**San Jose State University**

**Department of Computer Engineering**

**CMPE 125 Spring 2017**

**Lab 5 Report**

**Sequential Building Blocks**

**Date \_\_\_3/25/17\_\_\_\_\_\_\_**

**By**

**Name \_\_\_Anahit Sarao\_\_\_\_\_\_\_ SID \_\_\_008435583\_\_\_**

**Name \_\_\_\_Maxwell Cheshier\_\_\_\_\_\_ SID \_\_009193717\_\_\_**

**Lab Record**

| **Tasks** | **Designed by (print name)** | **Verified by (print name)** | **\*Completion Status** |
| --- | --- | --- | --- |
| **1** | Maxwell | Ryan | A |
| **2** | Anahit | Ryan | B |

| **Task** | **Performed by (print name)** | **Validated by (print name)** | **\*Completion Status** |
| --- | --- | --- | --- |
| **3** | Both | Ryan | x |

**\* Enter the following:**

**A – if the task was successfully completed**

**B – if the task was partially completed**

**X – if the task was failed or not performed**

**If you entered B or X, detailed description about the incompletion or failure must be given in the report.**

***Introduction***

The lab consists of three parts: designing a parameterized D register, plus the sourcing of older lab material such as creating single AND units and preparing lab fours Carry Look Ahead Adder (CLA) design to be compatible for this lab. The second part utilized the D register and CLA to design a 4-bit combinational unsigned integer multiplier. The design methodology was to follow the parallel architecture, with a self testing testbench. An inferred multiplier was used to model the building blocks of the combinational multiplier. The last task involved designing a two-stage pipelined integer multiplier which would utilize the D registers. Task three required hardware and stimuli verification. This lab involved complex design methodologies, the complexity was handled by breaking down modules to smaller designs and building up to a top-level module.

There were many issues with certain parts of the lab during the stimuli of the combinational integer multiplier the output was 8 bits wide. For a majority of the lab only bits 0-3 and bit 7 only displayed readable values. This was solved after by utilizing the RTL inspection features within Xilinx. The pipelined multiplier was not fully verified however fixes and modifications of design and verification code yielded better results.

The parts listed below were not fully completed in lab but were finished afterword:

* Verification stimuli of combinational multiplier
* Design of pipelined integer multiplier
* Stimuli and Hardware Verification of pipelined integer multiplier

***Design Methodology***

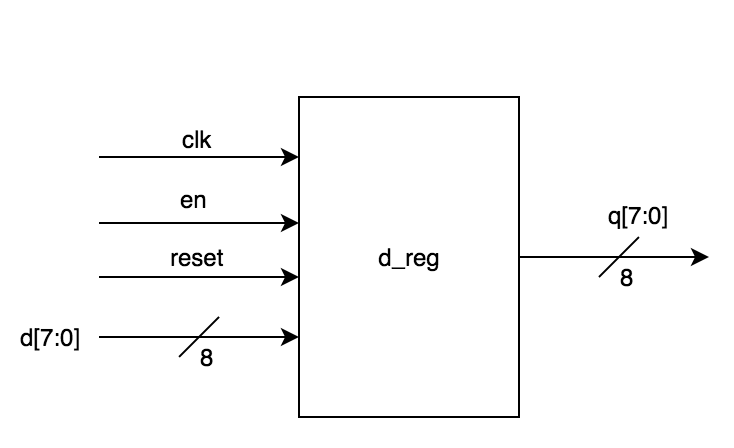
These three designs are the building blocks for the bigger designs discussed later in this report. The D register was to be a simple D Flip Flop shown in figure one, which would take in an input and be able to output during the rising edge of the clock. The registers are used for the pipelined multiplier which streamlines the inputs at every stage, during each clock cycle the stages are evenly used leading to a performance increase. 

Figure 1. Parameterized D register Block Diagram

The AND module and 8-bit CLA are from lab four which are basic fundamental modules to create more complex modules. The 8-bit CLA consists of two 4-bit CLA adders in which multiple AND modules are used. Figure two and three show the block design diagrams for these imported modules. Minor changes were made to the code design for better readability and modularity.

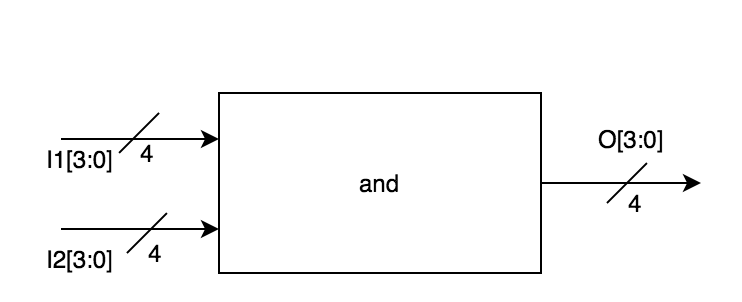


Figure 2. AND Module Block Diagram

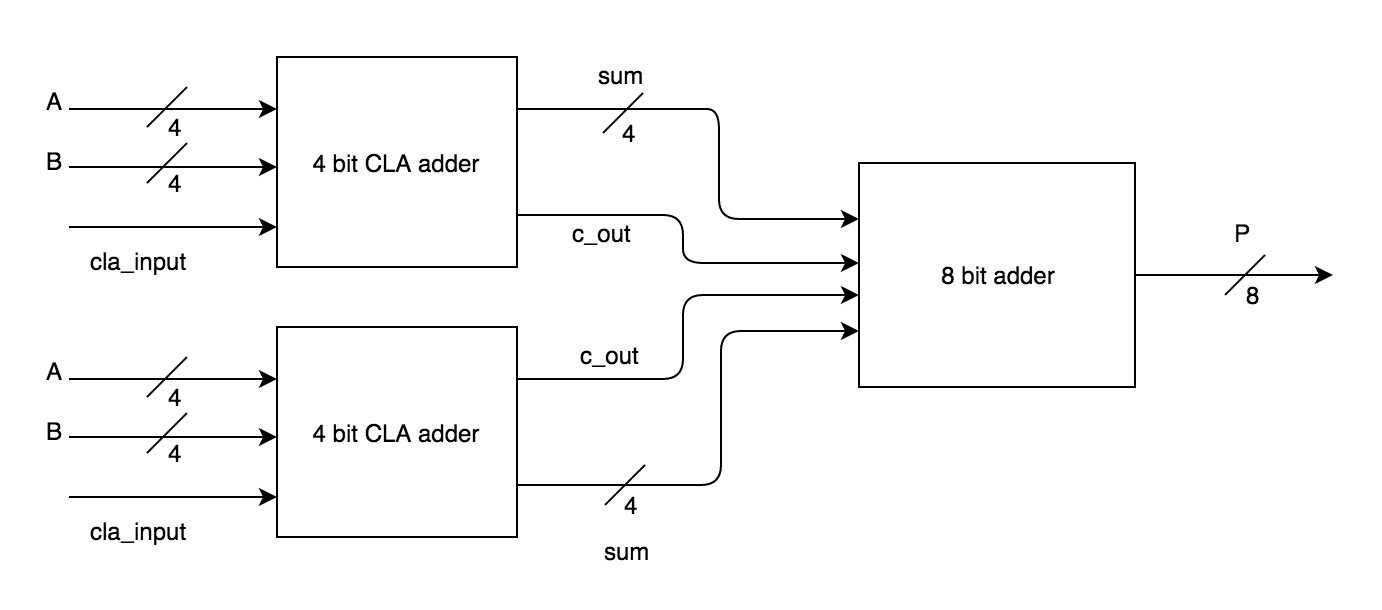


Figure 3. 8-bit CLA Block Diagram

*Combinational Multiplier:* The combinational multiplier uses multiple modules to construct a viable design. By using two 4-bit CLA adders to create a functional 8-bit CLA adder which will output a number with a max width of 8-bits. The multiplier follows a parallel as structure as we two sets of bits to which are shifted then added. This combinational design results in an 8-bit shifter. For verification a test bench was designed to achieve stimuli verification. Figure four shows the block level design off the parallel integer multiplier.

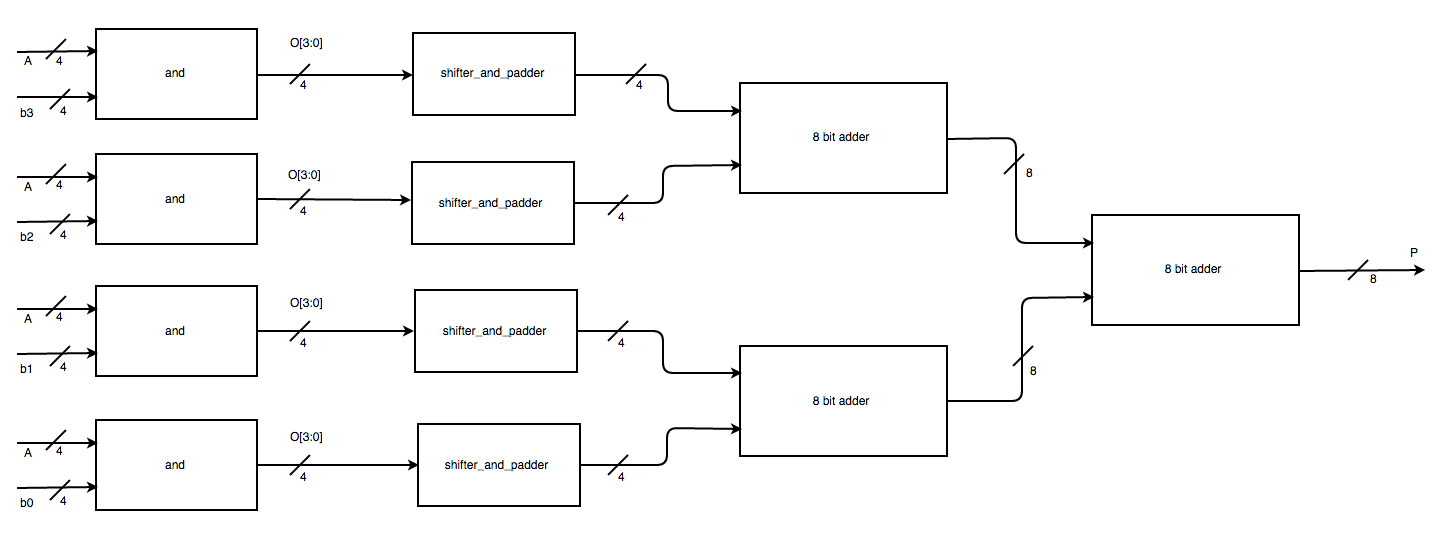


Figure 4. Combination Unsigned Multiplier Block Diagram

*Pipelined Multiplier:* Following a similar overall structure of the combinational multiplier the pipelined multiplier provides the same function. However as seen in figure five, this multiplier has 4 input attached registers and 2 output registers. This design creates a two-stage pipelined multiplier. The rising clock enabled register will now shift and add at every clock cycle.

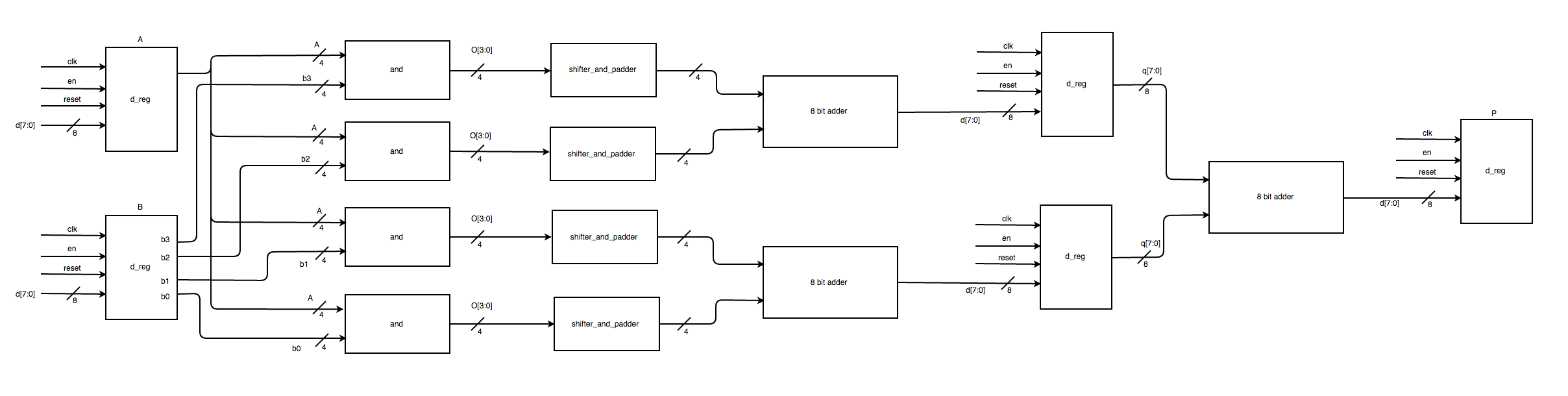
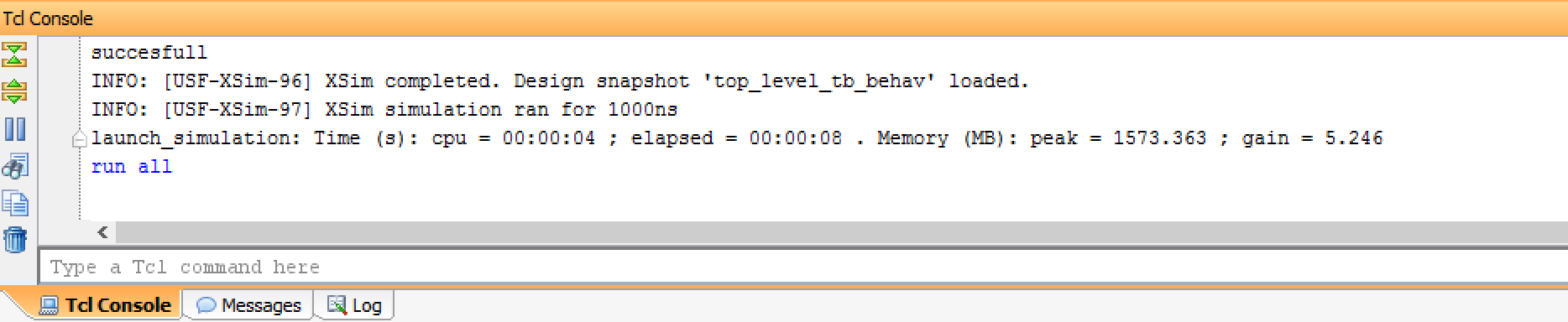


Figure 5. Pipelined Integer Multiplier

***Simulation Test Plan***

*Combinational Multiplier:* For simulation verification of the combinational multiplier the self testing test iterates through every possible input. The testbench uses two for loops to iterate from 0-15, while checking the input against the previously generated sum. Figure six shows a successful test bench. The single self checking testbench stimuli uses comparison logic and operational characteristics like if statements. The clock is set by the testbench which creates a defined state. This allows the values to be checked to the expected value. Additional waveforms can be found in the appendix.

Figure 6. Combination Unsigned Multiplier Simulation Verification



***FPGA Test Plan***

To attempt to validate the board, three new modules were added to be able to use the seven segment display. This was due to improper mapping of the ports. The XDC file was set to use the switches, RGB LED and seven segment display. However we were unable to get this part to work as the bitstream was not fully being generating.

***Conclusion***

In conclusion, our combinational multiplier performed as expected, however the pipelined multiplier contained some issues causing certain connections to have bus contentions. This was solved using the RTL synthesis. We have since rectified this and properly designed the lab reflecting the pipelined design. In addition to fixing the verification for the pipelined multiplier , we designed and successfully tested both the combinational and pipelined multiplier. We were unable to successfully test the FPGA board and could not verify the complete functionality of the board.

***Appendix***

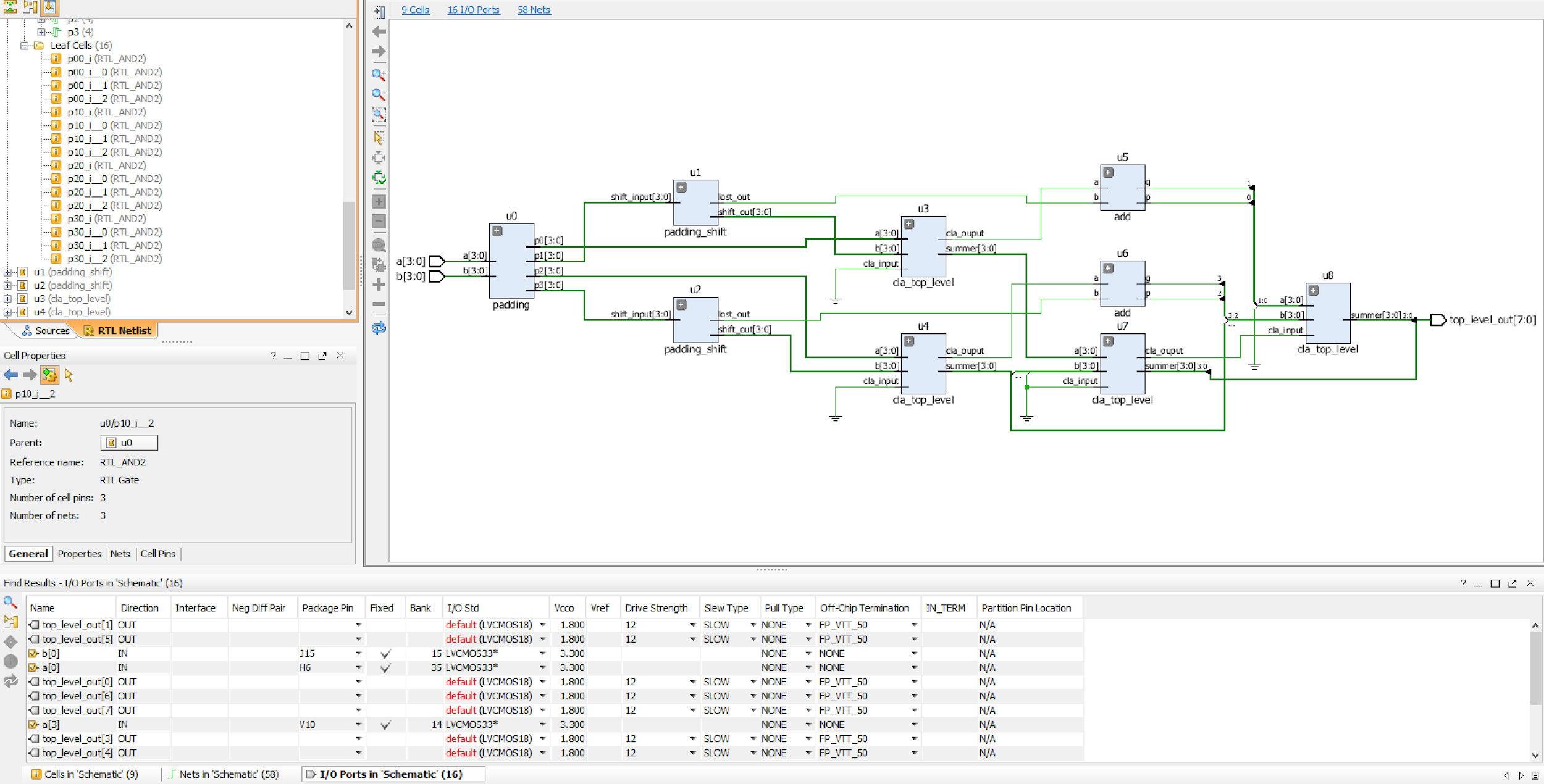


Figure 7. RTL Schematic Showing Bus Contention

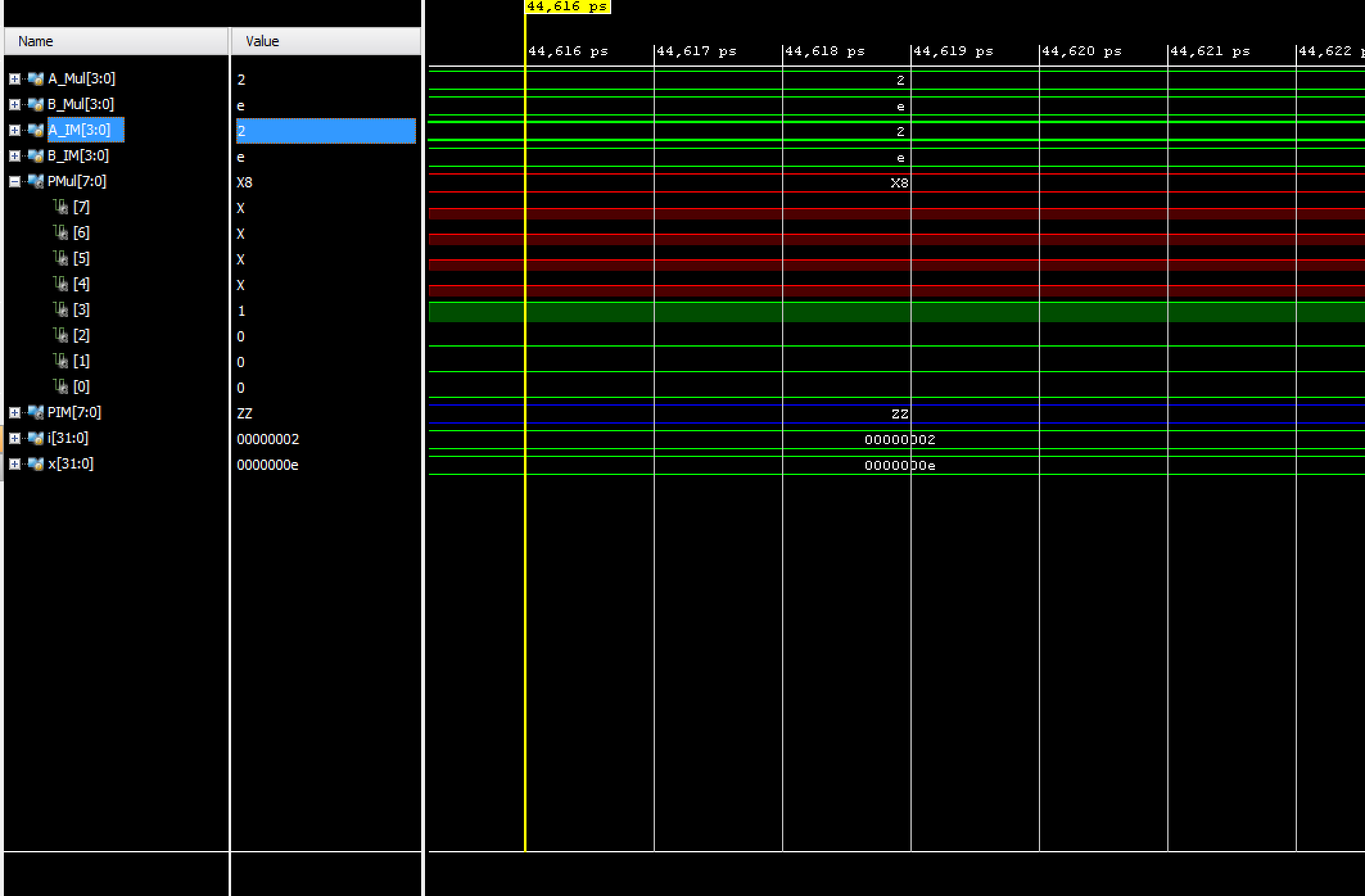


Figure 8. Bits Not Working for Parallel

***Source Code***

Top\_level.v

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 03/22/2017 04:15:14 PM

// Design Name:

// Module Name: top\_level\_tb

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module top\_level\_tb();

reg [3:0] a, b;

reg CLK\_tb;

reg RST\_tb;

reg EN\_tb;

reg [7:0] temp1, temp2, temp3;

wire [7:0] mem[2:0];

wire [7:0] sum;

wire [7:0] check0, check1, check2, check3, check4;

top\_level DUT0 (a, b, sum);

//top\_level DUT1(.a(A\_Mul), .b(B\_Mul), .top\_level\_out(PMul));

integer i,j,k, x, count;

initial

begin

a = 4'd0;

b = 4'd0;

RST\_tb = 1;

#1;

for(i= 0; i<16; i = i + 1)

begin

for(x=0; x<16; x = x + 1)

begin

A\_Mul= i;

A\_IM= i;

B\_Mul=x;

B\_IM=x;

#1;

if(PMul!=PIM)

begin

$display("Error");

$stop;

end

end

end

$display("succesfull");

$stop;

end

initial

begin

#5;

for(i = 0; i < 16; i = i + 1)

begin

a = i;

temp1 = sum;

for(j = 0; j < 16; j = j + 1)

begin

b = j;

temp3 = temp2;

temp2 = temp1;

temp1 = a \* b;

CLK\_tb = 1;

#5 CLK\_tb = 0;

#5;

if(temp3 != a \* b && CLK\_tb == 1)

$display("Error");

end

end

$display ("succesfull");

$finish;

end

endmodule

D\_reg.v

module d\_reg#(parameter WIDTH=8)

(input clk, reset, en,

input [WIDTH-1:0]d,

output reg[WIDTH-1:0]q);

always@(posedge clk, posedge reset)

if(reset) q<=0;

else if(en) q<=d;

else q<=q;

endmodule

Debouncer.v

module button\_debouncer #(parameter depth = 16) (

input wire clk, /\* 5 KHz clock \*/

input wire button, /\* Input button from constraints \*/

output reg debounced\_button);

localparam history\_max = (2\*\*depth)-1;

reg [depth-1:0] history;

always @ (posedge clk)

begin

history <= { button, history[depth-1:1] };

debounced\_button <= (history == history\_max) ? 1'b1 : 1'b0;

end

endmodule

D\_reg\_tb.v

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 03/22/2017 03:54:47 PM

// Design Name:

// Module Name: d\_reg\_tb.v

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module d\_reg\_tb();

parameter WIDTH = 8;

reg clk, reset, en;

reg [WIDTH-1:0]d\_tb;

wire [WIDTH-1:0]q\_tb;

d\_reg DUT (.clk(clk), .reset(reset), .en(en), .d(d\_tb), .q(q\_tb));

integer temp\_val;

initial

begin

for(temp\_val=0; temp\_val<256; temp\_val=temp\_val+1)

begin

clk= 0;

reset=0;

d\_tb=temp\_val;

#5;

if(q\_tb == d\_tb)

begin

$display("Error when clock =0 and rst=0");

$stop;

end

clk=1;

en=0;

#1;

if(q\_tb == d\_tb)

begin

$display("Error, when clk=1 and enable=0");

$stop;

end

en=1;

#5;

clk = 0;

#5;

clk = 1;

#1;

if(q\_tb != d\_tb)

begin

$display("Error, expected %d, output was %d", d\_tb, q\_tb);

$stop;

end

reset=1;

#1;

if(q\_tb != 0)

begin

$display("Error, when reset=1 and clock =1");

$stop;

end

end

$display("Test Bench Success");

$stop;

end

endmodule

Top\_level\_tb.v

`timescale 1ns / 1ps

//////////////////////////////////////////////////////////////////////////////////

// Company:

// Engineer:

//

// Create Date: 03/22/2017 04:15:14 PM

// Design Name:

// Module Name: top\_level\_tb

// Project Name:

// Target Devices:

// Tool Versions:

// Description:

//

// Dependencies:

//

// Revision:

// Revision 0.01 - File Created

// Additional Comments:

//

//////////////////////////////////////////////////////////////////////////////////

module top\_level\_tb();

reg [3:0] a, b;

reg CLK\_tb;

reg RST\_tb;

reg EN\_tb;

reg [7:0] temp1, temp2, temp3;

wire [7:0] mem[2:0];

wire [7:0] sum;

wire [7:0] check0, check1, check2, check3, check4;

top\_level DUT0 (a, b, sum);

//top\_level DUT1(.a(A\_Mul), .b(B\_Mul), .top\_level\_out(PMul));

integer i,j,k, x, count;

initial

begin

a = 4'd0;

b = 4'd0;

RST\_tb = 1;

#1;

for(i= 0; i<16; i = i + 1)

begin

for(x=0; x<16; x = x + 1)

begin

A\_Mul= i;

A\_IM= i;

B\_Mul=x;

B\_IM=x;

#1;

if(PMul!=PIM)

begin

$display("Error");

$stop;

end

end

end

$display("succesfull");

$stop;

end

initial

begin

#5;

for(i = 0; i < 16; i = i + 1)

begin

a = i;

temp1 = sum;

for(j = 0; j < 16; j = j + 1)

begin

b = j;

temp3 = temp2;

temp2 = temp1;

temp1 = a \* b;

CLK\_tb = 1;

#5 CLK\_tb = 0;

#5;

if(temp3 != a \* b && CLK\_tb == 1)

$display("Error");

end

end

$display ("succesfull");

$finish;

end

endmodule