UW Student: Brian Chen

EE 371

March 14, 2025

Lab6 Report

I. Procedure

Task 1: Parking Lot Breadboard

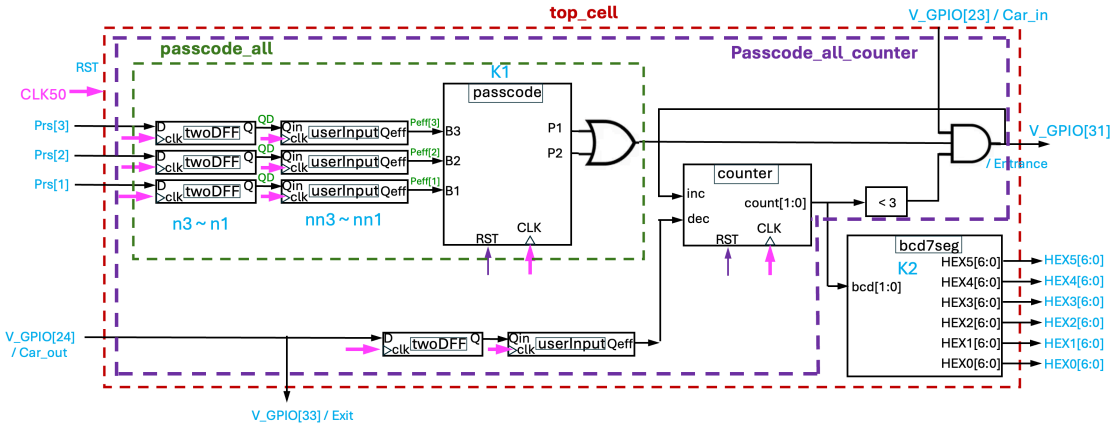

Figure 1: Block diagram of the “DE1_SoC” module of Task 1.

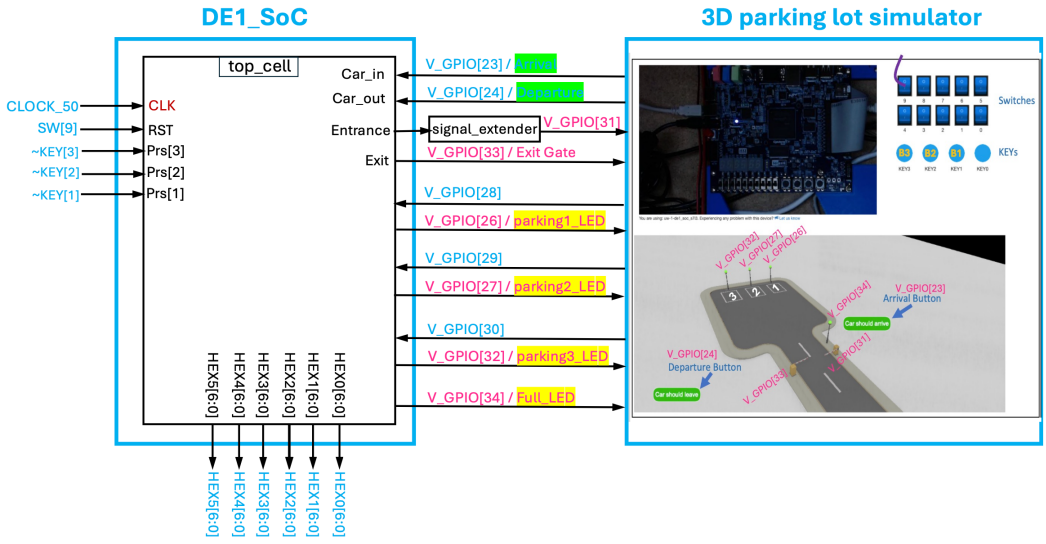

Figure 2: Block diagram of the “DE1_SoC” module and the 3D Parking Lot Simulator of Task 1.

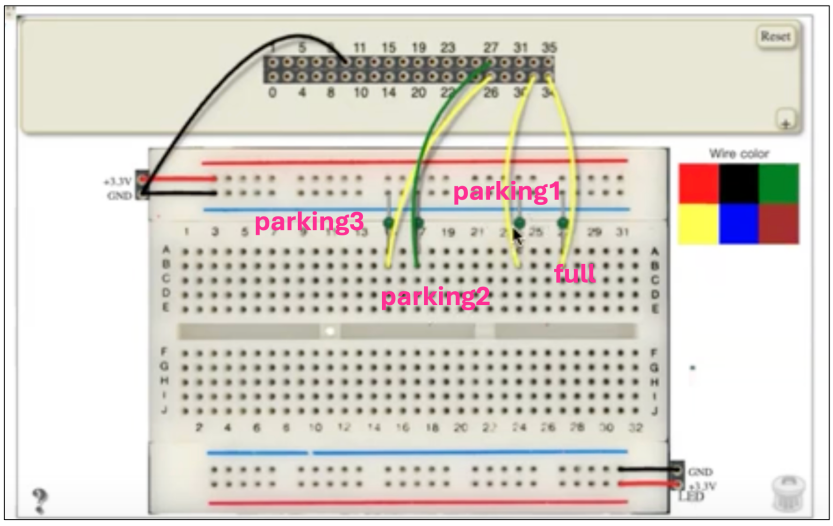

Figure 3: Screen shot of the virtual bread board, components, wires,

and the GPIO pins of Task 1

```
top_cell top (.CLK(CLOCK_50), .RST(SW[9]), .Prs(~KEY),
              .Car_in(V_GPIO[23]), .Car_out(V_GPIO[24]),
              .Entrance(hold), .Exit(V_GPIO[33]),
              .HEX5, .HEX4, .HEX3, .HEX2, .HEX1, .HEX0);

// use signal_extender module to extend the time duration for Entrance gate.
signal_extender top_ext(.in(hold), .out(hold2), .reset(SW[9]), .clk(CLOCK_50));

assign V_GPIO[31] = hold2;
assign V_GPIO[26] = V_GPIO[28];  // specify the color of LED1.
assign V_GPIO[27] = V_GPIO[29];  // specify the color of LED2.
assign V_GPIO[32] = V_GPIO[30];  // specify the color of LED3.

// specify the color (0:green / 1:red) of the full indicaror LED.
assign V_GPIO[34] = V_GPIO[28] & V_GPIO[29] & V_GPIO[30];
```

Figure 4: Core portion of the SystemVerilog codes in “DE1_SoC.sv” of Task 1.

Figure 1 shows the block diagram of the design in Task 1. Figure 2 shows the block diagram illustrating the interconnections between the DE1_SoC and the 3D Parking Lot Simulator. These two block diagrams are the same as that in LAB1b. Figure 3 shows the screen shot of the virtual bread board. On this virtual bread board, I placed four LEDs and wired them to the corresponding GPIO pines. In Figure 4, I captured the core portion of the SystemVerilog codes in “DE1_SoC.sv”. In this core portion, I specified the interconnections between V_GPIO pines. After finishing the SystemVerilog codes and the works on the virtual bread board, I simulated this design on Modelsim-Altera and verified the functions on LabsLand.

Task 2: Parking Lot 3D Simulation

Figure 5 shows the I/O arrangement for DE1_SoC in Task 2. I derived the ASMD chart as shown in Figure 6. There are 8 states (s_idle, s_go, s_rush_start, s_wait1, s_rush_end, s_wait2, s_done, and s_display), 6 output signals (hour_inc, wr, RH_start, RH_flag, RH_end, and show), and 6 input signals (reset, start, KEY0, full, empty, and hour_7) in this ASMD chart. Figure 7 is the block diagram of "parkinglot_datapath" module, "top_datapath" module, and "top_ctrl_datapath" module corresponding to the ASMD chart. As shown in Figure 7, the "parkinglot_datapath" module contains three portions: the register portion (data registers and the "count_updw" module), the multiplexor portion (mux2_1 array), and the "data7seg" modules. The "top_datapath" module contains three portions: the "parkinglot_datapath" module, the "simple3D" module, and the RAM (ram8x16). The "top_ctrl_datapath" module contains two portions: the "top_datapth" module and the Control Unit ("parkinglot_ctrl" module). The Control Unit generates 6 output signals (as printed in blue) to direct the Data Path how the data are processed. The Control Unit also receives 3 input signals (full, empty, hour_7) sent from the Data Path and 3 input signals (reset, start, KEY0) sent from the DE1_SOC to do the decision making. The RAM stores 8 words, each word has 16 bits. The 6 "data7seg" modules convert the 4-bit data to 7-bit signals for driving the corresponding 7-segment displays to show the information: current hour, rush_hour_start, rush_hour_end, rdaddress, RAM contents, and the remaining parking lots. Figure 8 shows the block diagram of the simple version of the interface between the DE1_SOC and the 3D Parking Lot Simulator. Figure 9 shows the complete block diagram of Task 2. After writing the SystemVerilog codes, I simulated this design on Modelsim-Altera and verifyied the functions on LabsLand.

I/O	Port Names	Functions	User Standpoint Descriptions
input	CLOCK_50	clock	DE1_SOC provides 50-MHz clock.
input	SW [9]	reset	Reset (restart) the system by turning SW[9] to the position “ON”.
input	SW[8]	start	SW[8] ==1: start
input	KEY [0]	hour <= hour +1	Press KEY[0] to do hour increment.
output	HEX0 (L)	remaining	Display the number of remaining parking lot spaces. Display “L” when the parking lot is full.
output	HEX1 (L)	hour_car	Display the total number of cars that have enter the parking lot by the specific hour. Display “L” when the parking lot is full.
output	HEX2 (U)	rdaddress	Display the RAM address. Display “U” when the parking lot is full.
output	HEX3 (F)	rush_hour_start	Display the start of rush hour or display “-”. Display “F” when the parking lot is full.
output	HEX4	rush_hour_end	Display the end of rush hour or display “-”.
output	HEX5	current hour	Show the current hour (0~7).
output	LEDR[0]	Parking 1	RED: a car in parking lot 1.
output	LEDR[1]	Parking 2	RED: a car in parking lot 2.
output	LEDR[2]	Parking 3	RED: a car in parking lot 3.
output	LEDR[3]	Presence Entrance	RED: a car at the Entrance.
output	LEDR[4]	Presence Exit	RED: a car at the Exit.

Figure 5: I/O arrangement for DE1_SoC of Task 2.

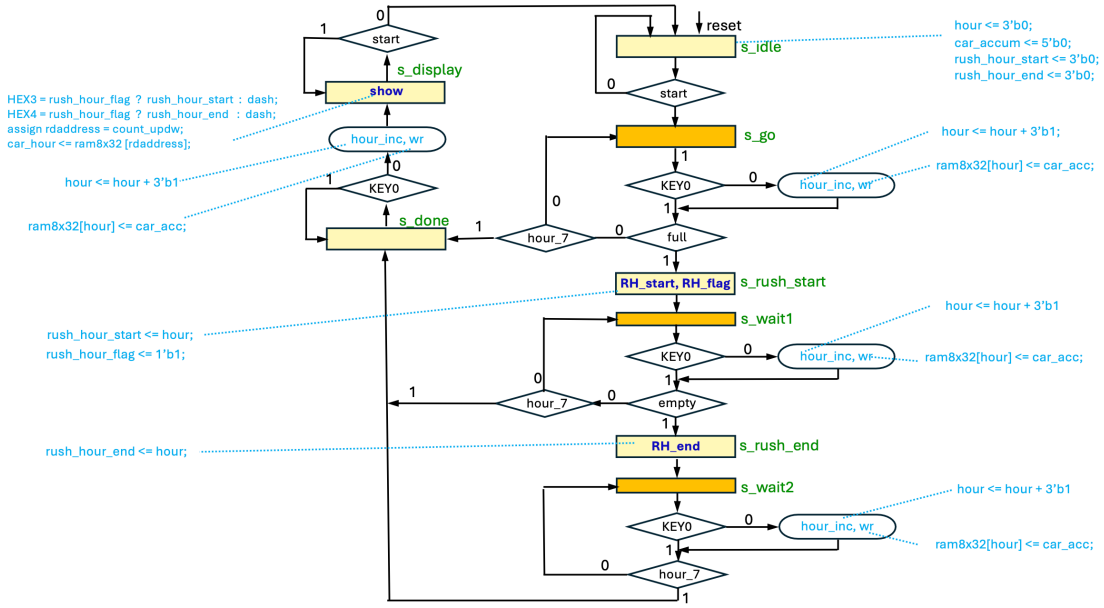

Figure 6: ASMD chart of “parkinglot_ctrl” module of Task 2.

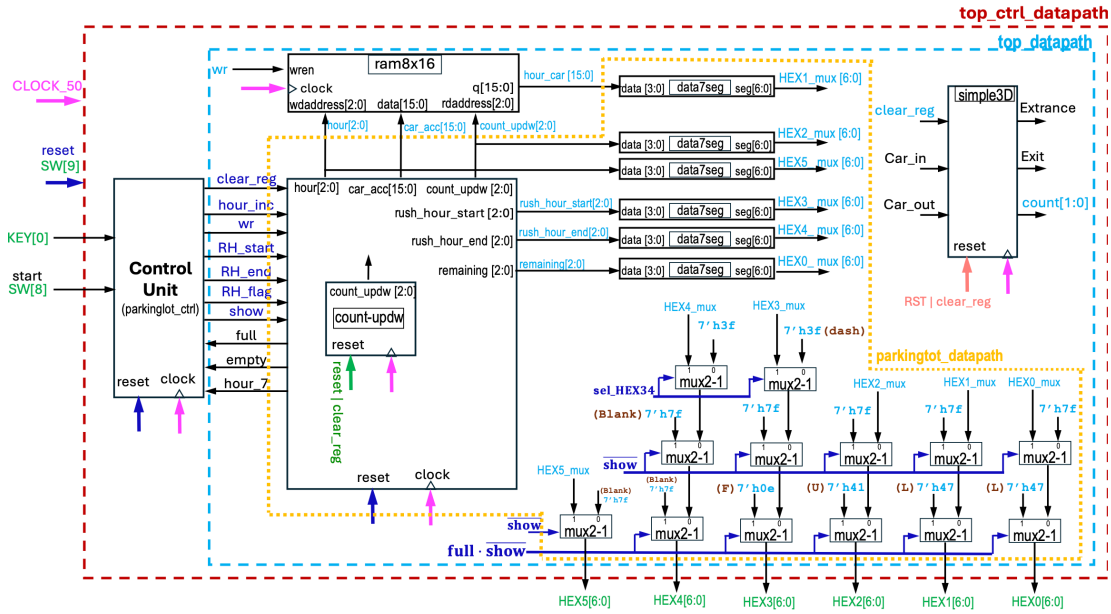

Figure 7: Block diagram of "parkinglot_datapath", "top_datapath", and "top_ctrl_datapath" modules of Task 2.

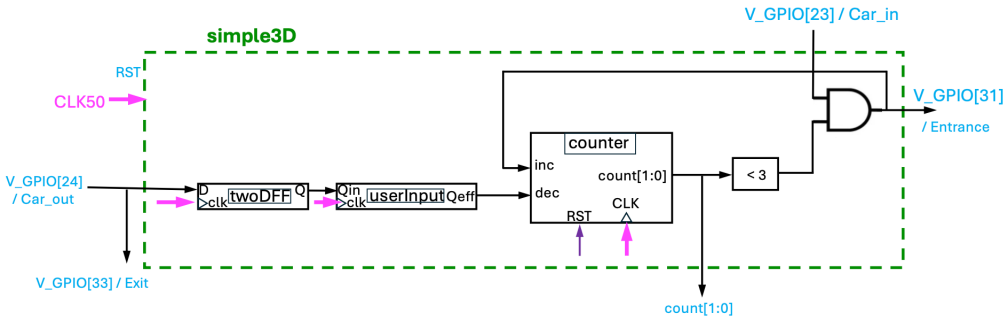

Figure 8: Block diagram of "simple3D" modules of Task 2.

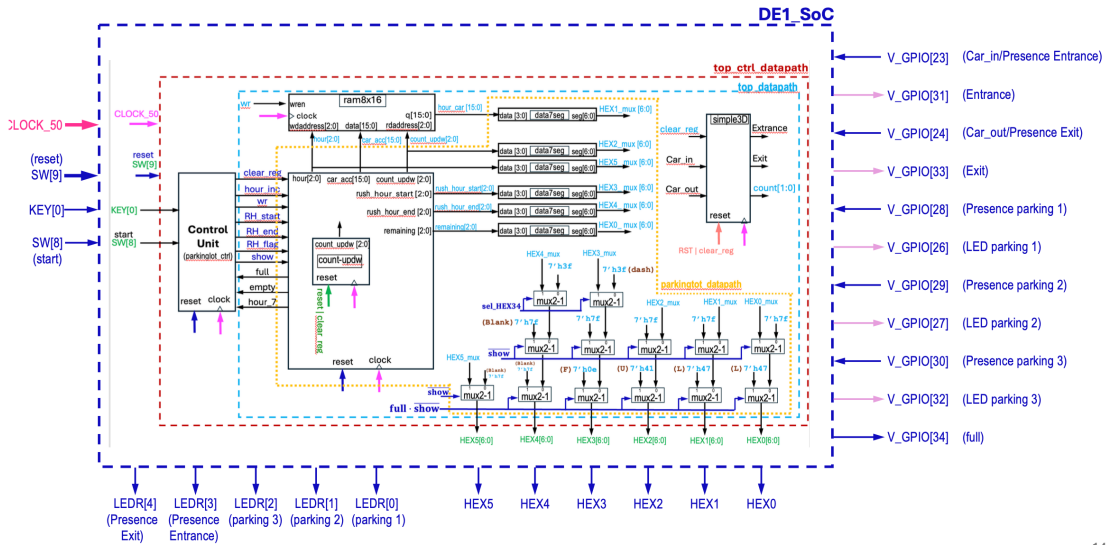

Figure 9: Block diagram of "DE1_SoC" module of Task 2.

II. Result

Task 1: Parking Lot Breadboard

Please see the video: https://youtu.be/IZmjTua7goQ

Task 2: Parking Lot 3D Simulation

Simulation Result with the Testbench

I wrote the testbench to test the modules in Task 2. Figures 10 to 18 show the simulation results of the "parkinglot_ctrl" module, the "parkinglot_datapath" module, the "top_datapath" module, the "top_ctrl_datapath" module, the "simple3D" module, the "count_updw" module, the "counter" module, the "twoDFF" module, and the "userInput" module, respectively.

Figure 19 shows the simulation results of the "top_ctrl_datapath" module during hours 0 to 7. As shown in this figure, the start of the first rush hour is 1 (full == 1, and HEX3~HEX0 display the message "FULL"), and the end of the first rush hour is 4 (empty == 1). After hour 7, HEX3 and HEX4 display the start of the first rush hour (7'h79) and the end of the first rush hour (7'h19).

Figure 20 shows the simulation results of the "top_ctrl_datapath" module during hours

0 to 7. As shown in this figure, there was no rush hours. After hour 7, both of HEX3 and HEX4 displayed "-" (7'h3f).

Figure 21 shows the simulation results of the "top_ctrl_datapath" module during hours 0 to 7. As shown in this figure, the total number of cars have entered the parking lot at each given business hour was written to the RAM.

Figure 22 shows the simulation results of the "top_ctrl_datapath" module at hour 8. As shown in this figure, the data stored in the RAM were read out cyclically according to the address sequence: 0, 1, 2, 3, 4, 5, 6, 7, 6, 5, 4, 3, 2, 1, 0, 1, 2, … until the signal "start" became 0.

Figures 23 to 26 shows the simulation results of the "DE1_SoC" module. The simulation results are similar to those in Figures 19 to 22.

Figure 10: Simulation results of the design "parkinglot_ctrl" in Task 2.

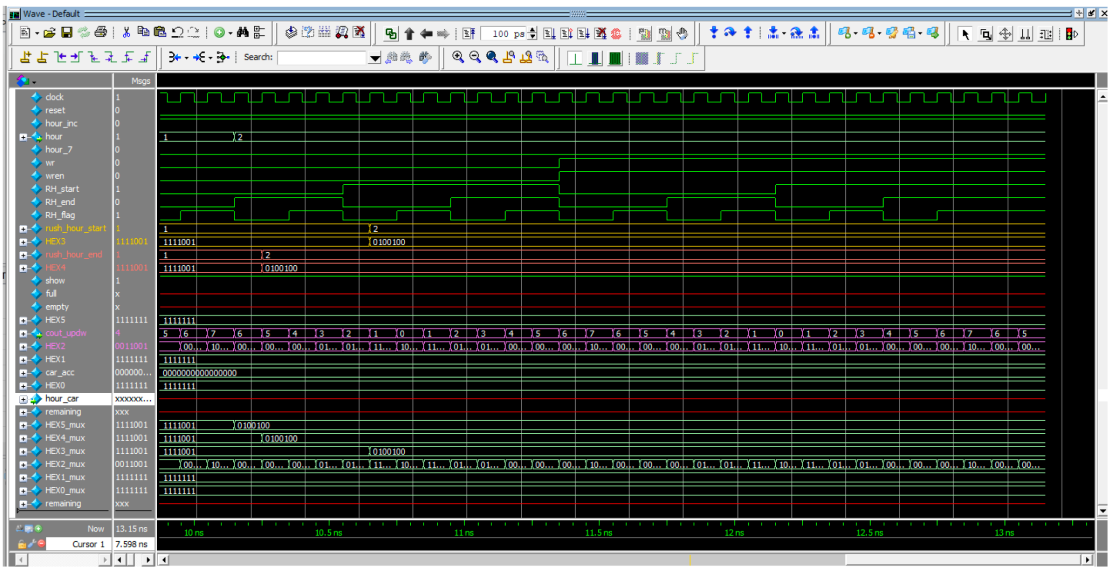

Figure 11: Simulation results of the design "parkinglot_datapath" in Task 2.

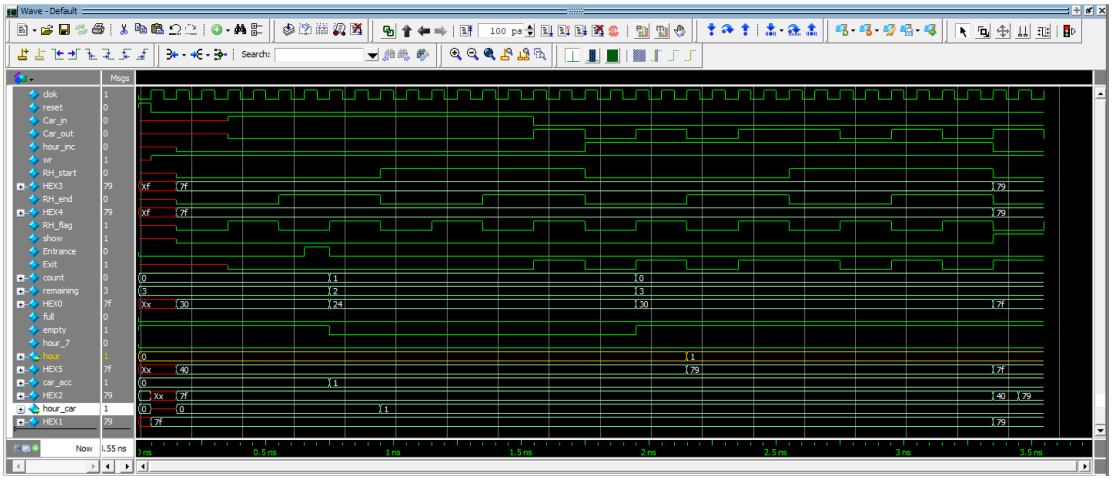

Figure 12: Simulation results of the design “top_datapath” in Task 2.

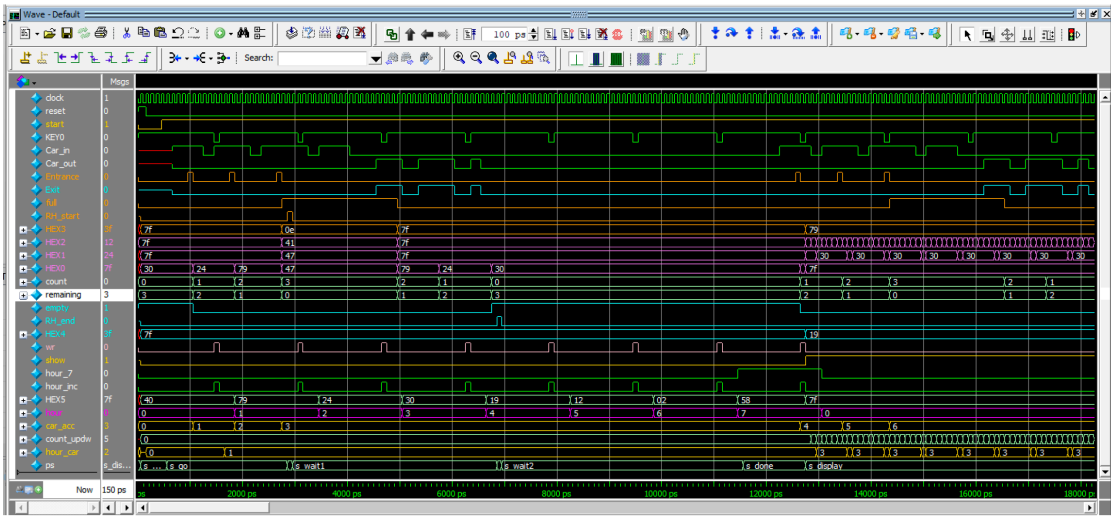

Figure 13: Simulation results of the design “top_ctrl_datapath” in Task 2.

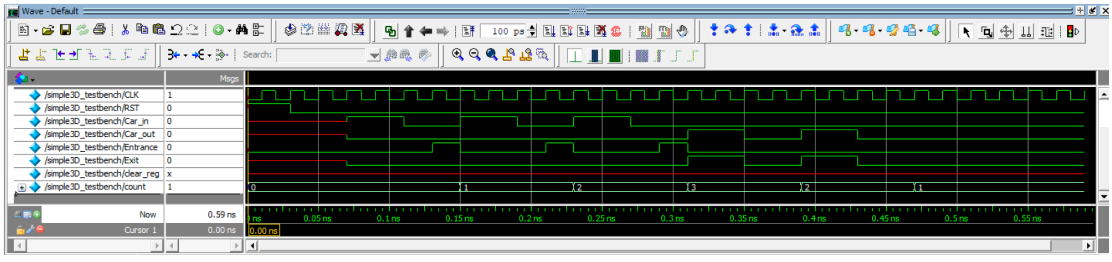

Figure 14: Simulation results of the design “simple3D” in Task 2.

Figure 15: Simulation results of the design “count_updw” in Task 2.

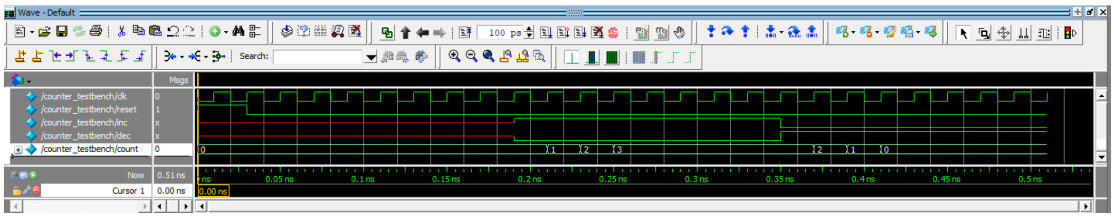

Figure 16: Simulation results of the design “counter” in Task 2.

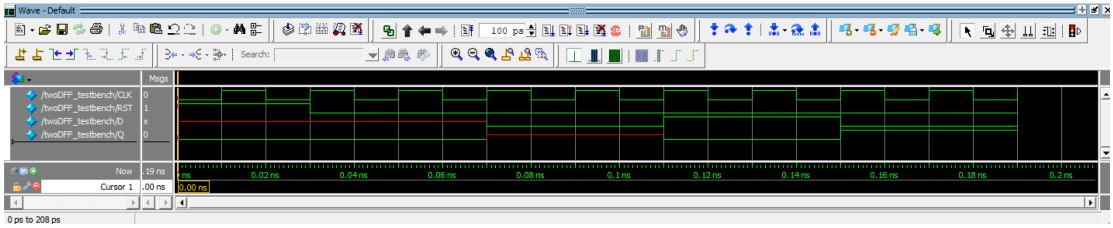

Figure 17: Simulation results of the design “twoDFF” in Task 2.

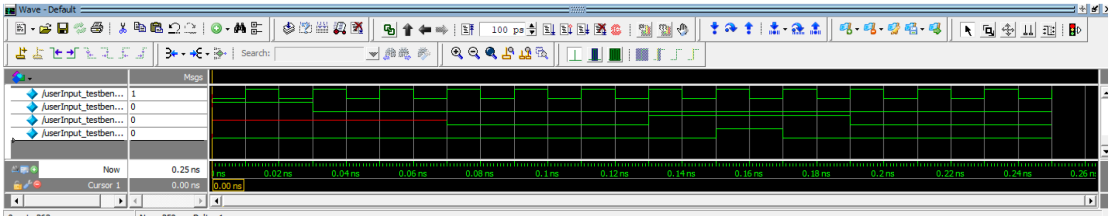

Figure 18: Simulation results of the design “userInput” in Task 2.

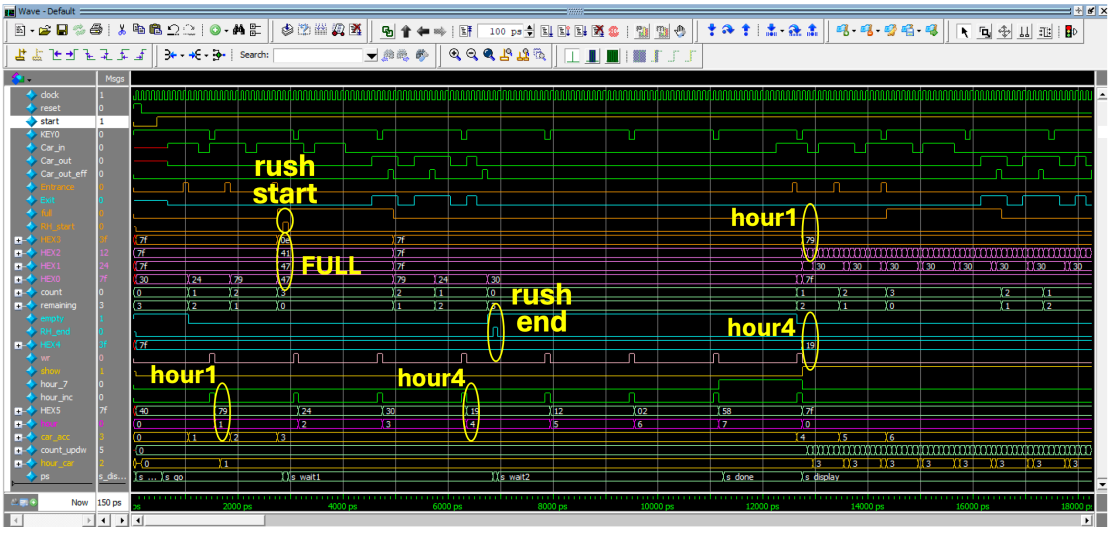

Figure 19: Simulation results of the design “top_ctrl_datapath” in Task 2.

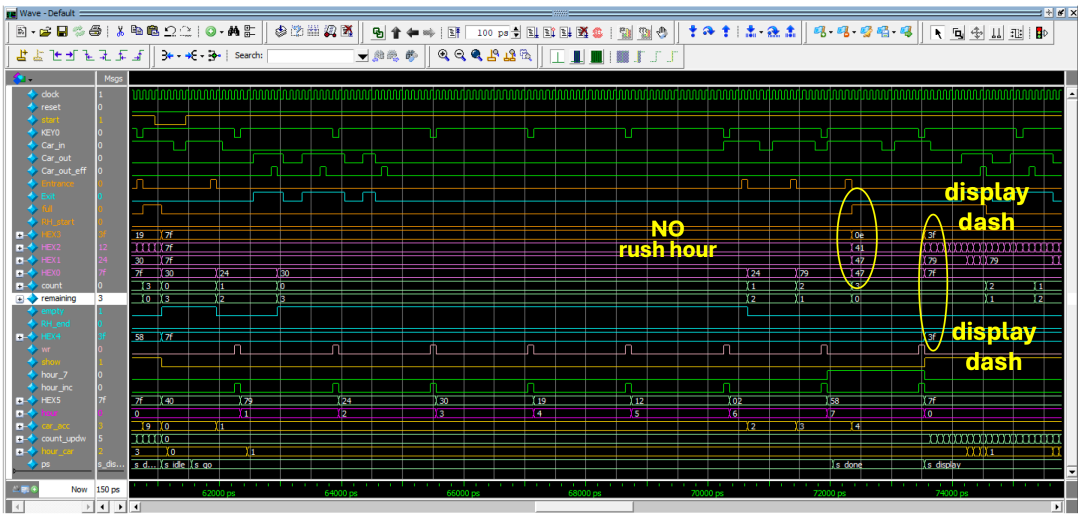

Figure 20: Simulation results of the design “top_ctrl_datapath” in Task 2.

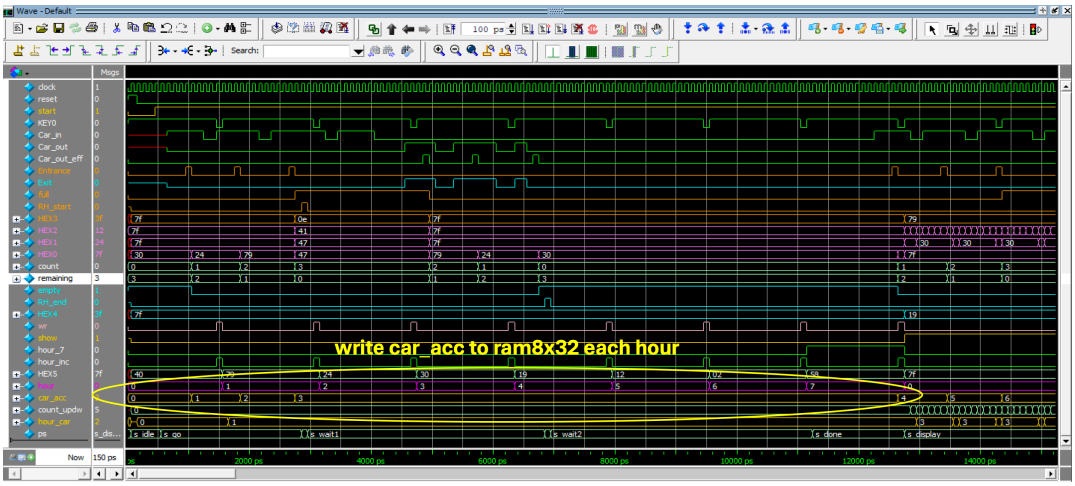

Figure 21: Simulation results of the design “top_ctrl_datapath” in Task 2.

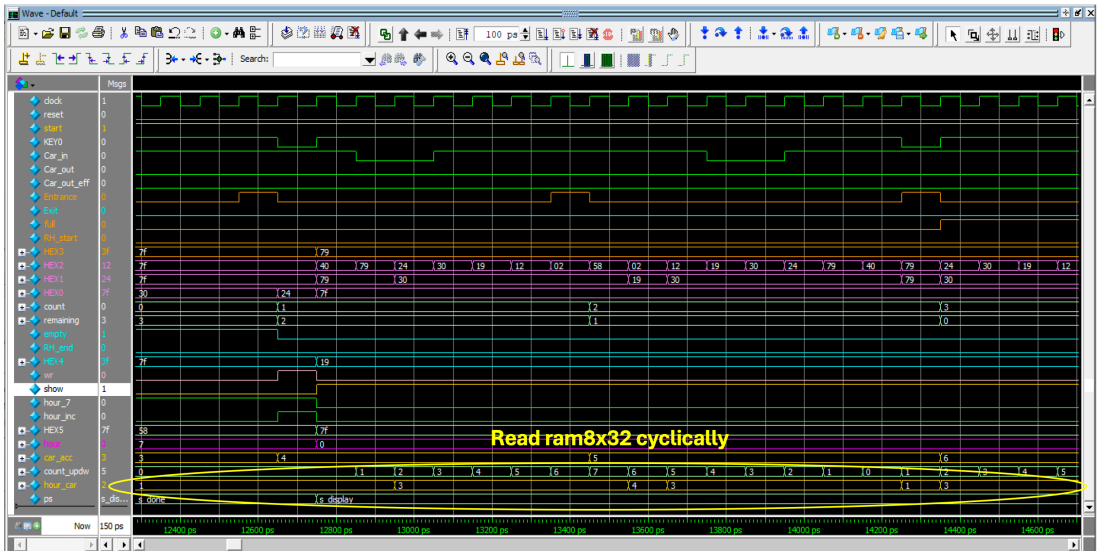

Figure 22: Simulation results of the design “top_ctrl_datapath” in Task 2.

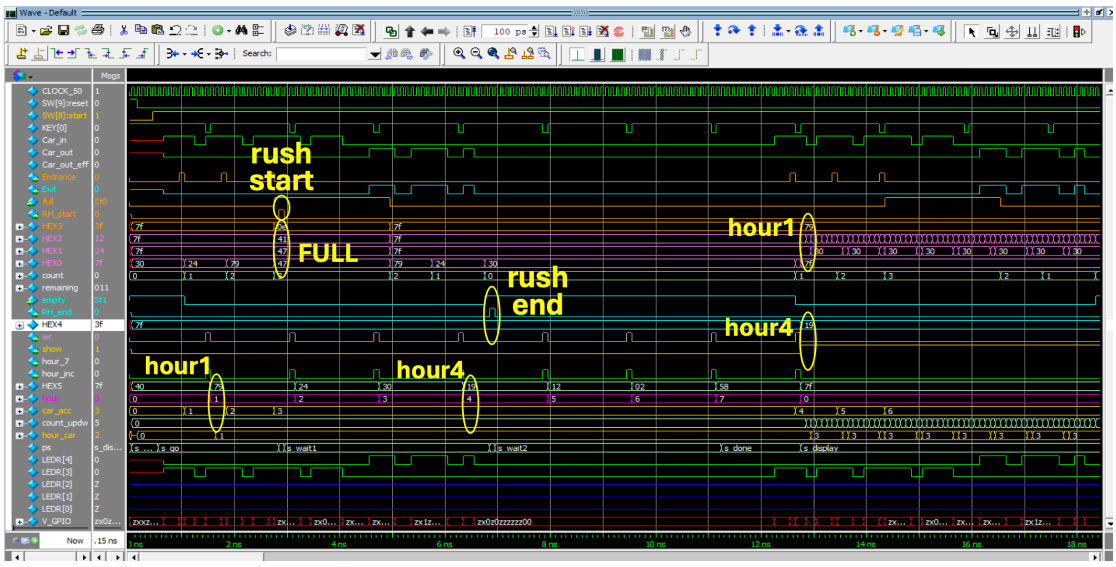

Figure 23: Simulation results of the design “DE1_SoC” in Task 2.

Figure 24: Simulation results of the design “DE1_SoC” in Task 2.

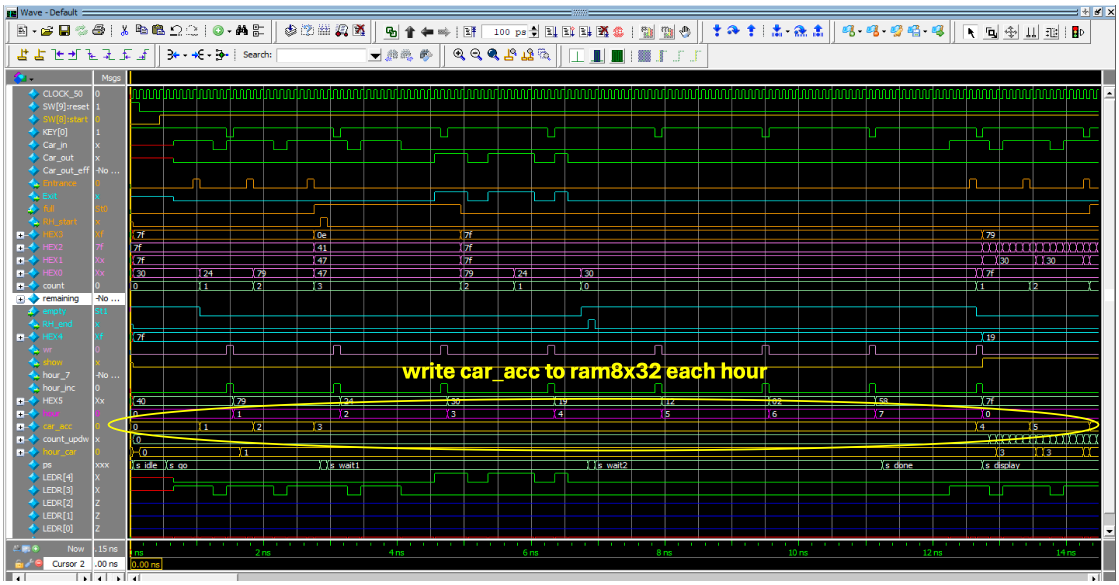

Figure 25: Simulation results of the design “DE1_SoC” in Task 2.

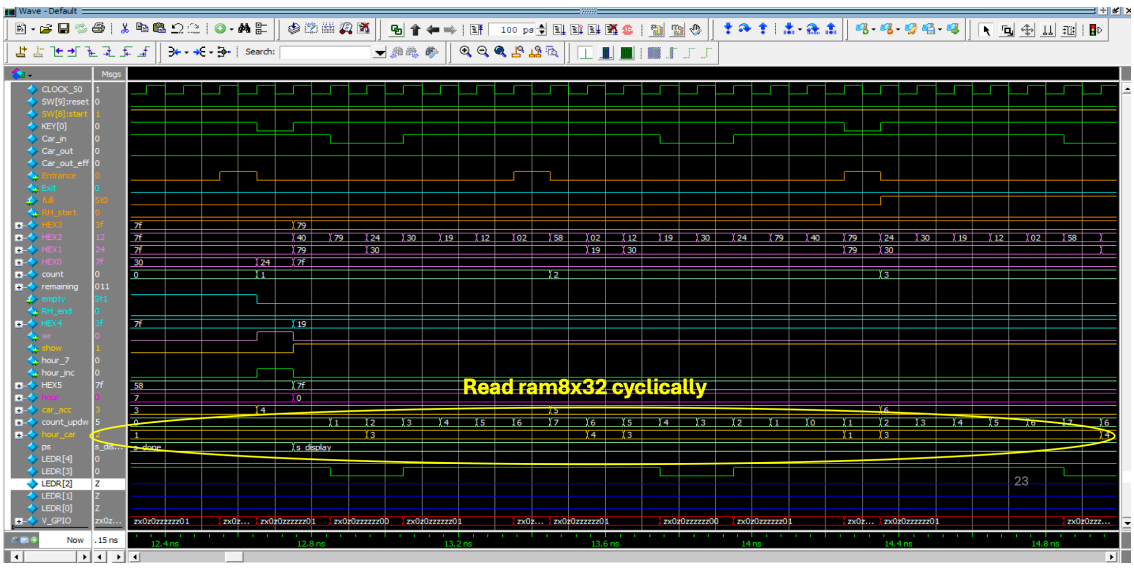

Figure 26: Simulation results of the design “DE1_SoC” in Task 2.

III. Final Product

I finished the design specified in Task 1 and Task 2. I organized each design as shown in the corresponding block diagrams, simulated on the Modelsim with testbenches, and verified the functions with DE1-SoC on the LabsLand. In this Lab, I finished design in from specifications to ASMD chart to SystemVerilog codes. The designs in this lab were well simulated on Modelsim and well-verified on LabsLand.

Besides, in the uploaded version of SystemVerilog files, for meeting the specification: at hour 8, cycle though the RAM at roughly 1 second, I add the "clock_divider" module to slow down the clock signal sent to the "count_updw" module such that the contents of RAM are read out cyclically with the period around 1 second.

IV. Appendix

1. Video for Task 1, https://youtu.be/IZmjTua7goQ
2. Video for Task 2 (Part 1), https://youtu.be/tAmPongdqfE
3. Video for Task 2 (Part 2), https://youtu.be/fEvy5qE5Xl0
4. SystemVerilog codes of Task 1, please see the following pages.
5. SystemVerilog codes of Task 2, please see the following pages.

SystemVerilog Codes of Task 1

```
1   /*=====================================================================*/
2   //  Name: Brian Chen
3   //  Date: 03-14-2025
4   //  EE/CSE371 LAB6--- System Verilog in the Real World (Task 1)
5   //  Device Under Test (dut) -- DE1_SoC
6   //  File Name: DE1_SoC.sv
7   /*=====================================================================*/
8   module DE1_SoC (HEX5, HEX4, HEX3, HEX2, HEX1, HEX0, KEY, SW, LEDR, V_GPIO, CLOCK_50);
9   // define ports
10      output logic [6:0]  HEX5, HEX4, HEX3, HEX2, HEX1, HEX0;
11      output logic [9:0]  LEDR;
12      input  logic [3:1]  KEY;
13      input  logic [9:0]  SW;
14      input logic CLOCK_50;
15      inout logic [35:23] V_GPIO;
16      logic hold, hold2;
17
18      top_cell top (.CLK(CLOCK_50), .RST(SW[9]), .Prs(~KEY),
19                    .Car_in(V_GPIO[23]), .Car_out(V_GPIO[24]),
20                    .Entrance(hold), .Exit(V_GPIO[33]),
21                    .HEX5, .HEX4, .HEX3, .HEX2, .HEX1, .HEX0);
22
23      // use signal_extender module to extend the time duration for Entrance
    gate.
24      signal_extender top_ext(.in(hold), .out(hold2), .reset(SW[9]), .clk(CLOCK_50));
25
26      assign V_GPIO[31] = hold2;
27      assign V_GPIO[26] = V_GPIO[28];  // specify the color of LED1.
28      assign V_GPIO[27] = V_GPIO[29];  // specify the color of LED2.
29      assign V_GPIO[32] = V_GPIO[30];  // specify the color of LED3.
30
31      // specify the color (0:green / 1:red) of the full indicaror LED.
32      assign V_GPIO[34] = V_GPIO[28] & V_GPIO[29] & V_GPIO[30];
33
34  endmodule // DE1_SoC
35
36  /*=====================================================================*/
37  //  Testbench
38  /*=====================================================================*/
39  module DE1_SoC_testbench();
40      logic [6:0]  HEX5, HEX4, HEX3, HEX2, HEX1, HEX0;
41      logic [9:0]  LEDR;
42      logic [3:1]  KEY;
43      logic [9:0]  SW;
44      logic CLOCK_50;
45      wire [35:23] V_GPIO;
46
47      logic arrival, departure;
48
49      // Set up a simulated clock: 50 MHz
50      parameter CLOCK_PERIOD = 20; // default timescale 1ns/1ns
51      initial begin
52          CLOCK_50 <= 0;
53          forever #(CLOCK_PERIOD/2) CLOCK_50 <= ~CLOCK_50;
54      end
55
56      integer i;
57
58      DE1_SoC dut (.HEX5, .HEX4, .HEX3, .HEX2, .HEX1, .HEX0, .KEY, .SW, .LEDR, .V_GPIO, .
    CLOCK_50);
59
60      assign V_GPIO[23] = arrival;
61      assign V_GPIO[24] = departure;
62
63      initial begin
64          SW[9] <= 1;
65          repeat(3)
66          @(posedge CLOCK_50);
67          SW[9] <= 0;
68
69          repeat(2)
70          @(posedge CLOCK_50);
71          {arrival, departure} = 2'b10;
72
73          repeat(5)
74          @(posedge CLOCK_50);
75          KEY[3:1] = 3'b101;
76
```

```
77              repeat(5)
78              @(posedge CLOCK_50);
79              KEY[3:1] = 3'b110;
80
81              repeat(5)
82              @(posedge CLOCK_50);
83              KEY[3:1] = 3'b011;
84
85              repeat(2)
86              @(posedge CLOCK_50);
87              {arrival, departure} = 2'b10;
88
89              repeat(5)
90              @(posedge CLOCK_50);
91              KEY[3:1] = 3'b110;
92
93              repeat(5)
94              @(posedge CLOCK_50);
95              KEY[3:1] = 3'b111;
96
97              repeat(5)
98              @(posedge CLOCK_50);
99              KEY[3:1] = 3'b110;
100
101             repeat(5)
102             @(posedge CLOCK_50);
103             KEY[3:1] = 3'b101;
104
105             repeat(5)
106             @(posedge CLOCK_50);
107             KEY[3:1] = 3'b111;
108
109             repeat(2)
110             @(posedge CLOCK_50);
111             {arrival, departure} = 2'b10;
112
113             repeat(5)
114             @(posedge CLOCK_50);
115             KEY[3:1] = 3'b101;
116
117             repeat(5)
118             @(posedge CLOCK_50);
119             KEY[3:1] = 3'b110;
120
121             repeat(5)
122             @(posedge CLOCK_50);
123             KEY[3:1] = 3'b011;
124
125             repeat(2)
126             @(posedge CLOCK_50);
127             {arrival, departure} = 2'b10;
128
129             repeat(5)
130             @(posedge CLOCK_50);
131             KEY[3:1] = 3'b101;
132
133             repeat(5)
134             @(posedge CLOCK_50);
135             KEY[3:1] = 3'b110;
136
137             repeat(5)
138             @(posedge CLOCK_50);
139             KEY[3:1] = 3'b011;
140
141             repeat(2)
142             @(posedge CLOCK_50);
143             {arrival, departure} = 2'b10;
144
145             repeat(5)
146             @(posedge CLOCK_50);
147             KEY[3:1] = 3'b101;
148
149             repeat(5)
150             @(posedge CLOCK_50);
151             KEY[3:1] = 3'b110;
152
153             repeat(5)
154             @(posedge CLOCK_50);
```

```
155             KEY[3:1] = 3'b011;
156 
157 
158             repeat(2)
159             @(posedge CLOCK_50);
160             {arrival, departure} = 2'b10;
161 
162             repeat(5)
163             @(posedge CLOCK_50);
164             KEY[3:1] = 3'b101;
165 
166             repeat(5)
167             @(posedge CLOCK_50);
168             KEY[3:1] = 3'b110;
169 
170             repeat(5)
171             @(posedge CLOCK_50);
172             KEY[3:1] = 3'b011;
173 
174             repeat(5)
175             @(posedge CLOCK_50);
176             {arrival, departure} = 2'b01;
177 
178             repeat(5)
179             @(posedge CLOCK_50);
180             {arrival, departure} = 2'b01;
181 
182             repeat(5)
183             @(posedge CLOCK_50);
184             {arrival, departure} = 2'b00;
185 
186             repeat(5)
187             @(posedge CLOCK_50);
188             {arrival, departure} = 2'b01;
189 
190             repeat(5)
191             @(posedge CLOCK_50);
192             {arrival, departure} = 2'b00;
193 
194             repeat(5)
195             @(posedge CLOCK_50);
196             {arrival, departure} = 2'b01;
197 
198             repeat(5)
199             @(posedge CLOCK_50);
200             {arrival, departure} = 2'b00;
201 
202             repeat(10)
203             @(posedge CLOCK_50);
204             KEY[3:1] = 3'b111;
205 
206             $stop;
207         end
208 
209     endmodule // DE1_SoC_testbench
```

```
1   /*=================================================================*/
2   //  Name: Brian Chen
3   //  Date: 03-14-2025
4   //  EE/CSE371 LAB6--- System Verilog in the Real World (Task 1)
5   //  Device Under Test (dut) -- top_cell
6   //  File Name: top_cell.sv
7   /*=================================================================*/
8   module top_cell (CLK, RST, Prs, Car_in, Car_out, Entrance, Exit, HEX5, HEX4, HEX3, HEX2,
    HEX1, HEX0);
9       input logic CLK, RST;
10     input logic [3:1] Prs;
11     input logic Car_in, Car_out;
12      output logic Entrance, Exit;
13     output logic [6:0] HEX5, HEX4, HEX3, HEX2, HEX1, HEX0;
14     logic [1:0] count;
15
16      passcode_all_counter K1 (.CLK, .RST, .Prs, .Car_in, .Car_out, .Entrance, .Exit, .count
    );
17      bcd7seg              K2 (.bcd(count), .HEX5, .HEX4, .HEX3, .HEX2, .HEX1, .HEX0);
18
19  endmodule
20  /*=================================================================*/
21  // Testbench
22  /*=================================================================*/
23  module top_cell_testbench();
24
25     logic CLK, RST, Car_in, Car_out, Entrance, Exit;
26     logic [3:1] Prs;
27     logic [6:0] HEX5, HEX4, HEX3, HEX2, HEX1, HEX0;
28
29      top_cell dut (.CLK, .RST, .Prs, .Car_in, .Car_out, .Entrance, .Exit,
30                    .HEX5, .HEX4, .HEX3, .HEX2, .HEX1, .HEX0);
31
32     parameter CLOCK_PERIOD = 20; // default timescale 1ns/1ns
33      initial begin
34          CLK <= 0;
35          forever #(CLOCK_PERIOD/2) CLK <= ~CLK;
36      end
37
38     initial begin
39         RST = 1;
40          repeat(2)
41        @(posedge CLK);
42         RST = 0;
43
44          repeat(2)
45        @(posedge CLK);
46         {Car_in, Car_out} = 2'b10;
47
48          repeat(5)
49        @(posedge CLK);
50        Prs = 3'b010;
51
52          repeat(5)
53        @(posedge CLK);
54        Prs = 3'b001;
55
56          repeat(5)
57        @(posedge CLK);
58        Prs = 3'b100;
59
60          repeat(2)
61        @(posedge CLK);
62         {Car_in, Car_out} = 2'b10;
63
64          repeat(5)
65        @(posedge CLK);
66        Prs = 3'b010;
67
68          repeat(5)
69        @(posedge CLK);
70        Prs = 3'b001;
71
72          repeat(5)
73        @(posedge CLK);
74        Prs = 3'b100;
75
76                repeat(2)
```

```
77          @(posedge CLK);
78           {Car_in, Car_out} = 2'b10;
79
80            repeat(5)
81          @(posedge CLK);
82          Prs = 3'b010;
83
84            repeat(5)
85          @(posedge CLK);
86          Prs = 3'b001;
87
88            repeat(5)
89          @(posedge CLK);
90          Prs = 3'b100;
91
92           repeat(2)
93          @(posedge CLK);
94           {Car_in, Car_out} = 2'b10;
95
96            repeat(5)
97          @(posedge CLK);
98          Prs = 3'b010;
99
100           repeat(5)
101         @(posedge CLK);
102         Prs = 3'b001;
103
104           repeat(5)
105         @(posedge CLK);
106         Prs = 3'b100;
107
108
109          repeat(2)
110         @(posedge CLK);
111          {Car_in, Car_out} = 2'b10;
112
113           repeat(5)
114         @(posedge CLK);
115         Prs = 3'b010;
116
117           repeat(5)
118         @(posedge CLK);
119         Prs = 3'b001;
120
121           repeat(5)
122         @(posedge CLK);
123         Prs = 3'b100;
124
125
126          repeat(2)
127         @(posedge CLK);
128          {Car_in, Car_out} = 2'b10;
129
130           repeat(5)
131         @(posedge CLK);
132         Prs = 3'b010;
133
134           repeat(5)
135         @(posedge CLK);
136         Prs = 3'b001;
137
138           repeat(5)
139         @(posedge CLK);
140         Prs = 3'b100;
141
142           repeat(5)
143         @(posedge CLK);
144          {Car_in, Car_out} = 2'b01;
145
146           repeat(5)
147         @(posedge CLK);
148          {Car_in, Car_out} = 2'b00;
149
150           repeat(5)
151         @(posedge CLK);
152          {Car_in, Car_out} = 2'b01;
153
154           repeat(5)
```

```
155         @(posedge CLK);
156          {Car_in, Car_out} = 2'b00;
157
158           repeat(5)
159         @(posedge CLK);
160          {Car_in, Car_out} = 2'b01;
161
162           repeat(10)
163         @(posedge CLK);
164         Prs = 3'b000;
165
166         $stop;
167       end
168   endmodule
```

```
1   /*=====================================================================*/
2   //  Name: Brian Chen
3   //  Date: 03-14-2025
4   //  EE/CSE371 LAB6--- System Verilog in the Real World (Task 1)
5   //  Device Under Test (dut) -- passcode_all_counter
6   //  File Name: passcode_all_counter .sv
7   /*=====================================================================*/
8   module passcode_all_counter (CLK, RST, Prs, Car_in, Car_out, Entrance, Exit, count);
9       input logic CLK, RST;
10      input logic [3:1] Prs;
11      input logic Car_in, Car_out;
12      output logic Entrance, Exit;
13      output logic [1:0] count;
14
15      logic Car_out_QD, Car_out_eff;
16      logic P1, P2;
17
18      // define the logic of Entrance
19      assign Entrance = Car_in & (P1 | P2) & (count < 3);
20      assign Exit = Car_out;
21
22      // add counter module, twoDFF module, and userInput module to passcode_all module
23      passcode_all m1 (.CLK(CLK), .RST(RST), .Prs(Prs), .P2(P2), .P1(P1));
24      counter      m2 (.clk(CLK), .reset(RST), .inc(Entrance), .dec(Car_out_eff), .count(
    count));
25      twoDFF       m3 (.CLK(CLK), .RST(RST), .D(Car_out), .Q(Car_out_QD));
26      userInput    m4 (.CLK(CLK), .RST(RST), .Qin(Car_out_QD), .Qeff(Car_out_eff));
27  endmodule
28
29  /*=====================================================================*/
30  // Testbench
31  /*=====================================================================*/
32  module passcode_all__counter_testbench();
33
34     logic CLK, RST, Car_in, Car_out, Entrance, Exit;
35     logic [3:1] Prs;
36     logic [1:0] count;
37
38     passcode_all_counter dut (.CLK, .RST, .Prs, .Car_in, .Car_out, .Entrance, .Exit, .count
    );
39
40     // clock generator
41     parameter CLOCK_PERIOD = 20; // default timescale 1ns/1ns
42      initial begin
43          CLK <= 0;
44          forever #(CLOCK_PERIOD/2) CLK <= ~CLK;
45      end
46
47     initial begin
48         RST = 1;
49          repeat(2)
50        @(posedge CLK);
51         RST = 0;
52
53          repeat(2)
54        @(posedge CLK);
55         {Car_in, Car_out} = 2'b10;
56
57          repeat(5)
58        @(posedge CLK);
59        Prs = 3'b010;
60
61          repeat(5)
62        @(posedge CLK);
63        Prs = 3'b001;
64
65          repeat(5)
66        @(posedge CLK);
67        Prs = 3'b100;
68
69          repeat(2)
70        @(posedge CLK);
71         {Car_in, Car_out} = 2'b10;
72
73          repeat(5)
74        @(posedge CLK);
75        Prs = 3'b010;
76
```

```
77            repeat(5)
78          @(posedge CLK);
79          Prs = 3'b001;
80
81            repeat(5)
82          @(posedge CLK);
83          Prs = 3'b100;
84
85                  repeat(2)
86          @(posedge CLK);
87           {Car_in, Car_out} = 2'b10;
88
89            repeat(5)
90          @(posedge CLK);
91          Prs = 3'b010;
92
93            repeat(5)
94          @(posedge CLK);
95          Prs = 3'b001;
96
97            repeat(5)
98          @(posedge CLK);
99          Prs = 3'b100;
100
101          repeat(2)
102         @(posedge CLK);
103          {Car_in, Car_out} = 2'b10;
104
105           repeat(5)
106         @(posedge CLK);
107         Prs = 3'b010;
108
109           repeat(5)
110         @(posedge CLK);
111         Prs = 3'b001;
112
113           repeat(5)
114         @(posedge CLK);
115         Prs = 3'b100;
116
117
118          repeat(2)
119         @(posedge CLK);
120          {Car_in, Car_out} = 2'b10;
121
122           repeat(5)
123         @(posedge CLK);
124         Prs = 3'b010;
125
126           repeat(5)
127         @(posedge CLK);
128         Prs = 3'b001;
129
130           repeat(5)
131         @(posedge CLK);
132         Prs = 3'b100;
133
134
135          repeat(2)
136         @(posedge CLK);
137          {Car_in, Car_out} = 2'b10;
138
139           repeat(5)
140         @(posedge CLK);
141         Prs = 3'b010;
142
143           repeat(5)
144         @(posedge CLK);
145         Prs = 3'b001;
146
147           repeat(5)
148         @(posedge CLK);
149         Prs = 3'b100;
150
151           repeat(5)
152         @(posedge CLK);
153          {Car_in, Car_out} = 2'b01;
154
```

```
155
156           repeat(5)
157         @(posedge CLK);
158         Prs = 3'b000;
159
160           repeat(5)
161         @(posedge CLK);
162         Prs = 3'b001;
163
164           repeat(5)
165         @(posedge CLK);
166         Prs = 3'b000;
167
168           repeat(5)
169         @(posedge CLK);
170         Prs = 3'b001;
171
172           repeat(5)
173         @(posedge CLK);
174         Prs = 3'b010;
175
176           repeat(5)
177         @(posedge CLK);
178         Prs = 3'b000;
179
180           repeat(5)
181         @(posedge CLK);
182         Prs = 3'b001;
183
184           repeat(5)
185         @(posedge CLK);
186         Prs = 3'b010;
187
188           repeat(5)
189         @(posedge CLK);
190         Prs = 3'b011;
191
192           repeat(8)
193         @(posedge CLK);
194         $stop;
195       end
196   endmodule
```

```
/*=====================================================================*/
//  Name: Brian Chen
//  Date: 03-14-2025
//  EE/CSE371 LAB6--- System Verilog in the Real World (Task 1)
//  Device Under Test (dut) -- passcode_all
//  File Name: passcode_all.sv
/*=====================================================================*/
module passcode_all (CLK, RST, Prs, P2, P1);
    input logic CLK, RST;
   input logic [3:1] Prs;
    output logic P2, P1;

   logic [3:1] QD, Peff;

    // add twoDFF module and userInput module to passcode module.
    twoDFF     n3 (.CLK(CLK), .RST(RST), .D(Prs[3]), .Q(QD[3]));
    twoDFF     n2 (.CLK(CLK), .RST(RST), .D(Prs[2]), .Q(QD[2]));
    twoDFF     n1 (.CLK(CLK), .RST(RST), .D(Prs[1]), .Q(QD[1]));
    userInput nn3 (.CLK(CLK), .RST(RST), .Qin(QD[3]), .Qeff(Peff[3]));
    userInput nn2 (.CLK(CLK), .RST(RST), .Qin(QD[2]), .Qeff(Peff[2]));
    userInput nn1 (.CLK(CLK), .RST(RST), .Qin(QD[1]), .Qeff(Peff[1]));

    passcode m4 (.CLK(CLK), .RST(RST), .B3(Peff[3]), .B2(Peff[2]), .B1(Peff[1]), .P2(P2),
.P1(P1));

endmodule

/*=====================================================================*/
// Testbench
/*=====================================================================*/

module passcode_all_testbench();

   logic CLK, RST, P2, P1;
   logic [3:1] Prs;

    passcode_all dut (.CLK, .RST, .Prs, .P2, .P1);

   parameter CLOCK_PERIOD = 20; // default timescale 1ns/1ns
    initial begin
        CLK <= 0;
        forever #(CLOCK_PERIOD/2) CLK <= ~CLK;
    end

   initial begin
       RST = 1;
        repeat(2)
      @(posedge CLK);
       RST = 0;
        repeat(5)
      @(posedge CLK);
      Prs = 3'b010;

        repeat(5)
      @(posedge CLK);
      Prs = 3'b001;

        repeat(5)
      @(posedge CLK);
      Prs = 3'b100;

        repeat(5)
      @(posedge CLK);
      Prs = 3'b000;

        repeat(5)
      @(posedge CLK);
      Prs = 3'b001;

        repeat(5)
      @(posedge CLK);
      Prs = 3'b000;

        repeat(5)
      @(posedge CLK);
      Prs = 3'b001;

        repeat(5)
```

```
78          @(posedge CLK);
79          Prs = 3'b010;
80
81            repeat(5)
82          @(posedge CLK);
83          Prs = 3'b000;
84
85            repeat(5)
86          @(posedge CLK);
87          Prs = 3'b001;
88
89            repeat(5)
90          @(posedge CLK);
91          Prs = 3'b010;
92
93            repeat(5)
94          @(posedge CLK);
95          Prs = 3'b011;
96
97            repeat(8)
98          @(posedge CLK);
99          $stop;
100       end
101   endmodule
```

```
/*=====================================================================*/
//  Name: Brian Chen
//  Date: 03-14-2025
//  EE/CSE371 LAB6--- System Verilog in the Real World (Task 1)
//  Device Under Test (dut) -- passcode
//  File Name: passcode.sv
/*=====================================================================*/
module passcode (CLK, RST, B3, B2, B1, P2, P1);
    input logic CLK, RST, B3, B2, B1;
    output logic P2, P1;

// State variables
    typedef enum logic [3:0] {
    s0, s1, s2, s3, s4, s5, s6, s7, s8, s9
    } state_t;
    state_t ps, ns;

    // assign ps_out = ps;

    // Next State logic
always_comb begin
    ns = s0; // Default to avoid latches
    case (ps)
        s0:    if (B1 == 1)
                   ns = s1;
               else if (B2 == 1)
                ns = s4;
               else if (B3 == 1)
                ns = s7;
               else ns = s0;

        s1:    if (B1 == 1)
                   ns = s2;
               else if ((B2 == 1) || (B3 == 1))
                ns = s8;
               else ns = s1;

        s2:    if (B2 == 1)
                   ns = s3;
               else if ((B1 == 1) || (B3 == 1))
                ns = s9;
               else ns = s2;

        s3:    ns = s0;

        s4:    if (B1 == 1)
                   ns = s5;
               else if ((B2 == 1) || (B3 == 1))
                ns = s8;
               else ns = s4;

        s5:    if (B3 == 1)
                   ns = s6;
               else if ((B1 == 1) || (B2 == 1))
                ns = s6;
               else ns = s5;

        s6:    ns = s0;

        s7:    if ((B1 == 0) && (B2 == 0) && (B3 == 0))
                ns = s7;
               else
               ns = s8;

        s8:    if ((B1 == 0) && (B2 == 0) && (B3 == 0))
                ns = s8;
               else
               ns = s9;

        s9:    ns = s0;

       default: ns = s0;
    endcase
end

    // Output logic
    always_comb begin
        {P2, P1} = 2'b00;  // Default to avoid latches
```

```
79            case (ps)
80                s0: {P2, P1} = 2'b00;
81                s1: {P2, P1} = 2'b00;
82                s2: {P2, P1} = 2'b00;
83                s3: {P2, P1} = 2'b10;
84                s4: {P2, P1} = 2'b00;
85                s5: {P2, P1} = 2'b00;
86                s6: {P2, P1} = 2'b01;
87                s7: {P2, P1} = 2'b00;
88                s8: {P2, P1} = 2'b00;
89                s9: {P2, P1} = 2'b00;
90                default: {P2, P1} = 2'b00;
91            endcase
92        end
93        // DFFs
94
95        always_ff @(posedge CLK or posedge RST) begin
96            if (RST)
97                ps <= s0;
98            else
99                ps <= ns;
100       end
101
102   endmodule
103
104   /*=============================================================================*/
105   // Testbench
106   /*=============================================================================*/
107
108   module passcode_testbench();
109
110      logic CLK, RST, B3, B2, B1, P2, P1;
111
112       passcode dut (.CLK, .RST, .B3, .B2, .B1, .P2, .P1);
113
114      parameter CLOCK_PERIOD = 20; // default timescale 1ns/1ns
115       initial begin
116           CLK <= 0;
117           forever #(CLOCK_PERIOD/2) CLK <= ~CLK;
118       end
119
120      initial begin
121          RST = 1;
122            repeat(2)
123          @(posedge CLK);
124           RST = 0;
125            repeat(8)
126          @(posedge CLK);
127          {B3, B2, B1} = 3'b010;
128
129            repeat(8)
130          @(posedge CLK);
131          {B3, B2, B1} = 3'b001;
132
133            repeat(8)
134          @(posedge CLK);
135          {B3, B2, B1} = 3'b100;
136
137            repeat(8)
138          @(posedge CLK);
139          {B3, B2, B1} = 3'b000;
140
141            repeat(8)
142          @(posedge CLK);
143          {B3, B2, B1} = 3'b001;
144
145            repeat(8)
146          @(posedge CLK);
147          {B3, B2, B1} = 3'b001;
148
149            repeat(8)
150          @(posedge CLK);
151          {B3, B2, B1} = 3'b010;
152
153          $stop;
154       end
155   endmodule
```

```
1
2   /*=================================================================*/
3   //  Name: Brian Chen
4   //  Date: 03-14-2025
5   //  EE/CSE371 LAB6--- System Verilog in the Real World (Task 1)
6   //  Device Under Test (dut) -- counter
7   //  File Name: counter.sv
8   /*=================================================================*/
9   module counter(clk, reset, inc, dec, count);
10
11     input logic clk, reset, inc ,dec;
12     output logic [1:0] count;
13
14     //counter logic
15
16     always_ff @(posedge clk or posedge reset) begin
17           if (reset)
18              count <= 2'b00; // reset count to 0
19
20           else begin
21              if (inc && !dec && count < 2'b11)
22                 count <= count + 1; // Increment if below max
23
24              else if (dec && !inc && count > 2'b00)
25                 count <= count - 1 ; // Decrement iff above 0
26
27           end
28     end
29
30  endmodule
31
32  /*=================================================================*/
33  // Testbench
34  /*=================================================================*/
35  module counter_testbench();
36
37     logic clk, reset, inc, dec;
38     logic [1:0] count;
39
40      counter dut (.clk, .reset, .inc, .dec, .count);
41
42     parameter CLOCK_PERIOD = 20; // default timescale 1ns/1ns
43      initial begin
44          clk <= 0;
45          forever #(CLOCK_PERIOD/2) clk <= ~clk;
46      end
47
48     initial begin
49         reset = 1;
50          repeat(2)
51        @(posedge clk);
52          reset = 0;
53          repeat(8)
54        @(posedge clk);
55        {inc, dec} = 2'b10;
56        repeat(8)
57        @(posedge clk);
58        {inc, dec} = 2'b01;
59        repeat(8)
60        @(posedge clk);
61        $stop;
62      end
63  endmodule
64
65
66
```

```
1   /*=================================================================*/
2   //  Name: Brian Chen
3   //  Date: 03-14-2025
4   //  EE/CSE371 LAB6--- System Verilog in the Real World (Task 1)
5   //  Device Under Test (dut) -- bcd7seg
6   //  File Name: bcd7seg.sv
7   /*=================================================================*/
8   module bcd7seg(bcd, HEX5, HEX4, HEX3, HEX2, HEX1, HEX0);
9   input logic [1:0] bcd;
10  output logic [6:0] HEX5, HEX4, HEX3, HEX2, HEX1, HEX0;
11
12  parameter [6:0] Blank = 7'b111_1111; // 7'h7f
13  parameter [6:0] A = 7'b000_1000; // 7'h08
14  parameter [6:0] C = 7'b100_0110; // 7'h46
15  parameter [6:0] E = 7'b000_0110; // 7'h06
16  parameter [6:0] F = 7'b000_1110; // 7'h0e
17  parameter [6:0] L = 7'b100_0111; // 7'h47
18  parameter [6:0] r = 7'b000_1111; // 7'h0f
19  parameter [6:0] U = 7'b100_0001; // 7'h41
20  parameter [6:0] Chr_0 = 7'b100_0000;  // 7'h40
21  parameter [6:0] Chr_1 = 7'b111_1001;  // 7'h79
22  parameter [6:0] Chr_2 = 7'b010_0100;  // 7'h24
23  parameter [6:0] Chr_3 = 7'b011_0000;  // 7'h30
24
25  always_comb begin
26  case(bcd)
27      3'b000:  {HEX5, HEX4, HEX3, HEX2, HEX1, HEX0} = {C, L, E, A, r, Chr_0};
28      3'b001:  {HEX5, HEX4, HEX3, HEX2, HEX1, HEX0} = {Blank, Blank, Blank, Blank, Blank,
    Chr_1};
29      3'b010:  {HEX5, HEX4, HEX3, HEX2, HEX1, HEX0} = {Blank, Blank, Blank, Blank, Blank,
    Chr_2};
30      3'b011:  {HEX5, HEX4, HEX3, HEX2, HEX1, HEX0} = {F, U, L, L, Blank, Chr_3};
31     default: {HEX5, HEX4, HEX3, HEX2, HEX1, HEX0} = {Blank, Blank, Blank, Blank, Blank,
    Blank};
32  endcase
33  end
34
35  endmodule
36
37  /*=================================================================*/
38  // Testbench
39  /*=================================================================*/
40
41  module bcd7seg_testbench();
42
43     logic [1:0] bcd;
44     logic [6:0] HEX5, HEX4, HEX3, HEX2, HEX1, HEX0;
45     integer i;
46
47      bcd7seg dut (.bcd, .HEX5, .HEX4, .HEX3, .HEX2, .HEX1, .HEX0);
48
49     initial begin
50          for (i = 0; i < 4; i++) begin
51            bcd = i;
52           #10;
53          end
54        $stop;
55      end
56  endmodule
```

```
1   /*=====================================================================*/
2   //  Name: Brian Chen
3   //  Date: 03-14-2025
4   //  EE/CSE371 LAB6--- System Verilog in the Real World (Task 1)
5   //  Device Under Test (dut) -- twoDFF
6   //  File Name: twoDFF.sv
7   /*=====================================================================*/
8   module twoDFF (CLK, RST, D, Q);
9   input logic CLK, RST;
10  input logic D;
11  output logic Q;
12  logic Q1;
13
14      // DFFs
15      always_ff @(posedge CLK or posedge RST) begin
16          if (RST) begin
17            Q  <= 0;
18           Q1 <=0;
19        end
20          else begin
21            Q  <= Q1;
22           Q1 <= D;
23         end
24      end
25
26  endmodule
27
```

```
1   /*=====================================================================*/
2   //  Name: Brian Chen
3   //  Date: 03-14-2025
4   //  EE/CSE371 LAB6--- System Verilog in the Real World (Task 1)
5   //  Device Under Test (dut) -- userInput
6   //  File Name: userInput.sv
7   /*=====================================================================*/
8   module userInput (CLK, RST, Qin, Qeff);
9   input logic CLK, RST;
10  input logic Qin;
11  output logic Qeff;
12
13  // State variables
14      typedef enum logic [1:0] {
15          start, s1, s2
16      } state_t;
17      state_t ps, ns;
18
19      // Next State logic
20  always_comb begin
21      ns = start; // Default to avoid latches
22      case (ps)
23          start: if (Qin == 1) ns = s1;
24                 else ns = start;
25
26          s1:    if (Qin == 1) ns = s2;
27                 else ns = start;
28
29          s2:    if (Qin == 1) ns = s2;
30                 else ns = start;
31         default: ns = start;
32      endcase
33  end
34
35      // Output logic
36      always_comb begin
37          Qeff = 0;  // Default to avoid latches
38          case (ps)
39              start: Qeff = 0;
40              s1:    Qeff = 1;
41              s2:    Qeff = 0;
42              default: Qeff = 0;
43          endcase
44      end
45
46      // DFFs
47      always_ff @(posedge CLK or posedge RST) begin
48          if (RST)
49              ps <= start;
50          else
51              ps <= ns;
52      end
53
54  endmodule
55
```

```
1   /*=================================================================*/
2   //  Name: Brian Chen
3   //  Date: 03-14-2025
4   //  EE/CSE371 LAB6--- System Verilog in the Real World (Task 1)
5   //  Device Under Test (dut) -- signal_extender
6   //  File Name: signal_extender.sv
7   /*=================================================================*/
8   // Takes in a short pulse signal, extends the length of the pulse
9   // by 50,000,000 clock cycles for use with 3D simulator gates.
10
11  //in: 1-bit original input pulse
12  //out: 1-bit extended output pulse
13  //reset: 1-bit reset signal that sets counter to 0
14  //clk: 1-bit input clock controlling system
15  module signal_extender (in, out, reset, clk);
16
17     input logic in, reset, clk;
18     output logic out;
19     logic [31:0] count = 32'b0;
20     assign out = (count > 0);
21
22     always_ff @(posedge clk) begin
23         if (reset) begin
24             count <= 0;
25         end
26         else begin
27            if (in) count <= 50000000;
28            else if (count > 0) count <= count - 1;
29         end
30     end
31  endmodule
32
```

SystemVerilog Codes of Task 2

```
/*=================================================================*/
//  Name: Brian Chen
//  Date: 03-15-2025
//  EE/CSE371 LAB6--- System Verilog in the Real World (Task 2)
//  Device under Test (dut) --- DE1_SoC
//  File Name: DE1_SoC.sv
/*=================================================================*/
module DE1_SoC (CLOCK_50, HEX0, HEX1, HEX2, HEX3, HEX4, HEX5, KEY, SW, LEDR, V_GPIO);

   // define ports
   input  logic CLOCK_50;
   output logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
   input  logic [3:0] KEY;
   input  logic [9:0] SW;
   output logic [9:0] LEDR;
   inout  logic [35:23] V_GPIO;

   logic hold, hold2;

   // Generate clk off of CLOCK_50, whichClock picks rate.
   logic [31:0] div_clk;

   parameter whichClock = 23; // (50 MHz / 2^24 = 2.98 Hz)
   /*=================================================================*/
   //     clock_divider (File Name: clock_divider.sv)
   /*=================================================================*/
    clock_divider cdiv (.clock(CLOCK_50), .reset(SW[9]), .divided_clocks(div_clk));
   /*-----------------------------------------------------------------*/
   //     Clock selection;
   //     allows for easy switching between simulation and board clocks
   /*-----------------------------------------------------------------*/
   logic clkSelect;

    //Uncomment ONE of the following two lines depending on intention
   //*****************************************************************
    //assign clkSelect = CLOCK_50;            // for simulation
    assign clkSelect = div_clk[whichClock]; // for DE1_SoC  board
   //*****************************************************************


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

   // My connections
   assign V_GPIO[26] = V_GPIO[28];  // LED parking 1
   assign V_GPIO[27] = V_GPIO[29];  // LED parking 2
   assign V_GPIO[32] = V_GPIO[30];  // LED parking 3
   assign V_GPIO[34] = V_GPIO[28] & V_GPIO[29] & V_GPIO[30];   // LED full

   top_ctrl_datapath top (
       .clock(clkSelect),
       .reset(SW[9]),
       .start(SW[8]),
       .KEY0(KEY[0]),
       .Car_in(V_GPIO[23]),
       .Car_out(V_GPIO[24]),
       .Entrance(hold),    //<-----
       .Exit(V_GPIO[33]),
       .HEX5,
       .HEX4,
       .HEX3,
       .HEX2,
       .HEX1,
       .HEX0);

   // Use signal_extender to extend the open time of entrance gate.
   signal_extender top_ext(.in(hold), .out(hold2), .reset(SW[9]), .clk(CLOCK_50));
   assign V_GPIO[31] = hold2;

endmodule  // DE1_SoC

/*=================================================================*/
//  Testbench
/*=================================================================*/
```

```
  `timescale 1 ps / 1 ps
  module DE1_SoC_testbench();
      logic CLOCK_50;
      logic [6:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
      logic [3:0] KEY;
      logic [9:0] SW;
      logic [9:0] LEDR;
      wire [35:23] V_GPIO;

      logic Car_in, Car_out;
      logic KEY0, clock, reset, start;

      integer i, j, k;

      DE1_SoC dut (.*);

      assign V_GPIO[23] = Car_in;
      assign V_GPIO[24] = Car_out;
      assign KEY[0] = KEY0;
      assign SW[9] = reset;
      assign SW[8] = start;
      assign CLOCK_50 = clock;
      assign CLK = CLOCK_50;


      // clock generator
      parameter CLK_Period = 100;

      initial begin
          clock <= 1'b0;
         forever #(CLK_Period/2) clock <= ~clock;
      end

      // set test patterns for KEY0.
      initial begin
          for (i = 0; i < 100; i++) begin
              KEY0 <= 1;
              repeat(15)
              @(posedge clock);

              KEY0 <= 0;
              repeat(3)
              @(posedge clock);
          end
      end

      initial begin
          reset <= 1;

          repeat(2)
          @(posedge clock);
          reset <= 0;
      end

      // set test patterns for start.
      initial begin
          for (j = 0; j < 30; j++) begin
              start <= 0;
              repeat(5)
              @(posedge clock);

              start <= 1;
              repeat(300)
              @(posedge CLK);
          end
      end

      // set test patterns for Car_in and Car_out.
      initial begin
          for (k = 0; k < 12; k++) begin
              repeat(7)
              @(posedge CLK);
              {Car_in, Car_out} = 2'b10;

              repeat(6)
              @(posedge CLK);
              {Car_in, Car_out} = 2'b00;

```

```
157                 repeat(2)
158                 @(posedge CLK);
159                 {Car_in, Car_out} = 2'b10;
160 
161                 repeat(7)
162                 @(posedge CLK);
163                 {Car_in, Car_out} = 2'b00;
164 
165                 repeat(2)
166                 @(posedge CLK);
167                 {Car_in, Car_out} = 2'b10;
168 
169                 repeat(9)
170                 @(posedge CLK);
171                 {Car_in, Car_out} = 2'b00;
172 
173                 repeat(2)
174                 @(posedge CLK);
175                 {Car_in, Car_out} = 2'b10;
176 
177                 repeat(6)
178                 @(posedge CLK);
179                 {Car_in, Car_out} = 2'b00;
180 
181                 repeat(5)
182                 @(posedge CLK);
183                 {Car_in, Car_out} = 2'b01;
184 
185                 repeat(5)
186                 @(posedge CLK);
187                 {Car_in, Car_out} = 2'b00;
188 
189                 repeat(3)
190                 @(posedge CLK);
191                 {Car_in, Car_out} = 2'b01;
192 
193                 repeat(70)
194                 @(posedge CLK);
195                 {Car_in, Car_out} = 2'b00;
196 
197                 repeat(3)
198                 @(posedge CLK);
199                 {Car_in, Car_out} = 2'b01;
200 
201                 repeat(2)
202                 @(posedge CLK);
203                 {Car_in, Car_out} = 2'b00;
204                 repeat(50)
205                 @(posedge clock);
206             end
207             $stop;
208         end
209 
210     endmodule // DE1_SoC_testbench
211 
```

```
/*=====================================================================*/
//  Name: Brian Chen
//  Date: 03-15-2025
//  EE/CSE371 LAB6--- System Verilog in the Real World (Task 2)
//  Device under Test (dut) --- top_ctrl_datapath
//  File Name: top_ctrl_datapath.sv
/*=====================================================================*/
module top_ctrl_datapath (
    input logic clock,
    input logic reset,
    input logic start,
    input logic KEY0,          // press to increase hour by 1
    input logic Car_in,
    input logic Car_out,
    output logic Entrance,     // Extrance signal to 3D simulator
    output logic Exit,         // Exit signal to 3D simulator
    output logic [6:0] HEX5,   // currnt hour
    output logic [6:0] HEX4,   // the 1st rush hour end
    output logic [6:0] HEX3,   // the 1st rush hour start
    output logic [6:0] HEX2,   // RAM address
    output logic [6:0] HEX1,   // RAM contents
    output logic [6:0] HEX0);  // remaining parking lots

    logic full, empty, hour_7, hour_inc, wr, RH_start, RH_end, RH_flag, show, clear_reg;

    // Control Unit
    parkinglot_ctrl ctrl (
        .clock,
        .reset,
        .start,
        .KEY0,
        .full,         // to indicate the parking lot is full
        .empty,        // to indicate the parking lot is empty
        .hour_7,       // to indicate currnt hour is 7
        .hour_inc,     // to indicate hour shoud be increased by 1
        .wr,           // write enable signal
        .RH_start,     // to indicate the 1st rush hour start is found
        .RH_end,       // to indicate the 1st rush hour end is found
        .RH_flag,
        .show,         // show the rush hour and track RAM
        .clear_reg);

    // Datapath Unit
    top_datapath datapath (
        .clock,
        .reset,
        .clear_reg,
        .Car_in,
        .Car_out,
        .hour_inc,
        .wr,
        .RH_start,
        .RH_end,
        .RH_flag,
        .show,
        .Entrance,
        .Exit,
        .full,
        .empty,
        .hour_7,
        .HEX5,
        .HEX4,
        .HEX3,
        .HEX2,
        .HEX1,
        .HEX0);

endmodule
/*=====================================================================*/
//    Testbench for binsearch_top
/*=====================================================================*/
`timescale 1 ps / 1 ps
module top_ctrl_datapath_testbench();

    logic clock, CLK;
    logic reset;
    logic start;
    logic KEY0;
```

```
79        logic Car_in;
80        logic Car_out;
81        logic Entrance;
82        logic Exit;
83        logic [6:0] HEX5, HEX4, HEX3, HEX2, HEX1, HEX0;
84
85        assign CLK = clock;
86
87        integer i, j, k;
88
89        top_ctrl_datapath dut (.*);
90
91        // clock generator
92        parameter CLK_Period = 100;
93
94        initial begin
95            clock <= 1'b0;
96           forever #(CLK_Period/2) clock <= ~clock;
97        end
98
99        initial begin
100           for (i = 0; i < 100; i++) begin
101               KEY0 <= 1;
102               repeat(15)
103               @(posedge clock);
104
105               KEY0 <= 0;
106               repeat(1)
107               @(posedge clock);
108           end
109       end
110
111       initial begin
112           reset <= 1;
113
114           repeat(2)
115           @(posedge clock);
116           reset <= 0;
117       end
118
119
120       initial begin
121           for (j = 0; j < 30; j++) begin
122               start <= 0;
123               repeat(5)
124               @(posedge clock);
125
126               start <= 1;
127               repeat(300)
128               @(posedge CLK);
129           end
130       end
131
132       initial begin
133           for (k = 0; k < 12; k++) begin
134               repeat(7)
135               @(posedge CLK);
136               {Car_in, Car_out} = 2'b10;
137
138               repeat(6)
139               @(posedge CLK);
140               {Car_in, Car_out} = 2'b00;
141
142               repeat(2)
143               @(posedge CLK);
144               {Car_in, Car_out} = 2'b10;
145
146               repeat(7)
147               @(posedge CLK);
148               {Car_in, Car_out} = 2'b00;
149
150               repeat(2)
151               @(posedge CLK);
152               {Car_in, Car_out} = 2'b10;
153
154               repeat(9)
155               @(posedge CLK);
156               {Car_in, Car_out} = 2'b00;
```

```
157
158                 repeat(2)
159                 @(posedge CLK);
160                 {Car_in, Car_out} = 2'b10;
161
162                 repeat(6)
163                 @(posedge CLK);
164                 {Car_in, Car_out} = 2'b00;
165
166                 repeat(5)
167                 @(posedge CLK);
168                 {Car_in, Car_out} = 2'b01;
169
170                 repeat(5)
171                 @(posedge CLK);
172                 {Car_in, Car_out} = 2'b00;
173
174                 repeat(3)
175                 @(posedge CLK);
176                 {Car_in, Car_out} = 2'b01;
177
178                 repeat(7)
179                 @(posedge CLK);
180                 {Car_in, Car_out} = 2'b00;
181
182                 repeat(3)
183                 @(posedge CLK);
184                 {Car_in, Car_out} = 2'b01;
185
186                 repeat(2)
187                 @(posedge CLK);
188                 {Car_in, Car_out} = 2'b00;
189                 repeat(50)
190                 @(posedge clock);
191             end
192             $stop;
193         end
194
195     endmodule
```

```
/*=====================================================================*/
//  Name: Brian Chen
//  Date: 03-15-2025
//  EE/CSE371 LAB6--- System Verilog in the Real World (Task 2)
//  Device under Test (dut) --- top_datapath
//  File Name: top_datapath.sv
/*=====================================================================*/
module top_datapath (
    input logic clock,
    input logic reset,
    input logic clear_reg,
    input logic Car_in,
    input logic Car_out,
    input logic hour_inc,
    input logic wr,
    input logic RH_start, // indicate the 1st rush hour start
    input logic RH_end,   // indicate the 1st rush hour end
    input logic RH_flag,
    input logic show,      // display rush hour and track RAM contents
    output logic Entrance,
    output logic Exit,
    output logic full,     // parking lot status
    output logic empty,    // parking lot status
    output logic hour_7,   // hour == 7
    output logic [6:0] HEX5, // current hour
    output logic [6:0] HEX4,  // the 1st rush hour end
    output logic [6:0] HEX3,  // the 1st rush hour start
    output logic [6:0] HEX2,  // RAM address
    output logic [6:0] HEX1,  // RAM content
    output logic [6:0] HEX0); // remaining parking lots

    logic [15:0] car_acc;
    logic [1:0] count;
    logic [2:0] count_updw;  // read address
    logic [2:0] hour;
    logic [15:0] hour_car;

    // instantiate the modules
    simple3D simple
        (.CLK(clock), .RST(reset), .clear_reg, .Car_in, .Car_out,
         .Entrance, .Exit, .count);

    parkinglot_datapath  parkinglot
        (.clock, .reset, .clear_reg, .hour_inc, .wr, .RH_start,
         .RH_end, .RH_flag, .show, .count, .full, .empty, .hour_7,
         .hour, .count_updw, .hour_car,
         .HEX5, .HEX4, .HEX3, .HEX2, .HEX1, .HEX0);

    ram8x16 m1
        (.clock, .data(car_acc), .rdaddress(count_updw),
         .wraddress(hour), .wren(wr), .q(hour_car));

    // implement a accumulator.
    always_ff @ (posedge clock or posedge reset or posedge clear_reg)
        if (reset || clear_reg)
            car_acc <= 16'b0;
        else
            car_acc <= car_acc + Entrance;

endmodule
/*=====================================================================*/
//   Testbench for parkinglot_datapath
/*=====================================================================*/
`timescale 1 ps / 1 ps
module top_datapath_testbench();
    logic clock;
    logic reset;
    logic clear_reg;
    logic Car_in;
    logic Car_out;
    logic hour_inc;
    logic wr;
    logic RH_start;
    logic RH_end;
    logic RH_flag;
    logic show;
    logic Entrance;
    logic Exit;
```

```
79        logic [1:0] count;
80        logic full;
81        logic empty;
82        logic hour_7;
83        logic [6:0] HEX5;
84        logic [6:0] HEX4;
85        logic [6:0] HEX3;
86        logic [6:0] HEX2;
87        logic [6:0] HEX1;
88        logic [6:0] HEX0;
89        logic CLK, RST;
90
91        integer i;
92
93        top_datapath dut (.*);
94
95        // clock generator
96
97        parameter CLK_Period = 100;
98
99        initial begin
100           clock <= 1'b0;
101           forever #(CLK_Period/2) clock <= ~clock;
102       end
103
104       assign CLK = clock;
105       assign reset = RST;
106
107       initial begin
108           RST <= 1;
109
110           repeat(1)
111           @(posedge clock);
112           RST <= 0;
113           wr <= 1;
114           repeat(1)
115           @(posedge clock);
116
117           // use for loop to generate test patterns
118
119           for (i = 0; i < 2**5; i++)
120           begin
121               {show, hour_inc, RH_start, RH_end, RH_flag} = i;
122               repeat(2)
123               @(posedge clock);
124           end
125
126           repeat(2)
127           @(posedge clock);
128
129       end
130
131       initial begin
132           RST = 1;
133           repeat(2)
134           @(posedge CLK);
135           RST = 0;
136
137           repeat(2)
138           @(posedge CLK);
139           {Car_in, Car_out} = 2'b10;
140
141           repeat(2)
142           @(posedge CLK);
143           {Car_in, Car_out} = 2'b10;
144
145           repeat(2)
146           @(posedge CLK);
147           {Car_in, Car_out} = 2'b10;
148
149           repeat(2)
150           @(posedge CLK);
151           {Car_in, Car_out} = 2'b10;
152
153           repeat(2)
154           @(posedge CLK);
155           {Car_in, Car_out} = 2'b10;
156
```

```
157             repeat(2)
158             @(posedge CLK);
159             {Car_in, Car_out} = 2'b10;
160
161             repeat(2)
162             @(posedge CLK);
163             {Car_in, Car_out} = 2'b01;
164
165             repeat(2)
166             @(posedge CLK);
167             {Car_in, Car_out} = 2'b00;
168
169             repeat(2)
170             @(posedge CLK);
171             {Car_in, Car_out} = 2'b01;
172
173             repeat(2)
174             @(posedge CLK);
175             {Car_in, Car_out} = 2'b00;
176
177             repeat(2)
178             @(posedge CLK);
179             {Car_in, Car_out} = 2'b01;
180
181             repeat(2)
182             @(posedge CLK);
183             {Car_in, Car_out} = 2'b01;
184
185             repeat(2)
186             @(posedge CLK);
187             {Car_in, Car_out} = 2'b00;
188
189             repeat(2)
190             @(posedge CLK);
191             {Car_in, Car_out} = 2'b01;
192
193             repeat(2)
194             @(posedge CLK);
195             {Car_in, Car_out} = 2'b00;
196
197             repeat(2)
198             @(posedge CLK);
199             {Car_in, Car_out} = 2'b01;
200
201             repeat(2)
202             @(posedge CLK);
203             {Car_in, Car_out} = 2'b00;
204
205         $stop;
206       end
207   endmodule
```

```
1   /*=====================================================================*/
2   //  Name: Brian Chen
3   //  Date: 03-15-2025
4   //  EE/CSE371 LAB6--- System Verilog in the Real World (Task 2)
5   //  Device under Test (dut) --- simple3D
6   //  File Name: simple3D.sv
7   /*=====================================================================*/
8   module simple3D (CLK, RST, clear_reg, Car_in, Car_out, Entrance, Exit, count, full, empty);
9       input logic CLK, RST;
10      input logic clear_reg;
11      input logic Car_in, Car_out;
12      output logic Entrance, Exit;
13      output logic [1:0] count;  // count the number of cars in the parking lot
14      output logic full, empty;  // status of the parking lot
15
16      logic Car_out_QD, Car_out_eff;
17      logic reset;
18      logic Car_in_eff;
19
20      assign Entrance = Car_in_eff & (count < 3);  //<----specify the Entrance signal
21      assign Exit = Car_out;
22      assign full  = (count == 2'b11) ? 1'b1 : 1'b0;
23      assign empty = (count == 2'b00) ? 1'b1 : 1'b0;
24      assign reset = RST | clear_reg;
25
26      // instantiate the modules
27      counter      m2 (.clk(CLK), .reset(reset), .inc(Entrance), .dec(Car_out_eff), .count(
    count));
28      twoDFF       m3 (.CLK(CLK), .RST(reset), .D(Car_out), .Q(Car_out_QD));
29      userInput    m4 (.CLK(CLK), .RST(reset), .Qin(Car_out_QD), .Qeff(Car_out_eff));
30
31      twoDFF       m5 (.CLK(CLK), .RST(reset), .D(Car_in), .Q(Car_in_QD));
32      userInput    m6 (.CLK(CLK), .RST(reset), .Qin(Car_in_QD), .Qeff(Car_in_eff));
33  endmodule
34
35  /*=====================================================================*/
36  // Testbench
37  /*=====================================================================*/
38  module simple3D_testbench();
39
40      logic CLK, RST, Car_in, Car_out, Entrance, Exit;
41      logic clear_reg;
42      logic [1:0] count;
43
44      simple3D dut (.CLK, .RST, .clear_reg, .Car_in, .Car_out, .Entrance, .Exit, .count);
45
46
47      // clock generator
48      parameter CLOCK_PERIOD = 20; // default timescale 1ns/1ns
49
50      initial begin
51          CLK <= 0;
52          forever #(CLOCK_PERIOD/2) CLK <= ~CLK;
53      end
54
55      initial begin
56          RST = 1;
57          repeat(2)
58          @(posedge CLK);
59          RST = 0;
60
61          repeat(2)
62          @(posedge CLK);
63          {Car_in, Car_out} = 2'b10;
64
65          repeat(2)
66          @(posedge CLK);
67          {Car_in, Car_out} = 2'b00;
68
69          repeat(2)
70          @(posedge CLK);
71          {Car_in, Car_out} = 2'b10;
72
73          repeat(2)
74          @(posedge CLK);
75          {Car_in, Car_out} = 2'b00;
76
77          repeat(2)
```

```
78            @(posedge CLK);
79            {Car_in, Car_out} = 2'b10;
80
81            repeat(2)
82            @(posedge CLK);
83            {Car_in, Car_out} = 2'b00;
84
85            repeat(2)
86            @(posedge CLK);
87            {Car_in, Car_out} = 2'b01;
88
89            repeat(2)
90            @(posedge CLK);
91            {Car_in, Car_out} = 2'b00;
92
93            repeat(2)
94            @(posedge CLK);
95            {Car_in, Car_out} = 2'b01;
96
97            repeat(2)
98            @(posedge CLK);
99            {Car_in, Car_out} = 2'b00;
100           repeat(8)
101           @(posedge CLK);
102           $stop;
103       end
104   endmodule
```

```
/*=====================================================================*/
//  Name: Brian Chen
//  Date: 03-15-2025
//  EE/CSE371 LAB6--- System Verilog in the Real World (Task 2)
//  Device under Test (dut) --- parkinglot_datapath
//  File Name: parkinglot_datapath.sv
/*=====================================================================*/
module parkinglot_datapath (
   input logic clock,
   input logic reset,
   input logic clear_reg,
   input logic hour_inc,
   input logic wr,
   input logic RH_start,
   input logic RH_end,
   input logic RH_flag,
   input logic show,
   input logic [1:0] count,
   output logic full,
   output logic empty,
   output logic hour_7,
   output logic [2:0] hour,          // current hour
   output logic [2:0] count_updw,    // Read addresses
   input logic [15:0] hour_car,      // number of cars in the parking lot
   output logic [6:0] HEX5,          // current hour
   output logic [6:0] HEX4,          // the end of rush hour
   output logic [6:0] HEX3,          // the start of rush hour
   output logic [6:0] HEX2,          // read address
   output logic [6:0] HEX1,          // ram content
   output logic [6:0] HEX0);         // remaining parking lots

   logic wren;
   logic [15:0] car_acc;             // number of entrance till now
   logic [2:0] rush_hour_start;      // the 1st rush hour start
   logic [2:0] rush_hour_end;        // the 1st rush hour end
   logic rush_hour_flag_start;
   logic rush_hour_flag_end;
   logic [2:0] remaining;
   logic [6:0] HEX5_mux, HEX4_mux, HEX3_mux, HEX2_mux, HEX1_mux, HEX0_mux;

   logic select;
   logic [6:0] out13, out14, out20, out21, out22, out23, out24;


   logic hour_inc_QD, hour_inc_eff;


   assign wren = wr;
   assign hour_7 = (hour == 7) ? 1'b1 : 1'b0;
   assign full  = (count == 2'b11) ? 1'b1 : 1'b0;
   assign empty = (count == 2'b00) ? 1'b1 : 1'b0;
   assign remaining = 3'b11 - count;


   // define registers in the datapath.

   always_ff @(posedge clock or posedge reset or posedge clear_reg)
       if (reset) begin
           hour <= 3'b0;
           car_acc <= 16'b0;
           rush_hour_start <= 3'b0;
           rush_hour_end <= 3'b0;
           rush_hour_flag_start <= 1'b0;
           rush_hour_flag_end <= 1'b0;
       end
       else if (clear_reg) begin
           hour <= 3'b0;
           car_acc <= 16'b0;
           rush_hour_start <= 3'b0;
           rush_hour_end <= 3'b0;
           rush_hour_flag_start <= 1'b0;
           rush_hour_flag_end <= 1'b0;
       end
       else begin
           if (hour_inc_eff) hour <= hour + 3'b1;
           if (RH_start) rush_hour_start <= hour;
           if (RH_flag)  rush_hour_flag_start <= 1'b1;
           if (RH_end) begin
```

```
                rush_hour_end <= hour;
                rush_hour_flag_end <= 1'b1;
            end
        end
    //====================
     twoDFF       DFF2   (.CLK(clock), .RST(reset), .D(hour_inc), .Q(hour_inc_QD));
     userInput    userin (.CLK(clock), .RST(reset), .Qin(hour_inc_QD), .Qeff(hour_inc_eff));
    //====================


     // define select signal for HEX3 and HEX4.
     assign sel_HEX34 = rush_hour_flag_start && rush_hour_flag_end;

     data7seg d5 (.data({1'b0, hour}),        .seg(HEX5_mux));
     data7seg d4 (.data(rush_hour_end),       .seg(HEX4_mux));
     data7seg d3 (.data(rush_hour_start),     .seg(HEX3_mux));
     data7seg d2 (.data({1'b0, count_updw}), .seg(HEX2_mux));
     data7seg d1 (.data(hour_car[3:0]),       .seg(HEX1_mux));
     data7seg d0 (.data(remaining),           .seg(HEX0_mux));

     mux2_1 m14 (.s(sel_HEX34), .d1(HEX4_mux), .d0(7'h3f), .out(out14));
     mux2_1 m13 (.s(sel_HEX34), .d1(HEX3_mux), .d0(7'h3f), .out(out13));

     mux2_1 m24 (.s(~show), .d1(7'h7f),     .d0(out14),     .out(out24));
     mux2_1 m23 (.s(~show), .d1(7'h7f),     .d0(out13),     .out(out23));
     mux2_1 m22 (.s(~show), .d1(7'h7f),     .d0(HEX2_mux), .out(out22));
     mux2_1 m21 (.s(~show), .d1(7'h7f),     .d0(HEX1_mux), .out(out21));
     mux2_1 m20 (.s(~show), .d1(HEX0_mux), .d0(7'h7f),     .out(out20));

     // define select signal for HEX0~HEX5.
     assign select = full && ~show;

     mux2_1 m35 (.s(~show), .d1(HEX5_mux), .d0(7'h7f), .out(HEX5));
     mux2_1 m34 (.s(select), .d1(7'h7f),     .d0(out24), .out(HEX4));
     mux2_1 m33 (.s(select), .d1(7'h0e),     .d0(out23), .out(HEX3));
     mux2_1 m32 (.s(select), .d1(7'h41),     .d0(out22), .out(HEX2));
     mux2_1 m31 (.s(select), .d1(7'h47),     .d0(out21), .out(HEX1));
     mux2_1 m30 (.s(select), .d1(7'h47),     .d0(out20), .out(HEX0));


       // Generate clk off of CLOCK_50, whichClock picks rate.
    logic [31:0] div_clk;

    parameter whichClock = 1; // (50 MHz / 2^24 = 2.98 Hz)
    /*=====================================================================*/
    //     clock_divider (File Name: clock_divider.sv)
    /*=====================================================================*/
     clock_divider cdiv (.clock(clock), .reset(reset), .divided_clocks(div_clk));
    /*---------------------------------------------------------------------*/
    //     Clock selection;
    //     allows for easy switching between simulation and board clocks
    /*---------------------------------------------------------------------*/
    logic clkSelect;

     //Uncomment ONE of the following two lines depending on intention
    //*********************************************************************
     //assign clkSelect = CLOCK_50;           // for simulation
     assign clkSelect = div_clk[whichClock]; // for DE1_SoC  board
    //*********************************************************************

     // generate the read addresses
     count_updw c1 (.clk(clkSelect), .reset(~show), .count_updw(count_updw));

endmodule
/*=======================================================================*/
//   mux2_1
/*=======================================================================*/
module mux2_1 (
    input logic s,
    input logic [6:0] d1,
    input logic [6:0] d0,
    output logic [6:0] out);

    assign out = s ? d1 : d0;

endmodule
/*=======================================================================*/
//   Testbench for parkinglot_datapath
```

```
157  /*=============================================================================*/
158  `timescale 1 ps / 1 ps
159  module parkinglot_datapath_testbench();
160      logic clock;
161      logic reset;
162      logic clear_reg;
163      logic hour_inc;
164      logic wr;
165      logic RH_start;
166      logic RH_end;
167      logic RH_flag;
168      logic show;
169      logic [1:0] count;
170      logic full;
171      logic empty;
172      logic hour_7;
173      logic [2:0] hour;
174      logic [2:0] count_updw;
175      logic [15:0] hour_car;
176      logic [6:0] HEX5;
177      logic [6:0] HEX4;
178      logic [6:0] HEX3;
179      logic [6:0] HEX2;
180      logic [6:0] HEX1;
181      logic [6:0] HEX0;
182
183      integer i;
184
185      parkinglot_datapath dut (.*);
186
187      //clock generator
188
189      parameter CLK_Period = 100;
190
191      initial begin
192          clock <= 1'b0;
193         forever #(CLK_Period/2) clock <= ~clock;
194      end
195
196      initial begin
197          reset <= 1;
198
199          repeat(1)
200          @(posedge clock);
201          reset <= 0;
202
203          repeat(1)
204          @(posedge clock);
205
206          // Use for loop to generate test patterns.
207
208          for (i = 0; i < 2**6; i++) begin
209              {show, hour_inc, wr, RH_start, RH_end, RH_flag} = i;
210              repeat(2)
211              @(posedge clock);
212          end
213
214          repeat(2)
215          @(posedge clock);
216
217         $stop;
218      end
219
220  endmodule
```

```
1   // megafunction wizard: %RAM: 2-PORT%
2   // GENERATION: STANDARD
3   // VERSION: WM1.0
4   // MODULE: altsyncram
5
6   // ============================================================
7   // File Name: ram8x16.v
8   // Megafunction Name(s):
9   //          altsyncram
10  //
11  // Simulation Library Files(s):
12  //          altera_mf
13  // ============================================================
14  // ************************************************************
15  // THIS IS A WIZARD-GENERATED FILE. DO NOT EDIT THIS FILE!
16  //
17  // 17.0.0 Build 595 04/25/2017 SJ Lite Edition
18  // ************************************************************
19
20
21  //Copyright (C) 2017  Intel Corporation. All rights reserved.
22  //Your use of Intel Corporation's design tools, logic functions
23  //and other software and tools, and its AMPP partner logic
24  //functions, and any output files from any of the foregoing
25  //(including device programming or simulation files), and any
26  //associated documentation or information are expressly subject
27  //to the terms and conditions of the Intel Program License
28  //Subscription Agreement, the Intel Quartus Prime License Agreement,
29  //the Intel MegaCore Function License Agreement, or other
30  //applicable license agreement, including, without limitation,
31  //that your use is for the sole purpose of programming logic
32  //devices manufactured by Intel and sold by Intel or its
33  //authorized distributors.  Please refer to the applicable
34  //agreement for further details.
35
36
37  // synopsys translate_off
38  `timescale 1 ps / 1 ps
39  // synopsys translate_on
40  module ram8x16 (
41     clock,
42     data,
43     rdaddress,
44     wraddress,
45     wren,
46     q);
47
48     input   clock;
49     input [15:0]  data;
50     input [2:0]  rdaddress;
51     input [2:0]  wraddress;
52     input   wren;
53     output   [15:0]  q;
54  `ifndef ALTERA_RESERVED_QIS
55  // synopsys translate_off
56  `endif
57     tri1    clock;
58     tri0    wren;
59  `ifndef ALTERA_RESERVED_QIS
60  // synopsys translate_on
61  `endif
62
63     wire [15:0] sub_wire0;
64     wire [15:0] q = sub_wire0[15:0];
65
66     altsyncram  altsyncram_component (
67               .address_a (wraddress),
68               .address_b (rdaddress),
69               .clock0 (clock),
70               .data_a (data),
71               .wren_a (wren),
72               .q_b (sub_wire0),
73               .aclr0 (1'b0),
74               .aclr1 (1'b0),
75               .addressstall_a (1'b0),
76               .addressstall_b (1'b0),
77               .byteena_a (1'b1),
78               .byteena_b (1'b1),
```

```
79                .clock1 (1'b1),
80                .clocken0 (1'b1),
81                .clocken1 (1'b1),
82                .clocken2 (1'b1),
83                .clocken3 (1'b1),
84                .data_b ({16{1'b1}}),
85                .eccstatus (),
86                .q_a (),
87                .rden_a (1'b1),
88                .rden_b (1'b1),
89                .wren_b (1'b0));
90        defparam
91           altsyncram_component.address_aclr_b = "NONE",
92           altsyncram_component.address_reg_b = "CLOCK0",
93           altsyncram_component.clock_enable_input_a = "BYPASS",
94           altsyncram_component.clock_enable_input_b = "BYPASS",
95           altsyncram_component.clock_enable_output_b = "BYPASS",
96           altsyncram_component.intended_device_family = "Cyclone V",
97           altsyncram_component.lpm_type = "altsyncram",
98           altsyncram_component.numwords_a = 8,
99           altsyncram_component.numwords_b = 8,
100          altsyncram_component.operation_mode = "DUAL_PORT",
101          altsyncram_component.outdata_aclr_b = "NONE",
102          altsyncram_component.outdata_reg_b = "UNREGISTERED",
103          altsyncram_component.power_up_uninitialized = "FALSE",
104          altsyncram_component.ram_block_type = "M10K",
105          altsyncram_component.read_during_write_mode_mixed_ports = "DONT_CARE",
106          altsyncram_component.widthad_a = 3,
107          altsyncram_component.widthad_b = 3,
108          altsyncram_component.width_a = 16,
109          altsyncram_component.width_b = 16,
110          altsyncram_component.width_byteena_a = 1;
111
112
113   endmodule
114
115   // ============================================================
116   // CNX file retrieval info
117   // ============================================================
118   // Retrieval info: PRIVATE: ADDRESSSTALL_A NUMERIC "0"
119   // Retrieval info: PRIVATE: ADDRESSSTALL_B NUMERIC "0"
120   // Retrieval info: PRIVATE: BYTEENA_ACLR_A NUMERIC "0"
121   // Retrieval info: PRIVATE: BYTEENA_ACLR_B NUMERIC "0"
122   // Retrieval info: PRIVATE: BYTE_ENABLE_A NUMERIC "0"
123   // Retrieval info: PRIVATE: BYTE_ENABLE_B NUMERIC "0"
124   // Retrieval info: PRIVATE: BYTE_SIZE NUMERIC "8"
125   // Retrieval info: PRIVATE: BlankMemory NUMERIC "1"
126   // Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_A NUMERIC "0"
127   // Retrieval info: PRIVATE: CLOCK_ENABLE_INPUT_B NUMERIC "0"
128   // Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_A NUMERIC "0"
129   // Retrieval info: PRIVATE: CLOCK_ENABLE_OUTPUT_B NUMERIC "0"
130   // Retrieval info: PRIVATE: CLRdata NUMERIC "0"
131   // Retrieval info: PRIVATE: CLRq NUMERIC "0"
132   // Retrieval info: PRIVATE: CLRrdaddress NUMERIC "0"
133   // Retrieval info: PRIVATE: CLRrren NUMERIC "0"
134   // Retrieval info: PRIVATE: CLRwraddress NUMERIC "0"
135   // Retrieval info: PRIVATE: CLRwren NUMERIC "0"
136   // Retrieval info: PRIVATE: Clock NUMERIC "0"
137   // Retrieval info: PRIVATE: Clock_A NUMERIC "0"
138   // Retrieval info: PRIVATE: Clock_B NUMERIC "0"
139   // Retrieval info: PRIVATE: IMPLEMENT_IN_LES NUMERIC "0"
140   // Retrieval info: PRIVATE: INDATA_ACLR_B NUMERIC "0"
141   // Retrieval info: PRIVATE: INDATA_REG_B NUMERIC "0"
142   // Retrieval info: PRIVATE: INIT_FILE_LAYOUT STRING "PORT_B"
143   // Retrieval info: PRIVATE: INIT_TO_SIM_X NUMERIC "0"
144   // Retrieval info: PRIVATE: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
145   // Retrieval info: PRIVATE: JTAG_ENABLED NUMERIC "0"
146   // Retrieval info: PRIVATE: JTAG_ID STRING "NONE"
147   // Retrieval info: PRIVATE: MAXIMUM_DEPTH NUMERIC "0"
148   // Retrieval info: PRIVATE: MEMSIZE NUMERIC "128"
149   // Retrieval info: PRIVATE: MEM_IN_BITS NUMERIC "0"
150   // Retrieval info: PRIVATE: MIFfilename STRING ""
151   // Retrieval info: PRIVATE: OPERATION_MODE NUMERIC "2"
152   // Retrieval info: PRIVATE: OUTDATA_ACLR_B NUMERIC "0"
153   // Retrieval info: PRIVATE: OUTDATA_REG_B NUMERIC "0"
154   // Retrieval info: PRIVATE: RAM_BLOCK_TYPE NUMERIC "2"
155   // Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_MIXED_PORTS NUMERIC "2"
156   // Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_PORT_A NUMERIC "3"
```

```
157   // Retrieval info: PRIVATE: READ_DURING_WRITE_MODE_PORT_B NUMERIC "3"
158   // Retrieval info: PRIVATE: REGdata NUMERIC "1"
159   // Retrieval info: PRIVATE: REGq NUMERIC "1"
160   // Retrieval info: PRIVATE: REGrdaddress NUMERIC "1"
161   // Retrieval info: PRIVATE: REGrren NUMERIC "1"
162   // Retrieval info: PRIVATE: REGwraddress NUMERIC "1"
163   // Retrieval info: PRIVATE: REGwren NUMERIC "1"
164   // Retrieval info: PRIVATE: SYNTH_WRAPPER_GEN_POSTFIX STRING "0"
165   // Retrieval info: PRIVATE: USE_DIFF_CLKEN NUMERIC "0"
166   // Retrieval info: PRIVATE: UseDPRAM NUMERIC "1"
167   // Retrieval info: PRIVATE: VarWidth NUMERIC "0"
168   // Retrieval info: PRIVATE: WIDTH_READ_A NUMERIC "16"
169   // Retrieval info: PRIVATE: WIDTH_READ_B NUMERIC "16"
170   // Retrieval info: PRIVATE: WIDTH_WRITE_A NUMERIC "16"
171   // Retrieval info: PRIVATE: WIDTH_WRITE_B NUMERIC "16"
172   // Retrieval info: PRIVATE: WRADDR_ACLR_B NUMERIC "0"
173   // Retrieval info: PRIVATE: WRADDR_REG_B NUMERIC "0"
174   // Retrieval info: PRIVATE: WRCTRL_ACLR_B NUMERIC "0"
175   // Retrieval info: PRIVATE: enable NUMERIC "0"
176   // Retrieval info: PRIVATE: rden NUMERIC "0"
177   // Retrieval info: LIBRARY: altera_mf altera_mf.altera_mf_components.all
178   // Retrieval info: CONSTANT: ADDRESS_ACLR_B STRING "NONE"
179   // Retrieval info: CONSTANT: ADDRESS_REG_B STRING "CLOCK0"
180   // Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_A STRING "BYPASS"
181   // Retrieval info: CONSTANT: CLOCK_ENABLE_INPUT_B STRING "BYPASS"
182   // Retrieval info: CONSTANT: CLOCK_ENABLE_OUTPUT_B STRING "BYPASS"
183   // Retrieval info: CONSTANT: INTENDED_DEVICE_FAMILY STRING "Cyclone V"
184   // Retrieval info: CONSTANT: LPM_TYPE STRING "altsyncram"
185   // Retrieval info: CONSTANT: NUMWORDS_A NUMERIC "8"
186   // Retrieval info: CONSTANT: NUMWORDS_B NUMERIC "8"
187   // Retrieval info: CONSTANT: OPERATION_MODE STRING "DUAL_PORT"
188   // Retrieval info: CONSTANT: OUTDATA_ACLR_B STRING "NONE"
189   // Retrieval info: CONSTANT: OUTDATA_REG_B STRING "UNREGISTERED"
190   // Retrieval info: CONSTANT: POWER_UP_UNINITIALIZED STRING "FALSE"
191   // Retrieval info: CONSTANT: RAM_BLOCK_TYPE STRING "M10K"
192   // Retrieval info: CONSTANT: READ_DURING_WRITE_MODE_MIXED_PORTS STRING "DONT_CARE"
193   // Retrieval info: CONSTANT: WIDTHAD_A NUMERIC "3"
194   // Retrieval info: CONSTANT: WIDTHAD_B NUMERIC "3"
195   // Retrieval info: CONSTANT: WIDTH_A NUMERIC "16"
196   // Retrieval info: CONSTANT: WIDTH_B NUMERIC "16"
197   // Retrieval info: CONSTANT: WIDTH_BYTEENA_A NUMERIC "1"
198   // Retrieval info: USED_PORT: clock 0 0 0 0 INPUT VCC "clock"
199   // Retrieval info: USED_PORT: data 0 0 16 0 INPUT NODEFVAL "data[15..0]"
200   // Retrieval info: USED_PORT: q 0 0 16 0 OUTPUT NODEFVAL "q[15..0]"
201   // Retrieval info: USED_PORT: rdaddress 0 0 3 0 INPUT NODEFVAL "rdaddress[2..0]"
202   // Retrieval info: USED_PORT: wraddress 0 0 3 0 INPUT NODEFVAL "wraddress[2..0]"
203   // Retrieval info: USED_PORT: wren 0 0 0 0 INPUT GND "wren"
204   // Retrieval info: CONNECT: @address_a 0 0 3 0 wraddress 0 0 3 0
205   // Retrieval info: CONNECT: @address_b 0 0 3 0 rdaddress 0 0 3 0
206   // Retrieval info: CONNECT: @clock0 0 0 0 0 clock 0 0 0 0
207   // Retrieval info: CONNECT: @data_a 0 0 16 0 data 0 0 16 0
208   // Retrieval info: CONNECT: @wren_a 0 0 0 0 wren 0 0 0 0
209   // Retrieval info: CONNECT: q 0 0 16 0 @q_b 0 0 16 0
210   // Retrieval info: GEN_FILE: TYPE_NORMAL ram8x16.v TRUE
211   // Retrieval info: GEN_FILE: TYPE_NORMAL ram8x16.inc FALSE
212   // Retrieval info: GEN_FILE: TYPE_NORMAL ram8x16.cmp FALSE
213   // Retrieval info: GEN_FILE: TYPE_NORMAL ram8x16.bsf FALSE
214   // Retrieval info: GEN_FILE: TYPE_NORMAL ram8x16_inst.v FALSE
215   // Retrieval info: GEN_FILE: TYPE_NORMAL ram8x16_bb.v TRUE
216   // Retrieval info: LIB_FILE: altera_mf
217
```

```
/*=====================================================================*/
//  Name: Brian Chen
//  Date: 03-15-2025
//  EE/CSE371 LAB6--- System Verilog in the Real World (Task 2)
//  Device under Test (dut) --- count_updw
//  File Name: count_updw.sv
/*=====================================================================*/
module count_updw(clk, reset, count_updw);
    input logic clk, reset;
    output logic [2:0] count_updw;
// State variables
    typedef enum logic [3:0] {s0, s1u, s2u, s3u, s4u, s5u, s6u, s7,
                              s1d, s2d, s3d, s4d, s5d, s6d} state_t;
    state_t ps, ns;

/*    Combinational_Logic_Here   */
// Next State logic
// Use an FSM to design this counter.
    always_comb begin
        ns = s0; // Default to avoid latches
        case (ps)
            s0:  begin ns = s1u; count_updw = 3'b000; end
            s1u: begin ns = s2u; count_updw = 3'b001; end
            s2u: begin ns = s3u; count_updw = 3'b010; end
            s3u: begin ns = s4u; count_updw = 3'b011; end
            s4u: begin ns = s5u; count_updw = 3'b100; end
            s5u: begin ns = s6u; count_updw = 3'b101; end
            s6u: begin ns = s7;  count_updw = 3'b110; end
            s7:  begin ns = s6d; count_updw = 3'b111; end
            s6d: begin ns = s5d; count_updw = 3'b110; end
            s5d: begin ns = s4d; count_updw = 3'b101; end
            s4d: begin ns = s3d; count_updw = 3'b100; end
            s3d: begin ns = s2d; count_updw = 3'b011; end
            s2d: begin ns = s1d; count_updw = 3'b010; end
            s1d: begin ns = s0;  count_updw = 3'b001; end
         default: ns = s0;
      endcase
    end

/*    State Transition at posedge clk  */
    always_ff @(posedge clk or posedge reset) begin
        if (reset)
            ps <= s0;
        else
            ps <= ns;
    end

endmodule
/*=====================================================================*/
// Testbench
/*=====================================================================*/
module count_updw_testbench();

    logic clk, reset;
    logic [2:0] count_updw;

    count_updw dut (.clk, .reset, .count_updw);

    // clock generator
    parameter CLOCK_PERIOD = 20; // default timescale 1ns/1ns
    initial begin
        clk <= 0;
        forever #(CLOCK_PERIOD/2) clk <= ~clk;
    end

    initial begin
        reset = 1;
        repeat(1)
        @(posedge clk);

        reset = 0;
        repeat(30)
        @(posedge clk);
        $stop;
    end
endmodule
```

```
1   /*=================================================================*/
2   //  Name: Brian Chen
3   //  Date: 03-15-2025
4   //  EE/CSE371 LAB6--- System Verilog in the Real World (Task 2)
5   //  Device under Test (dut) --- counter
6   //  File Name: counter.sv
7   /*=================================================================*/
8   module counter(clk, reset, inc, dec, count);
9
10      input logic clk, reset, inc ,dec;
11      output logic [1:0] count;
12
13     //counter logic
14
15      always_ff @(posedge clk or posedge reset) begin
16          if (reset)
17              count <= 2'b00; // reset count to 0
18          else begin
19              if (inc && !dec && count < 2'b11)
20                  count <= count + 1; // Increment if below max
21              else if (dec && !inc && count > 2'b00)
22                  count <= count - 1 ; // Decrement iff above 0
23          end
24     end
25  endmodule
26  /*=================================================================*/
27  // Testbench
28  /*=================================================================*/
29  module counter_testbench();
30
31      logic clk, reset, inc, dec;
32      logic [1:0] count;
33
34      counter dut (.clk, .reset, .inc, .dec, .count);
35
36      parameter CLOCK_PERIOD = 20; // default timescale 1ns/1ns
37      initial begin
38          clk <= 0;
39          forever #(CLOCK_PERIOD/2) clk <= ~clk;
40      end
41
42      initial begin
43          reset = 1;
44          repeat(2)
45          @(posedge clk);
46          reset = 0;
47          repeat(8)
48          @(posedge clk);
49          {inc, dec} = 2'b10;
50          repeat(8)
51          @(posedge clk);
52          {inc, dec} = 2'b01;
53          repeat(8)
54          @(posedge clk);
55          $stop;
56      end
57  endmodule
```

```
/*=====================================================================*/
//  Name: Brian Chen
//  Date: 03-15-2025
//  EE/CSE371 LAB6--- System Verilog in the Real World (Task 2)
//  Device under Test (dut) --- parkinglot_ctrl
//  File Name: parkinglot_ctrl.sv
/*=====================================================================*/
module parkinglot_ctrl (
    input logic clock,
    input logic reset,
    input logic start,
    input logic KEY0,
    input logic full,       // indicate the status of parking lot
    input logic empty,      // indicate the status of parking lot
    input logic hour_7,     // indicate hour == 7.
    output logic hour_inc,
    output logic wr,
    output logic RH_start,  // indicate the 1st rush hour start.
    output logic RH_end,    // indicate the 1st rush hour end.
    output logic RH_flag,
    output logic show,      // show RAM contents and rush hour start and end
    output logic clear_reg);

// State variables
    typedef enum logic [2:0] {s_idle, s_go, s_rush_start, s_rush_end,
                              s_wait1, s_wait2,
                              s_done, s_display} state_t;
    state_t ps, ns;

/*    Combinational_Logic_Here   */
// Next State logic
    always_comb begin
        ns = s_idle; // Default to avoid latches
        case (ps)
            s_idle:
                ns = start ? s_go : s_idle;

            s_go:
                if (full)
                    ns = s_rush_start;
                else
                    if (hour_7)
                        ns = s_done;
                    else
                        ns = s_go;

            s_rush_start:    // record the start of the first rush hour.
                ns = s_wait1;

            s_wait1:
                if (empty)
                    ns = s_rush_end;
                else
                    if (hour_7)
                        ns = s_done;
                    else
                        ns = s_wait1;

            s_rush_end:    // record the end of the first rush hour.
                ns = s_wait2;

            s_wait2:
                ns = hour_7 ? s_done : s_wait2;

            s_done:
                ns = KEY0 ? s_done : s_display;

            s_display:   // display contents of RAM.
                ns = start ? s_display : s_idle;

            default: ns = s_idle;
        endcase
     end

/*    State Transition at posedge clk */
     always_ff @(posedge clock) begin
         if (reset)
             ps <= s_idle;
```

```
79              else
80                  ps <= ns;
81          end
82
83      /* Outputs   */
84          assign hour_inc = !KEY0 && ((ps != s_display) && (ps != s_idle));
85          assign wr       = !KEY0 && ((ps != s_display) && (ps != s_idle));
86          assign RH_start = (ps ==s_rush_start);
87          assign RH_end = (ps == s_rush_end);
88          assign RH_flag = (ps == s_rush_start);
89          assign show = (ps == s_display);
90          assign clear_reg = (ps == s_idle);
91
92      endmodule
93      /*=====================================================================*/
94      //   Testbench
95      /*=====================================================================*/
96      `timescale 1 ps / 1 ps
97      module parkinglot_ctrl_testbench();
98          logic clock;
99          logic reset;
100         logic start;
101         logic KEY0;
102         logic full;
103         logic empty;
104         logic hour_7;
105         logic hour_inc;
106         logic wr;
107         logic RH_start;
108         logic RH_end;
109         logic RH_flag;
110         logic show;
111         logic clear_reg;
112
113         integer i;
114
115         parkinglot_ctrl dut (.*);
116
117         // clock generator
118         parameter CLK_Period = 100;
119
120         initial begin
121             clock <= 1'b0;
122             forever #(CLK_Period/2) clock <= ~clock;
123         end
124
125         initial begin
126             reset <= 1;
127
128             repeat(1)
129             @(posedge clock);
130             reset <= 0;
131
132             repeat(1)
133             @(posedge clock);
134             start <= 0;
135
136             repeat(1)
137             @(posedge clock);
138             start <= 1;
139
140             // use for loop to generate test patterns.
141             for (i = 0; i < 2**4; i++) begin
142                 {hour_7, empty, full, KEY0} = i;
143                 repeat(2)
144                 @(posedge clock);
145             end
146
147             repeat(2)
148             @(posedge clock);
149
150             start <= 0;
151             repeat(2)
152             @(posedge clock);
153
154             $stop;
155         end
156
```

```
157     endmodule
```

```
1   /*=================================================================*/
2   //  Name: Brian Chen
3   //  Date: 03-15-2025
4   //  EE/CSE371 LAB6--- System Verilog in the Real World (Task 2)
5   //  Device under Test (dut) --- twoDFF
6   //  File Name: twoDFF.sv
7   /*=================================================================*/
8   module twoDFF (CLK, RST, D, Q);
9   input logic CLK, RST;
10  input logic D;
11  output logic Q;
12  logic Q1;
13
14      // DFFs
15      always_ff @(posedge CLK or posedge RST) begin
16          if (RST) begin
17            Q  <= 0;
18           Q1 <=0;
19        end
20          else begin
21            Q  <= Q1;
22           Q1 <= D;
23         end
24      end
25
26  endmodule
27  /*=================================================================*/
28  // Testbench
29  /*=================================================================*/
30  module twoDFF_testbench();
31
32      logic CLK, RST, D, Q;
33
34      twoDFF dut (.CLK, .RST, .D, .Q);
35
36      parameter CLOCK_PERIOD = 20; // default timescale 1ns/1ns
37
38      initial begin
39          CLK <= 0;
40          forever #(CLOCK_PERIOD/2) CLK <= ~CLK;
41      end
42
43      initial begin
44          RST = 1;
45          repeat(2)
46          @(posedge CLK);
47          RST = 0;
48
49          repeat(2)
50          @(posedge CLK);
51          D <= 0;
52
53          repeat(2)
54          @(posedge CLK);
55          D <= 1;
56
57          repeat(2)
58          @(posedge CLK);
59          D <= 0;
60
61          repeat(2)
62          @(posedge CLK);
63          $stop;
64       end
65  endmodule
```

```
/*=====================================================================*/
//  Name: Brian Chen
//  Date: 03-15-2025
//  EE/CSE371 LAB6--- System Verilog in the Real World (Task 2)
//  Device under Test (dut) --- userInput
//  File Name: userInput.sv
/*=====================================================================*/
module userInput (CLK, RST, Qin, Qeff);
    input logic CLK, RST;
    input logic Qin;
    output logic Qeff;

// State variables
    typedef enum logic [1:0] {
        start, s1, s2
    } state_t;
    state_t ps, ns;

    // Next State logic
    always_comb begin
        ns = start; // Default to avoid latches
        case (ps)
            start:
                if (Qin == 1) ns = s1;
                else ns = start;

            s1:
                if (Qin == 1) ns = s2;
                else ns = start;

            s2:
                if (Qin == 1) ns = s2;
                else ns = start;
            default: ns = start;
    endcase
end

    // Output logic
    always_comb begin
        Qeff = 0;  // Default to avoid latches
        case (ps)
            start: Qeff = 0;
            s1:    Qeff = 1;
            s2:    Qeff = 0;
            default: Qeff = 0;
        endcase
    end

    // DFFs
    always_ff @(posedge CLK or posedge RST) begin
        if (RST)
            ps <= start;
        else
            ps <= ns;
    end

endmodule

/*=====================================================================*/
// Testbench
/*=====================================================================*/
module userInput_testbench();

    logic CLK, RST, Qin, Qeff;

    userInput dut (.CLK, .RST, .Qin, .Qeff);

    parameter CLOCK_PERIOD = 20; // default timescale 1ns/1ns

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

    initial begin
        RST = 1;
        repeat(2)
        @(posedge CLK);
```

```
79            RST = 0;
80
81            repeat(2)
82            @(posedge CLK);
83            Qin <= 0;
84
85            repeat(3)
86            @(posedge CLK);
87            Qin <= 1;
88
89            repeat(3)
90            @(posedge CLK);
91            Qin <= 0;
92
93            repeat(3)
94            @(posedge CLK);
95            $stop;
96        end
97    endmodule
98
```

```
1   /*=================================================================*/
2   //  Name: Brian Chen
3   //  Date: 03-15-2025
4   //  EE/CSE371 LAB6--- System Verilog in the Real World (Task 2)
5   //  Device under Test (dut) --- data7seg
6   //  File Name: data7seg.sv
7   /*=================================================================*/
8   module data7seg(data, seg);
9
10      input logic [3:0] data;
11      output logic [6:0] seg;
12
13      parameter [6:0] Blank = 7'b111_1111;  // 7'h7f
14      parameter [6:0] Chr_0 = 7'b100_0000;  // 7'h40
15      parameter [6:0] Chr_1 = 7'b111_1001;  // 7'h79
16      parameter [6:0] Chr_2 = 7'b010_0100;  // 7'h24
17      parameter [6:0] Chr_3 = 7'b011_0000;  // 7'h30
18      parameter [6:0] Chr_4 = 7'b001_1001;  // 7'h19
19      parameter [6:0] Chr_5 = 7'b001_0010;  // 7'h12
20      parameter [6:0] Chr_6 = 7'b000_0010;  // 7'h02
21      parameter [6:0] Chr_7 = 7'b101_1000;  // 7'h58
22      parameter [6:0] Chr_8 = 7'b000_0000;  // 7'h00
23      parameter [6:0] Chr_9 = 7'b001_0000;  // 7'h10
24      parameter [6:0] Chr_A = 7'b000_1000;  // 7'h08
25      parameter [6:0] Chr_b = 7'b000_0011;  // 7'h03
26      parameter [6:0] Chr_C = 7'b100_0110;  // 7'h46
27      parameter [6:0] Chr_d = 7'b010_0001;  // 7'h21
28      parameter [6:0] Chr_E = 7'b000_0110;  // 7'h06
29      parameter [6:0] Chr_F = 7'b000_1110;  // 7'h0e
30
31      always_comb begin
32          case(data)
33              0:  seg = Chr_0;
34              1:  seg = Chr_1;
35              2:  seg = Chr_2;
36              3:  seg = Chr_3;
37
38              4:  seg = Chr_4;
39              5:  seg = Chr_5;
40              6:  seg = Chr_6;
41              7:  seg = Chr_7;
42
43              8:  seg = Chr_8;
44              9:  seg = Chr_9;
45             10:  seg = Chr_A;
46             11:  seg = Chr_b;
47
48             12:  seg = Chr_C;
49             13:  seg = Chr_d;
50             14:  seg = Chr_E;
51             15:  seg = Chr_F;
52             default: seg = Blank;
53         endcase
54      end
55  endmodule
```

```
1   /*=================================================================*/
2   //  Name: Brian Chen
3   //  Date: 03-15-2025
4   //  EE/CSE371 LAB6--- System Verilog in the Real World (Task 2)
5   //  Device under Test (dut) --- clock_divider
6   //  File Name: clock_divider.sv
7   /*=================================================================*/
8   // divided_clocks[0] = 25MHz, [1] = 12.5Mhz, ... [23] = 3Hz, [24] = 1.5Hz, [25] = 0.75Hz,
    ...
9   module clock_divider (clock, reset, divided_clocks);
10
11  input logic reset, clock;
12  output logic [31:0] divided_clocks = 0;
13
14  always_ff @(posedge clock) begin
15      divided_clocks <= divided_clocks + 1;
16  end
17
18  endmodule
19
```

```
1   // Takes in a short pulse signal, extends the length of the pulse
2   // by 50,000,000 clock cycles for use with 3D simulator gates.
3
4   //in: 1-bit original input pulse
5   //out: 1-bit extended output pulse
6   //reset: 1-bit reset signal that sets counter to 0
7   //clk: 1-bit input clock controlling system
8   module signal_extender (in, out, reset, clk);
9
10     input logic in, reset, clk;
11     output logic out;
12     logic [31:0] count = 32'b0;
13     assign out = (count > 0);
14
15     always_ff @(posedge clk) begin
16         if (reset) begin
17             count <= 0;
18         end
19         else begin
20            if (in) count <= 50000000;
21            else if (count > 0) count <= count - 1;
22         end
23     end
24  endmodule
25
```