Practice 3

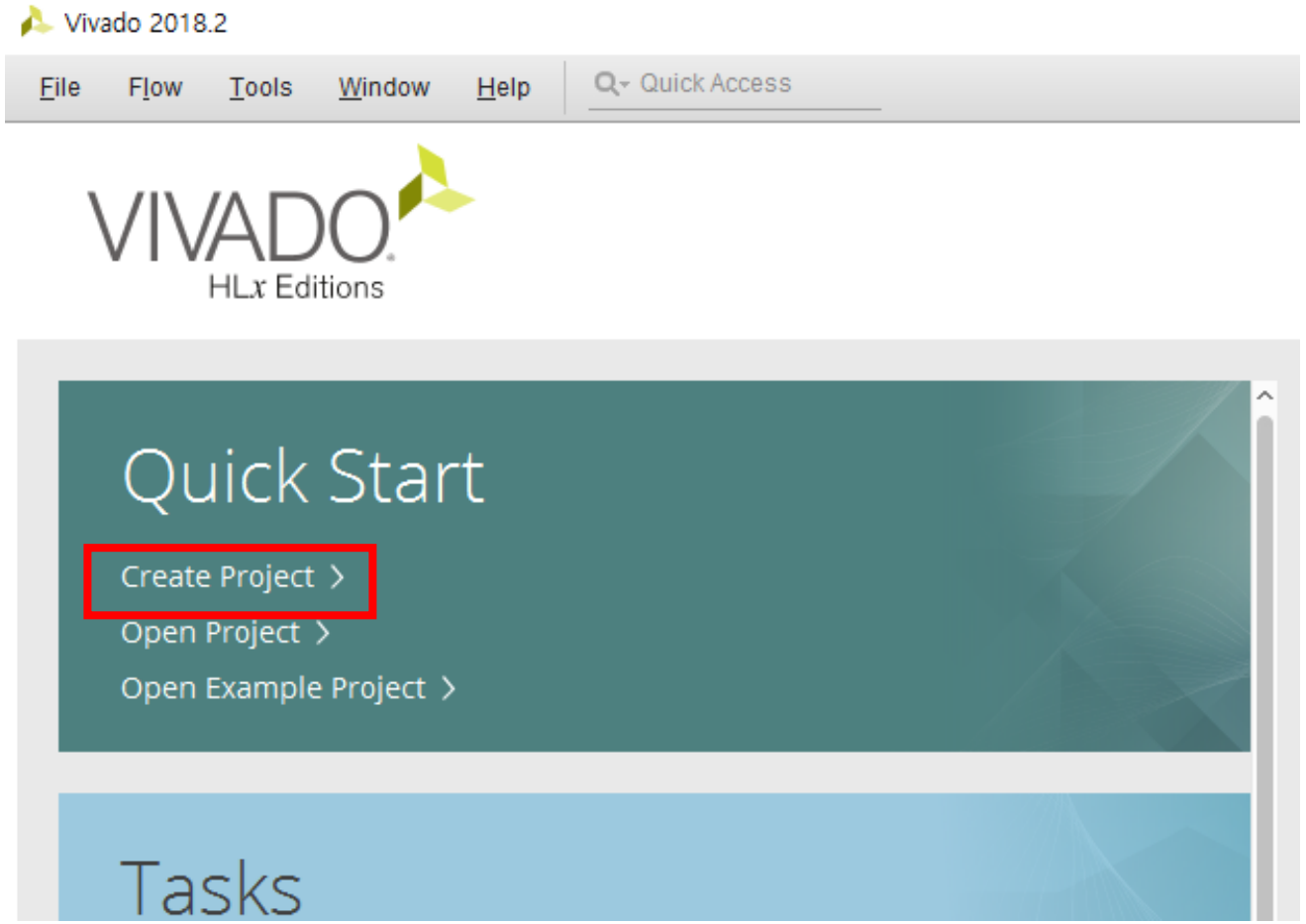
- How to use Vivado & Basic syntax

Computing Memory Architecture Lab.

Vivado Tutorial

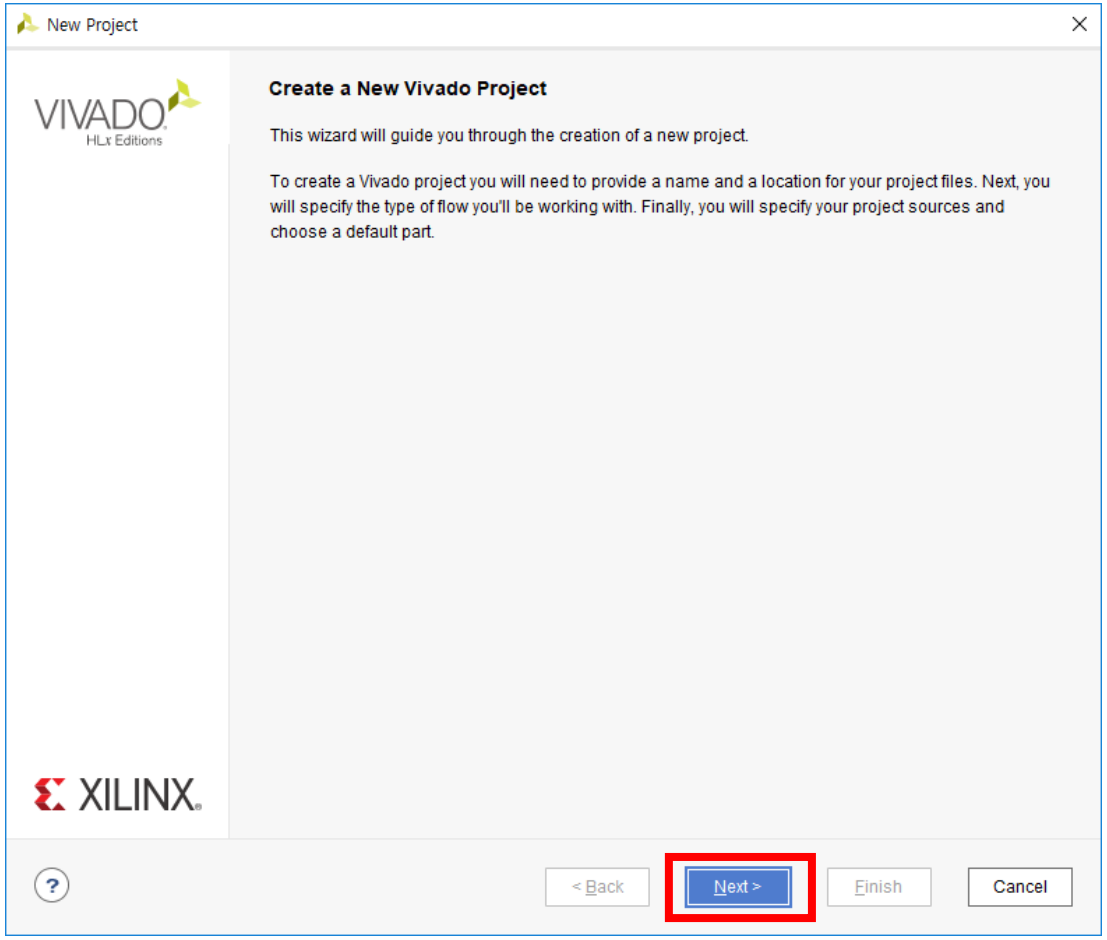
Vivado project creation

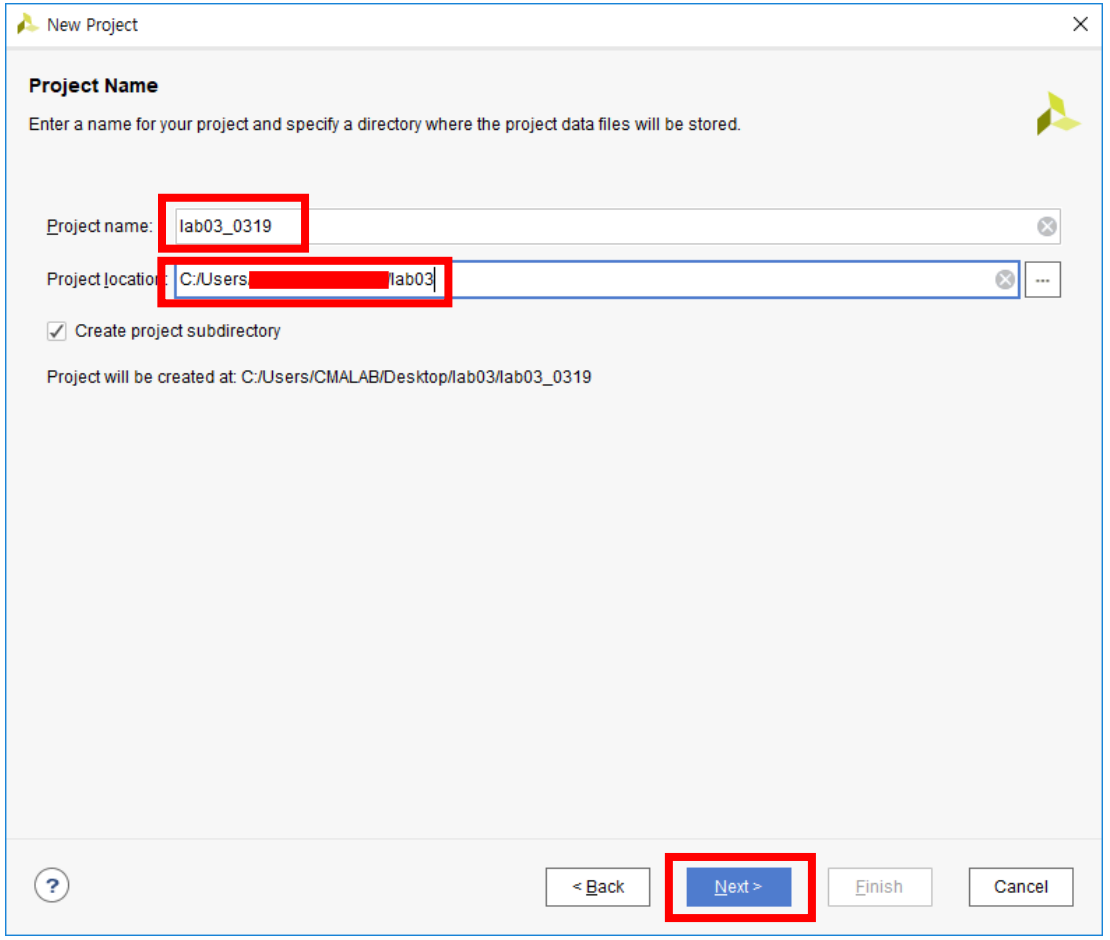
- Create New Project



Vivado project creation

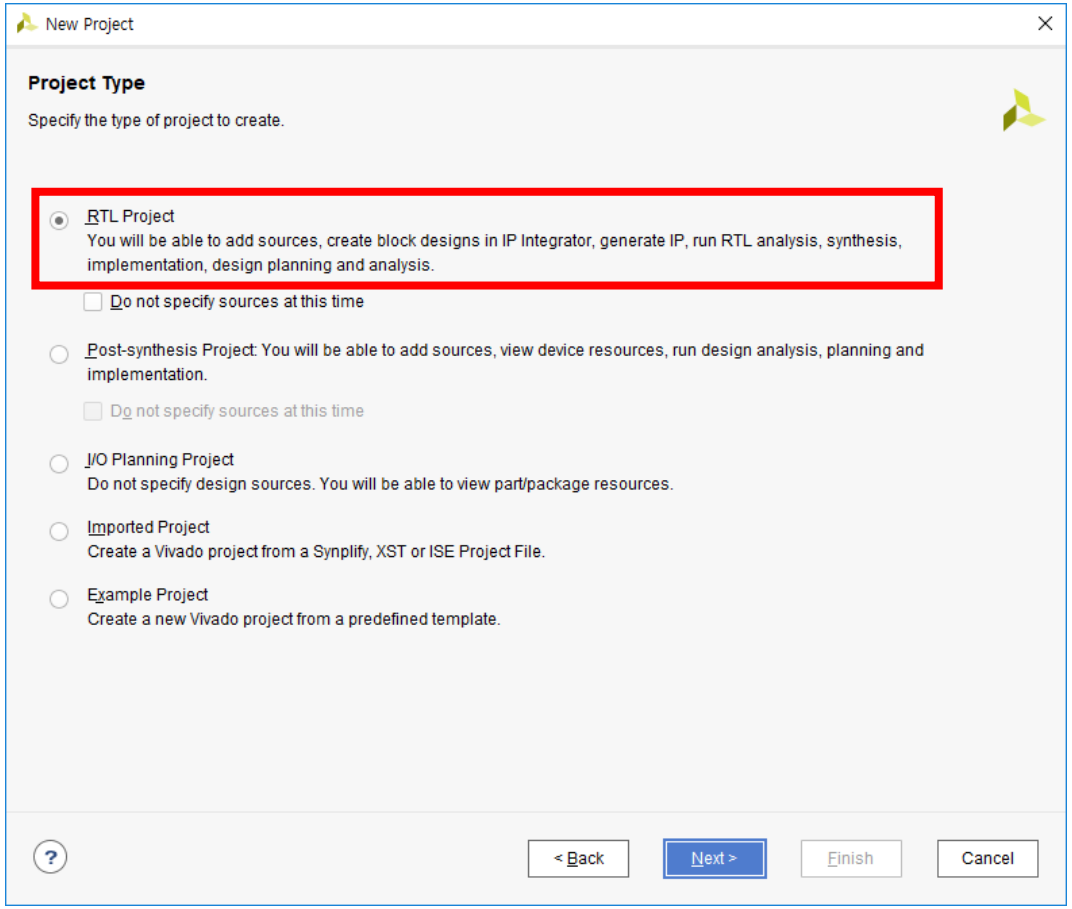
- Enter project name and location





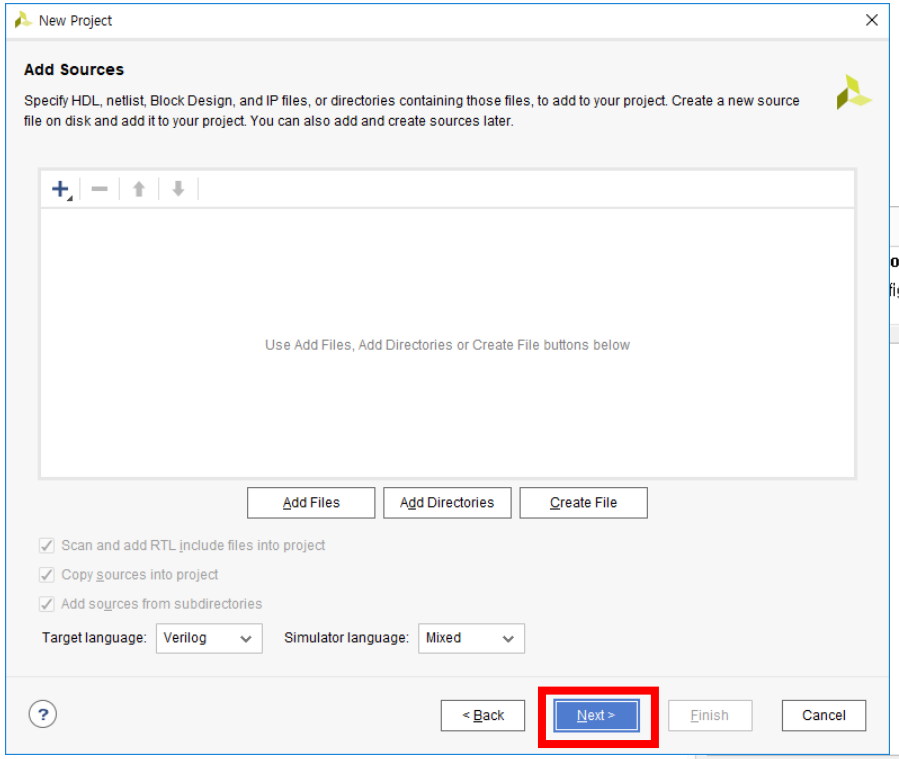
Vivado project creation

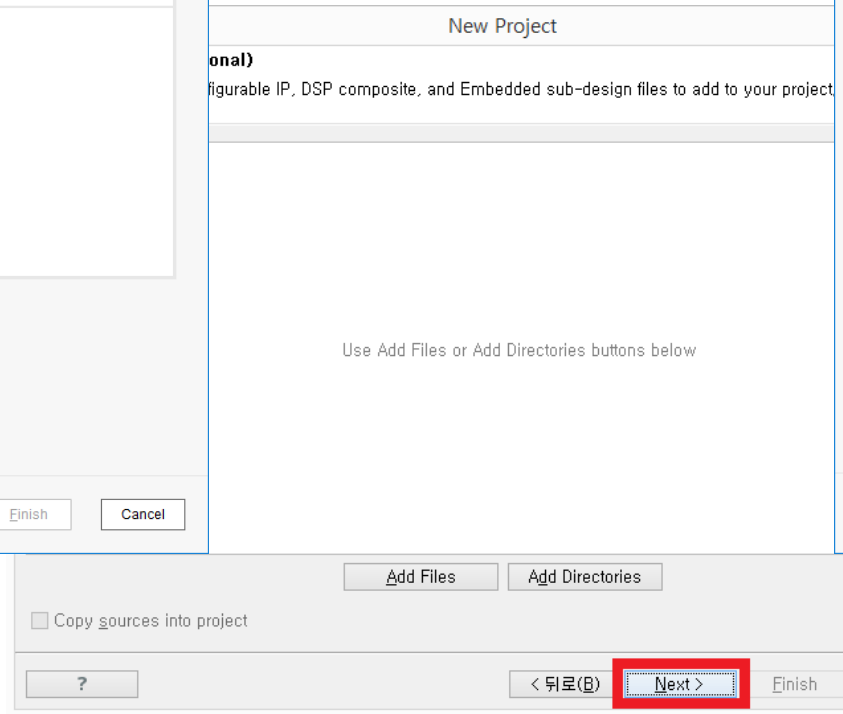
- Select project type
  - RTL Project

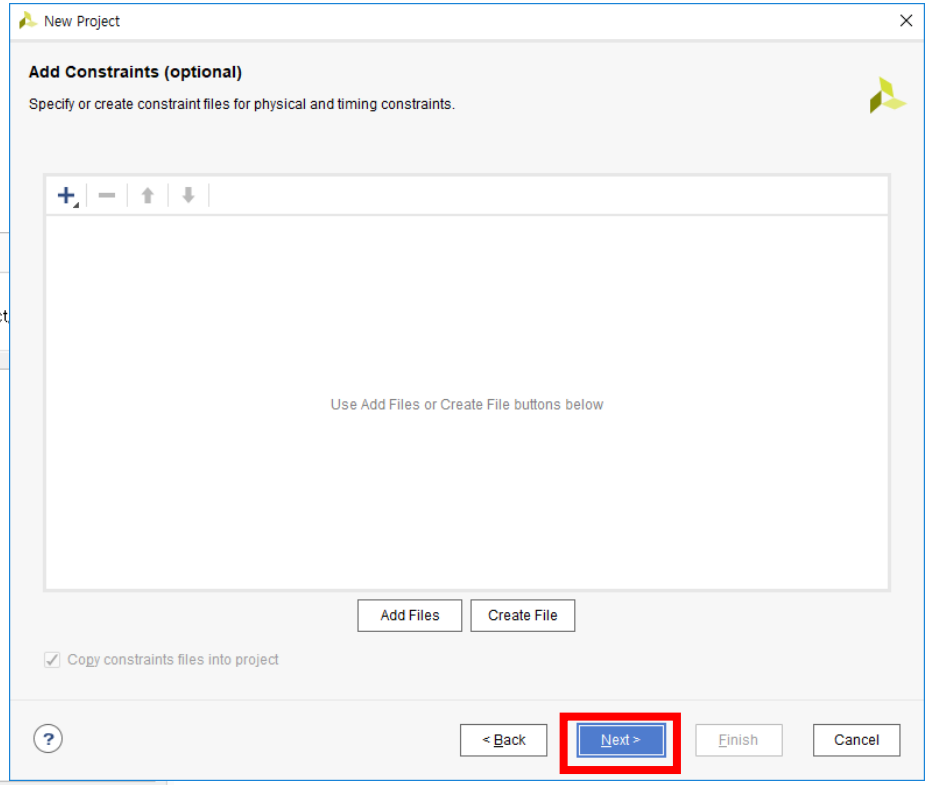


Vivado project creation

- (Optional) add sources or IPs, constraints

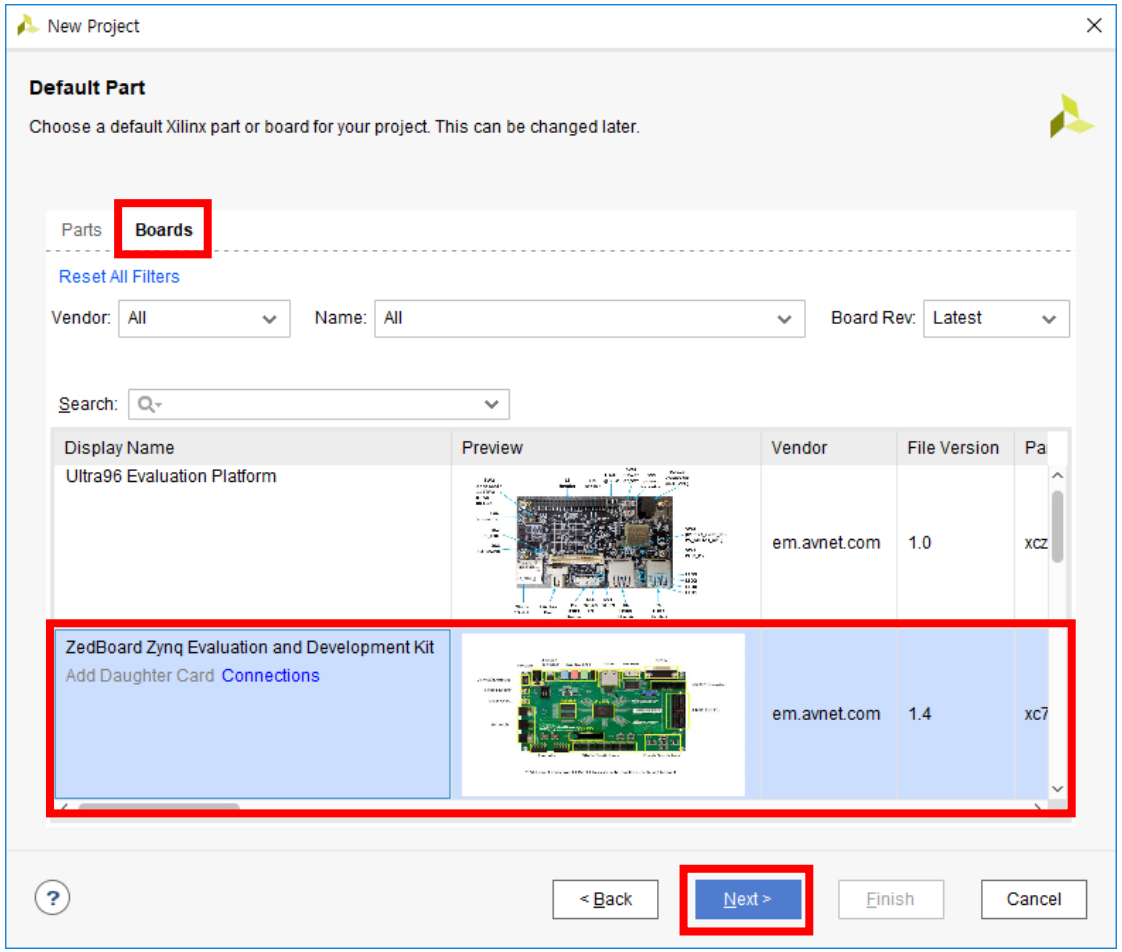






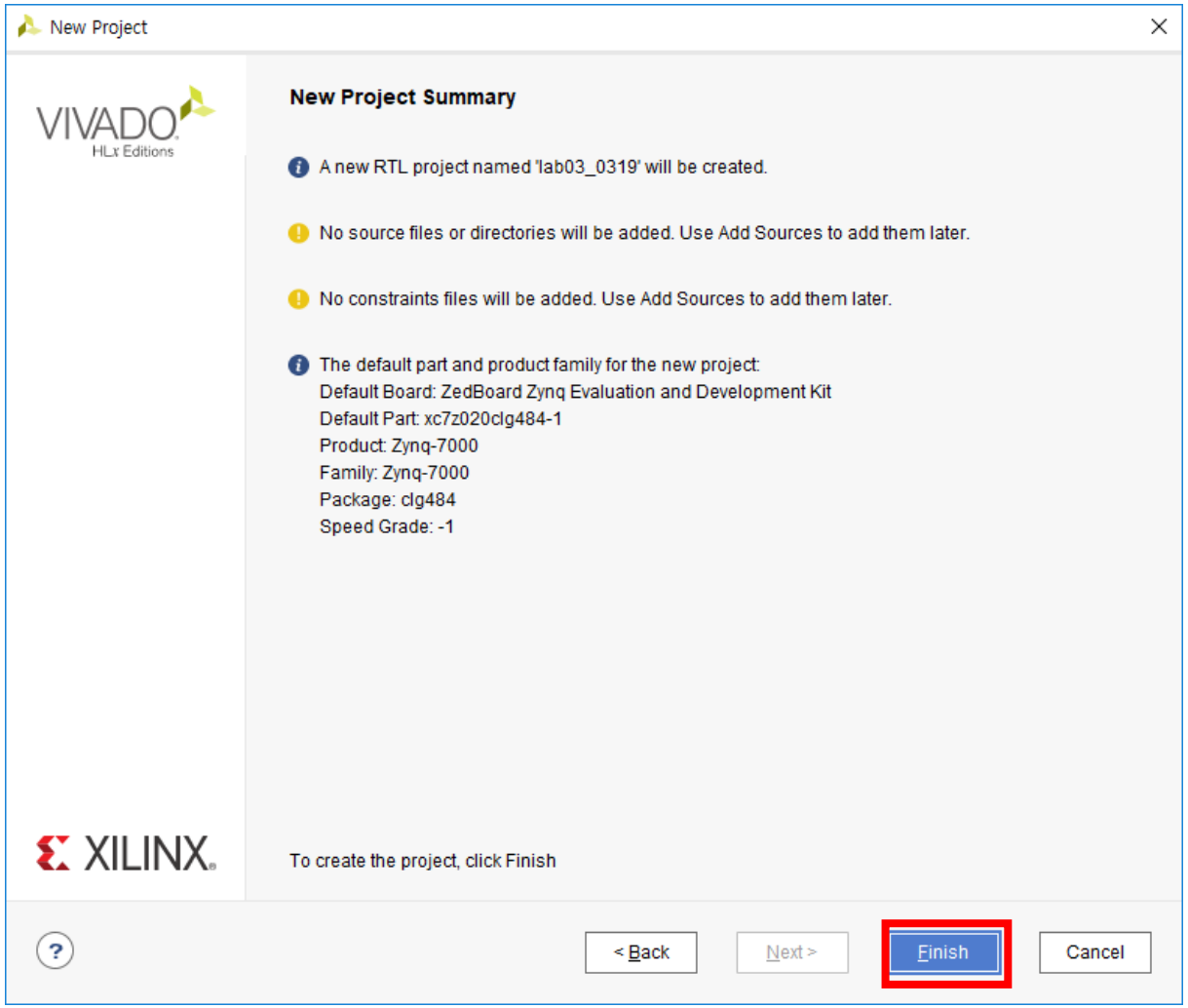
Vivado project creation

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

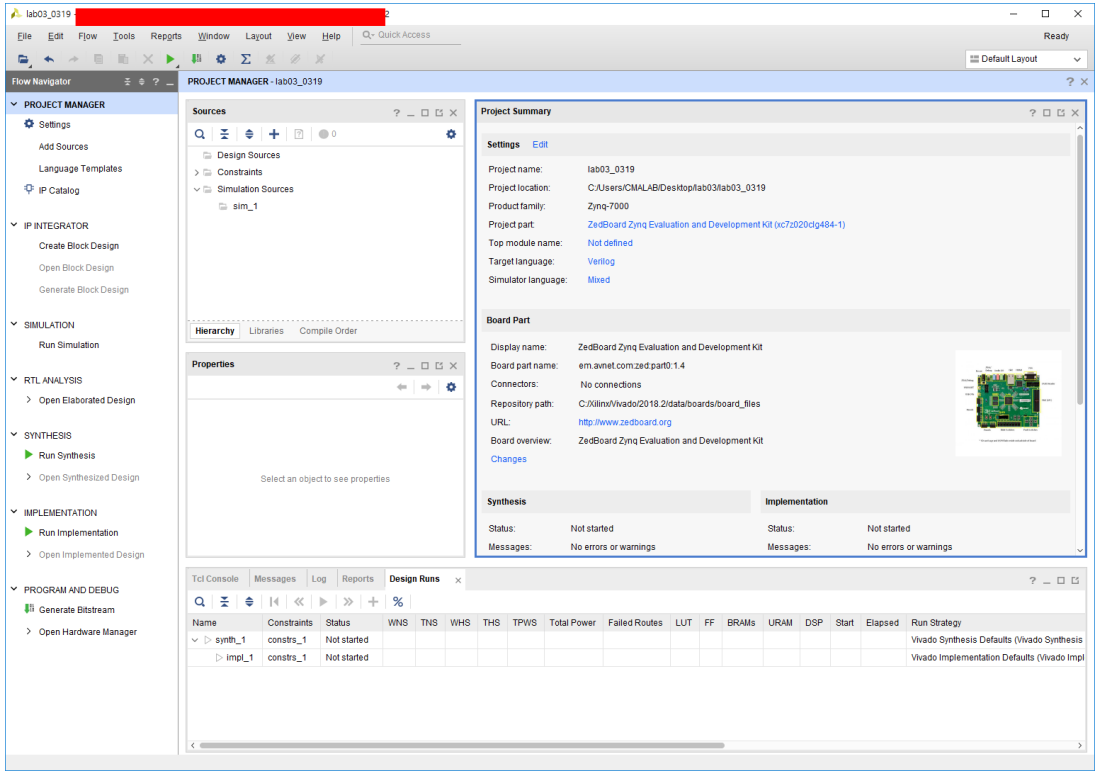


Vivado project creation

- Finish project creation

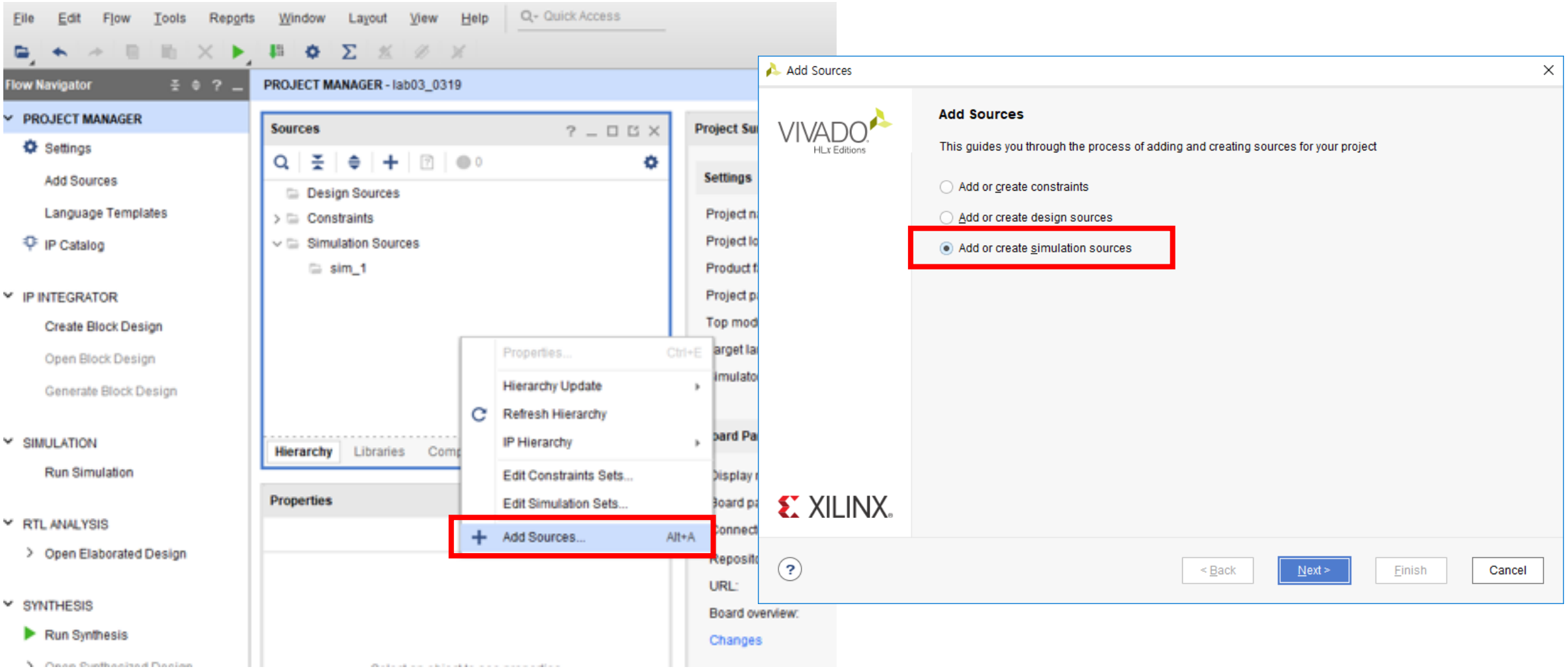


Main page



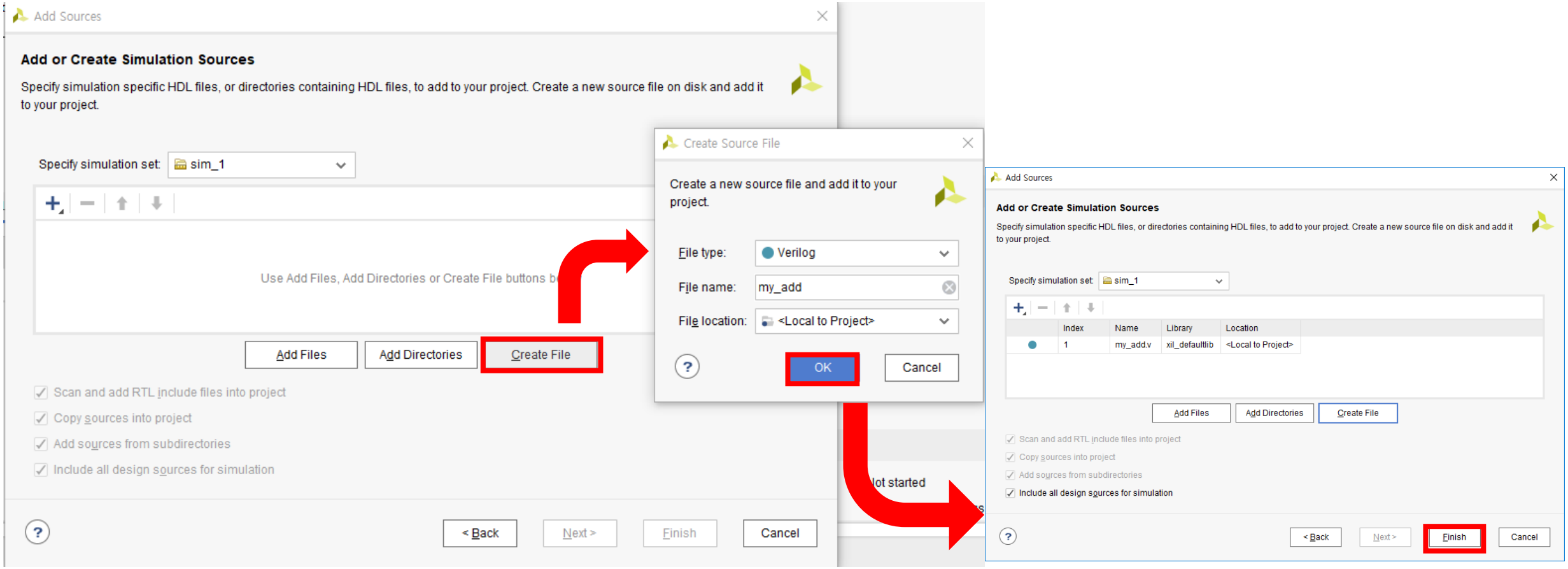
Create simulation sources

- Create simulation source



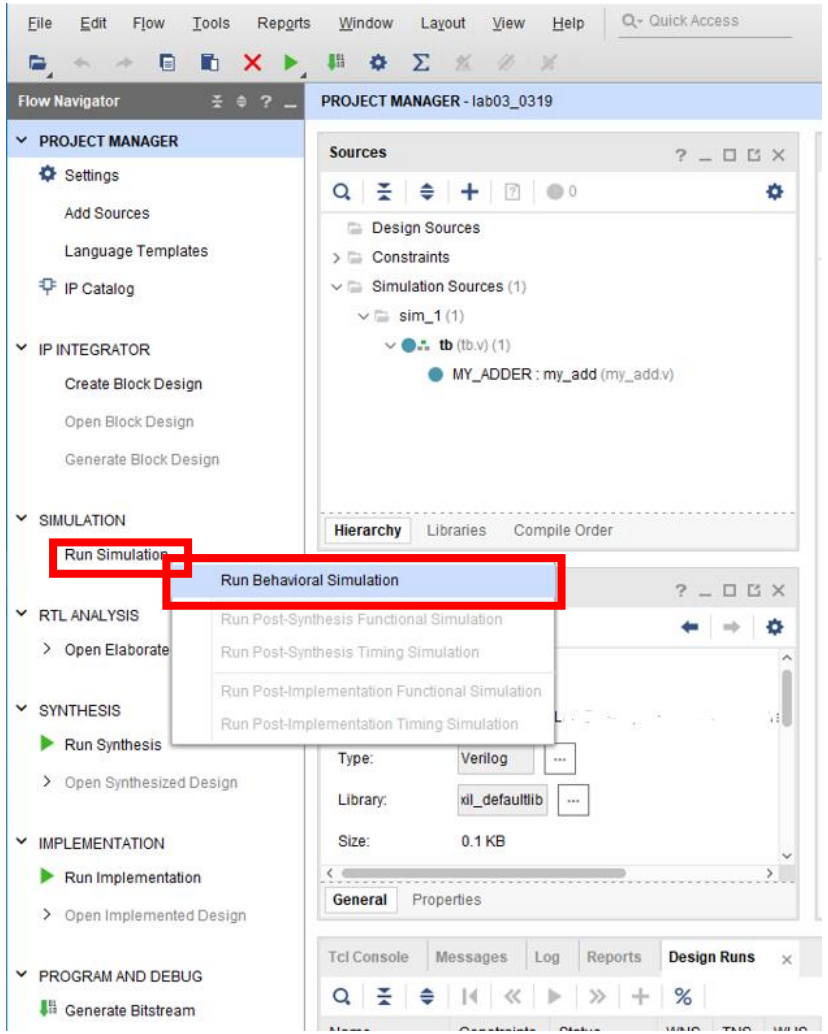
Create simulation sources

- Create simulation source
  - Modules and testbenches



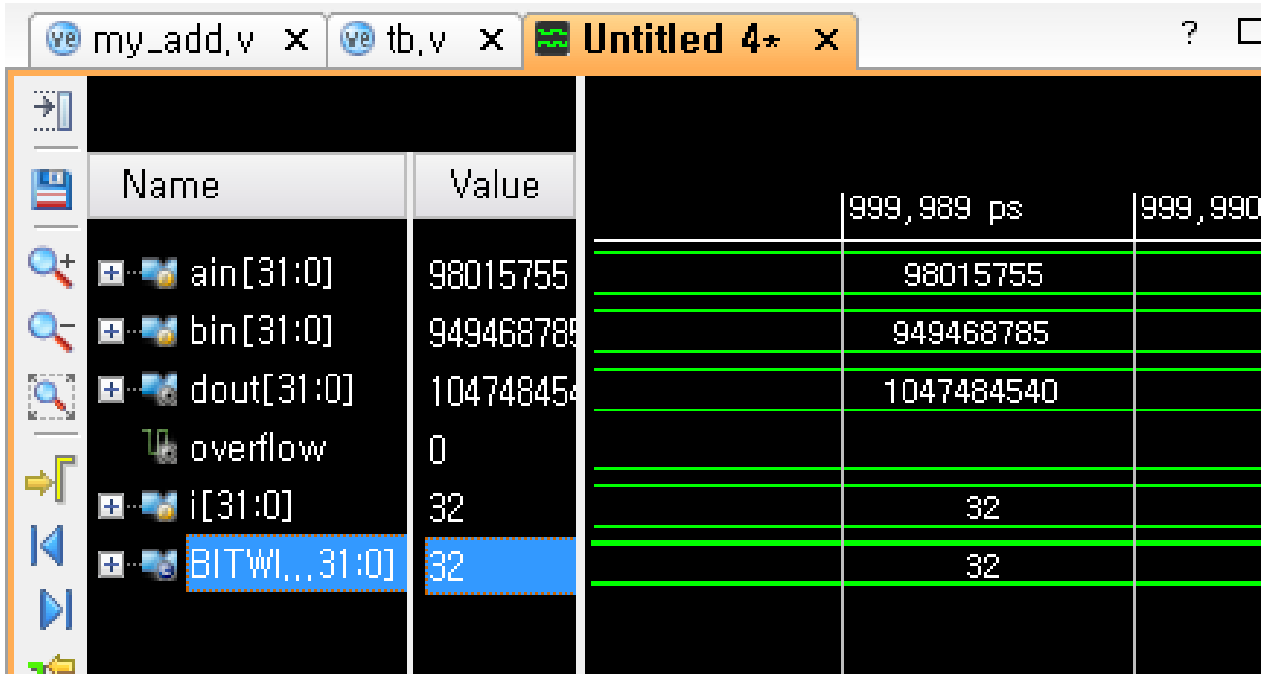
Simulation

- Run Simulation -> Run Behavioral Simulation



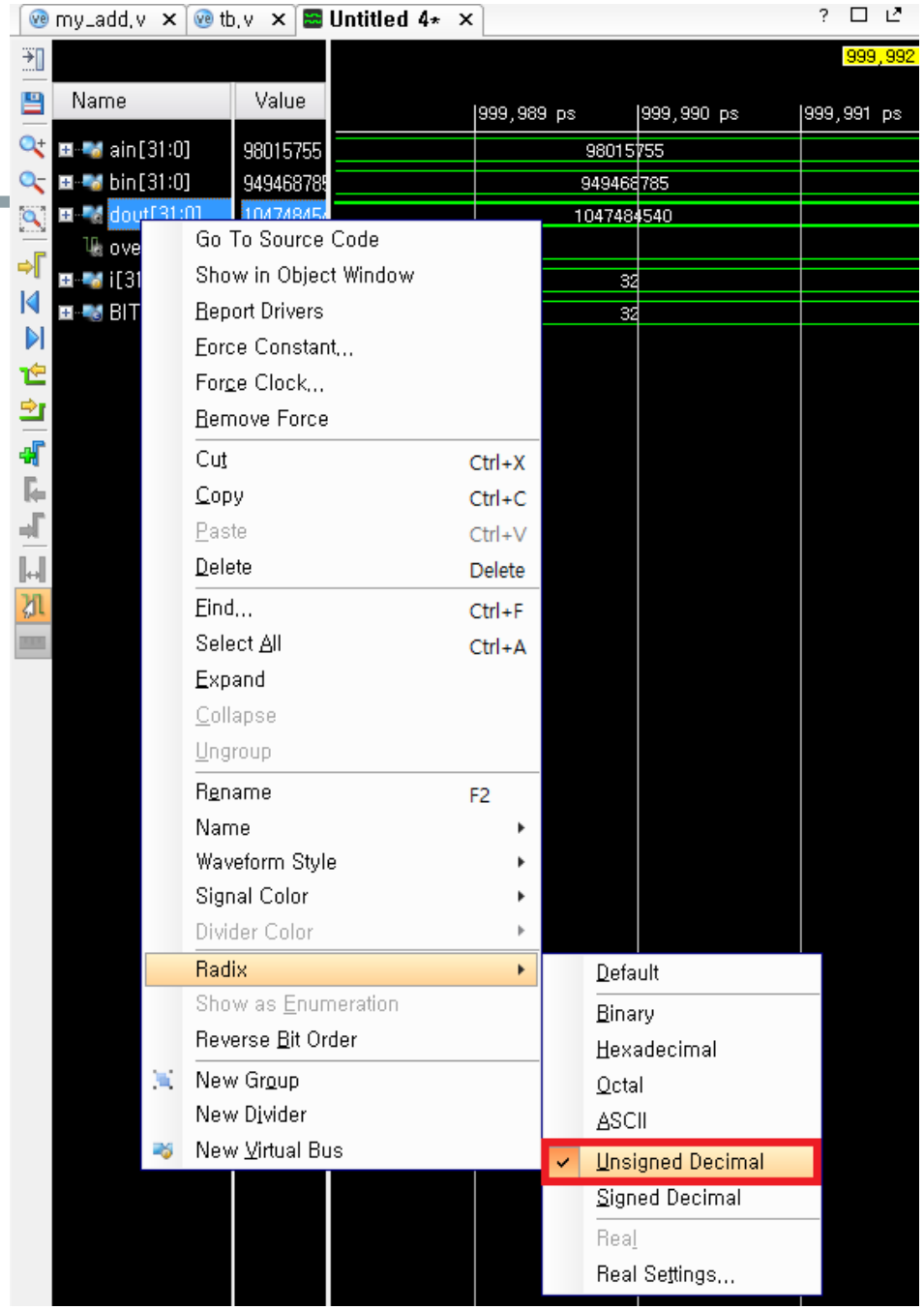
Simulation results

- Check result



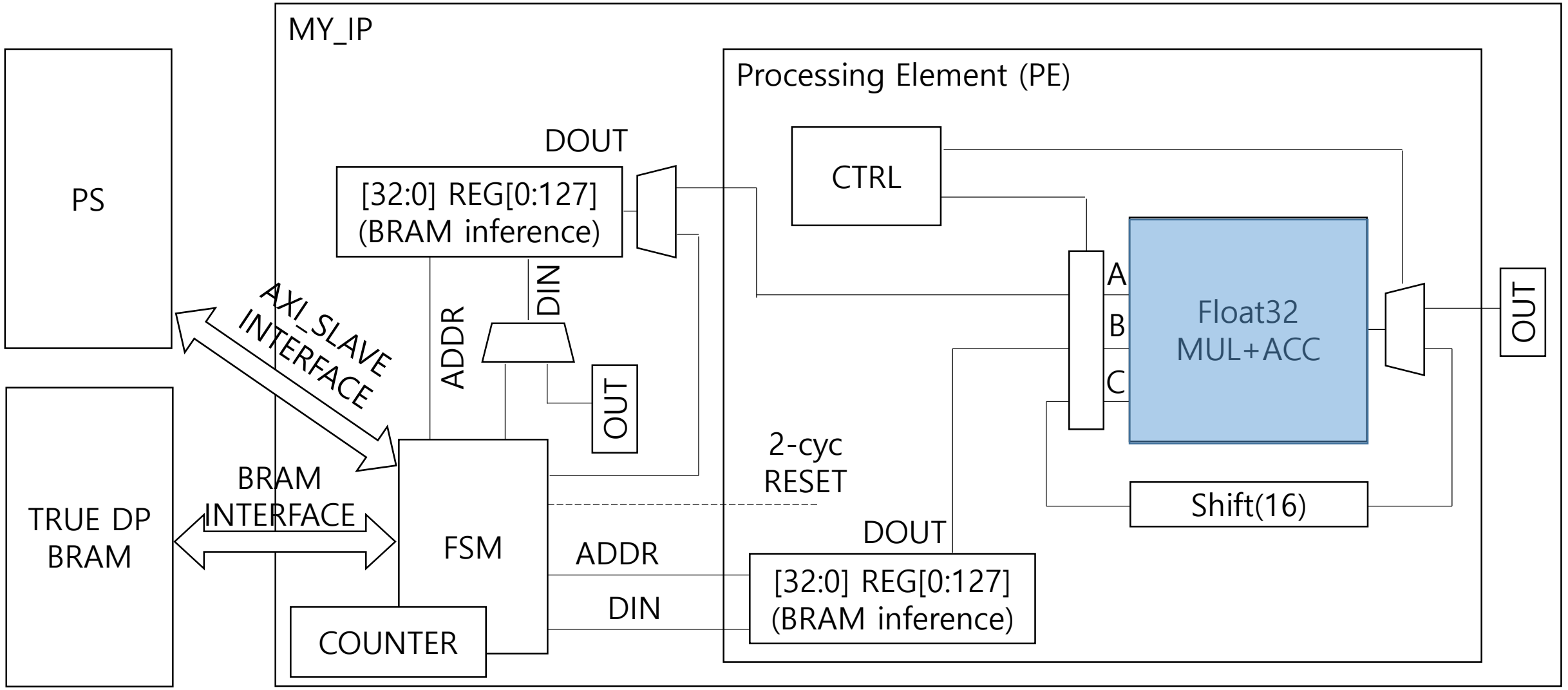
Simulation results

- 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 v ectors). 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
```

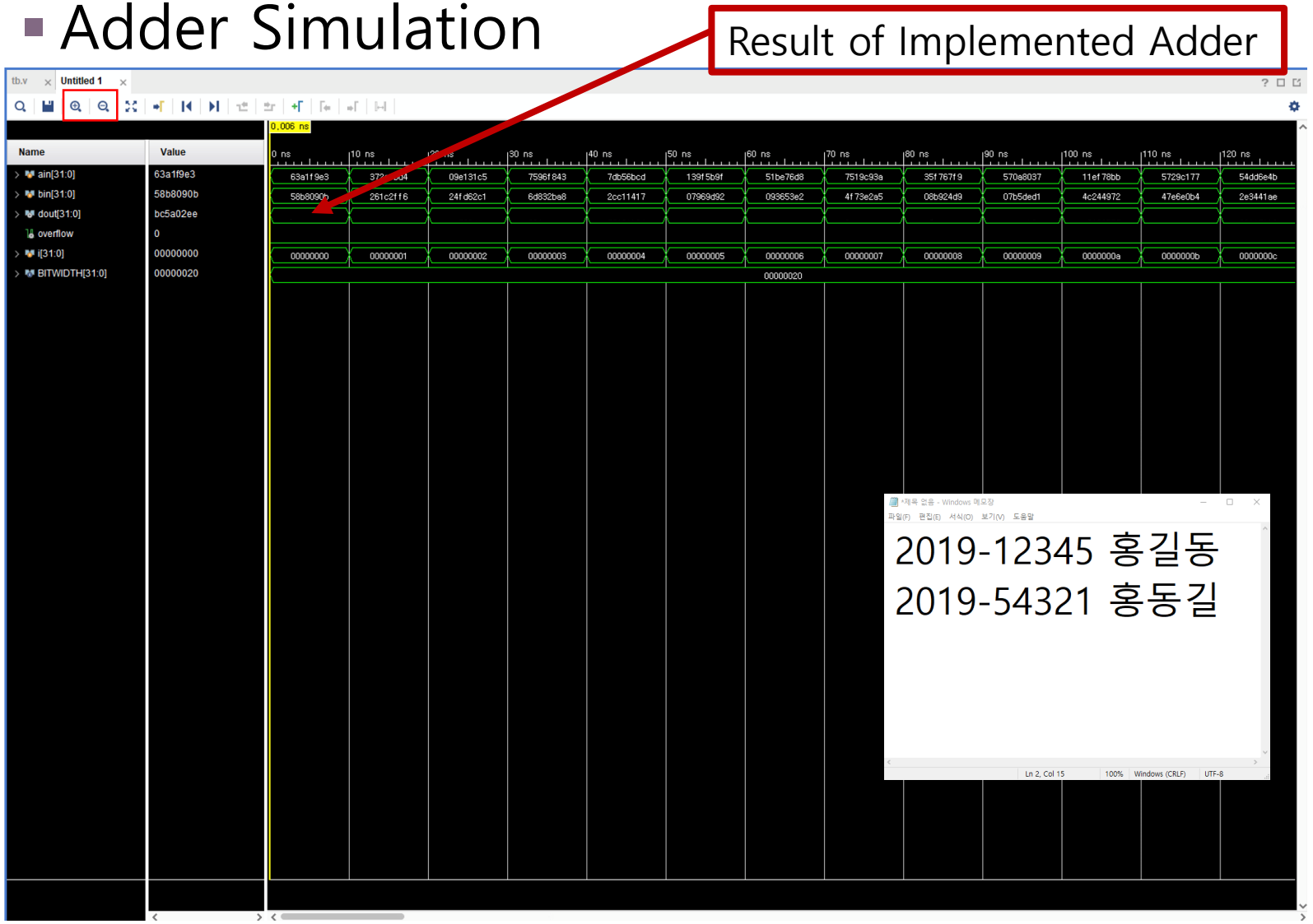
- **ain:** 1st operand
- **bin:** 2nd operand
- **dout:** multiplication and accumulation result
- **en**:
  - ==1: fused multiplier computes output
  - ==0: fused multiplier initialize output as 0
- **clk**: clock

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

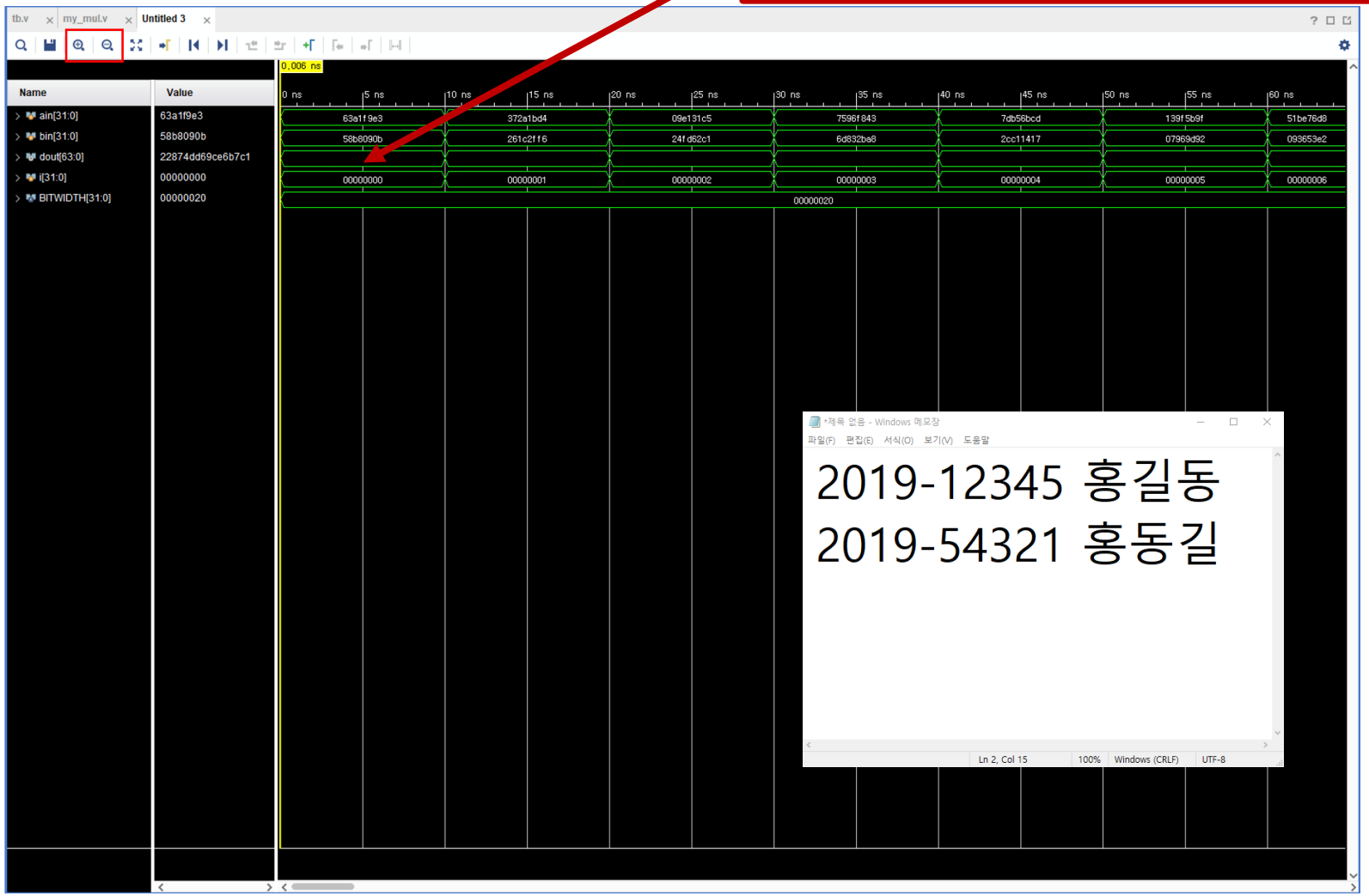
- Adder Simulation



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

Homework

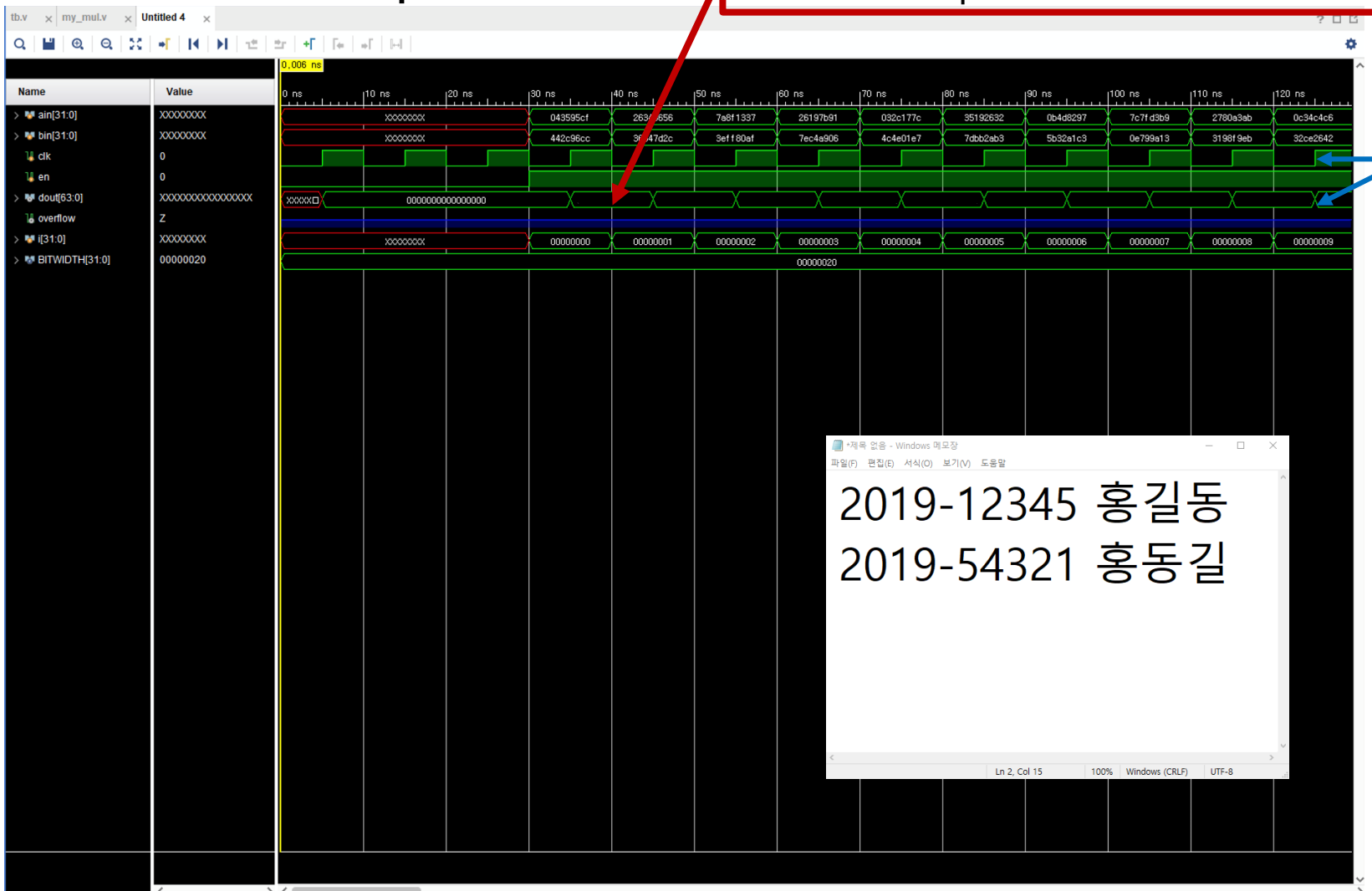
- Multiplier Simulation



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

Homework

- Fused-multiplier Simulation



- Every positive clock, fused-multiplier operation
- **Operation must be executed at positive clock**
- dout = dout + ain * bin