Lab #1: Vivado & Basic Verilog Syntax

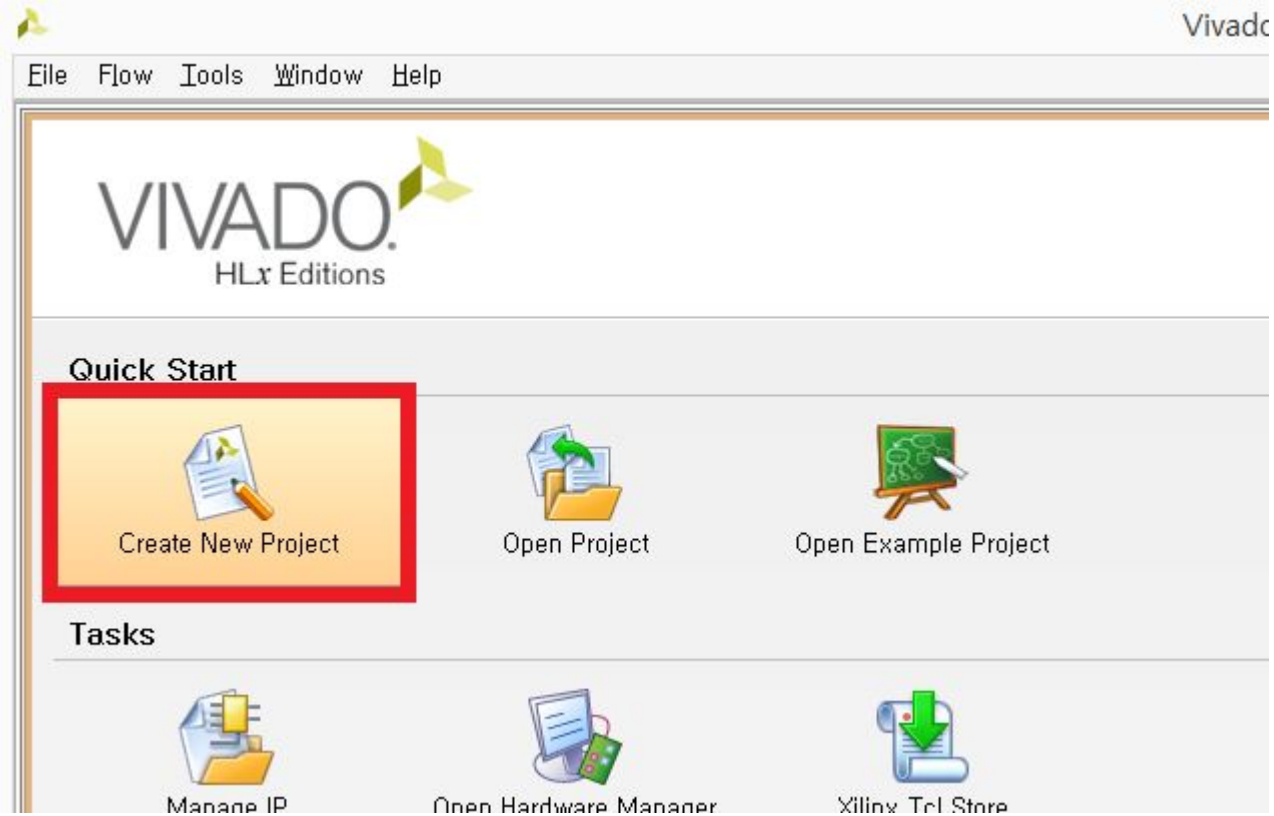
03/08/2018

4190.309A: Hardware System Design
(Spring 2018)

Vivado Tutorial

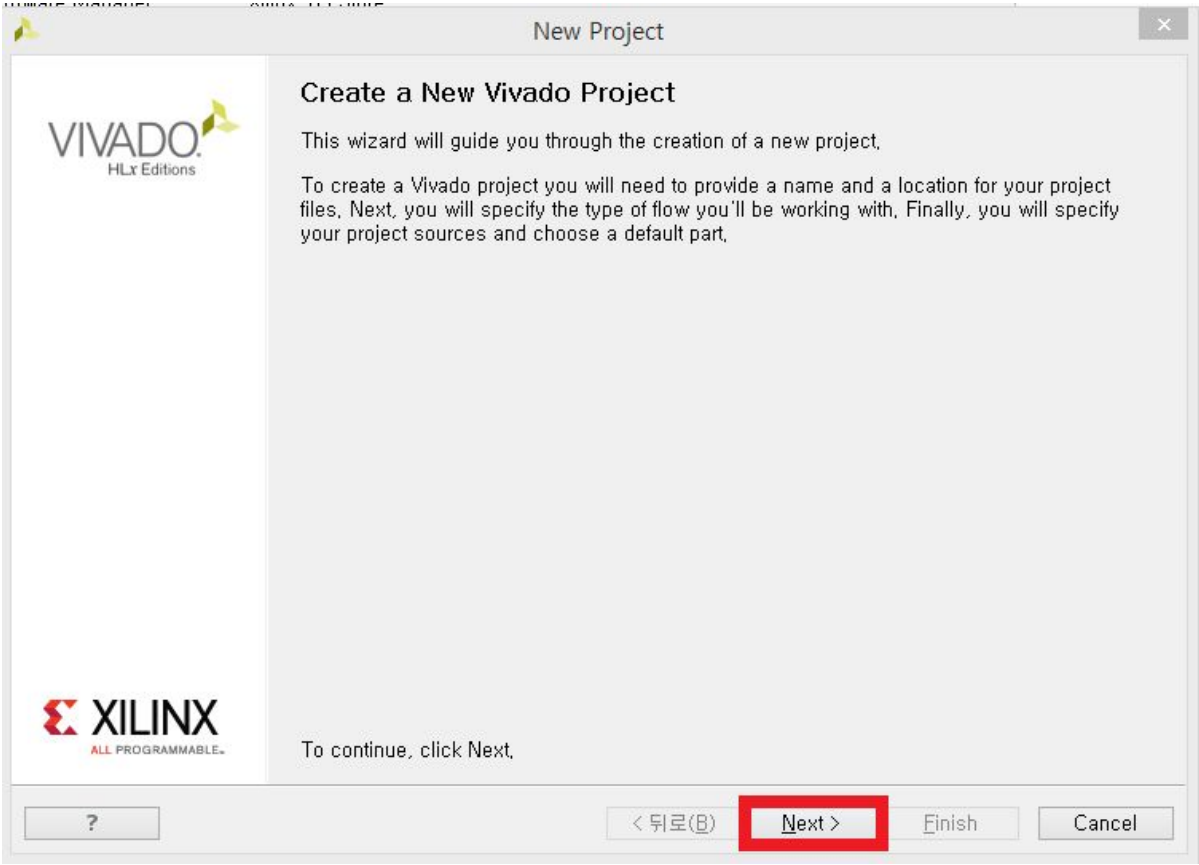
Vivado project creation

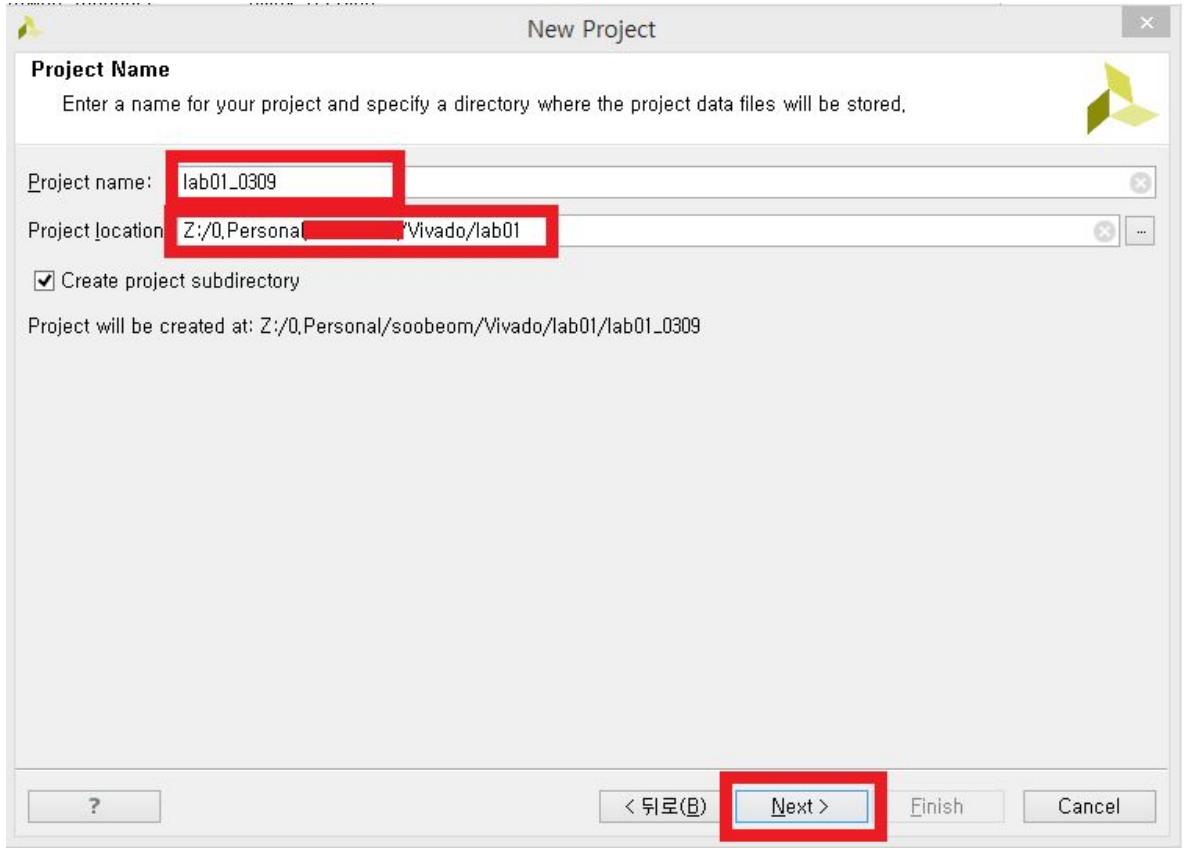
- Create New Project



Vivado project creation

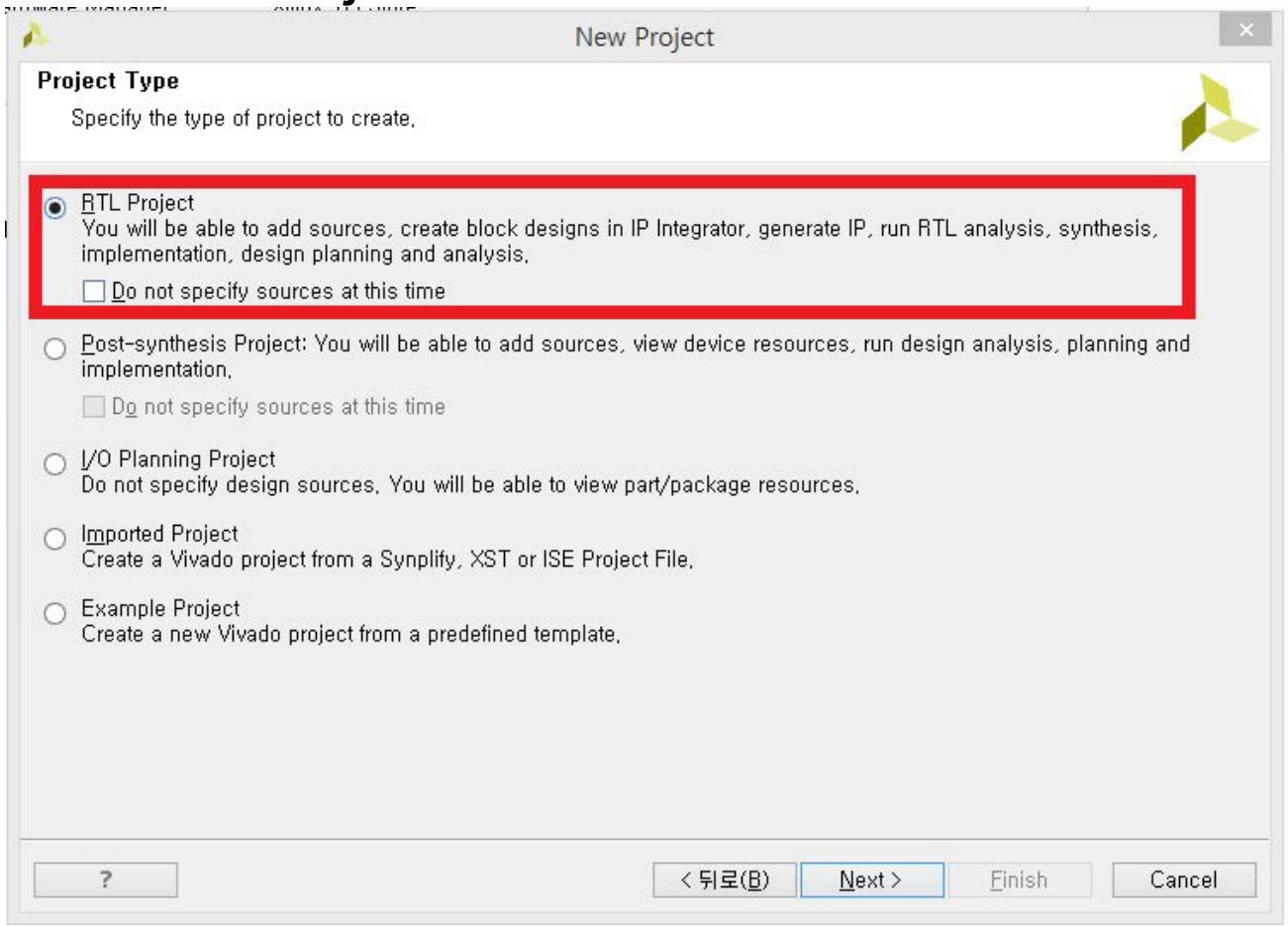
- Enter project name and location





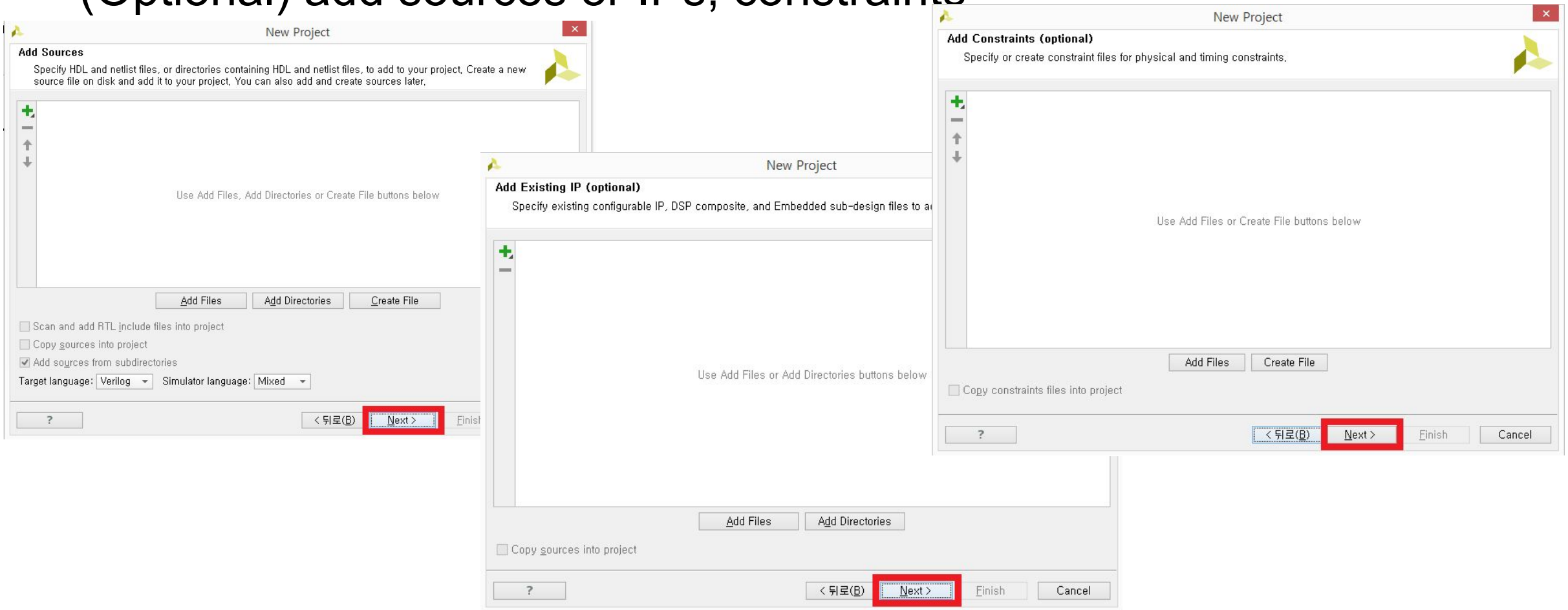
Vivado project creation

- Select project type
  - RTL Project



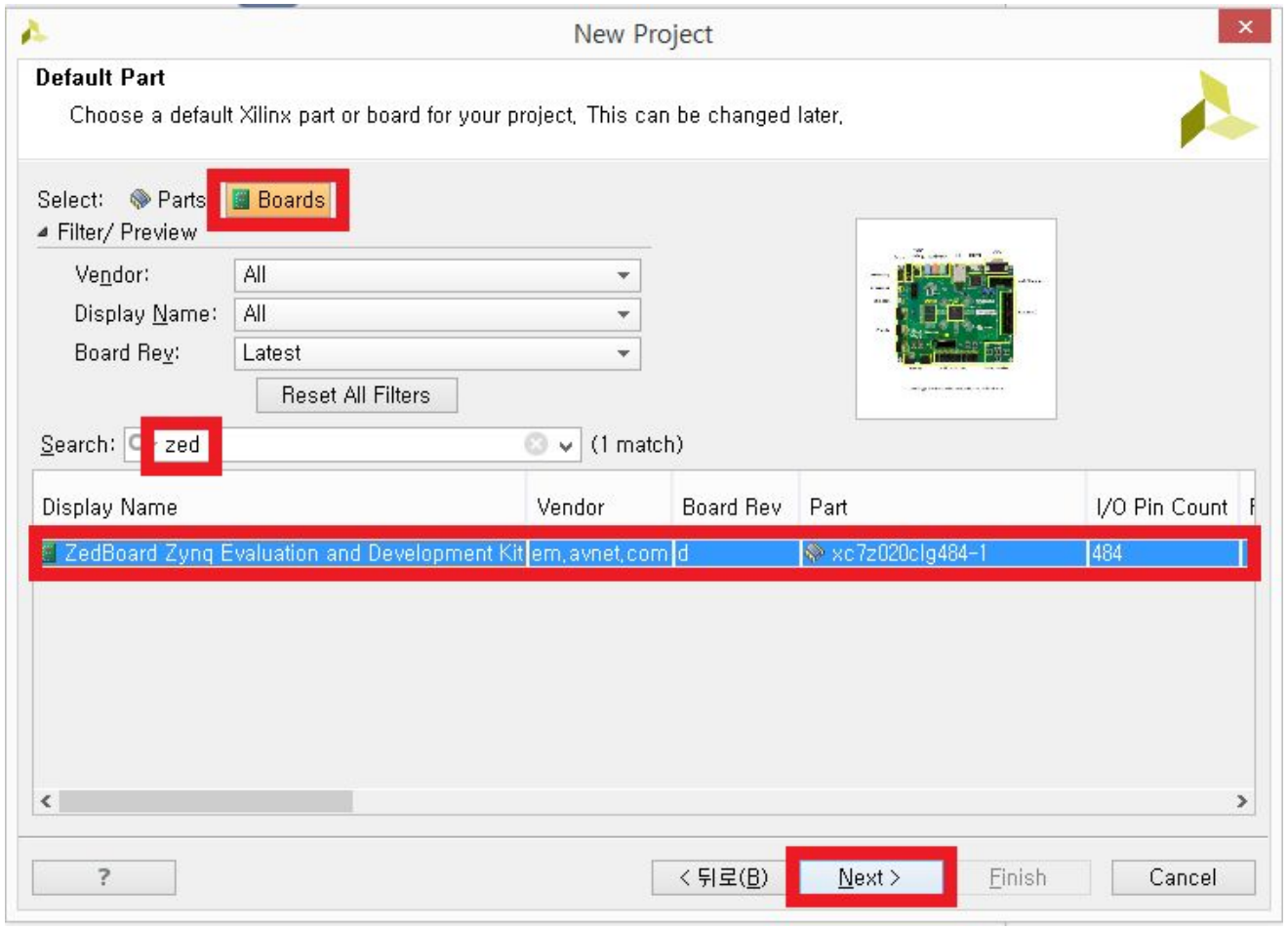
Vivado project creation

- (Optional) add sources or IPs, constraints



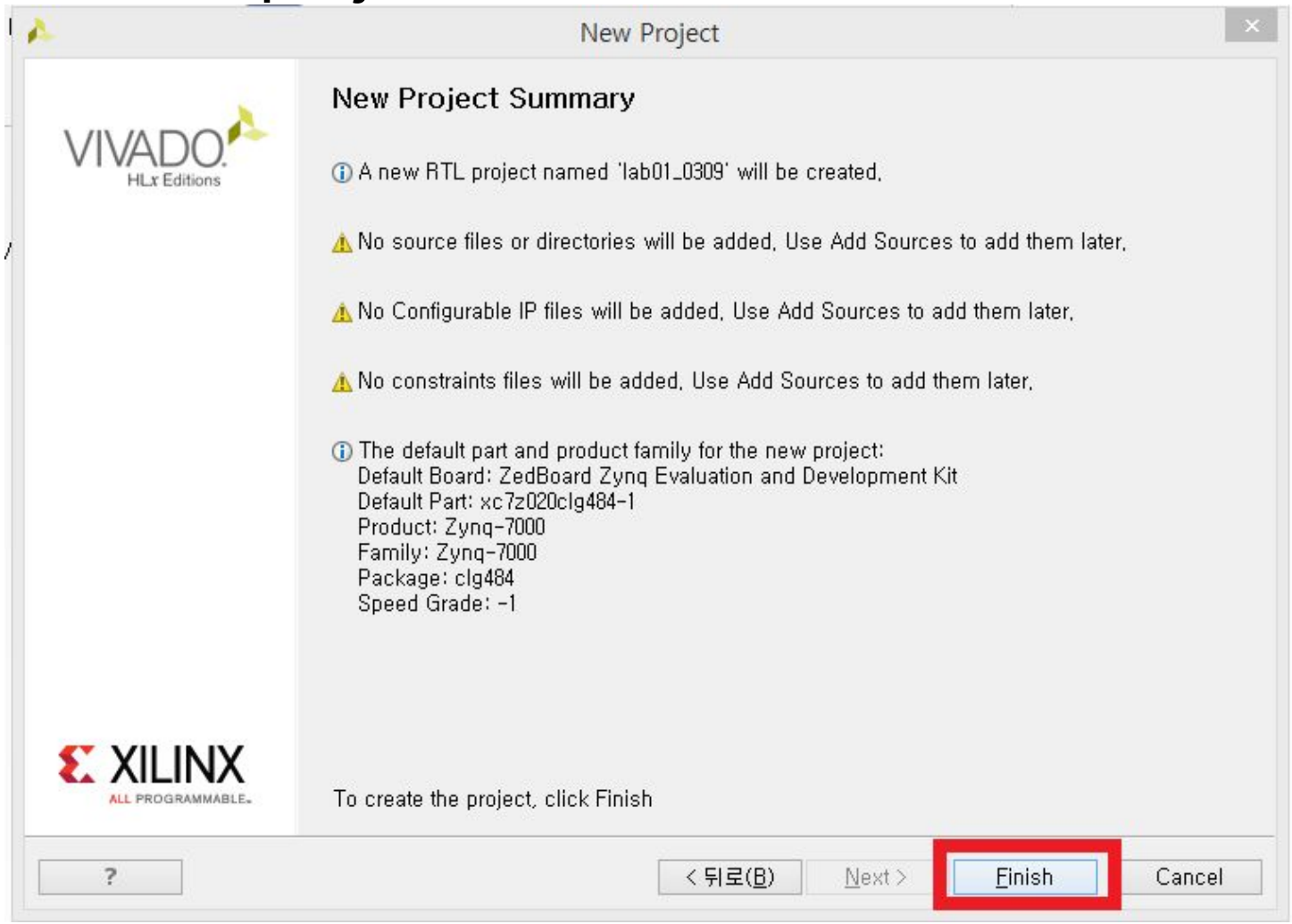
Vivado project creation

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



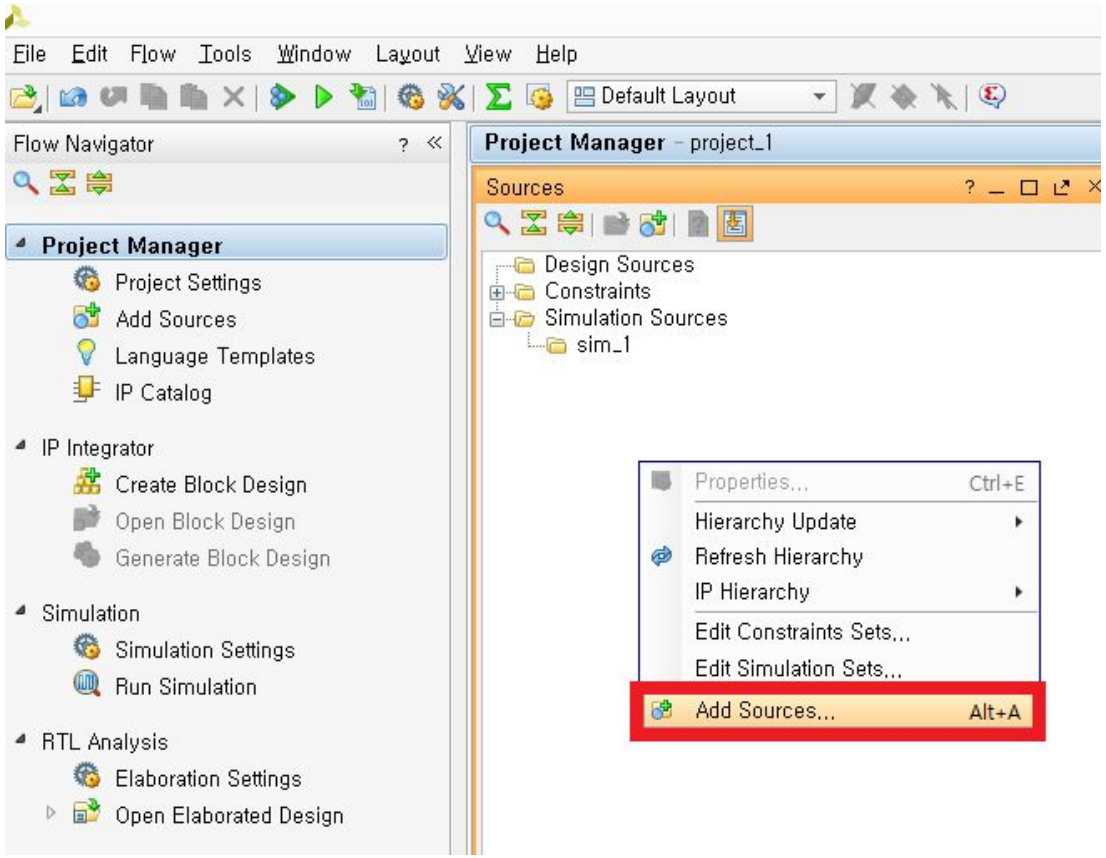
Vivado project creation

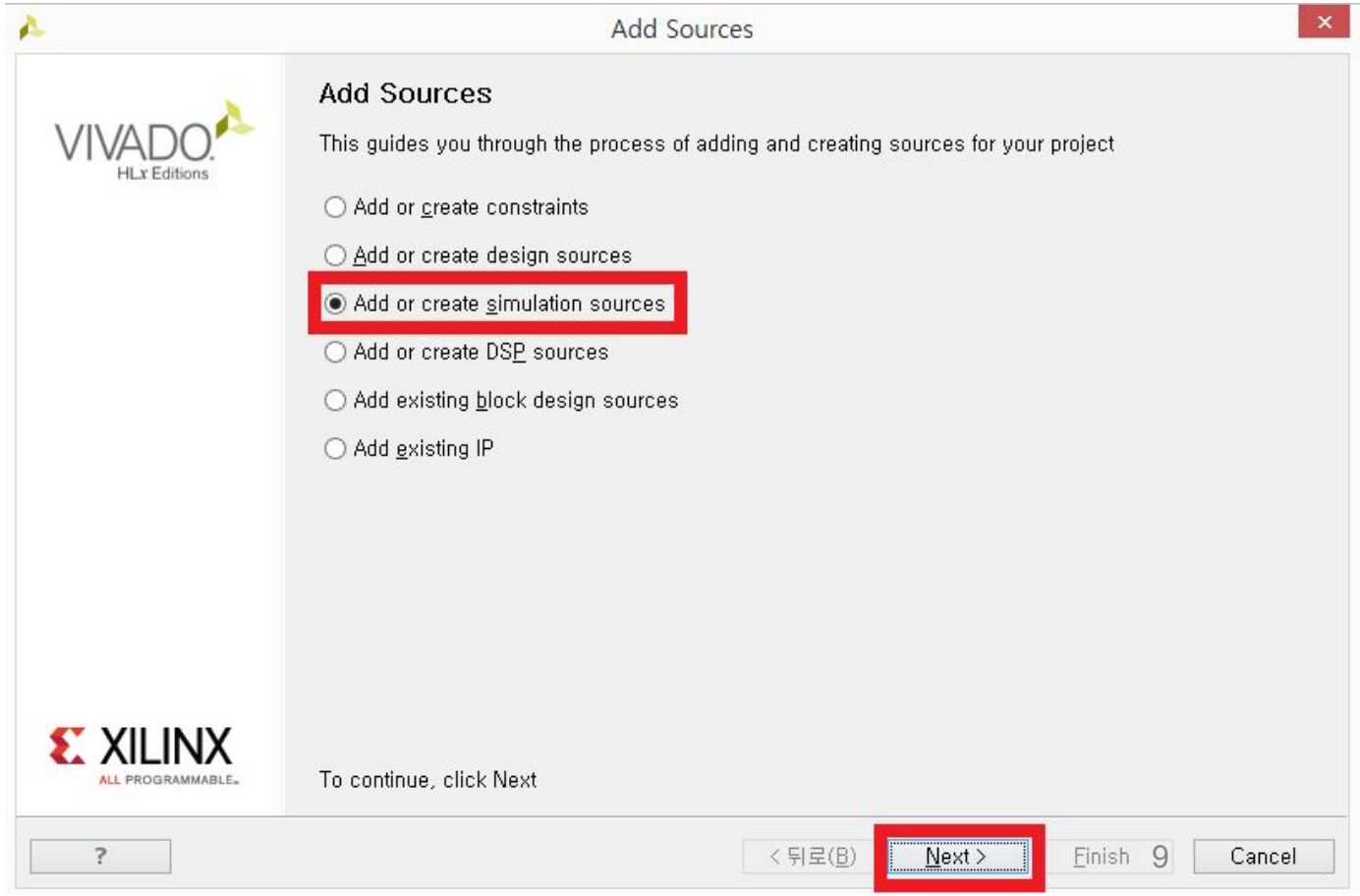
- Finish project creation



Create simulation sources

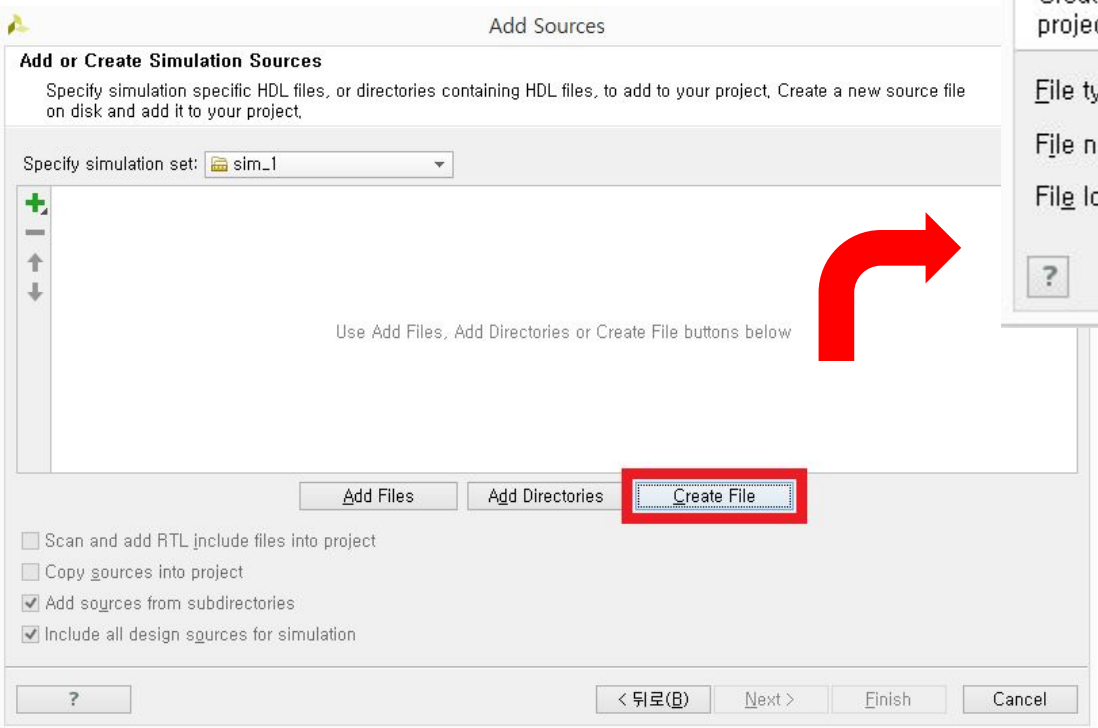
- Create simulation source

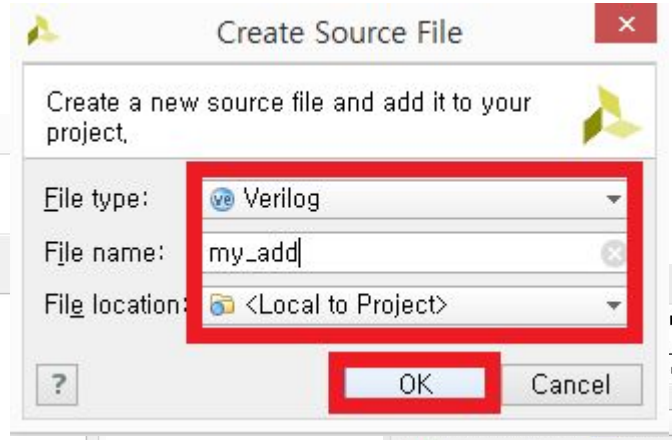


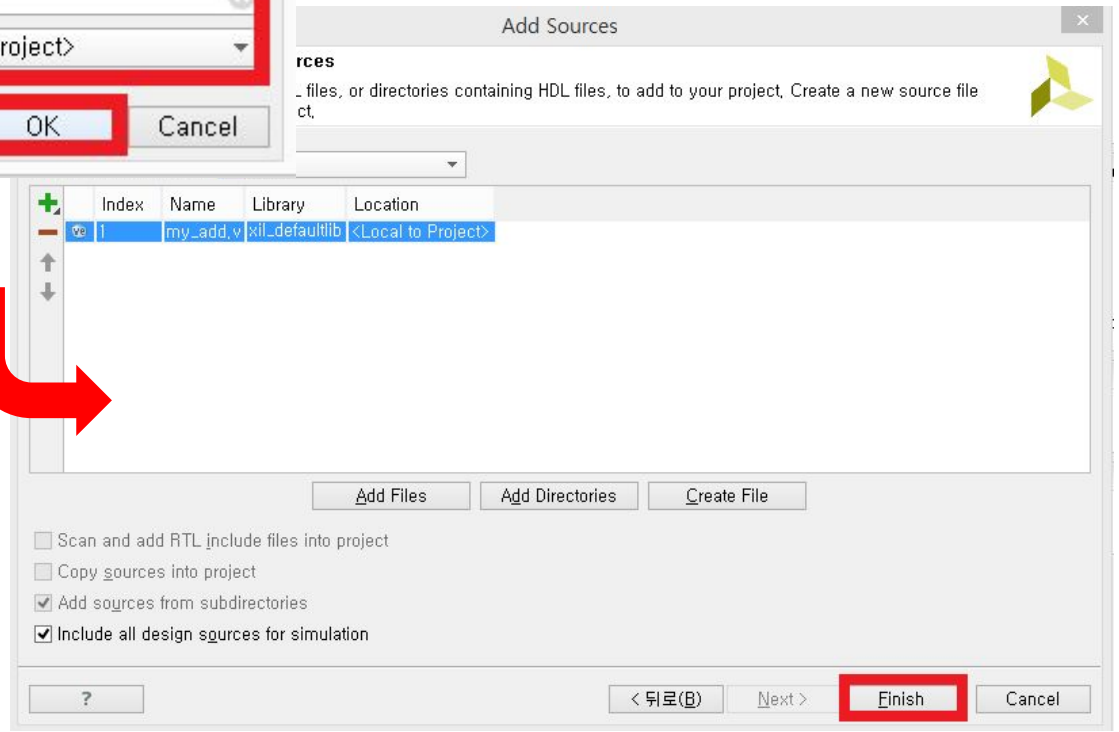


Create simulation sources

- Create simulation source
  - Modules and testbenches

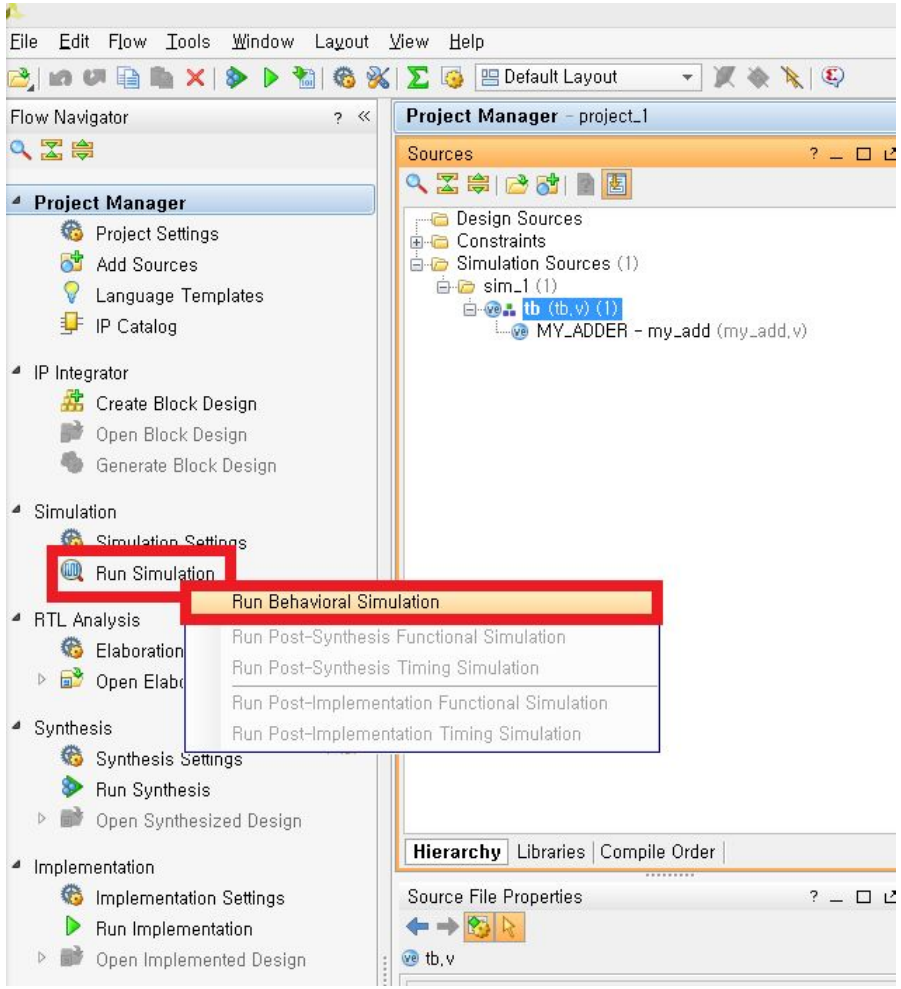






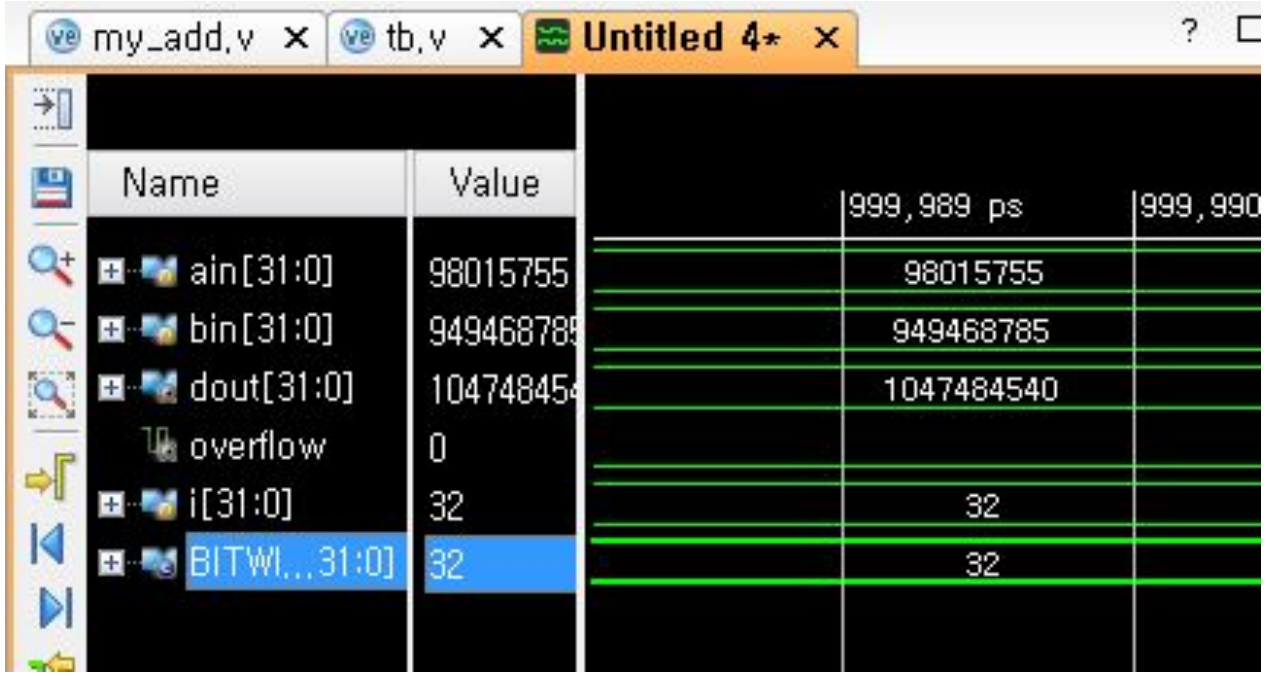
Simulation

- Simulation -> Run Behavioral Simulation



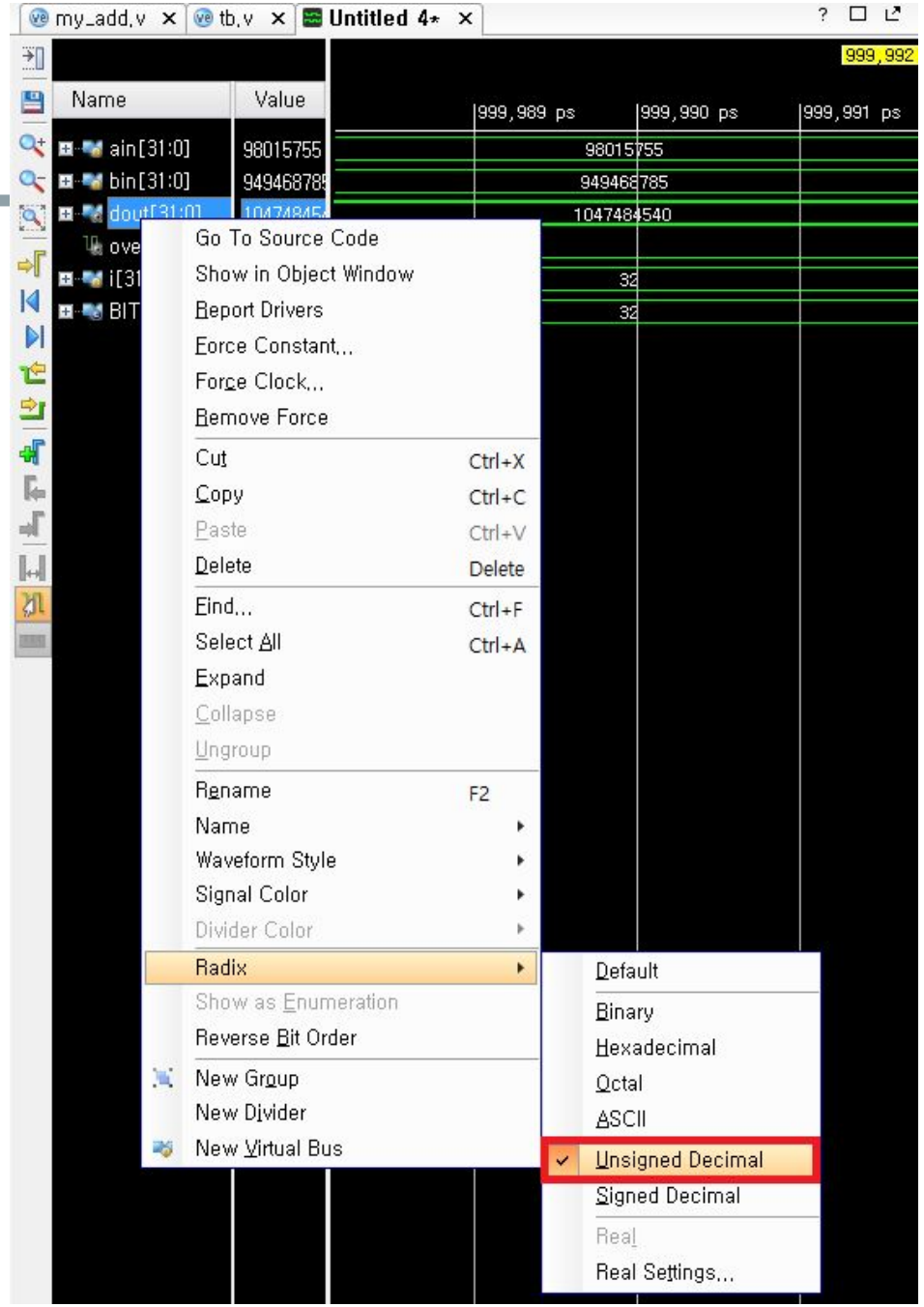
Simulation results

- Check result



Simulation results

- Change radix



Question

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 multiply-and-accumulate (MAC) (default 32 bit)

#1: 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
);
    assign {overflow, dout} = ain + bin;
endmodule
```

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

#2: 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
);
    assign dout = ain * bin;
endmodule
```

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

#3: n-bit Integer MAC (default 32 bit)

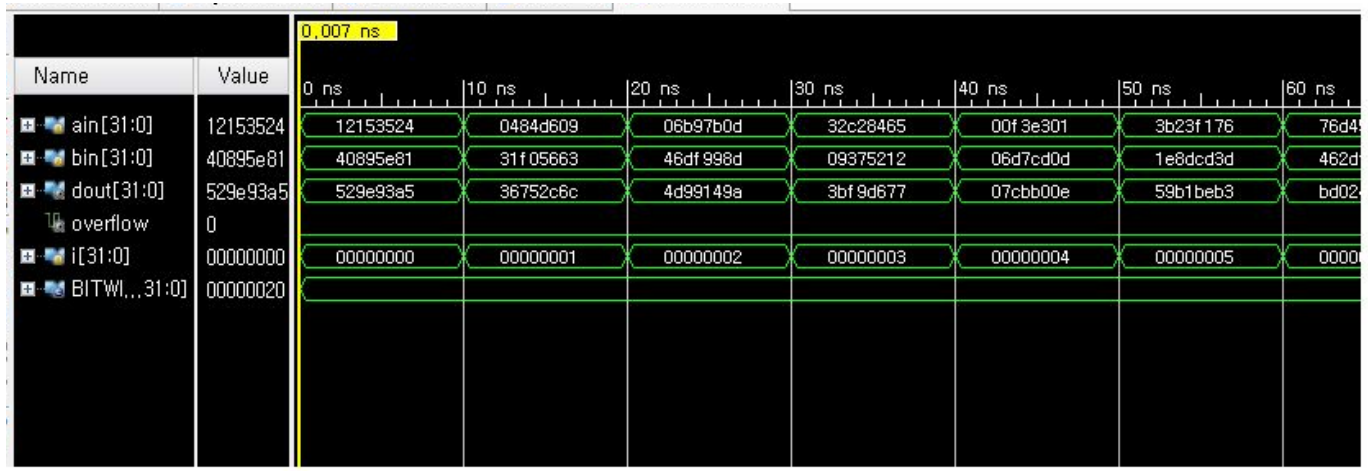
```
module my_mac #(
     parameter BITWIDTH = 32
)
(
     input [BITWIDTH-1:0] ain,
     input [BITWIDTH-1:0] bin,
     input en,
     input clk,
     output [2*BITWIDTH-1:0] dout
);
     reg [2*BITWIDTH-1:0] sum;
     assign dout = sum;

     always @(posedge clk) begin
          if(en) sum <= sum + ain * bin;
          else sum <= 0;
     end
endmodule
```

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

my_adder (Answer #1)

```
`timescale 1ns / 1ps

module tb_add();
    parameter BITWIDTH = 32;

    //for my IP
    reg [BITWIDTH-1:0] ain;
    reg [BITWIDTH-1:0] bin;
    wire [BITWIDTH-1:0] dout;
    wire   overflow;

    //for test
    integer i;
        //random test vector generation
    initial begin
      for(i=0; i<32; i=i+1) begin
        ain = $urandom%(2**31);
        bin = $urandom%(2**31);
        #10;
      end
    end

    //my IP
    my_add #(BITWIDTH) MY_ADDER(
        .ain(ain),
        .bin(bin),
        .dout(dout),
        .overflow(overflow)
    );
endmodule
```

Testbench for **integer adder** module



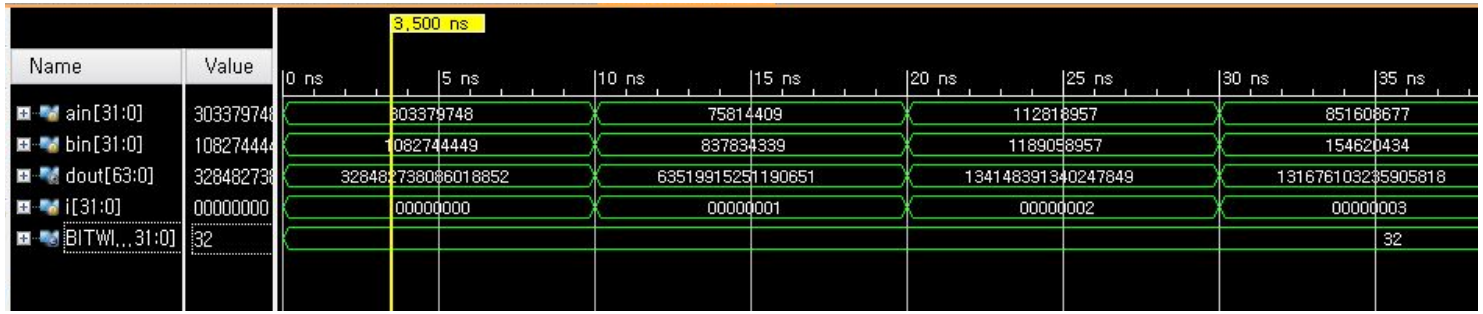
Waveform of randomly generated integers and their **addition** results

my_mul (Answer #2)

```
`timescale 1ns / 1ps

module tb_mul();
    parameter BITWIDTH = 32;

    //for my IP
    reg [BITWIDTH-1:0] ain;
    reg [BITWIDTH-1:0] bin;
    wire [2*BITWIDTH-1:0] dout;

    //for test
    integer i;
        //random test vector generation
```

```
initial begin
  for(i=0; i<32; i=i+1) begin
    ain = $urandom%(2**31);
    bin = $urandom%(2**31);
    #10;
  end
end

//my IP
my_mul #(BITWIDTH) MY_MUL(
    .ain(ain),
    .bin(bin),
    .dout(dout)
);


endmodule
```

Testbench for **integer multiplier** module



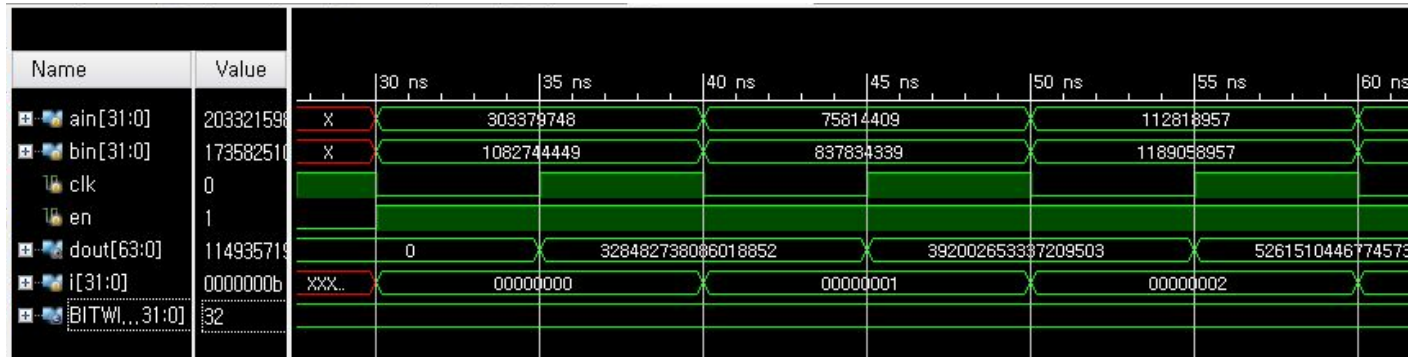
Waveform of randomly generated integers and their **multiplication** results

my_mac (Answer #3)

```
`timescale 1ns / 1ps
module tb_mac();
    parameter BITWIDTH = 32;

    //for my IP
    reg [BITWIDTH-1:0] ain;
    reg [BITWIDTH-1:0] bin;
    reg clk;
    reg en;
    wire [2*BITWIDTH-1:0] dout;


    //for test
    integer i;
        //random test vector generation
```

```
    initial begin
        clk<=0;
        en<=0;
        #30;
        en<=1;
      for(i=0; i<32; i=i+1) begin
        ain = $urandom%(2**31);
        bin = $urandom%(2**31);
        #10;
      end
    end

    //my IP
    my_mac #(BITWIDTH) MY_MAC(
        .ain(ain),
        .bin(bin),
        .en(en),
        .clk(clk),
        .dout(dout)
    );

    always #5 clk = ~clk;

endmodule
```

Testbench for **integer MAC** module



Waveform of randomly generated integers and their **MAC** results

References for Verilog syntax

- ASIC world – Verilog Tutorial (http://www.asic-world.com/verilog/index.html)
- Verilog HDL: A Guide to Digital Design and Synthesis, 2nd Ed. (http://freecomputerbooks.com/Verilog-HDL-A-Guide-to-Digital-Design-and-Synthesis.html)