

Practice 3

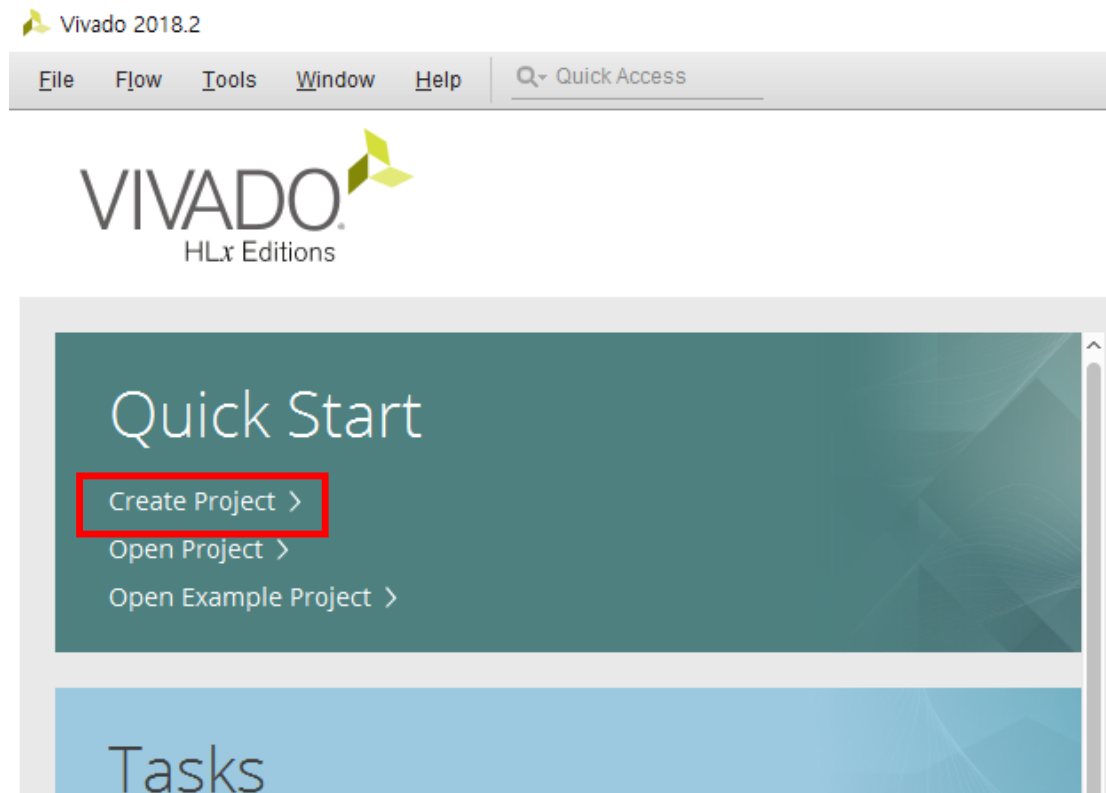
- How to use Vivado & Basic syntax

Computing Memory Architecture Lab.

Vivado Tutorial

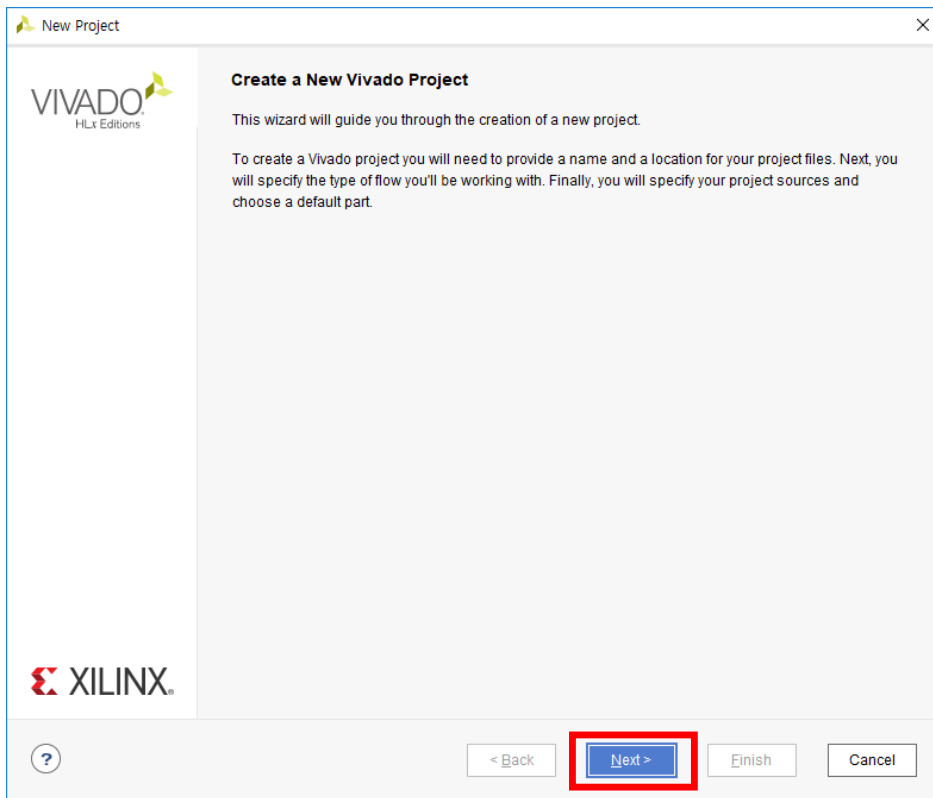
Vivado project creation

- Create New Project



Vivado project creation

- Enter project name and location



New Project

Create a New Vivado Project

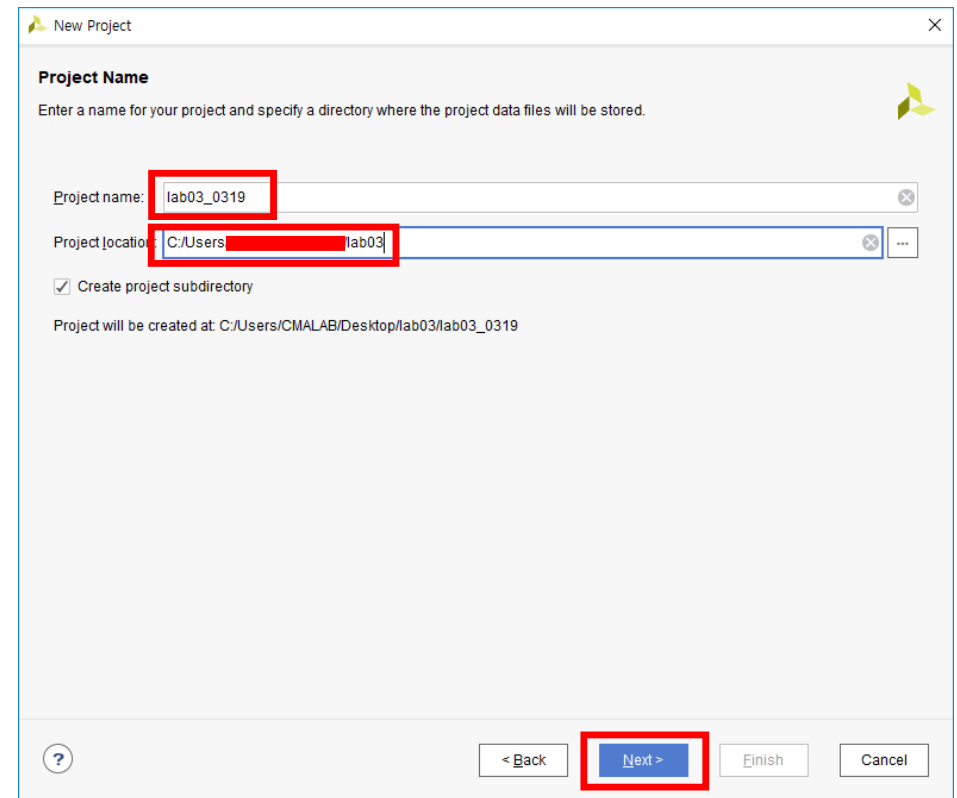
This wizard will guide you through the creation of a new project.

To create a Vivado project you will need to provide a name and a location for your project files. Next, you will specify the type of flow you'll be working with. Finally, you will specify your project sources and choose a default part.

VIVADO[®]
HLS Editions

XILINX[®]

< Back **Next >** Finish Cancel



New Project

Project Name

Enter a name for your project and specify a directory where the project data files will be stored.

Project name: lab03_0319

Project location: C:/Users/[redacted]/lab03

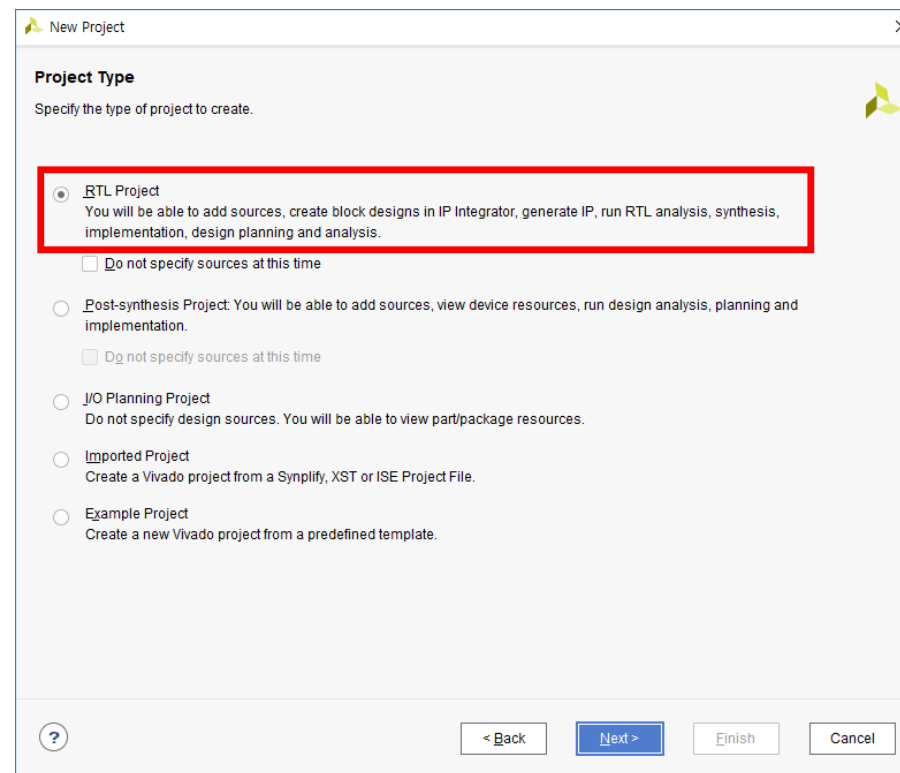
☒ Create project subdirectory

Project will be created at: C:/Users/CMALAB/Desktop/lab03/lab03_0319

? < Back **Next >** Finish Cancel

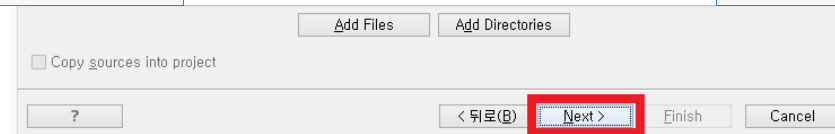
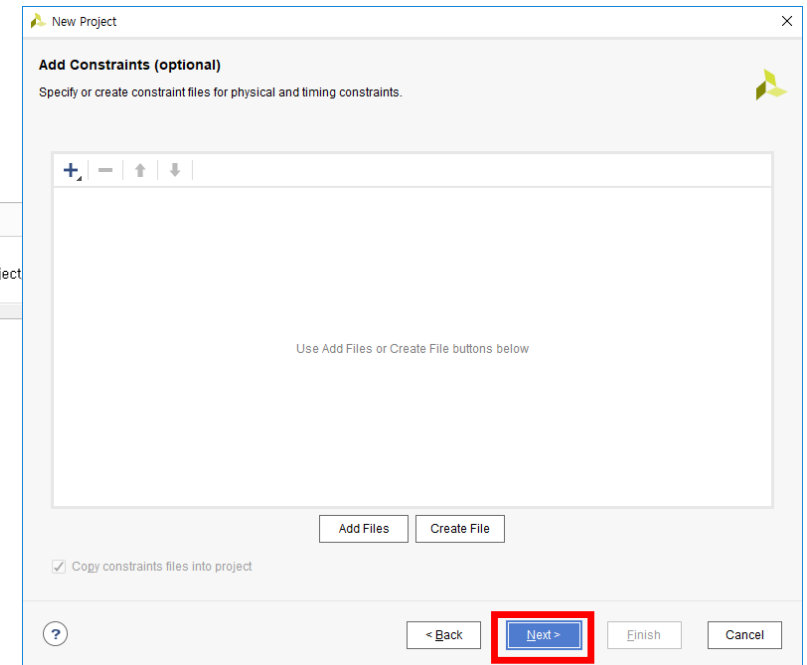
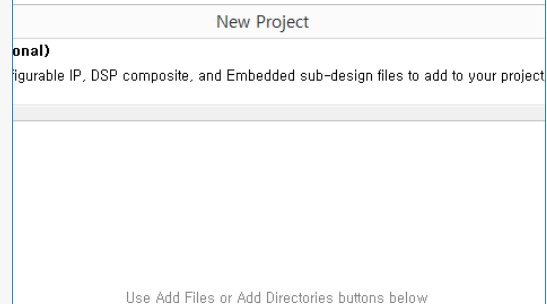
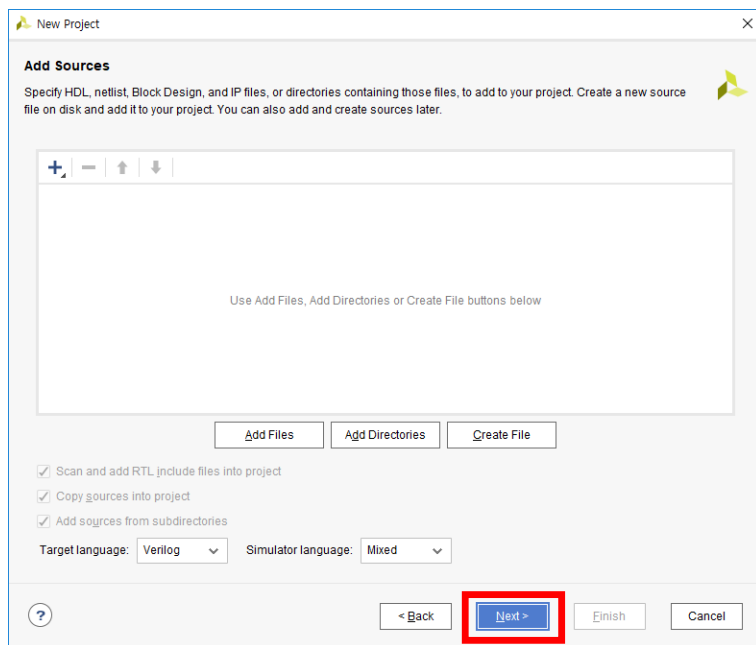
Vivado project creation

- Select project type
 - RTL Project



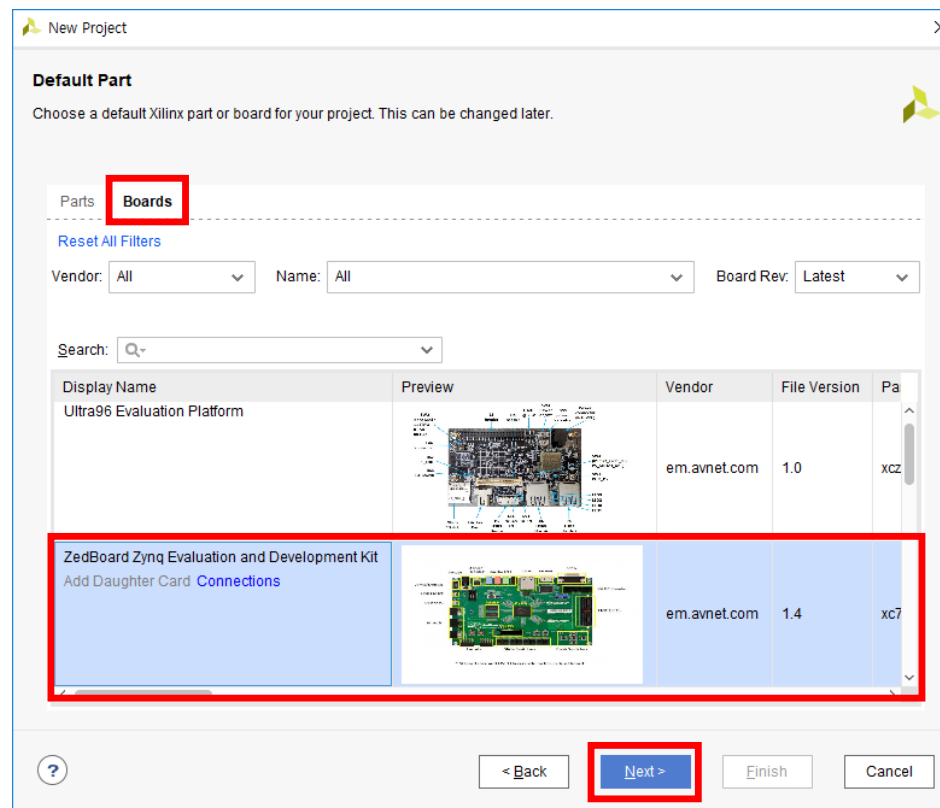
Vivado project creation

- (Optional) add sources or IPs, constraints



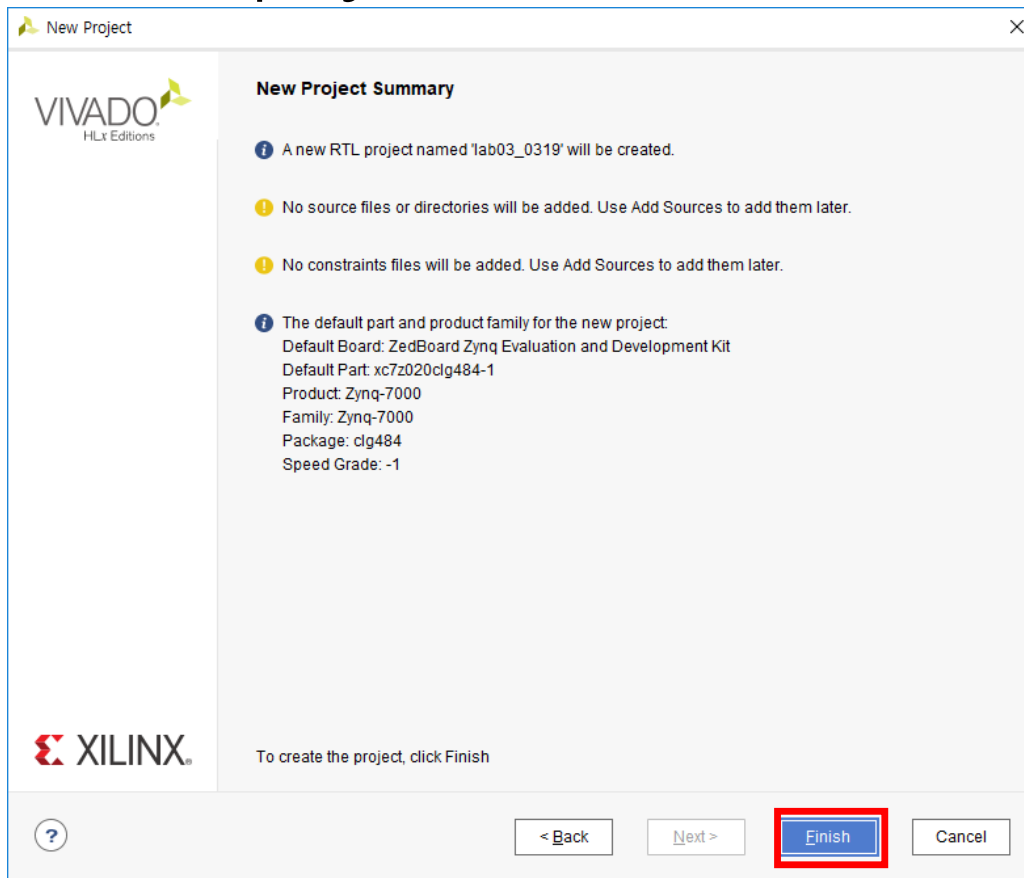
Vivado project creation

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

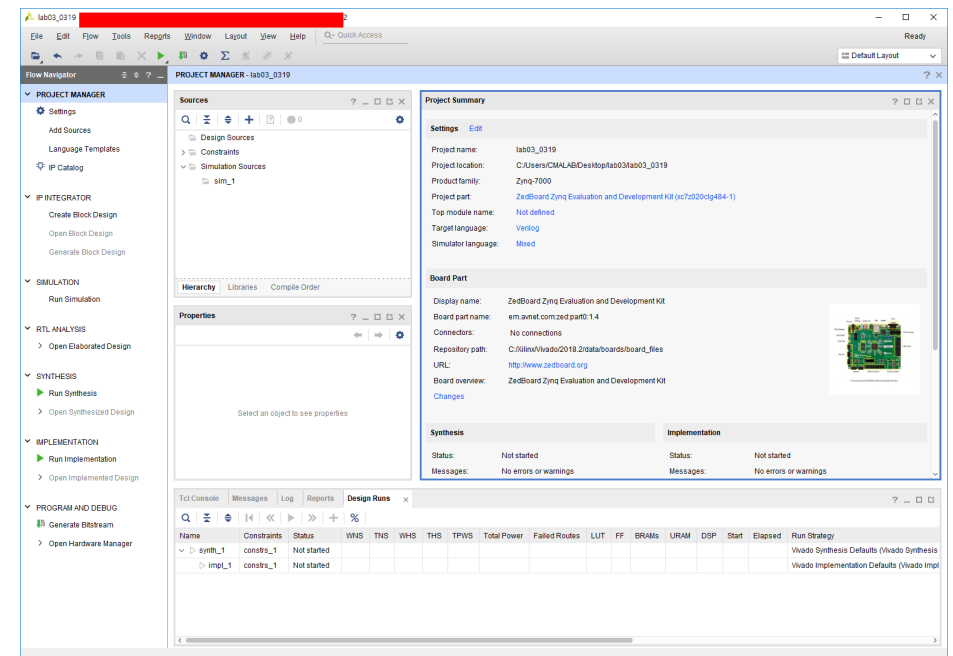


Vivado project creation

■ Finish project creation

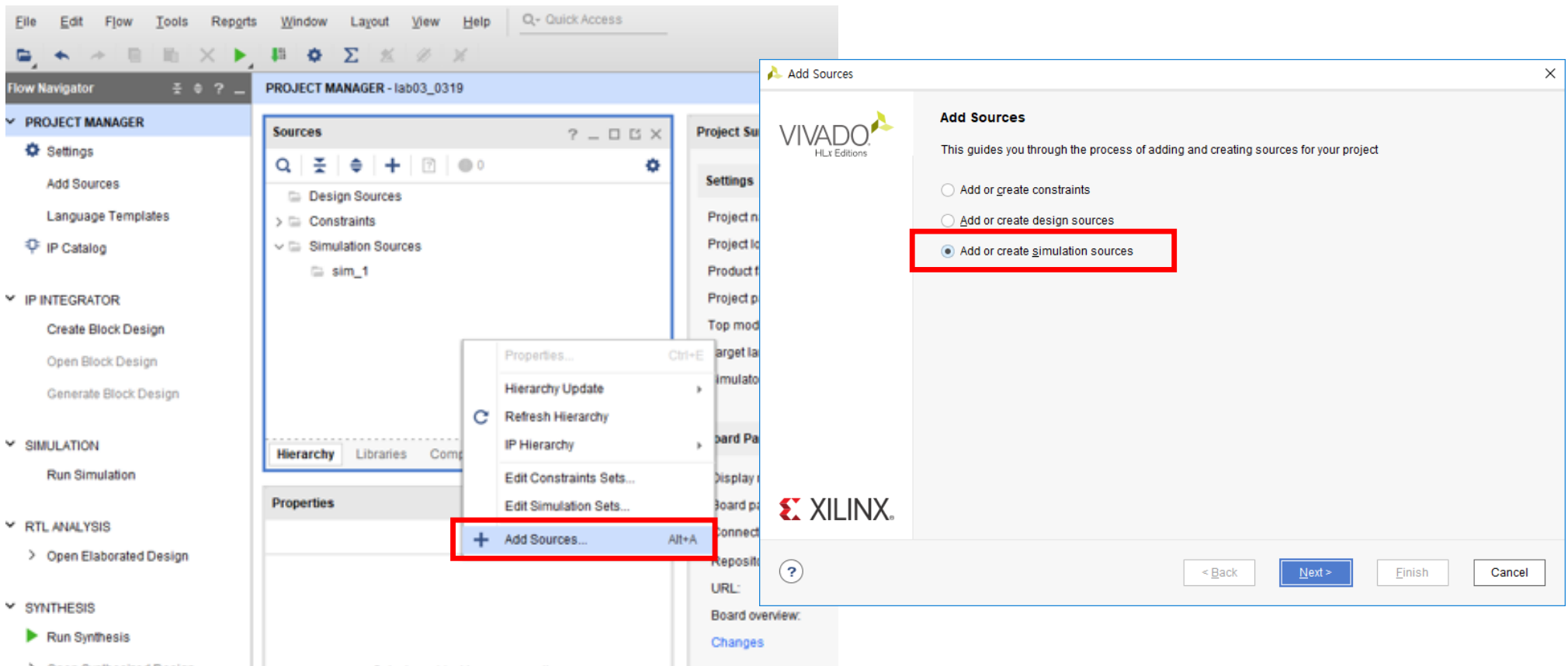


Main page



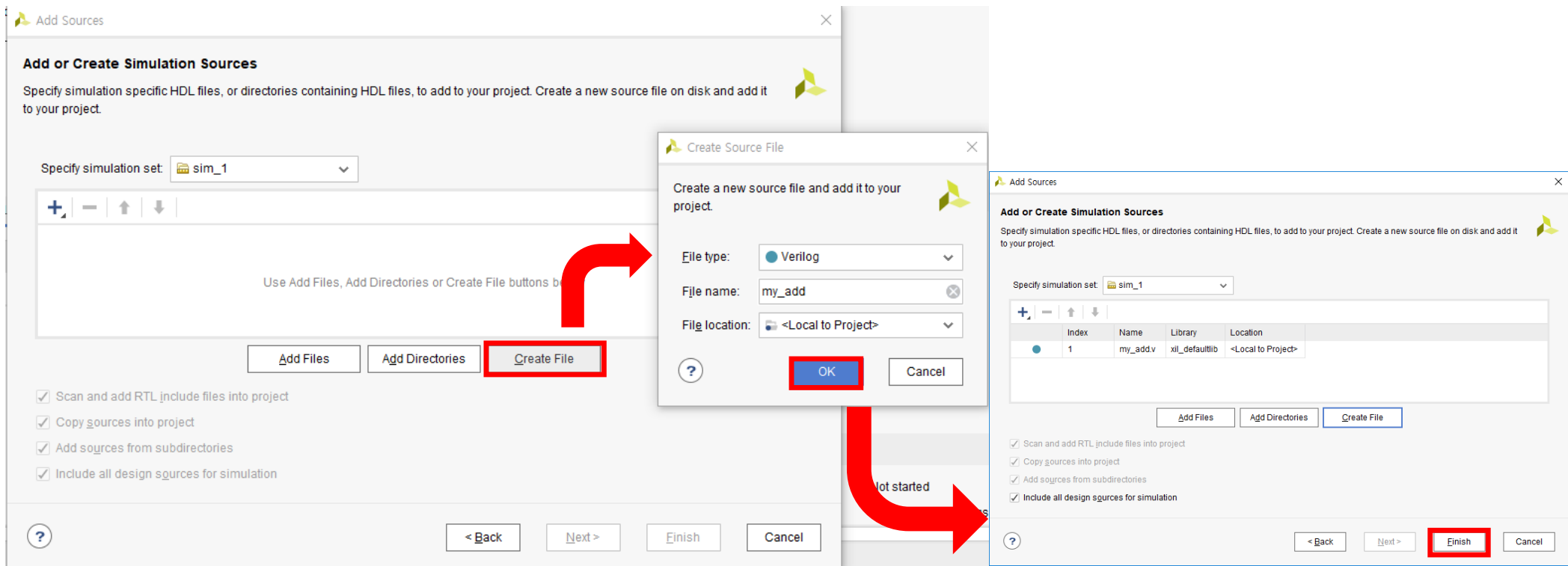
Create simulation sources

■ Create simulation source



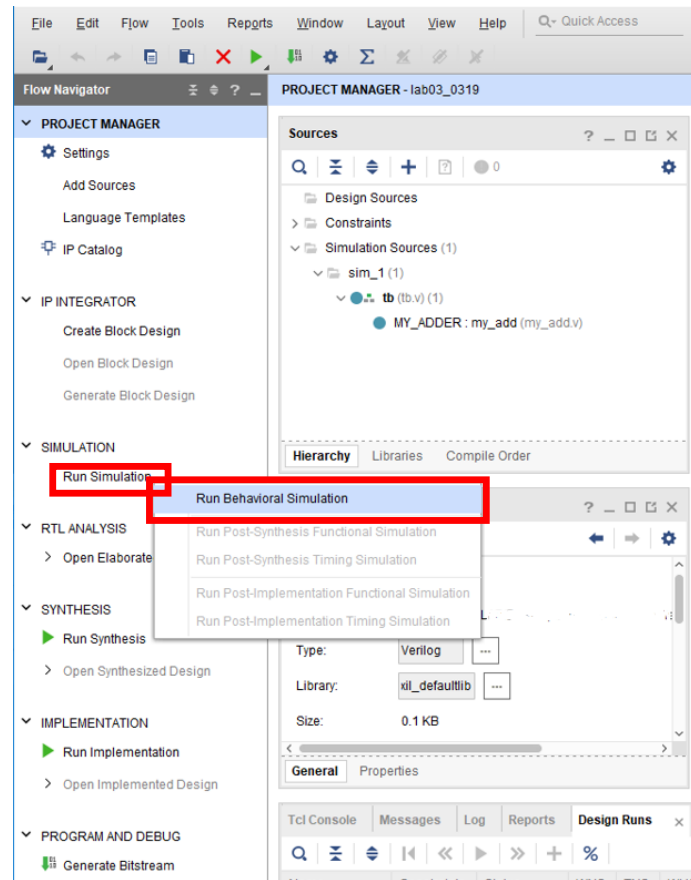
Create simulation sources

- Create simulation source
 - Modules and testbenches



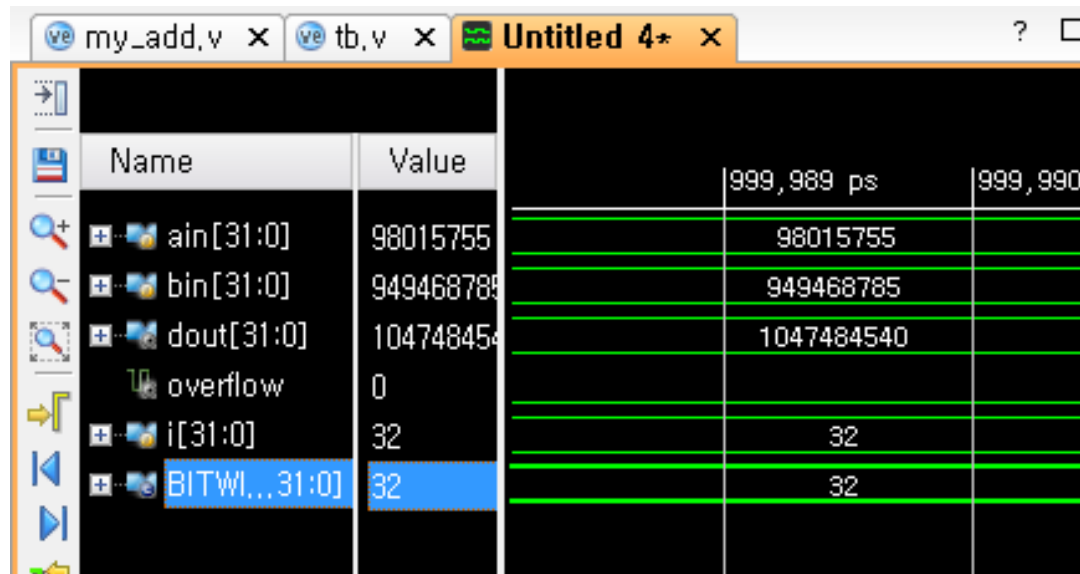
Simulation

- Run Simulation -> Run Behavioral Simulation



Simulation results

- Check result

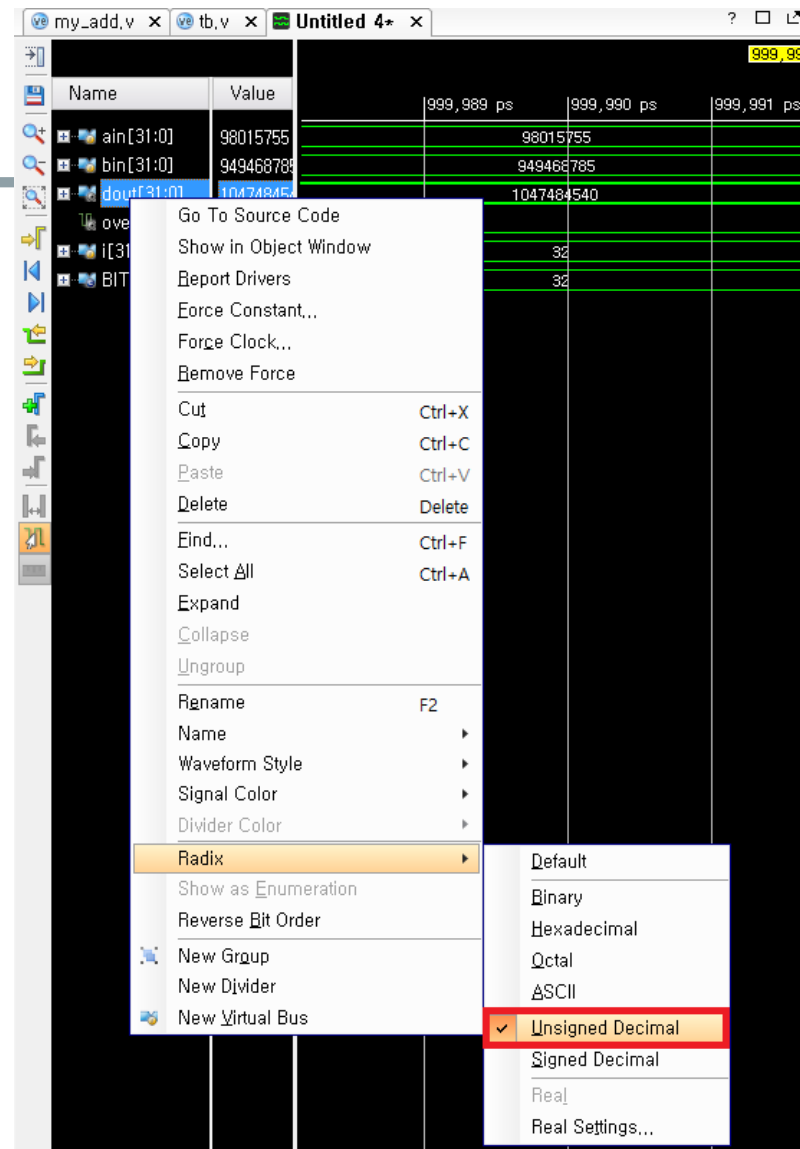


The screenshot shows a simulation tool interface with three tabs: 'my_add.v', 'tb.v', and 'Untitled 4*'. The 'Untitled 4*' tab is active, displaying a table of variable values and a waveform. The table has columns for 'Name', 'Value', and two time points: '999,989 ps' and '999,990 ps'. The variables listed are 'ain[31:0]', 'bin[31:0]', 'dout[31:0]', 'overflow', 'i[31:0]', and 'BITWL... 31:0'. The values for 'ain[31:0]', 'bin[31:0]', 'dout[31:0]', 'i[31:0]', and 'BITWL... 31:0' are 98015755, 949468785, 1047484540, 32, and 32 respectively. The 'overflow' variable has a value of 0. The waveform shows green horizontal lines for each variable across the time points.

Name	Value	999,989 ps	999,990 ps
ain[31:0]	98015755	98015755	
bin[31:0]	949468785	949468785	
dout[31:0]	1047484540	1047484540	
overflow	0		
i[31:0]	32	32	
BITWL... 31:0	32	32	

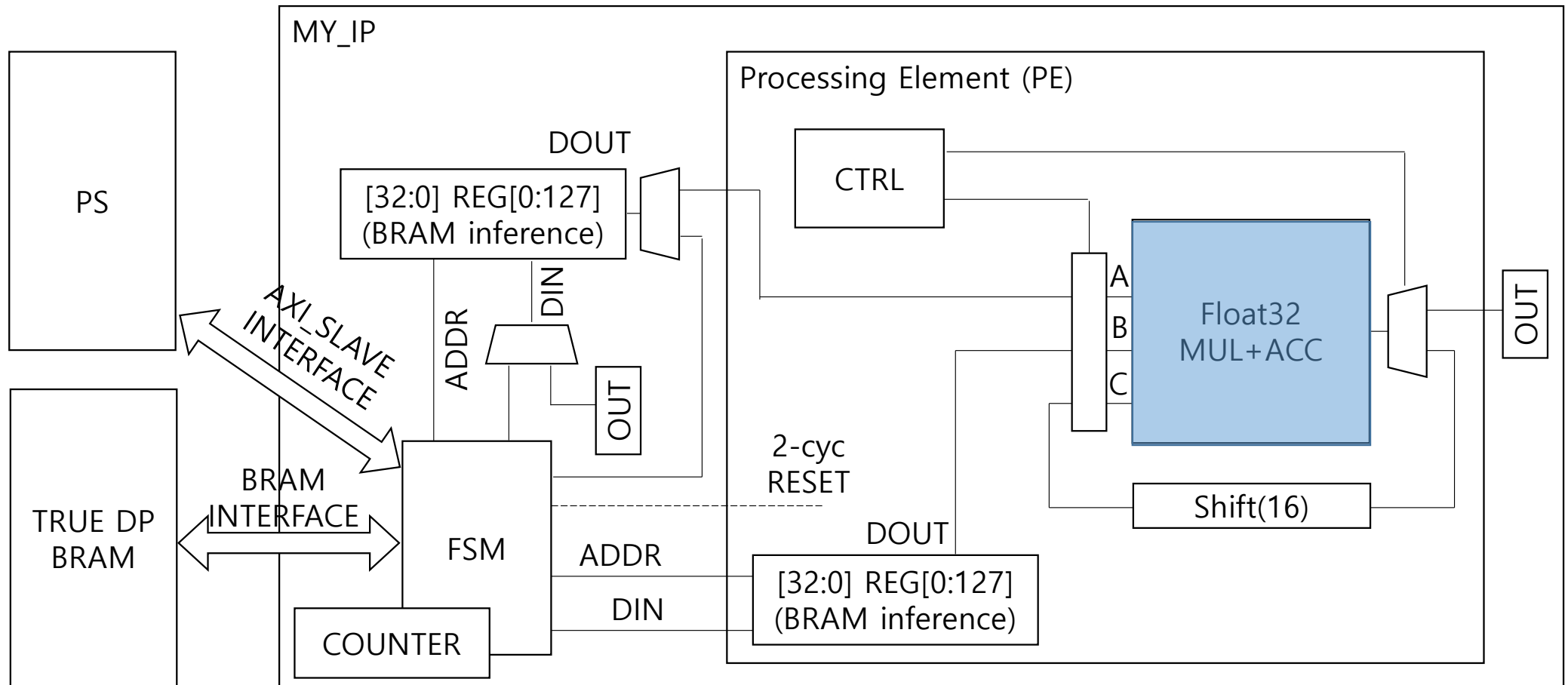
Simulation results

- Change radix



Main Practice

Final Project Overview: Matrix Multiplication IP



Practice

Implement following three combinational blocks and test them (i.e., show wave form). Design your own test bench for your blocks (test at least 32 vectors). You should follow given input and output declaration format.

1. Design n-bit integer adder (default 32 bit)
2. Design n-bit integer multiplier (default 32 bit)
3. Design n-bit integer fused multiplier (default 32 bit)

n-bit Integer Adder (default 32 bit)

```
module my_add #(
    parameter BITWIDTH = 32
)
(
    input [BITWIDTH-1:0] ain,
    input [BITWIDTH-1:0] bin,
    output [BITWIDTH-1:0] dout,
    output overflow
);
/* IMPLEMENT HERE! */
...
endmodule
```

- **ain:** 1st operand
- **bin:** 2nd operand
- **dout:** summation result
- **overflow:**
 - ==1: if overflow is detected
 - ==0: otherwise.

n-bit Integer Multiplier (default 32 bit)

```
module my_mul #(
    parameter BITWIDTH = 32
)
(
    input [BITWIDTH-1:0] ain,
    input [BITWIDTH-1:0] bin,
    output [2*BITWIDTH-1:0] dout
);
/* IMPLEMENT HERE! */
...
endmodule
```

- **ain:** 1st operand
- **bin:** 2nd operand
- **dout:** multiplication result

n-bit Integer fused multiplier (default 32 bit)

```
module my_fusedmult #(
    parameter BITWIDTH = 32
)
(
    input [BITWIDTH-1:0] ain,
    input [BITWIDTH-1:0] bin,
    input en,
    input clk,
    output [2*BITWIDTH-1:0] dout
);
/* IMPLEMENT HERE! */
...
endmodule
```

- **ain:** 1st operand
- **bin:** 2nd operand
- **dout:** multiplication and accumulation result
- **overflow:**
 - ==1: if overflow is detected
 - ==0: otherwise.
- **en:**
 - ==1: fused multiplier computes output
 - ==0: fused multiplier initialize output as 0
- **clk:** clock

Homework

■ Requirements

- Result

- Attach your project folder with all your Verilog codes (e.g., Adder, Multiplier, fused-multiplier, test-bench)
- Attach your waveform(simulation result) with [student_number, name]
 - Refer to slide 21-23 for details
- **Use the test-bench provided**

- Report

- Explain Adder, Multiplier, fused-multiplier that you implemented
- In your own words
- Either in Korean or in English
- # of pages does not matter
- **PDF only!!**

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

■ Upload (.zip) file on ETL

- Submit one (.zip) file

- zip file name : [Lab03]name (ex : [Lab03]홍길동)

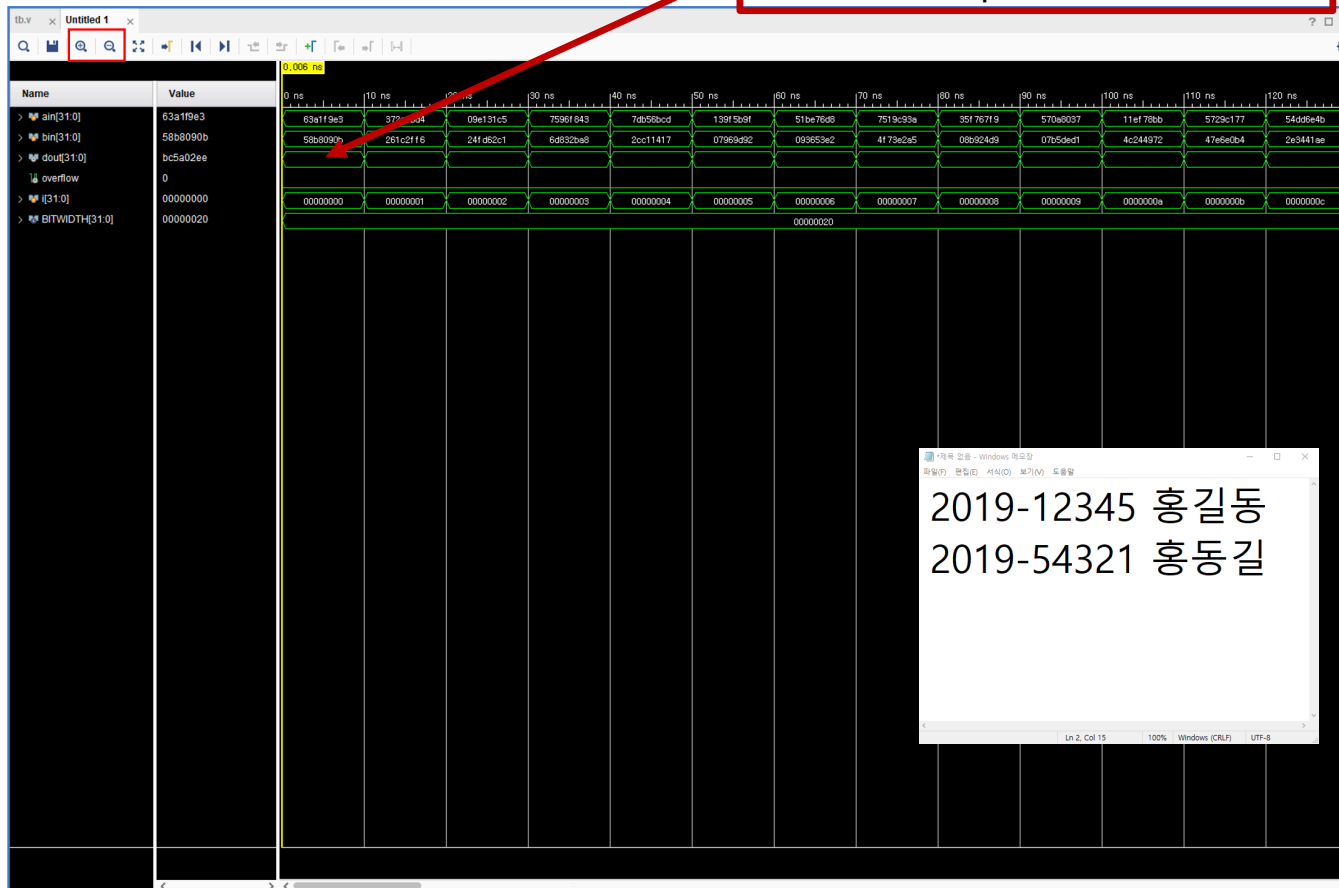
- Due: 3/30(MON) 23:59

- **No Late Submission**

Homework

■ Adder Simulation

Result of Implemented Adder

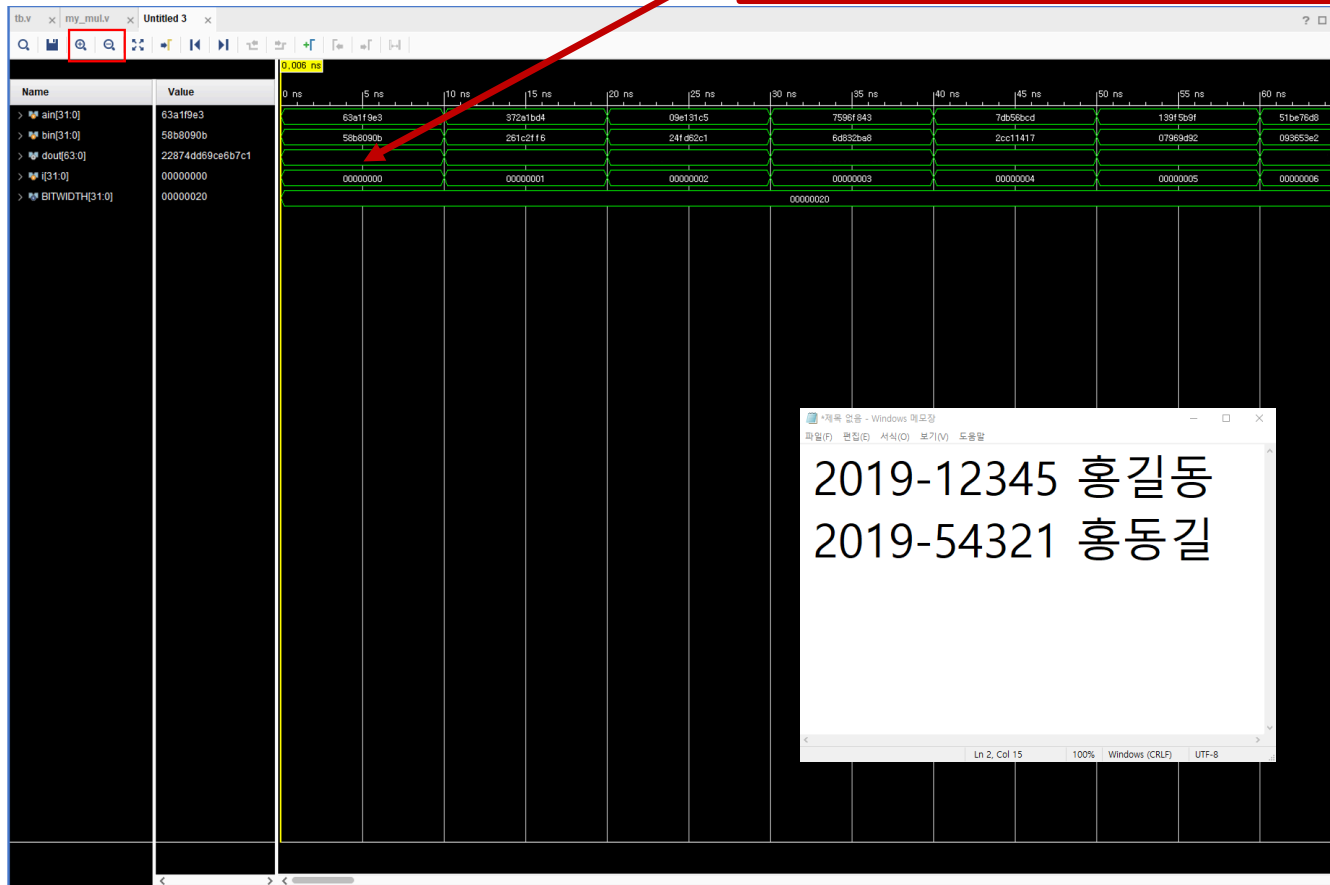


- $dout = ain + bin$
- Match time-scale using +/- buttons

Homework

■ Multiplier Simulation

Result of Implemented Multiplier

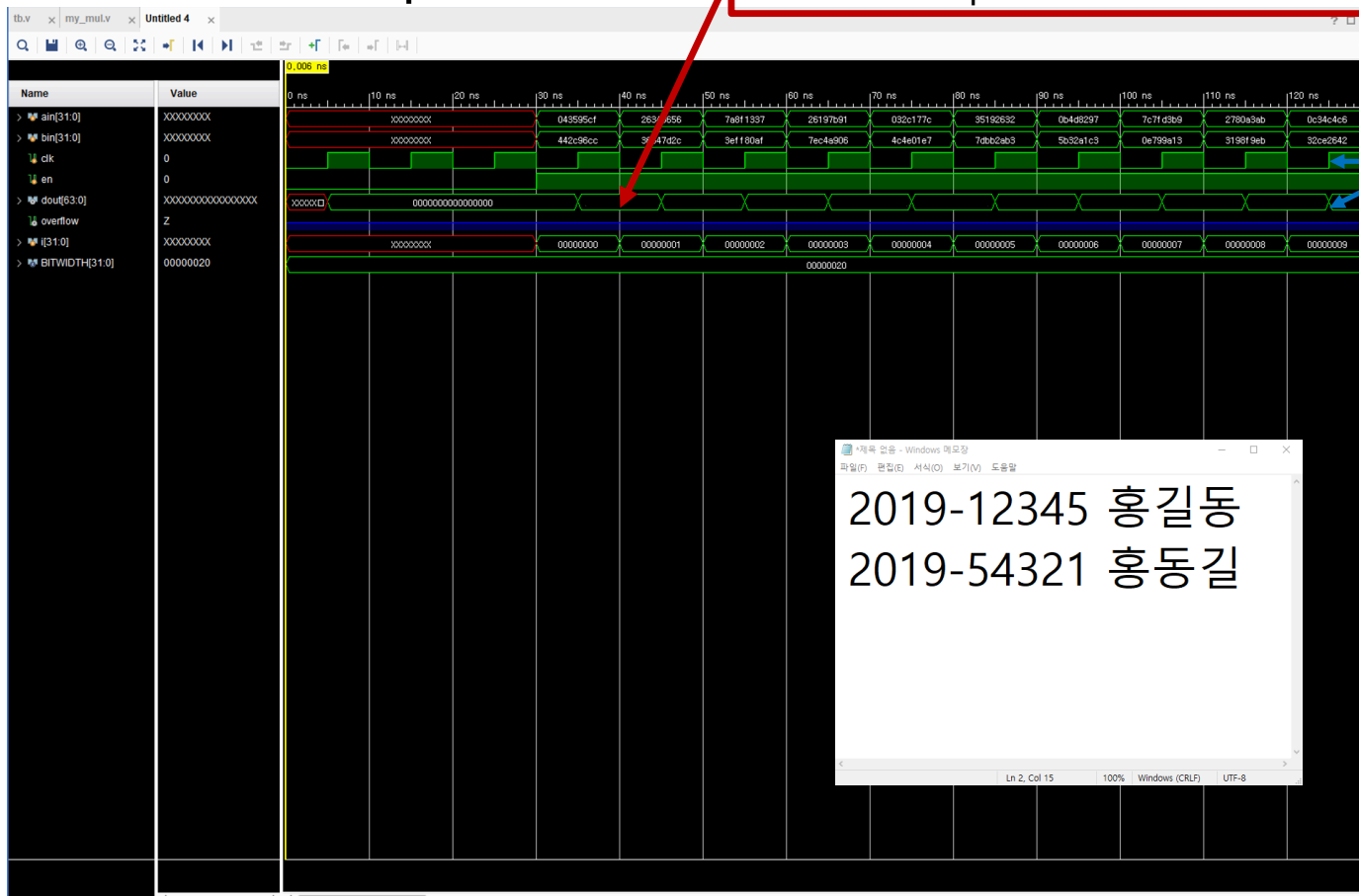


- $dout = ain * bin$
- Match time-scale using +/- buttons

Homework

■ Fused-multiplier Simulation

Result of Implemented fused-multiplier



- Every positive clock, fuse d-multiplier operation

Operation must be executed at positive clock

- $dout = dout + ain * bin$