**1) Given a 4-variable logic expression, simplify it using appropriate technique and simulate the same using basic gates.**

1. F(W,X,Y,Z)=∑(1,3,4,5,6,7,9,11,13,15)

WX

00

YZ

00

01

11

10

1

1

01

1

1

1

1

11

1

1

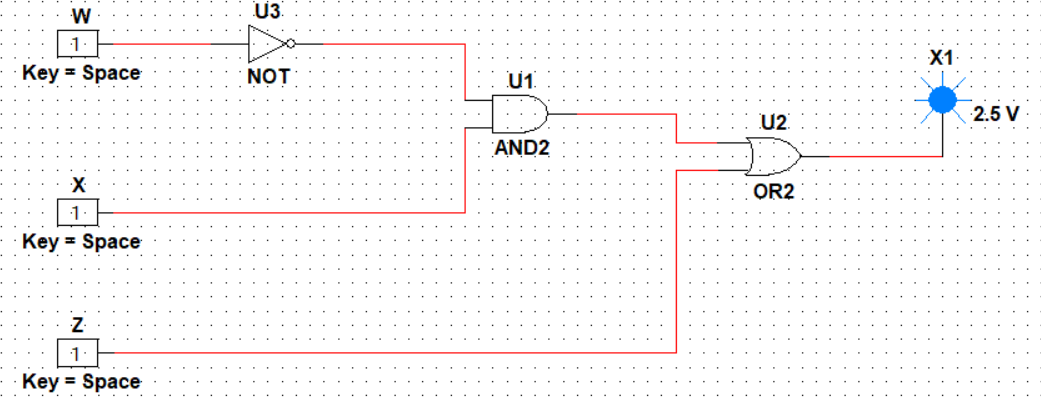
10

1

1

# F=Z+W’X

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **W** | **X** | **Y** | **Z** | **F** |
| 0 | 0 | 0 | 0 | 0 |
| 0 | 0 | 0 | 1 | 1 |
| 0 | 0 | 1 | 0 | 0 |
| 0 | 0 | 1 | 1 | 1 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 0 |
| 1 | 0 | 0 | 1 | 1 |
| 1 | 0 | 1 | 0 | 0 |
| 1 | 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |



2. F(A,B,C,D)=∑(0,2,4,5,6,7,8,10,13,15)

# 00 01 11 10

AB CD

00

|  |  |  |  |
| --- | --- | --- | --- |
| 1 |  |  | 1 |
| 1 | 1 | 1 | 1 |
|  | 1 | 1 |  |
| 1 |  |  | 1 |

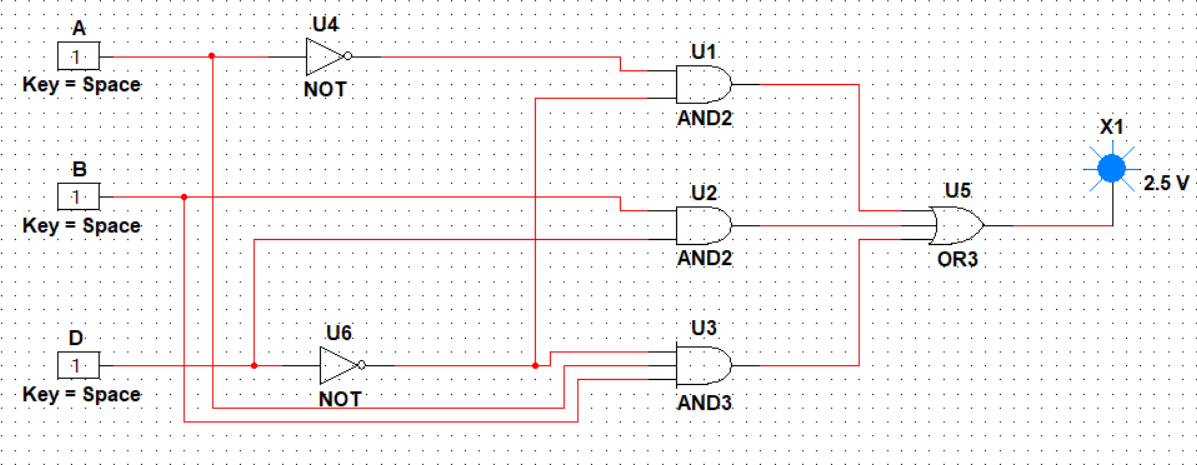
# 01

11

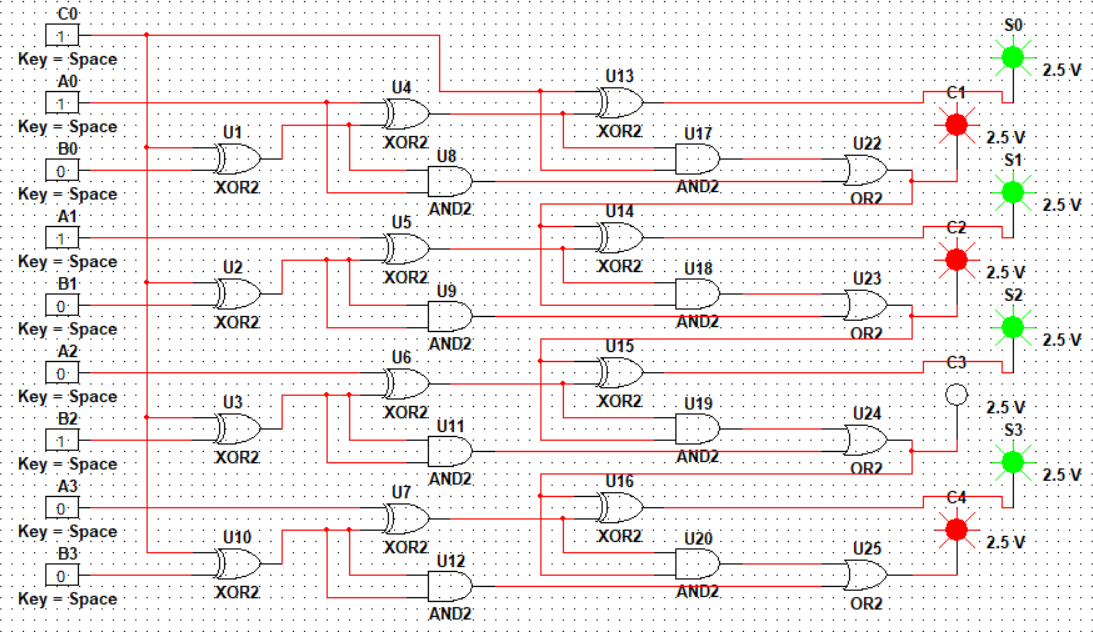
# 10

F=A’D’+BD+ABD’

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **A** | **B** | **C** | **D** | **F** |
| 0 | 0 | 0 | 0 | 1 |
| 0 | 0 | 0 | 1 | 0 |
| 0 | 0 | 1 | 0 | 1 |
| 0 | 0 | 1 | 1 | 0 |
| 0 | 1 | 0 | 0 | 1 |
| 0 | 1 | 0 | 1 | 1 |
| 0 | 1 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 | 1 |
| 1 | 0 | 0 | 0 | 1 |
| 1 | 0 | 0 | 1 | 0 |
| 1 | 0 | 1 | 0 | 1 |
| 1 | 0 | 1 | 1 | 0 |
| 1 | 1 | 0 | 0 | 0 |
| 1 | 1 | 0 | 1 | 1 |
| 1 | 1 | 1 | 0 | 0 |
| 1 | 1 | 1 | 1 | 1 |



# **2) Design a 4-bit full adder and subtractor and simulate the same using basic gates.**



**[Results verified by truth table]**

1. **Design Verilog HDL to implement simple circuits using structural, dataflow and behavioral model.**

**Testbench code:**

module TestModule;

//Inputs reg a,b,c;

//Outputs wire d,e;

//Instantiate the Unit Under Test(UUT) LogicGates uut(.a(a),.b(b),.c(c),.d(d),.e(e)); initial begin

$dumpfile("dump.vcd");

$dumpvars(1); end

initial begin

//Initialize Inputs a=0;b=0;c=0;

#5 a=0;b=0;c=1;

#5 a=0;b=1;c=0;

#5 a=0;b=1;c=1;

#5 a=1;b=0;c=0;

#5 a=1;b=0;c=1;

#5 a=1;b=1;c=0;

#5 a=1;b=1;c=1;

#5;

$finish; end endmodule

**Verilog code for structural model:**

module LogicGates(a,b,c,d,e); input a,b,c;

output d,e; wire w1; and(w1,a,b);

not(e,c);

or(d,w1,e); endmodule

**EpWave for structural model:**

**Verilog code for dataflow model:** module LogicGates(a,b,c,d,e); input a,b,c;

output d,e;

assign d=(a&&b)||(!c); assign e=!c;

endmodule

**EpWave for dataflow model:**

**Verilog code for behavioral model:**

module LogicGates(a,b,c,d,e); input a,b,c;

output reg d,e; always @(a,b,c) begin

case({a,b,c}) 3'b000: d=1;

3'b001: d=0;

3'b010: d=1;

3'b011: d=0;

3'b100: d=1;

3'b101: d=0;

3'b110: d=1;

3'b111: d=1;

default:d=0;

endcase case({a,b,c}) 3'b000: e=1;

3'b001: e=0;

3'b010: e=1;

3'b011: e=0;

3'b100: e=1;

3'b101: e=0;

3'b110: e=1;

3'b111: e=0;

default:e=0; endcase

end endmodule

**EpWave for behavioral model:**



# **[Results verified by truth table]**

1. **Design Verilog HDL to implement Binary Adder-Subtractor—Half and Full Adder, Half and Full Subtractor.**

**Testbench code for half adder:**

module TestModule; reg x,y;

wire sum,carryout;

// Instantiate the Unit Under Test (UUT)

halfadder uut(.x(x),.y(y), .sum(sum),.carryout(carryout)); initial begin

$dumpfile("dump.vcd");

$dumpvars(1); end

initial begin

// Initialize Inputs x = 0; y = 0;

#5 x = 0;y = 1;

#5 x = 1;y = 0;

#5 x = 1;y = 1;

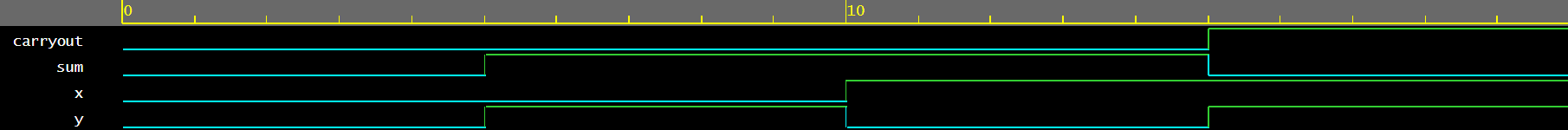
#5;

$finish end endmodule

**Verilog code for half adder:** module halfadder(x,y,sum,carryout); input x,y;

output sum,carryout;

assign sum=((!x)&&y)||(x&&(!y)); assign carryout=(x&&y); endmodule

**EpWave for half adder:**

**Testbench code for full adder:**

module TestModule; reg x,y,z;

wire sum,carryout;

fulladder uut(.x(x),.y(y),.z(z),.sum(sum),.carryout(carryout)); initial begin

$dumpfile("dump.vcd");

$dumpvars(1); end

initial begin x=0;y=0;z=0;

#5 x=0;y=0;z=1;

#5 x=0;y=1;z=0;

#5 x=0;y=1;z=1;

#5 x=1;y=0;z=0;

#5 x=1;y=0;z=1;

#5 x=1;y=1;z=0;

#5 x=1;y=1;z=1;

#5;

$finish; end endmodule

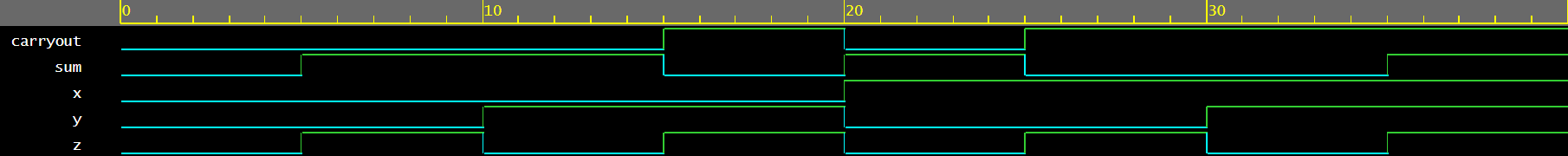
**Verilog code for full adder:**

module fulladder(x,y,z,sum,carryout); input x,y,z;

output sum,carryout;

assign sum=((!x)&&(!y)&&z)||((!x)&&y&&(!z))||(x&&(!y)&&(!z))||(x&&y&&z); assign carryout=(x&&y)||(x&&z)||(y&&z);

endmodule

**EpWave for full adder:**

**Testbench code for half subtractor:**

module TestModule; reg x,y;

wire diff,brr;

halfsub uut(.x(x),.y(y),.diff(diff),.brr(brr)); initial begin

$dumpfile("dump.vcd");

$dumpvars(1); end

initial begin x=0;y=0; #5 x=0;y=1;

#5 x=1;y=0;

#5 x=1;y=1;

#5;

$finish; end endmodule

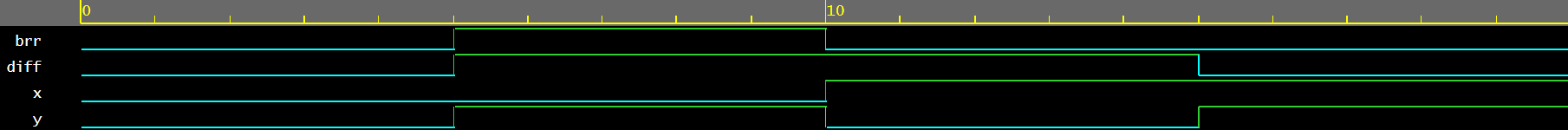
**Verilog code for half subtractor:**

module halfsub(x,y,diff,brr); input x,y;

output diff,brr;

assign diff=((!x)&&y)||(x&&(!y)); assign brr=((!x)&&y);

endmodule

**EpWave for half subtractor:**

**Testbench code for full subtractor:**

module TestModule; reg x,y,z;

wire diff,brr;

// Instantiate the Unit Under Test (UUT)

fullsub uut(.x(x),.y(y),.z(z),.diff(diff),.brr(brr)); initial begin

$dumpfile("dump.vcd");

$dumpvars(1); end

initial begin

// Initialize Inputs x = 0;y = 0;z = 0;

#5 x = 0;y = 0;z = 1; #5 x = 0;y = 1;z = 0; #5 x = 0;y = 1;z = 1; #5 x = 1;y = 0;z = 0; #5 x = 1;y = 0;z = 1; #5 x = 1;y = 1;z = 0; #5 x = 1;y = 1;z = 1; #5;

$finish; end endmodule

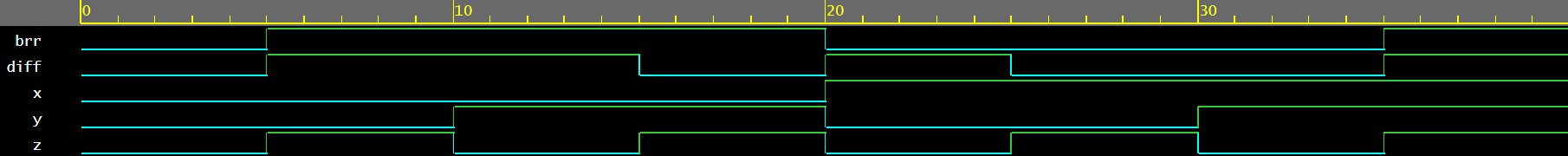
**Verilog code for full subtractor:**

module fullsub(x,y,z,diff,brr); input x,y,z;

output diff,brr;

assign diff = ((!x)&&(!y)&&z)||((!x)&&y&&(!z))||(x&&(!y)&&(!z))||(x&&y&&z); assign brr=((!x)&&y)||((!x)&&z)||(y&&z);

endmodule

**EpWave for full subtractor:**

# **[Results verified by truth table]**

1. **Design Verilog HDL to implement Decimal adder.**

**Testbench code for Decimal adder:**

module testbenchdadder;

// Inputs

reg [3:0] a;

reg [3:0] b; reg carry\_in;

// Outputs

wire [3:0] sum; wire carry;

// Instantiate the Unit Under Test (UUT)

decimaladder uut (.a(a),.b(b),.carry\_in(carry\_in),.sum(sum), .carry(carry)); initial begin

$dumpfile("dump.vcd");

$dumpvars(1); end

initial begin

// Apply Inputs

a = 0; b = 0; carry\_in = 0;

#100 a = 6; b = 9; carry\_in = 0;

#100 a = 3; b = 3; carry\_in = 1;

#100 a = 4; b = 5; carry\_in = 0;

#100 a = 8; b = 2; carry\_in = 0;

#100 a = 9; b = 9; carry\_in = 1;

#100;

$finish; end endmodule

**Verilog code for Decimal adder:**

module decimaladder(a,b,carry\_in,sum,carry);

//declare the inputs and outputs of the module with their sizes. input [3:0] a,b;

input carry\_in; output [3:0] sum; output carry;

//Internal variables reg [4:0] sum\_temp;

reg [3:0] sum; reg carry;

//always block for doing the addition always @(a,b,carry\_in)

begin

sum\_temp = a+b+carry\_in; //add all the inputs if(sum\_temp>9)

begin

sum\_temp = sum\_temp+6; //add 6, if result is more than 9. carry = 1; //set the carry output

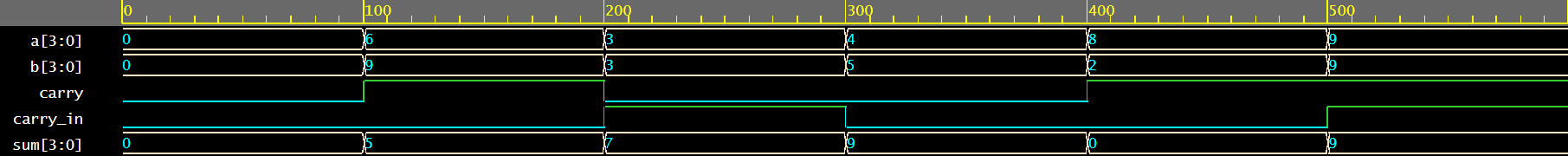
sum = sum\_temp[3:0]; end

else

begin carry = 0;

sum = sum\_temp[3:0]; end

end endmodule

**EpWave for Decimal adder:**

# **[Results verified by truth table]**

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

**Testbench code for 2:1 Multiplexer:**

module mux\_2to1\_tb\_behavioral; reg S,A,B;

wire Y;

mux\_2to1\_behavioral uut (.S(S),.A(A),.B(B),.Y(Y)); initial begin

$dumpfile("dump.vcd");

$dumpvars(1); end

initial begin

$monitor("S=%b, A=%b, B=%b, Y=%b", S, A, B, Y);

// Test case 1: S=0, A=1, B=0 S = 0;

A = 1;

B = 0;

#10;

// Test case 2: S=1, A=0, B=1 S = 1;

A = 0;

B = 1;

#10;

// Test case 3: S=0, A=0, B=1 S = 0;

A = 0;

B = 1;

#10;

// Test case 4: S=1, A=1, B=0 S = 1;

A = 1;

B = 0;

#10;

$finish; end endmodule

**Verilog code for 2:1 Multiplexer:** module mux\_2to1\_behavioral (Y,A,B,S); output Y;

input A,B,S; reg Y;

always @(A,B,S) begin

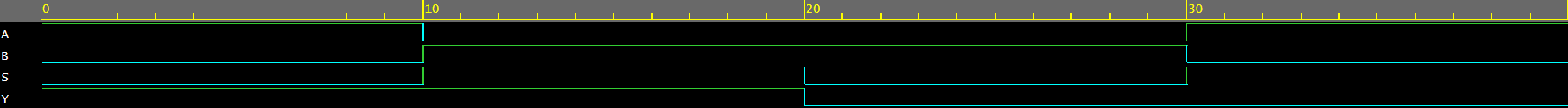
if (S == 0) Y = A;

else

Y = B;

end endmodule

**EpWave for 2:1 Multiplexer:**



**Testbench code for 4:1 Multiplexer:**

module mux\_4to1\_tb\_behavioral; reg in0, in1, in2, in3;

reg [1:0] s; wire out;

mux4to1 uut(.out(out),.in0(in0),.in1(in1),.in2(in2), .in3(in3), .s(s)); initial begin

$dumpfile("dump.vcd");

$dumpvars(1); end

initial begin

// Test case 1: s[1:0] = 2'b00;

in0=1; in1=0; in2=1;

in3=0; #10;

// Test case 2:

s[1:0] = 2'b01;

in0=1; in1=0; in2=1; in3=0;

#10;

// Test case 3:

s[1:0] = 2'b10;

in0=1; in1=0; in2=1; in3=0;

#10;

// Test case 4:

s[1:0] = 2'b11;

in0=1; in1=0; in2=1; in3=0;

#10;

$finish; end

endmodule

**Verilog code for 4:1 Multiplexer:**

module mux4to1 (out, in0, in1, in2, in3, s); output out;

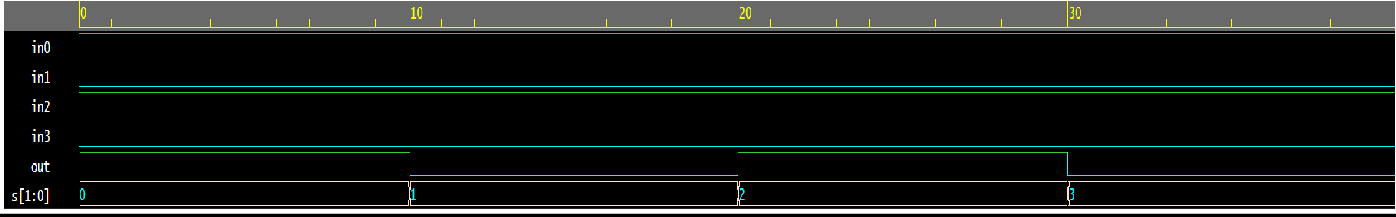
input in0, in1, in2, in3; input [1:0] s;

reg out;

always @(in0,in1,in2,in3,s) case(s)

2'b00: out = in0; 2'b01: out = in1;

2'b10: out = in2; 2'b11: out = in3; endcase endmodule

**EpWave for 4:1 Multiplexer:**

**Testbench code for 8:1 Multiplexer:**

module mux\_4to1\_tb\_behavioral;

reg in0, in1, in2, in3, in4, in5, in6, in7; reg [2:0] s;

wire out;

mux4to1 uut(.out(out),.in0(in0),.in1(in1),.in2(in2), .in3(in3),.in4(in4),.in5(in5),

.in6(in6),.in7(in7),.s(s)); initial begin

$dumpfile("dump.vcd");

$dumpvars(1); end

initial begin

// Test case 1: s[2:0] = 3'b000;

in0=1; in1=0; in2=1; in3=0; in4=1; in5=1; in6=0; in7=0;

#10;

// Test case 2: s[2:0] = 3'b001;

in0=1; in1=0;

in2=1; in3=0; in4=1; in5=1; in6=0; in7=0;

#10;

// Test case 3: s[2:0] = 3'b010;

in0=0; in1=1; in2=1; in3=0; in4=1; in5=1; in6=0; in7=0;

#10;

// Test case 4: s[2:0] = 3'b011;

in0=1; in1=0; in2=1; in3=0; in4=1; in5=1; in6=0; in7=0;

#10;

// Test case 5: s[2:0] = 3'b100;

in0=1; in1=0; in2=1; in3=0; in4=1; in5=1; in6=0; in7=0;

#10;

// Test case 6: s[2:0] = 3'b101;

in0=1; in1=0; in2=1; in3=0; in4=1; in5=1; in6=0; in7=0;

#10;

// Test case 7: s[2:0] = 3'b110;

in0=1; in1=0; in2=1; in3=0; in4=1; in5=1; in6=0; in7=0;

#10;

// Test case 8: s[2:0] = 3'b111;

in0=1; in1=0; in2=1; in3=0; in4=1; in5=1; in6=0; in7=0;

#10;

$finish; end endmodule

**Verilog code for 8:1 Multiplexer:**

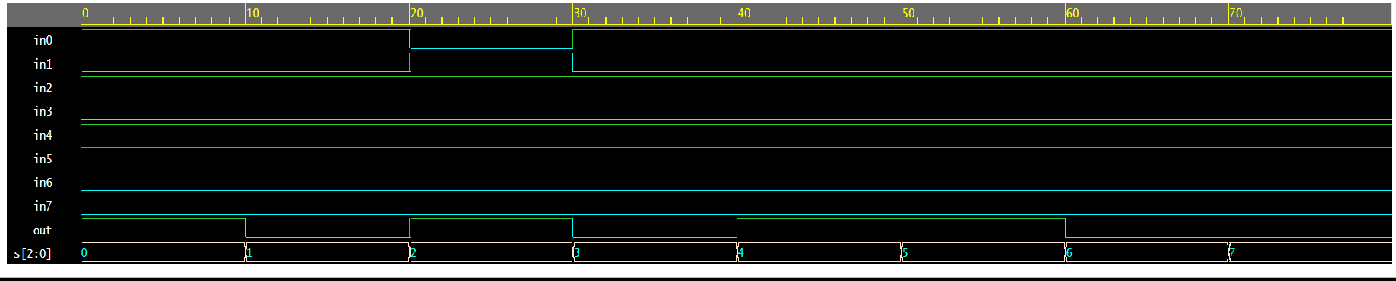
module mux4to1 (out, in0, in1, in2, in3, in4, in5, in6, in7, s); output out;

input in0, in1, in2, in3, in4, in5, in6, in7; input [2:0] s;

reg out;

always @(in0,in1, in2, in3, in4, in5, in6, in7, s) case(s)

3'b000: out = in0; 3'b001: out = in1; 3'b010: out = in2; 3'b011: out = in3; 3'b100: out = in4; 3'b101: out = in5; 3'b110: out = in6; 3'b111: out = in7; endcase endmodule

**EpWave for 8:1 Multiplexer:**

# **[Results verified by truth table]**

1. **Design Verilog program to implement types of De-Multiplexer.**

**Testbench code for 1:8 Demux:**

module test\_demux; reg [2:0]s;

reg a;

wire [7:0]y;

demux\_1\_8 mydemux(.y(y),.a(a),.s(s)); initial begin

$dumpfile("dump.vcd");

$dumpvars(1); end

initial begin a=1;

s[2:0] = 3'b000; #30;

a=0;

s[2:0] = 3'b001; #30;

a=1;

s[2:0] = 3'b010; #30;

a=0;

s[2:0] = 3'b011; #30;

a=0;

s[2:0] = 3'b100; #30;

a=1;

s[2:0] = 3'b101; #30;

a=0;

s[2:0] = 3'b110; #30;

a=1;

s[2:0] = 3'b111; #30;

$finish; end endmodule

**Verilog code for 1:8 Demux:** module demux\_1\_8(y,s,a); output reg [7:0]y;

input [2:0]s; input a;

always @(a,s) begin

y=0;

case(s)

3'b000: y[0]=a;

3'b001: y[1]=a;

3'b010: y[2]=a;

3'b011: y[3]=a;

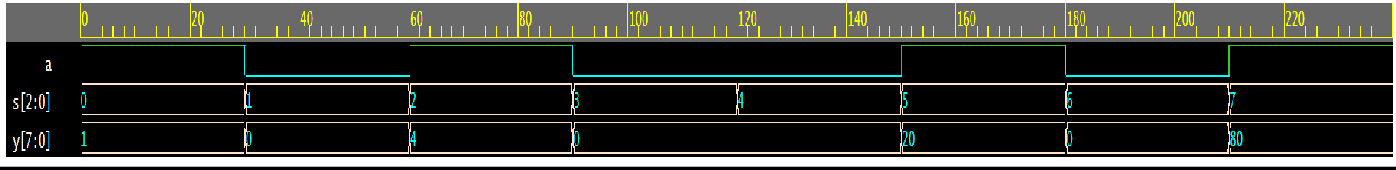
3'b100: y[4]=a;

3'b101: y[5]=a;

3'b110: y[6]=a;

3'b111: y[7]=a;

endcase end endmodule

**EpWave for 1:8 Demux:**

# **[Results verified by truth table]**

1. **Design Verilog program for implementing various types of Flip-Flops such as SR, JK and D.**

**Testbench code for SR FLIP FLOP:**

module ssff\_test(); reg s,r,clk;

wire q,qbar;

srff uut(.s(s),.r(r),.clk(clk),.q(q),.qbar(qbar)); initial begin

$dumpfile("dump.vcd");

$dumpvars(1); end

initial begin clk=0;

forever #10 clk = ~clk; end

initial begin

s= 1;r= 0;

#100; s= 0; r= 1;

#100; s= 0; r= 0;

#100; s= 1; r=1;

#100;$finish; end endmodule

**Verilog code for SR FLIP FLOP:**

module srff(s,r,clk,q,qbar); input s,r,clk;

output reg q,qbar; always@(posedge clk) begin

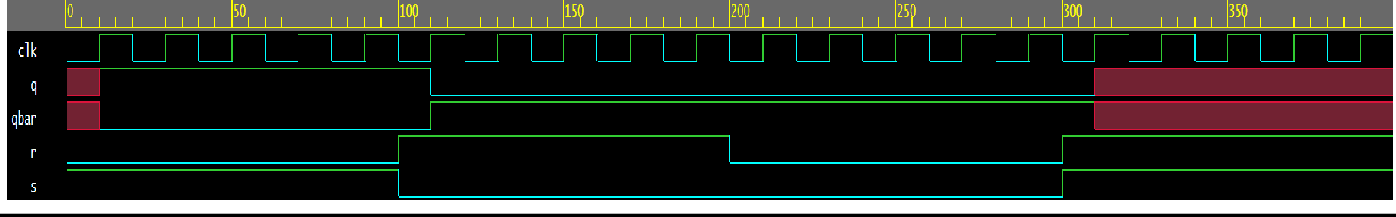
case({s,r})

2'b00:q<= q; // No change 2'b01:q<= 1'b0; // reset 2'b10:q<= 1'b1; // set

2'b11:q<= 1'bx; // Invalid inputs endcase

end

assign qbar = ~q; endmodule

**EpWave for SR FLIP FLOP:**

**Testbench code for JK FLIP FLOP:**

module jk\_ff\_test\_bench(); reg clk,j,k;

wire Q,Q\_bar;

jk\_flipflop uut(.j(j),.k(k),.clk(clk),.Q(Q),.Q\_bar(Q\_bar)); initial begin

$dumpfile("dump.vcd");

$dumpvars(1); end

initial begin clk=0;

forever #5 clk=~clk; end

initial begin

j = 1'b0; k = 1'b0;

#10;

j = 1'b0; k = 1'b1;

#10;

j = 1'b1; k = 1'b0;

#10;

j = 1'b1; k = 1'b1;

#10;

$finish; end endmodule

**Verilog code for JK FLIP FLOP:**

module jk\_flipflop(j,k,clk,Q,Q\_bar); input j,k,clk;

output reg Q,Q\_bar; always@(posedge clk) begin

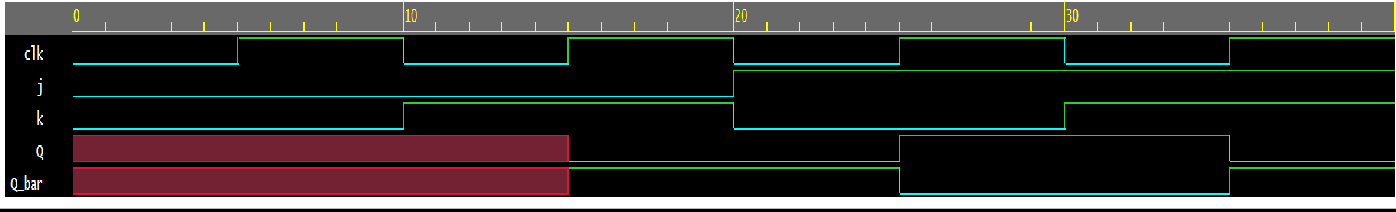
case({j, k}) 2'b00:{Q,Q\_bar}<={Q,Q\_bar};

2'b01:{Q,Q\_bar}<={1'b0,1'b1};

2'b10:{Q,Q\_bar}<={1'b1,1'b0};

2'b11:{Q,Q\_bar}<={~Q,Q};

endcase end endmodule

**EpWave for JK FLIP FLOP:**

**Testbench code for D FLIP FLOP:**

module tb\_DFF(); reg D;

reg clk; reg reset; wire Q;

DFlipFlop dut(.D(D),.clk(clk),.reset(reset),.Q(Q)); initial begin

$dumpfile("dump.vcd");

$dumpvars(1); end

initial begin clk=0;

forever #10 clk = ~clk; end

initial begin reset=1; D<= 0;

#100;

reset=0; D<= 1;

#100;

D<= 0;

#100;

D<= 1;

#100; $finish; end endmodule

**Verilog code for D FLIP FLOP:** module DFlipFlop(D,clk,reset,Q); input D; // Data input

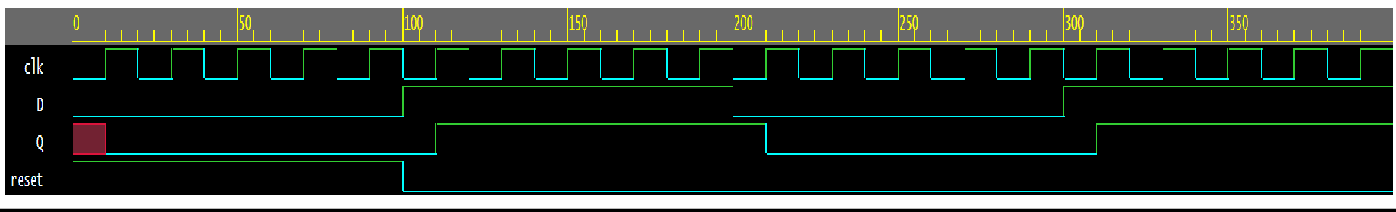
input clk; // clock input input reset; // reset output reg Q; // output Q always @(posedge clk) begin

if(reset==1'b1) Q<= 1'b0;

else

Q<= D;

end endmodule

**EpWave for D FLIP FLOP:**

# **[Results verified by truth table]**