```
 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

```systemverilog
 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...
```

Line length of 103 (max is 80)
Line contains tabs (each tab replaced by 2 spaces in this print)

```systemverilog
11                                                          currState, ne...
```

Line length of 85 (max is 80)
Line contains tabs (each tab replaced by 2 spaces in this print)

```systemverilog
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
```

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

```
 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    // BCDtoSevenSegment muwe (.bcd({1'b0, dut.q2, dut.q1, dut.q0}), segment(HE...
Line length of 82 (max is 80)
17
18    always_comb begin
19      if (dropVal)
20        LEDG = 8'b11111111;
21      else
22        LEDG = 8'b00000000;
23    end
24  endmodule: structuralChipInterface
```

```systemverilog
 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)
```systemverilog
27
```
Line contains tabs (each tab replaced by 2 spaces in this print)
```systemverilog
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
44
45
46
```
Line contains tabs (each tab replaced by 2 spaces in this print)
```systemverilog
47     always_comb begin
```
Line contains tabs (each tab replaced by 2 spaces in this print)
```systemverilog
48       drop = q2;
```
Line contains tabs (each tab replaced by 2 spaces in this print)
```systemverilog
49       credit = {1'b0, 1'b0, q1, q0};
```
Line contains tabs (each tab replaced by 2 spaces in this print)
```systemverilog
50     end
```
Line contains tabs (each tab replaced by 2 spaces in this print)
```systemverilog
51
```
Line contains tabs (each tab replaced by 2 spaces in this print)
```systemverilog
52  endmodule: myStructuralFSM
```
```systemverilog
53
```
Line contains tabs (each tab replaced by 2 spaces in this print)
```systemverilog
54
```
Line contains tabs (each tab replaced by 2 spaces in this print)

```
 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, credit=%d, dro...
```
Line length of 83 (max is 80)
```
 8             dut.q2, dut.q1, dut.q0, coin[1], coin[0], credit, drop);
 9     clock = 0;
10     forever #5 clock = ~clock;
11   end
12
13   initial begin
14   // initialize values
15     coin <= 'b00; reset <= 1;
16   // reset is synchronous, so must wait for a clock edge
17     @(posedge clock);
18     @(posedge clock);
19   // One edge is enough, but what the heck
20     @(posedge clock);
21   // release reset and start the vector 01 10 00 11
22     reset <= 0; // changes "after" the clock edge
23     coin <= 2'b01;
24     @(posedge clock); // 1 credit
25     @(posedge clock); // 2 credit
26     @(posedge clock); // 3 credit
27     @(posedge clock); // 0 credit Soda
28     @(posedge clock); // 0 init
29     @(posedge clock); // 1 credit
30     @(posedge clock); // 2 credit
31     coin <= 2'b10;
32     @(posedge clock); // 1 cred Soda
33     @(posedge clock); // 1 cred
34     coin <= 2'b11;
35     @(posedge clock); // 2 cred Soda
36     @(posedge clock); // 2 cred
37     coin <= 2'b11;
38     @(posedge clock); // 3 cred Soda
39     @(posedge clock); // 3 cred
40     coin <= 2'b11;
41     @(posedge clock); // 0 cred Soda
42     @(posedge clock); // 0 cred
43     coin = 2'b10;
44     @(posedge clock); // 3 cred
45     coin = 2'b10;
46     @(posedge clock); // 2 cred Soda
47     @(posedge clock); // 2 cred
48     // begin cycle 2
49     coin <= 2'b00;
50     @(posedge clock); // 2 cred
51     coin <= 2'b01;
52     @(posedge clock); // 3 cred
53     coin <= 2'b00;
54     @(posedge clock); // 3 cred
55     @(posedge clock);
56     reset <= 1;
57     @(posedge clock); // init
58     reset <= 0;
59     @(posedge clock); // init
60     coin <= 2'b11;
61     @(posedge clock); // 1 cred Soda
62     @(posedge clock); // 1 cred
63     coin <= 2'b00;
64     @(posedge clock); // 1 cred
65     coin <= 2'b10;
66     @(posedge clock);
67     #1 $finish;
68   end
69 endmodule: myFSM_test
```