**LAB 1 – Raja Aadhithan**

Design - Full adder using Half adder:

Code:

module half\_adder(input a,b,

             output sum,carry);

   //Understand the Data-flow abstraction

   assign sum   = a ^ b;

   assign carry = a & b;

endmodule

module full\_adder( 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(c\_in), .b(w1), .sum(sum\_out), .carry(w3));

or or1(carry\_out, w2,w3);

endmodule

Test bench:

module full\_adder\_tb();

   //Testbench global variables

   reg   a,b,cin;

   wire  sum,carry;

   //Variable for loop iteration

   integer i;

   //Step1 : Instantiate the full adder with order based port mapping

full\_adder DUT(a,b,cin,sum,carry);

   //Process to initialize the variables at 0ns

   initial

      begin

         a   = 1'b0;

    b   = 1'b0;

    cin = 1'b0;

      end

   //Process to generate stimulus using for loop

   initial

      begin

    for (i=0;i<8;i=i+1)

       begin

          {a,b,cin}=i;

          #10;

       end

      end

   //Process to monitor the changes in the variables

   initial

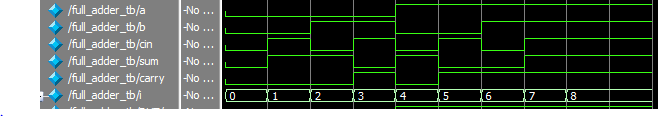
      $monitor("Input a=%b, b=%b, c=%b, Output sum =%b, carry=%b",a,b,cin,sum,carry);

   //Process to terminate simulation after 100ns

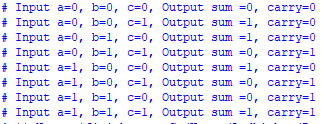
   initial #100 $finish;

endmodule

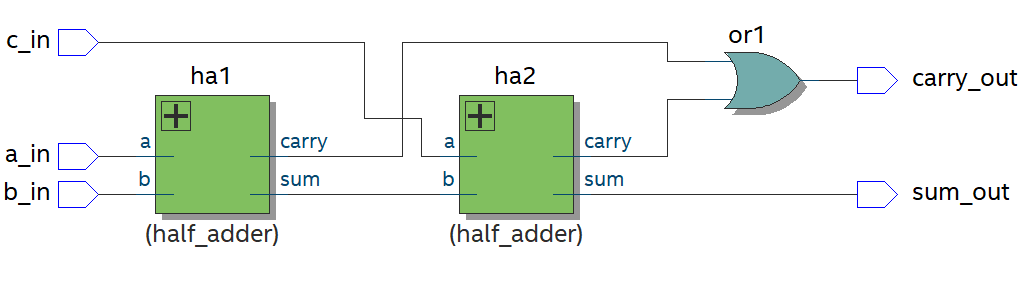
Wave:



Output:



RTL:



**Exercises:**

Design : Ripple adder

Code:

module ripple\_adder(input [3:0] a,b, input cin,

             output[3:0] sum,output carry);

wire [2:0] w;

full\_adder FA1(a[0],b[0],cin,sum[0],w[0]);

full\_adder FA2(a[1],b[1],w[0],sum[1],w[1]);

full\_adder FA3(a[2],b[2],w[1],sum[2],w[2]);

full\_adder FA4(a[3],b[3],w[2],sum[3],carry);

endmodule

Test bench:

module ripple\_adder\_tb();

reg [3:0]a,b;

reg c;

wire [3:0]s;

wire cy;

integer i;

initial begin

a=0;

b=0;

c=0; end

ripple\_adder DUT(a,b,c,s,cy);

initial begin

for (i=0;i<16;i=i+1)

begin

a=i;

b=i;

c=!c;

#10;

end

end

initial

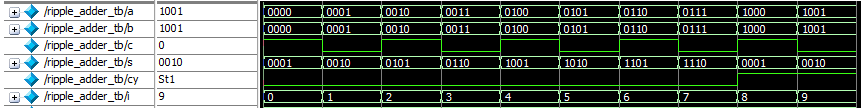
      $monitor("Input a=%b, b=%b, c=%b, Output sum =%b, carry=%b",a,b,c,s,cy);

   //Process to terminate simulation after 100ns

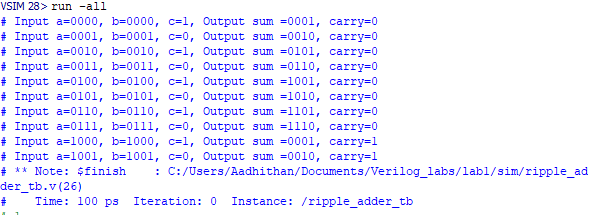
   initial #100 $finish;

endmodule

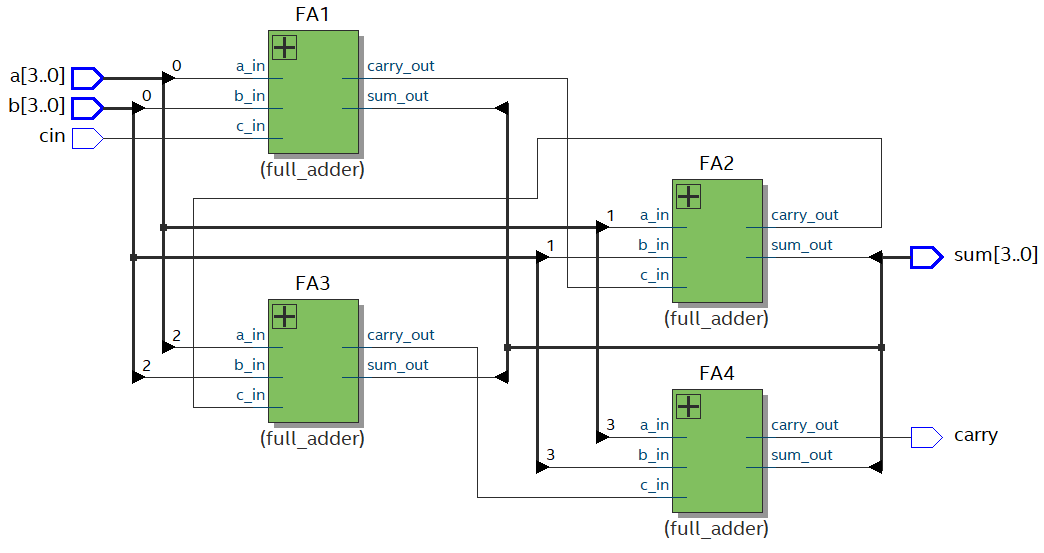
Wave:



Output:



RTL:



Design : 4:1 mux using 2:1 mux

Code:

module mux2(input a,b,s, output x);

assign x = s ? b : a;

endmodule

module mux4(input a,b,c,d,s1,s2,output x);

wire w1,w2;

mux2 mu1(a,b,s2,w1);

mux2 mu2(c,d,s2,w2);

mux2 mu3(w1,w2,s1,x);

endmodule

Test bench:

module mux4\_tb();

reg a,b,c,d,s1,s2;

wire x;

integer i;

mux4 DUT(a,b,c,d,s1,s2,x);

initial begin

   a=0;b=0;c=0;d=0;s1=0;s2=0;

end

initial begin

    for(i=0;i<64;i=i+1)

    begin

        {s1,s2,a,b,c,d}=i;

        #10;

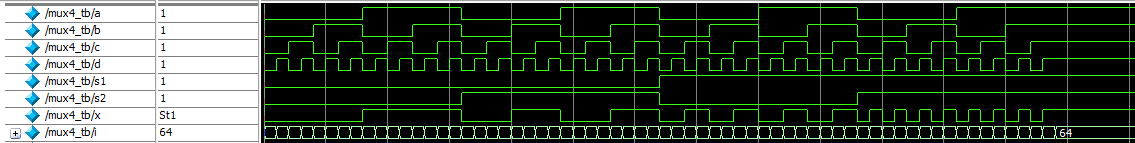
end

end

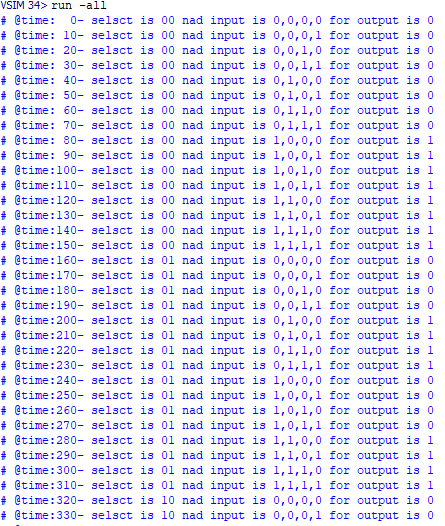
initial #1000 $finish;

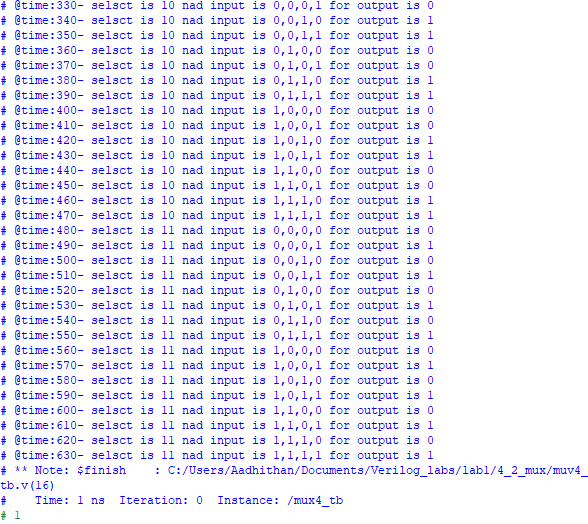
endmodule

Wave:

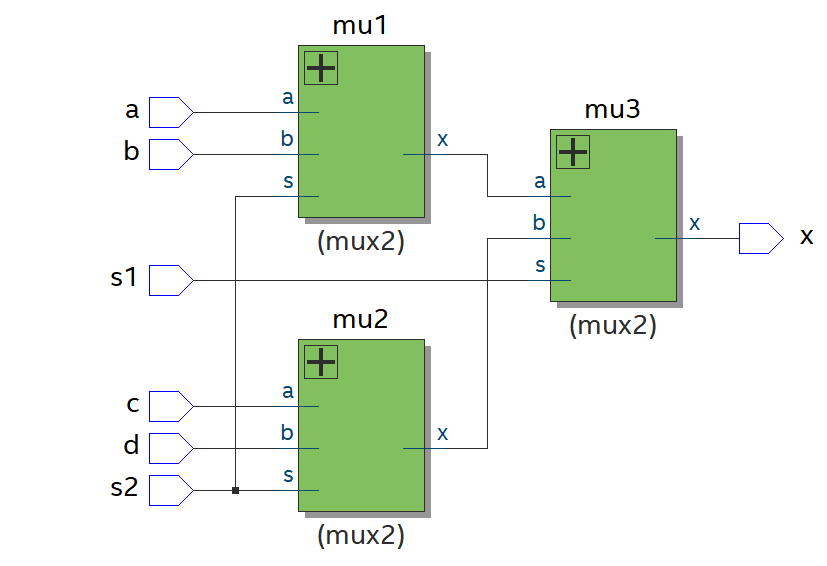


Output:





RTL:



**Assignment:**

Design: RTL for full adder.

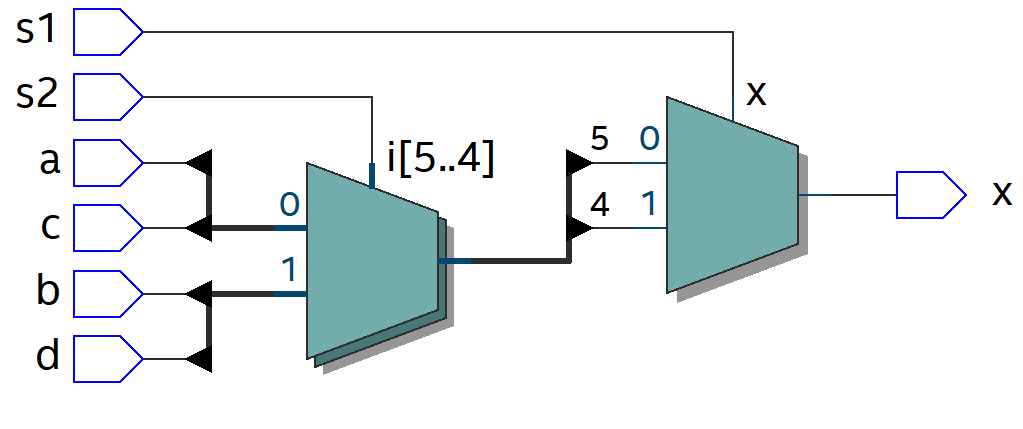
Code:

module dataflow\_fulladder(input a,b,c,d,s2,s1, output x);

assign x = s1 ? (s2 ? d : c) : (s2 ? b :a );

endmodule

RTL:



Design: 2x4 decoder.

Code:

module decoder(

    input [1:0] in,

    output [3:0] out

    );

  assign out[0] = (~in[1]) & (~in[0]) ;

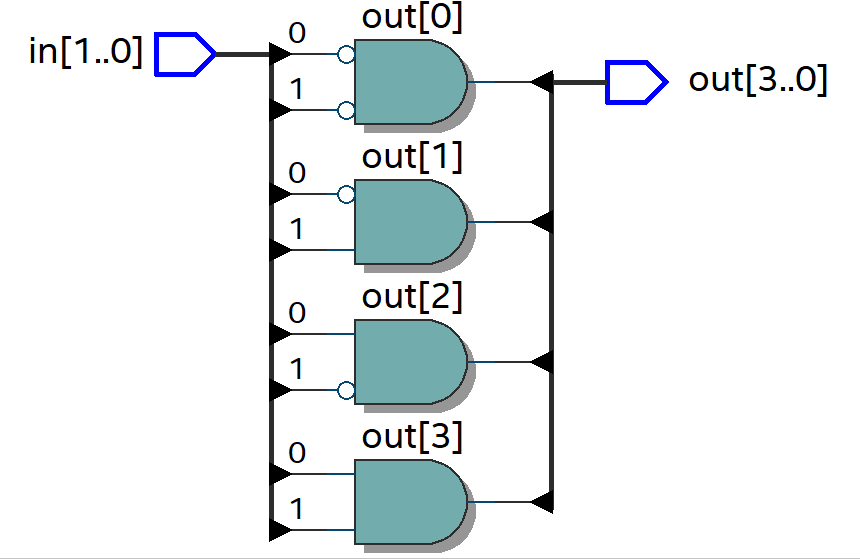
  assign out[1] = ( in[1]) & (~in[0]) ;

  assign out[2] = (~in[1]) & ( in[0]) ;

  assign out[3] = ( in[1]) & ( in[0]) ;

endmodule

RTL:



Design: 8x3 priority encoder.

Code:

module encoder(input [7:0]in , output [2:0] out);

wire[4:0] x;

wire[5:0] n;

not(n[5],in[5]);

not(n[4],in[4]);

not(n[3],in[3]);

not(n[2],in[6]);

not(n[1],in[2]);

or(out[2],in[7],in[6],in[5],in[4]);

and(x[0],in[3],n[5],n[4]);//!5,!4,3

and(x[1],n[5],n[4],n[3],in[2]);//!5,!4,!3,2

and(x[2],x[0],n[2]);//!6,!5,!4,3

and(x[3],n[2],in[5]);//!6,5

and(x[4],n[2],n[5],n[4],n[3],n[1],in[1]);//!6,!5,!4,!3,!2,1

or(out[1],in[7],in[6],x[0],x[1]);

or(out[0],in[7],x[3],x[4],x[2]);

endmodule

Test bench:

module encoder\_tb();

reg [7:0] x;

wire [2:0] y;

integer i;

encoder dat(x,y);

initial begin

    for(i=0; i<8; i = i+1)

    begin

        x = 2\*\*i;

        #10;

        x=2\*i;

        #10;

    end

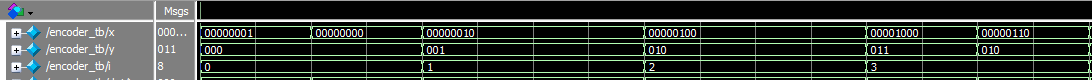
$finish;

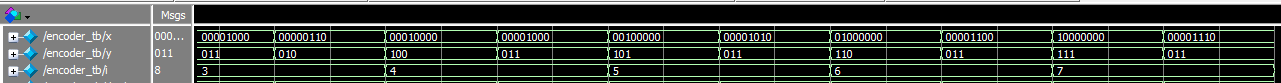
end

initial $monitor("@ time: %3dps the input is %8b output is %3b",$time,x,y);

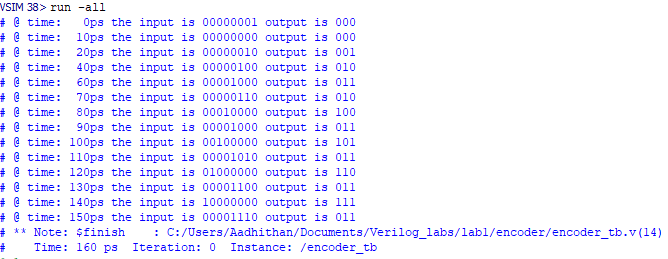
endmodule

Wave:

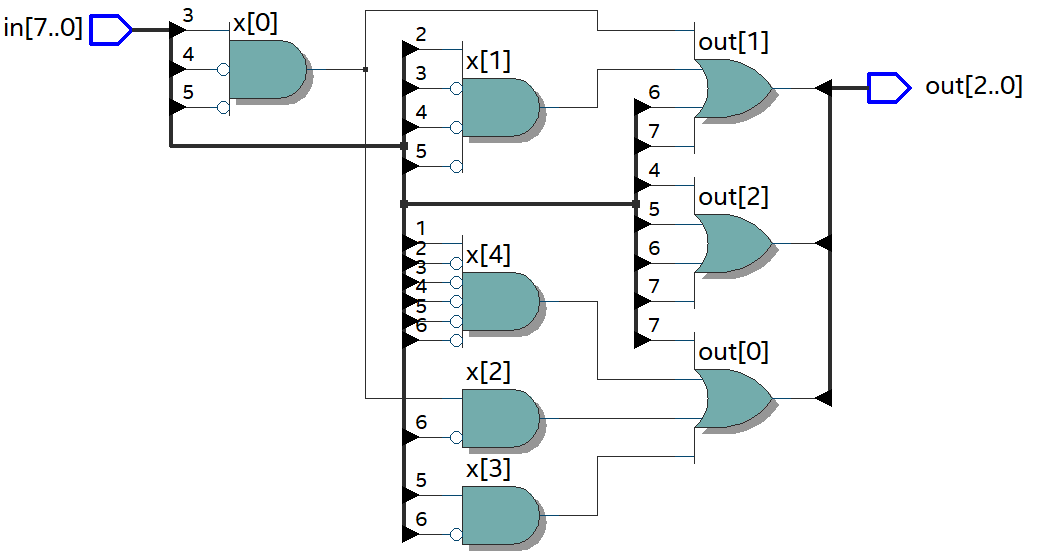




Output:



RTL:



Design: 4x1 mux.

Code:

module mux\_dec(input A,B,C,D, input [1:0] in, output Y);

wire [3:0] out;

reg x;

  assign out[0] = (~in[1]) & (~in[0]) ;

  assign out[1] = ( in[1]) & (~in[0]) ;

  assign out[2] = (~in[1]) & ( in[0]) ;

  assign out[3] = ( in[1]) & ( in[0]) ;

  always@(\*)begin

      case(in)

      2'b00:x <= out[0] ? A : 1'bz;

      2'b01:x <= out[2] ? B : 1'bz ;

      2'b10:x <= out[1] ? C : 1'bz ;

      2'b11:x <= out[3] ? D : 1'bz ;

      endcase

  end

assign Y = x;

endmodule

Test bench:

module mux4\_tb();

reg a,b,c,d;

reg [1:0] s;

wire x;

integer i;

mux\_dec DUT(a,b,c,d,s,x);

initial begin

   a=0;b=0;c=0;d=0;s = 0;

end

initial begin

    for(i=0;i<64;i=i+1)

    begin

        {s,a,b,c,d}=i;

        #10;

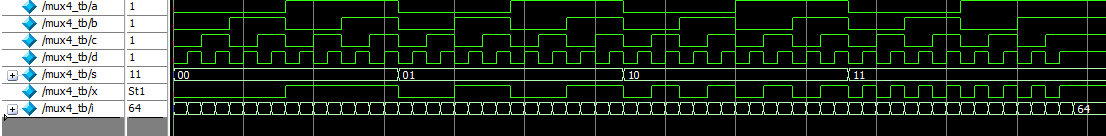
end

end

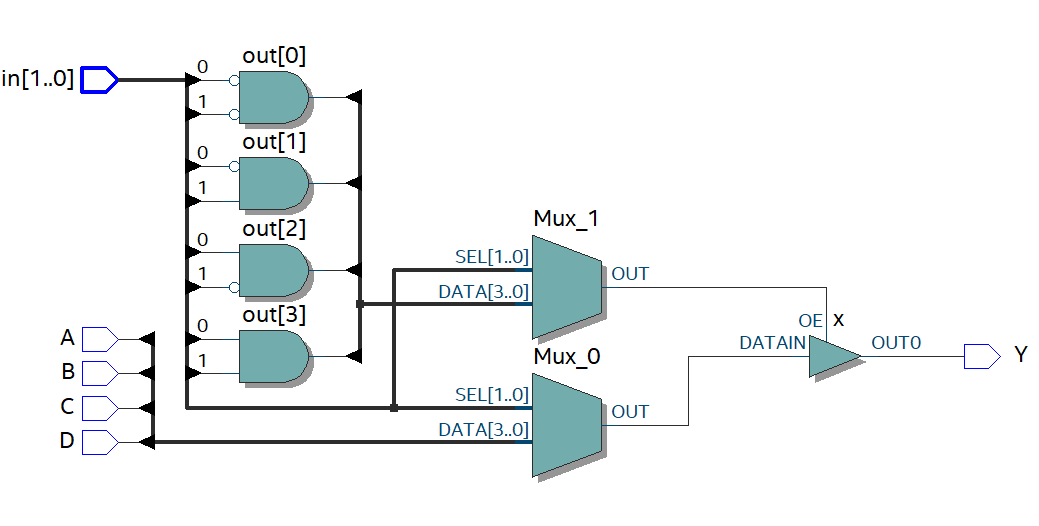
initial #1000 $finish;

endmodule

Wave:



RTL:



Design: Bidirectional buffer

Code:

module bufer(inout a,b, input c);

bufif1(b,a,c);

bufif0(a,b,c);

endmodule

Testbench:

module bufer\_tb();

wire a,b;

reg c,ta,tb;

integer i;

bufer dut(a,b,c);

initial begin

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

begin

        {ta,tb,c}=i[2:0];

        #10;

end

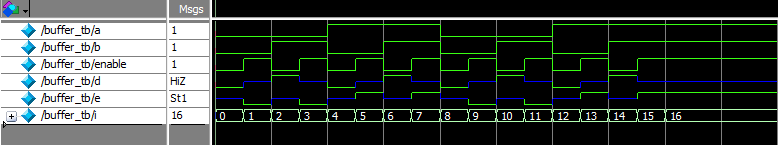
end

assign a = c ? ta : 1'bz;

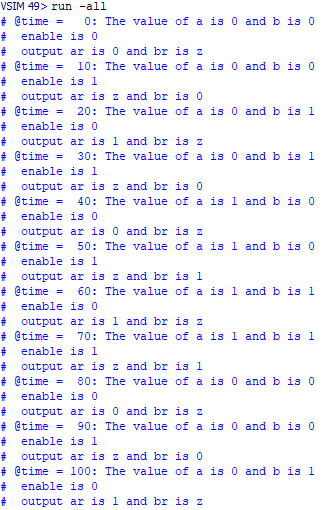
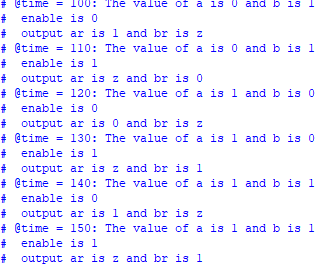
assign b = c ? 1'bz : tb;

endmodule

Wave:



Output:

RTL:

