VIETNAM NATIONAL UNIVERSITY - HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
Faculty of Computer Science and Engineering

LAB 1
INTRODUCTION TO SYSTEM ON CHIP
LABORATORY REPORT

Instructor: Pham Kieu Nhat Anh

Class: CC01
Name: Hoang Thuy Tram
Student ID: 2353202

Ho Chi Minh city, December 2025

Contents

1 Github

The Lab 1 project files (RTL source code, testbenches, simulation scripts) are located in the `Lab01` folder. They are available at the following repository: https://github.com/trxmhoang/SoC.git

2 Exercise 1

2.1 RTL implementation

The `debounce` module is designed to filter out mechanical noise, is also known as bouncing, that occurs when a physical button is pressed. It ensures that the FPGA only registers a single, clean signal instead of multiple rapid spikes. It outputs two signals, a stable level through signal `btn_stable` and a single-cycle pulse through signal `btn_pulse`, triggered when the button is pressed.

```
module debounce (
    input wire clk,
    input wire rst_n,
    input wire btn_in,
    output reg btn_stable,
    output reg btn_pulse
);

`ifdef SIMULATION
    //for sim, debounce is 10 cycles
    parameter STABLE_CNT = 21'd10;
`else
    //for fpga, use the real delay
    parameter STABLE_CNT = 21'd1_250_000;
`endif

reg [20:0] cnt;
reg btn1;
reg btn2;
reg btn_stable_pre;

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        btn1 <= 1;
        btn2 <= 1;
        btn_stable <= 1;
        btn_stable_pre <= 1;
        btn_pulse <= 0;
        cnt <= 0;
    end else begin
        btn_stable_pre <= btn_stable;
        btn1 <= btn_in;
        btn2 <= btn1;

        if (btn2 != btn_stable)
            if (cnt < STABLE_CNT) cnt <= cnt + 1;
        else
```

```
            cnt <= 0;

        if (cnt == STABLE_CNT)
            btn_stable <= btn2;

        if (btn_stable == 0 && btn_stable_pre == 1)
            btn_pulse <= 1;
        else
            btn_pulse <= 0;
    end
end
endmodule
```

Program 1: RTL implementation for the debounce module of Exercise 1.

The `ex1` module implements the traffic light control system that can be configured. It uses a Finite State Machine to cycle thorugh the standard traffic light sequence: Red, green and yellow for two directions - horizontal and vertical.
The system operates in three modes, selectable via Button 0 through signal `button[0]`:

- Mode 1 (normal operation): The traffic lights cycle automatically. The duration for green and yellow lights is determined by `green_time` and `yellow_time` registers. A 1-second pulse (`pulse_1s`), generated by a clock divider, decrements the timer trigger state transitions.
- Mode 2 (config green lights): The user can adjust the green lights duration (`green_time`) using Button 1 (`button[1]`) to increase and Button 2 (`button [2]`) to decrease, Button 3 (`button[3]`) saves the new value when pressed.
- Mode 3 (config yellow lights): Similar to mode 2, but allows adjustment and saving for the yellow lights duration (`yellow_time`).

The first two 7-segment LEDs on the left-hand side display the current mode, while the two 7-segment LEDs on the right-hand side display the timer while green and yellow LEDs running. Two additional LEDs indicate the current state of the vertical and horizontal traffic lights. During the configuration (Mode 2 and Mode 3), these two LEDs have the same color, all green in Mode 2 and all yellow in Mode 3, indicating to the users that they are configuring which LED timer. When the reset switch is activated, the 7-segment LEDs default to Mode 1 and the timer LEDs are turned off, meanwhile the horizontal and vertical traffic lights both become red to indicate the reset is on. The configured values for the green and yellow timers are preserved during reset, retaining the user's settings.

```
module ex1 (
    input wire clk,
    input wire rst_n,
    input wire [3:0] button,
    output reg [2:0] hor_led,
    output reg [2:0] ver_led,
    output reg [3:0] bcd0,
    output reg [3:0] bcd1,
    output reg [3:0] bcd2,
    output reg [3:0] bcd3
);
```

```
12
13 parameter RED = 3'b100;
14 parameter YELLOW = 3'b110;
15 parameter GREEN = 3'b010;
16
17 parameter INIT = 3'd0; //initial state
18 parameter HR_VG = 3'd1; //horizontal red - vertical green
19 parameter HR_VY = 3'd2; //horizontal red - vertical yellow
20 parameter HG_VR = 3'd3; //horizontal green - vertical red
21 parameter HY_VR = 3'd4; //horizontal yellow - vertical red
22
23 parameter MODE1 = 2'd1;
24 parameter MODE2 = 2'd2;
25 parameter MODE3 = 2'd3;
26
27 `ifdef SIMULATION
28     //for sim, a sec is 100 cycles
29     parameter PULSE_COUNT = 27'd100;
30 `else
31     //for fpga, a sec is 125m cycles
32     parameter PULSE_COUNT = 27'd125_000_000;
33 `endif
34
35 //clock divider
36 reg [26:0] clk_cnt;
37 wire pulse_1s;
38
39 always @(posedge clk or negedge rst_n) begin
40     if(!rst_n)
41         clk_cnt <= 27'd0;
42     else if (clk_cnt == PULSE_COUNT - 1)
43         clk_cnt <= 27'd0;
44     else
45         clk_cnt <= clk_cnt + 1;
46 end
47
48 assign pulse_1s = (clk_cnt == PULSE_COUNT - 1);
49
50 //debounce handling
51 wire [3:0] btn_stable;
52 wire [3:0] btn_pulse;
53 wire [3:0] btn_press;
54
55 debounce u_db0 (
56     .clk        (clk),
57     .rst_n      (rst_n),
58     .btn_in     (button[0]),
59     .btn_stable (btn_stable[0]),
60     .btn_pulse  (btn_pulse[0])
61 );
62
63 debounce u_db1 (
64     .clk        (clk),
```

```
    .rst_n      (rst_n),
    .btn_in     (button[1]),
    .btn_stable (btn_stable[1]),
    .btn_pulse  (btn_pulse[1])
);

debounce u_db2 (
    .clk        (clk),
    .rst_n      (rst_n),
    .btn_in     (button[2]),
    .btn_stable (btn_stable[2]),
    .btn_pulse  (btn_pulse[2])
);

debounce u_db3 (
    .clk        (clk),
    .rst_n      (rst_n),
    .btn_in     (button[3]),
    .btn_stable (btn_stable[3]),
    .btn_pulse  (btn_pulse[3])
);

assign btn_press = btn_pulse;

//button handling
reg[2:0] state;
reg[6:0] timer;
reg[1:0] mode;
reg[6:0] green_time = 7'd3, yellow_time = 7'd2;
reg[6:0] temp_green_time, temp_yellow_time;

always @(posedge clk or negedge rst_n) begin
    if(!rst_n) begin
        mode <= MODE1;
        temp_green_time <= green_time;
        temp_yellow_time <= yellow_time;
    end else begin
        case(btn_press)
            4'b0001: begin
                case(mode)
                    MODE1: mode <= MODE2;
                    MODE2: begin
                        mode <= MODE3;
                        temp_green_time <= green_time;
                    end
                    MODE3: begin
                        mode <= MODE1;
                        temp_yellow_time <= yellow_time;
                    end
                    default: begin
                        mode <= MODE1;
                        state <= HR_VG;
                        timer <= green_time;
```

```
                    end
                endcase
            end

            4'b0010: begin
                case(mode)
                    MODE2:
                        if(temp_green_time < 99) temp_green_time <=
                            temp_green_time + 1;
                    MODE3:
                        if(temp_yellow_time < 20) temp_yellow_time <=
                            temp_yellow_time + 1;
                endcase
            end

            4'b0100: begin
                case(mode)
                    MODE2:
                        if(temp_green_time > 2) temp_green_time <=
                            temp_green_time - 1;
                    MODE3:
                        if(temp_yellow_time > 1) temp_yellow_time <=
                            temp_yellow_time - 1;
                endcase
            end

            4'b1000: begin
                case(mode)
                    MODE2:
                        green_time <= temp_green_time;
                    MODE3:
                        yellow_time <= temp_yellow_time;
                endcase
            end
        endcase
    end
end

//led handling
always @(posedge clk or negedge rst_n) begin
    if(!rst_n) begin
        state <= INIT;
        timer <= 0;
    end else if (mode != MODE1) begin
        state <= INIT;
        timer <= 0;
    end else if (pulse_1s) begin
        case(state)
            INIT: begin
                state <= HR_VG;
                timer <= green_time;
            end

```

```
            HR_VG: begin
                if(timer <= 1) begin
                    state <= HR_VY;
                    timer <= yellow_time;
                end else begin
                    timer <= timer - 1;
                end
            end

            HR_VY: begin
                if(timer <= 1) begin
                    state <= HG_VR;
                    timer <= green_time;
                end else begin
                    timer <= timer - 1;
                end
            end

            HG_VR: begin
                if(timer <= 1) begin
                    state <= HY_VR;
                    timer <= yellow_time;
                end else begin
                    timer <= timer - 1;
                end
            end

            HY_VR: begin
                if(timer <= 1) begin
                    state <= HR_VG;
                    timer <= green_time;
                end else begin
                    timer <= timer - 1;
                end
            end

            default: state <= INIT;
        endcase
    end else begin
        state <= state;
        timer <= timer;
    end
end

//led display
always @(*) begin
    case (mode)
        MODE1: begin
            case (state)
                INIT: begin
                    hor_led = RED;
                    ver_led = RED;
                end
```

```
220
221                 HR_VG: begin
222                     hor_led = RED;
223                     ver_led = GREEN;
224                 end
225
226                 HR_VY: begin
227                     hor_led = RED;
228                     ver_led = YELLOW;
229                 end
230
231                 HG_VR: begin
232                     hor_led = GREEN;
233                     ver_led = RED;
234                 end
235
236                 HY_VR: begin
237                     hor_led = YELLOW;
238                     ver_led = RED;
239                 end
240
241                 default: begin
242                     hor_led = RED;
243                     ver_led = RED;
244                 end
245             endcase
246         end
247
248         MODE2: begin
249             hor_led = GREEN;
250             ver_led = GREEN;
251         end
252
253         MODE3: begin
254             hor_led = YELLOW;
255             ver_led = YELLOW;
256         end
257
258         default: begin
259             hor_led = RED;
260             ver_led = RED;
261         end
262     endcase
263 end
264
265 //segment handling
266 reg[7:0] display_val;
267 reg[3:0] ten_digit;
268 reg[3:0] one_digit;
269
270 always @(*) begin
271     case(mode)
272         MODE1: display_val = timer;
```

```
273         MODE2: display_val = temp_green_time;
274         MODE3: display_val = temp_yellow_time;
275         default: display_val = timer;
276     endcase
277
278     ten_digit = display_val / 10;
279     one_digit = display_val % 10;
280
281     bcd0 = one_digit;
282     bcd1 = ten_digit;
283     bcd2 = 4'hF;
284     bcd3 = mode;
285 end
286 endmodule
```

Program 2: RTL implementation of Exercise 1.

2.2 Testbench

The provided testbench is designed to verify the functional correctness of the FSM controls a traffic light system, as implemented in section 2.1. The testbench runs multiple scenarios to ensure the operation of the FSM in various modes and input conditions:

- Test 1 (Auto mode test): Verifies automatic cycling of light states and timers, confirming correct encoding and transitions of the 7-segment display, horizontal (`hor_led`) and vertical (`ver_led`) traffic signals based on predefined time intervals.
- Test 2 (FSM button test): Assesses manual interaction with the system through simulated button presses. This test set checks the increase and decrease of green and yellow light timings and saving/no saving scenarios, ensuring the controller updates the timer values and LED signals as expected.
- Test 3 (Reset test): Checks the system's response to a reset event. This test ensures the FSM correctly returns to its initial state and reinitializes the timer and LED outputs - both during and after the reset release.
- Test 4 (Min, Max boundary test): Verifies correct boundary handling of timer values. The system is subjected to scenarios where green and yellow timers are increased or decreased to their maximum and minimum allowable limits, ensuring no overflow or underflow occurs.

```
1  module tb;
2  reg clk;
3  reg rst_n;
4  reg [3:0] button;
5  wire [2:0] hor_led;
6  wire [2:0] ver_led;
7  wire [3:0] bcd0;
8  wire [3:0] bcd1;
9  wire [3:0] bcd2;
10 wire [3:0] bcd3;
11
12 parameter RED = 3'b100;
```

```
13  parameter YELLOW = 3'b110;
14  parameter GREEN = 3'b010;
15  parameter CLK_PERIOD = 8; // 8ns => 125MHz
16
17  integer pass, fail;
18
19  `ifdef SIMULATION
20      //for sim, debounce is 100 cycles + margin
21      parameter DB = (100 * CLK_PERIOD) + 70; // 870ns
22      //for sim, a sec is 100 clock cycles + margin
23      parameter WAIT_1S  = (100 * CLK_PERIOD) * 1 + 20;
24      parameter WAIT_2S  = (100 * CLK_PERIOD) * 2 + 20;
25      parameter WAIT_3S  = (100 * CLK_PERIOD) * 3 + 20;
26      parameter WAIT_4S  = (100 * CLK_PERIOD) * 4 + 20;
27      parameter WAIT_5S  = (100 * CLK_PERIOD) * 5 + 20;
28      parameter WAIT_6S  = (100 * CLK_PERIOD) * 6 + 20;
29  `else
30      parameter DB = 10_000_000; // 10ms
31      parameter WAIT_1S  = 1_100_000_000;
32      parameter WAIT_2S  = 2_000_000_000;
33      parameter WAIT_3S  = 64'd3_000_000_000;
34      parameter WAIT_4S  = 64'd4_000_000_000;
35      parameter WAIT_5S  = 64'd5_000_000_000;
36      parameter WAIT_6S  = 64'd6_000_000_000;
37  `endif
38
39  ex1 dut (
40      .clk (clk),
41      .rst_n (rst_n),
42      .button (button),
43      .hor_led (hor_led),
44      .ver_led (ver_led),
45      .bcd0 (bcd0),
46      .bcd1 (bcd1),
47      .bcd2 (bcd2),
48      .bcd3 (bcd3)
49  );
50
51  task divi;
52      $display ("
        ----------------------------------------------------------
53      --------------");
54  endtask
55
56  task msg (input [700:0] txt);
57      begin
58          divi();
59          $display ("%0s", txt);
60          divi();
61      end
62  endtask
63
64  task check (input [3:0] exp_bcd3, input [3:0] exp_bcd2, input [3:0]
```

```
    exp_bcd1, input [3:0] exp_bcd0, input [2:0] exp_hor_led, input
    [2:0] exp_ver_led);
65      begin
66          #10;
67          $display ("[EXPECT] Time = %0t | 7SEGs = %0h%0h-%0h%0h,
               HOR_LED = %b, VER_LED = %b", $time, exp_bcd3, exp_bcd2,
               exp_bcd1, exp_bcd0, exp_hor_led, exp_ver_led);
68
69          $display ("[OUTPUT] Time = %0t | 7SEGs = %0h%0h-%0h%0h,
               HOR_LED = %b, VER_LED = %b", $time, bcd3, bcd2, bcd1, bcd0,
                hor_led, ver_led);
70
71          if ((bcd0 == exp_bcd0) && (bcd1 == exp_bcd1) && (bcd2 ==
               exp_bcd2) && (bcd3 == exp_bcd3) && (hor_led == exp_hor_led)
                && (ver_led == exp_ver_led)) begin
72              $display ("======> PASSED");
73              pass = pass + 1;
74          end else begin
75              $display ("======> FAILED");
76              fail = fail + 1;
77          end
78      end
79  endtask
80
81  task incr;
82      begin
83          button = 4'b0010;
84          #(DB);
85          button = 4'b0000;
86          #(DB);
87      end
88  endtask
89
90  task decr;
91      begin
92          button = 4'b0100;
93          #(DB);
94          button = 4'b0000;
95          #(DB);
96      end
97  endtask
98
99  task save;
100     begin
101         button = 4'b1000;
102         #(DB);
103         button = 4'b0000;
104         #(DB);
105     end
106 endtask
107
108 task mode;
109     begin
```

```
110         button = 4'b0001;
111         #(DB);
112         button = 4'b0000;
113         #(DB);
114     end
115 endtask
116 
117 initial begin
118     clk = 0;
119     forever #4 clk = ~clk;
120 end
121 //F = 125MHz clock -> T = 8ns period
122 
123 initial begin
124     $dumpfile ("tb.vcd");
125     $dumpvars (0, tb);
126 
127     rst_n = 1'b0;
128     button = 4'b0000;
129     pass = 0;
130     fail = 0;
131     #100;
132     rst_n = 1'b1;
133     #100;
134 
135     msg ("TEST 1: FSM AUTO MODE TEST");
136     #(WAIT_1S);
137     check (4'd1, 4'd15, 4'd0, 4'd3, RED, GREEN);
138 
139     #(WAIT_3S);
140     check (4'd1, 4'd15, 4'd0, 4'd2, RED, YELLOW);
141 
142     #(WAIT_2S);
143     check (4'd1, 4'd15, 4'd0, 4'd3, GREEN, RED);
144 
145     #(WAIT_3S);
146     check (4'd1, 4'd15, 4'd0, 4'd2, YELLOW, RED);
147 
148     #(WAIT_2S);
149     check (4'd1, 4'd15, 4'd0, 4'd3, RED, GREEN);
150 
151     msg ("TEST 2: FSM BUTTON TEST");
152     msg ("\t1. Increase green 3 times, increase yellow 3 times, then
        save");
153     //green
154     mode();
155     repeat (3) incr();
156     check (4'd2, 4'd15, 4'd0, 3'd6, GREEN, GREEN);
157     save();
158     //yellow
159     mode();
160     repeat (3) incr();
161     check (4'd3, 4'd15, 4'd0, 4'd5, YELLOW, YELLOW);
```

```
162     save();
163     mode();
164
165     #(WAIT_2S);
166     check (4'd1, 4'd15, 4'd0, 4'd4, RED, GREEN);
167     #(WAIT_6S);
168     check (4'd1, 4'd15, 4'd0, 4'd3, RED, YELLOW);
169
170     msg ("\t2. Decrease green 2 times, decrease yellow 1 time, then
            save");
171     //green
172     mode();
173     repeat (2) decr();
174     check (4'd2, 4'd15, 4'd0, 3'd4, GREEN, GREEN);
175     save();
176     //yellow
177     mode();
178     decr();
179     check (4'd3, 4'd15, 4'd0, 4'd4, YELLOW, YELLOW);
180     save();
181     mode();
182
183     #(WAIT_1S);
184     check (4'd1, 4'd15, 4'd0, 4'd3, RED, GREEN);
185     #(WAIT_3S);
186     check (4'd1, 4'd15, 4'd0, 4'd4, RED, YELLOW);
187
188     msg ("\t3. Increase green 1 time, decrease yellow 1 time, then no
            save");
189     //green
190     mode();
191     incr();
192     check (4'd2, 4'd15, 4'd0, 3'd5, GREEN, GREEN);
193     //yellow
194     mode();
195     decr();
196     check (4'd3, 4'd15, 4'd0, 4'd3, YELLOW, YELLOW);
197     mode();
198
199     #(WAIT_2S);
200     check (4'd1, 4'd15, 4'd0, 4'd2, RED, GREEN);
201     #(WAIT_4S);
202     check (4'd1, 4'd15, 4'd0, 4'd2, RED, YELLOW);
203
204     msg ("\t4. Increase green 1 time, decrease yellow 1 time, then
            save");
205     //green
206     mode();
207     incr();
208     check (4'd2, 4'd15, 4'd0, 3'd5, GREEN, GREEN);
209     save();
210     //yellow
211     mode();
```

```
212     decr();
213     check (4'd3, 4'd15, 4'd0, 4'd3, YELLOW, YELLOW);
214     save();
215     mode();
216 
217     #(WAIT_2S);
218     check (4'd1, 4'd15, 4'd0, 4'd3, RED, GREEN);
219     #(WAIT_4S);
220     check (4'd1, 4'd15, 4'd0, 4'd2, RED, YELLOW);
221 
222     msg ("TEST 3: RESET TEST");
223     mode();
224     check (4'd2, 4'd15, 4'd0, 4'd5, GREEN, GREEN);
225     $display ("Resetting...");
226     rst_n = 1'b0;
227     #(WAIT_2S);
228     check (4'd1, 4'd15, 4'd0, 4'd0, RED, RED);
229 
230     mode();
231     check (4'd1, 4'd15, 4'd0, 4'd0, RED, RED);
232 
233     mode();
234     check (4'd1, 4'd15, 4'd0, 4'd0, RED, RED);
235 
236     $display ("Releasing reset...");
237     rst_n = 1'b1;
238     #800;
239     #(WAIT_1S);
240     check (4'd1, 4'd15, 4'd0, 4'd4, RED, GREEN);
241     #(WAIT_5S);
242     check (4'd1, 4'd15, 4'd0, 4'd2, RED, YELLOW);
243 
244     msg ("TEST 4: MIN, MAX BOUNDARY TEST");
245     msg ("\t1. Increase green 94 times, increase yellow 17 times to
        reach max");
246     //green
247     mode();
248     repeat (94) incr();
249     check (4'd2, 4'd15, 4'd9, 4'd9, GREEN, GREEN);
250     save();
251     //yellow
252     mode();
253     repeat (17) incr();
254     check (4'd3, 4'd15, 4'd2, 4'd0, YELLOW, YELLOW);
255     save();
256     mode();
257 
258     msg ("\t 2. Increase green 3 times, increase yellow 5 times");
259     //green
260     mode();
261     repeat (3) incr();
262     check (4'd2, 4'd15, 4'd9, 4'd9, GREEN, GREEN);
263     save();
```

```
264     //yellow
265     mode();
266     repeat (5) incr();
267     check (4'd3, 4'd15, 4'd2, 4'd0, YELLOW, YELLOW);
268     save();
269     mode();
270 
271     msg ("\t 3. Decrease green 97 times, decrease yellow 19 times to
           reach min");
272     //green
273     mode();
274     repeat (97) decr();
275     check (4'd2, 4'd15, 4'd0, 4'd2, GREEN, GREEN);
276     save();
277     //yellow
278     mode();
279     repeat (19) decr();
280     check (4'd3, 4'd15, 4'd0, 4'd1, YELLOW, YELLOW);
281     save();
282     mode();
283 
284     msg ("\t 4. Decrease green 8 times, decrease yellow 3 times");
285     //green
286     mode();
287     repeat (8) decr();
288     check (4'd2, 4'd15, 4'd0, 4'd2, GREEN, GREEN);
289     save();
290     //yellow
291     mode();
292     repeat (3) decr();
293     check (4'd3, 4'd15, 4'd0, 4'd1, YELLOW, YELLOW);
294     save();
295     mode();
296 
297     #100;
298     msg ("SUMMARY");
299     $display ("TOTAL TESTS : %0d", pass + fail);
300     $display ("TOTAL PASSED: %0d", pass);
301     $display ("TOTAL FAILED: %0d", fail);
302     if (fail == 0)
303         $display ("======> ALL TESTS PASSED");
304     else if (pass == 0)
305         $display ("======> ALL TESTS FAILED");
306     else
307         $display ("======> SOME TESTS FAILED");
308     #100;
309     $finish;
310 end
311 endmodule
```

Program 3: Testbench implementation of Exercise 1.

2.3 Testbench results

All test cases executed successfully, with each marked as `PASSED`. This outcome confirms that the test module behaves as intended and that the implemented RTL design meets the expected functionality.

```
# ----------------------------------------------------------------
# TEST 1: FSM AUTO MODE TEST
# ----------------------------------------------------------------
# [EXPECT] Time = 1030 | 7SEGs = 1f-03, HOR_LED = 100, VER_LED = 010
# [OUTPUT] Time = 1030 | 7SEGs = 1f-03, HOR_LED = 100, VER_LED = 010
# ======> PASSED
# [EXPECT] Time = 3460 | 7SEGs = 1f-02, HOR_LED = 100, VER_LED = 110
# [OUTPUT] Time = 3460 | 7SEGs = 1f-02, HOR_LED = 100, VER_LED = 110
# ======> PASSED
# [EXPECT] Time = 5090 | 7SEGs = 1f-03, HOR_LED = 010, VER_LED = 100
# [OUTPUT] Time = 5090 | 7SEGs = 1f-03, HOR_LED = 010, VER_LED = 100
# ======> PASSED
# [EXPECT] Time = 7520 | 7SEGs = 1f-02, HOR_LED = 110, VER_LED = 100
# [OUTPUT] Time = 7520 | 7SEGs = 1f-02, HOR_LED = 110, VER_LED = 100
# ======> PASSED
# [EXPECT] Time = 9150 | 7SEGs = 1f-03, HOR_LED = 100, VER_LED = 010
# [OUTPUT] Time = 9150 | 7SEGs = 1f-03, HOR_LED = 100, VER_LED = 010
# ======> PASSED
# ----------------------------------------------------------------
# TEST 2: FSM BUTTON TEST
# ----------------------------------------------------------------
# ----------------------------------------------------------------
#   1. Increase green 3 times, increase yellow 3 times, then save
# ----------------------------------------------------------------
# [EXPECT] Time = 16120 | 7SEGs = 2f-06, HOR_LED = 010, VER_LED = 010
# [OUTPUT] Time = 16120 | 7SEGs = 2f-06, HOR_LED = 010, VER_LED = 010
# ======> PASSED
# [EXPECT] Time = 24830 | 7SEGs = 3f-05, HOR_LED = 110, VER_LED = 110
# [OUTPUT] Time = 24830 | 7SEGs = 3f-05, HOR_LED = 110, VER_LED = 110
# ======> PASSED
# [EXPECT] Time = 29940 | 7SEGs = 1f-04, HOR_LED = 100, VER_LED = 010
# [OUTPUT] Time = 29940 | 7SEGs = 1f-04, HOR_LED = 100, VER_LED = 010
# ======> PASSED
# [EXPECT] Time = 34770 | 7SEGs = 1f-03, HOR_LED = 100, VER_LED = 110
# [OUTPUT] Time = 34770 | 7SEGs = 1f-03, HOR_LED = 100, VER_LED = 110
# ======> PASSED
# ----------------------------------------------------------------
#   2. Decrease green 2 times, decrease yellow 1 time, then save
# ----------------------------------------------------------------
# [EXPECT] Time = 40000 | 7SEGs = 2f-04, HOR_LED = 010, VER_LED = 010
# [OUTPUT] Time = 40000 | 7SEGs = 2f-04, HOR_LED = 010, VER_LED = 010
# ======> PASSED
# [EXPECT] Time = 45230 | 7SEGs = 3f-04, HOR_LED = 110, VER_LED = 110
# [OUTPUT] Time = 45230 | 7SEGs = 3f-04, HOR_LED = 110, VER_LED = 110
# ======> PASSED
# [EXPECT] Time = 49540 | 7SEGs = 1f-03, HOR_LED = 100, VER_LED = 010
# [OUTPUT] Time = 49540 | 7SEGs = 1f-03, HOR_LED = 100, VER_LED = 010
# ======> PASSED
```

```
# [EXPECT] Time = 51970 | 7SEGs = 1f-04, HOR_LED = 100, VER_LED = 110
# [OUTPUT] Time = 51970 | 7SEGs = 1f-04, HOR_LED = 100, VER_LED = 110
# ======> PASSED
# ------------------------------------------------------------------
#   3. Increase green 1 time, decrease yellow 1 time, then no save
# ------------------------------------------------------------------
# [EXPECT] Time = 55460 | 7SEGs = 2f-05, HOR_LED = 010, VER_LED = 010
# [OUTPUT] Time = 55460 | 7SEGs = 2f-05, HOR_LED = 010, VER_LED = 010
# ======> PASSED
# [EXPECT] Time = 58950 | 7SEGs = 3f-03, HOR_LED = 110, VER_LED = 110
# [OUTPUT] Time = 58950 | 7SEGs = 3f-03, HOR_LED = 110, VER_LED = 110
# ======> PASSED
# [EXPECT] Time = 62320 | 7SEGs = 1f-02, HOR_LED = 100, VER_LED = 010
# [OUTPUT] Time = 62320 | 7SEGs = 1f-02, HOR_LED = 100, VER_LED = 010
# ======> PASSED
# [EXPECT] Time = 65550 | 7SEGs = 1f-02, HOR_LED = 100, VER_LED = 110
# [OUTPUT] Time = 65550 | 7SEGs = 1f-02, HOR_LED = 100, VER_LED = 110
# ======> PASSED
# ------------------------------------------------------------------
#   4. Increase green 1 time, decrease yellow 1 time, then save
# ------------------------------------------------------------------
# [EXPECT] Time = 69040 | 7SEGs = 2f-05, HOR_LED = 010, VER_LED = 010
# [OUTPUT] Time = 69040 | 7SEGs = 2f-05, HOR_LED = 010, VER_LED = 010
# ======> PASSED
# [EXPECT] Time = 74270 | 7SEGs = 3f-03, HOR_LED = 110, VER_LED = 110
# [OUTPUT] Time = 74270 | 7SEGs = 3f-03, HOR_LED = 110, VER_LED = 110
# ======> PASSED
# [EXPECT] Time = 79380 | 7SEGs = 1f-03, HOR_LED = 100, VER_LED = 010
# [OUTPUT] Time = 79380 | 7SEGs = 1f-03, HOR_LED = 100, VER_LED = 010
# ======> PASSED
# [EXPECT] Time = 82610 | 7SEGs = 1f-02, HOR_LED = 100, VER_LED = 110
# [OUTPUT] Time = 82610 | 7SEGs = 1f-02, HOR_LED = 100, VER_LED = 110
# ======> PASSED
# ------------------------------------------------------------------
# TEST 3: RESET TEST
# ------------------------------------------------------------------
# [EXPECT] Time = 84360 | 7SEGs = 2f-05, HOR_LED = 010, VER_LED = 010
# [OUTPUT] Time = 84360 | 7SEGs = 2f-05, HOR_LED = 010, VER_LED = 010
# ======> PASSED
# Resetting...
# [EXPECT] Time = 85990 | 7SEGs = 1f-00, HOR_LED = 100, VER_LED = 100
# [OUTPUT] Time = 85990 | 7SEGs = 1f-00, HOR_LED = 100, VER_LED = 100
# ======> PASSED
# [EXPECT] Time = 87740 | 7SEGs = 1f-00, HOR_LED = 100, VER_LED = 100
# [OUTPUT] Time = 87740 | 7SEGs = 1f-00, HOR_LED = 100, VER_LED = 100
# ======> PASSED
# [EXPECT] Time = 89490 | 7SEGs = 1f-00, HOR_LED = 100, VER_LED = 100
# [OUTPUT] Time = 89490 | 7SEGs = 1f-00, HOR_LED = 100, VER_LED = 100
# ======> PASSED
# Releasing reset...
# [EXPECT] Time = 91120 | 7SEGs = 1f-04, HOR_LED = 100, VER_LED = 010
# [OUTPUT] Time = 91120 | 7SEGs = 1f-04, HOR_LED = 100, VER_LED = 010
# ======> PASSED
```

```
# [EXPECT] Time = 95150 | 7SEGs = 1f-02, HOR_LED = 100, VER_LED = 110
# [OUTPUT] Time = 95150 | 7SEGs = 1f-02, HOR_LED = 100, VER_LED = 110
# ======> PASSED
# ---------------------------------------------------------------------
# TEST 4: MIN, MAX BOUNDARY TEST
# ---------------------------------------------------------------------
# ---------------------------------------------------------------------
#   1. Increase green 94 times, increase yellow 17 times to reach max
# ---------------------------------------------------------------------
# [EXPECT] Time = 260460 | 7SEGs = 2f-99, HOR_LED = 010, VER_LED = 010
# [OUTPUT] Time = 260460 | 7SEGs = 2f-99, HOR_LED = 010, VER_LED = 010
# ======> PASSED
# [EXPECT] Time = 293530 | 7SEGs = 3f-20, HOR_LED = 110, VER_LED = 110
# [OUTPUT] Time = 293530 | 7SEGs = 3f-20, HOR_LED = 110, VER_LED = 110
# ======> PASSED
# ---------------------------------------------------------------------
#    2. Increase green 3 times, increase yellow 5 times
# ---------------------------------------------------------------------
# [EXPECT] Time = 303980 | 7SEGs = 2f-99, HOR_LED = 010, VER_LED = 010
# [OUTPUT] Time = 303980 | 7SEGs = 2f-99, HOR_LED = 010, VER_LED = 010
# ======> PASSED
# [EXPECT] Time = 316170 | 7SEGs = 3f-20, HOR_LED = 110, VER_LED = 110
# [OUTPUT] Time = 316170 | 7SEGs = 3f-20, HOR_LED = 110, VER_LED = 110
# ======> PASSED
# ---------------------------------------------------------------------
#    3. Decrease green 97 times, decrease yellow 19 times to reach min
# ---------------------------------------------------------------------
# [EXPECT] Time = 490180 | 7SEGs = 2f-02, HOR_LED = 010, VER_LED = 010
# [OUTPUT] Time = 490180 | 7SEGs = 2f-02, HOR_LED = 010, VER_LED = 010
# ======> PASSED
# [EXPECT] Time = 526730 | 7SEGs = 3f-01, HOR_LED = 110, VER_LED = 110
# [OUTPUT] Time = 526730 | 7SEGs = 3f-01, HOR_LED = 110, VER_LED = 110
# ======> PASSED
# ---------------------------------------------------------------------
#    4. Decrease green 8 times, decrease yellow 3 times
# ---------------------------------------------------------------------
# [EXPECT] Time = 545880 | 7SEGs = 2f-02, HOR_LED = 010, VER_LED = 010
# [OUTPUT] Time = 545880 | 7SEGs = 2f-02, HOR_LED = 010, VER_LED = 010
# ======> PASSED
# [EXPECT] Time = 554590 | 7SEGs = 3f-01, HOR_LED = 110, VER_LED = 110
# [OUTPUT] Time = 554590 | 7SEGs = 3f-01, HOR_LED = 110, VER_LED = 110
# ======> PASSED
# ---------------------------------------------------------------------
# SUMMARY
# ---------------------------------------------------------------------
# TOTAL TESTS : 35
# TOTAL PASSED: 35
# TOTAL FAILED: 0
# ======> ALL TESTS PASSED
# ** Note: $finish    : ../tb/tb.v(308)
#    Time: 558270 ns  Iteration: 0  Instance: /tb
```

Simulation 1: Testbench results of Exercise 1.

2.4 Simulation waveform

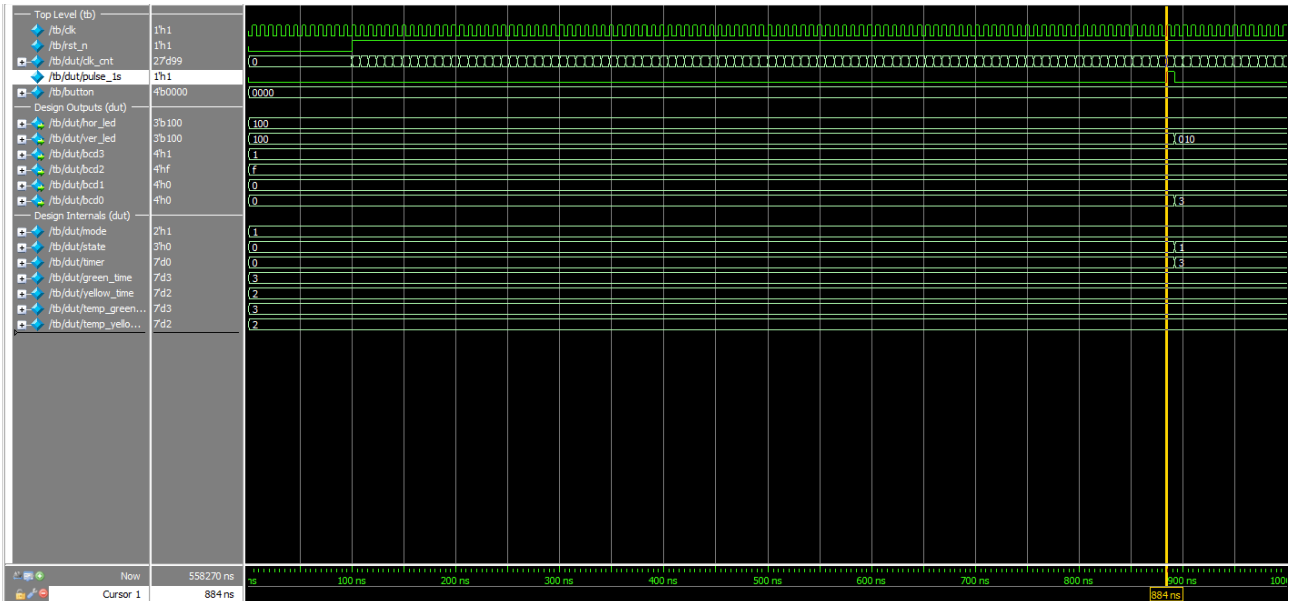

Figure 1.1

In the simulation, one second corresponds to 100 clock cycles, and the debounce interval is set to 10 cycles for easier observation compared to the actual FPGA values.
Figure 1.1 illustrates the behavior when the reset signal is asserted and then deasserted, so both the horizontal and vertical LEDs are initialized to their default state, showing red.
After 100 clock cycles, tracked by the `clk_cnt` signal, a pulse is generated to represent the one-second duration. At 884 ns, the `pulse_1s` signal is asserted for a single clock cycle and then deasserted in the following cycle.
On the next cycle after `pulse_1s` is asserted, the traffic light controller begins operation. The initial state sets the horizontal LED to red and the vertical LED to green, marking the start of the traffic light sequence.

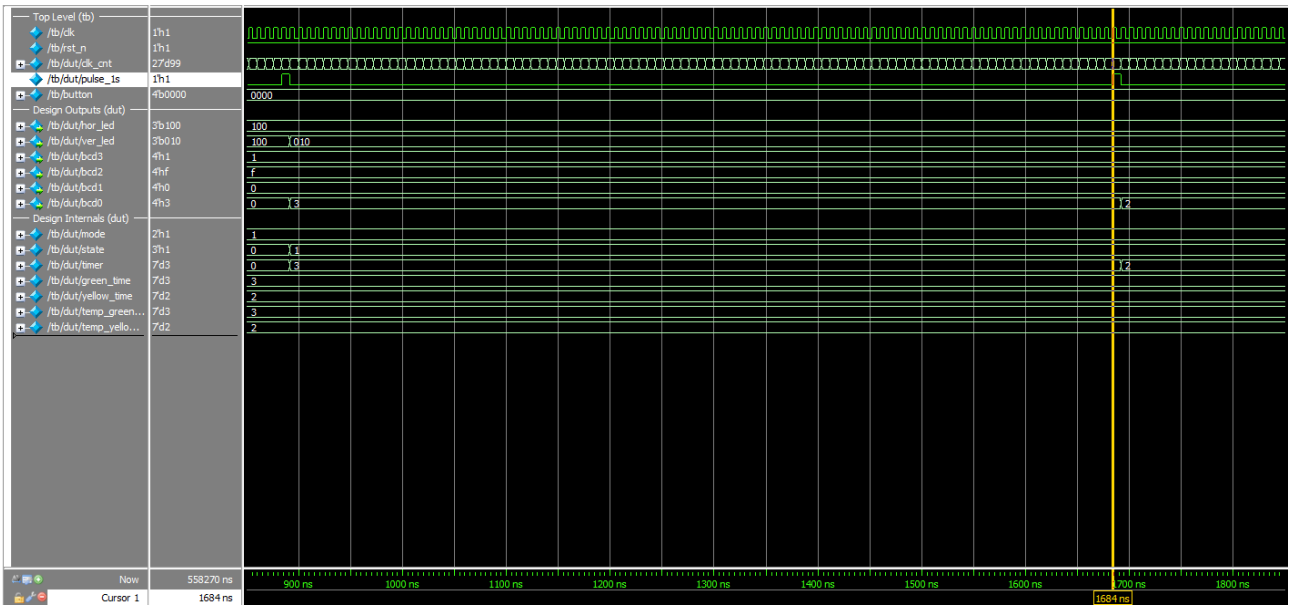

Figure 1.2

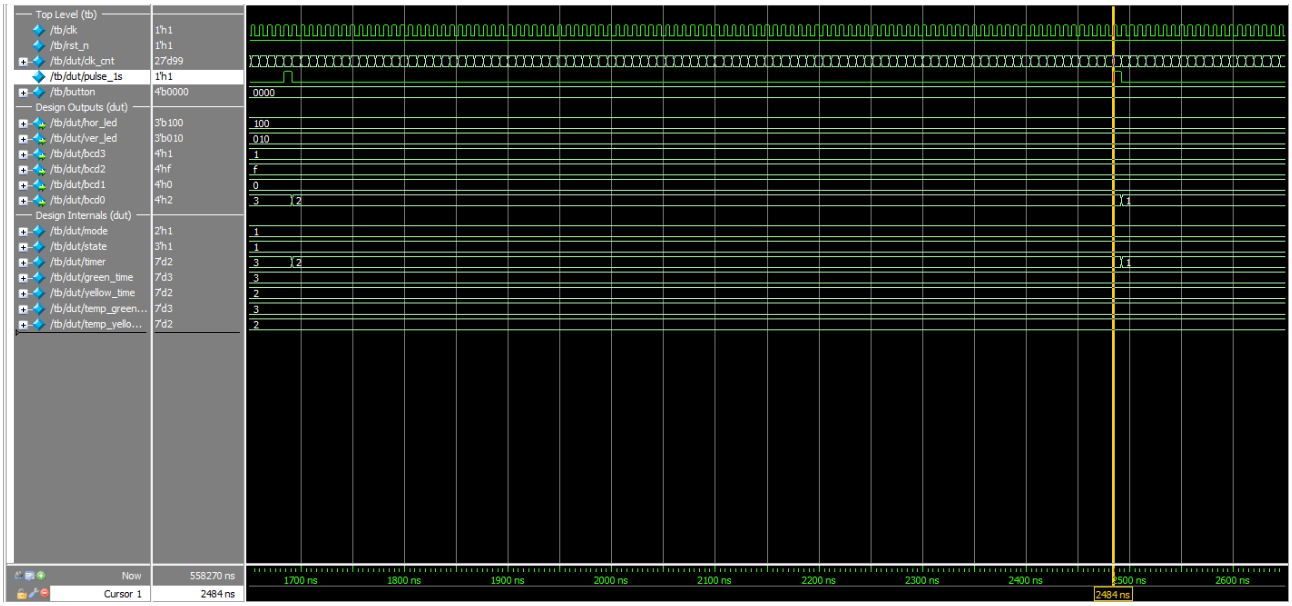

Figure 1.3

The `pulse_1s` signal indicates that one second has elapsed. The duration of the red LED is defined as the sum of the green and yellow LED intervals. For easier timing analysis, this module tracks only the durations of the green and yellow LEDs.
Every 100 clock cycles, the `pulse_1s` signal is asserted, and on the following cycle, the traffic lights are updated.
As shown in Figure 1.2, at 1684 ns, the horizontal traffic light remains red, while the vertical traffic light updates its green LED countdown from 3 seconds to 2 seconds.
In Figure 1.3, at 2484 ns, another `pulse_1s` event occurs, marking the passage of one second. This triggers the vertical traffic light to update again, reducing the green LED countdown from 2 seconds to 1 second.

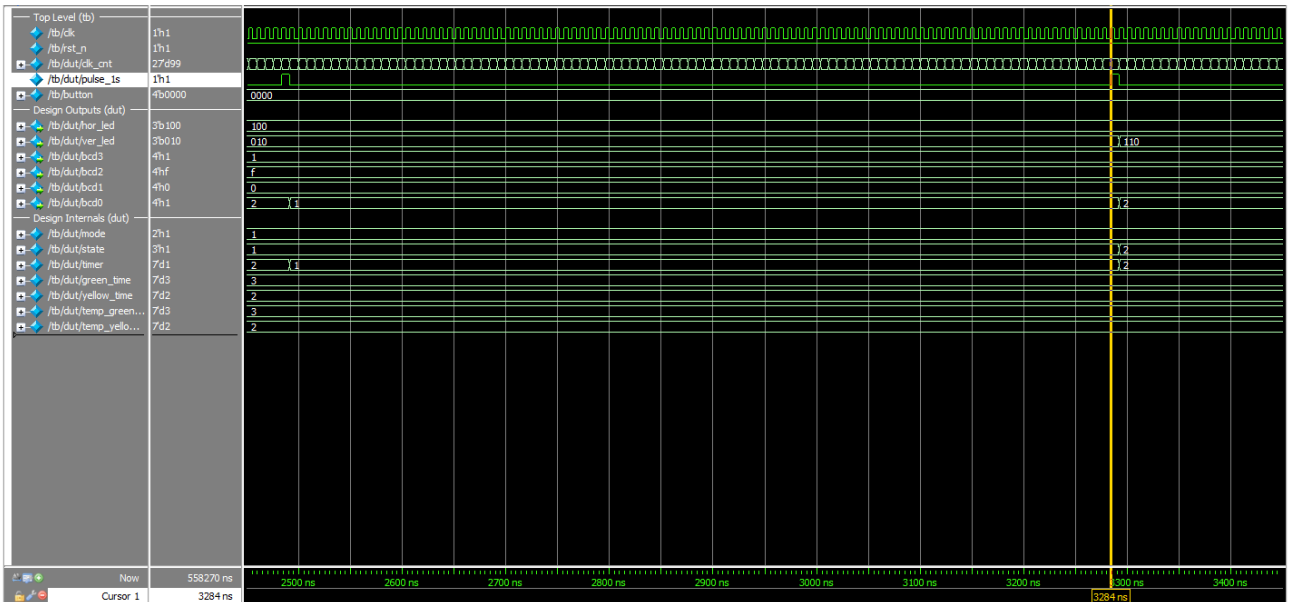

Figure 1.4

In Figure 1.4, another second has elapsed, completing the 3-second countdown for the vertical traffic light's green LED. Consequently, at the next clock cycle after `pulse_1s` is asserted at 3284 ns, the vertical traffic light transitions to the yellow LED, which has a 2-second duration. This demonstrates how both traffic lights automatically update their states. The behavior can

be observed more clearly in Test 1 of the test bench, where the system waits for one LED countdown to finish and then verifies that both traffic lights update accordingly.

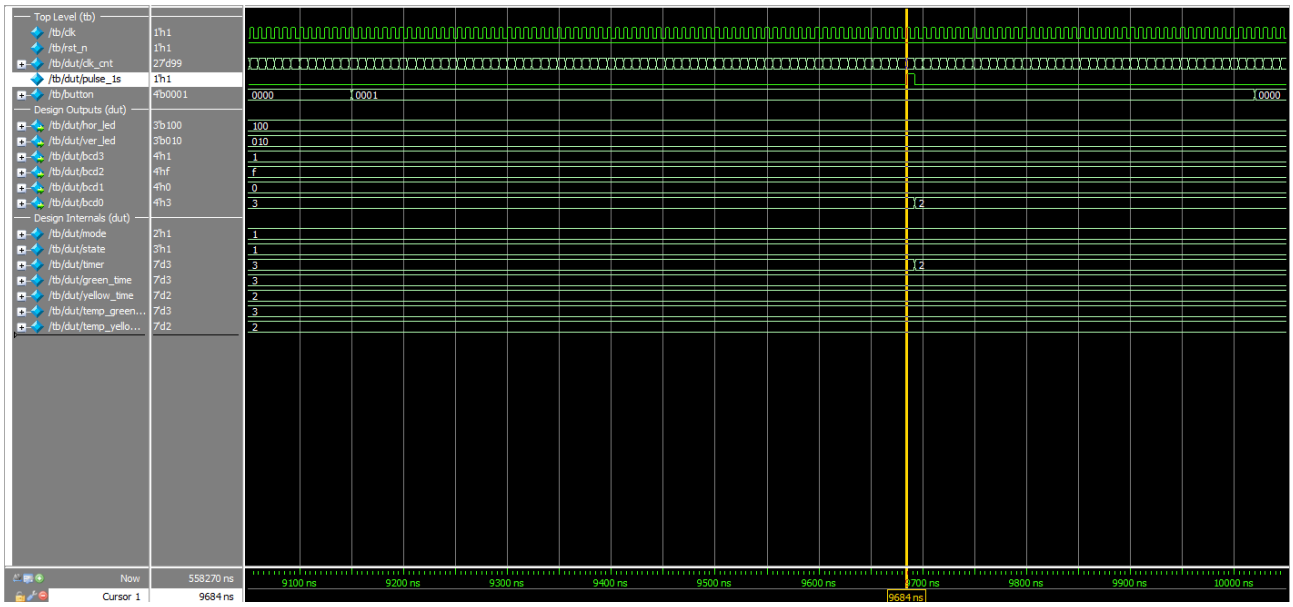

Figure 1.5

Figure 1.5 illustrates the conclusion of the auto mode test, during which no buttons are pressed. When Button 0 is activated (represented by the `button` signal), the system transitions out of auto mode.
Once the `button[0]` signal is asserted (Figure 1.5), then deasserted (Figure 1.6), the `bcd3` signal updates to a value of 2, indicating a switch to Mode 2. In this mode, the duration of the green LEDs for the traffic lights can be modified and both the horizontal and vertical LEDs turn green, confirming that Mode 2 is active.

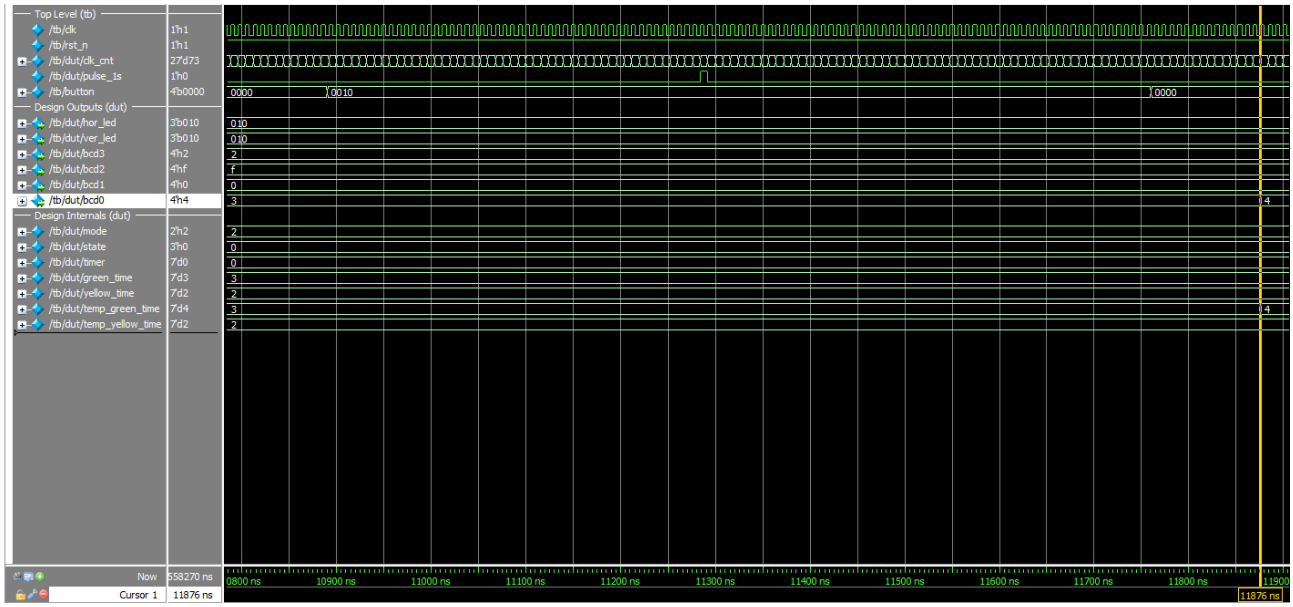

Figure 1.6

Figure 1.6 also shows that Button 1 is pressed, which functions as the increment control for the green LED duration. After the button is released and the debounce interval has passed, the `bcd0` and `temp_green_time` signals update to a value of 4. This confirms that the 7-segment LED display correctly reflects the button input.

Since Button 3 (the save function) has not yet been pressed, the `green_time` signal retains its previous value, meaning the updated duration for the green LEDs has not been stored.

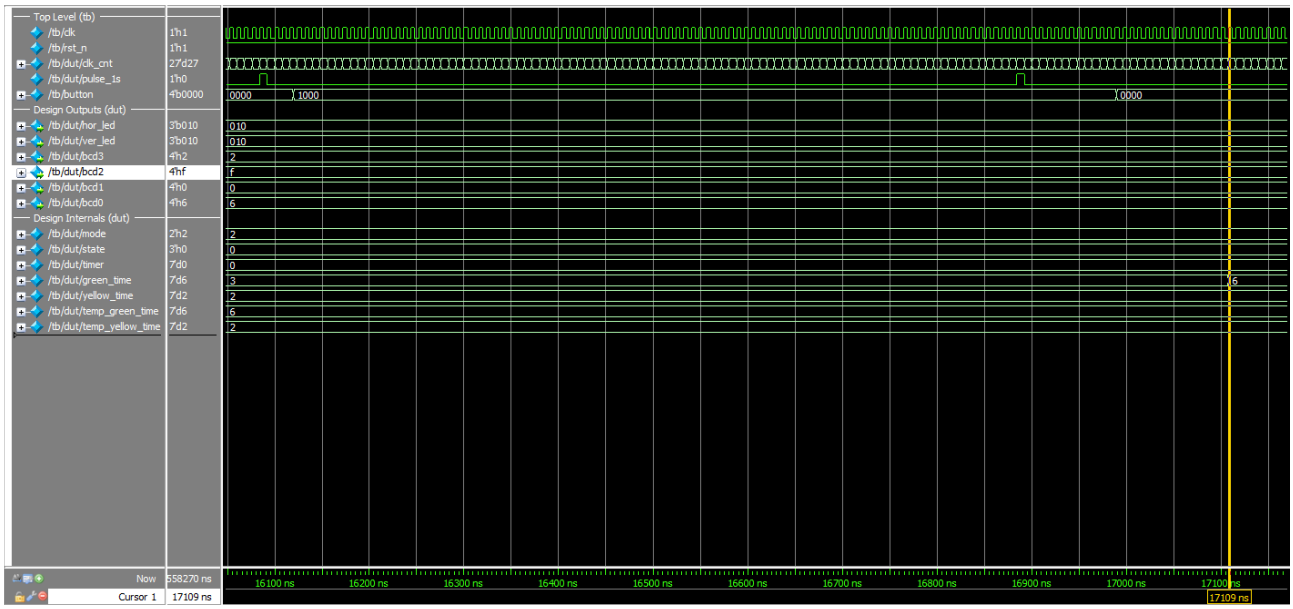

Figure 1.7

After pressing Button 1 three times, the green LED countdown reaches 6 seconds. To save this value, Button 3 is pressed.
As shown, once Button 3 is pressed and released - after the debounce interval - the `green_time` signal updates to 6, confirming that the green LEDs have been successfully reconfigured to count down for 6 seconds.

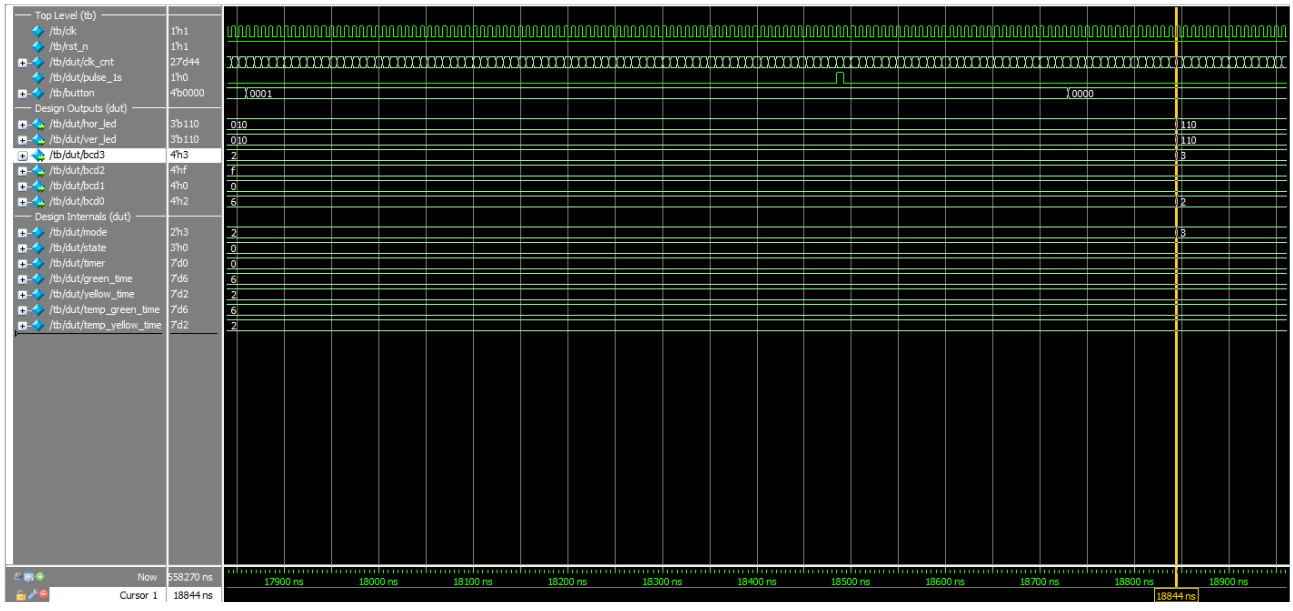

Figure 1.8

Figure 1.8 illustrates the action of pressing Button 0, which triggers the FSM to switch to Mode 3. In this mode, both the vertical and horizontal LEDs turn yellow, confirming that the system has successfully transitioned to Mode 3.

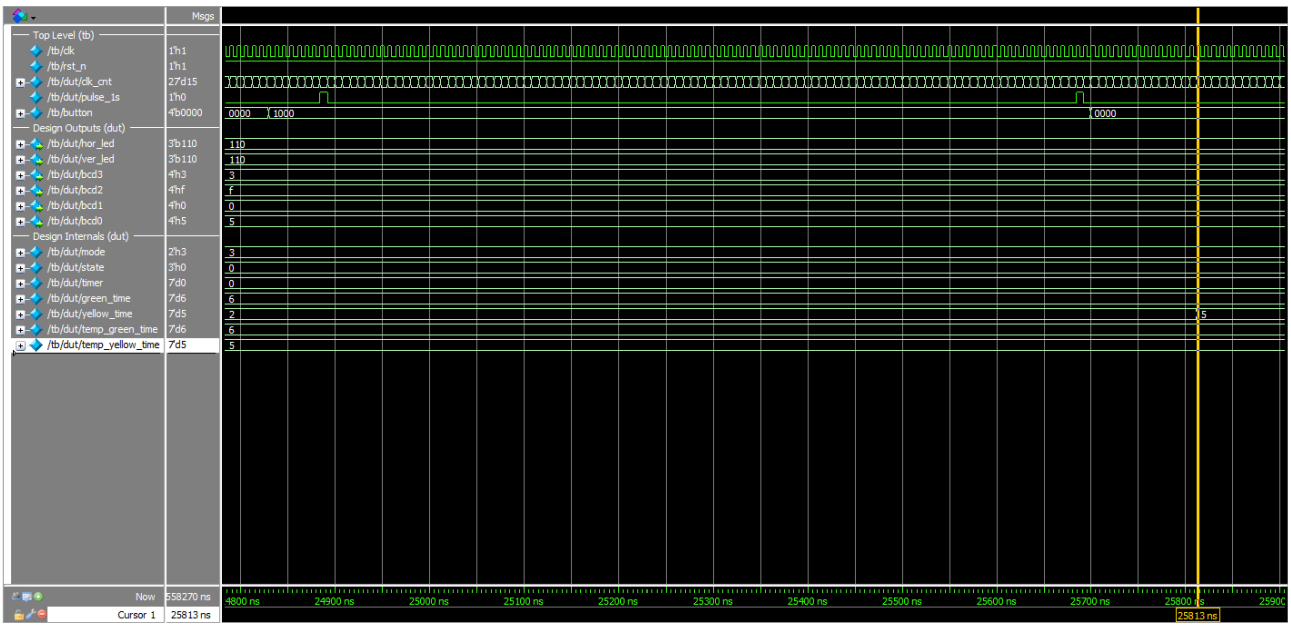

Figure 1.9

Figure 1.9 illustrates that the user has temporarily configured the yellow LED duration, increasing it from 2 seconds to 5 seconds through the `temp_yellow_time` signal.
When Button 3 is pressed, the updated value is saved, and the `yellow_time` signal is set to 5, confirming that the yellow LEDs have been successfully reconfigured to a 5-second duration.

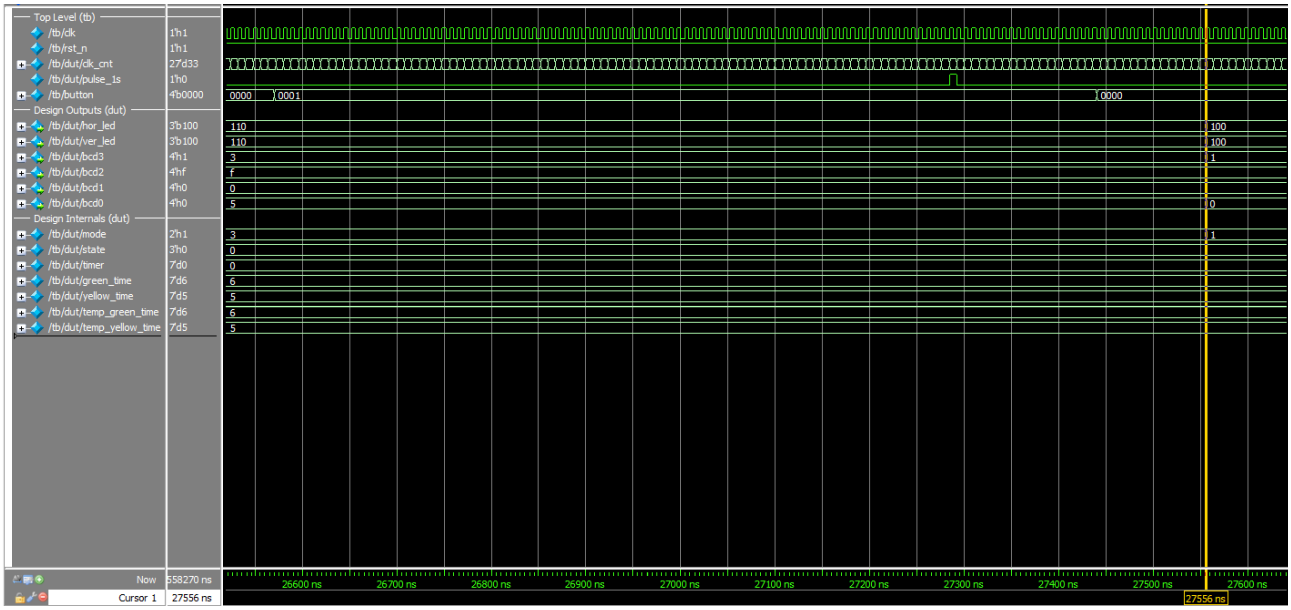

Figure 1.10

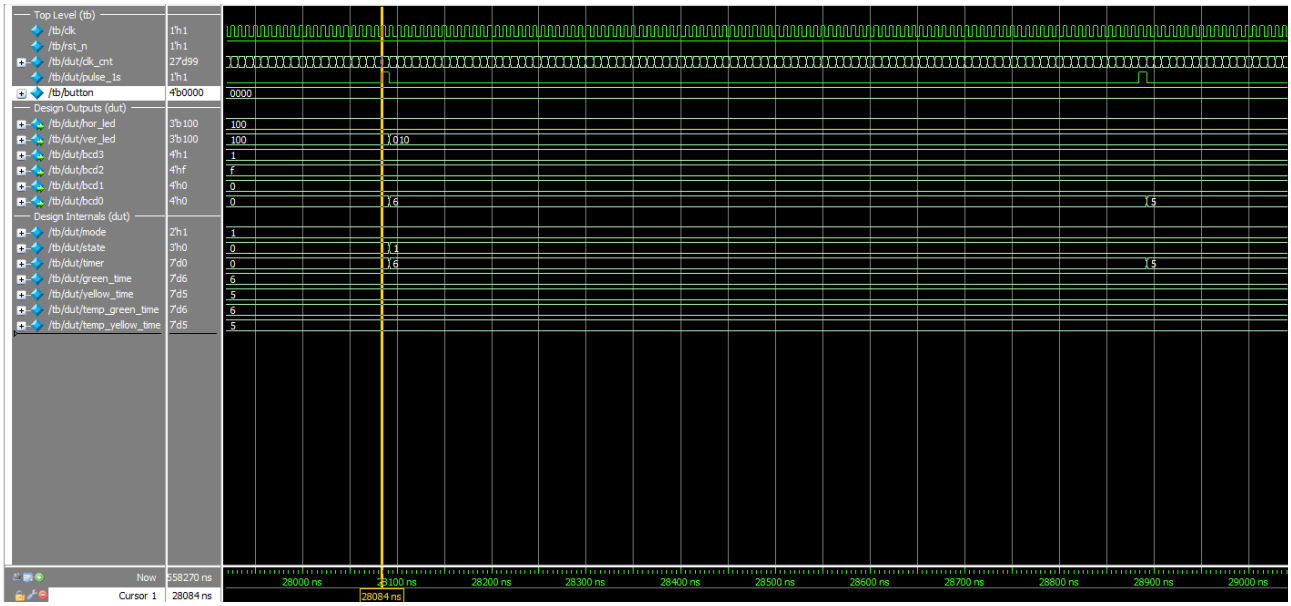

Figure 1.11

Figures 1.10 and 1.11 illustrate the system returning to running mode after Button 0 is pressed. At this point, the 7-segment display and LEDs reset to their initial state, with both directions showing red LEDs and the countdown timer set to 0.
When the next `pulse_1s` signal is asserted, the traffic lights resume normal operation using the newly configured values. The green light now runs with a 6-second duration, replacing the previous 3-second interval.
Upon the following `pulse_1s` assertion, the countdown successfully decreases to 5 seconds, confirming that the configuration has been saved and that the traffic lights operate correctly with the updated timing.

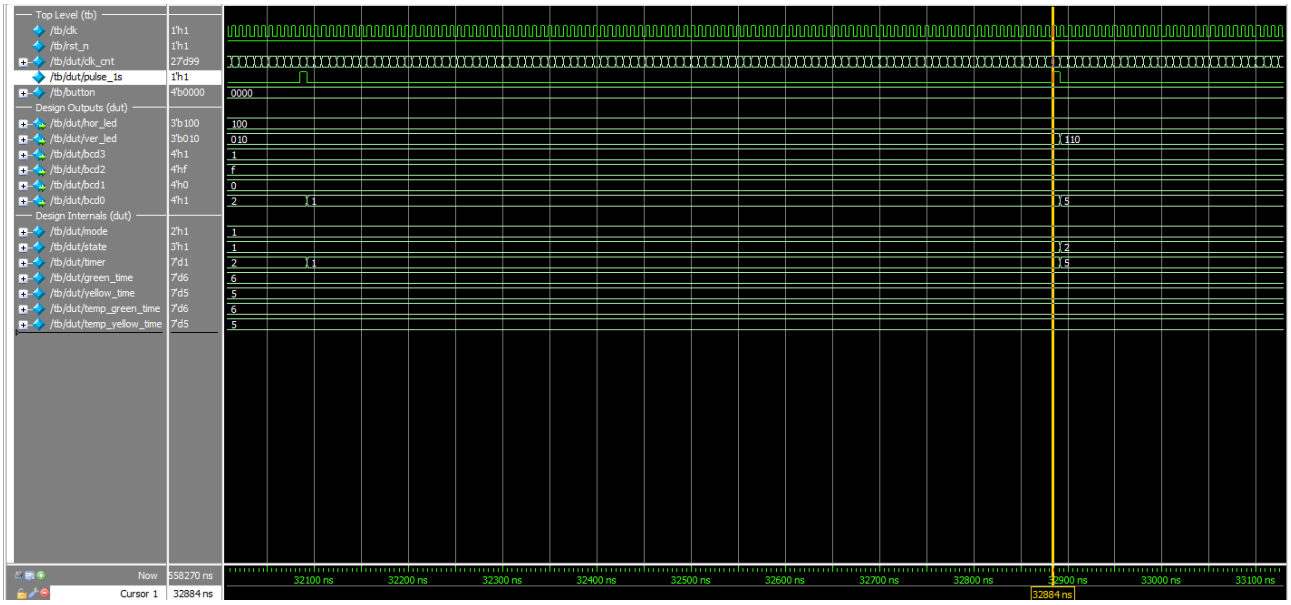

Figure 1.12

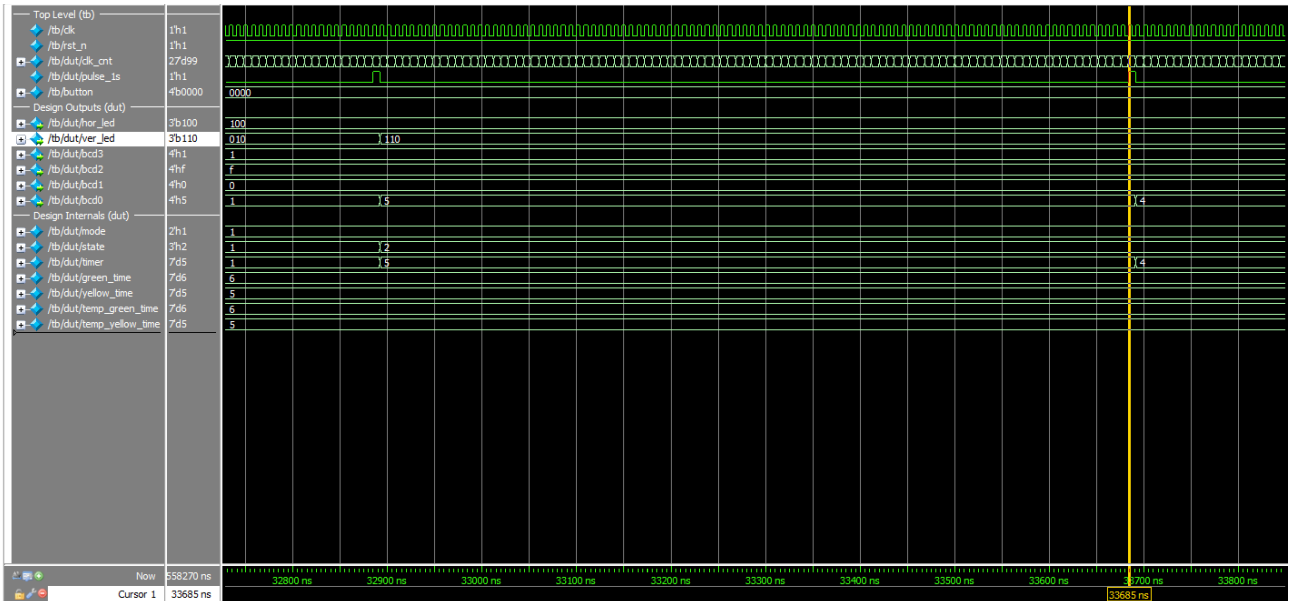

Figure 1.13

We now skip to the point where the vertical green LED finishes its countdown and transitions to the yellow LED, in order to verify whether the updated value has been applied.
Figure 1.12 shows that after the green LED counts down to 1 second, another `pulse_1s` is asserted, switching the vertical light to the yellow LED with a duration of 5 seconds, as indicated by the `timer` signal.
In Figure 1.13, when one second elapses (signaled by a high `pulse_1s`), the timer updates to 4 seconds. This confirms that both the green LED duration and the yellow LED duration have been successfully updated through button presses, and the traffic lights continue to operate correctly with the new values.

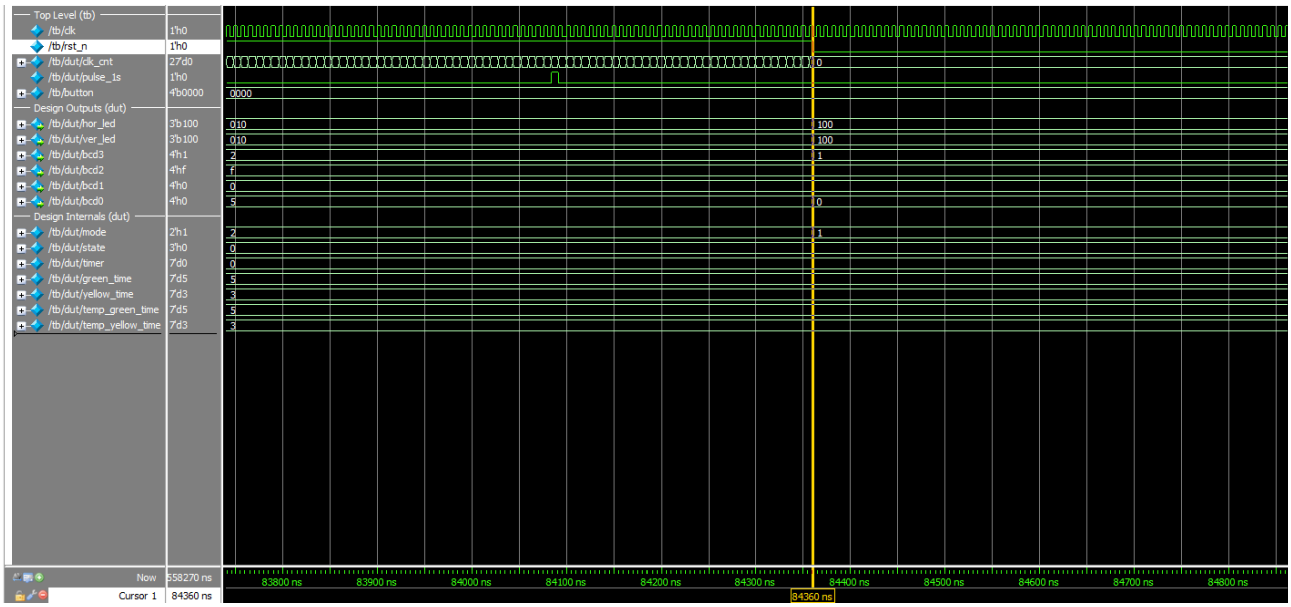

Figure 1.14

Since the design uses an asynchronous active-low reset, asserting the reset immediately forces both the vertical and horizontal LEDs to red. At the same time, the 7-segment display is updated to Mode 1, with the countdown value set to 0, as indicated by the signals `bcd3`, `bcd2`, `bcd1`, and `bcd0`.

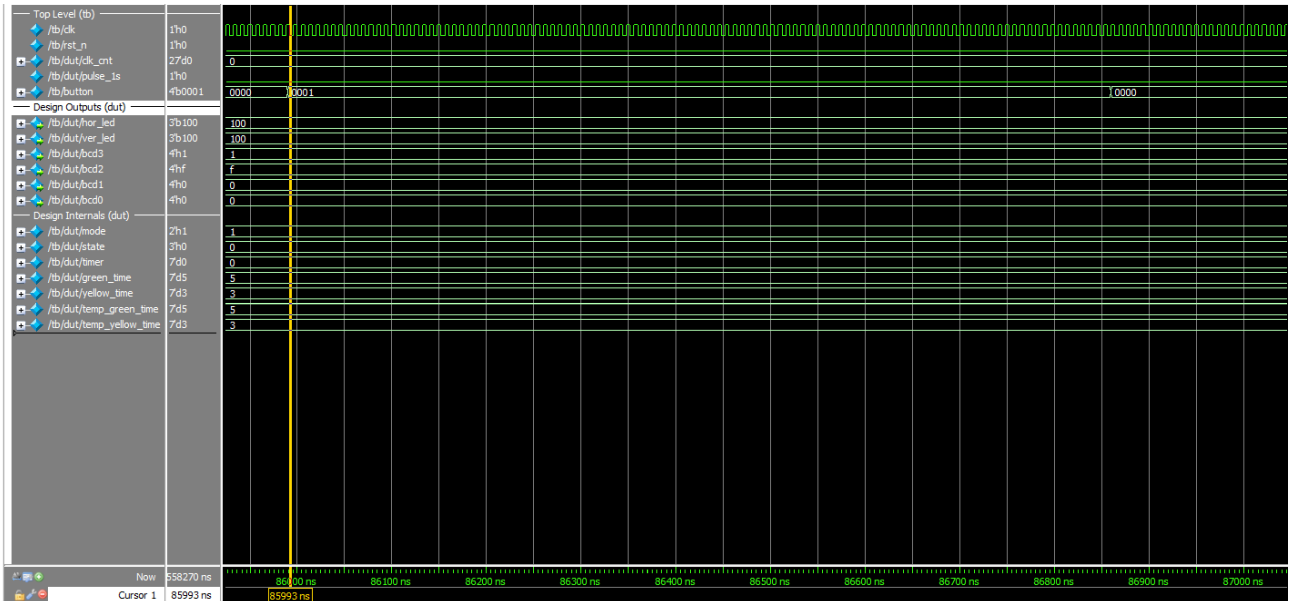

Figure 1.15

By observing Figure 1.15, we can see that pressing Button 1 has no effect while the reset signal is active. This confirms that the reset mechanism is functioning correctly.
The behavior is demonstrated more clearly in Test 3 of the test bench module, which specifically verifies the reset functionality.

2.5 FPGA results

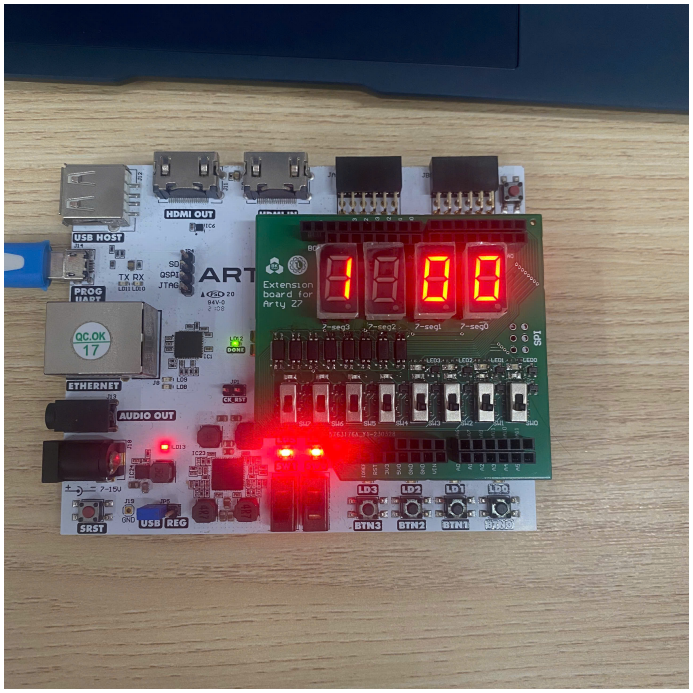

Figure 1.16

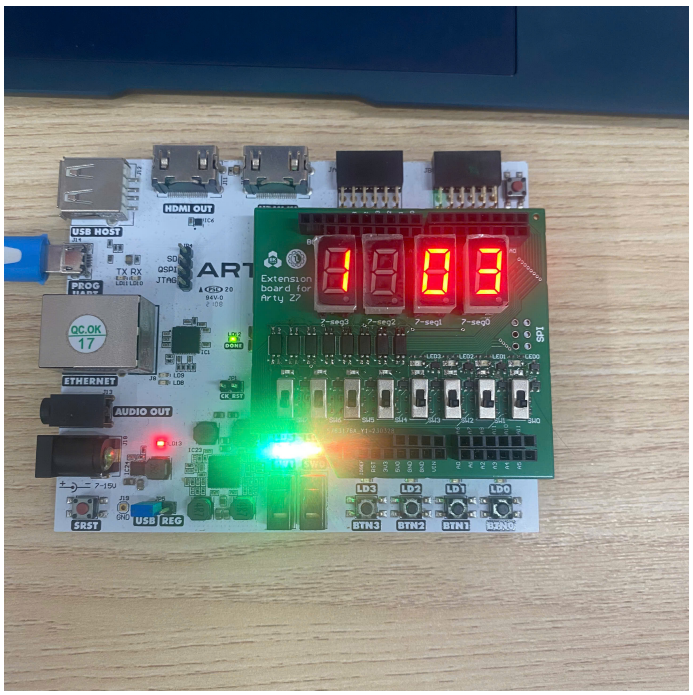

Figure 1.17

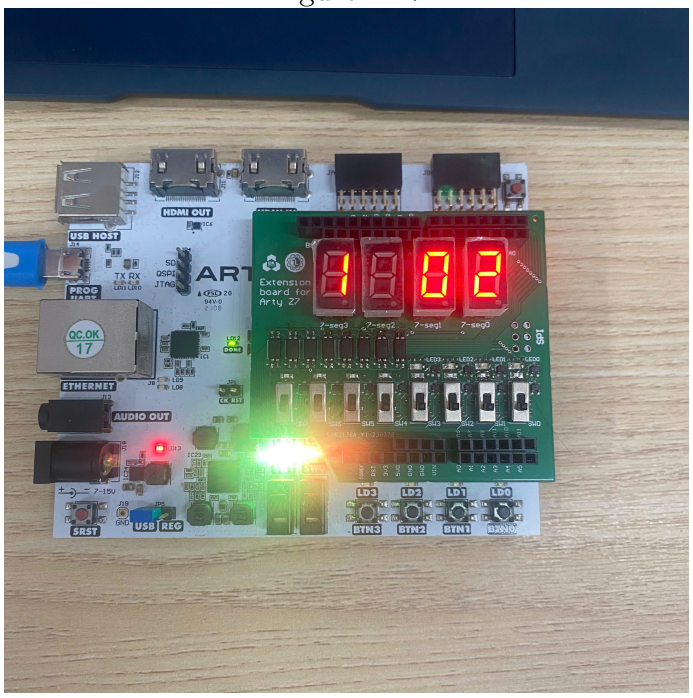

Figure 1.18

The three picture illustrates a part of the FPGA implementation results. Figure 1.16 shows the initial states when reset is on. Figure 1.17 shows the first run state where horizontal LED is red and vertical LED is green. The two leftmost 7-segment LEDs show number 1 meaning Mode 1 is on while the two rightmost 7-segment LEDs shows number 3 in Figure 1.17 meaning it start to count down at second 3 based on vertical green LED. Figure 1.18 shows what happen 1 seconds later, where the two rightmost 7-segment LEDs show number 2 meaning the FPGA has counted down successfully.

3 Exercise 2

3.1 RTL implementation

This module implements a digital display controller with LED indicators and 4-digit BCD outputs that scroll or bounce a numeric pattern across a physical 7-segment display. The system support two display effects:

- Effect 1: The 7-segment LEDs display the numeric “25” scrolling from left to right repeatedly. During this mode, a LED indicator will show red color to notify the user effect 1 is on.
- Effect 2: The 7-segment LEDs display the numeric “25”. When the test reach the right edge, it will shift left; on the other hand, when the test reaches the left edge, it will shift right. During this mode, a LED indicator will show green color to notify the user effect 2 is on.

A switch is used as a reset for the system. Once the switch is activated, the 7-segment LEDs display always present the text ”25” on the leftmost side. Another switch is used to set effect, when the switch is off, the system runs with effect 1, when the switch is on, it changes to effect 2. The switch manipulates the effect will not be affected by the reset switch. When user configures the system to run with effect 2 then turn on the reset switch, the LED indicator still shows green color, as for indicating when the reset is released the system will run with effect 2 from the start. When the text is displaying, switching effect will only affects how the text behaves, not reset the text to initial state then run in the activated effect.

```
module ex2 (
    input wire clk,
    input wire rst_n,
    input wire sw,
    output reg [2:0] led,
    output reg [15:0] bcd
);

`ifdef SIMULATION
    //for sim, a sec is 100 cycles
    parameter CLK_FREQ = 27'd100;
`else
    //for fpga, a sec is 125m cycles
    parameter CLK_FREQ = 27'd125_000_000;
`endif

parameter SHIFT_FREQ = 2;
parameter DIV_COUNT = CLK_FREQ / SHIFT_FREQ;

parameter RED = 3'b100;
parameter GREEN = 3'b010;

parameter BCD_2 = 4'd2;
parameter BCD_5 = 4'd5;
parameter BCD_BLANK = 4'hF;
parameter ALL_BLANK = 16'hFFFF;

```

```
parameter S0 = {BCD_2, BCD_5, BCD_BLANK, BCD_BLANK}; // "25__"
parameter S1 = {BCD_BLANK, BCD_2, BCD_5, BCD_BLANK}; // "_25_"
parameter S2 = {BCD_BLANK, BCD_BLANK, BCD_2, BCD_5}; // "__25"
parameter S3 = {BCD_5, BCD_BLANK, BCD_BLANK, BCD_2}; // "5__2"

reg [26:0] clk_cnt;
wire tick;
reg [1:0] state;
reg direction; //0: left to right, 1: right to left

assign tick = (clk_cnt == DIV_COUNT - 1);

always @(posedge clk or negedge rst_n) begin
    if(!rst_n)
        clk_cnt <= 27'd0;
    else if (tick)
        clk_cnt <= 27'd0;
    else
        clk_cnt <= clk_cnt + 1;
end

always @(posedge clk or negedge rst_n) begin
    if (!rst_n) begin
        state <= 0;
        direction <= 0;
    end else if (tick) begin
        //scroll
        if (sw == 0) begin
            if (state == 2'd3)
                state <= 2'd0;
            else
                state <= state + 1;
        end else begin
        //bounce
            if (direction == 0) begin
                if (state == 2'd2) begin
                    direction <= 1;
                    state <= 2'd1;
                end else if (state == 2'd3) begin
                    direction <= 1;
                    state <= 2'd2;
                end else
                    state <= state + 1;
            end else begin
                if (state == 2'd0) begin
                    direction <= 0;
                    state <= 2'd1;
                end else
                    state <= state - 1;
            end
        end
    end else begin
        state <= state;
```

```
81          direction <= direction;
82      end
83  end
84
85  always @(*) begin
86      case (state)
87          2'd0: bcd = S0;
88          2'd1: bcd = S1;
89          2'd2: bcd = S2;
90          2'd3: bcd = S3;
91          default: bcd = ALL_BLANK;
92      endcase
93
94      if (sw == 0)
95          led = RED;
96      else
97          led = GREEN;
98  end
99  endmodule
```

Program 4: RTL implementation of Exercise 2.

3.2 Testbench

The testbench used to verify the correct operation of the RTL design for Exercise 2. It tests the design across several operating scenarios to ensure correct state transitions, output functionality and response to user input.

- Test 1 (Initial values check): Verifies that, after reset, the output values display on the 7-segment LEDs and the indicator LED are set correctly to their initial, expected values.
- Test 2 (Effect 1 check): With the switch set to 0, the testbench checks that the display pattern scrolls through all states in the correct order, looping back to the start.
- Test 3 (Effect 2 check): With the switch set to 1, the testbenh verifies that the display pattern bounces back and forth between the ends, changing direction as expected.
- Test 4 and 5 (Mode switching): The test alternates between scroll and bounce modes, ensuring that the state machine handles mode changes correctly and outputs the proper LED color and display pattern immediately after each switch.
- Test 6 (Reset functionality check): The testbench asserts and de-asserts the reset signal during both scroll and bounce modes, checking whether the module correctly returns to its initial state after a reset.

```
1  module tb;
2  reg clk;
3  reg rst_n;
4  reg sw;
5  wire [2:0] led;
6  wire [15:0] bcd;
7
8  parameter CLK_PERIOD = 8; // 8ns => 125MHz
9  parameter TICK_WAIT = (CLK_PERIOD * 50);
```

```
10
11  parameter RED = 3'b100;
12  parameter GREEN = 3'b010;
13
14  parameter BCD_2 = 4'd2;
15  parameter BCD_5 = 4'd5;
16  parameter BCD_BLANK = 4'hF;
17  parameter ALL_BLANK = 16'hFFFF;
18
19  parameter S0 = {BCD_2, BCD_5, BCD_BLANK, BCD_BLANK}; // "25__"
20  parameter S1 = {BCD_BLANK, BCD_2, BCD_5, BCD_BLANK}; // "_25_"
21  parameter S2 = {BCD_BLANK, BCD_BLANK, BCD_2, BCD_5}; // "__25"
22  parameter S3 = {BCD_5, BCD_BLANK, BCD_BLANK, BCD_2}; // "5__2"
23
24  integer pass, fail;
25
26  ex2 dut (
27      .clk (clk),
28      .rst_n (rst_n),
29      .sw (sw),
30      .led (led),
31      .bcd (bcd)
32  );
33
34  task divi;
35      $display ("
           ------------------------------------------------------------
36      --------------");
37  endtask
38
39  task msg (input [700:0] txt);
40      begin
41          divi();
42          $display ("%0s", txt);
43          divi();
44      end
45  endtask
46
47  task check (input [15:0] exp_bcd, input [2:0] exp_led);
48      begin
49          $display ("[OUTPUT] Time = %0t | 7SEG = 16'h%4h, LED = 3'b%3b"
               , $time, bcd, led);
50          $display ("[EXPECT] Time = %0t | 7SEG = 16'h%4h, LED = 3'b%3b"
               , $time, exp_bcd, exp_led);
51
52          if ((bcd === exp_bcd) && (led === exp_led)) begin
53              $display ("======> PASSED");
54              pass = pass + 1;
55          end else begin
56              $display ("======> FAILED");
57              fail = fail + 1;
58          end
59      end
```

```
60  endtask
61
62  task mode (input tsw);
63      begin
64          divi();
65          $display ("[SWITCH] Time = %0t | Switching to effect %0d",
                $time, tsw + 1);
66          divi();
67          sw = tsw;
68      end
69  endtask
70
71  initial begin
72      clk = 0;
73      forever #(CLK_PERIOD/2) clk = ~clk;
74  end
75
76  initial begin
77      rst_n = 0;
78      sw = 0;
79      pass = 0;
80      fail = 0;
81      #100;
82      rst_n = 1;
83      #100;
84
85      msg ("TEST 1: INITIAL VALUES CHECK");
86      check (S0, RED);
87
88      msg ("TEST 2: SCROLL MODE CHECK");
89      #(TICK_WAIT);
90      check (S1, RED);
91
92      #(TICK_WAIT);
93      check (S2, RED);
94
95      #(TICK_WAIT);
96      check (S3, RED);
97
98      #(TICK_WAIT);
99      check (S0, RED);
100
101     #(TICK_WAIT);
102     check (S1, RED);
103
104     msg ("TEST 3: BOUNCE MODE CHECK");
105     mode (1);
106     #10;
107     check (S1, GREEN);
108
109     #(TICK_WAIT);
110     check (S2, GREEN);
111
```

```
112     #(TICK_WAIT);
113     check (S1, GREEN);
114 
115     #(TICK_WAIT);
116     check (S0, GREEN);
117 
118     #(TICK_WAIT);
119     check (S1, GREEN);
120 
121     msg ("TEST 4: SWITCH BACK TO SCROLL MODE");
122     mode (0);
123     #10;
124     check (S1, RED);
125 
126     #(TICK_WAIT);
127     check (S2, RED);
128 
129     #(TICK_WAIT);
130     check (S3, RED);
131 
132     msg ("TEST 5: SWITCH BACK TO BOUNCE MODE");
133     mode (1);
134     #10;
135     check (S3, GREEN);
136 
137     #(TICK_WAIT);
138     check (S2, GREEN);
139 
140     #(TICK_WAIT);
141     check (S1, GREEN);
142 
143     #(TICK_WAIT);
144     check (S0, GREEN);
145 
146     #(TICK_WAIT);
147     check (S1, GREEN);
148 
149     msg ("TEST 6: RESET CHECK");
150     $display ("Reseting...");
151     @(posedge clk);
152     rst_n = 0;
153     #10;
154     check (S0, GREEN);
155 
156     #(TICK_WAIT);
157     check (S0, GREEN);
158 
159     mode (0);
160     #10;
161     check (S0, RED);
162 
163     mode (1);
164     $display ("Releasing reset...");
```

```
    #(TICK_WAIT);
    rst_n = 1;
    #10;

    check (S0, GREEN);

    #(TICK_WAIT);
    check (S1, GREEN);

    #(TICK_WAIT);
    check (S2, GREEN);

    #(TICK_WAIT);
    check (S1, GREEN);

    #(TICK_WAIT);
    check (S0, GREEN);

    #(TICK_WAIT);
    check (S1, GREEN);

    msg ("SUMMARY");
    $display ("TOTAL TESTS : %0d", pass + fail);
    $display ("Total PASSED: %0d", pass);
    $display ("Total FAILED: %0d", fail);

    if (fail == 0)
        $display ("======> ALL TESTS PASSED");
    else if (pass == 0)
        $display ("======> ALL TESTS FAILED");
    else
        $display ("SOME TESTS FAILED");

    #100;
    $finish;
end
endmodule
```

Program 5: Testbench implementation of Exercise 2.

3.3 Testbench results

All test cases executed successfully, with each marked as `PASSED`. This outcome confirms that the test module behaves as intended and that the implemented RTL design meets the expected functionality.

```
# ---------------------------------------------------------------
# TEST 1: INITIAL VALUES CHECK
# ---------------------------------------------------------------
# [OUTPUT] Time = 200 | 7SEG = 16'h25ff, LED = 3'b100
# [EXPECT] Time = 200 | 7SEG = 16'h25ff, LED = 3'b100
# ======> PASSED
# ---------------------------------------------------------------
```

```
# TEST 2: SCROLL MODE CHECK
# ------------------------------------------------------------------
# [OUTPUT] Time = 600 | 7SEG = 16'hf25f, LED = 3'b100
# [EXPECT] Time = 600 | 7SEG = 16'hf25f, LED = 3'b100
# ======> PASSED
# [OUTPUT] Time = 1000 | 7SEG = 16'hff25, LED = 3'b100
# [EXPECT] Time = 1000 | 7SEG = 16'hff25, LED = 3'b100
# ======> PASSED
# [OUTPUT] Time = 1400 | 7SEG = 16'h5ff2, LED = 3'b100
# [EXPECT] Time = 1400 | 7SEG = 16'h5ff2, LED = 3'b100
# ======> PASSED
# [OUTPUT] Time = 1800 | 7SEG = 16'h25ff, LED = 3'b100
# [EXPECT] Time = 1800 | 7SEG = 16'h25ff, LED = 3'b100
# ======> PASSED
# [OUTPUT] Time = 2200 | 7SEG = 16'hf25f, LED = 3'b100
# [EXPECT] Time = 2200 | 7SEG = 16'hf25f, LED = 3'b100
# ======> PASSED
# ------------------------------------------------------------------
# TEST 3: BOUNCE MODE CHECK
# ------------------------------------------------------------------
# ------------------------------------------------------------------
# [SWITCH] Time = 2200 | Switching to effect 2
# ------------------------------------------------------------------
# [OUTPUT] Time = 2210 | 7SEG = 16'hf25f, LED = 3'b010
# [EXPECT] Time = 2210 | 7SEG = 16'hf25f, LED = 3'b010
# ======> PASSED
# [OUTPUT] Time = 2610 | 7SEG = 16'hff25, LED = 3'b010
# [EXPECT] Time = 2610 | 7SEG = 16'hff25, LED = 3'b010
# ======> PASSED
# [OUTPUT] Time = 3010 | 7SEG = 16'hf25f, LED = 3'b010
# [EXPECT] Time = 3010 | 7SEG = 16'hf25f, LED = 3'b010
# ======> PASSED
# [OUTPUT] Time = 3410 | 7SEG = 16'h25ff, LED = 3'b010
# [EXPECT] Time = 3410 | 7SEG = 16'h25ff, LED = 3'b010
# ======> PASSED
# [OUTPUT] Time = 3810 | 7SEG = 16'hf25f, LED = 3'b010
# [EXPECT] Time = 3810 | 7SEG = 16'hf25f, LED = 3'b010
# ======> PASSED
# ------------------------------------------------------------------
# TEST 4: SWITCH BACK TO SCROLL MODE
# ------------------------------------------------------------------
# ------------------------------------------------------------------
# [SWITCH] Time = 3810 | Switching to effect 1
# ------------------------------------------------------------------
# [OUTPUT] Time = 3820 | 7SEG = 16'hf25f, LED = 3'b100
# [EXPECT] Time = 3820 | 7SEG = 16'hf25f, LED = 3'b100
# ======> PASSED
# [OUTPUT] Time = 4220 | 7SEG = 16'hff25, LED = 3'b100
# [EXPECT] Time = 4220 | 7SEG = 16'hff25, LED = 3'b100
# ======> PASSED
# [OUTPUT] Time = 4620 | 7SEG = 16'h5ff2, LED = 3'b100
# [EXPECT] Time = 4620 | 7SEG = 16'h5ff2, LED = 3'b100
# ======> PASSED
```

```
# ----------------------------------------------------------------------
# TEST 5: SWITCH BACK TO BOUNCE MODE
# ----------------------------------------------------------------------
# ----------------------------------------------------------------------
# [SWITCH] Time = 4620 | Switching to effect 2
# ----------------------------------------------------------------------
# [OUTPUT] Time = 4630 | 7SEG = 16'h5ff2, LED = 3'b010
# [EXPECT] Time = 4630 | 7SEG = 16'h5ff2, LED = 3'b010
# ======> PASSED
# [OUTPUT] Time = 5030 | 7SEG = 16'hff25, LED = 3'b010
# [EXPECT] Time = 5030 | 7SEG = 16'hff25, LED = 3'b010
# ======> PASSED
# [OUTPUT] Time = 5430 | 7SEG = 16'hf25f, LED = 3'b010
# [EXPECT] Time = 5430 | 7SEG = 16'hf25f, LED = 3'b010
# ======> PASSED
# [OUTPUT] Time = 5830 | 7SEG = 16'h25ff, LED = 3'b010
# [EXPECT] Time = 5830 | 7SEG = 16'h25ff, LED = 3'b010
# ======> PASSED
# [OUTPUT] Time = 6230 | 7SEG = 16'hf25f, LED = 3'b010
# [EXPECT] Time = 6230 | 7SEG = 16'hf25f, LED = 3'b010
# ======> PASSED
# ----------------------------------------------------------------------
# TEST 6: RESET CHECK
# ----------------------------------------------------------------------
# Reseting...
# [OUTPUT] Time = 6246 | 7SEG = 16'h25ff, LED = 3'b010
# [EXPECT] Time = 6246 | 7SEG = 16'h25ff, LED = 3'b010
# ======> PASSED
# [OUTPUT] Time = 6646 | 7SEG = 16'h25ff, LED = 3'b010
# [EXPECT] Time = 6646 | 7SEG = 16'h25ff, LED = 3'b010
# ======> PASSED
# ----------------------------------------------------------------------
# [SWITCH] Time = 6646 | Switching to effect 1
# ----------------------------------------------------------------------
# [OUTPUT] Time = 6656 | 7SEG = 16'h25ff, LED = 3'b100
# [EXPECT] Time = 6656 | 7SEG = 16'h25ff, LED = 3'b100
# ======> PASSED
# ----------------------------------------------------------------------
# [SWITCH] Time = 6656 | Switching to effect 2
# ----------------------------------------------------------------------
# Releasing reset...
# [OUTPUT] Time = 7066 | 7SEG = 16'h25ff, LED = 3'b010
# [EXPECT] Time = 7066 | 7SEG = 16'h25ff, LED = 3'b010
# ======> PASSED
# [OUTPUT] Time = 7466 | 7SEG = 16'hf25f, LED = 3'b010
# [EXPECT] Time = 7466 | 7SEG = 16'hf25f, LED = 3'b010
# ======> PASSED
# [OUTPUT] Time = 7866 | 7SEG = 16'hff25, LED = 3'b010
# [EXPECT] Time = 7866 | 7SEG = 16'hff25, LED = 3'b010
# ======> PASSED
# [OUTPUT] Time = 8266 | 7SEG = 16'hf25f, LED = 3'b010
# [EXPECT] Time = 8266 | 7SEG = 16'hf25f, LED = 3'b010
# ======> PASSED
```

```
# [OUTPUT] Time = 8666 | 7SEG = 16'h25ff, LED = 3'b010
# [EXPECT] Time = 8666 | 7SEG = 16'h25ff, LED = 3'b010
# ======> PASSED
# [OUTPUT] Time = 9066 | 7SEG = 16'hf25f, LED = 3'b010
# [EXPECT] Time = 9066 | 7SEG = 16'hf25f, LED = 3'b010
# ======> PASSED
# ------------------------------------------------------------------
# SUMMARY
# ------------------------------------------------------------------
# TOTAL TESTS : 28
# Total PASSED: 28
# Total FAILED: 0
# ======> ALL TESTS PASSED
# ** Note: $finish    : ../tb/tb.v(198)
#    Time: 9166 ns  Iteration: 0  Instance: /tb
```

Simulation 2: Testbench results of Exercise 2.

3.4 Simulation waveform

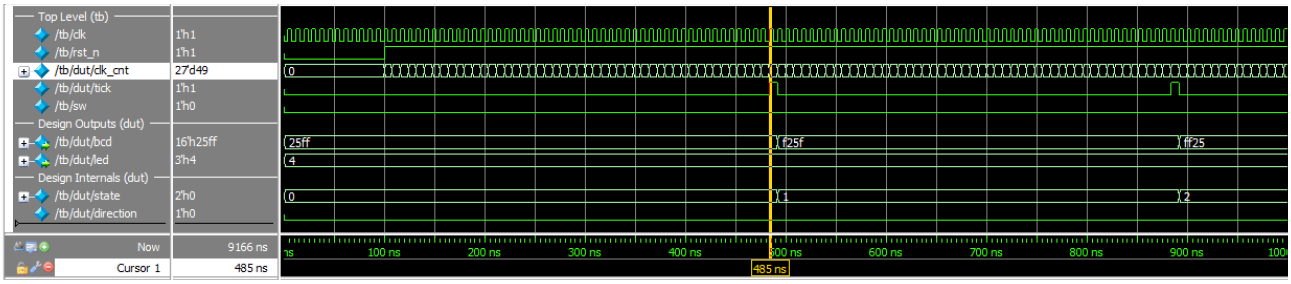

Figure 2.1

Figure 2.1 shows that after the reset is asserted and then deasserted, the module initializes to its initial state. In this state, the LED displays red, and the direction is set to 0, meaning the text string "25" moves from left to right.
In the simulation, one second corresponds to 100 clock cycles. The shifting effect occurs twice per second, so the string shifts every 50 clock cycles. When the `clk_cnt` signal counts from 0 to 49, the `tick` signal is raised high. On the following clock cycle, the string transitions to state 1, where "25" appears in the middle of the four 7-segment LED displays.
At the next tick, the string shifts again to state 2, positioning "25" on the two rightmost 7-segment LED displays.

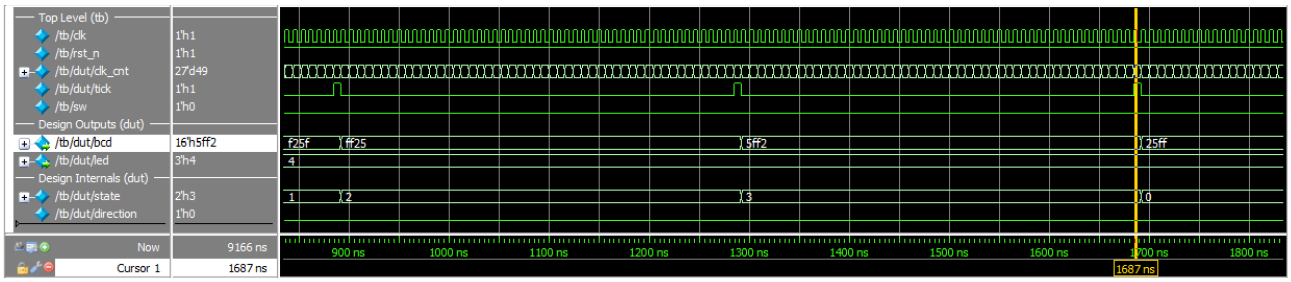

Figure 2.2

Since effect 1 scrolls from left to right, after reaching state 2 ("ff25"), the string shifts to state 3 ("5ff2") one clock cycle after the next tick signal is asserted, producing a smooth scrolling transition.
At the following `tick`, one clock cycle later, the string returns to state 0 ("25ff"), thereby restarting the scrolling sequence from the beginning.

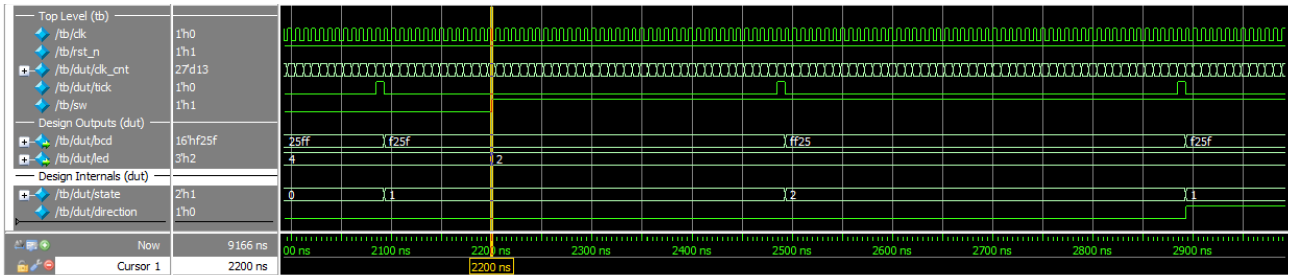

Figure 2.3

When the switch is turned on, indicated by the `sw` signal being asserted, the module activates effect 2, bouncing. In this mode, the LED changes to green, notifying the user that effect 2 is active.
Figure 2.3 shows that after the string reaches the right edge, the direction value changes to 1, meaning the string will now move from right to left. As a result, it bounces back to the middle of the four 7-segment LED displays.

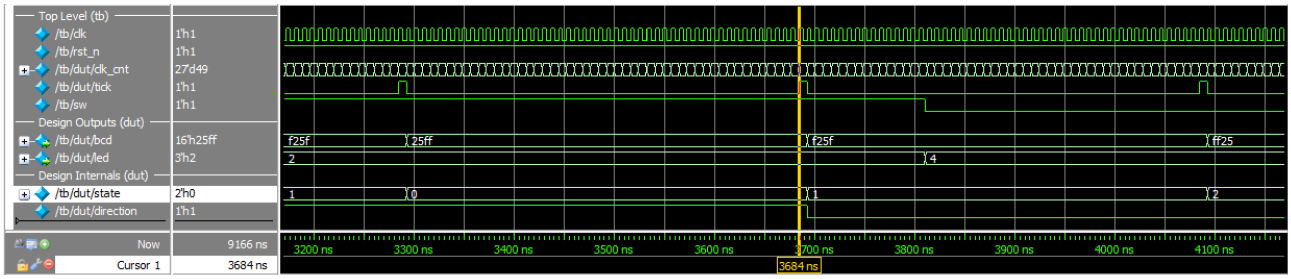

Figure 2.4

In Figure 2.4, the string shifts from right to left, and when it reaches the leftmost position, it bounces back to the right. At this point, the direction signal changes from 1 to 0 immediately after the `tick` signal is asserted at 3648 ns.

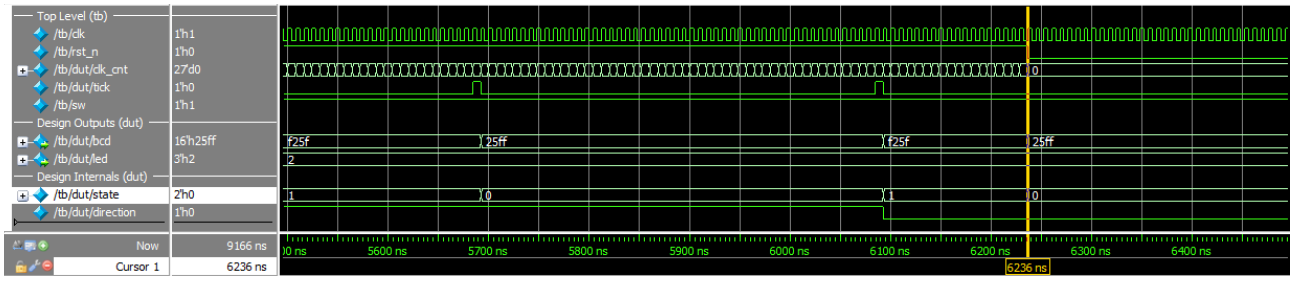

Figure 2.5

Skipping a few tests, Figure 2.5 demonstrates how the reset signal affects the module. When reset is asserted, the string immediately stops shifting and returns to the initial state (state 0). Importantly, the current effect remains active, and the user configuration is not reset.
Additional behavior can be observed more clearly through the test bench module, which provides easier tracking of the test results.

3.5 FPGA results

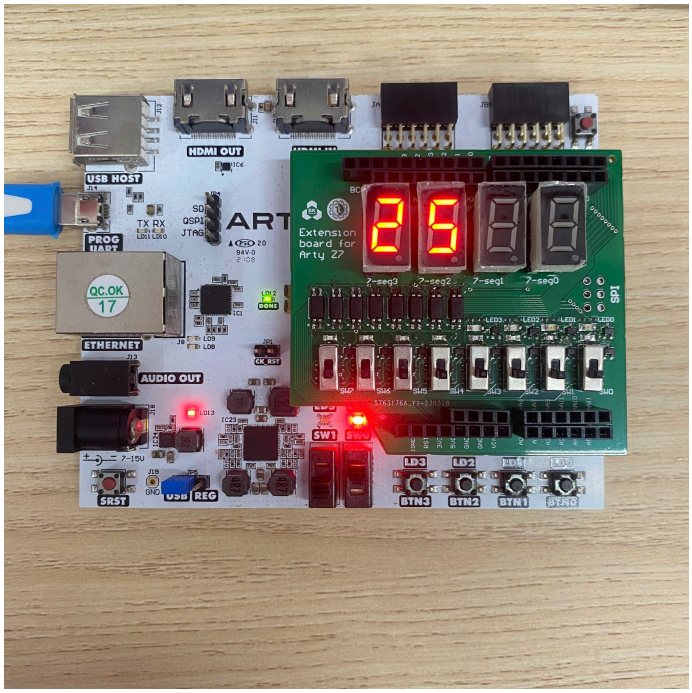

Figure 2.6

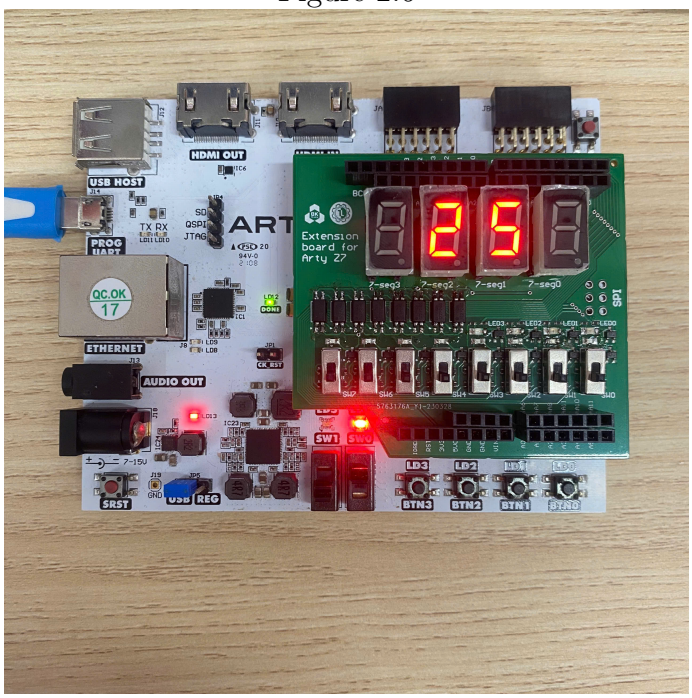

Figure 2.7

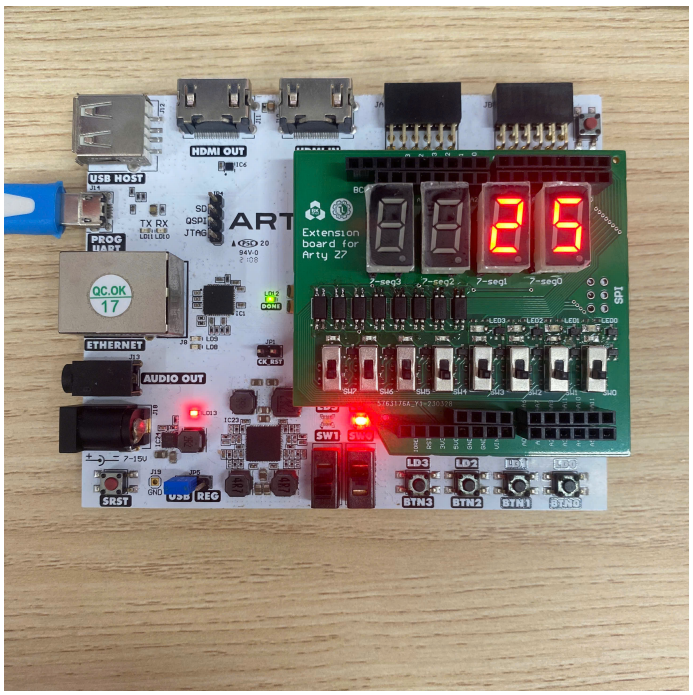

Figure 2.8

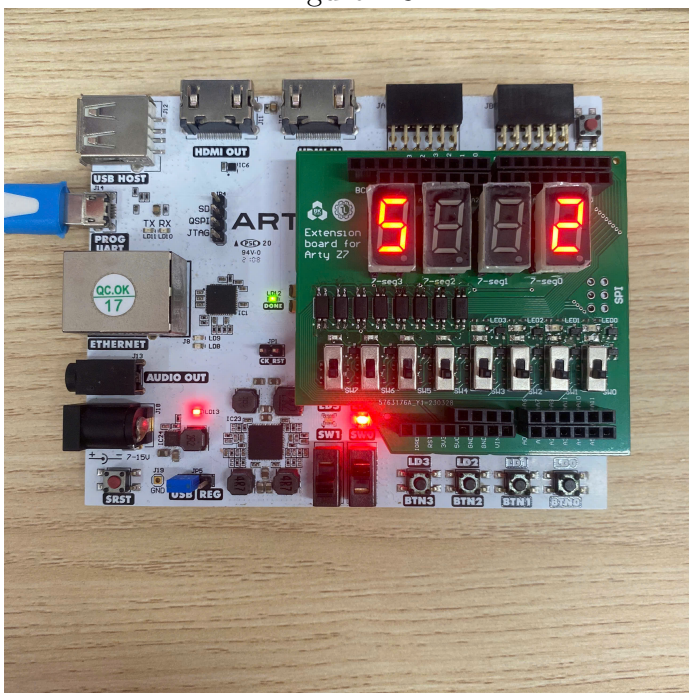

Figure 2.9

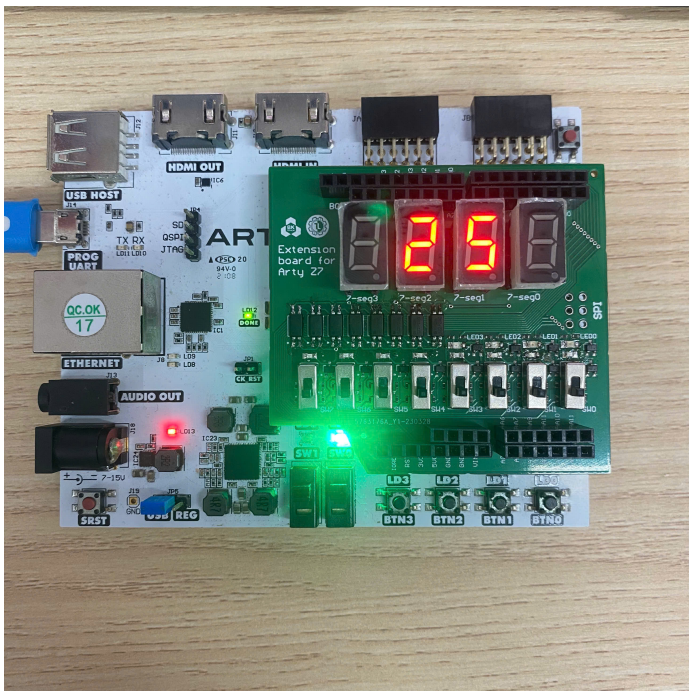

Figure 2.10

Figures 2.6 to 2.9 illustrate how the string “25” scrolls across the four 7-segment LED displays, moving from the leftmost to the rightmost position. During this process, the red LED remains on, indicating to the user that effect 1 is active.
Figure 2.10 shows the FPGA after the effect switch is turned on. At this point, the module transitions to effect 2, and the LED changes to green, signaling the mode switch.

4 Exercise 3

4.1 RTL implementation

This module used a FSM to handle a state machine that changes the display mode of a bit string. Initially, LEDs show the default 4-bit random string, which is performed by a reset signal, in this module we particularly set the default bit string to be 4'b0011. Buttons are used to change the display mode as follow:

- Button 0: Mode reset: Show the default 4-bit string on LEDs.
- Button 1: Mode circular shift left ring: Shift 4-bit string to left in a ring every 1 second.
- Button 2: Mode circular shift right ring: Shift 4-bit string to right in a ring every 1 second.
- Button 3: Pause: Pause the current shifting string.

```
module ex3 (
    input wire clk,
    input wire rst_n,
    input wire [3:0] btn,
    output reg [3:0] led
);

`ifdef SIMULATION
    //for sim, a sec is 100 cycles
    parameter CLK_FREQ = 27'd100;
`else
    //for fpga, a sec is 125m cycles
    parameter CLK_FREQ = 27'd125_000_000;
`endif

parameter PATTERN = 4'b0011;
parameter MODE0 = 2'd0;
parameter MODE1 = 2'd1;
parameter MODE2 = 2'd2;
parameter MODE3 = 2'd3;

reg [1:0] mode;
reg [26:0] clk_cnt;
wire tick_1s;
assign tick_1s = (clk_cnt == CLK_FREQ - 1);

always @(posedge clk or negedge rst_n) begin
    if (!rst_n)
        clk_cnt <= 27'd0;
    else if (tick_1s)
        clk_cnt <= 27'd0;
    else
        clk_cnt <= clk_cnt + 1;
end

always @(posedge clk or negedge rst_n) begin
```

```
37      if (!rst_n)
38          mode <= MODE0;
39      else if (btn[0])
40          mode <= MODE0;
41      else if (btn[1])
42          mode <= MODE1;
43      else if (btn[2])
44          mode <= MODE2;
45      else if (btn[3])
46          mode <= MODE3;
47      else
48          mode <= mode;
49  end
50
51  always @(posedge clk or negedge rst_n) begin
52      if (!rst_n)
53          led <= PATTERN;
54      else if (tick_1s) begin
55          case (mode)
56              MODE1: led <= {led[2:0], led[3]}; //shift left
57              MODE2: led <= {led[0], led[3:1]}; //shift right
58              MODE3: led <= led;                //pause
59              default: led <= PATTERN;          //reset pattern
60          endcase
61      end else if (btn[0])
62          led <= PATTERN;
63      else
64          led <= led;
65  end
66  endmodule
```

Program 6: RTL implementation of Exercise 3.

4.2 Testbench

The testbench executes a sequence of functional tests for Exercise 3, including:

- Test 1 (Initial values check): After applying reset, the testbench verifies that the LED outputs initialized to the expected default pattern, which is 4'b0011.
- Test 2 (Shift left test): The test simulates pressing button 1 to shift left and confirms that subsequent clock ticks continuously shift the pattern left, with wrapping, across several cycles.
- Test 3 (Shift right test): By pressing button 2 to shift right, the testbench checks that the LED pattern shifts right as intended, with the correct bit wrapping behavior, for multiple cycles.
- Test 4 (Pause mode test): Pressing button 3 tests the pause functionality, confirming that the LED outputs remain unchanged even after additional clock ticks.
- Test 5 (Reset check through button 0): The testbench validates the effect of button 0 press, ensuring the LED outputs restore to the default pattern, mode and pattern updates are blocked during reset.

- Test 6 (Reset check through system reset): The same effect with button 0 press. However, during the global reset activation, no button press can be operated, the system remains with the default pattern bit string displayed on the LEDs. After the reset is released, the system must behave normally as expected.

```
module tb;
reg clk;
reg rst_n;
reg [3:0] btn;
wire [3:0] led;

parameter CLK_PERIOD = 8; // 8ns = 125MHz
parameter DEBOUNCE_WAIT = CLK_PERIOD * 5;
parameter TICK_WAIT = CLK_PERIOD * 100;
parameter PATTERN = 4'b0011;

integer pass, fail;

ex3 dut (
    .clk   (clk),
    .rst_n (rst_n),
    .btn   (btn),
    .led   (led)
);

task divi;
    $display ("
       ------------------------------------------------------------
    --------------");
endtask

task msg (input [700:0] txt);
    begin
        divi();
        $display ("%0s", txt);
        divi();
    end
endtask

task check (input [3:0] exp_led);
    begin
        $display ("[OUTPUT] Time = %0t | LED = 4'b%4b", $time, led);
        $display ("[EXPECT] Time = %0t | LED = 4'b%4b", $time, exp_led
           );

        if (led === exp_led) begin
            $display ("======> PASSED");
            pass = pass + 1;
        end else begin
            $display ("======> FAILED");
            fail = fail + 1;
        end
    end
```

```
47 endtask
48
49 task press (input [3:0] btn_press);
50     begin
51         case (btn_press)
52             4'b0001: $display ("Button 0 is pressed");
53             4'b0010: $display ("Button 1 is pressed");
54             4'b0100: $display ("Button 2 is pressed");
55             4'b1000: $display ("Button 3 is pressed");
56             default: $display ("More than one button is pressed");
57         endcase
58
59         btn = btn_press;
60         #(DEBOUNCE_WAIT);
61         btn = 4'b0000;
62         #(DEBOUNCE_WAIT);
63     end
64 endtask
65
66 initial begin
67     clk = 0;
68     forever #(CLK_PERIOD/2) clk = ~clk;
69 end
70
71 initial begin
72     rst_n = 0;
73     btn = 4'b0000;
74     pass = 0;
75     fail = 0;
76     #100;
77     rst_n = 1;
78     #100;
79
80     msg ("TEST 1: INITIAL VALUE CHECK");
81     check (PATTERN);
82
83     msg ("TEST 2: BUTTON 1 PRESS (SHIFT LEFT) CHECK");
84     press (4'b0010);
85
86     #(TICK_WAIT);
87     check (4'b0110);
88
89     #(TICK_WAIT);
90     check (4'b1100);
91
92     #(TICK_WAIT);
93     check (4'b1001);
94
95     #(TICK_WAIT);
96     check (4'b0011);
97
98     msg ("TEST 3: BUTTON 2 PRESS (SHIFT RIGHT) CHECK");
99     press (4'b0100);
```

```
100
101     #(TICK_WAIT);
102     check (4'b1001);
103
104     #(TICK_WAIT);
105     check (4'b1100);
106
107     #(TICK_WAIT);
108     check (4'b0110);
109
110     #(TICK_WAIT);
111     check (4'b0011);
112     msg ("TEST 4: BUTTON 3 PRESS (PAUSE) CHECK");
113     #(TICK_WAIT);
114     check (4'b1001);
115
116     press (4'b1000);
117
118     #(TICK_WAIT);
119     check (4'b1001);
120
121     #(TICK_WAIT);
122     check (4'b1001);
123
124     msg ("TEST 5: BUTTON 0 PRESS (RESET) CHECK");
125     press (4'b0001);
126
127     #(TICK_WAIT);
128     check (PATTERN);
129
130     #(TICK_WAIT);
131     check (PATTERN);
132
133     #(TICK_WAIT);
134     check (PATTERN);
135
136     press (4'b0100);
137
138     #(TICK_WAIT);
139     check (4'b1001);
140
141     #(TICK_WAIT);
142     check (4'b1100);
143
144     msg ("TEST 6: RESET CHECK");
145     $display ("Reseting...");
146     rst_n = 0;
147
148     #(TICK_WAIT);
149     check (PATTERN);
150
151     press (4'b0010);
152     #(TICK_WAIT);
```

```
153     check (PATTERN);
154
155     #(TICK_WAIT);
156     check (PATTERN);
157
158     press (4'b0100);
159     #(TICK_WAIT);
160     check (PATTERN);
161
162     #(TICK_WAIT);
163     check (PATTERN);
164
165     #(CLK_PERIOD * 5);
166     $display ("Releasing reset...");
167     rst_n = 1;
168     #10;
169
170     check (PATTERN);
171
172     press (4'b0010);
173
174     #(TICK_WAIT);
175     check (4'b0110);
176
177     #(TICK_WAIT);
178     check (4'b1100);
179
180     msg ("SUMMARY");
181     $display ("TOTAL TESTS : %0d", pass + fail);
182     $display ("Total PASSED: %0d", pass);
183     $display ("Total FAILED: %0d", fail);
184
185     if (fail == 0)
186         $display ("======> ALL TESTS PASSED");
187     else if (pass == 0)
188         $display ("======> ALL TESTS FAILED");
189     else
190         $display ("SOME TESTS FAILED");
191
192     #100;
193     $finish;
194 end
195 endmodule
```

Program 7: Testbench implementation of Exercise 3.

4.3 Testbench results

All test cases executed successfully, with each marked as `PASSED`. This outcome confirms that the test module behaves as intended and that the implemented RTL design meets the expected functionality.

```
# ----------------------------------------------------------------------
# TEST 1: INITIAL VALUE CHECK
# ----------------------------------------------------------------------
# [OUTPUT] Time = 200 | LED = 4'b0011
# [EXPECT] Time = 200 | LED = 4'b0011
# ======> PASSED
# ----------------------------------------------------------------------
# TEST 2: BUTTON 1 PRESS (SHIFT LEFT) CHECK
# ----------------------------------------------------------------------
# Button 1 is pressed
# [OUTPUT] Time = 1080 | LED = 4'b0110
# [EXPECT] Time = 1080 | LED = 4'b0110
# ======> PASSED
# [OUTPUT] Time = 1880 | LED = 4'b1100
# [EXPECT] Time = 1880 | LED = 4'b1100
# ======> PASSED
# [OUTPUT] Time = 2680 | LED = 4'b1001
# [EXPECT] Time = 2680 | LED = 4'b1001
# ======> PASSED
# [OUTPUT] Time = 3480 | LED = 4'b0011
# [EXPECT] Time = 3480 | LED = 4'b0011
# ======> PASSED
# ----------------------------------------------------------------------
# TEST 3: BUTTON 2 PRESS (SHIFT RIGHT) CHECK
# ----------------------------------------------------------------------
# Button 2 is pressed
# [OUTPUT] Time = 4360 | LED = 4'b1001
# [EXPECT] Time = 4360 | LED = 4'b1001
# ======> PASSED
# [OUTPUT] Time = 5160 | LED = 4'b1100
# [EXPECT] Time = 5160 | LED = 4'b1100
# ======> PASSED
# [OUTPUT] Time = 5960 | LED = 4'b0110
# [EXPECT] Time = 5960 | LED = 4'b0110
# ======> PASSED
# [OUTPUT] Time = 6760 | LED = 4'b0011
# [EXPECT] Time = 6760 | LED = 4'b0011
# ======> PASSED
# ----------------------------------------------------------------------
# TEST 4: BUTTON 3 PRESS (PAUSE) CHECK
# ----------------------------------------------------------------------
# [OUTPUT] Time = 7560 | LED = 4'b1001
# [EXPECT] Time = 7560 | LED = 4'b1001
# ======> PASSED
# Button 3 is pressed
# [OUTPUT] Time = 8440 | LED = 4'b1001
# [EXPECT] Time = 8440 | LED = 4'b1001
# ======> PASSED
```

```
# [OUTPUT] Time = 9240 | LED = 4'b1001
# [EXPECT] Time = 9240 | LED = 4'b1001
# ======> PASSED
# ------------------------------------------------------------------------
# TEST 5: BUTTON 0 PRESS (RESET) CHECK
# ------------------------------------------------------------------------
# Button 0 is pressed
# [OUTPUT] Time = 10120 | LED = 4'b0011
# [EXPECT] Time = 10120 | LED = 4'b0011
# ======> PASSED
# [OUTPUT] Time = 10920 | LED = 4'b0011
# [EXPECT] Time = 10920 | LED = 4'b0011
# ======> PASSED
# [OUTPUT] Time = 11720 | LED = 4'b0011
# [EXPECT] Time = 11720 | LED = 4'b0011
# ======> PASSED
# Button 2 is pressed
# [OUTPUT] Time = 12600 | LED = 4'b1001
# [EXPECT] Time = 12600 | LED = 4'b1001
# ======> PASSED
# [OUTPUT] Time = 13400 | LED = 4'b1100
# [EXPECT] Time = 13400 | LED = 4'b1100
# ======> PASSED
# ------------------------------------------------------------------------
# TEST 6: RESET CHECK
# ------------------------------------------------------------------------
# Reseting...
# [OUTPUT] Time = 14200 | LED = 4'b0011
# [EXPECT] Time = 14200 | LED = 4'b0011
# ======> PASSED
# Button 1 is pressed
# [OUTPUT] Time = 15080 | LED = 4'b0011
# [EXPECT] Time = 15080 | LED = 4'b0011
# ======> PASSED
# [OUTPUT] Time = 15880 | LED = 4'b0011
# [EXPECT] Time = 15880 | LED = 4'b0011
# ======> PASSED
# Button 2 is pressed
# [OUTPUT] Time = 16760 | LED = 4'b0011
# [EXPECT] Time = 16760 | LED = 4'b0011
# ======> PASSED
# [OUTPUT] Time = 17560 | LED = 4'b0011
# [EXPECT] Time = 17560 | LED = 4'b0011
# ======> PASSED
# Releasing reset...
# [OUTPUT] Time = 17610 | LED = 4'b0011
# [EXPECT] Time = 17610 | LED = 4'b0011
# ======> PASSED
# Button 1 is pressed
# [OUTPUT] Time = 18490 | LED = 4'b0110
# [EXPECT] Time = 18490 | LED = 4'b0110
# ======> PASSED
# [OUTPUT] Time = 19290 | LED = 4'b1100
```

```
# [EXPECT] Time = 19290 | LED = 4'b1100
# ======> PASSED
# ------------------------------------------------------------------------
# SUMMARY
# ------------------------------------------------------------------------
# TOTAL TESTS : 25
# Total PASSED: 25
# Total FAILED: 0
# ======> ALL TESTS PASSED
# ** Note: $finish    : ../tb/tb.v(192)
#    Time: 19390 ns  Iteration: 0  Instance: /tb
```

Simulation 3: Testbench results of Exercise 3.

4.4 Simulation waveform

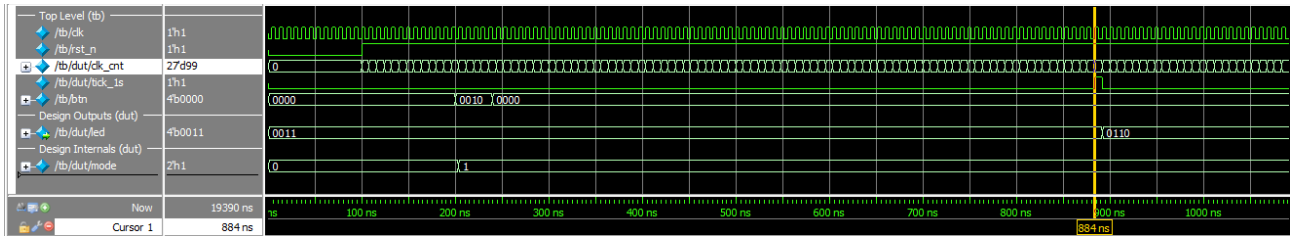

Figure 3.1

In this simulation, one second corresponds to 100 clock cycles.
Figure 3.1 shows that after the reset is asserted and then deasserted, the LEDs display the default string 4'b0011. When Button 1 (shift left) is pressed, the system waits 100 clock cycles until the `tick_1s` signal is asserted, indicating that one second has elapsed.
On the following clock cycle, the string shifts to the left, resulting in 4'b0110, as reflected by the `led` signal.

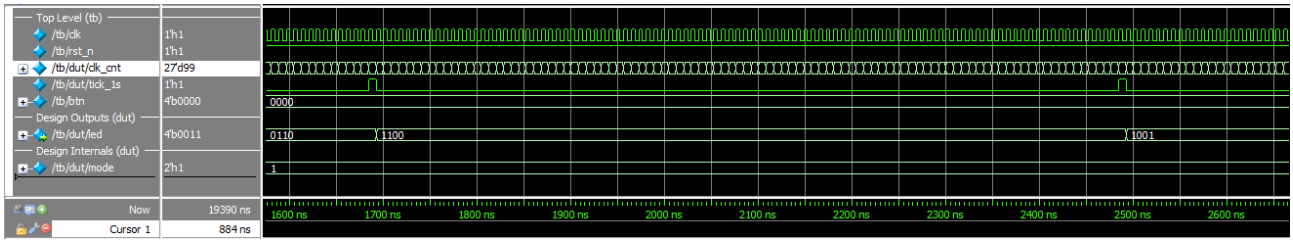

Figure 3.2

Figure 3.2 illustrates how the string continues to shift left at 1-second intervals, triggered by the assertion of the `tick_1s` signal.

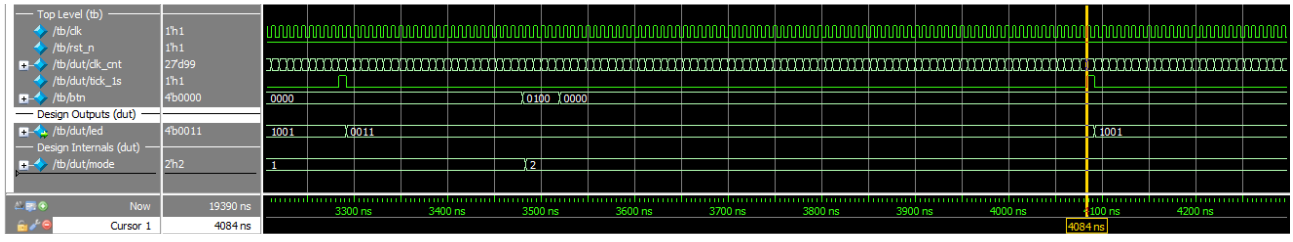

Figure 3.3

Figure 3.3 shows that the string initially holds the value 4'b0011. After pressing Button 2 (shift right), the string shifts to the right. One clock cycle after the `tick_1s` signal is asserted, the result becomes 4'b1001.

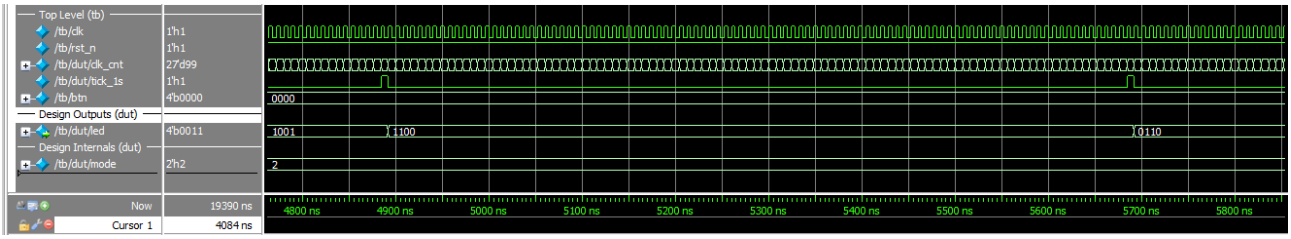

Figure 3.4

Figure 3.4 illustrates how the string continues to shift to the right at 1-second intervals, triggered by the assertion of the `tick_1s` signal.

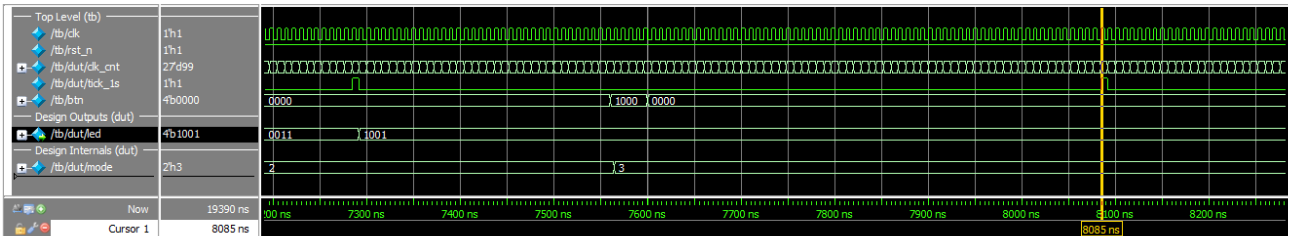

Figure 3.5

Figure 3.5 demonstrates that when Button 3 (pause) is pressed, the string remains unchanged. Even after the next second, indicated by the assertion of the `tick_1s` signal, the value stays at 4'b1001. This confirms that the pause function operates correctly, preventing any further shifts while active.

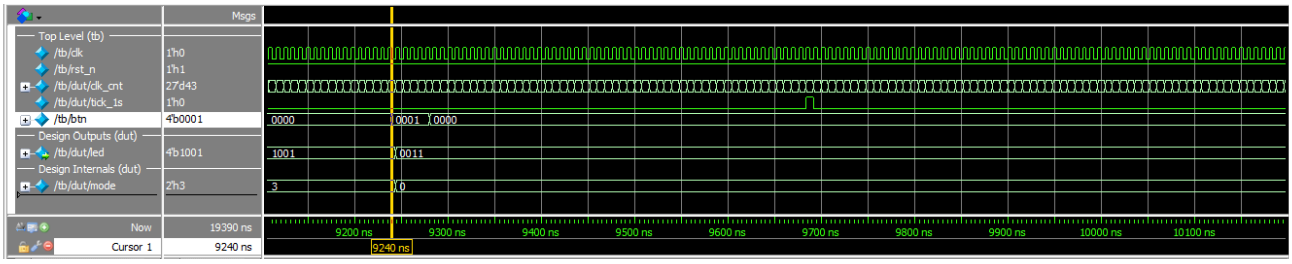

Figure 3.6

Button 0 is used to reset the string, and it is designed to take effect immediately upon being pressed. For example, when the string holds the value 4'b1001, pressing Button 0 resets it instantly to the default value 4'b0011.

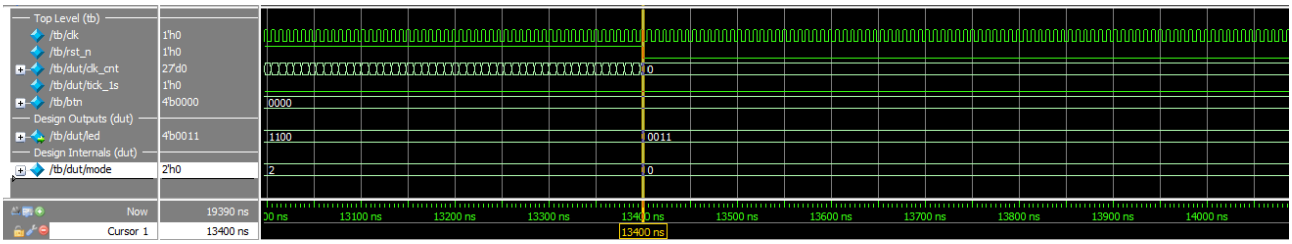

Figure 3.7

Figure 3.7 illustrates the behavior of the reset mechanism. Since the design uses an asynchronous active-low reset, asserting the reset immediately forces the string value 4'b1100 back to its default value 4'b0011 at 13400 ns.

4.5 FPGA results

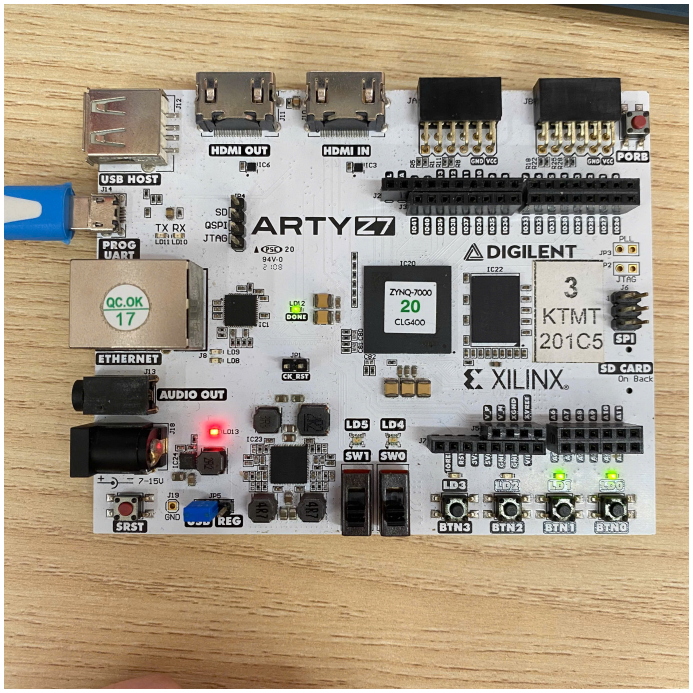

Figure 3.8

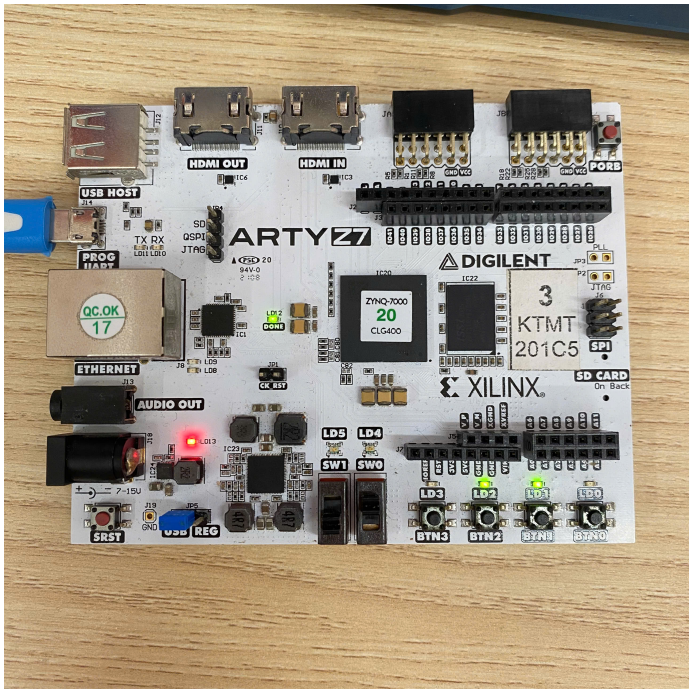

Figure 3.9

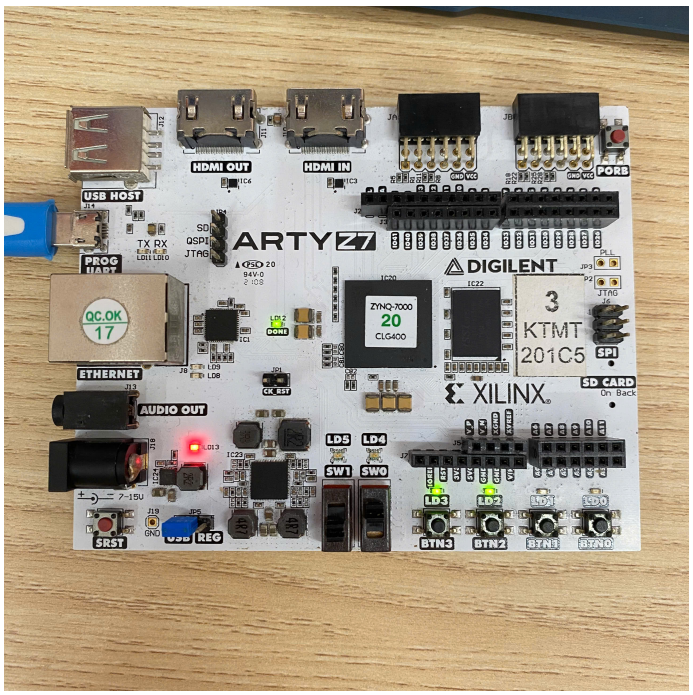

Figure 3.10

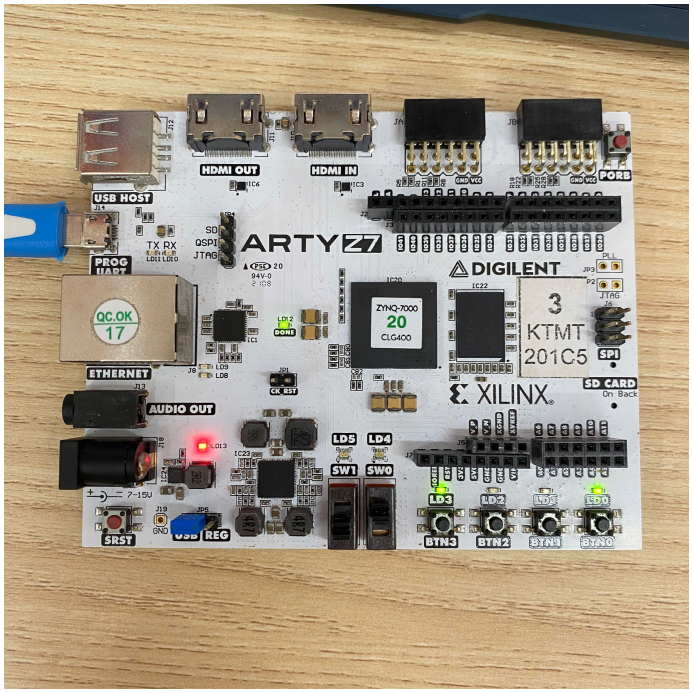

Figure 3.11

This section presents part of the FPGA demonstration. Figures 3.8 to 3.11 illustrate how the string shifts to the left when Button 1 is pressed. The sequence progresses from the initial value 4'b0011 to 4'b0110, then to 4'b1100, and finally to 4'b1001.

Figure 3.12

Figure 3.12 shows the transmition of the current LED string to the PC via UART. This functionality was successfully implemented and verified by observing the UART output in the terminal window. The terminal logs show a sequence of operations including reset, left shift, right shift, and pause, with each action followed by an updated LED value. For example, after issuing a SHIFT LEFT command, the LED string transitions from 0011 to 0110, then to 1100, and so on. Similarly, SHIFT RIGHT operations correctly reverse the pattern, and PAUSE maintains the current value across ticks. These outputs confirm that the system reliably communicates the LED state to the PC in real time using UART.

5 Divider

5.1 RTL implementation

These modules are built to perform unsigned division and remainder operations based on the provided algorithm in Lab 1 documentation. The two module implemented with combinational logic only and doesn't handle the case of a divisor of zero. In this implementation, no `/` or `%` in the Verilog code.
The `divu_1iter` module is intended to perform a single iteration of unsiged integer division, updating remainder, quotient and divided registers based on its inputs.

```
module divu_1iter (
    input wire [31:0] dividend_in,
    input wire [31:0] quotient_in,
    input wire [31:0] remainder_in,
    input wire [31:0] divisor,

    output wire [31:0] dividend_out,
    output wire [31:0] quotient_out,
    output wire [31:0] remainder_out
);

wire [31:0] new_remainder, subtract;
wire can_subtract;
assign new_remainder = {remainder_in[30:0], dividend_in[31]};
assign subtract = new_remainder - divisor;
assign can_subtract = new_remainder >= divisor;

assign remainder_out = can_subtract ? subtract : new_remainder;
assign quotient_out = {quotient_in[30:0], can_subtract};
assign dividend_out = dividend_in << 1;
endmodule
```

Program 8: RTL implementation of the divu_1iter module.

The `divider_unsigned` module computes the quotient and remainder for unsigned integer division operations, a fundamental operation in many digital systems.

```
module divider_unsigned (
    input wire [31:0] dividend,
    input wire [31:0] divisor,
    output wire [31:0] quotient,
    output wire [31:0] remainder
);

wire [31:0] dividend_arr [0:32];
wire [31:0] quotient_arr [0:32];
wire [31:0] remainder_arr [0:32];

assign dividend_arr[0] = dividend;
assign quotient_arr[0] = 32'b0;
assign remainder_arr[0] = 32'b0;

genvar i;
```

```
generate
    for (i = 0; i < 32; i = i + 1) begin : loop
        divu_1iter div_iter (
            .dividend_in   (dividend_arr[i]),
            .quotient_in   (quotient_arr[i]),
            .remainder_in  (remainder_arr[i]),
            .divisor       (divisor),
            .dividend_out  (dividend_arr[i+1]),
            .quotient_out  (quotient_arr[i+1]),
            .remainder_out (remainder_arr[i+1])
        );
    end
endgenerate

assign quotient = quotient_arr[32];
assign remainder = remainder_arr[32];
endmodule
```

Program 9: RTL implementation of the divider_unsigned module.

5.2 Testbench

This testbench is used to check the arithmetic and logical accuracy of the `divu_1iter` design over a diverse set of test cases, simulating a variety of operand values and initial conditions. It ensures that the module correctly implements the intended division iteration logic, including handling edge cases, large operands and multiple input combinations.

```
module tb_divu_1iter;
reg [31:0] remainder_in;
reg [31:0] quotient_in;
reg [31:0] dividend_in;
reg [31:0] divisor;

wire [31:0] remainder_out;
wire [31:0] quotient_out;
wire [31:0] dividend_out;

integer pass, fail;

divu_1iter u_dut (
    .remainder_in  (remainder_in),
    .quotient_in   (quotient_in),
    .dividend_in   (dividend_in),
    .divisor       (divisor),

    .remainder_out (remainder_out),
    .quotient_out  (quotient_out),
    .dividend_out  (dividend_out)
);

task divi;
    $display ("
       ------------------------------------------------------------
```

```
26      --------------");
27  endtask
28
29  task check (input [31:0] exp_rem, input [31:0] exp_quo, input [31:0]
        exp_div);
30      begin
31          $display ("[OUTPUT] Time = %0t | remainder_out = %0d,
                quotient_out = %0d, dividend_out = %h", $time,
                remainder_out, quotient_out, dividend_out);
32          $display ("[EXPECT] Time = %0t | remainder_out = %0d,
                quotient_out = %0d, dividend_out = %h", $time, exp_rem,
                exp_quo, exp_div);
33
34          if ((remainder_out === exp_rem) && (quotient_out === exp_quo)
                && (dividend_out === exp_div)) begin
35              $display ("======> PASSED");
36              pass = pass + 1;
37          end else begin
38              $display ("======> FAILED");
39              fail = fail + 1;
40          end
41      end
42  endtask
43
44  task test (input [31:0] div_in, input [31:0] dvsr, input [31:0] rem_in
        , input [31:0] quo_in);
45      begin
46          divi();
47          $display ("[TEST]   Time = %0t | dividend_in = %0h, divisor =
                %0d, remainder_in = %0d, quotient_in = %0d", $time, div_in,
                 dvsr, rem_in, quo_in);
48          divi();
49
50          dividend_in  = div_in;
51          divisor      = dvsr;
52          remainder_in = rem_in;
53          quotient_in  = quo_in;
54          #10;
55      end
56  endtask
57
58  initial begin
59      pass = 0;
60      fail = 0;
61      #10;
62
63      test (32'h8000_0000, 32'd2, 32'd0, 32'd8);
64      check (32'd1, 32'd16, 32'd0);
65
66      test (32'h8000_0000, 32'd1, 32'd0, 32'd8);
67      check (32'd0, 32'd17, 32'd0);
68
69      test (32'h0000_0004, 32'd3, 32'd2, 32'd1);
```

```
70      check (32'd1, 32'd3, 32'd8);
71
72      test (32'h4000_0000, 32'd5, 32'd3, 32'd0);
73      check (32'd1, 32'd1, 32'h8000_0000);
74
75      test (32'h0000_0000, 32'd10, 32'd4, 32'd1);
76      check (32'd8, 32'd2, 32'h0000_0000);
77
78      test (32'ha000_0000, 32'd3, 32'd2, 32'd5);
79      check(32'd2, 32'd11, 32'h4000_0000);
80
81      test (32'h8000_0000, 32'd6, 32'd2, 32'd0);
82      check(32'd5, 32'd0, 32'h0000_0000);
83
84      test (32'h8000_0000, 32'd5, 32'd2, 32'd0);
85      check (32'd0, 32'd1, 32'h0000_0000);
86
87      test (32'hFFFF_FFFF, 32'hFFFF_FFFF, 32'hFFFF_FFFF, 32'hFFFF_FFFF);
88      check (32'd0, 32'hFFFF_FFFF, 32'hFFFF_FFFE);
89
90      test (32'h1234_5678, 32'd100, 32'd200, 32'd42);
91      check (32'd300, 32'd85, 32'h2468_ACF0);
92
93      test (32'h1234_5678, 32'd500, 32'd200, 32'd42);
94      check (32'd400, 32'd84, 32'h2468_ACF0);
95
96      test (32'h0000_0000, 32'd1, 32'd0, 32'd123);
97      check(32'd0, 32'd246, 32'h0000_0000);
98
99      test (32'h8000_0000, 32'hFFFF_FFFF, 32'd0, 32'd0);
100     check(32'd1, 32'd0, 32'h0000_0000);
101
102     divi();
103     $display ("SUMMARY");
104     divi();
105     $display ("TOTAL TESTS : %0d", pass + fail);
106     $display ("Total PASSED: %0d", pass);
107     $display ("Total FAILED: %0d", fail);
108
109     if (fail == 0)
110         $display ("======> ALL TESTS PASSED");
111     else
112         $display ("======> SOME TESTS FAILED");
113
114     #100;
115     $finish;
116 end
117 endmodule
```

Program 10: Testbench implementation of divu_1iter module.

This testbench applies a wide range of input combinations to the divider_unsigned module and checks the resulting quotient and remainder against the mathematically expected results. It ensures that the divider correctly process both typical and edge-cases inputs, guaranteeing reliability for synthesis or further hardware integration.

```
module tb_divider_unsigned;
reg  [31:0] dividend;
reg  [31:0] divisor;
wire [31:0] quotient;
wire [31:0] remainder;

integer pass, fail;

divider_unsigned u_dut (
    .dividend  (dividend),
    .divisor   (divisor),
    .quotient  (quotient),
    .remainder (remainder)
);

task divi;
    $display ("
       ------------------------------------------------------------
    --------------");
endtask

task msg (input [700:0] txt);
    begin
        divi();
        $display ("%0s", txt);
        divi();
    end
endtask

task check (input [31:0] exp_quo, input [31:0] exp_rem);
    begin
        $display ("[OUTPUT] Time = %0t | quotient = %0d, remainder =
           %0d", $time, quotient, remainder);
        $display ("[EXPECT] Time = %0t | quotient = %0d, remainder =
           %0d", $time, exp_quo, exp_rem);

        if ((quotient === exp_quo) && (remainder === exp_rem)) begin
            $display ("======> PASSED");
            pass = pass + 1;
        end else begin
            $display ("======> FAILED");
            fail = fail + 1;
        end
    end
endtask

task test (input [31:0] div_in, input [31:0] dvsr);
    begin
        divi();
```

```
47          $display ("[TEST]    Time = %0t | dividend = %0d, divisor = %0d
                ", $time, div_in, dvsr);
48          divi();
49
50          dividend = div_in;
51          divisor  = dvsr;
52          #10;
53      end
54  endtask
55
56  initial begin
57      pass = 0;
58      fail = 0;
59      #10;
60
61      test (32'd4, 32'd2);
62      check (32'd2, 32'd0);
63
64      test (32'd4, 32'd4);
65      check (32'd1, 32'd0);
66
67      test (32'd10, 32'd4);
68      check (32'd2, 32'd2);
69
70      test (32'd2, 32'd4);
71      check(32'd0, 32'd2);
72
73      test (32'd100, 32'd1);
74      check (32'd100, 32'd0);
75
76      test (32'd100, 32'd7);
77      check (32'd14, 32'd2);
78
79      test (32'd0, 32'd50);
80      check (32'd0, 32'd0);
81
82      test (32'hFFFF_FFFF, 32'd1);
83      check (32'hFFFF_FFFF, 32'd0);
84
85      test (32'hFFFF_FFFF, 32'hFFFF_FFFF);
86      check (32'd1, 32'd0);
87
88      test (32'hFFFF_FFFF, 32'd2);
89      check (32'd2147483647, 32'd1);
90
91      test (32'h8000_0000, 32'd2);
92      check (32'd1073741824, 32'd0);
93
94      test (32'd123456789, 32'd987);
95      check (32'd125082, 32'd855);
96
97      test (32'd400000, 32'd399999);
98      check (32'd1, 32'd1);
```

```
99
100     test (32'd1024, 32'd1025);
101     check (32'd0, 32'd1024);
102
103     msg ("SUMMARY");
104     $display ("TOTAL TESTS : %0d", pass + fail);
105     $display ("Total PASSED: %0d", pass);
106     $display ("Total FAILED: %0d", fail);
107
108     if (fail == 0)
109         $display ("======> ALL TESTS PASSED");
110     else
111         $display ("======> SOME TESTS FAILED");
112
113     #100;
114     $finish;
115 end
116 endmodule
```

Program 11: Testbench implementation of divider_unsigned module.

5.3 Testbench results

All test cases executed successfully for both modules, with each marked as `PASSED`. This outcome confirms that the test modules behave as intended and that the implemented RTL designs meets the expected functionality.

```
1  # ----------------------------------------------------------------
2  # [TEST]   Time = 10 | dividend_in = 80000000, divisor = 2,
       remainder_in = 0, quotient_in = 8
3  # ----------------------------------------------------------------
4  # [OUTPUT] Time = 20 | remainder_out = 1, quotient_out = 16,
       dividend_out = 00000000
5  # [EXPECT] Time = 20 | remainder_out = 1, quotient_out = 16,
       dividend_out = 00000000
6  # ======> PASSED
7  # ----------------------------------------------------------------
8  # [TEST]   Time = 20 | dividend_in = 80000000, divisor = 1,
       remainder_in = 0, quotient_in = 8
9  # ----------------------------------------------------------------
10 # [OUTPUT] Time = 30 | remainder_out = 0, quotient_out = 17,
       dividend_out = 00000000
11 # [EXPECT] Time = 30 | remainder_out = 0, quotient_out = 17,
       dividend_out = 00000000
12 # ======> PASSED
13 # ----------------------------------------------------------------
14 # [TEST]   Time = 30 | dividend_in = 4, divisor = 3, remainder_in = 2,
        quotient_in = 1
15 # ----------------------------------------------------------------
16 # [OUTPUT] Time = 40 | remainder_out = 1, quotient_out = 3,
       dividend_out = 00000008
17 # [EXPECT] Time = 40 | remainder_out = 1, quotient_out = 3,
       dividend_out = 00000008
```

```
18 # ======> PASSED
19 # ----------------------------------------------------------------------
20 # [TEST]   Time = 40 | dividend_in = 40000000, divisor = 5,
    remainder_in = 3, quotient_in = 0
21 # ----------------------------------------------------------------------
22 # [OUTPUT] Time = 50 | remainder_out = 1, quotient_out = 1,
    dividend_out = 80000000
23 # [EXPECT] Time = 50 | remainder_out = 1, quotient_out = 1,
    dividend_out = 80000000
24 # ======> PASSED
25 # ----------------------------------------------------------------------
26 # [TEST]   Time = 50 | dividend_in = 0, divisor = 10, remainder_in =
    4, quotient_in = 1
27 # ----------------------------------------------------------------------
28 # [OUTPUT] Time = 60 | remainder_out = 8, quotient_out = 2,
    dividend_out = 00000000
29 # [EXPECT] Time = 60 | remainder_out = 8, quotient_out = 2,
    dividend_out = 00000000
30 # ======> PASSED
31 # ----------------------------------------------------------------------
32 # [TEST]   Time = 60 | dividend_in = a0000000, divisor = 3,
    remainder_in = 2, quotient_in = 5
33 # ----------------------------------------------------------------------
34 # [OUTPUT] Time = 70 | remainder_out = 2, quotient_out = 11,
    dividend_out = 40000000
35 # [EXPECT] Time = 70 | remainder_out = 2, quotient_out = 11,
    dividend_out = 40000000
36 # ======> PASSED
37 # ----------------------------------------------------------------------
38 # [TEST]   Time = 70 | dividend_in = 80000000, divisor = 6,
    remainder_in = 2, quotient_in = 0
39 # ----------------------------------------------------------------------
40 # [OUTPUT] Time = 80 | remainder_out = 5, quotient_out = 0,
    dividend_out = 00000000
41 # [EXPECT] Time = 80 | remainder_out = 5, quotient_out = 0,
    dividend_out = 00000000
42 # ======> PASSED
43 # ----------------------------------------------------------------------
44 # [TEST]   Time = 80 | dividend_in = 80000000, divisor = 5,
    remainder_in = 2, quotient_in = 0
45 # ----------------------------------------------------------------------
46 # [OUTPUT] Time = 90 | remainder_out = 0, quotient_out = 1,
    dividend_out = 00000000
47 # [EXPECT] Time = 90 | remainder_out = 0, quotient_out = 1,
    dividend_out = 00000000
48 # ======> PASSED
49 # ----------------------------------------------------------------------
50 # [TEST]   Time = 90 | dividend_in = ffffffff, divisor = 4294967295,
    remainder_in = 4294967295, quotient_in = 4294967295
51 # ----------------------------------------------------------------------
52 # [OUTPUT] Time = 100 | remainder_out = 0, quotient_out = 4294967295,
    dividend_out = fffffffe
53 # [EXPECT] Time = 100 | remainder_out = 0, quotient_out = 4294967295,
```

```
   dividend_out = fffffffe
# ======> PASSED
# ----------------------------------------------------------------------
# [TEST]   Time = 100 | dividend_in = 12345678, divisor = 100,
   remainder_in = 200, quotient_in = 42
# ----------------------------------------------------------------------
# [OUTPUT] Time = 110 | remainder_out = 300, quotient_out = 85,
   dividend_out = 2468acf0
# [EXPECT] Time = 110 | remainder_out = 300, quotient_out = 85,
   dividend_out = 2468acf0
# ======> PASSED
# ----------------------------------------------------------------------
# [TEST]   Time = 110 | dividend_in = 12345678, divisor = 500,
   remainder_in = 200, quotient_in = 42
# ----------------------------------------------------------------------
# [OUTPUT] Time = 120 | remainder_out = 400, quotient_out = 84,
   dividend_out = 2468acf0
# [EXPECT] Time = 120 | remainder_out = 400, quotient_out = 84,
   dividend_out = 2468acf0
# ======> PASSED
# ----------------------------------------------------------------------
# [TEST]   Time = 120 | dividend_in = 0, divisor = 1, remainder_in =
   0, quotient_in = 123
# ----------------------------------------------------------------------
# [OUTPUT] Time = 130 | remainder_out = 0, quotient_out = 246,
   dividend_out = 00000000
# [EXPECT] Time = 130 | remainder_out = 0, quotient_out = 246,
   dividend_out = 00000000
# ======> PASSED
# ----------------------------------------------------------------------
# [TEST]   Time = 130 | dividend_in = 80000000, divisor = 4294967295,
   remainder_in = 0, quotient_in = 0
# ----------------------------------------------------------------------
# [OUTPUT] Time = 140 | remainder_out = 1, quotient_out = 0,
   dividend_out = 00000000
# [EXPECT] Time = 140 | remainder_out = 1, quotient_out = 0,
   dividend_out = 00000000
# ======> PASSED
# ----------------------------------------------------------------------
# SUMMARY
# ----------------------------------------------------------------------
# TOTAL TESTS : 13
# Total PASSED: 13
# Total FAILED: 0
# ======> ALL TESTS PASSED
# ** Note: $finish    : ../../tb/tb_divu_1iter.v(114)
#    Time: 240 ns  Iteration: 0  Instance: /tb_divu_1iter
```

Simulation 4: Testbench results of the divu_1iter module.

```
# ----------------------------------------------------------------------
# [TEST]   Time = 10 | dividend = 4, divisor = 2
# ----------------------------------------------------------------------
# [OUTPUT] Time = 20 | quotient = 2, remainder = 0
```

```
# [EXPECT] Time = 20 | quotient = 2, remainder = 0
# ======> PASSED
# ----------------------------------------------------------------------
# [TEST]   Time = 20 | dividend = 4, divisor = 4
# ----------------------------------------------------------------------
# [OUTPUT] Time = 30 | quotient = 1, remainder = 0
# [EXPECT] Time = 30 | quotient = 1, remainder = 0
# ======> PASSED
# ----------------------------------------------------------------------
# [TEST]   Time = 30 | dividend = 10, divisor = 4
# ----------------------------------------------------------------------
# [OUTPUT] Time = 40 | quotient = 2, remainder = 2
# [EXPECT] Time = 40 | quotient = 2, remainder = 2
# ======> PASSED
# ----------------------------------------------------------------------
# [TEST]   Time = 40 | dividend = 2, divisor = 4
# ----------------------------------------------------------------------
# [OUTPUT] Time = 50 | quotient = 0, remainder = 2
# [EXPECT] Time = 50 | quotient = 0, remainder = 2
# ======> PASSED
# ----------------------------------------------------------------------
# [TEST]   Time = 50 | dividend = 100, divisor = 1
# ----------------------------------------------------------------------
# [OUTPUT] Time = 60 | quotient = 100, remainder = 0
# [EXPECT] Time = 60 | quotient = 100, remainder = 0
# ======> PASSED
# ----------------------------------------------------------------------
# [TEST]   Time = 60 | dividend = 100, divisor = 7
# ----------------------------------------------------------------------
# [OUTPUT] Time = 70 | quotient = 14, remainder = 2
# [EXPECT] Time = 70 | quotient = 14, remainder = 2
# ======> PASSED
# ----------------------------------------------------------------------
# [TEST]   Time = 70 | dividend = 0, divisor = 50
# ----------------------------------------------------------------------
# [OUTPUT] Time = 80 | quotient = 0, remainder = 0
# [EXPECT] Time = 80 | quotient = 0, remainder = 0
# ======> PASSED
# ----------------------------------------------------------------------
# [TEST]   Time = 80 | dividend = 4294967295, divisor = 1
# ----------------------------------------------------------------------
# [OUTPUT] Time = 90 | quotient = 4294967295, remainder = 0
# [EXPECT] Time = 90 | quotient = 4294967295, remainder = 0
# ======> PASSED
# ----------------------------------------------------------------------
# [TEST]   Time = 90 | dividend = 4294967295, divisor = 4294967295
# ----------------------------------------------------------------------
# [OUTPUT] Time = 100 | quotient = 1, remainder = 0
# [EXPECT] Time = 100 | quotient = 1, remainder = 0
# ======> PASSED
# ----------------------------------------------------------------------
# [TEST]   Time = 100 | dividend = 4294967295, divisor = 2
# ----------------------------------------------------------------------
```

```
# [OUTPUT] Time = 110 | quotient = 2147483647, remainder = 1
# [EXPECT] Time = 110 | quotient = 2147483647, remainder = 1
# ======> PASSED
# ---------------------------------------------------------------------
# [TEST]   Time = 110 | dividend = 2147483648, divisor = 2
# ---------------------------------------------------------------------
# [OUTPUT] Time = 120 | quotient = 1073741824, remainder = 0
# [EXPECT] Time = 120 | quotient = 1073741824, remainder = 0
# ======> PASSED
# ---------------------------------------------------------------------
# [TEST]   Time = 120 | dividend = 123456789, divisor = 987
# ---------------------------------------------------------------------
# [OUTPUT] Time = 130 | quotient = 125082, remainder = 855
# [EXPECT] Time = 130 | quotient = 125082, remainder = 855
# ======> PASSED
# ---------------------------------------------------------------------
# [TEST]   Time = 130 | dividend = 400000, divisor = 399999
# ---------------------------------------------------------------------
# [OUTPUT] Time = 140 | quotient = 1, remainder = 1
# [EXPECT] Time = 140 | quotient = 1, remainder = 1
# ======> PASSED
# ---------------------------------------------------------------------
# [TEST]   Time = 140 | dividend = 1024, divisor = 1025
# ---------------------------------------------------------------------
# [OUTPUT] Time = 150 | quotient = 0, remainder = 1024
# [EXPECT] Time = 150 | quotient = 0, remainder = 1024
# ======> PASSED
# ---------------------------------------------------------------------
# SUMMARY
# ---------------------------------------------------------------------
# TOTAL TESTS : 14
# Total PASSED: 14
# Total FAILED: 0
# ======> ALL TESTS PASSED
# ** Note: $finish    : ../../tb/tb_divider_unsigned.v(113)
#    Time: 250 ns  Iteration: 0  Instance: /tb_divider_unsigned
```

Simulation 5: Testbench results of the divider_unsigned module.