Lab Code [10 points]
Filename: abstractChipInterface.sv

```
 1 `default_nettype none
 2
 3 module abstractChipInterface (
 4   input logic [2:0] KEY,
 5   input logic [17:0] SW,
 6   output logic [6:0] HEX0,
 7   output logic [7:0] LEDG);
 8
 9   logic [3:0] creditVal;
10   logic dropVal;
11
12   myAbstractFSM dut(.clock(KEY[0]), .reset_N(SW[5]), .coin(SW[1:0]),
13                     .credit(creditVal), .drop(dropVal));
14
15   BCDtoSevenSegment muah(.bcd(creditVal), .segment(HEX0));
16
17   always_comb begin
18     if (dropVal)
19       LEDG = 8'b11111111;
20     else
21       LEDG = 8'b00000000;
22   end
23 endmodule: abstractChipInterface
```

Lab Code [10 points]
Filename: abstractFSM.sv

```
 1 `default_nettype none
 2
 3 module myAbstractFSM (
 4   input logic [1:0] coin,
 5   output logic drop,
 6   output logic [3:0] credit,
 7   input logic clock, reset_N);
 8   enum logic [2:0] {init = 3'b000, cred1 = 3'b001, cred2 = 3'b010,
 9                     cred3 = 3'b011, cred0Soda = 3'b100, cred1Soda = 3'b101,
10                     cred2Soda = 3'b110, cred3Soda = 3'b111}
Line contains tabs (each tab replaced by 2 spaces in this print)
11                     currState, nextState;
12 // (increase bitwidth if you need more than eight states)
13 // (don't specify state encoding values)
14 // Next state logic is defined here. You are basically
15 // transcribing the "next-state" column of the state transition
16 // table into a SystemVerilog case statement.
17   always_comb begin
18     case (currState)
19       init: begin
20               if (coin == 2'b00)
21                 nextState = init;
22               else if (coin == 2'b01)
23                 nextState = cred1;
24               else if (coin == 2'b10)
25                 nextState = cred3;
26               else
27                 nextState <= cred1Soda;
28              end
29    cred1: begin
30             if (coin == 2'b00)
31               nextState = cred1;
32             else if (coin == 2'b01)
33               nextState = cred2;
34             else if (coin == 2'b10)
35               nextState = cred0Soda;
36             else
37               nextState = cred2Soda;
38           end
39
40    cred2: begin
41             if (coin == 2'b00)
42               nextState = cred2;
43             else if (coin == 2'b01)
44               nextState = cred3;
45             else if (coin == 2'b10)
46               nextState = cred1Soda;
47             else
48               nextState = cred3Soda;
49           end
50
51    cred3: begin
52             if (coin == 2'b00)
53               nextState = cred3;
54             else if (coin == 2'b01)
55               nextState = cred0Soda;
56             else if (coin == 2'b10)
57               nextState = cred2Soda;
58             else
59               nextState = cred0Soda;
60           end
61
62    cred0Soda: nextState = init;
63    cred1Soda: nextState = cred1;
64    cred2Soda: nextState = cred2;
65    cred3Soda: nextState = cred3;
66    default: nextState = init;
67    endcase
68   end
69
```

```
 70
 71   always_comb begin
 72     drop = 0;
 73     credit = 4'b0000;
 74     unique case (currState)
 75    init: credit = 4'b0000;
 76       cred1: credit = 4'b0001;
 77    cred2: credit = 4'b0010;
 78    cred3: credit = 4'b0011;
 79    cred0Soda: drop = 1;
 80    cred1Soda: begin
 81                 drop = 1;
 82                 credit = 4'b0001;
 83               end
 84    cred2Soda: begin
 85                 drop = 1;
 86                 credit = 4'b0010;
 87               end
 88    cred3Soda: begin
 89                 drop = 1;
 90                 credit = 4'b0011;
Line contains tabs (each tab replaced by 2 spaces in this print)
 91               end
Line contains tabs (each tab replaced by 2 spaces in this print)
 92     endcase
 93   end
 94
 95 // Synchronous state update described here as an always block.
 96 // In essence, these are your flip flops that will hold the state
 97 // This doesn't do anything interesting except to capture the new
 98 // state value on each clock edge. Also, synchronous reset.
 99   always_ff @(posedge clock)
100   if (~reset_N)
101     currState <= init; // or whatever the reset state is
102   else
103     currState <= nextState;
104
105 endmodule: myAbstractFSM
```

Lab Code [10 points]
Filename: abstractFSMtest.sv

```
module myAFSM_test();
  logic [3:0] credit;
  logic [1:0] coin;
  logic drop, clock, reset_N;
  myAbstractFSM dut(.*);
  initial begin
    $monitor($time,, "state=%s, coin=%d, credit=%d, drop=%b",
             dut.currState.name, coin, credit, drop);
    clock = 0;
    forever #5 clock = ~clock;
  end

  initial begin
  // initialize values
    coin <= 2'b00; reset_N <= 0;
  // reset is synchronous, so must wait for a clock edge
    @(posedge clock);
    @(posedge clock);
  // One edge is enough, but what the heck
    @(posedge clock);
  // release reset and start the vector 01 10 00 11
    reset_N <= 1; // changes "after" the clock edge
    coin <= 2'b01;
    @(posedge clock); // 1 credit
    @(posedge clock); // 2 credit
    @(posedge clock); // 3 credit
    @(posedge clock); // 0 credit Soda
    @(posedge clock); // 0 init
    @(posedge clock); // 1 credit
    @(posedge clock); // 2 credit
    coin <= 2'b10;
    @(posedge clock); // 1 cred Soda
    @(posedge clock); // 1 cred
    coin <= 2'b11;
    @(posedge clock); // 2 cred Soda
    @(posedge clock); // 2 cred
    coin <= 2'b11;
    @(posedge clock); // 3 cred Soda
    @(posedge clock); // 3 cred
    coin <= 2'b11;
    @(posedge clock); // 0 cred Soda
    @(posedge clock); // 0 cred
    coin = 2'b10;
    @(posedge clock); // 3 cred
    coin = 2'b10;
    @(posedge clock); // 2 cred Soda
    @(posedge clock); // 2 cred
    // begin cycle 2
    coin <= 2'b00;
    @(posedge clock); // 2 cred
    coin <= 2'b01;
    @(posedge clock); // 3 cred
    coin <= 2'b00;
    @(posedge clock); // 3 cred
    reset_N <= 0;
    @(posedge clock); // init
    reset_N <= 1;
    @(posedge clock); // init
    coin <= 2'b11;
    @(posedge clock); // 1 cred Soda
    @(posedge clock); // 1 cred
    coin <= 2'b00;
    @(posedge clock); // 1 cred
    @(posedge clock);
    coin <= 2'b10;
    @(posedge clock);

    #1 $finish;
  end
endmodule: myAFSM_test
```

Lab Code [10 points]
Filename: structuralChipInterface.sv

```
 1 `default_nettype none
 2
 3 module structuralChipInterface (
 4   input logic [2:0] KEY,
 5   input logic [17:0] SW,
 6   output logic [6:0] HEX1, HEX0,
 7   output logic [7:0] LEDG);
 8
 9   logic [3:0] creditVal;
10   logic dropVal;
11
12   myStructuralFSM dut(.clock(KEY[0]), .reset(SW[5]), .coin(SW[1:0]),
13                       .credit(creditVal), .drop(dropVal));
14
15   BCDtoSevenSegment muah(.bcd(creditVal), .segment(HEX0));
16
17   always_comb begin
18     if (dropVal)
19       LEDG = 8'b11111111;
20     else
21       LEDG = 8'b00000000;
22   end
23 endmodule: structuralChipInterface
```

Lab Code [10 points]
Filename: structuralFSM.sv

```
 1 `default_nettype none
 2
 3 module dFlipFlop(
 4   output logic q,
 5   input logic d, clock, reset);
 6
 7   always @(posedge clock)
 8   if (reset == 1)
 9     q <= 0;
10   else
11     q <= d;
12
13 endmodule: dFlipFlop
14
15 module myStructuralFSM(
16   input logic [1:0] coin,
17   input logic clock, reset,
18   output logic [3:0] credit,
19   output logic drop);
20
21   logic q0, q1, q2;
22   logic d0, d1, d2;
23
24   dFlipFlop ff0 (.d(d0), .q(q0), .clock, .reset),
25             ff1 (.d(d1), .q(q1), .clock, .reset),
26       ff2 (.d(d2), .q(q2), .clock, .reset);
Line contains tabs (each tab replaced by 2 spaces in this print)
27
Line contains tabs (each tab replaced by 2 spaces in this print)
28   // state logic
29   // state assignments: init = 000, 1cred = 001,
30   // 2cred = 010, 3cred = 011, 0credSoda = 100,
31   // 1credSoda = 101, 2credSoda = 110, 3credSoda = 111
32   always_comb begin
33     d0 = (q2 & q0) | (~q2 & ~q0 & coin[0]) | (q0 & ~coin[1]
34           & ~coin[0]) | (~q2 & ~q0 & coin[1]) | (q2 & q0);
35     d1 = (q1 & ~coin[1] & ~coin[0]) | (q1 & ~q0 & coin[0]) |
36          (q1 & q0 & coin[1] & ~coin[0]) | (~q2 & q1 & ~q0 &
37           coin[0]) | (~q2 & ~q1 & ~q0 & coin[1] & ~coin[0]) |
38           (q2 & q1) | (~q2 & ~q1 & q0 & coin[0]);
39     d2 = (~q2 & q1 & coin[1]) | (~q2 & coin[1] & q0) | (~q2 &
40           q1 & q0 & coin[0]) | (~q2 & ~q1 & coin[1] & coin[0]);
41   end
42
43   always_comb begin
Line contains tabs (each tab replaced by 2 spaces in this print)
44     drop = q2;
Line contains tabs (each tab replaced by 2 spaces in this print)
45     credit = {1'b0, 1'b0, q1, q0};
Line contains tabs (each tab replaced by 2 spaces in this print)
46   end
Line contains tabs (each tab replaced by 2 spaces in this print)
47
Line contains tabs (each tab replaced by 2 spaces in this print)
48 endmodule: myStructuralFSM
49
Line contains tabs (each tab replaced by 2 spaces in this print)
50
Line contains tabs (each tab replaced by 2 spaces in this print)
```

Lab Code [10 points]
Filename: structuralFSMtest.sv

```
 1 module myFSM_test();
 2   logic [3:0] credit;
 3   logic [1:0] coin;
 4   logic drop, clock, reset;
 5   myStructuralFSM dut(.*);
 6   initial begin
 7     $monitor($time,, "q2=%b, q1=%b, q0=%b, coin1=%b, coin0=%b, /
 8                       credit=%d, drop=%b",
 9              dut.q2, dut.q1, dut.q0, coin[1], coin[0], credit, drop);
10     clock = 0;
11     forever #5 clock = ~clock;
12   end
13
14   initial begin
15   // initialize values
16     coin <= 'b00; reset <= 1;
17   // reset is synchronous, so must wait for a clock edge
18     @(posedge clock);
19     @(posedge clock);
20   // One edge is enough, but what the heck
21     @(posedge clock);
22   // release reset and start the vector 01 10 00 11
23     reset <= 0; // changes "after" the clock edge
24     coin <= 2'b01;
25     @(posedge clock); // 1 credit
26     @(posedge clock); // 2 credit
27     @(posedge clock); // 3 credit
28     @(posedge clock); // 0 credit Soda
29     @(posedge clock); // 0 init
30     @(posedge clock); // 1 credit
31     @(posedge clock); // 2 credit
32     coin <= 2'b10;
33     @(posedge clock); // 1 cred Soda
34     @(posedge clock); // 1 cred
35     coin <= 2'b11;
36     @(posedge clock); // 2 cred Soda
37     @(posedge clock); // 2 cred
38     coin <= 2'b11;
39     @(posedge clock); // 3 cred Soda
40     @(posedge clock); // 3 cred
41     coin <= 2'b11;
42     @(posedge clock); // 0 cred Soda
43     @(posedge clock); // 0 cred
44     coin = 2'b10;
45     @(posedge clock); // 3 cred
46     coin = 2'b10;
47     @(posedge clock); // 2 cred Soda
48     @(posedge clock); // 2 cred
49     // begin cycle 2
50     coin <= 2'b00;
51     @(posedge clock); // 2 cred
52     coin <= 2'b01;
53     @(posedge clock); // 3 cred
54     coin <= 2'b00;
55     @(posedge clock); // 3 cred
56     @(posedge clock);
57     reset <= 1;
58     @(posedge clock); // init
59     reset <= 0;
60     @(posedge clock); // init
61     coin <= 2'b11;
62     @(posedge clock); // 1 cred Soda
63     @(posedge clock); // 1 cred
64     coin <= 2'b00;
65     @(posedge clock); // 1 cred
66     coin <= 2'b10;
67     @(posedge clock);
68     #1 $finish;
69   end
70 endmodule: myFSM_test
```