Lab 4

學號 ：109000168 姓名 ： 許媄香

1. 實作過程

- lab4_1

In this lab, it needs 7 segments display to display counter number and direction, controlled by the push buttons. Here it required 5 buttons, **rst**, **en**, **dir**, **speed_up** and **speed_down**. I instantiate debounce module for all button except **rst** since it doesn't really matter if **rst** button were bouncing. Besides, I also instantiate one pulse module to convert those buttons but not for **dir** because the it will count down if we hold the **dir** button and will count up if we release it. Therefore, it doesn't need to convert it to one clock cycle signal. For the debounce and onepulse code, I use the code from professor's lecture slides.

```
debounce direction (.pb_debounced(cnt_down), .pb(dir), .clk(clk));

debounce enable_debounced (.pb_debounced(enable), .pb(en), .clk(clk));
onepulse enable_one_pulse (.pb_debounced(enable), .clk(clk), .pb_1pulse(pause));

debounce faster_debounce (.pb_debounced(fast), .pb(speed_up), .clk(clk));
onepulse faster_one_pulse (.pb_debounced(fast), .clk(clk), .pb_1pulse(faster));

debounce slower_debounce (.pb_debounced(slow), .pb(speed_down), .clk(clk));
onepulse slower_one_pulse (.pb_debounced(slow), .clk(clk), .pb_1pulse(slower));
```

en (connected to **BTNU**): the control signal to toggle between the *start/resume* and *paus***e** mode.
Press the button to start the counting. Press it again to pause the counting. Press it one more time to resume the counting, and so on. After the reset, the counter is initialized to 00 and remains unchanged. That is, the counter will be in the pause mode initially. The counter will stop at 99 eventually when counting up; the counter will stop at 00 at the end when counting down.

To make the **en** button works as above requirement, I make a new reg **start** to keep in track whether it was paused or not using the **en** that has been processed with one pulse converter (**pause**). If **start** signal is 0 the counting will be stopped otherwise it will start counting.

```
always @(posedge clk, posedge rst) begin
    if(rst) start <= 1'b0;
    else begin
        if(pause) start <= !start;
    end
end
```

For the counter, I make two reg, **cnt_1** (right digit) and **cnt_2** (left digit). If positive edge reset signal were triggered, both **cnt_1** and **cnt_2** will become 0, otherwise it will check the **start** signal. If it's 1, then it will go to the next state, otherwise it will still in the current state.

```
always @(posedge clk_div, posedge rst) begin
    if(rst) begin
        cnt_1 <= 4'd0;
        cnt_2 <= 4'd0;
    end else begin
        if(start) begin
            cnt_1 <= cnt_1_next;
            cnt_2 <= cnt_2_next;
        end else begin
            cnt_1 <= cnt_1;
            cnt_2 <= cnt_2;
        end
    end
end
```

To assign the next state of **cnt_1** (**cnt_1_next**) and **cnt_2 (cnt_2_next)**. First, need to check the counting direction is up or down.

- Counting upward
  If the right number is not 9, then the number will be added by one. Otherwise, check if the left number number is 9 or not. If it's not 9 then the right number will become 0 and the left number will be added by one. And it it's 9 then it will not change.
- Counting downward
  It's similar with counting upward, just the opposite of it.

```
always @(*) begin
    cnt_1_next = cnt_1;
    cnt_2_next = cnt_2;
    if(!cnt_down) begin //count up
        if(cnt_1 != 4'd9) cnt_1_next = cnt_1 + 4'd1;
        else begin
            if(cnt_2 != 4'd9) begin
                cnt_1_next = 4'd0;
                cnt_2_next = cnt_2 + 4'd1;
            end
        end
    end else begin //count down
        if(cnt_1 != 4'd0) cnt_1_next = cnt_1 - 4'd1;
        else begin
            if(cnt_2 != 4'd0) begin
                cnt_1_next = 4'd9;
                cnt_2_next = cnt_2 - 4'd1;
            end
        end
    end
end
```

To control the counting speed, check if **faster** (speed_up one pulse signal) or **slower** (speed_down one pulse signal) signal were triggered. If **faster** were triggered and **speed** not in the max speed, then **speed_next** will be added by one, same if **slower** were triggered and it's not in the slowest speed, **speed_next** will be decreased by one. If posedge reset signal were triggered, **speed** will change to the slowest speed, 0.

```
always @(posedge clk, posedge rst) begin
    if(rst) speed <= 2'd0;
    else begin
        speed <= speed_next;
    end
end
```

```
always @* begin
    speed_next = speed;
    if(faster) begin
        case (speed)
            2'd0 : speed_next = 2'd1;
            2'd1 : speed_next = 2'd2;
        endcase
    end
    if(slower) begin
        case (speed)
            2'd1 : speed_next = 2'd0;
            2'd2 : speed_next = 2'd1;
        endcase
    end
end
```

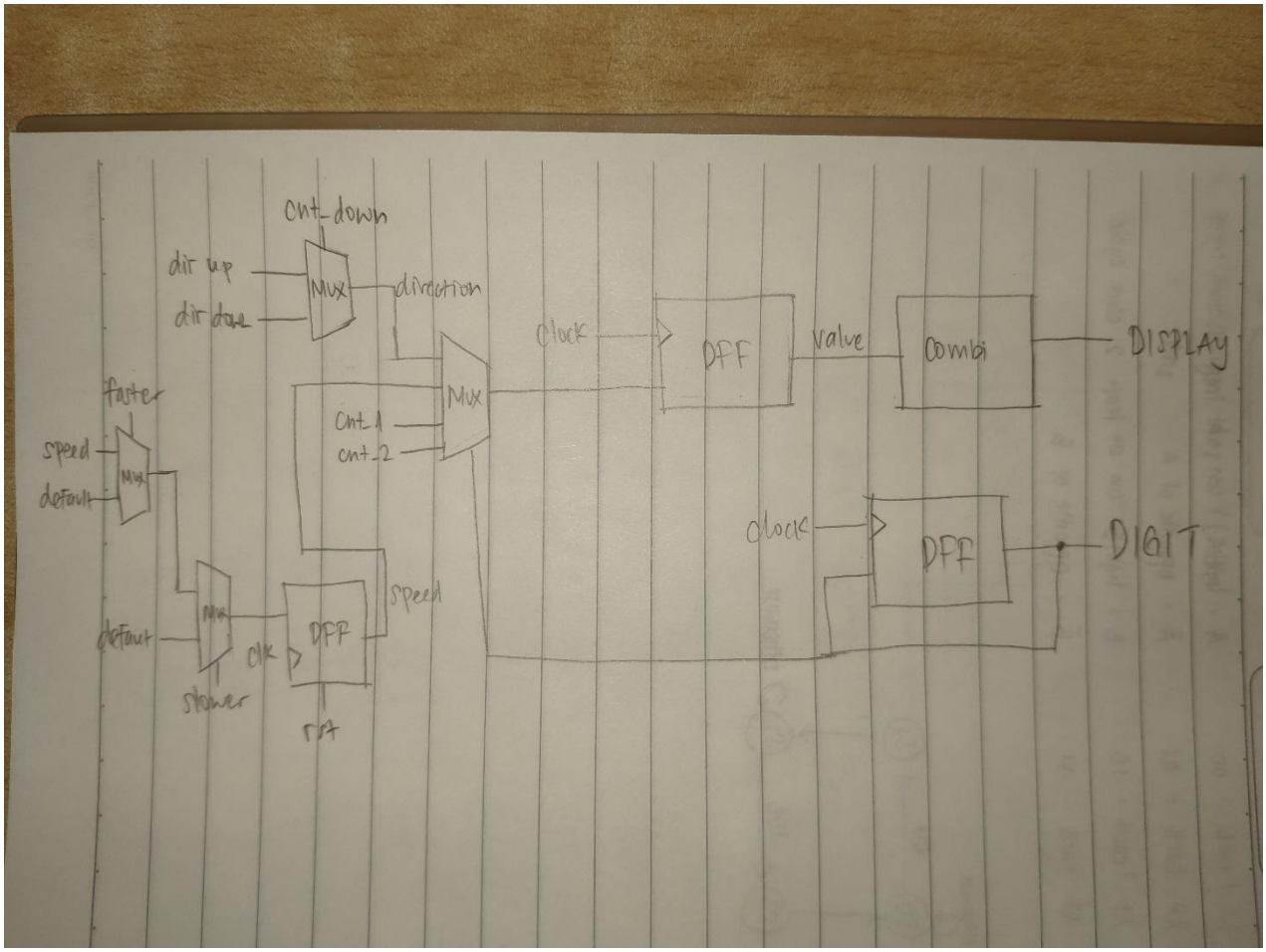
max (LED0): 1 if the counter reaches the largest number (99), and 0 otherwise.

min (LED1): 1 if the counter reaches the smallest number (00), and 0 otherwise.

LED0 will on if both **cnt_1** and **cnt_2** are 9.
LED1 will on if both **cnt_1** and **cnt_2** are 0.

```
assign max = (cnt_1 == 4'd9 && cnt_2 == 4'd9) ? 1'b1 : 1'b0;
assign min = (cnt_1 == 4'd0 && cnt_2 == 4'd0) ? 1'b1 : 1'b0;
```

Last is the **DISPLAY** and **DIGIT**. I also use the code from professor's slide to do this lab. And the block diagram is shown as below.



For the speed value, I assign it decimal 0, 1 and 2. 0 is the slowest and 2 is the fastest. I instantiate module for the clock speed in advance. And use multiplexer to chose the clock speed.

```
clock_divider_2s #(28) speed0 (.clk(clk), .clk_div(clk0));
clock_divider_1s #(27) speed1 (.clk(clk), .clk_div(clk1));
clock_divider_half #(26) speed2 (.clk(clk), .clk_div(clk2));
clock_divider #(13) clock_div (.clk(clk), .clk_div(clock));
```

```
always @* begin
    clk_div = clk0;
    case (speed)
        2'd0 : clk_div = clk0;
        2'd1 : clk_div = clk1;
        2'd2 : clk_div = clk2;
    endcase
end
```

- lab4_2
  I also instantiate debounce and onepulse module for all button, **enter**, **input_number**, **count_down**, except for reset button.

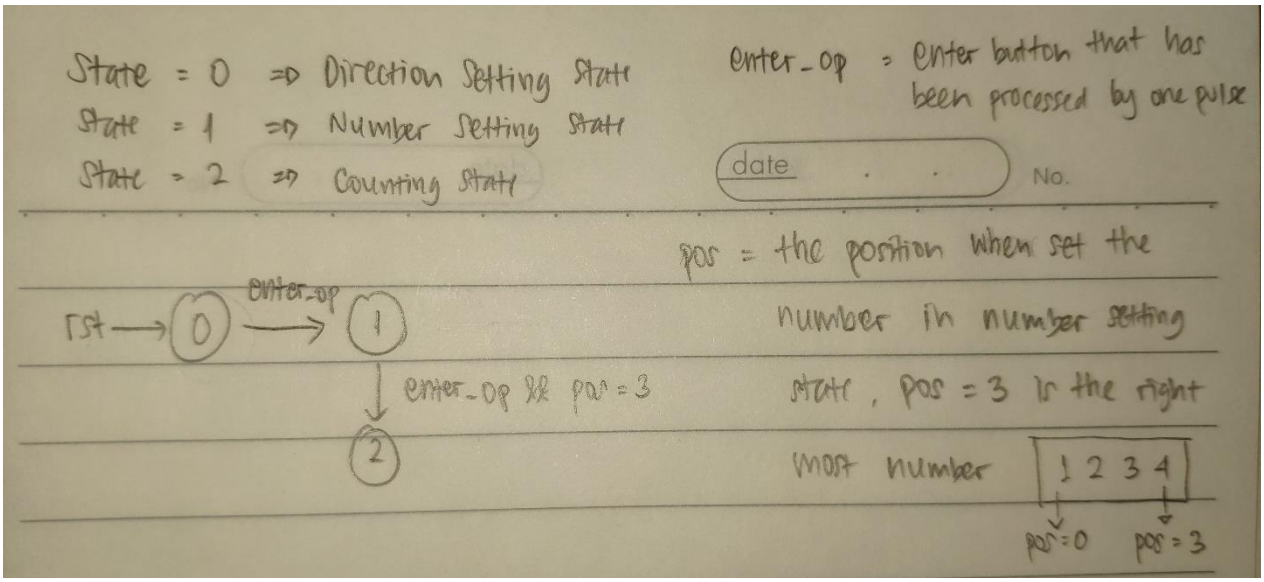
```
debounce countdown_db (.pb_debounced(cnt_down_db), .pb(count_down), .clk(clk));
onepulse countdown_op (.pb_debounced(cnt_down_db), .clk(clk), .pb_1pulse(cnt_down_op));

debounce enter_button_db (.pb_debounced(enter_db), .pb(enter), .clk(clk));
onepulse enter_button_op (.pb_debounced(enter_db), .clk(clock), .pb_1pulse(enter_op));

debounce input_num_db (.pb_debounced(num_sel_db), .pb(input_number), .clk(clk));
onepulse input_num_op (.pb_debounced(num_sel_db), .clk(clk_ds), .pb_1pulse(num_sel_op));
```

  In this lab, the **count_down** button is similar to **en** button in the lab4_1. To keep in track the direction is up or down. And **led0** will on the direction is to count down (**cnt_down** signal is 1), otherwise the light is off.

```
always @(posedge clk or posedge rst) begin
    if(rst) cnt_down <= 1'b0;
    else begin
        if(cnt_down_op) cnt_down <= !cnt_down;
    end
end
```

```
assign led0 = (cnt_down) ? 1'b1 : 1'b0;
```

  For FSM, I design it as below.




```
always @(posedge clock, posedge rst) begin
    if(rst) begin
        state <= 2'd0;
        pos <= 2'd0;
        dir <= 1'd0;
    end else begin
        state <= next_state;
        pos <= next_pos;
        dir <= next_dir;
    end
end
```

  dir is the counting direction, signal 1 to count down, 0 to count up.
  - **state** = 0 (direction setting state)
    This state is to decide the counting direction. If **enter_op** signal were triggered, it will go

to number setting state with the current direction.

- **state** = 1 (number setting state)
  Here I track which number is being set with **pos** reg (2-bit). **pos** = 0 means it sets the left most number. **pos** will be added by one if enter button is pressed. It will go to counting state only if the current position is 3 and enter button is pressed.

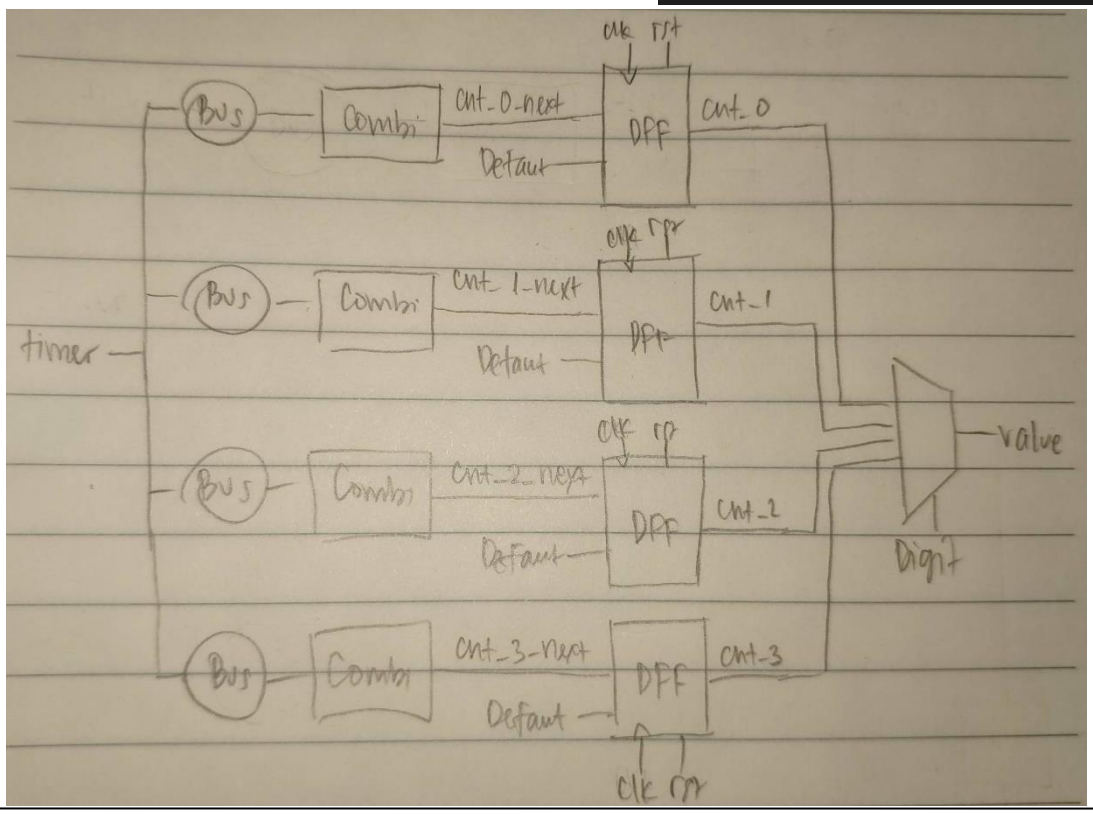
To set the number, it's also similar to lab4_1 counting. Just added **pos** to choose which number to set.

```
case (pos)
    2'd0 : cnt_0_next = (cnt_0 == 4'd1) ? 4'd0 : (cnt_0 + 4'd1);
    2'd1 : cnt_1_next = (cnt_1 == 4'd5) ? 4'd0 : (cnt_1 + 4'd1);
    2'd2 : cnt_2_next = (cnt_2 == 4'd9) ? 4'd0 : (cnt_2 + 4'd1);
    2'd3 : cnt_3_next = (cnt_3 == 4'd9) ? 4'd0 : (cnt_3 + 4'd1);
endcase
```

Next is how I do the counting. First, I convert the number that has been set in the number setting state (I also store it to **goal** reg to use it in counting up) to desi second. Then I add it or subtract it by one depends on the direction status. After that I convert each digit to **cnt_0**, **cnt_1**, **cnt_2**, **cnt3** using mod and division.

```
if(en) begin
    if(dir) begin //count down
        if(timer > 11'd0) begin
            cnt_0_next = timer / 600;
            cnt_1_next = (timer % 600) / 100;
            cnt_2_next = (timer % 100) / 10;
            cnt_3_next = timer % 10;
            timer_next = timer - 11'd1;
        end else begin
            cnt_0_next = timer / 600;
            cnt_1_next = (timer % 600) / 100;
            cnt_2_next = (timer % 100) / 10;
            cnt_3_next = timer % 10;
            timer_next = timer;
        end
```

```
end else begin //count up
    if(timer < goal) begin
        cnt_0_next = timer / 600;
        cnt_1_next = (timer % 600) / 100;
        cnt_2_next = (timer % 100) / 10;
        cnt_3_next = timer % 10;
        timer_next = timer + 11'd1;
    end else begin
        cnt_0_next = timer / 600;
        cnt_1_next = (timer % 600) / 100;
        cnt_2_next = (timer % 100) / 10;
        cnt_3_next = timer % 10;
        timer_next = timer;
    end
end
```



2. 學到的東西與遇到的困難

I spend a lot of time in doing the bonus part. I was so confuse why the code in the left side didn't work.

```
module clock_ds (clk, clk_div);
    parameter n = 24;
    input clk;
    output clk_div;
    reg [n-1 : 0] num = 0;
    wire [n-1 : 0] next_num;
    reg flag = 0;
    always@(posedge clk) begin
        num <= next_num;
    end
    assign next_num = (num != 25'd5000000) ? (num + 1) : 26'd0;
    assign clk_div = (num >= 25'd10000000) ? 1'b1 : 1'b0;
endmodule
```

It count really fast, not in 0.1 second. When I check the wave form, it shows a 0.1second waveform. But when I run in the board, it keep going so fast. However, when I change the code become the right side, it works perfectly.

```
module clock_ds (clk, clk_div);
    parameter n = 24;
    input clk;
    output clk_div;
    reg [n-1 : 0] num = 0;
    wire [n-1 : 0] next_num;
    reg flag = 0;
    always@(posedge clk) begin
        if(num == 25'd5000000) begin
            flag <= ~flag;
            num <= next_num;
        end else if(num == 25'd10000000) begin
            flag <= ~flag;
            num <= 0;
        end else begin
            flag <= flag;
            num <= next_num;
        end
    end
    assign next_num = num + 1;
    assign clk_div = flag;
endmodule
```

Later on, when the professor explain in the class, I just knew that the output from sequential and combinational is matter in this case. It make sense since sequential output always stable and updated depend on the clock cycle.

3. 想對老師或助教說的話