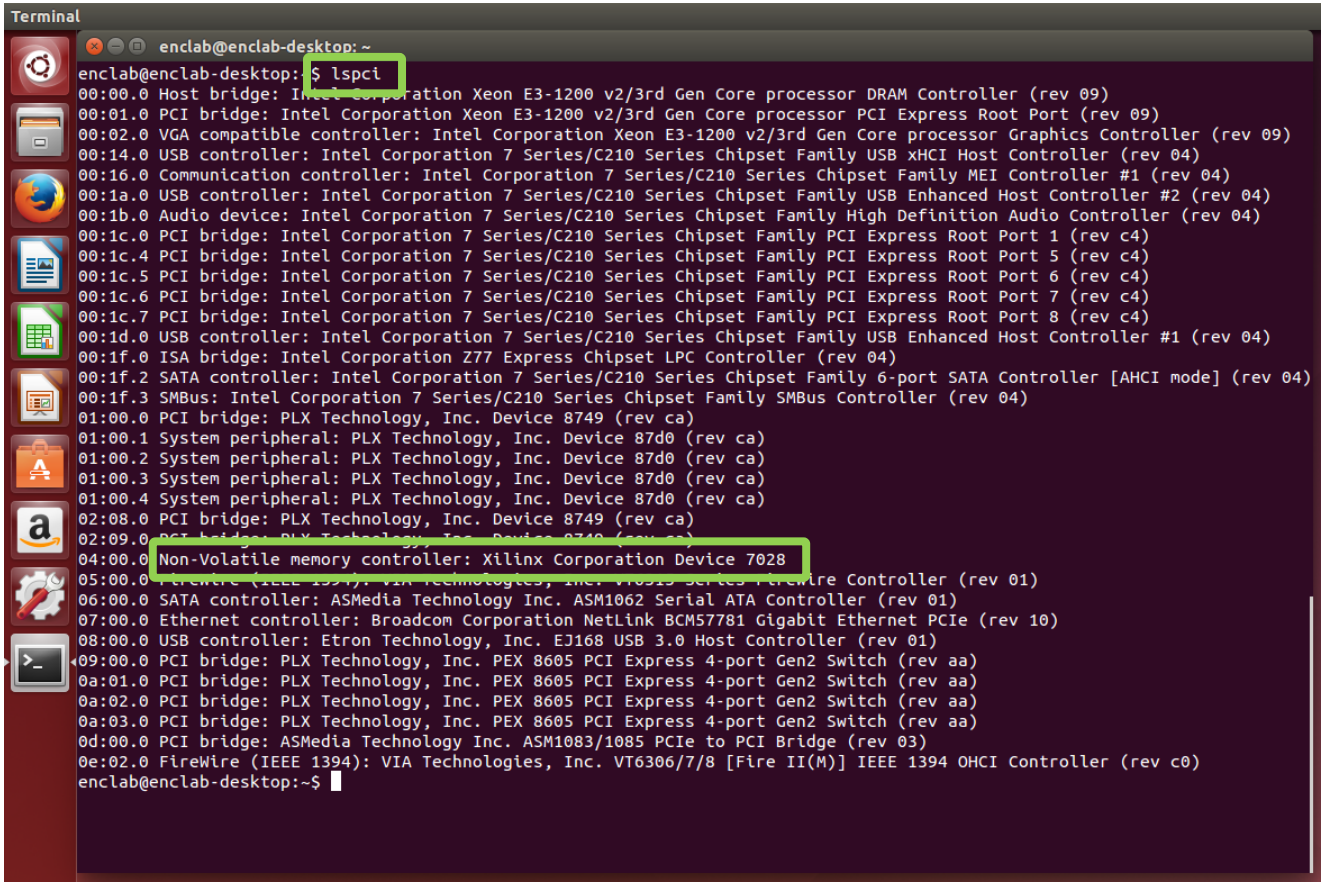
Open a Terminal (2 / 2)

- **Click the terminal icon**



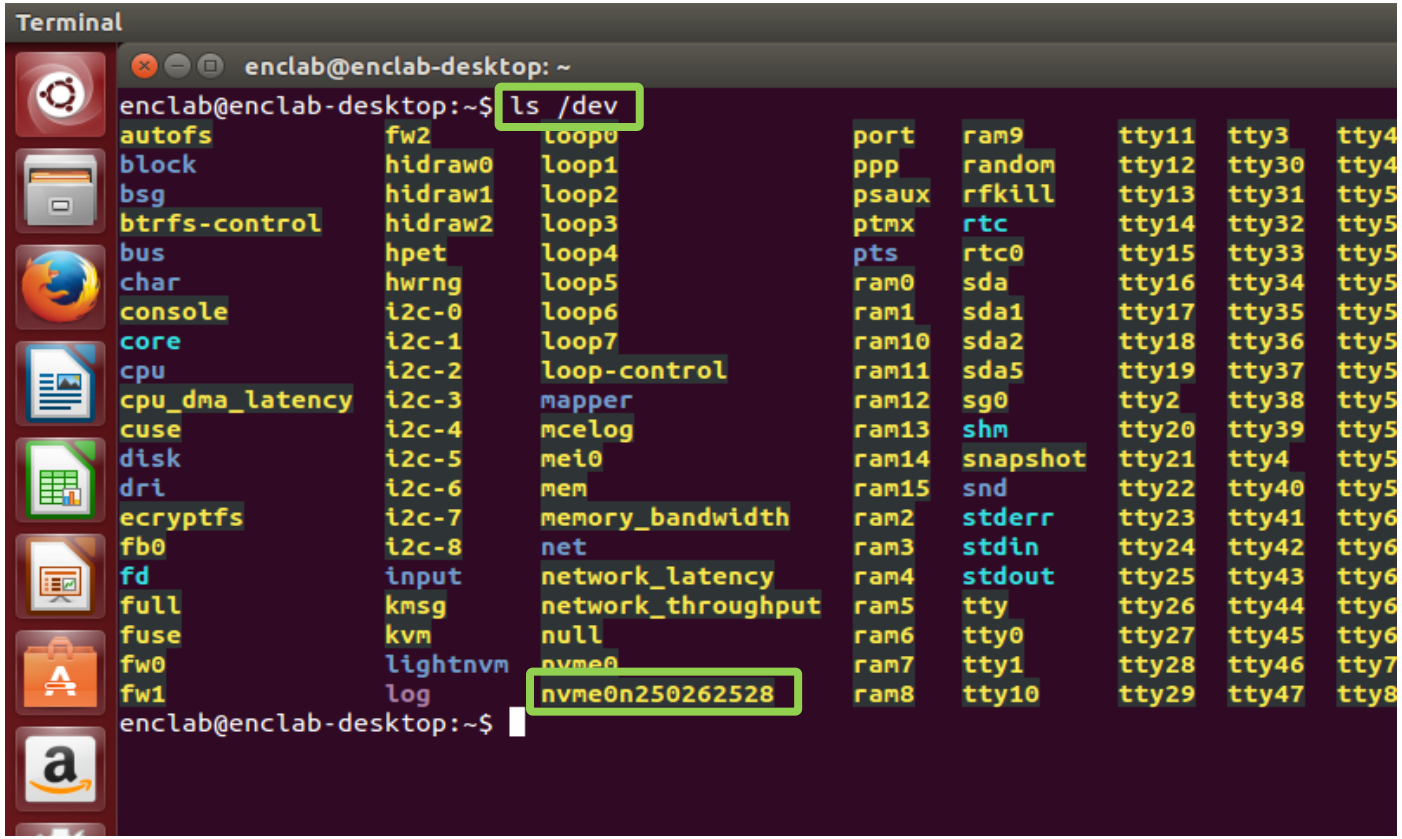
Check Device Recognition (1 / 2)

- **Types “lspci” -> press ENTER -> check “Non-Volatile memory controller: Xilinx Corporation Device 7028” on the PCI device list**



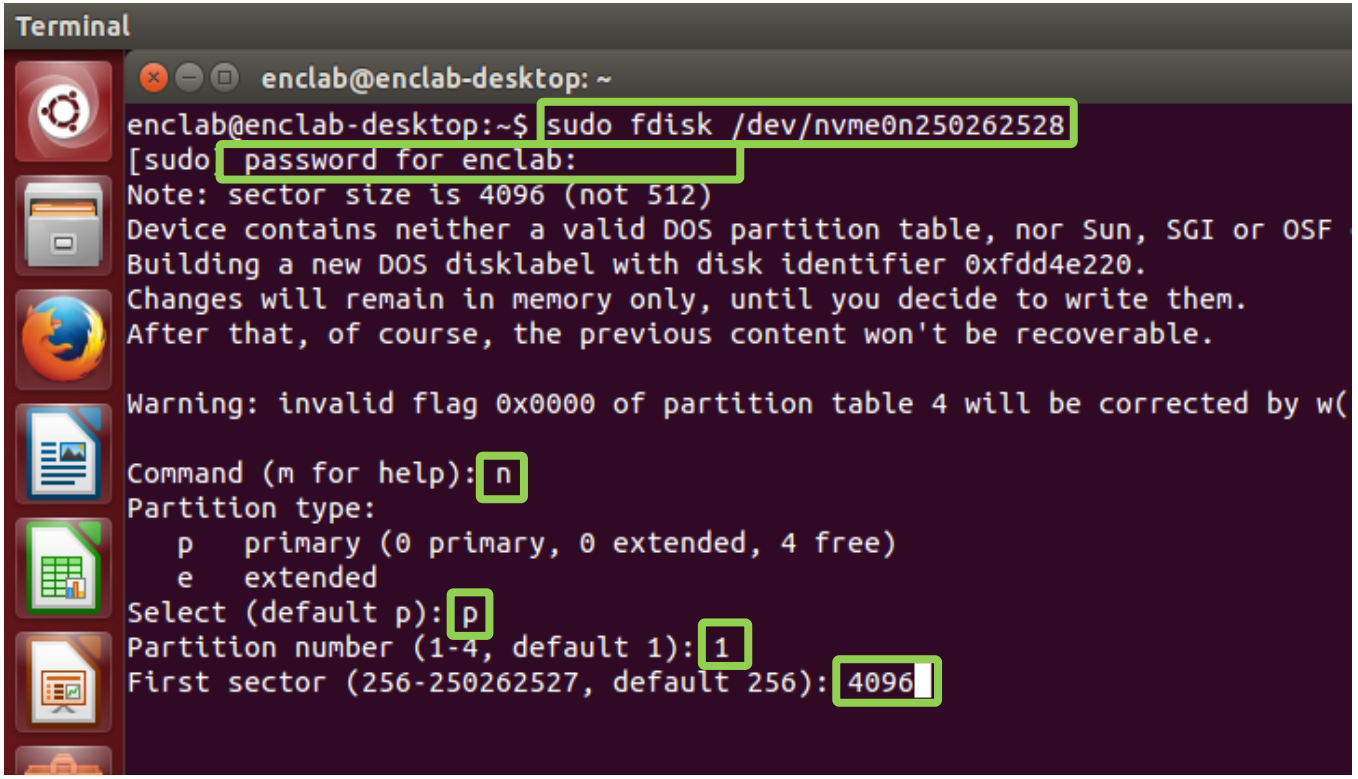
Check Device Recognition (2 / 2)

- **Types “ls /dev” -> press ENTER -> check “nvme0nxxxx” on the device list**



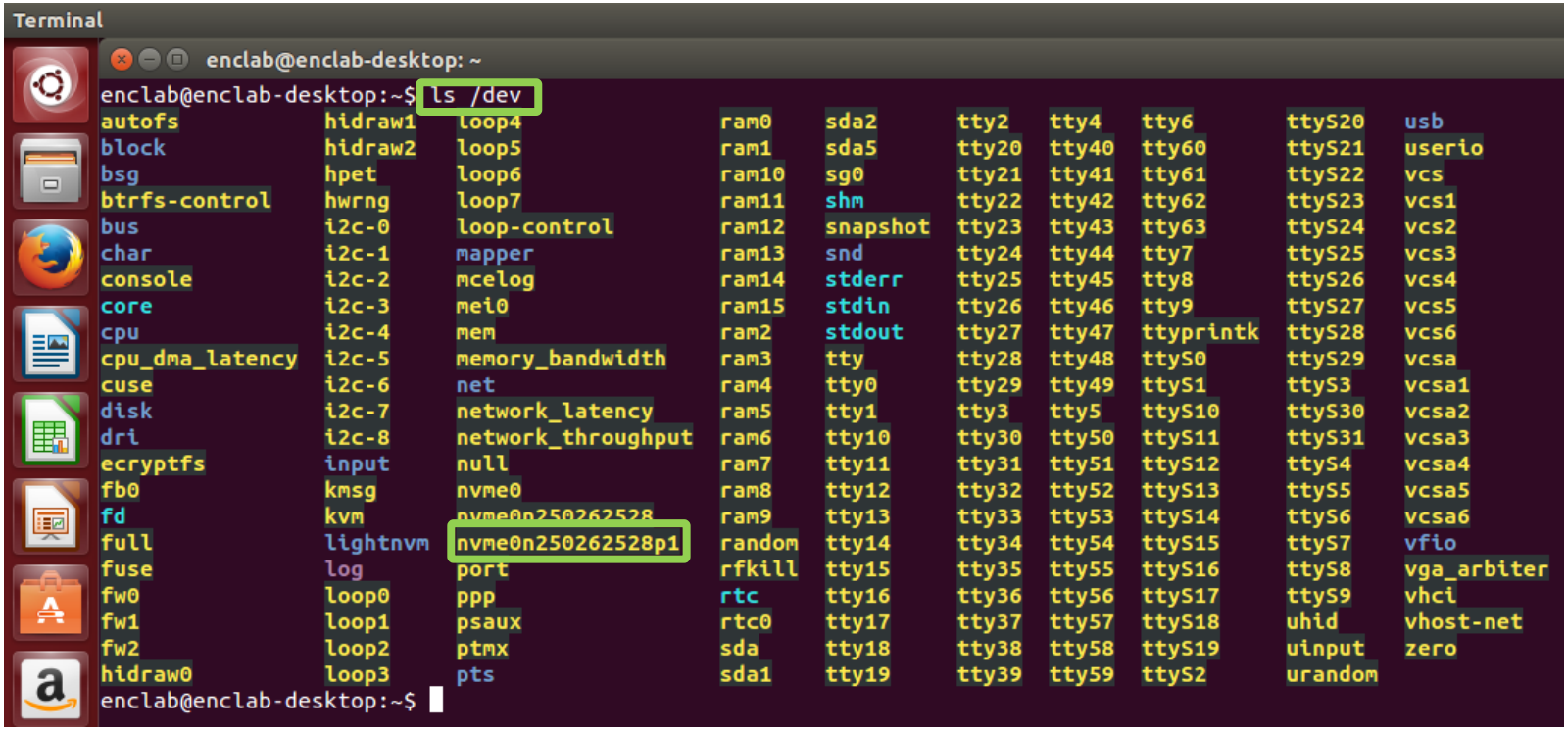
Create a Partition

- **Type “sudo fdisk /dev/nvme0nxxxx”, press ENTER -> type your password, press ENTER -> type “n”, press ENTER -> type “p”, press ENTER -> type “1”, press ENTER -> type “4096”, press ENTER**



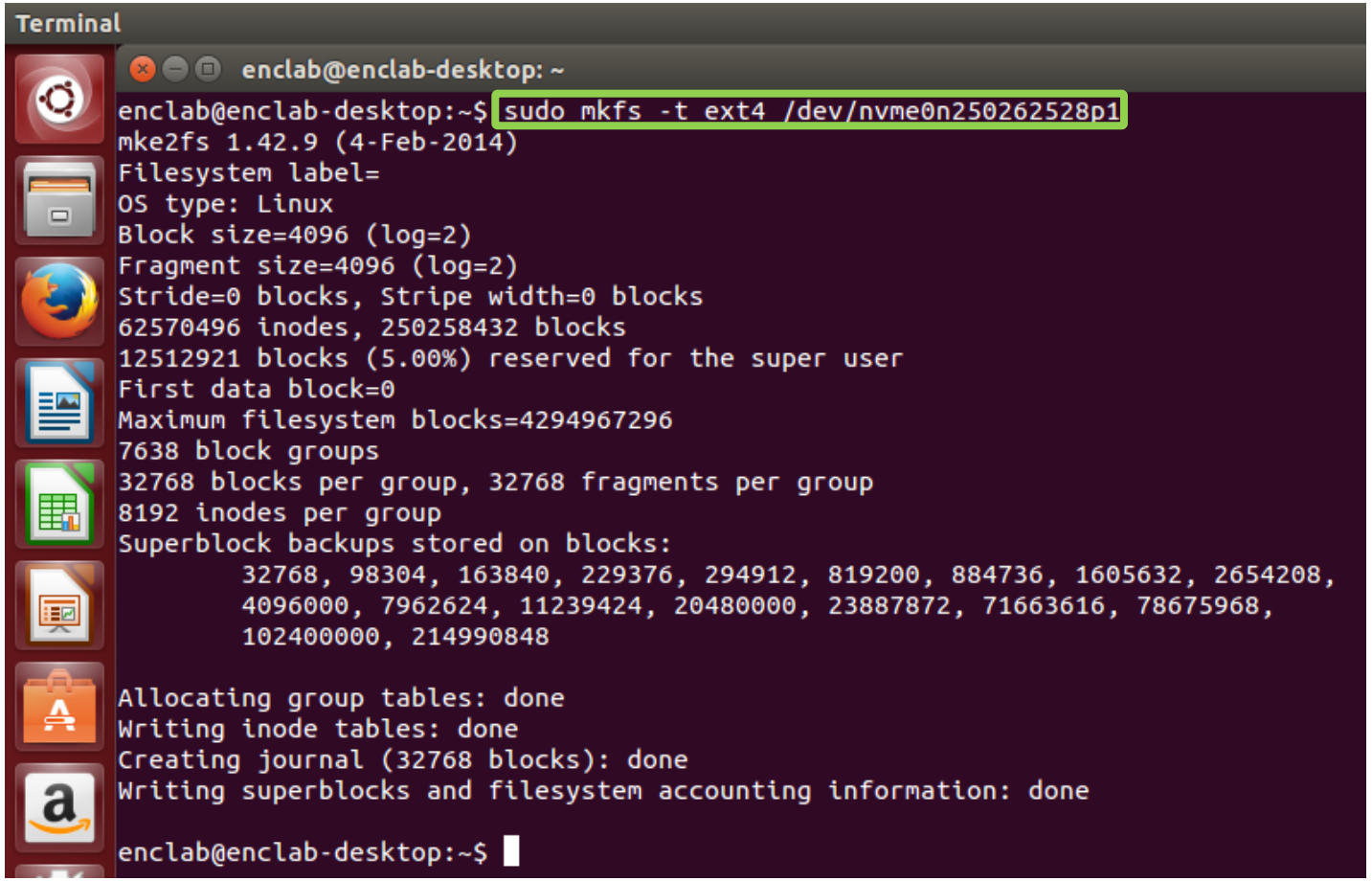
Check the Created Partition

- **Types “ls /dev” -> press ENTER -> check “nvme0nxxxxp1” on the device list**



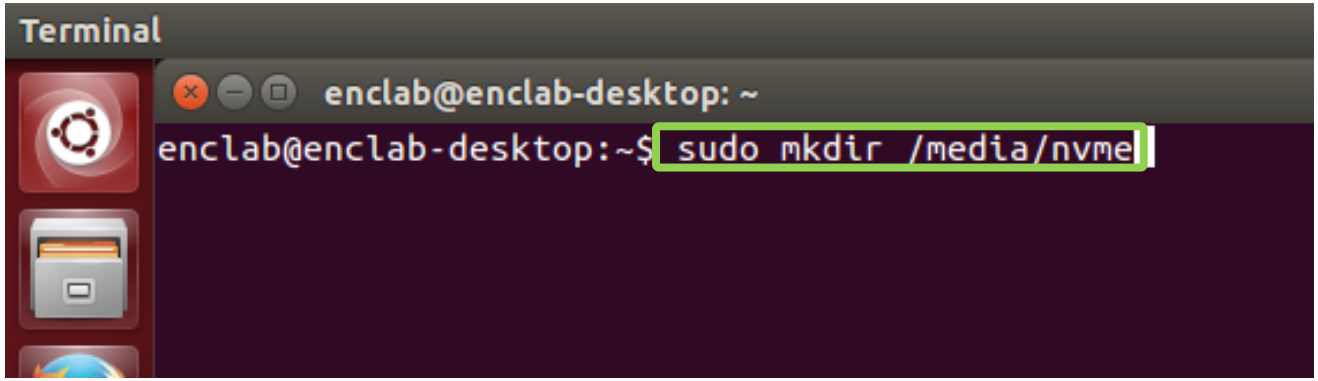
Format the Partition

- Type “mkfs -t ext4 / dev/nvme0nxxxxp1”, press ENTER



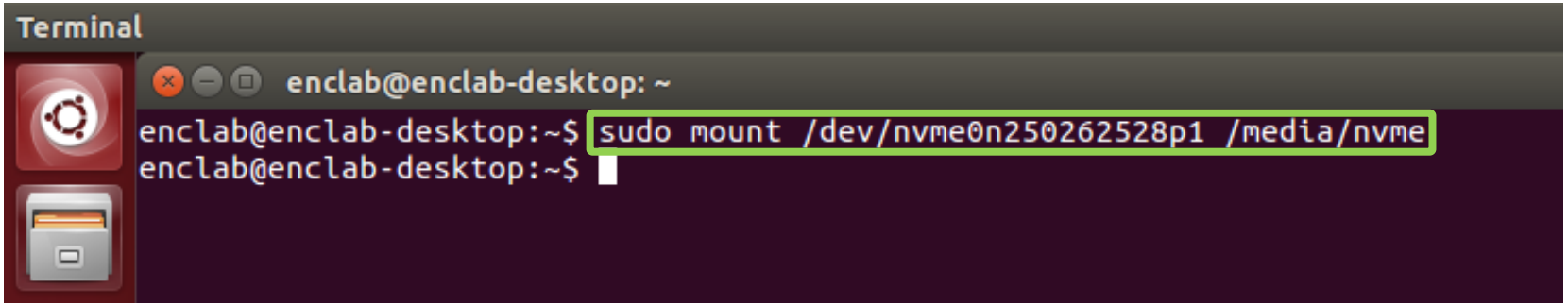
Create a Mount Point

- Type “sudo mkdir /media/nvme”, press ENTER



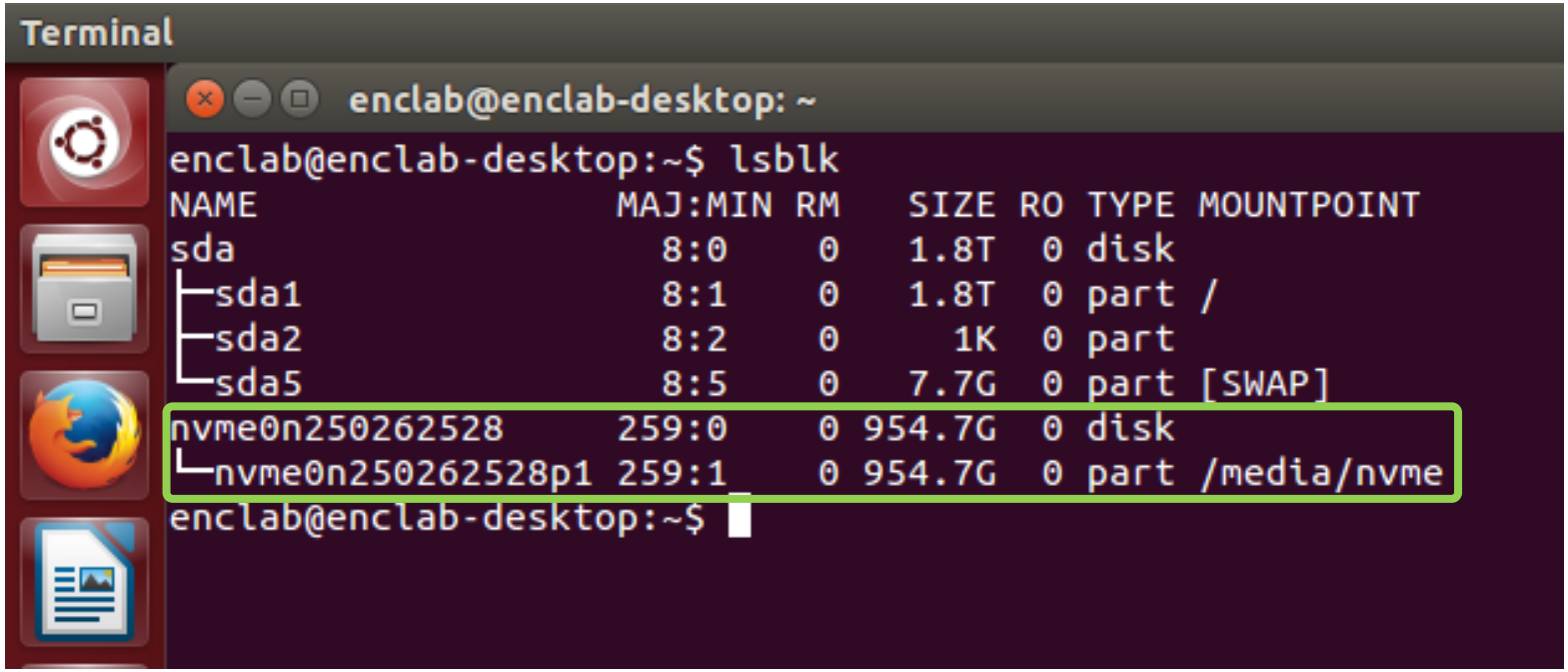
Mount the Partition

- **Type “sudo mount /dev/nvme0nxxxxp1 /media/nvme”, press ENTER**



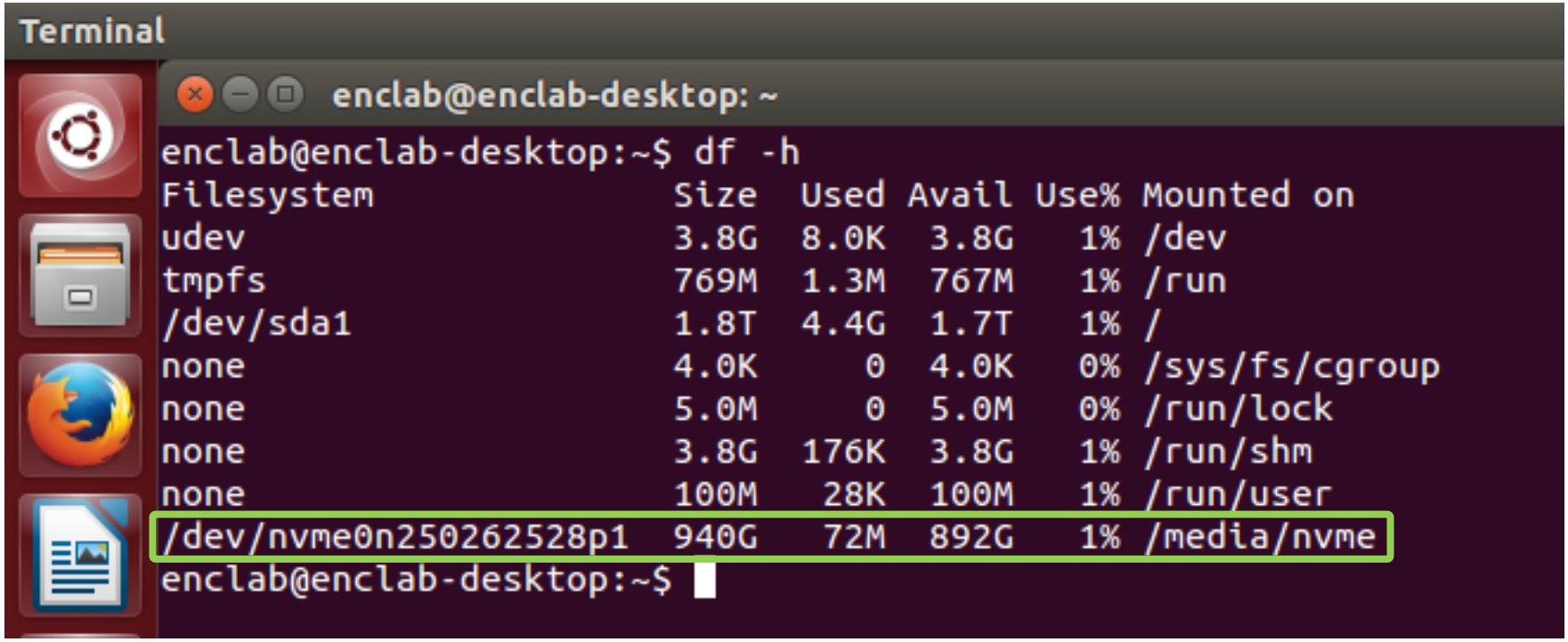
Check the Mounted Partition (1 / 2)

- **Type “lsblk”, press ENTER -> check the mounted partition on the bloc k device list**



Check the Mounted Partition (2 / 2)

- **Type “df -h”, press ENTER -> check the mounted partition on the stora ge list**

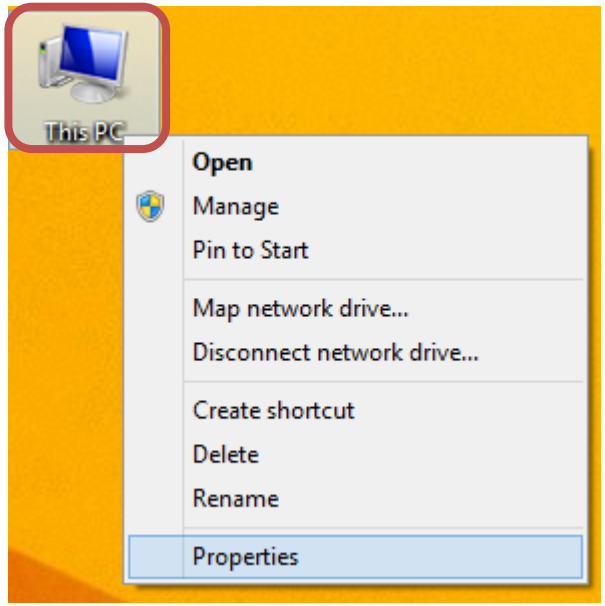


Operating Cosmos+ OpenSSD (Windows)

1. **Check device recognition**
2. **Create a partition**
3. **Format the partition**

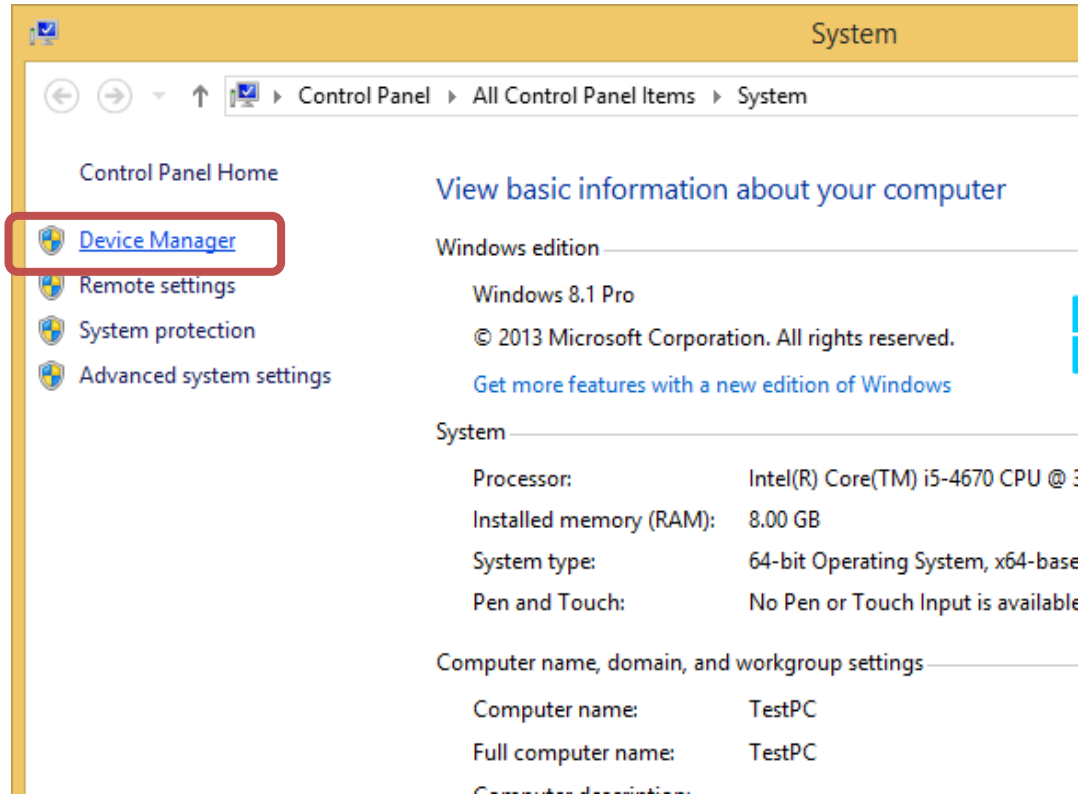
Check Device Recognition (1 / 3)

- **This PC → click left mouse button → click “Properties”**



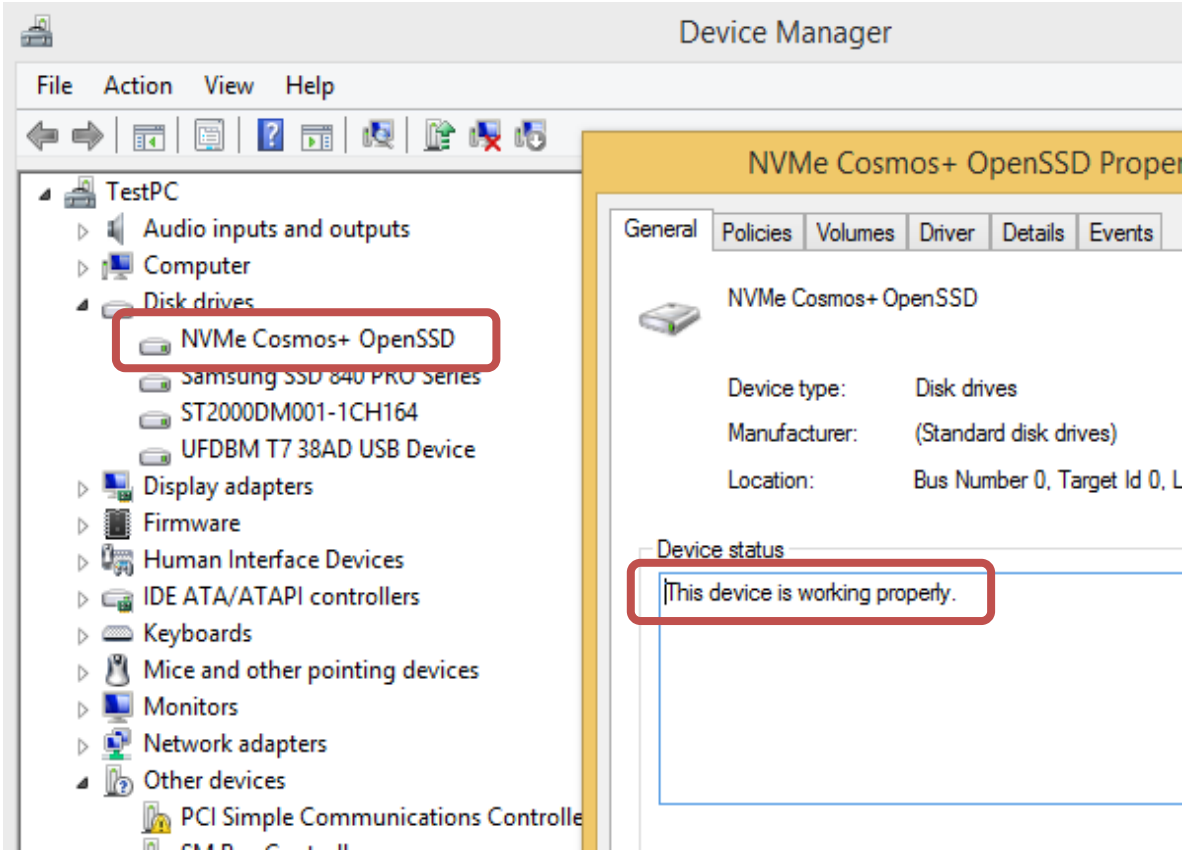
Check Device Recognition (2 / 3)

- **System → click “Device Manager”**



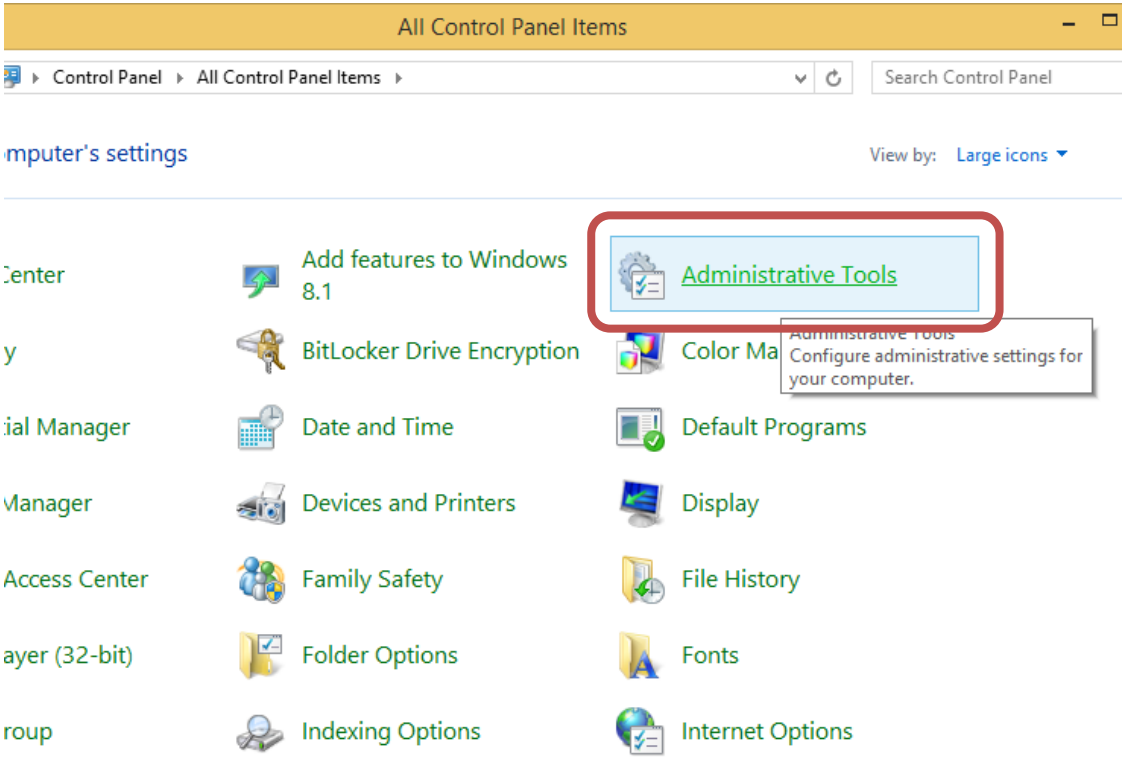
Check Device Recognition (3 / 3)

- **Disk drives → double-click “NVMe Cosmos+ OpenSSD”**



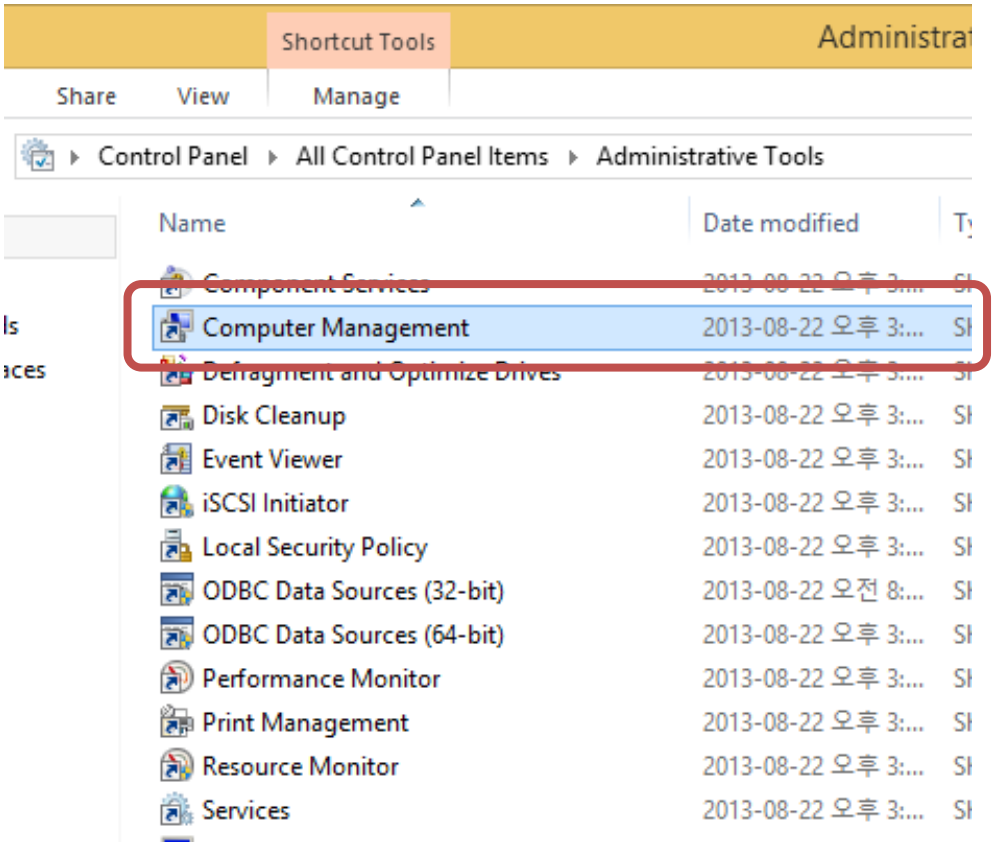
Create a Partition (1 / 5)

- **Control panel → click “Administrative Tools”**



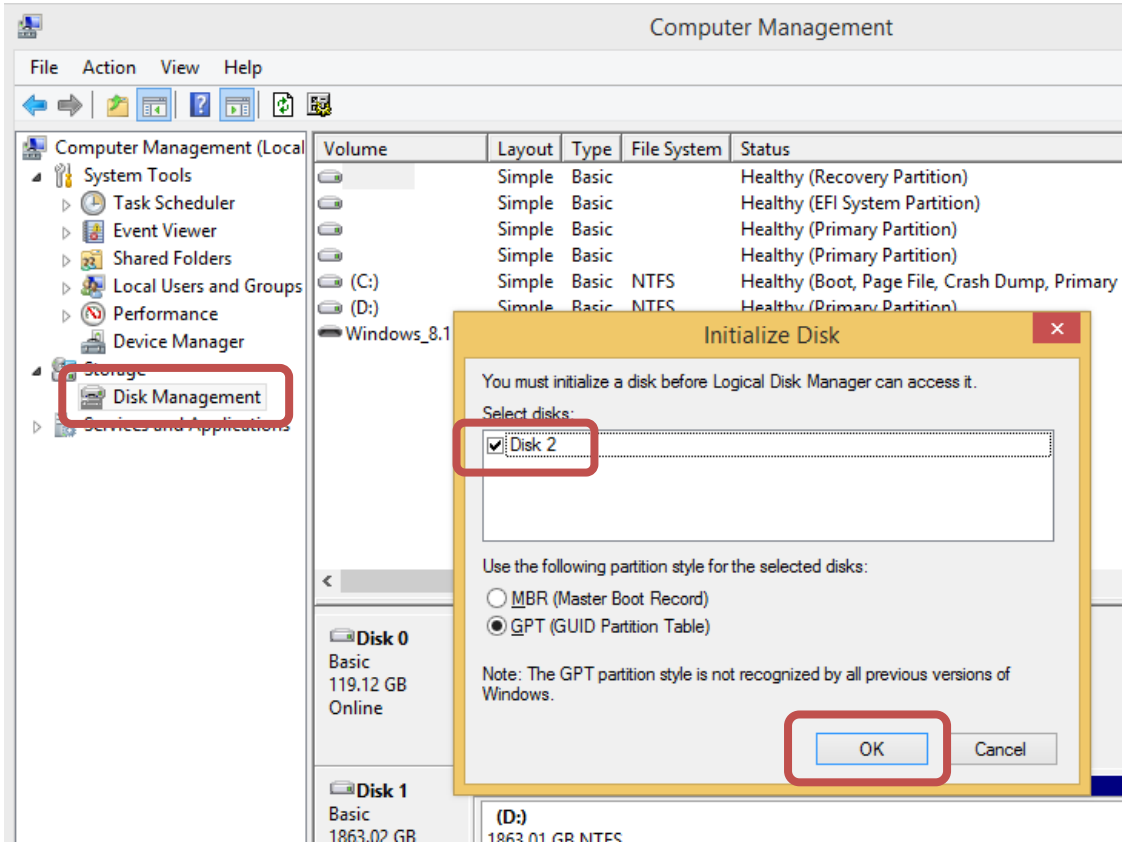
Create a Partition (2 / 5)

- **Administrative tools → double-click “Computer Management”**



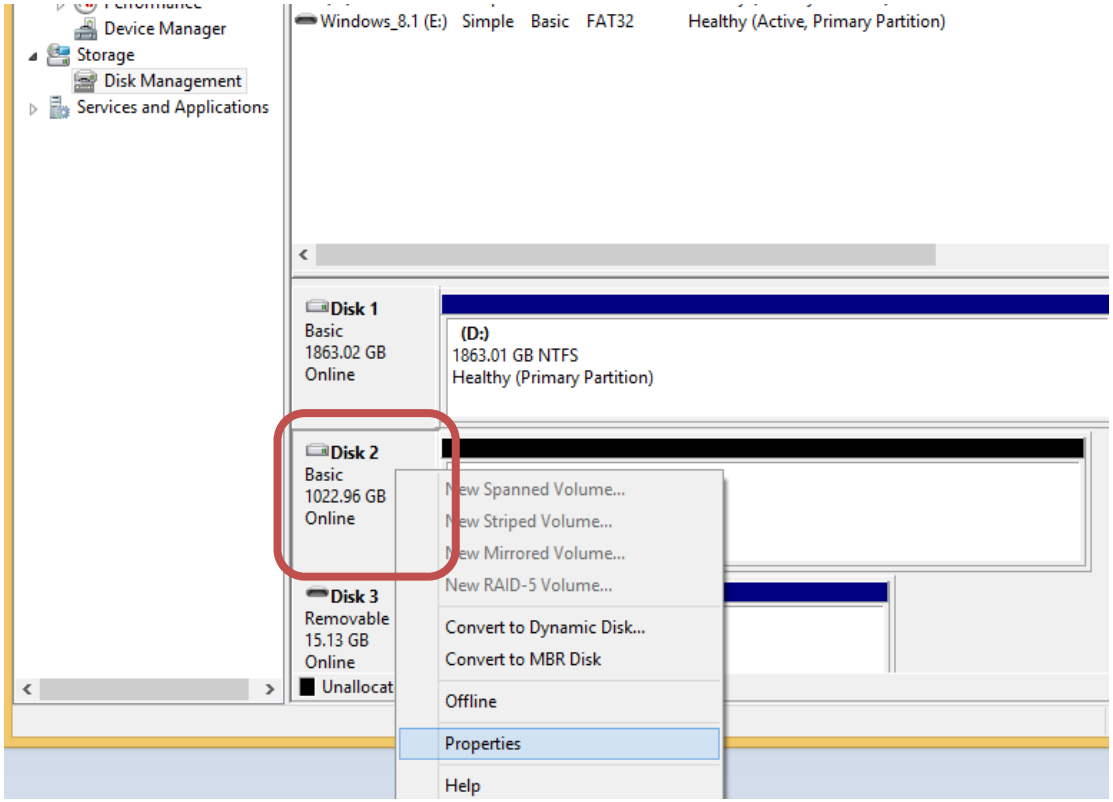
Create a Partition (3 / 5)

- **Computer management → click “Disk Management” → click “OK” to confirm disk initialization**



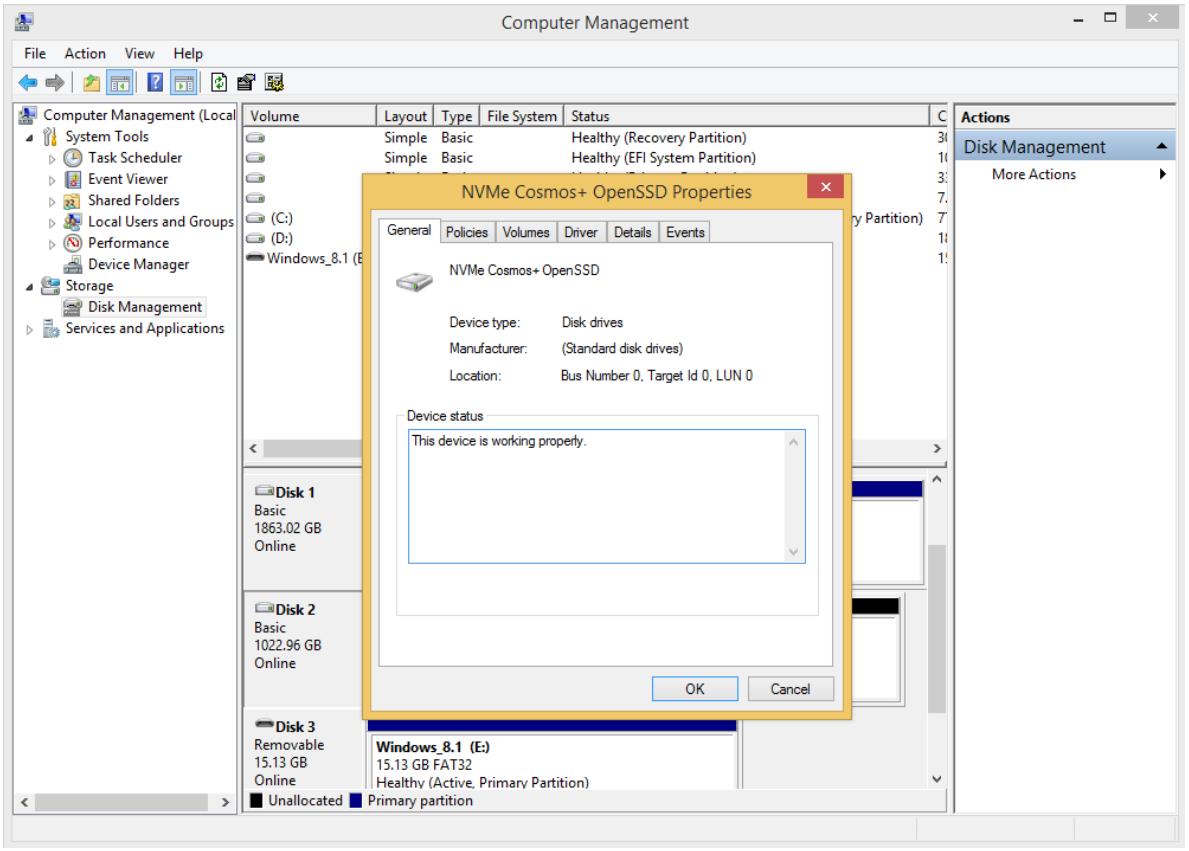
Create a Partition (4 / 5)

- **Click right mouse button on “Disk 2” which was shown in 3rd step → click “Properties”**



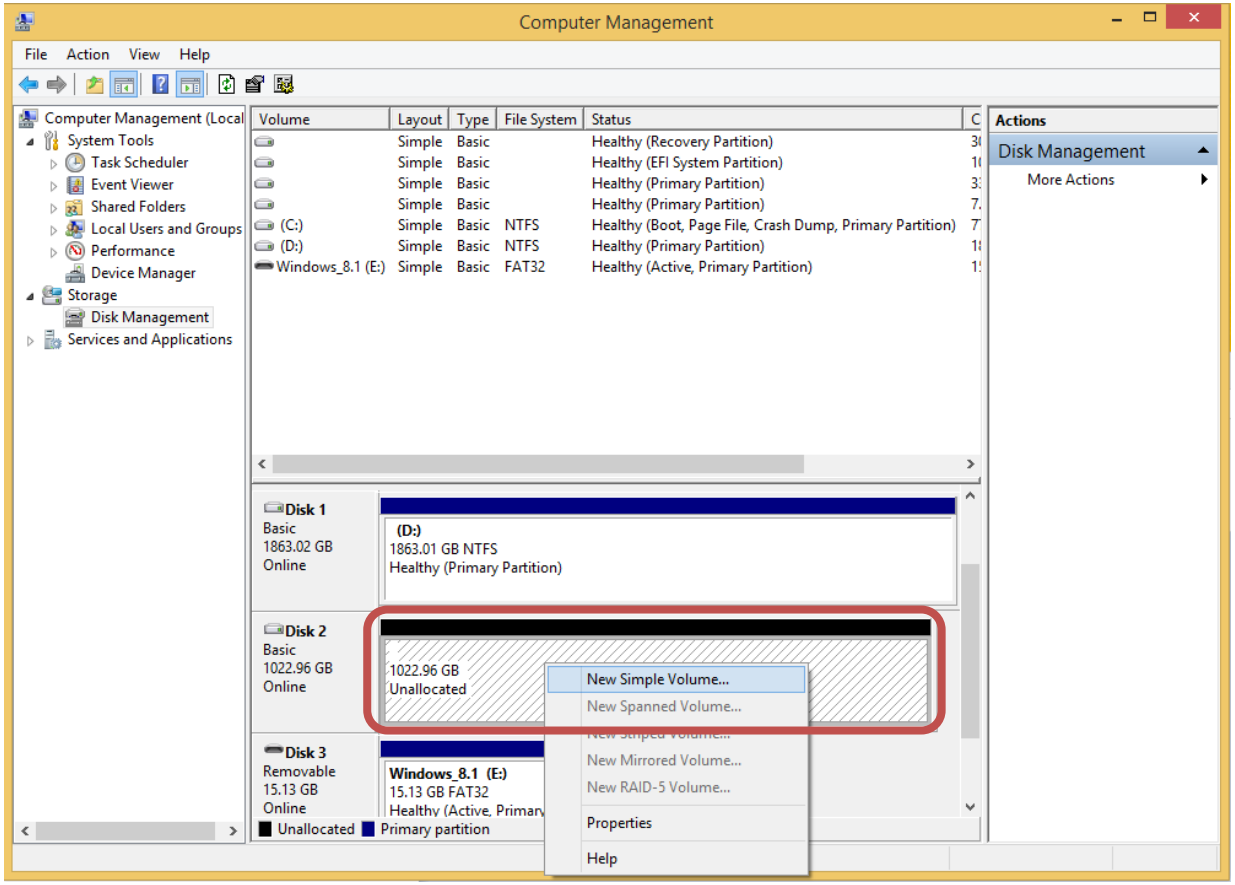
Create a Partition (5 / 5)

- **Make sure that the “Disk 2” is Cosmos+ OpenSSD before you proceed to the next step**



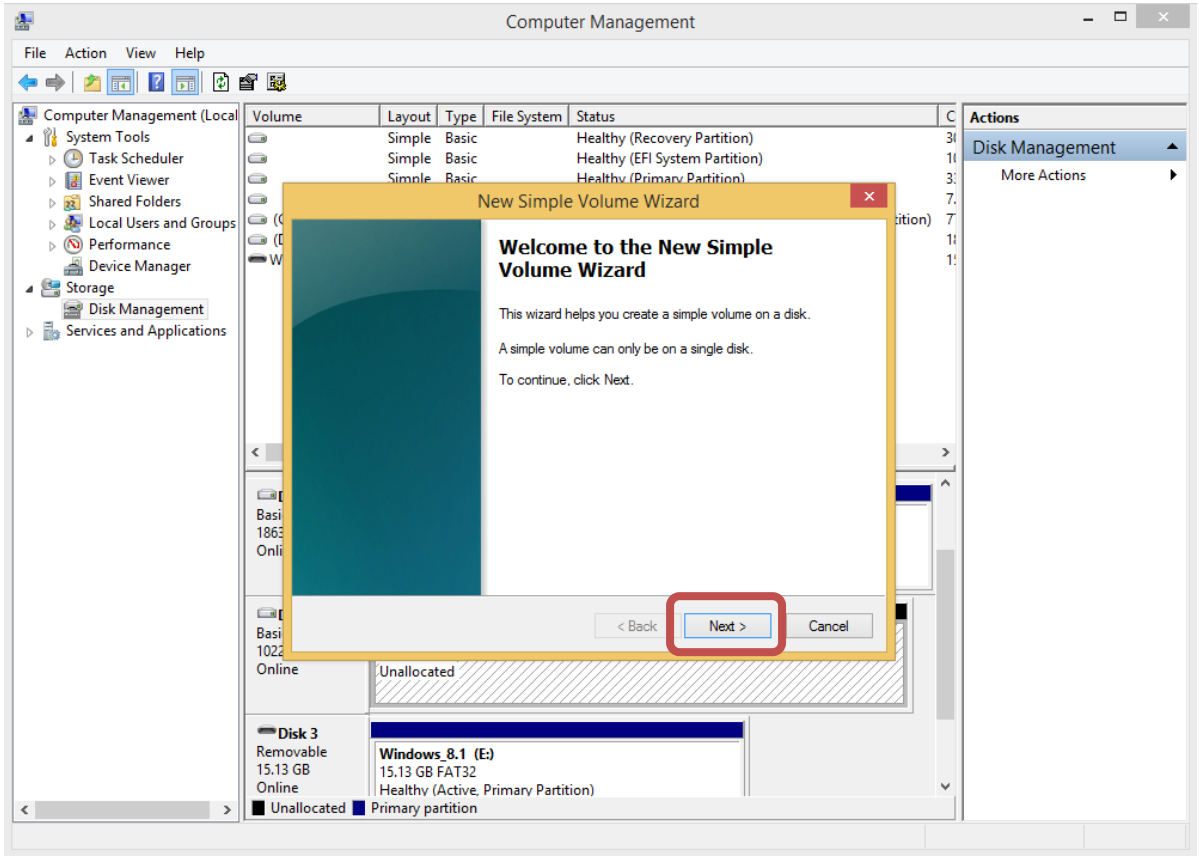
Format the Partition (1 / 8)

- **Click right mouse button on the right part of “Disk 2” → click “New Simple Volume”**



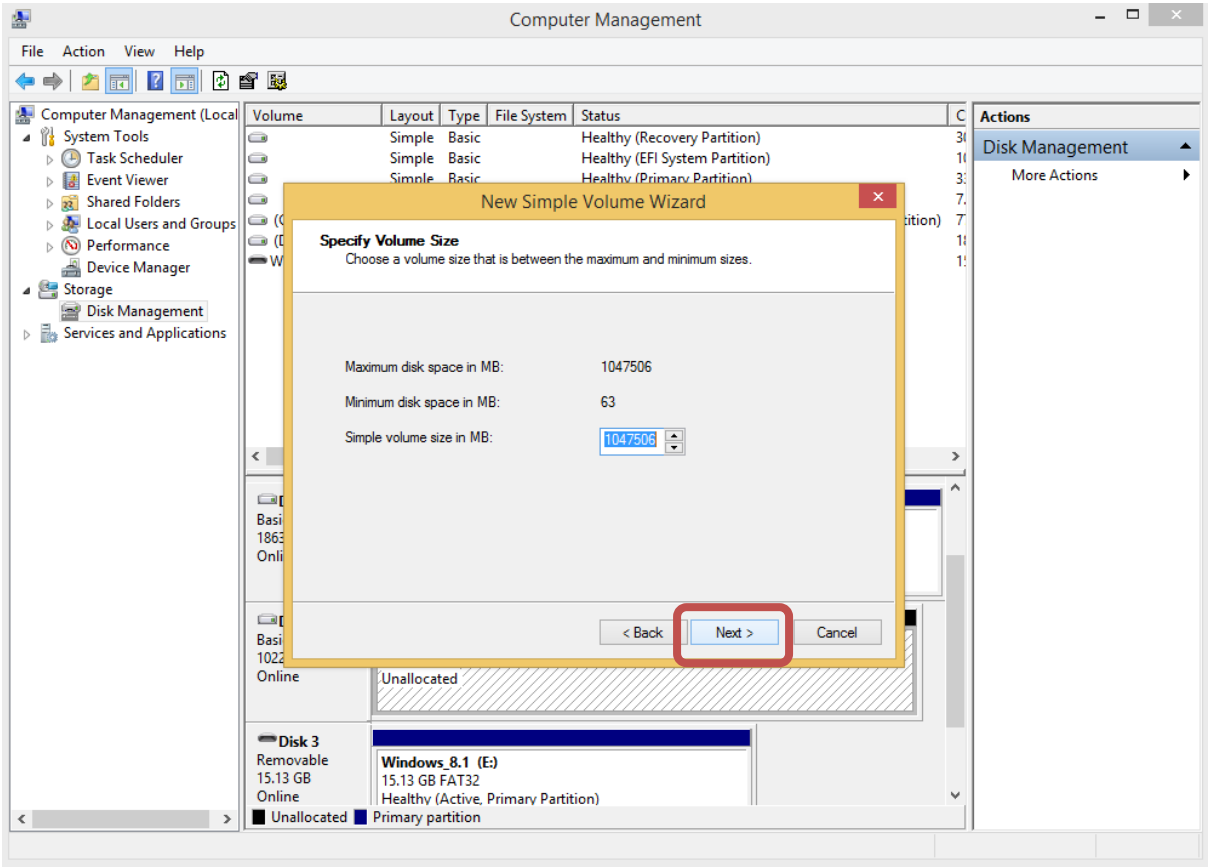
Format the Partition (2 / 8)

- **Click “Next”**



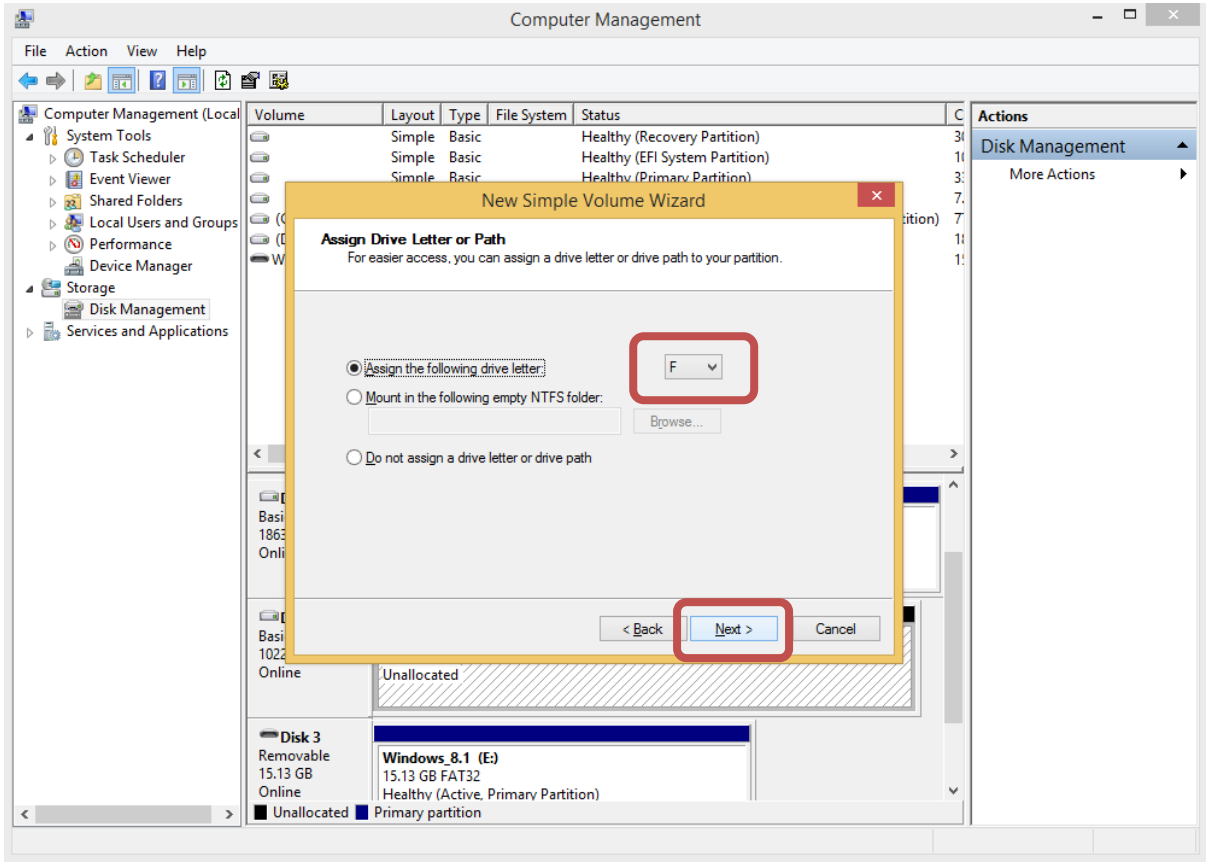
Format the Partition (3 / 8)

- **Click “Next”**



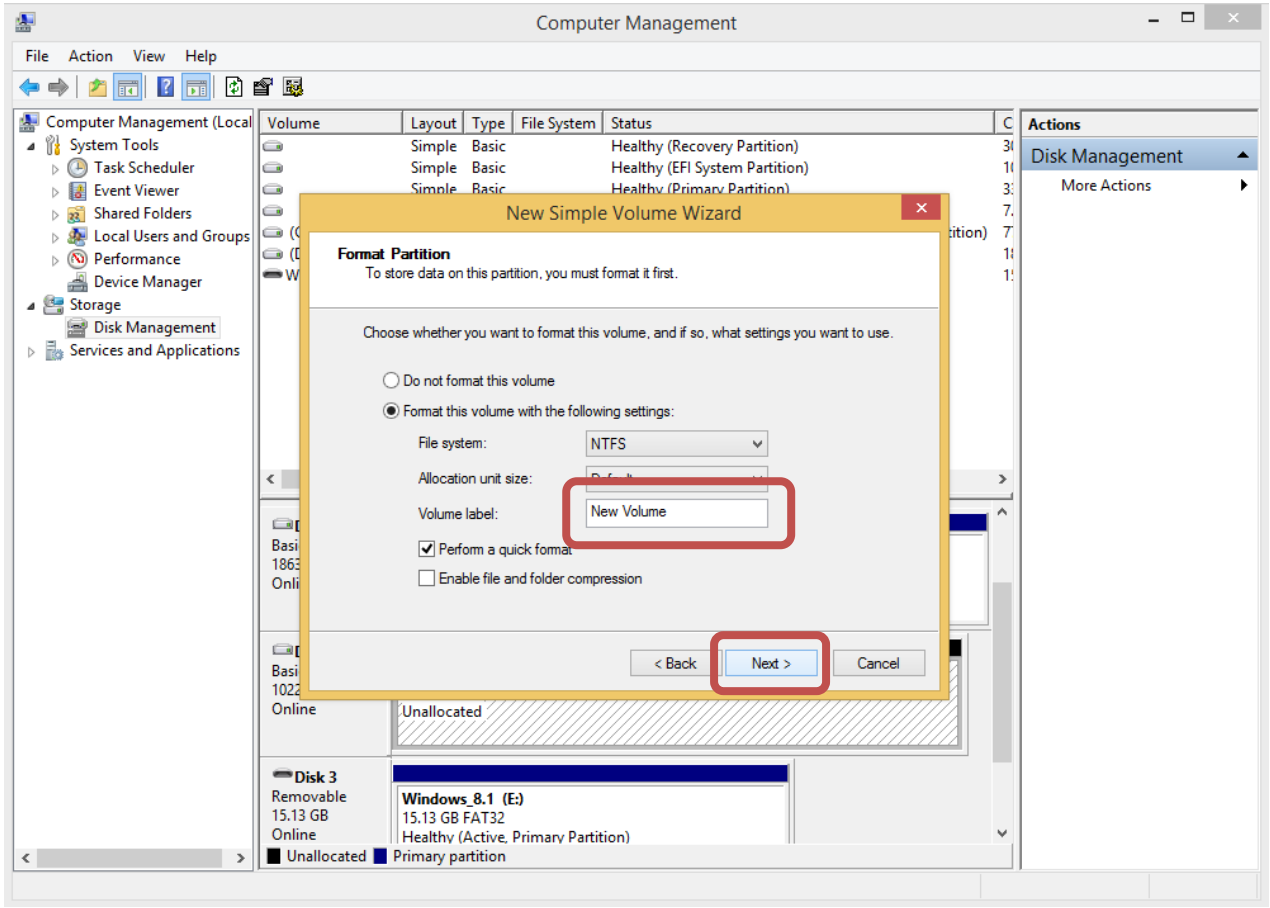
Format the Partition (4 / 8)

- **Select desired drive letter → Click “Next”**



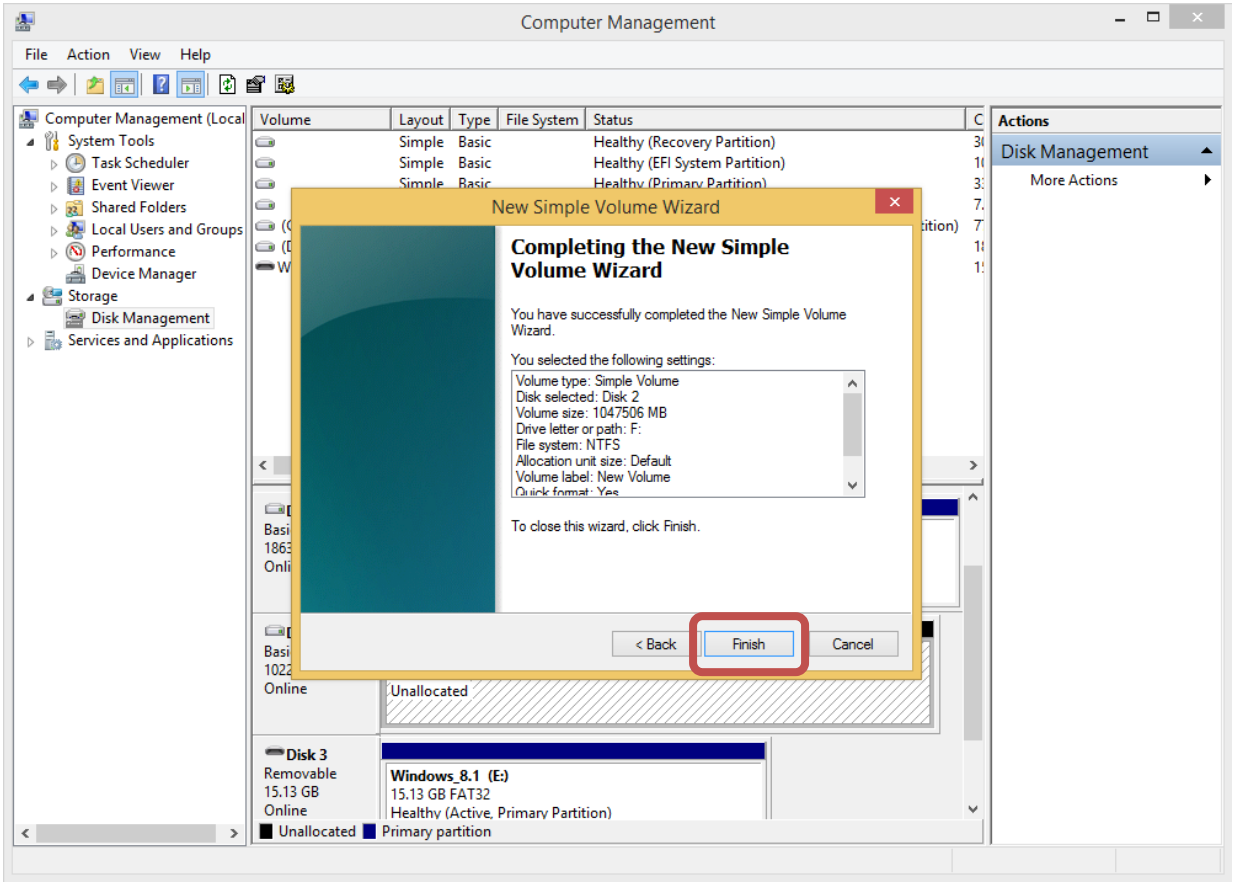
Format the Partition (5 / 8)

- **Type desired volume label → Click “Next”**



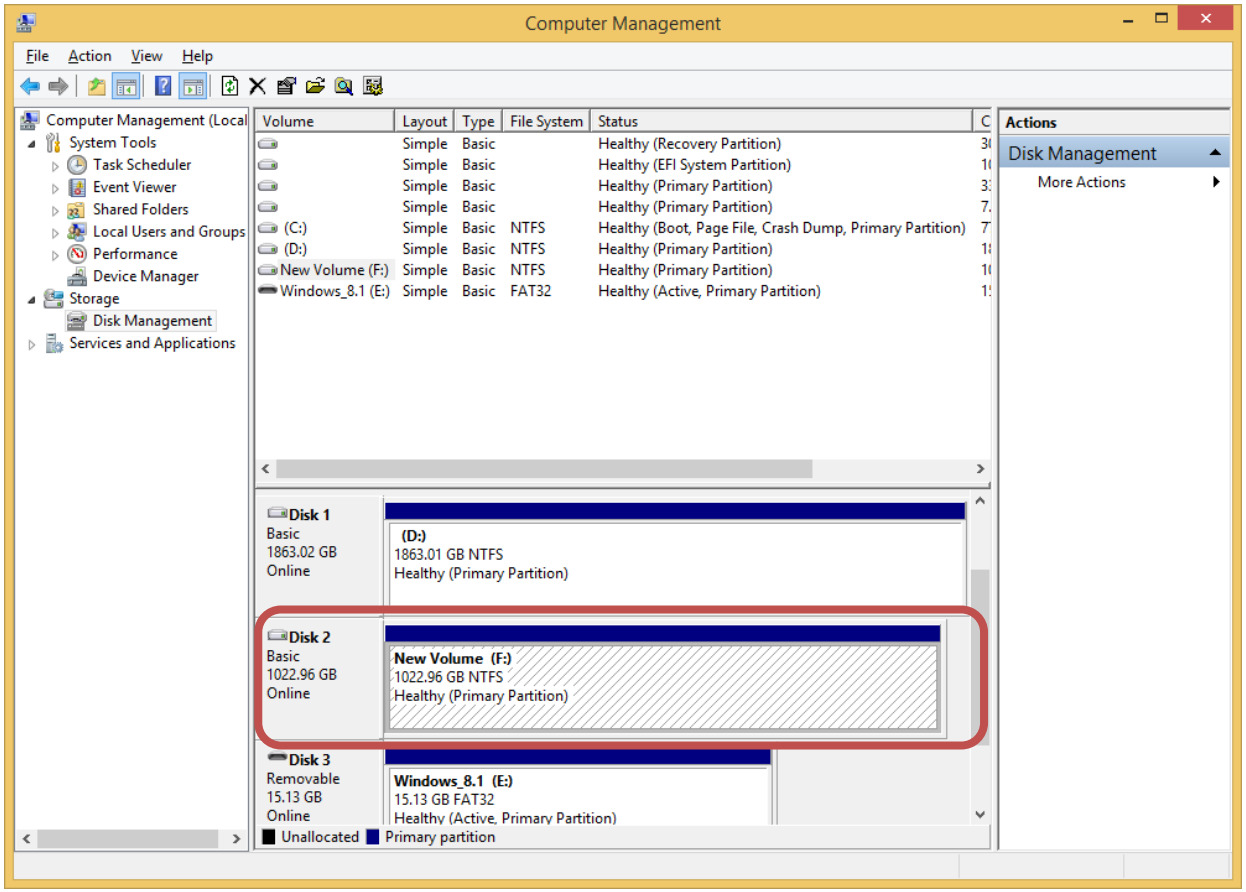
Format the Partition (6 / 8)

- **Click “Finish”**



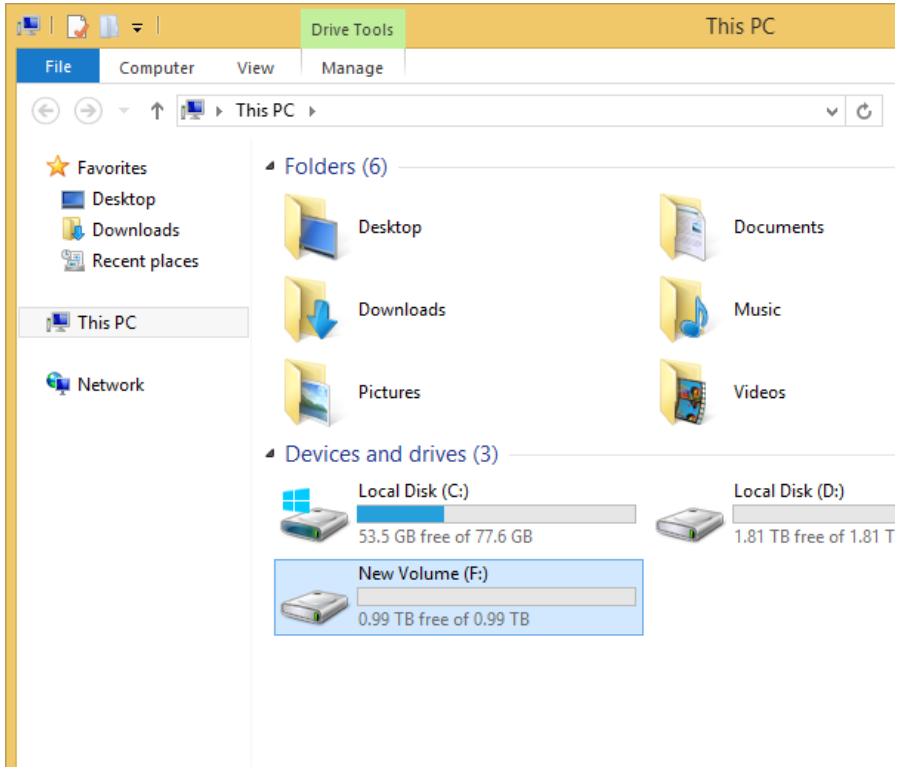
Format the Partition (7 / 8)

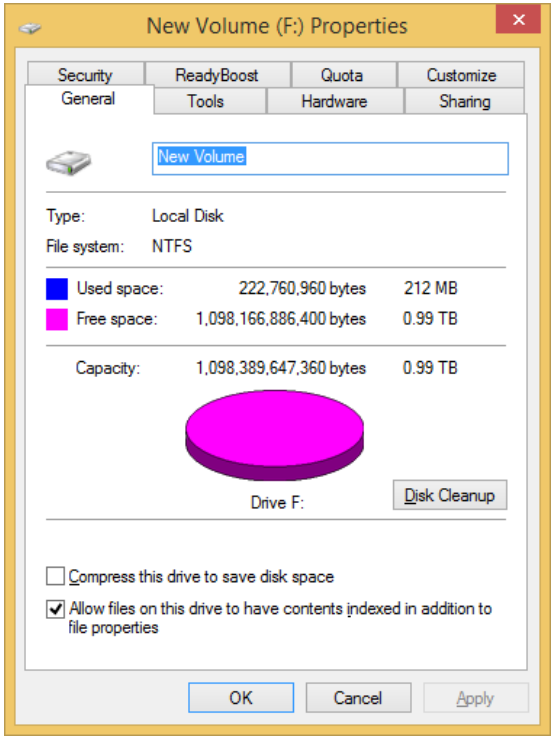
- **Formatting is now finished**



Format the Partition (8 / 8)

- **Now you can find the formatted Cosmos+ OpenSSD at “This PC”**



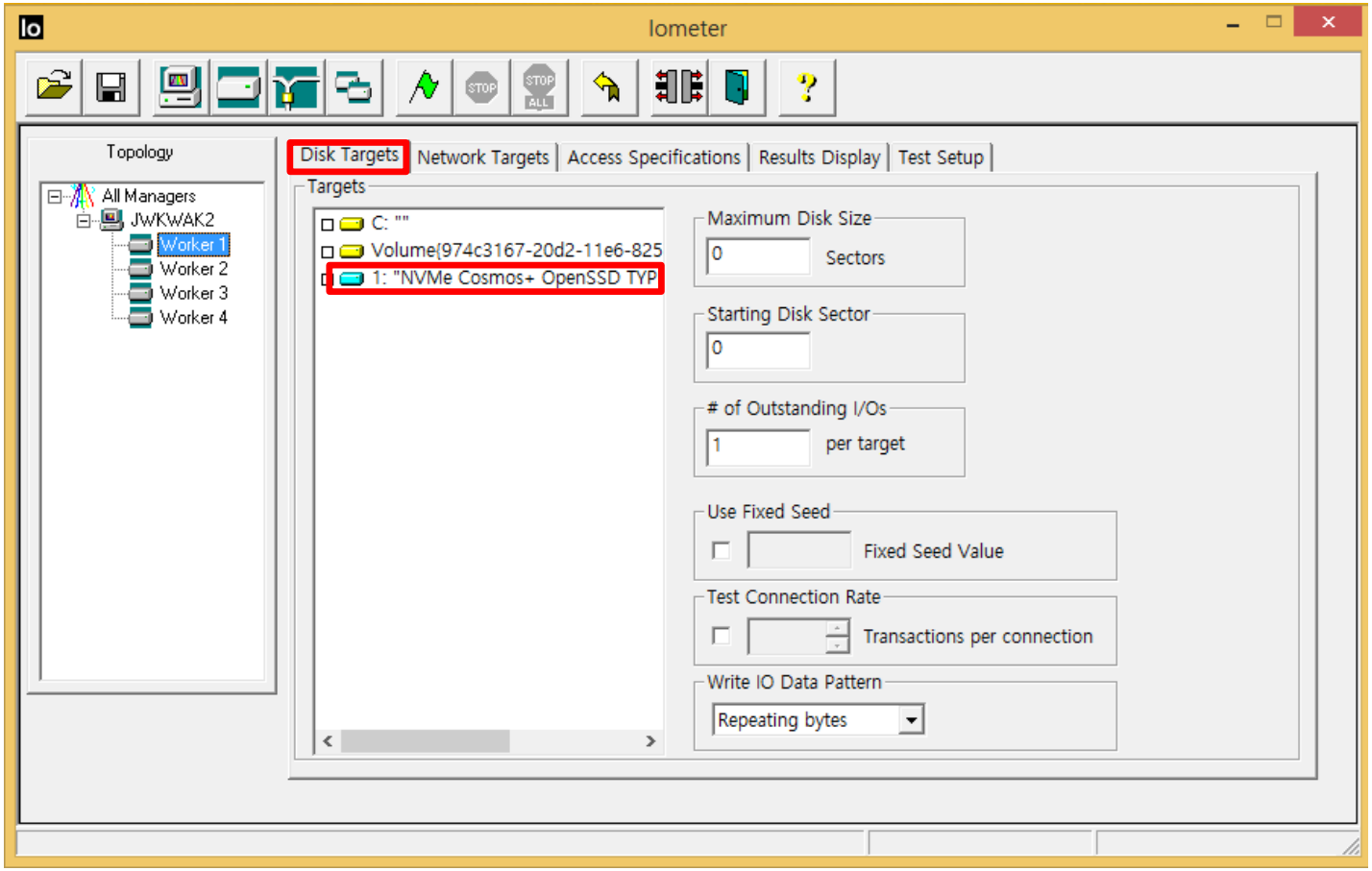


Evaluating Cosmos+ OpenSSD Performance

1. **Install benchmark application (Iometer)**
2. **Disconnect workers except one worker**
3. **Generate a access specification**
4. **Set the sufficient number of outstanding I/Os**
5. **Assign a access specification**
6. **Run an evaluation**
7. **Check evaluation results**

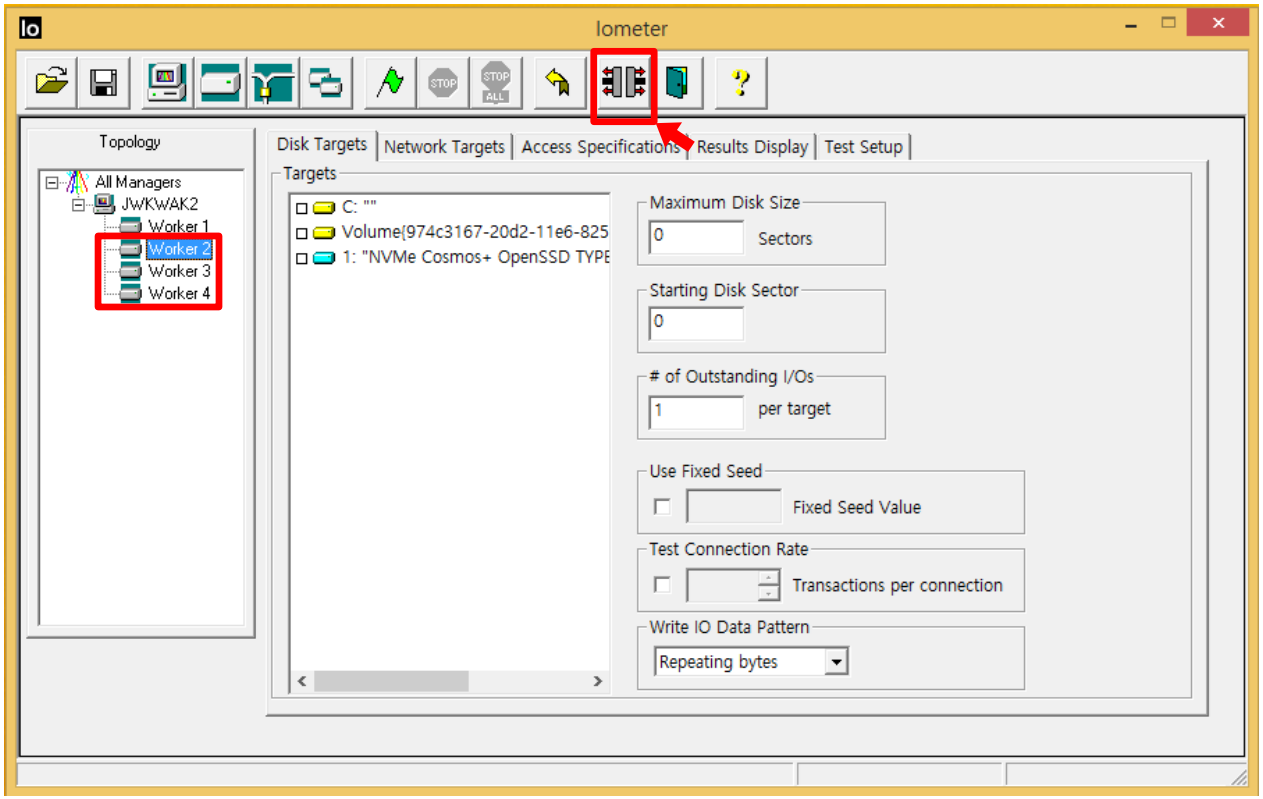
Install Benchmark Application

- **Iometer 1.1.0 (http://www.iometer.org/doc/downloads.html)**
  - Cosmos+ OpenSSD is recognized as NVMe storage device



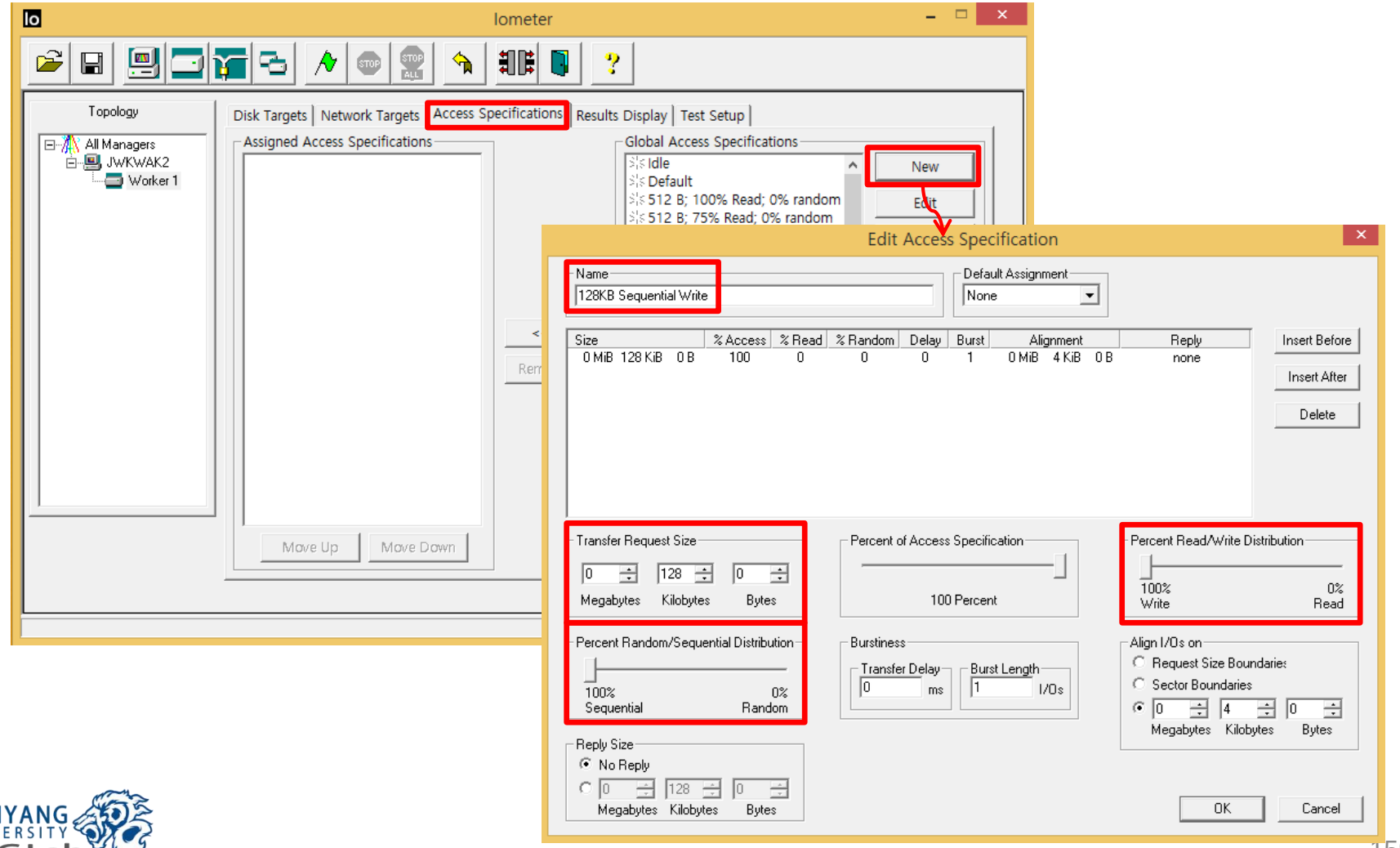
Disconnect workers except one worker

- **Avoid Workers having a same access specifications**
  - Workers can access the same logical address almost the same time
    - Increase the data buffer hit ratio
  - Performance can be measured higher than real performance



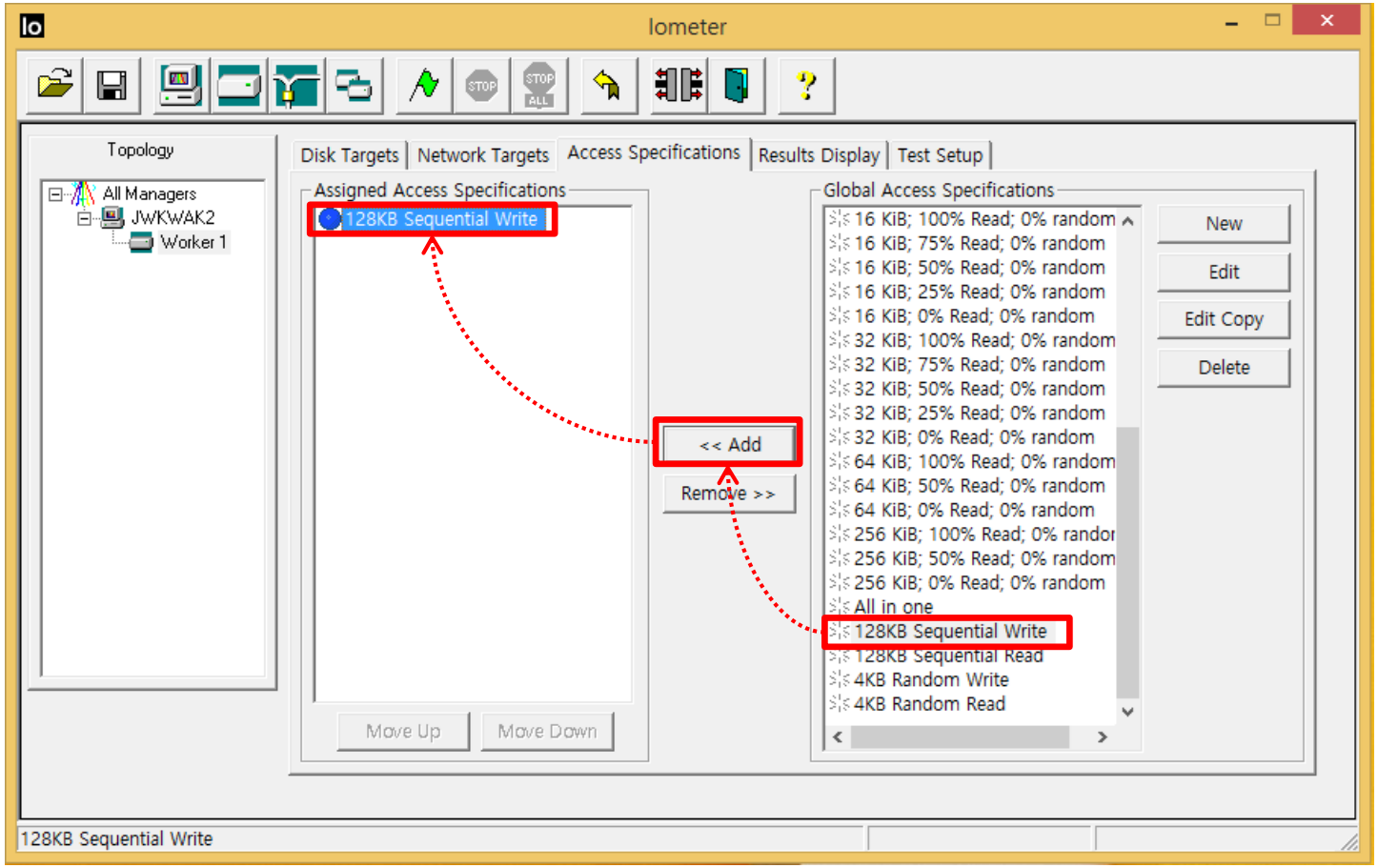
Generate a access specification

- **User can define a access specification**



Assign a access specification

- **Select a desired access specification and click “Add” button**

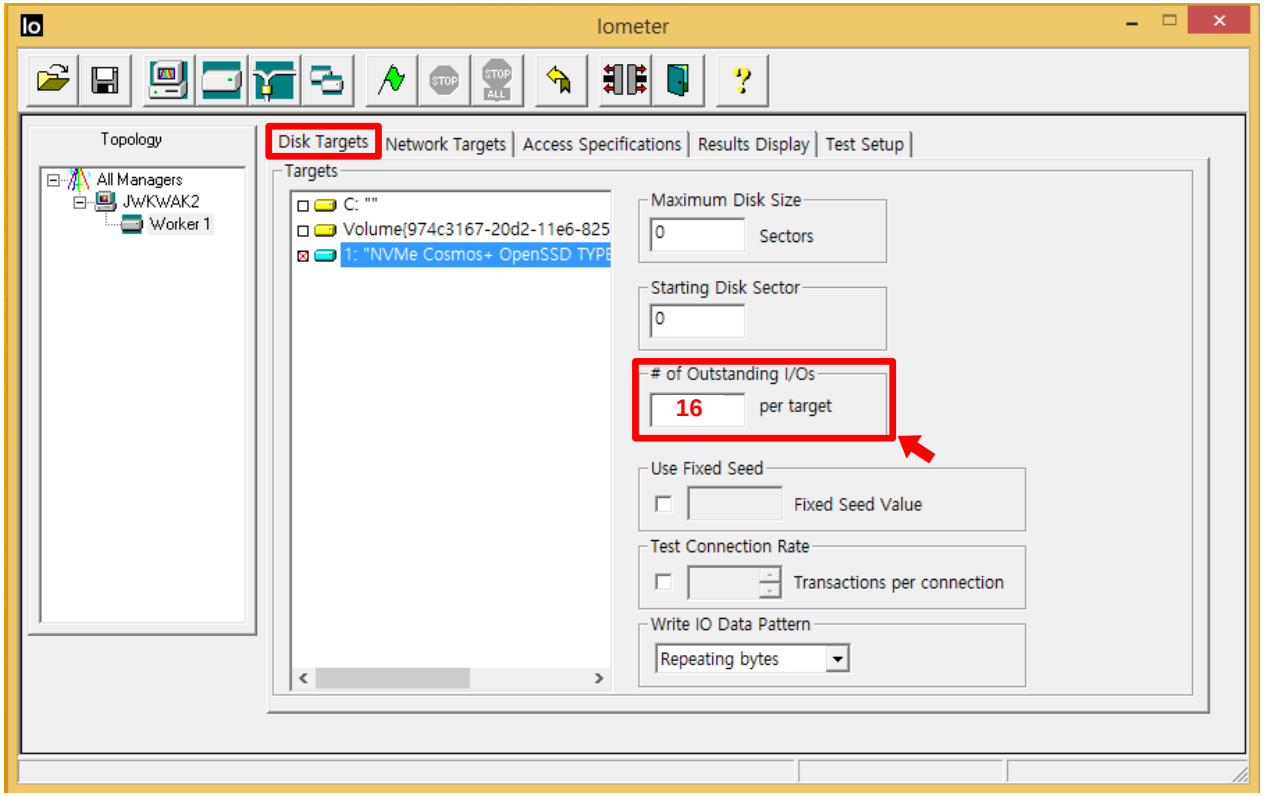


Set the Sufficient Number of Outstanding I/Os

- **X channel – Y way flash array needs "X * Y" outstanding flash requests at least for utilizing multi channel/way parallelism**
  - In case of a Cosmos+ OpenSSD configuration (8 channel – 8 way, 16KB page size), "128KB sequential write" access specification needs 8 outstanding I/Os at least

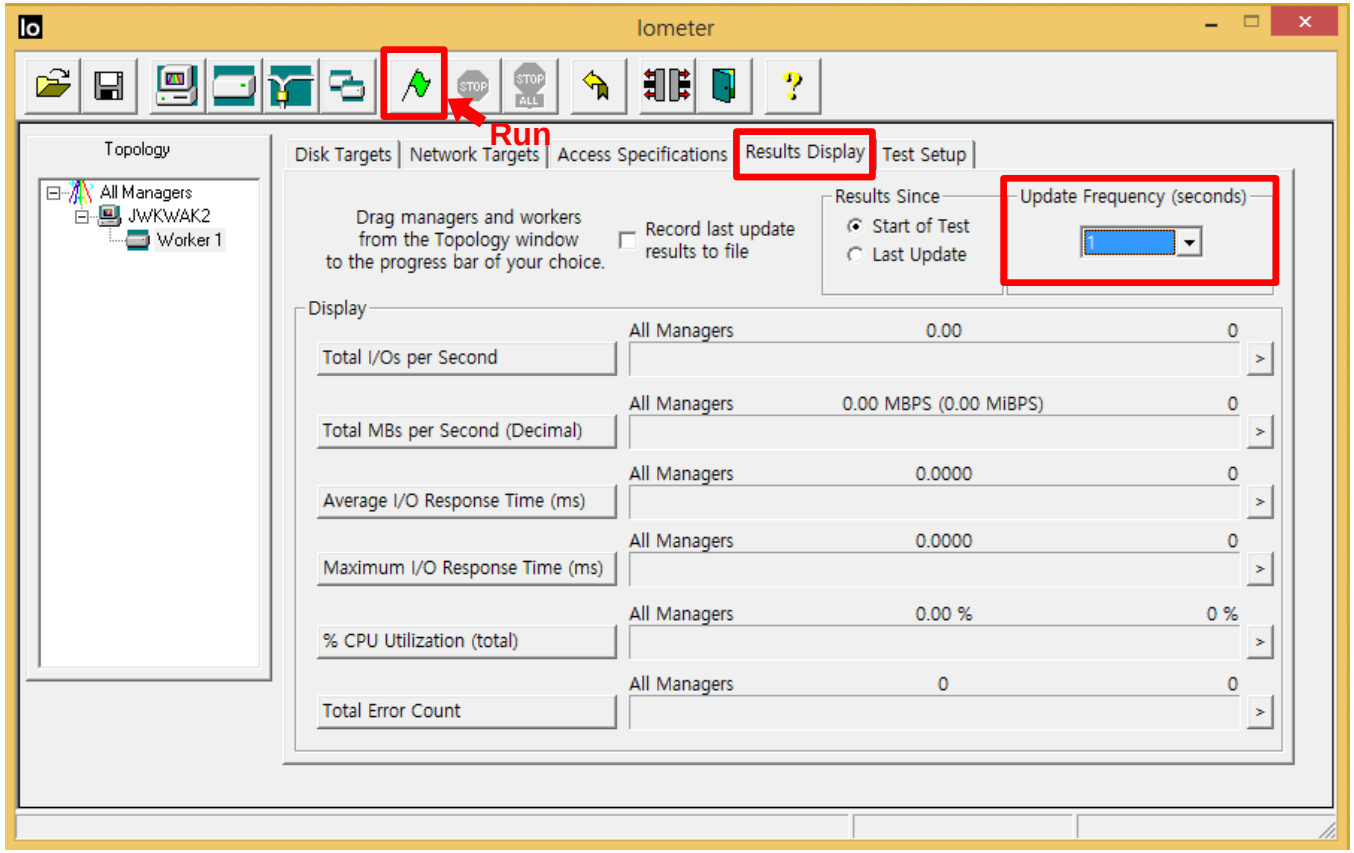
64 (128KB/16KB * 8) outstanding flash requests

  - Recommend the environment generating 2 * X * Y outstanding flash requests



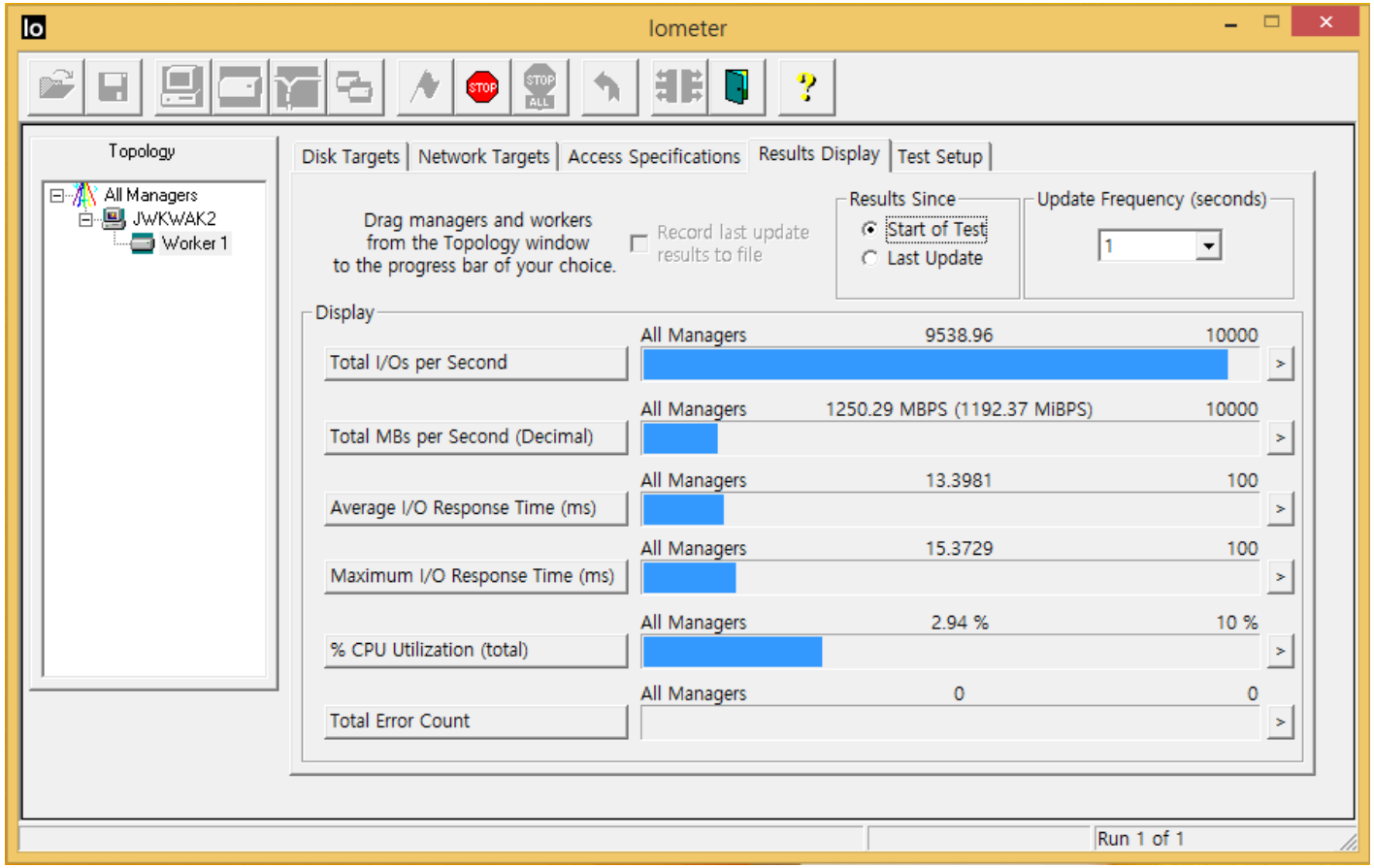
Run an Evaluation

- **Set the update frequency and click “Run” button**



Check evaluation results

- **"Results display" tab shows the performance evaluation results**
  - IOPs, throughput, average/maximum response time

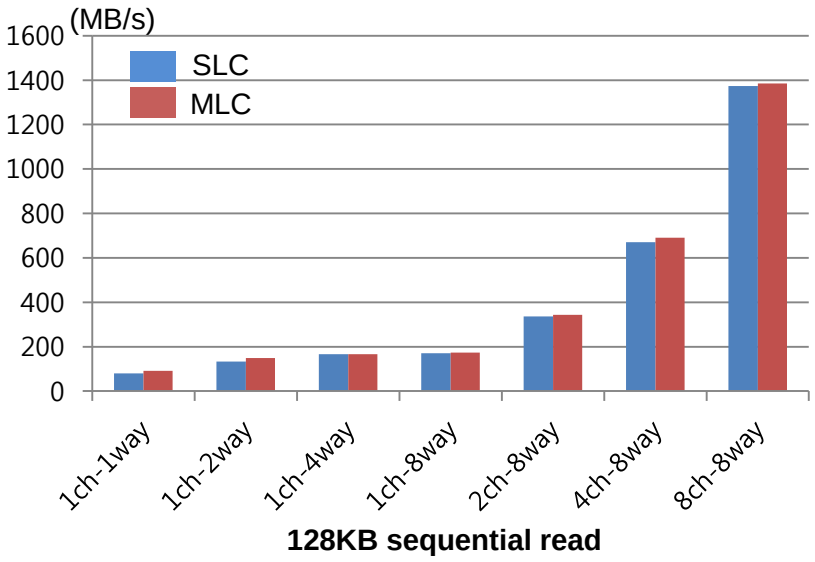


Evaluation Guideline

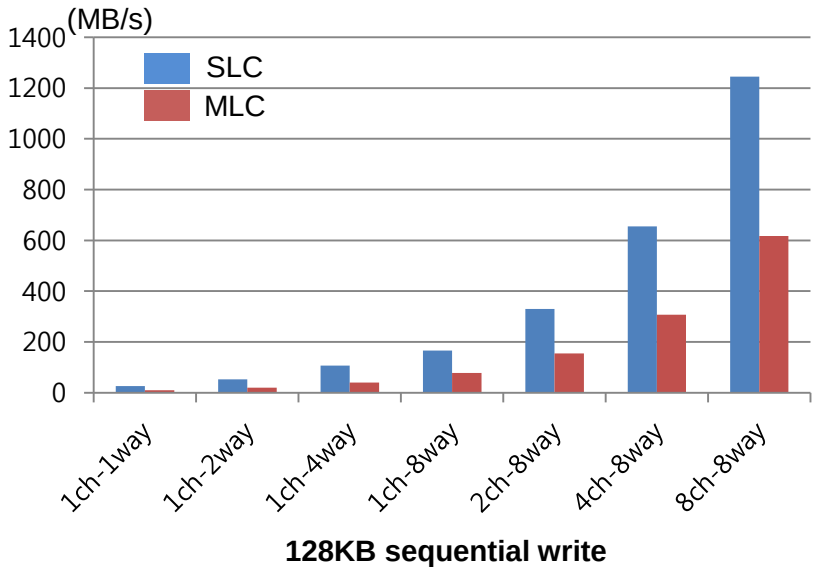
- **Perform pre-fill process before the read performance evaluation**
  - There are no mapping information for unwritten data
- **Set the number of outstanding I/Os equal or less than 256**
  - Unknown problem of host interface
- **Set the write request size equal or larger than the page size**
  - Read-modify-write process can degrade the performance
    - In case of “4KB random write”, IOPs can be decreased as the experiment progresses

Evaluation Results (1 / 3)

- **Maximum throughput/channel ≒ 173 MB/s**
  - 100Mhz DDR flash bus (bit width: 8) → 200MB/s
  - 16,384 + 1,664(spare) byte page → 90% (16,384/18048) of 200MB/s = 181MB/s
  - Overhead of flash memory controller → 173 MB/s
- **Measured throughput/channel of 8channel-8way configuration**
  - Sequential read: 99% of maximum throughput
  - Sequential write: 45~90% of maximum throughput



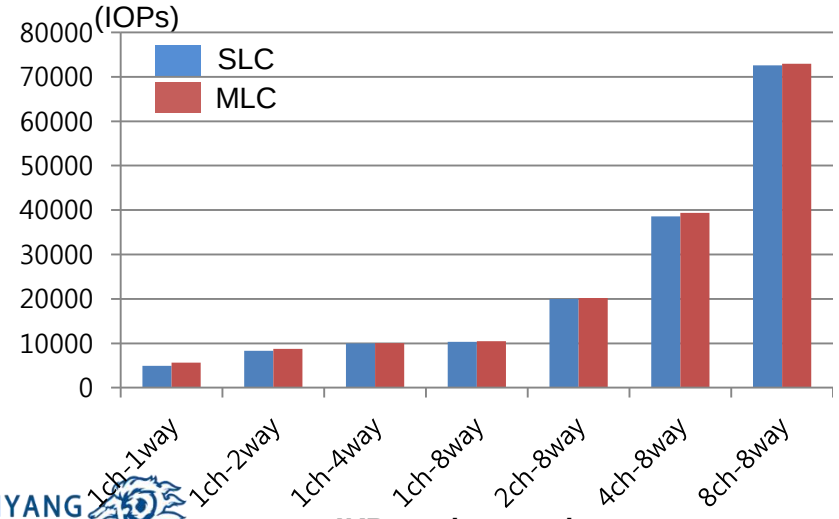
128KB sequential read



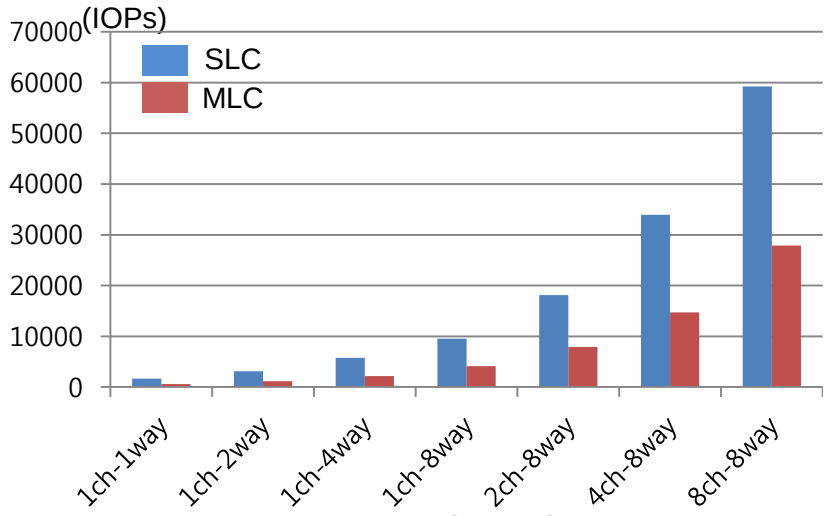
128KB sequential write

Evaluation Results (2 / 3)

- **Maximum 4KB IOPs/channel ≒ 10812 IOPs**
  - Page mapping → a page is accessed in order to access 4KB data
  - 173MB/s(Maximum throughput/channel) ÷ 16KB (page size) = 10812 IOPs
- **Measured throughput/channel**
  - 1channel-8way configuration
    - Random 4KB read: 96% of maximum 4KB IOPs
    - Random 4KB write: 38~88% of maximum 4KB IOPs
  - 8channel-8way configuration
    - SW-based scheduling has a larger latency in many channel/way configuration
    - Scheduling latency can increase the idle time of hardware controllers



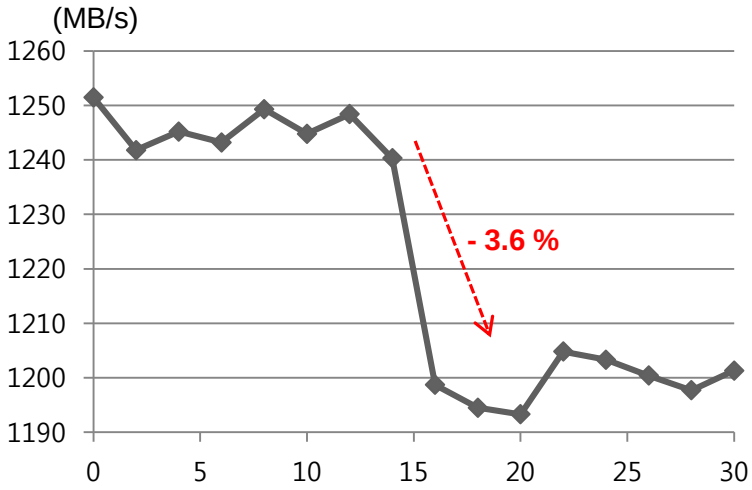
4KB random read



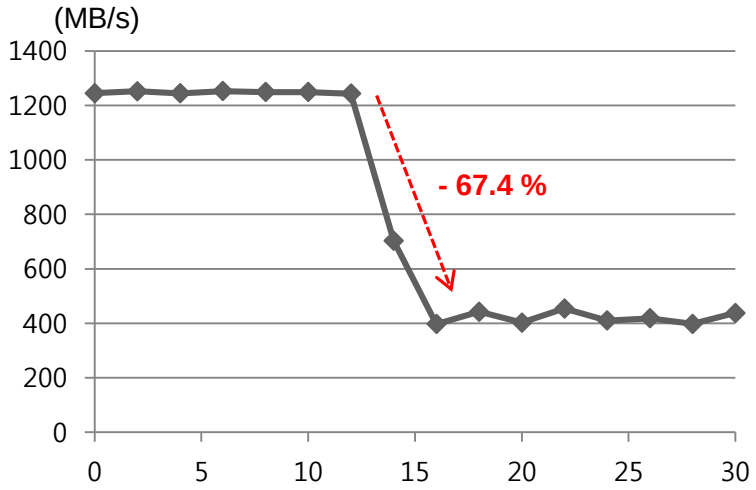
4KB random write (w/o read-modify-write)

Evaluation Results (3 / 3)

- **Performance degradation by on-demand garbage collection**
  - After all available blocks are used, garbage collection is triggered steadily
  - Effect of performance degradation varies depending on copy operation overhead
    - Copy operation overhead depends on the number of valid page belong to victim blocks



Victim blocks with no valid pages
in SLC 8channel-8way configuration



Victim blocks with valid pages (half of total page)
in SLC 8channel-8way configuration

Thank You