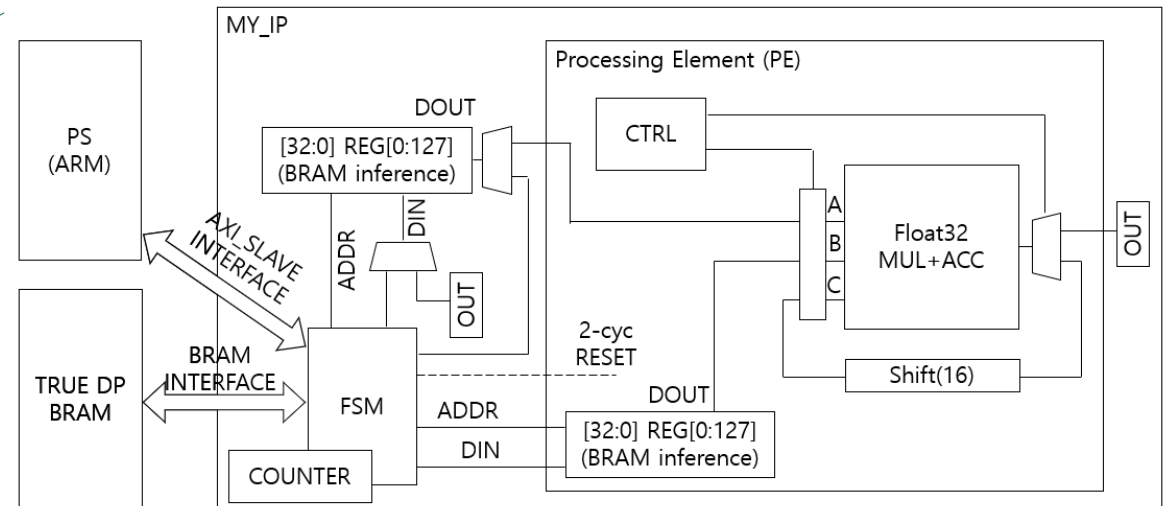
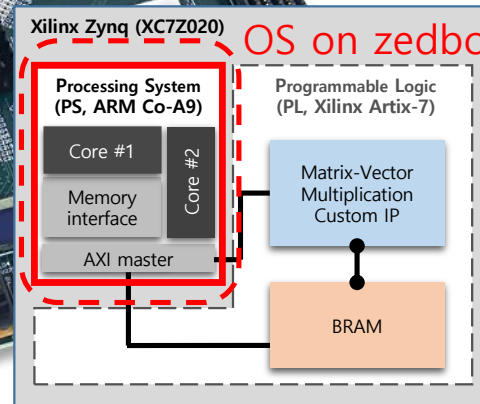
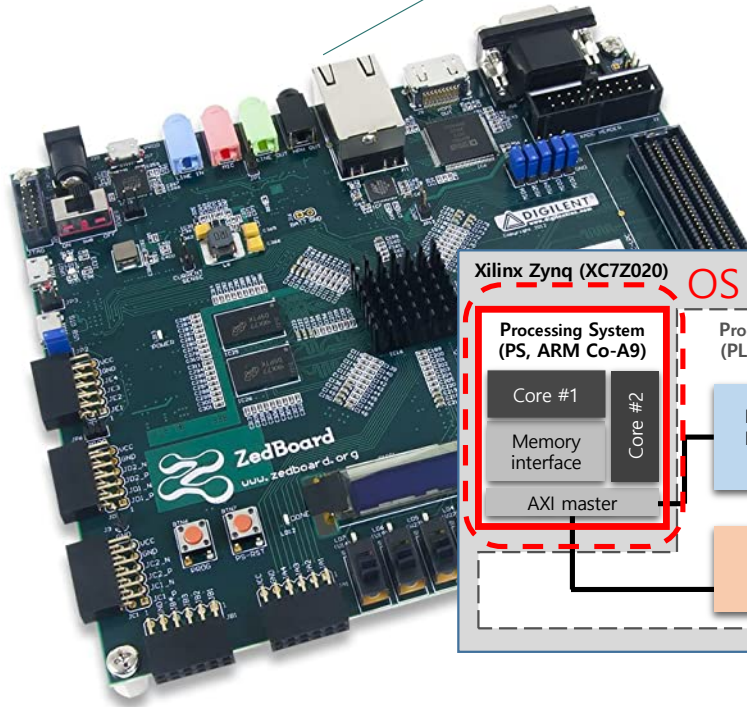


# Practice 9

- OS + FPGA System

Computing Memory Architecture Lab.

# Final Project Overview: Matrix Multiplication IP



# Overview

---

## ■ Vivado Block Design Tutorial

- First block design = Processing System + BRAM + Connectivity
  - Note) Term project = PS + BRAM + Connectivity + Custom IP

## ■ FPGA + Linux Tutorial

- Debian Linux on zedboard
- Access BRAM via C program

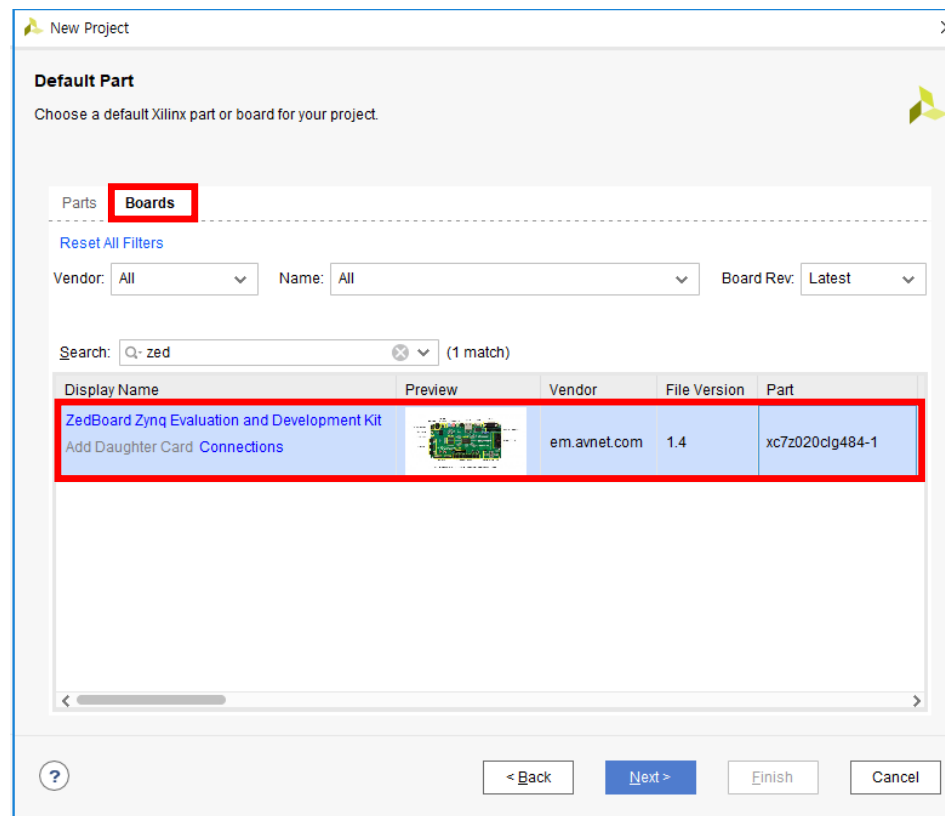
## ■ Practice

- Running a sample project

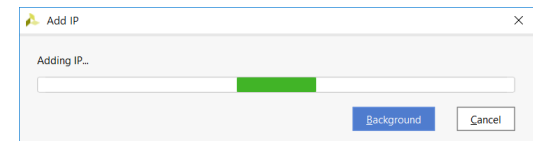
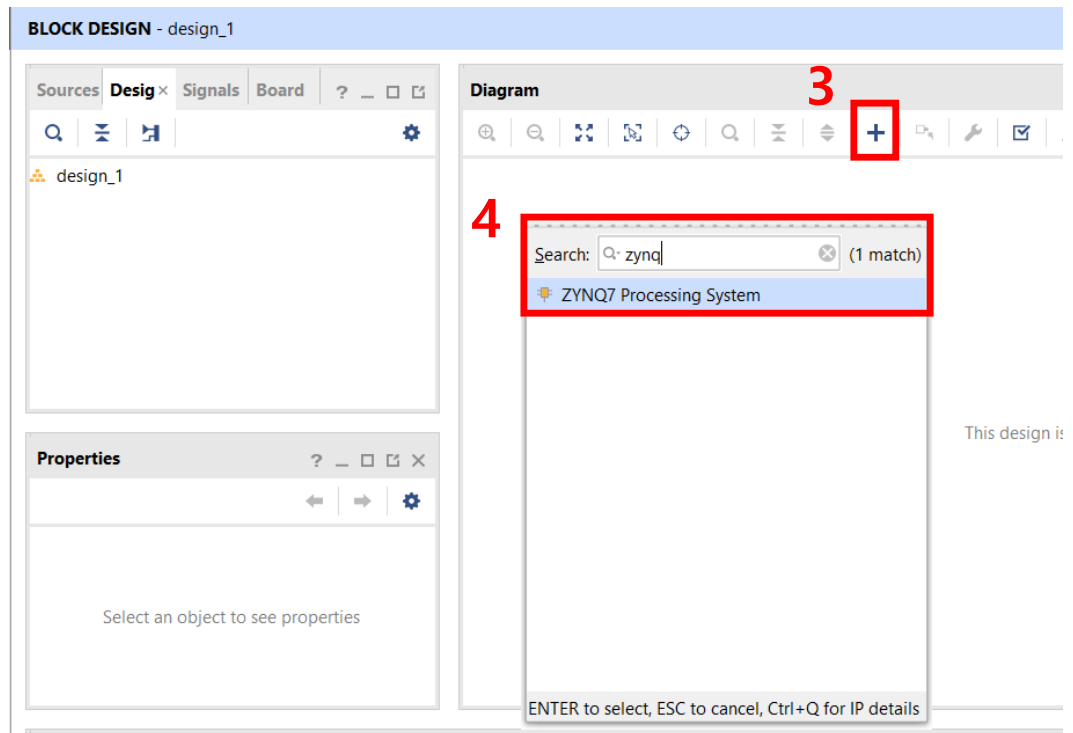
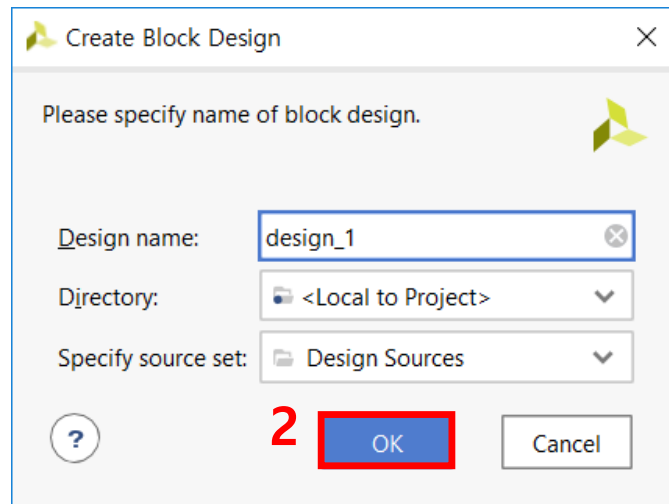
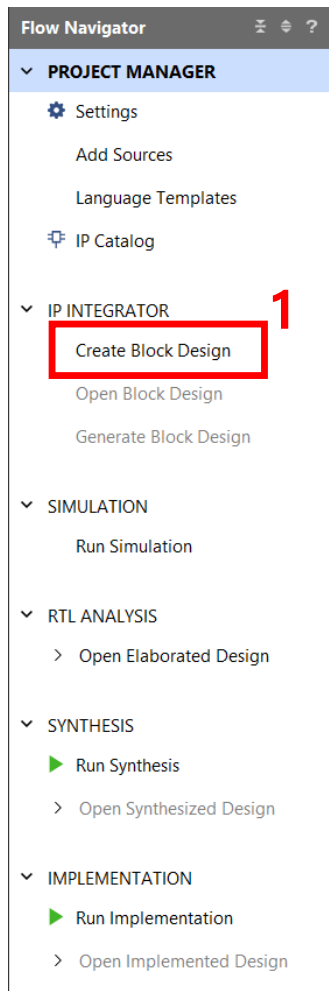
# Vivado Block Design Tutorial

# Vivado project creation

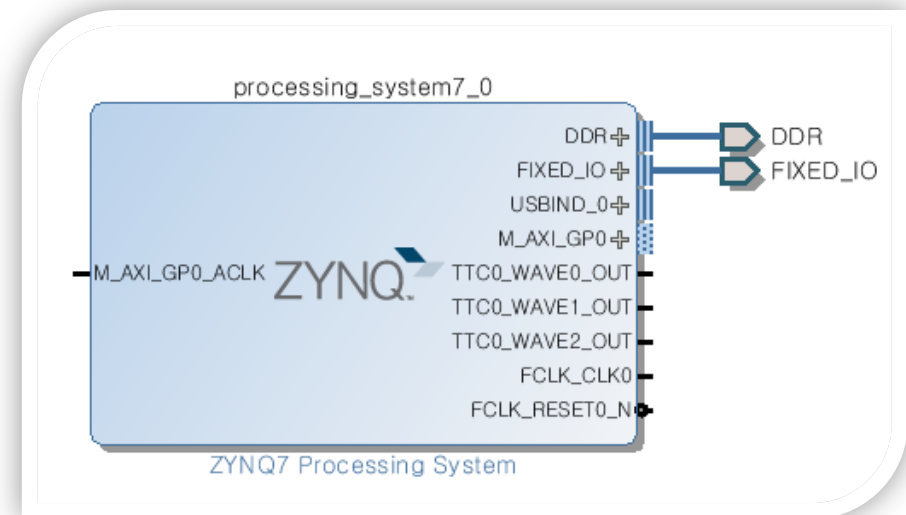
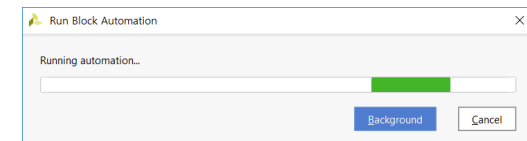
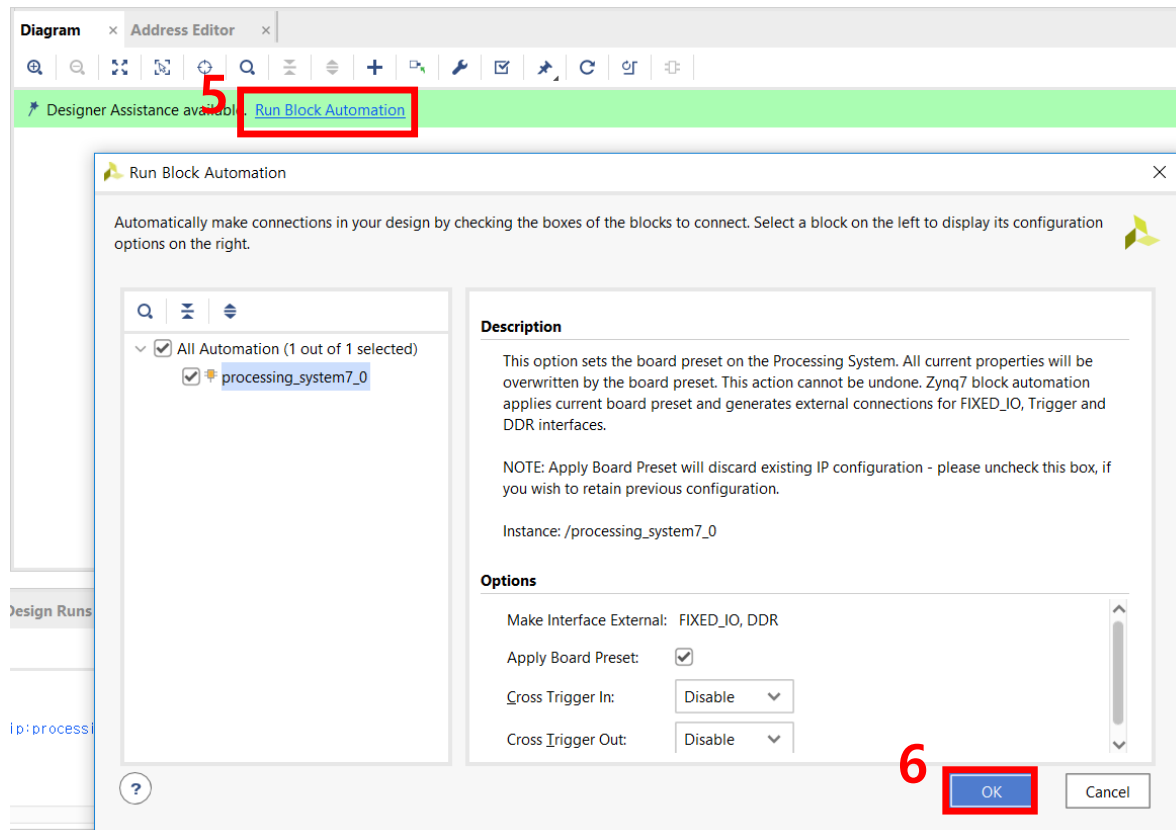
- Choose part or board
  - We are going to use ZedBoard



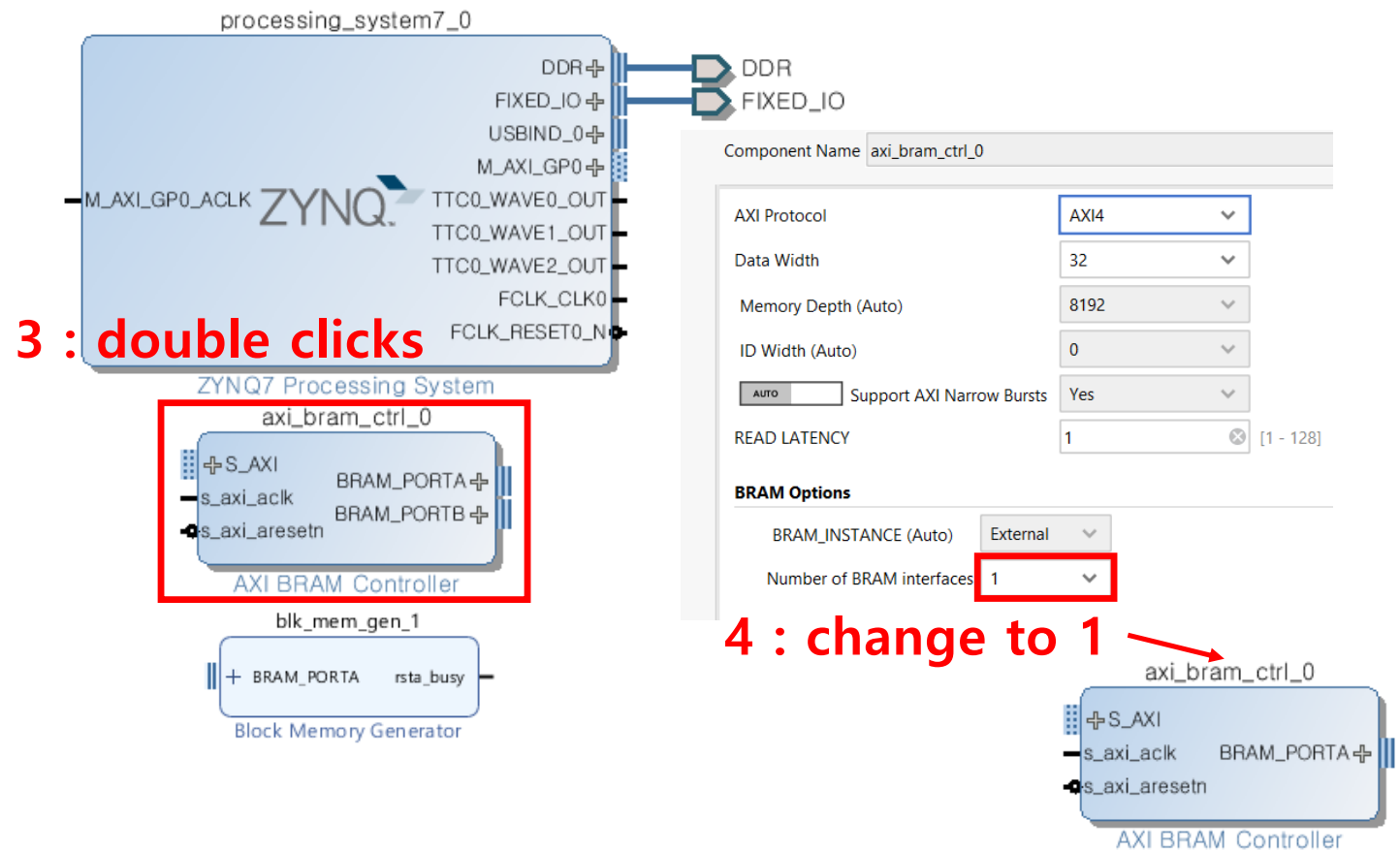
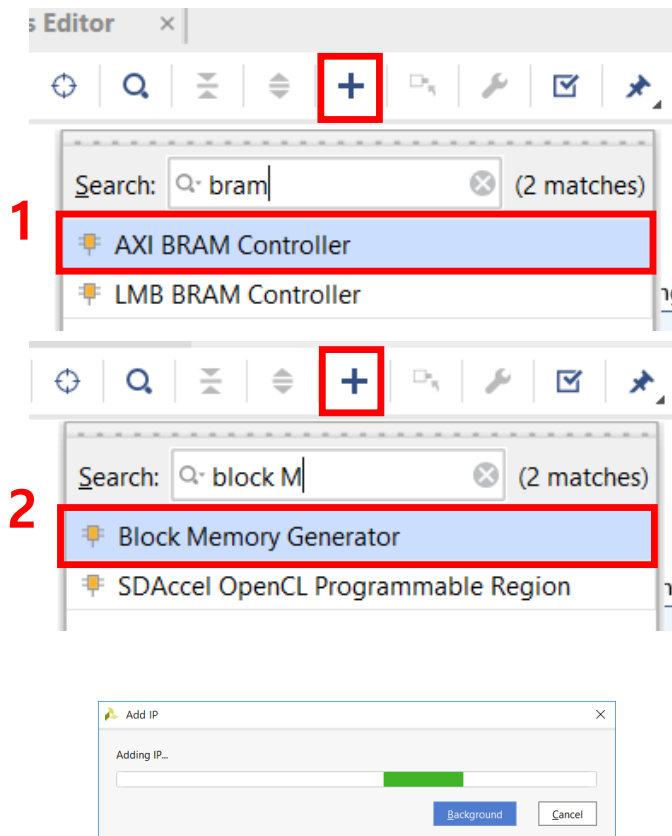
# Block Design - PS



# Block Design - PS

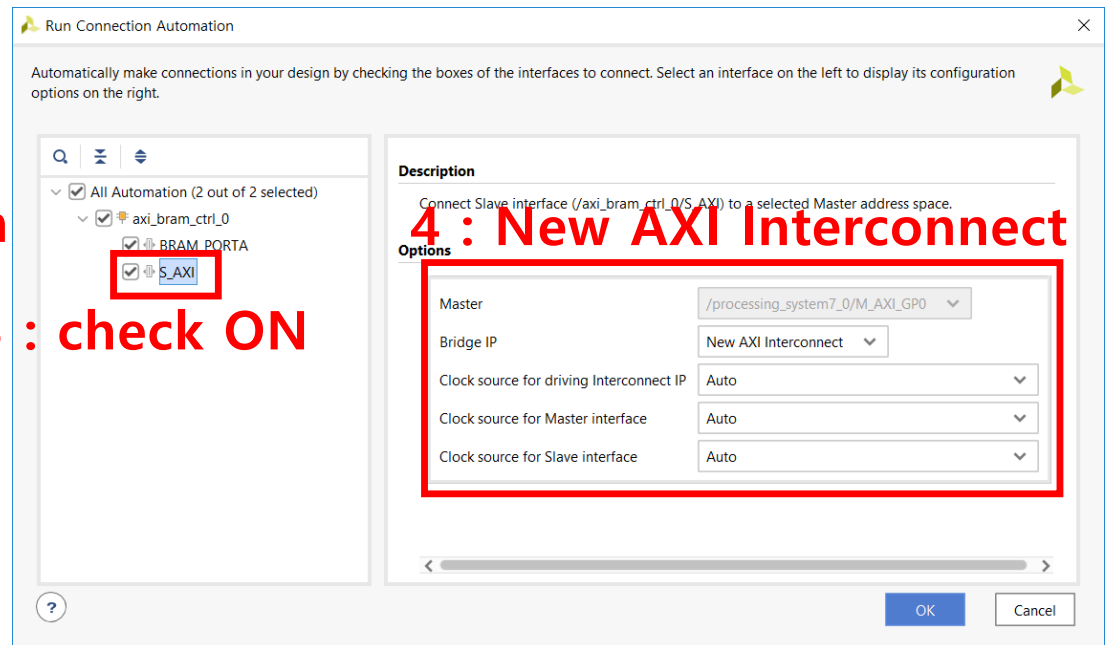
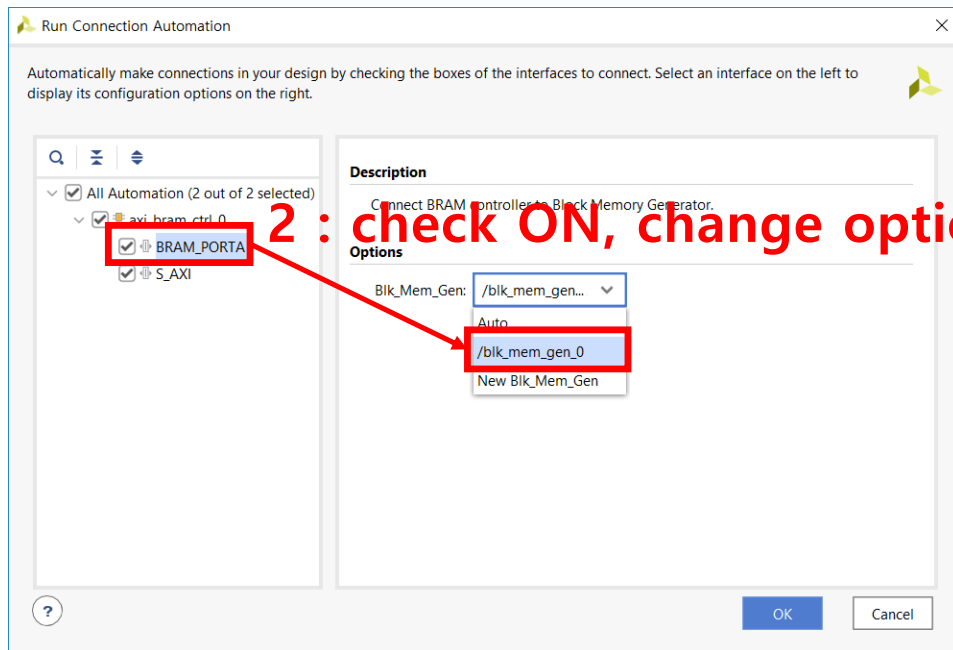
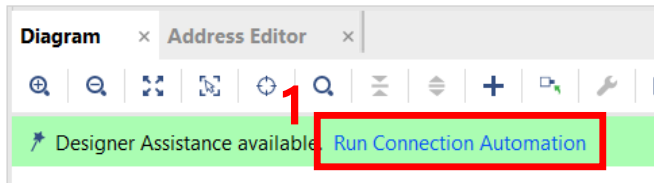


# Block Design - BRAM

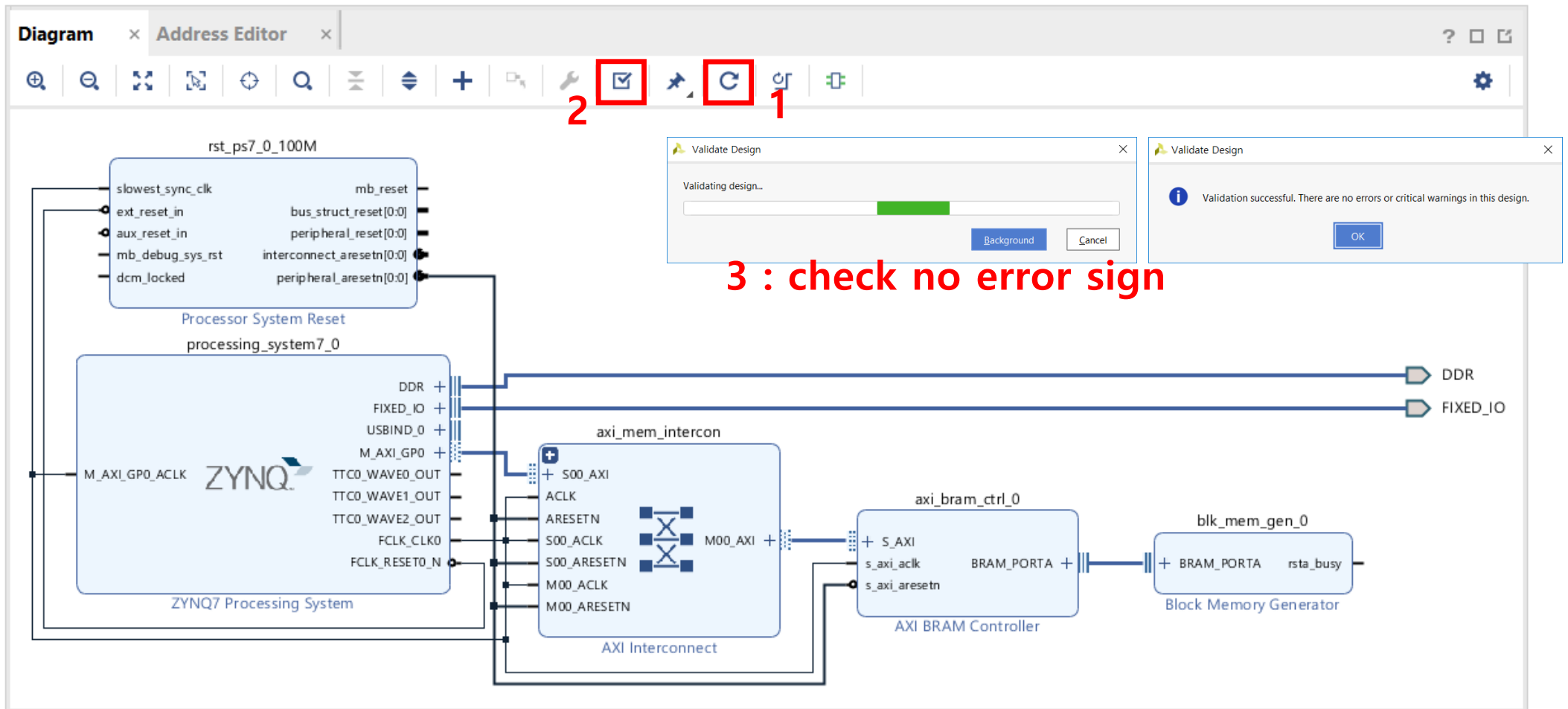




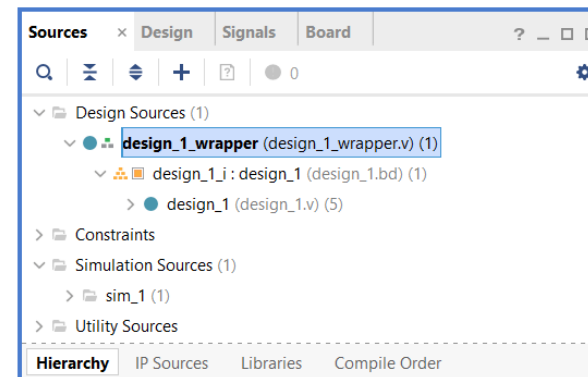
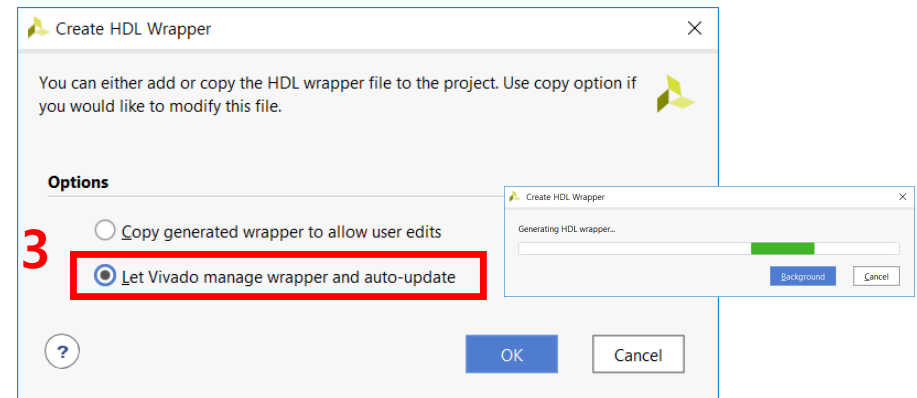
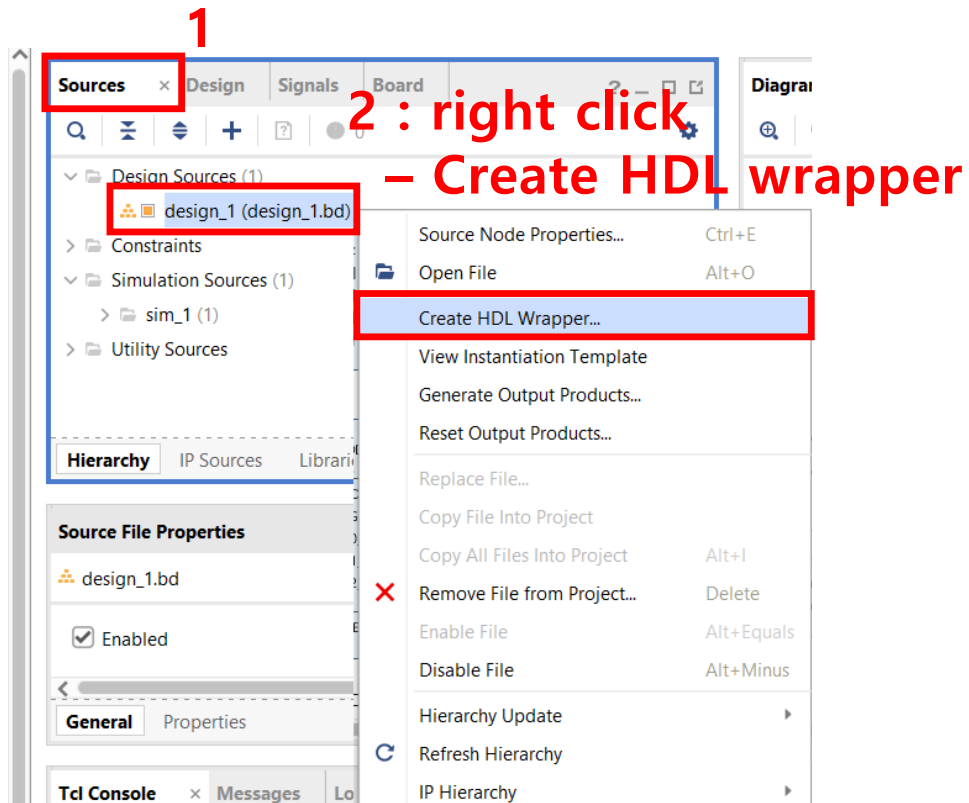
# Block Design - Connectivity



# Block Design - Connectivity



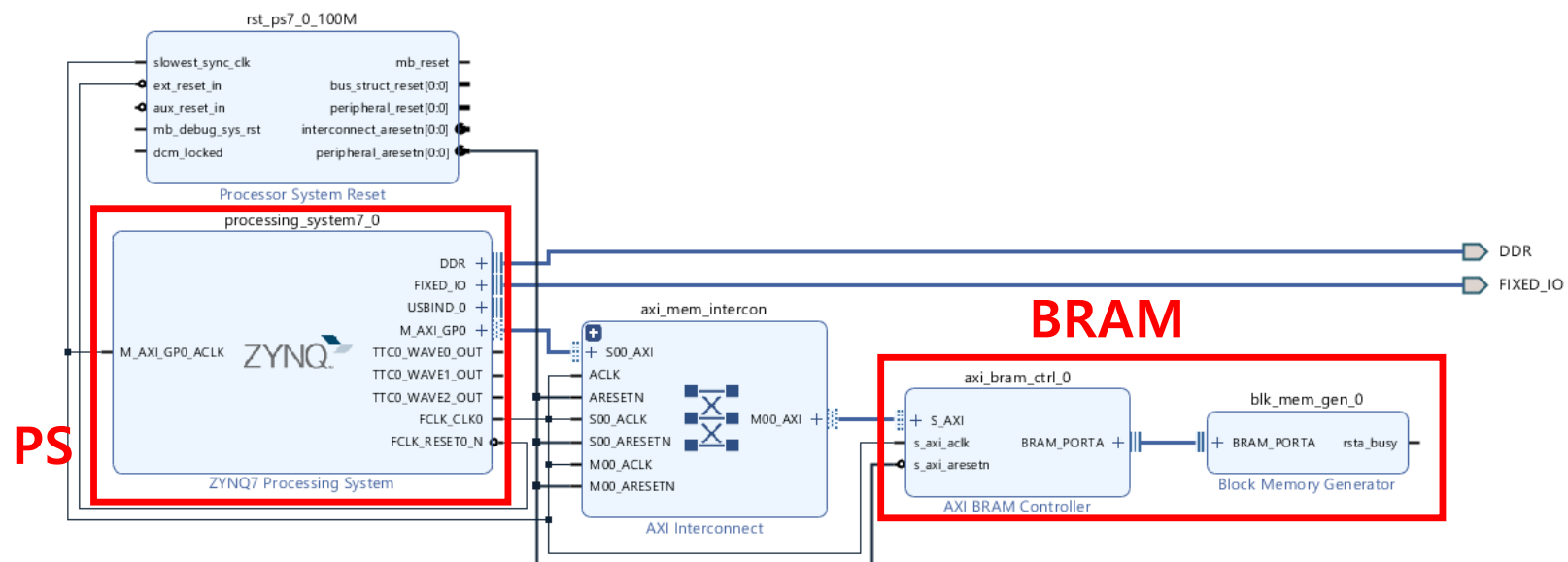
# Block Design - Bitstream



- ▼ SYNTHESIS
  - ▶ Run Synthesis
  - > Open Synthesized Design
- ▼ IMPLEMENTATION
  - ▶ Run Implementation
  - > Open Implemented Design
- ▼ PROGRAM AND DEBUG
  - 4 ▶ **Generate Bitstream**
  - > Open Hardware Manager

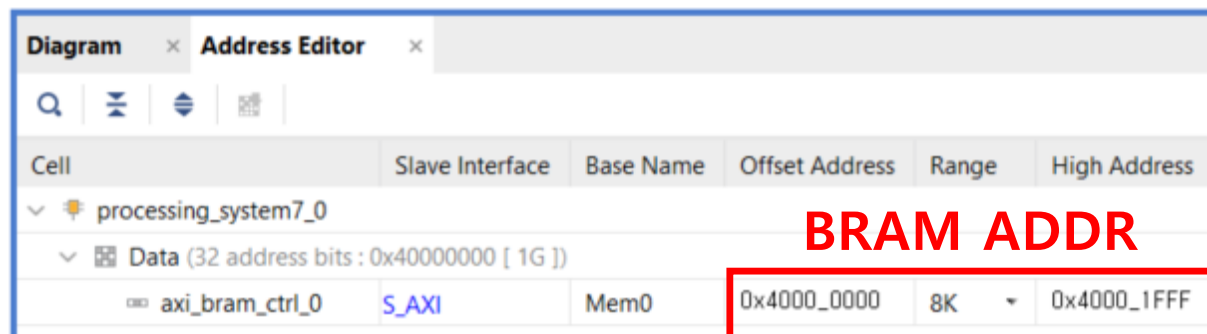
# Block Design - Summary

- Having built **a first block design** equipped with
  - PS: Processing System (part of Zynq-PS, ARM Cortex A9)
  - BRAM: Block Random Access Memory (part of Zynq-FPGA)
  - Connectivity (AXI interconnect, part of Zynq-FPGA)



# Programmer's Perspective

- BRAM is @ address 0x4000\_0000 ~ 0x4000\_1FFF
  - Note) DRAM (BD.IC25/26) is @ address 0x0000\_0000 ~ 0x3FFF\_FFFF
- System call **mmap** can be used to access BRAM (*TBD*)
  - `int foo = open("/dev/mem", O_RDWR);`
  - `float *ptr = mmap(NULL, size, PROT_READ|PROT_WRITE, MAP_SHARED, foo, 0x40000000);`



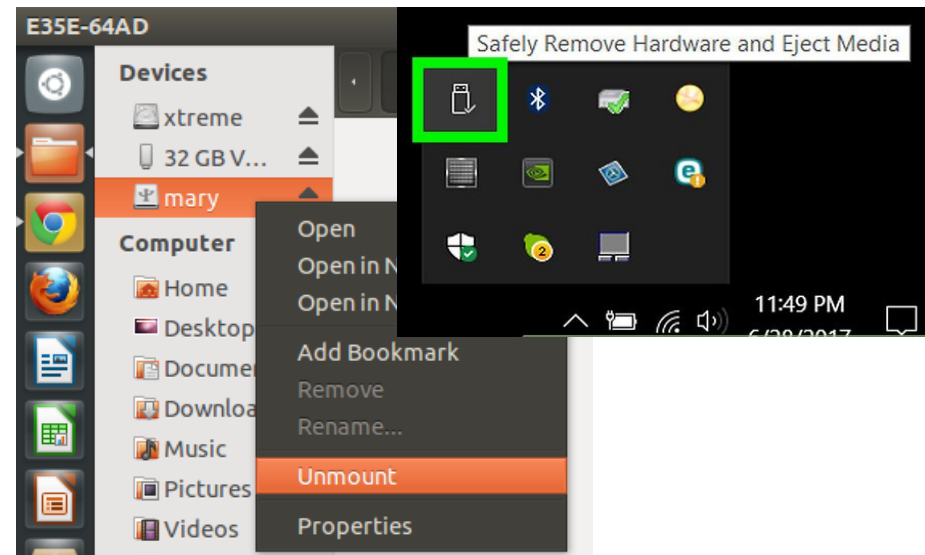
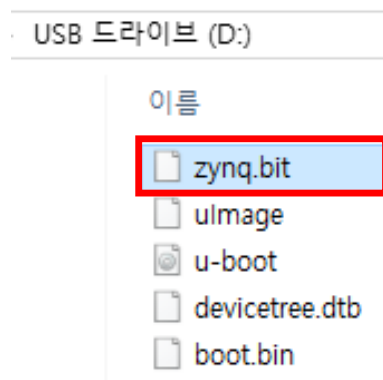
The screenshot shows the 'Address Editor' window with a table of memory components. The table has columns: Cell, Slave Interface, Base Name, Offset Address, Range, and High Address. The 'Cell' column shows a tree structure with 'processing\_system7\_0' expanded to 'Data (32 address bits : 0x40000000 [ 1G ])', which is further expanded to 'axi\_bram\_ctrl\_0'. The 'Slave Interface' is 'S\_AXI' and the 'Base Name' is 'Mem0'. The 'Offset Address' is '0x4000\_0000', the 'Range' is '8K', and the 'High Address' is '0x4000\_1FFF'. A red box highlights the 'Offset Address', 'Range', and 'High Address' columns. Above this box, the text 'BRAM ADDR' is written in red.

Cell	Slave Interface	Base Name	Offset Address	Range	High Address
processing_system7_0					
Data (32 address bits : 0x40000000 [ 1G ])					
axi_bram_ctrl_0	S_AXI	Mem0	0x4000_0000	8K	0x4000_1FFF

# FPGA + Linux Tutorial

# Preparing the Bitstream

- Find your bitstream(.bit) file in (\$project\_name).runs/impl\_1/design\_1\_wrapper.bit
  - Move your bitstream file to partition1 and change file name to zynq.bit
  - (before detach the sdcard from computer) unmount or eject your sdcard from your computer. Or not your sdcard will be broken. (penalty ☺)



# Notations

---

- HOST\$ XXX
  - Type XXX @ the terminal of your Ubuntu-PC
- BOARD\$ YYY
  - Type YYY @ the terminal of ZedBoard
    - Which is equivalent to the after-terminal of below command
      - HOST\$ minicom -D /dev/ttyACM0



# Preparing the board

- Insert SD card (1)
- Set SD boot mode (2)
  - BD.JP7 ~ BD.JP11
    - 5'b00000 -> **5'b01100**
- Insert LAN Cable (3)
- Insert power cable (4)
  - Yet stay power OFF
- Connect USB cable (5)
  - BD.J14
  - HOST\$ dmesg

- Micro 5pin cable  
If 뻣뻣(stiff) -> 칼팁(sharp tip) -> 후크(hook)  
No further labs for broken board

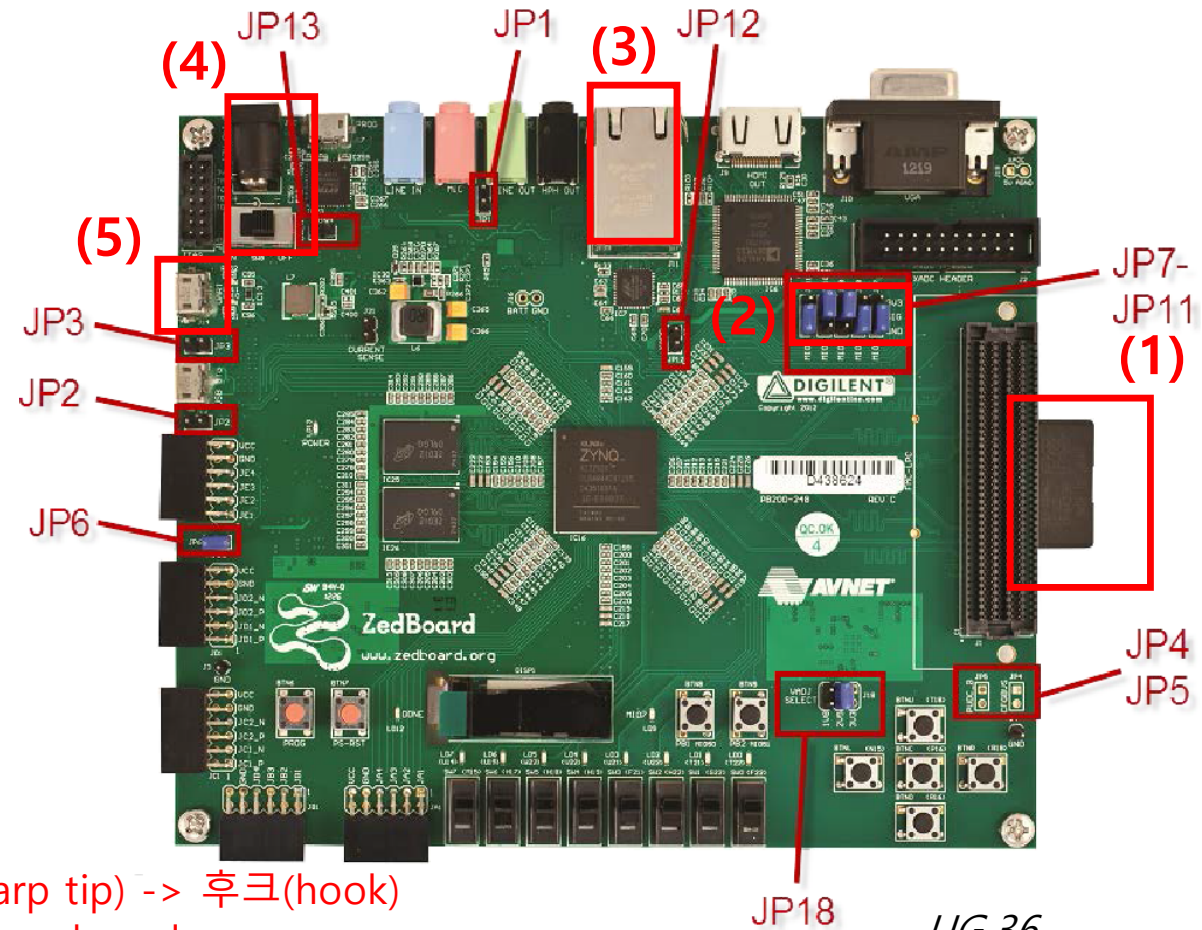


Figure 20 - ZedBoard Jumper Map

UG.36

# First Contact to Embedded Linux

- Board Power ON
- Open terminal @ HOST
  - HOST\$ minicom -D /dev/ttyACM0
  - Login ID/PW: zed/zedzed
- Run example program
  - BOARD\$ git clone [https://github.com/tahsd/hsd20\\_lab09](https://github.com/tahsd/hsd20_lab09)
  - BOARD\$ cd hsd20\_lab09
  - BOARD\$ make

- (before un-plugging the board) BOARD\$ sudo poweroff
  - If you do not execute poweroff on terminal and eject the sdcard, your sdcard will not work permanently. (penalty ☺)

```
Debian GNU/Linux 8 debian-zynq ttyPS0
debian-zynq login: zed
입력 :
마지막 로그인 : 목 1월 1 00:01:10 UTC 1970 일시 ttyPS0
Linux debian-zynq 4.0.0-xilinx #1 SMP PREEMPT Tue Jan 10 23:55:15 KST 2017 armv7l
The programs included with the Debian GNU/Linux system are free software;
the exact distribution terms for each program are described in the
individual files in /usr/share/doc/*/copyright.
Debian GNU/Linux comes with ABSOLUTELY NO WARRANTY, to the extent
permitted by applicable law.
zed@debian-zynq:~$
```

```
[ OK ] Reached target Multi-User
System.
[ OK ] Reached target Graphical
Interface.
Starting Update UTMP about
System Runlevel Changes...
[ OK ] Started Update UTMP about
System Runlevel Changes.
```

```
Debian GNU/Linux 8 debian-zynq ttyPS0
debian-zynq login:
```

addr	FPGA(hex)
0	0
1	2
2	4
3	6
...	

# Source Code – main.c

---

```
int foo = open("/dev/mem", O_RDWR);           // 0x4000_0000 refers to offset of the file descriptor
// Given a pathname for a file, open() returns a file descriptor
// 'dev/mem' refers to the system's physical memory
// O_RDWR means both readable and writable access mode
int *fpga_bram = mmap(NULL, SIZE *
sizeof(int), PROT_READ|PROT_WRITE,
MAP_SHARED, foo, 0x40000000);
// mmap() creates a new mapping in the virtual address space of
the calling process
// NULL means that the kernel chooses the address for mapping
// SIZE specifies the length of the mapping
// PROT_ arguments describe the memory protection (RD/WR)
// MAP_SHARED makes updates visible to other processes
// foo indicates the file descriptor to be mapped

for (i = 0; i < SIZE; i++)
    *(fpga_bram + i) = (i * 2);
// write arbitrary data on the BRAM area
printf("%-10s%-10s\n", "addr", "FPGA(hex)");
for (i = 0; i < SIZE; i++)
    printf("%-10d%-10X\n", i, *(fpga_bram + i));
// read and show the data to check if BRAM's working correctly
```

# Source Code – Makefile

---

```
all: main.c
```

```
    gcc main.c && sudo ./a.out
```

```
// target : prerequisites
```

```
// [TAB]recipe
```

```
//
```

```
// ex)
```

```
// make $target : run recipes of $target using prerequisites
```

```
// make : (missing $target arguments) run the first target
```

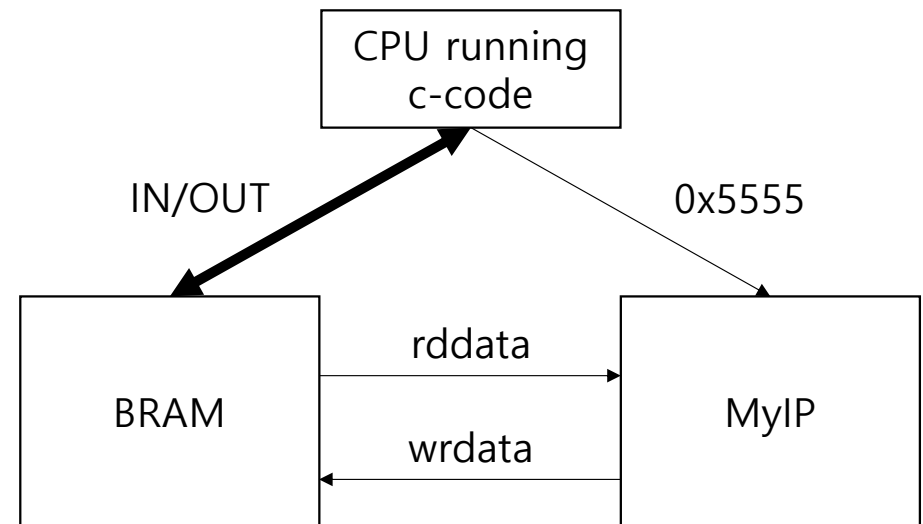
# Main Practice

# Practice

---

## ■ Running a sample project

- Run a given sample project in your FPGA linux.
  - Run and explain a brief functionality of sample project on report.
- Sample Project
  - [https://github.com/tahsd/hsd20\\_lab09\\_practice](https://github.com/tahsd/hsd20_lab09_practice)
    - Sample HW: zynq.bit (made by MyIP)
    - Sample SW: main.c Makefile
  - You don't need anything to edit or implement on HW.
  - For investigation purposes, you can modify the c-code at your disposal.



# Comments from TA

---

- Q. How to export my own \*.bit into SD card?
  - Copy \*.bit **A** to **B**
  - **A**: {project}.runs/impl\_1/design\_1\_wrapper.bit
  - **B**: /SDCARD\_1.1G\_PARTITION/zynq.bit
- Q. How to quit my minicom instance?
  - Press **ctrl A q** -> yes
- Q. How to turn the board off and on without deterring minicom instance?
  - Press BD.BTN7
- Click [eject] before removing SD card from the card reader

# Homework

---

- Requirements

- Result

- Attach your C codes & Bitstream files you modified (just for Main Practice)
    - Attach a screenshot that can show your code works in ZedBoard

- Report

- Explain functionality of MyIP in Main Practice
    - In your own words
    - Either in Korean or in English
    - # of pages does not matter
    - **PDF only!!**

- **Result + Report to a .zip**

- Upload (.zip) file on ETL

- Submit one (.zip) file

- zip file name : [Lab09]name.zip (ex : [Lab09]홍길동.zip)

- Due: 5/19(TUE) 23:59

- **No Late Submission**



# Term Project

# Submit schedule

## ▪ Lab 6 (V\*V)

- V\*V (~5/12 23:59pm)

## ▪ Term project v0 중간제출(Interim check) (due : ~5/26 23:59pm, no late submission)

- Expand V\*V into M\*V
- Demo : Waveform
- Report : pdf report + every files you've implemented (Verilog files, etc ..)
- Make your own input file.

## ▪ Lab 10 (Custom IP)

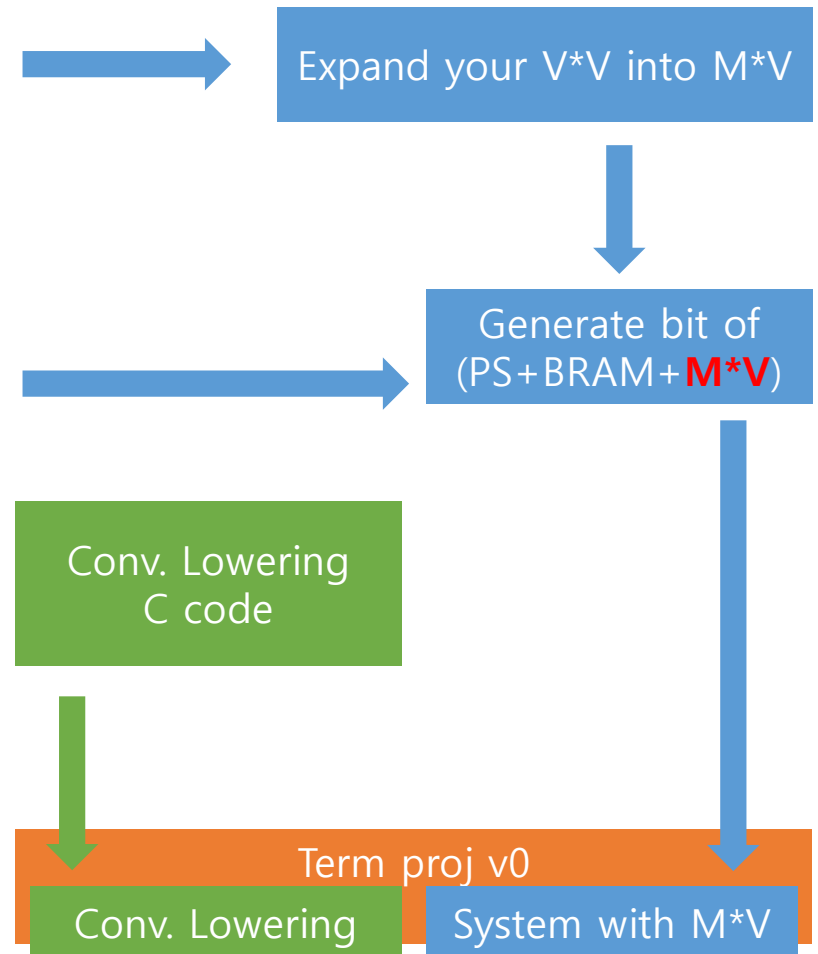
- Refer tutorial with shifter IP
- Generate bitstream of (PS+BRAM+**shifter IP**)
- Demo : no
- Report : no

## ▪ +) Convolution Lowering in FPGA (Additional Part in Lab10)

- Convert conv. Operation into M\*V
- Starting from c code of Lab 2
- Demo : no
- Report : no

## ▪ Term project v0 최종제출(final check) (due : ~6/16 23:59pm, no late submission)

- Conv. Lowering + Custom IP system with MV
- Demo : Zedboard
- Report : pdf report + every files you've implemented(bit stream, ip\_repo, Verilog files, c files etc.. everything you've used)
- Refer Lab10



# Term Project v0 중간제출

---

## ■ Requirements

### - Result

- Attach your project folder with all your verilog codes (e.g., M\*V, test bench)
- Attach your M\*V waveform(simulation result) with [student\_number, name]
  - The correct waveform should be shown to confirm the operation of your code.
  - Refer to Practice3 about screenshot.

### - Report

- Explain operation of M\*V with waveform that you implemented
- In your own words
- Either in Korean or in English
- # of pages does not matter
- **PDF only!!**

### - **Result + Report to one .zip file**

## ■ Upload (.zip) file on ETL

### - Submit one (.zip) file

- zip file name : [Term0-mid]name.zip (ex : [Term0-mid]홍길동.zip)

### - Due: 5/26(TUE) 23:59

- **Students who don't submit report until 5/26 lab start have to attend lab offline (6:30~ in hardware practice room)**
- **No Late Submission**

# Term Project v0

---

- Due: 6/16(TUE) 23:59
  - **Details will be noticed next week**

# Term Project v1, v2

---

- Due: 6/23(TUE) 23:59
  - Extra Credits will be given
    - Good Performance Gain
    - Computation Time, Utilization
  - Details will be noticed later

# Appendix – Linux Booting on Zynq

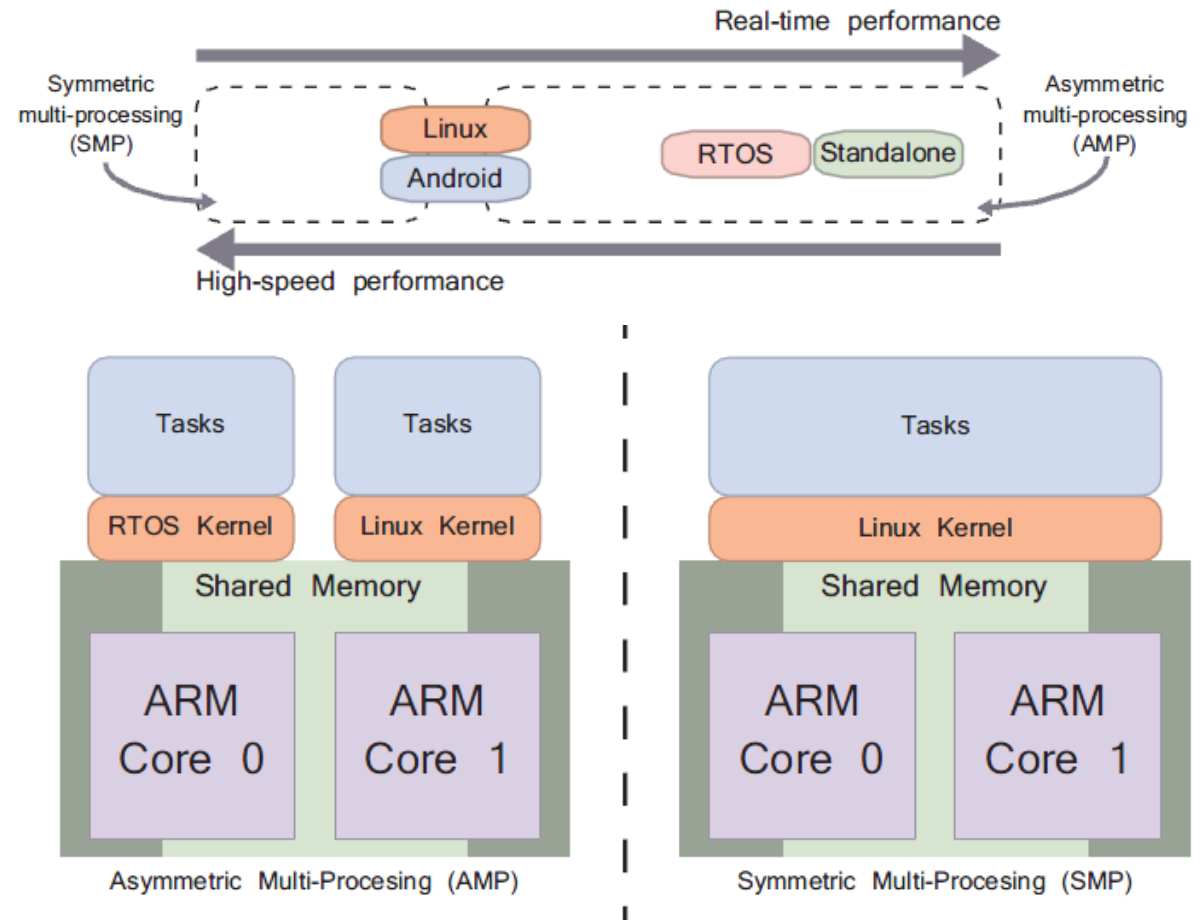
# Introduction to OS on Zynq

---

- Why Use an Embedded Operating System?
  - Reducing Time to Market
  - Make Use of Existing Features
  - Reduce Maintenance and Development Costs
- Choosing the Right Type of Operating System
  - Standalone Operating Systems
  - Real-Time Operating Systems (RTOS)
  - Others: Linux, Android, ...

# Multi-Processor Systems

- Asymmetric/Symmetric Multi-Processing (AMP/SMP)





# Note) Zynq AMP Example

---

Application Note: Zynq-7000 AP SoC



XAPP1078 (v1.0) February 14, 2013

## Simple AMP Running Linux and Bare-Metal System on Both Zynq SoC Processors

Author: John McDougall

### Summary

The Zynq™-7000 All Programmable SoC contains two ARM® Cortex™-A9 processors that can be configured to concurrently run independent software stacks or executables. This application note describes a method of starting up both processors, each running its own operating system and application, and allowing each processor to communicate with the other through shared memory.

### Design Overview

In this reference design, each of the two Cortex-A9 processors is configured to run its own software. CPU0 is configured to run Linux and CPU1 is configured to run a bare-metal application.

In this AMP example, the Linux operating system running on CPU0 is the master of the system and is responsible for:

- System initialization
- Controlling CPU1's startup
- Communicating with CPU1
- Interacting with the user

The bare-metal application running on CPU1 is responsible for:

- Managing a "heart beat" that can be monitored by Linux on CPU0
- Communicating with Linux on CPU0
- Servicing interrupts from a core in the programmable logic (PL)
- Communicating interrupt events to Linux running on CPU0

# Xilinx Zynq-Linux

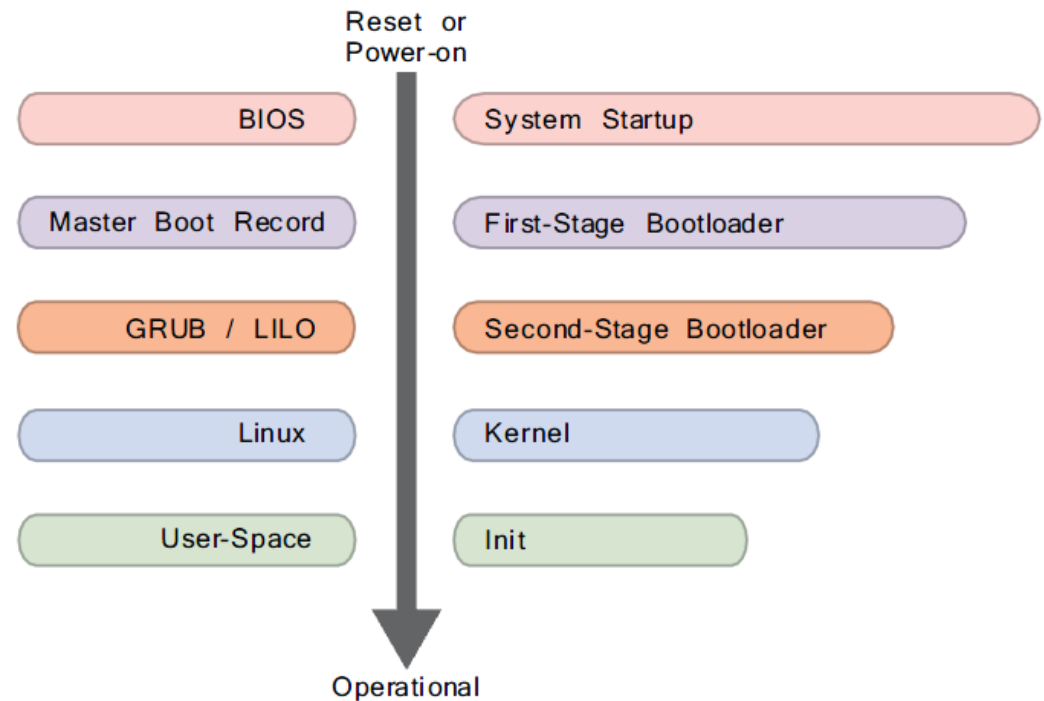
- Linux kernel + Xilinx BSP & device drivers

Component	Driver Location	In Mainline Kernel
Analog-to-Digital Converter	drivers/hwmon/xilinx-xadcps.c	No
ARM global timer	drivers/clocksource/arm_global_timer.c	Yes
ARM local timers	arch/arm/kernel/smp_twd.c	Yes
CAN Controller	drivers/net/can/xilinx_can.c	No
DMA Controller (PL330)	drivers/dma/pl330.c	Yes
Ethernet MAC	drivers/net/ethernet/xilinx/xilinx_emacps.c	No
	drivers/net/ethernet/cadence/macb.c	Yes
GPIO	drivers/gpio/gpio-xilnxps.c	No
I2C Controller	drivers/i2c/busses/i2c-cadence.c	Yes
Interrupt Controller	arch/arm/common/gic.c	Yes
L2 Cache Controller (PL310)	arch/arm/mm/cache-l2x0.c	Yes

# Generic Linux Booting

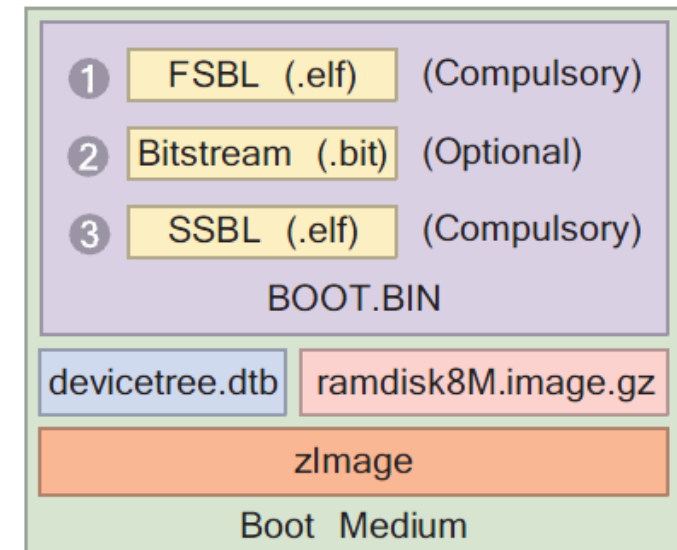
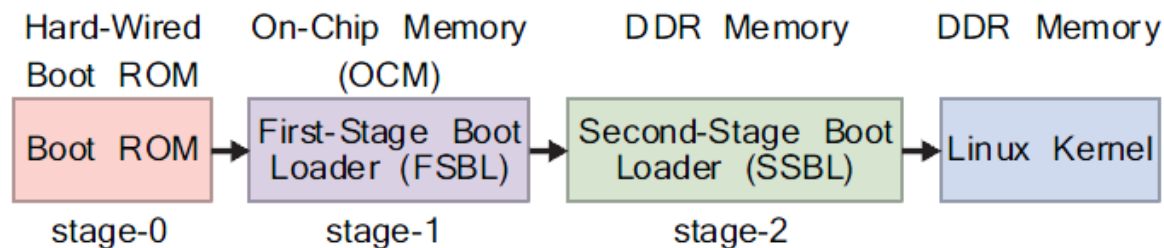
## ■ Overview

- Basic I/O System (BIOS)
- First Stage Boot Loader (FSBL)
- Second SBL (SSBL)
- Kernel
- Init



# Zynq Linux Booting

- Example Boot Files
  - BOOT.BIN
  - zImage
  - devicetree.dtb
  - ramdisk8M.image.gz



# Zynq Linux Boot Process

- Boot ROM
- FSBL
- SSBL
- Linux

