

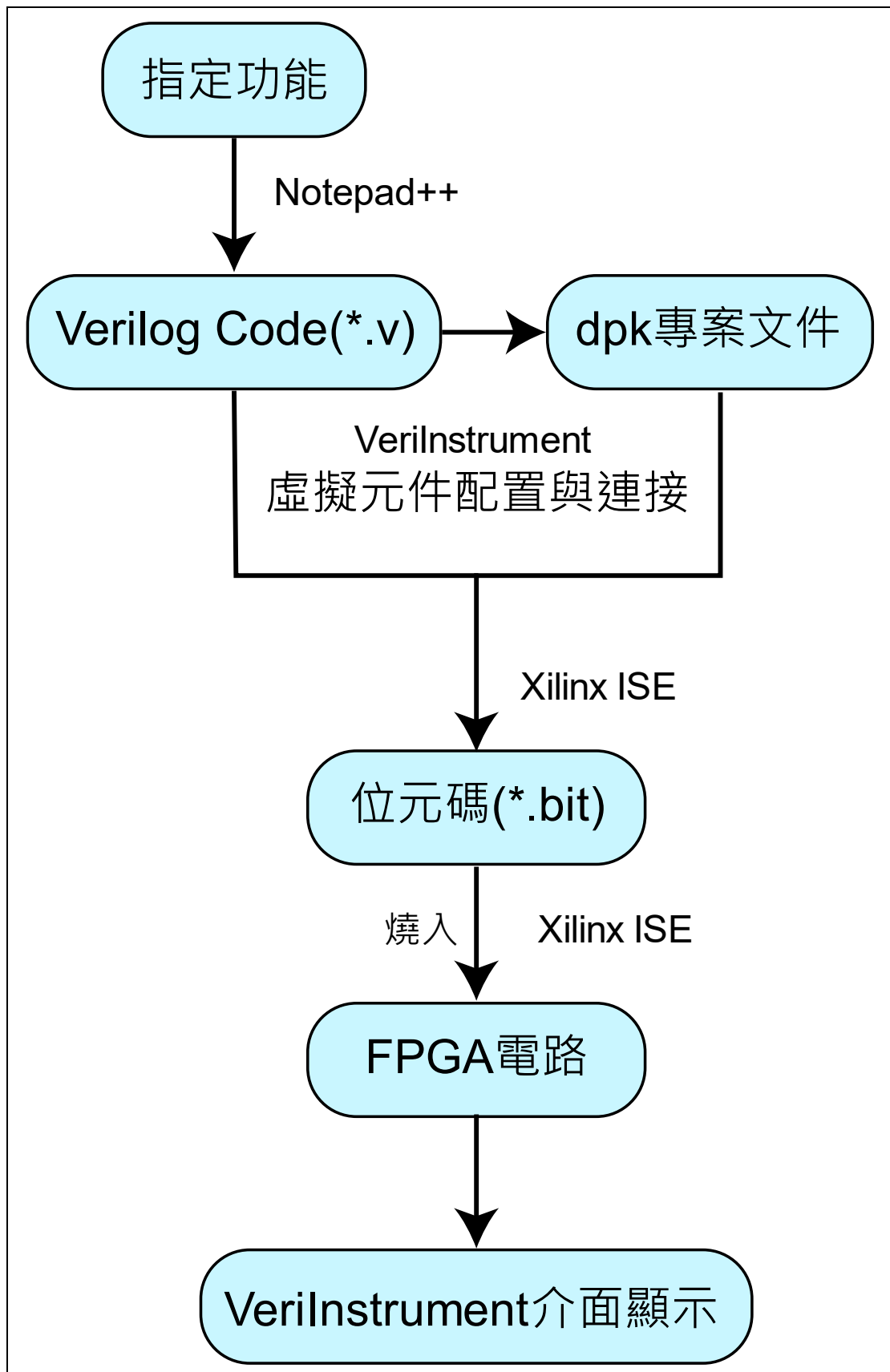
邏輯系統實驗

Lab 10

2021/05/13(四)

第 1 組	
組員姓名	學號
陳旭祺	E24099059
張振杰	E24085034
何啟造	E34085337

● 實驗流程圖(同實驗 Lab7、Lab8、Lab9)



● 實作題(一):

1. Verilog Code

Oneshot.v
<pre> module oneshot(clk,din,dout); input clk,din; output dout; reg [1:0] cs; always@(posedge clk) begin case(cs) 2'b00: begin if(din) cs <= 2'b01; end 2'b01: cs <= 2'b10; 2'b10: begin if(~din) cs <= 2'b00; end default: cs <= 2'b00; endcase end assign dout = cs[0]; endmodule </pre>
lab10_1.v
<pre> module lab10_1(clk,reset,up,down,left,right,R,C); input clk; input reset; input up,down,left,right; output reg [7:0] R,C; reg[7:0] R_t,C_t; wire up_oneshot,down_oneshot,left_oneshot,right_oneshot; //de-bounce logic </pre>

```

oneshot u_oneshot1(clk,up, up_oneshot);
oneshot u_oneshot2(clk,down, down_oneshot);
oneshot u_oneshot3(clk,left, left_oneshot);
oneshot u_oneshot4(clk,right,right_oneshot);

//8*8 LED row & column control
always@(posedge clk or posedge reset) begin
    if(reset) begin
        R <= 8'b00000001;
        C <= 8'b00000001;
    end

    else begin
        R <= R_t;
        C <= C_t;
    end
end

always@(*) begin
    //row
    if(down_oneshot) begin
        R_t[7:1] = R[6:0];
        R_t[0] = R[7];
    end

    else if(up_oneshot) begin
        R_t[6:0] = R[7:1];
        R_t[7] = R[0];
    end

    else
        R_t = R;
end

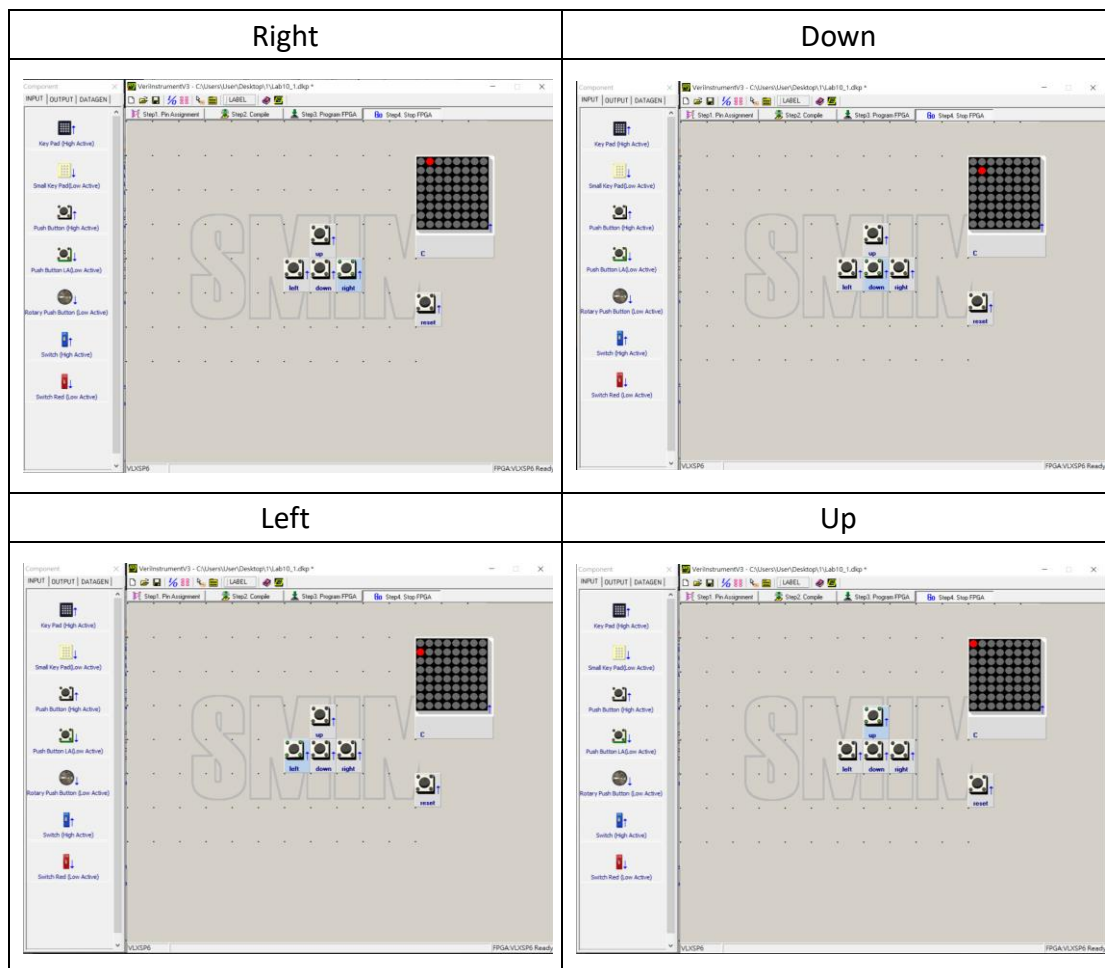
always@(*) begin
    //column
    if(right_oneshot) begin
        C_t[7:1] = C[6:0];

```

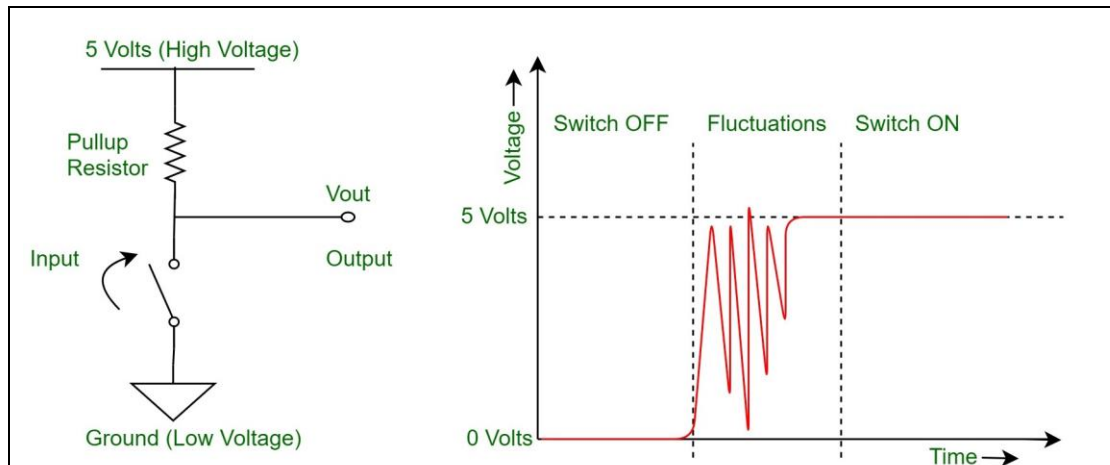
邏輯系統實驗 Lab10 第 1 組

```
C_t[0] = C[7];  
end  
  
else if(left_one-shot) begin  
    C_t[6:0] = C[7:1];  
    C_t[7] = C[0];  
end  
  
else  
    C_t = C;  
end  
  
endmodule
```

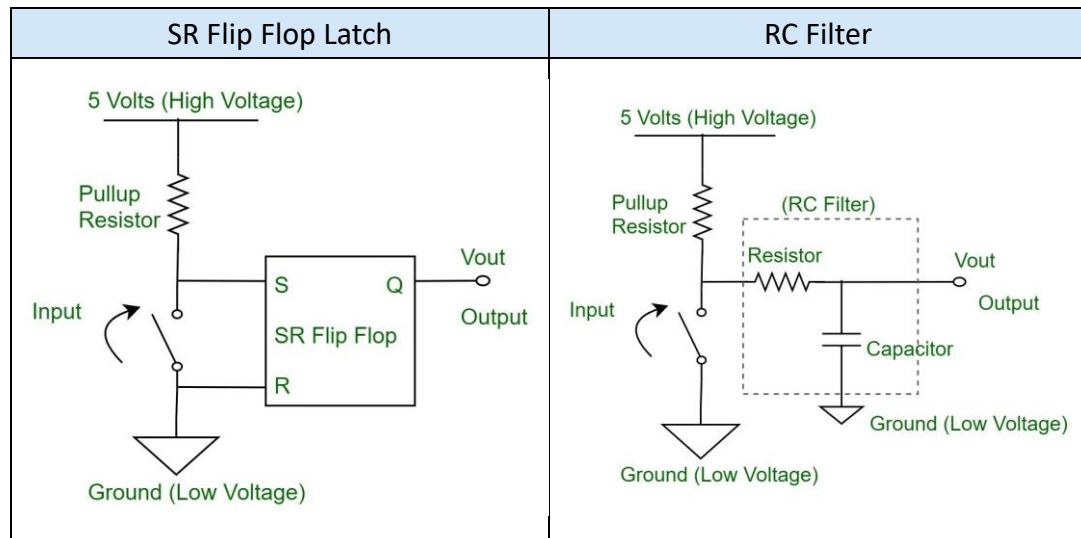
2. FPGA result



3. 功能解釋



由於按鍵屬於機械結構,按下時會有機械彈跳的狀況出現,在 0 與 1 之間多次彈跳,因此有兩個做法:一是從軟體程式去下手;而二是從硬體電路下手,設計 Debounce 電路,即為本次實際 oneshot.v 這個 module 的功能,程式是用一個 state machine 去解決,一開始初始化為第 1 個 state,輸出為 0;當正緣觸發且輸入為 1 時,進入第 2 的 state 輸出是 1;下一個 CLK 就卡在第 3 個 state,輸出為 0;一直等到正緣觸發且輸入為 0 時才回到原本第 1 個 state,才可接受下一次輸入的訊號,否則會被卡在第 3 個 state,輸出為 0,即視為同一次的輸入訊號,在第 2 的 state 輸出是 1 時已經傳過該值了,因此不會在重新輸出 1,簡單來說如果使用者長按按鈕時只會在第 2 個 state 經過一次輸出一訊號而已,而不會重複讀值。以下兩種為實際硬體接線的解決方法:



● 實作題(二):

1. Verilog Code

<code>`include "oneshot.v"</code>
<code>clock_div.v</code>
<pre> module clock_div(clk_48MHZ, clk_1HZ); //reduce clock frequency input clk_48MHZ; output reg clk_1HZ; reg[31:0] counter_1HZ; always@(posedge clk_48MHZ) begin if(counter_1HZ == 32'd48000_000) counter_1HZ <= 32'd0; //return to 0 else counter_1HZ <= counter_1HZ + 32'b1; //plus 1 repeatedly if(counter_1HZ < 32'd24000_000) clk_1HZ <= 1'd1; else clk_1HZ <= 1'd0; end endmodule </pre>
<code>lab10_2.v</code>
<pre> module lab10_2(clk,reset,pb,R,C); input clk; input reset; input pb; output reg [7:0] R; output reg [7:0] C; reg [7:0] C1,C2,C3,C4,C5; reg [2:0] display; reg [2:0] counter; </pre>

```

wire pb_one-shot;
wire reset_one-shot;
wire clk_1000HZ;

//de-bounce logic
one-shot u_one-shot1(clk,pb,pb_one-shot);
one-shot u_one-shot2(clk,reset,reset_one-shot);

//reduce clock frequency
clock_div u_clock_div(clk,clk_1000HZ);

always@(posedge clk_1000HZ or posedge reset_one-shot)
begin
    if(reset_one-shot)
        counter <= 3'b0;
    else
        counter <= counter + 3'b1;
//Overflow = counter return 0
end
//L
always@(counter) begin
    case(counter)
        3'd0: C1 = 8'b00000000;
        3'd1: C1 = 8'b00000100;
        3'd2: C1 = 8'b00000100;
        3'd3: C1 = 8'b00000100;
        3'd4: C1 = 8'b00000100;
        3'd5: C1 = 8'b00111100;
        3'd6: C1 = 8'b00000000;
        3'd7: C1 = 8'b00000000;
    endcase
end

//0
always@(counter) begin
    case(counter)
        3'd0: C2 = 8'b00000000;
        3'd1: C2 = 8'b00111100;

```



```

        3'd2: C2 = 8'b00100100;
        3'd3: C2 = 8'b00100100;
        3'd4: C2 = 8'b00100100;
        3'd5: C2 = 8'b00111100;
        3'd6: C2 = 8'b00000000;
        3'd7: C2 = 8'b00000000;
    endcase
end

//G
always@(counter) begin
    case(counter)
        3'd0: C3 = 8'b00000000;
        3'd1: C3 = 8'b01110000;
        3'd2: C3 = 8'b00001000;
        3'd3: C3 = 8'b00000100;
        3'd4: C3 = 8'b01100100;
        3'd5: C3 = 8'b01011000;
        3'd6: C3 = 8'b01110000;
        3'd7: C3 = 8'b00000000;
    endcase
end

//I
always@(counter) begin
    case(counter)
        3'd0: C4 = 8'b00000000;
        3'd1: C4 = 8'b00011100;
        3'd2: C4 = 8'b00001000;
        3'd3: C4 = 8'b00001000;
        3'd4: C4 = 8'b00001000;
        3'd5: C4 = 8'b00011100;
        3'd6: C4 = 8'b00000000;
        3'd7: C4 = 8'b00000000;
    endcase
end

//C

```

```

always@(counter) begin
    case(counter)
        3'd0: C5 = 8'b00000000;
        3'd1: C5 = 8'b01110000;
        3'd2: C5 = 8'b00001000;
        3'd3: C5 = 8'b00000100;
        3'd4: C5 = 8'b00000100;
        3'd5: C5 = 8'b00001000;
        3'd6: C5 = 8'b01110000;
        3'd7: C5 = 8'b00000000;
    endcase
end

//display = display + 1 when posedge pb_one-shot is
triggered
//If reset_one-shot == 1 or display == 4, display will
return 0
always@(posedge pb_one-shot or posedge reset_one-shot)
begin
    if(reset_one-shot | display == 3'd4)
        display <= 3'd0;
    else
        display <= display + 3'd1;
    end

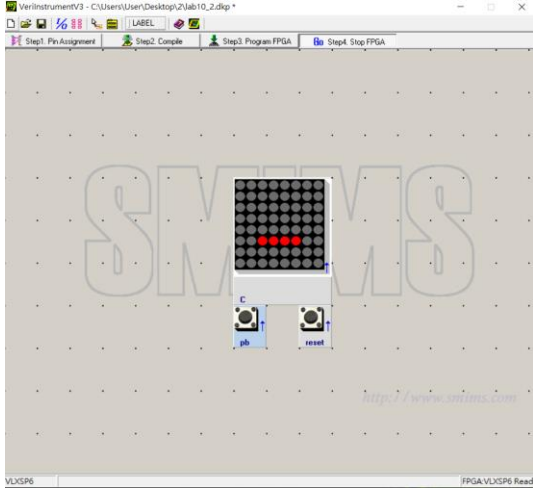
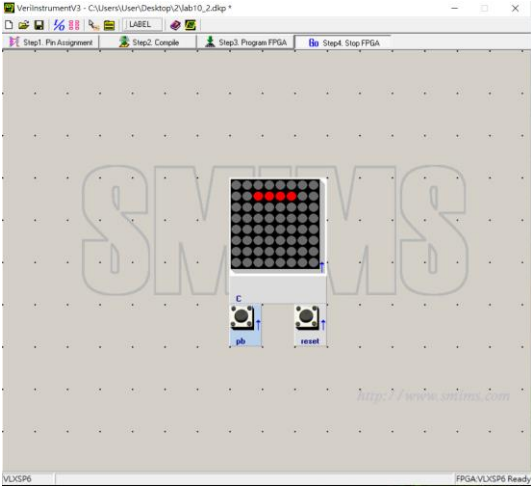
//To choose display which alphabet is
always@(*) begin
    case(display)
        3'd0: C = C1;
        3'd1: C = C2;
        3'd2: C = C3;
        3'd3: C = C4;
        3'd4: C = C5;
        default: C = C1;
    endcase
end

//scan row one by one by clock

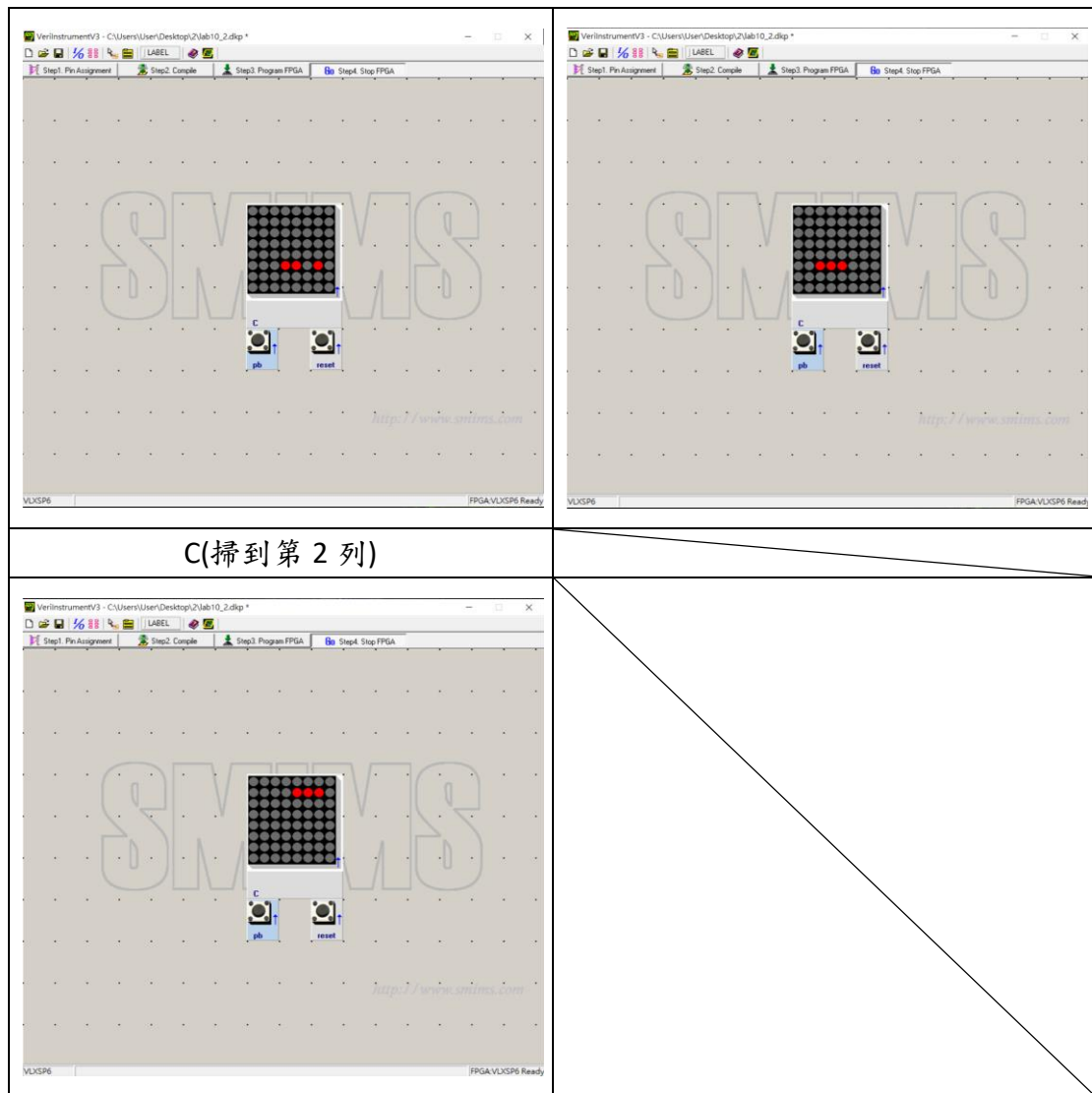
```

```
//R[X]=1 will make the value in that row valid
always@(posedge clk_1000HZ or posedge reset_onehot)
begin
    if(reset_onehot)
        R <= 8'b00000001;
    else begin
        R[7:1] <= R[6:0];
        R[0] <= R[7];
    end
end
endmodule
```

2. FPGA result

L(掃到第 6 列)	O(掃到第 2 列)
	
G(掃到第 6 列)	I(掃到第 6 列)

邏輯系統實驗 Lab10 第 1 組



C(掃到第 2 列)

3. 功能解釋

延續實作一的内容做出 L、O、G、I、C 圖示逐行掃描的類似幻燈片的效果，reset 鍵重置顯示第一個字母“L”；每一次 push button 就顯示下一個文字，到“C”再按下 push button 則回到“L”。原理為利用每次顯示字母時讓counter從 0 數到 7，由於counter只有 3 個 bit 因此數到 8 時會 overflow 變為 0，剛好就是重新繼續數下個字母，接下來我以舉例的方式進行說明，設定 $counter == 3'd0:C1 = 8'b11000000$ ；就是指定 Column 這排的燈要亮與否，以這個狀況來說就是後兩個 LED 陣列的燈會亮，而還需指定 row，假定 $R == 8'b00000001$ ，代表 row 的第 1 列 enable，只有在 $row == 1$ 與 $column == 1$ 時 LED 才會亮(兩者的交集)，總結來說就是第 1 列的後兩排 LED 會亮。因此利用這個性質，讓counter加下去，row 換做第 2 列 enable，而 column 改變指定的數值，就可以做出逐行掃描的 LED 效果了。

● 心得

1. 組員一 陳旭祺

這次實驗卡最久的是實作二一開始宣告的位元數錯誤，而又一直沒有發現，以至於最後挑戰題沒能做完，有些可惜。

<code>reg [7:0] counter;</code> 應更正為 <code>reg [2:0] counter;</code>
--

加上其實這種直接燒到 FPGA 檢查錯誤的方法，我認為沒有很好 debug，如果寫 top module 相應的 testbench 去跑模擬看波型或許會更有效率去 debug。由於本土疫情延燒，這次實驗應該是最後一次跑 FPGA 了，好在也只剩下兩次實驗和 Final Project，這個影響已經算小了，希望下學期疫情能回穩，實體上課與做實驗，對我來說學習效率還是比較高的。

2. 組員二 張振杰

這次實驗主要是透過除頻的 code 來去除 button 在切換時所產生的震盪現象，然後在主程式呼叫除頻程式從而達到除頻效果，之後再主程式做設計。在 8x8 LED 矩陣操作的部分，我對部分的 code 之間的銜接還不是很清楚，簡單來說就是比較難想像，導致在 debug 的時候給不了實質的幫助，也麻煩了助教幫我們 debug。

3. 組員三 何啟造

這次的實驗是做讓一個光點在 LED 矩陣上移動和讓 LED 顯示字母。這個實驗最難的部分是搞懂它的邏輯，本來在光點那部分就思考了很久了，但沒想到 LED 的問題更大，就是一直顯示不出來。到後面雖然顯示出了第一個 L，但還是沒辦法使用按鈕來切換成別的字母。不過好在最後成功解決問題完成實驗，感謝助教的幫忙。