VLSI PROJECT

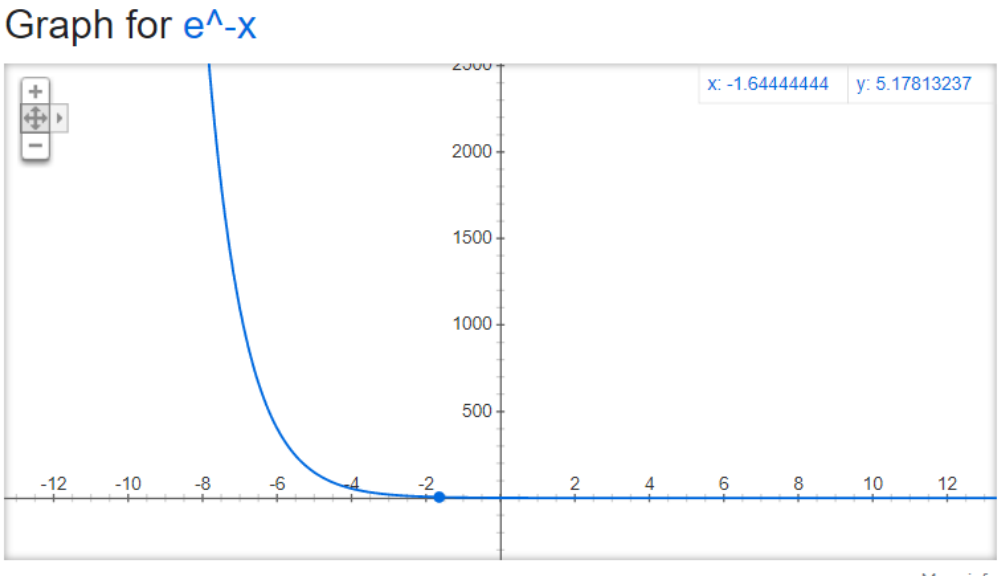
Project Instructor:-Dr. Babita Jajodia

OBJECTIVES:-

=>Design the y = exp(-x) when x is of 16 bit.

Theory:-

=>The exponential function is a mathematical function denoted by F(x) = exp(-x).



=>The Expansion of this function using Taylor series.

=>The Taylor series of a function is an infinite sum of terms that are expressed in terms of the functions derivatives at a single point.

=>$(e)^{-x} = \sum_{n=0}^{inf}(-x)^n/n!$ -inf < x < +inf

=>For simplicity, I take n = 3 which reduce the equation to

$$e^x = 1 - \frac{x}{1!} + \frac{x^2}{2!} - \frac{x^3}{3!} + \cdots, \qquad -\infty < x < \infty$$

Algorithm:-

For Implementation of Exponential function, Basically I used Three algorithm:-

1).Booth's Multiplier =>

for x^2, I just take a 16 bit 'x' and 16 bit 'y' as a input and my output is 32 bit 'z'.
After that for x^3, I just multiply another 16 bit 'x' into 'z' to get my final 48 bit output.
By Using Booth's Multiplier we get our 3rd and 4th term of the expression without divisor i.e. x^2 ans x^3.

2).Non-Restoring Division =>

Once we get x^2 and x^3 our next task is to create x^2/2 and x^3/6.
This is done by Non-restoring division algorithm. Basically till now we have our 32 bit and 48 bit dividend. We have to just create two divisor one is

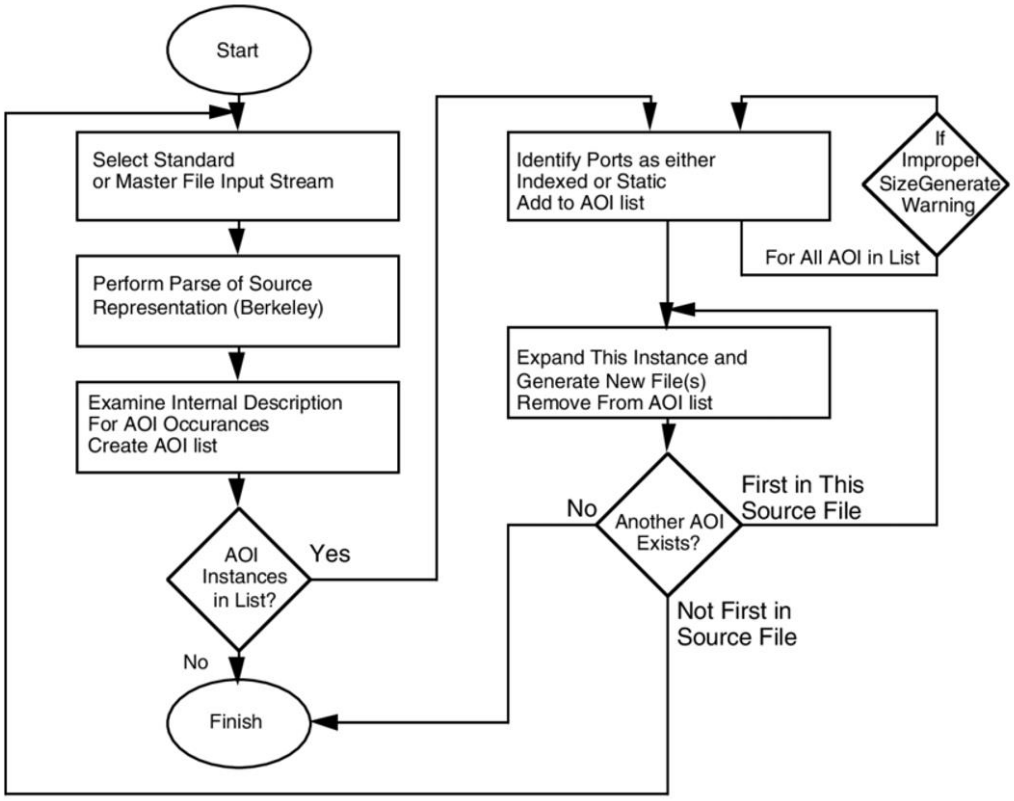
2(which is 32 bit). and another is 6(which is 48 bits).
We have also take two parameter as quotient and remainder which is 48 bit(each).
After calling this Non-restoring division algorithm in our main function with the parameter as discussed above we get our last two term of exponential function which is ‘x^2/2’ and ‘x^3/6’.

3).Adder Algorithm=>

Basically this algorithm is give us the summation of all the term.
But before that, we have pass our ‘x’ and ‘x^3/6’ through negation(just apply ‘~’) because it is negative in expression.
After passing through negation we get our actual terms with sign.
Finally we have to start the adding procedure.
Just add last two terms and stores in any new term which is 48 bits(let’s say z).
After that we add z with remaining terms so that it takes two term as a output.
Through this process we can add all the terms in the exponential expression and store in the final output which is 48 bits.

Now, Finally I created a Main function where I call every algorithm with valid parameter.
In the Test bench we have to just give the value of x and it automatically gives the output after all calculation.
Like for x = 0 => exp(-x) = 1.

=>Flowchart Diagram:-



=>Code:-

```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////////
// Company:
```

```
// Engineer:
//
// Create Date: 02.03.2022 18:21:22
// Design Name:
// Module Name: Main
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////////////////


module Adder(A, B, Ci, S, Co);
input[47:0] A;
input[47:0] B;
input Ci;
output[47:0] S;
output Co;
wire[48:0] Sum33;
assign Sum33 = A + B + Ci ;
```

```
assign S = Sum33[47:0] ;
assign Co = Sum33[48] ;
endmodule

//multiplication for 32
module Multiplier(x,y,z);
input signed[15:0] x;
input signed[15:0] y;
output signed [31:0] z;
reg signed[31:0] z;
reg [1:0] temp;
reg e;
reg [15:0] y1;
integer i;
always@(x,y)
begin
e=1'd0;
z=32'd0;
for(i=0;i<16;i=i+1)
begin
temp={x[i],e};
y1=-y;

case(temp)
2'd2:z[31:16]=z[31:16]+y1;
2'd1:z[31:16]=z[31:16]+y;
default:begin
end
```

```
endcase
z=z >>1;
z[31]=z[30];
e=x[i];
end
if(y==16'd32768)
z=-z;
end
endmodule

//////////////////////////////////////multiplicatio
n for 48 bit////////////////////////////////
module Multiplier2(x,y,z);
input signed[31:0] x;
input signed[31:0] y;
output signed [63:0] z;
reg signed[63:0] z;
reg [1:0] temp;
reg e;
reg [31:0] y1;
integer i;
always@(x,y)
begin
e=1'd0;
z=64'd0;
for(i=0;i<32;i=i+1)
begin
temp={x[i],e};
```

```
y1=-y;

case(temp)
2'd2:z[63:32]=z[63:32]+y1;
2'd1:z[63:32]=z[63:32]+y;
default:begin
end
endcase
z=z >>1;
z[63]=z[62];
e=x[i];
end
if(y==32'd2147483648)
z=-z;
end
endmodule

////////////////////////////////division//////////////
//////////
module Divide(clk, reset, Dividend, Divisor,
Quotient, Remainder);
	input clk, reset;
	input [47:0] Dividend, Divisor;

	output [47:0] Quotient, Remainder;
	reg [47:0] Quotient, Remainder;

	reg [47:0] p, a, temp;
```

```
        integer i;

        always @(posedge clk, negedge reset)
        begin
            if( !reset )
            begin
                Quotient <= 0;
                Remainder <= 0;
            end
            else
            begin
                Quotient <= a;
                Remainder <= p;
            end
        end

        always @(*)
        begin
            a = Dividend;
            p = 0;

            for(i = 0; i < 48; i = i+1)
            begin
                //Shift Left carrying a's MSB into p's
LSB
                p = (p << 1) | a[47];
                a = a << 1;
```

```
                //store value in case we have to
restore
                temp = p;

                //Subtract
                p = p - Divisor;

                if( p[47] ) // if p < 0
                        p = temp; //restore value
                else
                        a = a | 1;
            end
        end

endmodule

//////////////////////////main_module//////////////
///////

module main_exp(clk,reset,x,z);

output [47:0] z;
input clk,reset;
input [15:0] x;

wire [31:0] mult_32;
Multiplier mult32(x,x,mult_32);
wire [47:0] mult_48;
```

```
Multiplier2 mult48(mult_32,x,mult_48);
wire [47:0] div6,div2;
wire [47:0] remainder6,remainder2;
Divide
divsix(clk,reset,mult_48,48'b110,div6,remainder6);
Divide
divtwo(clk,reset,mult_32,48'b10,div2,remainder2);

wire [47:0] sum1,sum2,z;
wire co1,co2,co3;
Adder add1(~div6,div2,0,sum1,co1);
Adder add2(sum1,~x,co1,sum2,co2);
Adder add3(48'b1,sum2,co2,z,co3);


Endmodule
```

=>Test Bench:-

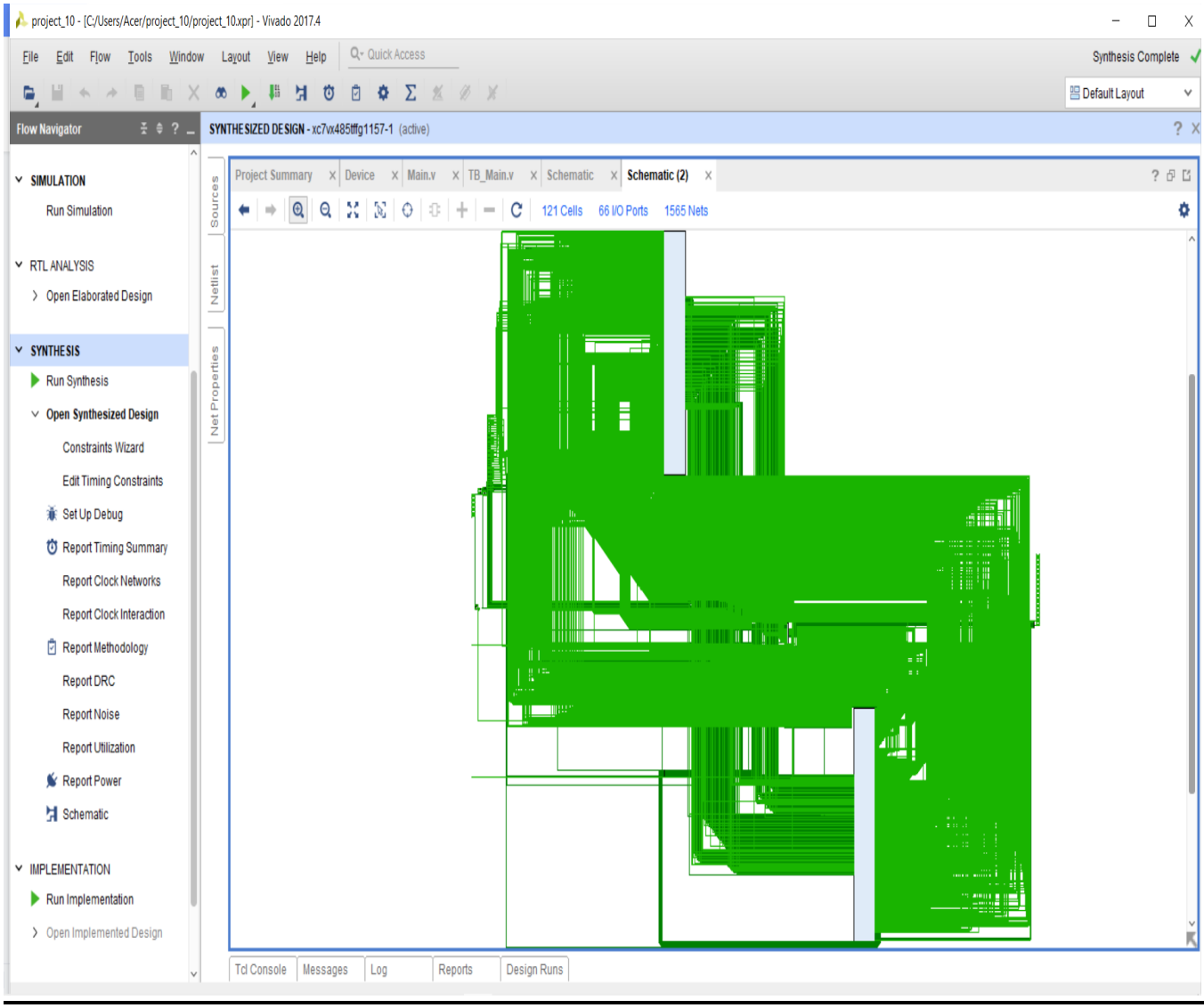
```
`timescale 1ns / 1ps
//////////////////////////////////////////////////////////////////////////////
//////////////////////////
// Company:
// Engineer:
//
// Create Date: 02.03.2022 18:22:56
// Design Name:
// Module Name: TB_Main
```

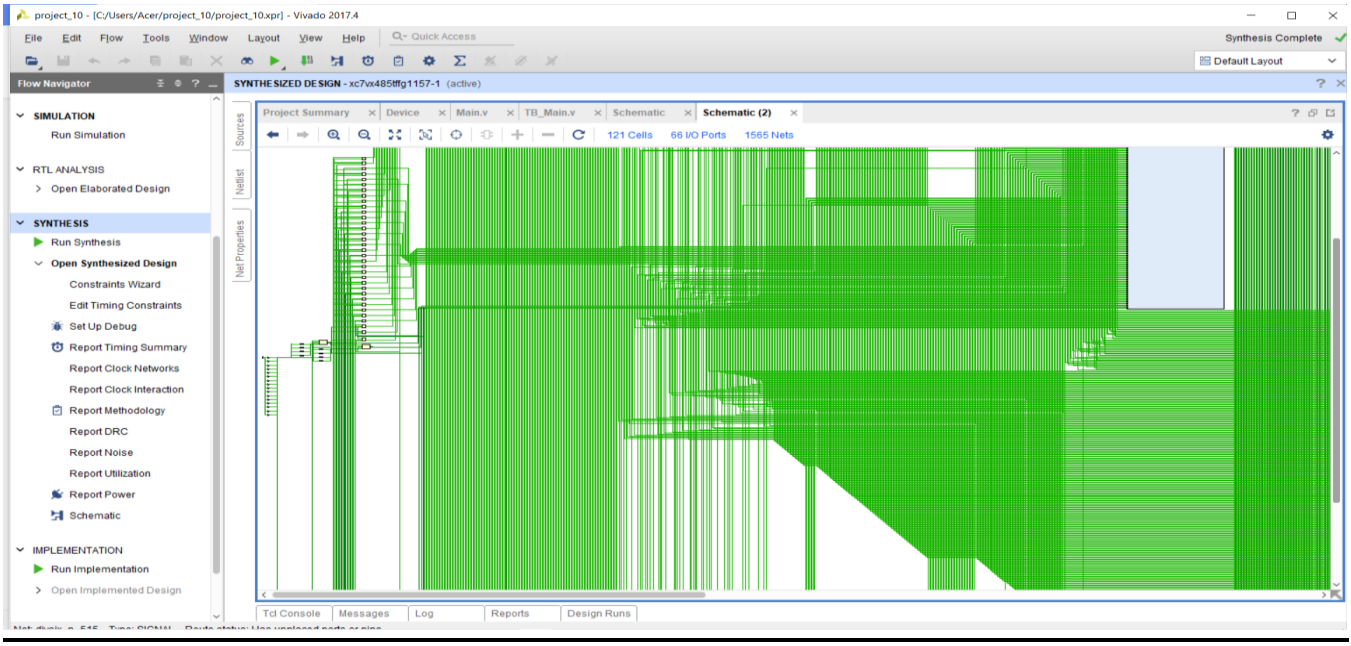
```
// Project Name:
// Target Devices:
// Tool Versions:
// Description:
//
// Dependencies:
//
// Revision:
// Revision 0.01 - File Created
// Additional Comments:
//
//////////////////////////////////////////////////////////////////
//////////////////////////


module tb_negExp;
	reg clk, reset;
	reg [15:0] x;
	wire [47:0] z;

	main_exp negExp(clk, reset,x,z);

	initial
		forever #1 clk = ~clk;

	initial
		$monitor($time,"e^%b is %b", x,z);
```

```
initial
begin
        clk = 0;
        reset = 0;

        #1;
        reset = 1;
        x = 16'b1;

        #5;
        x=16'b0;



        #5;
        x= 16'b10111;



        #5;
        x= 16'b1000;



        #5;
    x=16'b1010;

        #5;
        $finish;
end
```

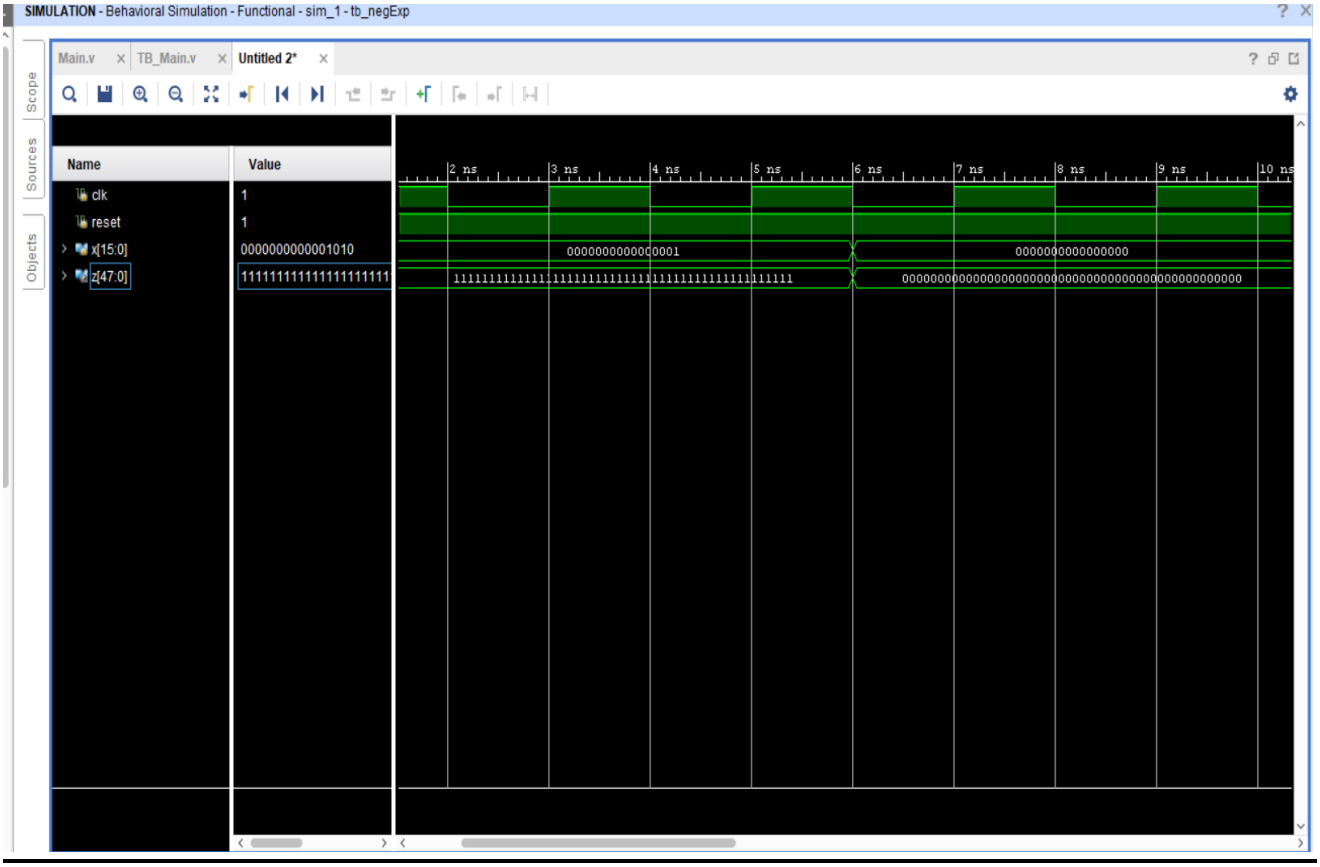
endmodule

=>Schematic Diagram:-

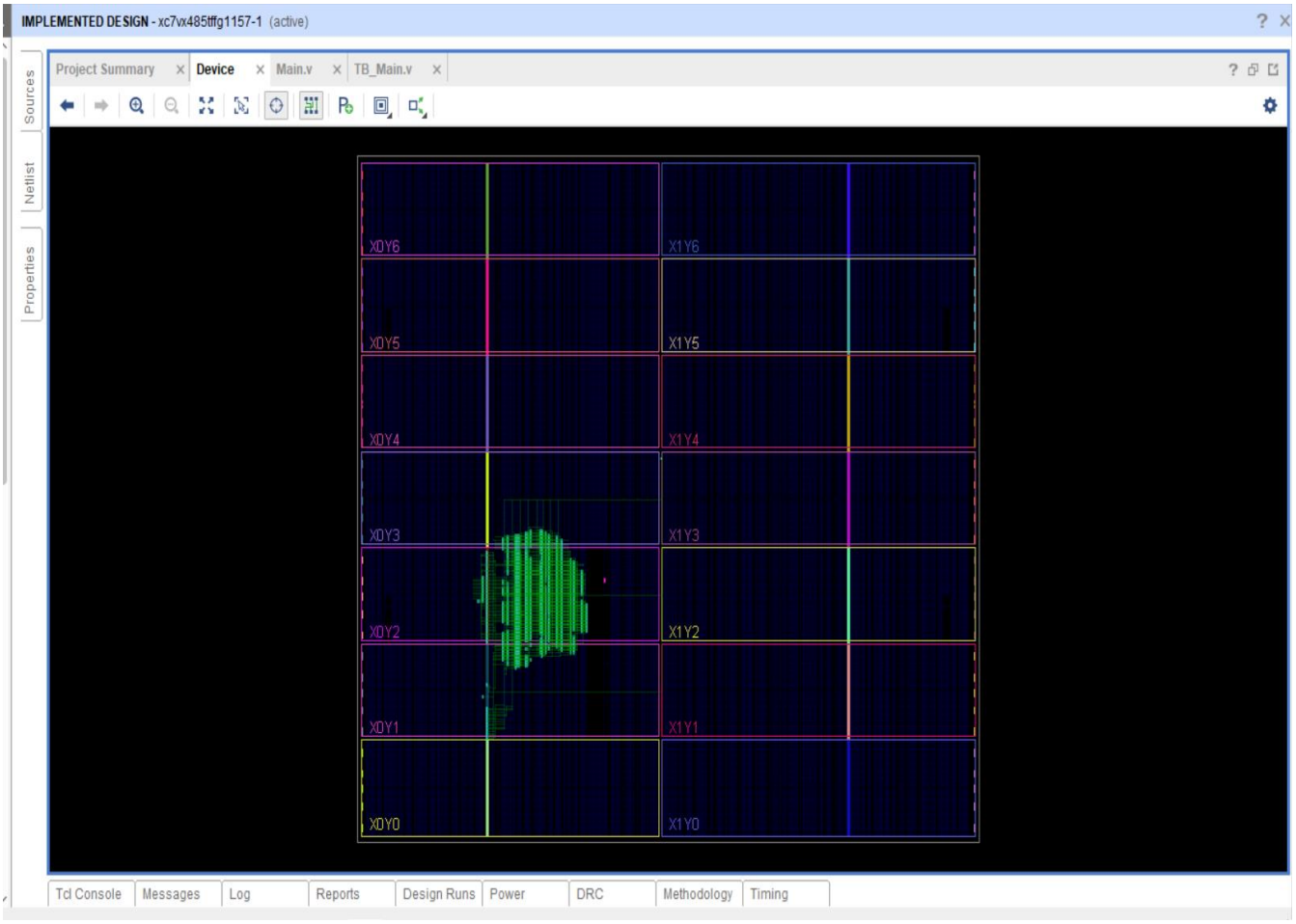




=>Output Result:-



=>Hardware Implementation(Implemented Design):-



=>Conclusion:-

=>We have successfully implement the exponential algorithm using Verilog code.
First of all getting the knowledge of all the algorithm which helps to implement this project like Booth's multiplier, Non-restoring divide, adder etc.

Name=>Sanu Kumar
Roll=> 1901173
Group=>11
Submission Date=>02/03/22