```
/*
   Ben Davis
   3/4/24
   EE 371
   Lab 6, Task 2

   This module instantiates all of my submodules for the
   Task 2 parking lot. It has V_GPIO inputs, and LED outputs
   on the simulator. It has HEX display outputs onto the FPGA,
   and KEY0 and Switch9 inputs.
*/

module DE1_SoC (CLOCK_50, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, KEY, SW, LEDR, V_GPIO);

   // define ports
   input  logic CLOCK_50;
   output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
   input  logic [3:0] KEY; // KEY0 to advance day
   input  logic [9:0] SW; // SW9 as reset
   output logic [9:0] LEDR;
   inout  logic [35:23] V_GPIO;

   // FPGA output
   assign V_GPIO[26] = V_GPIO[28];  // LED parking 1
   assign V_GPIO[27] = V_GPIO[29];  // LED parking 2
   assign V_GPIO[32] = V_GPIO[30];  // LED parking 3
   assign V_GPIO[34] = (V_GPIO[28] && V_GPIO[29] && V_GPIO[30]);  // LED full
   assign V_GPIO[31] = LEDR[3];  // Open entrance
   assign V_GPIO[33] = LEDR[4];  // Open exit

   // FPGA input
   assign LEDR[0] = V_GPIO[28];  // Presence parking 1
   assign LEDR[1] = V_GPIO[29];  // Presence parking 2
   assign LEDR[2] = V_GPIO[30];  // Presence parking 3
   assign LEDR[3] = V_GPIO[23];  // Presence entrance
   assign LEDR[4] = V_GPIO[24];  // Presence exit

/***
counting for parking spot 1
***/
   logic car1; //car1 input
   logic [4:0] spot1; //spot1 total cars for day
   singlePress P1 (.clk(CLOCK_50), .rst(SW[9]), .key(V_GPIO[28]), .onePress(car1));
   counter_25 space1 (.clk(CLOCK_50), .rst(SW[9]), .inc(car1), .dec(1'b0), .out(spot1));

/***
counting for parking spot 2
***/
   logic car2; //car2 input
   logic [4:0] spot2; //spot2 total cars for day
   singlePress P2 (.clk(CLOCK_50), .rst(SW[9]), .key(V_GPIO[29]), .onePress(car2));
   counter_25 space2 (.clk(CLOCK_50), .rst(SW[9]), .inc(car2), .dec(1'b0), .out(spot2));

/***
counting for parking spot 3
***/
   logic car3; //car3 input
   logic [4:0] spot3; //spot3 total cars for day
   singlePress P3 (.clk(CLOCK_50), .rst(SW[9]), .key(V_GPIO[30]), .onePress(car3));
   counter_25 space3 (.clk(CLOCK_50), .rst(SW[9]), .inc(car3), .dec(1'b0), .out(spot3));

//TIME (HOUR) INCREMENTER
   logic [3:0] hour; //current hour
   logic inc_hour; //incrementing hour
   singlePress SP (.clk(CLOCK_50), .rst(1'b0), .key(~KEY[0]), .onePress(inc_hour));
   hour_time tracker (.clk(CLOCK_50), .rst(SW[9]), .inc(inc_hour), .hour(hour));

//RUSH HOUR TRACKING
   logic [3:0] rush_start; //start of rush hour
   logic [3:0] rush_end; // end of rush hour
   rush_hour log (.clk(CLOCK_50), .rst(SW[9]), .car1(V_GPIO[28]), .car2(V_GPIO[29]),
      .car3(V_GPIO[30]), .hour(hour), .rush_start(rush_start), .rush_end(rush_end));

```

```
74    //RAM inputs
75       logic [15:0] total; // total cars for the day at that point in the day
76       assign total = spot1 + spot2 + spot3;
77       logic [15:0] whole_day; // output of ram of total
78       ram8x16 ram (.clock(CLOCK_50), .data(total), .rdaddress(rdaddress),
79                    .wraddress(hour), .wren(inc_hour), .q(whole_day));
80
81       logic [3:0] rdaddress;
82       always_ff @(posedge CLOCK_50) begin
83          if(hour != 4'b1000) begin
84             rdaddress <= hour;
85          end else begin
86             rdaddress <= scroll;
87       end end
88
89    //RAM Scroll
90       logic [31:0] clock;
91       clock_divider cdiv (.clock(CLOCK_50), .divided_clocks(clock));
92       logic scroll_start;
93       singlePress hour8 (.clk(CLOCK_50), .rst(SW[9]),
94                    .key(hour == 4'b1000), .onePress(scroll_start));
95       logic [3:0] scroll;
96       scroller scrl (.clk(clock[25]), .rst(scroll_start), .out(scroll));
97
98
99       /**********\
100      HEX DISPLAYS
101      \**********/
102
103   // HEXES 0-3
104      //logic for during the day, showing lot space
105      logic [1:0] cars;
106      assign cars = (V_GPIO[28] + V_GPIO[29] + V_GPIO[30]);
107      logic [6:0] hexzero;
108      logic [6:0] hexone;
109      logic [6:0] hextwo;
110      logic [6:0] hexthree;
111      lot_spots LS (.cars(cars), .hx0(hexzero), .hx1(hexone),
112                               .hx2(hextwo), .hx3(hexthree));
113
114      //logic for end of day parking information
115      logic [6:0] hexone_data;
116      logic [6:0] hextwo_addr;
117      logic [3:0] lot_data;
118      assign lot_data = whole_day[3:0];
119      hexadecimal hx1 (.in(lot_data), .out(hexone_data));
120      hexadecimal hx2 (.in(rdaddress), .out(hextwo_addr));
121
122      //clocked always to switch between data
123      logic [6:0] hexzero_fin;
124      logic [6:0] hexone_fin;
125      logic [6:0] hextwo_fin;
126      logic [6:0] hexthree_fin;
127      always_ff @(posedge CLOCK_50) begin
128         if(hour != 4'b1000) begin
129            hexzero_fin <= hexzero;
130            hexone_fin <= hexone;
131            hextwo_fin <= hextwo;
132            hexthree_fin <= hexthree;
133         end else begin
134            hexzero_fin <= 7'b1111111;
135            hexone_fin <= hexone_data;
136            hextwo_fin <= hextwo_addr;
137            if(rush_start == 4'b1111) begin
138               hexthree_fin <= 7'b0111111;
139            end else begin
140               hexthree_fin <= hexthree_rush;
141      end end end
142
143      //logic for if the day has ended and there is no rush hour
144      logic [6:0] hexthree_rush;
145      hexadecimal hx3 (.in(rush_start), .out(hexthree_rush));
146
```

```
147       assign HEX0 = hexzero_fin;
148       assign HEX1 = hexone_fin;
149       assign HEX2 = hextwo_fin;
150       assign HEX3 = hexthree_fin;
151    // END OF HEXES 0-3
152
153    // HEXES 4-5
154       logic [6:0] hexfour;
155       logic [6:0] hexfive;
156       logic [6:0] hexfive_hour;
157       logic [6:0] hexfour_final;
158       logic [6:0] hexfive_final;
159
160       //hex display converters
161       hexadecimal hx4_rush (.in(rush_end), .out(hexfour));
162       hexadecimal hx5_hour (.in(hour), .out(hexfive_hour));
163
164       //flip flops to allow the hex to display
165       //different datapaths during and after the day
166       always_ff @(posedge CLOCK_50) begin
167          if(hour != 4'b1000) begin
168             hexfour_final <= 7'b1111111;
169             hexfive_final <= hexfive_hour;
170          end else begin
171             if(rush_end == 4'b1111) begin
172                hexfour_final <= 7'b0111111;
173             end else begin
174                hexfour_final <= hexfour; end
175             hexfive_final <= 7'b1111111;
176       end end
177
178       //giving the hexes inputs
179       assign HEX4 = hexfour_final;
180       assign HEX5 = hexfive_final;
181    // END OF HEXES 4-5
182
183
184    endmodule  // DE1_SoC
185    //testbench
186    `timescale 1ps/1ps
187    module DE1_SoC_tb ();
188       //define ports
189       logic CLOCK_50;
190       logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
191       logic [3:0] KEY; // KEY0 to advance day
192       logic [9:0] SW; // SW9 as reset
193       logic [9:0] LEDR;
194       logic [35:23] V_GPIO;
195
196       DE1_SoC dut (.CLOCK_50, .HEX0, .HEX1, .HEX2, .HEX3, .HEX4, .HEX5, .KEY, .SW, .LEDR, .
       V_GPIO);
197
198       //clock setup
199       parameter clock_pd = 100;
200       initial begin
201          CLOCK_50 <= 0;
202          forever #(clock_pd /2) CLOCK_50 <= ~CLOCK_50;
203       end //of clock setup
204
205       //tests an instance where a car joins hour 1, another joins hour 3
206       //a third joins hour 5, and they all leave hour 7
207       initial begin
208          SW[9] <= 0; KEY[0] <= 0; V_GPIO[30:28] <= 3'b000; @(posedge CLOCK_50); //hour 0
209          SW[9] <= 0; KEY[0] <= 1; V_GPIO[30:28] <= 3'b000; @(posedge CLOCK_50);
210          SW[9] <= 0; KEY[0] <= 0; V_GPIO[30:28] <= 3'b001; @(posedge CLOCK_50); //hour 1
211          SW[9] <= 0; KEY[0] <= 0; V_GPIO[30:28] <= 3'b001; @(posedge CLOCK_50);
212          SW[9] <= 0; KEY[0] <= 0; V_GPIO[30:28] <= 3'b001; @(posedge CLOCK_50); //hour 2
213          SW[9] <= 0; KEY[0] <= 0; V_GPIO[30:28] <= 3'b001; @(posedge CLOCK_50);
214          SW[9] <= 0; KEY[0] <= 0; V_GPIO[30:28] <= 3'b011; @(posedge CLOCK_50); //hour 3
215          SW[9] <= 0; KEY[0] <= 0; V_GPIO[30:28] <= 3'b011; @(posedge CLOCK_50);
216          SW[9] <= 0; KEY[0] <= 0; V_GPIO[30:28] <= 3'b011; @(posedge CLOCK_50); //hour 4
217          SW[9] <= 0; KEY[0] <= 0; V_GPIO[30:28] <= 3'b011; @(posedge CLOCK_50);
218          SW[9] <= 0; KEY[0] <= 0; V_GPIO[30:28] <= 3'b111; @(posedge CLOCK_50); //hour 5
```

```
219         SW[9] <= 0; KEY[0] <= 0; V_GPIO[30:28] <= 3'b111; @(posedge CLOCK_50);
220         SW[9] <= 0; KEY[0] <= 0; V_GPIO[30:28] <= 3'b111; @(posedge CLOCK_50); //hour 6
221         SW[9] <= 0; KEY[0] <= 0; V_GPIO[30:28] <= 3'b111; @(posedge CLOCK_50);
222         SW[9] <= 0; KEY[0] <= 0; V_GPIO[30:28] <= 3'b000; @(posedge CLOCK_50); //hour 7
223         SW[9] <= 0; KEY[0] <= 0; V_GPIO[30:28] <= 3'b000; @(posedge CLOCK_50);
224         SW[9] <= 0; KEY[0] <= 0; V_GPIO[30:28] <= 3'b000; @(posedge CLOCK_50);//closing time
225         SW[9] <= 0; KEY[0] <= 0; V_GPIO[30:28] <= 3'b000; @(posedge CLOCK_50);
226         SW[9] <= 0; KEY[0] <= 0; V_GPIO[30:28] <= 3'b000; @(posedge CLOCK_50);//scroll time
227         $stop;
228      end //of test
229   endmodule
```