VIET NAM NATIONAL UNIVERSITY HO CHI MINH CITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

CO3098 - LSI DESIGN LAB

LAB 1

BOUND FLASHER

Instructor: **PROF. NGUYEN THIEN AN**

Student: Nguyen Khanh Nam - 2153599

HO CHI MINH CITY, MARCH 2024

Contents

1 Problem Implementation

1.1 Problem-Solving Strategies

The problem can be effectively solved by dividing it into states because there are multiple operations. Beside, in order to make the lamps turn on or off gradually, a delay mechanism needs to be applied. In this problem, I will use an integer **counter** variable that will count up to a specific number **TIMER**. Once the **counter** attains the value of **TIMER**, the operation to either turn on or turn off the lamps will be executed.

1.2 Finite State Machine

An effective solution can be devised by employing a Finite State Machine (FSM) approach. By breaking down the task into eight distinct states, we can systematically address each operation outlined in the question. Each state in the FSM corresponds to a specific phase of the task and is designed to execute the required operations in a structured manner. This modular and organized approach not only enhances the clarity of the solution but also facilitates efficient management of the entire process.

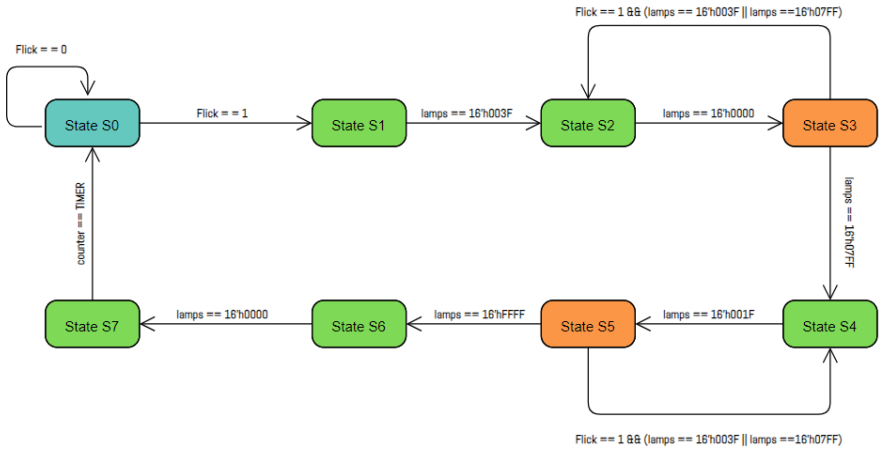


Figure 1: Finite State Machine for the problem.

1.3 Code Implementation

First of all, I will define inputs, output, states and some internal signals.

```
1  `timescale 1ns / 1ps
2
3  module boundFlasher(
4      input clk, flick, rst,
5      output [15:0] lamps
6  );
7
8      // Define states
9      parameter S0 = 0;
10     parameter S1 = 1;
```

```
    parameter S2 = 2;
    parameter S3 = 3;
    parameter S4 = 4;
    parameter S5 = 5;
    parameter S6 = 6;
    parameter S7 = 7;

    // Internal signals
    parameter TIMER = 200;
    integer counter = 0;
    reg [3:0] state = S0;
    reg [15:0] temp;

    // State machine
    always @(posedge clk) begin
        if(rst) begin
            state <= S0;
        end
        else begin
            case(state)
                S0:
                    begin
                        temp <= 16'h0000;
                        if(flick == 0) state <= S0;
                        else state <= S1;
                    end
                S1:
                    begin
                        counter <= counter + 1;
                        if(counter == TIMER) begin
                            temp <= (temp << 1) + 1;
                            counter <= 0;
                        end
                        if(temp == 16'h003F) begin
                            state <= S2;
                            counter <= 0;
                        end
                    end
                S2:
                    begin
                        counter <= counter + 1;
                        if (counter == TIMER) begin
                            temp <= temp >> 1;
                            counter <= 0;
                        end
                        if (temp == 16'h0000) begin
                            state <= S3;
                            counter <= 0;
                        end
```

```
60                      end
61                  S3:
62                      begin
63                          counter <= counter + 1;
64                          if (counter == TIMER) begin
65                              temp <= (temp << 1) + 1;
66                              counter <= 0;
67                          end
68                          // Case flick
69                          if(flick == 1 && temp <= 16'h07FF) begin
70                              if(temp == 16'h003F || temp == 16'h07FF) begin
71                                  counter <= 0;
72                                  state <= S2;
73                              end
74                          end
75                          // Case no flick
76                          else begin
77                              if(temp == 16'h07FF) begin
78                                  counter <= 0;
79                                  state <= S4;
80                              end
81                          end
82                      end
83                  S4:
84                      begin
85                          counter <= counter + 1;
86                          if (counter == TIMER) begin
87                              temp <= temp >> 1;
88                              counter <= 0;
89                          end
90                          if (temp == 16'h001F) begin
91                              counter <= 0;
92                              state <= S5;
93                          end
94                      end
95                  S5:
96                      begin
97                          counter <= counter + 1;
98                          if(counter == TIMER) begin
99                              temp <= (temp << 1) + 1;
100                             counter <= 0;
101                         end
102                         // Case flick
103                         if(flick == 1 && temp <= 16'h07FF) begin
104                             if(temp == 16'h003F || temp == 16'h07FF) begin
105                                 counter <= 0;
106                                 state <= S4;
107                             end
108                         end
```

```
                        // Case no flick
                        else begin
                            if(temp == 16'hFFFF) begin
                                counter <= 0;
                                state <= S6;
                            end
                        end
                    end
                S6:
                    begin
                        counter <= counter + 1;
                        if(counter == TIMER) begin
                            temp <= temp >> 1;
                            counter <= 0;
                        end
                        if (temp == 16'h0000) begin
                            counter <= 0;
                            state <= S7;
                        end
                    end
                S7:
                    begin
                        temp <= 16'hFFFF;
                        counter <= counter + 1;
                        if(counter == TIMER) begin
                            state <= S0;
                            counter <= 0;
                        end
                    end
                endcase
            end
        end
    // Assign output
    assign lamps = temp;

endmodule
```

The descriptions for the states have been demonstrated detailed in the designspec file.

1.4 Simulation

1.4.1 Normal Test

For testing purpose, I will create a testbench file with the below information:

```
module boundFlasher_tb;
        reg clk;
        reg flick;
        reg rst;
        wire [15:0] lamps;

        boundFlasher UUT (
                .clk(clk),
                .flick(flick),
                .rst(rst),
                .lamps(lamps)
    );

        // Create clock
        always #1 clk = !clk;
        initial begin
                // Reset the thing
                clk = 0;
                flick = 0;
                #4;

                // Normal test
                flick = 1;
                #4;
                flick = 0;
                #40000;
                $finish;
        end
        initial begin
                $recordfile ("waves");
                $recordvars ("depth=0", boundFlasher_tb);
        end
endmodule
```

The output of the module will be as below.

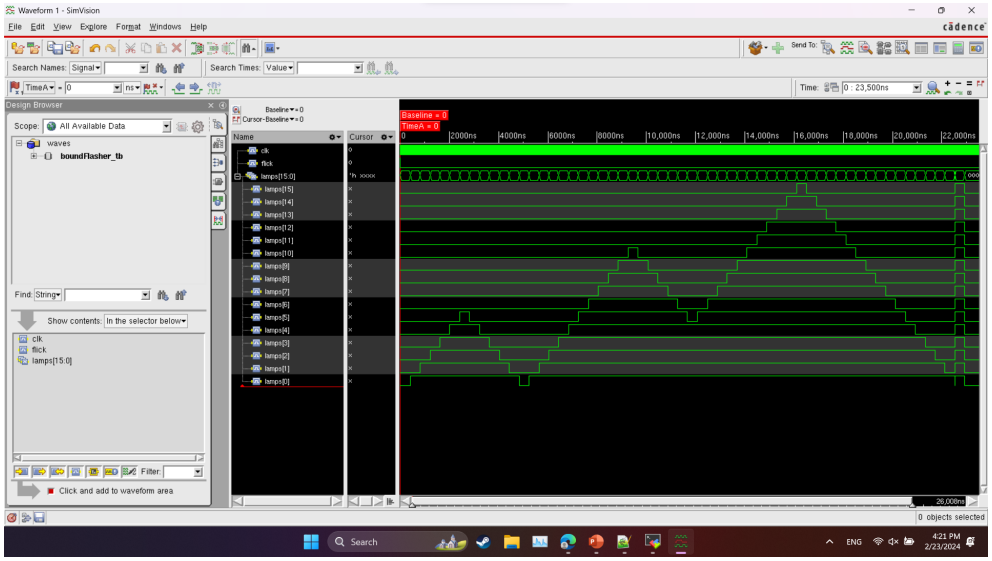
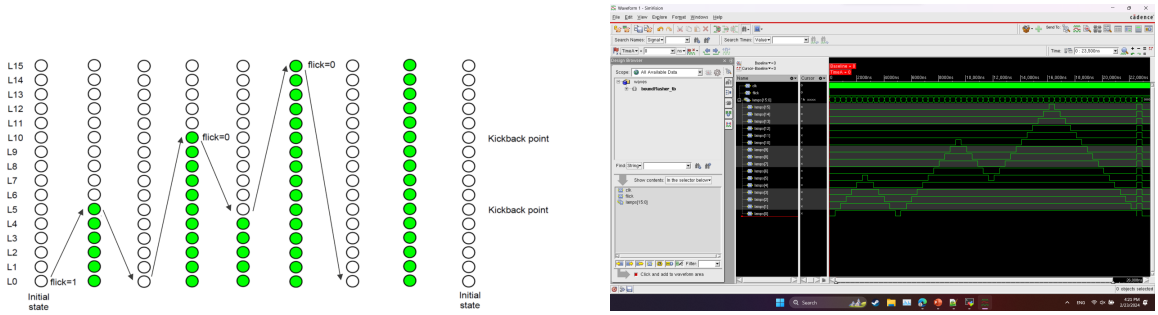


Figure 2: Normal Test without kickback.

It can be seen that, in normal condition, the module work well as expected. The waveform has the same shape as in the theory.



(a) Expected Output (b) Actual Output

Figure 3: Normal Test without additional condition.

1.4.2 Additional Condition Testing

Similar methods are employed in this section. Below is the content of the testbench file.

```
module boundFlasher_tb;
    reg clk;
    reg flick;
    wire [15:0] lamps;

    boundFlasher UUT (
        .clk(clk),
        .flick(flick),
        .lamps(lamps)
    );
    // Create clock
    always #1 clk = !clk;
    initial begin
        // Reset the thing
        clk = 0;
        flick = 0;
        #4;

        // Normal test
        flick = 1;
        #4;
        flick = 0;

        // Slide flick waveform test
        @(UUT.state == 3) begin
            #3500;
            flick = 1;
        end
        @(UUT.state == 2) flick = 0;
        #40000;
        $finish;
    end
    initial begin
        $recordfile ("waves");
        $recordvars ("depth=0", boundFlasher_tb);
    end
endmodule
```

And here is the output of the program.

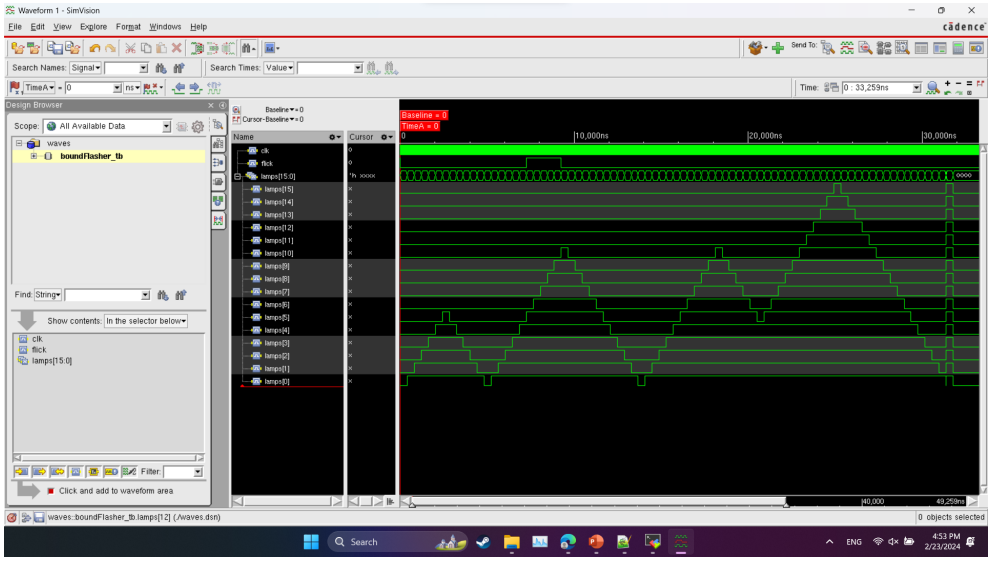
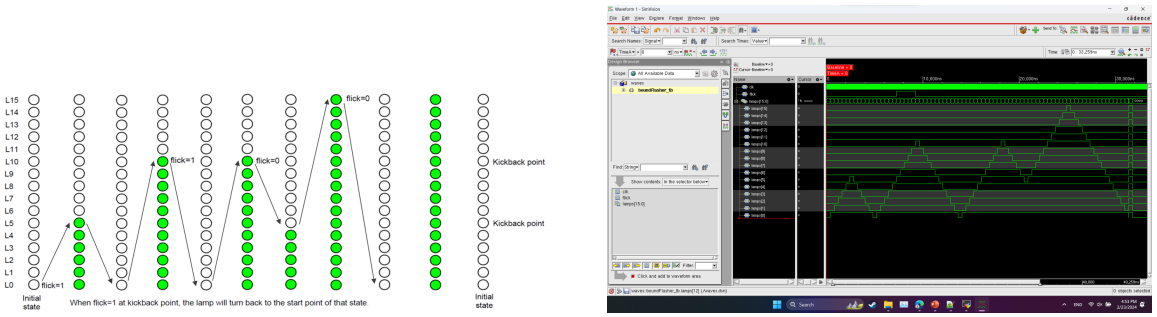


Figure 4: Test with Additional Condition.

Compare between the expected output in the slide with the recent output we will have:



(a) Expected Output (b) Actual Output

Figure 5: Normal Test without additional condition.

1.4.3 Extra Case

In this section, I will create my own test case to see if the system works well as I expected or not. The content below belongs to the testbench file of this test case.

```
module boundFlasher_tb;
    reg clk;
    reg flick;
    wire [15:0] lamps;

    boundFlasher UUT (
        .clk(clk),
        .flick(flick),
        .lamps(lamps)
    );
    // Create clock
    always #1 clk = !clk;
    initial begin
        // Reset the thing
        clk = 0;
        flick = 0;
        #4;

        // Normal test
        flick = 1;
        #4;
        flick = 0;

        // Myself flick waveform test
        @(UUT.state == 5) flick = 1;
        @(UUT.state == 4) flick = 0;
        @(UUT.state == 5) begin
            #2000;
            flick = 1;
        end
        @(UUT.state == 4) flick = 0;
        #40000;
        $finish;
    end
    initial begin
        $recordfile ("waves");
        $recordvars ("depth=0", boundFlasher_tb);
    end
endmodule
```

Here is the output picture.

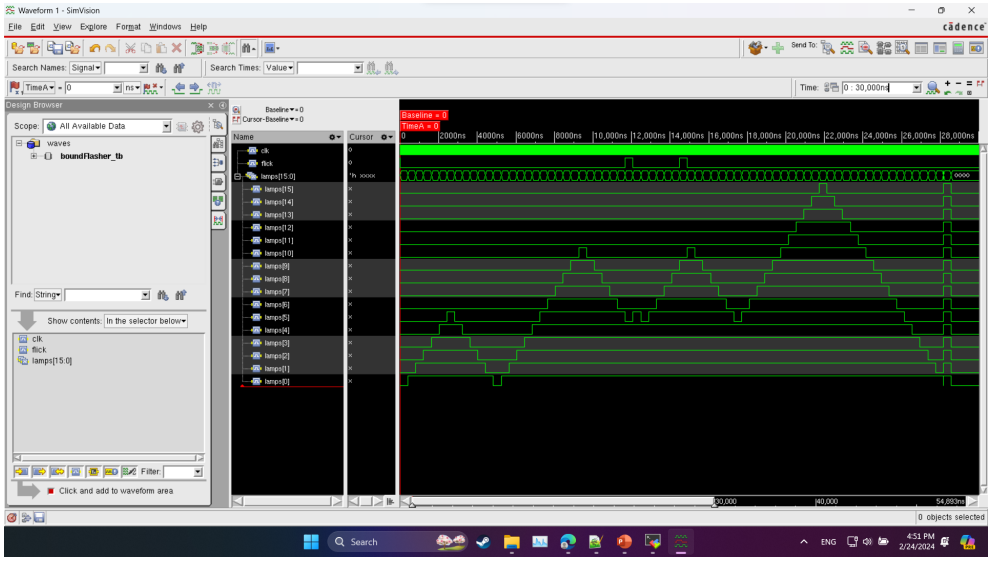


Figure 6: Extra case waveform.

What actually happened is that in state S5 at the beginning, the Flick signal was triggered so that when lamps[5] turned on, it would become the kickback point so that it would go back to state S4. Next, when it came back to state S5, after some delay time, the Flick signal was triggered again at the time lamps[9] turned on. So that when the lamps[10] turned on, it will then become the kickback point and go back to state S4. After all, the system will work normally.

1.4.4 Reset Signal Test

The main target of this section is to test the **Reset** input signal. The test bench will be as below:

```
module boundFlasher_tb;
        reg clk;
        reg flick;
        reg rst;
        wire [15:0] lamps;

        boundFlasher UUT (
                .clk(clk),
                .flick(flick),
                .rst(rst),
                .lamps(lamps)
    );

        // Create clock
        always #1 clk = !clk;
        initial begin
                // Reset the thing
                clk = 0;
                flick = 0;
```

```
20                  #4;
21
22                  // Normal test
23                  flick = 1;
24                  #4;
25                  flick = 0;
26
27                  // Slide flick waveform test
28                  //@(UUT.state == 3) begin
29                  //      #3500;
30                  //      flick = 1;
31                  //end
32                  //@(UUT.state == 2) flick = 0;
33
34                  // Myself flick waveform test
35                  @(UUT.state == 5) flick = 1;
36                  @(UUT.state == 4) flick = 0;
37                  @(UUT.state == 5) begin
38                          #2000;
39                          flick = 1;
40                  end
41                  @(UUT.state == 4) flick = 0;
42
43                  // Rst signal test
44                  #500
45                  rst = 1;
46                  #20;
47                  rst = 0;
48                  flick = 1;
49                  #20;
50                  flick = 0;
51
52                  #40000;
53
54                  $finish;
55          end
56          initial begin
57                  $recordfile ("waves");
58                  $recordvars ("depth=0", boundFlasher_tb);
59          end
60  endmodule
```

Here is the output picture:

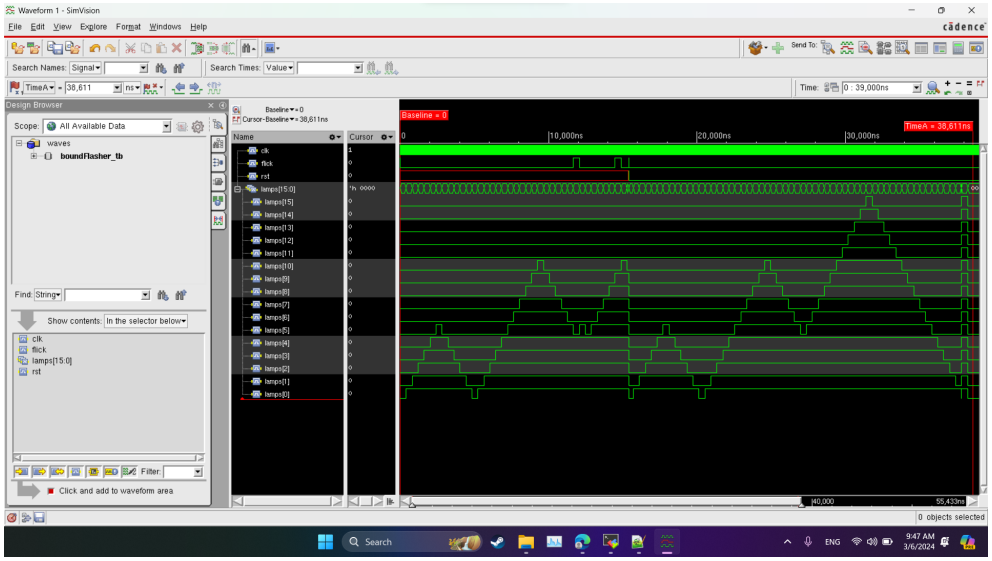


Figure 7: Reset signal test waveform.

It can be seen that when there is the reset signal, the module state will dramatically change to the initial state which has no lamp turned on.

The source code of the problem will be available at: GitHub