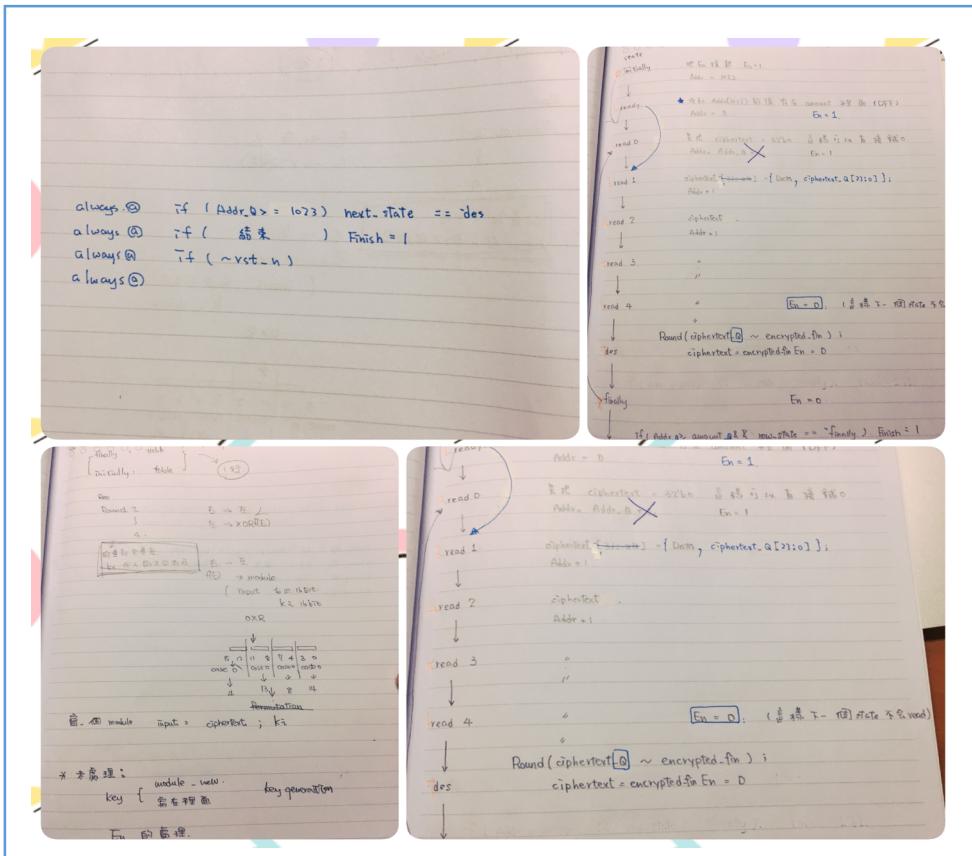


Lab4

105062139

林楷宸

1 設計：



這次 lab，我覺得最重要的部分大概就是設計我們的 state。

一開始，我先用紙筆打草稿，重複修改了好幾次，才有上面那份 state 設計，當然，打成 code 之後，還是有錯，所以又再修改過了幾次。

我認為設計 state 最重要的是每一個 state 要做什麼事，總共要有幾個 state，這也是我認為這次 lab 最難的地方，我也問過很多人，每個人的設計都不盡相同，光是 state 數就有好幾種，最後，我認為最佳的 state 數是 9 個，以下會說明這 9 個分別在做什麼。

2 state 介紹：

```
1 `define initially 10'd0
2 `define ready    10'd1
3 `define read0   10'd2
4 `define read1   10'd3
5 `define read2   10'd4
6 `define read3   10'd5
7 `define read4   10'd6
8 `define des     10'd7
9 `define finally 10'd8
10 //-
11 //-
```

state 0 : initially : ->ready

initially 有如 lab3 中的 IDLE，負責電路執行後，收取 Addr[1023]的值
(下面簡稱 **amount**)。

state 1 : ready : ->read1

ready 是用來將 Addr 設定為 Addr[0]的，不然之後進入 read 階段了話，Addr 就不能設定為零，所以在這裡多做一個 state 將 Addr 直接賦值為 Addr[0]。

state 2 : read0 : ->read1

ready 過後，會直接接 read1，而這個 read0，是用來在最後一個 state 結束後，將整個 ciphertext 變成 32'b0 在進入 read1，這樣之後如果資料不夠了話，就可以直接進入 des，而不用再補零，(因為他一開始就是 0，所以沒有賦值了話，就會一直等於 0)，

state 3 : read1 : ->read2

state 4 : read2 : ->read3

state 5 : read3 : ->read4

state 6 : read4 : ->des

由於這一次的 input 分為四組 8 bit 讀取，所以，我使用 4 個 state，來做這一個工作，這樣不只好控管所有 state 是否要讓 En = 1 or 0，同時也比較好監測到底是哪一個 state 會讀取錯的資料。

每一個 state 的工作如下：

```
382     `read1:begin
383         En = 1'b1;
384         ciphertext = {Data,ciphertext_Q[23:0]};
385     end
386     `read2:begin
387         En = 1'b1;
388         ciphertext = {ciphertext_Q[31:24], Data, ciphertext_Q[15:0]};
389     end
390     `read3:begin
391         En = 1'b1;
392         ciphertext = {ciphertext_Q[31:16], Data, ciphertext_Q[7:0]};
393     end
394     `read4:begin
395         // En = 1'b0;
396         En = 1'b1;
397         ciphertext = {ciphertext_Q[31:8], Data};
398     end
```

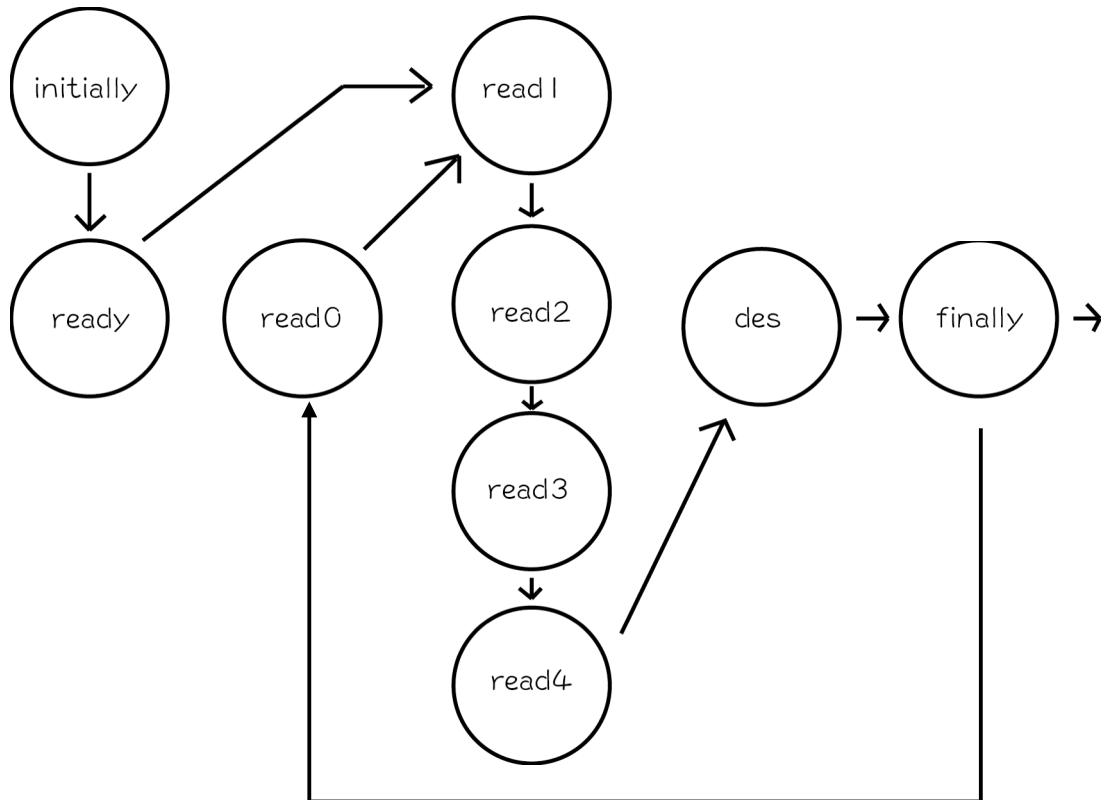
由上面的程式碼可知，只要是讀取的 state，En = 1，而 ciphertext 的部分，可以很方便的用大括號，把 Data 放在想放的地方，就不會再寫一些多餘的電路了。

state 7 : des : ->finally

由於在加密的部分為 combinational，所以，我將 Initial permutation，Round 1，Round 2，Round 3，Round 4，Final permutation，全部寫在同一個 state 裡面。而各個過程則是分成不同的 module 來處理。(下面會再詳細講解不同 module 的功用)。

state 8 : finally : ->read0

finally 就是輸出這次加密的結果。此時才把 Finish 拉起來，而下一個 state 的時候，再把 Finish 放下，也就是說只有這個 state，Finish 是拉起來的。



State 的關係圖大致上就如同上圖所展示的一樣，initially -> ready -> read1 -> read2 -> read3 -> read4 -> des -> finally -> read0 -> read1 -> read2 ->一直下去，直到資料讀到 amount 且 state 到達 finally 時，就會結束輪迴。

state 探討：

這一次的 state 設計，似乎比之前的還要多，我也問過很多人的 state 設計，但最後，我認為與其使用很多的 if else 或是其他判斷，或是用 counter 等的方式，沒有比這樣，把讀取分成 4 個 state 來讀取還要來得優，一來是因為分成四的 state 了話，就不用在進行特判，另一方面也是因為分成 4 個 state，我就能一一的把 ciphertext 進行與 Data 結合，不用再接多餘的電路。Code 寫起來也比較乾淨好看。

3 主要 module 介紹：

part 1：造 key：

```
258 //*****  
259 wire [27:0] kp;  
260 wire [27:0] k1,k2,k3,k4;  
261 wire [15:0] key1,key2,key3,key4;  
262 assign kp = { key[31:25], key[23:17], key[15:9],key[7:1]};  
263 assign k1 = { kp[26:14], kp[27],  
264           kp[12: 0] ,kp[13]  
265       };  
266 assign key1 = { k1[11], k1[ 9], k1[18], k1[ 6],  
267           k1[16], k1[ 8], k1[24], k1[17],  
268           k1[14], k1[23], k1[ 2], k1[26],  
269           k1[15], k1[ 1], k1[20], k1[25]  
270       };  
271 assign k2 = { k1[26:14], k1[27],  
272           k1[12: 0] ,k1[13]  
273       };  
274 assign key2 = { k2[11], k2[ 9], k2[18], k2[ 6],  
275           k2[16], k2[ 8], k2[24], k2[17],  
276           k2[14], k2[23], k2[ 2], k2[26],  
277           k2[15], k2[ 1], k2[20], k2[25]  
278       };  
279 assign k3 = { k2[26:14], k2[27],  
280           k2[12: 0] ,k2[13]  
281       };  
282 assign key3 = { k3[11], k3[ 9], k3[18], k3[ 6],  
283           k3[16], k3[ 8], k3[24], k3[17],  
284           k3[14], k3[23], k3[ 2], k3[26],  
285           k3[15], k3[ 1], k3[20], k3[25]  
286       };  
287 assign k4 = { k3[26:14], k3[27],  
288           k3[12: 0] ,k3[13]  
289       };  
290 assign key4 = { k4[11], k4[ 9], k4[18], k4[ 6],  
291           k4[16], k4[ 8], k4[24], k4[17],  
292           k4[14], k4[23], k4[ 2], k4[26],  
293           k4[15], k4[ 1], k4[20], k4[25]  
294       };  
295 //*****
```

由於 key1，key2，key3，key4 從一開始就會進來且不會再進行改變，所以我就直接將他 assign 住，不再改變他的值了，

kp	為 key 進行完 parity 的結果
k1~k4	為進行 left shift 後的結果
key1~key4	為 compression 後的結果

但之後跟同學討論後，得到了一個更好的寫法是先用 c++ 計算出 generation 後的結果直接進行 assign，但我認為這樣有一點點失去本來想要的理念，所以我還是維持這個寫法，雖然那樣寫可以減少一半的 area。

part 2 : 各式各樣的 module input and output :

```
295      //*****  
296      DFF #(32) dff_ciphertext (clk, rst_n, ciphertext, ciphertext_Q);  
297      DFF #(10) dff_addr      (clk, rst_n, Addr,   Addr_Q  );  
298      DFF #(10) dff_amount    (clk, rst_n, amount,  amount_Q);  
299      R_ini round_ini      (ciphertext_Q, encrypted_0);  
300      Round round_1        (encrypted_0, key1, encrypted_1);  
301      Round round_2        (encrypted_1, key2, encrypted_2);  
302      Round round_3        (encrypted_2, key3, encrypted_3);  
303      Round round_4        (encrypted_3, key4, encrypted_4);  
304      R_fin round_fin     (encrypted_4, encrypted_fin);  
305      //*****  
306
```

三個 DFF	傳值用
round_ini	進行 initialy permutation
round_1	進行第一次的加密
round_2	進行第二次的加密
round_3	進行第三次的加密
round_4	進行第四次的加密
round_fin	進行最後的 permutation

其中 5 個 round 都是在同一個 state 中進行的，所以可以發現，我直接把上一個的 output 接到下一個的 input 這樣進行運算。

part 3 : next_state 的判斷 :

```
313  
314      always(@*)begin  
315          if(Addr_Q == amount_Q-1 && now_state != `initially && now_state != `finally && now_state != `des )  next_state = `des;  
316          else if ( now_state == `initially)  next_state = `ready;  
317          else if ( now_state == `ready)  next_state = `read1;  
318          else if ( now_state == `read0)  next_state = `read1;  
319          else if ( now_state == `read1)  next_state = `read2;  
320          else if ( now_state == `read2)  next_state = `read3;  
321          else if ( now_state == `read3)  next_state = `read4;  
322          else if ( now_state == `read4)  next_state = `des;  
323          else if ( now_state == `des)  next_state = `finally;  
324          else if ( now_state == `finally)  next_state = `read0;  
325      end
```

第一個 if 是用來判斷，當所有資料都讀進來之後，就結束所有的 read

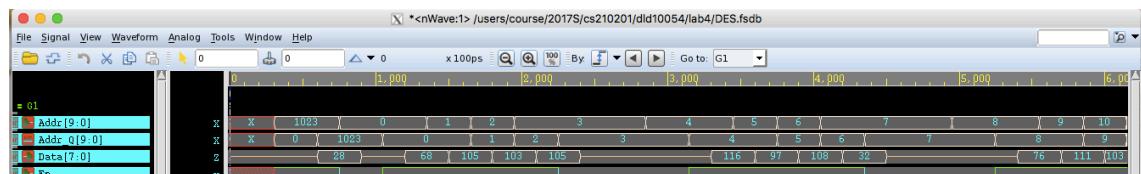
state，直接進入 des 進行加密。

part 4：Addr 的處理：

```
351 //*****
352 always@(*)begin
353     case(now_state)
354         `initially:Addr = 1023;
355         `ready      :Addr = 0;
356         `read0     :Addr = Addr_Q+1;
357         `read1     :Addr = Addr_Q;
358         `read2     :Addr = Addr_Q+1;
359         `read3     :Addr = Addr_Q+1;
360         `read4     :Addr = Addr_Q+1;
361         `des       :Addr = Addr_Q;
362         `finally   :Addr = Addr_Q;
363     endcase
```

這裡比較特別的地方在於 read1 的地方，我並沒有進行 Addr=Addr_Q+1 是因為，第一次剛從 ready 進入 read1 時，Addr 本身就是 Addr[0]，如果在+1 了話，他就讀不到 Addr[0] 了，所以這裡我改成在 read2～read4 和 read0 有進行 Addr = Addr_Q+1。

而其他的 state 則一直將 Addr 用 DFF hold 住，不讓他的值改變。



從 Wave 圖可以清楚的看出變化。

part 5 : ciphertext 跟 En 的賦值 :

```
365 //*****
366 always@(*)begin
367     case(now_state)
368         `initially:begin
369             En = 1'b1;
370             ciphertext = 32'b0;
371         end
372         `ready:begin
373             // En = 1'b1;
374             En = 1'b0;
375             ciphertext = 32'b0;
376         end
377         `read0:begin
378             En = 1'b0;
379             // En = 1'b1;
380             ciphertext = 32'b0;
381         end
382         `read1:begin
383             En = 1'b1;
384             ciphertext = {Data,ciphertext_Q[23:0]};
385         end
386         `read2:begin
387             En = 1'b1;
388             ciphertext = {ciphertext_Q[31:24], Data, ciphertext_Q[15:0]};
389         end
390         `read3:begin
391             En = 1'b1;
392             ciphertext = {ciphertext_Q[31:16], Data, ciphertext_Q[7:0]};
393         end
394         `read4:begin
395             // En = 1'b0;
396             En = 1'b1;
397             ciphertext = {ciphertext_Q[31:8], Data};
398         end
399         `des:begin
400             En = 1'b0;
401             ciphertext = encrypted_fin;
402         end
403         `finally:begin
404             En = 1'b0;
405             ciphertext = ciphertext_Q;
406         end
407     endcase
408 end
409
410 endmodule
411
```

4 else module 介紹：

Module 1 : DFF

這部分直接使用上一次 lab 的 DFF，所以沒什麼太大問題。

Module 2 : Round:

進行 permutaion 的 left、right 交換 並把 right 與 f-funtion X O R 的結果傳回去。

```
26 //-
27 //-
28 module Round(text, key_i, encrypted);
29     input text, key_i;
30     output encrypted;
31
32     wire [31:0] text;
33     wire [15:0] key_i;
34     wire [31:0] encrypted;
35     wire [15:0] fout;
36
37     f function(text[15:0], key_i, fout);
38
39     assign encrypted = {text[15:0], fout^text[31:16]};
40 endmodule
41 //-
42 //-
```

Module 3 : f-funtion

進行 X O R 跟 S – B O X 還有 Permutation，這裡我避免會寫錯，所以將整個排版寫得很整齊，並在必要時進行註解，避免自己哪裡少寫而導致 D E B U G 困難。

```

41 //-----
42 //-----
43 module f(right, key_i, Out);
44     input right, key_i;
45     output Out;
46
47     wire [15:0] right, key_i, temp, Out;
48     reg [15:0] t2;
49     assign temp = right ^ key_i;
50
51     always@(*)begin
52         case(temp[15:12])
53             0: t2[15:12] = 4'd 4;
54             1: t2[15:12] = 4'd 1;
55             2: t2[15:12] = 4'd 5;
56             3: t2[15:12] = 4'd 0;
57             4: t2[15:12] = 4'd11;
58             5: t2[15:12] = 4'd13;
59             6: t2[15:12] = 4'd15;
60             7: t2[15:12] = 4'd 8;
61             8: t2[15:12] = 4'd 6;
62             9: t2[15:12] = 4'd 3;
63             10: t2[15:12] = 4'd12;
64             11: t2[15:12] = 4'd14;
65             12: t2[15:12] = 4'd 2;
66             13: t2[15:12] = 4'd 7;
67             14: t2[15:12] = 4'd 9;
68             15: t2[15:12] = 4'd10;
69         endcase
70     end
71     always@(*)begin
72         case(temp[11: 8])
73             0: t2[11:8] = 4'd13;
74             1: t2[11:8] = 4'd 0;
75             2: t2[11:8] = 4'd 2;
76             3: t2[11:8] = 4'd12;
77             4: t2[11:8] = 4'd11;
78             5: t2[11:8] = 4'd 9;
79             6: t2[11:8] = 4'd 1;
80             7: t2[11:8] = 4'd 8;
81             8: t2[11:8] = 4'd 6;
82             9: t2[11:8] = 4'd14;
83             10: t2[11:8] = 4'd 7;
84             11: t2[11:8] = 4'd15;
85             12: t2[11:8] = 4'd 4;
86             13: t2[11:8] = 4'd10;
87             14: t2[11:8] = 4'd 3;
88             15: t2[11:8] = 4'd 5;
89         endcase
90     end
91
92     assign Out = {
93         t2[15], //15
94         t2[ 6], //14
95         t2[12], //13
96         t2[14], //12
97         t2[ 8], //11
98         t2[ 7], //10
99         t2[ 0], //09
100        t2[13], //08
101        t2[ 9], //07
102        t2[ 2], //06
103        t2[10], //05
104        t2[ 3], //04
105        t2[ 4], //03
106        t2[11], //02
107        t2[ 5], //01
108        t2[ 1] //00
109    };
110
111 endmodule
112 //-

```

```

91     always@(*)begin
92         case(temp[7: 4])
93             0: t2[7:4] = 4'd 8;
94             1: t2[7:4] = 4'd10;
95             2: t2[7:4] = 4'd11;
96             3: t2[7:4] = 4'd 2;
97             4: t2[7:4] = 4'd 0;
98             5: t2[7:4] = 4'd15;
99             6: t2[7:4] = 4'd 9;
100            7: t2[7:4] = 4'd3;
101            8: t2[7:4] = 4'd14;
102            9: t2[7:4] = 4'd 5;
103            10: t2[7:4] = 4'd 1;
104            11: t2[7:4] = 4'd12;
105            12: t2[7:4] = 4'd 7;
106            13: t2[7:4] = 4'd 4;
107            14: t2[7:4] = 4'd 6;
108            15: t2[7:4] = 4'd13;
109        endcase
110    end
111    always@(*)begin
112        case(temp[3: 0])
113            0: t2[3:0] = 4'd14;
114            1: t2[3:0] = 4'd13;
115            2: t2[3:0] = 4'd8;
116            3: t2[3:0] = 4'd7;
117            4: t2[3:0] = 4'd11;
118            5: t2[3:0] = 4'd3;
119            6: t2[3:0] = 4'd15;
120            7: t2[3:0] = 4'd2;
121            8: t2[3:0] = 4'd6;
122            9: t2[3:0] = 4'd9;
123            10: t2[3:0] = 4'd5;
124            11: t2[3:0] = 4'd0;
125            12: t2[3:0] = 4'd12;
126            13: t2[3:0] = 4'd10;
127            14: t2[3:0] = 4'd1;
128            15: t2[3:0] = 4'd4;
129        endcase
130    end
131

```

Module 3 and Module 4 : initial permutation and final permutation

整齊的排版成就有效率的 D E B U G ，這點我認為很重要的，畢竟一開始我的 permutation table 錯了“一個”數字，所以導致我無法全對，不過也因為一開始就有排版整齊，所以，在跟同學對數字時，很快就找出錯誤。

```
151 //-----  
152 module R_ini(text,text_out);  
153     input text;  
154     output text_out;  
155  
156     wire [31:0] text;  
157     wire [31:0] text_out;  
158  
159     assign text_out = { text[31],//31  
160             text[20],//30  
161             text[ 4],//29  
162             text[26],//28  
163             text[28],//27  
164             text[18],//26  
165             text[16],//25  
166             text[ 9],//24  
167             text[ 5],//23  
168             text[ 6],//22  
169             text[21],//21  
170             text[23],//20  
171             text[19],//19  
172             text[ 2],//18  
173             text[13],//17  
174             text[22],//16  
175             text[30],//15  
176             text[25],//14  
177             text[ 1],//13  
178             text[24],//12  
179             text[12],//11  
180             text[15],//10  
181             text[27],//9  
182             text[29],//8  
183             text[17],//7  
184             text[10],//6  
185             text[14],//5  
186             text[ 0],//4  
187             text[ 8],//3  
188             text[ 7],//2  
189             text[11],//1  
190             text[ 3],//0  
191         };  
192     endmodule  
193 //-----  
194 //-----  
195 module R_fin(text,text_out);  
196     input text;  
197     output text_out;  
198  
199     assign text_out = { text[31],//31  
200             text[15],//30  
201             text[ 8],//29  
202             text[27],//28  
203             text[ 9],//27  
204             text[28],//26  
205             text[14],//25  
206             text[12],//24  
207             text[20],//23  
208             text[16],//22  
209             text[21],//21  
210             text[30],//20  
211             text[19],//19  
212             text[ 2],//18  
213             text[13],//17  
214             text[22],//16  
215             text[30],//15  
216             text[25],//14  
217             text[ 1],//13  
218             text[24],//12  
219             text[12],//11  
220             text[15],//10  
221             text[27],//9  
222             text[29],//8  
223             text[17],//7  
224             text[10],//6  
225             text[14],//5  
226             text[ 0],//4  
227             text[ 8],//3  
228             text[ 7],//2  
229             text[11],//1  
230             text[ 3],//0  
231         };  
232     endmodule  
233 //-----  
234 //-----
```

5 總結 and 問題討論：

經過上次 lab 的洗禮後，我認為我對 sequential 的理解及實作有更進一步的成長，所以在這一次的 lab 上花的時間主要是在架構，而不是 debug 又或是專研一些語法，或是一些莫名其妙的 make syn 問題，像上次 lab，make syn 總是會錯誤，而這部分卻是電腦無法給一個正確的解答好讓我修改的。導致

我一直 Wrong Answer。但這次，由於對 verilog 有了更進一步的認識，所以在寫法上進行了很大的進步，也讓我在 make syn 能一次就過。這算是這次 lab 最大的收穫吧。至於這次 lab 的問題討論，其實沒有很多，大部分都是在架構時，要考慮很多因素，然後再實作後發現有問題，再回去重新架構，並重新改寫自己的 code，來來回回有三次左右吧。但基本上都不是太大的問題。

```
linkaichen — ssh -X dld10054@nthucad.cs.nthu.edu.tw — 143x47
Warning! Timing violation
$setuphold<setup>{ negedge G && (SandR == 1):1185989 PS, negedge D:1185959 PS, 0.116 : 116 PS, 0.076 : 76 PS };
File: /theda21_2/CBDK_IC_Contest/cur/Verilog/tsmc13.v, line = 26241
Scope: OP_tb.des.\next_state_reg[2]
Time: 1185989 PS

Warning! Timing violation
$setuphold<setup>{ negedge G && (SandR == 1):1245876 PS, negedge D:1245843 PS, 0.116 : 116 PS, 0.076 : 76 PS };
File: /theda21_2/CBDK_IC_Contest/cur/Verilog/tsmc13.v, line = 26241
Scope: OP_tb.des.\next_state_reg[2]
Time: 1245876 PS

time: 1313, ciphertext: b8597d18, ans: b8597d18

Warning! Timing violation
$setuphold<hold>{ negedge G && (SandR == 1):1335950 PS, negedge D:1335957 PS, 0.109 : 109 PS, 0.083 : 83 PS };
File: /theda21_2/CBDK_IC_Contest/cur/Verilog/tsmc13.v, line = 26241
Scope: OP_tb.des.\next_state_reg[1]
Time: 1335957 PS

Warning! Timing violation
$setuphold<setup>{ negedge G && (SandR == 1):1395989 PS, negedge D:1395959 PS, 0.116 : 116 PS, 0.076 : 76 PS };
File: /theda21_2/CBDK_IC_Contest/cur/Verilog/tsmc13.v, line = 26241
Scope: OP_tb.des.\next_state_reg[2]
Time: 1395989 PS

Warning! Timing violation
$setuphold<setup>{ negedge G && (SandR == 1):1455876 PS, negedge D:1455843 PS, 0.116 : 116 PS, 0.076 : 76 PS };
File: /theda21_2/CBDK_IC_Contest/cur/Verilog/tsmc13.v, line = 26241
Scope: OP_tb.des.\next_state_reg[2]
Time: 1455876 PS

time: 1523, ciphertext: 227d5d41, ans: 227d5d41

Score: 7 / 7
Congratulations!!!

Simulation complete via $finish(1) at time 1522902 PS + 0
./DES_tb.v:110          $finish;
ncsim> exit
[dld10054@ic27 ~/lab4]$
```

最後附上一張 make syn 第一次就成功的圖。。