Xuanchang Hu
03/13/2023
EE 371
Lab #6

Procedure

The goal for lab 6 is to redesign a parking lot counter. For this, I work on two tasks. The first task aims to help us familiarize ourselves with the new ports on 3D parking lots. We need to connect the switch to input V_GPIO and then connect the LEDR to output V_GPIO. For task2, I designed a more complicated parking lot counter. The parking lot has 3 spaces. The work time is 8 hours. We need to show the time and number of cars in the parking lot. If The parking is full, display FULL on HEX. After the end of day, display the rush hour(The first hour when parking is full) and the end of rush hour. Also it shows the maximum number of cars at each hour on HEX. To implement that, we need to use RAM to store the number of cars in each hour.

Task 1

For task 1, I already did it in lab2. But this time, I need to connect input and output to V_GPIO. To do this, I connect resetSW to V_GPIO[23]; (The leftmost switch is reset); leftSW to V_GPIO[24]; rightSW to V_GPIO[28]; V_GPIO[32] to V_GPIO[24]; V_GPIO[33] to V_GPIO[28]. When upload to labsland, I need to manually connect all the switch and LEDR to breadboard.

Task 2

In task 2, I designed an upgraded version of the 3D parking lots. This time, the parking lot has 3 spaces, and the work day has 8 hours. In the 8 hours, cars will enter and exit the parking lots. HEX5 is designed to show the time, HEX0 is designed to show the number of cars. When parking is full, HEX3 to 0 should show FULL. So I designed a **fsm** to trace the time. The **fsm** has 10 states: none, s1, s2, s3, s4, s5, s6, s7, s8, sDisplay. Everytime I increase the time, it goes to the next state. And s1 to s8 represents the hours. The **fsm** will output the hours. I also designed a datapath. It can count the number of cars, output the rushhour, end of rushhour and maximum number of cars. All the numbers will be sent to display.sv and converted to HEX numbers. After the end of 8 hours, I also need to display the maximum number of each hour on HEX1 and 0. So I designed a 8X16 RAM to store the maximum car data. Each hour corresponds to an address. In the end, HEX3 and 2 will show rushhour and endHour. HEX1 and 0 will show the address(hour) and number of maximum cars. The HEX display is controlled by control files. In the always_ff, when in the work day(endDay == 0), if num is 3, HEX5 is hour, HEX3 to 0 is FULL. If num is smaller than 3, HEX5 is hour, HEX0 is number of cars, other numbers don't show. When the condition is end of day(endDay == 1), HEX3 and 2 is the time of rushhour, end of rushhour. HEX1 and 0 will show the address(hour) and number of maximum cars. The control.sv file is the overall control of the program. And DE1_SoC will control the control.sv file.

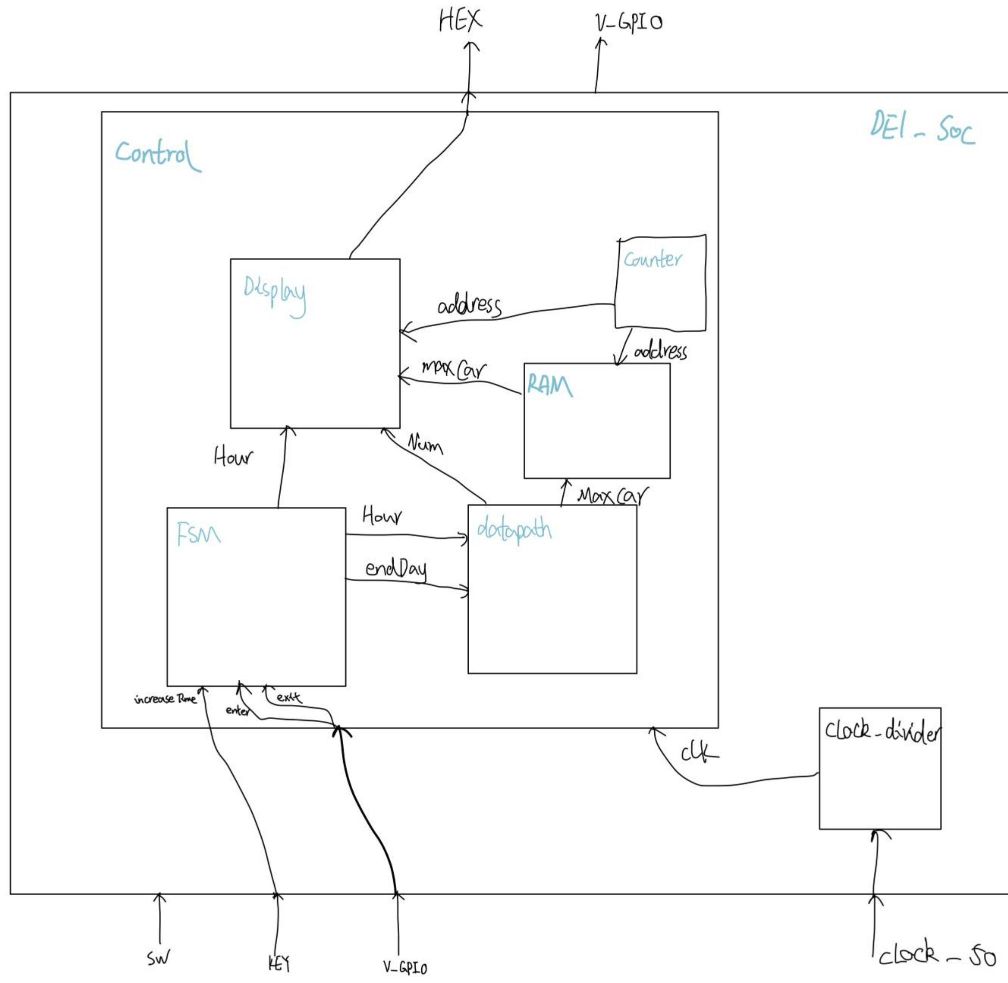


Figure 1:Block diagram of Lab 2 task 2

Task 2

ASMD

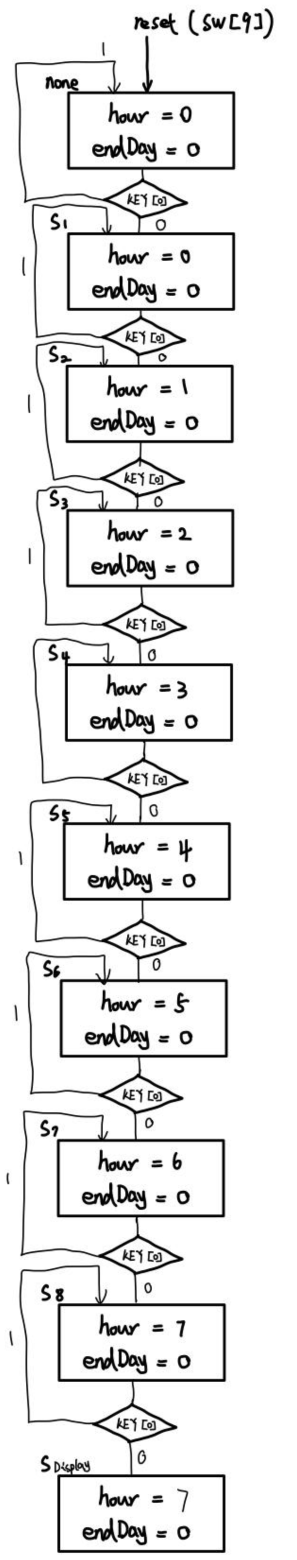


Figure 2:ASMD of Lab 2 task 2

FSM

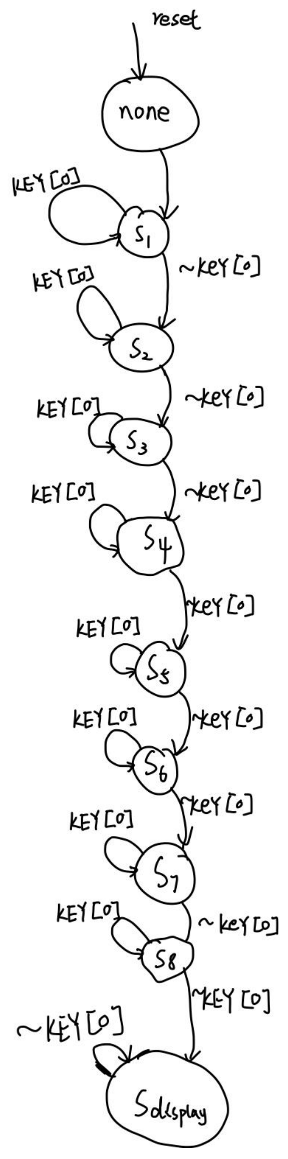


Figure 3:fsm of Lab 2 task 2

Results
Task 2

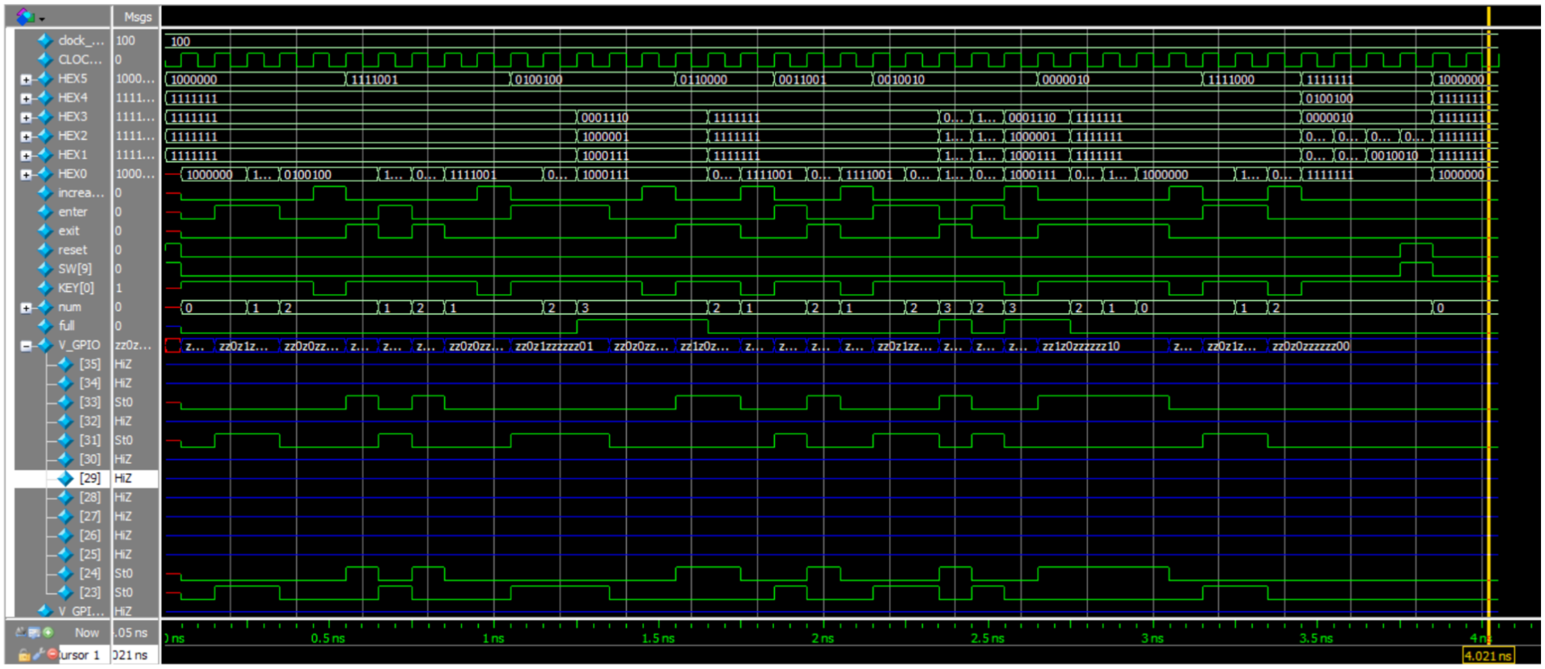


Figure 4: The waveform generated by the DE1_SoC_task2

For task 2 DE1_SoC, the testbench tests the cases of enter cars, exit cars. HEX5 shows the hours during 8 hour work time. When thr number of cars are 3, HEX3 to 0 shows FULL. Otherwise, HEX0 shows the number of cars. After the end of work day, the HEX3 to 2 will shows the rushhour and endHour. HEX1 will show the address(Hour) and HEX0 will show the maximum cars.

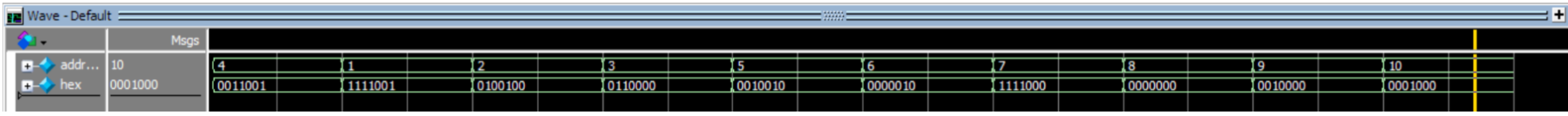


Figure 5: The waveform generated by the display

For task 2 display, It will take input numbers and output correspond HEX.

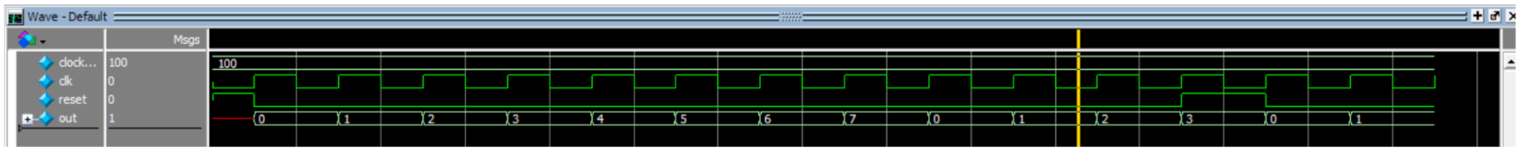


Figure 6: The waveform generated by counter

For task 2 counter, It will count from 0 to 7. After reset, the counter starts from 0 again.

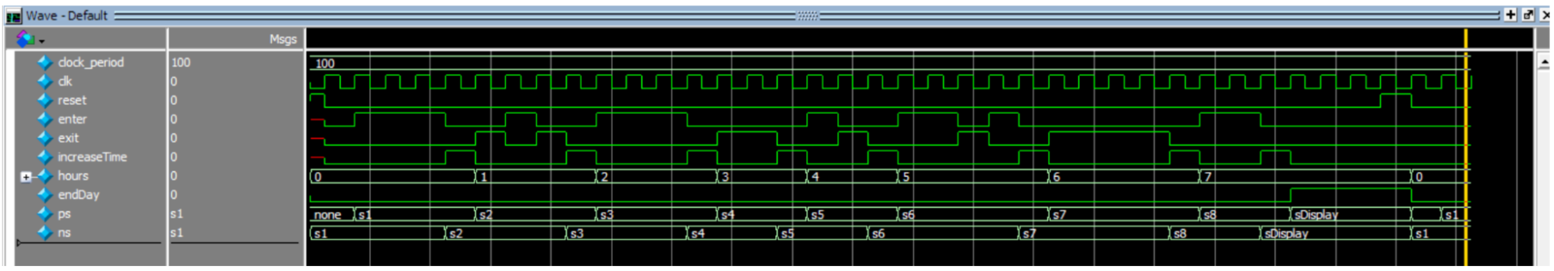


Figure 7: The waveform generated by fsm

For task 2 fsm, It takes enter, exit and increase time as input, output current time. Everytime time increases, the state will move to next state. Each state represents an hour and the hour will be outputed. In the simulation everytime I increase time, the hour increases by 1. At sDisplay state, the hour will remain 7. Only when reset can make sDisplay state go to none state and start over.

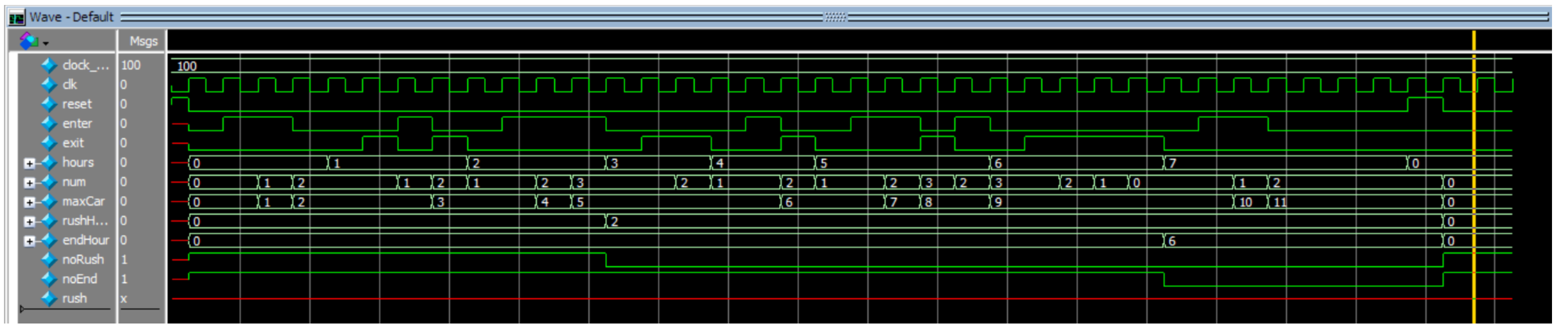


Figure 8: The waveform generated by datapath

For task 2 datapath, it count the number of cars in parking lots. It also output the maximum number of cars. rushHour was originally 0. When number is 3, the rushHour will be the current hour. Endhour also was 0. When number go back to 0, endHour will be current hour.

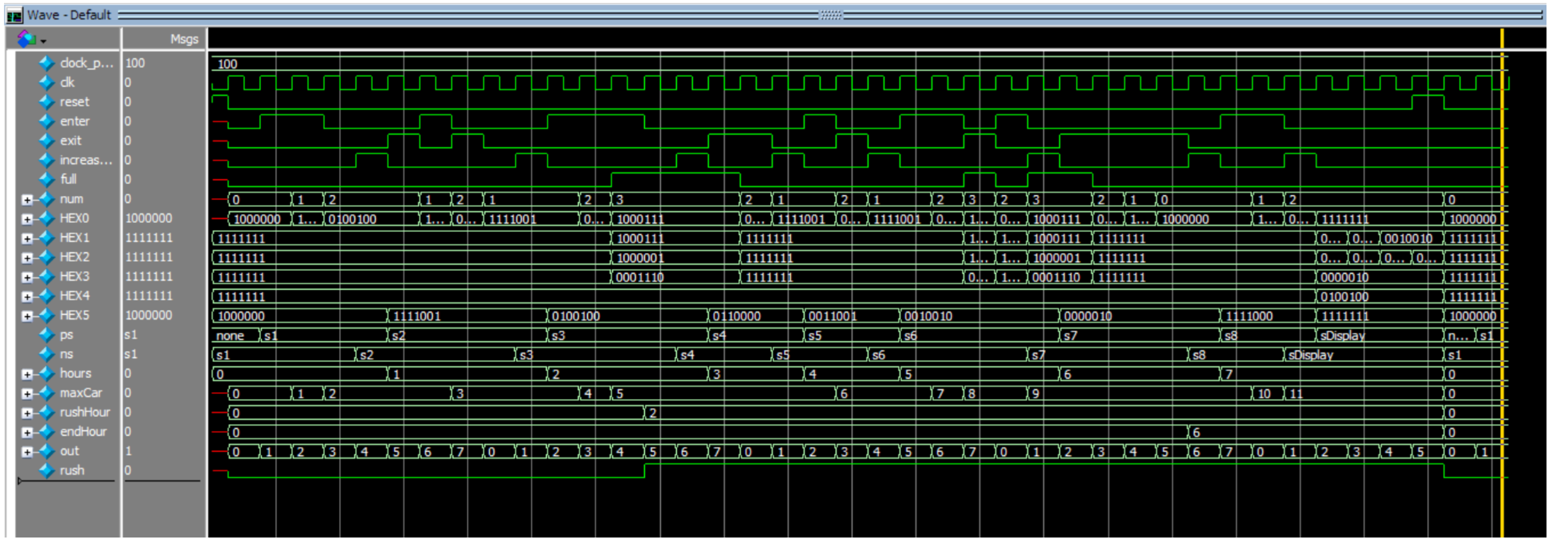


Figure 9: The waveform generated by control

For task 2 control, it's the overall control of the program. The testbench tests the cases of enter cars, exit cars. HEX5 shows the hours during 8 hour work time. When thr number of cars are 3, HEX3 to 0 shows FULL. Otherwise, HEX0 shows the number of cars. After the end of work day, the HEX3 to 2 will shows the rushhour and endHour. HEX1 will show the address(Hour) and HEX0 will show the maximum cars. The control will then be used in DE1_SoC to connect will V_GPIO ports.

Final products

For task1, I'm able to familiar with V_GPIO ports. For task2, The overall goal of this task is to learn to connect RAM, fsm, datapath together with V_GPIO ports. To keep track of the hour, I need to use fsm. To count the number of cars, maximum cars, rushHour and endHour, I need to use datapath.

And to store data of maximum cars, I need to use RAM. I also learned how to set HEX in different conditions. And to combine all these modules together, I created control. DE1_SoC only needs the control to run the whole program. I also learn that ASMD and FSM chart can help me draw a sketch of the program. I can implement the code directly through the charts I draw.

Appendix

Task 1

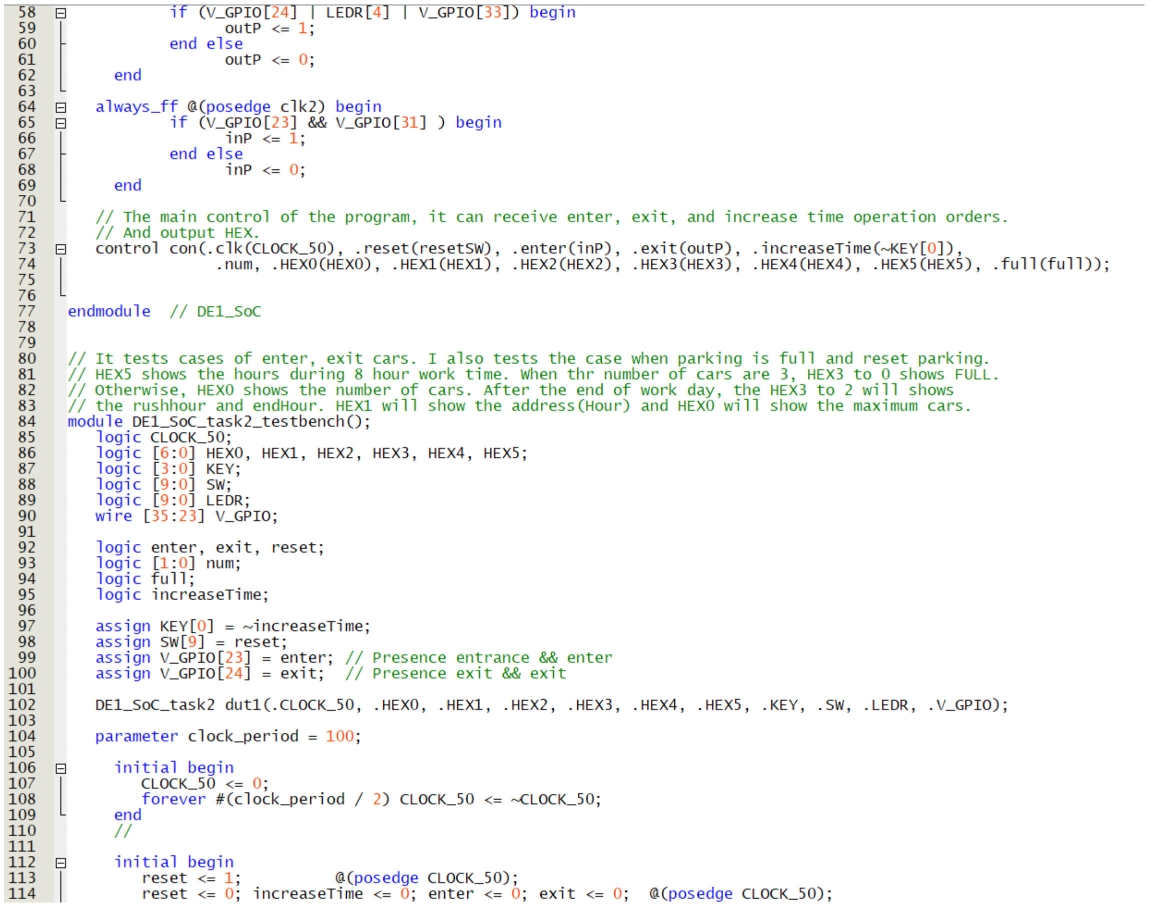
(1) DE1_SoC_task1

```
//Xuanchang Hu
//01/13/2023
//EE 371
//Lab 6

// DE1_SoC takes a clock and a 34-bits GPIo_0 as input, and 7-bits HEX0,
// HEX1, HEX2, HEX3, HEX4, HEX5 as output. The GPIO_0[10], GPIO_0[12], GPIO_0[14] serves for switches
// and GPIO_0[26] and GPIO_0[27] serves for leds. This module is the top-level for the parking lot
// system to implement.
// Top-level module that defines the I/Os for the DE-1 SoC board

module DE1_SoC_task1(CLOCK_50, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, V_GPIO);
    input logic CLOCK_50;
    inout logic [35:23] V_GPIO;
    output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;

    logic leftSW, rightSW, resetSW;

    assign resetSW = V_GPIO[23]; //The leftmost switch is reset A
    assign leftSW = V_GPIO[24]; //The middle switch among the 3 is connected to pin 15
    assign rightSW = V_GPIO[28]; //The right switch among the 3 is connected to pin 17
    assign V_GPIO[32] = V_GPIO[24]; // left led is connected to pin 15
    assign V_GPIO[33] = V_GPIO[28]; // right led is connected to pin 17

   logic enter, exit;
   logic [1:0] num;
   logic [1:0] units, tens;

   // fsm cars takes CLOCK_50 as clk input, and resetSW as reset input, leftSW, rightSW as sensor a and b input.
   // and returns if cars enters or exit parking lot as enter and exit.
   fsm cars(.clk(CLOCK_50), .reset(resetSW), .a(leftSW), .b(rightSW), .enter, .exit);

   // counter number takes CLOCK_50 as clk input, and resetSW as reset input, and enter and exit as inc and dec
   // it returns num as number of cars, tens as the tens digit of num, units as units digit of num.
   counter number(.clk(CLOCK_50), .reset(resetSW), .inc(enter), .dec(exit), .num, .tens, .units);

   // light display takes num as car number inputs, tens as the tens digit of num input,
   // units as units digit of num input.
   // it returns HEX from 5 to 0 to the display
   light display(.hex5(HEX5), .hex4(HEX4), .hex3(HEX3), .hex2(HEX2), .hex1(HEX1), .hex0(HEX0), .num, .tens, .units);


endmodule
```

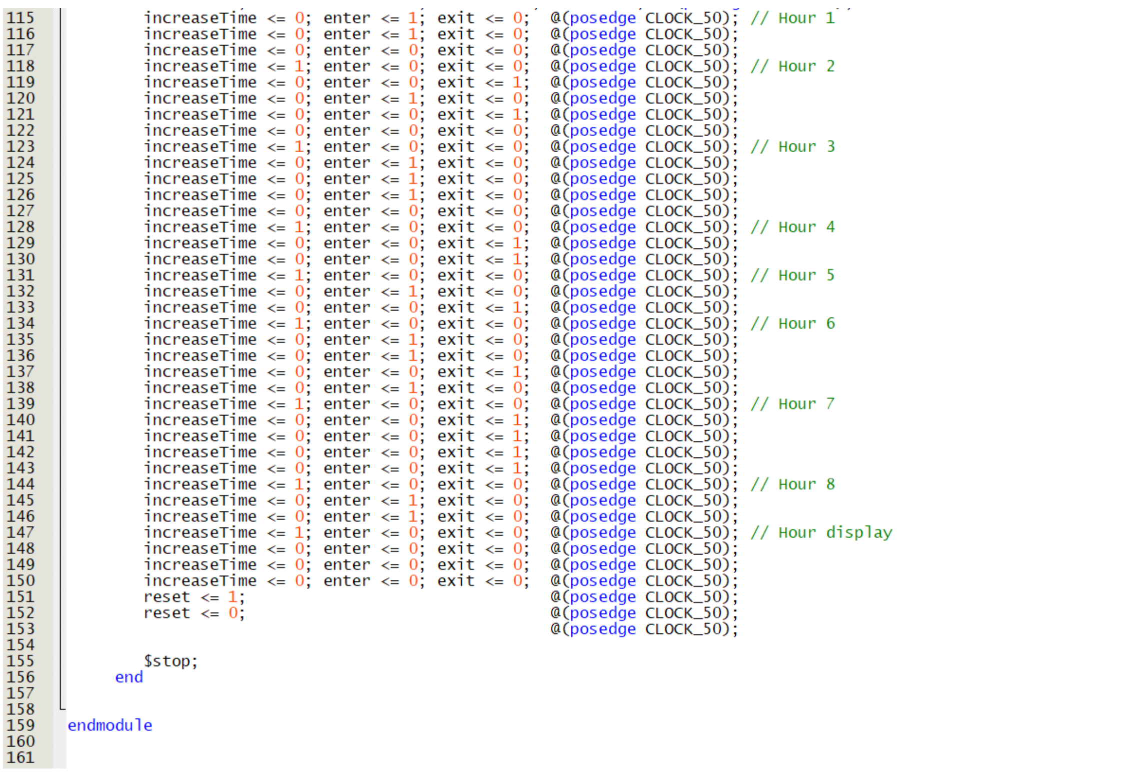
Task 2

1) **DE1_SoC_task2**

```
//Xuanchang Hu
//03/13/2023
//EE 371
//Lab6 task2
`timescale 1 ps / 1 ps

// This module is the top level module of parking system. It takes CLOCK_50, HEX, KEY, SW as input.
// LEDR as output. And V_GPIO for both input and output. It can make the parking system open gate, count
// show the time, number of cars, rushHour, endHOur, RAM address and maximum number of cars on HEX.
// It can also show whether the parking is full or not.
module DE1_SoC_task2 (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 inP, outP, resetSW; // into parking, exit parking, reset switch
   logic [1:0] num;  //Number
   logic full; //full
   logic inc; //increase


   assign resetSW = SW[9]; // reset switch is SW[9]


   // FPGA output
   assign V_GPIO[26] = V_GPIO[28];  // LED parking 1 && V_GPIO[28]
   assign V_GPIO[27] = V_GPIO[29];  // LED parking 2 && V_GPIO[29]
   assign V_GPIO[32] = V_GPIO[30];  // LED parking 3 && V_GPIO[30]
   assign V_GPIO[34] = V_GPIO[28] & V_GPIO[29] & V_GPIO[30];   // LED that show parking is 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

   logic [31:0] div_clk;

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

   logic clk1, clk2;
   parameter whichClock = 26; // 0.75 Hz clock

   //assign clkSelect = CLOCK_50; // for simulation
   assign clk1 = div_clk[1]; // for enter clock
   assign clk2 = div_clk[24]; // for exit clock


   always_ff @(posedge clk1) begin
```

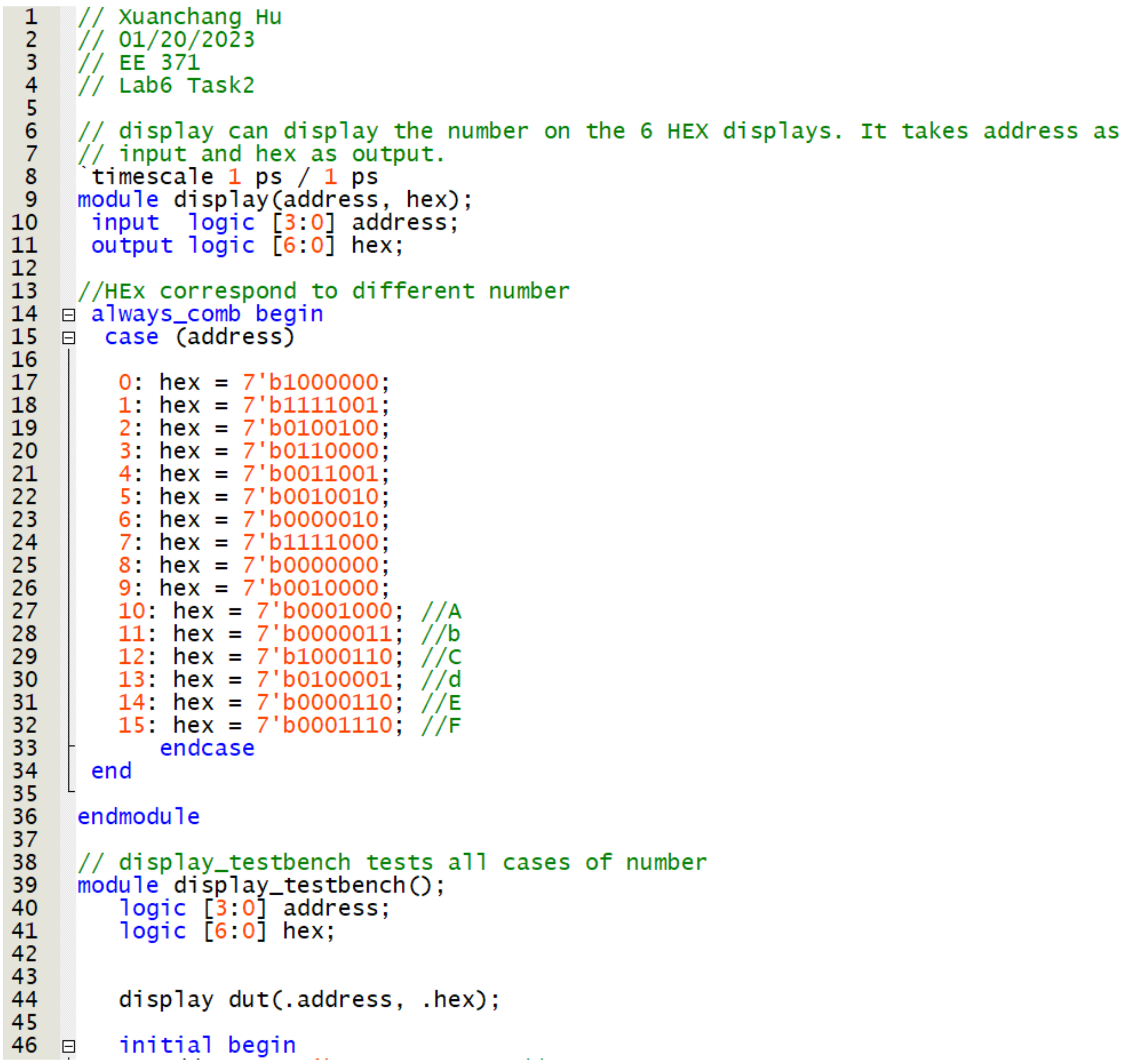


```
            if (V_GPIO[24] | LEDR[4] | V_GPIO[33]) begin
                outP <= 1;
            end else
                outP <= 0;
        end

    always_ff @(posedge clk2) begin
            if (V_GPIO[23] && V_GPIO[31] ) begin
                inP <= 1;
            end else
                inP <= 0;
        end

    // The main control of the program, it can receive enter, exit, and increase time operation orders.
    // And output HEX.
    control con(.clk(CLOCK_50), .reset(resetSW), .enter(inP), .exit(outP), .increaseTime(~KEY[0]),
                .num, .HEX0(HEX0), .HEX1(HEX1), .HEX2(HEX2), .HEX3(HEX3), .HEX4(HEX4), .HEX5(HEX5), .full(full));


endmodule  // DE1_SoC


// It tests cases of enter, exit cars. I also tests the case when parking is full and reset parking.
// HEX5 shows the hours during 8 hour work time. When thr number of cars are 3, HEX3 to 0 shows FULL.
// Otherwise, HEX0 shows the number of cars. After the end of work day, the HEX3 to 2 will shows
// the rushhour and endHour. HEX1 will show the address(Hour) and HEX0 will show the maximum cars.
module DE1_SoC_task2_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 enter, exit, reset;
    logic [1:0] num;
    logic full;
    logic increaseTime;

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

    DE1_SoC_task2 dut1(.CLOCK_50, .HEX0, .HEX1, .HEX2, .HEX3, .HEX4, .HEX5, .KEY, .SW, .LEDR, .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);
         reset <= 0; increaseTime <= 0; enter <= 0; exit <= 0;  @(posedge CLOCK_50);
```

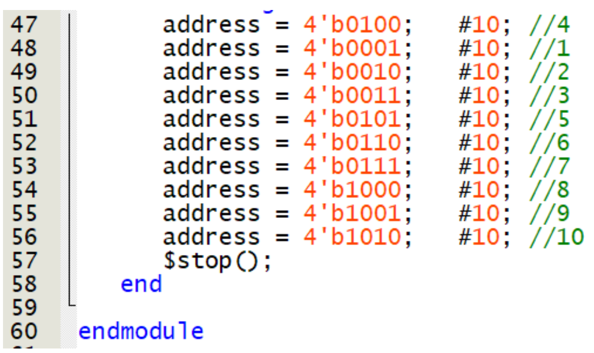


```
         increaseTime <= 0; enter <= 1; exit <= 0;  @(posedge CLOCK_50); // Hour 1
         increaseTime <= 0; enter <= 1; exit <= 0;  @(posedge CLOCK_50);
         increaseTime <= 0; enter <= 0; exit <= 0;  @(posedge CLOCK_50);
         increaseTime <= 1; enter <= 0; exit <= 0;  @(posedge CLOCK_50); // Hour 2
         increaseTime <= 0; enter <= 0; exit <= 1;  @(posedge CLOCK_50);
         increaseTime <= 0; enter <= 1; exit <= 0;  @(posedge CLOCK_50);
         increaseTime <= 0; enter <= 0; exit <= 1;  @(posedge CLOCK_50);
         increaseTime <= 0; enter <= 0; exit <= 0;  @(posedge CLOCK_50);
         increaseTime <= 1; enter <= 0; exit <= 0;  @(posedge CLOCK_50); // Hour 3
         increaseTime <= 0; enter <= 1; exit <= 0;  @(posedge CLOCK_50);
         increaseTime <= 0; enter <= 1; exit <= 0;  @(posedge CLOCK_50);
         increaseTime <= 0; enter <= 1; exit <= 0;  @(posedge CLOCK_50);
         increaseTime <= 0; enter <= 0; exit <= 0;  @(posedge CLOCK_50);
         increaseTime <= 1; enter <= 0; exit <= 0;  @(posedge CLOCK_50); // Hour 4
         increaseTime <= 0; enter <= 0; exit <= 1;  @(posedge CLOCK_50);
         increaseTime <= 0; enter <= 0; exit <= 1;  @(posedge CLOCK_50);
         increaseTime <= 1; enter <= 0; exit <= 0;  @(posedge CLOCK_50); // Hour 5
         increaseTime <= 0; enter <= 1; exit <= 0;  @(posedge CLOCK_50);
         increaseTime <= 0; enter <= 0; exit <= 1;  @(posedge CLOCK_50);
         increaseTime <= 1; enter <= 0; exit <= 0;  @(posedge CLOCK_50); // Hour 6
         increaseTime <= 0; enter <= 1; exit <= 0;  @(posedge CLOCK_50);
         increaseTime <= 0; enter <= 1; exit <= 0;  @(posedge CLOCK_50);
         increaseTime <= 0; enter <= 0; exit <= 1;  @(posedge CLOCK_50);
         increaseTime <= 0; enter <= 1; exit <= 0;  @(posedge CLOCK_50);
         increaseTime <= 1; enter <= 0; exit <= 0;  @(posedge CLOCK_50); // Hour 7
         increaseTime <= 0; enter <= 0; exit <= 1;  @(posedge CLOCK_50);
         increaseTime <= 0; enter <= 0; exit <= 1;  @(posedge CLOCK_50);
         increaseTime <= 0; enter <= 0; exit <= 1;  @(posedge CLOCK_50);
         increaseTime <= 0; enter <= 0; exit <= 1;  @(posedge CLOCK_50);
         increaseTime <= 1; enter <= 0; exit <= 0;  @(posedge CLOCK_50); // Hour 8
         increaseTime <= 0; enter <= 1; exit <= 0;  @(posedge CLOCK_50);
         increaseTime <= 0; enter <= 1; exit <= 0;  @(posedge CLOCK_50);
         increaseTime <= 1; enter <= 0; exit <= 0;  @(posedge CLOCK_50); // Hour display
         increaseTime <= 0; enter <= 0; exit <= 0;  @(posedge CLOCK_50);
         increaseTime <= 0; enter <= 0; exit <= 0;  @(posedge CLOCK_50);
         increaseTime <= 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) Display

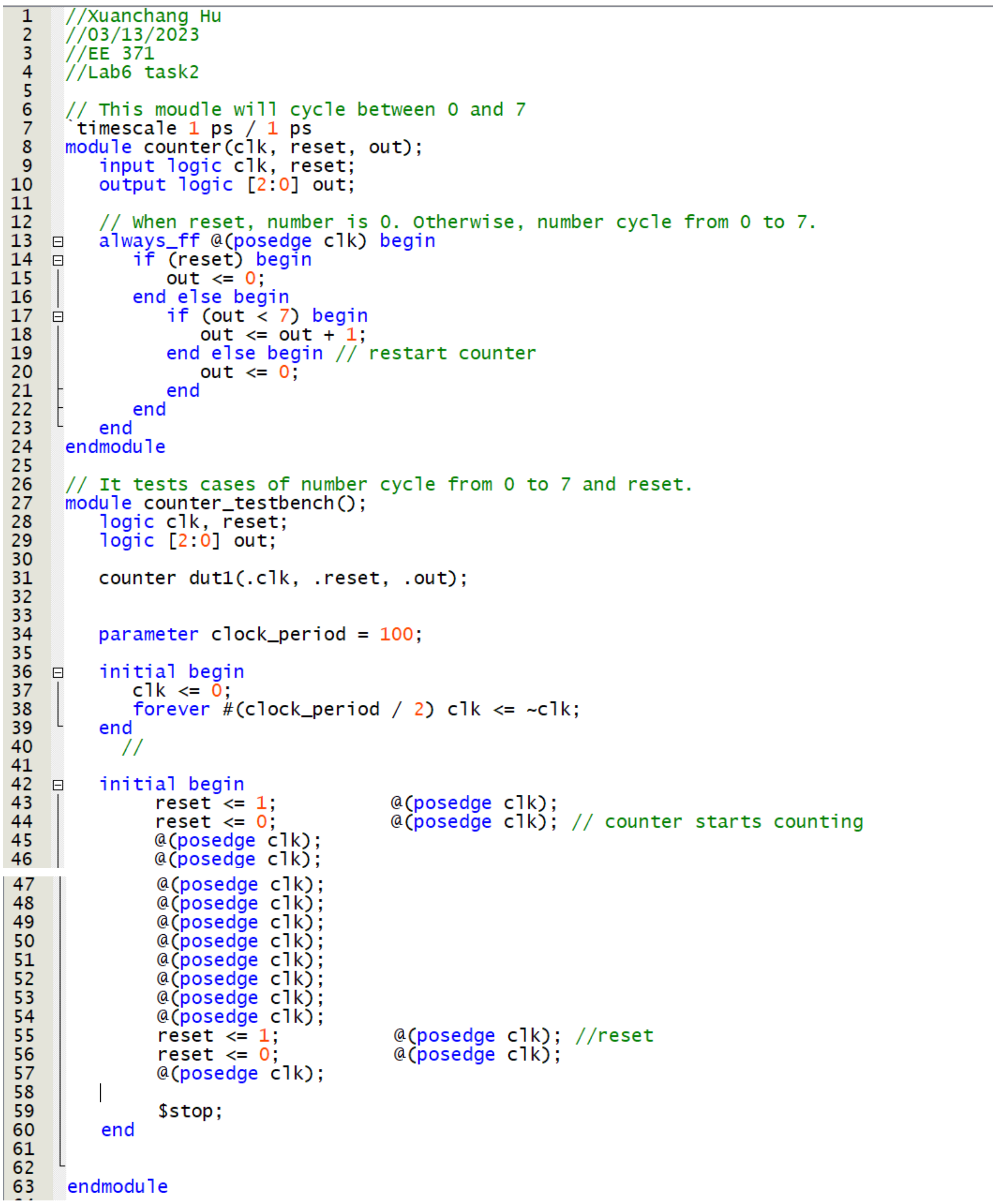


```
// Xuanchang Hu
// 01/20/2023
// EE 371
// Lab6 Task2

// display can display the number on the 6 HEX displays. It takes address as
// input and hex as output.
`timescale 1 ps / 1 ps
module display(address, hex);
 input  logic [3:0] address;
 output logic [6:0] hex;

//HEx correspond to different number
 always_comb begin
  case (address)

   0: hex = 7'b1000000;
   1: hex = 7'b1111001;
   2: hex = 7'b0100100;
   3: hex = 7'b0110000;
   4: hex = 7'b0011001;
   5: hex = 7'b0010010;
   6: hex = 7'b0000010;
   7: hex = 7'b1111000;
   8: hex = 7'b0000000;
   9: hex = 7'b0010000;
   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

// display_testbench tests all cases of number
module display_testbench();
   logic [3:0] address;
   logic [6:0] hex;


   display dut(.address, .hex);

   initial begin
```



```
      address = 4'b0100;   #10; //4
      address = 4'b0001;   #10; //1
      address = 4'b0010;   #10; //2
      address = 4'b0011;   #10; //3
      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
```

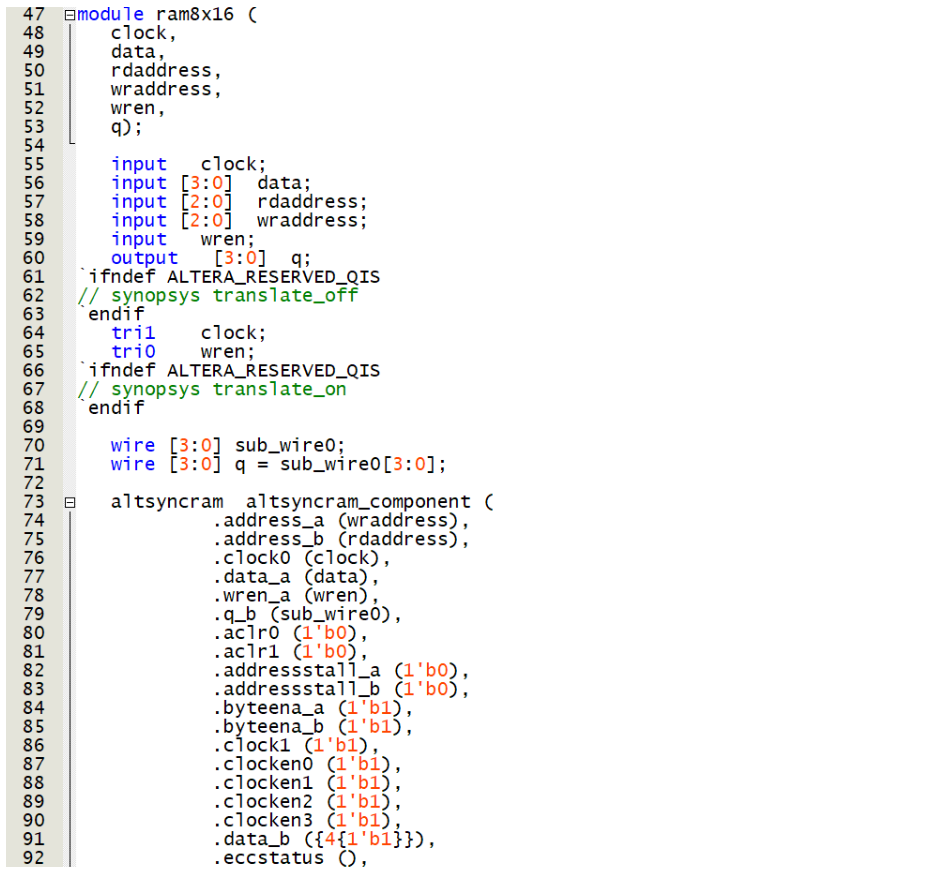
3) Counter



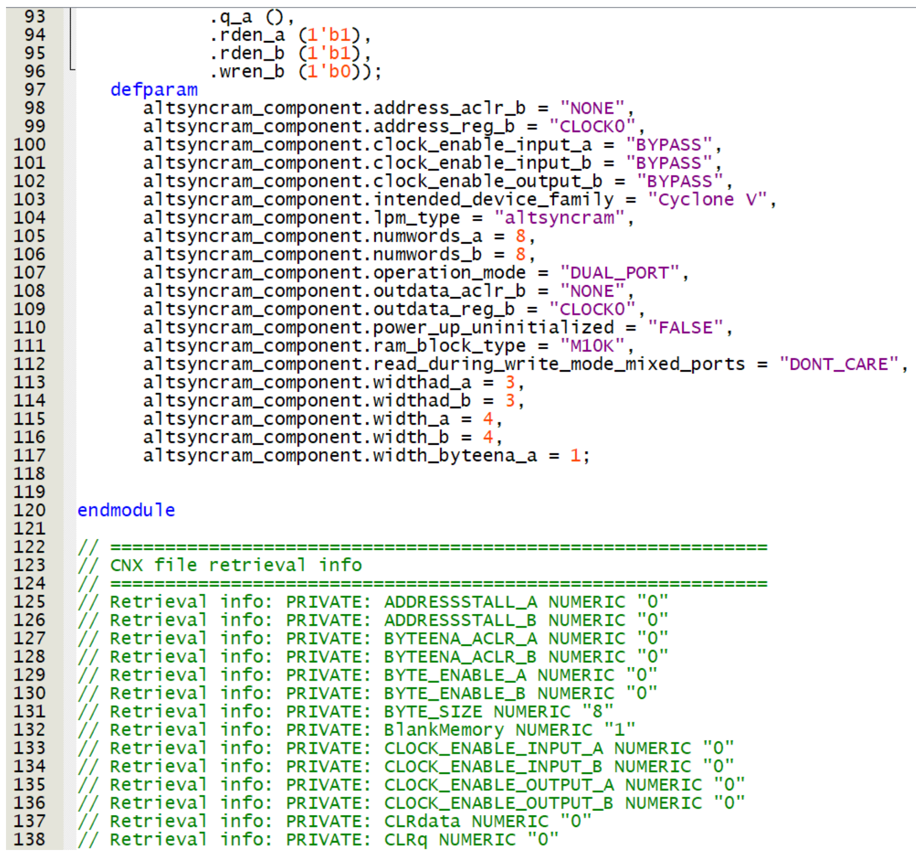
```
//Xuanchang Hu
//03/13/2023
//EE 371
//Lab6 task2

// This moudle will cycle between 0 and 7
`timescale 1 ps / 1 ps
module counter(clk, reset, out);
   input logic clk, reset;
   output logic [2:0] out;

   // When reset, number is 0. Otherwise, number cycle from 0 to 7.
   always_ff @(posedge clk) begin
      if (reset) begin
         out <= 0;
      end else begin
         if (out < 7) begin
            out <= out + 1;
         end else begin // restart counter
            out <= 0;
         end
      end
   end
endmodule

// It tests cases of number cycle from 0 to 7 and reset.
module counter_testbench();
   logic clk, reset;
   logic [2:0] out;

   counter dut1(.clk, .reset, .out);


   parameter clock_period = 100;

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

   initial begin
        reset <= 1;          @(posedge clk);
        reset <= 0;          @(posedge clk); // counter starts counting
        @(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
        reset <= 0;          @(posedge clk);
        @(posedge clk);

        $stop;
   end


endmodule
```

4) **Ram8x16**

```
//Xuanchang Hu
//03/13/2023
//EE 371
//Lab6 task2

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

// ============================================================
// File Name: ram8x16.v
// Megafunction Name(s):
//          altsyncram
//
// Simulation Library Files(s):
//          altera_mf
// ============================================================
// ************************************************************
// THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
//
// 17.0.0 Build 595 04/25/2017 SJ Lite Edition
// ************************************************************


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

// synopsys translate_off
`timescale 1 ps / 1 ps
// synopsys translate_on

//This module can store the data of maximum cars. The input address is the hour.
// read address is also the hour from counter. data and q are maximum number of cars
```

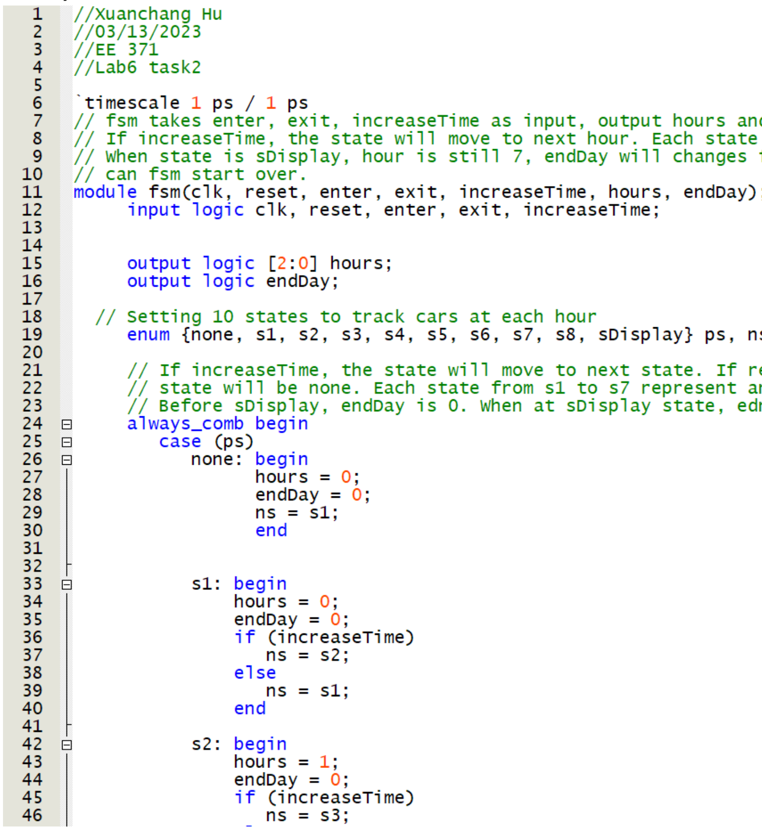


```
module ram8x16 (
    clock,
    data,
    rdaddress,
    wraddress,
    wren,
    q);

    input    clock;
    input [3:0]  data;
    input [2:0]  rdaddress;
    input [2:0]  wraddress;
    input    wren;
    output   [3:0]  q;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_off
`endif
    tri1     clock;
    tri0     wren;
`ifndef ALTERA_RESERVED_QIS
// synopsys translate_on
`endif

    wire [3:0] sub_wire0;
    wire [3:0] q = sub_wire0[3:0];

    altsyncram  altsyncram_component (
                .address_a (wraddress),
                .address_b (rdaddress),
                .clock0 (clock),
                .data_a (data),
                .wren_a (wren),
                .q_b (sub_wire0),
                .aclr0 (1'b0),
                .aclr1 (1'b0),
                .addressstall_a (1'b0),
                .addressstall_b (1'b0),
                .byteena_a (1'b1),
                .byteena_b (1'b1),
                .clock1 (1'b1),
                .clocken0 (1'b1),
                .clocken1 (1'b1),
                .clocken2 (1'b1),
                .clocken3 (1'b1),
                .data_b ({4{1'b1}}),
                .eccstatus (),
```

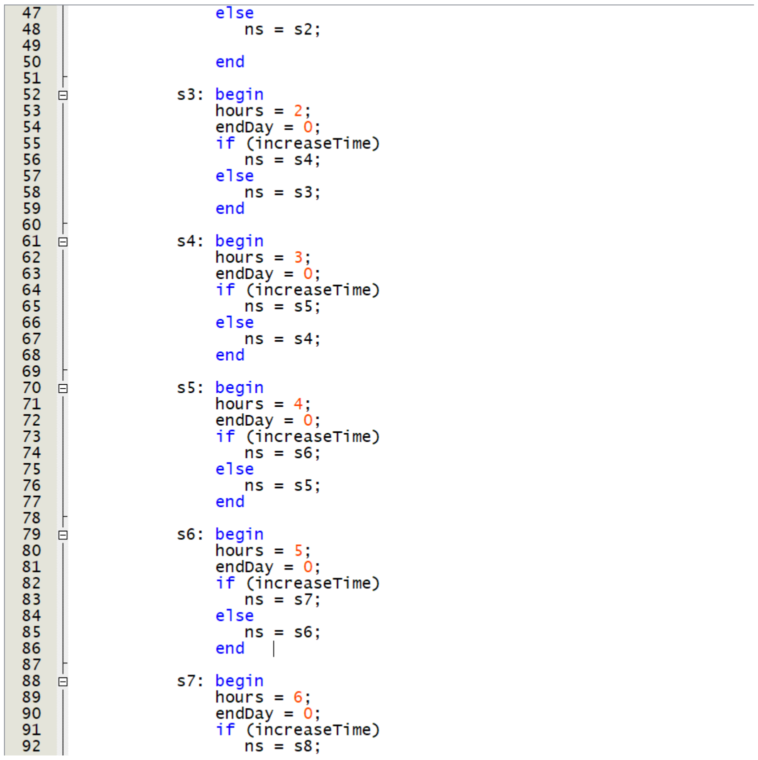


```
                .q_a (),
                .rden_a (1'b1),
                .rden_b (1'b1),
                .wren_b (1'b0));
    defparam
        altsyncram_component.address_aclr_b = "NONE",
        altsyncram_component.address_reg_b = "CLOCK0",
        altsyncram_component.clock_enable_input_a = "BYPASS",
        altsyncram_component.clock_enable_input_b = "BYPASS",
        altsyncram_component.clock_enable_output_b = "BYPASS",
        altsyncram_component.intended_device_family = "Cyclone V",
        altsyncram_component.lpm_type = "altsyncram",
        altsyncram_component.numwords_a = 8,
        altsyncram_component.numwords_b = 8,
        altsyncram_component.operation_mode = "DUAL_PORT",
        altsyncram_component.outdata_aclr_b = "NONE",
        altsyncram_component.outdata_reg_b = "CLOCK0",
        altsyncram_component.power_up_uninitialized = "FALSE",
        altsyncram_component.ram_block_type = "M10K",
        altsyncram_component.read_during_write_mode_mixed_ports = "DONT_CARE",
        altsyncram_component.widthad_a = 3,
        altsyncram_component.widthad_b = 3,
        altsyncram_component.width_a = 4,
        altsyncram_component.width_b = 4,
        altsyncram_component.width_byteena_a = 1;


endmodule

// ============================================================
// CNX file retrieval info
// ============================================================
// Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
// Retrieval info: PRIVATE: ADDRESSSTALL_B NUMERIC "0"
// Retrieval info: PRIVATE: BYTEENA_ACLR_A NUMERIC "0"
// Retrieval info: PRIVATE: BYTEENA_ACLR_B NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_ENABLE_A NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_ENABLE_B NUMERIC "0"
// Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
// Retrieval info: PRIVATE: BlankMemory NUMERIC "1"
// Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_B NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
// Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_B NUMERIC "0"
// Retrieval info: PRIVATE: CLRdata NUMERIC "0"
// Retrieval info: PRIVATE: CLRq NUMERIC "0"
```

5) Fsm

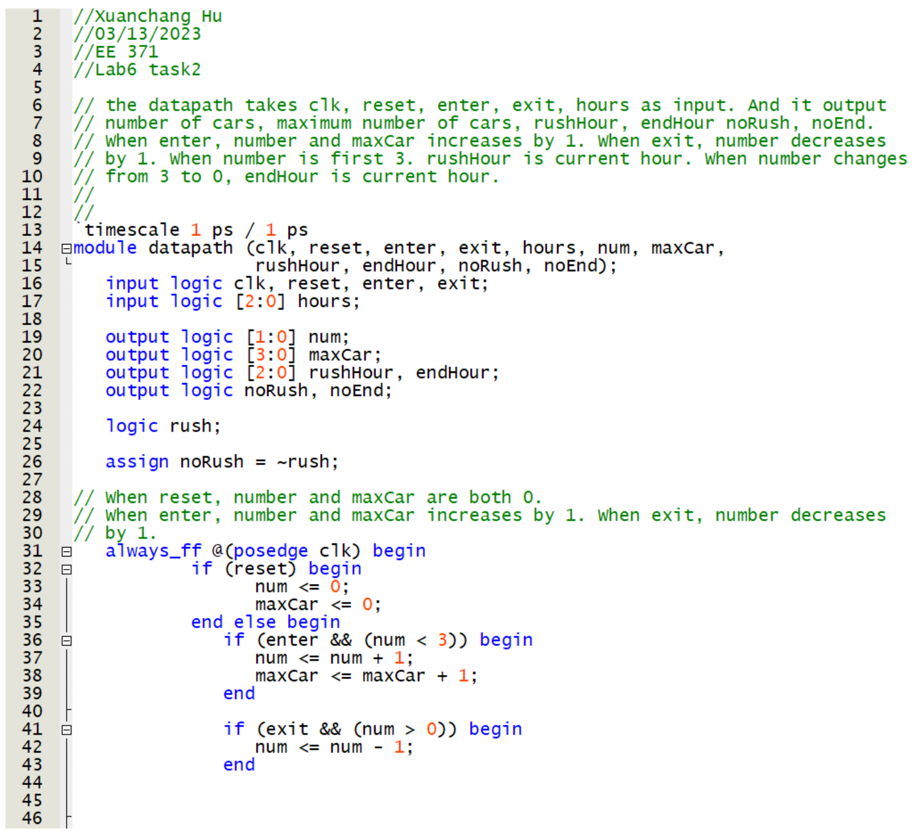


```
//Xuanchang Hu
//03/13/2023
//EE 371
//Lab6 task2

`timescale 1 ps / 1 ps
// fsm takes enter, exit, increaseTime as input, output hours and endDay.
// If increaseTime, the state will move to next hour. Each state from s1 to s7 represent an hour.
// When state is sDisplay, hour is still 7, endDay will changes from 0 to 1. Only reset
// can fsm start over.
module fsm(clk, reset, enter, exit, increaseTime, hours, endDay);
     input logic clk, reset, enter, exit, increaseTime;


     output logic [2:0] hours;
     output logic endDay;

  // Setting 10 states to track cars at each hour
     enum {none, s1, s2, s3, s4, s5, s6, s7, s8, sDisplay} ps, ns;

     // If increaseTime, the state will move to next state. If reset,
     // state will be none. Each state from s1 to s7 represent an hour
     // Before sDisplay, endDay is 0. When at sDisplay state, ednDisplay is 1.
     always_comb begin
        case (ps)
           none: begin
                 hours = 0;
                 endDay = 0;
                 ns = s1;
                 end


           s1: begin
               hours = 0;
               endDay = 0;
               if (increaseTime)
                  ns = s2;
               else
                  ns = s1;
               end

           s2: begin
               hours = 1;
               endDay = 0;
               if (increaseTime)
                  ns = s3;
```

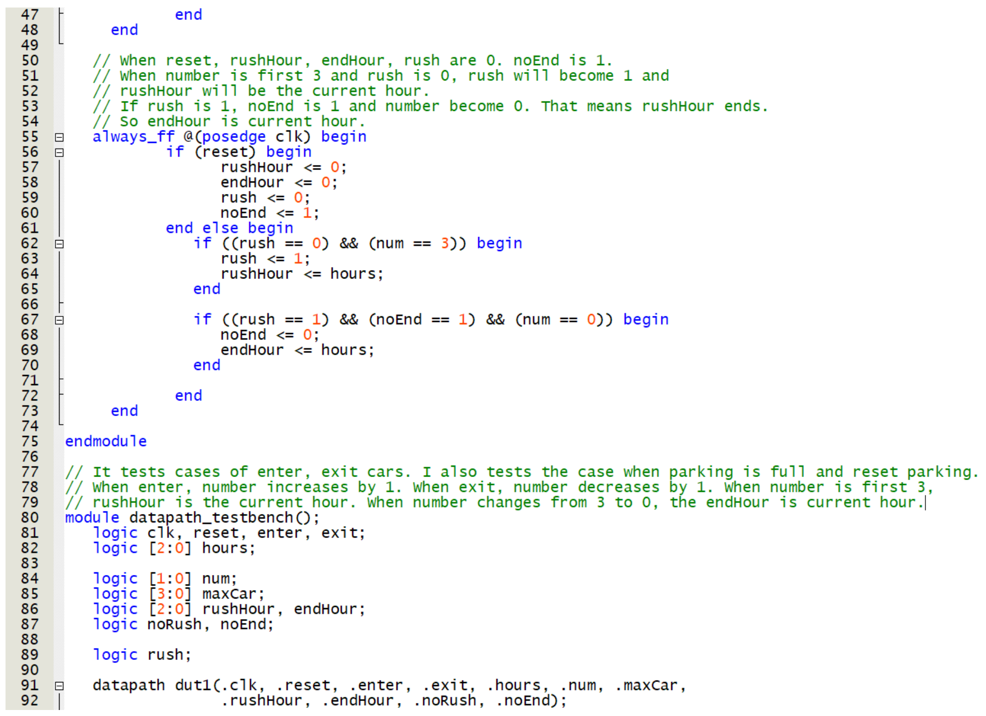


```
               else
                  ns = s2;

               end

           s3: begin
               hours = 2;
               endDay = 0;
               if (increaseTime)
                  ns = s4;
               else
                  ns = s3;
               end

           s4: begin
               hours = 3;
               endDay = 0;
               if (increaseTime)
                  ns = s5;
               else
                  ns = s4;
               end

           s5: begin
               hours = 4;
               endDay = 0;
               if (increaseTime)
                  ns = s6;
               else
                  ns = s5;
               end

           s6: begin
               hours = 5;
               endDay = 0;
               if (increaseTime)
                  ns = s7;
               else
                  ns = s6;
               end

           s7: begin
               hours = 6;
               endDay = 0;
               if (increaseTime)
                  ns = s8;
```

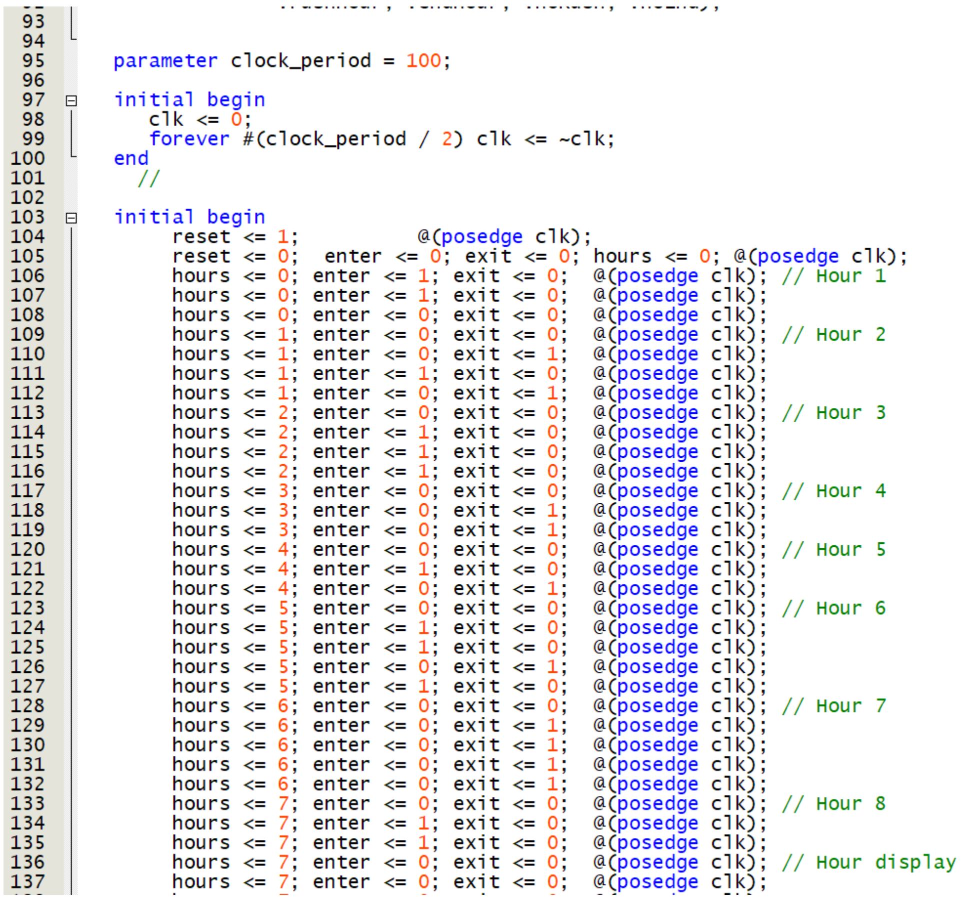
6) Datapath



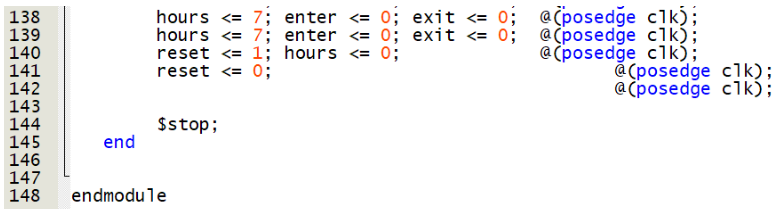
```
//Xuanchang Hu
//03/13/2023
//EE 371
//Lab6 task2

// the datapath takes clk, reset, enter, exit, hours as input. And it output
// number of cars, maximum number of cars, rushHour, endHour noRush, noEnd.
// When enter, number and maxCar increases by 1. When exit, number decreases
// by 1. When number is first 3. rushHour is current hour. When number changes
// from 3 to 0, endHour is current hour.
//
//
`timescale 1 ps / 1 ps
module datapath (clk, reset, enter, exit, hours, num, maxCar,
                 rushHour, endHour, noRush, noEnd);
   input logic clk, reset, enter, exit;
   input logic [2:0] hours;

   output logic [1:0] num;
   output logic [3:0] maxCar;
   output logic [2:0] rushHour, endHour;
   output logic noRush, noEnd;

   logic rush;

   assign noRush = ~rush;

// When reset, number and maxCar are both 0.
// When enter, number and maxCar increases by 1. When exit, number decreases
// by 1.
   always_ff @(posedge clk) begin
           if (reset) begin
                 num <= 0;
                 maxCar <= 0;
           end else begin
              if (enter && (num < 3)) begin
                 num <= num + 1;
                 maxCar <= maxCar + 1;
              end

              if (exit && (num > 0)) begin
                 num <= num - 1;
              end
```



```
           end
     end

   // When reset, rushHour, endHour, rush are 0. noEnd is 1.
   // When number is first 3 and rush is 0, rush will become 1 and
   // rushHour will be the current hour.
   // If rush is 1, noEnd is 1 and number become 0. That means rushHour ends.
   // So endHour is current hour.
   always_ff @(posedge clk) begin
           if (reset) begin
                 rushHour <= 0;
                 endHour <= 0;
                 rush <= 0;
                 noEnd <= 1;
           end else begin
              if ((rush == 0) && (num == 3)) begin
                 rush <= 1;
                 rushHour <= hours;
              end

              if ((rush == 1) && (noEnd == 1) && (num == 0)) begin
                 noEnd <= 0;
                 endHour <= hours;
              end

           end
     end

endmodule

// It tests cases of enter, exit cars. I also tests the case when parking is full and reset parking.
// When enter, number increases by 1. When exit, number decreases by 1. When number is first 3,
// rushHour is the current hour. When number changes from 3 to 0, the endHour is current hour.
module datapath_testbench();
   logic clk, reset, enter, exit;
   logic [2:0] hours;

   logic [1:0] num;
   logic [3:0] maxCar;
   logic [2:0] rushHour, endHour;
   logic noRush, noEnd;

   logic rush;

   datapath dut1(.clk, .reset, .enter, .exit, .hours, .num, .maxCar,
                 .rushHour, .endHour, .noRush, .noEnd);
```



```verilog

   parameter clock_period = 100;

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

   initial begin
        reset <= 1;          @(posedge clk);
        reset <= 0;  enter <= 0; exit <= 0; hours <= 0; @(posedge clk);
        hours <= 0; enter <= 1; exit <= 0;  @(posedge clk); // Hour 1
        hours <= 0; enter <= 1; exit <= 0;  @(posedge clk);
        hours <= 0; enter <= 0; exit <= 0;  @(posedge clk);
        hours <= 1; enter <= 0; exit <= 0;  @(posedge clk); // Hour 2
        hours <= 1; enter <= 0; exit <= 1;  @(posedge clk);
        hours <= 1; enter <= 1; exit <= 0;  @(posedge clk);
        hours <= 1; enter <= 0; exit <= 1;  @(posedge clk);
        hours <= 2; enter <= 0; exit <= 0;  @(posedge clk); // Hour 3
        hours <= 2; enter <= 1; exit <= 0;  @(posedge clk);
        hours <= 2; enter <= 1; exit <= 0;  @(posedge clk);
        hours <= 2; enter <= 1; exit <= 0;  @(posedge clk);
        hours <= 3; enter <= 0; exit <= 0;  @(posedge clk); // Hour 4
        hours <= 3; enter <= 0; exit <= 1;  @(posedge clk);
        hours <= 3; enter <= 0; exit <= 1;  @(posedge clk);
        hours <= 4; enter <= 0; exit <= 0;  @(posedge clk); // Hour 5
        hours <= 4; enter <= 1; exit <= 0;  @(posedge clk);
        hours <= 4; enter <= 0; exit <= 1;  @(posedge clk);
        hours <= 5; enter <= 0; exit <= 0;  @(posedge clk); // Hour 6
        hours <= 5; enter <= 1; exit <= 0;  @(posedge clk);
        hours <= 5; enter <= 1; exit <= 0;  @(posedge clk);
        hours <= 5; enter <= 0; exit <= 1;  @(posedge clk);
        hours <= 5; enter <= 1; exit <= 0;  @(posedge clk);
        hours <= 6; enter <= 0; exit <= 0;  @(posedge clk); // Hour 7
        hours <= 6; enter <= 0; exit <= 1;  @(posedge clk);
        hours <= 6; enter <= 0; exit <= 1;  @(posedge clk);
        hours <= 6; enter <= 0; exit <= 1;  @(posedge clk);
        hours <= 6; enter <= 0; exit <= 1;  @(posedge clk);
        hours <= 7; enter <= 0; exit <= 0;  @(posedge clk); // Hour 8
        hours <= 7; enter <= 1; exit <= 0;  @(posedge clk);
        hours <= 7; enter <= 1; exit <= 0;  @(posedge clk);
        hours <= 7; enter <= 0; exit <= 0;  @(posedge clk); // Hour display
        hours <= 7; enter <= 0; exit <= 0;  @(posedge clk);
```

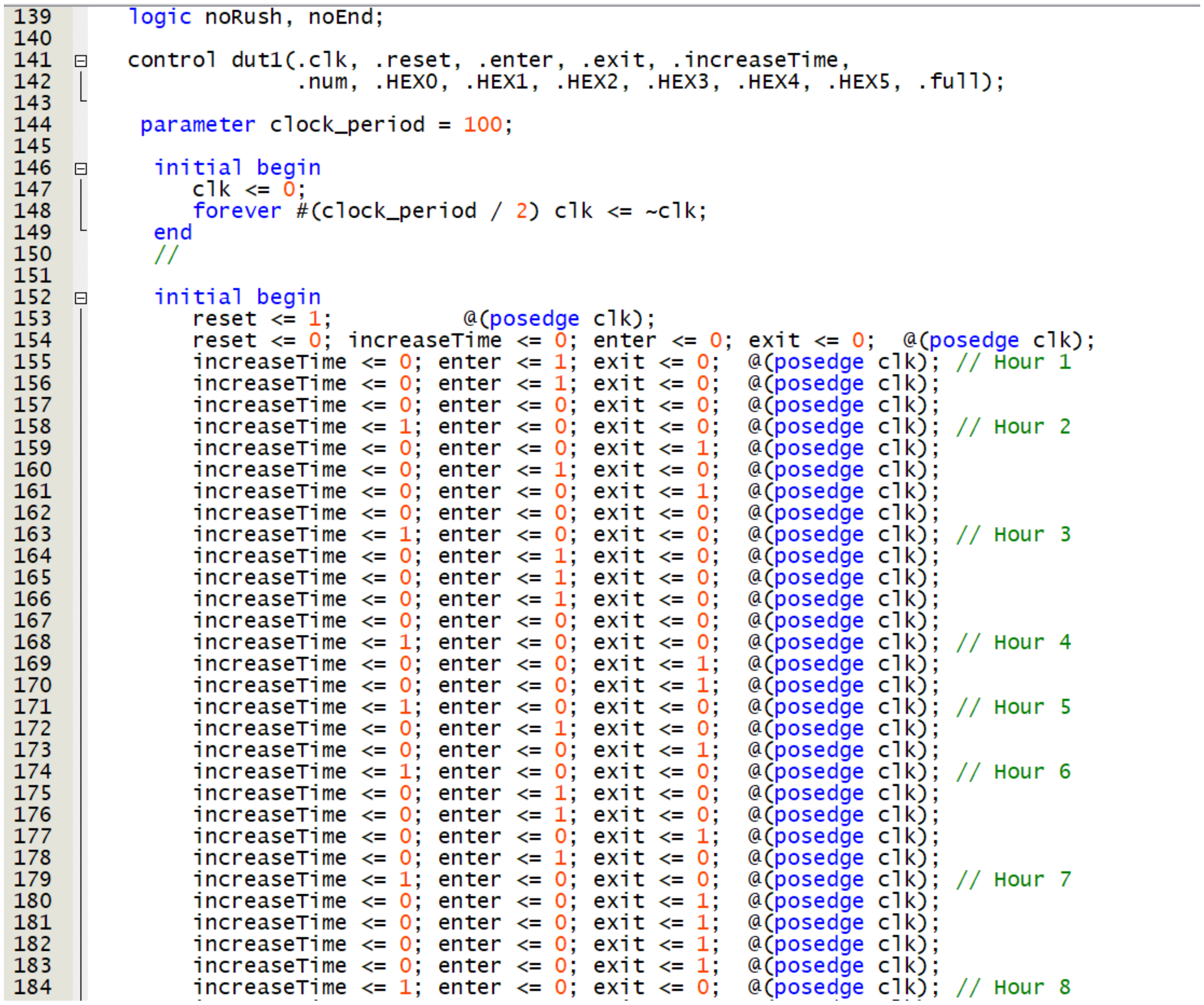


```verilog
        hours <= 7; enter <= 0; exit <= 0;  @(posedge clk);
        hours <= 7; enter <= 0; exit <= 0;  @(posedge clk);
        reset <= 1; hours <= 0;             @(posedge clk);
        reset <= 0;                                @(posedge clk);
                                                   @(posedge clk);

        $stop;
   end

endmodule
```

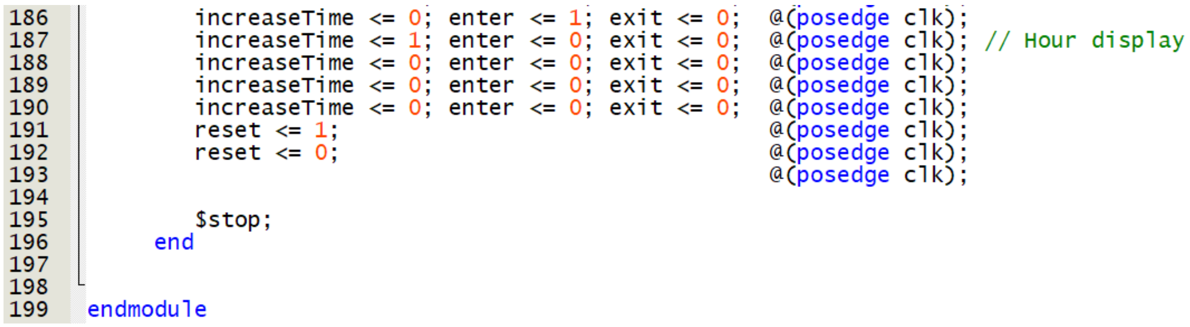
7)Control

```
//Xuanchang Hu
//03/13/2023
//EE 371
//Lab6 task2

`timescale 1 ps / 1 ps
// This module is to control the parking counter. It takes clk, reset, enter, exit
// increaseTime as input, output HEX anb whether parking full. It combines all the other submodules
// to implement the parking system.
module control (clk, reset, enter, exit, increaseTime,
                num, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, full);

   input logic clk, reset, enter, exit, increaseTime;

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

   logic [2:0] hours;
   logic [3:0] maxCar;
   logic [2:0] rushHour, endHour;
   logic [2:0] out;
   logic endDay;

   logic [3:0] maxOut;
   logic [6:0] cars, dayTime, outNum, carNum, rushTime, endTime;
   logic noRush, noEnd;

   assign full = (num == 3);

   // Clock divider setup
   logic [31:0] div_clk;

   // The clock divider can decreases the clock speed in simulation.
   //It takes clk as input and output divider clock.
   clock_divider clkdivide (.clock(clk), .divided_clocks(div_clk));

   logic clkSelect;
   parameter whichClock = 25; // 0.75 Hz clock

   assign clkSelect = clk; // for simulation
   //assign clkSelect = div_clk[whichClock]; // for board



   // finite state machines, it can output hours to show time
   fsm statemachine(.clk(clkSelect), .reset, .enter, .exit, .increaseTime, .hours, .endDay);


   // datapath can output number of cars in parking, also output rush hour, end of rush hour.
   datapath data(.clk(clkSelect), .reset, .enter, .exit, .hours, .num, .maxCar, .rushHour, .endHour,
                 .noRush, .noEnd);


   // counter that cycle between 0-7
   counter cycle(.clk(clkSelect), .reset, .out);

   // It can stores the max value of cars at each hour
   ram8x16 ram(.clock(clkSelect), .data(maxCar), .rdaddress(out), .wraddress(hours), .wren(1'b1), .q(maxOut));


   display h5(.address({1'b0, hours}), .hex(dayTime)); // HEX that show daytime
   display ca(.address({2'b0, num}), .hex(cars));       // HEX that show number of cars
   display rushTim(.address({1'b0, rushHour}), .hex(rushTime)); // HEX that show rush time
   display endTim(.address({1'b0, endHour}), .hex(endTime)); // HEX that show end of rush time
   display outNu(.address({1'b0, out}), .hex(outNum)); // HEX that show address counter number
   display carNu(.address(maxOut), .hex(carNum)); // HEX that show max number of cars in ram

// It can change HEx output based on different conditions. When in the work time(endDay == 0)
// HEX5 shows the hours during 8 hour work time. When thr number of cars are 3, HEX3 to 0 shows FULL.
// Otherwise, HEX0 shows the number of cars. After the end of work day(endDay == 1),
// the HEX3 to 2 will shows the rushhour and endHour. HEX1 will show the address(Hour)
// and HEX0 will show the maximum cars.
   always_comb begin
      if (endDay == 0) begin //If day not end
         if (num == 3) begin
            HEX5 = dayTime;     //hour
            HEX4 = 7'b1111111; //blank
            HEX3 = 7'b0001110; // F
            HEX2 = 7'b1000001; // U
            HEX1 = 7'b1000111; // L
            HEX0 = 7'b1000111; // L
         end else begin
            HEX5 = dayTime;    //hour
            HEX4 = 7'b1111111;
            HEX3 = 7'b1111111;
            HEX2 = 7'b1111111;
            HEX1 = 7'b1111111;
            HEX0 = cars;       //number of cars
         end
      end else begin          //If day ends
         if (noRush) begin
```

```
            HEX4 = 7'b0111111; //show rush time
         end else begin
            HEX4 = rushTime;
         end

         if (noEnd) begin        //show end of rush time
            HEX3 = 7'b0111111;
         end else begin
            HEX3 = endTime;
         end

         HEX2 = outNum;          // Show address
         HEX1 = carNum;          //show maximum number of cars
         HEX5 = 7'b1111111;
         HEX0 = 7'b1111111;
      end

   end
   //





endmodule

// It tests cases of enter, exit cars. I also tests the case when parking is full and reset parking.
// HEX5 shows the hours during 8 hour work time. When the number of cars are 3, HEX3 to 0 shows FULL.
// Otherwise, HEX0 shows the number of cars. After the end of work day, the HEX3 to 2 will shows
// the rushhour and endHour. HEX1 will show the address(Hour) and HEX0 will show the maximum cars.
module control_testbench();
   logic clk, reset, enter, exit, increaseTime;

   logic [1:0] num;
   logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
   logic full;

   //just logics
   logic [2:0] hours;
   logic [6:0] maxCar;
   logic [2:0] rushHour, endHour;
   logic [2:0] out;
   logic endDay;

   logic [3:0] maxOut;
   logic [6:0] cars, dayTime, outNum, carNum, rushTime, endTime;
```



```
   logic noRush, noEnd;

   control dut1(.clk, .reset, .enter, .exit, .increaseTime,
                .num, .HEX0, .HEX1, .HEX2, .HEX3, .HEX4, .HEX5, .full);

    parameter clock_period = 100;

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

     initial begin
        reset <= 1;           @(posedge clk);
        reset <= 0; increaseTime <= 0; enter <= 0; exit <= 0;  @(posedge clk);
        increaseTime <= 0; enter <= 1; exit <= 0;  @(posedge clk); // Hour 1
        increaseTime <= 0; enter <= 1; exit <= 0;  @(posedge clk);
        increaseTime <= 0; enter <= 0; exit <= 0;  @(posedge clk);
        increaseTime <= 1; enter <= 0; exit <= 0;  @(posedge clk); // Hour 2
        increaseTime <= 0; enter <= 0; exit <= 1;  @(posedge clk);
        increaseTime <= 0; enter <= 1; exit <= 0;  @(posedge clk);
        increaseTime <= 0; enter <= 0; exit <= 1;  @(posedge clk);
        increaseTime <= 0; enter <= 0; exit <= 0;  @(posedge clk);
        increaseTime <= 1; enter <= 0; exit <= 0;  @(posedge clk); // Hour 3
        increaseTime <= 0; enter <= 1; exit <= 0;  @(posedge clk);
        increaseTime <= 0; enter <= 1; exit <= 0;  @(posedge clk);
        increaseTime <= 0; enter <= 1; exit <= 0;  @(posedge clk);
        increaseTime <= 0; enter <= 0; exit <= 0;  @(posedge clk);
        increaseTime <= 1; enter <= 0; exit <= 0;  @(posedge clk); // Hour 4
        increaseTime <= 0; enter <= 0; exit <= 1;  @(posedge clk);
        increaseTime <= 0; enter <= 0; exit <= 1;  @(posedge clk);
        increaseTime <= 1; enter <= 0; exit <= 0;  @(posedge clk); // Hour 5
        increaseTime <= 0; enter <= 1; exit <= 0;  @(posedge clk);
        increaseTime <= 0; enter <= 0; exit <= 1;  @(posedge clk);
        increaseTime <= 1; enter <= 0; exit <= 0;  @(posedge clk); // Hour 6
        increaseTime <= 0; enter <= 1; exit <= 0;  @(posedge clk);
        increaseTime <= 0; enter <= 1; exit <= 0;  @(posedge clk);
        increaseTime <= 0; enter <= 0; exit <= 1;  @(posedge clk);
        increaseTime <= 0; enter <= 1; exit <= 0;  @(posedge clk);
        increaseTime <= 1; enter <= 0; exit <= 0;  @(posedge clk); // Hour 7
        increaseTime <= 0; enter <= 0; exit <= 1;  @(posedge clk);
        increaseTime <= 0; enter <= 0; exit <= 1;  @(posedge clk);
        increaseTime <= 0; enter <= 0; exit <= 1;  @(posedge clk);
        increaseTime <= 0; enter <= 0; exit <= 1;  @(posedge clk);
        increaseTime <= 1; enter <= 0; exit <= 0;  @(posedge clk); // Hour 8
```



```
        increaseTime <= 0; enter <= 1; exit <= 0;  @(posedge clk);
        increaseTime <= 1; enter <= 0; exit <= 0;  @(posedge clk); // Hour display
        increaseTime <= 0; enter <= 0; exit <= 0;  @(posedge clk);
        increaseTime <= 0; enter <= 0; exit <= 0;  @(posedge clk);
        increaseTime <= 0; enter <= 0; exit <= 0;  @(posedge clk);
        reset <= 1;                                @(posedge clk);
        reset <= 0;                                @(posedge clk);
                                                   @(posedge clk);


        $stop;
     end

endmodule
```

8) clock_divider

```
// Xuanchang Hu
// 01/20/2023
// EE 371
// Lab6 Task 2

// divided_clocks[0] = 25MHz, [1] = 12.5Mhz, ... [23] = 3Hz, [24] = 1.5Hz, [25] = 0.75Hz, ...
// It takes clock as input and divided_clocks as output. it can divide clock into
// different frequency.
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
```