FLOATING POINT SYSTOLIC ARRAY

BY VIDHAYA DATTA REDDY METTA

Abstract

This study presents the design and simulation of an 8x8 systolic array for performing matrix multiplication with 32-bit floating point numbers. Leveraging parallelism and pipelining, the array efficiently processes subsets of input matrices, propagating partial products along rows and columns. Through Verilog hardware description languages and industry-standard simulation tools (Vivado), we simulate the systolic array with a test bench. Outputs are compared and evaluations are made.

Explanation and Results

- Adder
- Multiplier
- PE Block
- 4X4 Systolic Array
- 8X8 Systolic Array

Problems

An 8x8 Systolic Array for Floating point numbers needs floating point adders and FP multipliers.

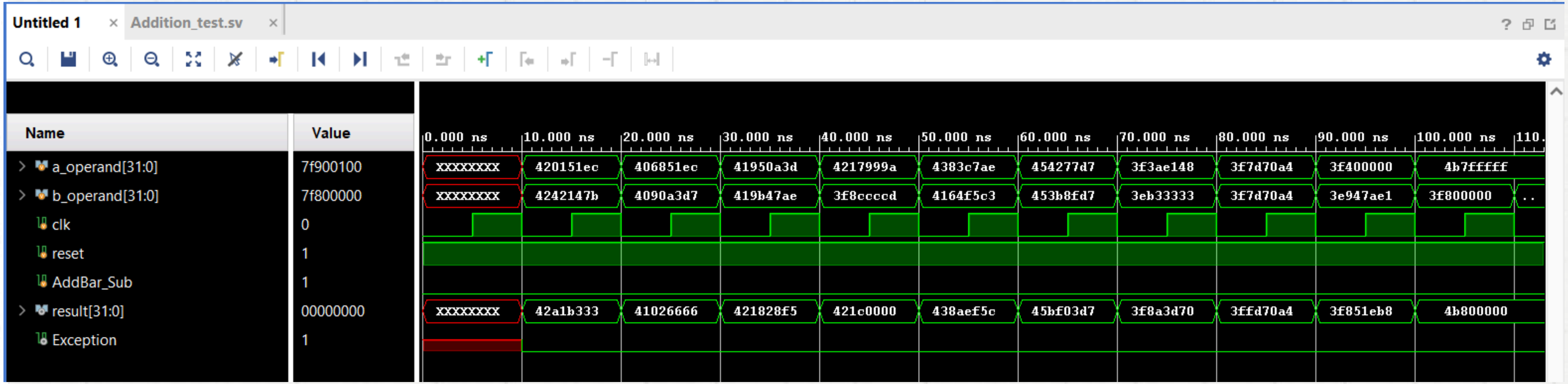
Code Snippet - Adder

```
assign significand_add = (perform & operation_sub_addBar) ? (significand_a + significand_b_add_sub) : 25'd0;

//Result will be equal to Most 23 bits if carry generates else it will be Least 22 bits.
assign add_sum[22:0] = significand_add[24] ? significand_add[23:1] : significand_add[22:0];

//If carry generates in sum value then exponent must be added with 1 else feed as it is.
assign add_sum[30:23] = significand_add[24] ? (1'b1 + operand_a[30:23]) : operand_a[30:23];
```

Input

```
iteration (32'h4201_51EC,32'h4242_147B,32'h42A1_B333,`__LINE__); //32.33 + 48.52 = 80.85

iteration (32'h4068_51EC,32'h4090_A3D7,32'h4102_6666,`__LINE__); //3.63 + 4.52 = 8.15.

iteration (32'h4195_0A3D,32'h419B_47AE,32'h4218_28F6,`__LINE__); //18.63 + 19.41 = 38.04.

iteration (32'h4217_999A,32'h3F8C_CCCD,32'h421C_0000,`__LINE__); //37.9 + 1.1 = 39.

iteration (32'h4383_C7AE,32'h4164_F5C3,32'h438A_EF5C,`__LINE__); //263.56 + 14.31 = 277.87

iteration (32'h4542_77D7,32'h453B_8FD7,32'h45BF_03D7,`__LINE__); //3111.49 + 3000.99 = 6112.48

iteration (32'h3F3A_E148,32'h3EB33333,32'h3F8A_3D71,`__LINE__); //0.73 + 0.35 = 1.08.

iteration (32'h3F7D_70A4,32'h3F7D_70A4,32'h3FFD_70A4,`__LINE__); //0.99 + 0.99 = 1.98

iteration (32'h3F40_0000,32'h3E94_7AE1,32'h3F85_1EB8,`__LINE__); //0.75 + 0.29 = 1.04
```

Output


Untitled 1
Addition_test.sv
Name
Value
a_operand[31:0]
7f900100
b_operand[31:0]
7f800000
clk
0
reset
1
AddBar_Sub
1
result[31:0]
00000000
Exception
1
0.000 ns
10.000 ns
20.000 ns
30.000 ns
40.000 ns
50.000 ns
60.000 ns
70.000 ns
80.000 ns
90.000 ns
100.000 ns
xxxxxxxx
420151ec
406851ec
41950a3d
4217999a
4383c7ae
454277d7
3f3ae148
3f7d70a4
3f400000
4b7fffff
xxxxxxxx
4242147b
4090a3d7
419b47ae
3f8ccccd
4164f5c3
453b8fd7
3eb33333
3f7d70a4
3e947ae1
3f800000
xxxxxxxx
42a1b333
41026666
421828f5
421c0000
438aef5c
45bf03d7
3f8a3d70
3ffd70a4
3f851eb8
4b800000

Comparison

```
Test Passed - Line Number ->           44
Test Passed - Line Number ->           46
Test Failed
 A => 41950a3d,
 B => 419b47ae,
 Result Obtained => 421828f5,
 Expected Value => 421828f6 - Line Number          48
Test Passed - Line Number ->           50
Test Passed - Line Number ->           52
Test Passed - Line Number ->           54
```

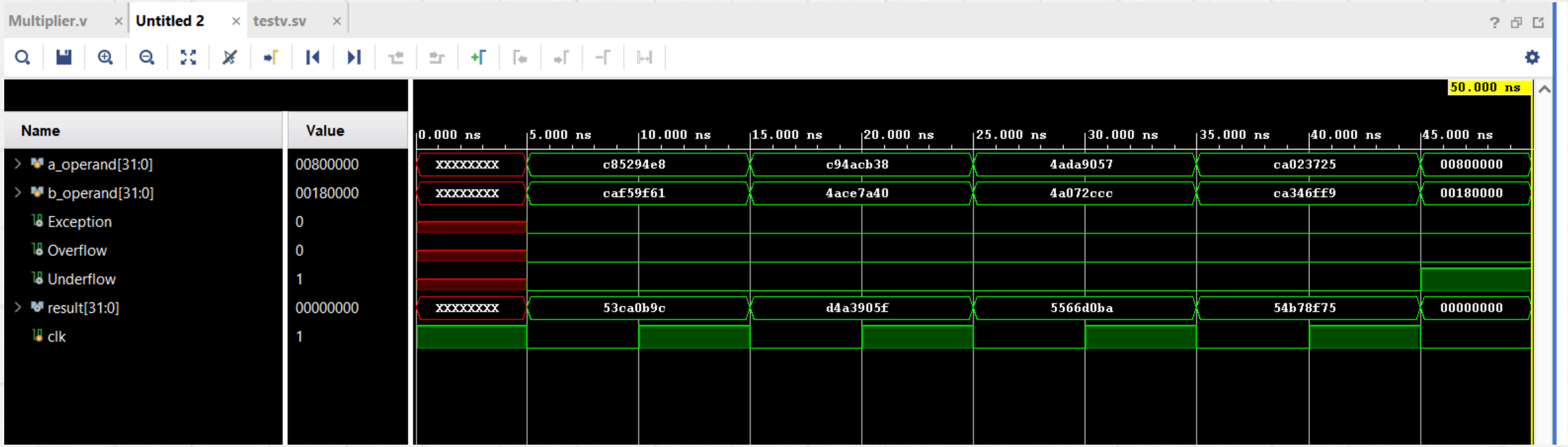
Code Snippet - Multiplier

```
//Exception flag sets 1 if either one of the exponent is 255.
assign Exception = (&a_operand[30:23]) | (&b_operand[30:23]);

//Assigining significand values according to Hidden Bit.
//If exponent is equal to zero then hidden bit will be 0 for that respective significand else it will be 1

assign operand_a = (|a_operand[30:23]) ? {1'b1,a_operand[22:0]} : {1'b0,a_operand[22:0]};

assign operand_b = (|b_operand[30:23]) ? {1'b1,b_operand[22:0]} : {1'b0,b_operand[22:0]};

assign product = operand_a * operand_b;          //Calculating Product

assign product_round = |product_normalised[22:0];  //Ending 22 bits are OR'ed for rounding operation.

assign normalised = product[47] ? 1'b1 : 1'b0;

assign product_normalised = normalised ? product : product << 1;     //Assigning Normalised value based on 48th bit

//Final Manitssa.
assign product_mantissa = product_normalised[46:24] + (product_normalised[23] & product_round);

assign zero = Exception ? 1'b0 : (product_mantissa == 23'd0) ? 1'b1 : 1'b0;

assign sum_exponent = a_operand[30:23] + b_operand[30:23];

assign exponent = sum_exponent - 8'd127 + normalised;

assign Overflow = ((exponent[8] & !exponent[7]) & !zero) ; //If overall exponent is greater than 255 then Overflow condition.
//Exception Case when exponent reaches its maximu value that is 384.
```

Input

```
initial
begin
iteration (32'hc85294e8,32'hcaf59f61,1'b0,1'b0,1'b0,32'h53ca0b9c,`__LINE__); // 45.13 * 63.13 = 2849.0569;

iteration (32'hc94acb38,32'h4ace7a40,1'b0,1'b0,1'b0,32'hd4a3905f,`__LINE__); //3.15 * -14.39 = -45.3285

iteration (32'h4ada9057,32'h4a072ccc,1'b0,1'b0,1'b0,32'h5566d0ba,`__LINE__); //-13.15 * -48.16 = 633.304

//iteration (32'h4580_0000,32'h4580_0000,1'b0,1'b0,1'b0,32'h4B80_0000,`__LINE__); //4096 * 4096 = 16777216

//iteration (32'h3ACA_62C1,32'h3ACA_62C1,1'b0,1'b0,1'b0,32'h361F_FFE7,`__LINE__); //0.00154408081 * 0.00154408081 = 0.00000238418

//iteration (32'h0000_0000,32'h0000_0000,1'b0,1'b0,1'b0,32'h0000_0000,`__LINE__); // 0 * 0 = 0;

//iteration (32'hC152_6666,32'h0000_0000,1'b0,1'b0,1'b0,32'h441E_5375,`__LINE__); //-13.15 * 0 = 0;

iteration (32'hca023725,32'hca346ff9,1'b0,1'b0,1'b0,32'h54b78f75,`__LINE__);

iteration (32'h0080_0000,32'h00180_000,1'b0,1'b0,1'b1,32'h0000_0000,`__LINE__);
```

Output


Multiplier.v
Untitled 2
testv.sv
Name
Value
a_operand[31:0]
00800000
b_operand[31:0]
00180000
Exception
0
Overflow
0
Underflow
1
result[31:0]
00000000
clk
1
50.000 ns
0.000 ns
5.000 ns
10.000 ns
15.000 ns
20.000 ns
25.000 ns
30.000 ns
35.000 ns
40.000 ns
45.000 ns
xxxxxxxx
c85294e8
c94acb38
4ada9057
ca023725
00800000
xxxxxxxx
caf59f61
4ace7a40
4a072ccc
ca346ff9
00180000
xxxxxxxx
53ca0b9c
d4a3905f
5566d0ba
54b78f75
00000000

Comparison

```
# }
# run 1000ns
Test Passed -           15
Test Passed -           17
Test Passed -           19
Test Passed -           29
Test Passed -           31
$stop called at time : 50 ns : File "D:/CDAC Stuff/Multiplier.srcs/sim_1/new/testv.sv" Line 33
```

Code Snippet - PE Block

```
  // Instantiate FPU multiplier module
  mul fpu_multiplier (
    .flp_a(flp_a),
    .flp_b(flp_b),
    .sign(sign),
    .exponent(exponent),
    .exp_unbiased(exp_unbiased),
    .exp_sum(exp_sum),
    .prod(prod),
    .sum(sum)
  );

  Addition_Subtraction AD(

 .a_operand(sum),.b_operand(result2), //Inputs in the format of IEEE-754 Representation.
.AddBar_Sub(AddBar),    //If Add_Sub is low then Addition else Subtraction.
.Exception(Exception),
.result(result1) //Outputs in the format of IEEE-754 Representation.
);

    always @(posedge rst or posedge clk) begin
        if(rst) begin
            //result <= 0;
            outp_east <= 0;
            outp_south <= 0;
            result2 <= 0;
            //a_operand <= 0;
            //b_operand <= 0;
        end
        else begin
            //a_operand <= flp_a;
           // b_operand <= flp_b;
            //sum2 <= sum2 + sum;
            //b_operand <= sum2;
            result2 <= result1;

            outp_east <= flp_a;
            outp_south <= flp_b;
        end
    end
```

```
    assign result = result1;
endmodule
```

Code Snippet - Sys4

```
//from north and west
block P0 (inp_north0, inp_west0, clk, rst,AddBar, outp_south0, outp_east0, result0);
//from north
block P1 (inp_north1, outp_east0, clk, rst,AddBar, outp_south1, outp_east1, result1);
block P2 (inp_north2, outp_east1, clk, rst,AddBar, outp_south2, outp_east2, result2);
block P3 (inp_north3, outp_east2, clk, rst,AddBar, outp_south3, outp_east3, result3);

//from west
block P4 (outp_south0, inp_west4, clk, rst,AddBar, outp_south4, outp_east4, result4);
block P8 (outp_south4, inp_west8, clk, rst,AddBar, outp_south8, outp_east8, result8);
block P12 (outp_south8, inp_west12, clk, rst,AddBar, outp_south12, outp_east12, result12);

//no direct inputs
//second row
block P5 (outp_south1, outp_east4, clk, rst,AddBar, outp_south5, outp_east5, result5);
block P6 (outp_south2, outp_east5, clk, rst,AddBar, outp_south6, outp_east6, result6);
block P7 (outp_south3, outp_east6, clk, rst,AddBar, outp_south7, outp_east7, result7);
//third row
block P9 (outp_south5, outp_east8, clk, rst,AddBar, outp_south9, outp_east9, result9);
block P10 (outp_south6, outp_east9, clk, rst,AddBar, outp_south10, outp_east10, result10);
block P11 (outp_south7, outp_east10, clk, rst,AddBar, outp_south11, outp_east11, result11);
//fourth row
block P13 (outp_south9, outp_east12, clk, rst,AddBar, outp_south13, outp_east13, result13);
block P14 (outp_south10, outp_east13, clk, rst,AddBar, outp_south14, outp_east14, result14);
block P15 (outp_south11, outp_east14, clk, rst,AddBar, outp_south15, outp_east15, result15);
```

Code Snippet - Sys8

```
//first row
    sys4 A0 ( iw0, iw1, iw2, iw3, in0, in1, in2, in3 , clk,rst, Addbar, oe0_0, oe0_1, oe0_2, oe0_3, os0_0, os0_1, os0_2, os0_3, result0, result1, result2, result3, result4, result5, result6, result7
    sys4 A1 ( oe0_0, oe0_1, oe0_2, oe0_3, in4, in5, in6, in7 ,  clk,rst, Addbar,  oe1_0, oe1_1, oe1_2, oe1_3, os1_0, os1_1, os1_2, os1_3, result16, result17, result18, result19, result20, result21,

//second row
    sys4 A2 ( iw4, iw5, iw6, iw7, os0_0, os0_1, os0_2, os0_3  ,  clk,rst, Addbar, oe2_0, oe2_1, oe2_2, oe2_3, os2_0, os2_1, os2_2, os2_3, result32, result33, result34, result35, result36, result37,
    sys4 A3 ( oe2_0, oe2_1, oe2_2, oe2_3, os1_0, os1_1, os1_2, os1_3 ,  clk,rst, Addbar,  oe3_0, oe3_1, oe3_2, oe3_3, os3_0, os3_1, os3_2, os3_3, result48, result49, result50, result51, result52, re
```

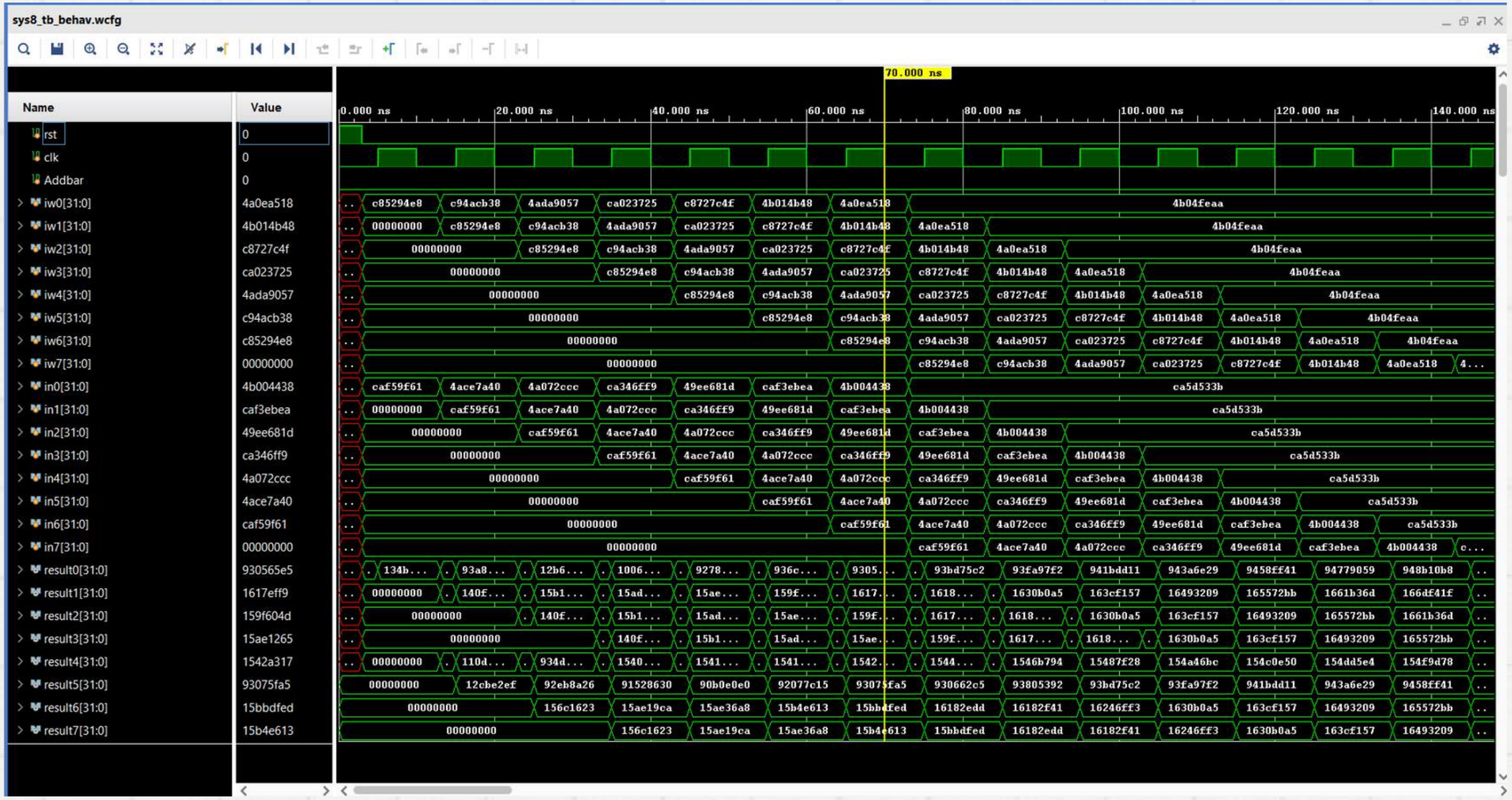
Inputs

```
 initial begin
#3 iw0 <= 32'hc85294e8;
    in0 <= 32'hcaf59f61;
#10 iw0 <= 32'hc94acb38;
    in0 <= 32'h4ace7a40;
#10 iw0 <= 32'h4ada9057;
    in0 <= 32'h4a072ccc;
#10 iw0 <= 32'hca023725;
    in0 <= 32'hca346ff9;
#10 iw0 <= 32'hc8727c4f;
    in0 <= 32'h49ee681d;
#10 iw0 <= 32'h4b014b48;
    in0 <= 32'hcaf3ebea;
#10 iw0 <= 32'h4a0ea518;
    in0 <= 32'h4b004438;
#10 iw0 <= 32'h4b04feaa;
    in0 <= 32'hca5d533b;

end

initial begin
#3 iw1 <= 32'h0;
    in1 <= 32'h0;
#10 iw1 <= 32'hc85294e8;
    in1 <= 32'hcaf59f61;
#10 iw1 <= 32'hc94acb38;
    in1 <= 32'h4ace7a40;
#10 iw1 <= 32'h4ada9057;
    in1 <= 32'h4a072ccc;
#10 iw1 <= 32'hca023725;
    in1 <= 32'hca346ff9;
#10 iw1 <= 32'hc8727c4f;
    in1 <= 32'h49ee681d;
#10 iw1 <= 32'h4b014b48;
    in1 <= 32'hcaf3ebea;
#10 iw1 <= 32'h4a0ea518;
    in1 <= 32'h4b004438;
#10 iw1 <= 32'h4b04feaa;
    in1 <= 32'hca5d533b;
```

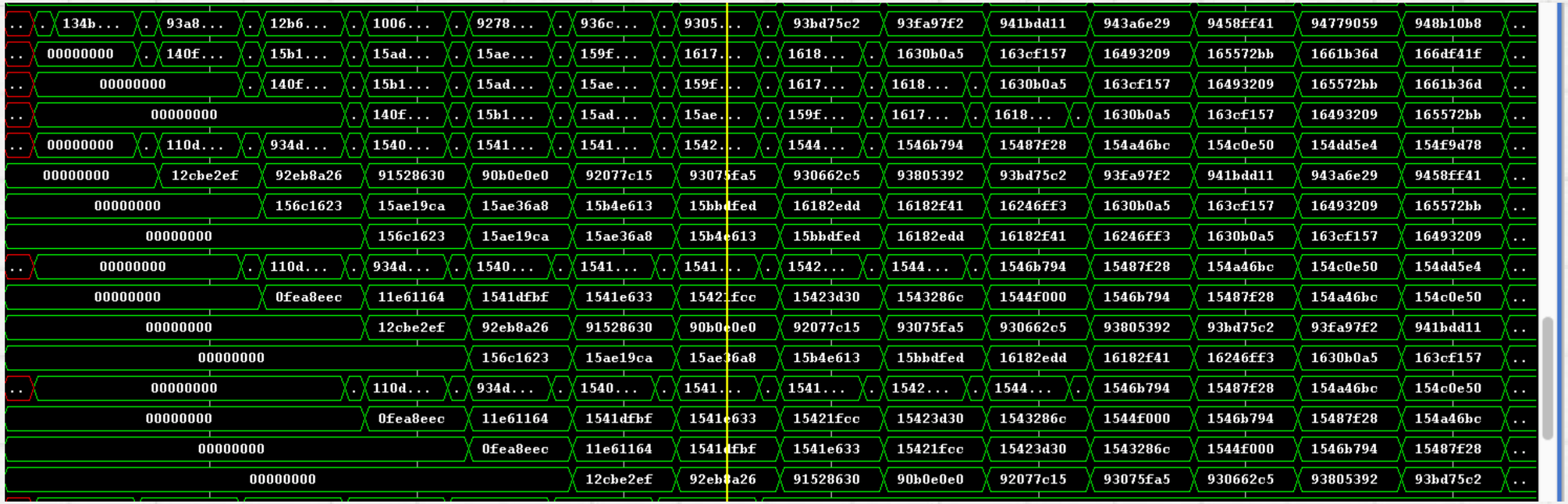
```
initial begin
#3 iw2 <= 32'h0;
    in2 <= 32'h0;
#10 iw2 <= 32'h0;
    in2 <= 32'h0;
#10 iw2 <= 32'hc85294e8;
    in2 <= 32'hcaf59f61;
#10 iw2 <= 32'hc94acb38;
    in2 <= 32'h4ace7a40;
#10 iw2 <= 32'h4ada9057;
    in2 <= 32'h4a072ccc;
#10 iw2 <= 32'hca023725;
    in2 <= 32'hca346ff9;
#10 iw2 <= 32'hc8727c4f;
    in2 <= 32'h49ee681d;
#10 iw2 <= 32'h4b014b48;
    in2 <= 32'hcaf3ebea;
#10 iw2 <= 32'h4a0ea518;
    in2 <= 32'h4b004438;
#10 iw2 <= 32'h4b04feaa;
    in2 <= 32'hca5d533b;

end

initial begin
#3 iw3 <= 32'h0;
    in3 <= 32'h0;
#10 iw3 <= 32'h0;
    in3 <= 32'h0;
#10 iw3 <= 32'h0;
    in3 <= 32'h0;
#10 iw3 <= 32'hc85294e8;
    in3 <= 32'hcaf59f61;
#10 iw3 <= 32'hc94acb38;
    in3 <= 32'h4ace7a40;
#10 iw3 <= 32'h4ada9057;
    in3 <= 32'h4a072ccc;
#10 iw3 <= 32'hca023725;
    in3 <= 32'hca346ff9;
#10 iw3 <= 32'hc8727c4f;
```

```
    in3 <= 32'h4a072ccc;
#10 iw3 <= 32'hca023725;
    in3 <= 32'hca346ff9;
#10 iw3 <= 32'hc8727c4f;
    in3 <= 32'h49ee681d;
#10 iw3 <= 32'h4b014b48;
    in3 <= 32'hcaf3ebea;
#10 iw3 <= 32'h4a0ea518;
    in3 <= 32'h4b004438;
#10 iw3 <= 32'h4b04feaa;
    in3 <= 32'hca5d533b;

end

initial begin
#3 iw4 <= 32'h0;
    in4 <= 32'h0;
#10 iw4 <= 32'h0;
    in4 <= 32'h0;
#10 iw4 <= 32'h0;
    in4 <= 32'h0;
#10 iw4 <= 32'h0;
    in4 <= 32'h0;
#10 iw4 <= 32'hc85294e8;
    in4 <= 32'hcaf59f61;
#10 iw4 <= 32'hc94acb38;
    in4 <= 32'h4ace7a40;
#10 iw4 <= 32'h4ada9057;
    in4 <= 32'h4a072ccc;
#10 iw4 <= 32'hca023725;
    in4 <= 32'hca346ff9;
#10 iw4 <= 32'hc8727c4f;
    in4 <= 32'h49ee681d;
#10 iw4 <= 32'h4b014b48;
    in4 <= 32'hcaf3ebea;
#10 iw4 <= 32'h4a0ea518;
    in4 <= 32'h4b004438;
#10 iw4 <= 32'h4b04feaa;
    in4 <= 32'hca5d533b;
```

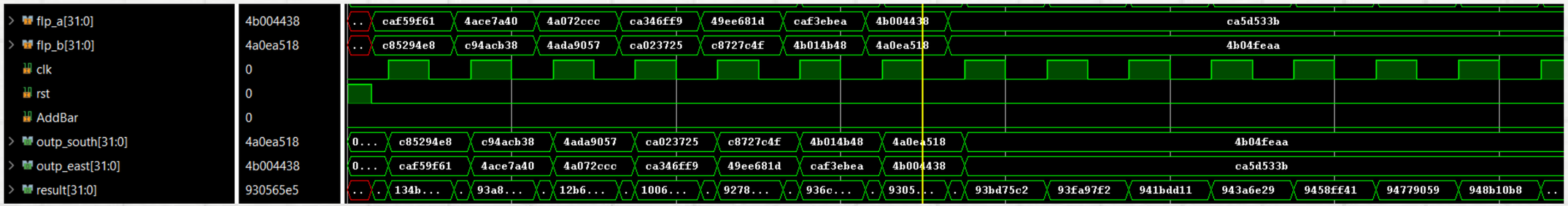
Outputs SYS8



Outputs SYS4



Outputs PE Block



Python Codes Used for comparison

```python
import random
import struct

span = 10_000_000
iteration = 100_000

def ieee754(flt):
    """Convert a floating-point number to IEEE-754 32-bit representation and return its hex format."""
    return struct.pack('f', flt).hex()

with open("TestVectorAddition", "w") as f:
    for i in range(iteration):
        # Generate random floats in the range [-span, span]
        a = random.uniform(-span, span)
        b = random.uniform(-span, span)

        # Convert the floats to IEEE-754 32-bit hex format
        a_hex = ieee754(a)
        b_hex = ieee754(b)

        # Calculate the sum of the floats
        ab = a + b

        # Convert the sum to IEEE-754 32-bit hex format
        ab_hex = ieee754(ab)
```

```python
# Write the hex representations to the file in the specified format
f.write(f"{a_hex}_{b_hex}_{ab_hex}\n")
```

Python Codes Used for comparison

C: > Users > Dell > Downloads > CARLA_0.9.12 > WindowsNoEditor > PythonAPI > CapstoneProjects >

```python
import bitstring, random

span = 10000000
iteration = 100000

def ieee754(flt):
    b = bitstring.BitArray(float=flt, length=32)
    return b

with open("TestVectorMultiply", "w") as f:

    for i in range(iteration):
        a = ieee754(random.uniform(-span, span))
        b = ieee754(random.uniform(-span, span))
        ab = ieee754(a.float * b.float)

        f.write(a.hex +"_" +  b.hex  +  "_" + ab.hex + "\n")

```

Output Format

```
a990beca_1ta24dc9_ec44d8ca
7620874a_9c0c88ca_ee25ecc6
bf8a17cb_46e1014b_c74badc9
3503a7ca_68f6e6ca_cefc46cb
8ea00ec8_f4a0f748_ac50b048
beac004b_4e6ea9ca_5ad62f4a
e038bf4a_89352d49_91dfd44a
09dd7a4a_d0154bca_e31c3f49
6baedb4a_7fb781ca_daed334a
18ff164b_385696c9_5134044b
cad813cb_50580e4b_300fb0c8
52439a4a_d640104a_bc63e24a
6bbdc9ca_5e82e34a_9e274e49
908c8c4a_15faed4a_53433d4b
9090124b_577e80ca_c8a2a44a
c8b99b4a_44251849_71beae4a
8acb01cb_20b815cb_d5c18bcb
3896124b_b327ea4a_09d5834b
9229f8ca_cbc4a54a_8fc924ca
b0ba0a49_d9c9c2c9_02d97ac9
```


Thank
you!