Practice 4

- How to use IP catalog & Synthesize

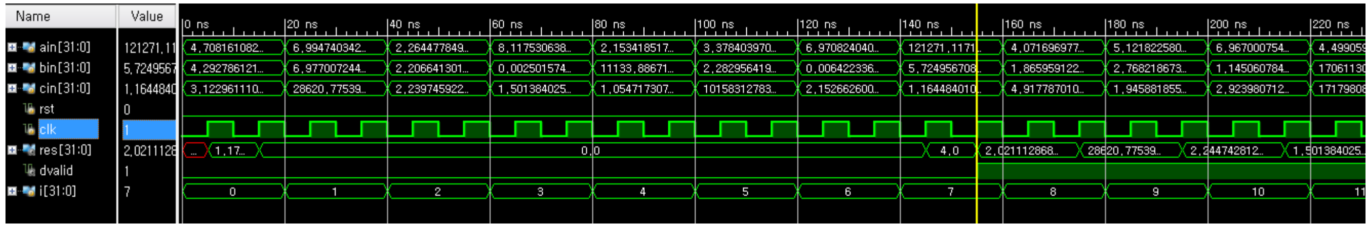
Computing Memory Architecture Lab.

Prototyping HW module

① Design your HW module
- Verilog files

```
module my_fusedmult #(
        parameter BITWIDTH = 32
)
```

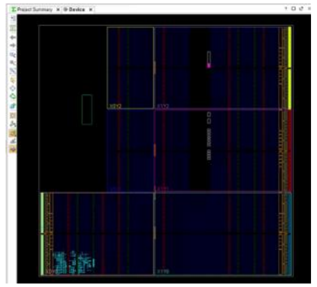
② Behavioral simulation
- Run sim with tb
- Functionality check



```
Name        Value       |0 ns          |20 ns         |40 ns         |60 ns         |80 ns         |100 ns        |120 ns        |140 ns        |160 ns        |180 ns        |200 ns        |220 ns
ain[31:0]   121271,11    4,708161082..  6,994740342..  2,264477849..  8,117530638..  2,153418517..  3,378403970..  6,970824040..  121271,1171..  4,071696977..  5,121822580..  6,967000754..  4,499059
bin[31:0]   5,7249567    4,292786121..  6,977007244..  2,206641301..  0,002501574..  11133,88671..  2,282956419..  0,006422336..  5,724956708..  1,865959122..  2,768218673..  1,145060784..  17061130
cin[31:0]   1,1644840    3,122961110..  28620,77539..  2,239745922..  1,501384025..  1,054717307..  10158312783..  2,152662600..  1,164484010..  4,917787010..  1,945881855..  2,923980712..  17179808
rst         0
clk         1
res[31:0]   2,0211128    ... 1,17..     0,0                                                                              4,0  2,021112868..  28620,77539..  2,244742812..  1,501384025..
dvalid      1
i[31:0]     7            0              1              2              3              4              5              6              7              8              9              10             11
```

③ Synthesis
- Compile into HW level

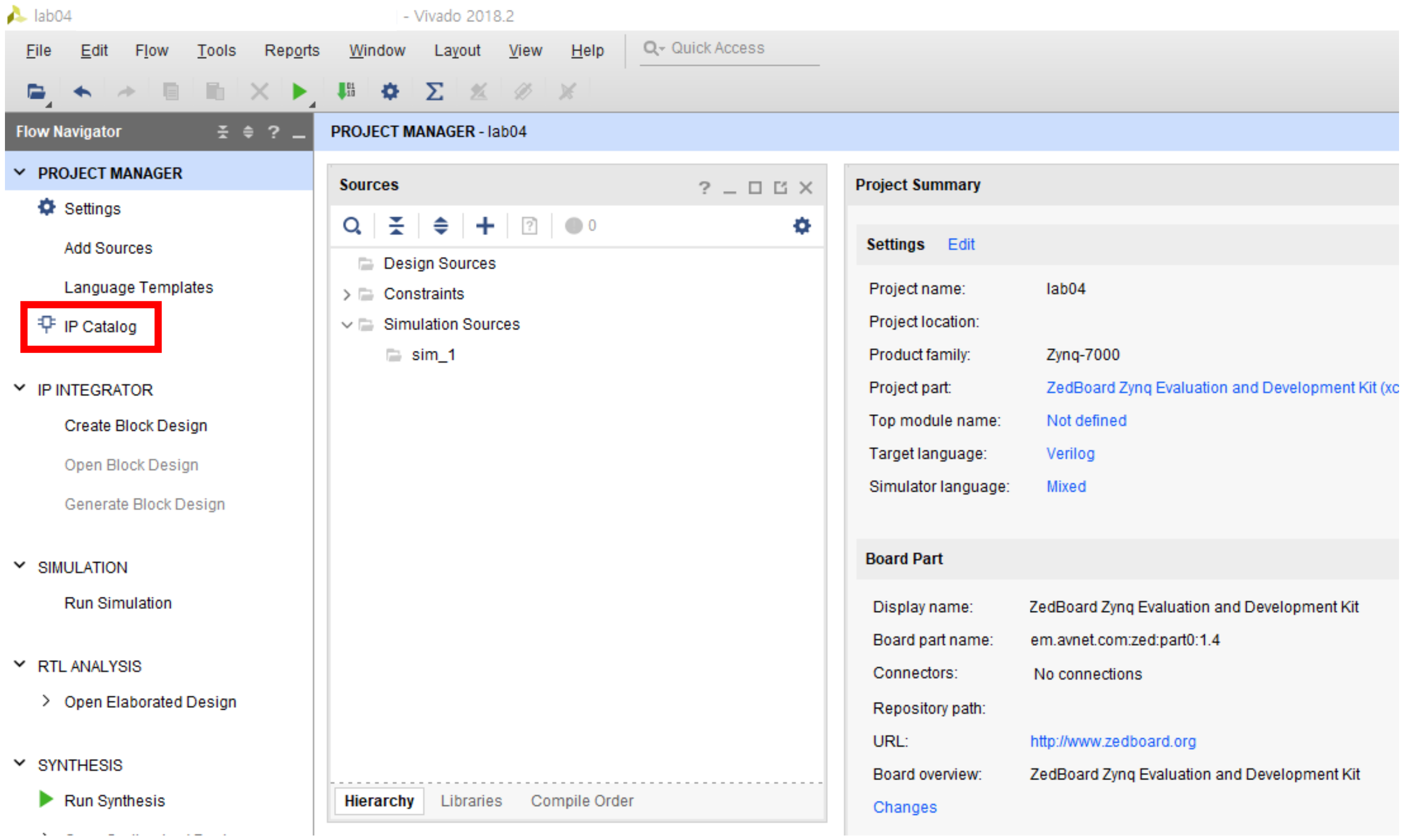
④ Implementation
- Place
- Route



How to use IP catalog

Generating floating point Multiply-Adder

- Using IP Catalog



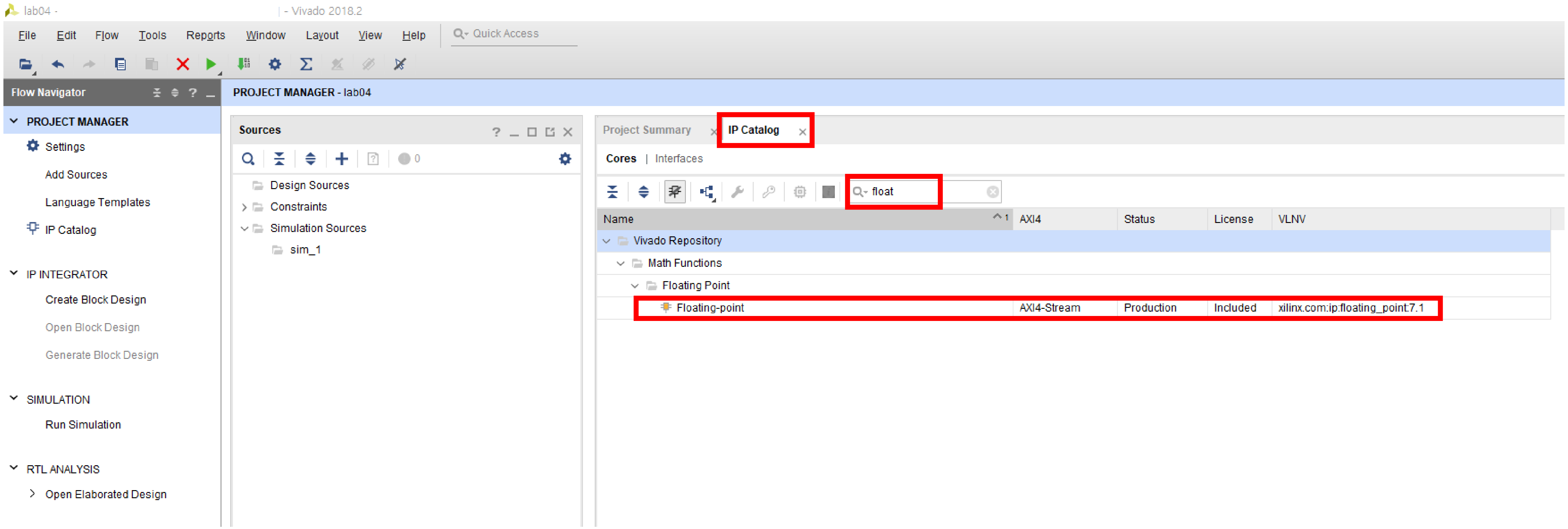
Project name:	lab04
Project location:	
Product family:	Zynq-7000
Project part:	ZedBoard Zynq Evaluation and Development Kit (xc
Top module name:	Not defined
Target language:	Verilog
Simulator language:	Mixed

Display name:	ZedBoard Zynq Evaluation and Development Kit
Board part name:	em.avnet.com:zed:part0:1.4
Connectors:	No connections
Repository path:	
URL:	http://www.zedboard.org
Board overview:	ZedBoard Zynq Evaluation and Development Kit

Changes

Generating floating point Multiply-Adder

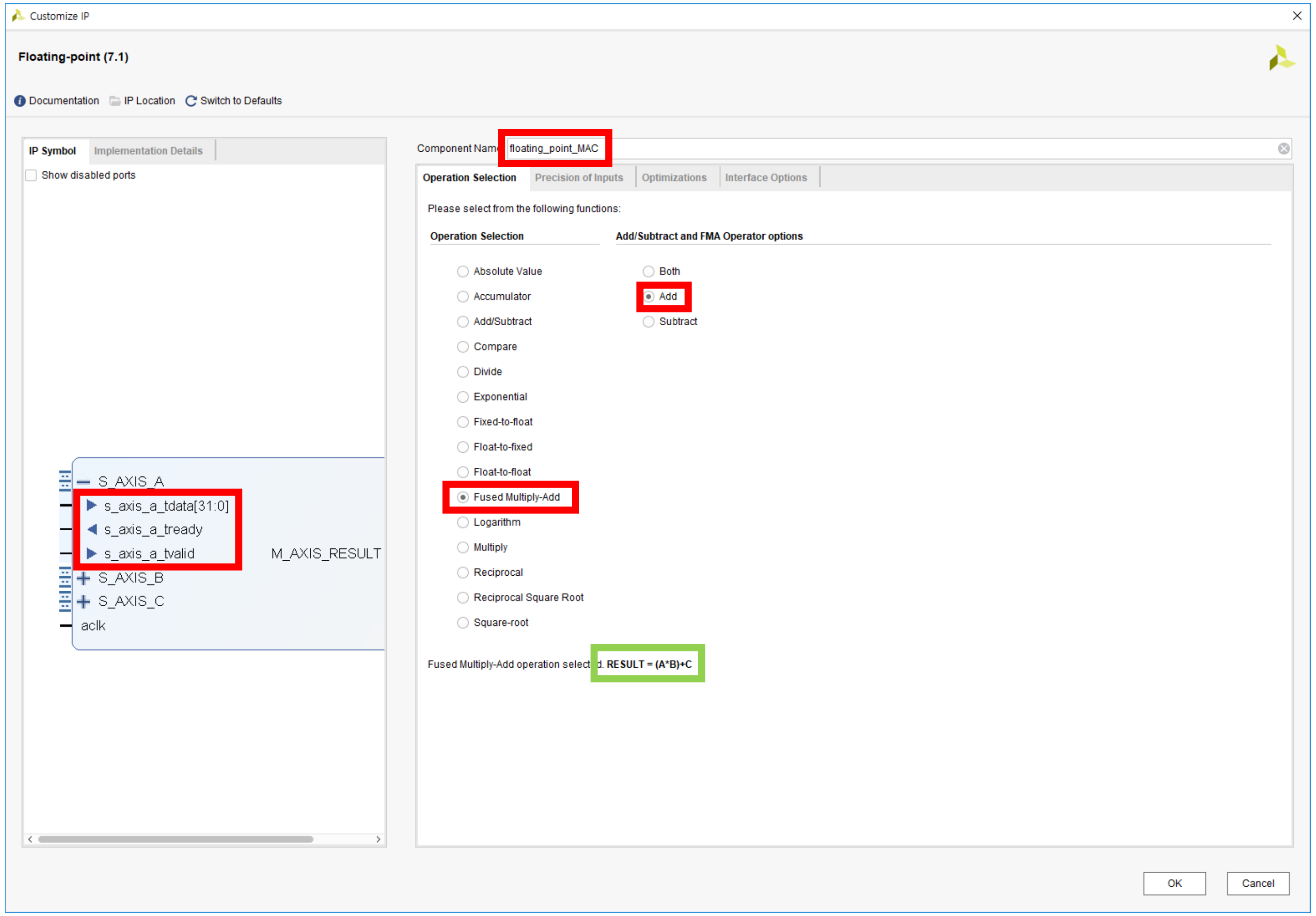
- Search IP
  - Floating-point (for fp) / Multiply adder (for int)



Name	AXI4	Status	License	VLNV
Vivado Repository				
Math Functions				
Floating Point				
Floating-point	AXI4-Stream	Production	Included	xilinx.com:ip:floating_point:7.1

Generating floating point Multiply-Adder

- Configuration



Customize IP

Floating-point (7.1)

Documentation IP Location Switch to Defaults

IP Symbol | Implementation Details

Show disabled ports

Component Name floating_point_MAC

Operation Selection | Precision of Inputs | Optimizations | Interface Options

Please select from the following functions:

Operation Selection

- Absolute Value
- Accumulator
- Add/Subtract
- Compare
- Divide
- Exponential
- Fixed-to-float
- Float-to-fixed
- Float-to-float
- [✓ Fused Multiply-Add]
- Logarithm
- Multiply
- Reciprocal
- Reciprocal Square Root
- Square-root

Add/Subtract and FMA Operator options

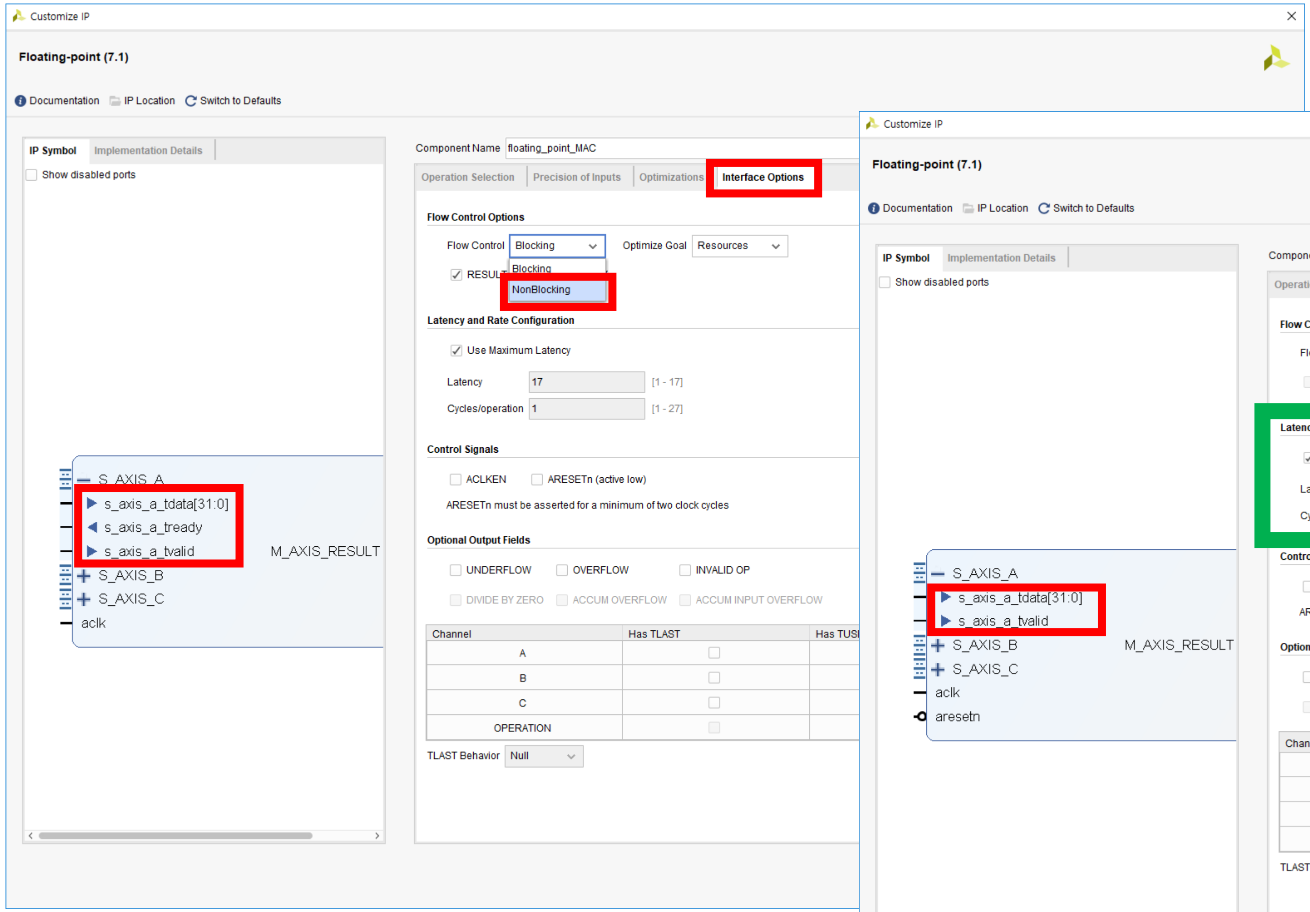
- Both
- [✓ Add]
- Subtract

Fused Multiply-Add operation selected. RESULT = (A*B)+C

OK Cancel

Generating floating point Multiply-Adder

- Configuration



Customize IP

Floating-point (7.1)

Documentation | IP Location | Switch to Defaults

IP Symbol | Implementation Details

Show disabled ports

S_AXIS_A
s_axis_a_tdata[31:0]
s_axis_a_tready
s_axis_a_tvalid
S_AXIS_B
S_AXIS_C
aclk
M_AXIS_RESULT

Component Name: floating_point_MAC

Operation Selection | Precision of Inputs | Optimizations | Interface Options

Flow Control Options

Flow Control: Blocking (Blocking / NonBlocking) — Optimize Goal: Resources

✓ RESULT

Latency and Rate Configuration

✓ Use Maximum Latency

Latency: 17 [1 - 17]

Cycles/operation: 1 [1 - 27]

Control Signals

ACLKEN — ARESETn (active low)

ARESETn must be asserted for a minimum of two clock cycles

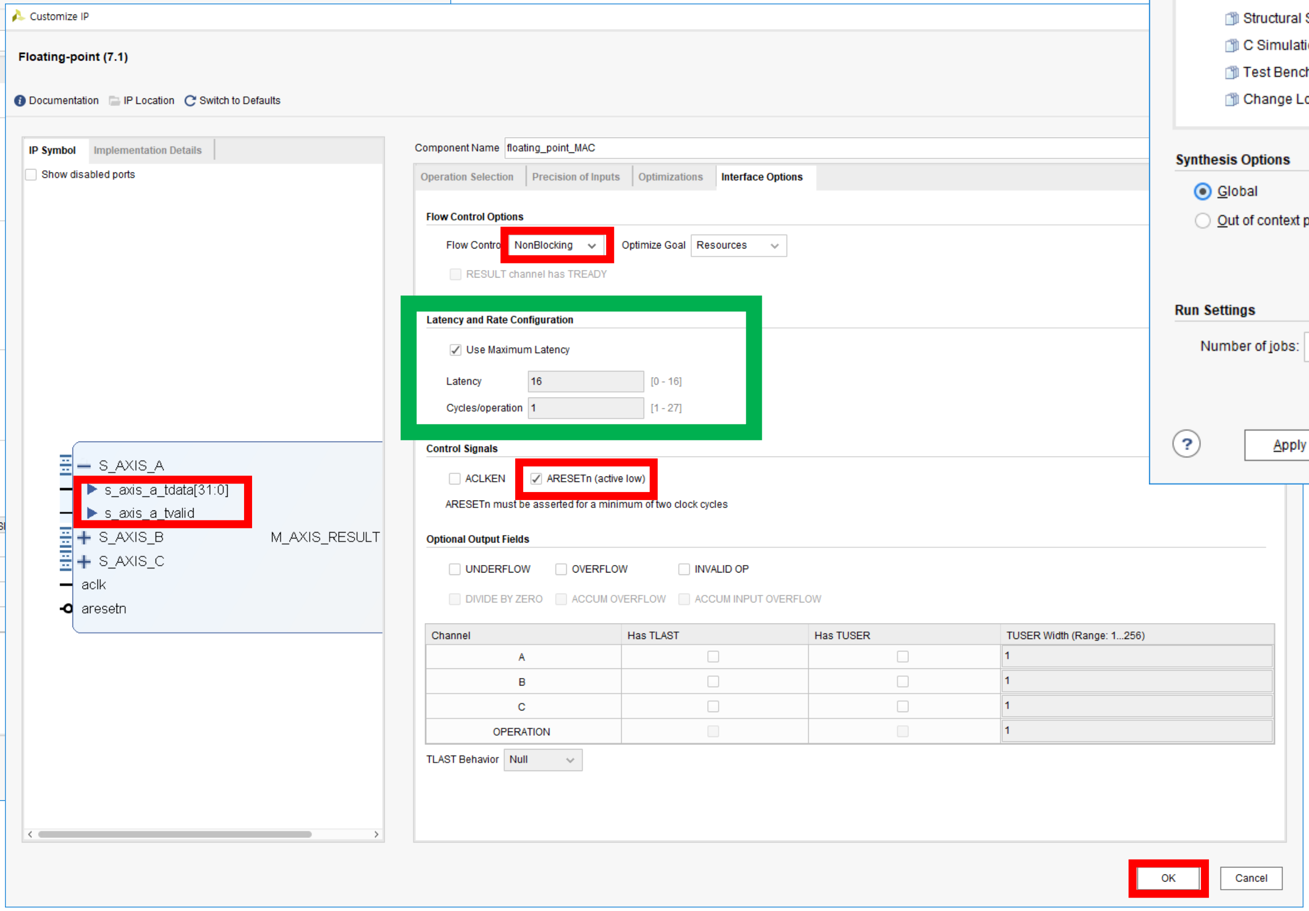
Optional Output Fields

UNDERFLOW — OVERFLOW — INVALID OP

DIVIDE BY ZERO — ACCUM OVERFLOW — ACCUM INPUT OVERFLOW

Channel	Has TLAST	Has TUSER
A	☐	
B	☐	
C	☐	
OPERATION	☐	

TLAST Behavior: Null



Customize IP

Floating-point (7.1)

Documentation | IP Location | Switch to Defaults

IP Symbol | Implementation Details

Show disabled ports

S_AXIS_A
s_axis_a_tdata[31:0]
s_axis_a_tvalid
S_AXIS_B
S_AXIS_C
aclk
aresetn
M_AXIS_RESULT

Component Name: floating_point_MAC

Operation Selection | Precision of Inputs | Optimizations | Interface Options

Flow Control Options

Flow Control: NonBlocking — Optimize Goal: Resources

RESULT channel has TREADY

Latency and Rate Configuration

✓ Use Maximum Latency

Latency: 16 [0 - 16]

Cycles/operation: 1 [1 - 27]

Control Signals

ACLKEN — ✓ ARESETn (active low)

ARESETn must be asserted for a minimum of two clock cycles

Optional Output Fields

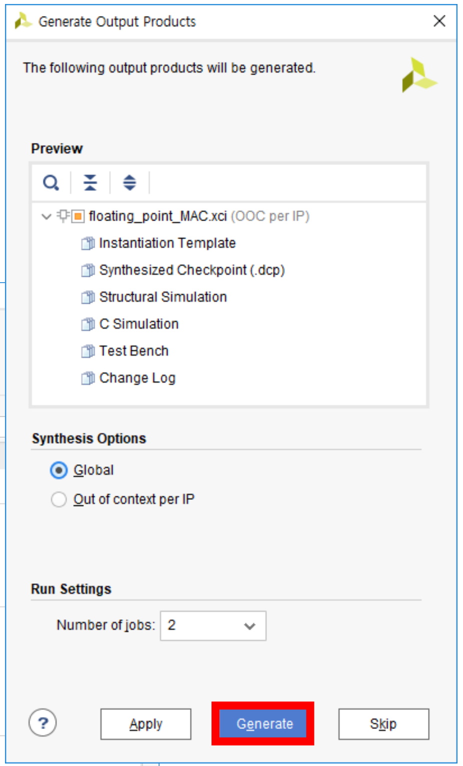
UNDERFLOW — OVERFLOW — INVALID OP

DIVIDE BY ZERO — ACCUM OVERFLOW — ACCUM INPUT OVERFLOW

Channel	Has TLAST	Has TUSER	TUSER Width (Range: 1...256)
A	☐	☐	1
B	☐	☐	1
C	☐	☐	1
OPERATION	☐	☐	1

TLAST Behavior: Null

OK — Cancel



Generate Output Products

The following output products will be generated.

Preview

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

Synthesis Options

[✓ Global] Out of context per IP

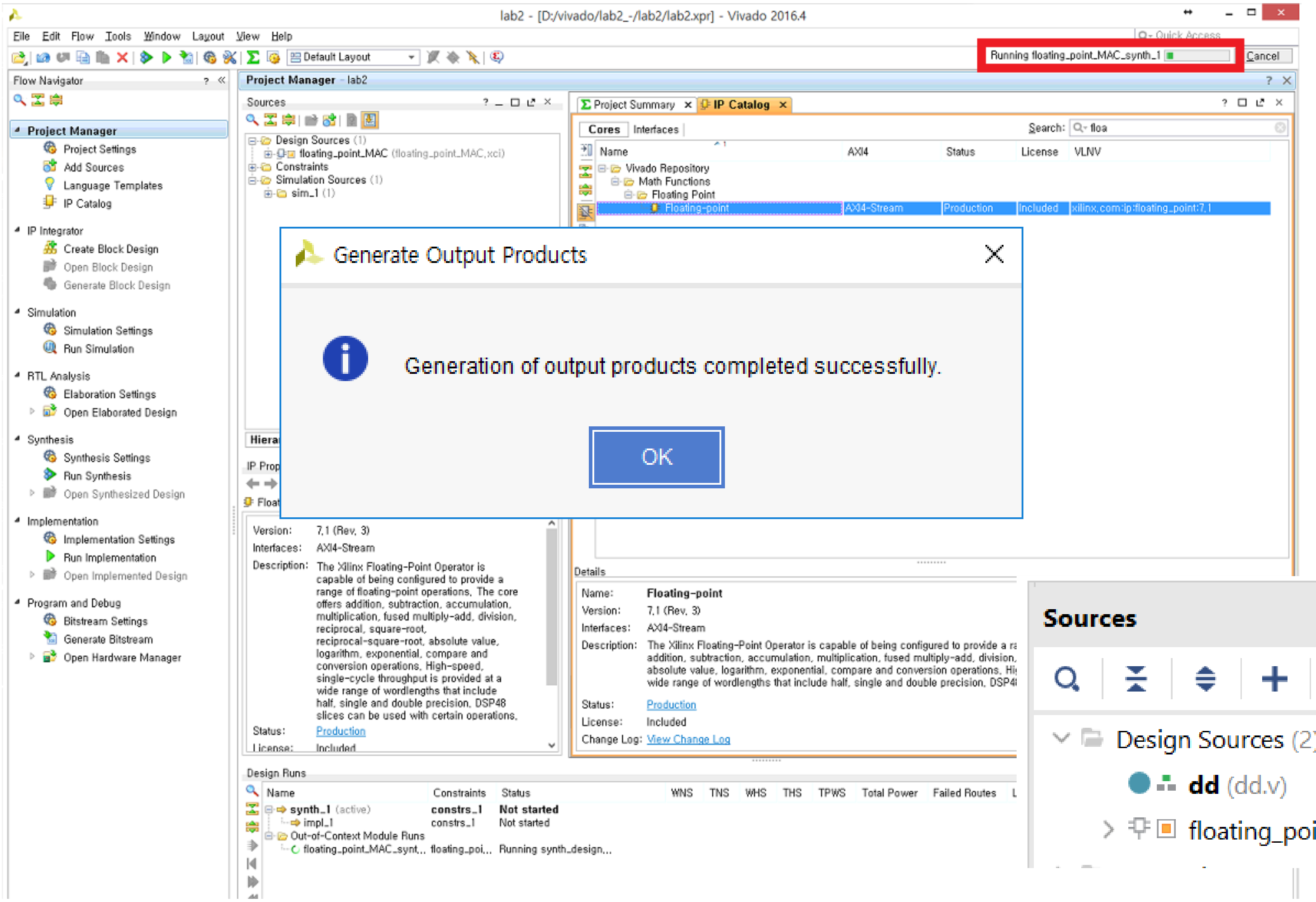
Run Settings

Number of jobs: 2

Apply — Generate — Skip

Generating floating point Multiply-Adder

- Generate customized floating point Multiply-Adder

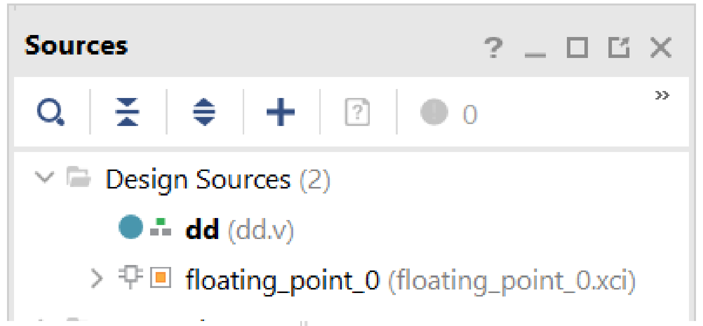


Generation of output products completed successfully.

Version: 7.1 (Rev. 3)
Interfaces: AXI4-Stream
Description: The Xilinx Floating-Point Operator is capable of being configured to provide a range of floating-point operations. The core offers addition, subtraction, accumulation, multiplication, fused multiply-add, division, reciprocal, square-root, reciprocal-square-root, absolute value, logarithm, exponential, compare and conversion operations. High-speed, single-cycle throughput is provided at a wide range of wordlengths that include half, single and double precision. DSP48 slices can be used with certain operations.
Status: Production
License: Included

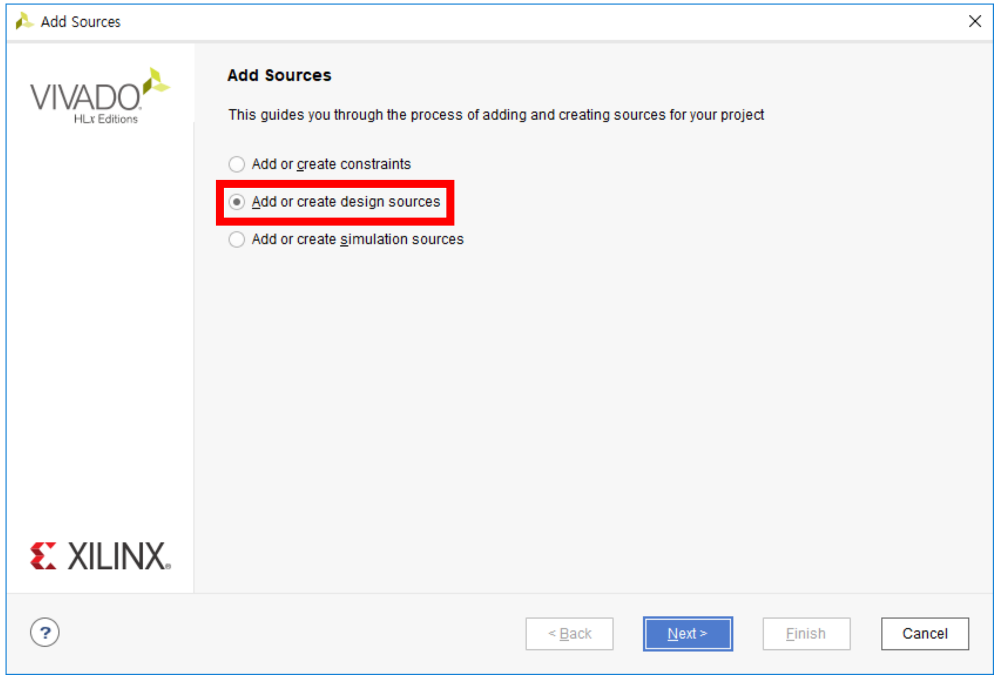
Name: Floating-point
Version: 7.1 (Rev. 3)
Interfaces: AXI4-Stream
Status: Production
License: Included
Change Log: View Change Log

Name	Constraints	Status	WNS	TNS	WHS	THS	TPWS	Total Power	Failed Routes
synth_1 (active)	constrs_1	Not started							
impl_1	constrs_1	Not started							
Out-of-Context Module Runs									
floating_point_MAC_synt...	floating_poi...	Running synth_design...							



Generating floating point Multiply-Adder

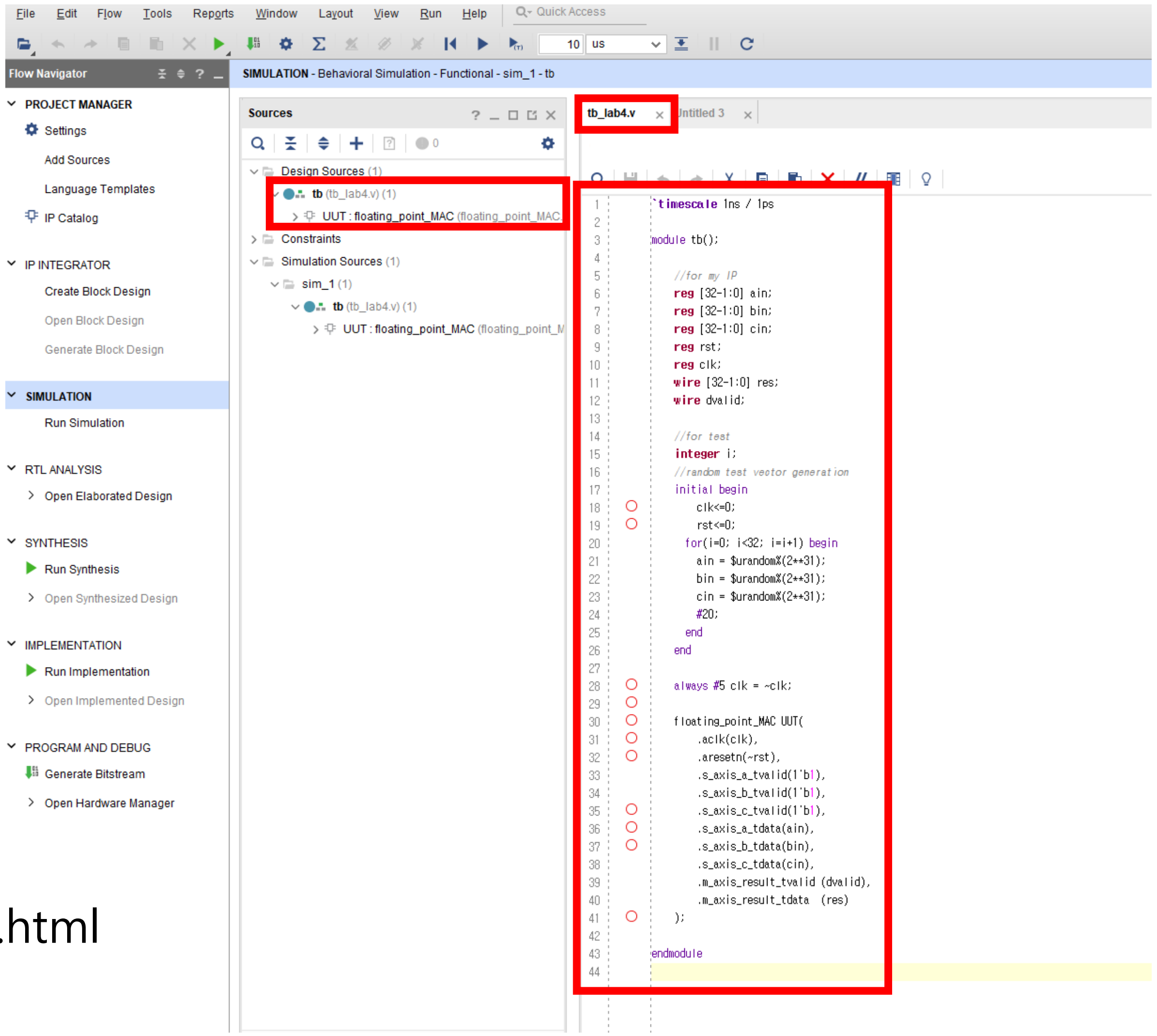
- Write source for top module



Add Sources

This guides you through the process of adding and creating sources for your project

( ) Add or create constraints
[✓ Add or create design sources]
( ) Add or create simulation sources



SIMULATION - Behavioral Simulation - Functional - sim_1 - tb

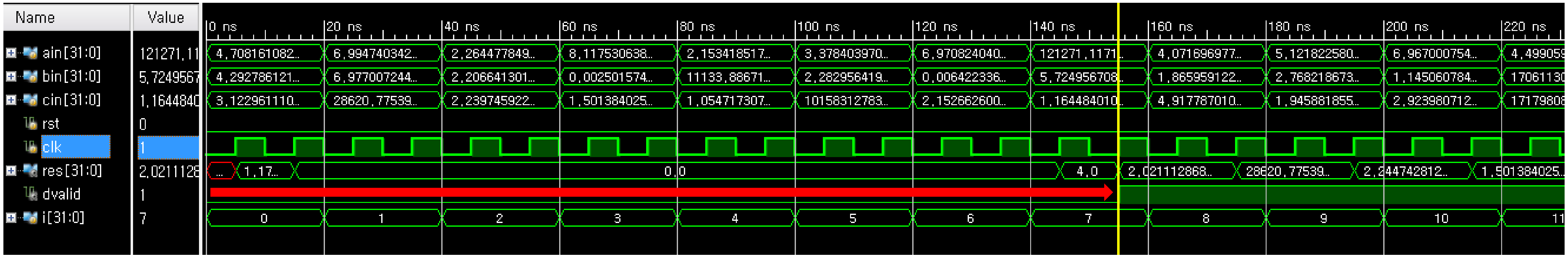
```
Design Sources (1)
  tb (tb_lab4.v) (1)
    UUT : floating_point_MAC (floating_point_MAC
Constraints
Simulation Sources (1)
  sim_1 (1)
    tb (tb_lab4.v) (1)
      UUT : floating_point_MAC (floating_point_M
```

```verilog
`timescale 1ns / 1ps

module tb();

    //for my IP
    reg [32-1:0] ain;
    reg [32-1:0] bin;
    reg [32-1:0] cin;
    reg rst;
    reg clk;
    wire [32-1:0] res;
    wire dvalid;

    //for test
    integer i;
    //random test vector generation
    initial begin
        clk<=0;
        rst<=0;
      for(i=0; i<32; i=i+1) begin
        ain = $urandom%(2**31);
        bin = $urandom%(2**31);
        cin = $urandom%(2**31);
        #20;
      end
    end

    always #5 clk = ~clk;

    floating_point_MAC UUT(
        .aclk(clk),
        .aresetn(~rst),
        .s_axis_a_tvalid(1'b1),
        .s_axis_b_tvalid(1'b1),
        .s_axis_c_tvalid(1'b1),
        .s_axis_a_tdata(ain),
        .s_axis_b_tdata(bin),
        .s_axis_c_tdata(cin),
        .m_axis_result_tvalid (dvalid),
        .m_axis_result_tdata  (res)
    );

endmodule
```

https://www.h-schmidt.net/FloatConverter/IEEE754.html

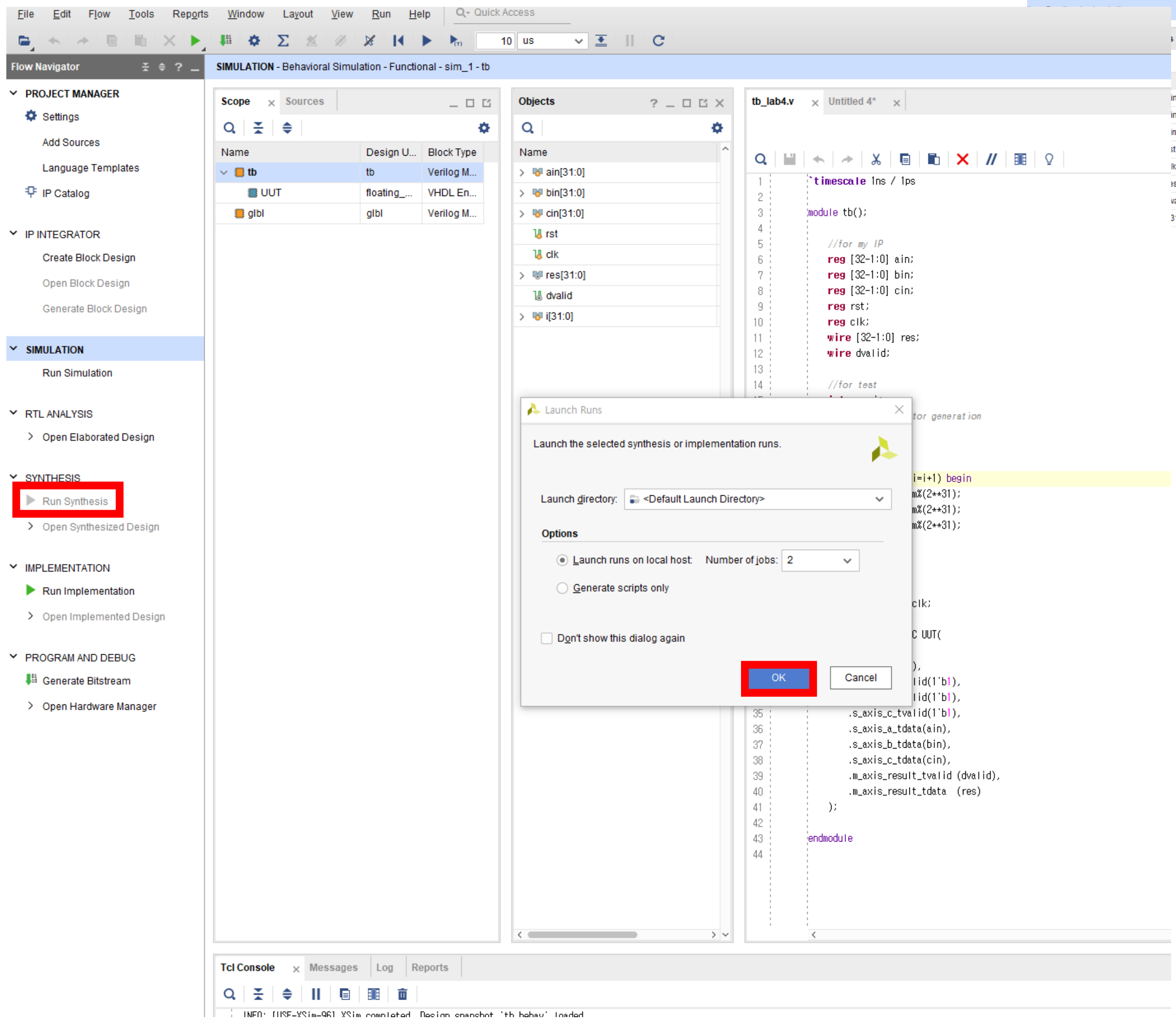
Generating floating point Multiply-Adder

- Result check with simulation



Generating floating point Multiply-Adder

- Run Synthesis



SIMULATION - Behavioral Simulation - Functional - sim_1 - tb

Name	Design U...	Block Type
tb	tb	Verilog M...
UUT	floating_...	VHDL En...
glbl	glbl	Verilog M...

Launch Runs

Launch the selected synthesis or implementation runs.

Launch directory: <Default Launch Directory>

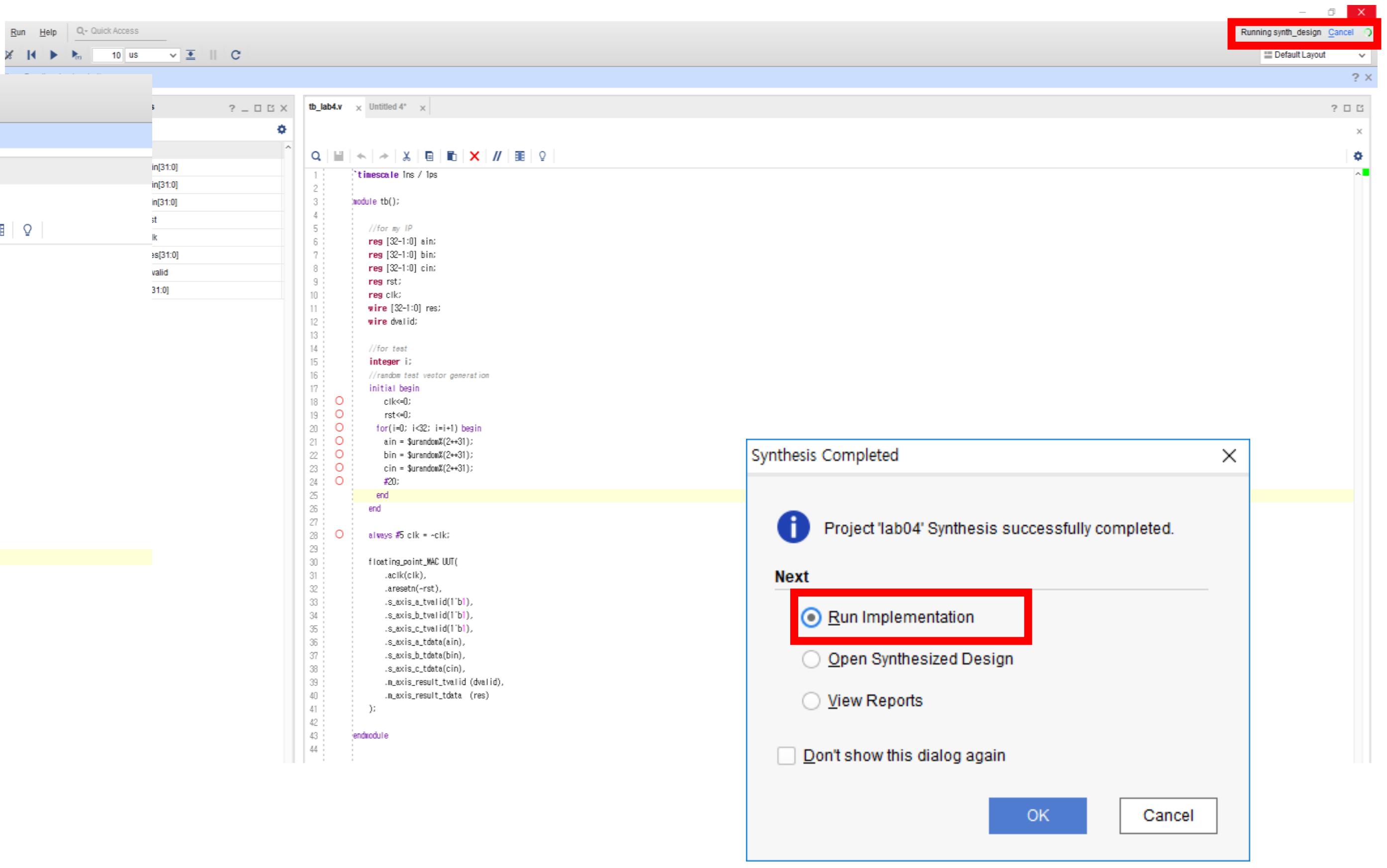
Options

[✓ Launch runs on local host] Number of jobs: 2

Generate scripts only

Don't show this dialog again

```
INFO: [USF-XSim-96] XSim completed. Design snapshot 'tb_behav' loaded.
```



```
`timescale 1ns / 1ps

module tb();

    //for my IP
    reg [32-1:0] ain;
    reg [32-1:0] bin;
    reg [32-1:0] cin;
    reg rst;
    reg clk;
    wire [32-1:0] res;
    wire dvalid;

    //for test
    integer i;
    //random test vector generation
    initial begin
        clk<=0;
        rst<=0;
      for(i=0; i<32; i=i+1) begin
        ain = $urandom%(2**31);
        bin = $urandom%(2**31);
        cin = $urandom%(2**31);
        #20;
      end
    end

    always #5 clk = ~clk;

    floating_point_MAC UUT(
        .aclk(clk),
        .aresetn(~rst),
        .s_axis_a_tvalid(1'b1),
        .s_axis_b_tvalid(1'b1),
        .s_axis_c_tvalid(1'b1),
        .s_axis_a_tdata(ain),
        .s_axis_b_tdata(bin),
        .s_axis_c_tdata(cin),
        .m_axis_result_tvalid (dvalid),
        .m_axis_result_tdata  (res)
    );

endmodule
```

Running synth_design Cancel

Synthesis Completed

Project 'lab04' Synthesis successfully completed.

Next

[✓ Run Implementation]

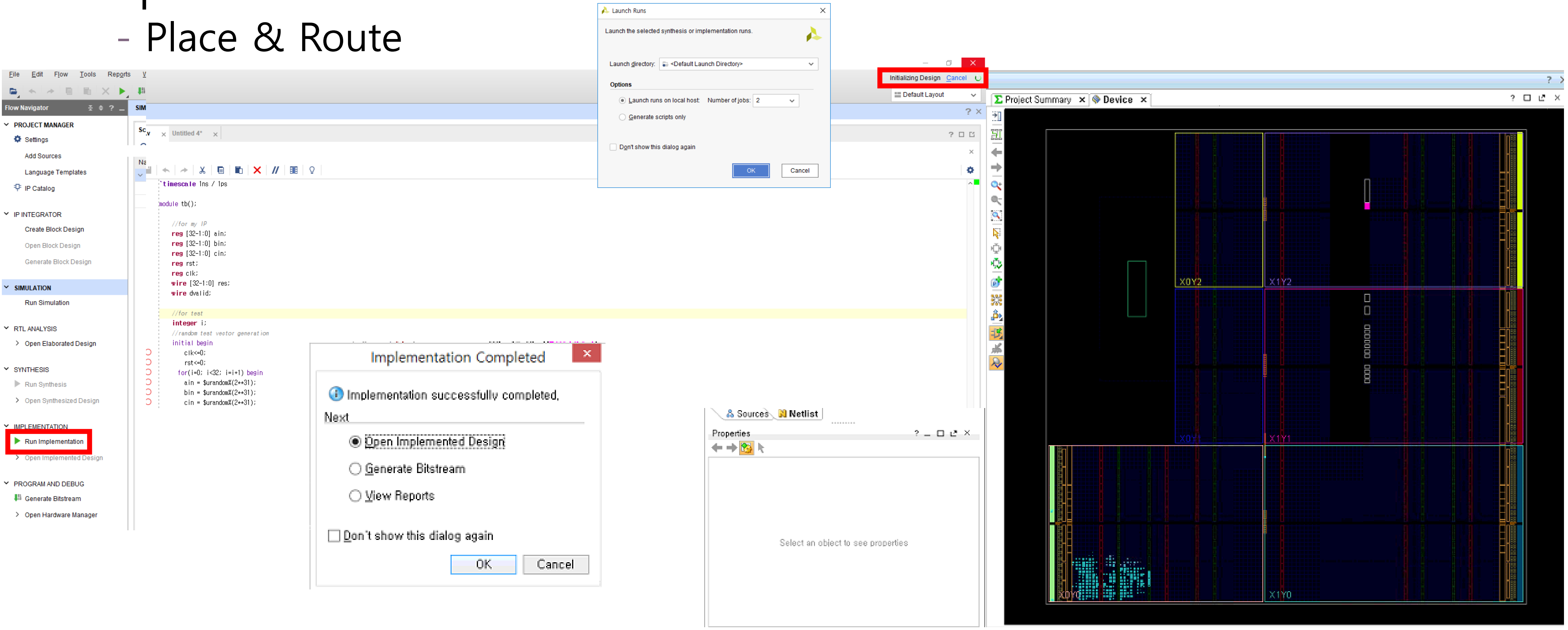
Open Synthesized Design

View Reports

Don't show this dialog again

Generating floating point Multiply-Adder

- Implementation
  - Place & Route



```
`timescale 1ns / 1ps

module tb();

    //for my IP
    reg [32-1:0] ain;
    reg [32-1:0] bin;
    reg [32-1:0] cin;
    reg rst;
    reg clk;
    wire [32-1:0] res;
    wire dvalid;

    //for test
    integer i;
    //random test vector generation
    initial begin
        clk<=0;
        rst<=0;
      for(i=0; i<32; i=i+1) begin
        ain = $urandom%(2**31);
        bin = $urandom%(2**31);
        cin = $urandom%(2**31);
```

Launch Runs

Launch the selected synthesis or implementation runs.

Launch directory: <Default Launch Directory>

Options

[✓ Launch runs on local host:] Number of jobs: 2

Generate scripts only

Don't show this dialog again

Initializing Design Cancel

Implementation Completed

Implementation successfully completed.

Next

[✓ Open Implemented Design]

Generate Bitstream

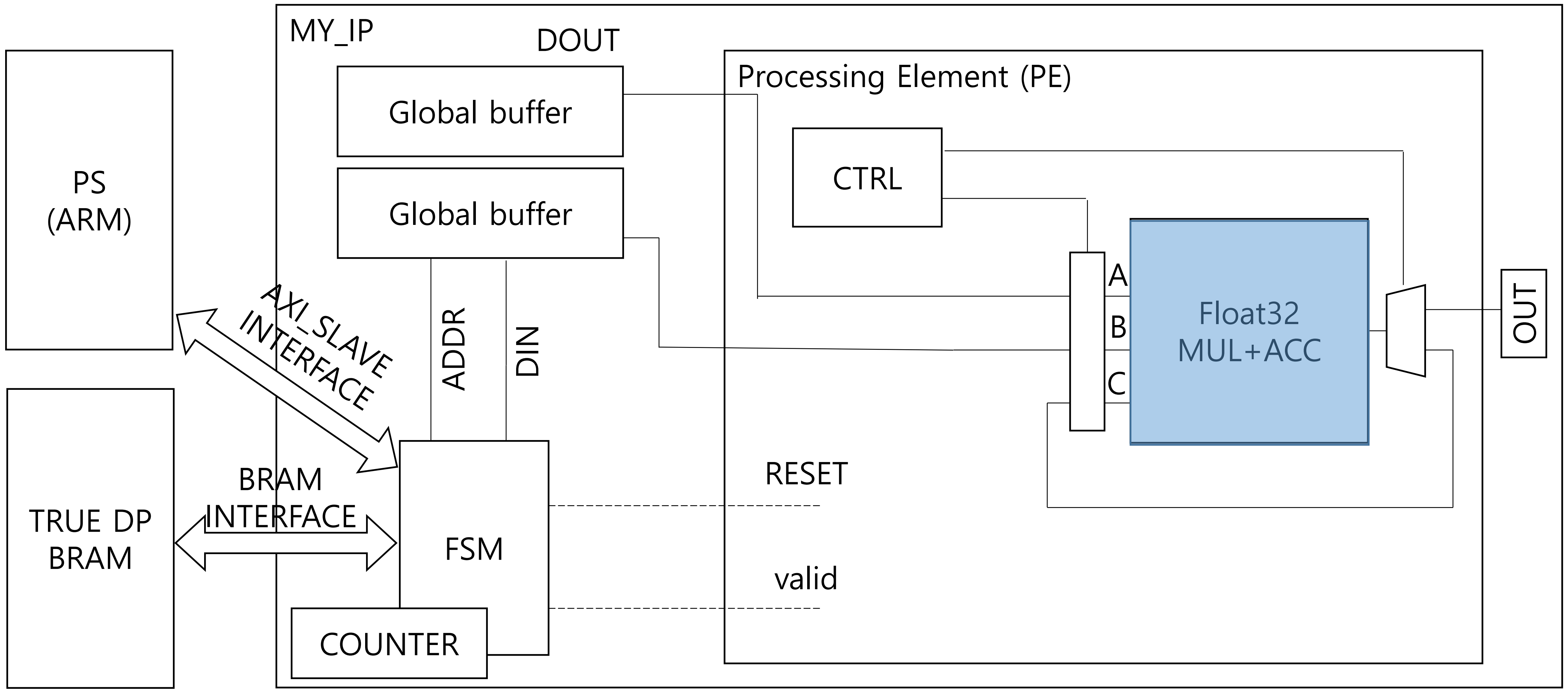
View Reports

Don't show this dialog again

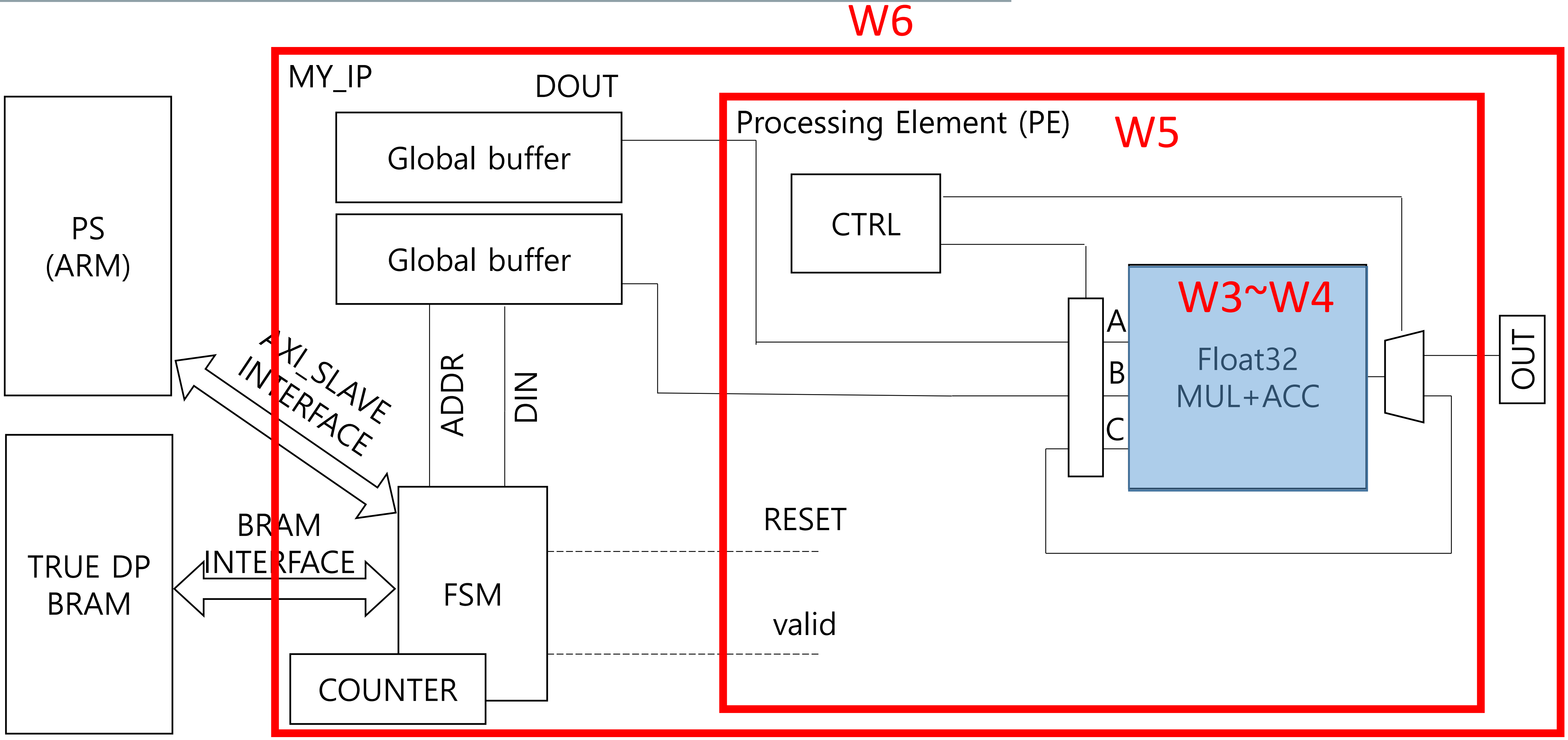
Select an object to see properties

Main Practice

Final Project Overview: Matrix Multiplication IP



Final Project Overview: Matrix Multiplication IP



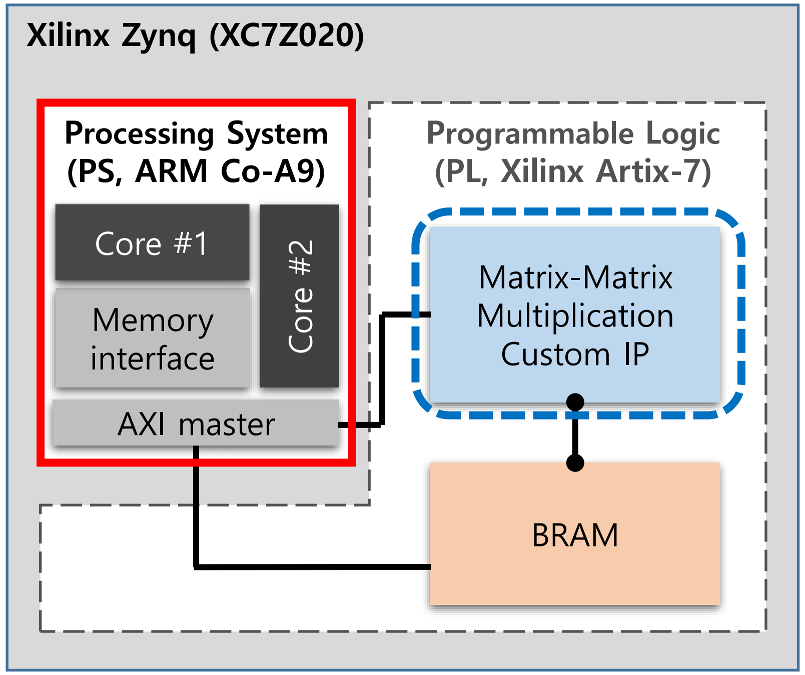
Final Project Overview: Matrix Multiplication IP

- CNN is our application
  - Convolution layer becomes a matrix-matrix multiplication after convolution lowering e.g., 32x64 matrix * 64x64 matrix ➔ 32x64 matrix
- ARM CPU runs the main function which calls your MM IP on PL
  - MM for 32x64 weight matrix * 64x64 input matrix multiplication
- BRAM is used for data transfer between SW and HW

MM function on Hardware (Software running on CPU)

```
for(i=0; i<32; i+=1) {
  for(j=0; j<64; j+=1) {
    for(k = 0; k < 64; k++){
      Output[i][j] += Input[i][k]*W[k][j]
    }
  }
}
```

Fused multiply



Xilinx Zynq (XC7Z020)
Processing System (PS, ARM Co-A9)
Core #1
Core #2
Memory interface
AXI master
Programmable Logic (PL, Xilinx Artix-7)
Matrix-Matrix Multiplication Custom IP
BRAM

Practice

1. **[IP catalog] Implement following two blocks and synthesize them by using IP catalog of Vivado.**
   ① Design 32 bit floating point Multiply-Adder (Tutorial)
      - Follow tutorial.
      - Design your own test bench and show the wave form. (test at least 32 vectors)
      - Simulation, Synthesis

   ② Design 32 bit integer Multiply-Adder
      - Design your own custom ip in ip catalog. (bit-width, delay, …)
      - Design your own test bench and show the wave form. (test at least 32 vectors)
      - Simulation, Synthesis

2. **[Verilog] Implement adder array consists of 4 adders.**
   - Use for-generate statement.
   - Use adder module (32 bit based) that you implemented last week.
   - Design your own test bench and show the wave form (test 4 random vectors for each 'cmd').
   - Simulation, Synthesis

Adder array

```
module adder_array (cmd, ain0, ain1, ain2, ai
n3, bin0, bin1, bin2, bin3, dout0, dout1, dou
t2, 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;
   . . .
  assign {ain[0], ain[1], ain[2], ain[3]} =
{ain0, ain1, ain2, ain3};
   . . .
endmodule
```

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

Homework

- Requirements
  - Result
    - Attach your project folder with all your Verilog codes (e.g., adder-array, test-bench, 32bit integer Multiply-Adder, 32bit floating point Multiply-Adder)
    - Attach your 32bit integer Multiply-Adder, 32bit floating point Multiply-Adder waveform(simulation result) with [student_number, name]
      - Test at least 32 vectors
      - Refer to Practice3 about Screenshot
    - Attach your Adder-array waveform(simulation result) with [student_number, name]
      - Test 4 random vectors for each "cmd"
      - Refer to Practice3 about Screenshot
  - Report
    - Explain 32bit integer Multiply-Adder, 32bit floating point Multiply-Adder that you implemented
    - Explain Adder-array 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 : [Lab04]name1_name2.zip (ex : [Lab04]홍길동_홍동길.zip **-> All Team members' name should be included**)
  - Due: 3/31(Wed) 23:59
    - **No Late Submission**