**Assignment 1**

1. Write a Verilog code for a half adder adder using data flow abstraction and verify   using test bench

**CODE:**

module halfadder(a,b,sum,carry);

input a, b;

output sum, carry;

assign sum = a ^ b;

assign carry = a & b;

endmodule

**TESTBENCH:**

module half\_adder\_tb;

reg a,b;

wire sum,carry;

half\_adder dut(a,b,sum,carry);

initial

begin

$display("Testing Half Adder");

// Test Case 1

a = 0; b = 0;

#10 $display("Case 1: a=%b, b=%b, Sum=%b, Carry=%b", a, b, sum, carry);

// Test Case 2

a = 0; b = 1;

#10 $display("Case 2: a=%b, b=%b, Sum=%b, Carry=%b", a, b, sum, carry);

// Test Case 3

a = 1; b = 0;

#10 $display("Case 3: a=%b, b=%b, Sum=%b, Carry=%b", a, b, sum, carry);

// Test Case 4

a = 1; b = 1;

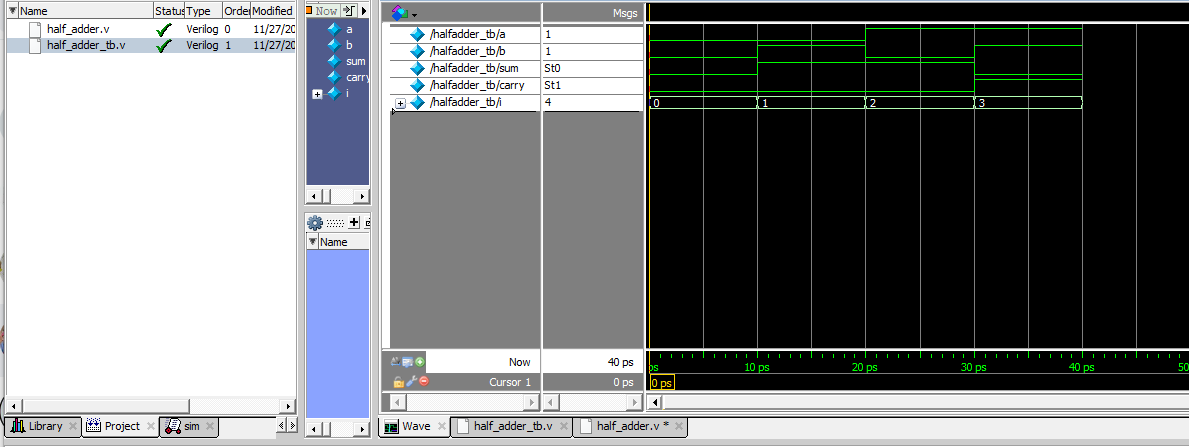
#10 $display("Case 4: a=%b, b=%b, Sum=%b, Carry=%b", a, b, sum, carry);

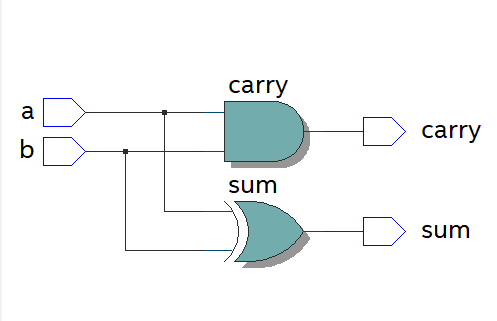
// Stop simulation

#10 $finish;

end

endmodule





**2. Write a Verilog for a 1 bit full adder using 2 half adder and 1 or gate and verify   using test bench**

**CODE:**

module half\_adder(a,b,sum,carry);

input a,b;

output sum,carry;

//Understand the Data-flow abstraction

assign sum = a ^ b;

assign carry = a & b;

endmodule

module full\_adder(a\_in,b\_in,c\_in,sum\_out,carry\_out);

input a\_in,b\_in,c\_in;

output sum\_out,carry\_out;

wire w1,w2,w3;

half\_adder HA1(.a(a\_in),.b(b\_in),.sum(w1),.carry(w2));

half\_adder HA2(.a(w1),.b(c\_in),.sum(sum\_out),.carry(w3)); or or1(carry\_out,w3,w2);

endmodule

**TESTBENCH:**

module full\_adder\_tb;

reg a,b,cin;

wire sum,carry;

integer i;

full\_adder DUT(a,b,cin,sum,carry);

initial

begin

a = 1'b0;

b = 1'b0;

cin = 1'b0;

end

initial

begin

for(i=0;i<8;i=i+1)

begin

{a,b,cin}=i;

#10;

end

end

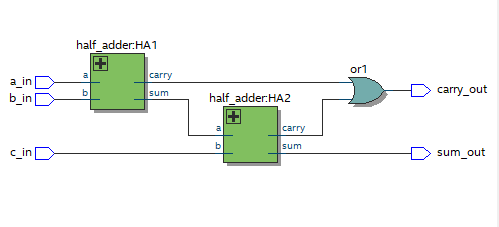
initial

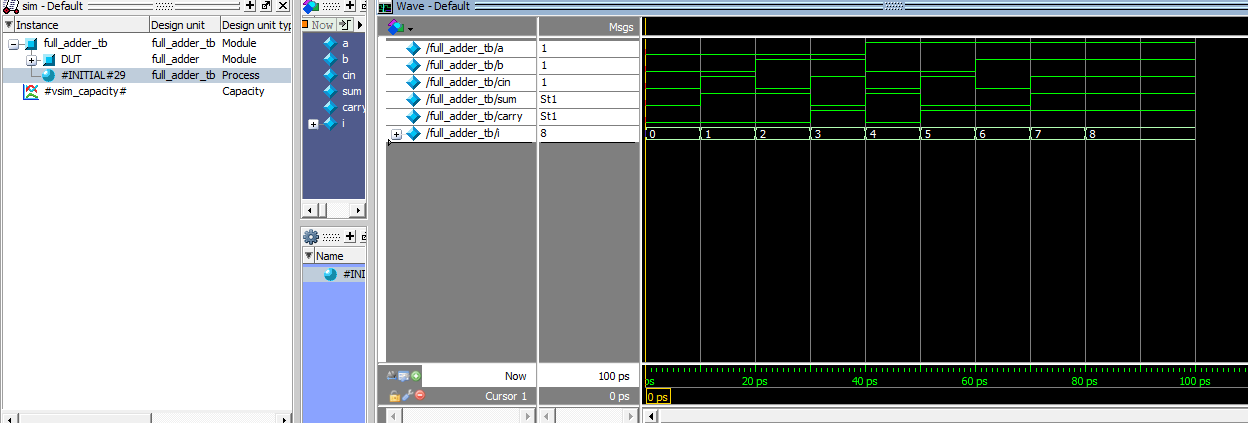
$monitor("Input a=%b, b=%b, c=%b, Output sum =%b, carry=%b",a,b,cin,sum,carry);

initial

#100 $finish;

endmodule





**3. Write a verilog code  for a full  adder adder using data flow abstraction and verify   using testbench**

**CODE:**

module full\_adder2(a,b,c,sum,carry);

input a,b,c;

output sum,carry;

assign sum=a^b^c;

assign carry=(a&b)|(b&c)|(a&c);

endmodule

**TESTBENCH**

module full\_adder2\_tb;

reg a,b,c;

wire sum,carry;

integer i;

full\_adder2 dut(a,b,c,sum,carry);

initial

begin

a=1'b0;b=1'b0;c=1'b0;

end

initial

begin

for(i=0;i<8;i=i+1)

begin

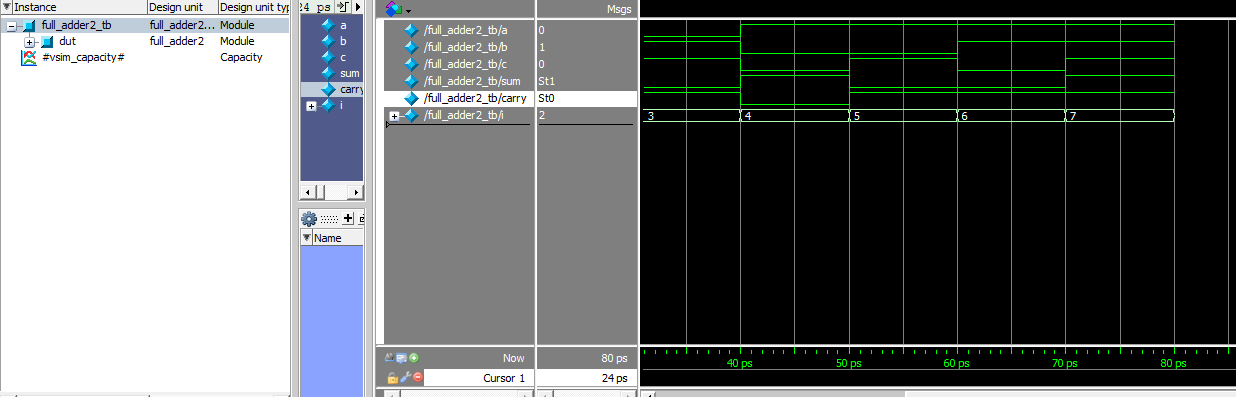
{a,b,c}=i;

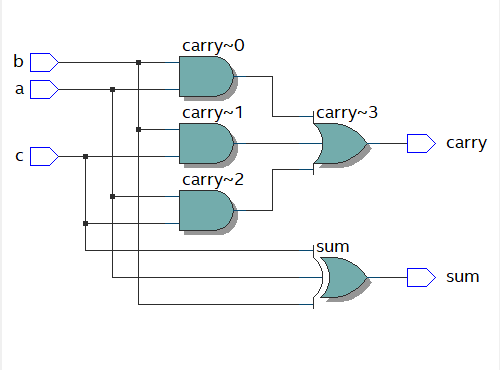
#10;

end

end

endmodule





**4. Write an RTL and test bench for 2:4 decoder using data flow abstraction**

**CODE:**

module decoder\_2\_4(input enable, input [1:0] in, output [3:0] out);

assign out[0] = (in == 2'b00) ? enable : 1'b0;

assign out[1] = (in == 2'b01) ? enable : 1'b0;

assign out[2] = (in == 2'b10) ? enable : 1'b0;

assign out[3] = (in == 2'b11) ? enable : 1'b0;

endmodule

**TESTBENCH:**

module decoder\_2\_4\_tb;

reg enable;

reg[1:0]in;

wire [3:0]out;

integer i;

decoder\_2\_4 dut(enable,in,out);

initial

begin

enable=1'b0;

in[0]=1'b0;

in[1]=1'b1;

end

initial

begin

enable=1'b1;

for(i=0;i<4;i=i+1)

begin

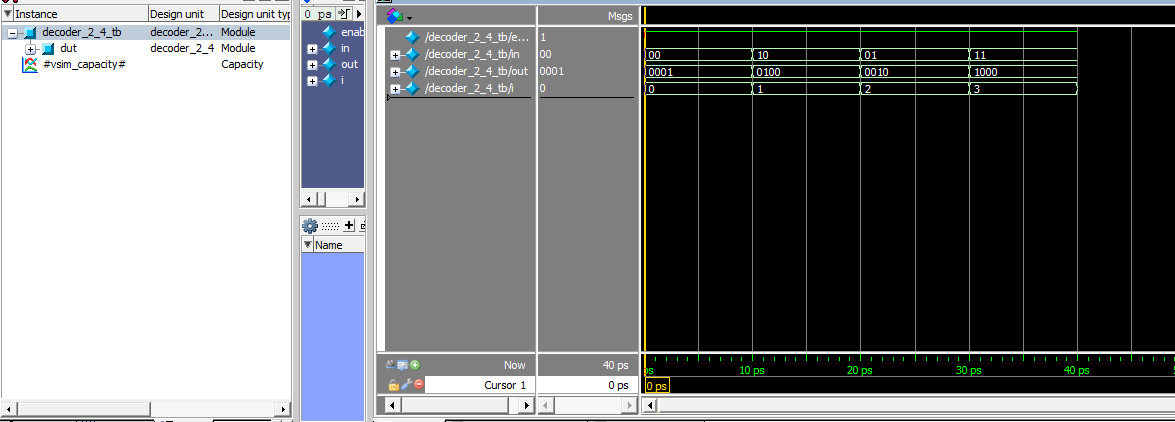
{in[0],in[1]}=i;

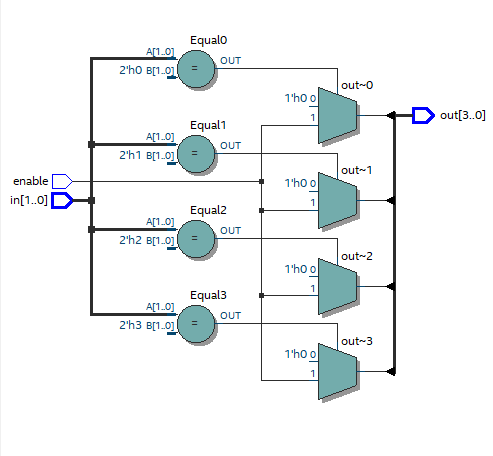
#10;

end

end

endmodule





**5. Write an RTL and test bench for a 4:1 MUX using 2:1 MUX**

**CODE:**

module mux\_4\_1(input [1:0]sel,input [3:0]in,output out);

wire out1,out2;

mux\_2\_1 i1(.a(in[0]),.b(in[1]),.sel(sel[0]),.out(out1));

mux\_2\_1 i2(.a(in[2]),.b(in[3]),.sel(sel[0]),.out(out2));

mux\_2\_1 i3(.a(out1),.b(out2),.sel(sel[1]),.out(out));

endmodule

**TESTBENCH:**

module mux\_4\_1\_tb;

reg [1:0]sel;

reg [3:0]in;

wire out;

integer i;

mux\_4\_1 i1(sel,in,out);

initial

begin

sel=2'b00;

in=4'b0000;

end

initial

begin

for(i=0;i<64;i=i+1)

begin

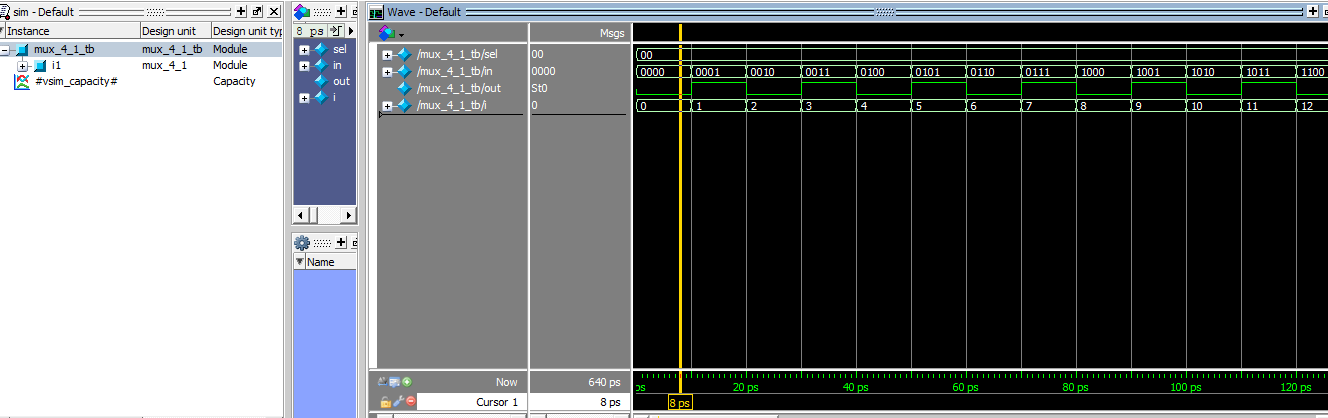
{sel,in}=i;

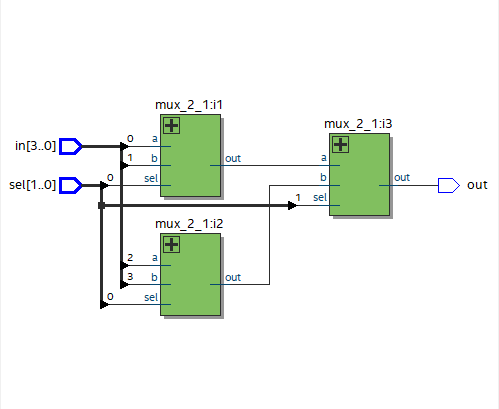
#10;

end

end

endmodule





6. Write an RTL for a bidirectional buffer and verify the same using test bench.

CODE:

module bi\_dir\_buf(send\_data,recv\_data,data\_to\_buffer);

input send\_data,recv\_data;

output[2:0] data\_to\_buffer;

wire [2:0]data\_to\_buffer;

wire [2:0] temp;

assign temp=recv\_data?data\_to\_buffer:3'bzzz;

assign data\_to\_buffer=send\_data?temp:data\_to\_buffer;

endmodule

TESTBENCH:

module bi\_dir\_buf\_tb;

// Inputs

reg send\_data, recv\_data;

// Outputs

wire [2:0] data\_to\_buffer;

// Instantiate the bidirectional buffer module

bi\_dir\_buf uut (

.send\_data(send\_data),

.recv\_data(recv\_data),

.data\_to\_buffer(data\_to\_buffer)

);

// Testbench stimulus

initial begin

$display("Testing Bidirectional Buffer");

// Test Case 1 (Send data)

send\_data = 1; recv\_data = 0;

#10 $display("Case 1: send\_data=%b, recv\_data=%b, data\_to\_buffer=%b", send\_data, recv\_data, data\_to\_buffer);

// Test Case 2 (Receive data)

send\_data = 0; recv\_data = 1;

#10 $display("Case 2: send\_data=%b, recv\_data=%b, data\_to\_buffer=%b", send\_data, recv\_data, data\_to\_buffer);

// Test Case 3 (No data transfer)

send\_data = 0; recv\_data = 0;

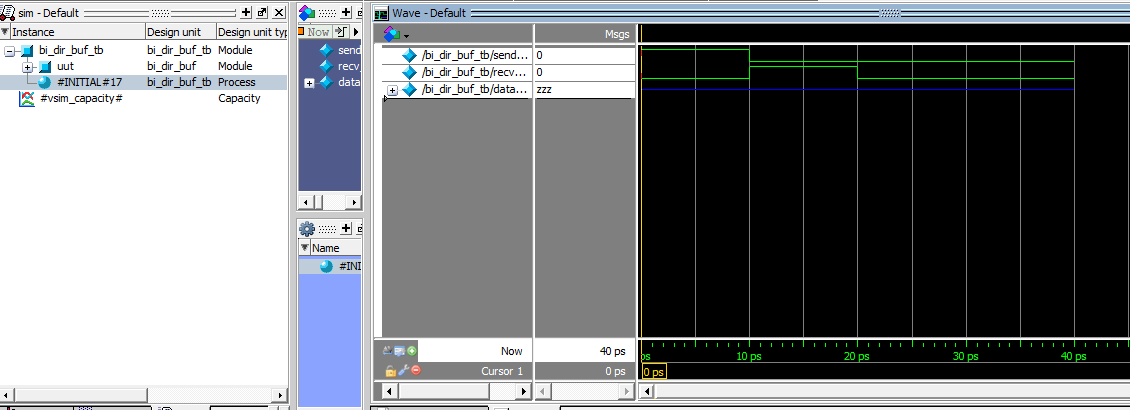
#10 $display("Case 3: send\_data=%b, recv\_data=%b, data\_to\_buffer=%b", send\_data, recv\_data, data\_to\_buffer);

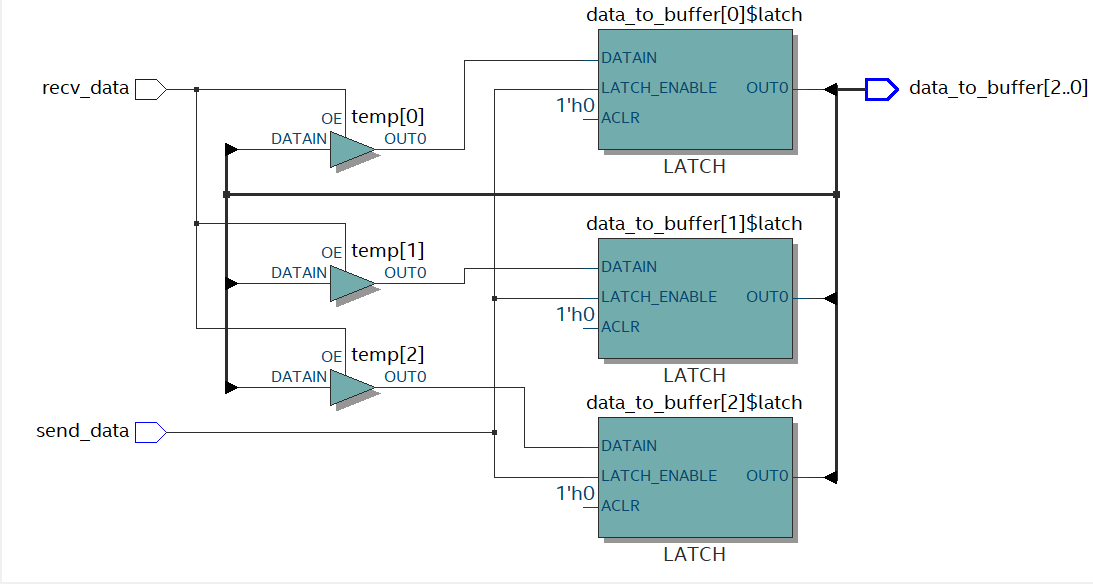
// Stop simulation

#10 $finish;

end

endmodule





**7. Write an RTL and test bench and test bench for a 4 bit Ripple carry adder using 1 bit   full adder**

**CODE:**

module full\_adder(a,b,c,sum,carry);

input a,b,c;

output sum,carry;

assign sum=a^b^c;

assign carry=(a&b)|(a&c)|(b&c);

endmodule

module ripple\_carry(input [3:0]A,input [3:0]B,input CIN,output [3:0]SUM,output CARRY);

wire C1,C2,C3;

full\_adder i1(A[0],B[0],CIN,SUM[0],C1);

full\_adder i2(A[1],B[1],C1,SUM[1],C2);

full\_adder i3(A[2],B[2],C2,SUM[2],C3);

full\_adder i4(A[3],B[3],C3,SUM[3],CARRY);

endmodule

**TESTBENCH:**

module ripple\_carry\_tb;

reg [3:0]A;

reg [3:0]B;

reg CIN;

wire SUM,CARRY;

integer i;

ripple\_carry dut(A,B,CIN,SUM,CARRY);

initial

begin

A=3'b000;

B=3'b000;

CIN=1'b0;

end

initial

begin

for(i=0;i<128;i=i+1)

begin

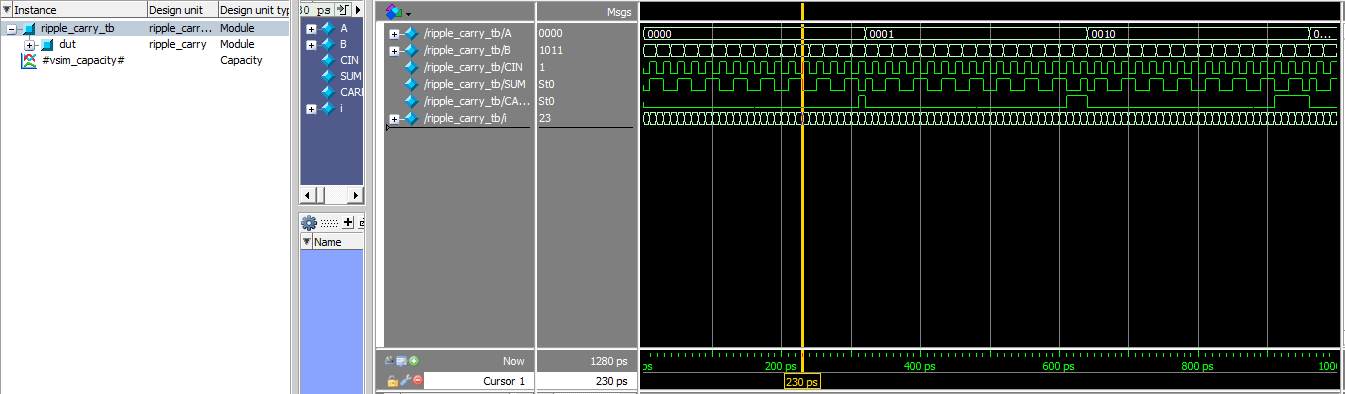
{A,B,CIN}=i;

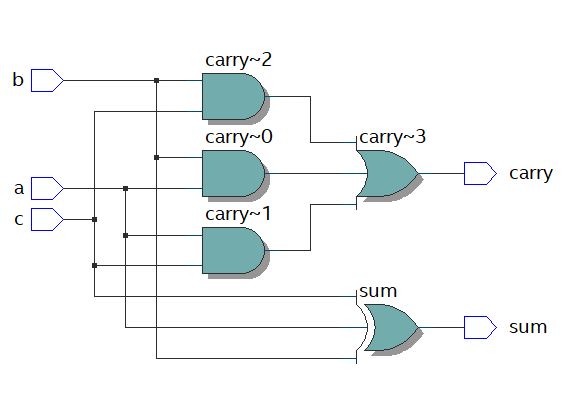
#10;

end

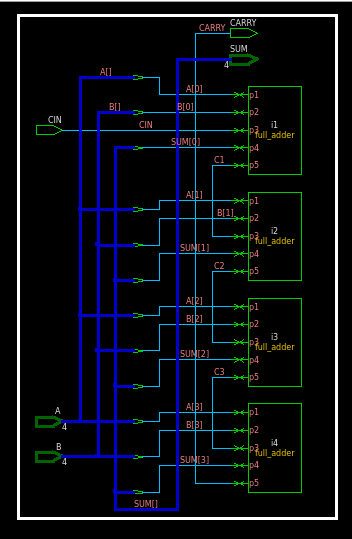
end

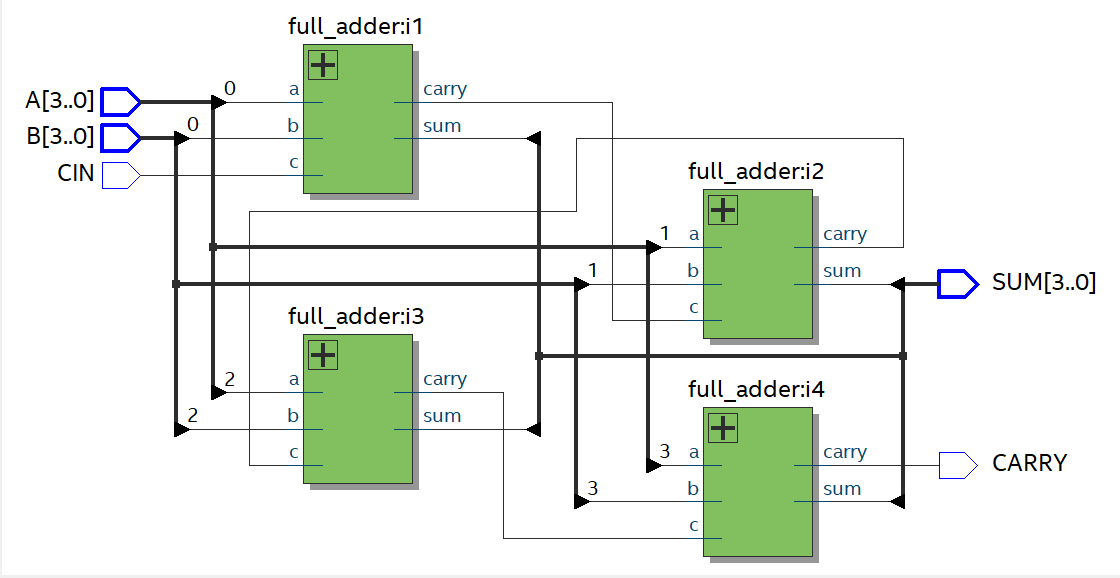
endmodule











**8. Write an RTL for 4X1 mux using decoder and tri state buffer and verify the same    using test bench**

**CODE:**

module buffer(in,control,out);

input in,control;

output out;

assign out=(control==1)?in:out;

endmodule

module decoder(input [1:0]in,output [3:0]out);

assign out[0]=(in==2'b00)?1'b1:1'b0;

assign out[1]=(in==2'b01)?1'b1:1'b0;

assign out[2]=(in==2'b10)?1'b1:1'b0;

assign out[3]=(in==2'b11)?1'b1:1'b0;

endmodule

module mux\_4\_dec\_buf(input [3:0]IN,input [1:0]SEL, output OUT);

wire [3:0]C;

wire [3:0]ot;

decoder d1(SEL,C);

buffer i1(IN[0],C[0],ot[0]);

buffer i2(IN[1],C[1],ot[1]);

buffer i3(IN[2],C[2],ot[2]);

buffer i4(IN[3],C[3],ot[3]);

assign OUT=(SEL==2'b00)?ot[0]:

(SEL==2'b01)?ot[1]:

(SEL==2'b10)?ot[2]:

(SEL==2'b11)?ot[3]:4'b0000;

endmodule

**TESTBENCH:**

module mux\_4\_dec\_buf\_tb;

reg [1:0]sel;

reg [3:0]in;

wire out;

integer i;

mux\_4\_dec\_buf i1(in,sel,out);

initial

begin

sel=2'b00;

in=4'b0000;

end

initial

begin

for(i=0;i<64;i=i+1)

begin

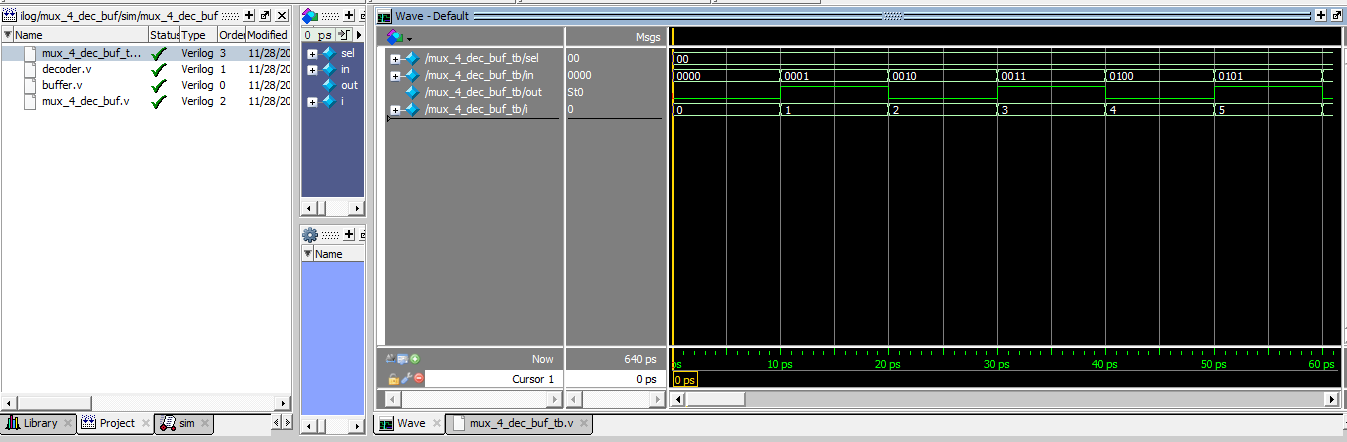
{sel,in}=i;

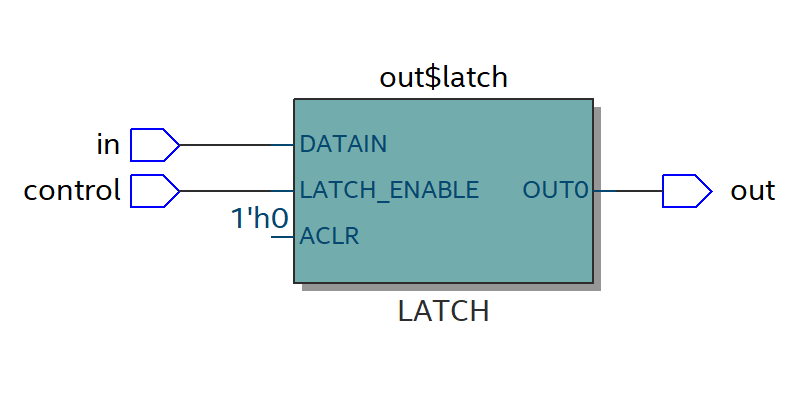
#10;

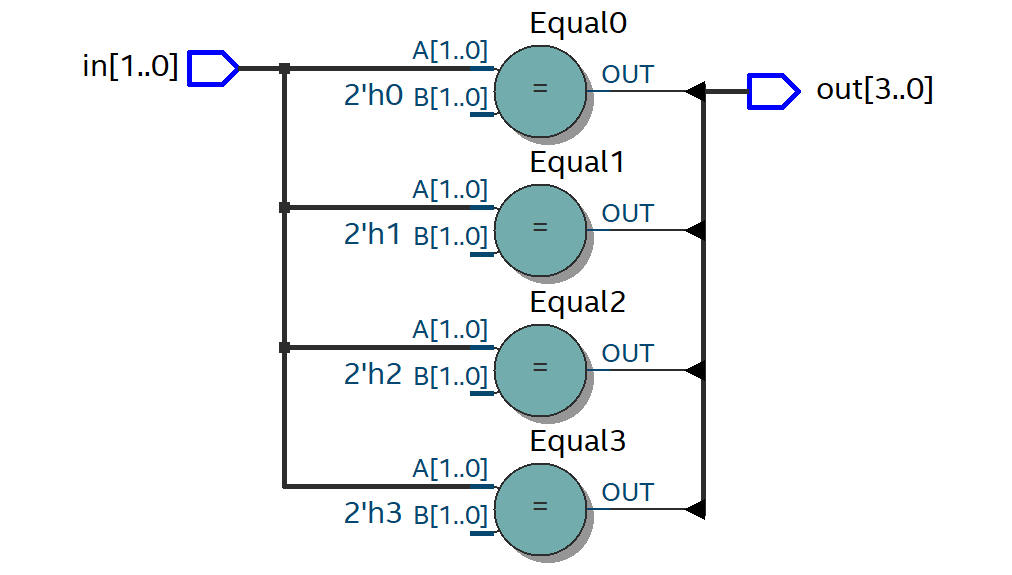
end

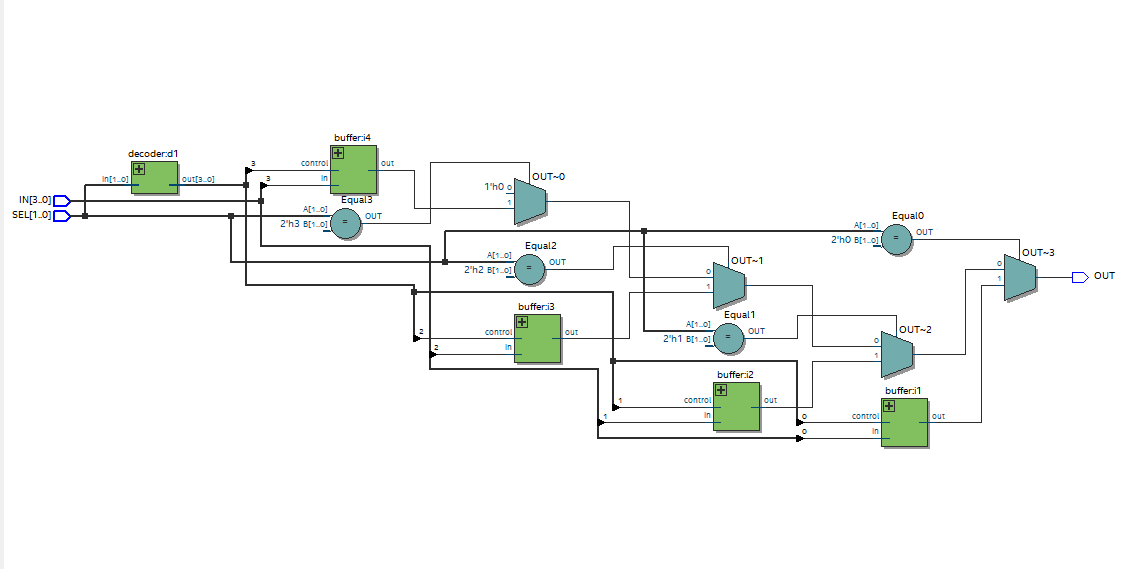
end

endmodule









9. write 8:3 priority  encoder using a structural model.

CODE:

module prio\_encoder\_8\_3(input [7:0] in, output [2:0] out);

assign out = (in[7]) ? 3'b111 :

(in[6]) ? 3'b110 :

(in[5]) ? 3'b101 :

(in[4]) ? 3'b100 :

(in[3]) ? 3'b011 :

(in[2]) ? 3'b010 :

(in[1]) ? 3'b001 :

3'b000;

endmodule

TESTBENCH:  
module prio\_encoder\_8\_3\_tb;

reg [7:0]IN;

wire [2:0]OUT;

integer i;

prio\_encoder\_8\_3 dut(IN,OUT);

initial

begin

IN=8'b0000\_0000;

end

initial

begin

for(i=0;i<256;i=i+1)

begin

IN=i;

#10;

end

end

endmodule

