

Lab Code [15 points]

Filename: ChipInterface.sv

```
1 `default_nettype none
2
3 module ChipInterface(
4     input  logic [7:0] KEY,
5     input  logic [17:0] SW,
6     input  logic      CLOCK_50,
7     output logic [6:0] HEX9, HEX8, HEX7, HEX6, HEX5, HEX3, HEX2, HEX1, HEX0,
8     output logic [7:0] LEDG);
9
10    logic [3:0] hMove, cMove;
11    logic [15:0] HHEX, CHEX;
12    logic win, clock, reset, reset_N, enter, newGame;
13    logic [6:0] segment0, segment1, segment2, segment3, segment4, segment5,
14               segment6, segment7, segment8;
15
16    logic newGame_int, newGame_sync, enter_int, enter_sync;
17
18    assign clock = CLOCK_50;
19    assign reset = SW[17];
20    assign reset_N = ~reset;
21    assign hMove = SW[3:0];
22    assign enter = ~KEY[3];
23    assign newGame = ~KEY[0];
24
25    assign HEX5 = segment0;
26    assign HEX3 = segment1;
27    assign HEX2 = segment2;
28    assign HEX1 = segment3;
29    assign HEX0 = segment4;
30    assign HEX9 = segment5;
31    assign HEX8 = segment6;
32    assign HEX7 = segment7;
33    assign HEX6 = segment8;
34    assign LEDG[7:0] = {8{win}};
35
36    BCDtoSevenSegment BSS0 (.bcd(cMove), .segment(segment0)),
37                          BSS1 (.bcd(CHEX[15:12]), .segment(segment1)),
38                          BSS2 (.bcd(CHEX[11:8]), .segment(segment2)),
39                          BSS3 (.bcd(CHEX[7:4]), .segment(segment3)),
40                          BSS4 (.bcd(CHEX[3:0]), .segment(segment4)),
41                          BSS5 (.bcd(HHEX[15:12]), .segment(segment5)),
42                          BSS6 (.bcd(HHEX[11:8]), .segment(segment6)),
43                          BSS7 (.bcd(HHEX[7:4]), .segment(segment7)),
44                          BSS8 (.bcd(HHEX[3:0]), .segment(segment8));
45    abstractFSM fsm (.hMove, .valid(enter_sync), .newGame(newGame_sync), .cMove,
46                   .HHEX, .CHEX, .win, .clock, .reset_N);
47
48    always_ff @(posedge clock)
49        newGame_int <= newGame;
50
51    always_ff @(posedge clock)
52        newGame_sync <= newGame_int;
53
54    always_ff @(posedge clock)
55        enter_int <= enter;
56
57    always_ff @(posedge clock)
58        enter_sync <= enter_int;
59
60 endmodule: ChipInterface
61
62 module BCDtoSevenSegment
63     (input logic [3:0] bcd,
64      output logic [6:0] segment);
65
66     always_comb
67         case (bcd)
68             4'b0001: segment = 7'b111_1001;
69             4'b0010: segment = 7'b010_0100;
70             4'b0011: segment = 7'b011_0000;
```

```
71         4'b0100: segment = 7'b001_1001;
72         4'b0101: segment = 7'b001_0010;
73         4'b0110: segment = 7'b000_0010;
74         4'b0111: segment = 7'b111_1000;
75         4'b1000: segment = 7'b000_0000;
76         4'b1001: segment = 7'b001_1000;
77         default: segment = 7'b111_1111; // Display is off by default
78     endcase
79 endmodule: BCDtoSevenSegment
```

Lab Code [15 points]
Filename: lab4.sv

```
1 `default_nettype none
2
3 module abstractFSM(
4   input  logic [3:0] hMove,
5   input  logic valid,
6   input  logic newGame,
7   output logic [3:0] cMove,
8   output logic [15:0] HHEX,
9   output logic [15:0] CHEX,
10  output logic      win,
11  input  logic      clock, reset_N
12 );
13
14  enum logic [4:0] {new_game, new_game_int,
15                   cStart, cStart_int,
16                   first, first_int,
17                   second,
18                   win12, win12_int,
19                   win13, win13_int,
20                   win14, win14_int,
21                   win17, win17_int,
22                   win18, win18_int,
23
24                   win22, win22_int,
25                   win24, win24_int,
26                   win27, win27_int,
27                   win28, win28_int}
28  currState, nextState;
29
30  // Next State Generator
31  always_comb begin
32    case (currState)
33      new_game: nextState= (valid && newGame) ? new_game_int : new_game;
34      new_game_int: nextState= (~valid && ~newGame) ? cStart : new_game_int;
35
36      cStart: nextState = (valid && hMove == 4'd6) ? cStart_int: cStart;
37      cStart_int: nextState = (~valid) ? first : cStart_int;
38
39      first: begin
40        if(valid) begin
41          case(hMove)
42            4'd2: nextState= win12_int;
43            4'd3: nextState= win13_int;
44            4'd4: nextState= win14_int;
45            4'd7: nextState= win17_int;
46            4'd8: nextState= win18_int;
47            4'd9: nextState= first_int;
48            default: nextState= first;
49          endcase
50        end
51        else nextState= first;
52      end
53      first_int: nextState= (~valid) ? second : first_int;
54
55      second: begin
56        if(valid) begin
57          case(hMove)
58            4'd2: nextState= win22_int;
59            4'd4: nextState= win24_int;
60            4'd7: nextState= win27_int;
61            4'd8: nextState= win28_int;
62            default: nextState= second;
63          endcase
64        end
65        else nextState= second;
66      end
67
68      win12_int: nextState= (~valid) ? win12 : win12_int;
69      win12: nextState= newGame ? new_game : win12;
70
```

```
71      win13_int: nextState= (~valid) ? win13 : win13_int;
72      win13: nextState= newGame ? new_game : win13;
73
74      win14_int: nextState= (~valid) ? win14 : win14_int;
75      win14: nextState= newGame ? new_game : win14;
76
77      win17_int: nextState= (~valid) ? win17 : win17_int;
78      win17: nextState= newGame ? new_game : win17;
79
80      win18_int: nextState= (~valid) ? win18 : win18_int;
81      win18: nextState= newGame ? new_game : win18;
82
83
84
85      win22_int: nextState= (~valid) ? win22 : win22_int;
86      win22: nextState= newGame ? new_game : win22;
87
88      win24_int: nextState= (~valid) ? win24 : win24_int;
89      win24: nextState= newGame ? new_game : win24;
90
91      win27_int: nextState= (~valid) ? win27 : win27_int;
92      win27: nextState= newGame ? new_game : win27;
93
94      win28_int: nextState= (~valid) ? win28 : win28_int;
95      win28: nextState= newGame ? new_game : win28;
96
97
98
99      default: nextState = cStart; //never exec'd
100  endcase
101  end
102
103  // Output Generator
104  always_comb begin
105      win = 1'b0;
106      HHEX = 16'h00_00;
107      unique case (currState)
108          new_game, new_game_int: begin
109              cMove= 4'd15; //invalid
110              HHEX= 16'd0;
111              CHEX= 16'd0;
112          end
113          cStart, cStart_int: begin
114              cMove= 4'd5;
115              CHEX= 16'h50_00;
116          end
117
118          first, first_int: begin
119              cMove = 4'd1;
120              HHEX= 16'h60_00;
121              CHEX= 16'h15_00;
122          end
123
124          win12, win12_int: begin
125              cMove = 4'd9;
126              HHEX= 16'h26_00;
127              CHEX= 16'h15_90;
128              win = 1'b1;
129          end
130
131          win13, win13_int: begin
132              cMove = 4'd9;
133              HHEX= 16'h36_00;
134              CHEX= 16'h15_90;
135              win = 1'b1;
136          end
137
138          win14, win14_int: begin
139              cMove = 4'd9;
140              HHEX= 16'h46_00;
141              CHEX= 16'h15_90;
```

```
142         win = 1'b1;
143     end
144
145     win17, win17_int: begin
146         cMove = 4'd9;
147         HHEX= 16'h67_00;
148         CHEX= 16'h15_90;
149         win = 1'b1;
150     end
151
152     win18, win18_int: begin
153         cMove = 4'd9;
154         HHEX= 16'h68_00;
155         CHEX= 16'h15_90;
156         win = 1'b1;
157     end
158
159
160
161
162     second: begin
163         cMove = 4'd3;
164         HHEX= 16'h69_00;
165         CHEX= 16'h13_50;
166     end
167
168     win22, win22_int: begin
169         cMove = 4'd2;
170         HHEX= 16'h26_90;
171         CHEX= 16'h12_35;
172         win = 1'b1;
173     end
174
175     win24, win24_int: begin
176         cMove = 4'd2;
177         HHEX= 16'h46_90;
178         CHEX= 16'h12_35;
179         win = 1'b1;
180     end
181
182     win27, win27_int: begin
183         cMove = 4'd2;
184         HHEX= 16'h67_90;
185         CHEX= 16'h12_35;
186         win = 1'b1;
187     end
188
189     win28, win28_int: begin
190         cMove = 4'd2;
191         HHEX= 16'h68_90;
192         CHEX= 16'h12_35;
193         win = 1'b1;
194     end
195
196     endcase
197 end
198
199 always_ff @(posedge clock)
200     if (~reset_N)
201         currState <= cStart;
202     else
203         currState <= nextState;
204 endmodule: abstractFSM
```

Lab Code [15 points]
Filename: testb.sv

```
1
2 module abstractFSM_test();
3     logic [3:0] cMove, hMove;
4     logic      win, clock, reset;
5
6     myStructuralFSM DUT (.*);
7
8     initial begin
9         $monitor($time,, "state=%d, hMove=%d, cMove=%d, win=%b",
10                {DUT.q2, DUT.q1, DUT.q0}, hMove, cMove, win);
11         clock = 0;
12         forever #5 clock = ~clock;
13     end
14
15     initial begin
16         // Initialize Values
17         hMove <= 4'b0000; reset <= 1;
18
19         @(posedge clock);
20
21         // Release reset
22         reset <= 0;
23         // Start test path for cStart
24         hMove <= 4'd0;
25         @(posedge clock);
26         hMove <= 4'd1;
27         @(posedge clock);
28         hMove <= 4'd2;
29         @(posedge clock);
30         hMove <= 4'd3;
31         @(posedge clock);
32         hMove <= 4'd4;
33         @(posedge clock);
34         hMove <= 4'd5;
35         @(posedge clock);
36         hMove <= 4'd7;
37         @(posedge clock);
38         hMove <= 4'd8;
39         @(posedge clock);
40         hMove <= 4'd9;
41         @(posedge clock);
42         hMove <= 4'd10;
43         @(posedge clock);
44         hMove <= 4'd11;
45         @(posedge clock);
46         hMove <= 4'd12;
47         @(posedge clock);
48         hMove <= 4'd13;
49         @(posedge clock);
50         hMove <= 4'd14;
51         @(posedge clock);
52         hMove <= 4'd15;
53         @(posedge clock);
54
55         hMove <= 4'd6;
56         @(posedge clock); // To first
57
58         // Start testing First
59         hMove <= 4'd0;
60         @(posedge clock);
61         hMove <= 4'd1;
62         @(posedge clock);
63         hMove <= 4'd5;
64         @(posedge clock);
65         hMove <= 4'd6;
66         @(posedge clock);
67         hMove <= 4'd10;
68         @(posedge clock);
69         hMove <= 4'd11;
70         @(posedge clock);
```

```
71     hMove <= 4'd12;
72     @(posedge clock);
73     hMove <= 4'd13;
74     @(posedge clock);
75     hMove <= 4'd14;
76     @(posedge clock);
77     hMove <= 4'd15;
78     @(posedge clock);
79
80     // To win1
81     hMove <= 4'd2;
82     @(posedge clock);
83     reset <= 1;
84     @(posedge clock);
85     reset <= 0;
86
87     hMove <= 4'd3;
88     @(posedge clock);
89     reset <= 1;
90     @(posedge clock);
91     reset <= 0;
92
93     hMove <= 4'd4;
94     @(posedge clock);
95     reset <= 1;
96     @(posedge clock);
97     reset <= 0;
98
99     hMove <= 4'd7;
100    @(posedge clock);
101    reset <= 1;
102    @(posedge clock);
103    reset <= 0;
104
105    hMove <= 4'd8;
106    @(posedge clock);
107
108    // Start testing win1
109    for (logic [3:0] i = 4'd0; i < 4'b1111; i++) begin
110        hMove <= i;
111        @(posedge clock);
112    end
113    hMove <= 4'b1111;
114    @(posedge clock);
115    reset <= 1;
116    @(posedge clock);
117    reset <= 0;
118
119    hMove <= 4'd6;
120    @(posedge clock);
121    reset <= 1;
122    @(posedge clock);
123    reset <= 0;
124
125    // Start testing second
126    hMove <= 4'd6;
127    @(posedge clock);
128    hMove <= 4'd9;
129    @(posedge clock);
130    reset <= 1;
131    @(posedge clock); // test reset
132    reset <= 0;
133    hMove <= 4'd6;
134    @(posedge clock);
135    hMove <= 4'd9;
136    @(posedge clock);
137    hMove <= 4'd0;
138    @(posedge clock); // test invalid cases
139    hMove <= 4'd1;
140    @(posedge clock);
141    hMove <= 4'd3;
```

```
142     @(posedge clock);
143     hMove <= 4'd5;
144     @(posedge clock);
145     hMove <= 4'd9;
146     @(posedge clock);
147     hMove <= 4'd10;
148     @(posedge clock);
149     hMove <= 4'd11;
150     @(posedge clock);
151     hMove <= 4'd12;
152     @(posedge clock);
153     hMove <= 4'd13;
154     @(posedge clock);
155     hMove <= 4'd14;
156     @(posedge clock);
157     hMove <= 4'd15;
158     @(posedge clock);
159
160     hMove <= 4'd2;
161     @(posedge clock); // to win3
162
163     // Test win3
164     for (logic [3:0] i = 4'd0; i < 4'b1111; i++) begin
165         hMove <= i;
166         @(posedge clock);
167     end
168     hMove <= 4'b1111;
169     @(posedge clock);
170     reset <= 1;
171     @(posedge clock);
172     reset <= 0;
173     hMove <= 4'd6;
174     @(posedge clock);
175     hMove <= 4'd9;
176     @(posedge clock);
177     hMove <= 4'd4;
178     @(posedge clock);
179
180     reset <= 1;
181     @(posedge clock);
182     reset <= 0;
183     hMove <= 4'd6;
184     @(posedge clock);
185     hMove <= 4'd9;
186     @(posedge clock);
187     hMove <= 4'd7;
188     @(posedge clock);
189
190     reset <= 1;
191     @(posedge clock);
192     reset <= 0;
193     hMove <= 4'd6;
194     @(posedge clock);
195     hMove <= 4'd9;
196     @(posedge clock); // return to second
197     hMove <= 4'd8;
198     @(posedge clock);
199
200     // Test win2
201     for (logic [3:0] i = 4'd0; i < 4'b1111; i++) begin
202         hMove <= i;
203         @(posedge clock);
204     end
205     hMove <= 4'b1111;
206     @(posedge clock);
207     reset <= 1;
208     @(posedge clock);
209     reset <= 0;
210     @(posedge clock);
211     #1 $finish;
212 end
```



```
213 endmodule: abstractFSM_test
```

Lab Code [15 points]
Filename: tictac.sv

```

1 module dFlipFlop
2     (output logic q,
3      input logic d, clock, reset);
4
5     always @(posedge clock)
6         if (reset == 1)
7             q <= 0;
8         else
9             q <= d;
10 endmodule: dFlipFlop
11
12 module myStructuralFSM
13     (input logic [3:0] hMove,
14      output logic [3:0] cMove,
15      output logic win,
16      input logic clock, reset);
17
18 logic q0, q1, q2;
19 logic d0, d1, d2;
20
21 dFlipFlop ff2(.d(d2), .q(q2), .clock, .reset),
22             ff1(.d(d1), .q(q1), .clock, .reset),
23             ff0(.d(d0), .q(q0), .clock, .reset);
24
25 logic tmp, tmp1, tmp2, tmp3;
26 assign tmp= (~q2 & ~q1 & ~q0) &
27             ((hMove[3] ) |
28              (~hMove[3] & ~hMove[2] ) |
29              (~hMove[3] & hMove[2] & ~hMove[1] ) |
30              (~hMove[3] & hMove[2] & hMove[1] & hMove[0] ));
31
32 assign tmp1= (~q2 & ~q1 & q0) &
33             ((~hMove[3] & ~hMove[2] & ~hMove[1] ) |
34              (~hMove[3] & hMove[2] & ~hMove[1] & hMove[0] ) |
35              (~hMove[3] & hMove[2] & hMove[1] & ~hMove[0] ) |
36              (hMove[3] & ~hMove[2] & hMove[1] ) |
37              (hMove[3] & hMove[2] ));
38
39 assign tmp2= (~q2 & ~q1 & q0) &
40             ((~hMove[3] & ~hMove[2] & hMove[1] ) |
41              (~hMove[3] & hMove[2] & ~hMove[1] & ~hMove[0] ) |
42              (~hMove[3] & hMove[2] & hMove[1] & hMove[0] ) |
43              (hMove[3] & ~hMove[2] & ~hMove[1] & ~hMove[0] ));
44
45 assign tmp3= (~q2 & q1 & ~q0) &
46             ((~hMove[3] & ~hMove[2] & ~hMove[1] ) |
47              (~hMove[3] & ~hMove[2] & hMove[1] & hMove[0] ) |
48              (~hMove[3] & hMove[2] & ~hMove[1] & hMove[0] ) |
49              (~hMove[3] & hMove[2] & hMove[1] & ~hMove[0] ) |
50              (hMove[3] & ~hMove[2] & ~hMove[1] & hMove[0] ) |
51              (hMove[3] & ~hMove[2] & hMove[1] ) |
52              (hMove[3] & hMove[2] ));
53
54 /* Next State Logic */
55 assign d2= (~hMove[3] & ~hMove[2] & hMove[1] & ~hMove[0] & ~q2 & q1 & ~q0) |
56            (~hMove[3] & hMove[2] & ~hMove[1] & ~hMove[0] & ~q2 & q1 & ~q0) |
57            (~hMove[3] & hMove[2] & hMove[1] & hMove[0] & ~q2 & q1 & ~q0) |
58            (hMove[3] & ~hMove[2] & ~hMove[1] & ~hMove[0] & ~q2 & q1 & ~q0) |
59            (q2 & ~q1 & ~q0) |
60            (q2 & ~q1 & q0);
61
62 assign d1= (hMove[3] & ~hMove[2] & ~hMove[1] & hMove[0] & ~q2 & ~q1 & q0) |
63            tmp2 |
64            tmp3 |
65            (~q2 & q1 & q0);
66
67 assign d0= (~hMove[3] & hMove[2] & hMove[1] & ~hMove[0] & ~q2 & ~q1 & ~q0) |
68            tmp1 |
69            tmp2 |
70            (~hMove[3] & ~hMove[2] & hMove[1] & ~hMove[0] & ~q2 & q1 & ~q0) |

```

```
71          (~q2 & q1 & q0) |
72          (q2 & ~q1 & q0);
73
74
75 /* Output Logic */
76 assign cMove[3]= ~q2 & q1 & q0;
77
78 assign cMove[2]= (~hMove[3] & hMove[2] & hMove[1] & ~hMove[0] & ~q2 & ~q1 & ~q0)
79                 | tmp |
80                 (q2 & ~q1 & q0);
81
82 assign cMove[1]= tmp3 |
83                 (~hMove[3] & ~hMove[2] & hMove[1] & ~hMove[0] & ~q2 & q1 & ~q0)
84                 | (~hMove[3] & hMove[2] & ~hMove[1] & ~hMove[0] & ~q2 & q1 &
85                 ~q0) | (~hMove[3] & hMove[2] & hMove[1] & hMove[0] & ~q2 & q1
86                 & ~q0) | (hMove[3] & ~hMove[2] & ~hMove[1] & ~hMove[0] & ~q2 &
87                 q1 & ~q0) | (q2 & ~q1 & ~q0) | (q2 & ~q1 & q0);
88
89 assign cMove[0]= ~(q2 & ~q1 & ~q0);
90
91 assign win= (~q2 & q1 & q0) |
92            (q2 & ~q1 & ~q0) |
93            (q2 & ~q1 & q0);
94
95 endmodule : myStructuralFSM
```