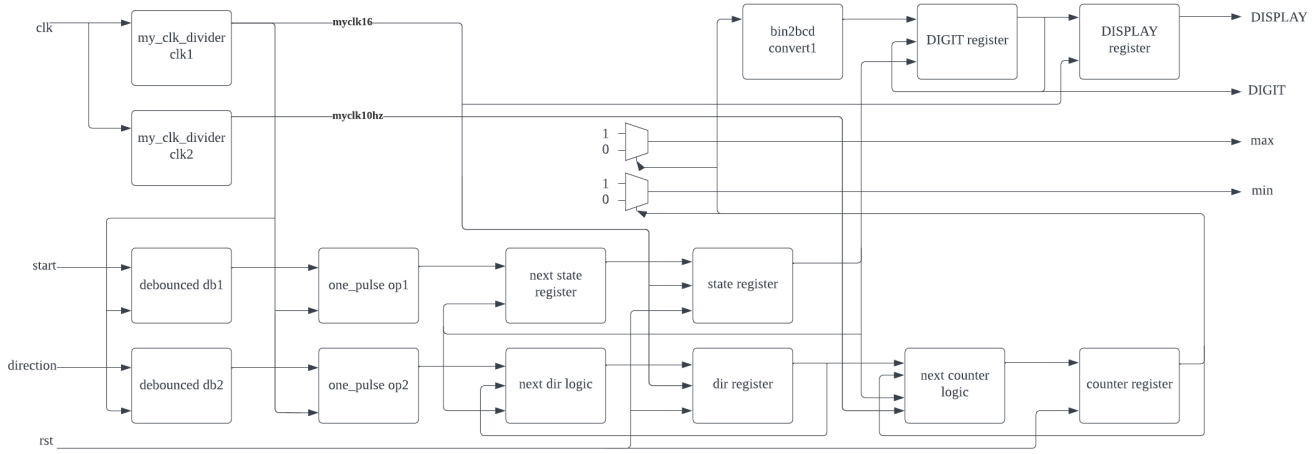
# Lab 4

學號: 110062131 姓名: 馬毓昇

## A. Lab Implementation

Lab4_1: 支援加減變換的 7 段顯示計數器。




Lab4_1 segment:

1. binary counter: 一般的 counter 寫法，但有用 dir 判斷加減，最大加到 999，最小減到 0。為了符合規格要求，以 slow clk（10hz）更新。

```
// binary counter
always @(posedge myclk10hz or posedge rst) begin
    if(rst) begin
        counter <= START;
    end
    else begin
        counter <= next_counter;
    end
end
// binary counter
always @* begin
    case(state)
        INIT : begin
            next_counter = START;
        end
        CNT : begin
            if(dir == 1'b0) begin
                if(counter == MAX) begin
                    next_counter = counter;
                end
                else begin
                    next_counter = counter + 1;
                end
            end
            else begin
                if(counter == MIN) begin
                    next_counter = counter;
                end
                else begin
                    next_counter = counter - 1;
                end
            end
        end
        STOP : begin
            next_counter = counter;
        end
    endcase
end
```

2. 10 bits binary to 4*4 bits BCD converter: 將 binary 的 counter 轉成 4*4 bits 的 BCD。運用 ”double dabble” 演算法，實作上會跑過每一個 binary bit，先判斷各 BCD 區，如果大於等於 5 的話就加 3，最後把 binary 的 bits 左移一位進 BCD 區，直到 binary bits 全部被移進 BCD 區，不再判斷。

```
13  wire [15:0] led;
```

```
31  bin2bcd convert1 (counter, led);
```

```
215 module bin2bcd (
216     input [9:0] bin,
217     output reg [15:0] bcd
218 );
219
220 integer i;
221
222 always @(bin) begin
223     bcd = 0;
224     for(i=0; i<10; i=i+1) begin
225         if(bcd[3:0] >= 5) begin
226             bcd[3:0] = bcd[3:0] + 3;
227         end
228         if(bcd[7:4] >= 5) begin
229             bcd[7:4] = bcd[7:4] + 3;
230         end
231         if(bcd[11:8] >= 5) begin
232             bcd[11:8] = bcd[11:8] + 3;
233         end
234         if(bcd[15:12] >= 5) begin
235             bcd[15:12] = bcd[15:12] + 3;
236         end
237         bcd = {bcd[14:0], bin[9-i]};
238     end
239 end
240
241 endmodule
```

3. 7 segment display: BCD0、BCD1、BCD2 直接用剛剛轉好的 BCD 來接；BCD3 則是用 state 與 dir 還有 direction 按下與否來判斷要呈現上、下、或一橫。4 碼 7 段顯示的部分用跑馬燈的方式分別更新，同時更新 value 至對應的 BCD 值。後面依據各 BCD 來編碼右邊三碼 7 段顯示與左邊一碼的加減狀態顯示，而依眼睛視覺暫留選擇合適 clk rate，所以用 fast clk（100Mhz/2^16）來更新。

```
52  wire [3:0] BCD0 = led[3:0];
53  wire [3:0] BCD1 = led[7:4];
54  wire [3:0] BCD2 = led[11:8];
55  wire [3:0] BCD3 = state == CNT ? dir == 0 ? 4'd10 : 4'd11 :
56                    direction_debounced == 1'b1 ? dir == 0 ? 4'd10 : 4'd11 : 4'd12;
```

```
135 // 7 segment display
136 always @(posedge myclk16) begin
137     case (DIGIT)
138         4'b1110 : begin
139             value = BCD1;
140             DIGIT = 4'b1101;
141         end
142         4'b1101 : begin
143             value = BCD2;
144             DIGIT = 4'b1011;
145         end
146         4'b1011 : begin
147             value = BCD3;
148             DIGIT = 4'b0111;
149         end
150         4'b0111 : begin
151             value = BCD0;
152             DIGIT = 4'b1110;
153         end
154         default : begin
155             value = BCD0;
156             DIGIT = 4'b1110;
157         end
158     endcase
159 end
```

```
// 7 segment display
always @* begin
    if(state == INIT) begin
        DISPLAY = 7'b011_1111;
    end
    else if(state == CNT || state == STOP) begin
        case(value)
            4'd0 : DISPLAY = 7'b100_0000;
            4'd1 : DISPLAY = 7'b111_1001;
            4'd2 : DISPLAY = 7'b010_0100;
            4'd3 : DISPLAY = 7'b011_0000;
            4'd4 : DISPLAY = 7'b001_1001;
            4'd5 : DISPLAY = 7'b001_0010;
            4'd6 : DISPLAY = 7'b000_0010;
            4'd7 : DISPLAY = 7'b111_1000;
            4'd8 : DISPLAY = 7'b000_0000;
            4'd9 : DISPLAY = 7'b001_0000;

            4'd10 : DISPLAY = 7'b101_1100; // UP
            4'd11 : DISPLAY = 7'b110_0011; // DOWN
            4'd12 : DISPLAY = 7'b011_1111; // PAUSE

            default : DISPLAY = 7'b111_1111;
        endcase
    end
end
```

4. dir: 用 direction_1pulse 切換加減狀態，next_dir = !dir 可以實現 0、1 狀態切換，0 是 UP、1 是 DOWN。以 fast clk 更新使操作反應順暢。

```
reg dir = 0; // 0 for UP, 1 for DOWN
reg next_dir;
//  dir
always @(posedge myclk16 or posedge rst) begin
    if(rst) begin
        dir <= 1'b0;
    end
    else begin
        dir <= next_dir;
    end
end
// dir
always @* begin
    case(state)
        INIT : begin
            next_dir = 1'b0;
        end
        CNT : begin
            next_dir = dir;
            if(direction_1pulse == 1'b1) begin
                next_dir = !dir;
            end
        end
        STOP : begin
            next_dir = dir;
        end
    endcase
end
```

5. clock_divider: 除數不局限於 2 的冪次的 clock divider。以 counter 邏輯實作，當 counter < DIVISOR/2 輸出 high，不然就輸出 low。

```
wire myclk10hz, myclk16;
my_clock_divider #(28'd10000000) clk1 (clk, myclk10hz);
my_clock_divider #(28'd65536) clk2 (clk, myclk16);
```

```
module my_clock_divider #(
    parameter DIVISOR = 28'd2
)(
    input clock_in,
    output reg clock_out
);

reg [27:0] counter = 28'd0;

always @(posedge clock_in) begin
    counter <= counter + 28'd1;
    if(counter >= (DIVISOR-1)) begin
        counter <= 28'd0;
    end
    clock_out <= (counter < DIVISOR/2) ? 1'b1 : 1'b0;
end

endmodule
```

6. debounce、one_pulse: 都是用上課的模板。以 fast clk 更新使操作反應順暢。debounce 需要連吃 4 個 high 才會輸出 high，不然輸出 low，以此消弭那些只有幾個 clk 的雜訊。one_pulse 偵測前一輸入是 0 現在是 1 的那個 clk 輸出脈衝，去除長按按鈕的誤差。

```
module debounce (
        input wire clk,
        input wire pb,
        output wire pb_debounced
);
        reg [3:0] shift_reg;

        always @(posedge clk) begin
                shift_reg[3:1] <= shift_reg[2:0];
                shift_reg[0] <= pb;
        end

        assign pb_debounced = ((shift_reg == 4'b1111) ? 1'b1 : 1'b0);

endmodule
```

```
module one_pulse (
    input wire clk,
    input wire pb_in,
    output reg pb_out
);

        reg pb_in_delay;

        always @(posedge clk) begin
                if (pb_in == 1'b1 && pb_in_delay == 1'b0) begin
                        pb_out <= 1'b1;
                end else begin
                        pb_out <= 1'b0;
                end
        end

        always @(posedge clk) begin
                pb_in_delay <= pb_in;
        end
endmodule
```

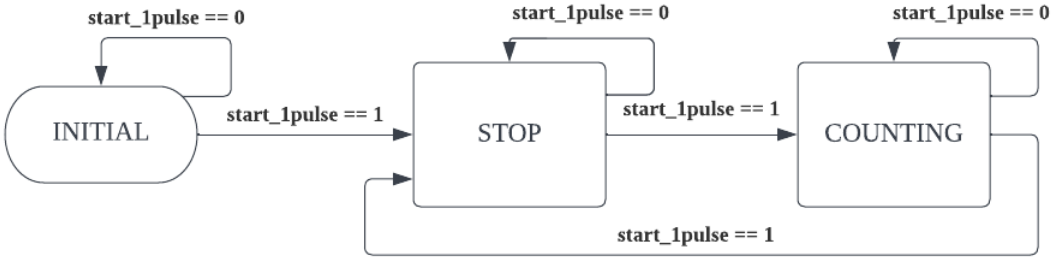
7. max、min: MUX 判斷 counter 值來決定 high or low。

```
assign max = counter == MAX ? 1 : 0;
assign min = counter == MIN ? 1 : 0;
```

8. state: 吃到 start_1pulse 就更新狀態，以 fast clk 更新使操作反應順暢。

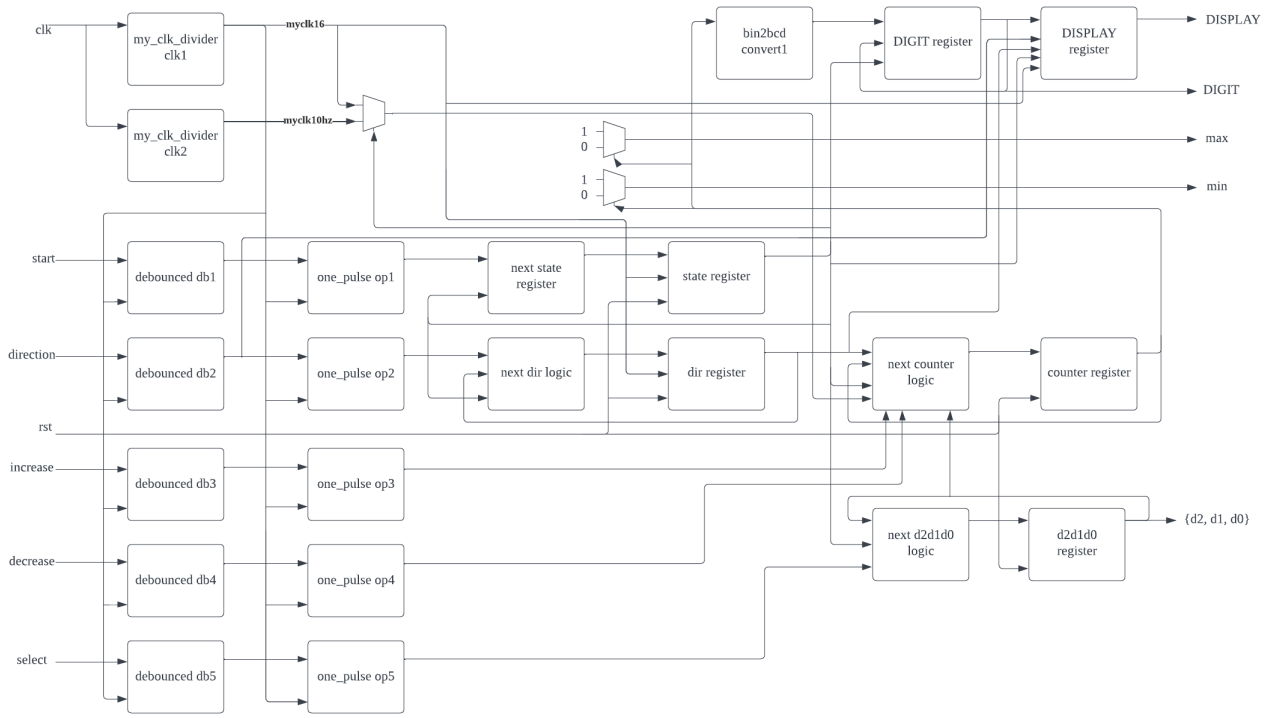
```
// state
always @(posedge myclk16 or posedge rst) begin
    if(rst) begin
        state <= INIT;
    end
    else begin
        state <= next_state;
    end
end
// state
always @* begin
    case(state)
        INIT : begin
            if(start_1pulse) begin
                next_state = STOP;
            end
            else begin
                next_state = INIT;
            end
        end
        STOP : begin
            if(start_1pulse) begin
                next_state = CNT;
            end
            else begin
                next_state = STOP;
            end
        end
        CNT : begin
            if(start_1pulse) begin
                next_state = STOP;
            end
            else begin
                next_state = CNT;
            end
        end
    endcase
end
```

Lab4_1 FSM:




1. INIT -> INIT: 沒按開始就不變。
2. INIT -> STOP: 按開始就進 STOP。
3. STOP -> STOP: 沒按開始就不變。
4. STOP -> CNT: 按開始就進 CNT。
5. CNT -> CNT: 沒按開始就不變。
6. CNT -> STOP: 按開始就進 STOP。

Lab4_2: 支援選擇位數加減調整與加減變換的 7 段顯示計數器。




Lab4_2 segment:

1. clkmux: 因為在 STOP state 繼續用 10hz 的 slow clk 的話，會跟不上 fast clk 的操作頻率而吃不到變化，所以用一個 MUX 來在兩個 clk 中切換。換成 fast clk 既可以吃到操作，反應上也會順暢。

```
54  wire myclk10hz, myclk16, clkmux;
55  my_clock_divider #(28'd10000000) clk1 (clk, myclk10hz);
56  my_clock_divider #(28'd65536) clk2 (clk, myclk16);
57  assign clkmux = state == STOP ? myclk16 : myclk10hz;
```

```
118 // binary counter
119 always @(posedge clkmux or posedge rst) begin
120     if(rst) begin
121         counter <= START;
122     end
123     else begin
124         counter <= next_counter;
125     end
126 end
```

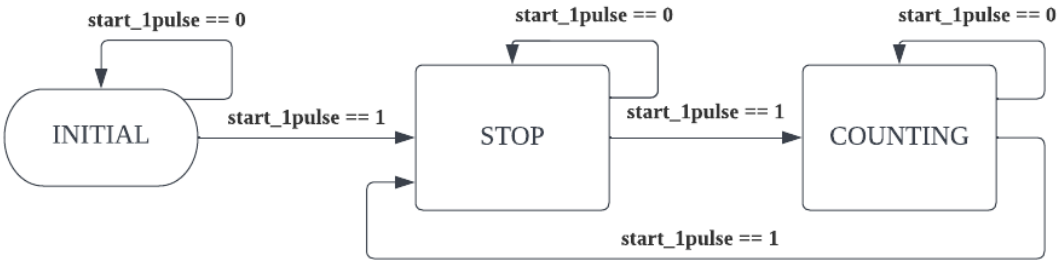
2. increase、decrease: 先決定要調整哪位數，再用除 10 的冪次+取餘的方式判斷 base10 下的那位數值做變化。個位數 9 變 0 可以看做是 counter + (-9)，其他位數同理。

```
STOP : begin
    next_counter = counter;
    if(increase_1pulse == 1'b1) begin
        case({d2, d1, d0})
            3'b001 : begin
                if(counter % 10 != 9) begin
                    next_counter = counter + 1;
                end
                else begin
                    next_counter = counter - 9;
                end
            end
            3'b010 : begin
                if((counter / 10) % 10 != 9) begin
                    next_counter = counter + 10;
                end
                else begin
                    next_counter = counter - 90;
                end
            end
            3'b100 : begin
                if((counter / 100) % 10 != 9) begin
                    next_counter = counter + 100;
                end
                else begin
                    next_counter = counter - 900;
                end
            end
            default : begin
                next_counter = counter;
            end
        endcase
    end
```

3. {d2, d1, d0}: 像跑馬燈一樣，每吃到一次 select_1pulse 就換。fast clk 更新使操作順暢。

```
296 //  d light
297 always @(posedge myclk16 or posedge rst) begin
298     if(rst) begin
299         {d2, d1, d0} <= 3'b000;
300     end
301     else begin
302         {d2, d1, d0} <= next_dlight;
303     end
304 end
305 // d light
306 always @* begin
307     case(state)
308         INIT : begin
309             next_dlight = 3'b000;
310         end
311         CNT : begin
312             next_dlight = 3'b000;
313         end
314         STOP : begin
315             next_dlight = {d2, d1, d0};
316             if({d2, d1, d0} == 3'b000) begin
317                 next_dlight = 3'b001;
318             end
319             if(select_1pulse == 1'b1) begin
320                 if({d2, d1, d0} == 3'b000) begin
321                     next_dlight = 3'b001;
322                 end
323                 else if({d2, d1, d0} == 3'b001) begin
324                     next_dlight = 3'b010;
325                 end
326                 else if({d2, d1, d0} == 3'b010) begin
327                     next_dlight = 3'b100;
328                 end
329                 else if({d2, d1, d0} == 3'b100) begin
330                     next_dlight = 3'b001;
331                 end
332             end
333         end
334     endcase
335 end
```

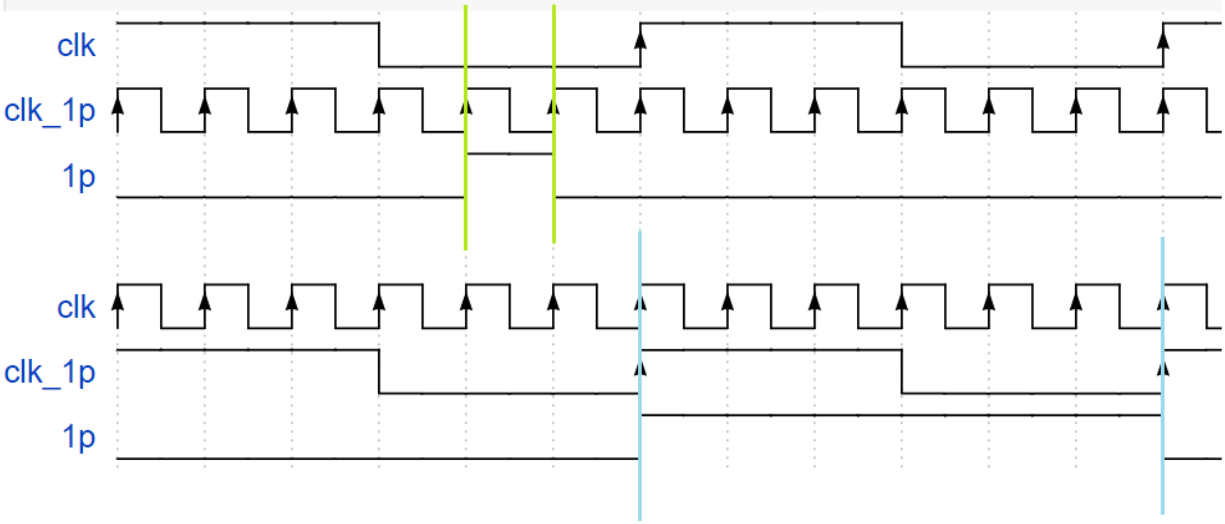
Lab4_2 FSM:




## B. Questions and Discussions

A: 按鈕內部是像彈簧一樣，壓放的時候總會有隨意的 μs 尺度彈跳訊號產生，又因為 FPGA 對脈衝的靈敏度有達到 ns 尺度，所以需要 debounce 來消除這些不希望出現的脈衝訊號。不需要，因為 debounce 負責的是穩定訊號，one-pulse 是依據訊號產生脈衝。debounce 的 clk 影響的是穩定訊號時對於原始訊號的靈敏度（clk 越快，越接受細碎的訊號），one-pulse 的 clk 影響的則是產生的脈衝長度（clk 愈慢，一次脈衝的時間越久），與穩定訊號的輸入的無關，兩者間並不一定要使用同個 clk，只是在合理的範圍內（不合理：用 100Mhz 跑 debounce，這樣根本沒有 debounce 到；或是 clk 慢到 one-pulse 吃不到輸入訊號）都能選用各種 clk rate。

B: 如果比較快的話，有可能會讓 FSM 沒反應到，於是就忽略了這次變化（如下圖綠框，沒有變化產生）；而比較慢的話，就有可能讓 FSM 因為一次脈衝而變化不只一次（如下圖藍框，其中每個 posedge clk 都會變化）。這都不是理想中的結果，應該要一次脈衝產生一次變化。




C: 因為有視覺暫留，當燈號的 4bits 依序變動得夠快，我們就會把他們當成是在同一時間一起變動的。

## C. Problem Encountered

1. 想破頭都想不出來 Question A 第二題要怎麼證明我的答案，最後只能盡力用文字敘述表達我的想法。

2. lab 中小卡的一點就是 4_2 要上 increase/decrease 變化那邊，一開始沒注意到 clk 的問題還另外接了 led 測試是不是 button 接觸不良，測試時看到 led 瞬間閃爍就馬上給了我一個想法，會不會是 button 的操作（fast clk）相對於 counter 的變化（10 hz slow clk）太快，最後發現還真的是，一下子就改成功了。

## D. Suggestions

我發現用 Lucidchart 畫圖很方便，說不定老師上課可以推薦給同學知道。(gapp, office365 信箱可以有免費 premium)