ASSIGNMENT-2

Digital System Design

Name: - Nirbhay Thakkar

ID: - 21EL092

Course Code: - 3EL41

Q1) Design 4-bit Ripple Carry Adder with the help of 1-bit adder.

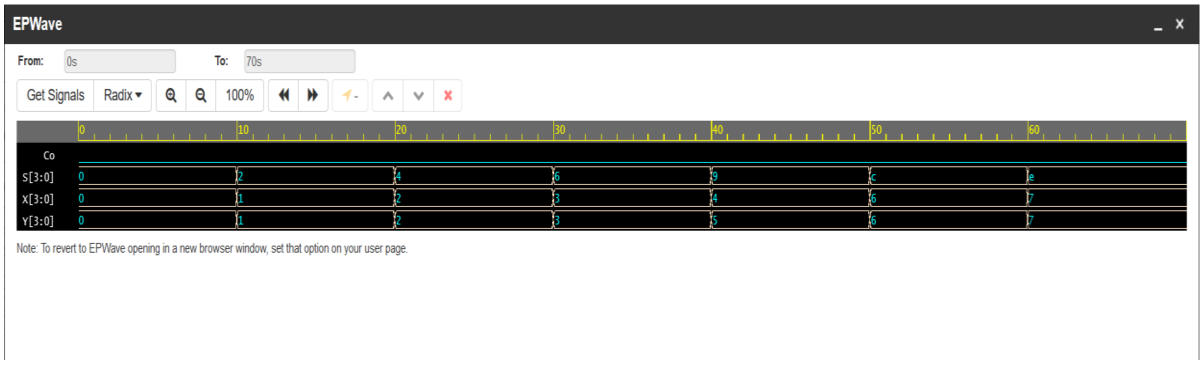
Design:-

```
module ripple_adder(X, Y, S, Co);
 input [3:0] X, Y;
 output [3:0] S;
 output Co;
 wire w1, w2, w3;

 fulladder u1(X[0], Y[0], 1'b0, S[0], w1);
 fulladder u2(X[1], Y[1], w1, S[1], w2);
 fulladder u3(X[2], Y[2], w2, S[2], w3);
 fulladder u4(X[3], Y[3], w3, S[3], Co);
endmodule


module fulladder(X, Y, Ci, S, Co);
  input X, Y, Ci;
  output S, Co;
  wire w1,w2,w3;

  xor G1(w1, X, Y);
  xor G2(S, w1, Ci);
  and G3(w2, w1, Ci);
  and G4(w3, X, Y);
  or G5(Co, w2, w3);

endmodule
```

Testbench :-

```verilog
module testbench_ripple_adder;
  reg [3:0] X, Y;
  wire [3:0] S;
  wire Co;


  ripple_adder uut (
    .X(X),
    .Y(Y),
    .S(S),
    .Co(Co)
  );

  initial begin

    $dumpfile("dump.vcd");
    $dumpvars;

    X = 4'b0000;
    Y = 4'b0000;
    #10

    X = 4'b0001;
    Y = 4'b0001;
    #10

    X = 4'b0010;
    Y = 4'b0010;
    #10

    X = 4'b0011;
    Y = 4'b0011;
    #10

    X = 4'b0100;
    Y = 4'b0101;
    #10

    X = 4'b0110;
    Y = 4'b0110;
    #10

    X = 4'b0111;
    Y = 4'b0111;
    #10
  end
endmodule
```

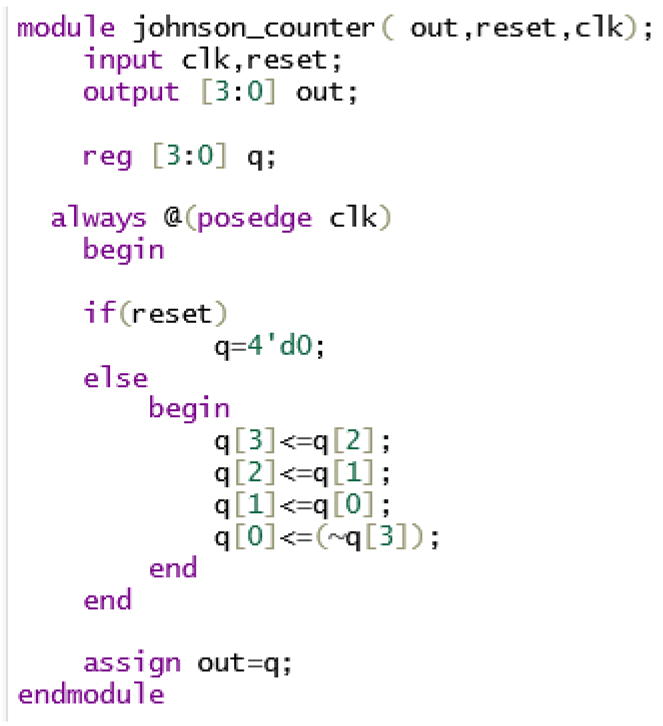
Output :-



Note: To revert to EPWave opening in a new browser window, set that option on your user page.

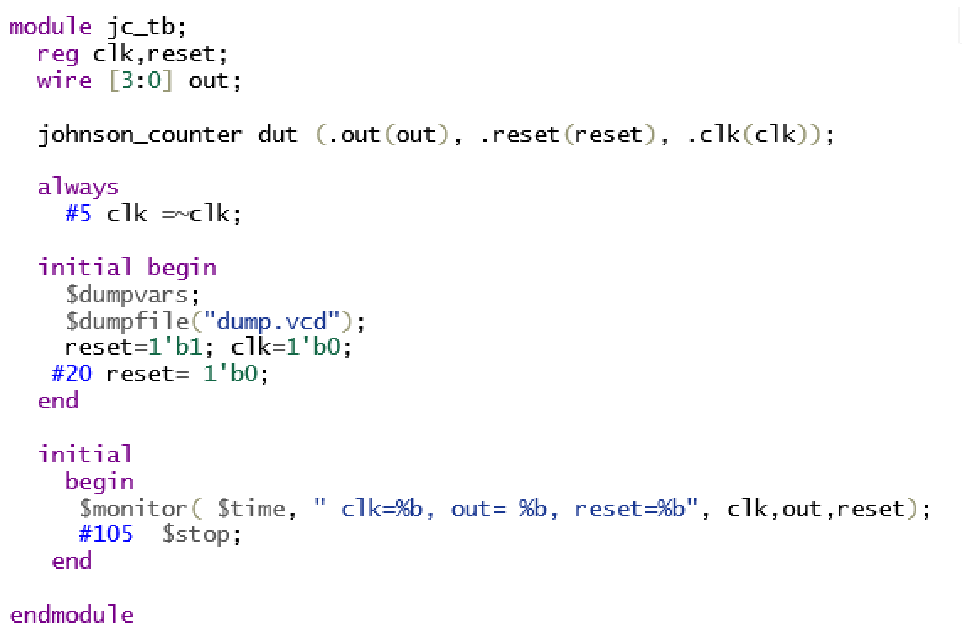
Q2) Design D-flipflop and reuse it to implement 4- bit Johnson Counter.

Design :-



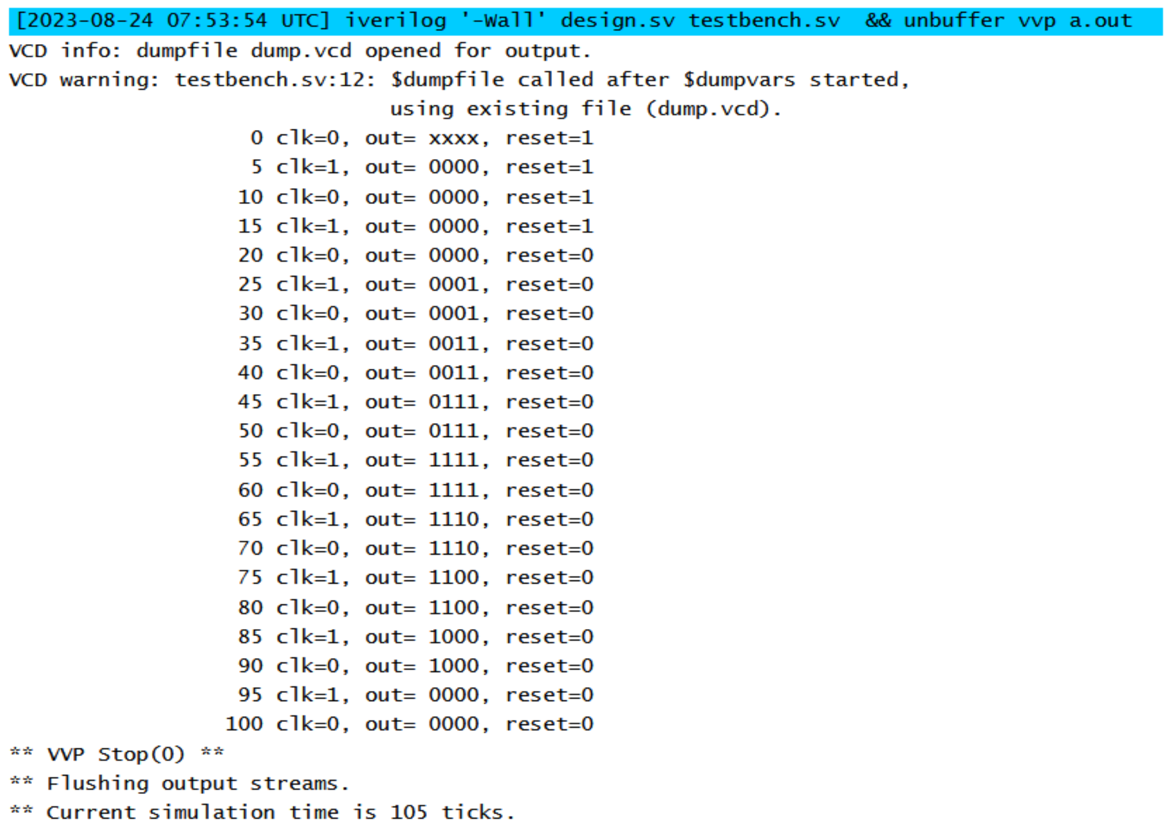
```
module johnson_counter( out,reset,clk);
    input clk,reset;
    output [3:0] out;

    reg [3:0] q;

  always @(posedge clk)
    begin

    if(reset)
            q=4'd0;
    else
        begin
            q[3]<=q[2];
            q[2]<=q[1];
            q[1]<=q[0];
            q[0]<=(~q[3]);
        end
    end

    assign out=q;
endmodule
```

Testbench:-



```verilog
module jc_tb;
  reg clk,reset;
  wire [3:0] out;

  johnson_counter dut (.out(out), .reset(reset), .clk(clk));

  always
    #5 clk =~clk;

  initial begin
    $dumpvars;
    $dumpfile("dump.vcd");
    reset=1'b1; clk=1'b0;
   #20 reset= 1'b0;
  end

  initial
    begin
     $monitor( $time, " clk=%b, out= %b, reset=%b", clk,out,reset);
     #105  $stop;
   end

endmodule
```

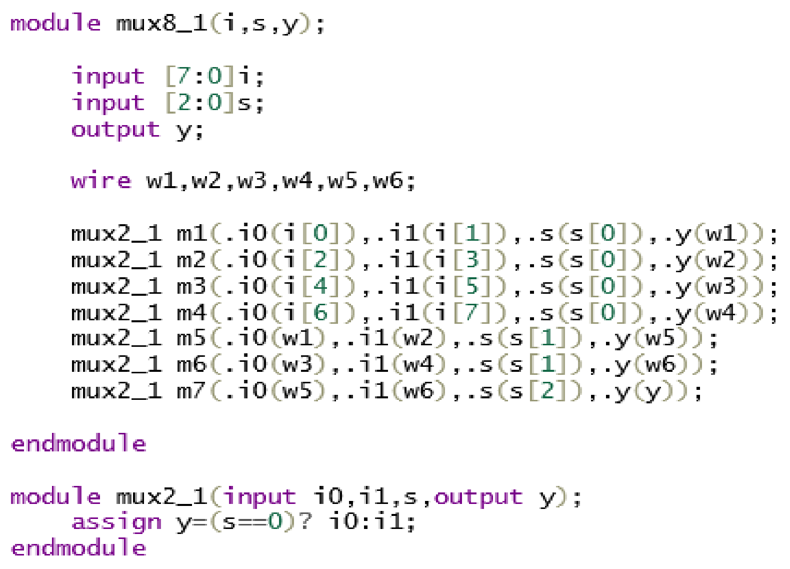
Output :-



```
[2023-08-24 07:53:54 UTC] iverilog '-Wall' design.sv testbench.sv  && unbuffer vvp a.out
VCD info: dumpfile dump.vcd opened for output.
VCD warning: testbench.sv:12: $dumpfile called after $dumpvars started,
                              using existing file (dump.vcd).
                   0 clk=0, out= xxxx, reset=1
                   5 clk=1, out= 0000, reset=1
                  10 clk=0, out= 0000, reset=1
                  15 clk=1, out= 0000, reset=1
                  20 clk=0, out= 0000, reset=0
                  25 clk=1, out= 0001, reset=0
                  30 clk=0, out= 0001, reset=0
                  35 clk=1, out= 0011, reset=0
                  40 clk=0, out= 0011, reset=0
                  45 clk=1, out= 0111, reset=0
                  50 clk=0, out= 0111, reset=0
                  55 clk=1, out= 1111, reset=0
                  60 clk=0, out= 1111, reset=0
                  65 clk=1, out= 1110, reset=0
                  70 clk=0, out= 1110, reset=0
                  75 clk=1, out= 1100, reset=0
                  80 clk=0, out= 1100, reset=0
                  85 clk=1, out= 1000, reset=0
                  90 clk=0, out= 1000, reset=0
                  95 clk=1, out= 0000, reset=0
                 100 clk=0, out= 0000, reset=0
** VVP Stop(0) **
** Flushing output streams.
** Current simulation time is 105 ticks.
```

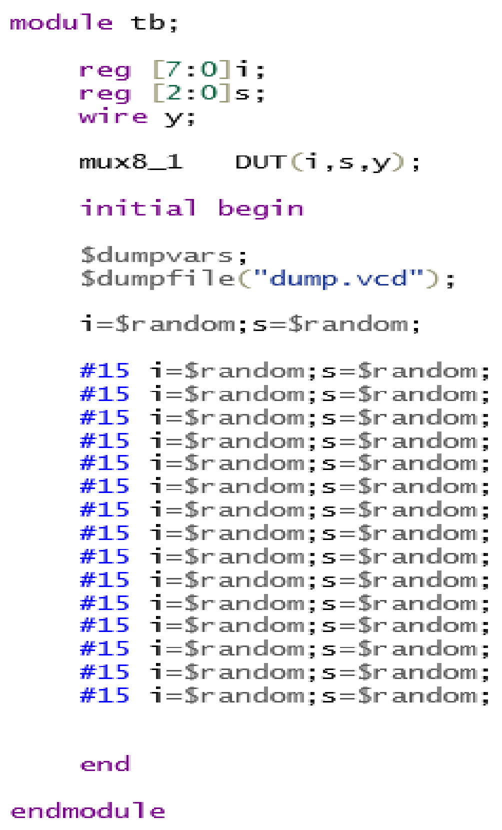
Q3) Reuse 2:1 Mux code to implement 8:1 Mux.

Design: -



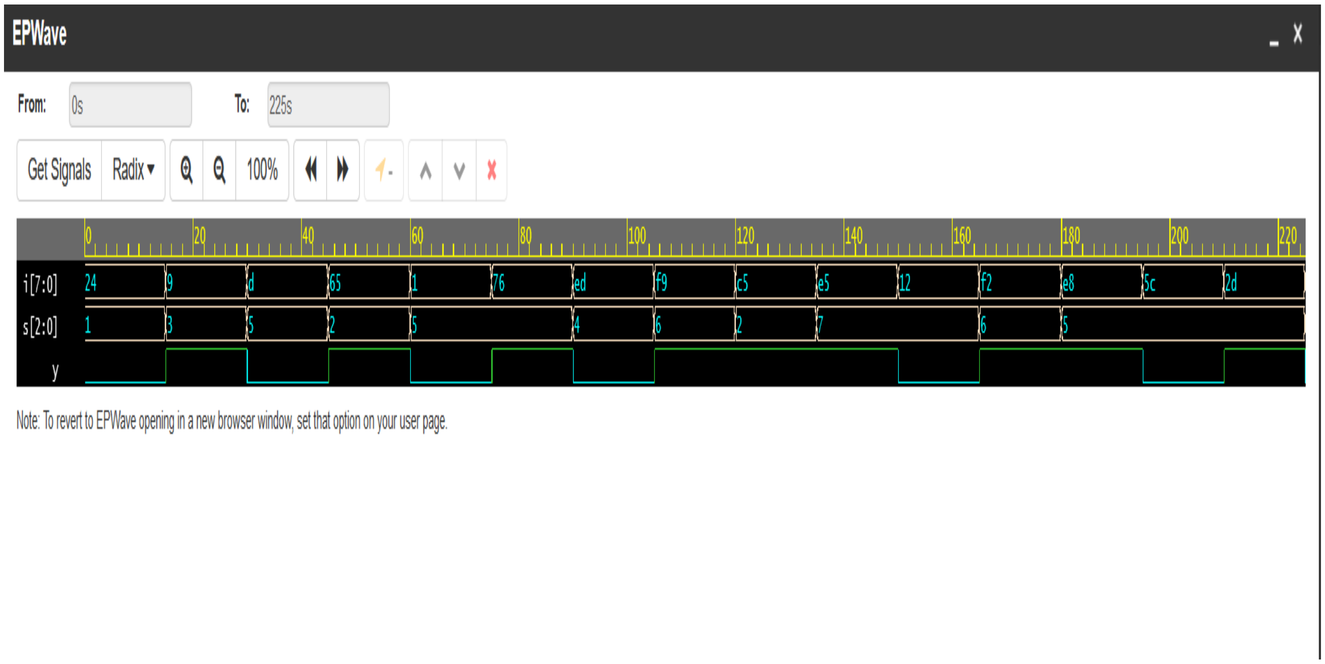
```
module mux8_1(i,s,y);

    input [7:0]i;
    input [2:0]s;
    output y;

    wire w1,w2,w3,w4,w5,w6;

    mux2_1 m1(.i0(i[0]),.i1(i[1]),.s(s[0]),.y(w1));
    mux2_1 m2(.i0(i[2]),.i1(i[3]),.s(s[0]),.y(w2));
    mux2_1 m3(.i0(i[4]),.i1(i[5]),.s(s[0]),.y(w3));
    mux2_1 m4(.i0(i[6]),.i1(i[7]),.s(s[0]),.y(w4));
    mux2_1 m5(.i0(w1),.i1(w2),.s(s[1]),.y(w5));
    mux2_1 m6(.i0(w3),.i1(w4),.s(s[1]),.y(w6));
    mux2_1 m7(.i0(w5),.i1(w6),.s(s[2]),.y(y));

endmodule

module mux2_1(input i0,i1,s,output y);
    assign y=(s==0)? i0:i1;
endmodule
```

Testbench: -



```
module tb;

    reg [7:0]i;
    reg [2:0]s;
    wire y;

    mux8_1   DUT(i,s,y);

    initial begin

    $dumpvars;
    $dumpfile("dump.vcd");

    i=$random;s=$random;

    #15 i=$random;s=$random;
    #15 i=$random;s=$random;
    #15 i=$random;s=$random;
    #15 i=$random;s=$random;
    #15 i=$random;s=$random;
    #15 i=$random;s=$random;
    #15 i=$random;s=$random;
    #15 i=$random;s=$random;
    #15 i=$random;s=$random;
    #15 i=$random;s=$random;
    #15 i=$random;s=$random;
    #15 i=$random;s=$random;
    #15 i=$random;s=$random;
    #15 i=$random;s=$random;
    #15 i=$random;s=$random;


    end

endmodule
```

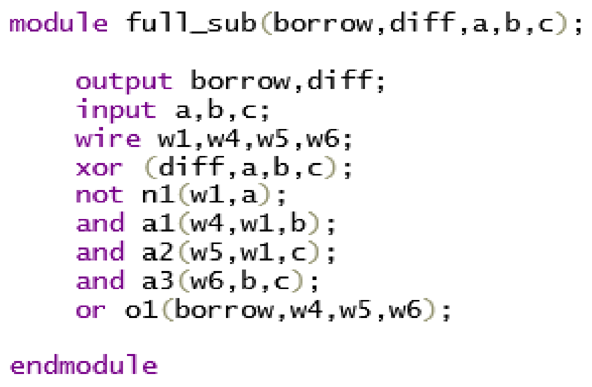
Output: -



Note: To revert to EPWave opening in a new browser window, set that option on your user page.

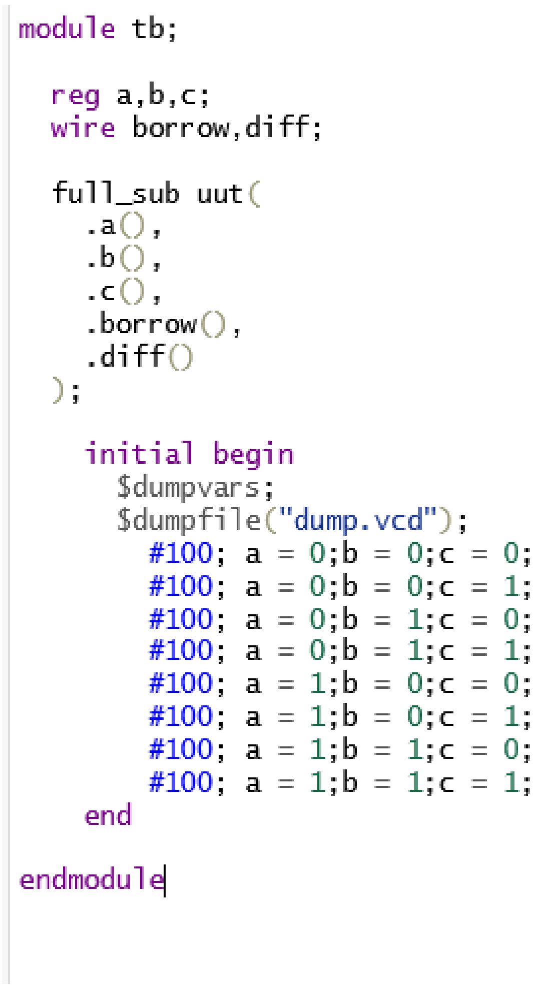
4) Design a Full Subtractor with Gate Level Modeling Style. (use primitive gates)

Design: -



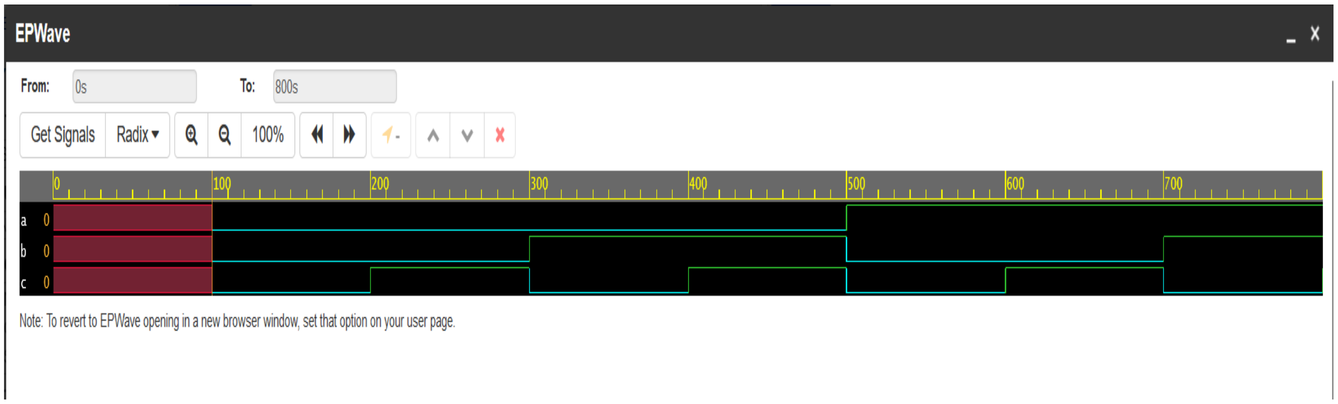
```
module full_sub(borrow,diff,a,b,c);

    output borrow,diff;
    input a,b,c;
    wire w1,w4,w5,w6;
    xor (diff,a,b,c);
    not n1(w1,a);
    and a1(w4,w1,b);
    and a2(w5,w1,c);
    and a3(w6,b,c);
    or o1(borrow,w4,w5,w6);

endmodule
```

Testbench: -



```
module tb;

  reg a,b,c;
  wire borrow,diff;

  full_sub uut(
    .a(),
    .b(),
    .c(),
    .borrow(),
    .diff()
  );

    initial begin
      $dumpvars;
      $dumpfile("dump.vcd");
        #100; a = 0;b = 0;c = 0;
        #100; a = 0;b = 0;c = 1;
        #100; a = 0;b = 1;c = 0;
        #100; a = 0;b = 1;c = 1;
        #100; a = 1;b = 0;c = 0;
        #100; a = 1;b = 0;c = 1;
        #100; a = 1;b = 1;c = 0;
        #100; a = 1;b = 1;c = 1;
    end

endmodule
```

Output: -



Note: To revert to EPWave opening in a new browser window, set that option on your user page.

Q5) Design 2x1 Decoder using gate level modeling

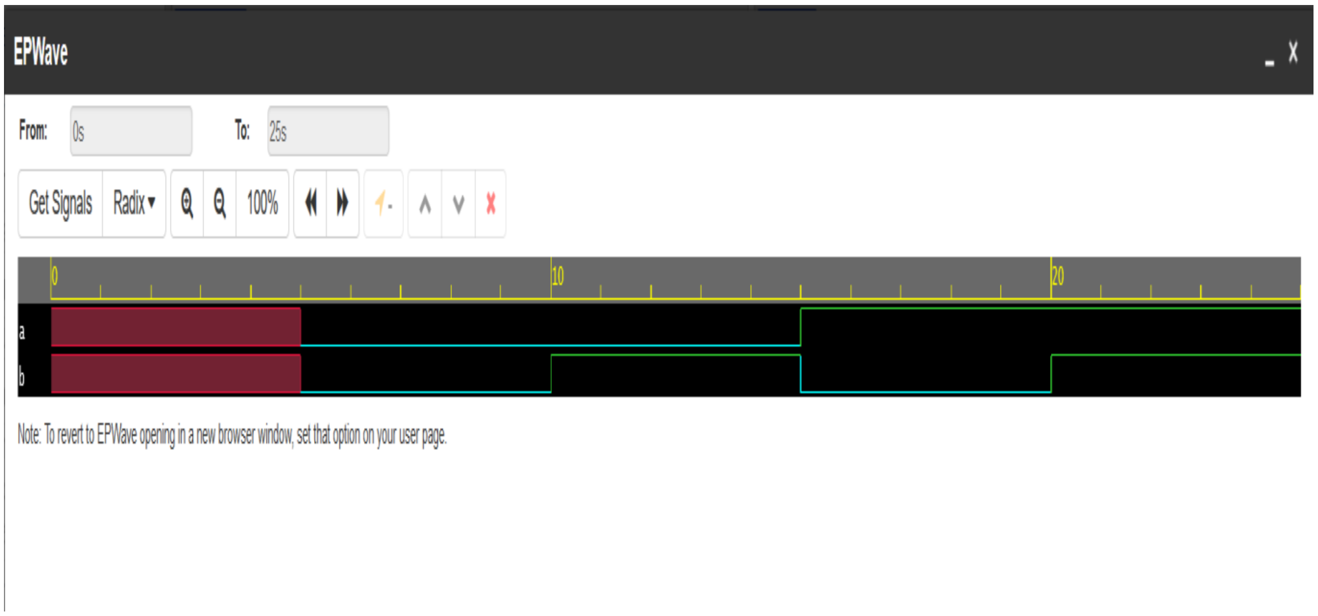
Design: -

```
module decoder24_gate(en,a,b,y);
    input en,a,b;
    output [3:0]y;
    wire enb,na,nb;
    not n0(enb,en);
    not n1(na,a);
    not n2(nb,b);

    nand n3(y[0],enb,na,nb);
    nand n4(y[1],enb,na,b);
    nand n5(y[2],enb,a,nb);
    nand n6(y[3],enb,a,b);

endmodule
```

Testbench: -

```
module tb;

  reg a,b,c;
  wire borrow,diff;

  full_sub uut(
    .a(),
    .b(),
    .c(),
    .borrow(),
    .diff()
  );

    initial begin
      $dumpvars;
      $dumpfile("dump.vcd");
        #100; a = 0;b = 0;c = 0;
        #100; a = 0;b = 0;c = 1;
        #100; a = 0;b = 1;c = 0;
        #100; a = 0;b = 1;c = 1;
        #100; a = 1;b = 0;c = 0;
        #100; a = 1;b = 0;c = 1;
        #100; a = 1;b = 1;c = 0;
        #100; a = 1;b = 1;c = 1;
    end

endmodule
```

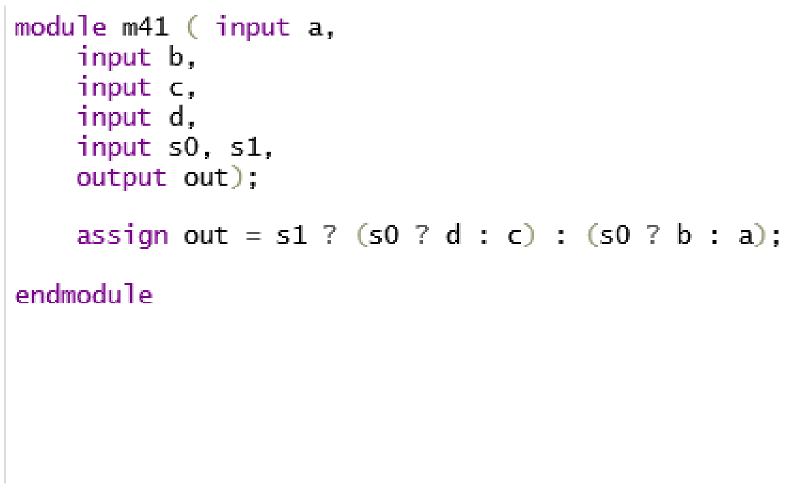
Output: -



Note: To revert to EPWave opening in a new browser window, set that option on your user page.

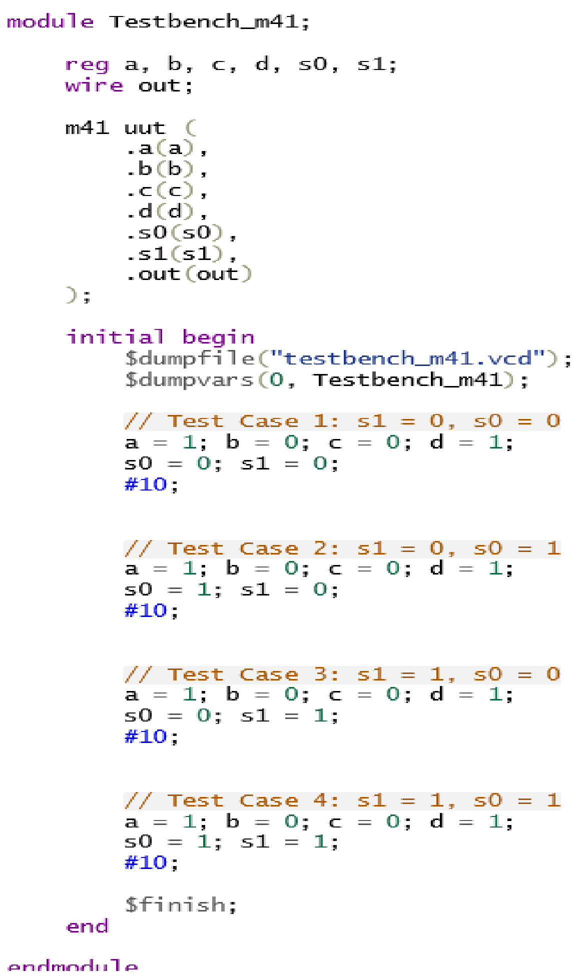
Q6) Design a 4x1 mux using operators. (use data flow)

Design: -



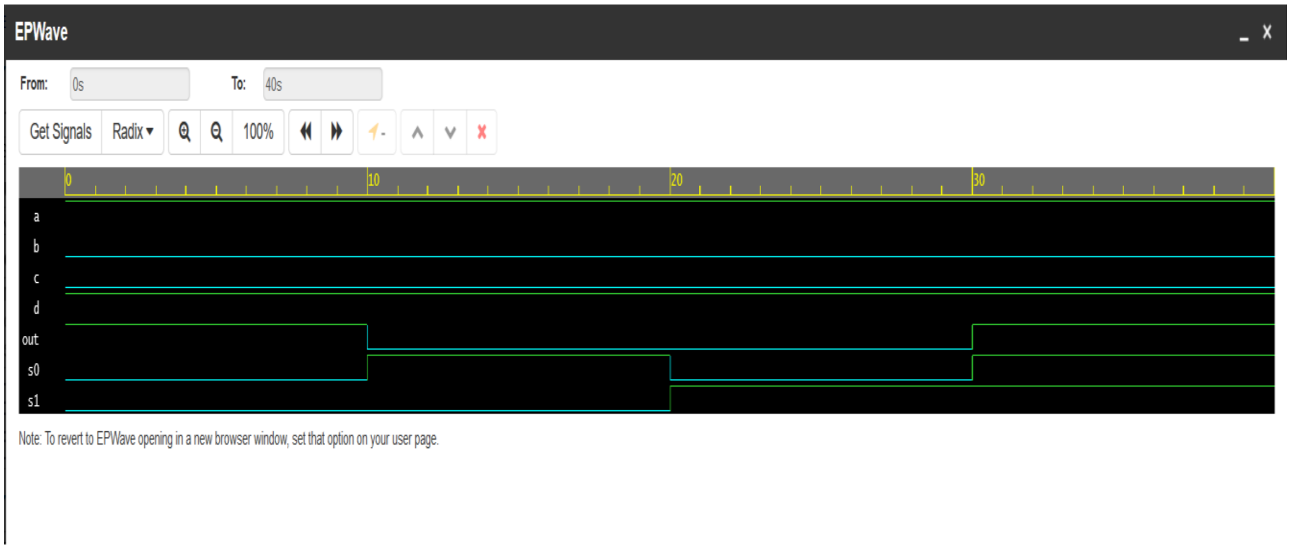
```
module m41 ( input a,
     input b,
     input c,
     input d,
     input s0, s1,
     output out);

     assign out = s1 ? (s0 ? d : c) : (s0 ? b : a);

endmodule
```

Testbench: -



```
module Testbench_m41;

    reg a, b, c, d, s0, s1;
    wire out;

    m41 uut (
        .a(a),
        .b(b),
        .c(c),
        .d(d),
        .s0(s0),
        .s1(s1),
        .out(out)
    );

    initial begin
        $dumpfile("testbench_m41.vcd");
        $dumpvars(0, Testbench_m41);

        // Test Case 1: s1 = 0, s0 = 0
        a = 1; b = 0; c = 0; d = 1;
        s0 = 0; s1 = 0;
        #10;


        // Test Case 2: s1 = 0, s0 = 1
        a = 1; b = 0; c = 0; d = 1;
        s0 = 1; s1 = 0;
        #10;


        // Test Case 3: s1 = 1, s0 = 0
        a = 1; b = 0; c = 0; d = 1;
        s0 = 0; s1 = 1;
        #10;


        // Test Case 4: s1 = 1, s0 = 1
        a = 1; b = 0; c = 0; d = 1;
        s0 = 1; s1 = 1;
        #10;

        $finish;
    end
endmodule
```

Output: -



Note: To revert to EPWave opening in a new browser window, set that option on your user page.

Q7) Design a Full adder using half adder

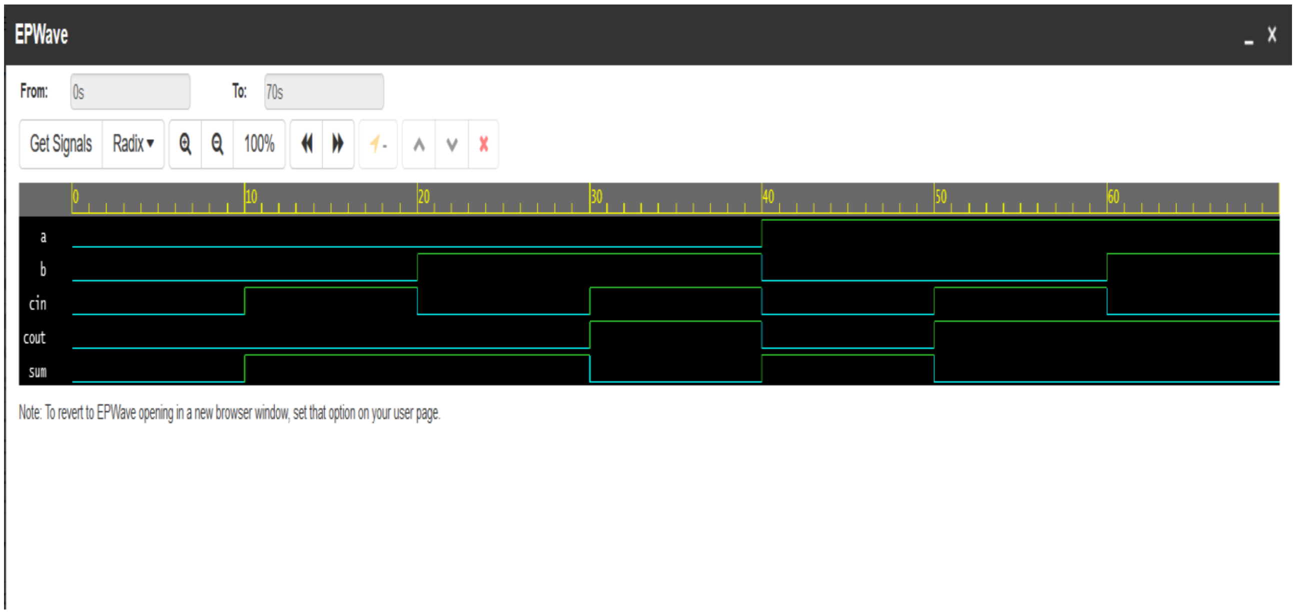
Design: -

```
module full_adder(
    input a,b,cin,
    output sum,carry
);

    wire c,c1,s;

    half_adder ha0(a,b,s,c);
    half_adder ha1(cin,s,sum,c1);

    assign carry = c | c1 ;

endmodule

module half_adder (
    input a,b,
    output sum,carry
);

    assign sum = a ^ b;
    assign carry = a & b;

endmodule
```

Testbench: -

```
module tb;

  reg a,b,cin;
  wire sum,cout;

  full_adder uut(a,b,cin,sum,cout);

  initial begin
    $dumpvars(1);
    $dumpfile("dump.vcd");

    a = 0; b = 0; cin = 0;
    #10
    a = 0; b = 0; cin = 1;
    #10
    a = 0; b = 1; cin = 0;
    #10
    a = 0; b = 1; cin = 1;
    #10
    a = 1; b = 0; cin = 0;
    #10
    a = 1; b = 0; cin = 1;
    #10
    a = 1; b = 1; cin = 0;
    #10
    $finish();
  end
endmodule
```

Output: -



Note: To revert to EPWave opening in a new browser window, set that option on your user page.