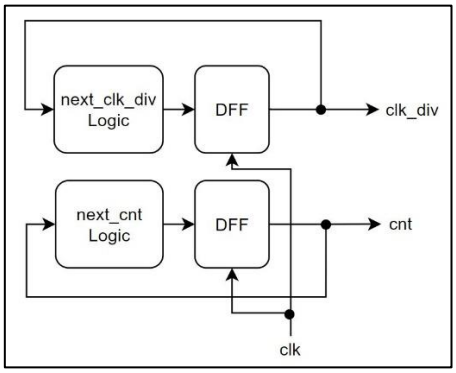
Lab 4

學號: 110011138 姓名: 楊立慈

A. Lab Implementation

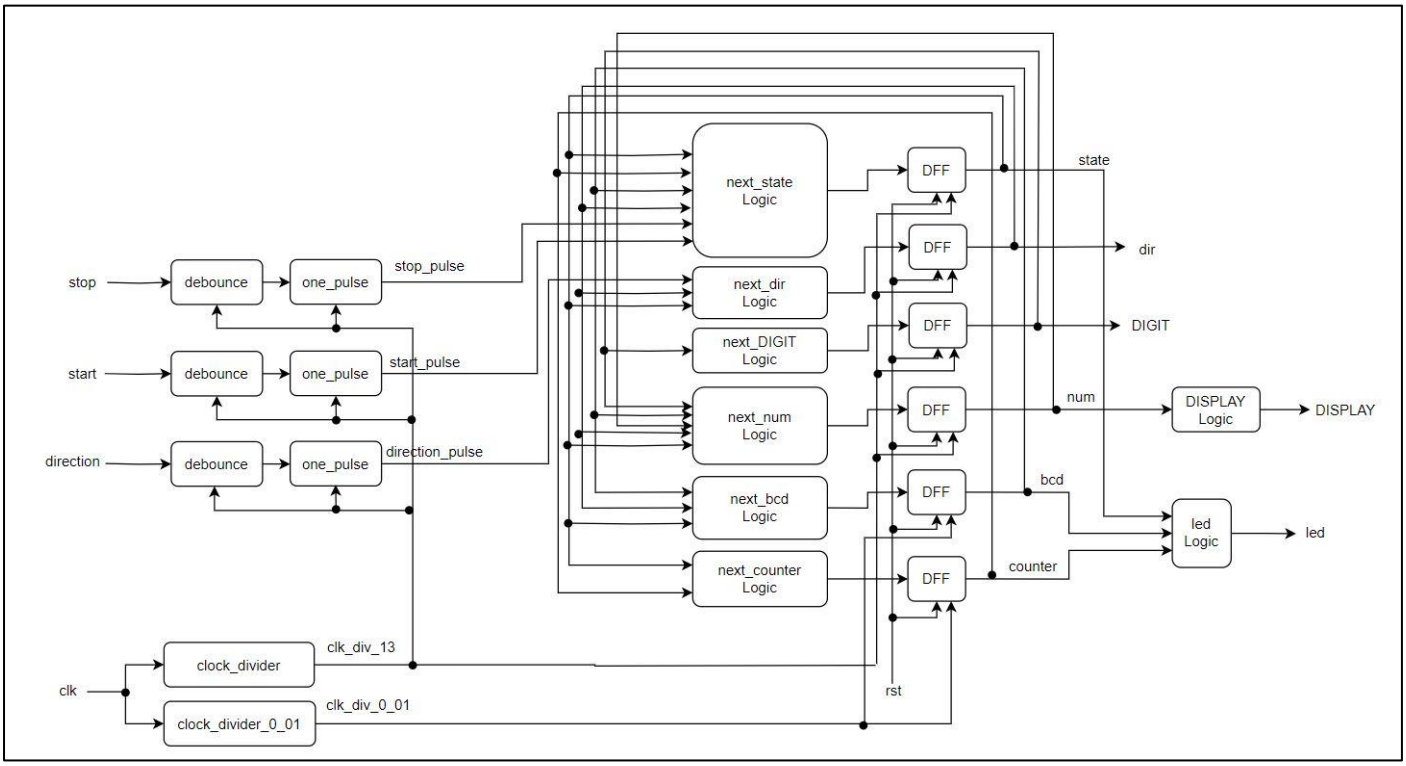
1. Lab4_1

下左圖為 lab4_1 中的 clock_divider_0_01 module 的 block diagram，其功能為將 100MHZ 的 clk 轉為 period 為 0.01s 的 clk_div signal。利用 cnt 從 0 開始數到 499_999 時再重新從 0 開始，並以 clk 作為更新 cnt 的頻率，藉此讓 clk_div 每當 cnt 數了 500_000 個 clk 的 cycle 後會 invert 自己原本的值。核心部分 next_clk_div 和 next_cnt 的 Logic 的 code 如下右圖所示，基本上當 cnt 等於 LIMIT(499_999)時下一個 cnt 會歸零、clk_div 會 invert，不然就正常 cnt + 1、維持 clk_div。




```
always @(*) begin
    if (cnt == LIMIT) begin
        next_cnt = 19'd0;
        next_clk_div = ~clk_div;
    end
    else begin
        next_cnt = cnt + 1'b1;
        next_clk_div = clk_div;
    end
end
```

下圖為 lab4_1 的 block diagram。利用 clock_divider、clock_divider_0_01 modules 做出了兩個 clock signal 並分別用來 drive 不同的 DFF，如圖所示。再利用提供的 debounce、one_pulse modules 處理 button 的輸入訊號成 one-pulse 的形式。其中 dir 為使用者設定的方向、num 為 DISPLAY 用的值、counter 用來數 0.01 的倍數秒、bcd 用來儲存數到的數字。

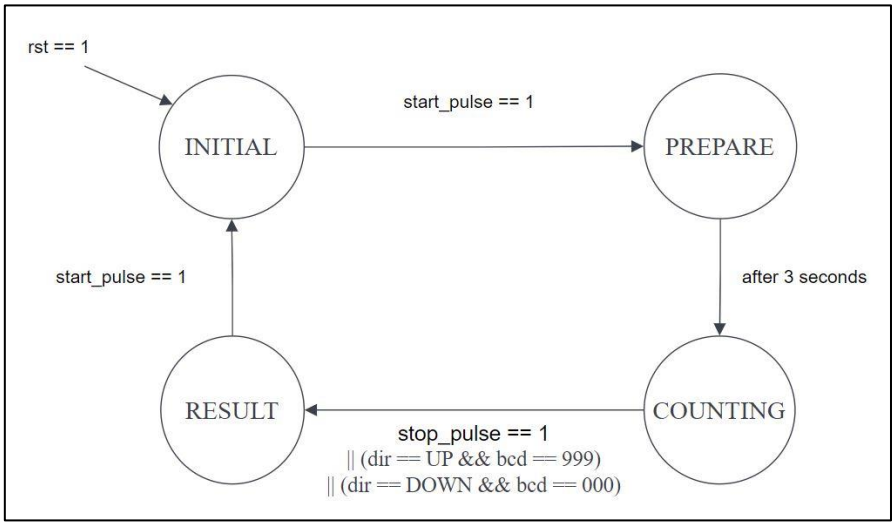


下左圖為 block diagram 中 next_state、next_bcd、next_num、next_counter 等訊號的 Combinational Logic 的代碼實作(以 PREPARE state 為例)。由於在此 state 要等 3 秒才能進入 COUNTING state，所以我利用 counter 從 0 數到 299 並以 clk_div_0_01 的頻率更新 counter，以此來數 3 秒，並在 counter == 299 時更新 next_state 為 COUNTING。同時，我們針對設定不同的 dir 給予相對應的初始 bcd 值。由於我利用 num 訊號來做為目前 7-segment display 的 active digit 所應該印出來的值，因此針對不同 active digit 要給不同的值作為下一個 active digit 的值。下圖右為 block diagram 中 DISPLAY Logic 的代碼，可看到我們依據上述的 num 值來決定目前 active 的 Digit 有哪些 segment 應該亮(對應到 0)。

```
PREPARE: begin
    // next_state
    if (counter == 9'd299) next_state = COUNTING;

    // next_bcd
    if (dir == UP) next_bcd = 10'd0;
    else next_bcd = 10'd999;

    // next_num
    case (DIGIT)
        4'b1110: next_num = D_OFF;
        4'b1101: next_num = D_OFF;
        4'b1011: next_num = D_P;
        4'b0111: next_num = D_OFF;
    endcase

    // next_counter
    if (counter == 9'd299) next_counter = counter;
    else next_counter = counter + 1'b1;
end
```

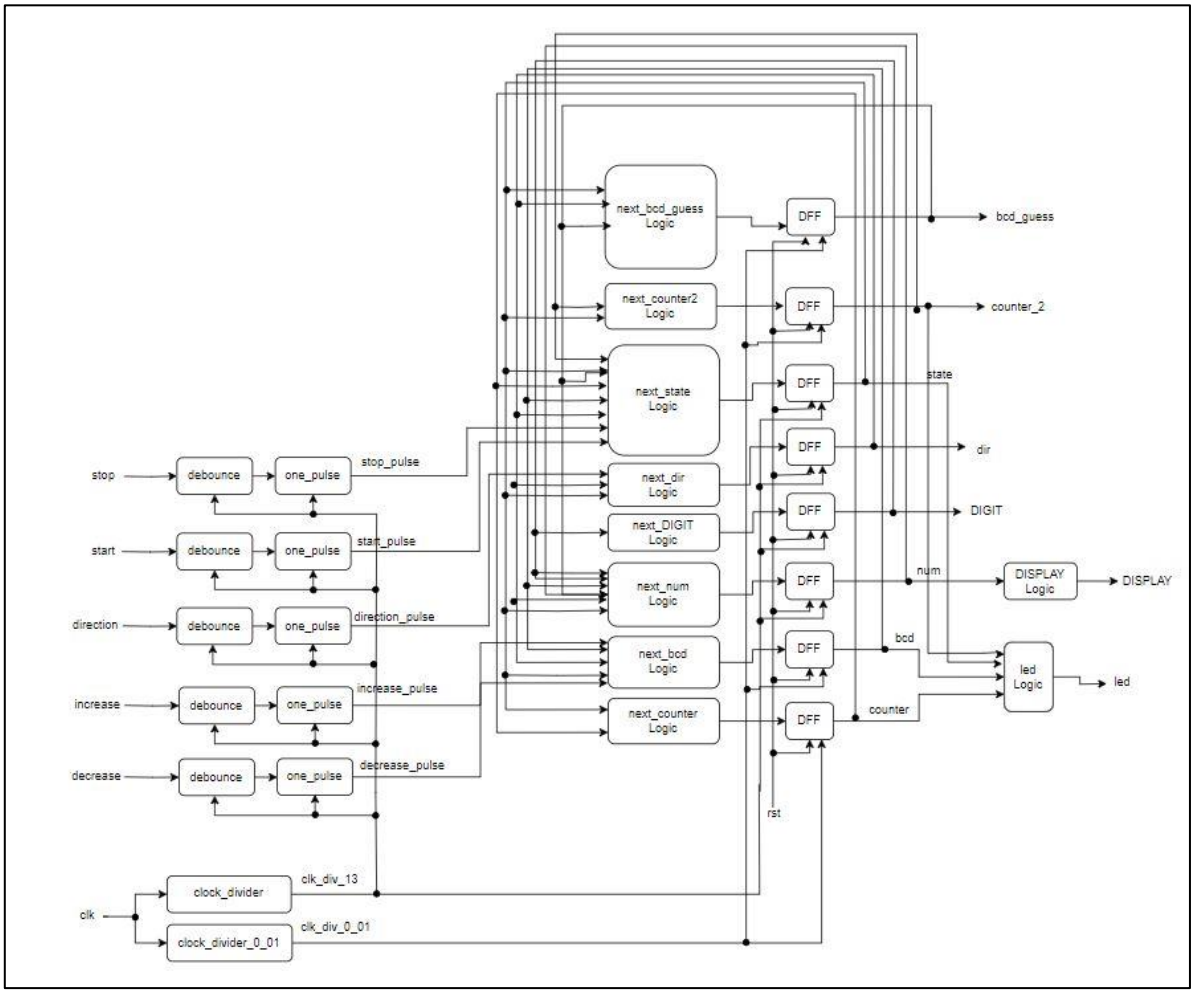
```
always @(*) begin
    DISPLAY = 7'b111_1111;
    case (num)
        4'd0: DISPLAY = 7'b100_0000;
        4'd1: DISPLAY = 7'b111_1001;
        4'd2: DISPLAY = 7'b010_0100;
        4'd3: DISPLAY = 7'b011_0000;
        4'd4: DISPLAY = 7'b001_1001;
        4'd5: DISPLAY = 7'b001_0010;
        4'd6: DISPLAY = 7'b000_0010;
        4'd7: DISPLAY = 7'b111_1000;
        4'd8: DISPLAY = 7'b000_0000;
        4'd9: DISPLAY = 7'b001_0000;
        D_DASH: DISPLAY = 7'b011_1111;
        D_UP: DISPLAY = 7'b101_1100;
        D_DOWN: DISPLAY = 7'b110_0011;
        D_P: DISPLAY = 7'b000_1100;
        D_OFF: DISPLAY = 7'b111_1111;
    endcase
end
```

下圖為 lab4_1 的 state diagram。Reset 後會進入 INITIAL state，並等到 start_pulse 是 1(start 按鈕被按了)才進入 PREPARE state。在 PREPARE state 如前所述會利用 counter 數 3 秒後就直接到 COUNTING state。在 COUNTING state 我們會等到 stop_pulse 是 1(stop 按鈕被按了)或者 bcd 往上數數到極限(999)了或者 bcd 往下數數到極限(000)了，才進到 RESULT state。在 RESULT state 則會等到 stop 按鈕被按了就回到 INITIAL state。



2. Lab4_2

下圖為 lab4_2 的 block diagram，大致概念與 4_1 類似，不過多了 increase、decrease 的 one_pulse signal 以及 counter2、bcd_guess 訊號。其中 bcd_guess 為使用者在 COUNTING state 猜測的數字，bcd 則為使用者在 INITIAL state 設定的目標數字。



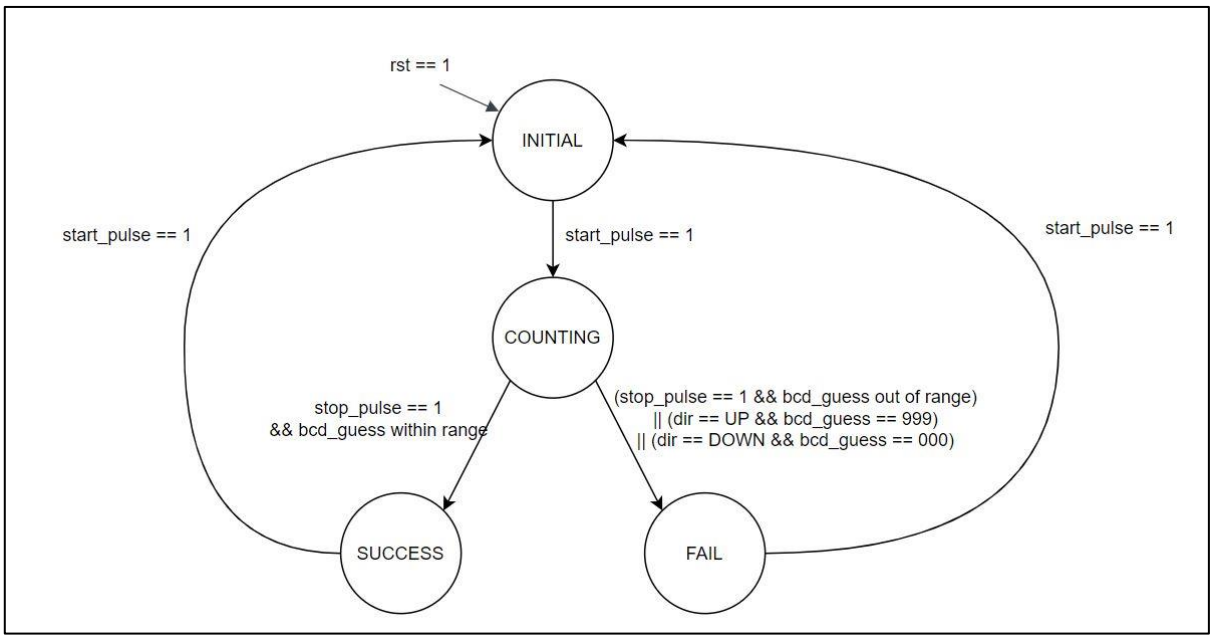
下圖為 next_bcd 在 INITIAL state 的代碼，也就是負責設定 increase/decrease digit 的部分，首先，我用了 tmp_bcd_d1、tmp_bcd_d2、tmp_bcd_d3 等 temp variable 來儲存原本 bcd 的個十百位數字。當 increase_pulse==1 時，我們逐一確認每個 Digit 是否需要增加，並確認如果原本該位數是 9 的話，增加會回到 0。最後在把個十百位數運算回 next_bcd。當 decrease_pulse==1 也是類似的概念。

```
tmp_bcd_d1 = bcd % 10;
tmp_bcd_d2 = (bcd / 10) % 10;
tmp_bcd_d3 = bcd / 100;
if (increase_pulse == 1'b1) begin
    if (Digit_1 == 1'b1) tmp_bcd_d1 = (bcd % 10 == 9) ? 10'd0 : (bcd % 10 + 10'd1);
    if (Digit_2 == 1'b1) tmp_bcd_d2 = ((bcd / 10) % 10 == 9) ? 10'd0 : ((bcd / 10) % 10 + 10'd1);
    if (Digit_3 == 1'b1) tmp_bcd_d3 = (bcd / 100 == 9) ? 10'd0 : (bcd / 100 + 10'd1);
    next_bcd = 100 * tmp_bcd_d3 + 10 * tmp_bcd_d2 + tmp_bcd_d1;
end
if (decrease_pulse == 1'b1) begin
    if (Digit_1 == 1'b1) tmp_bcd_d1 = (bcd % 10 == 0) ? 10'd9 : (bcd % 10 - 10'd1);
    if (Digit_2 == 1'b1) tmp_bcd_d2 = ((bcd / 10) % 10 == 0) ? 10'd9 : ((bcd / 10) % 10 - 10'd1);
    if (Digit_3 == 1'b1) tmp_bcd_d3 = (bcd / 100 == 0) ? 10'd9 : (bcd / 100 - 10'd1);
    next_bcd = 100 * tmp_bcd_d3 + 10 * tmp_bcd_d2 + tmp_bcd_d1;
end
```

由於其他 next_state、next_num、next_counter、next_dir、DISPLAY 等 code 與 lab4_1 類似，因此這裡就沒放了。比較不同的是在 led Logic，如下圖，這裡用到了 2 個 counter，這是由於我的 counter 是用 clk_div_0_01 去驅動的，但 FSM 是用 clk_div_13 的 clk 驅動，也就是說 counter 的 clk 比 FSM 的慢，因此在改變 state 時會發生來不及改變 counter 值去準備下一個 state 的數數的問題。因此我用了 2 個 counter 交替使用，一個 counter 正在當前 state 用時，就可以重置另一個 counter 準備下一個 state 從 0 開始數。

```
always @(*) begin
    led = 16'b0;
    case (state)
        INITIAL: led = 16'b1111_1111_1111_1111;
        COUNTING: begin
            if (counter < 9'd300) led = 16'b1111_1111_1111_1111;
            else led = 16'b0;
        end
        FAIL: begin
            if (counter2 / 100 == 1 || counter2 / 100 == 3 || counter2 / 100 == 5) led = 16'b0;
            else led = 16'b1111_1111_1111_1111;
        end
        SUCCESS: begin
            if (counter2 / 100 == 1 || counter2 / 100 == 3) led = 16'b0;
            else led = 16'b1111_1111_1111_1111;
        end
    endcase
end
```

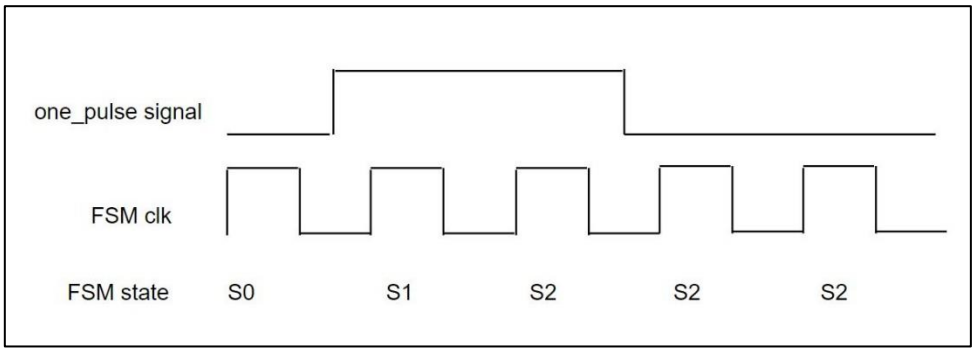
下圖為 lab4_2 的 state diagram。在 Reset 後會進到 INITIAL state，並在 start 按鈕被按下後進入 COUNTING state。在 COUNTING state 時，假如 stop 按鈕沒有被按，且 bcd_guess 往上數超過極限了(bcd_guess==999)或者 bcd_guess 往下數超過極限了(bcd_guess==000)，那就進入 FAIL state；假如使用者有按 stop 按鈕，則如果 bcd_guess 在目標 bcd 值的某個 range 內($bcd \pm 100$)的話，就進入 SUCCESS，否則就進入 FAIL。在 SUCCESS 跟 FAIL state 都同樣是在 start 按鈕被按下後回到 INITIAL state。



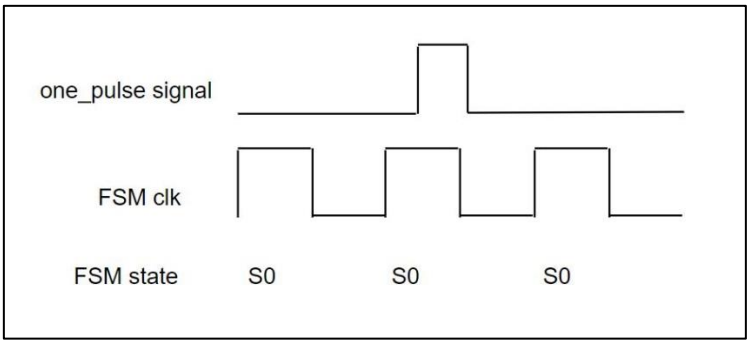
B. Questions and Discussions

1. 我們需要 debounce、one_pulse modules 是因為按鈕按下去時可能會在一段時間內有震動，造成短時間內訊號有 0 跟 1 不穩定的震盪。因此我們需要 debounce 把該震盪去除，得到較整齊穩定的訊號。而 one_pulse 是由於我們按按鈕想要的其實只是一個 cycle 的 1 的訊號，然而我們人手按按鈕不太可能準確的按好 1 個 cycle 後放開，因此我們需要 one_pulse 來把可能維持很多個 cycles 的 1 個訊號轉為只拉起來一個 cycle 的 1 的訊號。

   由於我們 FSM 切換 state 的判斷條件是利用 stop、start 等按鈕經過 one_pulse 處理後的訊號，因此 FSM 的 clock rate 必須要與 one_pulse 的 clock rate 相同。(假設 FSM 是 posedge 且會在按一次按鈕後進到下一個 state，然後假設 one_pulse signal 為該按鈕 one_pulse 後的訊號)若 one_pulse clock 比 FSM clock 還慢的話可能會發生如下圖的情況，也就是明明只按一次按鈕應該只會從 S0 到 S1，卻因為 one_pulse 的一個 cycle 太久，造成 FSM 以為按鈕按了兩次。



類似的，如果 one_pulse clock 比 FSM clock 還快的話可能會發生如下圖的情況，也就是按鈕的 one_pulse signal 不夠久，造成 FSM 還沒改變 state 那個 pulse 就沒了。



2. 第一個方法是以一個最快的 clk 為主，利用該 clk 以及 counters 來做出比較慢的 clock signal，並利用讓 counter 數到不同值來產生不同 rate 的 clock。本次 lab 我便是利用這種方法操作。並讓有不同 clk rate 需求的 DFF 各自用各自的 counter 去達到需要的 clock rate。

   第二個方法是同時有多個 clock signals，並依據目前的需求，利用 MUX 之類的選擇器來決定要以哪個 clock signal 為目前 DFF 的 clock source。

C. Problem Encountered

在寫 lab4_2 的 INITIAL state 的調 bcd digit 機制時，我一開始忘記 tmp_bcd_d1、tmp_bcd_d2、tmp_bcd_d3 也要把所有可能情形的值列出來(或者先設一個初始值)，而造成有 latch inference 的產生，如下圖所示(前 3 行被 comment 掉的部分)。會發現這個問題是因為在測試調 bcd digit 時，會發

生明明某個 switch 沒有拉起來，那個 switch 對應的 digit 卻改變了的現象。後來重新 trace code 一遍才發現 latch inference 的問題。

```
// tmp_bcd_d1 = bcd % 10;
// tmp_bcd_d2 = (bcd / 10) % 10;
// tmp_bcd_d3 = bcd / 100;
if (increase_pulse == 1'b1) begin
    if (Digit_1 == 1'b1) tmp_bcd_d1 = (bcd % 10 == 9) ? 10'd0 : (bcd % 10 + 10'd1);
    if (Digit_2 == 1'b1) tmp_bcd_d2 = ((bcd / 10) % 10 == 9) ? 10'd0 : ((bcd / 10) % 10 + 10'd1);
    if (Digit_3 == 1'b1) tmp_bcd_d3 = (bcd / 100 == 9) ? 10'd0 : (bcd / 100 + 10'd1);
    next_bcd = 100 * tmp_bcd_d3 + 10 * tmp_bcd_d2 + tmp_bcd_d1;
end
if (decrease_pulse == 1'b1) begin
    if (Digit_1 == 1'b1) tmp_bcd_d1 = (bcd % 10 == 0) ? 10'd9 : (bcd % 10 - 10'd1);
    if (Digit_2 == 1'b1) tmp_bcd_d2 = ((bcd / 10) % 10 == 0) ? 10'd9 : ((bcd / 10) % 10 - 10'd1);
    if (Digit_3 == 1'b1) tmp_bcd_d3 = (bcd / 100 == 0) ? 10'd9 : (bcd / 100 - 10'd1);
    next_bcd = 100 * tmp_bcd_d3 + 10 * tmp_bcd_d2 + tmp_bcd_d1;
end
```

D. Suggestions

希望可以提供關於每個 lab report 評分的項目，像是有哪些點是一定要寫的、哪些是不一定要寫的，因為我發現常常會寫了很多，但卻漏掉我以為不重要的點，而因此被扣一點分數。也因為這樣在寫 report 時常常會猶豫某段 code 要不要解釋，因為解釋的話會怕貼上來的 code 太多被扣分，但不解釋又怕那個是評分項目之一。又或者是那段 code 在 demo 已經被 TA 問過了，也會猶豫要不要寫進來。