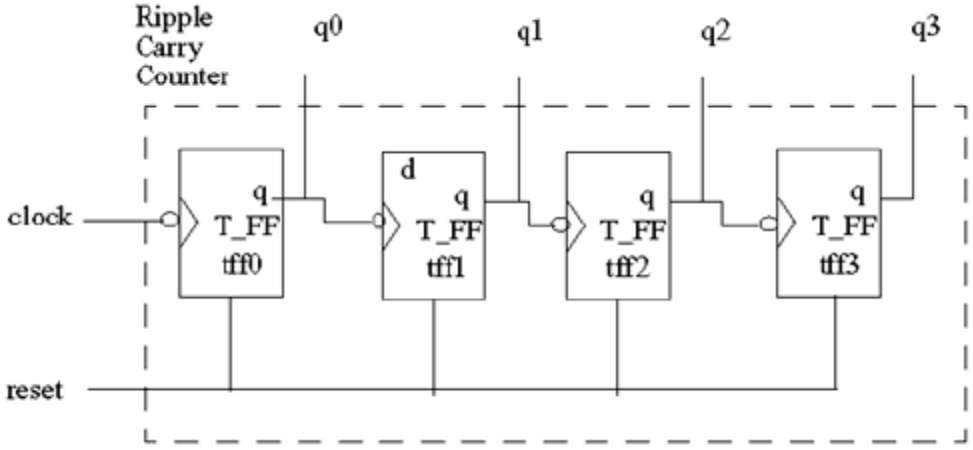
# VERILOG HDL

## ModelSim-Altera 6.6d (Quartus II 11.0sp1) Starter Edition


Author: Inaam Ahmed

July 2014

## Experiment#01: Ripple Carry Counter implementation in verilog

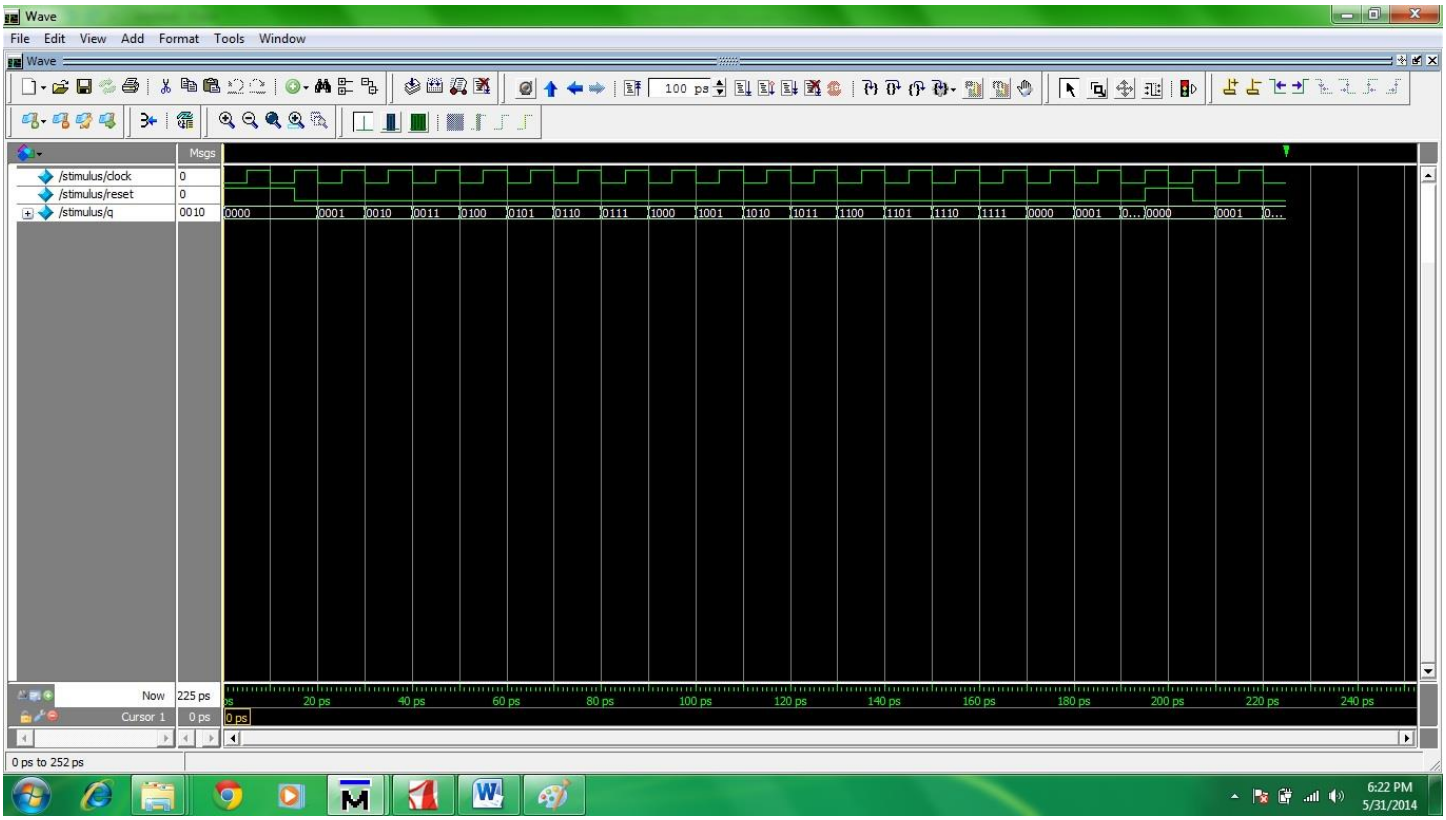



**Verilog Code**

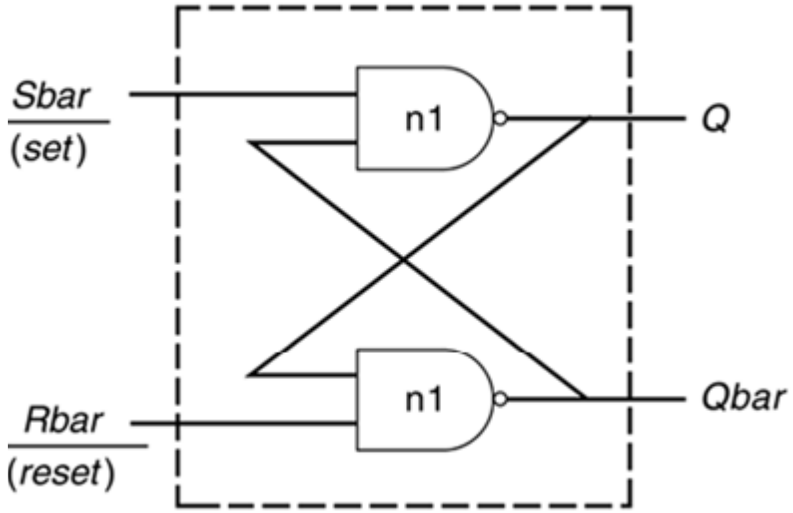
```
module ripple_carry_counter(q,clock,reset);
  output [3:0]q;
  input clock,reset;
  T_FF tff0(q[0],clock,reset);
  T_FF tff1(q[1],q[0],reset);
  T_FF tff2(q[2],q[1],reset);
  T_FF tff3(q[3],q[2],reset);
endmodule
module T_FF(q,clock,reset);
  output q;
  input clock,reset;
  wire d;
  not (d,q);
  D_FF dff(q,d,clock,reset);
endmodule
module D_FF(q,d,clock,reset);
  output q;
  input d,clock,reset;
  reg q;
```

```
 always @(posedge reset or negedge clock)
if (reset)
q <= 1'b0;
else
q <= d;
endmodule
module stimulus;
    reg clock;
    reg reset;
    wire [3:0] q;
  ripple_carry_counter R(q,clock,reset);
  initial
  clock=1'b0;
  always
  #5 clock=~clock;
  initial
   begin
   reset = 1'b1;
  #15 reset = 1'b0;
  #180 reset = 1'b1;
  #10 reset = 1'b0;
  #20 $finish;
   end
   initial
   $monitor($time,"Output q=%d",q);
 endmodule
```

## Simulation Results



## Experiment#02 SR-Latch implementation in Verilog

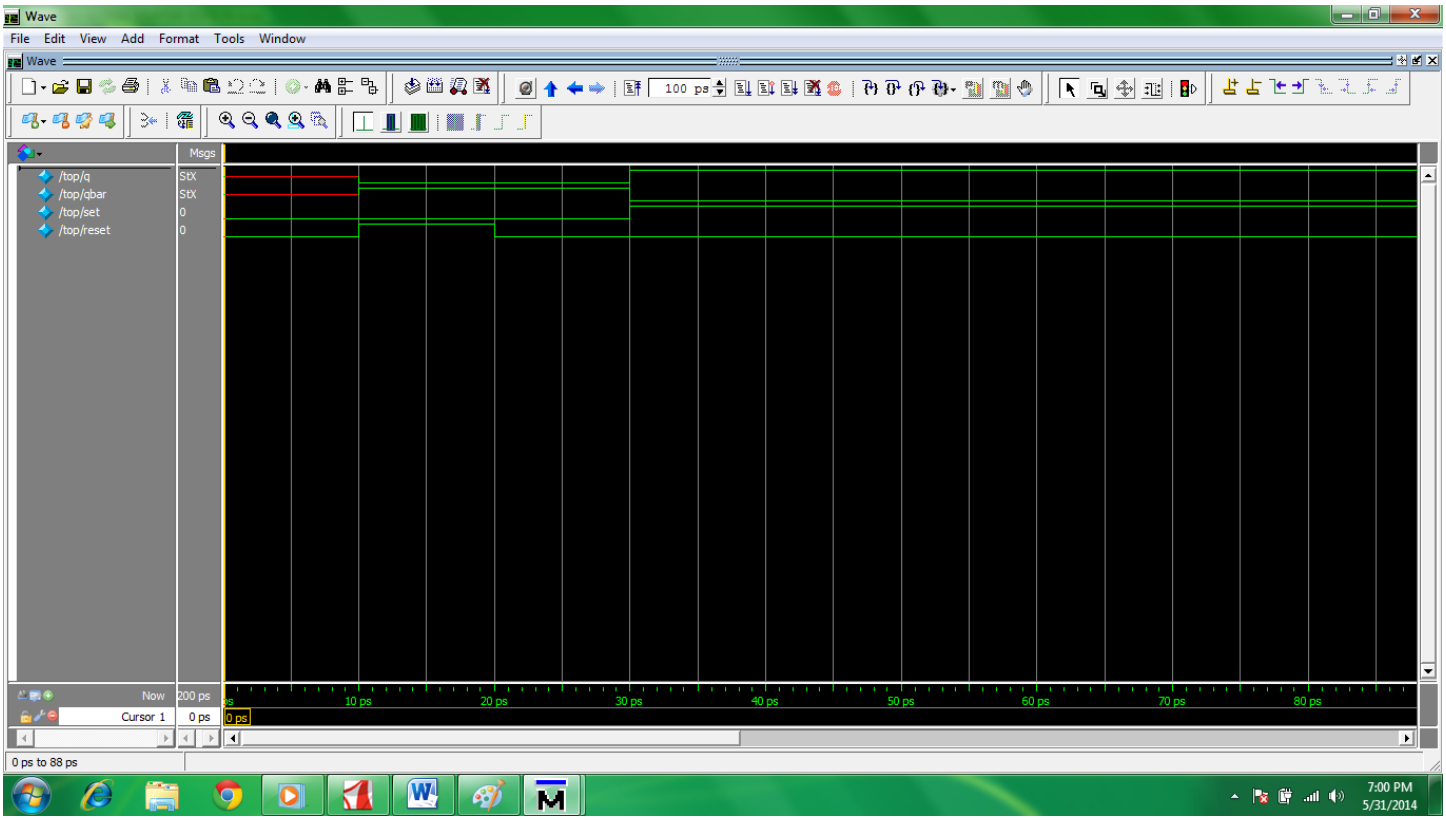



**Verilog Code**

```
module SR_latch(q,qbar,set,reset);
  output q,qbar;
  input set,reset;
  nand(q,qbar,set);
  nand(qbar,q,reset);
endmodule


module top;
  wire q,qbar;
  reg set,reset;
  SR_latch sr1(q,qbar,~set,~reset);
  initial
  begin
    $monitor($time,"set=%b,reset=%b,q=%b\n",set,reset,q);
    set=0;reset=0;
    #10 reset=1;
    #10 reset=0;
    #10 set=1;
  end
```

**endmodule**

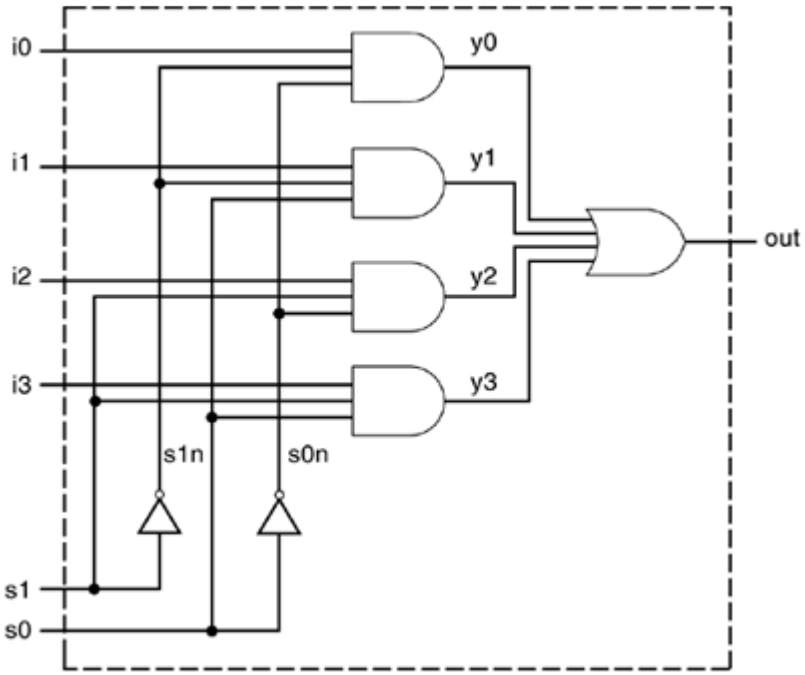
## Simulation Results

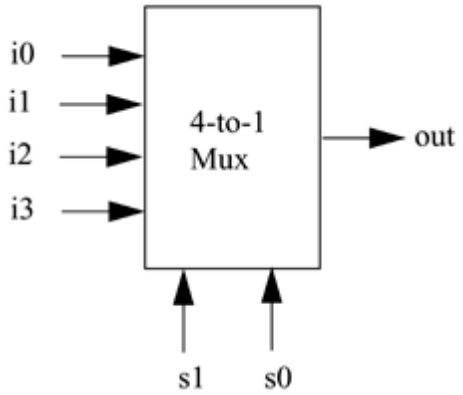


**Experiment#03 4_1 Mux using  Gate Level Abstraction**

- **Behavioral Modeling**

**Gate Level Abstraction**




| s1 | s0 | out |
|---|---|---|
| 0 | 0 | I0 |
| 0 | 1 | I1 |
| 1 | 0 | I2 |
| 1 | 1 | I3 |

**Verilog Code**

**module mux4_to_1 (out, i0, i1, i2, i3, s1, s0);**

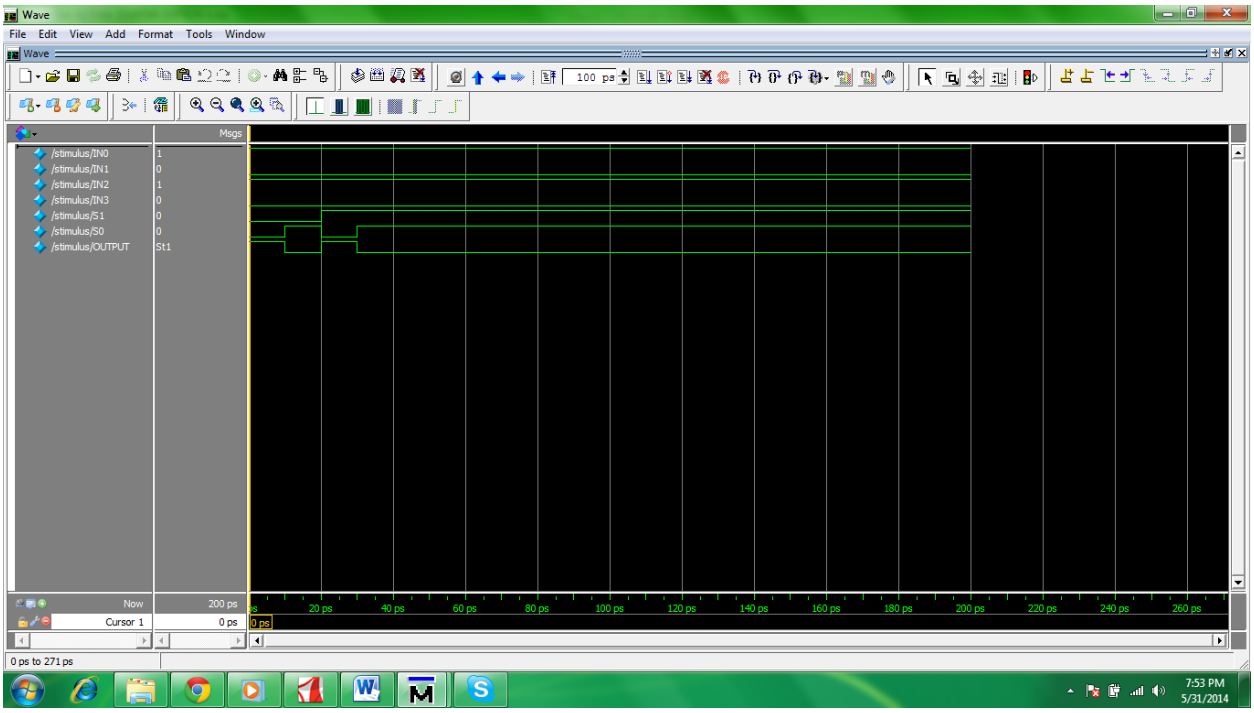
output out;

input i0, i1, i2, i3;

input s1, s0;

```
wire s1n, s0n;
wire y0, y1, y2, y3;
not (s1n, s1);
not (s0n, s0);
and (y0, i0, s1n, s0n);
and (y1, i1, s1n, s0);
and (y2, i2, s1, s0n);
and (y3, i3, s1, s0);
or (out, y0, y1, y2, y3);
endmodule
module stimulus;
reg IN0, IN1, IN2, IN3;
reg S1, S0;
wire OUTPUT;
mux4_to_1 mymux(OUTPUT, IN0, IN1, IN2, IN3, S1, S0);
initial
begin
IN0 = 1; IN1 = 0; IN2 = 1; IN3 = 0;
$monitor("IN0= %b, IN1= %b, IN2= %b, IN3= %b, S0= %b, S1= %b\n",IN0,IN1,IN2,IN3,S0,S1);
S1 = 0; S0 = 0;
#10 S1 = 0; S0 = 1;
#10 S1 = 1; S0 = 0;
#10 S1 = 1; S0 = 1;
end
endmodule
```

**Simulation Results**

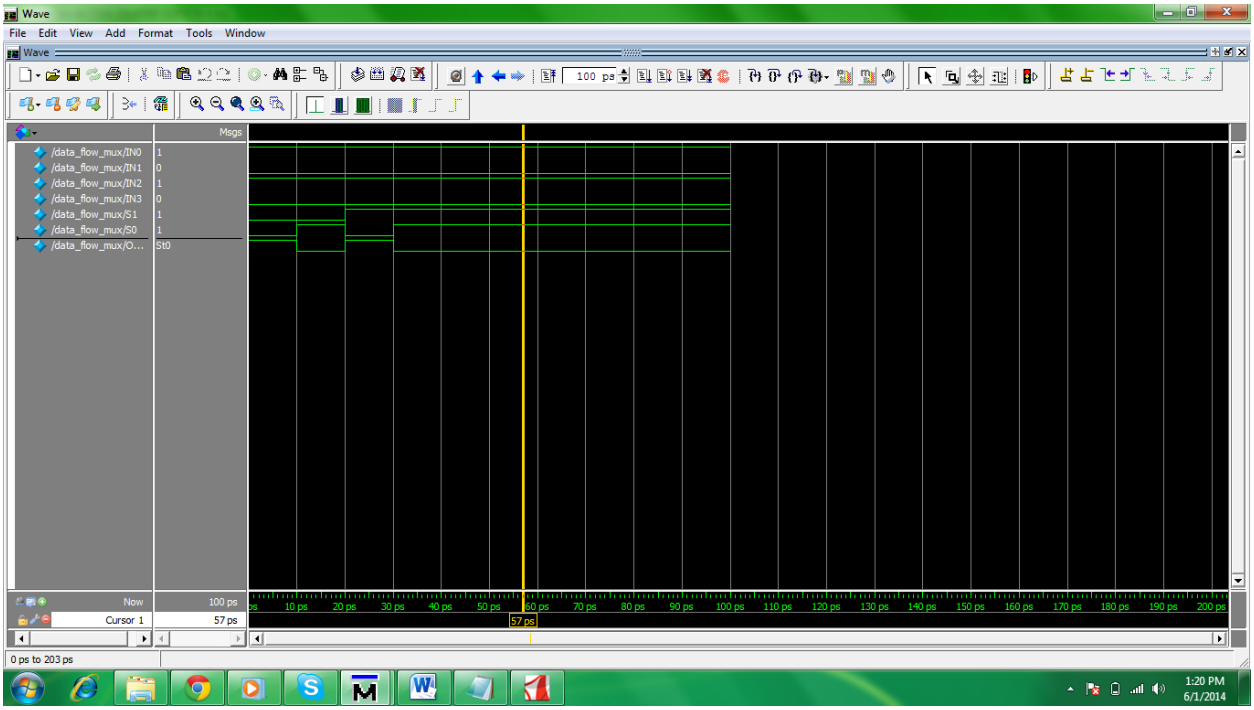

Wave
File Edit View Add Format Tools Window
Wave
Msgs
/stimulus/IN0
/stimulus/IN1
/stimulus/IN2
/stimulus/IN3
/stimulus/S1
/stimulus/S0
/stimulus/OUTPUT
1
0
1
0
0
0
St1
Now
200 ps
Cursor 1
0 ps
0 ps
20 ps
40 ps
60 ps
80 ps
100 ps
120 ps
140 ps
160 ps
180 ps
200 ps
220 ps
240 ps
260 ps
0 ps to 271 ps
7:53 PM
5/31/2014

# Experiment#04 4_1 Mux  Data Flow Level

**Using logic Equation**

```
module mux4_to_1 (out, i0, i1, i2, i3, s1, s0);
output out;
input i0, i1, i2, i3;
input s1, s0;
assign out = (~s1 & ~s0 & i0)|
(~s1 & s0 & i1) |
(s1 & ~s0 & i2) |
(s1 & s0 & i3) ;
endmodule
module data_flow_mux;
reg IN0, IN1, IN2, IN3;
reg S1, S0;
wire OUTPUT;
mux4_to_1 mymux(OUTPUT, IN0, IN1, IN2, IN3, S1, S0);
initial
begin
IN0 = 1; IN1 = 0; IN2 = 1; IN3 = 0;
$monitor("IN0= %b, IN1= %b, IN2= %b, IN3= %b, S0= %b, S1= %b\n",IN0,IN1,IN2,IN3,S0,S1);
S1 = 0; S0 = 0;
#10 S1 = 0; S0 = 1;
#10 S1 = 1; S0 = 0;
#10 S1 = 1; S0 = 1;
end
endmodule
```

**Simulation Results**

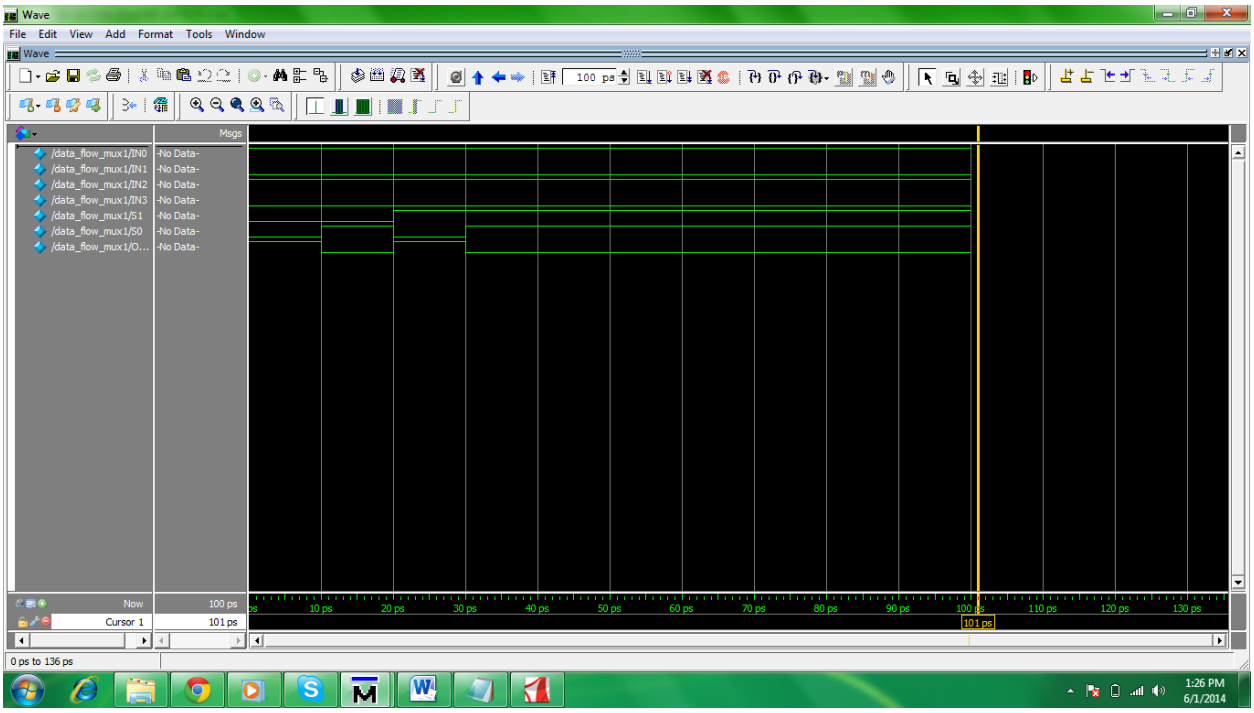



**Using Conditional Operator**

module multiplexer4_to_1 (out, i0, i1, i2, i3, s1, s0);

```
output out;
input i0, i1, i2, i3;
input s1, s0;
assign out = s1 ? ( s0 ? i3 : i2) : (s0 ? i1 : i0) ;
endmodule
module data_flow_mux1;
reg IN0, IN1, IN2, IN3;
reg S1, S0;
wire OUTPUT;
mux4_to_1 mymux(OUTPUT, IN0, IN1, IN2, IN3, S1, S0);
initial
begin
IN0 = 1; IN1 = 0; IN2 = 1; IN3 = 0;
$monitor("IN0= %b, IN1= %b, IN2= %b, IN3= %b, S0= %b, S1= %b\n",IN0,IN1,IN2,IN3,S0,S1);
S1 = 0; S0 = 0;
#10 S1 = 0; S0 = 1;
#10 S1 = 1; S0 = 0;
#10 S1 = 1; S0 = 1;
end
endmodule
```

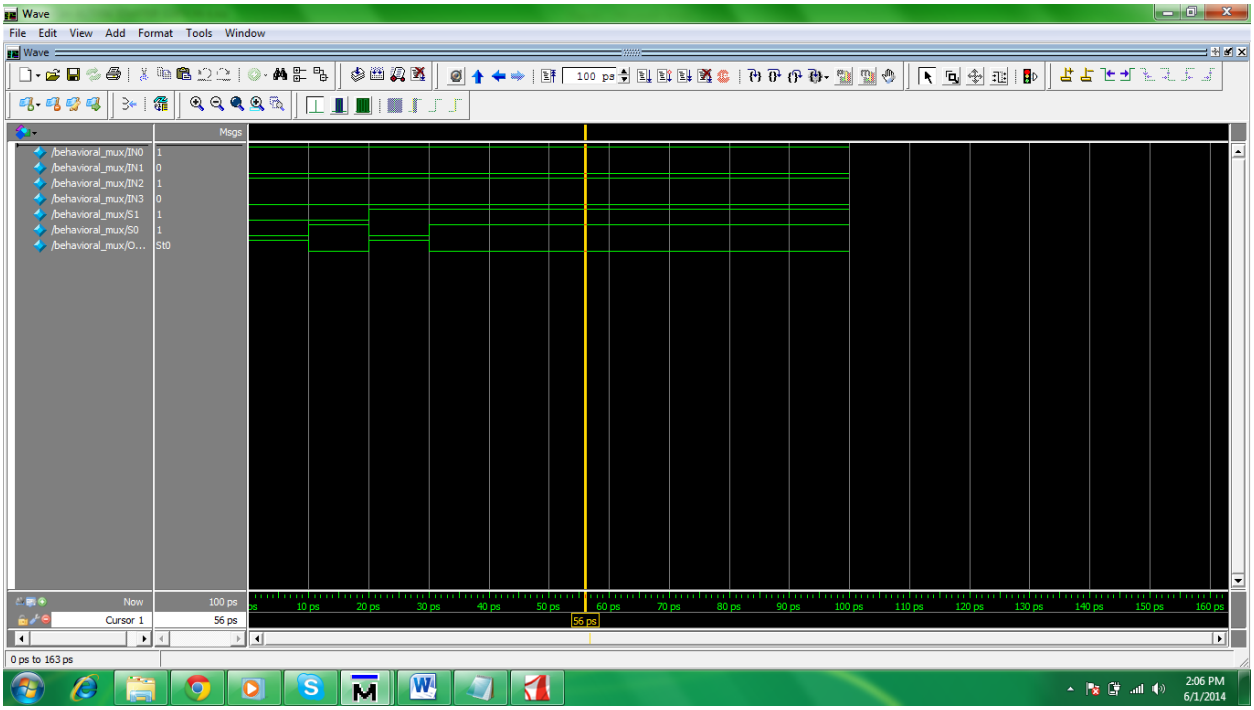
**Simulation Results**


Wave
File Edit View Add Format Tools Window
Wave
Msgs
/data_flow_mux1/IN0 -No Data-
/data_flow_mux1/IN1 -No Data-
/data_flow_mux1/IN2 -No Data-
/data_flow_mux1/IN3 -No Data-
/data_flow_mux1/S1 -No Data-
/data_flow_mux1/S0 -No Data-
/data_flow_mux1/O... -No Data-
Now 100 ps
Cursor 1 101 ps
10 ps 20 ps 30 ps 40 ps 50 ps 60 ps 70 ps 80 ps 90 ps 100 ps 110 ps 120 ps 130 ps
101 ps
0 ps to 136 ps
1:26 PM
6/1/2014

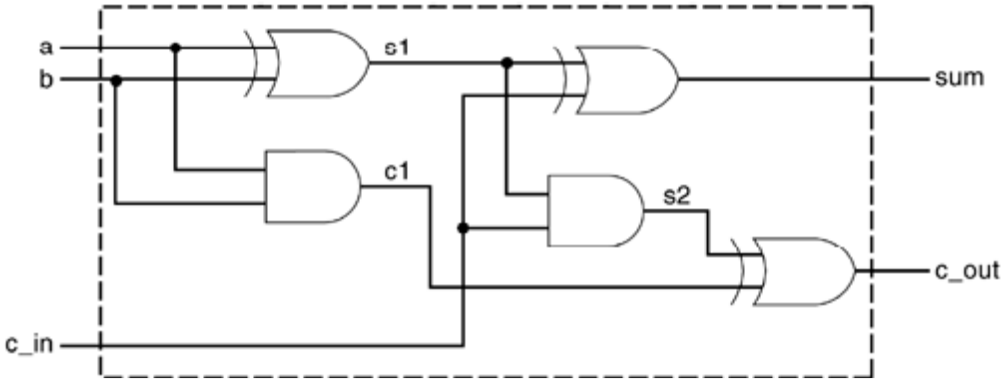
## Experiment#05 4_1 MUX Behavioral Modeling

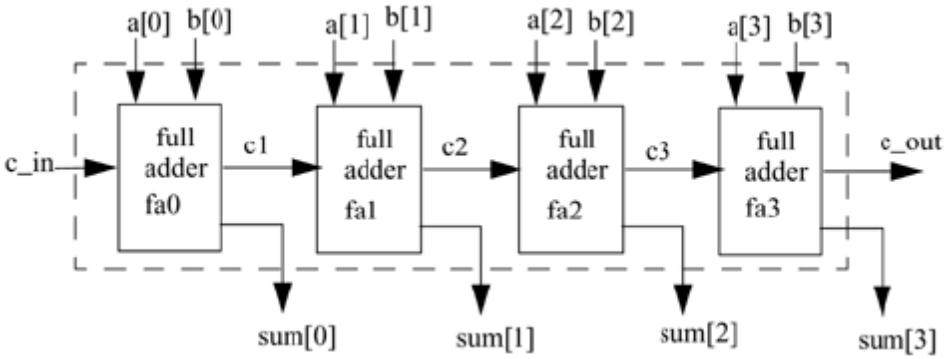
```
module mux4_to_1_behavioral(out, i0, i1, i2, i3, s1, s0);
output out;
input i0, i1, i2, i3;
input s1, s0;
reg out;
always @(s1 or s0 or i0 or i1 or i2 or i3)
begin
case ({s1, s0})
2'b00: out = i0;
2'b01: out = i1;
2'b10: out = i2;
2'b11: out = i3;
default: out = 1'bx;
endcase
end
endmodule
module behavioral_mux;
reg IN0, IN1, IN2, IN3;
reg S1, S0;
wire OUTPUT;
mux4_to_1_behavioral mymux(OUTPUT, IN0, IN1, IN2, IN3, S1, S0);
initial
begin
IN0 = 1; IN1 = 0; IN2 = 1; IN3 = 0;
$monitor("IN0= %b, IN1= %b, IN2= %b, IN3= %b, S0= %b, S1= %b\n",IN0,IN1,IN2,IN3,S0,S1);
S1 = 0; S0 = 0;
#10 S1 = 0; S0 = 1;
#10 S1 = 1; S0 = 0;
#10 S1 = 1; S0 = 1;
end
endmodule
```

## Simulation Results



# Experiment#06 4-bit Ripple Carry Full Adder




**Verilog Code**

**module fulladd(sum, c_out, a, b, c_in);**

output sum, c_out;

input a, b, c_in;

wire s1, c1, c2;

xor (s1, a, b);

and (c1, a, b);

xor (sum, s1, c_in);

and (c2, s1, c_in);

xor (c_out, c2, c1);

**endmodule**

**module fulladd4(sum, c_out, a, b, c_in);**

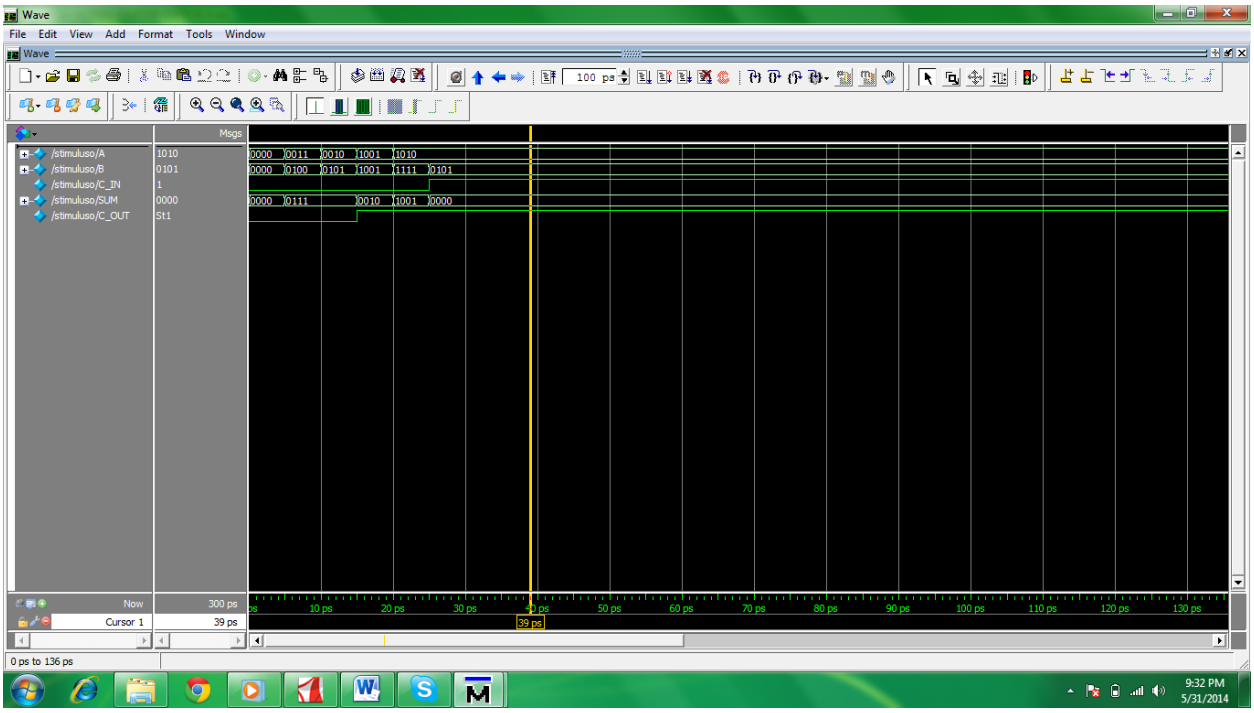
output [3:0] sum;

output c_out;

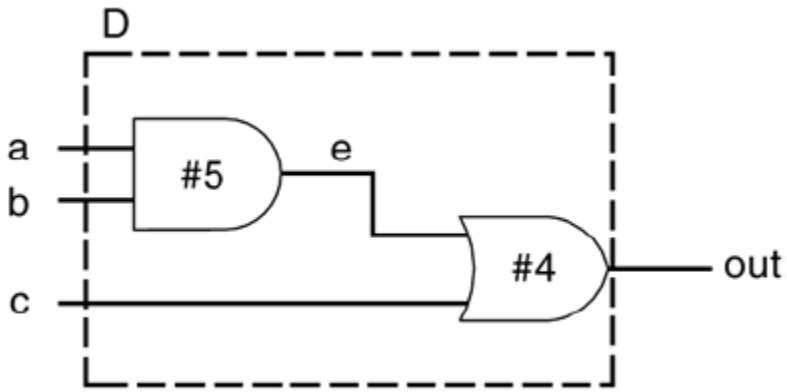
input[3:0] a, b;

```
input c_in;
wire c1, c2, c3;
fulladd fa0(sum[0], c1, a[0], b[0], c_in);
fulladd fa1(sum[1], c2, a[1], b[1], c1);
fulladd fa2(sum[2], c3, a[2], b[2], c2);
fulladd fa3(sum[3], c_out, a[3], b[3], c3);
endmodule
module stimuluso;
reg [3:0] A, B;
reg C_IN;
wire [3:0] SUM;
wire C_OUT;
fulladd4 FA1_4(SUM, C_OUT, A, B, C_IN);
initial
begin
$monitor($time," A= %b, B=%b, C_IN= %b, --- C_OUT= %b, SUM= %b\n",
A, B, C_IN, C_OUT, SUM);
end
initial
begin
A = 4'd0; B = 4'd0; C_IN = 1'b0;
#5 A = 4'd3; B = 4'd4;
#5 A = 4'd2; B = 4'd5;
#5 A = 4'd9; B = 4'd9;
#5 A = 4'd10; B = 4'd15;
#5 A = 4'd10; B = 4'd5; C_IN = 1'b1;
end
endmodule
```

**Simulation Results**


Wave
File Edit View Add Format Tools Window
Wave
Msgs
/stimuluso/A
/stimuluso/B
/stimuluso/C_IN
/stimuluso/SUM
/stimuluso/C_OUT
1010
0101
1
0000
St1
0000 0011 0010 1001 1010
0000 0100 0101 1001 1111 0101
0000 0111 0010 1001 0000
Now
300 ps
Cursor 1
39 ps
10 ps 20 ps 30 ps 40 ps 50 ps 60 ps 70 ps 80 ps 90 ps 100 ps 110 ps 120 ps 130 ps
39 ps
0 ps to 136 ps
9:32 PM
5/31/2014

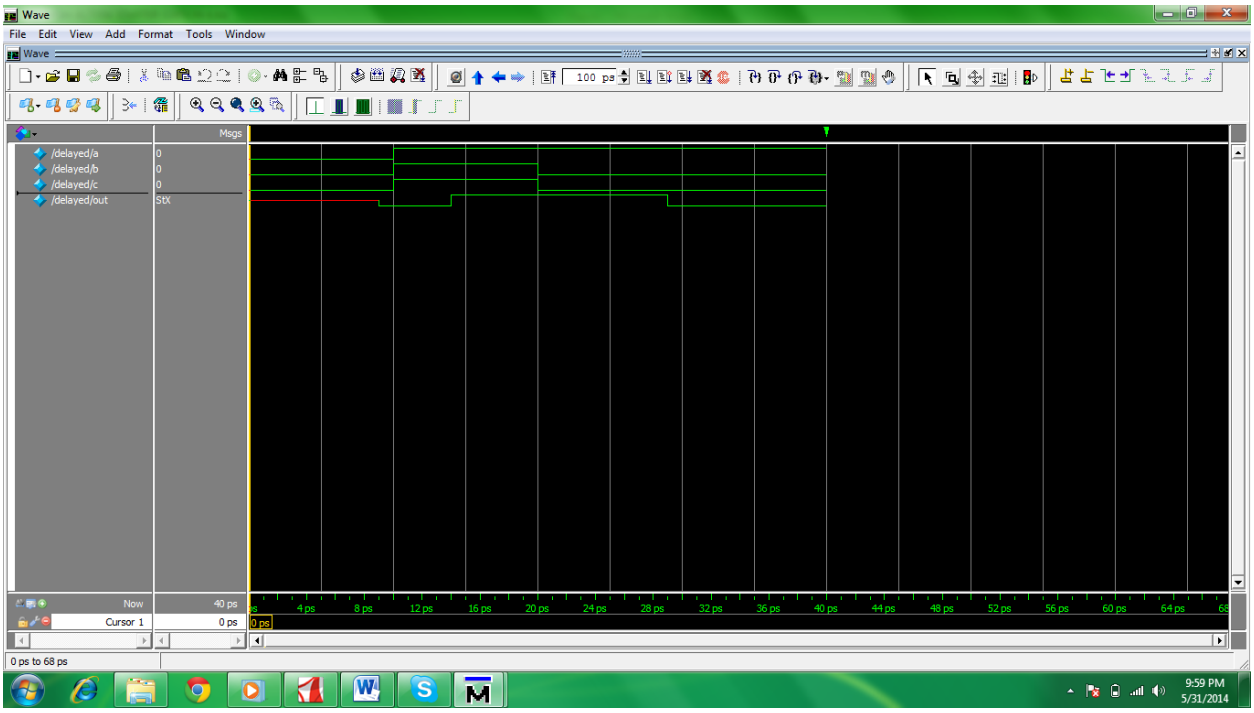
# Experiment#07 Gate Delays Implementation in Verilog




**Verilog code**

```
module delay (out,a,b,c);
  output out;
  input a,b,c;
  wire e;
  and #(5) delayed_and(e,a,b);
  or #(4) delayed_or(out,e,c);
endmodule


module delayed;
  reg a,b,c;
  wire out;

  delay my_gates(out,a,b,c);

  initial
  begin
    a=1'b0;b=1'b0;c=1'b0;
    #10 a=1'b1;b=1'b1;c=1'b1;
    #10 a=1'b1;b=1'b0;c=1'b0;
    #20 $finish;
  end
endmodule
```

**Simulation Results**


Wave
File Edit View Add Format Tools Window
Wave
Msgs
/delayed/a
/delayed/b
/delayed/c
/delayed/out
0
0
0
StX
Now
40 ps
Cursor 1
0 ps
0 ps
4 ps
8 ps
12 ps
16 ps
20 ps
24 ps
28 ps
32 ps
36 ps
40 ps
44 ps
48 ps
52 ps
56 ps
60 ps
64 ps
0 ps to 68 ps
9:59 PM
5/31/2014

# Experiment#08 Hierarchical Modeling Concepts

**How to create modules**

```
// interaction with Verilog

module mem;
endmodulem

odule sc;
endmodule

module xbar;
endmodule
```

**How to instantiate modules(Objects)**

```
module mem;

endmodule

module sc;

endmodule

module xbar;

endmodule

module is;

  mem mem1;

  sc sc1;

  xbar xbar1;

endmodule


module top;

  is is1;

endmodule
```

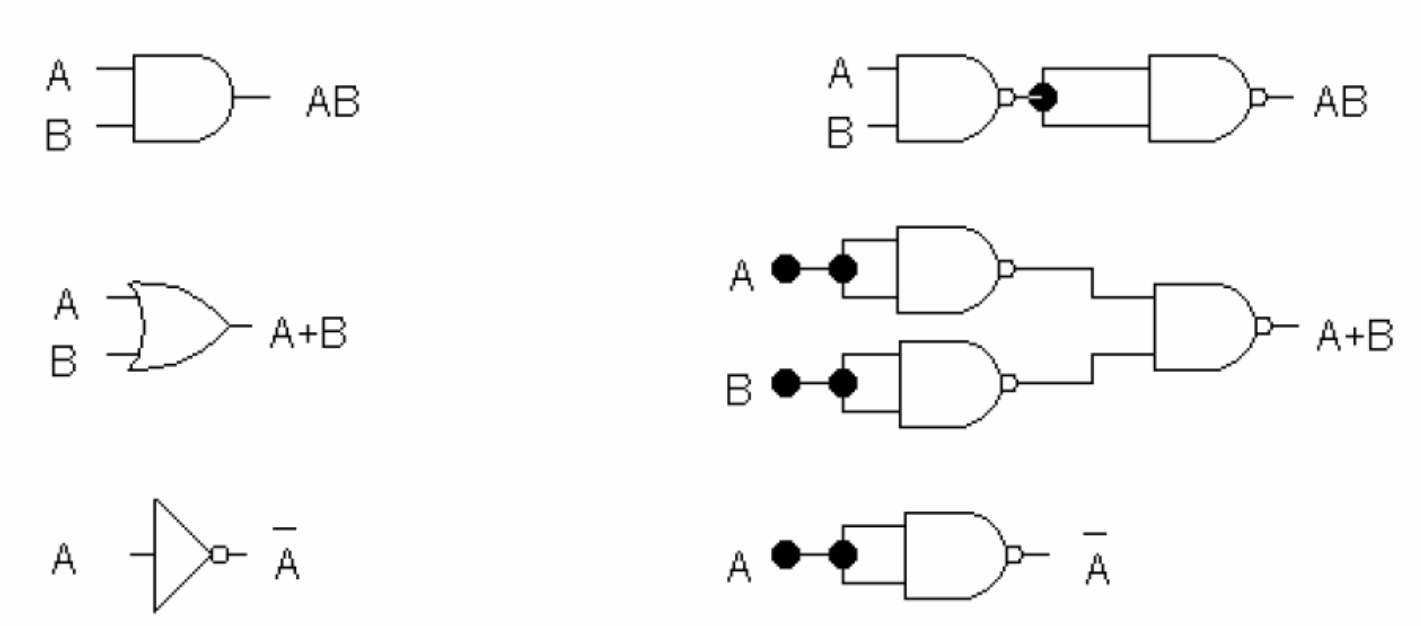
**Example of Ripple Adder**

```
module FA;

endmodule

module ripple_add;

  FA fa_0;

  FA fa_1;

  FA fa_2;
```

```
  FA fa_3;
endmodule
```

## Experiment#09 Implementation of

- **AND gate using NAND gate**
- **OR gate using NAND gate**
- **NOT gate using NAND**

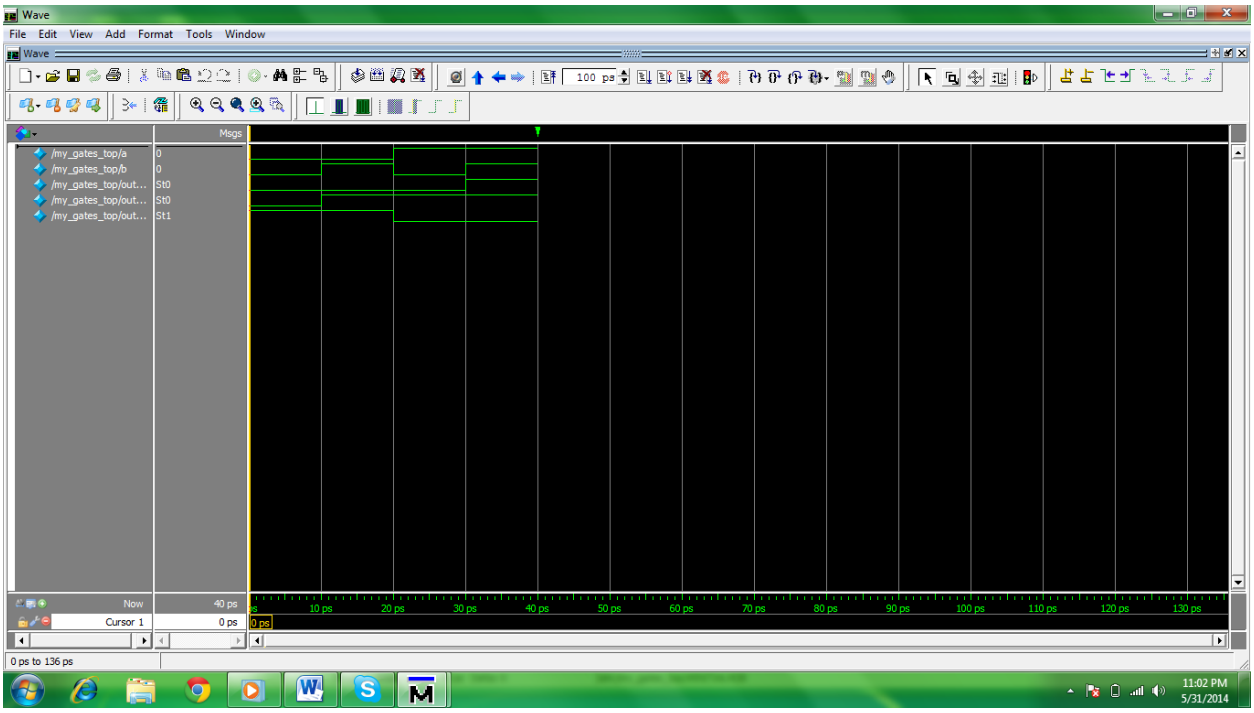



**Verilog code**

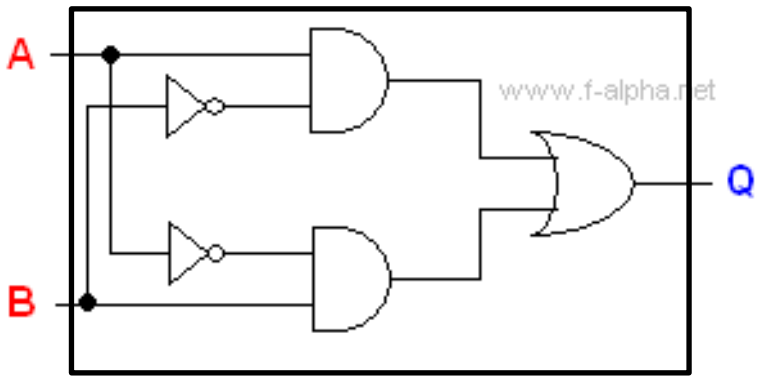
```
module my_gates(out_and,out_or,out_not,a,b);
  output out_and,out_or,out_not;
  input a,b;
//my_or
  wire x,y;
  nand(x,a,a);
  nand(y,b,b);
  nand(out_or,x,y);
//my_and
  wire z;
  nand(z,a,b);
  nand(out_and,z,z);
//my_not
  nand(out_not,a,a);
  endmodule
module my_gates_top;
```

```
  reg a,b;
  wire out_and,out_or,out_not;
  initial
  begin
    a=1'b0;b=1'b0;
#10 a=1'b0;b=1'b1;
#10 a=1'b1;b=1'b0;
#10 a=1'b1;b=1'b1;
#10 $finish;
  end

  my_gates inst(out_and,out_or,out_not,a,b);
  initial
  begin
    $monitor($time,"a=%b,b=%b---AND=%b---OR=%b---NOT=%b",a,b,out_and,out_or,out_not);
  end
endmodule
```

**Verilog Simulation Results**


Wave
File Edit View Add Format Tools Window
Wave
Msgs
/my_gates_top/a
/my_gates_top/b
/my_gates_top/out...
/my_gates_top/out...
/my_gates_top/out...
0
0
St0
St0
St1
Now
40 ps
Cursor 1
0 ps
10 ps
20 ps
30 ps
40 ps
50 ps
60 ps
70 ps
80 ps
90 ps
100 ps
110 ps
120 ps
130 ps
0 ps
0 ps to 136 ps
11:02 PM
5/31/2014

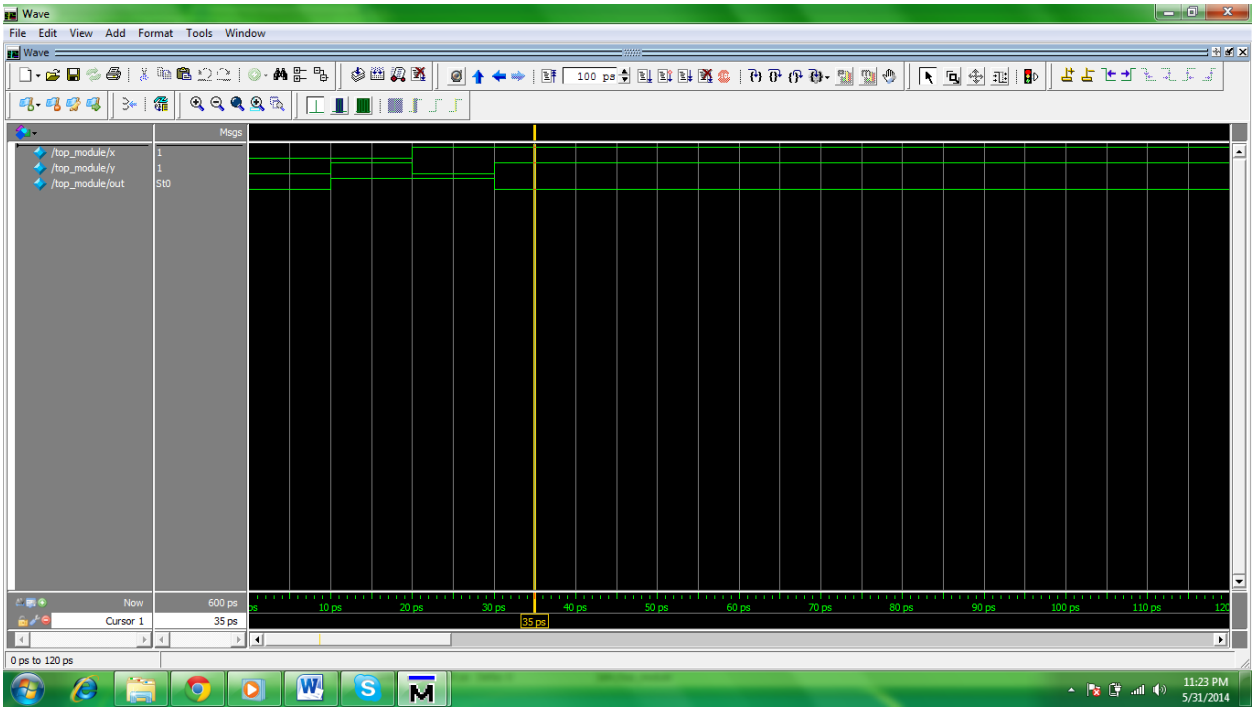
# Experiment#10 XOR gate implementation in Verilog



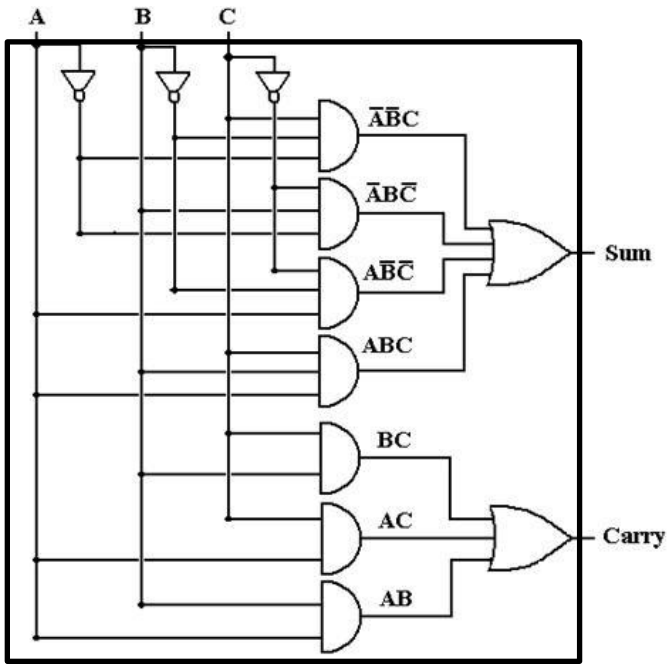

**Source Code**

```
module xor_gate(out,x,y);
  output out;
  input x,y;
  wire x1,y1,o1,o2;
  not(x1,x);
  not(y1,y);
  and(o1,x,y1);
  and(o2,y,x1);
  or my_xor(out,o1,o2);
endmodule
module top_module;
  reg x,y;
  wire out;
  xor_gate xor_block(out,x,y);
  initial
  begin
    $monitor($time,"x=%b,y=%b,OUT=%b\n",x,y,out);
  x=1'b0;y=1'b0;
#10 x=1'b0;y=1'b1;
#10 x=1'b1;y=1'b0;
#10 x=1'b1;y=1'b1;
  end
endmodule
```

**Verilog Simulation Results**


Wave
File Edit View Add Format Tools Window
Wave
Msgs
/top_module/x
/top_module/y
/top_module/out
1
1
St0
Now
600 ps
Cursor 1
35 ps
10 ps
20 ps
30 ps
40 ps
50 ps
60 ps
70 ps
80 ps
90 ps
100 ps
110 ps
35 ps
0 ps to 120 ps
11:23 PM
5/31/2014

# Experiment#11 Full Adder Implementation in Verilog

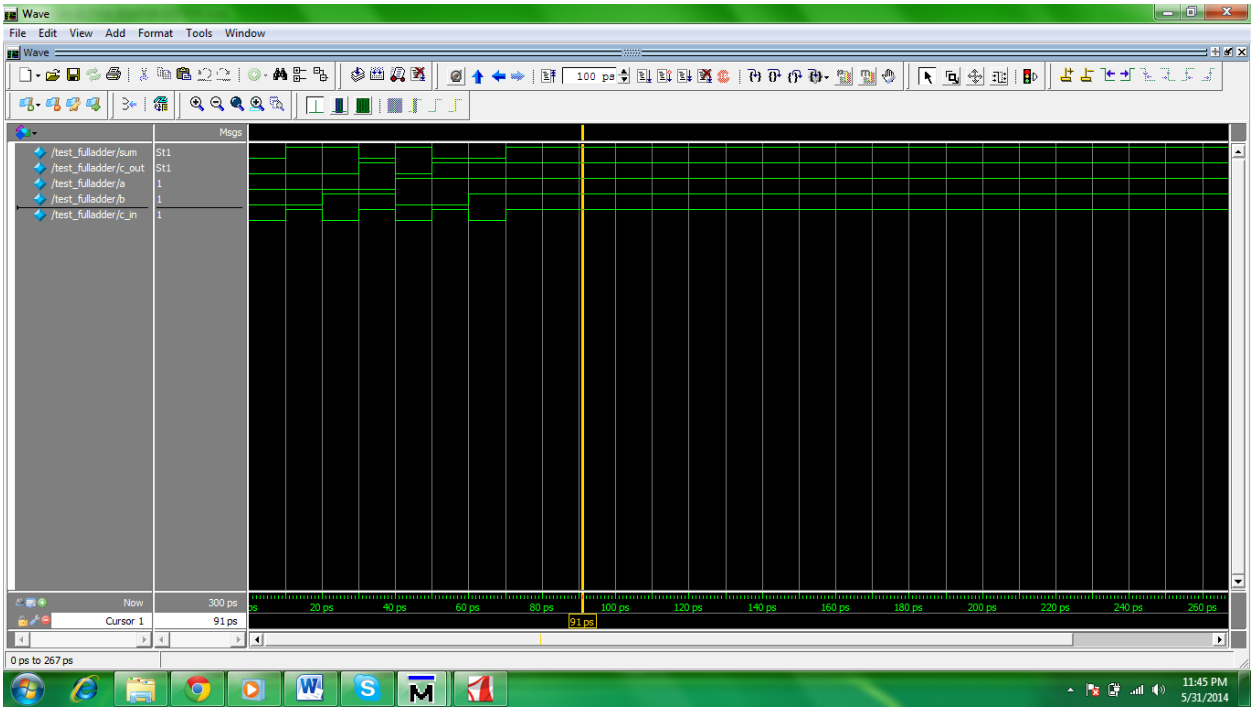



**Source Code**

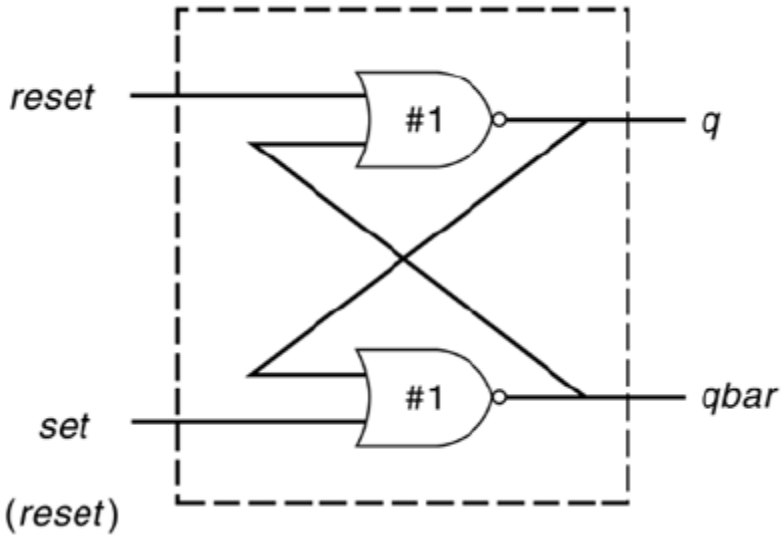
```
module F_A(sum,c_out,a,b,c_in);
  output sum,c_out;
  input a,b,c_in;
  wire a,b,c_in;
  wire a1,b1,c_in1,s1,s2,s3,s4;
  wire c1,c2,c3;
  not(a1,a);not(b1,b);not(c_in1,c_in);
  and(s1,a,b,c_in);
  and(s2,a1,b,c_in1);
  and(s3,a1,b1,c_in);
  and(s4,a,b1,c_in1);

  and(c1,a,b);
  and(c2,b,c_in);
  and(c3,a,c_in);

  or sum_1(sum,s1,s2,s3,s4);
  or carry_1(c_out,c1,c2,c3);
```

```
endmodule
module test_fulladder;
  wire sum,c_out;
  reg a,b,c_in;
 F_A full_adder(sum,c_out,a,b,c_in);
 initial
 begin
   $monitor($time,"A=%b,B=%b,C_IN=%b----SUM=%b,C_OUT=%b",a,b,c_in,sum,c_out);
   a=1'b0;b=1'b0;c_in=1'b0;
   #10 a=1'b0;b=1'b0;c_in=1'b1;
   #10 a=1'b0;b=1'b1;c_in=1'b0;
   #10 a=1'b0;b=1'b1;c_in=1'b1;
   #10 a=1'b1;b=1'b0;c_in=1'b0;
   #10 a=1'b1;b=1'b0;c_in=1'b1;
   #10 a=1'b1;b=1'b1;c_in=1'b0;
   #10 a=1'b1;b=1'b1;c_in=1'b1;
 end
 endmodule
```

**Verilog Simulation Results**



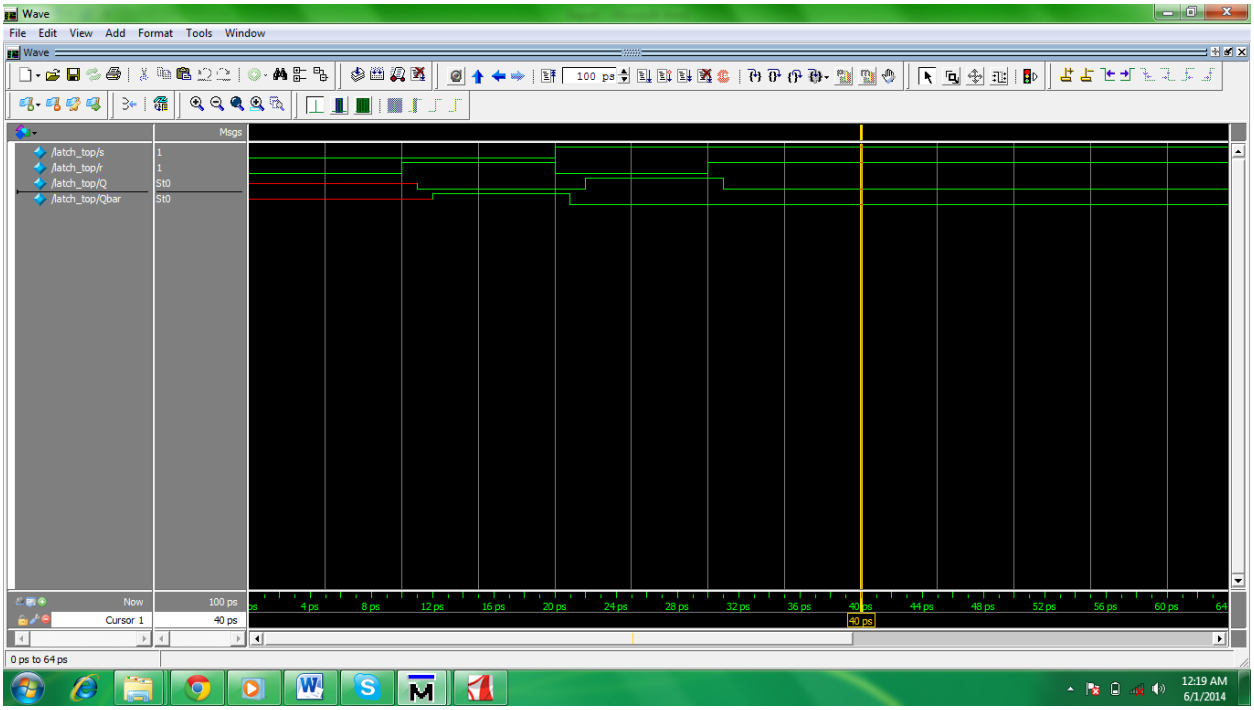
## Experiment#12 RS_Latch with delay(1).



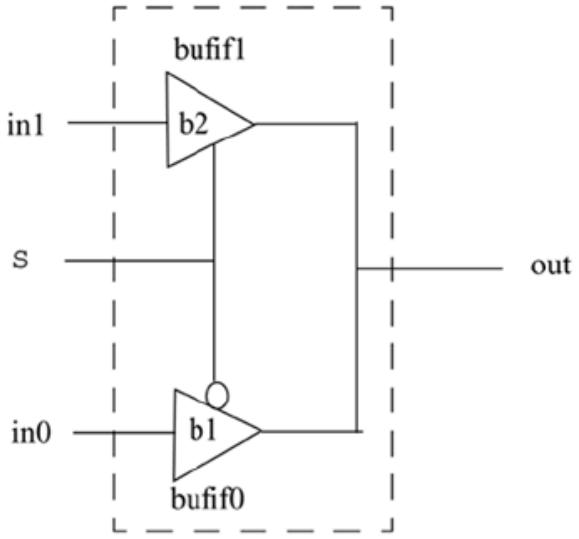

**Verilog Source Code**

```
module SR_latch(q,qbar,set,reset);
  input set,reset;
  output q,qbar;
  nor #(1)(q,reset,qbar);
  nor #(1)(qbar,set,q);
endmodule
module latch_top;
  reg s,r;
  wire Q,Qbar;
  SR_latch my_latch(.q(Q),.qbar(Qbar),.set(s),.reset(r));
  initial
  begin
    s=1'b0;r=1'b0;
    #10 s=1'b0;r=1'b1;
    #10 s=1'b1;r=1'b0;
    #10 s=1'b1;r=1'b1;
  end
endmodule
```

## Verilog Simulation Results



## Experiment#13 2-to-1 mux using multiplexer




The delay specification for gates b1 and b2 are as follows:

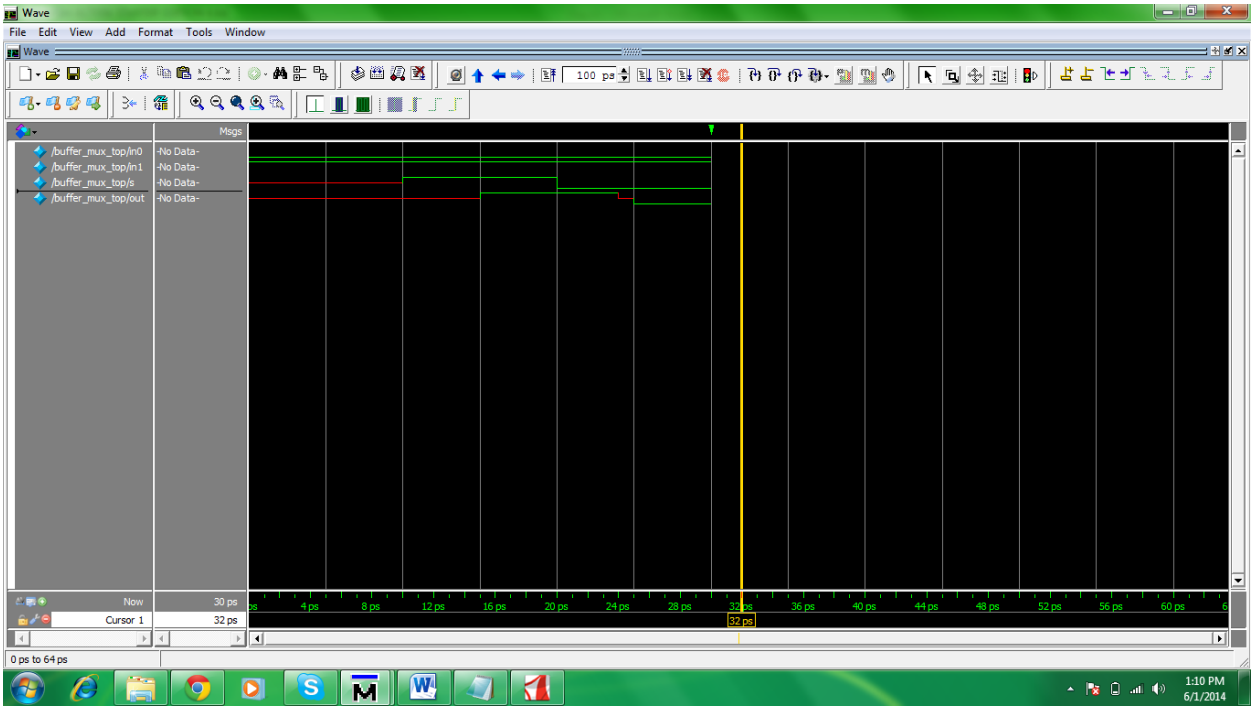
| | Min | Typ | Max |
|---|---|---|---|
| Rise | 1 | 2 | 3 |
| Fall | 3 | 4 | 5 |
| Turnoff | 5 | 6 | 7 |

**Verilog Source Code**

```
module buf_mux (out,in0,in1,s);
  output out;
  input in0,in1,s;
  bufif1 #(1:3:5,2:4:6,3:5:7) b2(out,in1,s);
  bufif0 #(1:3:5,2:4:6,3:5:7) b1(out,in0,s);
endmodule
module buffer_mux_top;
reg in0,in1,s;
wire out;
buf_mux my_mux(out,in0,in1,s);
initial
begin
$monitor($time,"IN1=%b,IN0=%b,----S=%b,OUT=%b\n",in1,in0,s,out);
in0=1'b0;in1=1'b1;
10 s=1'b1;
```

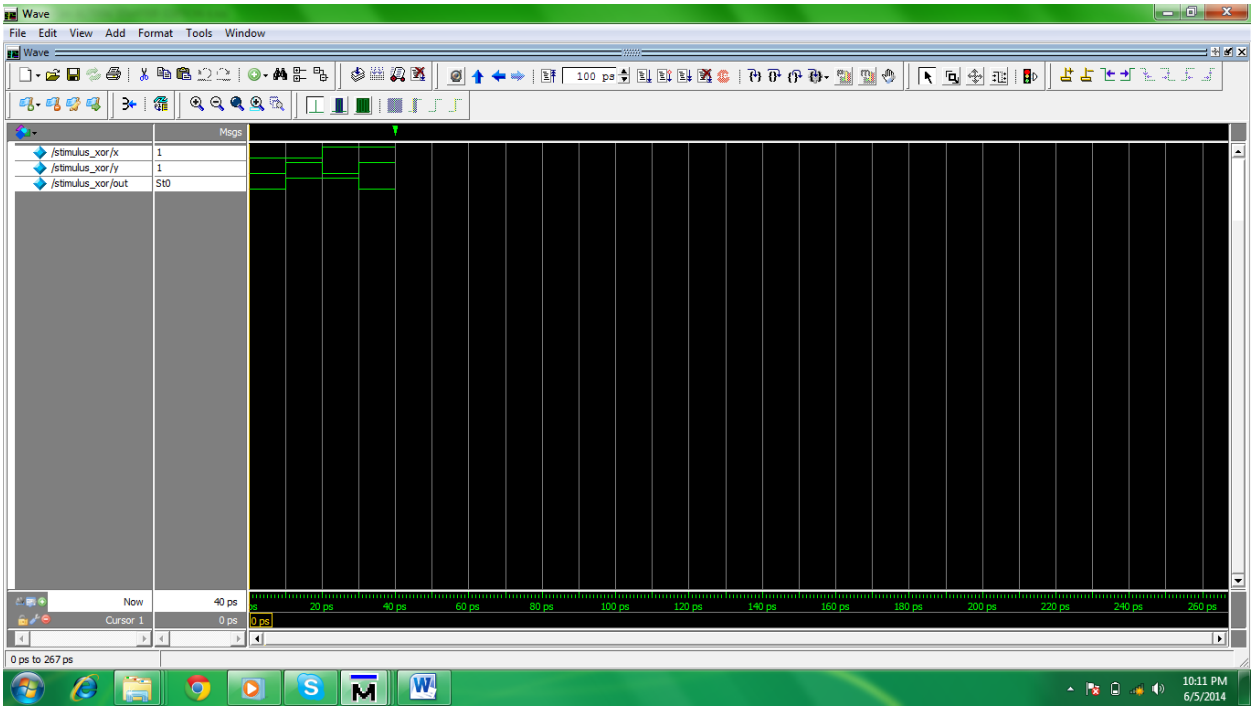
```
#10 s=1'b0;
10 $finish;
end
endmodule
```

**Verilog Simulation Results**



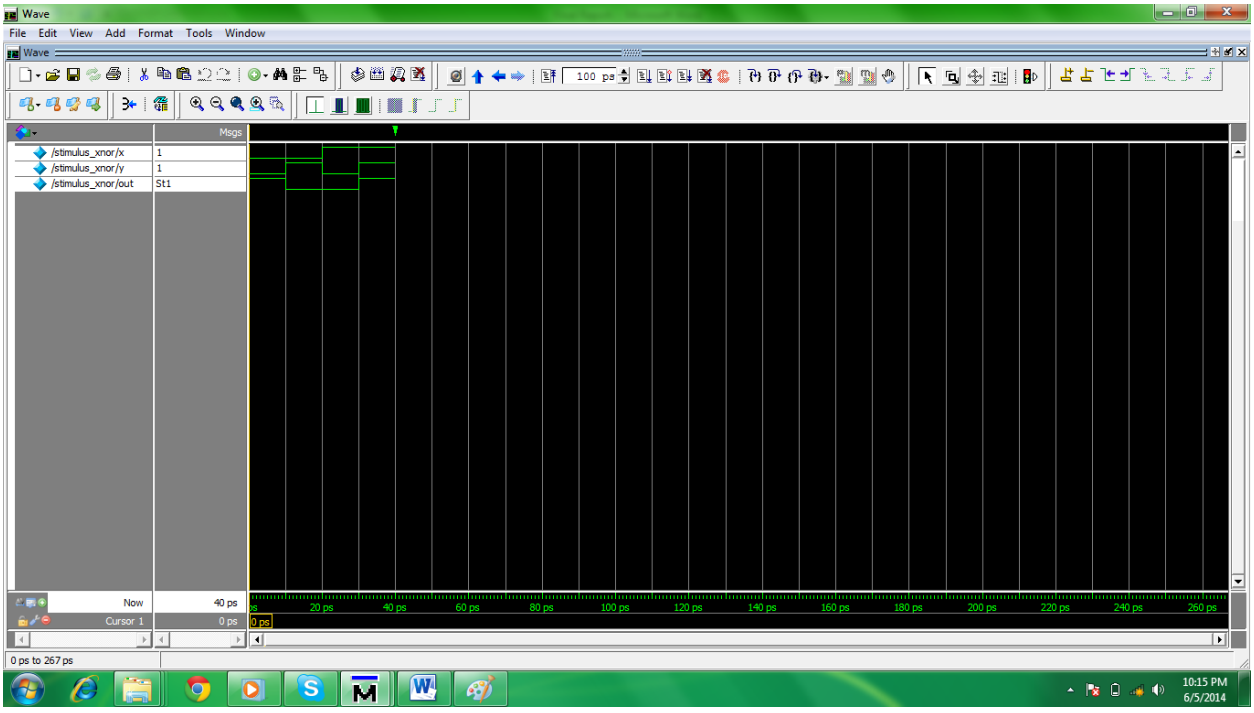
**Experiment#14 XOR using dataflow modeling**

```
module xor_gate(out,x,y);
  output out;
  input x,y;
  assign out=x^y;
endmodule

module stimulus_xor;
  reg x,y;
  wire out;
  xor_gate xorgate(out,x,y);
  initial
  begin
    $monitor("x=%b y=%b---out=%b",x,y,out);
    x=1'b0;y=1'b0;
#10 x=1'b0;y=1'b1;
#10 x=1'b1;y=1'b0;
#10 x=1'b1;y=1'b1;
#10 $finish;
  end
endmodule
```

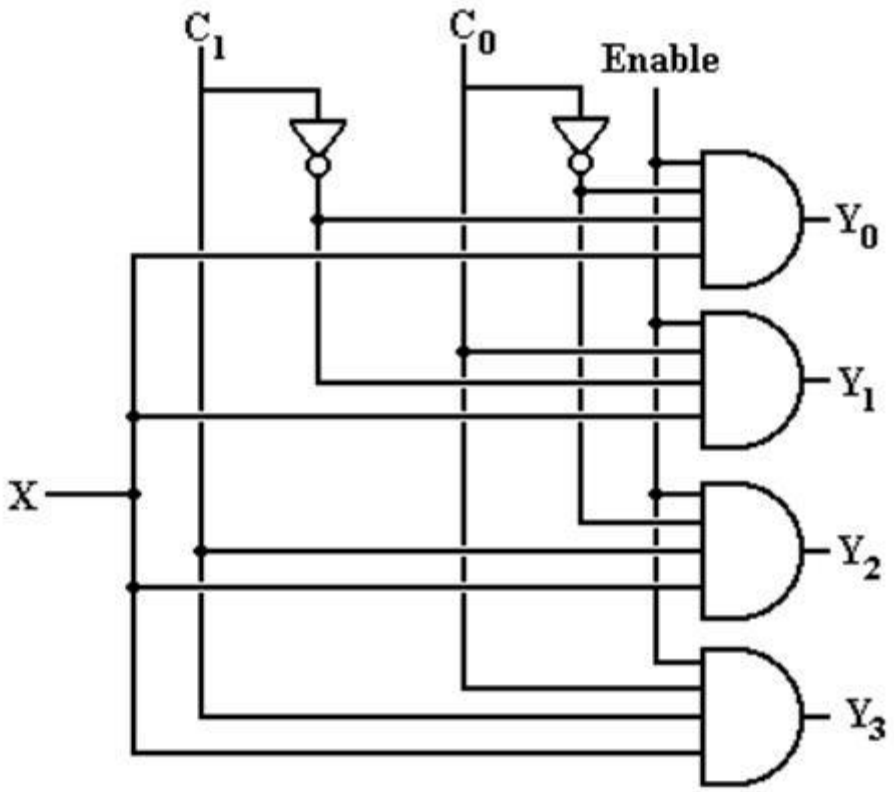


**Experiment#15 XNOR using dataflow modeling**

```
module xnor_gate(out,x,y);
  output out;
  input x,y;
  assign out=x~^y;
endmodule
module stimulus_xnor;
  reg x,y;
  wire out;
    xnor_gate xnorgate(out,x,y);
  initial
  begin
    $monitor("x=%b y=%b---out=%b",x,y,out);
    x=1'b0;y=1'b0;
#10 x=1'b0;y=1'b1;
#10 x=1'b1;y=1'b0;
#10 x=1'b1;y=1'b1;
#10 $finish;
  end
endmodule
```



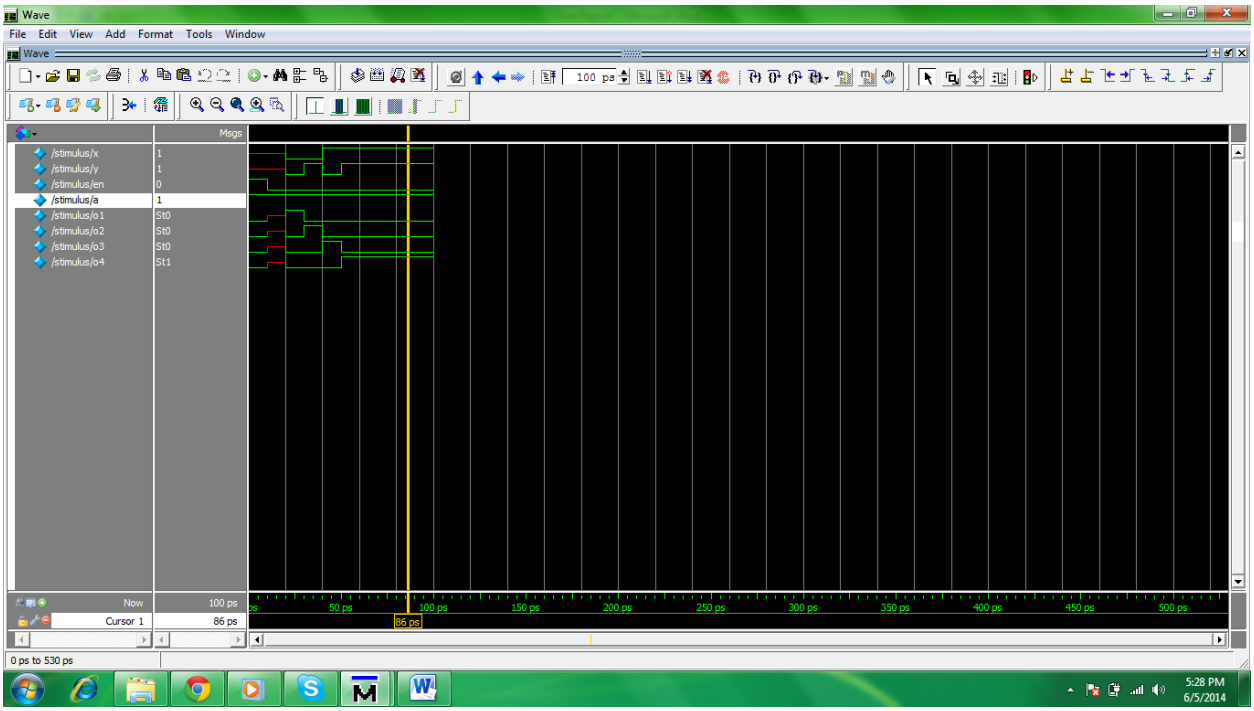

**1:4 demux using implementation in verilog**




## Verilog Code

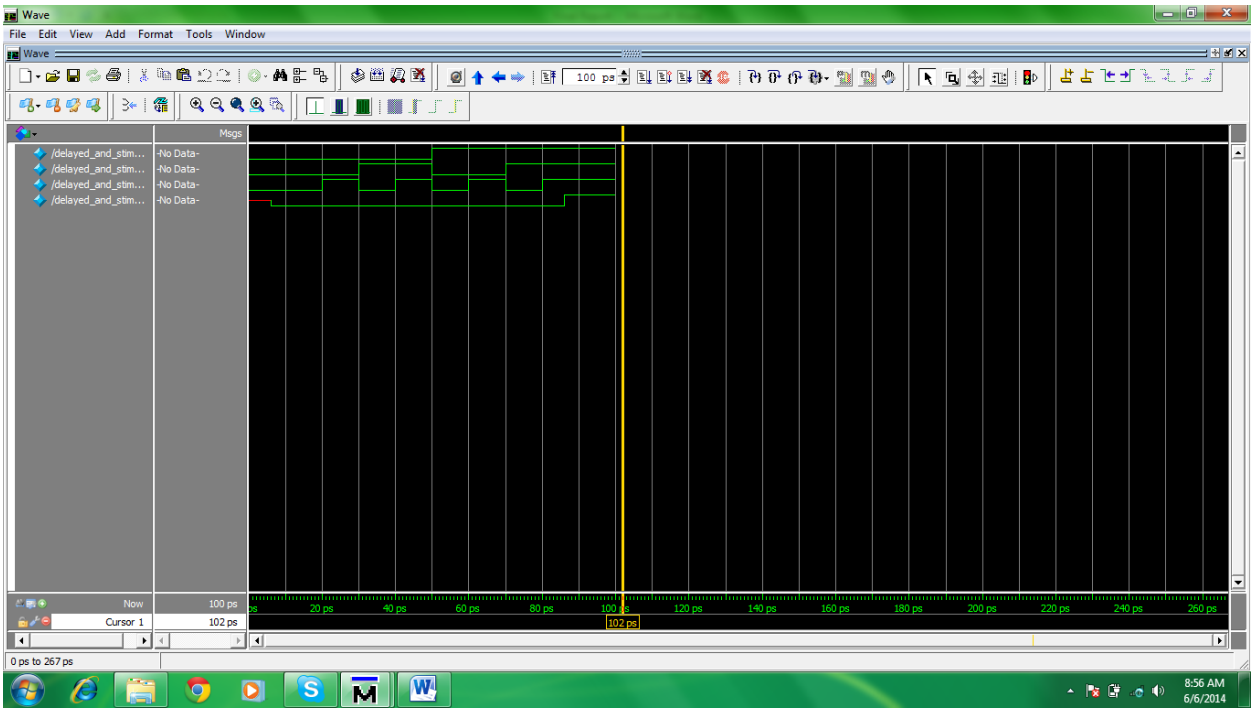
```
module demux(o1,o2,o3,o4,x,y,en,a);

  output o1,o2,o3,o4;

  input x,y,en,a;

  wire x_not,y_not,en_not;


not(en_not,en);

not(x_not,x);

not(y_not,y);


and(o1,x_not,y_not,en_not,a);

and(o2,x_not,y,en_not,a);

and(o3,x,y_not,en_not,a);

and(o4,x,y,en_not,a);


endmodule

module stimulus;
```

```
  reg x,y,en,a;
  wire o1,o2,o3,o4;
  demux d_m1(o1,o2,o3,o4,x,y,en,a);
  initial
  begin
    $monitor(" En=%b, x=%b, y=%b, Input=%b-----o1=%b o2=%b o3=%b o4=%b",en,x,y,a,o1,o2,o3,o4);
    en=1'b1;a=1'b1;
#10 en=1'b0;
#10 x=1'b0;y=1'b0;
#10 x=1'b0;y=1'b1;
#10 x=1'b1;y=1'b0;
#10 x=1'b1;y=1'b1;
  end
endmodule
```




**Cascaded and with 6 unit delay each, output and gate is under a feedback of and gate.**

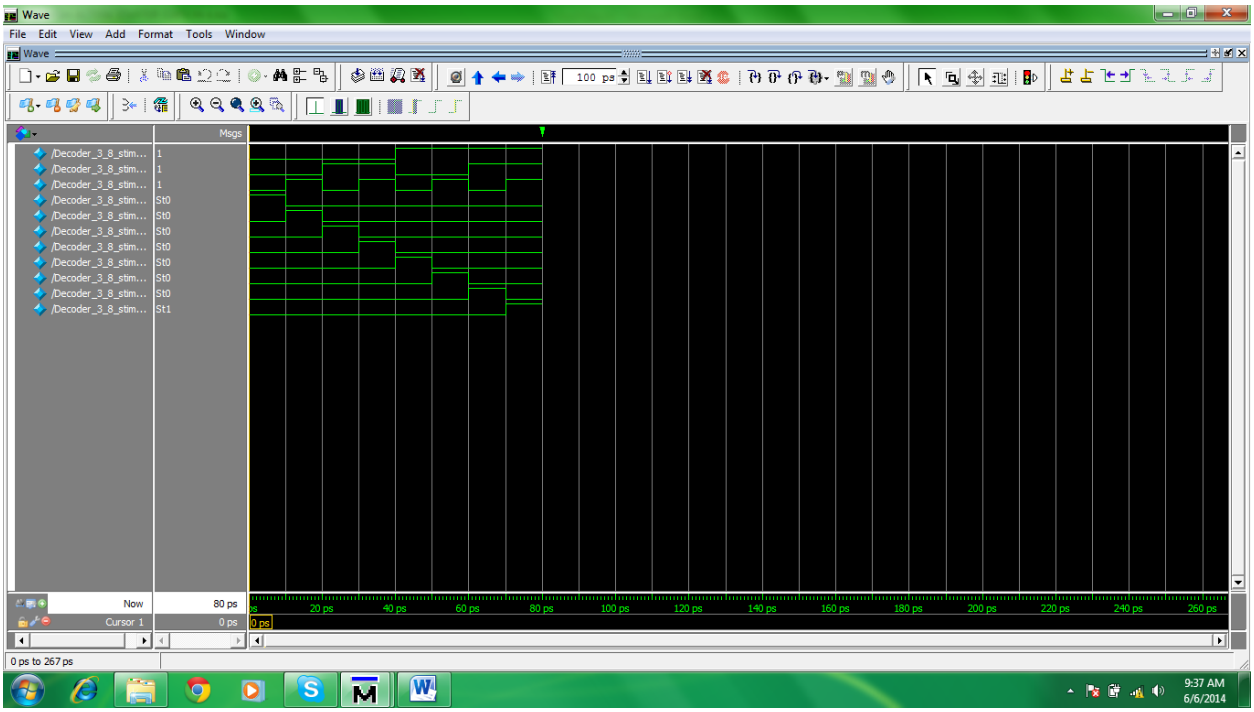
```
module delayed_and(out,x,y,z);
  output out;
  input x,y,z;
  wire e;
  assign #(6) e=x&y;
  assign #(6) out=e&z;
endmodule
module delayed_and_stimulus;
reg x,y,z;
wire out;
delayed_and my_delayed_and(out,x,y,z);
initial
begin
  $monitor("x=%b y= %b z=%b  ----- out=%b ",x,y,z,out);
  x=0;y=0;z=0;
  #10 x=0;y=0;z=0;
  #10 x=0;y=0;z=1;
  #10 x=0;y=1;z=0;
  #10 x=0;y=1;z=1;
  #10 x=1;y=0;z=0;
  #10 x=1;y=0;z=1;
  #10 x=1;y=1;z=0;
  #10 x=1;y=1;z=1;
end
endmodule
```


Wave
File Edit View Add Format Tools Window
Wave
Msgs
/delayed_and_stim...
/delayed_and_stim...
/delayed_and_stim...
/delayed_and_stim...
-No Data-
-No Data-
-No Data-
-No Data-
Now
100 ps
Cursor 1
102 ps
0 ps to 267 ps
8:56 AM
6/6/2014

**3 To 8 Decoder**

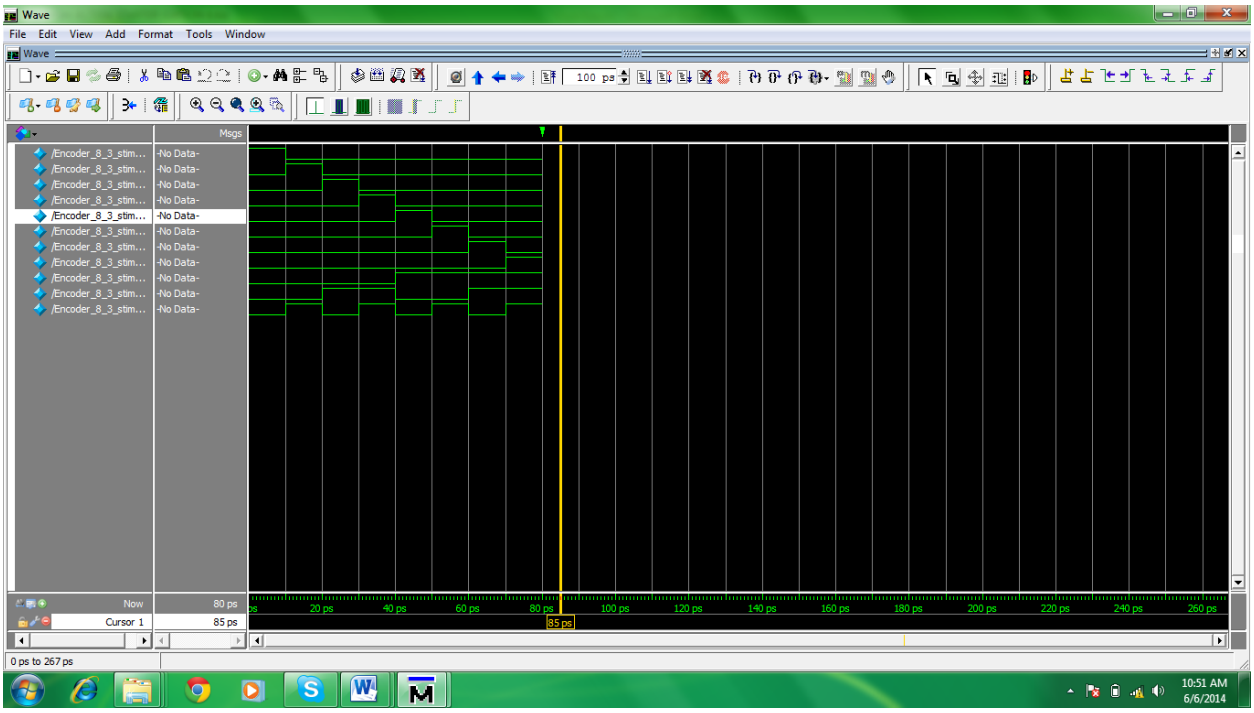
```
module Decoder_3_8(o0,o1,o2,o3,o4,o5,o6,o7,x,y,z);

output o0,o1,o2,o3,o4,o5,o6,o7;

input x,y,z;

wire x_not,y_not,z_not;

assign x_not=~x;

assign y_not=~y;

assign z_not=~z;

assign o0=x_not&y_not&z_not;

assign o1=x_not&y_not&z;

assign o2=x_not&y&z_not;

assign o3=x_not&y&z;

assign o4=x&y_not&z_not;

assign o5=x&y_not&z;

assign o6=x&y&z_not;

assign o7=x&y&z;

endmodule


module Decoder_3_8_stimulus;

reg x,y,z;

wire o0,o1,o2,o3,o4,o5,o6,o7;

Decoder_3_8 my_Decoder_3_8(o0,o1,o2,o3,o4,o5,o6,o7,x,y,z);

  initial

  begin

    $monitor("x=%b y=%b z=%b o0=%b o1=%b o2=%b o3=%b o4=%b o5=%b o6=%b o7=%b
",x,y,z,o0,o1,o2,o3,o4,o5,o6,o7);

    x=1'b0;y=1'b0;z=1'b0;

    #10 x=1'b0;y=1'b0;z=1'b1;

    #10 x=1'b0;y=1'b1;z=1'b0;

    #10 x=1'b0;y=1'b1;z=1'b1;
```

```
    #10 x=1'b1;y=1'b0;z=1'b0;
    #10 x=1'b1;y=1'b0;z=1'b1;
    #10 x=1'b1;y=1'b1;z=1'b0;
    #10 x=1'b1;y=1'b1;z=1'b1;
  end
endmodule
```




**8 To 3 Encoder**

```
module Encoder_8_3(x,y,z,o0,o1,o2,o3,o4,o5,o6,o7);
  output x,y,z;
  input o0,o1,o2,o3,o4,o5,o6,o7;
assign x=o4|o5|o6|o7;
assign y=o2|o3|o6|o7;
assign z=o1|o3|o5|o7;
endmodule

module Encoder_8_3_stimulus;
  reg o0,o1,o2,o3,o4,o5,o6,o7;
  wire x,y,z;
  Encoder_8_3 my_encoder(x,y,z,o0,o1,o2,o3,o4,o5,o6,o7);

  initial
  begin
    $monitor("x=%b y=%b z=%b o0=%b o1=%b o2=%b o3=%b o4=%b o5=%b o6=%b
o7=%b",x,y,z,o0,o1,o2,o3,o4,o5,o6,o7);
    o0=1'b1;o1=1'b0;o2=1'b0;o3=1'b0;o4=1'b0;o5=1'b0;o6=1'b0;o7=1'b0;
  #10 o0=1'b0;o1=1'b1;o2=1'b0;o3=1'b0;o4=1'b0;o5=1'b0;o6=1'b0;o7=1'b0;
  #10 o0=1'b0;o1=1'b0;o2=1'b1;o3=1'b0;o4=1'b0;o5=1'b0;o6=1'b0;o7=1'b0;
  #10 o0=1'b0;o1=1'b0;o2=1'b0;o3=1'b1;o4=1'b0;o5=1'b0;o6=1'b0;o7=1'b0;
  #10 o0=1'b0;o1=1'b0;o2=1'b0;o3=1'b0;o4=1'b1;o5=1'b0;o6=1'b0;o7=1'b0;
  #10 o0=1'b0;o1=1'b0;o2=1'b0;o3=1'b0;o4=1'b0;o5=1'b1;o6=1'b0;o7=1'b0;
  #10 o0=1'b0;o1=1'b0;o2=1'b0;o3=1'b0;o4=1'b0;o5=1'b0;o6=1'b1;o7=1'b0;
  #10 o0=1'b0;o1=1'b0;o2=1'b0;o3=1'b0;o4=1'b0;o5=1'b0;o6=1'b0;o7=1'b1;
  #10 $finish;
    end
endmodule
```




**1:8 Demux implementation.**

module Demux_1_8(o0,o1,o2,o3,o4,o5,o6,o7,x,en,s0,s1,s2);

```
  output o0,o1,o2,o3,o4,o5,o6,o7;
  input x,en,s0,s1,s2;
  wire en_not=~en;
  wire s0_not=~s0;
  wire s1_not=~s1;
  wire s2_not=~s2;
assign o0 =x&en_not&s0_not&s1_not&s2_not;
  assign o1 =x&en_not&s0_not&s1_not&s2;
  assign o2 =x&en_not&s0_not&s1&s2_not;
  assign o3 =x&en_not&s0_not&s1&s2;
  assign o4 =x&en_not&s0&s1_not&s2_not;
  assign o5 =x&en_not&s0&s1_not&s2;
  assign o6 =x&en_not&s0&s1&s2_not;
  assign o7 =x&en_not&s0&s1&s2;
  endmodule
module Demux_1_8_stimulus;
  reg x,en,s0,s1,s2;
  wire o0,o1,o2,o3,o4,o5,o6,o7;
  Demux_1_8 my_demux(o0,o1,o2,o3,o4,o5,o6,o7,x,en,s0,s1,s2);
  initial
  begin
    $monitor("o0=%b o1=%b o2=%b o3=%b o4=%b o5=%b o6=%b o7=%b x=%b en=%b s0=%b s1=%b s2=%b
",o0,o1,o2,o3,o4,o5,o6,o7,x,en,s0,s1,s2);
  x=1'b1;
  en=1'b1;
  #10 en=1'b0;
  s2=1'b0;s1=1'b0;s0=1'b0;
  #10 s2=1'b0;s1=1'b0;s0=1'b1;
  #10 s2=1'b0;s1=1'b1;s0=1'b0;
  #10 s2=1'b0;s1=1'b1;s0=1'b1;
```

```
  #10 s2=1'b1;s1=1'b0;s0=1'b0;
 #10 s2=1'b1;s1=1'b0;s0=1'b1;
 #10 s2=1'b1;s1=1'b1;s0=1'b0;
 #10 s2=1'b1;s1=1'b1;s0=1'b1;
  end
endmodule
```