Final project--小蜜蜂遊戲機

105061110 周柏宇 105061245 蕭郁澄

Design specification

Input:

clk,rst // clock and rst.

switch0, swith1 // two switch to select the level.

inout PS2_DATA , PS2_CLK; // control the vga.

Output:

[3:0] vgaRed, [3:0] vgaGreen, [3:0] vgaBlue

hsync,vsync,

[3:0] four,[7:0] ssd, // to control the SSD.

[15:0] led // 16LEDs to show the life.

I/O pin Assignment

(100M Hz clock) W5 [clk]

(Button) U18[rst]

(switch) V17[switch0] V16[switch1]

(keyboard port) C17 [PS2_CLK] B17 [PS2_DATA]

(seven segment display)

V7 [out[0]]	V5 [out[1]]	V5 [out[2]]	U5 [out[3]]
V8 [out[4]]	U8 [out[5]]	W6 [out[6]]	W7 [out[7]]
W4 [display[3]]	V4 [display[2]]	U4 [display[1]]	U2 [display[0]]

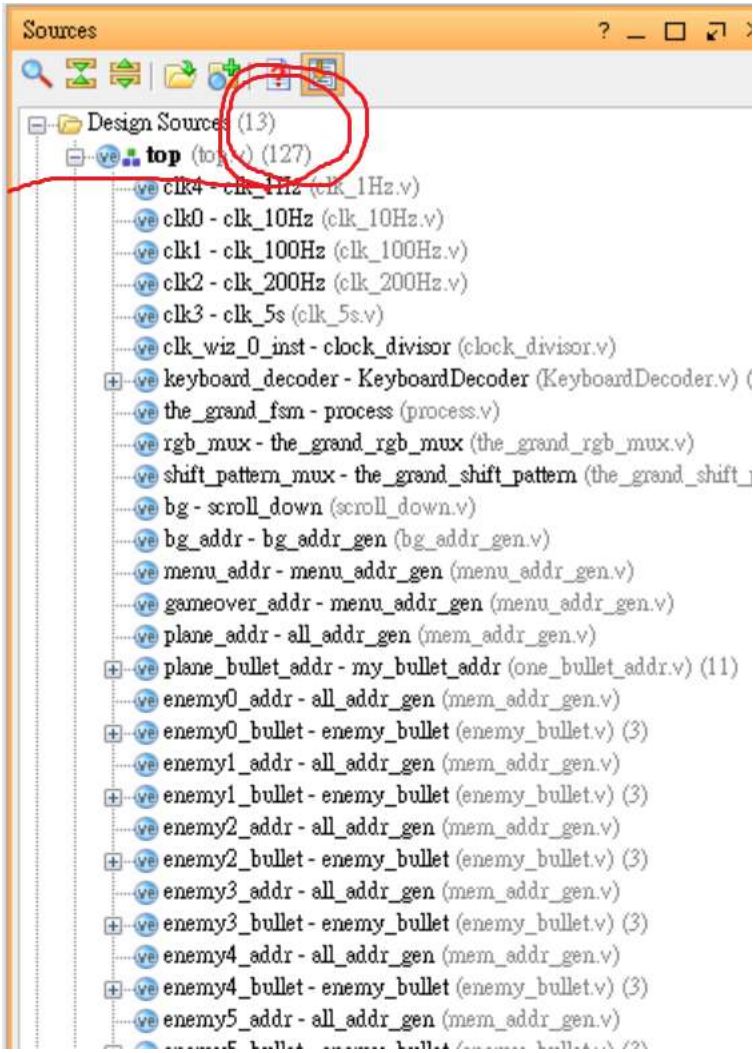
(LEDs)

L1 [led[0]]	P1 [led[1]]	N3 [led[2]]	P3 [led[3]]
U3 [led[4]]	W3 [led[5]]	V3 [led[6]]	V13 [led[7]]

V14 [led[8]] U14 [led[9]] U15 [led[10]] W18 [led[11]]
V19 [led[12]]U19 [led[13]] E19 [led[14]] U16 [led[15]]

Problems and solutions

因為我們一共有 127 個 module，不適合畫 block diagram，所以我們會挑重點功能說明。



|從 16 進位改成 2 進位|

在 lab9 當中，我們所使用的是長寬 16 進位的三位數 ip。但是這樣子所佔的記憶體會太多，於是我們決定將 16 進位改成 8 進位，想不到 verilog 並不接受 8 進位的 coe 檔，於是最終我們使用 2 進位的 9 位數字來表現 RGB，但老師所給的轉檔程式並無法改成 2 進位的 coe 檔，且網路上也找不到好的轉 coe 檔的程式，於是周柏宇同學便興起運用自己課餘時間研究的 Python 來寫轉檔程式，雖然過程花了不少時間，但最後的成果十分顯著，雖然經過降進位制的處理後，圖片的顏色可能會比較不豐富，但我們的 project 比起本來 16 進位制，更能有效率的使用記憶體空間，也較能支援我們多 ip 的寫法。

|9bits 變成 12ibts|

因為最終螢幕所接受的 RGB 是三個 4 個 bits，總共 12bits 的數，]。所以最後需要將 9bits 擴充成 12bits。

For example:

```
module nine_to_twe(pixel,new_pixel);
    input [8:0] pixel;
    output [11:0] new_pixel;
    assign new_pixel[3:0]= (pixel[2:0]==3'b111) ? 4'b1111 : 2*pixel[2:0];
    assign new_pixel[7:4]= (pixel[5:3]==3'b111) ? 4'b1111 : 2*pixel[5:3];
    assign new_pixel[11:8]=(pixel[8:6]==3'b111) ? 4'b1111 : 2*pixel[8:6];
endmodule
```

|圖片製作|

一開始我們使用小畫家來進行圖片製作，但是小畫家在轉存成.jpg 檔時，因我們飛機是 16*16(pixels^2)，所以他在存檔時會把鄰近的像素做混色，而且是類似變暗加強的方式，所以顏色出來的會變得較深，而也會導致我們本來設計的 default_color 被調整掉，於是最後放在螢幕上的結果就會是有一些雜亂的屑屑。而且常常重新改了圖片再次轉存後，顏色仍然會有偏差，所以後來乾脆就改用 Photoshop 來製作圖片。話雖如此，Photoshop 的優點主要是顏色在轉存.jpg 的時候，較不容易互相混色和變暗加強，但是有一個新的問題即是"透明度"。不同透明度會造成也一定機率被底色給混到，而造成最後的.coe 檔中的數值和我們本來所想要的數值不同，最後只好直接修改.coe 檔，將所有有偏差的顏色改為 default_color，再多次燒錄到 FPGA 板子上，投影到螢幕上才能確定自己調整的結果是否令人滿意。

|圖片去背|

我們在製作圖片時，發現如果使用 lab9 抓俄羅斯方塊的方形部分的方法的話，並不容易精準的抓到想要的圖片部分。因此我們把圖片上的底圖部分塗上一個 default color(9'b 101011010)，最終再把多個 ip 輸出的 rgb 值做比較，如果所有的"圖層"都是 defult color 的話，則顯示 background 的 rgb。

|模組化 module|

本來我們產生一個物件的 address 是使用 men_addr_gen 這個 module 所輸出的 pixel_addr，如果今天我要多增加一個物件的 pixel_addr 輸出，則這個 module 就必須多再增加一個新的 output variable.而考量到我們有自己的飛機、很多敵機和更多的子彈……，所以我們決定將 men_addr_gen 變成一個 general 的寫法，當我需要多增加一個物件的 pixel_addr 時，我們可以直接呼叫這個 general 的 module，只需要在外部輸入的時候改變輸入的長寬、初始位置……就能簡單且清楚的增加一個新的 pixel_addr。

For example:

```
    module all_addr_gen(
   input clk,
   input rst,
```

```
    input [9:0] h_cnt,
    input [9:0] v_cnt,
    input [9:0] length,
    input [9:0] height,
    input [9:0] h_offset,
    input [9:0] v_offset,
    output reg [12:0] pixel_addr
    );

    wire [9:0] h;
    wire [9:0] v;
    assign h = h_cnt%640;
    assign v = v_cnt%480;

    always @(*) begin
      if (h>=h_offset && h<(h_offset + length*2) &&
           v>=v_offset && v<(v_offset + height*2))
      begin
          pixel_addr = (((h - h_offset)>>1) + length*((v - v_offset)>>1));
      end
      else begin
         pixel_addr = length*height - 1;
      end
    end
endmodule
```

|飛機上下左右移動|

一開始使用 last_change 來當作判斷的條件，後來發現這樣子會導致只能判斷同一個瞬間按下的某一個鍵，而如果我想要做按下右和上的時候，自己的飛機往右上移動，則會無法實現。所以後來決定改成使用 key_down 來判斷複合按鍵，將上下左右 4 個按鍵的是否按下做組合搭配，在使用 if else if 來判斷該往哪個方向移動。

|我方飛機子彈射擊|

最初的想法是利用 key_valid 會跟按鍵按下去的瞬間同步的特性，但是如此一來在畫面上同時只能有一發子彈。若是複數子彈的狀況下，會有子彈一起發射，或者是子彈無法發射的狀況…..，經過我把訊號外接成 LED 燈來觀察後，我發現原來會有兩顆子彈同時被擊發的狀況，是因為判斷空白鍵是否被觸碰的 clock 很快，所以當我以為我自己只有按下”一下”空白鍵，其實在判斷空白鍵是否按下的 module 裡已經判斷無數次了。後來決定直接把 key_down[空白鍵]這個訊號接成 one_pulse，如此一來，只要再某個瞬間空白鍵被按下，就會有一顆子彈被射出。但是又有新的問題產生了，如果我想要連射該怎麼辦?

只好再透過 counter 來計數，當空白鍵被持續按下一段時間後(這個時間可

以由我們自由控制、調整，而這個時間也就影響到子彈連射的速度)，也會讓觸發子彈發射的變數等於 1，再透過判斷子彈是否到達邊界來判斷現在要發射哪顆子彈，就能實現可以連射子彈和單發射擊的功能了。

|敵方飛機子彈射擊|

相較於我方子彈射擊需要空白鍵來觸發射擊，敵人的子彈較為單純。只要將本來在判斷式中的接收空白鍵被按下的變數，改成一個適當頻率的 clock，敵機的子彈就會有個固定的射擊週期。此外，還有一個和自己飛機子彈的不同之處，敵軍子彈可以同時擁有 x 向量和 y 向量(也就是可以設斜的)，而自己的子彈則是直來直往的。(我們在 module 中所接的是週期 5s 的 clock。)

|碰撞—layer compare|

由於我們將每個"物件"的 rgb 由不同的 ip 輸出，所以我們可以將所有物件的 rgb 輸入到一個 compare 的 module 中做 priority 的比較。例如當同一個位置上，同時有自己飛機和子彈重疊的話，則顯示飛機的 rgb。並且如果某個點上的所有圖層的 rgb 都是 default_color 的話，則會顯示 background 的 rgb。

因此我們利用這個方式，把每個敵機的子彈還有敵機本身和自己的飛機做判斷，如果同時在某個點上都不是 default_color 的話，就表示發生碰撞!!

For example:

```
module layer_compare(
    input [8:0] rgb_plane,
    input [8:0] rgb_bullet_0,
    input [8:0] rgb_bullet_1,
    input [8:0] rgb_bullet_2,
    input [8:0] rgb_bg,
    input [8:0] rgb_enemy0,
    input [8:0] rgb_enemy1,
    input [8:0] rgb_enemy2,
    input [8:0] rgb_enemy0_bullet_0,
    input [8:0] rgb_enemy0_bullet_1,
    input [8:0] rgb_enemy0_bullet_2,
    input [8:0] rgb_enemy0_bullet_3,
    input [8:0] rgb_enemy0_bullet_4,
    input [8:0] default_color,
    output reg [8:0] pixel_out
    );

always @(*) begin
    if (rgb_enemy0!=default_color)
        pixel_out = rgb_enemy0;
    else if (rgb_enemy1!=default_color)
        pixel_out = rgb_enemy1;
    else if (rgb_enemy2!=default_color)
        pixel_out = rgb_enemy2;
    else if (rgb_plane!=default_color)
```

```
            pixel_out = rgb_plane;
        else if (rgb_bullet_0!=default_color)
            pixel_out = rgb_bullet_0;
        else if (rgb_bullet_1!=default_color)
            pixel_out = rgb_bullet_1;
        else if (rgb_bullet_2!=default_color)
            pixel_out = rgb_bullet_2;
        else if (rgb_enemy0_bullet_0!=default_color)
            pixel_out = rgb_enemy0_bullet_0;
        else if (rgb_enemy0_bullet_1!=default_color)
            pixel_out = rgb_enemy0_bullet_1;
        elseif (rgb_enemy0_bullet_2!=default_color)
            pixel_out = rgb_enemy0_bullet_2;
        else if (rgb_enemy0_bullet_3!=default_color)
            pixel_out = rgb_enemy0_bullet_3;
        else if (rgb_enemy0_bullet_4!=default_color)
            pixel_out = rgb_enemy0_bullet_4;
        else
            pixel_out = rgb_bg;
end
endmodule
```

|爆炸特效 object_phase|

做這個的動機單純是對真實感的追求，當子彈擊中飛機，應該要有爆炸的火花感，然後才會化為灰燼，消失。為了達到我們的目的，我們首先要有負責判斷碰撞的機制，也就是 module hit_detector，他會比較這個 pixel 是否為飛機和子彈顏色的重疊，如果是，則我要延長這個訊號，因為我想把它變成一個 counter(module explosion_latency)的 enabler，於是我把 hit_detector 出來的訊號(hit_enemy)接上一個 finite state machine(module permanent_en)，作用是一旦被打到，我就要它是"已擊中"(is_hit)。爆炸當然少不了火花，在 module explosion_phase 裡闡述了整台飛機從完整、爆炸到消失的過程，原理也是一個 finite state machine，從完整 state 到爆炸(火花 state)是被 is_hit 觸發，在爆炸 state，我想要產生一個電光一閃的效果，所以我把飛機塗成火花(rgb_explosion)，而且在 0.1 秒內消失。原理就是在進入爆炸 state 時，我開始用 counter 接上 clk_10Hz 數 1 到 10。當數到 10，立刻把爆炸塗成 default(在 module layer_compare 會被忽略)，這麼一來就產生一連串生動的狀態變化，貫徹"魔鬼藏在細節中"的職人精神。

```
module object_phase(
    input clk,
    input rst,
    input clk_10Hz,
    input [8:0] rgb_bullet_0,
    input [8:0] rgb_bullet_1,
    input [8:0] rgb_bullet_2,
    input [8:0] rgb_enemy,
    input [8:0] rgb_explosion,
    output [8:0] rgb_out,
```

```
        output is_hit
        );

wire hit_enemy;
wire last_phase_en;

hit_detector hit_detector(
        .rgb_bullet_0(rgb_bullet_0),
        .rgb_bullet_1(rgb_bullet_1),
        .rgb_bullet_2(rgb_bullet_2),
        .rgb_enemy(rgb_enemy),
        .default_color(9'b101011010),
        .hit_enemy(hit_enemy)
        );

permanent_en fsm_is_hit(
        .clk(clk),
        .rst(rst),
        .hit_plane(hit_enemy),
        .is_hit(is_hit)
        );

explosion_latency cnt(
        .clk(clk_10Hz),
        .rst(rst),
        .en(is_hit),
        .last_phase_en(last_phase_en)
        );

explosion_phase explosion_phase(
        .clk(clk),
        .rst(rst),
        .rgb_enemy(rgb_enemy),
        .rgb_explosion(rgb_explosion),
        .rgb_default(9'b101011010),
        .is_hit(is_hit),
        .last_phase_en(last_phase_en),
        .rgb_out(rgb_out)
        );
endmodule
```

心得:

當初跟 partner 決定做小蜜蜂當 final project 其實是一個機緣。正當我們被題目苦惱著，我就偷偷耍廢，打開瀏覽器，在搜尋欄打上”史萊姆的第一個家”，裡面形形色色的 flash 小遊戲就是我小時候的顏料，在我的記憶揮灑斑斕，其

中一個遊戲，在我腦海裡是最清晰，最簡潔，卻又是最刺激，就是小蜜蜂這遊戲。憑著一股傻勁，沒有頭緒，沒有 plan B，我們倆就踏上這崎嶇的開發之路。

一路上分工相當清晰，周柏宇負責 vga 顯示的部分，像是怎麼抓圖的 address，如何在有限的 RAM 裡最大化一個遊戲的視覺體驗，還有各式的疊圖、塗圖技巧。這一路艱辛，時常在決策上失誤，像是我們在製造爆炸特效 7 時，一度試圖使用 write_enable 的方式改變圖片，但經過一個下午無數次的合成，我最終是放棄此法，任憑光陰逝去。如果有一個女神掌管 Verilog，那我必受了她的祝福，在幾次山窮水盡，總是有一個靈光乍現的想法突破困境，像是我很自豪的爆炸特效和擊中後的無敵時間，都是 Verilog 女神凌晨在資電四樓指引，這幾筆畫龍點睛使我們的 project 鶴立雞群。而蕭郁澄則是不厭其煩的把 keyboard 那次的 lab 做優化，在這個過程，他發現 key_valid 這個訊號跟 ptt 描述的不太一致，儘管如此他也只是冷靜沉著的改變寫法，反而成效更佳，這是很重要的一點，因為我們的遊戲體驗完全仰賴鍵盤，而蕭郁澄提供了一個絕佳的鍵盤模組，使整個發開過程始終抱持流暢，遊戲也不用擔心卡頓。

一路上我們合作無間，沒有怨言，只有無怨無悔的付出和勉勵。我們時常約早上 7 點打 final，雙雙犧牲普物上課時間和睡眠時間，在資電 4 樓一待就是 8 小時(含吃午餐)。期間我們不斷的溝通、協調，也正因為如此，在合併雙方程式碼的過程沒有什麼阻力，在 demo 前的 1 個半小時，project 的 final version 問世。

雖然成品跟記憶中的小蜜蜂有落差(大多是設備的限制)，但成品的每個 pixel 每個 rgb 都是我們辛辛苦苦雕刻出來的。在成品誕生地當下，五味雜陳，有的是如釋重負的欣慰，有的是合作無間的肯定，更清晰的是，肝臟的微微抽痛。