Lab 6

學號: 109062173 姓名: 葉昱揚

A. Lab Implementation

1. Block diagram

Lab6-1

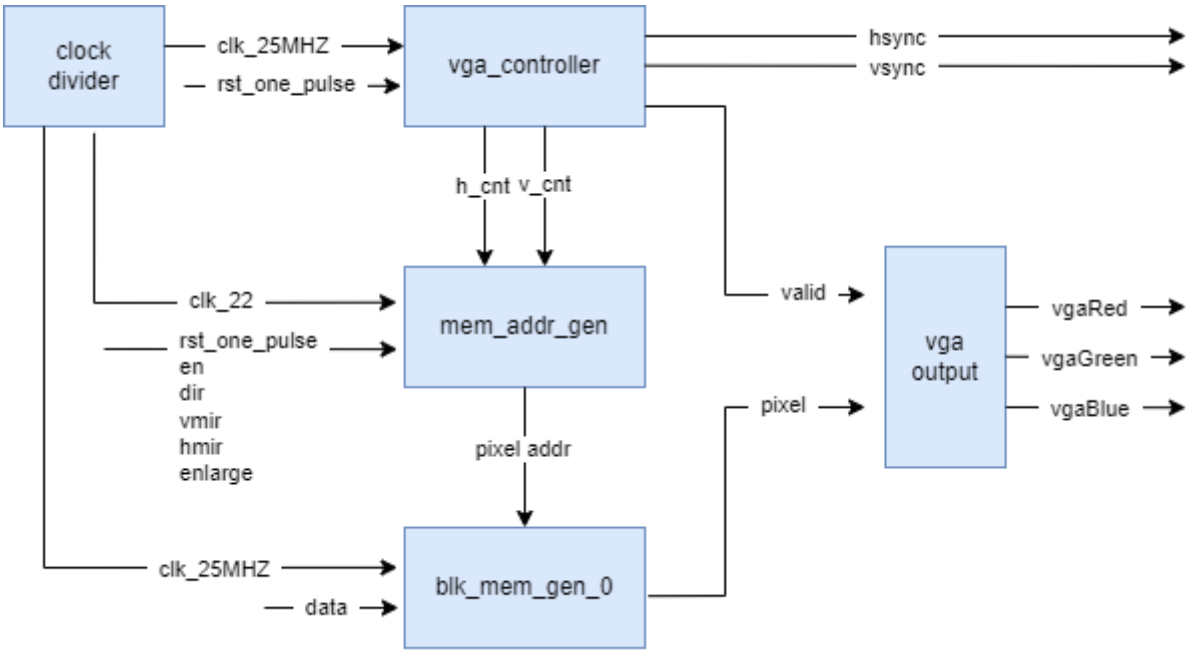

圖 1。6-1 block diagram

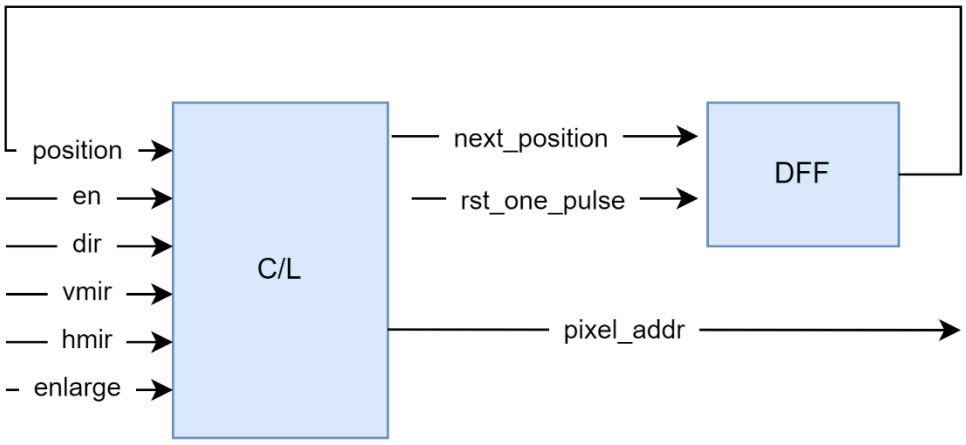

圖 2。mem_addr_gen block diagram

圖 1 是 lab6-1 的 block diagram，本 Lab 的重點在 mem_addr_gen 要輸出 data 的哪一個 pixel。
參考圖 2，最主要的處理方式是用 position 紀錄現在在哪一個 pixel :
position 會依據 dir 的值而遞增或遞減。

en 控制 position 是否需要停止遞增遞減。
Vmir 控制輸出的圖片是否需要垂直鏡像。
Hmir 控制輸出的圖片是否需要水平鏡像。
enlarge 控制輸出的圖片是否需要放大。
處理好的 pixel_addr 輸入給 blk_mem_gen_0，根據這個 address，從 data 中拿出 pixel 並輸出給 vga output。

Lab6-2

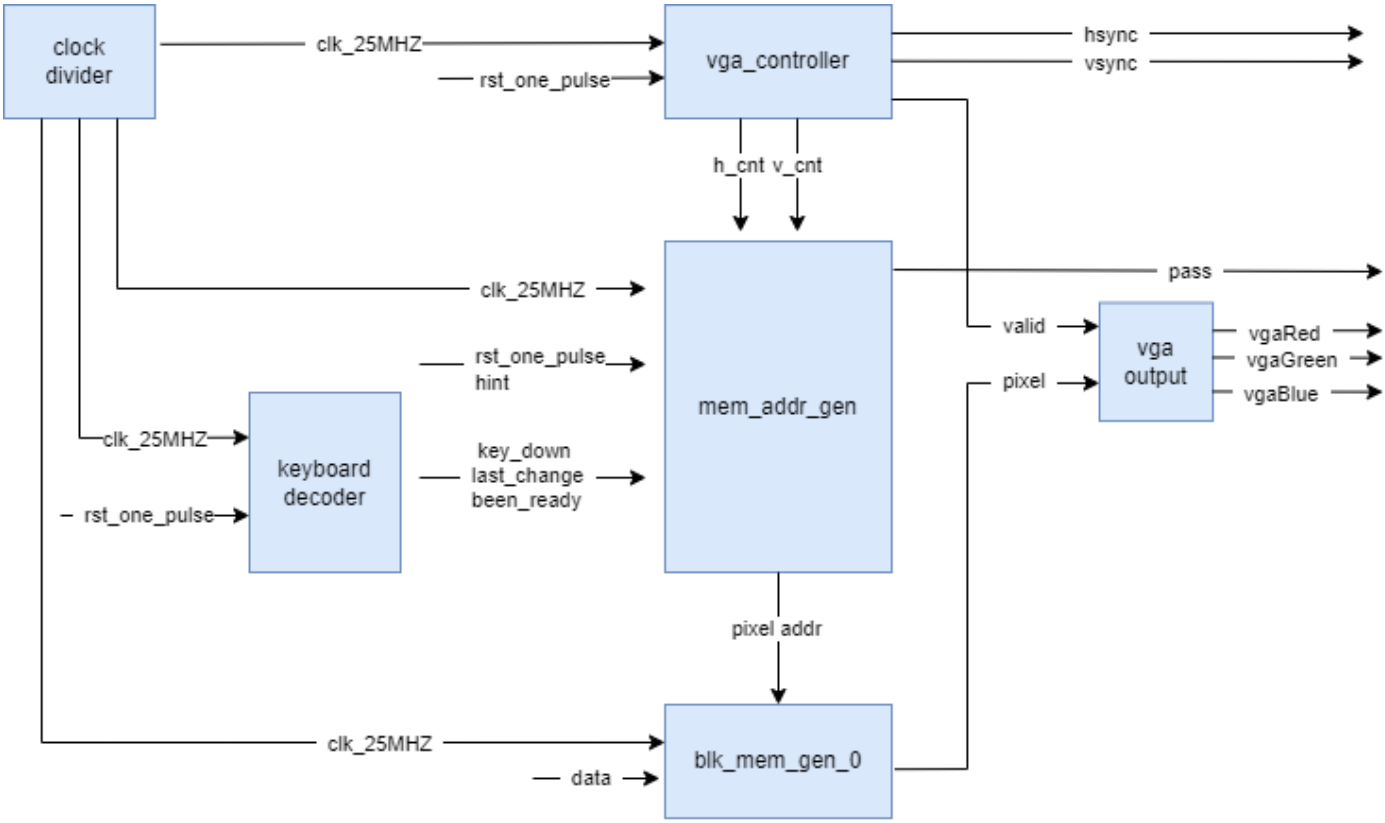

圖 3。6-2 block diagram

圖 3 是 lab6-2 的 block diagram，鍵盤信號 key_down、last_change、been_ready 會作為信號源輸入給 mem_addr_gen，mem_addr_gen 會判斷現在要輸出的圖片狀態，把對應的 pixel_addr 傳給 blk_mem_gen_0，並將 pixel 傳到 vga output。這個過程之中，如果拼圖完成，pass 的值會被拉起為 1，亮起 led 燈，並阻止後續除了 rst 以外的任何輸入。

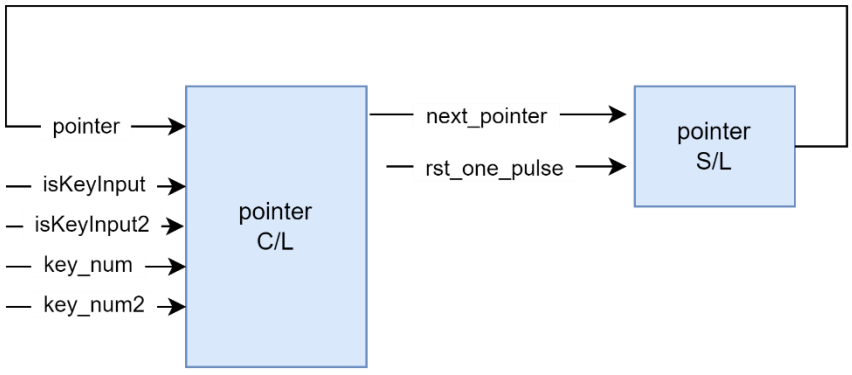

圖 4。6-2，各個區塊需要顯示的區塊

圖 4 是各個區塊需要顯示的區塊的 block diagram。
為了處理兩塊拼圖互相交換位置的問題，我的設計方法是將大圖片切成 16 個區塊，每一個區塊都會有個 pointer 去指向"需要顯示的區塊"。
交換的條件是判斷鍵盤有沒有輸入兩個需要交換的區塊(isKeyInput 和 isKeyInput2)，key_num 和 key_num2 就是需要交塊的區塊的數字。

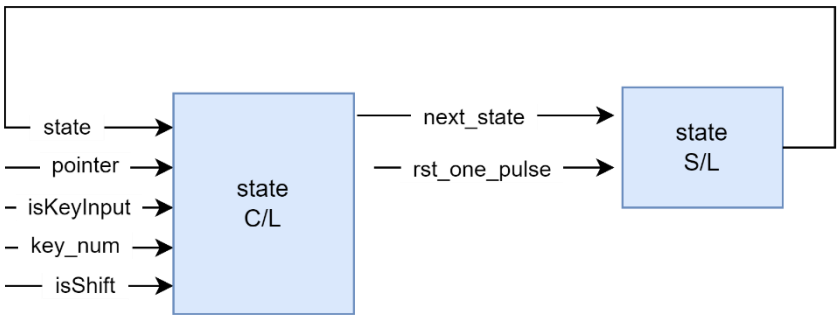

圖 5。6-2，16 個區塊圖片的狀態

圖 5 是 lab6-2 的 16 塊圖片的狀態(state)的 block diagram。
圖片只有兩個狀態: (1)正常擺放 (2)垂直鏡像
變換狀態的條件是現在壓著 shift (isShift) 和 壓著一個區塊 (isKeyInput)。若有，則改變 key_num 代表的區塊的 state。

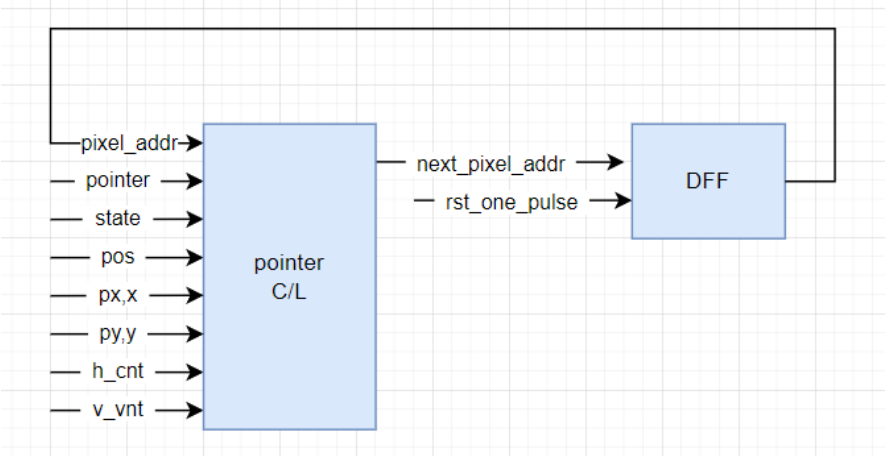

圖 6。6-2，計算 pixel_addr

圖 6 是 lab6-2 的 pixel_addr 計算方法的 block diagram。
pos 代表現在在哪一個區塊，pixel_addr 要輸出的是"pos 的 pointer 指向的區塊的狀態"。
px,x 用於計算 pos 和"pointer 指向的區塊"的 x 座標距離。
py,y 用於計算 pos 和"pointer 指向的區塊"的 y 座標距離。

2. Partial code screenshot with the explanation

Lab6-1

```
always@(*) begin
    if(!enlarge) begin
        if(hmir && vmir) pixel_addr = ((320-((h_cnt>>1)+position)%320) + 320*(240-(v_cnt>>1)))% 76800;
        else if(hmir) pixel_addr = ((320-((h_cnt>>1) + position)%320) + 320*(v_cnt>>1))% 76800;
        else if(vmir) pixel_addr = (((h_cnt>>1) + position)%320 + 320*(240-(v_cnt>>1)))% 76800;
        else pixel_addr = (((h_cnt>>1) + position)%320 + 320*(v_cnt>>1))% 76800;  //640*480 --> 320*240
    end
    else begin
        if(hmir && vmir) pixel_addr = ((320-((h_cnt>>2)+80+position)%320) + 320*(240-(((v_cnt>>1)+120)>>1)))% 76800;
        else if(hmir) pixel_addr = ((320-((h_cnt>>2)+80+position)%320) + 320*(((v_cnt>>1)+120)>>1))% 76800;
        else if(vmir) pixel_addr = (((h_cnt>>2) + position)%320 + 320*(240-(((v_cnt>>1)+120)>>1)))% 76800;
        else pixel_addr = ( ((h_cnt>>2)+80+position) %320 + 320*( ((v_cnt>>1)+120)>>1 ))% 76800;

    end
end
```

圖 7。6-1 的 pixel_addr 計算方式

黃色部分是圖片沒有放大的情況:

水平鏡像(hmir)的情況下，y 座標不動，需要動的是 x 座標。h_cnt 做完 shift 後會在 0~320 之間，假設此時位於(x,y)，就需要顯示(320-x,y)的 pixel。

垂直鏡像(vmir)的情況下，x 座標不動，需要動的是 y 座標。v_cnt 做完 shift 後會在 0~240 之間，假設此時位於(x,y)，就需要顯示(x,240-y)的 pixel。

紅色部分是圖片放大的情況:

先看最後一行 else，為了視覺變大的效果，我把 h_cnt 變得更小，螢幕上的水平 4 個點都用同一個 pixel 表示。但此時圖片不會從中心放大，所以需要把圖片水平位移一段距離，我在 x 座標加了 80，這個數字剛好是圖片長的 1/4，剛好可以達到圖片從中心放大的效果。

Lab6-2

```
// next_state
always @* begin
    for(i = 0; i < 16; i = i+1) begin
        next_state[i] = state[i];
    end
    if(!lock && !pass && !hint && been_ready && key_down[last_change]) begin
        if(isShift && isKeyInput) begin
            next_state[pointer[key_num]] = state[pointer[key_num]]-1'd1;
        end
    end
end
```

圖 8。圖片的 state 變化

因為只有正常擺放和翻轉兩種狀態，這邊的處理是簡單的 -1'b1，state 會在 0,1 不斷循環。
值得注意的是 state 的 index 放的是 pointer[key_num]。
假設 Q 區塊已經和 W 區塊換過圖片了，現在在 Q 區塊的 pointer 指向 W。如果想要翻轉 Q，實際上需要翻轉 W。

```
// next_pointer, swap pictire
always@(*) begin
    for(i = 0; i < 16; i = i+1) begin
        next_pointer[i] = pointer[i];
    end
    if(!lock && !pass && !hint && been_ready && key_down[last_change]) begin
        if(isKeyInput && isKeyInput_2) begin
            next_pointer[key_num] = pointer[key_num_2];
            next_pointer[key_num_2] = pointer[key_num];
        end
    end
end
```

圖 9。圖片交換

因為各區塊有自己的 pointer，在圖片交換時，是把各自的 pointer 指向對方。
例如: Q 和 W 互換，Q 的 Pointer 指向 W，W 的 Pointer 指向 Q。

```
// next_pixel_addr
always @* begin
    next_pixel_addr = 17'd0;
    if(hint) begin
        next_pixel_addr = (h_cnt>>1)+320*(v_cnt>>1);
    end else begin
        case(state[pointer[pos]])

            // 正常擺放
            1'd0: next_pixel_addr = ((160*(px-x)+h_cnt)>>1) + 320*(((120*(py-y)+v_cnt)>>1));

            // 垂直鏡像
            1'd1: next_pixel_addr = ((160*(px-x)+h_cnt)>>1) + 320*((U+(U+10'd120-(120*(py-y)+v_cnt)))>>1);

        endcase
    end
end
```

圖 10。pixel_addr 計算方法

注意條件式 case 裡面放的是 state[pointer[pos]]，也就是當前 pos 區塊 pointer 指向區塊的 state。
例如: 現在的 hcnt、vcnt 在 W 區塊，假設 W 的 pointer 指向 D。pos=W、pointer[W]=D ，
state[pointer[pos]] = state[D]。
也就是我們需要利用 W 區塊的 h_cnt、v_cnt 算出 D 區塊的圖片 address

每個區塊之間都有固定的位移量，詳細可以參考 problem encountered 的圖 11。
在 h_cnt 補上相差的 x 值 (px-x) 和 在 v_cnt 補上相差的 y 值 (py-y)，如此就能實現在 A 區塊顯示 B 區塊圖片的效果。

翻轉的部分與 6-1 非常相似。
這邊為每一個區塊設計了上邊界 (U)，利用 U 作為計算的 base line。
(U+120 - y 座標) = 翻轉效果，但這樣還不正確，各個區塊的 U 都不同還需要補上 y 座標位移量，也就是 U，最終的算式是 U+(U+120 - y 座標)。

B. Questions and Discussions

(a) If we want to turn the image with a special effect of the negative film (which means each pixel is complemented in color, like the example shown in below, the word 'POPCAT' is original white, but after complemented, it is shown in black), how would you modify your design of lab6_1?

原本的輸出如下圖:

```
always@(*) begin
    if(!valid) {vgaRed, vgaGreen, vgaBlue} = 12'h0;
    else {vgaRed, vgaGreen, vgaBlue} = pixel;
end
```

需要再 XOR 12'hfff 才行:

```
reg negative_film;
always@(*) begin
    if(!valid) {vgaRed, vgaGreen, vgaBlue} = 12'h0;
    else begin
        if(negative_film) {vgaRed, vgaGreen, vgaBlue} = (12'hfff^pixel);
        else {vgaRed, vgaGreen, vgaBlue} = pixel;
    end

end
```

(b) Suppose we want to create a VGA game with animations, how can you design the frame buffer and let the item on the monitor moves? (hint: you can use two frame buffers)

可以用 2 個 frame buffer，設計一個雙緩衝的 design。
核心概念是背後運行的那一個 buffer 負責 update，當 update 完時換它顯示在螢幕上，輪到另一個 buffer 做 update。
假設 buffer1 的圖片正顯示在螢幕上，我們可以在 buffer2 做動畫，當 buffer2 做完動畫、或 clk trigger 時，兩個 buffer switch，做好動畫的 buffer2 顯示在螢幕上，換 buffer1 做動畫，然後重複循環。

(c) Our FPGA equips with the BRAM of only 1800 Kbits, which a 640×480 image cannot fit in. If we want to implement a video game, apart from storing a smaller image (e.g., 320×240) like we did in this lab, please give at least 2 possible methods to reduce the BRAM usage

Method 1.
如果是一大塊同顏色的圖片，例如大片的藍色天空，我們可以只在 BRAM 存一小塊藍色圖片。
顯示到螢幕上時，看似一整片的藍色天空，實際上是複製一大堆同樣的藍色小圖片而成。

Method2.
減少像素位數，vgaRed, vgaGreen,vgaBlue 各自最高有 4 個像素數，4 個像素數裡用 1~2 個灰階也可以降低 BRAM usage。

C. Problem Encountered

在圖片交換上遇到非常大的困難，尤其是如何利用當前的 h_cnt 和 v_cnt 算出要顯示區塊的 h_cnt、v_cnt。最後的解法參考下圖 11。
號碼 0~15 的 16 個區塊，各自擁有的(x,y)座標如同右圖。假設區塊 5 和區塊 11 交換，它們之間的(x,y)差值就是(Δx, Δy)=(3-1,2-1)=(2,1)
一格 x 的實際長度是 640/4=160；一格 y 的實際長度是 480/4=120
如此，最後計算 address 的時候在 h_cnt 加上 160*Δx，v_cnt 加上 120*Δy 即可。

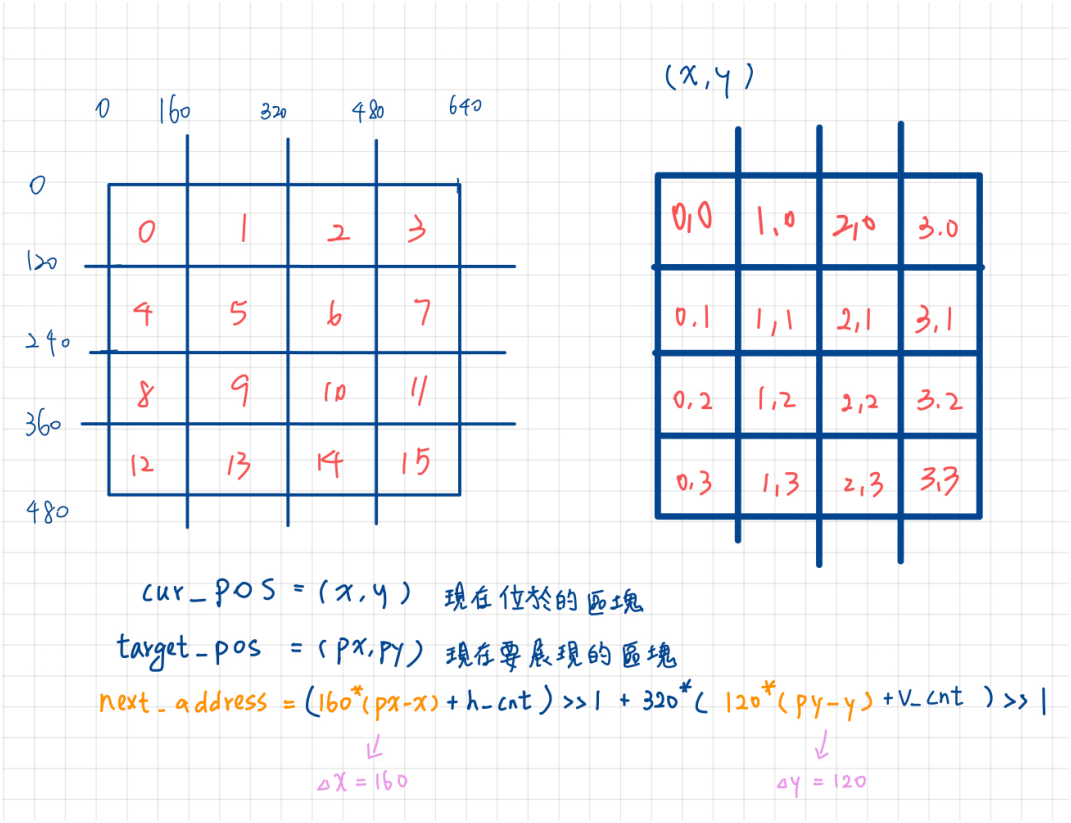

圖 11。區塊座標轉換

D. Suggestions

7-2 滿難的，需要交換圖片和翻轉，思考量很大很大。
還有這次 Question and Discussion 的 B、C 覺得特別難，非常希望在上課時或是 lab report 繳交截止後在課堂上講一下該怎麼做。假如我在 report 內都寫不出答案來，或是寫出來但寫錯，此時就很需要老師的講解拯救我。

A meme of version control.