1. Consider the circuit below. It has three inputs (x and clock), and one output (z). At reset, the circuit starts with the outputs of all flip-flops at 0.

(a) Is this a Mealy machine or a Moore machine?
The output depends on one of the present inputs (x) as well as the present state, so it is a Mealy machine.

(b) Derive the state transition table for the circuit
Call the output of the J-K flip-flop A and the output of the D flip-flop B. We need to find the J and K inputs to the J-K flip flop (J_A, K_A). From these, we can find the next state of A. The next state of B will be the input to the D flip-flop.

\begin{align*}
J_A &= Bx \\
K_A &= x + B' \\
D_B &= B'x' + A'B \\
z &= AB'x
\end{align*}

\[
\begin{array}{|c|c|c|c|c|c|}
\hline
A & B & x & J_A & K_A & A & B & z \\
\hline
0 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
0 & 0 & 1 & 0 & 1 & 0 & 0 & 0 \\
0 & 1 & 0 & 0 & 0 & 0 & 1 & 0 \\
0 & 1 & 1 & 1 & 1 & 1 & 1 & 0 \\
1 & 0 & 0 & 0 & 1 & 0 & 1 & 0 \\
1 & 0 & 1 & 0 & 1 & 0 & 0 & 1 \\
1 & 1 & 0 & 0 & 0 & 1 & 0 & 0 \\
1 & 1 & 1 & 1 & 1 & 0 & 0 & 0 \\
\hline
\end{array}
\]

(c) Draw a state diagram for the circuit.
(d) Write a Verilog program to implement the functionality of the circuit. Be sure to reset the machine with all flip-flops at 0.

We could implement a J-K flip-flop in a Verilog module, then use that flip-flop and the equations for $J_A$ and $K_A$. Instead, I will tell the Verilog compiler implement the state transition table.

```verilog
module hw8_p1(input x, clk, output z);

reg A,B;

always @(posedge clk)
    if (~A & ~B & ~x) begin
        A <= 0; B <= 1;
    end
    if (~A & ~B & x) begin
        A <= 0; B <= 0;
    end
    if (~A & B & ~x) begin
        A <= 0; B <= 1;
    end
    if (~A & B & x) begin
        A <= 1; B <= 1;
    end
    if ( A & ~B & ~x) begin
        A <= 0; B <= 1;
    end
    if ( A & ~B & x) begin
        A <= 0; B <= 0;
    end
    if ( A & B & ~x) begin
        A <= 1; B <= 0;
    end
    if ( A & B & x) begin
        A <= 0; B <= 0;
    end

assign z = A & ~B & x;

endmodule
```
2. Consider the following state transition table. It implements a twisted ring counter:

<table>
<thead>
<tr>
<th>Present State</th>
<th>Next State</th>
</tr>
</thead>
<tbody>
<tr>
<td>$A$ $B$ $C$</td>
<td>$A$ $B$ $C$</td>
</tr>
<tr>
<td>0 0 0</td>
<td>1 0 0</td>
</tr>
<tr>
<td>1 0 0</td>
<td>1 1 0</td>
</tr>
<tr>
<td>1 1 0</td>
<td>1 1 1</td>
</tr>
<tr>
<td>1 1 1</td>
<td>0 1 1</td>
</tr>
<tr>
<td>0 1 1</td>
<td>0 0 1</td>
</tr>
<tr>
<td>0 0 1</td>
<td>0 0 0</td>
</tr>
</tbody>
</table>

(a) Draw a state diagram for the system.

(b) Write a Verilog program to implement the system.

```verilog
module hw8_p2(input clock, output reg A, B, C);
always @( posedge clock)
  case ( {A,B,C} )
    3'b000: {A,B,C} <= 3'b100;
    3'b100: {A,B,C} <= 3'b110;
    3'b110: {A,B,C} <= 3'b111;
    3'b111: {A,B,C} <= 3'b011;
    3'b011: {A,B,C} <= 3'b001;
    3'b001: {A,B,C} <= 3'b000;
    default: {A,B,C} <= 3'b000;
  endcase
endmodule
```
3. Design a synchronous sequential circuit to control the operation of an automatic coffee machine. A cup of coffee costs 15¢ and the machine has two input slots. In one slot, only 10¢ coins can be inserted; in the other, only 5¢ coins. The machine will give change in 5¢ coins only, and only one such coin per transaction.

(a) Draw a state diagram for the circuit.

There are two inputs: a nickel deposited \((n)\) or a dime deposited \((d)\). I will assume you cannot put in a nickel and a dime at the same time.

There are two outputs: dispense of cup of coffee \((k)\) and give a nickel change \((c)\).

You could design this as a Mealy machine or a Moore machine. Here I design it as a Moore machine.

There are 5 states: No money deposited, 5¢ deposited, 10¢ deposited, 15¢ and 20¢. When 15¢ is deposited, the dispenser should dispense a cup of coffee. When 20¢ is deposited, the dispenser should dispense a cup of coffee and return 5¢ change.

I assume that another coin cannot be inserted between states \(S15\) and \(S0\), or between states \(S20\) and \(S0\). This is a reasonable assumption – it will take at least several milliseconds before a person can deposit another coin and the coin can fall into the coin box, while the time between state \(S15\) and \(S0\) is one clock cycle, probably less than a microsecond.
(b) Assign states, and draw a state transition table.

There are five states, so you need at least three flip-flops. To simplify the output logic, I make the following state assignments:

<table>
<thead>
<tr>
<th>State</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>n</th>
<th>d</th>
<th>A</th>
<th>B</th>
<th>C</th>
<th>k</th>
<th>c</th>
</tr>
</thead>
<tbody>
<tr>
<td>S0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>S5</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>S10</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>S15</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td>S20</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>0</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
<tr>
<td></td>
<td>1</td>
<td>0</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td>x</td>
<td></td>
</tr>
</tbody>
</table>
(c) Write a Verilog program to implement the circuit.

```verilog
module hw8_p3(input clock, n, d, output k, c);

reg [2:0] state;

parameter S0 = 3'b000,
              S5 = 3'b001,
              S10 = 3'b010,
              S15 = 3'b100,
              S20 = 3'b101;

always @( posedge clock )
    case (state)
        S0: if (n) state <= S5;
            else if (d) state <= S10;
            else state <= S0;
        S5: if (n) state <= S10;
            else if (d) state <= S15;
            else state <= S5;
        S10: if (n) state <= S15;
            else if (d) state <= S20;
            else state <= S10;
        S15: state <= S0;
        S20: state <= S0;
        default: state <= S0;
    endcase

    assign k = ((state == S15) || (state == S20)) ? 1'b1 : 1'b0;
    assign c = (state == S20) ? 1'b1 : 1'b0;

endmodule
```
4. Here I will repeat Problem 3, but design the system as a Mealy machine. For a Mealy machine, you don’t need the states S15 or S20. If you are in state S10 and someone deposits a nickel, the machine can dispense a cup of coffee and go back to state S0.

Here is the state machine:
module hw8_p3(input clock, n, d, output reg k, c);

reg [1:0] state;

parameter S0 = 3'b000,
             S5 = 3'b001,
             S10 = 3'b010;

always @(posedge clock)
case (state)
  S0: if (n) state <= S5;
       else if (d) state <= S10;
       else state <= S0;
  S5: if (n) state <= S10;
       else if (d) state <= S0;
       else state <= S5;
  S10: if (n) state <= S0;
       else if (d) state <= S0;
       else state <= S10;
  default: state <= S0;
endcase

always @(state, n, d)
begin
  if ((state == S5) && (d)) k = 1;
  else if ((state == S10) && (n)) k = 1;
  else if ((state == S10) && (d)) k = 1;
  else k = 0;

  if ((state == S10) && (d)) c = 1;
  else c = 0;
end
endmodule