FPGA Application Final Project

CYEE 10828241 Chen Da-Chuan

June 15, 2023

Contents

List of Figures

List of Tables

1 Topic

This project is called "Shot the Clock". It is the revised version of my midterm project and achieved with different method.

2 Introduction

State machine is adopted to control the meteor light effect on this revision of "Shot the Clock". With state machine, it is far more easy to coordinate action between multiple different modules and makes controlling logic far more realizable.

Table 1: Utility

No.	Utility
1.	24 hours clock
2.	Adjust hour, minute, second
3.	Fire the laser to destroy the clock

The 6 digits 7-segment LED displays 6 digits of 24h clock. 6 switches either add or subtract from the current hour, minute, or second count, and another switch responsible for resetting clock counter. One button fires the laser and the other reset the damage caused by laser.

Table 2: Control

ID	Function
SW9	Increase hour
SW8	Decrease hour
SW7	Increase minute
SW6	Decrease minute
SW5	Increase second
SW4	Decrease second
SW0	Reset to 00:00:00
KEY0	Fire the laser
KEY1	Turn all numbers back to normal

3 Block Diagram

The entire system has two parts:

1. 24h clock that will increase its value every 1 second. It can also be manually adjusted by switches. This part of system is driven by clock signal generated from "clock_all" module, switch signal from "SW_DEBOUNCE" module, logic processed in "eclock" module, and output to 7-segment LED driver "SEG7_LUT_6" module.

2. Meteor light state machine that triggered whenever KEY[0] is pressed. This state machine controlls all 10 of the LEDs above the switches and 6 of the 7-segment LEDs. The module "lightStateCtr" module controlls this and make them light up and close one by one.

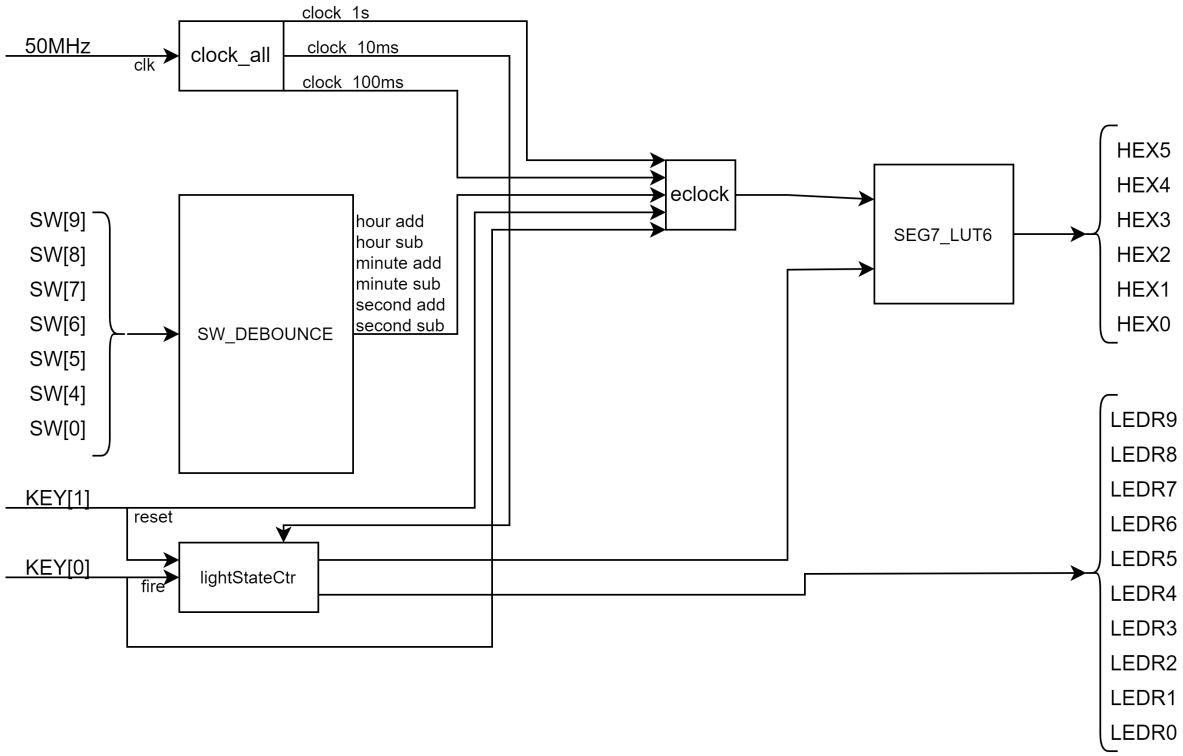


Figure 1: Block Diagram

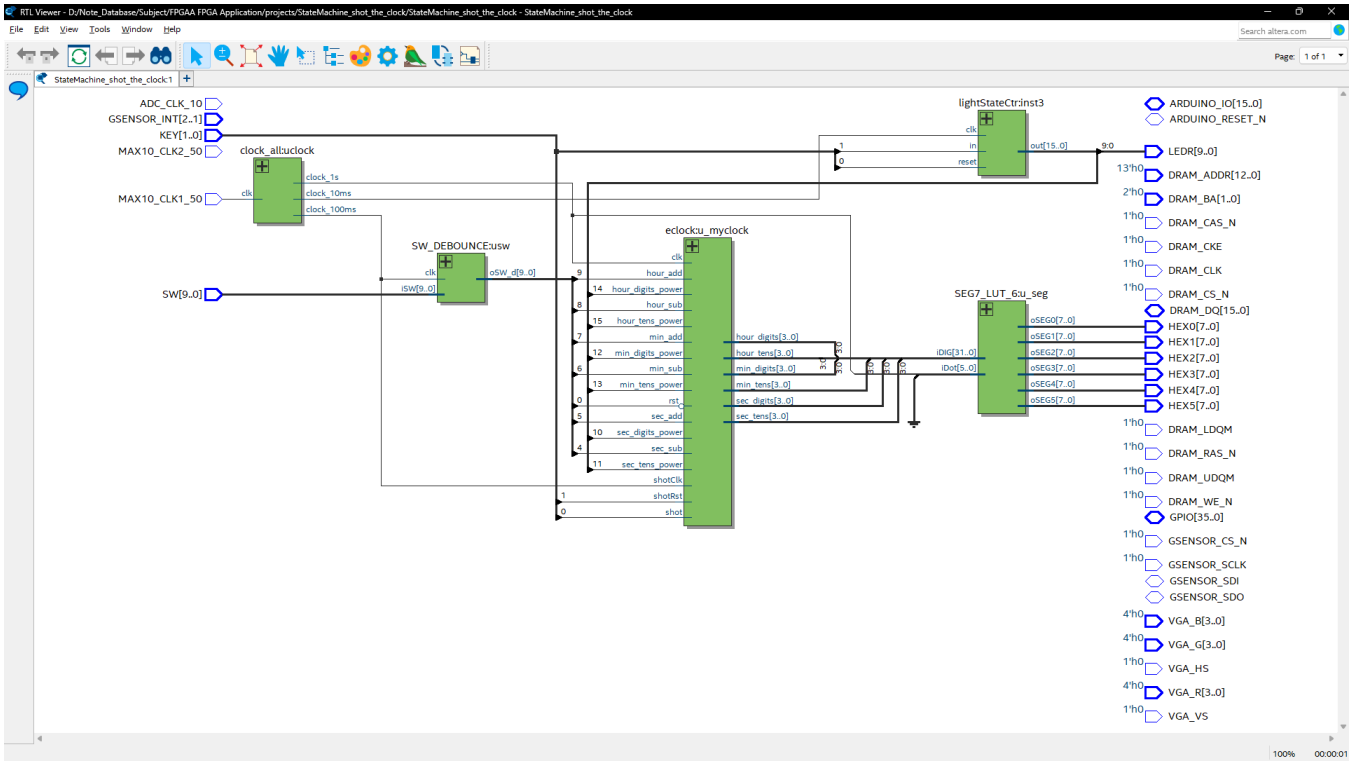


Figure 2: RTL viewer

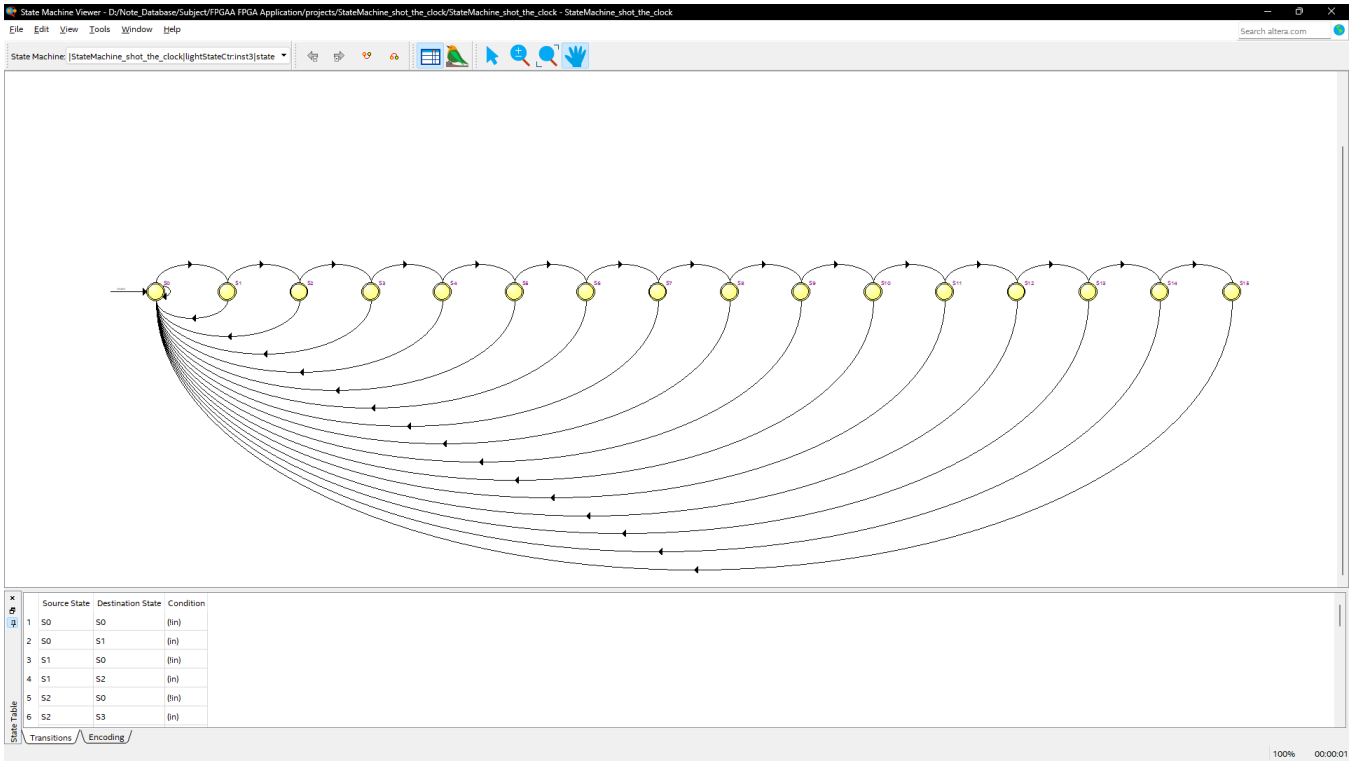


Figure 3: State Machine

4 System Flow

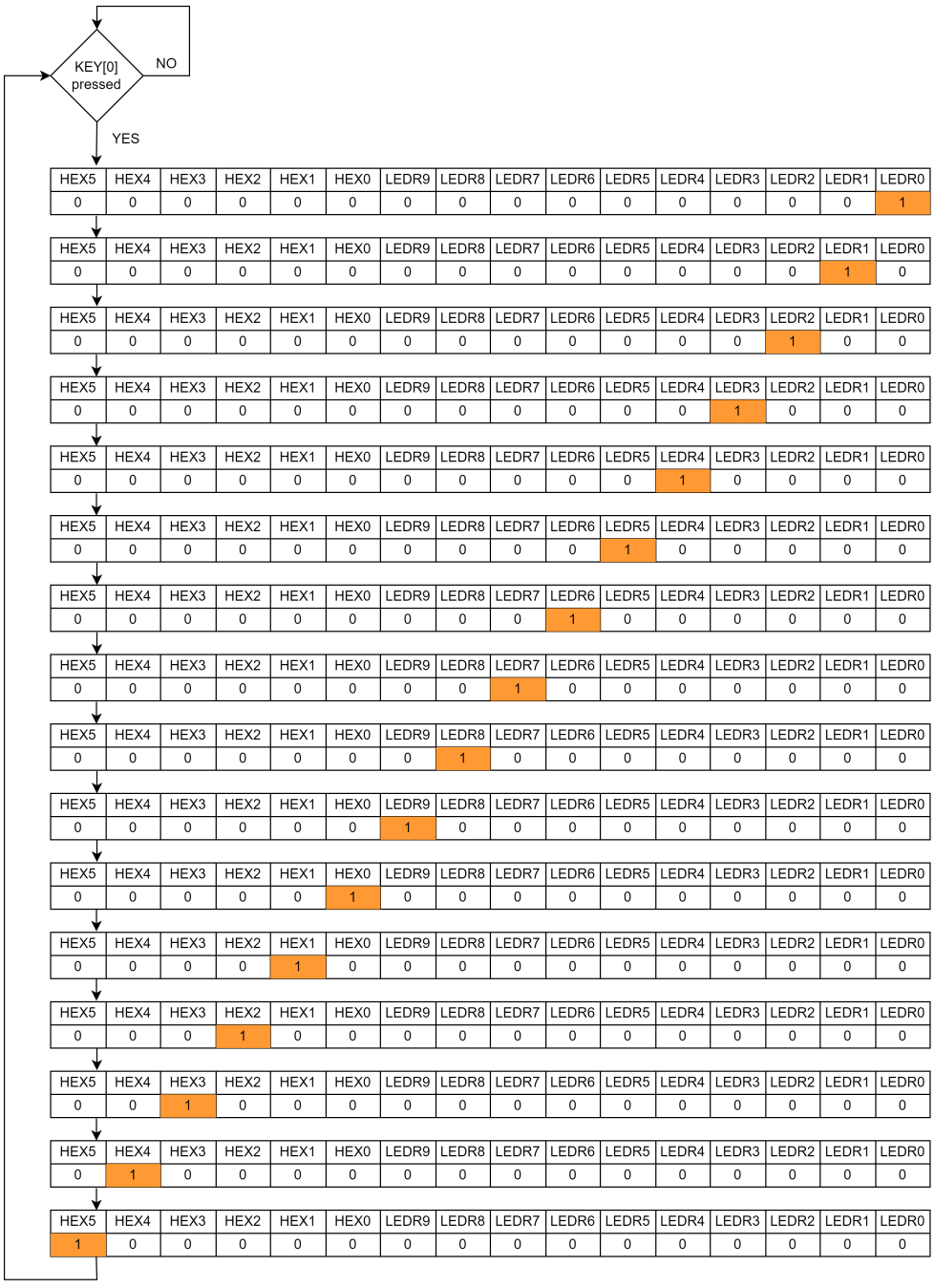


Figure 4: Flow Chart

5 ModelSim

For all of the simulations below, the code is modified to take different input clock so that it is possible to simulate without waiting super long time for seconds of simulation to finish.

In figure 5, after simulated switch signal at the red line, it triggers hour addition at the orange line.

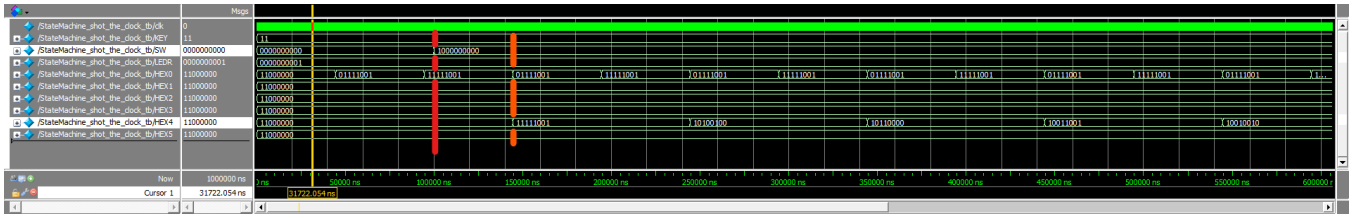


Figure 5: ModelSim clock hour addition

In figure 6, after simulated switch signal at the red line, it triggers hour subtraction at the orange line.

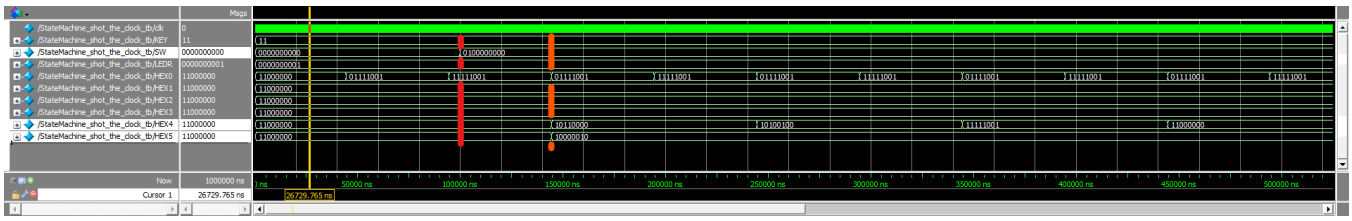


Figure 6: ModelSim clock hour subtraction

In figure 7, after simulated switch signal at the red line, it triggers minute addition at the orange line.

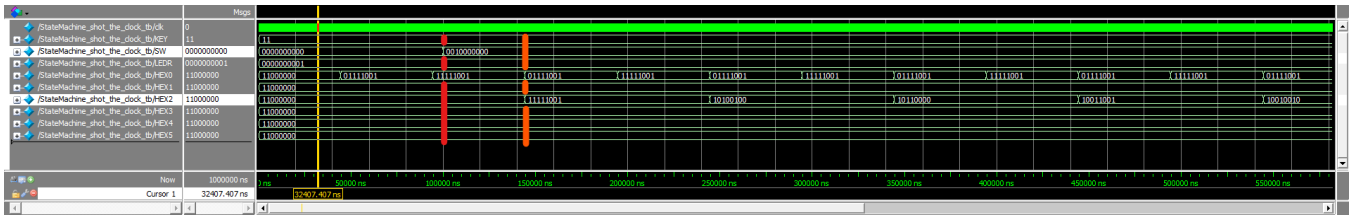


Figure 7: ModelSim clock minute addition

In figure 8, after simulated switch signal at the red line, it triggers minute subtraction at the orange line.

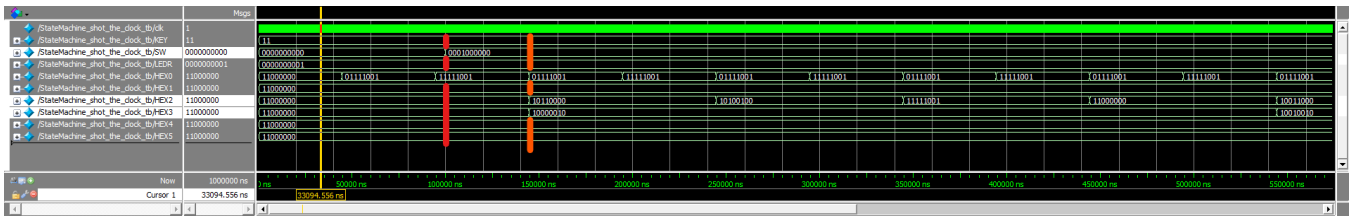


Figure 8: ModelSim clock minute subtraction

In figure 9, after simulated switch signal at the red line, it triggers second addition at the orange line.

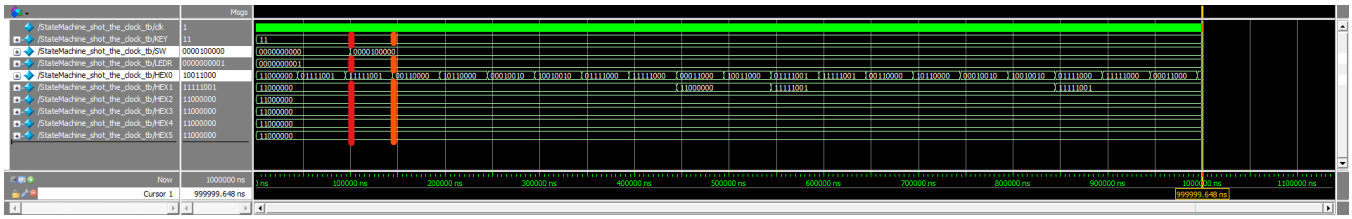


Figure 9: ModelSim clock second addition

In figure 10, after simulated switch signal at the red line, it triggers second subtraction at the orange line.

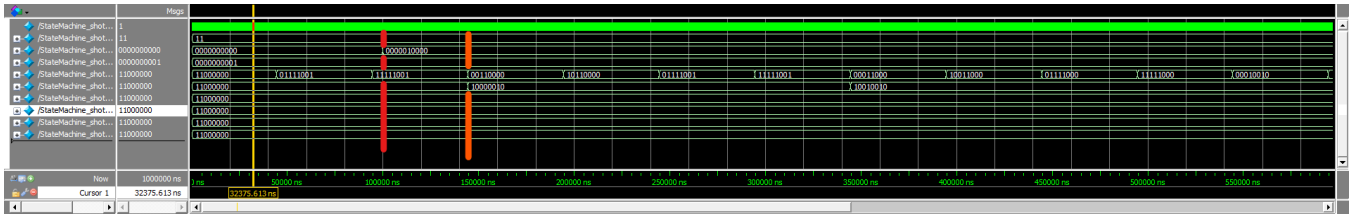


Figure 10: ModelSim clock second subtraction

In figure 11, after simulated switch signal at the red line, it triggers clock reset at the orange line.

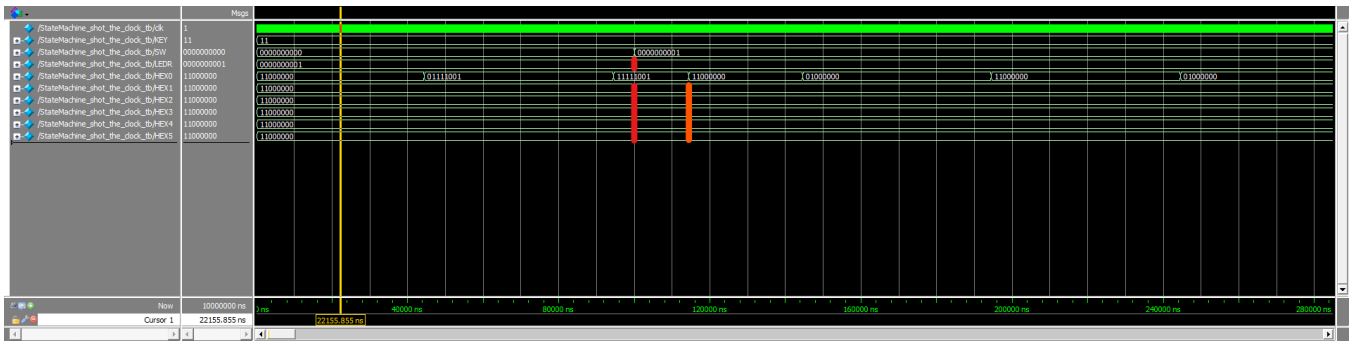


Figure 11: ModelSim clock reset

In figure 12, after simulated KEY[0] signal at both of the red line, it triggers meteor light state machine to run and reset at the first and the second orange line.

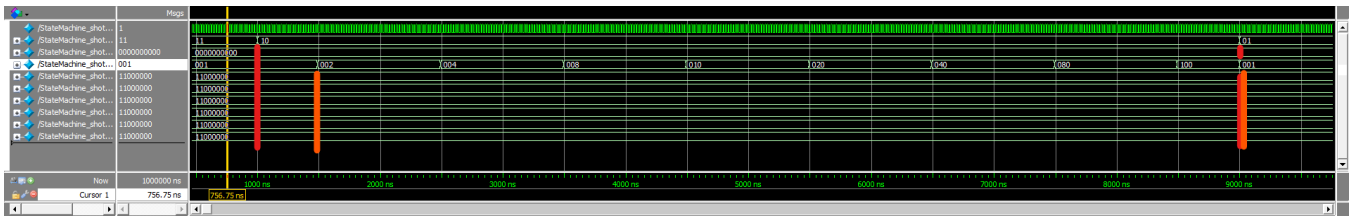


Figure 12: ModelSim meteor light

6 Signal Tap Logic Analyzer

Figure 13 is the signal tap from FPGA, the almost the same signal simulation is shown in figure 14. The slight difference might be caused by the different clock settings in simulation condition.

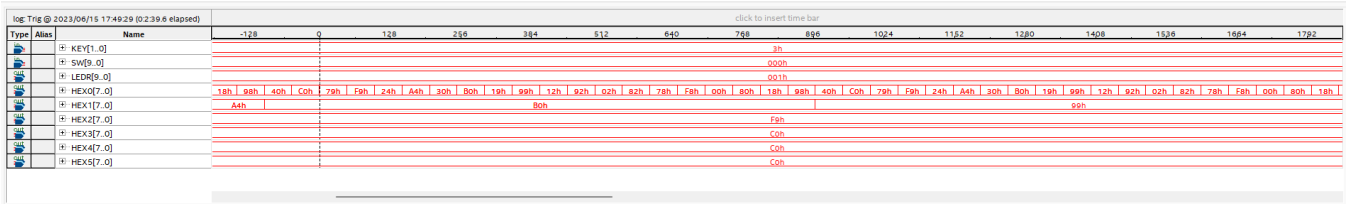


Figure 13: Signal Tap Logic Analyzer

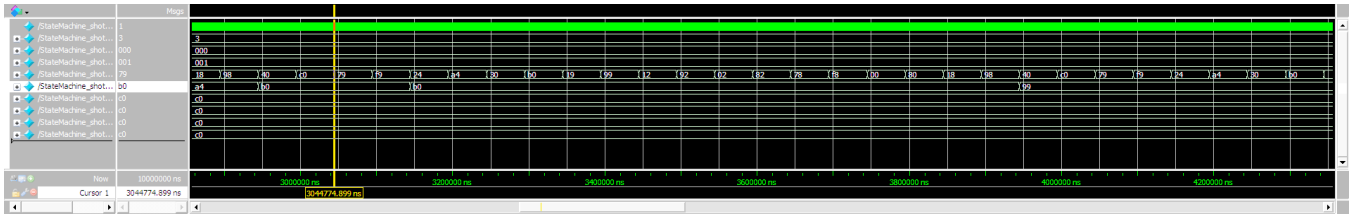


Figure 14: Signal Tap Logic Analyzer simulation

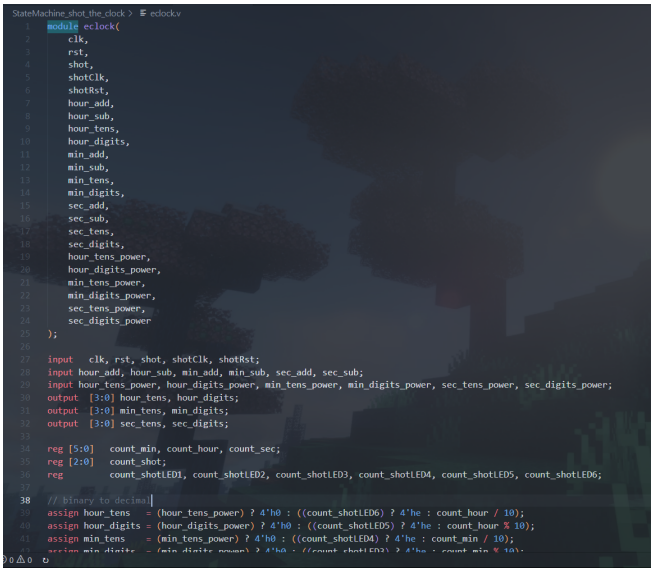
7 Demonstration

Project demonstration video is available at `https://youtu.be/BCmbcEDu8TM`.

8 Process

Module "SW_DEBOUNCE", "clock_all" and "SEG7_LUT_6" will not be covered below, since they are the exact same replica of the original code provided by professor.

1. Figure 15 Line1-36: Define module, ports and registers.
2. Figure 15-16 Line39-44: Let 7-segment displays to be controlled with meteor LED state machine, and turn off if hit by laser.
3. Figure 16 Line46-56: Count up every time KEY[0] is pressed.
4. Figure 16-17 Line58-93: Turn off a 7-segment display accordingly whenever KEY[0] is pressed.
5. Figure 17-18 Line95-153: 24h clock counting up, addition and subtraction of hour, minute, and seconds.
6. Figure 19 Line4-11: Define module, ports and registers.
7. Figure 19 Line14: Declare all possible states.
8. Figure 19-20 Line17-54: Define values of states.
9. Figure 20-22 Line57-143: Define the sequence of states.
10. Figure 23-25 Line6-83: Define module, ports and wires.
11. Figure 25 Line89-94: Include switch debounce module.
12. Figure 25 Line96-104: Include clock_all module.
13. Figure 25-26 Line106-132: Include eclock module.
14. Figure 26 Line134-144: Include 7-segment display driver module.
15. Figure 26 Line146-152: Include meteor light state machine module.
16. Figure 27 Line1-10: Define timescale, module, registers and wires.
17. Figure 27 Line12-23: Include top-level module for simulation.
18. Figure 27 Line25-28: Initial condition for normal clock incrementationn simulation.
19. Figure 27 Line25-32: Initial condition for meteor light state machine simulation.
20. Figure 27 Line25-28,34-40: Initial condition for hour addition, hour subtraction, minute addition, minute subtraction, second addition, second subtraction simulation.



```
module eclock(
    clk,
    rst,
    shot,
    shotClk,
    shotRst,
    hour_add,
    hour_sub,
    hour_tens,
    hour_digits,
    min_add,
    min_sub,
    min_tens,
    min_digits,
    sec_add,
    sec_sub,
    sec_tens,
    sec_digits,
    hour_tens_power,
    hour_digits_power,
    min_tens_power,
    min_digits_power,
    sec_tens_power,
    sec_digits_power
);

input   clk, rst, shot, shotClk, shotRst;
input hour_add, hour_sub, min_add, min_sub, sec_add, sec_sub;
input hour_tens_power, hour_digits_power, min_tens_power, min_digits_power, sec_tens_power, sec_digits_power;
output  [3:0] hour_tens, hour_digits;
output  [3:0] min_tens, min_digits;
output  [3:0] sec_tens, sec_digits;

reg [5:0]   count_min, count_hour, count_sec;
reg [2:0]   count_shot;
reg         count_shotLED1, count_shotLED2, count_shotLED3, count_shotLED4, count_shotLED5, count_shotLED6;

// binary to decimal
assign hour_tens   = (hour_tens_power) ? 4'h0 : ((count_shotLED6) ? 4'he : count_hour / 10);
assign hour_digits = (hour_digits_power) ? 4'h0 : ((count_shotLED5) ? 4'he : count_hour % 10);
assign min_tens    = (min_tens_power) ? 4'h0 : ((count_shotLED4) ? 4'he : count_min / 10);
```

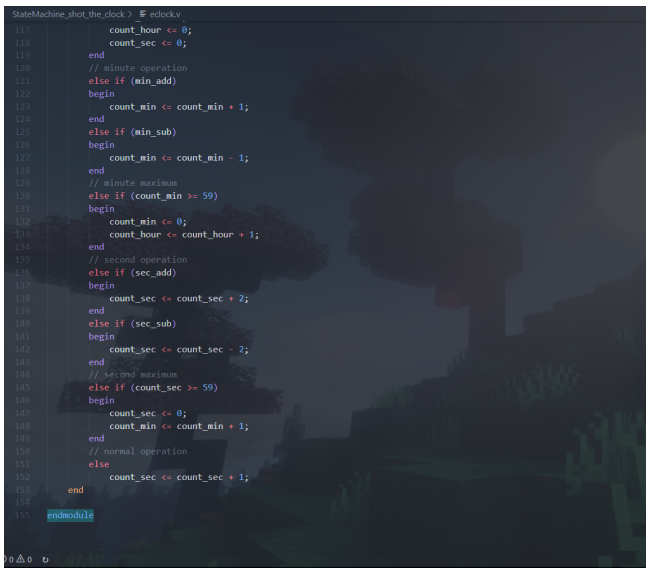
Figure 15: Module: eclock 1

```
assign min_tens    = (min_tens_power) ? 4'h0 : ((count_shotLED4) ? 4'he : count_min / 10);
assign min_digits  = (min_digits_power) ? 4'h0 : ((count_shotLED3) ? 4'he : count_min % 10);
assign sec_tens    = (sec_tens_power) ? 4'h0 : ((count_shotLED2) ? 4'he : count_sec / 10);
assign sec_digits  = (sec_digits_power) ? 4'h0 : ((count_shotLED1) ? 4'he : count_sec % 10);

    always @ (posedge shot or negedge shotRst)
    begin
        if (!shotRst)
        begin
            count_shot <= 0;
        end
        else
        begin
            count_shot <= count_shot + 1;
        end
    end

    always @ (posedge shotClk or negedge shotRst)
    begin
        if (!shotRst)
        begin
            count_shotLED1 <= 0;
            count_shotLED2 <= 0;
            count_shotLED3 <= 0;
            count_shotLED4 <= 0;
            count_shotLED5 <= 0;
            count_shotLED6 <= 0;
        end
        else if (count_shot == 1)
        begin
            count_shotLED1 <= 1;
        end
        else if (count_shot == 2)
        begin
            count_shotLED2 <= 1;
        end
        else if (count_shot == 3)
        begin
            count_shotLED3 <= 1;
        end
        else if (count_shot == 4)
```

Figure 16: Module: eclock 2

```
        end
        else if (count_shot == 4)
        begin
            count_shotLED4 <= 1;
        end
        else if (count_shot == 5)
        begin
            count_shotLED5 <= 1;
        end
        else if (count_shot == 6)
        begin
            count_shotLED6 <= 1;
        end
    end

    always @ (posedge clk or negedge rst)
    begin
        // reset
        if (!rst)
        begin
            count_hour  <= 0;
            count_min <= 0;
            count_sec <= 0;
        end
        // hour operation
        else if (hour_add)
        begin
            count_hour <= count_hour + 1;
        end
        else if (hour_sub )
        begin
            count_hour <= count_hour - 1;
        end
        // hour maximum
        else if (count_hour >= 23 && count_min >= 59 && count_sec >= 59 )
        begin
            count_min <= 0;
            count_hour <= 0;
            count_sec <= 0;
        end
        // minute operation
```

Figure 17: Module: eclock 3



```
            count_hour <= 0;
            count_sec <= 0;
        end
        // minute operation
        else if (min_add)
        begin
            count_min <= count_min + 1;
        end
        else if (min_sub)
        begin
            count_min <= count_min - 1;
        end
        // minute maximum
        else if (count_min >= 59)
        begin
            count_min <= 0;
            count_hour <= count_hour + 1;
        end
        // second operation
        else if (sec_add)
        begin
            count_sec <= count_sec + 2;
        end
        else if (sec_sub)
        begin
            count_sec <= count_sec - 2;
        end
        // second maximum
        else if (count_sec >= 59)
        begin
            count_sec <= 0;
            count_min <= count_min + 1;
        end
        // normal operation
        else
            count_sec <= count_sec + 1;
    end

endmodule
```

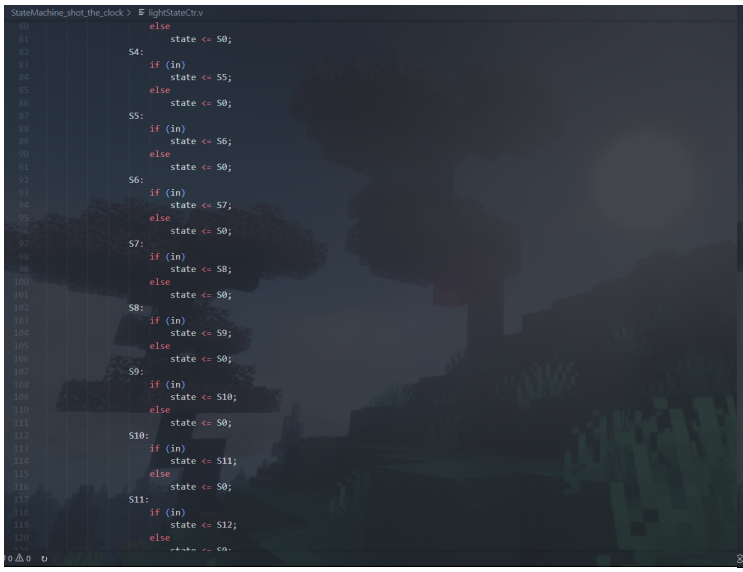
Figure 18: Module: eclock 4

```
StateMachine_shot_the_clock > lightStateCtr.v
1  // Quartus Prime Verilog Template
2  // User-encoded state machine
3
4  module lightStateCtr
5  (
6      input   clk, in, reset,
7      output reg [15:0] out
8  );
9
10     // Declare state register
11     (* syn_encoding = "user" *) reg [3:0] state;
12
13     // Declare states
14     parameter S0 = 0, S1 = 1, S2 = 2, S3 = 3, S4 = 4, S5 = 5, S6 = 6, S7 = 7, S8 = 8, S9 = 9, S10 = 10, S11 = 11, S12 = 12, S13 = 13
15
16     // Output depends only on the state
17     always @ (state) begin
18         case (state)
19             S0:
20                 out = 16'b0000000000000001;
21             S1:
22                 out = 16'b0000000000000010;
23             S2:
24                 out = 16'b0000000000000100;
25             S3:
26                 out = 16'b0000000000001000;
27             S4:
28                 out = 16'b0000000000010000;
29             S5:
30                 out = 16'b0000000000100000;
31             S6:
32                 out = 16'b0000000001000000;
33             S7:
34                 out = 16'b0000000010000000;
35             S8:
36                 out = 16'b0000000100000000;
37             S9:
38                 out = 16'b0000001000000000;
39             S10:
40                 out = 16'b0000010000000000;
41             S11:
```

Figure 19: Module: lightStateCtr 1

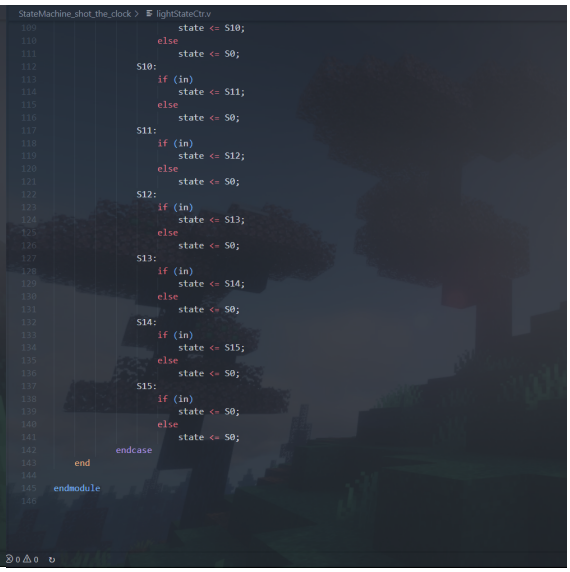
```
StateMachine_shot_the_clock > lightStateCtr.v
41             S11:
42                 out = 16'b0000100000000000;
43             S12:
44                 out = 16'b0001000000000000;
45             S13:
46                 out = 16'b0010000000000000;
47             S14:
48                 out = 16'b0100000000000000;
49             S15:
50                 out = 16'b1000000000000000;
51             default:
52                 out = 16'b0000000000000001;
53         endcase
54     end
55
56     // Determine the next state
57     always @ (posedge clk or posedge reset) begin
58         if (reset)
59             state <= S0;
60         else
61             case (state)
62                 S0:
63                     if (in)
64                         state <= S1;
65                     else
66                         state <= S0;
67                 S1:
68                     if (in)
69                         state <= S2;
70                     else
71                         state <= S0;
72                 S2:
73                     if (in)
74                         state <= S3;
75                     else
76                         state <= S0;
77                 S3:
78                     if (in)
79                         state <= S4;
80                     else
81                         state <= S0;
```

Figure 20: Module: lightStateCtr 2



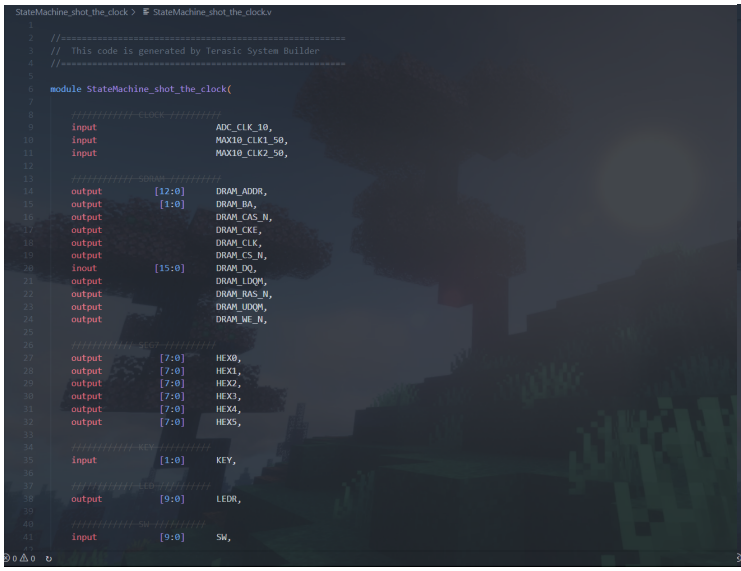
```
StateMachine_shot_the_clock > lightStateCtr.v
80                     else
81                         state <= S0;
82                 S4:
83                     if (in)
84                         state <= S5;
85                     else
86                         state <= S0;
87                 S5:
88                     if (in)
89                         state <= S6;
90                     else
91                         state <= S0;
92                 S6:
93                     if (in)
94                         state <= S7;
95                     else
96                         state <= S0;
97                 S7:
98                     if (in)
99                         state <= S8;
100                    else
101                        state <= S0;
102                S8:
103                    if (in)
104                        state <= S9;
105                    else
106                        state <= S0;
107                S9:
108                    if (in)
109                        state <= S10;
110                    else
111                        state <= S0;
112                S10:
113                    if (in)
114                        state <= S11;
115                    else
116                        state <= S0;
117                S11:
118                    if (in)
119                        state <= S12;
120                    else
```

Figure 21: Module: lightStateCtr 3



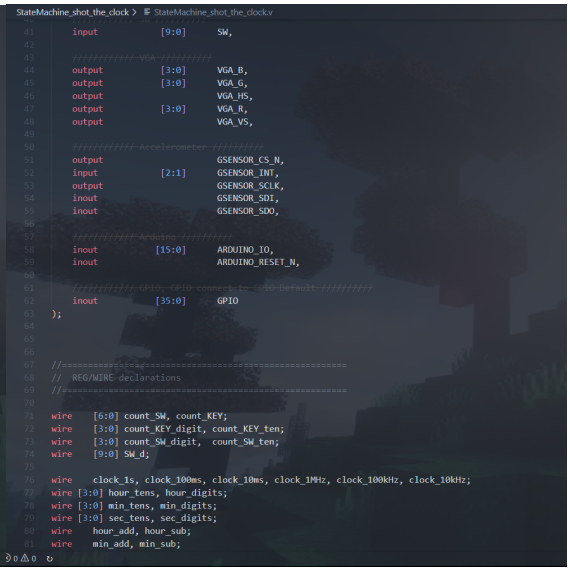
```
StateMachine_shot_the_clock > lightStateCtr.v
109                        state <= S10;
110                    else
111                        state <= S0;
112                S10:
113                    if (in)
114                        state <= S11;
115                    else
116                        state <= S0;
117                S11:
118                    if (in)
119                        state <= S12;
120                    else
121                        state <= S0;
122                S12:
123                    if (in)
124                        state <= S13;
125                    else
126                        state <= S0;
127                S13:
128                    if (in)
129                        state <= S14;
130                    else
131                        state <= S0;
132                S14:
133                    if (in)
134                        state <= S15;
135                    else
136                        state <= S0;
137                S15:
138                    if (in)
139                        state <= S0;
140                    else
141                        state <= S0;
142            endcase
143    end
144
145    endmodule
146
```

Figure 22: Module: lightStateCtr 4



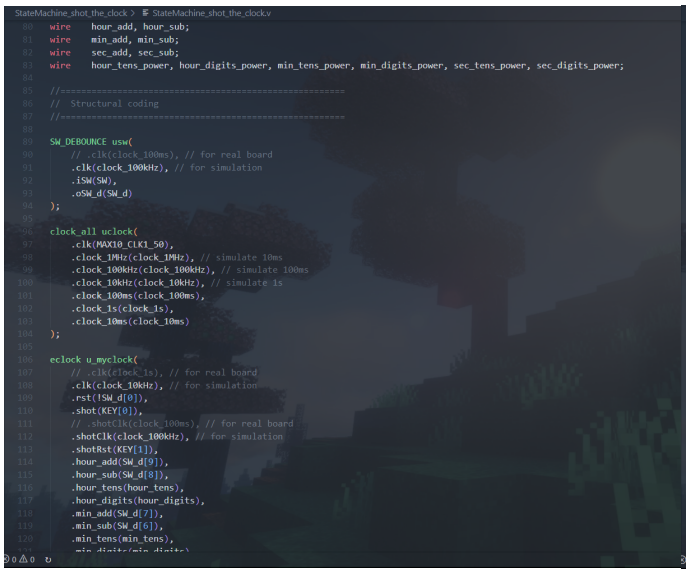
```
StateMachine_shot_the_clock > StateMachine_shot_the_clock.v

//=======================================================
//  This code is generated by Terasic System Builder
//=======================================================

module StateMachine_shot_the_clock(

    //////////// CLOCK //////////
    input                       ADC_CLK_10,
    input                       MAX10_CLK1_50,
    input                       MAX10_CLK2_50,

    //////////// SDRAM //////////
    output          [12:0]      DRAM_ADDR,
    output           [1:0]      DRAM_BA,
    output                      DRAM_CAS_N,
    output                      DRAM_CKE,
    output                      DRAM_CLK,
    output                      DRAM_CS_N,
    inout           [15:0]      DRAM_DQ,
    output                      DRAM_LDQM,
    output                      DRAM_RAS_N,
    output                      DRAM_UDQM,
    output                      DRAM_WE_N,

    //////////// SEG7 //////////
    output           [7:0]      HEX0,
    output           [7:0]      HEX1,
    output           [7:0]      HEX2,
    output           [7:0]      HEX3,
    output           [7:0]      HEX4,
    output           [7:0]      HEX5,

    //////////// KEY //////////
    input            [1:0]      KEY,

    //////////// LED //////////
    output           [9:0]      LEDR,

    //////////// SW //////////
    input            [9:0]      SW,
```

Figure 23: Module: StateMachine_shot_the_clock 1



```
StateMachine_shot_the_clock > StateMachine_shot_the_clock.v

    input            [9:0]      SW,

    //////////// VGA //////////
    output           [3:0]      VGA_B,
    output           [3:0]      VGA_G,
    output                      VGA_HS,
    output           [3:0]      VGA_R,
    output                      VGA_VS,

    //////////// Accelerometer //////////
    output                      GSENSOR_CS_N,
    input            [2:1]      GSENSOR_INT,
    output                      GSENSOR_SCLK,
    inout                       GSENSOR_SDI,
    inout                       GSENSOR_SDO,

    //////////// Arduino //////////
    inout           [15:0]      ARDUINO_IO,
    inout                       ARDUINO_RESET_N,

    //////////// GPIO, GPIO connect to GPIO Default //////////
    inout           [35:0]      GPIO
);


//=======================================================
//  REG/WIRE declarations
//=======================================================

wire    [6:0] count_SW, count_KEY;
wire    [3:0] count_KEY_digit, count_KEY_ten;
wire    [3:0] count_SW_digit,  count_SW_ten;
wire    [9:0] SW_d;

wire    clock_1s, clock_100ms, clock_10ms, clock_1MHz, clock_100kHz, clock_10kHz;
wire [3:0] hour_tens, hour_digits;
wire [3:0] min_tens, min_digits;
wire [3:0] sec_tens, sec_digits;
wire    hour_add, hour_sub;
wire    min_add, min_sub;
```

Figure 24: Module: StateMachine_shot_the_clock 2



```
StateMachine_shot_the_clock > StateMachine_shot_the_clock.v

wire    hour_add, hour_sub;
wire    min_add, min_sub;
wire    sec_add, sec_sub;
wire    hour_tens_power, hour_digits_power, min_tens_power, min_digits_power, sec_tens_power, sec_digits_power;

//=======================================================
//  Structural coding
//=======================================================

SW_DEBOUNCE usw(
    // .clk(clock_100ms), // for real board
    .clk(clock_100kHz), // for simulation
    .iSW(SW),
    .oSW_d(SW_d)
);

clock_all uclock(
    .clk(MAX10_CLK1_50),
    .clock_1MHz(clock_1MHz), // simulate 10ms
    .clock_100kHz(clock_100kHz), // simulate 100ms
    .clock_10kHz(clock_10kHz), // simulate 1s
    .clock_100ms(clock_100ms),
    .clock_1s(clock_1s),
    .clock_10ms(clock_10ms)
);

eclock u_myclock(
    // .clk(clock_1s), // for real board
    .clk(clock_10kHz), // for simulation
    .rst(!SW_d[0]),
    .shot(KEY[0]),
    // .shotClk(clock_100ms), // for real board
    .shotClk(clock_100kHz), // for simulation
    .shotRst(KEY[1]),
    .hour_add(SW_d[9]),
    .hour_sub(SW_d[8]),
    .hour_tens(hour_tens),
    .hour_digits(hour_digits),
    .min_add(SW_d[7]),
    .min_sub(SW_d[6]),
    .min_tens(min_tens),
```

Figure 25: Module: StateMachine_shot_the_clock 3

```
StateMachine_shot_the_clock > StateMachine_shot_the_clock.v

    .hour_add(SW_d[9]),
    .hour_sub(SW_d[8]),
    .hour_tens(hour_tens),
    .hour_digits(hour_digits),
    .min_add(SW_d[7]),
    .min_sub(SW_d[6]),
    .min_tens(min_tens),
    .min_digits(min_digits),
    .sec_add(SW_d[5]),
    .sec_sub(SW_d[4]),
    .sec_tens(sec_tens),
    .sec_digits(sec_digits),
    .hour_tens_power(hour_tens_power),
    .hour_digits_power(hour_digits_power),
    .min_tens_power(min_tens_power),
    .min_digits_power(min_digits_power),
    .sec_tens_power(sec_tens_power),
    .sec_digits_power(sec_digits_power)
);

SEG7_LUT_6 u_seg(
    .oSEG0(HEX0),
    .oSEG1(HEX1),
    .oSEG2(HEX2),
    .oSEG3(HEX3),
    .oSEG4(HEX4),
    .oSEG5(HEX5),
    .iDIG ({hour_tens, hour_digits, min_tens, min_digits, sec_tens, sec_digits }),
    // .iDot ({5'b0, clock_1s}) // for real board
    .iDot ({5'b0, clock_10kHz}) // for simulation
);

lightStateCtr inst3 (
    // .clk(clock_10ms), // for real board
    .clk(clock_1MHz), // for simulation
    .in(KEY[1]),
    .reset(KEY[0]),
    .out({hour_tens_power, hour_digits_power, min_tens_power, min_digits_power, sec_tens_power, sec_digits_power, LEDR})
);

endmodule
```

Figure 26: Module: StateMachine_shot_the_clock 4

Figure 27: Module: StateMachine_shot_the_clock_tb 1

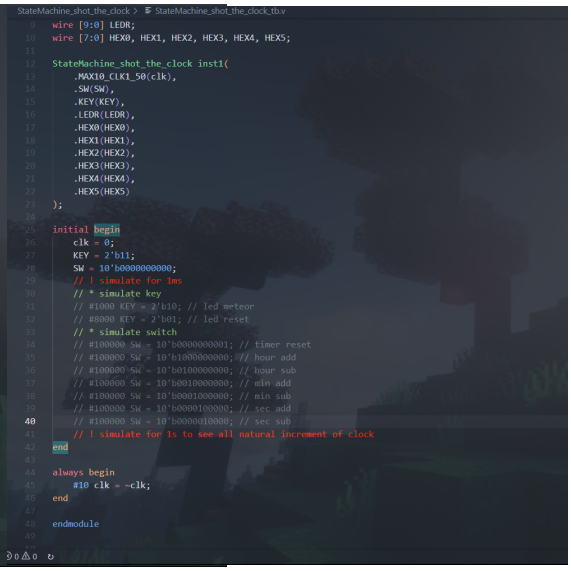


Figure 28: Module: StateMachine_shot_the_clock_tb 2

The entire project is archieved together and is available at `https://github.com/belongtothenight/FPGAA_Code/blob/main/src/StateMachine_shot_the_clock.qar`.