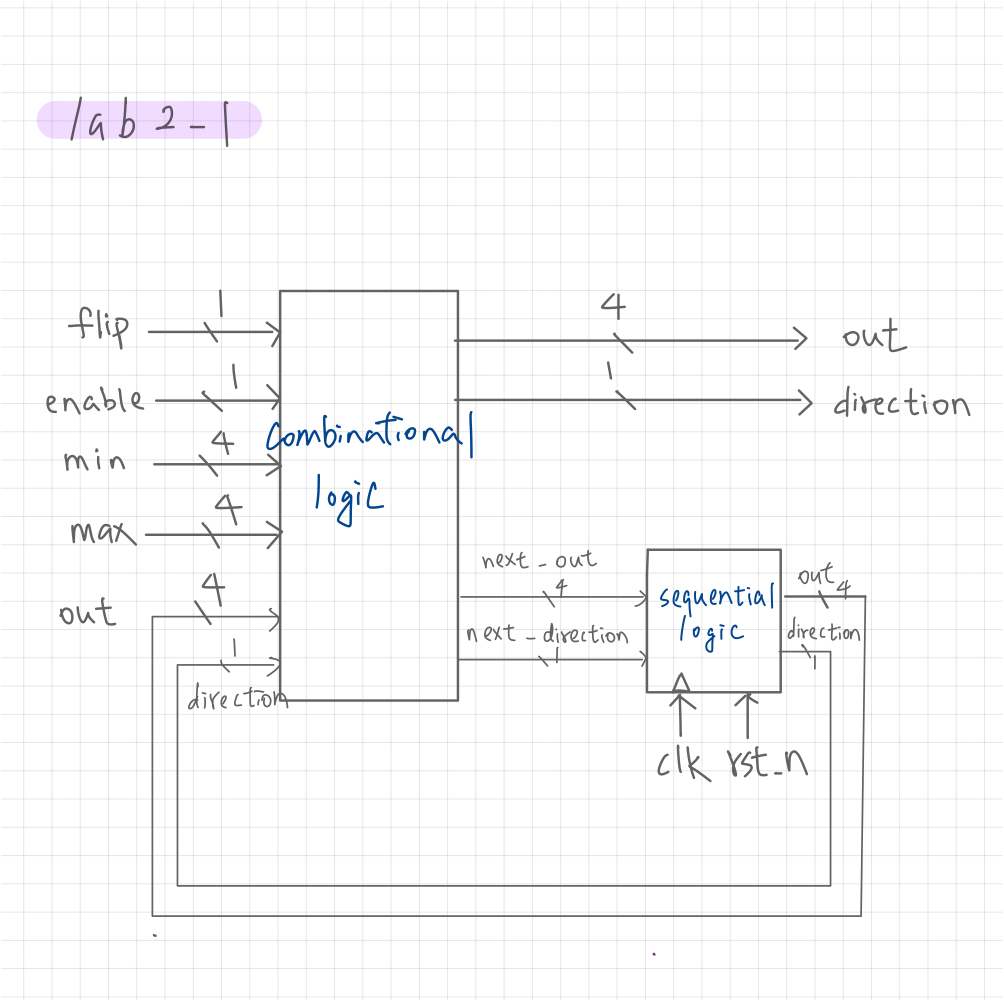
Lab 2

學號: 109062173 姓名: 葉昱揚

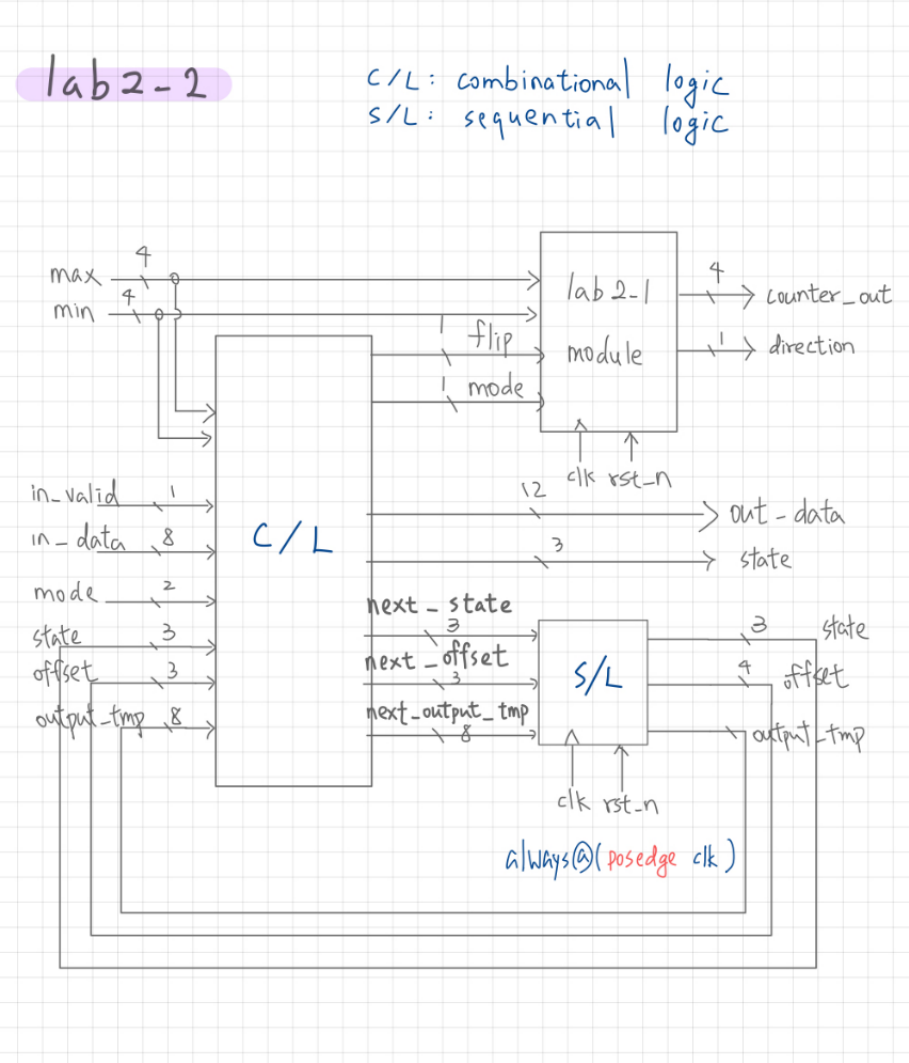
A. Lab Implementation

1. Block diagram of the design with an explanation.



圖。Lab2_1 block diagram

本次 LAB 需要 sequential circuiit 的觀念，out 和 direction 都需要 previous state 才能接著計算下去，比方說 out 可能為 0、1、2、……、15，當 direction 變化時 out 需要遞增或遞減，因此倆者都需要 register 去記住 previous state 的值，再用當下的 input signal 和 previous state 的值決定現在的 out 和 direction 是多少。方法就是多拉兩條電線 : next_out 和 next_direction，那麼這兩條電線是 combinational circuit 的 output，在 clock edge trigger 的時候再接給對應的電線。若不拉這兩條線，單純只有 out 和 direction 兩條電線，只能夠讓 out 和 direction 自己頭尾接著，這在電路中是危險且不建議的行為。頭尾接著一樣的電線，input 和 output 都是同一條電線很難實現。



圖。Lab2_2 block diagram

Lab2_2 的概念圖和 Lab2_1 非常相似，state、offset、output_tmp 都是需要 input 和 previous state 才能決定下一次的值的電線，同樣的幫它們都各自拉了電線 ：next_state、next_offset、next_output_tmp，三者的值藉由 combinational circuit 計算出來後在 clock edge trigger 時接給對應的電線。除此之外也需要 Lab2_1 構建 Lab2_2，可以看上圖的 max、min 和 C/L 的 output flip、mode 都需要接給 lab2_1，來計算出 counter_out 和 direction。

2. Partial code screenshot with the explanation: you don’t need to paste the entire code into the report. Just explain the kernel part.

```
/* determine the next_output_tmp value base on the current state */
/* You can store the in_data in the next_output_tmp (by using the value of offset_cnt reg)
   and then process these data in the PROCESS_DATA state */
always@(*) begin
    case(state)
        INIT: begin
            if(in_valid) next_output_tmp[offset] = in_data;
            else next_output_tmp[offset] = 8'd0;
        end
        GET_DATA: begin
            if(in_valid) next_output_tmp[offset+1] = in_data;
        end
        ENCRYPT_DATA: begin
            if(!in_valid && mode==2'b10) begin
                encrypt_data[offset] = (next_output_tmp[offset] + counter_out) % 256;

                /* hamming code encoding rules */
                hamming_code[offset][1]  = encrypt_data[offset][0] ^ encrypt_data[offset][1] ^ encrypt_data[offset][3] ^ encrypt_data[offset][4] ^ encrypt_data[offset][6];
                hamming_code[offset][2]  = encrypt_data[offset][0] ^ encrypt_data[offset][2] ^ encrypt_data[offset][3] ^ encrypt_data[offset][5] ^ encrypt_data[offset][6];
                hamming_code[offset][4]  = encrypt_data[offset][1] ^ encrypt_data[offset][2] ^ encrypt_data[offset][3] ^ encrypt_data[offset][7];
                hamming_code[offset][8]  = encrypt_data[offset][4] ^ encrypt_data[offset][5] ^ encrypt_data[offset][6] ^ encrypt_data[offset][7];

                hamming_code[offset][3]  = encrypt_data[offset][0];
                hamming_code[offset][5]  = encrypt_data[offset][1];
                hamming_code[offset][6]  = encrypt_data[offset][2];
                hamming_code[offset][7]  = encrypt_data[offset][3];
                hamming_code[offset][9]  = encrypt_data[offset][4];
                hamming_code[offset][10] = encrypt_data[offset][5];
                hamming_code[offset][11] = encrypt_data[offset][6];
                hamming_code[offset][12] = encrypt_data[offset][7];

            end
        end
        // OUTPUT_DATA: begin
        //     next_output_tmp[offset] = hamming_code[offset];
        // end
        default:begin
            next_output_tmp[offset] = output_tmp[offset];
        end
    endcase
end
```

這是 lab2_2 最核心的 code segment。原因是在 INIT 的最後一個 cycle，也就是 in_valid = 1 的當下就要存 input_data 進 vector；後面的 GET_DATA STATE 就要把存的位置也順移一格，在這個階段第 i 個 cycle 進來的 input 必須放在第 i+1 格，如此才能順利存入 data，這也是本次 lab 最難的地方。

3. How you test your design: waveform screenshot with explanation.

因為這次是固定 clock cycle 後重複循環的 lab，寫 tb 的時候發現 code 重複性很高，所以第一次試著把 code 用 define 定義好。這次 FSM 有四個 state，INIT、GET_DATA、ENCRYPT_DATA、OUTPUT_DATA，所以 define 了以下幾個 code segment :

```
/* INIT STATE*/
`define GIVE_INIT_INPUT(in_data_signal,mode_signal) \
    @(negedge clk) begin \
        in_valid = 1'b1; \
        in_data = ``in_data_signal``; \
        mode = ``mode_signal``; \
    end\

/* GET DATA STATE*/
`define DATA_INCREMENT(increment_value) \
    repeat(7) begin \
        #10; \
        in_data = in_data + ``increment_value``; \
        if(in_data == 8'd0) in_data = 1; \
    end  #10

/* ENCRYPT_DATA STATE*/
`define ENCRYPT_DATA \
    @(negedge clk) begin \
        in_valid = 1'b0; \
        mode = 2'b10; \
    end
```

OUTPUT_STATE 不是 tb 要做的事情，所以只需要寫前面三個 STATE 的 tb。而這些 macro 可以省略很多行數，除了提高可讀性以外，對後續程式的維護有很大的幫助。

設計過程也謹記要把所有的 input 可能性都測試一遍，全部的可能性、code segment 和 waveform 如下圖 ：

1. min < max ， INIT 時 mode=2’b00
2. min < max ， INIT 時 mode=2’b01

```
/**********            min < max                ********/

@(negedge clk) {min,max} = {4'd0,4'd15};
repeat(5) begin
    `GIVE_INIT_INPUT( ($urandom%255)+1 , 2'b00)
    `DATA_INCREMENT(($urandom % 100) +1);
    `ENCRYPT_DATA
    /* OUTPUT DATA . Just wait for output*/
    #STATE_DELAY;
end

@(negedge clk) {min,max} = {4'd0,4'd15};
repeat(5) begin
    `GIVE_INIT_INPUT( ($urandom%255)+1 , 2'b01)
    `DATA_INCREMENT(($urandom % 100) +1);
    `ENCRYPT_DATA
    /* OUTPUT DATA . Just wait for output*/
    #STATE_DELAY;
end

/*********************************************************/
```

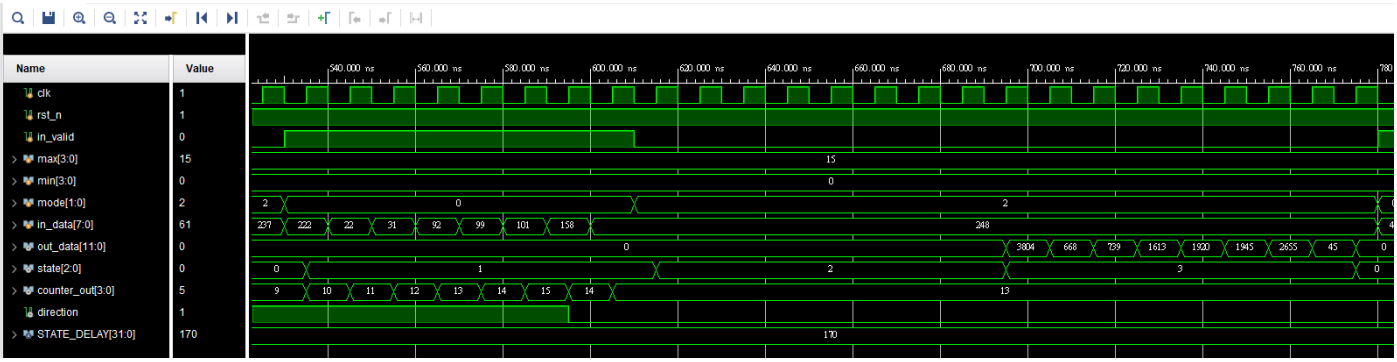


圖 min < max ， INIT 時 mode=2’b00

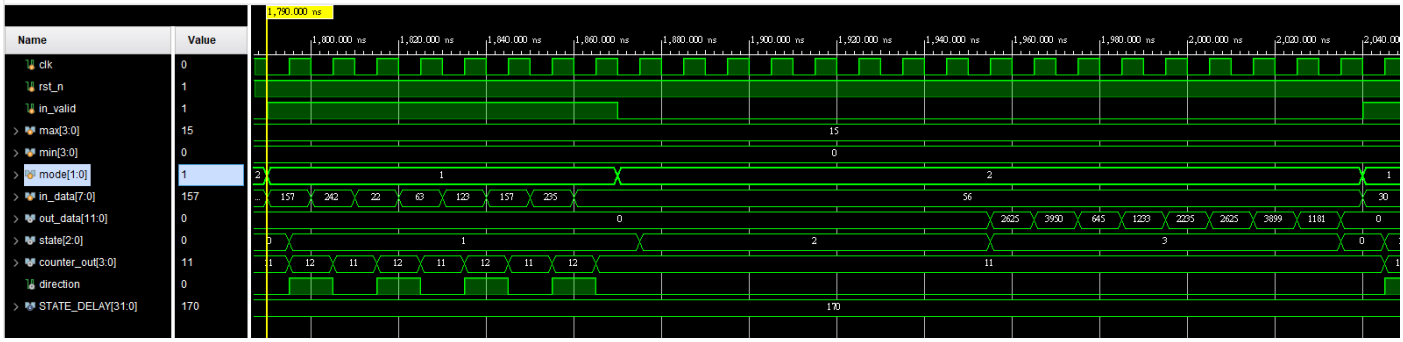


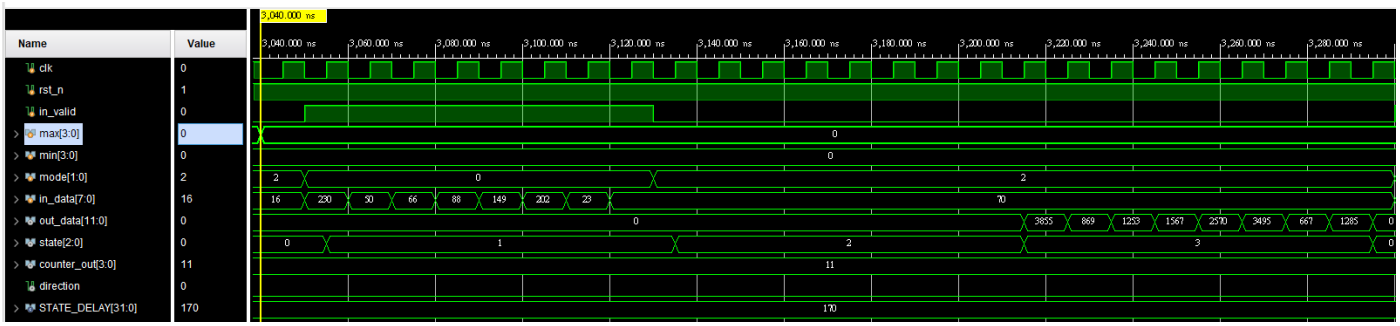
圖 min < max ， INIT 時 mode=2’b01

Mode=2’b00 如願看到 counter_out 沒有擅自反轉 ; Mode=2’b01 如願看到 counter_out 反轉。且因為 min < max，counter_out 持續遞增 or 遞減。

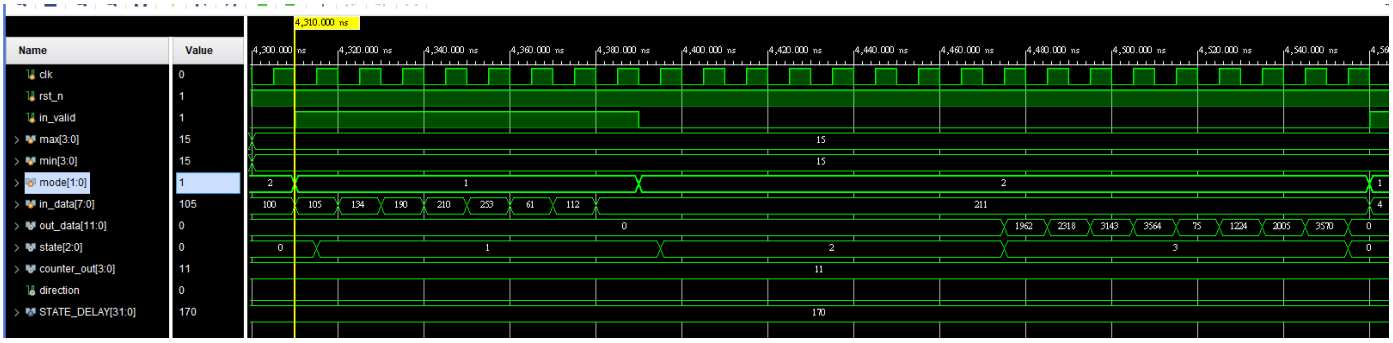
3. min = max ， INIT 時 mode=2’b00
4. min = max ， INIT 時 mode=2’b01

```
/**********              min = max                 **********/
@(negedge clk) {min,max} = {4'd0,4'd0};
repeat(5) begin
    `GIVE_INIT_INPUT( ($urandom%255)+1 , 2'b00)
    `DATA_INCREMENT(($urandom % 100) +1);
    `ENCRYPT_DATA
    /* OUTPUT DATA . Just wait for output*/
    #STATE_DELAY;
end

@(negedge clk) {min,max} = {4'd15,4'd15};
repeat(5) begin
    `GIVE_INIT_INPUT( ($urandom%255)+1 , 2'b01)
    `DATA_INCREMENT(($urandom % 100) +1);
    `ENCRYPT_DATA
    /* OUTPUT DATA . Just wait for output*/
    #STATE_DELAY;
end
/*************************************************************/
```



圖。min = max ， INIT 時 mode=2’b00



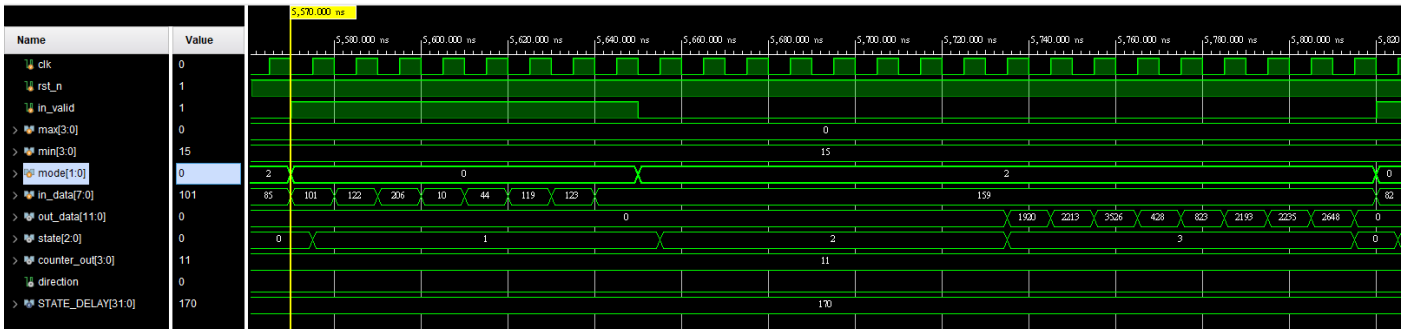
圖。min = max ， INIT 時 mode=2’b01

因為 min = max，所以 counter_out 如願看到定格在固定數字，也不會因為 mode 不同而擅自反轉。

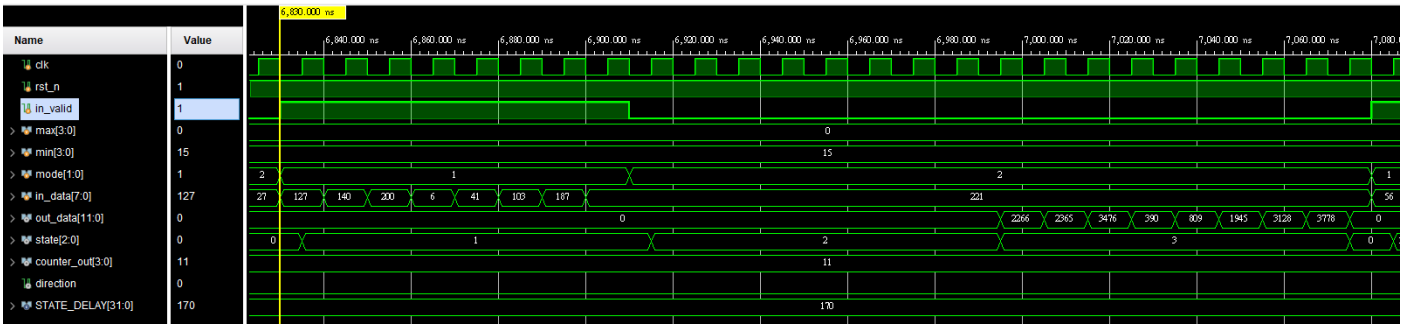
5. min > max ， INIT 時 mode=2’b00
6. min > max ， INIT 時 mode=2’b01

```
/***********                min > max                ***********/
@(negedge clk) {min,max} = {4'd15,4'd0};
repeat(5) begin
    `GIVE_INIT_INPUT( ($urandom%255)+1 , 2'b00)
    `DATA_INCREMENT(($urandom % 100) +1);
    `ENCRYPT_DATA
    /* OUTPUT DATA . Just wait for output*/
    #STATE_DELAY;
end

@(negedge clk) {min,max} = {4'd15,4'd0};
repeat(5) begin
    `GIVE_INIT_INPUT( ($urandom%255)+1 , 2'b01)
    `DATA_INCREMENT(($urandom % 100) +1);
    `ENCRYPT_DATA
    /* OUTPUT DATA . Just wait for output*/
    #STATE_DELAY;
end
/*************************************************************/
```



圖。min > max ， INIT 時 mode=2’b00



圖。min > max ， INIT 時 mode=2’b01

因為 min > max，所以 counter_out 如願看到定格在固定數字，也不會因為 mode 不同而擅自反轉。

4. Event-based FSM with the explanation

基本的原則就是多拉出幾條線，並命名為 next_xxxx，這些 next_xxxx 會是 sequential circuit 的 input，對應的 output 是 xxxx，xxxx 再接回 combinational circuit 的 input。比方說 next_state 會是 sequential circuit 的 input，對應的 output 會是 state，state 再接回 combinational circuit 的 input。 Combinational circuit 內部的 operation 依照 SPEC 的內容實作，GET_DATA 存 input、ENCRYPT_DATA 做加密。Offset 在各個階段則是往上遞增，到達最大值時變回 0。

B. Questions and Discussions

1. In this lab, our reset signal is a synchronous reset. What if it is an asynchronous reset? And how to modify your design to implement an asynchronous reset?

首先，先分清楚 synchronous reset 和 asynchronous reset 的區別 ：

synchronous reset : 暫存器(register)會維持原本的值。如果 clock edge trigger 的時候 reset signal active，register 則會被 reset ; 如果 reset signal active 的當下沒有 clock edge trigger，則不做 reset。亦即 reset 會跟著 clock edge 行動。
實作方式如下圖。若需要 reset，先決條件是 clock edge 是否 trigger :

```
/* synchronous reset */

always@(posedge clk) begin
    if(!rst_n)
        // 需要等 posedge clk 才能 reset register
    else
        // 不reset register，根據lab的operation賦予register新的值
end
```

圖。synchronous reset

asynchronous reset: 暫存器(register)會維持原本的值。每當 reset signal active，register 立刻被 reset，reset 的時間點與 clock edge 無關。
實作方式如下圖。不需要判斷 clock edge 是否 trigger :

```
/* asynchronous reset */

always@(posedge clk,negedge rst_n) begin
    if(!rst_n)
        // 不需要等到posedge clk , 可以立刻 reset register
    else
        // 不reset register，根據lab的operation賦予register新的值
end
```

圖。asynchronous reset

2. Why do we need both combinational circuits and sequential circuits in our design? What are the differences between a combinational circuit and a sequential circuit? Please explain how each of them works in detail.

本次 lab 需要設計出 finite state machine，並以此為基礎寫 code。此次 FSM 在各個 state 都有要做的事情 ： 需要等待 8 個 clock cycle 來接收 input、需要 8 個 cycle 來計算 vector 的 offset 等等，這些事情都需要先判斷上一個 clock cycle 時的狀態，才能決定現在該怎麼做。而我們只能利用 sequential circuit 達到儲存狀態的目的，因為 combinational circuit 不能儲存狀態，不能用

C/L 來做出這次的 FSM。C/L 本次的任務是基本的數值計算，只需要根據 input 算出對應的 output 就好，所以 C/L 的電路元件不需要存狀態，也不需要用 clock 來控制 output。

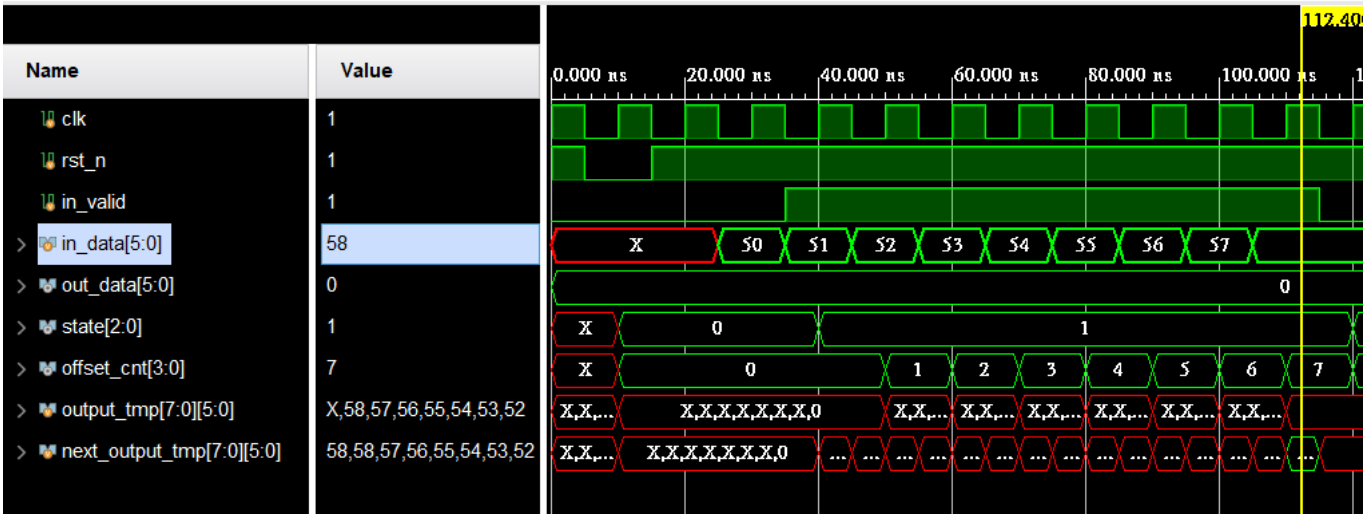
Combinational circuit 的特色是不受 clock 控制、output 完全由 input 決定。沒有儲存元件的功能，不能儲存上一次計算的值。一但輸入變化，立刻輸出計算後的 output value。

Sequential circuit 的特色是 output 由 input 和 previous state 決定，並且會用 clock 來控制 output。通常會設計多個 register 並用適當長度的 clock cycle 來 trigger 這份 sequential circuit。每當 clock edge trigger 的時候，register 可以根據 input 和 previous state 來判斷現在該輸出甚麼 output value。

這次 lab 也有個重要的任務，就是讓電路同步化。在 lab 1 有學習到 C/L 計算 output 會有 gate delay，而實際情況中每個電路元件需要的 delay time 不同，所以 S/L 用 clock 控制可以幫助我們確定在 clock edge trigger 前，output 不會被偷跑、計算好的訊號改變，造成輸出與期望中的不一樣。

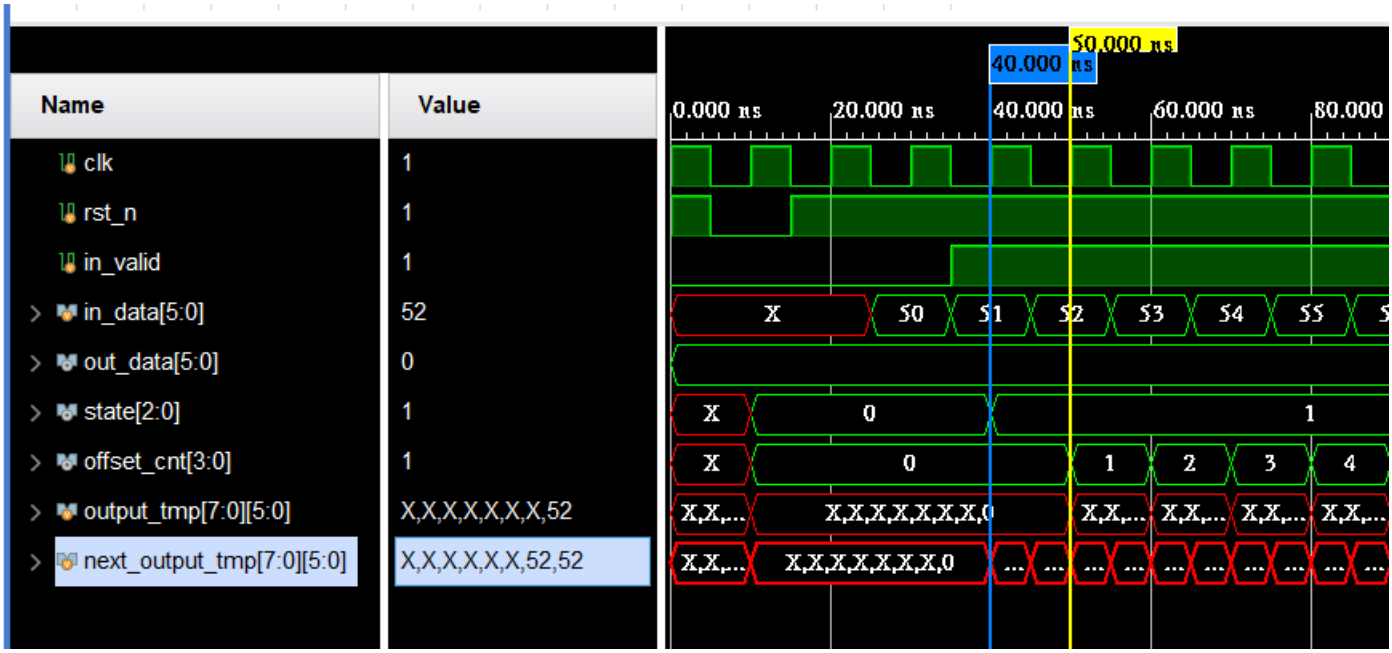
C. Problem Encountered

本次 lab 遇到最大的困難是無法順利把 input data 存進 vector 裡面。跑 tb 的時候，GET_DATA 階段 in_data 會是[51,52,53,54,55,56,57,58]，結束時 output_tmp 裡面會是 [51,52,53,54,55,56,57,58] 才對，接著在 PROCESS_DATA 的階段，將 output_tmp 的每個值+1，但是我的 code 在 GET_DATA 階段結束後 output_tmp 裡面裝的值會是[52,53,54,55,56,57,58,0]，waveform 如下圖 ：



圖。Input data 無法順利裝入 output tmp

兩相比較，我發現 input data 沒有裝在該裝的位子，該裝在 index = i+1 的值都被裝到 index = I 的地方了，當時認為晚了一個 clock cycle 讓 offset + 1，才會造成此結果。後來更細節的發現在 time=40ns~50ns 的時候 next_output_tmp 竟然被更動了兩次，如下圖 ：



圖。next_output_tmp 在 40ns~50ns 被更動兩次

理論上，next_output_tmp 應該只能被動一次，後來發現 next_offset 是用 C/L 來控制的，而 offset 是用 S/L 控制，所以這個 clock cycle 內因為 input data 從 51 變成 52，讓 next_offset 遞增，offset 因為還沒被 clock edge trigger，所以才會造成如此詭異地 input data 不能順利裝進 vector 的局面。

最後的解法是在 INIT STATE 就開始裝 input data。原先的設計是在 GET_DATA 才開始裝，結果裝不進去。又觀察到了整體 output 的 clock 有晚一個 cycle 的現象，才會設計出把全部 operation 都往前移一個 cycle 的電路，亦即在 INIT STATE 的最後一個 cycle 裝 input data，GET_DATA STATE 的第 i 個 cycle 接收到的值存入第 i+1 個 vector 內，如此一來成功存入 input data。

D. Suggestions

非常非常感謝唐助教本次 lab 的大力幫忙。稍微注意過，討論區的發問通常半小時內都可以得到回覆，回覆頻率太高、太勤勞、太令人敬佩了，希望以後的 TA 們不要因為忙於回答問題而讓自己被 DEADLINE 追殺。

閱讀這次 lab SPEC 的時候就發現有滿多隱藏條件沒有說清楚，比如在某些時候 flip 不會出現、direction 碰到邊界會自動反轉等等，許多條件是忘記在 SPEC 寫清楚，反而靠同學們在討論區一個一個問出來的，是以後可以多多注意的地方 ！

另外關於 TA time，老師和 TA 們覺得 TA time 改成開 google meet 或 microsoft team 線上解答如何 ？ 原因是這個季節又熱蚊子又多，加上晚上 8.30 TA 們還要特地勞碌地趕到資電館，有點麻煩，線上 TA time 也許效率會比較好 ？

最後附上梗圖一張