**Final Project**

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。

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:

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:

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。

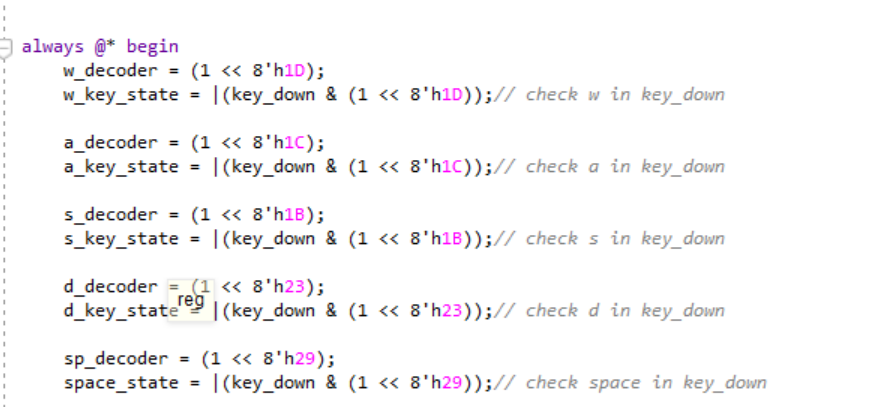
==========================================================

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。



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，又數字樂譜的形式，經過decoder後轉成音符頻率。左聲道回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，已達到和聲的效果。此外，或遭遇一些遊戲事件，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。

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

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，要先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，與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。

但注意，這裡還不是最終輸出的結果，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(留白)。

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。所以跟隨著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。

