Final Project: Battle tank

107061112 王昊文

107061113 李柏葳

Project objective

製作一個1v1雙人對戰遊戲，在地圖內互相向對方開火，兩分鐘內擊殺對方最多次的贏。

Design Specification

✓ I/O

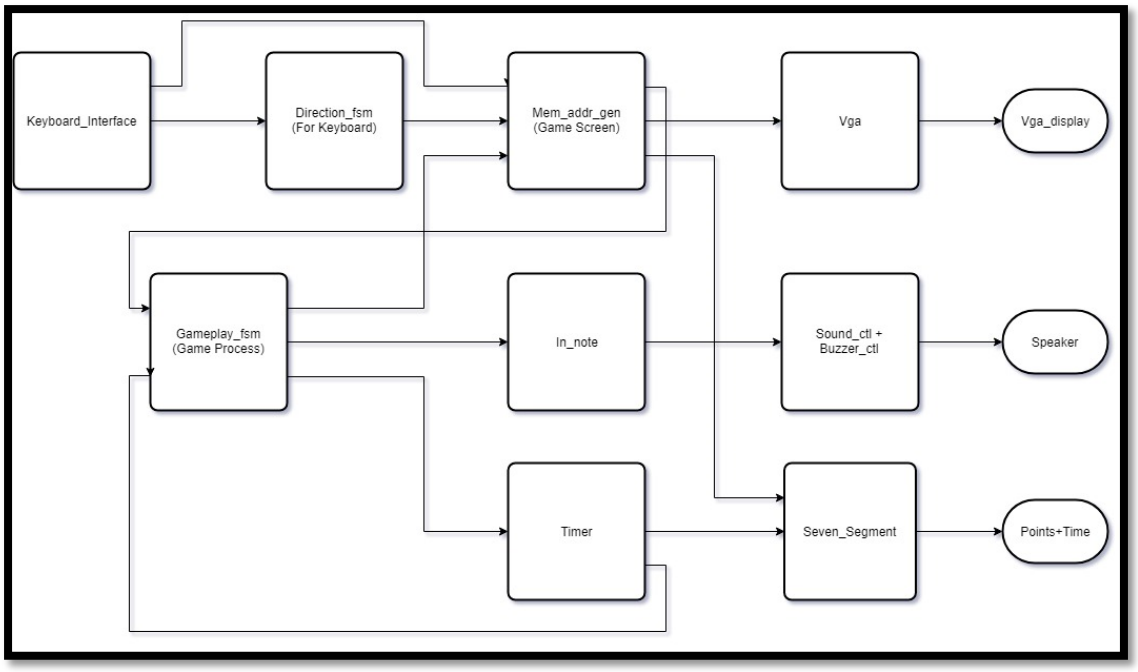
```
// basic input
Input clk            // crystal clock
Input rst            // high active reset
// DIP switch input
Input dis_mode       // choose display score or time
// for keyboard
Inout PS2_CLK
Inout PS2_DATA
// for vga
Output [3:0]vgaRed
Output [3:0]vgaGreen
Output [3:0]vgaBlue
// 7-segs display
Output [3:0]lightctl_out
Output [7:0]display
// speaker
Output audio_mclk       // master clock
Output audio_lrck       // left-right clock
Output audio_sck        // serial clock
Output audio_sdin       // serial audio data input
```

Design Instruction:

打開或重置裝置後，遊戲會先顯示開始介面。按Z可以向上選擇，按X可以向下選擇。選擇想要的模式後按C即可進入遊戲。遊戲一開始，在左上(綠)以及右下(紅)各有一台坦克。紅色為玩家一，由小數字鍵盤的5、2、1、3各控制上下左右，Enter鍵射擊。綠色為玩家二，由W、S、A、D各控制上下左右，空白

鍵射擊。子彈及坦克都不能穿牆，坦克射出的子彈碰到敵人會使敵人爆炸，並取得一分。遊玩時，分數及倒數時間會顯示在FPGA板上，時間歸零時，會顯示哪一個玩家取得勝利，平手則顯示沒有玩家勝出。

Logic Block:



此遊戲的晶片需具備的功能分別為VGA顯示輸出、坦克及子彈方向位置紀錄、牆及邊界及坦克間碰撞判定、時間及雙方擊殺數計算顯示、遊戲場景變換及speaker音樂訊號輸出。以下會一一說明其功能如何產生。

1. VGA顯示輸出:

要做出遊戲的第一步，就是要能顯示圖片到螢幕上。我們總共需要顯示起始畫面、遊戲畫面及結束後勝利畫面。

主要的訊號處理顯示模組都和lab11用的模組(vga & vga_clk)一樣，只是mem_addr_gen內輸出和當前h_cnt及v_cnt對應的pixel_addr以及儲存圖片的ROM內的素材圖片(附件一)不一樣。

圖片的移動及印刷都在mem_addr_gen中處理，在這裡只講印出的部分。我們定義每張圖片的最左上角為該圖片的座標點。而在素材圖片中也有其座標點和圖片大小。我們的做法是先判斷v_cnt及h_cnt是否在該圖片應顯示的地方上，也就是在螢幕上顯示位置座標 + 圖片大小內(水平和垂直都要判斷)。如果是的話，pixel_addr就是(目前v_cnt – 實際起始座標縱軸 + 素材圖片座標縱軸) * 素材圖片寬度 +(目前h_cnt – 實際起始座標橫軸 + 素材圖片座標橫軸)，目前cnt - 實際起始座標是因為要使其在圖片左上角時保持為零，但隨著cnt增加，該數字也能以相同速度增加，要加素材圖片座標則是讓起始點移動到該圖片在素材圖片上的座標點。如此，就能讓v_cnt & h_cnt在某個螢幕上像素時，pixel_addr

也能輸出對應的像素出去。

而圖形的優先顯示(哪個圖要在哪個圖上面)，則是由if else來協調，當第一張圖片符合條件時，就輸出第一張圖片的pixel_addr，不符合時就檢測第二張，就這樣持續下去，直到最後就沒有條件判定(如遊戲進行畫面的地圖)。

然後由於記憶體不夠，所以我們只能把原本遊戲地圖解析度從640 * 480調到320 * 240，而讀取方式是將h_cnt & v_cnt除以二，再取代條件判定及pixel_addr輸出的h_cnt及v_cnt，如此只要h_cnt除二的數字一樣，他就輸出一樣的pixel_addr，也因此圖片會被放大一倍。而起始畫面則是將重要圖片之外的背景都切掉，再塞進素材圖片裡。

2. 坦克及子彈目前方向&位置紀錄:

雙方坦克的位置和方向以及子彈的位置和方向都各用一些特定的暫存器去存，模組也是位於mem_addr_gen裡面。坦克的位置座標很簡單，由keydown直接判斷。當壓著w鍵時，P1的坦克座標會每過一個clk_19(100m / 2 ^18hz)就加一，a、s、d和P2以此類推。坦克的方向判定也有些類似，當壓下w鍵後，暫存器會存入00，代表向上，a、s、d則各存入01、10、11，當鍵盤沒壓時，就一直存入目前所在方向。坦克的方向將會決定輸出的坦克圖案以及子彈的方向。

子彈的方向是由坦克射出時砲管對準的方向為準，然後朝該方向直線前進。所以在玩家按下空白鍵時，子彈方向的暫存器會同時存入該玩家的方向，之後就不會改變方向。而子彈產生(被射出)的座標，是坦克目前的砲管中心決定，按空白鍵後會輸入其座標到該子彈座標上，然後再根據子彈的方向來決定座標縱軸或橫軸的值要增減。

3. 牆及邊界及坦克間碰撞判定:

子彈碰到物體要消失，坦克碰到物體要停下來，此乃理所當然，但放在verilog中並不好打。子彈由bullet_en來決定要不要顯示在螢幕上。在其左上角的點為該圖片座標點，以圖片的長與寬畫一個長方形，為其攻擊方塊判定。當按下空白鍵時，bullet_en變為一，並向前射擊，但當該長方形的邊界碰到牆及玩家判定邊界後，bullet_en會變回零，碰到的是另一個玩家時，會特別發出death訊號，之後用於計分和場景轉換。同時該子彈的座標會全為零，被放到左上角，等待下次的擊發，這些程序大部分由hit_wall_bullet處理。順帶一題，遊戲每位玩家一次只能射一發子彈是因為每台坦克只有一個子彈的模組，最多只能顯示一張子彈的圖片。

坦克的邊界判定也是由其左上角拉出一個長方形而成，預設是30 * 30。坦克和牆擊另一坦克的碰撞較難處理，因為碰到牆，不能讓它完全不能動，而是只有被阻擋的該方向不能動(ex.上面有牆，坦克只有上面不能繼續前進)。所以坦克我將四個方向分開來判斷，當坦克正方形右邊邊界碰到牆的邊界時，它向右的功能會被disable，但其他方向還是enable狀態，所以能朝其他方向移動，其

他以此類推，這些程序大部分由hit_wall_tank處理。牆的邊界有四十幾條，每條都還要去小畫家量起始和結束像素，這些就能花掉好幾個小時。

4. 時間及雙方擊殺數計算顯示:

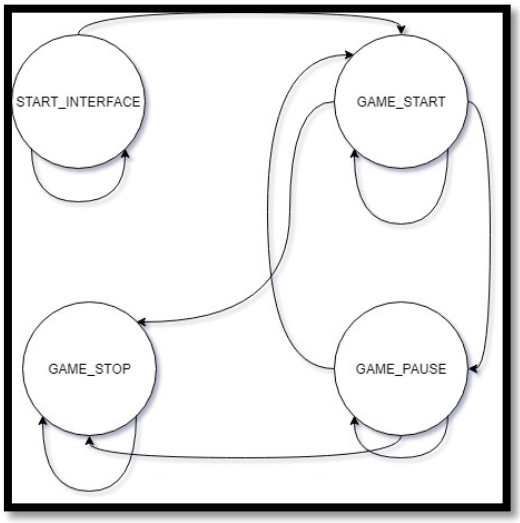
倒數計時器的模組—timer來自前面的lab。遊戲設定是2分鐘，enable來自gameplay_fsm，後面會講解每個state會輸出什麼訊號。其中一個state為game_start的state，這時fsm會輸出count_enable的訊號，讓timer成功進行計數。而當timer數到0的時候，這時timer會發出stop的訊號，讓gameplay_fsm進入stop的state。

關於擊殺數的計算，我們只需要判斷坦克的位置以及子彈的位置是否重疊即可。由於我們的設定是在子彈消失前（碰到牆或射到人）只能發射一發，因此關於position的判斷，我們必須使用mem_gen_addr的訊號。當position重疊時，我們在mem_addr_gen就設好兩個分數的變數來記錄分數，然後傳送給seven segment display來做顯示。

由於我們只有使用一個版子，因此我們還利用了dip switch來做選擇，撥動dip即可選擇想要輸出的模式。

5. 遊戲場景變換:

遊戲場景變換主要是由game_play_fsm模組來處理。在該模組中，遊戲階段被分成四個state，分別是



Start_Interface, Game_start, game_pause, Game_stop.這些state代表著遊戲的流程，透過切出這些state來控制很多變數。上圖為簡易的fsm示意圖，剛開始進入遊戲的時候會停留在start_interface，前面提到使用者按下S之後遊戲會開始，因此鍵盤上的S鍵即為start_interface跳動的關鍵。

再來，遊戲會進入開始狀態，這時候遊戲停下來只有兩個關鍵，一個是當

兩分鐘過後遊戲結束，因此必須利用timer回傳時間到的訊號。另外一個是當有人被射死的時候，遊戲會暫時暫停。然後再繼續。因此，我們必須判斷子彈的位置，固需從mem_addr_gen裡面拉出position的訊號。如果有人碰到了子彈，我們就必須將遊戲暫時暫停，然後重置兩個玩家的位置。

再來是遊戲暫停的state，剛剛提到有人被射到的時候會跳到暫停模式。這時候會播放有人死亡的音樂。大約兩秒鐘左右，因此當有人死亡的時候會停留在這個state約莫兩秒的時間，之後會繼續進行遊戲。同時間，我們也會同時進行判斷由席是否為結束狀態，如果感應到遊戲時間到的時候會進入stop state。

最後一個state為game_stop，為遊戲結束的state。只要感應到timer送來的stop訊號，遊戲就會停下來。當遊戲進到這邊的時候，會輸出訊號給mem_addr_gen，要輸出最後一張遊戲結束的畫面。

6. speaker音樂訊號輸出:

我們這組的音訊共有三種，一種是開頭的背景配樂，第二個是有玩家被射死的音樂，最後是時間到，遊戲結束的音樂。我們可以從gameplay_fsm來控制輸出的音訊信號。Gameplay_fsm將遊戲切成好幾個state來看，因此我們可在特定的state下輸出我們想要的音訊的enable訊號。譬如在前面我們提到的，我們將state切為初始介面、遊戲開始、遊戲暫停（有人死掉的時候）、以及結束state。因此，在處於初始介面的時候我們可以輸出播放背景音樂的訊號，其他音效依此類推，總之透過fsm來控制音效的輸出。

當音效enable從fsm裡面輸出的時候，會進入到in_note模組裡面。這個模組裡面記著所有背景音樂的音符。在設定的時候我是事先將曲子的速度，轉換成需要的clock frequency，然後利用一個counter去數，一下一下去填入想要的音高。我們使用的音樂為馬力歐的背景配樂，有很多的和弦，因此我們分成了好幾個聲道，因此in_note模組看起來又臭又長的原因是我們將好幾個聲道都輸了上去。

當我們輸出想要的音的頻率的時候，我們可以利用好幾個buzzer control來輸出震幅，然後將這些震幅全部相加丟進speaker_ctl來處理就可以啦(speaker_ctl以及buzzer_ctl皆為之前lab使用過的模組)。

在震幅相加的時候要注意震幅不能過大，不然聲音很容易爆掉。

✓ Design Implementation

Input		
clk	*rst*	*dis_mode*
W5	R2	T1

Inout	
PS2_CLK	*PS2_DATA*
C17	B17

Audio Output			
audio_mclk	*audio_lrck*	*audio_sck*	*audio_sdin*
A14	A16	B15	B16

Output ssd							
display[0]	***display[1]***	***display[2]***	***display[3]***	***display[4]***	***display[5]***	***display[6]***	***display[7]***
V7	U7	V5	U5	V8	U8	W6	W7

Output vgaRed			
vgaRed[0]	*vgaRed[1]*	*vgaRed[2]*	*vgaRed[3]*
G19	H19	J19	N19

Output vgaGreen			
vgaGreen[0]	*vgaGreen[1]*	*vgaGreen[2]*	*vgaGreen[3]*
J17	H17	G17	D17

Output vgaBlue			
vgaBlue[0]	*vgaBlue[1]*	*vgaBlue[2]*	*vgaBlue[3]*
N18	L18	K18	J18

Output vga	
vsync	*hsync*
R19	P19

Discussion:

這次的實驗我們統整這學期所學，並加深加廣，包括倒數計數器，鍵盤、speaker等等。最困難的部分大概就屬vga了。最難的部分在於如何有效率的使用內部記體，然後達到自己想要的顯示效果，包括如何運用一些flag來指示現在該顯示什麼。光是調整那些顯示座標就花了我們好多時間。

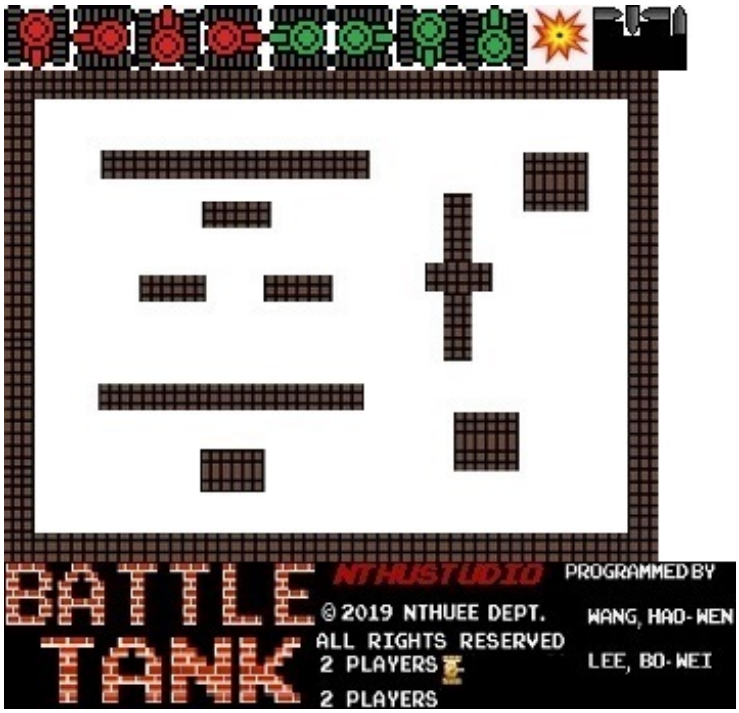
這次實驗中最特別的大概就是speaker的應用，一開始我們都不知道如何至做出和弦的感覺，後來發現原來只要將震幅相加起來就可以了，這讓我們的speaker的實用性提高很多，只要有心就可以用speaker製作出高品質的音樂。

最後這次lab中最難忘的經驗大概就是我們嘗試將兩個板子連起來，不知道是不是在過程中接線接錯，竟然將一台電腦直接燒掉，推測應該是power的地方接錯，才導致這個悲劇，不過詳細情況仍然不是很清楚，總之以後接線的時候會多注意一點就是了。

Conclusion:

這學期的邏設實驗總算是告一段落了，第一次讓我體驗到電機系住在實驗室還有通宵的感覺，整學期可以說是收穫滿滿。這次的遊戲製作出來可以說是成就感滿滿，也讓我學到團體寫程式究竟該如何雙向溝通，總算成功製作出project。

附錄一:



輸入到ROM blk_mem_gen_0的.coe檔圖片