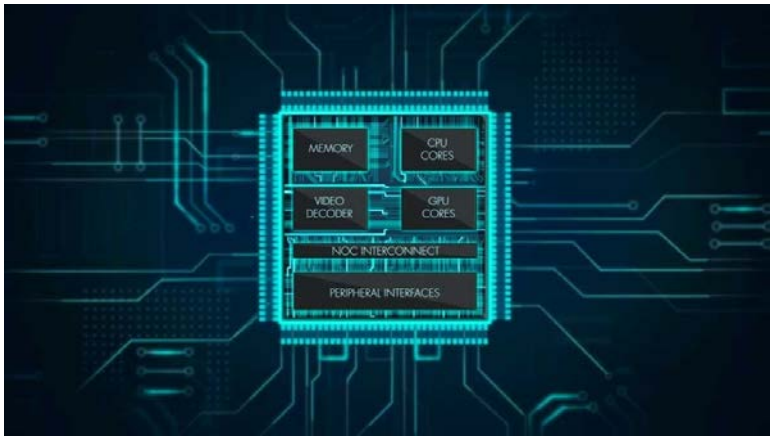
Digital System Design

Term-Project



Instructor : **Prof. Hyun Kim**

Lecture Note : eclass.seoultech.ac.kr

Office : Mirae Hall #526

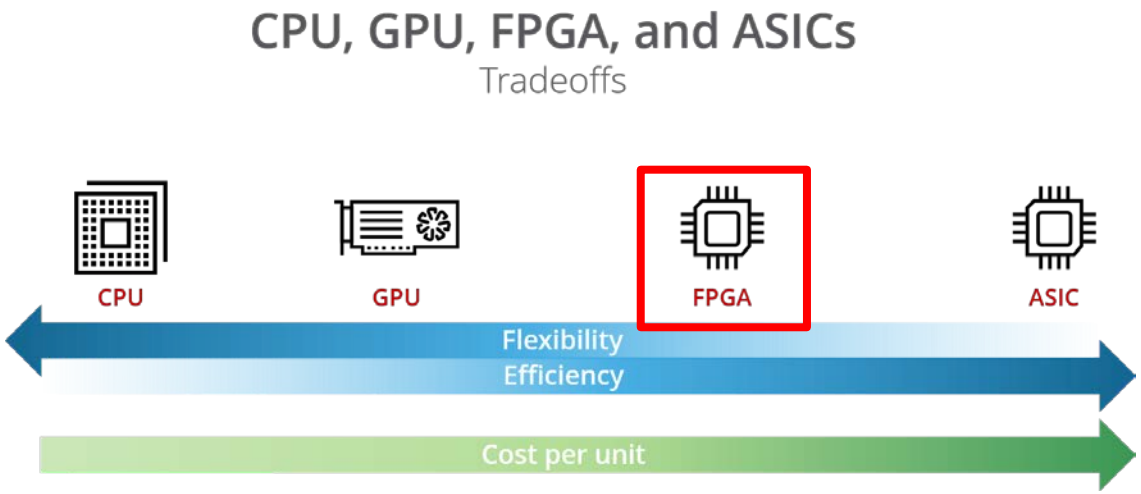
Web : idsl.seoultech.ac.kr

Contact : hyunkim@seoultech.ac.kr / 010-9600-5427

Contents

What is a FPGA?

- Field Programmable Gate Array: FPGA
- FPGA is a software-reconfigurable hardware substrate
  - Reconfigurable functions
  - Reconfigurable interconnection of functions
  - Reconfigurable input/output (IO)
  - …
- When used well, FPGAs:
  - Provide higher performance than running software on a CPU
  - Provide more flexibility of implementation than dedicated hardware on a CPU



FPGA Vendros

Top 2 FPGA Vendors

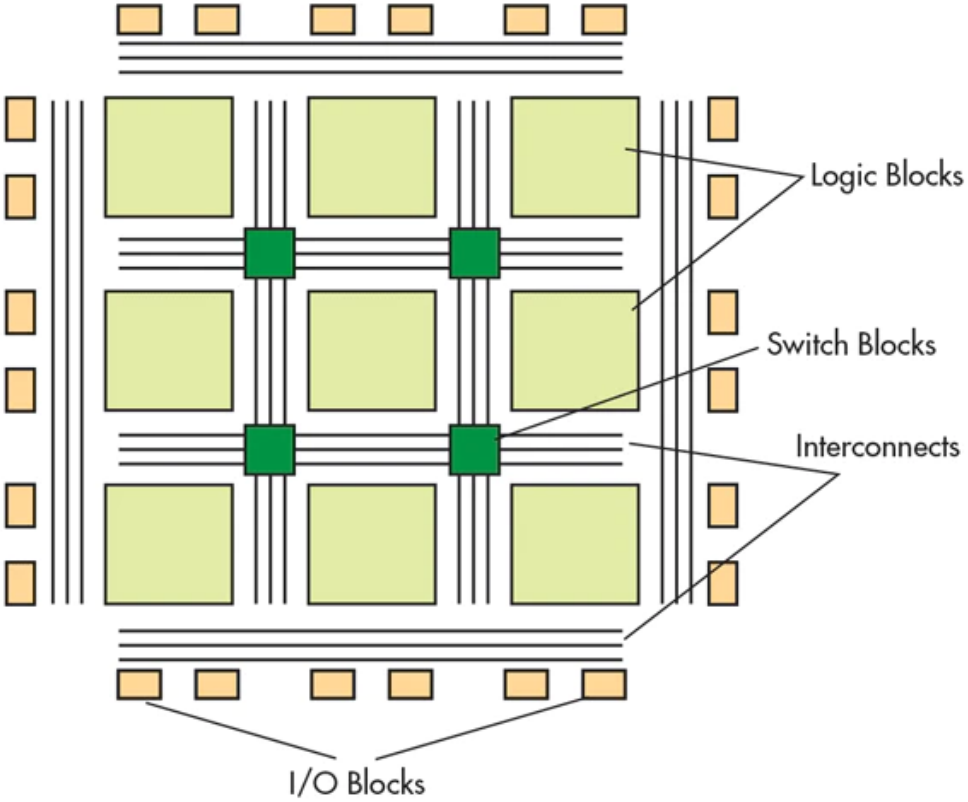
Market Share

Top 5 Company's Sales of FPGA/PLD, in All Applications, Worldwide, (Millions of U.S. Dollars)

Rank 2017	Rank 2018	Company Name	2017	2018	2018 Change	Share of Market
1	1	Xilinx	2,476	2,904	17.3%	51.1%
2	2	Intel	1,858	2,033	9.4%	35.8%
3	3	Microchip (formerly Microsemi)	321	376	17.1%	6.6%
4	4	Lattice Semiconductor	261	285	9.2%	5.0%
		Others	71	85	19.7%	1.5%
		Total Market	**4,987**	**5,683**	**14.0%**	**100.0%**

Source: Gartner (April 2019)

A High-Level Overview of FPGAs

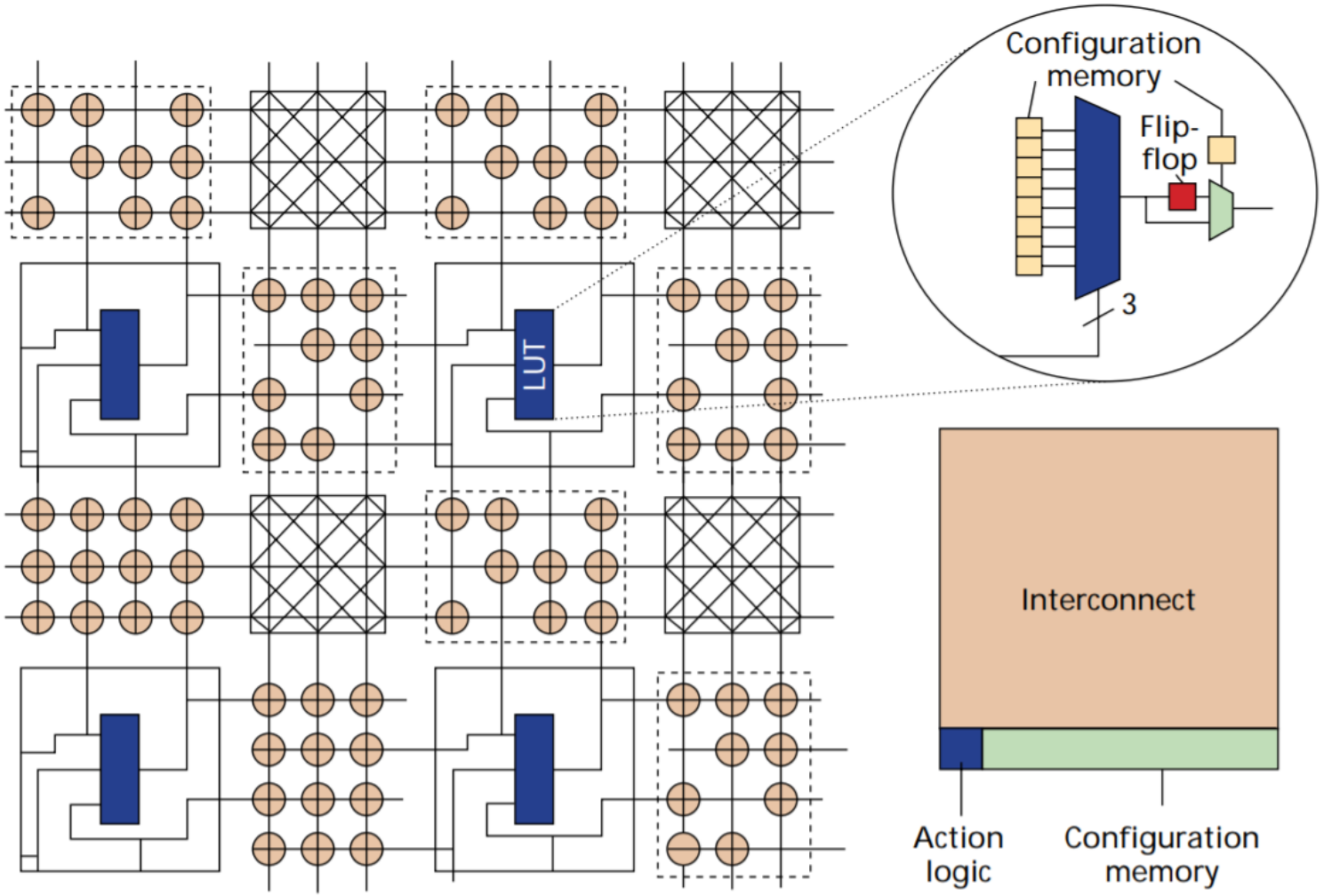


- We configure logic blocks, their connections, and IO blocks to create hardware circuits and map programs onto those circuits

Image source: https://www.electronicdesign.com/technologies/fpgas/article/21801527/the-principles-of-fpgas

FPGA Architecture

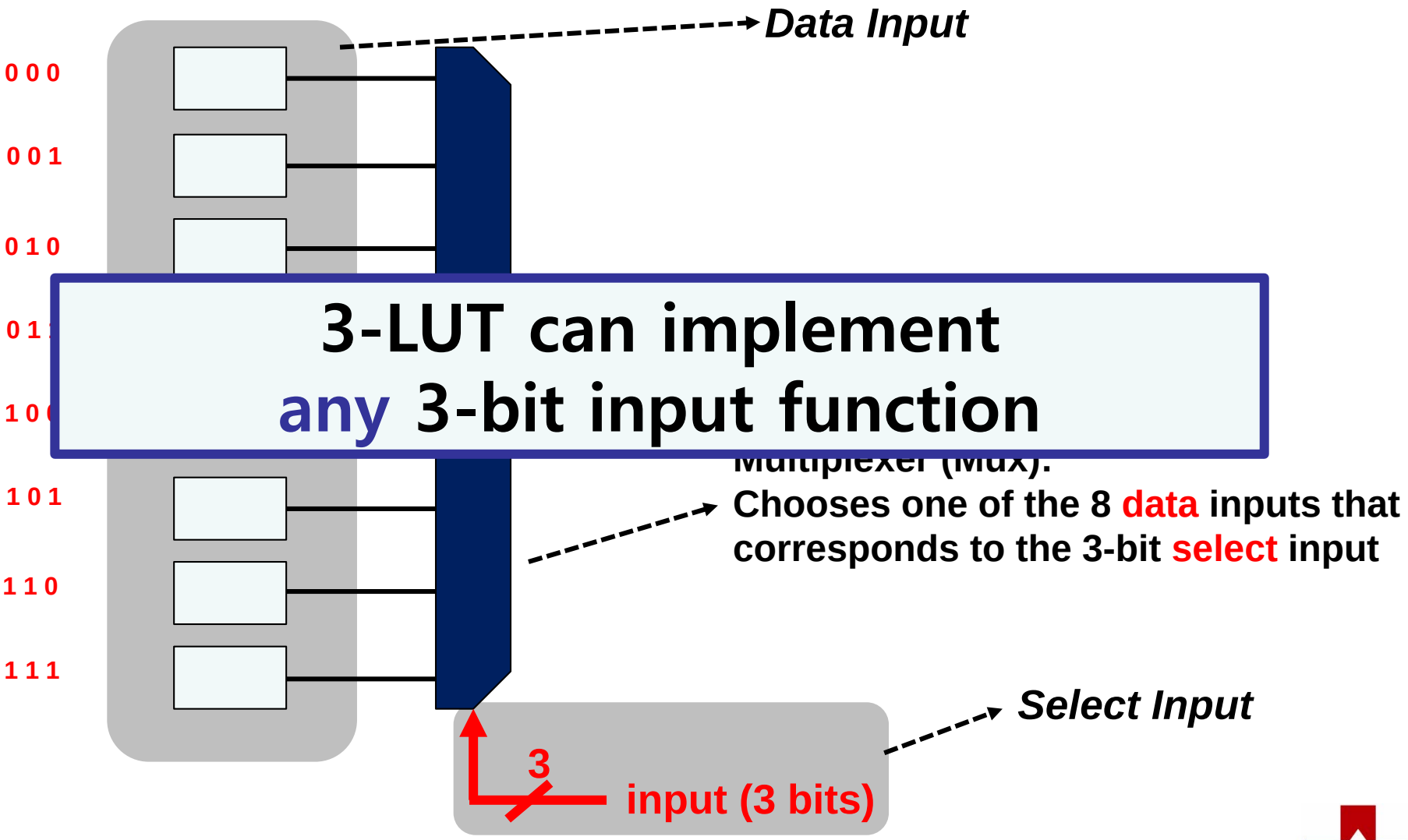
- Lookup Tables(LUT) and Switches



Andre DeHon, "The Density Advantage of Configurable Computing", *Computer*, 2000

Programming LUT

- 3-bit input LUT



An Example of Programming a LUT

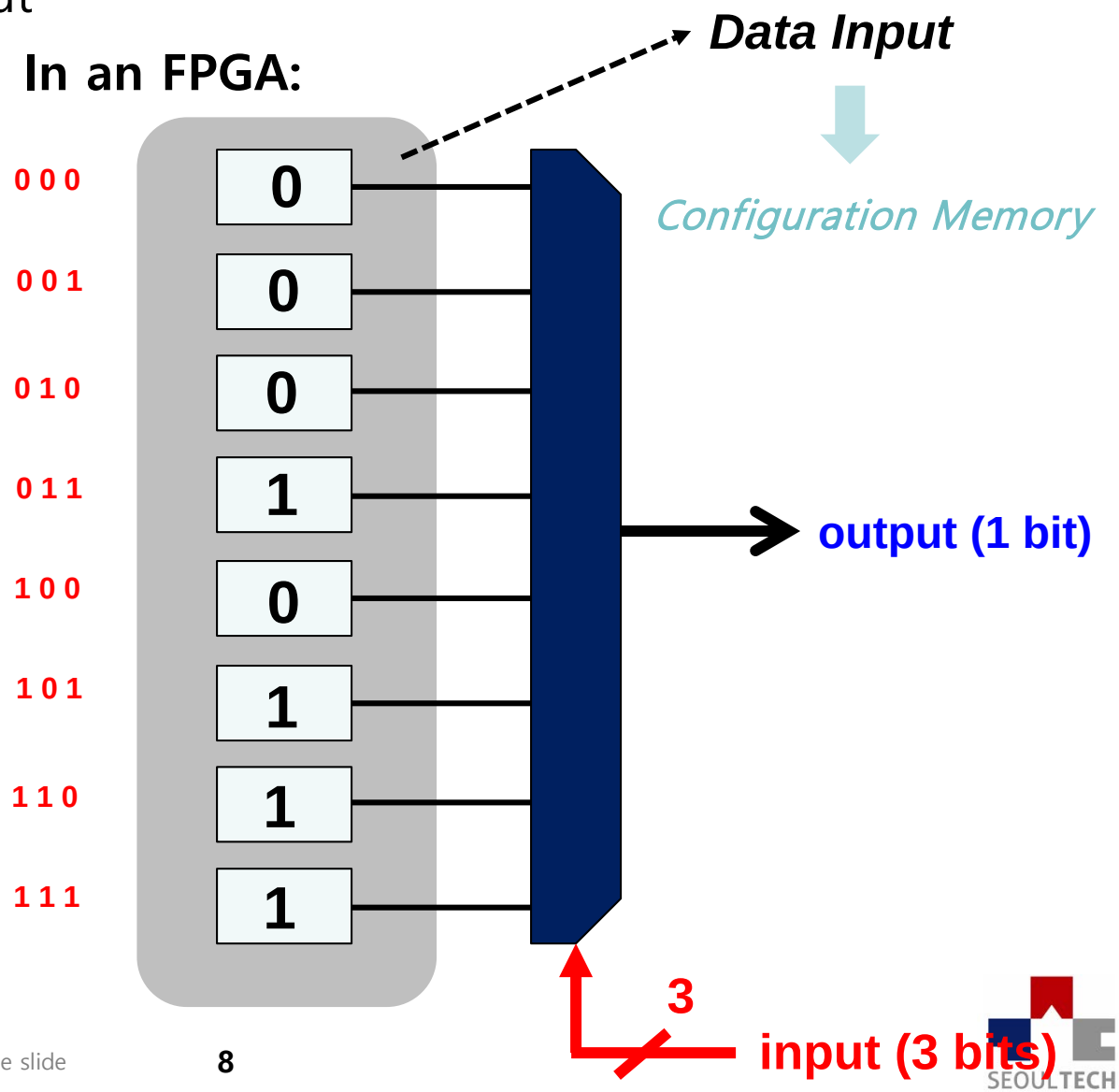
- Implementing a function that outputs '1' when there are at least two '1's in a 3-bit input

In C:

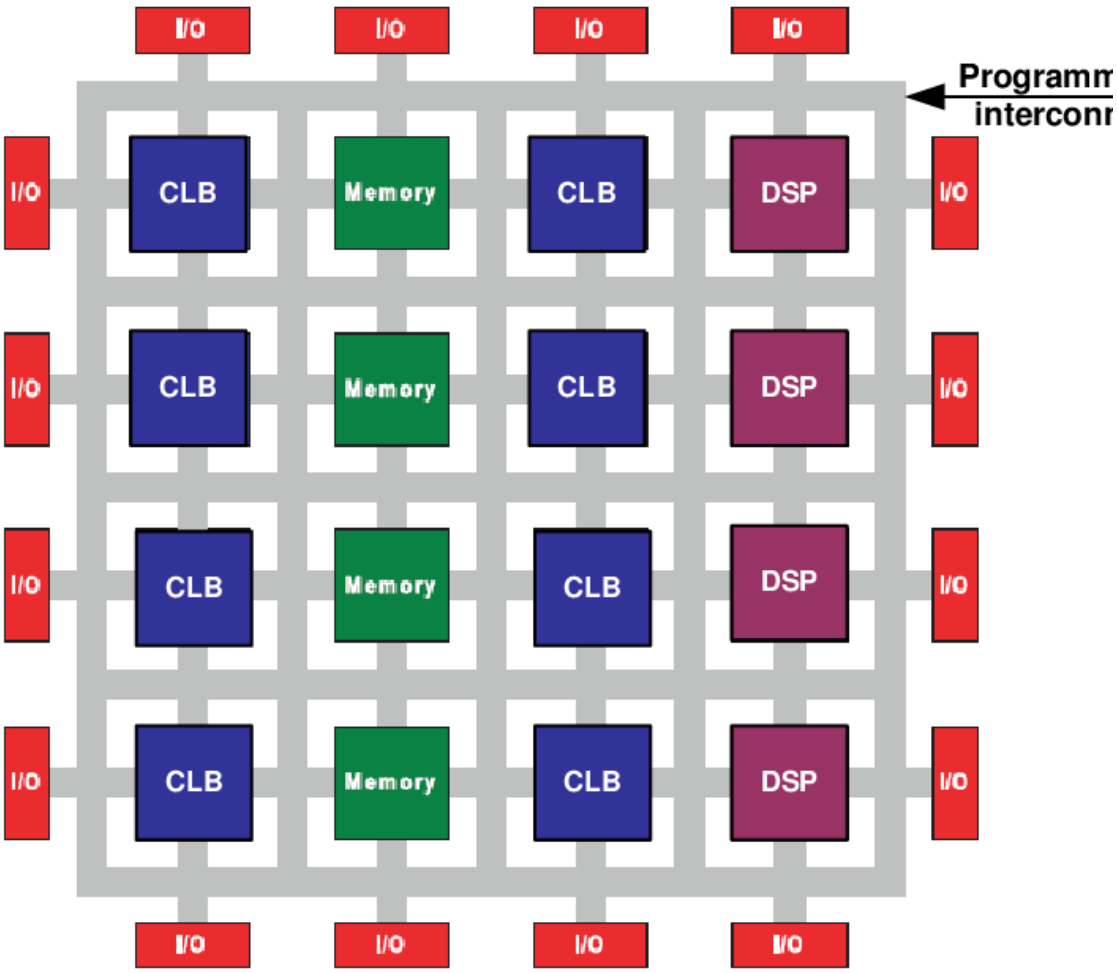
```
int count = 0;
for(int i = 0; i < 3; i++) {
    count += input & 1;
    input = input >> 1;
}

if(count > 1) return 1;

return 0;
```

```
switch(input){
   case 0:
   case 1:
   case 2:
   case 4:
      return 0;
   default:
      return 1;}
```

In an FPGA:



Xilinx's FPGA Architecture



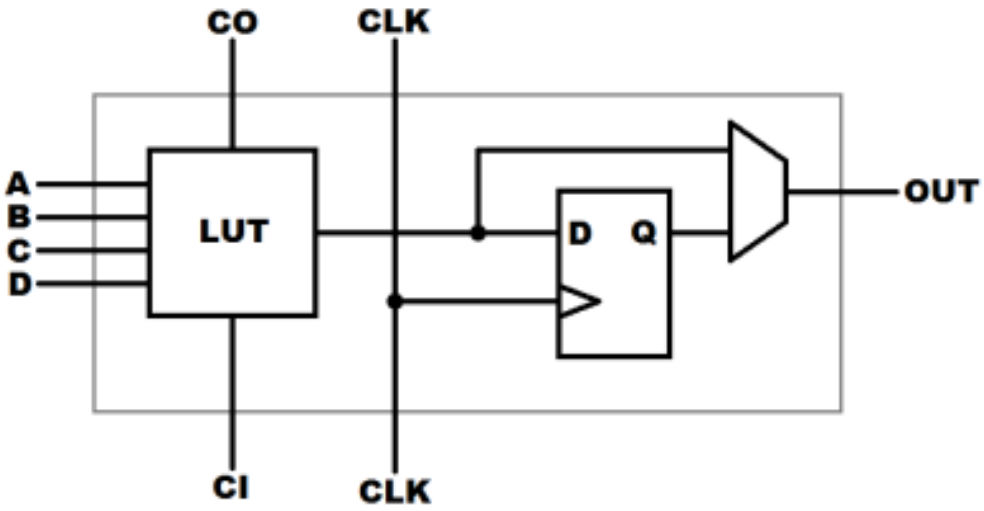
Configurable Logic Block (CLB)

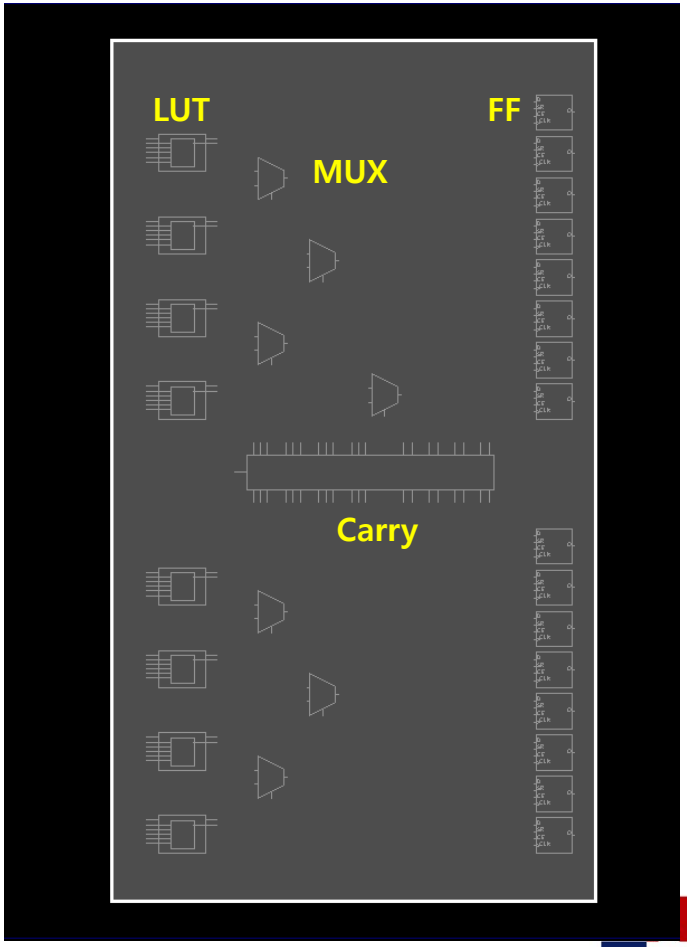
Image source: In book: Digital and Analogue Electronics Circuits and Systems (pp.53)

Components of FPGAs

CLB (Configurable Logic Block)

An individual CLB consists of a number of discrete logic components itself, such as look-up tables(LUTs) and flip-flops.

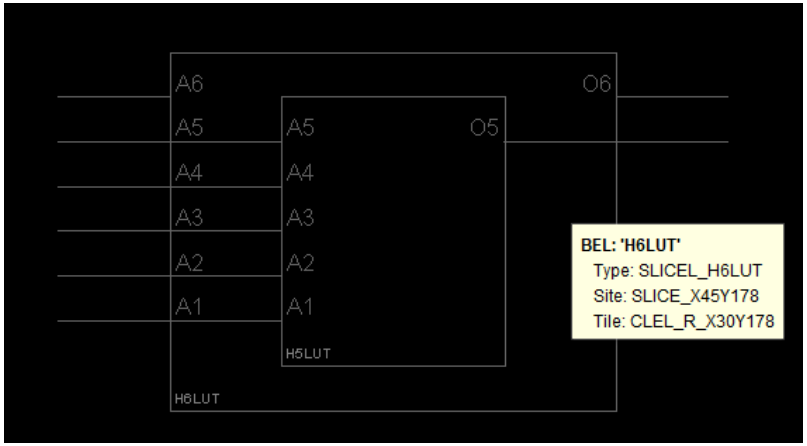




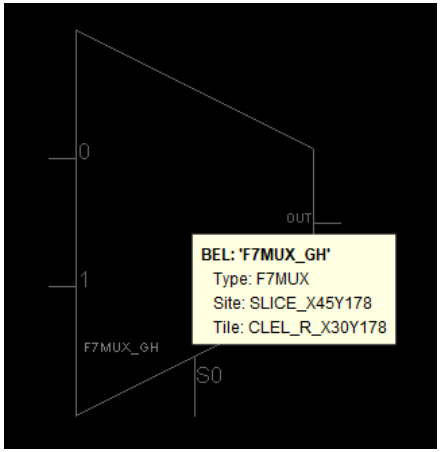
Components of FPGAs

- These components differ depending on FPGA Chips

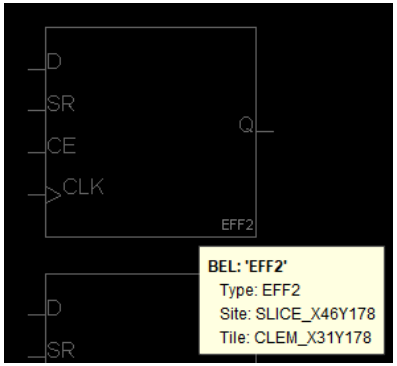
LUT



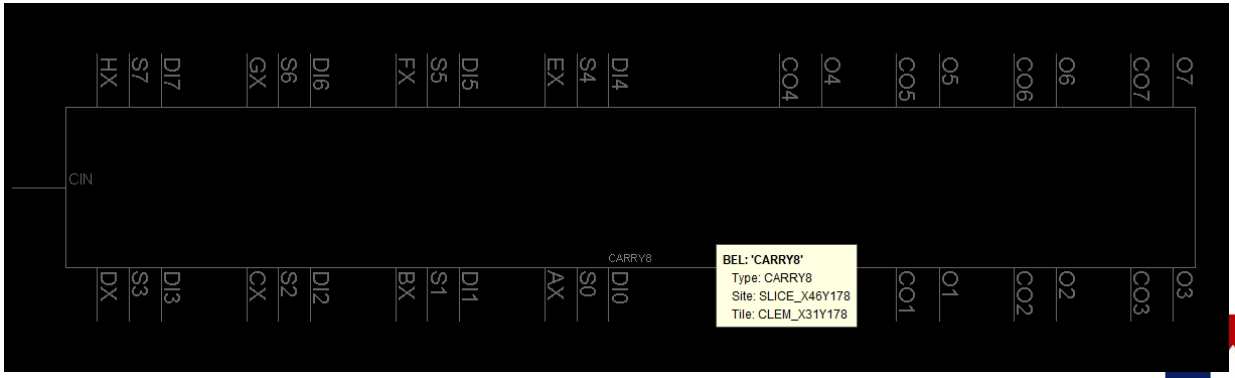
MUX



FF

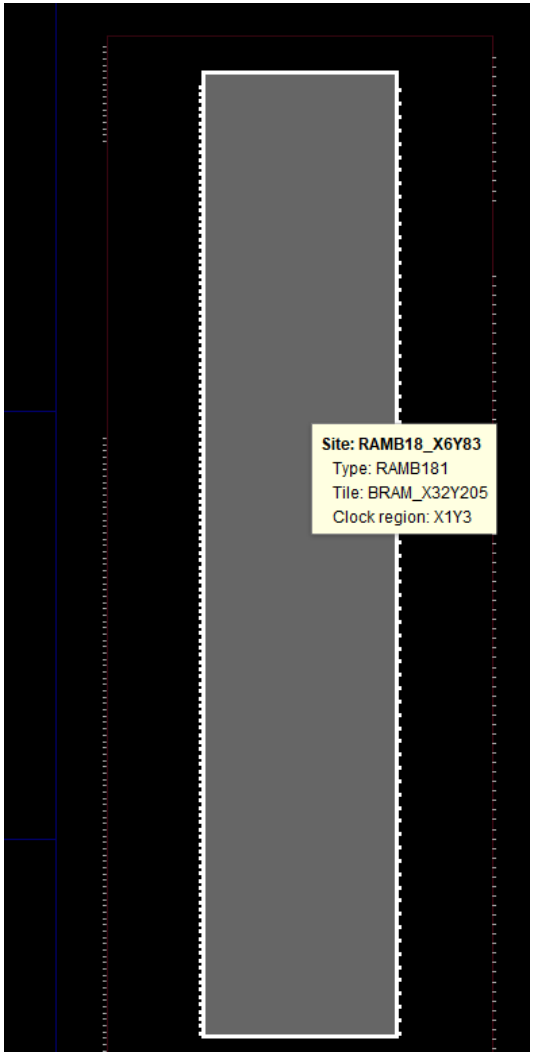


CARRY

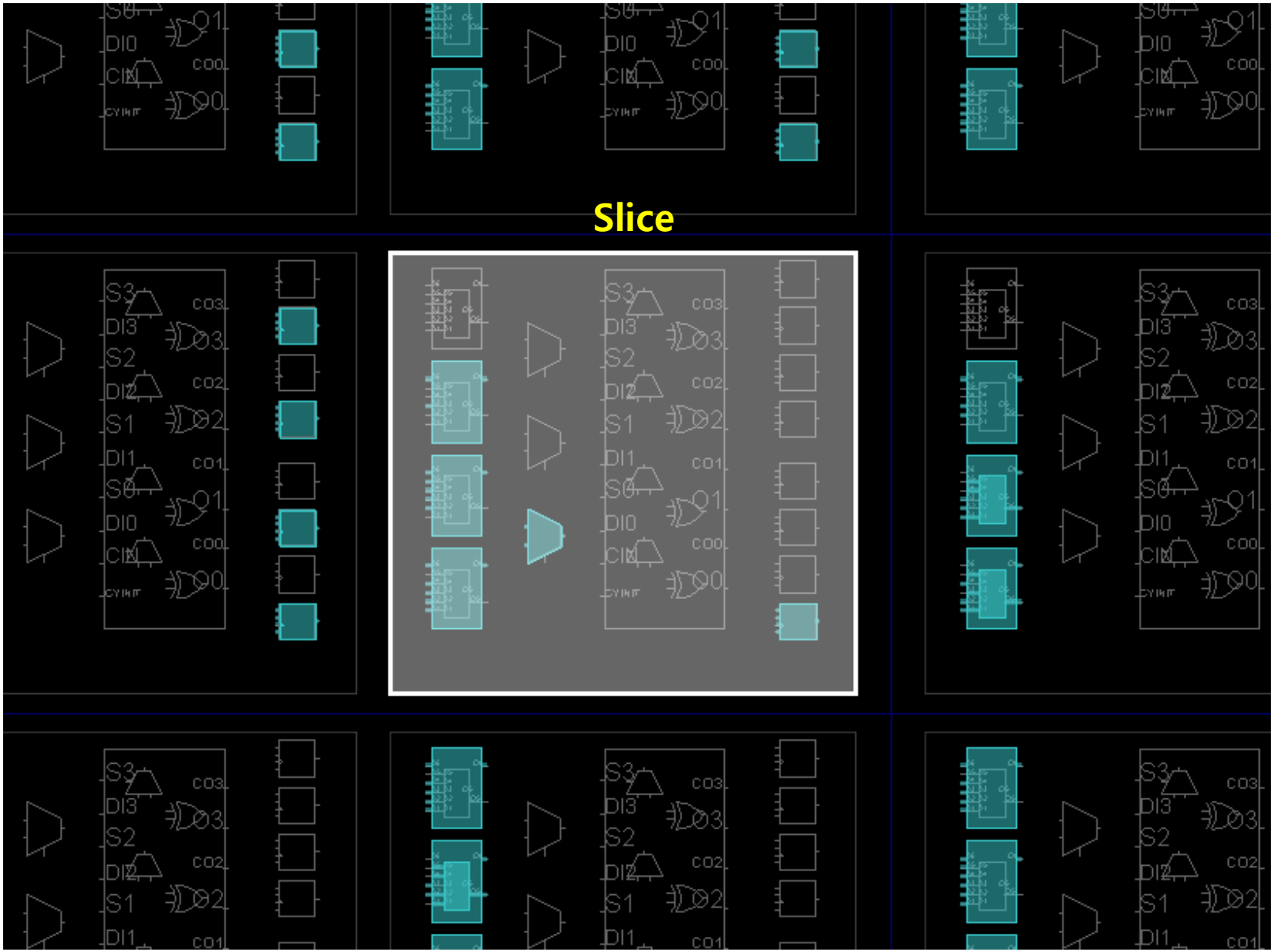


Components of FPGAs

MEMORY & DSP

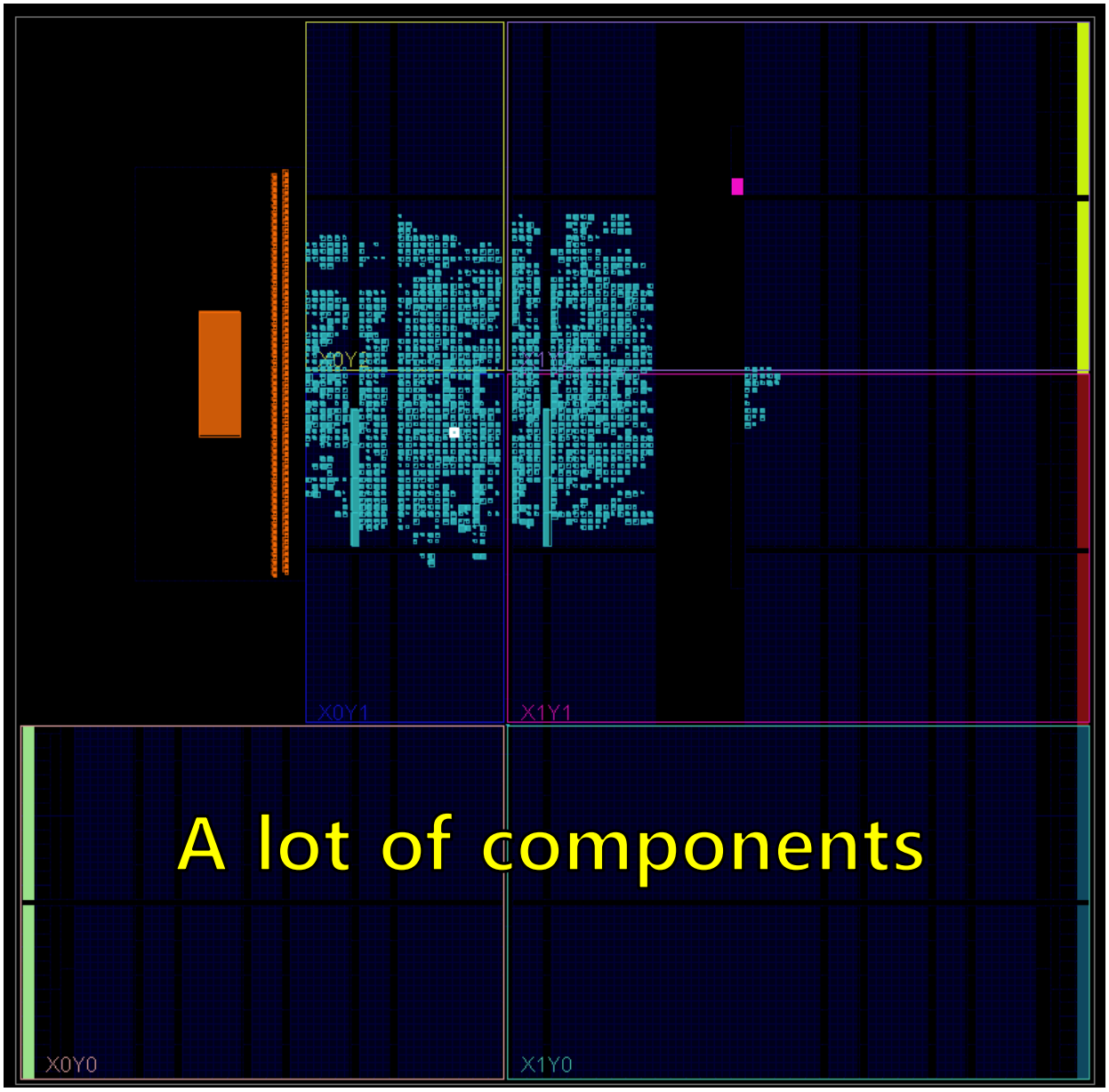


Components of FPGAs



Components of FPGAs

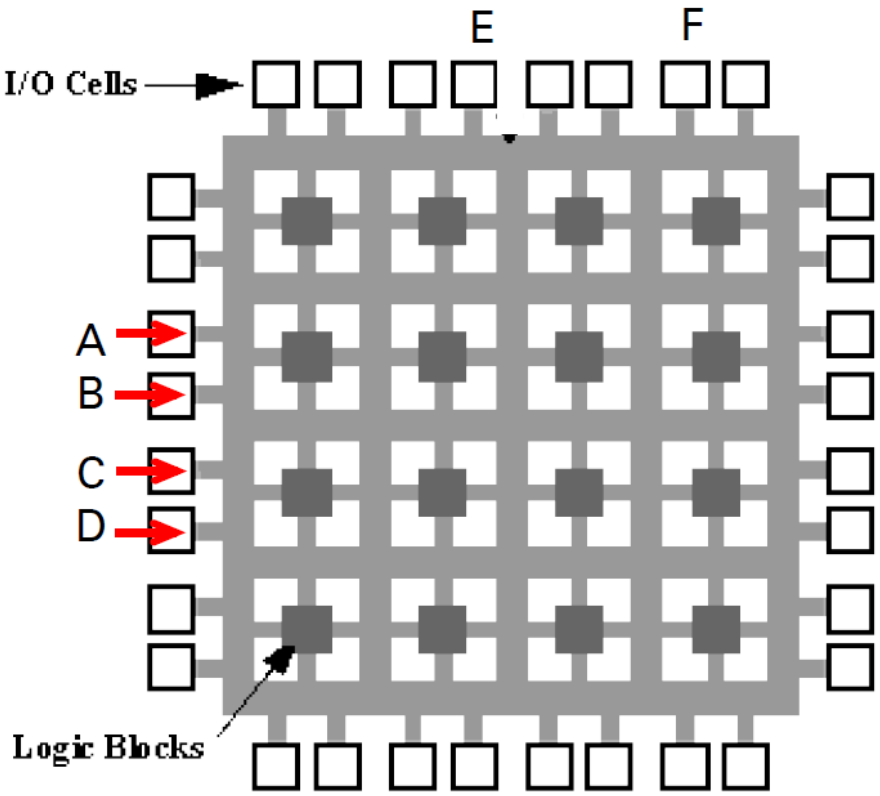
Components of FPGAs



Example of Implementing Logic

Implementing Logic on FPGA:

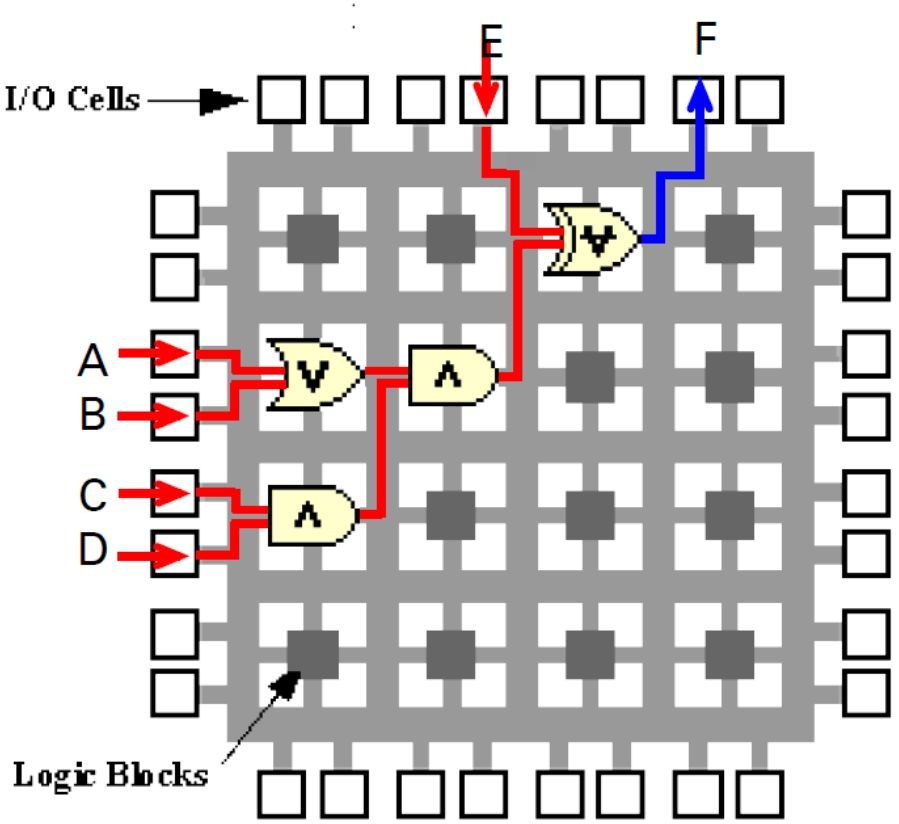
$$F = \{(A+B)CD\} \oplus E$$



Example of Implementing Logic

Implementing Logic on FPGA:

$$F = \{(A+B)CD\} \oplus E$$



Basys 3: A low-end FPGA Board

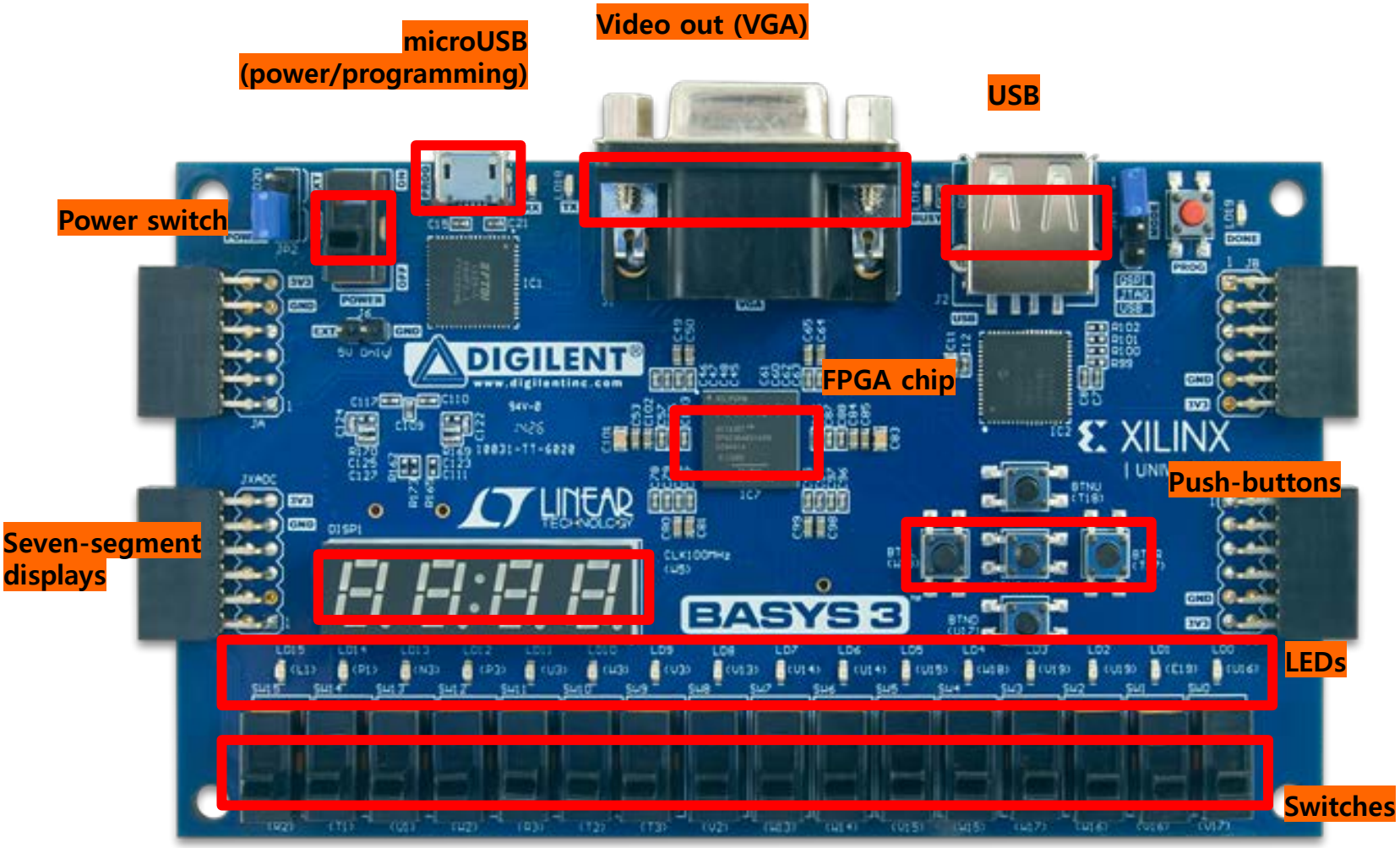
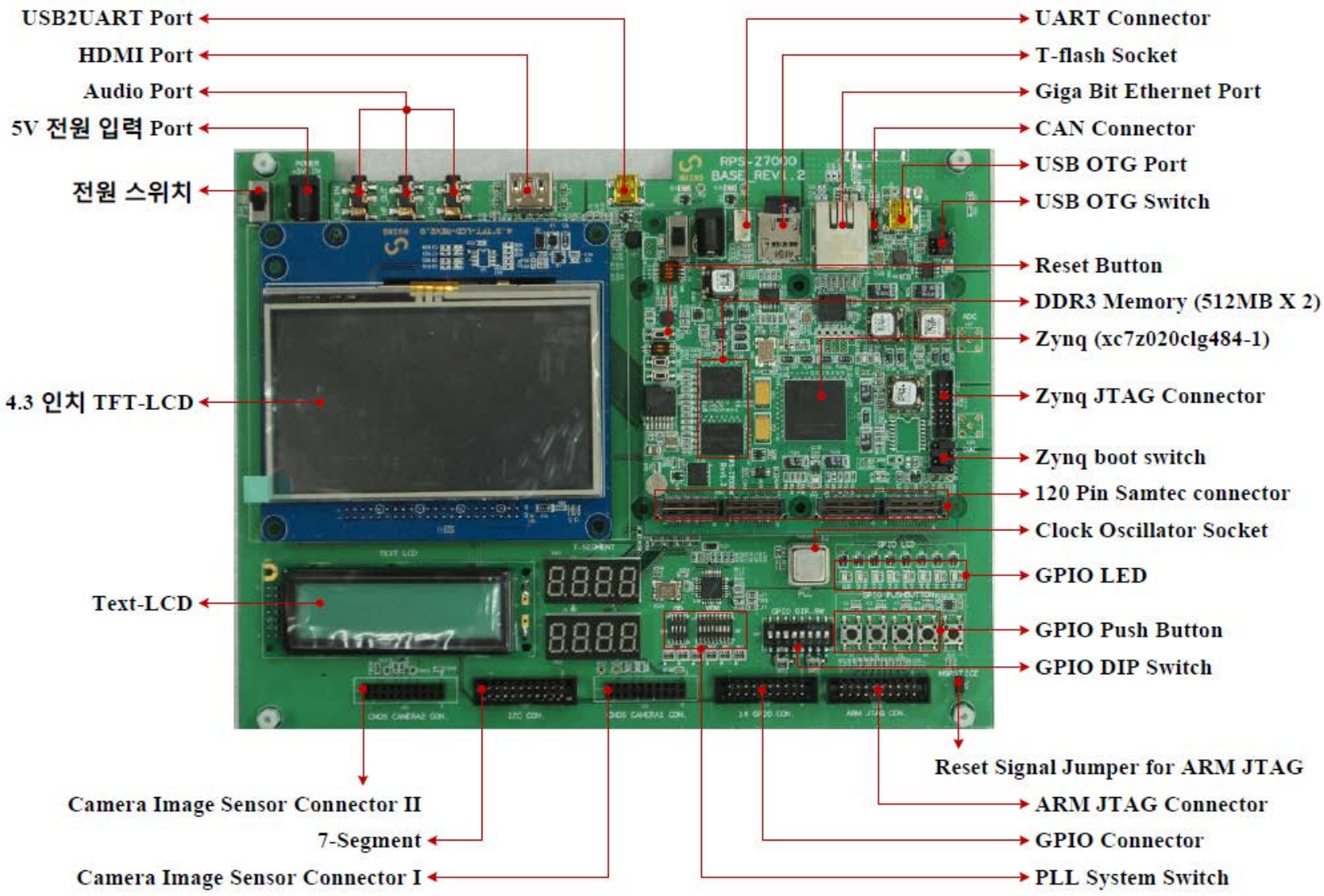


image source: onur-digitaldesign_comparch lecture slide

RPS-ZYNQ7000



What is a ZYNQ?

- Xilinx SoCs are processor-centric platforms that offer software, hardware and I/O programmability in a single chip. The Zynq-7000 family is based on the SoC architecture. Zynq-7000 products incorporate a dual core ARM Cortex-A9 based Processing System (PS) and Xilinx Programmable Logic in a single device.

What is a ZYNQ?

PS

PL

Processing System

Flash Controller NOR, NAND, SRAM, Quad SPI

Multiport DRAM Controller DDR3, DDR3L, DDR2

Processor I/O Mux

2x SPI

2x I2C

2x CAN

2x UART

GPIO

2x SDIO with DMA

2x USB with DMA

2x GigE with DMA

AMBA® Interconnect

AMBA® Interconnect

MPCore

NEON™ SIMD and FPU

NEON™ SIMD and FPU

ARM® Cortex™ - A9

ARM® Cortex™ - A9

Snoop Control Unit

512KB L2 Cache

256KB On-Chip Memory

GIC

JTAG and Trace

Configuration

Timers

DMA

AMBA® Interconnect

AMBA® Interconnect

Security AES, SHA, RSA

EMIO

General-Purpose AXI Ports

ACP

High-Performance AXI Ports

XADC 2x ADC, Mux, Thermal Sensor

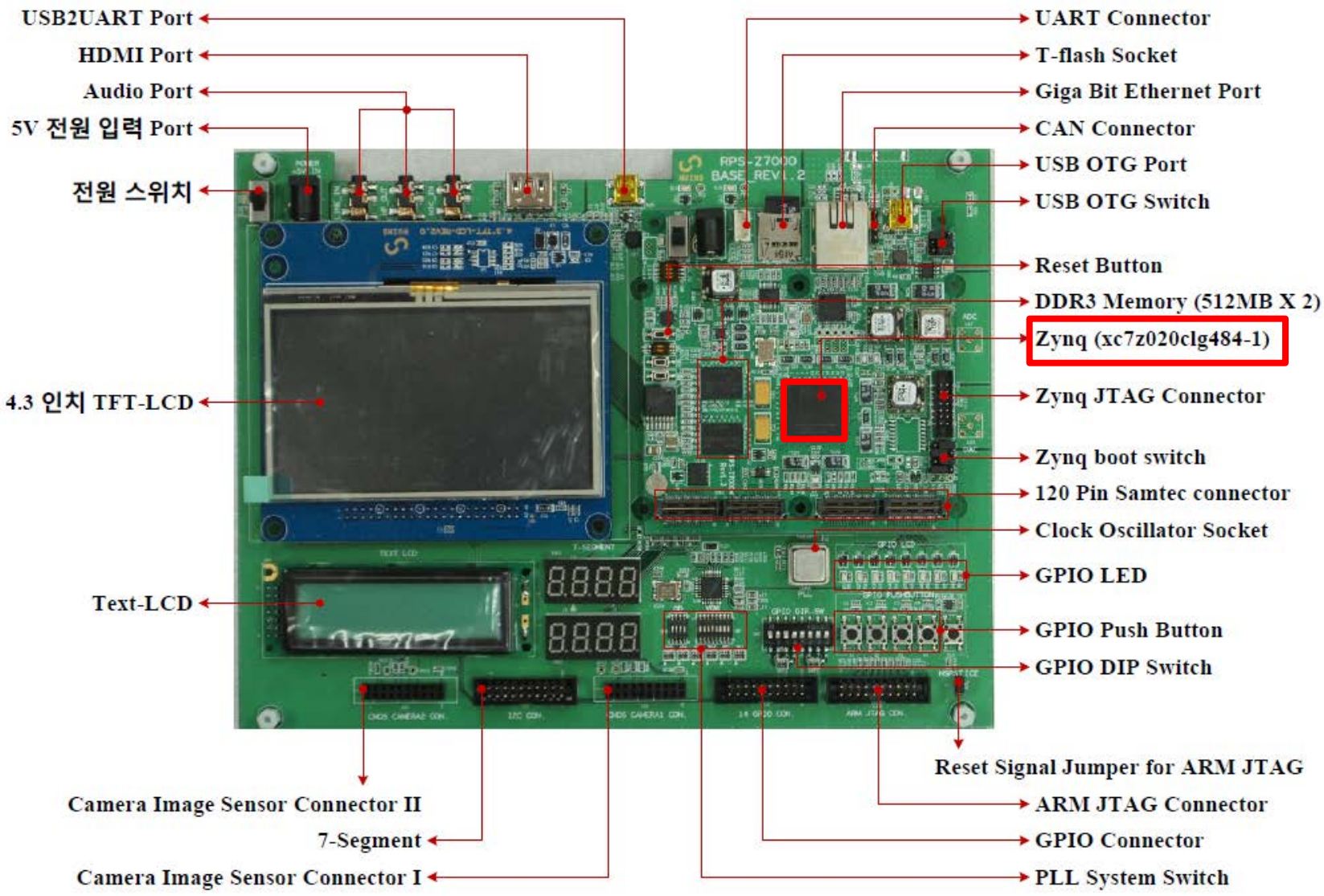
Programmable Logic (System Gates, DSP, RAM)

PCIe® Gen 2 1-8 Lanes

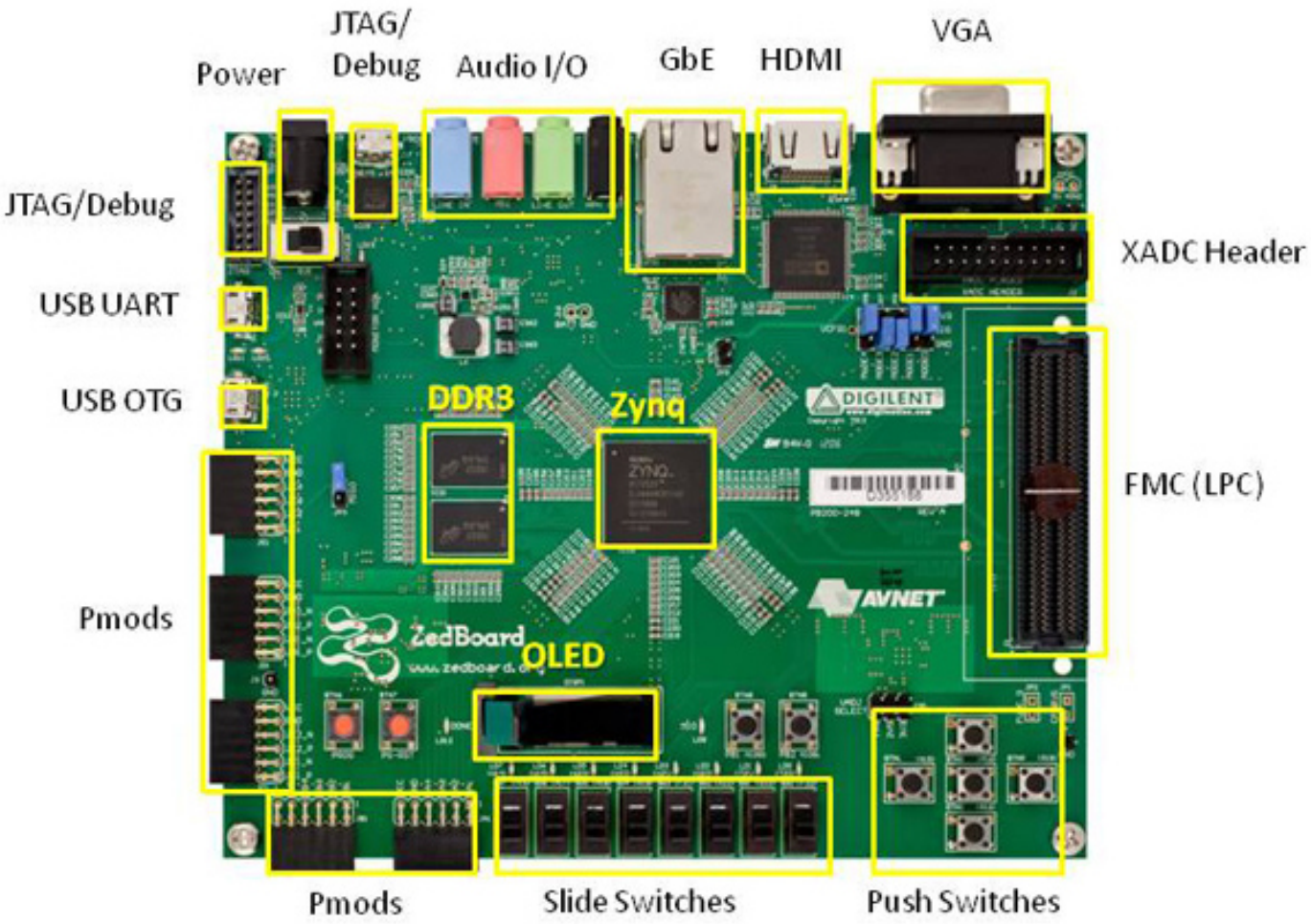
Multi-Standard I/Os (3.3V & High Speed 1.8V)

Serial Transceivers

RPS-ZYNQ7000



ZedBoard: Same FPGA chip that we use



* SD card cage and QSPI Flash reside on backside of board

ZedBoard: Same FPGA chip that we use

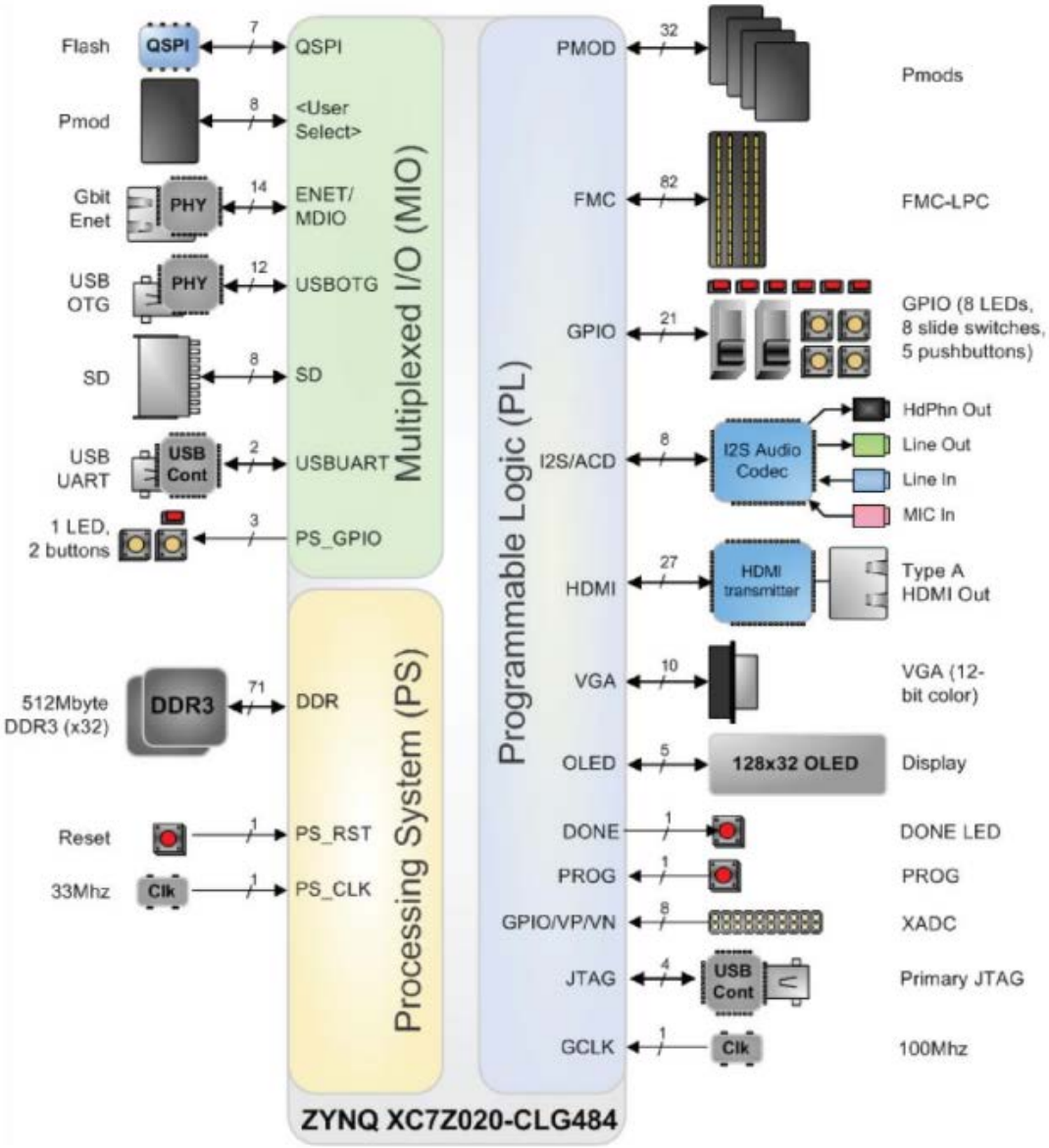


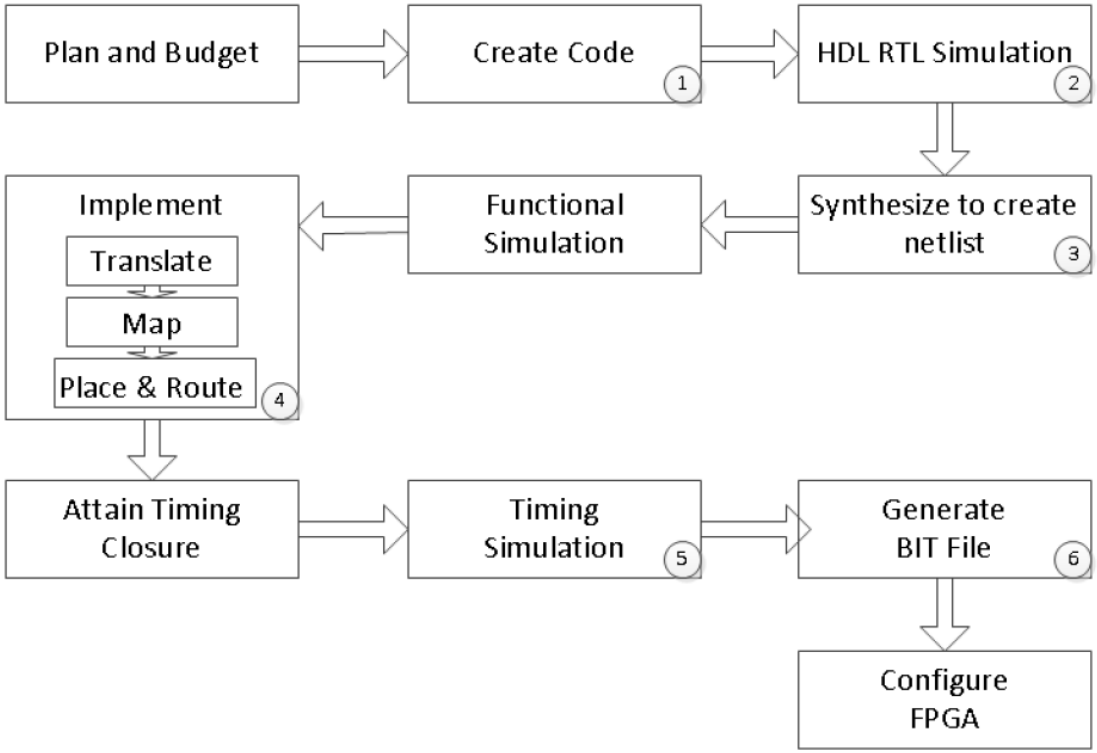
Figure 1 - ZedBoard Hardware Block Diagram

Vivado - Download

- About FPGA
- Design Flow
- Project explanation
- Vivado – Download
- Vivado – Create project
- Vivado – Add/Create sources
- Vivado – Simulation
- Vivado – Example: Simple PL Design
- Project – Reference code review (Video)
- Project – Vivado given sample code review (Video)
- Project – Vivado testbench examples (Video)

Design Flow

- We are going to use Vivado tool in order to model and verify design.
- After verifying design, we are going to program design(bit file) onto FPGAs.



VIVADO HLx Editions

Figure 1. A typical design flow

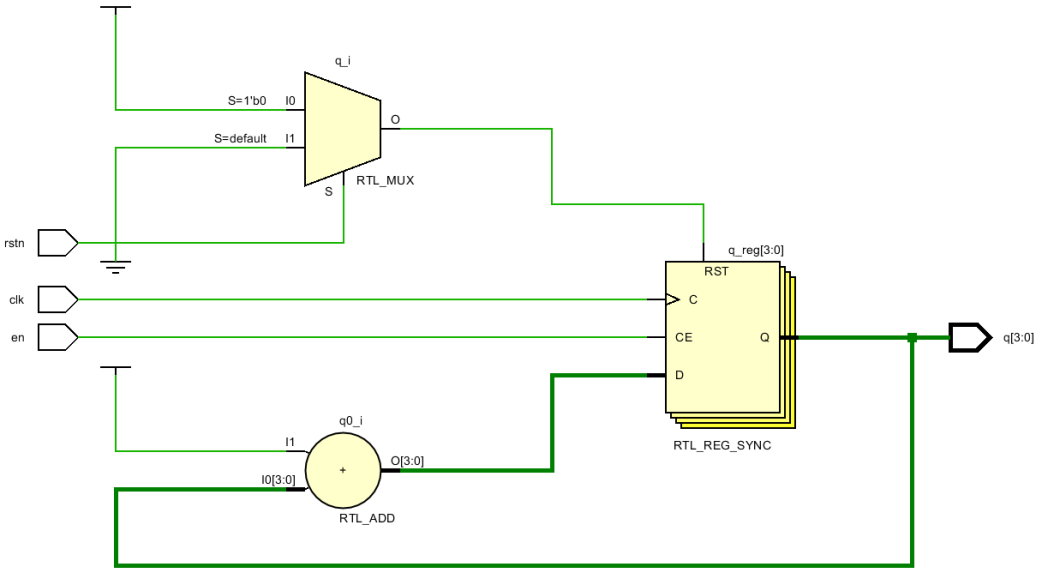
Design Flow

Before Synthesis

HDL Code, RTL Modeling

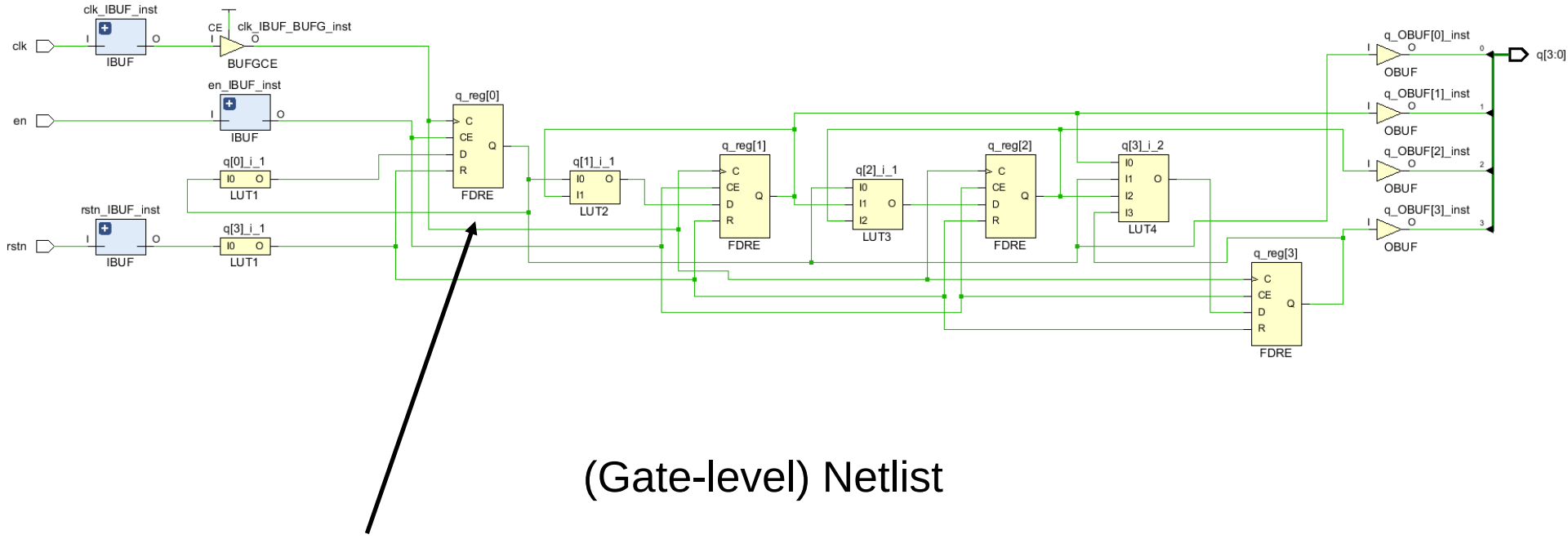
```
1  module test(
2      input wire clk,
3      input wire rstn,
4      input wire en,
5      output reg [3:0] q
6      );
7
8      always @ (posedge clk) begin
9          if (!rstn)
10             q <= 4'h0;
11         else if (en)
12             q <= q + 4'h1;
13     end
14
15 endmodule
```

(RTL Base)Elaborated Design Schematic



Design Flow

- After Synthesis, your RTL model is converted into a netlist, which describes the connections between different block available on the desired FPGA chip. The Synthesized model is a draft of the final design, made with the use of available resources.

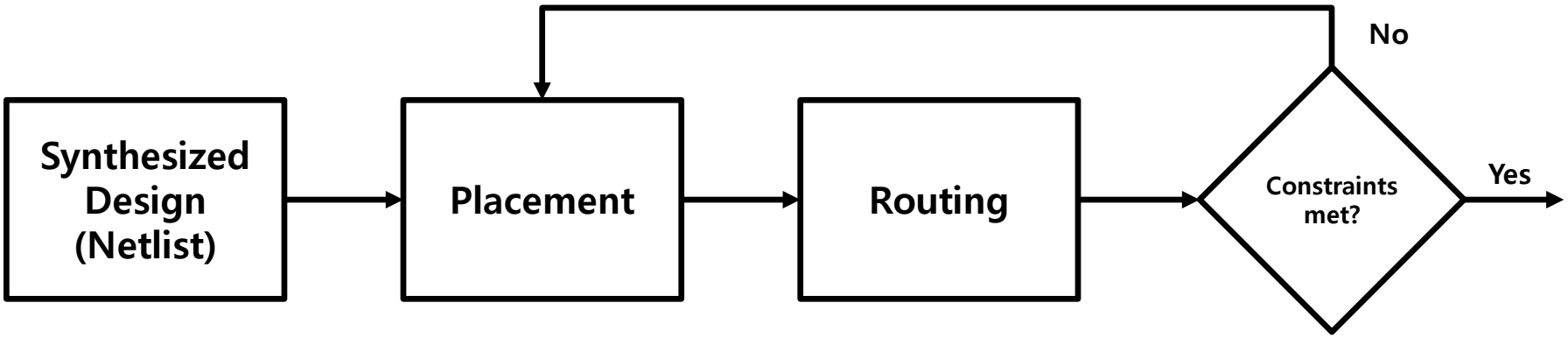


(Gate-level) Netlist

FDRE : Single D-type flip-flop with data (D), clock enable (CE), and synchronous reset (R) inputs and data output (Q)

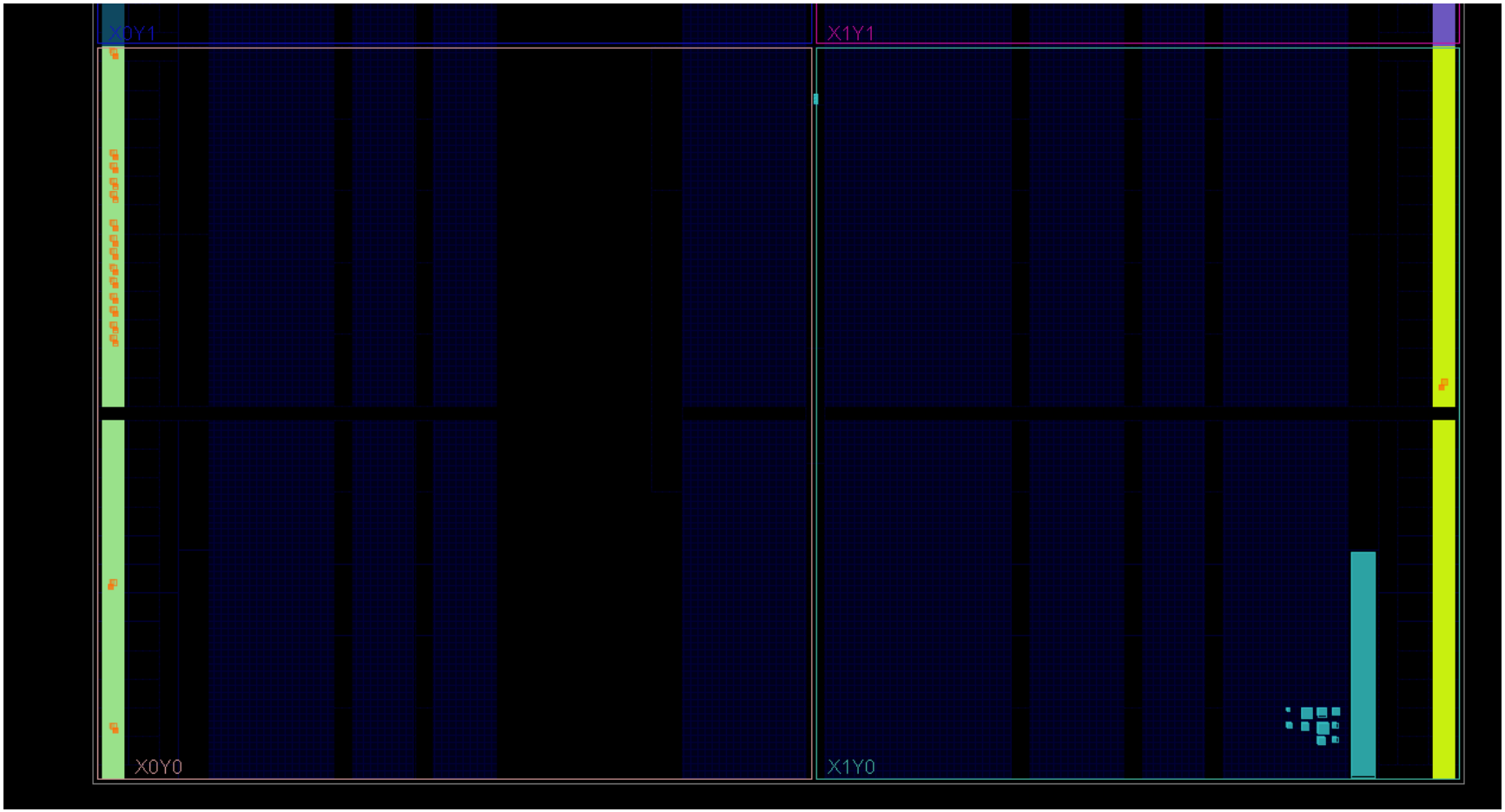
Design Flow

- Implementation(Place and Route) process do implement synthesized design into the target FPGA device. The implementation tool needs to have information about the physical composition of the device, routing paths between the different logical blocks and signal propagation timings.



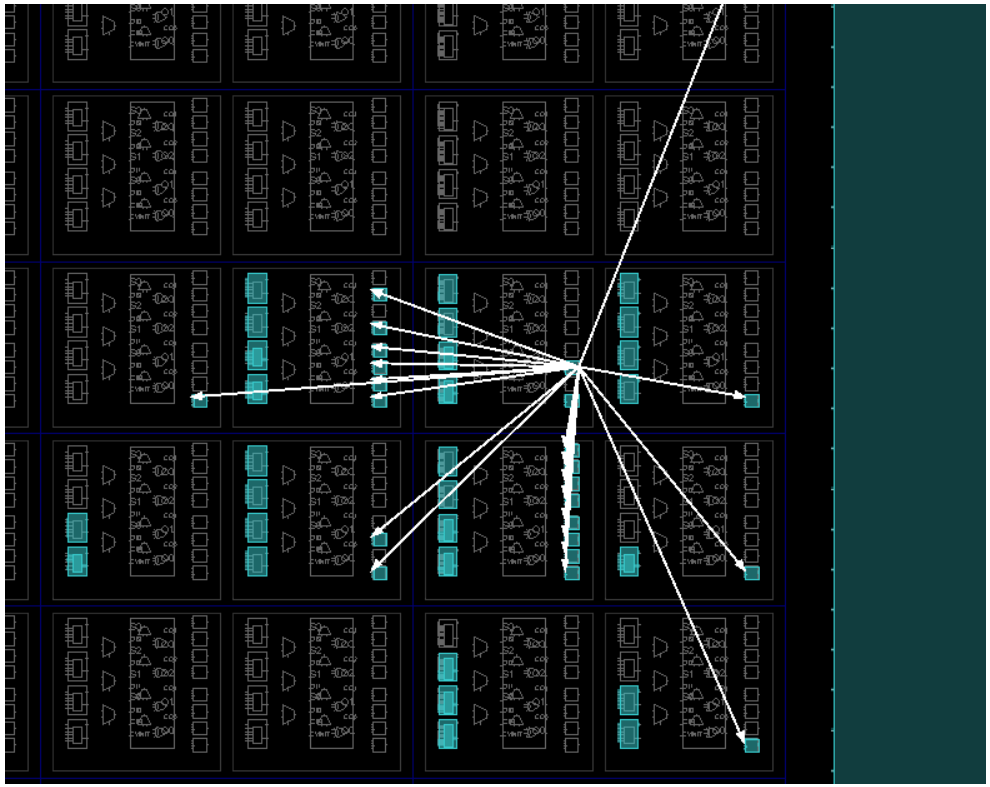
Design Flow

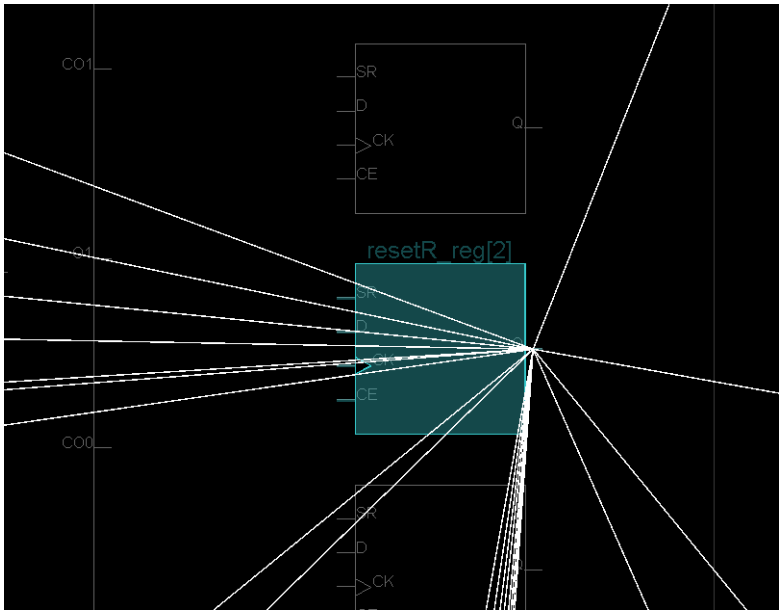
- Activated cells are indicated as follows



Design Flow

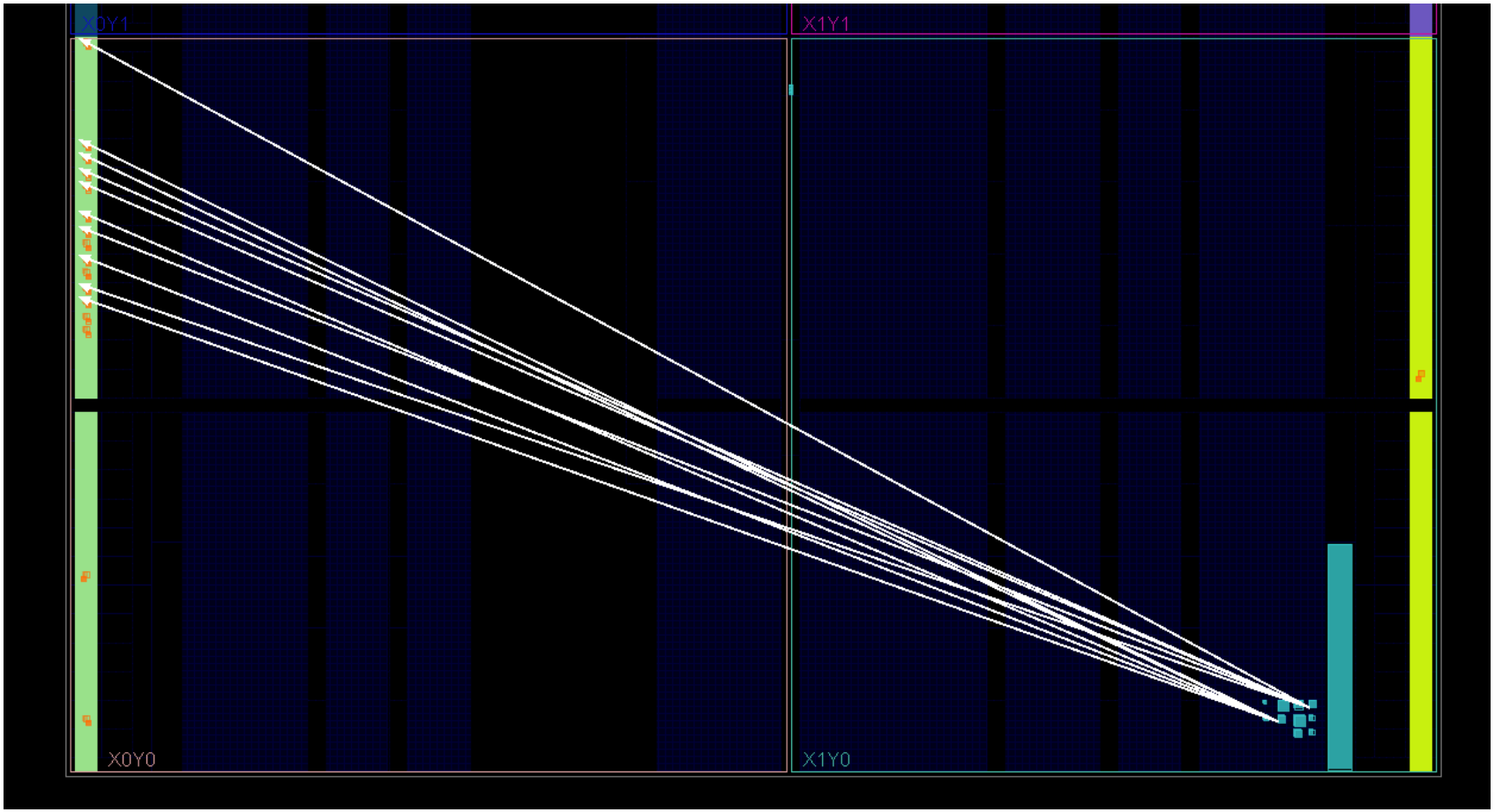
- You can see the connections between cells as follows





Design Flow

- Connections between cells and I/O ports.



Project explanation

- About FPGA
- Design Flow
- Project explanation
- Vivado – Download
- Vivado – Create project
- Vivado – Add/Create sources
- Vivado – Simulation
- Vivado – Example: Simple PL Design
- Project – Reference code review (Video)
- Project – Vivado given sample code review (Video)
- Project – Vivado testbench examples (Video)

Introduction

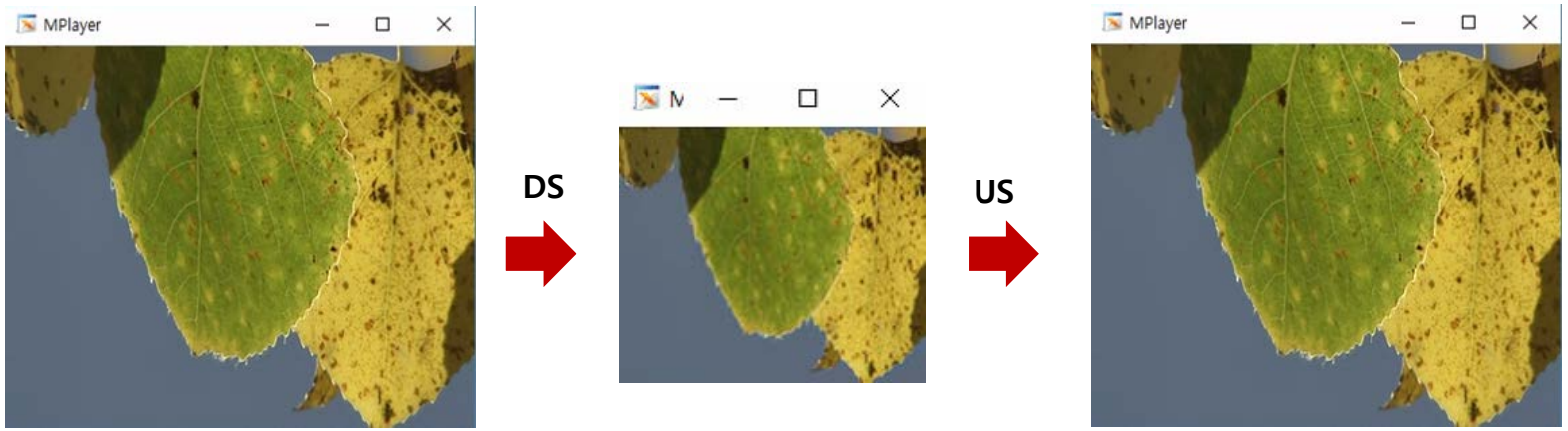
- Subject: ***Image down-sampling (DS) & up-sampling (US)***

- Common Task (Necessary)

  [256x256] -> [128x128] -> [256x256]

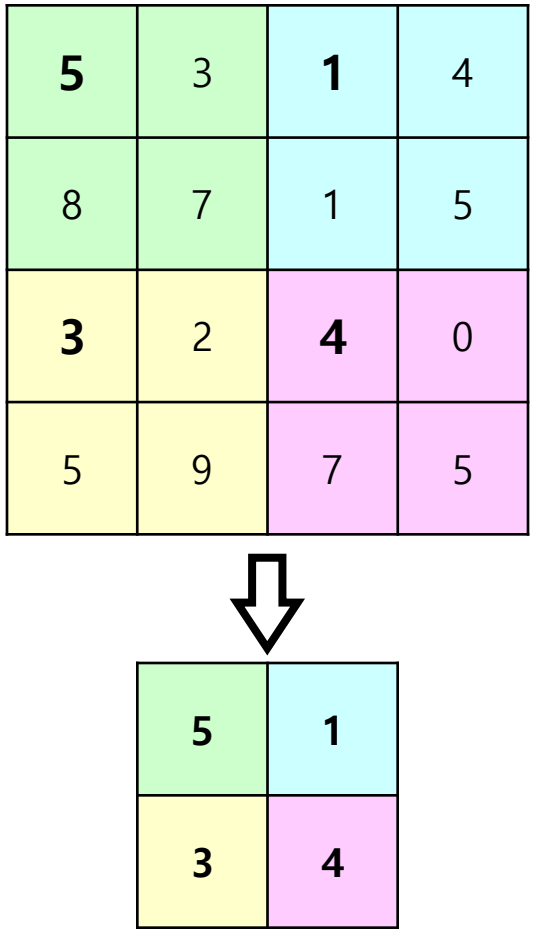
- Extra Task (Optional)

  [256x256] -> [96x96] -> [256x256]

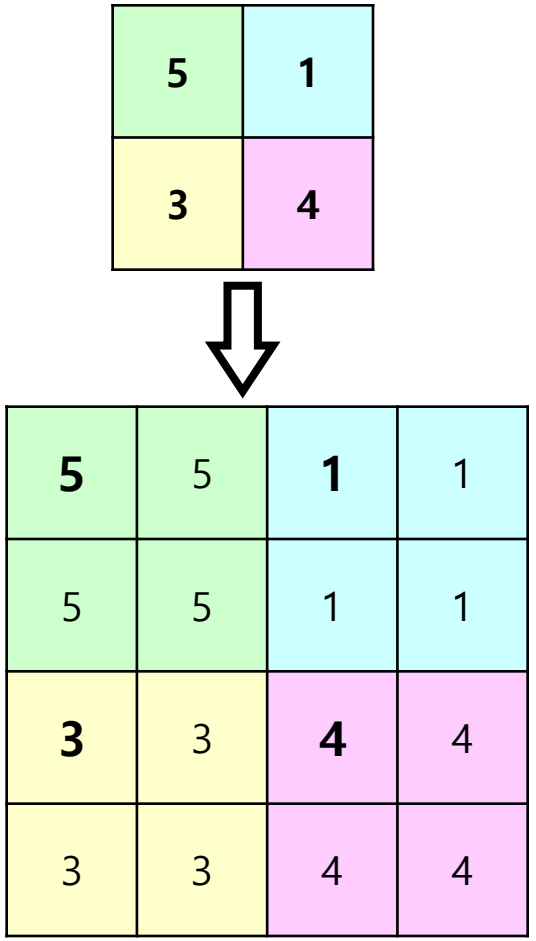


Explanation - DS & US Example

Down Sampling



Up Sampling

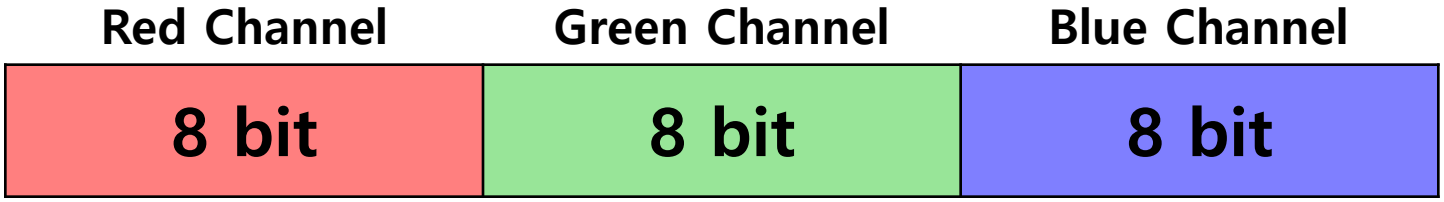


- There are various implementation methods of DS, so please apply your own ideas → Depending on the method, the image quality changes

Explanation – RGB IMG

Explanation – ROM Data Structure

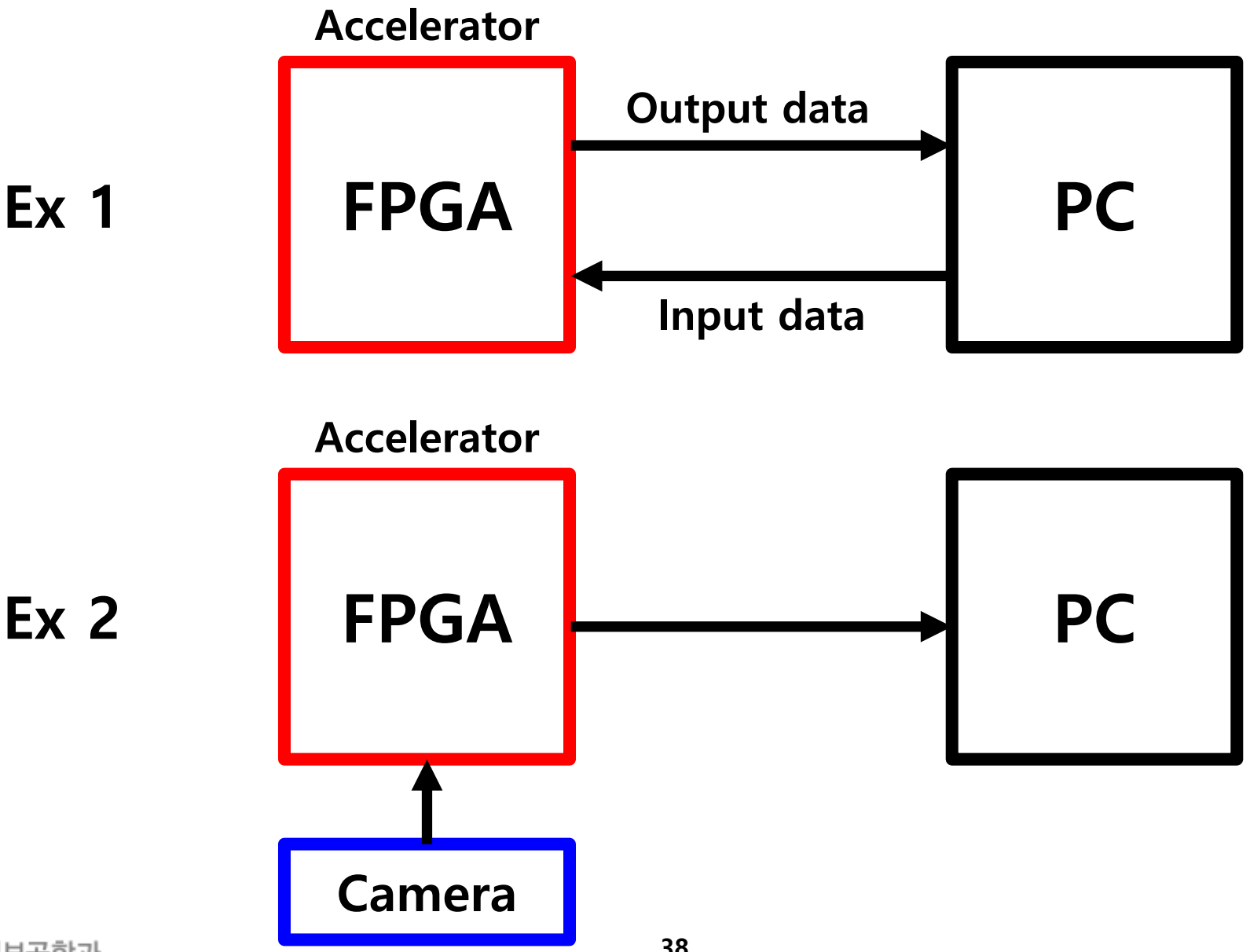
- 24 bits for expressing one pixel



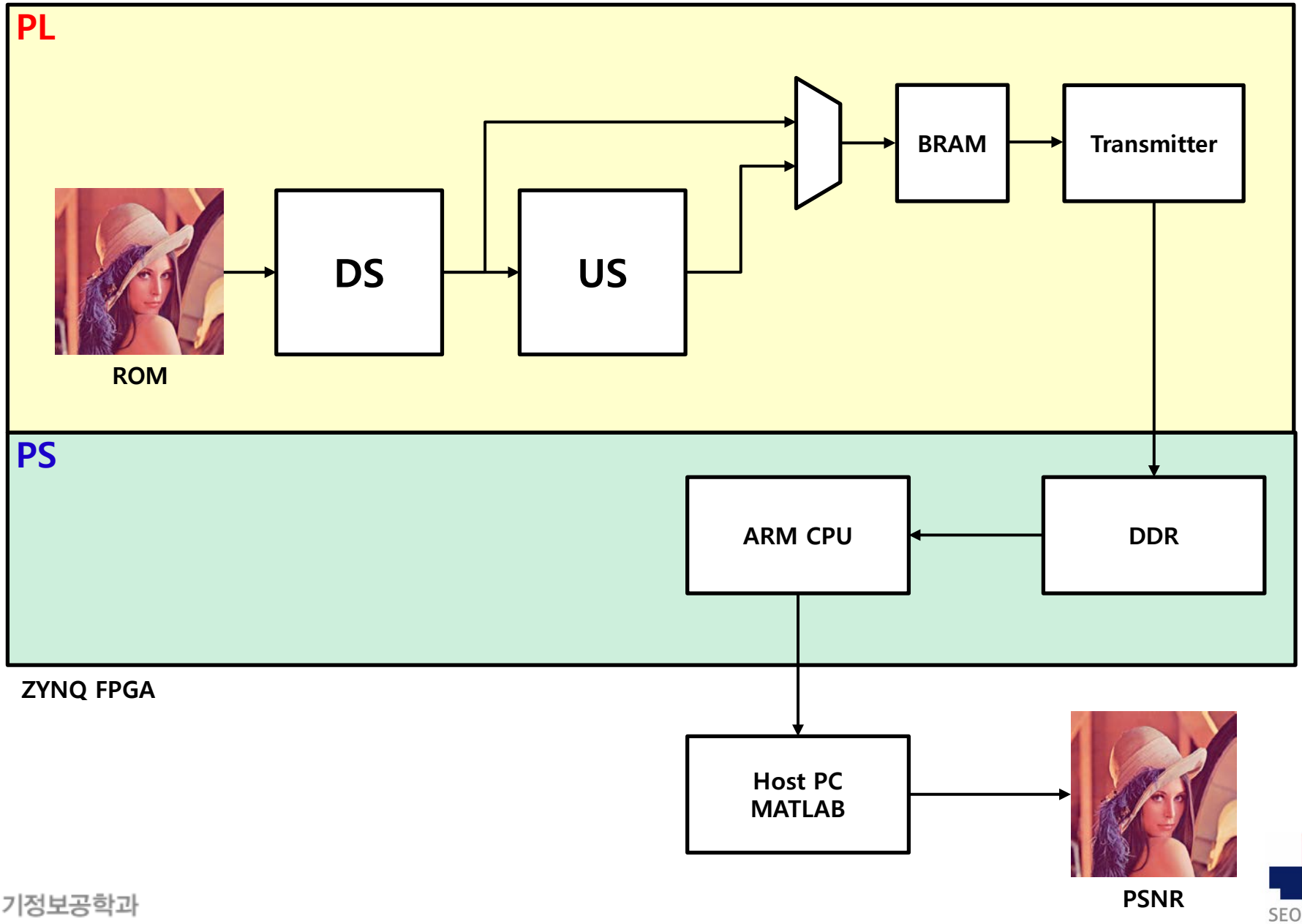
Example - Width : 24bit, Depth : 8

Addr	Data
0	24'b111000101000100101111110
1	24'b110111111000100110000010
2	24'b111000101000011001110111
3	24'b111000111000100001111011
4	24'b111000101000101001111100
5	24'b111000101000011001111001
6	24'b111000001000011101110011
7	24'b110111101000010101110101

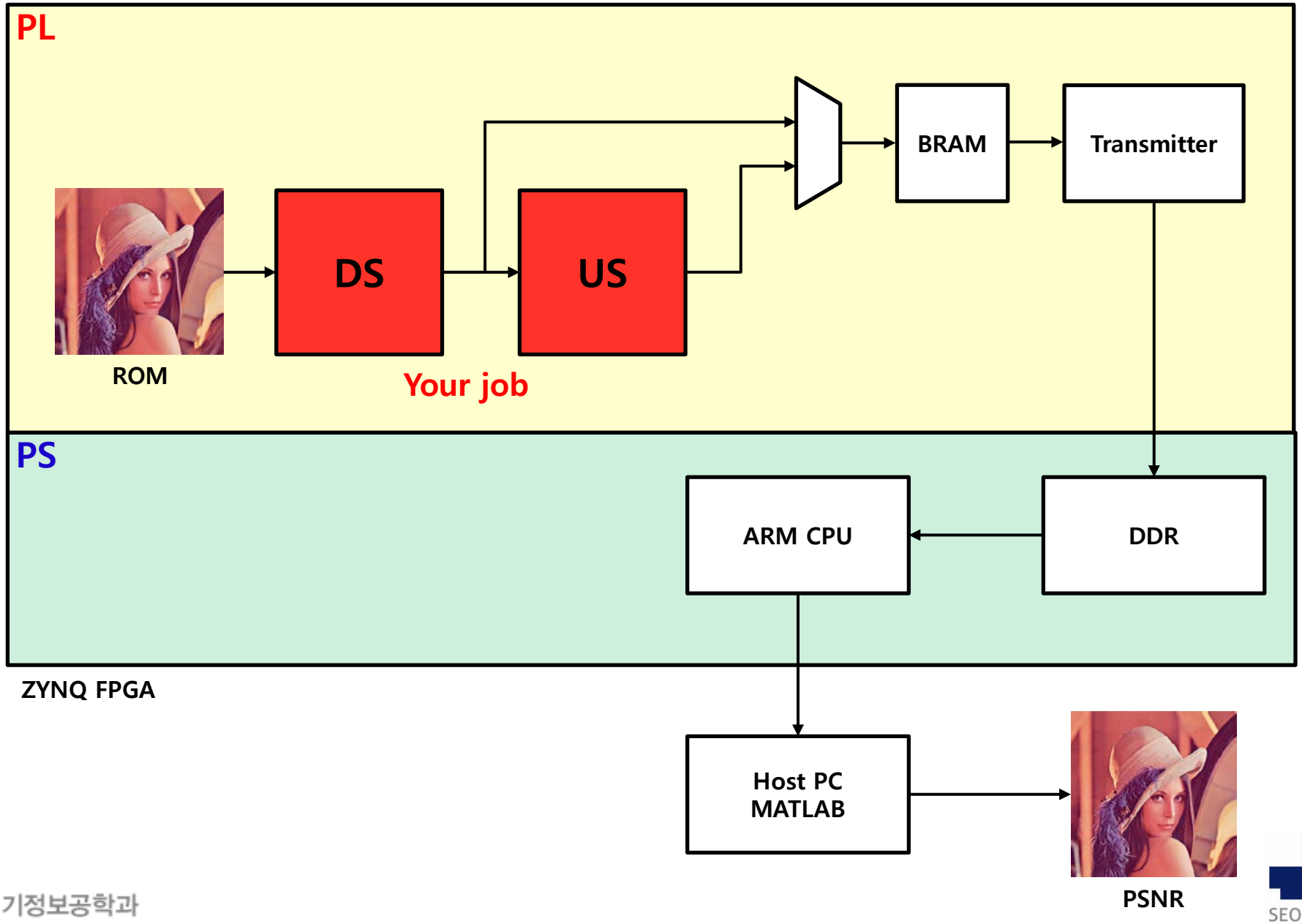
Explanation – Use of FPGA as An Accelerator



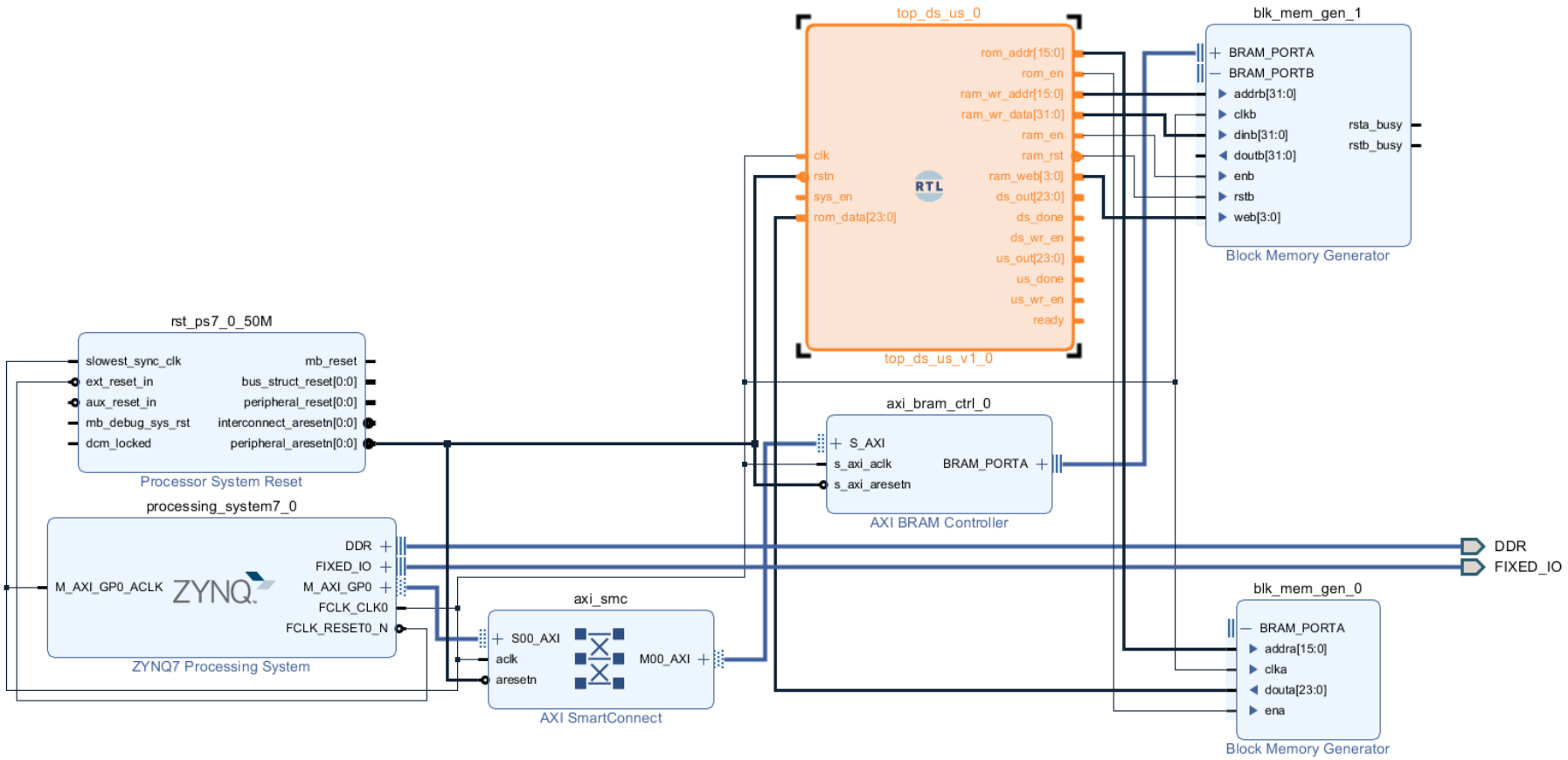
Configuration



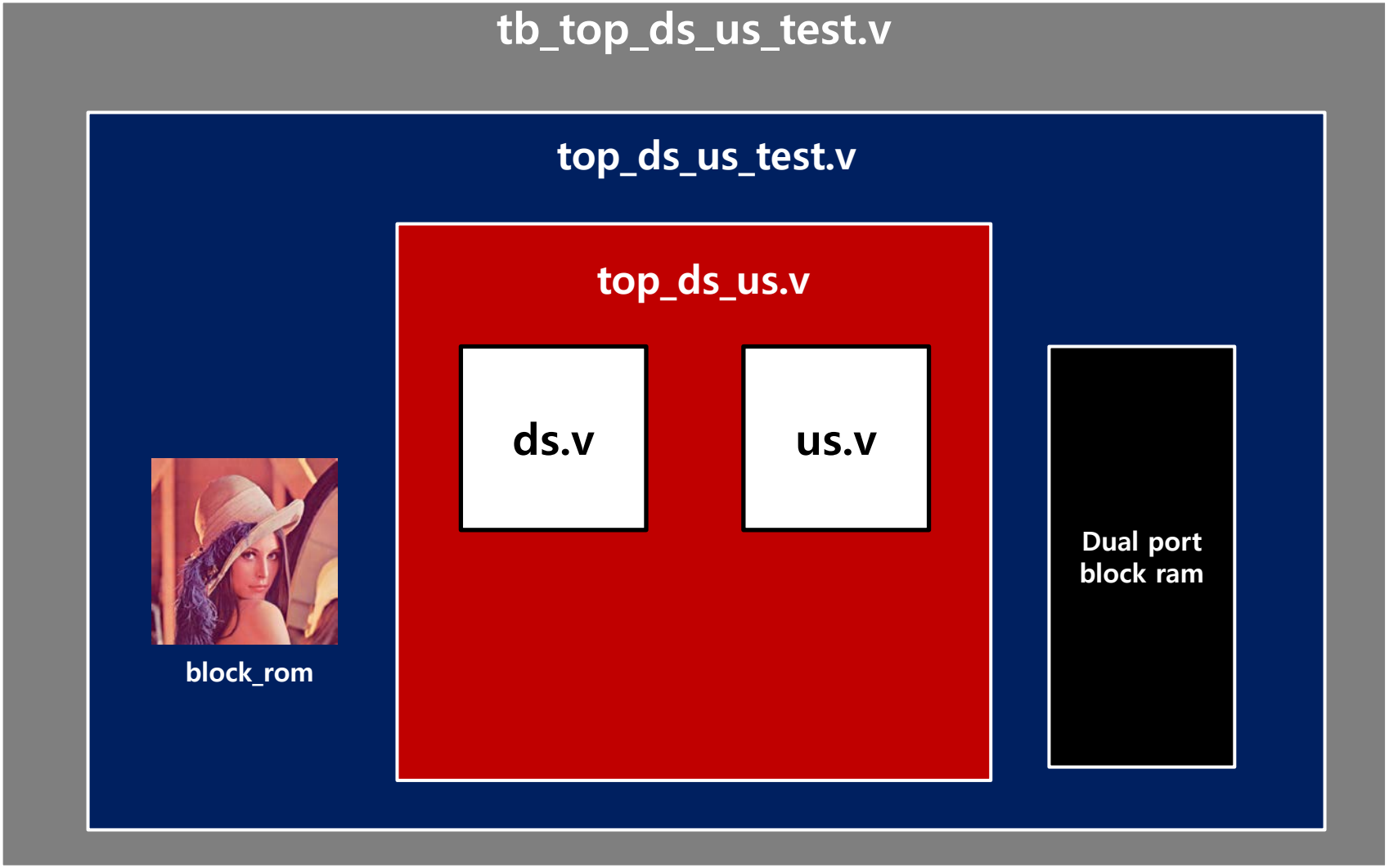
Configuration



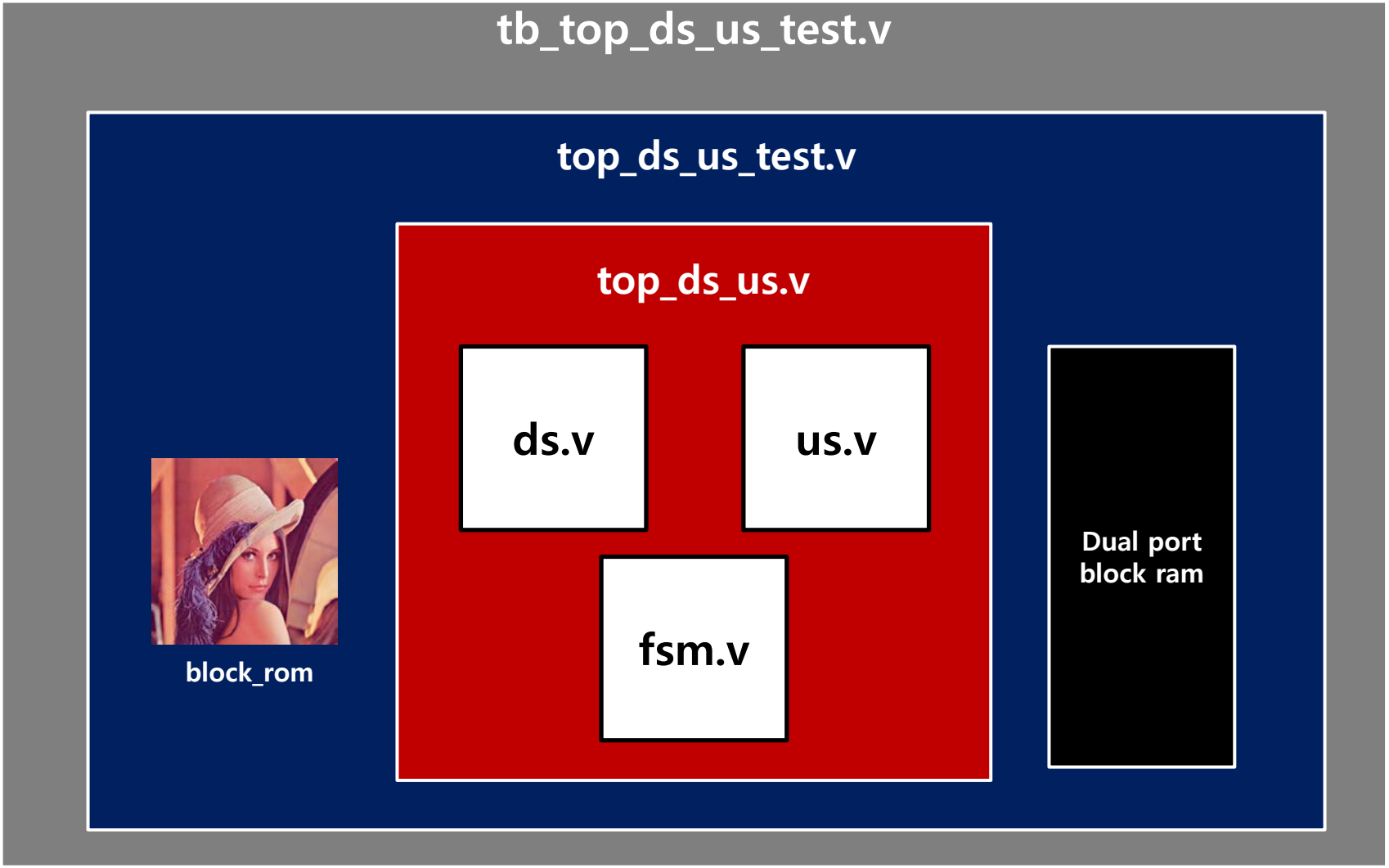
Configuration – Block Design



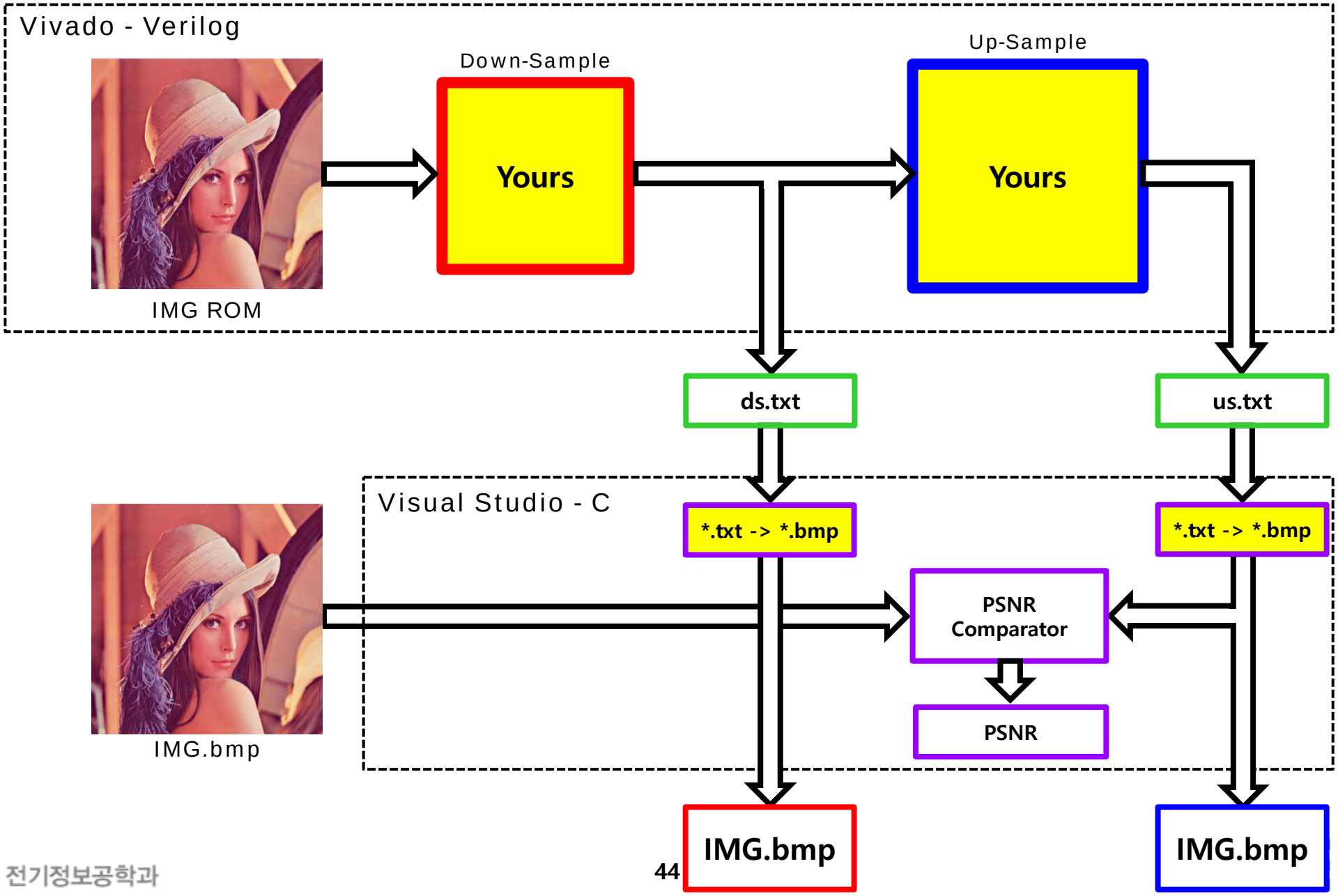
Design module hierarchy



Design module hierarchy



Simulation method example



Required Knowledge – bmp format (1)

I recommend you to try to see all the value yourself

Offset	Contents
0~53	Format Data
54~	Pixel Data

offset	**size**	**description**
0	2	signature, must be 4D42 hex
2	4	size of BMP file in bytes (unreliable)
6	2	reserved, must be zero
8	2	reserved, must be zero
10	4	offset to start of image data in bytes
14	4	size of BITMAPINFOHEADER structure, mus t be 40
18	4	image width in pixels
22	4	image height in pixels
26	2	number of planes in the image, must be 1
28	2	number of bits per pixel (1, 4, 8, or 24)
30	4	compression type (0=none, 1=RLE-8, 2=RL E-4)
34	4	size of image data in bytes (including padd ing)
38	4	horizontal resolution in pixels per meter (u nreliable)
42	4	vertical resolution in pixels per meter (unrel iable)
46	4	number of colors in image, or zero
50	4	number of important colors, or zero

Required Knowledge – bmp format (2)

I recommend you to try to see all the value yourself.

When you check this value, it returns 54, which means pixel data starts from 54th byte address

If the size of image is 128 by 64 (128x64), then 18th byte data contain the value of 128 and 22th byte data contain the value of 64

If you load a given image, 28th byte data would contain 24

offset	**size**	**description**
0	2	signature, must be 4D42 hex
2	4	size of BMP file in bytes (unreliable)
6	2	reserved, must be zero
8	2	reserved, must be zero
10	4	offset to start of image data in bytes
14	4	size of BITMAPINFOHEADER structure, must be 40
18	4	image width in pixels
22	4	image height in pixels
26	2	number of planes in the image, must be 1
28	2	number of bits per pixel (1, 4, 8, or 24)
30	4	compression type (0=none, 1=RLE-8, 2=RLE-4)
34	4	size of image data in bytes (including padding)
38	4	horizontal resolution in pixels per meter (unreliable)
42	4	vertical resolution in pixels per meter (unreliable)
46	4	number of colors in image, or zero
50	4	number of important colors, or zero

Required Knowledge – bmp format

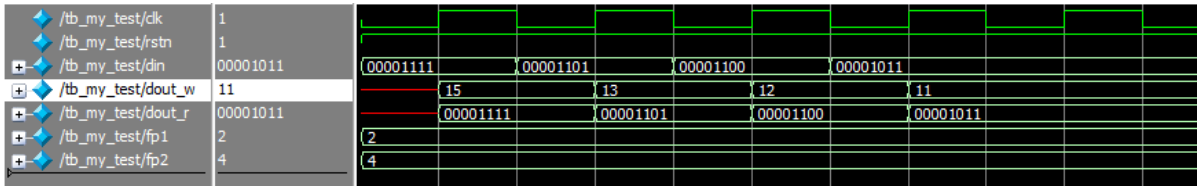
Example Code

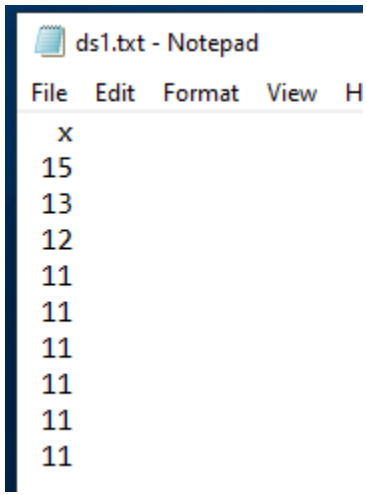
```
5     FILE *fp_read;
6     fopen_s(&fp_read, "D:\orinigal.bmp", "rb");
7
8     FILE *fp_write;
9     fopen_s(&fp_read, "D:\downsampled.bmp", "wb");
10
11    unsigned char* data = (unsigned char*)malloc(sizeof(unsigned char) * 256 * 256 * 3 + 54);
12    fread(data, sizeof(unsigned char), 256 * 256 * 3 + 54, fp1);
13
14    /*
15        Your logic, Pixel data : data[54] ~ data[256*256*3+54-1]
16    */
17
18    fclose(fp_read);
19    fclose(fp_write);
20    free(data);
```

※ This is just an example code and therefore, you don't have to follow this style!

File Write in Verilog

```
// file pointer
integer fp;

// file open
initial begin
    fp = $fopen("ds1.txt");
end

// data write
always @ ( posedge clk ) begin
    $fdisplay(fp, "%d", dout_w);
end

// file close
initial begin
    #1000    $finish ;
             $fclose(fp);
end
```





This txt file is stored in your project location

You can add some conditions to write proper data

Required Knowledge – PSNR (1)

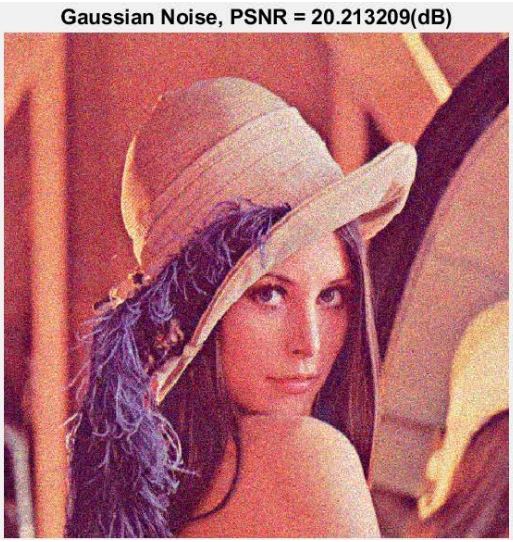
- Peak Signal to Noise Ratio (PSNR) is the standard way to measure fidelity

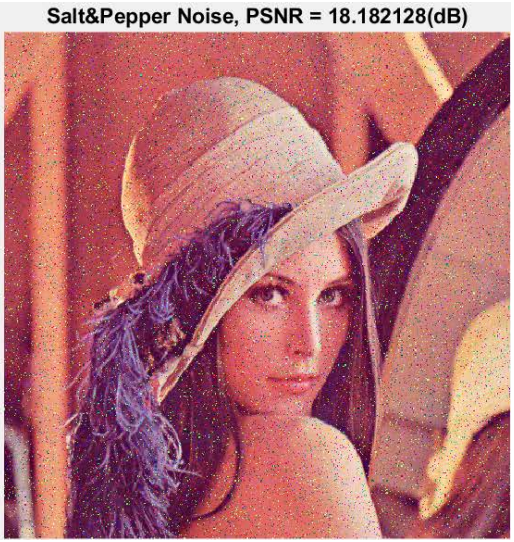
$$PSNR = 10 \log_{10} \left(\frac{m^2}{MSE} \right)$$

where m is the maximum value of a pixel possible.
For gray scale images (8 bits per pixel) m = 255

- PSNR is measured in decibel (dB)
- A high PSNR means less image quality damage (low MSE)

Required Knowledge – PSNR (2)





-As the difference between two images as greater, PSNR gets low value!
-.5 to 1 dB is said to be a perceptible difference
-In general, PSNRs of at least 30 dB are required to be considered acceptable

* We will provide a C code for calculating PSNRs

Required Knowledge – Cycle(Throughput)

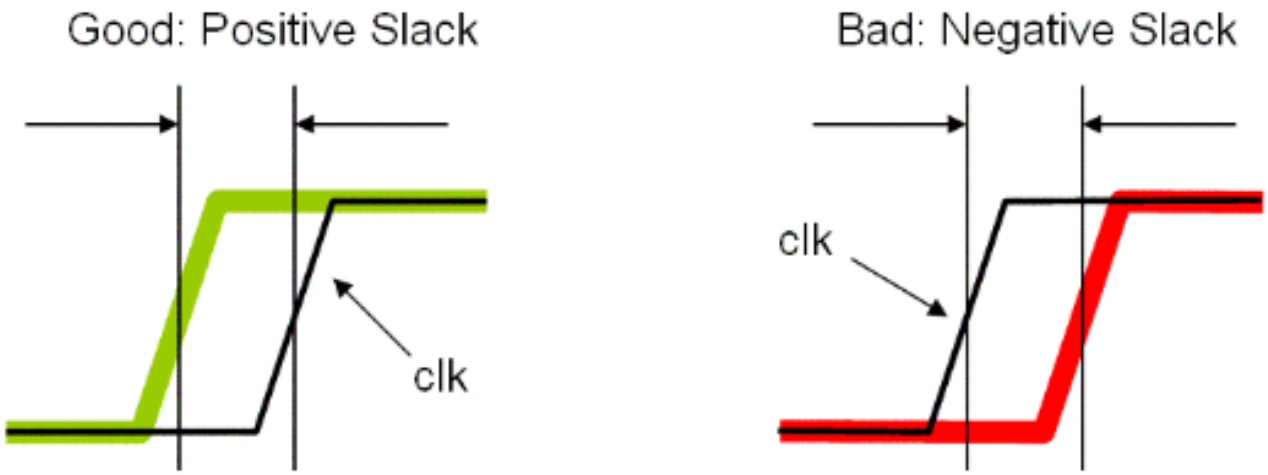
- In HW design, cycle is one of the important factors. If there are enough resource, it's better to exploit resource as many as possible in order to increase throughput (which means shorter cycle, less number of clocks per task).
- I'll give you a skeleton code "top.v, tb_top.v". There is an example circuit how to measure your total clock.

```
always @ (posedge clk or negedge rstn) begin
    if ( !rstn )
        total_clock <= 0;
    else begin
        if ( en_system )
            if ( tlast ) begin
                $display("Total Clock = %d", total_clock);
                total_clock <= 0;
            end
            else
                total_clock <= total_clock + 1;
        else
            total_clock <= 0;
    end
end
```

Required Knowledge – Timing

- After synthesis, you can see “estimated timing information”, which includes a few things about “slack”.
- We don’t have enough time to explain detail about slack, but simply speaking, if there is any negative slack in your logic, then your design would fail.

Positive slack : There is some spare time, so it’s good.
Negative slack : There need some time to meet design constraints, so it’s bad.



source from <http://babyworm.net/archives/170>

Required Knowledge – Timing

- Check your <Design Timing Summary> report, after synthesis. Following values should be greater than or equal to 0.
- This example tell us there might be some paths which can cause problems.

Design Timing Summary

should be positive

Setup		Hold	
Worst Negative Slack (WNS):	7.189 ns	Worst Hold Slack (WHS):	-0.107 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	-483.437 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	9887
Total Number of Endpoints:	20437	Total Number of Endpoints:	20437

Required Knowledge – Timing

- Post Synthesis

Design Timing Summary

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	7.189 ns	Worst Hold Slack (WHS):	-0.107 ns	Worst Pulse Width Slack (WPWS):	3.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	-483.437 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	9887	Number of Failing Endpoints:	0
Total Number of Endpoints:	20437	Total Number of Endpoints:	20437	Total Number of Endpoints:	6604

- Post Implementation

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	4.094 ns	Worst Hold Slack (WHS):	0.010 ns	Worst Pulse Width Slack (WPWS):	3.500 ns
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	0.000 ns
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	0
Total Number of Endpoints:	17209	Total Number of Endpoints:	17209	Total Number of Endpoints:	5708

- However, post synthesis report is not exact. It is just an estimated value. In order to obtain almost exact value, you have to perform "Implementation" and see post implementation timing report.
- In this case, even though there were some problems according to post-synthesis report, it turned out that your design would work well.

Required Knowledge – Timing

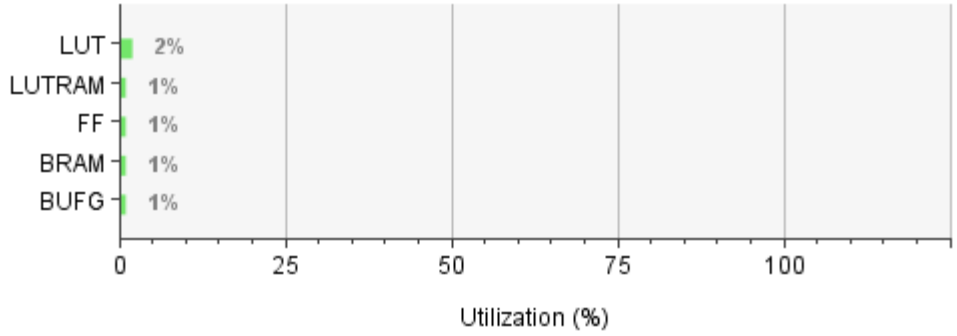
- Since our design is quite simple and there is no constraints, your timing result would be highly likely to be seen as follows.

Setup		Hold		Pulse Width	
Worst Negative Slack (WNS):	inf	Worst Hold Slack (WHS):	inf	Worst Pulse Width Slack (WPWS):	NA
Total Negative Slack (TNS):	0.000 ns	Total Hold Slack (THS):	0.000 ns	Total Pulse Width Negative Slack (TPWS):	NA
Number of Failing Endpoints:	0	Number of Failing Endpoints:	0	Number of Failing Endpoints:	NA
Total Number of Endpoints:	723	Total Number of Endpoints:	723	Total Number of Endpoints:	NA

Required Knowledge – HW Utilization

Summary

Resource	Utilization	Available	Utilization %
LUT	4375	274080	1.60
LUTRAM	1045	144000	0.73
FF	6501	548160	1.19
BRAM	2	912	0.22
BUFG	1	404	0.25



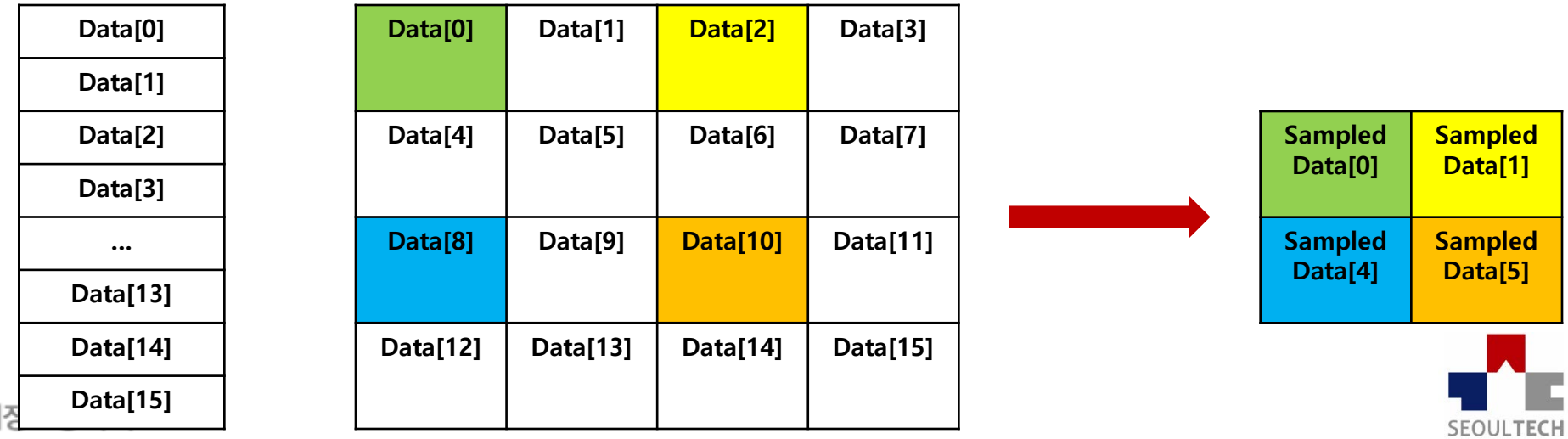
After synthesis of the designed logic, you can see how many resources are used in your design by checking utilization report

I'll address detail of it another slide.

Practice 4: Down-sample

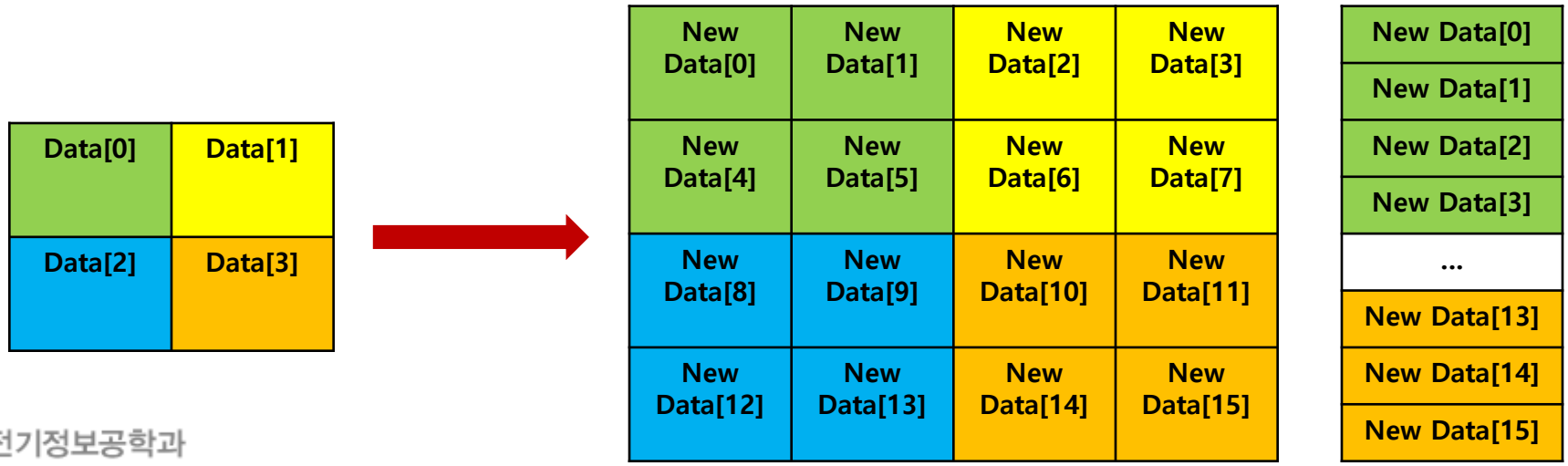
Simple Down-sampler

- Function: select one pixel data and discard the rest pixel data.
- Main circuits
  - Address generator
  - Data flow controller
- Main signals
  - system - enable
  - memory(read) - enable, address, data
  - effective data – valid, output data



Practice 4: Up-sample

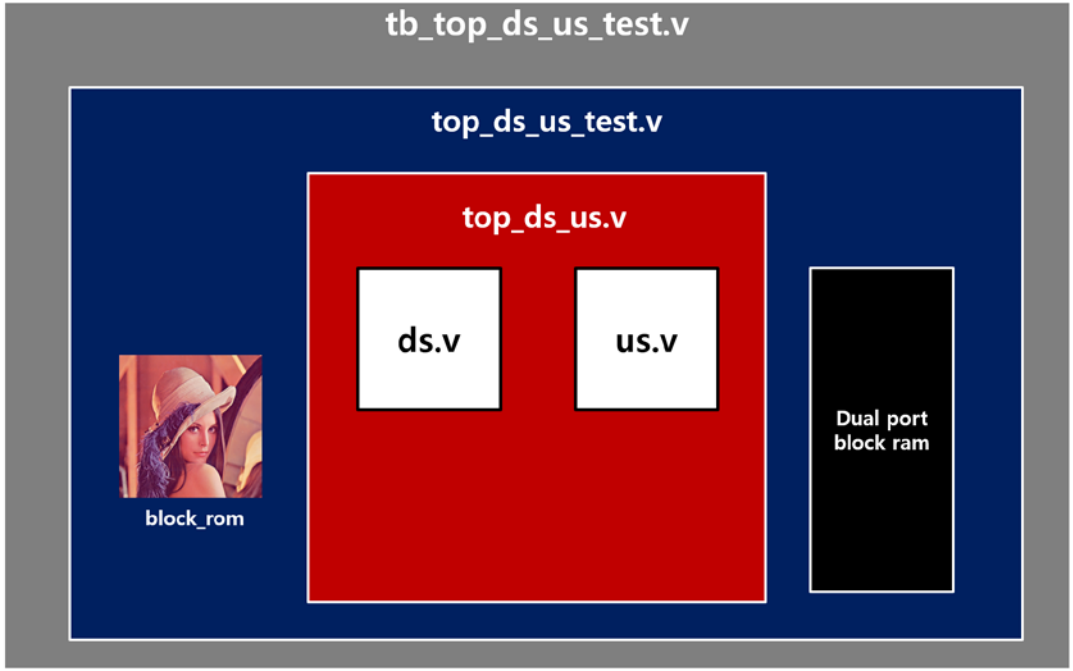
- Up-sample
  - Function: Copy and generate same pixel data
  - Main circuits
    - Address generator
    - Reproduction logic
    - Data flow controller
  - Main signals
    - system – enable
    - memory(read) – enable, address, data
    - memory(write) – write enable, enable, address, data



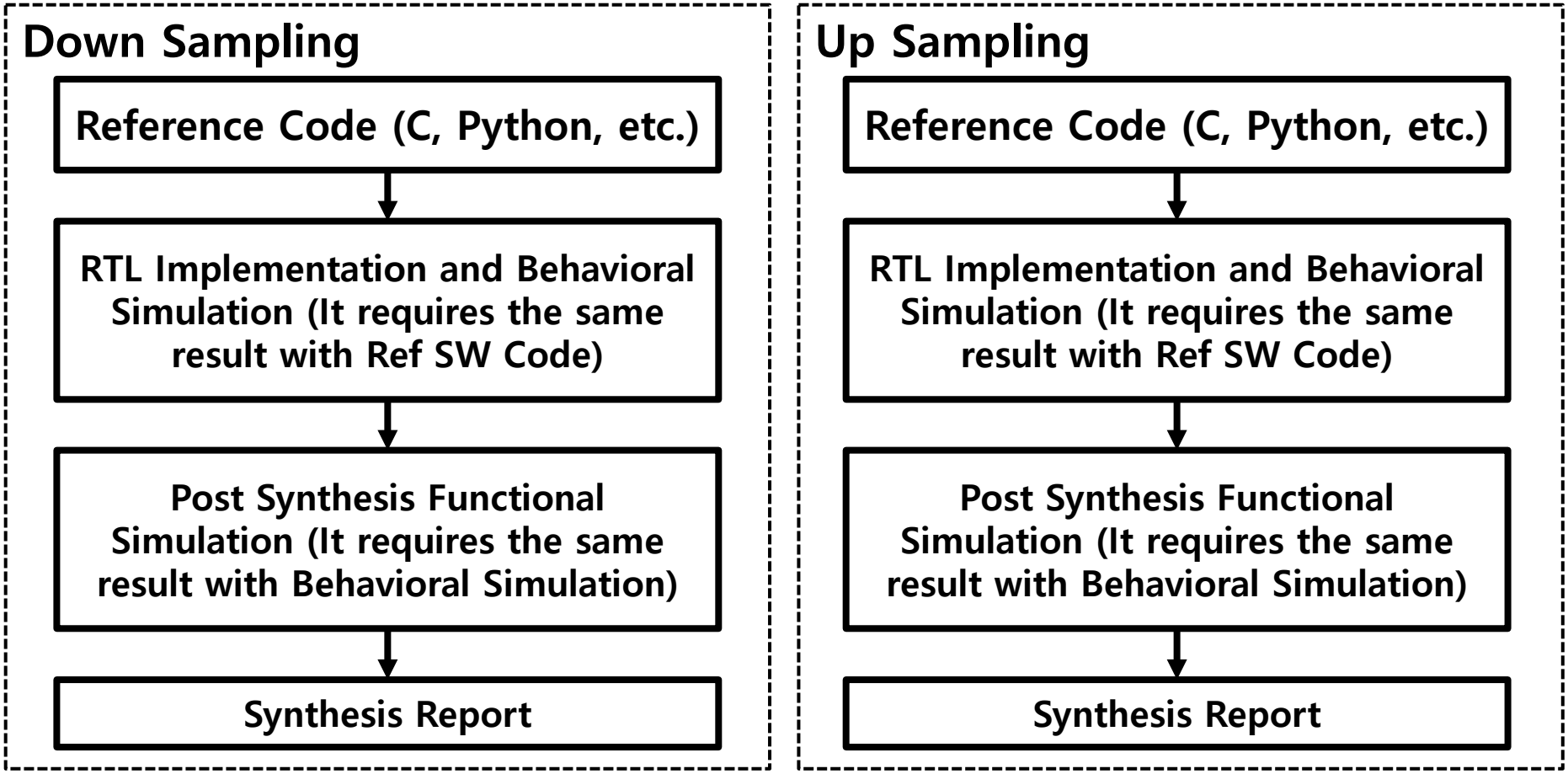
Practice 4: (Down-up sample)

Cascaded Down-Up Sample

- Function: Generate restored pixel data for the same period.
- Method: Wire down-sampler and up-sampler module
- Main circuits:
  - DS & US
  - Data flow controller
- Main signals:
  - system – enable
  - memory access
  - handshake signal



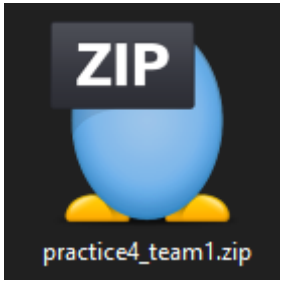
Practice 4 – Design flow

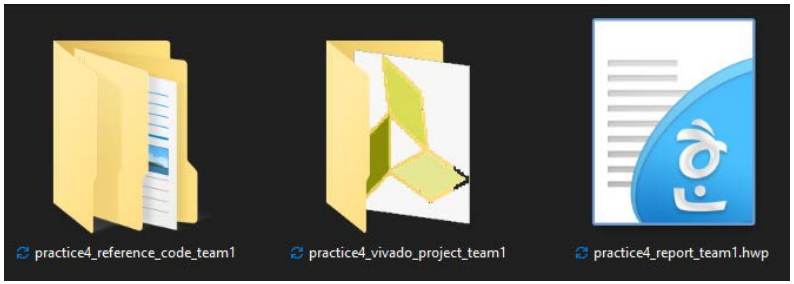


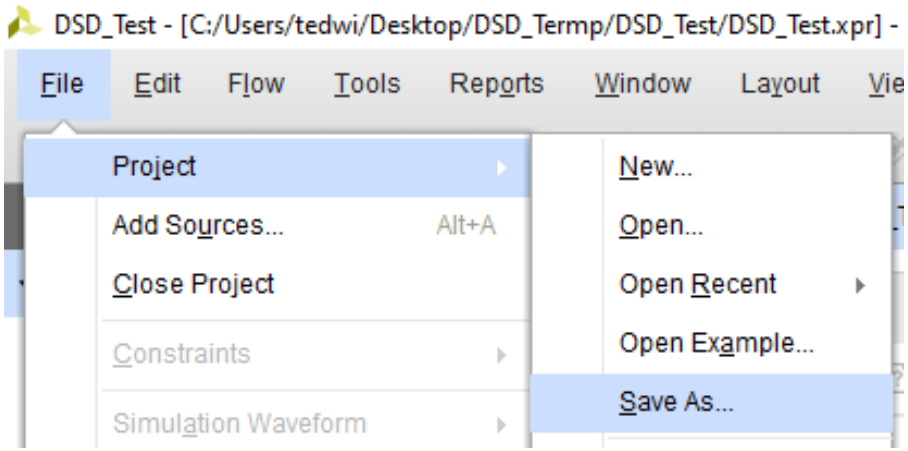
- Outputs to be submitted
  - DS/US reference SW codes
  - DS/US RTL codes
  - Practice 4 report should include following results (PSNR, cycle, HW utilization)

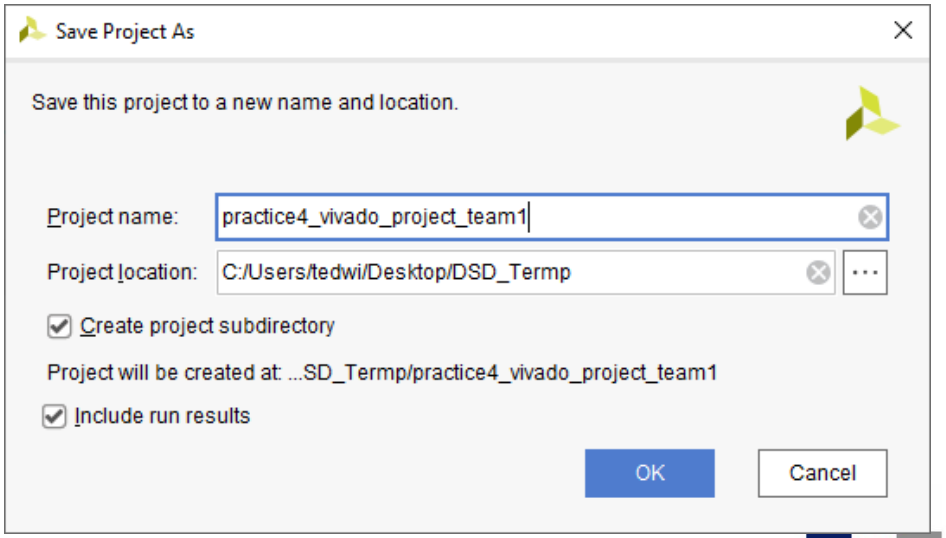
Practice 4 – Files to submit

- Zip file
  - file name: “practice4_team#.zip”
- Practice 4 - Report
  - file name: “practice4_report_tema#.hwp”
- Reference code
  - file name: “practice4_reference_code_team#”
- Vivado Project File
  - file name: “practice4_vivado_project_team#”









Term-project Proposal

◈You have to submit proposal of term-project.

- 1. Abstract
  - (Summarize your project goal, features and performance.
- 2. Objective
  - (Organize your objective and background)
- 3. Preliminary Research
  - *(Organize related knowledge or theories you searched to achieve your goal.)*
- 4. Design Contents
  - *(Summarize features and functions : DS & US algorithm and issues with regard to RTL, synthesis and e.t.c.)*

Term-project Diary

- Summarize and submit weekly term-project progress.

Digital System Design – Project Diary

Date : 2021-xx-xx

Student ID		Name	
Describe what you did in this week, design issues, designed code, and future plans, etc.			

Evaluation Criterion

<table>
<tr><th colspan="2">C</th><th colspan="2">RTL</th><th>Diary</th><th>PSNR</th></tr>
<tr><td>DS</td><td>US</td><td>DS</td><td>US</td><td rowspan="2">3.3 * 3
(=10)</td><td rowspan="2">15</td></tr>
<tr><td>5</td><td>5</td><td>10</td><td>10</td></tr>
<tr><th>Cycle</th><th>Timing</th><th>Utilization</th><th>Proposal</th><th>Demo & Presentation</th><th>Report</th></tr>
<tr><td>5</td><td>5</td><td>5</td><td>5</td><td>5</td><td>10</td></tr>
</table>

- C and Verilog codes are evaluated by the successful operation and the quality of the codes
- We will evaluate your results (especially, PSNRs) using both given images and new test image
- Relative evaluation of PSNR is applied only when C code is successful
- Relative evaluation of cycle is applied only when functional verification is successful
- Relative evaluation of timing and utilization is applied only when synthesis is successful
- If you succeed in an optional mission, we will give you additional 10 points

Schedule

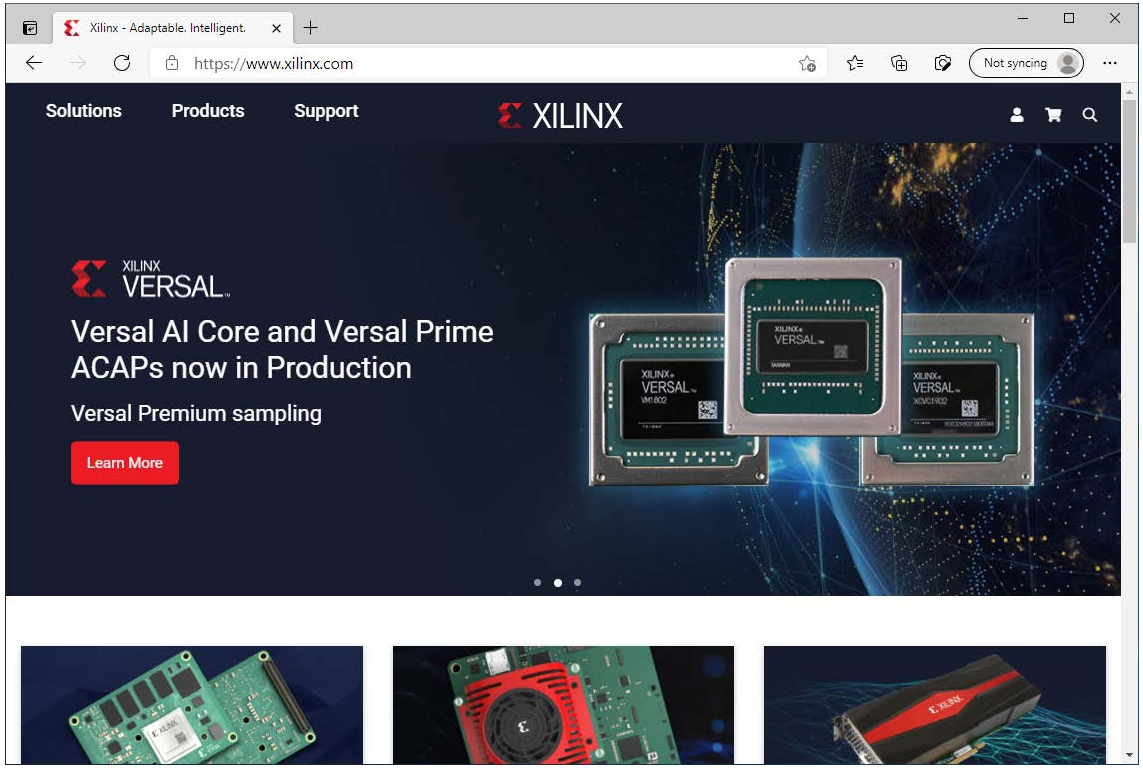
- Diary #1
  - 5/20(Thu.) ~ 5/26(Wed.)
- Diary #2, Practice4, Proposal
  - 5/27(Thu.) ~ 6/2(Wed.)
- Final Exam
  - 6/3(Thu.)
- Diary #3
  - 6/3(Thu.) ~ 6/9(Wed.)
- Term-project report & files
  - 6/10(Thu.) ~ 6/16(Wed.)
- Term-project demo & presentation
  - 6/17(Thu.)

Vivado - Download

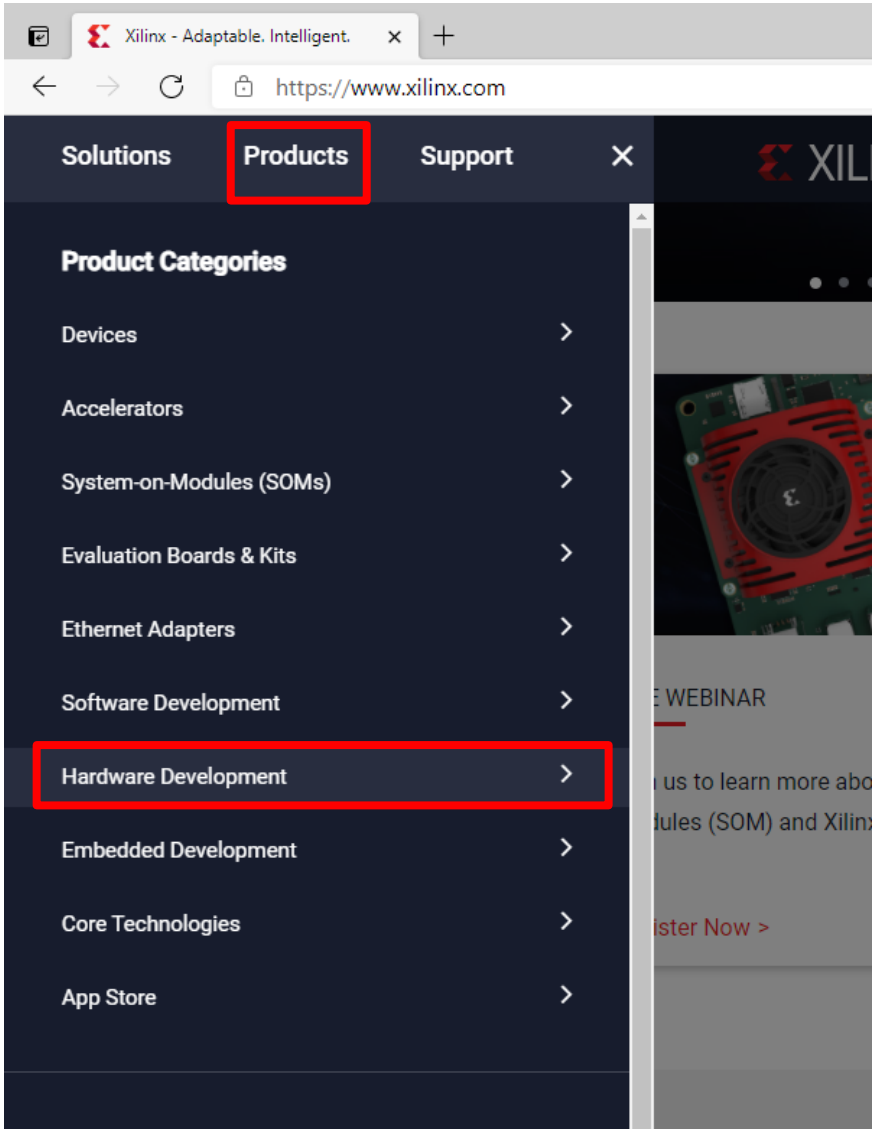
- About FPGA
- Design Flow
- Project explanation
- Vivado – Download
- Vivado – Create project
- Vivado – Add/Create sources
- Vivado – Simulation
- Vivado – Example: Simple PL Design
- Project – Reference code review (Video)
- Project – Vivado given sample code review (Video)
- Project – Vivado testbench examples (Video)

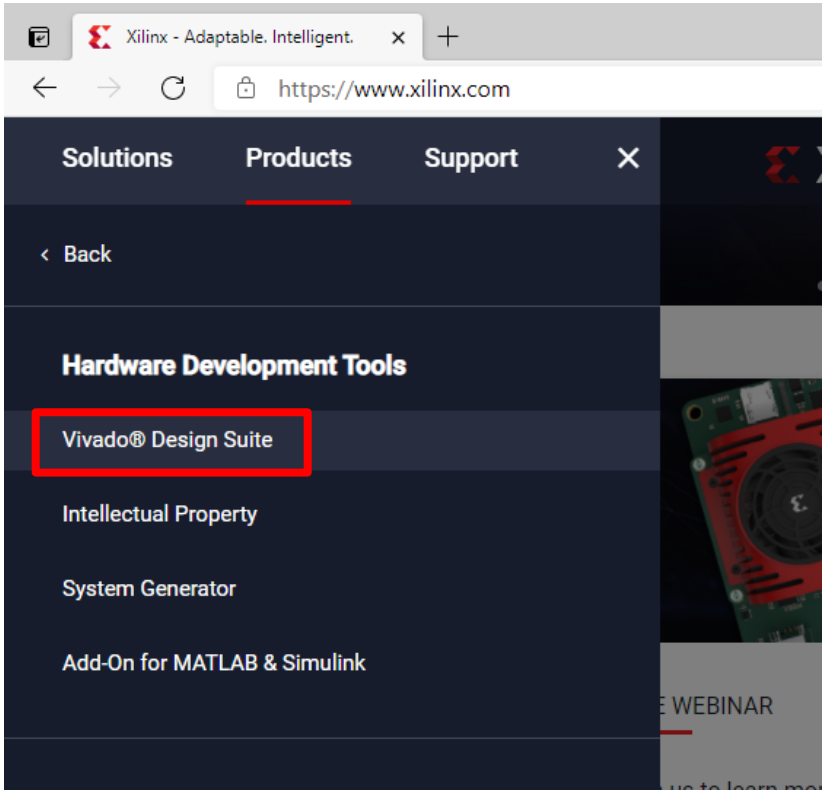
Download Vivado

- In order to download Vivado, you have to create a Xilinx account.
- https://www.xilinx.com/

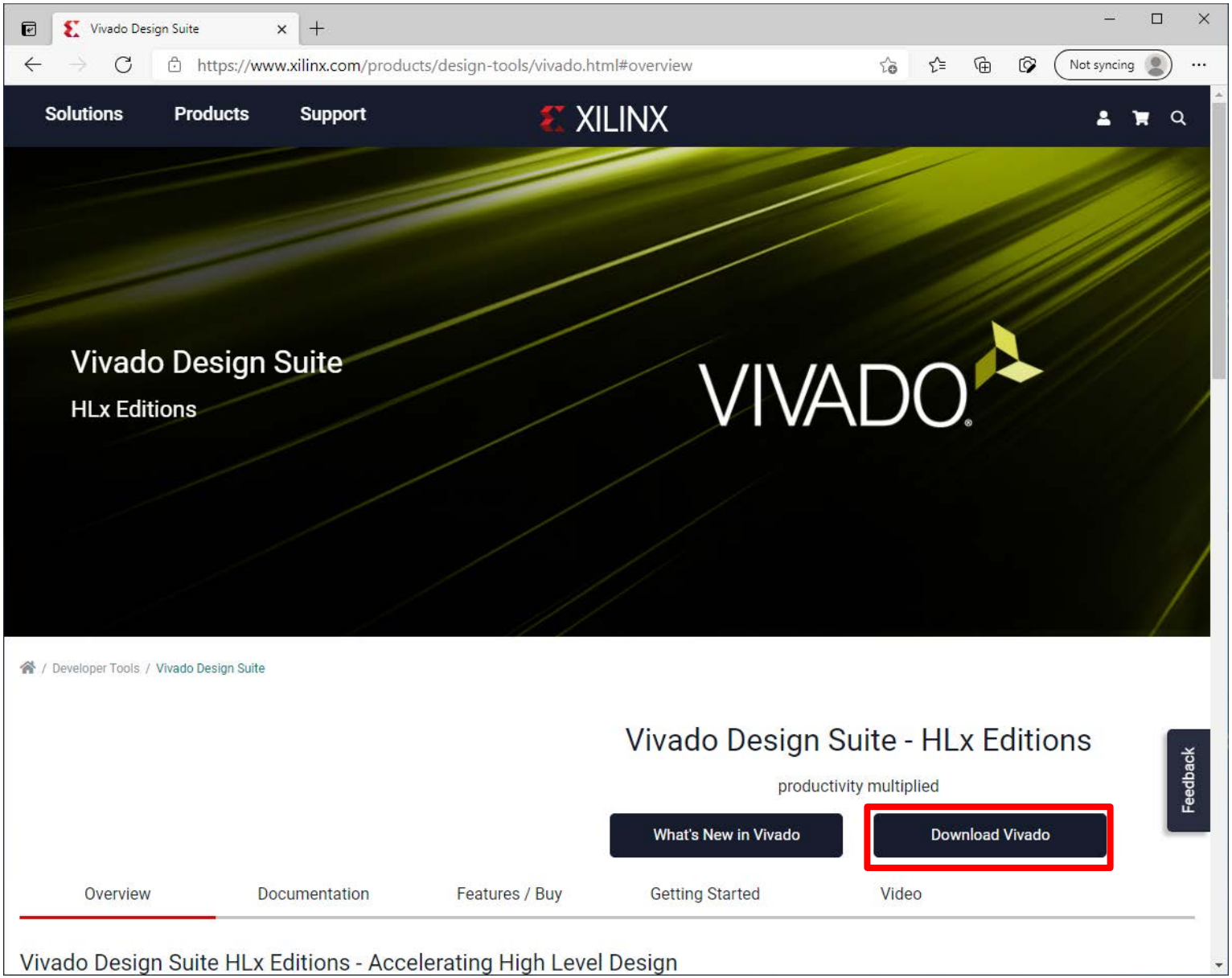


Download Vivado





Download Vivado



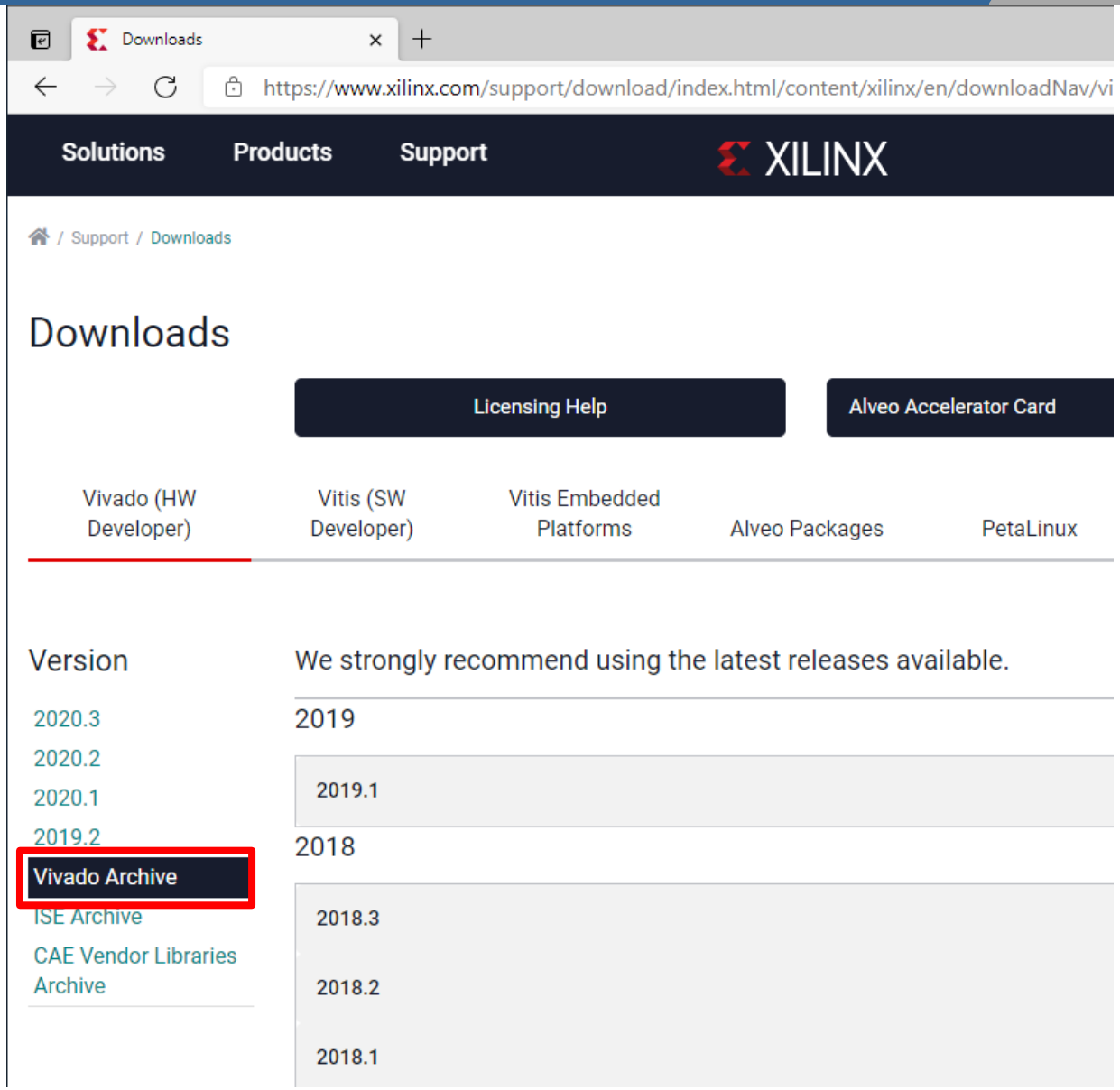
Download Vivado

- In this lab, we are going to use "Vivado HL WebPACK Edition".

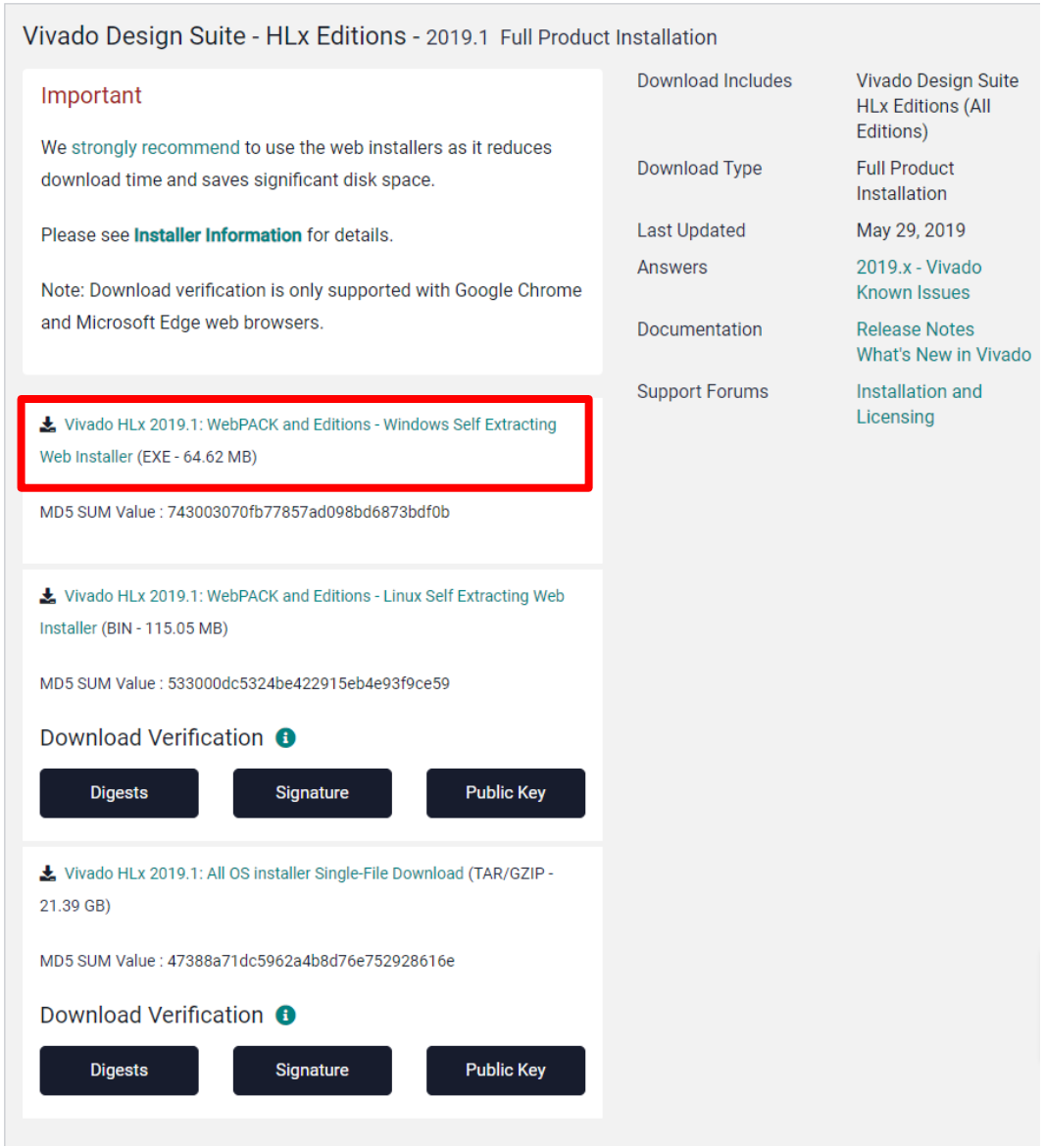
Vivado Design Suite - HLx Editions

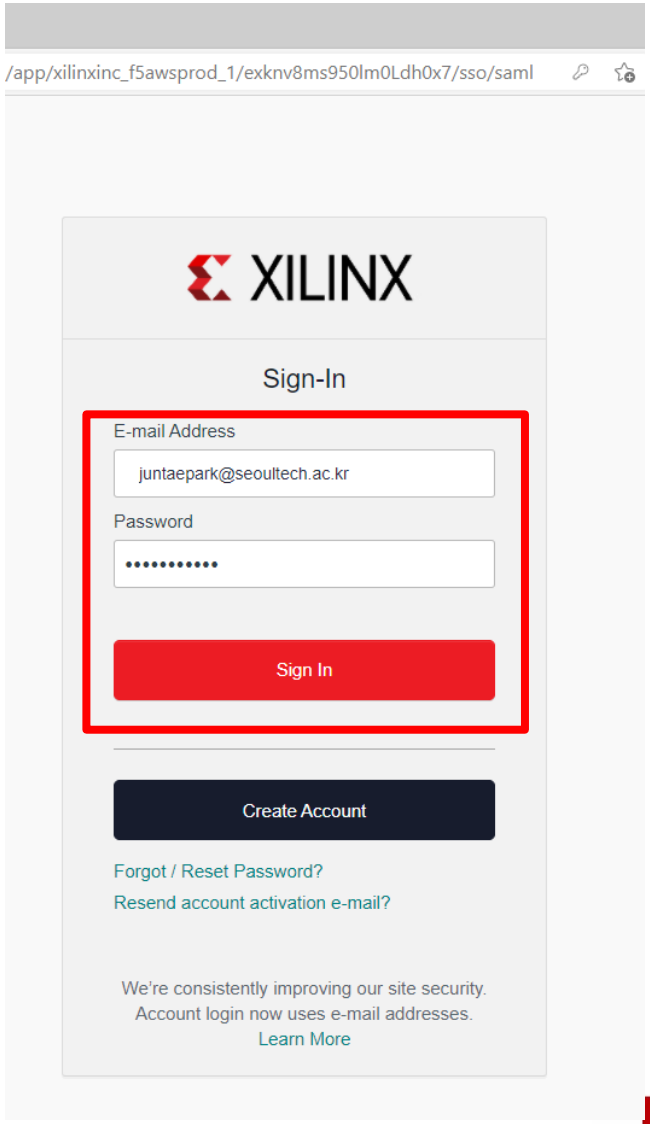
Vivado Design Suite - HLx Edition Features	Vivado HL Design Edition	Vivado HL System Edition	Vivado Lab Edition	Vivado HL WebPACK Edition (Device Limited)	Free 30-day Evaluation
Accelerating Implementation					
Synthesis and Place and Route	•	•		•	•
Dynamic Function eXchange	•	•		•	•
Accelerating Verification					
Vivado Simulator	•	•		•	•
Vivado Device Programmer	•	•	•	•	•
Vivado Logic Analyzer	•	•	•	•	•
Vivado Serial I/O Analyzer	•	•	•	•	•
Debug IP (ILA/VIO/IBERT)	•	•		•	•
Accelerating High Level Design					
Vivado High-Level Synthesis	•	•		•	•
Vivado IP Integrator	•	•		•	•
System Generator for DSP	*	•		*	•
Add-on for Matlab and Simulink	*	•		*	•

Download Vivado



Download Vivado





Download Vivado

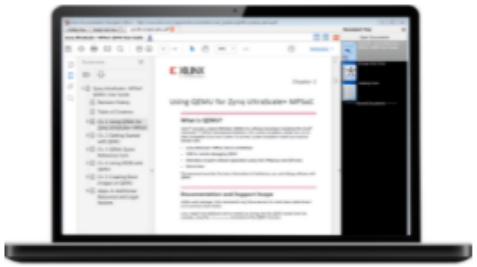
Job Function*

Student

For more information about how we process your personal information, please see our privacy policy.

Download

New to Vivado HLx?



Visit the Vivado HLx Getting Started page to launch your design! These resources include:

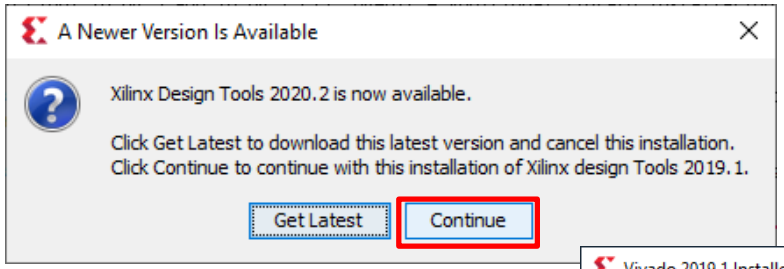
- Installation and Licensing information
- Best Practices, Getting Started and Methodology Guides
- On Demand Videos
- Development Boards and Kits
- Support Resources

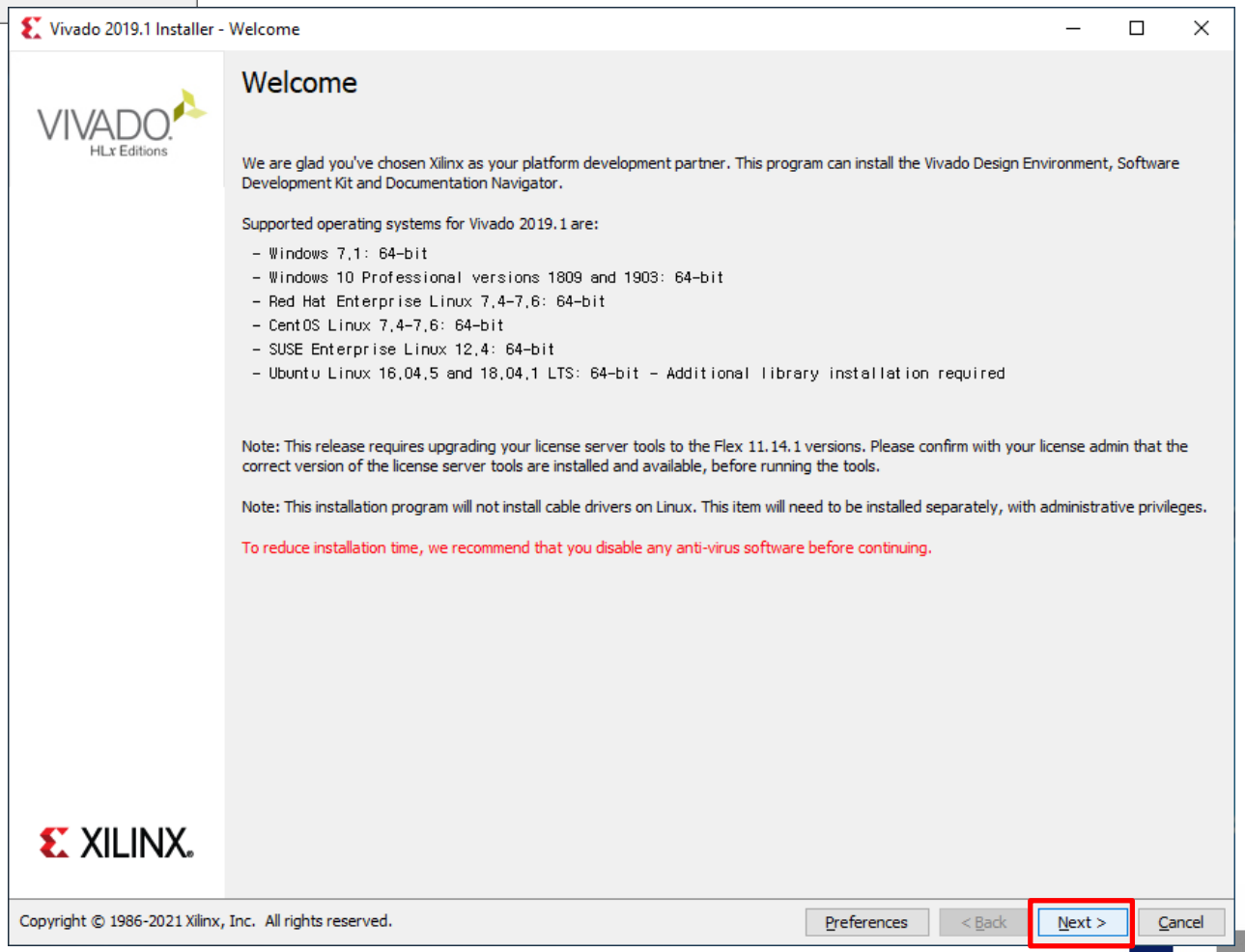
Thank you for download Vivado HLx!

Download Vivado

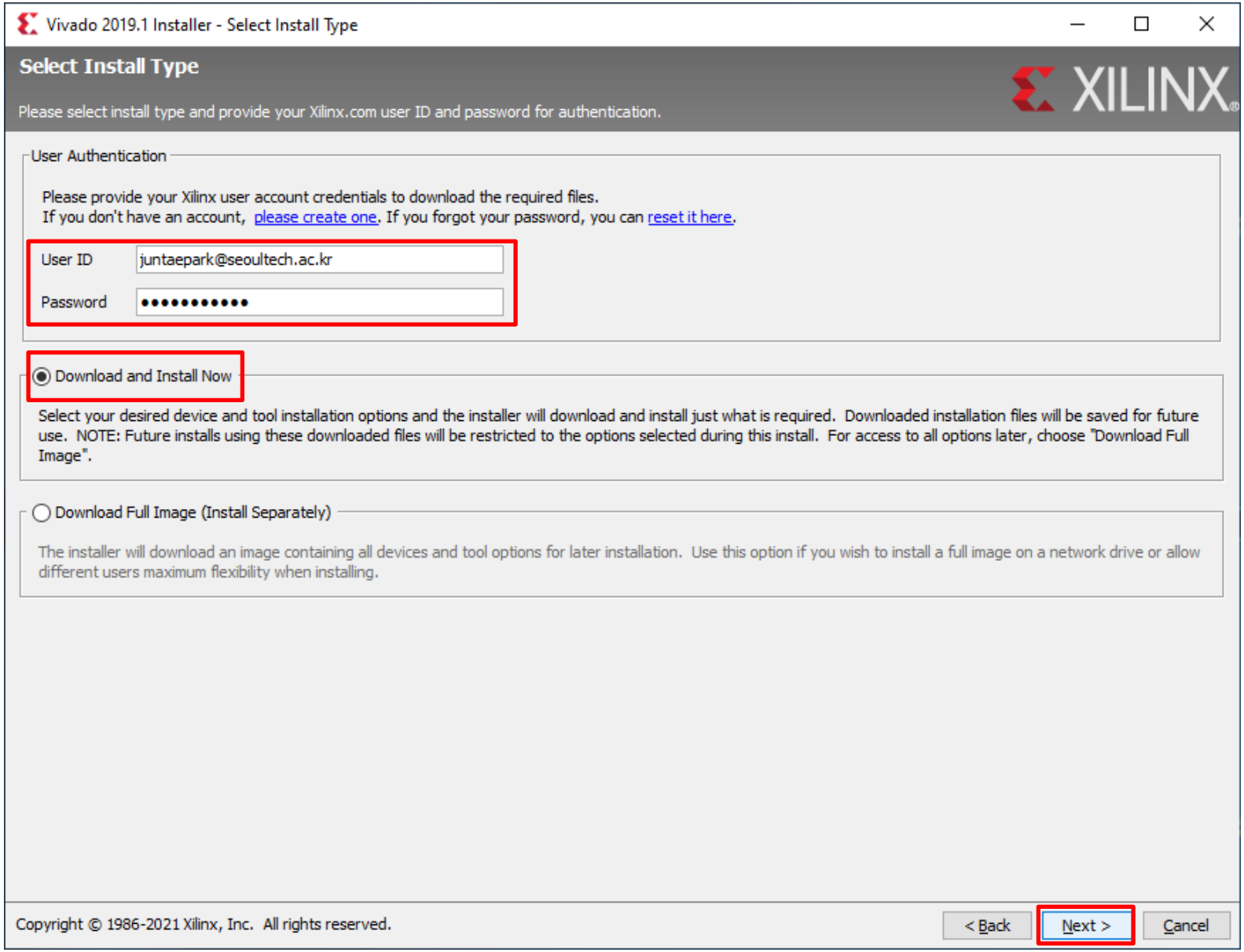
- Execute *.exe file to install Vivado

Download Vivado

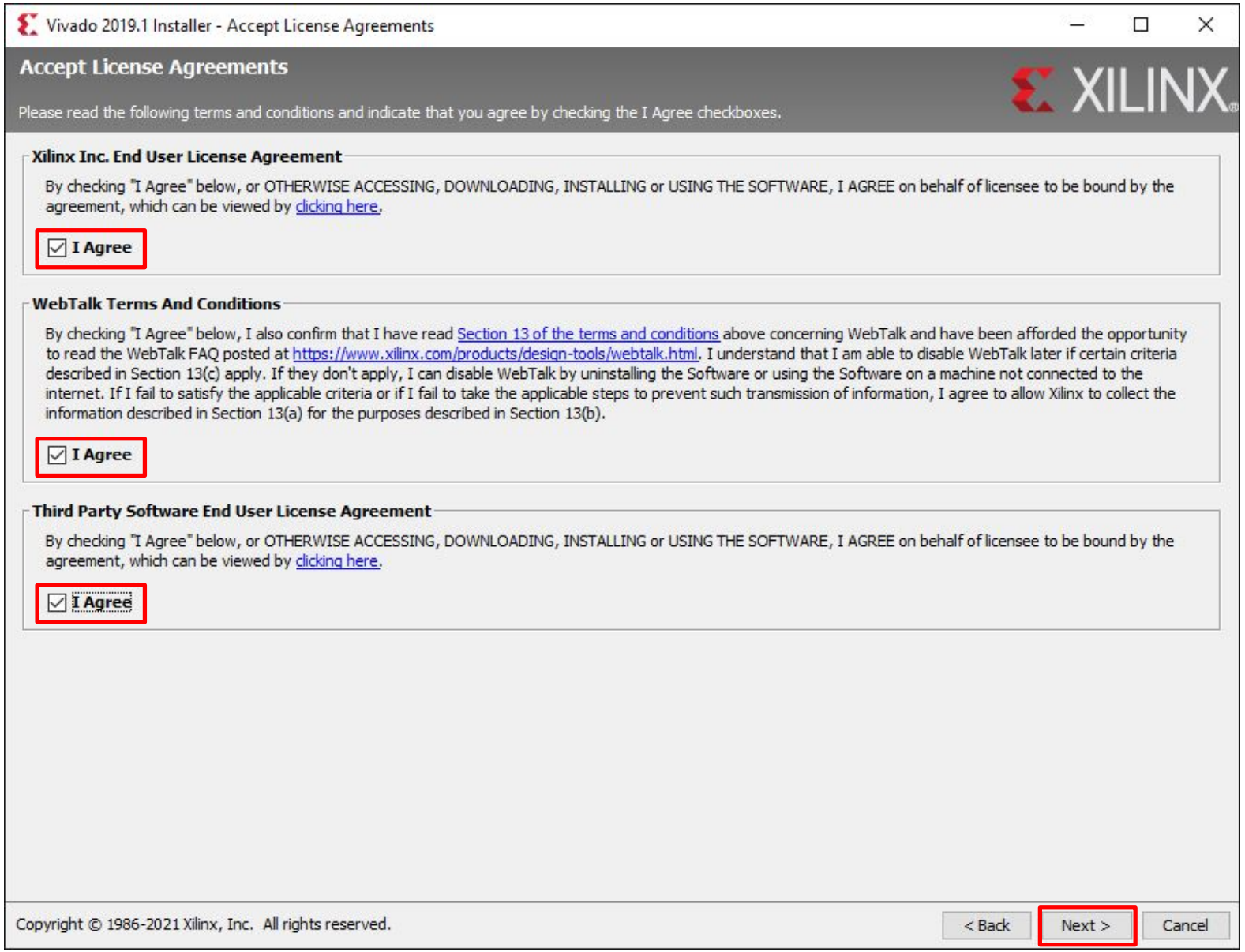




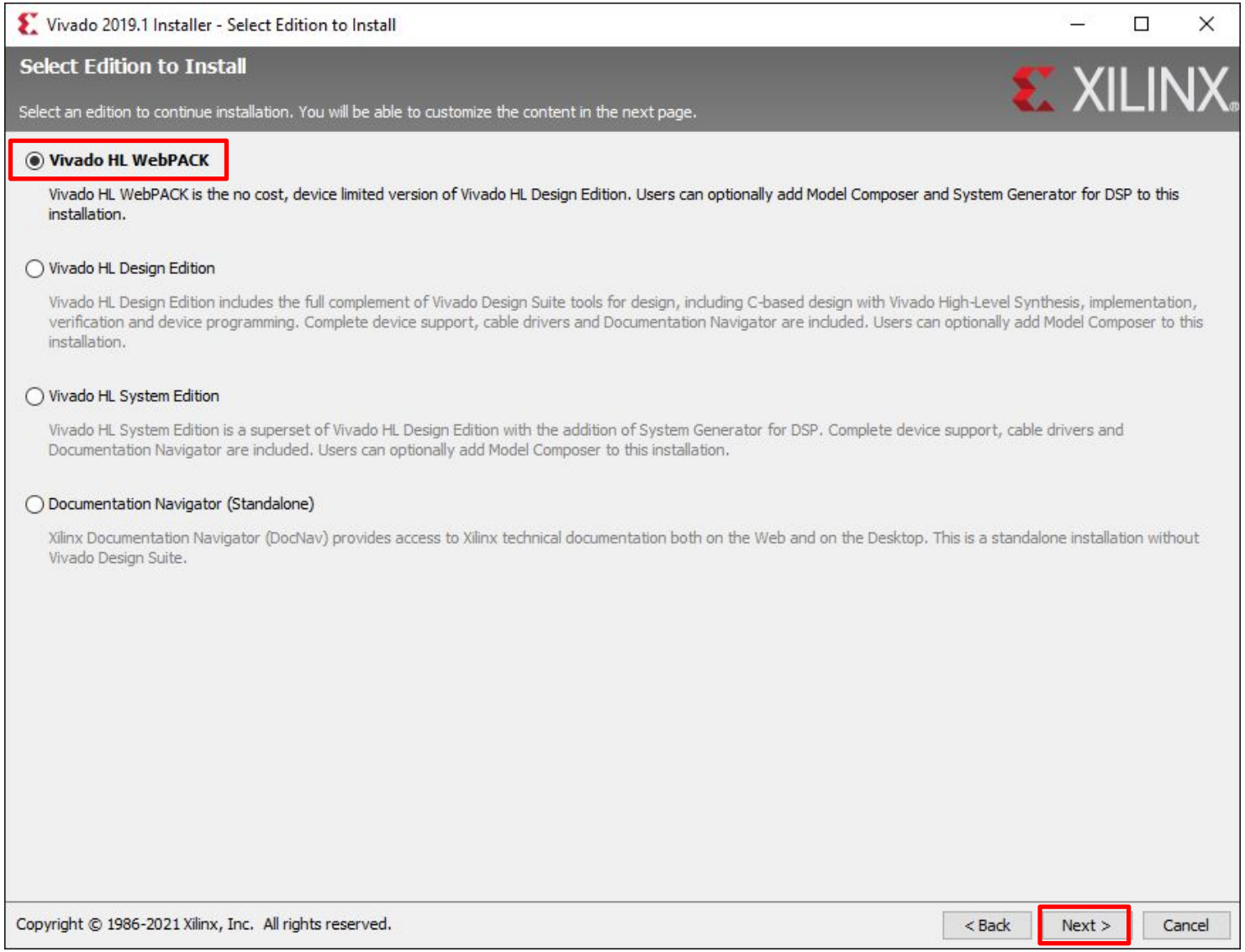
Download Vivado



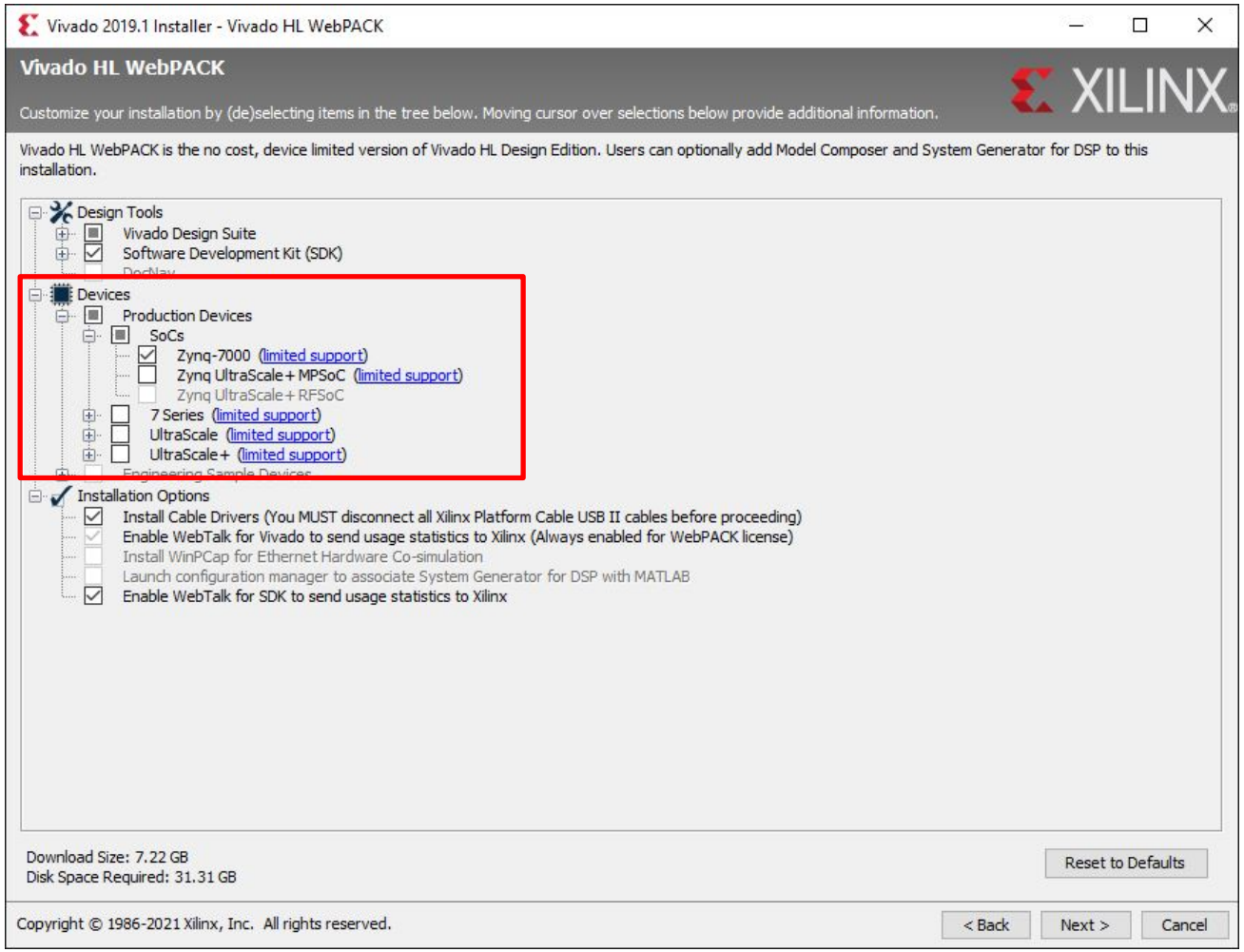
Download Vivado



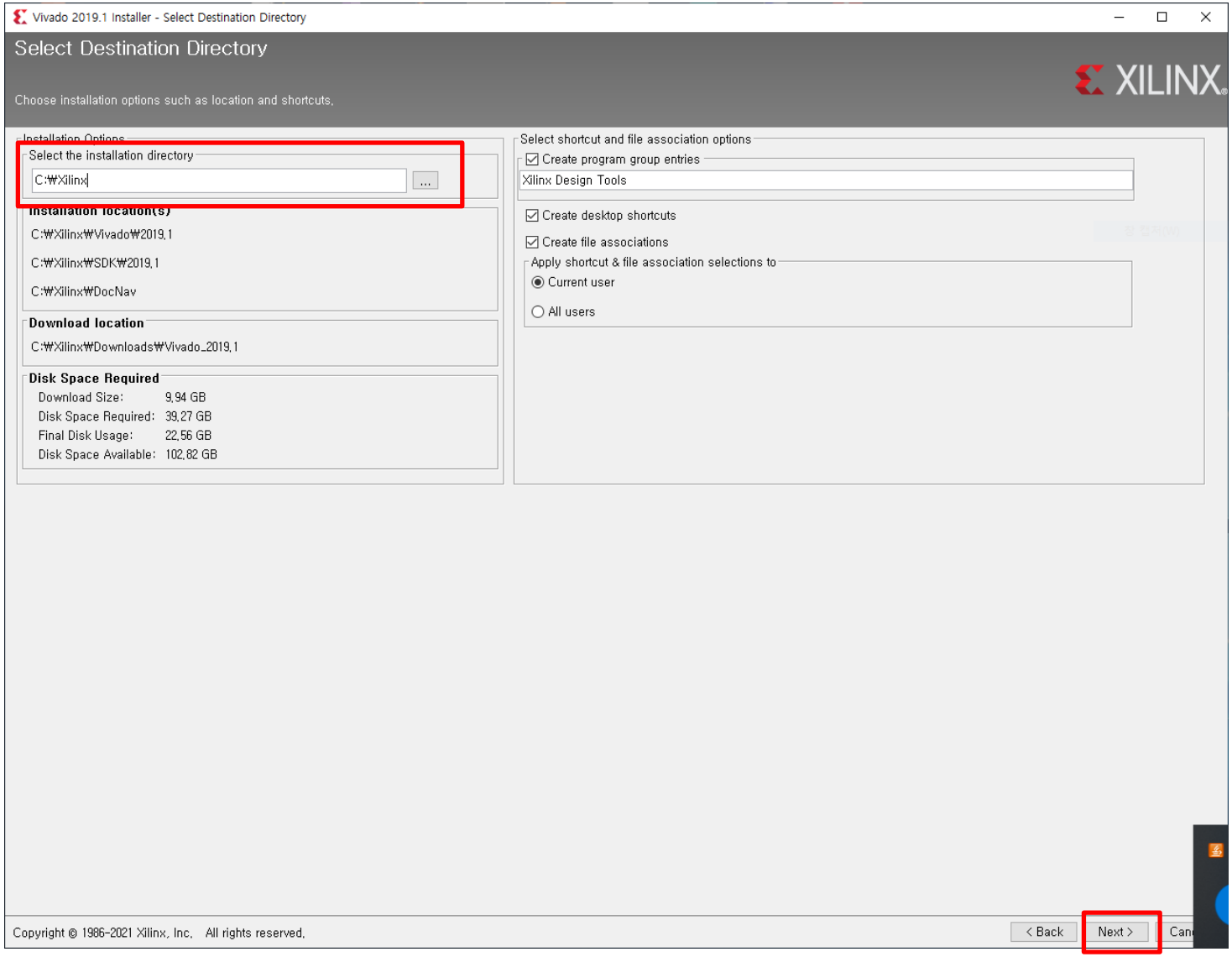
Download Vivado



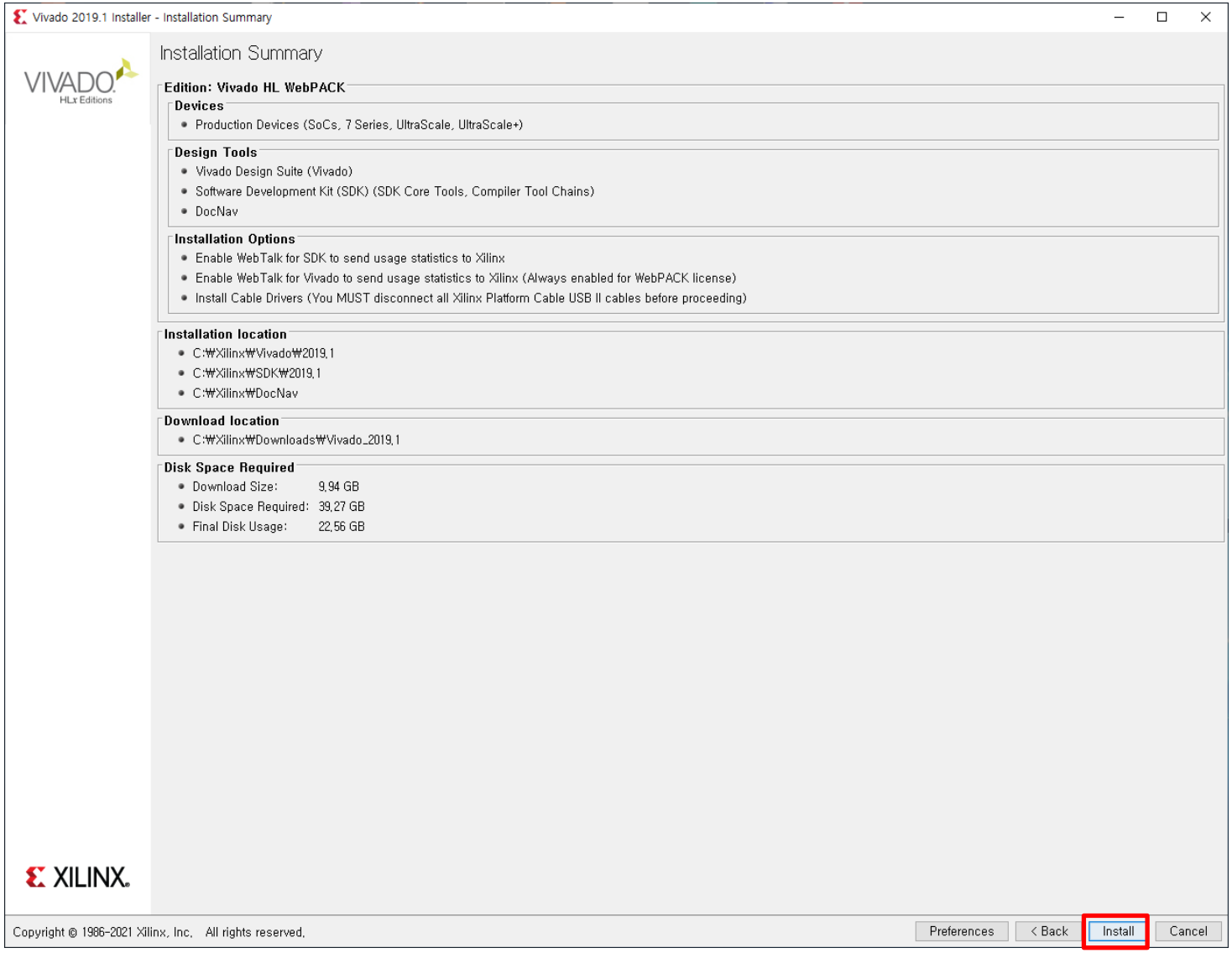
- In order to reduce required disk space, uncheck devices except for Zynq-7000 series which include our FPGA chip. Of course, it's not necessary if you're computer has enough free disk space.



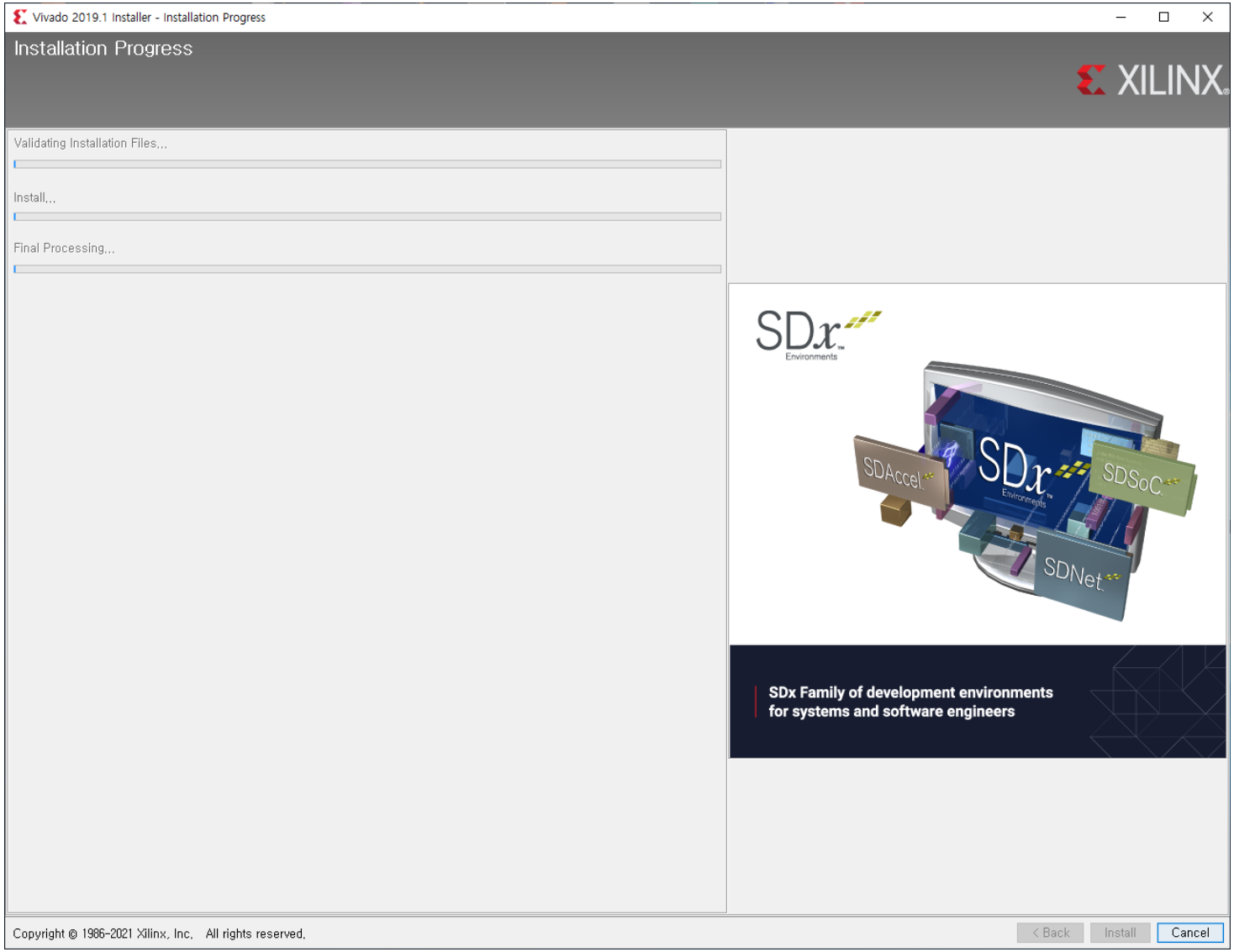
Download Vivado



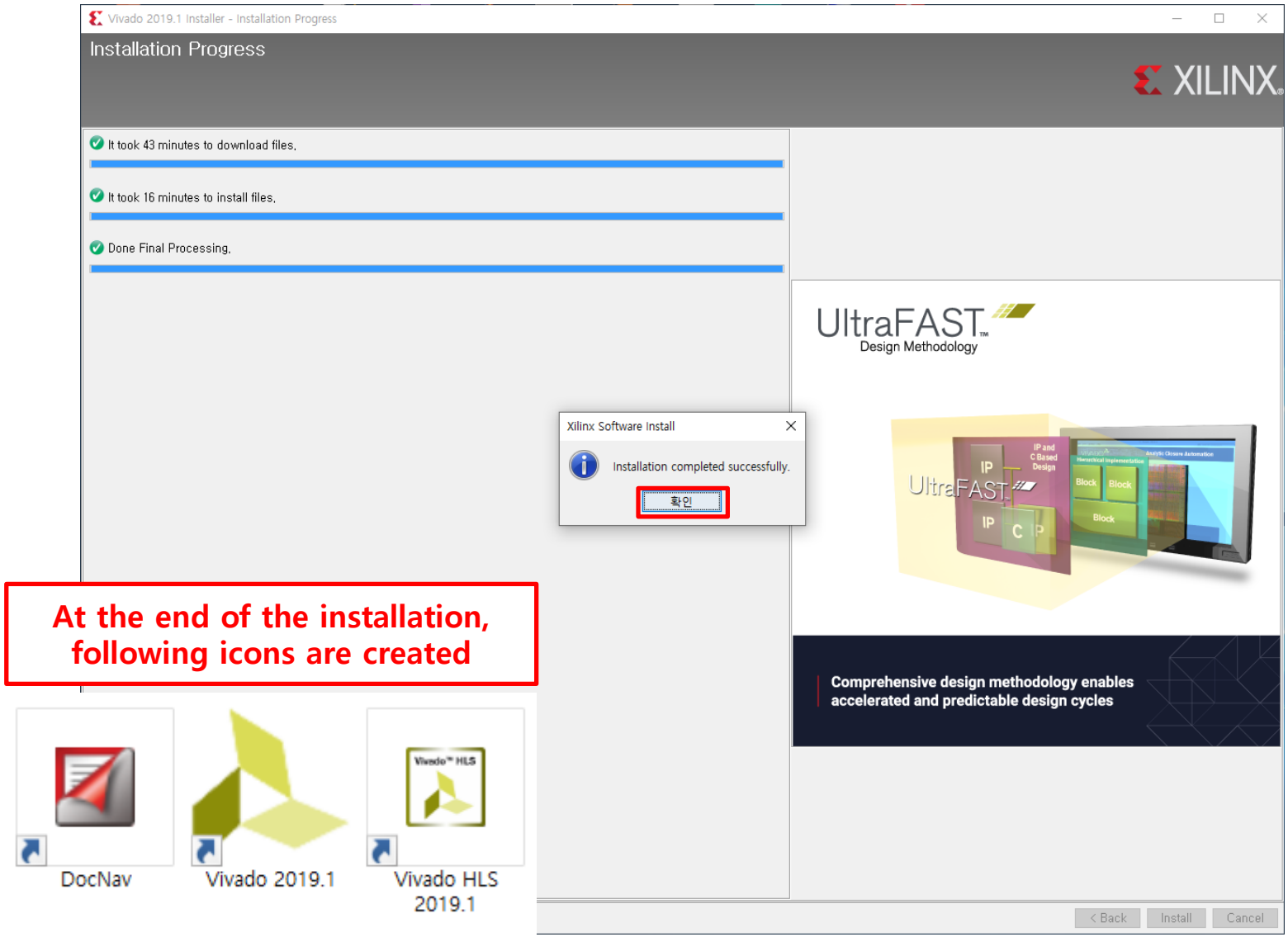
Download Vivado



Download Vivado



Download Vivado

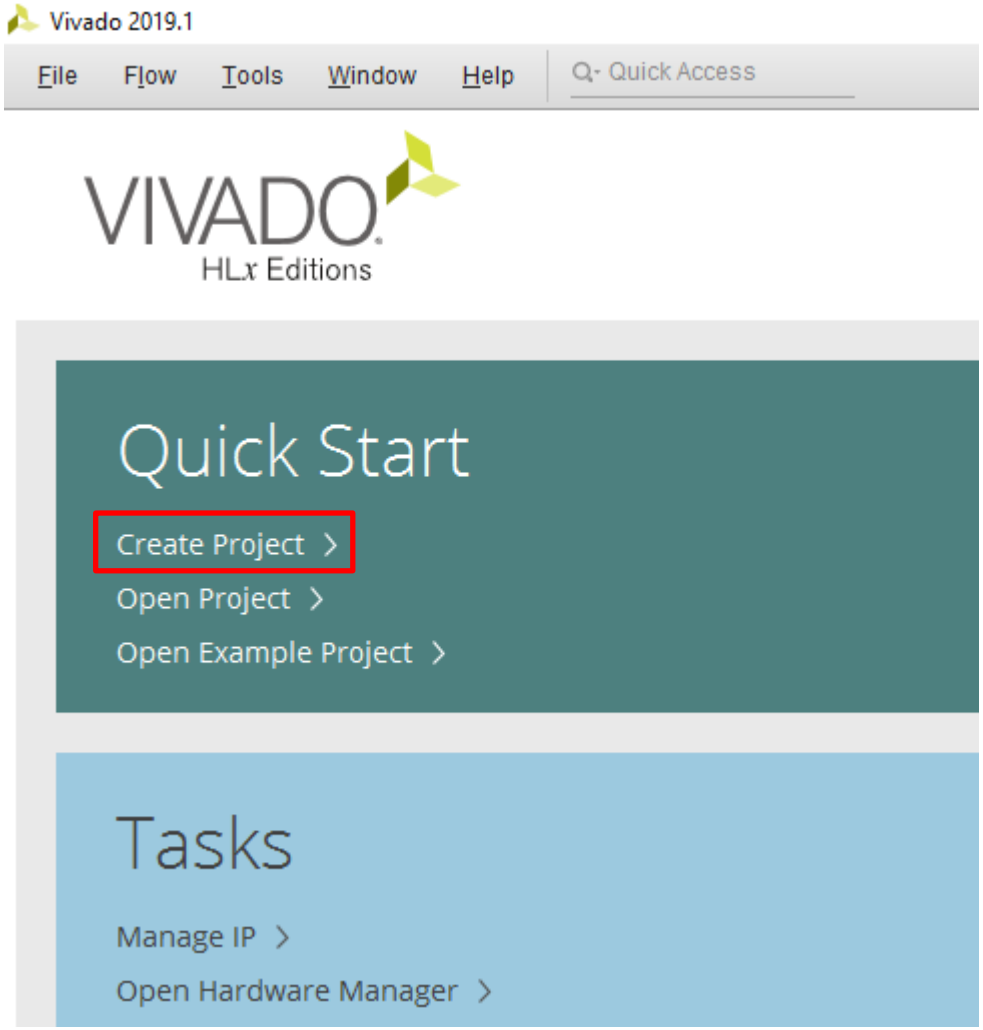


At the end of the installation, following icons are created

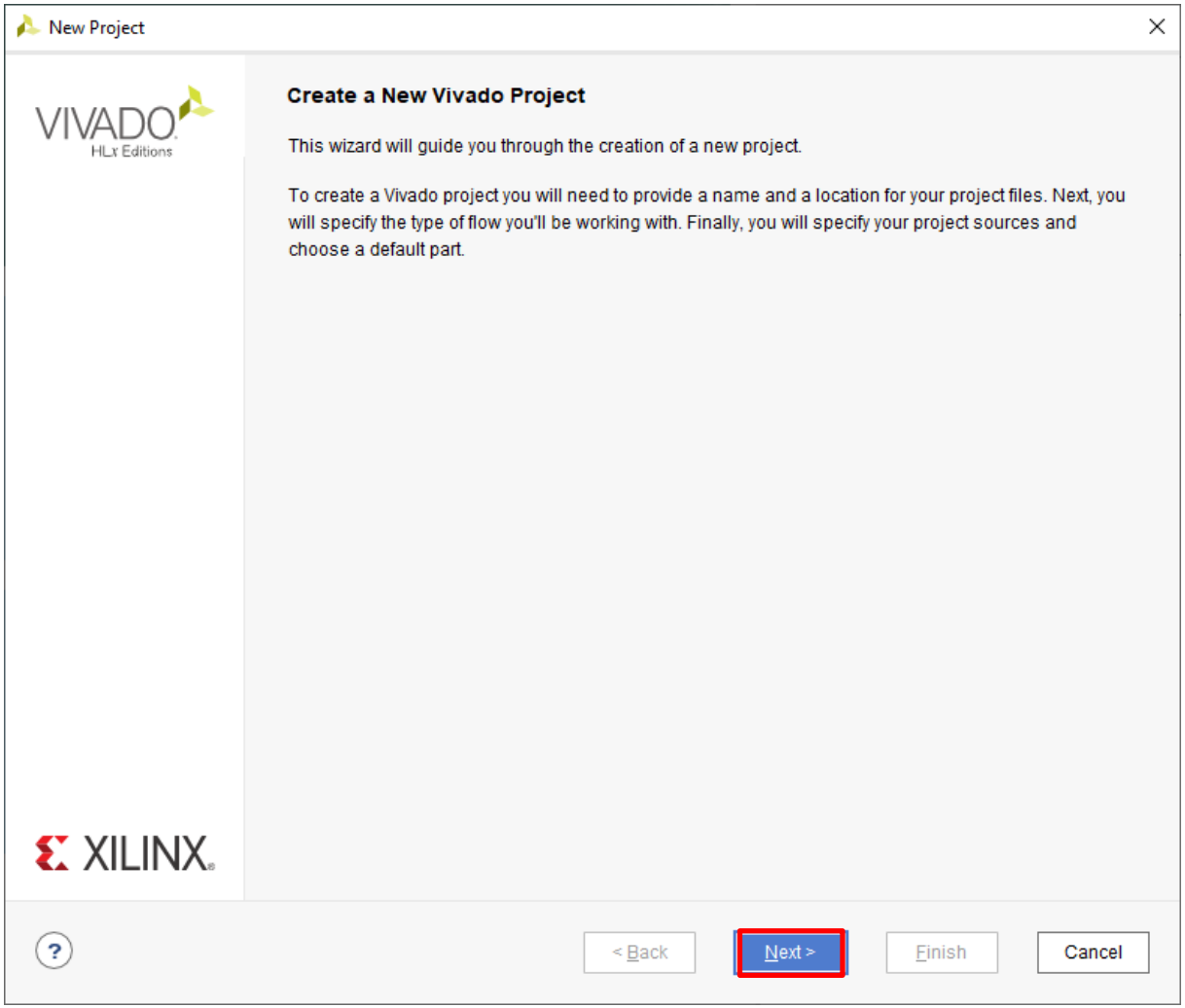
Vivado – Create project

- About FPGA
- Design Flow
- Project explanation
- Vivado – Download
- Vivado – Create project
- Vivado – Add/Create sources
- Vivado – Simulation
- Vivado – Example: Simple PL Design
- Project – Reference code review (Video)
- Project – Vivado given sample code review (Video)
- Project – Vivado testbench examples (Video)

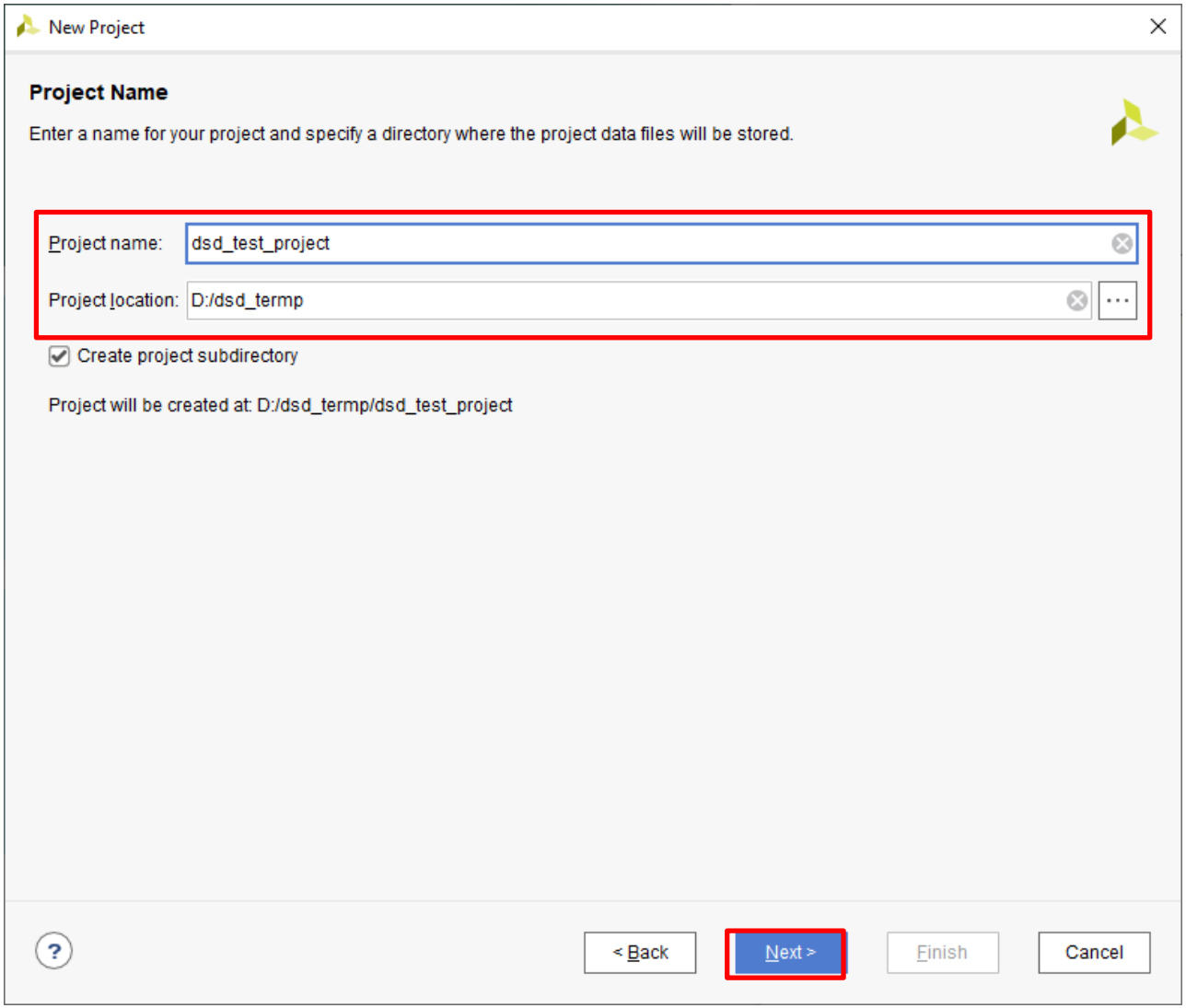
Vivado – Create Project



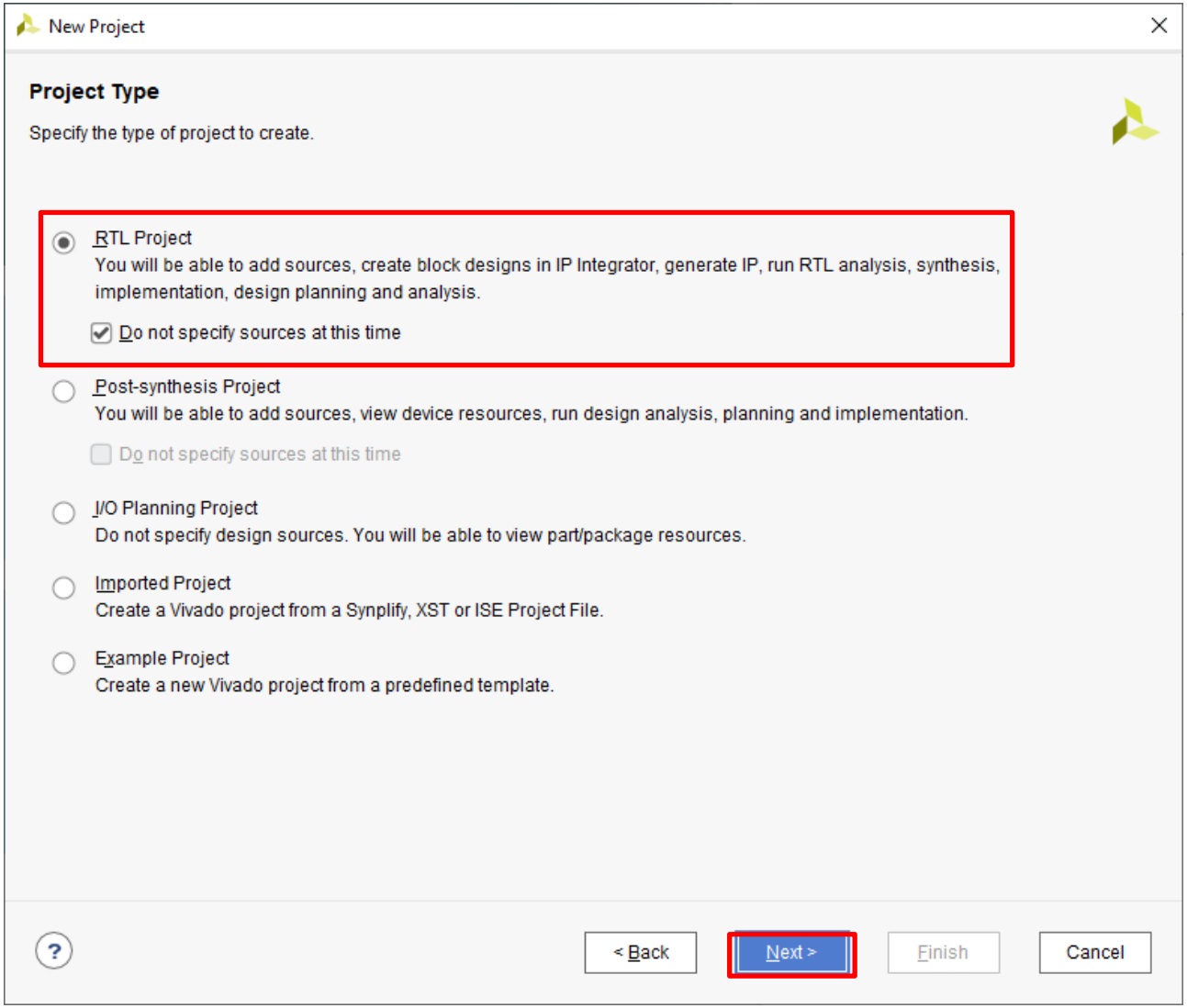
Vivado – Create Project



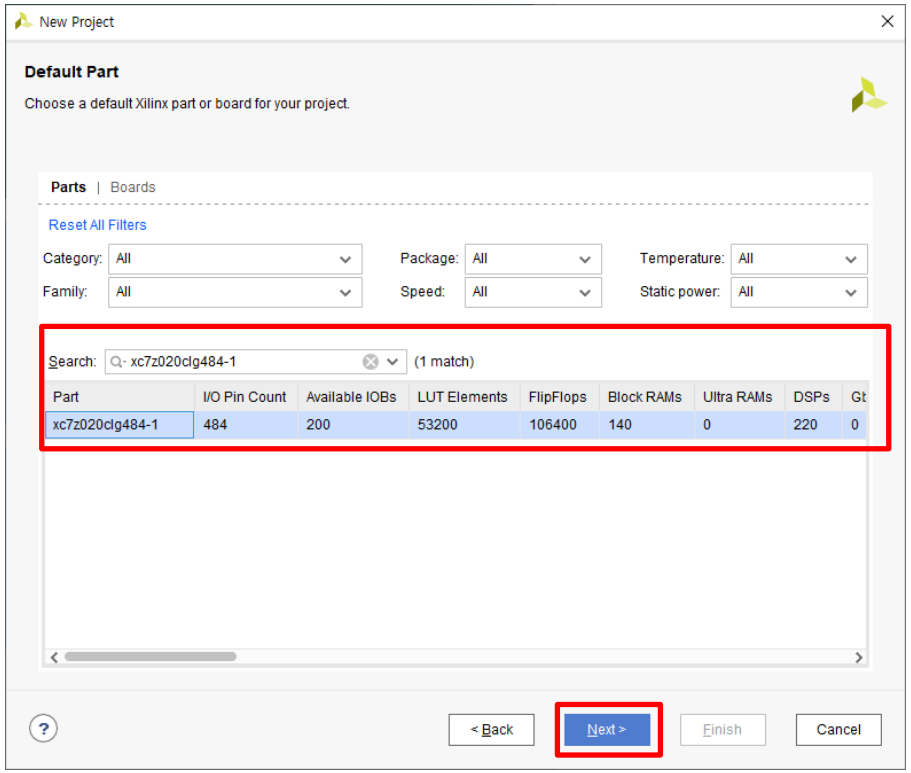
Vivado – Create Project

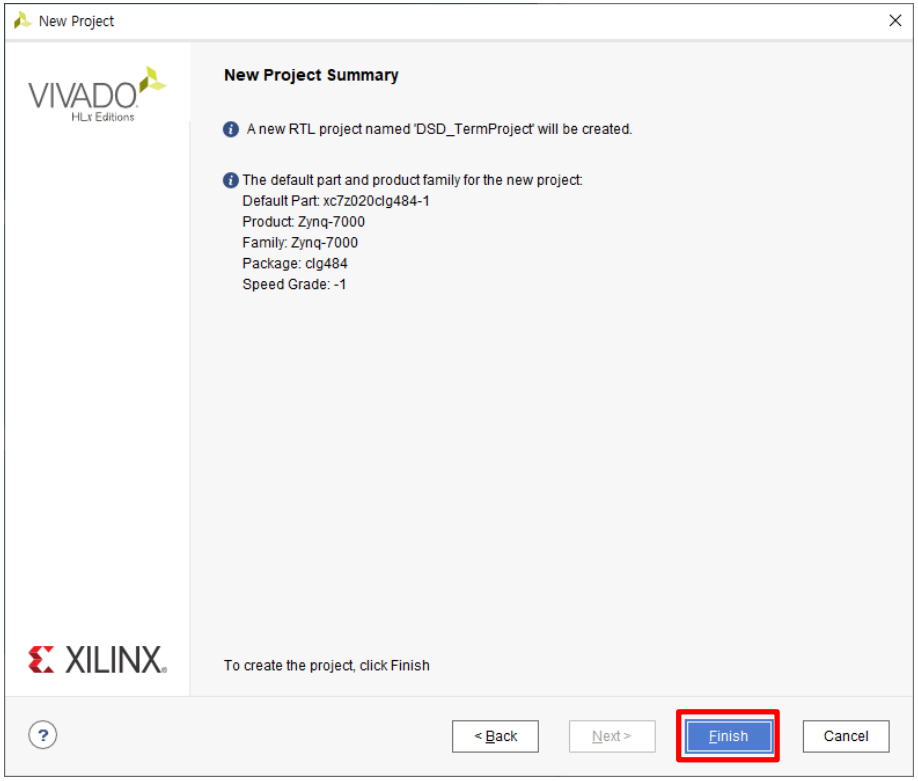


Vivado – Create Project



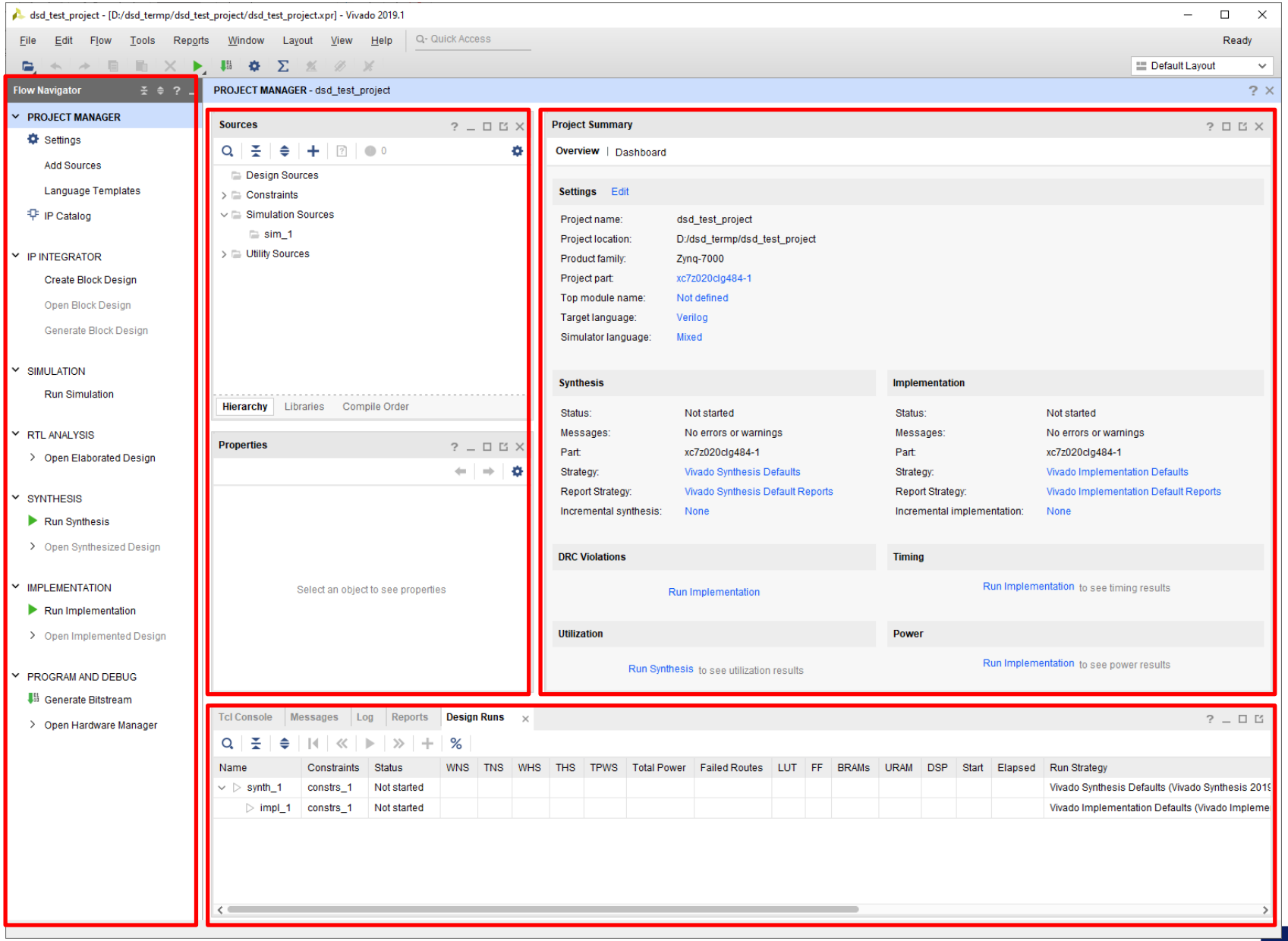
Vivado – Create Project



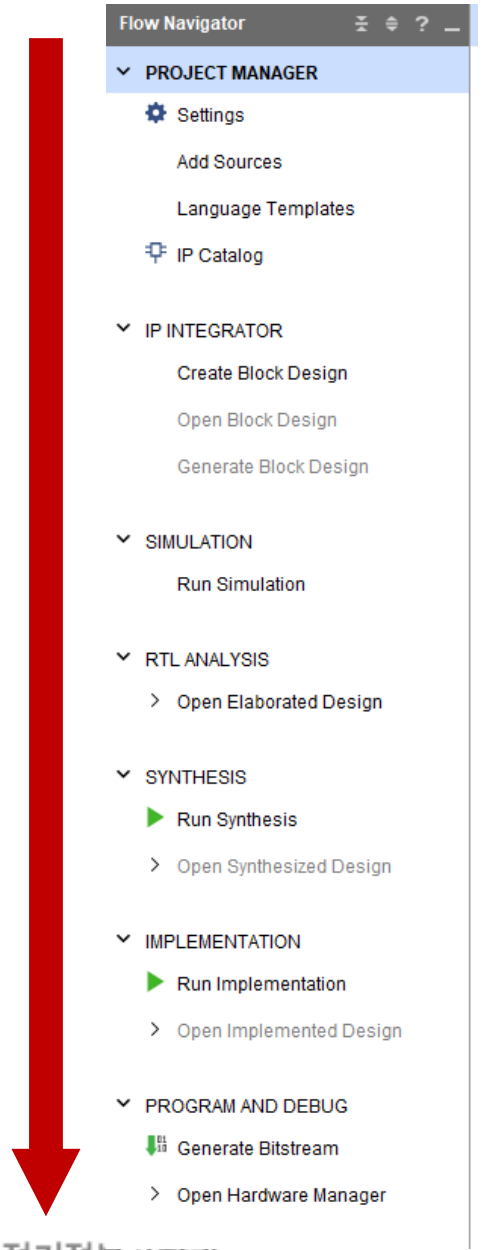


Parts name : xc7z020clg484-1

Vivado – Create Project



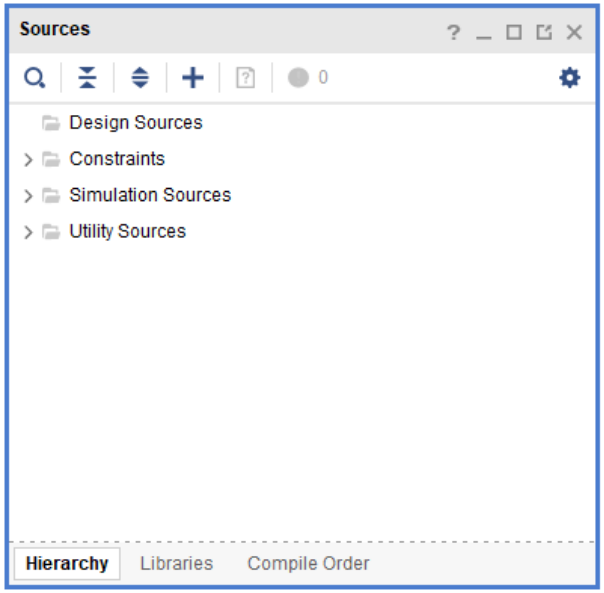
Vivado - Default Windows

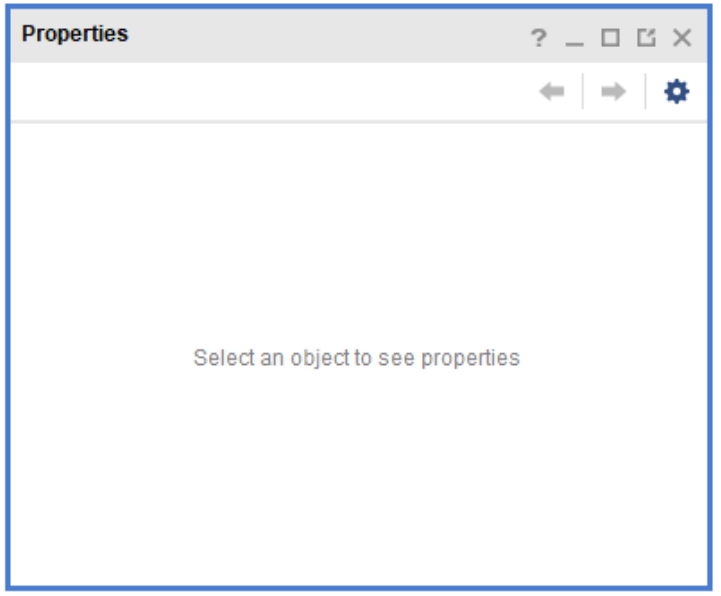


- You can see the flow navigator on the left side of Vivado.
- Main tools are located from top to bottom. In order to generate bit file and program it onto FPGAs, you have to follow this flow.

Vivado - Default Windows

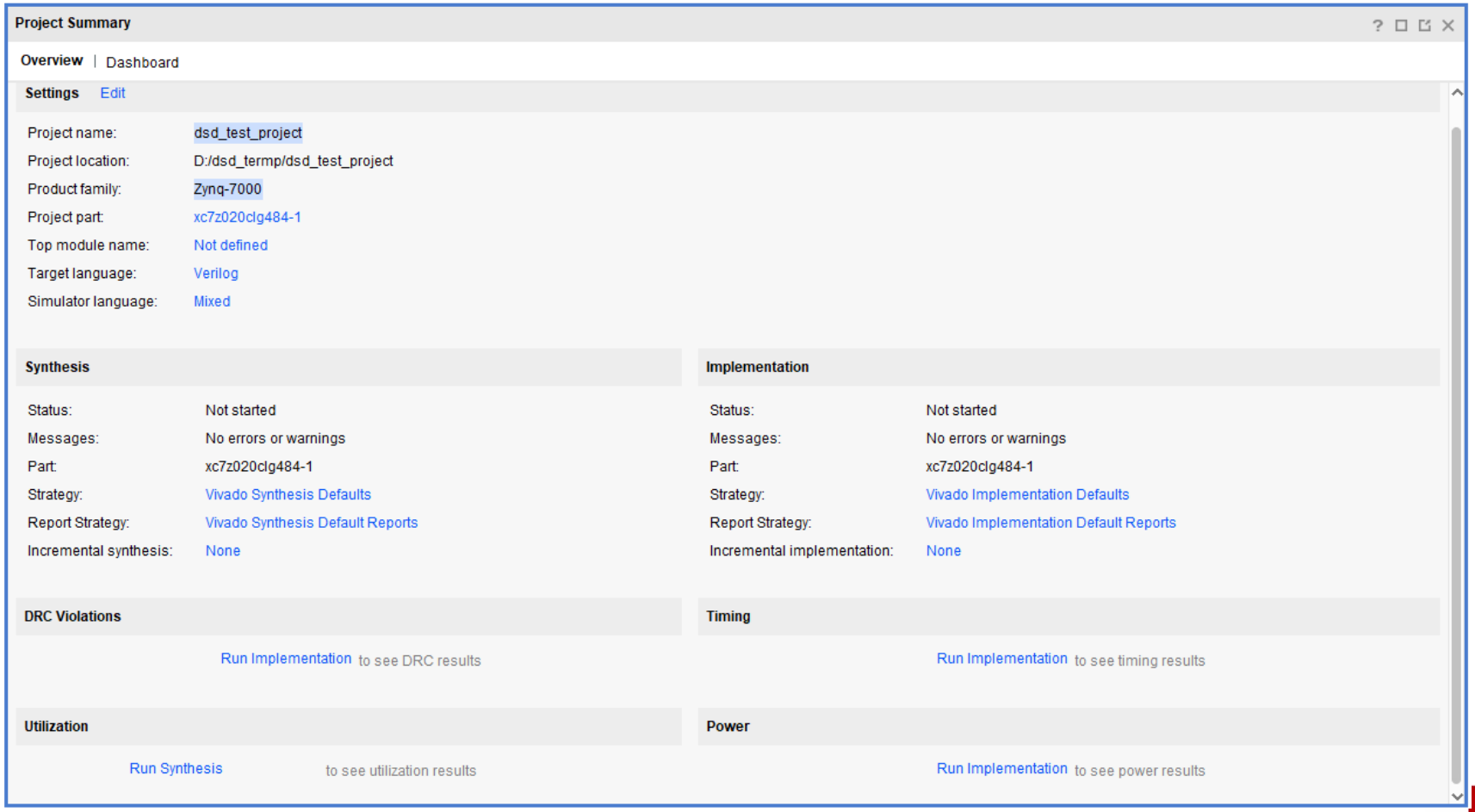
- In sources window, you can add or remove source files such as design sources, constraints and simulation sources.
- In properties window, you can see the properties of object.





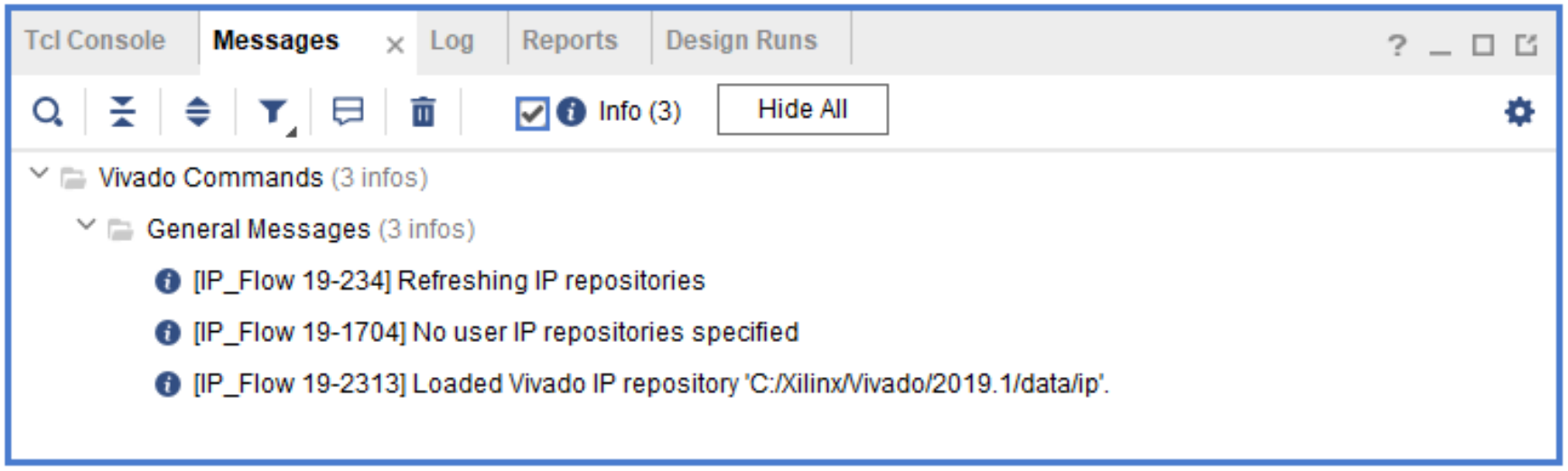
Vivado - Default Windows

- In project summary window, you can see the major information of design.



Vivado - Default Windows

- In the below window, there are 5 tabs.
- We will discuss it later.



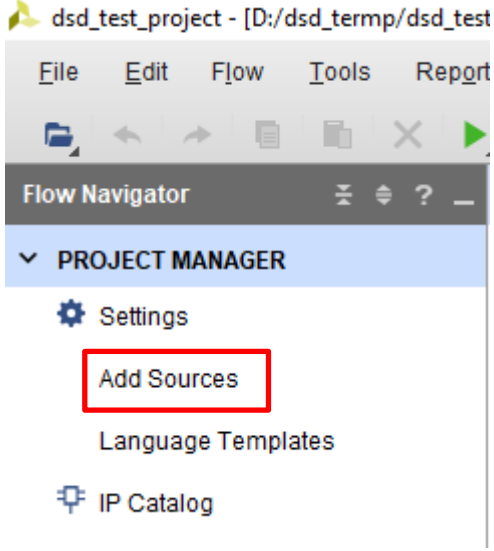
Vivado – Add/Create sources

- About FPGA
- Design Flow
- Project explanation
- Vivado – Download
- Vivado – Create project
- Vivado – Add/Create sources
- Vivado – Simulation
- Vivado – Example: Simple PL Design
- Project – Reference code review (Video)
- Project – Vivado given sample code review (Video)
- Project – Vivado testbench examples (Video)

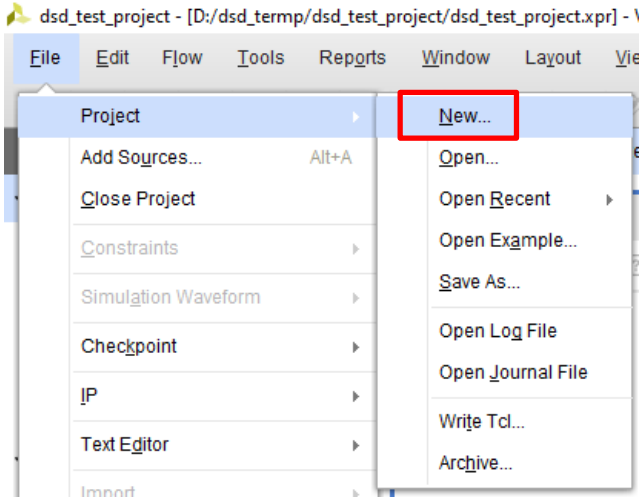
Vivado – Add sources

- You can add sources by following ways.

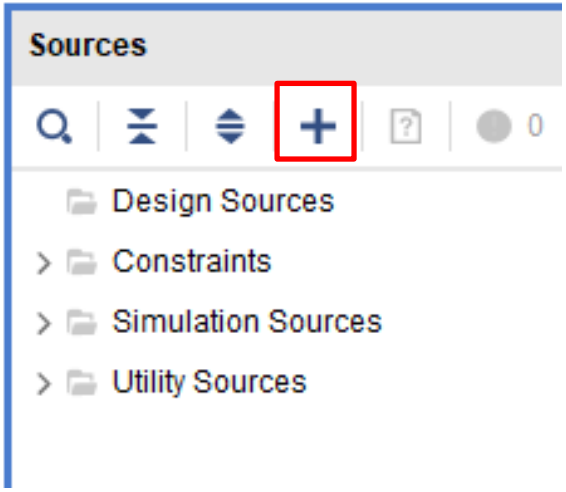
1



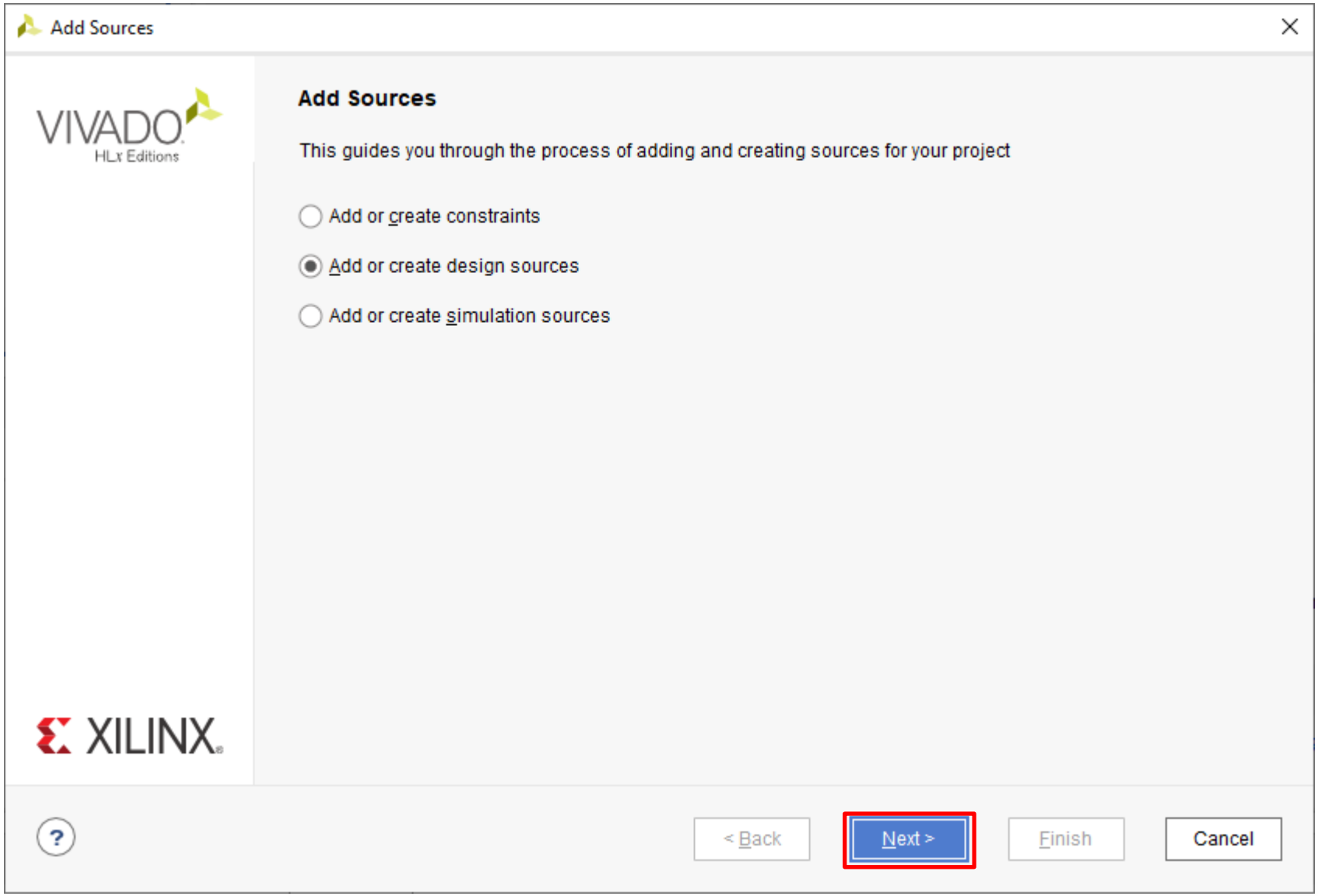
2



3

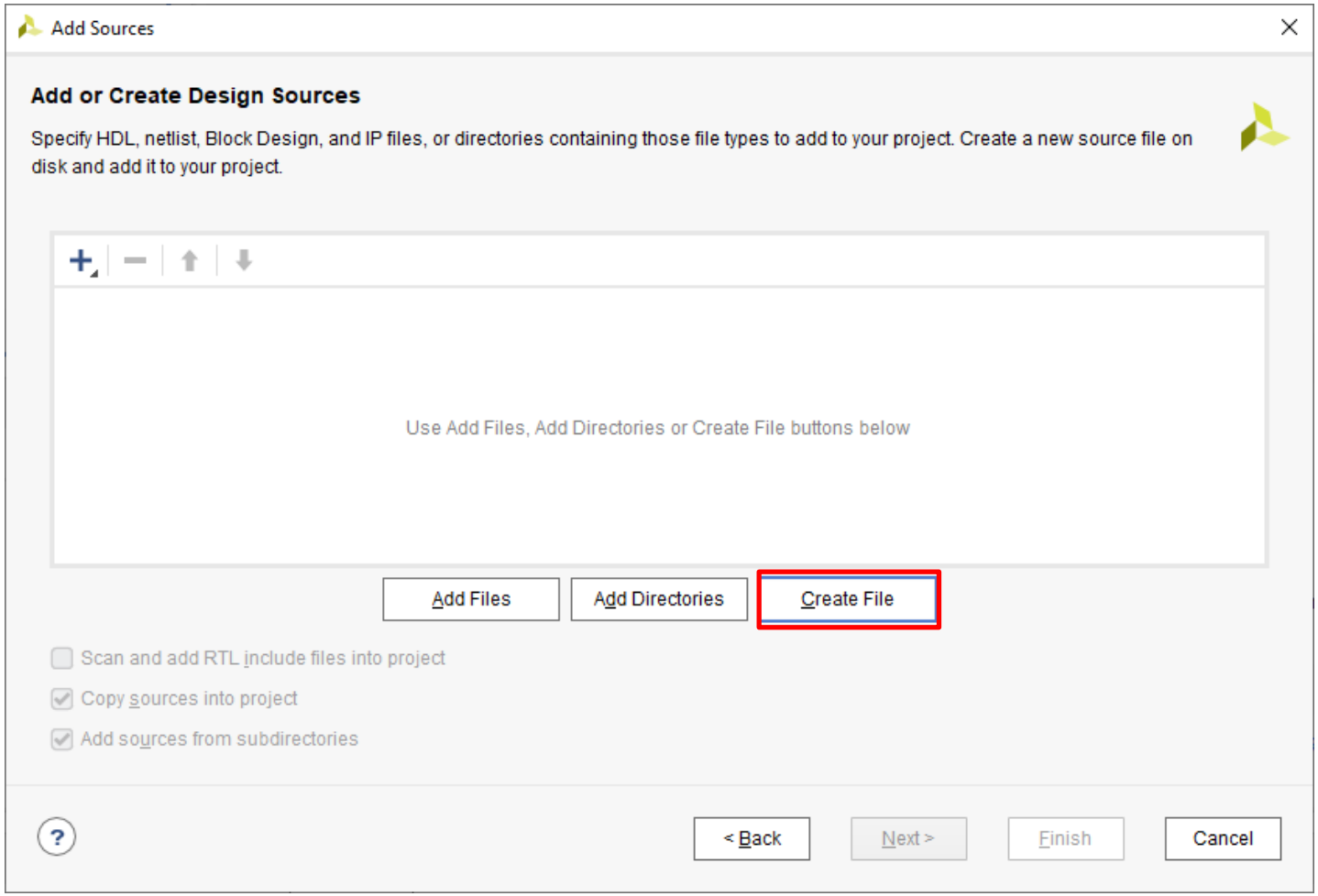


Vivado – Add sources



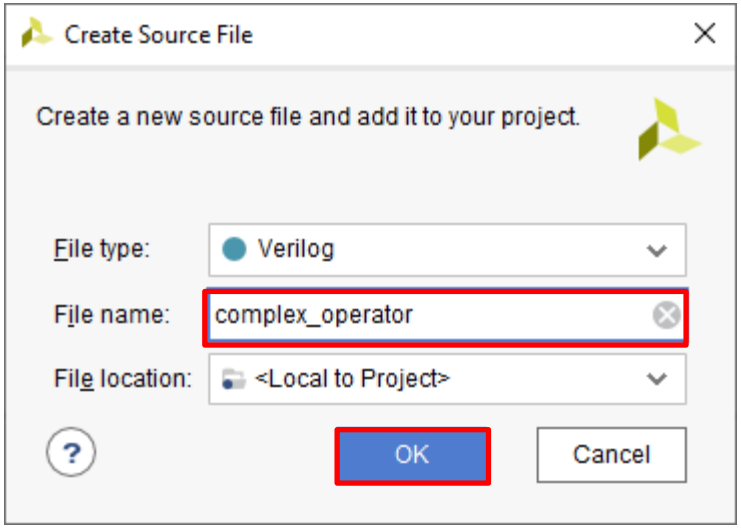
Vivado – Add sources

- Click “Create File”



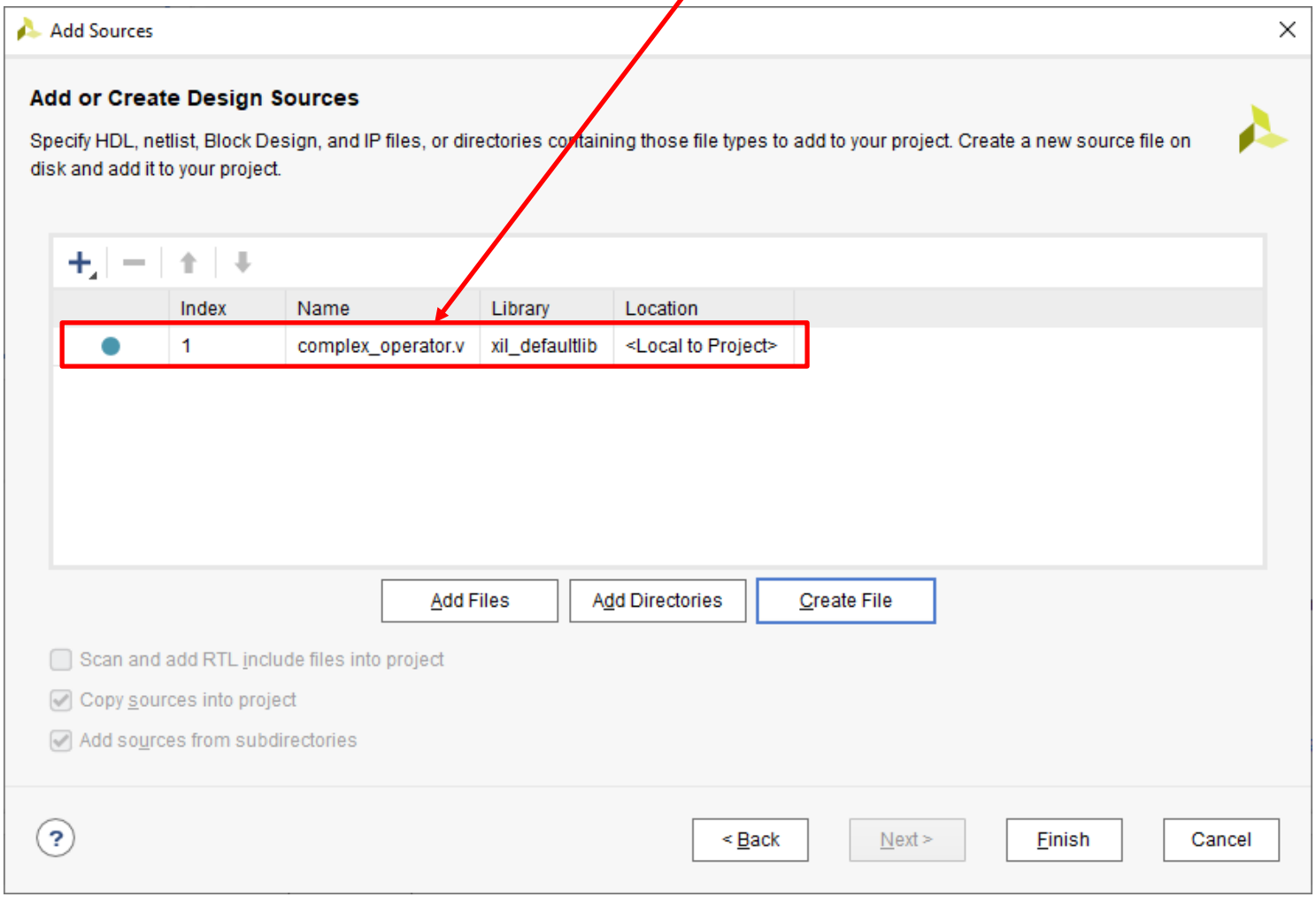
Vivado – Add sources

- Choose file type and write file name.



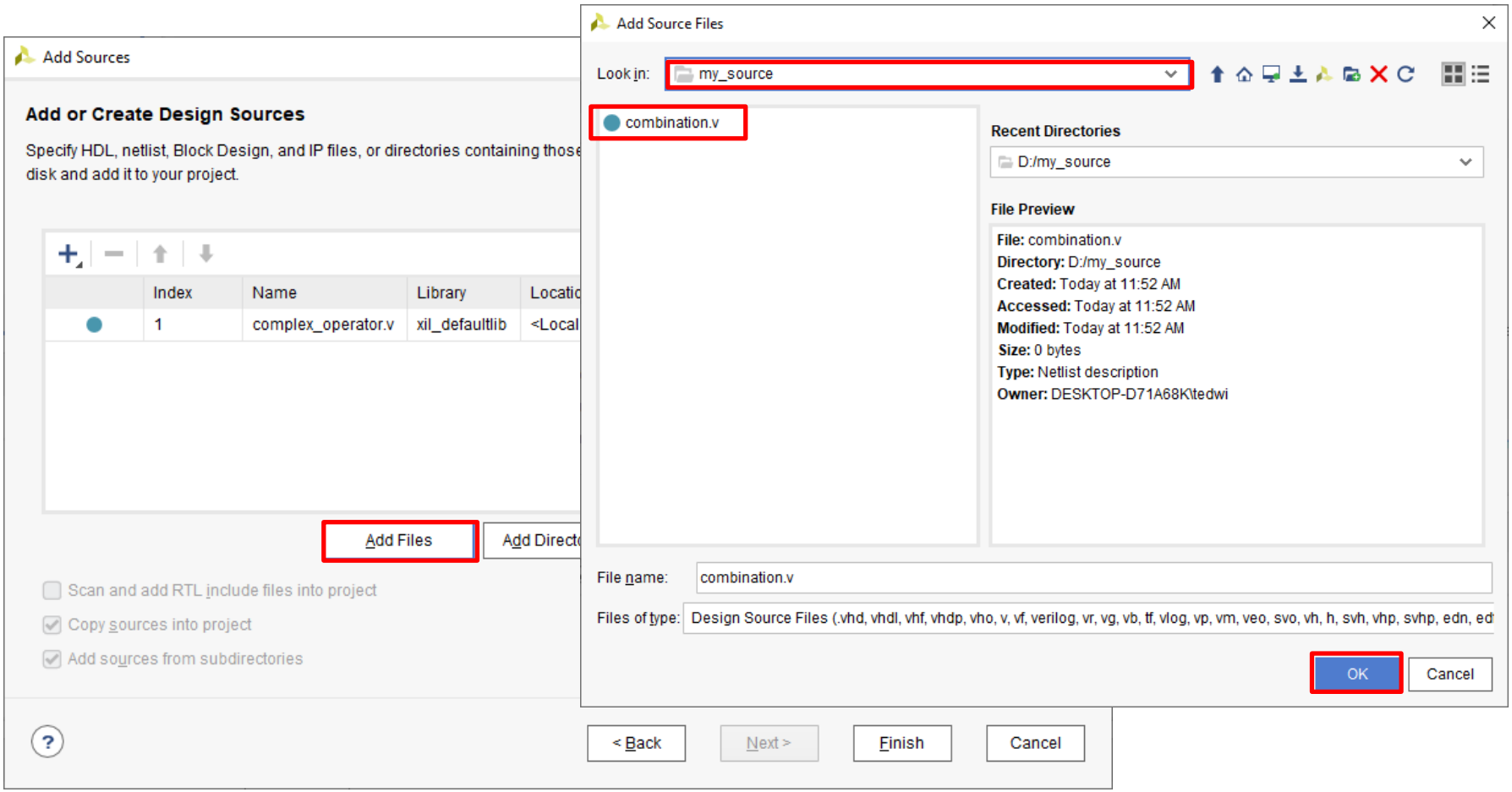
Vivado – Add sources

- Now you can see the new file is added.



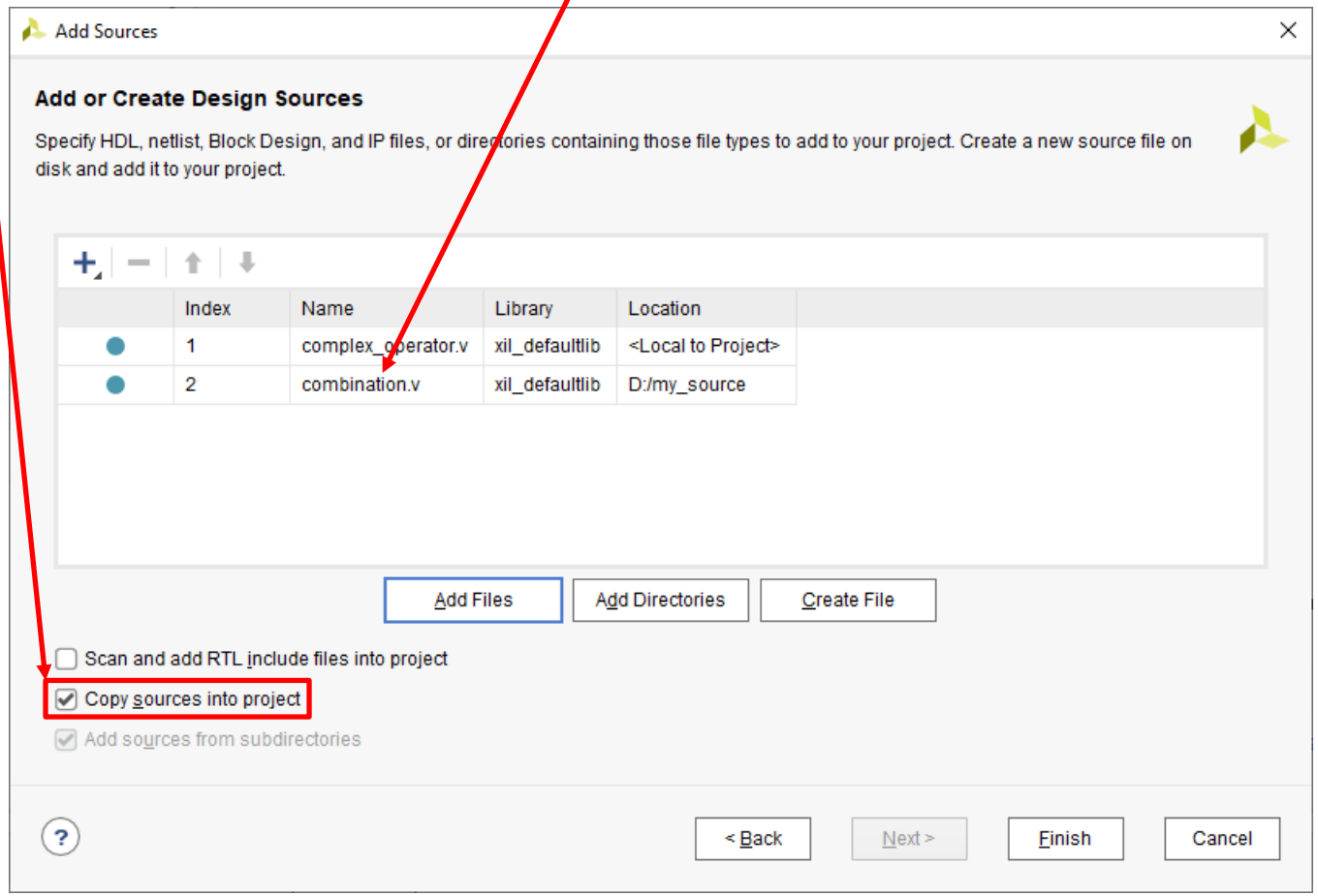
Vivado – Add sources

- You can also add existing sources.



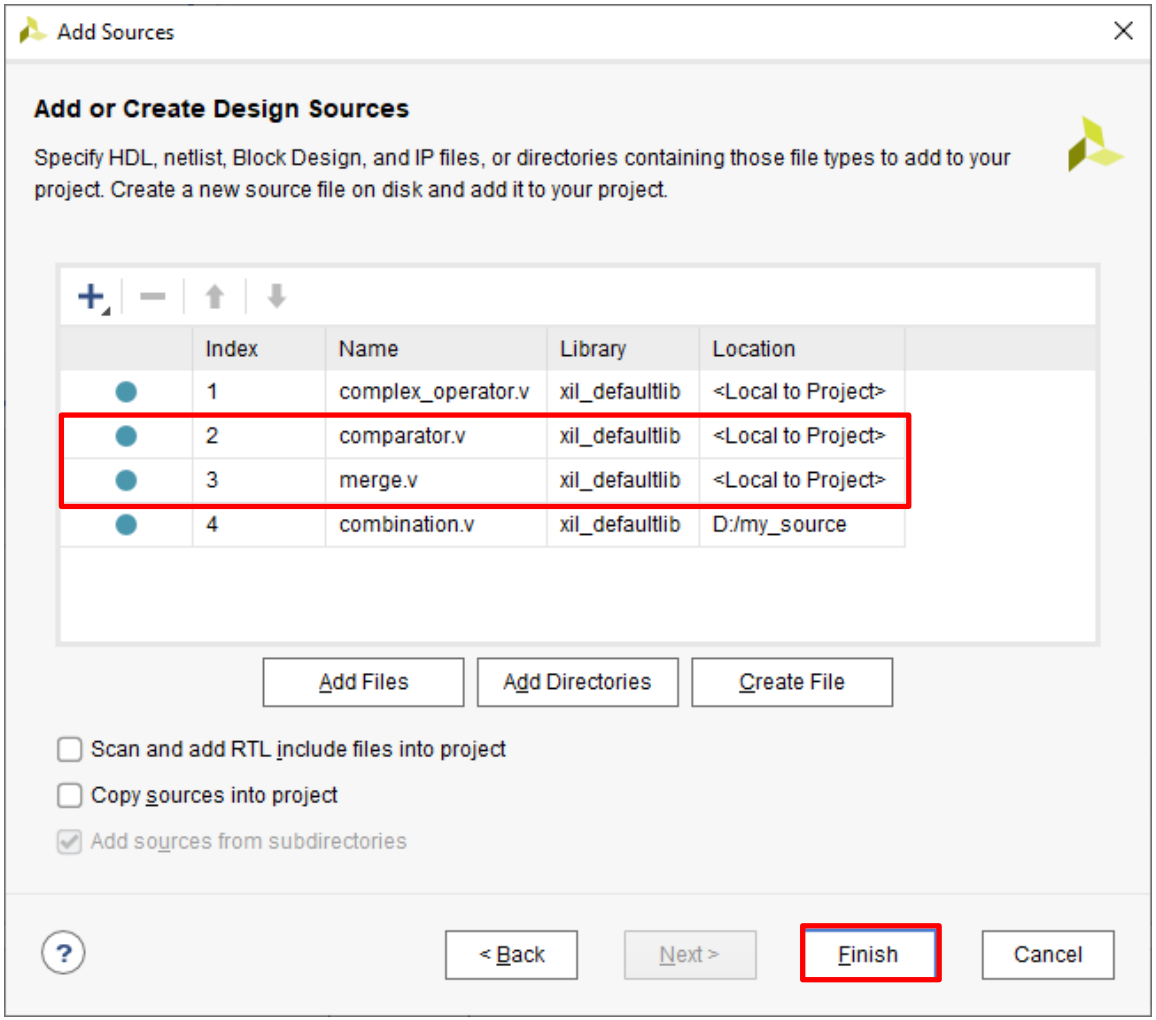
Vivado – Add sources

- Then, you can see your source file is added.
- If you don't want to change original file, then check "Copy sources into project" option.

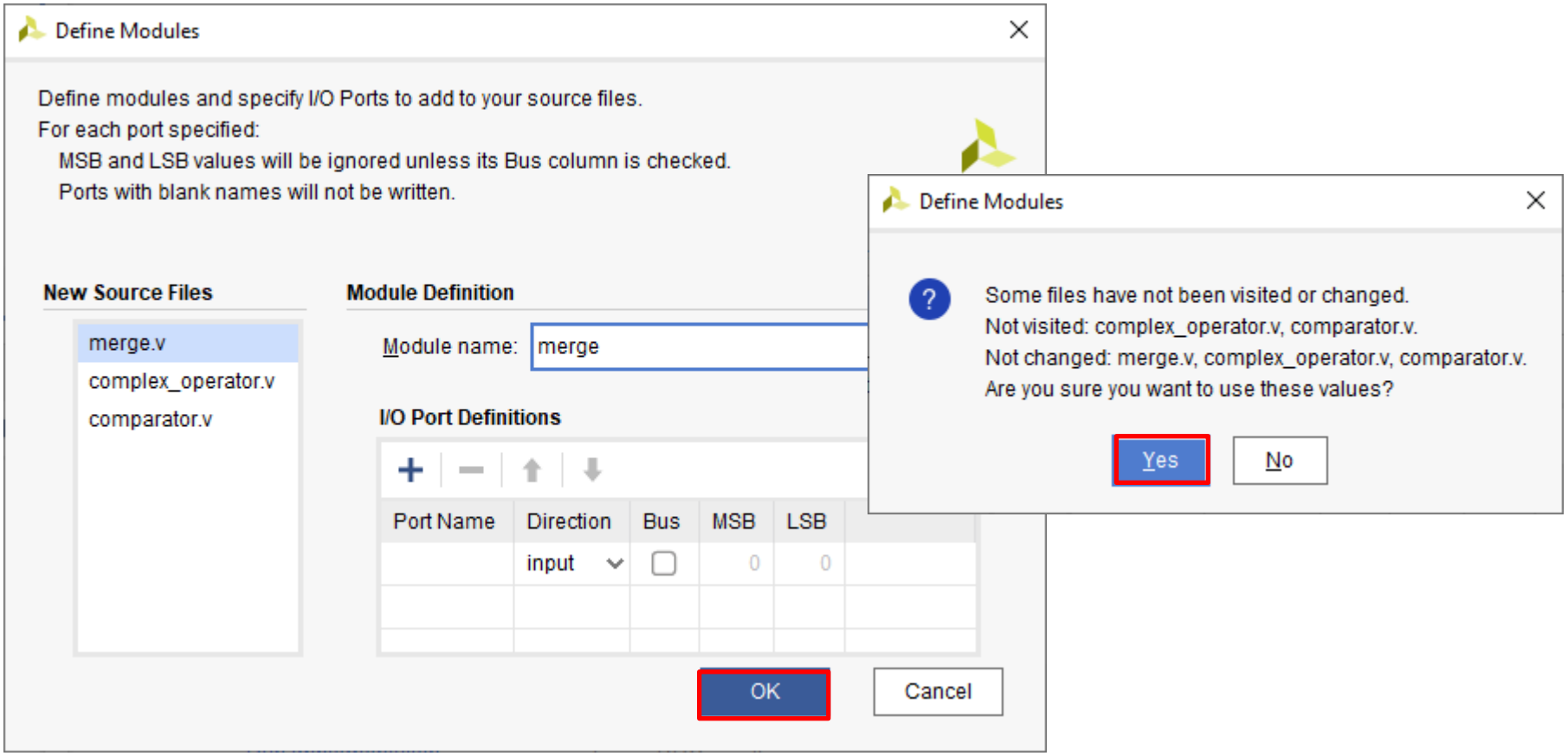


Vivado – Add sources

- Create ‘comparator.v’ and ‘merge.v’ file as you did before.

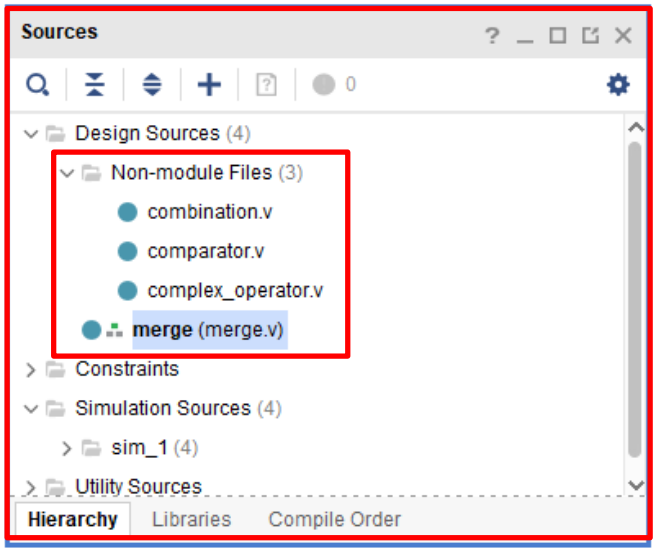


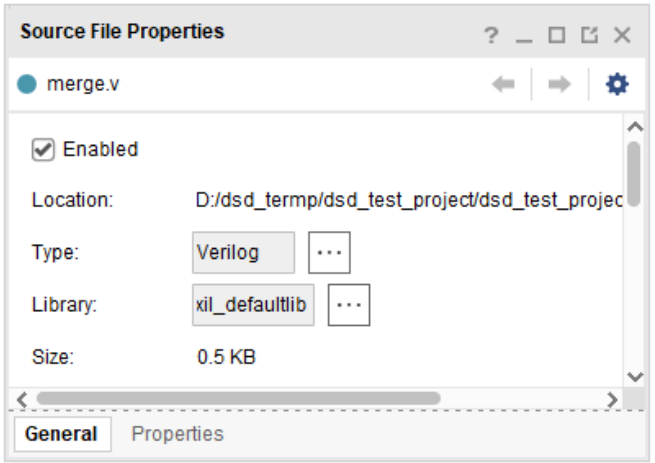
Vivado – Add sources

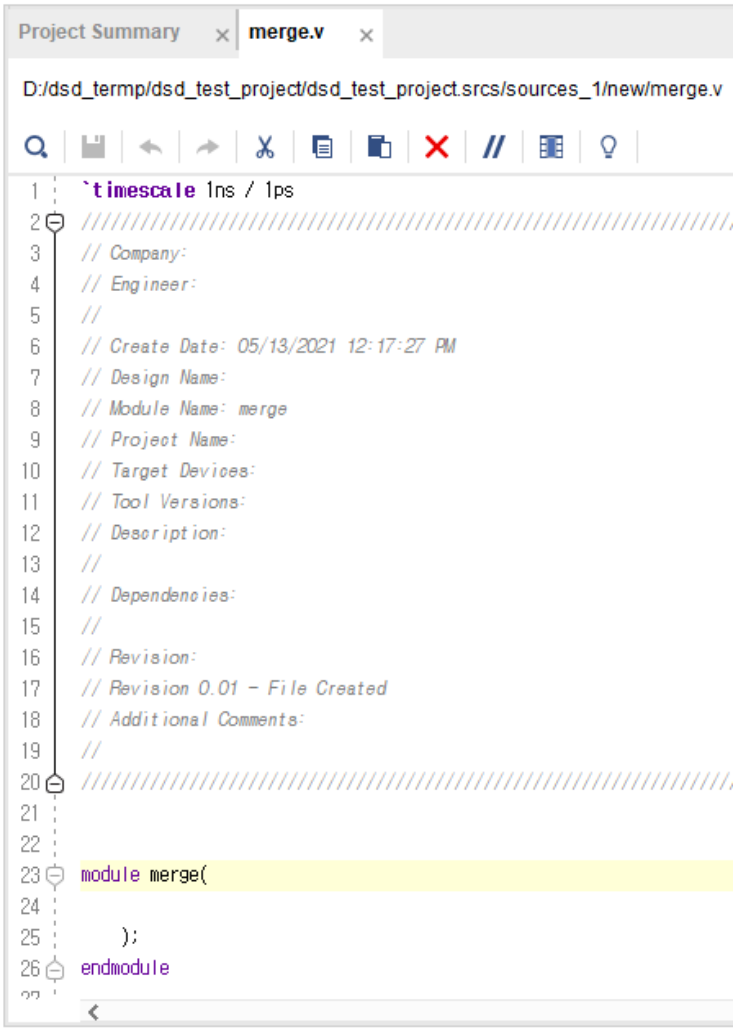


Vivado – Add sources

- You can see design sources through source tap
- You can see and edit source files by opening file(double click or right click -> ‘open file’).

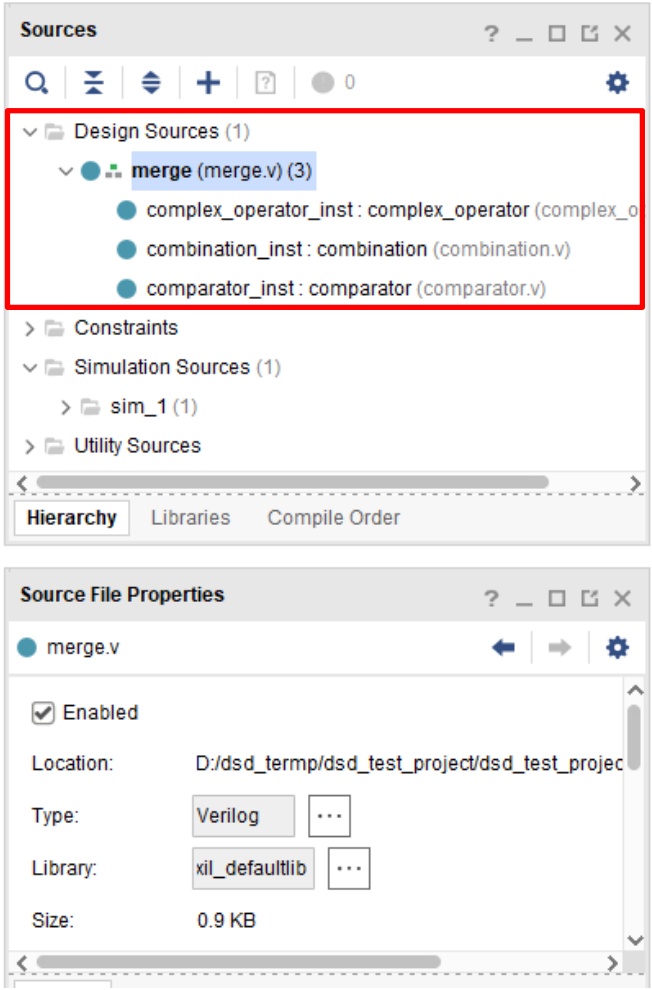


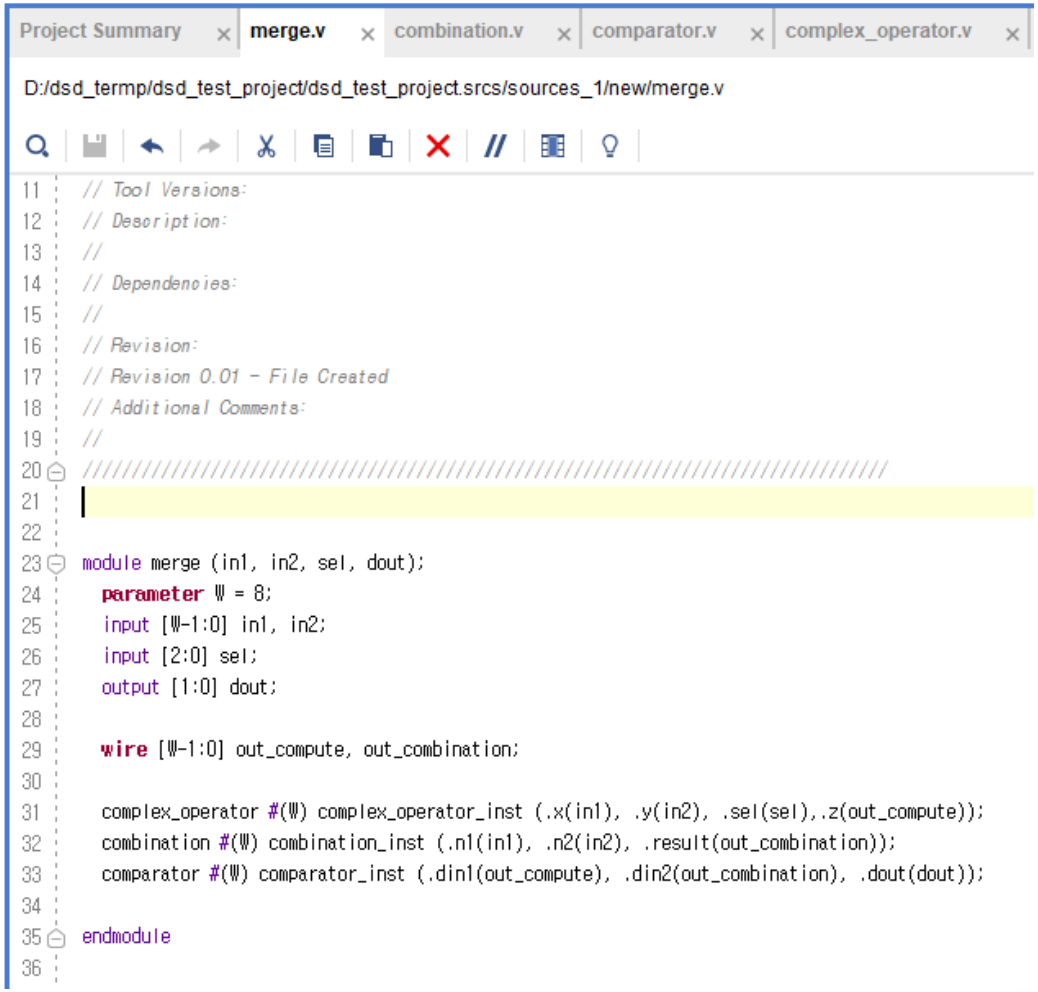




Vivado – Add sources

- Sources are organized hierarchically

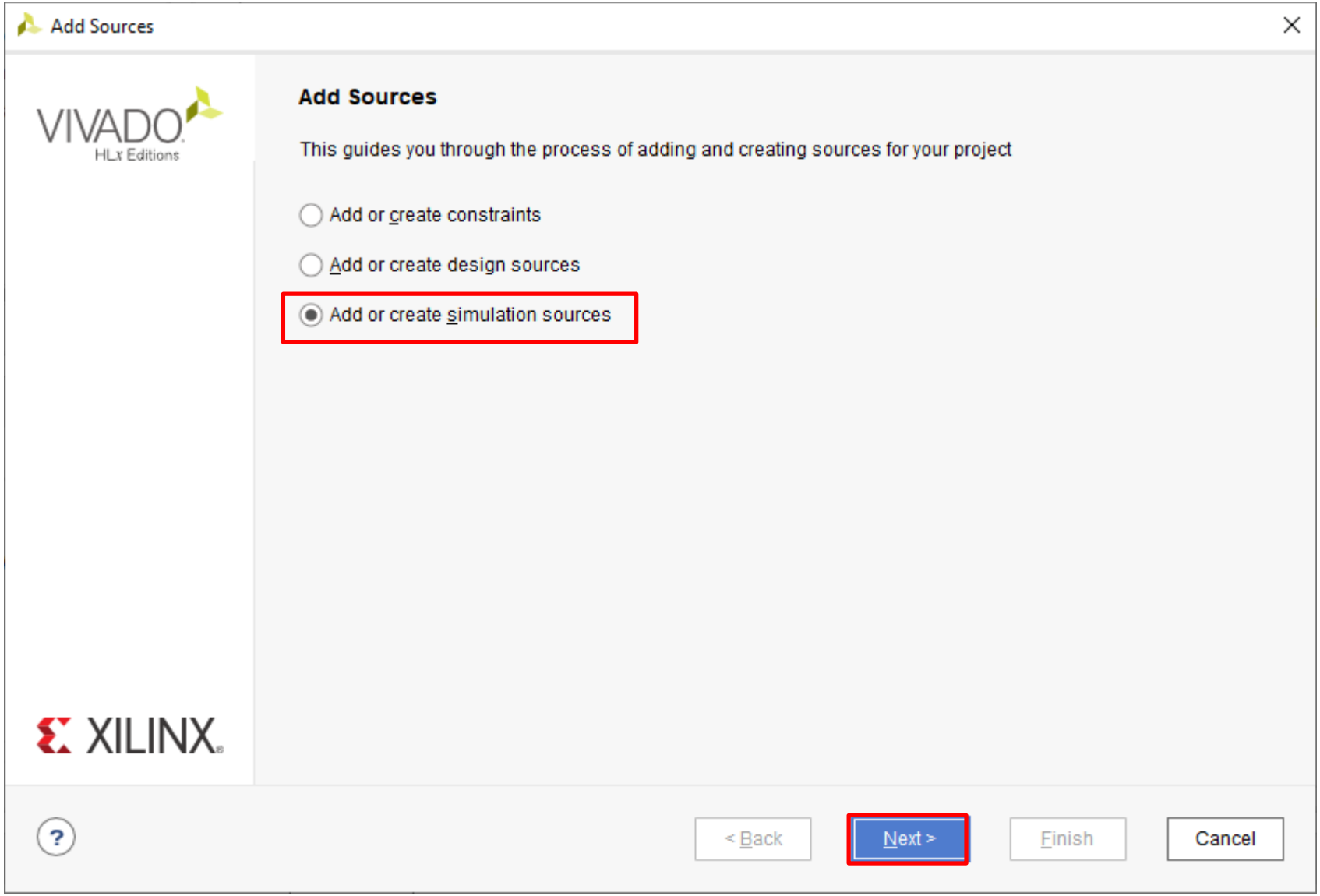




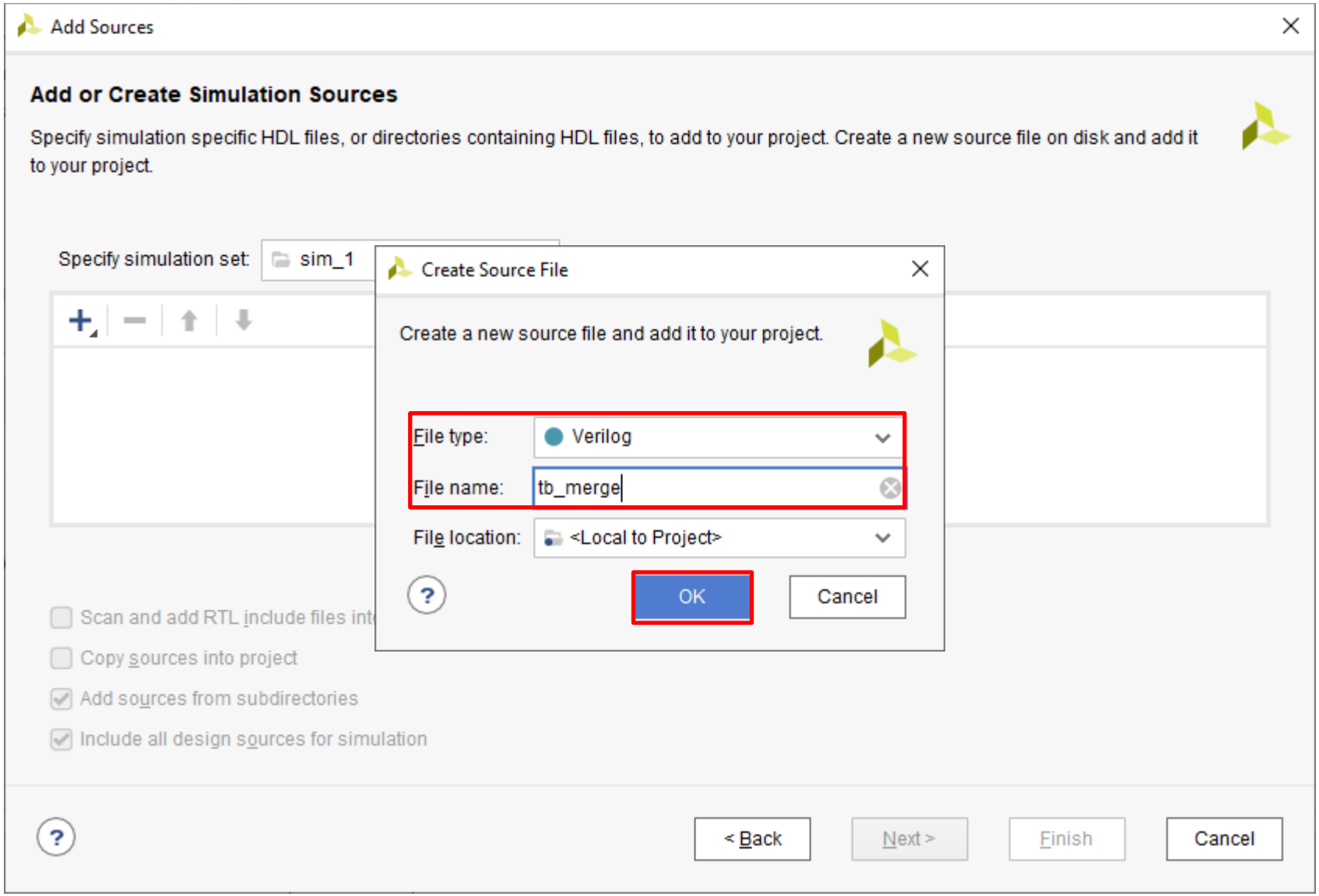
Vivado - Simulation

- About FPGA
- Design Flow
- Project explanation
- Vivado – Download
- Vivado – Create project
- Vivado – Add/Create sources
- Vivado – Simulation
- Vivado – Example: Simple PL Design
- Project – Reference code review (Video)
- Project – Vivado given sample code review (Video)
- Project – Vivado testbench examples (Video)

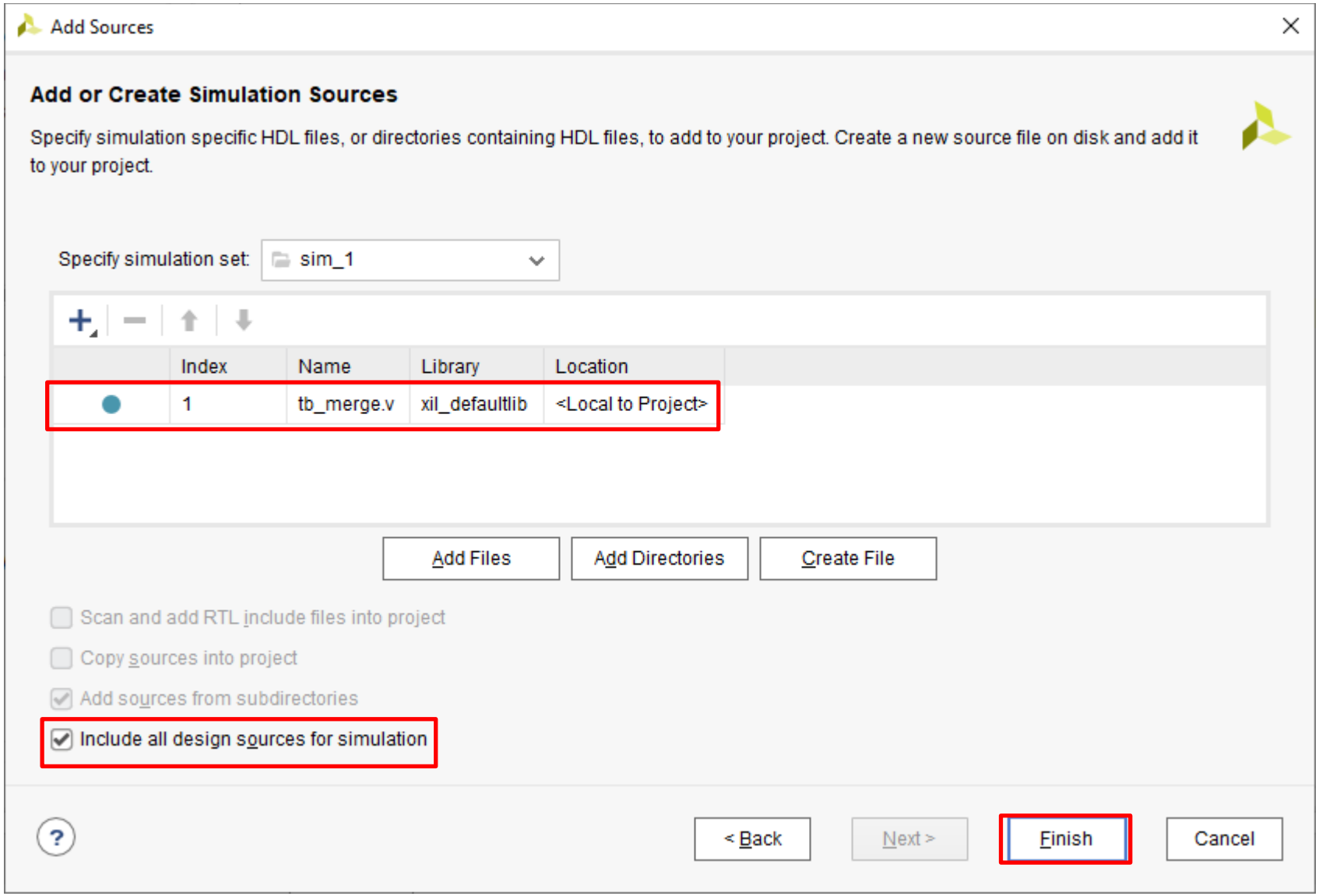
Vivado – Simulation



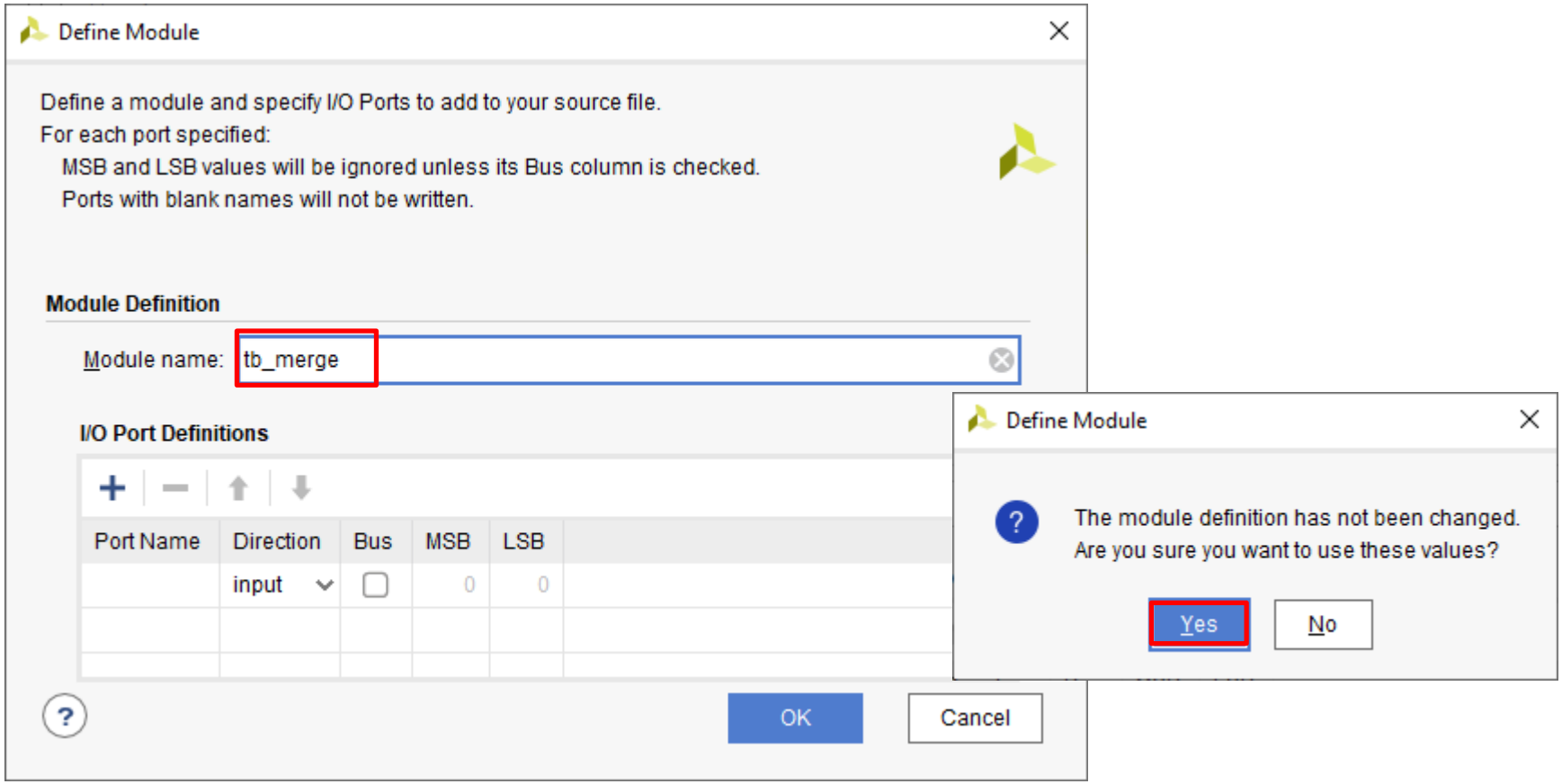
Vivado – Simulation



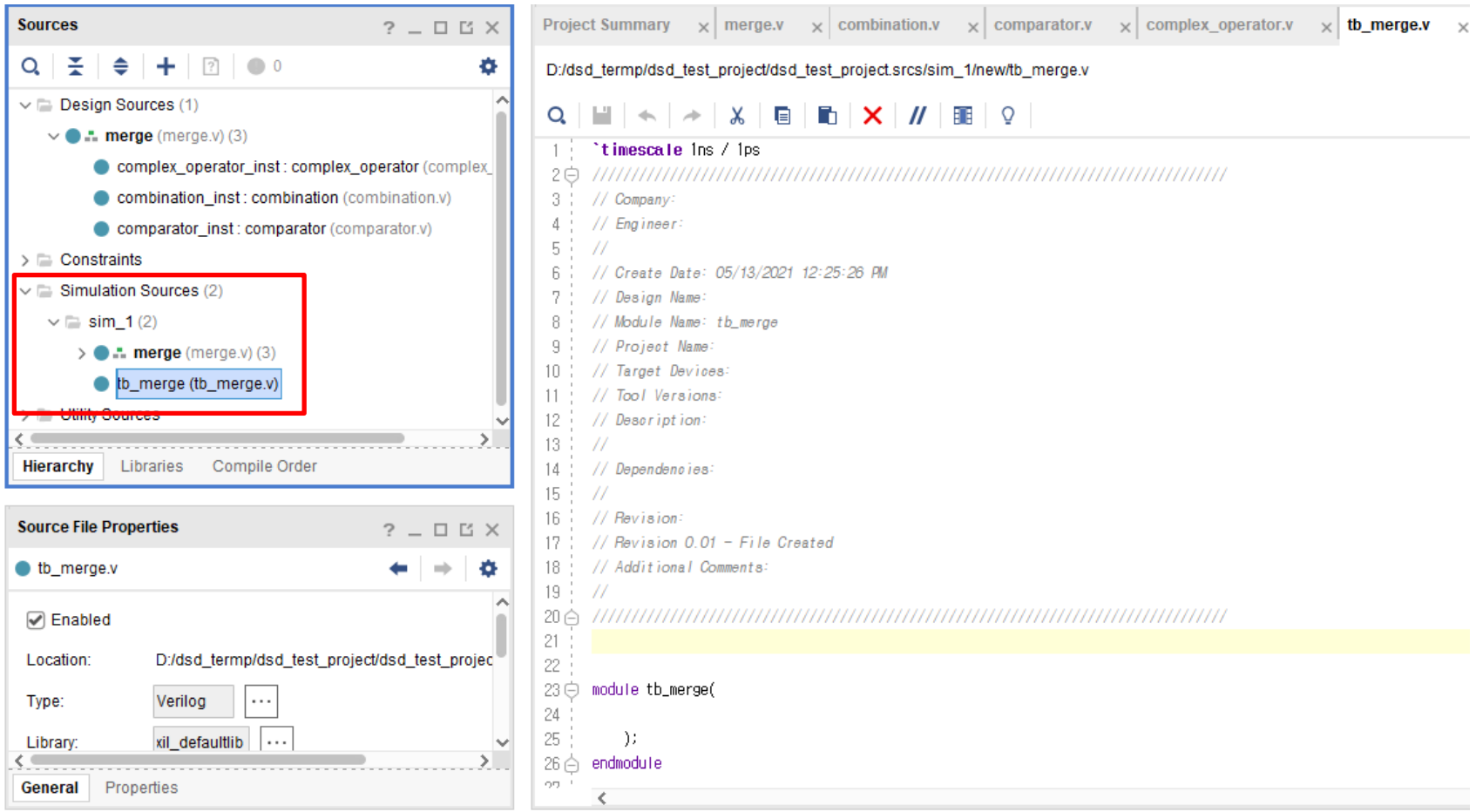
Vivado – Simulation



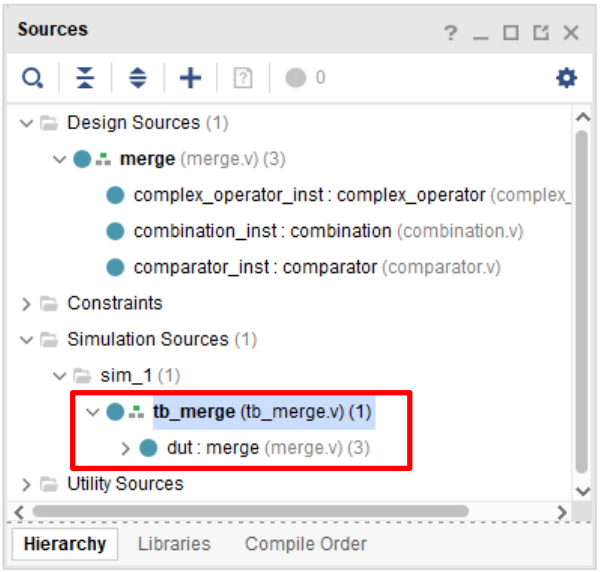
Vivado – Simulation

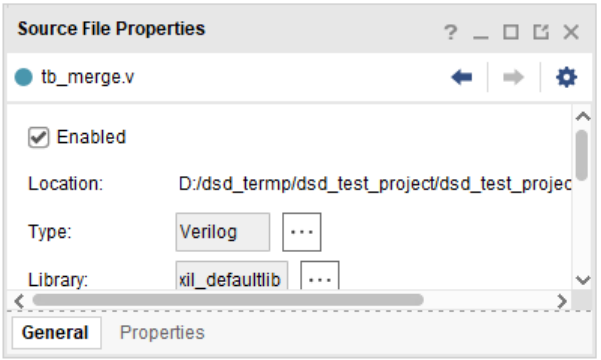


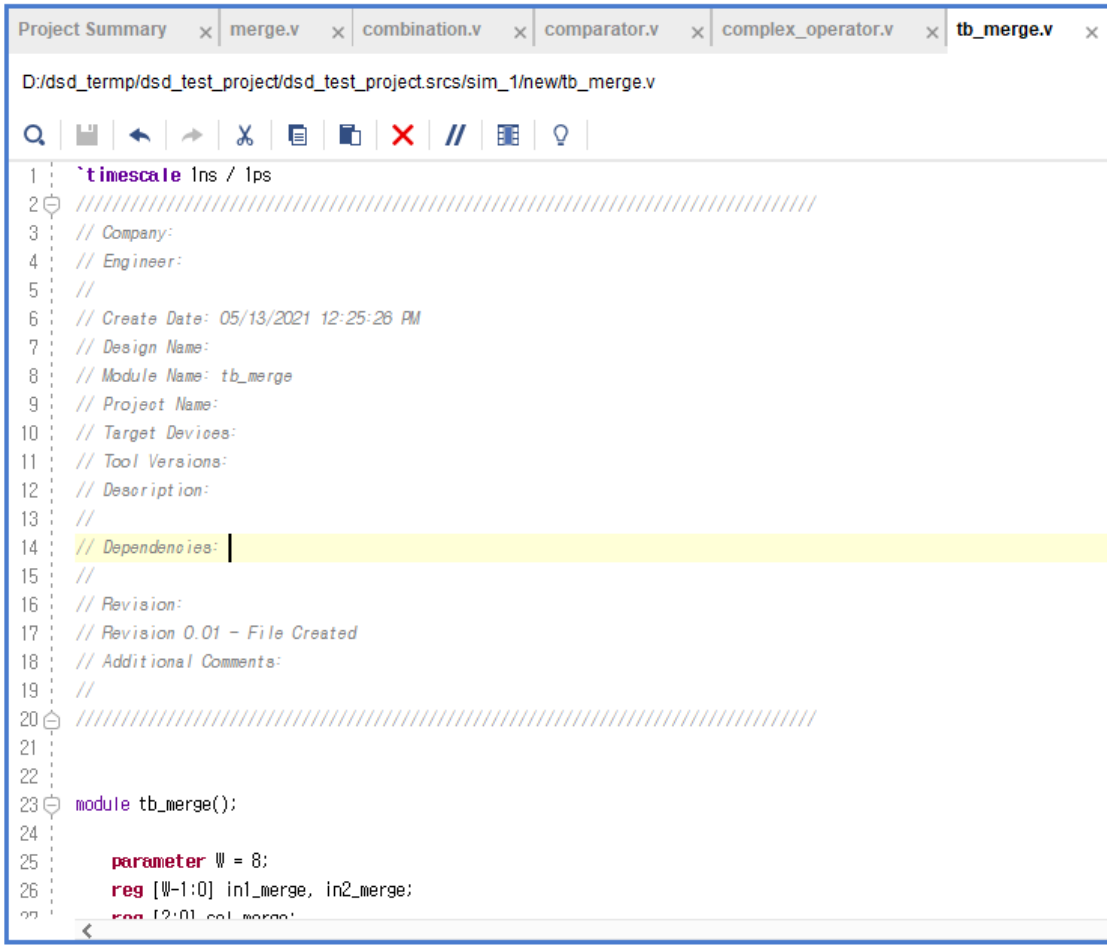
Vivado – Simulation



Vivado – Simulation

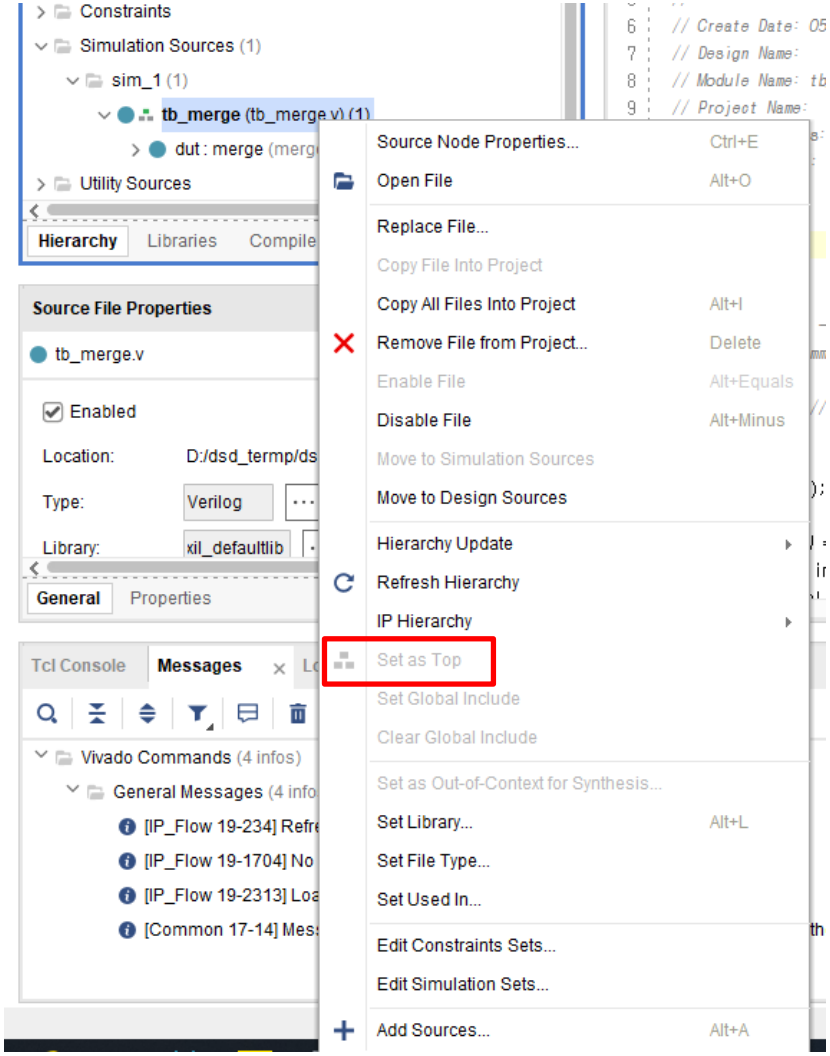




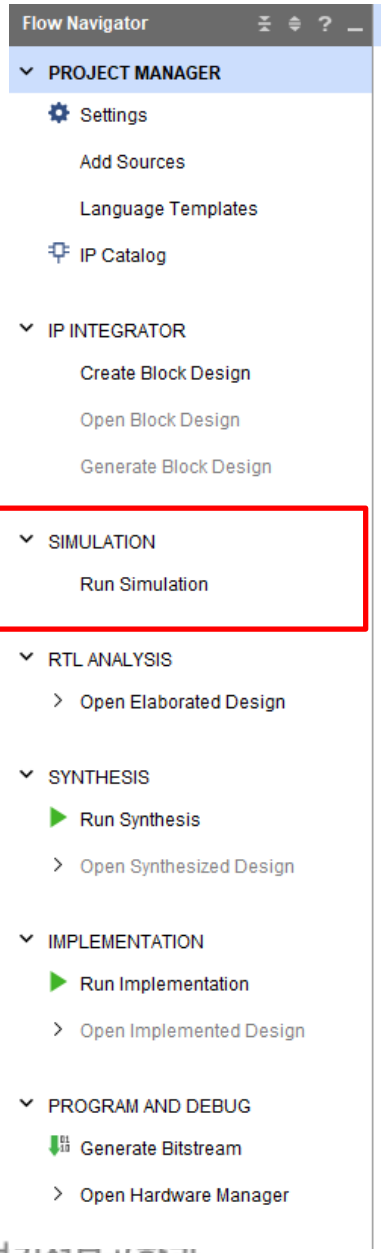


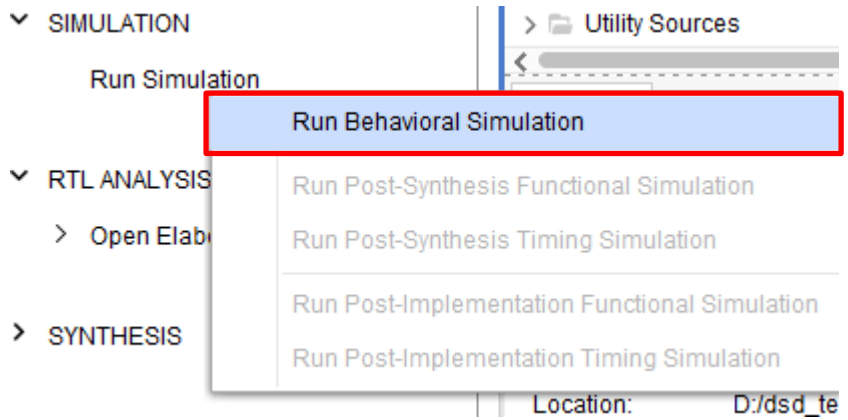
Vivado – Simulation

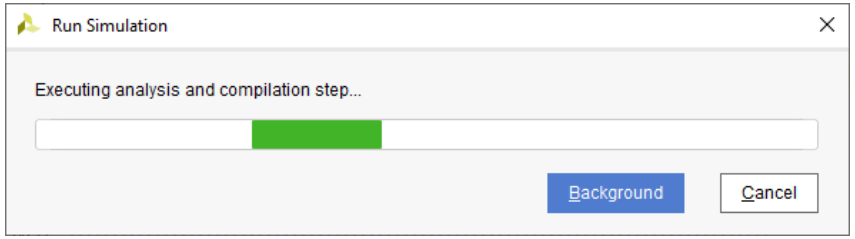
- If there are multiple testbench sources, then you can select a source that will be simulated by setting it as top file.



Vivado – Simulation

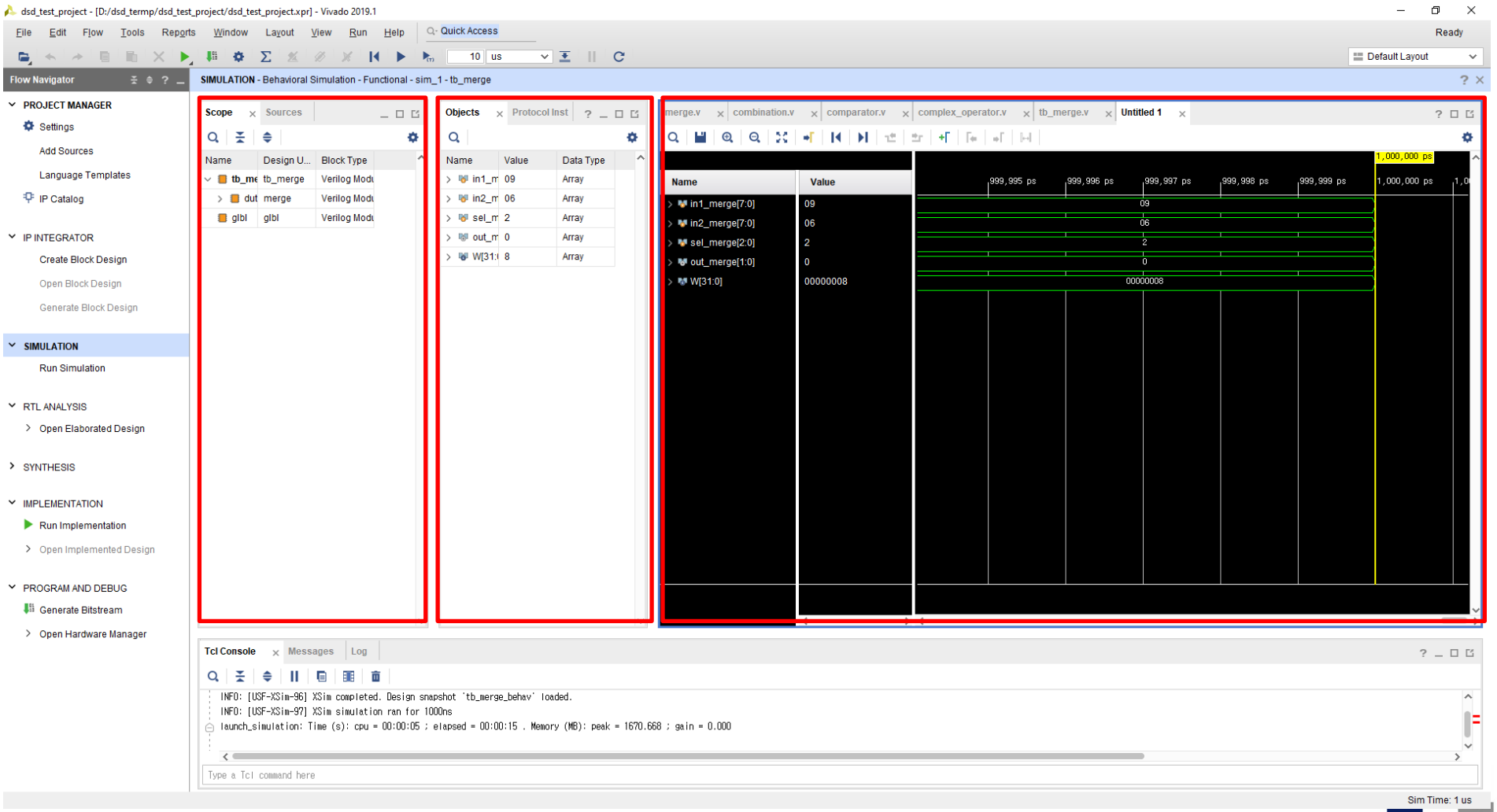






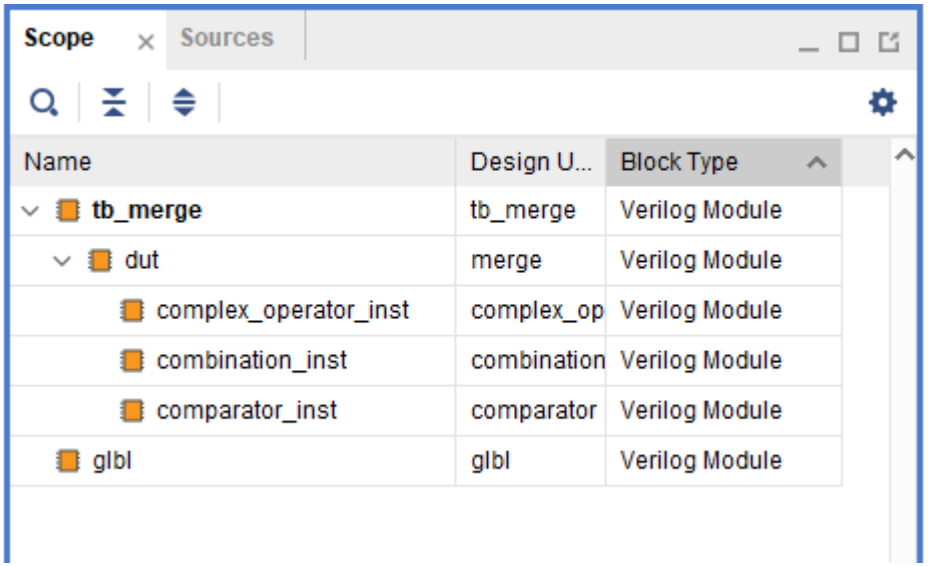
wait a minute...

Vivado – Simulation



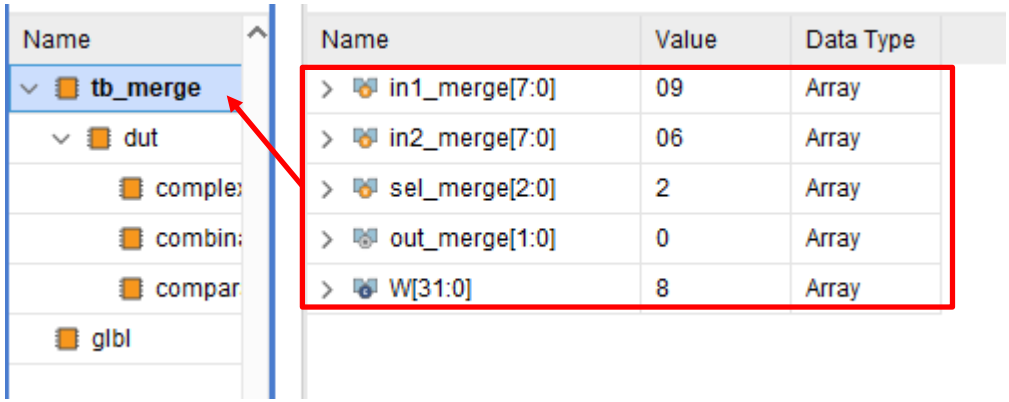
Vivado – Simulation

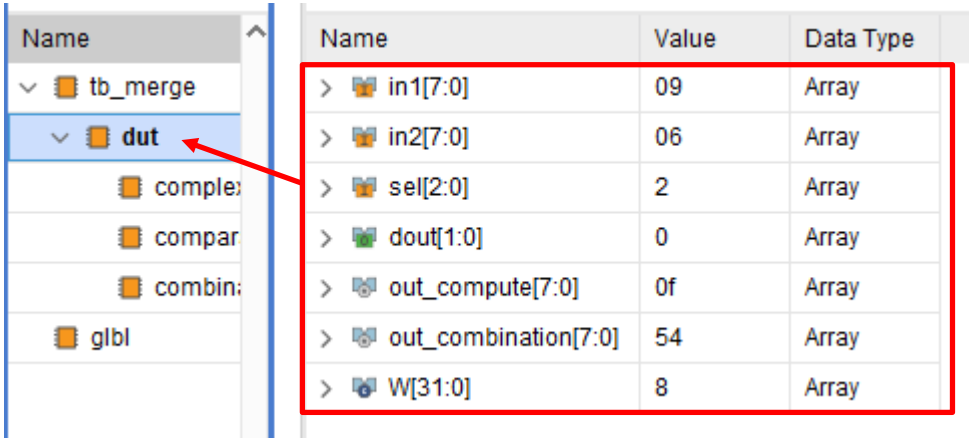
- You can see the simulated module instances.



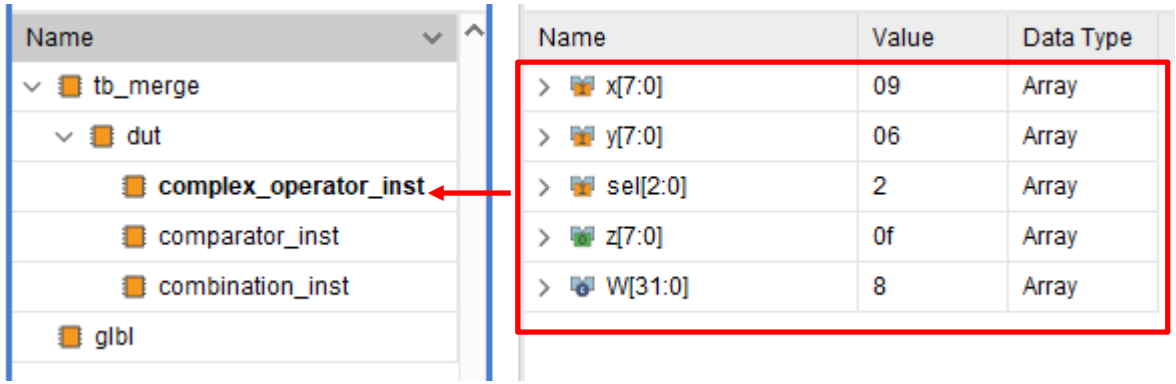
Vivado – Simulation

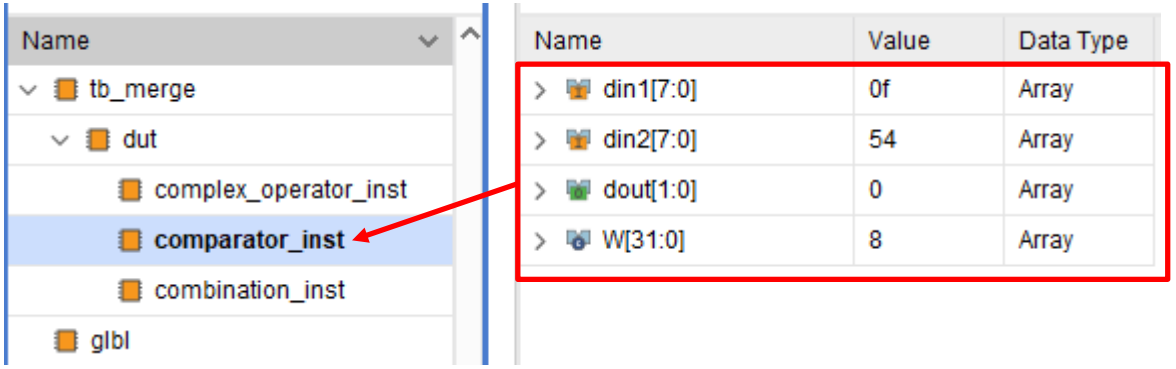
- You can see all the variables of selected instance through ‘Objects’ window.

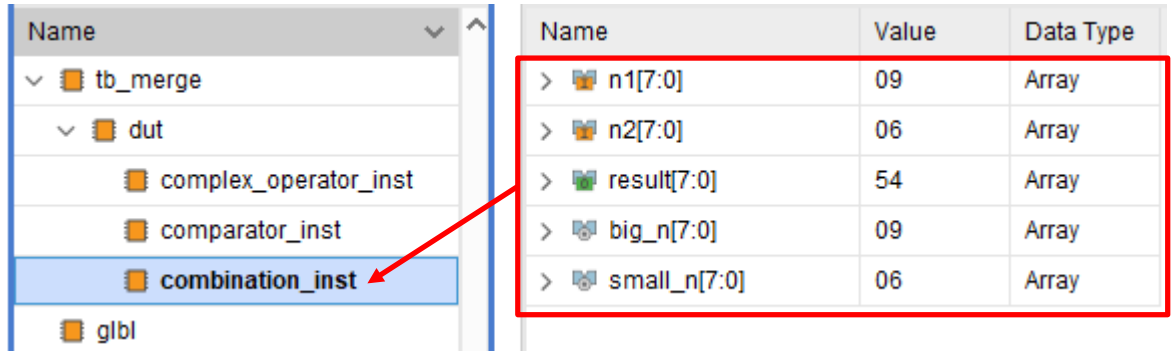




Vivado – Simulation

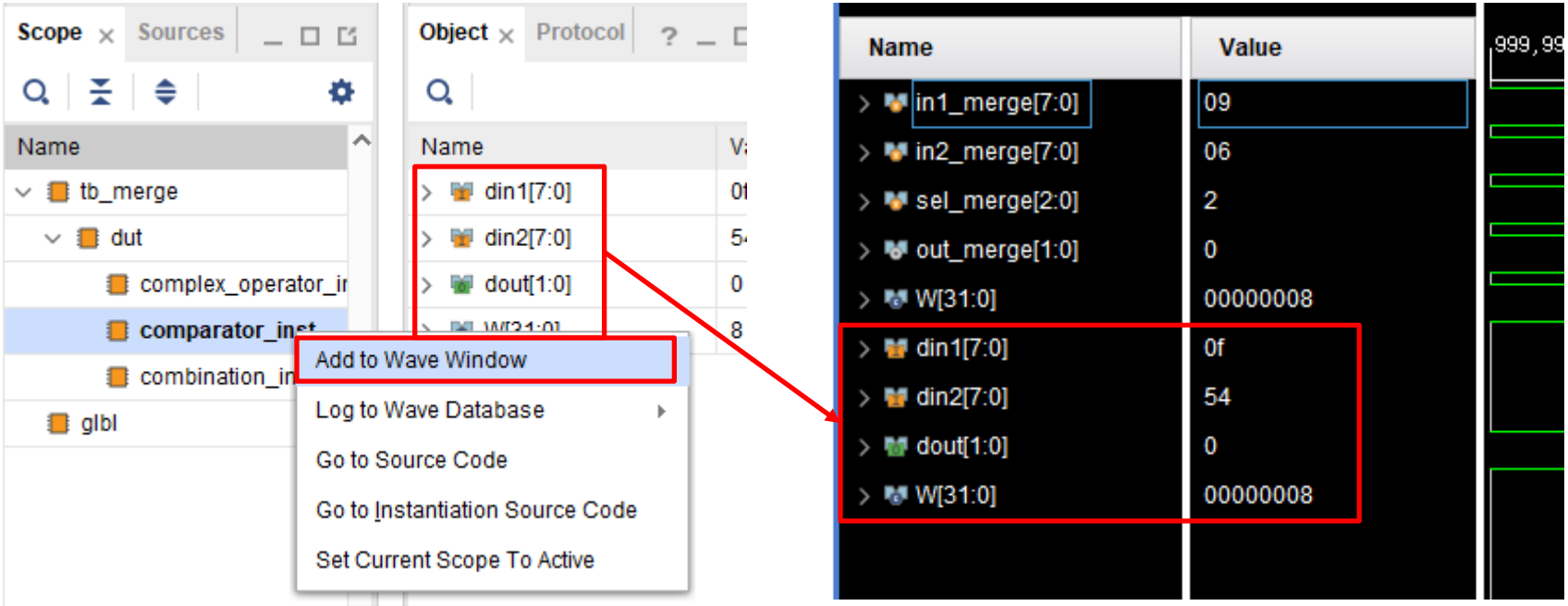






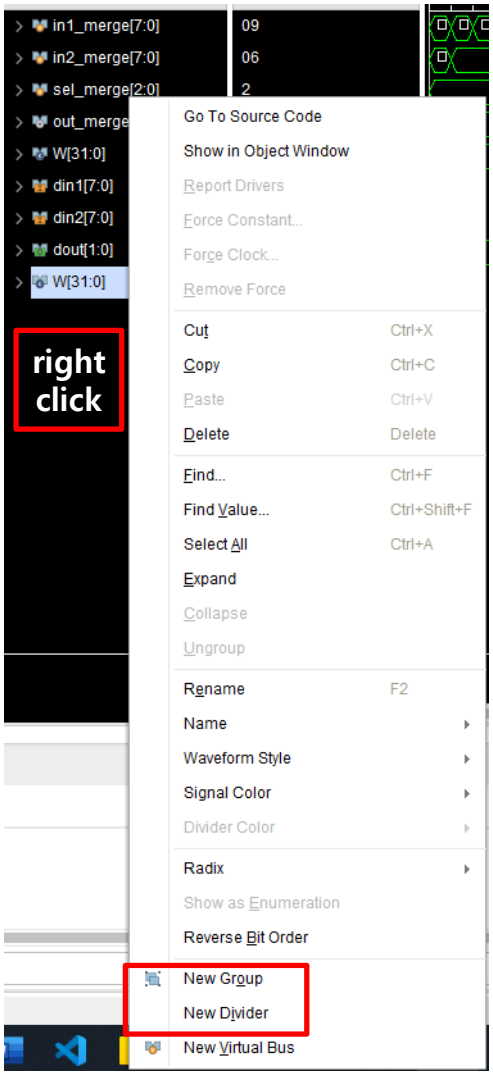
Vivado – Simulation

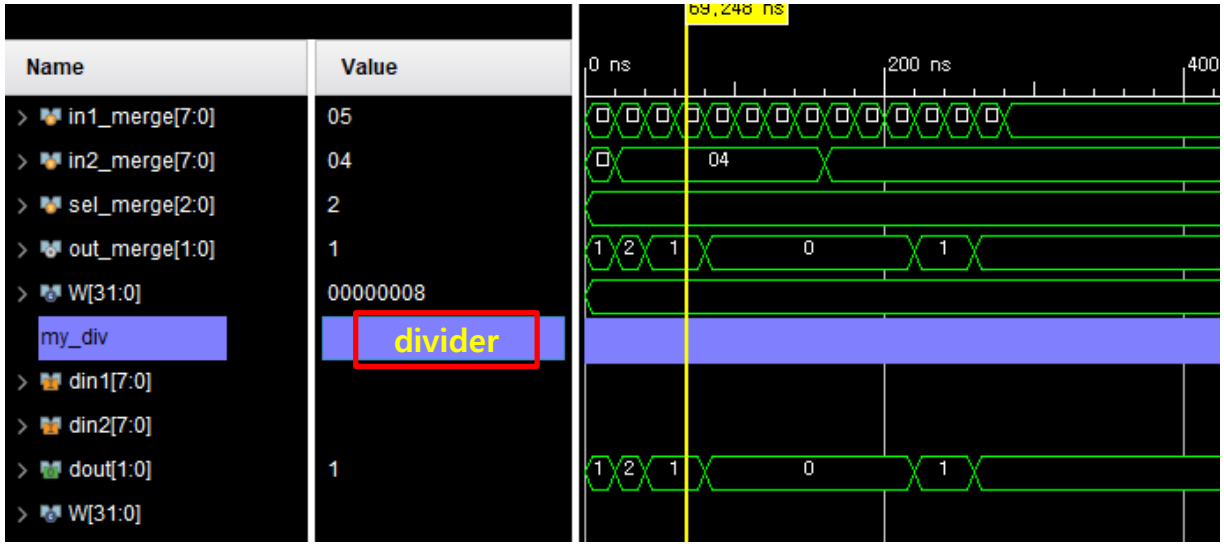
- You can add specific signals to wave window.



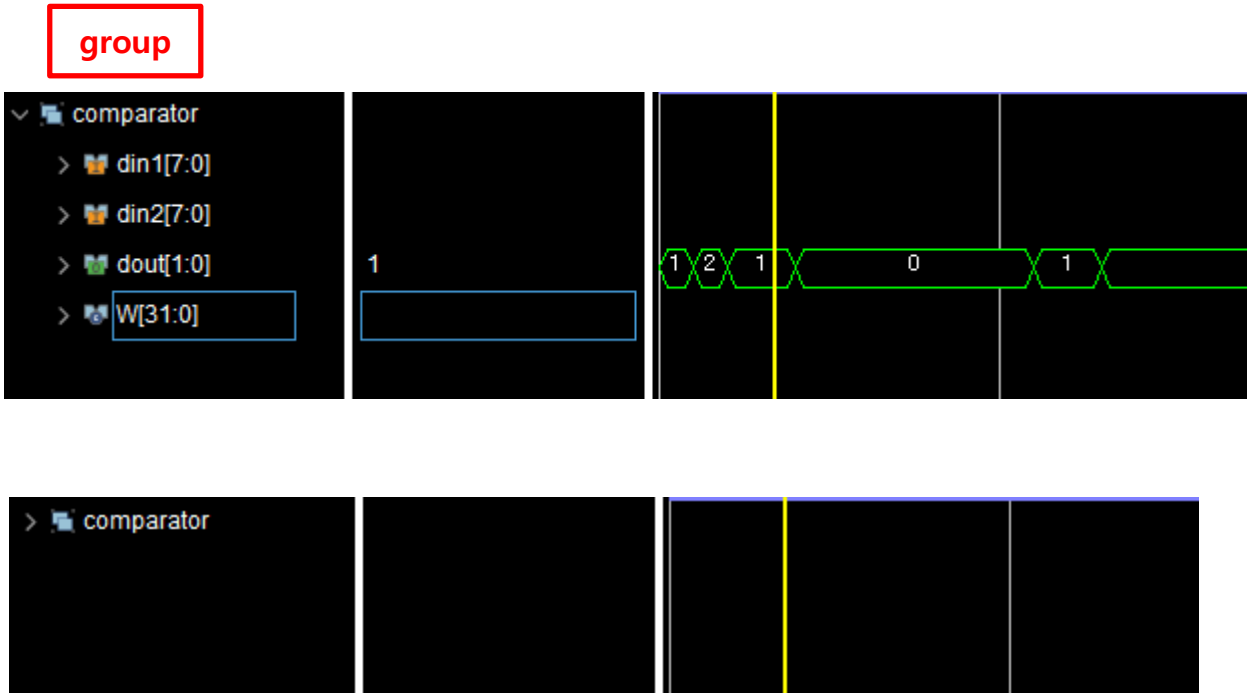
Vivado – Simulation

- You can add divider and group to the wave window.



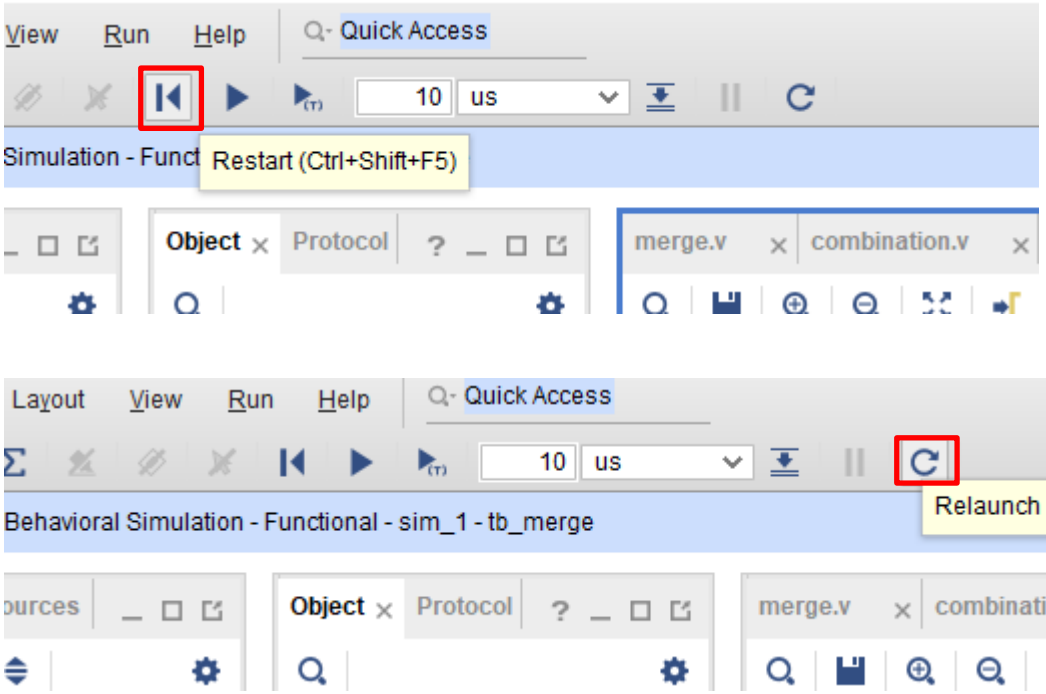


Vivado – Simulation



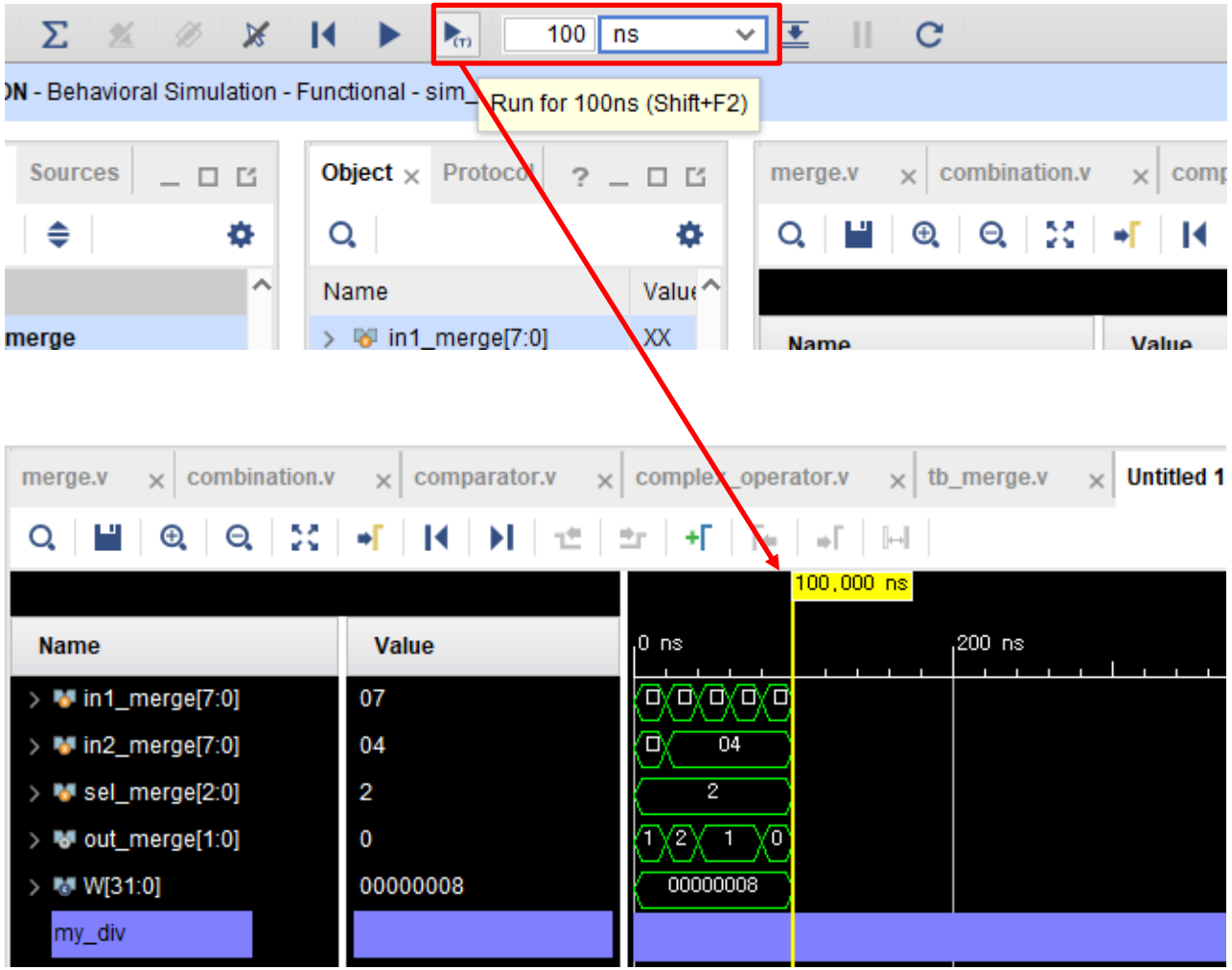
Vivado - Simulation

- Restart: restart simulation with the condition of last launched setting(changes in source files after launching simulation are not reflected)
- Relaunch Simulation: relaunch simulation(it takes more time than restarting simulation but changes in source files are reflected)

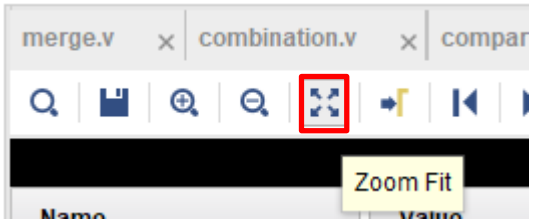


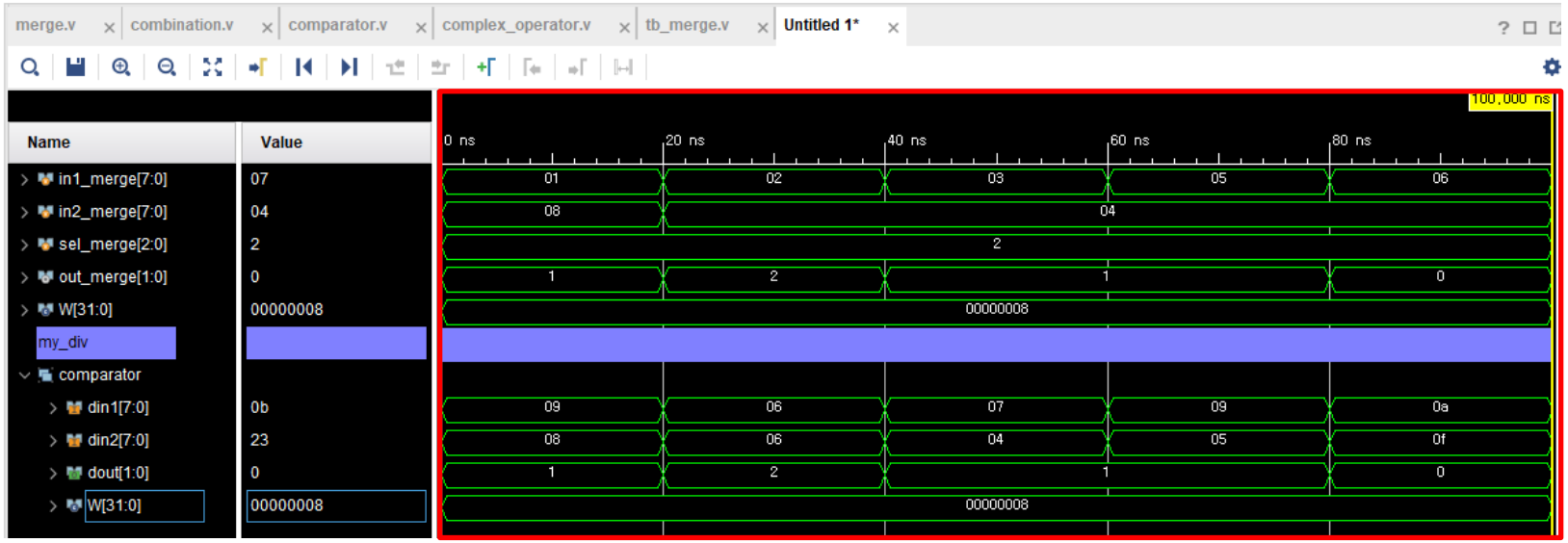
Vivado - Simulation

- df



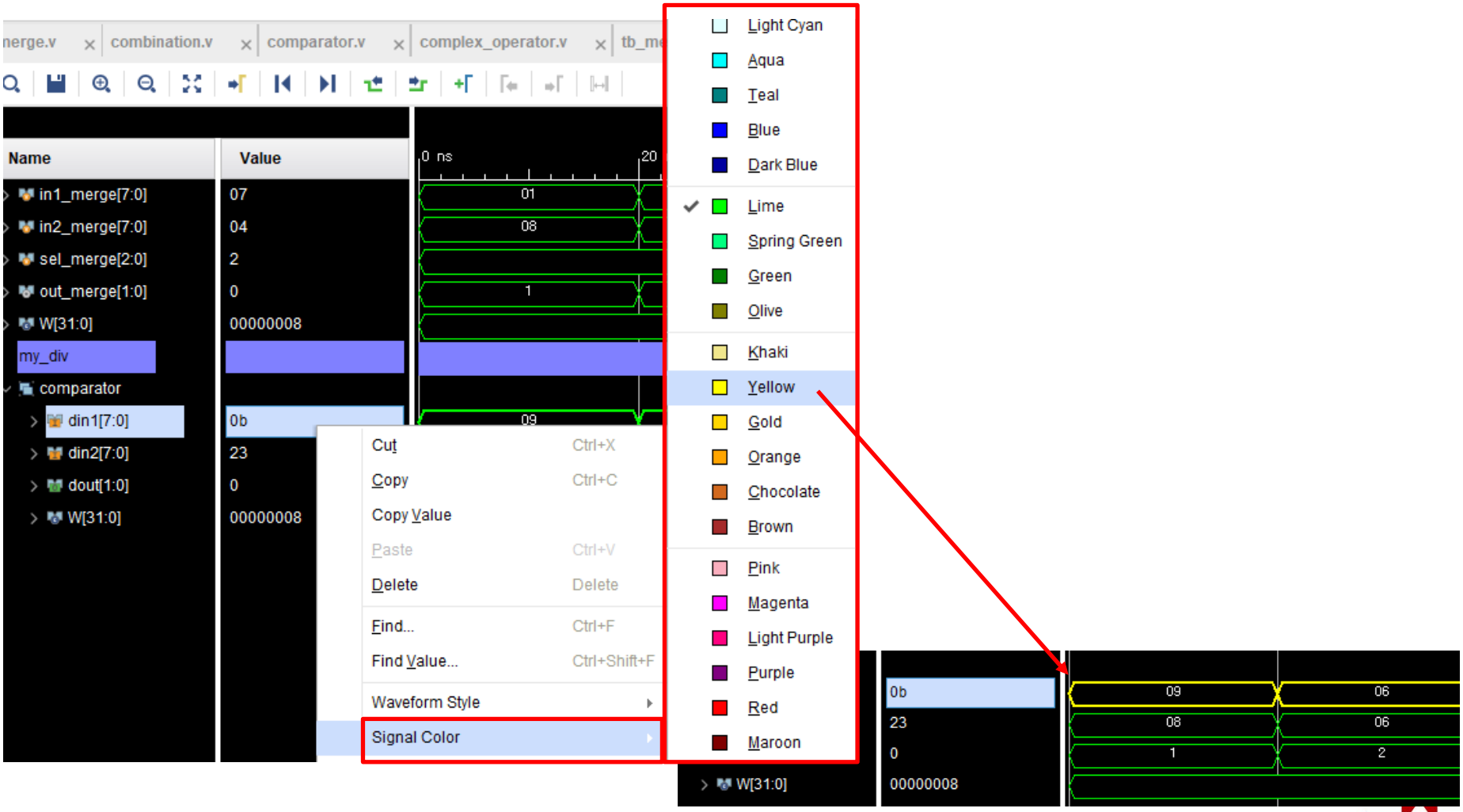
Vivado - Simulation





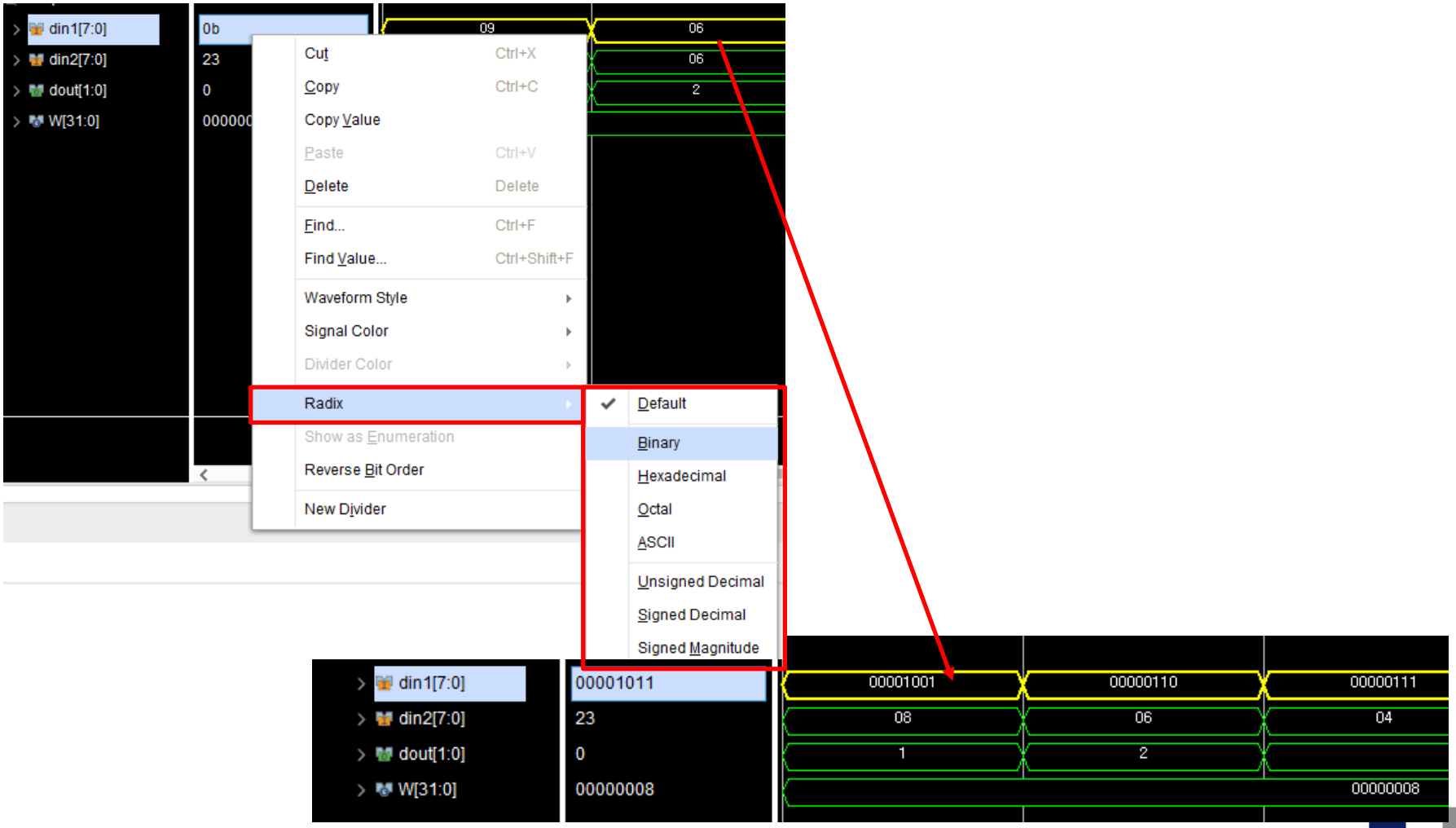
Vivado - Simulation

- You can change the color of signals



Vivado - Simulation

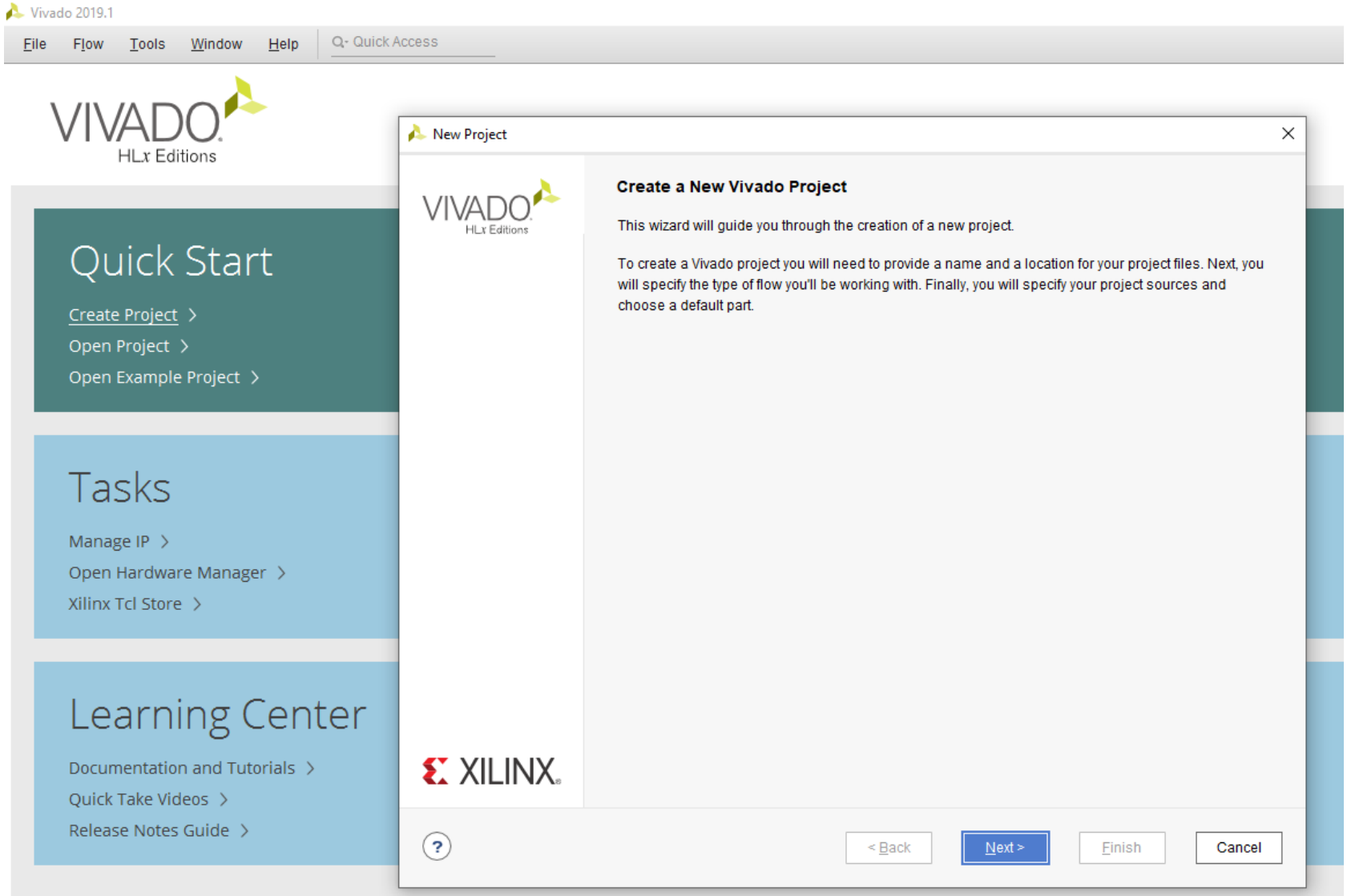
- You can also change the radix of signals.



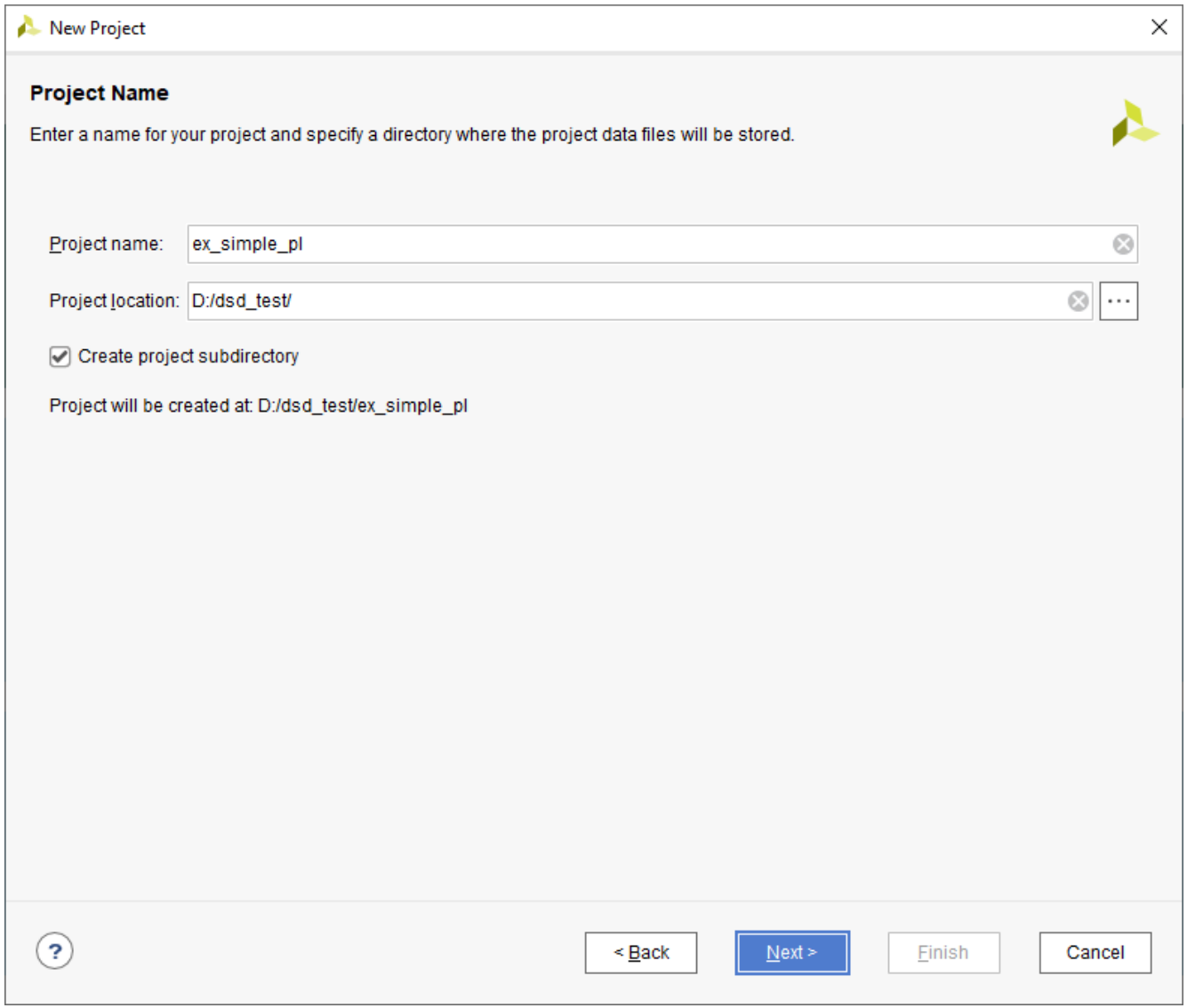
Vivado – Example: Simple PL Design

- About FPGA
- Design Flow
- Project explanation
- Vivado – Download
- Vivado – Create project
- Vivado – Add/Create sources
- Vivado – Simulation
- Vivado – Example: Simple PL Design
- Project – Reference code review (Video)
- Project – Vivado given sample code review (Video)
- Project – Vivado testbench examples (Video)

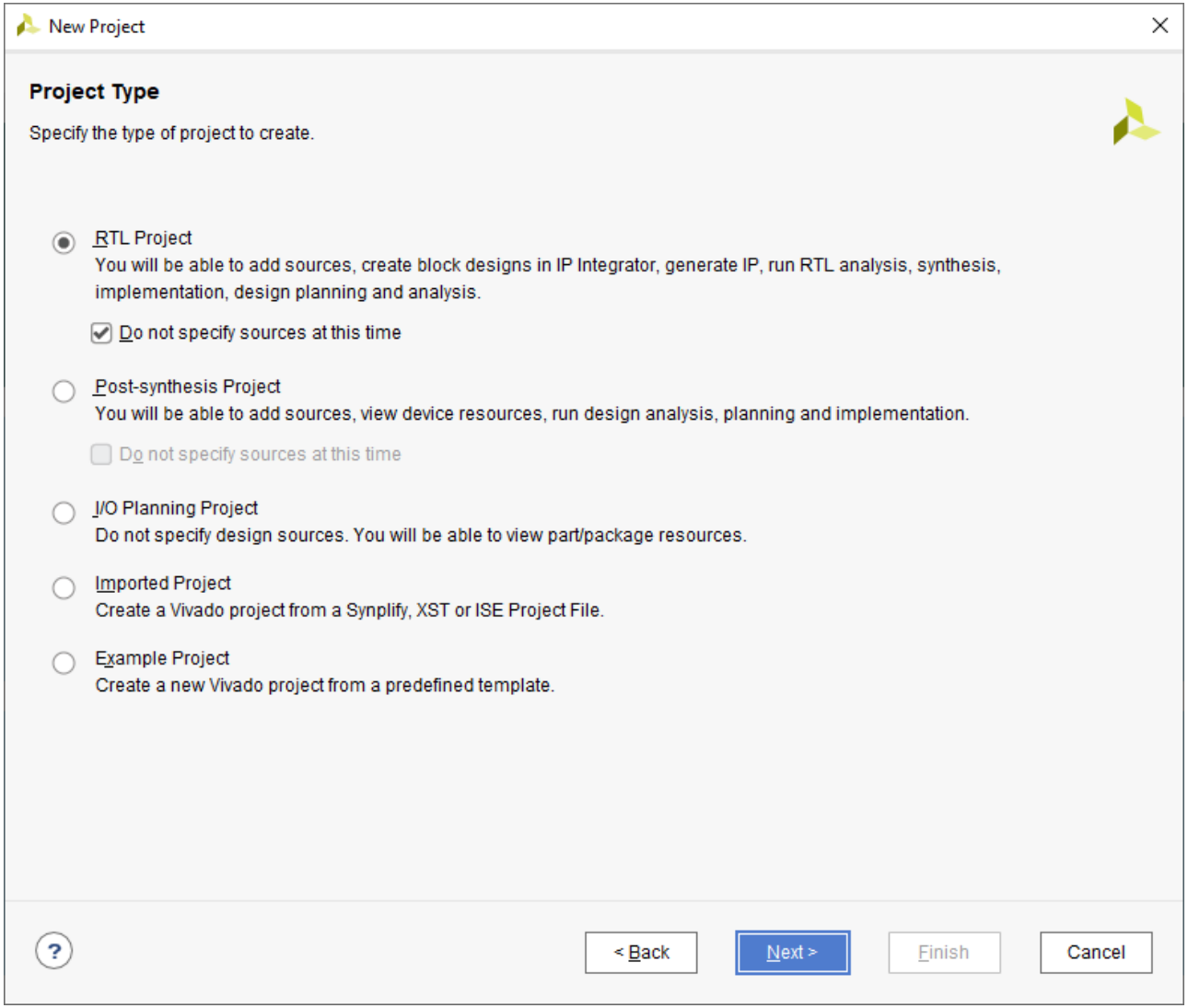
Example: Simple PL Design



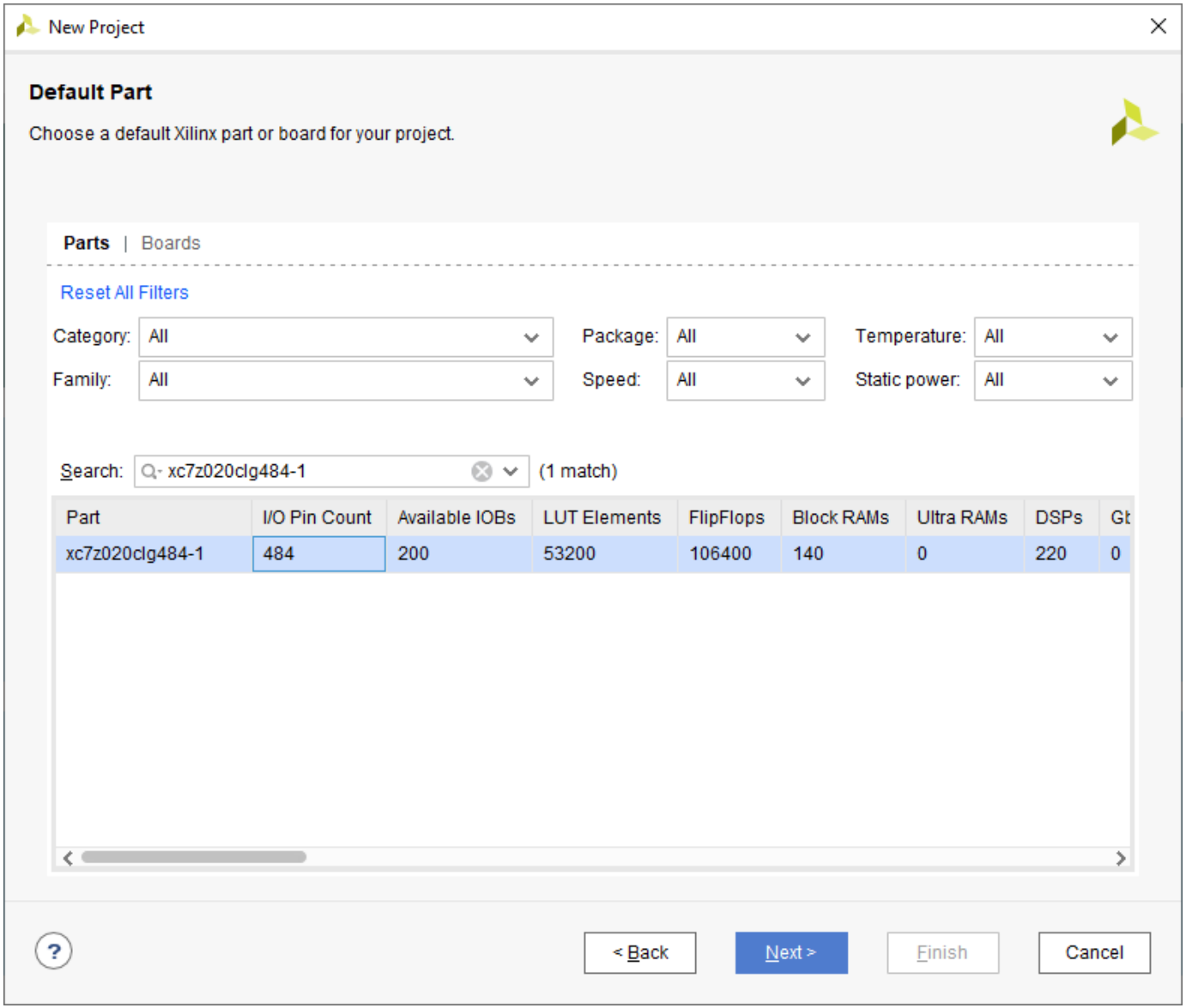
Example: Simple PL Design



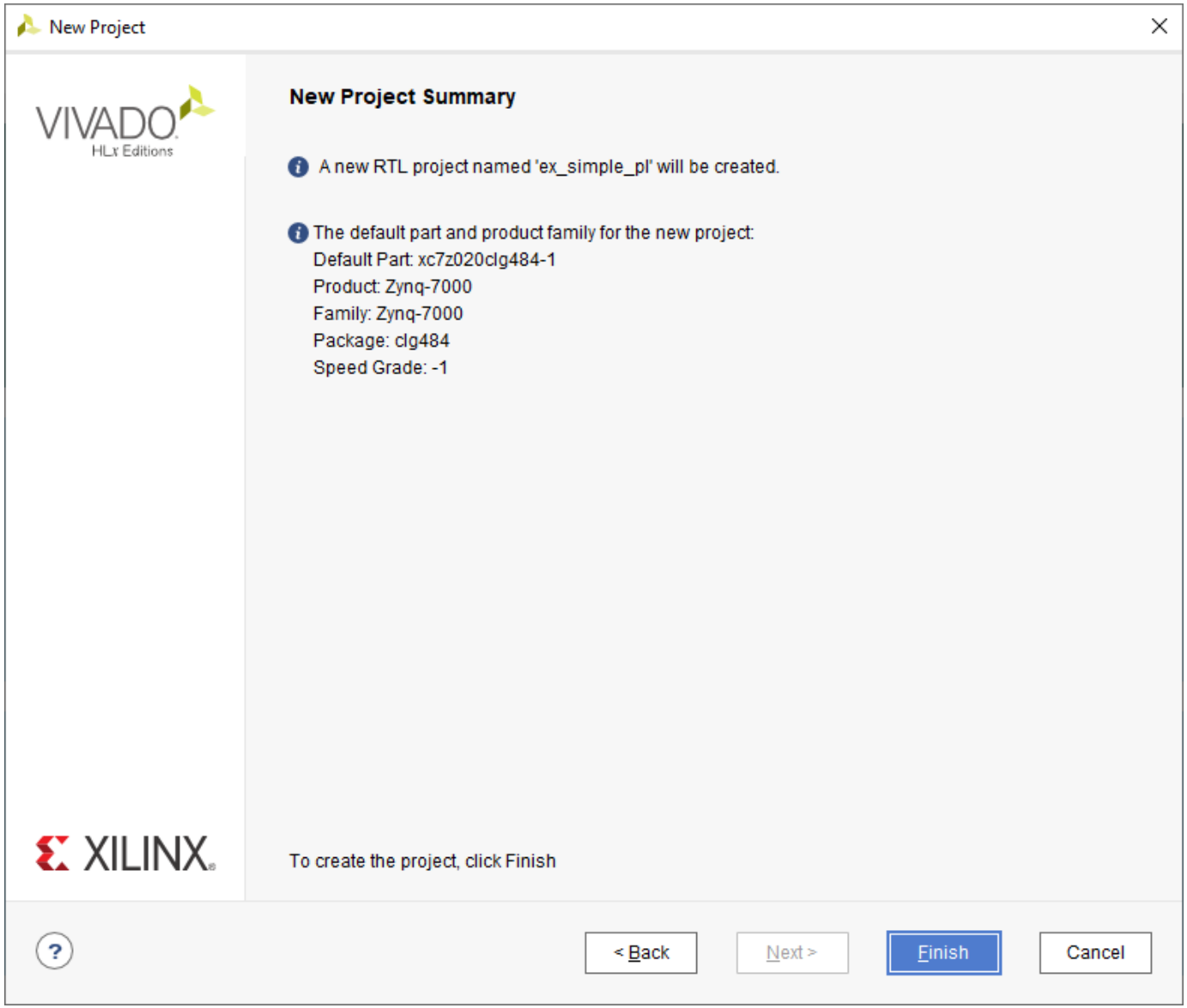
Example: Simple PL Design



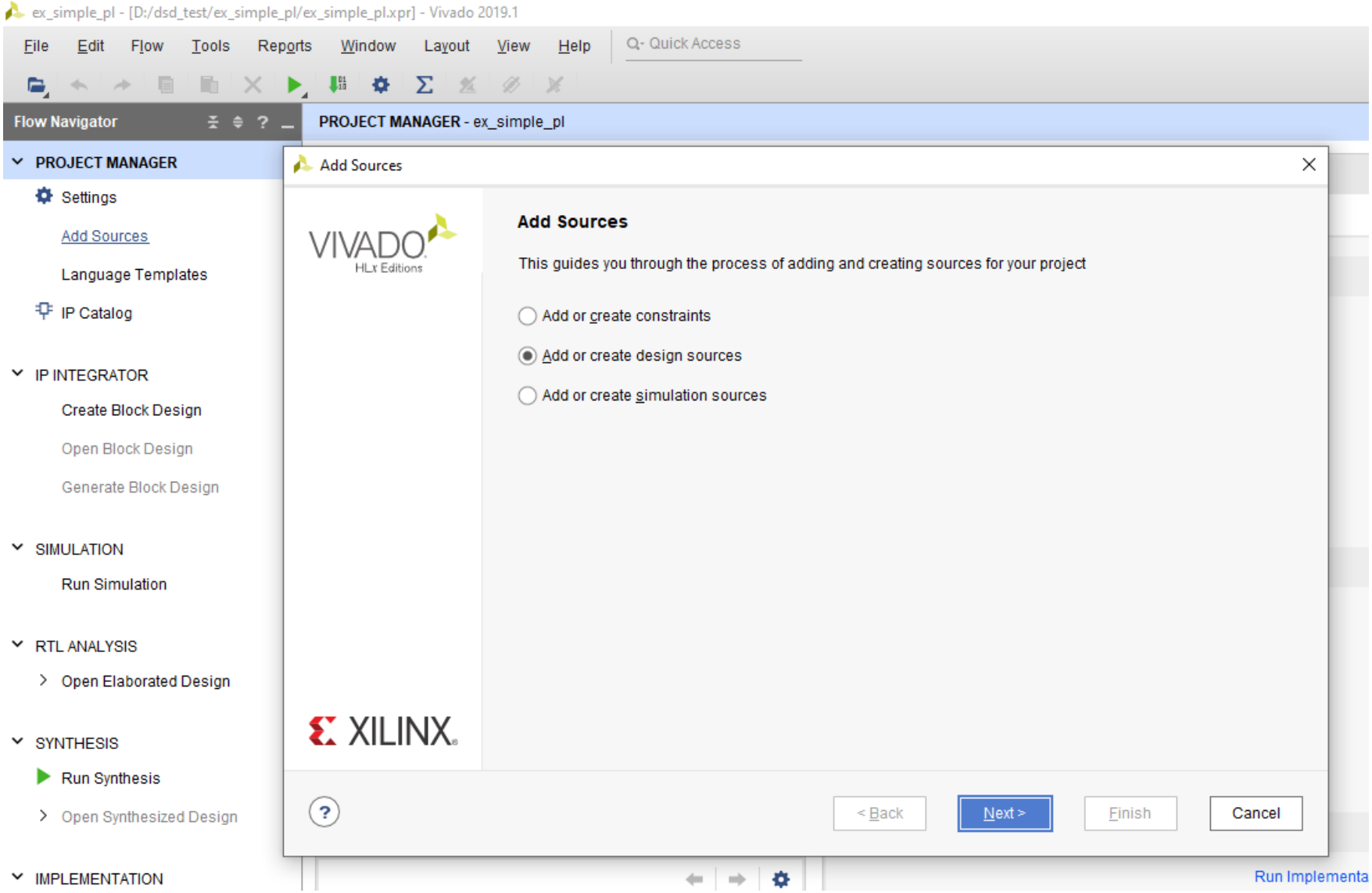
Example: Simple PL Design



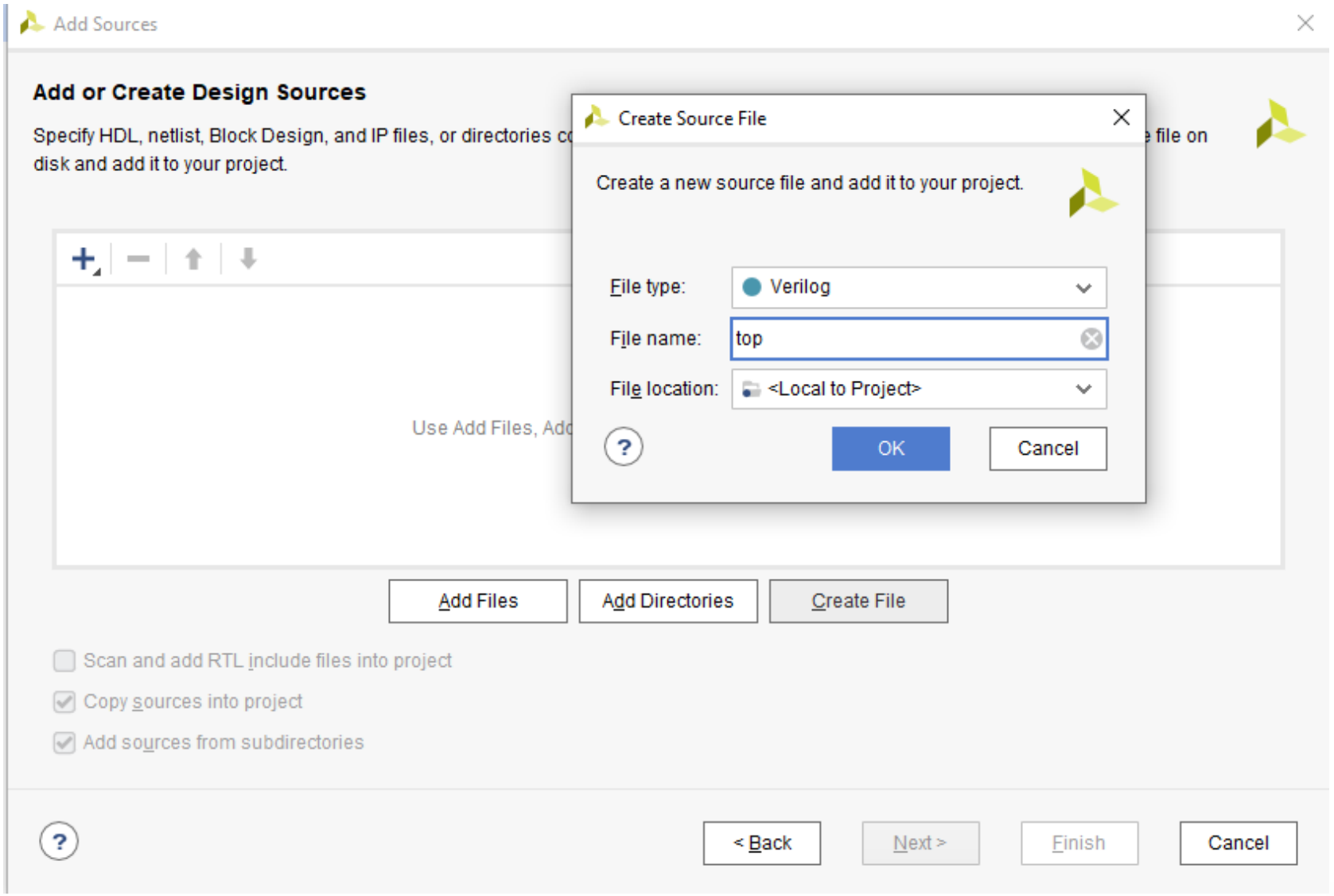
Example: Simple PL Design



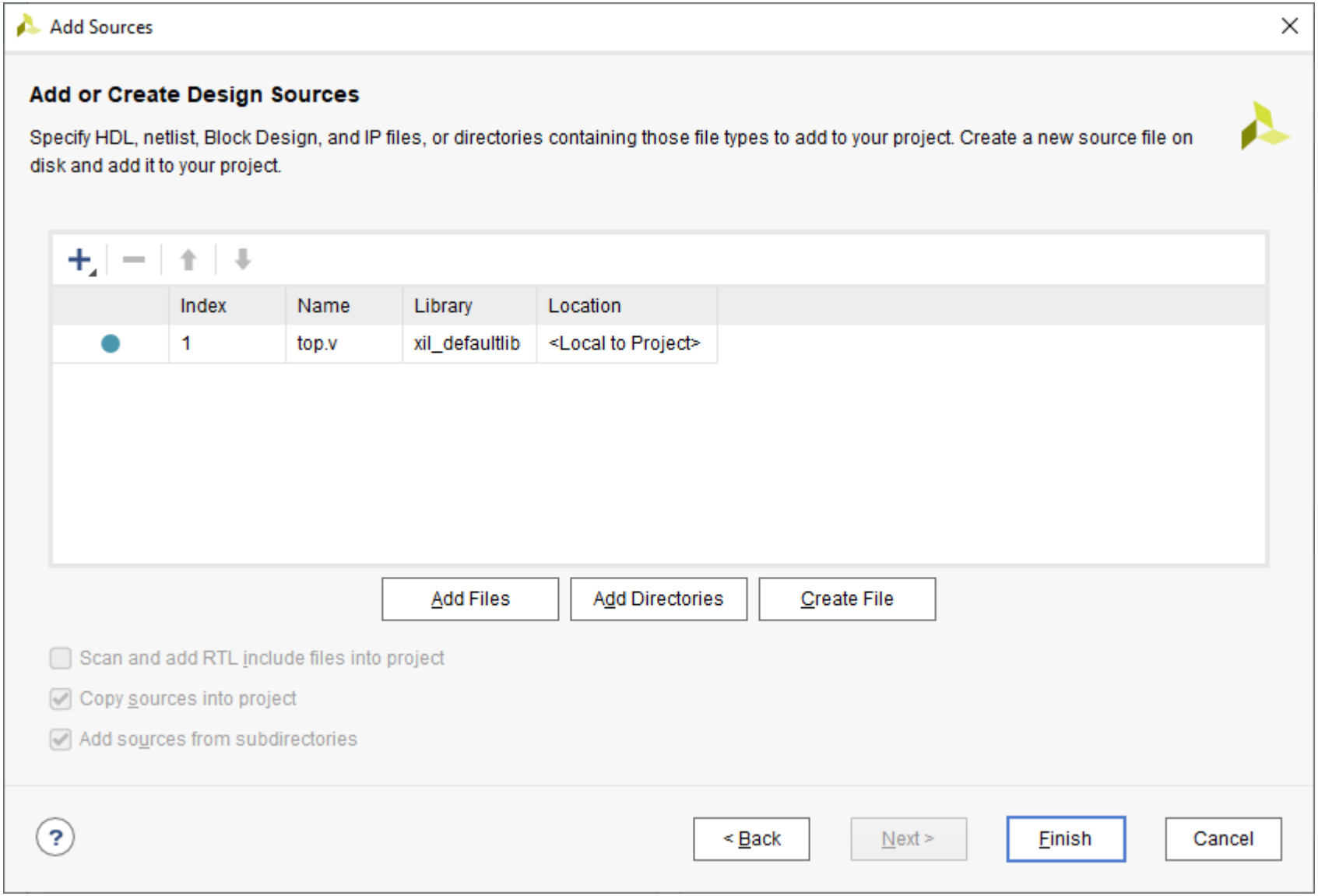
Example: Simple PL Design



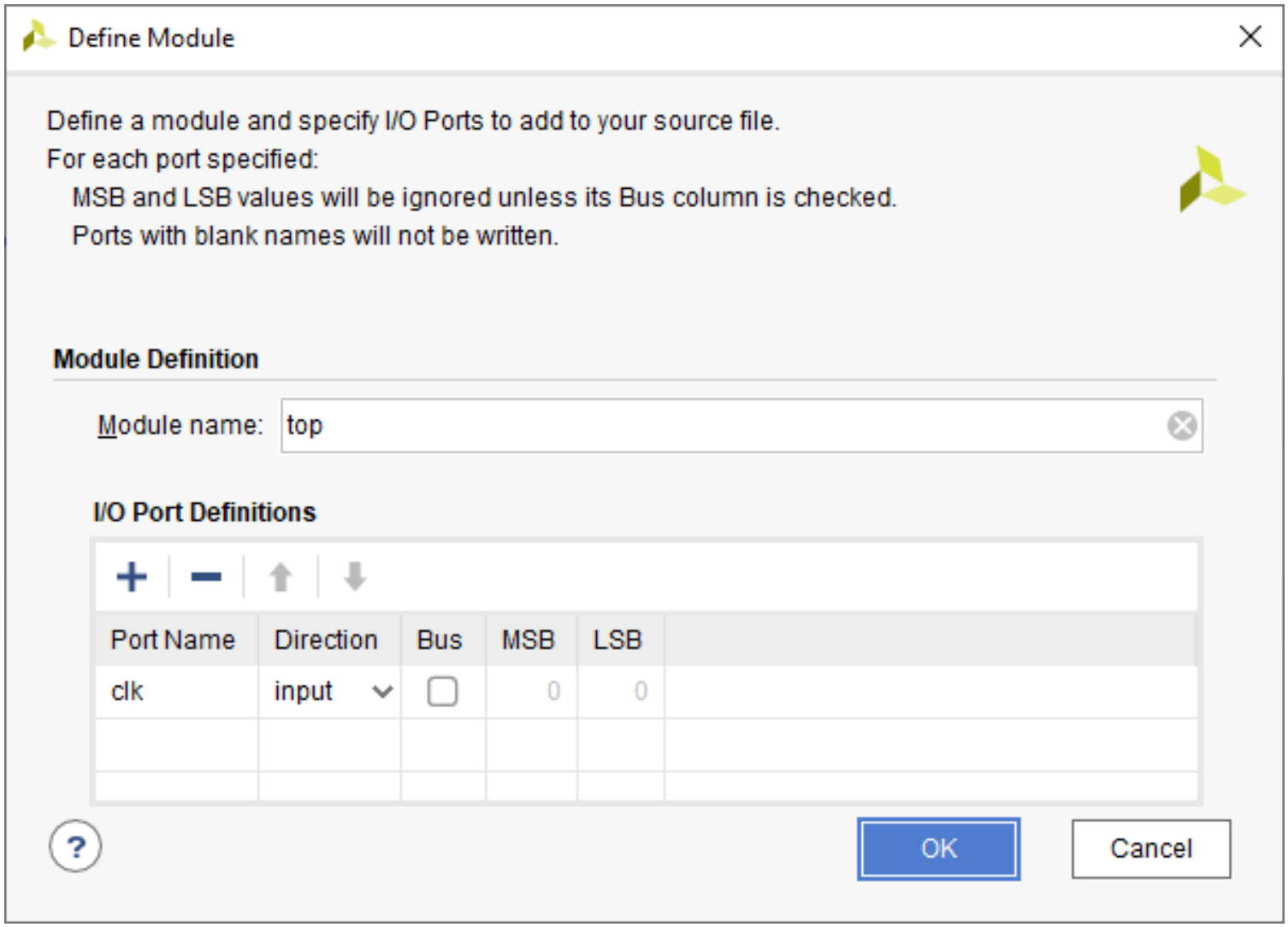
Example: Simple PL Design



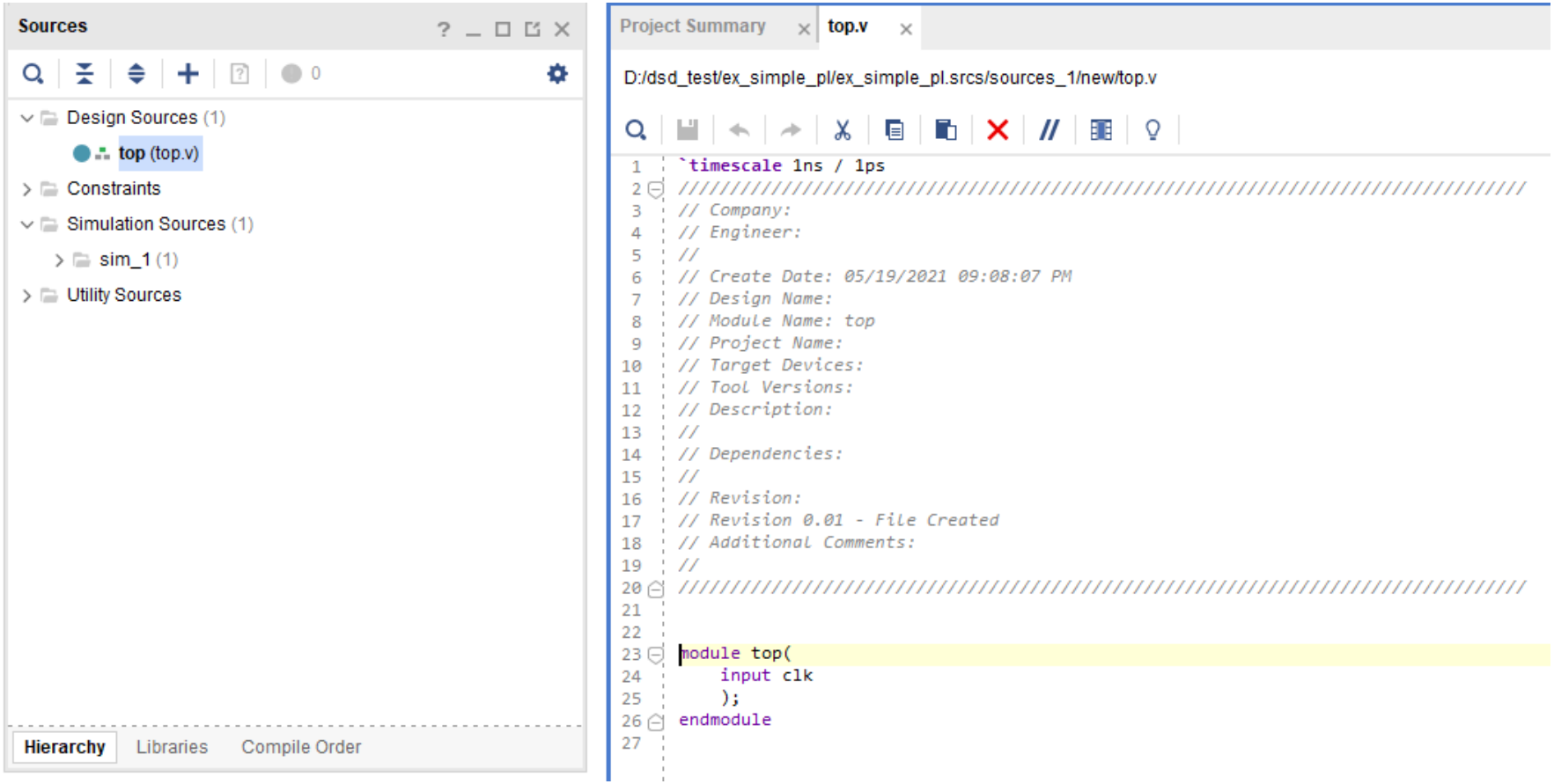
Example: Simple PL Design



Example: Simple PL Design



Example: Simple PL Design



Example: Simple PL Design

```
1  `timescale 1ns / 1ps
2
3  module top(
4      input wire clk,
5      input wire rstn,
6      input wire [2:0] pushbutton,
7      input wire [7:0] dipswitch,
8      output reg [7:0] led
9      );
10
11     reg [2:0] pushbutton_q;
12     reg [2:0] pushbutton_q2;
13     reg [2:0] pushbutton_q3;
14
15     always @ (posedge clk) begin
16         pushbutton_q <= pushbutton;
17         pushbutton_q2 <= pushbutton_q;
18         pushbutton_q3 <= pushbutton_q2;
19     end
20
21     always @ (posedge clk) begin
22         if (!rstn) begin
23             led <= 0;
24         end
25         else begin
26             if ((pushbutton_q2[0]) && (~pushbutton_q3[0])) begin
27                 led <= dipswitch;
28             end
29             else if ((pushbutton_q2[1]) && (~pushbutton_q3[1])) begin
30                 led <= {led[0], led[7:1]};
31             end
32             else if ((pushbutton_q2[2]) && (~pushbutton_q3[2])) begin
33                 led <= {led[6:0], led[7]};
34             end
35         end
36     end
37
38 endmodule
```

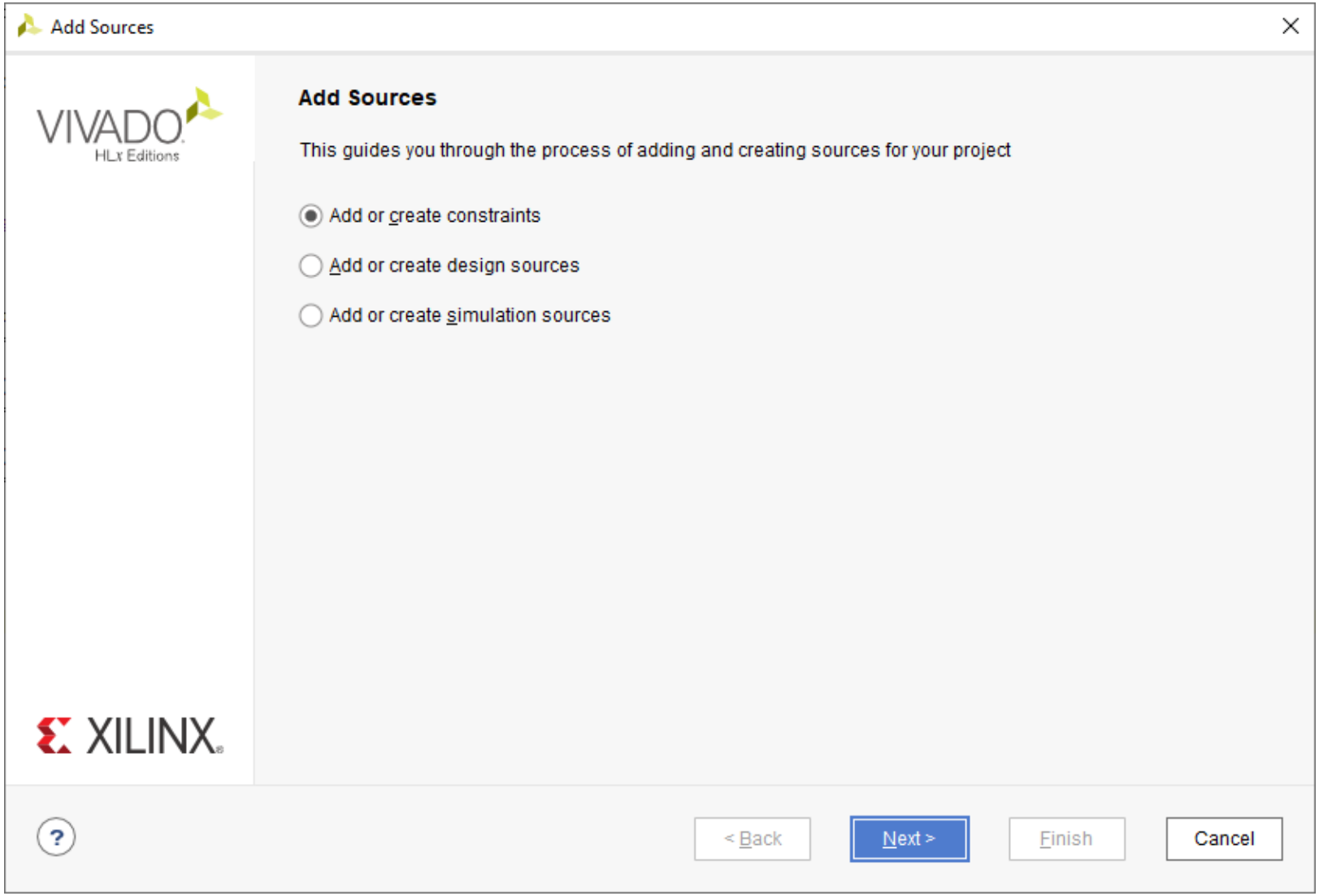
Example: Simple PL Design

```
`timescale 1ns / 1ps

module top(
    input wire clk,
    input wire rstn,
    input wire [2:0] pushbutton,
    input wire [7:0] dipswitch,
    output reg [7:0] led
    );

    reg [2:0] pushbutton_q;
    reg [2:0] pushbutton_q2;
    reg [2:0] pushbutton_q3;

    always @ (posedge clk) begin
        pushbutton_q <= pushbutton;
        pushbutton_q2 <= pushbutton_q;
        pushbutton_q3 <= pushbutton_q2;
    end
```

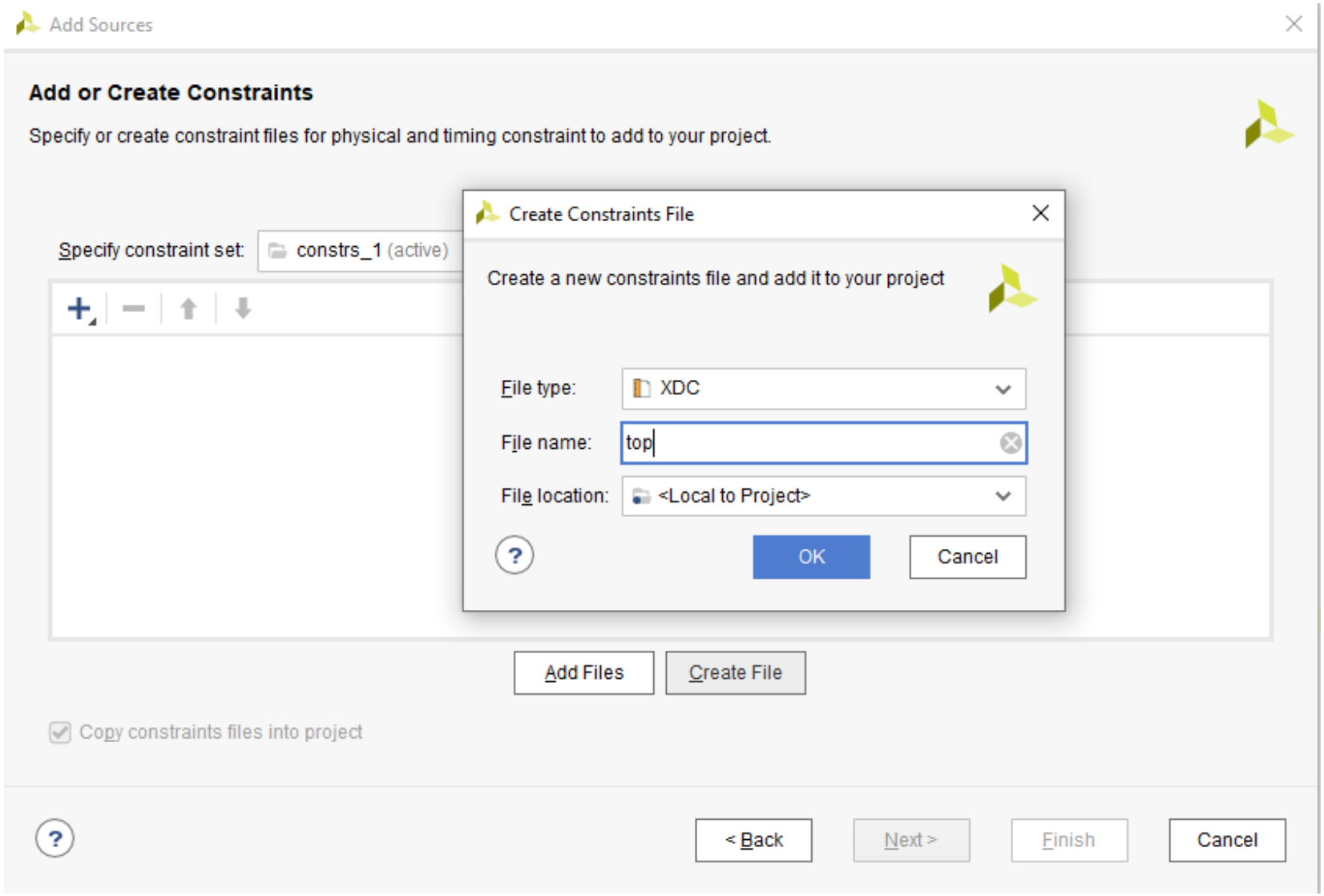
Example: Simple PL Design

```
always @ (posedge clk) begin
        if (!rstn) begin
            led <= 0;
        end
        else begin
            if ((pushbutton_q2[0]) && (~pushbutton_q3[0])) begin
                led <= dipswitch;
            end
            else if ((pushbutton_q2[1]) && (~pushbutton_q3[1])) begin
                led <= {led[0], led[7:1]};
            end
            else if ((pushbutton_q2[2]) && (~pushbutton_q3[2])) begin
                led <= {led[6:0], led[7]};
            end
        end
    end

endmodule
```

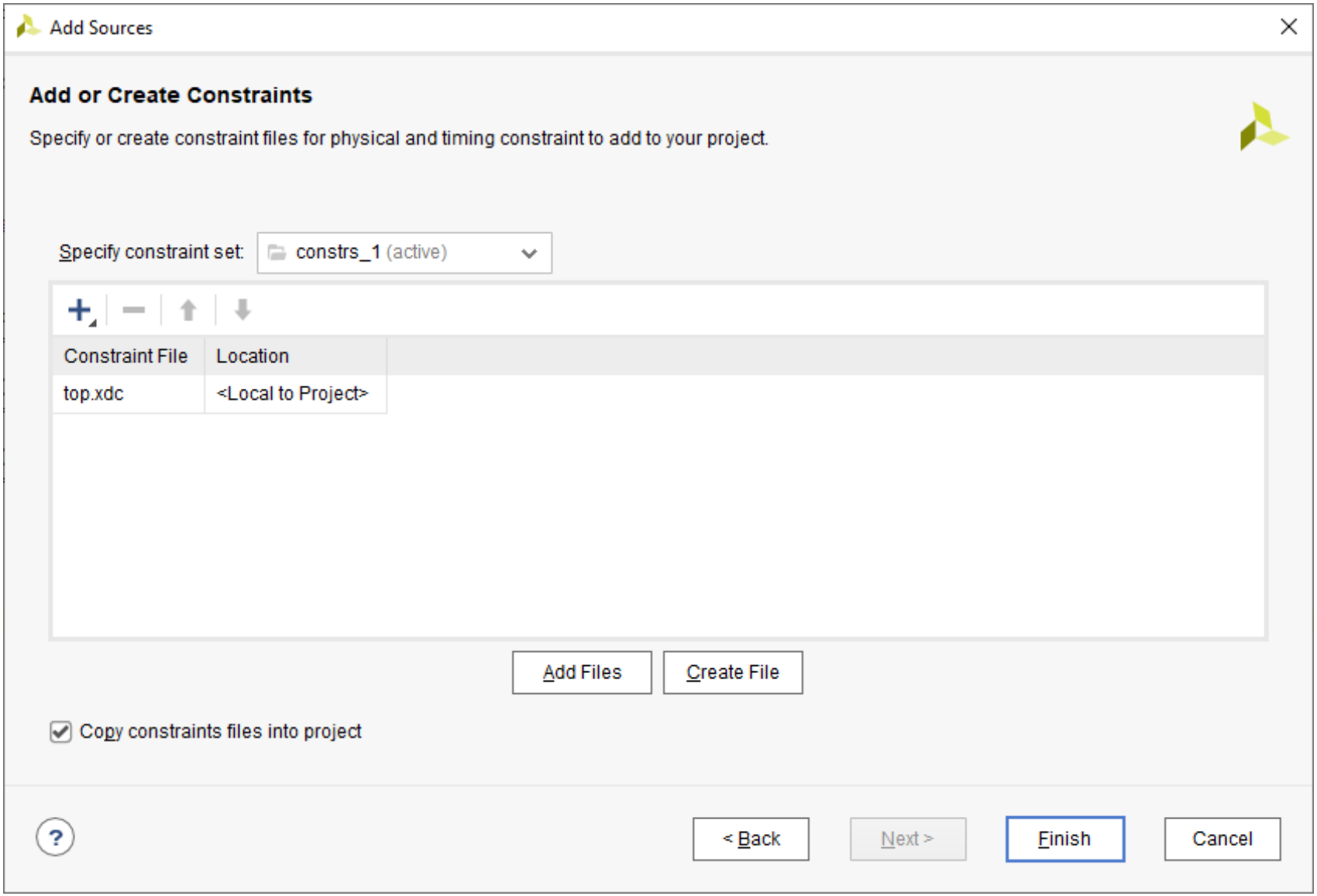
Example: Simple PL Design



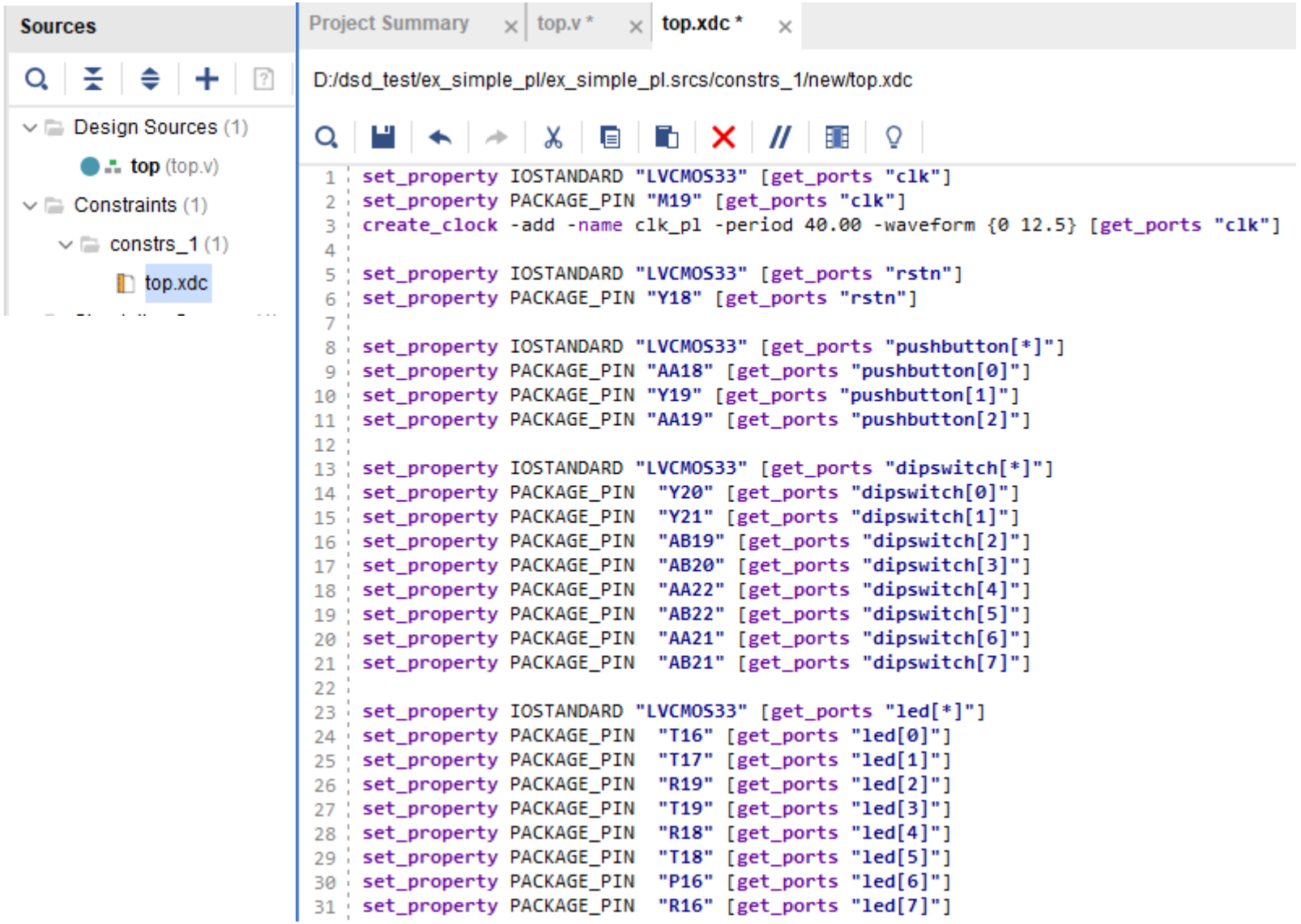
Example: Simple PL Design



Example: Simple PL Design



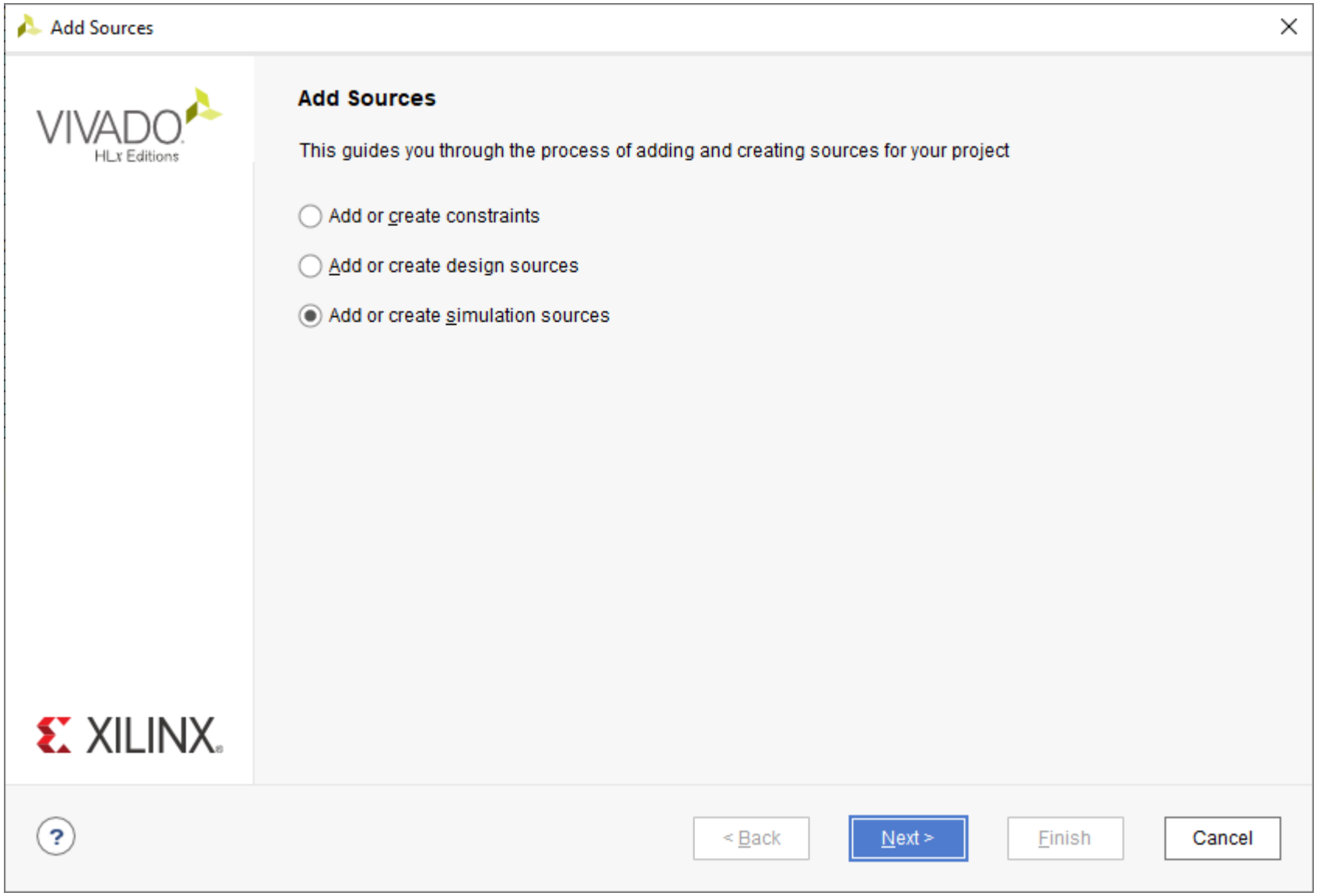
Example: Simple PL Design



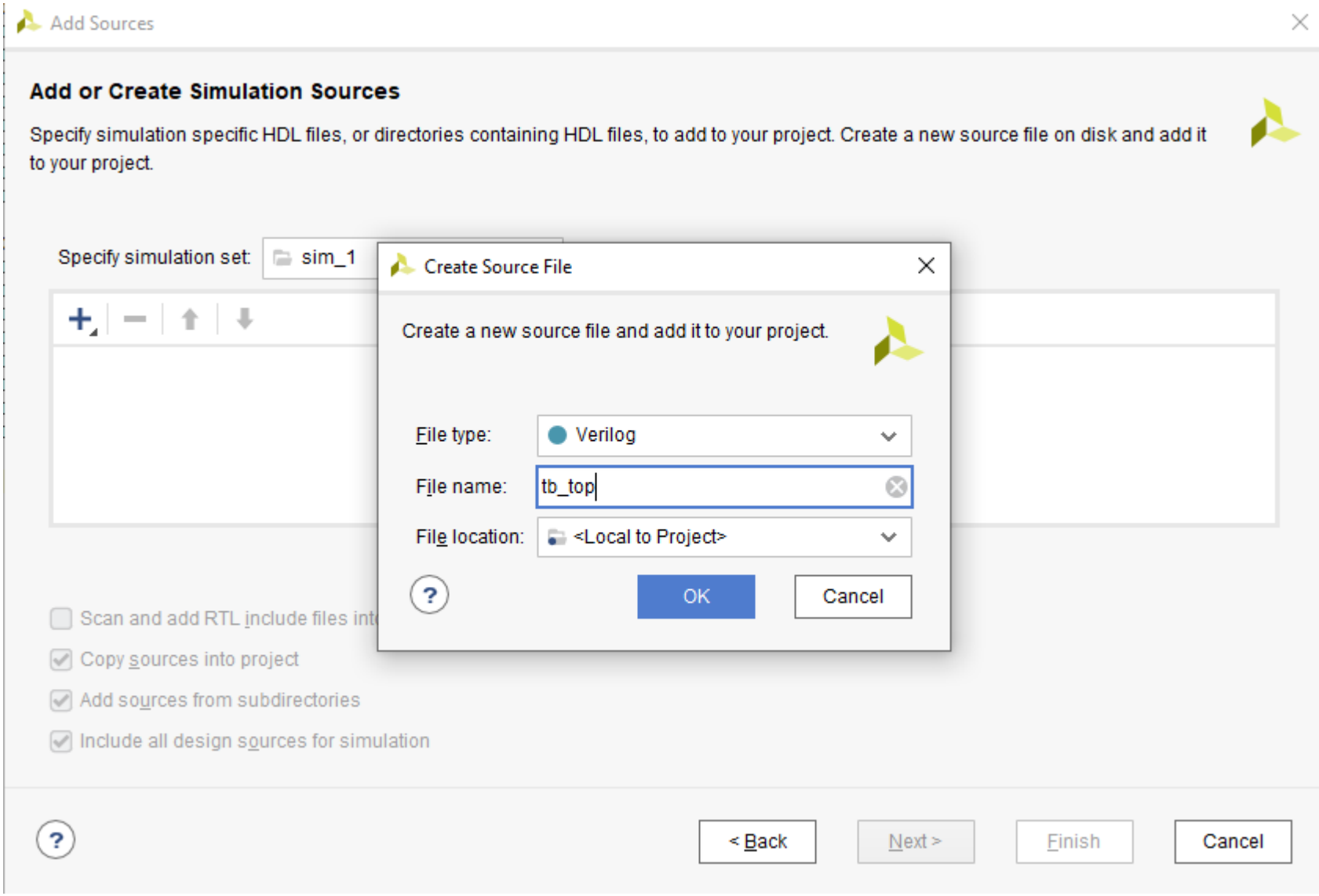
Example: Simple PL Design

```
set_property IOSTANDARD "LVCMOS33" [get_ports "clk"]
set_property PACKAGE_PIN "M19" [get_ports "clk"]
create_clock -add -name clk_pl -period 40.00 -waveform {0 12.5} [get_ports "clk"]

set_property IOSTANDARD "LVCMOS33" [get_ports "rstn"]
set_property PACKAGE_PIN "Y18" [get_ports "rstn"]

set_property IOSTANDARD "LVCMOS33" [get_ports "pushbutton[*]"]
set_property PACKAGE_PIN "AA18" [get_ports "pushbutton[0]"]
set_property PACKAGE_PIN "Y19" [get_ports "pushbutton[1]"]
set_property PACKAGE_PIN "AA19" [get_ports "pushbutton[2]"]

set_property IOSTANDARD "LVCMOS33" [get_ports "dipswitch[*]"]
set_property PACKAGE_PIN  "Y20" [get_ports "dipswitch[0]"]
set_property PACKAGE_PIN  "Y21" [get_ports "dipswitch[1]"]
set_property PACKAGE_PIN  "AB19" [get_ports "dipswitch[2]"]
set_property PACKAGE_PIN  "AB20" [get_ports "dipswitch[3]"]
set_property PACKAGE_PIN  "AA22" [get_ports "dipswitch[4]"]
set_property PACKAGE_PIN  "AB22" [get_ports "dipswitch[5]"]
set_property PACKAGE_PIN  "AA21" [get_ports "dipswitch[6]"]
set_property PACKAGE_PIN  "AB21" [get_ports "dipswitch[7]"]

set_property IOSTANDARD "LVCMOS33" [get_ports "led[*]"]
set_property PACKAGE_PIN  "T16" [get_ports "led[0]"]
set_property PACKAGE_PIN  "T17" [get_ports "led[1]"]
set_property PACKAGE_PIN  "R19" [get_ports "led[2]"]
set_property PACKAGE_PIN  "T19" [get_ports "led[3]"]
set_property PACKAGE_PIN  "R18" [get_ports "led[4]"]
set_property PACKAGE_PIN  "T18" [get_ports "led[5]"]
set_property PACKAGE_PIN  "P16" [get_ports "led[6]"]
set_property PACKAGE_PIN  "R16" [get_ports "led[7]"]
```

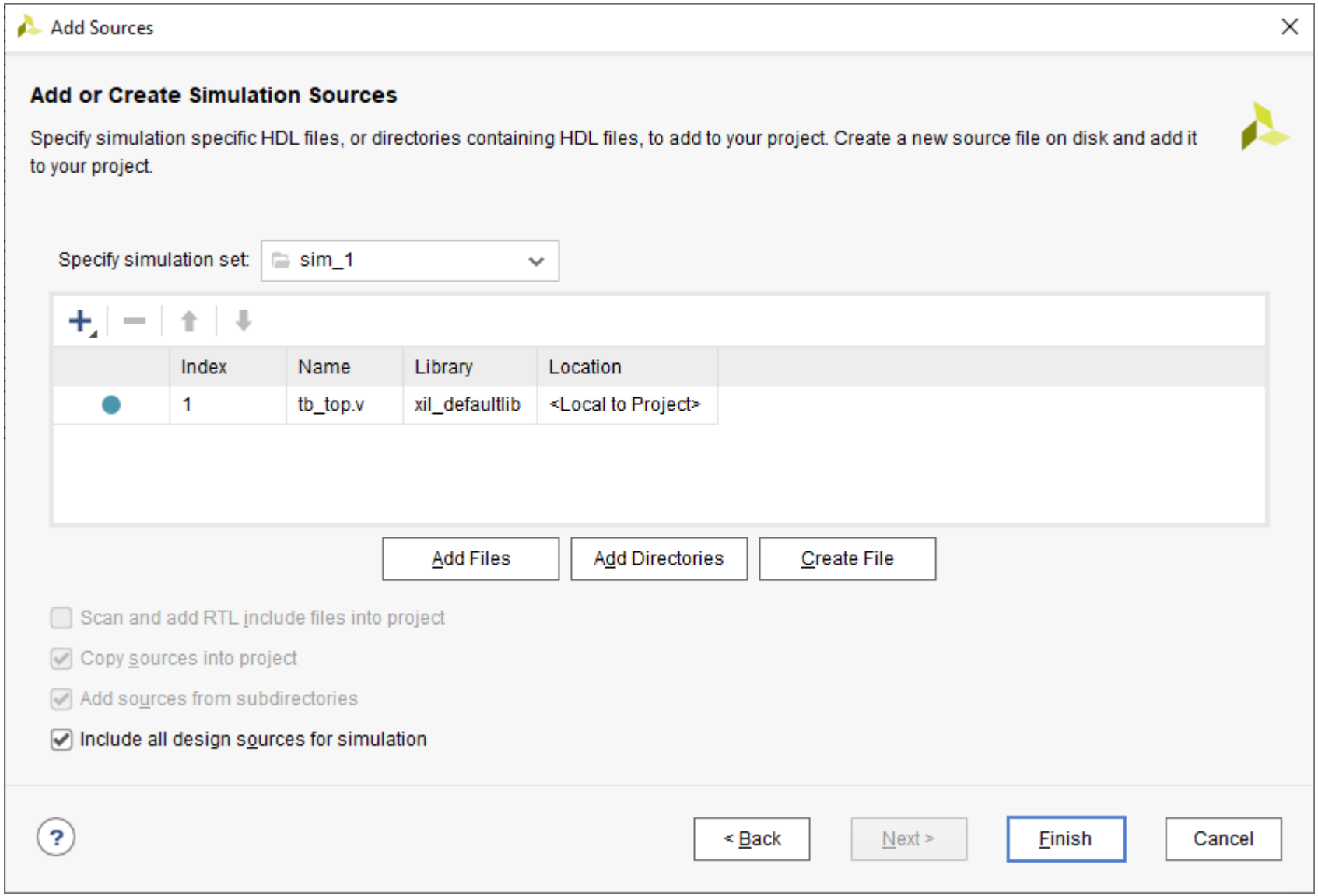
Example: Simple PL Design



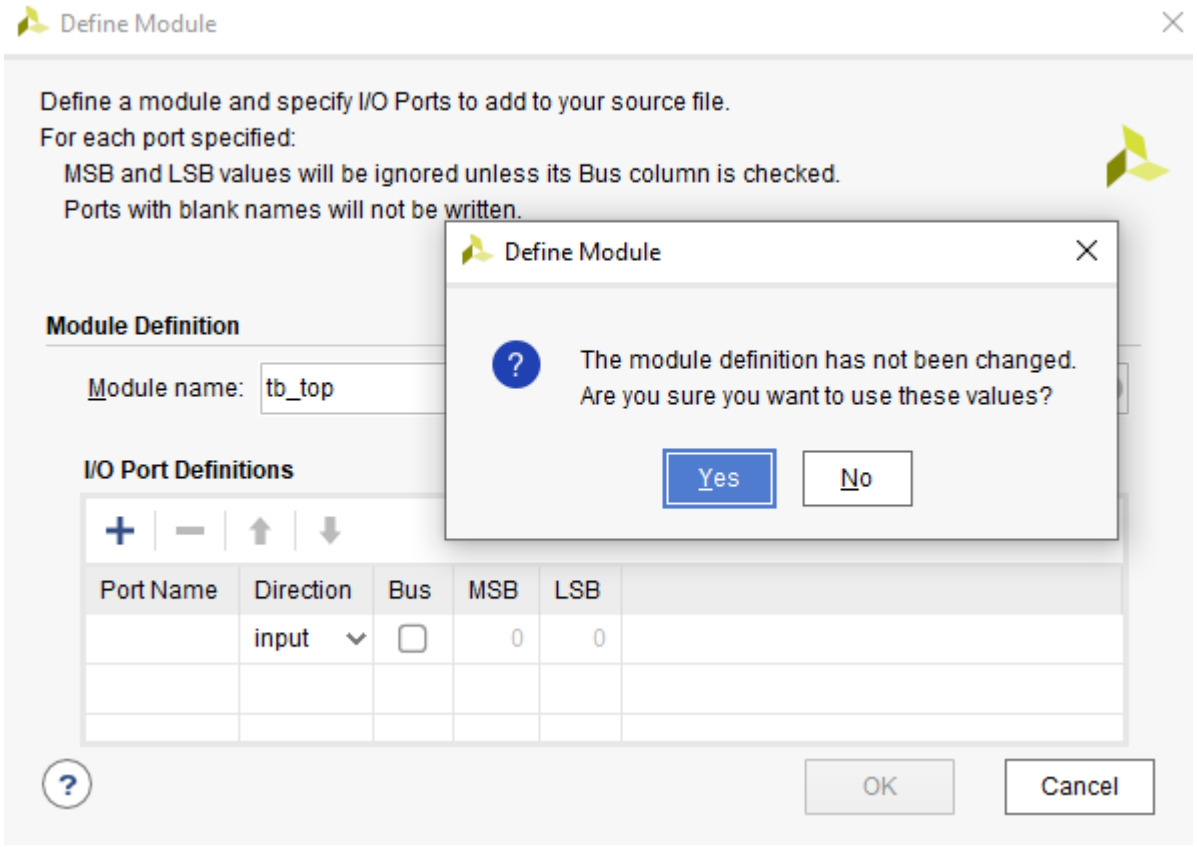
Example: Simple PL Design



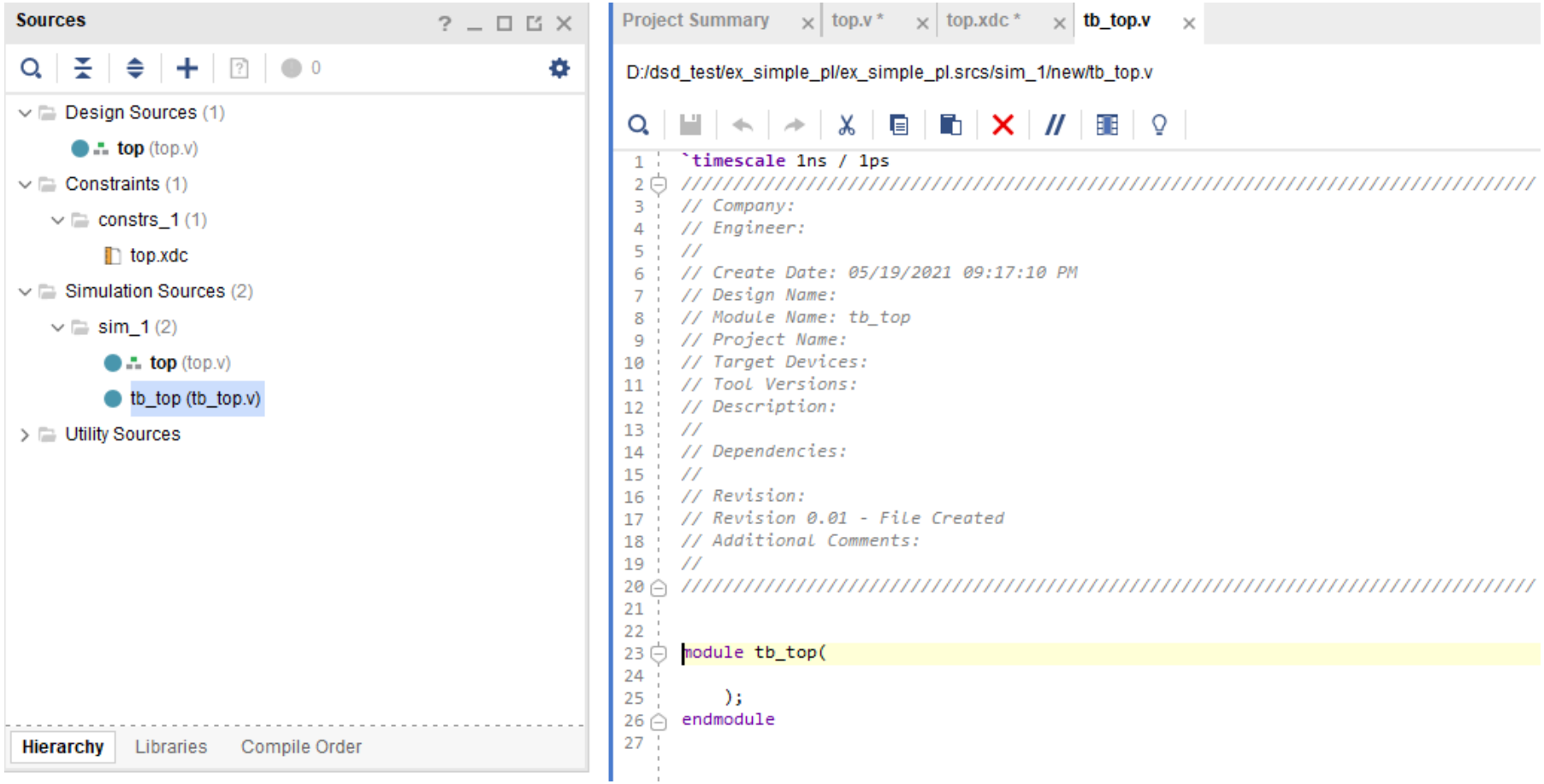
Example: Simple PL Design



Example: Simple PL Design



Example: Simple PL Design



Example: Simple PL Design

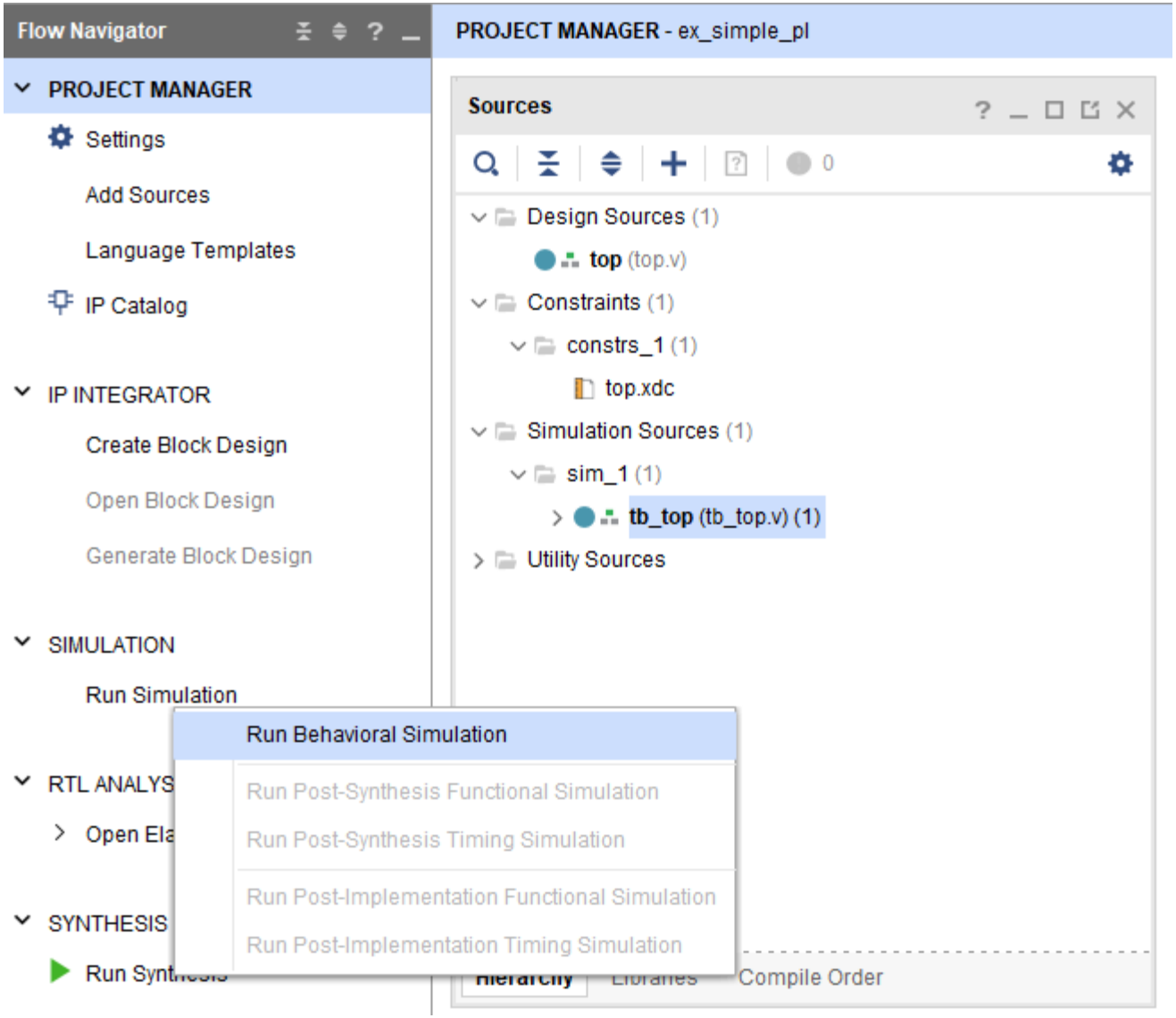
```
1  `timescale 1ns / 1ps
2
3  module tb_top();
4
5  reg clk, rstn;
6  reg [2:0] pushbutton;
7  reg [7:0] dipswitch;
8  wire [7:0] led;
9
10 top dut(
11     .clk(clk),
12     .rstn(rstn),
13     .pushbutton(pushbutton),
14     .dipswitch(dipswitch),
15     .led(led)
16 );
17
18 initial begin
19     clk = 0;
20     forever
21         #5 clk = ~clk;
22 end
23
24 initial begin
25             rstn = 1;
26     #200    rstn = 0;
27     #100    rstn = 1;
28 end
29
30 initial begin
31     // test pushbutton[0] after system reset
32             dipswitch = 8'b0000_1111;
33             pushbutton = 3'b000;
34     #400    pushbutton = 3'b001;
35     #50     pushbutton = 3'b000;
36     #50     dipswitch = 8'b1111_1110;
37     #50     pushbutton = 3'b001;
38     #50     pushbutton = 3'b000;
39     // test pushbutton[1]
40     #50     pushbutton = 3'b010;
41     #50     pushbutton = 3'b000;
42     #100    pushbutton = 3'b100;
43     #50     pushbutton = 3'b000;
44     #100    $stop;
45 end
46
47 endmodule
```

Example: Simple PL Design

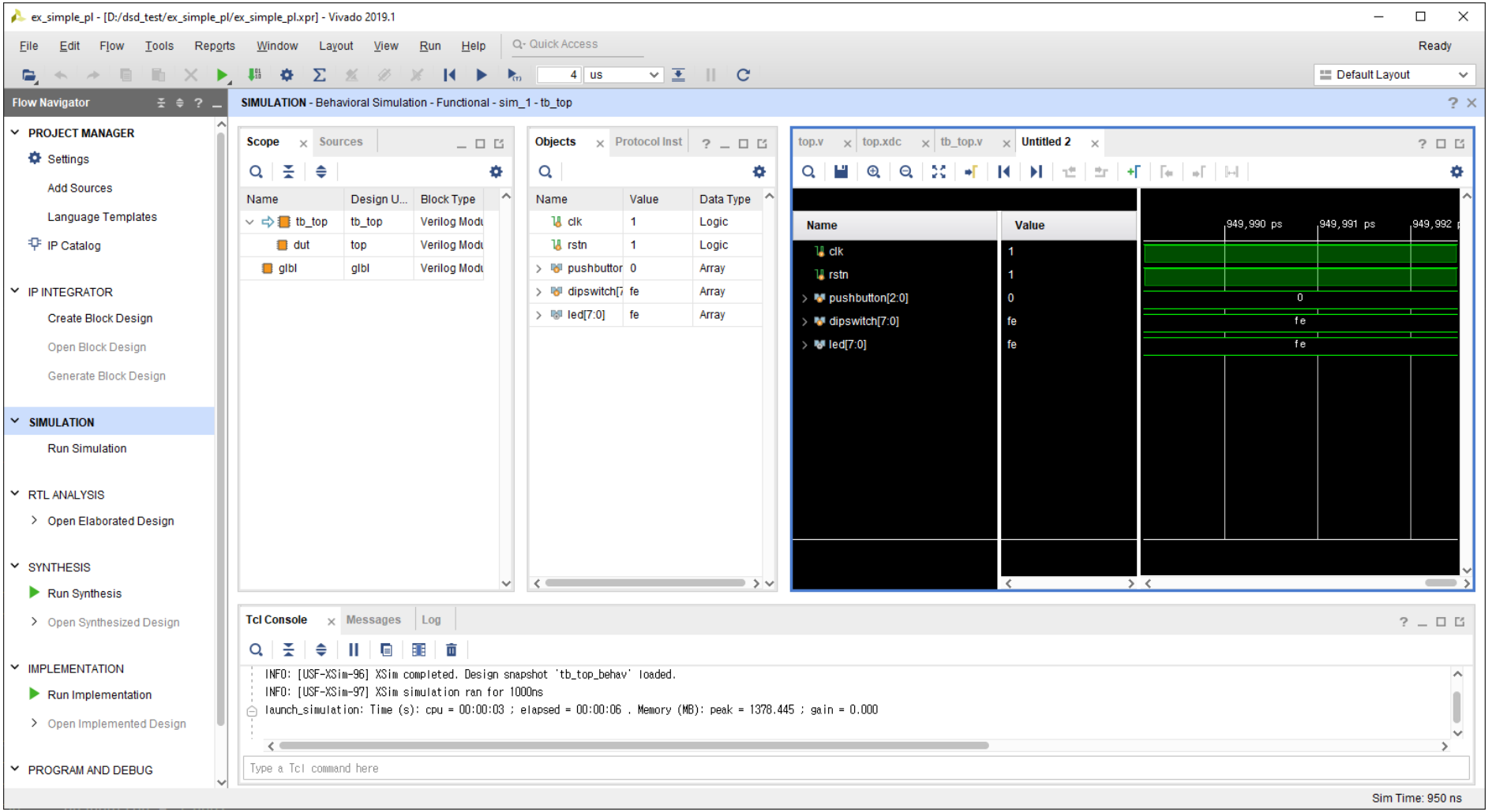
```
`timescale 1ns / 1ps

module tb_top();
reg clk, rstn;
reg [2:0] pushbutton;
reg [7:0] dipswitch;
wire [7:0] led;

top dut(
    .clk(clk),
    .rstn(rstn),
    .pushbutton(pushbutton),
    .dipswitch(dipswitch),
    .led(led)
);

initial begin
    clk = 0;
    forever
        #5 clk = ~clk;
end

initial begin
            rstn = 1;
    #200    rstn = 0;
    #100    rstn = 1;
end
```

```
initial begin
    // test pushbutton[0] after system reset
            dipswitch = 8'b0000_1111;
            pushbutton = 3'b000;
    #400     pushbutton = 3'b001;
    #50      pushbutton = 3'b000;
    #50      dipswitch = 8'b1111_1110;
    #50      pushbutton = 3'b001;
    #50      pushbutton = 3'b000;
    // test pushbutton[1]
    #50      pushbutton = 3'b010;
    #50      pushbutton = 3'b000;
    #100     pushbutton = 3'b100;
    #50      pushbutton = 3'b000;
    #100     $stop;
end

endmodule
```

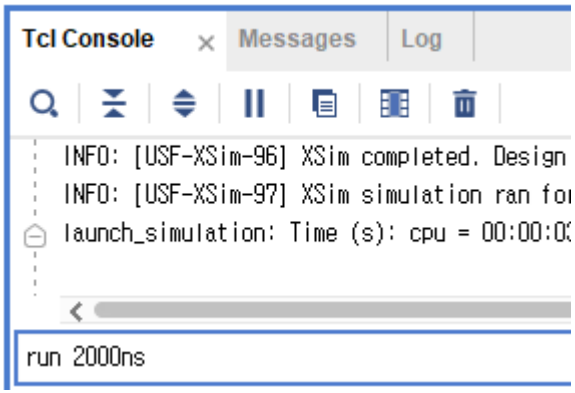
Example: Simple PL Design

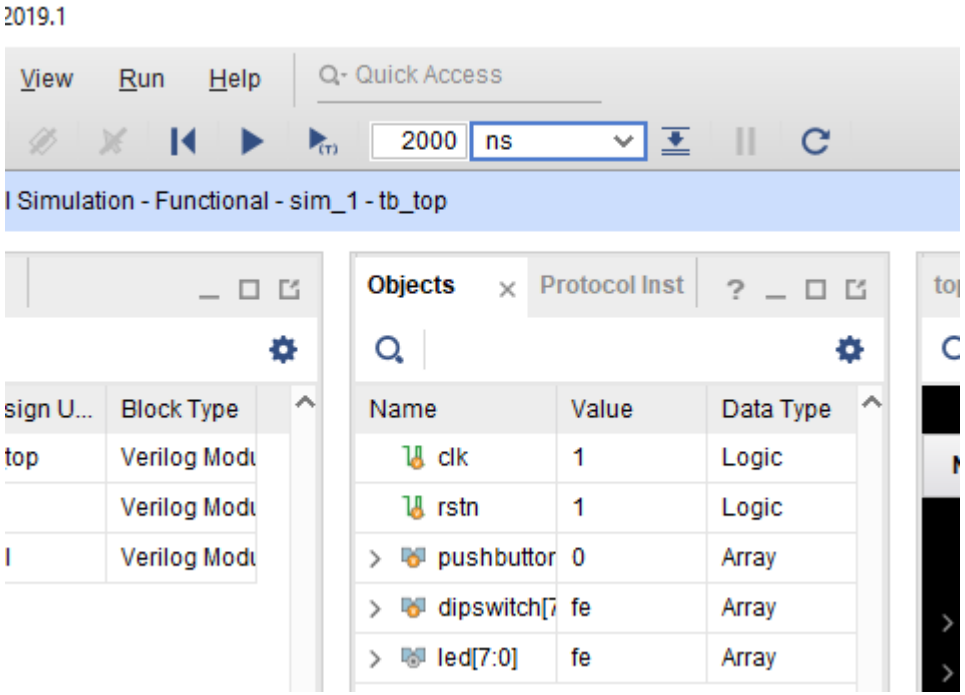


Example: Simple PL Design

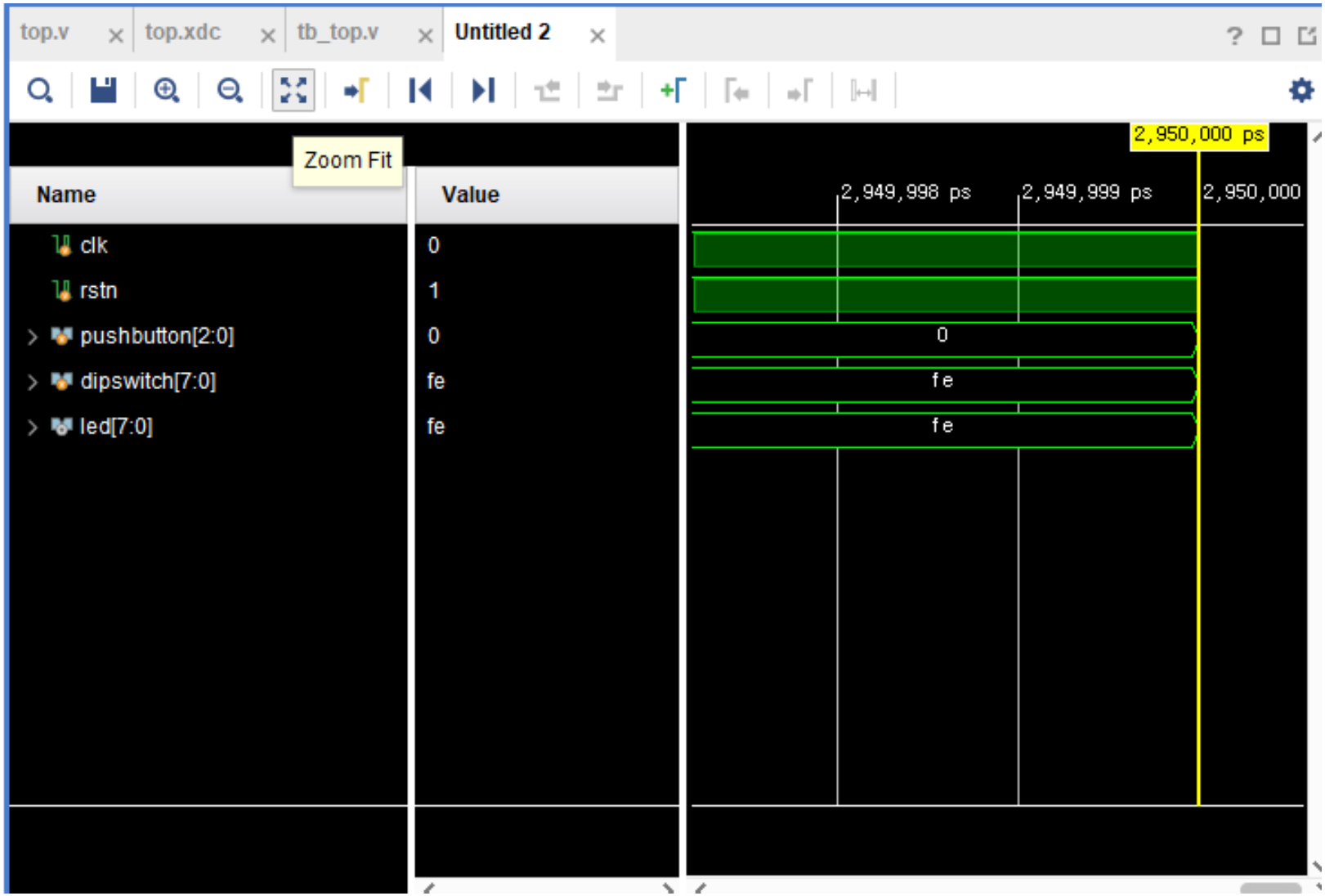


Example: Simple PL Design

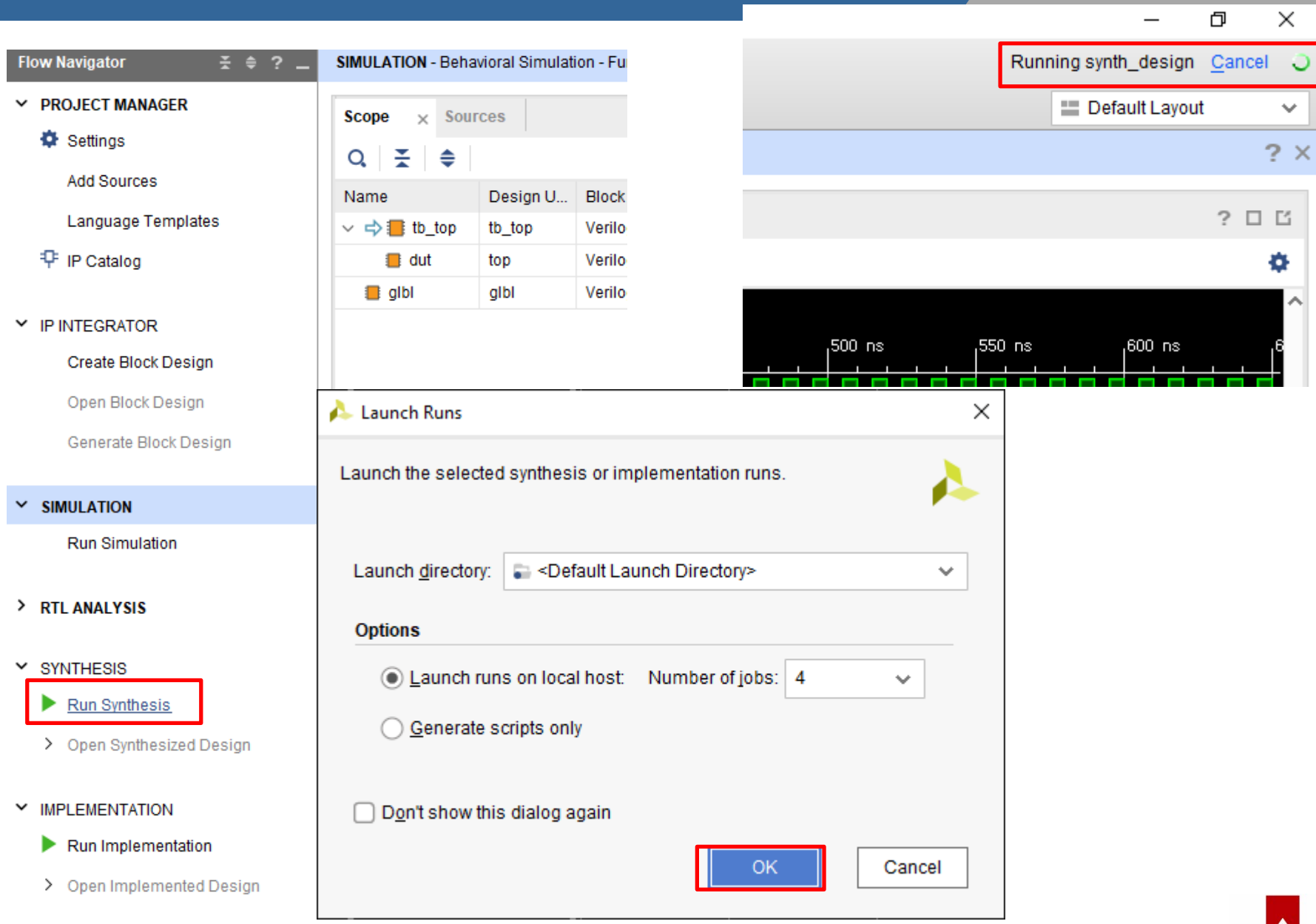




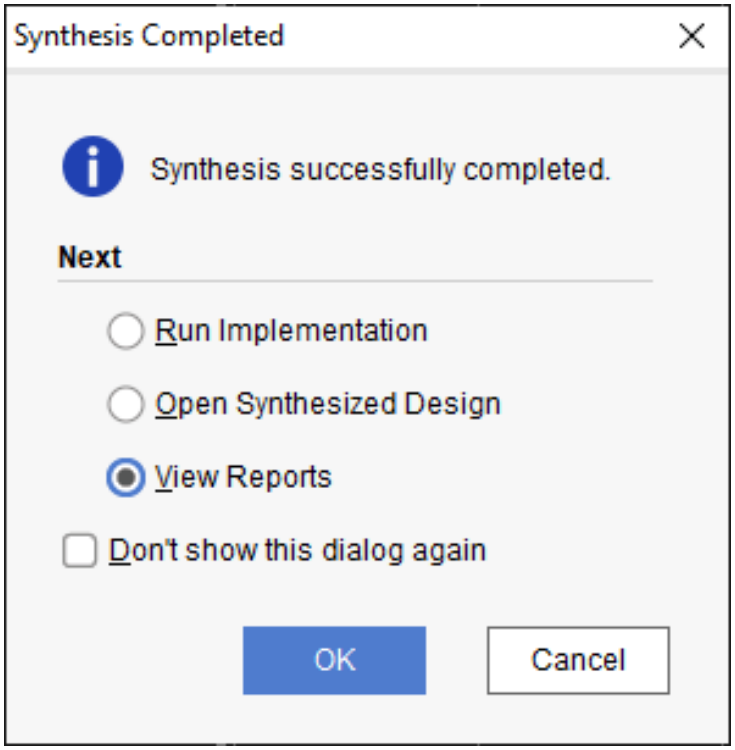
Example: Simple PL Design

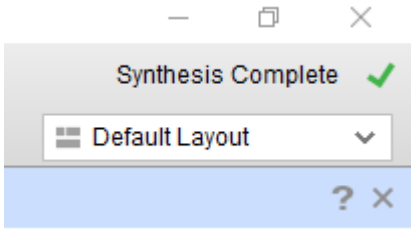


Example: Simple PL Design

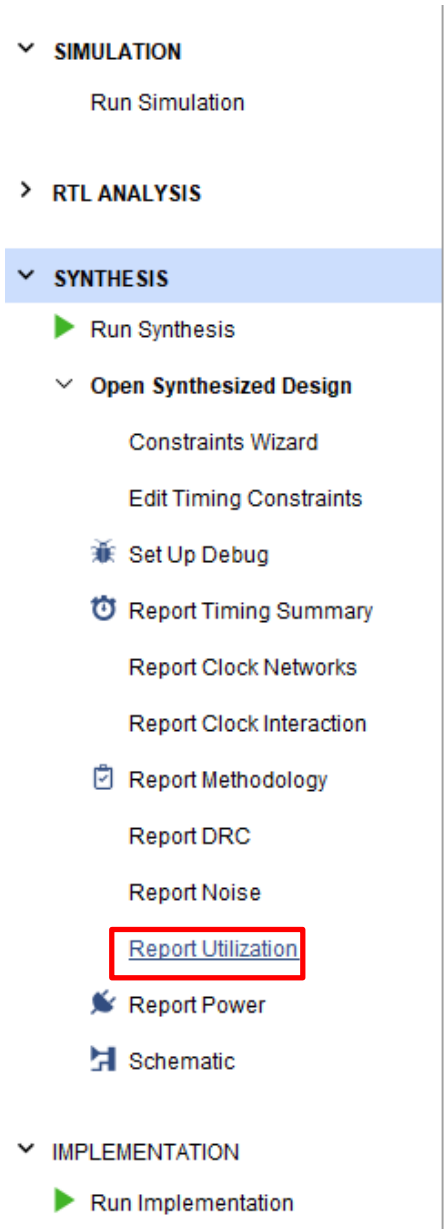


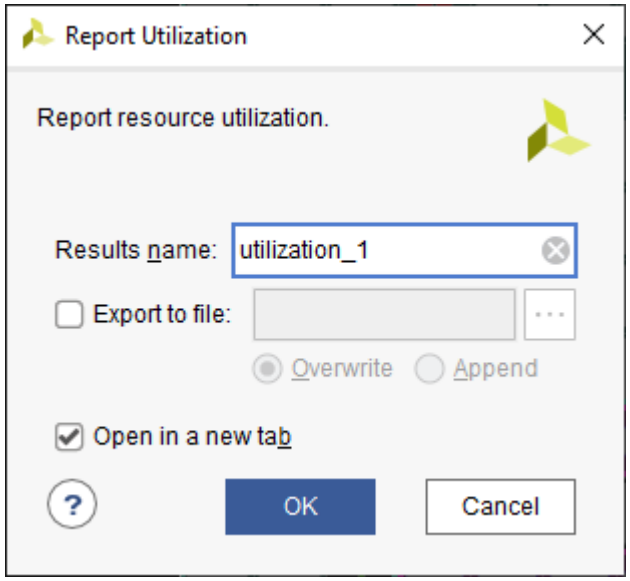
Example: Simple PL Design

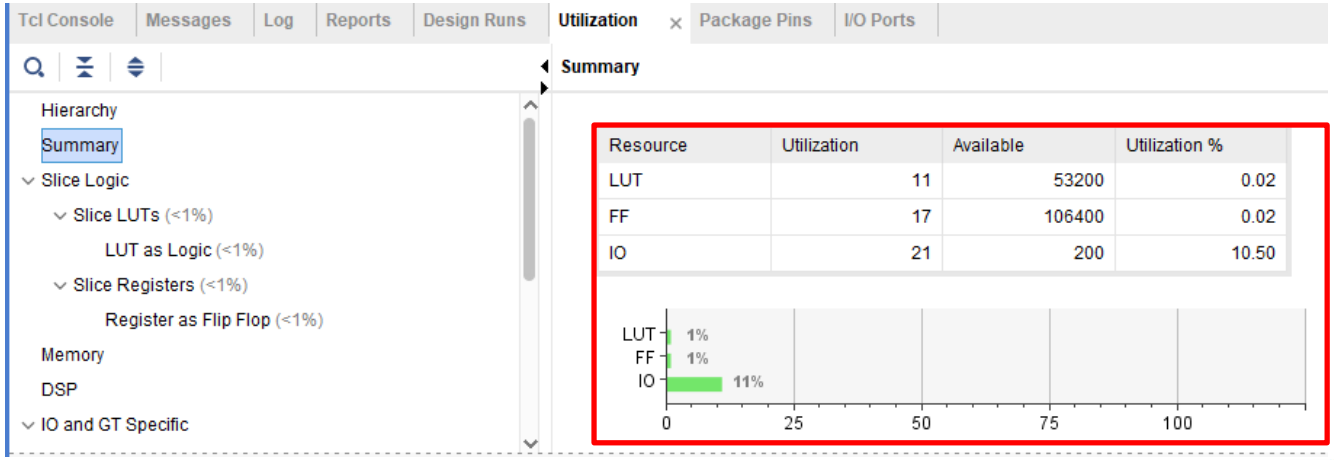




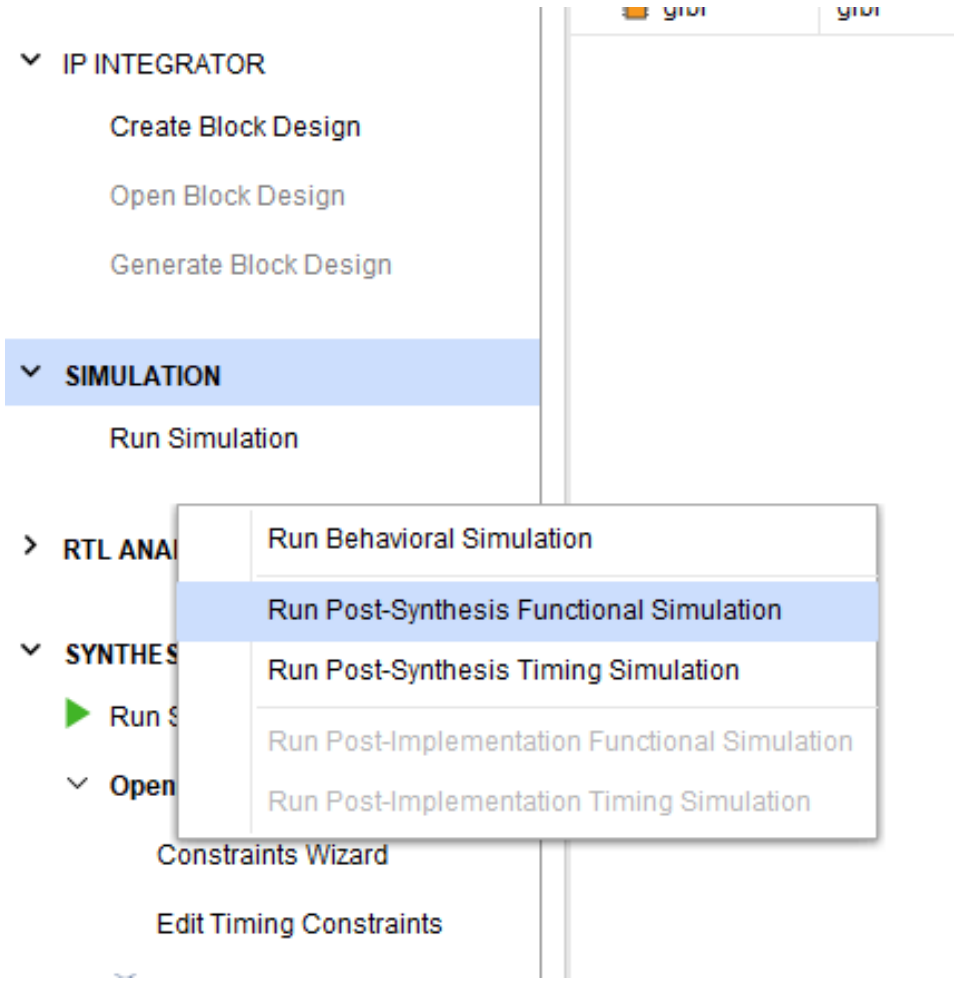
Example: Simple PL Design



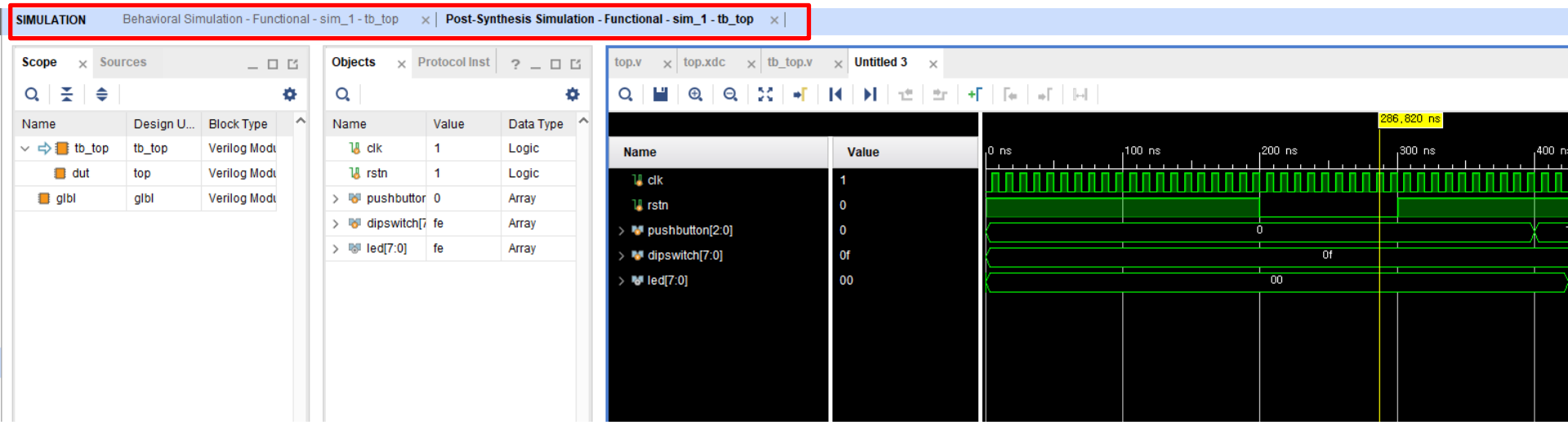


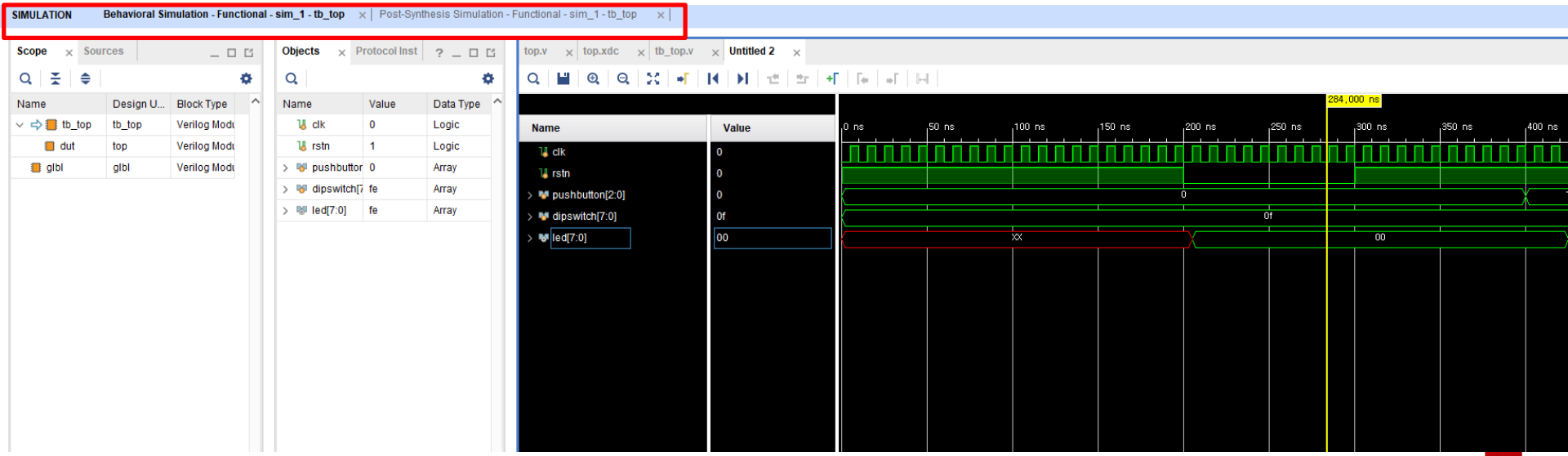


Example: Simple PL Design

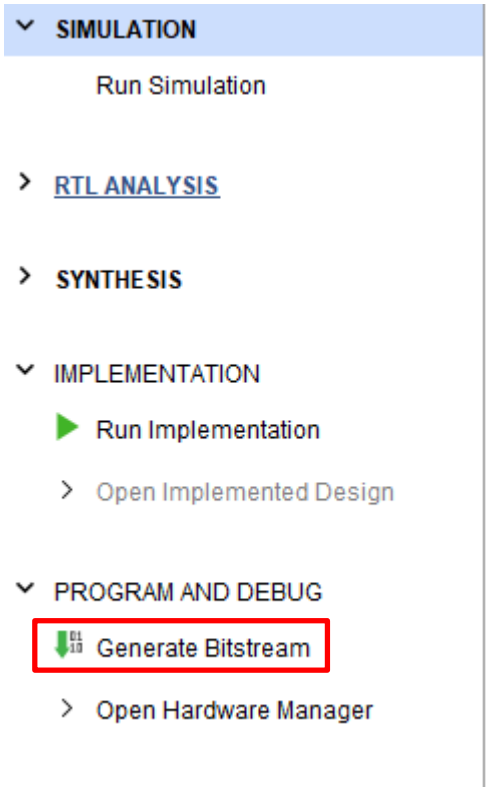


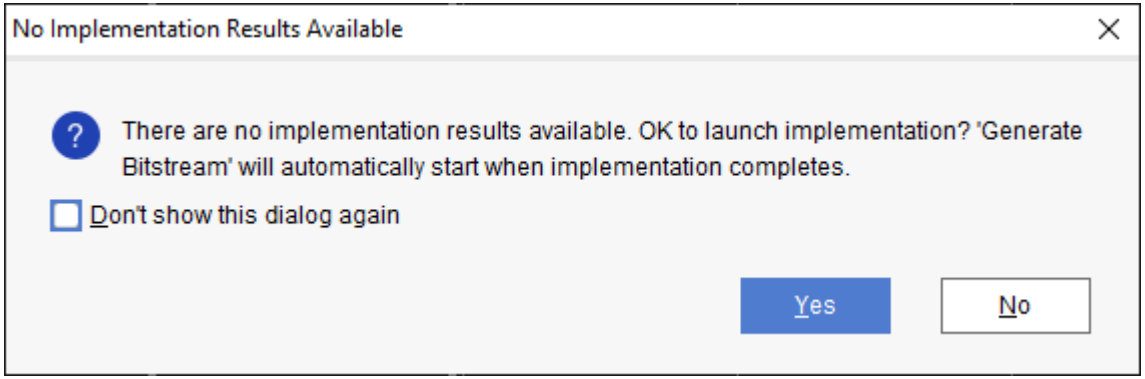
Example: Simple PL Design

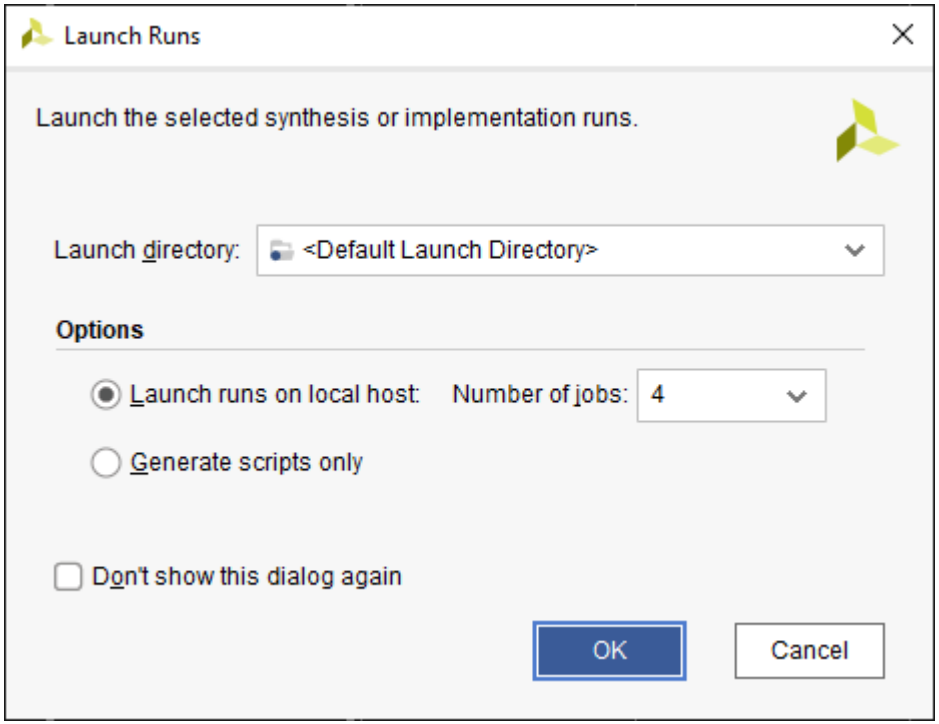




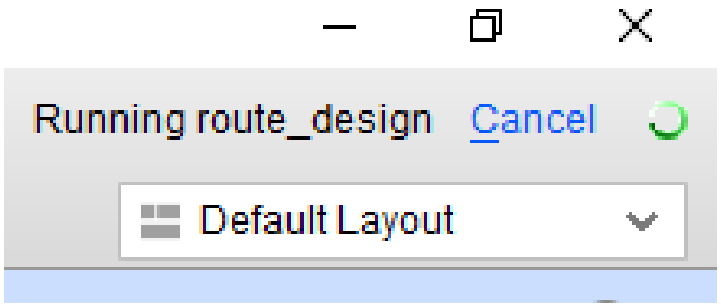
Example: Simple PL Design

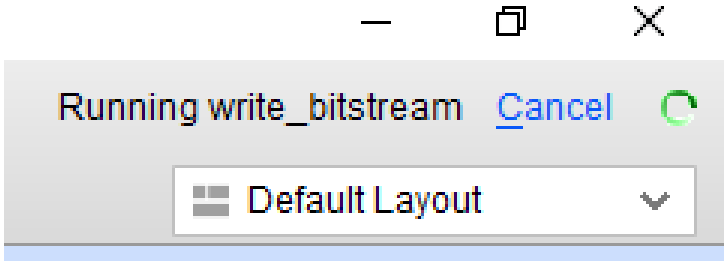


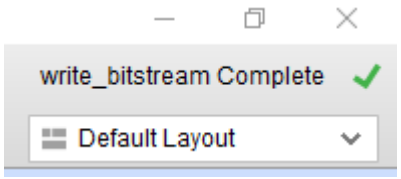




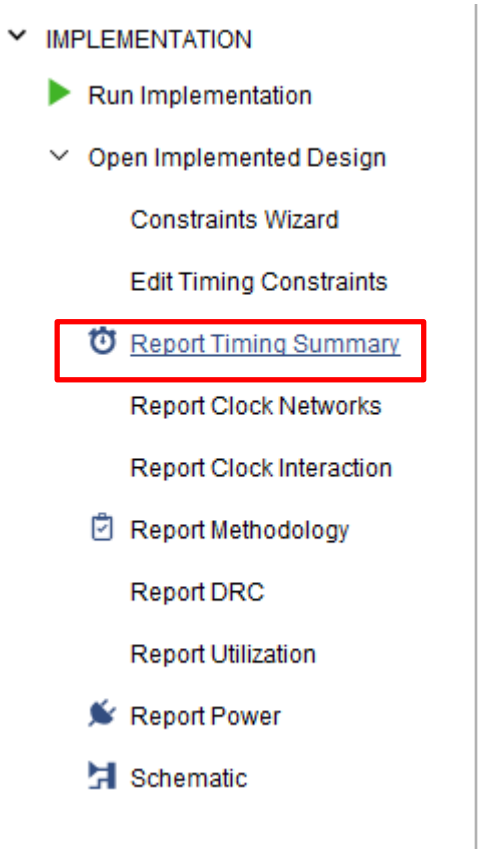
Example: Simple PL Design

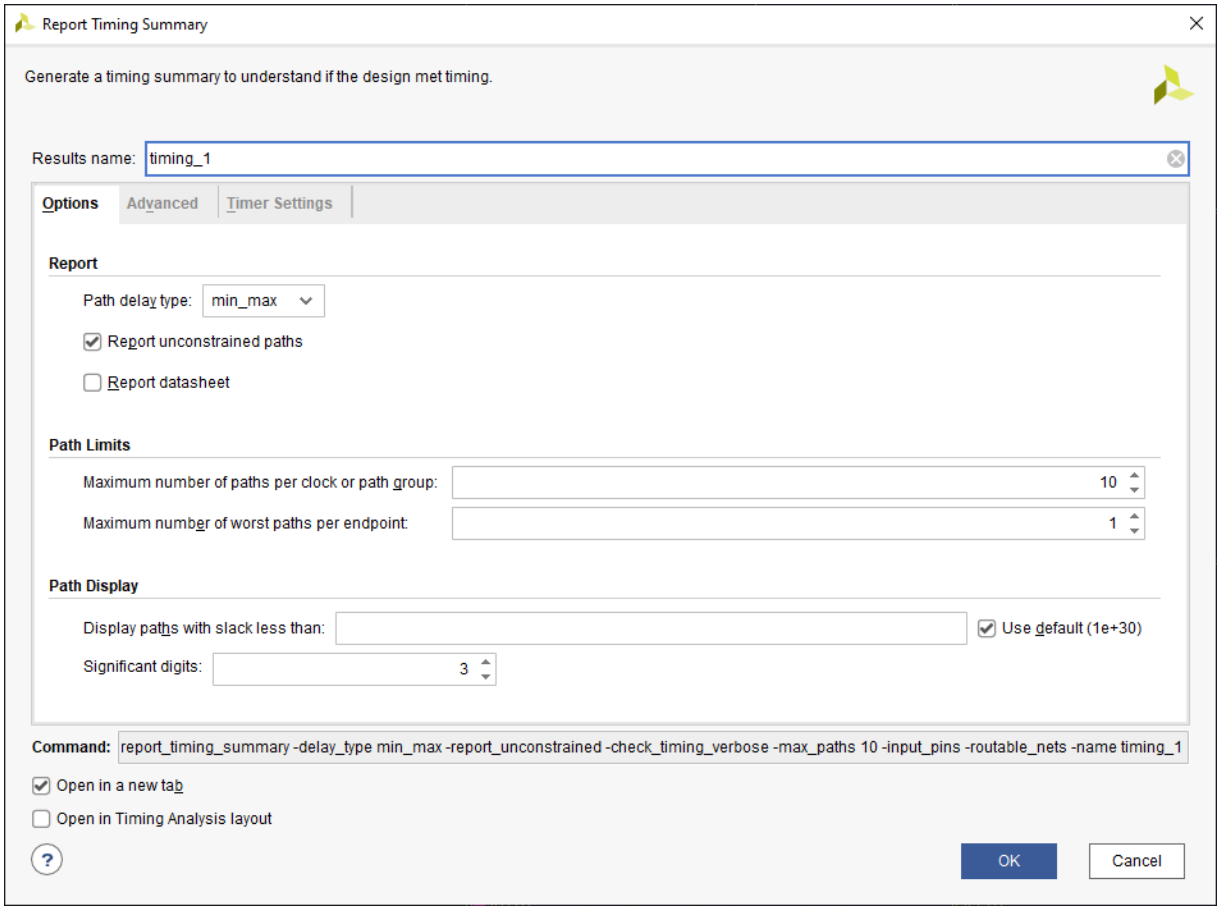




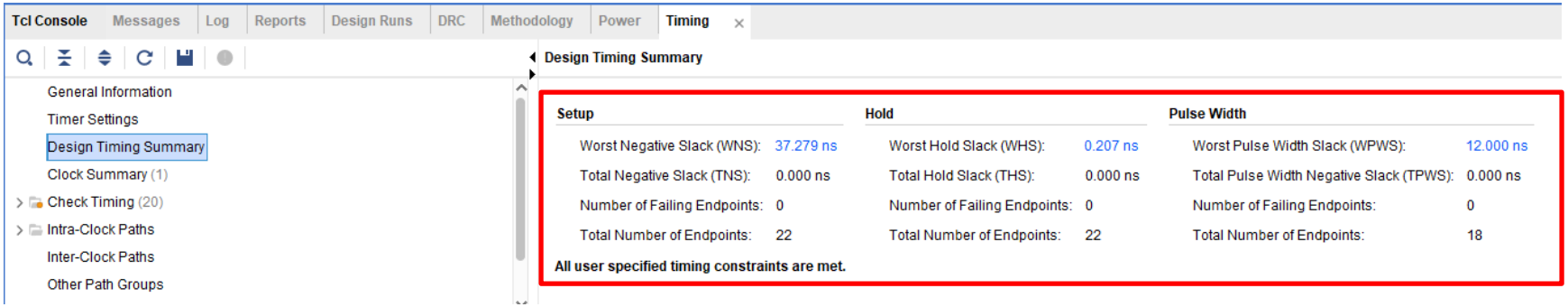


Example: Simple PL Design

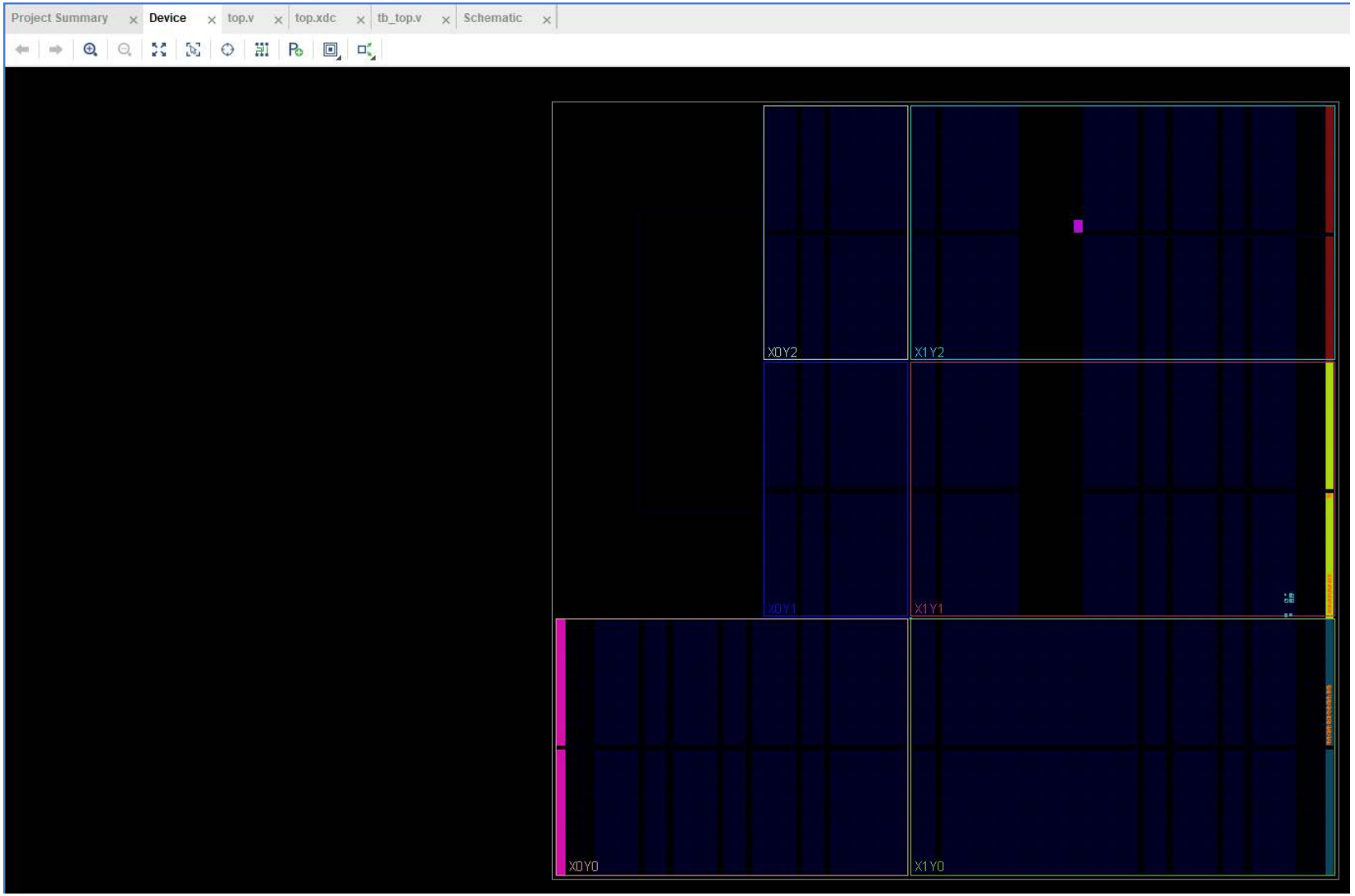




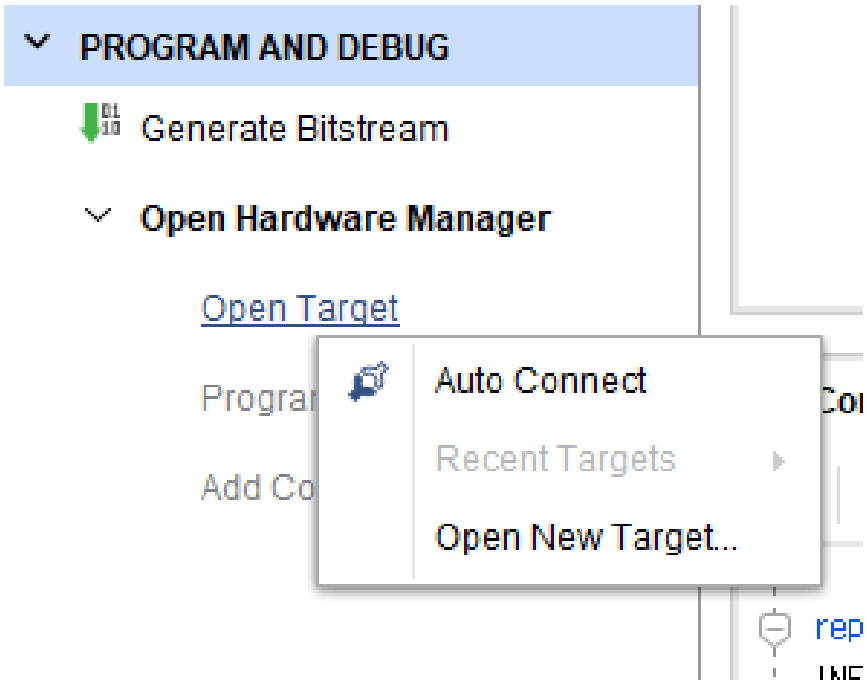
Example: Simple PL Design



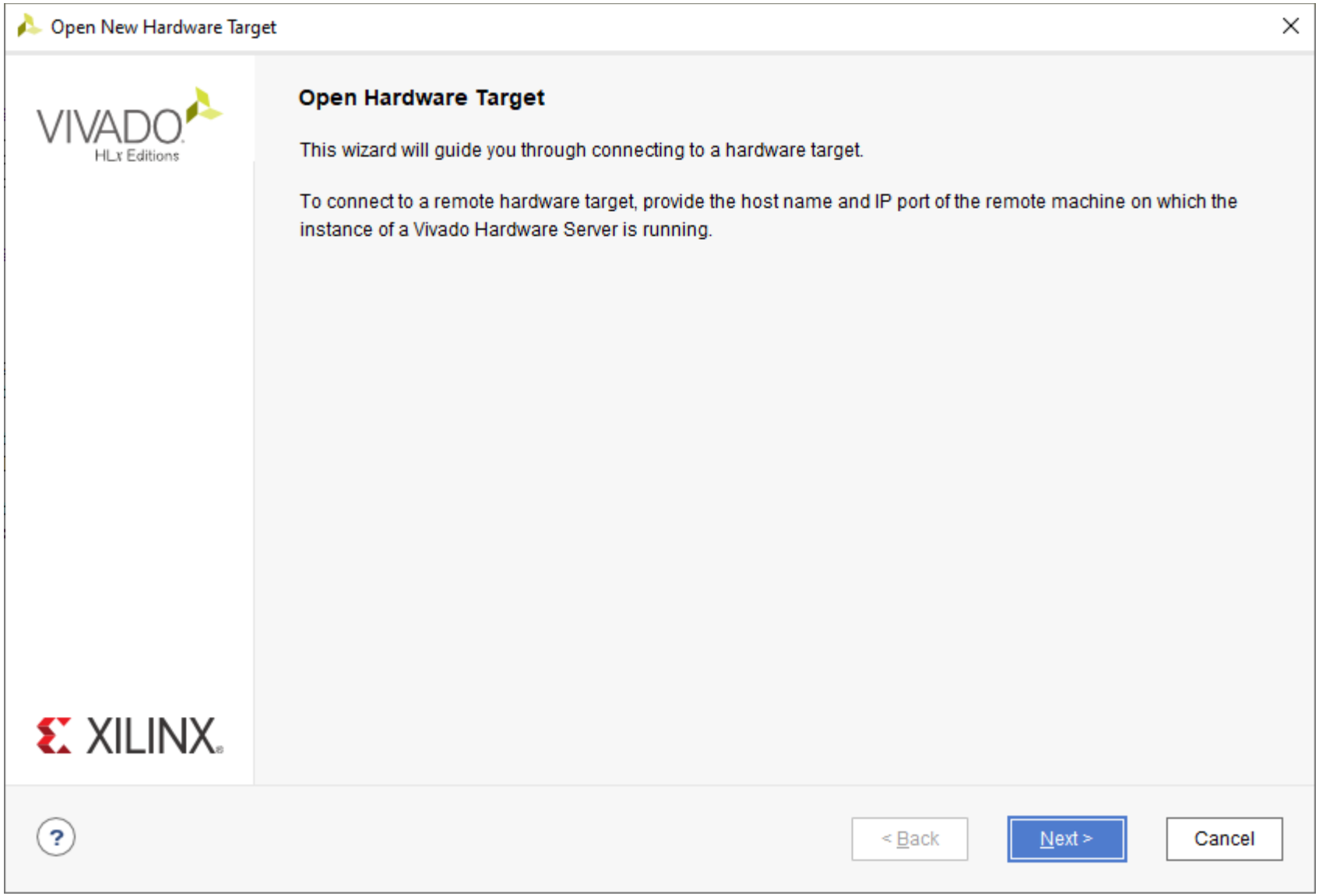
Example: Simple PL Design



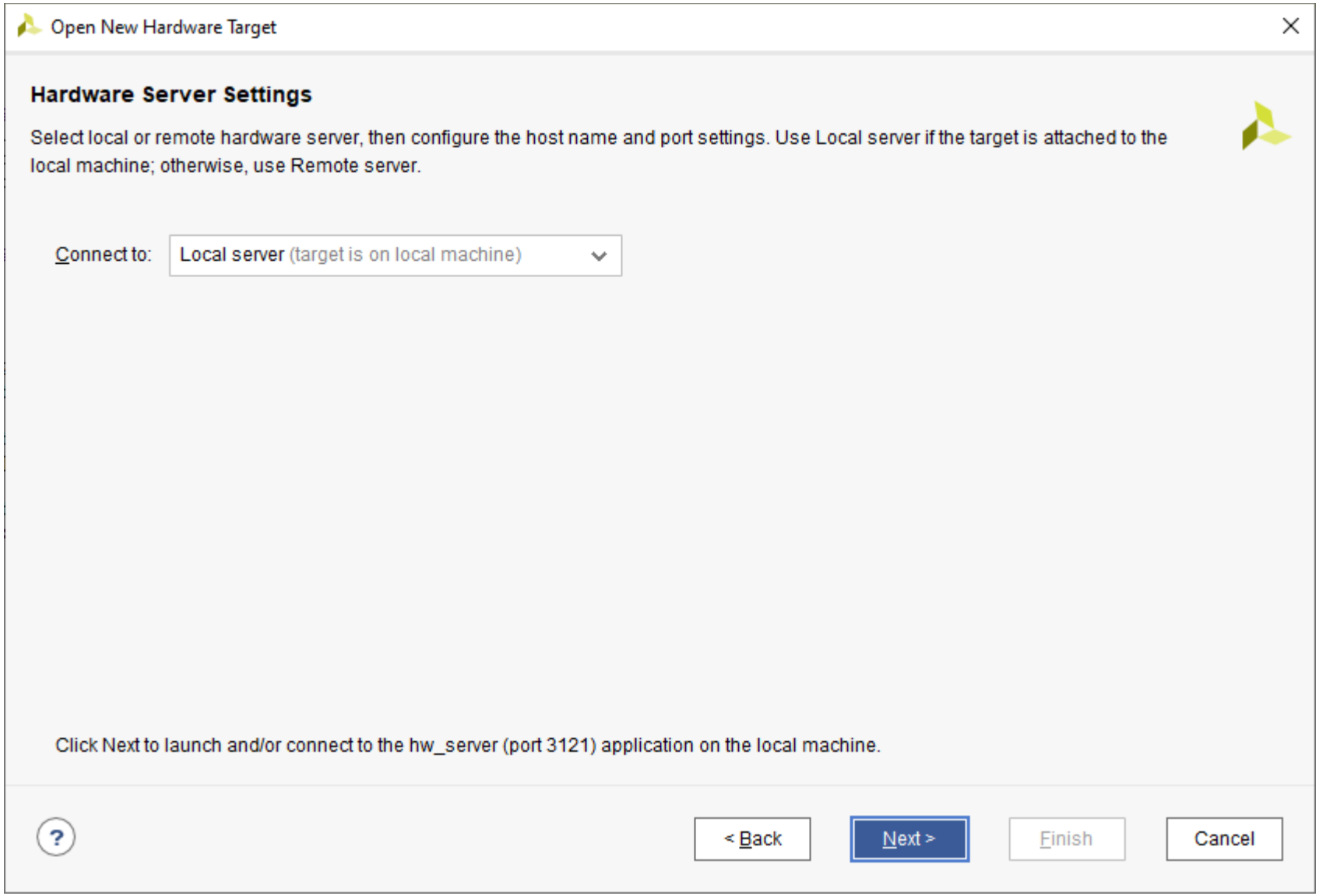
Example: Simple PL Design



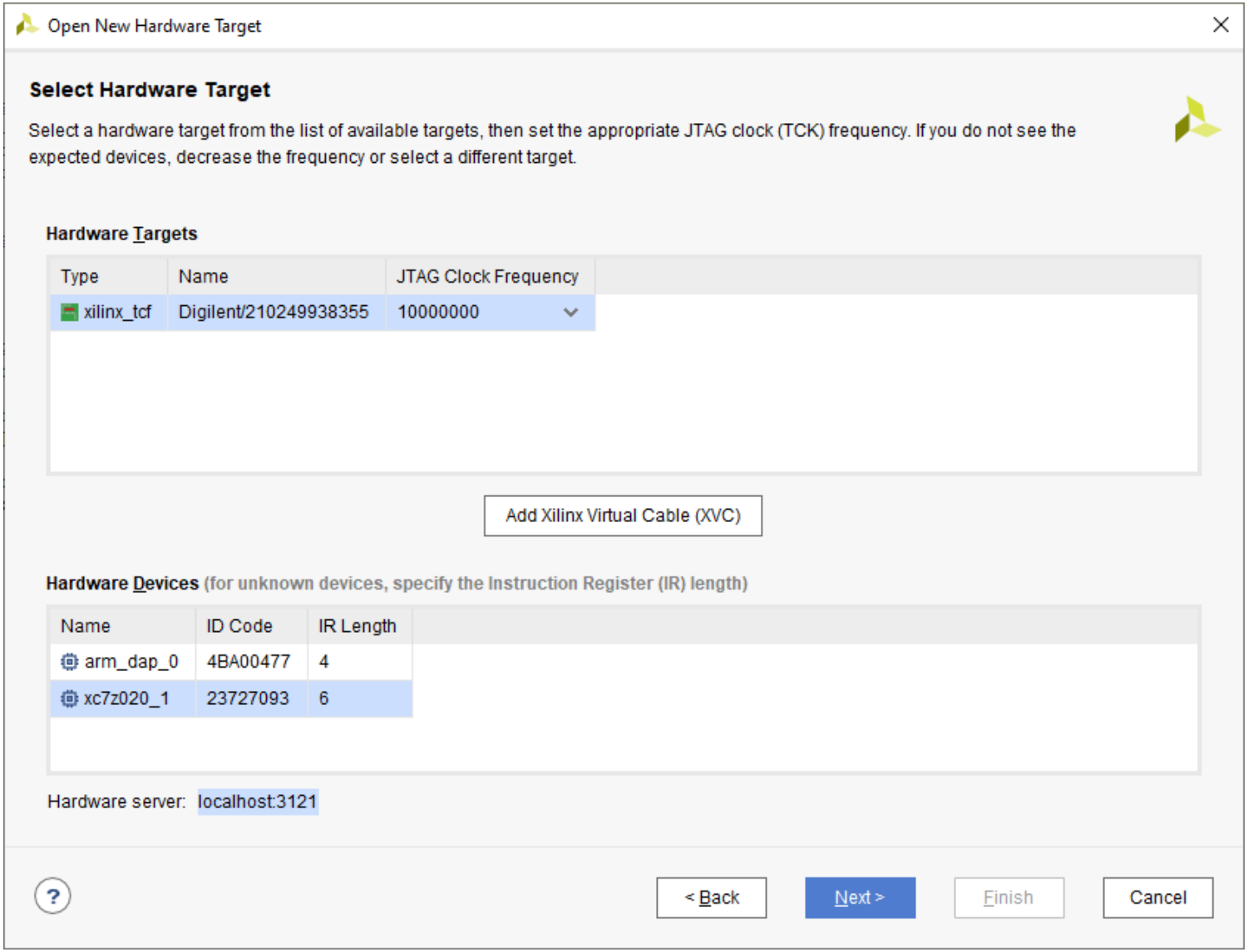
Example: Simple PL Design



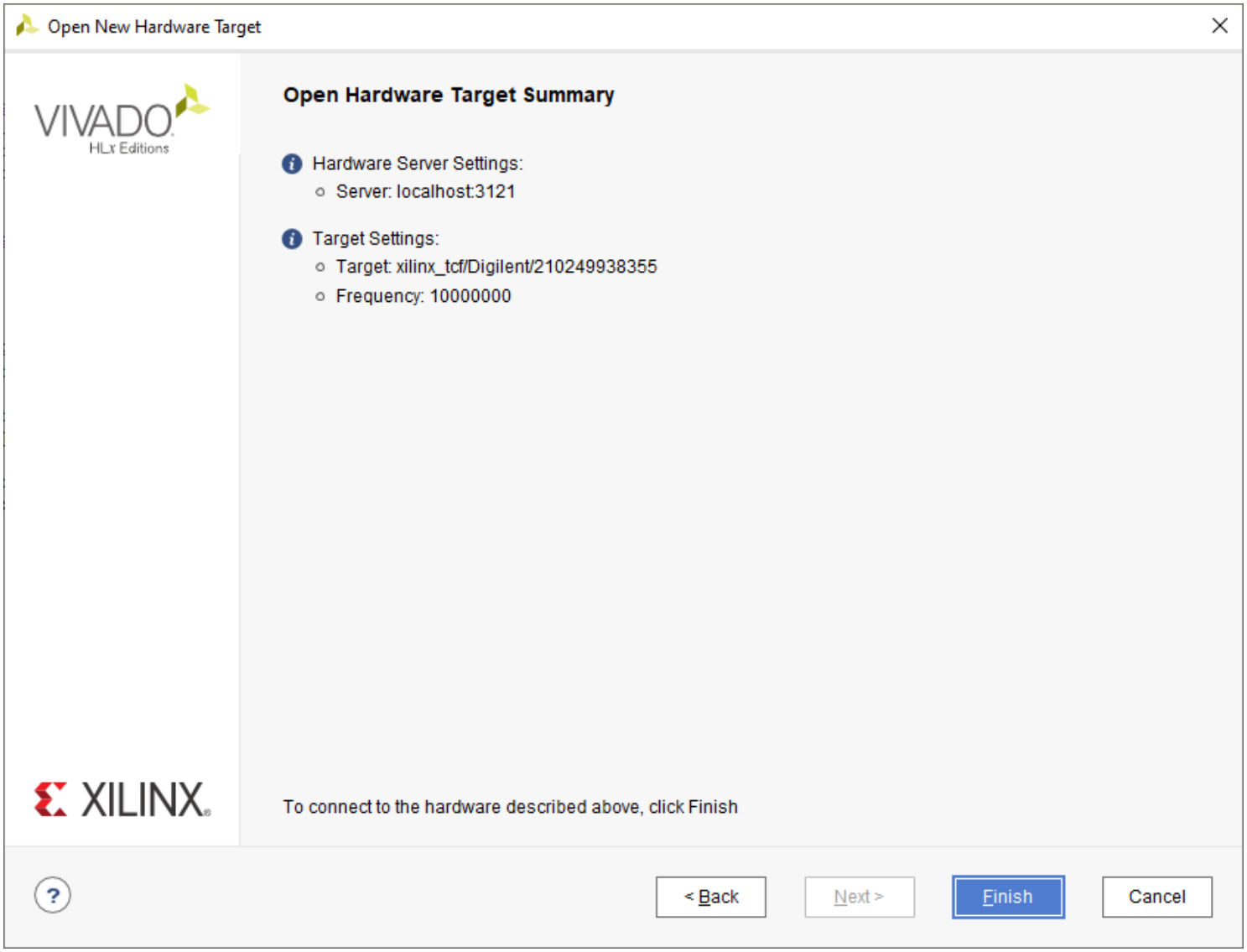
Example: Simple PL Design



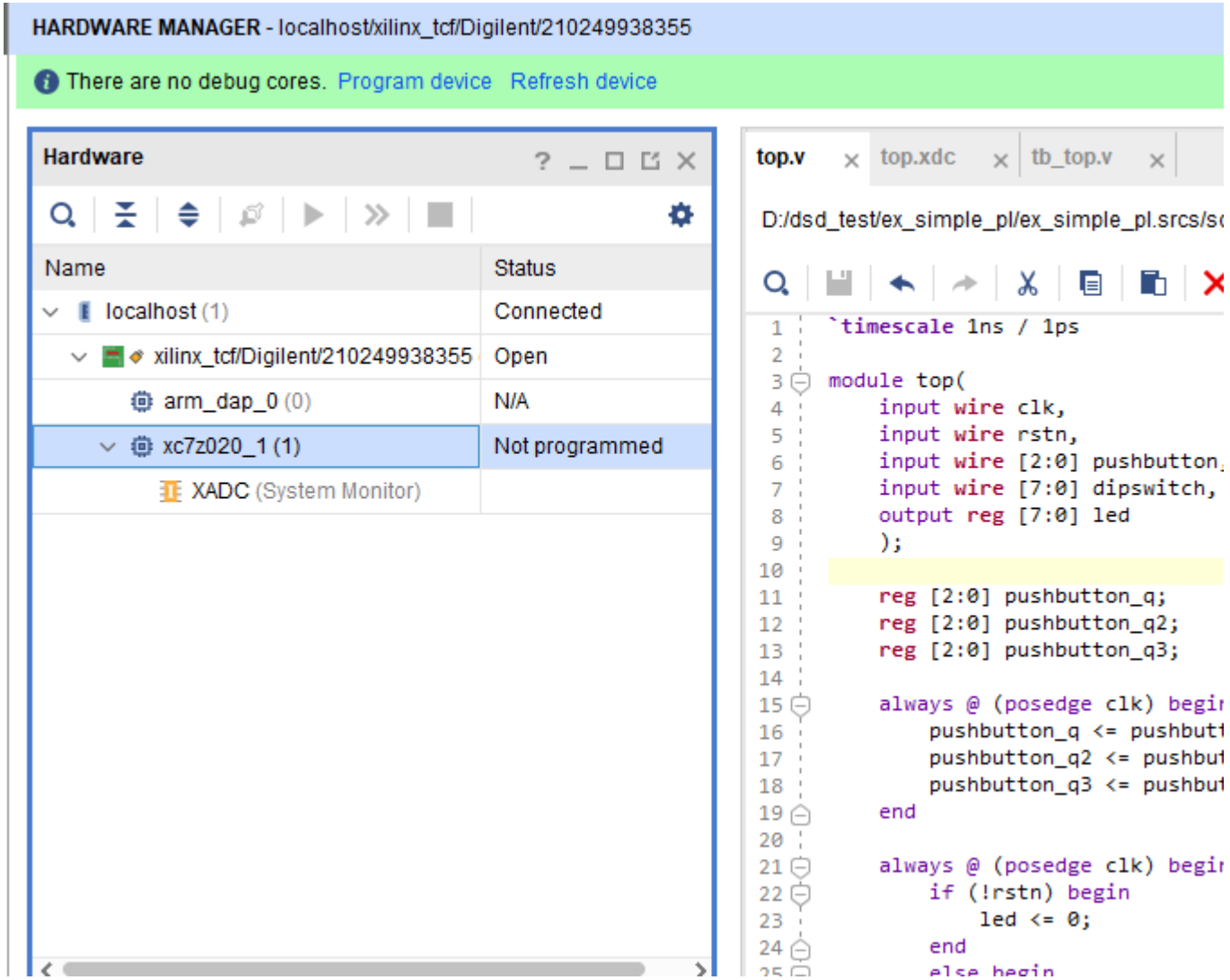
Example: Simple PL Design



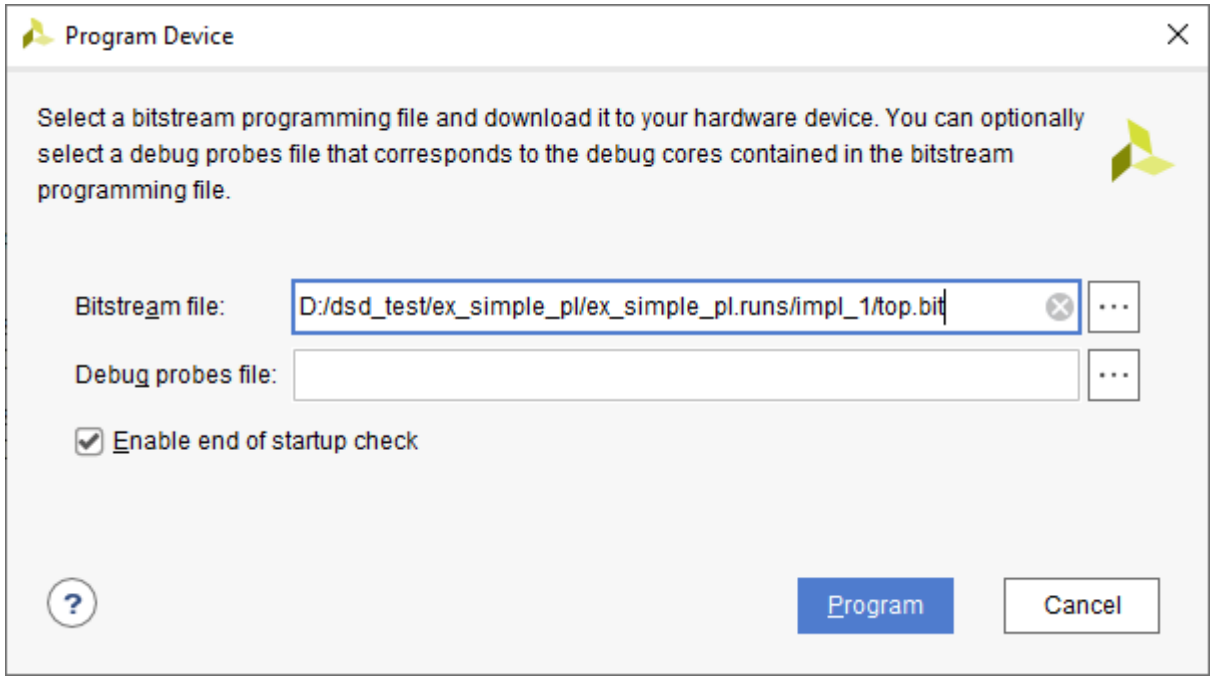
Example: Simple PL Design



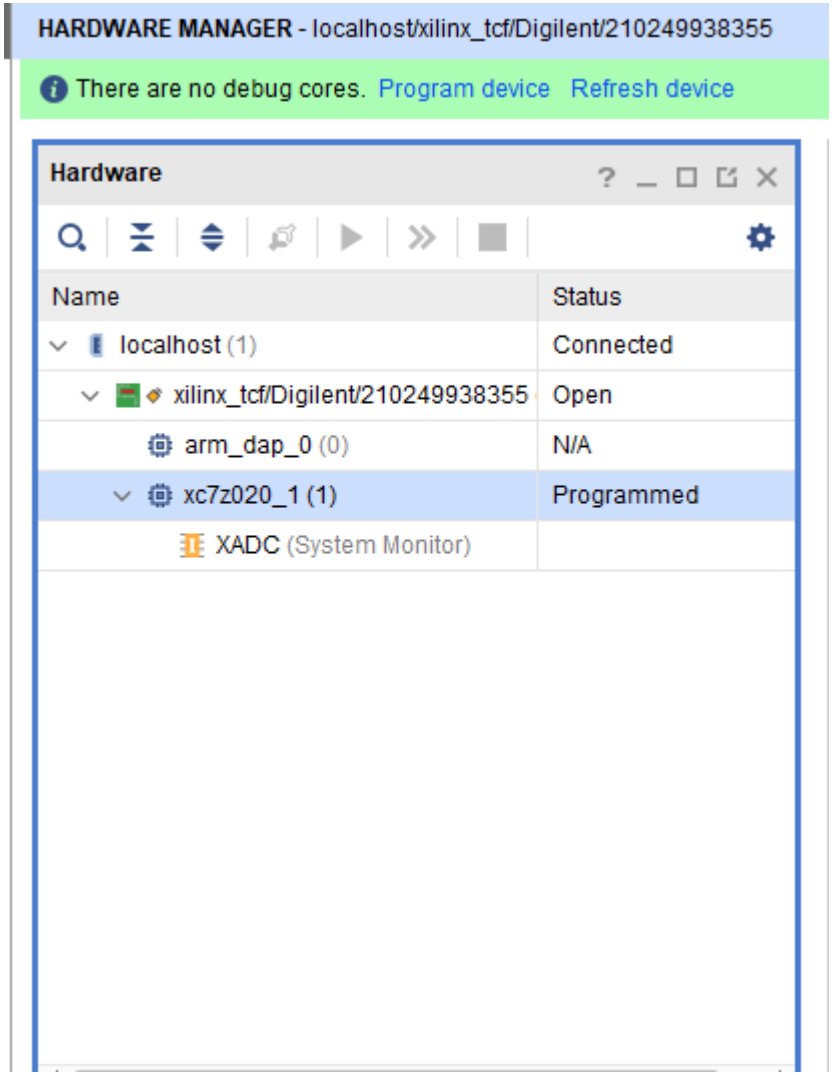
Example: Simple PL Design



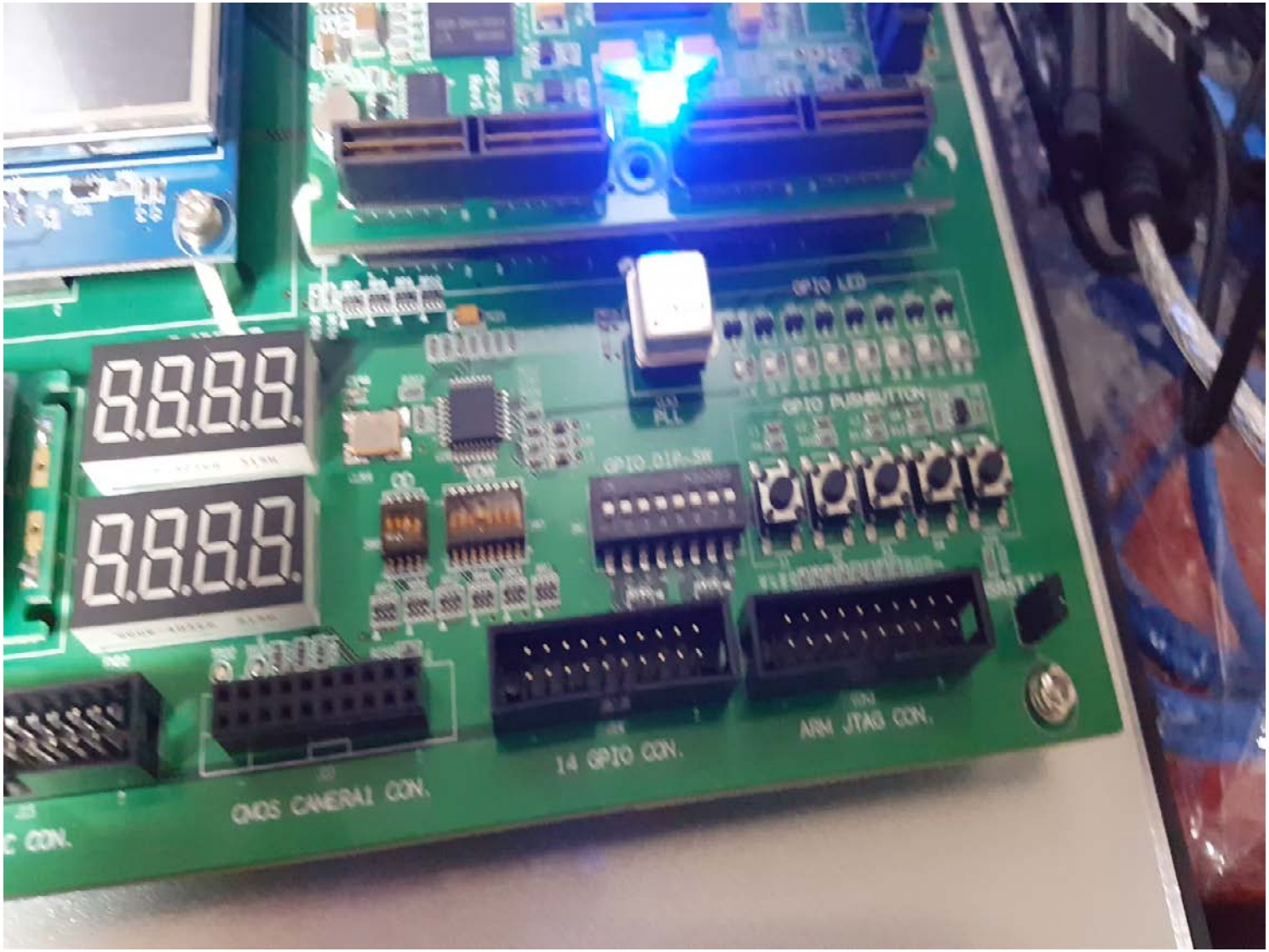
Example: Simple PL Design



Example: Simple PL Design



Example: Simple PL Design



Project – Reference code review

- About FPGA
- Design Flow
- Project explanation
- Vivado – Download
- Vivado – Create project
- Vivado – Add/Create sources
- Vivado – Simulation
- Vivado – Example: Simple PL Design
- Project – Reference code review (Video)
- Project – Vivado given sample code review (Video)
- Project – Vivado testbench examples (Video)

Project – Reference code review

```
////////////////////////////////////////////////////////////////////////////////
// open original bmp file
////////////////////////////////////////////////////////////////////////////////

FILE* fp_lenna_original;
fopen_s(&fp_lenna_original, "D:\\dsd_test\\lenna_256.bmp", "rb");

unsigned char header_info_lenna_original[54];
fread(header_info_lenna_original, sizeof(unsigned char), 54, fp_lenna_original);

unsigned int header_start_addr;
unsigned int header_width_original;
unsigned int header_height_original;

header_start_addr = *(unsigned int*)& header_info_lenna_original[10];
header_width_original = *(unsigned int*)& header_info_lenna_original[18];
header_height_original = *(unsigned int*)& header_info_lenna_original[22];

// let's check data
// header_start_address should be "54"
// header_width_original should be "128"
// header_height_original should be "128"
printf("header_start_addr: %u\n", header_start_addr);
printf("header_width_original: %u\n", header_width_original);
printf("header_height_original: %u\n", header_height_original);

unsigned int size_lenna_original;
size_lenna_original = 3 * header_width_original * header_height_original;

// load pixel data
unsigned char* data_lenna_original = (char*)malloc(sizeof(unsigned char) * size_lenna_original);
fread(data_lenna_original, sizeof(unsigned char), size_lenna_original, fp_lenna_original);
```

Project – Reference code review

```
////////////////////////////////////////////////////////////////////////////////
// lenna_256.coe memory file gen.
////////////////////////////////////////////////////////////////////////////////
FILE* fp_lenna_coe;
fopen_s(&fp_lenna_coe, "D:\\dsd_test\\lenna_256.coe", "wb");


fprintf(fp_lenna_coe, "memory_initialization_radix = 16;\n");
fprintf(fp_lenna_coe, "memory_initialization_vector=\n");
unsigned char tmp_byte_data = 0;

for (int row = 0; row < header_height_original; row++) {
    for (int col = 0; col < header_width_original; col++) {
        for (int channel = 2; channel >= 0; channel--) {
            tmp_byte_data = data_lenna_original[row * header_width_original * 3 + col * 3 + channel];
            fprintf(fp_lenna_coe, "%x", tmp_byte_data);
        }
        fprintf(fp_lenna_coe, ",\n");
    }
}
fseek(fp_lenna_coe, -2, SEEK_CUR);
fprintf(fp_lenna_coe, ";");

fclose(fp_lenna_coe);
```

Project – Reference code review

```
////////////////////////////////////////////////////////////////////////////////
// do down-sample and write down-sampled bmp file
////////////////////////////////////////////////////////////////////////////////

FILE* fp_lenna_ds;
fopen_s(&fp_lenna_ds, "D:\\dsd_test\\lenna_ds.bmp", "wb");

// make lenna_ds's header info
unsigned char header_info_lenna_ds[54];

// copy lenna_128's header info
for (int i = 0; i < 54; i++) {
    header_info_lenna_ds[i] = header_info_lenna_original[i];
}

// change width & height info
unsigned int header_width_ds;
unsigned int header_height_ds;
header_width_ds = header_width_original / 2;
header_height_ds = header_height_original / 2;
*(unsigned int*)& header_info_lenna_ds[18] = header_width_ds;
*(unsigned int*)& header_info_lenna_ds[22] = header_height_ds;

// check whether lenna_ds's header info is changed correctly
unsigned int test_tmp;
test_tmp = *(unsigned int*)& header_info_lenna_ds[18];
printf("ds_header_width: %u\n", test_tmp);
test_tmp = *(unsigned int*)& header_info_lenna_ds[22];
printf("ds_header_height: %u\n", test_tmp);
```

Project – Reference code review

```
// do simple down-sampling
// in this example, we will just select(sample) one of the 4 pixels
// you have to develop this part

unsigned int size_lenna_ds;
size_lenna_ds = 3 * header_width_ds * header_height_ds;

unsigned char* data_lenna_ds = (char*)malloc(sizeof(unsigned char) * size_lenna_ds);

int idx = 0;
for (int row = 0; row < header_height_original; row++) {
    for (int col = 0; col < header_width_original; col++) {
        if ((row % 2) == 1) {
            break;
        }
        else {
            if ((col % 2) == 0) {
                data_lenna_ds[idx + 0] = data_lenna_original[row * header_width_original * 3 + col * 3 + 0];
                data_lenna_ds[idx + 1] = data_lenna_original[row * header_width_original * 3 + col * 3 + 1];
                data_lenna_ds[idx + 2] = data_lenna_original[row * header_width_original * 3 + col * 3 + 2];
                idx = idx + 3;
            }
        }
    }
}

// write data
fwrite(header_info_lenna_ds, sizeof(unsigned char), 54, fp_lenna_ds);
fwrite(data_lenna_ds, sizeof(unsigned char), size_lenna_ds, fp_lenna_ds);
```

Project – Reference code review

```
////////////////////////////////////////////////////////////////////////////////
// do up-sample and write up-sampled bmp file
////////////////////////////////////////////////////////////////////////////////

FILE* fp_lenna_us;
fopen_s(&fp_lenna_us, "D:\\dsd_test\\lenna_us.bmp", "wb");

// make lenna_ds's header info
unsigned char header_info_lenna_us[54];

// copy lenna_128's header info
for (int i = 0; i < 54; i++) {
    header_info_lenna_us[i] = header_info_lenna_original[i];
}

// change width & height info
unsigned int header_width_us;
unsigned int header_height_us;
header_width_us = header_width_original;
header_height_us = header_height_original;
```

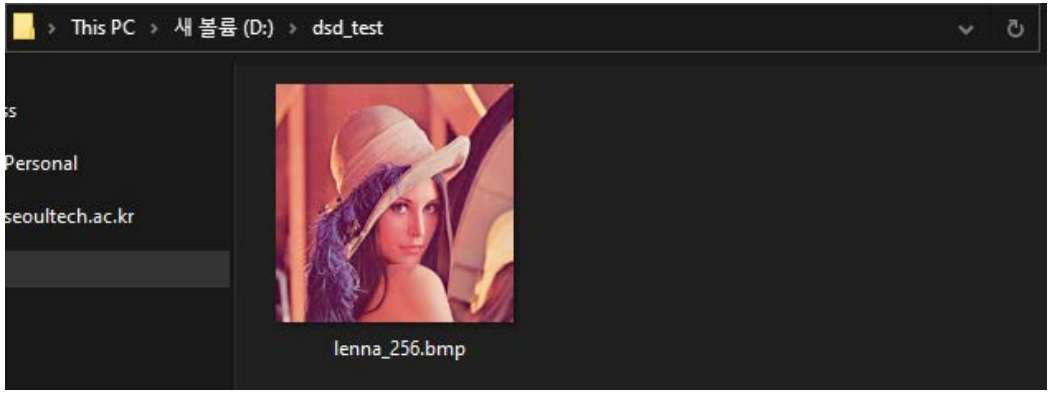
Project – Reference code review

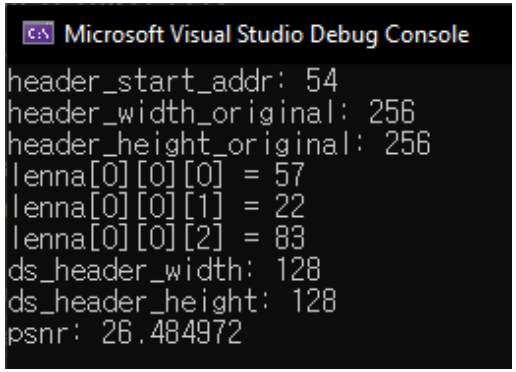
```
// do simple up-sampling
// in this example, we will just reproduct data
// you have to develop this part
unsigned int size_lenna_us;
size_lenna_us = 3 * header_width_us * header_height_us;

unsigned char* data_lenna_us = (char*)malloc(sizeof(unsigned char) * size_lenna_us);


for (int row = 0; row < header_height_ds; row++) {
    for (int col = 0; col < header_width_ds; col++) {
        for (int k = 0; k < 3; k++) {
            data_lenna_us[row * 2 * header_width_us * 3 + col * 2 * 3 + k] = data_lenna_ds[row * header_width_ds * 3 + col * 3 + k];
            data_lenna_us[row * 2 * header_width_us * 3 + col * 2 * 3 + k + 3] = data_lenna_ds[row * header_width_ds * 3 + col * 3 + k];
            data_lenna_us[(row * 2 + 1) * header_width_us * 3 + col * 2 * 3 + k] = data_lenna_ds[row * header_width_ds * 3 + col * 3 + k];
            data_lenna_us[(row * 2 + 1) * header_width_us * 3 + col * 2 * 3 + k + 3] = data_lenna_ds[row * header_width_ds * 3 + col * 3 + k];
        }
    }
}

// write data
fwrite(header_info_lenna_us, sizeof(unsigned char), 54, fp_lenna_us);
fwrite(data_lenna_us, sizeof(unsigned char), size_lenna_us, fp_lenna_us);
```

Project – Reference code review

```
////////////////////////////////////////////////////////////////////////////////
// calculate psnr
////////////////////////////////////////////////////////////////////////////////

double mse_tmp = 0;
double mse = 0;
double psnr = 0;
int size = 3 * header_height_original * header_width_original;

for (int i = 0; i < size; i++) {
    mse_tmp += pow(data_lenna_original[i] - data_lenna_us[i], 2);
}
mse = mse_tmp / size;

psnr = 20 * log10(255) - 10 * log10(mse);
printf("psnr: %lf\n", psnr);
```

Project – Reference code review





Project – Vivado given sample code review

- About FPGA
- Design Flow
- Project explanation
- Vivado – Download
- Vivado – Create project
- Vivado – Add/Create sources
- Vivado – Simulation
- Vivado – Example: Simple PL Design
- Project – Reference code review (Video)
- Project – Vivado given sample code review (Video)
- Project – Vivado testbench examples (Video)

Project – Vivado given sample code review

Project – Vivado given sample code review

```
1  `timescale 1ns / 1ps
2
3  module ds(
4      // global signals
5      input wire clk,
6      input wire rstn,
7      // system signals
8      input wire sys_en,
9      // brom interface
10     output reg [15:0] rom_addr,
11     input wire [23:0] rom_data,
12     output wire rom_en,
13     // down-sampled data & control signals
14     output wire [23:0] ds_data,
15     output wire ds_done,
16     output wire ds_wr_en
17 );
18
19     // Notice: to read data is delayed 1-clock cycle
20     // you should be aware of it!
21     // design your own ds code below!
22     // you can add any port or signals
23
24
25 endmodule
```

Project – Vivado given sample code review

```
1  `timescale 1ns/1ps
2
3  module us(
4       // global signals
5      input wire clk,
6      input wire rstn,
7      // system signals
8      input wire sys_en,
9      // down-sampled data & control signal
10     input wire [23:0] ds_data,
11     input wire ds_vld,
12     // bram interface
13     output reg [15:0] ram_wr_addr,
14     output wire [31:0] ram_wr_data,
15     output wire ram_en,
16     output wire ram_rst,
17     // ram write enable byte
18     output wire [3:0] ram_web,
19     // up-sampled control signals
20     output wire us_done,
21     output wire us_wr_en
22 );
23
24     // Notice: if you want to write data to bram,
25     // then set "ram_web" as "4'b1111"
26     // otherewise(i.e. you don't want to write data)
27     // set "ram_web" as "4'b0000"
28     // design your own ds code below!
29     // you can add any port or signals
30
31
32
33 endmodule
```

Project – Vivado given sample code review

```
1  `timescale 1ns/1ps
2
3  module top_ds_us(
4      // global signals
5      input wire clk,
6      input wire rstn,
7      // system signals
8      input wire sys_en,
9      // brom interface
10     output wire [15:0] rom_addr,
11     input wire [23:0] rom_data,
12     output wire rom_en,
13     // bram interface
14     output wire [15:0] ram_wr_addr,
15     output wire [31:0] ram_wr_data,
16     output wire ram_en,
17     output wire ram_rst,
18     output wire [3:0] ram_web,
19     // down-sampled data & control signals
20     output wire [23:0] ds_out,
21     output wire ds_done,
22     output wire ds_wr_en,
23     // up-sampled data & control signals
24     output wire [23:0] us_out,
25     output wire us_done,
26     output wire us_wr_en
27 );
28
29     assign us_out = ram_wr_data[23:0];
30
31     wire ds_vld;
32     assign ds_vld = ds_wr_en;
33
```

```
34     // down-sample module instantiation
35     ds  ds_inst(
36         // global signals
37         .clk(clk),
38         .rstn(rstn),
39         // system signals
40         .sys_en(sys_en),
41         // brom interface
42         .rom_addr(rom_addr),
43         .rom_data(rom_data),
44         .rom_en(rom_en),
45         // down-sampled data & control signals
46         .ds_data(ds_out),
47         .ds_done(ds_done),
48         .ds_wr_en(ds_wr_en)
49     );
50
51     //------------------------------------------------------------
52     // you can add control unit(module) here
53     //------------------------------------------------------------
54
55     // up-sample module instantiation
56     us  us_inst(
57         // global signals
58         .clk(clk),
59         .rstn(rstn),
60         // system signals
61         .sys_en(sys_en),
62         // down-sampled data & control signal
63         .ds_data(ds_out),
64         .ds_vld(ds_vld),
65         // bram interface
66         .ram_wr_addr(ram_wr_addr),
67         .ram_wr_data(ram_wr_data),
68         // port to check whether data is written correctly
69         //.ram_rd_data(ram_rd_data),
70         .ram_en(ram_en),
71         .ram_rst(ram_rst),
72         .ram_web(ram_web),
73          // up-sampled control signals
74         .us_done(us_done),
75         .us_wr_en(us_wr_en)
76     );
77
78 endmodule
```

Project – Vivado given sample code review

```
3  module top_ds_us_test(
4      // global signals
5      input wire clk,
6      input wire rstn,
7      // system signals
8      input wire sys_en,
9      // down-sampled data & control signals
10     output wire [23:0] ds_out,
11     output wire ds_done,
12     output wire ds_wr_en,
13     // up-sampled data & control signals
14     output wire [23:0] us_out,
15     output wire [15:0] us_addr,
16     output wire us_done,
17     output wire us_wr_en,
18     // test ram data
19     input wire ram_result_test_en,
20     input wire [15:0] ram_result_test_addr,
21     output wire [31:0] ram_result_test_data
22     );
23
24     wire [15:0] rom_addr;
25     wire [23:0] rom_data;
26     wire rom_en;
27
28     wire [15:0] ram_wr_addr;
29     wire [31:0] ram_wr_data;
30     wire ram_en;
31     wire ram_rst;
32     wire [3:0] ram_web;
33
34     wire ram_rd_en;
35     wire [18:0] ram_rd_addr;
36     wire [31:0] ram_rd_data;
37
38     assign us_addr = ram_wr_addr;
```

```
40     // block rom instanitation
41     block_rom_lenna256    rom_inst(
42         .clka(clk),
43         .ena(rom_en),
44         .addra(rom_addr),
45         .douta(rom_data)
46     );
47
48     top_ds_us   ds_us_inst(
49         // global signals
50         .clk(clk),
51         .rstn(rstn),
52         // system signals
53         .sys_en(sys_en),
54         // brom interface
55         .rom_addr(rom_addr),
56         .rom_data(rom_data),
57         .rom_en(rom_en),
58         // bram interface
59         .ram_wr_addr(ram_wr_addr),
60         .ram_wr_data(ram_wr_data),
61         .ram_en(ram_en),
62         .ram_rst(ram_rst),
63         .ram_web(ram_web),
64         // down-sampled data & control signals
65         .ds_out(ds_out),
66         .ds_done(ds_done),
67         .ds_wr_en(ds_wr_en),
68         // up-sampled data & control signals
69         .us_out(us_out),
70         .us_done(us_done),
71         .us_wr_en(us_wr_en)
72     );
```

Project – Vivado given sample code review

```
74      block_ram_dual_port_test  ram_inst(
75          // port A, cpu will read bram data through this port
76          .clka(clk),
77          .ena(ram_result_test_en),
78          .wea(),
79          .addra(ram_result_test_addr),
80          .dina(),
81          .douta(ram_result_test_data),
82          // port B, your output will be written through this port
83          .clkb(clk),
84          .rstb(ram_rst),
85          .enb(ram_en),
86          .web(ram_web),
87          .addrb(ram_wr_addr),
88          .dinb(ram_wr_data),
89          .doutb(),
90          .rsta_busy(),
91          .rstb_busy()
92      );
93
94  endmodule
```

Project – Vivado testbench examples

- About FPGA
- Design Flow
- Project explanation
- Vivado – Download
- Vivado – Create project
- Vivado – Add/Create sources
- Vivado – Simulation
- Vivado – Example: Simple PL Design
- Project – Reference code review (Video)
- Project – Vivado given sample code review (Video)
- Project – Vivado testbench examples (Video)

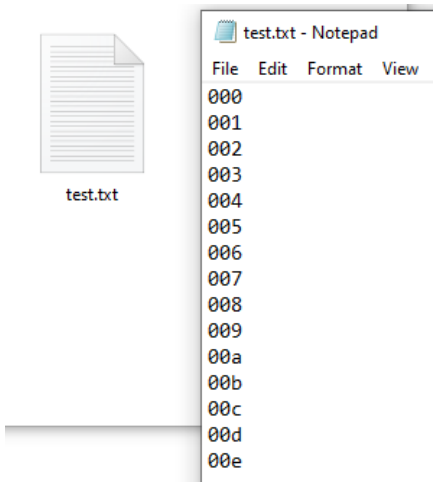
Project – Vivado testbench examples

- Let's learn to open file and write data

```
1  `timescale 1ns / 1ps
2
3  module tb_file_open_ex();
4      parameter wr_file_name = "D://dsd_test//test.txt";
5      integer fp_wr;
6
7      reg clk;
8      reg file_wr_en;
9      reg [9:0] addr;
10     wire end_of_frame_wr;
11
12     //-----------------------------------------------
13     // generate clock
14     //-----------------------------------------------
15     initial begin
16         clk = 0;
17     end
18     always begin
19         #5 clk = ~clk;
20     end
21
22     //-----------------------------------------------
23     // set file_write_enable as 1
24     //-----------------------------------------------
25     initial begin
26                 file_wr_en = 0;
27         #100    file_wr_en = 1;
28     end
29
```

Project – Vivado testbench examples

```
30      //-----------------------------------------------
31      // set file_write_enable as 0,
32      // if file_write operation reaches end of frame
33      //-----------------------------------------------
34      always @(*) begin
35          if (end_of_frame_wr == 1) begin
36              file_wr_en = 0;
37              $fclose(fp_wr);
38              $stop;
39          end
40      end
41      //-----------------------------------------------
42      // increase address according to file_write_enable
43      //-----------------------------------------------
44      initial begin
45          addr = 0;
46      end
47      always @(posedge clk) begin
48          if (file_wr_en) begin
49              addr <= addr + 1;
50          end
51      end
```




```
52      //-----------------------------------------------
53      // in this example, if address == 15,
54      // then stop file write operation
55      //-----------------------------------------------
56      assign end_of_frame_wr = (addr == 15) ? 1 : 0;
57
58      //-----------------------------------------------
59      // file open and write
60      //-----------------------------------------------
61      initial begin
62          fp_wr = $fopen(wr_file_name, "wb");
63          if (!fp_wr) begin
64              $display("write bmp file open error\n");
65              $finish;
66          end
67          $display("%s file opend\n", wr_file_name);
68       end
69
70      always @(posedge clk) begin
71          if (file_wr_en) begin
72              $fwrite(fp_wr, "%x\n", addr);
73          end
74      end
75
76
77  endmodule
```

Project – Vivado testbench examples

- Let's open bmp file and copy.

```
1  `timescale 1ns/1ps
2
3  module tb_file_copy_ex();
4      //----------------------------------------------------------------
5      // parameter declaration
6      //----------------------------------------------------------------
7      parameter rd_file_name = "D://dsd_test//lenna_256.bmp";
8      parameter wr_file_name = "D://dsd_test//lanna_copied.bmp";
9      parameter img_size = (256 * 256 * 3);
10
11     integer fp_rd, fp_wr;
12
13     reg [7:0] header_rd [53:0];
14     reg [7:0] header_wr [53:0];
15
16     reg [7:0] img_data_rd [(img_size-1):0];
17     reg [7:0] img_data_wr [(img_size-1):0];
18
19     reg [7:0] tmp_byte_data;
20     integer r, i, j, k;
21
```

Project – Vivado testbench examples

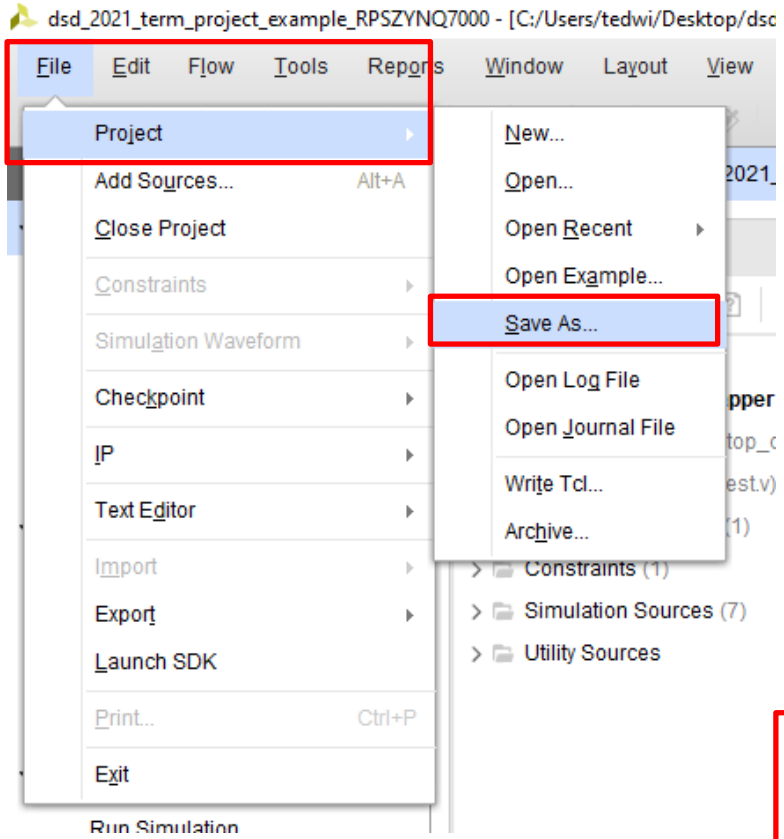
```
22      initial begin
23          //-----------------------------------------------
24          // file open and read original image
25          //-----------------------------------------------
26          fp_rd = $fopen(rd_file_name, "rb");
27          if (!fp_rd) begin
28              $display("read bmp file open error\n");
29              $finish;
30          end
31          $display("%s file opend\n", rd_file_name);
32
33          // read header information
34          for (i = 0; i < 54; i = i + 1) begin
35              #1 r = $fread(tmp_byte_data, fp_rd);
36              header_rd[i] = tmp_byte_data;
37          end
38
39          // read image data
40          for (i = 0; i < 196608; i = i + 1) begin
41              #1 r = $fread(tmp_byte_data, fp_rd);
42              img_data_rd[i] = tmp_byte_data;
43          end
```

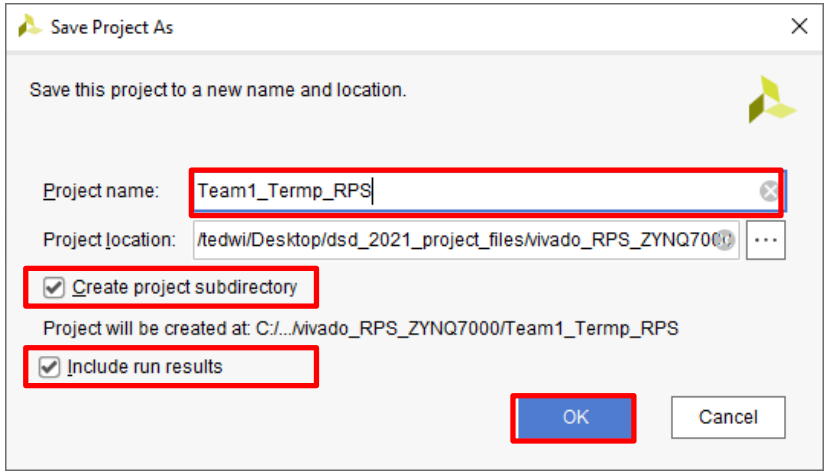
lanna_copied.bmp

lenna_256.bmp

```
45          //-----------------------------------------------
46          // copy data
47          //-----------------------------------------------
48          for (i = 0; i < 54; i = i + 1) begin
49              header_wr[i] = header_rd[i];
50          end
51          for (i = 0; i < 196608; i = i + 1) begin
52              img_data_wr[i] = img_data_rd[i];
53          end
54
55          //-----------------------------------------------
56          // file open and write copied image
57          //-----------------------------------------------
58          fp_wr = $fopen(wr_file_name, "wb");
59          if (!fp_wr) begin
60              $display("write bmp file open error\n");
61              $finish;
62          end
63          $display("%s file opend\n", wr_file_name);
64
65          // write header information
66          for (i = 0; i < 54; i = i + 1) begin
67              #1 $fwrite(fp_wr, "%c", header_wr[i]);
68          end
69
70          // write copied data
71          for (i = 0; i < 196608; i = i + 1) begin
72              $fwrite(fp_wr, "%c", img_data_rd[i]);
73          end
74
75          $stop;
76          $fclose(fp_rd);
77          $fclose(fp_wr);
78          $stop;
79      end
80
81  endmodule
```

Vivado project file



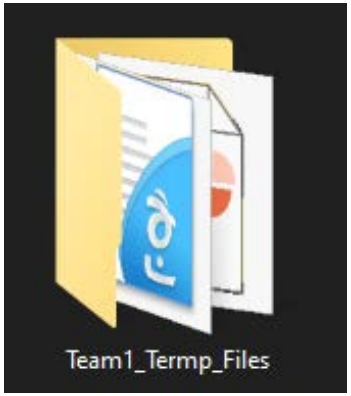


Project Name: Team{Number}_Termp_{Board Name}

Number: 1 ~ 10
Board Name: "RPS" or "ZYBO"

Ex1: Team1_Termp_RPS
Ex2: Team2_Termp_ZYBO

Submission Form



Team{number}_Termp_Files.zip
- Team{number}_Termp_Reference
- Team{number}_Termp_{board name}
- Team{number}_Project_Presentation
- Team{number}_Project_Final_Report