```
1  //////////////////////////////////////////////////////////////////////////////////
…  //////////
2  // Author:          Danny Pan, Mackenzie Collins
3  // Create Date:     02/17/2008, 2/6/2012
4  // File Name:       ee201_numlock_top.v [EXERCISE given to studnents]
5  // Description:
6  //
7  //
8  // Revision:        2.1
9  // Additional Comments: Students: Search for the "TODO" sections and
…  complete them.
10 //                                There are about eleven "TODO" sections.
11 //////////////////////////////////////////////////////////////////////////////////
…  //////////
12
13 `timescale 1ns / 1ps
14
15 module ee201_numlock_top (
16         MemOE, MemWR, RamCS, FlashCS, QuadSpiFlashCS, // Disable the
…  three memory chips
17
18         ClkPort,                            // the 100 MHz incoming clock
…  signal
19
20         // BtnL, BtnU, BtnD, BtnR,             // the Left, Up, Down, and
…  the Right buttons
21
22         //                                 // the center button (this is our
…  reset in most of our designs)
23         // Sw7, Sw6, Sw5, Sw4, Sw3, Sw2, Sw1, Sw0, // 8 switches
24
25         Ld7, Ld6, Ld5, Ld4, Ld3, Ld2, Ld1, Ld0, // 8 LEDs
26         An3, An2, An1, An0,                 // 4 anodes
27         Ca, Cb, Cc, Cd, Ce, Cf, Cg,         // 7 cathodes
28         Dp                                  // Dot Point Cathode on SSDs
29
30         BtnC, BtnL, BtnR                     //reset, U, Z
31       );
32
33
34     /*  INPUTS */
35     // Clock & Reset I/O
36     input       ClkPort;
37 // TODO: DEFINE THE INPUTS (buttons and switches) you need for this
…  project – done
```

```
38 // make sure to add those to the ee201_numlock_top PORT list also!
39     // Project Specific Inputs
40     input   Clk, reset, U, Z;
41
42
43
44     /*  OUTPUTS */
45     // Control signals on Memory chips  (to disable them)
46     output  MemOE, MemWR, RamCS, FlashCS, QuadSpiFlashCS;
47     // Project Specific Outputs
48     // LEDs
49     output  Ld0, Ld1, Ld2, Ld3, Ld4, Ld5, Ld6, Ld7;
50     // SSD Outputs
51     output  Cg, Cf, Ce, Cd, Cc, Cb, Ca, Dp;
52     output  An0, An1, An2, An3;
53
54
55
56     /*  LOCAL SIGNALS */
57     wire            reset, ClkPort;
58     wire            board_clk, sys_clk;
59     wire [1:0]      ssdscan_clk;
60     reg [26:0]      DIV_CLK;
61     wire            U, Z;
62     wire            q_I, q_G1get, q_G1, q_G10get, q_G10, q_G101get,
… q_G101, q_G1011get, q_G1011, q_Opening, q_Bad;
63     wire            Unlock;
64     reg [3:0]       state_num;
65     reg [3:0]       state_sum;
66     wire [3:0]      selected_state;
67     reg             hot1_state_error;
68     reg             selected_state_value;
69     reg [3:0]       SSD;
70     wire [3:0]      SSD0, SSD1, SSD2, SSD3;
71     reg [6:0]       SSD_CATHODES;
72     wire [6:0]      SSD_CATHODES_blinking;
73
74
75 //------------
76 // Disable the three memories so that they do not interfere with the rest
… of the design.
77     assign {MemOE, MemWR, RamCS, FlashCS, QuadSpiFlashCS} = 5'b11111;
78
79
80 //------------
```

```
81 // CLOCK DIVISION
82
83     // The clock division circuitary works like this:
84     //
85     // ClkPort ---> [BUFGP2] ---> board_clk
86     // board_clk ---> [clock dividing counter] ---> DIV_CLK
87     // DIV_CLK ---> [constant assignment] ---> sys_clk;
88
89     BUFGP BUFGP1 (board_clk, ClkPort);
90
91 // As the ClkPort signal travels throughout our design,
92 // it is necessary to provide global routing to this signal.
93 // The BUFGPs buffer these input ports and connect them to the global
94 // routing resources in the FPGA.
95
96     // BUFGP BUFGP2 (reset, BtnC); In the case of Spartan 3E (on Nexys-2
 … board), we were using BUFGP to provide global routing for the reset
 … signal. But Spartan 6 (on Nexys-3) does not allow this.
97     assign reset = BtnC;
98
99 //------------
100     // Our clock is too fast (100MHz) for SSD scanning
101     // create a series of slower "divided" clocks
102     // each successive bit is 1/2 frequency
103 // TODO: create the sensitivity list - done
104     always @ (posedge board_clk, posedge reset)
105     begin : CLOCK_DIVIDER
106       if (reset)
107             DIV_CLK <= 0;
108       else
109             DIV_CLK <= DIV_CLK + 1'b1;
110             // just incrementing makes our life easier
111 // TODO: add the incrementer code - done
112     end
113 //------------
114     // pick a divided clock bit to assign to system clock
115     // your decision should not be "too fast" or you will not see you
 … state machine working
116     assign  sys_clk = DIV_CLK[25]; // DIV_CLK[25] (~1.5Hz) = (100MHz /
 … 2**26)
117
118
119 //------------
120 // INPUT: SWITCHES & BUTTONS
121     // BtnL/BtnR is abstract
```

```
122     // let's form some wire aliases with easier naming (U and Z, for UNO
    … and ZERO)
123
124 // TODO: add the lines to assign your I/O inputs to U and Z - done
125     assign {U,Z} = {BtnL, BtnR};
126
127
128     // switches used to send the value of a specific state to LD6
129
130     assign selected_state = {Sw3, Sw2, Sw1, Sw0};
131
132
133 //------------
134 // DESIGN
135
136
137     ee201_numlock_sm SM1(.Clk(sys_clk), .reset(reset),
138                                  .q_I(q_I), .q_G1get(q_G1get),
    … .q_G1(q_G1), .q_G10get(.q_G10get),
139                                  .q_G10(q_G10), .q_G101get(q(G101get),
    … .q_G101(q_G101),
140                                  .q_G1011get(.q_G1011get),
    … .q_G1011(q_G1011), .q_Opening(q_Opening), .q_Bad(q_Bad));
141 // TODO: finish the port list - done
142 // make sure you are following the naming scheme above
143
144
145     // convert the 1-hot state to a hex-number for easy display
146
147     `define QI_NUM          4'b0000
148     `define QG1GET_NUM      4'b0001
149     `define QG1_NUM         4'b0010
150     `define QG10GET_NUM     4'b0011
151     `define QG10_NUM        4'b0100
152     `define QG101GET_NUM    4'b0101
153     `define QG101_NUM       4'b0110
154     `define QG1011GET_NUM   4'b0111
155     `define QG1011_NUM      4'b1000
156     `define QOPENING_NUM    4'b1001
157     `define QBAD_NUM        4'b1010
158
159     always @ ( q_I, q_G1get, q_G1, q_G10get, q_G10, q_G101get, q_G101,
    … q_G1011get, q_G1011, q_Opening, q_Bad )
160     begin : ONE_HOT_TO_HEX
161         (* full_case, parallel_case *) // to avoid prioritization
```

```
161… (Verilog 2001 standard)
162          case ( {q_I, q_G1get, q_G1, q_G10get, q_G10, q_G101get, q_G101,
  … q_G1011get, q_G1011, q_Opening, q_Bad} )
163
164 // TODO: complete the 1-hot encoder
165              11'b00000000001: state_num = `QI_NUM;
166              11'b00000000010: state_num = `QG1GET_NUM;
167              11'b00000000100: state_num = `QG1_NUM;
168              11'b00000001000: state_num = `QG10GET_NUM;
169              11'b00000010000: state_num = `QG10_NUM;
170              11'b00000100000: state_num = `QG101GET_NUM;
171              11'b00001000000: state_num = `QG101_NUM;
172              11'b00010000000: state_num = `QG1011GET_NUM;
173              11'b00100000000: state_num = `QG1011_NUM;
174              11'b01000000000: state_num = `QOPENING_NUM;
175              11'b10000000000: state_num = `QBAD_NUM;
176          endcase
177      end
178
179
180 //------------
181 // OUTPUT: LEDS
182      assign {Ld7,Ld6,Ld5,Ld4} = state_num;
183
184      // display 1-hot state errors
185      // add all of the state bits.  if the sum != 1 then we have a problem
186      // we need to support 0-10 so sum must be 4-bit
187
188      always @ (q_I, q_G1get, q_G1, q_G10get, q_G10, q_G101get, q_G101,
  … q_G1011get, q_G1011, q_Opening, q_Bad)
189      begin
190      // TODO: finish the logic for state_sum
191          state_sum = {q_I, q_G1get, q_G1, q_G10get, q_G10, q_G101get,
  … q_G101, q_G1011get, q_G1011, q_Opening, q_Bad};
192      end
193
194      // we could do the following with an assign statement also.
195      // Ofcourse, then we need to declare hot1_state_error as a wire.
196      // assign hot1_state_error = (state_sum != 4'b0001) ? 1'b1 : 1'b0;
197      // Or we can avoid this intermediate hot1_state_error altogether!
198      // assign Ld2 = (state_sum != 4'b0001) ? 1'b1 : 1'b0;
199
200      always @ (state_sum)
201      begin
202          hot1_state_error = (state_sum != 4'b0001) ? 1'b1 : 1'b0;
```

```
203     end
204
205     assign Ld2 = hot1_state_error;
206
207     // display the value of selected state
208
209     always @ ( selected_state, q_I, q_G1get, q_G1, q_G10get, q_G10,
… q_G101get, q_G101, q_G1011get, q_G1011, q_Opening, q_Bad )
210     begin : SELECTED_STATE_VALUE
211         (* full_case, parallel_case *) // to avoid prioritization
… (Verilog 2001 standard)
212         case ( selected_state )
213             `QI_NUM:            selected_state_value = q_I;
214             `QG1GET_NUM:        selected_state_value = q_G1get;
215             `QG1_NUM:           selected_state_value = q_G1;
216             `QG10GET_NUM:       selected_state_value = q_G10get;
217             `QG10_NUM:          selected_state_value = q_G10;
218             `QG101GET_NUM:      selected_state_value = q_G101get;
219             `QG101_NUM:         selected_state_value = q_G101;
220             `QG1011GET_NUM:     selected_state_value = q_G1011get;
221             `QG1011_NUM:        selected_state_value = q_G1011;
222             `QOPENING_NUM:      selected_state_value = q_Opening;
223             `QBAD_NUM:          selected_state_value = q_Bad;
224         endcase
225     end
226     assign Ld3 = selected_state_value;
227
228     assign {Ld1, Ld0} = {U, Z};
229
230
231
232 //------------
233 // SSD (Seven Segment Display)
234
235 // TODO: finish the assignment for SSD3, SSD2, SSD1
236     assign SSD3 = (2**state_num)/(16**2);
237     assign SSD2 = (2**state_num)/(16);
238     assign SSD1 = (2**state_num)%16;
239     assign SSD0 = state_num;
240
241
242     // need a scan clk for the seven segment display
243
244     // 100 MHz / 2^18 = 381.5 cycles/sec ==> frequency of DIV_CLK[17]
245     // 100 MHz / 2^19 = 190.7 cycles/sec ==> frequency of DIV_CLK[18]
```

```
246     // 100 MHz / 2^20 =  95.4 cycles/sec ==> frequency of DIV_CLK[19]
247
248     // 381.5 cycles/sec (2.62 ms per digit) [which means all 4 digits are
…   lit once every 10.5 ms (reciprocal of 95.4 cycles/sec)] works well.
249
250     //                  --|  |--|  |--|  |--|  |--|  |--|  |--|  |--|  |
251     //                    |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
252     //  DIV_CLK[17]       |__|  |__|  |__|  |__|  |__|  |__|  |__|  |__|
253     //
254     //                -----|     |-----|     |-----|     |-----|     |
255     //                     |  0  |  1  |  0  |  1  |     |     |     |
256     //  DIV_CLK[18]        |_____|     |_____|     |_____|     |_____|
257     //
258     //          -----------|           |-----------|           |
259     //                     |  0     0  |  1     1  |           |
260     //  DIV_CLK[19]        |___________|           |___________|
261     //
262
263     assign ssdscan_clk = DIV_CLK[19:18];
264
265     assign An0  = !(~(ssdscan_clk[1]) && ~(ssdscan_clk[0]));  // when
…   ssdscan_clk = 00
266     assign An1  = !(~(ssdscan_clk[1]) &&  (ssdscan_clk[0]));  // when
…   ssdscan_clk = 01
267     assign An2  = !( (ssdscan_clk[1]) && ~(ssdscan_clk[0]));  // when
…   ssdscan_clk = 10
268     assign An3  = !( (ssdscan_clk[1]) &&  (ssdscan_clk[0]));  // when
…   ssdscan_clk = 11
269
270
271     always @ (ssdscan_clk, SSD0, SSD1, SSD2, SSD3)
272     begin : SSD_SCAN_OUT
273         case (ssdscan_clk)
274
275 // TODO: finish the multiplexor to scan through SSD0-SSD3 with
…   ssdscan_clk[1:0]
276         2'b00: SSD = SSD0;
277         2'b01: SSD = SSD1;
278         2'b10: SSD = SSD2;
279         2'b11: SSD = SSD3;
280         endcase
281     end
282
283
284     // and finally convert SSD_num to ssd
```

```
285 // TODO: write the code to enable "blinking"
286     // we want the CATHODES to turn "on-off-on-off" with system clock
287     // while we are in state: OPENING
288     assign SSD_CATHODES_blinking = SSD_CATHODES | ( {7{q_Opening &
  … ~sys_clk}} );
289     assign {Ca, Cb, Cc, Cd, Ce, Cf, Cg, Dp} = {SSD_CATHODES_blinking,
  … 1'b1};
290
291     // Following is Hex-to-SSD conversion. Even though
292     always @ (SSD)
293     begin : HEX_TO_SSD
294         case (SSD)
295 // TODO: write cases for 0-9
296             4'b0001: SSD_CATHODES = 7'b0000001 ; // 1
297             4'b0010: SSD_CATHODES = 7'b0001111 ; // 2
298             4'b0011: SSD_CATHODES = 7'b0010010 ; // 3
299             4'b0100: SSD_CATHODES = 7'b0000110 ; // 4
300             4'b0101: SSD_CATHODES = 7'b0100100 ; // 5
301             4'b0110: SSD_CATHODES = 7'b0100000 ; // 6
302             4'b0111: SSD_CATHODES = 7'b0001111 ; // 7
303             4'b1000: SSD_CATHODES = 7'b0000000 ; // 8
304             4'b1001: SSD_CATHODES = 7'b0000100 ; // 9
305             4'b1010: SSD_CATHODES = 7'b0001000 ; // A
306             4'b1011: SSD_CATHODES = 7'b1100000 ; // B
307             4'b1100: SSD_CATHODES = 7'b0110001 ; // C
308             4'b1101: SSD_CATHODES = 7'b1000010 ; // D
309             4'b1110: SSD_CATHODES = 7'b0110000 ; // E
310             4'b1111: SSD_CATHODES = 7'b0111000 ; // F
311             default: SSD_CATHODES = 7'bXXXXXXX ; // default is not needed
  … as we covered all cases
312         endcase
313     end
314
315 endmodule
316
317
318
```