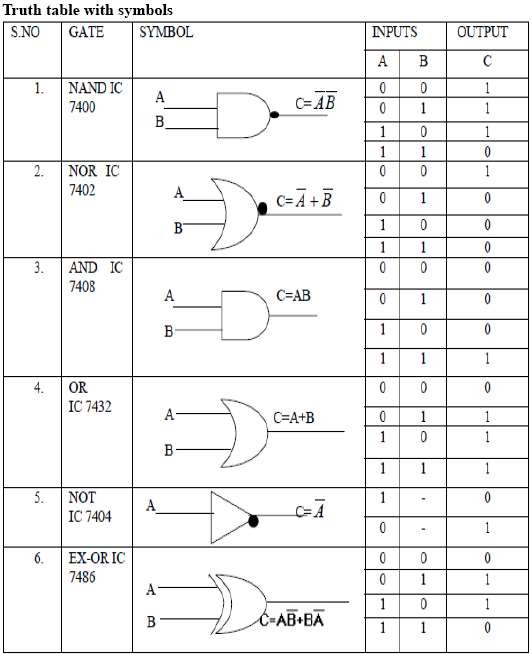
|  |
| --- |
| **VERILOG PROGRAMING TUTORIAL 1** |
|  |

**IMPLEMENTATION OF BASIC COMBINATION LOGIC CIRCUIT USING VERILOG IN FPGA KIT**

### Experiment 1

##### AIM: WRITE HDL CODE TO REALIZE ALL LOGIC GATES

A logic gate performs a logical operation on one or more logic inputs and produces a single logic output. The logic normally performed is boolean logic and is most commonly found in digital circuits



a) Write the Verilog module implement the all the basic logic gates using Structural Model :

module allgate ( a, b, yand,yor,ynot,ynand,ynor,yxor,yxnor );

input a,b;

output yand, yor, ynot, ynand, ynor, yxor, yxnor;

and G1(yand,a,b); // AND Operation

or G2(yor,a, b); // OR Operation

not G3(ynot,a) ; // NOT Operation

nand G4 (ynand,a,b); // NAND Operation

nor G5(ynor,a,b); //NOR Operation

xor G6(yxor,a,b); //XOR Operation

xnor G7(yxnor,a,b); //XNOR Operation

endmodule // END of the module

### Write the testbench to implement these modules in DE2-FPGA Kit

module lab1\_ex1(SW,LEDG,LEDR);

input[17:0] SW;

output[7:0] LEDG;

output[17:0] LEDR;

assign LEDR=SW;

allgate DUT(SW[1],SW[2],LEDG[5],LEDG[4],LEDG[3],LEDG[2],LEDG[1],LEDG[0]);

endmodule

b) Write the Verilog module implement the all the basic logic gates using Dataflow model:

module allgate ( a, b, yand,yor,ynot,ynand,ynor,yxor,yxnor );

input a,b;

output yand, yor, ynot, ynand, ynor, yxor, yxnor;

assign yand = a & b; // AND Operation

assign yor = a | b; // OR Operation

assign ynot = ~a ; // NOT Operation

assign ynand = ~(a & b); // NAND Operation

assign ynor = ~(a | b); //NOR Operation

assign yxor = a ^ b; //XOR Operation

assign yxnor =~(a^b); //XNOR Operation

endmodule // END of the module

### Write the testbench to implement these modules in DE2-FPGA Kit

module lab1\_ex1(SW,LEDG,LEDR);

input[17:0] SW;

output[7:0] LEDG;

output[17:0] LEDR;

assign LEDR=SW;

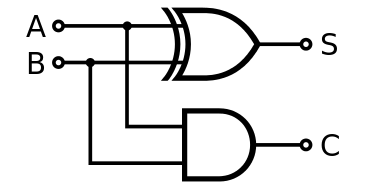
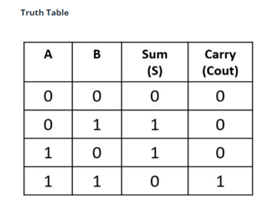
allgate DUT(SW[1],SW[2],LEDG[5],LEDG[4],LEDG[3],LEDG[2],LEDG[1],LEDG[0]);

endmodule

**EXPERIMENT 2**

**AIM: WRITE VERILOG HDL CODES TO SIMULATE AND IMPLEMENT THE** HALF ADDER

C**IRCUIT:**

**Table 1**. Half Adder Dataflow Model

module half\_adder\_structeral(input a, b, output s, Cout);

xor G1(s,a,b);

and G2(Cout,a,b);

endmodule

module half\_adder\_dataflow(input a, b, output s, Cout);

assign s = a ^ b;

assign Cout = a & b;

endmodule

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

output sum,carry;

input a,b;

reg sum,carry;

always @(a,b)

begin

sum <= a ^ b;

carry <= a&b ;

end

endmodule

module Half\_Adder\_test\_FPGA(SW,LEDG,LEDR);

input [17:0] SW;

output [7:0] LEDG;

output[17:0] LEDR;

assign LEDR=SW;

half\_adder\_dataflow HA\_DTFL\_FPGA(SW[1],SW[0], LEDG[1], LEDG[0]);

half\_adder\_structeral HA\_STRU\_FPGA(SW[3],SW[2], LEDG[3], LEDG[2]);

half\_adder\_structeral HA\_BHVR\_FPGA(SW[5],SW[4], LEDG[5], LEDG[4]);

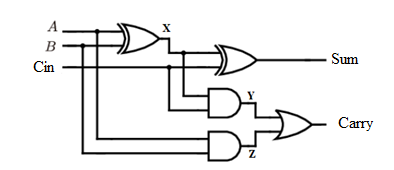
endmodule

TESTING RESULTS:

|  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | Sum | Carry | Sum (Sim) | Carry (Sim) | Sum (FPGA) | Carry (FPGA) |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |

**EXPERIMENT 3**

**AIM: WRITE VERILOG HDL CODES TO SIMULATE AND IMPLEMENT THE FULL ADDER CIRCUIT:**

** A table of input output

Description automatically generated**

module full\_adder\_STRU(a,b,cin,Cum,Carry);

output Sum,Carry;

input A,B,Cin;

wire x,y,z;

xor g1(x,A,B);

xor g2(Sum,x,Cin);

and g3(y,x,Cin);

and g4(z,A,B);

or g5(Carry,x,y);

endmodule

module full\_adder\_DTFL(input a, b, cin, output S, Cout);

assign S = a ^ b ^ cin;

assign Cout = (a & b) | (b & cin) | (a & cin);

endmodule

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

output sum,carry;

input a,b,c;

reg sum,carry;

always @ (a,b,c)

begin

sum <= a^ b^c;

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

end

endmodule

module Full\_Adder\_test\_FPGA(SW,LEDG,LEDR);

input [17:0] SW;

output [7:0] LEDG;

output [17:0] LEDR;

assign LEDR=SW;

full\_adder\_DTFL DUT1(SW[2],SW[1],SW[0], LEDG[1], LEDG[0]);

full\_adder\_STRU DUT2(SW[5],SW[4],SW[3], LEDG[3], LEDG[2]);

full\_adder\_behavior DUT3(SW[8],SW[7],SW[6], LEDG[5], LEDG[4]);

endmodule

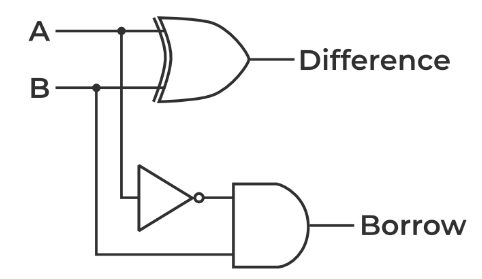
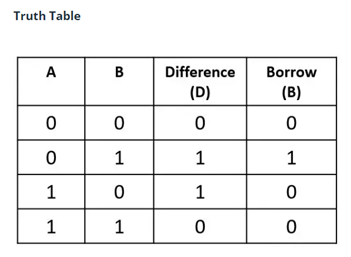
TESTING RESULTS:

**Table 4.** Full Adder

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| A | B | Cin | Sum | Carry | Sum (Sim) | Carry (Sim) | Sum (FPGA) | Carry (FPGA) |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 0 | 1 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 0 | 0 | 1 | 0 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 | 1 |

**EXPERIMENT 4**

**AIM: WRITE VERILOG HDL CODES TO SIMULATE AND IMPLEMENT THE HALF SUBTRACTOR CIRCUIT:**

****  

module half\_sub\_structeral(input a, b, output dif, bor);

xor(dif,a,b);

and(bor,~a,b);

endmodule

module half\_sub\_dataflow(input a, b, output dif, bor);

assign dif = a ^ b;

assign bor = ~a & b;

endmodule

module half\_sub\_behaviour(a,b,diff,borrow);

output diff, borrow;

input a, b;

reg diff,borrow;

always @(a,b)

begin

diff = a ^ b;

borrow =(~a&b);

end

endmodule

module Half\_Sub\_test\_FPGA(SW,LEDG,LEDR);

input[17:0] SW;

output[7:0] LEDG;

output[17:0] LEDR;

assign LEDR=SW;

half\_sub\_dataflow HB\_DTFL\_FPGA(SW[1],SW[0], LEDG[1], LEDG[0]);

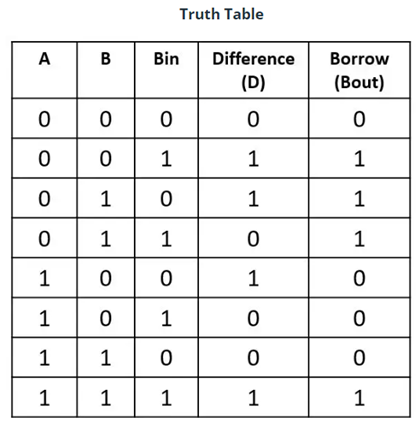
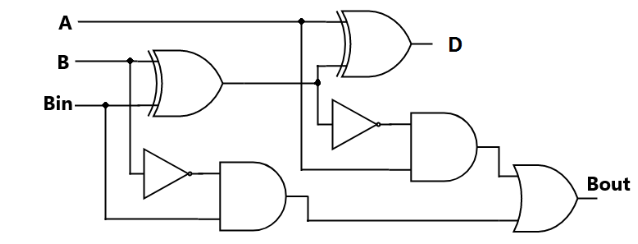
half\_sub\_structeral HB\_STRU\_FPGA(SW[3],SW[2], LEDG[3], LEDG[2]);

half\_sub\_behaviour HB\_BHVR\_FPGA(SW[5],SW[4], LEDG[5], LEDG[4]);

endmodule

**EXPERIMENT 5**

**AIM: WRITE VERILOG HDL CODES TO SIMULATE AND IMPLEMENT THE FULL SUBTRACTOR CIRCUIT:**

****

module full\_subtractor(input a, b, Bin, output D, Bout);

assign D = a ^ b ^ Bin;

assign Bout = (~a & b) | (~(a ^ b) & Bin);

endmodule

module Full\_Sub\_test\_FPGA(SW,LEDG,LEDR);

input[17:0] SW;

output[7:0] LEDG;

output[17:0] LEDR;

assign LEDR=SW;

full\_sub\_dataflow FPGA(SW[2],SW[1],SW[0], LEDG[1], LEDG[0]);

endmodule

**EXPERIMENT 6**

**AIM: WRITE VERILOG HDL CODES TO SIMULATE AND IMPLEMENT THE RIPPLE CARRY ADDER CIRCUIT:**

**A diagram of a circuit

Description automatically generated**

module full\_adder\_STRU(a,b,cin,Cum,Carry);

output Sum,Carry;

input A,B,Cin;

wire x,y,z;

xor g1(x,A,B);

xor g2(Sum,x,Cin);

and g3(y,x,Cin);

and g4(z,A,B);

or g5(Carry,x,y);

endmodule

A diagram of a computer component

Description automatically generated

module four\_bit\_adder(cin,a,b,s,cout);

input [3:0] a,b;

input cin;

output [3:0] s;

output cout;

wire [2:0] w\_carry;

full\_adder\_STRU C1(a[0],b[0],cin,s[0], w\_carry[0]);

full\_adder\_STRU C2(a[1],b[1], w\_carry[0],s[1], w\_carry[1]);

full\_adder\_STRU C3(a[2],b[2], w\_carry[1],s[2], w\_carry[2]);

full\_adder\_STRU C4(a[3],b[3], w\_carry[2],s[3], cout);

endmodule

module Ripple\_Carry\_Adder\_test\_FPGA(SW,LEDR);

input [17:0] SW;

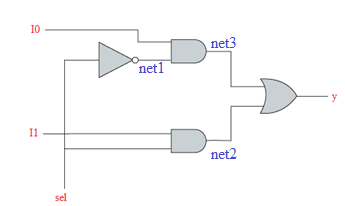
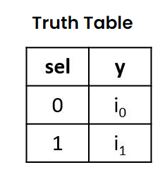
output [17:0] LEDR;

four\_bit\_adder DUT\_FPGA(SW[8],SW[7:4],SW[3:0],LEDR[4:1],LEDR[0]);

endmodule

**EXPERIMENT 7**

**AIM: WRITE VERILOG HDL CODES TO SIMULATE AND IMPLEMENT 2:1 MULTIPLEXER CIRCUIT:**

module mux21\_structural(i0,i1,sel,y);

input i0,i1,sel;

output y;

wire net1,net2,net3;

not g1(net1,sel);

and g2(net2,i1,sel);

and g3(net3,i0,net1);

or g4(y,net3,net2);

endmodule

module mux21\_data\_flow(i0,i1,sel,y);

input i0,i1,sel;

output y;

assign y =(i0&(~sel))|(i1&sel);

endmodule

module mux21\_Behavioural (i0,i1,sel,y);

input i0,i1,sel;

output y;

reg y;

always@(\*)

begin

if(sel==0) y=i0;

if(sel==1)y=i1;

end

endmodule

module Multiplexer\_2\_1(SW,LEDR,LEDG);

input [17:0] SW;

output [17:0] LEDR;

output [7:0] LEDG;

assign LEDR=SW;

mux21\_structural DUT1(.i0(SW[8]), .i1(SW[7]),.s(SW[6]),.y(LEDG[2]));

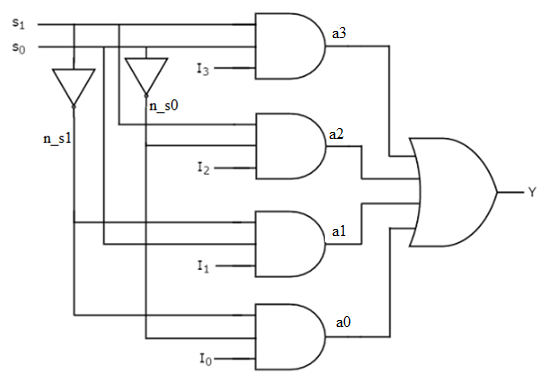
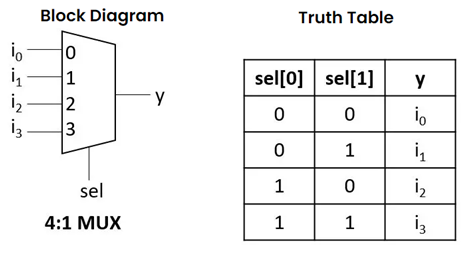
mux21\_data\_flow DUT2(.i0(SW[5]), .i1(SW[4]),.s(SW[3]),.y(LEDG[1]));

mux21\_Behavioural DUT3(.i0(SW[2]), .i1(SW[1]),.s(SW[0]),.y(LEDG[0]));

endmodule

**EXPERIMENT 8**

**AIM: WRITE VERILOG HDL CODES TO SIMULATE AND IMPLEMENT 4:1 MULTIPLEXER CIRCUIT:**

module mux41\_strutural (i0,i1,i2,i3,s0,s1,y);

input i0,i1,i2,i3,s0,s1;

output y;

wire n\_s0,n\_s1, a0,a1,a2,a3;

not g0(n\_s0,s0);

not g1(n\_s1,s1);

and g2(a0,i0,n\_s0,n\_s1);

and g3(a1,i1,n\_s1,s0);

and g4(a2,i2,s1,n\_s0);

and g5(a3,i3,s1,s0);

or g6(y,a0,a1,a2,a3);

endmodule

module mux41\_df (i0,i1,i2,i3,s0,s1,y);

input i0,i1,i2,i3,s0,s1;

output y;

assign y= i0&(~s1)&(~s0) | i1 &(~s1)&s0 | i2&s1&(~s0) | i3&s1&s0;

endmodule

module mux41beh\_v1(in,s,y );

output y ;

input [3:0] in ;

input [1:0] s ;

reg y;

always @ (in,s)

begin

if (s[0]==0&s[1]==0)

y <= in[0];

else if (s[0]==0&s[1]==1)

y <= in[1];

else if (s[0]==1&s[1]==0)

y <= in[2];

else

y <= in[3];

end

endmodule

module mux41beh\_v2(in,s,y );

output y ;

input [3:0]in ;

input [1:0]s ;

reg y;

always@(in,s)

begin

case ({s[1],s[0]})

2'b00: y <= in[0];

2'b01: y <= in[1];

2'b10: y <= in[2];

2'b11: y <= in[3];

endcase

end

endmodule

module Mux\_4\_1(SW,LEDR,LEDG);

input [17:0] SW;

output [17:0] LEDR;

output [7:0] LEDG;

assign LEDR=SW;

mux41\_strutural C1(.i0(SW[5]),.i1(SW[4]),.i2(SW[3]), .i3(SW[2]), .s1(SW[1]), .s0(SW[0]),.y (LEDG[3]));

mux41\_df (.i0(SW[5]), .i1(SW[4]), .i2(SW[3]), .i3(SW[2]), .s1(SW[1]), .s0(SW[0]),.y(LEDG[2]));

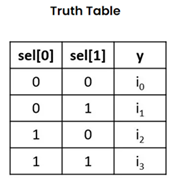
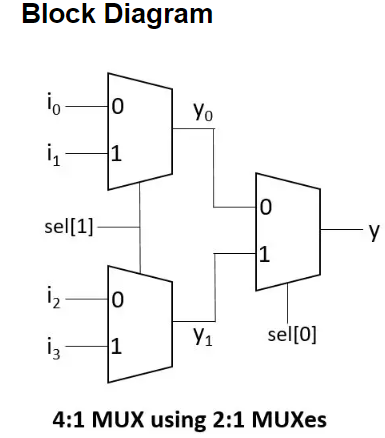
mux41beh\_v1 (.in(SW[5:2]), .s(SW[1:0]),.y (LEDG[1]));

mux41beh\_v2 (.in(SW[5:2]), .s(SW[1:0]), .y (LEDG[0]));

endmodule

**EXPERIMENT 9**

**AIM: WRITE VERILOG HDL CODES TO SIMULATE 4:1 MULTIPLEXER USING 2:1 MULTIPLEXER:**



module mux\_2\_1(

input sel,

input i0, i1,

output y);

assign y = sel ? i1 : i0;

endmodule

module mux\_4\_1\_using\_2\_to\_1(

input sel0, sel1,

input i0,i1,i2,i3,

output reg y);

wire y0, y1;

mux\_2\_1 m1(sel1, i2, i3, y1);

mux\_2\_1 m2(sel1, i0, i1, y0);

mux\_2\_1 m3(sel0, y0, y1, y);

endmodule

module tb;

reg sel0, sel1;

reg i0,i1,i2,i3;

wire y;

mux\_4\_1\_using\_2\_to\_1 DUT(sel0, sel1, i0, i1, i2, i3, y);

initial begin

$monitor("sel0=%b, sel1=%b -> i3 = %0b, i2 = %0b ,i1 = %0b, i0 = %0b -> y = %0b", sel0,sel1,i3,i2,i1,i0, y);

{i3,i2,i1,i0} = 4'h5;

repeat(6) begin

{sel0, sel1} = $random;

#5;

end

end

endmodule

module Mux\_4\_1\_test\_FPGA(SW,LEDR,LEDG);

input [17:0] SW;

output [17:0] LEDR;

output [7:0] LEDG;

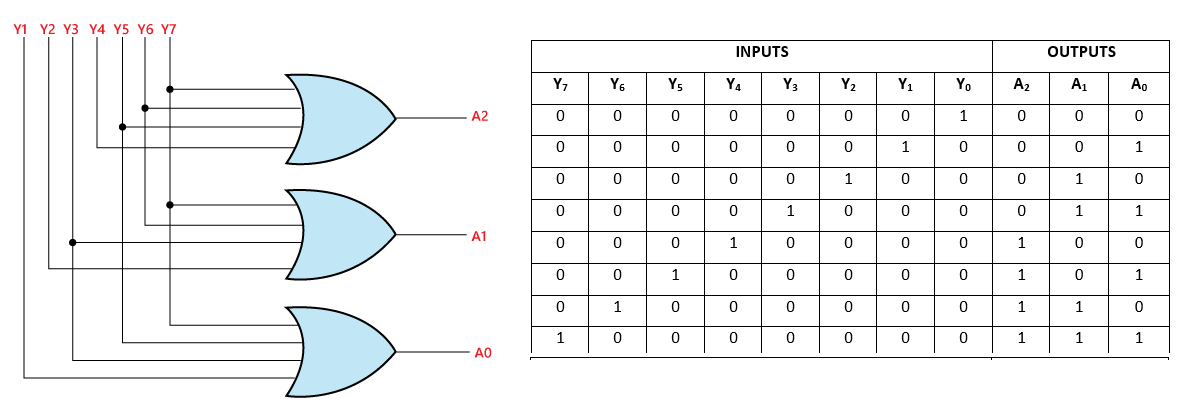
assign LEDR=SW;

mux\_4\_1\_using\_2\_to\_1 C1(.i0(SW[5]),.i1(SW[4]),.i2(SW[3]), .i3(SW[2]), .s1(SW[1]), .s0(SW[0]),.y (LEDG[3]));

endmodule

**EXPERIMENT 10**

**AIM: WRITE VERILOG HDL CODES TO SIMULATE AND IMPLEMENT 8 TO 3 ENCODER WITHOUT PRIORITY:**



module encoder\_stru (din, dout);

input [7:0] din;

output [2:0] dout;

reg [2:0] dout;

or g1(dout[0],din[1],din[3],din[5],din[7]);

or g2(dout[1],din[2],din[3],din[6],din[7]);

or g3(dout[2],din[4]4,din[5],din[6],din[7]);

endmodule

module encoder\_df(din, dout);

input [7:0] din;

output [2:0] dout;

reg [2:0] dout;

assign=dout[0],din[1],din[3],din[5],din[7]);

or g2(dout[1],din[2],din[3],din[6],din[7]);

or g3(dout[2],din[4]4,din[5],din[6],din[7]);

endmodule

module encoder (din, dout);

input [7:0] din;

output [2:0] dout;

reg [2:0] dout;

always @(din)

begin

if (din ==8'b00000001) dout=3'b000;

else if (din==8'b00000010) dout=3'b001;

else if (din==8'b00000100) dout=3'b010;

else if (din==8'b00001000) dout=3'b011;

else if (din==8'b00010000) dout=3'b100;

else if (din ==8'b00100000) dout=3'b101;

else if (din==8'b01000000) dout=3'b110;

else if (din==8'b10000000) dout=3'b111;

else dout=3'bX;

end

endmodule

module encoder\_without\_prio(a,en,y);

input [7:0] a;

input en;

output reg [2:0] y;

always@(a or en)

begin

if(!en)

y<=1'b0;

else

case(a)

8'b00000001:y<=3'b000;

8'b00000010:y<=3'b001;

8'b00000100:y<=3'b010;

8'b00001000:y<=3'b011;

8'b00010000:y<=3'b100;

8'b00100000:y<=3'b101;

8'b01000000:y<=3'b110;

8'b10000000:y<=3'b111;

endcase

end

endmodule