Lab 5 - Timer and Stopwatch

106033233 資工大四 周聖諺 (Sheng-Yen Chou)

2022-03-28

Contents

Lab 5 - Timer and Stopwatch Report

106033233 資工大四 周聖諺 (Sheng-Yen Chou)

Lab 5 - Pre Lab: 40-Second Timer

Design Specification

Source Code

Debounce

Input: rst, clk, push

Output: push_debounced

One-Pulse

Input: rst, clk, push

Output: push_onepulse

Finite State Machine

Input pause, clk, restart

Output is_pause, is_restart

Frequency Divider

Input: rst, clk

Output: clk_out

2-Digit Synchronous Binary Down Counter

Input: rst, clk

Output [7:0]q

LEDs Controller

Input: q, is_pause, is_restart

Output: leds

Extractor

Input: [7:0] x

Output: [3:0] d1, [3:0] d2

Binary To 7-Segment Convertor

Input: [3:0] i

Output: [3:0]P, [7:0]D

7-Segment7 Frequency Divider

Input: clk, rst

Output: clk_out

7-Segment Display

Output: [0:3]d_sel, [7:0]d_out

Input: clk, rst, [7:0]d0, [7:0]d1, [7:0]d2, [7:0]d3

40-Second Timer

Output [15:0]leds, [3:0]D_SEL, [7:0]D_OUT

Input restart, push, rst, clk

Design Implementation

Global Variables The global variables are used across the whole project.

```
// Segment-7 Displayer
`define SEGMENT_7_INPUT_BITS_N 4
`define SEGMENT_7_DISPALY_DIGIT_N 4
`define SEGMENT_7_SEGMENT_N 8
`define SEGMENT_7_NONE `SEGMENT_7_SEGMENT_N'b1111111_1
`define SEGMENT_7_EMPTY `SEGMENT_7_SEGMENT_N'b1111111_0

// Segment-7 Displayer Frequency Divider
`define SEGMENT_7_FREQ_DIV_BITS 30
// 1000 Hz
`define SEGMENT_7_FREQ_DIV_COUNT `FREQ_DIV_BITS'd50000
//`define SEGMENT_7_FREQ_DIV_COUNT `FREQ_DIV_BITS'd2
// 1 Hz
//`define SEGMENT_7_FREQ_DIV_COUNT `FREQ_DIV_BITS'd5000000
`define RST_OFF 1'b1

// BCD Counter
`define BCD_COUNTER_BITS 8
`define BCD_COUNTER_LIMIT `BCD_COUNTER_BITS'd40
`define BCD_COUNTER_ZERO `BCD_COUNTER_BITS'd0
```

```
21
22 // Frequency Divider
23 `define FREQ_DIV_BITS 30
24 // 1 Hz
25 `define FREQ_DIV_COUNT `FREQ_DIV_BITS'd50000000
26
27 // LED control
28 `define LEDS_NUM 16
```

Debounce For each click, the module will delay 4 clock cycle and then raise the debounce pulse. I use 4 registers to represent the delay state and send a pulse while 4 registers are all 1s.

```
1  module debounce (
2      rst,
3      clk,
4      push,
5      push_debounced
6      );
7
8      input rst;
9      input clk;
10     input push;
11     output push_debounced;
12     // declare the outputs
13     reg push_debounced;
14     // declare the shifting registers
15     reg[3:0] push_window;
16
17     always @(posedge clk or posedge rst)
18     begin
19         if (~rst) begin
20             push_window <= 4'b0;
21             push_debounced <= 1'b0;
22         end else begin
23             push_window<={push, push_window[3:1]};
24
25             if (push_window[3:0] == 4'b1111) begin
26                 push_debounced <= 1'b1;
27             end else begin
28                 push_debounced <= 1'b0;
29             end
30         end
31     end
32 endmodule
```

One-Pulse The one-pulse module raise a pulse while the debounce module raises a pulse. As a result, it will produce high voltage while the previous debounce signal is higher than the current de-

bounce signal.

```
module onepulse (
    rst,
    clk,
    push,
    push_onepulse
    );

    input clk, rst;
    input push;
    output push_onepulse;
    // internal registers
    reg push_onepulse_next;
    reg push_debounced_delay;
    reg push_onepulse;
    wire push_debounced;

    debounce U0(.clk(clk), .rst(rst), .push(push), .push_debounced(
        push_debounced));

    always @* begin
        push_onepulse_next = push_debounced & ~push_debounced_delay;
    end

    always @(posedge clk or posedge rst)
    begin
        if (~rst) begin
            push_onepulse <= 1'b0;
            push_debounced_delay <= 1'b0;
        end else begin
            push_onepulse <= push_onepulse_next;
            push_debounced_delay <=push_debounced;
        end
    end
endmodule
```

Finite State Machine The finite state machine controls 2 signals: `is_pause` and `is_restart`. The down counter will pause while the signal `is_pause` is at high voltage and resume to count while the signal is at low voltage. It is controlled independently, so I only need to inverse the signal whenever the button `restart` is pressed.

On the other hand, the down counter will reset to 0 when the signal `is_restart` raises. It only depends on `restart` button, so we only need to inverse the signal whenever the button is clicked.

```
`define STATE_START 0

module fsm(
    is_pause,
```

```
    is_restart,
    pause,
    clk,
    restart
    );

    output is_pause;
    output is_restart;
    input pause;
    input clk;
    input restart;

    reg is_restart;
    reg is_pause;
    reg state;

    initial
    begin
        state = `STATE_START;
    end

    always@(posedge restart)
    begin
        is_restart = is_restart ^ 1;
    end

    always@(posedge pause)
    begin
          if(state == `STATE_PAUSE)
          begin
            state = `STATE_START;
            is_pause = 0;
          end
          else if(state == `STATE_START)
          begin
            state = `STATE_PAUSE;
            is_pause = 1;
          end
    end
endmodule
```

Frequency Divider To generate the 1 Hz clock, I use variables counter_in and counter_out to count from 0 to 50M. The counter_in will store the value for the next time step and pass the value to the counter_out when the clock raises. The reason why we need 50M counting is each counting is triggered only when the clock raises, so the circuit will count 1 more for every twice clock pulses.

```
`include "global.v"

//`define FREQ_DIV_BITS 30
```

```
4  //`define FREQ_DIV_COUNT `FREQ_DIV_BITS'd1000000
5  //`define FREQ_DIV_COUNT `FREQ_DIV_BITS'd50000000
6
7  module frequency_divider(
8      clk_out,
9      clk,
10     rst
11     );
12
13     input clk;
14     input rst;
15     output clk_out;
16
17     reg clk_in;
18     reg clk_out;
19     reg [`FREQ_DIV_BITS-1:0] counter_in;
20     reg [`FREQ_DIV_BITS-1:0] counter_out;
21
22     always@(counter_out or clk_out)
23         if(counter_out < (`FREQ_DIV_COUNT - 1))
24         begin
25             counter_in <= counter_out + `FREQ_DIV_BITS'd1;
26             clk_in <= clk_out;
27         end
28         else
29         begin
30             counter_in <= `FREQ_DIV_BITS'd0;
31             clk_in <= ~clk_out;
32         end
33
34     always@(posedge clk or negedge rst)
35         if(~rst)
36         begin
37             counter_out <= `FREQ_DIV_BITS'd0;
38             clk_out <= 1'd0;
39         end
40         else
41         begin
42             counter_out <= counter_in;
43             clk_out <= clk_in;
44         end
45 endmodule
```

2-Digit Synchronous Binary Down Counter To implement the binary down counter, I use a variable q_in to count from 0 to 15. Whenever the output of the counter q changes, the variable q_in should be changed to q - 1. In addition, when the circuit detects the raise of the clock, the output of the counter will change to the variable q_in. On the other hand, if the reset switch to 0 or the counter hits 0, q will be reset to the upper limit 15.

Verilog Code

```
`include "global.v"

module binary_down_2digit_counter(
    q,
    clk,
    rst,
    is_pause
    );

    output [`BCD_COUNTER_BITS-1:0]q;
    input clk;
    input rst;
    input is_pause;

    reg [`BCD_COUNTER_BITS-1:0]q;
    reg [`BCD_COUNTER_BITS-1:0]q_in;

    initial
    begin
        q <= `BCD_COUNTER_LIMIT;
    end

    always@(q)
    begin
//        if(q == (`BCD_COUNTER_LIMIT - `BCD_COUNTER_BITS'd1))
        if(q <= `BCD_COUNTER_BITS'd1)
        begin
            q_in <= `BCD_COUNTER_ZERO;
        end
        else
        begin
            if(~is_pause)
            begin
                q_in <= q - `BCD_COUNTER_BITS'd1;
            end
            else
            begin
                q_in <= q;
            end
        end
    end

    always@(posedge clk or negedge rst)
    begin
        if(~rst)
        begin
            q <= `BCD_COUNTER_LIMIT;
        end
        else
```

```
50          begin
51              q <= q_in;
52          end
53      end
54  endmodule
```

LEDs Controller The LEDs will all light up whenever the the module counts to 0. In addition, `LED[0]` lights up when the counter is counting and goes out when the counter is paused.

```
1   `include "global.v"
2
3   module led_control(
4       q,
5       is_pause,
6       is_restart,
7       leds
8       );
9
10      input [`BCD_COUNTER_BITS-1:0]q;
11      input is_pause;
12      input is_restart;
13      output [`LEDS_NUM-1:0]leds;
14
15      reg [`LEDS_NUM-1:0]leds;
16
17      always@(q or is_pause or is_restart)
18      begin
19          if(q == `BCD_COUNTER_ZERO)
20          begin
21              leds = {`LEDS_NUM{1'b1}};
22          end
23          else if((!is_pause) && (is_restart))
24          begin
25              leds = `LEDS_NUM'b1;
26          end
27          else
28          begin
29              leds = `LEDS_NUM'b0;
30          end
31      end
32  endmodule
```

Extractor I use mod of 10 to extract the first decimal digit and use divided by 10 to extract the second decimal digit.

```
1   `include "global.v"
2
```

```
3  module extract(
4      input [`BCD_COUNTER_BITS-1:0] x,
5      output [`SEGMENT_7_INPUT_BITS_N-1:0] d1,
6      output [`SEGMENT_7_INPUT_BITS_N-1:0] d2
7      );
8
9      wire [`BCD_COUNTER_BITS-1:0]mod;
10     wire [`BCD_COUNTER_BITS-1:0]div;
11     assign mod = x % 10;
12     assign div = x / 10;
13
14     assign d1 = mod[`SEGMENT_7_INPUT_BITS_N-1:0];
15     assign d2 = div[`SEGMENT_7_INPUT_BITS_N-1:0];
16 endmodule
```

Binary To 7-Segment Convertor Convert 4-bit binary number to 7-segment display with switch-case syntax.

```
1  `include "global.v"
2
3  module segment7(
4      i,
5      P,
6      D
7      );
8
9      input [`SEGMENT_7_INPUT_BITS_N:0] i;
10     output [`SEGMENT_7_DISPALY_DIGIT_N-1:0]P;
11     output [`SEGMENT_7_SEGMENT_N-1:0]D;
12
13     reg [`SEGMENT_7_SEGMENT_N-1:0]D;
14
15     assign P = ~4'b0001;
16     always@(i)
17         case(i)
18             `SEGMENT_7_INPUT_BITS_N'd0: D=`SEGMENT_7_SEGMENT_N'
                   b0000001_1;
19             `SEGMENT_7_INPUT_BITS_N'd1: D=`SEGMENT_7_SEGMENT_N'
                   b1001111_1;
20             `SEGMENT_7_INPUT_BITS_N'd2: D=`SEGMENT_7_SEGMENT_N'
                   b0010010_1;
21             `SEGMENT_7_INPUT_BITS_N'd3: D=`SEGMENT_7_SEGMENT_N'
                   b0000110_1;
22             `SEGMENT_7_INPUT_BITS_N'd4: D=`SEGMENT_7_SEGMENT_N'
                   b1001100_1;
23             `SEGMENT_7_INPUT_BITS_N'd5: D=`SEGMENT_7_SEGMENT_N'
                   b0100100_1;
24             `SEGMENT_7_INPUT_BITS_N'd6: D=`SEGMENT_7_SEGMENT_N'
                   b0100000_1;
```

```
25              `SEGMENT_7_INPUT_BITS_N'd7: D=`SEGMENT_7_SEGMENT_N'
                   b0001111_1;
26              `SEGMENT_7_INPUT_BITS_N'd8: D=`SEGMENT_7_SEGMENT_N'
                   b0000000_1;
27              `SEGMENT_7_INPUT_BITS_N'd9: D=`SEGMENT_7_SEGMENT_N'
                   b0000100_1;
28              `SEGMENT_7_INPUT_BITS_N'd10: D=`SEGMENT_7_SEGMENT_N'
                   b0001000_1;
29              `SEGMENT_7_INPUT_BITS_N'd11: D=`SEGMENT_7_SEGMENT_N'
                   b1100000_1;
30              `SEGMENT_7_INPUT_BITS_N'd12: D=`SEGMENT_7_SEGMENT_N'
                   b0110001_1;
31              `SEGMENT_7_INPUT_BITS_N'd13: D=`SEGMENT_7_SEGMENT_N'
                   b1000010_1;
32              `SEGMENT_7_INPUT_BITS_N'd14: D=`SEGMENT_7_SEGMENT_N'
                   b0110000_1;
33              `SEGMENT_7_INPUT_BITS_N'd15: D=`SEGMENT_7_SEGMENT_N'
                   b0111000_1;
34              default: D=`SEGMENT_7_SEGMENT_N'b0111000_1;
35          endcase
36
37  endmodule
```

7-Segment Display Since we can only control one digit of the 7-segment display each time, I design a module that takes the 4-digit patterns as input and shows the 1 digit on the display when the clock raises. Whenever the clock raises, the module will switch the control d_sel to different digit and shows the corresponding digit. Take an example, when the first clock raise occur, the module will set d_sel = $4'$ b1110 and d_out = d0. As for second clock pulse, the module will output d_sel = $4'$ b1101 and d_out = d1 and so on.

```
1   `include "global.v"
2
3   //`define DIGIT_N 4
4   //`define SEGMENT_N 8
5
6
7   module display_7seg(
8       d_sel,
9       d_out,
10      clk,
11      rst,
12      d0,
13      d1,
14      d2,
15      d3
16      );
17
18      output [0:`SEGMENT_7_DISPALY_DIGIT_N-1]d_sel;
```

```
19      output [`SEGMENT_7_SEGMENT_N-1:0]d_out;
20      input clk;
21      input rst;
22      input [`SEGMENT_7_SEGMENT_N-1:0]d0;
23      input [`SEGMENT_7_SEGMENT_N-1:0]d1;
24      input [`SEGMENT_7_SEGMENT_N-1:0]d2;
25      input [`SEGMENT_7_SEGMENT_N-1:0]d3;
26
27      reg [0:`SEGMENT_7_DISPALY_DIGIT_N-1]d_sel;
28      reg [`SEGMENT_7_SEGMENT_N-1:0]d_out;
29      reg [0:`SEGMENT_7_DISPALY_DIGIT_N-1]d_sel_temp;
30      reg [`SEGMENT_7_SEGMENT_N-1:0]d_out_temp;
31      wire clk_out;
32
33  //    initial
34  //    begin
35  //        d_sel <= `SEGMENT_7_DISPALY_DIGIT_N'b1110;
36  //        d_out <= `SEGMENT_7_EMPTY;
37  //    end
38
39      segment7_frequency_divider U0(.clk(clk), .rst(`RST_OFF), .clk_out(
            clk_out));
40
41      always@(d_sel)
42      begin
43          case((d_sel << 1) | (d_sel >> (`SEGMENT_7_DISPALY_DIGIT_N-1)))
44              `SEGMENT_7_DISPALY_DIGIT_N'b1110: d_out_temp <= d0;
45              `SEGMENT_7_DISPALY_DIGIT_N'b1101: d_out_temp <= d1;
46              `SEGMENT_7_DISPALY_DIGIT_N'b1011: d_out_temp <= d2;
47              `SEGMENT_7_DISPALY_DIGIT_N'b0111: d_out_temp <= d3;
48              default: d_out_temp <= `SEGMENT_7_NONE;
49          endcase
50          d_sel_temp <= (d_sel << 1) | (d_sel >> (
                `SEGMENT_7_DISPALY_DIGIT_N-1));
51      end
52
53      always@(posedge clk_out or negedge rst)
54      begin
55          if(~rst)
56          begin
57              d_out <= `SEGMENT_7_EMPTY;
58              d_sel <= `SEGMENT_7_DISPALY_DIGIT_N'b1110;
59  //            d_sel <= d_sel_temp;
60          end
61          else
62          begin
63              d_out <= d_out_temp;
64              d_sel <= d_sel_temp;
65          end
66      end
67
```

```
68 endmodule
```

40-Second Timer

```
1  `include "global.v"
2
3  //`define BCD_COUNTER_BITS 8
4  //`define RST_HIGH 1'b1
5
6  //`define INPUT_BITS_N 4
7  //`define SEGMENT_7_DISPALY_DIGIT_N 4
8  //`define SEGMENT_7_SEGMENT_N 8
9
10 //`define P 4'b1111
11 //`define NONE_SEG7 `SEGMENT_7_SEGMENT_N'b1111111_1
12
13 module prelab(
14     leds,
15     D_SEL,
16     D_OUT,
17     restart,
18     push,
19     rst,
20     clk
21     );
22     output [`LEDS_NUM-1:0]leds;
23     output [`SEGMENT_7_DISPALY_DIGIT_N-1:0]D_SEL;
24     output [`SEGMENT_7_SEGMENT_N-1:0]D_OUT;
25     input restart;
26     input push;
27     input rst;
28     input clk;
29
30     //    reg [`BCD_COUNTER_BITS-1:0]q;
31     wire DIV_CLK;
32     wire [`SEGMENT_7_INPUT_BITS_N-1:0]D1_BINARY;
33     wire [`SEGMENT_7_INPUT_BITS_N-1:0]D2_BINARY;
34     wire [`SEGMENT_7_SEGMENT_N-1:0]D1_SEGMENT7;
35     wire [`SEGMENT_7_SEGMENT_N-1:0]D2_SEGMENT7;
36     wire PAUSE_ONEPULSE;
37     wire RESTART_ONEPULSE;
38     wire IS_PAUSE;
39     wire IS_RESTART;
40
41     wire [`BCD_COUNTER_BITS-1:0]Q;
42
43 //    assign IS_RESTART = 1;
44 //    assign IS_PAUSE = 0;
45     onepulse PauseBtn(.rst(rst), .clk(clk), .push(push), .push_onepulse
       (PAUSE_ONEPULSE));
46     onepulse RestartBtn(.rst(rst), .clk(clk), .push(restart), .
```

```
        push_onepulse(RESTART_ONEPULSE));
47      fsm FSM(.clk(clk), .pause(PAUSE_ONEPULSE), .restart(
        RESTART_ONEPULSE), .is_pause(IS_PAUSE), .is_restart(IS_RESTART))
        ;
48
49      // 2-Digits Binary up counter
50      frequency_divider U0(.clk(clk), .rst(IS_RESTART), .clk_out(DIV_CLK)
        );
51      binary_down_2digit_counter U1(.clk(DIV_CLK), .rst(IS_RESTART), .
        is_pause(IS_PAUSE), .q(Q));
52
53      // LEDs controller
54      led_control LED_CONTROL(.q(Q), .is_pause(IS_PAUSE), .is_restart(
        IS_RESTART), .leds(leds));
55
56      // Extract digits
57      extract U2(.x(Q), .d1(D1_BINARY), .d2(D2_BINARY));
58
59      // Convert binary to 7-segment
60      segment7 U3(.i(D1_BINARY), .D(D1_SEGMENT7));
61      segment7 U4(.i(D2_BINARY), .D(D2_SEGMENT7));
62
63      // Show
64      display_7seg(.clk(clk), .rst(rst), .d0(D1_SEGMENT7), .d1(
        D2_SEGMENT7), .d2(`SEGMENT_7_NONE), .d3(`SEGMENT_7_NONE), .d_sel
        (D_SEL), .d_out(D_OUT));
65  //    display_hex_7seg (.clk(clk), .rst(rst), .d0(D1_BINARY), .d1(
    D2_BINARY));
66  endmodule
```

I/O Pin Assignment

I/O	clk	rst	push	restart	D_SEL[0]	D_SEL[1]	D_SEL[2]	D_SEL[3]
LOC	W5	V17	T17	W19	U2	U4	V4	W4

I/O	D_OUT[0]	D_OUT[1]	D_OUT[2]	D_OUT[3]	D_OUT[4]	D_OUT[5]	D_OUT[6]	D_OUT[7]
LOC	V7	U7	V5	U5	V8	U8	W6	W7

I/O	leds[0]	leds[1]	leds[2]	leds[3]	leds[4]	leds[5]	leds[6]	leds[7]
LOC	U16	E19	U19	V19	W18	U15	U14	V14

I/O	leds[8]	leds[9]	leds[10]	leds[11]	leds[12]	leds[13]	leds[14]	leds[15]
LOC	V13	V3	W3	U3	P3	N3	P1	L1

Block Diagram

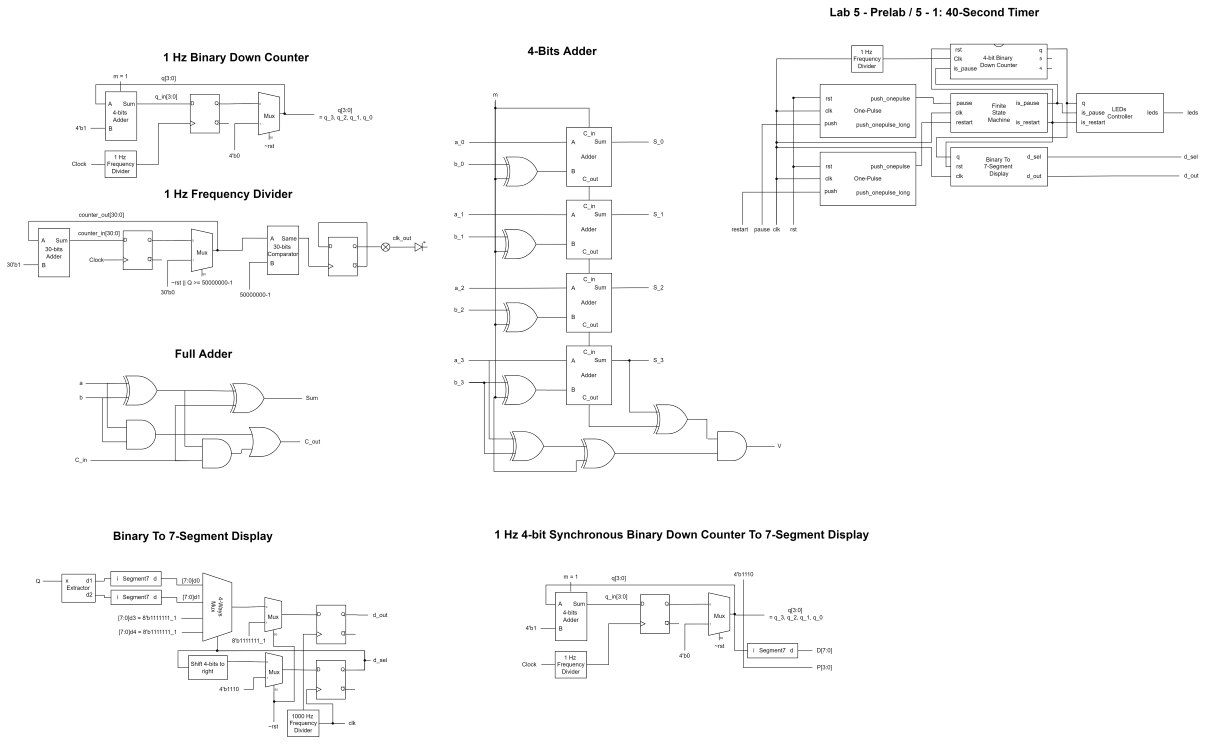

Figure 1: Lab 5-pre Logic Diagram

RTL Simulation

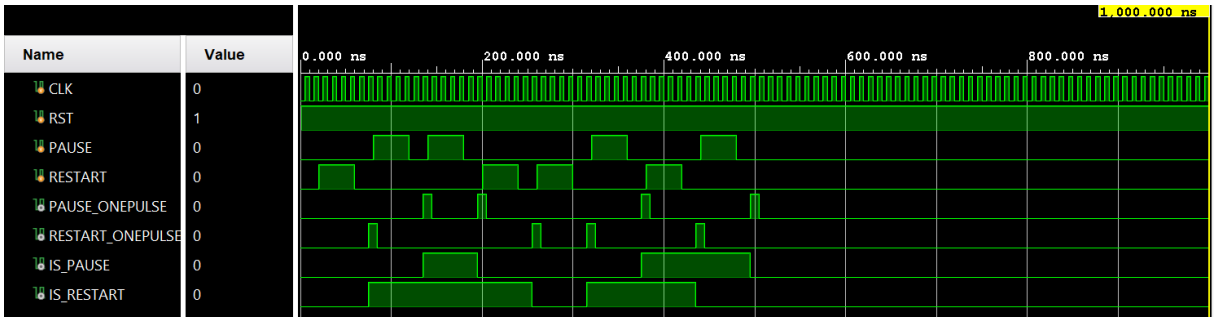

Figure 2: Lab 5-pre RTL Simulation

Lab 5-1: 40-Second Down Counter

Because I′ ve finished all requirements in the prelab, please refers to the section Lab5-Prelab.

Lab 5 - 2: Stopwatch

Design Specification

Source Code

Debounce

Input: rst, clk, push

Output: push_debounced

One-Pulse

Input: rst, clk, push

Output: push_onepulse

Finite State Machine

Input pause, clk, restart, lap

Output is_pause, is_restart, is_lap

Lap Controller

Output [11:0]q

Input [11:0]i, is_lap

Frequency Divider

Input: rst, clk

Output: clk_out

2-Digit Synchronous Binary Up Counter

Input: rst, clk

Output [11:0]q

LEDs Controller

Input: q, is_pause, is_restart, is_lap

Output: [15:0]leds

Minute-Second Separator

Input [11:0]i

Output [7:0]d0_min, [7:0]d1_min, [7:0]d0_sec, [7:0]d1_sec

Binary To 7-Segment Convertor

Input: [3:0] i

Output: [3:0]P, [7:0]D

7-Segment7 Frequency Divider

Input: clk, rst

Output: clk_out

7-Segment Display

Output: [0:3]d_sel, [7:0]d_out

Input: clk, rst, [7:0]d0, [7:0]d1, [7:0]d2, [7:0]d3

7-Segment Time Display

Output [3:0]d_sel, [7:0]d_out;

Input [11:0]q_lap, clk, rst

Stopwatch

Output [15:0]leds, [3:0]D_SEL, [7:0]D_OUT

Input restart, push, rst, clk

Design Implementation

Global Variables

```
// Segment-7 Displayer
`define SEGMENT_7_INPUT_BITS_N 4
`define SEGMENT_7_DISPALY_DIGIT_N 4
`define SEGMENT_7_SEGMENT_N 8
`define SEGMENT_7_NONE `SEGMENT_7_SEGMENT_N'b1111111_1
`define SEGMENT_7_EMPTY `SEGMENT_7_SEGMENT_N'b1111111_0

// @Segment-7 Displayer Frequency Divider
`define SEGMENT_7_FREQ_DIV_BITS 30
// 1000 Hz
`define SEGMENT_7_FREQ_DIV_COUNT `FREQ_DIV_BITS'd50000
//`define SEGMENT_7_FREQ_DIV_COUNT `FREQ_DIV_BITS'd2
// 1 Hz
//`define SEGMENT_7_FREQ_DIV_COUNT `FREQ_DIV_BITS'd5000000
`define RST_OFF 1'b1

// @BCD Counter
`define BCD_COUNTER_BITS 12
`define BCD_COUNTER_LIMIT `BCD_COUNTER_BITS'd3600
`define BCD_COUNTER_ZERO `BCD_COUNTER_BITS'd0

// @Frequency Divider
`define FREQ_DIV_BITS 30
// 1 Hz
`define FREQ_DIV_COUNT `FREQ_DIV_BITS'd50000000
//`define FREQ_DIV_COUNT `FREQ_DIV_BITS'd2

// @LED control
`define LEDS_NUM 16

// @Minute-Second Seperater
`define UNIT_BITS 6

// One Pulse
`define COUNTER_BITS_N 30
`define PRESS_CYCLE_N `COUNTER_BITS_N'd200000000
//`define PRESS_CYCLE_N `COUNTER_BITS_N'd2
```

Debounce Same as lab 5-Prelab.

One-Pulse It's similar to the `one-pulse` module in lab 5-Prelab. I use a counter to count the clock cycles during the button is pressed. If the counting exceed a threshold, it will trigger a long press pulse `push_onepulse_long`. Otherwise, it will trigger a click pulse `push_onepulse`.

```
`include "global.v"
```

```
2
3  //`define LONG_PRESS_N 3
4  //`define COUNTER_BITS_N 30
5
6  module onepulse (
7      rst,
8      clk,
9      clk_long,
10     push,
11     push_onepulse,
12     push_onepulse_long,
13     push_debounced,
14     push_sig,
15     push_sig_long
16     );
17
18     input clk, clk_long, rst;
19     input push;
20     output push_onepulse;
21     output push_onepulse_long;
22     output push_debounced;
23     output push_sig;
24     output push_sig_long;
25     // internal registers
26     reg push_onepulse_next;
27     reg push_debounced_delay;
28     reg push_onepulse;
29
30     reg push_onepulse_next_long;
31     reg push_debounced_delay_long;
32     reg push_onepulse_long;
33
34     reg [`COUNTER_BITS_N-1:0]counter_temp;
35     reg [`COUNTER_BITS_N-1:0]counter;
36     reg push_sig;
37     reg push_sig_long;
38
39     debounce U0(.clk(clk), .rst(rst), .push(push), .push_debounced(
           push_debounced));
40 //    debounce U1(.clk(clk_long), .rst(rst), .push(push), .
       push_debounced(push_debounced_long));
41
42     initial begin
43         push_onepulse = 1'b0;
44         push_onepulse_long = 1'b0;
45     end
46
47     always@(*) begin
48         push_onepulse = push_sig;
49         push_onepulse_long = push_sig_long;
50     end
```

```
51
52      // Switching long or short
53      always@(counter) begin
54          counter_temp <= counter + `COUNTER_BITS_N'b1;
55      end
56
57      always @(posedge clk or negedge rst) begin
58          if (~rst) begin
59              counter <= `COUNTER_BITS_N'b0;
60              push_sig_long <= 1'b0;
61              push_sig <= 1'b0;
62          end else if(~push_debounced) begin
63               if(counter > `PRESS_CYCLE_N) begin
64                  push_sig <= 1'b0;
65                  push_sig_long <= 1'b1;
66               end else if(counter > 0) begin
67                  push_sig <= 1'b1;
68                  push_sig_long <= 1'b0;
69               end else begin
70                  push_sig <= 1'b0;
71                  push_sig_long <= 1'b0;
72               end
73               counter <= `COUNTER_BITS_N'b0;
74          end else if(push_debounced) begin
75              counter <= counter_temp;
76              push_sig <= 1'b0;
77              push_sig_long <= 1'b0;
78          end
79      end
80  endmodule
```

Finite State Machine It's similar to the finite state machine in the Lab 5-Prelab. I just add a new control signal `is_lap`. Whenever the button `lap` is clicked, the signal `is_lap` will be inverted.

```
1   `define STATE_LAP 2
2   `define STATE_PAUSE 1
3   `define STATE_START 0
4
5   module fsm(
6       is_pause,
7       is_restart,
8       is_lap,
9       clk,
10      pause,
11      restart,
12      lap
13      );
14
15      output is_pause;
```

```
16      output is_restart;
17      output is_lap;
18      input clk;
19      input pause;
20      input restart;
21      input lap;
22
23      reg is_lap;
24      reg is_restart;
25      reg is_pause;
26      reg state;
27
28      initial begin
29          state = `STATE_START;
30      end
31
32      always@(posedge restart) begin
33          is_restart <= is_restart ^ 1;
34      end
35
36      always@(posedge lap) begin
37          is_lap <= ~is_lap;
38      end
39
40      always@(posedge pause) begin
41        if(state == `STATE_PAUSE) begin
42          state = `STATE_START;
43          is_pause = 0;
44        end
45        else if(state == `STATE_START) begin
46          state = `STATE_PAUSE;
47          is_pause = 1;
48        end
49      end
50  endmodule
```

Lap Controller In stopwatch, we need to freeze the display in the lap. The lap controller will stop updating the latest counting while `is_lap` is **`true`**.

```
1   `include "global.v"
2
3   module lap_controller(
4       q,
5       i,
6       is_lap
7       );
8
9       output [`BCD_COUNTER_BITS-1:0]q;
10      input [`BCD_COUNTER_BITS-1:0]i;
```

```
    input is_lap;

    reg [`BCD_COUNTER_BITS-1:0]q;

    initial begin
        q <= i;
    end

    always@(is_lap or i) begin
        if(!is_lap) begin
            q <= i;
        end
    end
endmodule
```

Frequency Divider Same as lab 5-Prelab

2-Digit Synchronous Binary Up Counter It' s similar to the down counter in the Lab 5-Prelab. The only different thing is this counter counts up rather than down.

```
`include "global.v"

module binary_up_4digit_counter(
    q,
    clk,
    rst,
    is_pause
    );

    output [`BCD_COUNTER_BITS-1:0]q;
    input clk;
    input rst;
    input is_pause;

    reg [`BCD_COUNTER_BITS-1:0]q;
    reg [`BCD_COUNTER_BITS-1:0]q_in;

    initial
    begin
//        q <= `BCD_COUNTER_LIMIT;
        q <= `BCD_COUNTER_ZERO;
    end

    always@(q or is_pause)
    begin
//        if(q == (`BCD_COUNTER_LIMIT - `BCD_COUNTER_BITS'd1))
        if(q >= `BCD_COUNTER_LIMIT)
        begin
```

```
//            q_in <= `BCD_COUNTER_ZERO;
            q_in <= `BCD_COUNTER_LIMIT;
        end
        else
        begin
            if(~is_pause)
            begin
                q_in <= q + `BCD_COUNTER_BITS'd1;
            end
            else
            begin
                q_in <= q;
            end
        end
    end

    always@(posedge clk or negedge rst)
    begin
        if(~rst)
        begin
//            q <= `BCD_COUNTER_LIMIT;
            q <= `BCD_COUNTER_ZERO;
        end
        else
        begin
            q <= q_in;
        end
    end
endmodule
```

LEDs Controller It's similar to the LEDs controller in the Lab 5-Prelab. The LEDs will all light up while the counter hit the limit. On the other hand, while the counter is counting, `leds[0]` will be turned on. While the stopwatch enters a lap, `leds[1]` will be turned on.

```
`include "global.v"

module led_control(
    q,
    is_pause,
    is_restart,
    is_lap,
    leds
    );

    input [`BCD_COUNTER_BITS-1:0]q;
    input is_pause;
    input is_restart;
    input is_lap;
    output [`LEDS_NUM-1:0]leds;
```

```
16
17      reg [`LEDS_NUM-1:0]leds;
18
19      always@(q or is_pause or is_restart or is_lap) begin
20          if(q == `BCD_COUNTER_LIMIT)begin
21              leds = {`LEDS_NUM{1'b1}};
22          end else begin
23              leds[`LEDS_NUM-1:2] = {(`LEDS_NUM-2){1'b0}};
24              if((!is_pause) && (is_restart)) begin
25                  leds[0] = 1'b1;
26              end else begin
27                  leds[0] = 1'b0;
28              end
29
30              if(is_lap) begin
31                  leds[1] = 1'b1;
32              end else begin
33                  leds[1] = 1'b0;
34              end
35          end
36      end
37  endmodule
```

Minute-Second Separator This module converts the time in seconds to the form of minutes and seconds. I extract the minute unit with dividing to 60 and the seconds unit with taking the modulo to 60. Finally, we can separate the digits with similar ways.

```
1   `include "global.v"
2
3   module min_sec_seperate(
4       i,
5       d0_min,
6       d1_min,
7       d0_sec,
8       d1_sec
9       );
10
11      input [`BCD_COUNTER_BITS-1:0]i;
12      output [`SEGMENT_7_INPUT_BITS_N-1:0]d0_min;
13      output [`SEGMENT_7_INPUT_BITS_N-1:0]d1_min;
14      output [`SEGMENT_7_INPUT_BITS_N-1:0]d0_sec;
15      output [`SEGMENT_7_INPUT_BITS_N-1:0]d1_sec;
16
17      wire [`BCD_COUNTER_BITS-1:0]MINS;
18      wire [`BCD_COUNTER_BITS-1:0]SECS;
19      wire [`BCD_COUNTER_BITS-1:0]D0_MIN;
20      wire [`BCD_COUNTER_BITS-1:0]D1_MIN;
21      wire [`BCD_COUNTER_BITS-1:0]D0_SEC;
22      wire [`BCD_COUNTER_BITS-1:0]D1_SEC;
```

```
23
24      assign MINS = i / 60;
25      assign SECS = i % 60;
26
27      assign D0_MIN = MINS % 10;
28      assign D1_MIN = MINS / 10;
29      assign D0_SEC = SECS % 10;
30      assign D1_SEC = SECS / 10;
31
32      assign d0_min = D0_MIN[`SEGMENT_7_INPUT_BITS_N-1:0];
33      assign d1_min = D1_MIN[`SEGMENT_7_INPUT_BITS_N-1:0];
34      assign d0_sec = D0_SEC[`SEGMENT_7_INPUT_BITS_N-1:0];
35      assign d1_sec = D1_SEC[`SEGMENT_7_INPUT_BITS_N-1:0];
36  endmodule
```

Binary To 7-Segment Convertor Same as lab 5-Prelab

7-Segment Display Same as lab 5-Prelab

7-Segment Time Display This module shows the time on the 7-segment display. It takes seconds as input and output the control signals of the 7-segment display. We can implement this specialty with the modules `Minute-Second Separator`, `Binary To 7-Segment Convertor`, and `7-Segment Display`. First, I parse the time in second to the form of minutes and seconds. Then, convert them into the patterns shown on the 7-segment display.

```
1   `include "global.v"
2
3   module display_time_7_segment(
4       d_sel,
5       d_out,
6       q_lap,
7       clk,
8       rst
9       );
10
11      output [`SEGMENT_7_DISPALY_DIGIT_N-1:0]d_sel;
12      output [`SEGMENT_7_SEGMENT_N-1:0]d_out;
13      input [`BCD_COUNTER_BITS-1:0]q_lap;
14      input clk;
15      input rst;
16
17      wire [`SEGMENT_7_INPUT_BITS_N-1:0]D0_MIN_BINARY;
18      wire [`SEGMENT_7_INPUT_BITS_N-1:0]D1_MIN_BINARY;
19      wire [`SEGMENT_7_INPUT_BITS_N-1:0]D0_SEC_BINARY;
20      wire [`SEGMENT_7_INPUT_BITS_N-1:0]D1_SEC_BINARY;
21      wire [`SEGMENT_7_SEGMENT_N-1:0]D0_MIN_SEGMENT7;
```

```
22      wire [`SEGMENT_7_SEGMENT_N-1:0]D1_MIN_SEGMENT7;
23      wire [`SEGMENT_7_SEGMENT_N-1:0]D0_SEC_SEGMENT7;
24      wire [`SEGMENT_7_SEGMENT_N-1:0]D1_SEC_SEGMENT7;
25
26      // Extract digits
27      min_sec_seperate SEP(.i(q_lap), .d0_min(D0_MIN_BINARY), .d1_min(
            D1_MIN_BINARY), .d0_sec(D0_SEC_BINARY), .d1_sec(D1_SEC_BINARY));
28
29      // Convert binary to 7-segment
30      segment7 D0_MIN_CONV(.i(D0_MIN_BINARY), .D(D0_MIN_SEGMENT7));
31      segment7 D1_MIN_CONV(.i(D1_MIN_BINARY), .D(D1_MIN_SEGMENT7));
32      segment7 D0_SEC_CONV(.i(D0_SEC_BINARY), .D(D0_SEC_SEGMENT7));
33      segment7 D1_SEC_CONV(.i(D1_SEC_BINARY), .D(D1_SEC_SEGMENT7));
34
35      // Show
36      display_7seg(.clk(clk), .rst(rst), .d0(D0_SEC_SEGMENT7), .d1(
            D1_SEC_SEGMENT7), .d2(D0_MIN_SEGMENT7), .d3(D1_MIN_SEGMENT7), .
            d_sel(d_sel), .d_out(d_out));
37  endmodule
```

Stopwatch To implement a stopwatch, I only need to combine all modules into one. The `Frequency Divider` generate 1 Hz clock to trigger the `2-Digit Synchronous Binary Up Counter`. The `One-Pulse` module send signal whenever the button is clicked or long pressed to control the `Finite State Machine`. The `Lap Controller` controls the number shown on the display by the module `7-Segment Time Display`.

```
1   `include "global.v"
2
3   module exp_2(
4       leds,
5       D_SEL,
6       D_OUT,
7       restart,
8       push,
9       rst,
10      clk
11      );
12      output [`LEDS_NUM-1:0]leds;
13      output [`SEGMENT_7_DISPALY_DIGIT_N-1:0]D_SEL;
14      output [`SEGMENT_7_SEGMENT_N-1:0]D_OUT;
15      input restart;
16      input push;
17      input rst;
18      input clk;
19
20      //    reg [`BCD_COUNTER_BITS-1:0]q;
21      wire DIV_CLK;
22      wire [`SEGMENT_7_INPUT_BITS_N-1:0]D0_MIN_BINARY;
```

```
23      wire [`SEGMENT_7_INPUT_BITS_N-1:0]D1_MIN_BINARY;
24      wire [`SEGMENT_7_INPUT_BITS_N-1:0]D0_SEC_BINARY;
25      wire [`SEGMENT_7_INPUT_BITS_N-1:0]D1_SEC_BINARY;
26      wire [`SEGMENT_7_SEGMENT_N-1:0]D0_MIN_SEGMENT7;
27      wire [`SEGMENT_7_SEGMENT_N-1:0]D1_MIN_SEGMENT7;
28      wire [`SEGMENT_7_SEGMENT_N-1:0]D0_SEC_SEGMENT7;
29      wire [`SEGMENT_7_SEGMENT_N-1:0]D1_SEC_SEGMENT7;
30
31      wire PAUSE_ONEPULSE;
32      wire RESTART_ONEPULSE;
33      wire LAP_ONEPULSE;
34      wire IS_PAUSE;
35      wire IS_RESTART;
36      wire IS_LAP;
37
38      wire [`BCD_COUNTER_BITS-1:0]Q;
39      wire [`BCD_COUNTER_BITS-1:0]Q_LAP;
40
41  //    assign IS_RESTART = 1;
42  //    assign IS_PAUSE = 0;
43
44      // 1 Hz Clock
45      frequency_divider U0(.clk(clk), .rst(rst), .clk_out(DIV_CLK));
46
47      onepulse PauseBtn(.rst(rst), .clk(clk), .clk_long(DIV_CLK), .push(
            push), .push_onepulse(PAUSE_ONEPULSE));
48      onepulse RestartBtn(.rst(rst), .clk(clk), .clk_long(DIV_CLK), .push
            (restart), .push_onepulse(LAP_ONEPULSE), .push_onepulse_long(
            RESTART_ONEPULSE));
49      fsm FSM(.clk(clk), .pause(PAUSE_ONEPULSE), .restart(
            RESTART_ONEPULSE), .lap(LAP_ONEPULSE), .is_pause(IS_PAUSE), .
            is_restart(IS_RESTART), .is_lap(IS_LAP));
50
51      // 2-Digits Binary up counter
52      binary_up_4digit_counter U1(.clk(DIV_CLK), .rst(IS_RESTART), .
            is_pause(IS_PAUSE), .q(Q));
53      lap_controller LAP_CONTROL(.q(Q_LAP), .i(Q), .is_lap(IS_LAP));
54
55      // LEDs controller
56      led_control LED_CONTROL(.q(Q), .is_pause(IS_PAUSE), .is_restart(
            IS_RESTART), .is_lap(IS_LAP), .leds(leds));
57      display_time_7_segment TIME_DISPLAY(.q_lap(Q_LAP), .clk(clk), .rst(
            rst), .d_sel(D_SEL), .d_out(D_OUT));
58  endmodule
```

I/O Pin Assignment

I/O	clk	rst	push	restart	D_SEL[0]	D_SEL[1]	D_SEL[2]	D_SEL[3]
LOC	W5	V17	T17	W19	U2	U4	V4	W4

I/O	D_OUT[0]	D_OUT[1]	D_OUT[2]	D_OUT[3]	D_OUT[4]	D_OUT[5]	D_OUT[6]	D_OUT[7]
LOC	V7	U7	V5	U5	V8	U8	W6	W7

I/O	leds[0]	leds[1]	leds[2]	leds[3]	leds[4]	leds[5]	leds[6]	leds[7]
LOC	U16	E19	U19	V19	W18	U15	U14	V14

I/O	leds[8]	leds[9]	leds[10]	leds[11]	leds[12]	leds[13]	leds[14]	leds[15]
LOC	V13	V3	W3	U3	P3	N3	P1	L1

Block Diagram

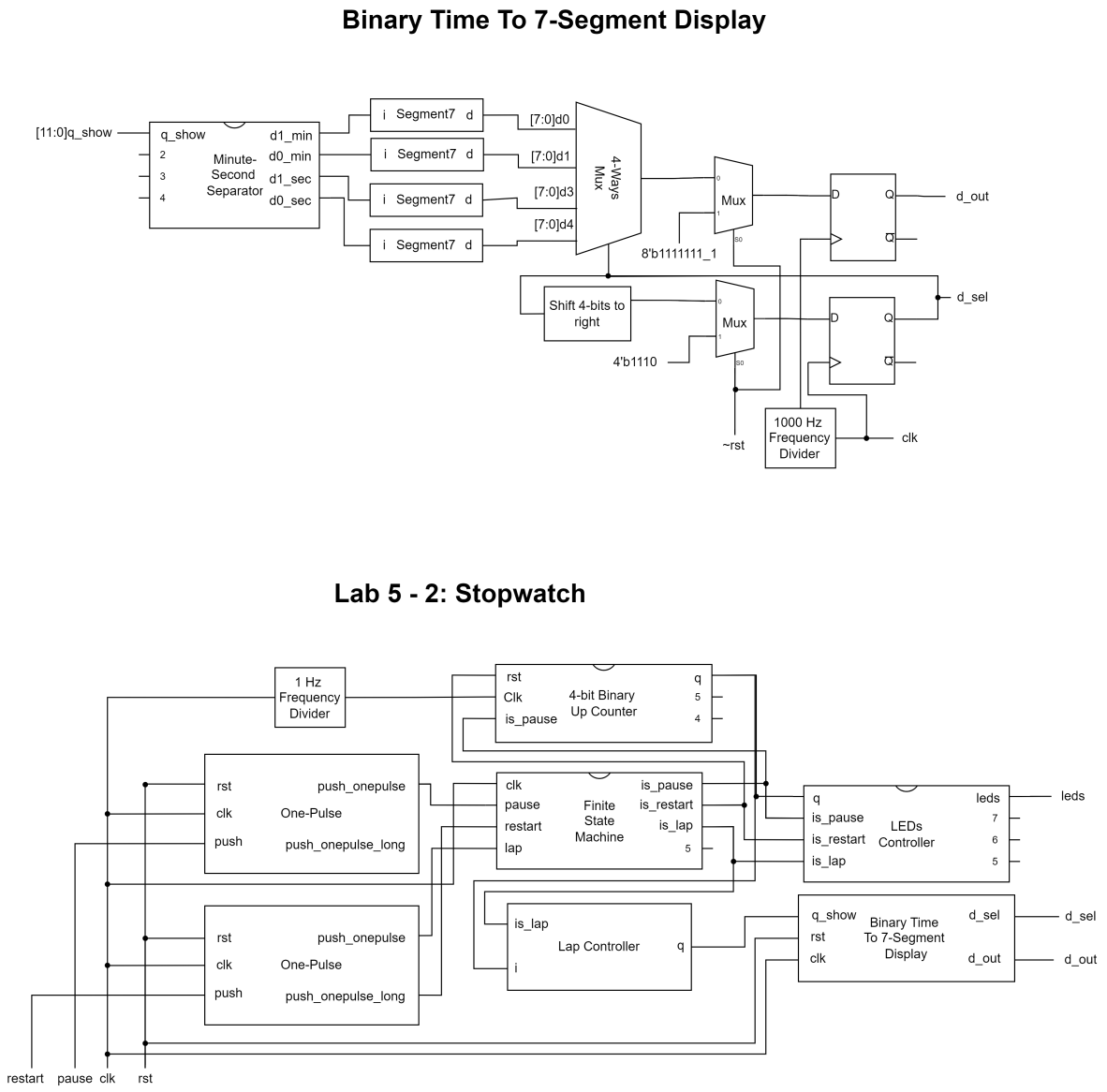

Figure 3: Lab 5-2 Logic Diagram

RTL Simulation

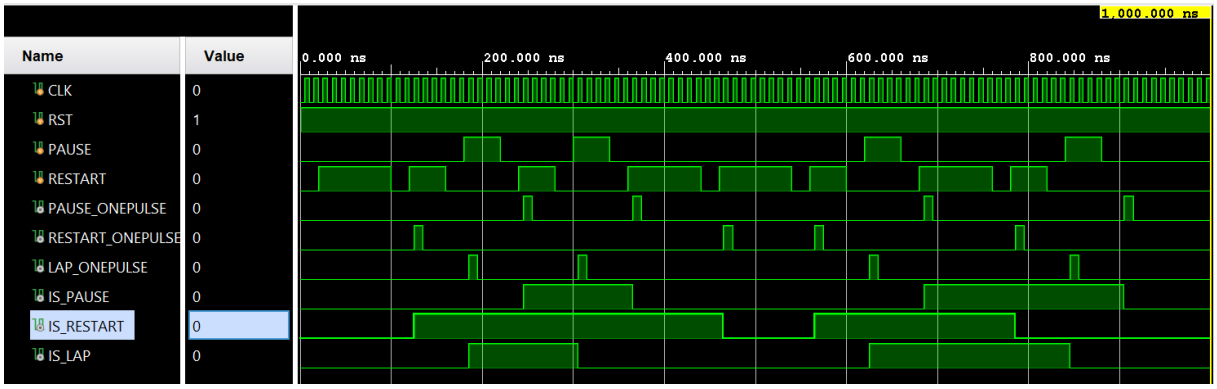

Figure 4: Lab 5-2 RTL Simulation

Lab 5 - 3: Timer

Design Specification

Source Code

Debounce

Input: rst, clk, push

Output: push_debounced

One-Pulse

Input: rst, clk, push

Output: push_onepulse

Finite State Machine

Output is_pause, is_restart, is_setting, [11:0]q_target

Input clk, pause, restart, mode_switch

Frequency Divider

Input: rst, clk

Output: clk_out

2-Digit Synchronous Binary Down Counter

Input: rst, clk

Output [11:0]q

LEDs Controller

Input: q, is_pause, is_restart, is_lap

Output: [15:0]leds

Minute-Second Separator

Input [11:0]i

Output [7:0]d0_min, [7:0]d1_min, [7:0]d0_sec, [7:0]d1_sec

Binary To 7-Segment Convertor

Input: [3:0] i

Output: [3:0]P, [7:0]D

7-Segment7 Frequency Divider

Input: clk, rst

Output: clk_out

7-Segment Display

Output: [0:3]d_sel, [7:0]d_out

Input: clk, rst, [7:0]d0, [7:0]d1, [7:0]d2, [7:0]d3

7-Segment Time Display

Output [3:0]d_sel, [7:0]d_out;

Input [11:0]q_lap, clk, rst

Timer

Output [15:0]leds, [3:0]D_SEL, [7:0]D_OUT

Input restart, push, rst, clk

Design Implementation

Global Variables

```
// Segment-7 Displayer
`define SEGMENT_7_INPUT_BITS_N 4
`define SEGMENT_7_DISPALY_DIGIT_N 4
`define SEGMENT_7_SEGMENT_N 8
`define SEGMENT_7_NONE `SEGMENT_7_SEGMENT_N'b1111111_1
`define SEGMENT_7_EMPTY `SEGMENT_7_SEGMENT_N'b1111111_0
```

```
 7
 8  // @Segment-7 Displayer Frequency Divider
 9  `define SEGMENT_7_FREQ_DIV_BITS 30
10  // 1000 Hz
11  `define SEGMENT_7_FREQ_DIV_COUNT `FREQ_DIV_BITS'd50000
12  //`define SEGMENT_7_FREQ_DIV_COUNT `FREQ_DIV_BITS'd2
13  // 1 Hz
14  //`define SEGMENT_7_FREQ_DIV_COUNT `FREQ_DIV_BITS'd5000000
15  `define RST_OFF 1'b1
16
17  // @BCD Counter
18  `define BCD_COUNTER_BITS 12
19  `define BCD_COUNTER_LIMIT `BCD_COUNTER_BITS'd60
20  `define BCD_COUNTER_ZERO `BCD_COUNTER_BITS'd0
21
22  // @Frequency Divider
23  `define FREQ_DIV_BITS 30
24  // 1 Hz
25  `define FREQ_DIV_COUNT `FREQ_DIV_BITS'd50000000
26  //`define FREQ_DIV_COUNT `FREQ_DIV_BITS'd2
27
28  // @LED control
29  `define LEDS_NUM 16
30
31  // @Minute-Second Seperater
32  `define UNIT_BITS 6
33
34  // One Pulse
35  `define COUNTER_BITS_N 30
36  `define PRESS_CYCLE_N `COUNTER_BITS_N'd200000000
37  //`define PRESS_CYCLE_N `COUNTER_BITS_N'd2
```

Debounce Same as lab 5-Prelab.

One-Pulse Same as lab 5-2.

Finite State Machine The finite state machine controls 3 signals: `is_pause`, is_setting and `is_restart`. The down counter will pause while the signal `is_pause` is at high voltage and resume to count while the signal is at low voltage. It is controlled independently, so I only need to inverse the signal whenever the button `restart` is pressed.

On the other hand, the down counter will reset to 0 when the signal `is_restart` raises. It only depends on `restart` button, so we only need to inverse the signal whenever the button is clicked.

Similarly, the signal `is_setting` indicates whether we can set the timer or not. It′ s controlled by the DIP switch directly.

To set up the timer limit, I use 2 always loop to detect the button. One is for minutes and the other is for seconds. They count the click of each button independently. Then, I sum them up together as the output.

```
`include "global.v"

`define STATE_SETTING 2
`define STATE_PAUSE 1
`define STATE_START 0

module fsm(
    is_pause,
    is_restart,
    is_setting,
    q_target,
    pause_trig,
    restart_trig,
    clk,
    pause,
    restart,
    mode_switch
    );

    output is_pause;
    output is_restart;
    output is_setting;
    output [`BCD_COUNTER_BITS-1:0]q_target;
    output restart_trig;
    output pause_trig;
    input clk;
    input pause;
    input restart;
    input mode_switch;

    reg is_restart, is_restart_temp;
    reg is_pause, is_pause_temp;
    reg state, state_temp;
    reg [`BCD_COUNTER_BITS-1:0]q_target;
    reg [`BCD_COUNTER_BITS-1:0]q_target_min;
    reg [`BCD_COUNTER_BITS-1:0]q_target_sec;
//    reg last_restart;
//    reg last_pause;

    assign is_setting = mode_switch;

    initial begin
        q_target = `BCD_COUNTER_BITS'd0;
//        last_state = `STATE_START;
        state = `STATE_START;
        state_temp = `STATE_START;
```

```
47
48          is_restart = 1'b0;
49          is_restart_temp = 1'b0;
50          is_pause = 1'b0;
51          is_pause_temp = 1'b0;
52  //        last_restart = 1'b0;
53  //        last_is_pause = 1'b0;
54  //        last_pause = 1'b0;
55
56          q_target_min = `BCD_COUNTER_BITS'd0;
57          q_target_sec = `BCD_COUNTER_BITS'd0;
58      end
59
60      always@(posedge restart) begin
61          if(!mode_switch) begin
62              is_restart_temp <= is_restart ^ 1;
63          end
64      end
65
66      always@(posedge pause) begin
67          if(!mode_switch) begin
68              if(state == `STATE_PAUSE) begin
69  //                last_state <= state;
70  //                last_is_pause <= is_pause;
71
72                  state_temp <= `STATE_START;
73                  is_pause_temp <= 0;
74              end else if(state == `STATE_START) begin
75  //                last_state <= state;
76  //                last_is_pause <= is_pause;
77
78                  state_temp <= `STATE_PAUSE;
79                  is_pause_temp <= 1;
80              end else begin
81                  state_temp <= `STATE_START;
82                  is_pause_temp <= 1'b0;
83
84  //                last_state <= `STATE_SETTING;
85  //                last_is_pause <= 0;
86              end
87          end
88      end
89
90      always@(posedge clk) begin
91  //        is_restart <= is_restart_temp;
92          q_target <= q_target_min + q_target_sec;
93
94          if(mode_switch) begin
95              state <= `STATE_SETTING;
96              is_pause <= 0;
97              is_restart <= 1'b1;
```

```
98          end else begin
99              state <= state_temp;
100             is_pause <= is_pause_temp;
101             is_restart <= is_restart_temp;
102         end
103     end
104
105     always@(posedge restart) begin
106         if(mode_switch) begin
107             q_target_min <= q_target_min + `BCD_COUNTER_BITS'd60;
108         end
109     end
110
111     always@(posedge pause) begin
112         if(mode_switch) begin
113             q_target_sec <= q_target_sec + `BCD_COUNTER_BITS'd1;
114         end
115     end
116
117 //    always@(posedge pause or mode_switch) begin
118 //        if(mode_switch) begin
119 //            state <= `STATE_SETTING;
120 //            is_pause <= 0;
121 //        end else if(state == `STATE_PAUSE) begin
122 //            state <= `STATE_START;
123 //            is_pause <= 0;
124 //        end else if(state == `STATE_START) begin
125 //            state <= `STATE_PAUSE;
126 //            is_pause <= 1;
127 //        end
128 //    end
129 endmodule
```

Frequency Divider Same as lab 5-2.

2-Digit Synchronous Binary Down Counter Same as lab 5-2

LEDs Controller It′ s similar to the `LEDs controller` in Lab5-2. While we are setting up the timer, `leds[1]` lights up. While the counter is counting, “leds[0] is bright. Whenever the counter hits 0, all the LEDs light up.

```
1  `include "global.v"
2
3  module led_controller(
4      q,
5      is_pause,
```

```
6       is_restart,
7       is_setting,
8       leds
9       );
10
11      input [`BCD_COUNTER_BITS-1:0]q;
12      input is_pause;
13      input is_restart;
14      input is_setting;
15      output [`LEDS_NUM-1:0]leds;
16
17      reg [`LEDS_NUM-1:0]leds;
18
19      always@(q or is_pause or is_restart or is_setting) begin
20          if(is_setting) begin
21              leds[`LEDS_NUM-1:2] = {(`LEDS_NUM-2){1'b0}};
22              leds[1] = 1'b1;
23              leds[0] = 1'b0;
24          end else begin
25              if(q == `BCD_COUNTER_ZERO)begin
26                  leds = {`LEDS_NUM{1'b1}};
27              end else begin
28                  leds[`LEDS_NUM-1:1] = {(`LEDS_NUM-1){1'b0}};
29
30                  if((!is_pause) && (is_restart)) begin
31                      leds[0] = 1'b1;
32                  end else begin
33                      leds[0] = 1'b0;
34                  end
35              end
36          end
37      end
38  endmodule
```

Minute-Second Separator Same as lab 5-2

Binary To 7-Segment Convertor Same as lab 5-Prelab

7-Segment Display Same as lab 5-Prelab

7-Segment Time Display Same as lab 5-2

Timer Combine all the module listed above. `One-Pulse` trigger the `Finite State Machine` and then affect the 3 signal `IS_PAUSE`, `IS_RESTART`, and `IS_SETTING`. These 3 signals control the be-

haviors of `Timer Controller` and `LED Controller` and decide the showing of the display and the LEDs.

```
`include "global.v"

module exp_3(
    leds,
    D_SEL,
    D_OUT,
    restart,
    push,
    mode_switch,
    rst,
    clk
    );
    output [`LEDS_NUM-1:0]leds;
    output [`SEGMENT_7_DISPALY_DIGIT_N-1:0]D_SEL;
    output [`SEGMENT_7_SEGMENT_N-1:0]D_OUT;
    input restart;
    input push;
    input mode_switch;
    input rst;
    input clk;

    //    reg [`BCD_COUNTER_BITS-1:0]q;
    wire DIV_CLK;
    wire [`SEGMENT_7_INPUT_BITS_N-1:0]D0_MIN_BINARY;
    wire [`SEGMENT_7_INPUT_BITS_N-1:0]D1_MIN_BINARY;
    wire [`SEGMENT_7_INPUT_BITS_N-1:0]D0_SEC_BINARY;
    wire [`SEGMENT_7_INPUT_BITS_N-1:0]D1_SEC_BINARY;
    wire [`SEGMENT_7_SEGMENT_N-1:0]D0_MIN_SEGMENT7;
    wire [`SEGMENT_7_SEGMENT_N-1:0]D1_MIN_SEGMENT7;
    wire [`SEGMENT_7_SEGMENT_N-1:0]D0_SEC_SEGMENT7;
    wire [`SEGMENT_7_SEGMENT_N-1:0]D1_SEC_SEGMENT7;

    wire PAUSE_ONEPULSE;
    wire RESTART_ONEPULSE;
    wire IS_PAUSE;
    wire IS_RESTART;
    wire IS_SETTING;

    wire [`BCD_COUNTER_BITS-1:0]Q;
    wire [`BCD_COUNTER_BITS-1:0]Q_SHOW;
    wire [`BCD_COUNTER_BITS-1:0]Q_TARGET;

//    assign IS_RESTART = 1;
//    assign IS_PAUSE = 0;

//    assign Q_TARGET = `BCD_COUNTER_BITS'd70;

    // 1 Hz Clock
```

```
49      frequency_divider U0(.clk(clk), .rst(rst), .clk_out(DIV_CLK));
50
51      onepulse PauseBtn(.rst(rst), .clk(clk), .push(push), .push_onepulse
            (PAUSE_ONEPULSE));
52      onepulse RestartBtn(.rst(rst), .clk(clk), .push(restart), .
            push_onepulse(RESTART_ONEPULSE));
53      fsm FSM(.clk(clk), .mode_switch(mode_switch), .pause(PAUSE_ONEPULSE
            ), .restart(RESTART_ONEPULSE), .is_pause(IS_PAUSE), .is_restart(
            IS_RESTART), .is_setting(IS_SETTING), .q_target(Q_TARGET));
54
55      // 2-Digits Binary up counter
56      binary_down_4digit_counter U1(.clk(DIV_CLK), .rst(IS_RESTART), .
            counter_limit(Q_TARGET), .is_pause(IS_PAUSE), .q(Q));
57      timer_controller TIMER_CONTROL(.counting(Q), .target(Q_TARGET), .
            is_setting(IS_SETTING), .q_show(Q_SHOW));
58
59      // LEDs controller
60      led_controller LED_CONTROL(.q(Q), .is_pause(IS_PAUSE), .is_restart(
            IS_RESTART), .is_setting(IS_SETTING), .leds(leds));
61      display_time_7_segment TIME_DISPLAY(.q_show(Q_SHOW), .clk(clk), .
            rst(rst), .d_sel(D_SEL), .d_out(D_OUT));
62  endmodule
```

I/O Pin Assignment

I/O	clk	rst	push	restart	D_SEL[0]	D_SEL[1]	D_SEL[2]	D_SEL[3]
LOC	W5	V17	T17	W19	U2	U4	V4	W4

I/O	D_OUT[0]	D_OUT[1]	D_OUT[2]	D_OUT[3]	D_OUT[4]	D_OUT[5]	D_OUT[6]	D_OUT[7]
LOC	V7	U7	V5	U5	V8	U8	W6	W7

I/O	leds[0]	leds[1]	leds[2]	leds[3]	leds[4]	leds[5]	leds[6]	leds[7]
LOC	U16	E19	U19	V19	W18	U15	U14	V14

I/O	leds[8]	leds[9]	leds[10]	leds[11]	leds[12]	leds[13]	leds[14]	leds[15]
LOC	V13	V3	W3	U3	P3	N3	P1	L1

Block Diagram

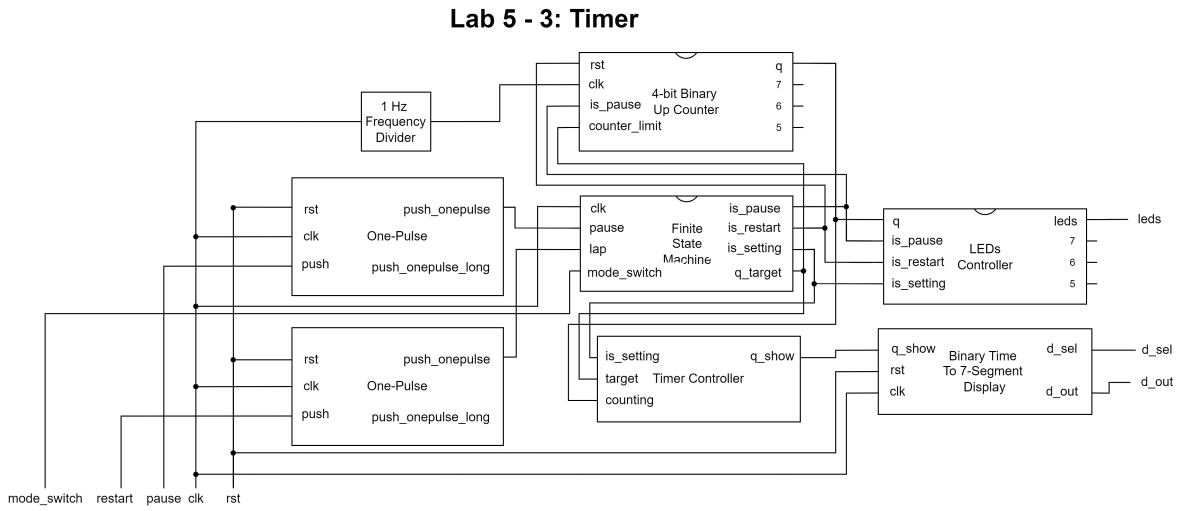

Figure 5: Lab 5-3 Logic Diagram

RTL Simulation

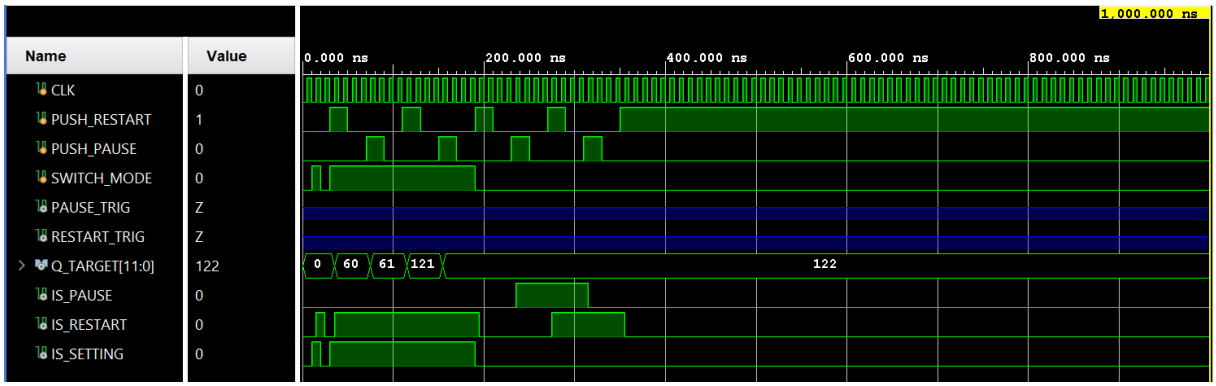

Figure 6: Lab 5-3 RTL Simulation

Discussion

In lab 5-2, it took me lots of time to design the state machine because the always block cannot be activated by 2 edge-triggered variables. Finally, I store the variables in the 2 independent variables and add them together as the clock raises. In addition, when I implement the one-pulse module, it took me much time to detect long press and click in the mean time. In the end, I use a counter to count the clock cycle during the pressed button.

Conclusion

The lab5 is much more difficult than previous labs. We′ ve learned how to design a more complex circuit with the concept of state machine. We also gradually get used to a larger project.

Reference

- None