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

// controlUnit takes in all enabling inputs to generate reliable
// control signals which manipulate the multiple paths data can take
// in the dataPath unit. As inputs, it takes in the current hour,
// status of the parking lots and outputs whether it is the start
// of a rush, the end, the end of the game and if rush has officially ended
// These states are used to determine what should be outputted to the
// HEX values based on the control outputs.
`timescale 1 ps / 1 ps
module controlUnit (occupied, noneOccupied, hour, reset, clk, startRush, stopRush,
endGameHexOut, rushEnded);

   // Occupied is the input of if all 3 spots are occupied
   // noneOccupied is the input of if none of the spots are occupied,
   // it is not the opposite of occupied, as this is
   // specifically for if NONE of the spots are occupied.
   // The last input is the current hour.
   // The occupied input is used to determine the startRush logic.
   // If all 3 spots are occupied, it is currently rush hour.
   // If none are occupied, rushHour is over.
   // If the current state reaches the true end of the parking
   // state, there was a proper rush hour, thus outputting
   // "rushEnded". Lastly, if the current hour is 8, it is
   // the end of the parking lot, regardless of rushHour occuring
   // or not.
   input logic occupied, noneOccupied, reset, clk;
   input logic [3:0] hour;
   output logic startRush, stopRush, endGameHexOut, rushEnded;

   // An enum to determine the current state of the lot.
   enum {s_regular_op, s_rush_start, s_rush_end, s_end} ps, ns;


   // Resets the state to the beginning if reset.
   always_ff @ (posedge clk) begin
      if (reset)
         ps <= s_regular_op;
      else
         ps <= ns;

   end

   // Iterates through different states based on
   // input logic signals while also defining outputs
   // based on current state. Mealy machine.
   always_comb begin
      case(ps)
         s_regular_op:
            if (startRush) ns = s_rush_start;
            else ns = s_regular_op;
         s_rush_start:
            if (stopRush) ns = s_rush_end;
            else ns = s_rush_start;
         s_rush_end:
            if (endGameHexOut) ns = s_end;
            else ns = s_rush_end;
         s_end:
            if (endGameHexOut) ns = s_end;
            else ns = s_end;
      endcase
   end

   // Assigns the output signals to different input conditions.
   assign startRush = occupied;
   assign stopRush = noneOccupied;
   assign rushEnded = (ps == s_end);

   // a comb unit to ensure the endGame is triggered
```

```
73      // for the hour being greater than (shouldn't happen)
74      // or equal to 8.
75      always_comb begin
76          case (hour)
77              4'b0000: endGameHexOut = 0;
78            4'b0001: endGameHexOut = 0;
79            4'b0010: endGameHexOut = 0;
80            4'b0011: endGameHexOut = 0;
81            4'b0100: endGameHexOut = 0;
82            4'b0101: endGameHexOut = 0;
83            4'b0110: endGameHexOut = 0;
84            4'b0111: endGameHexOut = 0;
85            4'b1000: endGameHexOut = 1;
86            4'b1001: endGameHexOut = 1;
87            4'b1010: endGameHexOut = 1;
88            4'b1011: endGameHexOut = 1;
89            4'b1100: endGameHexOut = 1;
90            4'b1101: endGameHexOut = 1;
91            4'b1110: endGameHexOut = 1;
92            4'b1111: endGameHexOut = 1;
93            default: endGameHexOut = 4'bx;
94          endcase
95      end
96  endmodule
97
98  // controlUnit_testbench tests all expected, unexpected, and edgecase behaviors
99  module controlUnit_testbench();
100     logic CLOCK_50;
101     logic occupied, noneOccupied, reset, clk;
102     logic [3:0] hour;
103     logic startRush, stopRush, endGameHexOut, rushEnded;
104
105
106     controlUnit dut (occupied, noneOccupied, hour, reset, CLOCK_50, startRush, stopRush,
    endGameHexOut, rushEnded);
107
108     // Setting up the clock.
109     parameter CLOCK_PERIOD = 100;
110     initial begin
111        CLOCK_50 <= 0;
112        forever #(CLOCK_PERIOD/2) CLOCK_50 <= ~CLOCK_50; // toggle the clock forever
113     end // initial
114
115     initial begin
116        reset <= 1;                @(posedge CLOCK_50); // reset
117        reset <= 0;                @(posedge CLOCK_50); // inc past max limit
118        hour <= 4'b0000;           @(posedge CLOCK_50);
119        occupied <=1;              @(posedge CLOCK_50);
120        occupied <=0;              @(posedge CLOCK_50);
121        occupied <=0;              @(posedge CLOCK_50);
122        noneOccupied <=1;                @(posedge CLOCK_50);
123        noneOccupied <=0;                @(posedge CLOCK_50);
124        noneOccupied <=0;                @(posedge CLOCK_50);
125        hour <= 4'b1000;           @(posedge CLOCK_50);
126        $stop;
127     end // initial
128 endmodule // seg7_testbench
```