IMPLEMENTATION OF 32-BIT COMBINED INTEGER AND FLOATING POINT MULTIPLIER USING REVERSIBLE LOGIC

A Project Report submitted in partial fulfillment of the requirement for the award of the Degree of

BACHELOR OF TECHNOLOGY

In

ELECTRONICS AND COMMUNICATION ENGINEERING

By

CHOWDADA BHARGAVI (20JG1A0419)

KUDDIGANA GURUTEJITHA (20JG1A0450)

KARANAM DEEKSHITHA (20JG1A0440)

AKULA SRAVANI (20JG1A0401)

Project Domain : Digital VLSI Design

Batch No : 10

Under the Esteemed guidance of

Ms. B. DIVYA SATHI

Assistant professor

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

GAYATRI VIDYA PARISHAD COLLEGE OF ENGINEERING FOR WOMEN

(Affiliated to Jawaharlal Nehru Technological University Kakinada)

MADHURAWADA, VISAKHAPATNAM-48

(2020-2024)

PROJECT IMPLEMENTATION CODES:

Integer Multiplier:

PERES GATE:

CODE:

```
module PeresGate ( input a,input b,  input c, output o1,output o2,  output o3);

assign o1 = a;

assign o2 = a^b;

assign o3 = ((a&b)^c);

endmodule
```

TEST BENCH:

```
`timescale 1ns / 1ps

module PeresGate_tb;

reg a;reg b;reg c;

// Outputs

wire o1;wire o2;wire o3;

// Instantiate the Unit Under Test (UUT)

PeresGate uut (.a(a), .b(b), .c(c), .o1(o1), .o2(o2), .o3(o3));

initial begin

$dumpfile("PeresGate_power.vcd");

$dumpvars();

    a = 0; b = 0; c = 0;#100;

    a = 0; b = 0; c = 1;#100;

    a = 0; b = 1; c = 0;#100;
```

a = 0; b = 1; c = 1;#100;

a = 1; b = 0; c = 0;#100;

a = 1; b = 0; c = 1;#100;

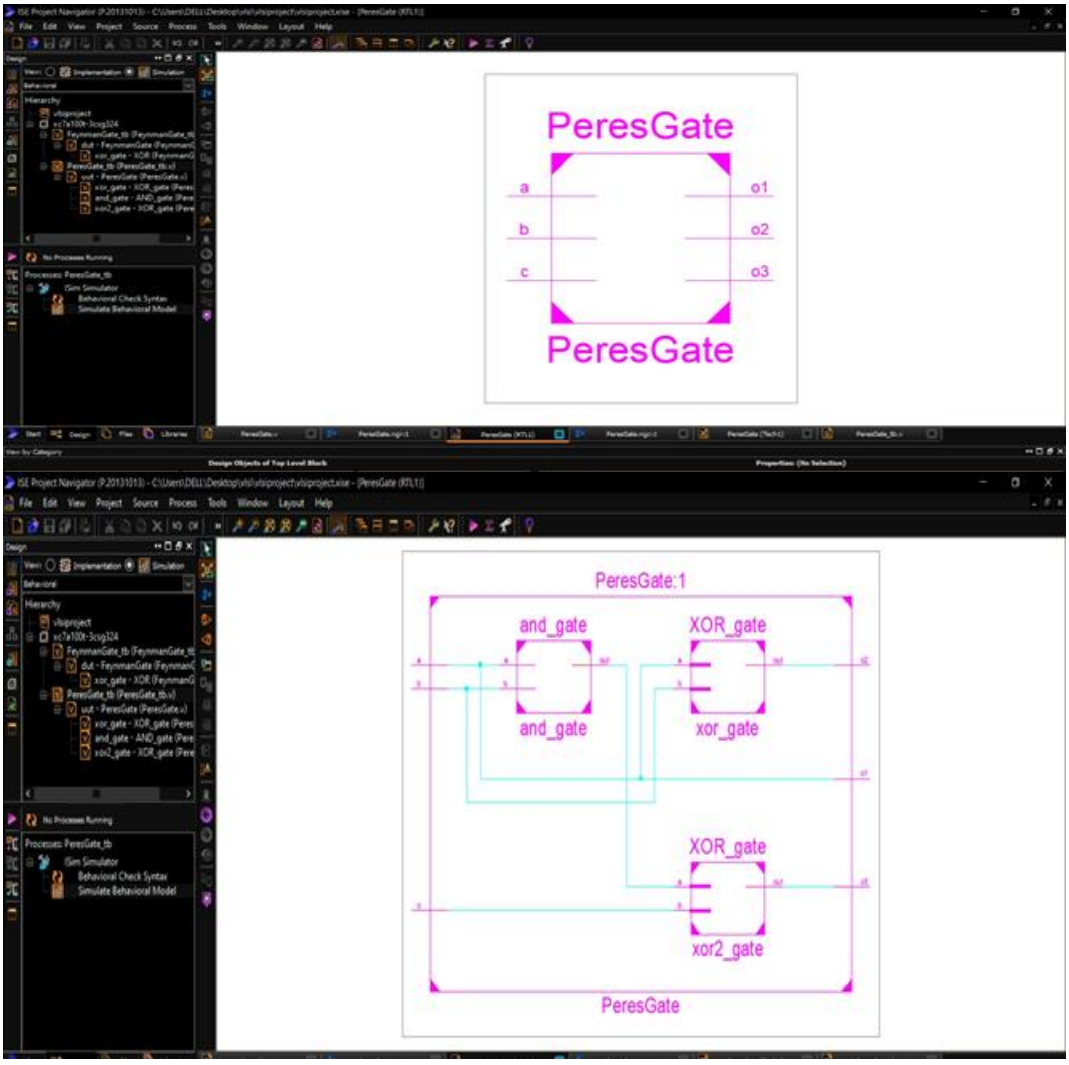
a = 1; b = 1; c = 0;#100;

a = 1; b = 1; c = 1;#100;

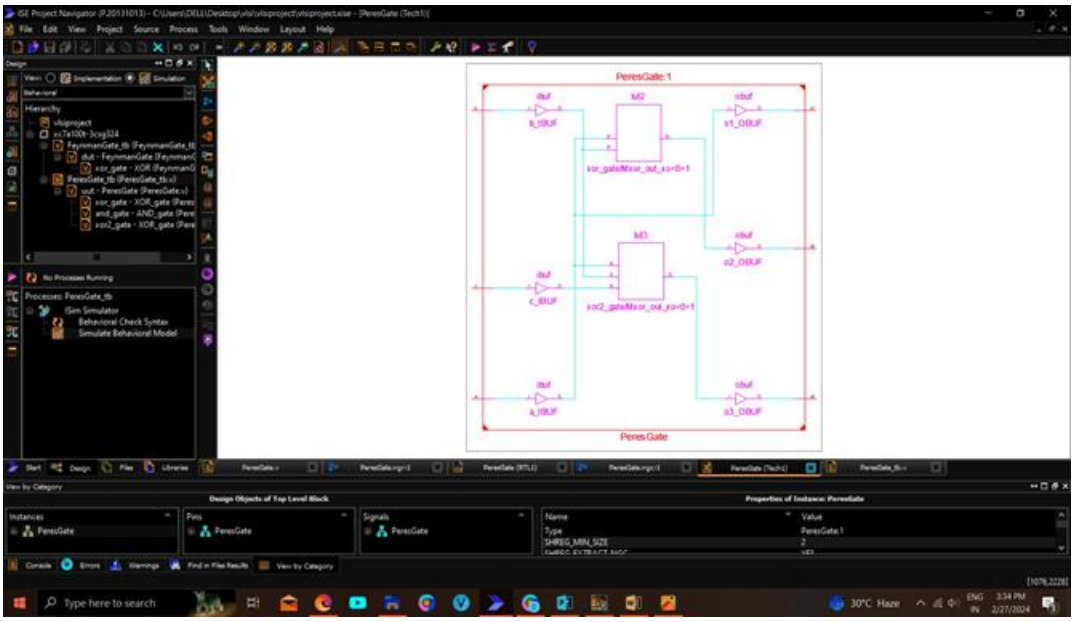
end

endmodule

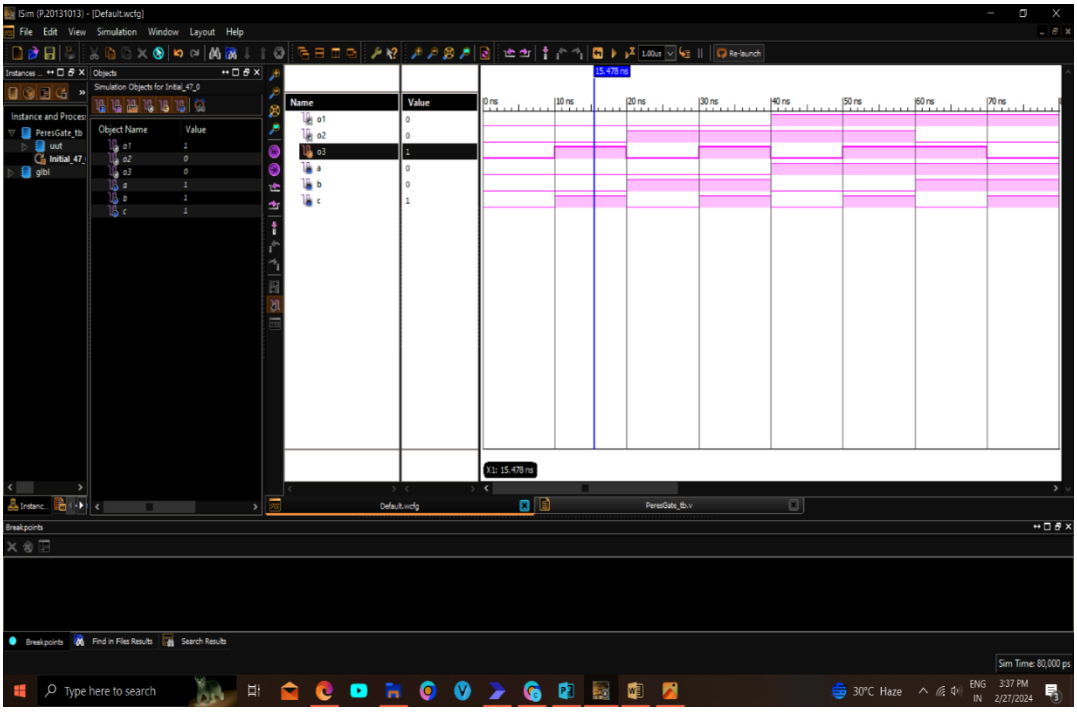
RTL SCHEMATIC



TECHNOLOGY SCHEMATIC



OUTPUT:



FEYNMAN GATE:

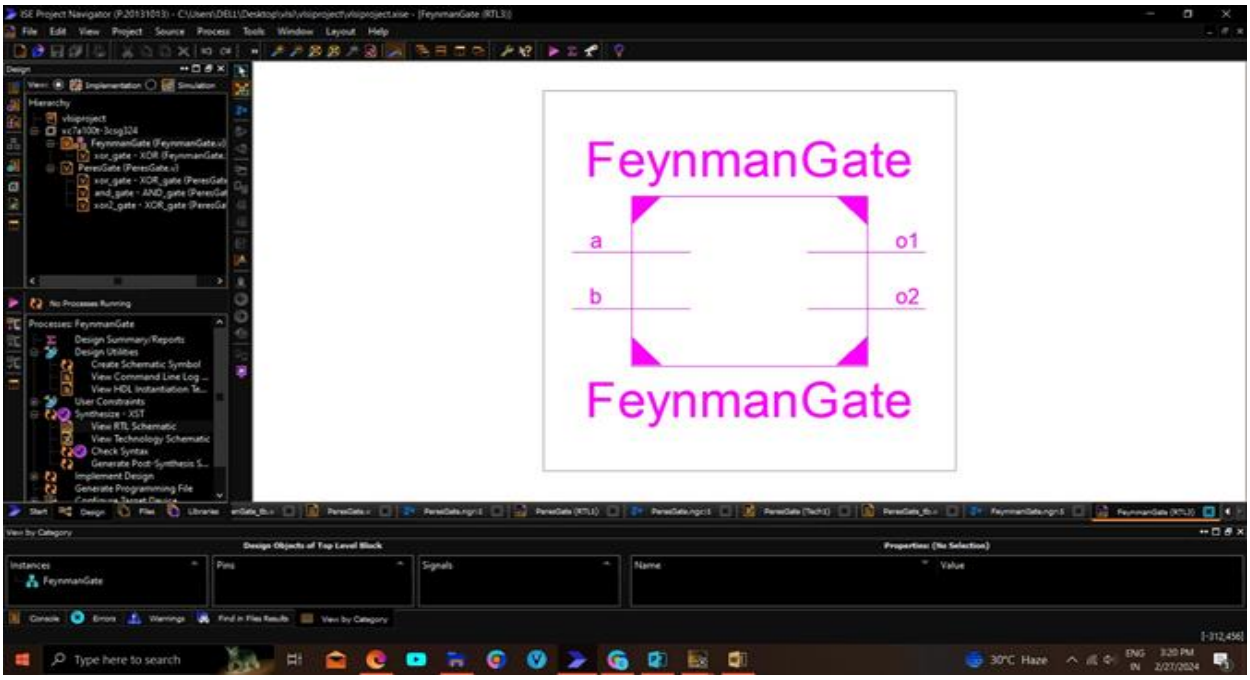
Code:

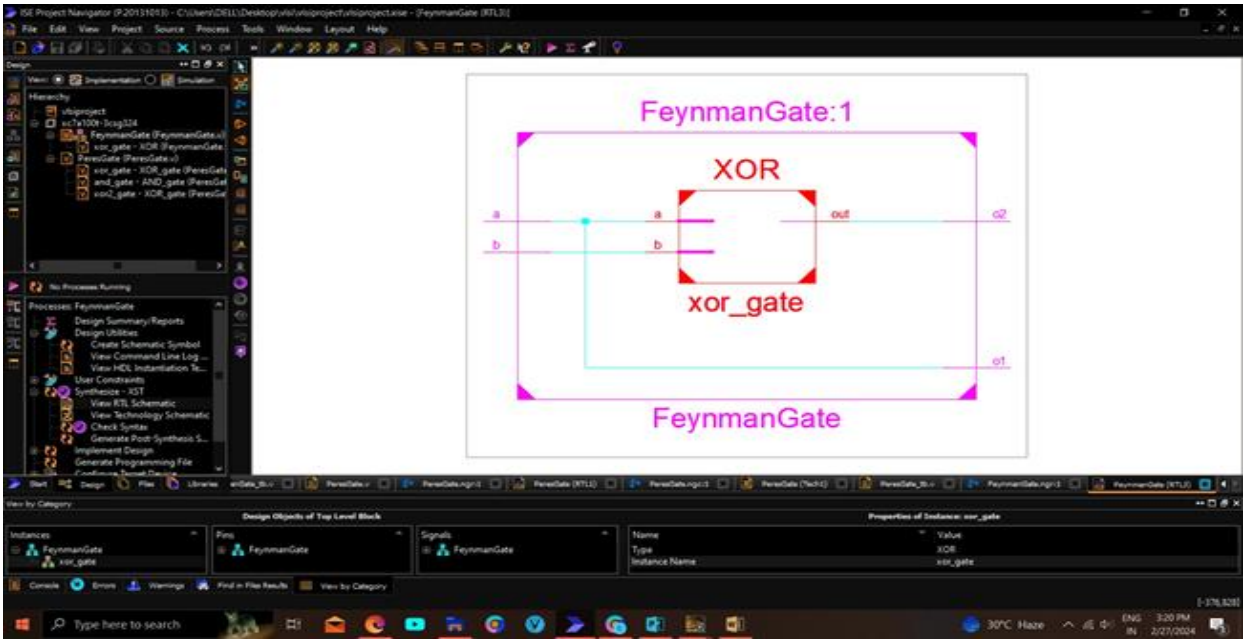
```
module Feynmangate ( input a, input b,output o1,output o2);
assign o1 = a;
assign o2 = a^b;
endmodule
```

TEST BENCH:

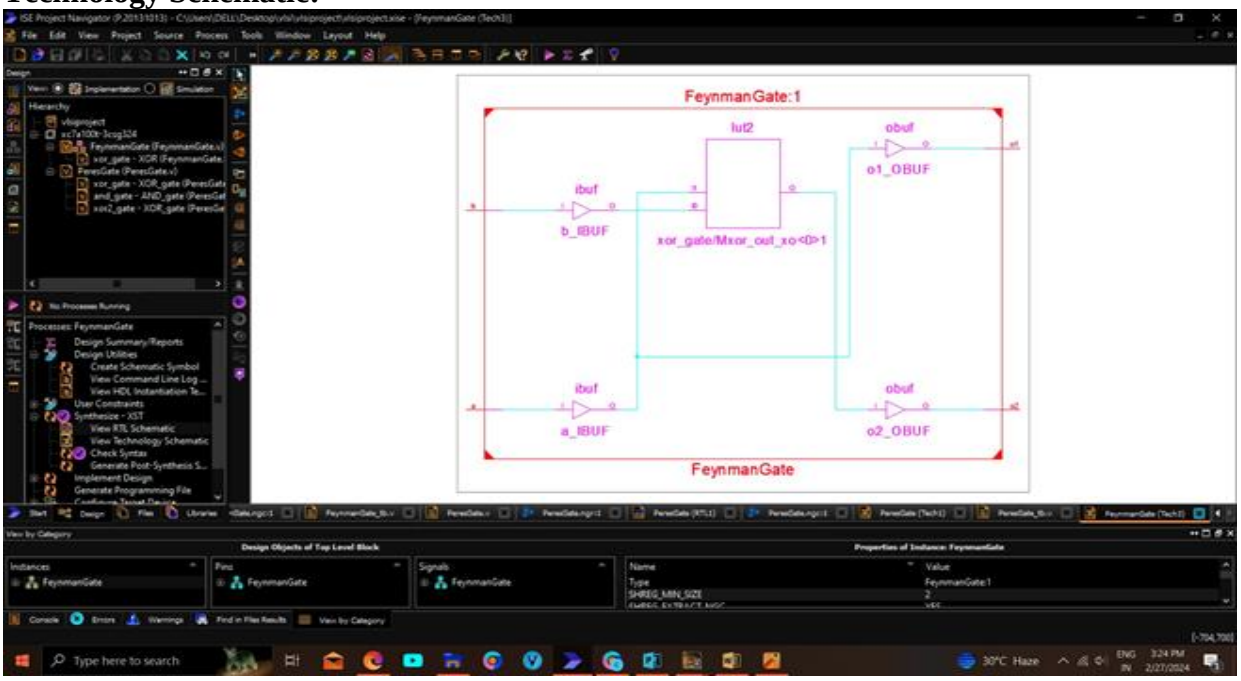
```
`timescale 1ns / 1ps
module Feynmangate_tb;
reg a;reg b;
wire o1;wire o2;
// Instantiate the Unit Under Test (UUT)
Feynmangate uut (.a(a), .b(b), .o1(o1), .o2(o2));
            initial begin
            $dumpfile("Feynman_power.vcd");
           $dumpvars();
    a = 0; b = 0;#100;
    a = 1; b = 0;#100;
    a = 0; b = 1;#100;
    a = 1; b = 1;#100;
           end
endmodule
```

RTL SCHEMATIC:

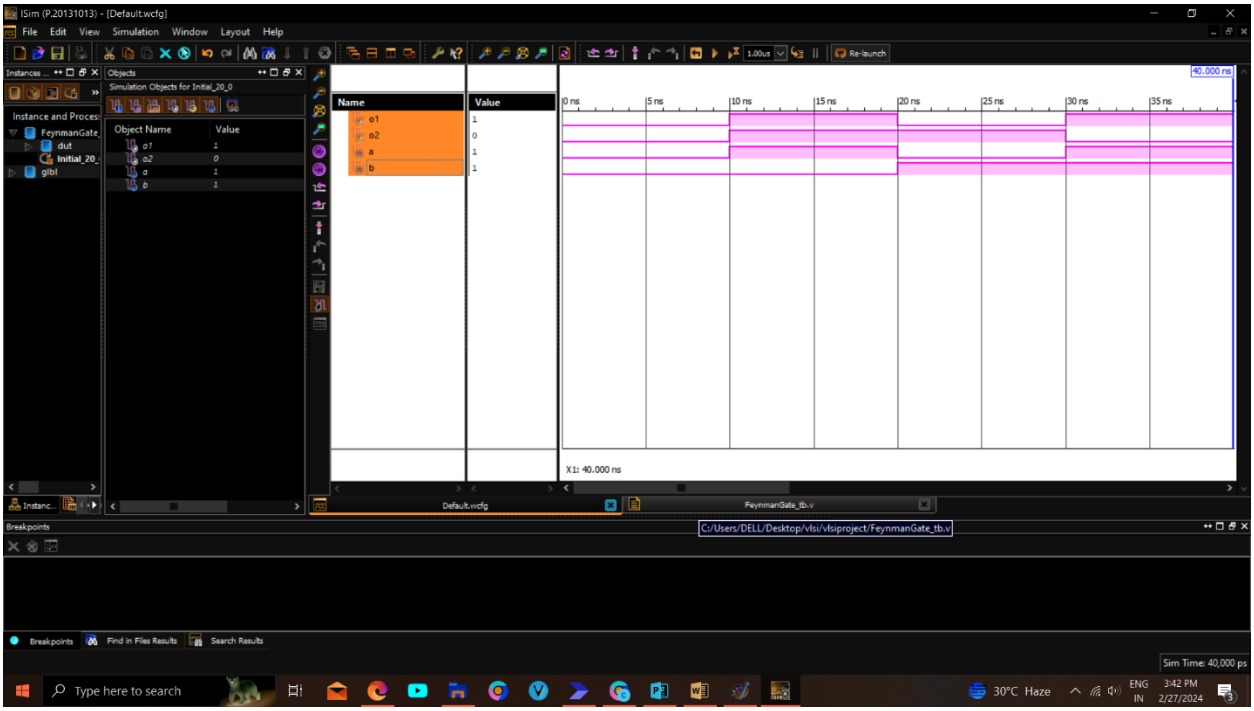




Technology Schematic:



OUTPUT:



Peres Half Adder:

CODE:

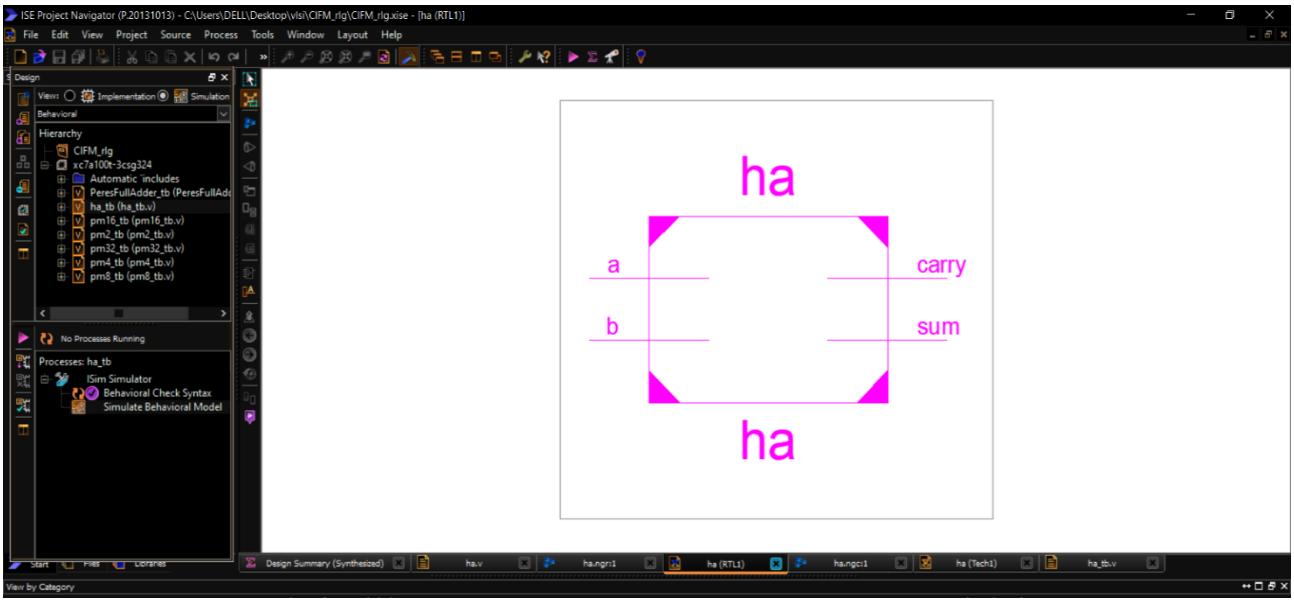
```
`include "PeresGate.v"
module ha ( input a,input b,output sum,output carry);
  wire o1, o2, o3;
  PeresGate half_adder (
    .a(a),
    .b(b),
    .c(1'b0), // c is set to 0 for the half adder
    .o1(o1),
    .o2(o2),
    .o3(o3)
  );
  assign sum = o2;
  assign carry = o3;
endmodule
```

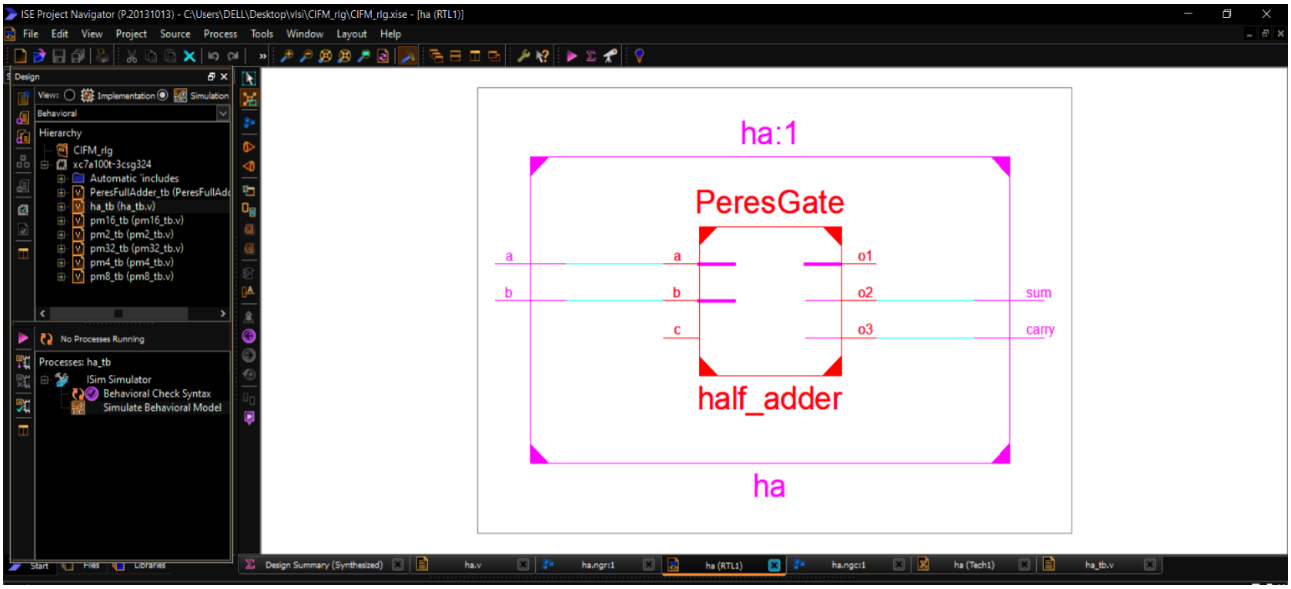
Test code:

```
`timescale 1ns / 1ps
module ha_tb;
reg a;reg b;
wire sum;wire carry;
// Instantiate the Unit Under Test (UUT)
ha uut (.a(a), .b(b), .sum(sum), .carry(carry));
initial begin
```

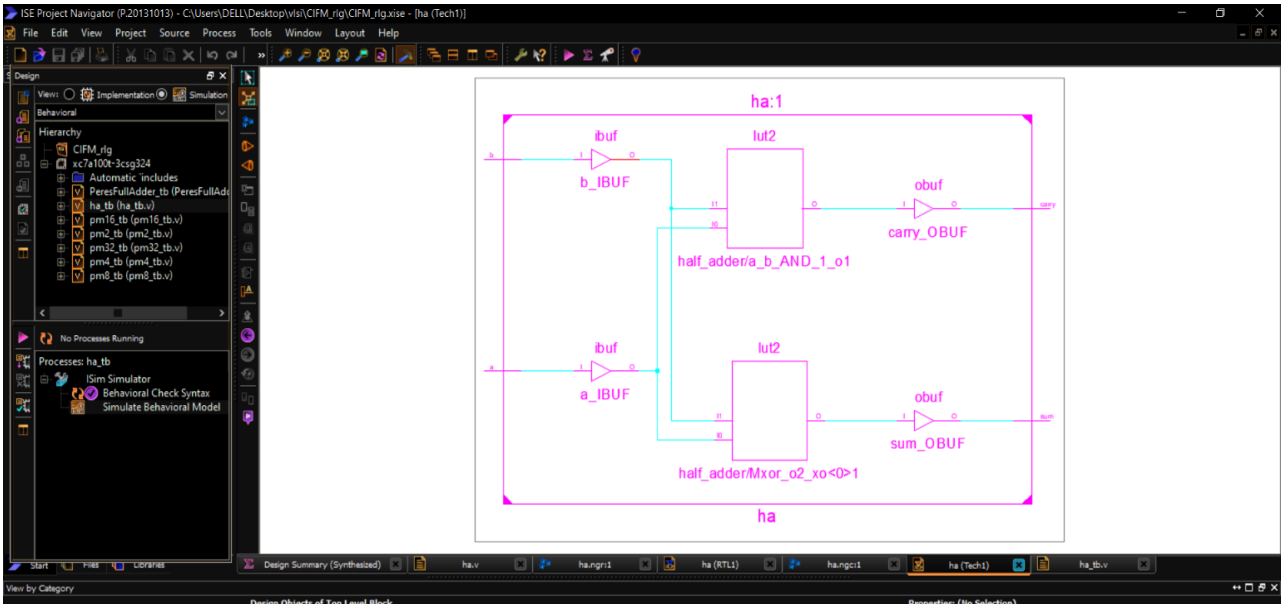
```
    a = 0; b = 0;#100;

    a = 0; b = 1;#100;

    a = 1; b = 0;#100;

    a = 1; b = 1;#100;

  end

endmodule
```

RTL Schemantic:

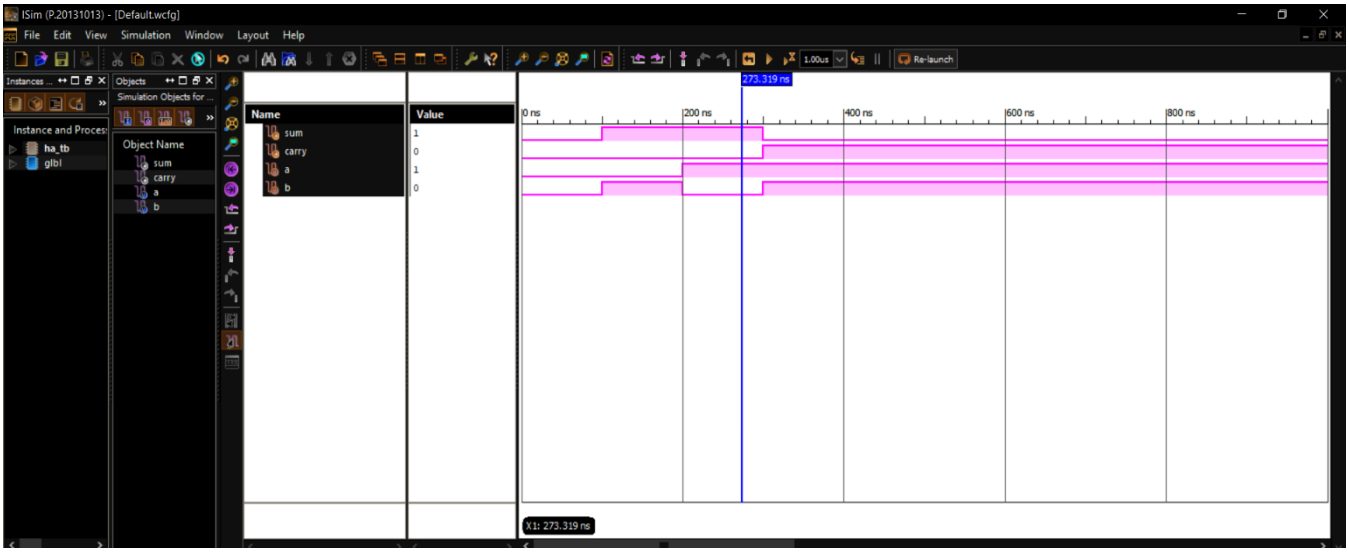




Technology Schematic:



Wave forms:



2 BIT MULTIPLIER:

Code:

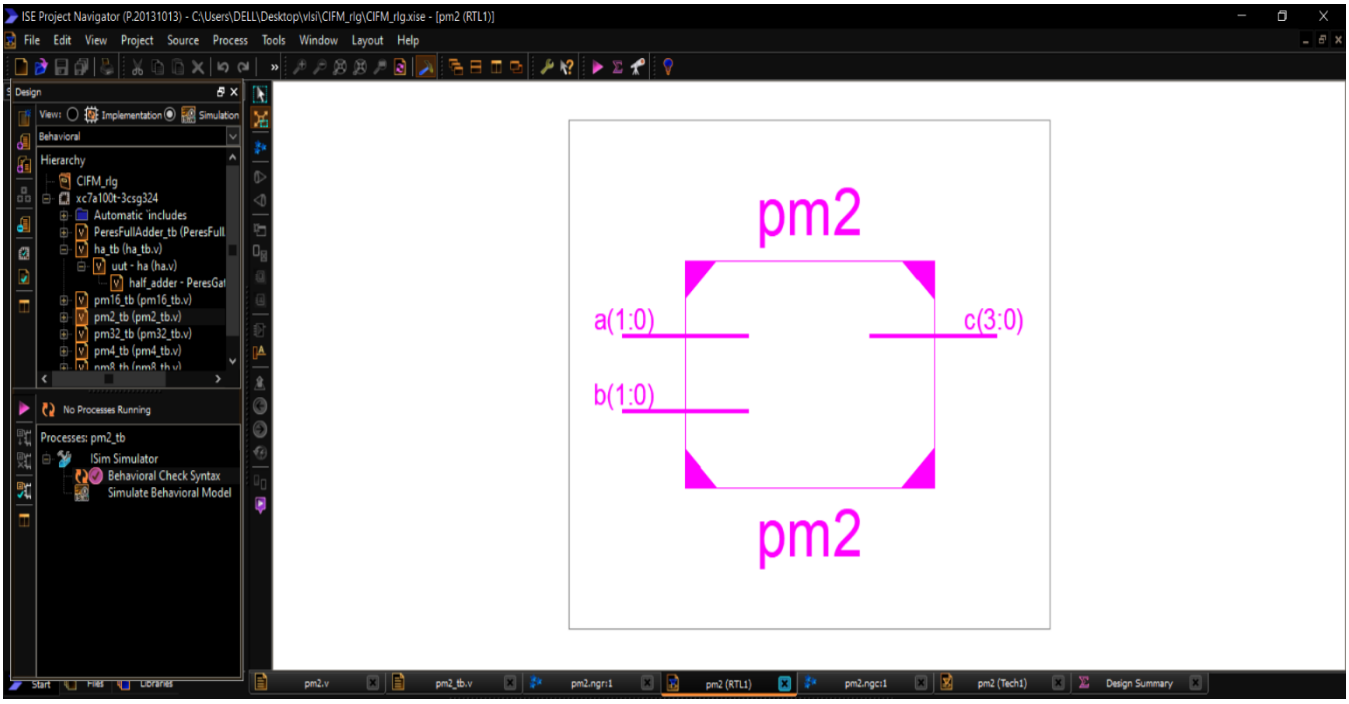
```
module pm2(a, b, c);
    input [1:0]a, b;
    output [3:0]c;
```

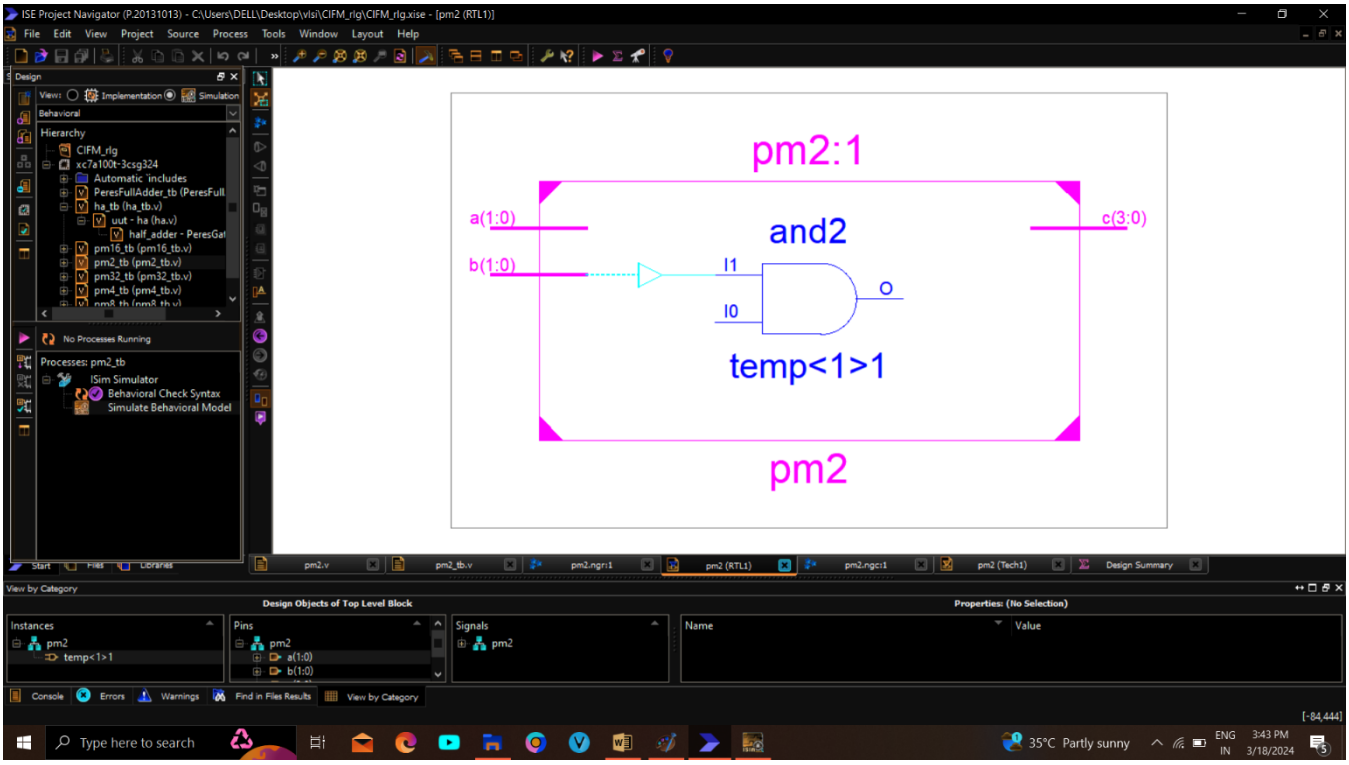
```
    wire [3:0]c, temp;
    assign c[0]=a[0]&b[0];
    assign temp[0]=a[1]&b[0];
    assign temp[1]=a[0]&b[1];
    assign temp[2]=a[1]&b[1];
    ha z1(temp[0],temp[1],c[1],temp[3]);
    ha z2(temp[2],temp[3],c[2],c[3]);
endmodule
```

Test code:

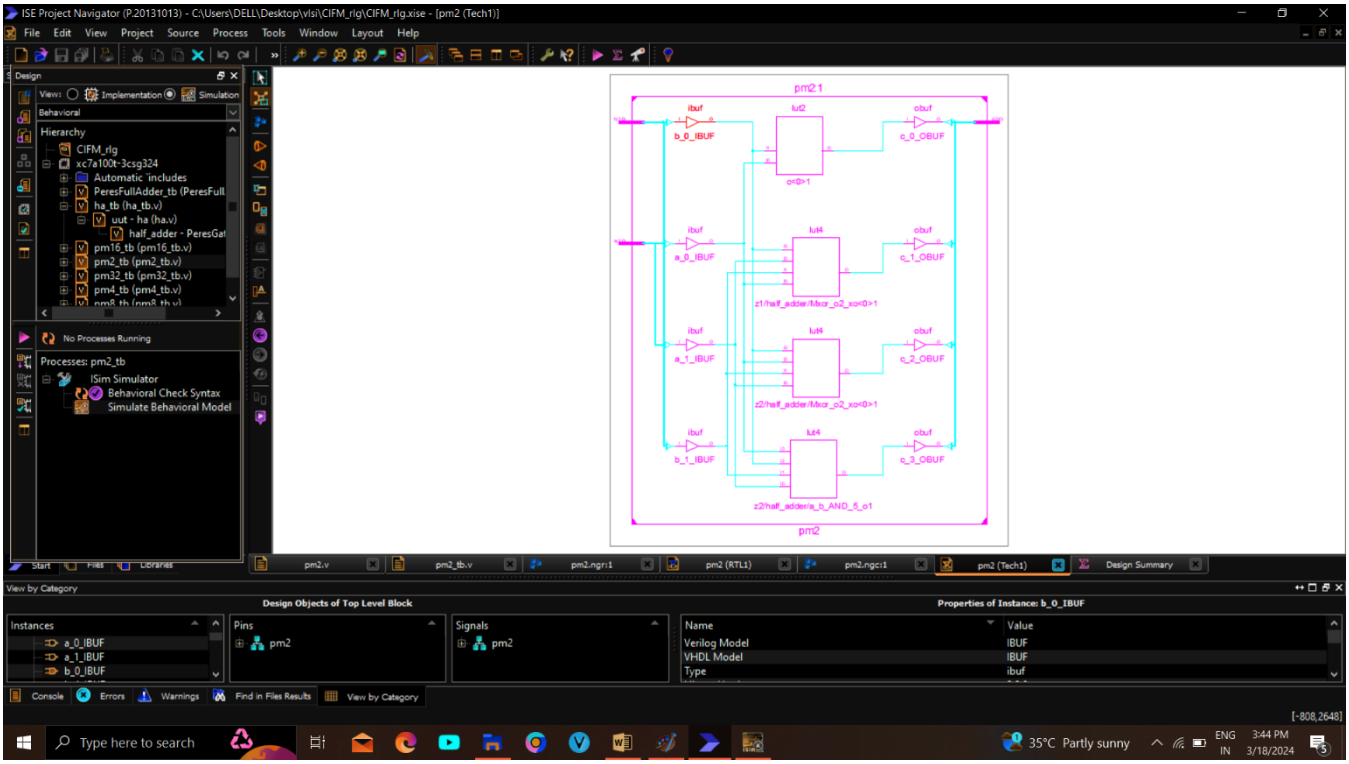
```
`timescale 1ns / 1ps
module pm2_tb;
reg [1:0] a;reg [1:0] b;
wire [3:0] c;
// Instantiate the Unit Under Test (UUT)
pm2 uut (.a(a), .b(b), .c(c));
initial begin
                a = 2'b11; b = 2'b11; #100;
               a = 2'b11; b = 2'b01; #100;
               a = 2'b10; b = 2'b01; #100;
               a = 2'b10; b = 2'b10; #100;
           end
endmodule
```

RTL Schemantics:

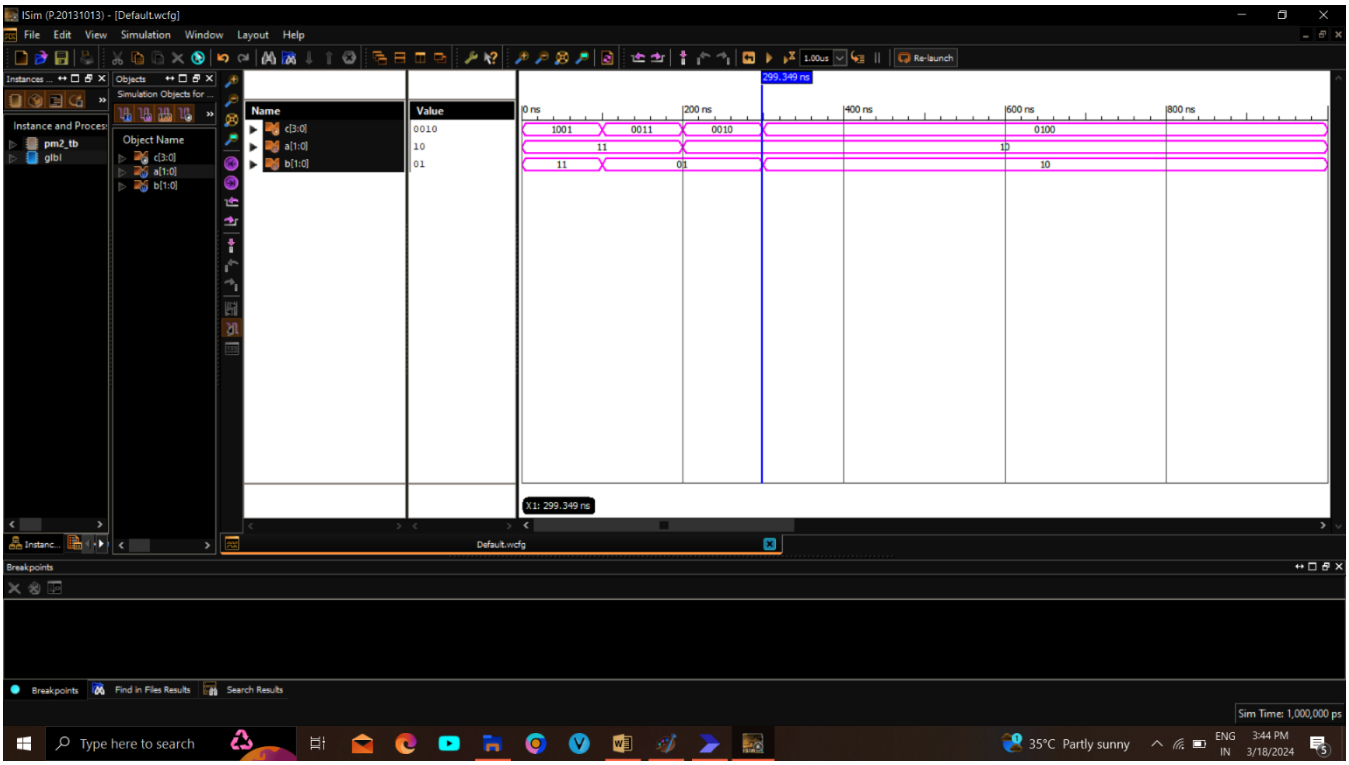




Technology Schematic:



Wave form:



4 Bit Multiplier:

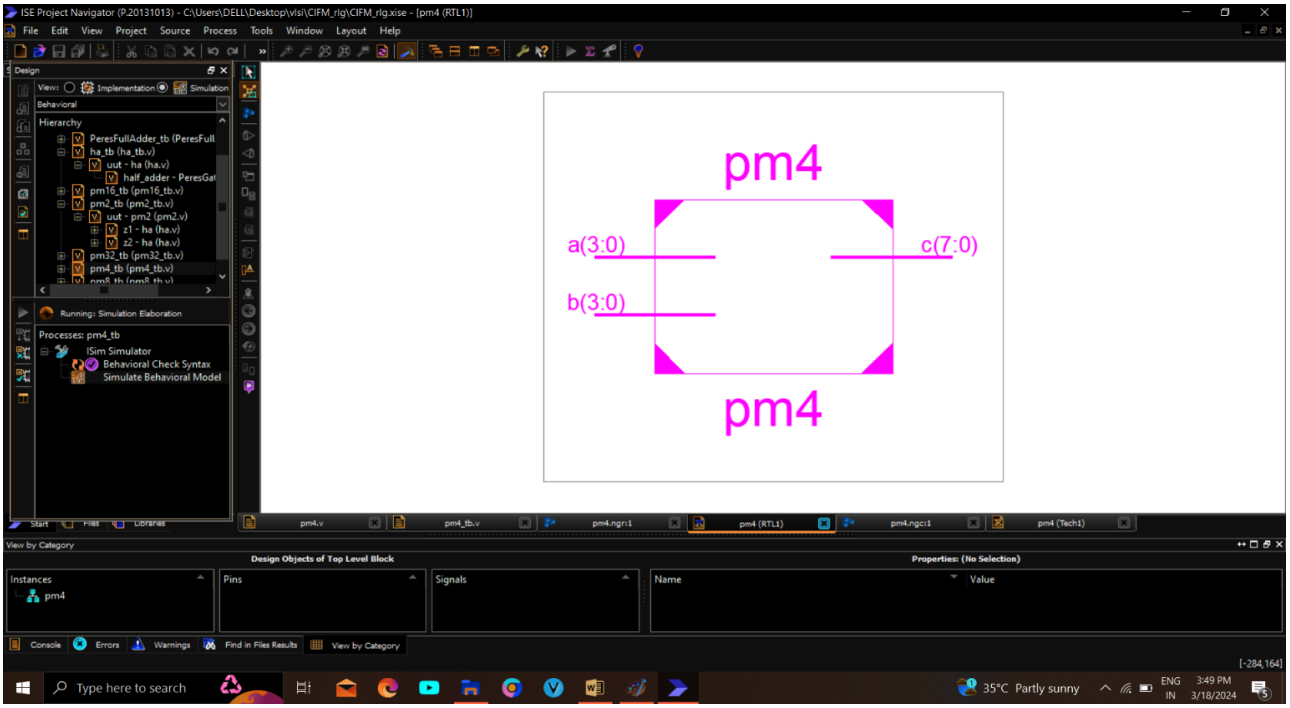
Code:

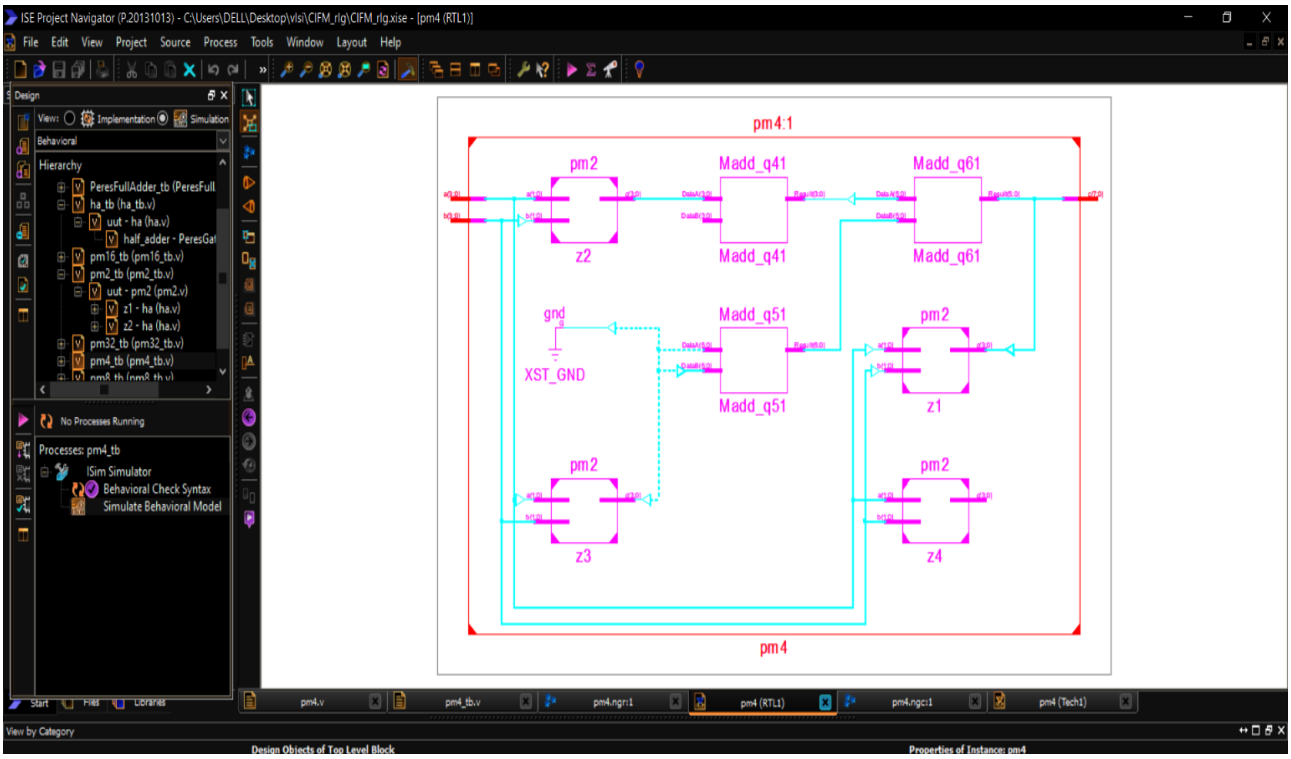
```
module pm4(a,b,c);
    input [3:0]a, b;
    output [7:0]c;
    wire [3:0]q0,q1,q2,q3,q4,temp1;
    wire [7:0]c;
    wire [5:0]q5,q6,temp2,temp3,temp4;
    pm2 z1(a[1:0],b[1:0],q0[3:0]);
    pm2 z2(a[3:2],b[1:0],q1[3:0]);
    pm2 z3(a[1:0],b[3:2],q2[3:0]);
    pm2 z4(a[3:2],b[3:2],q3[3:0]);

    assign temp1 ={2'b0,q0[3:2]};
    assign q4 = q1[3:0]+temp1;
    assign temp2 ={2'b0,q2[3:0]};
    assign temp3 ={q3[3:0],2'b0};
    assign q5 = temp2+temp3;
    assign temp4={2'b0,q4[3:0]};
    assign q6 = temp4+q5;
    assign c[1:0]=q0[1:0];
    assign c[7:2]=q6[5:0];
endmodule
```

Test code:

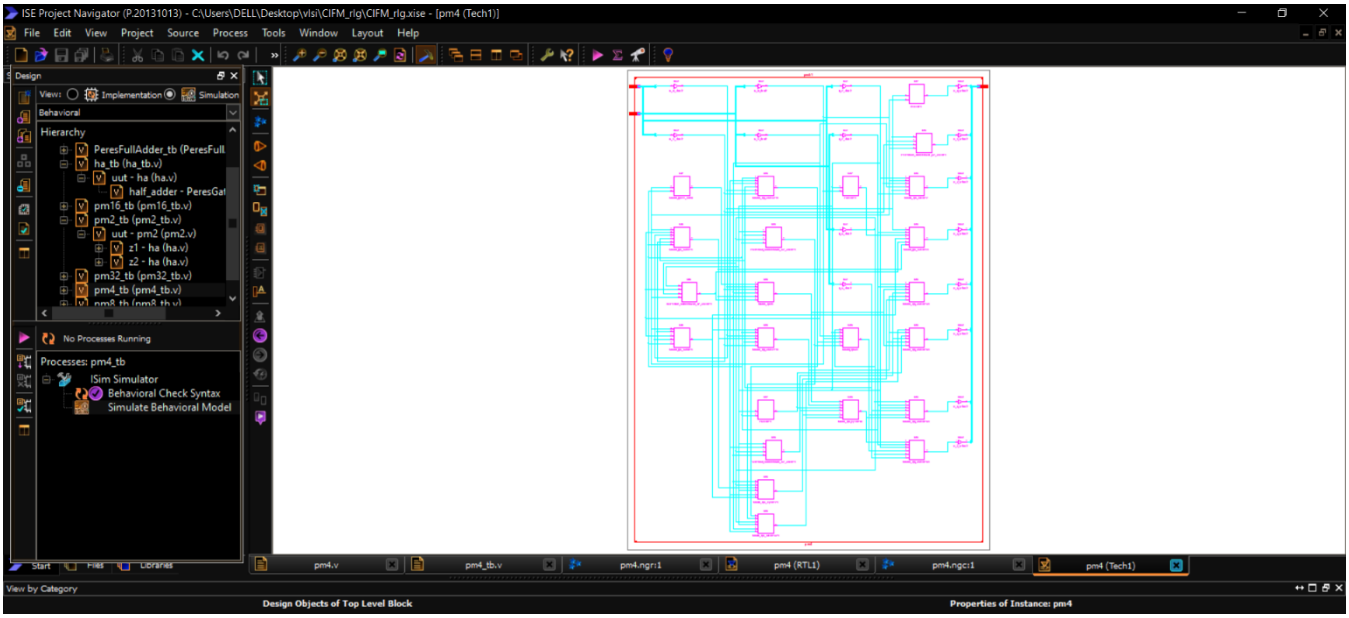
```
`timescale 1ns / 1ps
module pm4_tb;
	// Inputs
	reg [3:0] a;
	reg [3:0] b;
	// Outputs
	wire [7:0] c;
	// Instantiate the Unit Under Test (UUT)
	pm4 uut (
		.a(a),
		.b(b),
		.c(c)
	);
	initial begin
	 a = 4'b0010;b = 4'b0100;#100;
    a = 4'b1010;b = 4'b0101;#100;
	 a = 4'b1110;b = 4'b0111;#100;
	 a = 4'b1111;b = 4'b0011;#100;
	end
endmodule
```

RTL Schemantics:

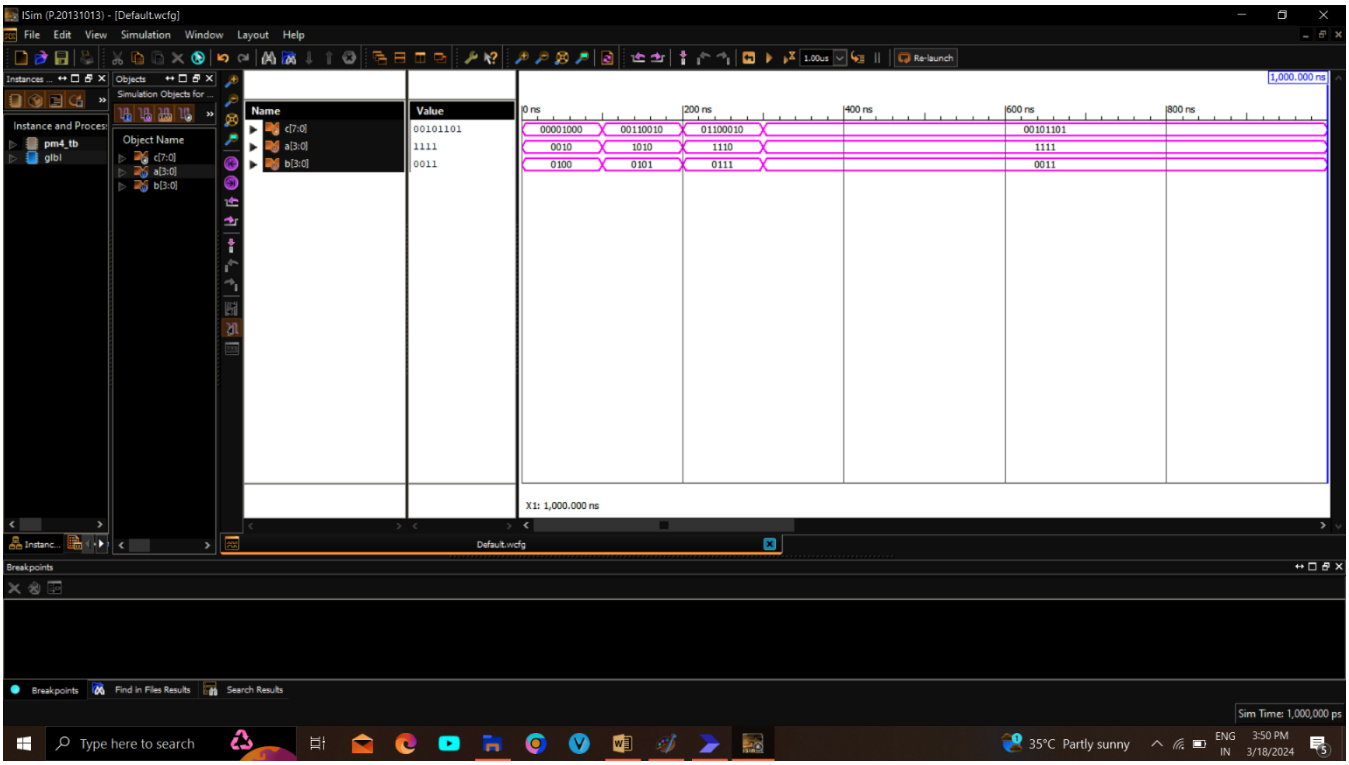




Technology Schematic:



Wave forms:



8 bit Multiplier:

Code:

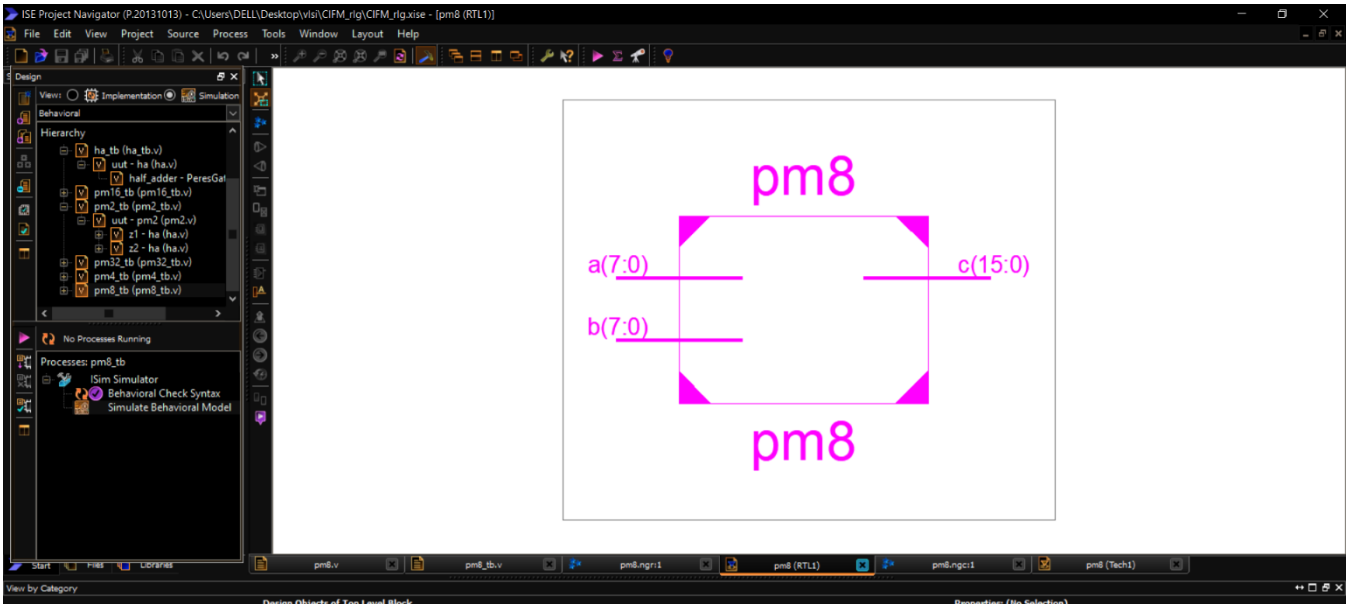
```
module pm8(a,b,c);
    input [7:0]a,b;
    output [15:0]c;
    wire [15:0]q0,q1,q2,q3,c;
    wire [7:0]q4,temp1;
    wire [11:0]q5,q6,temp2,temp3,temp4;
    pm4 z1(a[3:0],b[3:0],q0[15:0]);
    pm4 z2(a[7:4],b[3:0],q1[15:0]);
    pm4 z3(a[3:0],b[7:4],q2[15:0]);
    pm4 z4(a[7:4],b[7:4],q3[15:0]);
    assign temp1 ={4'b0,q0[7:4]};
    assign q4 = q1[7:0]+temp1;
    assign temp2 ={4'b0,q2[7:0]};
    assign temp3 ={q3[7:0],4'b0};
    assign q5 = temp2+temp3;
    assign temp4={4'b0,q4[7:0]};
    assign q6 = temp4+q5;
    assign c[3:0]=q0[3:0];
    assign c[15:4]=q6[11:0];
endmodule
```

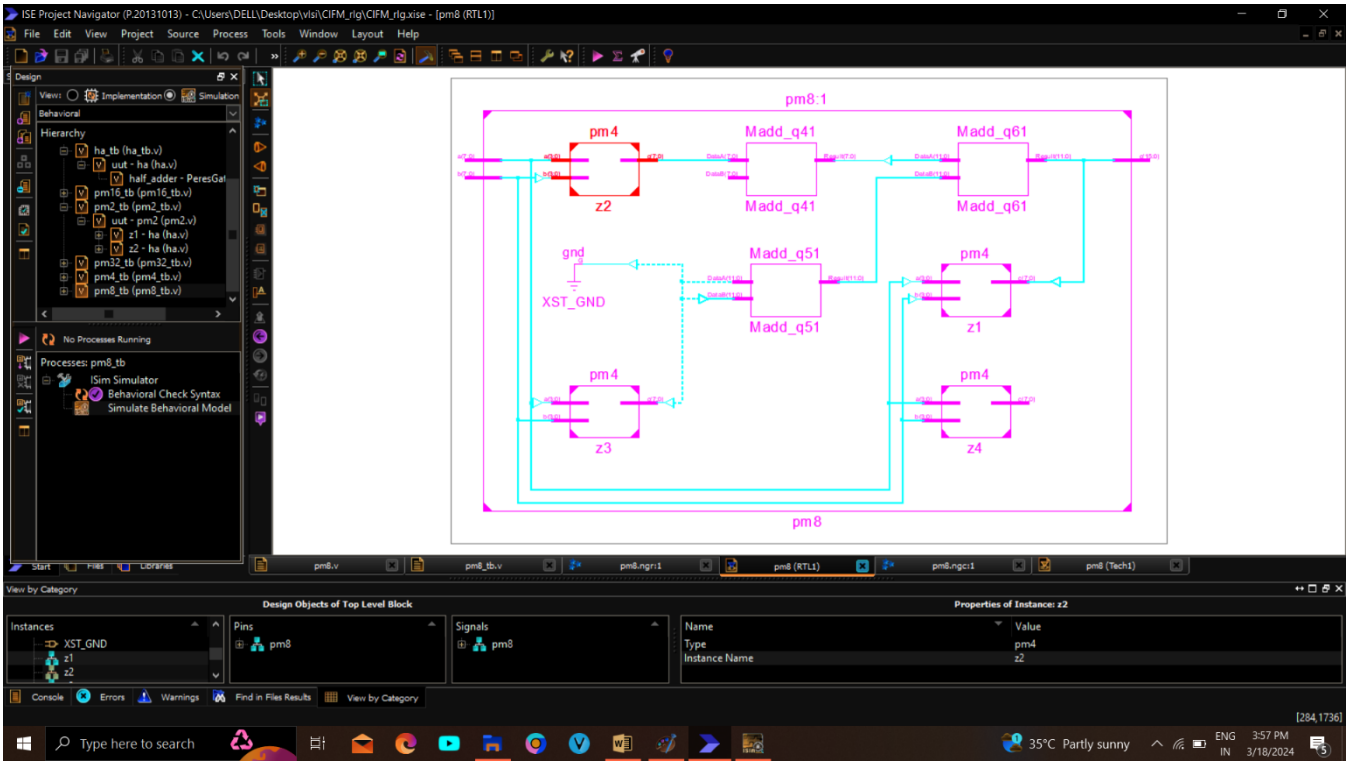
Test code:

```
`timescale 1ns / 1ps
module pm8_tb;
```

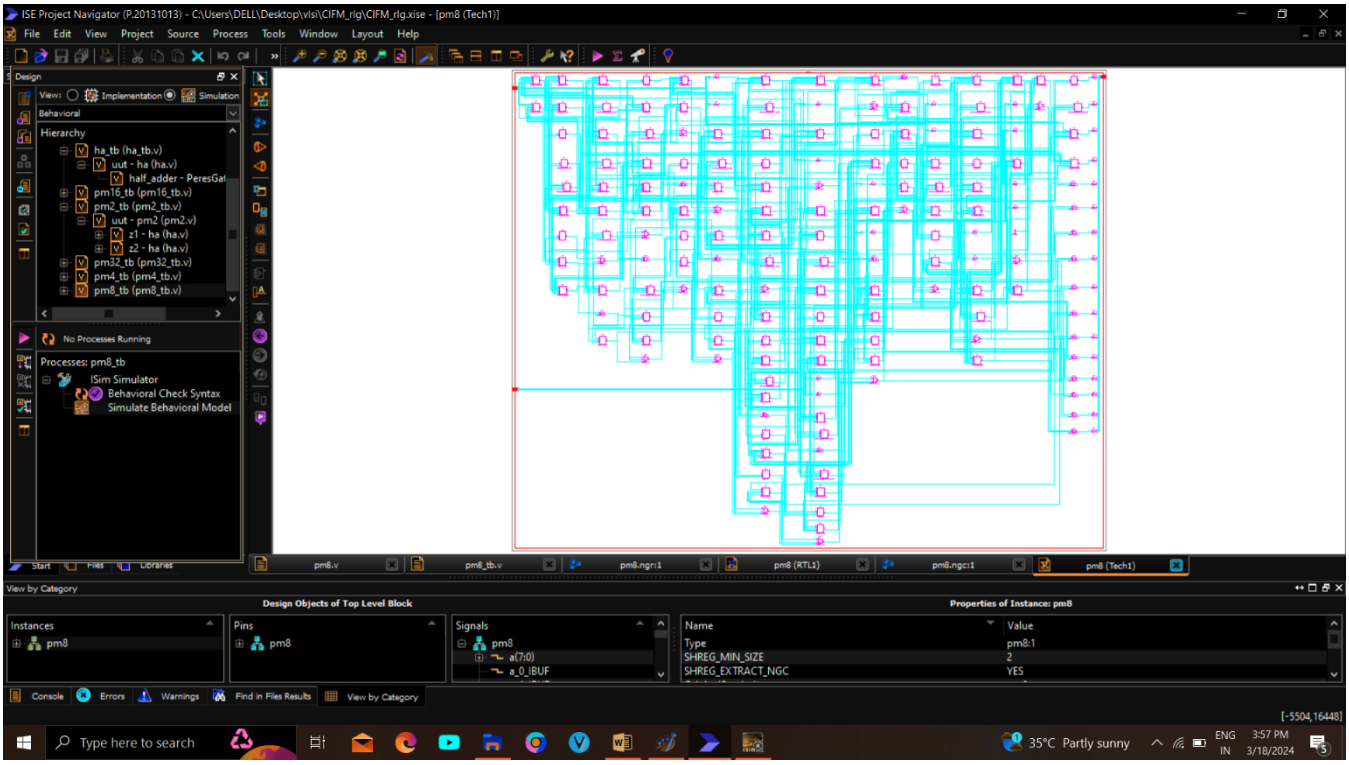
```
        reg [7:0] a;
        reg [7:0] b;
        // Outputs
        wire [15:0] c;
        // Instantiate the Unit Under Test (UUT)
        pm8 uut (.a(a), .b(b), .c(c));
        initial begin
         a = 8'b00000101;b = 8'b00000111;#100;
         a = 8'b11111111;b = 8'b00000001;#100;
         a = 8'b00100101;b = 8'b01100111;#100;
         a = 8'b11111111;b = 8'b10100001;#100;
        end
endmodule
```

RTL Schematics:

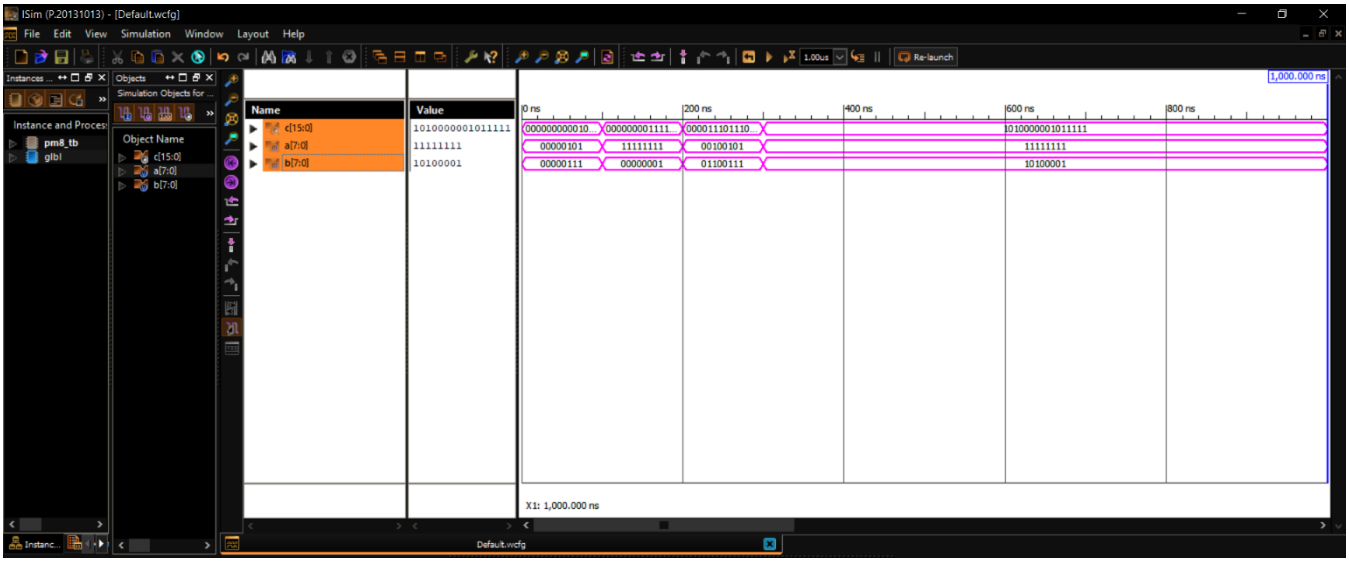




Technology Schematic:



Wave Forms:



16 bit Multiplier:

Code:

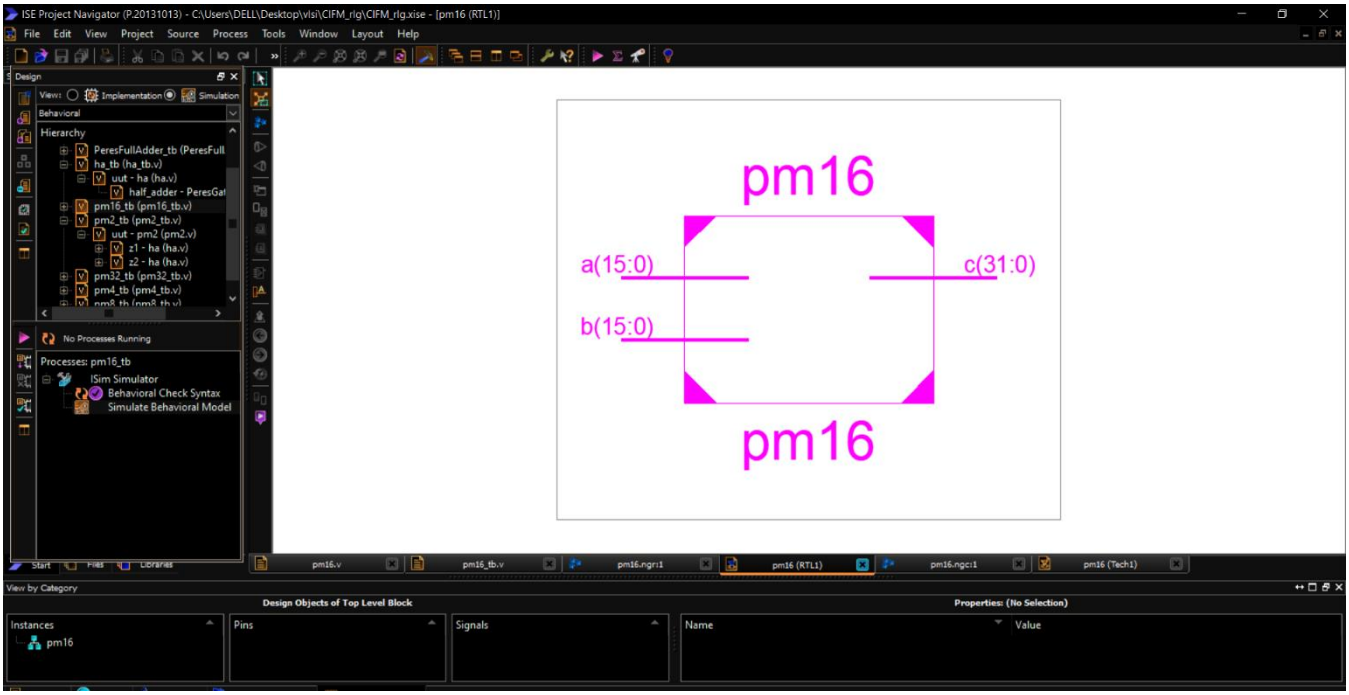
```
module pm16(a,b,c);
    input [15:0]a,b;
   output [31:0]c;
   wire [15:0]q0,q1,q2,q3,q4,temp1;
   wire [31:0]c;
   wire [23:0]q5,q6,temp2,temp3,temp4;
   pm8 z1(a[7:0],b[7:0],q0[15:0]);
   pm8 z2(a[15:8],b[7:0],q1[15:0]);
   pm8 z3(a[7:0],b[15:8],q2[15:0]);
   pm8 z4(a[15:8],b[15:8],q3[15:0]);
   assign temp1 ={8'b0,q0[15:8]};
   assign q4 = q1[15:0]+temp1;
   assign temp2 ={8'b0,q2[15:0]};
```

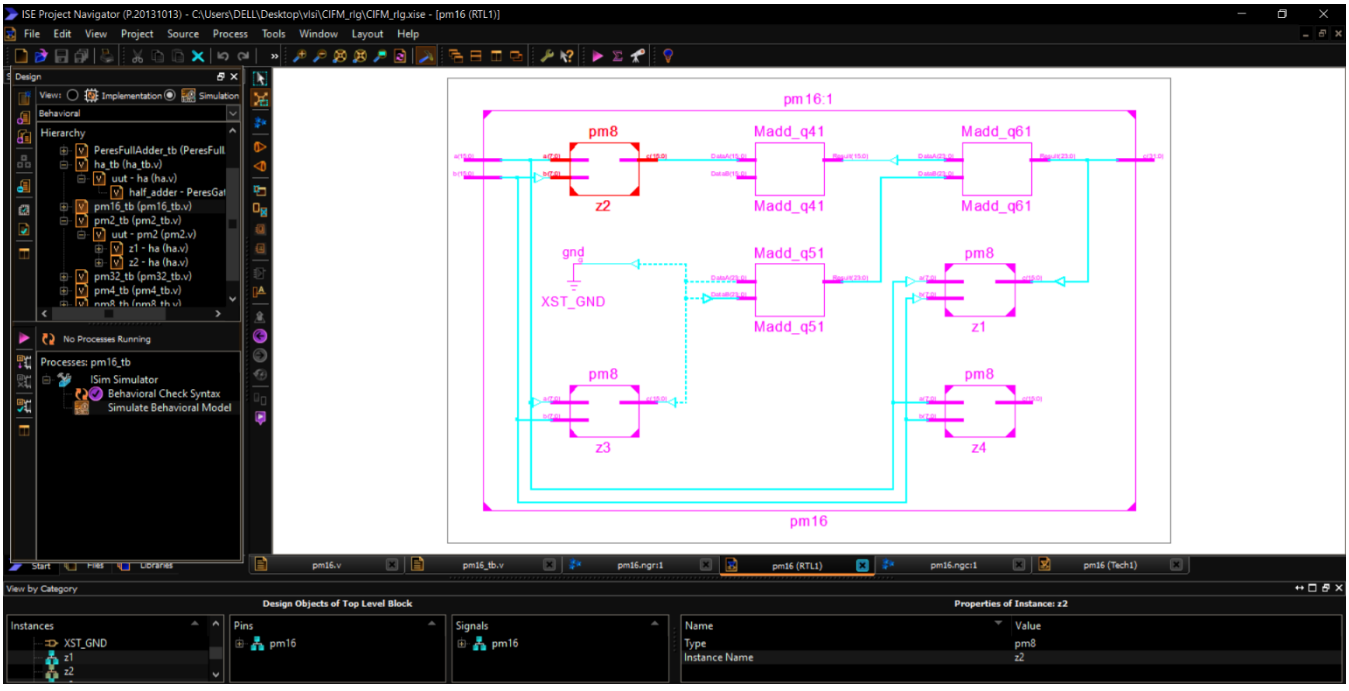
```
    assign temp3 ={q3[15:0],8'b0};
    assign q5 = temp2+temp3;
    assign temp4={8'b0,q4[15:0]};
    assign q6 = temp4 + q5;
    assign c[7:0]=q0[7:0];
    assign c[31:8]=q6[23:0];
endmodule
```

Test code:

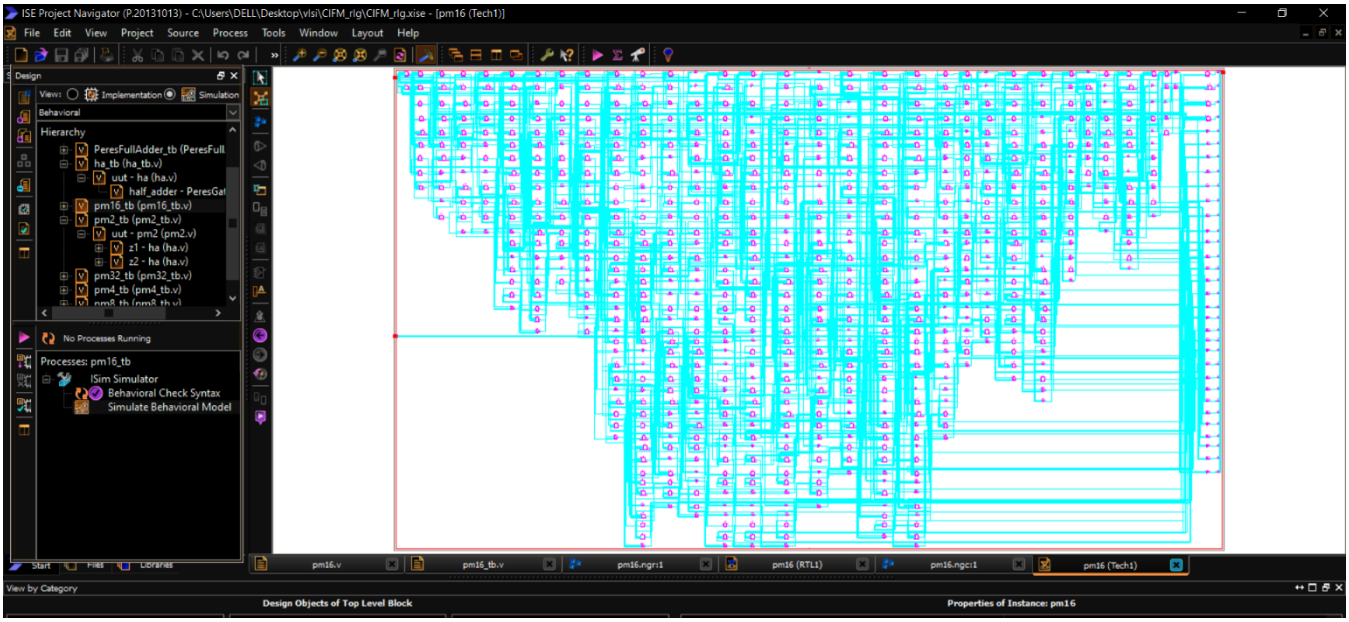
```
`timescale 1ns / 1ps
 module pm16_tb;
reg [15:0] a;reg [15:0] b;
wire [31:0] c;
// Instantiate the Unit Under Test (UUT)
pm16 uut (.a(a), .b(b), .c(c));
initial begin
a = 16'h0004;b = 16'h0006;#100;
a = 16'h004c;b = 16'h0018;#100;
a = 16'b0010110011101100;
b = 16'b1100111011101100;
#100;
a = 16'b1010101010101011;
b = 16'b0010110011001100;
#100;
          end
endmodule
```

RTL Schemantics:

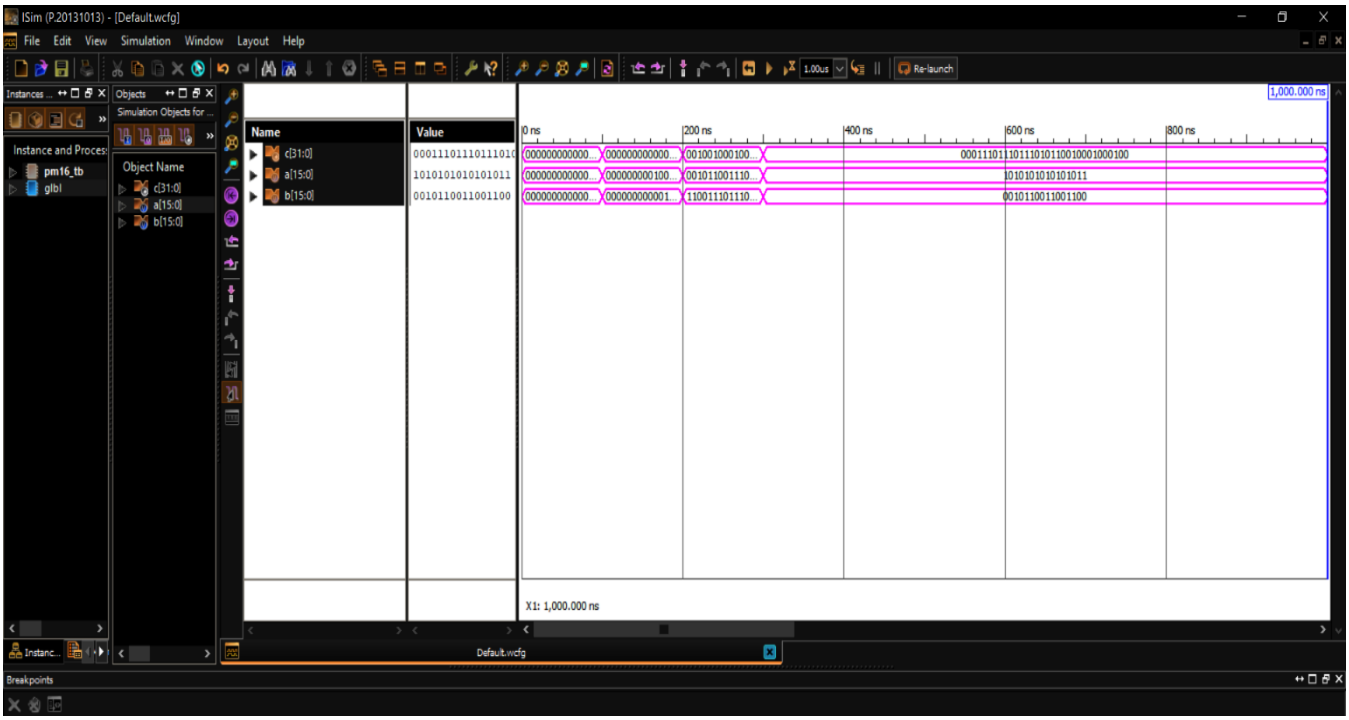




Technology Schematic:



Wave Forms:



32 bit Multiplier:

Code:

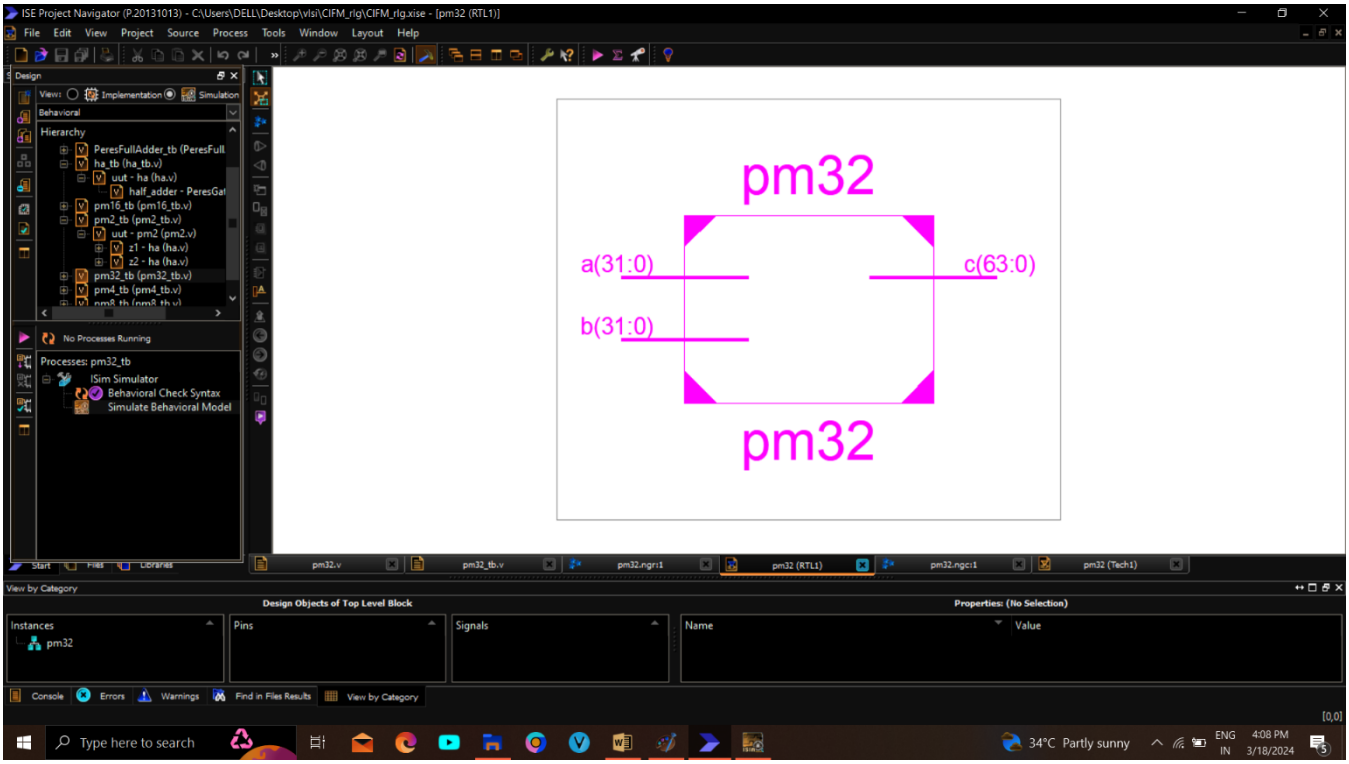
```
module pm32(a,b,c);
    input [31:0]a,b;
    output [63:0]c;
    wire [31:0]q0,q1,q2,q3,q4,temp1;
    wire [63:0]c;
    wire [47:0]q5,q6,temp2,temp3,temp4;
    pm16 z1(a[15:0],b[15:0],q0[31:0]);
    pm16 z2(a[31:16],b[31:16],q1[31:0]);
    pm16 z3(a[15:0],b[31:16],q2[31:0]);
    pm16 z4(a[31:16],b[31:16],q3[31:0]);
    assign temp1 ={16'b0,q0[31:16]};
    assign q4 = q1[31:0]+temp1;
    assign temp2 ={16'b0,q2[31:0]};
    assign temp3 ={q3[31:0],16'b0};
    assign q5 = temp2+temp3;
    assign temp4={16'b0,q4[31:0]};
    assign q6 = temp4 + q5;
    assign c[15:0]=q0[15:0];
    assign c[63:16]=q6[47:0];
 endmodule
```

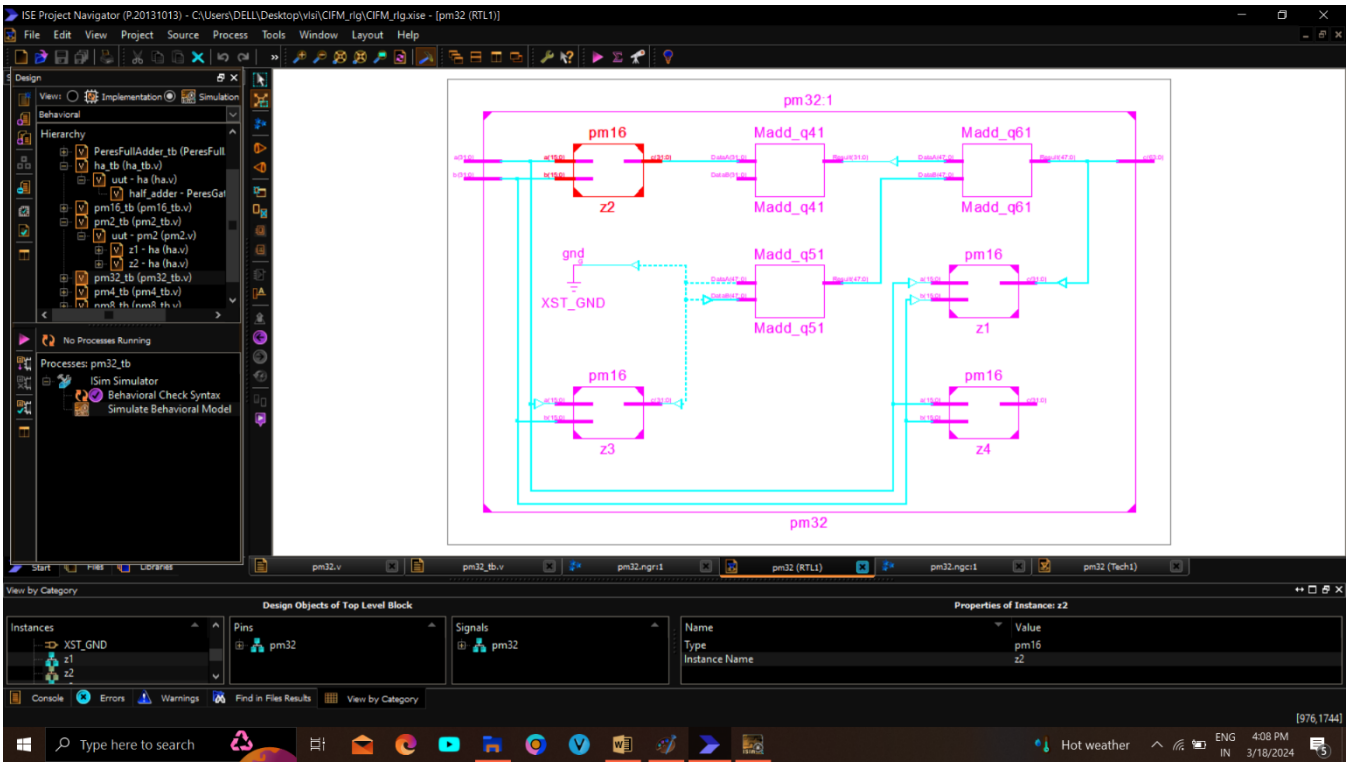
Test code:

```
`timescale 1ns / 1ps
module pm32_tb;
```

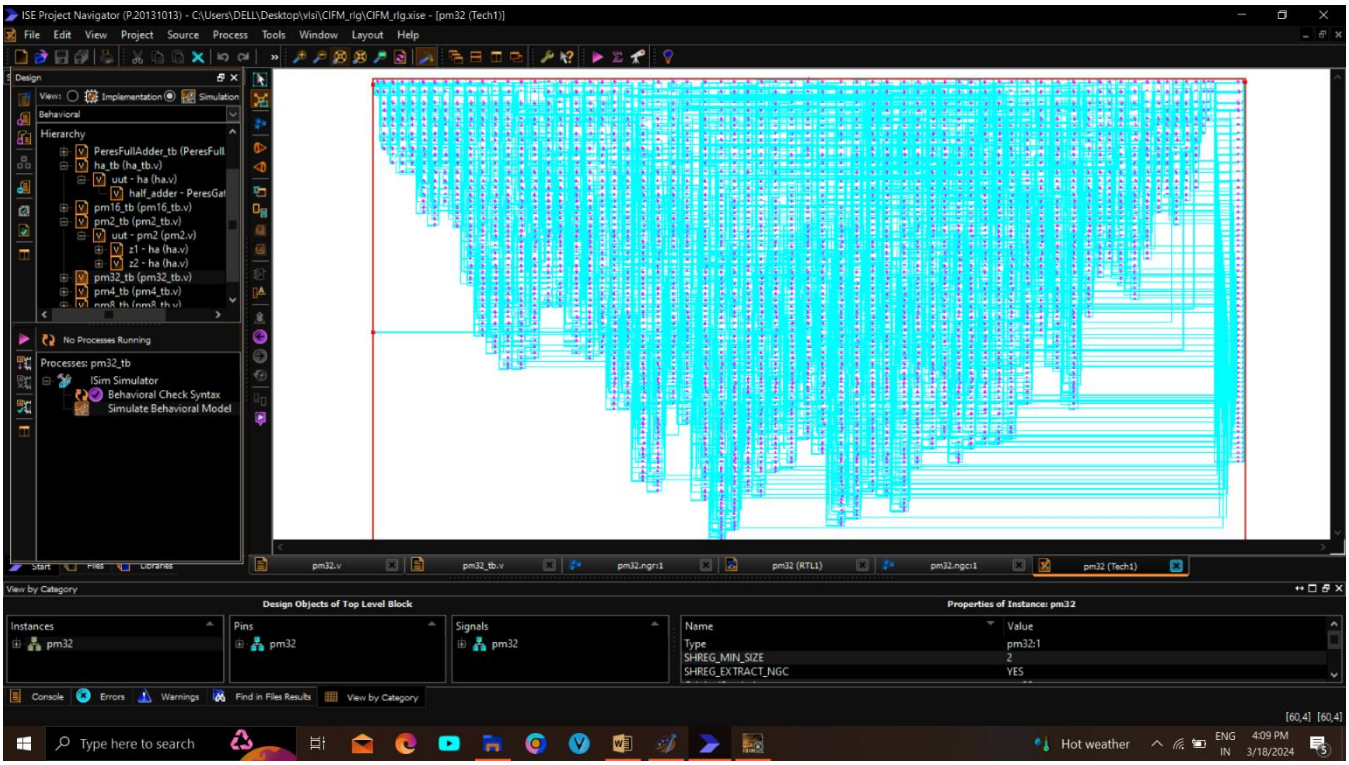
```
        reg [31:0] a;
        reg [31:0] b;
        wire [63:0] c;
        // Instantiate the Unit Under Test (UUT)
        pm32 uut (.a(a),
                .b(b),
                .c(c));
        initial begin
    a = 32'h00000001; // 1
    b = 32'h00000002; // 2
    #100;
    a = 32'h00000064; // 100
    b = 32'h000000c8; // 200
    #100;
    a = 32'h0000ffff; // 65535
    b = 32'h0000ffff; // 65535
    #100;
    a = 32'b11010101010110010110010011001010; // 12345678
    b = 32'b01010011011001000111010100110011; // 87654321
    #100;
         a = 32'hffffffff; // 65535
    b = 32'hffffffff; // 65535
    #100;
        end
endmodule
```

RTL Schematics:

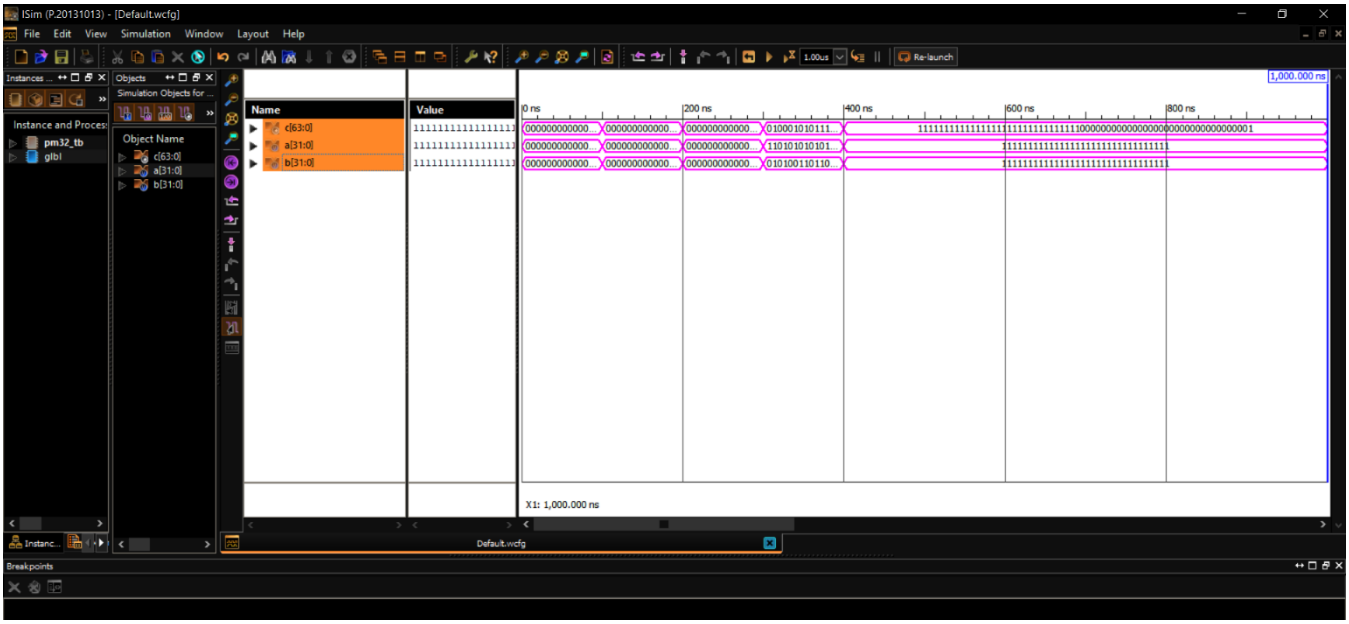




Technology Schematic:



Wave forms:



Floating Point Multiplier

1.Sign bit identification:

Code:

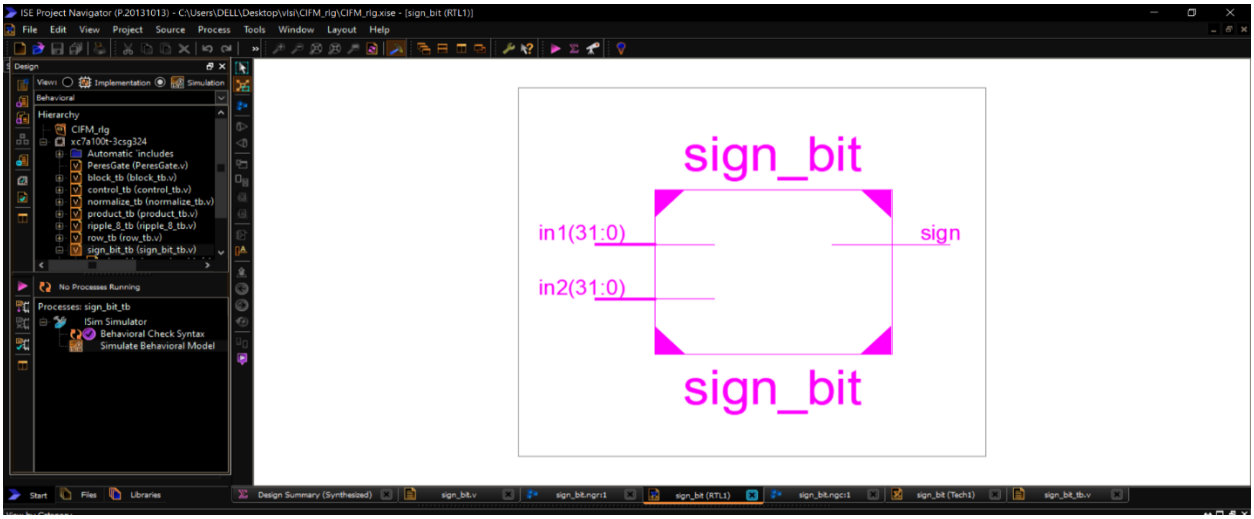
```
`include "Feynmangate.v"
module sign_bit (output wire sign,input wire [31:0] in1, input wire [31:0] in2);
wire [31:0] xor_result;
  genvar i;
  generate
    for (i = 0; i < 32; i = i + 1) begin
      Feynmangate feynman_gate_inst .a(in1[i]), .b(in2[i]), .o1(), .o2(xor_result[i]) );
    end
  endgenerate
  assign sign = xor_result[31];
endmodule
```

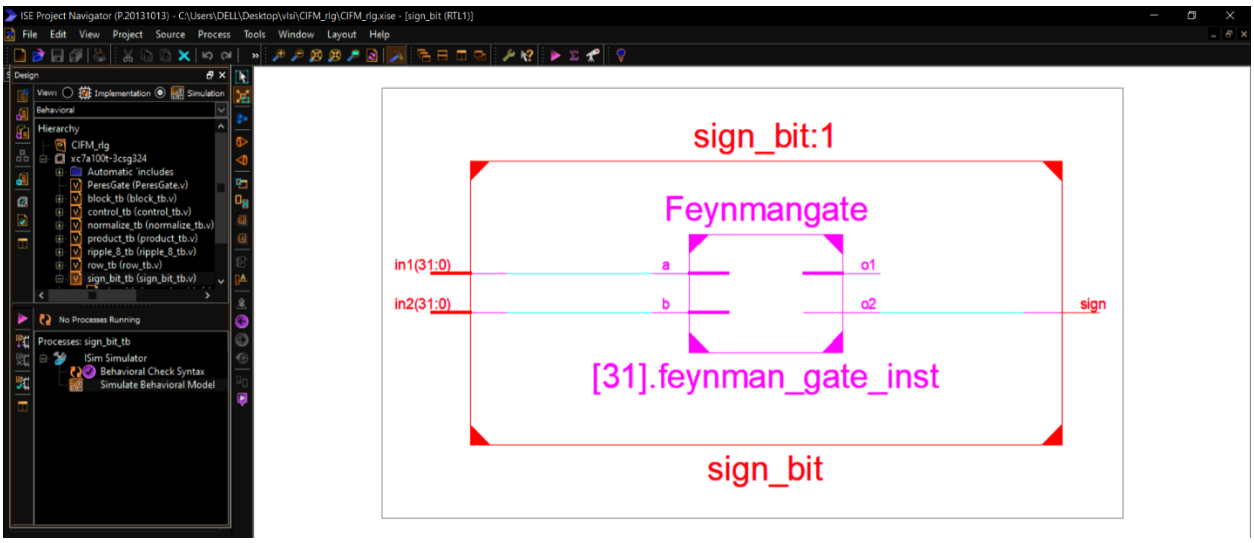
Test code:

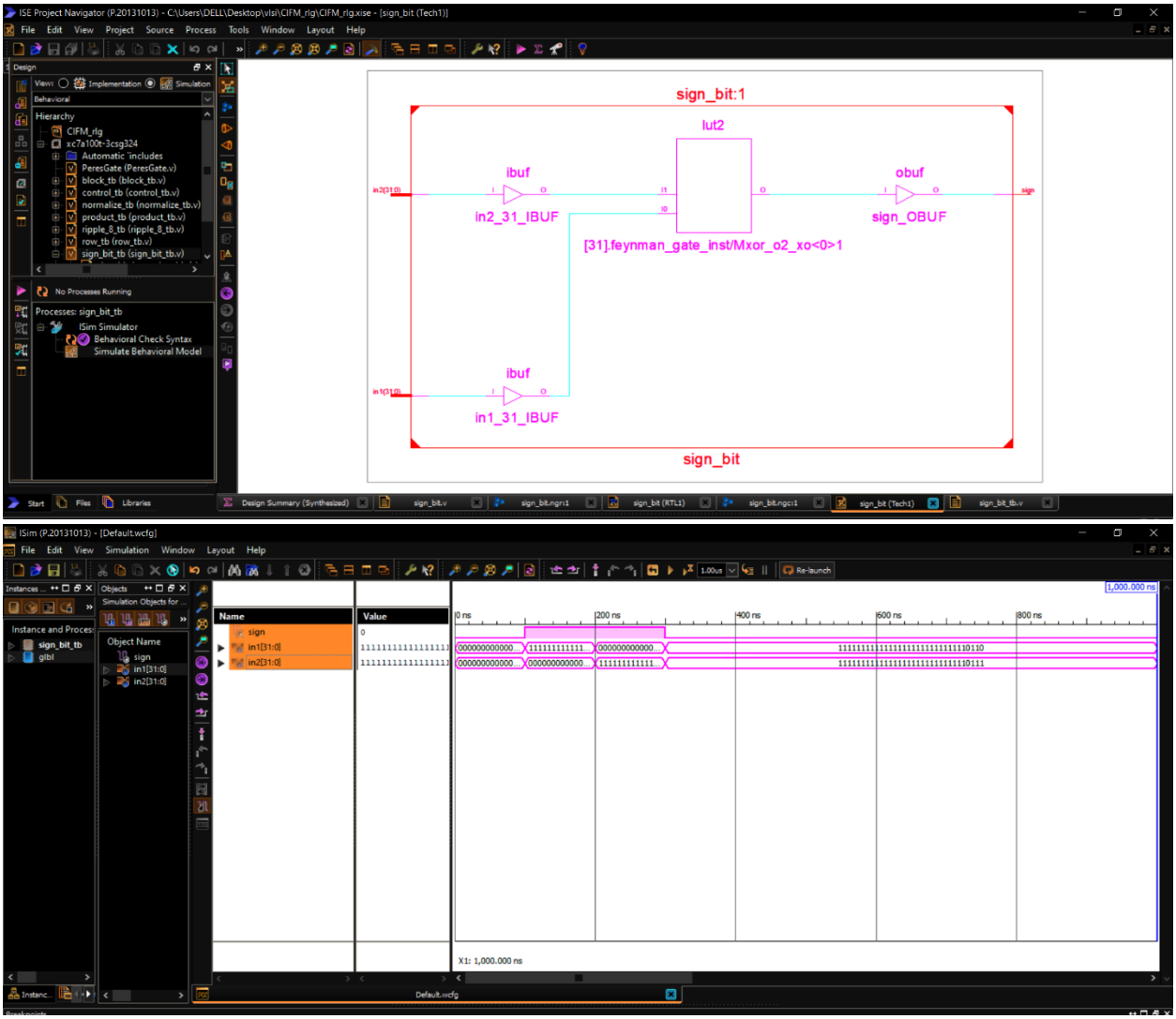
```
`timescale 1ns / 1ps
module sign_bit_tb;
  reg [31:0] in1;
  reg [31:0] in2;
  wire sign;
  sign_bit sign_bit_inst (.sign(sign),
    .in1(in1),.in2(in2));
initial begin
    in1 = 32'h00000001;
    in2 = 32'h00000002;
```

```
    #100;
    in1 = 32'hFFFFFFF0;
    in2 = 32'h00000003;#100;
    in1 = 32'h00000004;
    in2 = 32'hFFFFFFF5;#100;
    in1 = 32'hFFFFFFF6;
    in2 = 32'hFFFFFFF7;#100;
  end
endmodule
```

Schematics:







2.Adding Exponents:

Full adder using half adder:

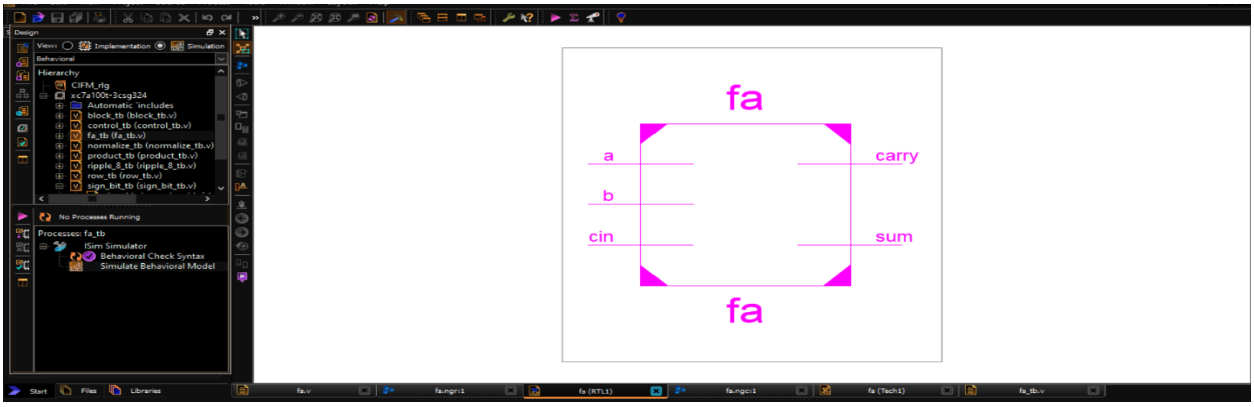
```
`include "ha.v"

module fa (input a,input b,input cin,

output sum,output carry);

wire h1_sum, h1_carry, h2_carry;

ha half_adder1 (.a(a),.b(b),

 .sum(h1_sum),.carry(h1_carry));

ha half_adder2 (.a(h1_sum),.b(cin),
```

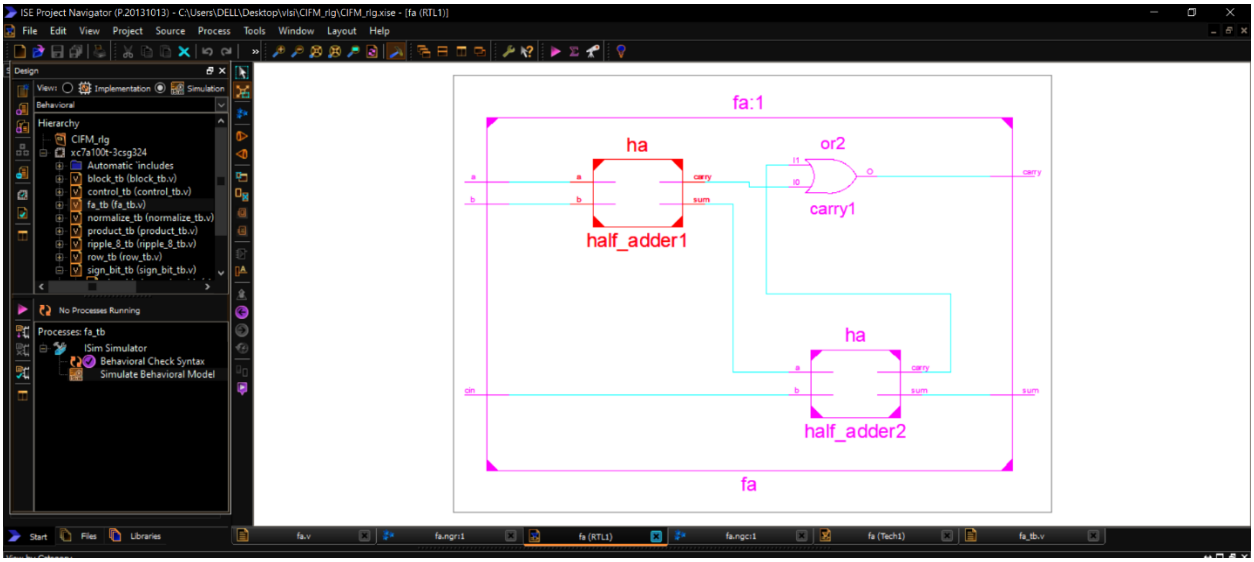
```
 .sum(sum),.carry(h2_carry));
assign carry = h1_carry | h2_carry;
endmodule
```

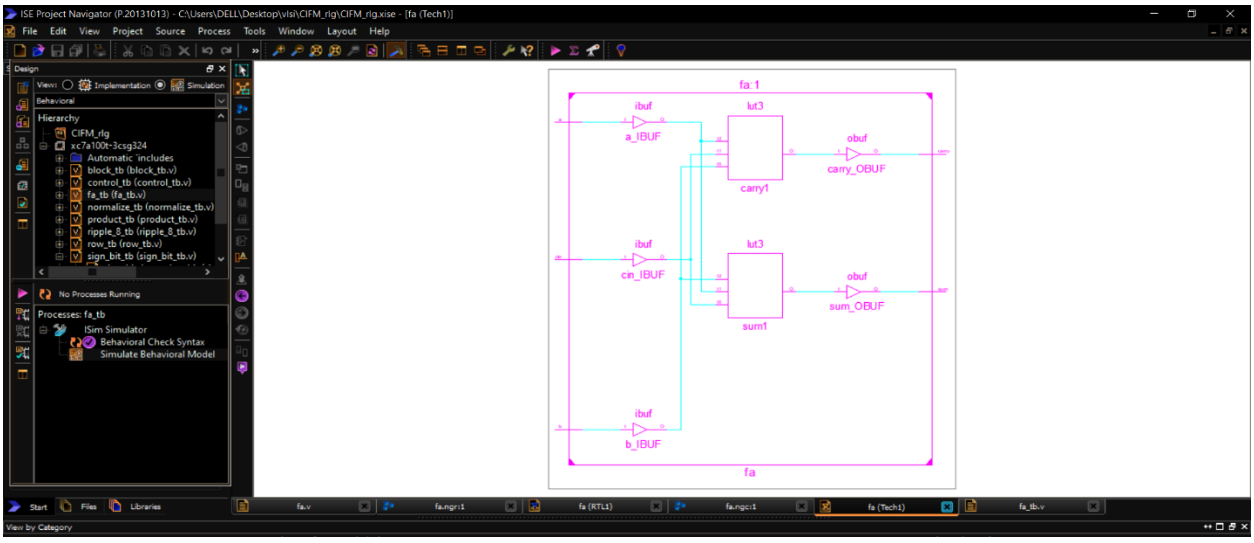
Test code:

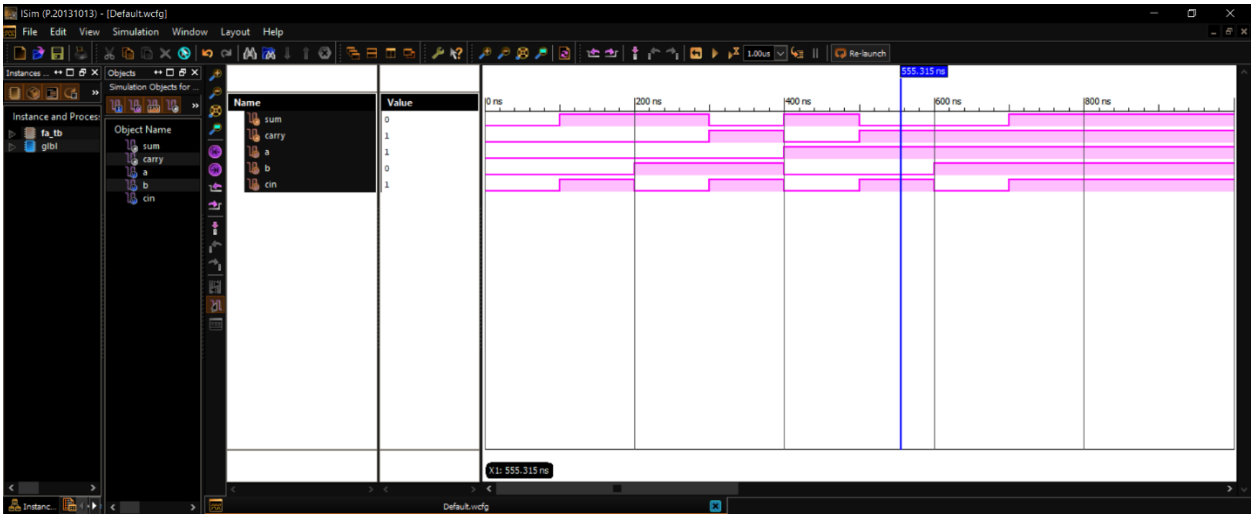
```
`timescale 1ns / 1ps
module fa_tb;
        reg a; reg b;
        reg cin;
        wire sum;
        wire carry;
fa uut (.a(a), .b(b), .cin(cin), .sum(sum), .carry(carry));
initial begin
          a = 0; b = 0; cin = 0; #100; a = 0; b = 0; cin = 1; #100;
          a = 0; b = 1; cin = 0; #100; a = 0; b = 1; cin = 1; #100;
          a = 1; b = 0; cin = 0; #100;a = 1; b = 0; cin = 1; #100;
          a = 1; b = 1; cin = 0; #100; a = 1; b = 1; cin = 1; #100;
        end
endmodule
```

Schematics:




fa:1
ha
or2
half_adder1
carry1
ha
half_adder2
fa


fa:1
ibuf
a_IBUF
lut3
carry1
obuf
carry_OBUF
ibuf
cin_IBUF
lut3
sum1
obuf
sum_OBUF
ibuf
b_IBUF
fa


Name
Value
sum
carry
a
b
cin
555.315 ns
200 ns
400 ns
600 ns
800 ns

8bit Ripple Carry Adder:

Code:

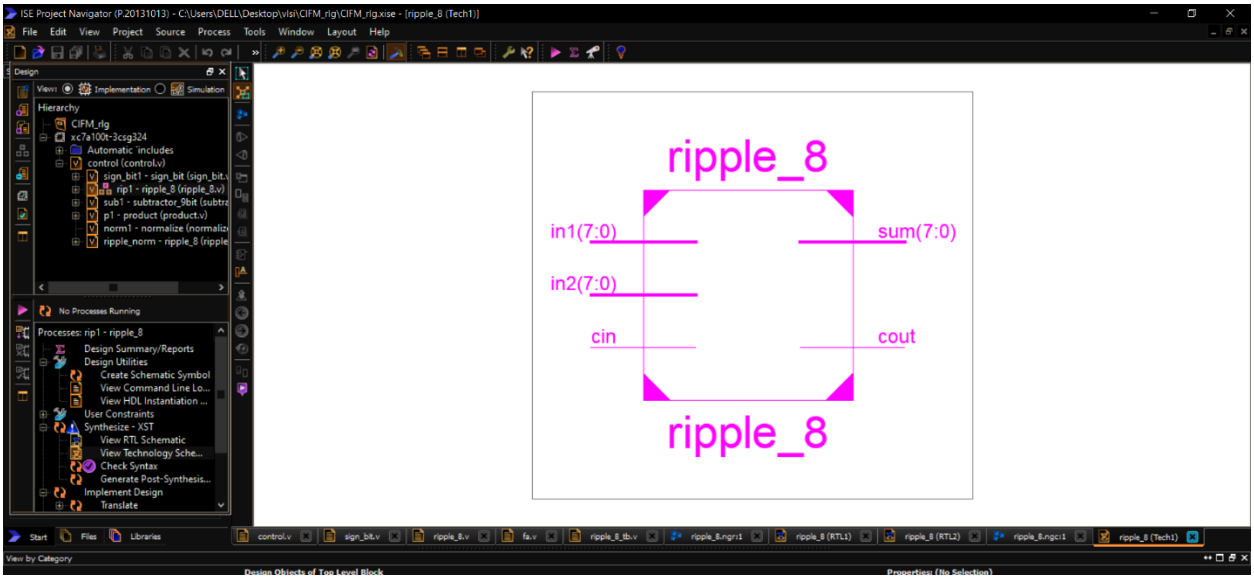
```
module ripple_8(output wire [7:0] sum,output wire cout,
  input wire [7:0] in1, input wire [7:0] in2,
  input wire cin);
wire c1, c2, c3, c4, c5, c6, c7;
  fa FA1( .a(in1[0]),.b( in2[0]), .cin(cin),.sum(sum[0]),.carry( c1));
  fa FA2( .a(in1[1]),.b( in2[1]), .cin(c1),.sum(sum[1]),.carry( c2));
  fa FA3( .a(in1[2]),.b( in2[2]), .cin(c2),.sum(sum[2]),.carry( c3));
  fa FA4( .a(in1[3]),.b( in2[3]), .cin(c3),.sum(sum[3]),.carry( c4));
  fa FA5( .a(in1[4]),.b( in2[4]), .cin(c4),.sum(sum[4]),.carry( c5));
  fa FA6( .a(in1[5]),.b( in2[5]), .cin(c5),.sum(sum[5]),.carry( c6));
  fa FA7( .a(in1[6]),.b( in2[6]), .cin(c6),.sum(sum[6]),.carry( c7));
  fa FA8( .a(in1[7]),.b( in2[7]), .cin(c7),.sum(sum[7]),.carry( cout));
endmodule
```

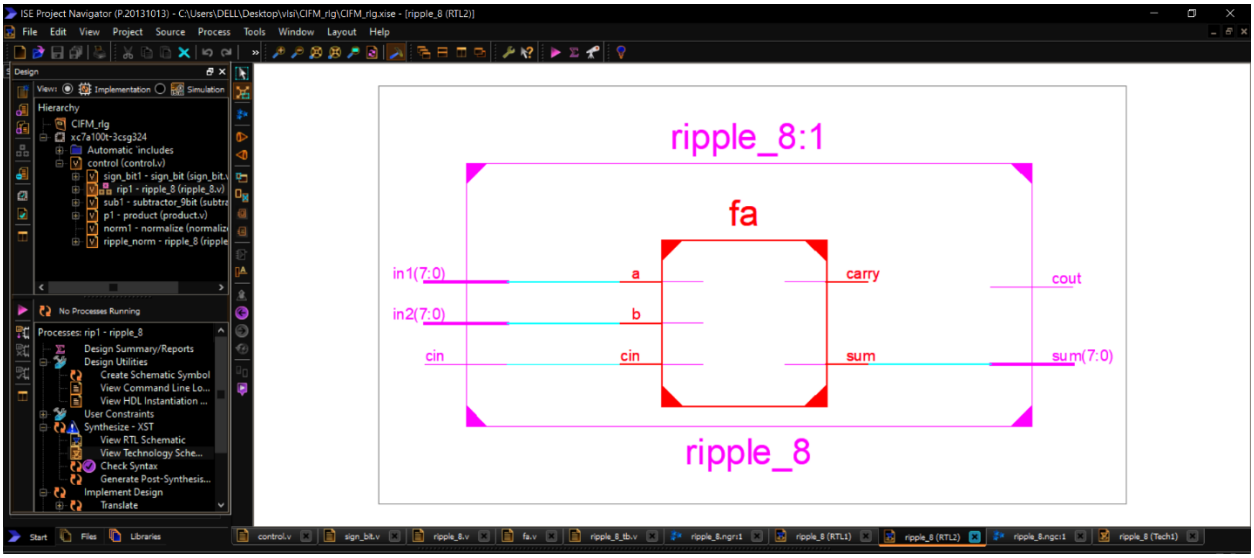
test code:

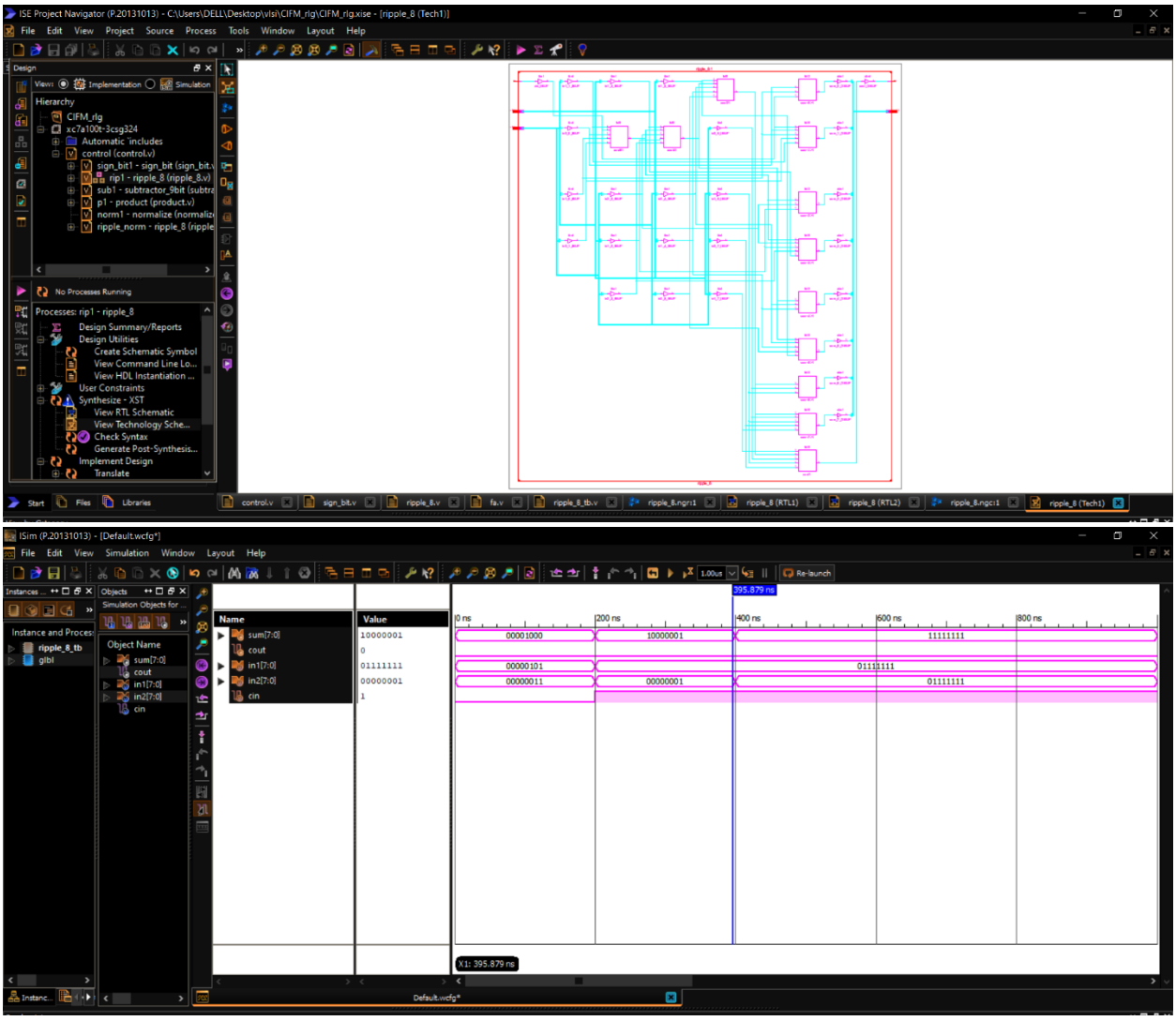
```
module ripple_8_tb;
reg [7:0] in1;
reg [7:0] in2; reg cin;
wire [7:0] sum;
wire cout;
// Instantiate the ripple_8 module
ripple_8 ripple_adder (.sum(sum),.cout(cout),
    .in1(in1),.in2(in2),.cin(cin));
initial begin
```

```
    in1 = 8'b00000101; in2 = 8'b00000011;cin = 1'b0;#200;
    in1 = 8'b01111111; in2 = 8'b00000001;cin = 1'b1; #200;
    in1 = 8'b01111111; in2 = 8'b01111111;cin = 1'b1; #200;
  end
endmodule
```

Schematics:







3.Subtracting the Bias:

Code:

```
module fullsub1(output wire diff,output wire bout,
input wire min,input wire bin);
assign diff= min ~^ bin;
assign bout=~min|bin;
endmodule
module fullsub0(output wire diff,output wire bout,
input wire min,input wire bin);
```

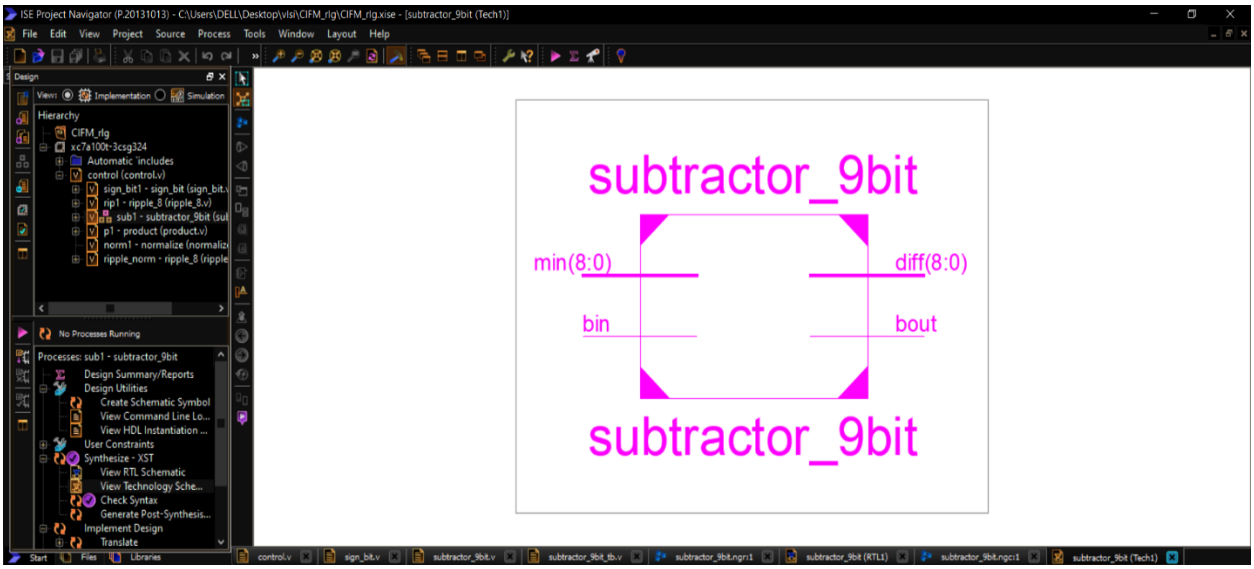
```
assign diff= min^bin;
assign bout= ~min & bin;
endmodule
module subtractor_9bit(
    output wire [8:0] diff,
    output wire bout,
    input wire [8:0] min,
    input wire bin);
    wire [7:0] b;
    fullsub1 sub1(diff[0], b[0], min[0], bin);
    fullsub1 sub2(diff[1], b[1], min[1], b[0]);
    fullsub1 sub3(diff[2], b[2], min[2], b[1]);
    fullsub1 sub4(diff[3], b[3], min[3], b[2]);
    fullsub1 sub5(diff[4], b[4], min[4], b[3]);
    fullsub1 sub6(diff[5], b[5], min[5], b[4]);
    fullsub1 sub7(diff[6], b[6], min[6], b[5]);
    fullsub0 sub8(diff[7], b[7], min[7], b[6]);
    fullsub0 sub9(diff[8], bout, min[8], b[7]);
endmodule
```

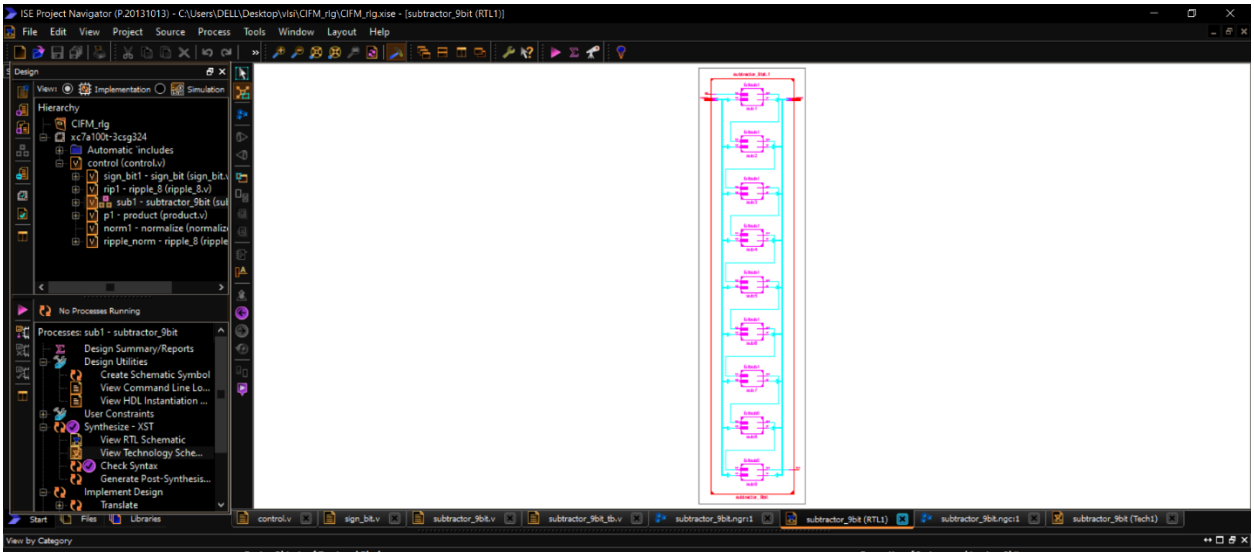
Test Code:

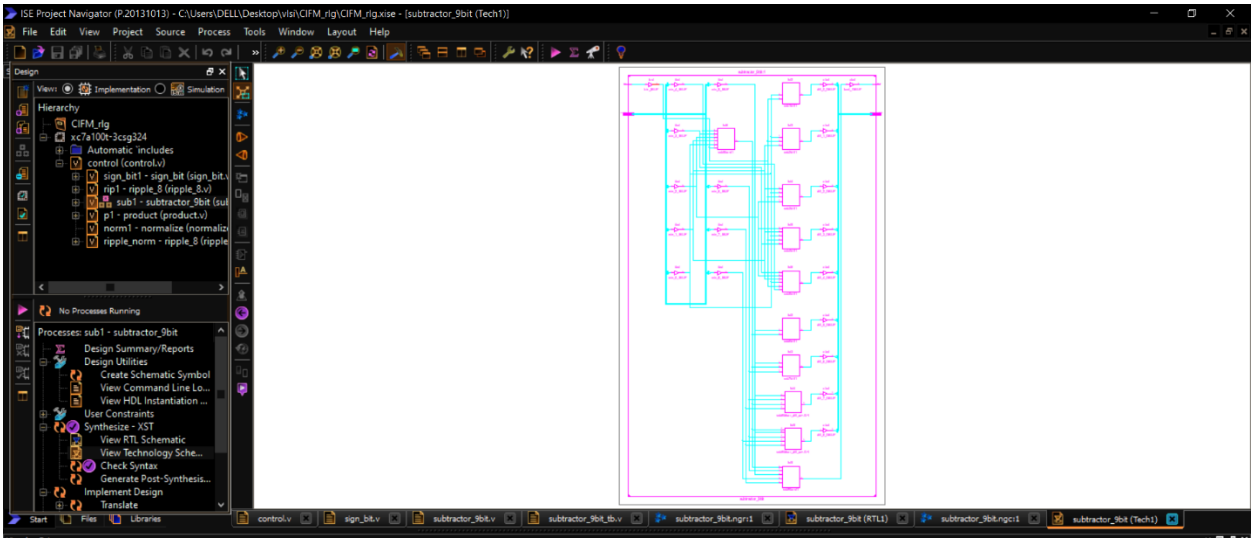
```
module subtractor_9bit_tb;
    reg [8:0] min; reg bin;
    wire [8:0] diff;
    wire bout;
    // Instantiate the subtractor_9bit module
```

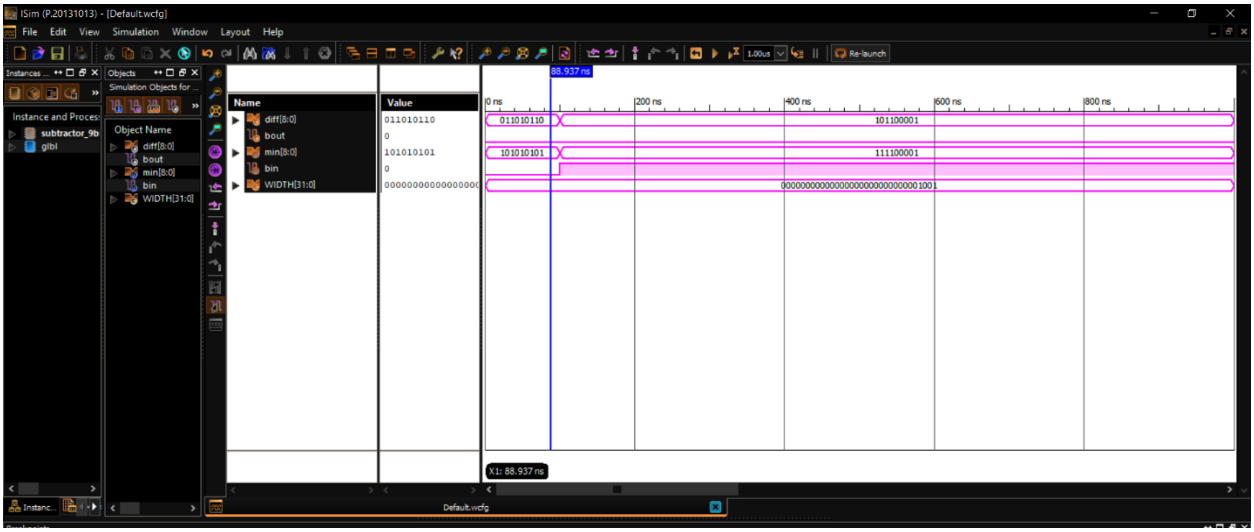
```
subtractor_9bit uut (.diff(diff),.bout(bout),
        .min(min),.bin(bin));
initial begin
min = 9'b101010101; bin = 1'b0;#100;
min = 9'b111100001; bin = 1'b1;#100;
end
endmodule
```

Schematics:









4.Multiplying the Mantissa:

Code: (Block)

```
`include "fa.v"

module block(

output wire ppo, //output partial product term

output wire cout, //output carry out

output wire mout, //output multiplicand term

input wire min, //input multiplicand term

input wire ppi, //input partial product term
```

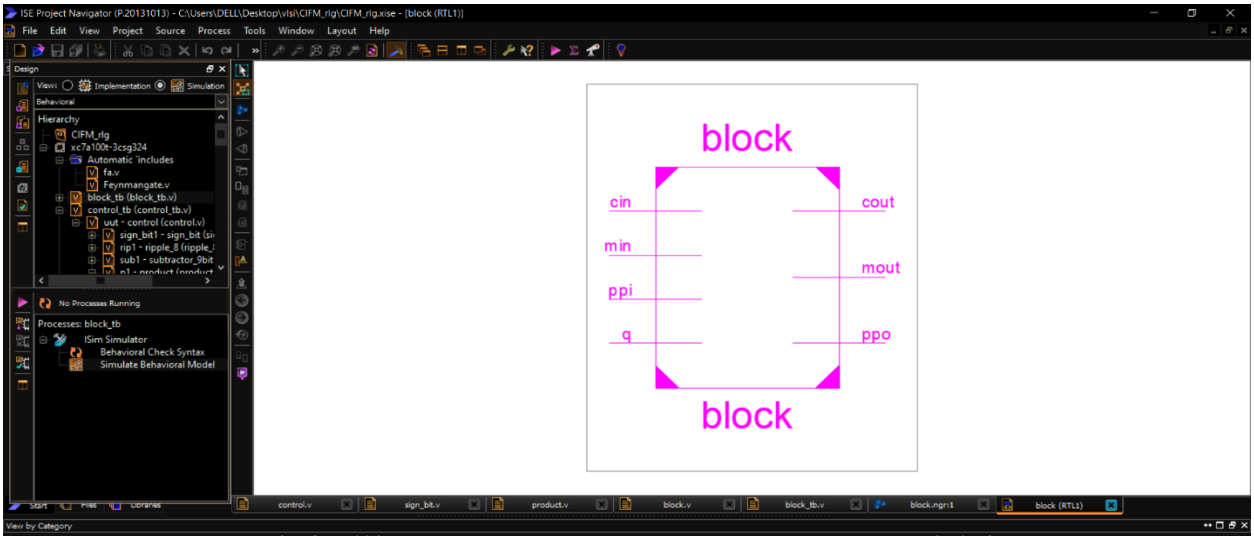
```
input wire q, //input multiplier term
input wire cin );//input carry in
wire temp;
and(temp,min,q);
fa FA( .a(ppi),.b( temp),.cin(cin),.sum(ppo),.carry(cout));
or (mout, min, 1'b0);
endmodule
```

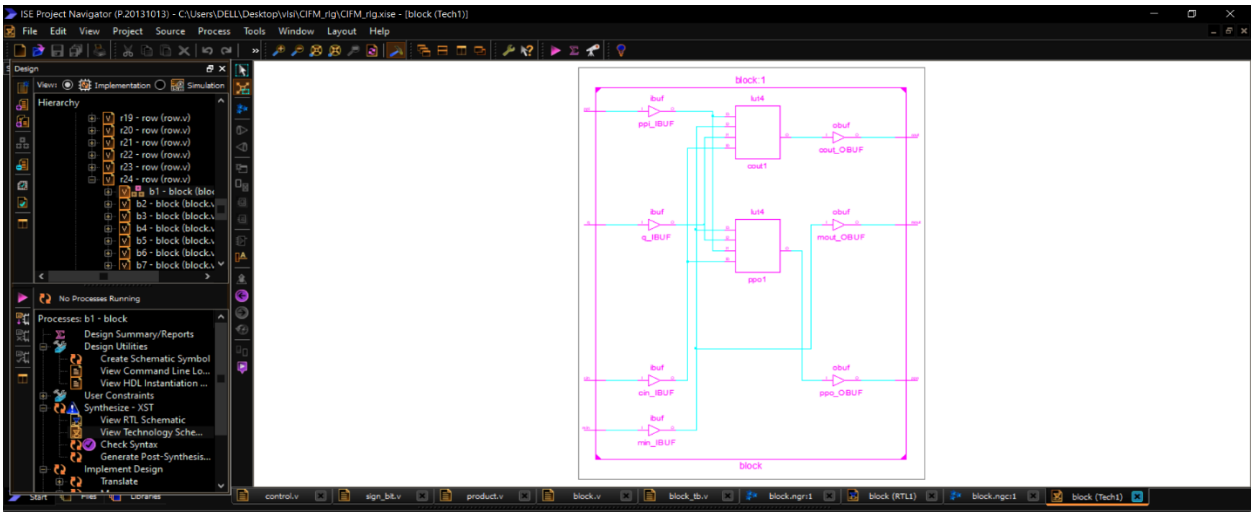
Test code:

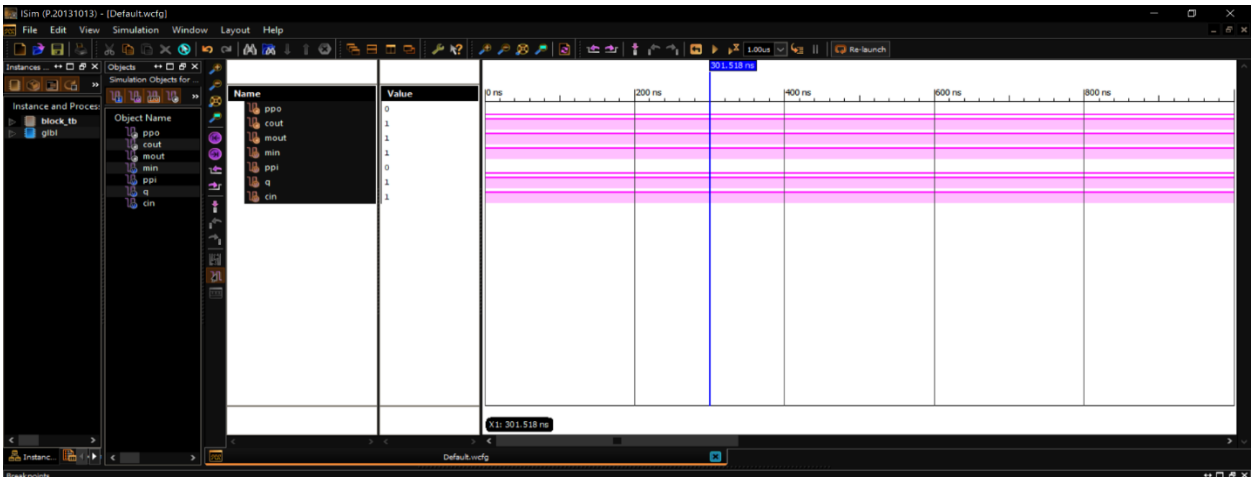
```
`timescale 1ns / 1ps
module block_tb;
        // Inputs
        reg min;
        reg ppi;
        reg q;
        reg cin;
        // Outputs
        wire ppo;
        wire cout;
        wire mout;
        // Instantiate the Unit Under Test (UUT)
        block uut (.ppo(ppo), .cout(cout), .mout(mout),
        .min(min), .ppi(ppi), .q(q), .cin(cin));
        initial begin
        min = 1;ppi = 0;q = 1;cin = 1;#100;
        end
```

endmodule

Schematics:







Code:(Row level)

```
module row(output wire [23:0] ppo, output wire [23:0] mout,
output wire sum, input wire [23:0] min,input wire [23:0] ppi,
input wire q );
wire c1,c2,c3,c4,c5,c6,c7, c8, c9, c10;
wire c11,c12,c13,c14,c15,c16,c17,c18,c19,c20;
wire c21, c22, c23;
block b1 (sum,c1,mout [0],min[0],ppi [0], q,1'b0);
block b2 (ppo [0], c2, mout [1], min [1], ppi [1], q, c1);
block b3 (ppo [1], c3, mout [2], min [2], ppi [2], q, c2);
block b4 (ppo [2], c4, mout [3], min [3], ppi [3], q, c3);
block b5 (ppo [3], c5, mout [4], min [4], ppi [4], q, c4);
block b6 (ppo [4], c6, mout [5], min [5], ppi [5], q, c5);
block b7 (ppo [5], c7, mout [6], min [6], ppi [6], q, c6);
block b8 (ppo [6], c8, mout [7], min [7], ppi [7], q, c7);
block b9 (ppo [7], c9, mout [8], min [8], ppi [8], q, c8);
block b10 (ppo [8], c10, mout [9], min [9], ppi [9], q, c9);
block b11(ppo [9], c11, mout [10], min [10], ppi [10], q, c10);
block b12(ppo [10], c12, mout [11], min [11], ppi [11], q, c11);
block b13(ppo [11], c13, mout [12], min [12], ppi [12], q, c12);
block b14(ppo [12], c14, mout [13], min [13], ppi [13], q, c13);
block b15(ppo [13], c15, mout [14], min [14], ppi [14], q, c14);
block b16(ppo [14], c16, mout [15], min [15], ppi [15], q, c15);
block b17(ppo[15], c17, mout [16], min [16], ppi [16], q, c16);
block b18(ppo[16], c18, mout [17], min [17], ppi [17], q, c17);
```

block b19(ppo[17], c19, mout [18], min [18], ppi [18], q, c18);

block b20(ppo [18], c20, mout [19], min [19], ppi [19], q, c19);

block b21(ppo [19], c21, mout [20], min [20], ppi [20], q, c20);

block b22(ppo [20], c22, mout [21], min [21], ppi [21], q, c21);

block b23(ppo [21], c23, mout [22], min [22], ppi [22], q, c22);

block b24(ppo [22], ppo [23], mout [23], min [23], ppi [23], q, c23);

endmodule

Test code:

`timescale 1ns / 1ps

module row_tb;

reg [23:0] min; reg [23:0] ppi; reg q;

wire [23:0] ppo; wire [23:0] mout; wire sum;

row uut (.ppo(ppo), .mout(mout), .sum(sum), .min(min), .ppi(ppi), .q(q));

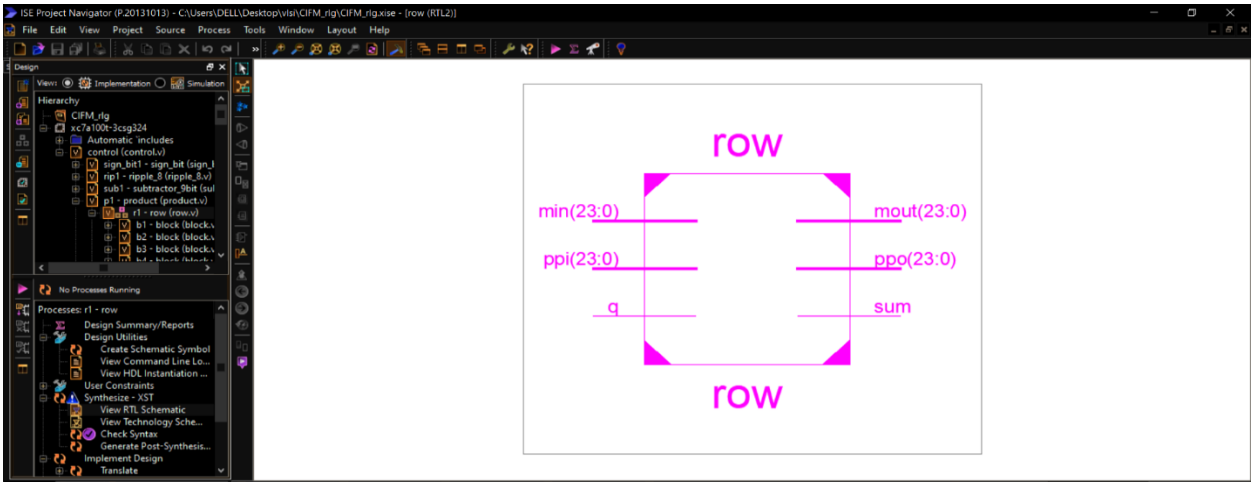
initial begin

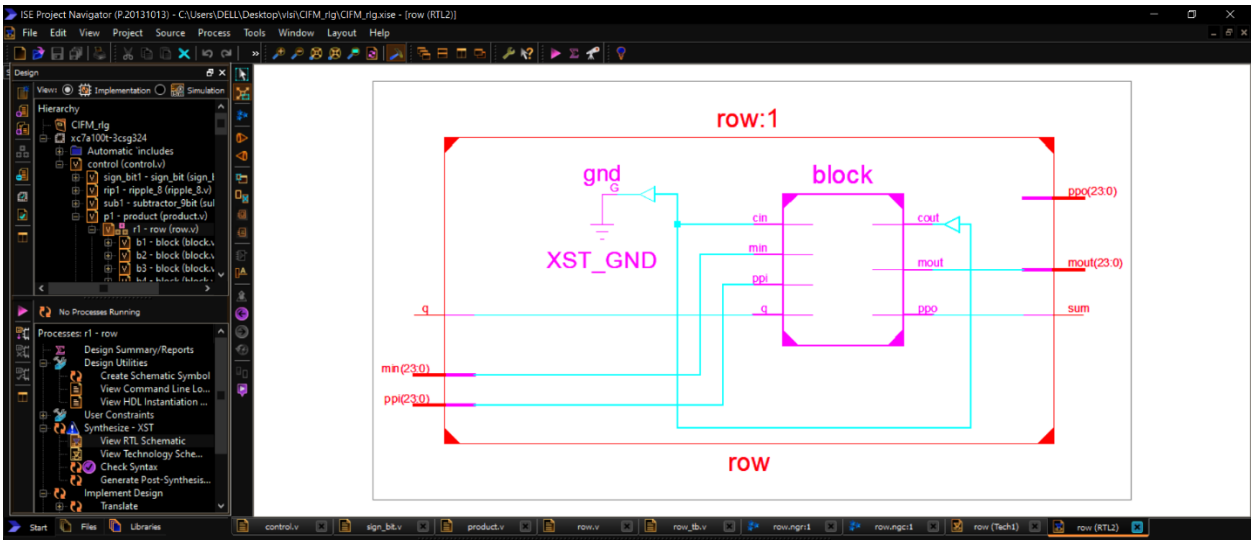
min = 24'b110011001100110011001100;

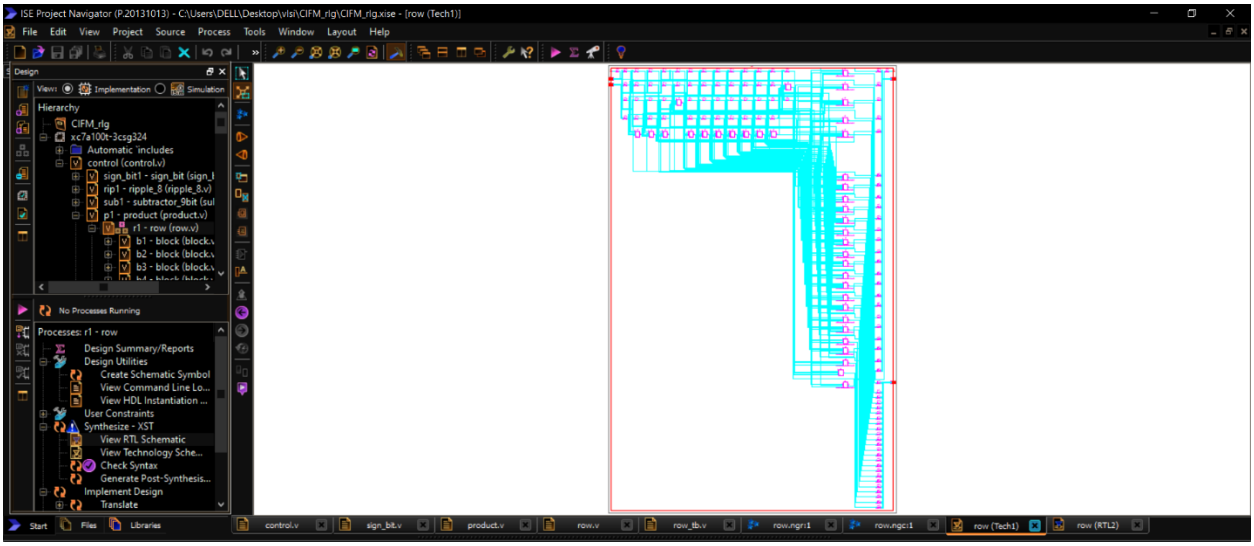
ppi = 24'b101010101010101010101010; q = 1'b1; #100;

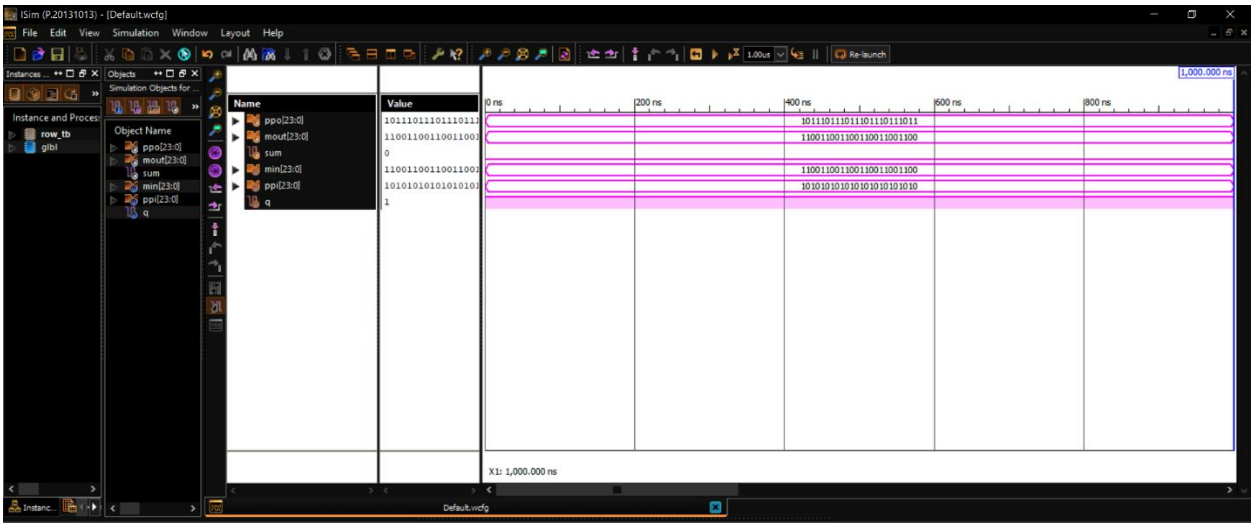
end endmodule

Schematics:




row:1
gnd
G
XST_GND
block
cin
min
ppi
q
cout
mout
ppo
ppo(23:0)
mout(23:0)
sum
min(23:0)
ppi(23:0)
row




Name
Value
ppo[23:0]
mout[23:0]
sum
min[23:0]
ppi[23:0]
q
0 ns
200 ns
400 ns
600 ns
800 ns
1,000.000 ns
X1: 1,000.000 ns

Code: (Final Product)

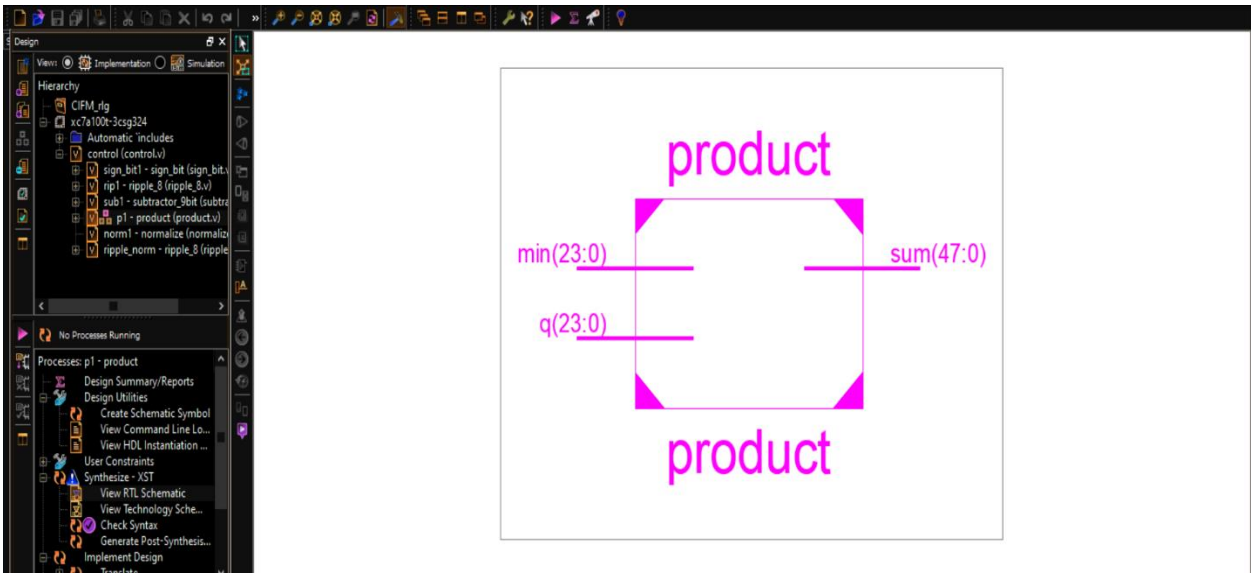
```
module product(output wire [47:0] sum,

input wire [23:0] min,input wire [23:0]q );

wire [23:0] temp1, temp2, temp3, temp4, temp5,temp6,temp7,temp8,temp9,temp10; //diagonal
m

wire [23:0] temp11,temp12, temp13, temp14, temp15, temp16,temp17, temp18,temp19,temp20;

wire [23:0] temp21, temp22,temp23,temp24;

wire [23:0] ptemp1, ptemp2, ptemp3, ptemp4, ptemp5, ptemp6, ptemp7, ptemp8,ptemp9,
ptemp10;

//vertical p

wire [23:0] ptemp11,ptemp12,ptemp13,
ptemp14,ptemp15,ptemp16,ptemp17,ptemp18,ptemp19,ptemp20;

wire [23:0] ptemp21, ptemp22,ptemp23;

row r1 (ptemp1, temp1, sum[0], min, 24'h000000, q[0]);

row r2 (ptemp2, temp2, sum[1], temp1, ptemp1, q[1]);

row r3 (ptemp3, temp3, sum [2], temp2, ptemp2, q[2]);

row r4 (ptemp4, temp4, sum [3],temp3, ptemp3, q[3]);

row r5 (ptemp5, temp5, sum [4], temp4, ptemp4, q[4]);

row r6 (ptemp6, temp6, sum [5],  temp5, ptemp5, q[5]);

row r7 (ptemp7, temp7, sum [6], temp6, ptemp6, q[6]);

row r8 (ptemp8, temp8, sum [7],  temp7, ptemp7, q[7]);

row r9 (ptemp9, temp9, sum [8], temp8, ptemp8, q[8]);

row r10(ptemp10, temp10, sum [9], temp9, ptemp9, q[9]);

row r11(ptemp11, temp11, sum[10], temp10, ptemp10, q[10]);

row r12(ptemp12, temp12, sum[11], temp11, ptemp11, q[11]);

row r13(ptemp13, temp13, sum[12], temp12, ptemp12, q[12]);
```

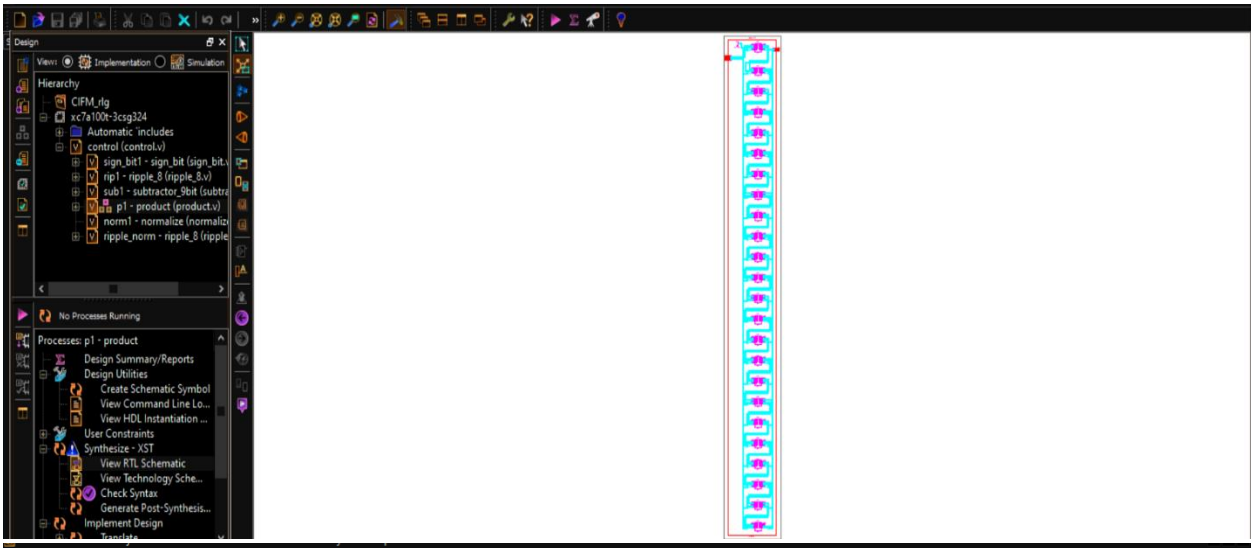
```
row r14(ptemp14, temp14, sum [13], temp13, ptemp13, q[13]);
row r15(ptemp15, temp15, sum [14],temp14, ptemp14, q[14]);
row r16(ptemp16, temp16, sum [15],  temp15, ptemp15, q[15]);
row r17(ptemp17, temp17, sum [16], temp16, ptemp16, q[16]);
row r18(ptemp18, temp18, sum [17],  temp17, ptemp17, q[17]);
row r19(ptemp19, temp19, sum [18], temp18, ptemp18, q[18]);
row r20(ptemp20, temp20, sum [19],  temp19, ptemp19, q[19]);
row r21 (ptemp21, temp21, sum [20],  temp20, ptemp20, q[20]);
row r22(ptemp22, temp22, sum [21], temp21, ptemp21, q[21]);
row r23(ptemp23, temp23, sum [22], temp22, ptemp22, q[22]);
row r24(sum [47:24], temp24, sum [23], temp23, ptemp23, q[23]);
endmodule
```

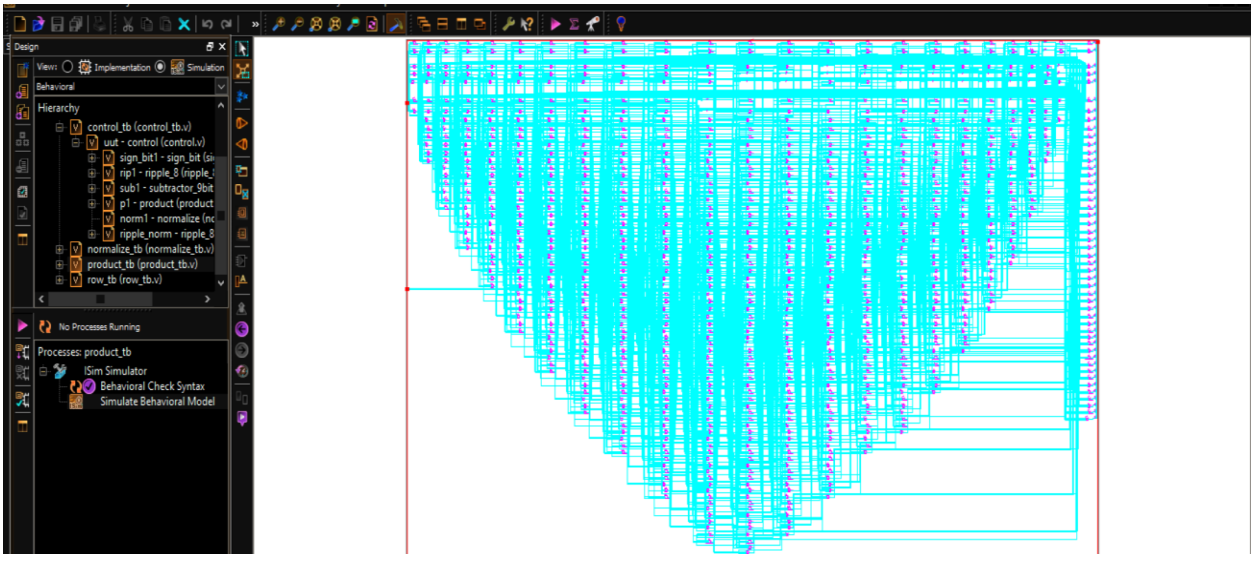
Test code:

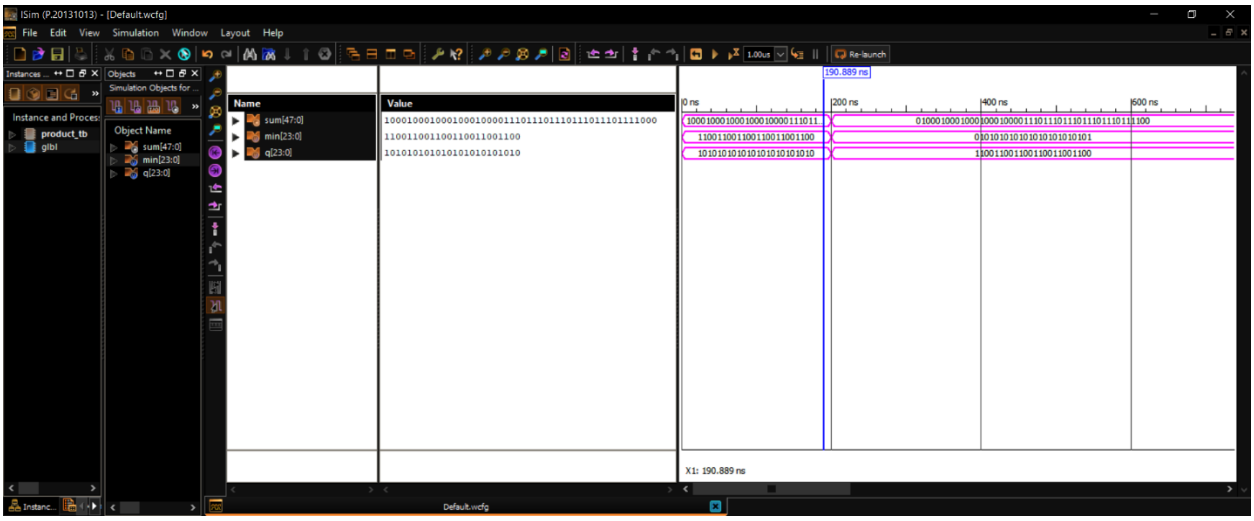
```
`timescale 1ns / 1ps
module product_tb;
reg [23:0] min; reg [23:0] q;
wire [47:0] sum;
product uut (.sum(sum), .min(min), .q(q));
initial begin
min = 24'b110011001100110011001100;
q = 24'b101010101010101010101010;  #200;
min = 24'b010101010101010101010101;
q = 24'b110011001100110011001100;  #200;
end
endmodule
```

Schematics:


product
min(23:0)
sum(47:0)
q(23:0)
product







5.Placing the binary point and normalizing the mantissa:

Code:

module normalize(

output wire [22:0] adj_mantissa, //adjusted mantissa (after extracting out required part)

output wire norm_flag, input wire [47:0] prdt); //returns norm =1 if normalization needs to be done.

and (norm_flag, prdt[47], 1'b1);

wire [22:0] results[1:0];

assign results[0] = prdt [45:23];

assign results[1] = prdt [46:24];

assign adj_mantissa = {results[norm_flag+0]};

endmodule

Test code:

`timescale 1ns / 1ps

module normalize_tb;

reg [47:0] prdt;

wire [22:0] adj_mantissa;

wire norm_flag;

normalize uut (.adj_mantissa(adj_mantissa), .norm_flag(norm_flag), .prdt(prdt));

initial begin

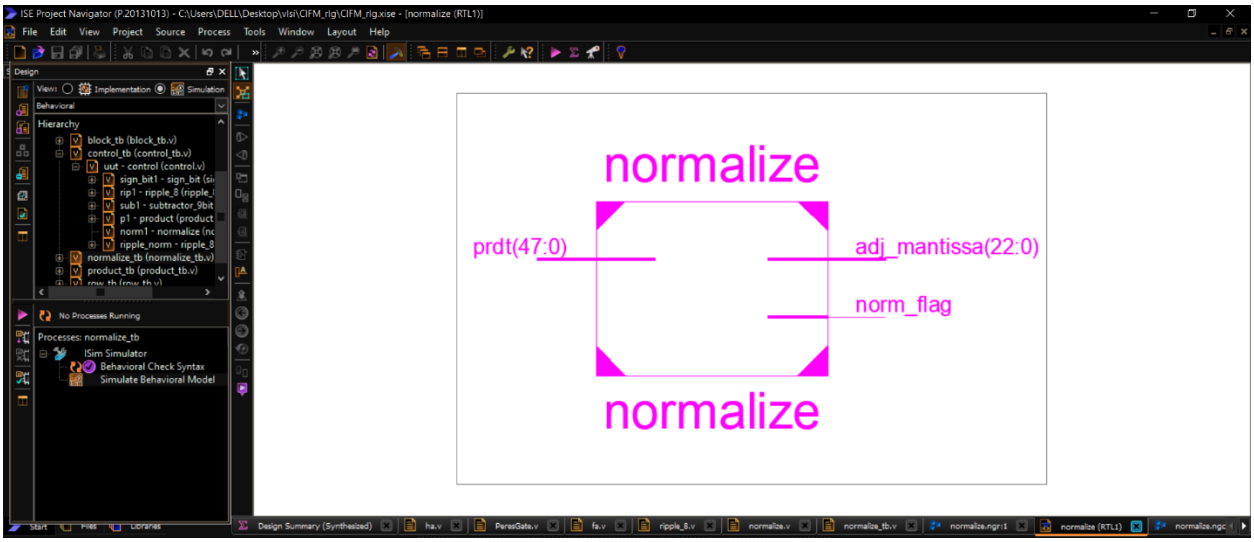
prdt = 48'b101101010010110110010101101010101100110101011001;

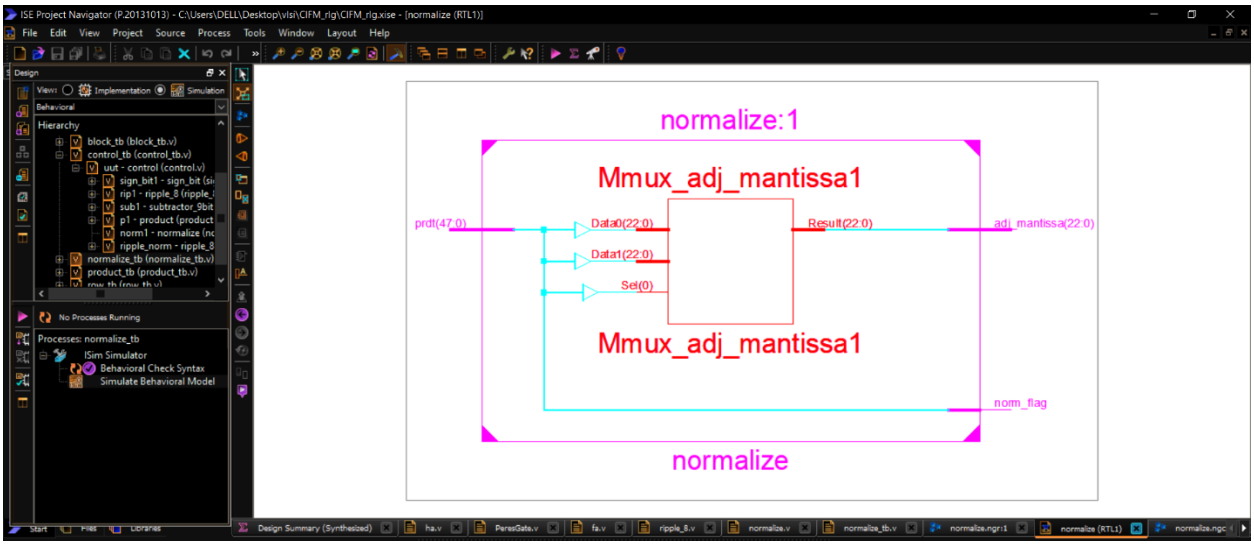
#200;

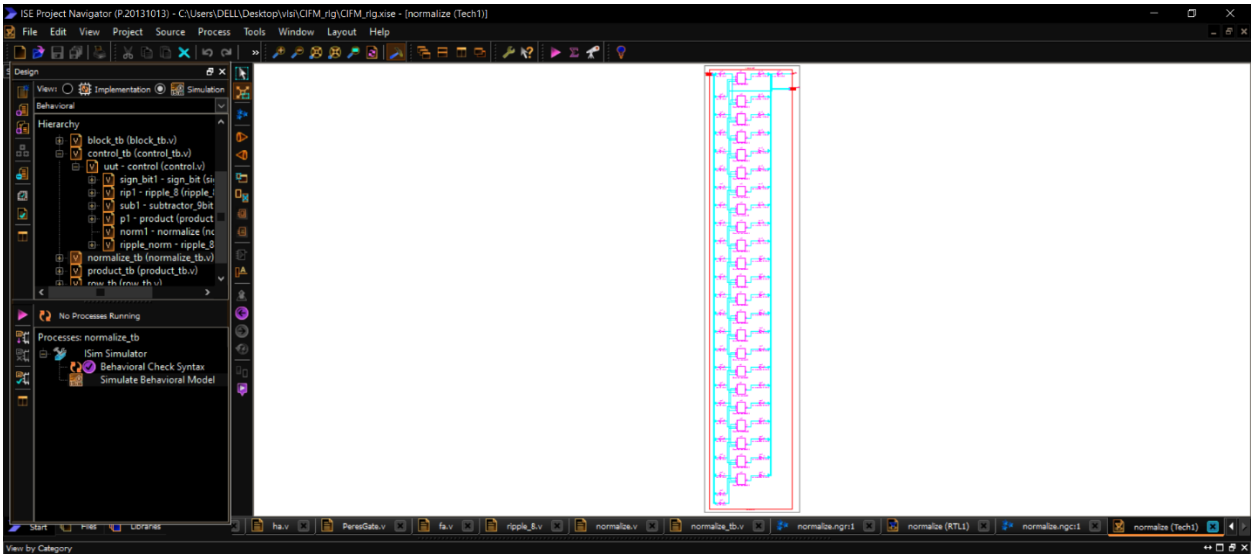
end

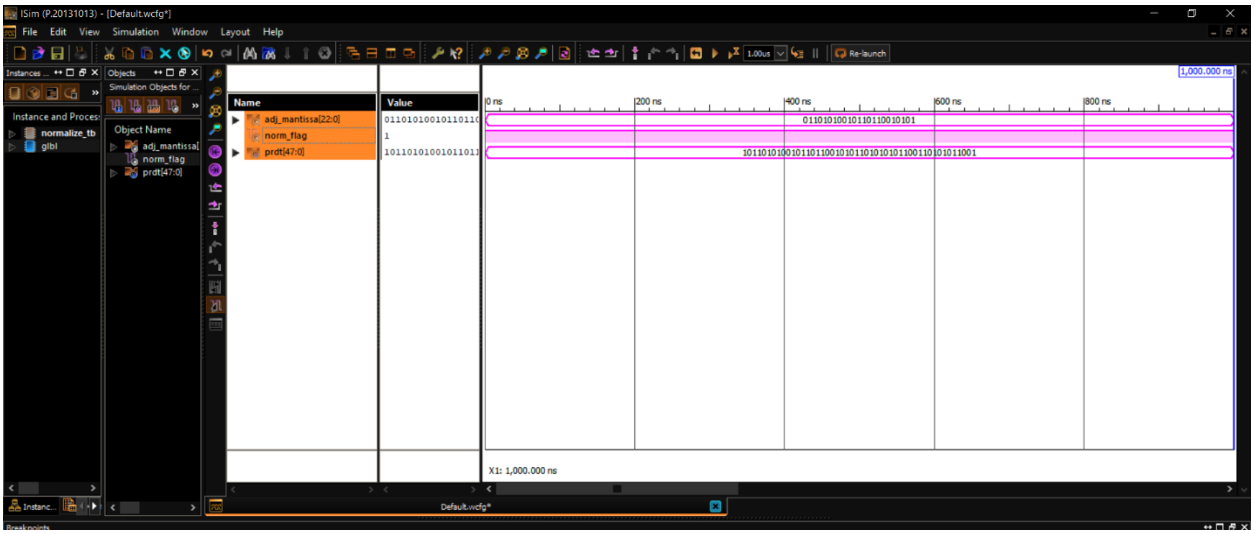
endmodule

Schematics:









6.Underflow /overflow checking and Control module for operating other modules:

Code:

```
module control(input wire [31:0] inp1,input wire [31:0] inp2,

output wire [31:0] out,output wire underflow,

output wire overflow);

wire sign;wire [7:0] exp1; wire [7:0] exp2;

wire [7:0] exp_out; wire [7:0] test_exp;

wire [22:0] mant1; wire [22:0] mant2;
```

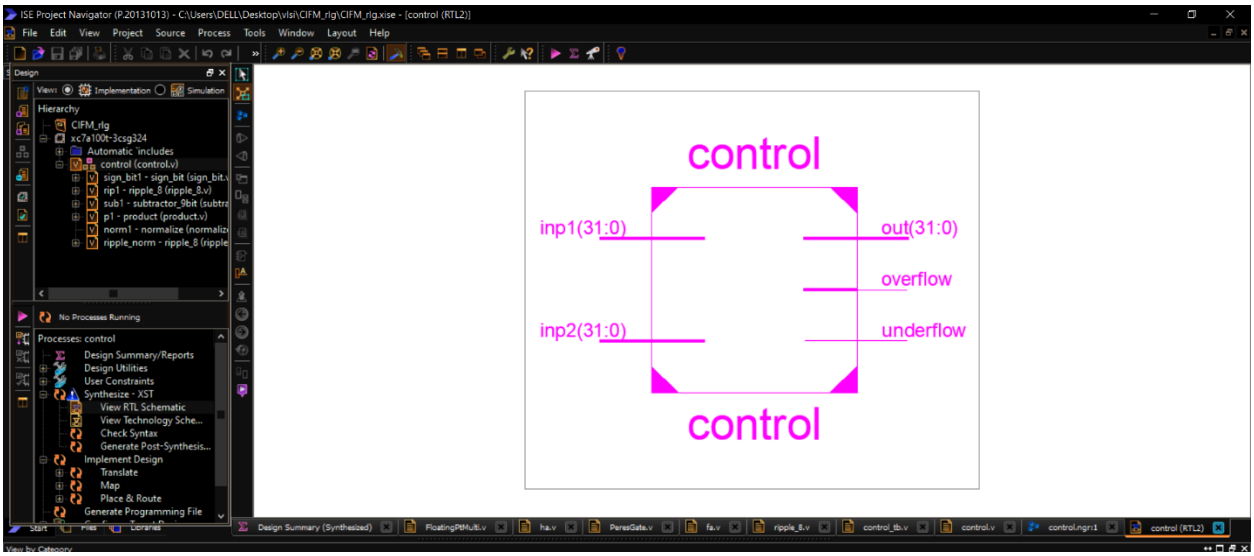
```
wire [22:0] mant_out;
sign_bit sign_bit1(sign, inp1, inp2);
wire [7:0] temp1;
wire dummy; //to connect unused cout ports of adder wire carry
wire [8:0] sub_temp;
ripple_8 rip1(temp1, carry, inp1 [30:23], inp2 [30:23],1'b0);

subtractor_9bit sub1 (sub_temp, underflow, {carry, temp1}, 1'b0);
//if there is a carry out => underflow
and (overflow, sub_temp [8], 1'b1); //if the exponent has more than 8 bits: overflow
//taking product of mantissa:
wire [47:0] prdt;
product p1 (prdt, {1'b1, inp1 [22:0]},{1'b1, inp2[22:0]});
wire norm_flag;
wire [22:0] adj_mantissa;
normalize norm1(adj_mantissa, norm_flag,prdt);
ripple_8 ripple_norm(test_exp, dummy,sub_temp [7:0], {7'b0, norm_flag}, 1'b0);
assign out [31] = sign;
assign out [30:23] = test_exp;
assign out [22:0] = adj_mantissa;
endmodule
```

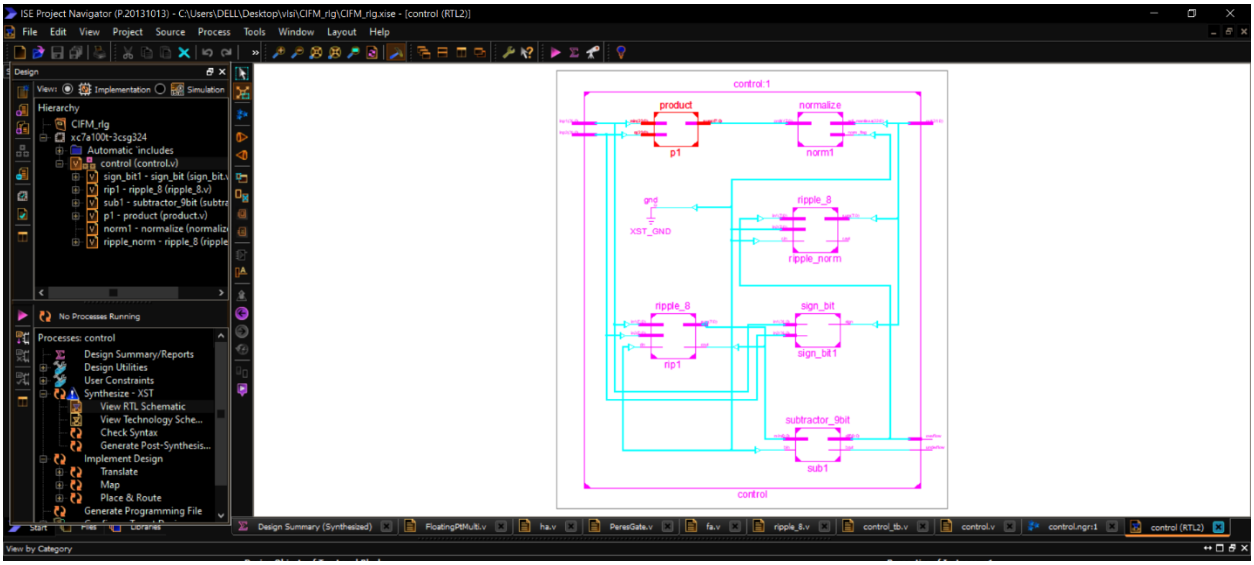
Test code:

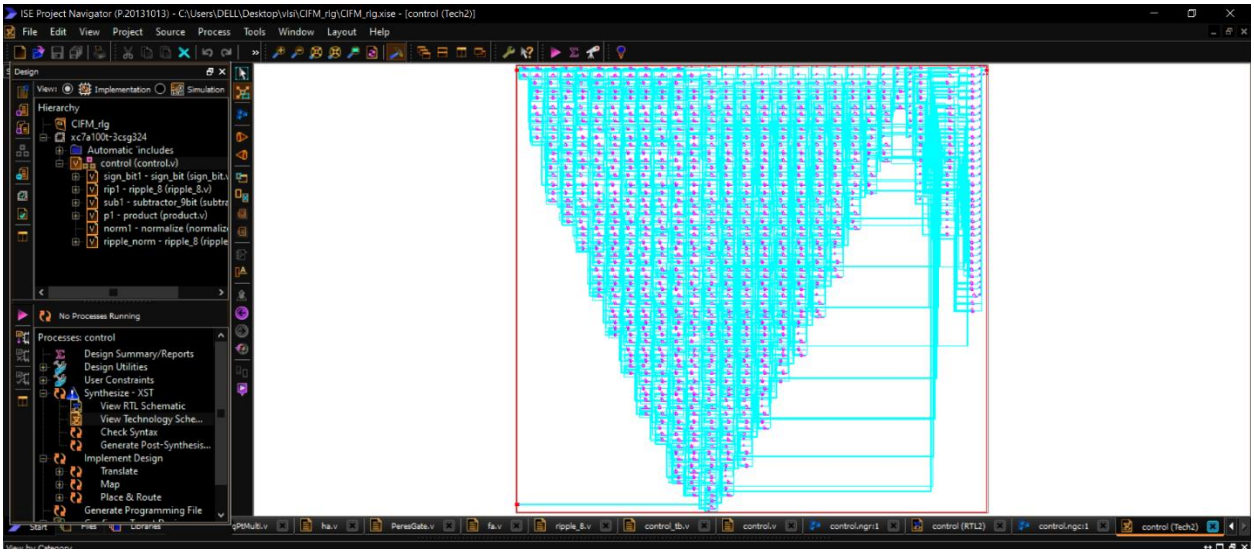
```
`timescale 1ns / 1ps
module control_tb;
reg [31:0] inp1; reg [31:0] inp2;
```

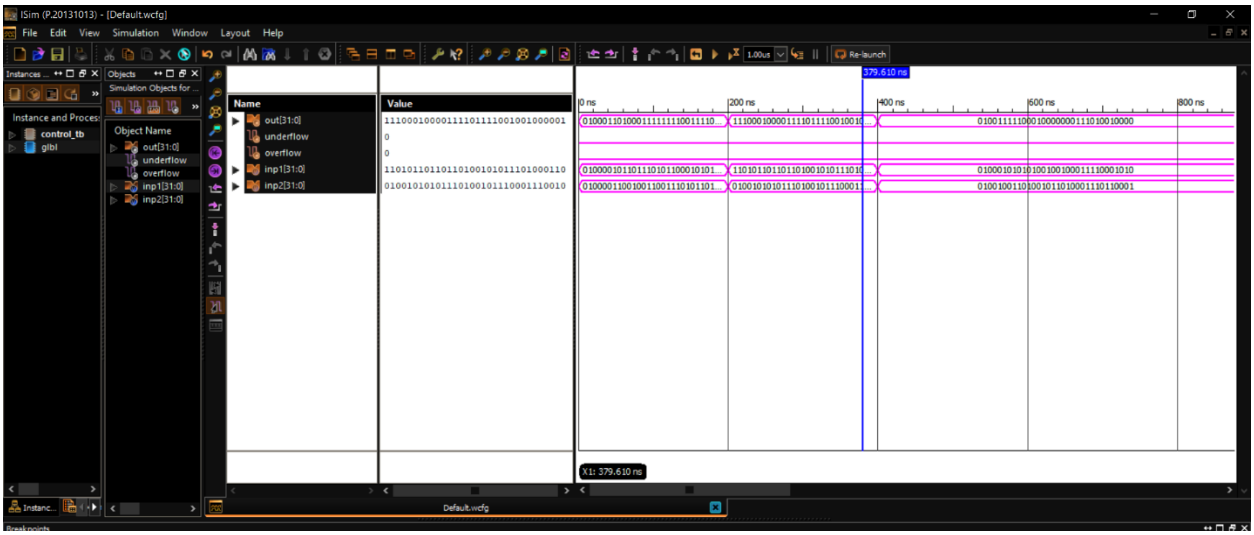
```
wire [31:0] out;
wire underflow;
wire overflow;
// Instantiate the Unit Under Test (UUT)
control uut (.inp1(inp1), .inp2(inp2), .out(out),
                .underflow(underflow), .overflow(overflow));
initial begin
inp1 = 32'b01000010110111010110001010110010;
inp2 = 32'b01000011001001100111010110110110;#200;
inp1 = 32'b11010110110110100101011101000110;
inp2 = 32'b01001010101110100101110001110010;#200;
inp1 = 32'b01000101010100100100011110001010;
inp2= 32'b01001001101001011010001110110001;#200;
end
endmodule
```

Schematics:




ISE Project Navigator (P.20131013) - C:\Users\DELL\Desktop\vlsi\CIFM_rlg\CIFM_rlg.xise - [control (RTL2)]
File Edit View Project Source Process Tools Window Layout Help
control:1
product
p1
normalize
norm1
gnd
XST_GND
ripple_8
ripple_norm
ripple_8
rip1
sign_bit
sign_bit1
subtractor_9bit
sub1
control


ISE Project Navigator (P.20131013) - C:\Users\DELL\Desktop\vlsi\CIFM_rlg\CIFM_rlg.xise - [control (Tech2)]
File Edit View Project Source Process Tools Window Layout Help


ISim (P.20131013) - [Default.wcfg]
File Edit View Simulation Window Layout Help
Name
Value
out[31:0]
underflow
overflow
inp1[31:0]
inp2[31:0]
379.610 ns
0 ns
200 ns
400 ns
600 ns
800 ns
X1: 379.610 ns
Default.wcfg

Combined integer and floating point multiplier

Code:

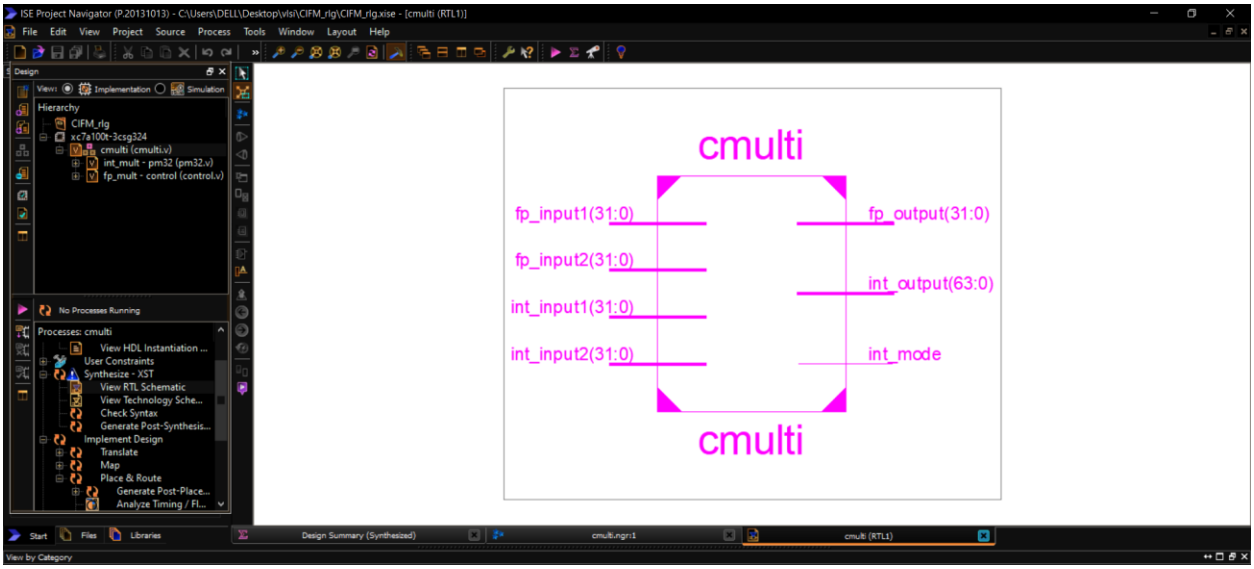
```
module cmulti(
    input wire [31:0] int_input1, // Integer input 1
    input wire [31:0] int_input2, // Integer input 2
    input wire [31:0] fp_input1,  // Floating-point input 1
    input wire [31:0] fp_input2,  // Floating-point input 2
    output wire [63:0] int_output, // Integer output
    output wire [31:0] fp_output,  // Floating-point output
    output wire int_mode    );
    wire [63:0] int_out;
    wire [31:0] fp_out;
    pm32 int_mult(
        .a(int_input1),
        .b(int_input2),
        .c(int_out) );
   control fp_mult( .inp1(fp_input1),
        .inp2(fp_input2),
        .out(fp_out),
        .underflow(), // Assuming underflow and overflow are not needed for floating-point output
        .overflow() );
    assign int_output = int_out;
    assign fp_output = fp_out;
    assign int_mode = (int_input1 !== 32'b0 && int_input2 !== 32'b0) ? 1'b1 : 1'b0;
endmodule
```

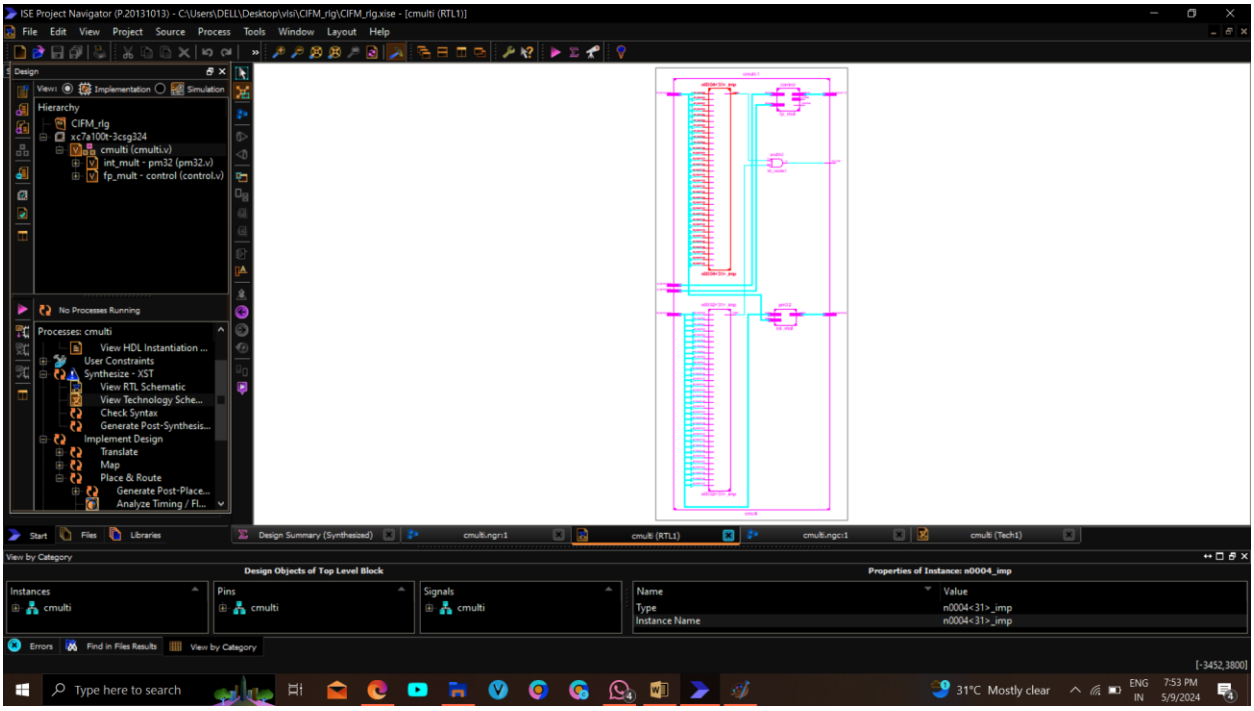
Test code:

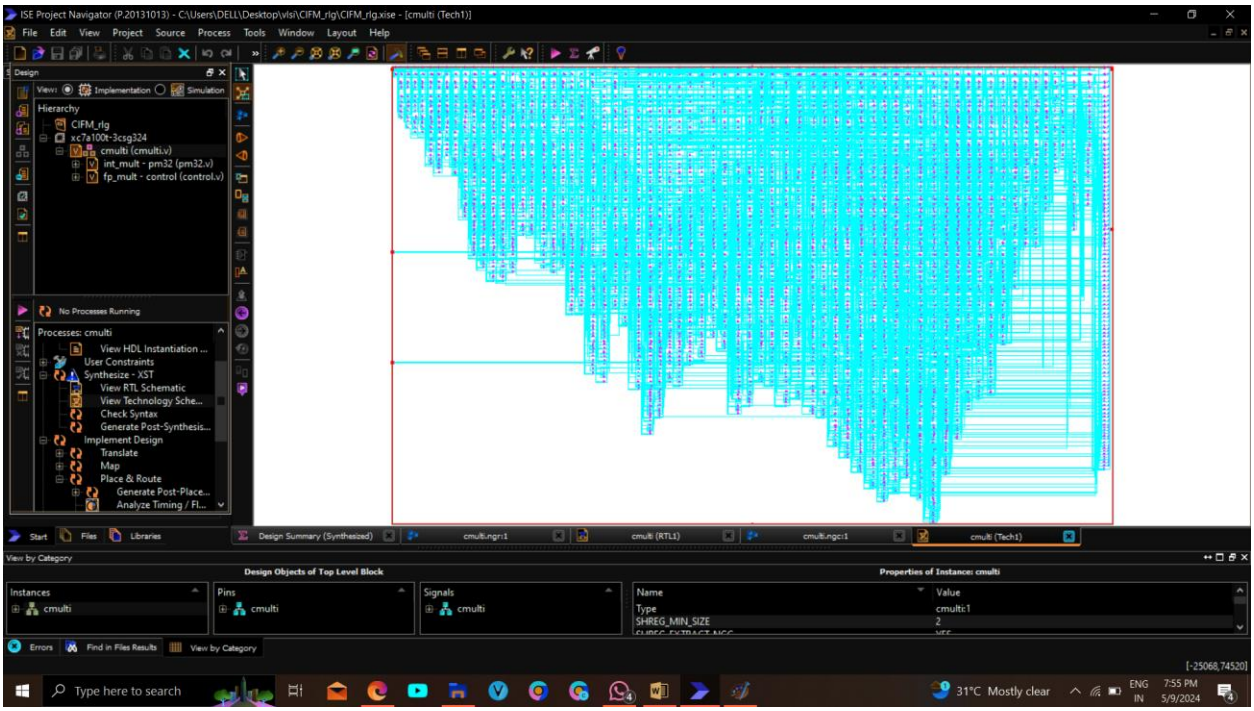
```
`timescale 1ns / 1ps
module cmulti_tb;
        reg [31:0] int_input1;
        reg [31:0] int_input2;
        reg [31:0] fp_input1;
        reg [31:0] fp_input2;
        wire [63:0] int_output;
        wire [31:0] fp_output;
        wire int_mode;
        cmulti uut (.int_input1(int_input1), .int_input2(int_input2),
                .fp_input1(fp_input1), .fp_input2(fp_input2),
                .int_output(int_output), .fp_output(fp_output), .int_mode(int_mode));
        initial begin
      int_input1 =32'h0000ffff;//65535
      int_input2 =32'h0000ffff;//65535
      fp_input1=0;fp_input2=0;#200;
      int_input1=0; int_input2=0;
      fp_input1 = 32'b01000000001000000000000000000000;//2.5
      fp_input2 = 32'b01000000001000000000000000000000;//2.5#200;
      int_input1=0; int_input2=0;
      fp_input1 = 32'b01000000000000000000000000000000;//2.0
      fp_input2 = 32'b01000000001000000000000000000000;//2.5#200;
      int_input1=32'h0000ffff; int_input2=32'h0000ffff;
      fp_input1 = 32'b01000000001000000000000000000000;//2.5
```

```
        fp_input2 = 32'b01000000001000000000000000000000;//2.5#200;
        int_input1=0;int_input2=0;fp_input1 = 32'b01000010110111010110001010110010;
        fp_input2 = 32'b01000011001001100111010110110110;
end
endmodule
```

Schematics:







Waveform: