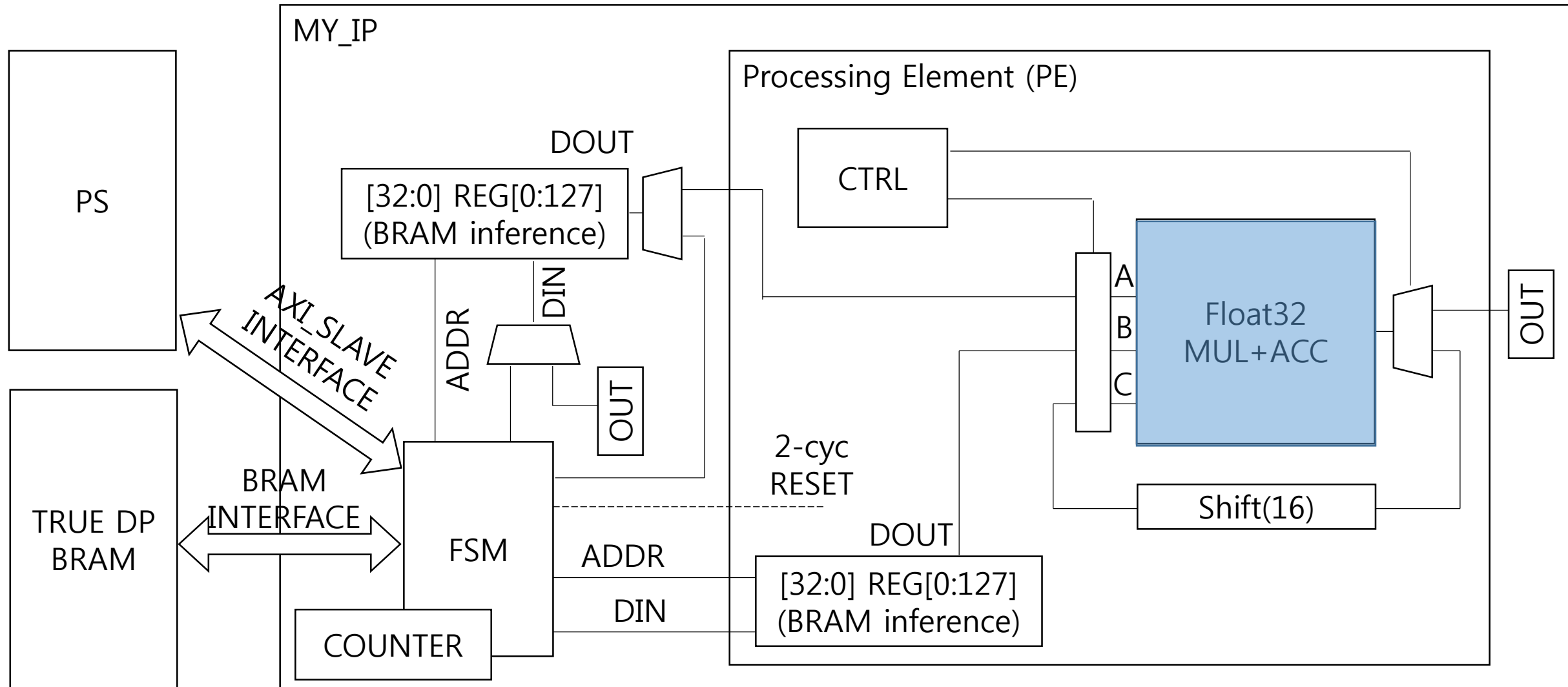


Lab #2: Using IP Catalog and Synthesis

03/15/2018

4190.309A: Hardware System Design
(Spring 2018)

Custom HW Example: Matrix Multiplication IP



Practice #1

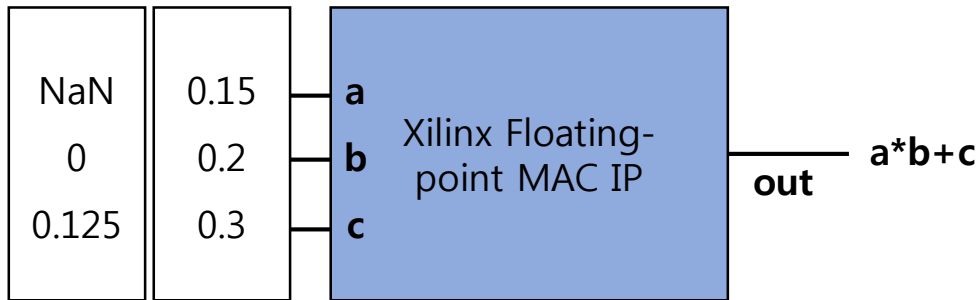
Practice #1. Using Vivado IP catalog

- Using IP catalog of Vivado, implement following two blocks and synthesize them. Write down a given test bench (next slide) and show the waveform.
1. Design 32-bit integer MAC
 - Use Xilinx MAC IP ("multiply adder" IP in the IP catalog)
 2. Design 32-bit floating point MAC
 - Use Xilinx floating point IP ("floating point" IP in the IP catalog)
 - Write-down a test bench that **checks following two test cases**
 - 1) $a=0.15$ (0x3e19999a), $b=0.2$ (0x3e4ccccd), $c=0.3$ (0x40400000)
 - 2) $a=\text{NaN}$ (0x7fcccccd), $b=0$ (0x0), $c=0.125$ (0x3e00000)
 - Hex numbers in parenthesis are IEEE-754 representation

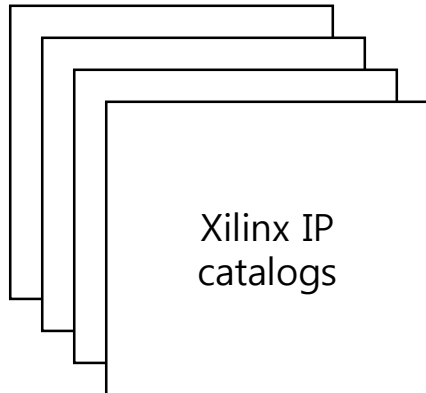
Practice #1. Using Vivado IP catalog

- Write test benches for IP (provided)

Test bench ((1) **test_bench.v: Write your TB by modifying the sample**)



(2) Load floating-point MAC IP from IP catalogs (See Slides 12~19)



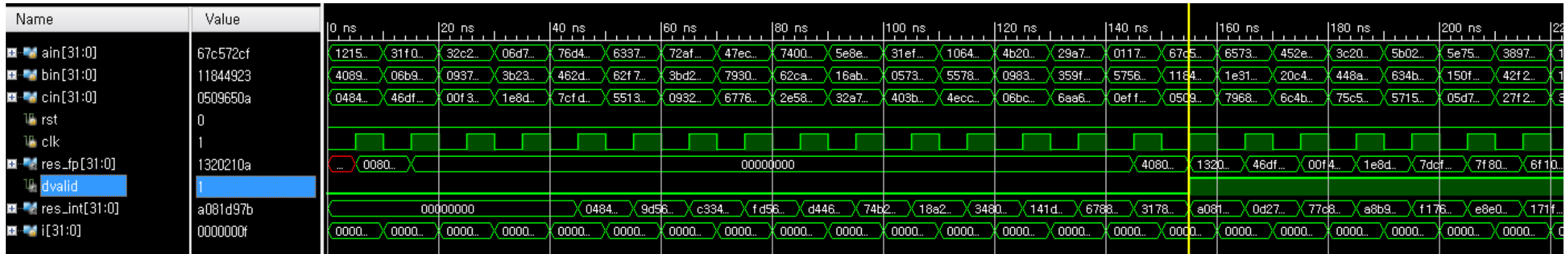
```
1 `timescale 1ns / 1ps
2
3 module tb();
4
5     //for my IP
6     reg [32-1:0] ain;
7     reg [32-1:0] bin;
8     reg [32-1:0] cin;
9     reg rst;
10    reg clk;
11    wire [32-1:0] res_fp, res_int;
12    wire dvalid;
13
14    //for test
15    integer i;
16    //random test vector generation
17    initial begin
18        clk<=0;
19        rst<=0;
20        for(i=0; i<32; i=i+1) begin
21            ain = $urandom%(2**31);
22            bin = $urandom%(2**31);
23            cin = $urandom%(2**31);
24            #10;
25        end
26    end
27
28    always #5 clk = ~clk;
```

```
29
30 floating_point_MAC fpUUT(
31     .aclk(clk),
32     .aresetn(~rst),
33     .s_axis_a_tvalid(1'b1),
34     .s_axis_b_tvalid(1'b1),
35     .s_axis_c_tvalid(1'b1),
36     .s_axis_a_tdata(ain),
37     .s_axis_b_tdata(bin),
38     .s_axis_c_tdata(cin),
39     .m_axis_result_tvalid(dvalid),
40     .m_axis_result_tdata (res_fp)
41 );
42
43
44
45
46
47
48
49
50
51 endmodule
52
```

Sample test_bench.v

Practice #1. Using Vivado IP catalog

- Check results with simulation



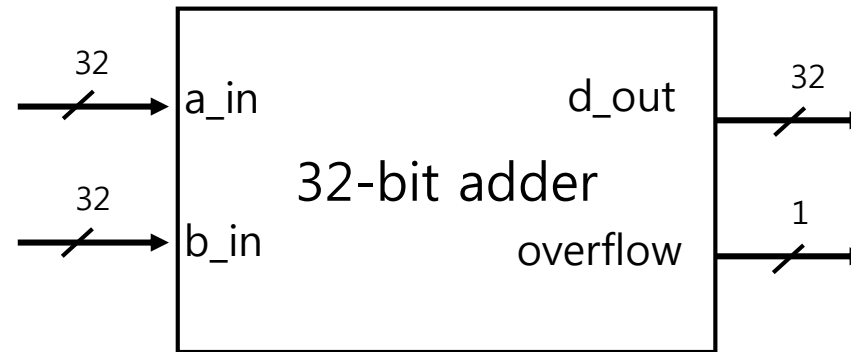
Practice #2

Practice #2. Design a 32b adder array

- Using for-generate statement and adder module (32 bit based) that you implemented last week, implement an “adder array” consisting of 4 adders. Design your own test bench and show the wave form (test 4 random vectors for each ‘cmd’).
 - Simulation, Synthesis

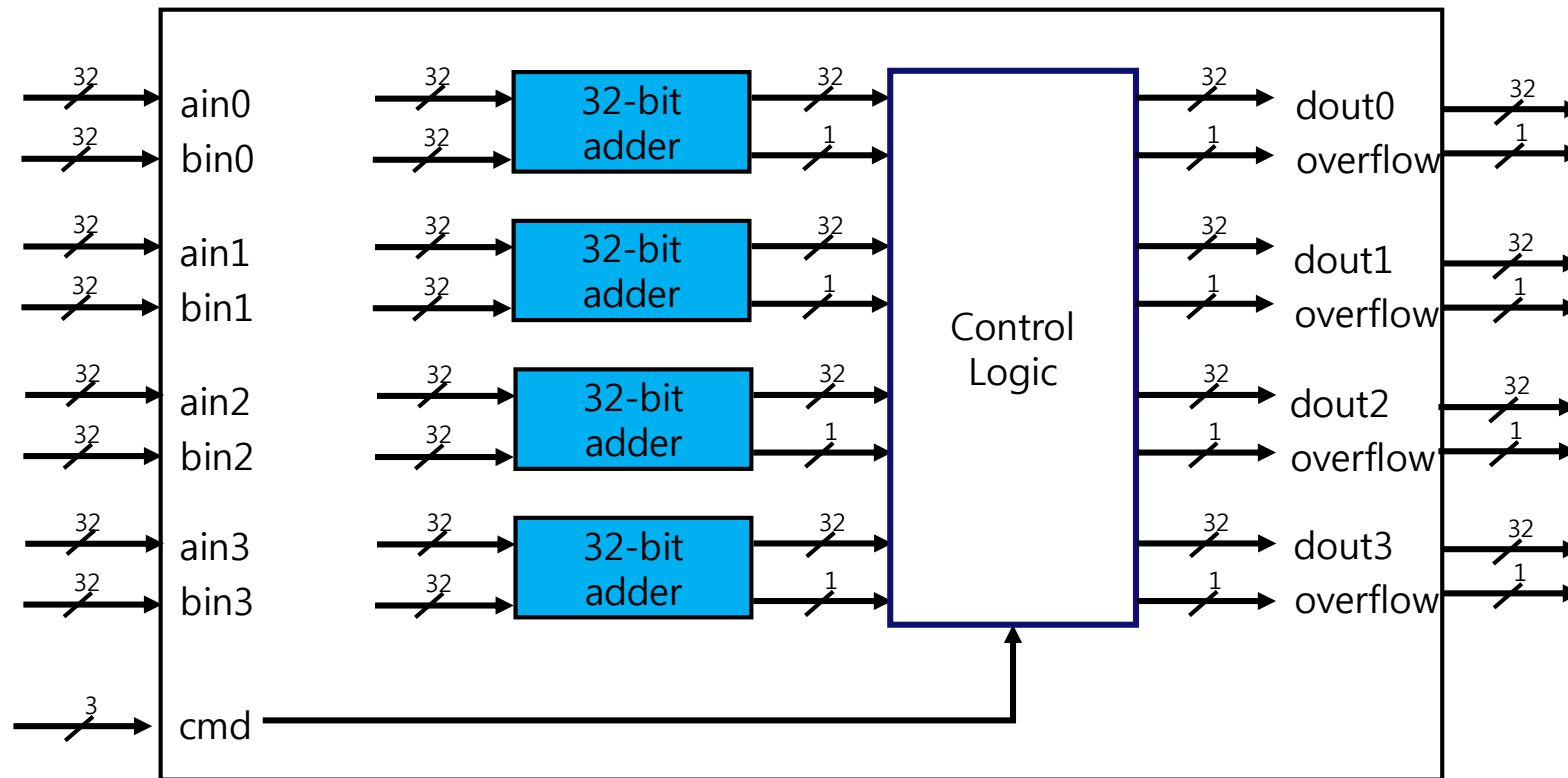
Practice #2. Design a 32b adder array

- In the last lab, we designed 32-bit integer ADD/MUL/MAC units



Practice #2. Design a 32b adder array

- Using your adder, implement vector adder (4 input / output)
- **Note:** "cmd" signal for output configuration (See next slide for details)



Practice #2. Design a 32b adder array

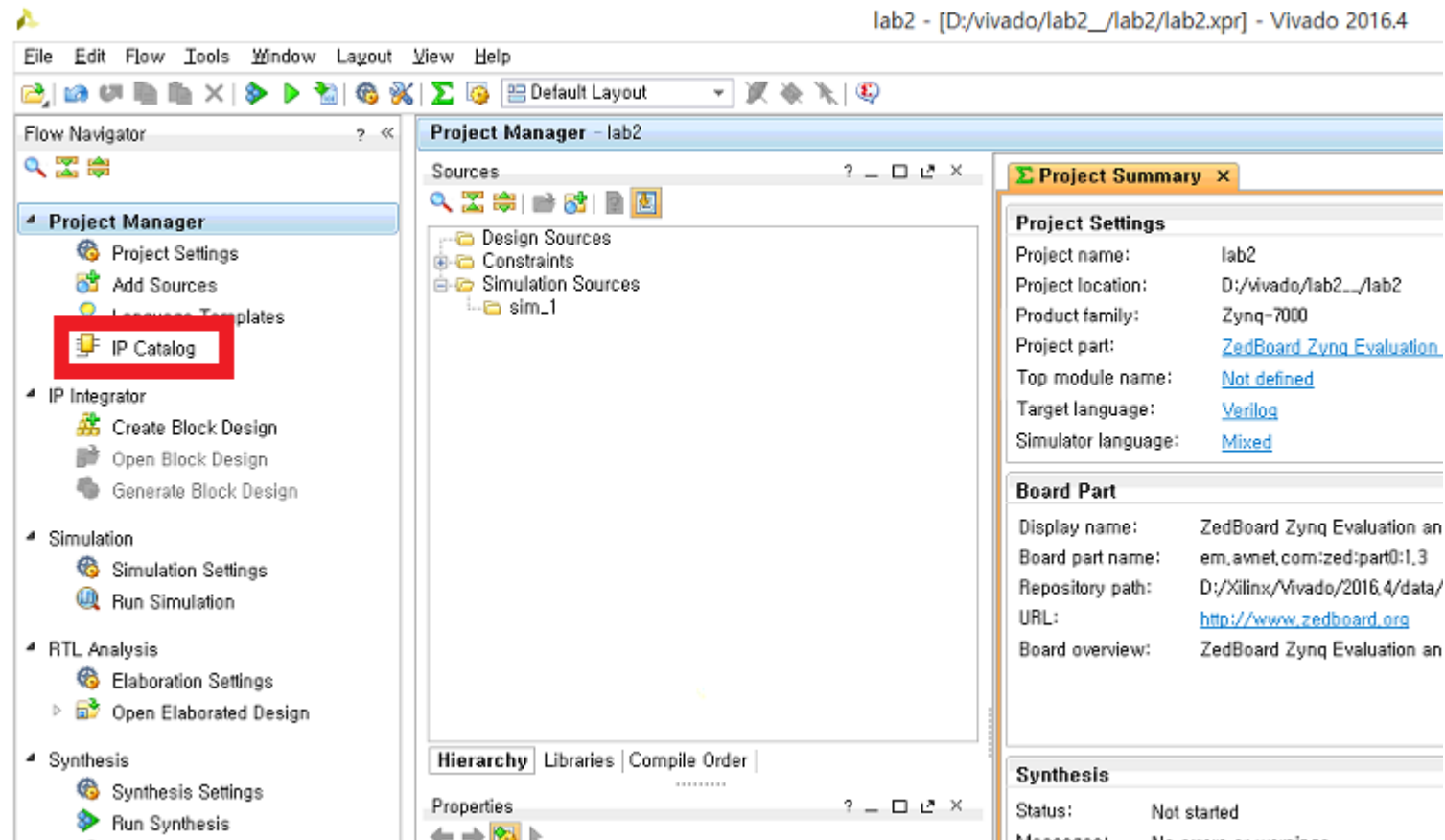
```
module adder_array
(cmd,
ain0, ain1, ain2, ain3,
bin0, bin1, bin2, bin3,
dout0, dout1, dout2, dout3,
overflow);
    input [2:0] cmd;
    input [31:0] ain0, ain1, ain2, ain3;
    input [31:0] bin0, bin1, bin2, bin3;
    output [31:0] dout0, dout1, dout2, dout3;
    output [3:0] overflow;
    ... WRITE YOUR LOGIC HERE ...
endmodule
```

- **cmd:** operating mode
 - ==0, output port **dout0** shows **ain0+bin0**, and the others show 0.
 - ==1, output port **dout1** shows **ain1+bin1**, and the others show 0.
 - ==2, output port **dout2** shows **ain2+bin2**, and the others show 0.
 - ==3, output port **dout3** shows **ain3+bin3**, and the others show 0.
 - ==4, every output port show its own addition result.
- **ain:** 1st operand
- **bin:** 2nd operand
- **dout:** mac result
- **overflow:** ==1, if overflow is detected; ==0, otherwise. **overflow[i]** is overflow bit for **dout_i** (e.g., **overflow[1]** is overflow bit for **dout1**)

How to Use IP Catalog (for Practice #1)

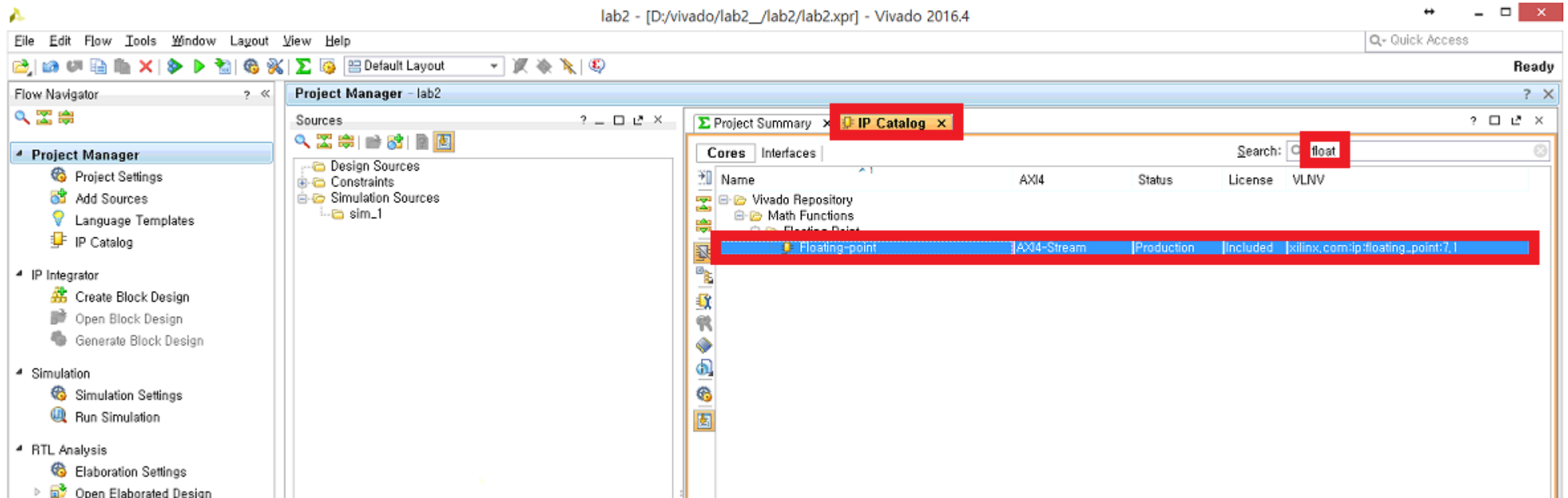
How to use IP catalog

■ Using IP Catalog



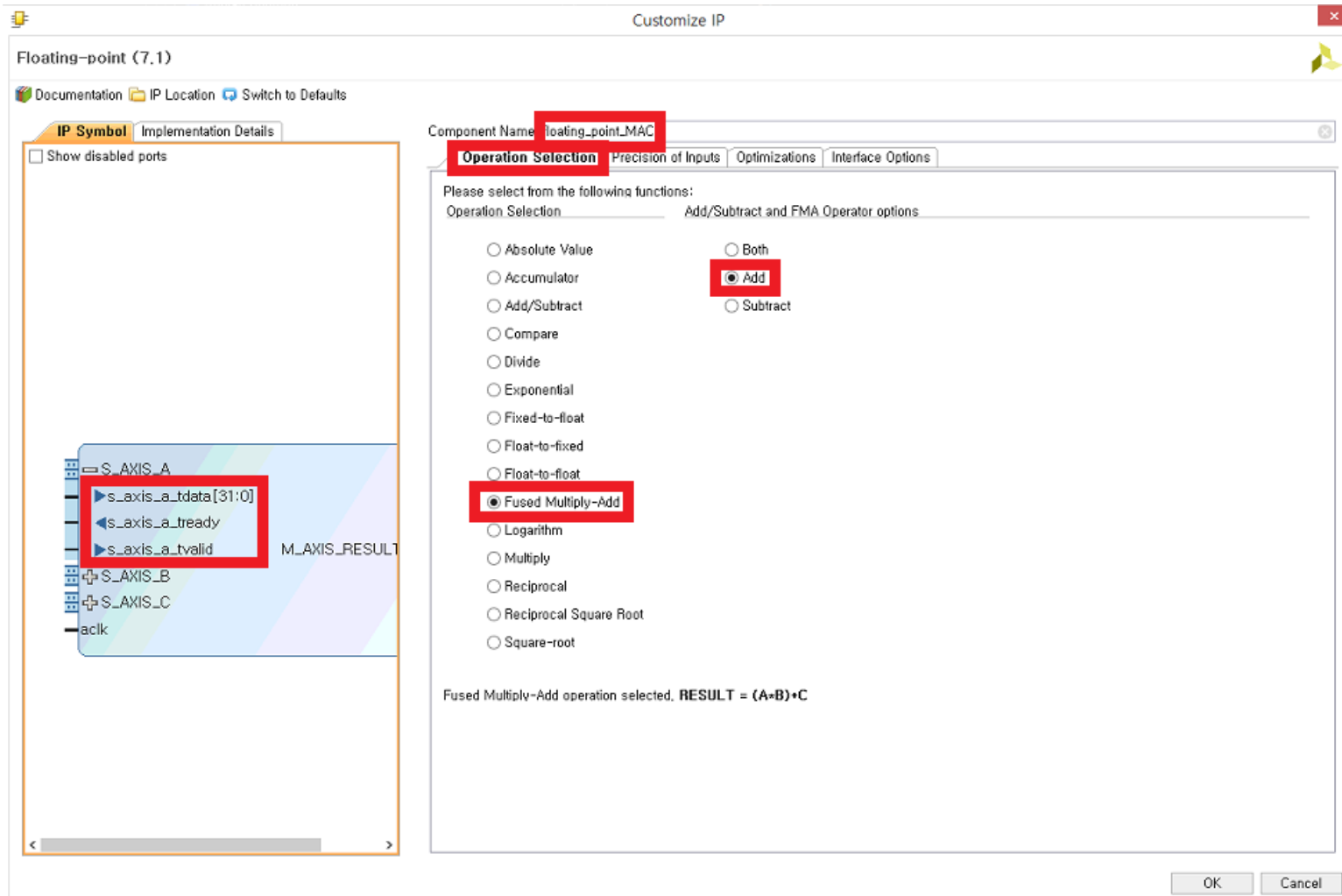
How to use IP catalog

- Search floating point IP
 - Floating-point(for fp) / Multiply adder(for int)



How to use IP catalog

■ Configuration



How to use IP catalog

■ Remove redundant pins

The image shows the Xilinx IP Customization tool interface for the 'floating_point (7.1)' IP. The 'Interface Options' tab is selected, and the 'Flow Control Options' section is expanded. The 'Flow Control' dropdown is set to 'NonBlocking', and the 'RESULT' checkbox is checked. The 'Latency and Rate Configuration' section shows 'Use Maximum Latency' checked, with 'Latency' set to 17 and 'Cycles/operation' set to 1. The 'Control Signals' section shows 'ARESETn (active low)' checked. The 'Optional Output Fields' section shows 'UNDERFLOW', 'OVERFLOW', 'DIVIDE BY ZERO', and 'ACCUM OVERFLOW' all unchecked. The 'Channel' table shows 'A', 'B', and 'C' channels, all with 'Has TLAST' checked. The 'TLAST Behavior' is set to 'Null'.

The 'Generate Output Products' dialog is open, showing the following output products will be generated:

- floating_point_0.xci (OOC per IP)
- Instantiation Template
- Synthesized Checkpoint (.dcp)
- Structural Simulation
- C Simulation
- Test Bench
- Change Log

The 'Synthesis Options' section shows 'Global' selected. The 'Run Settings' section shows 'Number of jobs' set to 2. The 'Generate' button is highlighted.

The 'Customize IP' dialog shows the 'Interface Options' tab selected. The 'Flow Control Options' section is expanded, showing 'Flow Control' set to 'NonBlocking' and 'RESULT' checked. The 'Latency and Rate Configuration' section shows 'Use Maximum Latency' checked, with 'Latency' set to 11 and 'Cycles/operation' set to 1. The 'Control Signals' section shows 'ARESETn (active low)' checked. The 'Optional Output Fields' section shows 'UNDERFLOW', 'OVERFLOW', 'INVALID OP', 'DIVIDE BY ZERO', 'ACCUM OVERFLOW', and 'ACCUM INPUT OVERFLOW' all unchecked. The 'Channel' table shows 'A', 'B', and 'C' channels, all with 'Has TLAST' checked. The 'TLAST Behavior' is set to 'Null'.

The 'Generate Output Products' dialog is open, showing the following output products will be generated:

- floating_point_0.xci (OOC per IP)
- Instantiation Template
- Synthesized Checkpoint (.dcp)
- Structural Simulation
- C Simulation
- Test Bench
- Change Log

The 'Synthesis Options' section shows 'Global' selected. The 'Run Settings' section shows 'Number of jobs' set to 2. The 'Generate' button is highlighted.

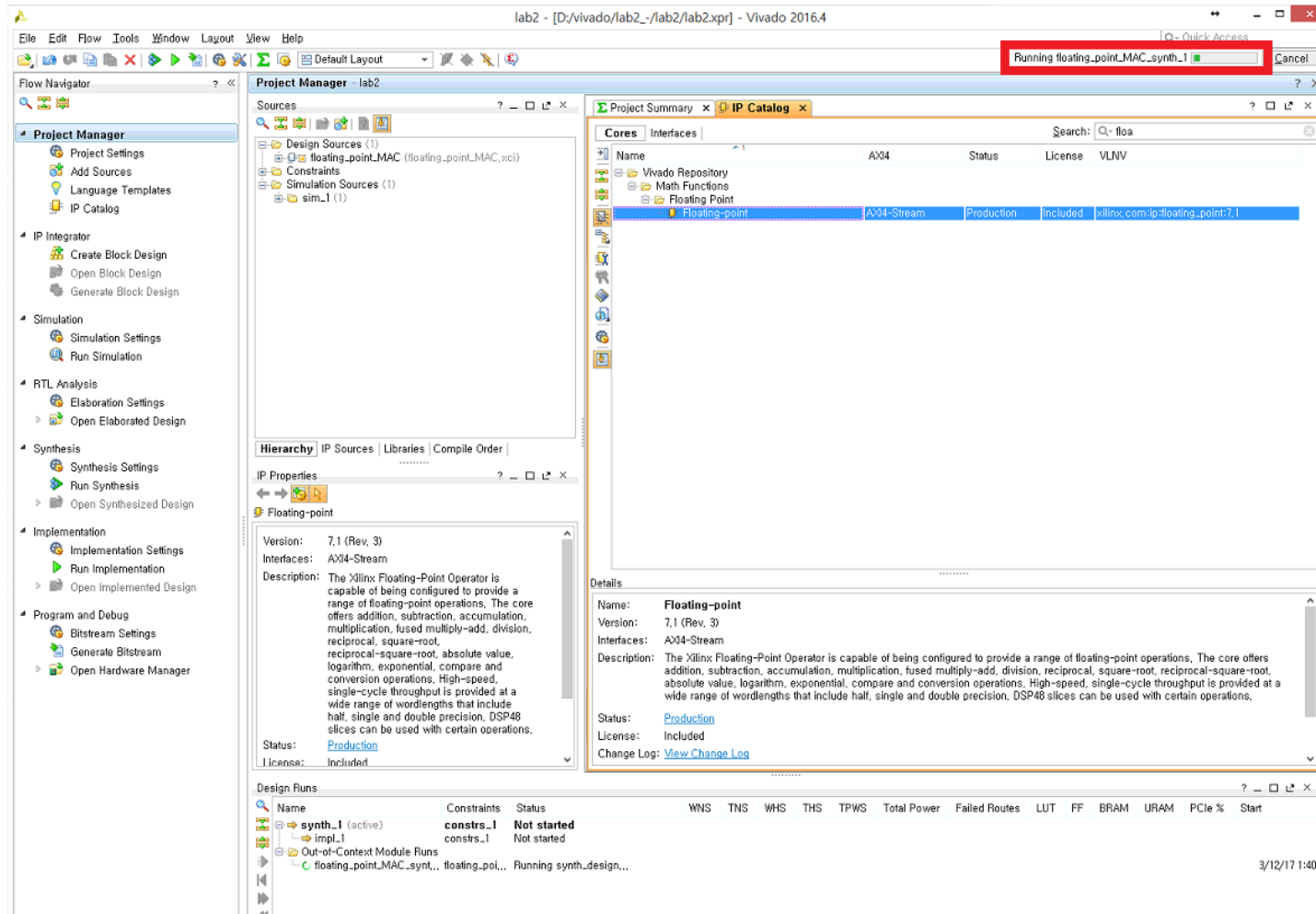
The 'Customize IP' dialog shows the 'Interface Options' tab selected. The 'Flow Control Options' section is expanded, showing 'Flow Control' set to 'NonBlocking' and 'RESULT' checked. The 'Latency and Rate Configuration' section shows 'Use Maximum Latency' checked, with 'Latency' set to 11 and 'Cycles/operation' set to 1. The 'Control Signals' section shows 'ARESETn (active low)' checked. The 'Optional Output Fields' section shows 'UNDERFLOW', 'OVERFLOW', 'INVALID OP', 'DIVIDE BY ZERO', 'ACCUM OVERFLOW', and 'ACCUM INPUT OVERFLOW' all unchecked. The 'Channel' table shows 'A', 'B', and 'C' channels, all with 'Has TLAST' checked. The 'TLAST Behavior' is set to 'Null'.

The 'Generate Output Products' dialog is open, showing the following output products will be generated:

- floating_point_0.xci (OOC per IP)
- Instantiation Template
- Synthesized Checkpoint (.dcp)
- Structural Simulation
- C Simulation
- Test Bench
- Change Log

The 'Synthesis Options' section shows 'Global' selected. The 'Run Settings' section shows 'Number of jobs' set to 2. The 'Generate' button is highlighted.

- Generate customized floating point MAC



Generating floating point MAC

■ Run Synthesis (Optional)

The screenshot displays the Vivado 2016.4 IDE interface for a project named 'lab2'. The main window shows the 'Sources' pane with the project hierarchy, including 'top_lab2.v' and 'floating_point_MAC.xci'. The 'Project Manager' pane on the left shows the 'Run Synthesis' option under the 'Synthesis' section, which is highlighted with a red box. The 'Launch Runs' dialog box is open, showing the 'Launch directory' and 'Options' (Launch runs on local host, Number of jobs: 2). The 'OK' button is highlighted with a red box. The 'Design Runs' table at the bottom shows the synthesis status.

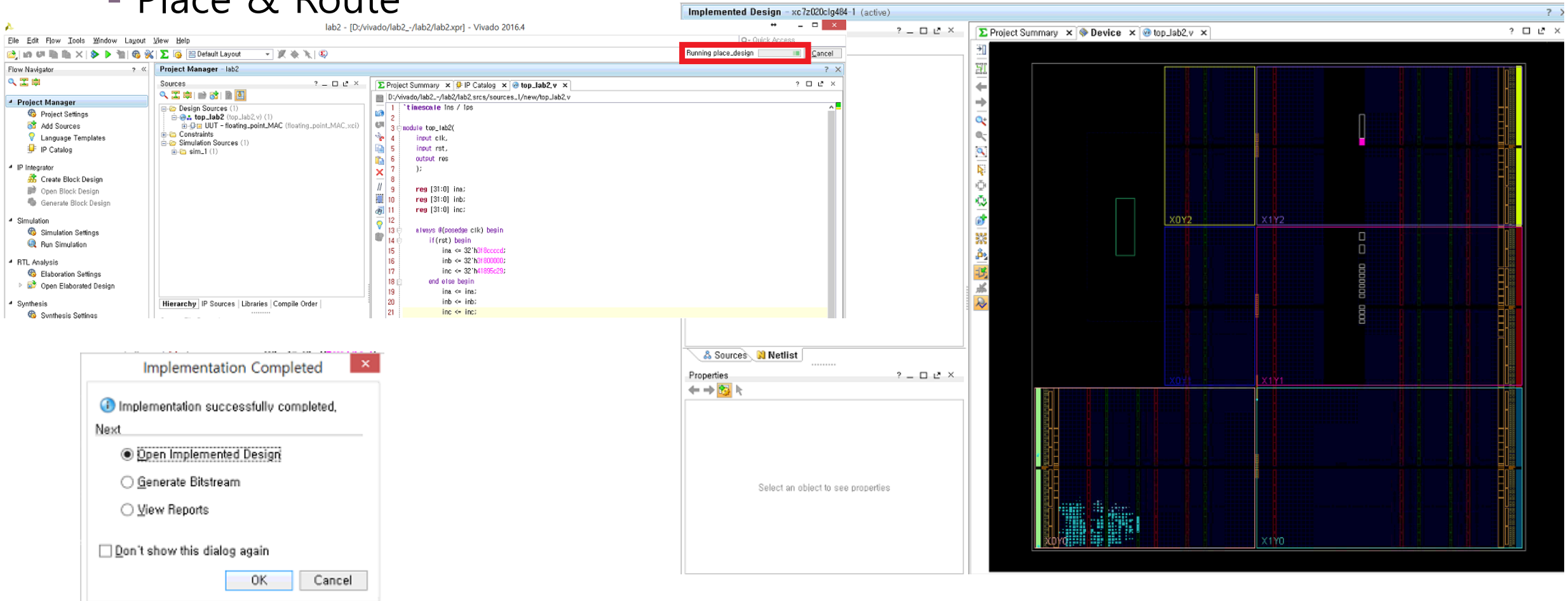
Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes	LUT	FF	BRAM	URAM	PCle %	Start
synth_1 (active)	constraints_1	Not started													
impl_1	constraints_1	Not started													
Out-of-Context Module Runs															
floating_point_MAC.synth_1	floating_point_MAC.constraints_1	synth_design Complete!								193	315	0	0	0.000	3/12/17 1:40

The 'Design Runs' table shows the synthesis status for the project. The 'floating_point_MAC.synth_1' entry is marked as 'synth_design Complete!'. The 'Design Runs' table also includes columns for WNS, TNS, WHS, THS, TPWS, Total Power, Failed Routes, LUT, FF, BRAM, URAM, PCle %, and Start time.

The 'Synthesis Completed' dialog box is also visible, showing the message 'Synthesis successfully completed.' and the 'Next' options: 'Run Implementation' (selected), 'Open Synthesized Design', and 'View Reports'. The 'OK' button is highlighted with a red box.

Generating floating point MAC

- Implementation (Optional)
 - Place & Route



Notice

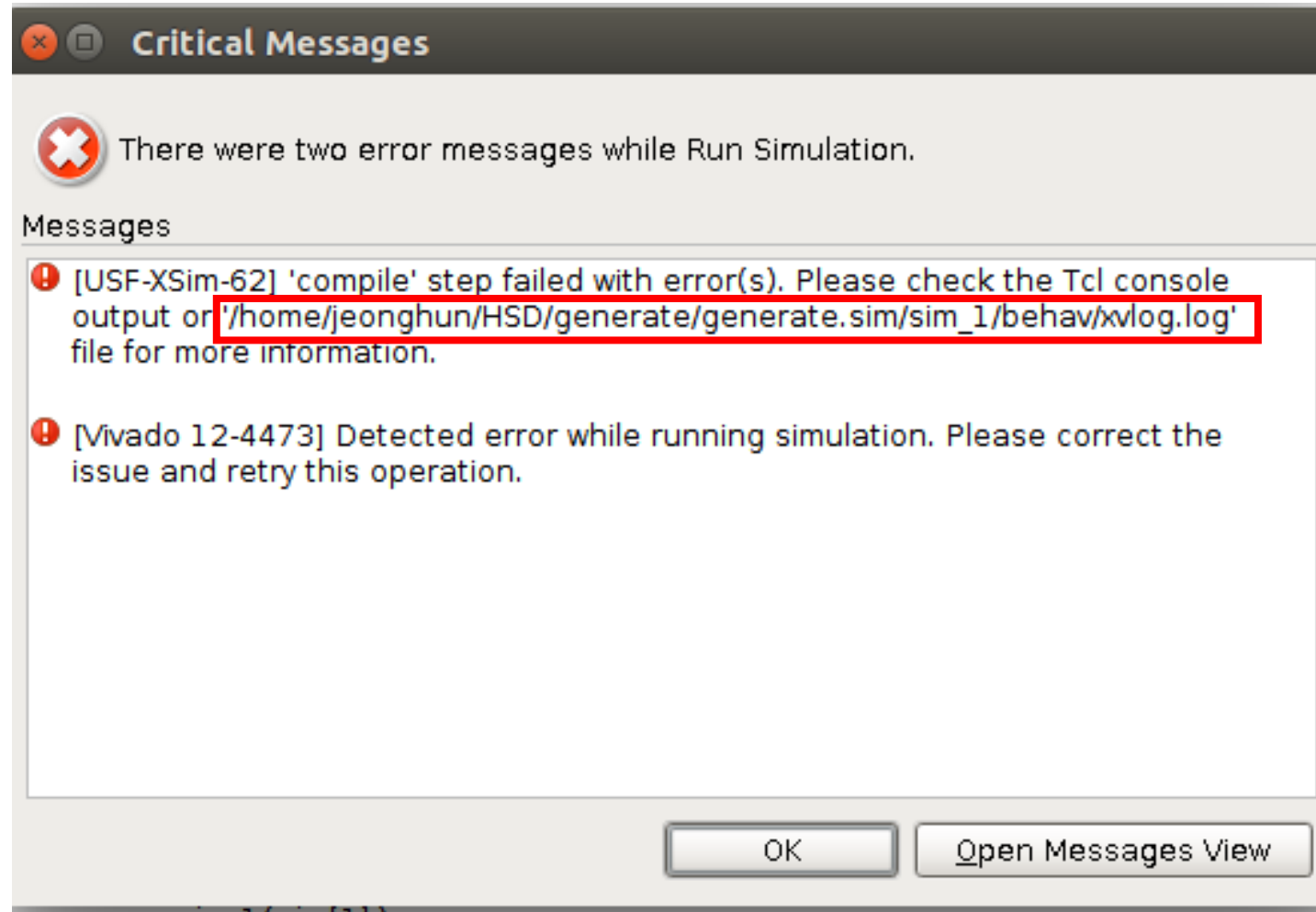
Notice: Clean Up after the Lab

- Please clean up your desk and put your chairs back - TAs will be very happy! 😊 😊 😊 😊



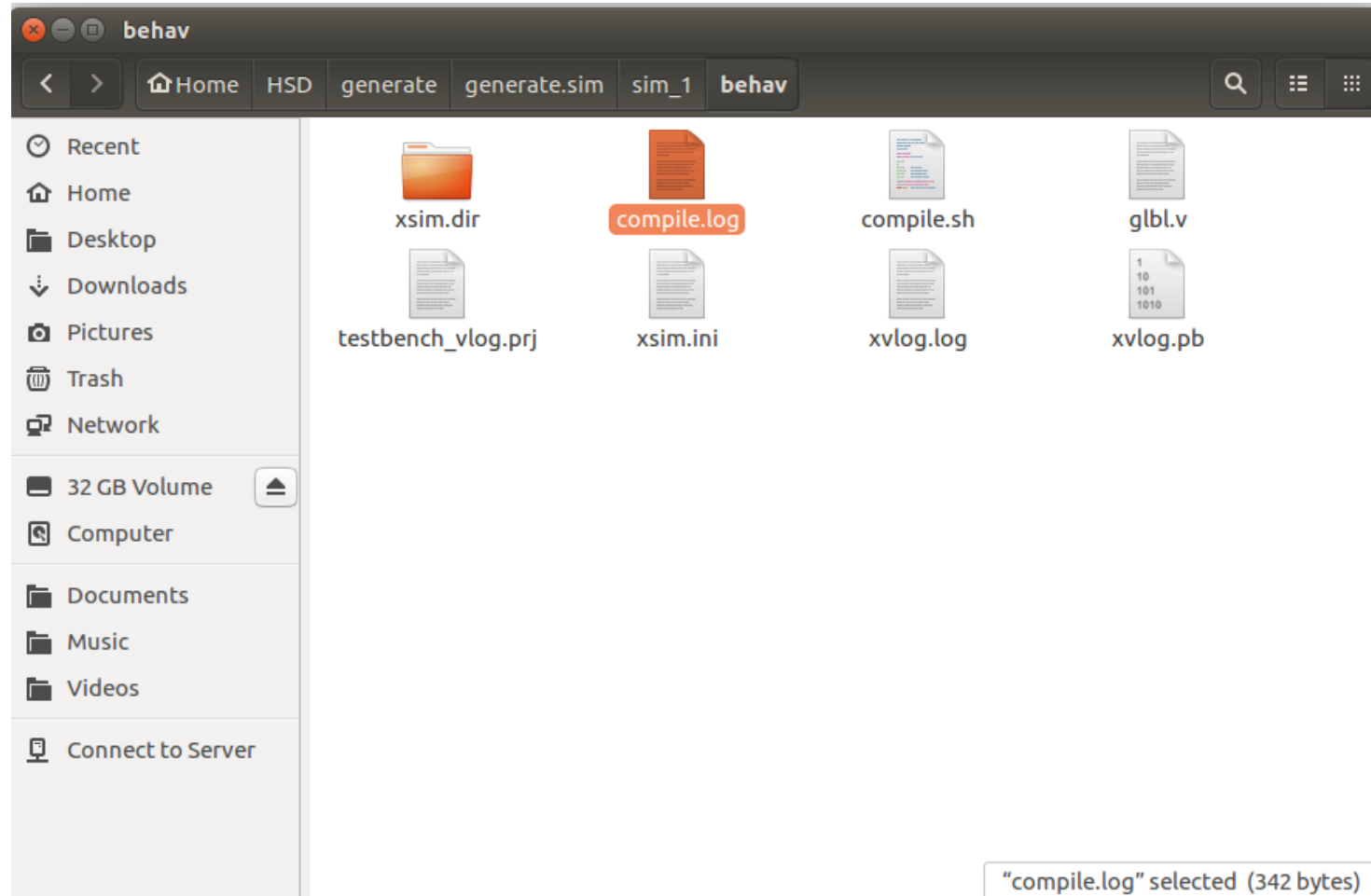
Tips for Debugging

- Check the log file to understand error descriptions



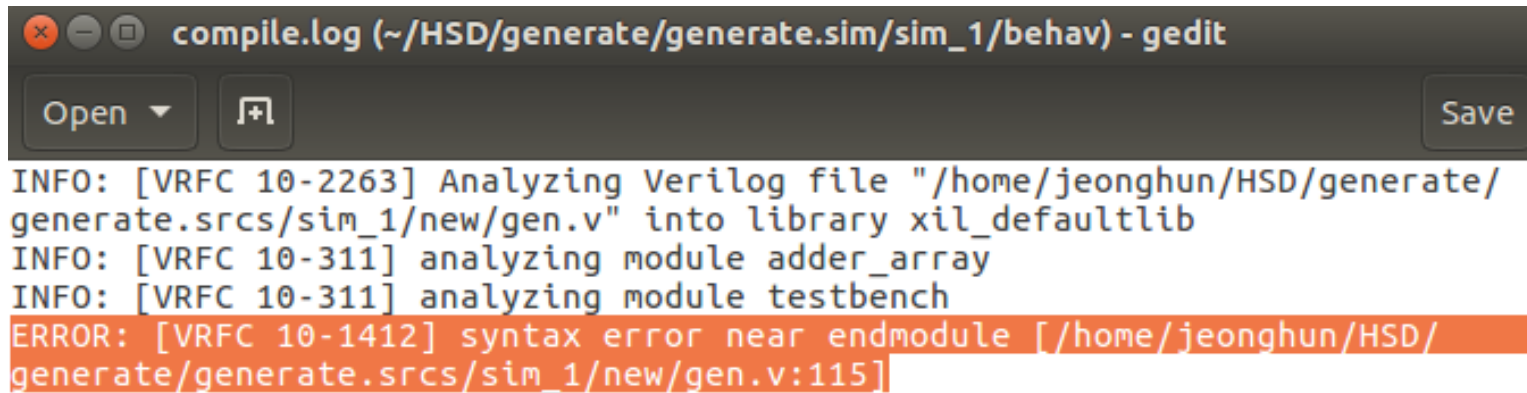
Tips for Debugging

- Locate the log file on file browser



Tips for Debugging

- Check error messages in the log file



```
compile.log (~/HSD/generate/generate.sim/sim_1/behav) - gedit
Open ▾ [icon] Save
INFO: [VRFC 10-2263] Analyzing Verilog file "/home/jeonghun/HSD/generate/
generate.srscs/sim_1/new/gen.v" into library xil_defaultlib
INFO: [VRFC 10-311] analyzing module adder_array
INFO: [VRFC 10-311] analyzing module testbench
ERROR: [VRFC 10-1412] syntax error near endmodule [ /home/jeonghun/HSD/
generate/generate.srscs/sim_1/new/gen.v:115]
```