**Final Project Report**

**第21組 112061105陳睿倬 112061240吳宇哲**

**Topic**

A simple shooting game

**Input/Output table**

**(藍字為後來修改的部分(和proposal不同))**

|  |  |  |  |
| --- | --- | --- | --- |
| **Input** | **function** | **output** | **Function** |
| Keypad w | Character move up | 7-Segement 4 digits | Show the game static |
| Keypad a | Character move left |  |  |
| Keypad s | Character move down | Speaker | Game sound effects |
| Keypad d | Character move right |  |  |
| Keypad space | shoot | LED L1 | Now\_state  顯示當前七段顯示器數字所代表的資訊  00:pts, 01:hpts,  10:time, 11:rounds |
|  |  | LED P1 |
| Button center | Switch showing mode | LED P3 | stop\_state |
|  |  | LED V13 | otk\_led |
| Switch R2 | Open/Off the game | LED P3 | Pause state |
| Switch V17 | Pause/Resume |  |  |
| Switch T1 | Switch background |  |  |
|  |  | LCD | Show the game screen |

**Functions of the Shooting game**

Switch 1(R2)開關，用於控制遊戲開關，這個撥桿也有助於處理重設reset的問題。

進入遊戲後：（IO 設定）

Input

1. 鍵盤wasd控制角色移動
2. 空白鍵用於射擊
3. Button (center button)用於切換七段顯示器顯示的數據（當前分數、最高分數、時間（計時）、死亡次數）
4. Switch (V17)用於遊戲中暫停，於暫停狀況下，LED 2(led P3)會亮起，反之正常遊玩時不會。
5. Switch T1 切換背景圖片

Output

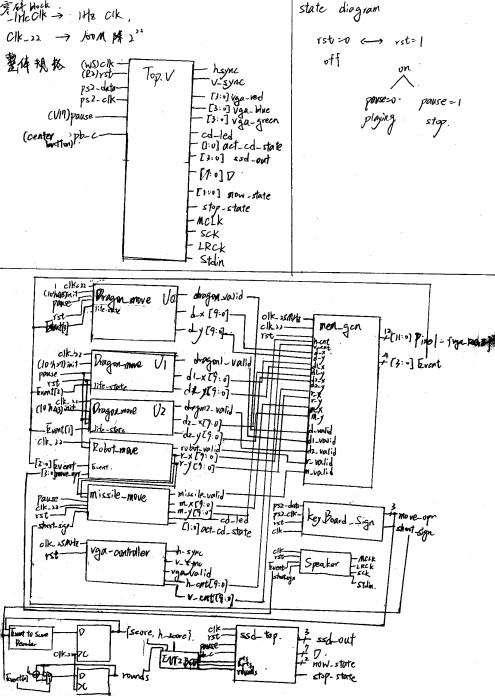
1. 七段顯示器顯示Input 3當前分數、最高分數、時間（計時）、死亡次數）

要顯示的數字

1. Speaker用於音效輸出（如射擊音效，角色死亡音效、擊敗敵人之音效）
2. LED 1 顯示遊戲（程式正常運行） 🡪拔除
3. LED P3顯示當前是否處於暫停狀態。或是也可用於結束暫停狀態時的倒數
4. LED V13顯示(閃爍)當前處於大招狀態(必殺技狀態)
5. LED L1, P1代表當前七段顯示器的顯示狀態00:分數, 01:最高分數(每次生命),10:time(遊玩時間), 11:rounds(死亡次數)

**Logic Diagram**

* **Top module**



* **dragon\_move.v**

**一張含有 文字, 圖表, 方案, 圖解 的圖片

自動產生的描述**

* **robot\_move.v**

**一張含有 文字, 圖表, 方案, 工程製圖 的圖片

自動產生的描述**

* **missile\_move.v**

**一張含有 文字, 圖表, 方案, 工程製圖 的圖片

自動產生的描述一張含有 圖表, 行, 文字, 方案 的圖片

自動產生的描述**

* **LFSR.v**

**一張含有 圖表, 方案, 文字, 行 的圖片

自動產生的描述**

* **INT2BCD.v**

**一張含有 圖表, 方案, 寫生, 行 的圖片

自動產生的描述**

* **keyboard.v**

**一張含有 圖表, 方案, 工程製圖, 寫生 的圖片

自動產生的描述**

* **VGA.v (同老師提供的檔案)**

**一張含有 圖表, 工程製圖, 方案, 圖解 的圖片

自動產生的描述**

**一張含有 文字, 圖表, 方案, 工程製圖 的圖片

自動產生的描述**

* **\_7SegShow.v**

**一張含有 圖表, 文字, 方案, 圖解 的圖片

自動產生的描述**

* **BTNC\_signal\_ctl.v**

**一張含有 文字, 圖表, 方案, 工程製圖 的圖片

自動產生的描述**

**一張含有 圖表, 文字, 方案, 工程製圖 的圖片

自動產生的描述**

BTNC\_signal\_ctl.v右半邊

* **display\_number\_to\_cnt.v**

**一張含有 圖表, 方案, 行, 圖解 的圖片

自動產生的描述**

* **Timer.v**

**一張含有 文字, 圖表, 方案, 工程製圖 的圖片

自動產生的描述**

**一張含有 文字, 圖表, 方案, 工程製圖 的圖片

自動產生的描述**

* **FSM.v**

**一張含有 圖表, 方案, 圖解, 工程製圖 的圖片

自動產生的描述一張含有 文字, 圖表, 方案, 圖解 的圖片

自動產生的描述一張含有 文字, 圖表, 方案, 圖解 的圖片

自動產生的描述**

* **speaker.v**

**一張含有 文字, 圖表, 方案, 工程製圖 的圖片

自動產生的描述一張含有 文字, 圖表, 方案, 工程製圖 的圖片

自動產生的描述**

Speaker.v右半邊

**一張含有 文字, 圖表, 方案, 圖解 的圖片

自動產生的描述一張含有 文字, 圖表, 方案, 平行 的圖片

自動產生的描述**

* **mem\_gen.v**

**一張含有 文字, 圖表, 平行, 行 的圖片

自動產生的描述一張含有 文字, 平行, 圖表, 方案 的圖片

自動產生的描述**

**一張含有 文字, 圖表, 方案, 平行 的圖片

自動產生的描述一張含有 文字, 平行, 圖表, Rectangle 的圖片

自動產生的描述**

**一張含有 文字, 螢幕擷取畫面, Rectangle, 平行 的圖片

自動產生的描述**

**一張含有 文字, 設計 的圖片

自動產生的描述**mem\_gen.v下半部

**Design Specification/Implementation**

* **Dragon\_move.v**

規格：

input clk\_22, rst, pause; // clk\_22: 100M/2^22, R2, V17

input [9:0]init; // random seed

input life\_state; // 外部廣播訊號(from mem\_gen): 龍的死亡事件廣播

output reg [9:0]d\_x, d\_y; // 現在位置，定位座標為角色圖片的左上角

output reg show\_valid; // 現在是存活或死亡狀態，死亡狀態不會顯示角色

首先講龍的狀態alive，共有兩個：存活、死亡。

在存活狀態下：

若碰到邊界或碰到飛彈或機器人(life\_state == 1)就進入死亡狀態。

在死亡狀態下：

若cd\_cnt == 100，重生時間結束，回到存活狀態。

若在死亡狀態下碰到邊界(重生位置在邊界上)也會再次進入死亡狀態(不過系統裡有處理重生範圍，因此不太會發生)。

接著是它移動的原理，先談它的生成位置。只有遊戲一開始三隻龍的位置是固定的，之後每次的重生位置都會不一樣(因為有不一樣的init(random seed)。透過模數運算加一個常數調整重生範圍。移動時會恆向左邊移動，但會往上或往下是隨機的，由隨機數字direction的奇偶數決定往上或往下。

若pause(V17)啟動，狀態和位置都不會改變。

PS: 三隻龍用的模組都一樣

* **Robot\_move.v**

規格：

input clk\_22, rst, pause; // clk\_22: 100M/2^22, R2, V17

input [3:0]move\_opr; // wasd的鍵盤控制

input [3:0]Event; // 外部廣播訊號(from mem\_gen): 龍和機器人的死亡事件廣播

output reg [9:0]r\_x, r\_y; // 現在位置，定位座標為角色圖片的左上角

output reg show\_valid; // 現在是存活或死亡狀態，死亡狀態不會顯示角色

首先先講機器人的狀態alive，共有兩個：存活、死亡。

在存活狀態下：

若碰到龍(Event[0] == 1)就進入死亡狀態。

在死亡狀態下：

若cd\_cnt == 100，重生時間結束，回到存活狀態。

Cf補充：

在機器人的角色，碰到邊界不會死亡，但會強制停在那個位置(後面講述原理)，防止它移出範圍(右、下邊界可能會超出一點，因為座標定位在左上角。所以可能會出現座標未出界，但圖片出界的問題。)

接著是它移動的原理，它的重生位置恆固定。相比龍是直接改變座標，機器人會先由鍵盤訊號(move\_opr)經decoder決定nxt\_r\_x, nxt\_r\_y機器人的下個位置。若下個位置會超出邊界，那機器人的座標不會變(維持原座標)。否則，若符合要求，那現在位置就會變成下個位置的座標。

若pause(V17)啟動，狀態和位置都不會改變。

* **Missile\_move.v**

規格：

input clk\_22, rst, pause; // clk\_22: 100M/2^22, R2, V17

input shoot\_sign; // keyboard space key sign

input [9:0]r\_x, r\_y; // robot's pos

output reg [9:0]m\_x, m\_y;// 現在位置，定位座標為角色圖片的左上角

output reg show\_valid; //現在是發射中或未發射的狀態，未發射狀態不會顯示

output reg cd\_sign; // reload 訊號(代表飛彈在裝填，還不能發射)(連接到led)

output reg [1:0]act\_cd\_state; //飛彈的狀態(連接到led)

首先先講飛彈的狀態act\_cd\_state，共有三個：

準備好發射(00)、發射中(01)、裝填(10)。

在準備好發射狀態下00：

其位置會一直跟隨著機器人的位置，直到按下space(shoot\_sign == 1)，狀態變成發射中狀態。

在此狀態下，show\_valid會關閉，不會顯示飛彈圖片。

在發射中狀態下：

只會從準備發射狀態到這狀態。會從機器人的位置開始，直直向右邊移動，碰到龍會直接穿過它。若在此狀態下碰到邊界，設cd\_cnt = 0，進入裝填狀態。

在裝填狀態下：

cd\_cnt一直上數直到cd\_cnt = 10。回到準備發射狀態，這期間也會一直跟著機器人的位置。

接著是它移動的原理，非常簡單，根據狀態的不同改變其模式。若不是發射中狀態，那麼飛彈就會一直跟著機器人。反之，會從機器人的位置作為起始點，直直向右發射直到碰到邊界。

若pause(V17)啟動，狀態和位置都不會改變。

* **KeyBoard\_Sign.v**

只偵測wasd, space這五個按鍵。從key\_down偵測它是否是處於按下的狀態。一張含有 文字, 螢幕擷取畫面, 字型, 代數 的圖片

自動產生的描述

只是相比於直接用idx做拜訪，我這裡是用位元運算去做訪問key\_down該位元的數值(0 or 1)

Speaker.v

幾乎直接搬lab7的程式碼，只改了top的部分。由於Speaker\_CTL和Buzzer的設計幾乎大同小異，這邊就不特別敘述了，只講top的部分。

由於想做到和聲的效果，所以設置了兩個decoder，讓左右聲道可以不同。

背景音樂是由b\_music設定，用數字樂譜的形式，經過decoder後轉成音符頻率。左聲道會持續地撥放背景音樂，而右聲道在預設狀態下也會撥放背景音樂。但右聲道的背景音樂播放順序和左聲道不同，是顛倒的。這樣的設計是為了達到和聲的效果。此外，若遭遇一些遊戲事件，如：射擊、機器人死亡，右聲道會出現一些單一音符來提醒玩家。

一些零碎的Decoder。

\_1HzClk: 產生1Hz的訊號

INT2BCD.v： 將binary code convert to BCD code

Clk\_22.v： 產生100MHz除2^22頻

LFSR： 產生亂數0~511，輸入init作為亂數種子

* **mem\_gen module(Dragon\_mem\_gen.v)**

該模組是負責處理記憶體位置的生成和VGA的顯示。

首先，要知道如何判斷該位置是否要顯示角色圖片。

先計算當前vga\_ctl數到的座標，和角色的座標距離是多少。再根據它是否在角色座標的右下方(因為角色做標是定位在圖片左上角)且距離是否在圖片大小內。並根據當前角色的存活狀態，決定是該點是否要顯示角色的圖片。若皆符合的話，角色的enable會設為1，代表要顯示該角色。再經由距離生成它的記憶體位置，經blk\_mem\_gen後輸出該角色圖片的pixel。但要注意，這裡還不是最終輸出的結果，因為還要考慮其他角色的碰撞。

所有角色圖片的生成方法都同上述所解釋的原理。

最後，我們再藉由d\_enable, d1\_enable, d2\_enable, m\_enable, r\_enable送進decoder(case)，處理多個角色圖片重疊(碰撞)時的狀況，來決定該座標最後應該要輸出誰的圖片，或是都不輸出(背景圖片)。

此外，我們藉由這裡處理角色碰撞(重疊)的同時，判斷角色的死亡事件。所以decoder的輸出不只有Pixel，還有r\_die, d\_die, d1\_die, d2\_die。

注意，這裡說這些變數只是事件而已。因為這些事件不會一直保存(所以我不會叫它「狀態」)。

因此，這裡會遇到一個難題。在生成這些pixel的時候，我們的時脈是25M Hz。而這些事件還要廣播到各自角色的move 模組，但這些模組只有100M / 2^22的頻率，相比25M 小很多。為了我們要讓這個事件的訊號延長一點，我們設計一個cnt來補足這兩個頻率間的時差，讓clk\_22可以讀到這個事件的訊號。但是就算有了轉換，clk\_22還是會跟不上當前最新的事件。導致可能出現遊戲畫面和實際碰撞後的結果有所誤差。所以我們Event只送出(廣播)最新的事件而已。這樣的優點是我們可以達到即時的效果，而不會出現延遲。但缺點是，如果舊的資訊還沒讓clk\_22的時脈讀到，就被新的事件蓋掉時，可能會出現封包的損失。不過我們設計時認為事件的改變不會那麼頻繁，所以就暫時擱置了這樣的狀況。

但實際遊玩時，還是有機率出現延遲的狀況，就是因為出現了資訊的損失(還沒讀到就被蓋掉了)。不過我們把它當成此遊戲的隨機性，讓這遊戲多一些刺激感。一張含有 文字, 字型, 螢幕擷取畫面, 代數 的圖片

自動產生的描述

* **ssd\_top module(ssd\_top.v)**

該模組負責dip\_switch 和button對七段顯示器的控制與其顯示。

其中包含了五個子模組，分別為：

1. Timer.v：

當使用dip\_switch(rst)重置整個系統時，該模組會開始計時，並同時times(時間)、time\_out與stop\_state傳回ssd\_top。

當計時到59分59秒：

模組回傳time\_out為1，計時器停止計時。

當撥動dip\_switch(pause)：

計時器停止計時，times數值固定，模組回傳stop\_state為1。

當再次撥動dip\_switch(pause)：

計時器開始計時，times數值更新，模組回傳stop\_state為1。

1. BTNC\_signal\_ctl.v：

該模組負責pb\_center的訊號(pb\_signal)產生。

1. FSM：

該模組負責pb\_center對七段顯示器控制的判斷。

FSM中有4個state，每一個state負責一種(pts/hpts/times/rounds)的顯示，當接收到pb\_center的訊號時，會切換state並且輸出對應的數值到display\_number。

當整個系統重置：

state為STATE\_1，輸出pts的數值到display\_number。

當按下pb\_center：

state切換到下個state，輸出對應的數值到display\_number。

1. display\_number\_to\_cnt.v：

當FSM將要從七段顯示器顯示的數字輸出(display\_number)後，該數字會傳到此模組中進行拆分，把一個16位元分解為四個4位元(cnt0/1/2/3)，再輸入到\_7SegShow做輸出顯示。

1. \_7SegShow.v：

該模組負責在七段顯示器上輸出從display\_number\_to\_cnt輸入的數字。

* **CF後續擴充：**

由於後面又更新了一些功能(切換背景、大招)所以在此就只用文字敘述。

大招的部分：

寫在mem\_gen裡面，再外接一些訊號到led。若擊殺數累積10次，就會自動進入大招模式—會有三個飛彈一起發射。基本的控制就由otk\_sign, otk\_cnt和cd\_cnt處理，透過簡單的fsm控制。擊殺時，otk\_cnt 累加直至10次後歸0，進入大招狀態(otk\_sign = 1)。大招狀態開啟時，將cd\_cnt歸零，然後開始累加直至200。期間otk\_cnt會維持0的狀態，而led V13會一直閃爍。過了時間後就回到原本狀態(otk\_sign = 0)，otk\_cnt重新累加。

而如何一次發射三顆飛彈呢?透過計算，在飛彈座標的上下20距離的地方假設兩個新的飛彈座標。在透過一些計算和enable偵測，就能在不多開state和module的情況下生成。(可以把這些計算想像成座標換算)

背景圖片：

多了背景圖片，使畫面不要那麼空虛。礙於角色非長方形，但還好它們的背景圖片都是白色。因此只要偵測該點的pixel為白色時，就顯示背景圖片。至於背景圖片的切換，就用多工器處理即可。

**//由於final project的架構複雜，推薦可以看程式碼裡的註解來了解。**

**// logic diagram和code都有上傳到github**

**//更簡單清楚的logic\_diagram可以在vivado的RTL Analysis看到，由於版面原因，這裡就不再附上。**

**// github link:** [**https://github.com/rickyC3/Logic\_Design\_Lab\_Final\_project**](https://github.com/rickyC3/Logic_Design_Lab_Final_project)

**Discussion**

在我們的專案中，我們設計了一個涉及龍與機器人的射擊遊戲。由於專案的複雜性，我們採用了模組化的方法，將移動控制、碰撞檢測和VGA顯示等任務分成獨立的模組，這樣的設計讓除錯、測試和整合變得更加容易。

主要發現

模組化設計的效率:

採用獨立的模組來處理龍和機器人的移動（例如 Dragon\_move.v 和 Robot\_move.v）是非常有效的。每個模組都可以獨立開發和測試，確保特定功能在整合前運作正常。這種模組化設計也便於故障排除，因為問題可以被隔離在單一組件中​​。

碰撞檢測:

透過座標檢查和狀態管理來實現碰撞檢測對遊戲動態至關重要。我們基於像素距離和座標的碰撞檢測在測試中得到驗證，確保遊戲中的互動可靠。然而，管理狀態變化的時機（例如從存活狀態轉變為死亡狀態）需要謹慎同步，以避免漏檢或誤報​​。

隨機性與遊戲動態:

在遊戲中加入隨機元素，例如龍的位置和移動方向的隨機種子，增加了遊戲的重玩性，確保每次遊戲會話都是獨特的。這種方法證明了平衡可預測性和隨機性以維持玩家參與的重要性。

**Problems and solutions**

同步問題:

一個重大挑戰是不同模組之間的同步，特別是考慮到不同時鐘速度（例如像素生成的25MHz與事件處理的較慢速度）。這種差異偶爾會導致時間問題，導致狀態變化未能在所有模組中正確反映。利用counter去彌補這樣的時差，使較快頻率的訊號存在時間(valid time)久一點，讓慢頻率變數可以收到事件訊號。

資源管理:

有效管理資源，特別是記憶體和處理能力，是另一個挑戰。mem\_gen 模組需要處理多個圖像輸出，同時確保平滑的性能。優化技術（如最小化記憶體使用和優化像素生成算法）至關重要，但這需要相當多的開發時間。透過適當的縮小圖片，以及座標變換的方法，讓我們在製作大招時，不需要太多的module去處理，節省運算效能。

角色的不規則性

由於角色不是一般的長方體，所以在處理角色和背景以及角色間的顯示就變得有點挑戰性。但只要細想一下，便能想到透過統一的角色的背景顏色，在利用顏色的判斷來知曉這一個像素是否是角色本體還是只是白色背景，再判斷要不要輸出背景圖片。

**Schedule and Work Table**

|  |  |
| --- | --- |
| **週數** | **內容** |
| 第14周 | 分工、畫好logic diagram |
| 第15周 | 完成七段顯示器顯示功能的程式 |
| 第16周 | 完成按鍵功能的程式 |
| 第17周 | 完成LCD顯示功能 |
| 第18周 | 測試、除錯 |

|  |  |
| --- | --- |
| **組員** | **工作** |
| 陳睿倬 | LCD顯示功能(角色移動功能)  Speaker  測試、除錯  logic diagram  report |
| 吳宇哲 | 七段顯示器與按鍵功能  Key\_Board  logic diagram  report |

**小結**：

本final project致敬組員陳睿倬在國小時做的scratch小遊戲，歡迎遊玩

<https://scratch.mit.edu/projects/136886511/>

最後附上由vivado生成的block diagram(較簡潔、整齊)。