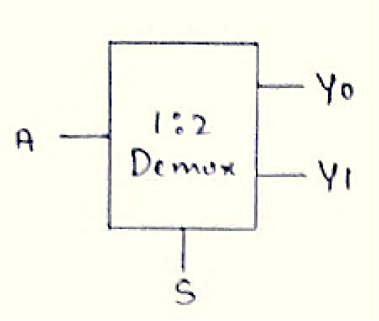
Experiment-7

Design Verilog program to implement types of De-Multiplexer

Aim: To develop Verilog code to implement types of De-multiplexer.

a) 1:2 Demux

Symbol



Truth Table

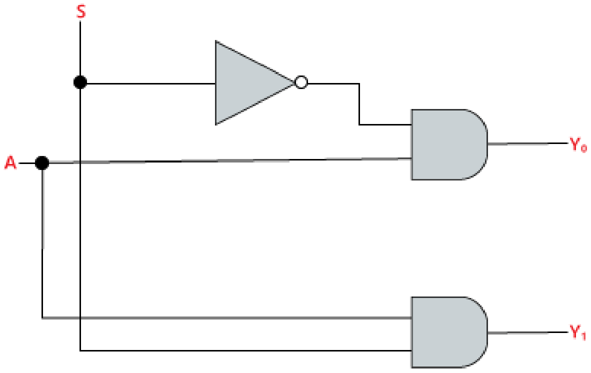
Input	Outputs	
S	Y1	Y0
0	0	A
1	A	0

Expression

Y1=S.A

Y0=S'.A

Logic Diagram



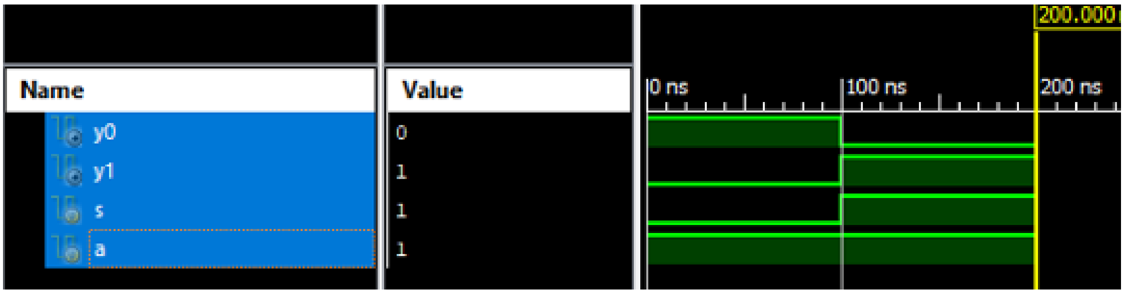
Verilog dataflow description code

```
module demux_12 (s, a, y0, y1);
input s, a;
output y0, y1;
assign {y0, y1}=s?{1'b0, a}:{a,1'b0};
endmodule
```

Testbench code

```
module demux12_tb;
reg s;
reg a;
wire y0;
wire y1;
demux_12 uut (.s(s), .a(a), .y0(y0), .y1(y1));
initial begin
s =0; a=1; #100;
s =1; a=1; #100;
end
endmodule
```

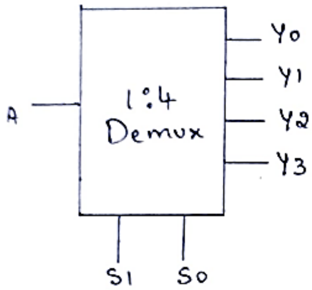
Timing Diagram



Name	Value
y0	0
y1	1
s	1
a	1

b) 1:4 Demux

Symbol



Truth Table

Inputs		Outputs			
S_1	S_0	Y_3	Y_2	Y_1	Y0
0	0	0	0	0	A
0	1	0	0	A	0
1	0	0	A	0	0
1	1	A	0	0	0

Expression

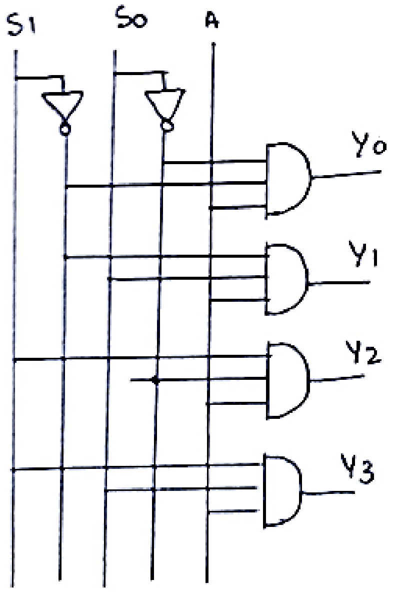
Y0 = S1’. S0’. A

Y3 = S1’. S0. A

Y3 = S1. S0’. A

Y3 = S1. S0. A

Logic Diagram



Verilog behavioral description code

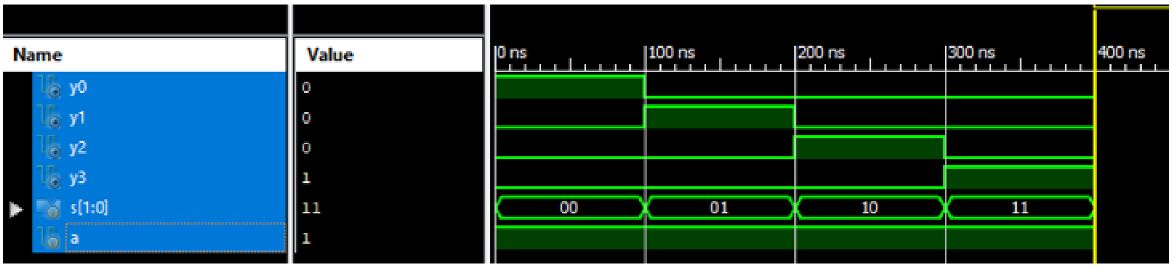
```
module demux_14(s, a, y0, y1, y2, y3);
input [1:0] s;
input a;
output reg y0, y1, y2, y3;
always @ (*)
begin
  case(s)
    2’b00: {y3, y2, y1, y0} = {3’b0, a};
    2’b01: {y3, y2, y1, y0} = {2’b0, a, 1’b0};
    2’b10: {y3, y2, y1, y0} = {1’b0, a, 2’b0};
    2’b11: {y3, y2, y1, y0} = {a, 3’b0};
  endcase
 end
endmodule
```

Testbench code

```
module demux14_tb;
reg [1:0] s;
reg a;
wire y0, y1, y2, y3;
```

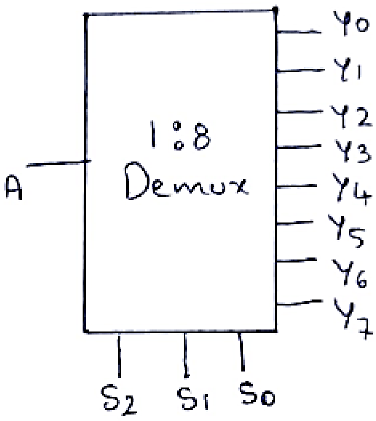
```
demux_14 uut(.s(s), .a(a), .y0(y0), .y1(y1), .y2(y2), .y3(y3));
initial begin
s=2’b00; a=1; #100;
s=2’b01; a=1; #100;
s=2’b10; a=1; #100;
s=2’b11; a=1; #100;
end
endmodule
```

Timing Diagram



c) 1:8 Demux

Symbol



Truth Table

Inputs			Outputs							
S_2	S_1	S_0	Y_7	Y_6	Y_5	Y_4	Y_3	Y_2	Y_1	Y_0
0	0	0	0	0	0	0	0	0	0	A
0	0	1	0	0	0	0	0	0	A	0
0	1	0	0	0	0	0	0	A	0	0
0	1	1	0	0	0	0	A	0	0	0
1	0	0	0	0	0	A	0	0	0	0
1	0	1	0	0	A	0	0	0	0	0
1	1	0	0	A	0	0	0	0	0	0
1	1	1	A	0	0	0	0	0	0	0

Expression

$Y0=S2'.S1'.S0'.A$

$Y1=S2'.S1'.S0.A$

$Y2=S2'.S1.S0'.A$

$Y3=S2'.S1.S0.A$

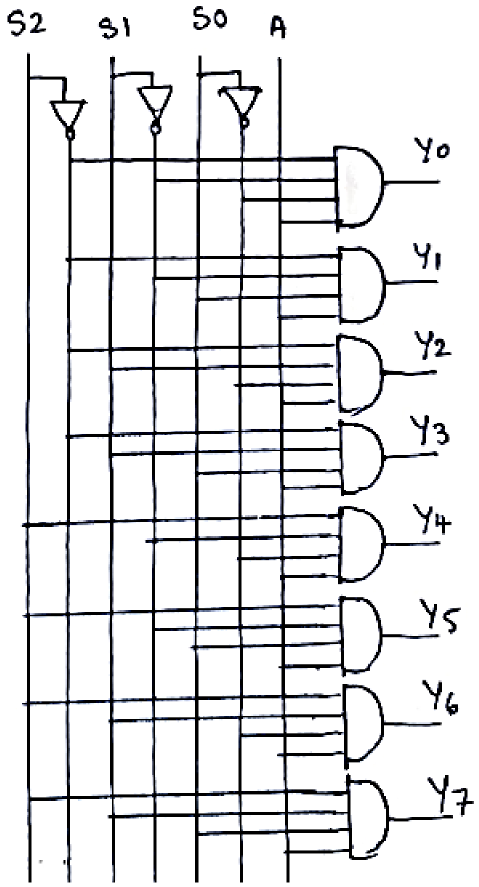
$Y4=S2.S1'.S0'.A$

$Y5=S2.S1'.S0.A$

$Y6=S2.S1.S0'.A$

$Y7=S2.S1.S0.A$

Logic Diagram



Verilog behavioral description code

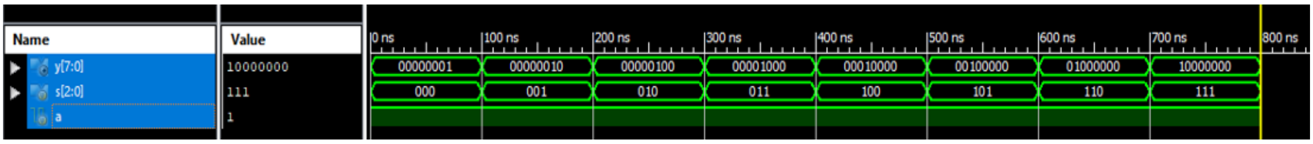
```
module demux_18(s, a, y);
input [2:0] s;
input a;
output reg [7:0] y;
always @ (*)
```

```
begin
case (s)
3’b000: {y[7], y[6], y[5], y[4], y[3], y[2], y[1], y[0]}={7’b0,a};
3’b001: {y[7], y[6], y[5], y[4], y[3], y[2], y[1], y[0]}={6’b0, a, 1’b0};
3’b010: {y[7], y[6], y[5], y[4], y[3], y[2], y[1], y[0]}={5’b0, a, 2’b0};
3’b011: {y[7], y[6], y[5], y[4], y[3], y[2], y[1], y[0]}={4’b0, a, 3’b0};
3’b100: {y[7], y[6], y[5], y[4], y[3], y[2], y[1], y[0]}={3’b0, a, 4’b0};
3’b101: {y[7], y[6], y[5], y[4], y[3], y[2], y[1], y[0]}={2’b0, a, 5’b0};
3’b110: {y[7], y[6], y[5], y[4], y[3], y[2], y[1], y[0]}={1’b0, a, 6’b0 };
3’b111: {y[7], y[6], y[5], y[4], y[3], y[2], y[1], y[0]}={a, 7’b0};
endcase
end
endmodule
```

Testbench code

```
module demux18_tb;
reg [2:0] s;
reg a;
wire [7:0] y;
demux_18 uut(.s(s), .a(a), .y(y));
initial begin
s=3’b000; a=1; #100;
s=3’b001; a=1; #100;
s=3’b010; a=1; #100;
s=3’b011; a=1; #100;
s=3’b100; a=1; #100;
s=3’b101; a=1; #100;
s=3’b110; a=1; #100;
s=3’b111; a=1; #100;
end
endmodule
```

Timing Diagram



Result

Verilog programs are developed to implement types of De-Multiplexer.