Logic Design Lab Final Project: Flappy Bird

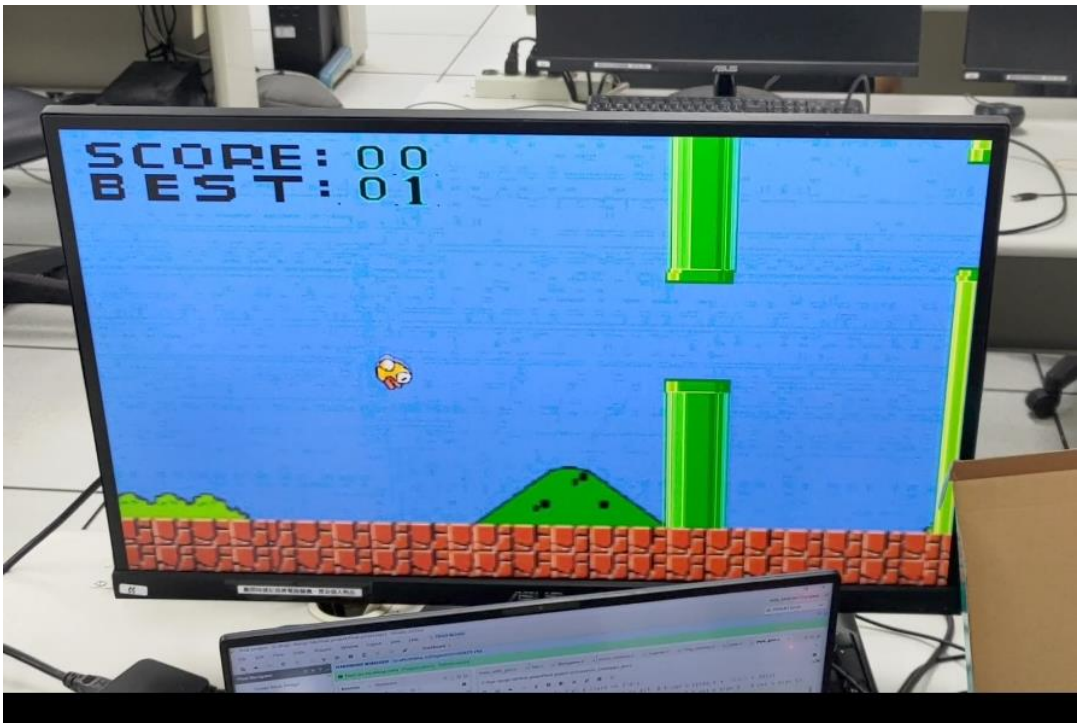
Member

EE24 清 109030003 吳璨霖

EE25 清 110030039 張育碩

Project name

Flappy Bird



Brief Introduction

這次的 project 是模仿前五年左右蠻有名的遊戲 Flappy Brid。一開始我們會先進到待機畫面，之後一按下空白鍵就會開始遊戲，一按空白鍵，鳥就會往上飛，如果停止連點空白鍵，鳥就會往下墜，當我們碰到地板，天花板，水管，遊戲就會結束。遊戲的目標是要設法讓鳥通過更多的水管，獲得更高的分數。

此外，為了增加遊戲的互動性，我們在不同的遊戲狀態下產生不同的背景音樂，希望能使這個遊戲更有活力。並且我們也設立了記分板，讓玩家可以知道目前的分數以及歷史的最佳成績。

User Interface

Input	Function	Output	Function
Button UP	Volume Up	**7-segment digit 0**	Scrolling BGM name
Button DW	Volume Down	**7-segment digit 1**	Scrolling BGM name
Keyboard Space	Bird Flap	**7-segment digit 2**	Scrolling BGM name
Keyboard Enter	Pause/ Restart	**7-segment digit 3**	Scrolling BGM name
DIP SW14	BGM on/off	**LCD Display**	Gameplay
DIP SW15	reset	**Speaker**	Background Music

Preface

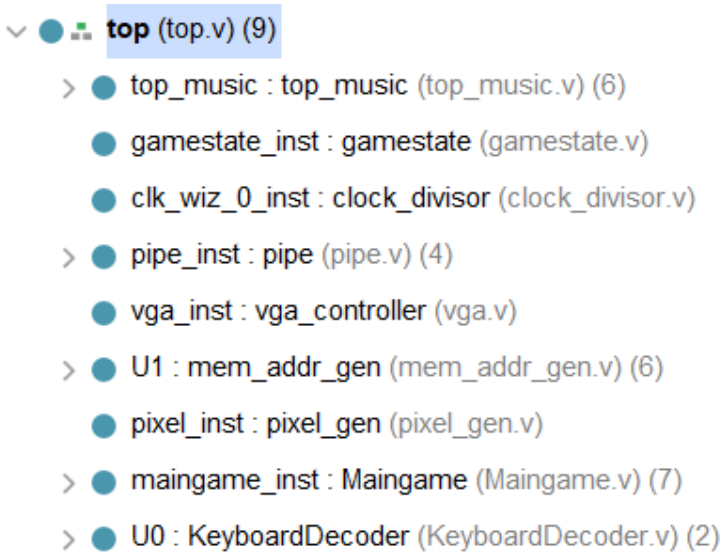
我們把這次的 project 分成兩個部分，第一個部分是 VGA 顯示的部分，包含背景畫面的生成、鳥是如何飛行、如何生成水管並且能在畫面中向左移動，這一部分由電機 24 吳燦霖負責。

第二個部分是有關音效方面、整合整個遊戲以及報告，這個部分是由電機 25 張育碩負責。因此在報告中我們會主要針對個別負責的項目撰寫。雖然是分工合作，不過我們兩個人也都了解另外一個人負責的項目，只是比重不同而已。

Content (1/2)

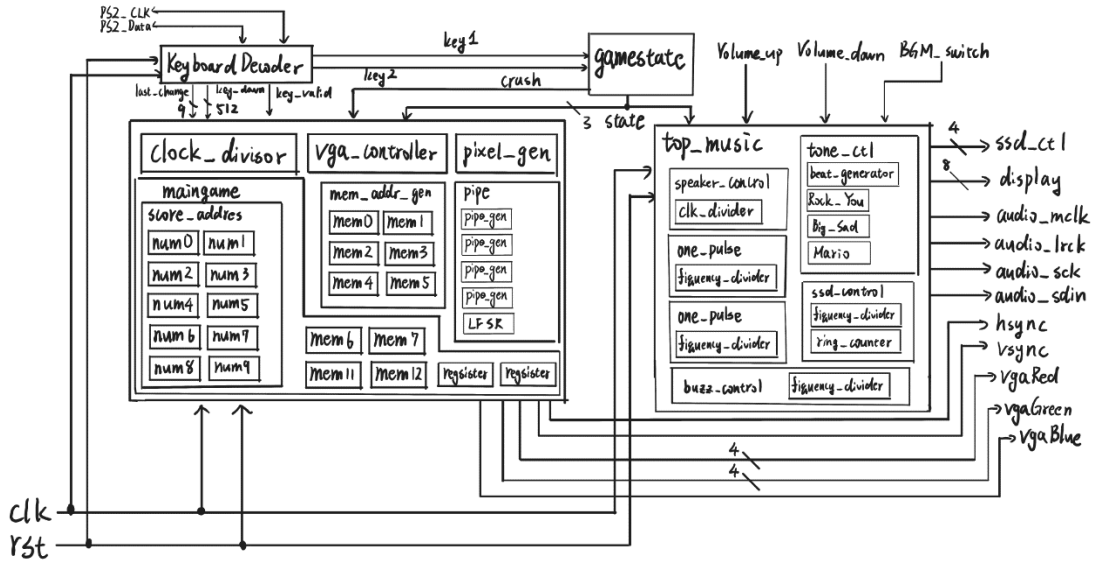
Content (2/2)

Top Module




```
23  module top(
24      input clk,
25      input rst,
26      inout PS2_CLK,
27      inout PS2_DATA,
28      input volume_up,
29      input volume_down,
30      input BGM_sitch,
31      output [3:0] vgaRed,
32      output [3:0] vgaGreen,
33      output [3:0] vgaBlue,
34      output hsync,
35      output vsync,
36      output [3:0]ssd_ctl,
37      output [7:0]display,
38      output audio_mclk,
39      output audio_lrck,
40      output audio_sck,
41      output audio_sdin
42      );
```

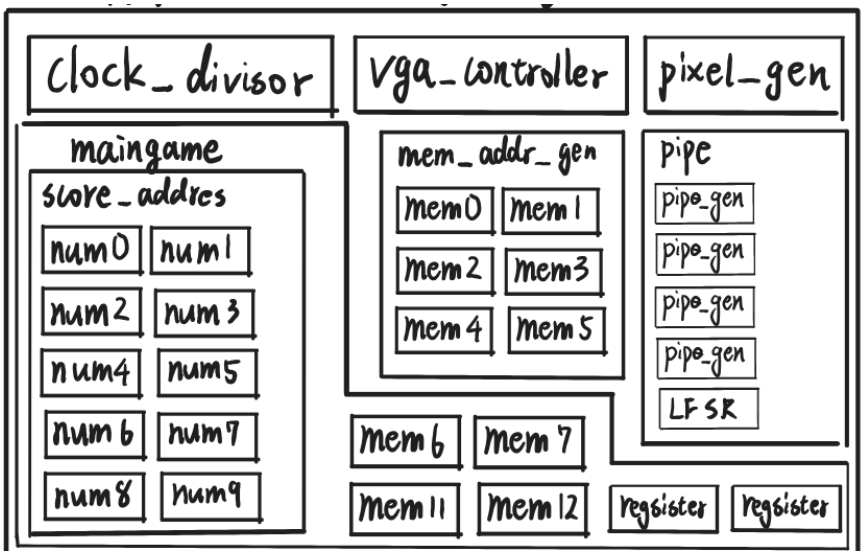
Block Diagram (including main blocks)



I/O pins assignment

display[7]	display[6]	display[5]	display[4]	display[3]	display[2]
W7	W6	U8	V8	U5	V5
display[1]	display[0]	ssd_ctl[3]	ssd_ctl[2]	ssd_ctl[1]	ssd_ctl[0]
U7	V7	W4	V4	U7	V7
vgaBlue[3]	vgaBlue[2]	vgaBlue[1]	vgaBlue[0]	vgaGreen[3]	vgaGreen[2]
J18	K18	L18	N18	D17	G17
vgaGreen[1]	vgaGreen[0]	vgaRed[3]	vgaRed[2]	vgaRed[1]	vgaRed[0]
H17	J17	N19	J19	H19	G19
audio_lrck	audio_mclk	audio_sck	audio_sdin	BGM_sitch	hsync
A16	A14	B15	B16	V16	P19
vsync	PS2_CLK	PS2_DATA	volume_down	volume_up	clk
R19	C17	B17	U17	T18	W5
rst					
V17					

Main game and VGA control



此遊戲基本上遊戲本體都是在 vga 的 block 中去操作以及運算，下面就開始介紹我 VGA 部分用到哪些 Block。首先先給各位看 top module 中主要分成哪些 block，如下圖。

```
74  gamestate gamestate_inst(
75  .clk(clk),
76  .rst(rst),
77  .key1(key_down[9'h29]),
78  .key2(key_down[9'h5A]),
79  .crash(crash),
80  .state(state)
81  );
82
83  clock_divisor clk_wiz_0_inst(
84   .clk(clk),
85   .clk1(clk_25MHz),
86   .clk21(clk_21)
87  );
88
89  pipe pipe_inst(
90  .clk_21(clk_21),
91  .clk(clk),
92  .rst(rst),
93  .state(state),
94  .h_cnt(h_cnt),
95  .v_cnt(v_cnt),
96  .pixel_pipebody(pixel_pipebody),
97  .pipebody_valid(pipebody_valid),
98  .pixel_pipemouth(pixel_pipemouth),
99  .pipemouth_valid(pipemouth_valid),
100 .pixel_pipebase(pixel_pipebase),
101 .pipebase_valid(pipebase_valid),
102 .score(score)
103 );
```

```
105 vga_controller   vga_inst(
106   .pclk(clk_25MHz),
107   .reset(rst),
108   .hsync(hsync),
109   .vsync(vsync),
110   .valid(valid),
111   .h_cnt(h_cnt),
112   .v_cnt(v_cnt)
113 );
114
115 mem_addr_gen U1(
116 .clk_21(clk_21),
117 .clk(clk),
118 .rst(rst),
119 .state(state),
120 .pixel_pipemouth(pixel_pipemouth),
121 .pipemouth_valid(pipemouth_valid),
122 .pixel_pipebase(pixel_pipebase),
123 .pipebase_valid(pipebase_valid),
124 .pipebody_valid(pipebody_valid),
125 .pixel_pipebody(pixel_pipebody),
126 .h_cnt(h_cnt),
127 .v_cnt(v_cnt),
128 .key(key_down[9'h29]),
129 .crash(crash),
130 .pixel_bg(pixel_bg),
131 .pixel_bird(pixel_bird),
132 .bird_valid(bird_valid)
133 );
134
```

```
135  pixel_gen pixel_inst(
136  .valid(valid),
137  .bird_valid(bird_valid),
138  .start_valid(start_valid),
139  .gameover_valid(gameover_valid),
140  .pause_valid(pause_valid),
141  .score_valid(score_valid),
142  .b0_valid(b0_valid),
143  .b1_valid(b1_valid),
144  .s0_valid(s0_valid),
145  .s1_valid(s1_valid),
146  .pipebody_valid(pipebody_valid),
147  .pixel_pipemouth(pixel_pipemouth),
148  .pipemouth_valid(pipemouth_valid),
149  .pixel_pipebase(pixel_pipebase),
150  .pipebase_valid(pipebase_valid),
151  .pixel_bg(pixel_bg),
152  .pixel_bird(pixel_bird),
153  .pixel_start(pixel_start),
154  .pixel_gameover(pixel_gameover),
155  .pixel_pause(pixel_pause),
156  .pixel_score(pixel_score),
157  .pixel_num(pixel_num),
158  .pixel_pipebody(pixel_pipebody),
159  .vgaRed(vgaRed),
160  .vgaGreen(vgaGreen),
161  .vgaBlue(vgaBlue)
162  );
163
```

```
164  Maingame maingame_inst(
165  .clk(clk),
166  .rst(rst),
167  .state(state),
168  .v_cnt(v_cnt),
169  .h_cnt(h_cnt),
170  .score(score),
171  .start_valid(start_valid),
172  .gameover_valid(gameover_valid),
173  .pause_valid(pause_valid),
174  .score_valid(score_valid),
175  .s1_valid(s1_valid),
176  .s0_valid(s0_valid),
177  .b1_valid(b1_valid),
178  .b0_valid(b0_valid),
179  .pixel_start(pixel_start),
180  .pixel_gameover(pixel_gameover),
181  .pixel_pause(pixel_pause),
182  .pixel_score(pixel_score),
183  .pixel_num(pixel_num)
184      );
185
186  KeyboardDecoder U0(
187    .key_down(key_down),
188    .last_change(last_change),
189    .key_valid(key_valid),
190    .PS2_DATA(PS2_DATA),
191    .PS2_CLK(PS2_CLK),
192    .rst(rst),
193    .clk(clk)
194  );
195
```

Clock divisor：

此 module 主要用於製造 25MHz 的 clk_25MHz 給 vga 這個 module 使用，以及產生一個將 globle clock 除頻 21 次的 clk_21 作為遊戲中的時間單位，運作模式基本上與前面的 lab 相同，使用 counter 的方式來除頻，除幾個 2 的次方就是看 output 取到的 bit 是第幾個 bit 來決定。

vga_controller：

此 module 用來定義顯示的範圍，根據當前數到的 v_cnt 以及 h_cnt 來判斷現在準備要顯示的 bit 是否為可以顯示的位置，根據其內部邏輯判斷的結果來輸出 valid 訊號給外面的 block，此 module 為前面 lab9 老師給的 reference code 中的其中之一，因此也不做細部的介紹。

KeyboardDecoder：

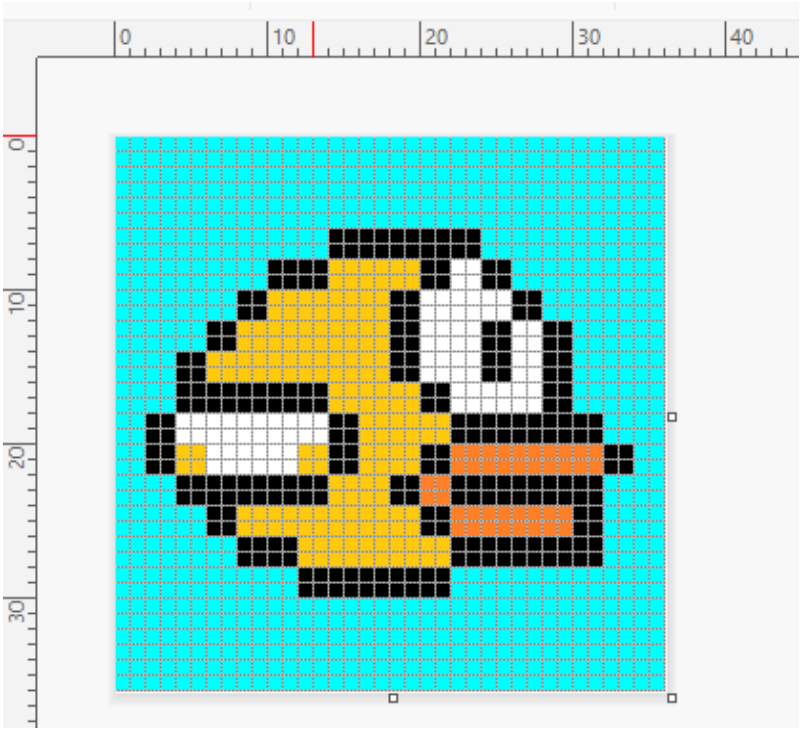
此 module 用來將鍵盤輸入給 FPGA 的訊號 PS2_DATA 以及 PS2_CLK 根據按下的鍵盤按鍵轉為輸出 last_change 以及 key_down 等訊號，我們需要這個 module 的 last_change 以及 key_down 來取得目前按下的鍵盤按鍵資訊，此 module 為前面 lab8 老師給的 reference code 中的其中之一，因此也不做細部的介紹。

pixel_gen：

```
always @*
if (valid)
    if(score_valid & pixel_score !=12'hOFF)
        {vgaRed, vgaGreen, vgaBlue} = pixel_score;
    else if(b0_valid & pixel_num !=12'hOFF)
        {vgaRed, vgaGreen, vgaBlue} = pixel_num;
    else if(b1_valid & pixel_num !=12'hOFF)
        {vgaRed, vgaGreen, vgaBlue} = pixel_num;
    else if(s0_valid & pixel_num !=12'hOFF)
        {vgaRed, vgaGreen, vgaBlue} = pixel_num;
    else if(s1_valid & pixel_num !=12'hOFF)
        {vgaRed, vgaGreen, vgaBlue} = pixel_num;
    else if(start_valid & pixel_start !=12'hOFF)
        {vgaRed, vgaGreen, vgaBlue} = pixel_start;
    else if(gameover_valid & pixel_gameover !=12'hOFF)
        {vgaRed, vgaGreen, vgaBlue} = pixel_gameover;
    else if(pause_valid & pixel_pause !=12'hOFF)
        {vgaRed, vgaGreen, vgaBlue} = pixel_pause;
    else if (bird_valid & pixel_bird != 12'hOFF)
        {vgaRed, vgaGreen, vgaBlue} = pixel_bird;
    else if(pipemouth_valid & pixel_pipemouth != 12'hOFF)
        {vgaRed, vgaGreen, vgaBlue} = pixel_pipemouth;
    else if(pipebase_valid & pixel_pipebase != 12'hOFF)
        {vgaRed, vgaGreen, vgaBlue} = pixel_pipebase;
    else if(pipebody_valid & pixel_pipebody != 12'hOFF)
        {vgaRed, vgaGreen, vgaBlue} = pixel_pipebody;
    else
        {vgaRed, vgaGreen, vgaBlue} = pixel_bg;
else
        {vgaRed, vgaGreen, vgaBlue} = 12'h0;

endmodule
```

接著要介紹的是我們 VGA 顯示中最核心的一個部分，也就是 pixel_gen，此 module 中，我們實現了將不同的圖片顯示在同一個畫面上的方法，也就是分圖層顯示，我們分圖層顯示的方法就是想要顯示一張圖的時候，我會根據現在顯示的範圍是否要顯示這張圖來給一個 picture 的 valid 值，例如我想要顯示我的 bird 這張圖，我就先決定好我在 h_cnt 以及 v_cnt 數到多少的時候 bird_valid 為高電位，而我們在判斷要給哪張圖片的 pixel RGB 值的時候，就是使用這樣的方法，若是在合法的顯示範圍，也就是 valid 為高電位的時候，使用上圖的 if else 邏輯，先被判斷的就是比較高的圖層，在我們的遊戲中，score 就是在最高的圖層；以此類推 b0、b1、s0、s1 是 best 分數以及本場遊戲的分數 Score 數字對應的圖；再下去是開始遊戲 Get Ready 的圖、遊戲結束 gameover 的圖、以及遊戲途中暫停的圖；再下一層就會是 bird，也就是玩家操作的鳥的圖；再下去是水管的管身、管口的圖；最後若是以上的東西都不是 valid，那我就會顯示 pixel_bg，也就是背景的圖。而若是不合法的輸入，那我就會輸出 12'h0，也就是全嘿，不給訊號。

前面的 code 中也可以看到，我在 if else 中都會有一個 pixel_XXX != 12'0FF 這個判斷條件，這個判斷條件的功能就是用於去背，因為我們輸入圖的時候都是輸入矩形的範圍，但是像是鳥，我要顯示的就會是一個不規則的範圍，這麼時候就需要使用去背的技巧，我使用小畫家，將我不要顯示的部分都漆成 12'0FF 的 RGB 顏色，我使用我其中一張圖來舉例如下圖，這樣子只要 RGB 顏色遇到 12'h0FF 經過我寫的邏輯判斷式判斷，就會跳過這次的邏輯，顯示下一個圖層，也就成功達到了去背的效果。



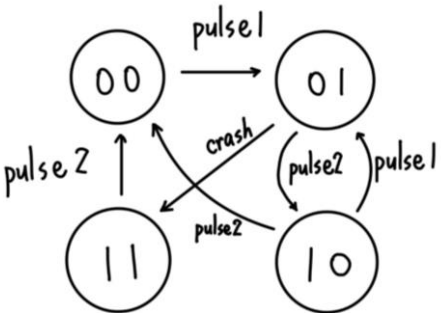
Gamestate：

```
assign pulse1 = key1 & ~key1_delay;
assign pulse2 = key2 & ~key2_delay;

always@(*)begin
    case(state)
    3'd0:begin...
    3'd1:...
    3'd2:...
    3'd3:...
    default:...
    endcase
end

always@(posedge clk or posedge rst)begin
    if(rst) begin
        state <= 3'd0;
        key1_delay = 0;
        key2_delay = 0;
    end
    else begin
        state <= state_next;
        key1_delay <= key1;
        key2_delay <= key2;
    end
end
```

接著介紹一下遊戲的 state 部分，基本上分為 4 個 state，如下圖的 state diagram，state 是 00 的時候就是起始的 state，按下空白鍵就會開始遊戲，進入到 state 01，在這個 state 中，若是按下 enter 鍵，那麼就會進入到 state 10，將正在進行的遊戲暫停，在暫停的 state 10 中，若是按下 enter 就會讓遊戲重新開始，回到 state 00，

而若是按下的是空白鍵的話，那麼 state 將會回到 01，使遊戲繼續進行。若是在 state 01，也就是遊戲進行的時候，玩家操縱的鳥撞到了障礙物，那 crash 這個變數就會變為高電位，使 state 變為 11 進入到 gameover 的 state，在 state 11 也就是 gameover 的 state 的時候，需要按下 enter 鍵才可以回到初始的 state 00 重新開始遊戲。

而我觸發轉換 state 的 pulse1 以及 pulse2 是使用 key_down[9'h29]以及 key_down[9'h5A]這兩個訊號，先進行一個 cycle 的 delay 後再與當前時間做比較，若是前一刻是低電位但當前為高電位，那麼 pulse 就會變成高電位，其餘時間都會是低電位，這樣就可以產生空白鍵以及 enter 按鍵的 onepluse 訊號。



mem_addr_gen：

接著進到比較複雜的 block，也就是 mem_addr_gen，這個 block 主要的功能是控制鳥的動作以及高度，還有背景畫面的移動部分。下面開始解釋其原理。

```
54  assign reset = (state_delay != 3'd0 & state == 3'd0);
55  assign crash = (bird_v >= 380 | bird_v <= 0 ) | (bird_valid & (pipebody_valid | pipebase_valid | pipemouth_valid)
56                  & pixel_bird != 12'hOFF & pixel_pipebody != 12'hOFF & pixel_pipemouth != 12'hOFF & pixel_pipebase != 12'hOFF);
57  assign pixel_addr_bg = (((h_cnt>>1) + cnt)%320 + 320 * (v_cnt>>1)) % 76800;   // 640*480 -> 320*240
58  assign bird_valid = h_cnt >= `BIRD_H & h_cnt < (`BIRD_H + `BIRD_SIDE ) &
59                      v_cnt >= bird_v & v_cnt < (bird_v  + `BIRD_SIDE );
60  assign pixel_addr_bird = ((h_cnt - `BIRD_H + 1 + (v_cnt - bird_v) * `BIRD_SIDE));
```

首先可以看到我新 assign 的一些變數，reset 就是在辨別 state 變成 00，也就是從 gameover 重新開始的那個瞬間將會使這個變數變為高電位，其餘時刻將會是低電位，需要這個變數是因為每次重新開始遊戲的時候需要將鳥的高度重新設定成初始的位置，而不是只有在整個遊戲 reset 的時候才要重設位置。

pixel_addr_bg 以及 pixel_addr_bird 分別就是其在對應的 h_cnt 以及 v_cnt 需要取到的圖片 address 位置，計算方式就是看需要先知道我們的圖片大小有多少，並且知道我們想要顯示的 h_cnt 座標以及 v_cnt 座標，這樣我們就將我們要顯示的位置用座標移動的方式來評移我們的圖片到我們想要顯示的位置，概念很像我們在 x y 坐標軸移動，若是要右移 k 格，那我就用(x-k)來代替 x 這樣，我們可以

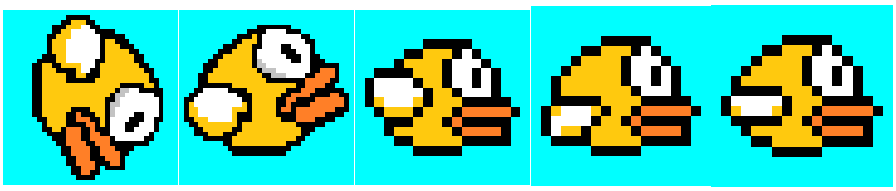
看到 pixel_addr_bg 的部分，我們將 h_cnt 以及 v_cnt 算術右移了 1 個 bit，這樣的原因是我們的記憶體是有限的，因此我們在背景的部分就只放了 320*240 的背景而不是 640*480 的，這樣子可以省下相當多的記憶體大小，使我們可以放下更多的圖片。

Bird_valid 的部分基本上就是控制鳥要在螢幕的甚麼座標範圍進行顯示，我們將鳥的最左半部固定在 h_cnt 座標 200 的位置，並且使用 bird_v 這個變數來調整鳥的高度，使鳥可以根據這個變數自由上下移動。

Crash 的部分則是當 bird_valid 以及 pipe_valid 同時是高電位的時候且顏色不是去背的顏色，這樣就代表鳥撞到水管了，此時 crash 就會變成高電位，或者是 bird_v 這個變數大於 380 或是小於等於 0，這分別代表鳥撞到地板了，或者是鳥撞到天花板了，這兩種狀況也會使鳥死掉，使遊戲結束，因此此時 crash 也要是高電位。

```
62  always@(*)begin
63  if(state != 3'd0)
64      case(bird_state[3:1])
65      3'd0: pixel_bird = pixel_birddown;
66      3'd1: pixel_bird = pixel_birdfly;
67      3'd2: pixel_bird = pixel_birdfly;
68      3'd3: pixel_bird = pixel_birdideal;
69      3'd4: pixel_bird = pixel_birdfly2;
70      3'd5: pixel_bird = pixel_birdup;
71      3'd6: pixel_bird = pixel_birdup;
72      3'd7: pixel_bird = pixel_birdfly2;
73      endcase
74  else
75      pixel_bird = pixel_birdideal;
76  end
```

為了使遊戲內容更加的精緻，我在鳥的飛行狀態下也是有加一些手腳，我鳥的飛行狀態有五種不一樣的圖案，然後根據不一樣的 state 會顯示不一樣的圖案，實現方法就是根據鳥現在的 state 來將不同圖片輸出出來的 pixel 值丟給 pixel_bird 來做顯示，而且若是 state 在準備階段的時候，我就將其固定為初始狀態的鳥的造型，下面展示一下我五張鳥的圖的動作。



```
always@(*)begin
if(state == 3'd1) begin
    case(bird_state[3:1])
    3'd0: bird_v_next = (bird_v + 5) < 380 ? (bird_v + 5) : 380;
    3'd1: bird_v_next = (bird_v + 3) < 380 ? (bird_v + 3) : 380;
    3'd2: bird_v_next = (bird_v + 1) < 380? (bird_v + 1) : 380;
    3'd3: bird_v_next = bird_v > 0 ? bird_v : 0;
    3'd4: bird_v_next = (bird_v - 1) > 0 ? (bird_v - 1) : 0;
    3'd5: bird_v_next = (bird_v - 3) > 0 ? (bird_v - 3) : 0;
    3'd6: bird_v_next = (bird_v - 5) > 0 ? (bird_v - 5) : 0;
    3'd7: bird_v_next = (bird_v - 7) > 0 ? (bird_v - 7) : 0;
    endcase
    ini_cnt_next = ini_cnt;
    if(cnt < 320)  cnt_next = cnt + 1;
    else cnt_next = 0;
end
else if(state == 3'd0) begin
    case(ini_cnt[3:2])
        2'd0: bird_v_next = (bird_v + 2);
        2'd1: bird_v_next = (bird_v + 1);
        2'd2: bird_v_next = (bird_v - 1);
        2'd3: bird_v_next = (bird_v - 2);
    endcase
    ini_cnt_next = ini_cnt + 1;
    if(cnt < 320) cnt_next = cnt + 1;
    else  cnt_next = 0;
end
else begin
    cnt_next = cnt;
    bird_v_next = bird_v;
    ini_cnt_next = ini_cnt;
end
end
```

```
always@(*)begin
if(state == 3'd1)
    if(key & ~key_delay)
        bird_state_next = 4'b1111;
    else if(bird_state == 4'd0)
        bird_state_next = 4'd0;
    else
        bird_state_next = bird_state - 1;
else
    bird_state_next = bird_state;
end
```

而在遊戲進行的階段，也就是 state＝1，那根據我現在的 state，鳥會飛起不同的高度，state 會隨遊戲單位時間的過去而慢慢遞減到 0，以 0 的 state 來看，就是每次一個遊戲單位時間過去，鳥的高度就會往下掉三格，而在不同的 state 的有不同的上升以及下降的速度，這是為了更加的符合物理的概念，在飛起的時候，會有一種等加速度的感覺，這樣遊戲玩起來也會更加的真實，每當遊戲進行過程中，按下空白鍵就會將 bird_state 變為第七個 state，這樣鳥就會先拍動翅膀上升，然後變平的飛，最後開始下降，在這個過程中，鳥的動作也會依據前面所提到的對應動作來顯示。

值得說的是我們的空白鍵按下模式是按一次會跳起來一次的模式，也就是使用 onepluse 的模式來進行遊玩，更貼近原本遊戲的玩法。

然後在準備狀態也就是 state＝0 的時候，為了讓鳥也有在動的感覺使遊戲變得更加生動，我也加上了 4 個 state，並且這個 state 的部分會一直持續重複使鳥上下上下飛，這樣也可以讓我的畫面看起來更加的動感，也比較符合原版遊戲的設計。

而背景畫面也是同時需要移動的，我的方法就是將 background 的 h_cnt 成分加上一個計數器 cnt，並且取對 320 的餘數，這樣就可以使整個畫面隨著時間漸漸的左移，讓鳥真的有往右飛的感覺，並且同時背景畫面的位置也不會跑掉。

```
124 always @(posedge clk_21 or posedge rst) begin
125         if(rst)
126         state_delay <= 0;
127         else
128         state_delay <= state;
129     end
130 
131 always@(posedge clk_21 or posedge rst) begin
132     if(rst|reset) begin
133         bird_y <= 200;
134         bird_state <= 5'b01111;
135         key_delay <= 0;
136         ini_cnt <= 0;
137         cnt <= 0;
138     end
139     else if(state == 3'd0) begin
140         bird_y <= bird_y_next;
141         bird_state <= 5'b011111;
142         key_delay <= 0;
143         ini_cnt <= ini_cnt_next;
144         cnt <= cnt_next;
145     end
146     else begin
147         bird_y <= bird_y_next;
148         bird_state <= bird_state_next;
149         key_delay <= key;
150         ini_cnt <= 2'd0;
151         cnt <= cnt_next;
152     end
153 end
```

最後可以看到這個 module 中，當我 rst(globle reset)或是 reset 是高電位的時候就會重製我鳥的高度為 200，這樣就可以遊戲結束後重新開始可以使鳥在同一個位置上開始。

pipe：

在說明 pipe 這個 module 之前，由於 pipe 會使用到 pipe_gen 這個 module，還會使用到 LFSR 這個 module，因此也在這個部分介紹。

(1) pipe_gen:

```
47  LFSR LFSR_inst(.clk21(clk_21),.rst(rst),.random(pipe_v_new));
48
49  assign reset = (state_delay != 3'd0 & state == 3'd0);
50  assign pipebody_valid = (v_cnt < pipe_v | (v_cnt <= 415  & v_cnt > (pipe_v + `SPACE + 30)))
51                        & (pipe_h >= 50 ? h_cnt >= (pipe_h - 50) & h_cnt < pipe_h : h_cnt < pipe_h);
52  assign pixel_addr_body = (h_cnt - pipe_h + 4) % 50;
53
54  assign pipebase_valid =  (v_cnt >= pipe_v & v_cnt < pipe_v + 15)
55                        & (pipe_h >= 50 ? h_cnt >= (pipe_h - 50) & h_cnt < pipe_h : h_cnt < pipe_h);
56  assign pixel_addr_base = (h_cnt - pipe_h + 51) + (v_cnt - pipe_v) * 50;
57  assign pipemouth_valid =  (v_cnt > (pipe_v + `SPACE + 15) & v_cnt <= (pipe_v + `SPACE + 30))
58                        & (pipe_h >= 50 ? h_cnt >= (pipe_h - 50) & h_cnt < pipe_h : h_cnt < pipe_h);
59  assign pixel_addr_mouth = (h_cnt - pipe_h + 51) + (v_cnt - pipe_v - `SPACE - 16) * 50;
60
61  always @* begin
62      if (new) begin
63          pipe_h_next = 690;
64          pipe_v_next = pipe_v_new % 220 + 60;
65      end
66      else if (state == 3'd1) begin
67          if (pipe_h > 0) begin
68              pipe_h_next = pipe_h - 2;
69              pipe_v_next = pipe_v;
70          end
71          else begin
72              pipe_h_next = pipe_h;
73              pipe_v_next = pipe_v;
74          end
75      end
76      else begin
77          pipe_h_next = pipe_h;
78          pipe_v_next = pipe_v;
79      end
80  end
```

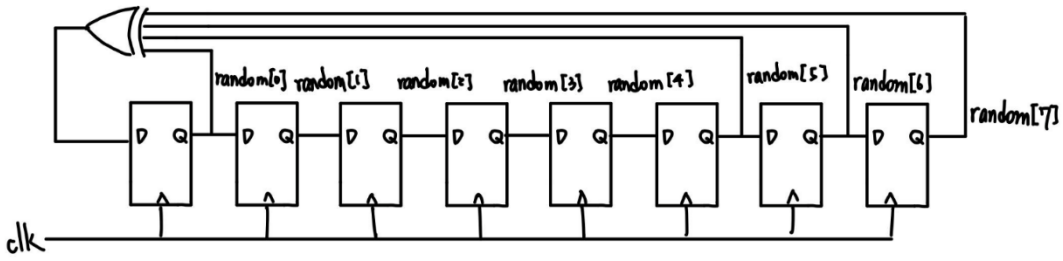
```
82      always @(posedge clk_21 or posedge rst) begin
83          if(rst)
84          state_delay <= 0;
85          state_delay <= state;
86      end
87
88      always @ (posedge clk_21 or posedge rst) begin
89          if (rst|reset) begin
90              pipe_h <= 0;
91              pipe_v <= 0;
92          end else begin
93              pipe_h <= pipe_h_next;
94              pipe_v <= pipe_v_next;
95          end
96      end
```

這個 module 主要是要生成一根會動以及會自動生成隨機高度的水管，會使用到 LFSR 這個 module，因此我首先先介紹 LFSR:

(2) LFSR:

這個 LFSR 的部分主要是用來產生一個隨機的高度，而我想要很高的隨機性，所以我將產生一個 8 個 bit 的 LFSR，並且使用 clk_21 這個遊戲內的單位時間來進行更新的頻率，這個 module 的輸出就會是 random[7:0]這個 8bit 的訊號，可以用來決定 256 的水管高度，而在遊戲中我只會使用到 220 個高度，因此在外面會

再進行餘數的處理，架構圖就是使用上課所教過的 LFSR 來做，產生 $1+x^1+x^6+x^7+x^8$ 的訊號 feedback 給 shift register 的開頭。



回到 pipe_gen 的部分，我在這個部分由於想要產生一個隨機高度的水管，因此我的水管會分為上管口、下管口以及管身的部分，分別在 code 中的名字是 mouth、base、body。然後由於我們的水管是會移動的並且有隨機的高度，因此我們需要一個變數 pipe_v 來表達此水管當前的高度，以及 pipe_h 來表達此水管目前的水平位置。

pixel_addr 以及 valid 的部分與前面在做鳥飛行的時候概念差不多，只是我現在需要使上管口以及下管口之間有一點間隔使鳥飛的過去，因此我有 define 一個 SPACE 這個數字也就是管口之間差了多少的 pixel 值。值得注意的是，我在給 body 的 address 值的時候我直接拿 h_cnt -pipe_h 再憑移以後的值去對 50 取餘數，這是因為我的管身每一列都是一樣的，並且寬度都是 50 個 pixel，因此可以把精簡成一直重複顯示第一行的部分，這樣也可以更方便的將 220 個不一樣的高度的管身實現出來，下方也貼出我的水管管身上管口以及下管口的部分。

下方的邏輯判斷式，當 new 為高電位的時候，也就是要產生一個新的 pipe，但是剛產生的時候不可以馬上顯示在螢幕上，而是要使水管由右邊的螢幕邊緣慢慢出現，因此我剛生成的水管會把 pipe_h 設為 690，也就是銀幕的邊緣 640 加上水管的寬度 50，並且在此同時我會將 LFSR 產生的隨機位置先對 220 取餘數以後再經過平移丟給 pipe_v，這樣子就可以確保每次新產生一個 pipe，都會重新從右邊出發並且有一個隨機的高度。而若是 state 為 1，也就是爺系進行中的狀態，那麼就要隨著遊戲每過一個單位的時間就要把水管的水平位置往做 shift 兩格，之所以是兩格是為了跟前面的背景移動速度相同，而背景是 320*240 放大兩倍的結果，因此移動 address 一格就是等效移動兩格的意思，其餘時候管子就都不會有移動。最後在 reset，也就是遊戲結束按下 enter 重新開始的時候，我會將管子的位置設定成 0，也就是在銀幕左端的左邊。

```
51  assign pipebody_valid = !body_valid;
52  assign pipebase_valid = !base_valid;
53  assign pipemouth_valid = !mouth_valid;
54  assign new[0] = (cnt > `INTERVAL) & ~|pipe_h0;
55  assign new[1] = (cnt > `INTERVAL) &  |pipe_h0 & ~|pipe_h1;
56  assign new[2] = (cnt > `INTERVAL) &  |pipe_h0 &  |pipe_h1 & ~|pipe_h2;
57  assign new[3] = (cnt > `INTERVAL) &  |pipe_h0 &  |pipe_h1 &  |pipe_h2 & ~|pipe_h3;
58
59  always@(*)begin
60    case(body_valid)
61         4'b0001:   pixel_pipebody = pixel_pipebody0;
62         4'b0010:   pixel_pipebody = pixel_pipebody1;
63         4'b0100:   pixel_pipebody = pixel_pipebody2;
64         4'b1000:   pixel_pipebody = pixel_pipebody3;
65         default:   pixel_pipebody = pixel_pipebody0;
66    endcase
67  end
68
69  always@(*)begin
70    case(mouth_valid)...
77  end
78
79  always@(*)begin
80    case(base_valid)...
87  end
```

```
89  always @(posedge clk_21 or posedge rst) begin
90    if(rst | (state_delay != 3'd0 && state == 3'd0)) begin
91        state_delay <= 0;
92        cnt <= 0;
93        score <= 0;
94    end
95    else begin
96        state_delay <= state;
97        cnt <=  cnt_next;
98        score <= score_next;
99    end
100 end
101
102 always @* begin
103 if (state == 3'd1) begin
104     if (cnt > `INTERVAL)
105         cnt_next = 0;
106     else
107         cnt_next = cnt + 1;
108     if (pipe_h0 == 200 | pipe_h1 == 200 | pipe_h2 == 200 | pipe_h3 == 200)
109          score_next = score + 1;
110      else
111          score_next = score;
112 end
113 else begin
114     cnt_next = cnt;
115     score_next = score;
116 end
117 end
```

```
pipe_gen g1(
.clk_21(clk_21),
.clk(clk),
.rst(rst),
.state(state),
.h_cnt(h_cnt),
.v_cnt(v_cnt),
.new(new[0]),
.pipe_h(pipe_h0),
.pixel_pipebody(pixel_pipebody0),
.pipebody_valid(body_valid[0]),
.pixel_pipemouth(pixel_pipemouth0),
.pipemouth_valid(mouth_valid[0]),
.pixel_pipebase(pixel_pipebase0),
.pipebase_valid(base_valid[0])
);

pipe_gen g2(
.clk_21(clk_21),
.clk(clk),
.rst(rst),
.state(state),
.h_cnt(h_cnt),
.v_cnt(v_cnt),
.new(new[1]),
.pipe_h(pipe_h1),
.pixel_pipebody(pixel_pipebody1),
.pipebody_valid(body_valid[1]),
.pixel_pipemouth(pixel_pipemouth1),
.pipemouth_valid(mouth_valid[1]),
.pixel_pipebase(pixel_pipebase1),
.pipebase_valid(base_valid[1])
);
```

```
pipe_gen g3(
.clk_21(clk_21),
.clk(clk),
.rst(rst),
.state(state),
.h_cnt(h_cnt),
.v_cnt(v_cnt),
.new(new[2]),
.pipe_h(pipe_h2),
.pixel_pipebody(pixel_pipebody2),
.pipebody_valid(body_valid[2]),
.pixel_pipemouth(pixel_pipemouth2),
.pipemouth_valid(mouth_valid[2]),
.pixel_pipebase(pixel_pipebase2),
.pipebase_valid(base_valid[2])
);

pipe_gen g4(
.clk_21(clk_21),
.clk(clk),
.rst(rst),
.state(state),
.h_cnt(h_cnt),
.v_cnt(v_cnt),
.new(new[3]),
.pipe_h(pipe_h3),
.pixel_pipebody(pixel_pipebody3),
.pipebody_valid(body_valid[3]),
.pixel_pipemouth(pixel_pipemouth3),
.pipemouth_valid(mouth_valid[3]),
.pixel_pipebase(pixel_pipebase3),
.pipebase_valid(base_valid[3])
);
```

回到 pipe 這個 module，在這裡由於我想要在螢幕上顯示最多 4 個水管，因此我將四個 pipe_gen 的 module 都丟了進來，並且給了不同的名字如上圖。

在遊戲中，遊戲剛開始的時候我不會使水管跑出來，遊戲開始後才會開始跑出第一根水管第二根水管到第四根水管這樣，因此為了實現這樣子的功能，我就運用了我再 pipe_gen 所設計的 new 以及他的其中一個 output 也就是 pipe_h。我設計了一個計數器，每次 counter 數到了 100，就會判斷說現在需不需要生成一個新的水管，並且歸零重新開始數，而判斷是否生成新的水管的方式就是看我這個水管是不是已經跑到了螢幕的最左邊了，我為了方便，就把其 pipe_h 的位置取了 bitwise 的 nor，這樣子當全部 bit 都是 0 的時候就會輸出高電位，然後每一個 pipe 的 new 就以這樣的方式來產生，第一個 pipe 只要自己跑完就會自己生成，第二個 pipe 要自己跑完且第一個 pipe 還正再跑的時候生成，以此類推。

接著我需要把這四個 pipe 的 valid 以及 pixel 的值都整合在一起，因此我就使用了 case 來根據 vaild 的值來判斷現在顯示到的是哪個 pipe 我就給 pixel 哪個 pipe 的 pixel 值，pipe_valid 的部分的話就是只要有任何一個 pipe 的 valid 是高電位我就讓他變成高電位。

最後這個遊戲也有計分的功能，當有任何一個 pipe 的 pipe_h 經過了 200，也就是前面鳥的水平位置的話，我就把 score 這個變數加一分，這樣子就可以在我每次越過一個水管的時候將 score 加一分，達到計分的效果，而當 state 變為 0

的時候，也就是遊戲結束重新開始的時候我也要將 score 歸零，代表重新開始遊戲。

Maingame：

此 module 負責處理的是遊戲剩下的開始暫停 gameover 以及記分板相關的顯示，在這個 module 之下有 register 以及 score_address 兩個子 module 因此我在這邊先介紹這兩個 module。

(1) score_address:

```
always@* begin
case(num)
 4'd01: pixel= pixel1;
 4'd02: pixel= pixel2;
 4'd03: pixel= pixel3;
 4'd04: pixel= pixel4;
 4'd05: pixel= pixel5;
 4'd06: pixel= pixel6;
 4'd07: pixel= pixel7;
 4'd08: pixel= pixel8;
 4'd09: pixel= pixel9;
 4'd0: pixel= pixel0;
 default: pixel= pixel0;
 endcase
end
```

由於我們將我們的分數顯示在螢幕上，因此我們也需要輸出 0123456789 這幾種數字的圖案到螢幕上，此 module 就是將要顯示的數字輸入進來並且給了 pixel_addr，就會 output 出對應的 pixel 的 RGB 值給外面，基本上就只是使用 MUX 的功能，使用這個 module 的原因是為了壓縮記憶體，因為我們原本是每個顯示的數字都對應一個 module，但發現若是把 block memory 放在裡面的 block 會重複計算記憶體的使用量，造成記憶體不夠的現象，因此我們就特別使用一個 module 將所有的數字以及輸出對應的 pixel 都統整起來，藉由外面傳進來的 NUM 來進行取值。

(2) register:

此部分就是需要控制 BEST SCORE 用的，當 load_en 為高電位的時候，代表 SCORE 大於或是等於 BEST SCORE，此時就會將 SCORE 的值傳入 BEST 中。

```
9  always@*begin
10 if(load_en)
11   reg_out_temp = reg_in;
12   else
13 reg_out_temp = reg_out;
14 end
15
16
17 always@(posedge clk or posedge rst)begin
18     if(rst)
19     reg_out <= 5'd0;
20     else
21     reg_out <= reg_out_temp;
22 end
```

回到 Maingame 這個 module，下方的部分有我所 assign 的 valid 以及 pixel 的值，上面的部分就是當 state 為 00 的時候那我畫面中間就要顯示遊戲開始的畫面，若是 state 為 10，也就是暫停的時候，那就要顯示暫停的畫面，最後是當 state 為 11 的時候，也就是 gameover 的時候，就要顯示遊戲結束的畫面，並且附上我們對應的圖。

```
assign start_valid = (state == 3'd0) & (h_cnt >= 230 & h_cnt < 410)
                   & (v_cnt >= 200 & v_cnt < 260);
assign gameover_valid = (state == 3'd3) & (h_cnt >= 200 & h_cnt < 430)
                   & (v_cnt >= 200 & v_cnt < 260);
assign pause_valid = (state == 3'd2) & (h_cnt >= 260 & h_cnt < 380)
                   & (v_cnt >= 150 & v_cnt < 270);
assign score_valid = (h_cnt >= 10 & h_cnt < 190) & (v_cnt >= 10 & v_cnt < 70);

assign s1_valid = (h_cnt >= 200 & h_cnt < 230) & (v_cnt >= 10 & v_cnt < 40);
assign s0_valid = (h_cnt >= 230 & h_cnt < 260) & (v_cnt >= 10 & v_cnt < 40);
assign b1_valid = (h_cnt >= 200 & h_cnt < 230) & (v_cnt >= 40 & v_cnt < 70);
assign b0_valid = (h_cnt >= 230 & h_cnt < 260) & (v_cnt >= 40 & v_cnt < 70);

/*(((h_cnt>>1) + cnt)%320 + 320 * (v_cnt>>1)) % 76800; // 640*480 -> 320*240*/

assign pixel_addr_start = (((h_cnt - 230 + 2)>> 1) + ((v_cnt - 200)>> 1) * 90);
assign pixel_addr_gameover =  (((h_cnt - 200 + 2)>> 1) + ((v_cnt - 200) >> 1) * 115);
assign pixel_addr_pause =  (((h_cnt - 260+2 )>> 1) + ((v_cnt - 150) >> 1) * 60);
assign pixel_addr_score =  (((h_cnt - 10 + 2)>> 1) + ((v_cnt - 10)>> 1) * 90);

assign pixel_addr_s1 =  ((h_cnt - 200 + 2) + ((v_cnt - 10)) * 30);
assign pixel_addr_s0 =  ((h_cnt - 230 + 2) + ((v_cnt - 10)) * 30);
assign pixel_addr_b1 =  (h_cnt - 200 + 2) + ((v_cnt - 40) * 30);
assign pixel_addr_b0 =  (h_cnt - 230 + 2) + ((v_cnt - 40) * 30);

register r0(.reg_out(num_b0),.reg_in(num_s0),.load_en(load_en),.rst(rst),.clk(clk));
register r1(.reg_out(num_b1),.reg_in(num_s1),.load_en(load_en),.rst(rst),.clk(clk));

assign load_en= (score>=(10*num_b1+num_b0)) & (score!=0);
assign num_s0=score%10;
assign num_s1=score/10;
```

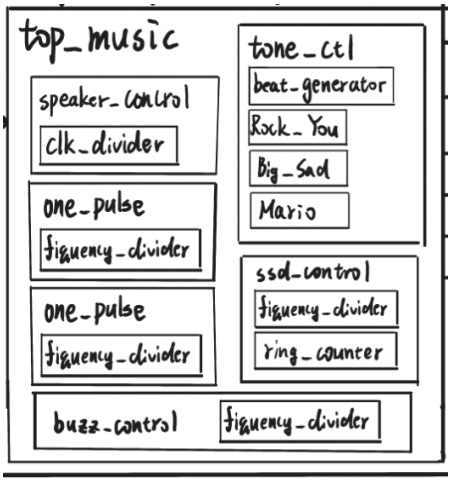
```
always@*begin
    if(s1_valid)begin
        pixel_addr_num= pixel_addr_s1;
        num=num_s1;
    end
    else if(s0_valid)begin
        pixel_addr_num= pixel_addr_s0;
        num=num_s0;
    end
    else if(b0_valid)begin
        pixel_addr_num= pixel_addr_b0;
        num=num_b0;
    end
    else if (b1_valid)begin
        pixel_addr_num= pixel_addr_b1;
        num=num_b1;
    end
    else begin
        pixel_addr_num= 0;
        num=0;
    end
end

score_address S(
.pixel(pixel_num),
.clk(clk),
.pixel_addr(pixel_addr_num),
.num(num)
);
```

顯示的方式基本上也跟前面的 module 差不多，只不過多了 state 的判斷條件，為了壓縮記憶體空間，我這些圖的 address 也是有使用前面所提到的將 address bit 數右移一次的技巧，這樣就可以將使用到的記憶體大小變為原本的 1/4 倍。

下方則是關於 score 以及 best 的分數的顯示，顯示方式也是相同的，但由於我 score 有兩位數需要輸入且 best 也有兩位數需要輸入，因此我就根據現在正在顯示哪個數字的位置來將那個位置現在應該要有的數字，也就是當要顯示的位數現在是 valid，就把要顯示的數字丟進 score_address 顯示對應的 pixel RGB 值。

值得說的部份就是我們的 best score 是不會像是 score 因為 state 跑到 00 就重製，只有在遊戲重製 globle rst 被啟動的時候才會歸零，符合 best score 應該要有的功能。

Background music



Top_music model:

```
23  module top_music(
24      input f_crystal,
25      input reset,
26      input BGM_on_off,   // switch on => BGM on vice versa
27      input vol_up_btn,
28      input vol_down_btn,
29      input [2:0] game_state,
30      output [3:0] ssd_ctl,
31      output [7:0] display,
32      output audio_mclk,
33      output audio_lrck,
34      output audio_sck,
35      output audio_sdin
36      );
37      wire [15:0] audio1, audio2;
38      wire [21:0] note_div1, note_div2;
39      wire vol_up_pulse, vol_down_pulse;
40      reg rst_state;
41
42  // needs to add these three declaration to input declaration above
43  // needs a FSM to determine which music to play depending on the state of game
44      assign game_start =  (game_state == 3'b000);              // the opening music
45      assign pause = (game_state == 3'b010);
46      assign gaming = (game_state == 3'b001);                   // game playing
47      assign game_over = (game_state == 3'b011);     // game over music
48
```

```
49      speaker_control d2(         // paralell to serial
50          .f_crystal(f_crystal),
51          .reset(reset),
52          .audio1(audio1),
53          .audio2(audio2),
54          .audio_mclk(audio_mclk),
55          .audio_lrck(audio_lrck),
56          .audio_sck(audio_sck),
57          .audio_sdin(audio_sdin)
58      );
59
60      tone_ctl g0(
61          .reset(reset),
62          .f_crystal(f_crystal),
63          .game_start(game_start),
64          .gaming(gaming),
65          .game_over(game_over),
66          .pause(pause),
67          .note_div1(note_div1),
68          .note_div2(note_div2)
69      );
70
```

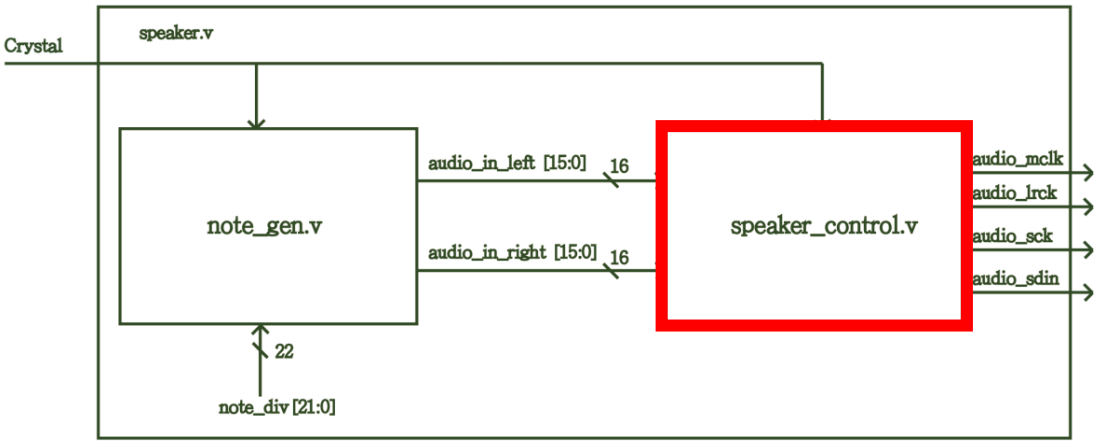
```
71      buzz_control d3(
72          .f_crystal(f_crystal),
73          .reset(reset),
74          .BGM_on_off(BGM_on_off),
75          .note_div1(note_div1),
76          .note_div2(note_div2),
77          .vol_up_pulse(vol_up_pulse),
78          .vol_down_pulse(vol_down_pulse),
79          .audio1(audio1),
80          .audio2(audio2)
81      );
82
83      one_pulse d4(
84          .reset(reset),
85          .f_crystal(f_crystal),
86          .pb_in(vol_up_btn),
87          .pb_pulse(vol_up_pulse)
88      );
```

```
89      one_pulse d5(
90          .reset(reset),
91          .f_crystal(f_crystal),
92          .pb_in(vol_down_btn),
93          .pb_pulse(vol_down_pulse)
94      );
95
96      ssd_control d6(
97          .reset(reset|pause),
98          .f_crystal(f_crystal),
99          .game_start(game_start),
100         .gaming(gaming),
101         .game_over(game_over),
102         .ssd_ctl(ssd_ctl),
103         .display(display)
104     );
105
106 endmodule
```

Top_music model 是連接以下與音樂相關的音樂模組的總模組，其中除了接線外沒有其他的功能。接者我們介紹以下的模組。

Speaker_control module:

```
module speaker_control(
    input f_crystal,
    input reset,
    input [15:0] audio1, audio2,
    output audio_mclk,      // 25MHZ
    output audio_lrck,      // 25/128 MHZ
    output audio_sck,       // 25/4 MHZ
    output reg audio_sdin
);
    wire sampling_clk;
    reg audio_sdin_next;
    reg [4:0] count;
    reg [4:0] count_next;

    clk_divider U0(
    .f_crystal(f_crystal),
    .reset(reset),
    .audio_mclk(audio_mclk),
    .audio_lrck(audio_lrck),
    .sampling_clk(sampling_clk)
    );

     assign audio_sck = sampling_clk;

   always@(posedge sampling_clk) begin
       if (count == 5'd31) count_next <= 5'd0;
       else count_next <= count + 5'd1;
   end

    always@(posedge sampling_clk or posedge reset) begin
        if (reset) count <= 5'd0;
        else count <= count_next;
    end

    always@(posedge sampling_clk or posedge reset) begin
        if (reset) audio_sdin <= 1'b0;
        else audio_sdin <= audio_sdin_next;
    end

    always@* begin
        case(count)
            5'd0: audio_sdin_next  <= audio1[15];
            5'd1: audio_sdin_next  <= audio1[14];
            5'd2: audio_sdin_next  <= audio1[13];
            5'd3: audio_sdin_next  <= audio1[12];
            5'd4: audio_sdin_next  <= audio1[11];
            5'd5: audio_sdin_next  <= audio1[10];
            5'd6: audio_sdin_next  <= audio1[9];
            5'd7: audio_sdin_next  <= audio1[8];
            5'd8: audio_sdin_next  <= audio1[7];
            5'd9: audio_sdin_next  <= audio1[6];
            5'd10: audio_sdin_next  <= audio1[5];
            5'd11: audio_sdin_next  <= audio1[4];
            5'd12: audio_sdin_next  <= audio1[3];
            5'd13: audio_sdin_next  <= audio1[2];
            5'd14: audio_sdin_next  <= audio1[1];
            5'd15: audio_sdin_next  <= audio1[0];
            5'd16: audio_sdin_next  <= audio2[15];
            5'd17: audio_sdin_next  <= audio2[14];
            5'd18: audio_sdin_next  <= audio2[13];
            5'd19: audio_sdin_next  <= audio2[12];
            5'd20: audio_sdin_next  <= audio2[11];
            5'd21: audio_sdin_next  <= audio2[10];
            5'd22: audio_sdin_next  <= audio2[9];
            5'd23: audio_sdin_next  <= audio2[8];
            5'd24: audio_sdin_next  <= audio2[7];
            5'd25: audio_sdin_next  <= audio2[6];
            5'd26: audio_sdin_next  <= audio2[5];
            5'd27: audio_sdin_next  <= audio2[4];
            5'd28: audio_sdin_next  <= audio2[3];
            5'd29: audio_sdin_next  <= audio2[2];
            5'd30: audio_sdin_next  <= audio2[1];
            5'd31: audio_sdin_next  <= audio2[0];
            default: audio_sdin_next  <= audio_sdin;
        endcase
    end
endmodule
```



Speaker_control 這個模組與 Lab 的 speaker control 的概念並沒有不一樣。其中我們要利用板子上 100Mhz 的石英震盪器去生成 Master Clock, Left-Right Clock, 以及 Serial Clock。並且將原本我們原本左右聲道的 parallel 的資料，轉換成 serial 的資料。

clk_divider module:

```
module clk_divider(
    input f_crystal,
    input reset,
    output audio_mclk,
    output audio_lrck,
    output sampling_clk
    );
    reg [8:0] q, q_temp;

    always@* begin
       q_temp <= q + 9'd1;
    end

    always@(posedge f_crystal or posedge reset) begin
        if (reset) begin
           q <= 9'd0;
         end
        else begin
           q <= q_temp;
      end
    end

 assign  audio_mclk = q[1];
 assign  audio_lrck = q[8];
 assign  sampling_clk = q[3];

endmodule
```

這個模組是將 global 的 clock 利用一連串的 counters，做二進位的除頻，以達到我們預期的頻率。

Tone_ctl module:

這個模組是整個背景音樂為關鍵的模組，包含了產生適合不同音樂的頻率的除頻器，以及三個把樂譜上每一個音符轉成與頻率相關數值的小模組。接著我們要分別來介紹這四個模組：

(1) Beat_generator module:

這個模組分別產生四個頻率給音樂使用，其中的頻率是用來決定歌曲也就是每一個音符之間的時間，因為不是每一首音樂的頻率都要使用不同的頻率，我們最後只使用兩種頻率。其實作方法與 Lab 的除頻器相同，是利用有上限的 counters 來幫助我們降低使用的頻率。

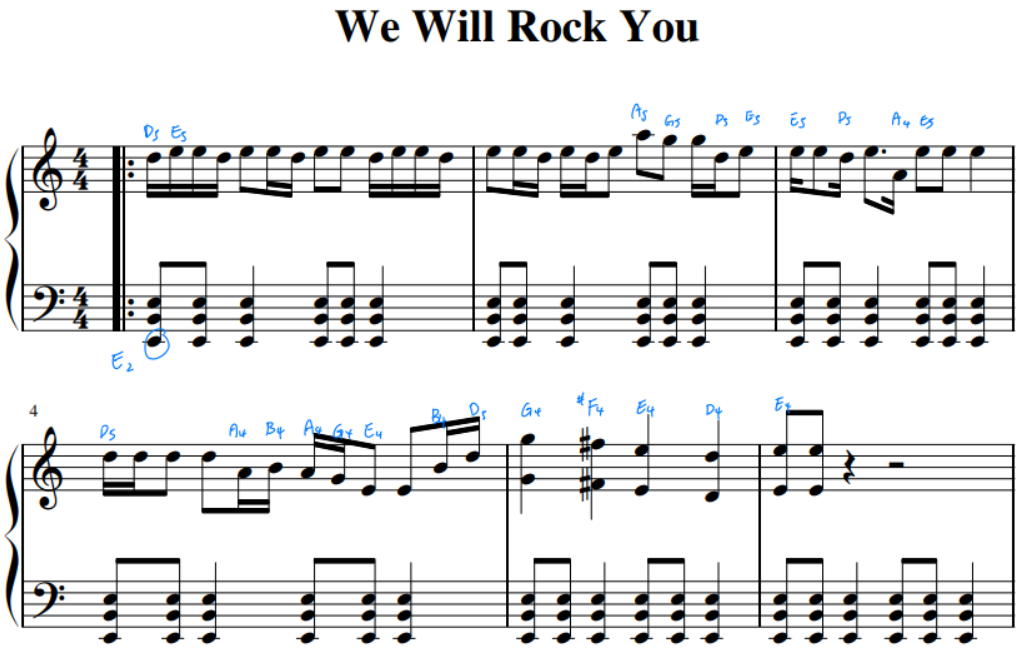
(2) Rock_You module:

此模組名為 Rock you 是因為我們使用很久以前世界足球錦標賽的主題曲 we will rock you。而我們利用這個音樂來當作我們的待機音樂。

```
40  always@(posedge beat or posedge reset) begin
41      if (reset) count <= 9'd256;
42      else   count <= count_next;
43  end
44
45  always@(posedge beat or posedge reset) begin
46      if (reset) count_2 <= 4'd15;
47      else   count_2 <= count_next_2;
48  end
49
50  always@* begin
51      if (count == 9'd256) count_next <= 9'd1;
52      else count_next <= count + 9'd1;
53  end
54
55  always@* begin
56      if (count_2 == 4'd15) count_next_2 <= 4'd0;
57      else count_next_2 <= count_2 + 4'd1;
58  end
59
60    always@(posedge beat or posedge reset) begin
61      if (reset) begin
62          note_div1 <= 22'd0;
63          note_div2 <= 22'd0;
64      end
65      else  begin
66          note_div1 <= note_div1_next;
67          note_div2 <= note_div2_next;
68       end
69    end
```

```
72  always@(*) begin
73      case (count)
74      9'd0: note_div1_next <= 0;
75
76      9'd1: note_div1_next <= `D5;
77      9'd2: note_div1_next <= 0;
78      9'd3: note_div1_next <= `E5;
79      9'd4: note_div1_next <= 0;
80      9'd5: note_div1_next <= `E5;
81      9'd6: note_div1_next <= 0;
82      9'd7: note_div1_next <= `D5;
83      9'd8: note_div1_next <= 0;
84
85      9'd9: note_div1_next <= `E5;
86      9'd10: note_div1_next <= `E5;
87      9'd11: note_div1_next <= `E5;
88      9'd12: note_div1_next <= 0;
89      9'd13: note_div1_next <= `E5;
90      9'd14: note_div1_next <= 0;
91      9'd15: note_div1_next <= `D5;
92      9'd16: note_div1_next <= 0;
```

我們可以看到左邊的程式，屬於設立一些初始條件，並且為了使的音樂能循環，我們必須先計算我們需要使用的音符數。而右邊的程式是產生音樂的核心(只展示部分程式碼)，我們分別上網找尋各個音樂的鋼琴譜，並且翻成簡譜，然後再分別用左右不同的聲道來讓音樂更有層次(mario 的模組只有單聲道)。以下附上當時我們使用的鋼琴(簡譜)

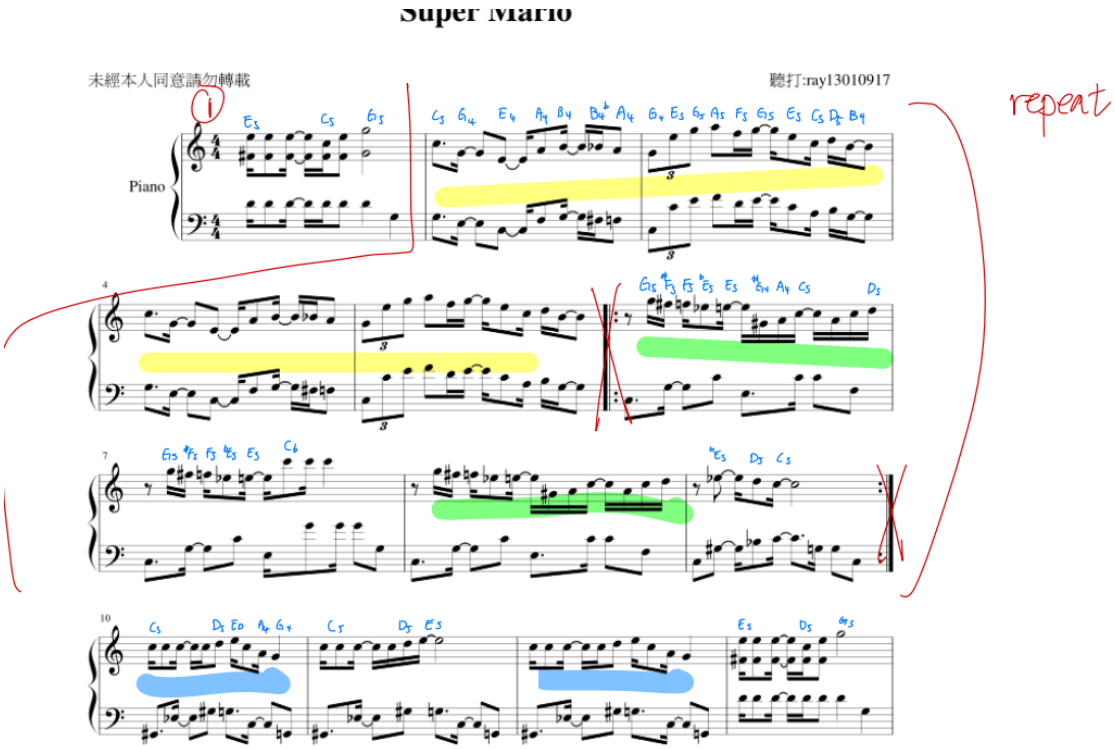


(3) Big_Sad module:

此模組是用來撥放死亡時候的背景音樂的，原本預計使用大悲咒，索性取名叫 big sad，然而後來考量到播出來可能沒什麼人能聽出來，很難產生共鳴。並且因為我們用的 speaker 沒辦法產生太平滑的音樂，只能產生節奏分明的電子音。剛好有朋友跟我們用一樣的方法實作音樂，他們使用前幾年很紅的黑人抬棺音樂，我們覺得很有效果，所以我們就跟他借這部分的程式。

(4) Mario module:

此模組是用來撥放遊玩遊戲時的背景音樂，我們使用原版馬力歐兄弟的主題曲，輕快的節奏以及本來就屬於電子音，實作出來的效果蠻接近原版的音樂的。以下附上當時的譜。



Buzz_control module:

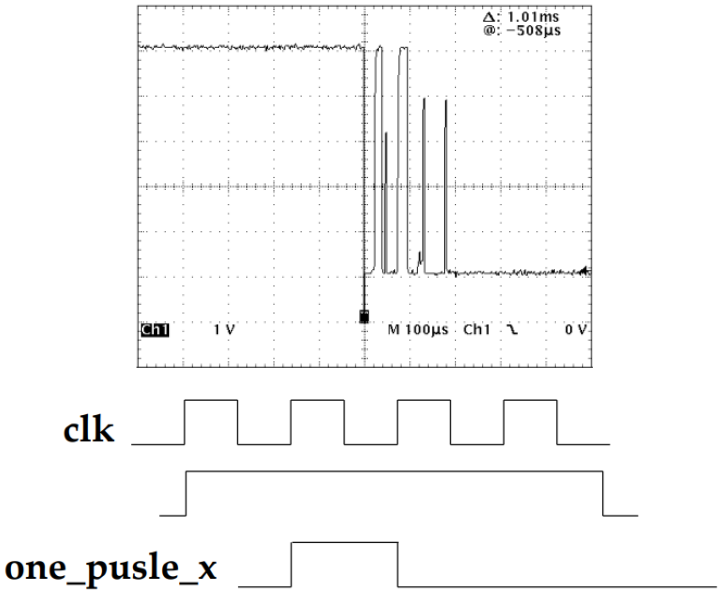
此模組是用來調整音樂音量的模組，當我們把 DIP switch 打開時我們就可以撥放我們的音樂，反之則關閉音樂(使得音量輸出為零，並不是停止音樂)。除了此功能之外，我們可以按調高(調低)音量的按鈕來調整音量，這個功能與我們在 Lab 的實作相同。

Frequency_divider module:

此模組運用在 buzz_control 以及接下來的兩個模組，並且此模組與我們之前討論過的除頻器並無二致，略過不再討論。

One_pulse module:

此模組與之前 Lab 的 one pulse generator 一樣，是為了去除按鈕本身機械元件具有彈簧，會使得按下按鈕的瞬間產生不穩定的高低電位(附圖上)，以及我們希望以按壓下去的動作為觸發單位，也就是說我們並沒有長按調整音量的功能(附圖下)。而這邊實作稍微不同的是我們將之前去除音按鈕產生反覆震盪的電位的 debounce 模組，直接整合進這次的 one pulse 模組。



Ssd_control module:

此模組是用來顯示目前撥放的是什麼音樂。因為要顯示的英文字母超過 4 個(FPGA 版上七段顯示器的數量)，所以我們使用之前 Lab 使用過 scrolling 的顯示器方法，並且我們要隨著狀態去改變目前要顯示的內容。因此我們除了最基本的七段顯示器控制外，還要有一個 ring counter 去決定不同狀態下應該要輸出何種圖型給七段顯示器的控制器。

Ring_counter:

```
48  always@(posedge clk or posedge reset) begin
49      if (reset || (~gaming)) begin
50          M1 <= `M;
51          M2 <= `A;
52          M3 <= `R;
53          M4 <= `I;
54          M5 <= `O;
55          M6 <= 8'b11111111;
56      end
57      else begin
58          M1 <= M2;
59          M2 <= M3;
60          M3 <= M4;
61          M4 <= M5;
62          M5 <= M6;
63          M6 <= M1;
64      end
65  end
```

Discussion

吳璨霖：

在這次的 Final project 中，我負責的是 VGA 的部分，我認為 VGA 的部分非常的需要製作者的大量耐心，以及清晰的思考模式，因為從前面可以看到在計算想要顯示的範圍的時候都需要很完美的將其 valid 的範圍以及 address 的範圍設定好，不能有偏差，若是有偏差選到了不合理的值，甚至會跑出亂碼，使 debug 變得非常的艱難，而且當想要顯示的範圍會隨著遊戲進行而改變的時候那個範圍的界定就需要更加的嚴謹，並且可能需要重複輸出調整 address 好幾次嘗試將取的位置+1+2 會不會使圖片變得比較好看一點。

另一個我認為比較困難的部分就是圖層部份的設計，在這次的 project 中，我嘗試了一陣子才順利的將想要的圖片做好去背並且將多個圖片同時顯示在螢幕上。

而我使用 valid 訊號來規定其顯示圖片的範圍也是有其優點，那就是碰撞的部分比較好實現，我只需要判斷是否 h_cnt 以及 v_cnt 在掃的時候會同時出現兩個圖片的 valid 都是高電位的時候並且重疊的部分都不是要去背掉的部分就好了。

不過在實際上做四個 pipe 的顯示的時候，會出現一點小小的破圖問題，當我在顯示一個 pipe 的時候不管怎麼跑他的 pipe 形狀都會是非常的漂亮，但我將顯示的 pipe 數量變為 4 個的時候，就會出現些許的 pixel 顏色不對，也就是破圖的狀況，後來在經過 address 顯示的調整後，我們改善了大部分的破圖，不過 pipe body 還是會在遊戲進行到一個部分的時候產生些許的破圖，判斷可能是 memory 使用的太滿導致的，不過不影響遊戲體驗。

遊戲進行速度以及鳥的飛行速度的部分，我們也有做一些手腳，往回看可以看到我們其實不是取全部的 bit，這是我們調整以後的結果，在玩遊戲的時候覺得鳥飛行的 state 改變的速度有點太快，因此我就只取上面的幾個 bit 而不取 LSB，這樣就可以讓 state 改變的速度除以二。若是想要改變 pipe 間的間隔以及 pipe 的上下差也是可以很方便的調，就只要分別調整 define 好的 SPACE 以及 INTERVAL 這兩個參數就可以了，若是要改變難度也是相當方便，原本是想要做不同模式可以調整這兩個參數以調整難度，但可惜沒有時間將其實踐出來。

最後是我們的 state 有先互相溝通好要長甚麼樣子，因此最後在跟隊友整合兩邊的 code 的時候就相當的方便，達到了一個有效的分工效果。

張育碩：

這次 project 我負責的部分是背景音樂的部分，記分板以及最後的整合。相較於吳璨霖而言，我做的部分比較沒有那麼燒腦。但是因為雖然音樂的簡譜看似簡單，但是要讓音樂比較好聽，我們必須在每個音符之間加入斷點，但也就是斷點的加入，會使的原本要有的音符數是原來的 2 倍，要把一個一個不同長的音符(有八分音符，十六分音符，休止符之類的)輸入進去，注意斷點且不能出錯，其實也是一件蠻費心力的事。順帶一提，因為我音樂基礎並不是很好，看譜的速度不是很快，因此我請我學過鋼琴的室友幫我寫簡譜，這大大加速了音樂的製作

並且我也在最後的整合部分參與了記分板以及暫停畫面的製作。相對其他部分，原本我其實對 VGA 的操作其實沒有很有把握。不過經過最後為了製作暫停畫面以及記分板讓我更理解 VGA 的操作原理。但是在實作的過程中，我也時常遇到圖片顯示亂碼的狀況，只能再嘗試其他辦法。。並且到最後的部分，bitstream 都要跑很久才能生成出來，有一次我跑出去工作站上廁所再裝一下水，bitstream 都還沒跑好，在接近 deadline 時跑那麼久有時候還蠻絕望的。

除此之外，因為我們的 project 使用到了不少的圖片，也意味著我們要使用更多的 memory，結果在做記分板快結束時，因為我讓現在分數，歷史最佳分數分別二位數字共四個數字，去獨立的運作，還跳出來我們記憶體使用太多，似乎 messages 是說只能用 100 個電容之類的，然後我們用了 129 個，當下真的慌了一下，不過幸好冷靜一下就想出解決的辦法。下次真的要早點動工，不要在壓線做期末專題了。

最後，這算是我第一次做比較大的 final project。一開始因為這個實驗要合成、燒板子之類的，我有點懷疑要怎麼要才能分工的好。我沒想到最後其實能分工的部分真的不少，只要事前先大概溝通部分的規格，把各自的部分做好後其實真的不用花太多時間整合，這真的讓我體會團隊合作的強大（雖然比較困難的部分不是我負責的）。

Conclusion

這次的 final project，我們整合了大多數之前 Lab 使用過的喇叭、鍵盤、VGA 以及滾動的七段顯示器，是實作了一個新的遊戲，也是複習之前的所學。在實作的過程中遇到了許多讓我們感到十分的挫折的困難，但是最後我們大都能找到解決的辦法。最後做出成品時真的有一種說不出的成就感。

除了實作本身，我們也要跟隊友有良好的溝通，才能讓專題進行得更順利。因此我們也學到了除了精進本身的能力以外，好的社交、溝通能力也是我們要培養的目標之一。很開心最後的專題能在最後期限內做完，也做出了我們兩個都滿意的作品，希望這次的專題甚至整個實驗課程使我們產生的成長能在未來的某一天帶給我們幫助，給我們更有深度的見解。

Reference

1. Handouts and Assignments of the logic design lab

   複習我們學過各種外加元件的操作原理以及一些 pin 的接腳。

2. www.convertworld.com

   幫助我們做各種進位制轉換的網站。

3. tw.everyonepiano.com

   Mario 以及 Rock you 鋼琴譜的網站。

4. nicechord.com 五線譜：一次搞懂所有你該知道的、基本的事情

   介紹了許多不同種類音符的網站。

5. https://pages.mtu.edu/~suits/notefreqs.html

   讓我們找到不同音高的音符對應的頻率

6. https://flappybird.ee/

   試玩網路上的 flippy bird。

7. https://www.pngkit.com/view/u2q8o0r5i1i1t4e6_flappy-bird-background-png-banner-stock-flappy-bird/

   各種 flippy bird 飛行時會用到的圖片。