

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.

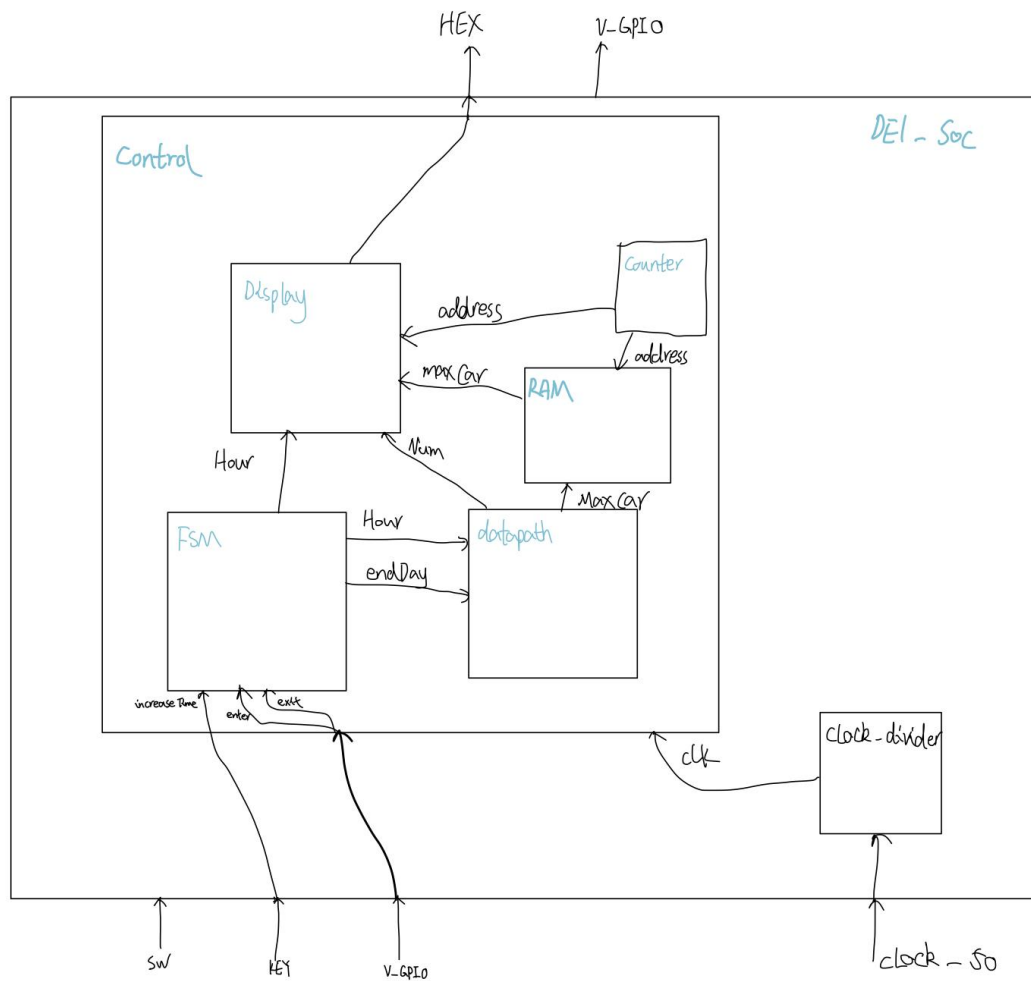


Figure 1:Block diagram of Lab 2 task 2

Task 2
ASMD

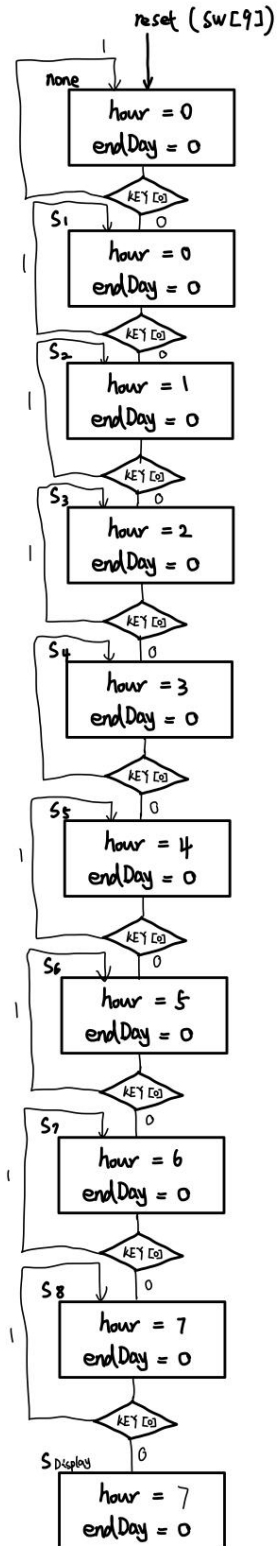


Figure 2:ASMD of Lab 2 task 2

FSM

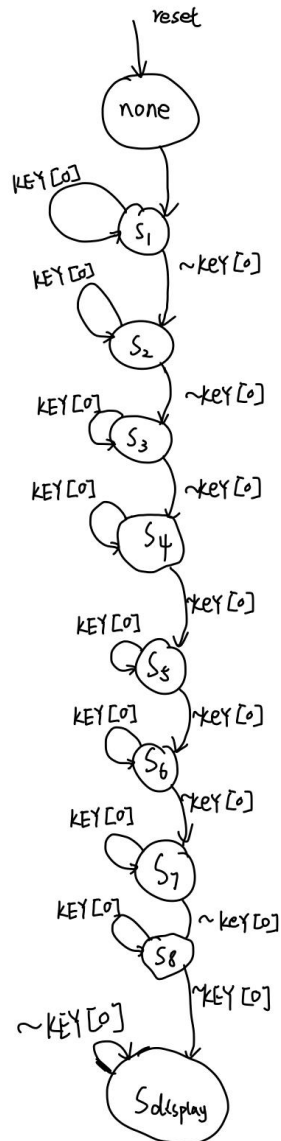


Figure 3:fsm of Lab 2 task 2

Results
Task 2

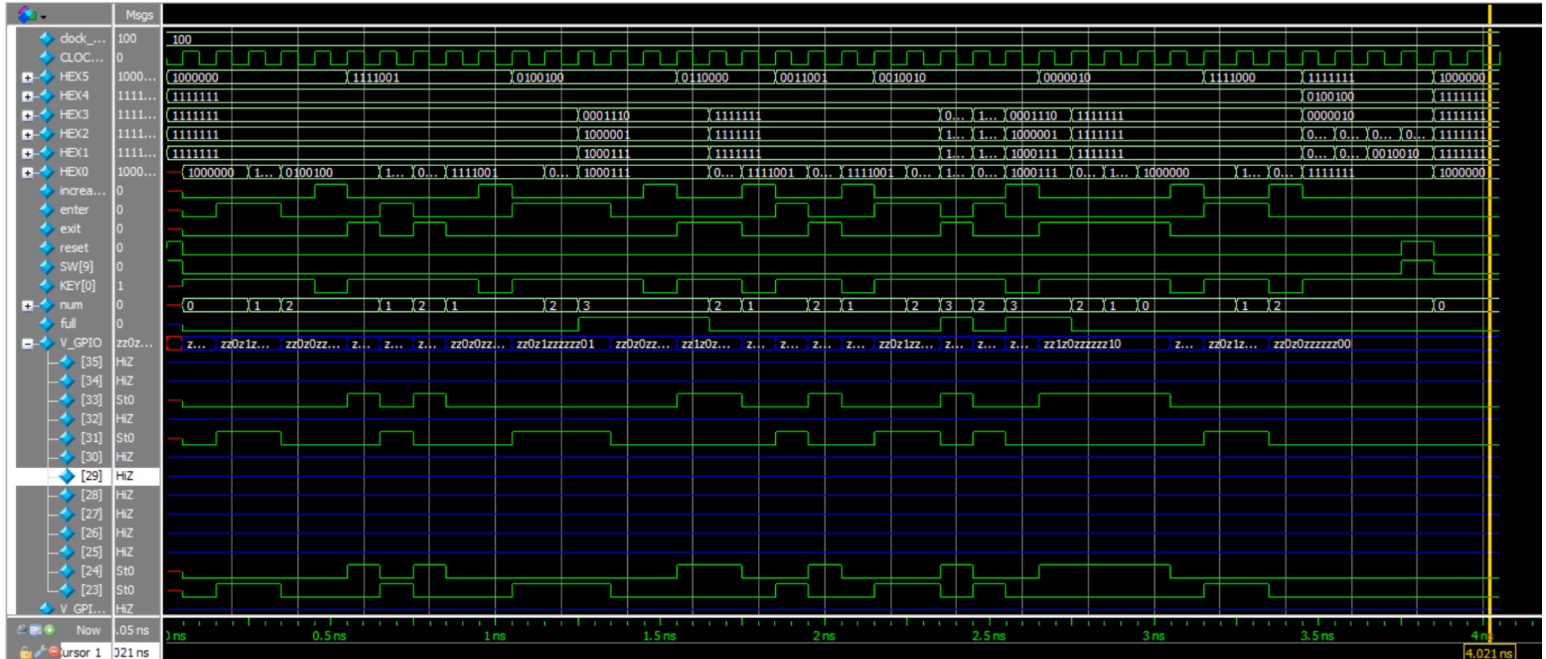


Figure 4: The waveform generated by the DE1_SoC_task2

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 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 show the rush hour and endHour. HEX1 will show the address(Hour) and HEX0 will show the maximum cars.

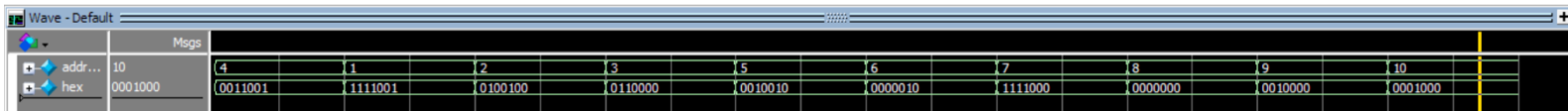


Figure 5: The waveform generated by the display

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

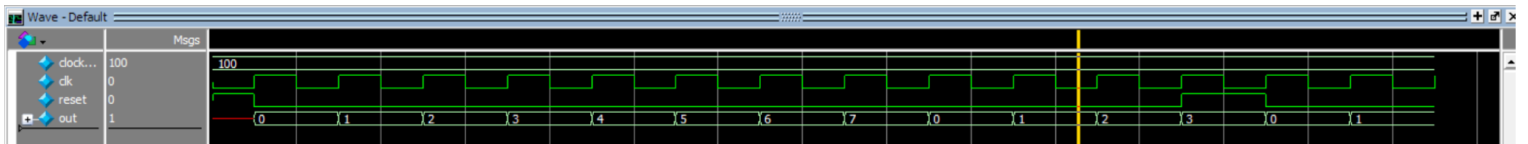


Figure 6: The waveform generated by counter

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

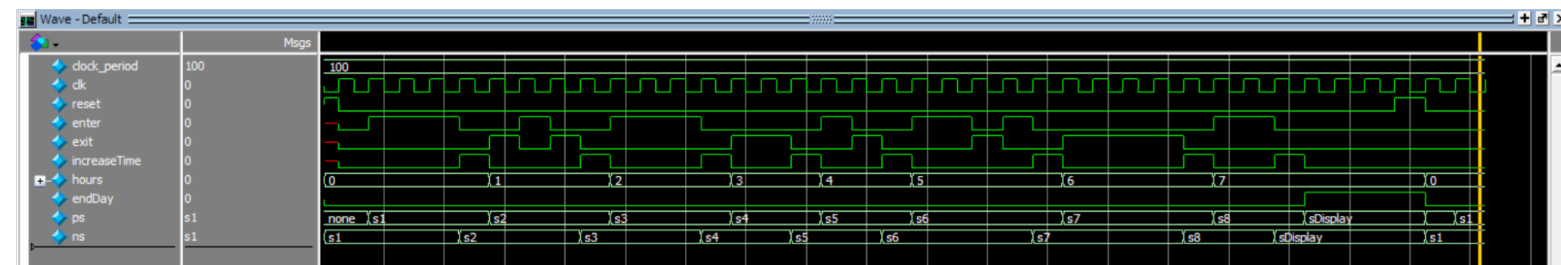


Figure 7: The waveform generated by fsm

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 outputted. 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.

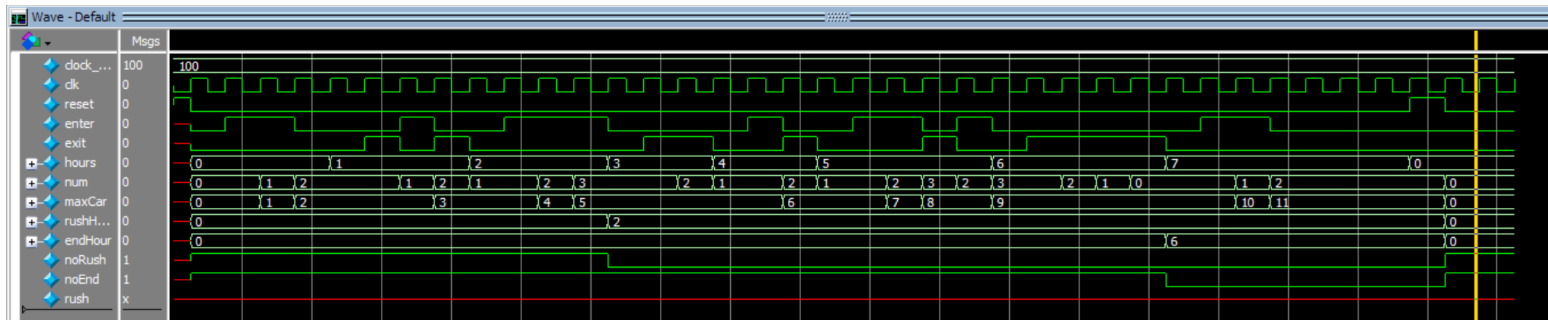


Figure 8: The waveform generated by datapath

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.

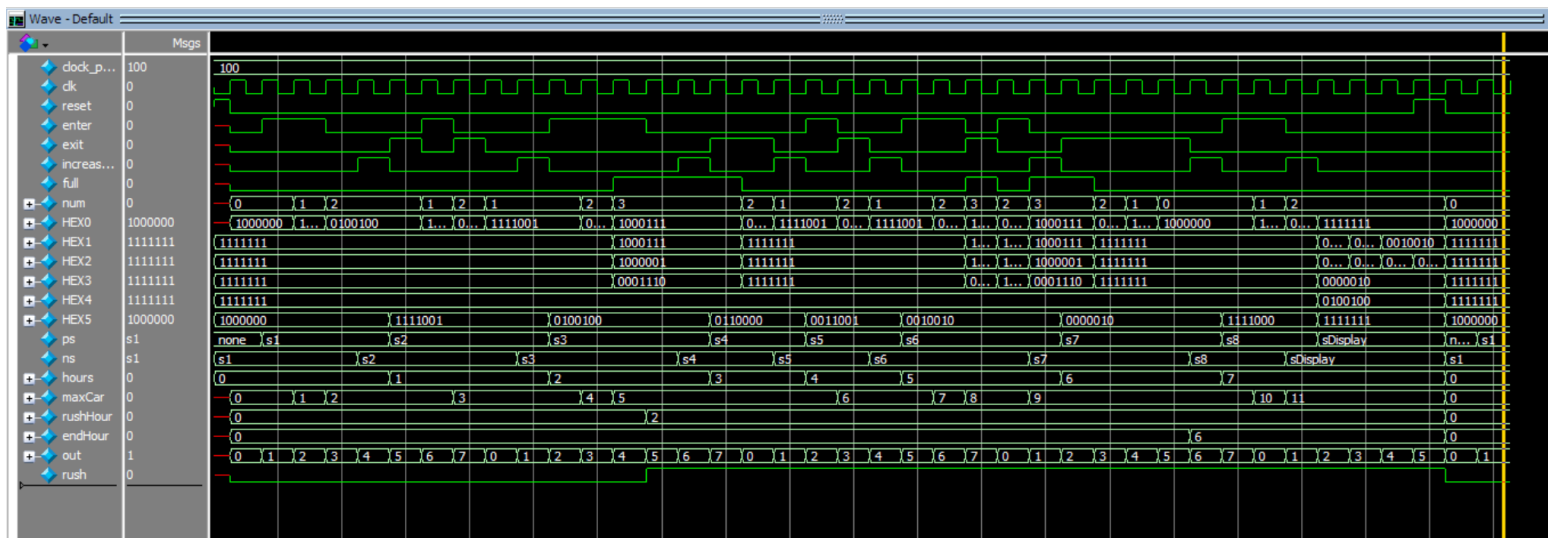


Figure 9: The waveform generated by control

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

```

1 //Xuanchang Hu
2 //01/13/2023
3 //EE 371
4 //Lab 6
5
6 // DE1_SoC takes a clock and a 34-bits GPIO_0 as input, and 7-bits HEX0,
7 // HEX1, HEX2, HEX3, HEX4, HEX5 as output. The GPIO_0[10], GPIO_0[12], GPIO_0[14] serves for switches
8 // and GPIO_0[26] and GPIO_0[27] serves for leds. This module is the top-level for the parking lot
9 // system to implement.
10 // Top-level module that defines the I/Os for the DE-1 SoC board
11
12 module DE1_SoC_task1(CLOCK_50, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, V_GPIO);
13     input logic CLOCK_50;
14     inout logic [35:23] V_GPIO;
15     output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
16
17     logic leftSW, rightSW, resetSW;
18
19     assign resetSW = V_GPIO[23]; //The leftmost switch is reset A
20     assign leftSW = V_GPIO[24]; //The middle switch among the 3 is connected to pin 15
21     assign rightSW = V_GPIO[28]; //The right switch among the 3 is connected to pin 17
22     assign V_GPIO[32] = V_GPIO[24]; // left led is connected to pin 15
23     assign V_GPIO[33] = V_GPIO[28]; // right led is connected to pin 17
24
25     logic enter, exit;
26     logic [1:0] num;
27     logic [1:0] units, tens;
28
29     // fsm cars takes CLOCK_50 as clk input, and resetSW as reset input, leftSW, rightSW as sensor a and b input.
30     // and returns if cars enters or exit parking lot as enter and exit.
31     fsm cars(CLOCK_50, .reset(resetSW), .a(leftSW), .b(rightSW), .enter, .exit);
32
33     // counter number takes CLOCK_50 as clk input, and resetSW as reset input, and enter and exit as inc and dec
34     // it returns num as number of cars, tens as the tens digit of num, units as units digit of num.
35     counter number(CLOCK_50, .reset(resetSW), .inc(enter), .dec(exit), .num, .tens, .units);
36
37     // light display takes num as car number inputs, tens as the tens digit of num input,
38     // units as units digit of num input.
39     // it returns HEX from 5 to 0 to the display
40     light display(.hex5(HEX5), .hex4(HEX4), .hex3(HEX3), .hex2(HEX2), .hex1(HEX1), .hex0(HEX0), .num, .tens, .units);
41
42
43 endmodule

```

Task 2

1) DE1_SoC_task2

```

1 //Xuanchang Hu
2 //03/13/2023
3 //EE 371
4 //Lab6 task2
5 timescale 1 ps / 1 ps
6
7 // This module is the top level module of parking system. It takes CLOCK_50, HEX, KEY, SW as input.
8 // LEDR as output. And V_GPIO for both input and output. It can make the parking system open gate, count
9 // show the time, number of cars, rushHour, endHour, RAM address and maximum number of cars on HEX.
10 // It can also show whether the parking is full or not.
11 module DE1_SoC_task2 (CLOCK_50, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, KEY, SW, LEDR, V_GPIO);
12
13 // define ports
14 input logic CLOCK_50;
15 output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
16 input logic [3:0] KEY;
17 input logic [9:0] SW;
18 output logic [9:0] LEDR;
19 inout logic [35:23] V_GPIO;
20
21 logic inP, outP, resetSW; // into parking, exit parking, reset switch
22 logic [1:0] num; //Number
23 logic full; //full
24 logic inc; //increase
25
26
27 assign resetSW = SW[9]; // reset switch is SW[9]
28
29
30 // FPGA output
31 assign V_GPIO[26] = V_GPIO[28]; // LED parking 1 && V_GPIO[28]
32 assign V_GPIO[27] = V_GPIO[29]; // LED parking 2 && V_GPIO[29]
33 assign V_GPIO[32] = V_GPIO[30]; // LED parking 3 && V_GPIO[30]
34 assign V_GPIO[34] = V_GPIO[28] & V_GPIO[29] & V_GPIO[30]; // LED that show parking is full
35 assign V_GPIO[31] = V_GPIO[23]; // Open entrance
36 assign V_GPIO[33] = V_GPIO[24]; // Open exit
37
38 // FPGA input
39 assign LEDR[0] = V_GPIO[28]; // Presence parking 1
40 assign LEDR[1] = V_GPIO[29]; // Presence parking 2
41 assign LEDR[2] = V_GPIO[30]; // Presence parking 3
42 assign LEDR[3] = V_GPIO[23]; // Presence entrance
43 assign LEDR[4] = V_GPIO[24]; // Presence exit
44
45 logic [31:0] div_clk;
46
47 cLock_divider clkdivide (.clock(CLOCK_50), .divided_clocks(div_clk));
48
49 logic clk1, clk2;
50 parameter whichClock = 26; // 0.75 Hz clock
51
52 //assign clkSelect = CLOCK_50; // for simulation
53 assign clk1 = div_clk[1]; // for enter clock
54 assign clk2 = div_clk[24]; // for exit clock
55
56
57 always_ff @(posedge clk1) begin

```



```

58     if (V_GPIO[24] | LEDR[4] | V_GPIO[33]) begin
59         outP <= 1;
60     end else
61         outP <= 0;
62     end
63
64     always_ff @(posedge clk2) begin
65         if (V_GPIO[23] && V_GPIO[31] ) begin
66             inP <= 1;
67         end else
68             inP <= 0;
69         end
70
71         // The main control of the program, it can receive enter, exit, and increase time operation orders.
72         // And output HEX.
73         control con(.clk(CLOCK_50), .reset(resetSW), .enter(inP), .exit(outP), .increaseTime(~KEY[0]),
74             .num, .HEX0(HEX0), .HEX1(HEX1), .HEX2(HEX2), .HEX3(HEX3), .HEX4(HEX4), .HEX5(HEX5), .full(full));
75
76     endmodule // DE1_Soc
77
78
79
80     // It tests cases of enter, exit cars. I also tests the case when parking is full and reset parking.
81     // HEX5 shows the hours during 8 hour work time. When thr number of cars are 3, HEX3 to 0 shows FULL.
82     // Otherwise, HEX0 shows the number of cars. After the end of work day, the HEX3 to 2 will shows
83     // the rushhour and endHour. HEX1 will show the address(Hour) and HEX0 will show the maximum cars.
84     module DE1_Soc_task2_testbench();
85         logic CLOCK_50;
86         logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
87         logic [3:0] KEY;
88         logic [9:0] SW;
89         logic [9:0] LEDR;
90         wire [35:23] V_GPIO;
91
92         logic enter, exit, reset;
93         logic [1:0] num;
94         logic full;
95         logic increaseTime;
96
97         assign KEY[0] = ~increaseTime;
98         assign SW[9] = reset;
99         assign V_GPIO[23] = enter; // Presence entrance && enter
100        assign V_GPIO[24] = exit; // Presence exit && exit
101
102        DE1_Soc_task2 dut1(.CLOCK_50, .HEX0, .HEX1, .HEX2, .HEX3, .HEX4, .HEX5, .KEY, .SW, .LEDR, .V_GPIO);
103
104        parameter clock_period = 100;
105
106        initial begin
107            CLOCK_50 <= 0;
108            forever #(clock_period / 2) CLOCK_50 <= ~CLOCK_50;
109        end
110        //
111
112        initial begin
113            reset <= 1; @ (posedge CLOCK_50);
114            reset <= 0; increaseTime <= 0; enter <= 0; exit <= 0; @ (posedge CLOCK_50);
115
116            increaseTime <= 0; enter <= 0; exit <= 1; @ (posedge CLOCK_50); // Hour 1
117            increaseTime <= 0; enter <= 1; exit <= 0; @ (posedge CLOCK_50);
118            increaseTime <= 1; enter <= 0; exit <= 0; @ (posedge CLOCK_50); // Hour 2
119            increaseTime <= 0; enter <= 0; exit <= 1; @ (posedge CLOCK_50);
120            increaseTime <= 0; enter <= 1; exit <= 0; @ (posedge CLOCK_50);
121            increaseTime <= 0; enter <= 0; exit <= 1; @ (posedge CLOCK_50);
122            increaseTime <= 0; enter <= 0; exit <= 0; @ (posedge CLOCK_50);
123            increaseTime <= 1; enter <= 0; exit <= 0; @ (posedge CLOCK_50); // Hour 3
124            increaseTime <= 0; enter <= 1; exit <= 0; @ (posedge CLOCK_50);
125            increaseTime <= 0; enter <= 1; exit <= 0; @ (posedge CLOCK_50);
126            increaseTime <= 0; enter <= 1; exit <= 0; @ (posedge CLOCK_50);
127            increaseTime <= 0; enter <= 0; exit <= 0; @ (posedge CLOCK_50);
128            increaseTime <= 1; enter <= 0; exit <= 0; @ (posedge CLOCK_50); // Hour 4
129            increaseTime <= 0; enter <= 0; exit <= 1; @ (posedge CLOCK_50);
130            increaseTime <= 0; enter <= 0; exit <= 1; @ (posedge CLOCK_50);
131            increaseTime <= 1; enter <= 0; exit <= 0; @ (posedge CLOCK_50); // Hour 5
132            increaseTime <= 0; enter <= 1; exit <= 0; @ (posedge CLOCK_50);
133            increaseTime <= 0; enter <= 0; exit <= 1; @ (posedge CLOCK_50);
134            increaseTime <= 1; enter <= 0; exit <= 0; @ (posedge CLOCK_50); // Hour 6
135            increaseTime <= 0; enter <= 1; exit <= 0; @ (posedge CLOCK_50);
136            increaseTime <= 0; enter <= 1; exit <= 0; @ (posedge CLOCK_50);
137            increaseTime <= 0; enter <= 0; exit <= 1; @ (posedge CLOCK_50);
138            increaseTime <= 0; enter <= 1; exit <= 0; @ (posedge CLOCK_50);
139            increaseTime <= 1; enter <= 0; exit <= 0; @ (posedge CLOCK_50); // Hour 7
140            increaseTime <= 0; enter <= 0; exit <= 1; @ (posedge CLOCK_50);
141            increaseTime <= 0; enter <= 0; exit <= 1; @ (posedge CLOCK_50);
142            increaseTime <= 0; enter <= 0; exit <= 1; @ (posedge CLOCK_50);
143            increaseTime <= 0; enter <= 0; exit <= 1; @ (posedge CLOCK_50);
144            increaseTime <= 1; enter <= 0; exit <= 0; @ (posedge CLOCK_50); // Hour 8
145            increaseTime <= 0; enter <= 1; exit <= 0; @ (posedge CLOCK_50);
146            increaseTime <= 0; enter <= 1; exit <= 0; @ (posedge CLOCK_50);
147            increaseTime <= 1; enter <= 0; exit <= 0; @ (posedge CLOCK_50); // Hour display
148            increaseTime <= 0; enter <= 0; exit <= 0; @ (posedge CLOCK_50);
149            increaseTime <= 0; enter <= 0; exit <= 0; @ (posedge CLOCK_50);
150            increaseTime <= 0; enter <= 0; exit <= 0; @ (posedge CLOCK_50);
151            reset <= 1; @ (posedge CLOCK_50);
152            reset <= 0; @ (posedge CLOCK_50);
153
154            $stop;
155        end
156
157    endmodule
158
159
160
161

```

2) Display

```

1 // Xuanchang Hu
2 // 01/20/2023
3 // EE 371
4 // Lab6 Task2
5
6 // display can display the number on the 6 HEX displays. It takes address as
7 // input and hex as output.
8 timescale 1 ps / 1 ps
9 module display(address, hex);
10 input logic [3:0] address;
11 output logic [6:0] hex;
12
13 //HEX correspond to different number
14 always_comb begin
15     case (address)
16
17         0: hex = 7'b1000000;
18         1: hex = 7'b1111001;
19         2: hex = 7'b0100100;
20         3: hex = 7'b0110000;
21         4: hex = 7'b0011001;
22         5: hex = 7'b0010010;
23         6: hex = 7'b0000010;
24         7: hex = 7'b1111000;
25         8: hex = 7'b0000000;
26         9: hex = 7'b0010000;
27         10: hex = 7'b0001000; //A
28         11: hex = 7'b0000011; //b
29         12: hex = 7'b1000110; //C
30         13: hex = 7'b0100001; //d
31         14: hex = 7'b0000110; //E
32         15: hex = 7'b0001110; //F
33     endcase
34 end
35
36 endmodule
37
38 // display_testbench tests all cases of number
39 module display_testbench();
40     logic [3:0] address;
41     logic [6:0] hex;
42
43
44     display dut(.address, .hex);
45
46     initial begin
47         address = 4'b0100; #10; //4
48         address = 4'b0001; #10; //1
49         address = 4'b0010; #10; //2
50         address = 4'b0011; #10; //3
51         address = 4'b0101; #10; //5
52         address = 4'b0110; #10; //6
53         address = 4'b0111; #10; //7
54         address = 4'b1000; #10; //8
55         address = 4'b1001; #10; //9
56         address = 4'b1010; #10; //10
57         $stop();
58     end
59
60 endmodule

```

3) Counter

```

1 //Xuanchang Hu
2 //03/13/2023
3 //EE 371
4 //Lab6 task2
5
6 // This module will cycle between 0 and 7
7 timescale 1 ps / 1 ps
8 module counter(clk, reset, out);
9     input logic clk, reset;
10    output logic [2:0] out;
11
12    // When reset, number is 0. Otherwise, number cycle from 0 to 7.
13    always_ff @(posedge clk) begin
14        if (reset) begin
15            out <= 0;
16        end else begin
17            if (out < 7) begin
18                out <= out + 1;
19            end else begin // restart counter
20                out <= 0;
21            end
22        end
23    end
24 endmodule
25
26 // It tests cases of number cycle from 0 to 7 and reset.
27 module counter_testbench();
28     logic clk, reset;
29     logic [2:0] out;
30
31     counter dut1(.clk, .reset, .out);
32
33
34     parameter clock_period = 100;
35
36     initial begin
37         clk <= 0;
38         forever #(clock_period / 2) clk <= ~clk;
39     end
40     //
41
42     initial begin
43         reset <= 1;           @(posedge clk);
44         reset <= 0;           @(posedge clk); // counter starts counting
45         @(posedge clk);
46         @(posedge clk);
47         @(posedge clk);
48         @(posedge clk);
49         @(posedge clk);
50         @(posedge clk);
51         @(posedge clk);
52         @(posedge clk);
53         @(posedge clk);
54         @(posedge clk);
55         reset <= 1;           @(posedge clk); //reset
56         reset <= 0;           @(posedge clk);
57         @(posedge clk);
58
59         $stop;
60     end
61
62 endmodule
63

```

4) Ram8x16

```

1 //Xuanchang Hu
2 //03/13/2023
3 //EE 371
4 //Lab6 task2
5
6 // megafunction wizard: %RAM: 2-PORT%
7 // GENERATION: STANDARD
8 // VERSION: WM1.0
9 // MODULE: altsyncram
10
11 // =====
12 // File Name: ram8x16.v
13 // Megafunction Name(s):
14 //     altsyncram
15 //
16 // Simulation Library File(s):
17 //     altera_mf
18 // =====
19 // *****
20 // THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
21 //
22 // 17.0.0 Build 595 04/25/2017 SJ Lite Edition
23 // *****
24
25
26 //Copyright (C) 2017 Intel Corporation. All rights reserved.
27 //Your use of Intel Corporation's design tools, logic functions
28 //and other software and tools, and its AMPP partner logic
29 //functions, and any output files from any of the foregoing
30 //(including device programming or simulation files), and any
31 //associated documentation or information are expressly subject
32 //to the terms and conditions of the Intel Program License
33 //Subscription Agreement, the Intel Quartus Prime License Agreement,
34 //the Intel MegaCore Function License Agreement, or other
35 //applicable license agreement, including, without limitation,
36 //that your use is for the sole purpose of programming logic
37 //devices manufactured by Intel and sold by Intel or its
38 //authorized distributors. Please refer to the applicable
39 //agreement for further details.
40
41 // synopsys translate_off
42 // timescale 1 ps / 1 ps
43 // synopsys translate_on
44
45 //This module can store the data of maximum cars. The input address is the hour.
46 // read address is also the hour from counter. data and q are maximum number of cars

```

```

47 module ram8x16 (
48     clock,
49     data,
50     rdaddress,
51     wraddress,
52     wren,
53     q);
54
55     input    clock;
56     input [3:0] data;
57     input [2:0] rdaddress;
58     input [2:0] wraddress;
59     input    wren;
60     output [3:0] q;
61 `ifndef ALTERA_RESERVED_QIS
62 // synopsys translate_off
63 `endif
64     tri1    clock;
65     tri0    wren;
66 `ifndef ALTERA_RESERVED_QIS
67 // synopsys translate_on
68 `endif
69
70     wire [3:0] sub_wire0;
71     wire [3:0] q = sub_wire0[3:0];
72
73     altsyncram altsyncram_component (
74         .address_a (wraddress),
75         .address_b (rdaddress),
76         .clock0 (clock),
77         .data_a (data),
78         .wren_a (wren),
79         .q_b (sub_wire0),
80         .aclr0 (1'b0),
81         .aclr1 (1'b0),
82         .addressstall_a (1'b0),
83         .addressstall_b (1'b0),
84         .byteena_a (1'b1),
85         .byteena_b (1'b1),
86         .clock1 (1'b1),
87         .clocken0 (1'b1),
88         .clocken1 (1'b1),
89         .clocken2 (1'b1),
90         .clocken3 (1'b1),
91         .data_b ({4{1'b1}}),
92         .eccstatus (),
93         .q_a (),
94         .rden_a (1'b1),
95         .rden_b (1'b1),
96         .wren_b (1'b0));
97
98     defparam
99         altsyncram_component.address_aclr_b = "NONE",
100         altsyncram_component.address_reg_b = "CLOCK0",
101         altsyncram_component.clock_enable_input_a = "BYPASS",
102         altsyncram_component.clock_enable_input_b = "BYPASS",
103         altsyncram_component.clock_enable_output_b = "BYPASS",
104         altsyncram_component.intended_device_family = "Cyclone V",
105         altsyncram_component.lpm_type = "altsyncram",
106         altsyncram_component.numwords_a = 8,
107         altsyncram_component.numwords_b = 8,
108         altsyncram_component.operation_mode = "DUAL_PORT",
109         altsyncram_component.outdata_aclr_b = "NONE",
110         altsyncram_component.outdata_reg_b = "CLOCK0",
111         altsyncram_component.power_up_uninitialized = "FALSE",
112         altsyncram_component.ram_block_type = "M10K",
113         altsyncram_component.read_during_write_mode_mixed_ports = "DONT_CARE",
114         altsyncram_component.widthad_a = 3,
115         altsyncram_component.widthad_b = 3,
116         altsyncram_component.width_a = 4,
117         altsyncram_component.width_b = 4,
118         altsyncram_component.width_byteena_a = 1;
119
120 endmodule
121
122 // =====
123 // CNX file retrieval info
124 // =====
125 // Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
126 // Retrieval info: PRIVATE: ADDRESSSTALL_B NUMERIC "0"
127 // Retrieval info: PRIVATE: BYTEENA_ACLR_A NUMERIC "0"
128 // Retrieval info: PRIVATE: BYTEENA_ACLR_B NUMERIC "0"
129 // Retrieval info: PRIVATE: BYTE_ENABLE_A NUMERIC "0"
130 // Retrieval info: PRIVATE: BYTE_ENABLE_B NUMERIC "0"
131 // Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
132 // Retrieval info: PRIVATE: BlankMemory NUMERIC "1"
133 // Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
134 // Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_B NUMERIC "0"
135 // Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
136 // Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_B NUMERIC "0"
137 // Retrieval info: PRIVATE: CLRdata NUMERIC "0"
138 // Retrieval info: PRIVATE: CLRq NUMERIC "0"

```

5) Fsm

```
1 //Xuanchang Hu
2 //03/13/2023
3 //EE 371
4 //Lab6 task2
5
6 `timescale 1 ps / 1 ps
7 // fsm takes enter, exit, increaseTime as input, output hours and endDay.
8 // If increaseTime, the state will move to next hour. Each state from s1 to s7 represent an hour.
9 // When state is sDisplay, hour is still 7, endDay will changes from 0 to 1. Only reset
10 // can fsm start over.
11 module fsm(clk, reset, enter, exit, increaseTime, hours, endDay);
12     input logic clk, reset, enter, exit, increaseTime;
13
14     output logic [2:0] hours;
15     output logic endDay;
16
17     // Setting 10 states to track cars at each hour
18     enum {none, s1, s2, s3, s4, s5, s6, s7, s8, sDisplay} ps, ns;
19
20     // If increaseTime, the state will move to next state. If reset,
21     // state will be none. Each state from s1 to s7 represent an hour
22     // Before sDisplay, endDay is 0. When at sDisplay state, endDay is 1.
23     always_comb begin
24         case (ps)
25             none: begin
26                 hours = 0;
27                 endDay = 0;
28                 ns = s1;
29             end
30
31             s1: begin
32                 hours = 0;
33                 endDay = 0;
34                 if (increaseTime)
35                     ns = s2;
36                 else
37                     ns = s1;
38             end
39
40             s2: begin
41                 hours = 1;
42                 endDay = 0;
43                 if (increaseTime)
44                     ns = s3;
45                 else
46                     ns = s2;
47
48             end
49
50             s3: begin
51                 hours = 2;
52                 endDay = 0;
53                 if (increaseTime)
54                     ns = s4;
55                 else
56                     ns = s3;
57             end
58
59             s4: begin
60                 hours = 3;
61                 endDay = 0;
62                 if (increaseTime)
63                     ns = s5;
64                 else
65                     ns = s4;
66             end
67
68             s5: begin
69                 hours = 4;
70                 endDay = 0;
71                 if (increaseTime)
72                     ns = s6;
73                 else
74                     ns = s5;
75             end
76
77             s6: begin
78                 hours = 5;
79                 endDay = 0;
80                 if (increaseTime)
81                     ns = s7;
82                 else
83                     ns = s6;
84             end
85
86             s7: begin
87                 hours = 6;
88                 endDay = 0;
89                 if (increaseTime)
90                     ns = s8;
91                 else
92                     ns = s7;
```

6) Datapath

```

1 //Xuanchang Hu
2 //03/13/2023
3 //EE 371
4 //Lab6 task2
5
6 // the datapath takes clk, reset, enter, exit, hours as input. And it output
7 // number of cars, maximum number of cars, rushHour, endHour noRush, noEnd.
8 // When enter, number and maxCar increases by 1. When exit, number decreases
9 // by 1. When number is first 3, rushHour is current hour. When number changes
10 // from 3 to 0, endHour is current hour.
11 //
12 //
13 timescale 1 ps / 1 ps
14 module datapath (clk, reset, enter, exit, hours, num, maxCar,
15                 rushHour, endHour, noRush, noEnd);
16     input logic clk, reset, enter, exit;
17     input logic [2:0] hours;
18
19     output logic [1:0] num;
20     output logic [3:0] maxCar;
21     output logic [2:0] rushHour, endHour;
22     output logic noRush, noEnd;
23
24     logic rush;
25
26     assign noRush = ~rush;
27
28     // When reset, number and maxCar are both 0.
29     // When enter, number and maxCar increases by 1. When exit, number decreases
30     // by 1.
31     always_ff @(posedge clk) begin
32         if (reset) begin
33             num <= 0;
34             maxCar <= 0;
35         end else begin
36             if (enter && (num < 3)) begin
37                 num <= num + 1;
38                 maxCar <= maxCar + 1;
39             end
40
41             if (exit && (num > 0)) begin
42                 num <= num - 1;
43             end
44         end
45     end
46
47     end
48
49     // When reset, rushHour, endHour, rush are 0. noEnd is 1.
50     // When number is first 3 and rush is 0, rush will become 1 and
51     // rushHour will be the current hour.
52     // If rush is 1, noEnd is 1 and number become 0. That means rushHour ends.
53     // So endHour is current hour.
54     always_ff @(posedge clk) begin
55         if (reset) begin
56             rushHour <= 0;
57             endHour <= 0;
58             rush <= 0;
59             noEnd <= 1;
60         end else begin
61             if ((rush == 0) && (num == 3)) begin
62                 rush <= 1;
63                 rushHour <= hours;
64             end
65
66             if ((rush == 1) && (noEnd == 1) && (num == 0)) begin
67                 noEnd <= 0;
68                 endHour <= hours;
69             end
70         end
71     end
72     end
73
74 endmodule
75
76 // It tests cases of enter, exit cars. I also tests the case when parking is full and reset parking.
77 // When enter, number increases by 1. When exit, number decreases by 1. When number is first 3,
78 // rushHour is the current hour. When number changes from 3 to 0, the endHour is current hour.
79 module datapath_testbench();
80     logic clk, reset, enter, exit;
81     logic [2:0] hours;
82
83     logic [1:0] num;
84     logic [3:0] maxCar;
85     logic [2:0] rushHour, endHour;
86     logic noRush, noEnd;
87
88     logic rush;
89
90     datapath dut1(.clk, .reset, .enter, .exit, .hours, .num, .maxCar,
91                  .rushHour, .endHour, .noRush, .noEnd);
92

```



```

93
94
95 parameter clock_period = 100;
96
97 initial begin
98     clk <= 0;
99     forever #(clock_period / 2) clk <= ~clk;
100 end
101 //
102
103 initial begin
104     reset <= 1; @ (posedge clk);
105     reset <= 0; enter <= 0; exit <= 0; hours <= 0; @ (posedge clk);
106     hours <= 0; enter <= 1; exit <= 0; @ (posedge clk); // Hour 1
107     hours <= 0; enter <= 1; exit <= 0; @ (posedge clk);
108     hours <= 0; enter <= 0; exit <= 0; @ (posedge clk);
109     hours <= 1; enter <= 0; exit <= 0; @ (posedge clk); // Hour 2
110     hours <= 1; enter <= 0; exit <= 1; @ (posedge clk);
111     hours <= 1; enter <= 1; exit <= 0; @ (posedge clk);
112     hours <= 1; enter <= 0; exit <= 1; @ (posedge clk);
113     hours <= 2; enter <= 0; exit <= 0; @ (posedge clk); // Hour 3
114     hours <= 2; enter <= 1; exit <= 0; @ (posedge clk);
115     hours <= 2; enter <= 1; exit <= 0; @ (posedge clk);
116     hours <= 2; enter <= 1; exit <= 0; @ (posedge clk);
117     hours <= 3; enter <= 0; exit <= 0; @ (posedge clk); // Hour 4
118     hours <= 3; enter <= 0; exit <= 1; @ (posedge clk);
119     hours <= 3; enter <= 0; exit <= 1; @ (posedge clk);
120     hours <= 4; enter <= 0; exit <= 0; @ (posedge clk); // Hour 5
121     hours <= 4; enter <= 1; exit <= 0; @ (posedge clk);
122     hours <= 4; enter <= 0; exit <= 1; @ (posedge clk);
123     hours <= 5; enter <= 0; exit <= 0; @ (posedge clk); // Hour 6
124     hours <= 5; enter <= 1; exit <= 0; @ (posedge clk);
125     hours <= 5; enter <= 1; exit <= 0; @ (posedge clk);
126     hours <= 5; enter <= 0; exit <= 1; @ (posedge clk);
127     hours <= 5; enter <= 1; exit <= 0; @ (posedge clk);
128     hours <= 6; enter <= 0; exit <= 0; @ (posedge clk); // Hour 7
129     hours <= 6; enter <= 0; exit <= 1; @ (posedge clk);
130     hours <= 6; enter <= 0; exit <= 1; @ (posedge clk);
131     hours <= 6; enter <= 0; exit <= 1; @ (posedge clk);
132     hours <= 6; enter <= 0; exit <= 1; @ (posedge clk);
133     hours <= 7; enter <= 0; exit <= 0; @ (posedge clk); // Hour 8
134     hours <= 7; enter <= 1; exit <= 0; @ (posedge clk);
135     hours <= 7; enter <= 1; exit <= 0; @ (posedge clk);
136     hours <= 7; enter <= 0; exit <= 0; @ (posedge clk); // Hour display
137     hours <= 7; enter <= 0; exit <= 0; @ (posedge clk);
138     hours <= 7; enter <= 0; exit <= 0; @ (posedge clk);
139     hours <= 7; enter <= 0; exit <= 0; @ (posedge clk);
140     reset <= 1; hours <= 0; @ (posedge clk);
141     reset <= 0; @ (posedge clk);
142     @ (posedge clk);
143     @ (posedge clk);
144     $stop;
145 end
146
147
148 endmodule

```

7)Control


```

1 //Xuanchang Hu
2 //03/13/2023
3 //EE 371
4 //Lab6 task2
5
6 `timescale 1 ps / 1 ps
7 // This module is to control the parking counter. It takes clk, reset, enter, exit
8 // increaseTime as input, output HEX and whether parking full. It combines all the other submodules
9 // to implement the parking system.
10 module control (clk, reset, enter, exit, increaseTime,
11               num, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, full);
12
13     input logic clk, reset, enter, exit, increaseTime;
14
15     output logic [1:0] num;
16     output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
17     output logic full;
18
19     logic [2:0] hours;
20     logic [3:0] maxCar;
21     logic [2:0] rushHour, endHour;
22     logic [2:0] out;
23     logic endDay;
24
25     logic [3:0] maxOut;
26     logic [6:0] cars, dayTime, outNum, carNum, rushTime, endTime;
27     logic noRush, noEnd;
28
29     assign full = (num == 3);
30
31     // clock divider setup
32     logic [31:0] div_clk;
33
34     // The clock divider can decreases the clock speed in simulation.
35     // It takes clk as input and output divider clock.
36     clock_divider clkdivide (.clock(clk), .divided_clocks(div_clk));
37
38     logic clkSelect;
39     parameter whichClock = 25; // 0.75 Hz clock
40
41     assign clkSelect = clk; // for simulation
42     //assign clkSelect = div_clk[whichClock]; // for board
43
44
45
46     // finite state machines, it can output hours to show time
47     fsm statemachine(clk(clkSelect), .reset, .enter, .exit, .increaseTime, .hours, .endDay);
48
49
50     // datapath can output number of cars in parking, also output rush hour, end of rush hour.
51     datapath data(clk(clkSelect), .reset, .enter, .exit, .hours, .num, .maxCar, .rushHour, .endHour,
52                 .noRush, .noEnd);
53
54
55     // counter that cycle between 0-7
56     counter cycle(clk(clkSelect), .reset, .out);
57
58     // It can stores the max value of cars at each hour
59     ram8x16 ram(.clock(clkSelect), .data(maxCar), .rdaddress(out), .wraddress(hours), .wren(1'b1), .q(maxOut));
60
61
62     display h5(.address({1'b0, hours}), .hex(dayTime)); // HEX that show daytime
63     display ca(.address({2'b0, num}), .hex(cars)); // HEX that show number of cars
64     display rushTim(.address({1'b0, rushHour}), .hex(rushTime)); // HEX that show rush time
65     display endTim(.address({1'b0, endHour}), .hex(endTime)); // HEX that show end of rush time
66     display outNu(.address({1'b0, out}), .hex(outNum)); // HEX that show address counter number
67     display carNu(.address(maxOut), .hex(carNum)); // HEX that show max number of cars in ram
68
69     // It can change HEX output based on different conditions. When in the work time(endDay == 0)
70     // HEX5 shows the hours during 8 hour work time. When thr number of cars are 3, HEX3 to 0 shows FULL.
71     // Otherwise, HEX0 shows the number of cars. After the end of work day(endDay == 1),
72     // the HEX3 to 2 will shows the rushhour and endHour. HEX1 will show the address(Hour)
73     // and HEX0 will show the maximum cars.
74     always_comb begin
75         if (endDay == 0) begin //If day not end
76             if (num == 3) begin
77                 HEX5 = dayTime; //hour
78                 HEX4 = 7'b1111111; //blank
79                 HEX3 = 7'b0001110; // F
80                 HEX2 = 7'b1000001; // U
81                 HEX1 = 7'b1000111; // L
82                 HEX0 = 7'b1000111; // L
83             end else begin
84                 HEX5 = dayTime; //hour
85                 HEX4 = 7'b1111111;
86                 HEX3 = 7'b1111111;
87                 HEX2 = 7'b1111111;
88                 HEX1 = 7'b1111111;
89                 HEX0 = cars; //number of cars
90             end
91         end else begin //If day ends
92             if (noRush) begin

```

```

93         HEX4 = 7'b0111111; //show rush time
94     end else begin
95         HEX4 = rushTime;
96     end
97
98     if (noEnd) begin //show end of rush time
99         HEX3 = 7'b0111111;
100     end else begin
101         HEX3 = endTime;
102     end
103
104     HEX2 = outNum; // Show address
105     HEX1 = carNum; //show maximum number of cars
106     HEX5 = 7'b1111111;
107     HEX0 = 7'b1111111;
108 end
109
110 end
111 //
112
113
114
115
116
117 endmodule
118
119 // It tests cases of enter, exit cars. I also tests the case when parking is full and reset parking.
120 // HEX5 shows the hours during 8 hour work time. When the number of cars are 3, HEX3 to 0 shows FULL.
121 // Otherwise, HEX0 shows the number of cars. After the end of work day, the HEX3 to 2 will shows
122 // the rushhour and endHour. HEX1 will show the address(Hour) and HEX0 will show the maximum cars.
123 module control_testbench();
124     logic clk, reset, enter, exit, increaseTime;
125
126     logic [1:0] num;
127     logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
128     logic full;
129
130     //just logics
131     logic [2:0] hours;
132     logic [6:0] maxCar;
133     logic [2:0] rushHour, endHour;
134     logic [2:0] out;
135     logic endDay;
136
137     logic [3:0] maxOut;
138     logic [6:0] cars, dayTime, outNum, carNum, rushTime, endTime;
139
140

```

```

139 logic noRush, noEnd;
140
141 control dut1(.clk, .reset, .enter, .exit, .increaseTime,
142             .num, .HEX0, .HEX1, .HEX2, .HEX3, .HEX4, .HEX5, .full);
143
144 parameter clock_period = 100;
145
146 initial begin
147     clk <= 0;
148     forever #(clock_period / 2) clk <= ~clk;
149 end
150 //
151
152 initial begin
153     reset <= 1;
154     reset <= 0; increaseTime <= 0; enter <= 0; exit <= 0; @ (posedge clk);
155     increaseTime <= 0; enter <= 1; exit <= 0; @ (posedge clk); // Hour 1
156     increaseTime <= 0; enter <= 1; exit <= 0; @ (posedge clk);
157     increaseTime <= 0; enter <= 0; exit <= 0; @ (posedge clk);
158     increaseTime <= 1; enter <= 0; exit <= 0; @ (posedge clk); // Hour 2
159     increaseTime <= 0; enter <= 0; exit <= 1; @ (posedge clk);
160     increaseTime <= 0; enter <= 1; exit <= 0; @ (posedge clk);
161     increaseTime <= 0; enter <= 0; exit <= 1; @ (posedge clk);
162     increaseTime <= 0; enter <= 0; exit <= 0; @ (posedge clk);
163     increaseTime <= 1; enter <= 0; exit <= 0; @ (posedge clk); // Hour 3
164     increaseTime <= 0; enter <= 1; exit <= 0; @ (posedge clk);
165     increaseTime <= 0; enter <= 1; exit <= 0; @ (posedge clk);
166     increaseTime <= 0; enter <= 1; exit <= 0; @ (posedge clk);
167     increaseTime <= 0; enter <= 0; exit <= 0; @ (posedge clk);
168     increaseTime <= 1; enter <= 0; exit <= 0; @ (posedge clk); // Hour 4
169     increaseTime <= 0; enter <= 0; exit <= 1; @ (posedge clk);
170     increaseTime <= 0; enter <= 0; exit <= 1; @ (posedge clk);
171     increaseTime <= 1; enter <= 0; exit <= 0; @ (posedge clk); // Hour 5
172     increaseTime <= 0; enter <= 1; exit <= 0; @ (posedge clk);
173     increaseTime <= 0; enter <= 0; exit <= 1; @ (posedge clk);
174     increaseTime <= 1; enter <= 0; exit <= 0; @ (posedge clk); // Hour 6
175     increaseTime <= 0; enter <= 1; exit <= 0; @ (posedge clk);
176     increaseTime <= 0; enter <= 1; exit <= 0; @ (posedge clk);
177     increaseTime <= 0; enter <= 0; exit <= 1; @ (posedge clk);
178     increaseTime <= 0; enter <= 1; exit <= 0; @ (posedge clk);
179     increaseTime <= 1; enter <= 0; exit <= 0; @ (posedge clk); // Hour 7
180     increaseTime <= 0; enter <= 0; exit <= 1; @ (posedge clk);
181     increaseTime <= 0; enter <= 0; exit <= 1; @ (posedge clk);
182     increaseTime <= 0; enter <= 0; exit <= 1; @ (posedge clk);
183     increaseTime <= 0; enter <= 0; exit <= 1; @ (posedge clk);
184     increaseTime <= 1; enter <= 0; exit <= 0; @ (posedge clk); // Hour 8
185
186     increaseTime <= 0; enter <= 1; exit <= 0; @ (posedge clk);
187     increaseTime <= 1; enter <= 0; exit <= 0; @ (posedge clk); // Hour display
188     increaseTime <= 0; enter <= 0; exit <= 0; @ (posedge clk);
189     increaseTime <= 0; enter <= 0; exit <= 0; @ (posedge clk);
190     increaseTime <= 0; enter <= 0; exit <= 0; @ (posedge clk);
191     reset <= 1;
192     reset <= 0;
193
194     $stop;
195 end
196
197
198
199 endmodule

```

8) clock_divider

```

1 // Xuanchang Hu
2 // 01/20/2023
3 // EE 371
4 // Lab6 Task 2
5
6 // divided_clocks[0] = 25MHz, [1] = 12.5Mhz, ... [23] = 3Hz, [24] = 1.5Hz, [25] = 0.75Hz, ...
7 // It takes clock as input and divided_clocks as output. it can divide clock into
8 // different frequency.
9 module clock_divider(clock, divided_clocks);
10     input logic clock;
11     output logic [31:0] divided_clocks = 0;
12
13     always_ff @(posedge clock) begin
14         divided_clocks <= divided_clocks + 1;
15     end
16
17 endmodule
18
19 module clock_divider_testbench();
20     logic clock;
21     logic [31:0] divided_clocks;
22
23     `timescale 1 ps / 1 ps
24
25     clock_divider dut(.clock, .divided_clocks);
26
27     initial begin
28         @(posedge clock);
29         @(posedge clock);
30
31     end
32
33
34
35 endmodule

```