```
/**********************************

sensor.sv

**********************************/

/*
 * Connor Aksama
 * 01/13/2023
 * CSE 371
 * Lab 1
*/

/*
 * FSM implementation to determine whether a car has entered/exited the lot given sensor inputs.

 * Inputs:
 *      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.
 *      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.
 *      clk [1 bit] - The clock signal to use for this module.

 * Outputs:
 *  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.
 *      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.
*/
module sensor(
    output logic enter, exit
    ,input logic a, b, clk
    );

    // FSM states
    typedef enum logic [1:0] {s00, s01, s10, s11} state;

    state ps, ns;

    // Handle FSM transitions
    always_comb begin

        if (ps == s11) begin
            // Transition out of s11 only if b nor a

            if (b | a) ns = s11;
            else ns = s00;

        end else begin
            // Transition to state determined by 'ba'

            ns = state'({b, a});

        end

        enter = 1'b0;
        exit = 1'b0;

        // Raise enter if going from s01->s11
        if (ps == s01 & ns == s11)
            enter = 1'b1;

        // Raise exit if going from s10->s11
        if (ps == s10 & ns == s11)
            exit = 1'b1;

    end

```

```
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 5'11111.
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  */
```

```
module fiveb_counter(
    output logic [4:0] count
    ,input logic inc, dec, clk, reset
    );

    // Handles increment/decrement/reset operations
    always_ff @(posedge clk) begin

        if (reset)
            count <= 5'b0;
        else if (inc & ~dec)
            count <= count + 5'b1;
        else if (dec & ~inc)
            count <= count - 5'b1;

    end

endmodule  // fiveb_counter

/*
 * Tests the functionality of the fiveb_counter module.
*/
module fiveb_counter_testbench();

    logic [4:0] count;
    logic inc, dec, clk, reset;

    fiveb_counter dut (.count, .inc, .dec, .clk, .reset);

    parameter CLOCK_PERIOD = 100;
    initial begin
        clk <= 0;
        forever  #(CLOCK_PERIOD / 2) clk <= ~clk;
    end

    initial begin
        integer i;

        @(posedge clk) reset <= 1;
        @(posedge clk) reset <= 0; inc <= 0; dec <= 0;

        // Increment counter from 0 to max, then past max
        @(posedge clk) inc <= 1;
        for (i = 0; i < 40; i++) begin
            @(posedge clk);
        end

        // Hold counter constant
        @(posedge clk) inc <= 0;
        for (i = 0; i < 10; i++) begin
            @(posedge clk);
        end

        // Decrement counter below 0, then max to 0, etc.
        @(posedge clk) dec <= 1;
        for (i = 0; i < 45; i++) begin
            @(posedge clk);
        end

        // Hold counter constant
        @(posedge clk) dec <= 0;
        for (i = 0; i < 10; i++) begin
            @(posedge clk);
        end

        // reset to 0
        @(posedge clk) reset <= 1;
        @(posedge clk) reset <= 0;

```

```
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
        between submodules.
226 
227      * Inputs:f
228      *  CLOCK_50 [1 bit] - The system clock source to use for this system.
229 
230      * Outputs:
231      *  HEX0 [7 bit] - Data to show on the HEX0 display, formatted in standard 7 segment
        display format.
232      *  HEX1 [7 bit] - Data to show on the HEX1 display, formatted in standard 7 segment
        display format.
233      *  HEX2 [7 bit] - Data to show on the HEX2 display, formatted in standard 7 segment
        display format.
234      *  HEX3 [7 bit] - Data to show on the HEX3 display, formatted in standard 7 segment
        display format.
235      *  HEX4 [7 bit] - Data to show on the HEX4 display, formatted in standard 7 segment
        display format.
236      *  HEX5 [7 bit] - Data to show on the HEX5 display, formatted in standard 7 segment
        display format.
237 
238      * Inouts:
239      *      GPIO_0 [34 bit] - GPIO ports on the target board. Ports 5-22 are inputs and
        ports 26-27 are outputs.
240     */
241     module parking_lot #(
242         parameter capacity = 25
243         )
244         (
245         output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5
246         ,input logic CLOCK_50
247         ,inout logic [33:0] GPIO_0
248         );
249 
250         // Connections between sensor output and counter input
251         logic enter, exit;
252 
253         logic [6:0] HEX1_temp;
254 
255         assign reset = GPIO_0[7];
256 
257         // Connect SW1 and SW2 to the sensor a and b inputs of the sensor module
258         // Store outputs to local logic
259         sensor s (.a(GPIO_0[5]), .b(GPIO_0[6]), .enter, .exit, .clk(CLOCK_50));
260 
261         // Connect the data from SW1 and SW2 to the left and right LEDs, respectively
262         assign GPIO_0[26] = GPIO_0[5];  // Left LED
263         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
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
      [0-99].
384   * Output data for input numbers with 3 digits is undefined.
385
386   * Inputs:
387   *      num [7 bit] - An unsigned integer value [0-99] to display
388
389   * Outputs:
390   *  HEX0 [7 bit] - A HEX display for the ones digit of the input num, formatted in
      standard seven segment display format
391   *  HEX1 [7 bit] - A HEX display for the tens digit of the input num, formatted in
      standard seven segment display format
392   *                          HEX1 is blank if num < 10.
393  */
394  module double_seg7(
395      output logic [6:0] HEX0, HEX1
396      ,input logic [6:0] num
397      );
```

```
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'b0111111;  // 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
```