Eugene Ngo: 1965514

3/13/2023

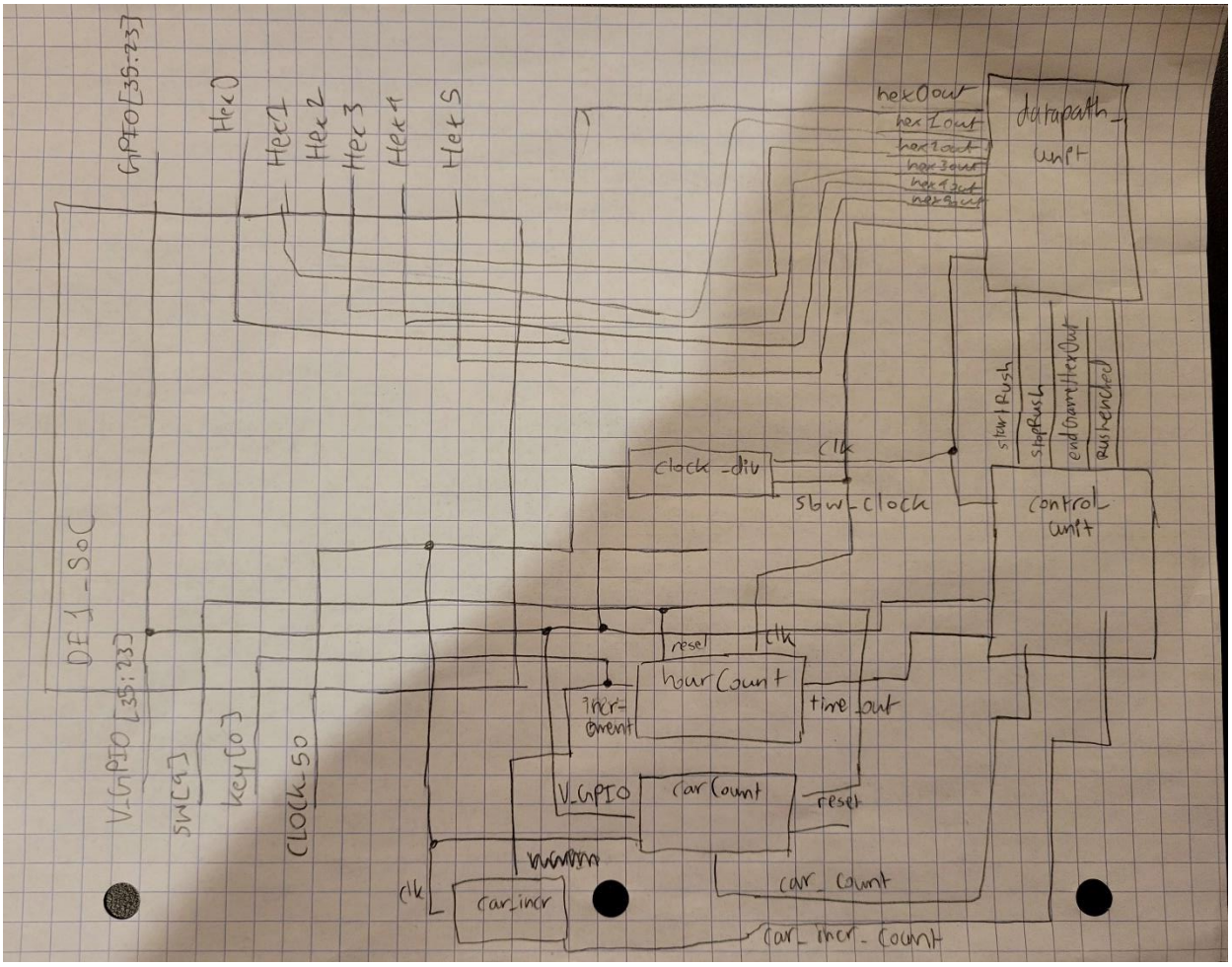
EE 371 – Lab 6 Report

Procedure

This lab was a continuation or development of lab 1 and comprises of two tasks. The first task was to take the same implementation from lab 1 and demonstrate its functionality onto the new breadboard remote lab surface. This entailed changing from “GPIO_0” references to “V_GPIO” references in my top-level module. Then when demonstrating the task on the new breadboard remote lab surface, I had to quickly wire the LEDs and switches to show the full functionality of the task.

The second task was to built upon Lab 1 and the first task from this lab and implement a 3D Parking Lot simulation. The simulation was meant to represent an 8 hour work day and the rush hour that occurs within it. After the work day ends, or 8 hours pass, then the FPGA will display how many cars entered the parking lot at each hour of the day and will show when a rush hour sequence begins—the 3-car parking lot is full—and when it ends—the lot is emptied again. If no rush hour occurred, then a ‘-’ will instead be displayed on both output HEXs. This was achieved by using a dual-port ram that I generated from the IP catalog.

Block Diagram



Task 1

Reading the updated GPIO guide, I changed the task 1 to use the new V_GPIO infrastructure, changing GPIO 5 and 7 to 23 and 24 and then changing GPIO 26 and 27 to 32 and 35. Lastly, GPIO 30 was used for reset instead of GPIO 9. Then, using the newly mapped GPIO, I just updated the logic for the incrementation the same way, if the first sensor was triggered and then the second, and then the first is released after the second, creating an entrance output. Reversing that for exiting, the same parts from lab 1 to increment and decrement the signals were used.

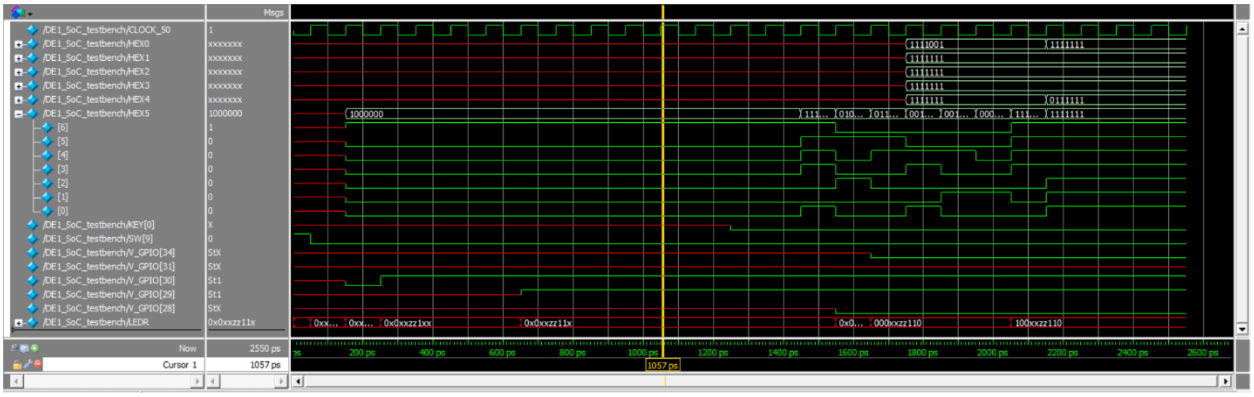
Task 2

Task 2 required using an ASMD with a datapath and control module to create a parking lot simulation with rush hour logic. Initially, the first things that were developed was mapping the

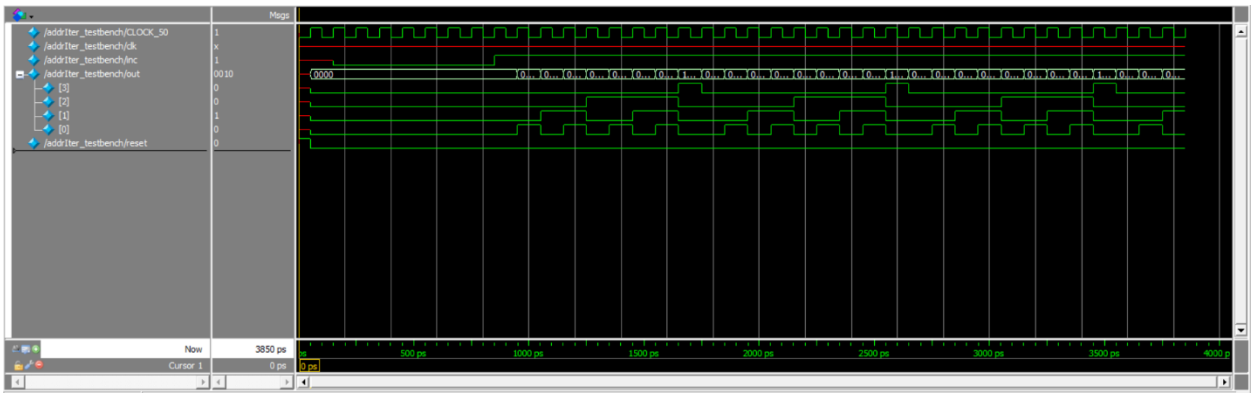
V_GPIO properly to ensure the behavior of the cars on entrance and exit was as expected. Once the behavior was proper, then I mapped out the logic for incrementing the number of hours, the number of cars entering and the number of cars present, using the V_GPIO and FSM logic. Then, using pipelining the number logics to a new unit which would become the control path. Within the control path, I created control signals as outputs that would be set to true based on the GPIO and number conditions. If the hour was 8, the game was in EndGameHex condition. Then, taking in the logic from control path, I created the datapath which was designed to update the Hex values based on the control signals. If the game was in endgame, hex0 to 5 took on values based on the endgame requirements. If the game was in the endgame, RAM was read from, otherwise it was written to. Pipelining all these units into each other, results in the full integration of the parking lot simulation with rush hour logic.

Results

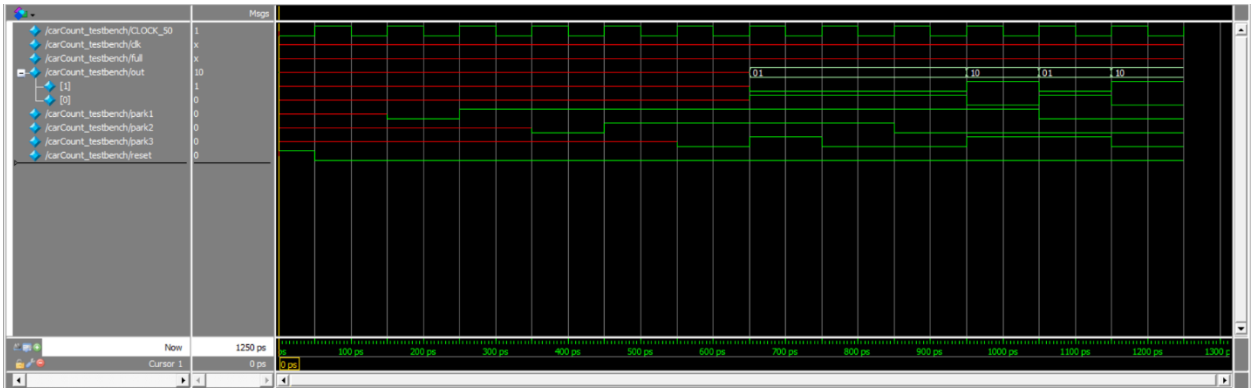
Task 2 Simulations



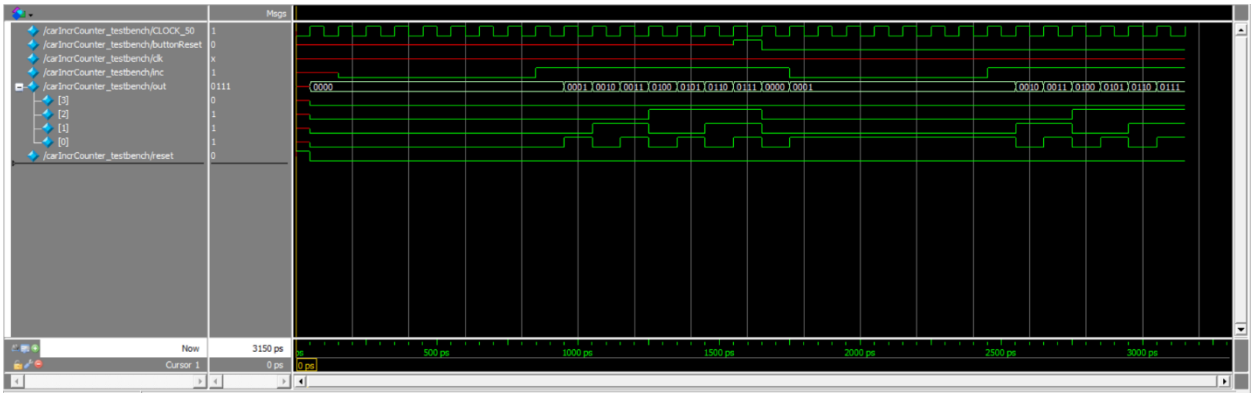
This is the simulation for the DE1_SoC, main module for lab 6, the parking lot simulation. This shows the changing data at Hex0-5, based on the changing hours and parking lot conditions. HEX5 outputs 1111111 at the end due to being switched off after hours are 8 or above.



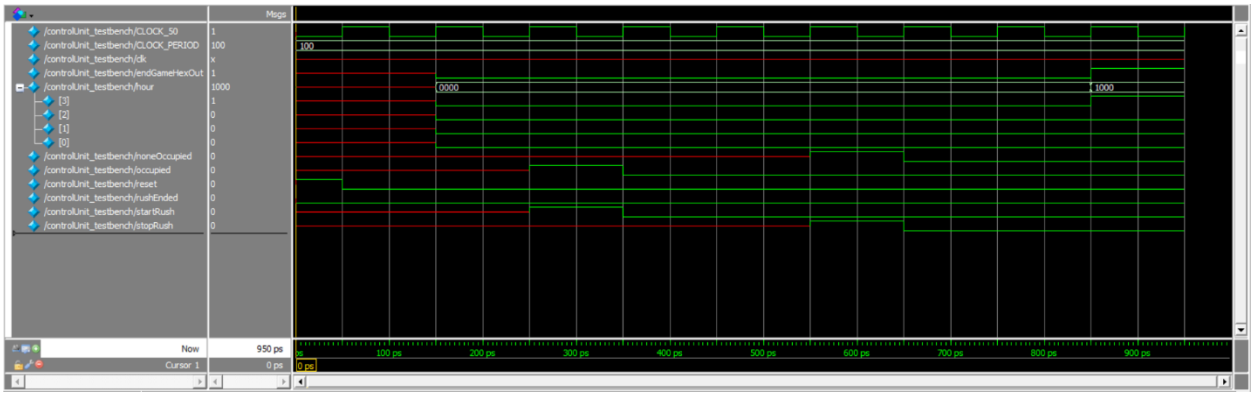
This unit is the address iterator. This is used by the datapath module to read and write from the RAM in different conditions. As you can see with the out variable, it iterates from 0 to 7 and then cycles back to 0 again.



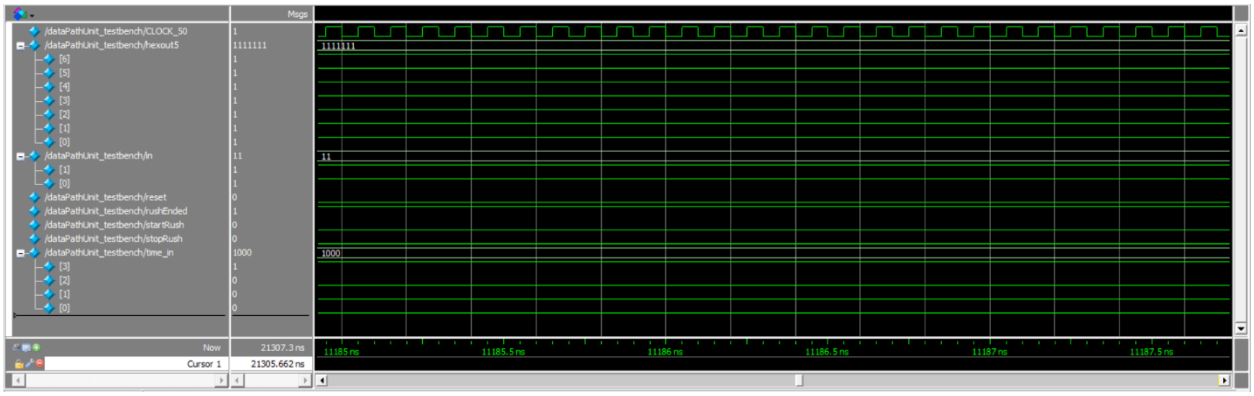
This unit is the car count. This is a module that keeps track of the number of spaces left in the parking lot based on the number of spaces taken first. As you can see the output changes based on if parking 1 or 2 or 3 is taken.



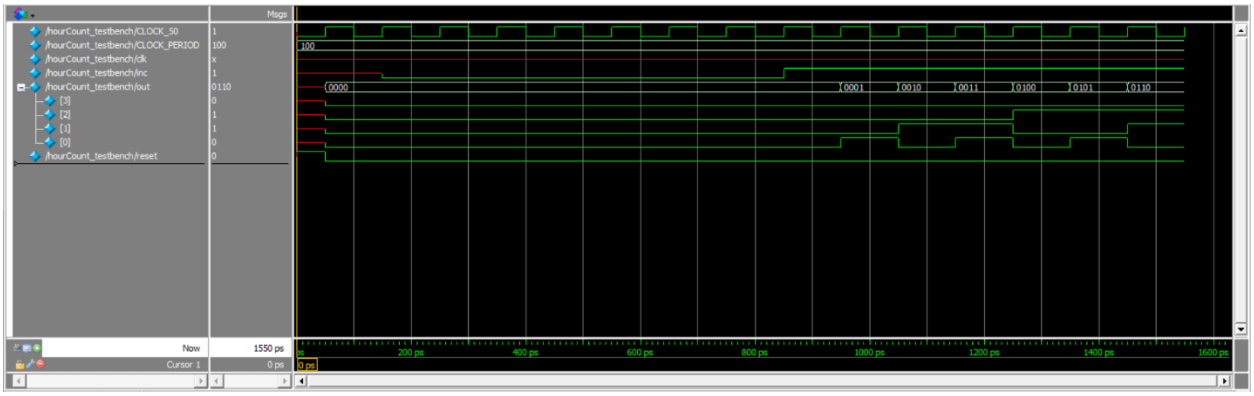
This unit is the car increment counter. This counts up every time there is a car that enters the parking lot and never decrements. It is reset every hour, to 0, so that the RAM can save the number of cars that entered for a specific hour.



The control unit is listed above. This updates various control signals such as startRush,stopRush, rushEnded, endGameHexOut, based on the inputs from VGPIO and number logic. As you can see the signals update based on various GPIO changes.



The datapath unit is listed above. This updates the Hex outputs based on the control signals. As you can see the hex output for HEX5 changes to be off based on the endGameHex signal while also enabling the RAM to be saved to.



This simulation I for the hour count. The hour is updated based on the input Key[0] or the inc signal. This shows that the output is updating every clock cycle based on the inc signal until t reaches the value 8 which it stays at, until reset.

Overview

Overall, this was an interesting lab using the new GPIO and breadboard system. The interactions in the 3D simulation were interesting to program. I learned more about utilizing ASMD logic in combination with a RAM module to then output to HEX0 to HEX5 with different values. It was interesting and I learned a lot about combining just about every aspect of what I learned in 371 to create this 3D parking lot simulator. The spec was long but necessarily so and provided all of the needed supporting documents. It was thorough in teaching us how to properly use the 3D parking simulator.

Overall, the systems work as expected.

Appendix

See following code

```
1   /* Name: Eugene Ngo
2      Date: 1/13/2023
3      Class: EE 371
4      Lab 6
5      Taken from Lab 1 and adapted for lab 6 task 1 */
6
7   // DE1_SoC is the top-level module that defines the I/Os for the DE-1 SoC board.
8   // DE1_SoC takes three switches from the GPIO as inputs, and outputs to 2 LEDs on the
    breadboard through GPIO and 6 7-bit
9   // hex displays (HEX0-HEX5). It displays "full" or "clear" messages when the parking lot is
    either full or clear, and it
10  // displays the decimal value of the number of cars in the lot accordingly.
11
12  module DE1_SoC #(parameter MAX=25) (HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, V_GPIO, CLOCK_50);
13      output logic [6:0]  HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
14      inout logic [35:23] V_GPIO;
15      input logic CLOCK_50;
16
17     // Assigning and clk to CLOCK_50
18     logic clk;
19     assign clk = CLOCK_50;
20
21     logic enter, exit, full, clear;
22     logic [4:0] counter_out;
23
24     // Outputting a and b to breadboard
25     assign V_GPIO[32] = V_GPIO[23];
26     assign V_GPIO[35] = V_GPIO[24];
27
28     // carSensor parkCheck takes two switches from the breadboard as the input of the two
    parking sensors,
29     // and outputs to enter and exit when an entering or exiting vehicle is detected.
30     carSensor parkCheck (.a(V_GPIO[24]), .b(V_GPIO[23]), .enter, .exit, .clk, .reset(V_GPIO[
    30]));
31
32     // carCount counter takes enter and exit from sensors, and outputs the car count to
    counter_out.
33     // it als outputs full and clear status to full and clear.
34     carCount #(MAX) counter (.inc(enter), .dec(exit), .out(counter_out), .full, .clear, .clk,
     .reset(V_GPIO[30]));
35
36     // seg7 display takes counter_out, full, and clear from ct, and outputs decimal values
    or status messages to hex displays.
37     seg7 display (.in(counter_out), .full, .clear, .hexout0(HEX0), .hexout1(HEX1), .hexout2(
    HEX2), .hexout3(HEX3), .hexout4(HEX4), .hexout5(HEX5));
38
39  endmodule // DE1_SoC
40
41  // DE1_SoC_testbench tests all expected, unexpected, and edgecase behaviors
42  module DE1_SoC_testbench();
43     logic [6:0]  HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
44     wire [35:23] V_GPIO;
45     logic CLOCK_50;
46
47     DE1_SoC #(5) dut (.HEX0, .HEX1, .HEX2, .HEX3, .HEX4, .HEX5, .V_GPIO, .CLOCK_50);
48
49     // Setting up a simulated clock.
50     parameter CLOCK_PERIOD = 100;
51     initial begin
52        CLOCK_50 <= 0;
53        forever #(CLOCK_PERIOD/2) CLOCK_50 <= ~CLOCK_50; // Forever toggle the clock
54     end
55
56     // Assiging logic to wire
57     logic reset, a, b;
58     assign V_GPIO[30] = reset;
59     assign V_GPIO[24] = a;
60     assign V_GPIO[23] = b;
61
62     // Testing the module
63     initial begin
64        // reset
65        reset <= 1;                    repeat(3) @(posedge CLOCK_50);
```

```
66
67          //enters until full
68          reset <= 0;       a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50);
69                            a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
70                            a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
71                            a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
72                            a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50); // 1st car enters
73                            a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
74                            a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
75                            a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
76                            a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50); // 2nd car enters
77                            a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
78                            a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
79                            a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
80                            a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50); // 3rd car enters
81                            a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
82                            a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
83                            a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
84                            a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50); // 4th car enters
85                            a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
86                            a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
87                            a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
88                            a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50); // 5th car enters, full
89
90      // exits until clear
91                            a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50);
92                            a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
93                            a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
94                            a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
95                            a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50); // 1st car exits
96                            a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
97                            a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
98                            a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
99                            a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50); // 2nd car exits
100                           a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
101                           a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
102                           a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
103                           a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50); // 3rd car exits
104                           a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
105                           a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
106                           a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
107                           a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50); // 4th car exits
108                           a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
109                           a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
110                           a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
111                           a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50); // 5th car exits, clear
112
113         // direction changes while entering
114                           a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50);
115                           a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
116                           a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
117                           a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
118                           a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
119                           a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
120                           a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
121                           a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
122                           a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50);
123         // direction changes while exiting
124                           a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50);
125                           a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
126                           a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
127                           a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
128                           a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
129                           a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
130                           a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
131                           a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
132                           a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50);
133
134         $stop;
135     end
136 endmodule // DE1_SoC_testbench
```

```
1   /* Name: Eugene Ngo
2      Date: 1/13/2023
3      Class: EE 371
4      Lab 6
5      Taken from Lab 1 and adapted for lab 6 task 1 */
6
7   // seg7 outputs correct decimal value to hex displays based on the output given by the
    counter.
8   // It also displays "FULL" and "CLEAR" according to the indicator outputs given by the
    counter.
9    module seg7 (in, full, clear, hexout0, hexout1, hexout2, hexout3, hexout4, hexout5);
10
11     input logic full, clear;
12     input logic [4:0] in;
13     output logic [6:0] hexout0, hexout1, hexout2, hexout3, hexout4, hexout5;
14
15     // Assigning hex display variables on necessary numbers.
16     logic [6:0] hex0, hex1, hex2, hex3, hex4, hex5, hex6, hex7, hex8, hex9;
17     assign hex0 = 7'b1000000; // 0
18     assign hex1 = 7'b1111001; // 1
19     assign hex2 = 7'b0100100; // 2
20     assign hex3 = 7'b0110000; // 3
21     assign hex4 = 7'b0011001; // 4
22     assign hex5 = 7'b0010010; // 5
23     assign hex6 = 7'b0000010; // 6
24     assign hex7 = 7'b1111000; // 7
25     assign hex8 = 7'b0000000; // 8
26     assign hex9 = 7'b0010000; // 9
27
28     // Assigning hex display variables on necessary letters.
29     logic [6:0] hexf, hexu, hexl, hexc, hexe, hexa, hexr, hexoff;
30     assign hexf = 7'b0001110; // F
31     assign hexu = 7'b1000001; // U
32     assign hexl = 7'b1000111; // L
33     assign hexc = 7'b1000110; // C
34     assign hexe = 7'b0000110; // E
35     assign hexa = 7'b0001000; // A
36     assign hexr = 7'b0101111; // R
37     assign hexoff = 7'b1111111; // off
38
39     // Logic for hexout0: 26 different cases for 26 numbers. (0-25)
40     always_comb begin
41        case(in)
42           5'b00000: hexout0 = hex0;
43           5'b00001: hexout0 = hex1;
44           5'b00010: hexout0 = hex2;
45           5'b00011: hexout0 = hex3;
46           5'b00100: hexout0 = hex4;
47           5'b00101: hexout0 = hex5;
48           5'b00110: hexout0 = hex6;
49           5'b00111: hexout0 = hex7;
50           5'b01000: hexout0 = hex8;
51           5'b01001: hexout0 = hex9;
52           5'b01010: hexout0 = hex0;
53           5'b01011: hexout0 = hex1;
54           5'b01100: hexout0 = hex2;
55           5'b01101: hexout0 = hex3;
56           5'b01110: hexout0 = hex4;
57           5'b01111: hexout0 = hex5;
58           5'b10000: hexout0 = hex6;
59           5'b10001: hexout0 = hex7;
60           5'b10010: hexout0 = hex8;
61           5'b10011: hexout0 = hex9;
62           5'b10100: hexout0 = hex0;
63           5'b10101: hexout0 = hex1;
64           5'b10110: hexout0 = hex2;
65           5'b10111: hexout0 = hex3;
66           5'b11000: hexout0 = hex4;
67           5'b11001: hexout0 = hex5;
68           default: hexout0 = 7'bx;
69        endcase
70     end // always_comb
71
```

```
72      // Logic for hexout1: 26 different cases for 26 numbers. (0-25)
73      always_comb begin
74         case(in)
75            5'b00000: hexout1 = hexr;
76            5'b00001: hexout1 = hexoff;
77            5'b00010: hexout1 = hexoff;
78            5'b00011: hexout1 = hexoff;
79            5'b00100: hexout1 = hexoff;
80            5'b00101: hexout1 = hexoff;
81            5'b00110: hexout1 = hexoff;
82            5'b00111: hexout1 = hexoff;
83            5'b01000: hexout1 = hexoff;
84            5'b01001: hexout1 = hexoff;
85            5'b01010: hexout1 = hex1;
86            5'b01011: hexout1 = hex1;
87            5'b01100: hexout1 = hex1;
88            5'b01101: hexout1 = hex1;
89            5'b01110: hexout1 = hex1;
90            5'b01111: hexout1 = hex1;
91            5'b10000: hexout1 = hex1;
92            5'b10001: hexout1 = hex1;
93            5'b10010: hexout1 = hex1;
94            5'b10011: hexout1 = hex1;
95            5'b10100: hexout1 = hex2;
96            5'b10101: hexout1 = hex2;
97            5'b10110: hexout1 = hex2;
98            5'b10111: hexout1 = hex2;
99            5'b11000: hexout1 = hex2;
100           5'b11001: hexout1 = hex2;
101           default: hexout1 = 7'bx;
102        endcase
103     end // always_comb
104
105     // Logic for hexout5 - hexout2: display letters when full or clear, turn off otherwise.
106     always_comb begin
107        if (full) begin
108           hexout5 = hexf;
109           hexout4 = hexu;
110           hexout3 = hexl;
111           hexout2 = hexl;
112        end
113        else if (clear) begin
114           hexout5 = hexc;
115           hexout4 = hexl;
116           hexout3 = hexe;
117           hexout2 = hexa;
118        end
119        else begin
120           hexout5 = hexoff;
121           hexout4 = hexoff;
122           hexout3 = hexoff;
123           hexout2 = hexoff;
124        end
125     end // always_comb
126
127   endmodule // seg7
128
129   // seg7_testbench tests all expected, unexpected, and edgecase behaviors
130   module seg7_testbench();
131     logic full, clear;
132     logic [4:0] in;
133     logic [6:0] hexout0, hexout1, hexout2, hexout3, hexout4, hexout5;
134
135     seg7 dut (.in, .full, .clear, .hexout0, .hexout1, .hexout2, .hexout3, .hexout4, .hexout5);
136
137     initial begin
138        in = '0; clear = 1; full = 0; #10; // testing clear output
139        in = 5'b11001; clear = 0; full = 1; #10; // testing full output
140        in = 5'b10011; clear = 0; full = 0; #10; // testing regular output
141        $stop;
142     end // initial
143  endmodule // seg7_testbench
```

```
1   /* Name: Eugene Ngo
2      Date: 1/13/2023
3      Class: EE 371
4      Lab 6
5      Taken from Lab 1 and adapted for lab 6 task 1 */
6
7   // carCount takes two inputs (inc, dec). It adds 5'b00001 to out when inc is true, and
    subtracts 5'b00001
8   // from out when dec is true. Out has a minimum value of 5'b00000 and a maximum value
    determined by the
9   // parameter (25 by default).
10  module carCount #(parameter MAX=25) (inc, dec, out, full, clear, clk, reset);
11
12     input logic inc, dec, clk, reset;
13     output logic [4:0] out;
14     output logic full, clear;
15
16     // Sequential logic for counting up and counting down depending on the input.
17     always_ff @(posedge clk) begin
18        if (reset) begin
19           out <= 5'b00000;
20           full <= 1'b0;
21           clear <= 1'b0;
22        end
23        else if (inc & out < MAX) begin //increment when not at max
24           out <= out + 5'b00001;
25           clear <= 1'b0;
26        end
27        else if (dec & out > 5'b00000) begin // decrement when not at min
28           out <= out - 5'b00001;
29           full <= 1'b0;
30        end
31        else if (out == MAX) begin // hold value at max, output full
32           out <= MAX;
33           full <= 1'b1;
34        end
35        else if (out == 5'b00000) begin // hold value at min, output clear
36           out <= 5'b00000;
37           clear <= 1'b1;
38        end
39        else
40           out <= out; // hold value otherwise
41     end // always_ff
42
43  endmodule
44
45  // carCount_testbench tests all expected, unexpected, and edgecase behaviors
46  module carCount_testbench();
47     logic inc, dec, full, clear, reset;
48     logic [4:0] out;
49     logic CLOCK_50;
50
51     carCount #(5) dut (.inc, .dec, .out, .full, .clear, .clk(CLOCK_50), .reset);
52
53     // Setting up the clock.
54     parameter CLOCK_PERIOD = 100;
55     initial begin
56        CLOCK_50 <= 0;
57        forever #(CLOCK_PERIOD/2) CLOCK_50 <= ~CLOCK_50; // toggle the clock forever
58     end // initial
59
60     initial begin
61        reset <= 1;                          repeat(3) @(posedge CLOCK_50); // reset
62        reset <= 0; inc <= 1;                repeat (7) @(posedge CLOCK_50); // inc past max limit
63                    inc <= 0; dec <= 1; repeat(7) @(posedge CLOCK_50); // dec past min limit
64        $stop;
65     end
66  endmodule // counter_testbench
```

```
1   /* Name: Eugene Ngo
2      Date: 1/13/2023
3      Class: EE 371
4      Lab 6
5      Taken from Lab 1 and adapted for lab 6 task 1 */
6
7   // carSensor takes inputs from two sensors, a and b, and output "1" to either enter or exit
    for 1 clock cycle
8   // whenever an entering or exiting vehicle is detected.
9   module carSensor (a, b, enter, exit, clk, reset);
10     input logic a, b, clk, reset;
11     output logic enter; // car entering
12     output logic exit; //  car exiting
13
14     enum {none, entering01, exiting01, entering11, exiting11, entering10, exiting10, idle} ps
    , ns;
15
16     // Logic for next state
17     always_comb begin
18        case(ps)
19           none: if (~a & ~b) ns = none;
20                 else if (~a & b) ns = entering01;
21                 else if (a & ~b) ns = exiting10;
22                 else ns = idle;
23           entering01: if (~a & ~b) ns = none;
24                 else if (~a & b) ns = entering01;
25                 else if (a & ~b) ns = idle;
26                 else ns = entering11;
27           exiting01: if (~a & ~b) ns = none;
28                 else if (~a & b) ns = exiting01;
29                 else if (a & ~b) ns = idle;
30                 else ns = exiting11;
31           entering11: if (~a & ~b) ns = none;
32                 else if (~a & b) ns = entering01;
33                 else if (a & ~b) ns = entering10;
34                 else ns = entering11;
35           exiting11: if (~a & ~b) ns = none;
36                 else if (~a & b) ns = exiting01;
37                 else if (a & ~b) ns = exiting10;
38                 else ns = exiting11;
39           entering10: if (~a & ~b) ns = none;
40                 else if (~a & b) ns = idle;
41                 else if (a & ~b) ns = entering10;
42                 else ns = entering11;
43           exiting10: if (~a & ~b) ns = none;
44                 else if (~a & b) ns = idle;
45                 else if (a & ~b) ns = exiting10;
46                 else ns = exiting11;
47           idle: if (~a & ~b) ns = none;
48                 else ns = idle;
49        endcase
50     end // always_comb
51
52     //output logic for exiting: outputs 1 to exit when an exiting vehicle is detected.
53     always_comb begin
54        case(ps)
55           exiting01: if (~a & ~b) exit = 1'b1;
56                      else exit = 1'b0;
57           default: exit = 1'b0;
58        endcase
59     end // always_comb
60
61     //DFFs
62     always_ff @(posedge clk) begin
63        if (reset)
64           ps <= none;
65        else
66           ps <= ns;
67     end // always_ff
68  endmodule // carSensor
69
70  // carSensor_testbench tests all expected, unexpected, and edgecase behaviors
71  module carSensor_testbench();
```

```
72      logic a, b, clk, reset, enter, exit;
73      logic CLOCK_50;
74
75      carSensor dut (.a(b), .b(a), .clk(CLOCK_50), .reset, .enter, .exit);
76
77      // Setting up a clock.
78      parameter CLOCK_PERIOD = 100;
79      initial begin
80         CLOCK_50 <= 0;
81         forever #(CLOCK_PERIOD/2) CLOCK_50 <= ~CLOCK_50; // toggle the clock forever
82      end // initial
83
84      initial begin
85         // reset
86         reset <= 1;                      repeat(3) @(posedge CLOCK_50);
87
88         //enters
89         reset <= 0;       a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50);
90                           a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
91                           a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
92                           a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
93                           a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50);
94         //exits
95                           a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50);
96                           a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
97                           a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
98                           a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
99                           a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50);
100
101        // direction changes while entering
102                          a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50);
103                          a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
104                          a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
105                          a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
106                          a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
107                          a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
108                          a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
109                          a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
110                          a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50);
111
112        // direction changes while exiting
113                          a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50);
114                          a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
115                          a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
116                          a <= 0; b <= 1; repeat(2) @(posedge CLOCK_50);
117                          a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
118                          a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
119                          a <= 1; b <= 1; repeat(2) @(posedge CLOCK_50);
120                          a <= 1; b <= 0; repeat(2) @(posedge CLOCK_50);
121                          a <= 0; b <= 0; repeat(2) @(posedge CLOCK_50);
122        $stop;
123     end // intial
124
125  endmodule // carSensor_testbench
126
```

```
/* Name: Eugene Ngo
   Date: 3/7/2023
   Class: EE 371
   Lab 6: Parking Lot 3D Simulation
*/
// DE1_SoC combines all the modules together, pipelining the control and
// various incrememnt modules to Switches and VGPIO to then
// send them to the datapath module to output to the HEX and RAM modules.
`timescale 1 ps / 1 ps
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;
   input  logic [9:0] SW;
   output logic [9:0] LEDR;
   inout  logic [35:23] V_GPIO;

   logic clk;
   assign clk = CLOCK_50;

   logic [1:0] counter_out;
   logic [3:0] hour_out, incr_out, addr_out;
   logic [31:0] divided_clocks;

   // FPGA input
   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

   // Parking spot 1 occupied and Parking spot 2 occupied and Parking spot 3 occupied
   // = parking lot = full.
   assign V_GPIO[34] = V_GPIO[28] & V_GPIO[29] & V_GPIO[30];   // LED full
   // Parking lot isnt full and presence at entrance = open entrance gate
   assign V_GPIO[31] = ~V_GPIO[34] & V_GPIO[23];   // Open entrance
   // If anyone at exit sensor, open exit gate.
   assign V_GPIO[33] = V_GPIO[24];  // Open exit


   // Helper logic, passed along to different modules.
   logic zeroOccupied;
// logic [6:0] rushHourBegin, rushHourEnd;
   assign zeroOccupied = !(V_GPIO[28] | V_GPIO[29] | V_GPIO[30]);

   logic startRush;
   logic rushEnded;
   logic stopRush;
   logic endGameHexOut;

   // FPGA output, for debugging
   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
   assign LEDR[9] = endGameHexOut;  // Presence exit
   assign LEDR[8] = startRush;
   assign LEDR[7] = stopRush;

   // clockDivide generates slower clocks to process different inputs
   clock_divider clockDivide (.clock(clk), .reset(SW[9]), .divided_clocks(divided_clocks));

   // hourCount counter takes in the KEY[0] signal and increases the
   // current hour, progressing the day in the parking lot
   hourCount timeCounter (.inc(~KEY[0]), .clk(clk), .reset(SW[9]), .out(hour_out));

   // carCount counter takes in the parking spot signals from VGPIO 28-30 and outputs
   // the number of spots left
   carCount #(3) counter (.park1(V_GPIO[28]), .park2(V_GPIO[29]), .park3(V_GPIO[30]), .full(
V_GPIO[34]), .clk(clk), .reset(SW[9]), .out(counter_out));

   // The carIncrCounter, used for saving the values to RAM
   carIncrCounter incrCounter (.inc(V_GPIO[31]), .buttonReset(~KEY[0]), .clk(clk), .reset(SW
```

```
[9]), .out(incr_out));

   // The addressIterator, used for reading the values from RAM
   addrIter addrIterator (.inc(endGameHexOut), .clk(clk), .reset(SW[9]), .out(addr_out));

   // Pipelining the values from hourCount and carCount, we can generate control signals that are used to control the seven segment display.
   controlUnit controls (.occupied(V_GPIO[34]), .noneOccupied(zeroOccupied), .hour(hour_out), .reset(SW[9]), .clk(clk), .startRush(startRush), .stopRush(stopRush), .endGameHexOut(endGameHexOut), .rushEnded(rushEnded));

   // Pipelining the values from control signals to the datapath to generate the rush hour values and then push them to the seg7 display
   dataPathUnit path (.startRush(startRush), .stopRush(stopRush), .endGameHexOut(endGameHexOut), .clk(clk), .in(counter_out), .time_in(hour_out), .incr_in(incr_out), .addr_in(addr_out),
                      .reset(SW[9]), .rushEnded(rushEnded), .slowClk(divided_clocks[25]), .hexout0(HEX0), .hexout1(HEX1), .hexout2(HEX2), .hexout3(HEX3), .hexout4(HEX4), .hexout5(HEX5));

endmodule  // DE1_SoC

// DE1_SoC_testbench tests all expected, unexpected, and edgecase behaviors
module DE1_SoC_testbench();
   logic CLOCK_50;
   logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
   logic [3:0] KEY;
   logic [9:0] SW;
   logic [9:0] LEDR;
   wire [35:23] V_GPIO;

   logic[4:0] sensors;

   assign {V_GPIO[31], V_GPIO[34], V_GPIO[28], V_GPIO[29], V_GPIO[30]} = sensors;

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

   // Setting up the clock.
   parameter CLOCK_PERIOD = 100;
   initial begin
      CLOCK_50 <= 0;
      forever #(CLOCK_PERIOD/2) CLOCK_50 <= ~CLOCK_50; // toggle the clock forever
   end // initial

   initial begin
      SW[9] <= 1;                @(posedge CLOCK_50); // reset
      SW[9] <= 0;                @(posedge CLOCK_50); // inc past max limit
      sensors[0] <= 0;           @(posedge CLOCK_50);
      sensors[0] <= 1;           @(posedge CLOCK_50);
      sensors[0] <= 1;           @(posedge CLOCK_50);
      sensors[0] <= 1;           @(posedge CLOCK_50);
      sensors[0] <= 1;           @(posedge CLOCK_50);
      sensors[1] <= 1;           @(posedge CLOCK_50);
      sensors[1] <= 1;           @(posedge CLOCK_50);
      sensors[1] <= 1;           @(posedge CLOCK_50);
      sensors[1] <= 1;           @(posedge CLOCK_50);
      sensors[1] <= 1;           @(posedge CLOCK_50);
      sensors[1] <= 1;           @(posedge CLOCK_50);
      KEY[0] <= 0;               @(posedge CLOCK_50);
      KEY[0] <= 0;               @(posedge CLOCK_50);
      KEY[0] <= 0;               @(posedge CLOCK_50);
      sensors[2] <= 0;           @(posedge CLOCK_50);
      sensors[3] <= 0;           @(posedge CLOCK_50);
      sensors[4] <= 0;           @(posedge CLOCK_50);
      KEY[0] <= 0;               @(posedge CLOCK_50);
      KEY[0] <= 0;               @(posedge CLOCK_50);
      KEY[0] <= 0;               @(posedge CLOCK_50);
      KEY[0] <= 0;               @(posedge CLOCK_50);
      KEY[0] <= 0;               @(posedge CLOCK_50);
      KEY[0] <= 0;               @(posedge CLOCK_50);
      KEY[0] <= 0;               @(posedge CLOCK_50);
      $stop;
   end // initial
```

```
137     endmodule
```

```
1   /* Name: Eugene Ngo
2      Date: 3/7/2023
3      Class: EE 371
4      Lab 6: Parking Lot 3D Simulation
5   */
6
7   // carCount takes 6 inputs (park1, park2, park3, full, clk, reset.
8   // It sets the state of the number of cars currently left based on
9   // the number of spots available. There is no incrementing or decrementing
10  // of an existing number, this is purely just sequential unit with a moore-like logic
11  // that outputs the number of spots available based on the number
12  // of spots taken. This is derived from the carCount from part 1
13  // therefore, the parameter is kept, but it is not a requirement.
14  `timescale 1 ps / 1 ps
15  module carCount #(parameter MAX=3) (park1, park2, park3, full, clk, reset, out);
16
17     input logic park1, park2, park3, full, clk, reset;
18     output logic [1:0] out;
19
20     // Sequential logic for setting the number of spaces left based on the
21     // parking sensors..
22     always_ff @(posedge clk) begin
23        // If everything is reset, set output to 3 spots left
24        if (reset) begin
25           out <= 2'b11;
26        end
27        // If 0 spots taken, output 3 spots left.
28        if (~park1 & ~park2 & ~park3) begin
29           out <= 2'b11;
30        end
31        // If 1 spot taken, output 2 spots left
32        else if ((park1 & ~park2 & ~park3) | (~park1 & park2 & ~park3) | (~park1 & ~park2 &
    park3)) begin
33           out <= 2'b10;
34        end
35        // If 2 spots taken, output 1 spot left
36        else if ((park1 & park2 & ~park3) | (~park1 & park2 & park3) | (park1 & ~park2 & park3
    )) begin // decrement when not at min
37           out <= 2'b01;
38        end
39        // If 3 spots taken, output 0 spot left (datapath should then convert this to "FULL").
40        else if (full == 1'b1) begin
41           out <= 2'b00;
42        end
43        else
44           out <= out; // hold value otherwise
45     end // always_ff
46
47  endmodule
48
49  // carCount_testbench tests all expected, unexpected, and edgecase behaviors
50  // to ensure the module updates the current number of spots available based on the number
51  // of cars in the parking lot.
52  module carCount_testbench();
53     // Same I/O as carCount()
54     logic park1, park2, park3, full, clk, reset;
55     logic [1:0] out;
56     logic CLOCK_50;
57
58     carCount #(3) dut (.park1, .park2, .park3, .full, .clk(CLOCK_50), .reset, .out);
59
60     // Setting up the clock.
61     parameter CLOCK_PERIOD = 100;
62     initial begin
63        CLOCK_50 <= 0;
64        forever #(CLOCK_PERIOD/2) CLOCK_50 <= ~CLOCK_50; // toggle the clock forever
65     end // initial
66
67     initial begin
68        reset <= 1;                @(posedge CLOCK_50); // reset
69        reset <= 0;                @(posedge CLOCK_50); // inc past max limit
70        park1 <= 0;                @(posedge CLOCK_50); // dec past min limit
71        park1 <= 1;                @(posedge CLOCK_50); // dec past min limit
```

```
72          park2 <= 0;                 @(posedge CLOCK_50); // dec past min limit
73          park2 <= 1;                 @(posedge CLOCK_50); // dec past min limit
74          park3 <= 0;                 @(posedge CLOCK_50); // dec past min limit
75          park3 <= 1;                 @(posedge CLOCK_50); // dec past min limit
76          park3 <= 0;                 @(posedge CLOCK_50); // dec past min limit
77          park2 <= 0;                 @(posedge CLOCK_50); // dec past min limit
78          park3 <= 1;                 @(posedge CLOCK_50); // dec past min limit
79          park1 <= 0;                 @(posedge CLOCK_50); // dec past min limit
80          park3 <= 0;                 @(posedge CLOCK_50); // dec past min limit
81          $stop;
82       end
83    endmodule // counter_testbench
```

```
1   /* Name: Eugene Ngo
2      Date: 3/7/2023
3      Class: EE 371
4      Lab 6: Parking Lot 3D Simulation
5   */
6
7   // addrIter iterates through the different addresses on the RAM
8   // this is used in the datapath unit, once the parking lot is in the
9   // state of showing RAM data. It iterates through the integers
10  // 0 - 7, cycling back to 0 once it reaches 7. It is reset
11  // via the reset input, SW[9].
12  `timescale 1 ps / 1 ps
13  module addrIter (inc, clk, reset, out);
14
15     // Basic I/O. Inc enables incrememenenting the address,
16     // this is triggered once parking lot is at the end of day.
17     // The output is a 4 bit number used to access RAM memory.
18
19     input logic inc, clk, reset;
20     output logic [3:0]  out;
21
22     // Sequential logic for counting up and counting down depending on the input.
23     always_ff @(posedge clk) begin
24        // reset if reset switch is flipped.
25        if (reset) begin
26           out <= 4'b0000;
27        end
28        // incremenet when the counter is not at max (7).
29        else if (inc & out < 4'b1000) begin //increment when not at max
30           out <= out + 4'b0001;
31        end
32        // reset to zero if the value ever exceeds max (7)
33        else if (out > 4'b0111) begin
34           out <= 4'b0000;
35        end
36        // keep the value at out if none of the other conditions are met.
37        else
38           out <= out; // hold value otherwise
39     end // always_ff
40
41  endmodule
42
43  // addrIter_testbench tests all expected, unexpected, and edgecase behaviors
44  // to ensure the module iterates through addresses 0 to 7, making sure the value
45  // output value is updated properly.
46  module addrIter_testbench();
47
48     // Same I/O as addrIter()
49     logic inc, clk, reset;
50     logic [3:0] out;
51     logic CLOCK_50;
52
53     addrIter dut (.inc, .clk(CLOCK_50), .reset, .out);
54
55     // Setting up the clock.
56     parameter CLOCK_PERIOD = 100;
57     initial begin
58        CLOCK_50 <= 0;
59        forever #(CLOCK_PERIOD/2) CLOCK_50 <= ~CLOCK_50; // toggle the clock forever
60     end // initial
61
62     initial begin
63        reset <= 1;                @(posedge CLOCK_50); // reset
64        reset <= 0;                @(posedge CLOCK_50); // inc past max limit
65        inc <= 0;                  repeat(7) @(posedge CLOCK_50); // dec past min limit
66        inc <= 1;                  repeat(30) @(posedge CLOCK_50); // dec past min limit
67        $stop;
68     end
69  endmodule // counter_testbench
```

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

// carIncrCounter takes in an increment and two reset inputs
// to update a stored logic variable that acts as a storage
// of the number of cars that have enterred on a given hour.
// This count is reset everytime the reset switch is flipped
// or if the hour increment button is hit, allowing for a new
// value to be enterred into the RAM storage with the addressIterator's
// address output.
`timescale 1 ps / 1 ps
module carIncrCounter (inc, buttonReset, clk, reset, out);
   // Inc logic is used to incremement the value of the
   // outputted car numbers until it reaches a maximum of 8
   // and then stopping.

   input logic inc, clk, reset, buttonReset;
   output logic [3:0]  out;

   // Sequential logic for counting up and counting down depending on the input.
   always_ff @(posedge clk) begin
      // if Sw[9] or Key[0] is pressed, reset.
      if (reset | buttonReset) begin
         out <= 4'b0000;
      end
      // Otherwise, just increment the value.
      else if (inc & out < 4'b1000) begin //increment when not at max
         out <= out + 4'b0001;
      end
      // If none of the above are met, hold onto the current value.
      else
         out <= out; // hold value otherwise
   end // always_ff

endmodule

// carIncrCounter_testbench tests all expected, unexpected, and edgecase behaviors
// of carIncr, ensuring it counts up to 8, and holds that value, unless the button
// or switch reset values are triggered. THis is used to store values into the RAM.
module carIncrCounter_testbench();
   logic inc, clk, reset, buttonReset;
   logic [3:0] out;
   logic CLOCK_50;

   carIncrCounter  dut (inc, buttonReset, CLOCK_50, reset, out);

   // Setting up the clock.
   parameter CLOCK_PERIOD = 100;
   initial begin
      CLOCK_50 <= 0;
      forever #(CLOCK_PERIOD/2) CLOCK_50 <= ~CLOCK_50; // toggle the clock forever
   end // initial

   initial begin
      reset <= 1;                @(posedge CLOCK_50); // reset
      reset <= 0;                @(posedge CLOCK_50); // inc past max limit
      inc <= 0;                  repeat(7) @(posedge CLOCK_50); // dec past min limit
      inc <= 1;                  repeat(7) @(posedge CLOCK_50); // dec past min limit
      buttonReset <= 1;                @(posedge CLOCK_50); // reset
      buttonReset <= 0;                @(posedge CLOCK_50); // inc past max limit
      inc <= 0;                  repeat(7) @(posedge CLOCK_50); // dec past min limit
      inc <= 1;                  repeat(7) @(posedge CLOCK_50); // dec past min limit
      $stop;
   end
endmodule // counter_testbench
```

```
1   /* Name: Eugene Ngo
2      Date: 3/7/2023
3      Class: EE 371
4      Lab 6: Parking Lot 3D Simulation
5
6   */
7
8   // Clock divider for hardware testing. Takes 1-bit input clock and reset and outputs
9   // 32-bit divided_clocks to get slower clocking for testing on the hardware.
10  // Module was borrowed from EE 271.
11  // divided_clocks[0] = 25MHz, [1] = 12.5Mhz, ... [23] = 3Hz, [24] = 1.5Hz, [25] = 0.75Hz, ..
12  `timescale 1 ps / 1 ps
13  module clock_divider (clock, reset, divided_clocks);
14     input logic reset, clock;
15     output logic [31:0] divided_clocks = 0;
16
17     always_ff @(posedge clock) begin
18        divided_clocks <= divided_clocks + 1;
19     end // always_ff
20
21  endmodule // clock_divider
22
```

```
/* 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
```

```
/* 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
```

```
1   /* Name: Eugene Ngo
2      Date: 3/7/2023
3      Class: EE 371
4      Lab 6: Parking Lot 3D Simulation
5   */
6
7   // hourCount takes 2 inputs (reset, incr) and outputs
8   // the hours counted thus far to progress the day in the parking lot system
9   `timescale 1 ps / 1 ps
10  module hourCount (inc, clk, reset, out);
11
12     input logic inc, clk, reset;
13     output logic [3:0]  out;
14
15     // Sequential logic for counting up and counting down depending on the input.
16     always_ff @(posedge clk) begin
17        if (reset) begin
18           out <= 4'b0000;
19        end
20        else if (inc & out < 4'b1000) begin //increment when not at max
21           out <= out + 4'b0001;
22        end
23        else
24           out <= out; // hold value otherwise
25     end // always_ff
26
27  endmodule
28
29  // hourCount_testbench tests all expected, unexpected, and edgecase behaviors
30  module hourCount_testbench();
31     logic inc, clk, reset;
32     logic [3:0] out;
33     logic CLOCK_50;
34
35     hourCount dut (.inc, .clk(CLOCK_50), .reset, .out);
36
37     // Setting up the clock.
38     parameter CLOCK_PERIOD = 100;
39     initial begin
40        CLOCK_50 <= 0;
41        forever #(CLOCK_PERIOD/2) CLOCK_50 <= ~CLOCK_50; // toggle the clock forever
42     end // initial
43
44     initial begin
45        reset <= 1;                @(posedge CLOCK_50); // reset
46        reset <= 0;                @(posedge CLOCK_50); // inc past max limit
47        inc <= 0;                  repeat(7) @(posedge CLOCK_50); // dec past min limit
48        inc <= 1;                  repeat(7) @(posedge CLOCK_50); // dec past min limit
49        $stop;
50     end
51  endmodule // counter_testbench
```

```
// megafunction wizard: %RAM: 2-PORT%
// GENERATION: STANDARD
// VERSION: WM1.0
// MODULE: altsyncram

// ============================================================
// File Name: RAM8x16.v
// Megafunction Name(s):
//          altsyncram
//
// Simulation Library Files(s):
//          altera_mf
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 17.0.0 Build 595 04/25/2017 SJ Lite Edition
// ************************************************************


//Copyright (C) 2017  Intel Corporation. All rights reserved.
//Your use of Intel Corporation's design tools, logic functions
//and other software and tools, and its AMPP partner logic
//functions, and any output files from any of the foregoing
//(including device programming or simulation files), and any
//associated documentation or information are expressly subject
//to the terms and conditions of the Intel Program License
//Subscription Agreement, the Intel Quartus Prime License Agreement,
//the Intel MegaCore Function License Agreement, or other
//applicable license agreement, including, without limitation,
//that your use is for the sole purpose of programming logic
//devices manufactured by Intel and sold by Intel or its
//authorized distributors.  Please refer to the applicable
//agreement for further details.


// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on
module RAM8x16 (
   clock,
   data,
   rdaddress,
   wraddress,
   wren,
   q);

   input   clock;
   input [3:0]  data;
   input [3:0]  rdaddress;
   input [3:0]  wraddress;
   input   wren;
   output   [7:0]  q;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_off
`endif
   tri1    clock;
   tri0    wren;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_on
`endif

   wire [7:0] sub_wire0;
   wire [7:0] q = sub_wire0[7:0];

   altsyncram  altsyncram_component (
            .address_a (wraddress),
            .address_b (rdaddress),
            .clock0 (clock),
            .data_a (data),
            .wren_a (wren),
            .q_b (sub_wire0),
            .aclr0 (1'b0),
```

```
74                .aclr1 (1'b0),
75                .addressstall_a (1'b0),
76                .addressstall_b (1'b0),
77                .byteena_a (1'b1),
78                .byteena_b (1'b1),
79                .clock1 (1'b1),
80                .clocken0 (1'b1),
81                .clocken1 (1'b1),
82                .clocken2 (1'b1),
83                .clocken3 (1'b1),
84                .data_b ({8{1'b1}}),
85                .eccstatus (),
86                .q_a (),
87                .rden_a (1'b1),
88                .rden_b (1'b1),
89                .wren_b (1'b0));
90        defparam
91           altsyncram_component.address_aclr_b = "NONE",
92           altsyncram_component.address_reg_b = "CLOCK0",
93           altsyncram_component.clock_enable_input_a = "BYPASS",
94           altsyncram_component.clock_enable_input_b = "BYPASS",
95           altsyncram_component.clock_enable_output_b = "BYPASS",
96           altsyncram_component.intended_device_family = "Cyclone V",
97           altsyncram_component.lpm_type = "altsyncram",
98           altsyncram_component.numwords_a = 16,
99           altsyncram_component.numwords_b = 16,
100          altsyncram_component.operation_mode = "DUAL_PORT",
101          altsyncram_component.outdata_aclr_b = "NONE",
102          altsyncram_component.outdata_reg_b = "CLOCK0",
103          altsyncram_component.power_up_uninitialized = "FALSE",
104          altsyncram_component.ram_block_type = "M10K",
105          altsyncram_component.read_during_write_mode_mixed_ports = "DONT_CARE",
106          altsyncram_component.widthad_a = 4,
107          altsyncram_component.widthad_b = 4,
108          altsyncram_component.width_a = 8,
109          altsyncram_component.width_b = 8,
110          altsyncram_component.width_byteena_a = 1;
111
112
113   endmodule
114
115   // ============================================================
116   // CNX file retrieval info
117   // ============================================================
118   // Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
119   // Retrieval info: PRIVATE: ADDRESSSTALL_B NUMERIC "0"
120   // Retrieval info: PRIVATE: BYTEENA_ACLR_A NUMERIC "0"
121   // Retrieval info: PRIVATE: BYTEENA_ACLR_B NUMERIC "0"
122   // Retrieval info: PRIVATE: BYTE_ENABLE_A NUMERIC "0"
123   // Retrieval info: PRIVATE: BYTE_ENABLE_B NUMERIC "0"
124   // Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
125   // Retrieval info: PRIVATE: BlankMemory NUMERIC "1"
126   // Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
127   // Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_B NUMERIC "0"
128   // Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
129   // Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_B NUMERIC "0"
130   // Retrieval info: PRIVATE: CLRdata NUMERIC "0"
131   // Retrieval info: PRIVATE: CLRq NUMERIC "0"
132   // Retrieval info: PRIVATE: CLRrdaddress NUMERIC "0"
133   // Retrieval info: PRIVATE: CLRrren NUMERIC "0"
134   // Retrieval info: PRIVATE: CLRwraddress NUMERIC "0"
135   // Retrieval info: PRIVATE: CLRwren NUMERIC "0"
136   // Retrieval info: PRIVATE: Clock NUMERIC "0"
137   // Retrieval info: PRIVATE: Clock_A NUMERIC "0"
138   // Retrieval info: PRIVATE: Clock_B NUMERIC "0"
139   // Retrieval info: PRIVATE: IMPLEMENT_IN_LES NUMERIC "0"
140   // Retrieval info: PRIVATE: INDATA_ACLR_B NUMERIC "0"
141   // Retrieval info: PRIVATE: INDATA_REG_B NUMERIC "0"
142   // Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_B"
143   // Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
144   // Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
145   // Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
146   // Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
```

```
147   // Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
148   // Retrieval info: PRIVATE: MEMSIZE NUMERIC "128"
149   // Retrieval info: PRIVATE: MEM_IN_BITS NUMERIC "0"
150   // Retrieval info: PRIVATE: MIFfilename STRING ""
151   // Retrieval info: PRIVATE: OPERATION_MODE NUMERIC "2"
152   // Retrieval info: PRIVATE: OUTDATA_ACLR_B NUMERIC "0"
153   // Retrieval info: PRIVATE: OUTDATA_REG_B NUMERIC "1"
154   // Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "2"
155   // Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_MIXED_PORTS NUMERIC "2"
156   // Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_PORT_A NUMERIC "3"
157   // Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_PORT_B NUMERIC "3"
158   // Retrieval info: PRIVATE: REGdata NUMERIC "1"
159   // Retrieval info: PRIVATE: REGq NUMERIC "1"
160   // Retrieval info: PRIVATE: REGrdaddress NUMERIC "1"
161   // Retrieval info: PRIVATE: REGrren NUMERIC "1"
162   // Retrieval info: PRIVATE: REGwraddress NUMERIC "1"
163   // Retrieval info: PRIVATE: REGwren NUMERIC "1"
164   // Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
165   // Retrieval info: PRIVATE: USE_DIFF_CLKEN NUMERIC "0"
166   // Retrieval info: PRIVATE: UseDPRAM NUMERIC "1"
167   // Retrieval info: PRIVATE: VarWidth NUMERIC "0"
168   // Retrieval info: PRIVATE: WIDTH_READ_A NUMERIC "8"
169   // Retrieval info: PRIVATE: WIDTH_READ_B NUMERIC "8"
170   // Retrieval info: PRIVATE: WIDTH_WRITE_A NUMERIC "8"
171   // Retrieval info: PRIVATE: WIDTH_WRITE_B NUMERIC "8"
172   // Retrieval info: PRIVATE: WRADDR_ACLR_B NUMERIC "0"
173   // Retrieval info: PRIVATE: WRADDR_REG_B NUMERIC "0"
174   // Retrieval info: PRIVATE: WRCTRL_ACLR_B NUMERIC "0"
175   // Retrieval info: PRIVATE: enable NUMERIC "0"
176   // Retrieval info: PRIVATE: rden NUMERIC "0"
177   // Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
178   // Retrieval info: CONSTANT: ADDRESS_ACLR_B STRING "NONE"
179   // Retrieval info: CONSTANT: ADDRESS_REG_B STRING "CLOCK0"
180   // Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_A STRING "BYPASS"
181   // Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_B STRING "BYPASS"
182   // Retrieval info: CONSTANT: CLOCK_ENABLE_OUTPUT_B STRING "BYPASS"
183   // Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
184   // Retrieval info: CONSTANT: LPM_TYPE STRING "altsyncram"
185   // Retrieval info: CONSTANT: NUMWORDS_A NUMERIC "16"
186   // Retrieval info: CONSTANT: NUMWORDS_B NUMERIC "16"
187   // Retrieval info: CONSTANT: OPERATION_MODE STRING "DUAL_PORT"
188   // Retrieval info: CONSTANT: OUTDATA_ACLR_B STRING "NONE"
189   // Retrieval info: CONSTANT: OUTDATA_REG_B STRING "CLOCK0"
190   // Retrieval info: CONSTANT: POWER_UP_UNINITIALIZED STRING "FALSE"
191   // Retrieval info: CONSTANT: RAM_BLOCK_TYPE STRING "M10K"
192   // Retrieval info: CONSTANT: READ_DURING_WRITE_MODE_MIXED_PORTS STRING "DONT_CARE"
193   // Retrieval info: CONSTANT: WIDTHAD_A NUMERIC "4"
194   // Retrieval info: CONSTANT: WIDTHAD_B NUMERIC "4"
195   // Retrieval info: CONSTANT: WIDTH_A NUMERIC "8"
196   // Retrieval info: CONSTANT: WIDTH_B NUMERIC "8"
197   // Retrieval info: CONSTANT: WIDTH_BYTEENA_A NUMERIC "1"
198   // Retrieval info: USED_PORT: clock 0 0 0 0 INPUT VCC "clock"
199   // Retrieval info: USED_PORT: data 0 0 8 0 INPUT NODEFVAL "data[7..0]"
200   // Retrieval info: USED_PORT: q 0 0 8 0 OUTPUT NODEFVAL "q[7..0]"
201   // Retrieval info: USED_PORT: rdaddress 0 0 4 0 INPUT NODEFVAL "rdaddress[3..0]"
202   // Retrieval info: USED_PORT: wraddress 0 0 4 0 INPUT NODEFVAL "wraddress[3..0]"
203   // Retrieval info: USED_PORT: wren 0 0 0 0 INPUT GND "wren"
204   // Retrieval info: CONNECT: @address_a 0 0 4 0 wraddress 0 0 4 0
205   // Retrieval info: CONNECT: @address_b 0 0 4 0 rdaddress 0 0 4 0
206   // Retrieval info: CONNECT: @clock0 0 0 0 0 clock 0 0 0 0
207   // Retrieval info: CONNECT: @data_a 0 0 8 0 data 0 0 8 0
208   // Retrieval info: CONNECT: @wren_a 0 0 0 0 wren 0 0 0 0
209   // Retrieval info: CONNECT: q 0 0 8 0 @q_b 0 0 8 0
210   // Retrieval info: GEN_FILE: TYPE_NORMAL RAM8x16.v TRUE
211   // Retrieval info: GEN_FILE: TYPE_NORMAL RAM8x16.inc FALSE
212   // Retrieval info: GEN_FILE: TYPE_NORMAL RAM8x16.cmp FALSE
213   // Retrieval info: GEN_FILE: TYPE_NORMAL RAM8x16.bsf FALSE
214   // Retrieval info: GEN_FILE: TYPE_NORMAL RAM8x16_inst.v FALSE
215   // Retrieval info: GEN_FILE: TYPE_NORMAL RAM8x16_bb.v TRUE
216   // Retrieval info: LIB_FILE: altera_mf
217
```