Name – SAI POOJITH (2025122010)

Practice Problems for Verilog

VLSI DESIGN

1

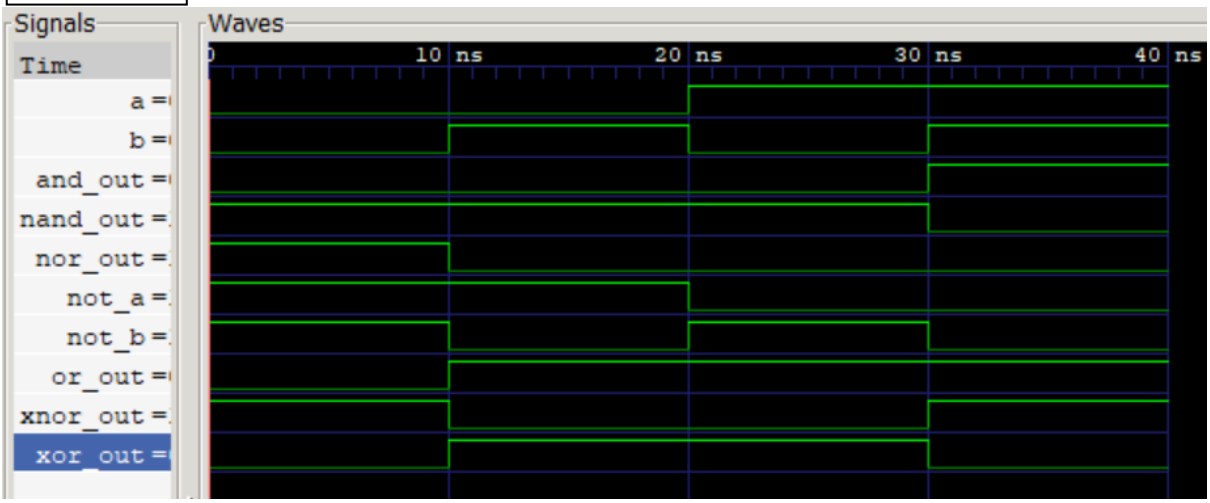
Verilog code

```
`timescale 1ns / 1ps

module combinational_gates(a, b, and_out, or_out, nand_out, nor_out, xor_out, xnor_out, not_a, not_b);
    input a, b;
    output and_out, or_out, nand_out, nor_out, xor_out, xnor_out, not_a, not_b;

    assign and_out  = a & b;
    assign or_out   = a | b;
    assign nand_out = ~(a & b);
    assign nor_out  = ~(a | b);
    assign xor_out  = a ^ b;
    assign xnor_out = ~(a ^ b);
    assign not_a    = ~a;
    assign not_b    = ~b;

endmodule
```

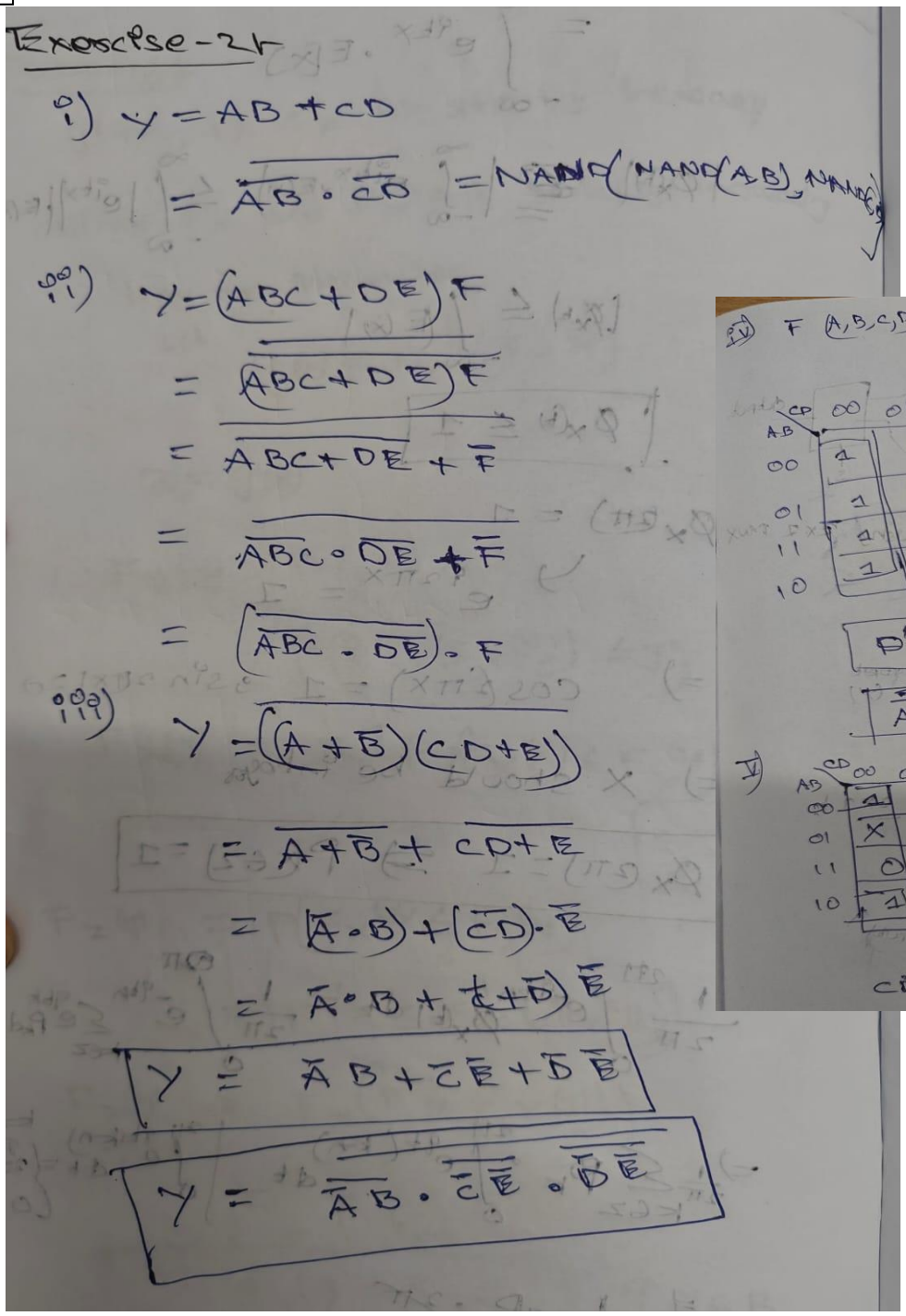
testbench

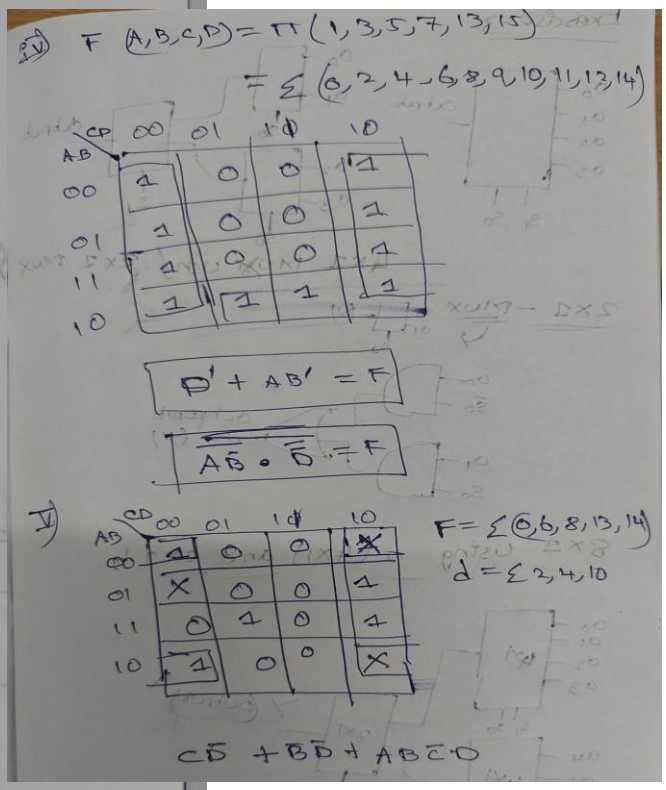
```
`timescale 1ns / 1ps

module combinational_gates_tb;
    reg a;
    reg b;
    wire and_out, or_out, nand_out, nor_out, xor_out, xnor_out, not_a, not_b;
    combinational_gates uut (
        .a(a),
        .b(b),
        .and_out(and_out),
        .or_out(or_out),
        .nand_out(nand_out),
        .nor_out(nor_out),
        .xor_out(xor_out),
        .xnor_out(xnor_out),
        .not_a(not_a),
        .not_b(not_b)
    );

    initial begin
        $dumpfile("combinational_gates.vcd");
        $dumpvars(0, combinational_gates_tb);

        $display("A B | AND OR NAND NOR XOR XNOR NOT_A NOT_B");
        $display("-----------------------------------------");

        a = 0; b = 0; #10;
        $display("%b %b |  %b   %b    %b    %b    %b    %b     %b     %b", a,b,and_out,or_out,nand_out,nor_out,xor_out,xnor_out,not_a,not_b);

        a = 0; b = 1; #10;
        $display("%b %b |  %b   %b    %b    %b    %b    %b     %b     %b", a,b,and_out,or_out,nand_out,nor_out,xor_out,xnor_out,not_a,not_b);

        a = 1; b = 0; #10;
        $display("%b %b |  %b   %b    %b    %b    %b    %b     %b     %b", a,b,and_out,or_out,nand_out,nor_out,xor_out,xnor_out,not_a,not_b);

        a = 1; b = 1; #10;
        $display("%b %b |  %b   %b    %b    %b    %b    %b     %b     %b", a,b,and_out,or_out,nand_out,nor_out,xor_out,xnor_out,not_a,not_b);
        $finish;
    end
endmodule
```

GTKWave




```
A B | AND OR NAND NOR XOR XNOR NOT_A NOT_B
-----------------------------------------
0 0 |  0   0    1    1    0    1     1     1
0 1 |  0   1    1    0    1    0     1     0
1 0 |  0   1    1    0    1    0     0     1
and_gate_tb.v:16: $finish called at 40000 (1ps)
1 1 |  1   1    0    0    0    1     0     0
```





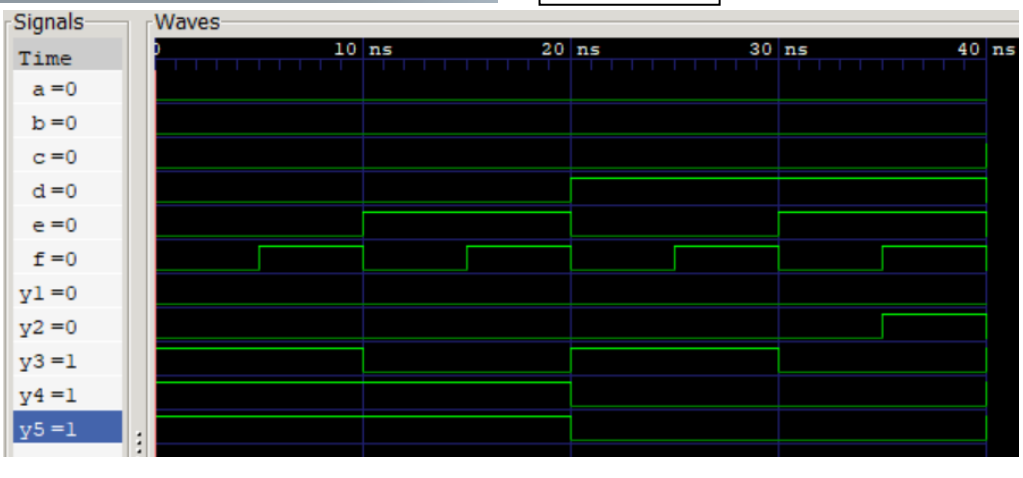
testbench

```
`timescale 1ns / 1ps
module compound_circuits_tb;
    reg a,b,c,d,e,f;
    wire y1,y2,y3,y4,y5;
    compound_circuits uut(a,b,c,d,e,f,y1,y2,y3,y4,y
5); initial begin
        $dumpfile("compound_circuits.vcd");
        $dumpvars(0, compound_circuits_tb);
        for (integer i=0; i<64; i=i+1) begin
            {a,b,c,d,e,f} = i;
            #5;
        end
        $finish;
    end
endmodule

```

Verilog code

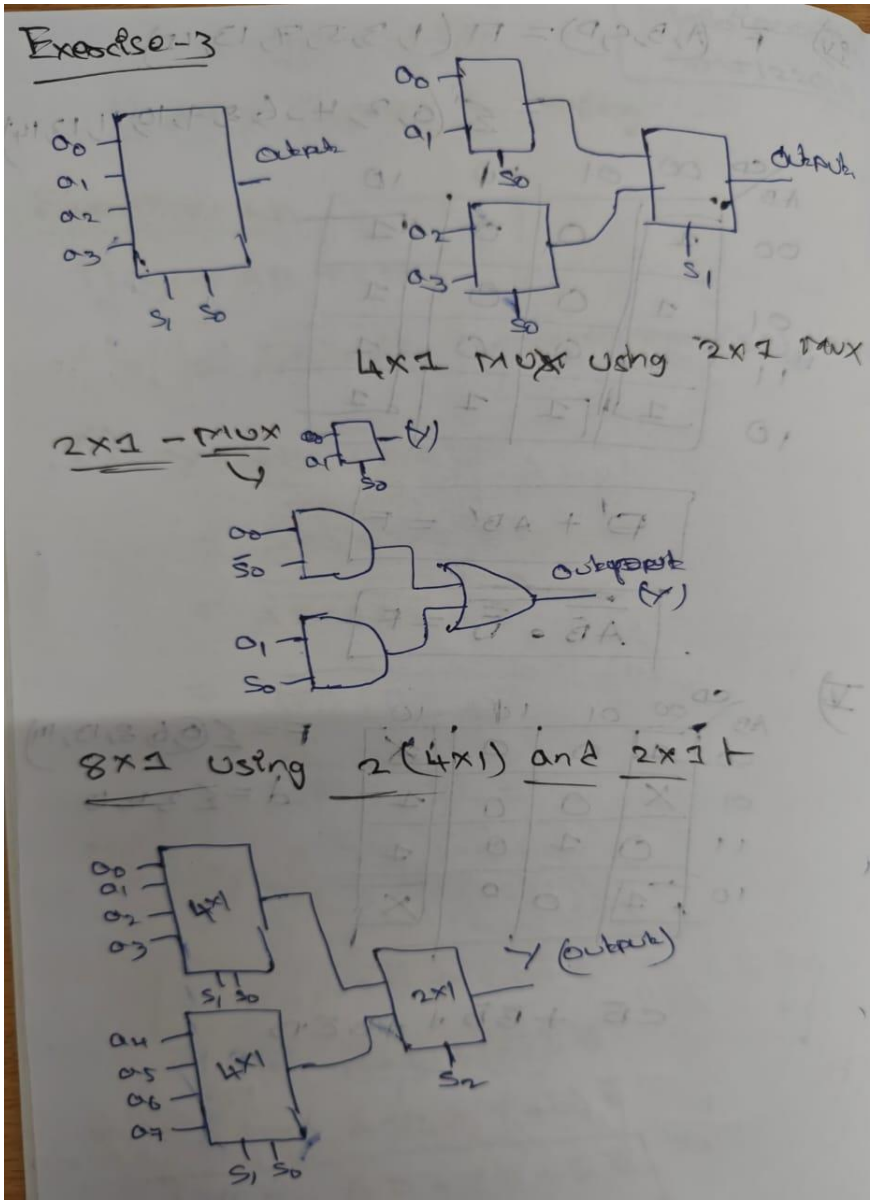
```
1  `timescale 1ns / 1ps
2  module compound_circuits(
3      input  a, b, c, d, e, f,
4      output y1, y2, y3, y4, y5);
5
6      wire na, nb, nc, nd, ne;
7      nand(na, a, a);
8      nand(nb, b, b);
9      nand(nc, c, c);
10     nand(nd, d, d);
11     nand(ne, e, e);
12
13     wire w1_ab_n, w1_cd_n;
14     nand(w1_ab_n, a, b);
15     nand(w1_cd_n, c, d);
16     nand(y1, w1_ab_n, w1_cd_n);
17
18     wire w2_abc_n, w2_de_n, w2_sum, w2_and_
19 n;  nand(w2_abc_n, a, b, c);
20     nand(w2_de_n, d, e);
21     nand(w2_sum, w2_abc_n, w2_de_n);
22     nand(w2_and_n, w2_sum, f);
23     nand(y2, w2_and_n, w2_and_n);
24
25     wire w3_t1, w3_t2, w3_t3;
26     nand(w3_t1, na, b);
27     nand(w3_t2, nc, ne);
28     nand(w3_t3, nd, ne);
29     nand(y3, w3_t1, w3_t2, w3_t3);
30
31     wire w4_abp_n;
32     nand(w4_abp_n, a, nb);
33     nand(y4, d, w4_abp_n);
34
35     wire w5_t1_n, w5_t2_n, w5_t3_n;
36     nand(w5_t1_n, c, nd);
37     nand(w5_t2_n, nb, nd);
38     nand(w5_t3_n, a, b, nc, d);
39     nand(y5, w5_t1_n, w5_t2_n, w5_t3_n);
40
41 endmodule
```

GTKWave



3

MUX



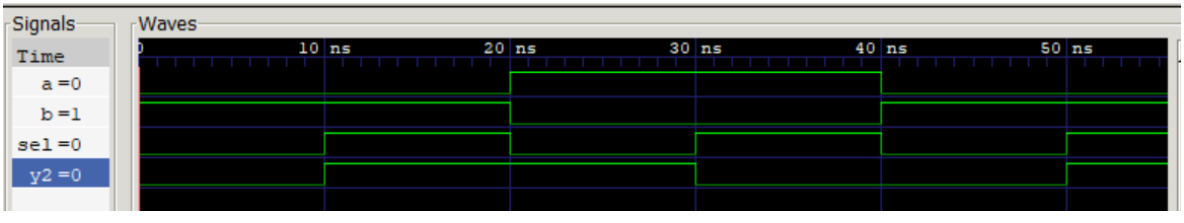
Verilog code

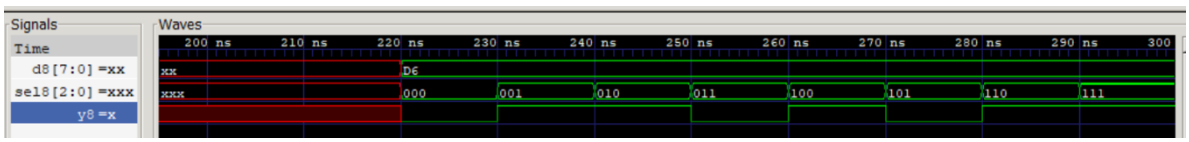
```
`timescale 1ns/1ps
module mux2_1 (
    input a,
    input b,
    input sel,
    output y
);
    wire nsel, and0, and1;

    not(nsel,sel);
    and(and0,a,nsel);
    and(and1,b,sel);
    or(y, and0, and1);
endmodule

module mux4_1 (
    input [3:0] d,
    input [1:0] sel,
    output y
);
    wire low, high;
    mux2_1 m0 (.a(d[0]), .b(d[1]), .sel(sel[0]), .y(low));
    mux2_1 m1 (.a(d[2]), .b(d[3]), .sel(sel[0]), .y(hig
h));mux2_1 m2 (.a(low), .b(high), .sel(sel[1]), .y(y));
endmodule

module mux8_1 (
    input [7:0] d,
    input [2:0] sel,
    output y
);
    wire low4, high4;
    mux4_1 mlow (.d(d[3:0]), .sel(sel[1:0]), .y(low4));
    mux4_1 mhigh(.d(d[7:4]), .sel(sel[1:0]), .y(high4));
    mux2_1 mtop (.a(low4), .b(high4), .sel(sel[2]), .y(y));
endmodule
```

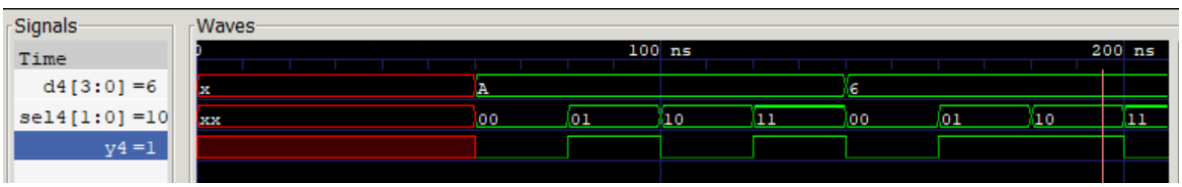
```
1  `timescale 1ns/1ps
2  module mux_tb;
3
4      reg a, b, sel;
5      wire y2;
6      reg [3:0] d4;
7      reg [1:0] sel4;
8      wire y4;
9      reg [7:0] d8;
10     reg [2:0] sel8;
11     wire y8;
12
13     mux2_1 uut2 (
14         .a(a),
15         .b(b),
16         .sel(sel),
17         .y(y2));
18     mux4_1 uut4 (
19         .d(d4),
20         .sel(sel4),
21         .y(y4));
22     mux8_1 uut8 (
23         .d(d8),
24         .sel(sel8),
25         .y(y8));
26
27     initial begin
28         $dumpfile("mux_wave.vcd");
29         $dumpvars(0, mux_tb);
30     end
31
32     initial begin
33         a = 0; b = 1;
34         sel = 0; #10;
35         sel = 1; #10;
36         a = 1; b = 0;
37         sel = 0; #10;
38         sel = 1; #10;
39         a = 0; b = 1;
40         sel = 0; #10;
41         sel = 1; #10;
42
43         d4 = 4'b1010;
44         sel4 = 2'b00; #20;
45         sel4 = 2'b01; #20;
46         sel4 = 2'b10; #20;
47         sel4 = 2'b11; #20;
48         d4 = 4'b0110;
49         sel4 = 2'b00; #20;
50         sel4 = 2'b01; #20;
51         sel4 = 2'b10; #20;
52         sel4 = 2'b11; #20;
53
54         d8 = 8'b11010110;
55         for (integer i = 0; i < 8; i = i + 1) begi
56 n           sel8 = i;
57             #10;
58         end
59         $finish;
60     end
61 endmodule
62
```

testbench

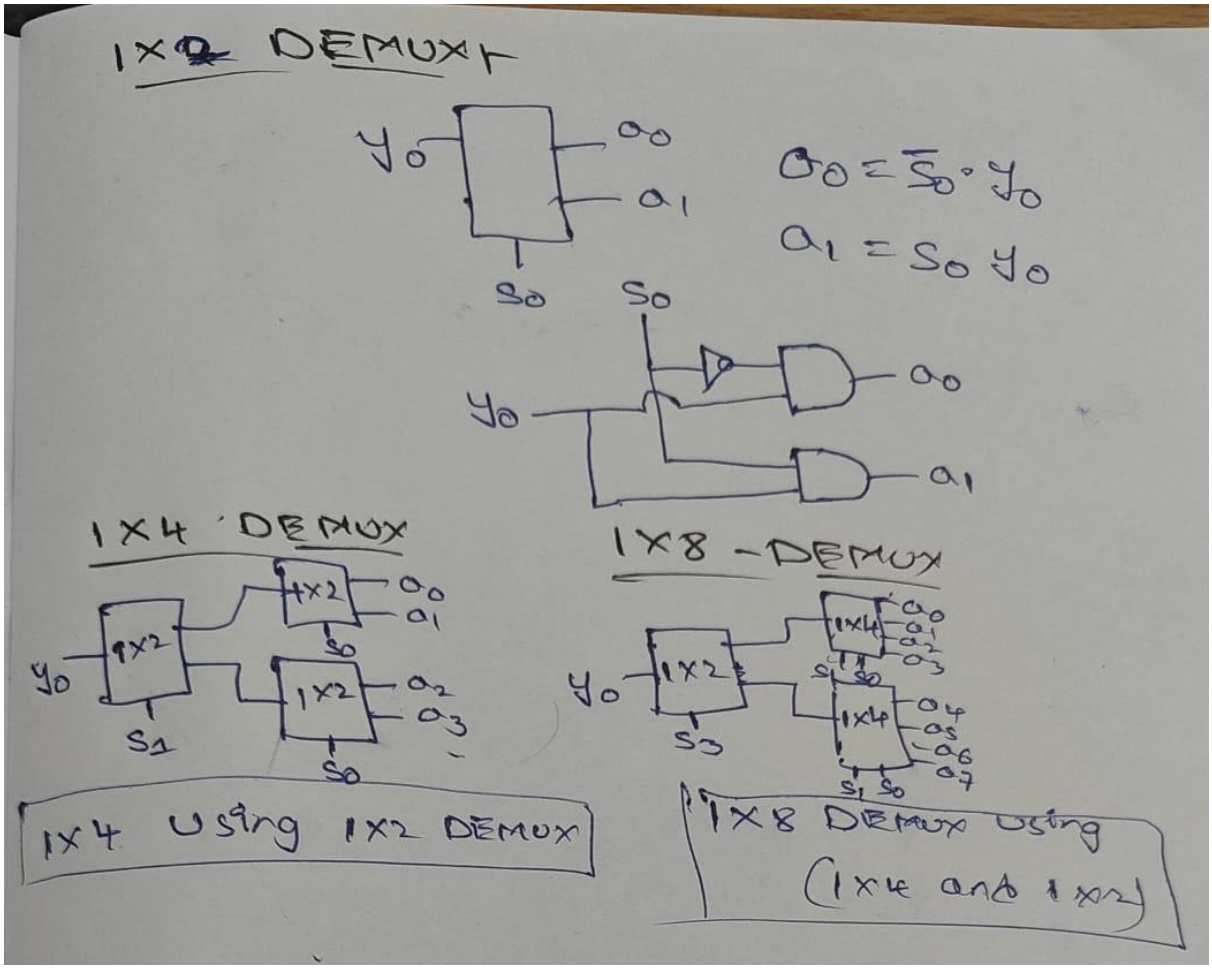
GTKWave







DEMUX



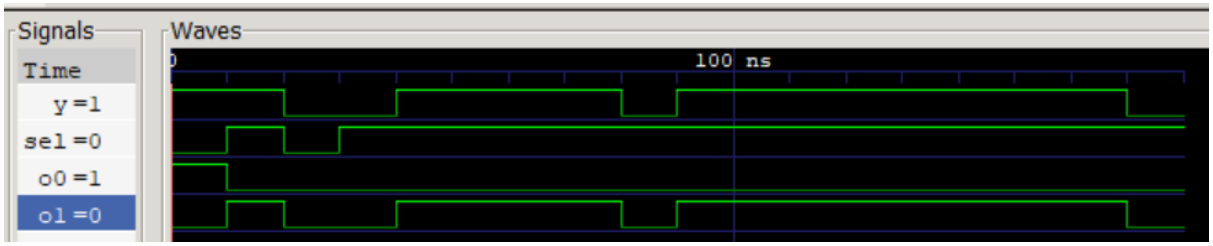
Verilog code

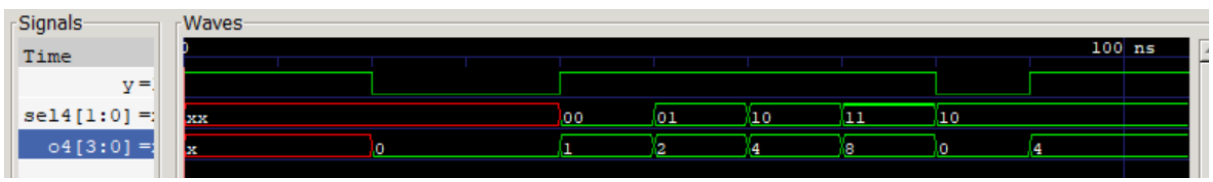
```verilog
`timescale 1ns/1ps

module demux1_2 (
    input y,
    input sel,
    output o0, o1
);
    wire nsel;
    not(nsel, sel);
    and(o0, y, nsel);
    and(o1, y, sel);
endmodule

module demux1_4 (
    input y,
    input [1:0] sel,
    output [3:0] o
);
    wire p0, p1;
    demux1_2 d0 (.y(y), .sel(sel[1]), .o0(p0), .o1(p1));
    demux1_2 d1 (.y(p0), .sel(sel[0]), .o0(o[0]), .o1(o[1]));
    demux1_2 d2 (.y(p1), .sel(sel[0]), .o0(o[2]), .o1(o[3]));
endmodule

module demux1_8 (
    input y,
    input [2:0] sel,
    output [7:0] o
);
    wire low_en, high_en;
    demux1_2 top (.y(y), .sel(sel[2]), .o0(low_en), .o1(high_e
n));demux1_4 low  (.y(low_en),  .sel(sel[1:0]), .o(o[3:0]));
    demux1_4 high (.y(high_en), .sel(sel[1:0]), .o(o[7:4]));
endmodule


```

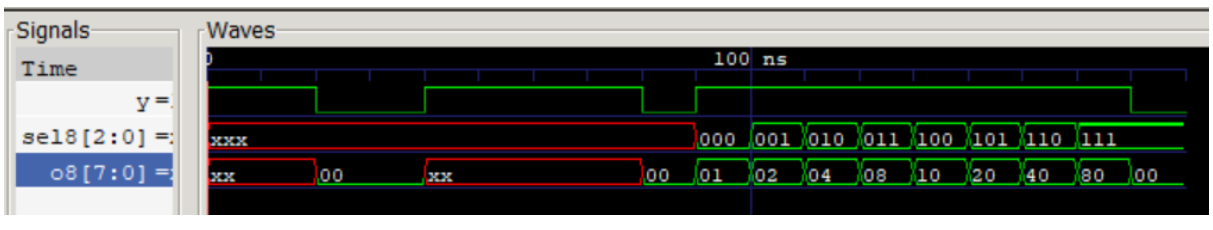
testbench

```
module demux_tb;
    reg y;
    reg sel;
    wire o0, o1;
    reg [1:0] sel4;
    wire [3:0] o4;
    reg [2:0] sel8;
    wire [7:0] o8;

    demux1_2 uut2 (.y(y), .sel(sel), .o0(o0), .o1(o1));
    demux1_4 uut4 (.y(y), .sel(sel4), .o(o4));
    demux1_8 uut8 (.y(y), .sel(sel8), .o(o8));

    initial begin
        $dumpfile("demux_wave.vcd");
        $dumpvars(0, demux_tb);
    end

    initial begin
        y = 1;
        sel = 0; #10;
        sel = 1; #10;
        y = 0;
        sel = 0; #10;
        sel = 1; #10;

        y = 1;
        sel4 = 2'b00; #10;
        sel4 = 2'b01; #10;
        sel4 = 2'b10; #10;
        sel4 = 2'b11; #10;
        y = 0;
        sel4 = 2'b10; #10;

        y = 1;
        sel8 = 3'b000; #10;
        sel8 = 3'b001; #10;
        sel8 = 3'b010; #10;
        sel8 = 3'b011; #10;
        sel8 = 3'b100; #10;
        sel8 = 3'b101; #10;
        sel8 = 3'b110; #10;
        sel8 = 3'b111; #10;

        y = 0; #10;
        $finish;
    end
endmodule
```

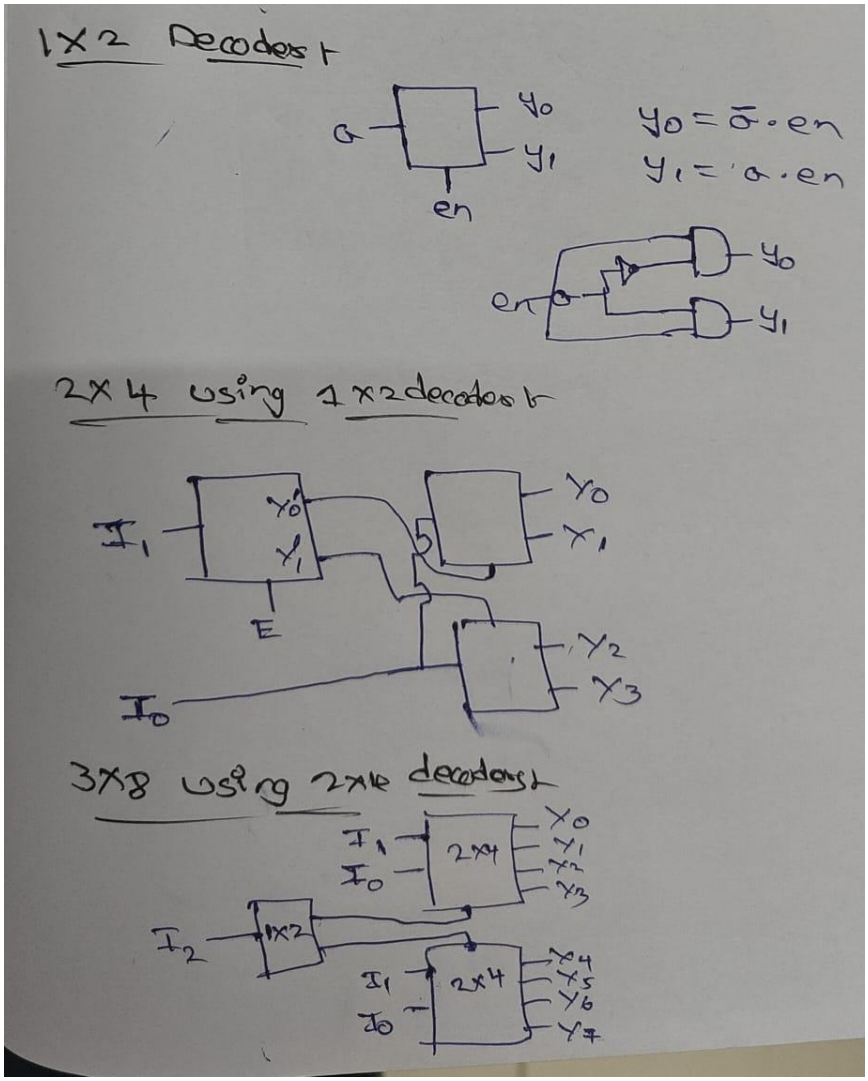
GTKWave







DECODER



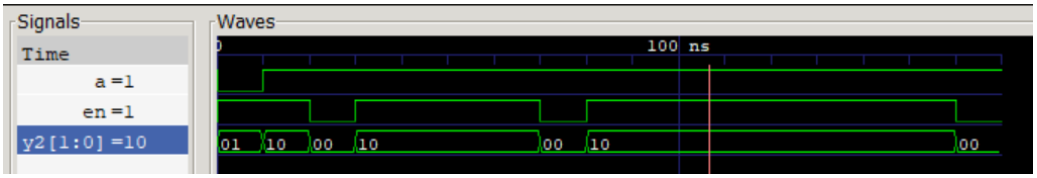
testbench

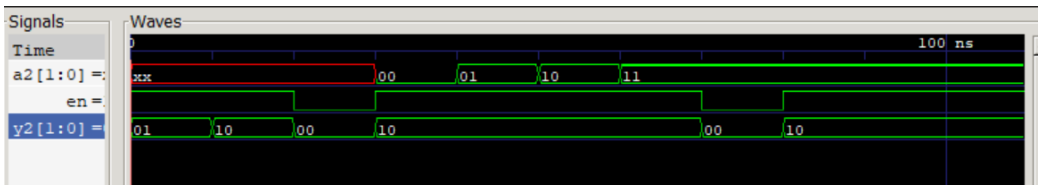
```
`timescale 1ns/1ps
module dec_tb;
    reg a;
    reg en;
    wire [1:0] y2;
    reg [1:0] a2;
    wire [3:0] y4;
    reg [2:0] a3;
    wire [7:0] y8;
    dec2_1 D2 (.a(a), .en(en), .y(y2));
    dec2_4 D4 (.a(a2), .en(en), .y(y
4));dec3_8 D8 (.a(a3), .en(en), .y(y
8));initial begin
        $dumpfile("decoder_wave.vcd");
        $dumpvars(0, dec_tb);
    end

    initial begin
        en = 1; a = 0; #10;
        a = 1; #10;
        en = 0; a = 1; #10;

        en = 1;
        a2 = 2'b00; #10;
        a2 = 2'b01; #10;
        a2 = 2'b10; #10;
        a2 = 2'b11; #10;
        en = 0; #10;

        en = 1;
        a3 = 3'b000; #10;
        a3 = 3'b001; #10;
        a3 = 3'b010; #10;
        a3 = 3'b011; #10;
        a3 = 3'b100; #10;
        a3 = 3'b101; #10;
        a3 = 3'b110; #10;
        a3 = 3'b111; #10;
        en = 0; #10;
        $finish;
    end
endmodule
```

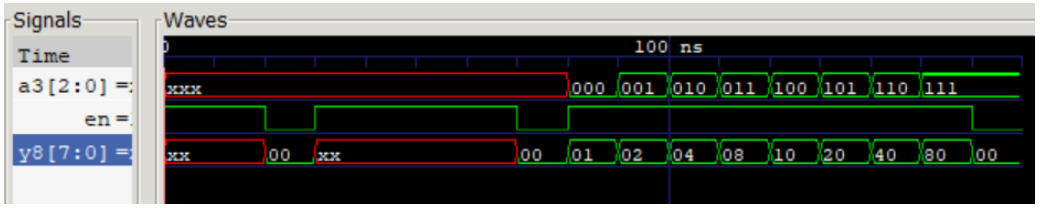
Verilog code

```
`timescale 1ns/1ps
module dec2_1 (
    input a,
    input en,
    output [1:0] y
);
    wire na;
    not(na, a);
    and(y[0], na, en);
    and(y[1], a, en);
endmodule

module dec2_4 (
    input [1:0] a,
    input en,
    output [3:0] y
);
    wire [1:0] e;
    dec2_1 d0 (.a(a[1]), .en(en), .y(e));
    dec2_1 low  (.a(a[0]), .en(e[0]), .y(y[1:0]));
    dec2_1 high (.a(a[0]), .en(e[1]), .y(y[3:2]));
endmodule

module dec3_8 (
    input [2:0] a,
    input en,
    output [7:0] y
);
    wire [1:0] e;
    dec2_1 top (.a(a[2]), .en(en), .y(e));
    dec2_4 low  (.a(a[1:0]), .en(e[0]), .y(y[3:
0]));dec2_4 high (.a(a[1:0]), .en(e[1]), .y(y[7:
4]));dule
```

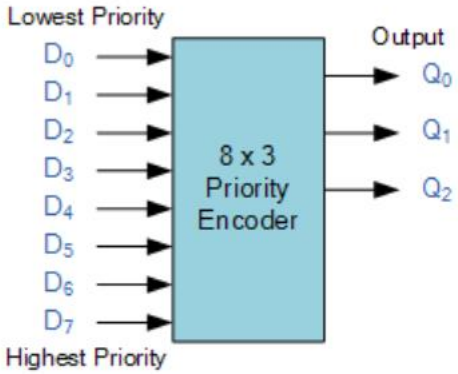
GTKWave







PRIORITY ENCODER



Inputs								Outputs		
D_7	D_6	D_5	D_4	D_3	D_2	D_1	D_0	Q_2	Q_1	Q_0
0	0	0	0	0	0	0	1	0	0	0
0	0	0	0	0	0	1	x	0	0	1
0	0	0	0	0	1	x	x	0	1	0
0	0	0	0	1	x	x	x	0	1	1
0	0	0	1	x	x	x	x	1	0	0
0	0	1	x	x	x	x	x	1	0	1
0	1	x	x	x	x	x	x	1	1	0
1	x	x	x	x	x	x	x	1	1	1

X = dont care

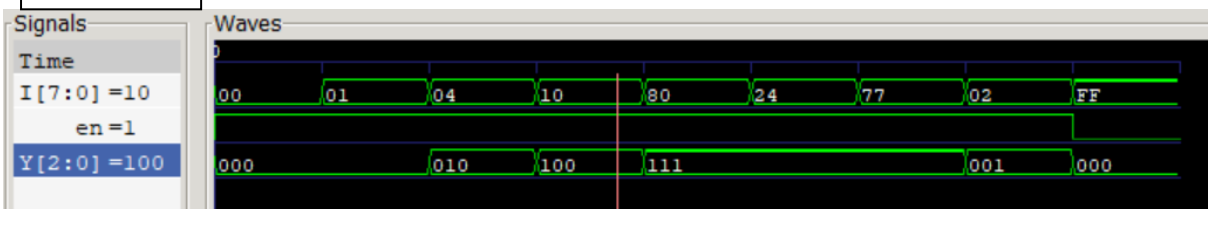
Verilog code

```
`timescale 1ns/1ps
module priority_encoder_8x3 (
    input [7:0] I,
    input en,
    output [2:0] Y,
    output valid);
    wire any;
    wire b2, b1, b0;
    or(any, I[0], I[1], I[2], I[3], I[4], I[5], I[6], I
[7]);or(b2, I[7], I[6], I[5], I[4]);
    or(b1, I[7], I[6], I[3], I[2]);
    or(b0, I[7], I[5], I[3], I[1]);
    and(Y[2], b2, en);
    and(Y[1], b1, en);
    and(Y[0], b0, en);
    and(valid, any, en);
endmodule
```

testbench

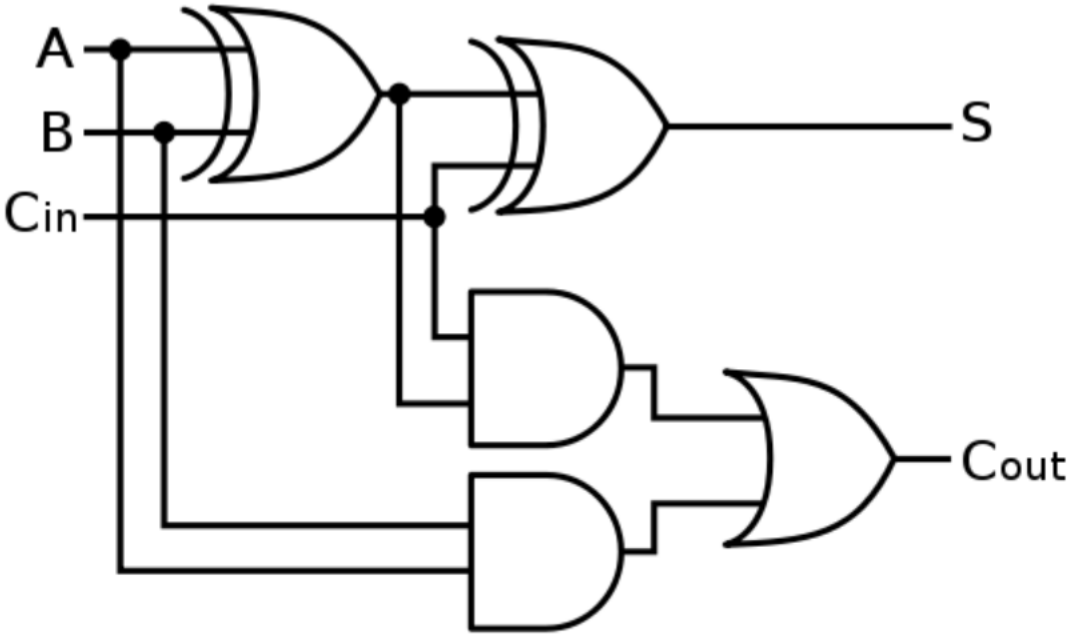
```
`timescale 1ns/1ps
module priority_encoder_tb;
    reg [7:0] I;
    reg en;
    wire [2:0] Y;
    wire valid;
    priority_encoder_8x3 dut (.I(I), .en(en), .Y(Y), .valid(valid));
    initial begin
        $dumpfile("priority_encoder_wave.vcd");
        $dumpvars(0, priority_encoder_tb);
    end

    initial begin
        en = 1;
        I = 8'b00000000; #10;
        I = 8'b00000001; #10;
        I = 8'b00000100; #10;
        I = 8'b00010000; #10;
        I = 8'b10000000; #10;
        I = 8'b00100100; #10;
        I = 8'b01110111; #10;
        I = 8'b00000010; #10;
        en = 0;
        I = 8'b11111111; #10;
        $finish;
    end
endmodule
```

GTKWave



4

1bit Full Adder



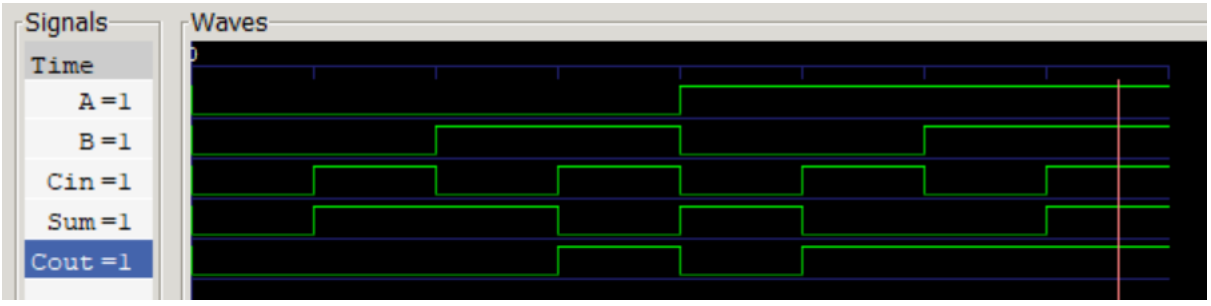
Verilog code

```verilog
`timescale 1ns / 1ps
module full_adder(
    input A, B, Cin,
    output Sum, Cout
);
    assign Sum=A^B^Cin;
    assign Cout=(A&B)|((A^B)&Ci
endmodule
```

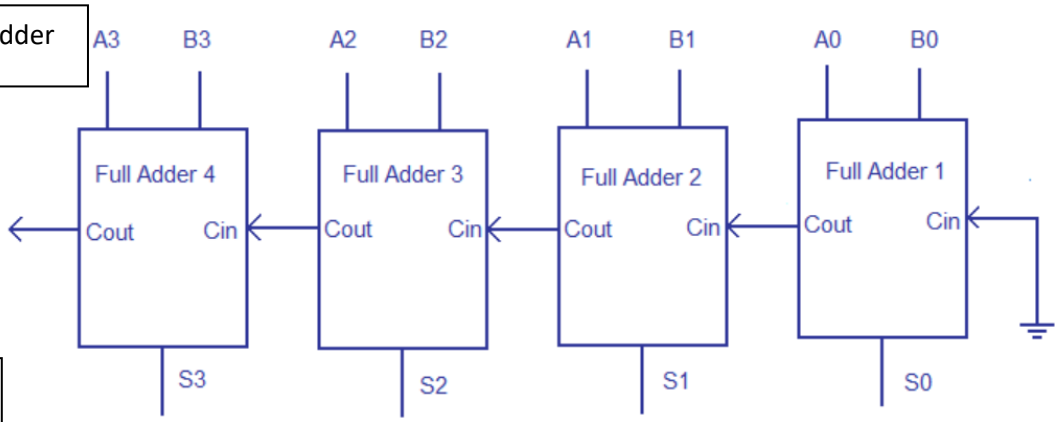
testbench

```verilog
`timescale 1ns / 1ps
module full_adder_tb;
    reg A, B, Cin;
    wire Sum, Cout;
    full_adder uut (
        .A(A),
        .B(B),
        .Cin(Cin),
        .Sum(Sum),
        .Cout(Cout)
    );
    initial begin
        $dumpfile("full_adder.vcd");
        $dumpvars(0, full_adder_tb);
        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
        A=1; B=1; Cin=1; #10
        $finish;
    end
endmodule
```

GTKWave



4bit Carry Ripple Adder



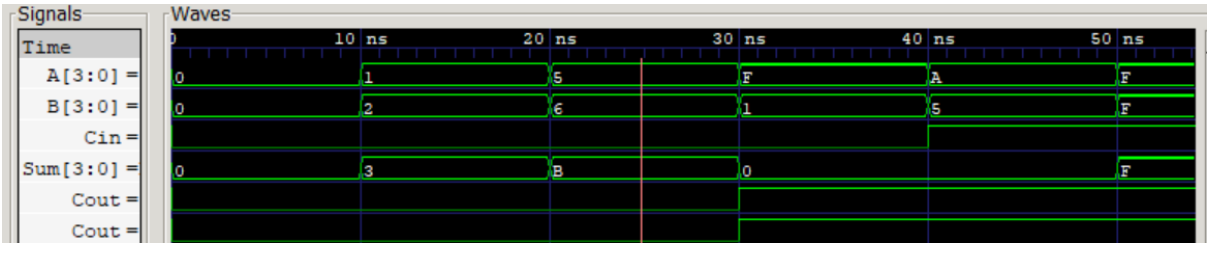
Verilog code

```
`timescale 1ns / 1ps
module full_adder(
    input A, B, Cin,
    output Sum, Cout);
    assign Sum  = A ^ B ^ Cin;
    assign Cout = (A & B) | ((A ^ B) & Cin);
endmodule

module ripple_carry_adder_4bit(
    input [3:0] A, B,
    input Cin,
    output [3:0] Sum,
    output Cout);
    wire C1, C2, C3;
    full_adder FA0 (.A(A[0]), .B(B[0]), .Cin(Cin), .Sum(Sum[0]), .Cout(C1));
    full_adder FA1 (.A(A[1]), .B(B[1]), .Cin(C1),  .Sum(Sum[1]), .Cout(C2));
    full_adder FA2 (.A(A[2]), .B(B[2]), .Cin(C2),  .Sum(Sum[2]), .Cout(C3));
    full_adder FA3 (.A(A[3]), .B(B[3]), .Cin(C3),  .Sum(Sum[3]), .Cout(Cou
t));endmodule
```

testbench

```
module ripple_carry_adder_4bit_tb;
    reg [3:0] A, B;
    reg Cin;
    wire [3:0] Sum;
    wire Cout;
    ripple_carry_adder_4bit uut (
        .A(A),
        .B(B),
        .Cin(Cin),
        .Sum(Sum),
        .Cout(Cout));
    initial begin
        $dumpfile("ripple_carry_adder_full.vcd");
        $dumpvars(0, ripple_carry_adder_4bit_tb);
        A=4'b0000; B=4'b0000; Cin=0; #10
        A=4'b0001; B=4'b0010; Cin=0; #10
        A=4'b0101; B=4'b0110; Cin=0; #10
        A=4'b1111; B=4'b0001; Cin=0; #10
        A=4'b1010; B=4'b0101; Cin=1; #10
        A=4'b1111; B=4'b1111; Cin=1; #10
        $finish;
    end
endmodule
```

GTKWave



16 Bit Carry Ripple Adder

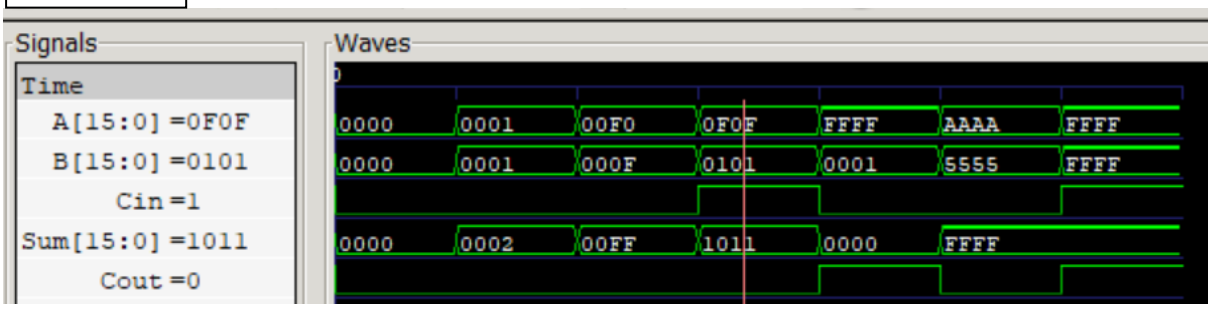
Verilog code

```
`timescale 1ns / 1ps
module full_adder(
    input A, B, Cin,
    output Sum, Cout);
    assign Sum  = A ^ B ^ Cin;
    assign Cout = (A & B) | ((A ^ B) & Cin);
endmodule

module ripple_carry_adder_4bit(
    input [3:0] A, B,
    input Cin,
    output [3:0] Sum,
    output Cout);
    wire C1, C2, C3;
    full_adder FA0 (.A(A[0]), .B(B[0]), .Cin(Cin), .Sum(Sum[0]), .Cout(C1));
    full_adder FA1 (.A(A[1]), .B(B[1]), .Cin(C1),  .Sum(Sum[1]), .Cout(C2));
    full_adder FA2 (.A(A[2]), .B(B[2]), .Cin(C2),  .Sum(Sum[2]), .Cout(C3));
    full_adder FA3 (.A(A[3]), .B(B[3]), .Cin(C3),  .Sum(Sum[3]), .Cout(Cout));
endmodule

module ripple_carry_adder_16bit(
    input  [15:0] A, B,
    input  Cin,
    output [15:0] Sum,
    output Cout);
    wire C1, C2, C3;
    ripple_carry_adder_4bit RCA0 (.A(A[3:0]),   .B(B[3:0]),   .Cin(Cin), .Sum(Sum[3:0]),   .Cout(C1));
    ripple_carry_adder_4bit RCA1 (.A(A[7:4]),   .B(B[7:4]),   .Cin(C1),  .Sum(Sum[7:4]),   .Cout(C2));
    ripple_carry_adder_4bit RCA2 (.A(A[11:8]),  .B(B[11:8]),  .Cin(C2),  .Sum(Sum[11:8]),  .Cout(C3));
    ripple_carry_adder_4bit RCA3 (.A(A[15:12]), .B(B[15:12]), .Cin(C3),  .Sum(Sum[15:12]), .Cout(Cou
t));
endmodule

```

testbench

```
module ripple_carry_adder_16bit_tb;
    reg  [15:0] A, B;
    reg  Cin;
    wire [15:0] Sum;
    wire Cout;
    ripple_carry_adder_16bit uut (
        .A(A),
        .B(B),
        .Cin(Cin),
        .Sum(Sum),
        .Cout(Cout));
    initial begin
        $dumpfile("ripple_carry_adder_16bit.vcd");
        $dumpvars(0, ripple_carry_adder_16bit_tb);
        A = 16'h0000; B = 16'h0000; Cin = 0; #10
        A = 16'h0001; B = 16'h0001; Cin = 0; #10
        A = 16'h00F0; B = 16'h000F; Cin = 0; #10
        A = 16'h0F0F; B = 16'h0101; Cin = 1; #10
        A = 16'hFFFF; B = 16'h0001; Cin = 0; #10
        A = 16'hAAAA; B = 16'h5555; Cin = 0; #10
        A = 16'hFFFF; B = 16'hFFFF; Cin = 1; #10
        $finish;
    end
endmodule

```

GTKWave



4bit Carry Look-Ahead (CLA)

A CLA improves speed by calculating carry bits in parallel, instead of waiting for them to ripple sequentially.

For each bit i:

$$G_i = A_i \cdot B_i \text{(Generate)}$$
$$P_i = A_i \oplus B_i \text{(Propagate)}$$

Carry and Sum equations:

$$C_1 = G_0 + P_0 C_{in}$$
$$C_2 = G_1 + P_1 G_0 + P_1 P_0 C_{in}$$
$$C_3 = G_2 + P_2 G_1 + P_2 P_1 G_0 + P_2 P_1 P_0 C_{in}$$
$$C_4 = G_3 + P_3 G_2 + P_3 P_2 G_1 + P_3 P_2 P_1 G_0 + P_3 P_2 P_1 P_0 C_{in}$$

$$S_i = P_i \oplus C_i$$
$$C_{out} = C_4$$

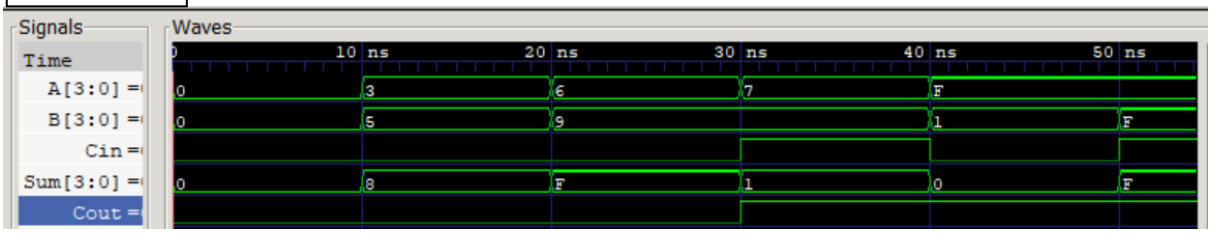
Verilog code

```
`timescale 1ns / 1ps
module carry_lookahead_adder_4bit(
    input [3:0] A, B,
    input Cin,
    output [3:0] Sum,
    output Cout);
    wire [3:0] G, P;
    wire [4:0] C;
    assign C[0] = Cin;
    assign G = A & B;
    assign P = A ^ B;
    assign C[1] = G[0] | (P[0] & C[0]);
    assign C[2] = G[1] | (P[1] & G[0]) | (P[1] & P[0] & C[0]);
    assign C[3] = G[2] | (P[2] & G[1]) | (P[2] & P[1] & G[0]) | (P[2] & P[1] & P[0] & C
[0]);assign C[4] = G[3] | (P[3] & G[2]) | (P[3] & P[2] & G[1]) |
                  (P[3] & P[2] & P[1] & G[0]) | (P[3] & P[2] & P[1] & P[0] & C[0]);
    assign Sum = P ^ C[3:0];
    assign Cout = C[4];
endmodule
```

testbench

```
`timescale 1ns / 1ps
module carry_lookahead_adder_4bit_tb;
    reg [3:0] A, B;
    reg Cin;
    wire [3:0] Sum;
    wire Cout;
    carry_lookahead_adder_4bit uut (
        .A(A),
        .B(B),
        .Cin(Cin),
        .Sum(Sum),
        .Cout(Cout));
    initial begin
        $dumpfile("carry_lookahead_adder_4bit.vcd");
        $dumpvars(0, carry_lookahead_adder_4bit_tb);
        A=4'b0000; B=4'b0000; Cin=0; #10
        A=4'b0011; B=4'b0101; Cin=0; #10
        A=4'b0110; B=4'b1001; Cin=0; #10
        A=4'b0111; B=4'b1001; Cin=1; #10
        A=4'b1111; B=4'b0001; Cin=0; #10
        A=4'b1111; B=4'b1111; Cin=1; #10
        $finish;
    end
endmodule
```

GTKWave



16 bit Carry Look-Ahead

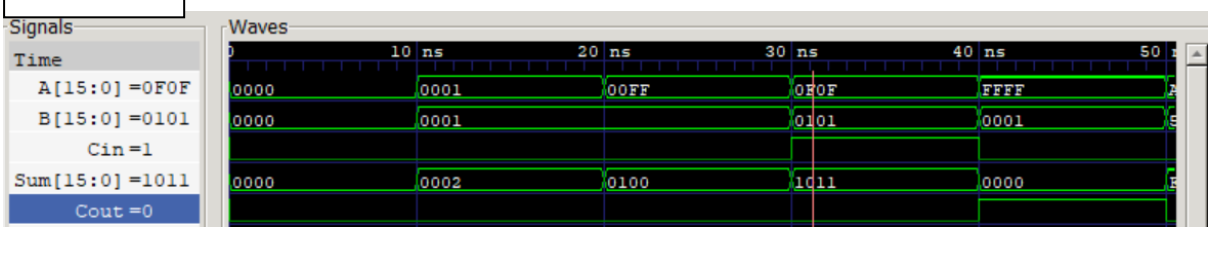
Verilog code

```
`timescale 1ns / 1ps
module carry_lookahead_adder_4bit(
    input [3:0] A, B,
    input Cin,
    output [3:0] Sum,
    output Cout,
    output G_group, P_group );
    wire [3:0] G, P;
    wire [4:0] C;
    assign C[0] = Cin;
    assign G = A & B;
    assign P = A ^ B;
    assign C[1] = G[0] | (P[0] & C[0]);
    assign C[2] = G[1] | (P[1] & G[0]) | (P[1] & P[0] & C[0]);
    assign C[3] = G[2] | (P[2] & G[1]) | (P[2] & P[1] & G[0]) | (P[2] & P[1] & P[0] & C[0]);
    assign C[4] = G[3] | (P[3] & G[2]) | (P[3] & P[2] & G[1]) |
                  (P[3] & P[2] & P[1] & G[0]) | (P[3] & P[2] & P[1] & P[0] & C[0]);
    assign Sum = P ^ C[3:0];
    assign Cout = C[4];
    assign G_group = G[3] | (P[3] & G[2]) |
                     (P[3] & P[2] & G[1]) |
                     (P[3] & P[2] & P[1] & G[0]);
    assign P_group = P[3] & P[2] & P[1] & P[0];
endmodule

module carry_lookahead_adder_16bit(
    input  [15:0] A, B,
    input  Cin,
    output [15:0] Sum,
    output Cout);
    wire [3:0] C;
    wire [3:0] G, P;
    assign C[0] = Cin;
    carry_lookahead_adder_4bit CLA0 (.A(A[3:0]),   .B(B[3:0]),   .Cin(C[0]), .Sum(Sum[3:0]),   .Cout(), .G_group(G[0]), .P_group(P[0]));
    carry_lookahead_adder_4bit CLA1 (.A(A[7:4]),   .B(B[7:4]),   .Cin(C[1]), .Sum(Sum[7:4]),   .Cout(), .G_group(G[1]), .P_group(P[1]));
    carry_lookahead_adder_4bit CLA2 (.A(A[11:8]),  .B(B[11:8]),  .Cin(C[2]), .Sum(Sum[11:8]),  .Cout(), .G_group(G[2]), .P_group(P[2]));
    carry_lookahead_adder_4bit CLA3 (.A(A[15:12]), .B(B[15:12]), .Cin(C[3]), .Sum(Sum[15:12]), .Cout(Cout), .G_group(G[3]), .P_group(P
[3]));assign C[1] = G[0] | (P[0] & C[0]);
    assign C[2] = G[1] | (P[1] & G[0]) | (P[1] & P[0] & C[0]);
    assign C[3] = G[2] | (P[2] & G[1]) | (P[2] & P[1] & G[0]) | (P[2] & P[1] & P[0] & C[0]);
endmodule

```

testbench

```
module carry_lookahead_adder_16bit_tb;
    reg [15:0] A, B;
    reg Cin;
    wire [15:0] Sum;
    wire Cout;
    carry_lookahead_adder_16bit uut (
        .A(A),
        .B(B),
        .Cin(Cin),
        .Sum(Sum),
        .Cout(Cout));
    initial begin
        $dumpfile("carry_lookahead_adder_16bit.vcd");
        $dumpvars(0, carry_lookahead_adder_16bit_tb);
        A = 16'h0000; B = 16'h0000; Cin = 0; #10
        A = 16'h0001; B = 16'h0001; Cin = 0; #10
        A = 16'h00FF; B = 16'h0001; Cin = 0; #10
        A = 16'h0F0F; B = 16'h0101; Cin = 1; #10
        A = 16'hFFFF; B = 16'h0001; Cin = 0; #10
        A = 16'hAAAA; B = 16'h5555; Cin = 0; #10
        A = 16'hFFFF; B = 16'hFFFF; Cin = 1; #10
        $finish;
    end
endmodule

```

GTKWave



5

flipflops

Q_N	Q_{N+1}	S	R	J	K	D	T
0	0	0	X	0	X	0	0
0	1	1	0	1	X	1	1
1	0	0	1	X	1	0	1
1	1	X	0	X	0	1	0

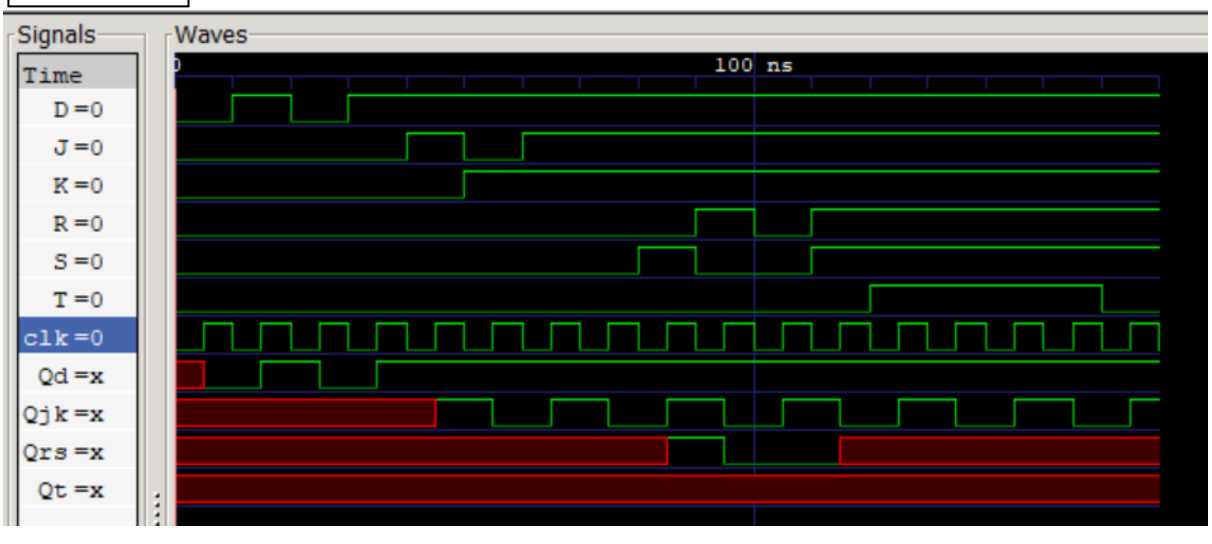
Verilog code

```
`timescale 1ns / 1ps
module d_flipflop(
    input D,
    input clk,
    output reg Q);
    always @(posedge clk)
        Q <= D;
endmodule

module jk_flipflop(
    input J, K, clk,
    output reg Q);
    always @(posedge clk) begi
n       case ({J, K})
            2'b00: Q <= Q;
            2'b01: Q <= 0;
            2'b10: Q <= 1;
            2'b11: Q <= ~Q;
        endcase
    end
endmodule

module rs_flipflop(
    input R, S, clk,
    output reg Q);
    always @(posedge clk) begi
n       case ({R, S})
            2'b00: Q <= Q;
            2'b01: Q <= 1;
            2'b10: Q <= 0;
            2'b11: Q <= 1'bx;
        endcase
    end
endmodule

module t_flipflop(
    input T, clk,
    output reg Q);
    always @(posedge clk) begi
n       if (T)
            Q <= ~Q;
        else
            Q <= Q;
    end
endmodule
```

testbench

```
module flipflops_all_tb;
    reg D, J, K, R, S, T, clk;
    wire Qd, Qjk, Qrs, Qt;
    d_flipflop  DFF (.D(D), .clk(clk), .Q(Qd));
    jk_flipflop JKFF (.J(J), .K(K), .clk(clk), .Q(Qjk));
    rs_flipflop RSFF (.R(R), .S(S), .clk(clk), .Q(Qrs));
    t_flipflop  TFF (.T(T), .clk(clk), .Q(Qt));
    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end
    initial begin
        $dumpfile("flipflops_all.vcd");
        $dumpvars(0, flipflops_all_tb);
        D=0; J=0; K=0; R=0; S=0; T=0;
        #10;
        D=1; #10;
        D=0; #10;
        D=1; #10;
        J=1; K=0; #10;
        J=0; K=1; #10;
        J=1; K=1; #20;
        R=0; S=1; #10;
        R=1; S=0; #10;
        R=0; S=0; #10;
        R=1; S=1; #10;
        T=1; #40;
        T=0; #10;
        $finish;
    end
    initial begin
        $monitor("%4t | %b  %b | %b %b  %b | %b %b  %b | %b  %b",
                 $time, D, Qd, J, K, Qjk, R, S, Qrs, T, Qt);
    end
endmodule

```

GTKWave



6

4 bit universal shift register

S1	S0	Operation	Description
0	0	Hold	. Keeps previous data (no change in output)
0	1	Shift Right	Shifts all bits to the right; MSB receives SR_in
1	0	Shift Left	Shifts all bits to the left; LSB receives SL_in
1	1	Parallel Load	Loads the 4-bit input P into the register

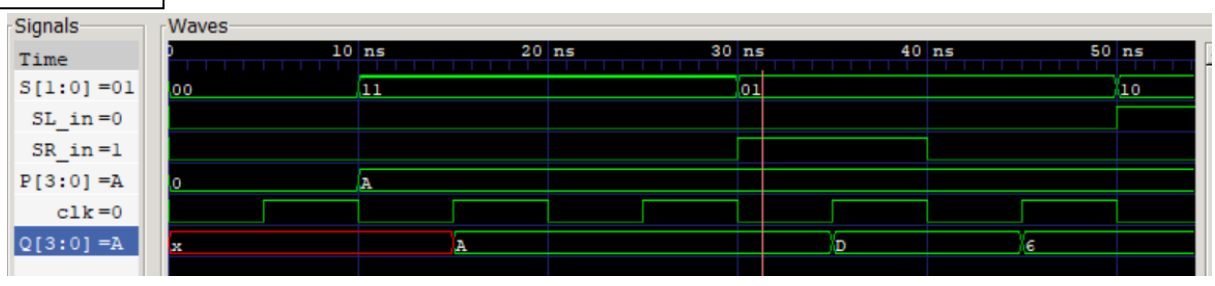
testbench

```
`timescale 1ns / 1ps
module universal_shift_register_tb;
    reg clk;
    reg [1:0] S;
    reg [3:0] P;
    reg SL_in, SR_in;
    wire [3:0] Q;
    universal_shift_register uut (
        .clk(clk),
        .S(S),
        .P(P),
        .SL_in(SL_in),
        .SR_in(SR_in),
        .Q(Q));
    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end
    initial begin
        $dumpfile("universal_shift_register.vcd");
        $dumpvars(0, universal_shift_register_tb);
        S = 2'b00; P = 4'b0000; SL_in = 0; SR_in =
0;      #10 S = 2'b11; P = 4'b1010;
        #10;
        #10 S = 2'b00;
        SR_in = 1; S = 2'b01; #10;
        SR_in = 0; #10;
        SL_in = 1; S = 2'b10; #10;
        SL_in = 0; #10;
        S = 2'b11; P = 4'b1100; #10;
        S = 2'b00; #10;
        $finish;
    end
    initial begin
        $monitor("%4t |  %b  | %b |  %b  %b | %b",
                 $time, S, P, SL_in, SR_in, Q);
    end
endmodule

```

Verilog code

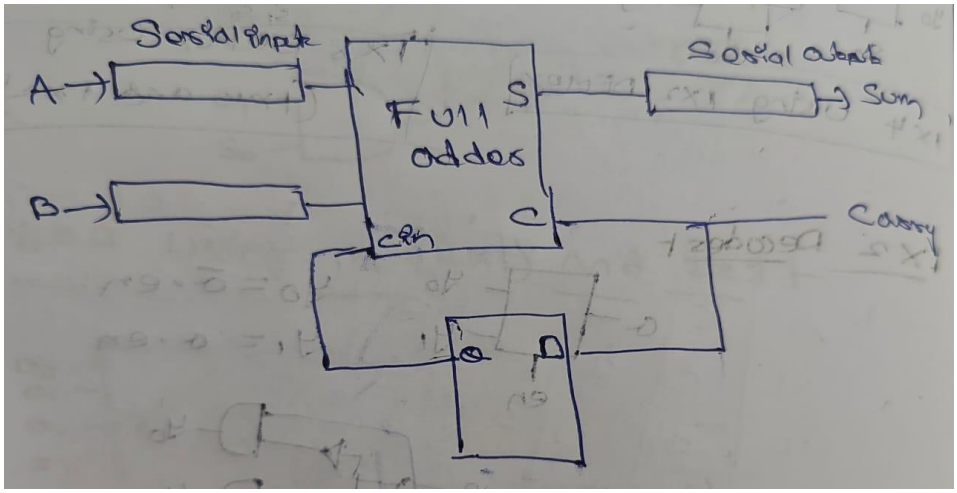
```
`timescale 1ns / 1ps
module universal_shift_register (
    input clk,
    input [1:0] S,
    input [3:0] P,
    input SL_in, SR_in,
    output reg [3:0] Q);
    always @(posedge clk) begin
        case (S)
            2'b00: Q <= Q;
            2'b01: Q <= {SR_in, Q[3:
1]};        2'b10: Q <= {Q[2:0], SL_i
n};         2'b11: Q <= P;
            default: Q <= Q;
        endcase
    end
endmodule
```

GTKWave



7

3 bit Serial Adder



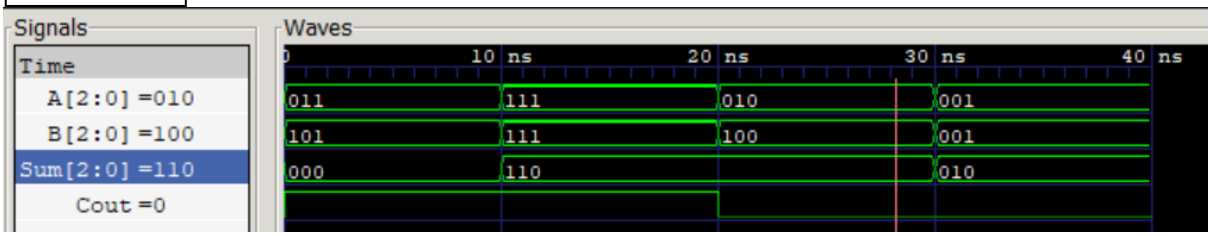
Verilog code

```verilog
`timescale 1ns / 1ps
module serial_adder(
    input [2:0] A, B,
    output reg [2:0] Sum,
    output reg Cout);
    integer i;
    reg carry;
    always @(*) begin
        carry = 0;
        for (i = 0; i < 3; i = i + 1) begin
            {carry, Sum[i]} = A[i] + B[i] + carr
y;      end
        Cout = carry;
    end
endmodule

```

testbench

```verilog
`timescale 1ns / 1ps
module serial_adder_tb;
    reg [2:0] A, B;
    wire [2:0] Sum;
    wire Cout;
    serial_adder uut (.A(A), .B(B), .Sum(Sum), .Cout(Cout));
    initial begin
        $dumpfile("serial_adder.vcd");
        $dumpvars(0, serial_adder_tb);
        A = 3'b011; B = 3'b101; #10;
        A = 3'b111; B = 3'b111; #10;
        A = 3'b010; B = 3'b100; #10;
        A = 3'b001; B = 3'b001; #10;
        $finish;
    end
endmodule

```

GTKWave



8

4bit synchronous Binary Counter

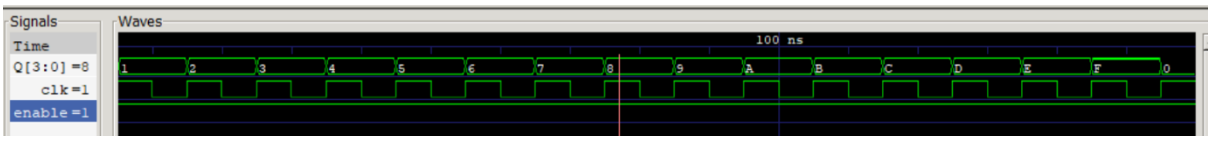
Verilog code

```verilog
`timescale 1ns / 1ps
module sync_counter_4bit(
    input clk,
    input enable,
    output reg [3:0] Q);
    initial Q = 4'b0000;
    always @(posedge clk) begi
n       if (enable)
            Q <= Q + 1;
    end
endmodule
```

testbench

```verilog
`timescale 1ns / 1ps
module sync_counter_4bit_tb;
    reg clk, enable;
    wire [3:0] Q;
    sync_counter_4bit uut (.clk(clk), .enable(enable), .Q
(Q));initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end
    initial begin
        $dumpfile("sync_counter_4bit.vcd");
        $dumpvars(0, sync_counter_4bit_tb);
        enable = 1; #160;
        $finish;
    end
    initial $monitor("Time=%0t  Count=%b", $time, Q);
endmodule

```

GTKWave



4bit up-down binary counter

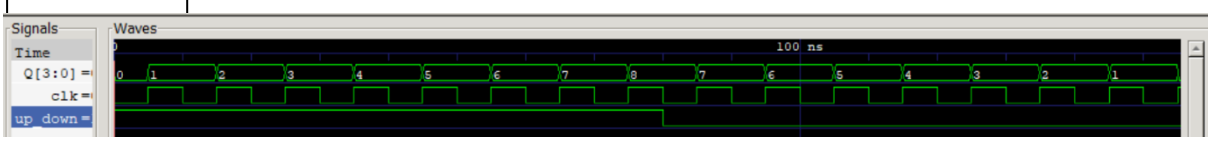
Verilog code

```
`timescale 1ns / 1ps
module updown_counter_4bit(
    input clk,
    input up_down,
    output reg [3:0] Q);
    initial Q = 4'b0000;
    always @(posedge clk) begi
n       if (up_down)
            Q <= Q + 1;
        else
            Q <= Q - 1;
    end
endmodule

```

testbench

```
`timescale 1ns / 1ps
module updown_counter_4bit_tb;
    reg clk, up_down;
    wire [3:0] Q;
    updown_counter_4bit uut (.clk(clk), .up_down(up_down), .Q
(Q));initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end
    initial begin
        $dumpfile("updown_counter_4bit.vcd");
        $dumpvars(0, updown_counter_4bit_tb);
        up_down = 1; #80;
        up_down = 0; #80;
        $finish;
    end
endmodule
```

GTKWave



4bit Ripple counter

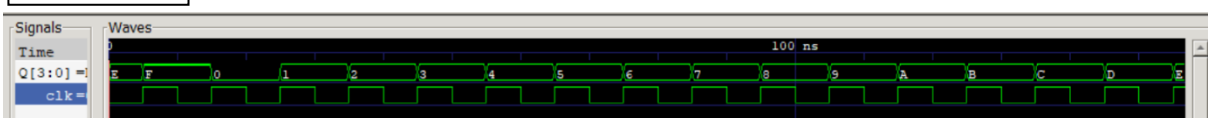
Verilog code

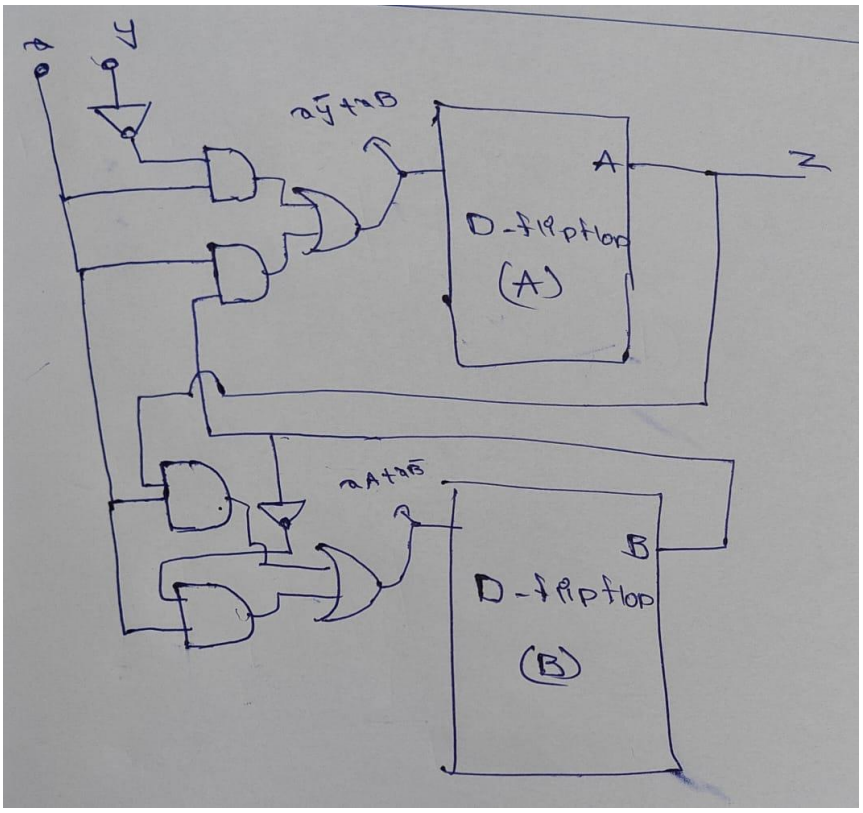
```
`timescale 1ns / 1ps
module ripple_counter_4bit(
    input clk,
    output [3:0] Q);
    reg [3:0] count = 0;
    always @(posedge clk)
        count[0] <= ~count
[0];always @(negedge count
[0])    count[1] <= ~count
[1];always @(negedge count
[1])    count[2] <= ~count
[2];always @(negedge count
[2])    count[3] <= ~count
[3];assign Q = count;
endmodule

```

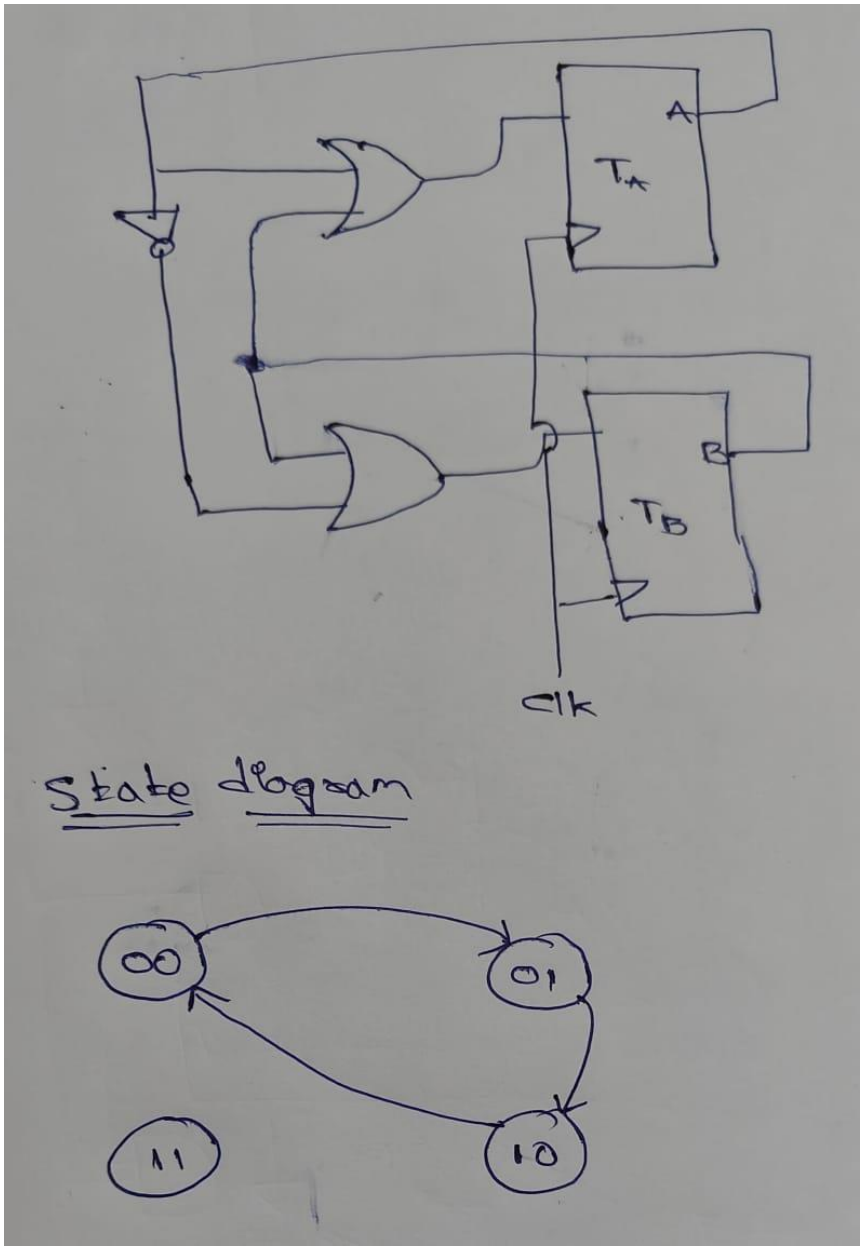
testbench

```
`timescale 1ns / 1ps
module ripple_counter_4bit_tb;
    reg clk;
    wire [3:0] Q;
    ripple_counter_4bit uut (.clk(clk), .Q
(Q));initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end
    initial begin
        $dumpfile("ripple_counter_4bit.vcd");
        $dumpvars(0, ripple_counter_4bit_tb);
        #160;
        $finish;
    end
endmodule

```

GTKWave







State diagram

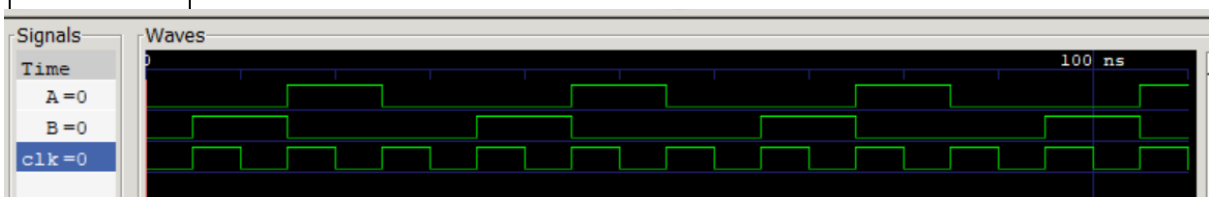
Verilog code

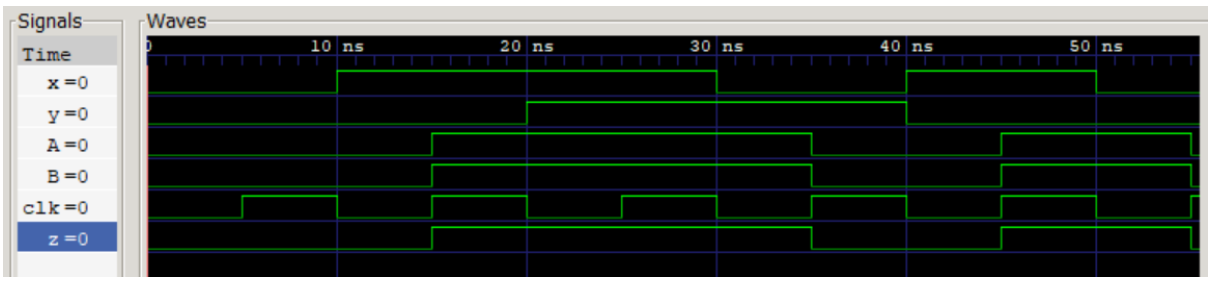
```
`timescale 1ns / 1ps
module sequential_circuit_DFF(
    input  clk,
    input  x, y,
    output reg z);
    reg A, B;
    wire A_next, B_next;
    initial begin
        A = 1'b0;
        B = 1'b0;
    end
    assign A_next = (x & ~y) | (x &
B); assign B_next = (x & A) | (x & ~
B); always @(posedge clk) begin
        A <= A_next;
        B <= B_next;
    end
    always @(*) z = A;
endmodule
```

testbench

```
module exercise9_tb;
    reg clk, x, y;
    wire z;
    sequential_circuit_DFF seq_dff (.clk(clk), .x(x), .y(y), .z
(z));sequential_circuit_TFF seq_tff (.clk(clk));
    initial begin
        clk = 0;
        forever #5 clk = ~clk;
    end
    initial begin
        $dumpfile("exercise9_final.vcd");
        $dumpvars(0, exercise9_tb);
        x = 0; y = 0; #10;
        x = 1; y = 0; #10;
        x = 1; y = 1; #10;
        x = 0; y = 1; #10;
        x = 1; y = 0; #10;
        x = 0; y = 0; #10;
        #50;
        $finish;
    end
endmodule
```

GTKWave





10

- Implement Moore FSM for given FSM diagram

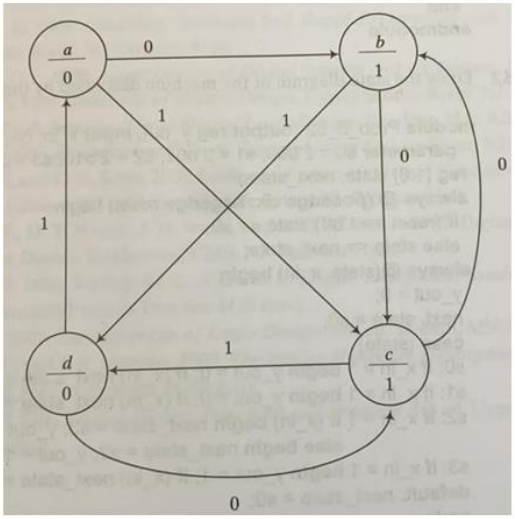


Figure 2: Circuit

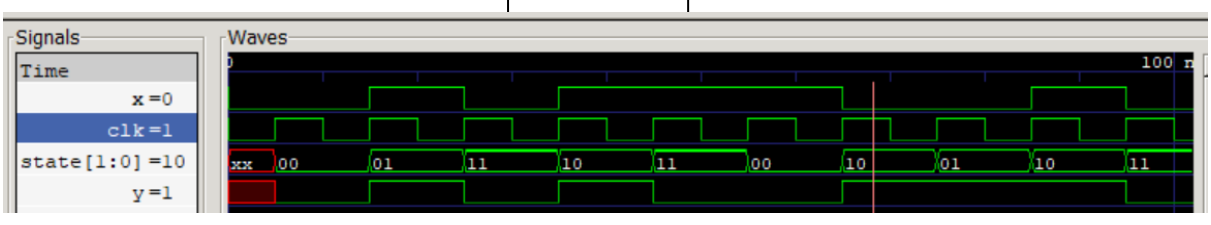
Verilog code

```
`timescale 1ns / 1ps
module moore_fsm(
    input  wire clk,
    input  wire x,
    output reg  y,
    output reg [1:0] state);
    localparam [1:0] A = 2'b00,
                     B = 2'b01,
                     C = 2'b10,
                     D = 2'b11;
    reg [1:0] next_state;
    always @(*) begin
        case (state)
            A: begin
                if (x == 1'b0) next_state =
B;              else           next_state = C;
            end
            B: begin
                if (x == 1'b0) next_state =
C;              else           next_state = D;
            end
            C: begin
                if (x == 1'b0) next_state =
B;              else           next_state = D;
            end
            D: begin
                if (x == 1'b0) next_state =
C;              else           next_state = A;
            end
            default: next_state = A;
        endcase
    end
    always @(posedge clk) begin
        if ($time < 1)
            state <= A;
        else
            state <= next_state;
    end
    always @(*) begin
        case (state)
            A: y = 1'b0;
            B: y = 1'b1;
            C: y = 1'b1;
            D: y = 1'b0;
            default: y = 1'b0;
        endcase
    end
endmodule
```

testbench

```
`timescale 1ns/1ps
module tb_moore_fsm;
    reg clk;
    reg x;
    wire y;
    wire [1:0] state;
    moore_fsm dut (
        .clk(clk),
        .x(x),
        .y(y),
        .state(state));
    initial clk = 0;
    always #5 clk = ~clk;
    initial begin
        x = 0;
        @(posedge clk); x = 1'b0; #1;
        @(posedge clk); x = 1'b1; #1;
        @(posedge clk); x = 1'b0; #1;
        @(posedge clk); x = 1'b1; #1;
        @(posedge clk); x = 1'b1; #1;
        @(posedge clk); x = 1'b1; #1;
        @(posedge clk); x = 1'b0; #1;
        @(posedge clk); x = 1'b0; #1;
        @(posedge clk); x = 1'b1; #1;
        @(posedge clk); x = 1'b0; #1;
        repeat (6) begin
            @(posedge clk); x = $random; #
1;      end
        #20;
        $finish;
    end
    initial begin
        $dumpfile("moore_fsm.vcd");
        $dumpvars(0, tb_moore_fsm);
    end
endmodule
```

GTKWave



- Implement Mealy FSM for given FSM diagram

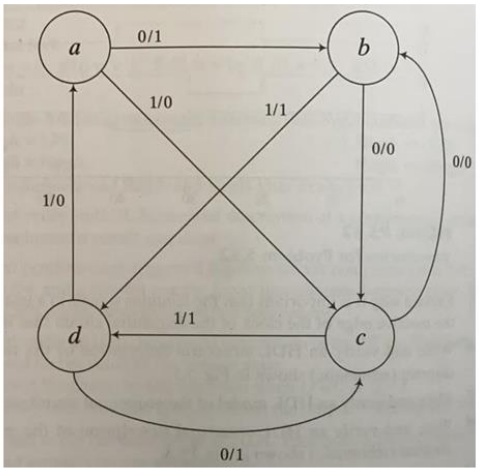


Figure 3: Circuit

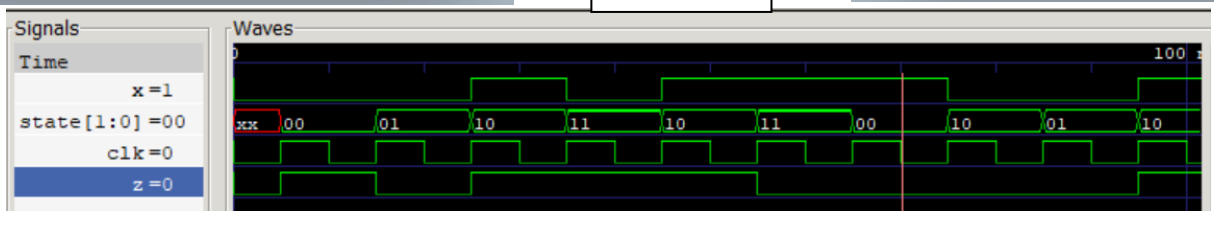
testbench

```
`timescale 1ns/1ps
module tb_mealy_fsm;
    reg clk;
    reg x;
    wire z;
    wire [1:0] state;
    mealy_fsm dut (
        .clk(clk),
        .x(x),
        .z(z),
        .state(state));
    initial clk = 0;
    always #5 clk = ~clk;
    function [8*4:1] state_name;
        input [1:0] s;
        begin
            case (s)
                2'b00: state_name = "A";
                2'b01: state_name = "B";
                2'b10: state_name = "C";
                2'b11: state_name = "D";
                default: state_name = "??";
            endcase
        end
    endfunction
    initial begin
        x = 0;
        #12;
        @(posedge clk); x = 1'b0; #1;
        @(posedge clk); x = 1'b1; #1;
        @(posedge clk); x = 1'b0; #1;
        @(posedge clk); x = 1'b1; #1;
        @(posedge clk); x = 1'b1; #1;
        @(posedge clk); x = 1'b1; #1;
        @(posedge clk); x = 1'b0; #1;
        @(posedge clk); x = 1'b0; #1;
        @(posedge clk); x = 1'b1; #1;
        @(posedge clk); x = 1'b0; #1;
        repeat (6) begin
            @(posedge clk); x = $random; #
1;      end

        #20;
        $finish;
    end
    initial begin
        $dumpfile("mealy_fsm.vcd");
        $dumpvars(0, tb_mealy_fsm);
    end
endmodule

```

Verilog code

```
`timescale 1ns/1ps
module mealy_fsm (
    input  wire clk,
    input  wire x,
    output reg  z,
    output reg [1:0] state);
    localparam [1:0] A = 2'b00,
                     B = 2'b01,
                     C = 2'b10,
                     D = 2'b11;
    reg [1:0] next_state;
    always @(*) begin
        next_state = A;
        z = 1'b0;
        case (state)
            A: begin
                if (x == 1'b0) begi
n                   next_state = B;
                    z = 1'b1;
                end else begin
                    next_state = C;
                    z = 1'b0;
                end
            end
            B: begin
                if (x == 1'b0) begi
n                   next_state = C;
                    z = 1'b0;
                end else begin
                    next_state = D;
                    z = 1'b1;
                end
            end
            C: begin
                if (x == 1'b0) begi
n                   next_state = B;
                    z = 1'b0;
                end else begin
                    next_state = D;
                    z = 1'b1;
                end
            end
            D: begin
                if (x == 1'b0) begi
n                   next_state = C;
                    z = 1'b1;
                end else begin
                    next_state = A;
                    z = 1'b0;
                end
            end
            default: begin
                next_state = A;
                z = 1'b0;
            end
        endcase
    end
    always @(posedge clk) begin
        state <= next_state;
    end
endmodule
```

GTKWave



This circuit is a finite state machine that counts the number of logic ‘1’s and logic ‘0’s given through a single input line. It detects the falling edges of the input states to count each valid input only once, even if the signal remains high for multiple clock cycles. Two internal counters keep track of the number of 1’s and 0’s received. When both counters reach at least two, the output is asserted high. The design uses clocked sequential logic with edge detection to ensure accurate input counting and reliable state transitions.

testbench

```
`timescale 1ns / 1ps
module tb_fsm_11;
    reg CLK;
    reg RESET;
    reg IN;
    wire OUT;
    fsm_11 dut (
        .CLK(CLK),
        .RESET(RESET),
        .IN(IN),
        .OUT(OUT));
    initial CLK = 0;
    always #5 CLK = ~CLK;
    initial begin
        $dumpfile("fsm_11.vcd");
        $dumpvars(0, tb_fsm_11);
        RESET = 1;
        IN = 0;
        #20 RESET = 0;
        #30 IN = 1;
        #30 IN = 0;
        #20 IN = 0;
        #20 IN = 1;
        #30 IN = 1;
        #30 IN = 0;
        #20 IN = 0;
        #20 IN = 1;
        #50 $finish;
    end
endmodule
```

Verilog code

```
`timescale 1ns / 1ps
module fsm_11(
    input  wire CLK,
    input  wire RESET,
    input  wire IN,
    output reg  OUT);
    wire ONE  = IN;
    wire ZERO = ~IN;
    reg prev_one;
    reg prev_zero;
    reg [1:0] count_one;
    reg [1:0] count_zero;
    always @(posedge CLK or posedge RESET) begin
        if (RESET) begin
            prev_one   <= 1'b0;
            prev_zero  <= 1'b0;
            count_one  <= 2'd0;
            count_zero <= 2'd0;
            OUT        <= 1'b0;
        end else begin
            if (prev_one && !ONE) begin
                count_one <= count_one + 1;
            end
            if (prev_zero && !ZERO) begin
                count_zero <= count_zero + 1;
            end
            prev_one  <= ONE;
            prev_zero <= ZERO;
            if ((count_one >= 2) && (count_zero >=
2))
                OUT <= 1'b1;
            else
                OUT <= 1'b0;
        end
    end
endmodule

```

GTKWave