

# FPGA Tool Basic

---

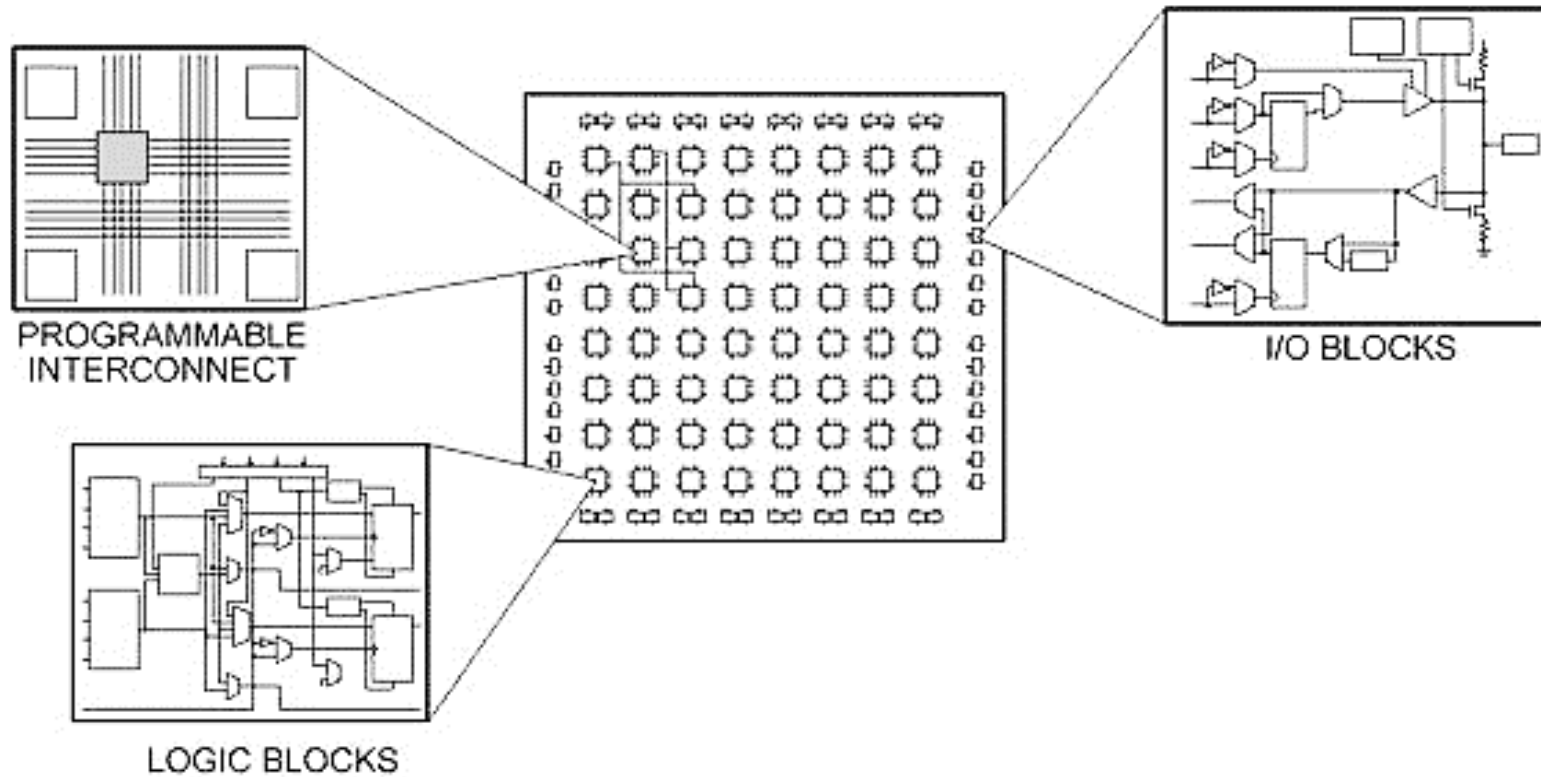
2023.07.19

Yi Chen

# What is FPGA?

- **FPGA(Field Programmable Gate Array)**

: Integrated Circuit designed to be configured by a customer or a designer after manufacturing.



# What is FPGA?

---

- **FPGA**

- Hardware Design Flow Capable with Single Tool (e.g. Xilinx or Quartus)
- Simple Design Revision
- More accessibility to IP
- Fixed Amount of Resource
- Limited Customization During Place & Route

- **ASIC**

- Multiple Tool for Each Design Steps (e.g. Cadence and Synopsys)
- High Cost for Hardware Revision
- Large Time Investment for Development
- Implementation of Desired Floorplan
- Full Customization and Control of Constraints

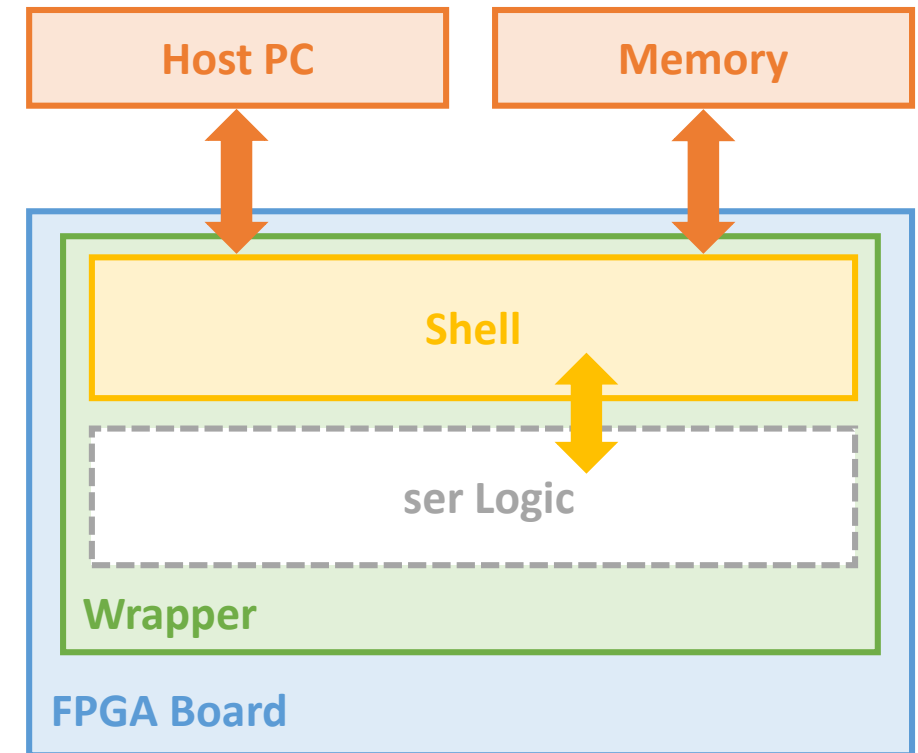
# FPGA Board Structure

- **FPGA Shell**

- : System-level component which administrates the basic operations of device such as host communication, external memory access
- : Contains PCIe controller, Memory controller, Bus, Clock manager

- **Top level Wrapper**

- : Verilog module which connects Shell & User Logic



# How to use Vivado?

- **Required files for synthesis and implementation**

- : Sources (Designed RTL files, IPs)

- : Constraints (Timing, Pin mapping, Pblock Setting, ... in .xdc files)

- **Steps to set up the project**

- 1. Unzip the zipped file

- 2. Move all of your working source codes of lab1 to 01\_RTL folder

- 3. Modify the vlist.f file accordingly to include all of your source codes.

Lab2 > FPGA\_TEMPLATE >

Name	Date modified	Type	Size
01_RTL	7/18/2023 11:03 AM	File folder	
02_TB	7/18/2023 11:03 AM	File folder	
03_LIST	7/18/2023 11:03 AM	File folder	
dma_constrs.xdc	1/13/2022 12:50 AM	XDC File	1 KB
mem0_init.mem	1/13/2022 12:50 AM	MEM File	1 KB
mem1_init.mem	1/13/2022 12:50 AM	MEM File	1 KB
shell.tcl	1/13/2022 12:56 PM	TCL File	15 KB
start_vivado.sh	1/13/2022 12:50 AM	SH Source File	1 KB

# How to use Vivado?

- Use the script to create the project, add constraints and pre-designed source

4. Script *shell.tcl* for generating pre-designed source (ex. Block Design) automatically

Make Tcl Script for customized Block Design using 'Export Block Design'

```
undergraduate@u50-server4:~/Yi$ ls
01_RTL  02_TB  03_LIST  dma_constrs.xdc  mem0_init.mem  mem1_init.mem  shell.tcl  start_vivado.sh
undergraduate@u50-server4:~/Yi$ ./start_vivado.sh
-bash: ./start_vivado.sh: Permission denied
undergraduate@u50-server4:~/Yi$ chmod 777 start_vivado.sh
undergraduate@u50-server4:~/Yi$ ./start_vivado.sh
undergraduate@u50-server4:~/Yi$
***** Vivado v2020.2 (64-bit)
**** SW Build 3064766 on Wed Nov 18 09:12:47 MST 2020
**** IP Build 3064653 on Wed Nov 18 14:17:31 MST 2020
** Copyright 1986-2020 Xilinx, Inc. All Rights Reserved.

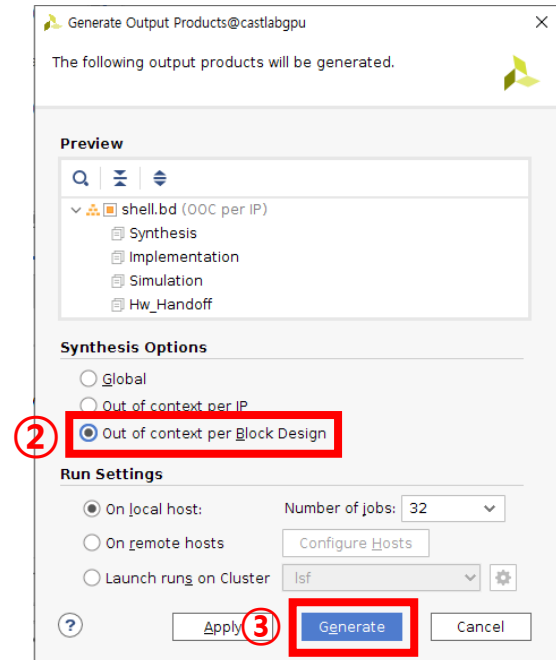
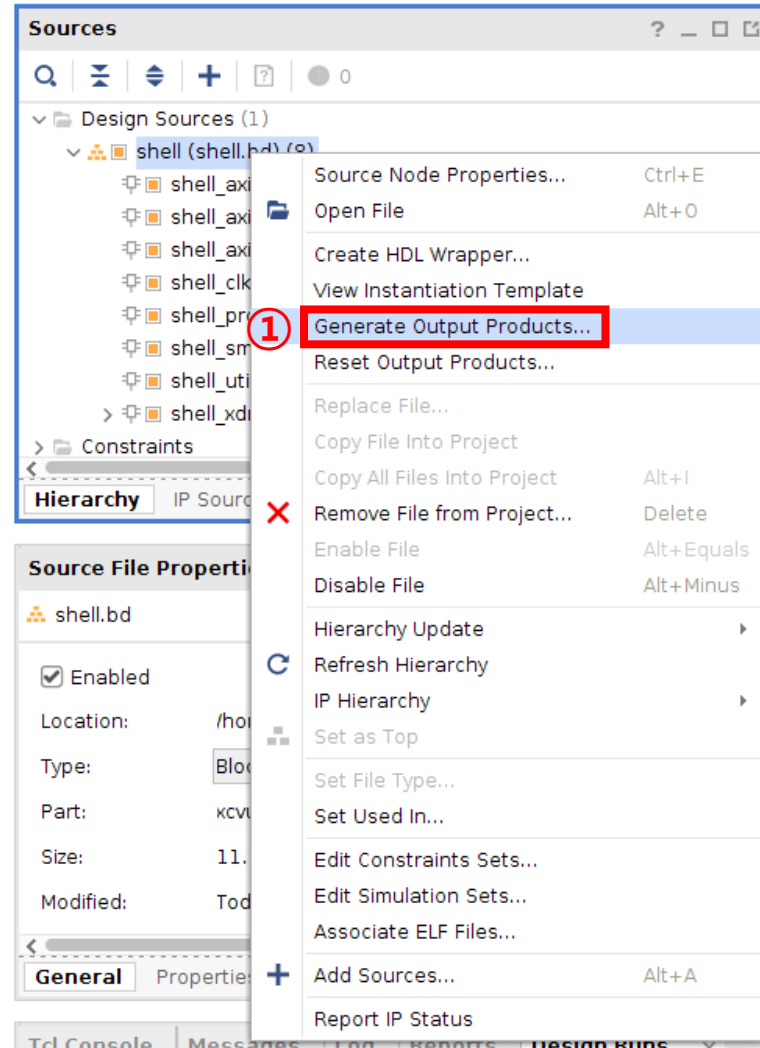
start_gui
```

# How to use Vivado?

- **Generate Output Products**

- : Convert Block Design to Verilog File

- : Generate Pre-Synthesized Module



# How to use Vivado?

- IP Generation

The screenshot displays the Vivado IDE interface. On the left, the 'Flow Navigator' pane shows the 'IP Catalog' option highlighted with a red box and a circled '1'. The main workspace is divided into several panes. The 'Sources' pane on the left shows the project hierarchy. The 'Properties' pane is empty. The 'IP Catalog' pane on the right is the primary focus, showing a search for 'Memory' with 26 matches. A red box and a circled '2' highlight the search input. Below the search bar, a table lists various IP blocks. A red box and a circled '3' highlight the 'Block Memory Generator' entry in the table.

**Search Results Table:**

Name	AXI4	Status	License	VLNV
<b>Vivado Repository</b>				
<b>AXI Infrastructure</b>				
AXI Central Direct Memory Access	AXI4	Production	Included	xilinx.com:ip:axi_cdma:4.1
AXI Direct Memory Access	AXI4, AXI4-Stream	Production	Included	xilinx.com:ip:axi_dma:7.1
AXI Memory Init	AXI4	Production	Included	xilinx.com:ip:axi_memory_init:1.0
AXI Memory Mapped to Stream Mapper	AXI4, AXI4-Stream	Production	Included	xilinx.com:ip:axi_mm2s_mapper:1.1
AXI Multi Channel Direct Memory Access	AXI4, AXI4-Stream	Production	Included	xilinx.com:ip:axi_mcdma:1.1
AXI Video Direct Memory Access	AXI4, AXI4-Stream	Production	Included	xilinx.com:ip:axi_vdma:6.3
SDx Memory Subsystem	AXI4	Beta	Included	xilinx.com:ip:sdx_memory_subsystem:1.0
<b>Memory Elements</b>				
Block Memory Generator	AXI4	Production	Included	xilinx.com:ip:blk_mem_gen:8.4
Distributed Memory Generator		Production	Included	xilinx.com:ip:dist_mem_gen:8.0



# How to use Vivado?

- IP Generation

Documentation IP Location Switch to Defaults

**IP Symbol** **Power Estimation**

☒ Show disabled ports

AXI\_SLAVE\_S\_AXI  
AXILite\_SLAVE\_S\_AXI  
+ BRAM\_PORTA  
+ BRAM\_PORTB

regcea sbiterr  
regceb dbiterr  
injectsbiterr rdaddress[15:0]  
injectdbiterr rsta\_busy  
eccpipece rstb\_busy  
sleep s\_axi\_sbiterr  
deepsleep s\_axi\_dbiterr  
shutdown s\_axi\_raddress[15:0]  
s\_ack  
s\_aresetn  
s\_axi\_injectsbiterr  
s\_axi\_injectdbiterr

Component Name blk\_mem\_gen\_0

**Basic** **Port A Options** **Other Options** **Summary**

Interface Type Native ☐ Generate address interface with 32 bits  
Memory Type Single Port RAM ☐ Common Clock

**ECC Options**

ECC Type No ECC  
☐ Error Injection Pins Single Bit Error Injection

**Write Enable**

☒ Byte Write Enable  
Byte Size (bits) 8

**Algorithm Options**

Defines the algorithm used to concatenate the block RAM primitives.  
Refer datasheet for more information.

Algorithm Minimum Area  
Primitive 8kx2

OK Cancel

④ Write Enable & Configure Byte Size

# How to use Vivado?

- IP Generation

The screenshot shows the Vivado IP configuration window for the component `blk_mem_gen_1`. The window is divided into two main panes. The left pane, titled 'IP Symbol', shows a tree view of the IP's internal components, including `AXI_SLAVE_S_AXI`, `AXILite_SLAVE_S_AXI`, `BRAM_PORTA`, and `BRAM_PORTB`. The right pane, titled 'Power Estimation', shows the configuration options for the component. The 'Basic' tab is selected, and the 'Memory Size' section is expanded. The 'Write Width' is set to 32, 'Read Width' is 32, 'Write Depth' is 28, and 'Read Depth' is 28. The 'Operating Mode' is set to 'Write First' and 'Enable Port Type' is 'Use ENA Pin'. The 'Port A Optional Output Registers' section shows that the 'Primitives Output Register' is selected. The 'Port A Output Reset Options' section shows that the 'RSTA Pin (set/reset pin)' is selected. The 'READ Address Change A' section shows that the 'Read Address Change A' option is selected. A red circle with the number 5 is placed next to the 'Memory Size' section, and a red circle with the number 6 is placed next to the 'Port A Optional Output Registers' section. A tooltip for the 'Primitives Output Register' checkbox explains its function: 'Adds register stage at primitive output of port A. This enables the embedded output registers in the block RAM primitives. (See the data sheet for more information.)' The window has 'OK' and 'Cancel' buttons at the bottom right.

Documentation IP Location Switch to Defaults

Component Name `blk_mem_gen_1`

**Basic** Port A Options Other Options Summary

**Memory Size**

Write Width 32 Range: 8 to 4096 (bits)

Read Width 32

Write Depth 28 Range: 2 to 1048576

Read Depth 28

Operating Mode Write First Enable Port Type Use ENA Pin

**Port A Optional Output Registers**

☒ Primitives Output Register ☐ Core Output Register

☐ SoC Adds register stage at primitive output of port A. This enables the embedded output registers in the block RAM primitives. (See the data sheet for more information.)

**Port A Output Reset Options**

☐ RSTA Pin (set/reset pin) Output Reset Value (Hex) 0

☐ Reset Memory Latch Reset Priority CE (Latch or Register Enable)

**READ Address Change A**

☐ Read Address Change A

OK Cancel

# How to use Vivado?

- IP Generation

Documentation IP Location Switch to Defaults

**IP Symbol** **Power Estimation**

☒ Show disabled ports

+ AXI\_SLAVE\_S\_AXI  
+ AXILite\_SLAVE\_S\_AXI  
+ **BRAM\_PORTA**  
+ BRAM\_PORTB

regcea sbiterr  
regceb dbiterr  
injectsbiterr rdaddressc[15:0]  
injectdbiterr rsta\_busy  
eccpipece rstb\_busy  
sleep s\_axi\_sbiterr  
deepsleep s\_axi\_dbiterr  
shutdown s\_axi\_rdaddressc[15:0]  
s\_aclk  
s\_aresetn  
s\_axi\_injectsbiterr  
s\_axi\_injectdbiterr

Component Name blk\_mem\_gen\_0

**Basic** **Port A Options** **Other Options** **Summary**

Pipeline Stages within Mux 0 Mux Size: 4x1

**Memory Initialization**

☒ Load Init File

Coe File ../../Freshman/FPGA\_TEMPLATE/01\_RTL/mem0.coe Browse Edit

☐ Fill Remaining Memory Locations

Remaining Memory Locations (Hex) 0

**Structural/UniSim Simulation Model Options**

Defines the type of warnings and outputs are generated when a read-write or write-write collision occurs.

Collision Warnings All

**Behavioral Simulation Model Options**

☐ Disable Collision Warnings ☐ Disable Out of Range Warnings

**Dynamic Power Saving**

☐ Sleep

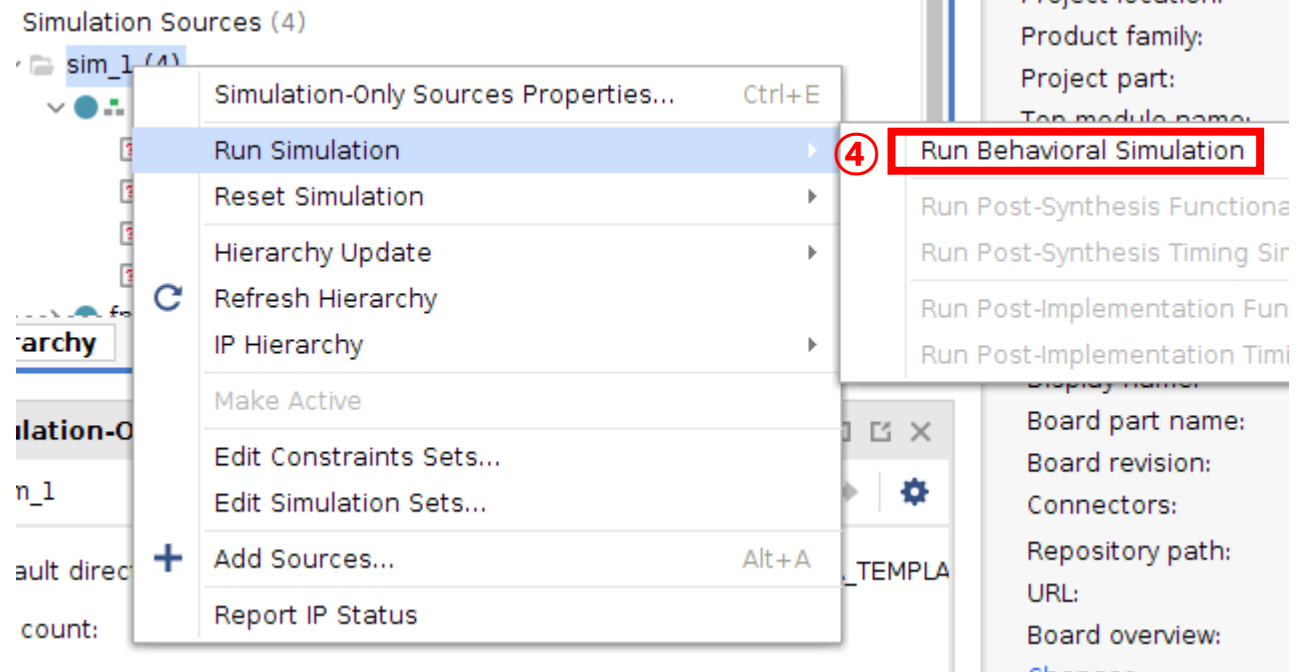
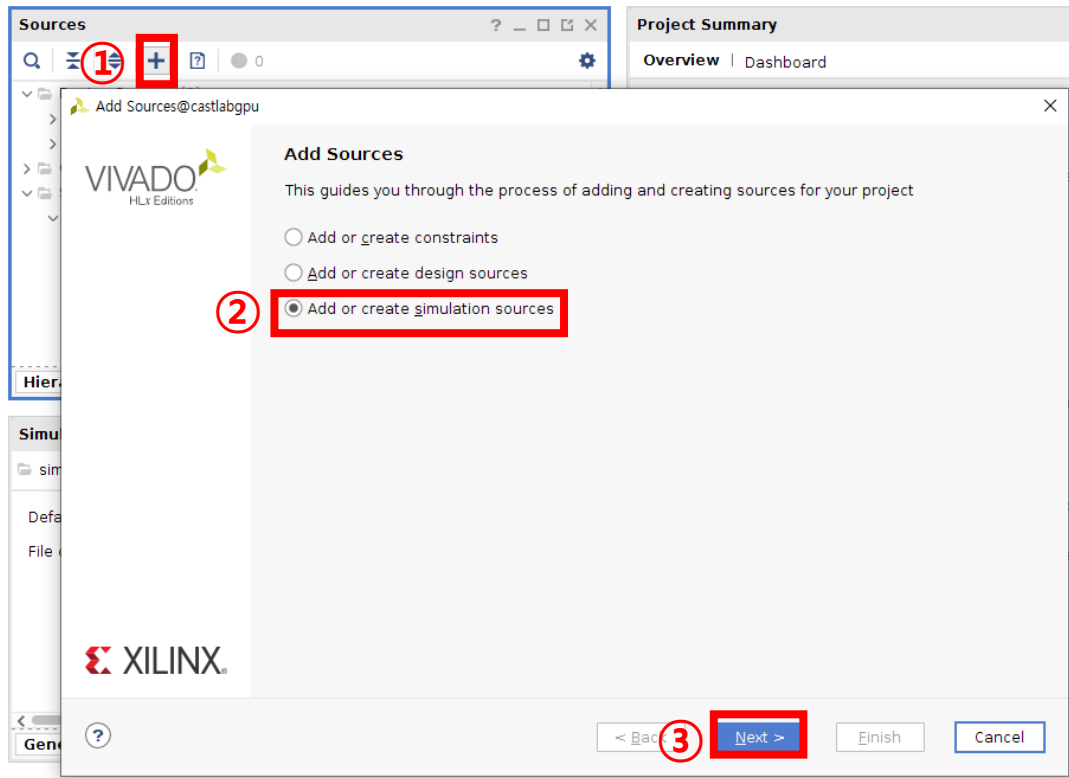
OK Cancel

⑦ Load .coe file

# How to use Vivado?

- **Functional Simulation**

: Simulate Functional Logic through customized Testbench



# How to use Vivado?

- Synthesis & Implementation & Generate Bitstream

The screenshot shows the Vivado IDE interface with the following components:

- Flow Navigator:** A sidebar on the left with a tree view. The 'SYNTHESIS' section is expanded, and 'Run Synthesis' is highlighted with a red box. Below it, 'IMPLEMENTATION' is expanded, and 'Run Implementation' is highlighted with a red box. At the bottom, 'PROGRAM AND DEBUG' is expanded, and 'Generate Bitstream' is highlighted with a red box.
- Project Manager - IP\_Gen:** A window showing the project structure. The 'Sources' tab is active, displaying a list of sources: Design Sources (11), Constraints, Simulation Sources (10), and Utility Sources. The 'Hierarchy' tab is selected, showing a tree view of the project hierarchy.
- IP Catalog:** A window showing the IP catalog. The 'Cores' tab is active, displaying a list of cores. The search bar contains 'Memory', and there are 26 matches. The table below shows the results:

Name	AXI4	Status	License	VLNV
Vivado Repository				
AXI Infrastructure				
AXI Central Direct Memory Access	AXI4	Production	Included	xilinx.com:ip:axi_cdma:4.1
AXI Direct Memory Access	AXI4, AXI4-Stream	Production	Included	xilinx.com:ip:axi_dma:7.1
AXI Memory Init	AXI4	Production	Included	xilinx.com:ip:axi_memory_init:1.0
AXI Memory Mapped to Stream Mapper	AXI4, AXI4-Stream	Production	Included	xilinx.com:ip:axi_mm2s_mapper:1.1
AXI Multi Channel Direct Memory Access	AXI4, AXI4-Stream	Production	Included	xilinx.com:ip:axi_mcdma:1.1
AXI Video Direct Memory Access	AXI4, AXI4-Stream	Production	Included	xilinx.com:ip:axi_vdma:6.3
SDx Memory Subsystem	AXI4	Beta	Included	xilinx.com:ip:sdx_memory_subsystem:1.0
Basic Elements				
Memory Elements				
Block Memory Generator	AXI4	Production	Included	xilinx.com:ip:blk_mem_gen:8.4
Distributed Memory Generator		Production	Included	xilinx.com:ip:dist_mem_gen:8.0

Red text labels with arrows pointing to the highlighted options in the Flow Navigator:

- Click for Synthesis (pointing to Run Synthesis)
- Click for Implementation (pointing to Run Implementation)
- Click for Bitstream (pointing to Generate Bitstream)

# Integrated Logic Analyzer

---

- **ILA(Integrated Logic Analyer)**

: The customizable Integrated Logic Analyzer (ILA) IP core is a logic analyzer that can be used to monitor the internal signals of a design.

: The ILA core includes many advanced features of modern logic analyzers, including boolean trigger equations and edge transition triggers. Because the ILA core is synchronous to the design being monitored, all design clock constraints that are applied to your design are also applied to the components of the ILA core.

# How to make ILA?

- Before Synthesis

: Write (\* MARK\_DEBUG="true" \*) ahead of input/output port which you want to check @ RTL

: Clock & Reset signals are not allowed to make ILA

```
// Memory 0 (Base Address : 0x0010_0000)
output logic                                mem0_en,
output logic [MEM_STRB_WIDTH-1:0]          mem0_we,
output logic [MEM_ADDR_WIDTH-1:0]          mem0_addr,
output logic [MEM_DATA_WIDTH-1:0]          mem0_wdata,
(* MARK_DEBUG="true" *) input logic [MEM_DATA_WIDTH-1:0] mem0_rdata,

// Memory 1 (Base Address : 0x0020_0000)
output logic                                mem1_en,
output logic [MEM_STRB_WIDTH-1:0]          mem1_we,
output logic [MEM_ADDR_WIDTH-1:0]          mem1_addr,
output logic [MEM_DATA_WIDTH-1:0]          mem1_wdata,
input logic [MEM_DATA_WIDTH-1:0]          mem1_rdata,

(* MARK_DEBUG="true" *) output logic [1:0] led
```

→ Probe

→ Probe

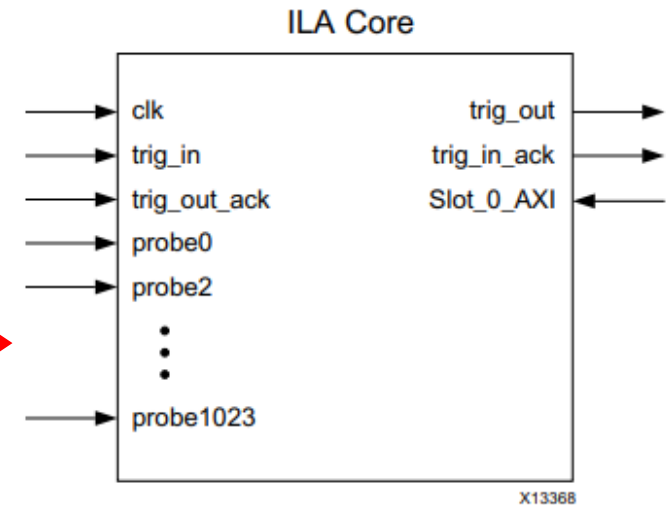


Figure 1-1: ILA Core Symbol

# How to make ILA?

- After Synthesis

## SYNTHESIS

① Run Synthesis

② Open Synthesized Design

Constraints Wizard

Edit Timing Constraints

③ Set Up Debug

Report Timing Summary

Report Clock Networks

Report Clock Interaction

Report Methodology

Report DRC

Report Noise

Report Utilization

Report Power

Schematic

Set Up Debug@castlabgpu

### Nets to Debug

The nets below will be debugged with ILA cores. To add nets click "Find Nets to Add". You can also select nets in the Netlist or other windows, then drag them to the list or click "Add Selected Nets".

Name	Clock Domain	Driver Cell	Probe T
> User_Logic_I/SLR0_I/HostControllerInst/configuration_data_out (256)	U50_CLK_WIZ_I/SLR0_CLK_WIZ/inst/CLK_CORE_DRP_I/clk_inst/SLR0_CLK	FDRE	Data ar
> User_Logic_I/SLR0_I/dma_ar_enable (2)	U50_CLK_WIZ_I/SLR0_CLK_WIZ/inst/CLK_CORE_DRP_I/clk_inst/SLR0_CLK	(Multiple)	Data ar
> User_Logic_I/SLR0_I/dma_ar_status (2)	U50_CLK_WIZ_I/SLR0_CLK_WIZ/inst/CLK_CORE_DRP_I/clk_inst/SLR0_CLK	FDRE	Data ar
> User_Logic_I/SLR0_I/dma_aw_enable (2)	U50_CLK_WIZ_I/SLR0_CLK_WIZ/inst/CLK_CORE_DRP_I/clk_inst/SLR0_CLK	LUT2	Data ar
> User_Logic_I/SLR0_I/dma_aw_status (2)	U50_CLK_WIZ_I/SLR0_CLK_WIZ/inst/CLK_CORE_DRP_I/clk_inst/SLR0_CLK	FDRE	Data ar
> User_Logic_I/SLR0_I/dma_usr_read_enable (2)	U50_CLK_WIZ_I/SLR0_CLK_WIZ/inst/CLK_CORE_DRP_I/clk_inst/SLR0_CLK	FDRE	Data ar
> User_Logic_I/SLR0_I/dma_usr_read_status (2)	U50_CLK_WIZ_I/SLR0_CLK_WIZ/inst/CLK_CORE_DRP_I/clk_inst/SLR0_CLK	FDRE	Data ar
> Select Clock Domain@castlabgpu			

The list below contains 'ALL\_CLOCK' nets.  
To see other types of clock nets use the drop-down button.

Find

ALL\_CLOCK Search hierarchically

- U50\_CLK\_WIZ\_I/HBM\_CLK\_WIZ/inst/CLK\_CORE\_DRP\_I/clk\_inst/clk\_out1
- U50\_CLK\_WIZ\_I/HBM\_REF\_CLK\_WIZ/inst/HBM\_REF\_CLK1\_U50\_CLK\_WIZ\_HBM
- U50\_CLK\_WIZ\_I/HBM\_REF\_CLK\_WIZ/inst/HBM\_REF\_CLK1
- U50\_CLK\_WIZ\_I/HBM\_REF\_CLK\_WIZ/inst/HBM\_REF\_CLK0\_U50\_CLK\_WIZ\_HBM
- U50\_CLK\_WIZ\_I/HBM\_REF\_CLK\_WIZ/inst/HBM\_REF\_CLK0
- U50\_CLK\_WIZ\_I/CMC\_CLK\_WIZ/inst/clk\_out2
- U50\_CLK\_WIZ\_I/CMC\_CLK\_WIZ/inst/clk\_out1\_BUFGE
- U50\_CLK\_WIZ\_I/CMC\_CLK\_WIZ/inst/clk\_out2\_BUFGE
- U50\_CLK\_WIZ\_I/CMC\_CLK\_WIZ/inst/clk\_out1
- U50\_CLK\_WIZ\_I/SLR0\_CLK\_WIZ/inst/CLK\_CORE\_DRP\_I/clk\_inst/SLR0\_CLK
- U50\_CLK\_WIZ\_I/SLR0\_CLK\_WIZ/inst/CLK\_CORE\_DRP\_I/clk\_inst/SLR0\_CLK

OK Cancel

④ Select proper Clock Domain

### ILA Core Options

Choose features for the ILA debug cores.

⑤ Sample of data depth: 1024  
Input pipe stages: 0

### Trigger and Storage Settings

☐ Capture control  
☐ Advanced trigger

⑥ Impl. & generate Bitstream

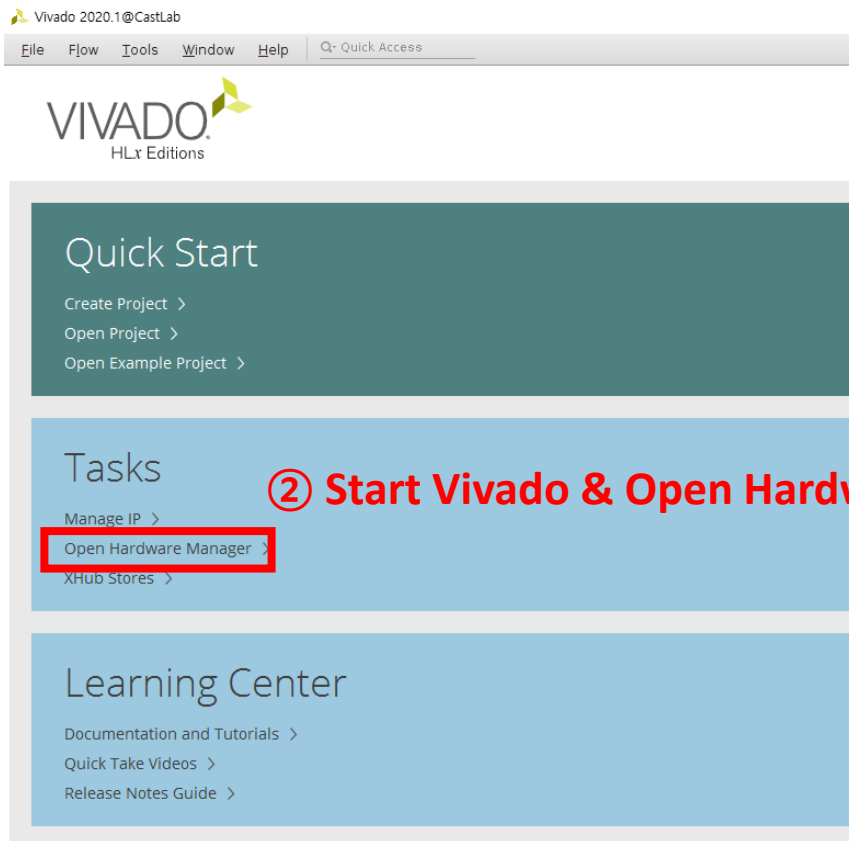


# How to do Board Verification?

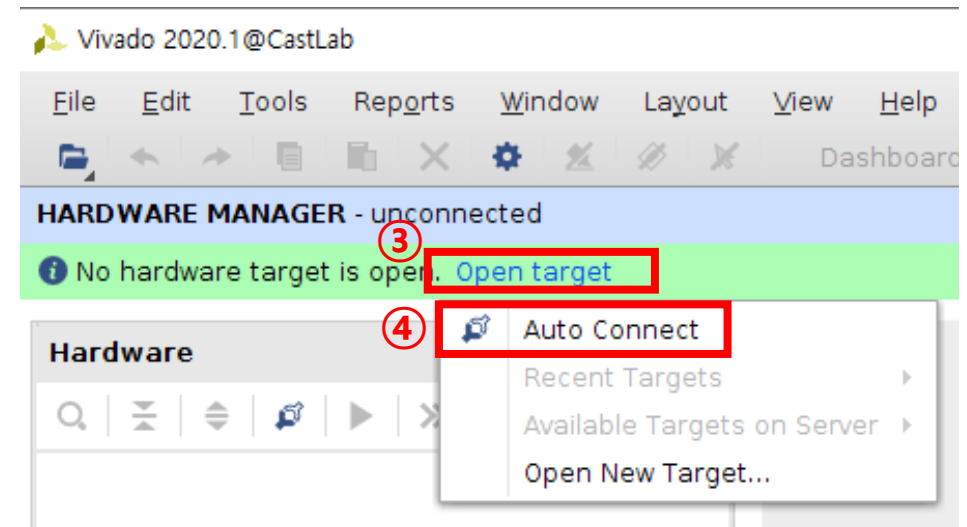
- Program Device

② Move your own bitstream (.bit) to test PC server (ILA (.ltx) file if necessary)

- File Location : \$project\_folder/\$project\_name/\$project\_name.runs/impl\_1



② Start Vivado & Open Hardware Manager



# How to do Board Verification?

- Program Device

The screenshot illustrates the process of programming a device in the Xilinx Vivado IDE. The 'Hardware' window on the left shows a tree view of the hardware components. The 'xcvu9p\_0' device is selected, and a context menu is open with 'Program Device...' highlighted. The 'Program Device' dialog box is shown on the right, with the 'Bitstream file' field highlighted. The 'Debug probes file' field is also highlighted. The 'Enable end of startup check' checkbox is checked. The 'Program' button is visible at the bottom right of the dialog box.

**Hardware Window:**

Name	Status
localhost (2) <b>VCU118</b>	Connected
xilinx_tcf/Digilent/210308AC64	Open
xcvu9p_0	

**Program Device Dialog:**

Select a bitstream programming file and download it to your hardware device. You can optionally select a debug probes file that corresponds to the debug cores contained in the bitstream programming file.

Bitstream file:

Debug probes file:

☒ Enable end of startup check

**Program** **Cancel**

# How to do Board Verification?

---

- **Test PC setting**

: Connect Host PC to Programmed FPGA Device

- File Location : \$test PC server(ID : freshman)/Workplace/Freshman-Curriculum/test/driver\_command.txt

```
cd /home/members/freshman/Workplace/XilinxAR65444/Linux/Xilinx_Answer_65444_Linux_Files/tests
sudo make clean

cd /home/members/freshman/Workplace/XilinxAR65444/Linux/
sudo ./build-install-driver-linux.sh

-

sudo reboot

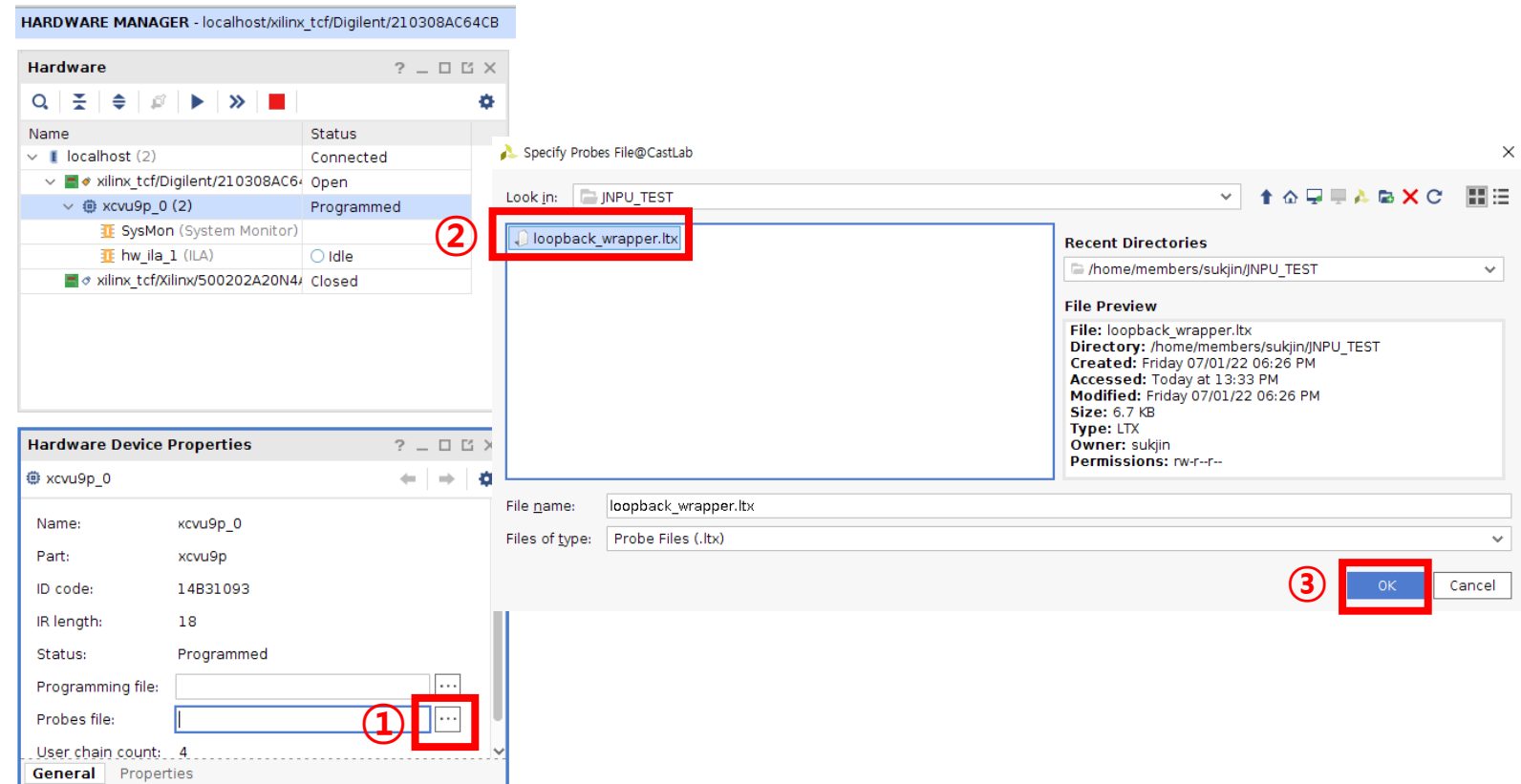
cd /home/members/freshman/Workplace/XilinxAR65444/Linux/Xilinx_Answer_65444_Linux_Files/tests
sudo ./load_driver.sh
```

# How to do Board Verification?

- Set ILA waveform

: After setting up test PC, Open Vivado and Do Program Device process 2,3,4 again (See previous Slide for Program Device)

: Load ILA file(.ltx) to ILA waveform



# How to do Board Verification?

- ILA waveform

: ILA Setting – Choose number & position of samples you want

: Trigger Setup – Set the specific conditions which trigger to get samples

The screenshot displays the ILA interface for 'hw\_ila\_1'. The main window shows a waveform with a time axis from 0 to 50. A table on the left lists the captured signals, with the first four signals highlighted by a red box and labeled 'Probe Signal' in red text:

Name	Value
u_shell_wrapper/shell...ect_0/S00_AXI_awready	
u_shell_wrapper/shell...ect_0/S00_AXI_awvalid	
u_shell_wrapper/shell...ect_0/S00_AXI_wready	
u_shell_wrapper/shell...ect_0/S00_AXI_wvalid	

Below the waveform, two panels are visible, both highlighted with red boxes and labeled in red text:

- ILA Setting:** The 'Settings - hw\_ila\_1' panel shows configuration options: Number of windows: 1, Window data depth: 2048, Trigger position in window: 1,024, and General Settings with a Refresh rate of 500 ms.
- Trigger Setup:** The 'Trigger Setup - hw\_ila\_1' panel shows a trigger condition: Name: u\_shell\_wrapper/shell\_i/smartconnect\_0/S00\_AXI\_wvalid, Operator: ==, and Radix: [B].

# How to do Board Verification?

- ILA waveform

hw\_ila\_1

Waveform - hw\_ila\_1

ILA Status: Idle

Name	Value
test_clk_out_tmp	
test_done_in	
test_en_tmp	
test_pin1_in_tmp	
test_pin1_out_tmp	
test_pin2_in_tmp	
test_pin2_out_tmp	
test_start_out	

Settings - hw\_ila\_1

Core status: Idle

Capture status: Window 1 of 1

Window sample 0 of 1024

Idle

Trigger Setup - hw\_ila\_1

Name	Operator	Radix	Value	Port	Comparator
test_done_in	==	[B]	X (don't care)	probel[0]	

1. Select Signal what you want

2. Select Trigger Condition

3. Select Trigger Condition

4. Write Command at Host PC

# How to do Board Verification?

- **Command for Host PC**

: Command to write & read data through AXI\_LITE & AXI

- File Location : \$test PC server(ID : freshman)/Workplace/Freshman-Curriculum/test/driver\_command.txt

```
reg_rw      program
xdma1_user  xdma name, 0 for u50, 1 for vcu118
0x00000000  address
w           r/w
0x00000007  value, if none, read
```

```
./reg_rw /dev/xdma1_user 0x00000000 w 0x00000007
```

 ① AXI\_LITE Read/Write Command

```
dma_to_device
dma_from_device      program
-d                   xdma name
xdma1_h2c_0
xdma1_c2h_0
-f                   file
-s                   size
-a                   address
```

② AXI Read/Write Command

```
./dma to device -d /dev/xdma1 h2c 0 -f data/in helloworld.bin -s 0x00000100 -a 0x00100000 -c 1
./dma from device -d /dev/xdma1 c2h 0 -f result/tmp test hi.bin -s 0x00000010 -a 0x00100000 -c 1
```

# Reference

---

- For detailed information, please refer to the '*XilinxDesignFlow\_v1*' file on DB server



# Revision History

---

- **Version 1.0 : 21/01/19, Sukbin Lim**
- **Version 1.1 : 21/10/01, Juyeong Yoon**
- **Version 2.0 : 22/07/12, Sukjin Lee**
- **Version 2.1 : 23/07/11, Yi Chen**