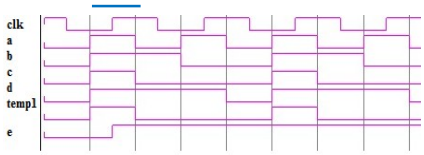
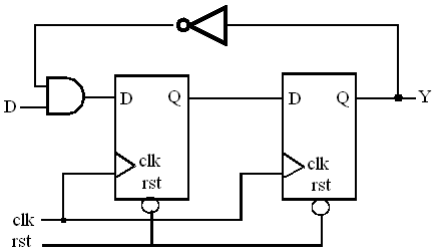
Complete the waveform for the hardware design below

```
assign temp1 = a & b;
assign c = temp1;
assign d = a|b;
always@(posedge clk)
begin
  c<=d;
end
```

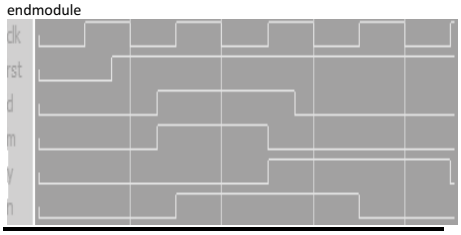
Solution:



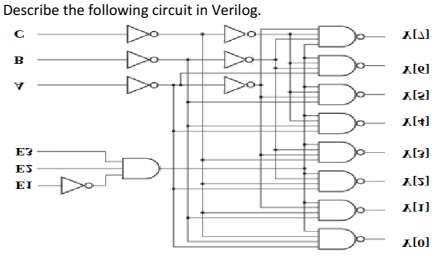
Write Verilog Program & TB for this circuit.



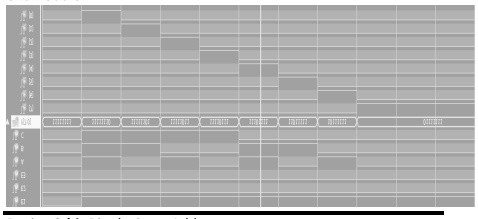

```
module program (clk, d, rst, y0;
input clk, d, rst;
output y;
reg y, n;
wire m;
assign m = d & (~y);
always@ (posedge clk or negedge rst)
begin
  if(~rst)
  begin
    n <= 0; y <= 0;
  end
  else
  begin
    n <= m; y <= n;
  end
end
endmodule
////////////////////////////////////////////////////////////
module program_tb;
reg clk, d, rst;
wire y;
program uut(clk, d, rst, y);
initial clk = 0;
always #10 clk = ~clk;
initial
begin
  rst = 0; d = 0;
  #16 rst = 1;
  #10 d = 1;
  #30 d = 0;
  #200 $stop;
end
endmodule
```



Describe the following circuit in Verilog.

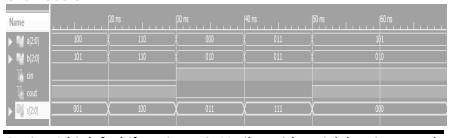



```
module decoder (e1, e2, e3, a, b, c, y);
input a, b, c, e1, e2, e3;
output [7:0] y;
wire m;
assign m = ~e1 & e2 & e3;
assign y[0] = ~((~c & ~b & ~a) & m);
assign y[1] = ~((~c & ~b & a) & m);
assign y[2] = ~((~c & b & ~a) & m);
assign y[3] = ~((~c & b & a) & m);
assign y[4] = ~((c & ~b & ~a) & m);
assign y[5] = ~((c & ~b & a) & m);
assign y[6] = ~((c & b & ~a) & m);
assign y[7] = ~((c & b & a) & m);
endmodule
////////////////////////////////////////////////////////////////////////////////
//////////////
module decoder_tb;
reg e1, e2, e3, a, b, c;
wire [7:0] y;
decoder uut(e1, e2, e3, a, b, c, y);
initial
begin
  {e1, e2, e3} = 3'b111;
  {c, b, a} = 3,b111;
  #10 {e1, e2, e3} = 3'b011;
  {c, b, a} = 3'b000;
  #10 {c, b, a} = 3'b001;
  #10 {c, b, a} = 3'b010;
  #10 {c, b, a} = 3'b011;
  #10 {c, b, a} = 3'b100;
  #10 {c, b, a} = 3'b101;
  #10 {c, b, a} = 3'b110;
  #10 {c, b, a} = 3'b111;
  #50 $stop;
end
endmodule
```



Design **3-bit** Ripple Carry Adder

```
module fulladder(a, b, cin, cout, s);
input a, b, cin;
output cout, s;
assign s = a ^ b ^ cin;
assign cout = ((a^b)&cin) | (a&b);
endmodule
////////////////////////////////////////////////////////////////
module ripplecarryadder3(a, b, cin, cout, s);
input [2:0] a, b;
input cin;
output [2:0] s;
output cout;
wire [1:0] m;
fulladder g1(.a(a[0]), .b(b[0]), .cin(cin), .cout(m[0]), .s(s[0]));
fulladder g2(.a(a[1]), .b(b[1]), .cin(m[0]), .cout(m[1]), .s(s[1]));
fulladder g3(a[2], b[2], m[1], cout, s[2]);
endmodule
////////////////////////////////////////////////////////////////
module ripplecarryadder3_tb;
reg [2:0] a, b;
reg cin;
wire[2:0] s;
wire cout;
ripplecarryadder3 uut(a, b, cin, cout, s);
initial
begin
  a = 0; b = 0; c = 0;
  #10 a = 4; b = 5;
  #10 a = 6; b = 6;
  #10 a = 0; b = 2; cin = 1;
  #10 a = 3; b = 3;
  #10 a = 5; b = 2;
  #20 $stop;
end
endmodule
```

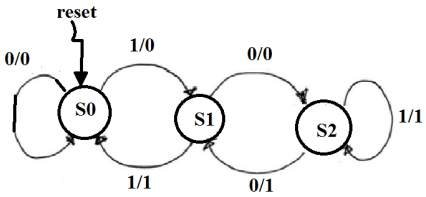


Design 4-bit left shift registers in Verilog with serial data input and serial data output.

```
module shift_L4(clk, din, dout);
input clk, din; output dout;
reg [3:0] q; assign dout = q[3];
always@(posedge clk)
begin
  //q <= { q[2:0], din };
  q[3] <= q[2];
  q[2]<= q[1];
  q[1] <= q[0];
  q[0] <= din;
end
endmodule
```

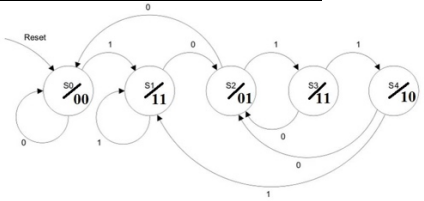
```
//Right shift//
module shift_R4(clk, din, dout);
input clk, din;
output dout;
reg [3:0] q;
assign dout = q[0];
always@(posedge clk)
begin
  q <= { din, q[3:1] };
end
endmodule
```

Design the following finite state machine (mealy)




```
module fsm(clk, reset, a, y);
input clk, reset, a;
output y;  reg y;
parameter s0 = 2'b00, s1=2'b01, s2=2'b10;
reg [1:0] cs, ns;
always@(posedge clk or posedge reset)
begin
  if(reset)  cs <= s0;
  else  cs <= ns;
end
always@(cs or a)
begin
  case(cs)  s0: if(a) ns = s1;
  else ns = s0; s1: if(~a) ns = s2;
  else ns = s0; s2: if(a) ns = s2;
  else ns = s1; default: ns = s0;
  endcase
end
always@(cs or a)
begin
  case(cs)  s0: y = 0;
  s1: if(a) y = 1; else y = 0;
  s2: y = 1;
  default: y = 0;
  endcase
end
endmodule
```

Design the following finite state machine (moore)




```
module fsm ( clk, reset, a, y );
input clk, reset, a;
output [1:0] y;
reg [1:0] y;
parameter s0 = 3'b000, s1=3'b001, s2=3'b010, s3=3'b011,
s4=3'b100; reg [2:0] cs, ns;
always@(posedge clk or negedge reset)
begin
  if(~reset) cs <= s0;
  else  cs <= ns;
end
always@(cs or a)
begin
  case(cs)  s0: if(a) ns = s1;
  else ns = s0; s1: if(~a) ns = s2;
  else ns = s0; s2: if(a) ns = s3;
  else ns = s0; s3: if(a) ns = s4;
  else ns = s2; s4: if(a) ns = s1;
  else default:
  endcase
end
always@(cs)
begin
  ns = s2; ns = s0;
  case(cs)
  s0: y = 2'b00;
  s1:  y = 2'b11;
  s2:  y = 2'b01;
  s3: y = 2'b11;
  s4: y = 2'b10;
  default: y = 2'b00;
  endcase
end
endmodule
```

Half Adder.v

```
module halfadder(sout,cout,a,b);
output sout,cout;
input a,b;
assign sout=a^b;
assign cout=(a&b);
endmodule
```

Half Adder_TB.v

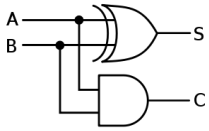
```
module halfadder_tb; //test batch
reg ta, tb;
wire tcout, tsum;
halfadder u1 (.a(ta), .b(tb), .cout(tcout), .sum(tsum)); //name
association
initial
begin
        ta = 0; tb = 0;
        #10; ta = 0; tb = 1;
        #10;
        ta = 1; tb = 0;
        #10;
        ta = 1; tb = 1;
        #10 $stop;
end
endmodule
```

Inputs		Outputs	
A	B	C_{out}	Sum
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	1

Logic Equations:

C_{out} = A & B

Sum = A ⊕ B



Full Adder.v

```
module fulladder(sout,cout,a,b,cin);
output sout,cout;
input a,b,cin;
assign sout=(a^b^cin);
assign cout=((a&b)|(a&cin)|(b&cin));
endmodule
```

Full Adder_TB.v

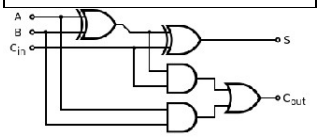
```
module fulladder_tb;
reg a, b, cin;
wire cout, sum;
fulladder u3(.a(a), .b(b), .cin(cin), .cout(cout), .sum(sum));
initial
begin
        a = 0; b = 0; cin = 0;
        #10 a = 0; b = 0; cin = 1;
        #10; a = 0; b = 1; cin = 0;
        #10; a = 0; b = 1; cin = 1;
        #10; a = 1; b = 0; cin = 0;
        #10; a = 1; b = 0; cin = 1;
        #10; a = 1; b = 1; cin = 0;
        #10; a = 1; b = 1; cin = 1;
        #10 $stop;
end
endmodule
```

Inputs			Outputs	
A	B	C_{in}	C_{out}	Sum
0	0	0	0	0
0	0	1	0	1
0	1	0	0	1
0	1	1	1	0
1	0	0	0	1
1	0	1	1	0
1	1	0	1	0
1	1	1	1	1

Logic Equations:

Sum = A ⊕ B ⊕ C_{in}

C_{out} = (A ⊕ B) C_{in} + A & B



Multiply4bits.v

```
module multiply4bits(pro,a,b);
output [7:0]pro;
input [3:0]a;
input [3:0]b;
assign pro[0]=(a[0] & b[0]);
wire x1,x2,x3,x4,x5,x6,x7,x8,x9,x10,x11,x12,x13,x14,x15,x16,x17;
halfadder halfadder1(pro[1],x1,(a[1] & b[0]),(a[0] & b[1]));
fulladder fulladder1(x2,x3,a[1]&b[1],(a[0]&b[2]),x1);
fulladder fulladder2(x4,x5,(a[1]&b[2]),(a[0]&b[3]),x3);
halfadder halfadder2(x6,x7,(a[1]&b[3]),x5);
halfadder halfadder3(pro[2],x15,x2,(a[2]&b[0]));
fulladder fulladder5(x14,x16,x4,(a[2]&b[1]),x15);
fulladder fulladder4(x13,x17,x6,(a[2]&b[2]),x16);
fulladder fulladder3(x9,x8,x7,(a[2]&b[3]),x17);
halfadder halfadder4(pro[3],x12,x14,(a[3]&b[0]));
fulladder fulladder8(pro[4],x11,x13,(a[3]&b[1]),x12);
fulladder fulladder7(pro[5],x10,x9,(a[3]&b[2]),x11);
fulladder fulladder6(pro[6],pro[7],x8,(a[3]&b[3]),x10);
endmodule
```

Multiply4bits_TB.v

```
module multiply4bits_tb();
reg [3:0]a;
reg [3:0]b;
wire [7:0]pro;
multiply4bits u1(.a(a), .b(b), .pro(pro));
initial
begin
                a = 4'b0000; b = 4'b0000;
                #10;
                a = 4'b0001; b = 4'b0001;
                #10;
                a = 4'b0010; b = 4'b0010;
                #10;
                a = 4'b0011; b = 4'b0011;
                #10;
                a = 4'b0100; b = 4'b0100;
                #10;
                a = 4'b0101; b = 4'b0101;
                #10;
                a = 4'b0110; b = 4'b0110;
                #10;
                a = 4'b0111; b = 4'b0111;
                #10;
                a = 4'b1000; b = 4'b1000;
                #10;
                a = 4'b1001; b = 4'b1001;
                #10;
                a = 4'b1010; b = 4'b1010;
                #10;
                a = 4'b1011; b = 4'b1011;
                #10;
                a = 4'b1100; b = 4'b1100;
                #10;
                a = 4'b1101; b = 4'b1101;
                #10;
                a = 4'b1110; b = 4'b1110;
                #10;
                a = 4'b1111; b = 4'b1111;
                #10; $stop;
end
endmodule
```

Inputs		Outputs
A[3:0]	B[3:0]	Pro[7:0]
0000	0000	00000000
0001	0001	00000001
0010	0010	00000100
0011	0011	00001001
0100	0100	00010000
0101	0101	00011001
0110	0110	00110001
0111	0111	00110001
1000	1000	01000000
1001	1001	01010001
1010	1010	01100100
1011	1011	01111001
1100	1100	10010000
1101	1101	10101001
1110	1110	11000100
1111	1111	11100001

Use Verilog to design a finite state machine to recognize the sequence 0110.

Analysis:

S_idle: does not detect anything

S1: detect 0

S2: detect 01

S3: detect 011

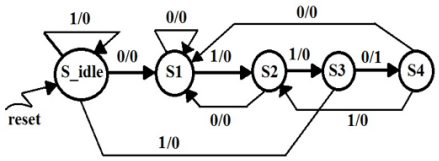
S4: detect 0110

State Table:

Current State	A	Next state	Y
S_idle	0	S1	0
	1	S_idle	0
S1	0	S1	0
	1	S2	0
S2	0	S1	0
	1	S3	0
S3	0	S4	1
	1	S_idle	0
S4	0	S1	0
	1	S2	0

Final Solution:

State Diagram:




```
module fsm_detector(reset, clk, a, y);
input reset, a, clk;
output y;
reg y;
parameter s_idle = 3'b000, s1=3'b001, s2=3'b010, s3=3'b011, s4=3'b100;
reg [2:0] cs, ns;
always@ (posedge clk or posedge reset)
begin
  if(reset) cs <= s_idle;
  else cs <= ns;
end
always@(cs or a)
begin
  case(cs)
    s_idle: if(a) ns = s_idle;
    else ns = s1;
    s1: if(a) ns = s2;
  else ns = s1;
    s2: if(a) ns = s3;
  else ns = s1;
    s3: if(~a) ns = s4;
  else ns = s_idle;
    s4: if(a) ns = s2;
  else ns = s1;
  default: ns = s_idle;
  endcase
end
always@(cs or a)
begin
  case(cs)
  s_idle: y = 0;
  s1: y=0;
  s2: y=0;
  s3:  if(~a) y=1;
       else y=0;
  s4:  y=0;
  default: y = 0;
  endcase
end
endmodule
```

D-Flip-Flop

```
module dffa ( reset, clk, load, da, qa);
input   reset, clk, load;
input  [3:0]  da;
output [3:0]  qa;
reg    [3:0]  qa;
always@(posedge clk or posedge reset)
begin
    if (reset)
        qa <= 4'b0000;
    else if (load)
        qa <= da;
end
endmodule
`timescale 1ns / 1ps
module dffa_tb;
                // Inputs
                reg reset;
                reg clk;
                reg load;
                reg [3:0] da;
                // Outputs
                wire [3:0] qa;
                // Instantiate the Unit Under Test (UUT)
                dffa uut (
                                .reset(reset),
                                .clk(clk),
                                .load(load),
                                .da(da),
                                .qa(qa)
                );
                //continuous 20 sec clk
                always
                begin
                                clk = 1'b1;
                                #10;
                                clk = 1'b0;
                                #10;
                end
                initial begin
                                // Initialize Inputs
                                load = 1'b1;
                                da = 1'b1;
                                reset = 1'b0;
                                #20;
                                load = 1'b0;
                                da = 1'b0;
                                reset = 1'b1;
                                #20;
                                load = 1'b0;
                                da = 1'b0;
                                reset = 1'b0;
                                #20;
                                load = 1'b0;
                                da = 1'b1;
                                reset = 1'b1;
                                #20;
                                load = 1'b1;
                                da = 1'b1;
                                reset = 1'b0;
                                #20
                                // Wait 100 ns for global reset to finish
                                 $stop;
                                // Add stimulus here
                end
endmodule
```