

Lab Code [15 points]
 Filename: 01_struct.sv
 AndrewID: ocarroll

```

1 `default_nettype none
2
3 module dFlipFlop(
4   output logic q,
5   input logic d, clock, reset);
6
7   always_ff @(posedge clock)
8     if (reset == 1'b1)
9       q <= 0;
10    else
11      q <= d;
12
13 endmodule : dFlipFlop
14
15 // -----
16 // |      | Q2  Q1  Q0 | description
17 // |-----|
18 // | State1 | 0    0  0 | computer #5:
19 // | State2 | 0    0  1 | computer #1, #5;
20 // | State3 | 0    1  0 | computer #1, #5, #9;      win
21 // | State4 | 0    1  1 | computer #1, #3, #5;      win
22 // | State5 | 1    0  0 | computer #1, #2, #3, #7; win
23 // | State6 | 1    0  1 | computer #1, #3, #5, #7; win
24 // |-----|
25
26 module myExplicitFSM(
27   output logic [3:0] cMove,
28   output logic      win,
29   output logic      q0, q1, q2,
30   input logic [3:0] hMove,
31   input logic      clock, reset);
32
33   logic d0, d1, d2;
34
35   // flip-flops instantiation
36   dFlipFlop ff0(.d(d0),
37                 .q(q0),
38                 .clock(clock),
39                 .reset(reset)),
40   ff1(.d(d1),
41        .q(q1),
42        .clock(clock),
43        .reset(reset)),
44   ff2(.d(d2),
45        .q(q2),
46        .clock(clock),
47        .reset(reset));
48
49   // next state generation
50   assign d2 = ((~q2) & q1 & q0 & (~hMove[3]) & (~hMove[2]) & hMove[1]
51 & (~hMove[0])) |
52 ((~q2) & q1 & q0 & (~hMove[3]) & (~hMove[2]) & hMove[1]
53 & (~hMove[0])) |
54 ((~q2) & q1 & q0 & (~hMove[3]) & hMove[2] & hMove[1]
55 & hMove[0]) |
56 ((~q2) & q1 & q0 & hMove[3] & (~hMove[2]) & (~hMove[1])
57 & (~hMove[0])) |
58 (q2 & (~q1) & (~q0) & (~hMove[3]) & (~hMove[2]) & (~hMove[1])
59 & hMove[0]) |
60 (q2 & (~q1) & (~q0) & (~hMove[3]) & (~hMove[2]) & hMove[1]
61 & (~hMove[0])) |
62 (q2 & (~q1) & (~q0) & (~hMove[3]) & (~hMove[2]) & hMove[1]
63 & hMove[0]) |
64 (q2 & (~q1) & (~q0) & (~hMove[3]) & hMove[2] & (~hMove[1])
65 & (~hMove[0])) |
66 (q2 & (~q1) & (~q0) & (~hMove[3]) & hMove[2] & (~hMove[1])
67 & hMove[0]) |
68 (q2 & (~q1) & (~q0) & (~hMove[3]) & hMove[2] & hMove[1]
69 & hMove[0]) |

```

```

70      & (~hMove[0])) |
71      (q2 & (~q1) & (~q0) & (~hMove[3]) & hMove[2] & hMove[1]
72      & hMove[0]) |
73      (q2 & (~q1) & (~q0) & hMove[3] & (~hMove[2]) & (~hMove[1])
74      & (~hMove[0])) |
75      (q2 & (~q1) & (~q0) & hMove[3] & (~hMove[2]) & (~hMove[1])
76      & hMove[0]) |
77
78      (q2 & (~q1) & q0 & (~hMove[3]) & (~hMove[2]) & (~hMove[1])
79      & hMove[0]) |
80      (q2 & (~q1) & q0 & (~hMove[3]) & (~hMove[2]) & hMove[1]
81      & (~hMove[0])) |
82      (q2 & (~q1) & q0 & (~hMove[3]) & (~hMove[2]) & hMove[1]
83      & hMove[0]) |
84      (q2 & (~q1) & q0 & (~hMove[3]) & hMove[2] & (~hMove[1])
85      & (~hMove[0])) |
86      (q2 & (~q1) & q0 & (~hMove[3]) & hMove[2] & (~hMove[1])
87      & hMove[0]) |
88      (q2 & (~q1) & q0 & (~hMove[3]) & hMove[2] & hMove[1]
89      & (~hMove[0])) |
90      (q2 & (~q1) & q0 & (~hMove[3]) & hMove[2] & hMove[1]
91      & hMove[0]) |
92      (q2 & (~q1) & q0 & hMove[3] & (~hMove[2]) & (~hMove[1])
93      & (~hMove[0])) |
94      (q2 & (~q1) & q0 & hMove[3] & (~hMove[2]) & (~hMove[1])
95      & hMove[0]);
96
97  assign d1 = ((~q2) & (~q1) & q0 & (~hMove[3]) & (~hMove[2]) & hMove[1]
98  & (~hMove[0])) |
99  ((~q2) & (~q1) & q0 & (~hMove[3]) & (~hMove[2]) & hMove[1]
100  & hMove[0]) |
101  ((~q2) & (~q1) & q0 & (~hMove[3]) & hMove[2] & (~hMove[1])
102  & (~hMove[0])) |
103  ((~q2) & (~q1) & q0 & (~hMove[3]) & hMove[2] & hMove[1]
104  & hMove[0]) |
105  ((~q2) & (~q1) & q0 & hMove[3] & (~hMove[2]) & (~hMove[1])
106  & (~hMove[0])) |
107  ((~q2) & (~q1) & q0 & hMove[3] & (~hMove[2]) & (~hMove[1])
108  & hMove[0]) |
109
110  ((~q2) & q1 & (~q0) & (~hMove[3]) & (~hMove[2]) & (~hMove[1])
111  & hMove[0]) |
112  ((~q2) & q1 & (~q0) & (~hMove[3]) & (~hMove[2]) & hMove[1]
113  & (~hMove[0])) |
114  ((~q2) & q1 & (~q0) & (~hMove[3]) & (~hMove[2]) & hMove[1]
115  & hMove[0]) |
116  ((~q2) & q1 & (~q0) & (~hMove[3]) & hMove[2] & (~hMove[1])
117  & (~hMove[0])) |
118  ((~q2) & q1 & (~q0) & (~hMove[3]) & hMove[2] & (~hMove[1])
119  & hMove[0]) |
120  ((~q2) & q1 & (~q0) & (~hMove[3]) & hMove[2] & hMove[1]
121  & (~hMove[0])) |
122  ((~q2) & q1 & (~q0) & (~hMove[3]) & hMove[2] & hMove[1]
123  & hMove[0]) |
124  ((~q2) & q1 & (~q0) & q0 & hMove[3] & (~hMove[2]) & (~hMove[1])
125  & (~hMove[0])) |
126  ((~q2) & q1 & (~q0) & hMove[3] & (~hMove[2]) & (~hMove[1])
127  & hMove[0]) |
128
129  ((~q2) & q1 & q0 & (~hMove[3]) & (~hMove[2]) & (~hMove[1])
130  & hMove[0]) |
131  ((~q2) & q1 & q0 & (~hMove[3]) & (~hMove[2]) & hMove[1]
132  & hMove[0]) |
133  ((~q2) & q1 & q0 & (~hMove[3]) & hMove[2] & (~hMove[1])
134  & hMove[0]) |
135  ((~q2) & q1 & q0 & (~hMove[3]) & hMove[2] & hMove[1]
136  & (~hMove[0])) |
137  ((~q2) & q1 & q0 & hMove[3] & (~hMove[2]) & (~hMove[1])
138  & hMove[0]);
139
140  assign d0 = ((~q2) & (~q1) & (~q0) & (~hMove[3]) & hMove[2] & hMove[1]

```

```

141 & (~hMove[0])) |
142
143 ((~q2) & (~q1) & q0 & (~hMove[3]) & (~hMove[2]) & (~hMove[1])
144 & hMove[0]) |
145 ((~q2) & (~q1) & q0 & (~hMove[3]) & hMove[2] & (~hMove[1])
146 & hMove[0]) |
147 ((~q2) & (~q1) & q0 & (~hMove[3]) & (~hMove[2]) & (~hMove[1])
148 & hMove[0]) |
149
150
151 ((~q2) & q1 & q0 & (~hMove[3]) & (~hMove[2]) & (~hMove[1])
152 & hMove[0]) |
153 ((~q2) & q1 & q0 & (~hMove[3]) & (~hMove[2]) & hMove[1]
154 & (~hMove[0])) |
155 ((~q2) & q1 & q0 & (~hMove[3]) & (~hMove[2]) & hMove[1]
156 & hMove[0]) |
157 ((~q2) & q1 & q0 & (~hMove[3]) & hMove[2] & (~hMove[1])
158 & hMove[0]) |
159 ((~q2) & q1 & q0 & (~hMove[3]) & hMove[2] & hMove[1]
160 & (~hMove[0])) |
161 ((~q2) & q1 & q0 & hMove[3] & (~hMove[2]) & (~hMove[1])
162 & hMove[0]) |
163
164 (q2 & (~q1) & q0 & (~hMove[3]) & (~hMove[2]) & (~hMove[1])
165 & hMove[0]) |
166 (q2 & (~q1) & q0 & (~hMove[3]) & (~hMove[2]) & hMove[1]
167 & (~hMove[0])) |
168 (q2 & (~q1) & q0 & (~hMove[3]) & (~hMove[2]) & hMove[1]
169 & hMove[0]) |
170 (q2 & (~q1) & q0 & (~hMove[3]) & hMove[2] & (~hMove[1])
171 & (~hMove[0])) |
172 (q2 & (~q1) & q0 & (~hMove[3]) & hMove[2] & (~hMove[1])
173 & hMove[0]) |
174 (q2 & (~q1) & q0 & (~hMove[3]) & hMove[2] & hMove[1]
175 & (~hMove[0])) |
176 (q2 & (~q1) & q0 & (~hMove[3]) & hMove[2] & hMove[1]
177 & hMove[0]) |
178 (q2 & (~q1) & q0 & q0 & hMove[3] & (~hMove[2])
179 & (~hMove[1]) & (~hMove[0])) |
180 (q2 & (~q1) & q0 & hMove[3] & (~hMove[2]) & (~hMove[1])
181 & hMove[0]);
182
183 // output logic generation
184 assign cMove[3] = (~q2) & q1 & (~q0);
185
186 assign cMove[2] = ((~q2) & (~q1) & (~q0)) |
187 (q2 & (~q1) & q0);
188
189 assign cMove[1] = ((~q2) & q1 & q0) |
190 (q2 & (~q1) & (~q0)) |
191 (q2 & (~q1) & q0);
192
193 assign cMove[0] = ((~q2) & (~q1) & (~q0)) |
194 ((~q2) & (~q1) & q0) |
195 ((~q2) & q1 & (~q0)) |
196 ((~q2) & q1 & q0) |
197 (q2 & (~q1) & q0);
198
199 assign win = ((~q2) & q1 & (~q0)) |
200 (q2 & (~q1) & (~q0)) |
201 (q2 & (~q1) & q0);
202
203 endmodule : myExplicitFSM
204

```

Lab Code [15 points]

Filename: 01_testbench.sv

AndrewID: ocarroll

```

1 `default_nettype none
2
3 module testBench();
4     logic w1, w2, w3, w4, w5, w6;
5     logic [3:0] w7, w8;
6     myExplicitFSM dut1(.clock(w1),
7                         .reset(w2),
8                         .q1(w3),
9                         .q2(w4),
10                        .q0(w5),
11                        .win(w6),
12                        .cMove(w7),
13                        .hMove(w8));
14     myFSM_test dut2(.clock(w1),
15                    .reset(w2),
16                    .q1(w3),
17                    .q2(w4),
18                    .q0(w5),
19                    .win(w6),
20                    .cMove(w7),
21                    .hMove(w8));
22
23 endmodule : testBench
24
25
26 module myFSM_test(
27     input logic [3:0] cMove,
28     input logic win,
29     input logic q2, q1, q0,
30     output logic [3:0] hMove,
31     output logic clock, reset);
32
33
34
35     initial begin
36         clock = 0;
37         forever #5 clock = ~clock;
38     end
39
40     initial begin
41         $monitor($time,, "state=%b, cMove=%d, hMove=%d, win=%b",
42                 {q2, q1, q0}, cMove, hMove, win);
43         // initialize values
44         hMove <= 4'hF;
45         reset <= 1'b1;
46
47         // reset the FSM
48         @(posedge clock); // wait for a positive clock edge
49         @(posedge clock); // one edge is enough, but what the heck
50         @(posedge clock);
51
52         @(posedge clock); // begin cycle 0
53         reset <= 1'b0; // release the reset
54
55
56         // start an example sequence -- not meaningful for the lab
57         hMove <= 4'h6; // these changes are after the clock edge
58                       // which means the state change happens
59                       // AFTER the next clock edge
60         @(posedge clock); // begin cycle 1
61         hMove <= 4'h1;
62
63
64         // reset the FSM
65         @(posedge clock);
66         @(posedge clock);
67         @(posedge clock);
68
69         reset <= 1'b1;

```

```
70      @(posedge clock);
71      reset <= 1'b0;
72      //start
73      hMove <= 4'h6;
74      @(posedge clock); // 2-7 TEST
75      hMove <= 4'h9;
76      @(posedge clock);
77      hMove <= 4'h2;
78
79
80      // reset the FSM
81      @(posedge clock);
82      @(posedge clock);
83      @(posedge clock);
84
85      reset <= 1'b1;
86      @(posedge clock);
87      reset <= 1'b0;
88      //start
89      hMove <= 4'h6;
90      @(posedge clock); // 7-2 TEST
91      hMove <= 4'h9;
92      @(posedge clock);
93      hMove <= 4'h7;
94
95
96      // reset the FSM
97      @(posedge clock);
98      @(posedge clock);
99      @(posedge clock);
100
101      reset <= 1'b1;
102      @(posedge clock);
103      reset <= 1'b0;
104      //start
105      hMove <= 4'h6;
106      @(posedge clock); // not 9 test
107      hMove <= 4'h5;
108
109
110      // reset the FSM
111      @(posedge clock);
112      @(posedge clock);
113      @(posedge clock);
114
115      reset <= 1'b1;
116      @(posedge clock);
117      reset <= 1'b0;
118      //start
119      hMove <= 4'h6;
120      @(posedge clock); // not 7-2 test
121      hMove <= 4'h9;
122      @(posedge clock);
123      hMove <= 4'h4;
124
125
126      // reset the FSM
127      @(posedge clock);
128      @(posedge clock);
129      @(posedge clock);
130
131      reset <= 1'b1;
132      @(posedge clock);
133      reset <= 1'b0;
134      //start
135      hMove <= 4'h4; // not 6 test
136      @(posedge clock); // not
137
138      // reset the FSM
139      @(posedge clock);
140      @(posedge clock);
```

```
141      @(posedge clock);
142      reset <= 1'b1;
143      @(posedge clock);
144      reset <= 1'b0;
145      //start
146      hMove <= 4'h4; // not 6 test
147      @(posedge clock); // not
148
149
150
151      // reset the FSM
152      @(posedge clock);
153      @(posedge clock);
154      @(posedge clock);
155      reset <= 1'b1;
156      @(posedge clock);
157      reset <= 1'b0;
158      //start
159      hMove <= 4'h6;
160      @(posedge clock); // not 7-2 test
161      hMove <= 4'h9;
162      @(posedge clock);
163      hMove <= 4'h4;
164
165
166      // reset the FSM
167      @(posedge clock);
168      @(posedge clock);
169      @(posedge clock);
170      reset <= 1'b1;
171      @(posedge clock);
172      reset <= 1'b0;
173      //start
174      hMove <= 4'h6;
175      @(posedge clock); // not 7-2 test
176      hMove <= 4'h9;
177      @(posedge clock);
178      hMove <= 4'h4;
179
180
181      // reset the FSM
182      @(posedge clock);
183      @(posedge clock);
184      @(posedge clock);
185      reset <= 1'b1;
186      @(posedge clock);
187      reset <= 1'b0;
188      //start
189      hMove <= 4'h2;
190      @(posedge clock); // not 7-2 test
191      hMove <= 4'h3;
192      @(posedge clock);
193      hMove <= 4'h9;
194
195
196      // reset the FSM
197      @(posedge clock);
198      @(posedge clock);
199      @(posedge clock);
200      reset <= 1'b1;
201      @(posedge clock);
202      reset <= 1'b0;
203      //start
204      hMove <= 4'h6;
205      @(posedge clock); // not 7-2 test
206      hMove <= 4'h9;
207      @(posedge clock);
208      hMove <= 4'h4;
209
210      // reset the FSM
211      @(posedge clock);
```

```
212     @(posedge clock);
213     @(posedge clock);
214     reset <= 1'b1;
215     @(posedge clock);
216     reset <= 1'b0;
217     //start
218     hMove <= 4'h6;
219     @(posedge clock); // not 7-2 test
220     hMove <= 4'h9;
221     @(posedge clock);
222     hMove <= 4'h3;
223     @(posedge clock);
224     hMove <= 4'h4;
225
226     // reset the FSM
227     @(posedge clock);
228     @(posedge clock);
229     @(posedge clock);
230     reset <= 1'b1;
231     @(posedge clock);
232     reset <= 1'b0;
233     //start
234     hMove <= 4'h6;
235     @(posedge clock); // not 7-2 test
236     hMove <= 4'h9;
237     @(posedge clock);
238     hMove <= 4'h3;
239     @(posedge clock);
240     hMove <= 4'h8;
241     @(posedge clock);
242
243     // reset the FSM
244     @(posedge clock);
245     @(posedge clock);
246     @(posedge clock);
247     reset <= 1'b1;
248     @(posedge clock);
249     reset <= 1'b0;
250     //start
251     hMove <= 4'h6;
252     @(posedge clock); // not 7-2 test
253     hMove <= 4'h9;
254     @(posedge clock);
255     hMove <= 4'h3;
256     @(posedge clock);
257     hMove <= 4'h7;
258     @(posedge clock);
259     hMove <= 4'h4;
260     @(posedge clock);
261
262
263     // reset the FSM
264     @(posedge clock);
265     @(posedge clock);
266     @(posedge clock);
267     reset <= 1'b1;
268     @(posedge clock);
269     reset <= 1'b0;
270     //start
271     hMove <= 4'h6;
272     @(posedge clock); // not 7-2 test
273     hMove <= 4'h9;
274     @(posedge clock);
275     hMove <= 4'h3;
276     @(posedge clock);
277     hMove <= 4'h2;
278     @(posedge clock);
279     hMove <= 4'h4;
280     @(posedge clock);
281
282     @(posedge clock);
```

```
283         @(posedge clock);
284         @(posedge clock);
285         reset <= 1'b1;
286         @(posedge clock);
287         reset <= 1'b0;
288
289
290
291
292         #1 $finish;
293     end
294 endmodule: myFSM_test
```


Lab Code [15 points]
Filename: 02_abstract.sv
AndrewID: ocarroll

```
1 `default_nettype none
2
3 module myAbstractFSM (
4     output logic [3:0] cMove,
5     output logic      win,
6     input logic [3:0] hMove,
7     input logic      clock, reset);
8
9     enum logic [2:0] {S0 = 3'b000, S1 = 3'b001,
10                      S2 = 3'b010, S3 = 3'b011,
11                      S4 = 3'b100, S5 = 3'b101} currState, nextState;
12
13 // next state generation
14 always_comb
15     unique case (currState)
16         S0:
17             nextState = (hMove == 4'h6) ? S1 : S0;
18         S1: begin
19             if (hMove == 4'h1 || hMove == 4'h5 || hMove == 4'h6)
20                 nextState = S1;
21             else
22                 nextState = (hMove == 4'h9) ? S3 : S2;
23         end
24         S2:
25             nextState = S2;
26         S3: begin
27             if (hMove == 4'h1 || hMove == 4'h4 || hMove == 4'h5 || hMove == 4'h6
28                 || hMove == 4'h9)
29                 nextState = S3;
30             else
31                 nextState = (hMove == 4'h2) ? S5 : S4;
32         end
33         S4:
34             nextState = S4;
35         S5:
36             nextState = S5;
37     endcase
38
39
40 // output generation
41 always_comb begin
42     cMove = 4'b0000;
43     win = 1'b1;
44     if (currState == S0) begin
45         cMove = 4'h5;
46         win = 0;
47     end
48     if (currState == S1) begin
49         cMove = 4'h1;
50         win = 0;
51     end
52     if (currState == S2) begin
53         cMove = 4'h9;
54         win = 1;
55     end
56     if (currState == S3) begin
57         cMove = 4'h3;
58         win = 0;
59     end
60     if (currState == S4) begin
61         cMove = 4'h2;
62         win = 1;
63     end
64     if (currState == S5) begin
65         cMove = 4'h7;
66         win = 1;
67     end
68 end
69
```

```
70 // register
71 always_ff @(posedge clock)
72     if (reset)
73         currState <= S0;
74     else
75         currState <= nextState;
76
77 endmodule: myAbstractFSM
78
```

Lab Code [15 points]

Filename: 03_testbench.sv

AndrewID: ocarroll

```
1 `default_nettype none
2
3 module testBench();
4     logic [3:0] cMove, hMove;
5     logic clock, reset, win;
6     myAbstractFSM dut1(.clock(clock),
7                         .reset(reset),
8                         .win(win),
9                         .cMove(cMove),
10                        .hMove(hMove));
11
12     initial begin
13         clock = 0;
14         forever #5 clock = ~clock;
15     end
16
17     initial begin
18         $monitor($time,, "state=%s, cMove=%d, hMove=%d, win=%b",
19                 dut1.currState.name, cMove, hMove, win);
20         // initialize values
21         hMove <= 4'hF;
22         reset <= 1'b1;
23
24         // reset the FSM
25         @(posedge clock); // wait for a positive clock edge
26         @(posedge clock); // one edge is enough, but what the heck
27         @(posedge clock);
28
29         @(posedge clock); // begin cycle 0
30         reset <= 1'b0; // release the reset
31
32
33         // start an example sequence -- not meaningful for the lab
34         hMove <= 4'h6; // these changes are after the clock edge
35                       // which means the state change happens
36                       // AFTER the next clock edge
37         @(posedge clock); // begin cycle 1
38         hMove <= 4'h1;
39
40
41         // reset the FSM
42         @(posedge clock);
43         @(posedge clock);
44         @(posedge clock);
45
46         reset <= 1'b1;
47         @(posedge clock);
48         reset <= 1'b0;
49         //start
50         hMove <= 4'h6;
51         @(posedge clock); // 2-7 TEST
52         hMove <= 4'h9;
53         @(posedge clock);
54         hMove <= 4'h2;
55
56
57         // reset the FSM
58         @(posedge clock);
59         @(posedge clock);
60         @(posedge clock);
61
62         reset <= 1'b1;
63         @(posedge clock);
64         reset <= 1'b0;
65         //start
66         hMove <= 4'h6;
67         @(posedge clock); // 7-2 TEST
68         hMove <= 4'h9;
69         @(posedge clock);
```

```
70         hMove <= 4'h7;
71
72
73         // reset the FSM
74         @(posedge clock);
75         @(posedge clock);
76         @(posedge clock);
77
78         reset <= 1'b1;
79         @(posedge clock);
80         reset <= 1'b0;
81         //start
82         hMove <= 4'h6;
83         @(posedge clock); // not 9 test
84         hMove <= 4'h5;
85
86
87         // reset the FSM
88         @(posedge clock);
89         @(posedge clock);
90         @(posedge clock);
91
92         reset <= 1'b1;
93         @(posedge clock);
94         reset <= 1'b0;
95         //start
96         hMove <= 4'h6;
97         @(posedge clock); // not 7-2 test
98         hMove <= 4'h9;
99         @(posedge clock);
100        hMove <= 4'h4;
101
102
103        // reset the FSM
104        @(posedge clock);
105        @(posedge clock);
106        @(posedge clock);
107
108        reset <= 1'b1;
109        @(posedge clock);
110        reset <= 1'b0;
111        //start
112        hMove <= 4'h4; // not 6 test
113        @(posedge clock); // not
114
115        // reset the FSM
116        @(posedge clock);
117        @(posedge clock);
118        @(posedge clock);
119        reset <= 1'b1;
120        @(posedge clock);
121        reset <= 1'b0;
122        //start
123        hMove <= 4'h4; // not 6 test
124        @(posedge clock); // not
125
126
127
128        // reset the FSM
129        @(posedge clock);
130        @(posedge clock);
131        @(posedge clock);
132        reset <= 1'b1;
133        @(posedge clock);
134        reset <= 1'b0;
135        //start
136        hMove <= 4'h6;
137        @(posedge clock); // not 7-2 test
138        hMove <= 4'h9;
139        @(posedge clock);
140        hMove <= 4'h4;
```

```
141
142
143     // reset the FSM
144     @(posedge clock);
145     @(posedge clock);
146     @(posedge clock);
147     reset <= 1'b1;
148     @(posedge clock);
149     reset <= 1'b0;
150     //start
151     hMove <= 4'h6;
152     @(posedge clock); // not 7-2 test
153     hMove <= 4'h9;
154     @(posedge clock);
155     hMove <= 4'h4;
156
157
158     // reset the FSM
159     @(posedge clock);
160     @(posedge clock);
161     @(posedge clock);
162     reset <= 1'b1;
163     @(posedge clock);
164     reset <= 1'b0;
165     //start
166     hMove <= 4'h2;
167     @(posedge clock); // not 7-2 test
168     hMove <= 4'h3;
169     @(posedge clock);
170     hMove <= 4'h9;
171
172
173     // reset the FSM
174     @(posedge clock);
175     @(posedge clock);
176     @(posedge clock);
177     reset <= 1'b1;
178     @(posedge clock);
179     reset <= 1'b0;
180     //start
181     hMove <= 4'h6;
182     @(posedge clock); // not 7-2 test
183     hMove <= 4'h9;
184     @(posedge clock);
185     hMove <= 4'h4;
186
187
188     // reset the FSM
189     @(posedge clock);
190     @(posedge clock);
191     @(posedge clock);
192     reset <= 1'b1;
193     @(posedge clock);
194     reset <= 1'b0;
195     //start
196     hMove <= 4'h6;
197     @(posedge clock); // not 7-2 test
198     hMove <= 4'h9;
199     @(posedge clock);
200     hMove <= 4'h3;
201     @(posedge clock);
202     hMove <= 4'h4;
203
204     // reset the FSM
205     @(posedge clock);
206     @(posedge clock);
207     @(posedge clock);
208     reset <= 1'b1;
209     @(posedge clock);
210     reset <= 1'b0;
211     //start
212     hMove <= 4'h6;
```

```
212     @(posedge clock); // not 7-2 test
213     hMove <= 4'h9;
214     @(posedge clock);
215     hMove <= 4'h3;
216     @(posedge clock);
217     hMove <= 4'h8;
218     @(posedge clock);
219
220     // reset the FSM
221     @(posedge clock);
222     @(posedge clock);
223     @(posedge clock);
224     reset <= 1'b1;
225     @(posedge clock);
226     reset <= 1'b0;
227     //start
228     hMove <= 4'h6;
229     @(posedge clock); // not 7-2 test
230     hMove <= 4'h9;
231     @(posedge clock);
232     hMove <= 4'h3;
233     @(posedge clock);
234     hMove <= 4'h7;
235     @(posedge clock);
236     hMove <= 4'h4;
237     @(posedge clock);
238
239     // reset the FSM
240     @(posedge clock);
241     @(posedge clock);
242     @(posedge clock);
243     reset <= 1'b1;
244     @(posedge clock);
245     reset <= 1'b0;
246     //start
247     hMove <= 4'h6;
248     @(posedge clock); // not 7-2 test
249     hMove <= 4'h9;
250     @(posedge clock);
251     hMove <= 4'h3;
252     @(posedge clock);
253     hMove <= 4'h2;
254     @(posedge clock);
255     hMove <= 4'h4;
256     @(posedge clock);
257
258     @(posedge clock);
259     @(posedge clock);
260     @(posedge clock);
261     reset <= 1'b1;
262     @(posedge clock);
263     reset <= 1'b0;
264
265     #1 $finish;
266 end
267
268 endmodule : testBench
269
270
271 module myFSM_test(
272     input logic [3:0] cMove,
273     input logic win,
274     input logic q2, q1, q0,
275     output logic [3:0] hMove,
276     output logic clock, reset);
277
278
279
280
281 endmodule: myFSM_test
```

Lab Code [15 points]

Filename: 04_chipInterface.sv

AndrewID: ocarroll

```
1 `default_nettype none // Required in every sv file
2 module chipInterface
3     (input logic [3:0] KEY,
4      input logic [17:0] SW,
5      output logic [6:0] HEX0,
6      output logic [7:0] LEDG);
7     logic [3:0] c;
8
9     logic ww;
10    myAbstractFSM(.clock(KEY[0]), .reset(SW[17]), .hMove(SW[9:6]), .cMove(c),
11     .win(ww));
12    assign LEDG = {ww, ww, ww, ww, ww, ww, ww, ww};
13    logic [7:0] blank;
14    assign blank = 8'b00000000;
15
16    SevenSegmentDisplay DUT2 (.BCX0(c),
17                             .blank(blank),
18                             .HEX0(HEX0));
19
20 endmodule: chipInterface
```

Lab Code [15 points]
Filename: 05_HexDisplay.sv
AndrewID: ocarroll

```
1 `default_nettype none
2
3 module SevenSegmentDisplay
4   (input logic [3:0] BCX0,
5    input logic [7:0] blank,
6    output logic [6:0] HEX0);
7
8   always_comb begin
9     HEX0 = 7'b00000000;
10    if (~blank[0])
11      case (BCX0)
12        4'h0: HEX0 = 7'b01111111;
13        4'h1: HEX0 = 7'b00001110;
14        4'h2: HEX0 = 7'b10110111;
15        4'h3: HEX0 = 7'b10011111;
16        4'h4: HEX0 = 7'b11001110;
17        4'h5: HEX0 = 7'b11011101;
18        4'h6: HEX0 = 7'b11111101;
19        4'h7: HEX0 = 7'b00001111;
20        4'h8: HEX0 = 7'b11111111;
21        4'h9: HEX0 = 7'b11001111;
22        4'ha: HEX0 = 7'b11101111;
23        4'hb: HEX0 = 7'b11111100;
24        4'hc: HEX0 = 7'b01110011;
25        4'hd: HEX0 = 7'b10111110;
26        4'he: HEX0 = 7'b11110011;
27        4'hf: HEX0 = 7'b11100011;
28        default: HEX0 = 7'b00000000;
29      endcase
30    HEX0 = ~HEX0;
31  end
32
33
34 endmodule : SevenSegmentDisplay
```


Lab Code [15 points]

Filename: 05_chipInterface.sv

AndrewID: ocarroll

```
1 `default_nettype none // Required in every sv file
2 module chipInterface
3     (input logic [3:0] KEY,
4      input logic [17:0] SW,
5      input logic CLOCK_50,
6      output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7,
7      output logic [7:0] LEDG,
8      output logic [3:0] LEDR);
9     logic [3:0] h_3, h_2, h_1, h_0, c_3, c_2, c_1, c_0;
10
11     logic ww;
12
13     task5(.newGame_L(KEY[0]),
14          .clock(CLOCK_50),
15          .h3(h_3),
16          .h2(h_2),
17          .h1(h_1),
18          .h0(h_0),
19          .c3(c_3),
20          .c2(c_2),
21          .c1(c_1),
22          .c0(c_0),
23          .enter_L(KEY[3]),
24          .reset(SW[17]),
25          .hMove(SW[3:0]),
26          .cMove(LEDR[3:0]),
27          .win(ww));
28
29     assign LEDG = {ww, ww, ww, ww, ww, ww, ww, ww};
30     logic [7:0] blank;
31     assign blank = 8'b00000000;
32
33     SevenSegmentDisplay(.BCX0(h_3), .blank(blank), .HEX0(HEX7));
34     SevenSegmentDisplay(.BCX0(h_2), .blank(blank), .HEX0(HEX6));
35     SevenSegmentDisplay(.BCX0(h_1), .blank(blank), .HEX0(HEX5));
36     SevenSegmentDisplay(.BCX0(h_0), .blank(blank), .HEX0(HEX4));
37     SevenSegmentDisplay(.BCX0(c_3), .blank(blank), .HEX0(HEX3));
38     SevenSegmentDisplay(.BCX0(c_2), .blank(blank), .HEX0(HEX2));
39     SevenSegmentDisplay(.BCX0(c_1), .blank(blank), .HEX0(HEX1));
40     SevenSegmentDisplay(.BCX0(c_0), .blank(blank), .HEX0(HEX0));
41
42
43 endmodule: chipInterface
44
45
```

Lab Code [15 points]

Filename: 05_testbench.sv

AndrewID: ocarroll

```

1 `default_nettype none
2
3 module testbench();
4     logic [3:0] cMove, hMove;
5     logic [3:0] h3, h2, h1, h0, c3, c2, c1, c0;
6     logic win, clock, reset, enter_L, newGame_L;
7
8     task5 DUT (.*)
9
10    initial begin
11        $monitor($time,, "state: %30s cMove: %d hMove: %d win: %b \
12 // h3: %d h2: %d h1: %d h0: %d c3: %d c2: %d c1: %d c0: %d",
13                DUT.currState.name, cMove, hMove, win,
14                h3, h2, h1, h0, c3, c2, c1, c0);
15        // init
16        clock = 0;
17        reset = 1;
18        reset <= 0;
19
20        forever #10 clock = ~clock;
21    end
22
23    initial begin
24        // C_5
25        hMove <= 4'd4;
26        enter_L <= 1'd0;
27        @(posedge clock) // #10 C_5_I
28        @(posedge clock) // #30
29        enter_L <= 1'd1;
30        @(posedge clock) // #50 C_5
31        hMove <= 4'd6;
32        @(posedge clock) // #70 C_5
33        enter_L <= 1'd0;
34        @(posedge clock) // #90 C_5_H_6_E
35        @(posedge clock) // #110 C_5_H_6_E
36        enter_L <= 1'd1;
37        @(posedge clock) // #130 C_1_5_H_6
38
39        enter_L <= 1'd0;
40        @(posedge clock) // #150 C_1_5_H_6_I
41        enter_L <= 1'd1;
42        @(posedge clock) // #170 C_1_5_H_6
43
44        hMove <= 4'd9;
45        @(posedge clock); // #190 C_1_5_H_6
46        enter_L <= 1'd0;
47        @(posedge clock); // #210 C_1_5_H_6_9_E
48        enter_L <= 1'd1;
49        @(posedge clock); // #230 C_1_3_5_H_6_9
50        enter_L <= 1'd0;
51        @(posedge clock); // #250 C_1_3_5_H_6_9_I
52        @(posedge clock); // #270 C_1_3_5_H_6_9_I
53        enter_L <= 1'd1;
54        @(posedge clock); // #290 C_1_3_5_H_6_9
55        hMove <= 4'd2;
56        @(posedge clock); // #310 C_1_3_5_H_6_9
57        enter_L <= 1'd0;
58        @(posedge clock); // #330 C_1_3_5_H_2_6_9_E
59        @(posedge clock); // #350 C_1_3_5_H_2_6_9_E
60        enter_L <= 1'd1;
61        @(posedge clock); // #370 C_1_3_5_7_H_2_6_9_W
62        @(posedge clock); // #390 C_1_3_5_7_H_2_6_9_W
63        newGame_L <= 1'd0;
64        @(posedge clock); // #410 C_1_3_5_7_H_2_6_9_W_N
65        @(posedge clock); // #430 C_1_3_5_7_H_2_6_9_W_N
66        newGame_L <= 1'd1;
67        @(posedge clock); // #450 C_5
68        hMove <= 4'd6;
69        enter_L <= 1'd0;

```

```
70    @(posedge clock); // #470 C_5_H_6_E
71    enter_L <= 1'd1;
72    @(posedge clock); // #490 C_1_5_H_6
73    enter_L <= 1'd0;
74    @(posedge clock); // #470 C_5_H_6_E
75    enter_L <= 1'd1;
76    @(posedge clock); // #490 C_1_5_H_6
77    enter_L <= 1'd0;
78    @(posedge clock); // #470 C_5_H_6_E
79    enter_L <= 1'd1;
80    @(posedge clock); // #490 C_1_5_H_6
81    enter_L <= 1'd0;
82    @(posedge clock); // #470 C_5_H_6_E
83    enter_L <= 1'd1;
84    @(posedge clock); // #490 C_1_5_H_6
85    enter_L <= 1'd0;
86    @(posedge clock); // #470 C_5_H_6_E
87    enter_L <= 1'd1;
88    @(posedge clock); // #490 C_1_5_H_6
89    enter_L <= 1'd0;
90    @(posedge clock); // #470 C_5_H_6_E
91    enter_L <= 1'd1;
92    @(posedge clock); // #490 C_1_5_H_6
93    enter_L <= 1'd0;
94    @(posedge clock); // #470 C_5_H_6_E
95    enter_L <= 1'd1;
96    @(posedge clock); // #490 C_1_5_H_6
97    enter_L <= 1'd0;
98    @(posedge clock); // #470 C_5_H_6_E
99    enter_L <= 1'd1;
100   @(posedge clock); // #490 C_1_5_H_6
101   enter_L <= 1'd0;
102   @(posedge clock); // #470 C_5_H_6_E
103   enter_L <= 1'd1;
104   @(posedge clock); // #490 C_1_5_H_6
105   enter_L <= 1'd0;
106   @(posedge clock); // #470 C_5_H_6_E
107   enter_L <= 1'd1;
108   @(posedge clock); // #490 C_1_5_H_6
109   enter_L <= 1'd0;
110   @(posedge clock); // #470 C_5_H_6_E
111   enter_L <= 1'd1;
112   @(posedge clock); // #490 C_1_5_H_6
113   enter_L <= 1'd0;
114   @(posedge clock); // #470 C_5_H_6_E
115   enter_L <= 1'd1;
116   @(posedge clock); // #490 C_1_5_H_6
117   enter_L <= 1'd0;
118   @(posedge clock); // #470 C_5_H_6_E
119   enter_L <= 1'd1;
120   @(posedge clock); // #490 C_1_5_H_6
121   enter_L <= 1'd0;
122   @(posedge clock); // #470 C_5_H_6_E
123   enter_L <= 1'd1;
124   @(posedge clock); // #490 C_1_5_H_6
125   enter_L <= 1'd0;
126   @(posedge clock); // #470 C_5_H_6_E
127   enter_L <= 1'd1;
128   @(posedge clock); // #490 C_1_5_H_6
129   enter_L <= 1'd0;
130   @(posedge clock); // #470 C_5_H_6_E
131   enter_L <= 1'd1;
132   @(posedge clock); // #490 C_1_5_H_6
133   enter_L <= 1'd0;
134   @(posedge clock); // #470 C_5_H_6_E
135   enter_L <= 1'd1;
136   @(posedge clock); // #490 C_1_5_H_6
137   enter_L <= 1'd0;
138   @(posedge clock); // #470 C_5_H_6_E
139   enter_L <= 1'd1;
140   @(posedge clock); // #490 C_1_5_H_6
```

```
141     enter_L <= 1'd0;
142     @(posedge clock); // #470 C_5_H_6_E
143     enter_L <= 1'd1;
144     @(posedge clock); // #490 C_1_5_H_6
145     enter_L <= 1'd0;
146     @(posedge clock); // #470 C_5_H_6_E
147     enter_L <= 1'd1;
148     @(posedge clock); // #490 C_1_5_H_6
149     enter_L <= 1'd0;
150     @(posedge clock); // #470 C_5_H_6_E
151     enter_L <= 1'd1;
152     @(posedge clock); // #490 C_1_5_H_6
153     enter_L <= 1'd0;
154     @(posedge clock); // #470 C_5_H_6_E
155     enter_L <= 1'd1;
156     @(posedge clock); // #490 C_1_5_H_6
157     enter_L <= 1'd0;
158     @(posedge clock); // #470 C_5_H_6_E
159     enter_L <= 1'd1;
160     @(posedge clock); // #490 C_1_5_H_6
161     enter_L <= 1'd0;
162     @(posedge clock); // #470 C_5_H_6_E
163     enter_L <= 1'd1;
164     @(posedge clock); // #490 C_1_5_H_6
165     enter_L <= 1'd0;
166     @(posedge clock); // #470 C_5_H_6_E
167     enter_L <= 1'd1;
168     @(posedge clock); // #490 C_1_5_H_6
169     enter_L <= 1'd0;
170     @(posedge clock); // #470 C_5_H_6_E
171     enter_L <= 1'd1;
172     @(posedge clock); // #490 C_1_5_H_6
173     enter_L <= 1'd0;
174     @(posedge clock); // #470 C_5_H_6_E
175     enter_L <= 1'd1;
176     @(posedge clock); // #490 C_1_5_H_6
177     enter_L <= 1'd0;
178     @(posedge clock); // #470 C_5_H_6_E
179     enter_L <= 1'd1;
180     @(posedge clock); // #490 C_1_5_H_6
181     enter_L <= 1'd0;
182     @(posedge clock); // #470 C_5_H_6_E
183     enter_L <= 1'd1;
184     @(posedge clock); // #490 C_1_5_H_6
185     enter_L <= 1'd0;
186     @(posedge clock); // #470 C_5_H_6_E
187     enter_L <= 1'd1;
188     @(posedge clock); // #490 C_1_5_H_6
189     enter_L <= 1'd0;
190     @(posedge clock); // #470 C_5_H_6_E
191     enter_L <= 1'd1;
192     @(posedge clock); // #490 C_1_5_H_6
193     enter_L <= 1'd0;
194     @(posedge clock); // #470 C_5_H_6_E
195     enter_L <= 1'd1;
196     @(posedge clock); // #490 C_1_5_H_6
197     enter_L <= 1'd0;
198     @(posedge clock); // #470 C_5_H_6_E
199     enter_L <= 1'd1;
200     @(posedge clock); // #490 C_1_5_H_6
201     enter_L <= 1'd0;
202     @(posedge clock); // #470 C_5_H_6_E
203     enter_L <= 1'd1;
204     @(posedge clock); // #490 C_1_5_H_6
205     enter_L <= 1'd0;
206     @(posedge clock); // #470 C_5_H_6_E
207     enter_L <= 1'd1;
208     @(posedge clock); // #490 C_1_5_H_6
209     enter_L <= 1'd0;
210     @(posedge clock); // #470 C_5_H_6_E
211     enter_L <= 1'd1;
```

```
212     @(posedge clock); // #490 C_1_5_H_6
213     enter_L <= 1'd0;
214     @(posedge clock); // #470 C_5_H_6_E
215     enter_L <= 1'd1;
216     @(posedge clock); // #490 C_1_5_H_6
217     enter_L <= 1'd0;
218     @(posedge clock); // #470 C_5_H_6_E
219     enter_L <= 1'd1;
220     @(posedge clock); // #490 C_1_5_H_6
221     enter_L <= 1'd0;
222     @(posedge clock); // #470 C_5_H_6_E
223     enter_L <= 1'd1;
224     @(posedge clock); // #490 C_1_5_H_6
225     enter_L <= 1'd0;
226     @(posedge clock); // #470 C_5_H_6_E
227     enter_L <= 1'd1;
228     @(posedge clock); // #490 C_1_5_H_6
229     #1 $finish;
230 end
231 endmodule : testbench
```

Lab Code [15 points]
Filename: 05_theBigGame.sv
AndrewID: ocarroll

```
1 `default_nettype none
2
3 module task5 (
4     output logic [3:0] cMove,
5     output logic      win,
6     output logic [3:0] h3, h2, h1, h0, c3, c2, c1, c0,
7     input logic [3:0] hMove,
8     input logic      clock, reset, enter_L, newGame_L);
9
10 // C_1_2_H_3_4_E_I means that
11 // computer's move: 1, 2
12 // human move: 3, 4
13 // enter_L pressed
14 // this is an invalid move
15 // 'W' means that computer wins
16 // 'N' means that newGame_L pressed
17 enum logic [5:0] {
18     C_5 = 6'd0,
19     C_5_I = 6'd1,
20     C_5_H_6_E = 6'd2,
21     C_1_5_H_6 = 6'd3,
22     C_1_5_H_6_I = 6'd4,
23     C_1_5_H_2_6_E = 6'd5,
24     C_1_5_H_3_6_E = 6'd6,
25     C_1_5_H_4_6_E = 6'd7,
26     C_1_5_H_6_7_E = 6'd8,
27     C_1_5_H_6_8_E = 6'd9,
28     C_1_5_H_6_9_E = 6'd10,
29     C_1_5_9_H_2_6_W = 6'd11,
30     C_1_5_9_H_2_6_W_N = 6'd12,
31     C_1_5_9_H_3_6_W = 6'd13,
32     C_1_5_9_H_3_6_W_N = 6'd14,
33     C_1_5_9_H_4_6_W = 6'd15,
34     C_1_5_9_H_4_6_W_N = 6'd16,
35     C_1_5_9_H_6_7_W = 6'd17,
36     C_1_5_9_H_6_7_W_N = 6'd18,
37     C_1_5_9_H_6_8_W = 6'd19,
38     C_1_5_9_H_6_8_W_N = 6'd20,
39     C_1_3_5_H_6_9 = 6'd21,
40     C_1_3_5_H_6_9_I = 6'd22,
41     C_1_3_5_H_2_6_9_E = 6'd23,
42     C_1_3_5_H_4_6_9_E = 6'd24,
43     C_1_3_5_H_6_7_9_E = 6'd25,
44     C_1_3_5_H_6_8_9_E = 6'd26,
45     C_1_2_3_5_H_4_6_9_W = 6'd27,
46     C_1_2_3_5_H_4_6_9_W_N = 6'd28,
47     C_1_2_3_5_H_6_7_9_W = 6'd29,
48     C_1_2_3_5_H_6_7_9_W_N = 6'd30,
49     C_1_2_3_5_H_6_8_9_W = 6'd31,
50     C_1_2_3_5_H_6_8_9_W_N = 6'd32,
51     C_1_3_5_7_H_2_6_9_W = 6'd33,
52     C_1_3_5_7_H_2_6_9_W_N = 6'd34
53 } currState, nextState;
54
55 // next state generation
56 always_comb begin
57     unique case (currState)
58         C_5: begin
59             if (~enter_L) begin
60                 if (hMove != 4'd6)
61                     nextState = C_5_I;
62                 else
63                     nextState = C_5_H_6_E;
64             end
65         else
66             nextState = C_5;
67     end
68     C_1_5_H_6: begin
69         if (~enter_L)
```

```

70         unique case (hMove)
71             4'd1,
72             4'd5,
73             4'd6: nextState = C_1_5_H_6_I;
74             4'd2: nextState = C_1_5_H_2_6_E;
75             4'd3: nextState = C_1_5_H_3_6_E;
76             4'd4: nextState = C_1_5_H_4_6_E;
77             4'd7: nextState = C_1_5_H_6_7_E;
78             4'd8: nextState = C_1_5_H_6_8_E;
79             4'd9: nextState = C_1_5_H_6_9_E;
80         endcase
81     else
82         nextState = C_1_5_H_6;
83 end
84 C_1_3_5_H_6_9:
85     if (~enter_L)
86         unique case (hMove)
87             4'd1,
88             4'd3,
89             4'd5,
90             4'd6,
91             4'd9: nextState = C_1_3_5_H_6_9_I;
92             4'd2: nextState = C_1_3_5_H_2_6_9_E;
93             4'd4: nextState = C_1_3_5_H_4_6_9_E;
94             4'd7: nextState = C_1_3_5_H_6_7_9_E;
95             4'd8: nextState = C_1_3_5_H_6_8_9_E;
96         endcase
97     else
98         nextState = C_1_3_5_H_6_9;
99 C_5_H_6_E:
100     if (enter_L)
101         nextState = C_1_5_H_6;
102     else
103         nextState = C_5_H_6_E;
104 C_1_5_H_2_6_E:
105     if (enter_L)
106         nextState = C_1_5_9_H_2_6_W;
107     else
108         nextState = C_1_5_H_2_6_E;
109 C_1_5_H_3_6_E:
110     if (enter_L)
111         nextState = C_1_5_9_H_3_6_W;
112     else
113         nextState = C_1_5_H_3_6_E;
114 C_1_5_H_4_6_E:
115     if (enter_L)
116         nextState = C_1_5_9_H_4_6_W;
117     else
118         nextState = C_1_5_H_4_6_E;
119 C_1_5_H_6_7_E:
120     if (enter_L)
121         nextState = C_1_5_9_H_6_7_W;
122     else
123         nextState = C_1_5_H_6_7_E;
124 C_1_5_H_6_8_E:
125     if (enter_L)
126         nextState = C_1_5_9_H_6_8_W;
127     else
128         nextState = C_1_5_H_6_8_E;
129 C_1_5_H_6_9_E:
130     if (enter_L)
131         nextState = C_1_3_5_H_6_9;
132     else
133         nextState = C_1_5_H_6_9_E;
134 C_1_3_5_H_2_6_9_E:
135     if (enter_L)
136         nextState = C_1_3_5_7_H_2_6_9_W;
137     else
138         nextState = C_1_3_5_H_2_6_9_E;
139 C_1_3_5_H_4_6_9_E:
140     if (enter_L)

```

```
141         nextState = C_1_2_3_5_H_4_6_9_W;
142     else
143         nextState = C_1_3_5_H_4_6_9_E;
144 C_1_3_5_H_6_7_9_E:
145     if (enter_L)
146         nextState = C_1_2_3_5_H_6_7_9_W;
147     else
148         nextState = C_1_3_5_H_6_7_9_E;
149 C_1_3_5_H_6_8_9_E:
150     if (enter_L)
151         nextState = C_1_2_3_5_H_6_8_9_W;
152     else
153         nextState = C_1_3_5_H_6_8_9_E;
154
155 C_1_5_9_H_2_6_W:
156     if (~newGame_L)
157         nextState = C_1_5_9_H_2_6_W_N;
158     else
159         nextState = C_1_5_9_H_2_6_W;
160 C_1_5_9_H_3_6_W:
161     if (~newGame_L)
162         nextState = C_1_5_9_H_3_6_W_N;
163     else
164         nextState = C_1_5_9_H_3_6_W;
165 C_1_5_9_H_4_6_W:
166     if (~newGame_L)
167         nextState = C_1_5_9_H_4_6_W_N;
168     else
169         nextState = C_1_5_9_H_4_6_W;
170 C_1_5_9_H_6_7_W:
171     if (~newGame_L)
172         nextState = C_1_5_9_H_6_7_W_N;
173     else
174         nextState = C_1_5_9_H_6_7_W;
175 C_1_5_9_H_6_8_W:
176     if (~newGame_L)
177         nextState = C_1_5_9_H_6_8_W_N;
178     else
179         nextState = C_1_5_9_H_6_8_W;
180 C_1_3_5_7_H_2_6_9_W:
181     if (~newGame_L)
182         nextState = C_1_3_5_7_H_2_6_9_W_N;
183     else
184         nextState = C_1_3_5_7_H_2_6_9_W;
185 C_1_2_3_5_H_4_6_9_W:
186     if (~newGame_L)
187         nextState = C_1_2_3_5_H_4_6_9_W_N;
188     else
189         nextState = C_1_2_3_5_H_4_6_9_W;
190 C_1_2_3_5_H_6_7_9_W:
191     if (~newGame_L)
192         nextState = C_1_2_3_5_H_6_7_9_W_N;
193     else
194         nextState = C_1_2_3_5_H_6_7_9_W;
195 C_1_2_3_5_H_6_8_9_W:
196     if (~newGame_L)
197         nextState = C_1_2_3_5_H_6_8_9_W_N;
198     else
199         nextState = C_1_2_3_5_H_6_8_9_W;
200
201 C_5_I:
202     if (enter_L)
203         nextState = C_5;
204     else
205         nextState = C_5_I;
206 C_1_5_H_6_I:
207     if (enter_L)
208         nextState = C_1_5_H_6;
209     else
210         nextState = C_1_5_H_6_I;
211 C_1_3_5_H_6_9_I:
```



```

212         if (enter_L)
213             nextState = C_1_3_5_H_6_9;
214         else
215             nextState = C_1_3_5_H_6_9_I;
216
217     C_1_5_9_H_2_6_W_N:
218         if (newGame_L)
219             nextState = C_5;
220         else
221             nextState = C_1_5_9_H_2_6_W_N;
222     C_1_5_9_H_3_6_W_N:
223         if (newGame_L)
224             nextState = C_5;
225         else
226             nextState = C_1_5_9_H_3_6_W_N;
227     C_1_5_9_H_4_6_W_N:
228         if (newGame_L)
229             nextState = C_5;
230         else
231             nextState = C_1_5_9_H_4_6_W_N;
232     C_1_5_9_H_6_7_W_N:
233         if (newGame_L)
234             nextState = C_5;
235         else
236             nextState = C_1_5_9_H_6_7_W_N;
237     C_1_5_9_H_6_8_W_N:
238         if (newGame_L)
239             nextState = C_5;
240         else
241             nextState = C_1_5_9_H_6_8_W_N;
242     C_1_2_3_5_H_4_6_9_W_N:
243         if (newGame_L)
244             nextState = C_5;
245         else
246             nextState = C_1_2_3_5_H_4_6_9_W_N;
247     C_1_2_3_5_H_6_7_9_W_N:
248         if (newGame_L)
249             nextState = C_5;
250         else
251             nextState = C_1_2_3_5_H_6_7_9_W_N;
252     C_1_2_3_5_H_6_8_9_W_N:
253         if (newGame_L)
254             nextState = C_5;
255         else
256             nextState = C_1_2_3_5_H_6_8_9_W_N;
257     C_1_3_5_7_H_2_6_9_W_N:
258         if (newGame_L)
259             nextState = C_5;
260         else
261             nextState = C_1_3_5_7_H_2_6_9_W_N;
262     endcase
263 end
264
265
266
267 always_comb begin
268     cMove = 4'b0000;
269     win = 1'b1;
270     {c3, c2, c1, c0, h3, h2, h1, h0} = 32'b0;
271     if (currState == C_5) begin
272         cMove = 4'h5;
273         win = 0;
274         c3 = 4'd5;
275     end
276
277     if (currState == C_5_I) begin
278         cMove = 4'h5;
279         win = 0;
280         c3 = 4'd5;
281     end
282

```

```
283     if (currState == C_5_H_6_E) begin
284         cMove = 4'h5;
285         win = 0;
286         c3 = 4'd5;
287     end
288
289     if (currState == C_1_5_H_6) begin
290         cMove = 4'h1;
291         win = 0;
292         c3 = 4'd1;
293         c2 = 4'd5;
294         h3 = 4'd6;
295     end
296
297     if (currState == C_1_5_H_6_I) begin
298         cMove = 4'h1;
299         win = 0;
300         c3 = 4'd1;
301         c2 = 4'd5;
302         h3 = 4'd6;
303     end
304
305
306     if (currState == C_1_5_H_2_6_E) begin
307         cMove = 4'h1;
308         win = 0;
309         c3 = 4'd1;
310         c2 = 4'd5;
311         h3 = 4'd6;
312     end
313     if (currState == C_1_5_H_3_6_E) begin
314         cMove = 4'h1;
315         win = 0;
316         c3 = 4'd1;
317         c2 = 4'd5;
318         h3 = 4'd6;
319     end
320     if (currState == C_1_5_H_4_6_E) begin
321         cMove = 4'h1;
322         win = 0;
323         c3 = 4'd1;
324         c2 = 4'd5;
325         h3 = 4'd6;
326     end
327     if (currState == C_1_5_H_6_7_E) begin
328         cMove = 4'h1;
329         win = 0;
330         c3 = 4'd1;
331         c2 = 4'd5;
332         h3 = 4'd6;
333     end
334     if (currState == C_1_5_H_6_8_E) begin
335         cMove = 4'h1;
336         win = 0;
337         c3 = 4'd1;
338         c2 = 4'd5;
339         h3 = 4'd6;
340     end
341     if (currState == C_1_5_H_6_9_E) begin
342         cMove = 4'h1;
343         win = 0;
344         c3 = 4'd1;
345         c2 = 4'd5;
346         h3 = 4'd6;
347     end
348
349
350     if (currState == C_1_5_9_H_2_6_W) begin
351         cMove = 4'h7;
352         win = 1;
353         c3 = 4'd1;
```

```
354     c2 = 4'd5;
355     c1 = 4'd9;
356     h3 = 4'd2;
357     h2 = 4'd6;
358 end
359 if (currState == C_1_5_9_H_2_6_W_N) begin
360     cMove = 4'h9;
361     win = 1;
362     c3 = 4'd1;
363     c2 = 4'd5;
364     c1 = 4'd9;
365     h3 = 4'd2;
366     h2 = 4'd6;
367 end
368
369 if (currState == C_1_5_9_H_3_6_W) begin
370     cMove = 4'h9;
371     win = 1;
372     c3 = 4'd1;
373     c2 = 4'd5;
374     c1 = 4'd7;
375     h3 = 4'd3;
376     h2 = 4'd6;
377 end
378 if (currState == C_1_5_9_H_3_6_W_N) begin
379     cMove = 4'h9;
380     win = 1;
381     c3 = 4'd1;
382     c2 = 4'd5;
383     c1 = 4'd9;
384     h3 = 4'd3;
385     h2 = 4'd6;
386 end
387
388 if (currState == C_1_5_9_H_4_6_W) begin
389     cMove = 4'h9;
390     win = 1;
391     c3 = 4'd1;
392     c2 = 4'd5;
393     c1 = 4'd9;
394     h3 = 4'd4;
395     h2 = 4'd6;
396 end
397 if (currState == C_1_5_9_H_4_6_W_N) begin
398     cMove = 4'h9;
399     win = 1;
400     c3 = 4'd1;
401     c2 = 4'd5;
402     c1 = 4'd9;
403     h3 = 4'd4;
404     h2 = 4'd6;
405 end
406
407 if (currState == C_1_5_9_H_6_7_W) begin
408     cMove = 4'h9;
409     win = 1;
410     c3 = 4'd1;
411     c2 = 4'd5;
412     c1 = 4'd9;
413     h3 = 4'd6;
414     h2 = 4'd7;
415 end
416 if (currState == C_1_5_9_H_6_7_W_N) begin
417     cMove = 4'h9;
418     win = 1;
419     c3 = 4'd1;
420     c2 = 4'd5;
421     c1 = 4'd9;
422     h3 = 4'd6;
423     h2 = 4'd7;
424 end
```

```
425
426
427
428     if (currState == C_1_5_9_H_6_8_W) begin
429         cMove = 4'h9;
430         win = 1;
431         c3 = 4'd1;
432         c2 = 4'd5;
433         c1 = 4'd9;
434         h3 = 4'd6;
435         h2 = 4'd8;
436     end
437     if (currState == C_1_5_9_H_6_8_W_N) begin
438         cMove = 4'h9;
439         win = 1;
440         c3 = 4'd1;
441         c2 = 4'd5;
442         c1 = 4'd9;
443         h3 = 4'd6;
444         h2 = 4'd8;
445     end
446
447
448     if (currState == C_1_3_5_H_6_9) begin
449         cMove = 4'h3;
450         win = 0;
451         c3 = 4'd1;
452         c2 = 4'd3;
453         c1 = 4'd5;
454         h3 = 4'd6;
455         h2 = 4'd9;
456     end
457     if (currState == C_1_3_5_H_6_9_I) begin
458         cMove = 4'h3;
459         win = 0;
460         c3 = 4'd1;
461         c2 = 4'd3;
462         c1 = 4'd5;
463         h3 = 4'd6;
464         h2 = 4'd9;
465     end
466
467     if (currState == C_1_3_5_H_2_6_9_E) begin
468         cMove = 4'h3;
469         win = 0;
470         c3 = 4'd1;
471         c2 = 4'd3;
472         c1 = 4'd5;
473         h3 = 4'd6;
474         h2 = 4'd9;
475     end
476
477     if (currState == C_1_3_5_H_4_6_9_E) begin
478         cMove = 4'h3;
479         win = 0;
480         c3 = 4'd1;
481         c2 = 4'd3;
482         c1 = 4'd5;
483         h3 = 4'd6;
484         h2 = 4'd9;
485     end
486
487     if (currState == C_1_3_5_H_6_7_9_E) begin
488         cMove = 4'h3;
489         win = 0;
490         c3 = 4'd1;
491         c2 = 4'd3;
492         c1 = 4'd5;
493         h3 = 4'd6;
494         h2 = 4'd9;
495     end
```

```
496
497   if (currState == C_1_3_5_H_6_8_9_E) begin
498       cMove = 4'h3;
499       win = 0;
500       c3 = 4'd1;
501       c2 = 4'd3;
502       c1 = 4'd5;
503       h3 = 4'd6;
504       h2 = 4'd9;
505   end
506
507   if (currState == C_1_2_3_5_H_4_6_9_W) begin
508       cMove = 4'h2;
509       win = 1;
510       c3 = 4'd1;
511       c2 = 4'd2;
512       c1 = 4'd3;
513       c0 = 4'd5;
514       h3 = 4'd4;
515       h2 = 4'd6;
516       h1 = 4'd9;
517   end
518
519   if (currState == C_1_2_3_5_H_4_6_9_W_N) begin
520       cMove = 4'h2;
521       win = 1;
522       c3 = 4'd1;
523       c2 = 4'd2;
524       c1 = 4'd3;
525       c0 = 4'd5;
526       h3 = 4'd4;
527       h2 = 4'd6;
528       h1 = 4'd9;
529   end
530
531   if (currState == C_1_2_3_5_H_6_7_9_W) begin
532       cMove = 4'h2;
533       win = 1;
534       c3 = 4'd1;
535       c2 = 4'd2;
536       c1 = 4'd3;
537       c0 = 4'd5;
538       h3 = 4'd6;
539       h2 = 4'd7;
540       h1 = 4'd9;
541   end
542
543   if (currState == C_1_2_3_5_H_6_7_9_W_N) begin
544       cMove = 4'h2;
545       win = 1;
546       c3 = 4'd1;
547       c2 = 4'd2;
548       c1 = 4'd3;
549       c0 = 4'd5;
550       h3 = 4'd6;
551       h2 = 4'd7;
552       h1 = 4'd9;
553   end
554
555
556   if (currState == C_1_2_3_5_H_6_8_9_W) begin
557       cMove = 4'h2;
558       win = 1;
559       c3 = 4'd1;
560       c2 = 4'd2;
561       c1 = 4'd3;
562       c0 = 4'd5;
563       h3 = 4'd6;
564       h2 = 4'd8;
565       h1 = 4'd9;
566   end
```

```
567
568     if (currState == C_1_2_3_5_H_6_8_9_W_N) begin
569         cMove = 4'h2;
570         win = 1;
571         c3 = 4'd1;
572         c2 = 4'd2;
573         c1 = 4'd3;
574         c0 = 4'd5;
575         h3 = 4'd6;
576         h2 = 4'd8;
577         h1 = 4'd9;
578     end
579
580     if (currState == C_1_3_5_7_H_2_6_9_W) begin
581         cMove = 4'h7;
582         win = 1;
583         c3 = 4'd1;
584         c2 = 4'd3;
585         c1 = 4'd5;
586         c0 = 4'd7;
587         h3 = 4'd2;
588         h2 = 4'd6;
589         h1 = 4'd9;
590     end
591
592     if (currState == C_1_3_5_7_H_2_6_9_W_N) begin
593         cMove = 4'h7;
594         win = 1;
595         c3 = 4'd1;
596         c2 = 4'd3;
597         c1 = 4'd5;
598         c0 = 4'd7;
599         h3 = 4'd2;
600         h2 = 4'd6;
601         h1 = 4'd9;
602     end
603 end
604
605 // register
606 always_ff @(posedge clock, posedge reset)
607     if (reset)
608         currState <= C_5;
609     else
610         currState <= nextState;
611
612 endmodule: task5
613
614
```

Lab Code [15 points]
Filename: chipInterface.sv
AndrewID: ocarroll

```
1 `default_nettype none // Required in every sv file
2 module chipInterface
3     (input logic [3:0] KEY,
4      input logic [17:0] SW,
5      input logic CLOCK_50,
6      output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, HEX6, HEX7,
7      output logic [7:0] LEDG,
8      output logic [3:0] LEDR);
9     logic [3:0] h_3, h_2, h_1, h_0, c_3, c_2, c_1, c_0;
10
11     logic ww;
12
13     task5(.newGame_L(KEY[0]),
14          .clock(CLOCK_50),
15          .h3(h_3),
16          .h2(h_2),
17          .h1(h_1),
18          .h0(h_0),
19          .c3(c_3),
20          .c2(c_2),
21          .c1(c_1),
22          .c0(c_0),
23          .enter_L(KEY[3]),
24          .reset(SW[17]),
25          .hMove(SW[3:0]),
26          .cMove(LEDR[3:0]),
27          .win(ww));
28
29     assign LEDG = {ww, ww, ww, ww, ww, ww, ww, ww};
30     logic [7:0] blank;
31     assign blank = 8'b00000000;
32
33     SevenSegmentDisplay(.BCX0(h_3), .blank(blank), .HEX0(HEX7));
34     SevenSegmentDisplay(.BCX0(h_2), .blank(blank), .HEX0(HEX6));
35     SevenSegmentDisplay(.BCX0(h_1), .blank(blank), .HEX0(HEX5));
36     SevenSegmentDisplay(.BCX0(h_0), .blank(blank), .HEX0(HEX4));
37     SevenSegmentDisplay(.BCX0(c_3), .blank(blank), .HEX0(HEX3));
38     SevenSegmentDisplay(.BCX0(c_2), .blank(blank), .HEX0(HEX2));
39     SevenSegmentDisplay(.BCX0(c_1), .blank(blank), .HEX0(HEX1));
40     SevenSegmentDisplay(.BCX0(c_0), .blank(blank), .HEX0(HEX0));
41
42
43 endmodule: chipInterface
44
45
```