Nithya Subramanian
EE371
February 12, 2025
Lab 6 Report

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.

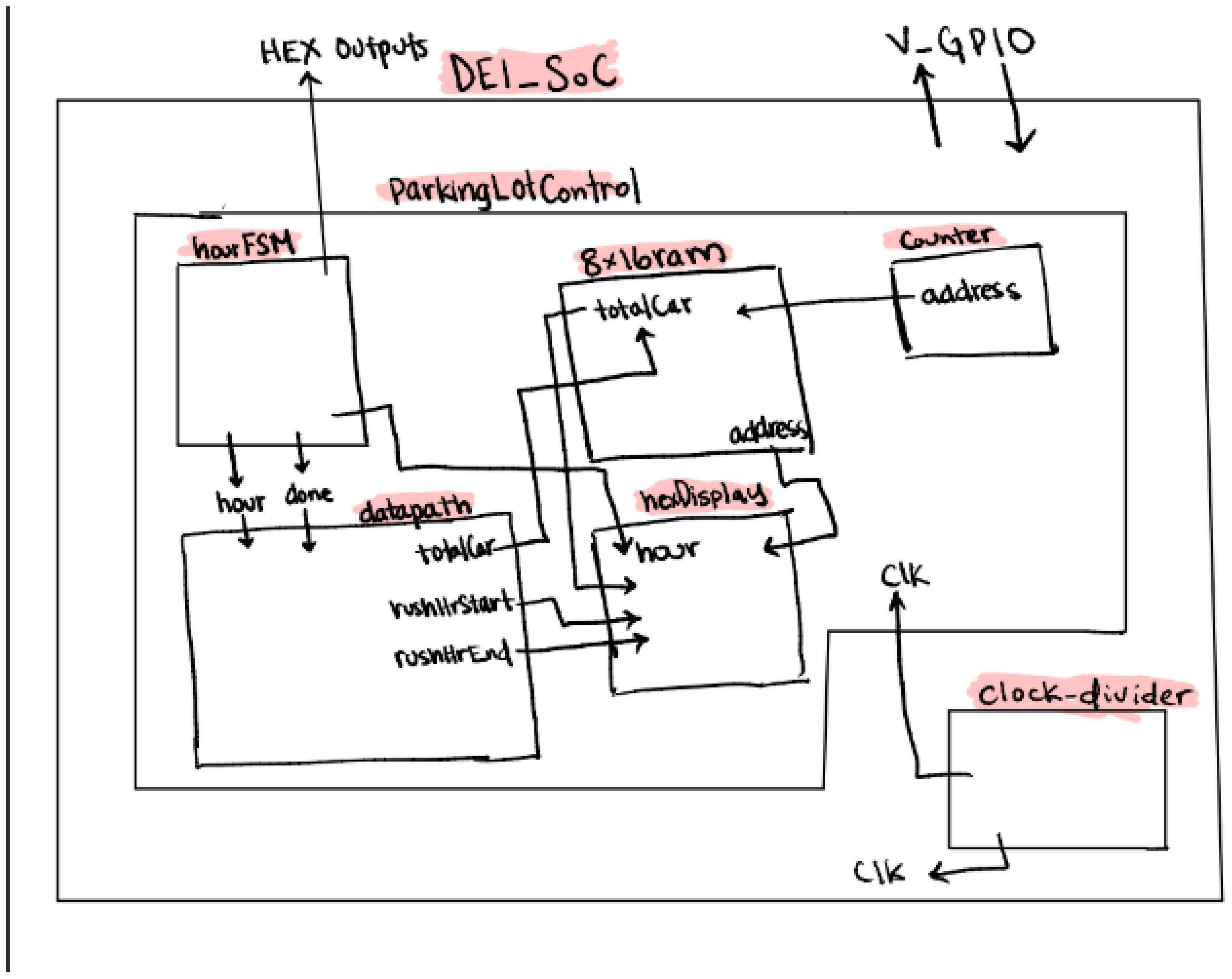

Figure 1: Block diagram of Task 2

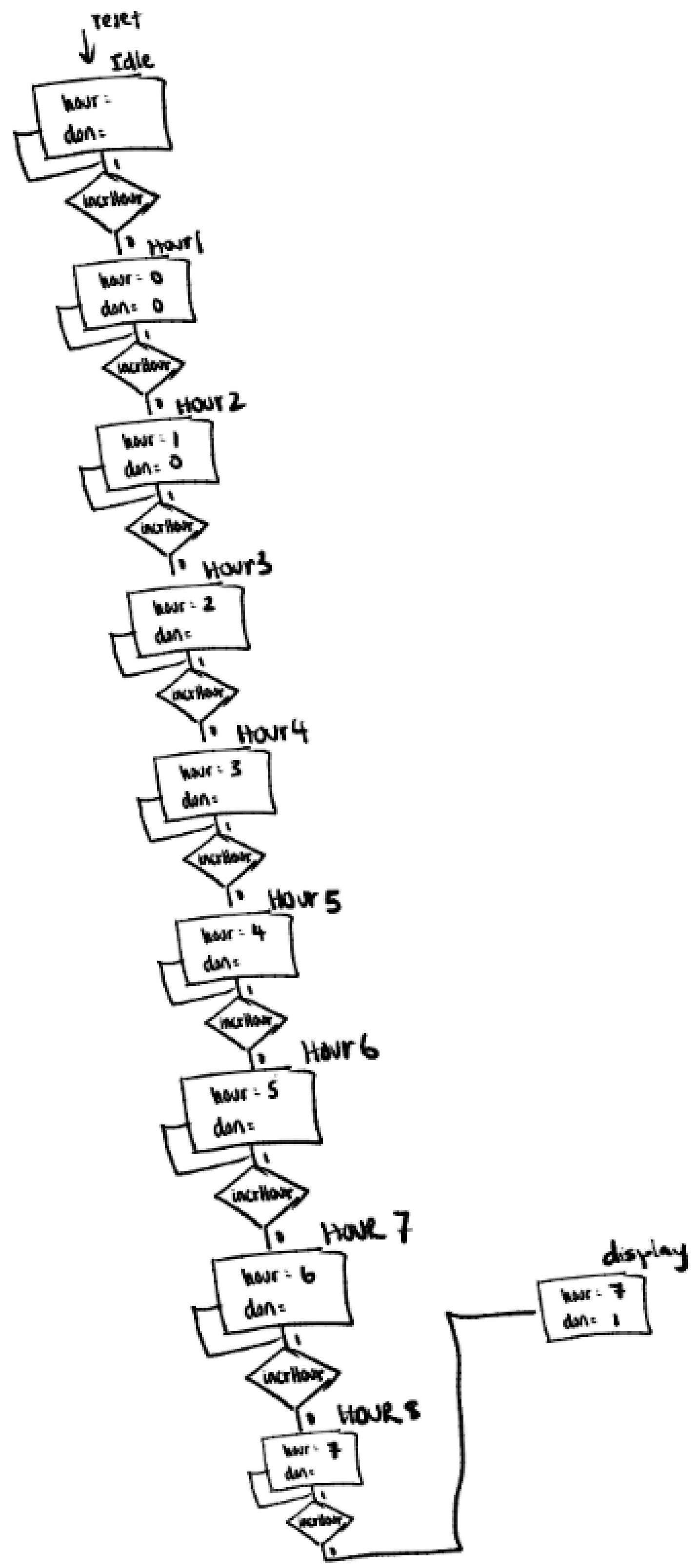

Figure 2: ASMD Diagram of Task 2

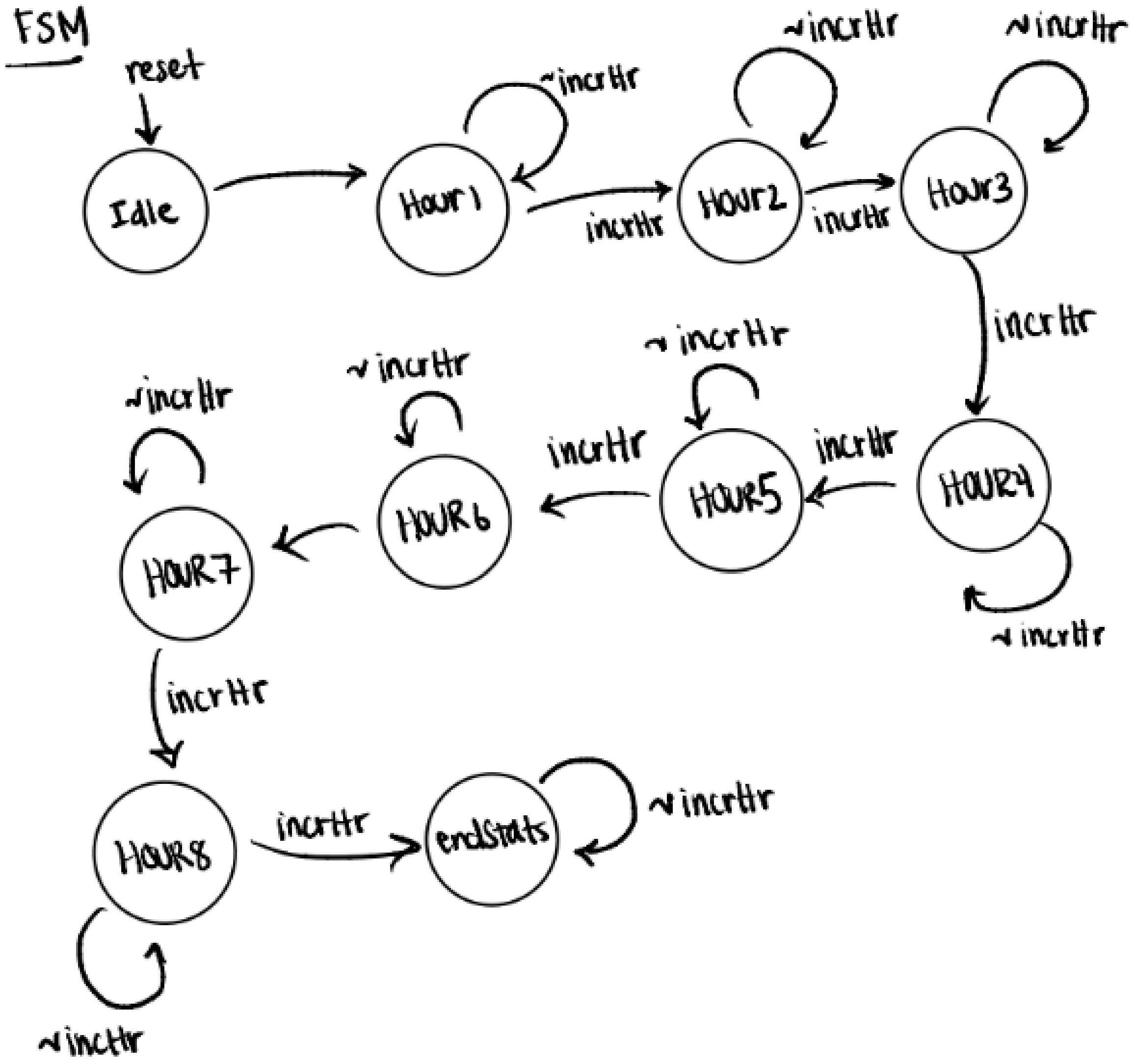

Figure 3: FSM diagram of Task 2

Section 2: Results

Task #2: Parking Lot Simulation

Each of the testbench simulations tested the following situations:

1. **DE1_SoC**: Top level module

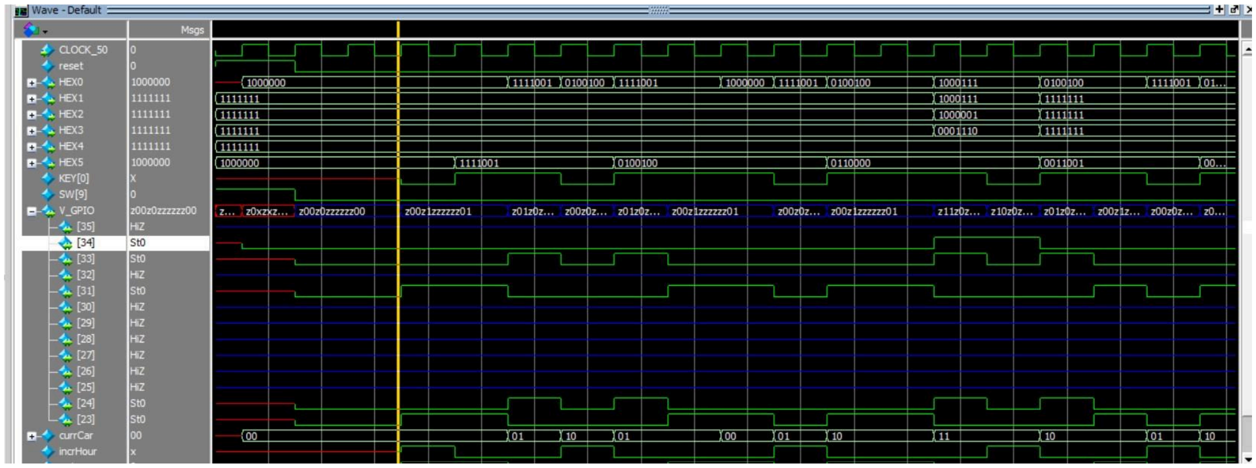

Figure 3.1: ModelSim waveform of DE1_SoC

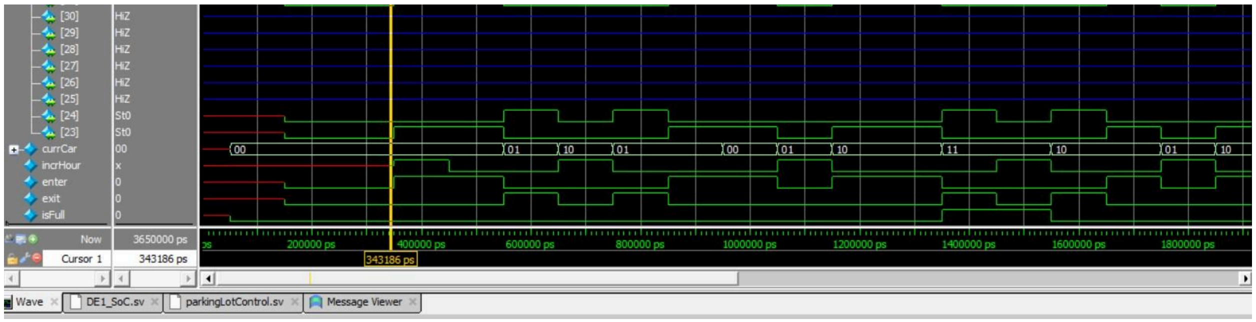

Figure 3.2: ModelSim waveform of DE1_SoC

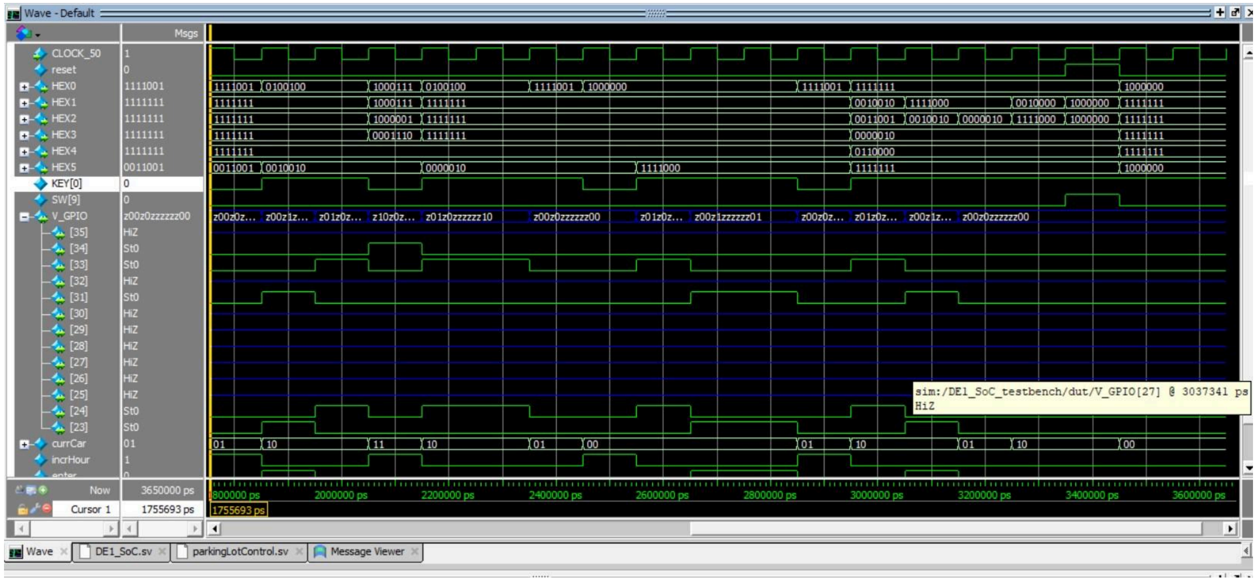

Figure 3.3: ModelSim waveform of DE1_SoC

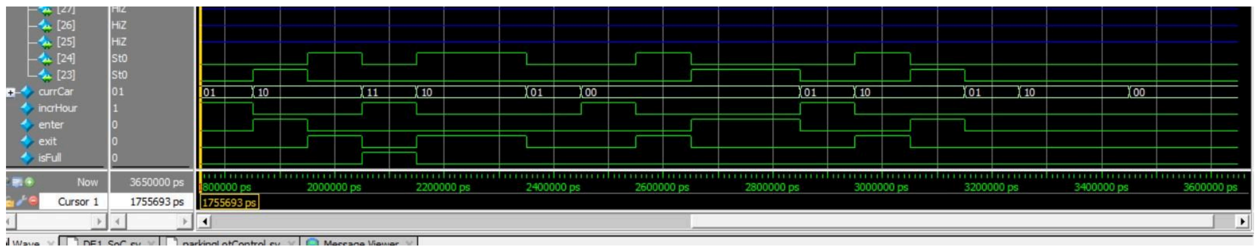

Figure 3.4: ModelSim waveform of DE1_SoC

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.

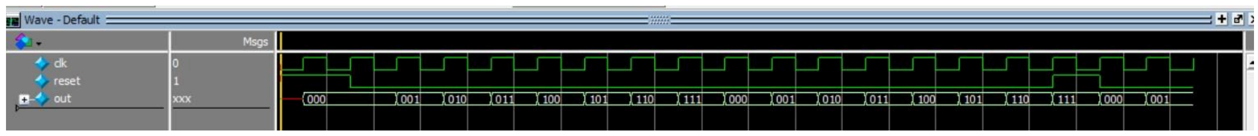

Figure 4: ModelSim waveform of counter

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.

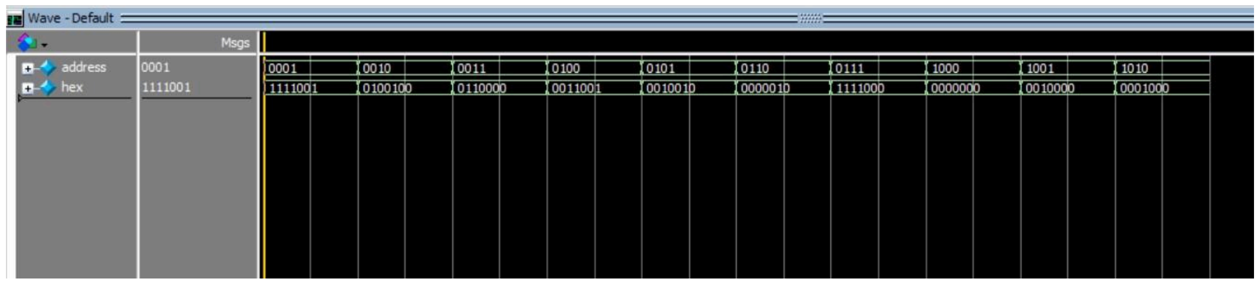

Figure 5: ModelSim waveform of hexDisplay

The hexDisplay talks in a 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 are 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.

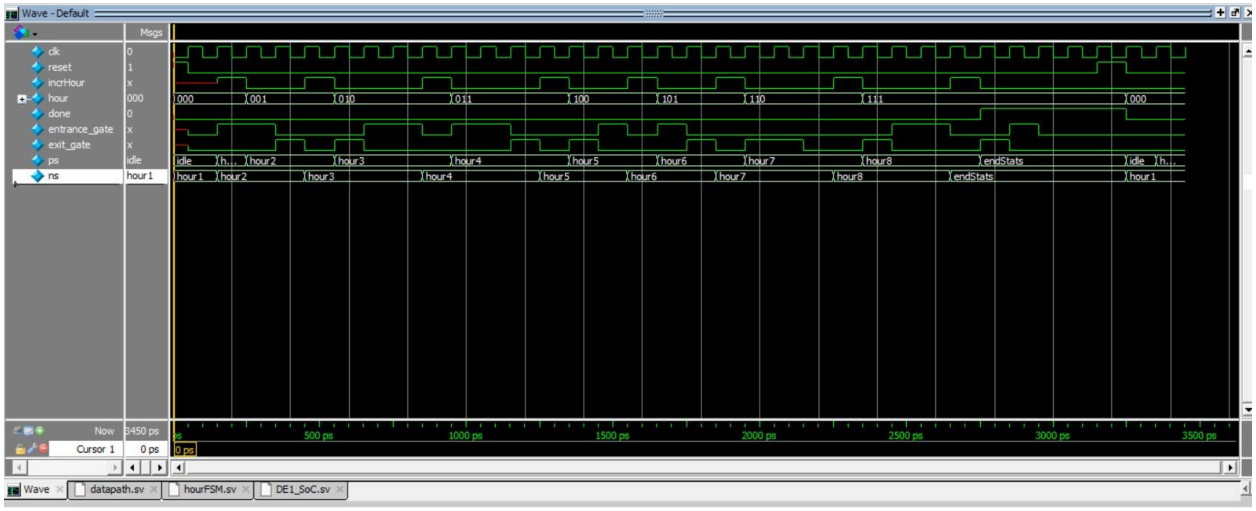

Figure 6: ModelSim waveform of hourFSM

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:**

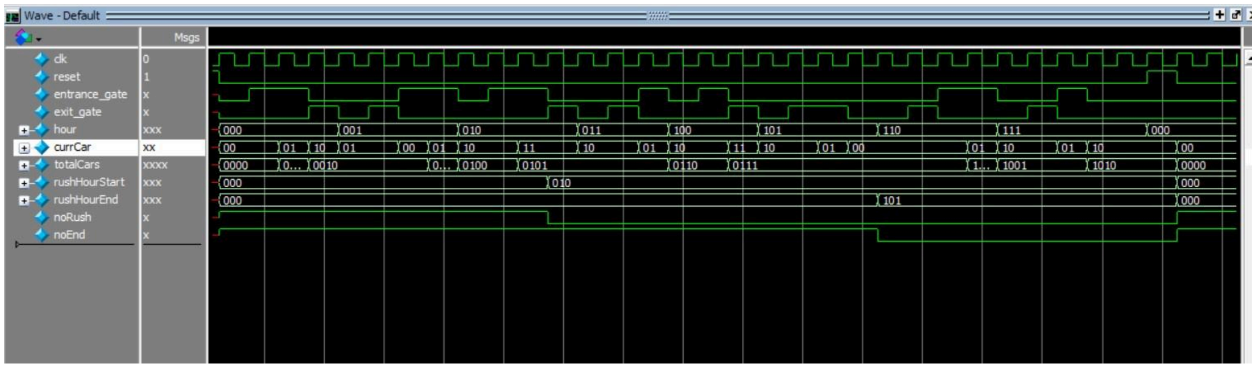

Figure 7: ModelSim waveform of datapath

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

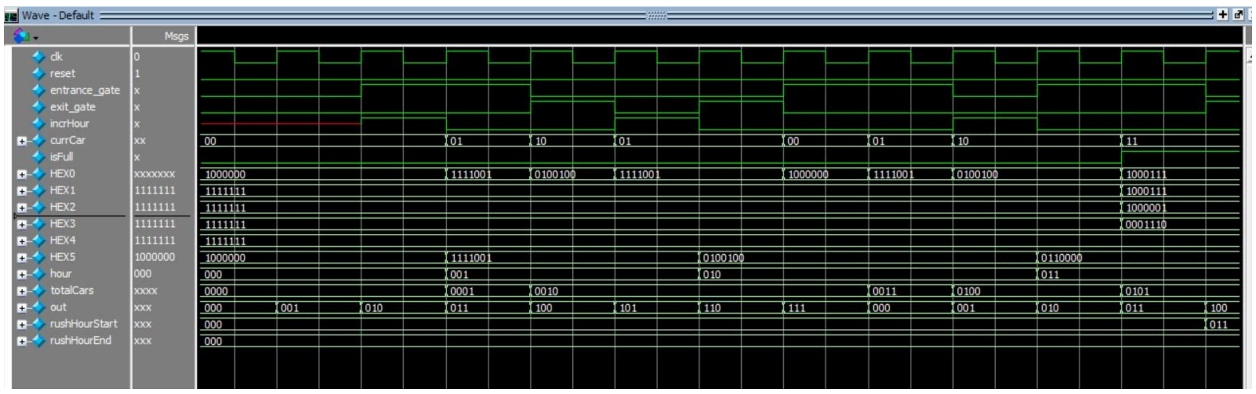

Figure 8.1: ModelSim waveform of parkingLotControl

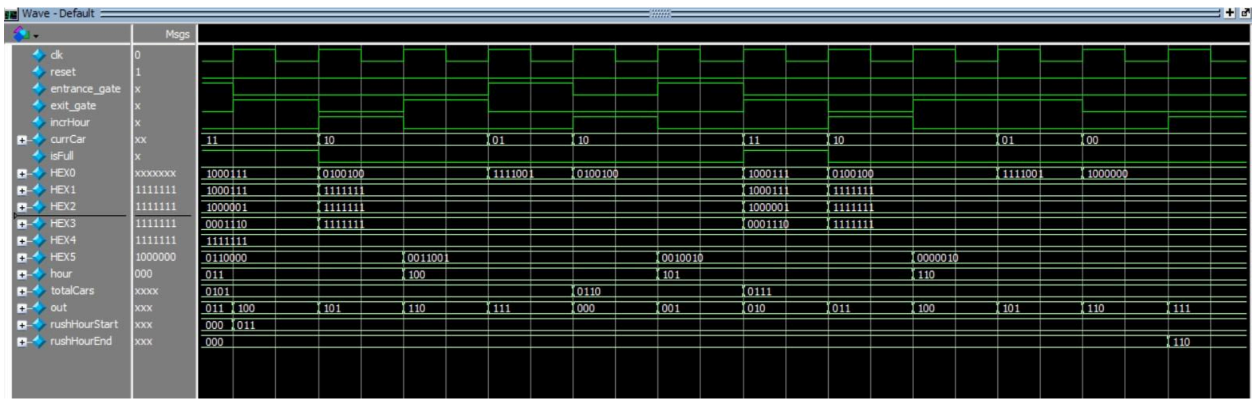

Figure 8.2: ModelSim waveform of parkingLotControl

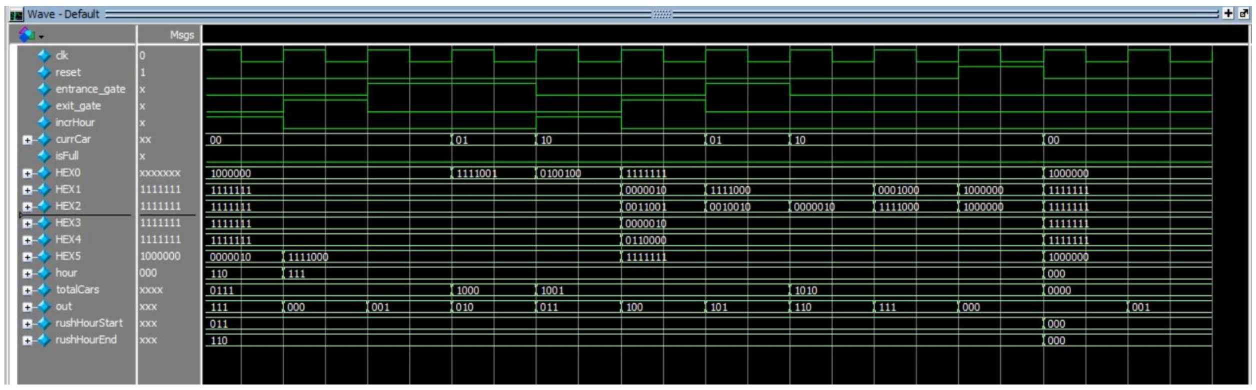

Figure 8.3: ModelSim waveform of parkingLotControl

The parkingLotControl testbench instanities 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 moudle 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 labsland 3D parking simulation. This product included getting more complex from our prerviosuly 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

```
/*
Nithya Subramanian
March 11th 2025
EE 371
Lab 6A, Task 2 */

/*top level entity of the project */
//DE1_SoC uses a 12 bit V_GPIO as inputs and returns a 7bit HEX0, HEX1, HEX2, HEX3, HEX4
//HEX5 and 10-bit LEDR as outputs.This display is driven as "0" and nothing as the enter
//or exit signal is high it increases or decreases respectivley and displayed on HEX0
//and HEX1. When the capacity of the car park is reached(3) HEX0-HEX3 display
//"FULL". This serves as the top-level module for the car sensor system
//implemented in this lab.
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 [1:0] currCar;
   logic reset;
   logic incrHour;
   logic enter, exit;

   //assign clk = CLOCK_50; //for simulation
   assign reset = SW[9];

   logic isFull;
   logic [31:0] div_clk;

   logic fullLight;

   // FPGA output
   assign V_GPIO[26] = V_GPIO[28];  // LED parking 1
   assign V_GPIO[27] = V_GPIO[29];  // LED parking 2
   assign V_GPIO[32] = V_GPIO[30];  // LED parking 3
   assign V_GPIO[34] = isFull;      //LED full
   assign V_GPIO[31] = V_GPIO[23];  // Open entrance
   assign V_GPIO[33] = V_GPIO[24];  // Open exit

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

   clock_divider clockDiv (.clock(CLOCK_50), .divided_clocks(div_clk));

   logic clk1, clk24;
   parameter whichClock = 26;

   assign clkSelect = CLOCK_50; //for simulation

   //assign clk1 = div_clk[1]; //for exiting
   //assign clk24 = div_clk[24]; //for entering

   always_ff @(posedge CLOCK_50) begin
      if (V_GPIO[23] && V_GPIO[31]) begin
         enter <= 1;
      end else
         enter <= 0;
   end

   always_ff @(posedge CLOCK_50) begin
      if (V_GPIO[24] | LEDR[4] | V_GPIO[33]) begin
         exit <= 1;
      end else
         exit <= 0;
   end

   parkingLotControl control (.clk(CLOCK_50), .reset, .entrance_gate(enter), .exit_gate(exit), .incrHour(~KEY[0]),
                              .HEX0(HEX0), .HEX1(HEX1), .HEX2(HEX2), .HEX3(HEX3), .HEX4(HEX4), .HEX5(HEX5), .currCar(currCar), .isFull(isFull));

endmodule  // DE1_SoC

`timescale 1ns / 1ps
//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 reset;
   logic [1:0] currCar;
   logic incrHour;
   logic isFull;
   logic enter, exit;


   assign KEY[0] = ~incrHour;
   assign SW[9] = reset;
   assign V_GPIO[23] = enter; // Presence entrance && enter
   assign V_GPIO[24] = exit;  // Presence exit && exit

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


   parameter CLOCK_PERIOD = 100;

   initial begin
      CLOCK_50 <= 0;
      forever #(CLOCK_PERIOD / 2) CLOCK_50 <= ~CLOCK_50;
   end

   initial begin
      reset <= 1;                                             @(posedge CLOCK_50);
                                                              @(posedge CLOCK_50);
      reset <= 0; enter <= 0; exit <= 0;                      @(posedge CLOCK_50);
                                                              @(posedge CLOCK_50);

      incrHour <= 1; enter <= 1; exit <= 0;  @(posedge CLOCK_50); // Hour 1, count = 1
      incrHour <= 0; enter <= 1; exit <= 0;  @(posedge CLOCK_50); //count = 2
      incrHour <= 0; enter <= 0; exit <= 1;  @(posedge CLOCK_50); //count = 1

      incrHour <= 1; enter <= 0; exit <= 0;  @(posedge CLOCK_50); // Hour 2 count = 1
      incrHour <= 0; enter <= 0; exit <= 1;  @(posedge CLOCK_50); //count = 0
      incrHour <= 0; enter <= 1; exit <= 0;  @(posedge CLOCK_50); //count = 1
      incrHour <= 0; enter <= 1; exit <= 0;  @(posedge CLOCK_50); //count = 2

      incrHour <= 1; enter <= 0; exit <= 0;  @(posedge CLOCK_50); // Hour 3 count = 2
      incrHour <= 0; enter <= 1; exit <= 0;  @(posedge CLOCK_50); //count = 3 //RUSH START
      incrHour <= 0; enter <= 1; exit <= 0;  @(posedge CLOCK_50); //count = 3
      incrHour <= 0; enter <= 0; exit <= 1;  @(posedge CLOCK_50); //count = 2

      incrHour <= 1; enter <= 0; exit <= 0;  @(posedge CLOCK_50); // Hour 4 count = 2
      incrHour <= 0; enter <= 0; exit <= 1;  @(posedge CLOCK_50); //count = 1
      incrHour <= 0; enter <= 1; exit <= 0;  @(posedge CLOCK_50); //count = 2

      incrHour <= 1; enter <= 0; exit <= 0;  @(posedge CLOCK_50); // Hour 5 count = 2
      incrHour <= 0; enter <= 1; exit <= 0;  @(posedge CLOCK_50); //count = 3
      incrHour <= 0; enter <= 0; exit <= 1;  @(posedge CLOCK_50); //count = 2

      incrHour <= 1; enter <= 0; exit <= 0;  @(posedge CLOCK_50); // Hour 6 count = 2
```

```verilog
        incrHour <= 0; enter <= 0; exit <= 1;  @(posedge CLOCK_50); //count = 1
        incrHour <= 0; enter <= 0; exit <= 1;  @(posedge CLOCK_50); //count = 0 //RUSH ENDS
        incrHour <= 0; enter <= 0; exit <= 0;  @(posedge CLOCK_50); //count = 0

        incrHour <= 1; enter <= 0; exit <= 0;  @(posedge CLOCK_50); // Hour 7 count = 0
        incrHour <= 0; enter <= 0; exit <= 1;  @(posedge CLOCK_50); //count = 0
        incrHour <= 0; enter <= 1; exit <= 0;  @(posedge CLOCK_50); //count = 1
        incrHour <= 0; enter <= 1; exit <= 0;  @(posedge CLOCK_50); //count = 2

        incrHour <= 1; enter <= 0; exit <= 0;  @(posedge CLOCK_50); // Hour 8 count = 2
        incrHour <= 0; enter <= 0; exit <= 1;  @(posedge CLOCK_50); //count = 1
        incrHour <= 0; enter <= 1; exit <= 0;  @(posedge CLOCK_50); //count = 2
        incrHour <= 0; enter <= 0; exit <= 0;  @(posedge CLOCK_50);
        incrHour <= 0; enter <= 0; exit <= 0;  @(posedge CLOCK_50);

        reset <= 1;                              @(posedge CLOCK_50);
        reset <= 0;                              @(posedge CLOCK_50);
                                                 @(posedge CLOCK_50);
        $stop();
    end
endmodule
```

2) Counter

```
/*
Nithya Subramanian
March 11th 2025
EE 371
Lab 6A, Task 2 */


//Counter takes in two 1-bit inputs called clk and reset and returns a 2-bit value called
//out. It cycles through the "hours" 0 - 7 and resets to 0 a either after the 7th hour
//or when reset signal is high
module counter (clk, reset, out);

   input logic clk, reset;
   output logic [2:0] out;

   always_ff @(posedge clk) begin
      if(reset) begin
         out <= 0;
      end else begin
         if(out < 7) begin
            out <= out + 1;
         end else begin
            out <= 0;
         end
      end
   end
endmodule

//testbench for counter tests all expected, unexpected and edgecase behaviors
module counter_testbench ();
   logic clk, reset;
   logic [2:0] out;

   counter dut (.clk(clk), .reset(reset), .out(out));

   parameter CLOCK_PERIOD = 100;

   initial begin
      clk <= 0;
      forever #(CLOCK_PERIOD / 2) clk <= ~clk;
   end

   initial begin
      reset <= 1;             @(posedge clk);
                              @(posedge clk);
      reset <= 0;             @(posedge clk);
                              @(posedge clk);
                              @(posedge clk);                             @(posedge clk);
                              @(posedge clk);
                              @(posedge clk);
                              @(posedge clk);
                              @(posedge clk);
                              @(posedge clk);
                              @(posedge clk);
                              @(posedge clk);
                              @(posedge clk);
                              @(posedge clk);
                              @(posedge clk);
                              @(posedge clk);
      reset <= 1;             @(posedge clk);
      reset <= 0;             @(posedge clk);
                              @(posedge clk);
      $stop();
   end
endmodule
```

3) hourFSM

```
/*
Nithya Subramanian
March 11th 2025
EE 371
Lab 6A, Task 2 */

// a state machine that increases the 3-bit hour signal when the input signal of
//incrHour is high and goes into the display state when we have completed
//a full day. It also outputs a one-bit signal that tells us when the day
//is complete and takes in 1-bit clk, reset, incrHour and.
module hourFSM (clk, reset, entrance_gate, exit_gate, incrHour, hour, done);
   input logic clk, reset, incrHour;
   input logic entrance_gate, exit_gate;

   output logic [2:0] hour;
   output logic done;

   enum {idle, hour1, hour2, hour3, hour4, hour5, hour6, hour7, hour8, endStats} ps, ns;

   //the state logic for moving through each work hour
   //if incrHour is high it moves to next state else it stays in same state
   //the done signal is only high in the endStats state and hour increases
   always_comb begin
      case(ps)
         idle: begin
            hour = 0;
            done = 0;
            ns = hour1;
         end

         hour1: begin
            hour = 0;
            done = 0;

            if(incrHour)
               ns = hour2;
            else
               ns = hour1;
         end

         hour2: begin
            hour = 1;
            done = 0;

            if(incrHour)
               ns = hour3;
            else
               ns = hour2;
         end

         hour3: begin
            hour = 2;
            done = 0;

            if(incrHour)
               ns = hour4;
            else
               ns = hour3;
         end

         hour4: begin
            hour = 3;
            done = 0;

            if(incrHour)
               ns = hour5;
            else
               ns = hour4;
         end

         hour5: begin
            hour = 4;
            done = 0;

            if(incrHour)
               ns = hour6;
            else
               ns = hour5;
         end

         hour6: begin
            hour = 5;
            done = 0;

            if(incrHour)
               ns = hour7;
            else
               ns = hour6;
         end

         hour7: begin
            hour = 6;
            done = 0;

            if(incrHour)
               ns = hour8;
            else
               ns = hour7;
         end

         hour8: begin
            hour = 7;
            done = 0;

            if(incrHour)
               ns = endStats;
            else
               ns = hour8;
         end

         endStats: begin
            hour = 7;
            done = 1;

            if(incrHour) begin
               hour = 0;
               ns = hour1;
            end
            else
               ns = endStats;
         end
      endcase
   end

   //if reset ps goes to idle state else it goes to ns
   always_ff @(posedge clk) begin
      if(reset) begin
         ps <= idle;
      end else begin
         ps <= ns;
      end
   end
endmodule

//testbench for hourFSM tests all expected, unexpected and edgecase behaviors
module hourFSM_testbench();
   logic clk, reset, incrHour;
   logic entrance_gate, exit_gate;
   logic [2:0] hour;
   logic done;

   hourFSM dut (.*);

   parameter CLOCK_PERIOD = 100;

   initial begin
```

```
        clk <= 0;

        forever #(CLOCK_PERIOD / 2) clk <= ~clk;
    end

    initial begin

        reset <= 1;                                                 @(posedge clk);
        reset <= 0; entrance_gate <= 0; exit_gate <= 0; hour <= 0;        @(posedge clk);
        incrHour <= 1; entrance_gate <= 1; exit_gate <= 0;  @(posedge clk); // Hour 1, count = 1
        incrHour <= 0; entrance_gate <= 1; exit_gate <= 0;  @(posedge clk); //count = 2
        incrHour <= 0; entrance_gate <= 0; exit_gate <= 1;  @(posedge clk); //count = 1

        incrHour <= 1; entrance_gate <= 0; exit_gate <= 0;  @(posedge clk); // Hour 2 count = 1
        incrHour <= 0; entrance_gate <= 0; exit_gate <= 1;  @(posedge clk); //count = 0
        incrHour <= 0; entrance_gate <= 1; exit_gate <= 0;  @(posedge clk); //count = 1
        incrHour <= 0; entrance_gate <= 1; exit_gate <= 0;  @(posedge clk); //count = 2

        incrHour <= 1; entrance_gate <= 0; exit_gate <= 0;  @(posedge clk); // Hour 3 count = 2
        incrHour <= 0; entrance_gate <= 1; exit_gate <= 0;  @(posedge clk); //count = 3
//RUSH START
        incrHour <= 0; entrance_gate <= 1; exit_gate <= 0;  @(posedge clk); //count = 3
        incrHour <= 0; entrance_gate <= 0; exit_gate <= 1;  @(posedge clk); //count = 2

        incrHour <= 1; entrance_gate <= 0; exit_gate <= 0;  @(posedge clk); // Hour 4 count = 2
        incrHour <= 0; entrance_gate <= 0; exit_gate <= 1;  @(posedge clk); //count = 1
        incrHour <= 0; entrance_gate <= 1; exit_gate <= 0;  @(posedge clk); //count = 2

        incrHour <= 1; entrance_gate <= 0; exit_gate <= 0;  @(posedge clk); // Hour 5 count = 2
        incrHour <= 0; entrance_gate <= 1; exit_gate <= 0;  @(posedge clk); //count = 3
        incrHour <= 0; entrance_gate <= 0; exit_gate <= 1;  @(posedge clk); //count = 2

        incrHour <= 1; entrance_gate <= 0; exit_gate <= 0;  @(posedge clk); // Hour 6 count = 2
        incrHour <= 0; entrance_gate <= 0; exit_gate <= 1;  @(posedge clk); //count = 1
        incrHour <= 0; entrance_gate <= 0; exit_gate <= 1;  @(posedge clk); //count = 0
//RUSH ENDS
        incrHour <= 0; entrance_gate <= 0; exit_gate <= 0;  @(posedge clk); //count = 0

        incrHour <= 1; entrance_gate <= 0; exit_gate <= 0;  @(posedge clk); // Hour 7 count = 0
        incrHour <= 0; entrance_gate <= 0; exit_gate <= 1;  @(posedge clk); //count = 0
        incrHour <= 0; entrance_gate <= 1; exit_gate <= 0;  @(posedge clk); //count = 1
        incrHour <= 0; entrance_gate <= 1; exit_gate <= 0;  @(posedge clk); //count = 2

        incrHour <= 1; entrance_gate <= 0; exit_gate <= 0;  @(posedge clk); // Hour 8 count = 2
        incrHour <= 0; entrance_gate <= 0; exit_gate <= 1;  @(posedge clk); //count = 1
        incrHour <= 0; entrance_gate <= 1; exit_gate <= 0;  @(posedge clk); //count = 2
        incrHour <= 0; entrance_gate <= 0; exit_gate <= 0;  @(posedge clk);
        incrHour <= 0; entrance_gate <= 0; exit_gate <= 0;  @(posedge clk);

        reset <= 1;                                 @(posedge clk);
        reset <= 0;                                 @(posedge clk);
                                                    @(posedge clk);

      $stop;
    end
endmodule
```

4) hexDisplay

```
/*
Nithya Subramanian
March 11th 2025
EE 371
Lab 6A, Task 2 */

//Takes in the address with a 4-bit input signal and outputs a converted
//7-seg 7-bit signal for the number that is inputed.
module hexDisplay (address, hex);
   input  logic [3:0] address;
   output logic [6:0] hex;

   always_comb begin
      case (address)

         0: hex = 7'b1000000; //0
         1: hex = 7'b1111001; //1
         2: hex = 7'b0100100; //2
         3: hex = 7'b0110000; //3
         4: hex = 7'b0011001; //4
         5: hex = 7'b0010010; //5
         6: hex = 7'b0000010; //6
         7: hex = 7'b1111000; //7
         8: hex = 7'b0000000; //8
         9: hex = 7'b0010000; //9
         10: hex = 7'b0001000; //A
         11: hex = 7'b0000011; //b
         12: hex = 7'b1000110; //C
         13: hex = 7'b0100001; //d
         14: hex = 7'b0000110; //E
         15: hex = 7'b0001110; //F
      endcase
   end
endmodule

// hexDisplay_testbench simulates all scenarios
module hexDisplay_testbench();
   logic [3:0] address;
   logic [6:0] hex;


   hexDisplay dut(.address, .hex);

   initial begin
      address = 4'b0001;   #10; //1
      address = 4'b0010;   #10; //2
      address = 4'b0011;   #10; //3
      address = 4'b0100;   #10; //4
      address = 4'b0101;   #10; //5
      address = 4'b0110;   #10; //6
      address = 4'b0111;   #10; //7
      address = 4'b1000;   #10; //8
      address = 4'b1001;   #10; //9
      address = 4'b1010;   #10; //10
      $stop();
   end

endmodule

```

5) Datapath

```
/*
Nithya Subramanian
March 11th 2025
EE 371
Lab 6A, Task 2 */

// This module handles the flow of car data within the system. It
// includes the logic for updating and maintaining the parking
// lot count, managing signals related to entry and exit,
// and interacting with other components of the system. This module
//also outputs the rushHourStart and end hours
module datapath (clk, reset, entrance_gate, exit_gate, hour, currCar, noRush,
                 noEnd, totalCars, rushHourStart, rushHourEnd);

   input logic clk, reset, entrance_gate, exit_gate;
   input logic [2:0] hour;

   output logic [1:0] currCar;
   output logic [3:0] totalCars;
   output logic [2:0] rushHourStart, rushHourEnd;
   output logic noRush, noEnd;

   logic rush;

   assign noRush = ~rush;

   //adds one to the currCar count if there is a car at the enterance and there
   //is a spot open and it decreases the currCar count if there is a car at exit
   //and their was atleast one car already parked
   always_ff @(posedge clk) begin
      if(reset) begin
         currCar <= 0;
         totalCars <= 0;
      end else begin
         if(entrance_gate & (currCar < 3)) begin
            currCar <= currCar + 1;
            totalCars <= totalCars + 1;

         end

         if(exit_gate & (currCar > 0)) begin
            currCar <= currCar - 1;
         end
      end
   end

   //Controls the rush hour data by reseting values to 0 except for noEnd = 1
   //is high and when there is no previous rush and the total number of cars
   //is 3 then rush hour has started and if the total count of cars is 0 but
   //previously the rush hasn't ended you can say it ended now and that hour
   //is the end hour of rush.
   always_ff @(posedge clk) begin
      if(reset) begin
         rushHourStart <= 0;
         rushHourEnd <= 0;
         rush <= 0;
         noEnd <= 1;
      end

      else begin
         if((rush == 0) && (currCar == 3)) begin
            rush <= 1;
            rushHourStart <= hour;
         end

         if((rush == 1) && (noEnd == 1) && (currCar == 0)) begin
            noEnd <= 0;
            rushHourEnd <= hour;
         end
      end
   end
endmodule //datapath
```

```
//testbench for datapath tests all expected, unexpected and edgecase behaviors
module datapath_testbench();
   logic clk, reset, entrance_gate, exit_gate;
   logic [2:0] hour;

   logic [1:0] currCar;
   logic [3:0] totalCars;
   logic [2:0] rushHourStart, rushHourEnd;
   logic noRush, noEnd;

   logic rush;

   datapath dut (.*);

   parameter CLOCK_PERIOD = 100;

   initial begin
      clk <= 0;

      forever #(CLOCK_PERIOD / 2) clk <= ~clk;
   end

   initial begin
      reset <= 1;                                                      @(posedge clk);
      reset <= 0; entrance_gate <= 0; exit_gate <= 0; hour <= 0;       @(posedge clk);
      hour <= 0; entrance_gate <= 1; exit_gate <= 0;  @(posedge clk); // Hour 1, count = 1
      hour <= 0; entrance_gate <= 1; exit_gate <= 0;  @(posedge clk); //count = 2
      hour <= 0; entrance_gate <= 0; exit_gate <= 1;  @(posedge clk); //count = 1

      hour <= 1; entrance_gate <= 0; exit_gate <= 0;  @(posedge clk); // Hour 2 count = 1
      hour <= 1; entrance_gate <= 0; exit_gate <= 1;  @(posedge clk); //count = 0
      hour <= 1; entrance_gate <= 1; exit_gate <= 0;  @(posedge clk); //count = 1
      hour <= 1; entrance_gate <= 1; exit_gate <= 0;  @(posedge clk); //count = 2

      hour <= 2; entrance_gate <= 0; exit_gate <= 0;  @(posedge clk); // Hour 3 count = 2
      hour <= 2; entrance_gate <= 1; exit_gate <= 0;  @(posedge clk); //count = 3 //RUSH
START
      hour <= 2; entrance_gate <= 1; exit_gate <= 0;  @(posedge clk); //count = 3
      hour <= 2; entrance_gate <= 0; exit_gate <= 1;  @(posedge clk); //count = 2

      hour <= 3; entrance_gate <= 0; exit_gate <= 0;  @(posedge clk); // Hour 4 count = 2
      hour <= 3; entrance_gate <= 0; exit_gate <= 1;  @(posedge clk); //count = 1
      hour <= 3; entrance_gate <= 1; exit_gate <= 0;  @(posedge clk); //count = 2

      hour <= 4; entrance_gate <= 0; exit_gate <= 0;  @(posedge clk); // Hour 5 count = 2
      hour <= 4; entrance_gate <= 1; exit_gate <= 0;  @(posedge clk); //count = 3
      hour <= 4; entrance_gate <= 0; exit_gate <= 1;  @(posedge clk); //count = 2

      hour <= 5; entrance_gate <= 0; exit_gate <= 0;  @(posedge clk); // Hour 6 count = 2
      hour <= 5; entrance_gate <= 0; exit_gate <= 1;  @(posedge clk); //count = 1
      hour <= 5; entrance_gate <= 0; exit_gate <= 1;  @(posedge clk); //count = 0 //RUSH ENDS
      hour <= 5; entrance_gate <= 0; exit_gate <= 0;  @(posedge clk); //count = 0

      hour <= 6; entrance_gate <= 0; exit_gate <= 0;  @(posedge clk); // Hour 7 count = 0
      hour <= 6; entrance_gate <= 0; exit_gate <= 1;  @(posedge clk); //count = 0
      hour <= 6; entrance_gate <= 1; exit_gate <= 0;  @(posedge clk); //count = 1
      hour <= 6; entrance_gate <= 1; exit_gate <= 0;  @(posedge clk); //count = 2

      hour <= 7; entrance_gate <= 0; exit_gate <= 0;  @(posedge clk); // Hour 8 count = 2
      hour <= 7; entrance_gate <= 0; exit_gate <= 1;  @(posedge clk); //count = 1
      hour <= 7; entrance_gate <= 1; exit_gate <= 0;  @(posedge clk); //count = 2
      hour <= 7; entrance_gate <= 0; exit_gate <= 0;  @(posedge clk);
      hour <= 7; entrance_gate <= 0; exit_gate <= 0;  @(posedge clk);

      reset <= 1; hour <= 0;                   @(posedge clk);
      reset <= 0;                              @(posedge clk);
                                               @(posedge clk);

      $stop;
   end
endmodule
```

6) parkingLotControl

```
/*
Nithya Subramanian
March 11th 2025
EE 371
Lab 6A, Task 2 */
// This module manages vehicle entry and exit using sensors. It
// keeps track of the number of cars in the lot, ensures the count
// does not exceed capacity, and controls entry/exit signals.
// The counter increments when a car enters and decrements when a
// car exits, ensuring real-time monitoring of availability.

`timescale 1ns / 1ps

module parkingLotControl (clk, reset, entrance_gate, exit_gate, incrHour,
                          HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, currCar, isFull);

   input logic clk, reset, entrance_gate, exit_gate, incrHour;

   output logic [1:0] currCar;
   output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
   output logic isFull;

   logic [2:0] hour;
   logic [3:0] totalCars;
   logic [2:0] rushHourStart, rushHourEnd;
   logic noRush, noEnd;
   logic done;

   logic [6:0] displayCars, displayTime, displayCurr, displayAddr, disRushStart, disRushEnd;
   logic [3:0] storedCarValue;
   logic [2:0] out;

   logic [31:0] div_clk;

   clock_divider divclk (.clock(clk), .divided_clocks(div_clk));

   logic clkSelect;
   parameter whichClock = 25;

   assign clkSelect = clk;
   //assign clkSelect = div_clk[whichClock];

   assign isFull = (currCar == 3);

   hourFSM fsm(.clk(clkSelect), .reset, .entrance_gate, .exit_gate, .incrHour, .hour, .done);

   datapath carData(.clk(clkSelect), .reset, .entrance_gate, .exit_gate, .hour, .currCar,
                    .noRush, .noEnd, .totalCars, .rushHourStart, .rushHourEnd);

   counter count(.clk(clkSelect), .reset, .out);

   ram8x16 ram(.clock(clkSelect), .data(totalCars), .rdaddress(out), .wraddress(hour), .wren
(1'b1), .q(storedCarValue));

   //Display assignements for different types of data
   hexDisplay dispTime (.address({1'b0, hour}), .hex(displayTime)); //shows current hour
   hexDisplay dispCars (.address({2'b0, currCar}), .hex(displayCurr)); //shows current car
count
   hexDisplay dispRushStart (.address({1'b0, rushHourStart}), .hex(disRushStart)); //shows
rush's start hour
   hexDisplay dispRushEnd (.address({1'b0, rushHourEnd}), .hex(disRushEnd)); //shows rush's
end hour
   hexDisplay dispOut (.address({1'b0, out}), .hex(displayAddr)); //shows address when
looping through
   hexDisplay dispStoredCars (.address(storedCarValue), .hex(displayCars)); //shows # of
cars in RAM

   //combinatinoal logic that helps decide what should be displayed on each HEX
   always_comb begin
      if(!done) begin
         if(currCar == 3) begin
            HEX5 = displayTime;
            HEX4 = 7'b1111111;
```

```
                HEX3 = 7'b0001110; //F
                HEX2 = 7'b1000001; //U
                HEX1 = 7'b1000111; //L
                HEX0 = 7'b1000111; //L
             end

             else begin
                HEX5 = displayTime;
                HEX4 = 7'b1111111;
                HEX3 = 7'b1111111;
                HEX2 = 7'b1111111;
                HEX1 = 7'b1111111;
                HEX0 = displayCurr;
             end
          end

          else begin
             HEX5 = 7'b1111111;
             HEX4 = (noRush) ? 7'b0111111 : disRushStart;
             HEX3 = (noEnd) ? 7'b0111111 : disRushEnd;
             HEX2 = displayAddr;
             HEX1 = displayCars;
             HEX0 = 7'b1111111;
          end
       end
    endmodule


    //parkingLotControl testbench that simulates all scenarios.
    module parkingLotControl_testbench();
       logic clk, reset, entrance_gate, exit_gate, incrHour;

       logic [1:0] currCar;
       logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
       logic isFull;

       logic [2:0] hour;
       logic [3:0] totalCars;
       logic [2:0] rushHourStart, rushHourEnd;
       logic noRush, noEnd;
       logic done;

       logic rush;

       logic [6:0] displayCars, displayTime, displayCurr, displayAddr, disRushStart, disRushEnd;
       logic [3:0] storedCarValue;
       logic [2:0] out;

       parkingLotControl dut (.*);

       parameter CLOCK_PERIOD = 100;

       initial begin
          clk <= 0;
          forever #(CLOCK_PERIOD / 2) clk <= ~clk;
       end

       initial begin
          reset <= 1;                                 @(posedge clk);
                                                      @(posedge clk);
          reset <= 0; entrance_gate <= 0; exit_gate <= 0; @(posedge clk);
                                                          @(posedge clk);

          incrHour <= 1; entrance_gate <= 1; exit_gate <= 0;  @(posedge clk); // Hour 1, count
    = 1
          incrHour <= 0; entrance_gate <= 1; exit_gate <= 0;  @(posedge clk); //count = 2
          incrHour <= 0; entrance_gate <= 0; exit_gate <= 1;  @(posedge clk); //count = 1

          incrHour <= 1; entrance_gate <= 0; exit_gate <= 0;  @(posedge clk); // Hour 2 count = 1
          incrHour <= 0; entrance_gate <= 0; exit_gate <= 1;  @(posedge clk); //count = 0
          incrHour <= 0; entrance_gate <= 1; exit_gate <= 0;  @(posedge clk); //count = 1
          incrHour <= 0; entrance_gate <= 1; exit_gate <= 0;  @(posedge clk); //count = 2

```

```
        incrHour <= 1; entrance_gate <= 0; exit_gate <= 0;  @(posedge clk); // Hour 3 count = 2
        incrHour <= 0; entrance_gate <= 1; exit_gate <= 0;  @(posedge clk); //count = 3
//RUSH START
        incrHour <= 0; entrance_gate <= 1; exit_gate <= 0;  @(posedge clk); //count = 3
        incrHour <= 0; entrance_gate <= 0; exit_gate <= 1;  @(posedge clk); //count = 2

        incrHour <= 1; entrance_gate <= 0; exit_gate <= 0;  @(posedge clk); // Hour 4 count = 2
        incrHour <= 0; entrance_gate <= 0; exit_gate <= 1;  @(posedge clk); //count = 1
        incrHour <= 0; entrance_gate <= 1; exit_gate <= 0;  @(posedge clk); //count = 2

        incrHour <= 1; entrance_gate <= 0; exit_gate <= 0;  @(posedge clk); // Hour 5 count = 2
        incrHour <= 0; entrance_gate <= 1; exit_gate <= 0;  @(posedge clk); //count = 3
        incrHour <= 0; entrance_gate <= 0; exit_gate <= 1;  @(posedge clk); //count = 2

        incrHour <= 1; entrance_gate <= 0; exit_gate <= 0;  @(posedge clk); // Hour 6 count = 2
        incrHour <= 0; entrance_gate <= 0; exit_gate <= 1;  @(posedge clk); //count = 1
        incrHour <= 0; entrance_gate <= 0; exit_gate <= 1;  @(posedge clk); //count = 0
//RUSH ENDS
        incrHour <= 0; entrance_gate <= 0; exit_gate <= 0;  @(posedge clk); //count = 0

        incrHour <= 1; entrance_gate <= 0; exit_gate <= 0;  @(posedge clk); // Hour 7 count = 0
        incrHour <= 0; entrance_gate <= 0; exit_gate <= 1;  @(posedge clk); //count = 0
        incrHour <= 0; entrance_gate <= 1; exit_gate <= 0;  @(posedge clk); //count = 1
        incrHour <= 0; entrance_gate <= 1; exit_gate <= 0;  @(posedge clk); //count = 2

        incrHour <= 1; entrance_gate <= 0; exit_gate <= 0;  @(posedge clk); // Hour 8 count = 2
        incrHour <= 0; entrance_gate <= 0; exit_gate <= 1;  @(posedge clk); //count = 1
        incrHour <= 0; entrance_gate <= 1; exit_gate <= 0;  @(posedge clk); //count = 2
        incrHour <= 0; entrance_gate <= 0; exit_gate <= 0;  @(posedge clk);
        incrHour <= 0; entrance_gate <= 0; exit_gate <= 0;  @(posedge clk);

        reset <= 1;                               @(posedge clk);
        reset <= 0;                               @(posedge clk);
                                                  @(posedge clk);

        $stop;
    end
endmodule
```

7) clock_divider

```
/*
Nithya Subramanian
March 11th 2025
EE 371
Lab 6A, Task 2 */

//a counter that divides a given clock input and outputs
//a divided_clocks output that is 32-bits.
module clock_divider(clock, divided_clocks);
   input logic clock;
   output logic [31:0] divided_clocks = 0;

   always_ff @(posedge clock) begin
      divided_clocks <= divided_clocks + 1;
   end

endmodule

module clock_divider_testbench();
   logic clock;
   logic [31:0] divided_clocks;

   `timescale 1 ps / 1 ps

   clock_divider dut(.clock, .divided_clocks);

   initial begin
      @(posedge clock);
      @(posedge clock);

   end
endmodule

```

8) 8x16ram

```
/*
Nithya Subramanian
March 11th 2025
EE 371
Lab 6A, Task 2 */

// This module represents an 8x16-bit RAM (Random Access Memory)
// with 8 memory locations, each capable of storing 4-bit data.
// It supports both read and write operations based on the provided
// control signals. The RAM is clocked on the rising edge of the clock signal.
// - clock: The clock signal that triggers the read and write operations.
// - data: A 4-bit input data to be written into the memory.
// - rdaddress: A 3-bit address input for selecting the memory location
//   to be read from.
// - wraddress: A 3-bit address input for selecting the memory location
//   to be written to.
// - wren: A write enable signal that controls whether data should
//   be written to the memory.
//
// Outputs:
// - q: A 4-bit output representing the data read from the memory
//   at the specified read address.

module ram8x16 (
    input logic clock,
    input logic [3:0] data,
    input logic [2:0] rdaddress,
    input logic [2:0] wraddress,
    input logic wren,
    output logic [3:0] q
);
    logic [3:0] mem [0:7];

    always_ff @(posedge clock) begin
        if (wren)
            mem[wraddress] <= data;
    end

    assign q = mem[rdaddress];

endmodule
```