

Name – SAI POOJITH (2025122010)

Practice Problems for Verilog

VLSI DESIGN

```

1 `timescale 1ns / 1ps
2
3 module combinational_gates(a, b, and_out, or_out, nand_out, nor_out, xor_out, xnor_out, not_a, not_b);
4   input a, b;
5   output and_out, or_out, nand_out, nor_out, xor_out, xnor_out, not_a, not_b;
6
7   assign and_out = a & b;
8   assign or_out = a | b;
9   assign nand_out = ~(a & b);
10  assign nor_out = ~(a | b);
11  assign xor_out = a ^ b;
12  assign xnor_out = ~(a ^ b);
13  assign not_a = ~a;
14  assign not_b = ~b;
15
16 endmodule

```

Verilog code

testbench

```

1 `timescale 1ns / 1ps
2
3 module combinational_gates_tb;
4   reg a;
5   reg b;
6   wire and_out, or_out, nand_out, nor_out, xor_out, xnor_out, not_a, not_b;
7   combinational_gates uut (
8     .a(a),
9     .b(b),
10    .and_out(and_out),
11    .or_out(or_out),
12    .nand_out(nand_out),
13    .nor_out(nor_out),
14    .xor_out(xor_out),
15    .xnor_out(xnor_out),
16    .not_a(not_a),
17    .not_b(not_b)
18  );
19
20 initial begin
21   $dumpfile("combinational_gates.vcd");
22   $dumpvars(0, combinational_gates_tb);
23
24   $display("A B | AND OR NAND NOR XOR XNOR NOT_A NOT_B");
25   $display("-----");
26
27   a = 0; b = 0; #10;
28   $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);
29
30   a = 0; b = 1; #10;
31   $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);
32
33   a = 1; b = 0; #10;
34   $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);
35
36   a = 1; b = 1; #10;
37   $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);
38   $finish;
39 end
40 endmodule
41

```

GTKWave



A	B	AND	OR	NAND	NOR	XOR	XNOR	NOT_A	NOT_B
<hr/>									
0	0	0	1	1	0	1	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	0	0	0	1	0	0	0

Exercise - 2

$$i) Y = AB + CD$$

$$\Rightarrow \overline{AB} \cdot \overline{CD} = \overline{\text{NAND}(\text{NAND}(AB), \text{NAND}(CD))}$$

$$ii) Y = (\overline{ABC} + \overline{DE})F$$

$$= \overline{ABC + DE}F$$

$$= \overline{ABC + DE} + \overline{F}$$

$$= \overline{\overline{ABC} \cdot \overline{DE}} + \overline{F}$$

$$= (\overline{ABC} \cdot \overline{DE}) \cdot F$$

$$iii) Y = ((A+B)(CD+E))$$

$$D = \overline{A+B} + \overline{CD+E}$$

$$= (\overline{A} \cdot \overline{B}) + (\overline{C} \cdot \overline{D}) \cdot \overline{E}$$

$$= \overline{A} \cdot \overline{B} + \overline{C} \cdot \overline{D} \cdot \overline{E}$$

$$Y = \overline{A} \cdot \overline{B} + \overline{C} \cdot \overline{E} + \overline{D} \cdot \overline{E}$$

$$Y = \overline{A} \cdot \overline{B} \cdot \overline{C} \cdot \overline{E} \cdot \overline{D} \cdot \overline{E}$$

$$iv) F(A, B, C, D) = \prod (1, 3, 5, 7, 13, 15)$$

$$= \sum (6, 2, 4, 8, 9, 10, 11, 13, 14)$$

	CD	00	01	10	11	
AB	1	0	0	1	1	
00	1	0	1	2	2	
01	1	0	0	1	1	
11	1	1	1	1	1	
10	1	1	1	1	1	

$$D' + AB' = F$$

$$\overline{A} \cdot \overline{B} \cdot \overline{D} = F$$

	CD	00	01	10	11	
AB	1	0	0	X	X	
00	1	0	0	1	1	
01	X	0	0	1	1	
11	0	1	0	1	1	
10	1	0	0	X	X	

$$F = \sum (6, 8, 13, 14)$$

$$d = \sum 3, 4, 10$$

$$CD + BD + ABCD$$

testbench

```

1 `timescale 1ns / 1ps
2 module compound_circuits_tb;
3   reg a,b,c,d,e,f;
4   wire y1,y2,y3,y4,y5;
5   compound_circuits uut(a,b,c,d,e,f,y1,y2,y3,y4,y
6   5); initial begin
7     $dumpfile("compound_circuits.vcd");
8     $dumpvars(0, compound_circuits_tb);
9     for (integer i=0; i<64; i=i+1) begin
10       {a,b,c,d,e,f} = i;
11       #5;
12     end
13     $finish;
14   end
15 endmodule
16

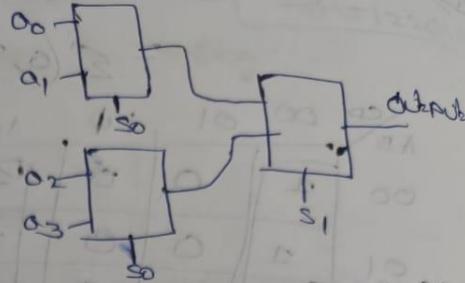
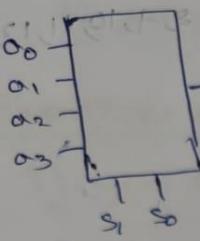
```

Verilog code

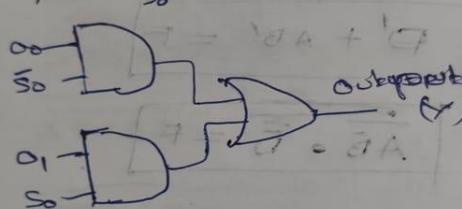
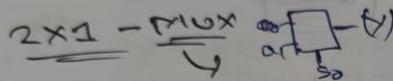
```
1 `timescale 1ns / 1ps
2 module compound_circuits(
3     input a, b, c, d, e, f,
4     output y1, y2, y3, y4, y5);
5
6     wire na, nb, nc, nd, ne;
7     nand(na, a, a);
8     nand(nb, b, b);
9     nand(nc, c, c);
10    nand(nd, d, d);
11    nand(ne, e, e);
12
13    wire w1_ab_n, w1_cd_n;
14    nand(w1_ab_n, 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

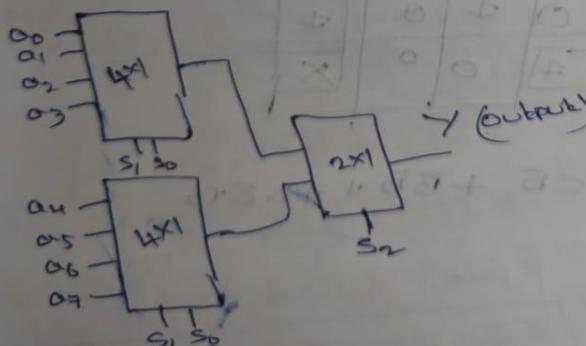


Exercise-3

4x1 MUX using 2x1 MUX



8x1 using 2(4x1) and 2x1

Verilog code

```

1 `timescale 1ns/1ps
2 module mux2_1 (
3     input a,
4     input b,
5     input sel,
6     output y
7 );
8     wire nsel, and0, and1;
9
10    not(nsel,sel);
11    and(and0,a,nsel);
12    and(and1,b,sel);
13    or(y, and0, and1);
14 endmodule
15
16 module mux4_1 (
17     input [3:0] d,
18     input [1:0] sel,
19     output y
20 );
21     wire low, high;
22     mux2_1 m0 (.a(d[0]), .b(d[1]), .sel(sel[0]), .y(low));
23     mux2_1 m1 (.a(d[2]), .b(d[3]), .sel(sel[0]), .y(high));
24     mux2_1 m2 (.a(low), .b(high), .sel(sel[1]), .y(y));
25 endmodule
26
27 module mux8_1 (
28     input [7:0] d,
29     input [2:0] sel,
30     output y
31 );
32     wire low4, high4;
33     mux4_1 mlow (.d(d[3:0]), .sel(sel[1:0]), .y(low4));
34     mux4_1 mhigh(.d(d[7:4]), .sel(sel[1:0]), .y(high4));
35     mux2_1 mtop (.a(low4), .b(high4), .sel(sel[2]), .y(y));
36 endmodule

```

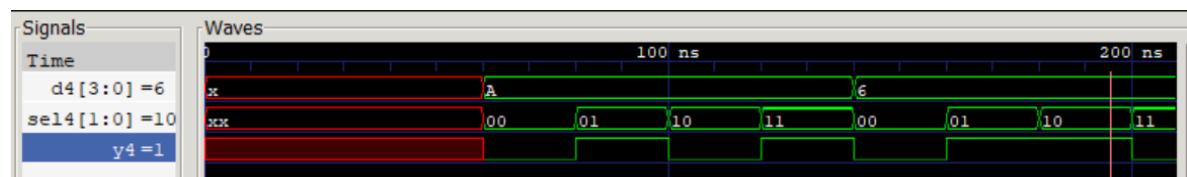
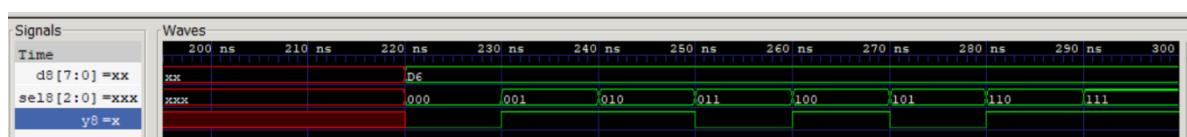
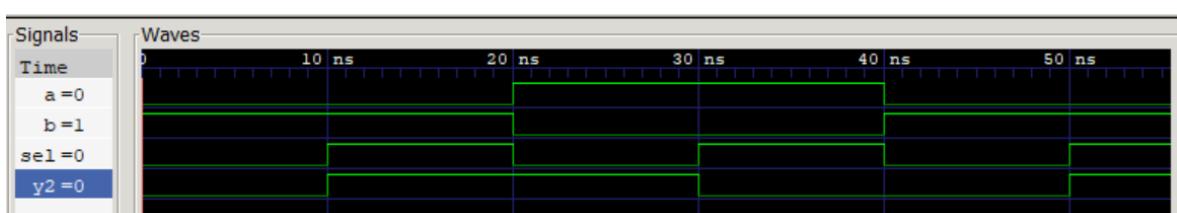
```

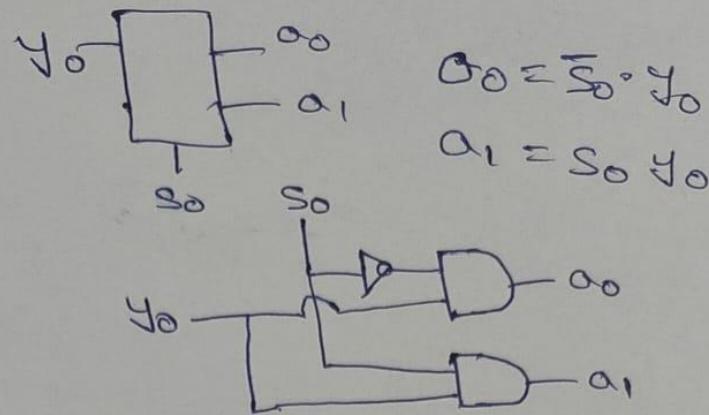
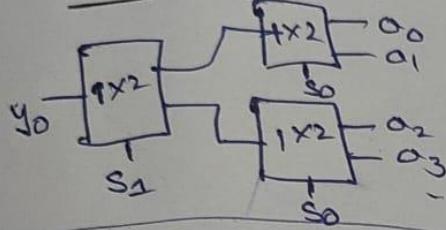
1 `timescale 1ns/1ps
2 module mux_tb;
3
4   reg a, b, sel;
5   wire y2;
6   reg [3:0] d4;
7   reg [1:0] sel4;
8   wire y4;
9   reg [7:0] d8;
10  reg [2:0] sel8;
11  wire y8;
12
13  mux2_1 uut2 (
14    .a(a),
15    .b(b),
16    .sel(sel),
17    .y(y2));
18  mux4_1 uut4 (
19    .d(d4),
20    .sel(sel4),
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) begin
56      n
57        sel8 = i;
58      #10;
59    end
60  end
61 endmodule
62

```

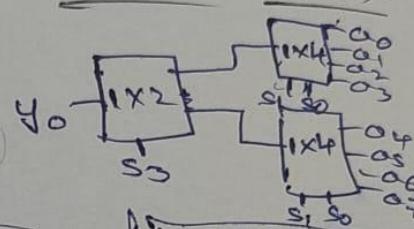
testbench

GTKWave



~~1x2 DEMUX~~~~1x4 DEMUX~~

1x4 Using 1x2 DEMUX

~~1x8 - DEMUX~~

1x8 DEMUX using
(1x4 and 1x2)

Verilog code

```

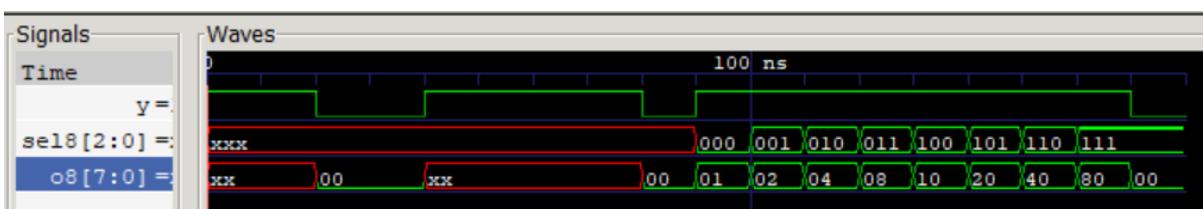
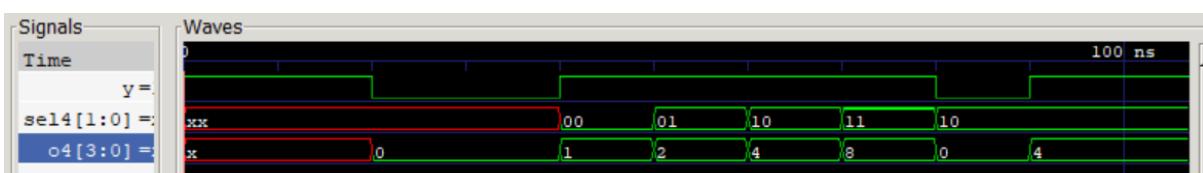
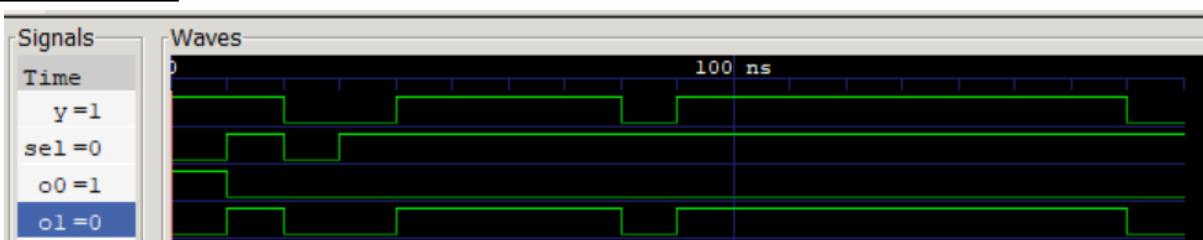
1 `timescale 1ns/1ps
2
3 module demux1_2 (
4     input y,
5     input sel,
6     output o0, o1
7 );
8     wire nsel;
9     not(nsel, sel);
10    and(o0, y, nsel);
11    and(o1, y, sel);
12 endmodule
13
14 module demux1_4 (
15     input y,
16     input [1:0] sel,
17     output [3:0] o
18 );
19     wire p0, p1;
20     demux1_2 d0 (.y(y), .sel(sel[1]), .o0(p0), .o1(p1));
21     demux1_2 d1 (.y(p0), .sel(sel[0]), .o0(o[0]), .o1(o[1]));
22     demux1_2 d2 (.y(p1), .sel(sel[0]), .o0(o[2]), .o1(o[3]));
23 endmodule
24
25 module demux1_8 (
26     input y,
27     input [2:0] sel,
28     output [7:0] o
29 );
30     wire low_en, high_en;
31     demux1_2 top (.y(y), .sel(sel[2]), .o0(low_en), .o1(high_en));
32     n);demux1_4 low (.y(low_en), .sel(sel[1:0]), .o(o[3:0]));
33     demux1_4 high (.y(high_en), .sel(sel[1:0]), .o(o[7:4]));
34 endmodule
35
36

```

testbench

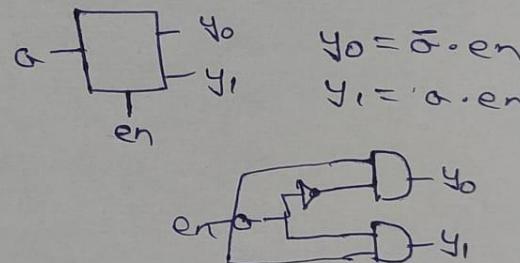
```
1  module demux_tb;
2    reg y;
3    reg sel;
4    wire o0, o1;
5    reg [1:0] sel4;
6    wire [3:0] o4;
7    reg [2:0] sel8;
8    wire [7:0] o8;
9
10   demux1_2 uut2 (.y(y), .sel(sel), .o0(o0), .o1(o1));
11   demux1_4 uut4 (.y(y), .sel(sel4), .o(o4));
12   demux1_8 uut8 (.y(y), .sel(sel8), .o(o8));
13
14   initial begin
15     $dumpfile("demux_wave.vcd");
16     $dumpvars(0, demux_tb);
17   end
18
19   initial begin
20     y = 1;
21     sel = 0; #10;
22     sel = 1; #10;
23     y = 0;
24     sel = 0; #10;
25     sel = 1; #10;
26
27     y = 1;
28     sel4 = 2'b00; #10;
29     sel4 = 2'b01; #10;
30     sel4 = 2'b10; #10;
31     sel4 = 2'b11; #10;
32     y = 0;
33     sel4 = 2'b10; #10;
34
35     y = 1;
36     sel8 = 3'b000; #10;
37     sel8 = 3'b001; #10;
38     sel8 = 3'b010; #10;
39     sel8 = 3'b011; #10;
40     sel8 = 3'b100; #10;
41     sel8 = 3'b101; #10;
42     sel8 = 3'b110; #10;
43     sel8 = 3'b111; #10;
44
45     y = 0; #10;
46     $finish;
47   end
48 endmodule
```

GTKWave

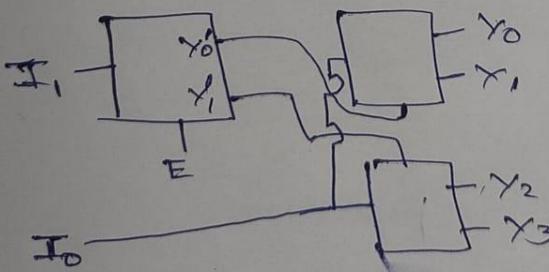


DECODER

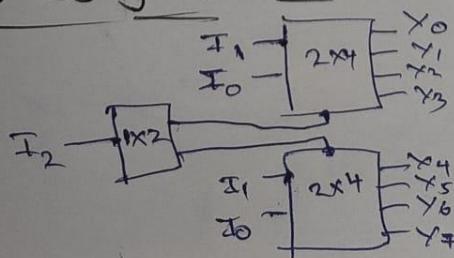
1x2 Decoders



2x4 using 1x2 decoders



3x8 using 2x4 decoders



testbench

```

1 `timescale 1ns/1ps
2 module dec_tb;
3   reg a;
4   reg en;
5   wire [1:0] y2;
6   reg [1:0] a2;
7   wire [3:0] y4;
8   reg [2:0] a3;
9   wire [7:0] y8;
10  dec2_1 D2 (.a(a), .en(en), .y(y2));
11  dec2_4 D4 (.a(a2), .en(en), .y(y
12  4));dec3_8 D8 (.a(a3), .en(en), .y(y
13  8));initial begin
14    $dumpfile("decoder_wave.vcd");
15    $dumpvars(0, dec_tb);
16  end
17
18  initial begin
19    en = 1; a = 0; #10;
20    a = 1; #10;
21    en = 0; a = 1; #10;
22
23    en = 1;
24    a2 = 2'b00; #10;
25    a2 = 2'b01; #10;
26    a2 = 2'b10; #10;
27    a2 = 2'b11; #10;
28    en = 0; #10;
29
30    en = 1;
31    a3 = 3'b000; #10;
32    a3 = 3'b001; #10;
33    a3 = 3'b010; #10;
34    a3 = 3'b011; #10;
35    a3 = 3'b100; #10;
36    a3 = 3'b101; #10;
37    a3 = 3'b110; #10;
38    a3 = 3'b111; #10;
39    en = 0; #10;
40    $finish;
41  end
42 endmodule

```

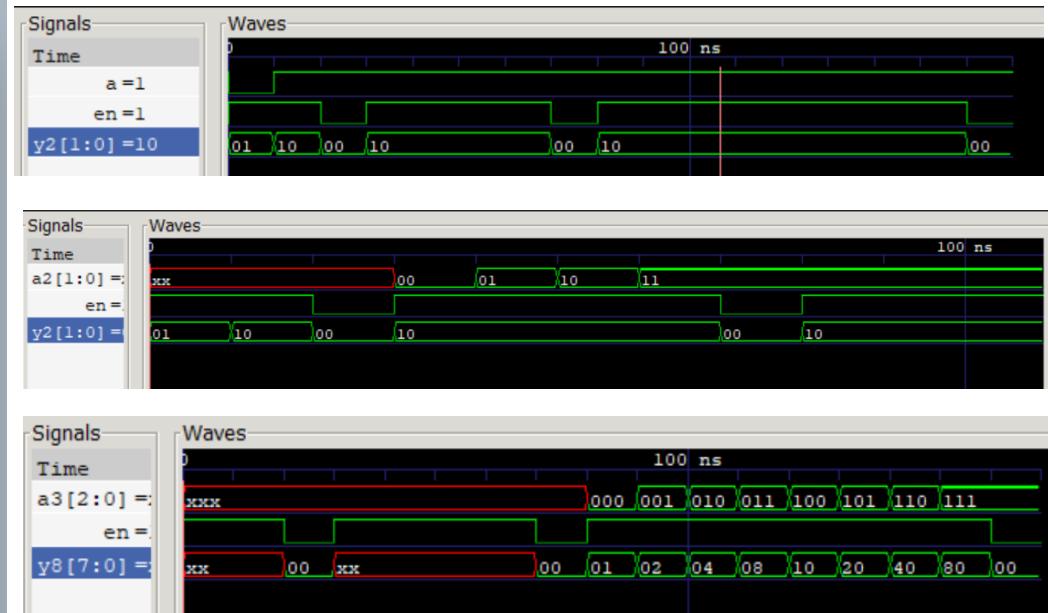
Verilog code

```

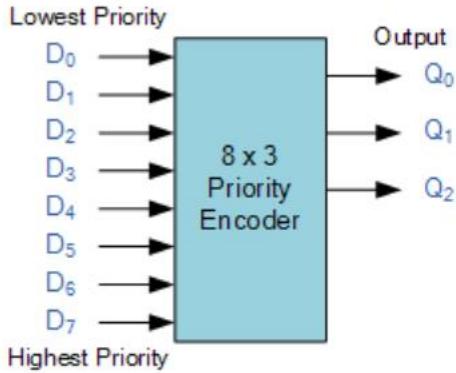
1 `timescale 1ns/1ps
2 module dec2_1 (
3   input a,
4   input en,
5   output [1:0] y
6 );
7   wire na;
8   not(na, a);
9   and(y[0], na, en);
10  and(y[1], a, en);
11 endmodule
12
13 module dec2_4 (
14   input [1:0] a,
15   input en,
16   output [3:0] y
17 );
18   wire [1:0] e;
19   dec2_1 d0 (.a(a[1]), .en(en), .y(e));
20   dec2_1 low (.a(a[0]), .en(e[0]), .y(y[1:0]));
21   dec2_1 high (.a(a[0]), .en(e[1]), .y(y[3:2]));
22 endmodule
23
24 module dec3_8 (
25   input [2:0] a,
26   input en,
27   output [7:0] y
28 );
29   wire [1:0] e;
30   dec2_1 top (.a(a[2]), .en(en), .y(e));
31   dec2_4 low (.a(a[1:0]), .en(e[0]), .y(y[3:
32  0]));
33   dec2_4 high (.a(a[1:0]), .en(e[1]), .y(y[7:
34  4]));
35 endmodule

```

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	0	1	0	0
0	0	0	0	0	0	0	1	x	0	0
0	0	0	0	0	1	x	x	x	0	1
0	0	0	0	1	x	x	x	x	0	1
0	0	0	1	x	x	x	x	x	1	0
0	0	1	x	x	x	x	x	x	1	0
0	1	x	x	x	x	x	x	x	1	1
1	x	x	x	x	x	x	x	x	1	1

X = don't care

Verilog code

```

1 `timescale 1ns/1ps
2 module priority_encoder_8x3 (
3     input [7:0] I,
4     input en,
5     output [2:0] Y,
6     output valid);
7     wire any;
8     wire b2, b1, b0;
9     or(any, I[0], I[1], I[2], I[3], I[4], I[5], I[6], I
10 [7]); or(b2, I[7], I[6], I[5], I[4]);
11     or(b1, I[7], I[6], I[3], I[2]);
12     or(b0, I[7], I[5], I[3], I[1]);
13     and(Y[2], b2, en);
14     and(Y[1], b1, en);
15     and(Y[0], b0, en);
16     and(valid, any, en);
17 endmodule

```

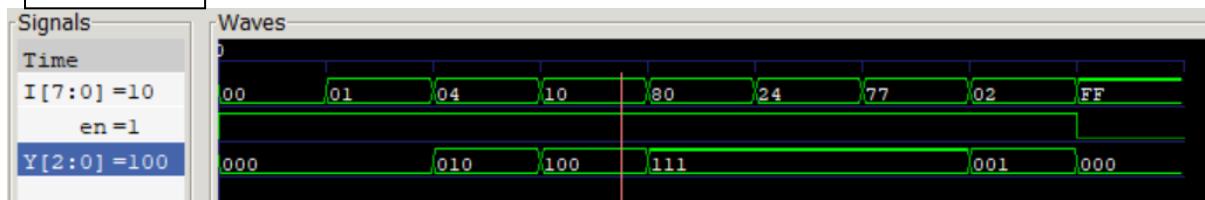
testbench

```

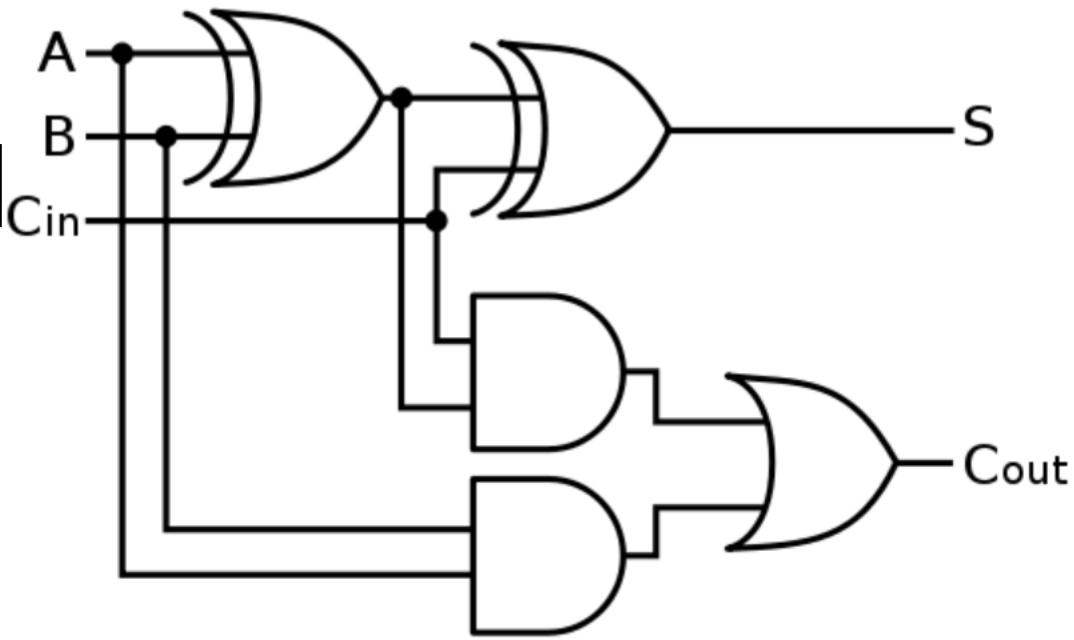
1 `timescale 1ns/1ps
2 module priority_encoder_tb;
3     reg [7:0] I;
4     reg en;
5     wire [2:0] Y;
6     wire valid;
7     priority_encoder_8x3 dut (.I(I), .en(en), .Y(Y), .valid(valid));
8     initial begin
9         $dumpfile("priority_encoder_wave.vcd");
10        $dumpvars(0, priority_encoder_tb);
11    end
12
13    initial begin
14        en = 1;
15        I = 8'b00000000; #10;
16        I = 8'b00000001; #10;
17        I = 8'b00000100; #10;
18        I = 8'b00010000; #10;
19        I = 8'b10000000; #10;
20        I = 8'b00100100; #10;
21        I = 8'b01110111; #10;
22        I = 8'b00000010; #10;
23        en = 0;
24        I = 8'b11111111; #10;
25        $finish;
26    end
27 endmodule

```

GTKWave



1bit Full Adder



Verilog code

```

1 `timescale 1ns / 1ps
2 module full_adder(
3     input A, B, Cin,
4     output Sum, Cout
5 );
6     assign Sum=A^B^Cin;
7     assign Cout=(A&B)|((A^B)&Cin);
8 endmodule
9

```

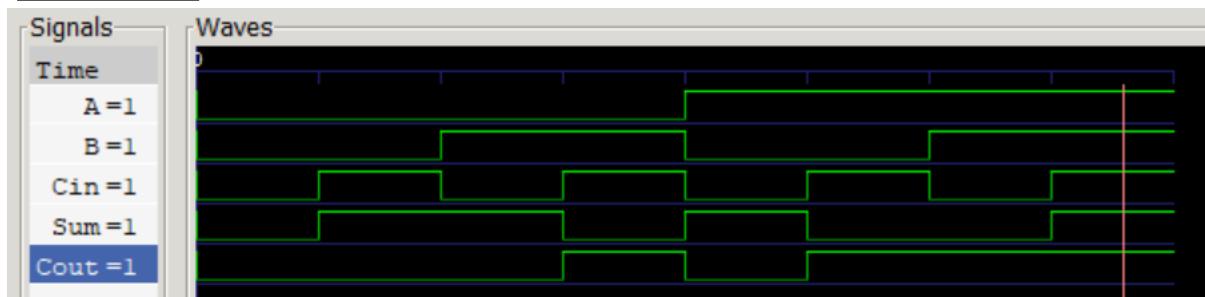
testbench

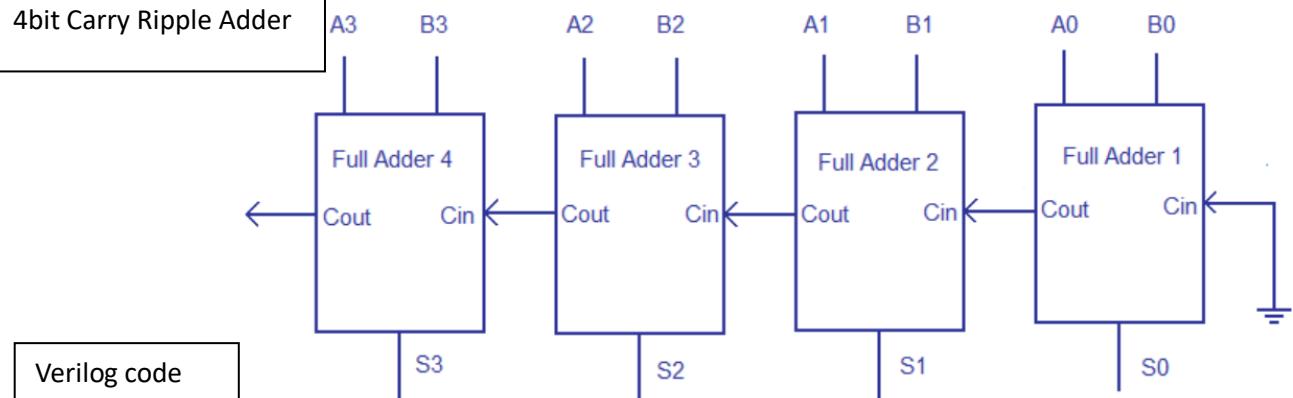
```

1 `timescale 1ns / 1ps
2 module full_adder_tb;
3     reg A, B, Cin;
4     wire Sum, Cout;
5     full_adder uut (
6         .A(A),
7         .B(B),
8         .Cin(Cin),
9         .Sum(Sum),
10        .Cout(Cout)
11    );
12 initial begin
13     $dumpfile("full_adder.vcd");
14     $dumpvars(0, full_adder_tb);
15     A=0; B=0; Cin=0; #10
16     A=0; B=0; Cin=1; #10
17     A=0; B=1; Cin=0; #10
18     A=0; B=1; Cin=1; #10
19     A=1; B=0; Cin=0; #10
20     A=1; B=0; Cin=1; #10
21     A=1; B=1; Cin=0; #10
22     A=1; B=1; Cin=1; #10
23     $finish;
24 end
25 endmodule

```

GTKWave





Verilog code

```

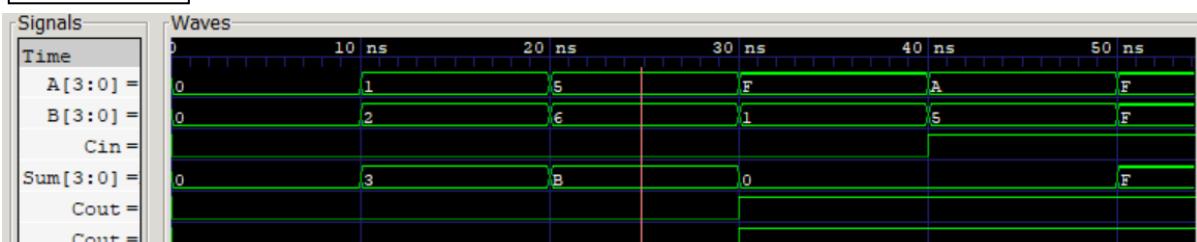
1 `timescale 1ns / 1ps
2 module full_adder(
3     input A, B, Cin,
4     output Sum, Cout);
5     assign Sum = A ^ B ^ Cin;
6     assign Cout = (A & B) | ((A ^ B) & Cin);
7 endmodule
8
9 module ripple_carry_adder_4bit(
10     input [3:0] A, B,
11     input Cin,
12     output [3:0] Sum,
13     output Cout);
14     wire C1, C2, C3;
15     full_adder FA0 (.A(A[0]), .B(B[0]), .Cin(Cin), .Sum(Sum[0]), .Cout(C1));
16     full_adder FA1 (.A(A[1]), .B(B[1]), .Cin(C1), .Sum(Sum[1]), .Cout(C2));
17     full_adder FA2 (.A(A[2]), .B(B[2]), .Cin(C2), .Sum(Sum[2]), .Cout(C3));
18     full_adder FA3 (.A(A[3]), .B(B[3]), .Cin(C3), .Sum(Sum[3]), .Cout(Cou
19     t);)module
20
21
  
```

testbench

```

1 module ripple_carry_adder_4bit_tb;
2     reg [3:0] A, B;
3     reg Cin;
4     wire [3:0] Sum;
5     wire Cout;
6     ripple_carry_adder_4bit uut (
7         .A(A),
8         .B(B),
9         .Cin(Cin),
10        .Sum(Sum),
11        .Cout(Cout));
12 initial begin
13     $dumpfile("ripple_carry_adder_full.vcd");
14     $dumpvars(0, ripple_carry_adder_4bit_tb);
15     A=4'b0000; B=4'b0000; Cin=0; #10
16     A=4'b0001; B=4'b0010; Cin=0; #10
17     A=4'b0101; B=4'b0110; Cin=0; #10
18     A=4'b1111; B=4'b0001; Cin=0; #10
19     A=4'b1010; B=4'b0101; Cin=1; #10
20     A=4'b1111; B=4'b1111; Cin=1; #10
21     $finish;
22 end
23 endmodule
  
```

GTKWave



16 Bit Carry Ripple Adder

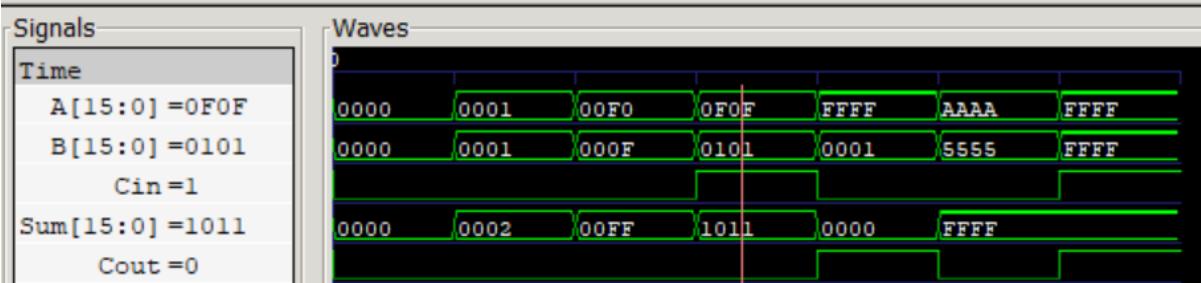
Verilog code

```
1 `timescale 1ns / 1ps
2 module full_adder(
3     input A, B, Cin,
4     output Sum, Cout);
5     assign Sum = A ^ B ^ Cin;
6     assign Cout = (A & B) | ((A ^ B) & Cin);
7 endmodule
8
9 module ripple_carry_adder_4bit(
10    input [3:0] A, B,
11    input Cin,
12    output [3:0] Sum,
13    output Cout);
14    wire C1, C2, C3;
15    full_adder FA0 (.A(A[0]), .B(B[0]), .Cin(Cin), .Sum(Sum[0]), .Cout(C1));
16    full_adder FA1 (.A(A[1]), .B(B[1]), .Cin(C1), .Sum(Sum[1]), .Cout(C2));
17    full_adder FA2 (.A(A[2]), .B(B[2]), .Cin(C2), .Sum(Sum[2]), .Cout(C3));
18    full_adder FA3 (.A(A[3]), .B(B[3]), .Cin(C3), .Sum(Sum[3]), .Cout(Cout));
19 endmodule
20
21 module ripple_carry_adder_16bit(
22    input [15:0] A, B,
23    input Cin,
24    output [15:0] Sum,
25    output Cout);
26    wire C1, C2, C3;
27    ripple_carry_adder_4bit RCA0 (.A(A[3:0]), .B(B[3:0]), .Cin(Cin), .Sum(Sum[3:0]), .Cout(C1));
28    ripple_carry_adder_4bit RCA1 (.A(A[7:4]), .B(B[7:4]), .Cin(C1), .Sum(Sum[7:4]), .Cout(C2));
29    ripple_carry_adder_4bit RCA2 (.A(A[11:8]), .B(B[11:8]), .Cin(C2), .Sum(Sum[11:8]), .Cout(C3));
30    ripple_carry_adder_4bit RCA3 (.A(A[15:12]), .B(B[15:12]), .Cin(C3), .Sum(Sum[15:12]), .Cout(Cou
31 endmodule
32
```

testbench

```
1 module ripple_carry_adder_16bit_tb;
2     reg [15:0] A, B;
3     reg Cin;
4     wire [15:0] Sum;
5     wire Cout;
6     ripple_carry_adder_16bit uut (
7         .A(A),
8         .B(B),
9         .Cin(Cin),
10        .Sum(Sum),
11        .Cout(Cout));
12     initial begin
13         $dumpfile("ripple_carry_adder_16bit.vcd");
14         $dumpvars(0, ripple_carry_adder_16bit_tb);
15         A = 16'h0000; B = 16'h0000; Cin = 0; #10
16         A = 16'h0001; B = 16'h0001; Cin = 0; #10
17         A = 16'h00F0; B = 16'h000F; Cin = 0; #10
18         A = 16'h0F0F; B = 16'h0101; Cin = 1; #10
19         A = 16'hFFFF; B = 16'h0001; Cin = 0; #10
20         A = 16'hAAAA; B = 16'h5555; Cin = 0; #10
21         A = 16'hFFFF; B = 16'hFFFF; Cin = 1; #10
22         $finish;
23     end
24 endmodule
25
```

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

```

1 `timescale 1ns / 1ps
2 module carry_lookahead_adder_4bit(
3     input [3:0] A, B,
4     input Cin,
5     output [3:0] Sum,
6     output Cout);
7     wire [3:0] G, P;
8     wire [4:0] C;
9     assign C[0] = Cin;
10    assign G = A & B;
11    assign P = A ^ B;
12    assign C[1] = G[0] | (P[0] & C[0]);
13    assign C[2] = G[1] | (P[1] & G[0]) | (P[1] & P[0] & C[0]);
14    assign C[3] = G[2] | (P[2] & G[1]) | (P[2] & P[1] & G[0]) | (P[2] & P[1] & P[0] & C
15 [0]);
16    assign C[4] = G[3] | (P[3] & G[2]) | (P[3] & P[2] & G[1]) |
17        (P[3] & P[2] & P[1] & G[0]) | (P[3] & P[2] & P[1] & P[0] & C[0]);
18    assign Sum = P ^ C[3:0];
19    assign Cout = C[4];
20 endmodule

```

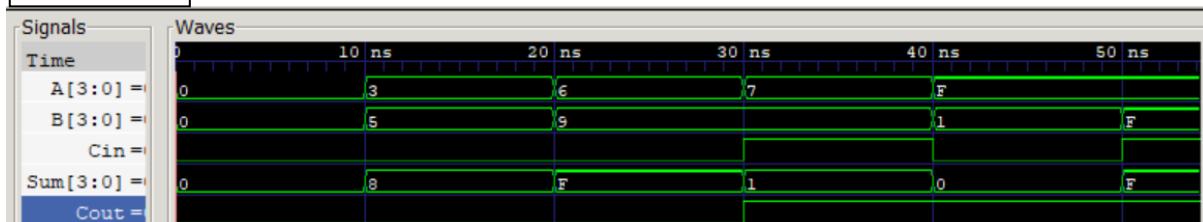
testbench

```

1 `timescale 1ns / 1ps
2 module carry_lookahead_adder_4bit_tb;
3     reg [3:0] A, B;
4     reg Cin;
5     wire [3:0] Sum;
6     wire Cout;
7     carry_lookahead_adder_4bit uut (
8         .A(A),
9         .B(B),
10        .Cin(cin),
11        .Sum(Sum),
12        .Cout(Cout));
13 initial begin
14     $dumpfile("carry_lookahead_adder_4bit.vcd");
15     $dumpvars(0, carry_lookahead_adder_4bit_tb);
16     A=4'b0000; B=4'b0000; Cin=0; #10
17     A=4'b0011; B=4'b0101; Cin=0; #10
18     A=4'b0110; B=4'b1001; Cin=0; #10
19     A=4'b1111; B=4'b1001; Cin=1; #10
20     A=4'b1111; B=4'b0001; Cin=0; #10
21     A=4'b1111; B=4'b1111; Cin=1; #10
22     $finish;
23 end
24 endmodule

```

GTKWave



16 bit Carry Look-Ahead

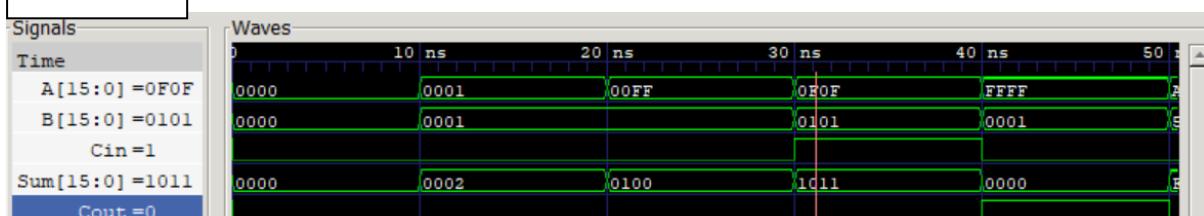
Verilog code

```
1 `timescale 1ns / 1ps
2 module carry_lookahead_adder_4bit(
3     input [3:0] A, B,
4     input Cin,
5     output [3:0] Sum,
6     output Cout,
7     output G_group, P_group );
8     wire [3:0] G, P;
9     wire [4:0] C;
10    assign C[0] = Cin;
11    assign G = A & B;
12    assign P = A ^ B;
13    assign C[1] = G[0] | (P[0] & C[0]);
14    assign C[2] = G[1] | (P[1] & G[0]) | (P[1] & P[0] & C[0]);
15    assign C[3] = G[2] | (P[2] & G[1]) | (P[2] & P[1] & G[0]) | (P[2] & P[1] & P[0] & C[0]);
16    assign C[4] = G[3] | (P[3] & G[2]) | (P[3] & P[2] & G[1]) |
17        (P[3] & P[2] & P[1] & G[0]) | (P[3] & P[2] & P[1] & P[0] & C[0]);
18    assign Sum = P ^ C[3:0];
19    assign Cout = C[4];
20    assign G_group = G[3] | (P[3] & G[2]) |
21        (P[3] & P[2] & G[1]) |
22        (P[3] & P[2] & P[1] & G[0]);
23    assign P_group = P[3] & P[2] & P[1] & P[0];
24 endmodule
25
26 module carry_lookahead_adder_16bit(
27     input [15:0] A, B,
28     input Cin,
29     output [15:0] Sum,
30     output Cout);
31     wire [3:0] C;
32     wire [3:0] G, P;
33     assign C[0] = Cin;
34     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]));
35     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]));
36     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]));
37     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
38 [3]));
39     assign C[1] = G[0] | (P[0] & C[0]);
40     assign C[2] = G[1] | (P[1] & G[0]) | (P[1] & P[0] & C[0]);
41     assign C[3] = G[2] | (P[2] & G[1]) | (P[2] & P[1] & G[0]) | (P[2] & P[1] & P[0] & C[0]);
42 endmodule
43
```

testbench

```
1 module carry_lookahead_adder_16bit_tb;
2     reg [15:0] A, B;
3     reg Cin;
4     wire [15:0] Sum;
5     wire Cout;
6     carry_lookahead_adder_16bit uut (
7         .A(A),
8         .B(B),
9         .Cin(Cin),
10        .Sum(Sum),
11        .Cout(Cout));
12 initial begin
13     $dumpfile("carry_lookahead_adder_16bit.vcd");
14     $dumpvars(0, carry_lookahead_adder_16bit_tb);
15     A = 16'h0000; B = 16'h0000; Cin = 0; #10
16     A = 16'h0001; B = 16'h0001; Cin = 0; #10
17     A = 16'h00FF; B = 16'h0001; Cin = 0; #10
18     A = 16'h0F0F; B = 16'h0101; Cin = 1; #10
19     A = 16'hFFFF; B = 16'h0001; Cin = 0; #10
20     A = 16'hAAAA; B = 16'h5555; Cin = 0; #10
21     A = 16'hFFFF; B = 16'hFFFF; Cin = 1; #10
22     $finish;
23 end
24 endmodule
25
```

GTKWave



Verilog code

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

testbench

```

1 module flipflops_all_tb;
2   reg D, J, K, R, S, T, clk;
3   wire Qd, Qjk, Qrs, Qt;
4   d_flipflop DFF (.D(D), .clk(clk), .Q(Qd));
5   jk_flipflop JKFF (.J(J), .K(K), .clk(clk), .Q(Qjk));
6   rs_flipflop RSFF (.R(R), .S(S), .clk(clk), .Q(Qrs));
7   t_flipflop TFF (.T(T), .clk(clk), .Q(Qt));
8   initial begin
9     clk = 0;
10    forever #5 clk = ~clk;
11  end
12  initial begin
13    $dumpfile("flipflops_all.vcd");
14    $dumpvars(0, flipflops_all_tb);
15    D=0; J=0; K=0; R=0; S=0; T=0;
16    #10;
17    D=1; #10;
18    D=0; #10;
19    D=1; #10;
20    J=1; K=0; #10;
21    J=0; K=1; #10;
22    J=1; K=1; #20;
23    R=0; S=1; #10;
24    R=1; S=0; #10;
25    R=0; S=0; #10;
26    R=1; S=1; #10;
27    T=1; #40;
28    T=0; #10;
29    $finish;
30  end
31  initial begin
32    $monitor("%4t | %b %b | %b %b %b | %b %b %b | %b %b",
33             $time, D, Qd, J, K, Qjk, R, S, Qrs, T, Qt);
34  end
35 endmodule

```

```

1 `timescale 1ns / 1ps
2 module d_flipflop(
3   input D,
4   input clk,
5   output reg Q);
6   always @(posedge clk)
7     Q <= D;
8 endmodule
9
10 module jk_flipflop(
11   input J, K, clk,
12   output reg Q);
13   always @(posedge clk) begin
14     n      case ({J, K})
15       2'b00: Q <= Q;
16       2'b01: Q <= 0;
17       2'b10: Q <= 1;
18       2'b11: Q <= ~Q;
19     endcase
20   end
21 endmodule
22
23 module rs_flipflop(
24   input R, S, clk,
25   output reg Q);
26   always @(posedge clk) begin
27     n      case ({R, S})
28       2'b00: Q <= Q;
29       2'b01: Q <= 1;
30       2'b10: Q <= 0;
31       2'b11: Q <= 1'bxx;
32     endcase
33   end
34 endmodule
35
36 module t_flipflop(
37   input T, clk,
38   output reg Q);
39   always @(posedge clk) begin
40     n      if (T)
41       Q <= ~Q;
42     else
43       Q <= Q;
44   end
45 endmodule

```

GTKWave



4 bit universal shift register

S1 S0 Operation

0 0 Hold

Description

. 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

```

1 `timescale 1ns / 1ps
2 module universal_shift_register_tb;
3   reg clk;
4   reg [1:0] S;
5   reg [3:0] P;
6   reg SL_in, SR_in;
7   wire [3:0] Q;
8   universal_shift_register uut (
9     .clk(clk),
10    .S(S),
11    .P(P),
12    .SL_in(SL_in),
13    .SR_in(SR_in),
14    .Q(Q));
15 initial begin
16   clk = 0;
17   forever #5 clk = ~clk;
18 end
19 initial begin
20   $dumpfile("universal_shift_register.vcd");
21   $dumpvars(0, universal_shift_register_tb);
22   S = 2'b00; P = 4'b0000; SL_in = 0; SR_in =
23 0; #10 S = 2'b11; P = 4'b1010;
24  #10;
25  #10 S = 2'b00;
26  SR_in = 1; S = 2'b01; #10;
27  SR_in = 0; #10;
28  SL_in = 1; S = 2'b10; #10;
29  SL_in = 0; #10;
30  S = 2'b11; P = 4'b1100; #10;
31  S = 2'b00; #10;
32  $finish;
33 end
34 initial begin
35   $monitor("%4t | %b | %b | %b %b | %b",
36             $time, S, P, SL_in, SR_in, Q);
37 end
38 endmodule
39

```

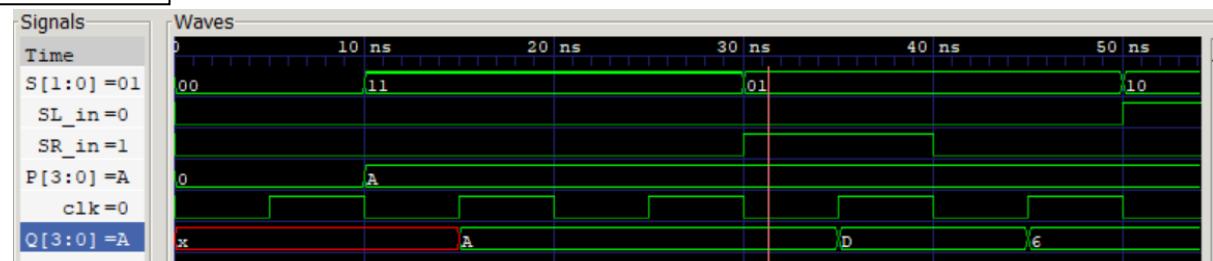
Verilog code

```

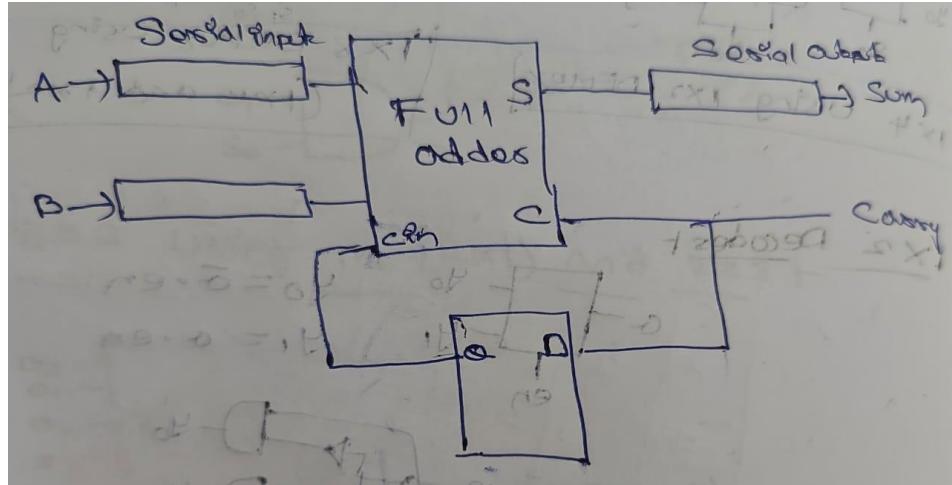
1 `timescale 1ns / 1ps
2 module universal_shift_register (
3   input clk,
4   input [1:0] S,
5   input [3:0] P,
6   input SL_in, SR_in,
7   output reg [3:0] Q);
8   always @(posedge clk) begin
9     case (S)
10       2'b00: Q <= Q;
11       2'b01: Q <= {SR_in, Q[3:
12 1]}; 2'b10: Q <= {Q[2:0], SL_i
13 n}; 2'b11: Q <= P;
14       default: Q <= Q;
15   endcase
16 end
17 endmodule

```

GTKWave



3 bit Serial Adder



Verilog code

```

1 `timescale 1ns / 1ps
2 module serial_adder(
3     input [2:0] A, B,
4     output reg [2:0] Sum,
5     output reg Cout);
6     integer i;
7     reg carry;
8     always @(*) begin
9         carry = 0;
10        for (i = 0; i < 3; i = i + 1) begin
11            {carry, Sum[i]} = A[i] + B[i] + carry;
12        end
13        Cout = carry;
14    end
15 endmodule
16

```

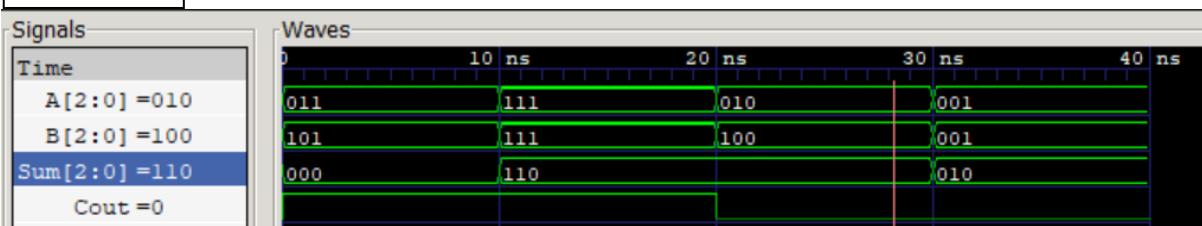
testbench

```

1 `timescale 1ns / 1ps
2 module serial_adder_tb;
3     reg [2:0] A, B;
4     wire [2:0] Sum;
5     wire Cout;
6     serial_adder uut (.A(A), .B(B), .Sum(Sum), .Cout(Cout));
7     initial begin
8         $dumpfile("serial_adder.vcd");
9         $dumpvars(0, serial_adder_tb);
10        A = 3'b011; B = 3'b101; #10;
11        A = 3'b111; B = 3'b111; #10;
12        A = 3'b010; B = 3'b100; #10;
13        A = 3'b001; B = 3'b001; #10;
14        $finish;
15    end
16 endmodule
17

```

GTKWave



4bit synchronous Binary Counter

Verilog code

```

1 `timescale 1ns / 1ps
2 module sync_counter_4bit(
3     input clk,
4     input enable,
5     output reg [3:0] Q);
6     initial Q = 4'b0000;
7     always @(posedge clk) begin
8         if (enable)
9             Q <= Q + 1;
10    end
11 endmodule

```

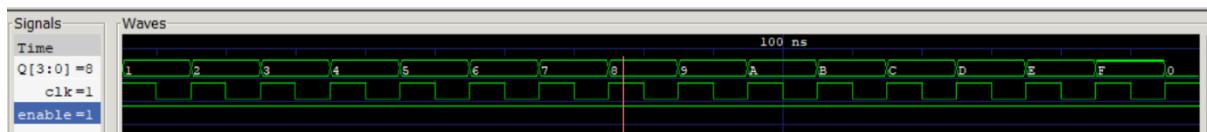
testbench

```

1 `timescale 1ns / 1ps
2 module sync_counter_4bit_tb;
3     reg clk, enable;
4     wire [3:0] Q;
5     sync_counter_4bit uut (.clk(clk), .enable(enable), .Q
6     (Q));
7     initial begin
8         clk = 0;
9         forever #5 clk = ~clk;
10    end
11    initial begin
12        $dumpfile("sync_counter_4bit.vcd");
13        $dumpvars(0, sync_counter_4bit_tb);
14        enable = 1; #160;
15        $finish;
16    end
17    initial $monitor("Time=%0t Count=%b", $time, Q);
18 endmodule

```

GTKWave



4bit up-down binary counter

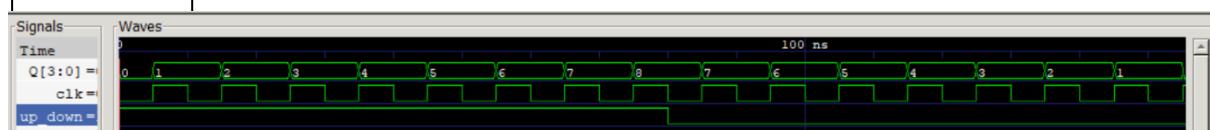
Verilog code

```
1 `timescale 1ns / 1ps
2 module updown_counter_4bit(
3     input clk,
4     input up_down,
5     output reg [3:0] Q);
6     initial Q = 4'b0000;
7     always @(posedge clk) begin
8         if (up_down)
9             Q <= Q + 1;
10        else
11            Q <= Q - 1;
12    end
13 endmodule
14
```

testbench

```
1 `timescale 1ns / 1ps
2 module updown_counter_4bit_tb;
3     reg clk, up_down;
4     wire [3:0] Q;
5     updown_counter_4bit uut (.clk(clk), .up_down(up_down), .Q
6     (Q));
7     initial begin
8         clk = 0;
9         forever #5 clk = ~clk;
10    end
11    initial begin
12        $dumpfile("updown_counter_4bit.vcd");
13        $dumpvars(0, updown_counter_4bit_tb);
14        up_down = 1; #80;
15        up_down = 0; #80;
16        $finish;
17    end
18 endmodule
```

GTKWave



4bit Ripple counter

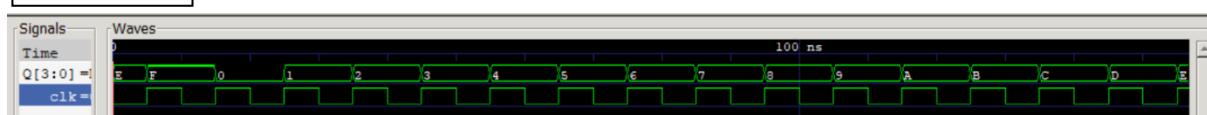
Verilog code

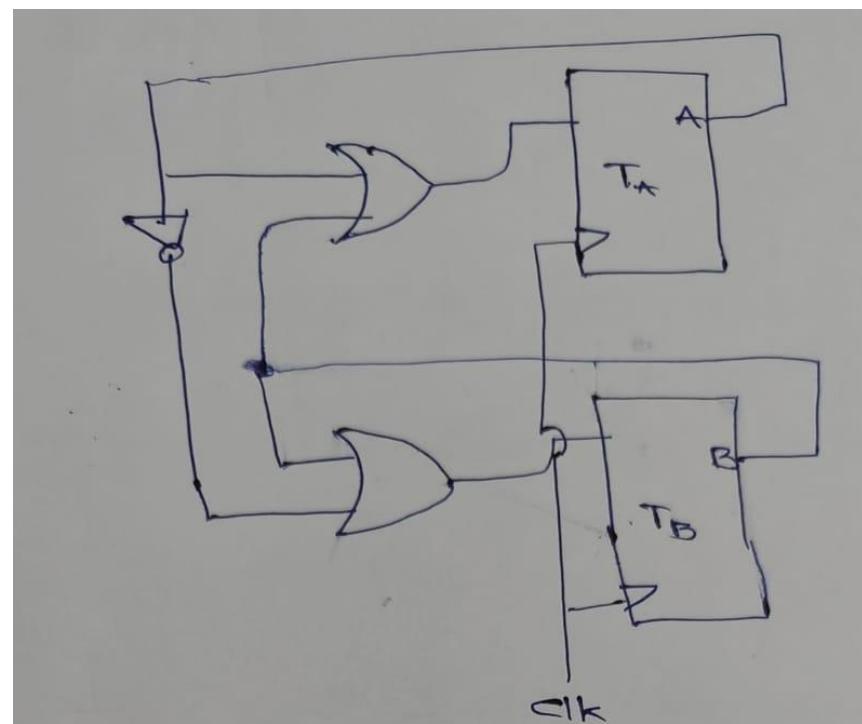
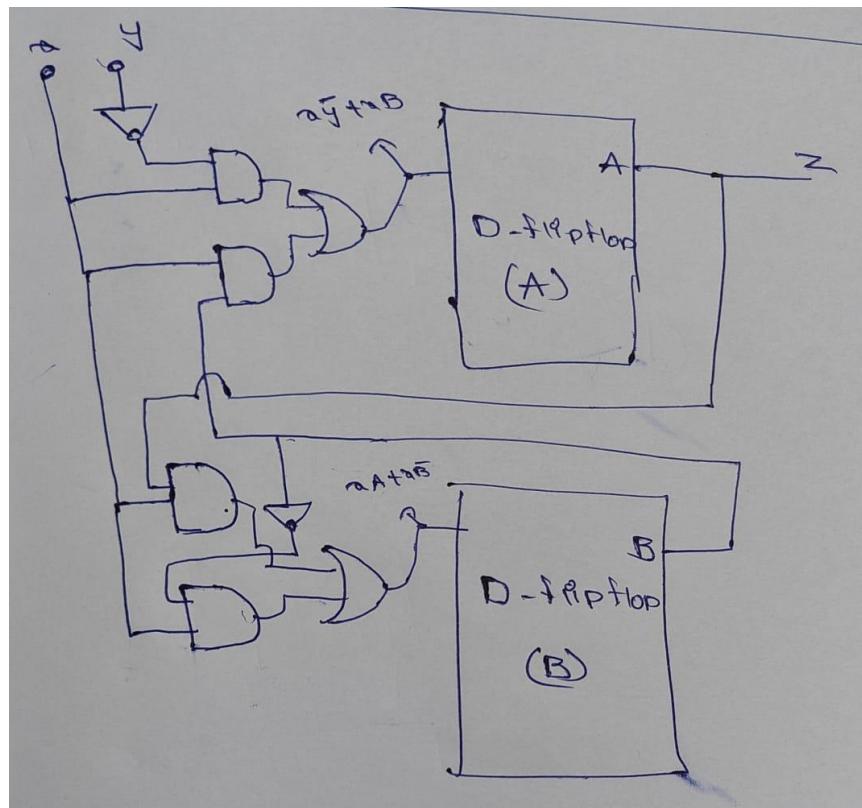
```
1 `timescale 1ns / 1ps
2 module ripple_counter_4bit(
3     input clk,
4     output [3:0] Q);
5     reg [3:0] count = 0;
6     always @(posedge clk)
7         count[0] <= ~count
8     [0];always @(negedge count
9     [0])    count[1] <= ~count
10    [1];always @(negedge count
11    [1])    count[2] <= ~count
12    [2];always @(negedge count
13    [2])    count[3] <= ~count
14    [3];assign Q = count;
15 endmodule
16
```

testbench

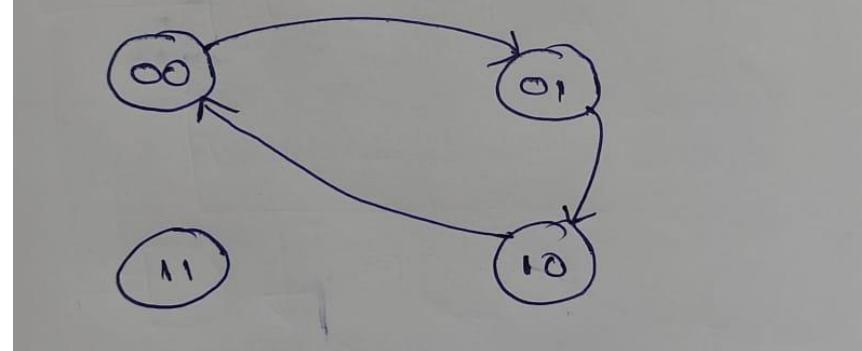
```
1 `timescale 1ns / 1ps
2 module ripple_counter_4bit_tb;
3     reg clk;
4     wire [3:0] Q;
5     ripple_counter_4bit uut (.clk(clk), .Q
6     (Q));
7     initial begin
8         clk = 0;
9         forever #5 clk = ~clk;
10    end
11    initial begin
12        $dumpfile("ripple_counter_4bit.vcd");
13        $dumpvars(0, ripple_counter_4bit_tb);
14        #160;
15        $finish;
16    end
17 endmodule
```

GTKWave





State diagram



```

1 `timescale 1ns / 1ps
2 module sequential_circuit_DFF(
3     input  clk,
4     input  x, y,
5     output reg z);
6     reg A, B;
7     wire A_next, B_next;
8     initial begin
9         A = 1'b0;
10        B = 1'b0;
11    end
12    assign A_next = (x & ~y) | (x &
13 B); assign B_next = (x & A) | (x & ~
14 B); always @ (posedge clk) begin
15     A <= A_next;
16     B <= B_next;
17 end
18 always @ (*) z = A;
19 endmodule

```

Verilog code

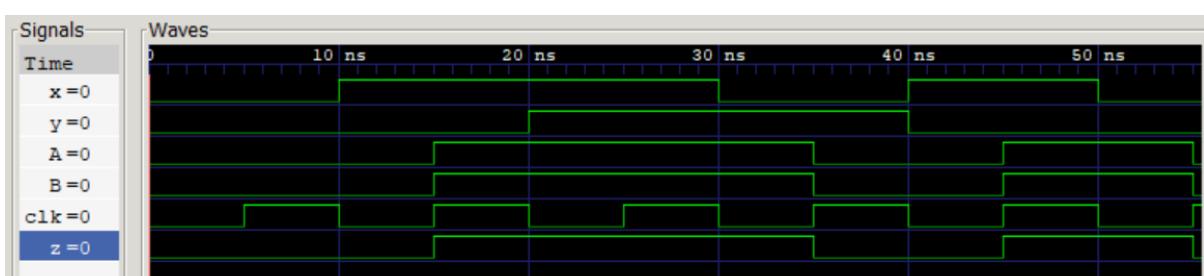
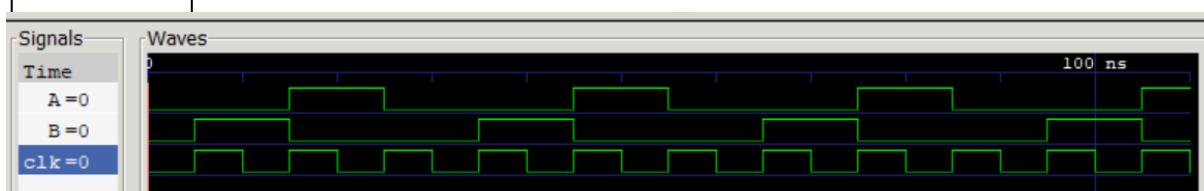
testbench

```

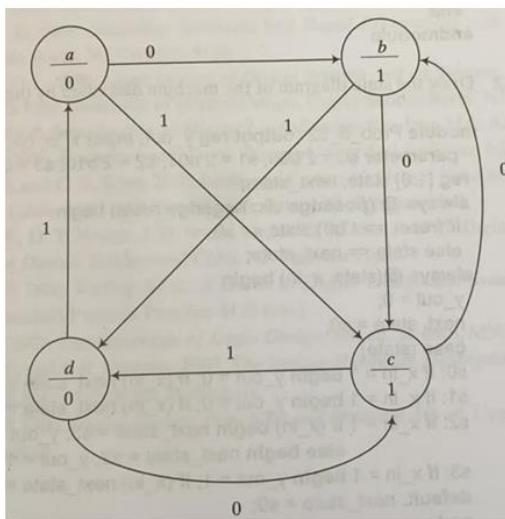
1 module exercise9_tb;
2     reg clk, x, y;
3     wire z;
4     sequential_circuit_DFF seq_dff (.clk(clk), .x(x), .y(y), .z
5 (z));
6     sequential_circuit_TFF seq_tff (.clk(clk));
7     initial begin
8         clk = 0;
9         forever #5 clk = ~clk;
10    end
11    initial begin
12        $dumpfile("exercise9_final.vcd");
13        $dumpvars(0, exercise9_tb);
14        x = 0; y = 0; #10;
15        x = 1; y = 0; #10;
16        x = 1; y = 1; #10;
17        x = 0; y = 1; #10;
18        x = 1; y = 0; #10;
19        x = 0; y = 0; #10;
20        #50;
21        $finish;
22    end
23 endmodule

```

GTKWave



- Implement Moore FSM for given FSM diagram



Verilog code

```

1 `timescale 1ns / 1ps
2 module moore_fsm(
3     input wire clk,
4     input wire x,
5     output reg y,
6     output reg [1:0] state);
7 localparam [1:0] A = 2'b00,
8                 B = 2'b01,
9                 C = 2'b10,
10                D = 2'b11;
11 reg [1:0] next_state;
12 always @(*) begin
13     case (state)
14         A: begin
15             if (x == 1'b0) next_state =
16             B; else next_state = C;
17         end
18         B: begin
19             if (x == 1'b0) next_state =
20             C; else next_state = D;
21         end
22         C: begin
23             if (x == 1'b0) next_state =
24             B; else next_state = D;
25         end
26         D: begin
27             if (x == 1'b0) next_state =
28             C; else next_state = A;
29         end
30         default: next_state = A;
31     endcase
32 end
33 always @(posedge clk) begin
34     if ($time < 1)
35         state <= A;
36     else
37         state <= next_state;
38 end
39 always @(*) begin
40     case (state)
41         A: y = 1'b0;
42         B: y = 1'b1;
43         C: y = 1'b1;
44         D: y = 1'b0;
45         default: y = 1'b0;
46     endcase
47 end
48 endmodule
49
  
```

testbench

Figure 2: Circuit

```

1 `timescale 1ns/1ps
2 module tb_moore_fsm;
3     reg clk;
4     reg x;
5     wire y;
6     wire [1:0] state;
7     moore_fsm dut (
8         .clk(clk),
9         .x(x),
10        .y(y),
11        .state(state));
12 initial clk = 0;
13 always #5 clk = ~clk;
14 initial begin
15     x = 0;
16     @(posedge clk); x = 1'b0; #1;
17     @(posedge clk); x = 1'b1; #1;
18     @(posedge clk); x = 1'b0; #1;
19     @(posedge clk); x = 1'b1; #1;
20     @(posedge clk); x = 1'b1; #1;
21     @(posedge clk); x = 1'b1; #1;
22     @(posedge clk); x = 1'b0; #1;
23     @(posedge clk); x = 1'b0; #1;
24     @(posedge clk); x = 1'b1; #1;
25     @(posedge clk); x = 1'b0; #1;
26     repeat (6) begin
27         @(posedge clk); x = $random; #
28         1; end
29     #20;
30     $finish;
31 end
32 initial begin
33     $dumpfile("moore_fsm.vcd");
34     $dumpvars(0, tb_moore_fsm);
35 end
36 endmodule
37
  
```

GTKWave



- Implement Mealy FSM for given FSM diagram

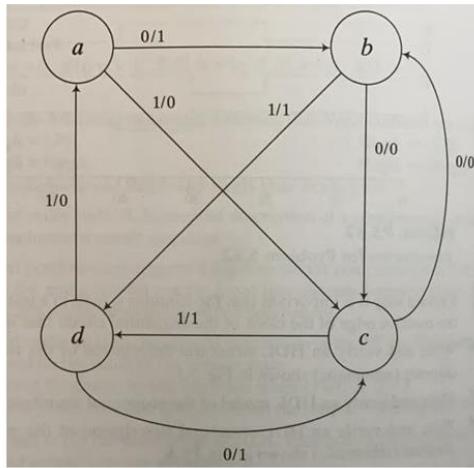


Figure 3: Circuit

testbench

```

1 `timescale 1ns/1ps
2 module tb_mealy_fsm;
3   reg clk;
4   reg x;
5   wire z;
6   wire [1:0] state;
7   mealy_fsm dut (
8     .clk(clk),
9     .x(x),
10    .z(z),
11    .state(state));
12 initial clk = 0;
13 always #5 clk = ~clk;
14 function [8*4:1] state_name;
15   input [1:0] s;
16   begin
17     case (s)
18       2'b00: state_name = "A";
19       2'b01: state_name = "B";
20       2'b10: state_name = "C";
21       2'b11: state_name = "D";
22       default: state_name = "?";
23     endcase
24   end
25 endfunction
26 initial begin
27   x = 0;
28   #12;
29   @(posedge clk); x = 1'b0; #1;
30   @(posedge clk); x = 1'b1; #1;
31   @(posedge clk); x = 1'b0; #1;
32   @(posedge clk); x = 1'b1; #1;
33   @(posedge clk); x = 1'b1; #1;
34   @(posedge clk); x = 1'b1; #1;
35   @(posedge clk); x = 1'b0; #1;
36   @(posedge clk); x = 1'b0; #1;
37   @(posedge clk); x = 1'b1; #1;
38   @(posedge clk); x = 1'b0; #1;
39   repeat (6) begin
40     @(posedge clk); x = $random; #
41   end
42   #20;
43   $finish;
44 end
45 initial begin
46   $dumpfile("mealy_fsm.vcd");
47   $dumpvars(0, tb_mealy_fsm);
48 end
49 endmodule
50
51
  
```

Verilog code

```

1 `timescale 1ns/1ps
2 module mealy_fsm (
3   input wire clk,
4   input wire x,
5   output reg z,
6   output reg [1:0] state);
7 localparam [1:0] A = 2'b00,
8               B = 2'b01,
9               C = 2'b10,
10              D = 2'b11;
11 reg [1:0] next_state;
12 always @(*) begin
13   next_state = A;
14   z = 1'b0;
15   case (state)
16     A: begin
17       if (x == 1'b0) begin
18         next_state = B;
19         z = 1'b1;
20       end else begin
21         next_state = C;
22         z = 1'b0;
23       end
24     end
25     B: begin
26       if (x == 1'b0) begin
27         next_state = C;
28         z = 1'b0;
29       end else begin
30         next_state = D;
31         z = 1'b1;
32       end
33     end
34     C: begin
35       if (x == 1'b0) begin
36         next_state = B;
37         z = 1'b0;
38       end else begin
39         next_state = D;
40         z = 1'b1;
41       end
42     end
43     D: begin
44       if (x == 1'b0) begin
45         next_state = C;
46         z = 1'b1;
47       end else begin
48         next_state = A;
49         z = 1'b0;
50       end
51     end
52   default: begin
53     next_state = A;
54     z = 1'b0;
55   end
56 endcase
57 end
58 always @(posedge clk) begin
59   state <= next_state;
60 end
61 endmodule
  
```

GTKWave

Signals

Time
x=1
state[1:0]=00
clk=0
z=0

Waves



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

```

1 `timescale 1ns / 1ps
2 module tb_fsm_11;
3   reg CLK;
4   reg RESET;
5   reg IN;
6   wire OUT;
7   fsm_11 dut (
8     .CLK(CLK),
9     .RESET(RESET),
10    .IN(IN),
11    .OUT(OUT));
12 initial CLK = 0;
13 always #5 CLK = ~CLK;
14 initial begin
15   $dumpfile("fsm_11.vcd");
16   $dumpvars(0, tb_fsm_11);
17   RESET = 1;
18   IN = 0;
19   #20 RESET = 0;
20   #30 IN = 1;
21   #30 IN = 0;
22   #20 IN = 0;
23   #20 IN = 1;
24   #30 IN = 1;
25   #30 IN = 0;
26   #20 IN = 0;
27   #20 IN = 1;
28   #50 $finish;
29 end
30 endmodule

```

Verilog code

```

1 `timescale 1ns / 1ps
2 module fsm_11(
3   input wire CLK,
4   input wire RESET,
5   input wire IN,
6   output reg OUT);
7   wire ONE = IN;
8   wire ZERO = ~IN;
9   reg prev_one;
10  reg prev_zero;
11  reg [1:0] count_one;
12  reg [1:0] count_zero;
13  always @ (posedge CLK or posedge RESET) begin
14    if (RESET) begin
15      prev_one <= 1'b0;
16      prev_zero <= 1'b0;
17      count_one <= 2'd0;
18      count_zero <= 2'd0;
19      OUT <= 1'b0;
20    end else begin
21      if (prev_one && !ONE) begin
22        count_one <= count_one + 1;
23      end
24      if (prev_zero && !ZERO) begin
25        count_zero <= count_zero + 1;
26      end
27      prev_one <= ONE;
28      prev_zero <= ZERO;
29      if ((count_one >= 2) && (count_zero >=
30        2)) begin
31        OUT <= 1'b1;
32      end
33    end
34  end
35 endmodule
36

```

GTKWave

