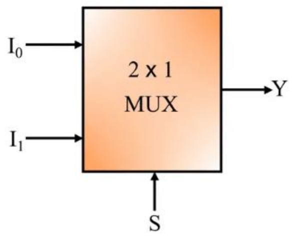
# Experiment-6

## Design Verilog program to implement Different types of multiplexers like 2:1, 4:1 and 8:1.

**Aim:** To design Verilog HDL code to implement 2:1, 4:1 and 8:1 multiplexer.

## a) 2:1 Multiplexer

### Symbol



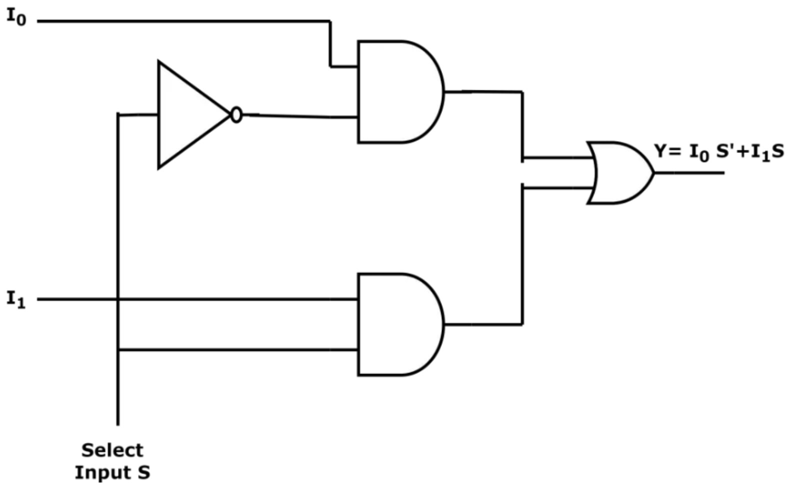
### Truth table

| Input (S) | Output (Y) |
|---|---|
| 0 | $I_0$ |
| 1 | $I_1$ |

### Expression

$$Y=S'.I0+S.I1$$

### Logic Diagram



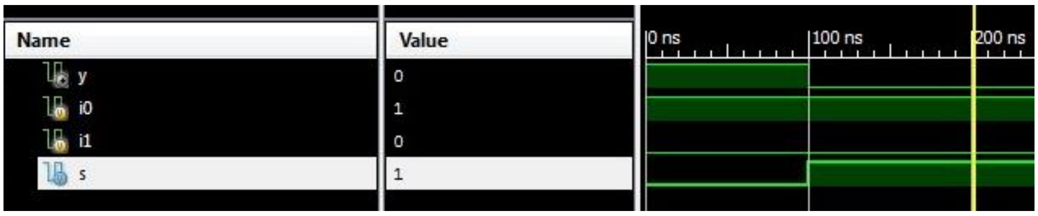
### Verilog Dataflow description code

```
module mux_21(s, i0, i1, y);
input s, i0, i1;
output y;
assign y = s ? i1 : i0;        // or    y=((~select) & i0) | (Select & i1);
endmodule
```

**Testbench code**

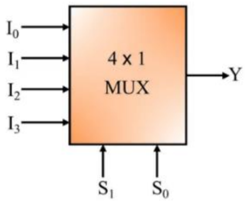
```
module mux21_tb;
  reg s, i0, i1;
  wire y;
  mux_21 uut(.s(s), .i0(i0), .i1(i1), .y(y));
  initial begin
     s=0; i0=1; i1=0; #100;
     s=1; i0=1; i1=0; #100;
  end
endmodule
```

**Timing Diagram**



## b) 4:1 Multiplexer

**Symbol**



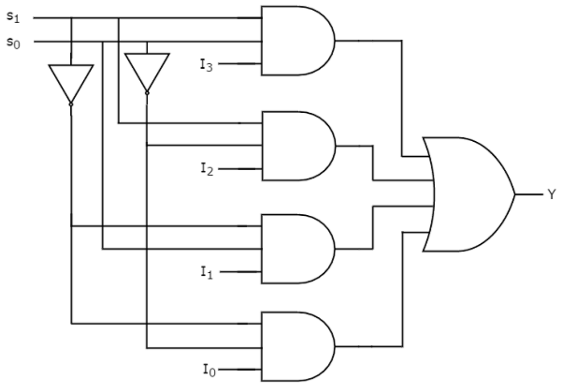
**Truth table**

| Inputs | | Output |
|---|---|---|
| S1 | S0 | Y |
| 0 | 0 | I0 |
| 0 | 1 | I1 |
| 1 | 0 | I2 |
| 1 | 1 | I3 |

**Expression**

Y=S1‘ S0‘ I0+S1‘ S0 I1+S1 S0‘ I2+S1 S0 I3

**Logic Diagram**



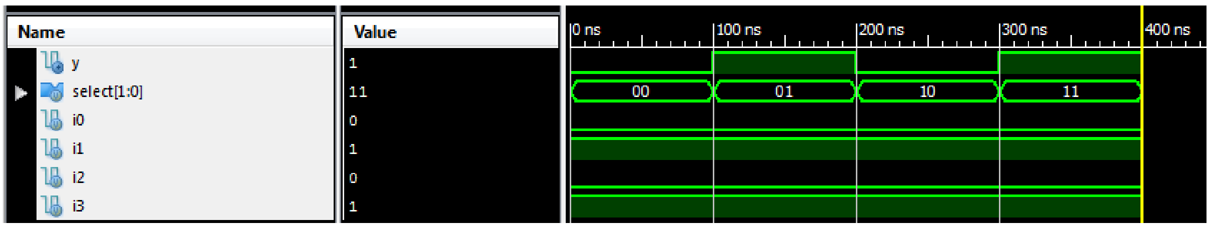
## Verilog Behavioral Description Code

```
module mux_41 (select, i0, i1, i2, i3, y);
input [1:0] select;
input i0, i1, i2, i3;
output reg y;
always @ (*)
   begin
   case (select)
      2'b00 : y=i0;
     2'b01 : y=i1;
     2'b10 : y=i2;
     2'b11 : y=i3;
   endcase
  end
endmodule
```

## Test bench code

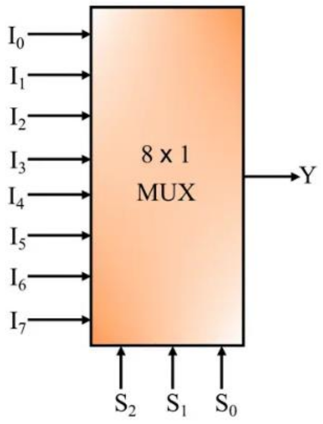
```
module mux41_tb;
reg [1:0] select;
reg i0, i1, i2, i3;
wire y;
mux_41 uut (.select(select), .i0(i0), .i1(i1), .i2(i2), .i3(i3), .y(y));
initial begin
select=2'b00; i0=0; i1=1; i3=0; i4=1; #100;
select=2'b01; i0=0; i1=1; i3=0; i4=1; #100;
select=2'b10; i0=0; i1=1; i3=0; i4=1; #100;
select=2'b11; i0=0; i1=1; i3=0; i4=1; #100;
end
endmodule
```

## Timing Diagram



### c) 8:1 Multiplexer

### Symbol



### Truth table

| Input | | | Output |
|---|---|---|---|
| S2 | S1 | S0 | y |
| 0 | 0 | 0 | I0 |
| 0 | 0 | 1 | I1 |
| 0 | 1 | 0 | I2 |
| 0 | 1 | 1 | I3 |
| 1 | 0 | 0 | I4 |
| 1 | 0 | 1 | I5 |
| 1 | 1 | 0 | I6 |
| 1 | 1 | 1 | I7 |

### Expression

Y=S0'.S1'.S2'.I0+S0.S1'.S2'I1+S0'.S1.S2'.I2+S0.S1.S2'.I3+S0'.S1'.S2 I4+S0.S1'.S2 I5 +S0'.S1.S2 .I6+S0.S1.S3.I7

### Verilog Behavioral description code

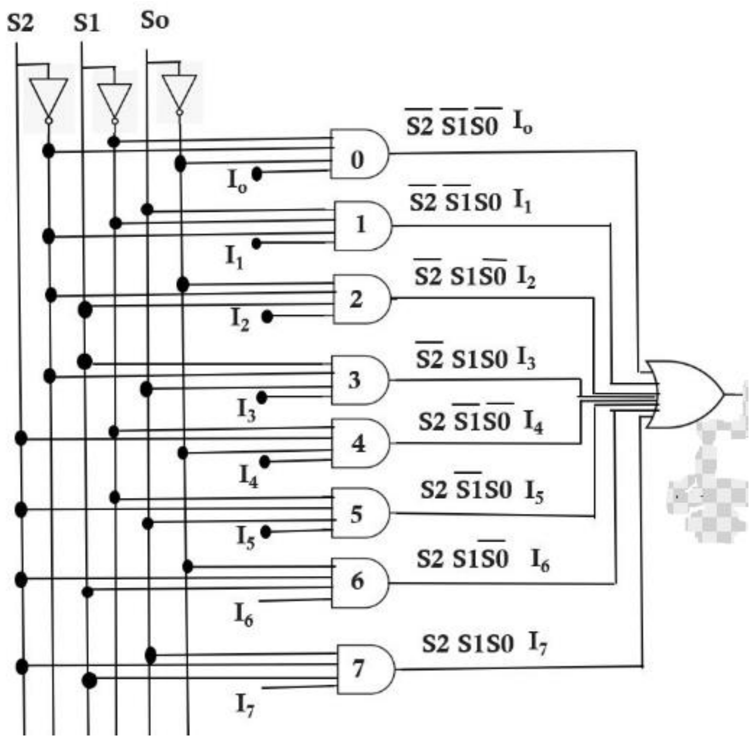
```
module mux_81(select, i, y);
   input [7:0] i;
   input [2:0] select;
   output reg y;
   always @ (*)
   begin
   case(select)
      3'b000: y=i[0];
      3'b001: y=i[1];
      3'b010: y=i[2];
      3'b011: y=i[3];
      3'b100: y=i[4];
      3'b101: y=i[5];
```

```
        3'b110: y=i[6];
        3'b111: y=i[7];
    endcase
  end
endmodule
```

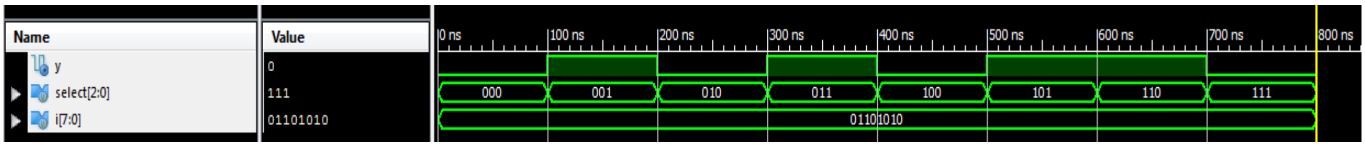
**Testbench code**

```
module mux81_tb;
reg [7:0] i;
reg [2:0] select;
wire y;
mux_81 uut (.select(select), .i(i), .y(y));
initial begin
select=3’b000; i=8’b01101010; #100;
select=3’b001; i=8’b01101010; #100;
select=3’b010; i=8’b01101010; #100;
select=3’b011; i=8’b01101010; #100;
select=3’b100; i=8’b01101010; #100;
select=3’b101; i=8’b01101010; #100;
select=3’b110; i=8’b01101010; #100;
select=3’b111; i=8’b01101010; #100;
end
endmodule
```

**Logic diagram**


S2
S1
So
$\overline{S2}\,\overline{S1}\,\overline{S0}$ $I_0$
$\overline{S2}\,\overline{S1}S0$ $I_1$
$\overline{S2}\,S1\overline{S0}$ $I_2$
$\overline{S2}\,S1S0$ $I_3$
$S2\,\overline{S1}\,\overline{S0}$ $I_4$
$S2\,\overline{S1}S0$ $I_5$
$S2\,S1\overline{S0}$ $I_6$
$S2\,S1S0$ $I_7$
0
1
2
3
4
5
6
7
$I_0$
$I_1$
$I_2$
$I_3$
$I_4$
$I_5$
$I_6$
$I_7$

## Timing Diagram



## Result

Verilog code for different types of multiplexer are designed and simulated.