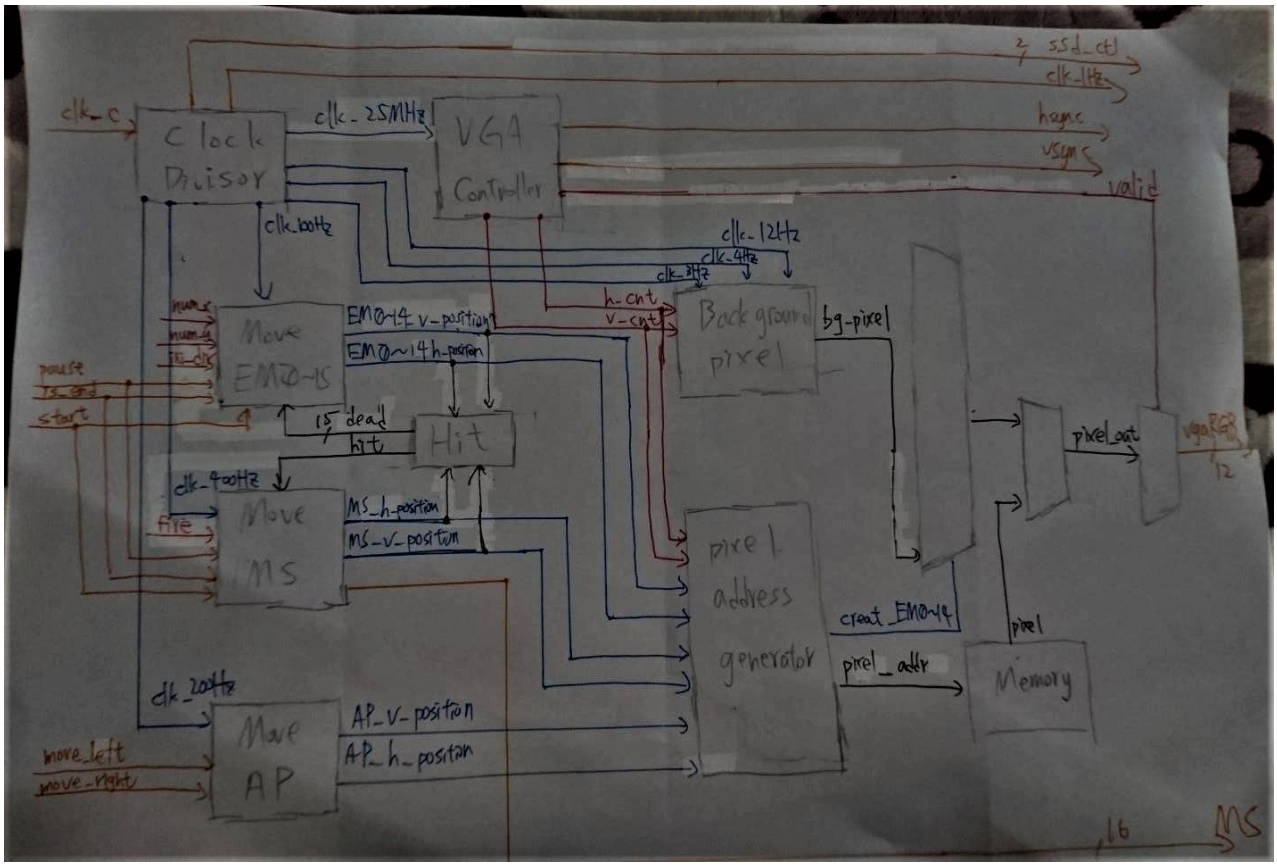
105060006 洪嘉廷
105060016 謝承儒

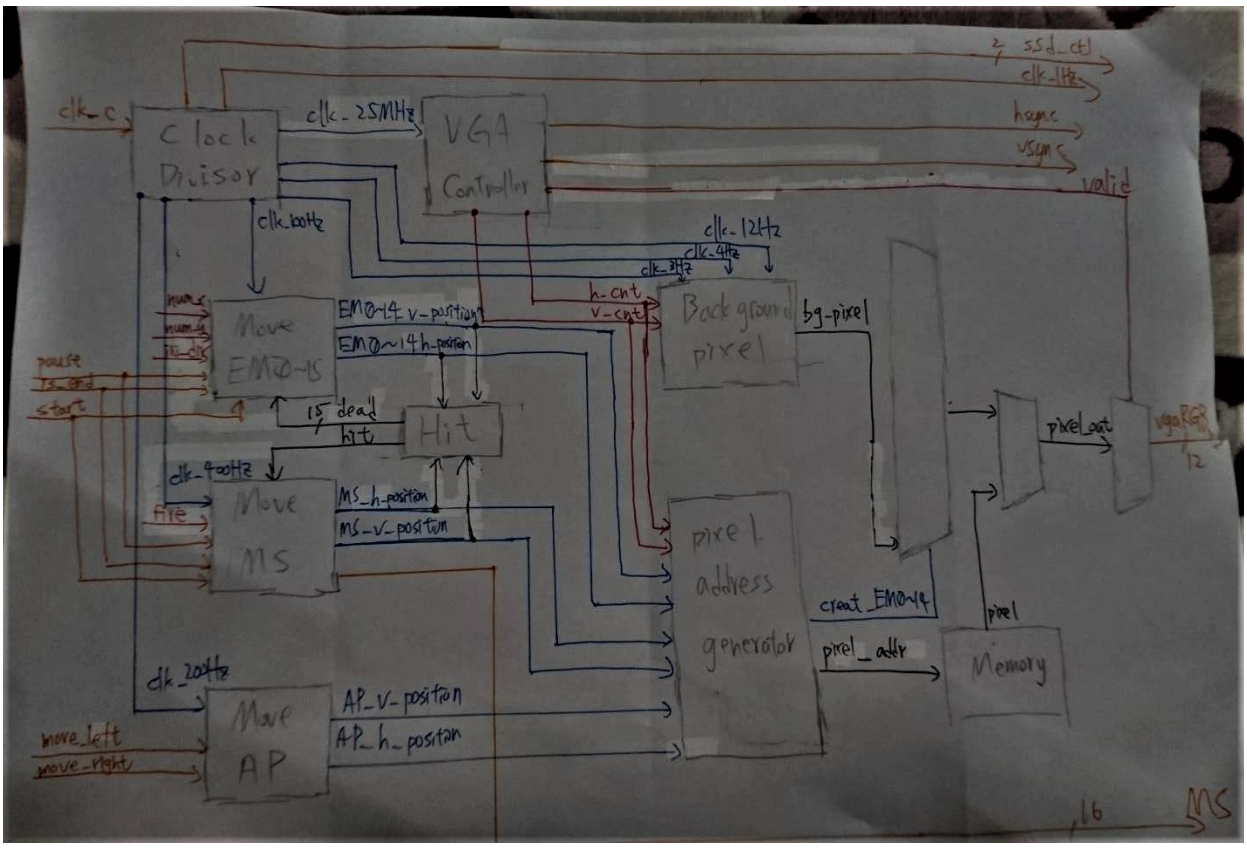
Final Project ---- 小蜜蜂遊戲

A. Design Specification

input unlimited (開啟的話，子彈變成無限制：子彈射完遊戲也不會結束)
input rst (重置所有功能)
input clk_c (100MHz 的 clock 訊號)
output [3:0] vgaRed, vgaGreen, vgaBlue (VGA 3 原色的明暗 0:暗→15:亮)
output [3:0] display_c (控制哪個七段顯示器)
output [7:0] display (控制對應的七段顯示器的 8 個燈)
output [15:0] MS (接到 16bit LED 燈，代表飛彈的數目)
output hsync
output vsync
inout PS2_DATA (鍵盤輸入的資料)
inout PS2_CLK (鍵盤輸入的 clock)

Block diagram:





B. Design Implementation

I/O assignment:

I/O	clk_c	rst	display_c[3]	display_c[2]	display_c[1]	display_c[0]
Port	W5	V17	W4	V4	U4	U2

I/O	display[7]	display[6]	display[5]	display[4]	display[3]	display[2]
Port	W7	W6	U8	V8	U5	V5

I/O	display[1]	display[0]	MS[15]	MS[14]	MS[13]	MS[12]
Port	U7	V7	L1	P1	N3	P3

I/O	MS[11]	MS[10]	MS[9]	MS[8]	MS[7]	MS[6]
Port	U3	W3	V3	V13	V14	U14

I/O	MS[5]	MS[4]	MS[3]	MS[2]	MS[1]	MS[0]
Port	U15	W18	V19	U19	E19	U16

I/O	hsync	vsync	PS2_CLK	PS2_DATA	unlimited
Port	P19	R19	C17	B17	R2

I/O	vgaGreen[3]	vgaGreen[2]	vgaGreen[1]	vgaGreen[0]
Port	D17	G17	H17	J17

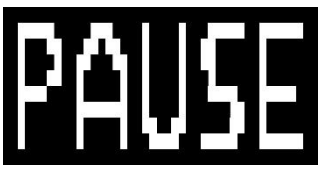
I/O	vgaBlue[3]	vgaBlue[2]	vgaBlue[1]	vgaBlue[0]
Port	J18	K18	L18	N18

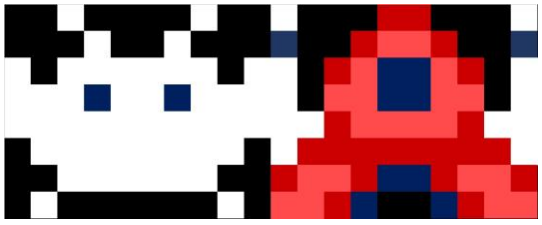
I/O	vgaRed [3]	vgaRed [2]	vgaRed [1]	vgaRed [0]
Port	N19	J19	H19	G19

C. Discussion

a. 整體運行過程

1. 將 clk_c 輸入到 clk_divisor 除頻成各種頻率的 clk，用來驅動不同的 block。
2. 利用 key_down 來判斷按下甚麼鍵，用右邊的 4、8 來當作飛機移動的方向，並輸出飛機所在的位置(AP_v_position、AP_h_position)。
3. 把飛機的位置輸入到 Move_MS，如此就能讓子彈的位置一直跟著飛機，若是按下空白鍵，則將子彈射出。要輸出子彈的位置(MS_v_position、MS_h_position)和子彈的數量(MS)。
4. 接著，Move_EM0~14 輸出敵人的位置(EM0~14_v_position、EM0~14_h_position)，加上子彈的位置判斷是否有擊中。
5. 把飛機、子彈和敵人的位置輸入到 pixel_address_generator，就能判斷出何處要顯示出甚麼物件，輸出 pixel_addr 和 creat_EM0~14。
6. pixel_addr 輸入到 blk_mem_gen_material，即可取得相應的混色比例(pixel)。
7. 利用 creat_EM0~14 和 pixel 可以來自行決定每隻敵人的顏色，輸出為 pixel_out。
8. 除了物件之外的其他 pixel_out 就選擇使用從 generate_background 輸出的 bg_pixel。
9. 最後將 pixel_out 輸出成 vgaRed、vgaGreen、vgaBlue 三部分，螢幕部分的就結束了。
10. 而 FPGA 板則是用 clk_1Hz 驅動 Counter 並輸出 s0~s2，代表經 過時間，經過 SSD_Decoder 和 SSD_CTL 就能顯示在七段顯示器上。此外，把子彈數數顯示在 16 個 LED 燈上，一發代表一個 LED。





b. 各 block 的構想

1. **move_AP : 飛機移動**

   因為我們的飛機只能水平移動，所以只需要取橫向的座標。我們取 airplane 最左上角當作位置的起始點。將左右方向鍵的 key_down 輸入進來，每當 clk 升緣，判斷左右鍵有沒有被按下，決定該往左/右移動或是不動。移動時把座標與邊界位置比較，假如移動後會超出邊界(airplane 左緣碰到左方邊界或右緣碰到右方邊界)的話就停留在原地。左緣就是直接取 airplane 的座標，右緣則是把座標加上 airplane 的寬度。

2. **move_EM(0~14) : 敵人移動**

   移動方式同 move_AP，但是多了縱向的位移所以需要考慮縱向的座標。當敵人碰到邊緣時，change_dir 訊號會變成 1。此時，左/右移動的訊號變成零，方向的訊號取 complement (反向)，向下移動的訊號變成 1。由於向下移動後的橫向位置依然在邊緣，所以當下一次 clk 過來時 change_dir 會變回 0，此時又能左右移動。左右移動過後，位置就脫離邊緣，change_dir 就會維持在 0。

   輸入死亡的訊號 dead，當敵人死亡後，就把它移到畫面右下方。

   當輸入的 pause 訊號等於 1 時，敵人會待在原地。

   當敵人的下緣碰到 deadline 上緣時，輸出 is_end 訊號，代表遊戲結束，敵人就會停留在原地。

3. **move_MS : 子彈移動**

   如果處於裝填狀態，則就利用輸進來的飛機位置，讓子彈位置一直跟著飛機移動。當按下空白鍵後，就會變成發射狀態，就不用改變水平位置，只需要改變垂直位置。

   若是撞到敵人(hit_valid)或是撞到頂部就重新裝填，也就是回到裝填狀態，並將 MS 往右移 1-bit 且空位補上 0，如此 LED 就會少亮一個，代表子彈數目減少一個。

4. **hit_target : 擊中判定**

   把敵人座標和子彈座標都輸進來，使用和 move_MS 相同的 clk，這樣每當子彈移動一格，就會判定一次。

   擊中判定(hit_valid)就是子彈的左座標(MS_h_position)要小於敵人的右座標(EM_h_position + 敵人寬度)、子彈的右座標(MS_h_position + 子彈寬度)要大於敵人的左座標(EM_h_position)、再加上子彈上緣(MS_v_position)要小於敵人下緣(EM_v_position + 敵人高度)。

5. **gen_background: 產生背景(閃爍的星星)的 pixel**

先用 C 語言亂數產生出 100 組座標(包括縱向+橫向)，這些座標代表著星星的起點位置。用三個 color 來控制星星的閃爍，並以 3 個不同的 counter 來控制。每個 counter 賦予不同的初值，讓閃爍時間能錯開來之外，也能讓星星產生不同顏色的閃爍(1:白→黑→白；2:黃→紅→黃；3:深藍→天藍→深藍)。

由於螢幕長寬比並非 1:1，所以以符合長寬比的頻率的兩個 clk 來分別改變星星橫坐標、縱座標位置。

輸入 h_cnt, v_cnt，當 h_cnt, v_cnt 等於星星橫、縱位置時，把 color 的值輸出去(bg_pixel)，否則的話 bg_pixel 就等於 0(黑色)。

c. 遇到的難點及 bug

1. 敵人有 15 隻，所以每個跟敵人有關的訊號都得有 15 份，再接模組的時候就得傳很多資訊，move_enemy 模組也得接 15 個，很佔空間，一不小心會接錯編號。
2. 圖片轉成 memory 時，除了寬度跟高度得要抓對之外，長寬比例也很重要，尤其我們的圖片是把很多素材排成一列，再根據長寬來分割。所以沒有注意好長寬比的話，分割的時候很常出現問題，例如:破圖或是切到素材的邊緣。
3. 在 move_enemy 裡，敵人撞到牆壁後要先向下移動一格後並往反方向走。一開始寫的時候，用 always@*驅動的部分形成了 combinational loop，後來將某些變數改成 Sequential 的形式，就能正常運作了。
4. 在完成 bit 檔之後，敵人之中，第 14 隻敵人(EM13) 有時候會呈現無敵的狀態，不管怎樣都射不死。有時候改了一些非相關的東西，如：敵人配色，背景顏色等，就突然又恢復正常了。一旦完成 synthesis，有問題的不管重跑幾次 write bitstream 幾次都會有問題；但相對的，沒問題的就還是沒問題。所以推測可能是 synthesis 的時候有產生某些不確定因素。我們找了很久，一直找不到這個 bug 的解決方式，明明 EM13 相關的 code 都和其他 14 隻一樣，卻偏偏只有它會出問題。假如是頭尾的部分出問題，可能是起始或結束的條件沒有考慮清楚。但是我們的 bug 卻是出現在中間的敵人，而且也不是每次都會出現這個 bug，讓我們困擾了很久，卻又不知從何改起。

Conclusion

寫小遊戲感覺是比較偏軟體的事情，用硬體描述語言來寫，常常有一些覺得不方便的地方。