Kiana Peterson and Nattapon Oonlmaom
EE 371
March 13, 2023
Lab 6 Report

Procedure

In this lab, the purpose was to reintroduce the parking lot occupancy from lab 1, but instead of using simple breadboard components, an actual, 3D parking lot, will be simulated while also integrating the concepts learned throughout the quarter.

Task #1: Parking Lot Breadboard *(Graded by Completion, so no simulation needed)*

The task was to take an FSM that monitors the activity of cars in a parking lot with a single entry and exit gate, implemented for lab 1, and demonstrate the functionality onto the new breadboard remote lab interface by changing GPIO_0 reference to V_GPIO in the top-level module of every line of code that contained GPIO_0, as well as wire it to different GPIO pin numbers. The more specific modifications are also documented in the code and can be seen below:

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

    input logic CLOCK_50;
    input logic [9:0] SW;
    input logic [3:0] KEY;
    output logic [6:0] HEX5, HEX4, HEX3, HEX2, HEX1, HEX0;
    output logic [9:0] LEDR;

    logic enter;
    logic exit;
    logic reset;
    logic [4:0] capacity;

    // Pinouts changed from GPIO_0 to V_GPIO. The pinouts are as follows:
    // Inputs:
    //     V_GPIO[24] = the left switch acting as sensor a
    //     V_GPIO[28] = the right switch acting as sensor b
    // Outputs:
    //     V_GPIO[32] = the left ledr corresponding to sensor a, given by the left switch
    //     V_GPIO[34] = the right ledr corresponding to sensor b, given by the right sitch
    inout logic [35:23] V_GPIO;
    assign V_GPIO[32] = V_GPIO[24]; // Wired the left LEDR (sensor a) to the left switch
    assign V_GPIO[34] = V_GPIO[28]; // Wired the right LEDR (sensor b) to the right switch
    assign reset = SW[9]; // Changed the reset to be attached to switch 9

    logic [31:0] clk;
    parameter whichClock = 20;

    // instantiates clock_divider to slow down clock
    clock_divider cdiv (.clock(CLOCK_50), .divided_clocks(clk));

    // instantiates parkingLot to track cars entering or exiting the lot with sensors
    // Inputs a and b changed to the correct pinouts
    parkingLot theParkingLot (.clk(clk[whichClock]), .reset(reset), .a(V_GPIO[24]), .b(V_GPIO[28]), .enter(enter), .exit(exit));

    // instantiates counter to keep track of how many cars are occupying the lot
    counter carCapacity (.clk(clk[whichClock]), .reset(reset), .inc(enter), .dec(exit), .out(capacity));

    // instantiate display to show how many cars are occupying the lot
    display carCount (.clk(clk[whichClock]), .count(capacity), .HEX5(HEX5), .HEX4(HEX4), .HEX3(HEX3), .HEX2(HEX2), .HEX1(HEX1), .HEX0(HEX0));

endmodule
```

Figure 1 Specific Changes Made in the Code for Task1

Task #2: Parking Lot 3D Simulation

This task was to implement the parking lot mechanism with an additional function into the 3D parking lot simulation provided by LabsLand. The parking lot has a single entry and exit with a maximum occupancy of three cars. One car can enter the parking lot, and one car can leave the parking lot at a time accordingly to the user controls. When fully occupied, The entering car will wait at the entrance/exit gate until the gate opens. When the lot is full, led shows red and green otherwise. Similarly, when the space is occupied, the ledr of the space shows red and green otherwise. Further, the occupancy of the parking lot will be displayed on HEX3-0 and “full” will be indicated when the maximum is reached. The additional function includes rush hour, where this parking lot is for a restaurant that has an eight-hour work day, wherein Pressing KEY[0] will increment the work day by an hour which will be displayed on HEX5. At the end of rush, the start and end of rush hour (when 3 cars occupied the lot and when the last car left) will be displayed on HEX4-3 while HEX2-1 cycle through the RAM storing memory of how many cars enter the parking lot according to the hour. The rough draft block diagram for this task can be seen below:

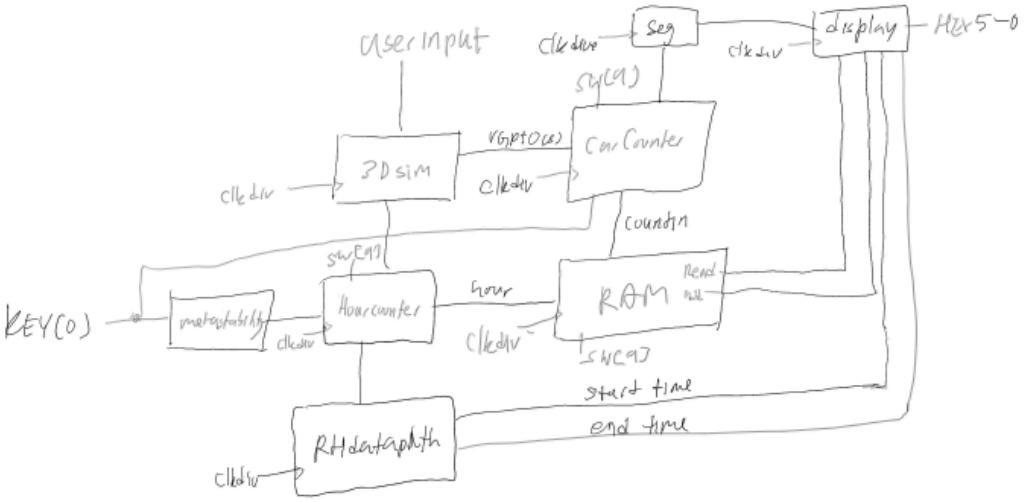


Figure 2.1 Block diagram of Task2

After struggling with many attempts in debugging, we implemented the system this way as is the most logical to solve the given problem. After considering how the components will be connected, we determined how each component will work using a state diagram. The ASMD and FSM of the system are shown below:

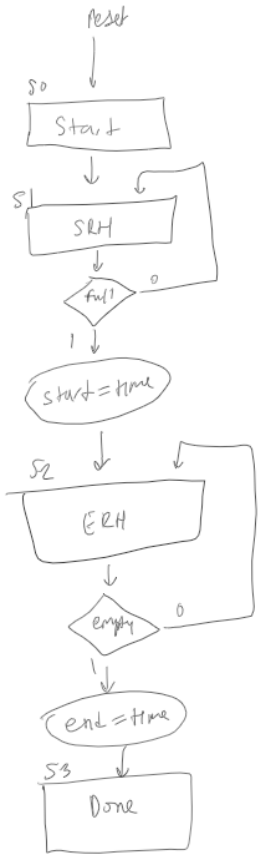


Figure 2.2 Rush Hour ASMD

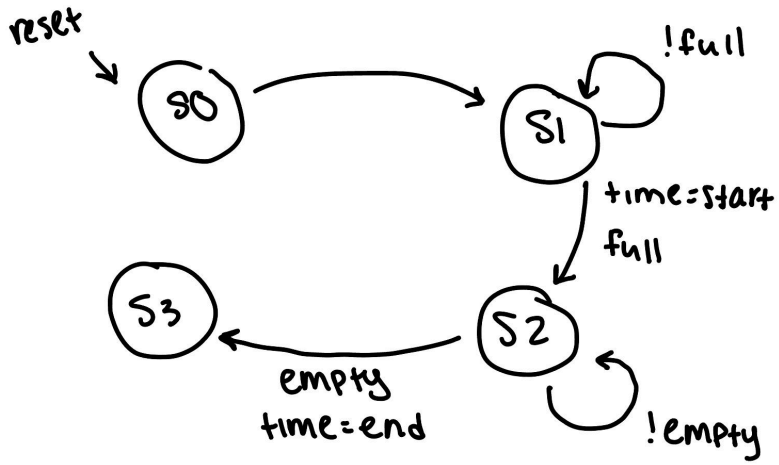


Figure 2.3 Rush Hour FSM

Results

Task #1: ***(Graded by Completion, so no simulation needed)***

Task #2:

\

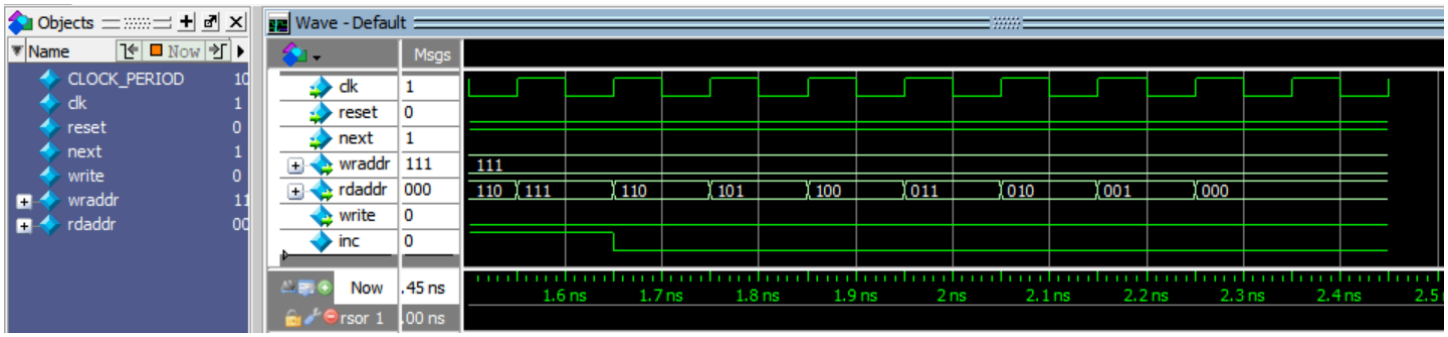


Figure 3.1 addresses Waveform

The addresses module worked as expected according to the waveform as it counts up by one up to 7 for the write address with the write enable as 1. When wraddress equals 7, rdaddress increments up to 7 and back down to 0 while write enable is 0.

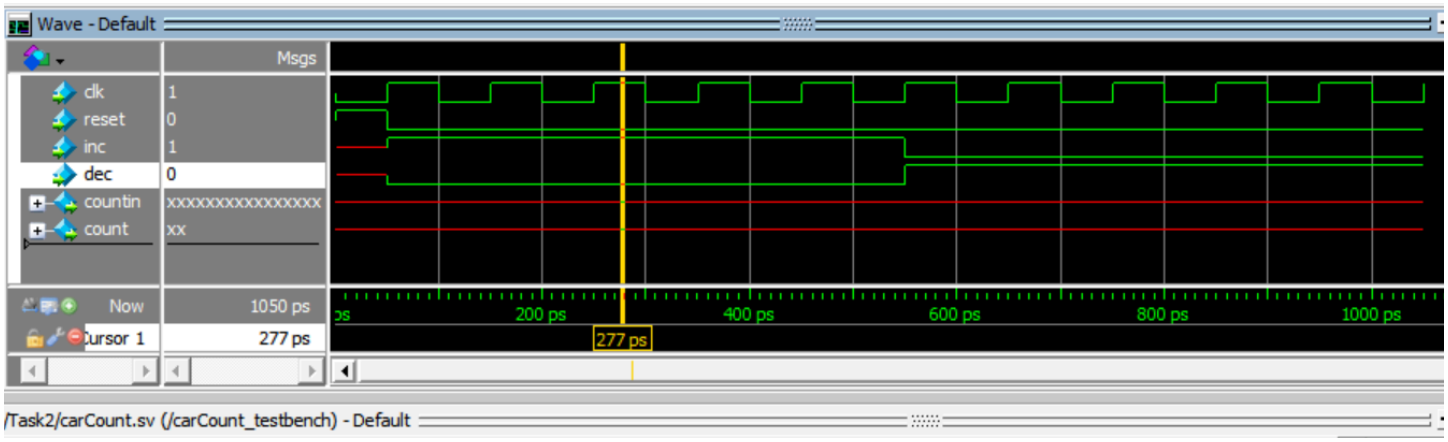


Figure 3.2 carCount Waveform

The carCount module worked as expected in the demo as it counts up by one up to 3 as incremented while counts down to 0 as it is decremented. It also gives the amount of cars that entered during the day. We changed variable names due to a lot of similar names and our testbench seems to have an issue with displaying the values

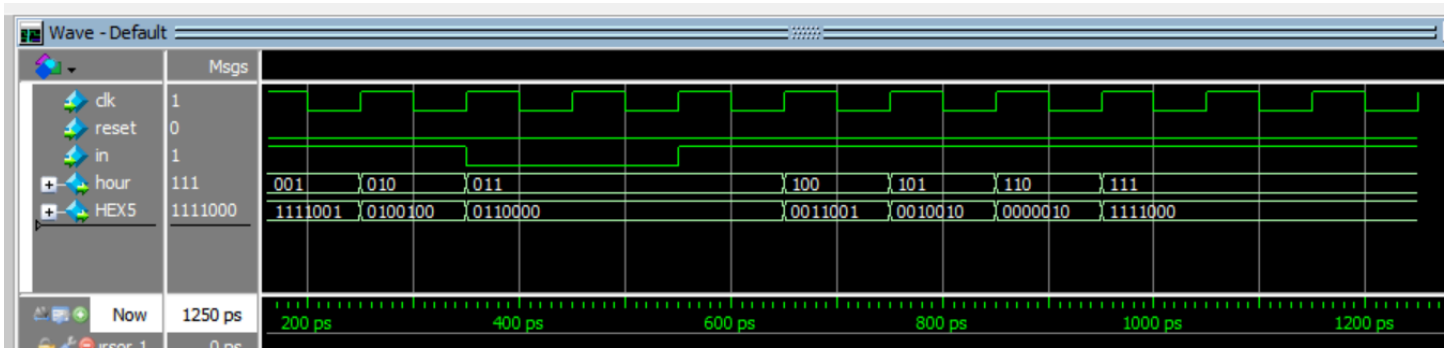


Figure 3.3 hourCounter Waveform

The hourCounter module worked as expected according to the waveform as it counts up by one up to 7 as incremented while counts down to 0 as it is decremented. It also gives the hex values for each hour

Figure 3.2 display Waveform

The display module is working as expected as it displays FULL25 when the parking lot is at its full occupancy with 25 cars, and CLEAR0 when there is zero car occupied. The displayed number on HEX0 is also incremented as occupancy changes.

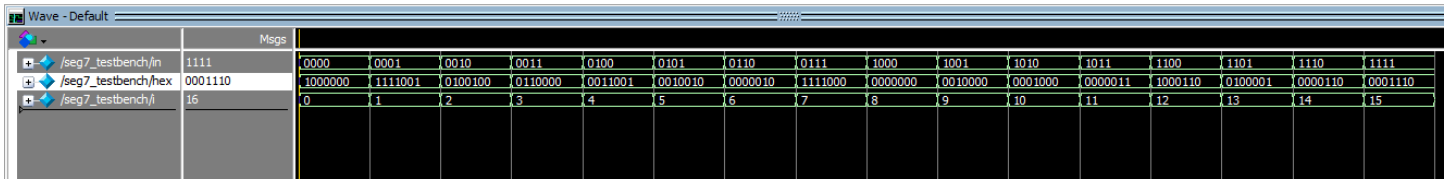


Figure 3.3 seg7.sv module Waveform

The seg7 module functions correctly as expected. The HEX display shows numbers correctly according to their assigned binary.

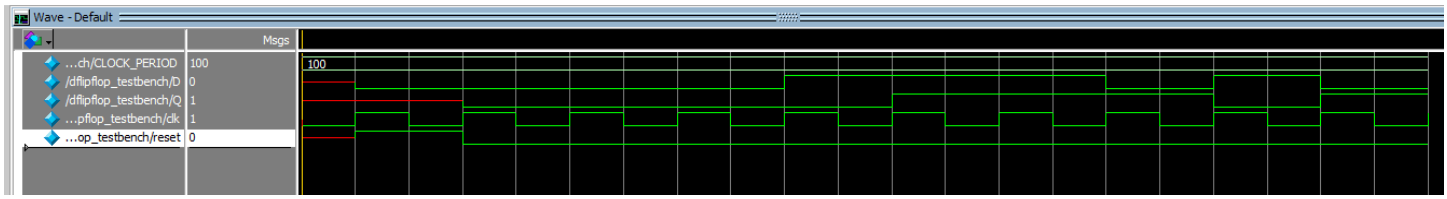


Figure 3.3 metastability.sv module Waveform

The metastability module works as intended. It takes in an input, set the output to 0 when reset, and return output delay by a clock cycle.

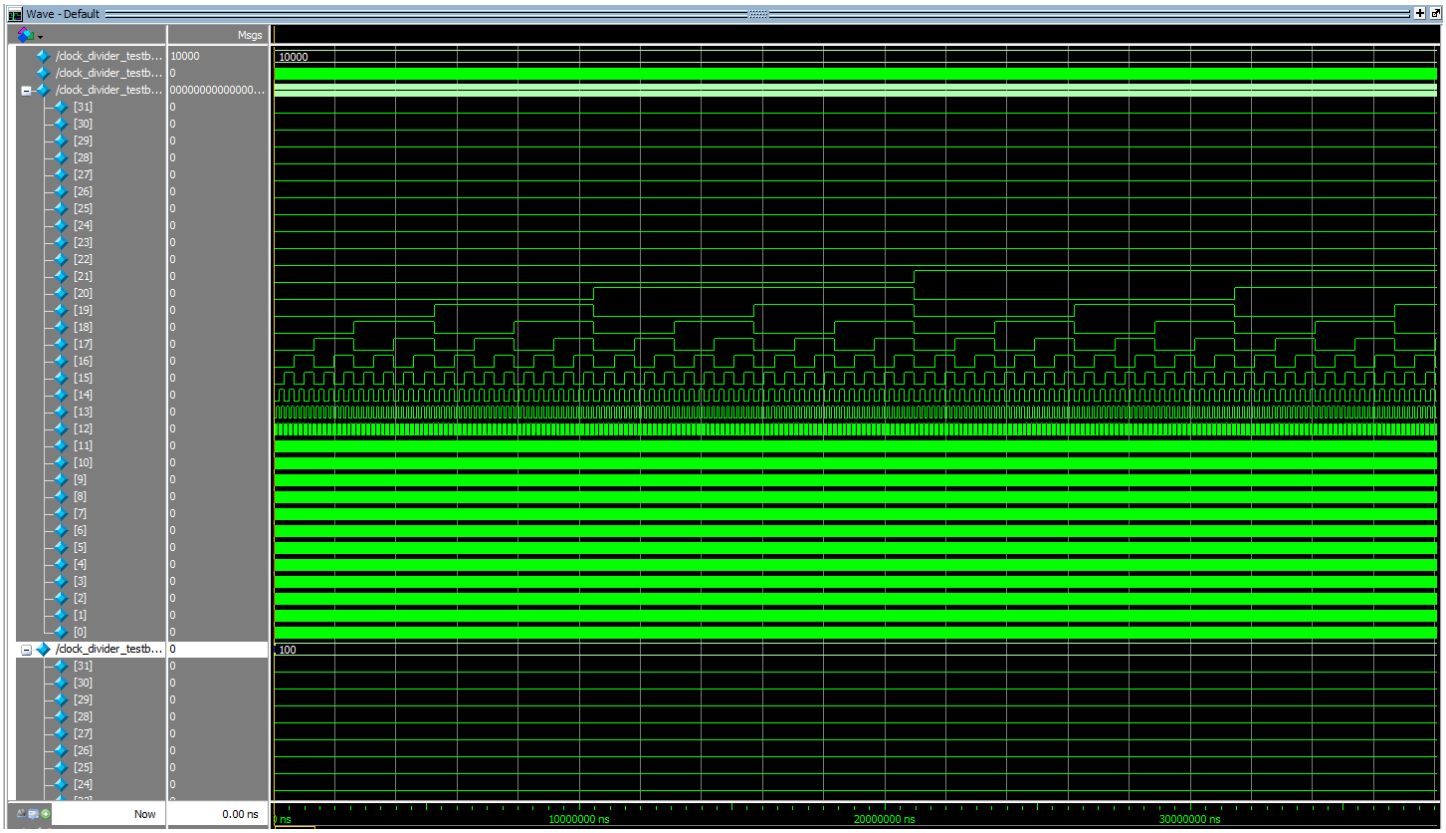


Figure 3.4 clock_divider.sv module Waveform

The clock divider works as expected. It is slowing down the clock by dividing as named.

Figure 3.6 Resource Utilization of System

Total System Utilization =

Demonstration

Task #1: *https://drive.google.com/file/d/14CbpZrPm4tZtAvTBbPAL9xKwR4BgR13F/view?usp=sharing*

Task #2:

Appendix

Task#1:

```
// Nattapon Oonlamom and Kiana Peterson
// 01/13/23
// EE 371
// Lab 1: Parkinglot Occupacy

// This module show how many cars are occupying the lot
// Display full when max capacity reach and clear when non occupied

// Overall inputs and outputs for the counter module listed below:
// Inputs: 1-bit clk, 5bit count
// Outputs: 6 7-bit HEXs display
module display(clk, count, HEX5, HEX4, HEX3, HEX2, HEX1, HEX0);
   input logic   clk;
   input logic   [4:0] count;
   output logic  [6:0] HEX5, HEX4, HEX3, HEX2, HEX1, HEX0;

   //                          6543210
   parameter [6:0]           zero  =   7'b1000000,
                     one   =  7'b1111001,
                     two   =  7'b0100100,
                     three =  7'b0110000,
                     four  =  7'b0011001,
                     five  =  7'b0010010,
                     six   =  7'b0000010,
                     seven =  7'b1111000,
                     eight =  7'b0000000,
                     nine  =  7'b0010000,
                     F       =  7'b0001110,
                     U     =  7'b1000001,
                     L     =  7'b1000111,
                     E     =  7'b0000110,
                     C     =  7'b1000110,
                     A     =  7'b0001000,
                     r     =  7'b1001110,
                     blk     =  7'b1111111;

   // combination logic for HEXs
   always_comb begin
      case(count)
         0: begin HEX5 = C;   HEX4 = L;   HEX3 = E;   HEX2 = A;     HEX1 = r;   HEX0 = zero;  end
         1:  begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk; HEX1 = blk; HEX0 = one;   end
         2:  begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk; HEX1 = blk; HEX0 = two;   end
         3:  begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk; HEX1 = blk; HEX0 = three; end
         4:  begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk; HEX1 = blk; HEX0 = four;  end
         5:  begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk; HEX1 = blk; HEX0 = five;  end
         6:  begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk; HEX1 = blk; HEX0 = six;   end
         7:  begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk; HEX1 = blk; HEX0 = seven; end
         8:  begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk; HEX1 = blk; HEX0 = eight; end
         9:  begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk; HEX1 = blk; HEX0 = nine;  end
         10: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk; HEX1 = one; HEX0 = zero;  end
         11: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk; HEX1 = one; HEX0 = one;   end
         12: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk; HEX1 = one; HEX0 = two;   end
         13: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk; HEX1 = one; HEX0 = three; end
         14: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk; HEX1 = one; HEX0 = four;  end
         15: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk; HEX1 = one; HEX0 = five;  end
         16: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk; HEX1 = one; HEX0 = six;   end
         17: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk; HEX1 = one; HEX0 = seven; end
         18: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk; HEX1 = one; HEX0 = eight; end
         19: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk; HEX1 = one; HEX0 = nine;  end
         20: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk; HEX1 = two; HEX0 = zero;  end
         21: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk; HEX1 = two; HEX0 = one;   end
         22: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk; HEX1 = two; HEX0 = two;   end
         23: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk; HEX1 = two; HEX0 = three; end
         24: begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk; HEX1 = two; HEX0 = four;  end
         25: begin HEX5 = F;   HEX4 = U;   HEX3 = L;   HEX2 = L;   HEX1 = two; HEX0 = five;  end
         default: begin HEX5 = 7'bx; HEX4 = 7'bx; HEX3 = 7'bx; HEX2 = 7'bx; HEX1 = 7'bx; HEX0 = 7'bx; end
      endcase
   end
endmodule
```

Figure 4 display Module

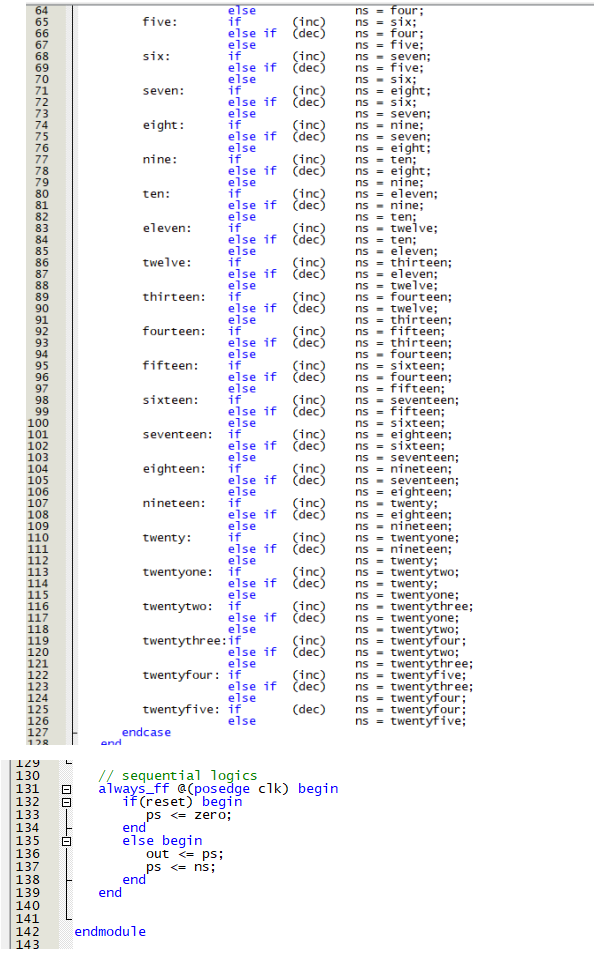
```
// Nattapon Oonlamom and Kiana Peterson
// 01/13/23
// EE 371
// Lab 1: Parkinglot Occupacy

// This module takes in a clock, reset, and an input (w) and creates a
// Finite State Machine that determines whether a player just pressed their button
// keep track of how many cars are occupying the lot

// Inputs:
//    clk: Clock input
//    reset: Sets the current state to a specified reset state
//    a, b: two parking lot sensors tracking movement
// Outputs:
//    enter: whether a car entered
//    exit: whether a car exited
module parkingLot(clk, reset, a, b, enter, exit);
   input logic clk, reset, a, b;
   output logic enter, exit;

   logic entering;

   enum {A, B, C, D} present, next; // Creates a present state and next state for the cases A and B

   // Goes through all possible cases of the FSM
    always_comb begin
      case(present)

         A: begin
            if (a & ~b) begin
               next = B;
            end
            else if (~a & b) begin
               next = B;
            end
            else begin
               next = A;
            end
         end

         B: begin

            if (~a & ~b) begin
               next = A;
            end

            else if (a & b) begin
               next = C;
            end

            else if (a & ~b) begin
               if (entering) begin
                  next = B;
               end

               else begin
                  next = A;
               end
            end

               else begin
                  if (entering) begin
                     next = A;
                  end

                  else begin
                     next = B;
                  end
               end
          end

          C: begin
             if (a & ~b) begin
                if (entering) begin
                   next = B;
                end

                else begin
                   next = D;
                end
             end

             else if (~a & b) begin
                if (entering) begin
                   next = D;
                end

                else begin
                   next = B;
                end
             end

             else if (a & b) begin
                next = C;
             end
             else begin
                next = A;
             end
          end

          D: begin
             if (a & b) begin
                next = C;
             end

             else if (~a & ~b) begin
                next = A;
             end

             else begin
                next = D;
             end
          end
      endcase
   end

    // sequential logic
    // Determines when the changes will happen on the clock
    // Positive edge triggered
    always_ff @(posedge clk) begin
       // Makes A the reset state
       if (reset) begin
          present <= A;
          enter <= 0;
          exit <= 0;
          entering <= 0;
       end

       // If no reset, the "present" state becomes the "next" state
       else begin
          if (present == A) begin
             if (a && ~b) begin
                entering <= 1;
             end
             else if (~a && b) begin
                entering <= 0;
             end
          end
          else if (present == D) begin
             if (~a && ~b) begin
                if (entering == 1) begin
                   enter <= 1;
                   exit <= 0;
                end
                else begin
                   exit <= 1;
                   enter <= 0;
                end
             end
          end
          if (enter == 1 || exit == 1) begin
             enter <= 0;
             exit <= 0;
             entering <= 0;
          end
          present <= next;
       end
    end
endmodule
```

Figure 5 parkinglot Module

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

   input logic CLOCK_50;
   input logic [9:0] SW;
   input logic [3:0] KEY;
   output logic [6:0] HEX5, HEX4, HEX3, HEX2, HEX1, HEX0;
   output logic [9:0] LEDR;

   logic enter;
   logic exit;
   logic reset;
   logic [4:0] capacity;

   // Pinouts changed from GPIO_0 to V_GPIO. The pinouts are as follows:
   // Inputs:
   //    V_GPIO[24] = the left switch acting as sensor a
   //    V_GPIO[28] = the right switch acting as sensor b
   // Outputs:
   //    V_GPIO[32] = the left ledr corresponding to sensor a, given by the left switch
   //    V_GPIO[34] = the right ledr corresponding to sensor b, given by the right sitch
   inout logic [35:23] V_GPIO;
   assign V_GPIO[32] = V_GPIO[24]; // Wired the left LEDR (sensor a) to the left switch
   assign V_GPIO[34] = V_GPIO[28]; // Wired the right LEDR (sensor b) to the right switch
   assign reset = SW[9]; // Changed the reset to be attached to switch 9

   logic [31:0] clk;
   parameter whichClock = 20;

   // instantiates clock_divider to slow down clock
   clock_divider cdiv (.clock(CLOCK_50), .divided_clocks(clk));

   // instantiates parkingLot to track cars entering or exiting the lot with sensors
   // Inputs a and b changed to the correct pinouts
   parkingLot theParkingLot (.clk(clk[whichClock]), .reset(reset), .a(V_GPIO[24]), .b(V_GPIO[28]), .enter(enter), .exit(exit));

   // instantiates counter to keep track of how many cars are occupying the lot
   counter carCapacity (.clk(clk[whichClock]), .reset(reset), .inc(enter), .dec(exit), .out(capacity));

   // instantiate display to show how many cars are occupying the lot
   display carCount (.clk(clk[whichClock]), .count(capacity), .HEX5(HEX5), .HEX4(HEX4), .HEX3(HEX3), .HEX2(HEX2), .HEX1(HEX1), .HEX0(HEX0));

endmodule
```

Figure 6 DE1_SoC Module (Task1)

```
// Nattapon Oonlamom and Kiana Peterson
// 01/13/23
// EE 371
// Lab 1: Parkinglot Occupacy

// This module keep track of how many cars are occupying the lot.
// Counter increment by 1 if the car enters
// and decrement by 1 if the car exit
// with an assumption that the max capacity is 25

// Overall inputs and outputs for the counter module listed below:
// Inputs: 1-bit clk, reset, inc, dec
// Outputs: 5 bits out
module counter(clk, reset, inc, dec, out);
   input logic clk, reset, inc, dec;
   output logic [4:0] out;

   // logic for comb_logic
   logic [4:0] ps, ns;

   parameter [4:0]   zero        = 5'b00000,
                     one         = 5'b00001,
                     two         = 5'b00010,
                     three       = 5'b00011,
                     four        = 5'b00100,
                     five        = 5'b00101,
                     six         = 5'b00110,
                     seven       = 5'b00111,
                     eight       = 5'b01000,
                     nine        = 5'b01001,
                     ten         = 5'b01010,
                     eleven      = 5'b01011,
                     twelve      = 5'b01100,
                     thirteen    = 5'b01101,
                     fourteen    = 5'b01110,
                     fifteen     = 5'b01111,
                     sixteen     = 5'b10000,
                     seventeen   = 5'b10001,
                     eighteen    = 5'b10010,
                     nineteen    = 5'b10011,
                     twenty      = 5'b10100,
                     twentyone   = 5'b10101,
                     twentytwo   = 5'b10110,
                     twentythree = 5'b10111,
                     twentyfour  = 5'b11000,
                     twentyfive  = 5'b11001;

   // combinational logic
   always_comb begin
      case(ps)
         zero:       if      (inc)    ns = one;
                     else             ns = zero;
         one:        if      (inc)    ns = two;
                     else if (dec)    ns = zero;
                     else             ns = one;
         two:        if      (inc)    ns = three;
                     else if (dec)    ns = one;
                     else             ns = two;
         three:      if      (inc)    ns = four;
                     else if (dec)    ns = two;
                     else             ns = three;
         four:       if      (inc)    ns = five;
                     else if (dec)    ns = three;
                     else             ns = four;
         five:       if      (inc)    ns = six;
```



```
                    else            ns = four;
        five:       if      (inc)   ns = six;
                    else if (dec)   ns = four;
                    else            ns = five;
        six:        if      (inc)   ns = seven;
                    else if (dec)   ns = five;
                    else            ns = six;
        seven:      if      (inc)   ns = eight;
                    else if (dec)   ns = six;
                    else            ns = seven;
        eight:      if      (inc)   ns = nine;
                    else if (dec)   ns = seven;
                    else            ns = eight;
        nine:       if      (inc)   ns = ten;
                    else if (dec)   ns = eight;
                    else            ns = nine;
        ten:        if      (inc)   ns = eleven;
                    else if (dec)   ns = nine;
                    else            ns = ten;
        eleven:     if      (inc)   ns = twelve;
                    else if (dec)   ns = ten;
                    else            ns = eleven;
        twelve:     if      (inc)   ns = thirteen;
                    else if (dec)   ns = eleven;
                    else            ns = twelve;
        thirteen:   if      (inc)   ns = fourteen;
                    else if (dec)   ns = twelve;
                    else            ns = thirteen;
        fourteen:   if      (inc)   ns = fifteen;
                    else if (dec)   ns = thirteen;
                    else            ns = fourteen;
        fifteen:    if      (inc)   ns = sixteen;
                    else if (dec)   ns = fourteen;
                    else            ns = fifteen;
        sixteen:    if      (inc)   ns = seventeen;
                    else if (dec)   ns = fifteen;
                    else            ns = sixteen;
        seventeen:  if      (inc)   ns = eighteen;
                    else if (dec)   ns = sixteen;
                    else            ns = seventeen;
        eighteen:   if      (inc)   ns = nineteen;
                    else if (dec)   ns = seventeen;
                    else            ns = eighteen;
        nineteen:   if      (inc)   ns = twenty;
                    else if (dec)   ns = eighteen;
                    else            ns = nineteen;
        twenty:     if      (inc)   ns = twentyone;
                    else if (dec)   ns = nineteen;
                    else            ns = twenty;
        twentyone:  if      (inc)   ns = twentytwo;
                    else if (dec)   ns = twenty;
                    else            ns = twentyone;
        twentytwo:  if      (inc)   ns = twentythree;
                    else if (dec)   ns = twentyone;
                    else            ns = twentytwo;
        twentythree:if      (inc)   ns = twentyfour;
                    else if (dec)   ns = twentytwo;
                    else            ns = twentythree;
        twentyfour: if      (inc)   ns = twentyfive;
                    else if (dec)   ns = twentythree;
                    else            ns = twentyfour;
        twentyfive: if      (dec)   ns = twentyfour;
                    else            ns = twentyfive;
    endcase
end

// sequential logics
always_ff @(posedge clk) begin
    if(reset) begin
        ps <= zero;
    end
    else begin
        out <= ps;
        ps <= ns;
    end
end

endmodule
```

Figure 7 counter Module

Task#2:

Figure 8