```
/* Name: Eugene Ngo
   Date: 3/7/2023
   Class: EE 371
   Lab 6: Parking Lot 3D Simulation
*/

// dataPathUnit outputs different values
// the hex displays based on many different control signals.
// These control signals are primarily determined by the controlUnit.
// This unit outputs data to the RAM and the HEX displays and
// then reads from the RAM into the Hex displays once the parking lot day is done.
`timescale 1 ps / 1 ps
 module dataPathUnit (startRush, stopRush, endGameHexOut, clk, in, time_in, incr_in, addr_in
, reset, rushEnded, slowClk, hexout0, hexout1, hexout2, hexout3, hexout4, hexout5);

   // In is the input from the carCount unit. 2 bits, to represent the number of cars
   // left.
   input logic [1:0] in;
   // Time in represents the current hour of the day, the output of the hourCount module.
   // Incr_in and addr_in are values used for the RAM module. Incr_in represents
   // the output of the carIncr unit and the addr_in represents the output of the
   // addrIncr unit. These are then inputted into the RAM module to first
   // write values in and then read from it once the parking lot is done.
   input logic [3:0] time_in, incr_in, addr_in;


   // main outputs of the control logic unit. These are used to
   // update HEX in specific conditions or set values that are to update the hex.
   // StartRush indicates to datapath the start of the rush and it stores that hour.
   // stopRush indicates the stop of the rush and it stores that hour.
   // endGameHexOut indicates that the parking lot day is done and that
   // the output is now to be rushHour info and RAM info.
   // clk and reset are standard.
   // rushEnded is the key to ensuring rushHour outputs only occur when
   // rushHours actually occured.
   // slowClk is the other clock used to load in RAM values to hex, slower
   // to ensure they are visible.
   input logic startRush, stopRush, endGameHexOut, clk, reset, rushEnded, slowClk;
   // Hexout0-5 output to the Hex displays in DE1_SoC
   output logic [6:0] hexout0, hexout1, hexout2, hexout3, hexout4, hexout5;

   // Different hex outputs that are stored to based on the control signals.
   // These the hexouts are eventually set if the appropriate control signals are triggered.

   logic[6:0] rushHourBegin, rushHourEnd, endingAddr, endingVal;


   // The clock that is selected at the end for units that require slower clocks.
   logic selectedClock;

   // Slow clock is divided_clocks[25] to act at 0.75 Hz which is approximately 0.75
updates per second
   // similar to the requirement of 1 update per second in the lab manual

   // Assigning hex display variables on necessary numbers.
   logic [6:0] hex0, hex1, hex2, hex3, hex4, hex5, hex6, hex7, hex8, hex9;
   logic [7:0] ramOutput;
   assign hex0 = 7'b1000000; // 0
   assign hex1 = 7'b1111001; // 1
   assign hex2 = 7'b0100100; // 2
   assign hex3 = 7'b0110000; // 3
   assign hex4 = 7'b0011001; // 4
   assign hex5 = 7'b0010010; // 5
   assign hex6 = 7'b0000010; // 6
   assign hex7 = 7'b1111000; // 7
   assign hex8 = 7'b0000000; // 8
   assign hex9 = 7'b0010000; // 9

   // Assigning hex display variables on necessary letters.
   logic [6:0] hexf, hexu, hexl, hexc, hexe, hexa, hexr, hexoff, hexdash;
   assign hexf = 7'b0001110; // F
   assign hexu = 7'b1000001; // U
   assign hexl = 7'b1000111; // L
```

```
72      assign hexoff = 7'b1111111; // off
73      assign hexdash = 7'b0111111; // -
74
75      // The selected clock. If it is the end game, use a slower clock. This
76      // selected clock is then used for the hex update for hex2 and 1 for the rAM
77      // readings.
78      assign selectedClock =  endGameHexOut ? slowClk : clk;
79
80
81      // Logic for hexout0. If it's the endgame, it's off otherwise
82      // displays the number of spots left. If the spots are full
83      // it displays the last l in full.
84      always_ff @ (posedge clk) begin
85         case(endGameHexOut)
86            1'b0:
87               case(in)
88                  2'b00: hexout0 = hex1;
89                  2'b01: hexout0 = hex1;
90                  2'b10: hexout0 = hex2;
91                  2'b11: hexout0 = hex3;
92                  default: hexout0 = 7'bX;
93               endcase
94
95            1'b1:
96               hexout0 <= hexoff;
97         endcase
98      end // always_comb
99
100     // Logic for hexout1. If it's the endgame, it displays the
101     // RAM value of the spot address being iterated over.
102     // Otherwise, it displays off, unless the spots are full
103     // in which case it displays the letter L
104     always_ff @ (posedge selectedClock) begin
105        case(endGameHexOut)
106           1'b0:
107              case(in)
108                 2'b00: hexout1 <= hex1;
109                 2'b01: hexout1 <= hexoff;
110                 2'b10: hexout1 <= hexoff;
111                 2'b11: hexout1 <= hexoff;
112                 default: hexout1 <= 7'bX;
113              endcase
114           1'b1:
115              hexout1 <= endingVal;
116        endcase
117     end // always_comb
118     // Logic for hexout2. If it's the endgame, it displays the
119     // RAM address of the address being iterated over.
120     // Otherwise, it displays off, unless the spots are
121     // full in which case it displays u.
122     always_ff @ (posedge selectedClock) begin
123        case(endGameHexOut)
124           1'b0:
125              case(in)
126                 2'b00: hexout2 <= hexu;
127                 2'b01: hexout2 <= hexoff;
128                 2'b10: hexout2 <= hexoff;
129                 2'b11: hexout2 <= hexoff;
130                 default: hexout2 <= 7'bX;
131              endcase
132           1'b1:
133              hexout2 <= endingAddr;
134        endcase
135     end
136     // Logic for hexout3. If it's the endgame
137     // it displays the rushHour beginning number.
138     // If there is no end to the rush hour, it displays hash.
139     // In non endgame. it displays the letter F in full.
140     always_ff @ (posedge selectedClock) begin
141        case(endGameHexOut)
142           1'b0:
143              case(in)
144                 2'b00: hexout3 <= hexf;
```

```
145                  2'b01: hexout3 <= hexoff;
146                  2'b10: hexout3 <= hexoff;
147                  2'b11: hexout3 <= hexoff;
148                  default: hexout3 <= 7'bX;
149               endcase
150            1'b1:
151               case (rushEnded)
152                  1'b1:
153                     hexout3 <= rushHourBegin;
154                  1'b0:
155                     hexout3 <= hexdash;
156               endcase
157         endcase
158      end
159
160      // Logic for hexout4. In endgame, it shows the hour
161      // that rush hour ended and displasy a dash if it never ended.
162      // Stays off otherwise.
163      always_ff @ (posedge clk) begin
164         case(endGameHexOut)
165            1'b0:
166               case(in)
167                  2'b00: hexout4 <= hexoff;
168                  2'b01: hexout4 <= hexoff;
169                  2'b10: hexout4 <= hexoff;
170                  2'b11: hexout4 <= hexoff;
171                  default: hexout4 <= 7'bX;
172               endcase
173            1'b1:
174               case (rushEnded)
175                  1'b1:
176                     hexout4 <= rushHourEnd;
177                  1'b0:
178                     hexout4 <= hexdash;
179               endcase
180         endcase
181      end
182      // Logic for setting the hour for rushHourbegin,
183      // If startRush is triggered, store the hour it
184      // was triggered in as a hex.
185      always_ff @ (posedge startRush) begin
186         case(time_in)
187            4'b0000: rushHourBegin <= hex0;
188            4'b0001: rushHourBegin <= hex1;
189            4'b0010: rushHourBegin <= hex2;
190            4'b0011: rushHourBegin <= hex3;
191            4'b0100: rushHourBegin <= hex4;
192            4'b0101: rushHourBegin <= hex5;
193            4'b0110: rushHourBegin <= hex6;
194            4'b0111: rushHourBegin <= hex7;
195            4'b1000: rushHourBegin <= hex8;
196            4'b1001: rushHourBegin <= hexoff;
197            4'b1010: rushHourBegin <= hexoff;
198            4'b1011: rushHourBegin <= hexoff;
199            4'b1100: rushHourBegin <= hexoff;
200            4'b1101: rushHourBegin <= hexoff;
201            4'b1110: rushHourBegin <= hexoff;
202            4'b1111: rushHourBegin <= hexoff;
203            default: rushHourBegin <= 7'bX;
204         endcase
205      end
206
207
208      // Logic for setting the hour for rushHourend,
209      // If stopRush is triggered, store the hour it
210      // was triggered in as a hex.
211      always_ff @ (posedge stopRush) begin
212         case(time_in)
213            4'b0000: rushHourEnd <= hex0;
214            4'b0001: rushHourEnd <= hex1;
215            4'b0010: rushHourEnd <= hex2;
216            4'b0011: rushHourEnd <= hex3;
217            4'b0100: rushHourEnd <= hex4;
```

```
            4'b0101: rushHourEnd <= hex5;
            4'b0110: rushHourEnd <= hex6;
            4'b0111: rushHourEnd <= hex7;
            4'b1000: rushHourEnd <= hex8;
            4'b1001: rushHourEnd <= hex9;
            4'b1010: rushHourEnd <= hexl;
            4'b1011: rushHourEnd <= hexl;
            4'b1100: rushHourEnd <= hexl;
            4'b1101: rushHourEnd <= hexl;
            4'b1110: rushHourEnd <= hexl;
            4'b1111: rushHourEnd <= hexl;
            default: rushHourEnd <= hexl;
        endcase
    end

    // Logic for setting Hexout5. This is
    // only set by the time in, if that exceeds
    // 7, sets it to off.
    always_ff @ (posedge clk) begin
        case(time_in)
            4'b0000: hexout5 <= hex0;
            4'b0001: hexout5 <= hex1;
            4'b0010: hexout5 <= hex2;
            4'b0011: hexout5 <= hex3;
            4'b0100: hexout5 <= hex4;
            4'b0101: hexout5 <= hex5;
            4'b0110: hexout5 <= hex6;
            4'b0111: hexout5 <= hex7;
            4'b1000: hexout5 <= hexoff;
            4'b1001: hexout5 <= hexoff;
            4'b1010: hexout5 <= hexoff;
            4'b1011: hexout5 <= hexoff;
            4'b1100: hexout5 <= hexoff;
            4'b1101: hexout5 <= hexoff;
            4'b1110: hexout5 <= hexoff;
            4'b1111: hexout5 <= hexoff;
            default: hexout5 <= 7'bx;
        endcase
    end


    // Translates the address the RAM is being
    // iterated into a HEX digit between 0 and 7.
    always_ff @ (posedge clk) begin
        case(addr_in)
            4'b0000: endingAddr <= hex0;
            4'b0001: endingAddr <= hex1;
            4'b0010: endingAddr <= hex2;
            4'b0011: endingAddr <= hex3;
            4'b0100: endingAddr <= hex4;
            4'b0101: endingAddr <= hex5;
            4'b0110: endingAddr <= hex6;
            4'b0111: endingAddr <= hex7;
            4'b1000: endingAddr <= hexoff;
            4'b1001: endingAddr <= hexoff;
            4'b1010: endingAddr <= hexoff;
            4'b1011: endingAddr <= hexoff;
            4'b1100: endingAddr <= hexoff;
            4'b1101: endingAddr <= hexoff;
            4'b1110: endingAddr <= hexoff;
            default: endingAddr <= 7'bx;
        endcase
    end

    // Translates the ramOutput value to
    // a hex digit to display on hex2.
    always_ff @ (posedge clk) begin
        case(ramOutput)
            8'b00000000: endingVal <= hex0;
            8'b00000001: endingVal <= hex1;
            8'b00000010: endingVal <= hex2;
            8'b00000011: endingVal <= hex3;
            8'b00000100: endingVal <= hex4;
```

```
291             8'b00000101: endingVal <= hex5;
292             8'b00000110: endingVal <= hex6;
293             8'b00000111: endingVal <= hex7;
294             8'b00001000: endingVal <= hex8;
295             8'b00001001: endingVal <= hexoff;
296             8'b00001010: endingVal <= hexoff;
297             8'b00001011: endingVal <= hexoff;
298             8'b00001100: endingVal <= hexoff;
299             8'b00001101: endingVal <= hexoff;
300             8'b00001110: endingVal <= hexoff;
301             default: endingVal <= 7'bX;
302          endcase
303       end
304
305       // Saves to ram when not in endgame, reads from ram in endgame.
306        RAM8x16  ram(.clock(clk), .data(incr_in), .rdaddress(addr_in),.wraddress(time_in),.wren
      (!endGameHexOut),.q(ramOutput));
307
308    endmodule // seg7
309
310    // dataPathUnit_testbench tests all expected, unexpected, and edgecase behaviors
311    // iterates through different inputs of time and inputs to
312    // see the impact on the parking lot.
313    module dataPathUnit_testbench();
314       logic CLOCK_50;
315       logic [1:0] in;
316       logic [3:0] time_in, incr_in, addr_in;
317       logic startRush, stopRush, endGameHexOut, clk, reset, rushEnded, slowClk;
318       logic [6:0] hexout0, hexout1, hexout2, hexout3, hexout4, hexout5;
319
320       dataPathUnit dut (startRush, stopRush, endGameHexOut, CLOCK_50, in, time_in, incr_in,
      addr_in, reset, rushEnded, CLOCK_50, hexout0, hexout1, hexout2, hexout3, hexout4, hexout5
      );
321
322       // Setting up the clock.
323       parameter CLOCK_PERIOD = 100;
324       initial begin
325          CLOCK_50 <= 0;
326          forever #(CLOCK_PERIOD/2) CLOCK_50 <= ~CLOCK_50; // toggle the clock forever
327       end // initial
328
329       initial begin
330          reset <= 1;                @(posedge CLOCK_50); // reset
331          reset <= 0;                @(posedge CLOCK_50); // inc past max limit
332          in <= 2'b10;               @(posedge CLOCK_50);
333          time_in <= 4'b0011;        @(posedge CLOCK_50);
334          startRush <= 1;            @(posedge CLOCK_50);
335          startRush <= 0;            @(posedge CLOCK_50);
336          time_in <= 4'b0100;        @(posedge CLOCK_50);
337          stopRush <= 1;             @(posedge CLOCK_50);
338          stopRush <= 0;             @(posedge CLOCK_50);
339          in <= 2'b11;               @(posedge CLOCK_50);
340          rushEnded <= 1;               @(posedge CLOCK_50);
341          time_in <= 4'b1000;        @(posedge CLOCK_50);
342          $stop;
343       end // initial
344    endmodule // seg7_testbench
```