**VERILOG-All Modeling Styles:**

**1).GATE LEVEL MODELING:**

Designing a complex circuit using basic Logic Gates is known as Gate-Level Modeling.

**For Example:** or(Y, A, B)

and(Y, A, B)

**2). DATA FLOW MODELING:**

Compared to Gate Level Modeling, Data Flow Modeling is higher level of abstraction. Data Flow Modeling Describes the circuits by their function rather than by their Gate Structure.

**For Example:** assign Y = A|B

assign Y = A & B

3). Behavioral Modeling:

[Behavioral modeling](https://technobyte.org/behavioral-modeling-verilog/) is the highest level of abstraction in the [Verilog HDL](https://technobyte.org/verilog-course-tutorials/). Behavioral models in Verilog contain [procedural assignment statements](https://technobyte.org/2020/03/behavioral-modeling-style-in-verilog/#Procedural_assignment_statements), which control the simulation and manipulate variables of the data types. This level of abstraction simulates the behavior of the circuits without specifying the details.

**For Example:** always @ (A or B) begin……end

always @ (A and B) begin……end

**Structural modeling**

Structural modeling describes the hardware structure of a digital system. It is usually written in RTL and is somewhat similar to gate-level modeling. The only difference is it doesn’t include any built-in gates. The components and connections all need to separately defined here. There’s a proper definition for the expression of the digital system within the module itself.  It includes module declaration and instantiation, port-list and it’s associates.

**Gate Level Modeling:**

module orgate123(

input a,

input b,

output c

);

or(c,a,b);

endmodule

**Test Bench:**

module ortb;

// Inputs

reg a;

reg b;

// Outputs

wire c;

// Instantiate the Unit Under Test (UUT)

orgate123 uut (

.a(a),

.b(b),

.c(c)

);

initial begin

// Initialize Inputs

a = 0;

b = 0;

// Wait 100 ns for global reset to finish

#100;

a = 0;

b = 1;

// Wait 100 ns for global reset to finish

#100;

a = 1;

b = 0;

// Wait 100 ns for global reset to finish

#100;

a = 1;

b = 1;

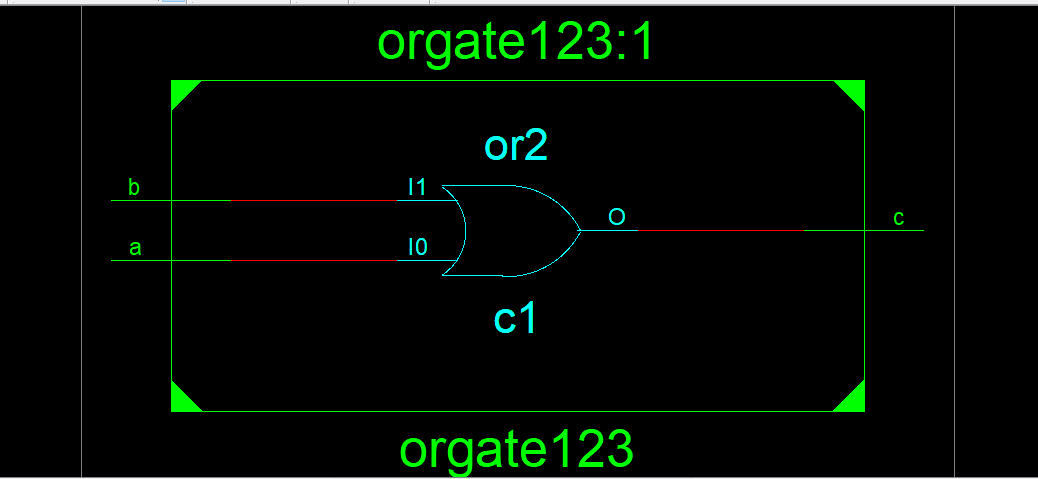
// Wait 100 ns for global reset to finish

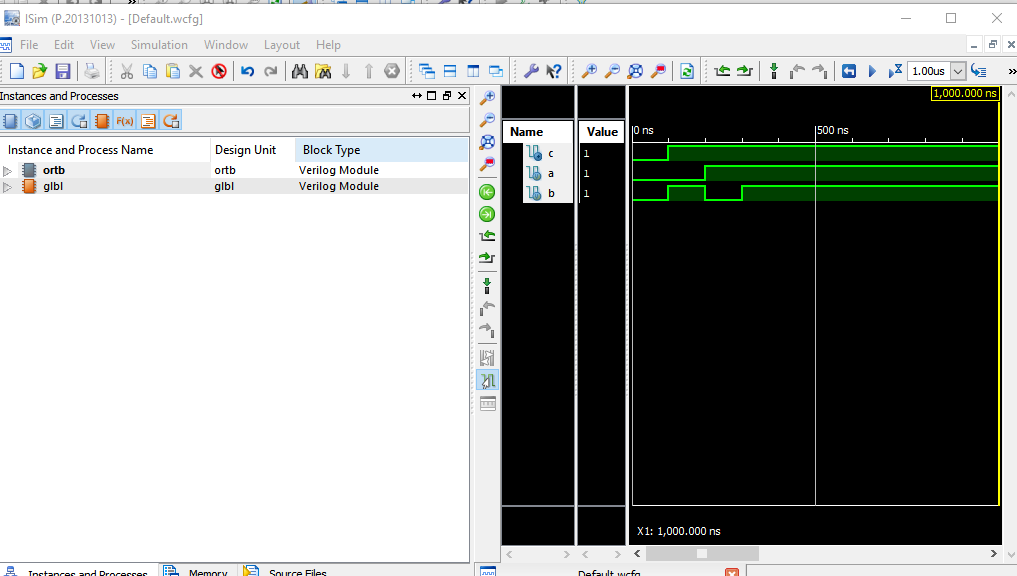
#100;

// Add stimulus here

end

endmodule

**RTLSchematic:**

**Graph:**

**Verilog and Gate:**

**Gate level Coding:**

module andgate(

input a,

input b,

output y

);

and(y,a,b);

endmodule

**Test-bench for and Gate:**

module andtb;

// Inputs

reg a;

reg b;

// Outputs

wire y;

// Instantiate the Unit Under Test (UUT)

andgate uut (

.a(a),

.b(b),

.y(y)

);

initial begin

// Initialize Inputs

a = 0;

b = 0;

// Wait 100 ns for global reset to finish

#100;

a = 0;

b = 1;

// Wait 100 ns for global reset to finish

#100;

a = 1;

b = 0;

// Wait 100 ns for global reset to finish

#100;

a = 1;

b = 1;

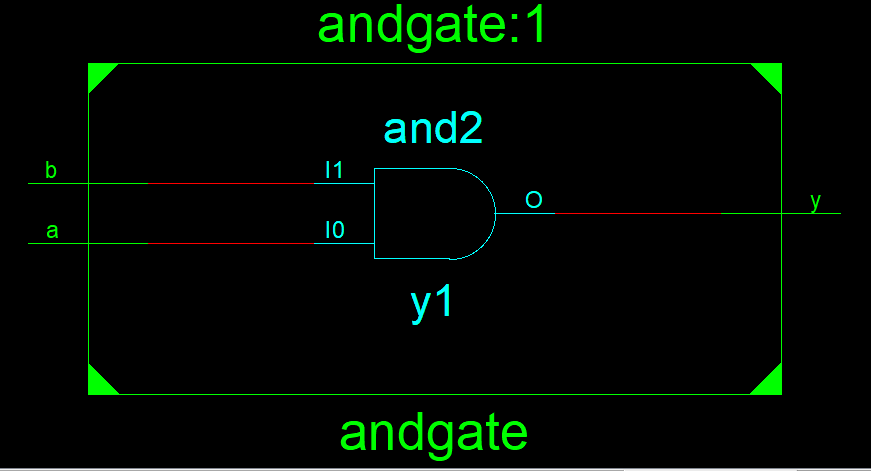
// Wait 100 ns for global reset to finish

#100;

// Add stimulus here

end

endmodule

**RTL SCHEMATIC:**

**Graph:**

****

**Verilog Not Gate:**

**Gate level coding:**

Module notgate(

input a,

output y

);

not(y,a);

endmodule

**Test Bench for Not Gate:**

module notgtb;

// Inputs

reg a;

// Outputs

wire y;

// Instantiate the Unit Under Test (UUT)

notgate uut (

.a(a),

.y(y)

);

initial begin

// Initialize Inputs

a = 0;

// Wait 100 ns for global reset to finish

#100;

// Initialize Inputs

a = 1;

// Wait 100 ns for global reset to finish

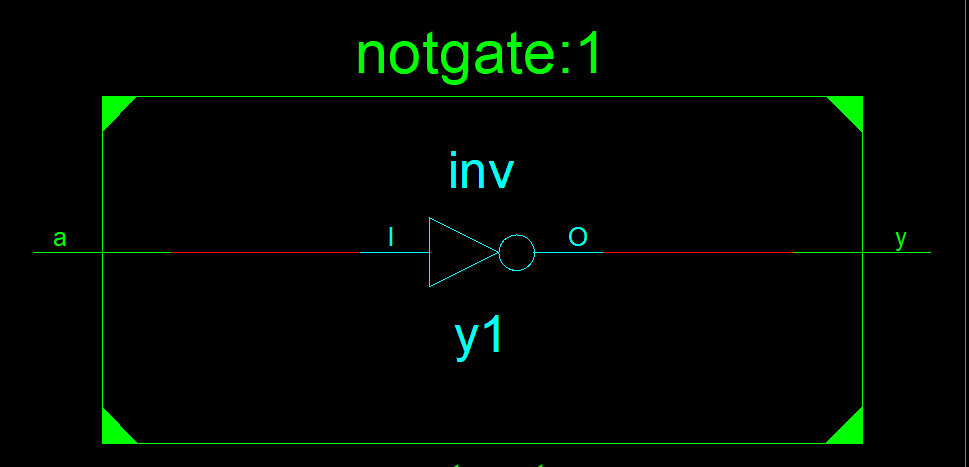
#100;

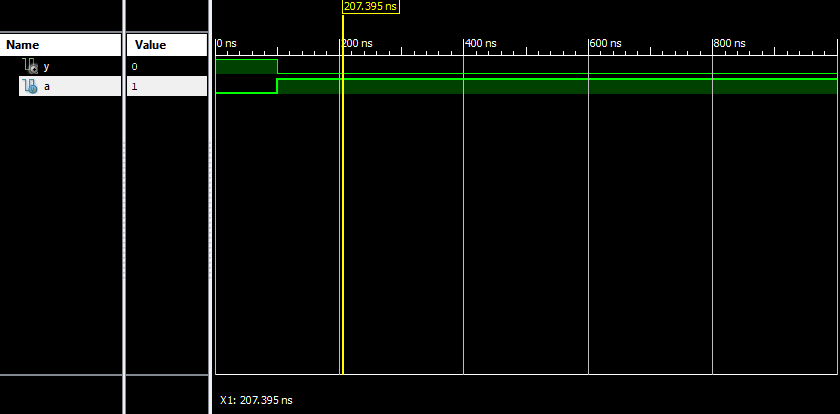
// Add stimulus here

end

endmodule

**RTL Schematic**

****

**Graph:**

**Verilog NOR Gate:**

**Gate level Coding:**

module norgate(

input a,

input b,

output x

);

nor(x,a,b);

endmodule

**Test Bench for NOR Gate:**

module norgtb;

// Inputs

reg a;

reg b;

// Outputs

wire x;

// Instantiate the Unit Under Test (UUT)

norgate uut (

.a(a),

.b(b),

.x(x)

);

initial begin

// Initialize Inputs

a = 0;

b = 0;

// Wait 100 ns for global reset to finish

#100

// Initialize Inputs

a = 0;

b = 1;

// Wait 100 ns for global reset to finish

#100;

// Initialize Inputs

a = 1;

b = 0;

// Wait 100 ns for global reset to finish

#100;

// Initialize Inputs

a = 1;

b = 1;

// Wait 100 ns for global reset to finish

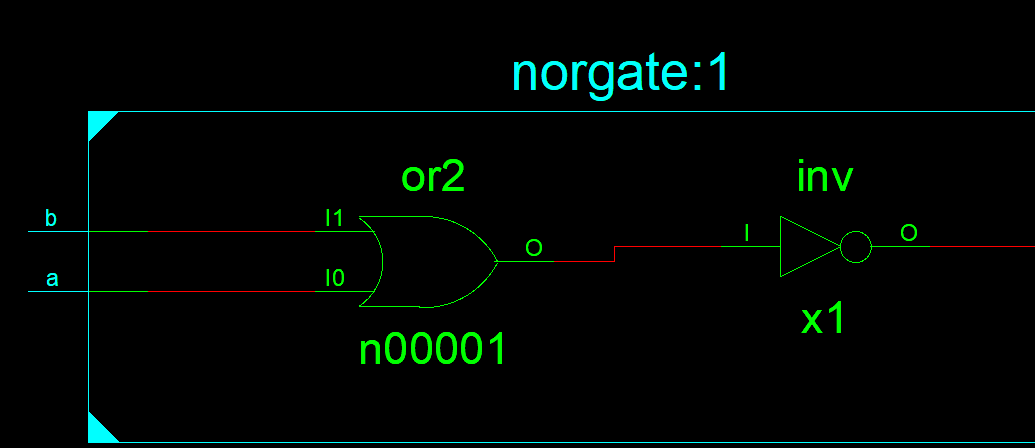
#100;

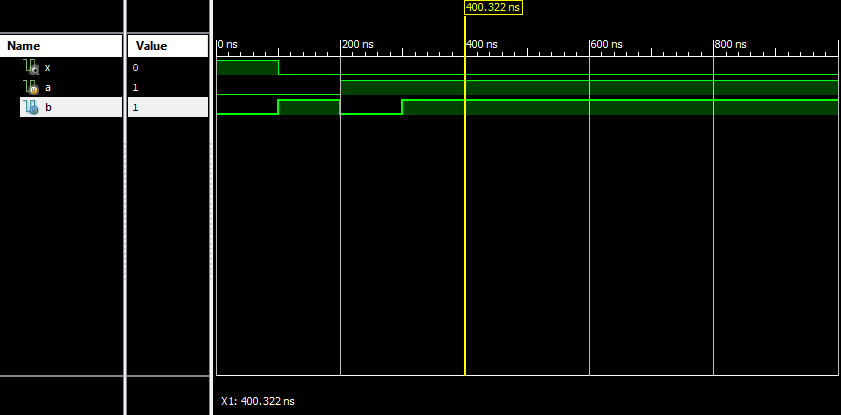
// Add stimulus here

end

endmodule

**RTL SCHEMATIC:**

****

**Graph:**

**Verilog code for NAND Gate:**

**Gate level coding:**

module nandgate(

input a,

input b,

output y

);

nand(y,a,b);

endmodule

**Test Bench for NAND Gate:**

module nandgtb;

// Inputs

reg a;

reg b;

// Outputs

wire y;

// Instantiate the Unit Under Test (UUT)

nandgate uut (

.a(a),

.b(b),

.y(y)

);

initial begin

// Initialize Inputs

a = 0;

b = 0;

// Wait 100 ns for global reset to finish

#100;

// Initialize Inputs

a = 0;

b = 1;

// Wait 100 ns for global reset to finish

#100;

// Initialize Inputs

a = 1;

b = 0;

// Wait 100 ns for global reset to finish

#100;

// Initialize Inputs

a = 1;

b = 1;

// Wait 100 ns for global reset to finish

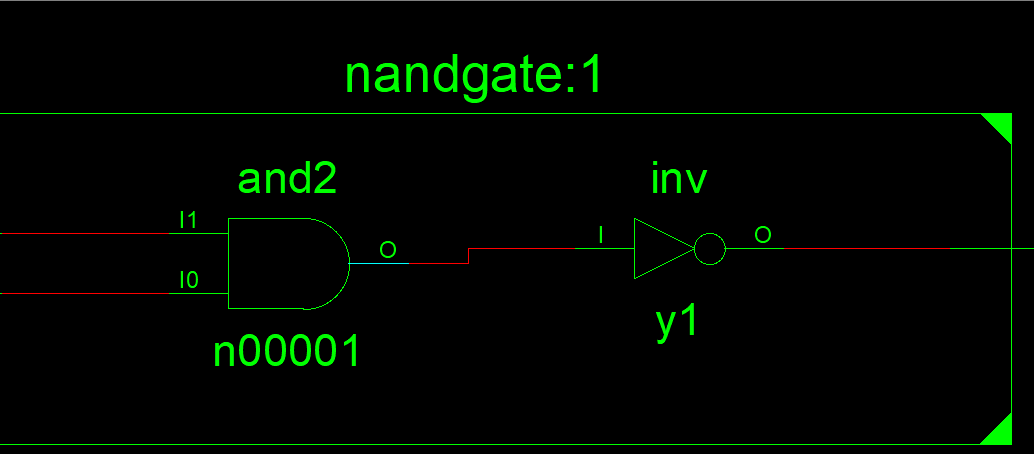
#100;

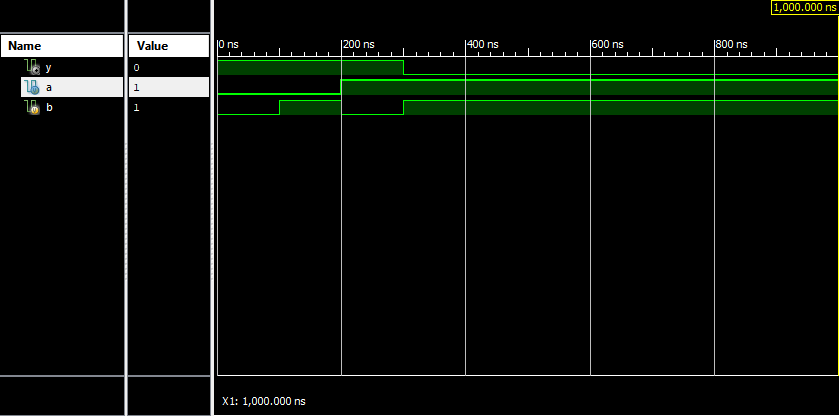
// Add stimulus here

end

endmodule

**RTL SCHEMATICS**

****

**Graph:**

**Verilog Code for XOR Gate:**

**Gate Level Modeling:**

module xorg(

input a,

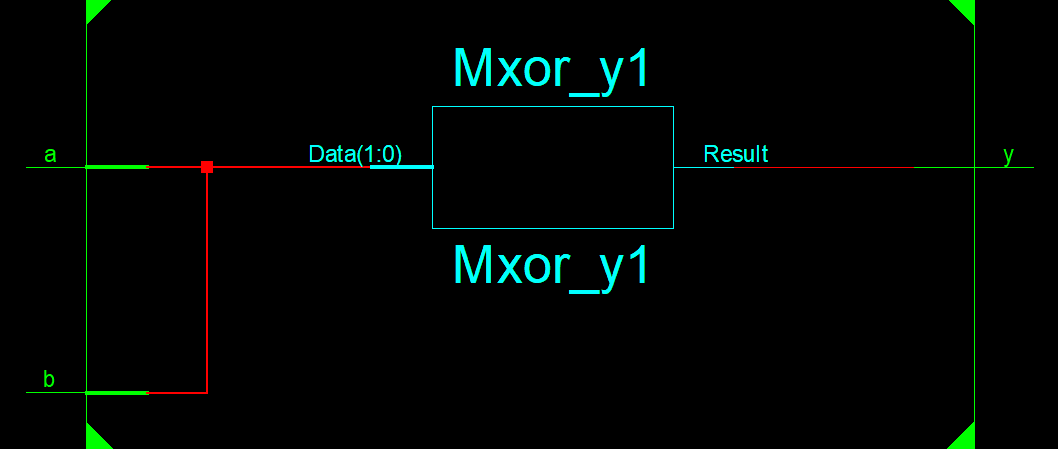
input b,

output y

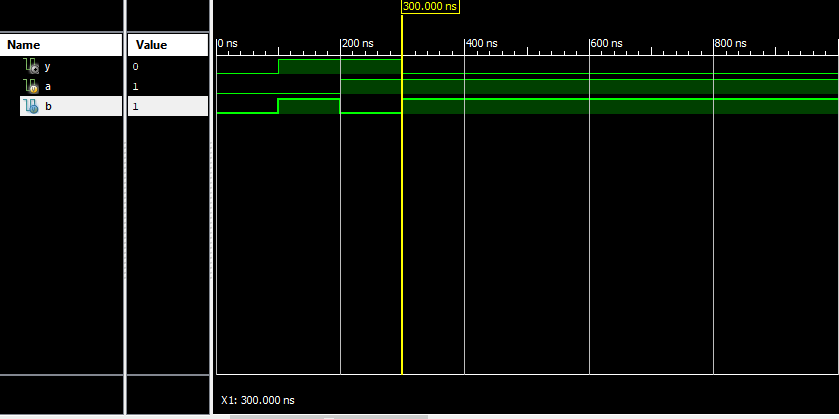
);

xor(y,a,b);

endmodule

**RTL Schematic:**

**Graph:**

****

**Verilog Code for XNOR code:**

**Gate level coding:**

module xnorg(

input a,

input b,

output y

);

xnor(y,a,b);

endmodule

**TestBench for XNOR Gate:**

module xorg;

// Inputs

reg a;

reg b;

// Outputs

wire y;

// Instantiate the Unit Under Test (UUT)

xnorg uut (

.a(a),

.b(b),

.y(y)

);

initial begin

// Initialize Inputs

a = 0;

b = 0;

// Wait 100 ns for global reset to finish

#100;

// Initialize Inputs

a = 0;

b = 1;

// Wait 100 ns for global reset to finish

#100

// Initialize Inputs

a = 1;

b = 0;

// Wait 100 ns for global reset to finish

#100;

// Initialize Inputs

a = 1;

b = 1;

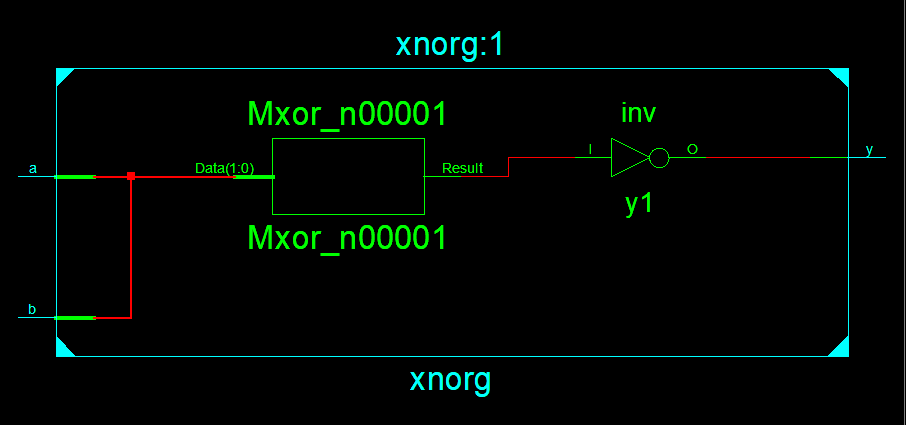
// Wait 100 ns for global reset to finish

#100;

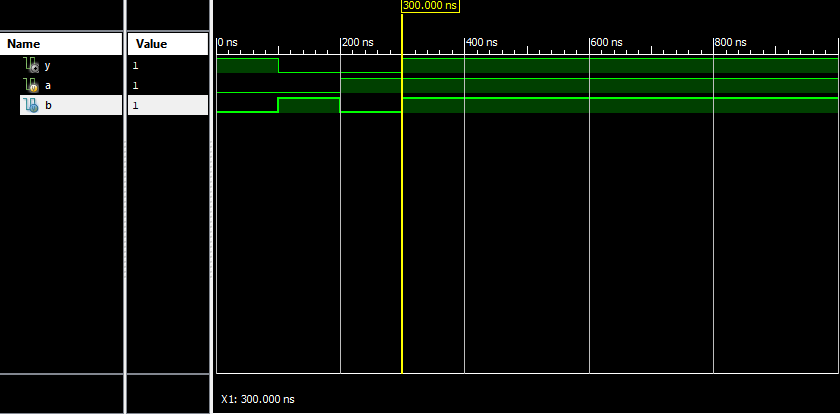
// Add stimulus here

end

endmodule

**RTL Schematics:**

**Graph:**

****

**Verilog code for Half Adder:**

**Gate Level Modeling:**

module halfAdder(

input A,

input B,

output Sum,

output Carry

);

xor(Sum,A,B);

and(Carry,A,B);

endmodule

**Test Bench for Half Adder:**

module halfaddertb;

// Inputs

reg A;

reg B;

// Outputs

wire Sum;

wire Carry;

// Instantiate the Unit Under Test (UUT)

halfAdder uut (

.A(A),

.B(B),

.Sum(Sum),

.Carry(Carry)

);

initial begin

// Initialize Inputs

A = 0;

B = 0;

// Wait 100 ns for global reset to finish

#100;

A = 0;

B = 1;

// Wait 100 ns for global reset to finish

#100;

A = 1;

B = 0;

// Wait 100 ns for global reset to finish

#100;

A = 1;

B = 1;

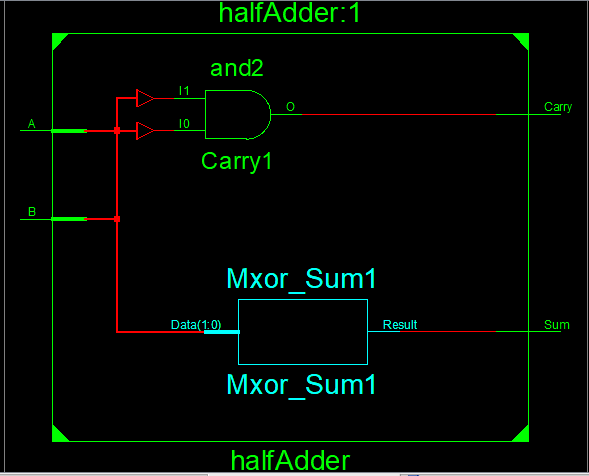
// Wait 100 ns for global reset to finish

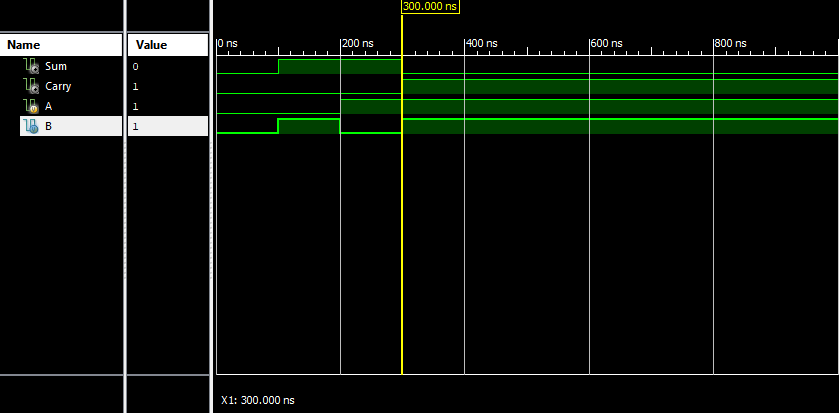
#100;

// Add stimulus here

end

endmodule

**RTL Schematic:**

**Graph:**

**Verilog Code for Full Adder:**

**Gate Level Coding:**

module fulladder(

input a,

input b,

input c,

output sum,

output carry

);

wire w1,w2,w3;

xor(w1,a,b);

and(w2,a,b);

xor(sum,w1,c);

and(w3,c,w1);

or(carry,w3,w2);

endmodule

**Test Bench For Full Adder:**

module fulladdertb;

// Inputs

reg a;

reg b;

reg c;

// Outputs

wire sum;

wire carry;

// Instantiate the Unit Under Test (UUT)

fulladder uut (

.a(a),

.b(b),

.c(c),

.sum(sum),

.carry(carry)

);

initial begin

// Initialize Inputs

a = 0;

b = 0;

c = 0;

// Wait 100 ns for global reset to finish

#100;

a = 0;

b = 0;

c = 1;

// Wait 100 ns for global reset to finish

#100;

a = 0;

b = 1;

c = 0;

// Wait 100 ns for global reset to finish

#100;

a = 0;

b = 1;

c = 1;

// Wait 100 ns for global reset to finish

#100;

a = 1;

b = 0;

c = 0;

// Wait 100 ns for global reset to finish

#100;

a = 1;

b = 0;

c = 1;

// Wait 100 ns for global reset to finish

#100;

a = 1;

b = 1;

c = 0;

// Wait 100 ns for global reset to finish

#100;

a = 1;

b = 1;

c = 1;

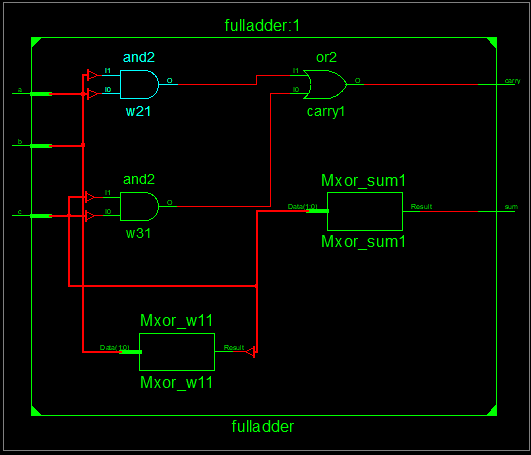
// Wait 100 ns for global reset to finish

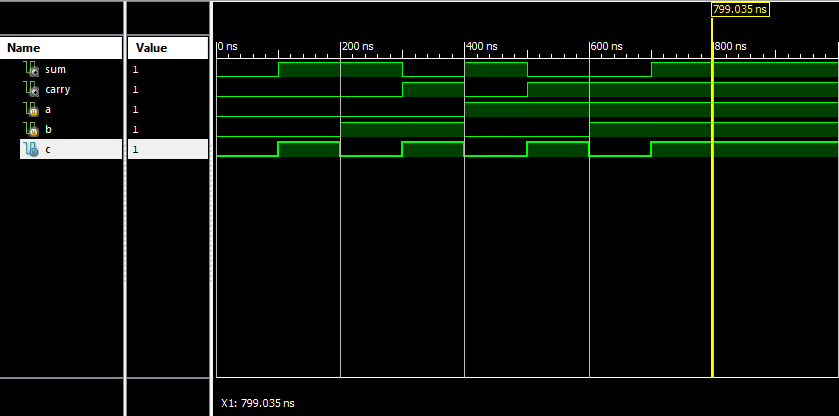
#100;

// Add stimulus here

end

endmodule

**RTL Schematics:**

**Graph:**

**Multiplexer:** A multiplexer is a device that selects one output from multiple inputs. It is also known as a data selector. Multiplexers are used in communication systems to increase the amount of data that can be sent over a network within a certain amount of time and bandwidth. It allows us to ‘squeeze’ multiple data lines into one data line.

The multiplexer (MUX) functions as a multi-input and single-output switch. The selection of the input is done using select lines. A MUX with **2^n** input lines have **n** select lines and is said to be a **2^n: 1** MUX with one output.

**Verilog Code for 2:1 Multiplexer:**

module multiplexer2\_1(

input d0,

input d1,

input s,

output y

);

wire w1,w2,sBar;

and(w1,d1,s),(w2,d0,sBar);

not(sBar,s);

or(y,w1,w2);

endmodule

**Test Bench for 2:1 Multiplexer:**

module multiplexer2\_1tb;

// Inputs

reg d0;

reg d1;

reg s;

// Outputs

wire y;

// Instantiate the Unit Under Test (UUT)

multiplexer2\_1 uut (

.d0(d0),

.d1(d1),

.s(s),

.y(y)

);

initial begin

// Initialize Inputs

d0 = 0;

d1 = 0;

s = 0

// Wait 100 ns for global reset to finish

#100;

// Add stimulus here

d0 = 0;

d1 = 1;

s = 0;

// Wait 100 ns for global reset to finish

#100;

// Add stimulus here

d0 = 1;

d1 = 0;

s = 1;

// Wait 100 ns for global reset to finish

#100;

d0 = 1;

d1 = 1;

s = 1;

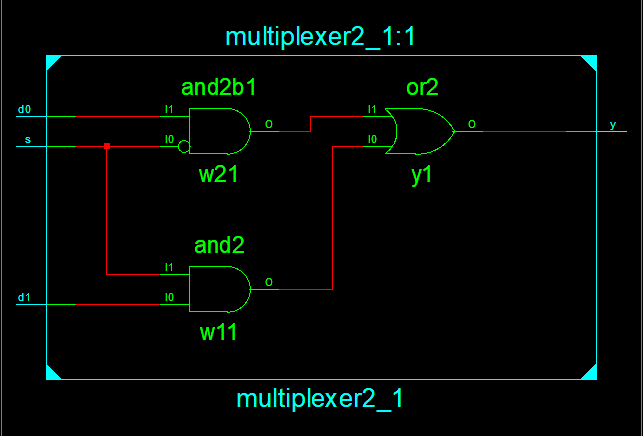
// Wait 100 ns for global reset to finish

#100;

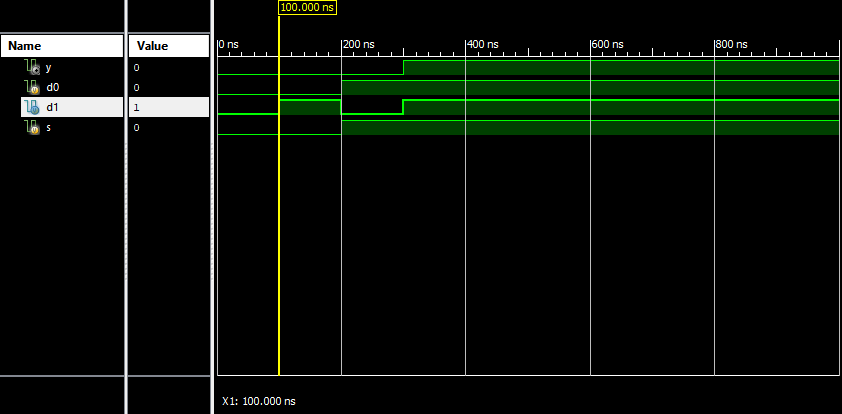
// Add stimulus here

end

endmodule

**RTL Schematics:**

**Graph:**



**Verilog Code for 4:1 Multiplexer:**

module mux4\_1(

input d0,

input d1,

input d2,

input d3,

input s0,

input s1,

output y

);

wire s0bar,s1bar,w1,w2,w3,w4;

not(s0bar,s0);

not(s1bar,s1);

and(w1,s0bar,s1bar,d0);

and(w2,s0,s1bar,d1);

and(w3,s0bar,s1,d2);

and(w4,s0,s1,d3);

or(y,w1,w2,w3,w4);

endmodule

**Test Bench for 4:1 Multiplexer:**

module mux4\_1tb1;

// Inputs

reg d0;

reg d1;

reg d2;

reg d3;

reg s0;

reg s1;

// Outputs

wire y;

// Instantiate the Unit Under Test (UUT)

mux4\_1 uut (

.d0(d0),

.d1(d1),

.d2(d2),

.d3(d3),

.s0(s0),

.s1(s1),

.y(y)

);

initial begin

// Initialize Inputs

d0 = 0;

d1 = 0;

d2 = 0;

d3 = 0;

s0 = 0;

s1 = 0;

// Wait 100 ns for global reset to finish

#100;

// Add stimulus here

// Initialize Inputs

d0 = 1;

d1 = 0;

d2 = 0;

d3 = 0;

s0 = 0;

s1 = 0;

// Wait 100 ns for global reset to finish

#100;

d0 = 0;

d1 = 0;

d2 = 0;

d3 = 0;

s0 = 1;

s1 = 0;

// Wait 100 ns for global reset to finish

#100

// Add stimulus here

// Initialize Inputs

d0 = 0;

d1 = 1;

d2 = 0;

d3 = 0;

s0 = 1;

s1 = 0;

#100;

d0 = 0;

d1 = 0;

d2 = 0;

d3 = 0;

s0 = 0;

s1 = 1;

// Wait 100 ns for global reset to finish

#100;

// Add stimulus here

// Initialize Inputs

d0 = 0;

d1 = 0;

d2 = 1;

d3 = 0;

s0 = 0;

s1 = 1;

#100;

d0 = 0;

d1 = 0;

d2 = 0;

d3 = 0;

s0 = 1;

s1 = 1;

// Wait 100 ns for global reset to finish

#100;

// Add stimulus here

// Initialize Inputs

d0 = 0;

d1 = 0;

d2 = 0;

d3 = 1;

s0 = 1;

s1 = 1;

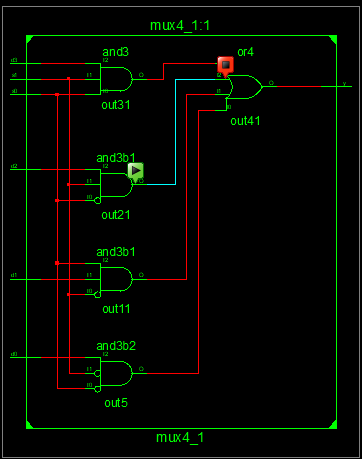
// Wait 100 ns for global reset to finish

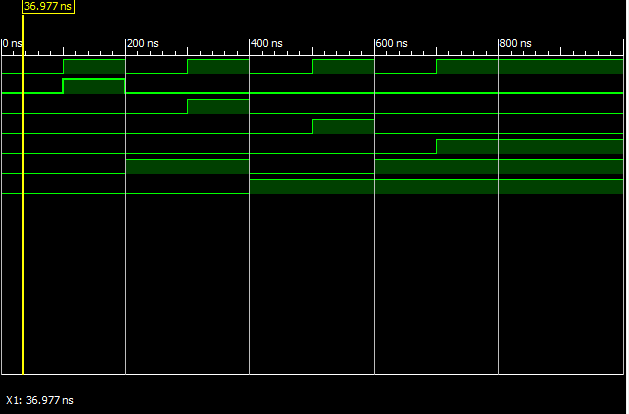
#100;

end

endmodule

**RTL Schematics:**

****

**Graph:**

**Verilog code for 8:1 Multiplexer:**

**Gate level coding:**

module mux8\_1(

input d0,

input d1,

input d2,

input d3,

input d4,

input d5,

input d6,

input d7,

input s0,

input s1,

input s2,

output out

);

wire T1, T2, T3, T4, T5, T6, T7, T8, T9, T10, T11;

not(T1, S0);

not(T2, S1);

not(T3, S2);

and(T4, D0, T1, T2, T3), (T5, D1, S0, T2, T3);

and(T6, D2, T1, S1, T3), (T7, D3, S0, S1, T3);

and(T8, D4, T1, T2, S2), (T9, D5, S0, T2, S2);

and(T10, D6, T1, S1, S2), (T11, D7, S0, S1, S2);

or(out, T4, T5, T6, T7, T8, T9, T10, T11);

endmodule

**Test Bench for 8:1 Multiplexer:**

module mux8\_1tb;

// Inputs

reg d0;

reg d1;

reg d2;

reg d3;

reg d4;

reg d5;

reg d6;

reg d7;

reg s0;

reg s1;

reg s2;

// Outputs

wire out;

// Instantiate the Unit Under Test (UUT)

mux8\_1 uut (

.d0(d0),

.d1(d1),

.d2(d2),

.d3(d3),

.d4(d4),

.d5(d5),

.d6(d6),

.d7(d7),

.s0(s0),

.s1(s1),

.s2(s2),

.out(out)

);

initial begin

// Initialize Inputs

d0 = 1'b0;

d1 = 1'b0;

d2 = 1'b0;

d3 = 1'b0;

d4 = 1'b0;

d5 = 1'b0;

d6 = 1'b0;

d7 = 1'b0;

s0 = 1'b0;

s1 = 1'b0;

s2 = 1'b0;

// Wait 100 ns for global reset to finish

#500;

// Add stimulus here

end

always #1 d0=~d0;

always #2 d1=~d1;

always #3 d2=~d2;

always #4 d3=~d3;

always #5 d4=~d4;

always #6 d5=~d5;

always #7 d6=~d6;

always #8 d7=~d7;

always #9 s0=~d0;

always #10 s1=~s1;

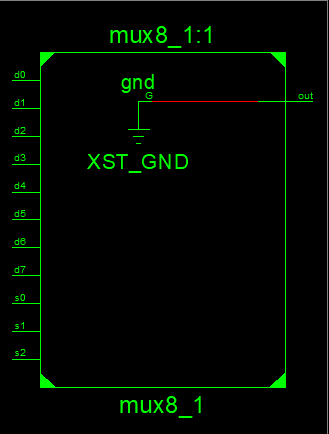
always #11 s2=~s2;

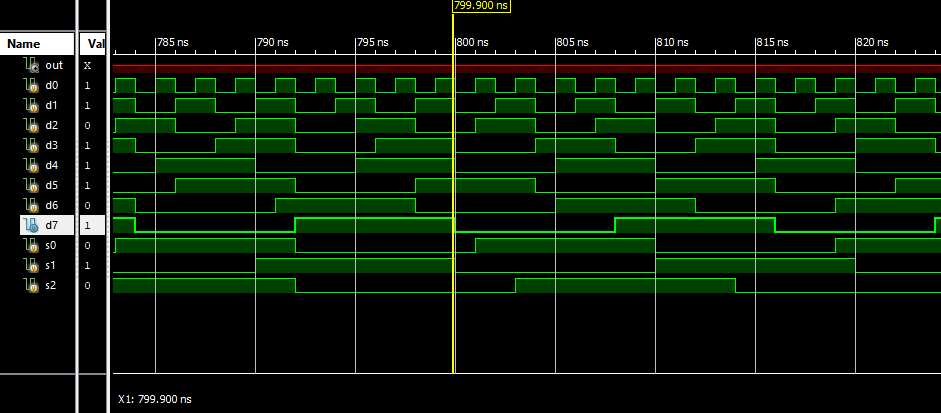
always@(d0 or d1 or d2 or d3 or d4 or d5 or d6 or d7 or s0 or s1 or s2)

$monitor("At time = %t, Output = %d", $time, out);

Endmodule

**RTL schematics:**

****

**Graph: **

**Verilog code for 1:2 Demultiplexers:**

**Gate level Modeling:**

module demux1\_2(

input s,

input d,

output y0,

output y1

);

wire sn;

not(sn,s);

and(y0,sn,d);

and(y1,s,d);

endmodule

**Test Bench for 1\_2 Demultiplexer:**

module demux2\_1tb;

// Inputs

reg s;

reg d;

// Outputs

wire y0;

wire y1;

// Instantiate the Unit Under Test (UUT)

demux1\_2 uut (

.s(s),

.d(d),

.y0(y0),

.y1(y1)

);

initial begin

// Initialize Inputs

s = 0;

d = 0;

// Wait 100 ns for global reset to finish

#100;

// Initialize Inputs

s = 0;

d = 1;

// Wait 100 ns for global reset to finish

#100;

// Initialize Inputs

s = 1;

d = 0;

// Wait 100 ns for global reset to finish

#100;

// Initialize Inputs

s = 1;

d = 1;

// Wait 100 ns for global reset to finish

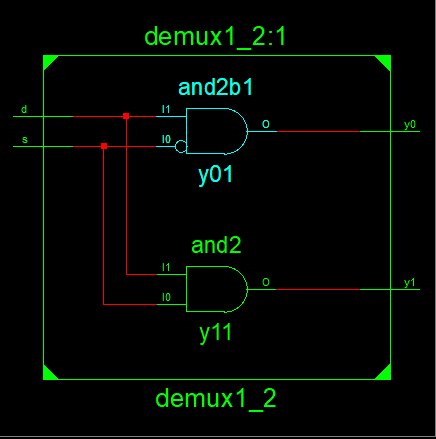
#100;

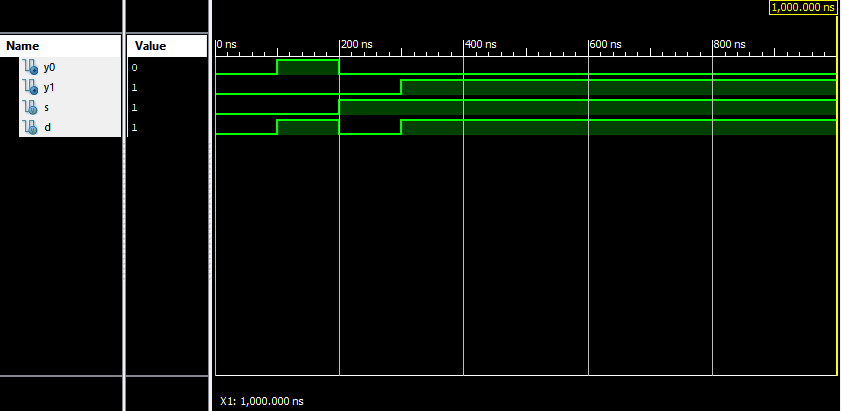
// Add stimulus here

end

endmodule

**RTL Schematics:**

****

**Graph: **

**Verilog Code for 1-4 demultiplexer:**

**Gate level Modeling:**

module demux1\_4(

input d,

input s0,

input s1,

output y0,

output y1,

output y2,

output y3

);

wire sn0,sn1;

not(sn0,s0);

not(sn1,s1);

and(y0,d,sn0,sn1);

and(y1,d,s0,sn1);

and(y2,d,sn0,s1);

and(y3,d,s0,s1);

endmodule

**Test-Bench for 1-4 Demultiplexer:**

module demux1\_4tb;

// Inputs

reg d;

reg s0;

reg s1;

// Outputs

wire y0;

wire y1;

wire y2;

wire y3;

// Instantiate the Unit Under Test (UUT)

demux1\_4 uut (

.d(d),

.s0(s0),

.s1(s1),

.y0(y0),

.y1(y1),

.y2(y2),

.y3(y3)

);

initial begin

// Initialize Inputs

d = 1;

s0 = 0;

s1 = 0;

// Wait 100 ns for global reset to finish

#100;

d = 1;

s0 = 0;

s1 = 1;

// Wait 100 ns for global reset to finish

#100;

d = 1;

s0 = 1;

s1 = 0;

// Wait 100 ns for global reset to finish

#100;

d = 1;

s0 = 1;

s1 = 1;

// Wait 100 ns for global reset to finish

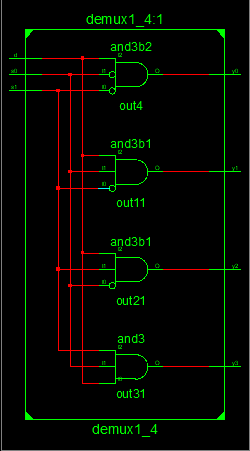
#100;

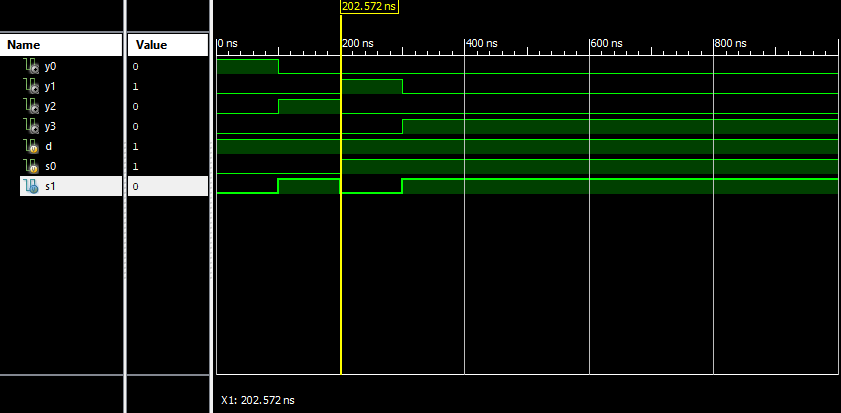
// Add stimulus here

end

endmodule

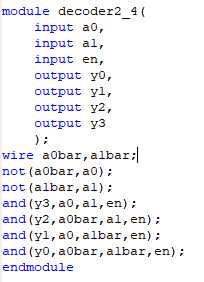
**RTL Schematics:**

****

**Graph: **

**Verilog Code for 2-4 Decoder:**

**Gate level coding:**

****

**Test-Bench for 2-4 Decoder:**

module decoder2\_4tb;

// Inputs

reg a0;

reg a1;

reg en;

// Outputs

wire y0;

wire y1;

wire y2;

wire y3;

// Instantiate the Unit Under Test (UUT)

decoder2\_4 uut (

.a0(a0),

.a1(a1),

.en(en),

.y0(y0),

.y1(y1),

.y2(y2),

.y3(y3)

);

initial begin

// Initialize Inputs

a0 = 0;

a1 = 0;

en = 1;

// Wait 100 ns for global reset to finish

#100;

a0 = 0;

a1 = 1;

en = 1;

// Wait 100 ns for global reset to finish

#100;

a0 = 1;

a1 = 0;

en = 1;

// Wait 100 ns for global reset to finish

#100;

a0 = 1;

a1 = 1;

en = 1;

// Wait 100 ns for global reset to finish

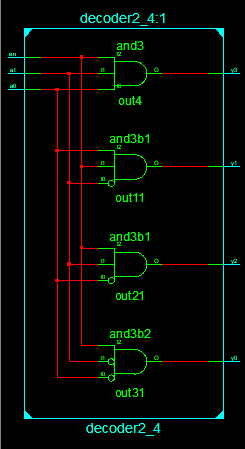
#100;

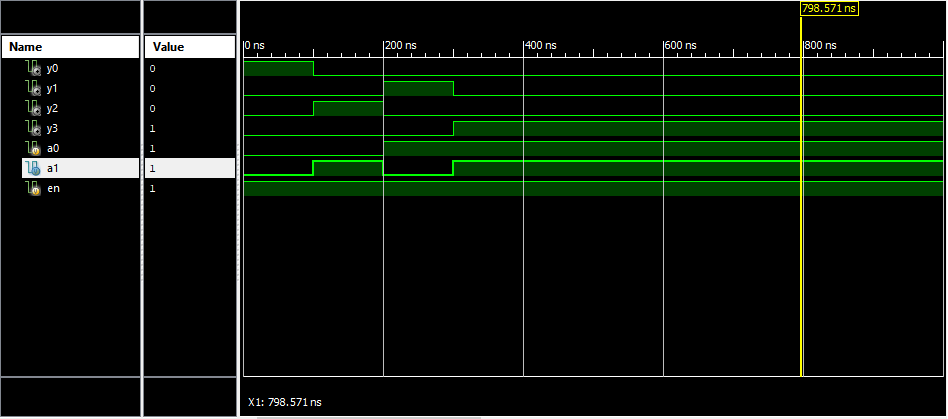
// Add stimulus here

end

endmodule

**RTL Schematics:**

****

**Graph: **

**Verilog Code for 3\_8 Decoder:**

**Gate level coding:**

module decoder3\_8(

input x,

input y,

input z,

output d0,

output d1,

output d2,

output d3,

output d4,

output d5,

output d6,

output d7

);

wire xbar,ybar,zbar;

not(xbar,x);

not(ybar,y);

not(zbar,z);

and(d0,xbar,ybar,zbar),(d1,xbar,ybar,z),(d2,xbar,y,zbar),(d3,xbar,y,z),(d4,x,ybar,zbar),(d5,x,ybar,z),(d6,x,y,zbar),(d7,x,y,z);

endmodule

**Test Bench for 3\_8 decoder:**

module decoder3\_8tb;

// Inputs

reg x;

reg y;

reg z;

// Outputs

wire d0;

wire d1;

wire d2;

wire d3;

wire d4;

wire d5;

wire d6;

wire d7;

// Instantiate the Unit Under Test (UUT)

decoder3\_8 uut (

.x(x),

.y(y),

.z(z),

.d0(d0),

.d1(d1),

.d2(d2),

.d3(d3),

.d4(d4),

.d5(d5),

.d6(d6),

.d7(d7)

);

initial begin

// Initialize Inputs

x = 0;

y = 0;

z = 0;

// Wait 100 ns for global reset to finish

#100;

x = 0;

y = 0;

z = 1;

// Wait 100 ns for global reset to finish

#100;

x = 0;

y = 1;

z = 0;

// Wait 100 ns for global reset to finish

#100;

x = 0;

y = 1;

z = 1;

// Wait 100 ns for global reset to finish

#100;

x = 1;

y = 0;

z = 0;

// Wait 100 ns for global reset to finish

#100;

x = 1;

y = 0;

z = 1;

// Wait 100 ns for global reset to finish

#100;

x = 1;

y = 1;

z = 0;

// Wait 100 ns for global reset to finish

#100;

x = 1;

y = 1;

z = 1;

// Wait 100 ns for global reset to finish

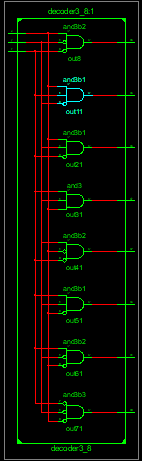
#100;

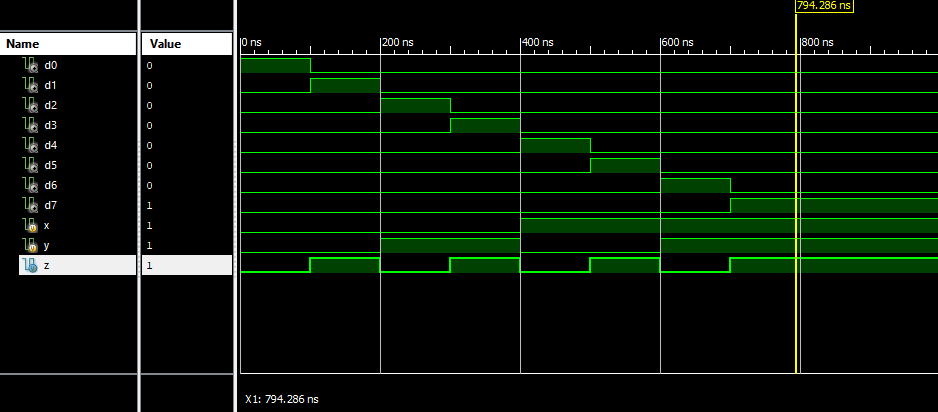
// Add stimulus here

end

endmodule

**RTL Schematics:**

****

**Graph: **

**Verilog code for SR Flip Flop:**

**Text

Description automatically generated**

**Test Bench for SR Flip Flop:**

module srfftb;

// Inputs

reg s;

reg r;

reg clk;

// Outputs

wire q;

wire qbar;

// Instantiate the Unit Under Test (UUT)

srff uut (

.s(s),

.r(r),

.clk(clk),

.q(q),

.qbar(qbar)

);

initial begin

// Initialize Inputs

s = 0;

r = 0;

clk = 0;

// Wait 100 ns for global reset to finish

#100;

s = 0;

r = 0;

clk = 1;

// Wait 100 ns for global reset to finish

#100;

s = 0;

r = 1;

clk = 0;

// Wait 100 ns for global reset to finish

#100;

s = 0;

r = 1;

clk = 1;

// Wait 100 ns for global reset to finish

#100;

s = 1;

r = 0;

clk = 0;

// Wait 100 ns for global reset to finish

#100;

s = 1;

r = 0;

clk = 1;

// Wait 100 ns for global reset to finish

#100;

s = 1;

r = 1;

clk = 0;

// Wait 100 ns for global reset to finish

#100;

s = 1;

r = 1;

clk = 1;

// Wait 100 ns for global reset to finish

#100;

// Add stimulus here

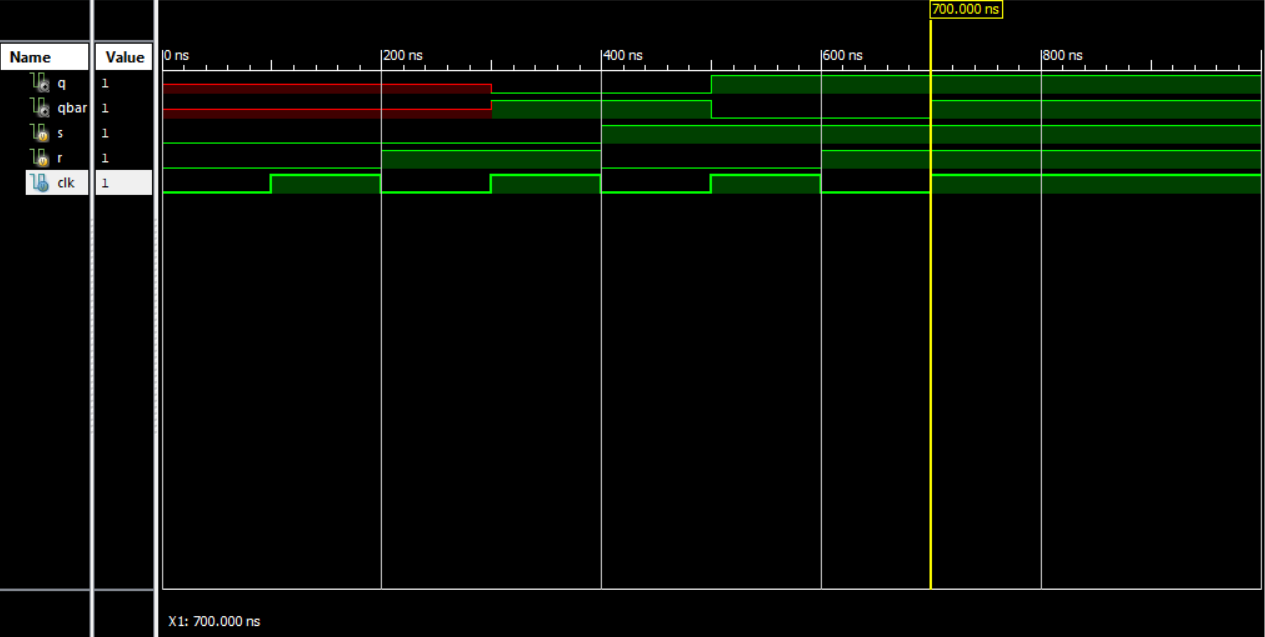
end

endmodule

**RTL Schematics:**

**Diagram

Description automatically generated**

**Graph:**

**Verilog code for D Flip Flop:**

module d\_flipflpo2v(

input d,

input clk,

output q,

output qbar

);

wire x,y,dbar;

not(dbar,d);

nand(x,clk,d);

nand(y,clk,dbar);

nand(q,qbar,y);

nand(qbar,q,x);

endmodule

**Test Bench for D-flip flop:**

module d\_flipflpotb;

// Inputs

reg d;

reg clk;

// Outputs

wire q;

wire qbar;

// Instantiate the Unit Under Test (UUT)

d\_flipflpo2v uut (

.d(d),

.clk(clk),

.q(q),

.qbar(qbar)

);

initial begin

// Initialize Inputs

d = 0;

clk = 0;

// Wait 100 ns for global reset to finish

#100;

d = 0;

clk = 1;

// Wait 100 ns for global reset to finish

#100;

d = 1;

clk = 0;

// Wait 100 ns for global reset to finish

#100;

d = 1;

clk = 1;

// Wait 100 ns for global reset to finish

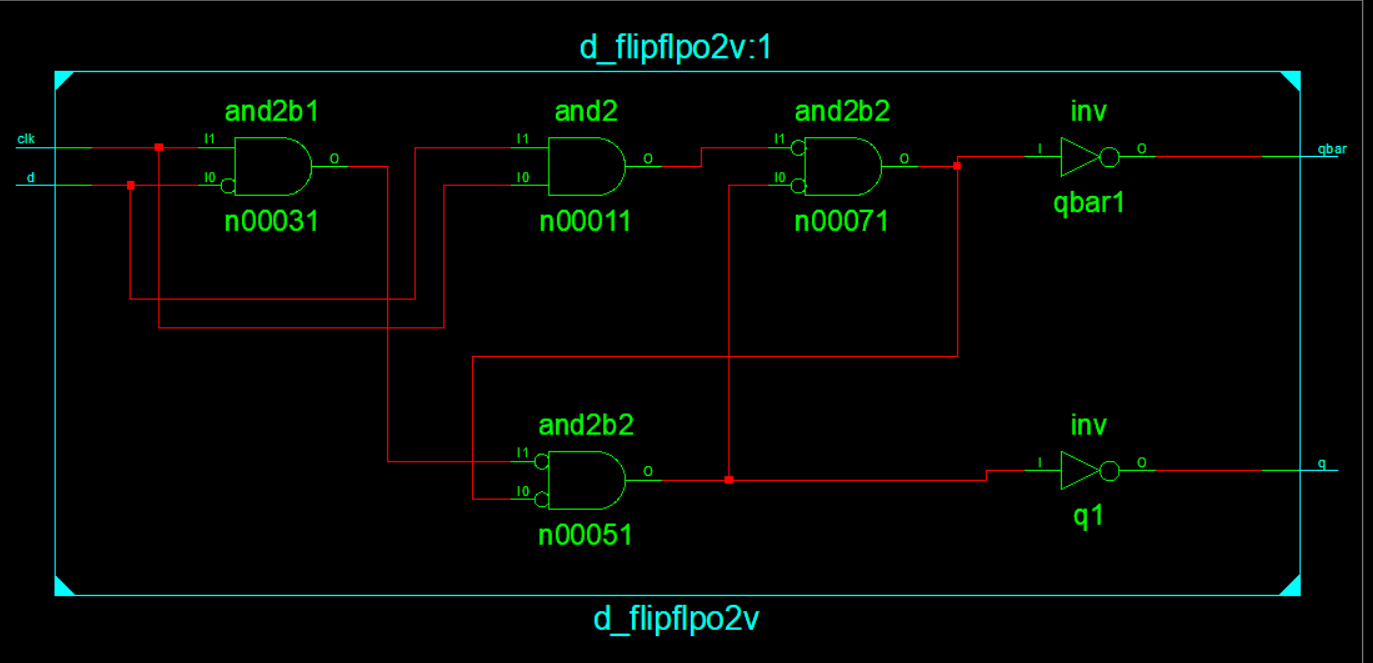
#100;

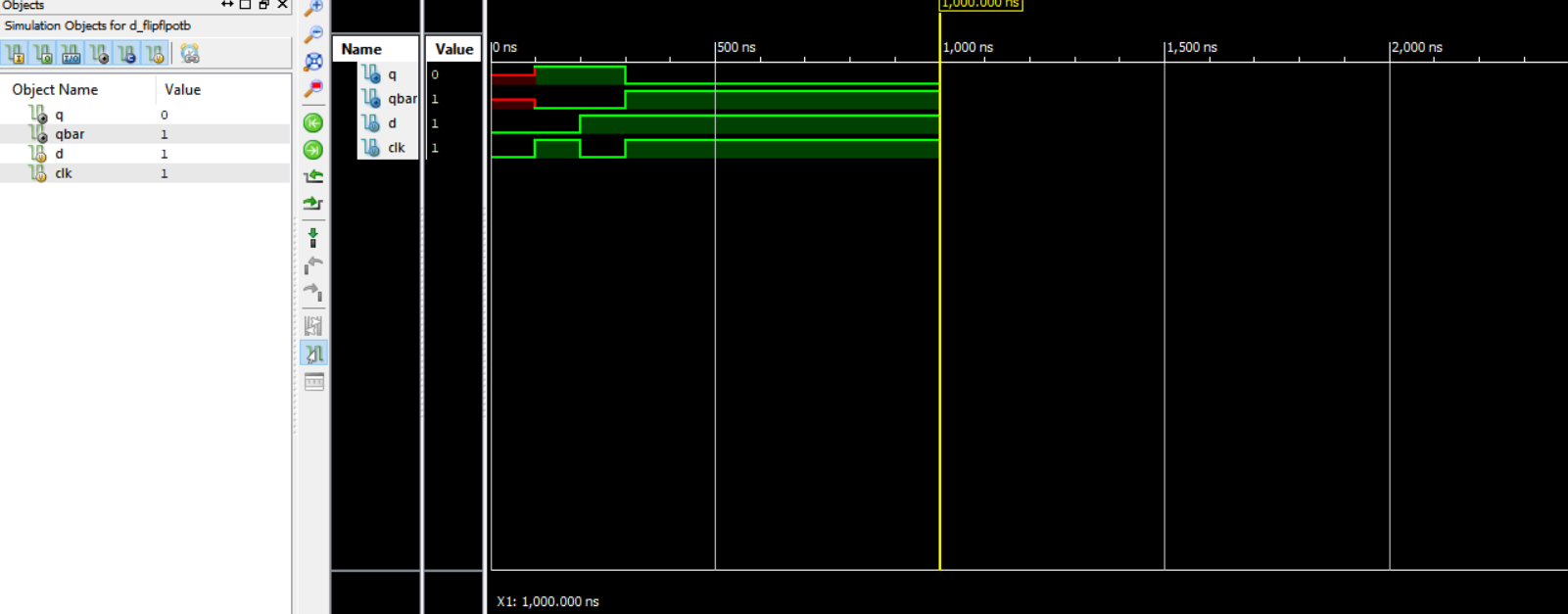
// Add stimulus here

end

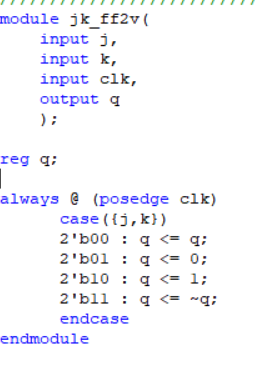
endmodule

**RTL schematics of D-Flip-Flop:**

****

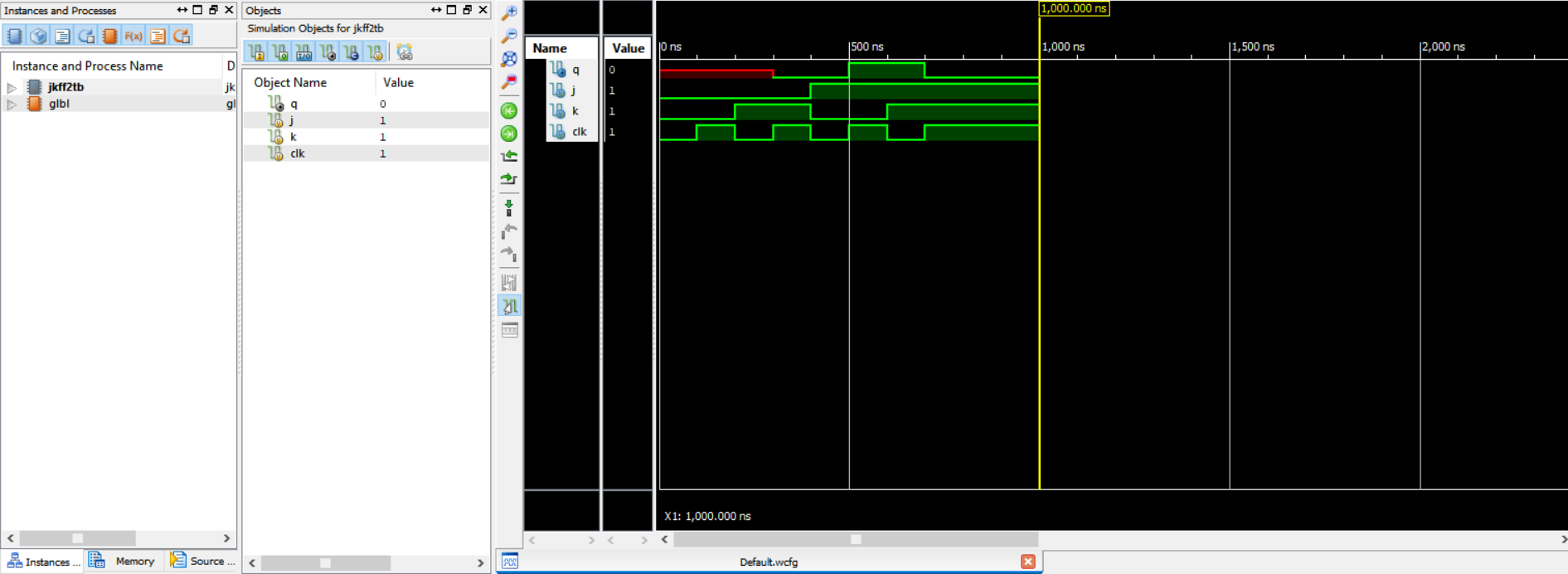
**Graph:**

**Verilog code for JK-Flip-Flop:**

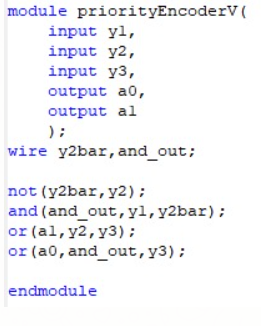
****

**RTL Schematics:**

**Graph:**

****

**Verilog Code for Priority Encoder:**

****

**Test Bench For Priority Encoder:**

module priorityEncoderTB;

// Inputs

reg y1;

reg y2;

reg y3;

// Outputs

wire a0;

wire a1;

// Instantiate the Unit Under Test (UUT)

priorityEncoderV uut (

.y1(y1),

.y2(y2),

.y3(y3),

.a0(a0),

.a1(a1)

);

initial begin

// Initialize Inputs

y1 = 0;

y2 = 0;

y3 = 0;

// Wait 100 ns for global reset to finish

#100;

// Initialize Inputs

y1 = 0;

y2 = 0;

y3 = 1;

// Wait 100 ns for global reset to finish

#100;

// Initialize Inputs

y1 = 0;

y2 = 1;

y3 = 0;

// Wait 100 ns for global reset to finish

#100;

// Initialize Inputs

y1 = 0;

y2 = 1;

y3 = 1;

// Wait 100 ns for global reset to finish

#100;

// Initialize Inputs

y1 = 1;

y2 = 0;

y3 = 0;

// Wait 100 ns for global reset to finish

#100;

// Initialize Inputs

y1 = 1;

y2 = 0;

y3 = 1;

// Wait 100 ns for global reset to finish

#100;

// Initialize Inputs

y1 = 1;

y2 = 1;

y3 = 0;

// Wait 100 ns for global reset to finish

#100;

// Initialize Inputs

y1 = 1;

y2 = 1;

y3 = 1;

// Wait 100 ns for global reset to finish

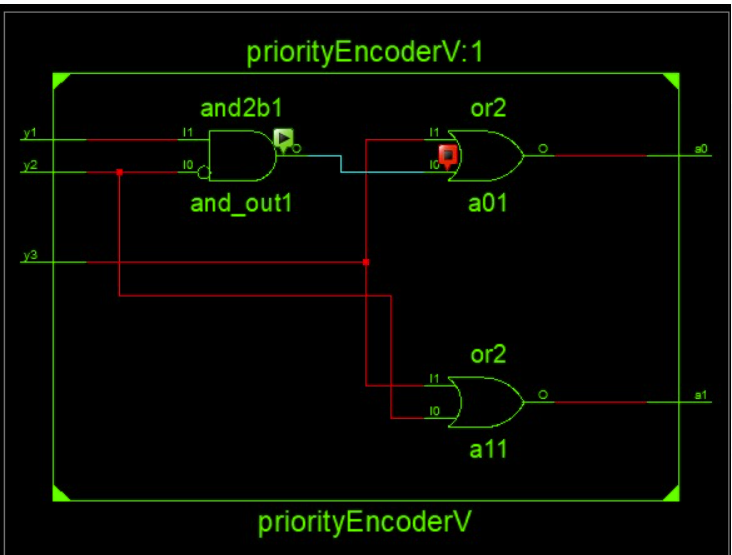
#100;

// Add stimulus here

end

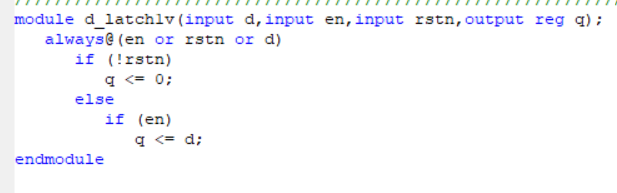
endmodule

**RTL Schematics:**

****

**Graph:** ****

**Verilog code for D\_latch:**



**Test-Bench for D\_Latch:**

module d\_latch1tb;

// Inputs

reg d;

reg en;

reg rstn;

// Outputs

wire q;

// Instantiate the Unit Under Test (UUT)

d\_latch1v uut (

.d(d),

.en(en),

.rstn(rstn),

.q(q)

);

initial begin

// Initialize Inputs

d = 0;

en = 0;

rstn = 0;

// Wait 100 ns for global reset to finish

#100;

d = 0;

en = 0;

rstn = 1;

// Wait 100 ns for global reset to finish

#100;

d = 0;

en = 1;

rstn = 0;

// Wait 100 ns for global reset to finish

#100;

d = 0;

en = 1;

rstn = 1;

// Wait 100 ns for global reset to finish

#100;

d = 1;

en = 0;

rstn = 0;

// Wait 100 ns for global reset to finish

#100;

d = 1;

en = 0;

rstn = 1;

// Wait 100 ns for global reset to finish

#100;

d = 1;

en = 1;

rstn = 0;

// Wait 100 ns for global reset to finish

#100;

d = 1;

en = 1;

rstn = 1;

// Wait 100 ns for global reset to finish

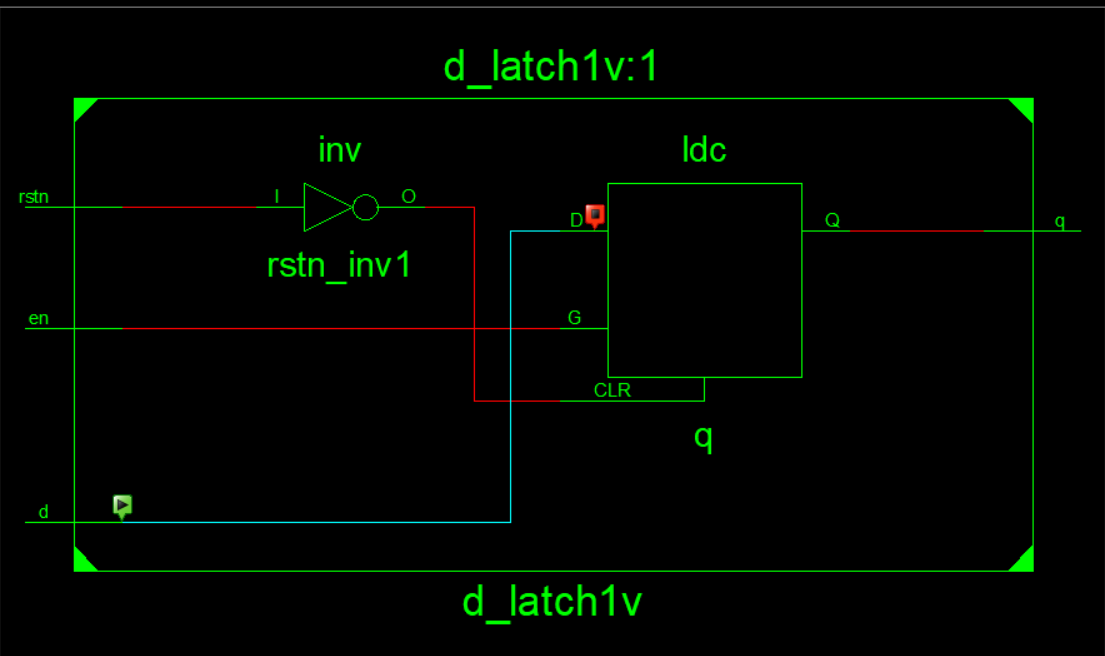
#100;

// Add stimulus here

end

endmodule

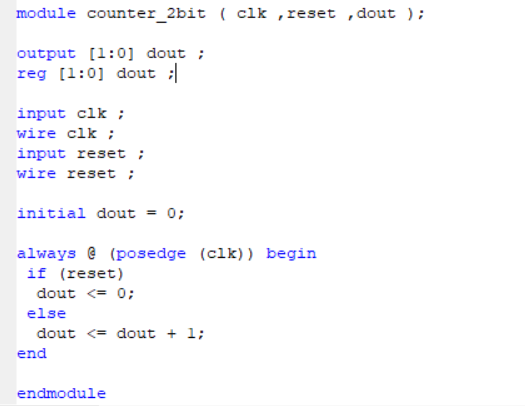
**RTL Schematics:**

****

**Graph:**

****

**Verilog code for 2\_Bit Counter:**

****

**Test Bench For 2\_Bit Counter:**

module counter\_2bit\_tb;

// Inputs

reg clk;

reg reset;

// Outputs

wire [1:0] dout;

// Instantiate the Unit Under Test (UUT)

counter\_2bit uut (

.clk(clk),

.reset(reset),

.dout(dout)

);

initial begin

// Initialize Inputs

clk = 0;

reset = 0;

// Wait 100 ns for global reset to finish

#100;

clk = 0;

reset = 1;

// Wait 100 ns for global reset to finish

#100;

clk = 1;

reset = 0;

// Wait 100 ns for global reset to finish

#100;

clk = 1;

reset = 1;

// Wait 100 ns for global reset to finish

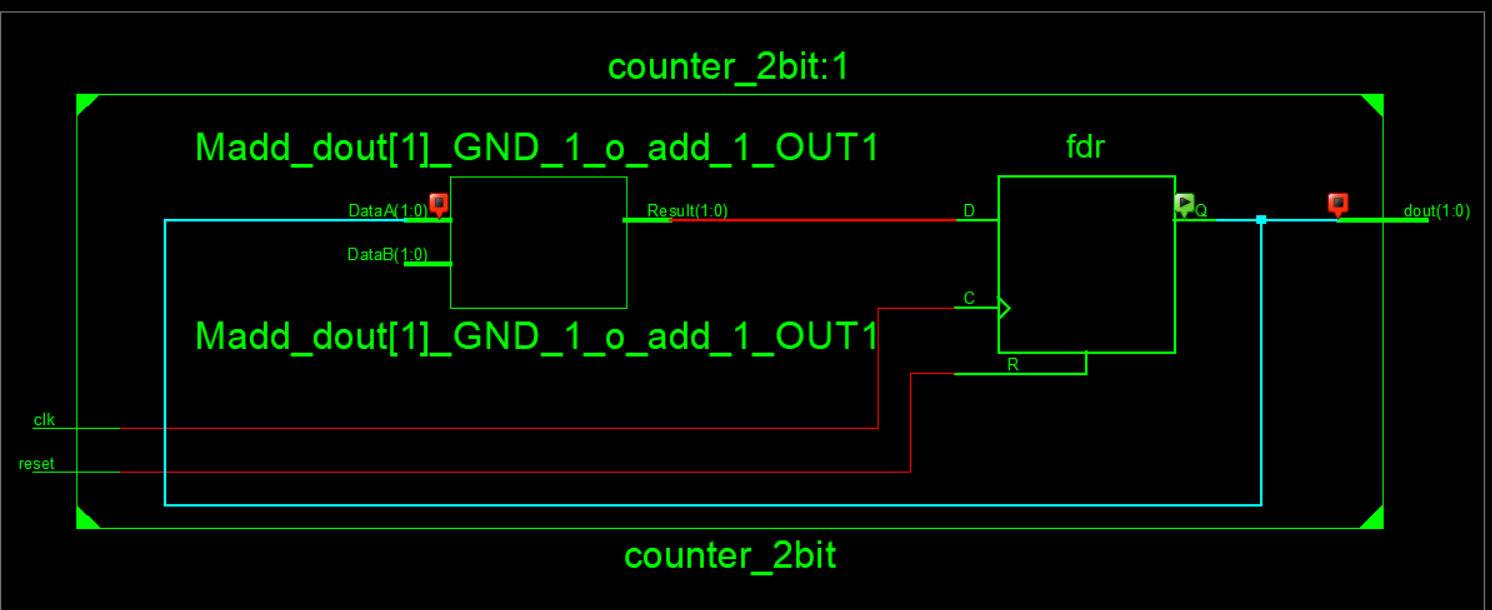
#100;

// Add stimulus here

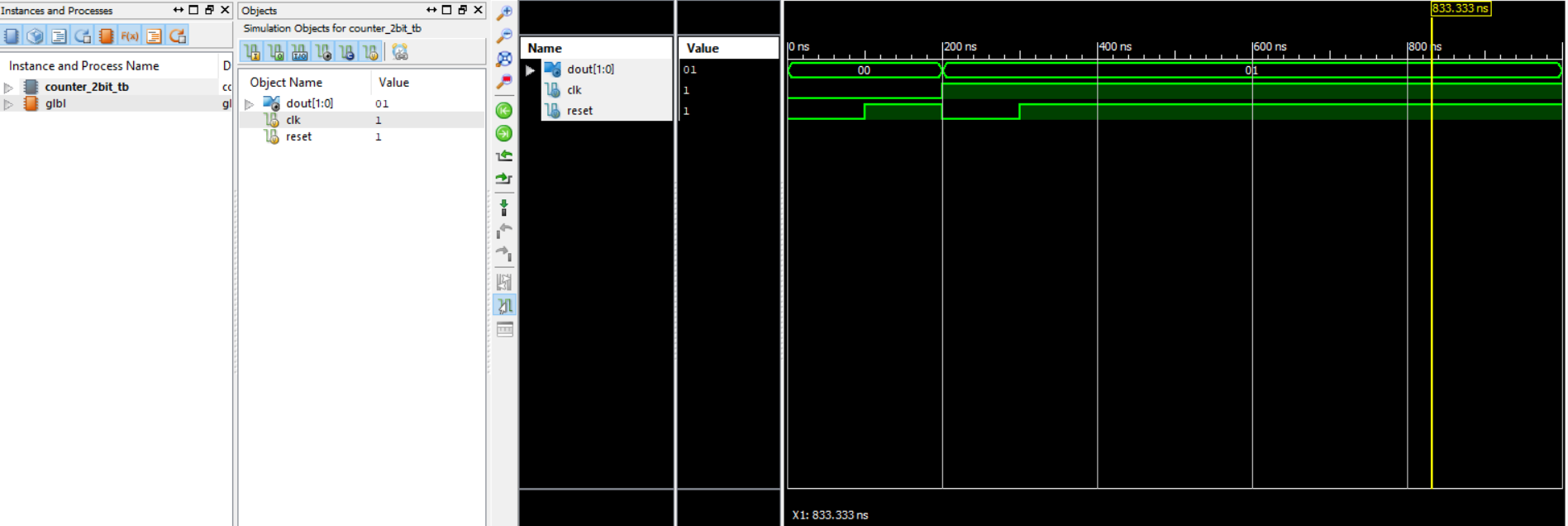
end

endmodule

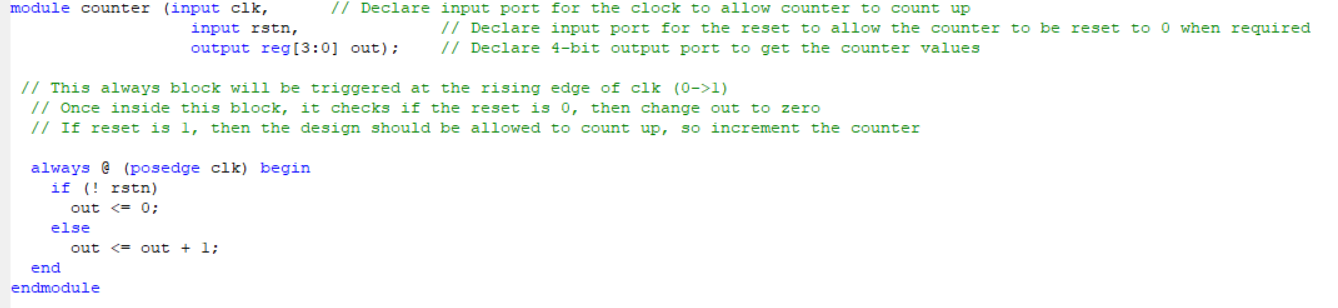
**RTL Schematics:**

****

**Graph:**

****

**4-Bit Counter Verilog code:**

****

**Test-Bench:**

module counter\_tb;

// Inputs

reg clk;

reg rstn;

// Outputs

wire [3:0] out;

// Instantiate the Unit Under Test (UUT)

counter uut (

.clk(clk),

.rstn(rstn),

.out(out)

);

initial begin

// Initialize Inputs

clk = 0;

rstn = 0;

// Wait 100 ns for global reset to finish

#100;

clk = 0;

rstn = 1;

// Wait 100 ns for global reset to finish

#100;

clk = 1;

rstn = 0;

// Wait 100 ns for global reset to finish

#100;

clk = 1;

rstn = 1;

// Wait 100 ns for global reset to finish

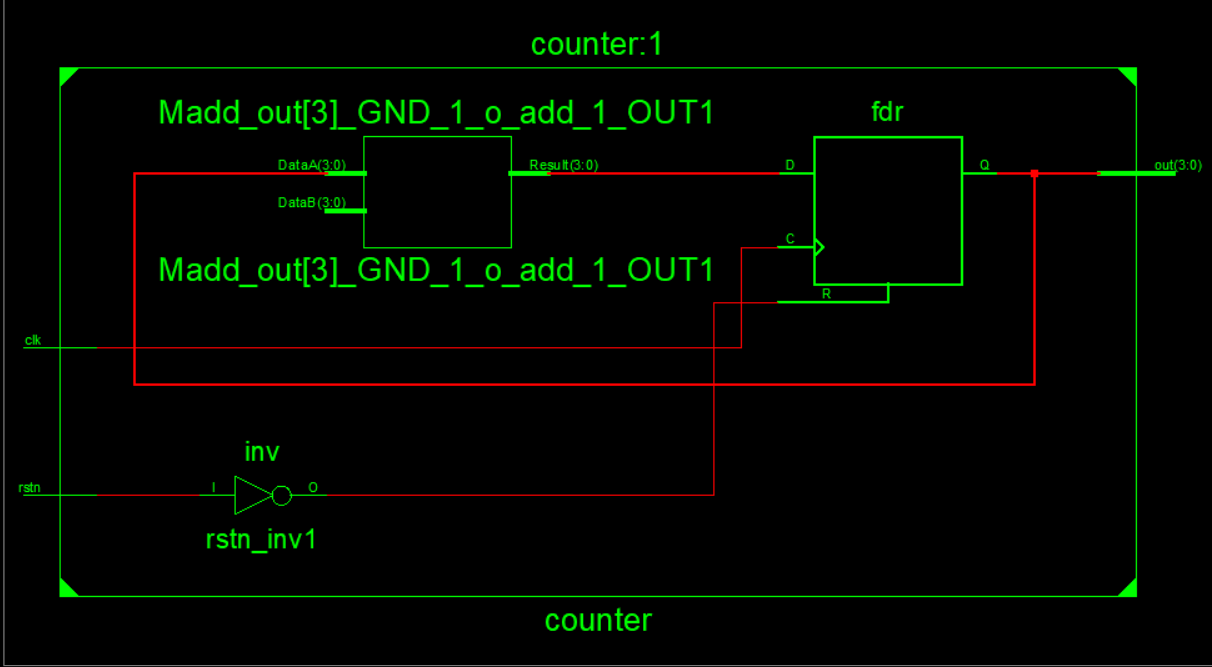
#100;

// Add stimulus here

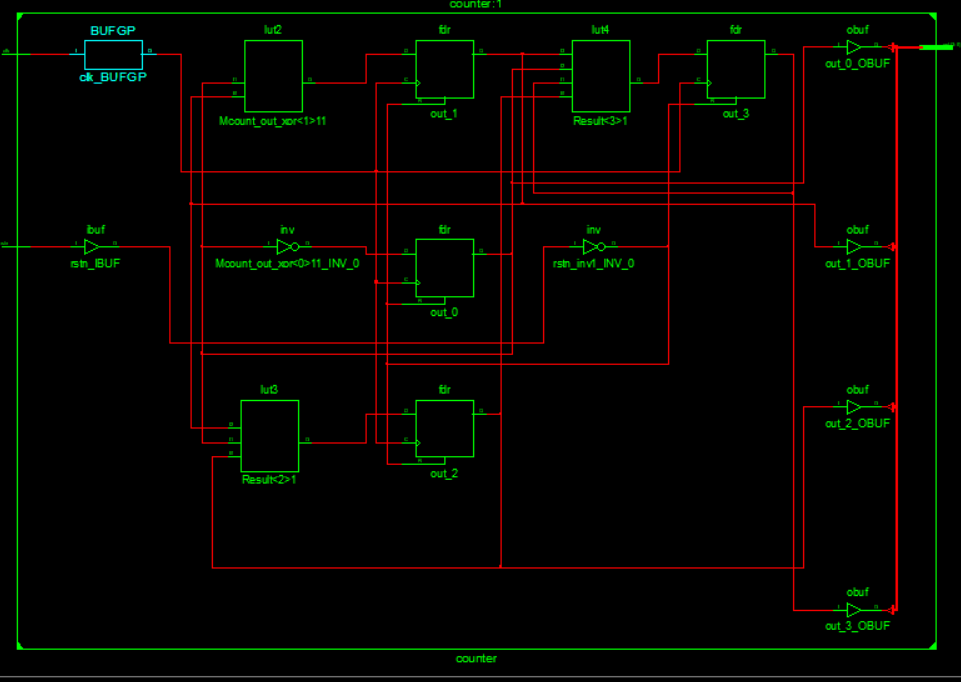
end

endmodule

**RTL Schematics:**

****

**Technology Schematics:**

****

**Graph:**

****

**Ripple Counter Verilog code:**

module dff (input d,

input clk,

input rstn,

output reg q,

output qn);

always @ (posedge clk or negedge rstn)

if (!rstn)

q <= 0;

else

q <= d;

assign qn = ~q;

endmodule

module ripple ( input clk,

input rstn,

output [3:0] out);

wire q0;

wire qn0;

wire q1;

wire qn1;

wire q2;

wire qn2;

wire q3;

wire qn3;

dff dff0 ( .d (qn0),

.clk (clk),

.rstn (rstn),

.q (q0),

.qn (qn0));

dff dff1 ( .d (qn1),

.clk (q0),

.rstn (rstn),

.q (q1),

.qn (qn1));

dff dff2 ( .d (qn2),

.clk (q1),

.rstn (rstn),

.q (q2),

.qn (qn2));

dff dff3 ( .d (qn3),

.clk (q2),

.rstn (rstn),

.q (q3),

.qn (qn3));

assign out = {qn3, qn2, qn1, qn0};

endmodule

**Test-bench for Ripple Counter:**

module ripple\_tb;

// Inputs

reg clk;

reg rstn;

// Outputs

wire [3:0] out;

// Instantiate the Unit Under Test (UUT)

ripple uut (

.clk(clk),

.rstn(rstn),

.out(out)

);

initial begin

// Initialize Inputs

clk = 0;

rstn = 0;

// Wait 100 ns for global reset to finish

#100;

clk = 0;

rstn = 1;

// Wait 100 ns for global reset to finish

#100;

clk = 1;

rstn = 0;

// Wait 100 ns for global reset to finish

#100;

clk = 1;

rstn = 1;

// Wait 100 ns for global reset to finish

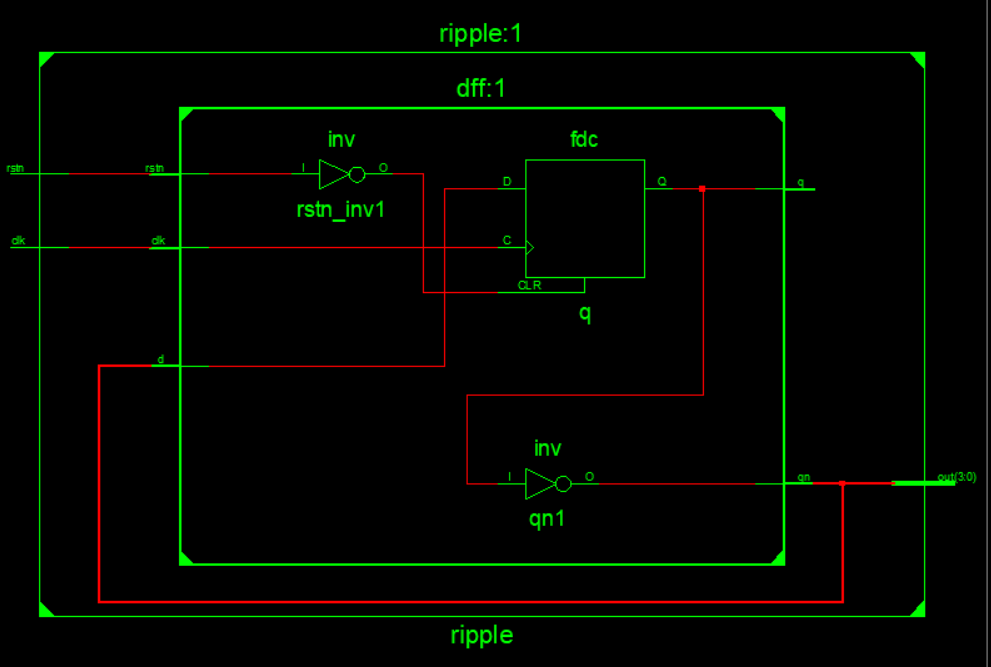
#100;

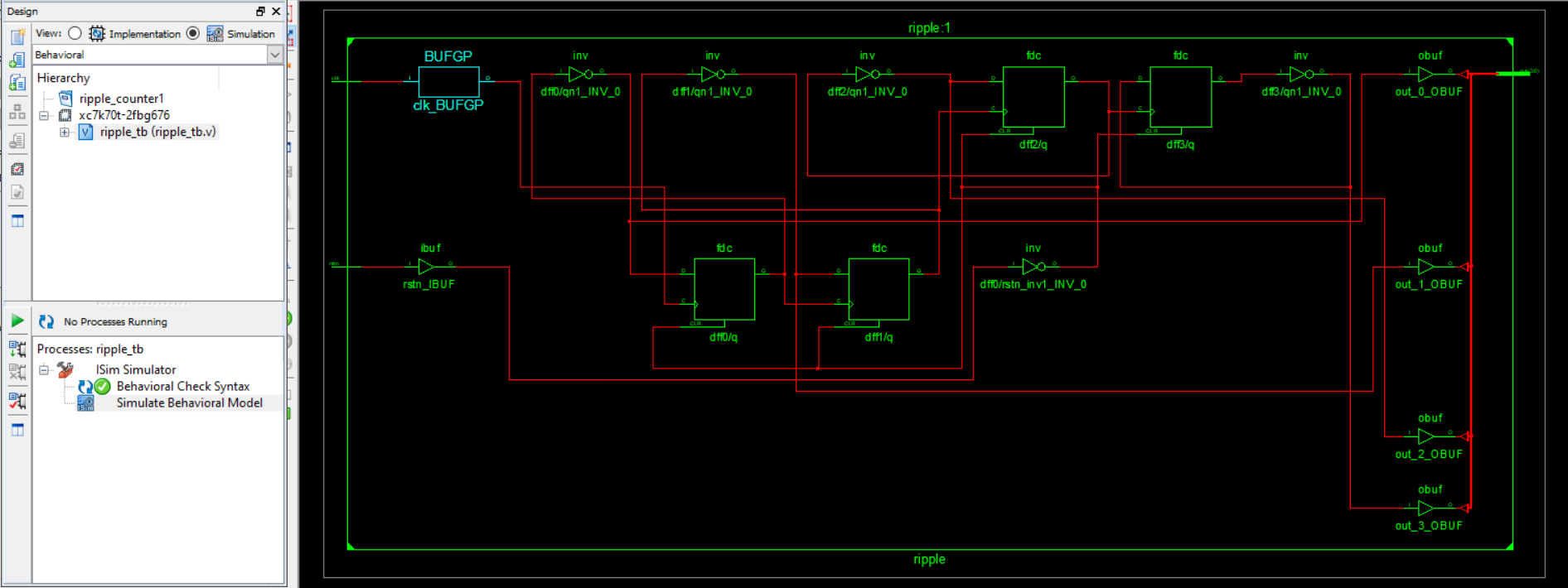
// Add stimulus here

end

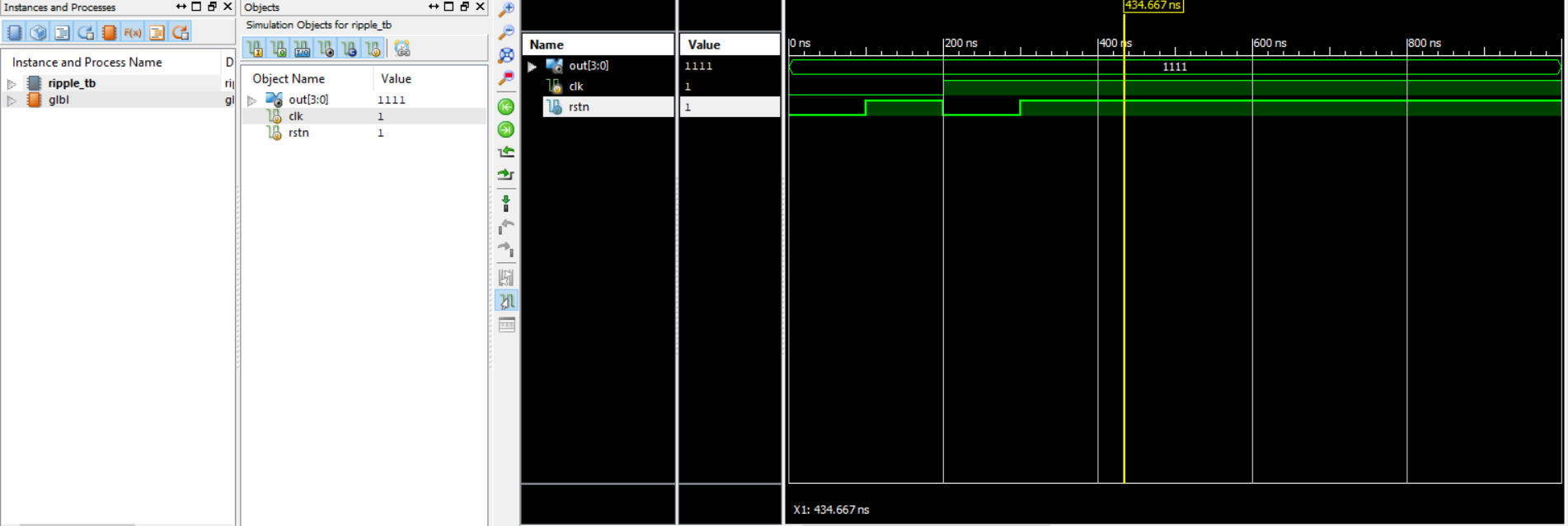
endmodule

**RTL schematics:**

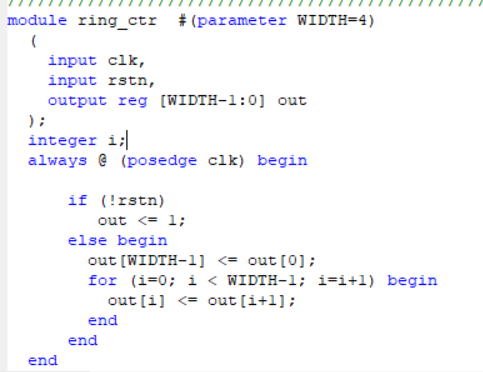
****

**Technology Schematics:**

**Graph:**

****

**Ring Counter Verilog code:**

****

**Ring Counter TB:**

module ring\_ctrTB;

// Inputs

reg clk;

reg rstn;

// Outputs

wire [3:0] out;

// Instantiate the Unit Under Test (UUT)

ring\_ctr uut (

.clk(clk),

.rstn(rstn),

.out(out)

);

initial begin

// Initialize Inputs

clk = 0;

rstn = 0;

// Wait 100 ns for global reset to finish

#100;

clk = 0;

rstn = 1;

// Wait 100 ns for global reset to finish

#100;

clk = 1;

rstn = 0;

// Wait 100 ns for global reset to finish

#100;

clk = 1;

rstn = 1;

// Wait 100 ns for global reset to finish

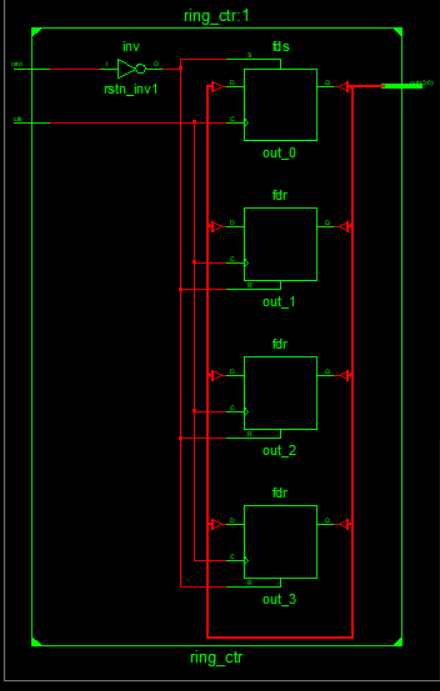
#100;

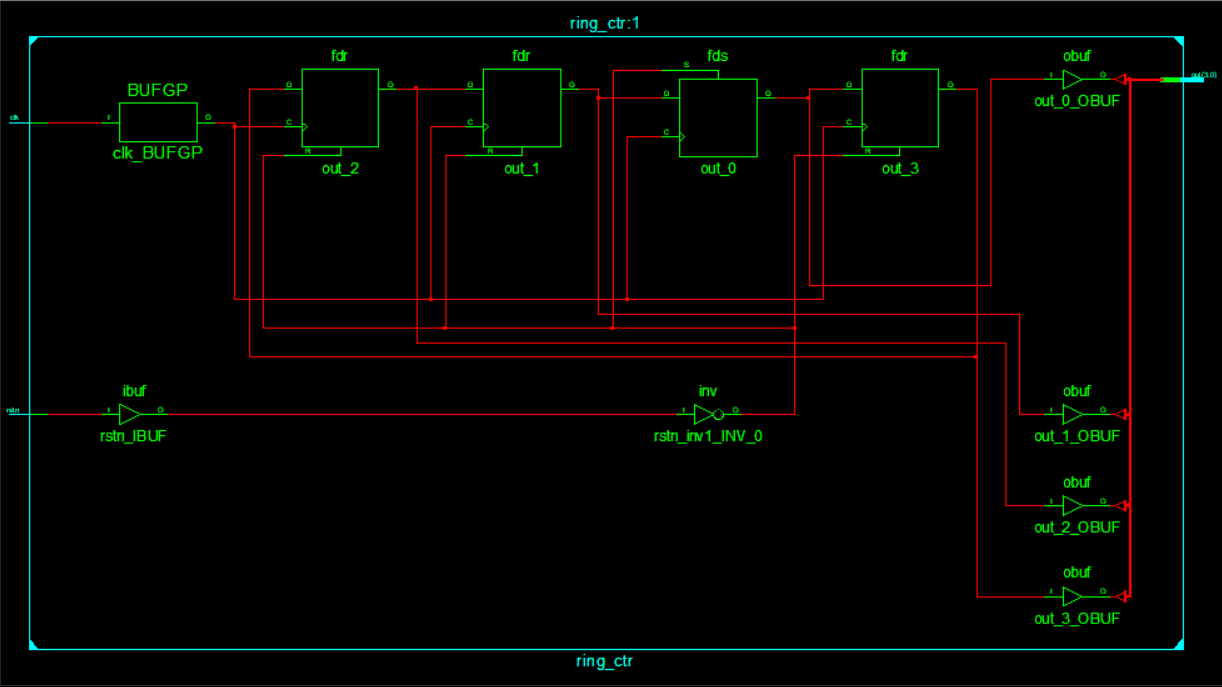
// Add stimulus here

end

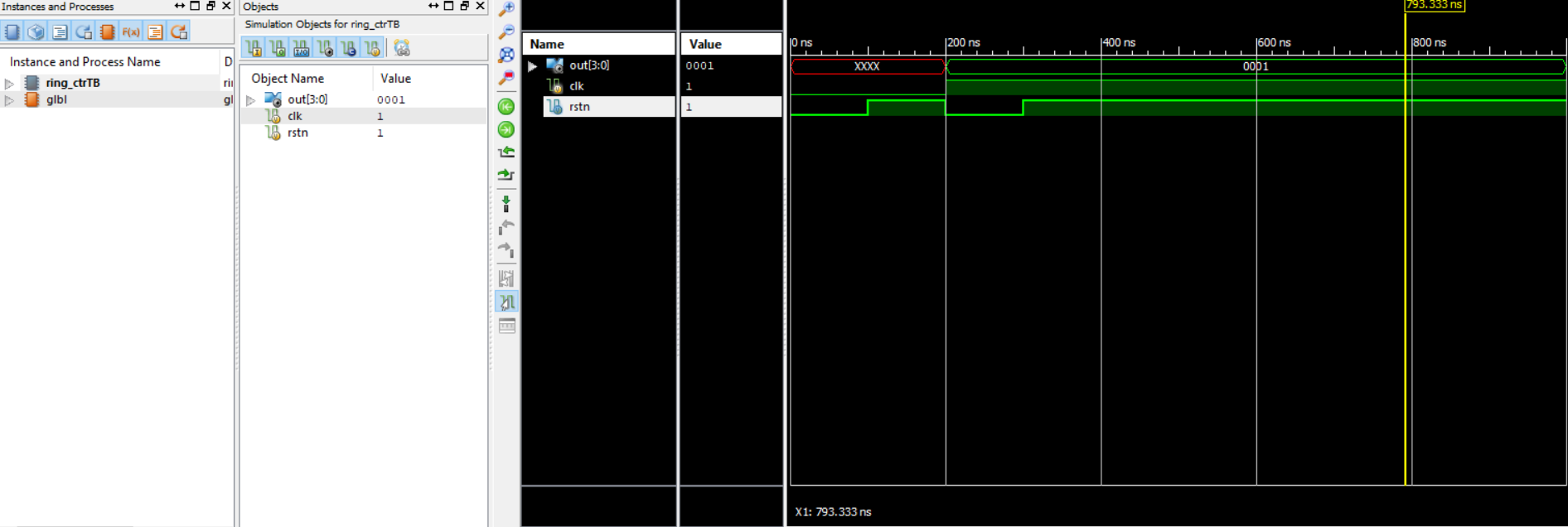
endmodule

**RTL schematics:**

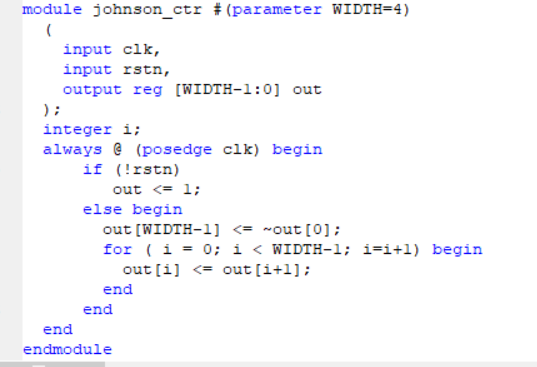
****

**Technology Schematics:**

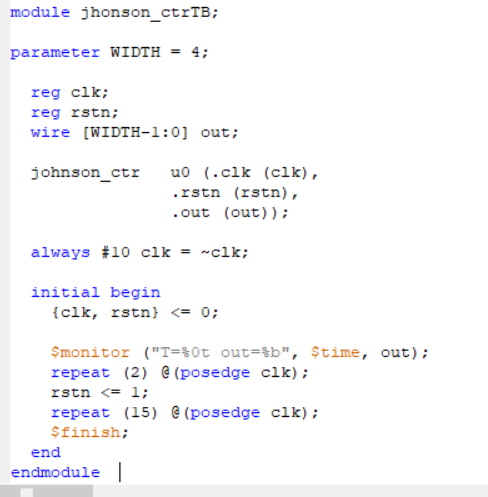
**Graph:**

****

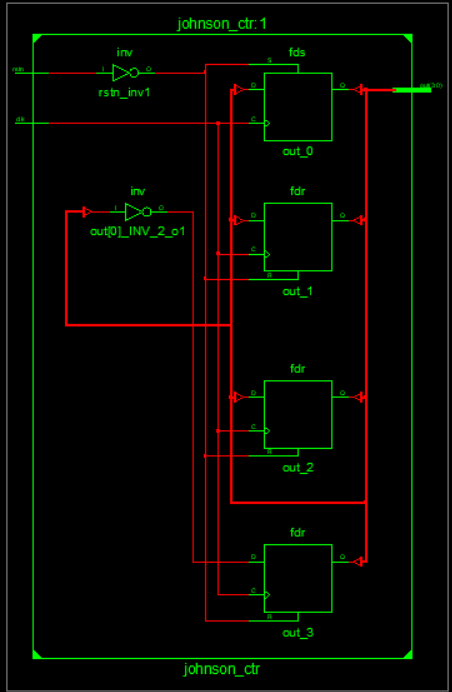
**Johnson Counter Verilog:**

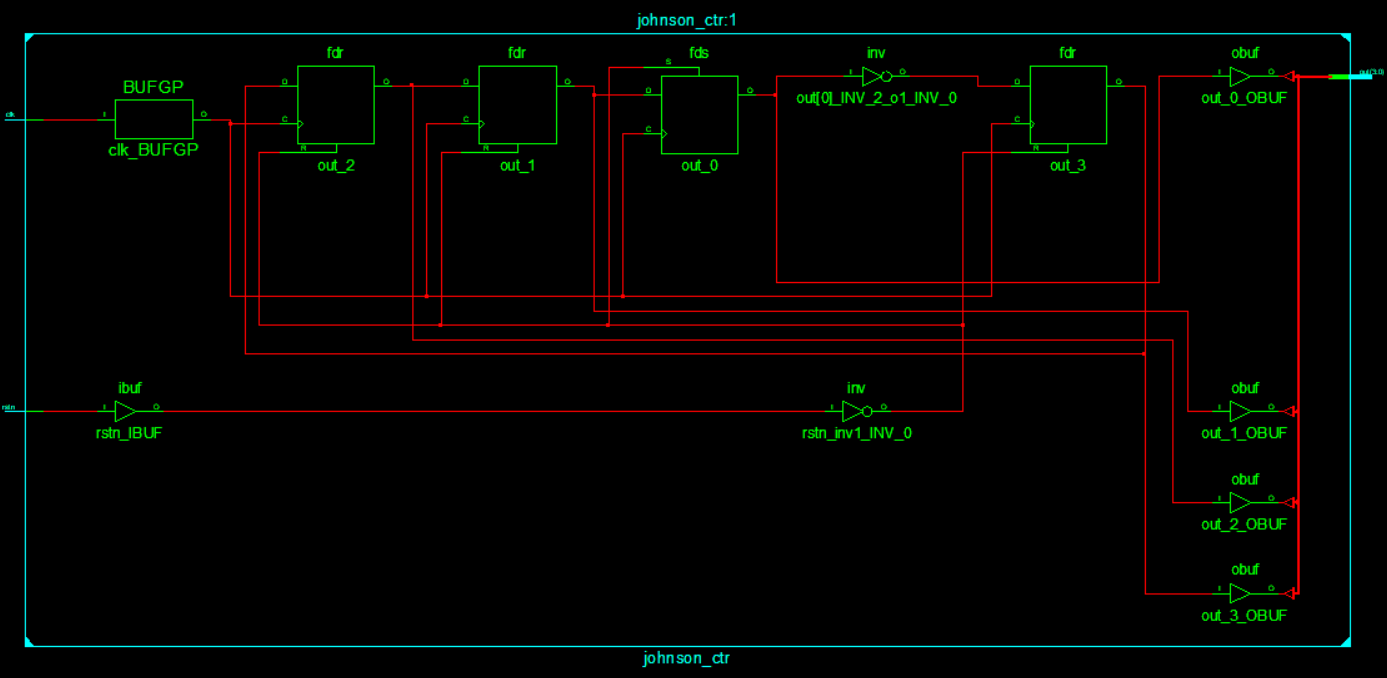
****

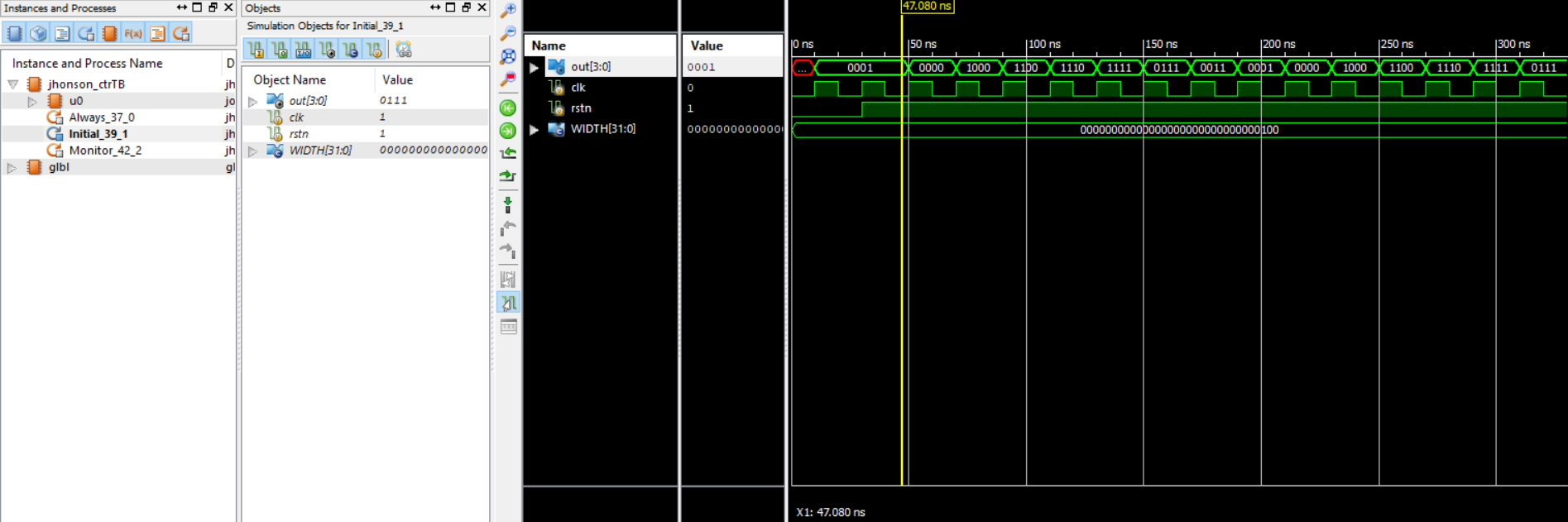
**Johnson counter TB:**

****

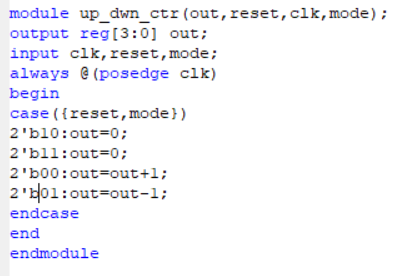
**RTL Schematics:**

****

**Technology schematics:**

**Graph:** ****

**Up-Down Counter Verilog:**

****

**Up-Down Counter Verilog:**

module updown\_counterTB;

// Inputs

reg reset;

reg clk;

reg mode;

// Outputs

wire [3:0] out;

// Instantiate the Unit Under Test (UUT)

up\_dwn\_ctr uut (

.out(out),

.reset(reset),

.clk(clk),

.mode(mode)

);

initial begin

// Initialize Inputs

reset = 1;

clk = 1;

mode = 0;

// Wait 100 ns for global reset to finish

#100;

reset = 0;

mode = 0;

// Wait 100 ns for global reset to finish

#200;

reset = 1;

mode = 0;

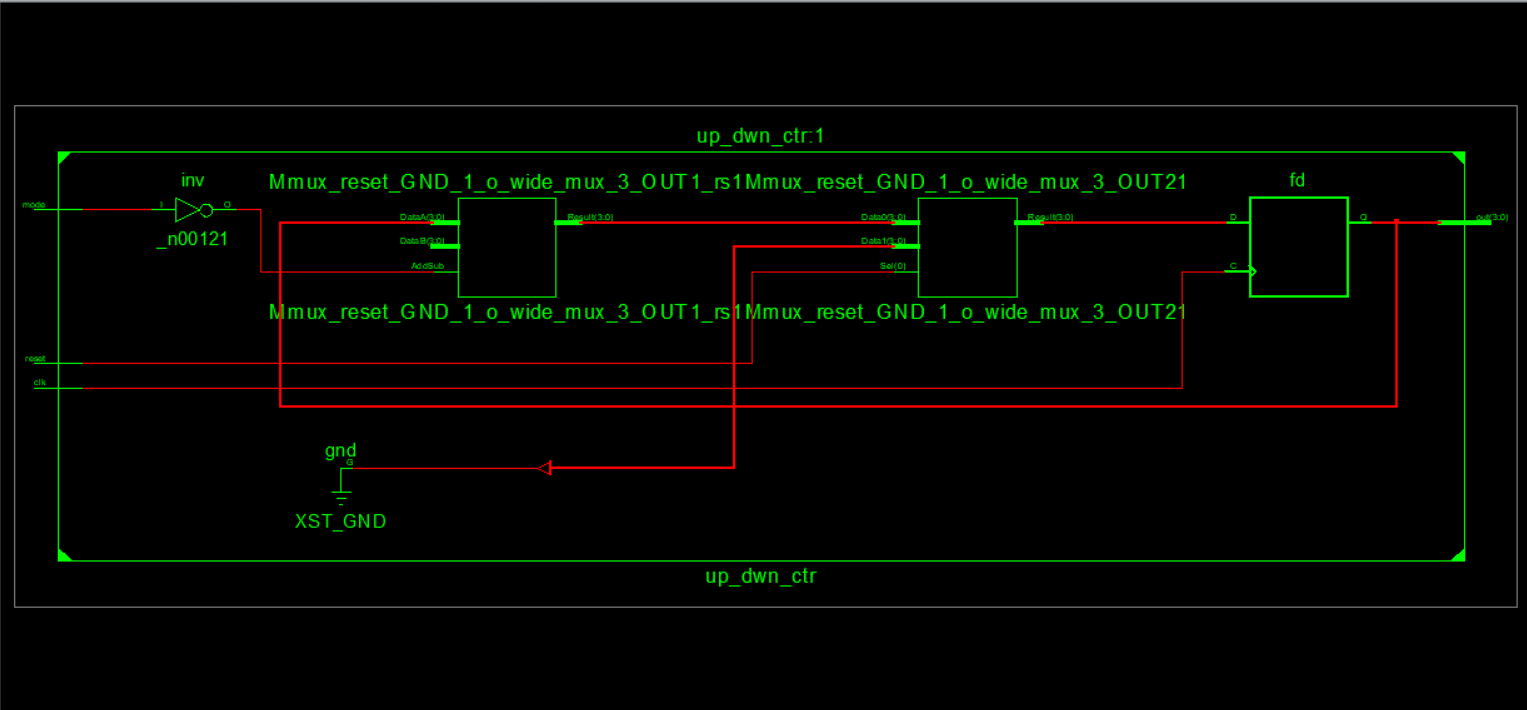
// Add stimulus here

end

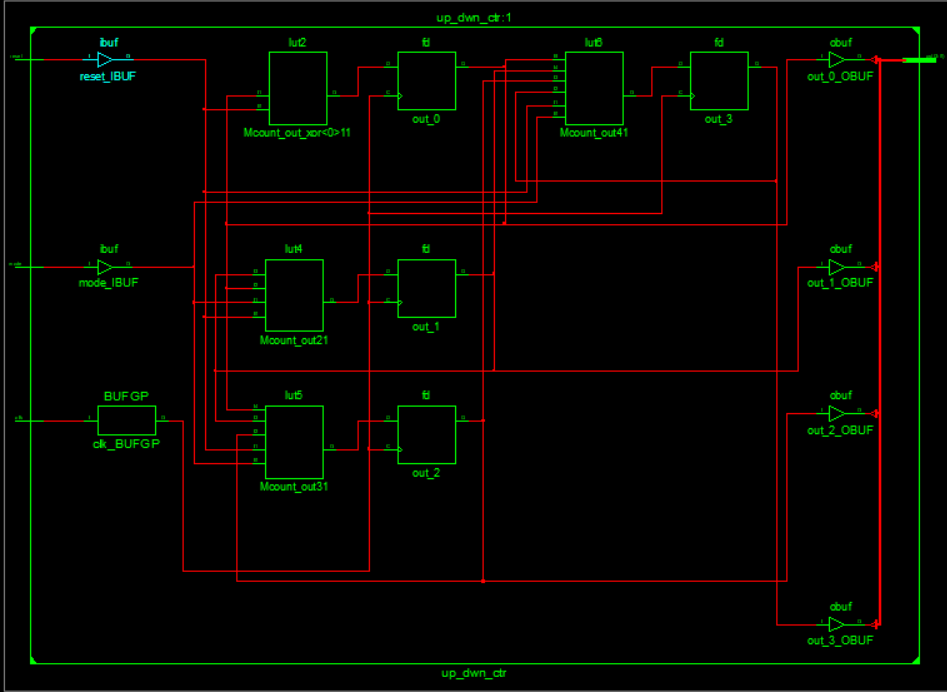
always #25 clk=~clk;

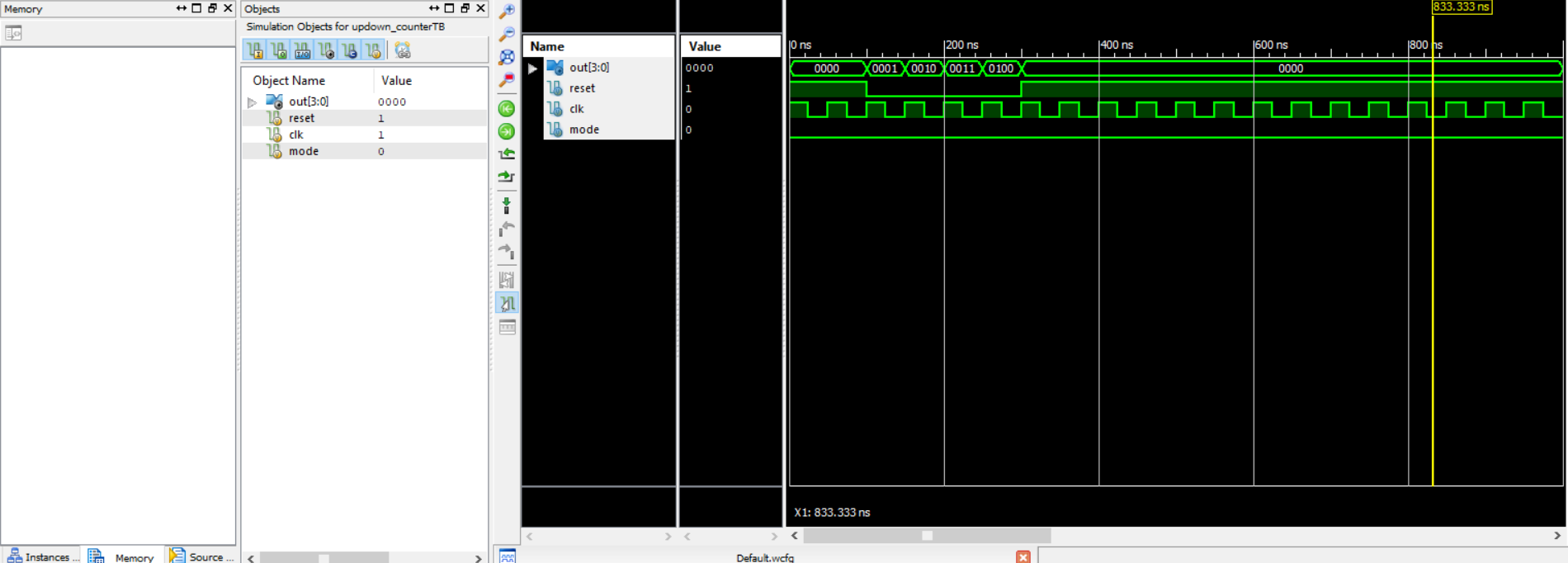
endmodule

**RTL schematics:**

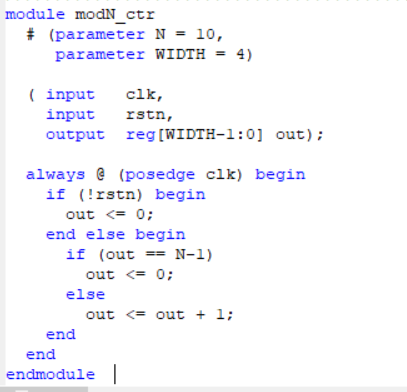
****

**Technology Schematics:**

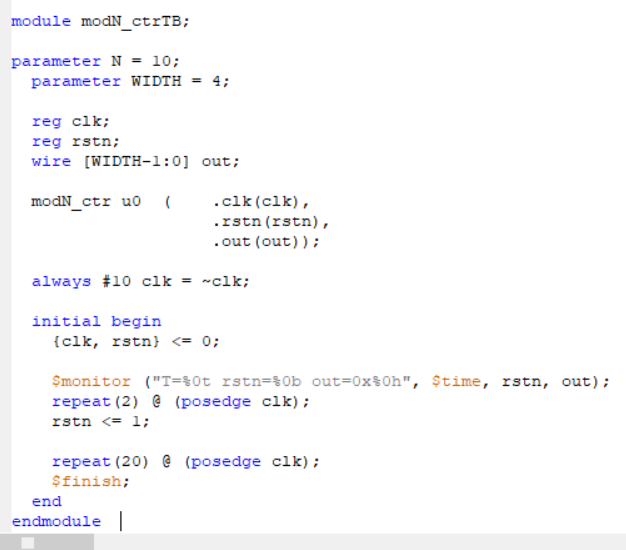
****

**Graph:** ****

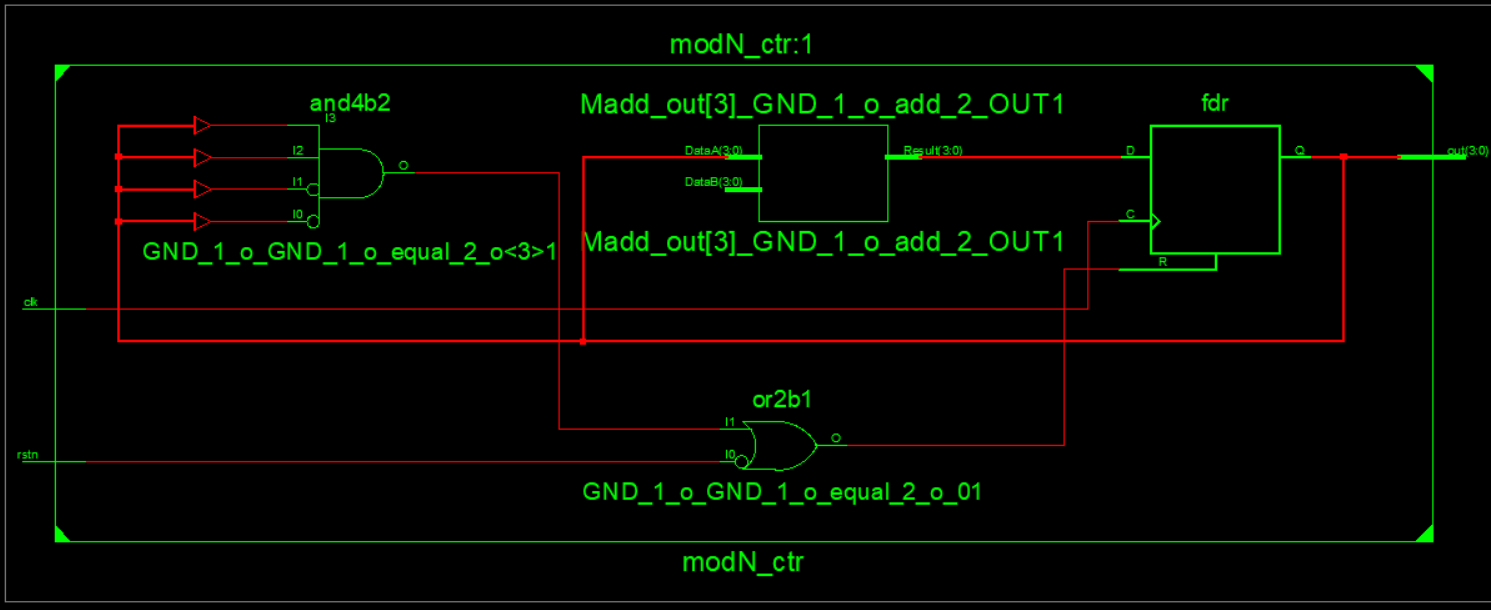
**MOD Counter Verilog:**

****

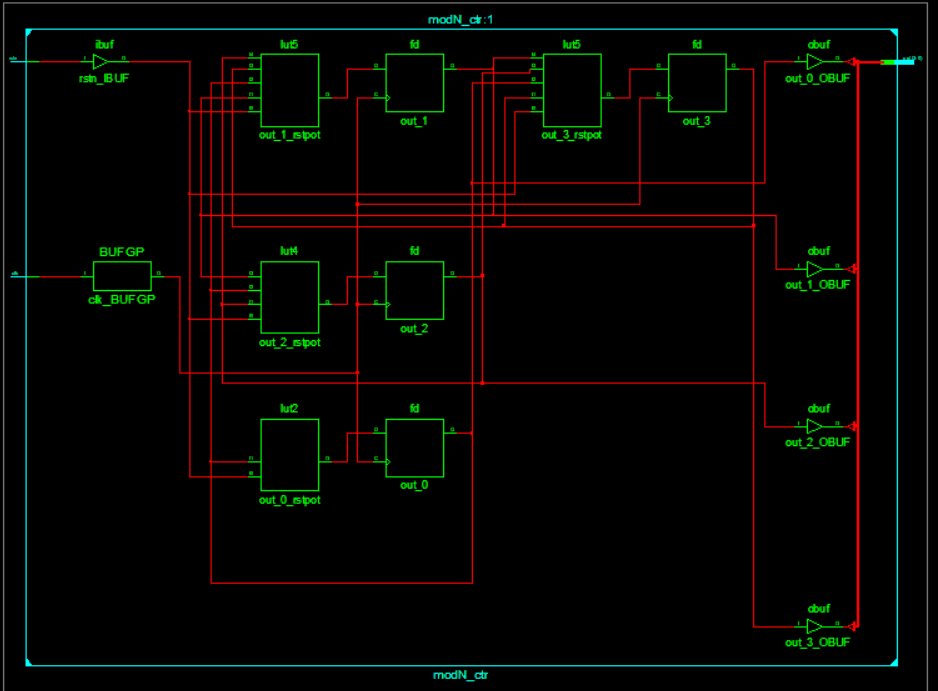
**MOD Counter Testbench:**

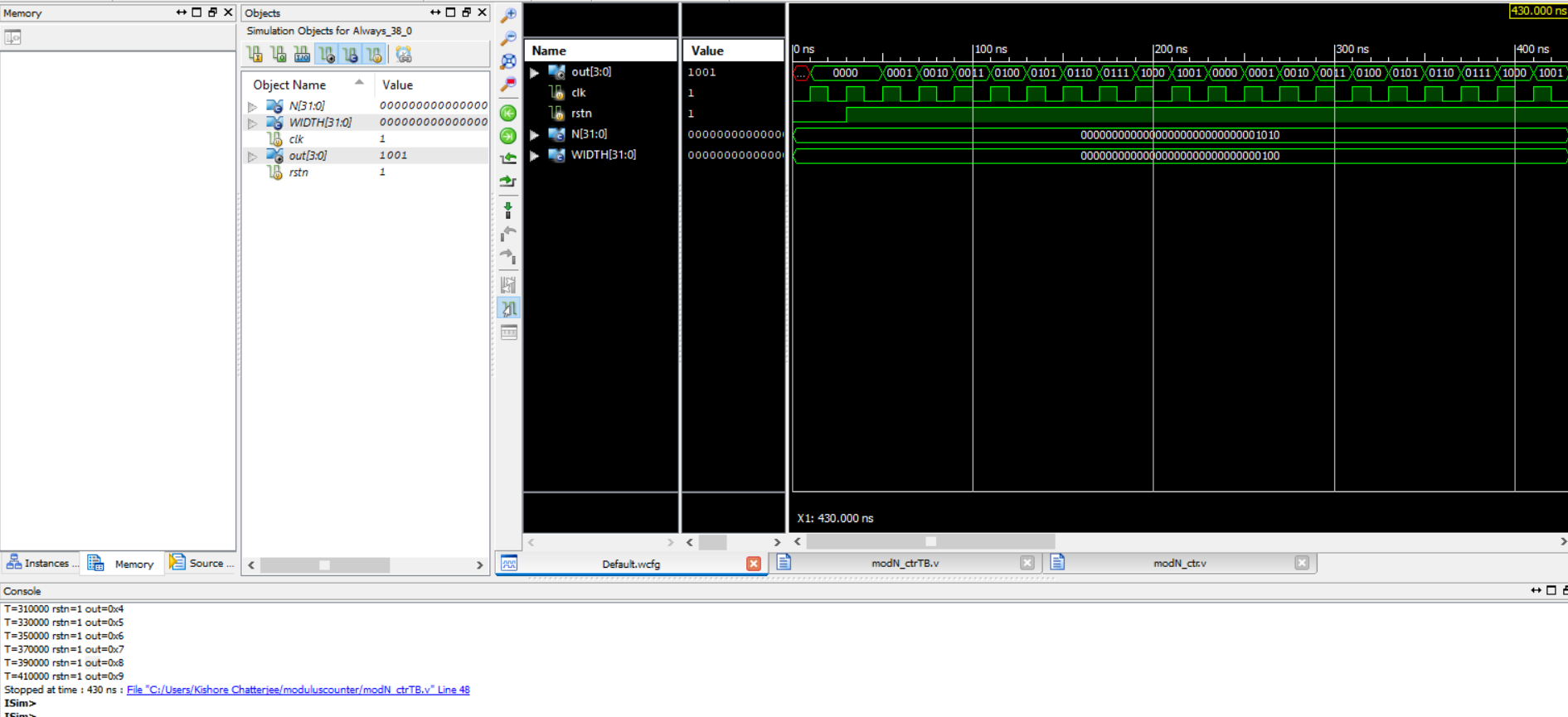
****

**RTL Schematics:**

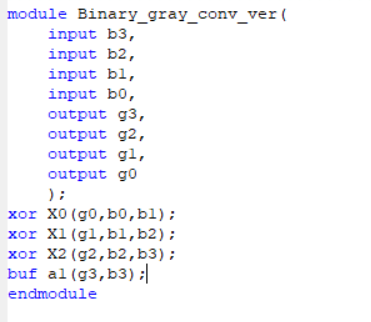
****

**Technology Schematics:**

****

**Graph**

**Binary to Gray Code Converter Verilog code:**

****

**Test\_bench:**

module b\_g\_conv\_TB;

// Inputs

reg b3;

reg b2;

reg b1;

reg b0;

// Outputs

wire g3;

wire g2;

wire g1;

wire g0;

// Instantiate the Unit Under Test (UUT)

Binary\_gray\_conv\_ver uut (

.b3(b3),

.b2(b2),

.b1(b1),

.b0(b0),

.g3(g3),

.g2(g2),

.g1(g1),

.g0(g0)

);

initial begin

// Initialize Inputs

b3 = 0;

b2 = 0;

b1 = 0;

b0 = 0;

// Wait 100 ns for global reset to finish

#100;

b3 = 0;

b2 = 0;

b1 = 0;

b0 = 1;

// Wait 100 ns for global reset to finish

#100;

b3 = 0;

b2 = 0;

b1 = 1;

b0 = 0;

// Wait 100 ns for global reset to finish

#100;

b3 = 0;

b2 = 0;

b1 = 1;

b0 = 1;

// Wait 100 ns for global reset to finish

#100;

b3 = 0;

b2 = 1;

b1 = 0;

b0 = 0;

// Wait 100 ns for global reset to finish

#100;

b3 = 0;

b2 = 1;

b1 = 0;

b0 = 1;

// Wait 100 ns for global reset to finish

#100;

b3 = 0;

b2 = 1;

b1 = 1;

b0 = 0;

// Wait 100 ns for global reset to finish

#100;

b3 = 0;

b2 = 1;

b1 = 1;

b0 = 1;

// Wait 100 ns for global reset to finish

#100;

b3 = 1;

b2 = 0;

b1 = 0;

b0 = 0;

// Wait 100 ns for global reset to finish

#100;

b3 = 1;

b2 = 0;

b1 = 0;

b0 = 1;

// Wait 100 ns for global reset to finish

#100;

b3 = 1;

b2 = 0;

b1 = 1;

b0 = 0;

// Wait 100 ns for global reset to finish

#100;

b3 = 1;

b2 = 0;

b1 = 1;

b0 = 1;

// Wait 100 ns for global reset to finish

#100;

b3 = 1;

b2 = 1;

b1 = 0;

b0 = 0;

// Wait 100 ns for global reset to finish

#100;

// Add stimulus here

b3 = 1;

b2 = 1;

b1 = 0;

b0 = 1;

// Wait 100 ns for global reset to finish

#100;

b3 = 1;

b2 = 1;

b1 = 1;

b0 = 0;

// Wait 100 ns for global reset to finish

#100;

b3 = 1;

b2 = 1;

b1 = 1;

b0 = 1;

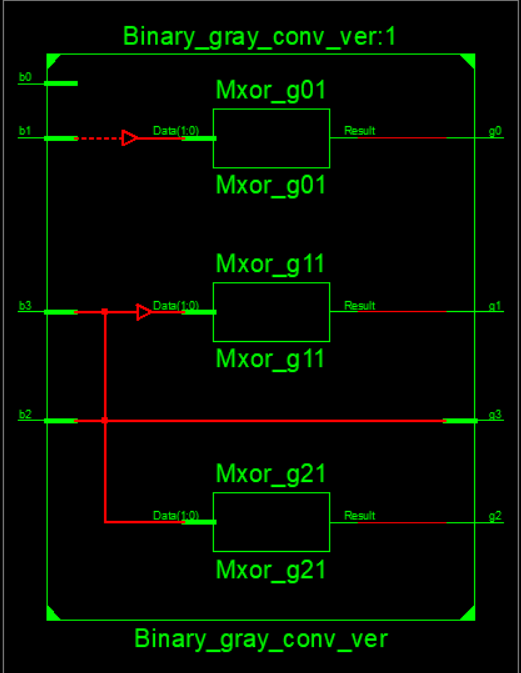
// Wait 100 ns for global reset to finish

#100;

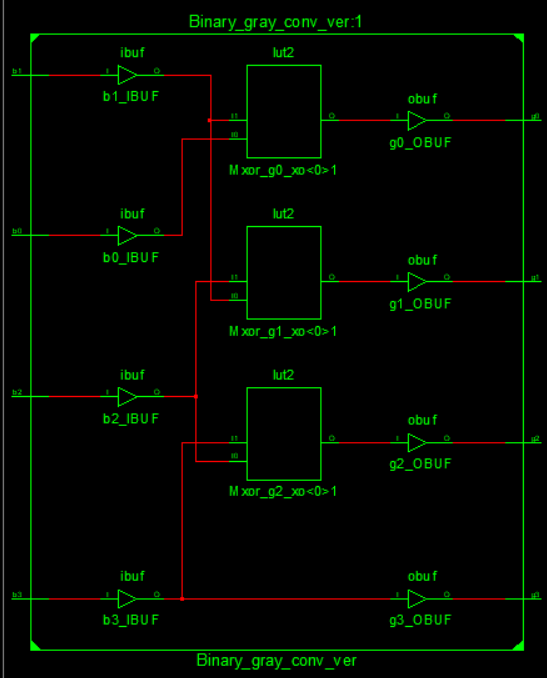
end

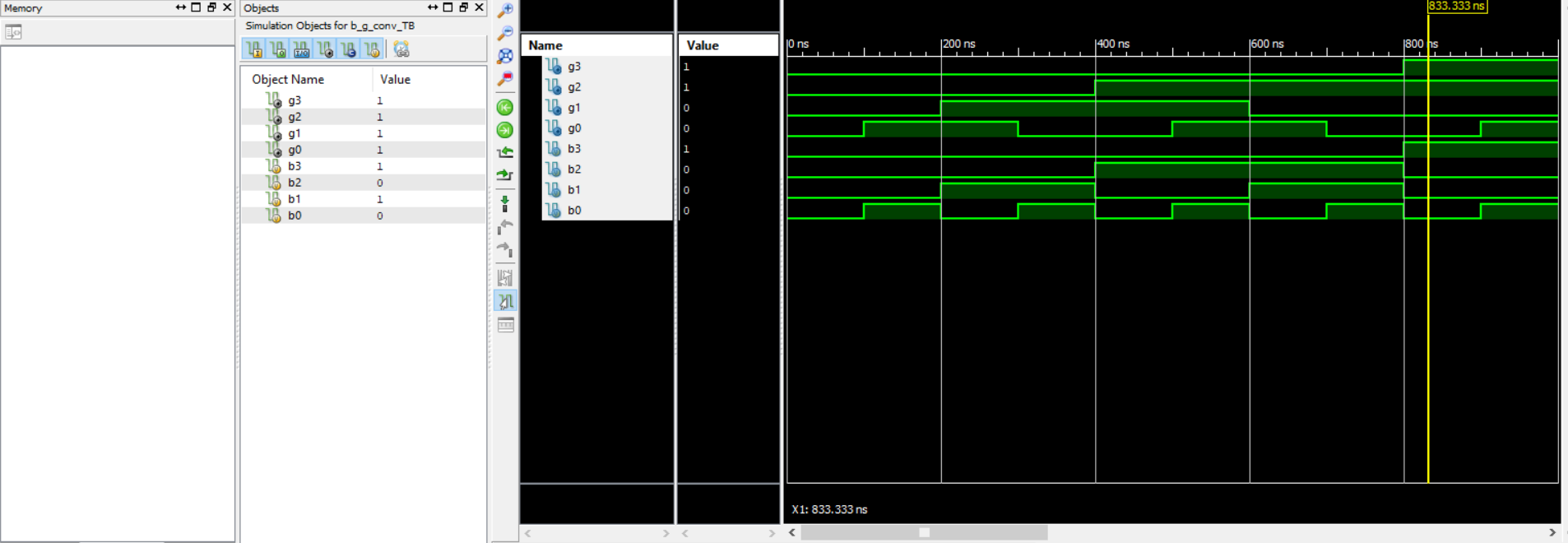
endmodule

**RTL Schematics:**

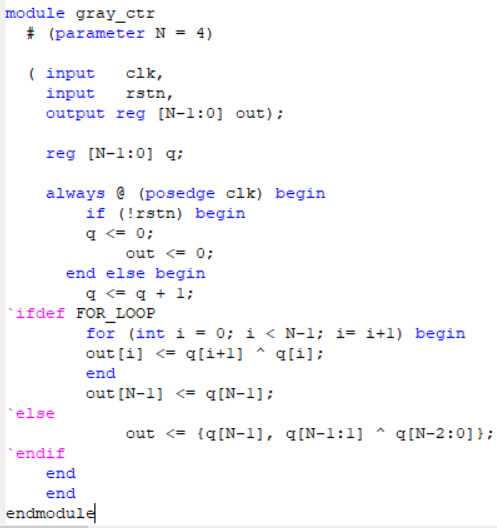
****

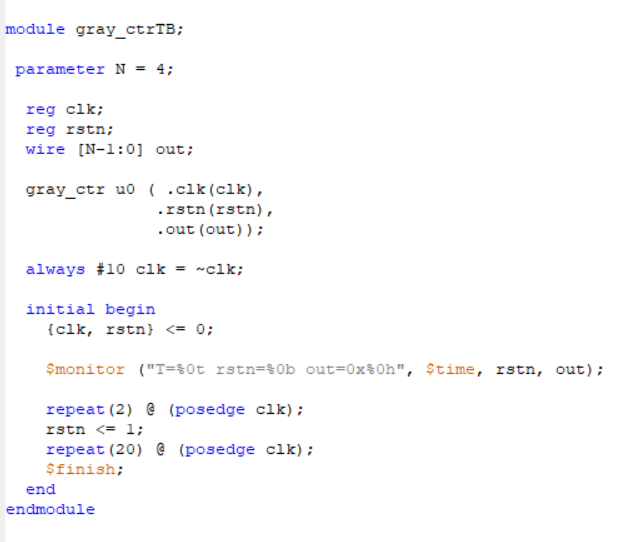
**Technology Schematics:**

****

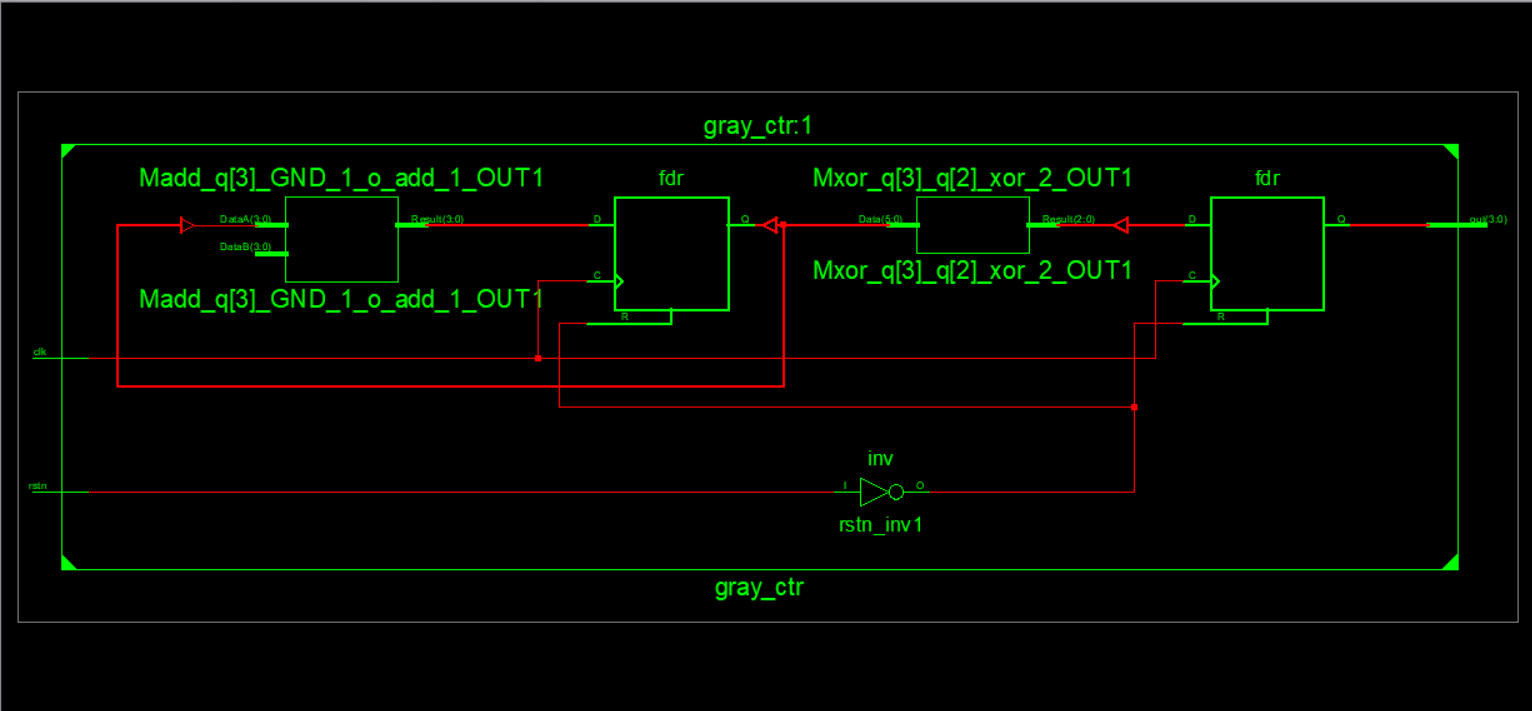
**Graph:** ****

**Gray code Counter Verilog Code:**

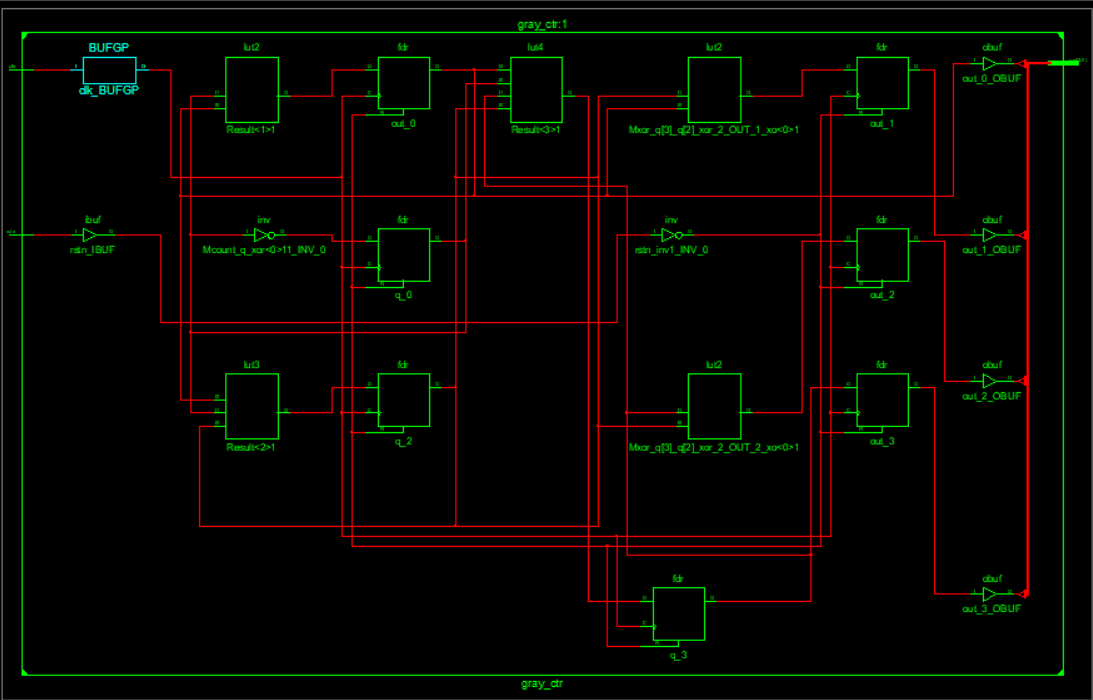


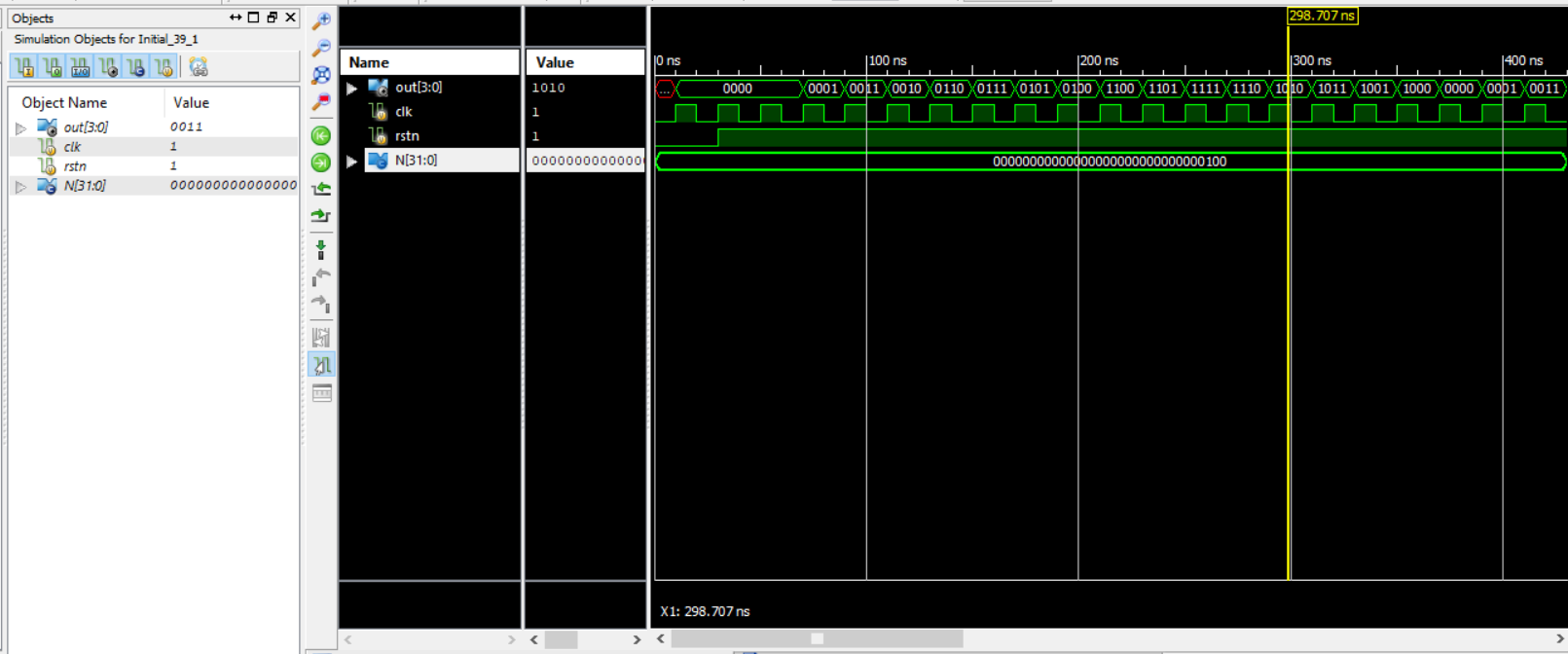
**Test Bench For Gray code Counter:**

**RTL schematics:**

****

**Technology Schematics:**

****

**Graph:** ****

**Verilog Code for Shift Register:**

****

**Test Bench For Shift Register:**

module tb\_sr;

parameter MSB = 16; // [Optional] Declare a parameter to represent number of bits in shift register

reg data; // Declare a variable to drive d-input of design

reg clk; // Declare a variable to drive clock to the design

reg en; // Declare a variable to drive enable to the design

reg dir; // Declare a variable to drive direction of shift register

reg rstn; // Declare a variable to drive reset to the design

wire [MSB-1:0] out; // Declare a wire to capture output from the design

// Instantiate design (16-bit shift register) by passing MSB and connect with TB signals

shft\_register\_v #(MSB) sr0 ( .d (data),

.clk (clk),

.en (en),

.dir (dir),

.rstn (rstn),

.out (out));

// Generate clock time period = 20ns, freq => 50MHz

always #10 clk = ~clk;

// Initialize variables to default values at time 0

initial begin

clk <= 0;

en <= 0;

dir <= 0;

rstn <= 0;

data <= 'h1;

end

// Drive main stimulus to the design to verify if this works

initial begin

// 1. Apply reset and deassert reset after some time

rstn <= 0;

#20 rstn <= 1;

en <= 1;

// 2. For 7 clocks, drive alternate values to data pin

repeat (7) @ (posedge clk)

data <= ~data;

// 3. Shift direction and drive alternate value to data pin for another 7 clocks

#10 dir <= 1;

repeat (7) @ (posedge clk)

data <= ~data;

// 4. Drive nothing for the next 7 clocks, allow shift register to shift based on dir simply

repeat (7) @ (posedge clk);

// 5. Finish the simulation

$finish;

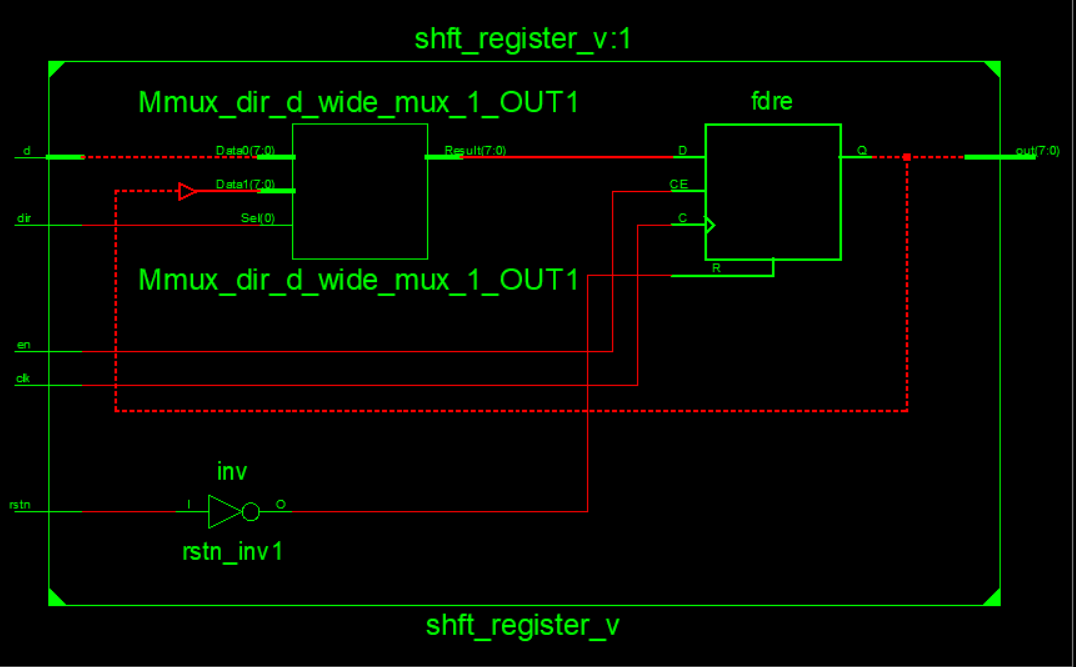
end

// Monitor values of these variables and print them into the log file for debug

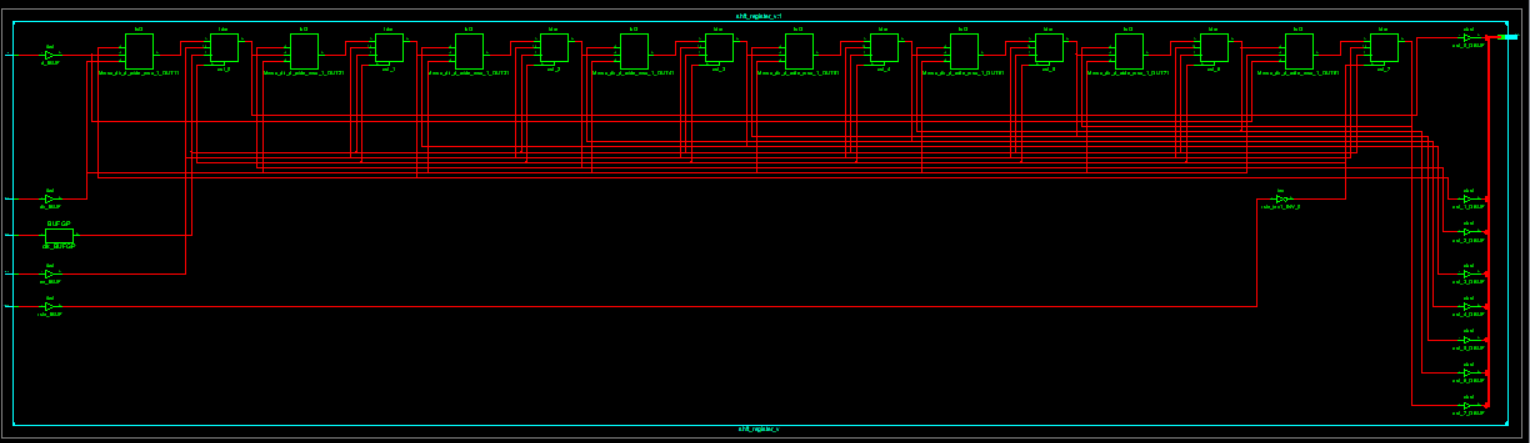
initial

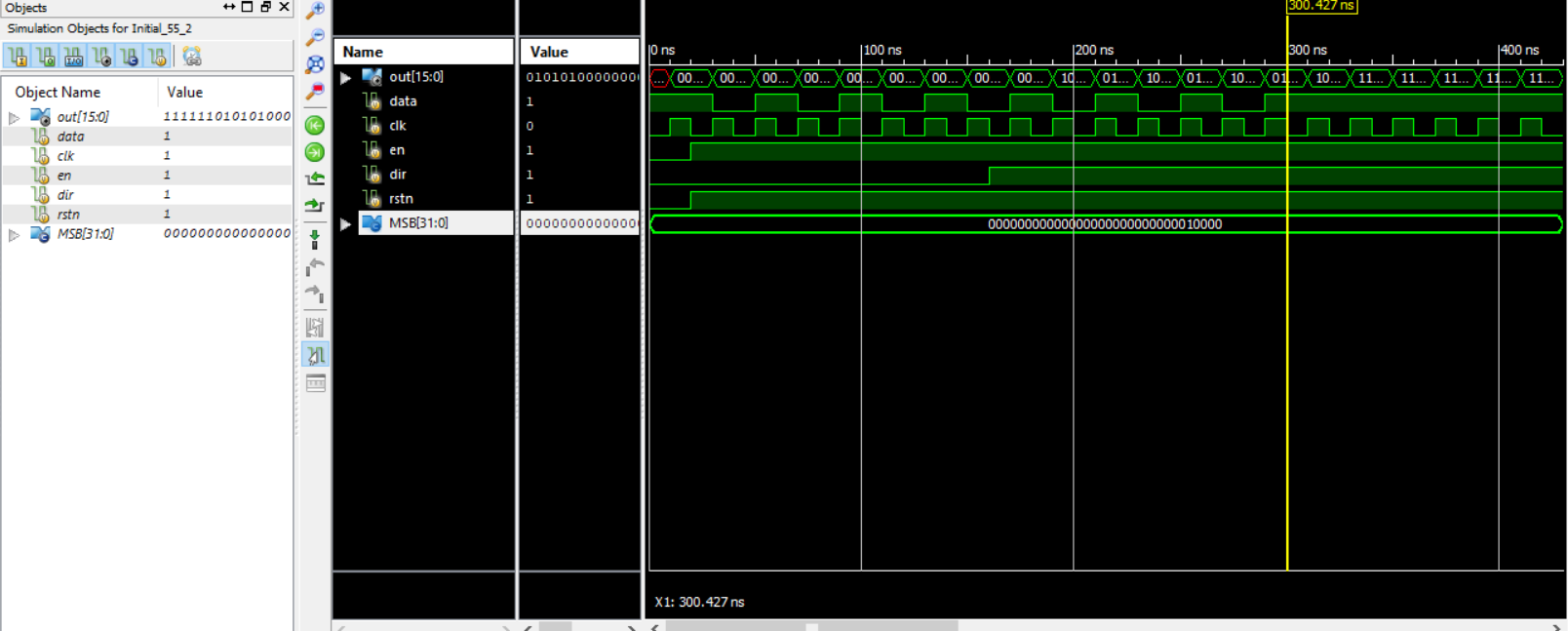
$monitor("rstn=%0b data=%b, en=%0b, dir=%0b, out=%b", rstn, data, en, dir, out);

Endmodule

**RTL schematics:**

**Technology Schematics:**

****

**Graph:**

**Verilog Code For Single Port RAM:**

// RAM control unit

module ramcu (

clk , // Clock Input

address , // Address Input

data , // Data bi-directional

cs , // Chip Select

we , // Write Enable/Read Enable

oe // Output Enable

);

// Setup some parameters

parameter DATA\_WIDTH = 8; // word size of the memory

parameter ADDR\_WIDTH = 8; // number of memory words, e.g. 2^8-1

parameter RAM\_DEPTH = 1 << ADDR\_WIDTH;

// Define inputs

input clk, cs, we, oe;

input [ADDR\_WIDTH-1:0] address;

// Data is bidirectional

inout [DATA\_WIDTH-1:0] data;

// Private registers

reg [DATA\_WIDTH-1:0] mem [0:RAM\_DEPTH-1]; // Set up the memory array

reg [DATA\_WIDTH-1:0] r\_data; // copy of data value to return

reg r\_oe; // delayed oe, r\_oe updates only when rdata updated

// START OF OPERATION: ///////////////////////////////////////////////

// Tri-State Buffer control:

// The data item is defined as an inout, only when oe, output enabled

// is high, should it send a value to data, otherwise it should keep

// the data port linked to high impedence (z) so as not to drive a value.

// i.e. output to data happens when oe = 1 & cs = 1 & we = 0

// Write to memory when: we = 1 & cs = 1

always@ (posedge clk)

begin

if (cs)

begin

if (we) mem[address] <= data;

r\_data <= mem[address];

end

r\_oe <= oe;

end

assign data = (oe && cs && !we)? r\_data : 8'bz;

endmodule // end ramcu

**Test Bench for Single port RAM:**

// EEE4120F Memory Control Unit Example

// Testbench for the ramcu RAM control unit

module ramcu\_tb ();

wire [7:0] data; // this is the connection to ramcu data port

reg [7:0] data\_value; // need a register to store value to send on a write

reg clk, cs, we, oe;

reg [7:0] addr;

// Instantiate the module to be tested

ramcu ramcu\_uut(clk,addr,data,cs,we,oe);

assign data = !oe ? data\_value : 'bz;

initial begin

// set up initial conditions

clk = 0;

cs = 0;

we = 0;

oe = 0;

addr = 0;

$display("clk cs we oe addr data");

$monitor("%b %b %b %b %03d %03d",

clk,cs,we,oe,addr,data);

// try doing a clock change just when nothing should happen (cs=0)

#5 clk = ~clk;

#5 clk = ~clk;

// let is do a memory write...

// note that you need to deselect the chip before setting up data

$display("write 99 to mem[1]");

oe = 0;

we = 1;

cs = 1;

addr = 8'd1;

data\_value = 8'd99;

#5 clk = ~clk; // do a clock pulse

#5 clk = ~clk;

// now disconnect data\_value from data line

$display("write junk into toplevel data\_value buffer");

cs = 0;

we = 0;

oe = 0;

data\_value = 8'd2;

#5 clk = ~clk; // do a clock pulse

#5 clk = ~clk;

// let's read the value back

$display("check reading back data");

addr = 8'd1;

we = 0;

oe = 1; // ask module to write out data

cs = 1;

#5 clk = ~clk; // do a clock pulse (reads the data)

#5 clk = ~clk;

// you should now have back the value that was written to memory

// disable chip select and write some more junk to local buffer

$display("write more junk into toplevel data\_value");

cs = 0;

data\_value = 8'd123;

#5 clk = ~clk; // do a clock pulse to see what happens

#5 clk = ~clk;

// OPTIONAL: here is a repeat loop to dump part of the memory

addr = 0;

#5

$display("dump some memory:");

repeat (5)

begin

$display("mem[%d]=%d",addr,ramcu\_uut.mem[addr]);

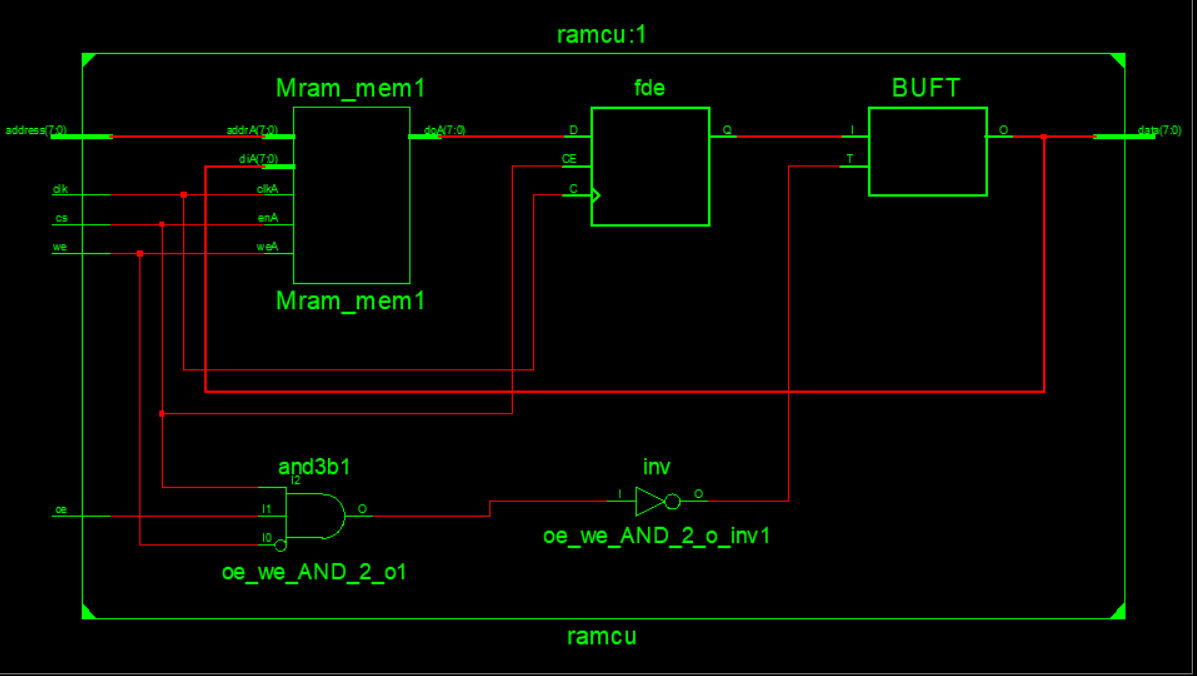
addr = addr + 1;

end

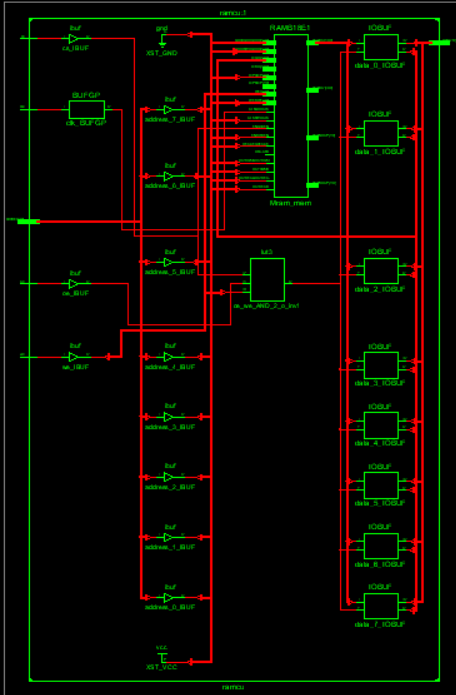
end // end initial

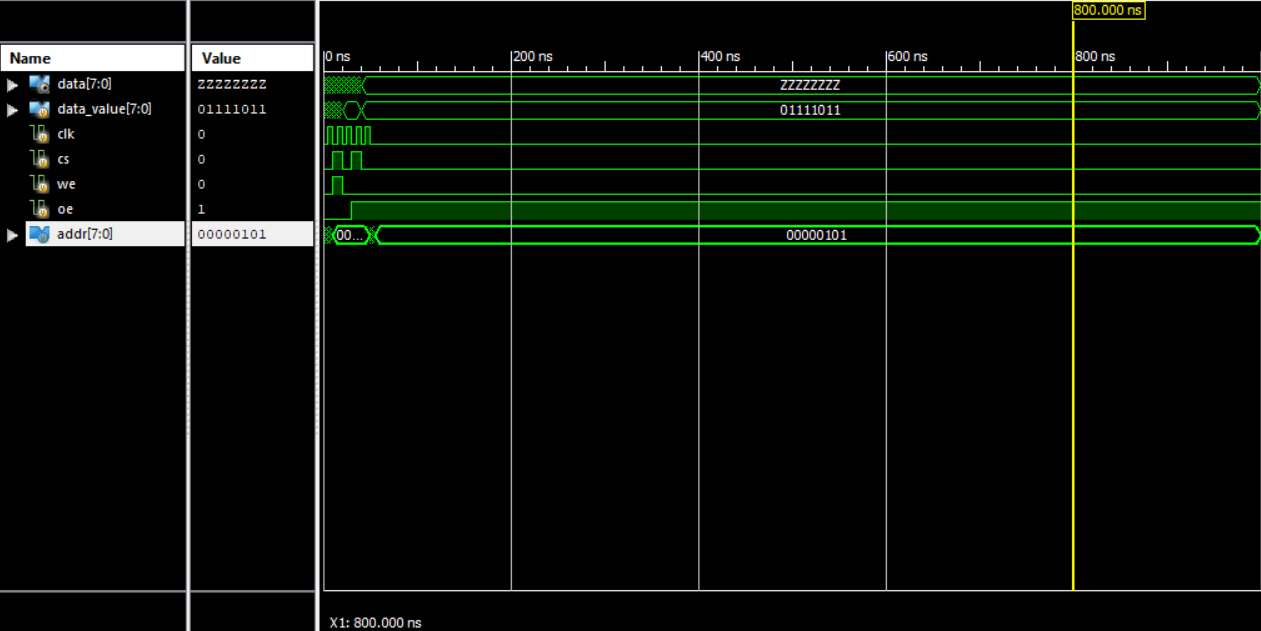
endmodule

**RTL Schematics:**

****

**Technology Schematics:**

****

**Graph:** ****

**Single port Ram verilog:**

module single\_port\_sync\_ram (

clk , // Clock Input

address , // Address Input

data , // Data bi-directional

cs , // Chip Select

we , // Write Enable/Read Enable

oe // Output Enable

);

// Setup some parameters

parameter DATA\_WIDTH = 8; // word size of the memory

parameter ADDR\_WIDTH = 8; // number of memory words, e.g. 2^8-1

parameter RAM\_DEPTH = 1 << ADDR\_WIDTH;

// Define inputs

input clk, cs, we, oe;

input [ADDR\_WIDTH-1:0] address;

// Data is bidirectional

inout [DATA\_WIDTH-1:0] data;

// Private registers

reg [DATA\_WIDTH-1:0] mem [0:RAM\_DEPTH-1]; // Set up the memory array

reg [DATA\_WIDTH-1:0] r\_data; // copy of data value to return

reg r\_oe; // delayed oe, r\_oe updates only when rdata updated

// START OF OPERATION: ///////////////////////////////////////////////

// Tri-State Buffer control:

// The data item is defined as an inout, only when oe, output enabled

// is high, should it send a value to data, otherwise it should keep

// the data port linked to high impedence (z) so as not to drive a value.

// i.e. output to data happens when oe = 1 & cs = 1 & we = 0

// Write to memory when: we = 1 & cs = 1

always@ (posedge clk)

begin

if (cs)

begin

if (we) mem[address] <= data;

r\_data <= mem[address];

end

r\_oe <= oe;

end

assign data = (oe && cs && !we)? r\_data : 8'bz;

endmodule //

**Test Bench:**

module tb;

parameter ADDR\_WIDTH = 4;

parameter DATA\_WIDTH = 16;

parameter DEPTH = 16;

reg clk;

reg cs;

reg we;

reg oe;

reg [ADDR\_WIDTH-1:0] address;

wire [DATA\_WIDTH-1:0] data;

reg [DATA\_WIDTH-1:0] tb\_data;

single\_port\_sync\_ram (.DATA\_WIDTH(DATA\_WIDTH))

u0(

.clk(clk),

.address(address),

.data(data),

.cs(cs),

.we(we),

.oe(oe)

);

always #10 clk = ~clk;

assign data = !oe ? tb\_data : 'hz;

integer i;

initial begin

{clk, cs, we, address, tb\_data, oe} <= 0;

repeat (2) @ (posedge clk);

for ( i = 0; i < 2\*\*ADDR\_WIDTH; i= i+1) begin

repeat (1) @(posedge clk) address <= i; we <= 1; cs <= 1; oe <= 0; tb\_data <= $random;

end

for ( i = 0; i < 2\*\*ADDR\_WIDTH; i= i+1) begin

repeat (1) @(posedge clk) address <= i; we <= 0; cs <= 1; oe <= 1;

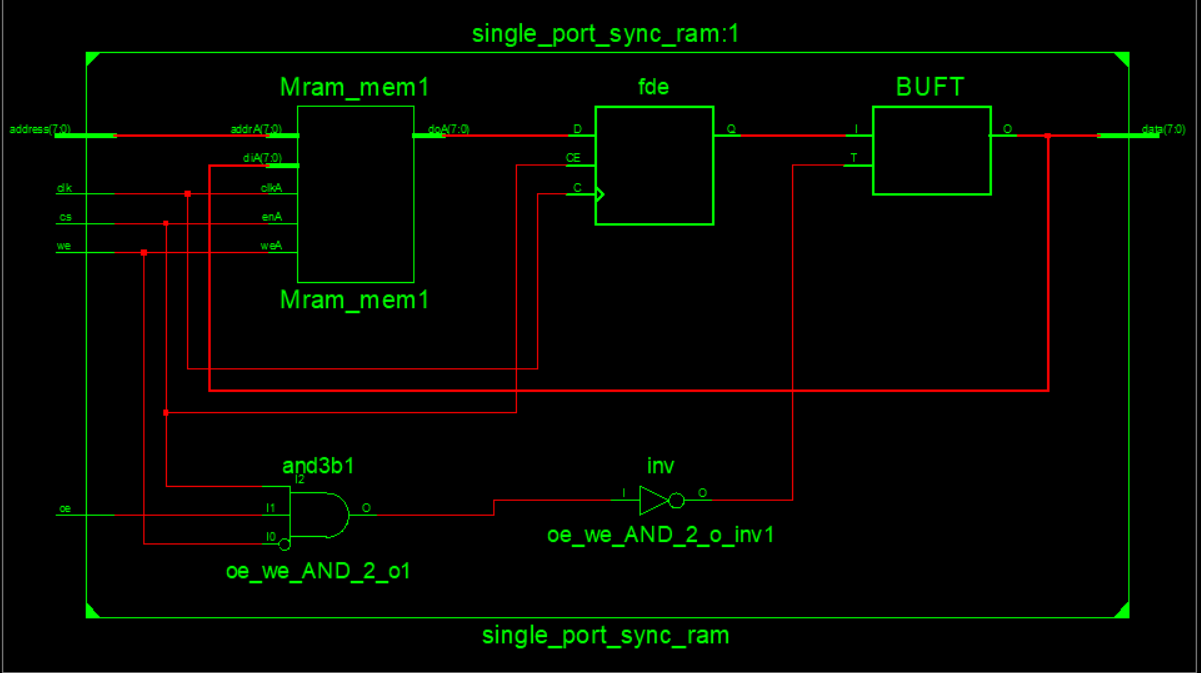
end

#20 $finish;

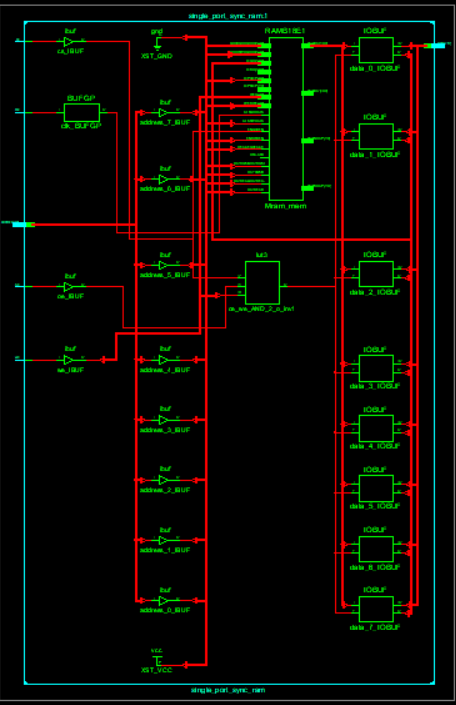
end

endmodule

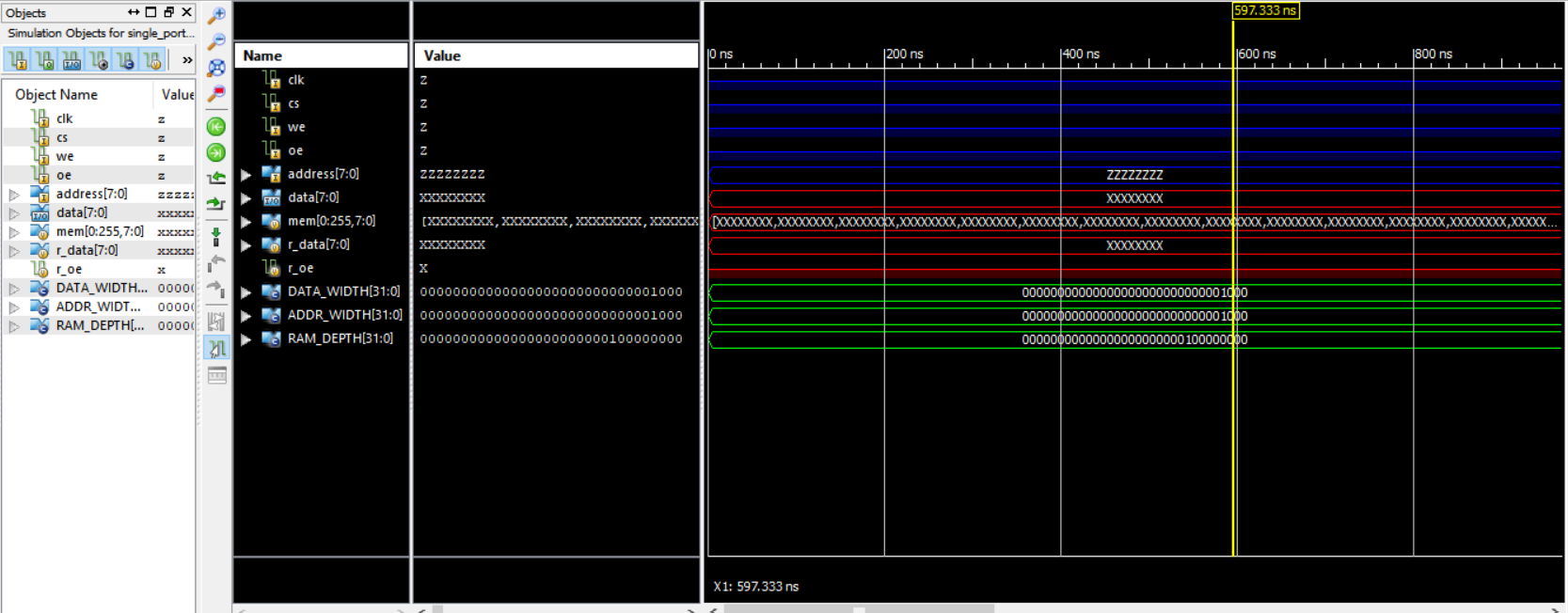
**RTL Schematics:**

****

**Technical Schematics:**

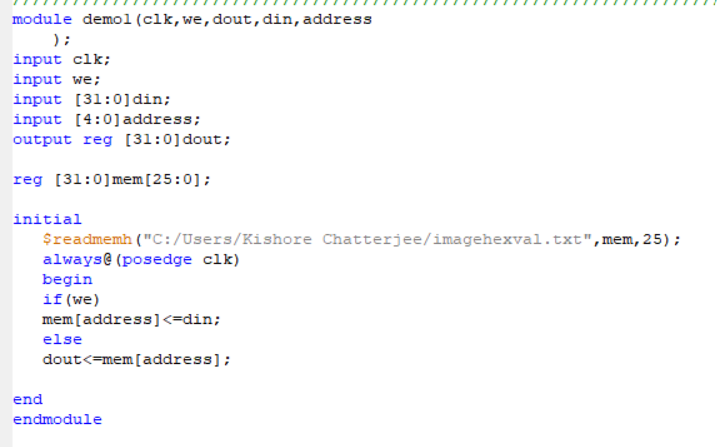
****

**Graph:**

****

**Working on Image file on verilog:**

**RAM verilog code:**

****

**Test Bench:**

module demo1tb;

// Inputs

reg clk;

reg we;

reg [31:0] din;

reg [4:0] address;

// Outputs

wire [31:0] dout;

// Instantiate the Unit Under Test (UUT)

demo1 uut (

.clk(clk),

.we(we),

.dout(dout),

.din(din),

.address(address)

);

initial begin

// Initialize Inputs

clk = 0;

we = 1;

din = 0;

address = 0;

// Wait 100 ns for global reset to finish

#100;

we = 0;

#100 address = 5'd1;

#110 address = 5'd1;

#120 address = 5'd2;

#130 address = 5'd3;

#140 address = 5'd4;

#150 address = 5'd25;

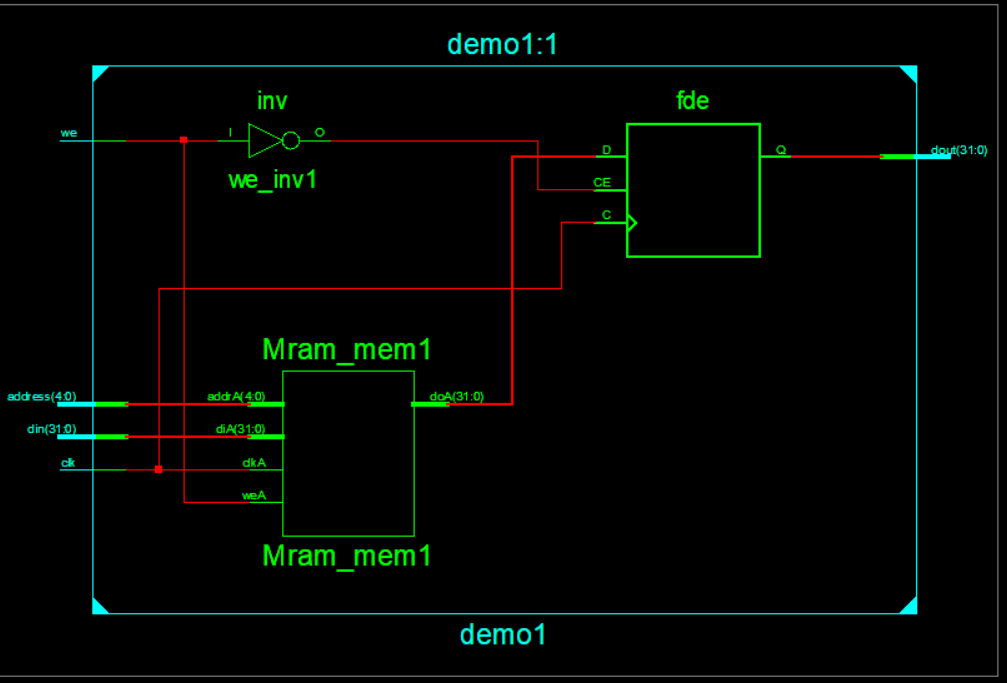
// Add stimulus here

end

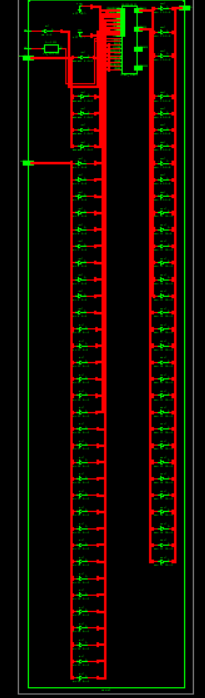
always #5 clk = ~clk;

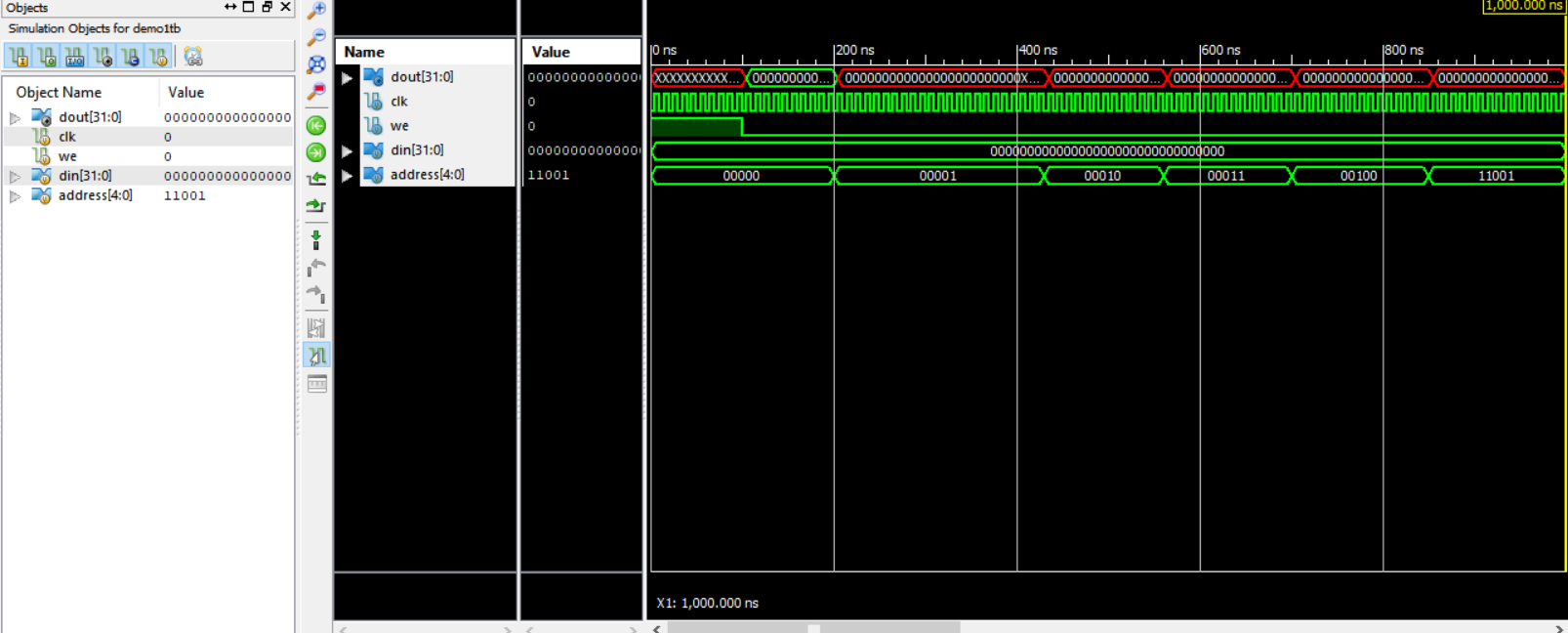
endmodule

**RTL schematics:**

****

**Technical Schematics:**

****

**Graph:** ****

**Python code for the conversion of Image to hexadecimal value:**

****

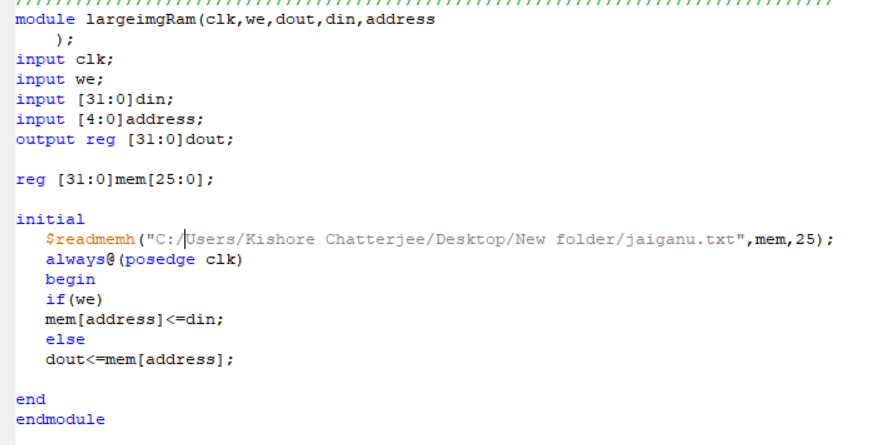
**Getting Hexadecimal Value of Image using Python:**

****

**Image and its Binary Format:**

****

**Working on RAM with Large Input Image Verilog Code:**

****

**Test Bench:**

module Gannutb;

// Inputs

reg clk;

reg we;

reg [31:0] din;

reg [4:0] address;

// Outputs

wire [31:0] dout;

// Instantiate the Unit Under Test (UUT)

largeimgRam uut (

.clk(clk),

.we(we),

.dout(dout),

.din(din),

.address(address)

);

initial begin

// Initialize Inputs

clk = 0;

we = 1;

din = 0;

address = 0;

// Wait 100 ns for global reset to finish

#100;

we = 0;

#100 address = 5'd1;

#110 address = 5'd1;

#120 address = 5'd2;

#130 address = 5'd3;

#140 address = 5'd4;

#150 address = 5'd25;

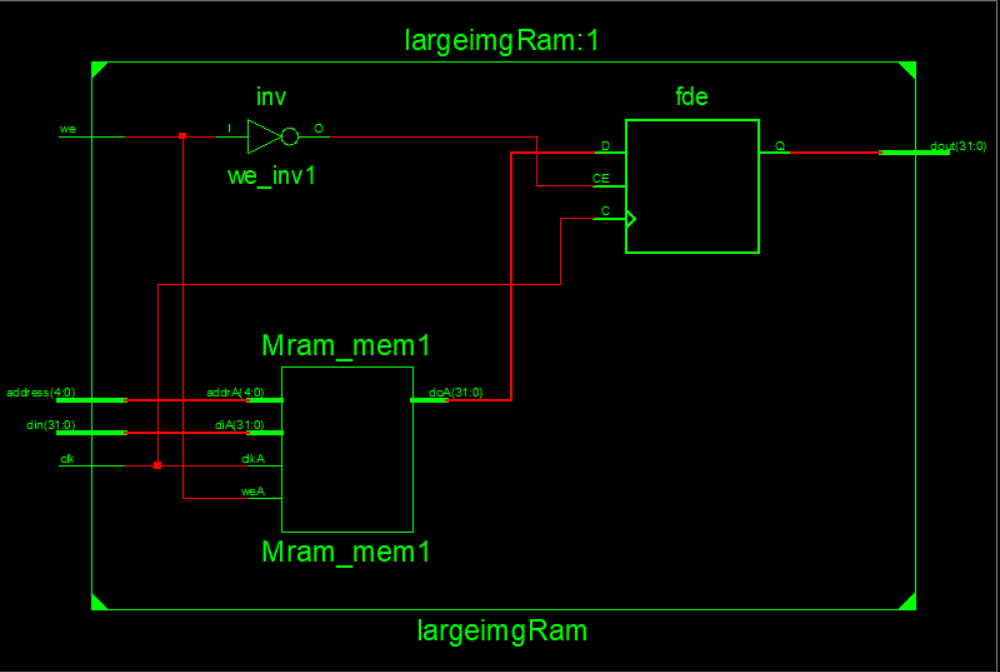
// Add stimulus here

end

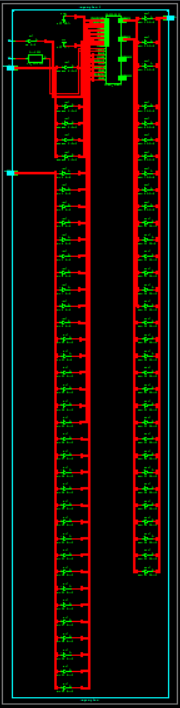
always #5 clk = ~clk;

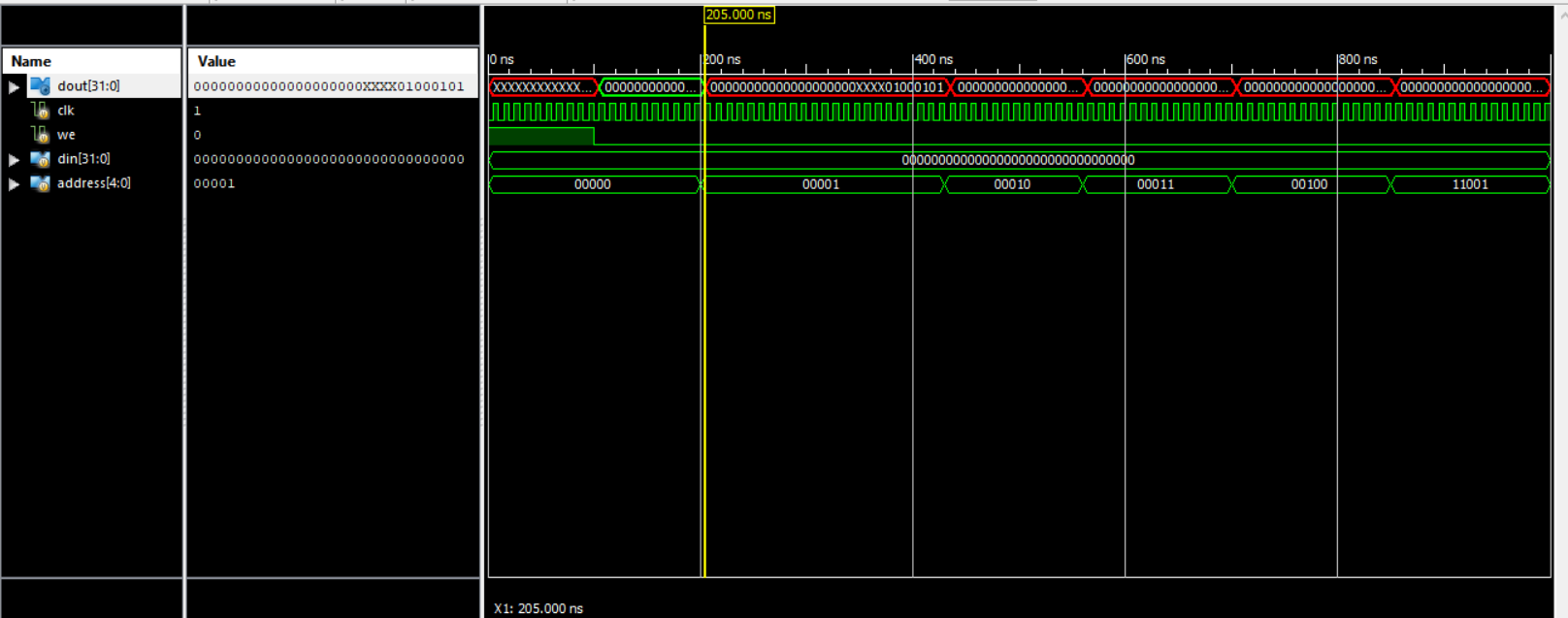
endmodule

**RTL scematics:**

****

**Technical Schematics:**

****

**Graph:** ****