

## **Section 1: Procedure**

This lab required modifying, implementing, and testing a car parking sensor system on using an online 3D simulation of a parking lot. The system detects vehicle entry and exit, tracks parking occupancy, and displays the status using LEDs and seven-segment displays. It utilizes GPIO inputs to monitor sensors and outputs signals to control gates and indicate when the lot is full. I divided the assignment into the following tasks, based on the major components:

1. No Deliverable for Task 1
2. Parking lot simulation for a working day

### **Task #2: Parking Lot Simulation for a Working Day**

In task two I used the 3D parking lot on labsland to create a system that would keep track of a 3 space parking lot. This lot would be open for 8 hours each day where cars could exit and enter the lot but only take an empty space if it was available. The system also kept track of when the lot of FULL at any time during the 8 hours and would display “FULL” on HEX’s 0-3 if the capacity of the lot(3 cars) was reached. The system also showed the current hour on HEX0 and would update as each hour passed.

The passing of hours was controlled by the hourFSM.sv module which consisted of 10 states. The first state was Idle where the FSM would start if the reset signal was high. It also set the totalCars and current amount of cars to 0-restarting the system. The module then had 8 states that were used for each hour of the work day and would only move to the next state when the incrHour signal was high. This signal would only be high if KEY[0] was pressed. The last state was the endStat state that would also make sure that the done signal was high to indicate the day was done so that the module would display the statistics of that current day. This statistic was held by the datapath module that kept track of the total number of cars, and current cars in the parking lot the hour that rushHour started and ending. These statistics were displayed at the end of each day on the HEX’s where HEX3 showed the hour rush started and HEX2 showed when it ended. HEX0 and HEX1 showed the number of cars correlated to each hour of that day by using a 8x16 ram module to keep the data for that day. I also created a clock divider module to ensure that the timings for exits and enters and other signals were timely changed/recorded. My parkingLotControl module instantiated all these modules and connected my datapath module with my FSM to ensure that my system was working appropriately. THE DE1\_SoC module acted as my top-level module that instantied the V\_GPIOs and connected them to the 3D simulator on labs land.



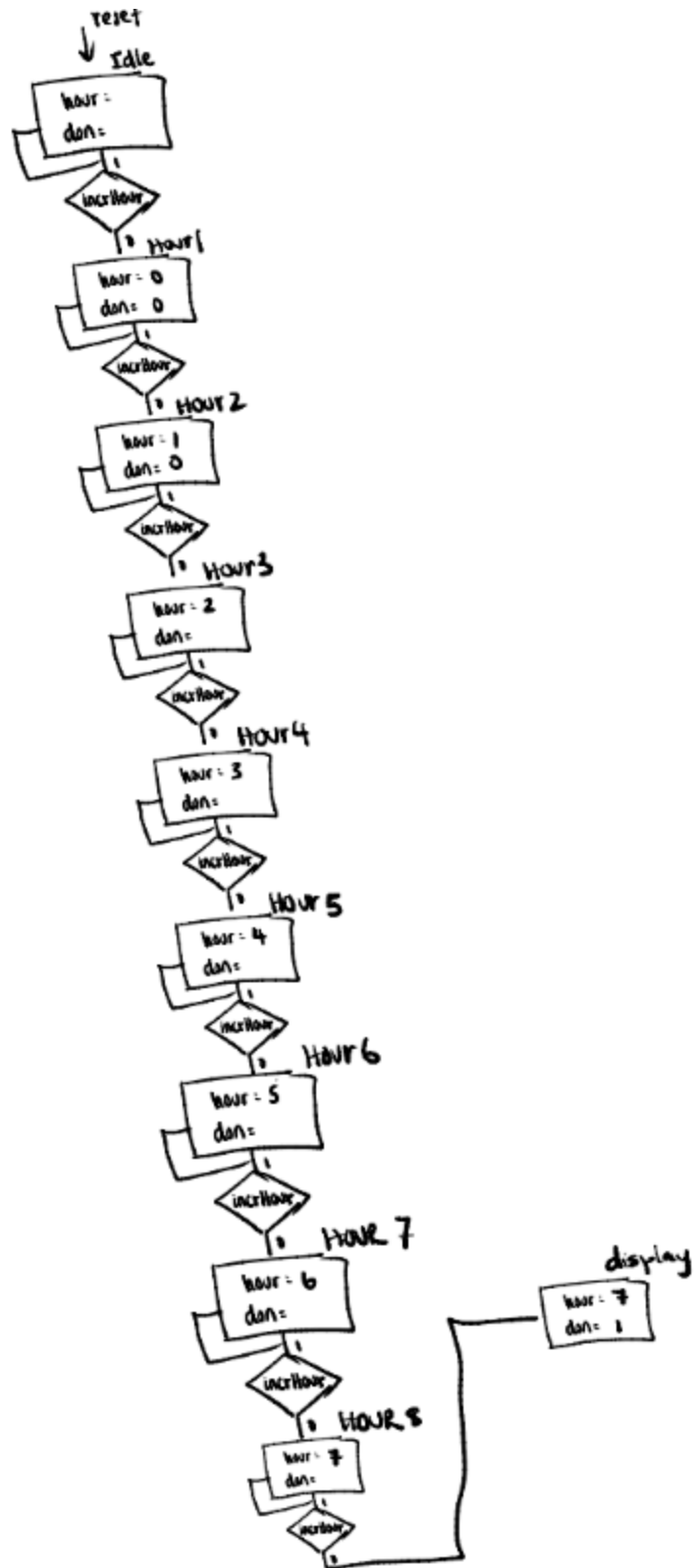


Figure 2: ASMD Diagram of Task 2

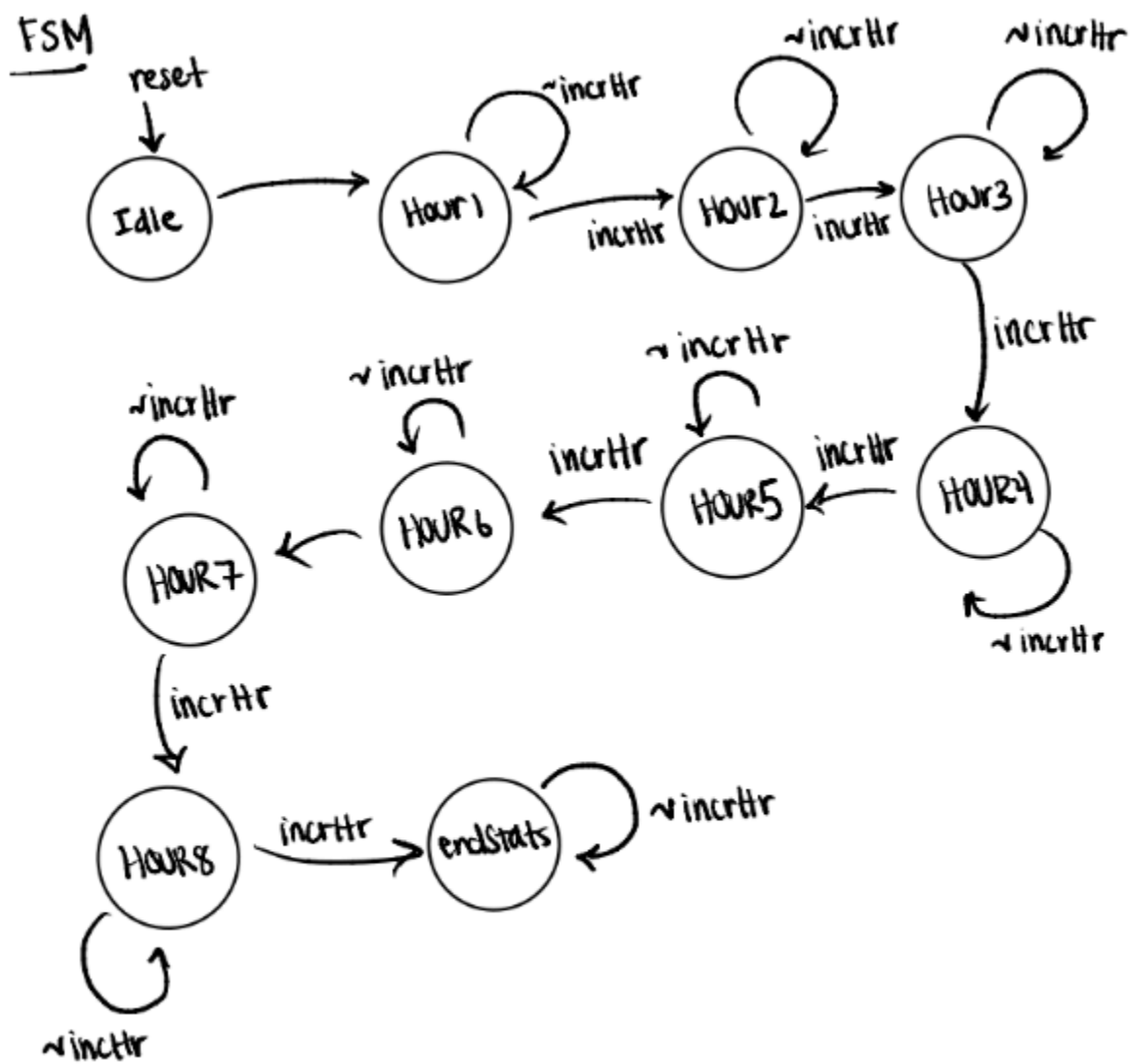


Figure 3: FSM diagram of Task 2

## Section 2: Results

### Task #2: Parking Lot Simulation

Each of the testbench simulations tested the following situations:

1. DE1\_SoC: Top level module

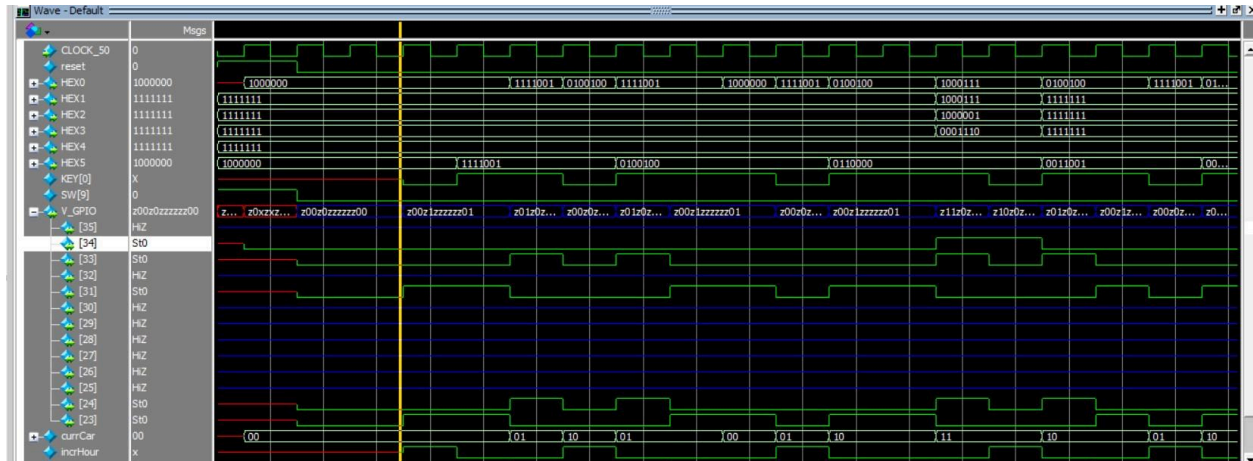


Figure 3.1: ModelSim waveform of DE1\_SoC

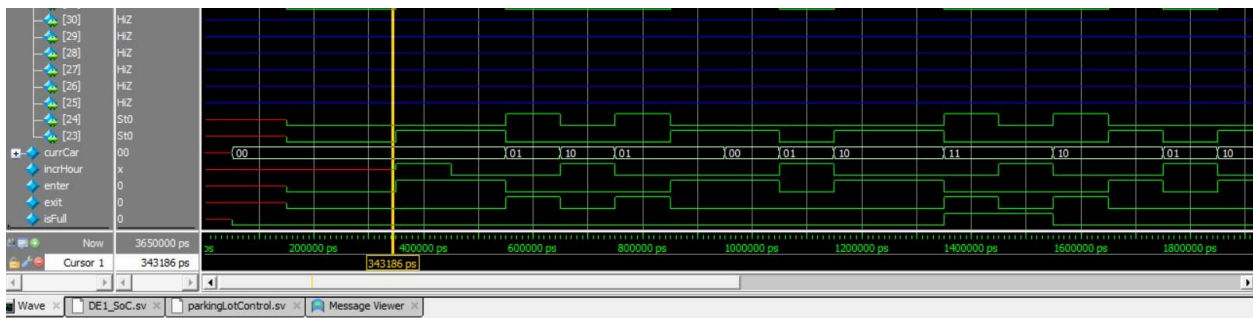
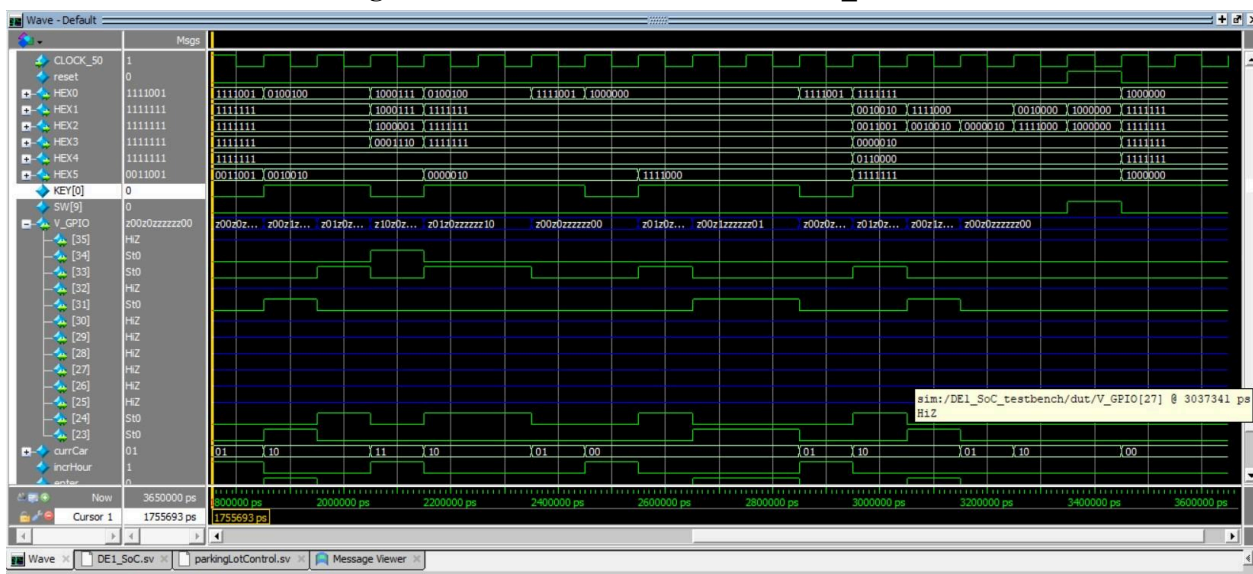
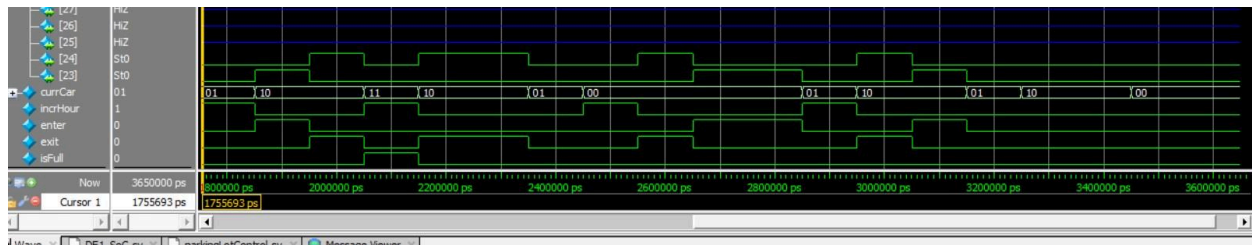


Figure 3.2: ModelSim waveform of DE1\_SoC



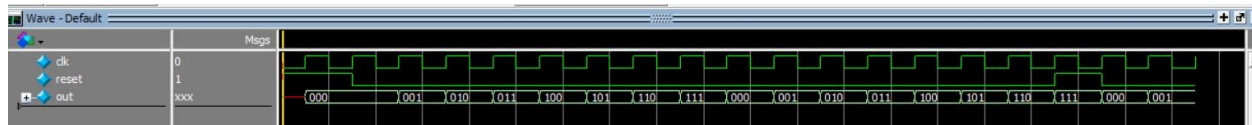
**Figure 3.3: ModelSim waveform of DE1\_SoC**



**Figure 3.4: ModelSim waveform of DE1\_SoC**

As you can see above for DE1\_SoC module I tested all the different cases and simulated in figures 3.1-3.4. In figure 3.1 you can see that as the enter signal goes high currCar increases and as exit goes high currCar decreases to show the exit and entrance of cars in this system. When the curCar count becomes three you can see HEX3 change to FULL. You can also see that the HEX's change when incrHour signal goes high 8 times-signaling the end of the work day. That HEX1 shows the hour and HEX0 shows the corresponding total number of cars in that hour. You can also see HEX3 and HEX2 change after the work day ends as they show the rush hour start and end.

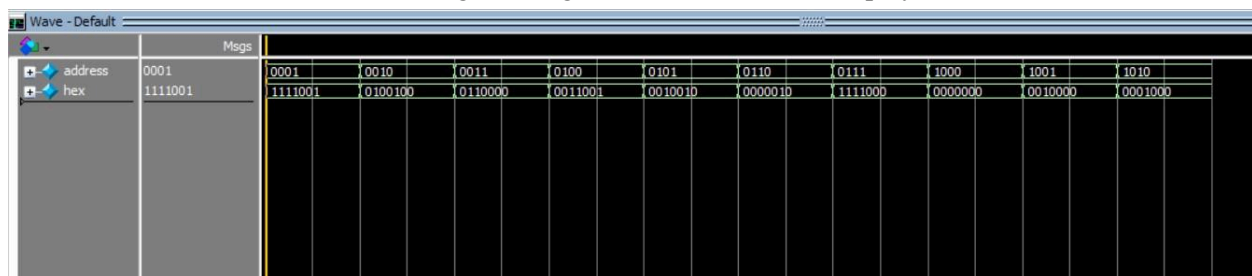
2. **counter\_testbench**: This testbench simulates 7 different types of line drawings, each of which are described below with figures that go along with the description.



**Figure 4: ModelSim waveform of counter**

The counter module is used to iterate through the number 0 - 7 and is later used in to iterate through the addresses in the ram(each number 0-7 representing a work hour). You can see that when reset is high the out value is 0 and that after it gets to 7 it counts again from 0.

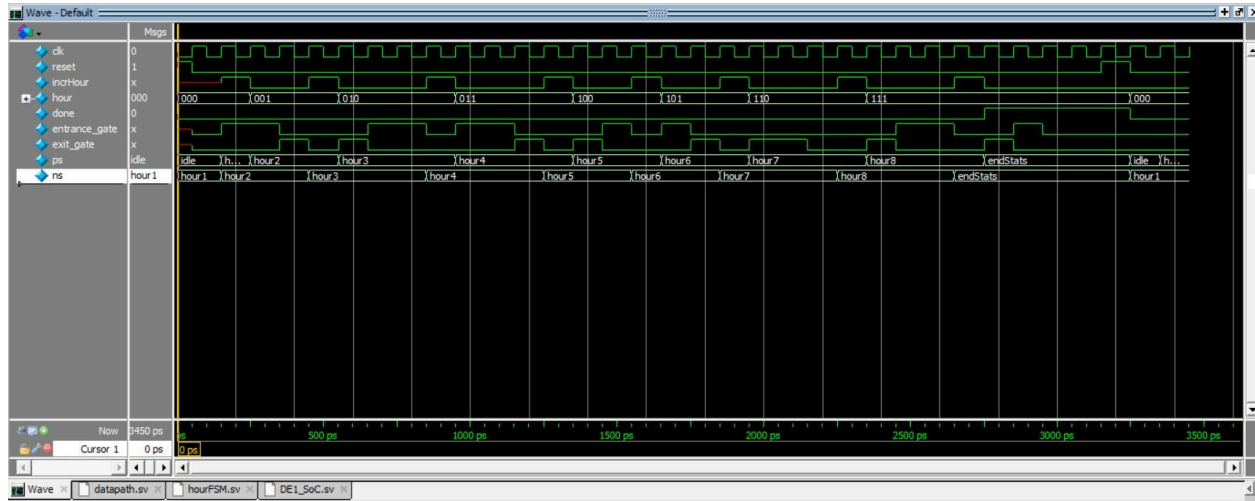
3. **hexDisplay\_testbench**: This testbench simulates different numbers that can be the address input and then is converted to a 7-seg 6-bit signal used for the HEX displays.



**Figure 5: ModelSim waveform of hexDisplay**

The hexDisplay talks in an address signal and outputs a corresponding 7-seg value so that the HEX's are able to display this information. You can see that I have simulated numbers 1-10 and their corresponding hex numbers in the hex signal of this wave form.

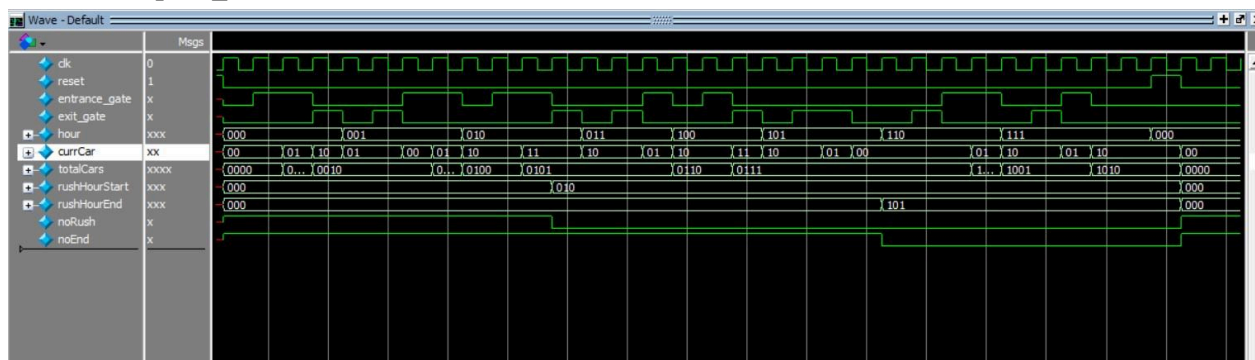
4. **hourFSM\_testbench:** This testbench is for the hourFSM and simulates the different states including the hours 0-8 and the display state.



**Figure 6: ModelSim waveform of hourFSM**

The hourFSM is used to iterate through the work hours. This iteration only happens when the incrHour signal is high (which is KEY[0]). You can see in figure 6 that each time incrHour is high the hour output signal increases by 1. You can also see that when reset is high it stays or goes back to setting hour and done to 0 going back to the idle state. The done signal also goes high at the end of the work day when the display state is ps and if incrHour goes high again it doesn't reset or change the state, it waits for reset in order to start over and go back to the idle state.

## 5. datapath\_testbench:



**Figure 7: ModelSim waveform of datapath**

The datapath module's testbench in figure 7 shows us what happens when a car enters or exits the parking lot in terms of currCar which is the current amount of cars in the parking lot and totalsCars which is the total amount of cars entering the parking lot during that hour in that time. You can see that when the



entrance\_gate signal is high the currCar count increases unless its at capacity (3) and decreases when exit\_gate is high but the totalCars count doesnt decrease until reset is high

## 6. parkingLotControl\_testbench

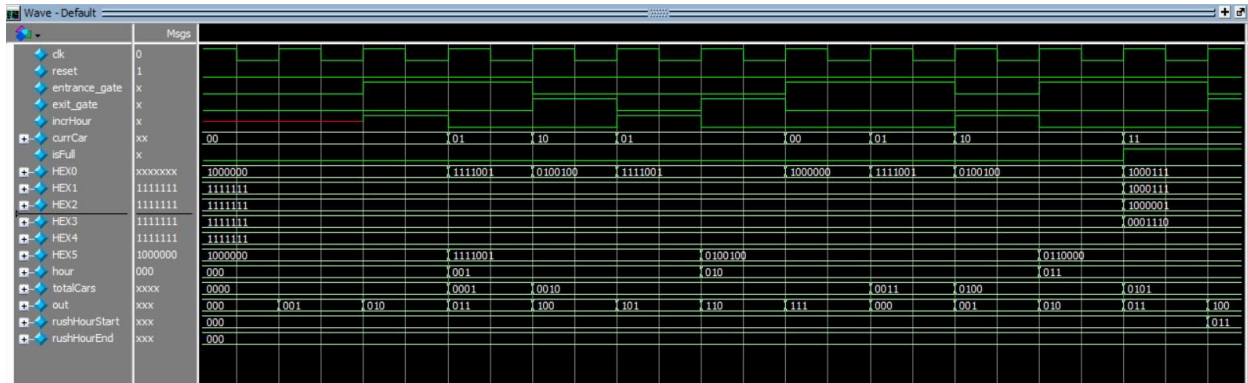


Figure 8.1: ModelSim waveform of parkingLotControl

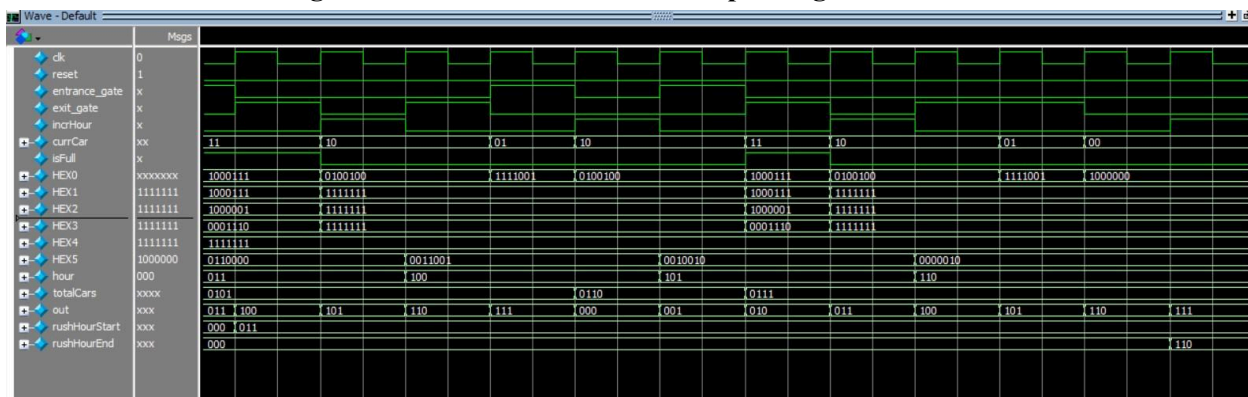


Figure 8.2: ModelSim waveform of parkingLotControl

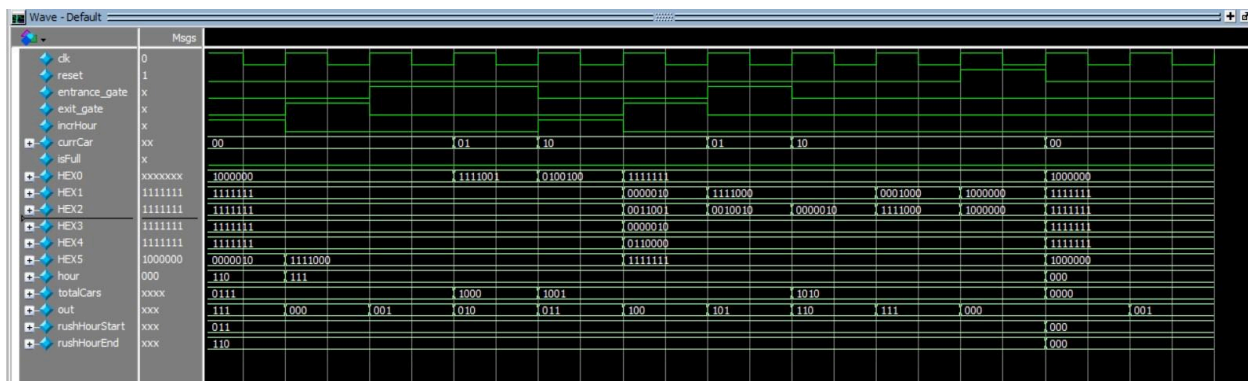


Figure 8.3: ModelSim waveform of parkingLotControl

The parkingLotControl testbench instantiates the other modules and also controls the datapath and FSM signals to make sure that the rest of the system is working cohesively. This module is connected to the



DE1\_SoC module. You can see in figures 8.1-8.3 that the module deals with the scenarios where the hour is increasing and the totalCars is also increasing while the currCar count is increasing and decreasing. You can see this in the HEX signals where HEX5 is constantly changing by showing the current work hour and HEX0 shows the number of spaces that are left in the parking lot. This happens until the hour becomes 8 which signals the end of the work day. Here the HEX's change to show HEX1/HEX0 hour and number of cars each and HEX2/3 showing the rush start and end hour.

### **Section 3: Final Product**

The final product for task 2 was to create a system that connected to the lab's 3D parking simulation. This product included getting more complex from our previously done parking counter. By having a certain work day and only limited parking spots with different data points to juggle the final product was able to create a system that kept track of the number of cars, the cars parked per an hour, the start of a rush hour and the end of it you taught me how to properly keep track of all of my data and further showed me how to use it and connect it to different modules and needs. In order to keep track of each hour's data I also used a 8x16 RAM and used a counter to parse/iterate through that to display it on an HEX display. By combining both ASMD and FSM I was able to visualize my needs which helped me plan out how I needed to implement my code.

### **Section 4: Appendix**

#### **TASK 2:**

## 1) DE1\_SoC

```

1  /*
2  Nithya Subramanian
3  March 11th 2025
4  EE 371
5  Lab 6A, Task 2 */
6
7  /*top level entity of the project */
8  //DE1_SoC uses a 12 bit V_GPIO as inputs and returns a 7bit HEX0, HEX1, HEX2, HEX3, HEX4
9  //HEX5 and 10-bit LEDR as outputs.This display is driven as "0" and nothing as the enter
10 //or exit signal is high it increases or decreases respectivley and displayed on HEX0
11 //and HEX1. When the capacity of the car park is reached(3) HEX0-HEX3 display
12 //"FULL". This serves as the top-level module for the car sensor system
13 //implemented in this lab.
14 module DE1_SoC (CLOCK_50, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, KEY, SW, LEDR, V_GPIO);
15
16     // define ports
17     input logic CLOCK_50;
18     output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
19     input logic [3:0] KEY;
20     input logic [9:0] SW;
21     output logic [9:0] LEDR;
22     inout logic [35:23] V_GPIO;
23
24     logic [1:0] currCar;
25     logic reset;
26     logic incrHour;
27     logic enter, exit;
28
29     //assign clk = CLOCK_50; //for simulation
30     assign reset = SW[9];
31
32     logic isFull;
33     logic [31:0] div_clk;
34
35     logic fullLight;
36
37     // FPGA output
38     assign V_GPIO[26] = V_GPIO[28]; // LED parking 1
39     assign V_GPIO[27] = V_GPIO[29]; // LED parking 2
40     assign V_GPIO[32] = V_GPIO[30]; // LED parking 3
41     assign V_GPIO[34] = isFull; //LED full
42     assign V_GPIO[31] = V_GPIO[23]; // Open entrance
43     assign V_GPIO[33] = V_GPIO[24]; // Open exit
44
45     // FPGA input
46     assign LEDR[0] = V_GPIO[28]; // Presence parking 1
47     assign LEDR[1] = V_GPIO[29]; // Presence parking 2
48     assign LEDR[2] = V_GPIO[30]; // Presence parking 3
49     assign LEDR[3] = V_GPIO[23]; // Presence entrance
50     assign LEDR[4] = V_GPIO[24]; // Presence exit
51
52     clock_divider clockDiv (.clock(CLOCK_50), .divided_clocks(div_clk));
53
54     logic clk1, clk24;
55     parameter whichClock = 26;
56
57     assign clkSelect = CLOCK_50; //for simulation
58
59     //assign clk1 = div_clk[1]; //for exiting
60     //assign clk24 = div_clk[24]; //for entering
61
62     always_ff @(posedge CLOCK_50) begin
63         if (V_GPIO[23] && V_GPIO[31]) begin
64             enter <= 1;
65         end else
66             enter <= 0;
67     end
68
69     always_ff @(posedge CLOCK_50) begin
70         if (V_GPIO[24] | LEDR[4] | V_GPIO[33]) begin
71             exit <= 1;
72         end else
73             exit <= 0;

```

```

74     end
75
76     parkingLotControl control (.clk(CLOCK_50), .reset, .entrance_gate(enter), .exit_gate(exit
77 ), .incrHour(~KEY[0]), .HEX0(HEX0), .HEX1(HEX1), .HEX2(HEX2), .HEX3(HEX3), .HEX4(HEX4), .
78 HEX5(HEX5), .currCar(currCar), .isFull(isFull));
79 endmodule // DE1_SoC
80
81 `timescale 1ns / 1ps
82 //DE1_SoC_testbench tests all expected, unexpected and edgecase behaviors
83 module DE1_SoC_testbench();
84
85     logic CLOCK_50;
86     logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
87     logic [3:0] KEY;
88     logic [9:0] SW;
89     logic [9:0] LEDR;
90     wire [35:23] V_GPIO;
91
92     logic reset;
93     logic [1:0] currCar;
94     logic incrHour;
95     logic isFull;
96     logic enter, exit;
97
98
99     assign KEY[0] = ~incrHour;
100    assign SW[9] = reset;
101    assign V_GPIO[23] = enter; // Presence entrance && enter
102    assign V_GPIO[24] = exit; // Presence exit && exit
103
104    DE1_SoC dut (.CLOCK_50(CLOCK_50), .HEX0(HEX0), .HEX1(HEX1), .HEX2(HEX2), .HEX3(HEX3), .
105    HEX4(HEX4), .HEX5(HEX5), .KEY(KEY), .SW(SW),
106    .LEDR(LEDR), .V_GPIO(V_GPIO));
107
108    parameter CLOCK_PERIOD = 100;
109
110    initial begin
111        CLOCK_50 <= 0;
112        forever #(CLOCK_PERIOD / 2) CLOCK_50 <= ~CLOCK_50;
113    end
114
115    initial begin
116        reset <= 1;
117        reset <= 0; enter <= 0; exit <= 0;
118        reset <= 0; enter <= 0; exit <= 0;
119        reset <= 0; enter <= 0; exit <= 0;
120
121        incrHour <= 1; enter <= 1; exit <= 0; @(posedge CLOCK_50); // Hour 1, count = 1
122        incrHour <= 0; enter <= 1; exit <= 0; @(posedge CLOCK_50); //count = 2
123        incrHour <= 0; enter <= 0; exit <= 1; @(posedge CLOCK_50); //count = 1
124
125        incrHour <= 1; enter <= 0; exit <= 0; @(posedge CLOCK_50); // Hour 2 count = 1
126        incrHour <= 0; enter <= 0; exit <= 1; @(posedge CLOCK_50); //count = 0
127        incrHour <= 0; enter <= 1; exit <= 0; @(posedge CLOCK_50); //count = 1
128        incrHour <= 0; enter <= 1; exit <= 0; @(posedge CLOCK_50); //count = 2
129
130        incrHour <= 1; enter <= 0; exit <= 0; @(posedge CLOCK_50); // Hour 3 count = 2
131        incrHour <= 0; enter <= 1; exit <= 0; @(posedge CLOCK_50); //count = 3 //RUSH START
132        incrHour <= 0; enter <= 1; exit <= 0; @(posedge CLOCK_50); //count = 3
133        incrHour <= 0; enter <= 0; exit <= 1; @(posedge CLOCK_50); //count = 2
134
135        incrHour <= 1; enter <= 0; exit <= 0; @(posedge CLOCK_50); // Hour 4 count = 2
136        incrHour <= 0; enter <= 0; exit <= 1; @(posedge CLOCK_50); //count = 1
137        incrHour <= 0; enter <= 1; exit <= 0; @(posedge CLOCK_50); //count = 2
138
139        incrHour <= 1; enter <= 0; exit <= 0; @(posedge CLOCK_50); // Hour 5 count = 2
140        incrHour <= 0; enter <= 1; exit <= 0; @(posedge CLOCK_50); //count = 3
141        incrHour <= 0; enter <= 0; exit <= 1; @(posedge CLOCK_50); //count = 2
142
143        incrHour <= 1; enter <= 0; exit <= 0; @(posedge CLOCK_50); // Hour 6 count = 2

```

```

144     incrHour <= 0; enter <= 0; exit <= 1; @(posedge CLOCK_50); //count = 1
145     incrHour <= 0; enter <= 0; exit <= 1; @(posedge CLOCK_50); //count = 0 //RUSH ENDS
146     incrHour <= 0; enter <= 0; exit <= 0; @(posedge CLOCK_50); //count = 0
147
148     incrHour <= 1; enter <= 0; exit <= 0; @(posedge CLOCK_50); // Hour 7 count = 0
149     incrHour <= 0; enter <= 0; exit <= 1; @(posedge CLOCK_50); //count = 0
150     incrHour <= 0; enter <= 1; exit <= 0; @(posedge CLOCK_50); //count = 1
151     incrHour <= 0; enter <= 1; exit <= 0; @(posedge CLOCK_50); //count = 2
152
153     incrHour <= 1; enter <= 0; exit <= 0; @(posedge CLOCK_50); // Hour 8 count = 2
154     incrHour <= 0; enter <= 0; exit <= 1; @(posedge CLOCK_50); //count = 1
155     incrHour <= 0; enter <= 1; exit <= 0; @(posedge CLOCK_50); //count = 2
156     incrHour <= 0; enter <= 0; exit <= 0; @(posedge CLOCK_50);
157     incrHour <= 0; enter <= 0; exit <= 0; @(posedge CLOCK_50);
158
159     reset <= 1;          @(posedge CLOCK_50);
160     reset <= 0;          @(posedge CLOCK_50);
161                          @(posedge CLOCK_50);
162     $stop();
163     end
164 endmodule

```

## 2) Counter

```
1  /*
2  Nithya Subramanian
3  March 11th 2025
4  EE 371
5  Lab 6A, Task 2 */
6
7
8  //Counter takes in two 1-bit inputs called clk and reset and returns a 2-bit value called
9  //out. It cycles through the "hours" 0 - 7 and resets to 0 either after the 7th hour
10 //or when reset signal is high
11 module counter (clk, reset, out);
12
13     input logic clk, reset;
14     output logic [2:0] out;
15
16     always_ff @(posedge clk) begin
17         if(reset) begin
18             out <= 0;
19         end else begin
20             if(out < 7) begin
21                 out <= out + 1;
22             end else begin
23                 out <= 0;
24             end
25         end
26     end
27 endmodule
28
29 //testbench for counter tests all expected, unexpected and edgecase behaviors
30 module counter_testbench ();
31     logic clk, reset;
32     logic [2:0] out;
33
34     counter dut (.clk(clk), .reset(reset), .out(out));
35
36     parameter CLOCK_PERIOD = 100;
37
38     initial begin
39         clk <= 0;
40         forever #(CLOCK_PERIOD / 2) clk <= ~clk;
41     end
42
43     initial begin
44         reset <= 1;          @(posedge clk);
45                             @(posedge clk);
46         reset <= 0;          @(posedge clk);
47                             @(posedge clk);
48                             @(posedge clk);
49                             @(posedge clk);
50                             @(posedge clk);
51                             @(posedge clk);
52                             @(posedge clk);
53                             @(posedge clk);
54                             @(posedge clk);
55                             @(posedge clk);
56                             @(posedge clk);
57                             @(posedge clk);
58                             @(posedge clk);
59                             @(posedge clk);
60         reset <= 1;          @(posedge clk);
61         reset <= 0;          @(posedge clk);
62                             @(posedge clk);
63         $stop();
64     end
65 endmodule
```

### 3) hourFSM

```
1  /*
2  Nithya Subramanian
3  March 11th 2025
4  EE 371
5  Lab 6A, Task 2 */
6
7  // a state machine that increases the 3-bit hour signal when the input signal of
8  // incrHour is high and goes into the display state when we have completed
9  // a full day. It also outputs a one-bit signal that tells us when the day
10 // is complete and takes in 1-bit clk, reset, incrHour and.
11 module hourFSM (clk, reset, entrance_gate, exit_gate, incrHour, hour, done);
12     input logic clk, reset, incrHour;
13     input logic entrance_gate, exit_gate;
14
15     output logic [2:0] hour;
16     output logic done;
17
18     enum {idle, hour1, hour2, hour3, hour4, hour5, hour6, hour7, hour8, endStats} ps, ns;
19
20     //the state logic for moving through each work hour
21     //if incrHour is high it moves to next state else it stays in same state
22     //the done signal is only high in the endStats state and hour increases
23     always_comb begin
24         case(ps)
25             idle: begin
26                 hour = 0;
27                 done = 0;
28                 ns = hour1;
29             end
30
31             hour1: begin
32                 hour = 0;
33                 done = 0;
34
35                 if(incrHour)
36                     ns = hour2;
37                 else
38                     ns = hour1;
39             end
40
41             hour2: begin
42                 hour = 1;
43                 done = 0;
44
45                 if(incrHour)
46                     ns = hour3;
47                 else
48                     ns = hour2;
49             end
50
51             hour3: begin
52                 hour = 2;
53                 done = 0;
54
55                 if(incrHour)
56                     ns = hour4;
57                 else
58                     ns = hour3;
59             end
60
61             hour4: begin
62                 hour = 3;
63                 done = 0;
64
65                 if(incrHour)
66                     ns = hour5;
67                 else
68                     ns = hour4;
69             end
70
71             hour5: begin
72                 hour = 4;
73                 done = 0;
```

```
74
75     if(incrHour)
76         ns = hour6;
77     else
78         ns = hour5;
79 end
80
81 hour6: begin
82     hour = 5;
83     done = 0;
84     if(incrHour)
85         ns = hour7;
86     else
87         ns = hour6;
88 end
89
90 hour7: begin
91     hour = 6;
92     done = 0;
93     if(incrHour)
94         ns = hour8;
95     else
96         ns = hour7;
97 end
98
99 hour8: begin
100     hour = 7;
101     done = 0;
102     if(incrHour)
103         ns = endStats;
104     else
105         ns = hour8;
106 end
107
108 endStats: begin
109     hour = 7;
110     done = 1;
111     if(incrHour) begin
112         hour = 0;
113         ns = hour1;
114     end
115     else
116         ns = endStats;
117     end
118 endcase
119 end
120
121 //if reset ps goes to idle state else it goes to ns
122 always_ff @(posedge clk) begin
123     if(reset) begin
124         ps <= idle;
125     end else begin
126         ps <= ns;
127     end
128 end
129 endmodule
130
131 //testbench for hourFSM tests all expected, unexpected and edgecase behaviors
132 module hourFSM_testbench();
133     logic clk, reset, incrHour;
134     logic entrance_gate, exit_gate;
135     logic [2:0] hour;
136     logic done;
137
138     hourFSM dut (.*);
139
140     parameter CLOCK_PERIOD = 100;
141
142     initial begin
```



```

147     clk <= 0;
148
149     forever #(CLOCK_PERIOD / 2) clk <= ~clk;
150 end
151
152 initial begin
153
154     reset <= 1;                                     @(posedge clk);
155     reset <= 0; entrance_gate <= 0; exit_gate <= 0; hour <= 0;    @(posedge clk);
156     incrHour <= 1; entrance_gate <= 1; exit_gate <= 0; @ (posedge clk); // Hour 1, count
157
158     = 1
159     incrHour <= 0; entrance_gate <= 1; exit_gate <= 0; @ (posedge clk); //count = 2
160     incrHour <= 0; entrance_gate <= 0; exit_gate <= 1; @ (posedge clk); //count = 1
161
162     incrHour <= 1; entrance_gate <= 0; exit_gate <= 0; @ (posedge clk); // Hour 2 count = 1
163     incrHour <= 0; entrance_gate <= 0; exit_gate <= 1; @ (posedge clk); //count = 0
164     incrHour <= 0; entrance_gate <= 1; exit_gate <= 0; @ (posedge clk); //count = 1
165     incrHour <= 0; entrance_gate <= 1; exit_gate <= 0; @ (posedge clk); //count = 2
166
167     incrHour <= 1; entrance_gate <= 0; exit_gate <= 0; @ (posedge clk); // Hour 3 count = 2
168     incrHour <= 0; entrance_gate <= 0; exit_gate <= 0; @ (posedge clk); //count = 3
169
170     //RUSH START
171     incrHour <= 0; entrance_gate <= 1; exit_gate <= 0; @ (posedge clk); //count = 3
172     incrHour <= 0; entrance_gate <= 0; exit_gate <= 1; @ (posedge clk); //count = 2
173
174     2
175     incrHour <= 1; entrance_gate <= 0; exit_gate <= 0; @ (posedge clk); // Hour 4 count =
176     incrHour <= 0; entrance_gate <= 0; exit_gate <= 1; @ (posedge clk); //count = 1
177     incrHour <= 0; entrance_gate <= 1; exit_gate <= 0; @ (posedge clk); //count = 2
178
179     incrHour <= 1; entrance_gate <= 0; exit_gate <= 0; @ (posedge clk); // Hour 5 count = 2
180     incrHour <= 0; entrance_gate <= 0; exit_gate <= 1; @ (posedge clk); //count = 3
181     incrHour <= 0; entrance_gate <= 0; exit_gate <= 1; @ (posedge clk); //count = 2
182
183     incrHour <= 1; entrance_gate <= 0; exit_gate <= 0; @ (posedge clk); // Hour 6 count = 2
184     incrHour <= 0; entrance_gate <= 0; exit_gate <= 1; @ (posedge clk); //count = 1
185     incrHour <= 0; entrance_gate <= 0; exit_gate <= 1; @ (posedge clk); //count = 0
186
187     //RUSH ENDS
188     incrHour <= 0; entrance_gate <= 0; exit_gate <= 0; @ (posedge clk); //count = 0
189
190     incrHour <= 1; entrance_gate <= 0; exit_gate <= 0; @ (posedge clk); // Hour 7 count = 0
191     incrHour <= 0; entrance_gate <= 0; exit_gate <= 1; @ (posedge clk); //count = 0
192     incrHour <= 0; entrance_gate <= 1; exit_gate <= 0; @ (posedge clk); //count = 1
193     incrHour <= 0; entrance_gate <= 1; exit_gate <= 0; @ (posedge clk); //count = 2
194
195     incrHour <= 1; entrance_gate <= 0; exit_gate <= 0; @ (posedge clk); // Hour 8 count = 2
196     incrHour <= 0; entrance_gate <= 0; exit_gate <= 1; @ (posedge clk); //count = 1
197     incrHour <= 0; entrance_gate <= 1; exit_gate <= 0; @ (posedge clk); //count = 2
198     incrHour <= 0; entrance_gate <= 0; exit_gate <= 0; @ (posedge clk);
199
200     reset <= 1;                                     @(posedge clk);
201     reset <= 0;                                     @(posedge clk);
202
203     $stop;
204 end
205 endmodule

```

#### 4) hexDisplay

```
1  /*
2  Nithya Subramanian
3  March 11th 2025
4  EE 371
5  Lab 6A, Task 2 */
6
7  //Takes in the address with a 4-bit input signal and outputs a converted
8  //7-seg 7-bit signal for the number that is inputed.
9  module hexDisplay (address, hex);
10     input logic [3:0] address;
11     output logic [6:0] hex;
12
13     always_comb begin
14         case (address)
15
16             0: hex = 7'b1000000; //0
17             1: hex = 7'b1111001; //1
18             2: hex = 7'b0100100; //2
19             3: hex = 7'b0110000; //3
20             4: hex = 7'b0011001; //4
21             5: hex = 7'b0010010; //5
22             6: hex = 7'b0000010; //6
23             7: hex = 7'b1111000; //7
24             8: hex = 7'b0000000; //8
25             9: hex = 7'b0010000; //9
26             10: hex = 7'b0001000; //A
27             11: hex = 7'b0000011; //b
28             12: hex = 7'b1000110; //C
29             13: hex = 7'b0100001; //d
30             14: hex = 7'b0000110; //E
31             15: hex = 7'b0001110; //F
32         endcase
33     end
34 endmodule
35
36 // hexDisplay_testbench simulates all scenarios
37 module hexDisplay_testbench();
38     logic [3:0] address;
39     logic [6:0] hex;
40
41
42     hexDisplay dut(.address, .hex);
43
44     initial begin
45         address = 4'b0001; #10; //1
46         address = 4'b0010; #10; //2
47         address = 4'b0011; #10; //3
48         address = 4'b0100; #10; //4
49         address = 4'b0101; #10; //5
50         address = 4'b0110; #10; //6
51         address = 4'b0111; #10; //7
52         address = 4'b1000; #10; //8
53         address = 4'b1001; #10; //9
54         address = 4'b1010; #10; //10
55         $stop();
56     end
57 endmodule
58
59
```

## 5) Datapath

```
1  /*
2  Nithya Subramanian
3  March 11th 2025
4  EE 371
5  Lab 6A, Task 2 */
6
7  // This module handles the flow of car data within the system. It
8  // includes the logic for updating and maintaining the parking
9  // lot count, managing signals related to entry and exit,
10 // and interacting with other components of the system. This module
11 // also outputs the rushHourStart and end hours
12 module datapath (clk, reset, entrance_gate, exit_gate, hour, currCar, noRush,
13                 noEnd, totalCars, rushHourStart, rushHourEnd);
14
15     input logic clk, reset, entrance_gate, exit_gate;
16     input logic [2:0] hour;
17
18     output logic [1:0] currCar;
19     output logic [3:0] totalCars;
20     output logic [2:0] rushHourStart, rushHourEnd;
21     output logic noRush, noEnd;
22
23     logic rush;
24
25     assign noRush = ~rush;
26
27     //adds one to the currCar count if there is a car at the entrance and there
28     //is a spot open and it decreases the currCar count if there is a car at exit
29     //and there was atleast one car already parked
30     always_ff @(posedge clk) begin
31         if(reset) begin
32             currCar <= 0;
33             totalCars <= 0;
34         end else begin
35             if(entrance_gate & (currCar < 3)) begin
36                 currCar <= currCar + 1;
37                 totalCars <= totalCars + 1;
38             end
39
40             if(exit_gate & (currCar > 0)) begin
41                 currCar <= currCar - 1;
42             end
43         end
44     end
45
46     //Controls the rush hour data by resetting values to 0 except for noEnd = 1
47     //is high and when there is no previous rush and the total number of cars
48     //is 3 then rush hour has started and if the total count of cars is 0 but
49     //previously the rush hasn't ended you can say it ended now and that hour
50     //is the end hour of rush.
51     always_ff @(posedge clk) begin
52         if(reset) begin
53             rushHourStart <= 0;
54             rushHourEnd <= 0;
55             rush <= 0;
56             noEnd <= 1;
57         end
58
59         else begin
60             if((rush == 0) && (currCar == 3)) begin
61                 rush <= 1;
62                 rushHourStart <= hour;
63             end
64
65             if((rush == 1) && (noEnd == 1) && (currCar == 0)) begin
66                 noEnd <= 0;
67                 rushHourEnd <= hour;
68             end
69         end
70     end
71 endmodule //datapath
72
73
```

```

74 //testbench for datapath tests all expected, unexpected and edgecase behaviors
75 module datapath_testbench();
76     logic clk, reset, entrance_gate, exit_gate;
77     logic [2:0] hour;
78
79     logic [1:0] currCar;
80     logic [3:0] totalCars;
81     logic [2:0] rushHourStart, rushHourEnd;
82     logic noRush, noEnd;
83
84     logic rush;
85
86     datapath dut (.*) ;
87
88     parameter CLOCK_PERIOD = 100;
89
90     initial begin
91         clk <= 0;
92
93         forever #(CLOCK_PERIOD / 2) clk <= ~clk;
94     end
95
96     initial begin
97         reset <= 1;
98         reset <= 0; entrance_gate <= 0; exit_gate <= 0; hour <= 0;
99         hour <= 0; entrance_gate <= 1; exit_gate <= 0;
100        hour <= 0; entrance_gate <= 1; exit_gate <= 0;
101        hour <= 0; entrance_gate <= 0; exit_gate <= 1;
102
103        hour <= 1; entrance_gate <= 0; exit_gate <= 0;
104        hour <= 1; entrance_gate <= 0; exit_gate <= 1;
105        hour <= 1; entrance_gate <= 1; exit_gate <= 0;
106        hour <= 1; entrance_gate <= 1; exit_gate <= 0;
107
108        hour <= 2; entrance_gate <= 0; exit_gate <= 0;
109        hour <= 2; entrance_gate <= 1; exit_gate <= 0;
110
111        hour <= 2; entrance_gate <= 1; exit_gate <= 0;
112        hour <= 2; entrance_gate <= 0; exit_gate <= 1;
113
114        hour <= 3; entrance_gate <= 0; exit_gate <= 0;
115        hour <= 3; entrance_gate <= 0; exit_gate <= 1;
116        hour <= 3; entrance_gate <= 1; exit_gate <= 0;
117
118        hour <= 4; entrance_gate <= 0; exit_gate <= 0;
119        hour <= 4; entrance_gate <= 1; exit_gate <= 0;
120        hour <= 4; entrance_gate <= 0; exit_gate <= 1;
121
122        hour <= 5; entrance_gate <= 0; exit_gate <= 0;
123        hour <= 5; entrance_gate <= 0; exit_gate <= 1;
124        hour <= 5; entrance_gate <= 0; exit_gate <= 0;
125        hour <= 5; entrance_gate <= 0; exit_gate <= 0;
126
127        hour <= 6; entrance_gate <= 0; exit_gate <= 0;
128        hour <= 6; entrance_gate <= 0; exit_gate <= 1;
129        hour <= 6; entrance_gate <= 1; exit_gate <= 0;
130        hour <= 6; entrance_gate <= 1; exit_gate <= 0;
131
132        hour <= 7; entrance_gate <= 0; exit_gate <= 0;
133        hour <= 7; entrance_gate <= 0; exit_gate <= 1;
134        hour <= 7; entrance_gate <= 1; exit_gate <= 0;
135        hour <= 7; entrance_gate <= 0; exit_gate <= 0;
136
137        reset <= 1; hour <= 0;
138        reset <= 0;
139
140
141        $stop;
142    end
143 endmodule
144
145

```



## 6) parkingLotControl

```

1  /*
2  Nithya Subramanian
3  March 11th 2025
4  EE 371
5  Lab 6A, Task 2 */
6  // This module manages vehicle entry and exit using sensors. It
7  // keeps track of the number of cars in the lot, ensures the count
8  // does not exceed capacity, and controls entry/exit signals.
9  // The counter increments when a car enters and decrements when a
10 // car exits, ensuring real-time monitoring of availability.
11
12 `timescale 1ns / 1ps
13
14 module parkingLotControl (clk, reset, entrance_gate, exit_gate, incrHour,
15                           HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, currCar, isFull);
16
17     input logic clk, reset, entrance_gate, exit_gate, incrHour;
18
19     output logic [1:0] currCar;
20     output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
21     output logic isFull;
22
23     logic [2:0] hour;
24     logic [3:0] totalCars;
25     logic [2:0] rushHourStart, rushHourEnd;
26     logic noRush, noEnd;
27     logic done;
28
29     logic [6:0] displayCars, displayTime, displayCurr, displayAddr, disRushStart, disRushEnd;
30     logic [3:0] storedCarValue;
31     logic [2:0] out;
32
33     logic [31:0] div_clk;
34
35     clock_divider divClk (.clock(clk), .divided_clocks(div_clk));
36
37     logic clkSelect;
38     parameter whichClock = 25;
39
40     assign clkSelect = clk;
41     //assign clkSelect = div_clk[whichClock];
42
43     assign isFull = (currCar == 3);
44
45     hourFSM fsm(.clk(clkSelect), .reset, .entrance_gate, .exit_gate, .incrHour, .hour, .done);
46
47     datapath carData(.clk(clkSelect), .reset, .entrance_gate, .exit_gate, .hour, .currCar,
48                     .noRush, .noEnd, .totalCars, .rushHourStart, .rushHourEnd);
49
50     counter count(.clk(clkSelect), .reset, .out);
51
52     ram8x16 ram(.clock(clkSelect), .data(totalCars), .rdaddress(out), .wraddress(hour), .wren
53 (1'b1), .q(storedCarValue));
54
55     //Display assignments for different types of data
56     hexDisplay dispTime (.address({1'b0, hour}), .hex(displayTime)); //shows current hour
57     hexDisplay dispCars (.address({2'b0, currCar}), .hex(displayCurr)); //shows current car
58     count
59     hexDisplay dispRushStart (.address({1'b0, rushHourStart}), .hex(disRushStart)); //shows
60     rush's start hour
61     hexDisplay dispRushEnd (.address({1'b0, rushHourEnd}), .hex(disRushEnd)); //shows rush's
62     end hour
63     hexDisplay dispOut (.address({1'b0, out}), .hex(displayAddr)); //shows address when
64     looping through
65     hexDisplay dispStoredCars (.address(storedCarValue), .hex(displayCars)); //shows # of
66     cars in RAM
67
68     //combinatinoal logic that helps decide what should be displayed on each HEX
69     always_comb begin
70         if(!done) begin
71             if(currCar == 3) begin
72                 HEX5 = displayTime;
73                 HEX4 = 7'b1111111;
74             end
75         end
76     end
77

```

```

68         HEX3 = 7'b0001110; //F
69         HEX2 = 7'b1000001; //U
70         HEX1 = 7'b1000111; //L
71         HEX0 = 7'b1000111; //L
72     end
73
74     else begin
75         HEX5 = displayTime;
76         HEX4 = 7'b1111111;
77         HEX3 = 7'b1111111;
78         HEX2 = 7'b1111111;
79         HEX1 = 7'b1111111;
80         HEX0 = displayCurr;
81     end
82 end
83
84     else begin
85         HEX5 = 7'b1111111;
86         HEX4 = (noRush) ? 7'b0111111 : disRushStart;
87         HEX3 = (noEnd) ? 7'b0111111 : disRushEnd;
88         HEX2 = displayAddr;
89         HEX1 = displayCars;
90         HEX0 = 7'b1111111;
91     end
92 end
93 endmodule
94
95
96 //parkingLotControl testbench that simulates all scenarios.
97 module parkingLotControl_testbench();
98     logic clk, reset, entrance_gate, exit_gate, incrHour;
99
100     logic [1:0] currCar;
101     logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
102     logic isFull;
103
104     logic [2:0] hour;
105     logic [3:0] totalCars;
106     logic [2:0] rushHourStart, rushHourEnd;
107     logic noRush, noEnd;
108     logic done;
109
110     logic rush;
111
112     logic [6:0] displayCars, displayTime, displayCurr, displayAddr, disRushStart, disRushEnd;
113     logic [3:0] storedCarValue;
114     logic [2:0] out;
115
116     parkingLotControl dut (.*);
117
118     parameter CLOCK_PERIOD = 100;
119
120     initial begin
121         clk <= 0;
122         forever #(CLOCK_PERIOD / 2) clk <= ~clk;
123     end
124
125     initial begin
126         reset <= 1;
127         @(posedge clk);
128         reset <= 0; entrance_gate <= 0; exit_gate <= 0;
129         @(posedge clk);
130
131         incrHour <= 1; entrance_gate <= 1; exit_gate <= 0; @(posedge clk); // Hour 1, count
132         = 1
133         incrHour <= 0; entrance_gate <= 1; exit_gate <= 0; @(posedge clk); //count = 2
134         incrHour <= 0; entrance_gate <= 0; exit_gate <= 1; @(posedge clk); //count = 1
135
136         incrHour <= 1; entrance_gate <= 0; exit_gate <= 0; @(posedge clk); // Hour 2 count = 1
137         incrHour <= 0; entrance_gate <= 0; exit_gate <= 1; @(posedge clk); //count = 0
138         incrHour <= 0; entrance_gate <= 1; exit_gate <= 0; @(posedge clk); //count = 1
139         incrHour <= 0; entrance_gate <= 1; exit_gate <= 0; @(posedge clk); //count = 2

```

```

140     incrHour <= 1; entrance_gate <= 0; exit_gate <= 0; @(posedge clk); // Hour 3 count = 2
141     incrHour <= 0; entrance_gate <= 1; exit_gate <= 0; @(posedge clk); //count = 3
142 //RUSH START
143     incrHour <= 0; entrance_gate <= 1; exit_gate <= 0; @(posedge clk); //count = 3
144     incrHour <= 0; entrance_gate <= 0; exit_gate <= 1; @(posedge clk); //count = 2
145     incrHour <= 1; entrance_gate <= 0; exit_gate <= 0; @(posedge clk); // Hour 4 count =
2
146     incrHour <= 0; entrance_gate <= 0; exit_gate <= 1; @(posedge clk); //count = 1
147     incrHour <= 0; entrance_gate <= 1; exit_gate <= 0; @(posedge clk); //count = 2
148
149     incrHour <= 1; entrance_gate <= 0; exit_gate <= 0; @(posedge clk); // Hour 5 count = 2
150     incrHour <= 0; entrance_gate <= 1; exit_gate <= 0; @(posedge clk); //count = 3
151     incrHour <= 0; entrance_gate <= 0; exit_gate <= 1; @(posedge clk); //count = 2
152
153     incrHour <= 1; entrance_gate <= 0; exit_gate <= 0; @(posedge clk); // Hour 6 count = 2
154     incrHour <= 0; entrance_gate <= 0; exit_gate <= 1; @(posedge clk); //count = 1
155     incrHour <= 0; entrance_gate <= 0; exit_gate <= 1; @(posedge clk); //count = 0
156 //RUSH ENDS
157     incrHour <= 0; entrance_gate <= 0; exit_gate <= 0; @(posedge clk); //count = 0
158
159     incrHour <= 1; entrance_gate <= 0; exit_gate <= 0; @(posedge clk); // Hour 7 count = 0
160     incrHour <= 0; entrance_gate <= 0; exit_gate <= 1; @(posedge clk); //count = 0
161     incrHour <= 0; entrance_gate <= 1; exit_gate <= 0; @(posedge clk); //count = 1
162     incrHour <= 0; entrance_gate <= 1; exit_gate <= 0; @(posedge clk); //count = 2
163
164     incrHour <= 1; entrance_gate <= 0; exit_gate <= 0; @(posedge clk); // Hour 8 count = 2
165     incrHour <= 0; entrance_gate <= 0; exit_gate <= 1; @(posedge clk); //count = 1
166     incrHour <= 0; entrance_gate <= 1; exit_gate <= 0; @(posedge clk); //count = 2
167     incrHour <= 0; entrance_gate <= 0; exit_gate <= 0; @(posedge clk);
168
169     reset <= 1;          @(posedge clk);
170     reset <= 0;          @(posedge clk);
171                          @(posedge clk);
172
173     $stop;
174 end
175 endmodule
176

```



## 7) clock\_divider

```
1  /*
2  Nithya Subramanian
3  March 11th 2025
4  EE 371
5  Lab 6A, Task 2 */
6
7  //a counter that divides a given clock input and outputs
8  //a divided_clocks output that is 32-bits.
9  module clock_divider(clock, divided_clocks);
10     input logic clock;
11     output logic [31:0] divided_clocks = 0;
12
13     always_ff @(posedge clock) begin
14         divided_clocks <= divided_clocks + 1;
15     end
16
17 endmodule
18
19 module clock_divider_testbench();
20     logic clock;
21     logic [31:0] divided_clocks;
22
23     `timescale 1 ps / 1 ps
24
25     clock_divider dut(.clock, .divided_clocks);
26
27     initial begin
28         @(posedge clock);
29         @(posedge clock);
30
31     end
32 endmodule
33
```

## 8) 8x16ram

```
1  /*
2  Nithya Subramanian
3  March 11th 2025
4  EE 371
5  Lab 6A, Task 2 */
6
7  // This module represents an 8x16-bit RAM (Random Access Memory)
8  // with 8 memory locations, each capable of storing 4-bit data.
9  // It supports both read and write operations based on the provided
10 // control signals. The RAM is clocked on the rising edge of the clock signal.
11 // - clock: The clock signal that triggers the read and write operations.
12 // - data: A 4-bit input data to be written into the memory.
13 // - rdaddress: A 3-bit address input for selecting the memory location
14 //   to be read from.
15 // - wraddress: A 3-bit address input for selecting the memory location
16 //   to be written to.
17 // - wren: A write enable signal that controls whether data should
18 //   be written to the memory.
19 //
20 // Outputs:
21 // - q: A 4-bit output representing the data read from the memory
22 //   at the specified read address.
23
24 module ram8x16 (
25     input logic clock,
26     input logic [3:0] data,
27     input logic [2:0] rdaddress,
28     input logic [2:0] wraddress,
29     input logic wren,
30     output logic [3:0] q
31 );
32     logic [3:0] mem [0:7];
33
34     always_ff @(posedge clock) begin
35         if (wren)
36             mem[wraddress] <= data;
37     end
38
39     assign q = mem[rdaddress];
40
41 endmodule
42
```