

Connor Aksama  
EE 371  
January 13, 2023  
Lab 1 Report

## Procedure

This lab is comprised of the design and testing of a parking lot occupancy simulator. The sequential system was designed in System Verilog, tested in ModelSim, and finally run on an FPGA. The sensors and system reset was simulated using hardware connections to GPIO ports on the given circuit board.

### Task 1

I approached this task by following the given development procedure outlined by the lab specification. First, I designed the FSM for the parking lot sensors, which took sensor readings as input, and output signals that indicated whether a car was entering or exiting the lot. Next, I designed a five-bit increment/decrement counter to keep track of the number of cars. Then, these modules were combined in a separate module, which handled GPIO connections, clock setup, and connection between modules. Each of these modules were individually tested by creating and examining waveforms in ModelSim.

Finally, I wired the breadboard circuit in the virtual lab environment with the necessary switches and LEDs, then uploaded the completed System Verilog code to the FPGA to perform final tests.

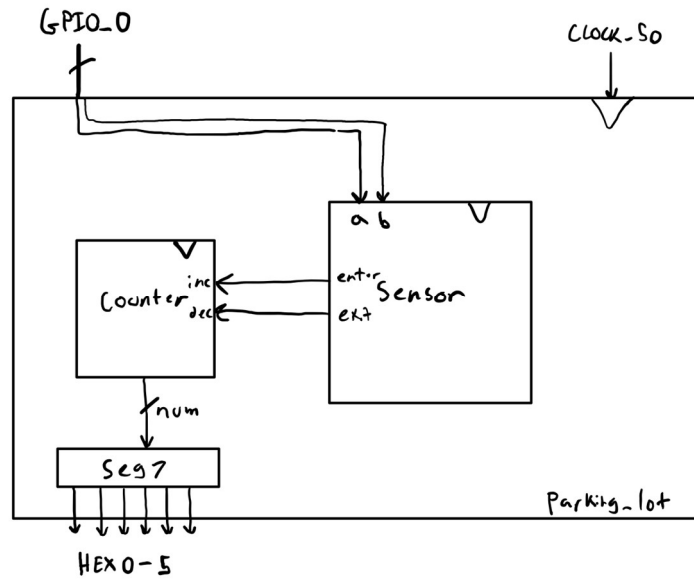


Figure 1. Parking Lot System Block Diagram

This block diagram illustrates the major components of the parking lot system and shows the important connections between modules.

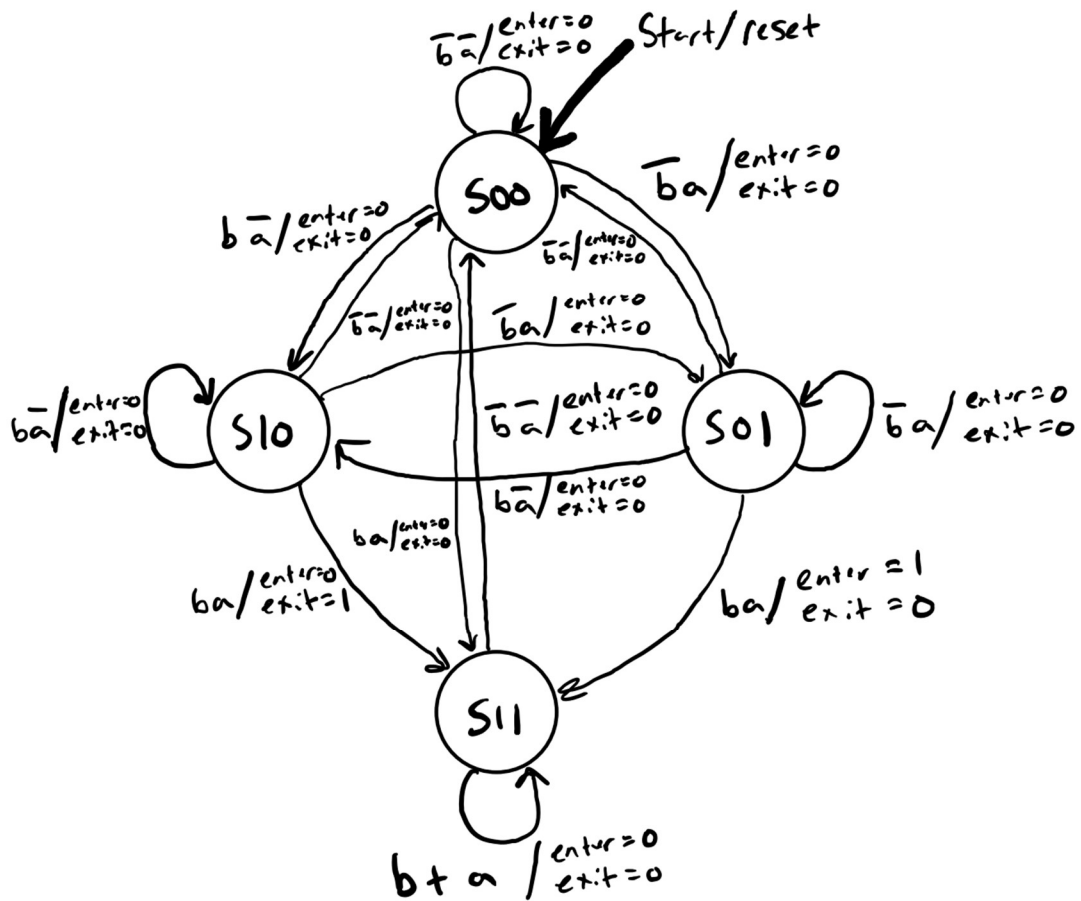


Figure 2. Sensor Module Finite State Machine

This finite state machine defines the behavior for the sensor module of the parking lot system. The signals “b” and “a” correspond to the output of each sensor. An enter or exit signal is only raised if both sensors start low, exactly one of sensors is high for at least one cycle, then both sensors are high. This prevents a singular car from artificially altering the count by moving in and out of one of the sensors.

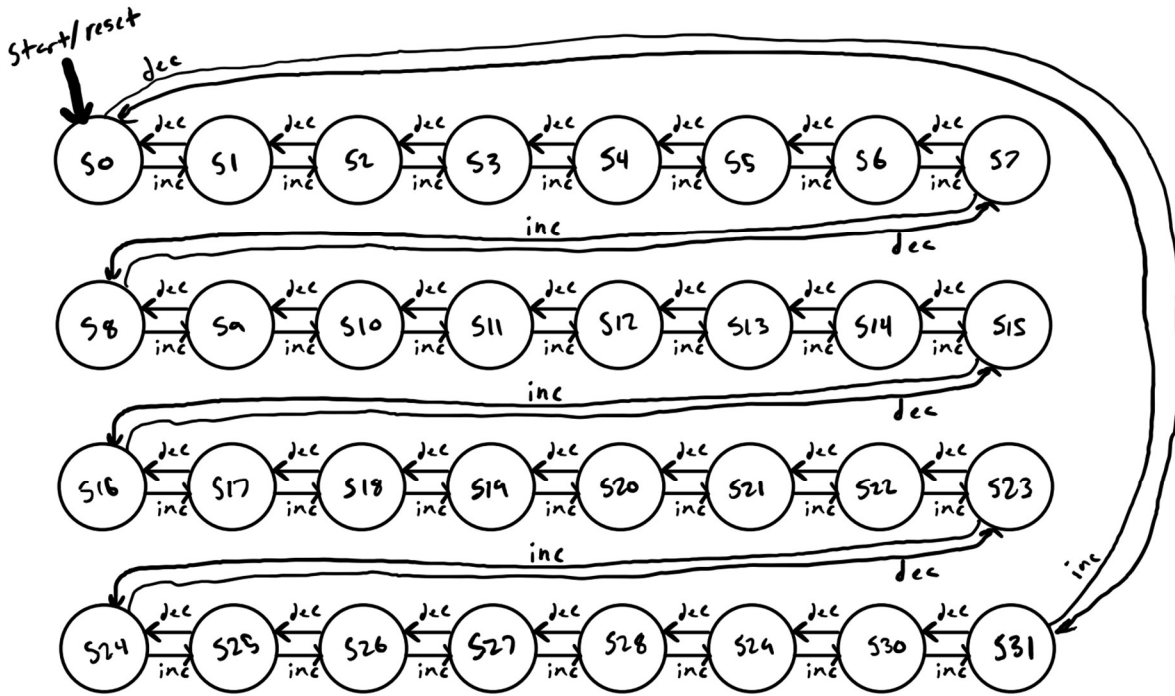


Figure 3. Five-Bit Counter Finite State Machine

The finite state machine for the five-bit counter module. On an “inc” signal, the counter output increases by one, and on a “dec” signal, the count decreases by one. At its maximum value of 31, an “inc” signal causes the output to wrap to 0. Similarly, at its minimum of 0, a “dec” signal causes a wrap to an output of 31. If neither signal is raised, the count remains constant.

## Results

### Task 1

The following are screenshots from ModelSim simulations for each module used in the parking lot system.



Figure 4. Sensor Module Waveform

This simulation runs through each possible transition between sensor output permutations (e.g. 00->00, 00->01, 00->10, etc.). The simulation demonstrates that the “enter” signal is only raised when transitioning from sensor output 01->11 and the “exit” signal is only raised when transitioning from 10->11.

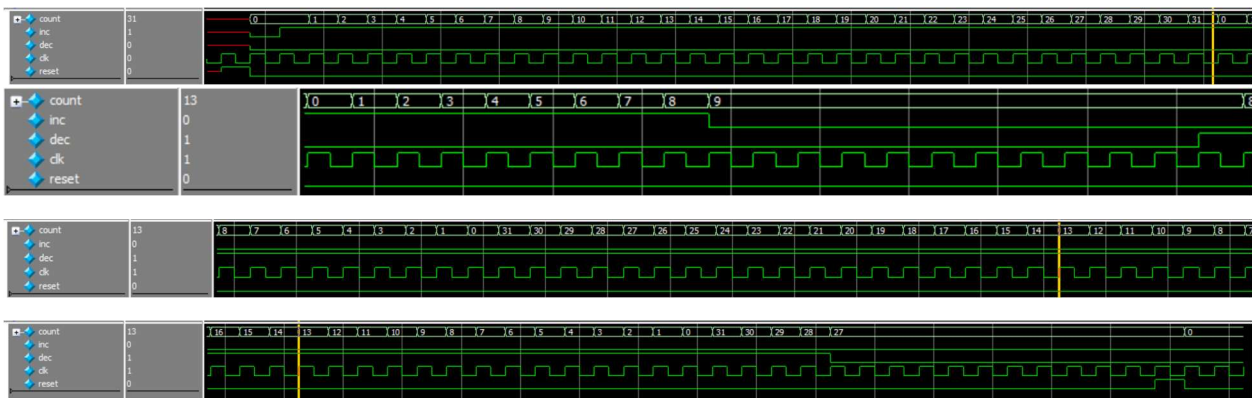


Figure 5. Five-Bit Counter Waveform

This simulation demonstrates that the counter can count from the minimum value 0 to the maximum 31 when “inc” is raised, wrap down to 0 from 31, and that an arbitrary value stays constant when no input is raised. Similarly, the simulation shows that the counter is able to count from the maximum value 31 down to the minimum 0, wrap up to 31 from 0, and stay constant with both inputs low. The simulation also shows the functionality of the reset signal.



## Appendix: SystemVerilog Code

(See next page)

```

1  /*****
2
3  sensor.sv
4
5  *****/
6
7  /*
8   * Connor Aksama
9   * 01/13/2023
10  * CSE 371
11  * Lab 1
12  */
13
14  /*
15   * FSM implementation to determine whether a car has entered/exited the lot given sensor
    inputs.
16
17   * Inputs:
18   *     a [1 bit] - The status of sensor a. a is 1 if sensor a is blocked. a is 0 if
    sensor a is not blocked.
19   *     b [1 bit] - The status of sensor b. b is 1 if sensor b is blocked. b is 0 if
    sensor b is not blocked.
20   *     clk [1 bit] - The clock signal to use for this module.
21
22   * Outputs:
23   *     enter [1 bit] - Signal indicating whether a car has entered the lot. enter is 1 if a
    car has entered the lot during the current clock cycle, 0 otherwise.
24   *     exit [1 bit] - Signal indicating whether a car has exited the lot. exit is 1 if
    a car has exited the lot during the current clock cycle, 0 otherwise.
25  */
26  module sensor(
27      output logic enter, exit
28      ,input logic a, b, clk
29      );
30
31      // FSM states
32      typedef enum logic [1:0] {s00, s01, s10, s11} state;
33
34      state ps, ns;
35
36      // Handle FSM transitions
37      always_comb begin
38
39          if (ps == s11) begin
40              // Transition out of s11 only if b nor a
41
42              if (b | a) ns = s11;
43              else ns = s00;
44
45          end else begin
46              // Transition to state determined by 'ba'
47
48              ns = state'({b, a});
49
50          end
51
52          enter = 1'b0;
53          exit = 1'b0;
54
55          // Raise enter if going from s01->s11
56          if (ps == s01 & ns == s11)
57              enter = 1'b1;
58
59          // Raise exit if going from s10->s11
60          if (ps == s10 & ns == s11)
61              exit = 1'b1;
62
63      end
64

```



```

65     // Update present state on posedge clk
66     always_ff @(posedge clk) begin
67
68         ps <= ns;
69
70     end
71
72 endmodule // sensor
73
74 /*
75  * Tests the functionality of the sensor module.
76  */
77 module sensor_testbench();
78
79     logic enter, exit;
80     logic a, b, clk;
81
82     sensor dut(.a, .b, .clk, .enter, .exit);
83
84     parameter CLOCK_PERIOD = 100;
85     initial begin
86         clk <= 0;
87         forever #(CLOCK_PERIOD / 2) clk <= ~clk;
88     end
89
90     initial begin
91         integer i, j;
92
93         // Test transition between every pair of states
94         for (i = 0; i < 4; i++) begin
95             for (j = 0; j < 4; j++) begin
96                 @(posedge clk) {b, a} <= i;
97                 @(posedge clk) {b, a} <= j;
98             end
99         end
100
101         @(posedge clk);
102         @(posedge clk);
103
104         $stop;
105     end
106
107 endmodule // sensor_testbench
108
109 /*****
110
111 fiveb_counter.sv
112
113 *****/
114
115 /*
116  * Connor Aksama
117  * 01/13/2023
118  * CSE 371
119  * Lab 1
120  */
121
122 /**
123  * A five-bit increment/decrement counter. Overflows to 5'b00000; underflows to 5'b11111.
124  * reset takes precedence; if inc and dec are simultaneously raised, count is unchanged.
125  * Inputs:
126  *   inc [1 bit] - Increments the count by 1 when raised. Does nothing if signal is low.
127  *   dec [1 bit] - Decrements the count by 1 when raised. Does nothing if signal is low.
128  *   clk [1 bit] - The clock to use for this module.
129  *   reset [1 bit] - Resets the counter to 0 when raised. Does nothing if signal is low.
130
131  * Outputs:
132  *   count [5 bit] - The count of this counter.
133  */

```

```

134 module fiveb_counter(
135     output logic [4:0] count
136     ,input logic inc, dec, clk, reset
137 );
138
139 // Handles increment/decrement/reset operations
140 always_ff @(posedge clk) begin
141
142     if (reset)
143         count <= 5'b0;
144     else if (inc & ~dec)
145         count <= count + 5'b1;
146     else if (dec & ~inc)
147         count <= count - 5'b1;
148
149 end
150
151 endmodule // fiveb_counter
152
153 /*
154  * Tests the functionality of the fiveb_counter module.
155  */
156 module fiveb_counter_testbench();
157
158     logic [4:0] count;
159     logic inc, dec, clk, reset;
160
161     fiveb_counter dut (.count, .inc, .dec, .clk, .reset);
162
163     parameter CLOCK_PERIOD = 100;
164     initial begin
165         clk <= 0;
166         forever #(CLOCK_PERIOD / 2) clk <= ~clk;
167     end
168
169     initial begin
170         integer i;
171
172         @(posedge clk) reset <= 1;
173         @(posedge clk) reset <= 0; inc <= 0; dec <= 0;
174
175         // Increment counter from 0 to max, then past max
176         @(posedge clk) inc <= 1;
177         for (i = 0; i < 40; i++) begin
178             @(posedge clk);
179         end
180
181         // Hold counter constant
182         @(posedge clk) inc <= 0;
183         for (i = 0; i < 10; i++) begin
184             @(posedge clk);
185         end
186
187         // Decrement counter below 0, then max to 0, etc.
188         @(posedge clk) dec <= 1;
189         for (i = 0; i < 45; i++) begin
190             @(posedge clk);
191         end
192
193         // Hold counter constant
194         @(posedge clk) dec <= 0;
195         for (i = 0; i < 10; i++) begin
196             @(posedge clk);
197         end
198
199         // reset to 0
200         @(posedge clk) reset <= 1;
201         @(posedge clk) reset <= 0;
202

```

```

203         @(posedge clk)
204         @(posedge clk)
205
206         $stop;
207     end
208
209 endmodule // fiveb_counter_testbench
210
211 /*****
212
213 parking_lot.sv
214
215 *****/
216
217 /*
218  * Connor Aksama
219  * 01/13/2023
220  * CSE 371
221  * Lab 1
222  */
223
224 /**
225  * Top-level module for Lab 1. Handles GPIO connections, HEX displays, and connections
226  * between submodules.
227
228  * Inputs:f
229  *   CLOCK_50 [1 bit] - The system clock source to use for this system.
230
231  * Outputs:
232  *   HEX0 [7 bit] - Data to show on the HEX0 display, formatted in standard 7 segment
233  *   display format.
234  *   HEX1 [7 bit] - Data to show on the HEX1 display, formatted in standard 7 segment
235  *   display format.
236  *   HEX2 [7 bit] - Data to show on the HEX2 display, formatted in standard 7 segment
237  *   display format.
238  *   HEX3 [7 bit] - Data to show on the HEX3 display, formatted in standard 7 segment
239  *   display format.
240  *   HEX4 [7 bit] - Data to show on the HEX4 display, formatted in standard 7 segment
241  *   display format.
242  *   HEX5 [7 bit] - Data to show on the HEX5 display, formatted in standard 7 segment
243  *   display format.
244
245  * Inouts:
246  *   GPIO_0 [34 bit] - GPIO ports on the target board. Ports 5-22 are inputs and
247  *   ports 26-27 are outputs.
248  */
249 module parking_lot #(
250     parameter capacity = 25
251 )
252 (
253     output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5
254     ,input logic CLOCK_50
255     ,inout logic [33:0] GPIO_0
256 );
257
258 // Connections between sensor output and counter input
259 logic enter, exit;
260
261 logic [6:0] HEX1_temp;
262
263 assign reset = GPIO_0[7];
264
265 // Connect SW1 and SW2 to the sensor a and b inputs of the sensor module
266 // Store outputs to local logic
267 sensor s (.a(GPIO_0[5]), .b(GPIO_0[6]), .enter, .exit, .clk(CLOCK_50));
268
269 // Connect the data from SW1 and SW2 to the left and right LEDs, respectively
270 assign GPIO_0[26] = GPIO_0[5]; // Left LED
271 assign GPIO_0[27] = GPIO_0[6]; // Right LED

```

```

264
265     logic [4:0] car_count;
266     logic reset;
267
268     // Store counter output in local logic bus
269     // Connect local enter/exit signals to inc/dec counter input, respectively
270     fiveb_counter counter (.count(car_count), .inc(enter), .dec(exit), .clk(CLOCK_50), .
    reset);
271
272     // Determine HEX0 and HEX1 displays using local count
273     double_seg7 count_display (.HEX0, .HEX1(HEX1_temp), .num({2'b00, car_count}));
274
275     // Handle HEX displays
276     always_comb begin
277
278         if (car_count == capacity) begin
279             // Lot is at capacity
280             HEX5 = ~7'b1110001; // F
281             HEX4 = ~7'b0111110; // U
282             HEX3 = ~7'b0111000; // L
283             HEX2 = ~7'b0111000; // L
284             HEX1 = HEX1_temp;
285         end else if (car_count == 0) begin
286             // Lot is empty
287             HEX5 = ~7'b0111001; // C
288             HEX4 = ~7'b0111000; // L
289             HEX3 = ~7'b1111001; // E
290             HEX2 = ~7'b1110111; // A
291             HEX1 = ~7'b0110001; // R
292         end else begin
293             HEX5 = ~7'b0000000;
294             HEX4 = ~7'b0000000;
295             HEX3 = ~7'b0000000;
296             HEX2 = ~7'b0000000;
297             HEX1 = HEX1_temp;
298         end
299     end
300 end
301
302 endmodule // parking_lot
303
304 /*
305  * Tests the functionality of the parking_lot module.
306  */
307 module parking_lot_testbench();
308
309     logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
310     logic CLOCK_50;
311     logic SW1, SW2, reset, clk;
312     wire [33:0] GPIO_0;
313
314     parking_lot dut (.HEX0, .HEX1, .HEX2, .HEX3, .HEX4, .HEX5, .CLOCK_50(clk), .GPIO_0);
315
316     assign GPIO_0[5] = SW1;
317     assign GPIO_0[6] = SW2;
318     assign GPIO_0[7] = reset;
319
320     parameter CLOCK_PERIOD = 100;
321     initial begin
322         clk <= 0;
323         forever #(CLOCK_PERIOD / 2) clk <= ~clk;
324     end
325
326     initial begin
327         integer i;
328
329         @(posedge clk) reset <= 1;
330         @(posedge clk) reset <= 0;
331

```

```

332         // Increase the lot to capacity
333         for (i = 0; i < 5; i++) begin
334
335             @(posedge clk) SW1 <= 0; SW2 <= 0;
336             @(posedge clk) SW1 <= 1;
337             @(posedge clk) SW2 <= 1;
338
339         end
340
341         // Decrease the lot to empty
342         for (i = 0; i < 5; i++) begin
343
344             @(posedge clk) SW1 <= 0; SW2 <= 0;
345             @(posedge clk) SW2 <= 1;
346             @(posedge clk) SW1 <= 1;
347
348         end
349
350         // Increase lot capacity
351         for (i = 0; i < 3; i++) begin
352
353             @(posedge clk) SW1 <= 0; SW2 <= 0;
354             @(posedge clk) SW1 <= 1;
355             @(posedge clk) SW2 <= 1;
356
357         end
358
359         // Reset simulation
360         @(posedge clk) reset <= 1;
361         @(posedge clk) reset <= 0;
362         @(posedge clk);
363
364         $stop;
365     end
366
367 endmodule // parking_lot_testbench
368
369 /*****
370
371 double_seg7.sv
372
373 *****/
374
375 /*
376  * Connor Aksama
377  * 01/13/2023
378  * CSE 371
379  * Lab 1
380  */
381
382 /**
383  * Defines data for 2-digit decimal HEX display given a 7-bit unsigned integer from
384  * [0-99].
385  * Output data for input numbers with 3 digits is undefined.
386  * Inputs:
387  *     num [7 bit] - An unsigned integer value [0-99] to display
388  * Outputs:
389  *     HEX0 [7 bit] - A HEX display for the ones digit of the input num, formatted in
390  *     standard seven segment display format
391  *     HEX1 [7 bit] - A HEX display for the tens digit of the input num, formatted in
392  *     standard seven segment display format
393  *     HEX1 is blank if num < 10.
394  */
395 module double_seg7(
396     output logic [6:0] HEX0, HEX1
397     ,input logic [6:0] num
398 );

```

```

398
399 // Drive HEX output signals given num
400 always_comb begin
401     // Light HEX0 using ones place of num
402     case (num % 10)
403         //      Light: 6543210
404         0: HEX0 = ~7'b01111111; // 0
405         1: HEX0 = ~7'b0000110;  // 1
406         2: HEX0 = ~7'b1011011;  // 2
407         3: HEX0 = ~7'b1001111;  // 3
408         4: HEX0 = ~7'b1100110;  // 4
409         5: HEX0 = ~7'b1101101;  // 5
410         6: HEX0 = ~7'b1111101;  // 6
411         7: HEX0 = ~7'b0000111;  // 7
412         8: HEX0 = ~7'b1111111;  // 8
413         9: HEX0 = ~7'b1101111;  // 9
414         default: HEX0 = 7'bX;
415     endcase
416
417     // Light HEX1 using tens place of num
418     case (num / 10)
419         //      Light: 6543210
420         0: HEX1 = ~7'b0000000;  // 0
421         1: HEX1 = ~7'b0000110;  // 1
422         2: HEX1 = ~7'b1011011;  // 2
423         3: HEX1 = ~7'b1001111;  // 3
424         4: HEX1 = ~7'b1100110;  // 4
425         5: HEX1 = ~7'b1101101;  // 5
426         6: HEX1 = ~7'b1111101;  // 6
427         7: HEX1 = ~7'b0000111;  // 7
428         8: HEX1 = ~7'b1111111;  // 8
429         9: HEX1 = ~7'b1101111;  // 9
430         default: HEX1 = 7'bX;
431     endcase
432
433 end
434
435 endmodule // double_seg7
436
437 /*
438  * Tests the functionality of the double_seg7 module.
439  */
440 module double_seg7_testbench();
441
442     logic [6:0] HEX0, HEX1;
443     logic [6:0] num;
444
445     double_seg7 dut (.HEX0, .HEX1, .num);
446
447     initial begin
448
449         integer i;
450
451         // Check HEX displays for integers 0-99
452         for (i = 0; i <= 99; i++) begin
453             #10 num = i;
454         end
455         #50;
456     end
457
458 endmodule // double_seg7_testbench
459

```