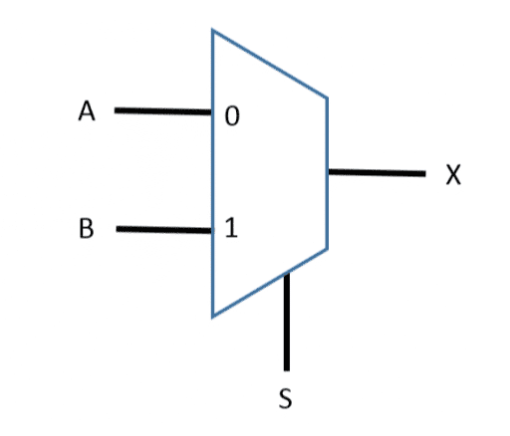
Altay İlker Yiğitel

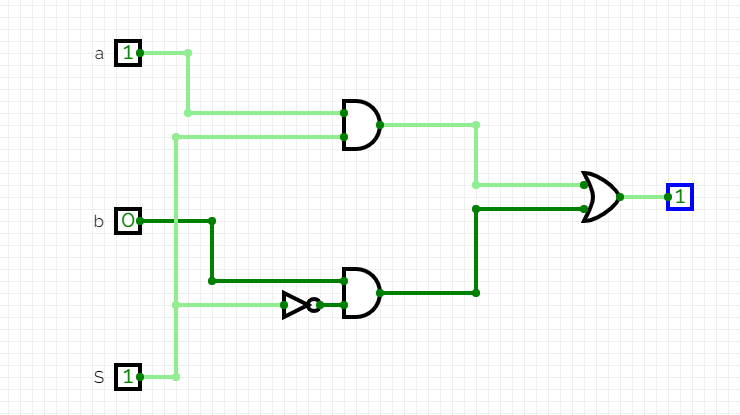
22203024

Section:01

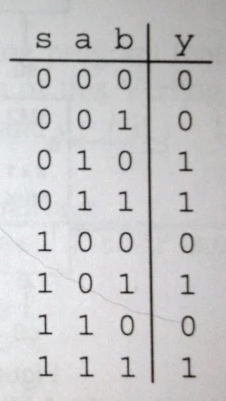
Mulitplexer Graphical Symbol:



Mulitplexer Logic Diagram :



Logic Diagram:



2-1 multiplexer behavioral module:

module mux-2-1(a,b,S0,y);

output y;

input a,b,S0;

assign y= (~S0 & a)|(S0&b);

endmodule

Testbench for 2-1 multiplexer:

module mux2\_1\_tb;

reg a, b, S0;

wire y;

mux\_2\_1 uut(a, b, S0, y);

initial begin

a = 0; b = 0; S0 = 0;

#100;

a = 0; b = 0; S0 = 1;

#100;

a = 0; b = 1; S0 = 0;

#100;

a = 0; b = 1; S0 = 1;

#100;

a = 1; b = 0; S0 = 0;

#100;

a = 1; b = 0; S0 = 1;

#100;

a = 1; b = 1; S0 = 0;

#100;

a = 1; b = 1; S0 = 1;

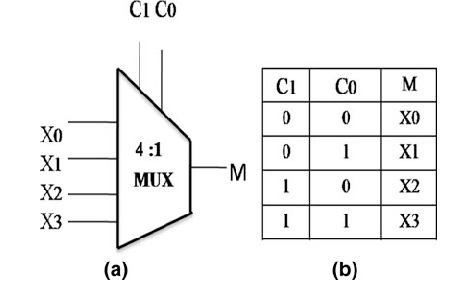
#100;

$finish();

end

endmodule

Graphical symbol and truth table of a 4-to-1 multiplexer:



Behavioral SystemVerilog module for 4-to-1 multiplexer by using three 2-to-1 multiplexers:

module mux4\_1(

input a, b, c, d, S0, S1,

output y2,

wire y0,y1

);

assign y0 = (~S0 & a) | (S0 & b);

assign y1 = (~S1 & c) | (S1 & d);

assign y2 = ((S0 ^ S1) & y0) | ((~(S0 ^ S1)) & y1);

endmodule

Testbench:

module mux\_4\_1\_tb();

reg a, b, c, d, S0, S1;

wire y0, y1, y2;

mux\_4\_1 uut (

.a(a),

.b(b),

.c(c),

.d(d),

.S0(S0),

.S1(S1),

.y0(y0),

.y1(y1),

.y2(y2);

initial begin

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

a = i[5];

b = i[4];

c = i[3];

d = i[2];

S1 = i[1];

S0 = i[0];

#100;

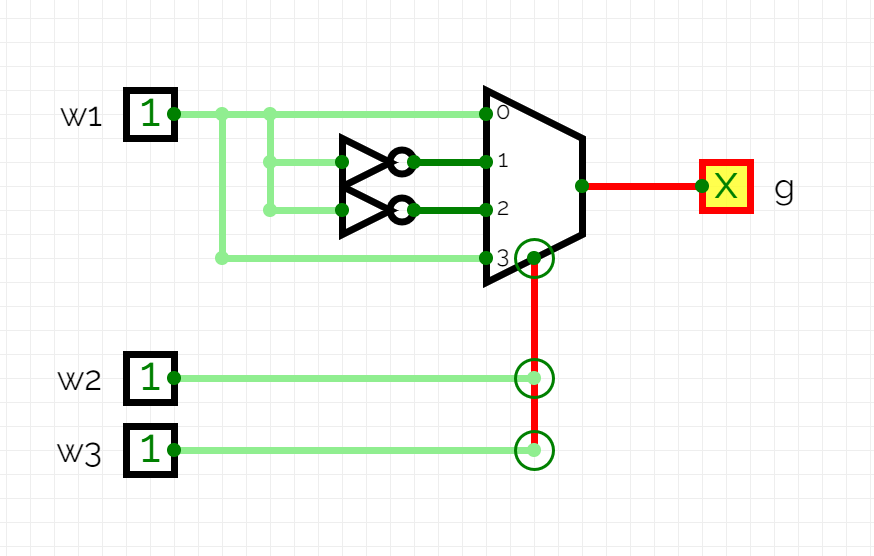
end

$finish;

Endmodule

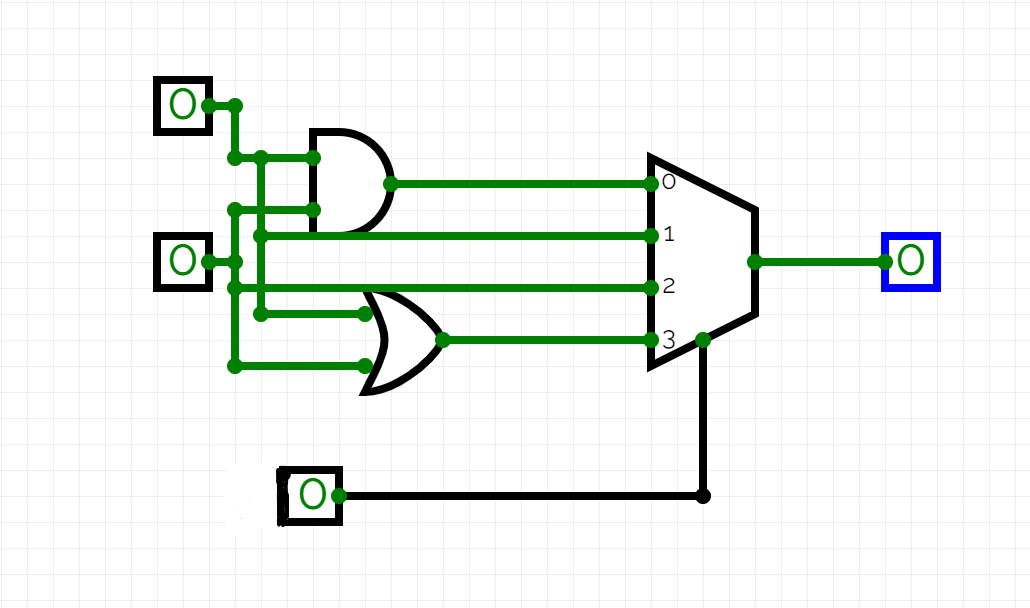
Three-input logic function g part a with single 4-to-1 multiplexer:

When w1 is true and others are flase 0 condition gives true output, 1 and 2 condition gives true only when w1 is false and one of w2 and w3 are true when all are true 3 condition is true so output i true as well.



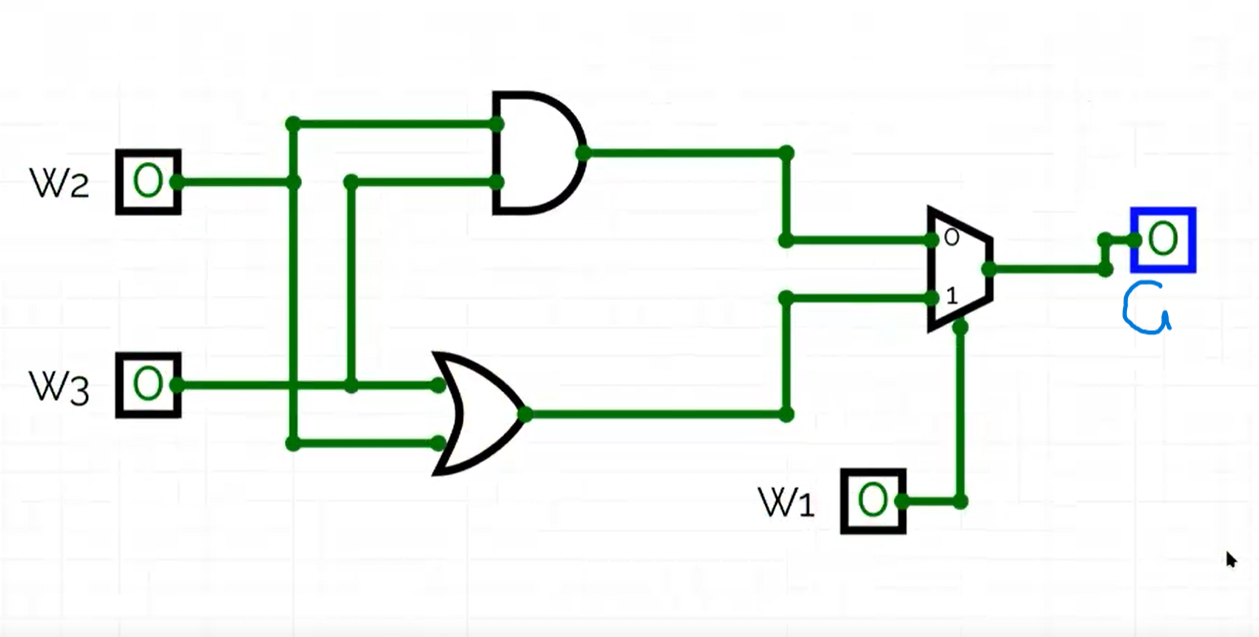
Three-input logic function g part b with single 4-to-1 multiplexer:

The function gives true when 2 or 3 of the switches are on so when controller is 0 both of the switches have to be 1 and when controller is 1 it doesnt matter weather one or two of them are true. 1 and 2 numbered inputs of multiplexer and second controller are unnecessary.



Three-input logic function g part b with single 2-to-1 multiplexer:

And gate in the diagram limits w1 when it is 0 and only when w2 and w3 are both true it is true and for the or gate whenever w2 or w3 becomes true and w1 is true it gives true as an answer.



Behavioral SystemVerilog module for the function implemented with a 2-to-1 multiplexer:

module oneB(

output y,

input w1, w2, w3

);

assign y = (~w1 & (w2&w3)) | (w1 & (w2^w3));

endmodule

Testbench:

module oneB\_tb;

reg w1, w2, w3;

wire y;

oneB uut ( .w1(w1),

.w2(w2),

.w3(w3),

.y(y) );

w1 = 0; w2 = 0; w3 = 0;

#10;

w1 = 0; w2 = 0; w3 = 1;

#10;

w1 = 0; w2 = 1; w3 = 0;

#10;

w1 = 0; w2 = 1; w3 = 1;

#10;

w1 = 1; w2 = 0; w3 = 0;

#10;

w1 = 1; w2 = 0; w3 = 1;

#10;

w1 = 1; w2 = 1; w3 = 0;

#10;

w1 = 1; w2 = 1; w3 = 1;

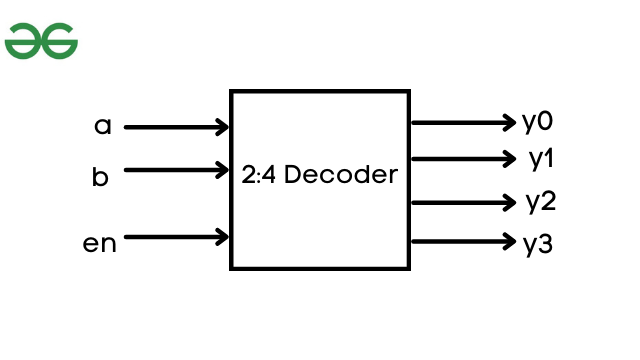
#10;

$finish;

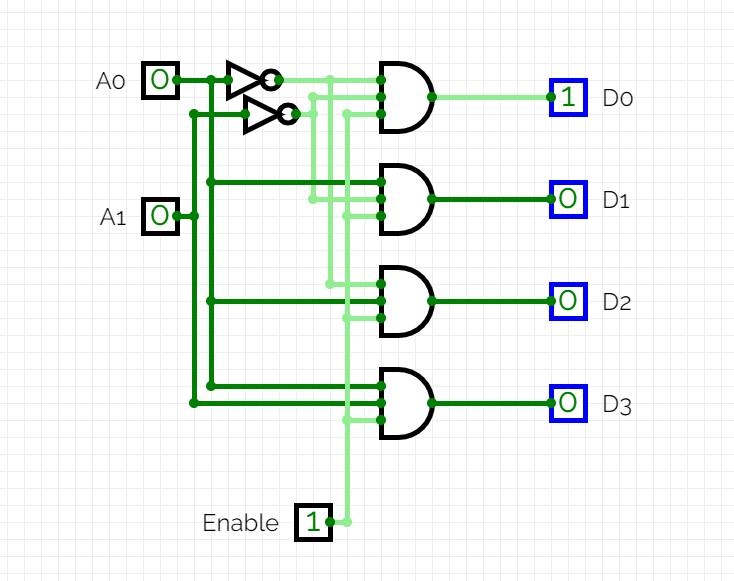
end

endmodule

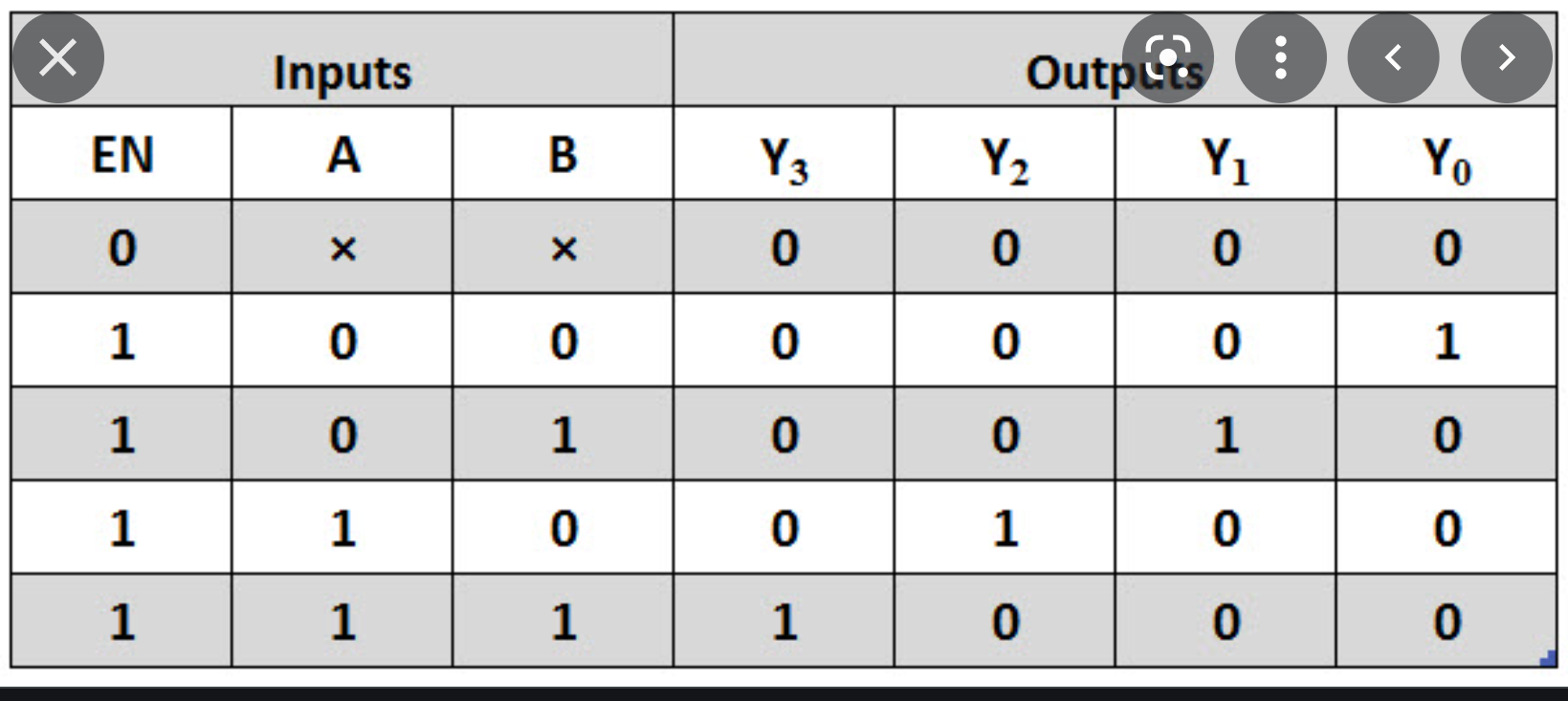
Graphical symbol of a 2-to-4 decoder:



Logic diagram of a 2-to-4 decoder:



Truth table of a 2-to-4 decoder:



Behavioral SystemVerilog module for 2-to-4 decoder:

module decoder4\_2(a,b,en,y0,y1,y2,y3);

output y0,y1,y2,y3;

input a, b, en;

assign y0 = (en&(~a&~b));

assign y1 = (en&(a&~b));

assign y2 = (en&(~a&b));

assign y3 = (en&(a&b));

endmodule

Testbench:

module decoder4\_2\_tb;

reg a, b, en;

wire y0, y1, y2, y3;

decoder4\_2 uut ( .a(a),

.b(b),

.en(en),

.y0(y0),

.y1(y1),

.y2(y2),

.y3(y3) );

initial begin

a = 0; b = 0; en = 0;

#100;

a = 0; b = 0; en = 1;

#100;

a = 0; b = 1; en = 0;

#100;

a = 0; b = 1; en = 1;

#100;

a = 1; b = 0; en = 0;

#100;

a = 1; b = 0; en = 1;

#100;

a = 1; b = 1; en = 0;

#100;

a = 1; b = 1; en = 1;

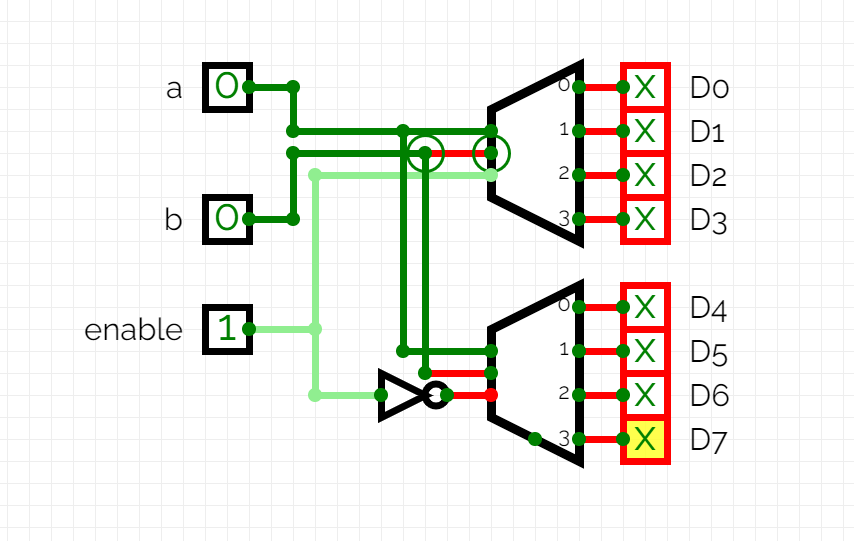
#100;

$finish();

end

endmodule

Logic diagram of 3-to-8 decoder using 2-to-4 decoders and minimal amount of gates:



Structural SystemVerilog module for 3-to-8 decoder using 2-to-4 decoders:

module decoder8\_3();

output y0,y1,y2,y3,y4,y5,y6,y7;

input a,b,en;

assign y0 = (en&(~a&~b));

assign y1 = (en&(a&~b));

assign y2 = (en&(~a&b));

assign y3 = (en&(a&b));

assign y4 = (~en&(~a&~b));

assign y5 = (~en&(a&~b));

assign y6 = (~en&(~a&b));

assign y7 = (~en&(a&b));

endmodule

Testbench:

module decoder8\_3\_tb;

reg a, b, en;

wire y0, y1, y2, y3,y4,y5,y6,y7;

decoder8\_3 uut ( .a(a),

.b(b),

.en(en),

.y0(y0),

.y1(y1),

.y2(y2),

.y3(y3)

.y4(y4),

.y5(y5),

.y6(y6),

.y7(y7),);

initial begin

a = 0; b = 0; en = 0;

#100;

a = 0; b = 0; en = 1;

#100;

a = 0; b = 1; en = 0;

#100;

a = 0; b = 1; en = 1;

#100;

a = 1; b = 0; en = 0;

#100;

a = 1; b = 0; en = 1;

#100;

a = 1; b = 1; en = 0;

#100;

a = 1; b = 1; en = 1;

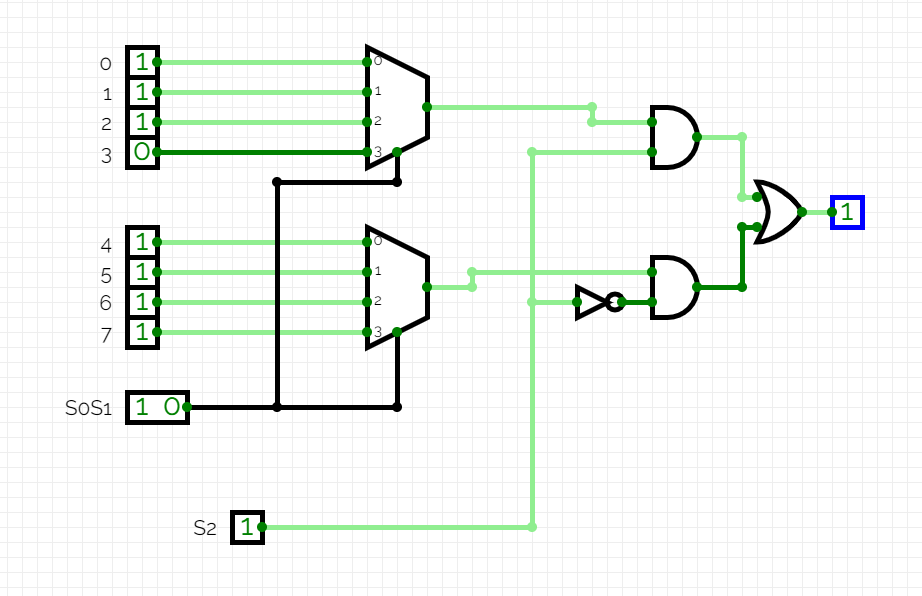
#100;

$finish();

end

Endmodule

Schematic (block diagram) of 8-to-1 MUX by using two 4-to-1 MUX modules:



SystemVerilog module of 8-to-1 MUX :

module mux8\_1(

input a, b, c, d,e,f,g,h ,S0, S1,S2,

output y6

);

//mux4\_1

assign y0 = (~S0 & a) | (S0 & b);

assign y1 = (~S1 & c) | (S1 & d);

assign y2 = ((S0 ^ S1) & y0) | ((~(S0 ^ S1)) & y1);

//mux4\_1\_(2)

assign y3 = (~S0 & e) | (S0 & f);

assign y4 = (~S1 & g) | (S1 & h);

assign y5 = ((S0 ^ S1) & y3) | ((~(S0 ^ S1)) & y4);

assign y6 = (y2&S2)^(y5&~S2);

endmodule

Testbench:

module mux8\_1\_tb;

reg a, b, c, d, e, f, g, h;

reg S0, S1, S2;

wire y6;

module mux8\_1 dut ( .a(a), .b(b), .c(c), .d(d), .e(e), .f(f), .g(g), .h(h), .S0(S0), .S1(S1), .S2(S2), .y6(y6) );

initial begin

S0=0, S1=0, S2=0 {a, b, c, d, e, f, g, h} = 8'b00000000;

{S0, S1, S2} = 3'b000;

#10;

S0=0, S1=0, S2=1 {S0, S1, S2} = 3'b001;

#10;

S0=0, S1=1, S2=0 {S0, S1, S2} = 3'b010;

#10;

S0=0, S1=1, S2=1 {S0, S1, S2} = 3'b011;

#10;

S0=1, S1=0, S2=0 {S0, S1, S2} = 3'b100;

#10;

S0=1, S1=0, S2=1 {S0, S1, S2} = 3'b101;

#10;

S0=1, S1=1, S2=0 {S0, S1, S2} = 3'b110;

#10;

S0=1, S1=1, S2=1 {S0, S1, S2} = 3'b111;

#10;

$finish;

end

endmodule