Lab 2

學號: 109000168 姓名: 許媄香

1. 實作過程

- lab2_1
  To design the 6-bit counter that count up and downward repeatedly, I declared two 6-bit reg, one is a temporary counter to store the where the counter have count, and the other is a index, to check if it's reseted or not. Also I add a 1-bit reg to check the which way the counter count. If the signal is '0', then it count upward, otherwise it's downward. If the counter count upward, it must increase with the condition below.

$$a_n = \begin{cases} 0 & if\ n = 0 \\ a_{n-1} - n & if\ a_{n-1} - n > 0 \\ a_{n-1} + n & otherwise \end{cases}$$

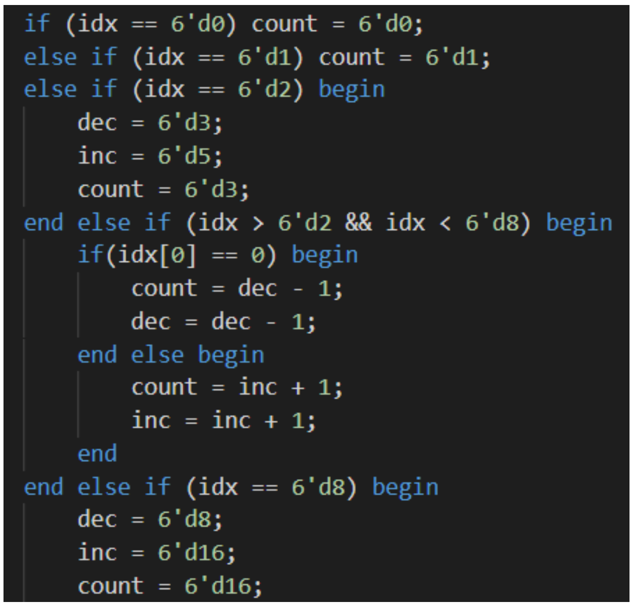
  I add those condition in the always block which triggered by positive edge clock and reset. However for the second condition "if (a_{n-1} – n > 0)", I substitute it with a new 7-bits signed reg since the result might be minus. Suppose the temporary counter has reach 63, I change the direction to '1' in order to make the counter count downward afterwards. Besides, I assign the output to be '0' and the index will be equal to 1.

  To count the counter downward like the counter below, I shift the counter 1 bit to the left everytime the counter is triggered. After the counter reach 0, I change the direction signal to 0 so then it will count upward again.

$$\begin{array}{lccccc} \text{Value} & X \rightarrow & (X-1) \rightarrow & ((X-1)-2) \rightarrow & ((X-1-2)-4) \rightarrow & \cdots \rightarrow 0 \\ \textbf{Sequence } \boldsymbol{b} & b_0 & b_1 & b_2 & b_3 & \cdots \end{array}$$

- lab2_1_t
  For the testbench, I add a pass signal to check if my ouput is correct or not, the signal will be 1 if it pass and 0 if it doesn't pass. I added the direction and index signal which is the same as lab2_1. Besides, I added 2 6-bit reg, inc and dec, for checking the output.

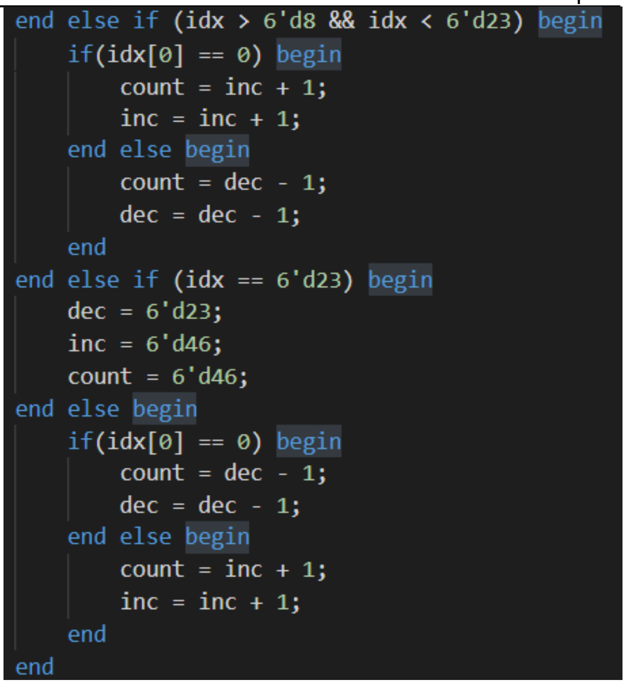
  In this testbench, I calculate the counter when the positive clock is triggered and update the pass signal when it's clock signal is negative. As to calculate the counter, I kind of declare it manually, Despite that, when the counter count upward, I found a little pattern. In a certain interval, whenever the number increased, the next number will be decreased. For this reason, I make a quite complicated if else statement to



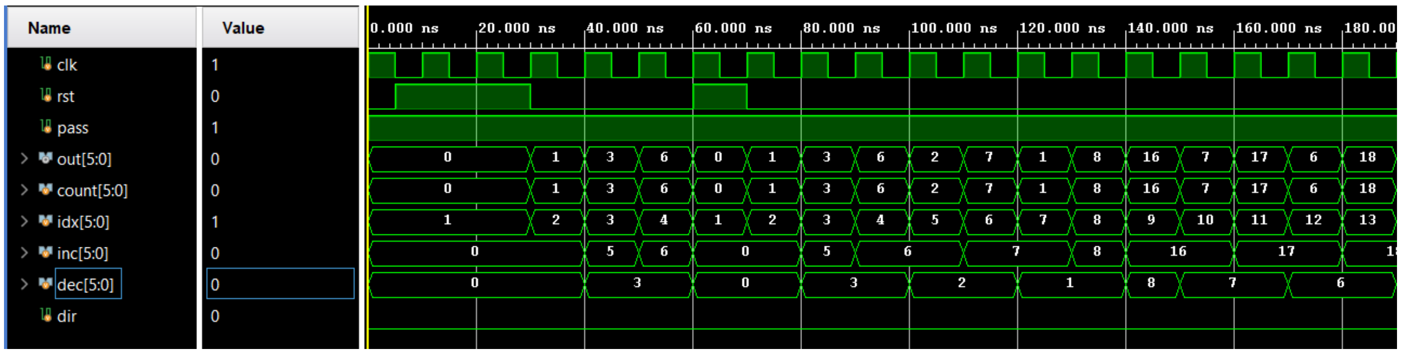
```
if (idx == 6'd0) count = 6'd0;
else if (idx == 6'd1) count = 6'd1;
else if (idx == 6'd2) begin
    dec = 6'd3;
    inc = 6'd5;
    count = 6'd3;
end else if (idx > 6'd2 && idx < 6'd8) begin
    if(idx[0] == 0) begin
        count = dec - 1;
        dec = dec - 1;
    end else begin
        count = inc + 1;
        inc = inc + 1;
    end
end else if (idx == 6'd8) begin
    dec = 6'd8;
    inc = 6'd16;
    count = 6'd16;
```

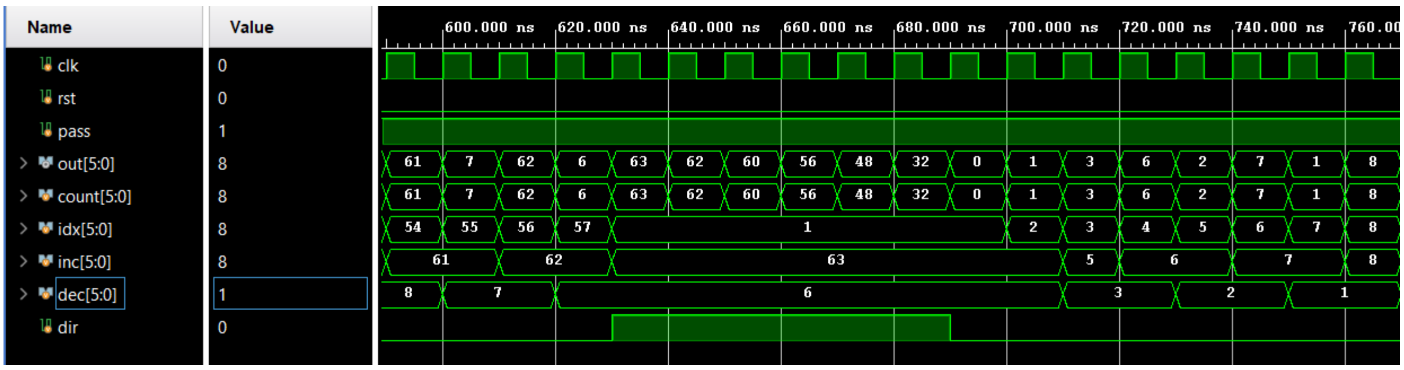
calculate the counter. However, for counting downward is simple. I take the last 5-bit of the previous output and put it as the first 5-bit in the current output and add a 0 bit in the last. It's the same as the lab1 shifter testbench. Similar to the lab2_1, after it reach 0 or 63, I change the direction signal according to the output.

To run the testbench, it's similar to the lab1 testbench but more simple since I only change the reset signal. First, I initial every signals to 0 except the clock and pass signals. Then, I make the reset signal become 1 for two and half clock cycle. After three clock cycle, I reset it again for one clock cycle. Next, the testbench will be finished after 70 clock cycles.



```
end else if (idx > 6'd8 && idx < 6'd23) begin
    if(idx[0] == 0) begin
        count = inc + 1;
        inc = inc + 1;
    end else begin
        count = dec - 1;
        dec = dec - 1;
    end
end else if (idx == 6'd23) begin
    dec = 6'd23;
    inc = 6'd46;
    count = 6'd46;
end else begin
    if(idx[0] == 0) begin
        count = dec - 1;
        dec = dec - 1;
    end else begin
        count = inc + 1;
        inc = inc + 1;
    end
end
```

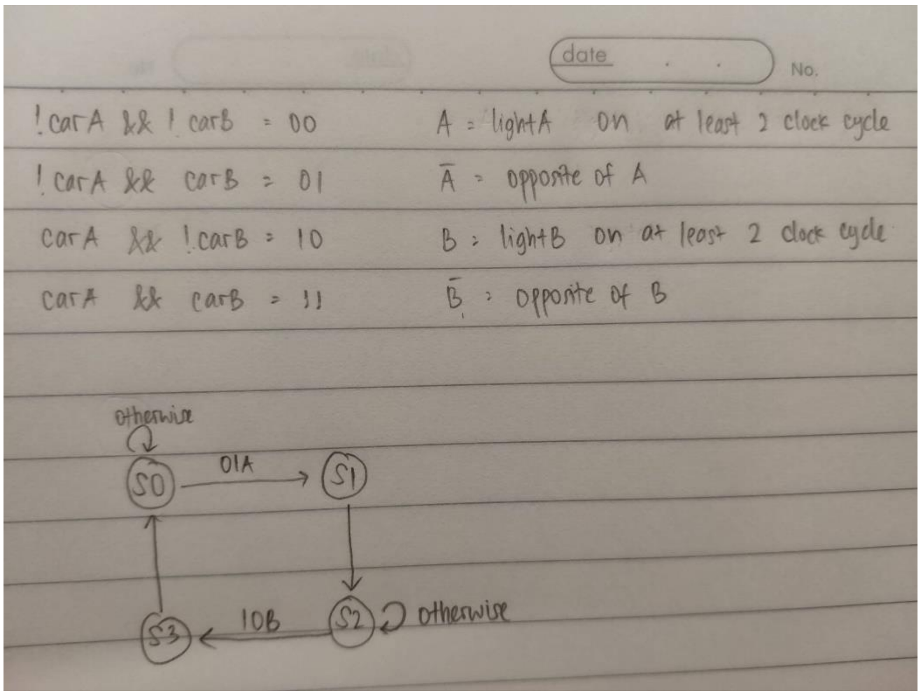




- lab2_2
  I design a FSM with 4 states.
  - S0 = lightA is green, lightB is red
  - S1 = lightA is yellow, ligthB is red
  - S2 = lightA is red, lightB is green
  - S3 = lightA is red, lightB is yellow

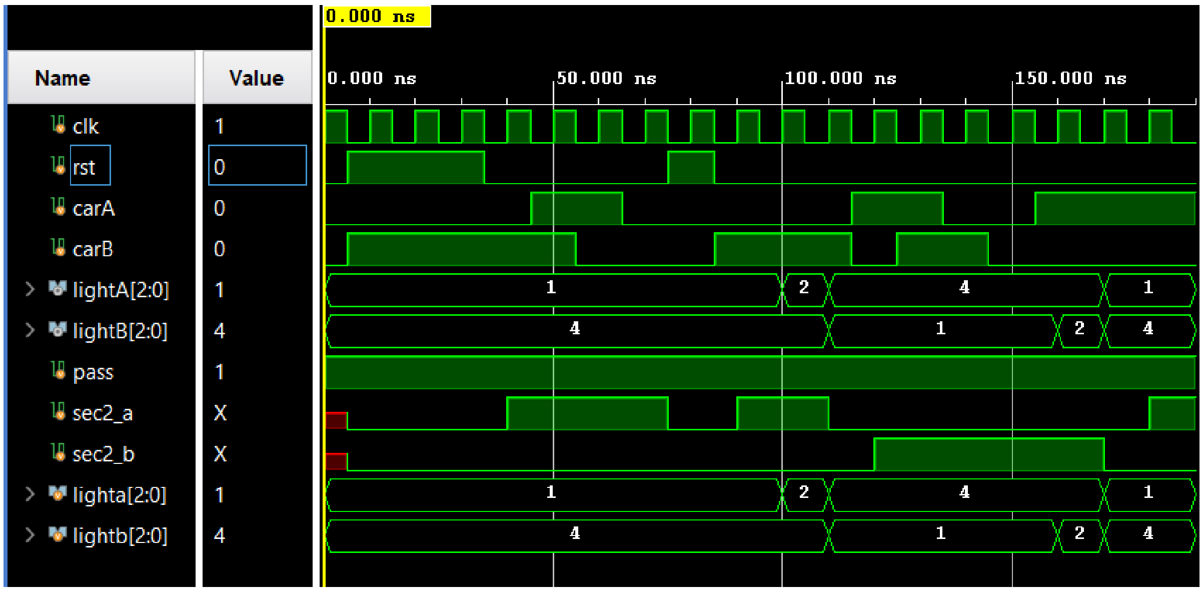
  S0 state change to S1 only if there is carB, not carA and lightA has been on for at least 2 clock cycle. Otherwise, it will keep in the S0 state and A signal become 1.
  S1 always change to S2.
  S2 state is the similar to S0, it will change to S3 if there is carA, not carB and lightB has been on for at least 2 clock cycle. Otherwise, it will keep in the S2 state and B signal become 1.

S3 always change to S0.



!carA && !carB = 00	A : lightA on at least 2 clock cycle
!carA && carB = 01	$\bar{A}$: opposite of A
carA && !carB = 10	B : lightB on at least 2 clock cycle
carA && carB = 11	$\bar{B}$: opposite of B

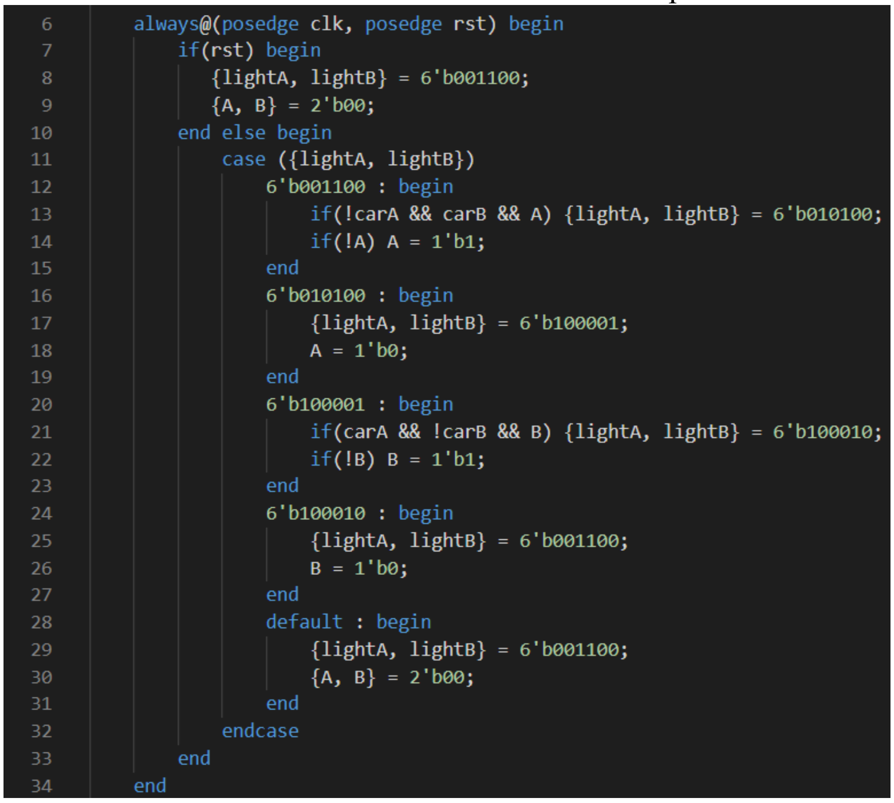
For the Verilog code, I design two always block. One is sequential, the other is combinational. In the sequential, the block will triggered when reset and clock signal make transition from 0 to 1. If the reset signal is 1, the state will be S0 and both A and B signal also become 0. Otherwise, the state will assign to be next state. In the combinatioanal block, it only triggered when the clock signal is positive. And I put the condition from the finate state machine in the combinational block. But I add a case where the reset signal is 1, both A and B signal also become 0 because in the sequential block, the A and B signal will be reset late since it's asynchronous.



2. 學到的東西與遇到的困難

In lab2_1 I spent a lot of time in debugging the code. The counter keep count from 0 and the next is 63. After couples of hour, I just found out that in this condition "if (a_{n-1} – n > 0)", when n = 1, a_0 = 0, the result of 0 – 1 is 63, meaning that it's still in unsigned binary. Therefore, I make that into a signed binary. Finally, it pass all the testcase.

When I was doing the lab2_2, I got some idea about how to design the module after reading the testbench from the TAs. And I wrote the code like the picture below and though that it was finished.



```verilog
    always@(posedge clk, posedge rst) begin
        if(rst) begin
            {lightA, lightB} = 6'b001100;
            {A, B} = 2'b00;
        end else begin
            case ({lightA, lightB})
                6'b001100 : begin
                    if(!carA && carB && A) {lightA, lightB} = 6'b010100;
                    if(!A) A = 1'b1;
                end
                6'b010100 : begin
                    {lightA, lightB} = 6'b100001;
                    A = 1'b0;
                end
                6'b100001 : begin
                    if(carA && !carB && B) {lightA, lightB} = 6'b100010;
                    if(!B) B = 1'b1;
                end
                6'b100010 : begin
                    {lightA, lightB} = 6'b001100;
                    B = 1'b0;
                end
                default : begin
                    {lightA, lightB} = 6'b001100;
                    {A, B} = 2'b00;
                end
            endcase
        end
    end
```

After a few days, I watch the professor lecture about the FSM in eeclass. I found out that it divided into two always block. One is sequential block which using a non blocking assignment and the others is combinational block and using a blocking assignmen. Then I was having a hard time to change my code since last semester when I was taking the Logic Design class, I don't really understand how the non-blocking works. However, after I read some information about it from the lecture pdf and google, I started to write the Verilog code. But, I faced a problem. In some cases, my state is not updated to the next_state. I was so confused about it and I found nothing wrong in my code. And I keep changing my code and finally it was done. Unfortunately, I found some weird output in the waveform. That is the output which I have never assign in my code, the lightA and lightB are green at the same time. I keep trying to change everything that is possible to change. Like in the beginning I assign {lightA, lightB} = next_state. And I change it into lightA = next_state[5:3], lightB = next_state[2:0]. Surprisingly, after I changed it become like that, it pass all the testbench. After demo, I was still curios why {lightA, lightB} = next_state didn't pass the testbench. I tried to run the simulation again and my Vivado 2021 keep freezing and crash. So I install the 2019 and run the simulation. Strangely, it pass all the testbench, lightA and lightB didn't come out green at the same time. Therefore, I think maybe I do make a mistake in the previous code and I didn't realize it or maybe it was just a bug in the waveform.

3. 想對老師或助教說的話

None