

# Final Project report 110030031 黃玄彰 110011222 陳立珩

---

## Flappy Panda

這是一款類似 flappy bird 的遊戲，但修改了原本過一個水管就得一分的機制，而改成當熊貓吃到竹子時便增加分數，增加了遊戲的挑戰以及樂趣

## Design Specification:

### IO:

**Input: clk, rst, start, restart, PS2\_DATA, PS2\_CLK, pb\_up, pb\_down**

clk: the clock of the FPGA as the original frequency (1-bit)

rst: the element to reset all the state (1-bit)

start/restart: the button controls the game start/restart(1-bit)

PS2\_DATA: use for receiving and transmitting keyboard data (1-bit)

PS2\_CLK: the working clock of the keyboard (1-bit)

pb\_up/down: the button which control the volume up/down(1-bit)

**Output: vgaRed, vgaGreen, vgaBlue, hsync, vsync, audio\_mclk, audio\_lrck, audio\_sck, audio\_sdin, segs, ssd\_ctl**

vgaRed: the signal control the red-light intensity of the display

vgaGreen: the signal control the blue light intensity of the display

vgaBlue: the signal control the green light intensity of the display

hsync: the signal control the horizontal displaying

vsync: the signal control the vertical displaying

audio\_mclk: clock to synchronize the audio data transmission (1-bit)

audio\_lrck: clock to control the side of the serial stereo output (1-bit)

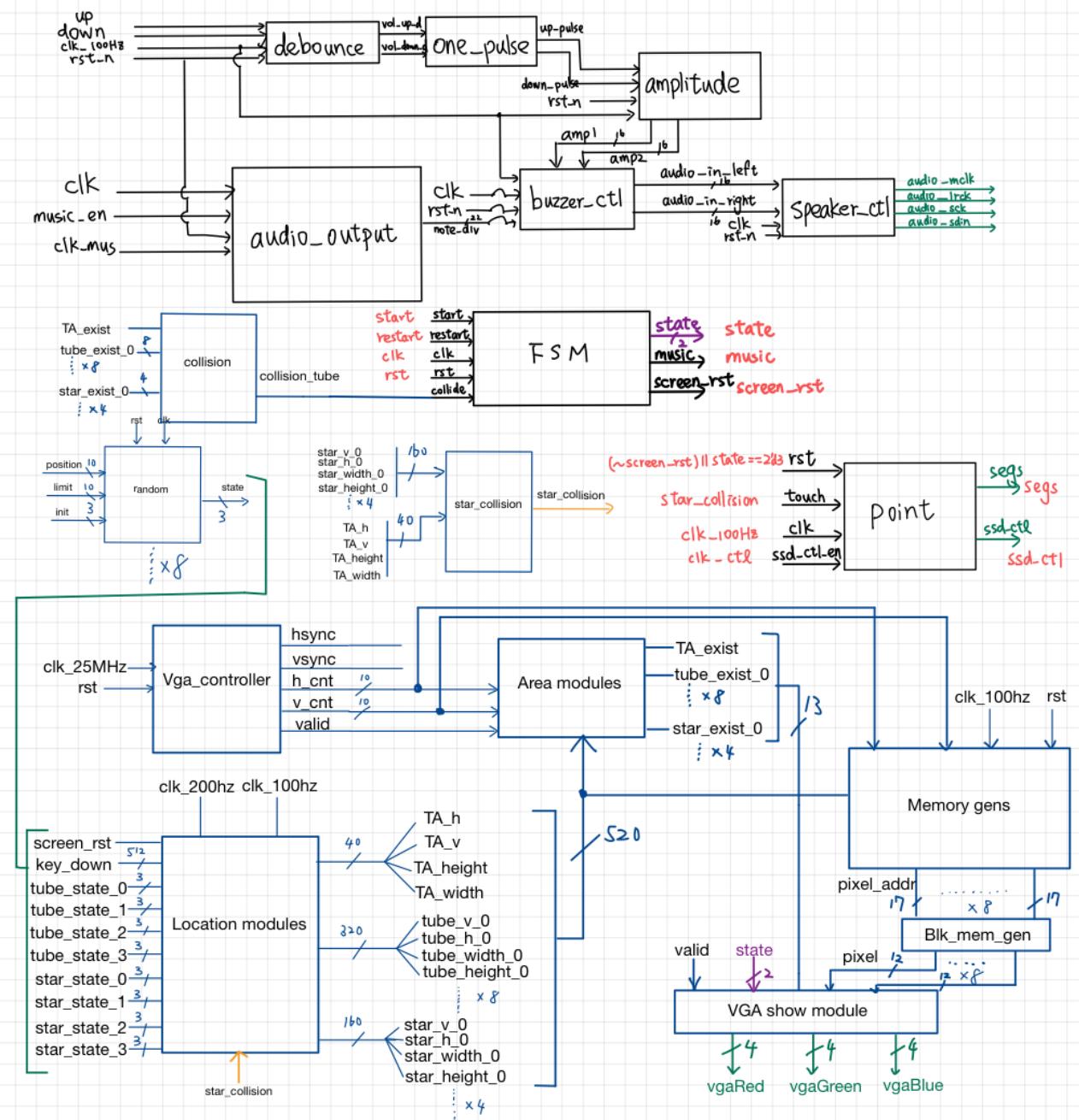
audio\_sck: clock to control the shifting of data into the input data buffer (1-bit)

audio\_sdin: the serial data output (1-bit)

segs: the seven parts to control the light of the 7-segment display (8-bits)

ssd\_ctl: the signal to control each display of seven segment display (4-bits)

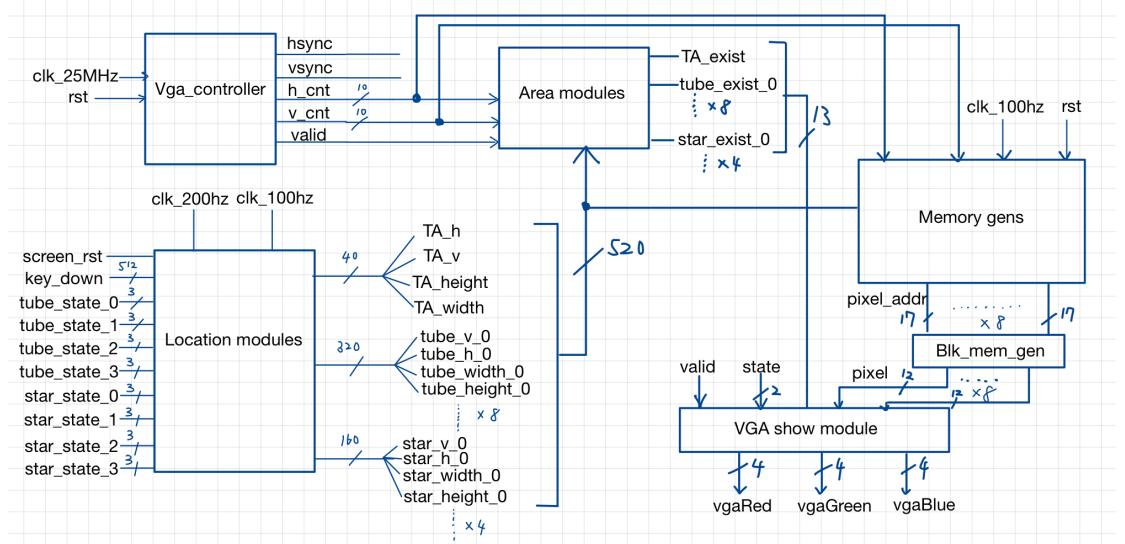
Block diagram:



## Design Implementation:

### 1. 影像呈現相關模組

為了同時讓螢幕顯示多個圖形，我們將顯示流程分成以下部分



首先利用 location modules 生成各個圖片的位置訊號（最左上小像素的位置），再將其傳給 Area modules 判斷螢幕中的哪些區域存在要印出的圖片，最後透過 VGA show module 來判斷影像圖層

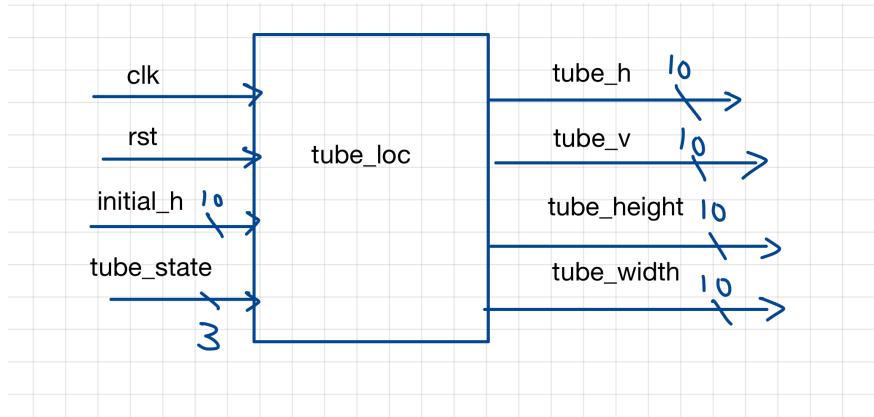
### (1). Vga\_controller:

此模組來自 lab9 的範例，用來產生顯示尺寸 640\*480 的 hsync, vsync, h\_cnt, v\_vnt, valid 等 vga 所需的訊號，在這裡我們將 h\_cnt, v\_cnt 以及 valid 訊號接出來，以供後續模組使用

### (2). Location modules:

此模組由多個模組組成，負責輸出所有欲顯示圖片的最左上角像素的位置

#### i. Tube\_loc:



#### a. 尺寸 (tube\_height, tube\_width) :

我們預設遊戲中的水管有三種不同的種類，每個種類皆由一個上水管以及一個下水管圖片組成，我們透過輸入 tube

state, 以及 updown 訊號並使用 case 語法來決定圖片其在上或下以及所對應的尺寸，以下是我們詳細的尺寸定義  
(U 代表上方水管)

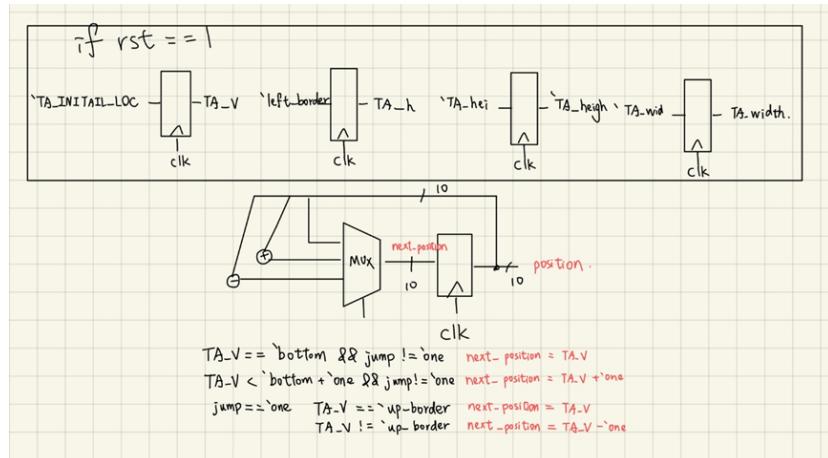
```
`define TUBE_0_W 60
`define TUBE_0_H 120
`define TUBE_1_W 60
`define TUBE_1_H 180
`define TUBE_2_W 60
`define TUBE_2_H 240
`define TUBE_3_W 60
`define TUBE_3_H 300
`define TUBE_INIT_V_0 360
`define TUBE_INIT_V_1 300
`define TUBE_INIT_V_2 240
`define TUBE_INIT_V_3 180
`define TUBE_0_W_U 60
`define TUBE_0_H_U 240
`define TUBE_1_W_U 60
`define TUBE_1_H_U 180
`define TUBE_2_W_U 60
`define TUBE_2_H_U 120
`define TUBE_3_W_U 60
`define TUBE_3_H_U 60
`define TUBE_INIT_V_0_U 0
`define TUBE_INIT_V_1_U 0
`define TUBE_INIT_V_2_U 0
`define TUBE_INIT_V_3_U 0
```

### b. 位置 (tube\_h, tube\_v) :

將 rst 訊號接到 fsm 所產生的 screen\_rst, 當進到遊戲模式時，會將輸出的水平位置(tube\_h)調到輸入的 initial\_h, 而垂直位置 (tube\_v) 則依據 updown 所提供的上下訊號來決定。而當每過一個 clk (100hz), 則像素點水平位置會-1 直到 0 並回到 849, 藉此產生水管向左移動的視覺效果

我們為了讓遊戲同時有四組水管在移動，使用了上下共 8 個此模組來紀錄水管位置與尺寸

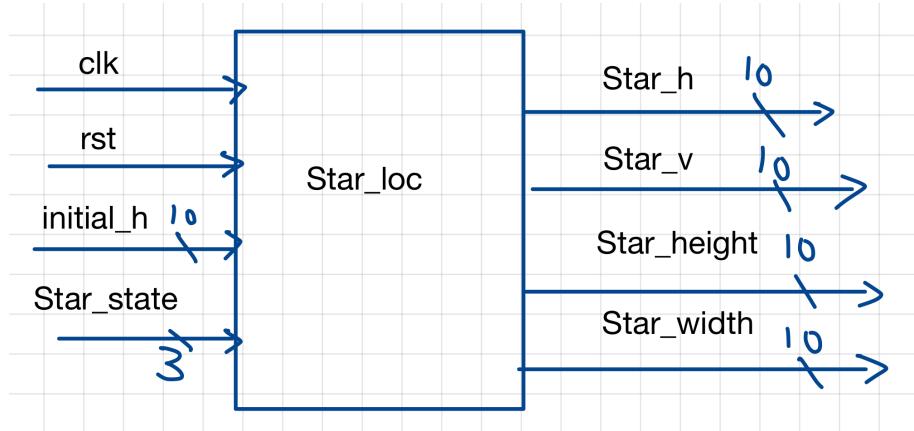
### ii. Character\_loc:



裡面的 if else 判斷比較複雜

- a. 當 TA\_V 為底層邊界且沒有跳動訊號時 next\_position = TA\_V(不動)
- b. 當 TA\_V 小於底層邊界+1 且沒有跳動訊號時, next\_position=TA\_V+1(往下掉落)
- c. 有跳動訊號時, 當 TA\_V 是上層邊界時, next\_position=TA\_V(不動), 否則 next\_position=TA\_V-1(往上移動)
- d. 用一個 flip-flop, 有 clk 進來時, 把 next\_position 值給 TA\_V, 如果 rst 為 1 時, 設定角色的長寬和位置。

iii. star\_loc:



a. 尺寸 (star\_height, star\_width) :

我們在遊戲中設計了一個得分機制，在碰到星星（遊戲中圖片為竹子）會加分，而碰得越久加分則越多，而因為其尺寸皆相同，所以我們在此直接輸出其尺寸數字（30\*30）

b. 位置 (tube\_h, tube\_v) :

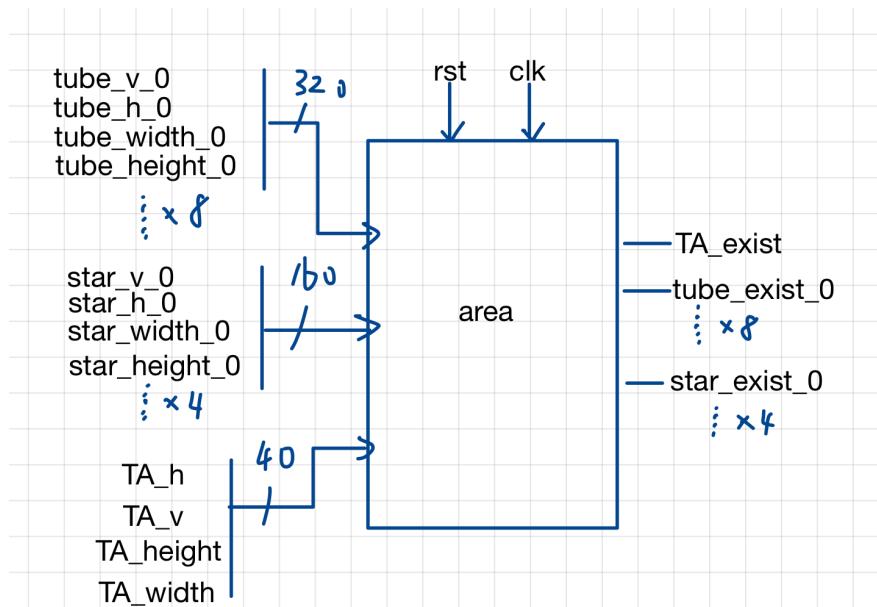
我們預設遊戲中星星（竹子）會有三種不同的位置（上、下、不顯示），我們透過輸入 star\_state 並使用 case 語法來決定圖片所對應的高度(tube\_v):

- i. 上 (3'd1, 3'd6) : 60
- ii. 下 (3'd2, 3'd5) : 300
- iii. 不顯示（其餘）: 500 (超出顯示範圍 480, 所以不會看到)

將 rst 訊號接到 fsm 所產生的 screen\_rst，當進到遊戲模式時，會將輸出的水平位置(star\_h)調到輸入的 initial\_h。而當每過一個 clk (100hz)，則像素點水平位置會-1 直到 0 並回到 849，藉此產生星星（竹子）向左移動的視覺效果

我們為了讓遊戲同時有四組星星（竹子）在移動，使用了共 4 個此模組來紀錄水管位置與尺寸

### (3). Area modules:



輸入目前螢幕欲輸出的像素點 (v\_vnt, h\_vnt) ，此模組便能夠過計算個圖片的長寬及位置來判斷此像素點應該顯示哪一個圖片的像素

因為我們輸出的水平影像範圍從 0~639，當圖片的 h\_cnt 跑到 0 點時會將其重置到 839 (顯示範圍右邊) 的位置，但此時圖片會分成兩半，因此我們需要針對兩種情況來設計邏輯使其正常顯示

#### i. 圖片最左上角位置位於 0~639:

此時圖片並沒有被螢幕分成兩半，所以只要當輸入的 h\_cnt > 圖片的水平位置並小於圖片的水平位置 + 圖片寬以及 v\_cnt >

圖片的垂直位置並小於圖片的垂直位置 + 圖片高，以上兩個條件同時成立，便可輸出 exist = 1，否則 exist = 0

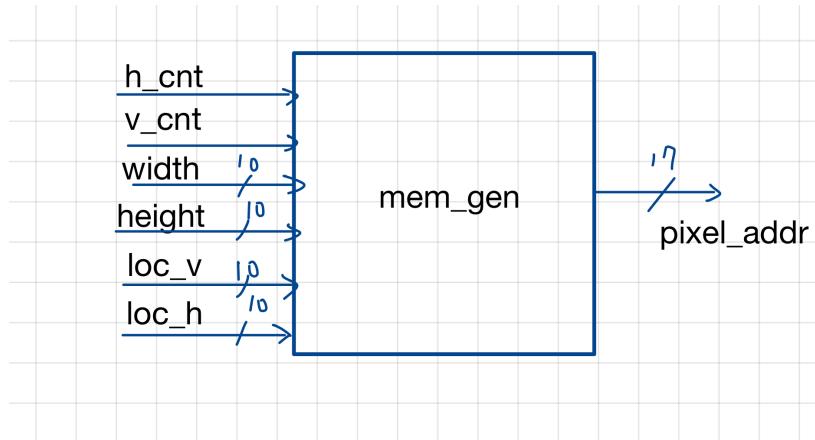
ii. 圖片最左上角位置位於 839-圖片寬~839：

此時圖片被螢幕分成兩半，因此我們的判斷條件改為，當圖片的位置處於 839-圖片寬~839 以及  $h\_cnt < \text{圖片尺寸} - (849 - \text{圖片水平位置} + 1)$ ，再加上  $v\_cnt > \text{圖片的垂直位置並小於圖片的垂直位置} + \text{圖片高}$ ，以上條件同時成立，便可輸出 exist = 1，否則 exist = 0

#### (4). Memory gens module

此模組由多個模組組成，負責輸出所有欲顯示圖片在不同  $h\_cnt$  以及  $v\_cnt$  應該要讀取的記憶體位置

i. Mem\_gen:



我們可以透過 if else 來判斷圖片在輸入  $h\_cnt$  以及  $v\_cnt$  處應該要取用何處的記憶體，但如上一個例子，我們必須處理圖片被分成兩半的問題，所以必須分成兩個情況來做討論

a. 圖片最左上角位置位於 0~639：

此時圖片並沒有被螢幕分成兩半，可以透過以下公式計算記憶體位置：

$$\text{位置} = ((h\_cnt - loc\_h) + width * (v\_cnt - loc\_v)) \% (height * weight)$$

b. 圖片最左上角位置位於 839-圖片寬~839：

此時圖片被螢幕分成兩半，因此我們的公式改為：

$$\text{位置} = (h\_cnt - loc\_h + 850) + width * (v\_cnt - loc\_v)) \% (height * weight)$$

## ii. Mem\_gen\_tube\_U:

因為上方的水管是上下顛倒，但我們採用的方式是將原圖翻轉 180 度，因此須要取用的記憶體位置也必須更改，一樣分成兩個情況來處理，並更改其取用記憶體的公式：

### a. 圖片最左上角位置位於 0~639:

此時圖片並沒有被螢幕分成兩半，可以透過以下公式計算記憶體位置：

$$\text{位置} = ((\text{h\_cnt}-\text{loc\_h}) + \text{width} * (\text{v\_cnt}-\text{loc\_v}+450-\text{height})) \% (27000)$$

### b. 圖片最左上角位置位於 839-圖片寬~839:

此時圖片被螢幕分成兩半，因此我們的公式改為：

$$\text{位置} = ((\text{h\_cnt}-\text{loc\_h}+850) + \text{width} * (\text{v\_cnt}-\text{loc\_v}+450-\text{height})) \% (27000)$$

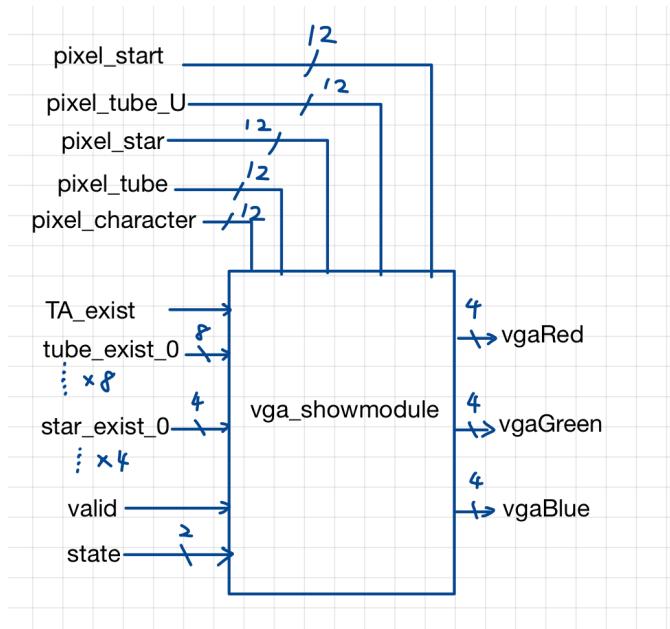
我們總共實作了 8 個 mem\_gen 模組（下方水管\*4, 星星\*4 以及角色\*1）合 4 個 mem\_gen\_U 模組（上方水管\*4）

## (5). Blk\_mem\_gen:

此模組由 vivado 依據不同圖片尺寸生成，會接收從 memory gen module 輸出的 pixel\_addr 訊號，並輸出其對應的記憶體儲存資訊，也就是我們要輸出的色彩訊號，在此專題中，我們共使用了下列不同圖片的 block memory generator

- i. blk\_mem\_gen\_start: 背景圖片，320\*240
- ii. blk\_mem\_gen\_panda: 角色圖片（熊貓）16\*16
- iii. blk\_mem\_gen\_bamboo: 星星圖片（竹子）30\*30
- iv. blk\_mem\_gen\_new\_tubeD: 上方水管圖片 (60\*450)
- v. blk\_mem\_gen\_new\_tubeU: 下方水管圖片 (60\*450)

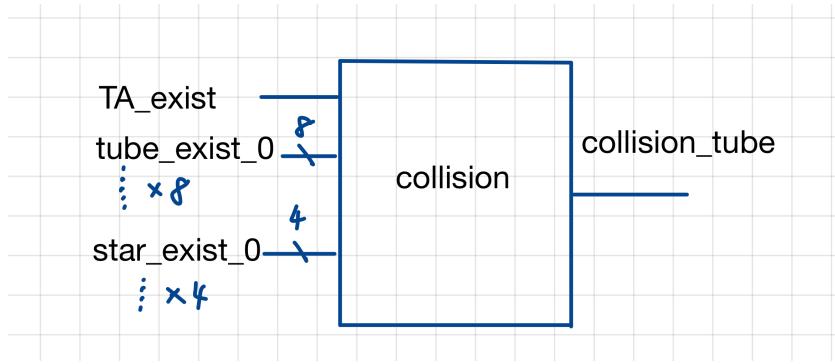
## (6). VGA show module:



此模組透過輸入 exist 訊號來判斷要輸出什麼圖片的訊號，舉例來說，當 star\_exist\_0 為 1 時，那麼我們的影像輸出 RGB 就會接到 pixel\_star，而當同時有兩個或以上 exist 為 1 的話，我們就可以利用 if-else if 來判斷優先順序，利用此種方法，便能夠實作圖片重疊的顯示

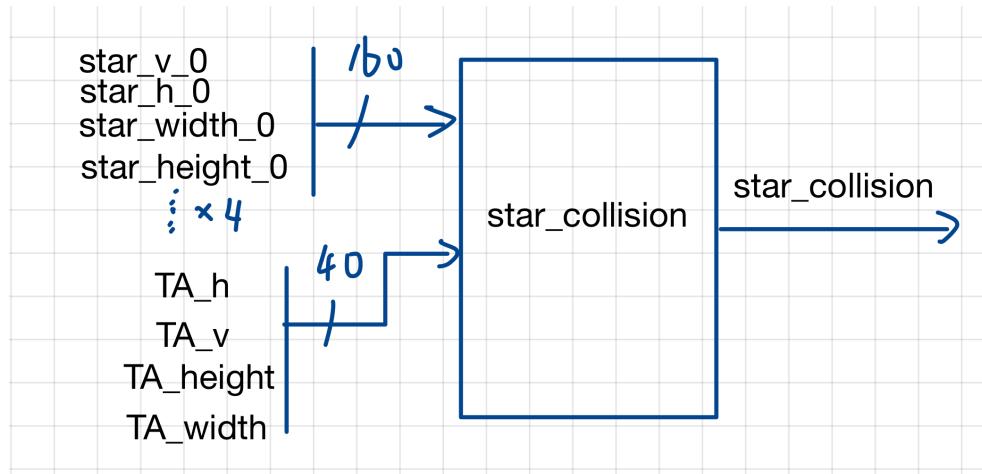
## 2. 圖片碰撞偵測

### (1). Collision module:



為了要能夠判斷遊戲什麼時候結束（主角撞到水管），我們需要一個模組來判斷何時主角與水管產生碰撞，實作的方式便是利用 Area module 所輸出的 exist 訊號，當人物的 exist 訊號與水管的 exist 訊號同時是 1 時，便代表在某一個像素點，人物與水管圖片發生了重疊，也就是碰撞，這時我們便將 collision tube 輸出 1

### (2). Star\_collision:



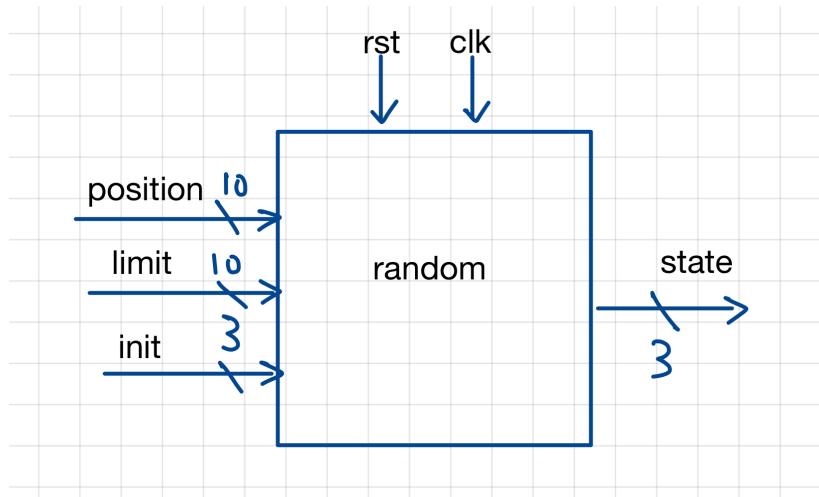
我們的得分機制設定成觸碰越久星星（竹子）則得分會越高，但若是使用上一個碰撞模組的方法來偵測人物與星星的碰撞的話，則會因為 clock 的頻率差而產生分數亂跳的結果，所以我們需要重新實作一個碰撞的模組，我們藉由輸入星星及角色(TA)的尺寸、位置，並判斷是否達成以下四個條件

- i. 人物水平位置 < 星星水平位置 + 星星寬
- ii. 人物水平位置 + 人物寬 > 星星水平位置
- iii. 人物垂直位置 < 星星垂直位置 + 星星高
- iv. 人物水平位置 + 人物高 > 星星垂直位置

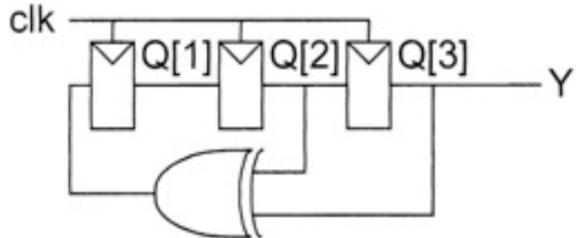
若全達成則 star\_collision 輸出一，其餘狀況則輸出 0

### 3. 隨機生成模組：

#### (1). Random module:



為了讓關卡有變化性，我們設計了 random module 使四組水管以及星星出現的位置及尺寸每次都會產生變化，我們使用在 lab 9 學到的 Linear Feedback Shift Register 來實作

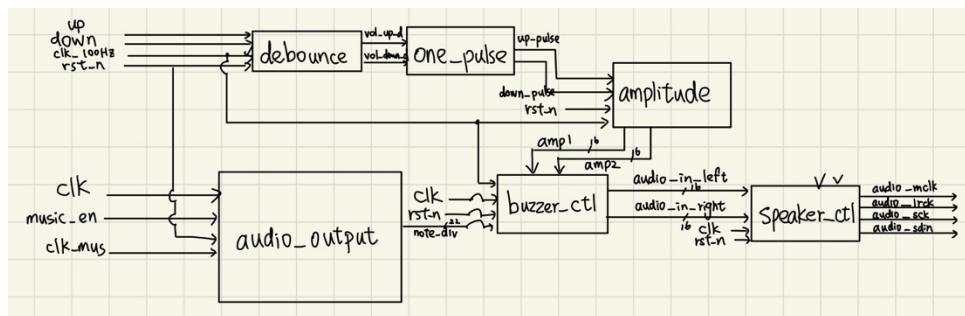


而我們觸發 random module 的條件則是當其輸入的 position 位置輸入的 limit 數值，也就是當圖片消失在左方螢幕時，此時 random module 就會被觸發，更新 star 以及 tube 的 state，其他 module 就會根據此 state，改變水管及星星的尺寸以及顯示的位置，而我們為了讓一開始的初始狀態不同，增加了輸入的 init 訊號，其功能有點類似 random seed，透過改變 module 的初始 register 設定值，來每個 module 都輸出不同的結果

我們為每個水管圖片（上、下）以及星星都建議一個 instance，所以最後共會生成 12 個 random module

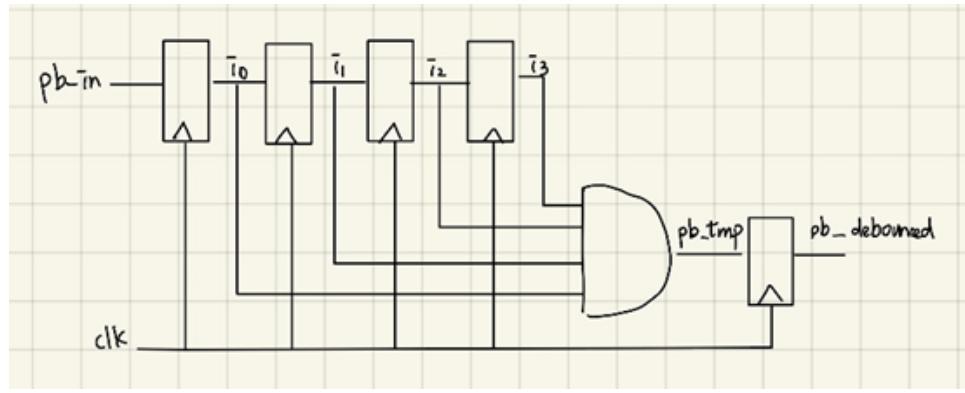
#### 4. 音樂相關模組:

##### (1). Music\_generate



###### i. debounce:

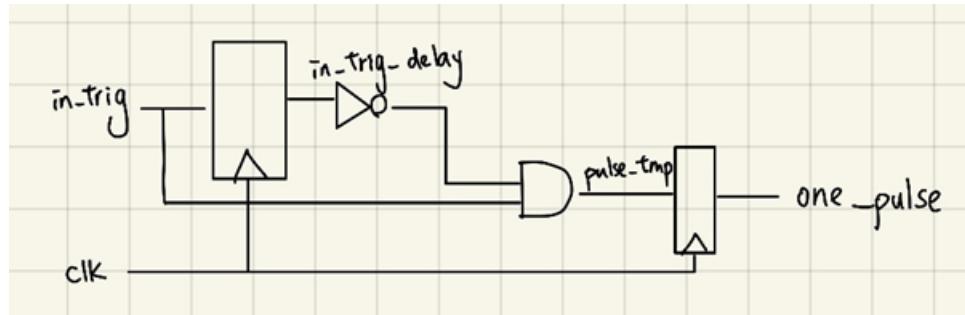
將每個按鈕訊號都經過 debounce 處理，可以穩定按鈕的訊號。



push bottom 會產生 bouncing 的現象，因此將 push bottom 的輸入通過 4 個 flip flop 後，再將每個 flip flop 的輸出(i0~i3)通過 and gate，可以得到穩定的訊號值(pb\_debounced)

### ii. one pulse:

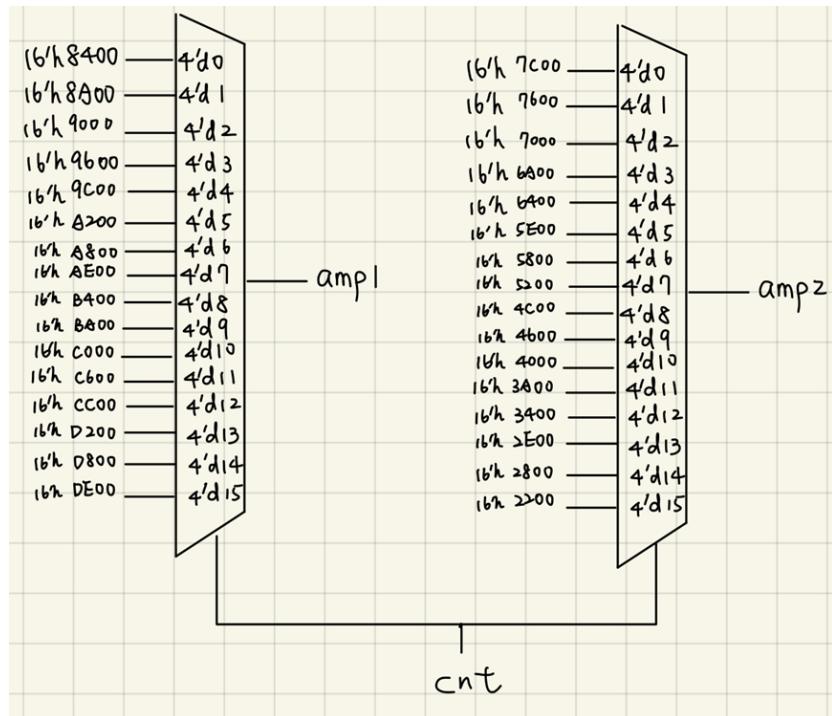
處理:因為每按一次音量鍵音量要上升一格或是下降一格，將控制音量的按鈕經過 one pulse 處理，保證按一次只有一個 clk，再將訊號傳給 amplitude 當作輸入。



因為每一次按下 push bottom 的時間長短都不一樣，如果只想在每一個 clock 裡有輸入訊號，那就要將輸入 in\_trig 延遲一個 clock 得到 in\_trig\_delay，在和下一次輸入的 in\_trig 做 and gate，就可以得到只有在一個 clock 裡有 positive signal 的輸出

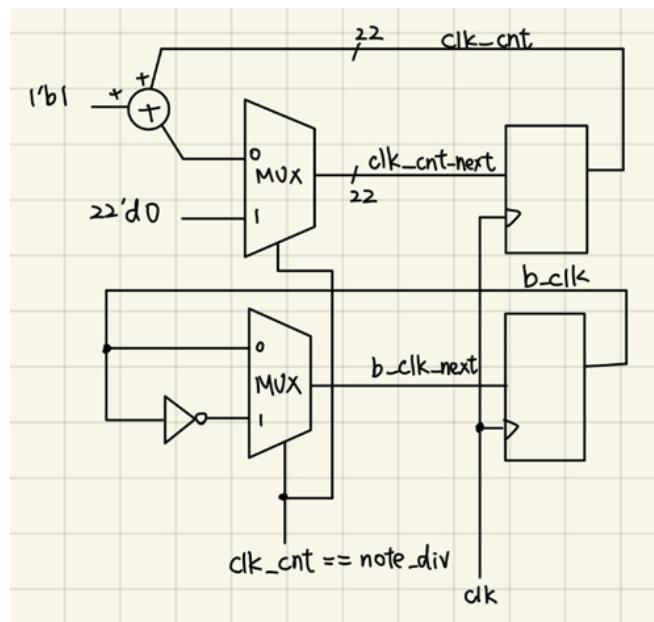
### iii. 振幅控制(amplitude):

造一個上限為 15，下限為 0 的 counter，當 one pulse 的 vol\_up 訊號來時，counter 加 1；當 one pulse 的 vol\_down 訊號來時，counter 減 1。counter 的值選擇目前的音量大小(振幅大小)，再將對應到的振幅輸出給 buzzer control 當作輸入。

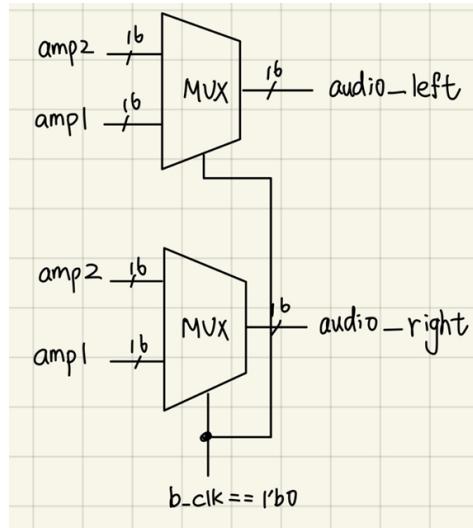


Counter 預設值為 8，所以預設負向振幅(amp1)為 16 進位的 B400；正向振幅 (amp2)為 16 進位的 4C00。counter 每加 1，振幅就加上 16 進位的 0600；counter 每減 1，振幅就減掉 16 進位的 0600(最後發現變化太小有再修改振幅)。如果 counter 小於 9 時，value1 會是 0，value 會是 counter 值加一，如果 value0 值大於或等於 9，Value1 會是 1、Value0 會等於 counter 的值減 9。

#### iv. buzzer control:



透過多工器選擇到的 note\_div 把 100MHz 的頻率除頻出對應到的音高頻率，並在這個頻率之下加上從 amplitude 得到的振幅大小，形成左右聲道各 16 bit 的 parallel data 再傳給 speaker control

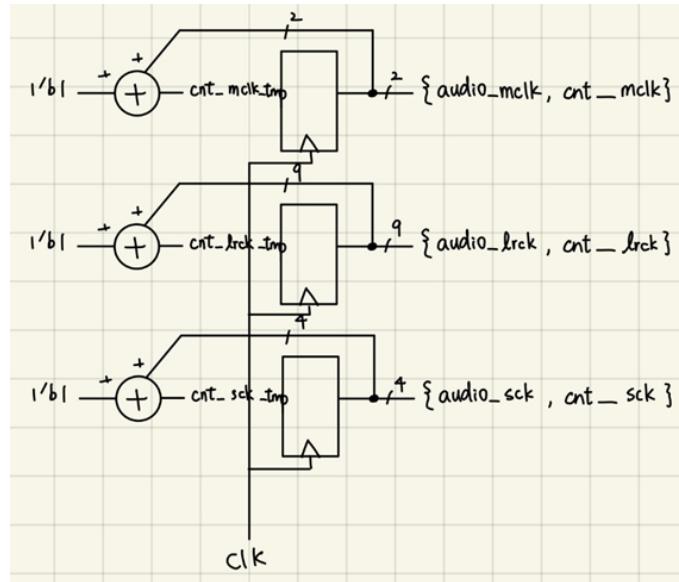


當  $b\_clk = 0$  時，輸入 amp1，也就是負向振幅；當  $b\_clk = 1$  時，輸入 amp2，也就是正向振幅，如此可以得到可以調振幅的音高頻率。

#### v. speaker control:

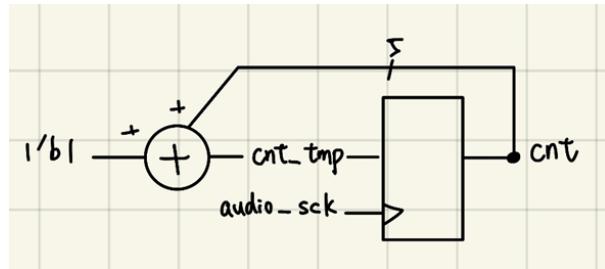
分成除頻和 parallel to serial 兩個功能，前者將 100MHz 的 crystal clock 除頻成 25MHz 的 main clock(audio\_mclk)、25MHz/128 的 sample rate clock (audio\_lrck) 以及 25MHz/4 的 serial clock (audio\_sck)；後者透過不同頻率將 buzzer control 輸出的 parallel data 轉換成 serial 的 data 並輸出。

- 除頻：輸出 100MHz/4 的 main clock(audio\_mclk)、100MHz/512 的 sample rate clock (audio\_lrck) 以及 100MHz/16 的 serial clock (audio\_sck)。



b. 上數器:

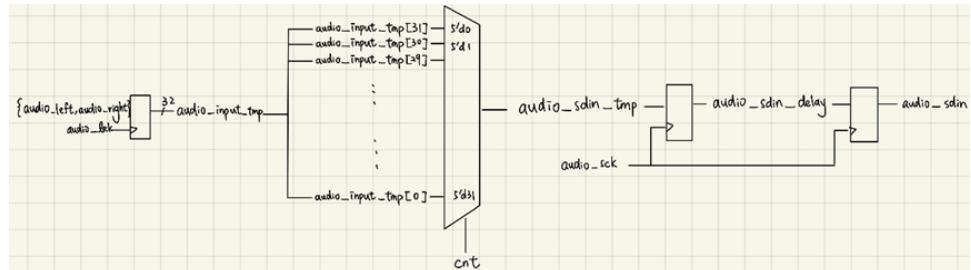
cnt 的值用來當作之後 MUX 的控制訊號。



這是一個上限為 32 的上數器，以 serial clock 的頻率讓 counter 每次加 1。

c. MUX:

透過不同頻率將 parallel 的資料轉換成 serial 的資料輸出。



在 sample clock(audio\_lrck)來的時候，將 32bits 的 audio input tmp 將左右聲道各 16bits 的 parallel data 做結合成 32bits 並 儲存起來，因為我們取用比 sample rate clock 還要快 32 倍的 serial clock 來數 5bits 的 cnt，因此 cnt 的值就可以用來選擇 MUX 選哪一 bits 要輸出，並且是在每一次 serial clock 來的時候將 32bits 資料 1bit 的輸出，因為頻率相差 32 倍，所以

sample rate clock=一次就等於 32 次的 serial clock，可以剛好輸出 32bits 的資料。

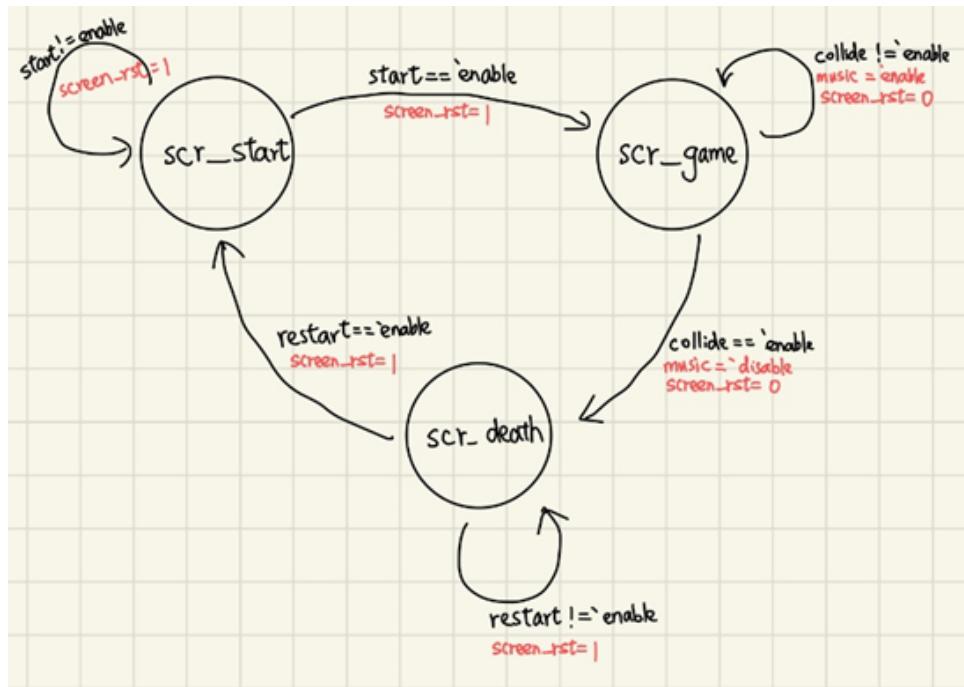
右方多加一個 D-Flip-Flop 是為了要有 serial clock delay 的效果。

vi. audio output:

裡面已經有 80 個 register 儲存好的音樂頻率，然後透過 clk\_mus(4hz)來將資料傳給 note\_div。

## 5. Finite state machine

FSM: (start、 collide、 restart 為控制 state 訊號)

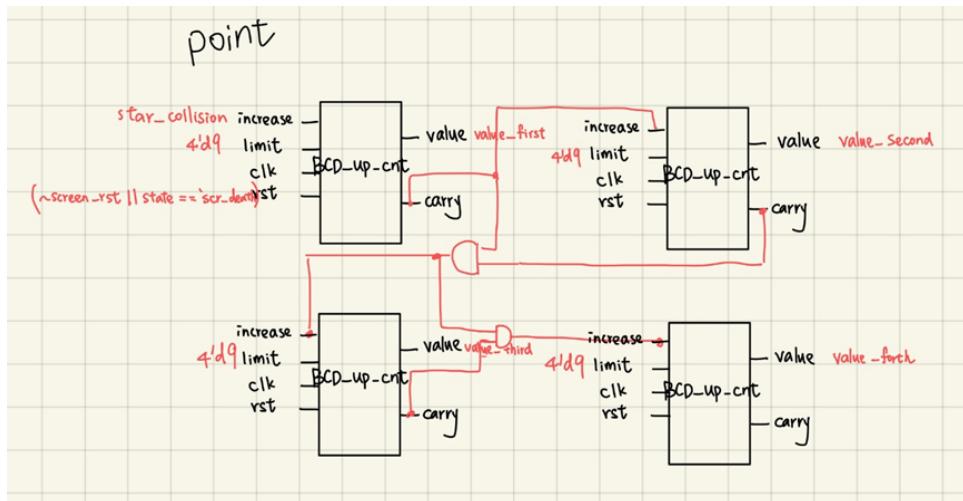


重置的狀態為 scr\_start(開始頁面)

1. scr\_start→scr\_game，當 start==1'b1 時，下一個 state 為 scr\_game，並輸出 screen\_rst==1'b1 訊號。
2. scr\_start→scr\_start，當 start!=1'b1 時，下一個 state 仍為 scr\_start，並輸出 screen\_rst==1'b1 訊號。
3. scr\_game→scr\_death，當 collide==1'b1 時，下一個 state 為 scr\_death，並輸出 screen\_rst==1'b0 訊號。
4. scr\_game→scr\_game，當 collide!=1'b1 時，下一個 state 仍為 scr\_game，並輸出 screen\_rst==1'b0 訊號。

5.  $\text{scr\_death} \rightarrow \text{scr\_start}$ , 當  $\text{restart} == 1'b1$  時, 下一個 state 為  $\text{scr\_start}$ ,  
並輸出  $\text{screen\_rst} == 1'b1$  訊號。

## 6. Point



總共有 4 個 counter，每個都長得一樣，不過差在 increase 的 enable，像是第一個的 increase 是當有碰撞到星星訊號時才會變成 1，接下來都會是與前幾個 carry 做 and 來判斷有沒有需要借位。

四個 counter 都會輸出 1 個值，再將這四個值帶到 scan\_ctl(與實作 lab 相同)，產生 ssd\_in 的訊號再進入 display 產生 seg 做最後七段顯示器的輸入。

### I/O Pins Assignment:

I/O	Assignment	I/O	Assignment
clk	W5	PS2_DATA	B17
rst	U18	PS2_CLK	C17
start	W19	audio_mclk	A14
restart	T17	audio_lrck	A16
pb_up	T18	audio_sck	B15
pb_down	U17	audio_sdin	B16
hsync	P19	segs [0]	V7
vsync	R19	segs [1]	U6
vgaRed [0]	G19	segs [2]	V5

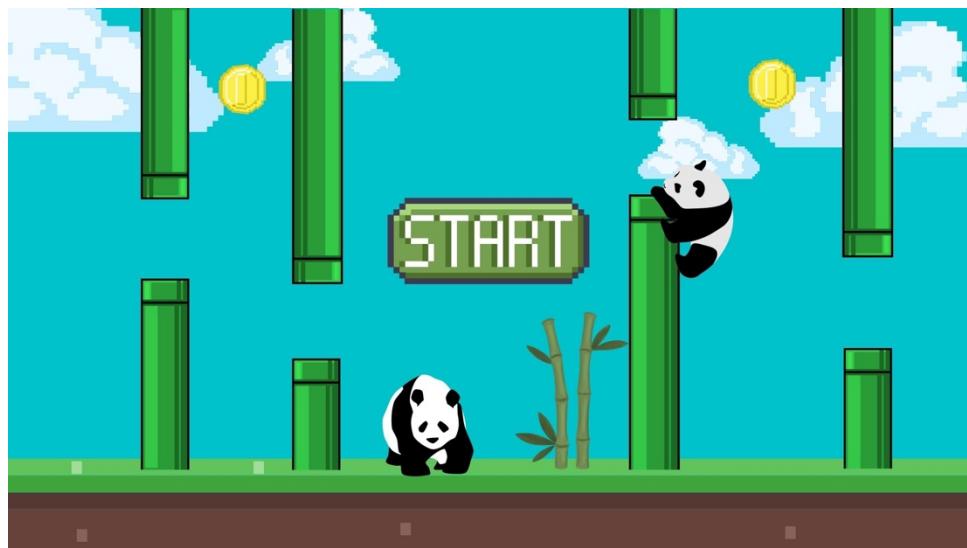
vgaRed [1]	H19	segs [3]	U5
vgaRed [2]	J19	segs [4]	V8
vgaRed [3]	N19	segs [5]	U8
vgaGreen [0]	J17	segs [6]	W6
vgaGreen [1]	H17	segs [7]	W7
vgaGreen [2]	G17	ssd_ctl [0]	U2
vgaGreen [3]	D17	ssd_ctl [1]	U4
vgaBlue [0]	N18	ssd_ctl [2]	V4
vgaBlue [1]	L18	ssd_ctl [3]	W4
vgaBlue [2]	K18		
vgaBlue [3]	J18		

## DISCUSSION:

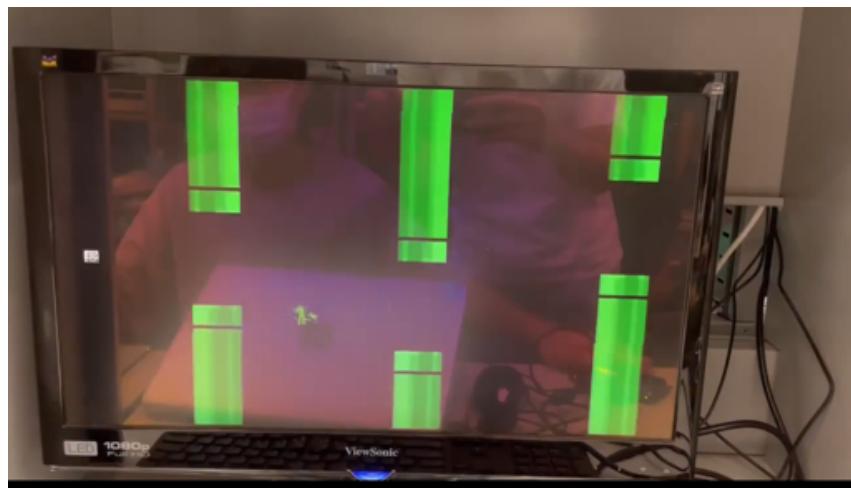
### FPGA results:

初始畫面:

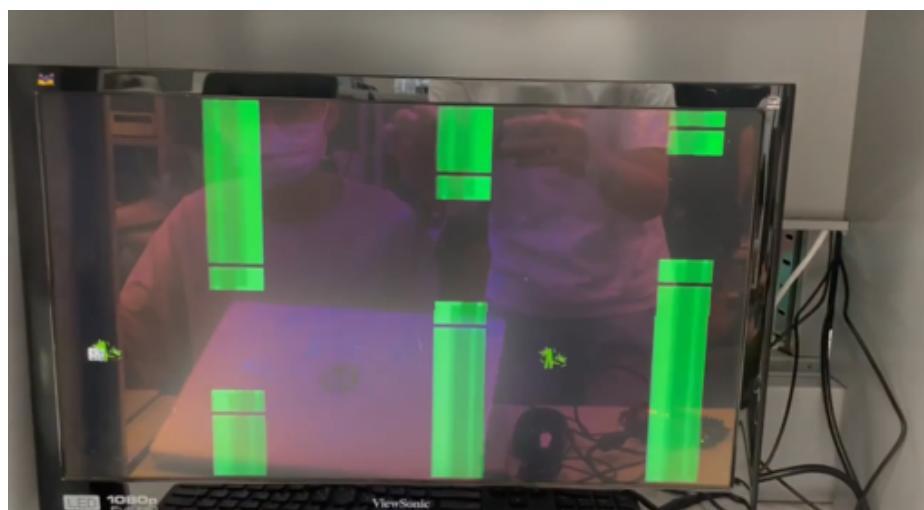
按下 START 鍵後會進入遊戲畫面



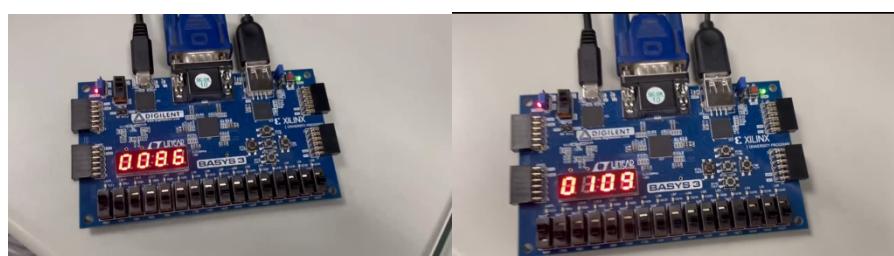
遊戲畫面:



角色會不斷向下掉落，直到按下鍵盤上的方向鍵"上"後，角色會往上移動，按越久上升越久。



當角色觸碰到星星(竹子)時，會在七段顯示器上加分，如果觸碰越久加的分數也越多。



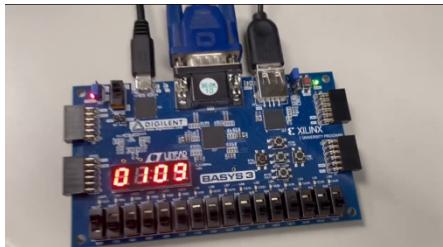
死亡畫面：



當角色觸碰到水管後，畫面會黑掉代表死亡，七段顯示器上會顯示遊戲的分數。如果想要重新開始遊戲，就要按下 restart 的 button。

如果想要再次開始遊戲，按下 restart 按鍵即可。

遊戲分數：



## Things went wrong and debug:

### 1. 決定角色跳動的形式：

一開始我本來是打算做角色的跳動是以 onepulse 的形式，所以翻閱了講義查看需要用哪個訊號輸出來做，一開始我想用 key\_valid 和 last\_change 來做，我一開始以為 key\_valid 訊號只有在按下與放開時會有訊號，但我將 key\_valid 的訊號輸出在 LED 上發現，按下去鍵盤按鍵後會產生很多個訊號，不知道是雜訊還是甚麼東西，思考片刻後我想到能加一個 counter 來計算 key\_valid 的訊號並將其中一個儲存的 bit 當輸出與 last\_change 為 9'h75 做 jump 的訊號，其實挺成功的，不過實測後發現有時候常按時會出現兩個

訊號，於是我又再回去翻講義發現有一個 key\_down 的訊號是儲存 address 的，於是我把此訊號想拿去做 one\_pulse，最後發現會出問題，沒辦法產出輸出，最後只好將就直接將 key\_down 訊號拿去當 jump 的輸入，代表長按時角色會一直向跳動，不過實測後發現，這樣設計比較好通關。

## 2. 音樂合成和聲音大小聲：

在音樂的設計上也遇到不少問題，原本是打算自己合成一個波來輸出聲音，因為相對 0 1 形成的方波，合成波產生的音訊會比較柔和和清晰，翻了很多資料然後還有做測試，但總是會有雜音出現，音質會變得很差，可能是因為網路上的資料音檔有存到音符撥放的時間長短，但要做到這種方法，必須要讀取 MIDI 文件還要分離軌道很複雜，所以最後就用單音撥放。在設計聲音大小的時候也發生問題，因為調整聲音大小其實與之前做過的 lab 很像，不過將 code 搬移近來稍作修改後，發現聲音大小其實沒有改變的特別明顯，這過程中我們還以為是按鈕輸入訊號沒有傳送進去 module 裡面，不過將訊號顯示在 led 燈上後才發現其實有輸進去，最後我想到可以條整看看振幅，發現讓振幅變得很誇張就有明顯的音樂大小聲改變，最後才解決問題！

## 3. 水管亂數更新問題：

在實作水管移動時，為了要讓水管同時在螢幕最左邊以及最右邊同時呈現，還要實作隨機亂數的功能，為此苦惱了很久，後來才意識到所謂的螢幕尺寸只不過是顯示的尺寸而已，超過螢幕尺寸的距離，只要沒有超出 register 的大小，其實都是可以使用的，於是以此為基礎，讓水管消失時回道螢幕範圍外，在看不見的地方去生成新的水管，如此一來便能夠實現畫面了流暢更新

## 4. 顯示畫面破圖：

在顯示螢幕畫面時，常常都會發現圖片有破圖的情形，而且還常常發生有噪點或是其他詭異畫面，在運行一段時間後發生，後來發現在 memory 的存取時都要非常小心，所有的存取最後都要取 mod 以防碰到不該碰的位置，一旦不小心沒注意到，顯示的畫面就可能變得非常奇怪

## 5. 版本控制問題：

平常在做實驗時，因為實作的內容相對較少，根本不用考慮到版本控制的問題，但在做期末專題時情況就不太一樣了，有時本來好好的程式，一旦不小心碰到電腦誤刪了幾行，要在幾千行中找到錯誤實在是非常困難，何況 verilog 不像平常在寫 c 或 python 能過快速的印出 debug 的資訊，而需要等待漫長的合成時間，所以在大型專案 debug 的又會更加困難，於是我們在其中還去學了 git 的版本控制系統，配合 github 來讓每階段的版本都能妥善的儲存，也增快了 merge code 跟找新舊版本差異的時間

## Conclusion of this final project:

### 1. 110030031 黃玄彰：

儘管以往曾經做過資工程設的大型專案，但 Verilog 帶來的感覺實在差很多，不像是一般的程式語言，在編寫的過程中我們不能邊寫邊改，而是要在一開始討論的過程中，就很明確的想好每個 module 的功能以及所需的 input/output，在實作的過程中也注意不同 module 間的 clk 是否會有頻率差的問題，以及有些寫法可能會導致合成不出電路（如：用非除頻產生的訊號當 clk），而在這次的專題中，我們也運用了許多在學期中所學到的技巧，像是如何產生亂數以及如何壓縮圖片的空間等。最後的 project 就像是幫整個課程做了一次總複習，從按鍵 debounce、onepulse，到聲音的訊號產生、影像的訊號輸出以及 keyboard 的按鍵判斷，都讓我對於學習的內容理解的更透徹，儘管在實作過程中常常會出現令人匪夷所思的 bug，像是明明一樣的 code 但 generate 出的結果竟然會不一樣，或是同一張照片產生的 IP 有時候會突然有一個有噪點等奇奇怪怪的問題，但這些 bug 也讓我們對於大型專案的除錯及應變有了更多的經驗。總而言之，雖然專題的實作非常折磨人，debug 更是常常弄到天亮，我依然從其中獲取了許多寶貴的知識。

### 2. 110011222 陳立珩：

Verilog 與 c 差很多，因為很多時候其實有想法了，但是沒辦法直接將想到的直觀打成 code 做表達，因為包含接線和定義都有特定的規則遵守，所以

其實在這次專案中，我們花了很多時間在找錯誤和 debug，尤其是加了圖片之後，整個跑合成都要跑超級久，所以在測試時，我們都先把圖片拿掉，改成純色塊，這樣省去很多合成時間。其實我們做的專案就是將這學期在 lab 中學到的各個功能做統合，從最基本的按鍵和七段顯示器，到後面很複雜的鍵盤和 VGA 螢幕顯示，我們終於能有一個機會將所學到的東西實作在一個題目裡，原本在做 lab 的時候其實不會覺得很難但當全部都丟在一起之後，才發現之前其實有很多東西沒有理解透徹，所以又回去把講義再看了一遍當作複習，雖然在實作過程遇到很多的問題，但是當遇到問題時我們處理方法通常是將線接出去到 LED 上看 output，再依次處理問題。本來我們想在遊戲畫面加上背景，死亡畫面也想用 death 畫面顯示，不過最後因為記憶體不夠，所以沒辦法加上去，稍微可惜。幸好每一次的 lab 都有自己慢慢實作，這學期真的從 debug 和 coding 中學到很多知識，這些知識未來將會用在我的大三專題上，希望未來一切都能夠順利！

## REFERENCES:

1. Logic design handout of the last semester, Chapters 4, 5, 6  
It helps me review the BCD counter of this lab.
2. [https://blog.csdn.net/qq\\_43686057/article/details/123813708](https://blog.csdn.net/qq_43686057/article/details/123813708) -verilog testbench 自動生成  
幫助我快速生成 instance 以及 testbench 的 code

## 分工：

- 1.影像呈現相關模組: 110030031 黃玄彰
- 2.圖片碰撞偵測: 110030031 黃玄彰
- 3.隨機生成模組: 110030031 黃玄彰
- 4.音樂相關模組: 110011222 陳立珩
- 5.FSM: 110011222 陳立珩
- 6.point: 110011222 陳立珩