

FPGA Application Final Project

CYEE 10828241 Chen Da-Chuan

June 15, 2023

Contents

1 Topic	2
2 Introduction	2
3 Block Diagram	2
4 System Flow	5
5 ModelSim	5
6 Signal Tap Logic Analyzer	7
7 Demonstration	8
8 Process	8

List of Figures

1 Block Diagram	3
2 RTL viewer	4
3 State Machine	4
4 Flow Chart	5
5 ModelSim clock hour addition	6
6 ModelSim clock hour subtraction	6
7 ModelSim clock minute addition	6
8 ModelSim clock minute subtraction	6
9 ModelSim clock second addition	6
10 ModelSim clock second subtraction	7
11 ModelSim clock reset	7
12 ModelSim meteor light	7
13 Signal Tap Logic Analyzer	7
14 Signal Tap Logic Analyzer simulation	8
15 Module: eclock 1	9
16 Module: eclock 2	9
17 Module: eclock 3	9
18 Module: eclock 4	9
19 Module: lightStateCtr 1	10
20 Module: lightStateCtr 2	10
21 Module: lightStateCtr 3	10
22 Module: lightStateCtr 4	10
23 Module: StateMachine_shot_the_clock 1	11
24 Module: StateMachine_shot_the_clock 2	11
25 Module: StateMachine_shot_the_clock 3	11
26 Module: StateMachine_shot_the_clock 4	11

27	Module: StateMachine_shot_the_clock_tb 1	12
28	Module: StateMachine_shot_the_clock_tb 2	12

List of Tables

1	Utility	2
2	Control	2

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 controls all 10 of the LEDs above the switches and 6 of the 7-segment LEDs. The module "lightStateCtr" module controls this and make them light up and close one by one.

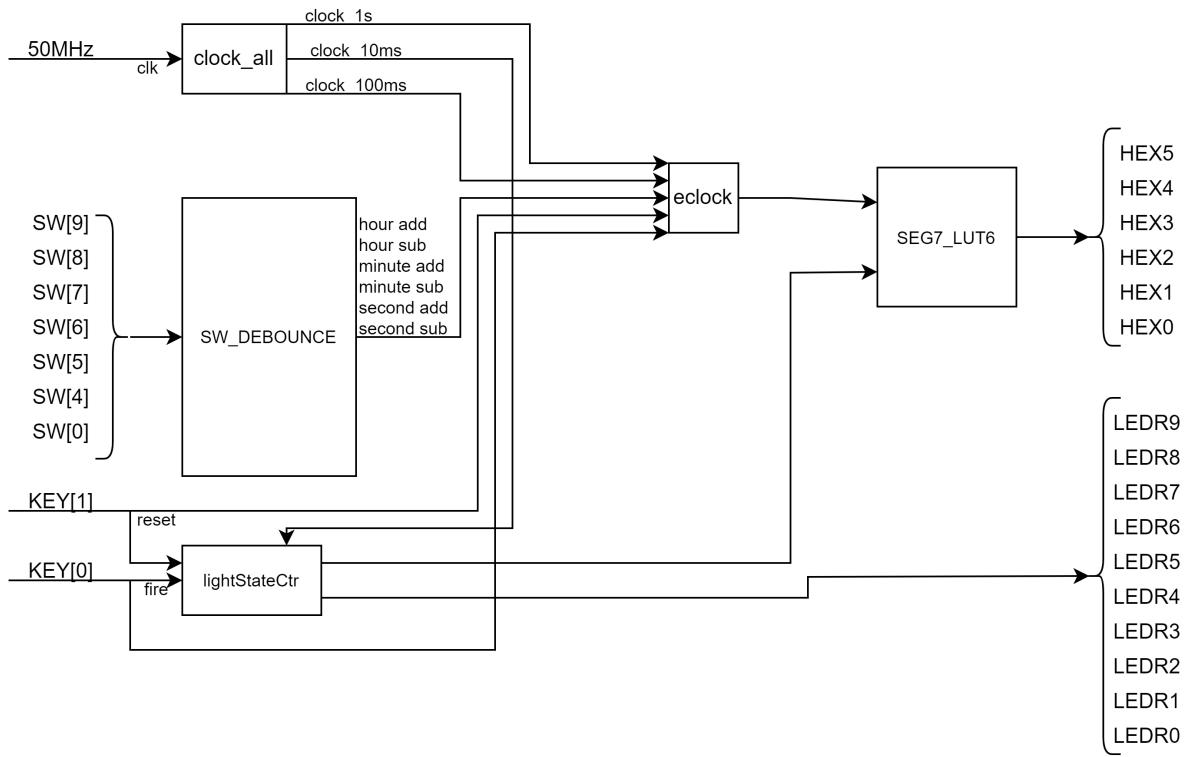


Figure 1: Block Diagram

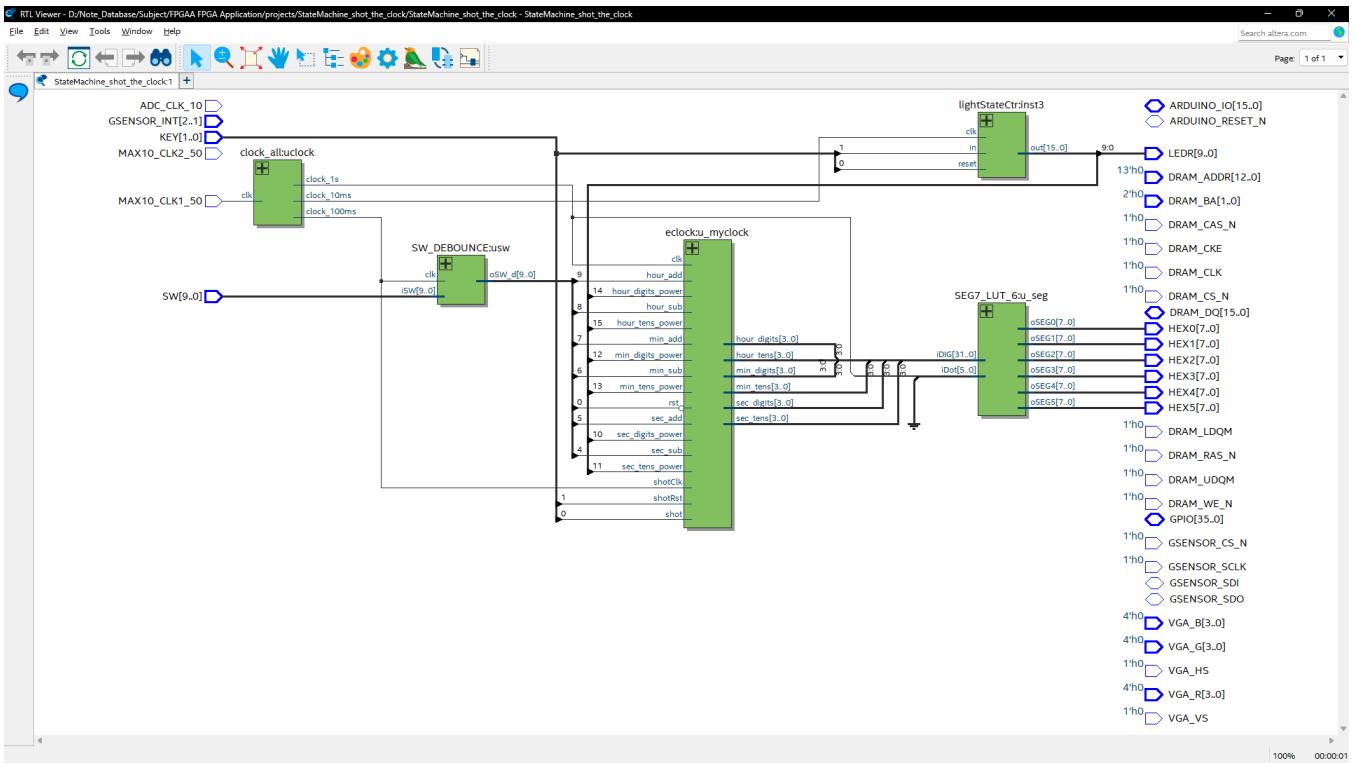


Figure 2: RTL viewer

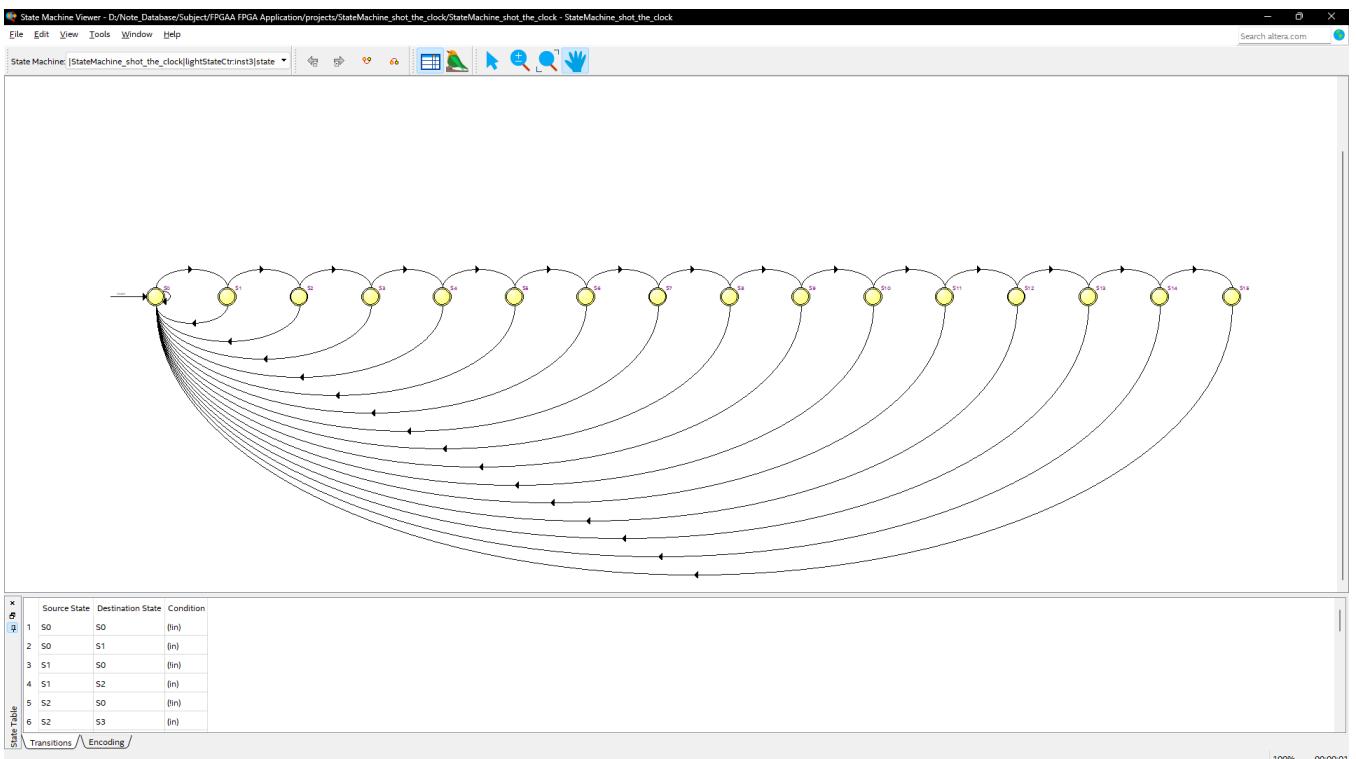


Figure 3: State Machine

4 System Flow

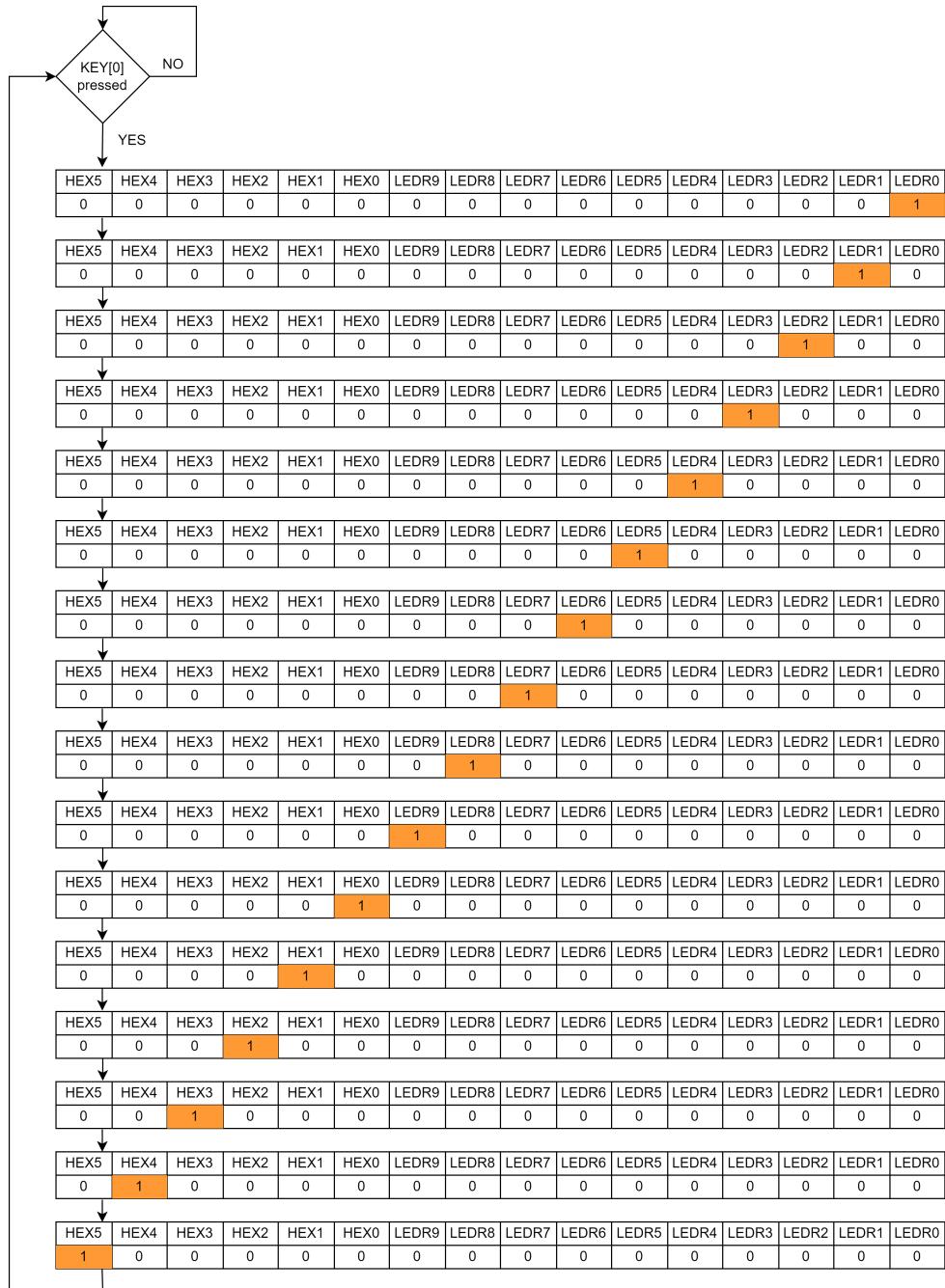


Figure 4: Flow Chart

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.

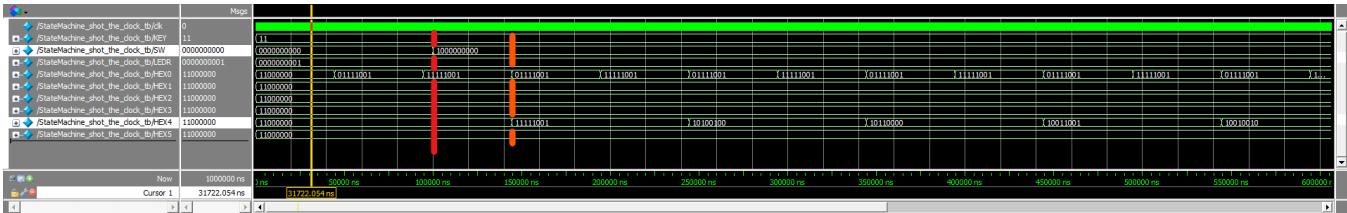


Figure 5: ModelSim clock hour addition

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

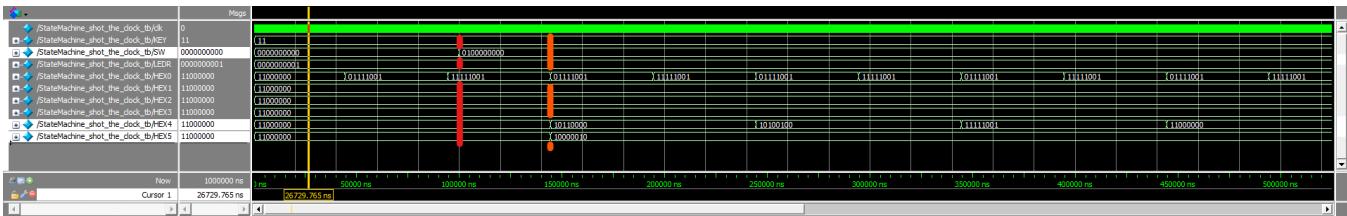


Figure 6: ModelSim clock hour subtraction

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

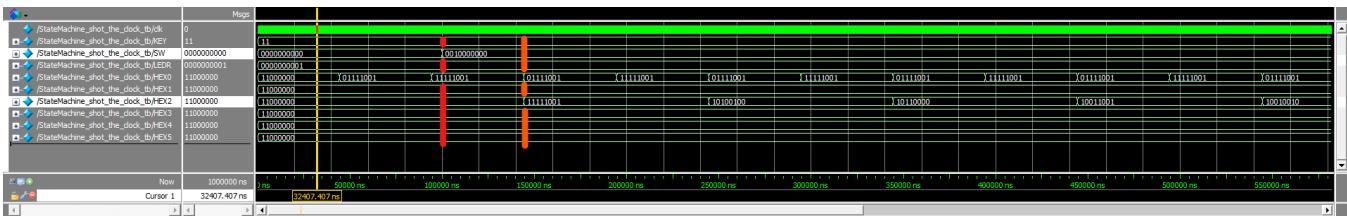


Figure 7: ModelSim clock minute addition

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

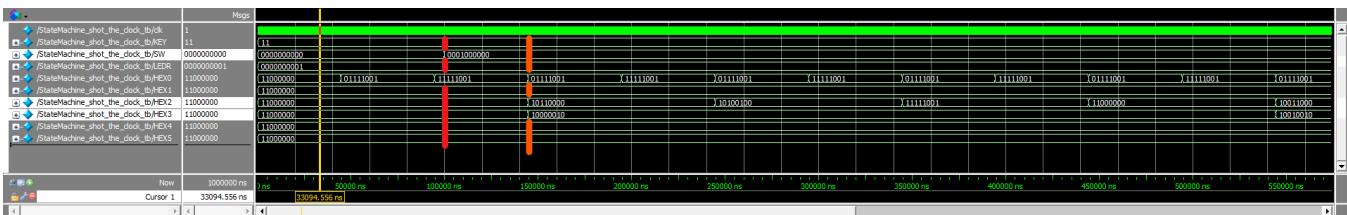


Figure 8: ModelSim clock minute subtraction

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

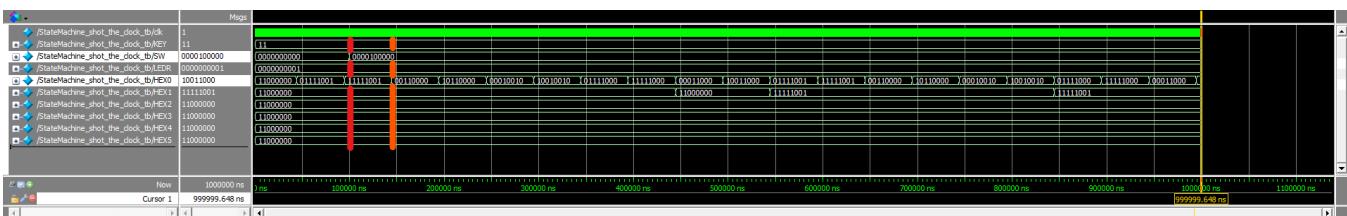


Figure 9: ModelSim clock second addition

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

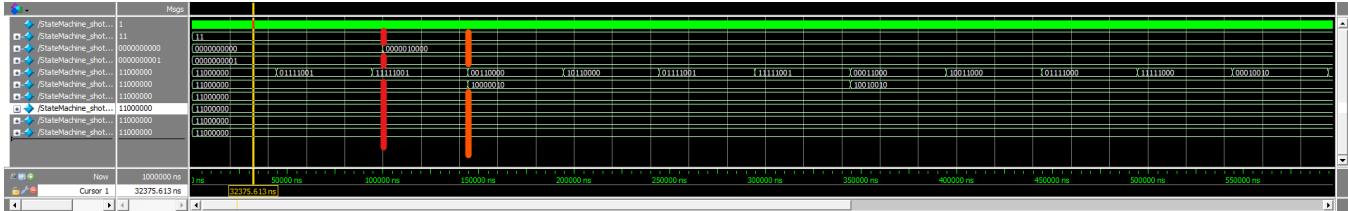


Figure 10: ModelSim clock second subtraction

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.

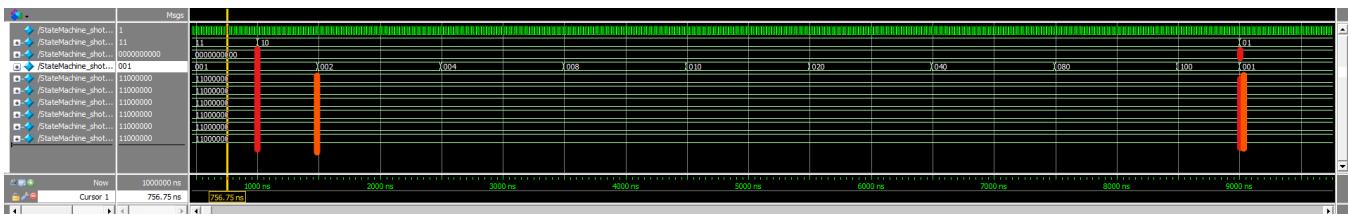


Figure 12: ModelSim meteor light

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.

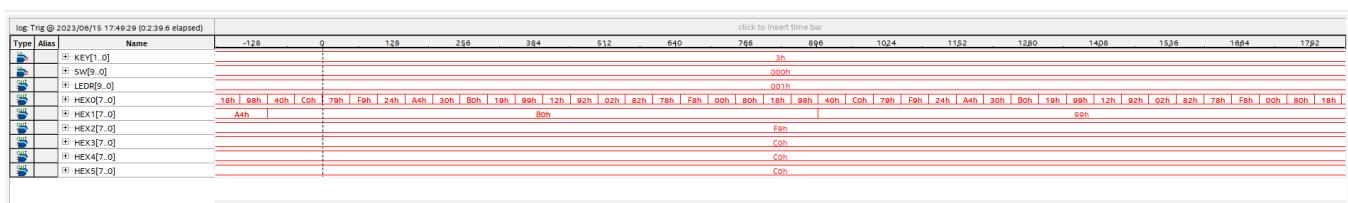


Figure 13: Signal Tap Logic Analyzer



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.

```

StateMachine.shot_the_clock > E eclock.v
1 module eclock(
2   clk,
3   rst,
4   shot,
5   shotClk,
6   shotRst,
7   hour_add,
8   hour_sub,
9   hour_tens,
10  hour_digits,
11  min_add,
12  min_sub,
13  min_tens,
14  min_digits,
15  sec_add,
16  sec_sub,
17  sec_tens,
18  sec_digits,
19  hour_tens_power,
20  hour_digits_power, min_tens_power, min_digits_power, sec_tens_power, sec_digits_power;
21  output [3:0] hour_tens, hour_digits;
22  output [3:0] min_tens, min_digits;
23  output [3:0] sec_tens, sec_digits;
24 );
25
26
27 input  clk, rst, shot, shotClk, shotRst;
28 input  hour_add, hour_sub, min_add, min_sub, sec_add, sec_sub;
29 input  hour_tens_power, hour_digits_power, min_tens_power, min_digits_power, sec_tens_power, sec_digits_power;
30 output [3:0] hour_tens, hour_digits;
31 output [3:0] min_tens, min_digits;
32 output [3:0] sec_tens, sec_digits;
33
34 reg [5:0] count_min, count_hour, count_sec;
35 reg [2:0] count_shot;
36 reg      count_shotLED1, count_shotLED2, count_shotLED3, count_shotLED4, count_shotLED5, count_shotLED6;
37
38 // binary to decimal]
39 assign hour_tens = (hour_tens_power) ? 4'h0 : ((count_shotLED6) ? 4'h1 : count_hour / 10);
40 assign hour_digits = (hour_digits_power) ? 4'h0 : ((count_shotLED5) ? 4'h1 : count_hour % 10);
41 assign min_tens = (min_tens_power) ? 4'h0 : ((count_shotLED4) ? 4'h1 : count_min / 10);
42 assign min_digits = (min_digits_power) ? 4'h0 : ((count_shotLED3) ? 4'h1 : count_min % 10);
43
44
45

```

Figure 15: Module: eclock 1

```

StateMachine.shot_the_clock > E eclock.v
41 assign hour_digits = (hour_digits_power) ? 4'h0 : ((count_shotLED2) ? 4'h1 : count_hour % 10);
42 assign min_digits = (min_digits_power) ? 4'h0 : ((count_shotLED3) ? 4'h1 : count_min / 10);
43 assign sec_digits = (sec_digits_power) ? 4'h0 : ((count_shotLED4) ? 4'h1 : count_sec / 10);
44
45
46 always @ (posedge shot or negedge shotRst)
47 begin
48   if (!shotRst)
49   begin
50     count_shot <= 0;
51   end
52   else
53   begin
54     count_shot <= count_shot + 1;
55   end
56 end
57
58 always @ (posedge shotClk or negedge shotRst)
59 begin
60   if (!shotRst)
61   begin
62     count_shotLED1 <= 0;
63     count_shotLED2 <= 0;
64     count_shotLED3 <= 0;
65     count_shotLED4 <= 0;
66     count_shotLED5 <= 0;
67     count_shotLED6 <= 0;
68   end
69   else if (count_shot == 1)
70   begin
71     count_shotLED1 <= 1;
72   end
73   else if (count_shot == 2)
74   begin
75     count_shotLED2 <= 1;
76   end
77   else if (count_shot == 3)
78   begin
79     count_shotLED3 <= 1;
80   end
81   else if (count_shot == 4)
82

```

Figure 16: Module: eclock 2

```

StateMachine.shot_the_clock > E eclock.v
80
81   end
82   else if (count_shot == 4)
83   begin
84     count_shotLED4 <= 1;
85   end
86   else if (count_shot == 5)
87   begin
88     count_shotLED5 <= 1;
89   end
90   else if (count_shot == 6)
91   begin
92     count_shotLED6 <= 1;
93   end
94
95
96 always @ (posedge clk or negedge rst)
97 begin
98   // reset
99   if (!rst)
100   begin
101     count_hour <= 0;
102     count_min <= 0;
103     count_sec <= 0;
104   end
105   // hour operation
106   else if (hour_add)
107   begin
108     count_hour <= count_hour + 1;
109   end
110   else if (hour_sub)
111   begin
112     count_hour <= count_hour - 1;
113   end
114   // hour maximum
115   else if (count_hour >= 23 && count_min >= 59 && count_sec >= 59 )
116   begin
117     count_min <= 0;
118     count_hour <= 0;
119     count_sec <= 0;
120   end
121   // minute operation
122   else if (min_add)
123

```

Figure 17: Module: eclock 3

```

117     count_hour <= 0;
118     count_sec <= 0;
119   end
120   // minute operation
121   else if (min_add)
122   begin
123     count_min <= count_min + 1;
124   end
125   else if (min_sub)
126   begin
127     count_min <= count_min - 1;
128   end
129   // minute maximum
130   else if (count_min >= 59)
131   begin
132     count_min <= 0;
133     count_hour <= count_hour + 1;
134   end
135   // second operation
136   else if (sec_add)
137   begin
138     count_sec <= count_sec + 2;
139   end
140   else if (sec_sub)
141   begin
142     count_sec <= count_sec - 2;
143   end
144   // second maximum
145   else if (count_sec >= 59)
146   begin
147     count_sec <= 0;
148     count_min <= count_min + 1;
149   end
150   // normal operation
151   else
152     count_sec <= count_sec + 1;
153
154
155 endmodule

```

Figure 18: Module: eclock 4

```

StateMachine_shot_the_clock > E 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:  out = 16'b0000000000000001;
20     S1:  out = 16'b0000000000000010;
21     S2:  out = 16'b0000000000000100;
22     S3:  out = 16'b00000000000001000;
23     S4:  out = 16'b000000000000010000;
24     S5:  out = 16'b0000000000000100000;
25     S6:  out = 16'b00000000000001000000;
26     S7:  out = 16'b000000000000010000000;
27     S8:  out = 16'b0000000000000100000000;
28     S9:  out = 16'b00000000000001000000000;
29     S10: out = 16'b000000000000010000000000;
30     S11: out = 16'b0000000000000100000000000;
31     S12: out = 16'b0000000000000100000000000;
32     S13: out = 16'b00000000000001000000000000;
33 end
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145

```

Figure 19: Module: lightStateCtr 1

```

StateMachine_shot_the_clock > E 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 @ (posedge clk or posedge reset) begin
18   if (reset)
19     state <= S0;
20   else
21     case (state)
22       S0:  if (in)
23             state <= S1;
24           else
25             state <= S0;
26       S1:  if (in)
27             state <= S2;
28           else
29             state <= S0;
30       S2:  if (in)
31             state <= S3;
32           else
33             state <= S0;
34       S3:  if (in)
35             state <= S4;
36           else
37             state <= S0;
38       S4:  if (in)
39             state <= S5;
40           else
41             state <= S0;
42       S5:  if (in)
43             state <= S6;
44           else
45             state <= S0;
46       S6:  if (in)
47             state <= S7;
48           else
49             state <= S0;
50       S7:  if (in)
51             state <= S8;
52           else
53             state <= S0;
54       S8:  if (in)
55             state <= S9;
56           else
57             state <= S0;
58       S9:  if (in)
59             state <= S10;
60           else
61             state <= S0;
62       S10: if (in)
63             state <= S11;
64           else
65             state <= S0;
66       S11: if (in)
67             state <= S12;
68           else
69             state <= S0;
70       S12: if (in)
71             state <= S13;
72           else
73             state <= S0;
74       S13: if (in)
75             state <= S14;
76           else
77             state <= S0;
78       S14: if (in)
79             state <= S15;
80           else
81             state <= S0;
82     endcase
83   end
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145

```

Figure 20: Module: lightStateCtr 2

```

StateMachine_shot_the_clock > E lightStateCtr.v
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145

```

Figure 21: Module: lightStateCtr 3

```

StateMachine_shot_the_clock > E lightStateCtr.v
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145

```

Figure 22: Module: lightStateCtr 4

```

StateMachine_shot_the_clock > E StateMachine_shot_the_clock.v
1
2 //=====
3 // This code is generated by Terasic System Builder
4 //=====
5
6 module StateMachine_shot_the_clock(
7
8     input [10:0] ADC_CLK_10,
9     input MAX10_CLK1_50,
10    input MAX10_CLK2_50,
11
12    output [12:0] DRAM_ADDR,
13    output [1:0] DRAM_BA,
14    output DRAM_CAS_N,
15    output DRAM_CKE,
16    output DRAM_CS_N,
17    output DRAM_DQ,
18    output DRAM_LDM,
19    output DRAM_RAS_N,
20    output DRAM_UDQM,
21    output DRAM_WE_N,
22
23    output [7:0] HEX0,
24    output [7:0] HEX1,
25    output [7:0] HEX2,
26    output [7:0] HEX3,
27    output [7:0] HEX4,
28    output [7:0] HEX5,
29
30    input [1:0] KEY,
31
32    output [9:0] LEDR,
33
34    input [9:0] SW,
35
36);
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155

```

Figure 23: Module: StateMachine_shot_the_clock 1

```

StateMachine_shot_the_clock > E StateMachine_shot_the_clock.v
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15
16
17
18
19
20
21
22
23
24
25
26
27
28
29
30
31
32
33
34
35
36
37
38
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155

```

Figure 24: Module: StateMachine_shot_the_clock 2

```

StateMachine_shot_the_clock > E StateMachine_shot_the_clock.v
80 wire hour_add, hour_sub;
81 wire min_add, min_sub;
82 wire sec_add, sec_sub;
83 wire hour_tens_power, hour_digits_power, min_tens_power, min_digits_power, sec_tens_power, sec_digits_power;
84
85 //=====
86 // Structural coding
87 //=====
88
89 SH_DEBOUNCE usw(
90     .clk(clock_100ms), // for real board
91     .clk(clock_100kHz), // for simulation
92     .iSW(SW),
93     .oSW_d(SW_d)
94 );
95
96
97 clock_all unlock(
98     .clock(MAX10_CLK1_50),
99     .clock_1MHz(clock_1MHz), // simulate 10ms
100    .clock_100kHz(clock_100kHz), // simulate 100ms
101    .clock_10kHz(clock_10kHz), // simulate 1s
102    .clock_100ms(clock_100ms),
103    .clock_1s(clock_1s),
104    .clock_10ms(clock_10ms)
105 );
106
107 eclock my_clock(
108     .clock(clock_10ms), // for real board
109     .clock(clock_10Hz), // for simulation
110     .rst(SW_d[0]),
111     .shot(KEY[0]),
112     .shotClk(clock_100ms), // for real board
113     .shotClk(clock_10kHz), // for simulation
114     .shotRst(KEY[1]),
115     .hour_add(SW_d[9]),
116     .hour_sub(SW_d[8]),
117     .hour_tens(hour_tens),
118     .hour_digits(hour_digits),
119     .min_add(SW_d[7]),
120     .min_sub(SW_d[6]),
121     .min_tens(min_tens),
122     .min_digits(min_digits),
123     .sec_add(SW_d[5]),
124     .sec_sub(SW_d[4]),
125     .sec_tens(sec_tens),
126     .sec_digits(sec_digits),
127     .hour_tens_power(hour_tens_power),
128     .hour_digits_power(hour_digits_power),
129     .min_tens_power(min_tens_power),
130     .min_digits_power(min_digits_power),
131     .sec_tens_power(sec_tens_power),
132     .sec_digits_power(sec_digits_power)
133 );
134
135 SEG7_LUT_6 u_seg(
136     .oSEG0(HEX0),
137     .oSEG1(HEX1),
138     .oSEG2(HEX2),
139     .oSEG3(HEX3),
140     .oSEG4(HEX4),
141     .oSEG5(HEX5),
142     .iDot((hour_tens, hour_digits, min_tens, min_digits, sec_tens, sec_digits )),
143     .iDot((SW_d[7], clock_1s)) // for real board
144     .iDot ({5'b0, clock_10kHz}) // for simulation
145 );
146
147 LightStateCtr inst3 (
148     .clk(clock_10ms), // for real board
149     .clk(clock_1MHz), // for simulation
150     .iN(KEY[1]),
151     .reset(KEY[0]),
152     .oOut(hour_tens_power, hour_digits_power, min_tens_power, min_digits_power, sec_tens_power, sec_digits_power, LEDR)
153 );
154
155 endmodule

```

Figure 25: Module: StateMachine_shot_the_clock 3

```

StateMachine_shot_the_clock > E StateMachine_shot_the_clock.v
114     .hour_add(SW_d[9]),
115     .hour_sub(SW_d[8]),
116     .hour_tens(hour_tens),
117     .hour_digits(hour_digits),
118     .min_add(SW_d[7]),
119     .min_sub(SW_d[6]),
120     .min_tens(min_tens),
121     .min_digits(min_digits),
122     .sec_add(SW_d[5]),
123     .sec_sub(SW_d[4]),
124     .sec_tens(sec_tens),
125     .sec_digits(sec_digits),
126     .hour_tens_power(hour_tens_power),
127     .hour_digits_power(hour_digits_power),
128     .min_tens_power(min_tens_power),
129     .min_digits_power(min_digits_power),
130     .sec_tens_power(sec_tens_power),
131     .sec_digits_power(sec_digits_power)
132 );
133
134 SEG7_LUT_6 u_seg(
135     .oSEG0(HEX0),
136     .oSEG1(HEX1),
137     .oSEG2(HEX2),
138     .oSEG3(HEX3),
139     .oSEG4(HEX4),
140     .oSEG5(HEX5),
141     .iDot((hour_tens, hour_digits, min_tens, min_digits, sec_tens, sec_digits )),
142     .iDot((SW_d[7], clock_1s)) // for real board
143     .iDot ({5'b0, clock_10kHz}) // for simulation
144 );
145
146 LightStateCtr inst3 (
147     .clk(clock_10ms), // for real board
148     .clk(clock_1MHz), // for simulation
149     .iN(KEY[1]),
150     .reset(KEY[0]),
151     .oOut(hour_tens_power, hour_digits_power, min_tens_power, min_digits_power, sec_tens_power, sec_digits_power, LEDR)
152 );
153
154 endmodule

```

Figure 26: Module: StateMachine_shot_the_clock 4

```

StateMachine_shot_the_clock_tb.v
1  `timescale 1ns/1ns
2
3  module StateMachine_shot_the_clock_tb;
4
5    reg clk;
6    reg [1:0] KEY;
7    reg [9:0] SW;
8
9    wire [9:0] LEDR;
10   wire [7:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
11
12  StateMachine_shot_the_clock inst1(
13    .MAX10_CLK1_50(clk),
14    .SM(SM),
15    .KEY(KEY),
16    .LEDR(LEDR),
17    .HEX0(HEX0),
18    .HEX1(HEX1),
19    .HEX2(HEX2),
20    .HEX3(HEX3),
21    .HEX4(HEX4),
22    .HEX5(HEX5)
23  );
24
25 initial begin
26   clk = 0;
27   KEY = 2'b11;
28   SM = 10'`h000000000000;
29   // * simulate for 1ms
30   // * simulate switch
31   // #1000 KEY = 2'b10; // led meteor
32   // #8000 KEY = 2'b01; // led reset
33   // * simulate switch
34   // #100000 SM = 10'`h000000000001; // timer reset
35   // #100000 SM = 10'`h100000000000; // hour add
36   // #100000 SM = 10'`h010000000000; // hour sub
37   // #100000 SM = 10'`h001000000000; // min add
38   // #100000 SM = 10'`h000100000000; // min sub
39   // #100000 SM = 10'`h000010000000; // sec add
40   // #100000 SM = 10'`h000001000000; // sec sub
41   // * simulate for 1s to see all natural increment of clock

```

```

StateMachine_shot_the_clock_tb.v
9   wire [9:0] LEDR;
10  wire [7:0] HEX0, HEX1, HEX2, HEX3, HEX4, HEX5;
11
12 StateMachine_shot_the_clock inst1(
13   .MAX10_CLK1_50(clk),
14   .SM(SM),
15   .KEY(KEY),
16   .LEDR(LEDR),
17   .HEX0(HEX0),
18   .HEX1(HEX1),
19   .HEX2(HEX2),
20   .HEX3(HEX3),
21   .HEX4(HEX4),
22   .HEX5(HEX5)
23 );
24
25 initial begin
26   CLK = 0;
27   KEY = 2'b11;
28   SM = 10'`h000000000000;
29   // * simulate for 1ms
30   // * simulate key
31   // #1000 KEY = 2'b10; // led meteor
32   // #8000 KEY = 2'b01; // led reset
33   // * simulate switch
34   // #100000 SM = 10'`h000000000001; // timer reset
35   // #100000 SM = 10'`h100000000000; // hour add
36   // #100000 SM = 10'`h010000000000; // hour sub
37   // #100000 SM = 10'`h001000000000; // min add
38   // #100000 SM = 10'`h000100000000; // min sub
39   // #100000 SM = 10'`h000010000000; // sec add
40   // #100000 SM = 10'`h000001000000; // sec sub
41   // * simulate for 1s to see all natural increment of clock
42 end
43
44 always begin
45   #10 clk = ~clk;
46 end
47
48 endmodule

```

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/FPGA_Code/blob/main/src/StateMachine_shot_the_clock.qar.