Lab 2

學號: 110011138 姓名: 楊立慈

A. Lab Implementation

(a) Lab2_1

1. 圖 1 為 lab2_1 的 block diagram，其中 F/F 為由 clk 訊號 positive edge-triggered、由 rst_n 代表 synchronous negative reset、en 代表 enable operation。其中 F/F 的兩個 ouptut out 跟 direction 即為整個 lab2_1 block 的 output，同時 out、direction 以及 max、min、flip 等等 lab2_1 block 的 input 會一起送進兩個 combinational circuit 裡計算 out 跟 direction 的 next state 並送回 F/F。

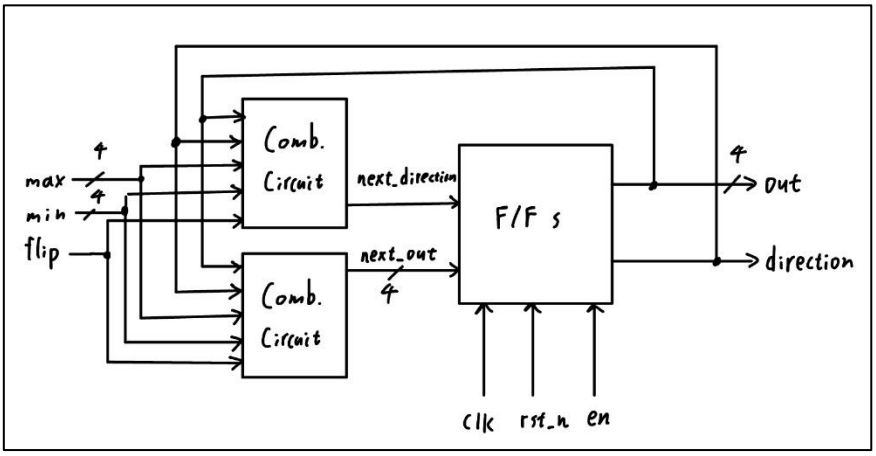

圖 1: lab2_1 block diagram

2. 圖 2 為 lab2_1 負責 state update 的 code。這裡主要負責圖 1 的 block diagram 中 F/F 的工作，並運用上課所教的 synchronous negative reset 以及 enable 的寫法，還有就是 sequential block 要使用 non-blocking assignment <=。

```
always @(posedge clk) begin
    if (rst_n == 1'b0) begin
        out <= min;
        direction <= 1'b1;
    end
    else if (enable == 1'b1) begin
        out <= next_out;
        direction <= next_direction;
    end
end
```

圖 2: lab2_1 state update code

圖 3 為 block diagram 中負責計算 output 的 next state 的 combinational circuit (由於計算 direction 的 code 類似，因此這裡就不放了)。第一行先把 next_out default 成 out，課堂有說過這樣寫的好處是可以不用擔心之後會不會有漏掉 case 造成 latch inference。邏輯上，當要 flip 時，下一個 out 會是原本的 out 改變方向後走一步。而當不用 flip 且 out、max、min 的

值都在正確範圍時，next_out 則依原本的 direction 去增加或減少，並注意若碰到邊界要回彈。

```
always @(*) begin
    next_out = out;

    if (flip == 1'b1) begin
        next_out = (direction ? out - 1'b1 : out + 1'b1);
    end
    else if (max > min && out <= max && out >= min) begin
        // touch the border
        if ((out == max && direction) || (out == min && ~direction)) begin
            next_out = (direction ? out - 1'b1 : out + 1'b1);
        end
        else begin
            next_out = (direction ? out + 1'b1 : out - 1'b1);
        end
    end
end
```

圖 3: lab2_1 output next state code

3. 圖 4 及圖 5 為 lab2_1 的 waveform，我主要測試了正常情況下(no flip、enabled、reset 過)不同 max、min 的值是否 out 都是預期的(像是碰到邊界會回彈等)，包括: min <= out <= max、min == out == max、out 超過 min/max 範圍、max < min 等等。以及測試 flip 在 posedge、negedge trigger 會不會有 bug。以及 enable、reset 等是否有正常運作。

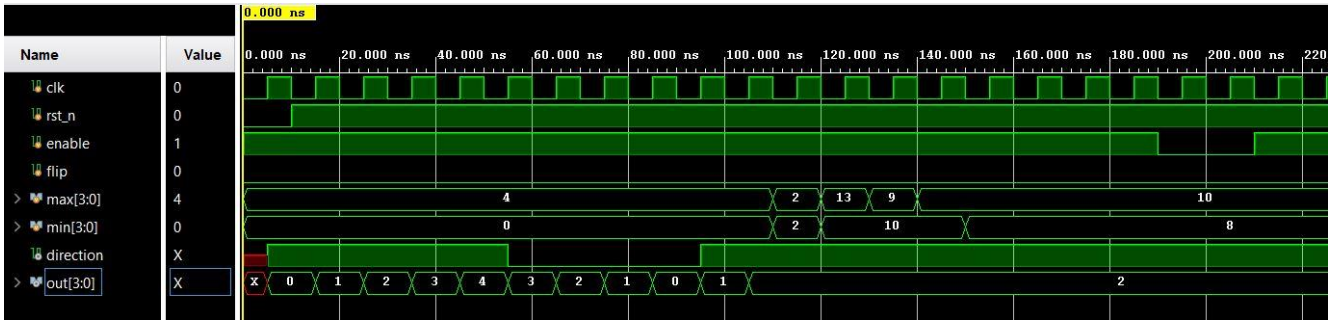

圖 4: lab2_1 waveform

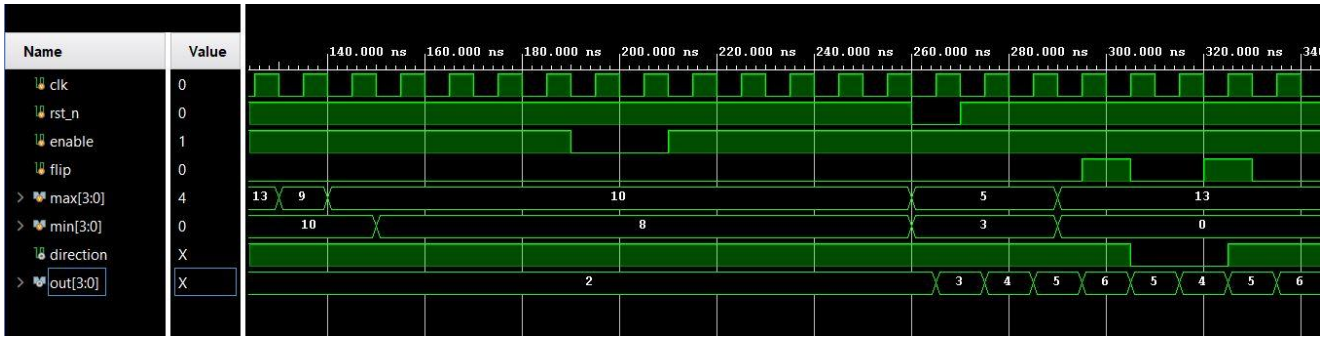

圖 5: lab2_1 waveform

(b) Lab2_2

1. 圖 6 為 lab2_2 內部使用 lab2_1 的 block diagram，圖 7 為 lab2_2 主要的 counter、output、state FSM 的 block diagram (為求簡潔，有些訊號像是 counter_out 等，沒有真畫出接到其他 FSM 的線。有些比較鄰近的 signal 則有畫出接出去的線)。

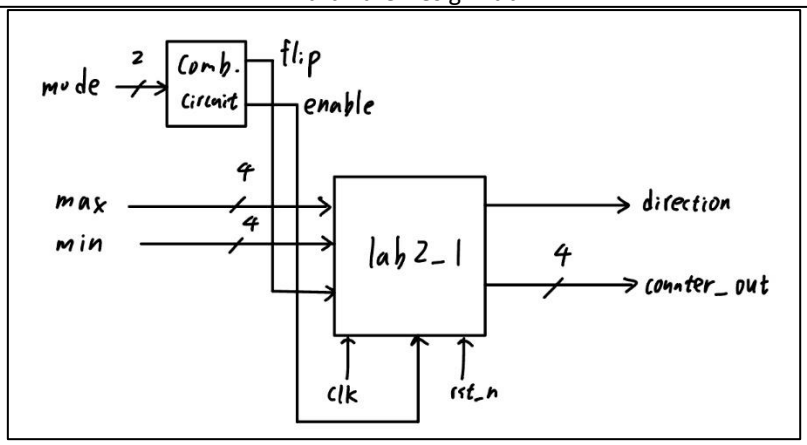

圖 6: lab2_2 block diagram of inner lab2_1

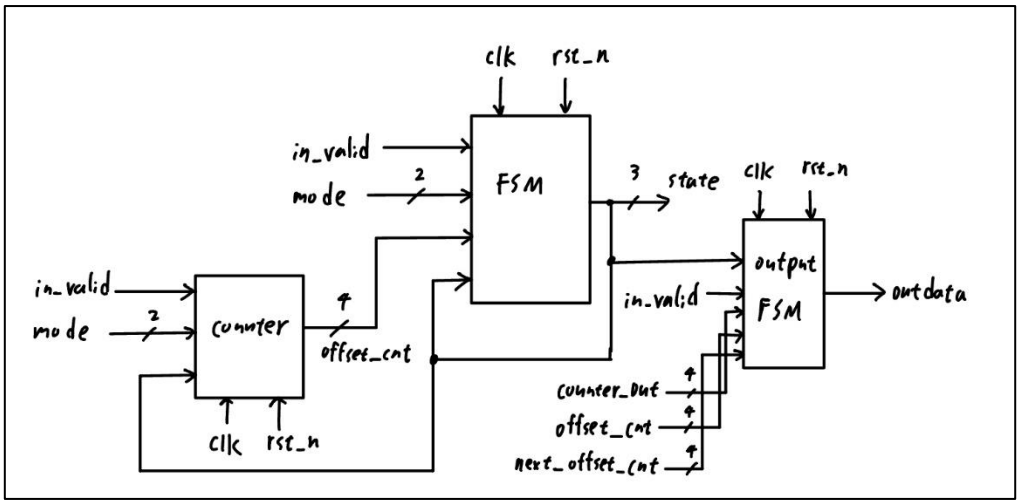

圖 7: lab2_2 main block diagram

2. 由於 state update 的部分與 lab2_1 很類似，因此這裡就不貼截圖了。圖 8 為部分的 state transition 的 code，主要就是實現圖 12 的 state diagram，根據當前的 state 以及 in_valid、mode 等 input 來更新 next_state。

```
always@(*) begin
    case(state)
    INIT: begin
        if(in_valid) next_state = GET_DATA;
        else next_state = INIT;
    end
    GET_DATA: begin
        if (!in_valid && mode == 2'b10) begin
            next_state = ENCRYPT_DATA;
        end
        else begin
            next_state = GET_DATA;
        end
    end
```

圖 8: lab2_2 state transition partial code

圖 9 為主要實現 encryption 的 code。我先用了一個暫時的 variable e 來儲存 encrypted message，接著再利用 concatenation 來拼湊出有了 error correction code 的結果，並存回 next_output_tmp。

```
ENCRYPT_DATA: begin
    e = (output_tmp[offset_cnt] + counter_out) % 256;
    next_output_tmp[offset_cnt] = {e[7:4],
                                   (e[4] ^ e[5] ^ e[6] ^ e[7]),
                                   e[3:1],
                                   (e[1] ^ e[2] ^ e[3] ^ e[7]),
                                   e[0],
                                   (e[0] ^ e[2] ^ e[3] ^ e[5] ^ e[6]),
                                   (e[0] ^ e[1] ^ e[3] ^ e[4] ^ e[6])};
end
```

圖 9: lab2_2 encryption code

3. 圖 10、圖 11 為 lab2_2 的 waveform。由於這題 input 的 constraints 較多，且 lab2_1 已經測試過了，因此這裡我主要測試我 FSM 內部的 counter 沒有數錯、encryption 有正確計算、GET_DATA 有確實讀進 8 筆 data、ENCRYPT_DATA 和 OUTPUT_DATA 有確實等 8 個 cycle，以及 reset、mode 等有正確維持、改變 state。

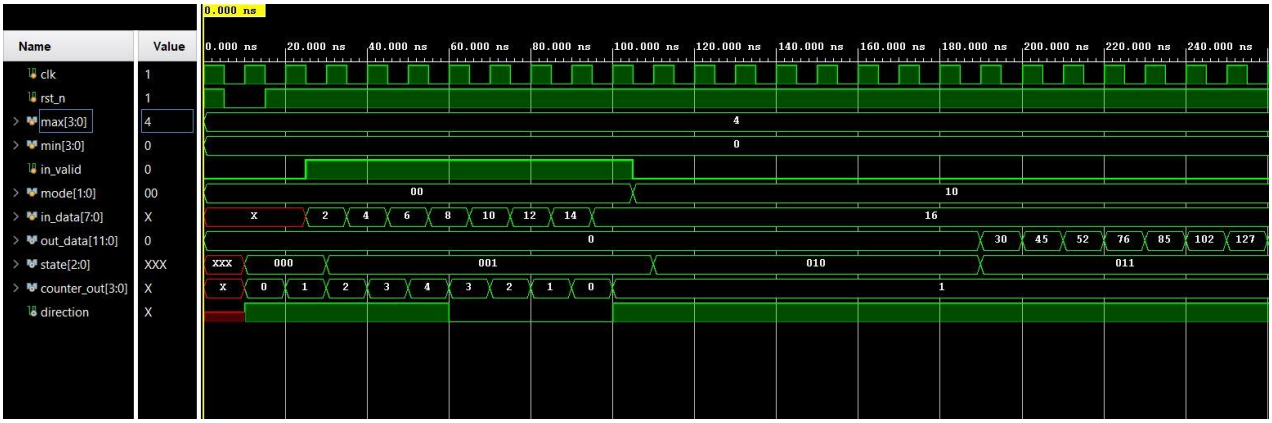

圖 10: lab2_2 waveform

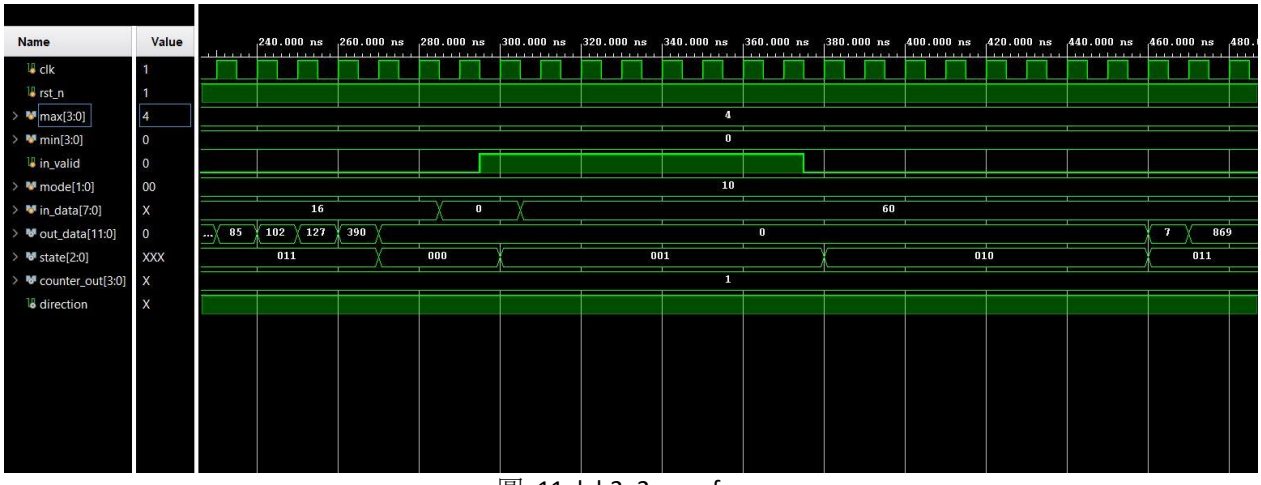

圖 11: lab2_2 waveform

4. 圖 12 為 lab2_2 的 state diagram。首先，當進行 reset 後，我們會進入 INIT 的初始 state，並且除非 in_valid 變為 1 則進入 GET_DATA，否則就一直待在 INIT state。

   在 GET_DATA 時除非 in_valid 是 0 且 mode 等於 2'b10 才會進入 ENCRYPT_DATA state。

由於 input 保證 in_valid 在輸入 8 個 input 時都會是 1，因此可以看成進入 GET_DATA 後的 8 個 cycle 會讀進 8 筆 data，並且當 mode 會使 lab2_1 的 counter 維持住時，我們才會進到下一個 state。

在 ENCRYPT_DATA state 則利用 counter 計算 8 個 cycle 後，再進入 OUTPUT_DATA state。在這 8 個 cycle 內，我們要分別對剛剛存入的 8 筆資料進行 encryption。

OUTPUT_DATA state 則單純同樣利用 counter 數 8 個 cycle，並在每個 cycle output 剛才算過的 encrypted message。8 個 cycle 後，都 output 完了就回到一開始的 INIT state，等待下一輪的計算。

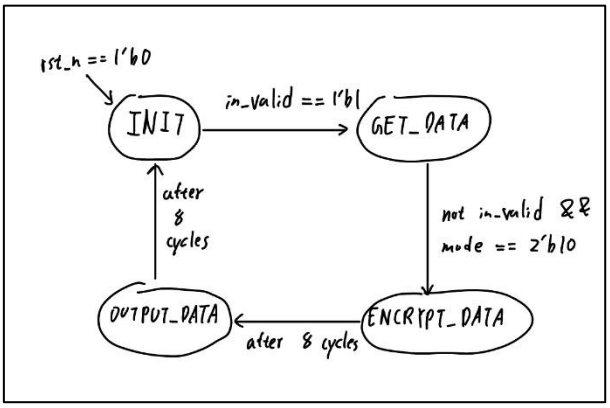

圖 12: lab2_2 state diagram

B. Questions and Discussions

1. 如果是 asynchronous reset 的話，則不論何時只要 reset 訊號降為 0 (假設是 negative reset)，我們就要進行 reset，而不像 synchronous reset 只在 posedge 出現且 reset 是 0 時才 reset。要改成使用 asynchronous reset，我們只要在圖 2 的 state update 的 always block 的 sensitivity list 改成@(posedge clk or negedge rst_n)即可，這樣當 rst_n 由 1 降為 0 時就會進入此 block 並進行 reset。

2. 因為我們的 FSM 需要 sequential circuit 負責記憶目前 finite state machine 的 state，並在 posedge 來時更新 state。也需要 combinational circuit 負責根據 FSM 的 input 以及 sequential circuit 記憶的當前的 state 計算 FSM 當前的 output 以及用來給 sequential circuit 更新 state 的 next state function。

   兩者最主要的差異在於 sequential circuit 具有記憶功能(利用 Flip Flop 來記憶)，且只會在 clock edge 來時才改變 output，同時 sequential circuit 也可以接收 reset 訊號來將整個 FSM 回歸到 initial state，並有 enable 訊號來暫停或繼續 sequential circuit 的運作。

   相反的，combinational circuit 則沒有上述的功能。Combinational circuit 只是單純接收一些 input，運用許多邏輯閘的連接來實現出 output function，也因此 combinational circuit 的值隨時都會隨 input 改變而變化。

C. Problem Encountered

在測試 lab2_2 時，常常發現第一個或是最後一個 output 的 data 不是 unknown 就是不對的值，比如說圖 13 的右邊那組 output 的第一個應該是 30。後來透過 testbench 印出更多內部的 signal 後才找到問題是在我的 counter 會比 FSM 晚一個 cycle 從 0 開始數，導致存進 output_tmp

的是上一個 cycle 的值(也就是圖 12 左邊 in_data 的第一個值 2)。後來透過修改我的 counter 的 state transition 機制，才成功改掉這個 bug。

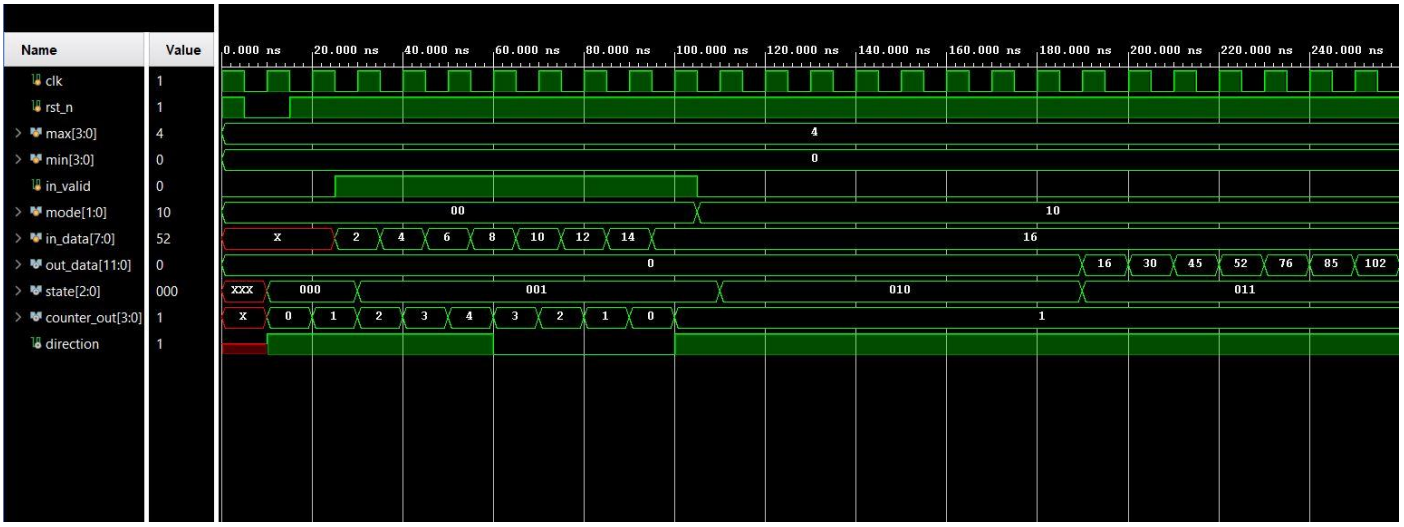

圖 13: lab2_2 bug

D. Suggestions

希望助教們能夠提供 lab 的參考解答，因為目前 lab 結束後我們只知道我們的寫法有得到想要的結果，但沒辦法知道我們自己的寫法是否是好的 coding style，比如說有些地方可能有更漂亮的寫法，或者說某些需求已經有固定的寫法等等。我相信若有參考解答能更加強我們對 verilog 的熟練度。謝謝助教!