2019 數位實驗第 20 組期末專題

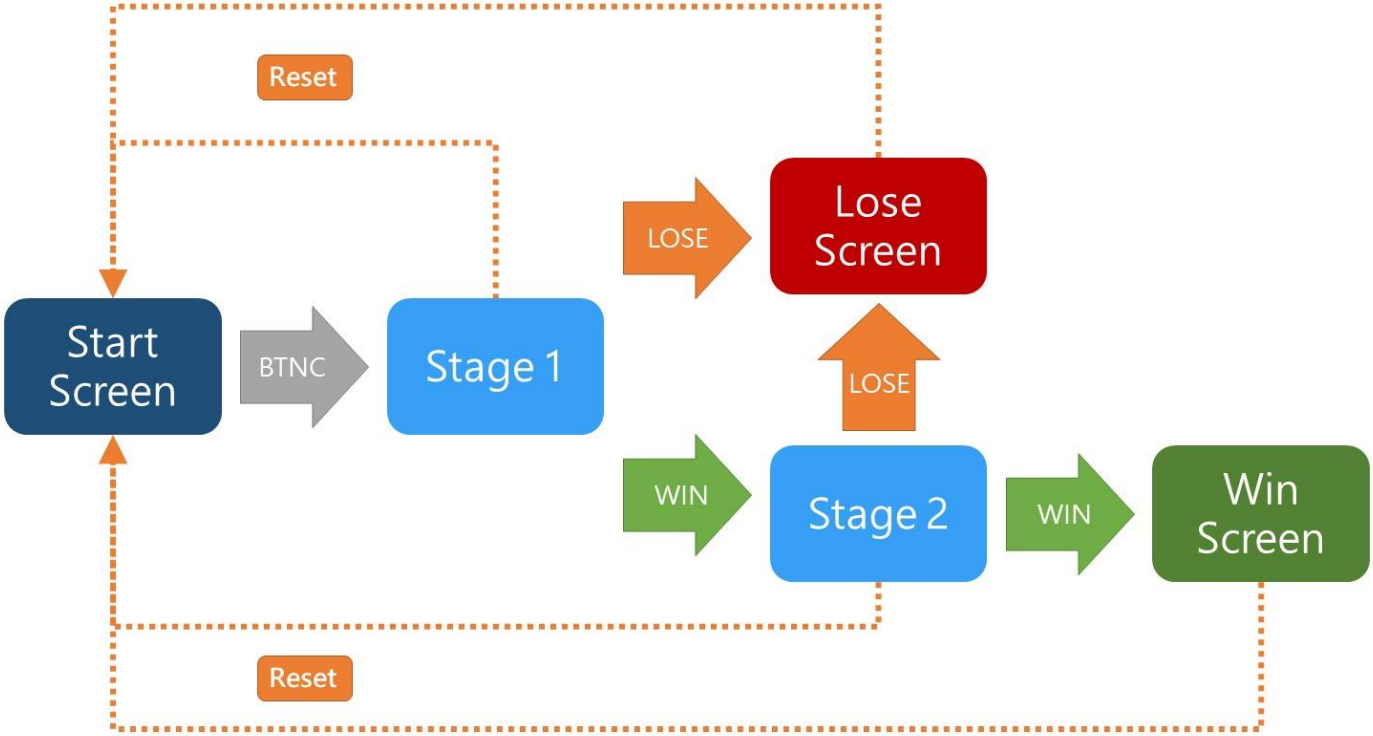
BB 大爆射

0710130 黃鼎翔　　0710145 洪若軒

壹、專題目標：

在此專題中，我們透過控制 FPGA 板上的按鈕來控制戰機的左右移動。透過我方戰機（血量 25）自動發射的子彈來與敵人對戰。

遊戲分成兩關，第一關是由兩隻小怪組成的關卡一，小怪只有橫向移動及連續發射子彈的能力，血量各為 50；第二關是魔王關，魔王有兩種攻擊模式以及隨機移動，血量為 200。被子彈射中一顆扣一滴血。



遊戲流程示意圖

貳、專題原理：

A. VGA 螢幕輸出

螢幕的顯示方式，是由左上角的第一個像素開始，依次向右顯示下一個像素，到顯示完該列的最後一個像素，就跳到下一列的第一個像素開始繼續顯示。直到整個畫面都顯示完畢時，又回到原點來顯示，如此不斷刷新畫面。在不同的像素時，就可藉由 RGB 的訊號輸出來控制顏色。而顯示不同的像素的時機，需由 Horizontal 與 Vertical 的 Sync signal 決定。

在 1024x768@60Hz 的頻率下，sync 訊號所需之 timing 如下：

General timing	
Screen refresh rate	60 Hz
Vertical refresh	48.363095238095 kHz
Pixel freq.	65.0 MHz

Horizontal timing		
Scanline part	Pixels	Time [µs]
Visible area	1024	15.753846153846
Front porch	24	0.36923076923077
Sync pulse	136	2.0923076923077
Back porch	160	2.4615384615385
Whole line	1344	20.676923076923

Vertical timing		
Frame part	Lines	Time [ms]
Visible area	768	15.879876923077
Front porch	3	0.062030769230769
Sync pulse	6	0.12406153846154
Back porch	29	0.59963076923077
Whole frame	806	16.6656

由這些數據與原理，編寫一個 module 負責控制 VGA 訊號的輸出，其主要由這兩部分構成：

```
// VGA timings
localparam HS_STA = 24;                 // horizontal sync start
localparam HS_END = 24 + 136;           // horizontal sync end
localparam HA_STA = 24 + 136 + 160;     // horizontal active pixel start
localparam VS_STA = 768 + 3;            // vertical sync start
localparam VS_END = 768 + 3 + 6;        // vertical sync end
localparam VA_END = 768;                // vertical active pixel end
localparam LINE   = 1344;               // complete line (pixels)
localparam SCREEN = 806;                // complete screen (lines)

reg [10:0] h_count;  // line position
reg [10:0] v_count;  // screen position
```

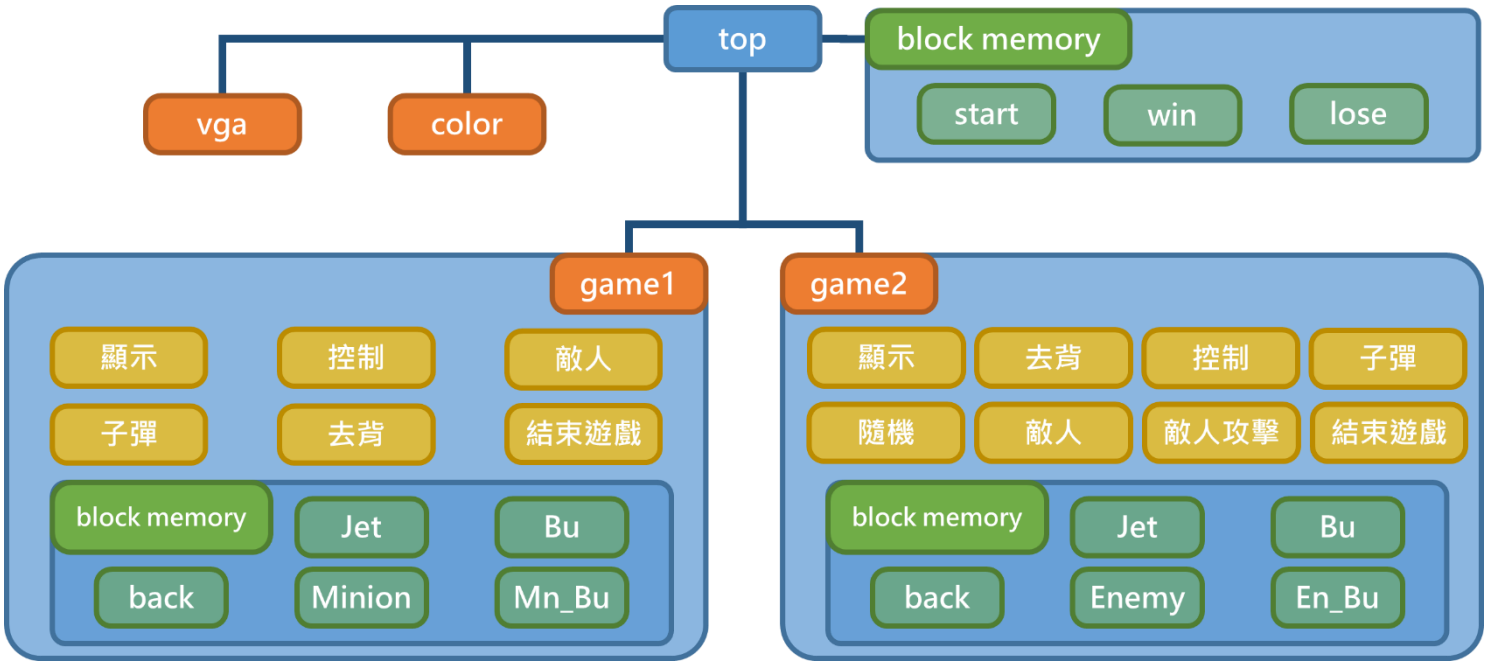
```
always @ (posedge i_clk)
begin
    if (i_rst)  // reset to start of frame
    begin
        h_count <= 0;
        v_count <= 0;
    end
    if(i_pix_stb)  // once per pixel
    begin
        if (h_count == LINE)  // end of line
        begin
            h_count <= 0;
            v_count <= v_count + 1;
        end
        else
            h_count <= h_count + 1;

        if (v_count == SCREEN)  // end of screen
            v_count <= 0;
    end
end
```

B. matlab 與 block memory

為了讓顯示的畫面更加精美，我們將圖片經過 matlab 轉檔，轉換成每一像素由 12bit 的資料表示的 coe 檔。再經由 Vivado 內的 IP Catalog 將 coe 檔寫入 FPGA 的記憶體中。

參、設計流程：

程式碼示意圖：



A. game1 與 game2 中，主要是由以下幾個 block 組成：

1. 顯示

- 背景顯示：

由 block memory 存取顏色資料，輸入 address，即可輸出所需之顏色。由於 FPGA 記憶體有限，而需要儲存的圖檔眾多，因此無法儲存解析度太高的圖檔，只夠儲存 256x192 的圖檔，所以必須將原圖放大 4 倍，才能在畫面上正常顯示。因此先將 Y 與 X 的 address 分別放大 4 倍，再合成為 address。

```
//Background
wire [10:0] picPixel_Y_B,picPixel_X_B;
//Zooming for background from 256x192 to 1024x768
assign picPixel_Y_B =  vgaPixel_Y [10:2];
assign picPixel_X_B =  vgaPixel_X [10:2];
//Address to be passed to the Block RAM
reg [15:0] Addr_back = 16'd0;
//Out from the Block RAM
wire [11:0] back_out;
// Block ROM to store the image information of background
blk_mem_gen_0 back (
  .clka(pCLK2),     // input wire clka
  .addra(Addr_back),  // input wire [17 : 0] addra
  .douta(back_out)  // output wire [11 : 0] douta
);
```

```
assign picPixel_Y_B =  vgaPixel_Y [10:2];
assign picPixel_X_B =  vgaPixel_X [10:2];

Addr_back <= {picPixel_Y_B[7:0], picPixel_X_B[7:0]};
vga_out <= back_out;
```

- 物件顯示：(以戰機為例)

同樣利用 block memory 存取顏色資料，不過 address 設置方式不同。在創建物件時，我們會給它一組 top、left 參考點以記錄物件位置。而由當前座標減去物件參考點，即可推算出在該物件內的相對座標，並合成出顯示所需之 address。

```
//Jet Display
//address out
reg [13:0] Addr_jet = 14'd0;
wire [11:0] Jet_out;
//jet image
blk_mem_gen_1 jet (
    .clka(pCLK2),
    .addra(Addr_jet),   // input [13:0] address
    .douta(Jet_out)     // output [11:0] jet
);
reg [10:0]Jet_top = 610;                          // Jet initial location Y
reg [10:0]Jet_left = 448;                         // Jet initial location X
wire [6:0]Jet_addr_y = vgaPixel_Y - Jet_top;  // Jet Address first 7 digit
wire [6:0]Jet_addr_x = vgaPixel_X - Jet_left; // Jet Address last 7 digit
```

- 血條顯示：

由血條參考點推算出血調顯示範圍，分別令其顏色顯示為白、紅、黑色。

2. 操控、移動

在戰機控制中，我們用 4 個 botton 控制八個方位 (右圖以上下左右為例)。我們將飛機參考點設為左 (Jet_left) 上 (Jet_top) 角，並透過移動參考點來移動整個飛機。If 裡面的條件則是使飛機不要超出邊界。

```
4'b1010 :   // Up Left
    begin
    if(Jet_left > 2)Jet_left <= Jet_left - 1;
    if(Jet_top > 2) Jet_top <= Jet_top - 1;
    end
4'b1001 :   // Up Right
    begin
    if(Jet_left < 896)Jet_left <= Jet_left + 1;
    if(Jet_top > 2) Jet_top <= Jet_top - 1;
    end
4'b0110 :   // Down Left
    begin
    if(Jet_left > 2)Jet_left <= Jet_left - 1;
    if(Jet_top < 633 )Jet_top <= Jet_top + 1;
    end
4'b0101 :   // Down Right
    begin
    if(Jet_left < 896)Jet_left <= Jet_left + 1;
    if(Jet_top < 633 )Jet_top <= Jet_top + 1;
    end
```

3. 敵方移動，攻擊：

關卡一有兩隻小怪，其移動方式相同(右圖取其一的移動方式)，一樣將左上角當作定位點，並控制 x 座標的移動，將怪物控制在螢幕內。而小怪的攻擊模式則與我方戰機相同，皆是連續射擊。

```
//////////////第一隻小怪左右移動

case(Mnstate0)
0:begin
    Mn_left[0] = Mn_left[0] + 1 ;
    if( (Mn_left[0]>=924) && (Mn_life[0]>0) )
        Mnstate0=1;
    else  Mnstate0=0;
end
1:begin
    Mn_left[0] = Mn_left[0] - 1 ;
    if( (Mn_left[0]<=0) && (Mn_life[0]>0) )
        Mnstate0=0;
    else Mnstate0=1;
end
endcase
```

關卡二為魔王關，其移動方式為透過假隨機數列造成隨機移動，隨機移動方式如右圖，設一個變數 En_move 從 0~7 去取隨機數列的值，然後去 case 這個變數，0~7 則分別代表上下左右等八方位（以上下左右為例）。

```
魔王移動
if(En_cnt>=500)begin
    En_cnt=0;En_move<=rand; end
else begin En_cnt=En_cnt+1; En_move=En_move; end

case(En_move)
0:begin if( En_top < 200 ) En_top  = En_top+1 ;end
1:begin if( En_left < 768) En_left = En_left+1;end
2:begin if( En_top > 34 ) En_top  = En_top-1 ;end
3:begin if( En_left > 5) En_left = En_left-1; end
```

魔王關還有另一個特點是其不一樣的攻擊模式，魔王關不同於第一關只能定時射出子彈，魔王有兩種不同的攻擊模式，其由 typecounter 從 0~5~去控制，第一種模式即為正常的連續射擊，第二個模式則是散彈槍，如果正面吃到散彈槍，則會扣 10 滴血。

```
//////////魔王第一種攻擊模式
 0,1:begin
    if(En_bcounter >= 1500)begin
        for(en_bnum = 0; en_bnum <= 9; en_bnum = en_bnum + 1)
            En_Bu_en[en_bnum] = 0;
        En_bcounter = 0;
        typecounter <= typecounter + 1;
    end
    else  En_bcounter = En_bcounter + 2;
    end
    //////////魔王第二種攻擊模式
  2,3,4,5:begin
        if(En_bcounter >= 800)begin
            for(en_bnum = 0; en_bnum <= 9; en_bnum = en_bnum + 1)
            En_Bu_en[en_bnum] = 0;
            En_bcounter = 0;
            if(typecounter>=5)
                typecounter=0;
            else typecounter <= typecounter + 1;
        end
            else  En_bcounter = En_bcounter + 2;
     end
  endcase
```

4. 子彈發射、移動與判定

我們讓每個子彈都有一個 Enable（Bu_en）；首先設一個 counter 從 0~1000，每 100 發射一顆子彈（Bu_en=1），下方的 for 迴圈則是設置子彈射出的位置在飛機的槍管前端。

```
begin
    ////////////////////////// 子彈定時產生

    if(bcounter >= 1000)
    bcounter = 0;
    else  bcounter = bcounter + 1;

    for(bnum=0;bnum<=9;bnum=bnum+1)begin
        if(bcounter==(bnum+1)*100)
            begin
            Bu_en[bnum] <= 1;
            Bu_top[bnum] <= Jet_top - 23;
            Bu_left[bnum] <= Jet_left + 61;
        end
    end
```

第二張圖則是飛機的移動，如果這顆子彈的 Enable 為 1 的話，其 y 左標 - 1（敵人則是 y+1）

```
if(Bu_en[bmove] == 1)
    Bu_top[bmove] = Bu_top[bmove] - 1;
else Bu_top[bmove] = Bu_top[bmove];
```

第三張圖是飛機的判定，if 後面是放子彈與戰機碰到的條件（因為長度關係只截取前面一段），如果碰到的話，Bu_en 歸零，然後飛機血量 – 1。

```
if((Bu_top[bmove] <= Mn_top[0] + 100) && (Bu_to
    begin
        Bu_en[bmove] = 0;
        if((Mn_life[0]!=0) && (Jet_life!=0) )
            Mn_life[0] = Mn_life[0] - 1;
    end
```

5. 假隨機數列

假隨機數列是將一組 8 個 bit 的數列隨機相加，例如 bit1 就是由 bit2+bit7，以此加法來使數列不容易被預測。而我們需要的隨機數字則從中取出 3 個位數即可，在這次專題我們需要一個 3bit 的數來控制魔王 8 個方位的移動，所以我們就選此數列的 bit 1,3,7，當作我們的移動參數。

```
//////      假隨機數列 用來判定魔王移動方向    //////////
 reg [7:0] rand_num;
parameter seed = 8'b1111_1111;
integer rcnt;
always@(posedge En_CLK)
begin
    if(pReset || start)
        rand_num  <= seed;
    else
      begin
        if(rcnt==100) begin
           rcnt=0;
            rand_num[0] <= rand_num[1] ;
            rand_num[1] <= rand_num[2] + rand_num[7];
            rand_num[2] <= rand_num[3] + rand_num[7];
            rand_num[3] <= rand_num[4] ;
            rand_num[4] <= rand_num[5] + rand_num[7];
            rand_num[5] <= rand_num[6] + rand_num[7];
            rand_num[6] <= rand_num[7] ;
            rand_num[7] <= rand_num[0] + rand_num[7];
           end
        else rcnt<=rcnt+1;
      end
end
```

6. 去背（顯示先後順序）

以戰機為例，在戰機的參考點所框出之範圍內，若需優先顯示的物件，如子彈、敵人、血條等，當前輸出不是白色或黑色（因物件圖檔背景色而異）時，優先指派上層顯示的東西。

而在最底層，即是當戰機輸出為白色（圖檔背景色）時，將指派為顯示背景，看起來即為「去背」。

```
if(Mn_Bu_out != 4095)
    vga_out <= Mn_Bu_out;
else if(Mn_life_out != 0)
    vga_out <= Mn_life_out;
else begin
    if(Mn_out != 4095)
    begin
        vga_out <= Mn_out;
    end
    if(Mn_out == 4095)
    begin
        if(Jet_out != 4095)
            vga_out <= Jet_out;    // output jet
        if(Jet_out == 4095)
            vga_out <= back_out;
    end
```

B. vga module

輸入 65MHz Clock（1024x768@60Hz 顯示所需），輸出 horizontal sync、vertical sync、x、y。

C. color

輸入 12 bit RGB 訊號，拆分成三組分別 4 bit 的 R、G、B 訊號輸出。

肆、問題討論：

Q：address 設置方法？

A：最開始 address 設定是在 always block 裡讓 address = address + 1 ，使其在每次進入範圍內時 address 自動增加 1。結果顯示出來的卻是如同錯位一般的亂碼。為了解決這個問題，改為多設置幾個變數，透過相對座標的概念，直接算出、指定座標給 address。

Q：如何去背？

A：最開始顯示物件時沒有想到要去背，結果顯示的物件全部都是一個一個方框，還有難看的黑白背景色。但是在顯示那邊多加幾個條件設定，就可以達到去背的效果了。後來更是用一樣的方法達到調整顯示順序的效果

Q：如何讓多顆子彈連續發射？

A：因為一個子彈其實就是一個物件，所以我當初在想子彈連續發射的時候，是直接設定 10 顆子彈，然後設定一個 counter 讓子彈一顆一顆發射，當第一顆子彈跑出螢幕消失，或者碰到敵人消失時，Enable 就會歸零，等待 counter 重新數到他的時候再接續發射。最後測試發現這個數目的子彈，可以剛好完整的呈現在螢幕上，不會發生子彈中間有中斷的情形。

Q：魔王關多重攻擊模式 bug？

A：在製作魔王的多重攻擊模式的時候，散彈槍一直發生突發狀況，像是只跑出一半的子彈，或者有奇怪的子彈從螢幕上方跑出來，有的時候甚至會「卡彈」，我後來發現是子彈的 Enable 沒有設定好，才會造成子彈有這種亂飛的情形。

Q：遊戲平衡的調整？

A：在每個遊戲中，最重要的就是平衡設定，平衡簡言之就是難易度，如果遊戲太簡單可能沒有吸引力，如果太難可能會讓玩家挫折感很重，因此在本次遊戲中，我們更改了很多次血量以及子彈發射速度，就是為了達到最好的遊戲平衡。

伍、專題結果：

如連結中影片所示：https://youtu.be/L7aoIpFIz1E

陸、分工合作：

0710130 黃鼎翔

top module 架設、vga module、color module、block memory 設定、

背景顯示、物件顯示、去背。

0710145 洪若軒

game module：

戰機操控、敵人移動、子彈生成、子彈射擊模式、子彈擊中判定、假隨機數列。