Kiana Peterson and Nattapon Oonlmaom
EE 371
January 13, 2023
Lab 1 Report

Procedure

In this lab, the purpose was to review concepts from EE 271. The task was to design an FSM that monitors the activity of cars in a parking lot with a single enter and exit gate. The first step was to brainstorm what is needed in the system and draw a block diagram of the system that represents how the FSMs will be connected.

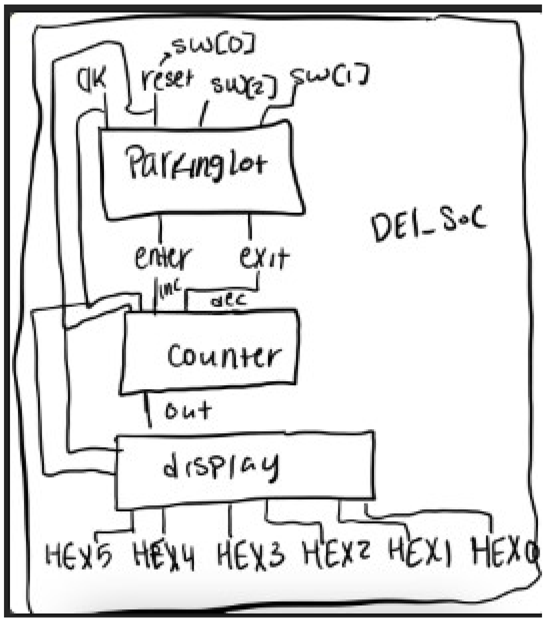
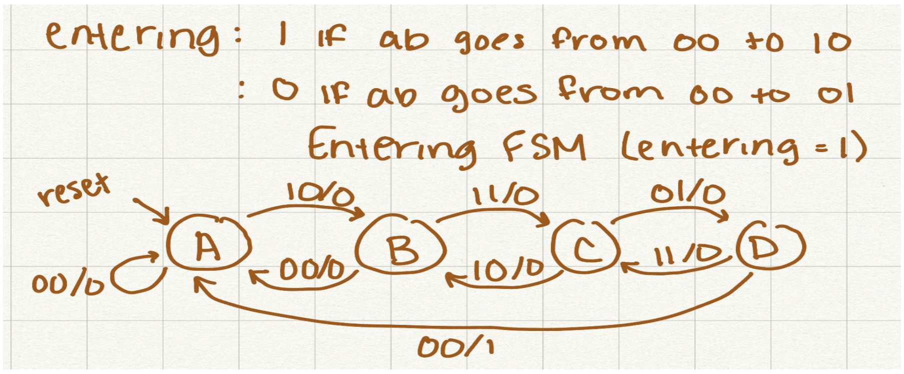


Figure 1 Block Diagram of the System

The system will be comprised of parkinglot, counter, and display, which will be instantiated in the main module DE1_SoC. The parkinglot is the field of sensors that detects whether cars come in or out. The counter keeps track of the occupancy of the parkinglot, while the display updates the occupancy and presents that data on a HEX display. 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.



entering : 1 if ab goes from 00 to 10
: 0 if ab goes from 00 to 01
Entering FSM (entering = 1)
reset
00/0
10/0
11/0
01/0
A
B
C
D
00/0
10/0
11/0
00/1

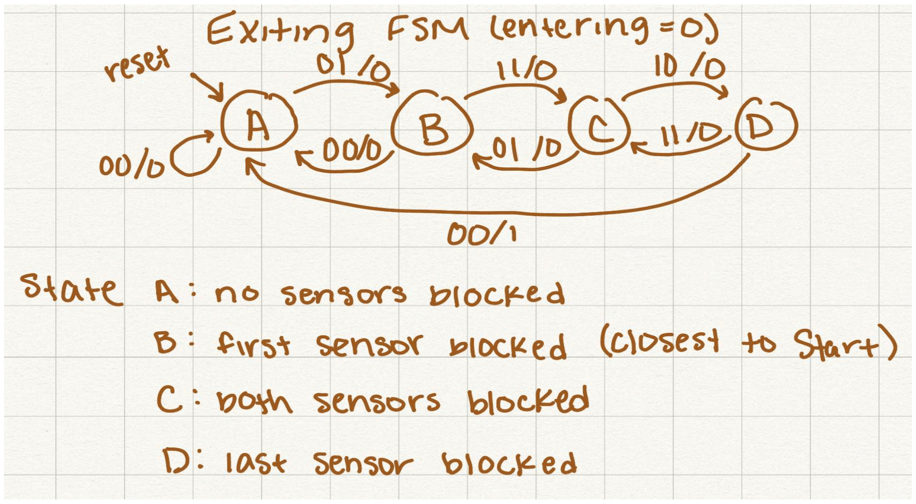

Exiting FSM (entering = 0)
reset
00/0
01/0
00/0
11/0
01/0
10/0
11/0
00/1
A
B
C
D
State A: no sensors blocked
B: first sensor blocked (closest to Start)
C: both sensors blocked
D: last sensor blocked

Figure 2 parkinglot State Diagram

Results

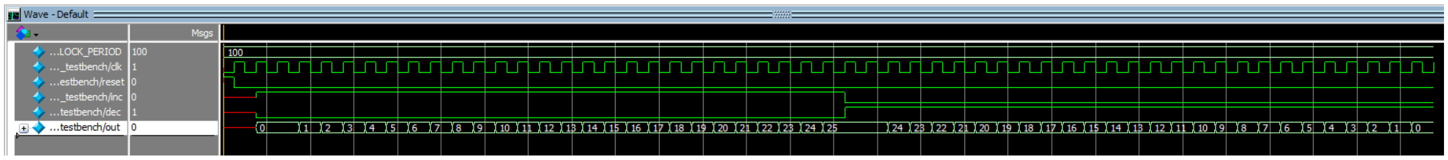


Figure 3 counter Waveform

The counter module worked as expected according to the waveform as it counts up by one up to 25 as incremented while counts down to 0 as it is decremented.

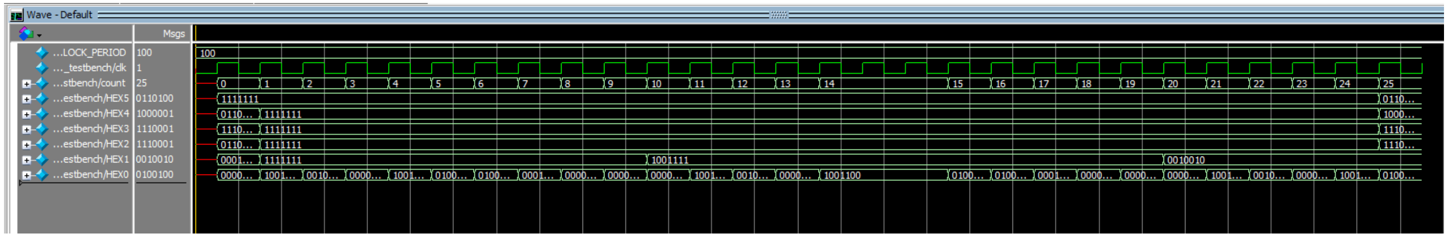


Figure 4 display Waveform

The display module is working as expected as is display FULL25 when the parkinglot 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.

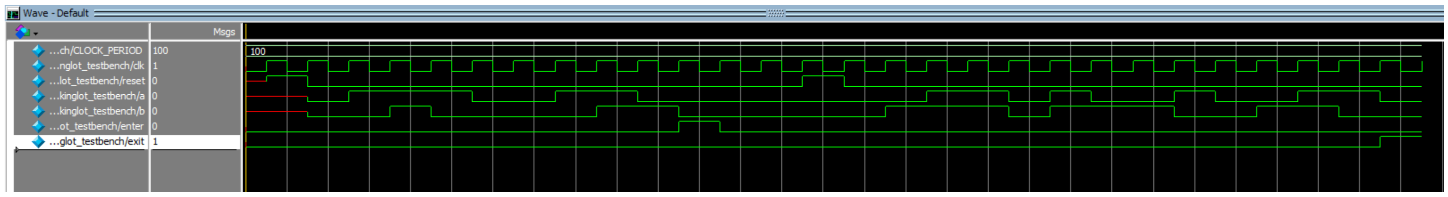


Figure 5 parkinglot Waveform

The parkinglot is working as expected as it senses if cars enter, exit, or change direction. It also does not count if the person is passed through the center.

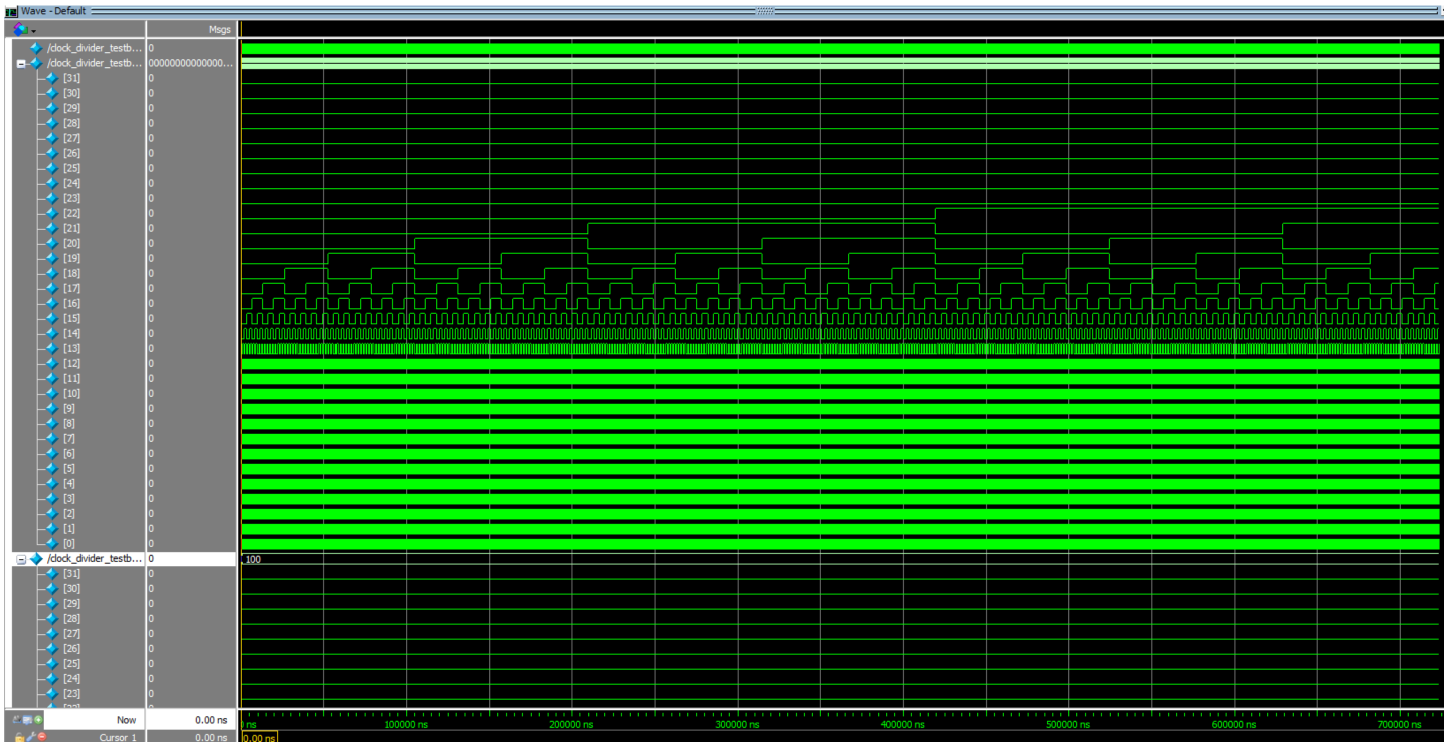


Figure 6 clock_divider Waveform

The clock divider functions to delay the clock as expected, so we can see the outputs.

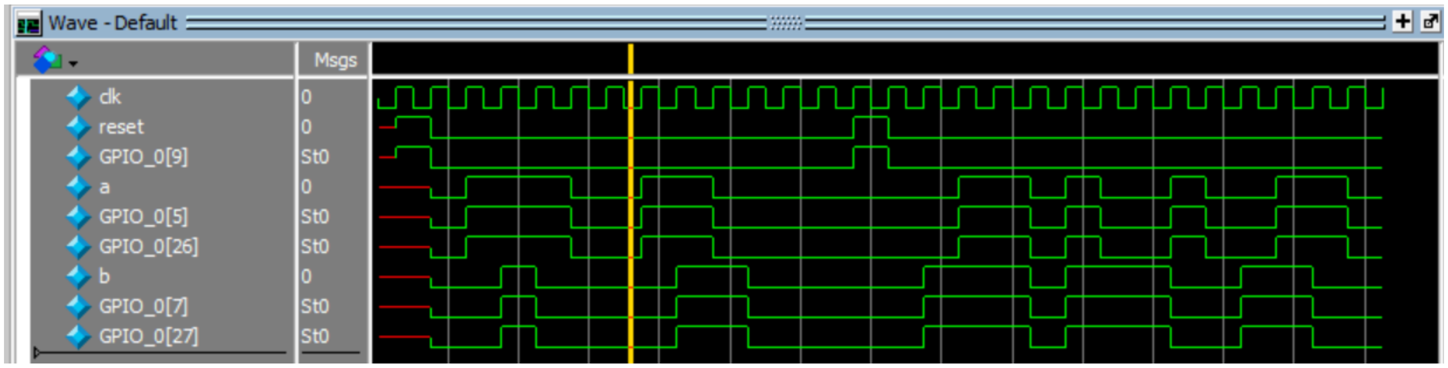
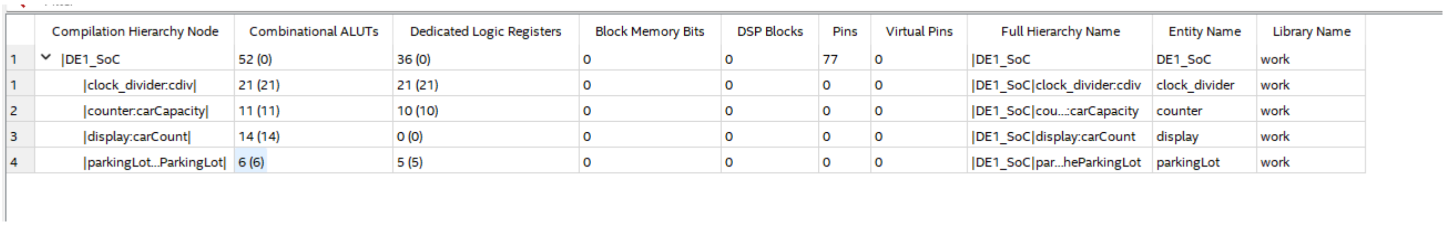


Figure 7 DE1_SoC Waveform

The DE1_SoC module is functioning correctly as it is displaying the correct output from the inputs given on the testbench and aligned with the truth table.



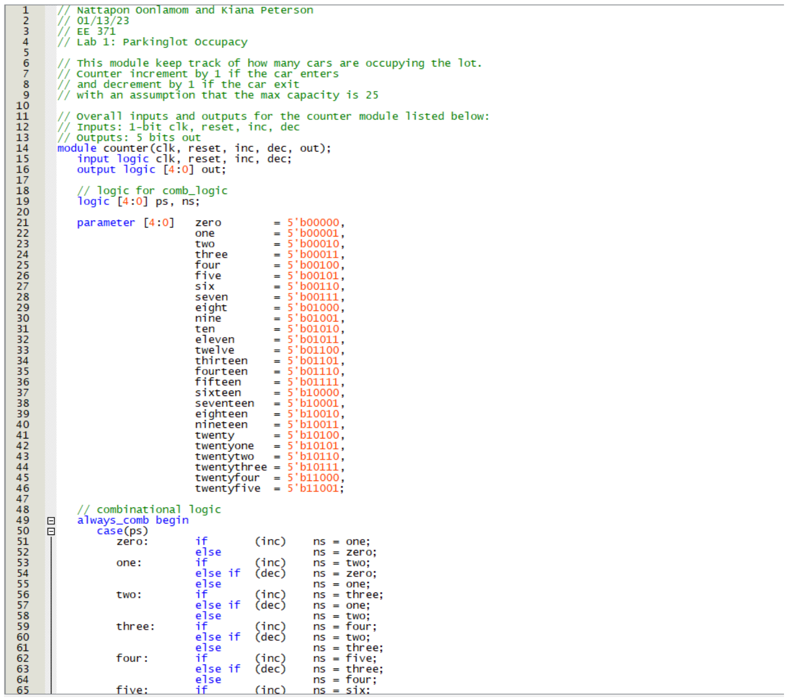
	Compilation Hierarchy Node	Combinational ALUTs	Dedicated Logic Registers	Block Memory Bits	DSP Blocks	Pins	Virtual Pins	Full Hierarchy Name	Entity Name	Library Name
1	|DE1_SoC	52 (0)	36 (0)	0	0	77	0	|DE1_SoC	DE1_SoC	work
1	|clock_divider:cdiv|	21 (21)	21 (21)	0	0	0	0	|DE1_SoC|clock_divider:cdiv	clock_divider	work
2	|counter:carCapacity|	11 (11)	10 (10)	0	0	0	0	|DE1_SoC|cou...:carCapacity	counter	work
3	|display:carCount|	14 (14)	0 (0)	0	0	0	0	|DE1_SoC|display:carCount	display	work
4	|parkingLot...ParkingLot|	6 (6)	5 (5)	0	0	0	0	|DE1_SoC|par...heParkingLot	parkingLot	work

Figure 8 Resource Utilization of System

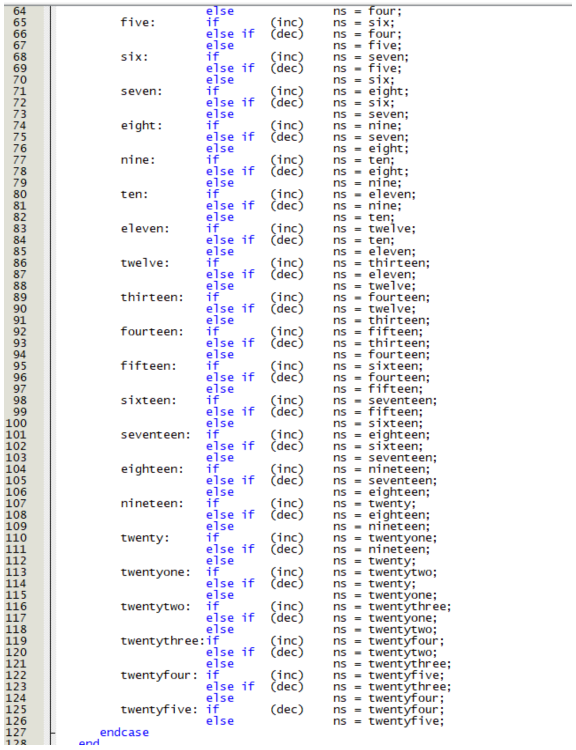
Total System Utilization = 52+36+11+10+14+6+5 = 134

Demonstration: https://drive.google.com/file/d/13Jikk0wxOI1IwPM3EYMZgVS55CwNk4a8/view

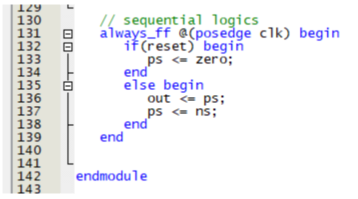
Appendix



```
// 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;
```

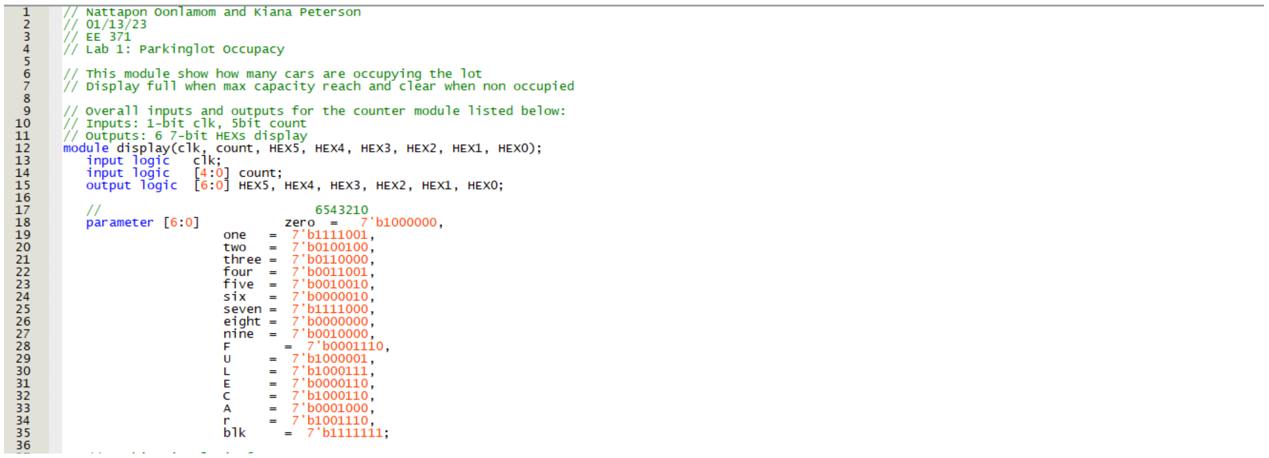


```
                     else              ns = four;
         five:       if       (inc)    ns = six;
                     else if  (dec)    ns = four;
                     else              ns = five;
         six:        if       (inc)    ns = seven;
                     else if  (dec)    ns = five;
                     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
```

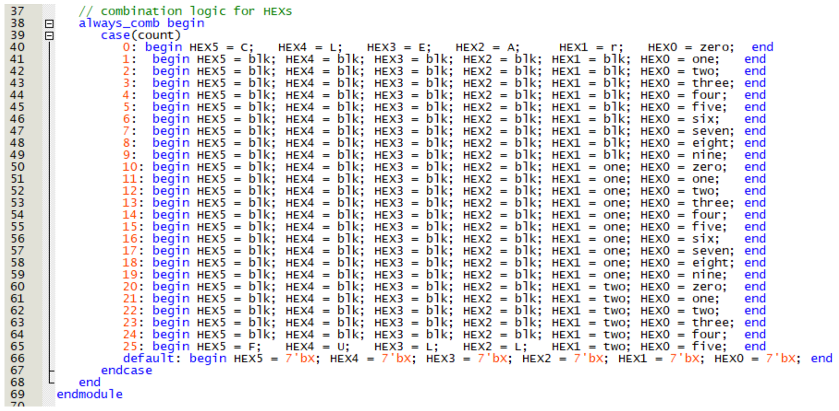


```
   // sequential logics
   always_ff @(posedge clk) begin
      if(reset) begin
         ps <= zero;
      end
      else begin
         out <= ps;
         ps <= ns;
      end
   end


endmodule
```

Figure 9 counter Module

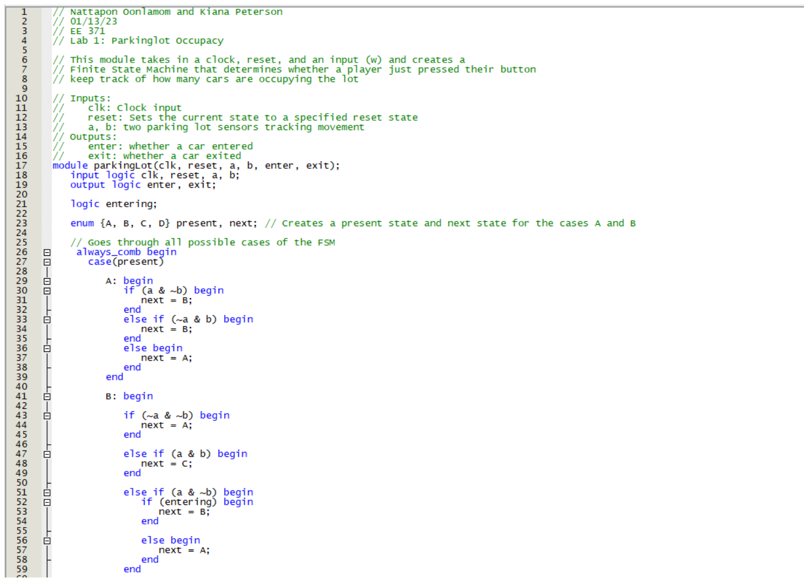


```
// Nattapon Oonlamom and Kiana Peterson
// 01/13/23
// EE 371
// Lab 1: Parkinglot Occupacy

// 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;
```

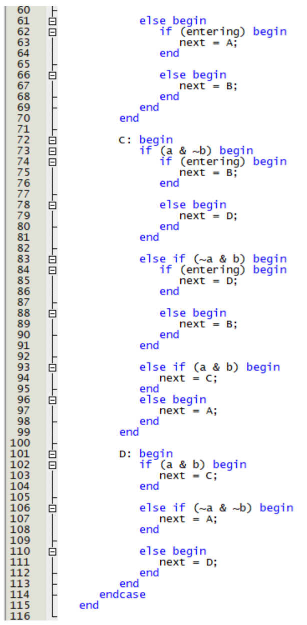


```
   // combination logic for HEXs
   always_comb begin
      case(count)
         0: begin HEX5 = C;   HEX4 = L;   HEX3 = E;   HEX2 = A;     HEX1 = r;   HEX0 = zero;  end
         1:  begin HEX5 = blk; HEX4 = blk; HEX3 = blk; HEX2 = blk; HEX1 = blk; HEX0 = one;   end
         2:  begin HEX5 = blk; 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
```

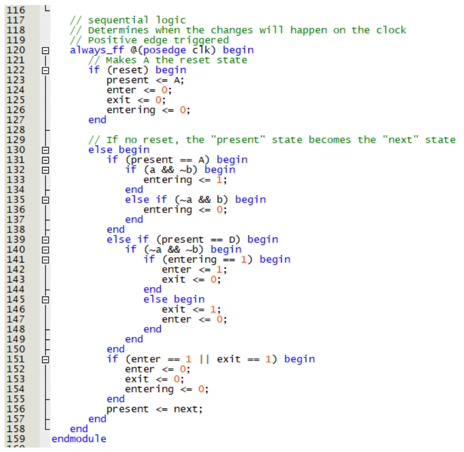
Figure 10 display Module



```
// 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
```

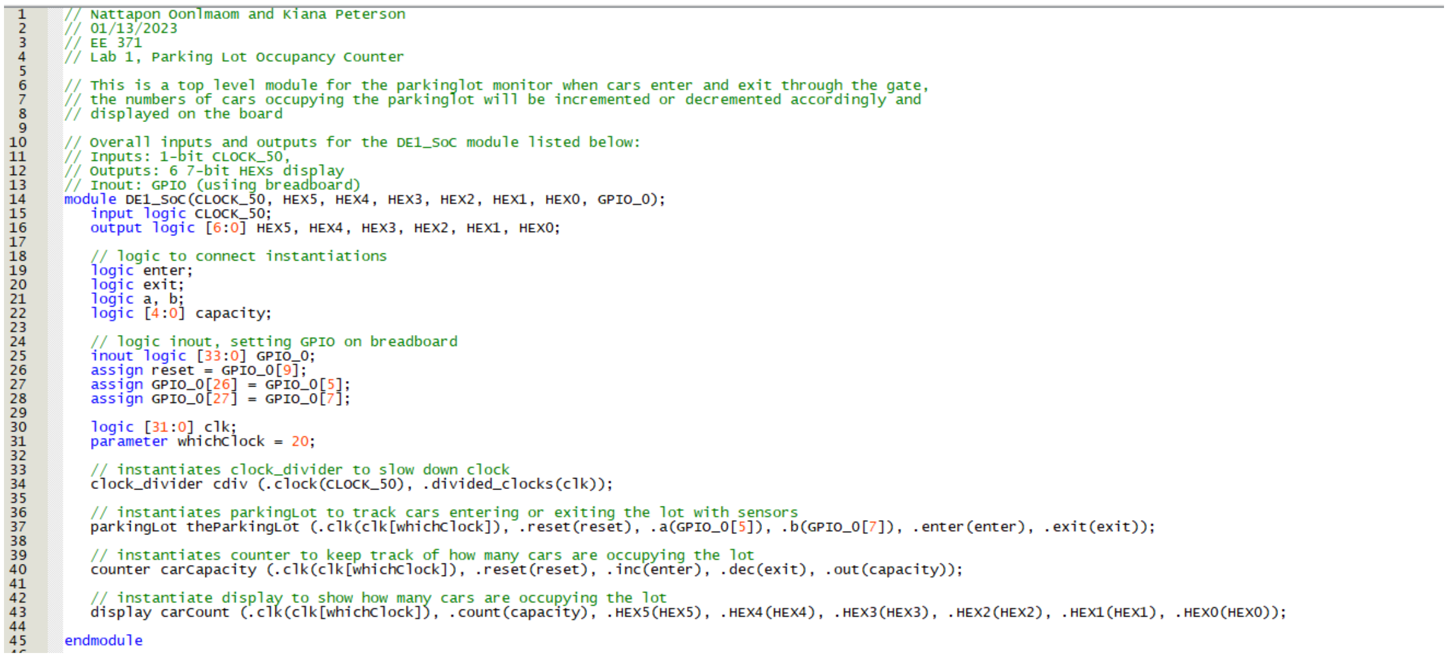


```

            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 11 parkinglot Module



```
// Nattapon Oonlmaom and Kiana Peterson
// 01/13/2023
// EE 371
// Lab 1, Parking Lot Occupancy Counter

// This is a top level module for the parkinglot monitor when cars enter and exit through the gate,
// the numbers of cars occupying the parkinglot will be incremented or decremented accordingly and
// displayed on the board

// Overall inputs and outputs for the DE1_SoC module listed below:
// Inputs: 1-bit CLOCK_50,
// Outputs: 6 7-bit HEXs display
// Inout: GPIO (usiing breadboard)
module DE1_SoC(CLOCK_50, HEX5, HEX4, HEX3, HEX2, HEX1, HEX0, GPIO_0);
   input logic CLOCK_50;
   output logic [6:0] HEX5, HEX4, HEX3, HEX2, HEX1, HEX0;

   // logic to connect instantiations
   logic enter;
   logic exit;
   logic a, b;
   logic [4:0] capacity;

   // logic inout, setting GPIO on breadboard
   inout logic [33:0] GPIO_0;
   assign reset = GPIO_0[9];
   assign GPIO_0[26] = GPIO_0[5];
   assign GPIO_0[27] = GPIO_0[7];

   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
   parkingLot theParkingLot (.clk(clk[whichClock]), .reset(reset), .a(GPIO_0[5]), .b(GPIO_0[7]), .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 12 DE1_SoC Module