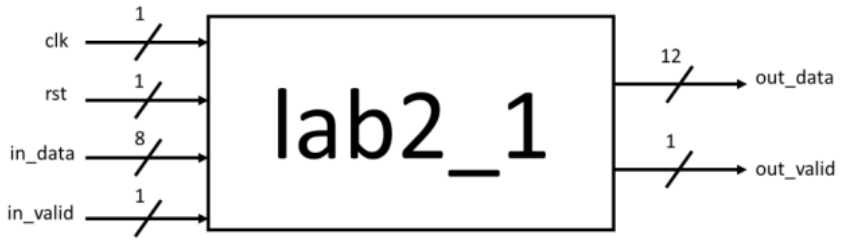
Lab 2

學號: 110062131 姓名: 馬毓昇

A. Lab Implementation

Lab2_1: 以 7-bit counter 加密 8-bit 的 in_data 再加上 4-bit 的 error correct code 做成 12-bit 的 out_data。



Lab2_1 segments:

1. 7-bit counter (挪用 offset_counter)

```
10  // 0-255 counter
11  reg [7:0] offset_counter, next_offset_counter;
```

2. 5 states

```
17  // state
18  parameter INIT = 3'd0;
19  parameter GET_DATA = 3'd1;
20  parameter ENCRYPT = 3'd2;
21  parameter CORRECT = 3'd3; // CALC
22  parameter OUTPUT_DATA = 3'd4;
23  reg [3:0] state, next_state;
```

3. 因為 in_data 是 8 bit 而 out_data 是 12 bit，所以用 concat 的方式在前面加上 4 個 0

```
212         GET_DATA : begin
213             for (i=0; i<=MAX_INPUT_LEN; i=i+1) begin
214                 next_output_data_save[i] = output_data_save[i];
215             end
216             if (in_valid) begin
217                 next_output_data_save[offset_counter] = {{4{1'b0}}, in_data};
218             end
219         end
```

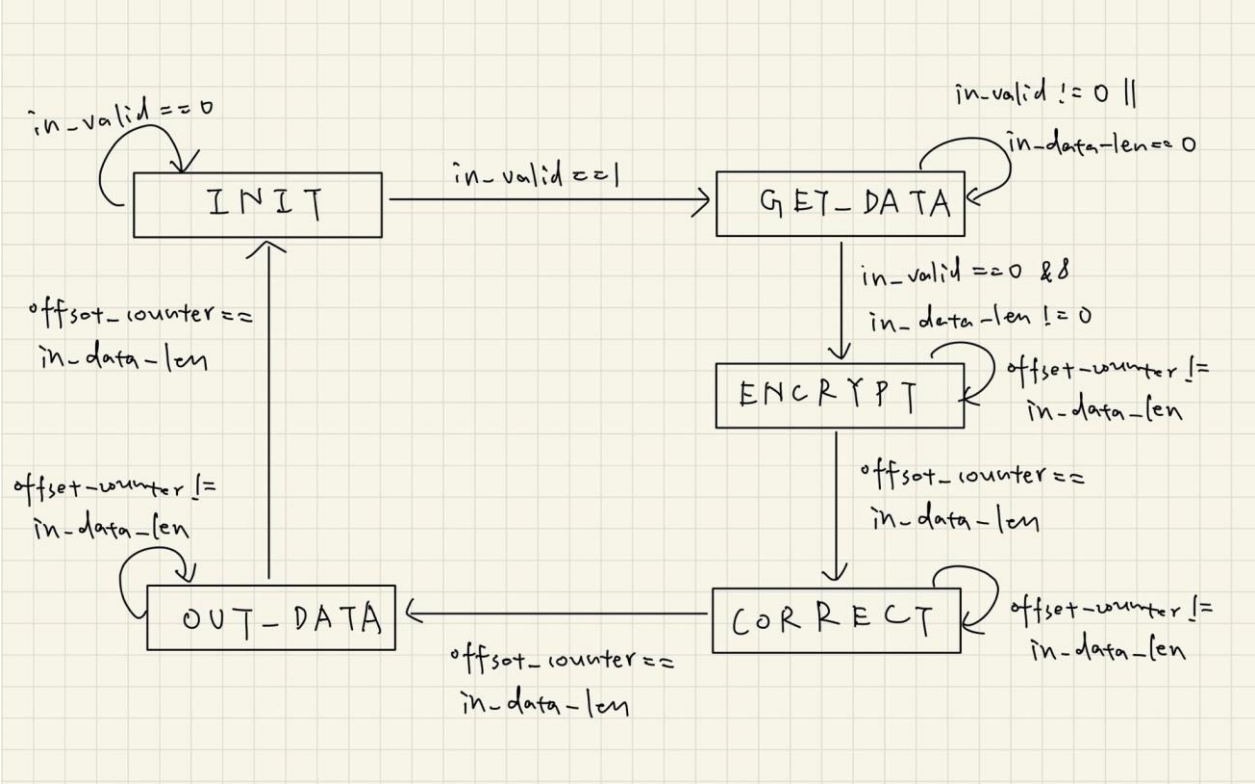
4. 用 offset_counter % 128 的方式來模擬 0~127 的 7-bit counter

```
220         ENCRYPT : begin
221             for (i=0; i<=MAX_INPUT_LEN; i=i+1) begin
222                 next_output_data_save[i] = output_data_save[i];
223             end
224             if (offset_counter != in_data_len) begin
225                 // + offset_counter % 128
226                 next_output_data_save[offset_counter] =
227                 output_data_save[offset_counter] + (offset_counter % 128);
228             end
229         end
```

5. 用 concat 的方式來加上 correct code 做成 12-bit 的 out_data

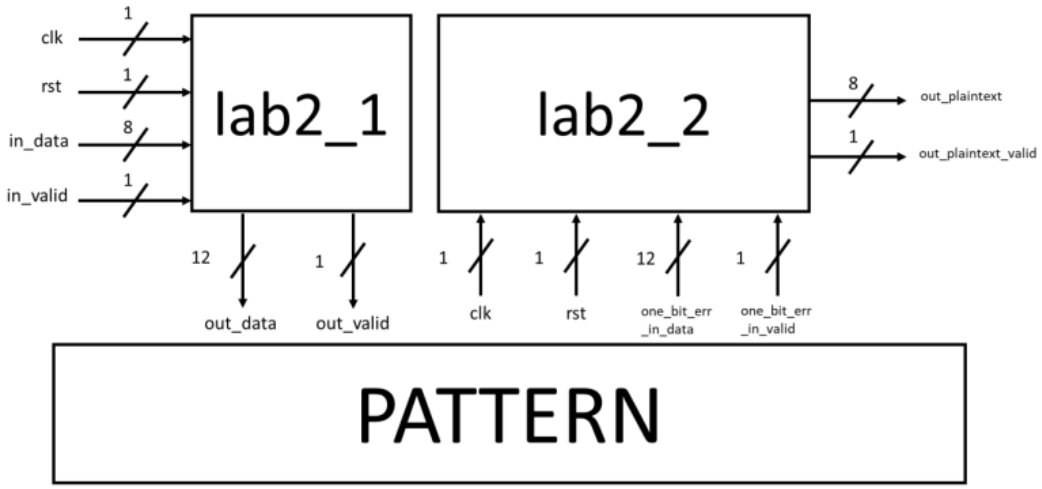
```
230         CORRECT : begin
231             for (i=0; i<=MAX_INPUT_LEN; i=i+1) begin
232                 next_output_data_save[i] = output_data_save[i];
233             end
234             if (offset_counter != in_data_len) begin
235                 // correct
236                 next_output_data_save[offset_counter] = {
237                     output_data_save[offset_counter][7:4],
238                     output_data_save[offset_counter][7] ^ output_data_save[offset_counter][6] ^ output_data_save
239                     output_data_save[offset_counter][3:1],
240                     output_data_save[offset_counter][7] ^ output_data_save[offset_counter][3] ^ output_data_save
241                     output_data_save[offset_counter][0],
242                     output_data_save[offset_counter][6] ^ output_data_save[offset_counter][5] ^ output_data_save
243                     output_data_save[offset_counter][6] ^ output_data_save[offset_counter][4] ^ output_data_save
244                 };
245             end
246         end
```

Lab2_1 FSM:



1. INIT -> GET_DATA:
   in_valid 是 1，代表正在接受 input。
2. GET_DATA -> ENCRYPT:
   in_valid 是 0，代表不在接收 input，如果此時 in_data_len 不是 0，就代表已經接收到 data 且接收完畢可以處理 data 了。
3. ENCRYPT -> CORRECT:
   offset_counter 數到 in_data_len 代表 data 已經全數處理好了可以進下一個階段。
4. CORRECT -> OUT_DATA:
   offset_counter 數到 in_data_len 代表 data 已經全數處理好了可以進下一個階段。
5. OUT_DATA -> INIT:
   offset_counter 數到 in_data_len 代表 data 已經全數處理好了可以進下一個階段。

Lab2_2: 依 Error correct algorithm 修正 12-bit 的 in_data，再用同樣的計數器邏輯解密 data 成 8-bit 的 out_data。



Lab2_2 segments:

1. 7-bit counter
2. 5 states

```
// state
parameter INIT = 3'd0;
parameter GET_DATA = 3'd1;
parameter FIX = 3'd2;
parameter DECRYPT = 3'd3; // CALC
parameter OUTPUT_DATA = 3'd4;
reg [3:0] state, next_state;
```

3. 依 code correct algorithm 來修正 data

```
// case(c1, c2, c3, c4)
case({output_data_save[offset_counter][0] ^ output_data_save[offset_counter][2] ^ output_data_sa
      output_data_save[offset_counter][1] ^ output_data_save[offset_counter][2] ^ output_data_sa
      output_data_save[offset_counter][3] ^ output_data_save[offset_counter][4] ^ output_data_sa
      output_data_save[offset_counter][7] ^ output_data_save[offset_counter][8] ^ output_data_sa
      })
    4'b1100: next_output_data_save[offset_counter][2] = ~output_data_save[offset_counter][2];
    4'b1010: next_output_data_save[offset_counter][4] = ~output_data_save[offset_counter][4];
    4'b0110: next_output_data_save[offset_counter][5] = ~output_data_save[offset_counter][5];
    4'b1110: next_output_data_save[offset_counter][6] = ~output_data_save[offset_counter][6];
    4'b1001: next_output_data_save[offset_counter][8] = ~output_data_save[offset_counter][8];
    4'b0101: next_output_data_save[offset_counter][9] = ~output_data_save[offset_counter][9];
    // not 4'b1100
    4'b1101: next_output_data_save[offset_counter][10] = ~output_data_save[offset_counter][10];
    4'b0011: next_output_data_save[offset_counter][11] = ~output_data_save[offset_counter][11];
    4'b1000: next_output_data_save[offset_counter][0] = ~output_data_save[offset_counter][0];
    4'b0100: next_output_data_save[offset_counter][1] = ~output_data_save[offset_counter][1];
```

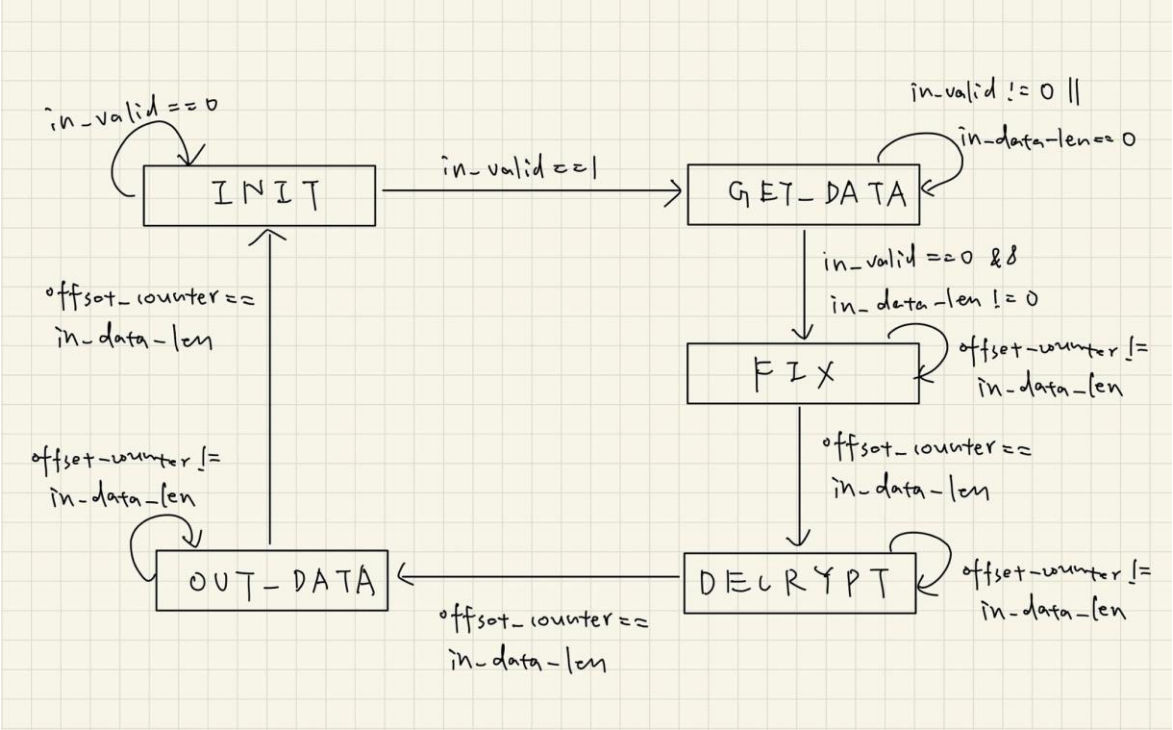
最後再用 concat 的方式做出 8-bit 的解密前 out_data (前面補 0)

```
next_output_data_save[offset_counter] =
{{4{1'b0}}, next_output_data_save[offset_counter][11:8], next_output_data_save[offset_counter][6:4], next_outpu
```

4. 解密

```
252         DECRYPT : begin
253             for (i=0; i<=MAX_INPUT_LEN; i=i+1) begin
254                 next_output_data_save[i] = output_data_save[i];
255             end
256             if (offset_counter != in_data_len) begin
257                 // - offset_counter % 128
258                 next_output_data_save[offset_counter] =
259                 output_data_save[offset_counter] - (offset_counter % 128);
260             end
261         end
```

Lab2_2 FSM:



1. INIT -> GET_DATA:
   in_valid 是 1，代表正在接受 input。
2. GET_DATA -> FIX:
   in_valid 是 0，代表不在接收 input，如果此時 in_data_len 不是 0，就代表已經接收到 data 且接收完畢可以處理 data 了。
3. FIX -> DECRYPT:
   offset_counter 數到 in_data_len 代表 data 已經全數處理好了可以進下一個階段。
4. DECRYPT -> OUT_DATA:
   offset_counter 數到 in_data_len 代表 data 已經全數處理好了可以進下一個階段。
5. OUT_DATA -> INIT:
   offset_counter 數到 in_data_len 代表 data 已經全數處理好了可以進下一個階段。

我在本次 lab 了解到了怎麼設計一個 FSM 還有用 verilog 實作，不僅學會了一種 error correct 的演算法，和重新複習了一遍 counter 的實作方式。

B. Questions and Discussions

A: 如果是非同步 reset 的話，只要 rst 是 HIGH，不論 clk 進來與否都會重置。

```
34  // state change engine
35  always @(posedge clk) begin
36      if (rst) begin
37          state <= INIT;
38      end
39      else begin
40          state <= next_state;
41      end
42  end
```

將這種有 if(rst) 的 always block 的 sensitivity list 加上 negedge rst 即可達成非同步 reset，例如：把 35 行改成 always @(posedge clk or negedge rst) begin。

B: 可以用 while、repeat 或是 forever+break。例如：

```
190         // for method
191         for (i=0; i<=MAX_INPUT_LEN; i=i+1) begin
192             output_data_save[i] <= 0;
193         end
194         // while method
195         i = 0;
196         while(i <= MAX_INPUT_LEN) begin
197             output_data_save[i] <= 0;
198             i = i+1;
199         end
200         // repeat method
201         i = 0;
202         repeat(MAX_INPUT_LEN) begin
203             output_data_save[i] <= 0;
204             i = i+1;
205         end
206         // forever
207         i = 0;
208         forever begin
209             output_data_save[i] <= 0;
210             i = i+1;
211             if(i > MAX_INPUT_LEN) begin
212                 break;
213             end
214         end
```

C. Problem Encountered

1. 一開始思考了很久，counter 究竟是每個 testcase 都會歸零還是會繼續數下去。解決辦法沒想到就是直接去比對 message 的那兩個檔案，馬上就真相大白了，又快又正確。
2. 在 lab2_2 fix bit error 實作的過程中，沒發現一開始給的 spec 的 B11 的 case 判斷是錯的，後來 debug 了一段時間才發現這邊怪怪的，於是自己整理了一次。雖然多花了一段時間 debug，但也因為這次自己整理的機會，能更加理解到為何透過 XOR 的使用，就能夠達成這個 Error correct algorithm，原來 XOR 的應用是這麼的廣。

D. Suggestions

希望之後的 spec 能開好！辛苦助教了！