Lab 1

學號: 109000205 姓名: 蕭皓隆

1. 實作過程

2-1：

利用 tempA 暫存 An 的運算結果；利用 tempB 暫存 Bn 的運算結果。n 為記錄目前第幾項。up_down 紀錄目前為往上數 or 往下數(up_down=0 往上數，up_down=1 往下數)。

```
8     reg [5:0] An, Bn, tempA, tempB;
9     reg [5:0] n;
10    reg up_down; //up=0, down=1;
```

在第一個 always block 裡面判斷 rst，若 rst=0 則進一步判斷 up_down 並進行 An 或 Bn 的運算，做完運算後將 n+1。

```
19    always @(posedge clk or posedge rst) begin
20        if(rst) begin
21            An = 0;
22            Bn = 6'd63;
23            n = 1;
24            up_down = 0;
25        end
26        else begin
27            if(!up_down) begin //up
28                if(An > n) begin
29                    tempA = An - n;
30                    An = tempA;
31                    n = n + 1;
32                end
33                else begin
34                    tempA = An + n;
35                    An = tempA;
36                    n = n + 1;
37                end
38            end
39            else begin //down
40                tempB = Bn - 2**(n-1);
41                Bn = tempB;
42                n = n + 1;
43            end
44        end
45    end
```

在第二個 always block 中則是判斷邊界條件，若 An==63 時代表要開始計算 Bn(往下數)並將 An 歸零以及將 n 調至正確的項數；Bn==0 時代表要開始計算 An(往上數)並將 Bn 回復至 63 以及將 n 調至正確的項數。

```
47      always @ (posedge clk) begin
48          if(An == 6'd63) begin
49              up_down <= 1;
50              An <= 0;
51              Bn <= 6'd63;
52              n <= 1;
53          end
54          else if(Bn == 0) begin
55              up_down <= 0;
56              An <= 0;
57              Bn <= 6'd63;
58              n <= 1;
59          end
60          else begin
61              up_down <= up_down;
62              An <= An;
63              Bn <= Bn;
64              n <= n;
65          end
66      end
```

最後一個 always block 則是依據往上數 or 往下數來決定要將 An 或是 Bn 給 output out。

```
68      always @ (*) begin
69          if(!up_down) begin
70              out <= An;
71          end
72          else begin
73              out <= Bn;
74          end
75      end
```

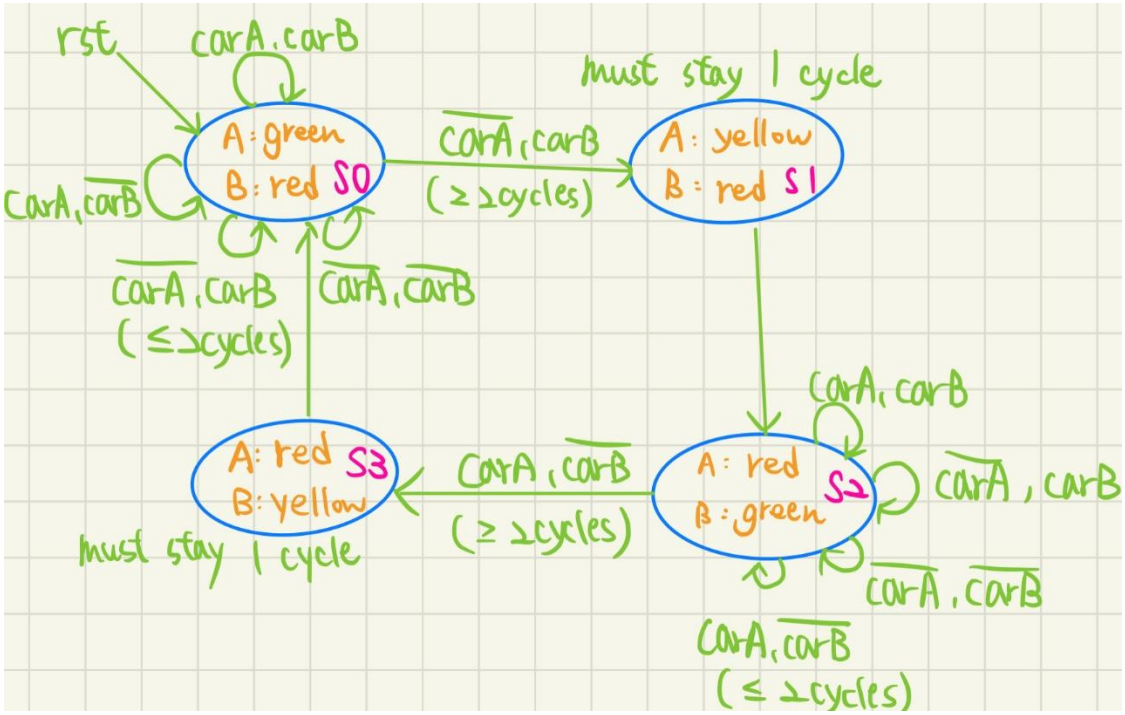
2-1 testbench：

Instantiate module lab2_1 後利用 always 讓 clk 運作，接下來用 initial begin 給予測試訊號，並在不同時間點測試 rst 是否正常運作。

```
1   `timescale 1ns/100ps
2
3   module lab2_1_t;
4
5       reg clk, rst;
6       wire [5:0] out;
7
8       lab2_1 c(.clk(clk), .rst(rst), .out(out));
9
10      always #10 clk = ~clk;
11
12      initial begin
13          clk = 1'b1;
14          rst = 1'b0;
15          #75
16          rst = 1'b1;
17          #25
18          rst = 1'b0;
19          #70
20          rst = 1'b1;
21          #60
22          rst = 1'b0;
23
24          //$monitor($time, " clk=%d, rst=%d, out=%d ", clk, rst, out);
25      end
26
27  endmodule
```

2-2：

利用 FSM 實作 moore machine，以下為 State Diagram：



使用 parameter 紀錄 4 個 states 的代號，state, next_state, pre_state 分別代表當前狀態、下個狀態、上一個狀態。

```
1   `timescale 1ns/100ps
2
3   module lab2_2 (
4       input clk,
5       input rst,
6       input carA,
7       input carB,
8       output reg [2:0] lightA,
9       output reg [2:0] lightB);
10
11      parameter S0 = 2'b00; //A : green, B : red;
12      parameter S1 = 2'b01; //A : yellow, B : red;
13      parameter S2 = 2'b10; //A : red, B : green;
14      parameter S3 = 2'b11; //A : red, B : yellow;
15      parameter unknown = 3'b111;
16      reg [2:0] state, next_state, pre_state;
```

第一個 always block 偵測 rst 是否為 1，若不為 1 則隨 clk 進行狀態更新，使用 pre_state 紀錄的原因為若 pre_state == state 的話就代表已在此 state 停留至少(含)2 個 cycle 以上了。

```
18      always @(posedge clk, posedge rst) begin
19          if(rst) begin
20              state <= S0;
21              pre_state <= unknown;
22          end
23          else begin
24              pre_state <= state;
25              state <= next_state;
26          end
27      end
```

最後一個 always block 則用來利用 case 來判斷狀態，由於是 moore machine，所以一進入 state 就給 lightA、lightB output，並用 if 來判斷是否要切換狀態抑或是 unchanged。

```
29      always @(*) begin
30          next_state = S0;
31          case (state)
32              S0 : begin
33                  lightA = 3'b001;
34                  lightB = 3'b100;
35                  if(!carA && carB && pre_state == S0)
36                      next_state = S1;
37                  else
38                      next_state = S0;
39              end
40              S1 : begin
41                  lightA = 3'b010;
42                  lightB = 3'b100;
43                  next_state = S2;
44              end
45              S2 : begin
46                  lightA = 3'b100;
47                  lightB = 3'b001;
48                  if(carA && !carB && pre_state == S2)
49                      next_state = S3;
50                  else
51                      next_state = S2;
52              end
53              S3 : begin
54                  lightA = 3'b100;
55                  lightB = 3'b010;
56                  next_state = S0;
57              end
58              default : begin
59                  lightA = 3'b001;
60                  lightB = 3'b100;
61                  next_state = S0;
62              end
63          endcase
64      end
```

2. 學到的東西與遇到的困難

學到的東西：自己實作 FSM 真的可以學到蠻多東西，例如了解整體運作狀態以及自己畫出 state diagram，這些技能在很多地方都會重複使用到。

遇到的困難：剛開始會擔心自己沒有考慮到所有 state，以及擔心自己是否能真正實作出想像中的 finite state machine。

3. 想對老師或助教說的話

謝謝助教幫我們一對一 demo lab1 lab2，也謝謝助教以及老師積極回答我們的問題！