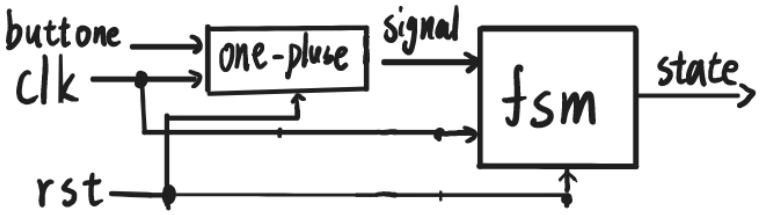
Logic Design Lab 5

Prelab 1

Construct a 50-second down counter (timer) with pause function. When the counter goes to 0, all the LEDs will be lighted up. You can use one push button for reset and one other for pause/start function.

Design Specification：

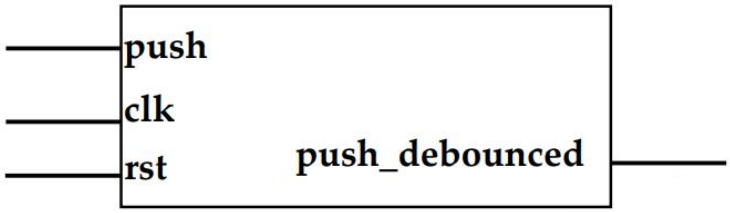
Input: clk, rst, button.
Output: state.
Block diagram:



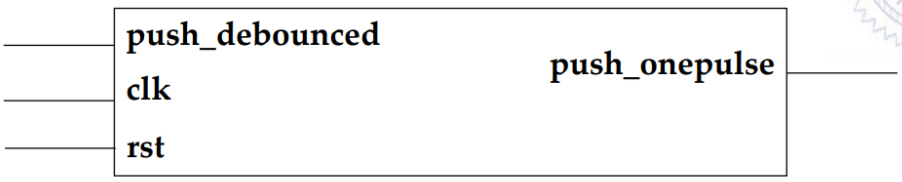
* one pluse with debounce

Design Implementation：

1. Debounce Circuit (copied from the handout)

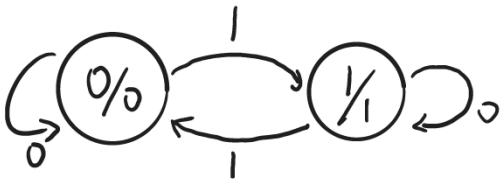


2. One-Pulse Generation (copied from the handout)



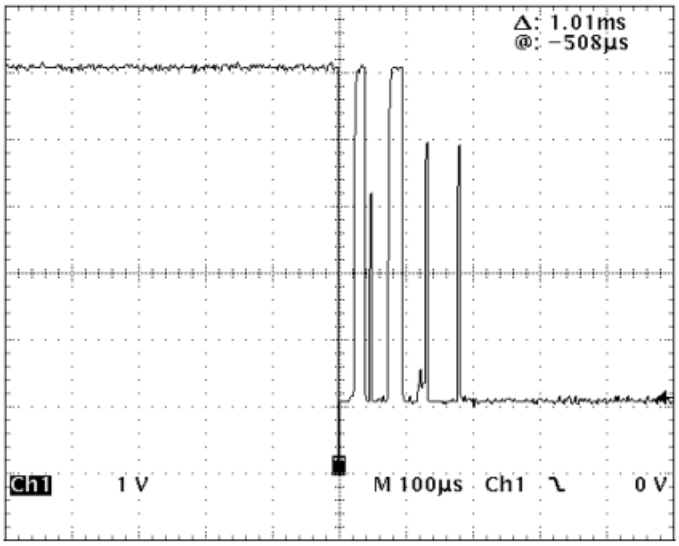
3. Finite state machine

State 0: pause
State 1: start



Discussion：

1. Because of the undesired variation of the value of the output when we use the button, we need to debounce the signal generated by some circuit. Besides, debouncing the signal, we also want to generate a “pulse” for typical use.

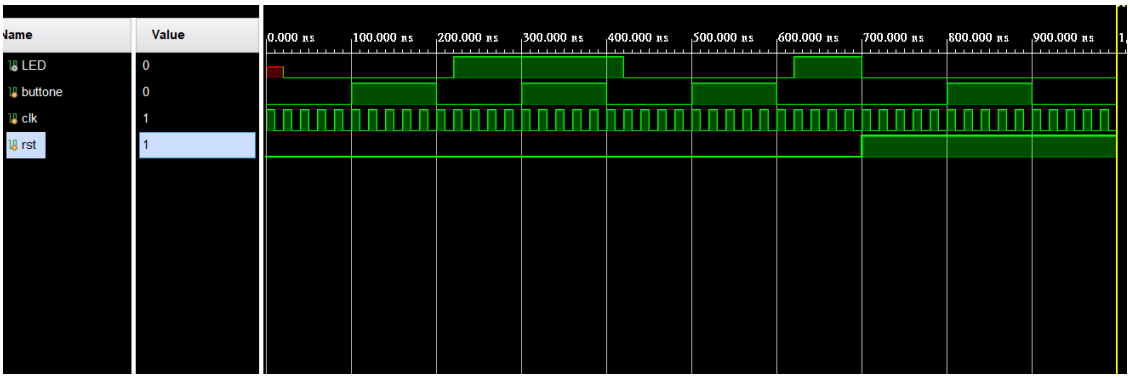


(Copied from the handout)

2. The method I use to solve the problem is just to follow the code from the handout of the class.

3. Because we want to assign more functions to our circuit, we start to use various state to design the input/ output relation and the transition of the states.

4. Simulation wave form



From the plot, we can observe when we press the button 1st time, the state turn from state0 to state1(in the simulation, the state is represented by the variable LED). Then, when we press the button 2nd time, the state falls to state 0. Now, we press the button again, it is obvious that the state goes to state1. However, if we turn the reset on, the state goes back to state0 and keep the state.

Conclusion：

From this experiment, we start to design our circuit with finite state machine (FSM). In this experiment, the FSM is simple, and we are also encouraged to design the FSM with Moore model to prevent the hazard of the circuit in the logic design course. That is to say, the output change will follow the state change.

Although, the FSM is quite simple, we need to think clearly to ensure the FSM will function well. If there is some wrongs in the FSM, the output will be weird and hard to debug.

Experiment 1

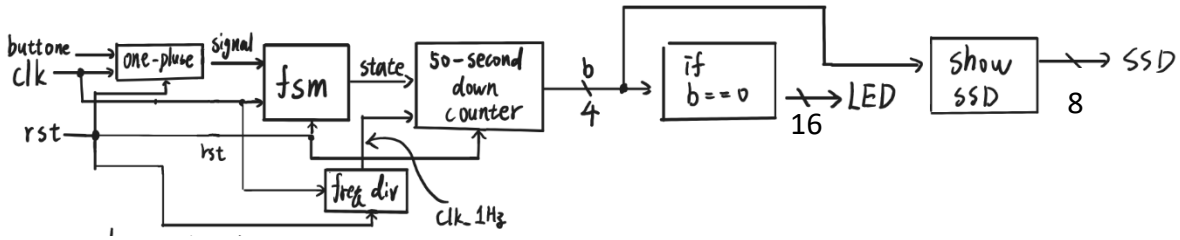
Construct a 50-second down counter (timer) with pause function. When the counter goes to 0, all the LEDs will be lighted up. You can use one push button for reset and one other for pause/start function.

Design Specification：

Input: clk, rst, button.

Output: LED [15:0], SSD [7:0].

Block diagram:

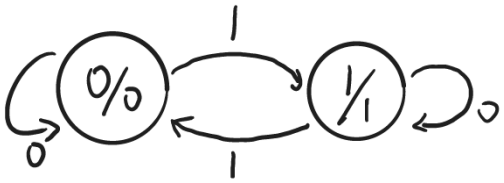


Design Implementation：

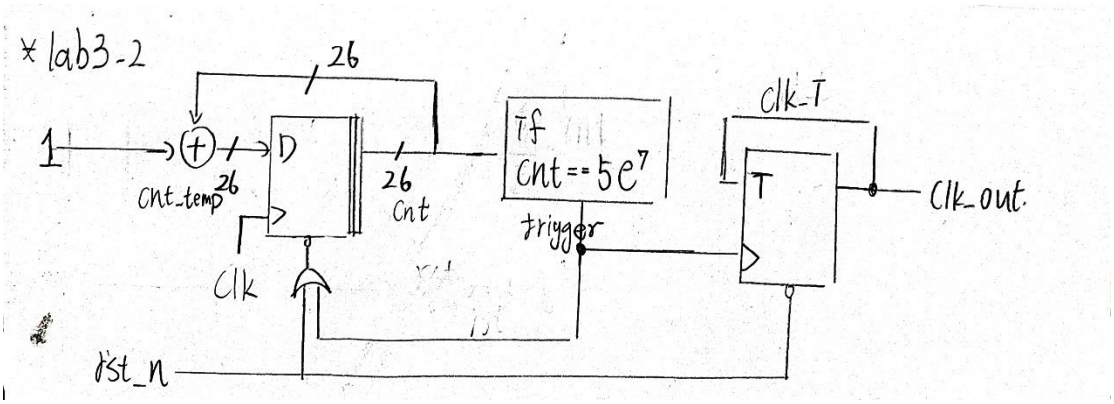
1. Debounce Circuit and One-Pulse Generation
   Follow the handout.

2. FSM

State 0: pause
State 1: start



3. Frequency divider
   We implemented it at lab3.



4. 50-second down counter

```
module down_counter(b0,b1,state,clk,rst_n);
output reg [3:0] b0;
output reg [3:0] b1;
input state;
input clk;
input rst_n;

reg [3:0]b0_temp;
reg [3:0]b1_temp;

//combination
always @*
begin
    case(state)
    1'b1: begin
          if(b0==4'd0 && b1==4'd0) begin b0_temp= 1'b0; b1_temp= 1'b0; end
          else if(b0==4'd0) begin  b0_temp= 4'd9; b1_temp= b1-1'b1; end
          else begin b0_temp= b0 - 1'd1; b1_temp= b1; end
          end
    1'b0: begin b0_temp=b0;b1_temp=b1;end
    default: begin b0_temp=b0;b1_temp=b1;end
    endcase
end

//sequencial
always @(posedge clk or posedge rst_n)
if (rst_n) begin
  b0 <= 4'd0;
  b1 <= 4'd5;
  end

else begin
  b0 <= b0_temp;
  b1 <= b1_temp;
end
endmodule
```

5. Show SSD (ssd output control)
   We have implemented it on previous labs.

Discussion：

1. The circuit of one pulse generation and FSM is the same as prelab5_1.
2. In this case, the number of the counter is fixed (50). Thus, we just set the initial value of the counter 50. However, we can expend the idea of this example to more flexible way. That is to say, we can assign the initial value of the counter by users.
3. I use two states to implement the FSM with Moore model. And, the FSM receive the signals from the button and then output the present state to down counter to decide the function of the down counter.

Conclusion：

Although this experiment doesn't look complicate, it still takes me a lot of time to do the experiment. the more modules we use, it is more likely we do mistakes. Maybe, it is possible that I do the experiment1 directly instead of following the hints of prelab1. Therefore, following the basic function of the module may help us design the circuit.

Experiment 2

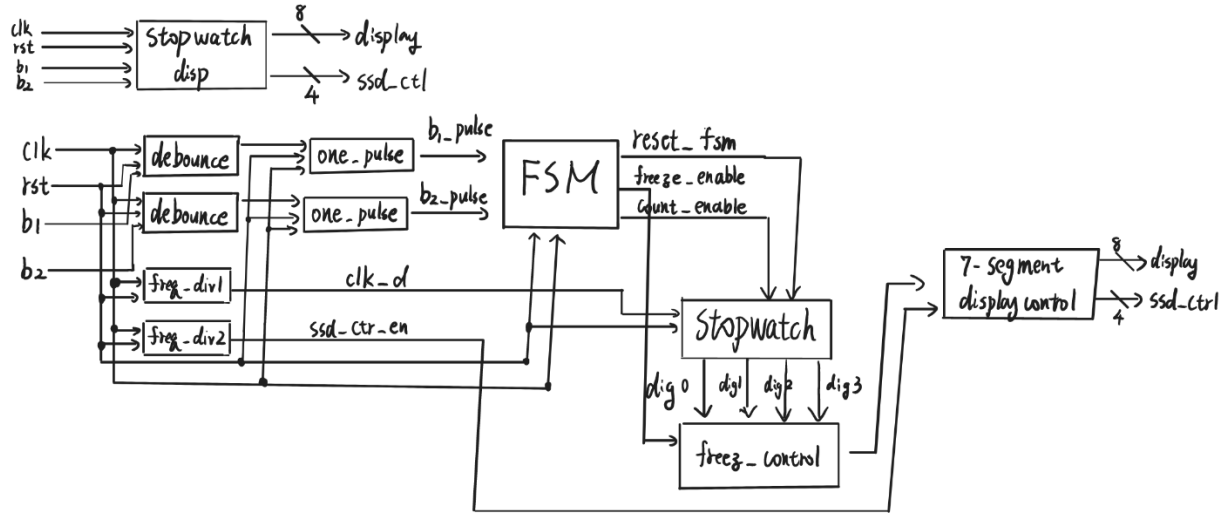
Implement a stopwatch function (00:00-59:59) with the FPGA board.

Design Specification：

Input: clk, rst, b1, b2.

Output: ssd_ctrl [3:0], display [7:0].

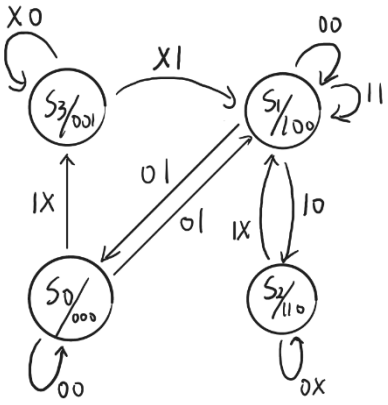
Block diagram:



Design Implementation：

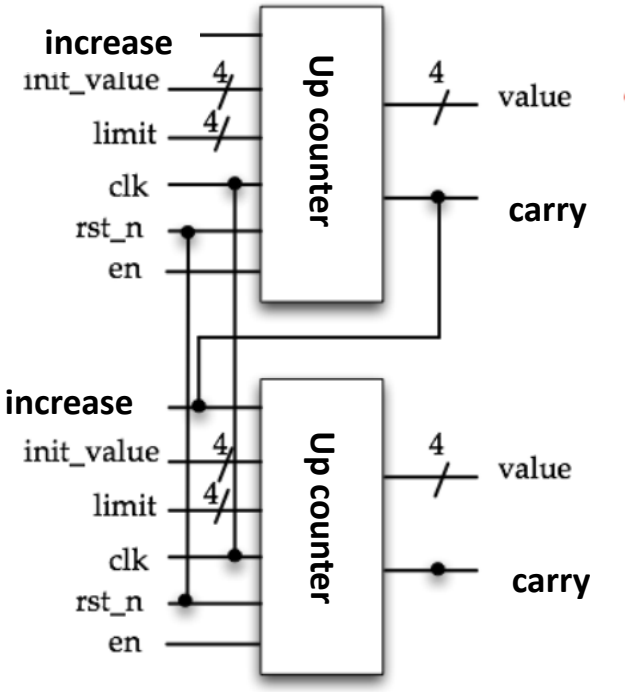
1. FSM

Let S0 : stop
S1 : start
S2 : pause display
S3 : Default
b1 : start/stop
b2 : lap/ reset



present state	input b1	input b2	Next state	output count_en	output freez_en	output rst
S0	0	0	S0	0	0	0
S0	0	1	S3	0	0	1
S0	1	0	S1	1	0	0
S0	1	1	X	0	0	0
S1	0	0	S1	1	0	0
S1	0	1	S2	1	1	0
S1	1	0	S0	0	0	0
S1	1	1	X	0	0	0
S2	0	0	S2	1	1	0
S2	0	1	S1	1	0	0
S2	1	0	S0	0	0	0
S2	1	1	X	0	0	0

2. Stopwatch implementation (cascade 4 up-counters)



(Revised from the handout)

3. Freeze control：

```
54  // Display latch
55  always @(posedge clk or posedge rst or posedge rst_fsm)
56  if (rst || rst_fsm) begin
57  dig0_out <= 4'd0;
58  dig1_out <= 4'd0;
59  dig2_out <= 4'd0;
60  dig3_out <= 4'd0;
61  end
62
63  else begin
64  dig0_out <= dig0_latch;
65  dig1_out <= dig1_latch;
66  dig2_out <= dig2_latch;
67  dig3_out <= dig3_latch;
68  end
```

```
70  // Whether the display is freezed
71  always @* begin
72  if (freeze_enable) begin
73  dig0_latch = dig0_out;
74  dig1_latch = dig1_out;
75  dig2_latch = dig2_out;
76  dig3_latch = dig3_out;
77  end
78
79  else
80  begin
81  dig0_latch = dig0;
82  dig1_latch = dig1;
83  dig2_latch = dig2;
84  dig3_latch = dig3;
85  end
86  end
```

Discussion：

1. From the FSM and logic table，we can get the relation between state as well as input / output, which provides us the method dealing with more complicated problem wisely and efficiency.
2. Simply put, the stopwatch is the module cascading some counters. And, we should be careful of the relation of different counters. For example, the enable control of the higher digit is carry of the lower digits. However, to make the stopwatch all-around, we need to use FSM to get more functions.

3. Freeze control is the function which allowed user to “freeze” the number showing om the displayer while the clock works continuous. This function can be implemented by latches which can store numbers showing on the screen.

Conclusion：

This experiment took me a lot of time to debug errors although we followed the method on the handout of logic design lab. When the modules become bigger and bigger, there are various types of the error. Sometimes, it takes of l a lots of time to debug and it turns out that is just a typo error. However, when I finished the experiment, I gain a great sense of achievement.

Experiment 3

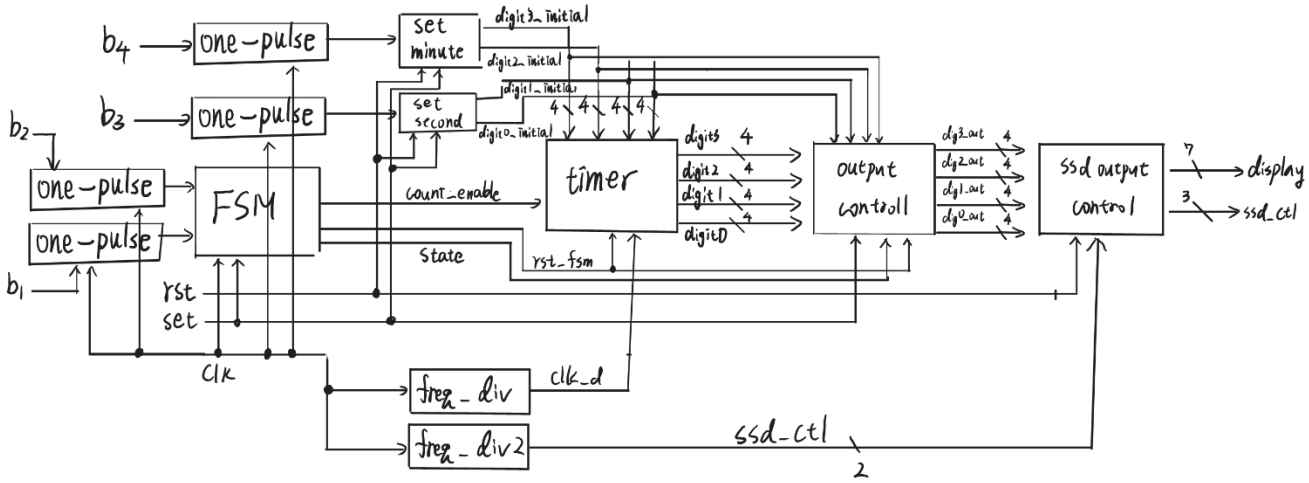
Implement a timer (can support as long as 59:59) with the following functions.

Design Specification：

Input: clk, rst, b1, b2, b1, b2.

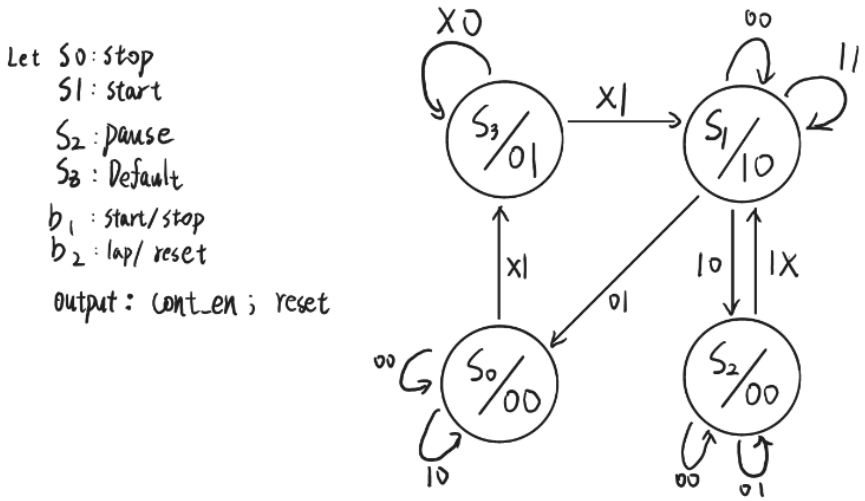
Output: ssd_ctrl [3:0], display [7:0].

Block diagram:



Design Implementation：

1. FSM：



2. Setting function

```
23  module setting
24  (
25  digit1_initial, //2nd digit of the down counter
26  digit0_initial, //1st digit of the down counter
27  signal, //global clock
28  rst, //active high reset
29  en //enable/disable for stopwatch
30  );
31
32  output [3:0] digit1_initial;
33  output [3:0] digit0_initial;
34  input signal; //global clock
35  input rst; //active high reset
36  input en; //enable/disable for stopwatch
37
38
39  wire carry0, carry1; //carry indicator
```

```
39  wire carry0, carry1; //carry indicator
40  wire increase_enable;
41
42  assign increase_enable = en && (~((digit0_initial==4'd9)
43                              && (digit1_initial==4'd5)));
44  //down counter
45  upcounter  Udc0(
46  .value(digit0_initial), //counter value
47  .carry(carry0), //carry indicator for counter to next stage
48  .signal(signal), //global clock signal
49  .rst(rst), //high active reset
50  .increase(increase_enable), //increase input from previous stage of counter
51  .init_value(0), //initial value for the counter
52  .limit(4'd9), //limit for the counter
53  .en(1) //enable/disable of the counter
```

```
56  upcounter Udc1(
57  .value(digit1_initial), //counter value
58  .carry(carry1), //carry indicator for counter to next stage
59  .signal(signal), //global clock signal
60  .rst(rst), //high active reset
61  .increase(carry0), //increase input from previous stage of counter
62  .init_value(0), //initial value for the counter
63  .limit(4'd5), //limit for the counter
64  .en(1) //enable/disable of the counter
65  );
66  endmodule
67
```

3. Timer

It is similar to the counter we use in the stopwatch. However, there are still something different and it will be discuss in discussion.

4. Output control

```
72  always @(rst_fsm,~rst,set,state)
73  begin
74      if (rst_fsm || ~rst) //setting state
75      begin
76      dig0_out=4'd0;
77      dig1_out=4'd0;
78      dig2_out=4'd0;
79      dig3_out=4'd0;
80      end
81      else if (set || state==3'd4)
82      begin
83      dig0_out= digit0_initial;
84      dig1_out= digit1_initial;
85      dig2_out= digit2_initial;
86      dig3_out= digit3_initial;
87      end
88      else
89      begin
90      dig0_out=dig0;
91      dig1_out=dig1;
92      dig2_out=dig2;
93      dig3_out=dig3;
94      end
```

Discussion：

1. Because I want to simplify the circuit, I don’t use another two states to set minute and second while I found that we set the time when the state is at the default state. Thus, I use two setting functions to set the time when the state is at the default. However, this may encounter the problem about loading the number on the counter.
2. To deal with this problem, I need to revise the design of the counter in the handout. It is not difficult to add loading function on the counter, but it still comes up the problem which happens when the counters start to count. The number on the displays may change to the counters themselves instead of the setting state.
3. Therefore, I need an output control to decide the final value transfering to the ssd output control.

Conclusion：

This experiment sees more complex. However, this experiment took me the least time of these three experiments. Perhaps, the structure is similar to the previous experiments, so I need to redesign the circuit base on the effort we have made. However, I think the fundamental reason is that we get more familiar to the design of counters and FSM, which allows us to design the circuit fast and accurately.

Reference：

1. The handout and the assignment of logic design
   To review some basic concept of the combination circuit and the problem bulls and cows.
2. The handout of logic design lab
   To program our FPGA board by following the method on the handout of logic design lab.