Lab 2

學號: 109021115 姓名: 吳嘉濬

A. Lab Implementation

以下是實作 Lab2_1 的 kernel code：

```
reg next_direction;
reg [3:0] next_out;

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

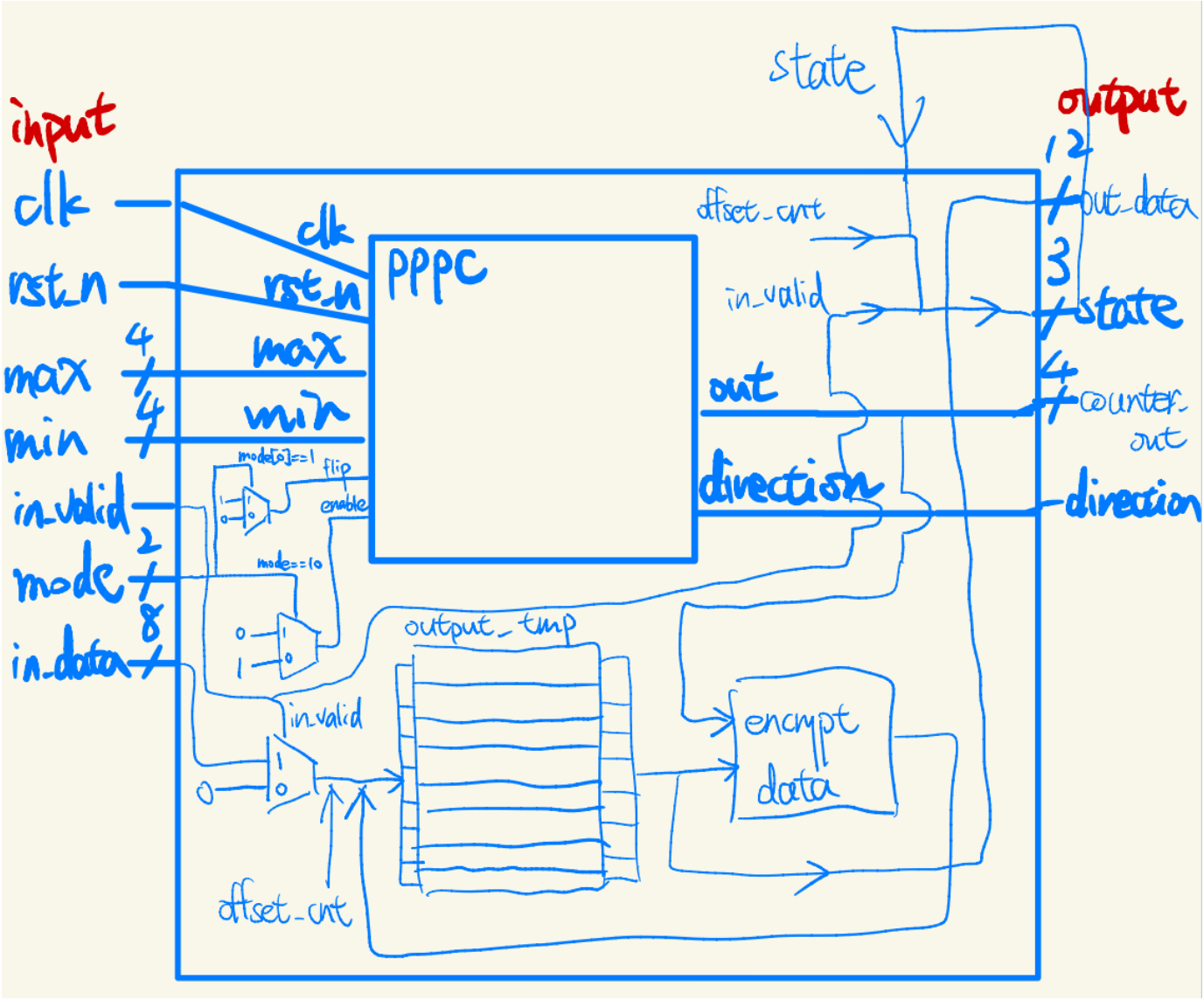
利用 sequential circuit 去儲存或更新 out 和 direction。

```
if(out>max || max<=min || out<min) begin
    next_out = out;
    next_direction = direction;
end else if(max==min && min==out) begin
    next_out = out;
    next_direction = direction;
end else if(out==max)begin
    next_out = out-1'b1;
    next_direction = 1'b0;
end else if(out==min) begin
    next_out = out+1'b1;
    next_direction = 1'b1;
end
```

考慮特殊情況以及 boundary case，並 assign 正確的值。

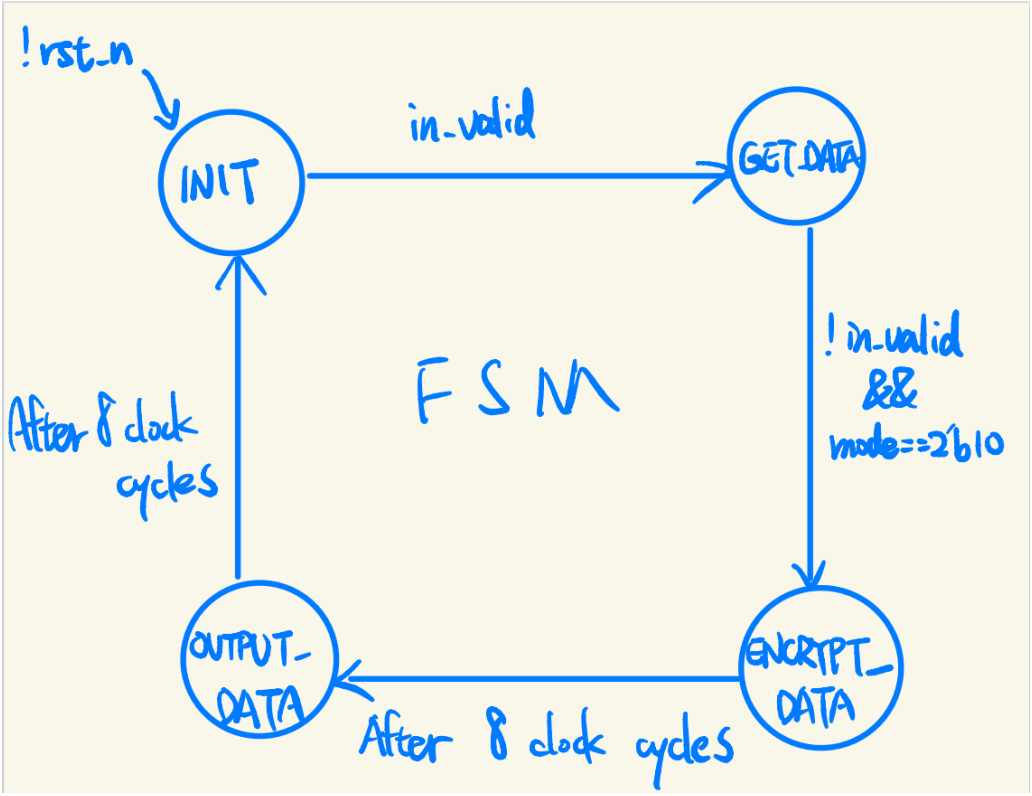
如此，Parameterized_Ping_Pong_Counter 可以正確地被實作出來。

以下為 lab2_2 的 block diagram：



因為是 sequential circuit，所以會有 current state 成為下一個 cycle 的 input 的情況(如圖右上方)。另外我們可以看到 pppc 的 flip 和 enable 是被 input mode 所決定的。而 offset_cnt 代表著 output_tmp 的 index，offset_cnt 的遞增幫助我們連續存取 output_tmp 內的 data 並作出相對應的操作，如 get(store) data、encrypt data 和 output data。其中 encrypt 的過程會使用到 pppc 的 counter_out，成為加密過程的參數。

以下為 lab2_2 的 FSM：



當 rst_n==1'b0 時，下個 cycle 會到 INIT state。在 INIT state 時，如果 in_valid==1’b1，下個 cycle 會進入 GET_DATA state，經過 8 個 cycle 並 store 完 8 組資料後，in_valid 一定會變回 1’b0(根據 spec 的說明)，而此時如果 mode==2’b10，則下個 cycle 進入 ENCRYPT_DATA state(如果 mode!=2’b10 則繼續等待)。經過 8 個 cycle 後，8 組 data 會依序 encrypt 完成，下個 cycle 會進入 OUTPUT_DATA state。再經過 8 個 cycle 後，8 組 data 會依序 output 完成，下個 cycle 回到 INIT state。

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?

Synchronous reset 和 asynchronous reset 在 waveform 中可觀察到最明顯的差別在於：當 synchronous reset 的值變成 1’b0，reset 不會馬上執行/生效，直到下一個 clk 的 posedge 才會有效(effective)；相對的，當 asynchronous reset 的值變成 1’b0，reset 則會立即生效(effective)。

左圖是 lab2_2.v 當中的某一段 code，右圖則是改成 asynchronous reset 的 code：

```
/* state transition */
always@(posedge clk)begin
    if(!rst_n) begin
        state <= INIT;
    end
    else begin
        state <= next_state;
    end
end
```

```
/* state transition */
always@(posedge clk, negedge rst_n)begin
    if(!rst_n) begin
        state <= INIT;
    end
    else begin
        state <= next_state;
    end
end
```

基本上就是在 sensitivity list 上加個 negedge rst_n 就可以達到 asynchronous reset 的效果，因為當 rst_n 被 de-assert 時產生 falling edge，即 negative edge，sensitivity list 內有 signal 產生變化，會立刻執行該 always block，不用等到下一個 posedge clk，因此產生了 asynchronous reset 的效

果。

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.

我們需要 combinational circuits(1)去判斷並決定下一個 state/下一個 offset_cnt；(2)去 process data(接收 8 組 data/做 encryption process+使用 Hamming codes 去建立 error correction code)；我們也需要 sequential circuits(1)去 update(in needed) the state/ the offset_cnt；(2)去儲存 8 組 data；(3)output 最終 8 組 data。

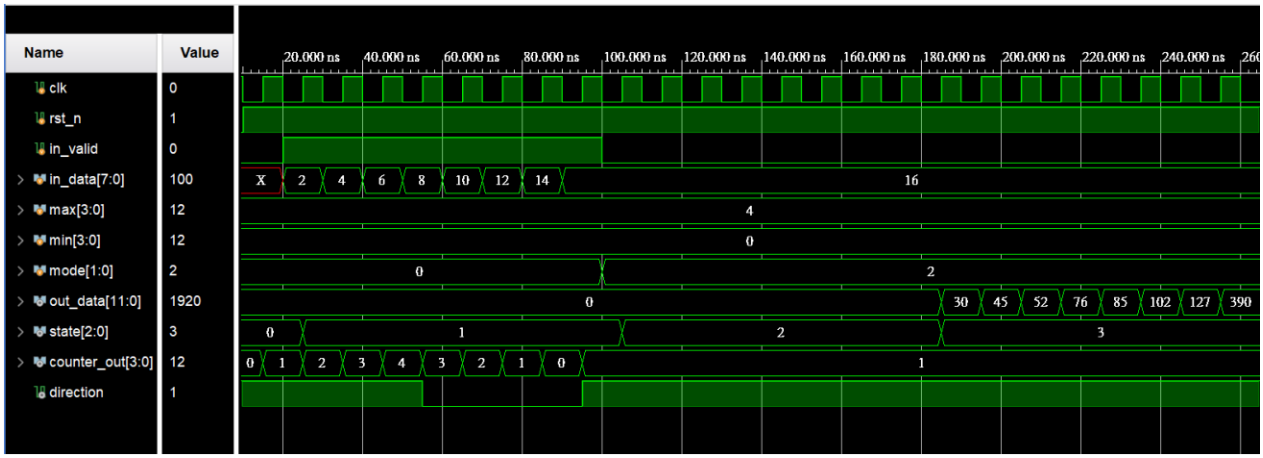
Combinational circuit 和 sequential circuit 的差異在於：combinational circuit 沒有 feedback path between input and output，所以 output depends on current input ONLY、time independent、可以對 data 做運算，但不具儲存功能；sequential circuit 有 feedback path between input and output，所以 output depends on current data and past data(或者說是 current data and current state)、time dependent、具有資料儲存功能。

C. Problem Encountered

最一開始在設計接收八組 data 時，是設計當我在 GET_DATA state 時，把 in_data 傳入 next_output_tmp[offset_cnt]，如下：

```
always@(*) begin
    case(state)
    INIT: begin
        next_output_tmp[offset_cnt] = output_tmp[offset_cnt];
    end
    GET_DATA: begin
        next_output_tmp[offset_cnt+1] = in_data;
    end
```

結果在跑 waveform 時發現，實際接收到的 in_data 不是在 GET_DATA state 那八個 cycle 對應到的 in_data，而是在 GET_DATA state 的第二個 cycle 才開始接收 in_data，一直到 ENCRYPT_DATA state 的第一個 cycle，簡單來說就是整個晚一個 cycle 收集 in_data。下方為舉例的圖片：



正確來說應該接收 2,4,6,8,10,12,14,16 這八組資料，但是如果用我原本錯誤的 code 會接收 4,6,8,10,12,14,16,16 這八組資料。

之後才發現一進入 GET_DATA state 的當下就要接收 in_data，所以必須要在進入 GET_DATA state 的前一個 cycle 就要先執行 next_output_tmp[offset_cnt] = in_data，才會讀到 GET_DATA state 的

第一個 in_data，而判斷進入 GET_DATA state 的前一個 cycle 的方式即當 in_valid==1 時，如下圖所示：

```
always@(*) begin
    case(state)
    INIT: begin
        if(in_valid) next_output_tmp[offset_cnt] = in_data;
        else next_output_tmp[offset_cnt] = output_tmp[offset_cnt];
    end
    GET_DATA: begin
        if(in_valid) begin
            next_output_tmp[offset_cnt] = in_data;
        end
        else next_output_tmp[offset_cnt] = output_tmp[offset_cnt];
```

利用這方法，才能夠得到正確的八組資料。

D. Suggestions

這次 lab2 的難度遠遠高於 lab1，是知道 lab 會越來越難沒錯，也知道第一次 lab 是在給我們試水溫，但是 lab2 的難度真的跳好多好多，花費的時間也變得好多好多，希望之後的 lab 難度不要再一次跳太多了，請用溫水煮青蛙的方式對待我們 QQ。