Lab 27
Final Project Report

邏輯設計實驗

組長 108020025 林昱夆

組員 108020022 周昱宏

題目：太鼓達人

Introduction

這次的 final project 實作我們決定設計眾所皆知的「太鼓達人」雙人打擊對戰遊戲，reset 後進入遊戲選單，挑戰的歌曲是「小蘋果」，藉由打擊不同的鼓可選擇不同難易度並進入遊戲。

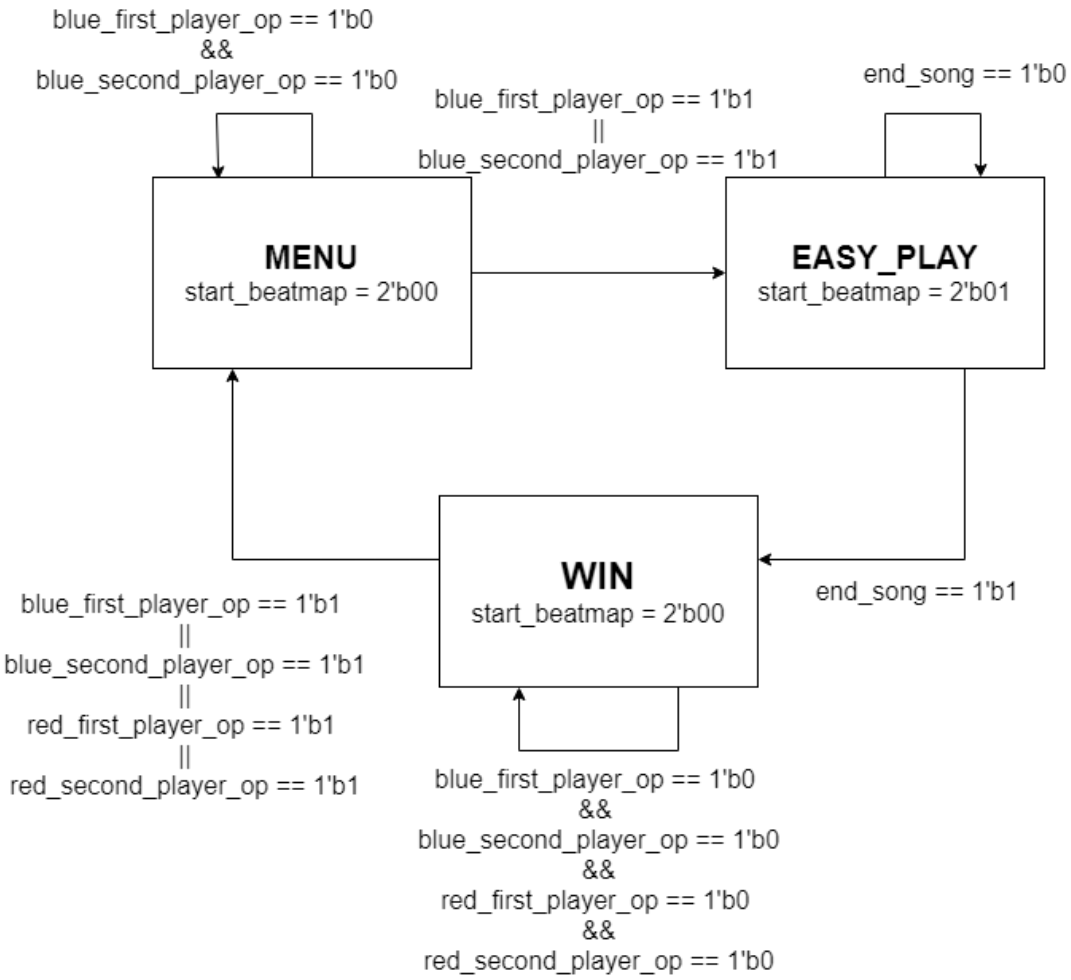
底下是遊戲主畫面，配合著歌曲的音樂與節奏，玩家必須在正確的鼓點打擊才能獲得分數，紅色方塊到達白色框框則要打紅色標示鼓，藍色方塊到達白色框框則要打藍色標示鼓，並依離白色框框的距離遠近給於不同分數（完美 100 分、良 50 分以及失誤 0 分）。

歌曲結束後統計兩人分數，分數高的人即獲勝，並打擊任一鼓跳回遊戲選單。

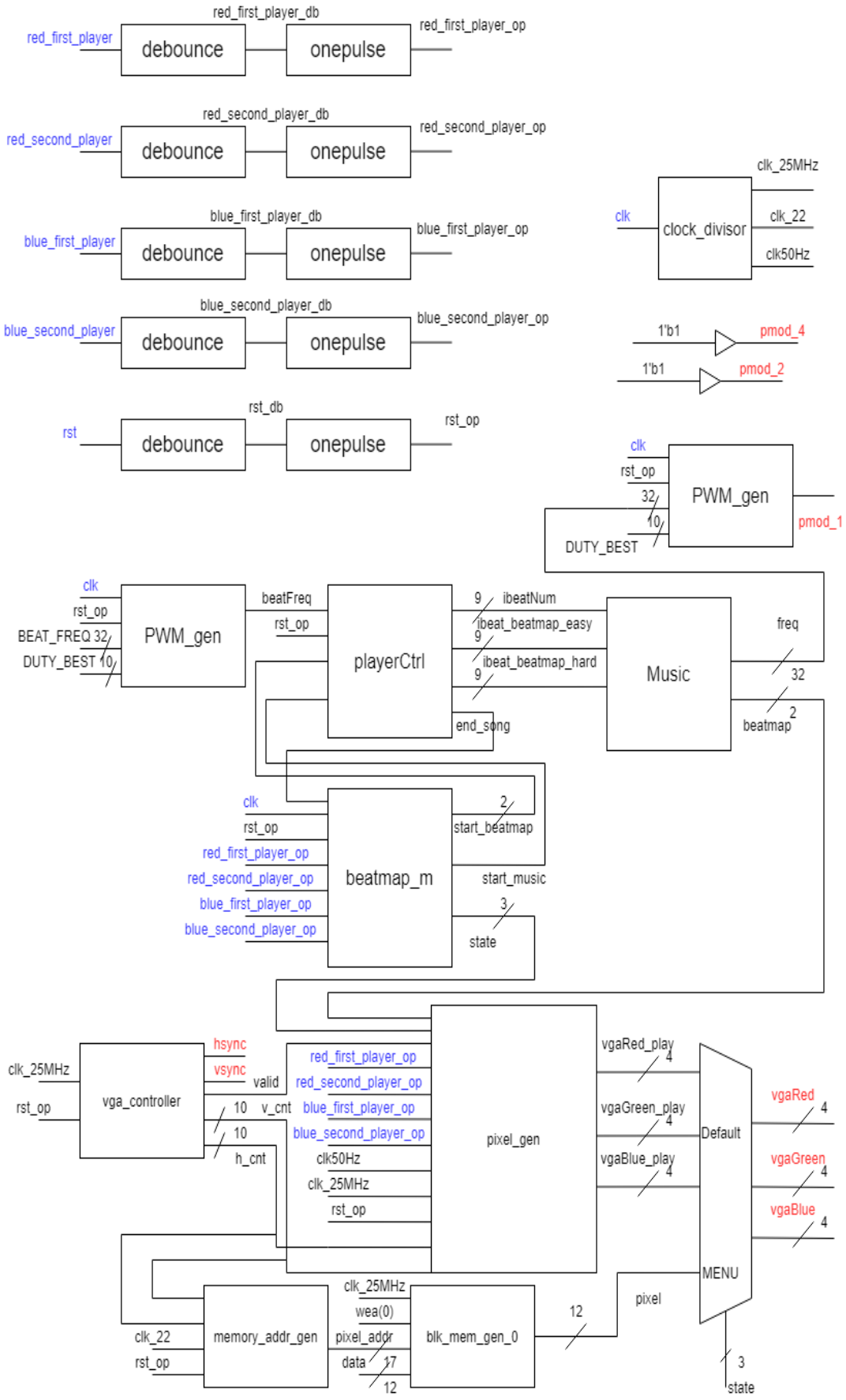
Motivation

經過小組討論後，我們決定要做遊玩類型的 fpga 實作，因此我們事先去電子材料行瀏覽各種不同的模組，看到「陶瓷震動傳感器」模組想到可以試試太鼓達人的遊戲，並查找此模組的輸出訊號是類比還是數位以及使用方式等相關資料，確認後購買小鼓，並了解線上太鼓達人的遊戲運行以及遊玩方式，收集好資料及材料後即進行製作。

State transition diagram



Block diagram



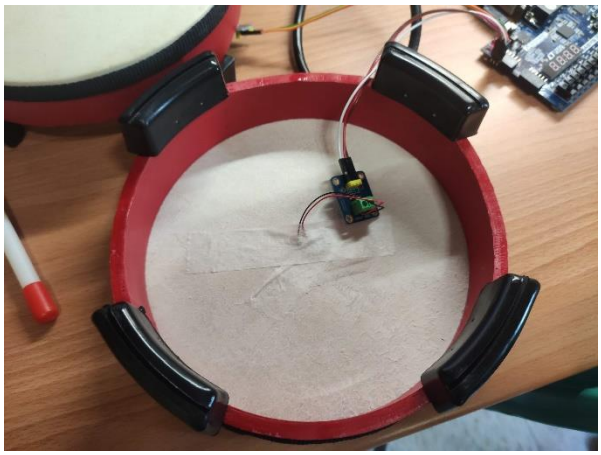
System specification

1。太鼓訊號輸入:

本次作業中，我們購買了 4 組鼓，並購買 4 個陶瓷震動傳感器，此傳感器只有三個 pin，正負極以及訊號輸出，選擇好 fpga 上的 port 後，將陶瓷片貼在鼓的背面就可以成功使用了。

下圖為陶瓷震動傳感器

下圖為感測器與 fpga 以及鼓相接的情形



2。聲音模組:

為了直接測試，我們直接利用在 lab5 用到的音樂模組來當作音樂，比較不同的是，為了要讓音樂有開始和結束，我有設計一個 start_music 的輸入信號來讓音樂開始，以及音樂結束後會輸出一個 finish(會接回 state_transition 中)，再來將 top 中的輸入與輸出的名稱調整好後，就成為了我們 project 中的一部份。

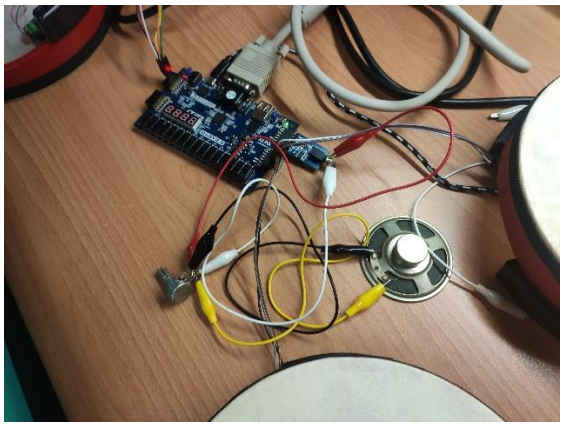
```
module PlayerCtrl (
    input clk,
    input reset,
    input start_music,
    input [1:0] start_beatmap,
    output reg [8:0] ibeat_music, ibeat_beatmap_easy, ibeat_beatmap_hard,
    output reg finish
);
parameter BEATLEAGTH = 300;

always @(posedge clk, posedge reset) begin
    if (reset) begin
        finish <= 1'b0;
        ibeat_music <= 0;
    end
    else if (start_music && ibeat_music < BEATLEAGTH) begin
        ibeat_music <= ibeat_music + 1;
        finish <= 0;
    end
    else if (start_music && ibeat_music == BEATLEAGTH) begin
        finish <= 1;
        ibeat_music <= 0;
    end
    else begin
        ibeat_music <= 0;
        finish <= 0;
    end
end
```

音樂延遲部分，我利用 state 當作開始計時的信號，計時到達定值後，start_music=1

```
else if(state == EASY_PLAY || state == TRICKY_PLAY) begin
    if(count <= 240_000_000) begin
        start_music <= 0;
        count <= count + 1;
    end
    else start_music <= 1;
end
else begin
    count <= 0;
    start_music <= 0;
end
end
```

下圖為音樂模組



3。螢幕輸出:

因為在不同的 state 中，vga 的輸出 module 不同，因此我們在 top 中利

用 state 來控制 vgaRed， vgaGreen， vgaBlue 該輸出哪一個 module 產生的 RGB，其中 pixel 是主畫面的 RGB，vgaXXX_play 是在打鼓畫面的 RGB。

```
always @(*) begin
    if(state == MENU) begin
        {vgaRed, vgaGreen, vgaBlue} = (valid == 1'b1) ? pixel : 12'h0;
    end
    else begin
        vgaRed = vgaRed_play;
        vgaGreen = vgaGreen_play;
        vgaBlue = vgaBlue_play;
    end

end
```

而我們分別用了將外部圖片輸出以及用 fpga 畫圖的技巧

1。主畫面:在主畫面時，我們利用 lab6 中的阿姆姆螢幕輸出模組，並將向上移動的功能移除，並且先在小畫家將進入遊戲的資訊畫上，這樣主畫面的螢幕輸出就完成了

```
module mem_addr_gen(
    input clk,
    input rst,
    input [9:0] h_cnt,
    input [9:0] v_cnt,
    output [16:0] pixel_addr
    );

    reg [7:0] position;

    assign pixel_addr = ((h_cnt>>1)+320*(v_cnt>>1)+ position*320 )% 76800;

    always @ (posedge clk or posedge rst) begin
        if(rst)
            position <= 0;
        else
            position <= 0;
    end

endmodule
```

2。打擊畫面:而在打擊譜面的畫面則是用 fpga 畫出，在 pixel_gen 中，我們首先優先畫出音符(就是要敲擊的符號)，利用一個陣列 position 來表示音符的位置，接著判斷每個 pixel 的四周是否有音符存在，如果有的話，我們就將那一格畫上該音符的顏色。

```
if(v_cnt <= 210 && v_cnt >= 170 && (position_R1[h_cnt-1] || position_R1[h_cnt-2] || position_R1[h_cnt-3] || position_R1[h_cnt-4] || posit
 || position_R1[h_cnt-6] || position_R1[h_cnt-7] || position_R1[h_cnt-8] || position_R1[h_cnt-9] || position_R1[h_cnt-10]
 || position_R1[h_cnt-11] || position_R1[h_cnt-12] || position_R1[h_cnt-13] || position_R1[h_cnt-14] || position_R1[h_cnt-15]
 || position_R1[h_cnt-16] || position_R1[h_cnt-17] || position_R1[h_cnt-18] || position_R1[h_cnt-19] || position_R1[h_cnt-20] ||
position_R1[h_cnt]
 || position_R1[h_cnt+1] || position_R1[h_cnt+2] || position_R1[h_cnt+3] || position_R1[h_cnt+4] || position_R1[h_cnt+5] ||
position_R1[h_cnt+6] || position_R1[h_cnt+7] || position_R1[h_cnt+8] || position_R1[h_cnt+9] || position_R1[h_cnt+10
] || position_R1[h_cnt+11] || position_R1[h_cnt+12] || position_R1[h_cnt+13] || position_R1[h_cnt+14] || position_R1[h_cnt+15
] || position_R1[h_cnt+16] || position_R1[h_cnt+17] || position_R1[h_cnt+18] || position_R1[h_cnt+19] || position_R1[h_cnt+20]
))
    {vgaRed, vgaGreen, vgaBlue} = 12'hf33;
```

例如:position_R1[300]==1(在 300 的位置有存在紅色音符)，當 v_cnt==200 && h_cnt==320 時(v_cnt 與 h_cnt 代表當下螢幕上長與寬的位置)，則因為 v_cnt 有在範圍內，且 position_R1[h_cnt-20]==1，所以這個點就必須輸出紅色。

其他顏色音符輸出也是如此運作

接著是軌道部分，判斷 v_cnt 是否在範圍內即可，並且如果 h_cnt 在要求範圍內，就畫成白色，當作敲擊的視覺範圍，其他情況則輸出偏黑的灰

色，當作軌道。

```
else if((v_cnt <= 330 && v_cnt >= 250) || v_cnt <= 230 && v_cnt >= 150) begin
  if(h_cnt == 80 || h_cnt == 120)
    {vgaRed, vgaGreen, vgaBlue} = 12'haaa;
  else
    {vgaRed, vgaGreen, vgaBlue} = 12'h333;
```

最後是分數部分，我們利用類似在做七段顯示器的方法來畫出數字，首先將數字轉成 10 進位，再將位數轉成七段顯示，最後在 pixel_gen 中判斷畫圖位置

```
else if(v_cnt >= position_v_s1 && v_cnt <= position_v_s1 + 21 && h_cnt >= one_position_h && h_cnt <= one_position_h + 11) begin
  if(!seven_seg_one_s1[6] && v_cnt >= position_v_s1 + 9 && v_cnt <= position_v_s1 + 10 && h_cnt >= one_position_h + 2 && h_cnt <= one_positi
      {vgaRed, vgaGreen, vgaBlue} = 12'h000;
  else if(!seven_seg_one_s1[5] && v_cnt >= position_v_s1 + 2 && v_cnt <= position_v_s1 + 9 && h_cnt >= one_position_h && h_cnt <= one_positi
      {vgaRed, vgaGreen, vgaBlue} = 12'h000;
  else if(!seven_seg_one_s1[4] && v_cnt >= position_v_s1 + 12 && v_cnt <= position_v_s1 + 19 && h_cnt >= one_position_h && h_cnt <= one_posi
      {vgaRed, vgaGreen, vgaBlue} = 12'h000;
  else if(!seven_seg_one_s1[3] && v_cnt >= position_v_s1 + 20 && v_cnt <= position_v_s1 + 21 && h_cnt >= one_position_h + 2 && h_cnt <= one_
      {vgaRed, vgaGreen, vgaBlue} = 12'h000;
  else if(!seven_seg_one_s1[2] && v_cnt >= position_v_s1 + 12 && v_cnt <= position_v_s1 + 19 && h_cnt >= one_position_h + 10 && h_cnt <= one
      {vgaRed, vgaGreen, vgaBlue} = 12'h000;
  else if(!seven_seg_one_s1[1] && v_cnt >= position_v_s1 + 2 && v_cnt <= position_v_s1 + 9 && h_cnt >= one_position_h + 10 && h_cnt <= one_p
      {vgaRed, vgaGreen, vgaBlue} = 12'h000;
  else if(!seven_seg_one_s1[0] && v_cnt >= position_v_s1 && v_cnt <= position_v_s1 + 1 && h_cnt >= one_position_h + 2 && h_cnt <= one_positi
      {vgaRed, vgaGreen, vgaBlue} = 12'h000;
  else
      {vgaRed, vgaGreen, vgaBlue} = 12'h0ff;
end
```

圖中，position_v_s1 是一個參數，代表 player1 分數的每個數字的最左上角的 v_cnt 位置，而 one_position_h 則是指第一位數字最左上角的 h_cnt 位置，seven_seg_one_s1[i]則是代表 player1 的第一位數字七段顯示的第 i 段位置，有了這些後，就用這些資訊來寫出我們該在哪邊畫出數字的判斷式。而每個玩家有六位數，且有兩個玩家，所以在參數設計時，one two 代表位數，s1 s2 代表玩家。

3。獲勝畫面:

在遊戲結束後，我們會將 state 切換至 WIN，在 WIN 時我們就在分數較高的一方畫上一個 WIN

```
if(state == WIN) begin
    //w print
    if(v_cnt >= win_position_v && v_cnt <= win_position_v + 60 && h_cnt >= w_position_h && h_cnt <= w_position_h + 55)begin

        if(v_cnt >= win_position_v && v_cnt <= win_position_v + 52 && h_cnt >= w_position_h && h_cnt <= w_position_h + 7)
            {vgaRed, vgaGreen, vgaBlue} = 12'h000;
        else if(v_cnt >= win_position_v && v_cnt <= win_position_v + 52 && h_cnt >= w_position_h + 24 && h_cnt <= w_position_h + 31)
            {vgaRed, vgaGreen, vgaBlue} = 12'h000;
        else if(v_cnt >= win_position_v && v_cnt <= win_position_v + 52 && h_cnt >= w_position_h + 48 && h_cnt <= w_position_h + 55)
            {vgaRed, vgaGreen, vgaBlue} = 12'h000;
        else if(v_cnt >= win_position_v + 53 && v_cnt <= win_position_v + 60 && h_cnt >= w_position_h && h_cnt <= w_position_h + 55)
            {vgaRed, vgaGreen, vgaBlue} = 12'h000;
        else
            {vgaRed, vgaGreen, vgaBlue} = 12'h0ff;


    end
    //i print
    else if(v_cnt >= win_position_v && v_cnt <= win_position_v + 60 && h_cnt >= i_position_h && h_cnt <= i_position_h + 55)begin

        if(v_cnt >= win_position_v && v_cnt <= win_position_v + 7 && h_cnt >= i_position_h && h_cnt <= i_position_h + 55)
            {vgaRed, vgaGreen, vgaBlue} = 12'h000;
        else if(v_cnt >= win_position_v + 53 && v_cnt <= win_position_v + 60 && h_cnt >= i_position_h && h_cnt <= i_position_h + 55)
            {vgaRed, vgaGreen, vgaBlue} = 12'h000;
        else if(v_cnt >= win_position_v && v_cnt <= win_position_v + 60 && h_cnt >= i_position_h + 24 && h_cnt <= i_position_h + 31)
            {vgaRed, vgaGreen, vgaBlue} = 12'h000;
        else
            {vgaRed, vgaGreen, vgaBlue} = 12'h0ff;
```

```
//n print
else if(v_cnt >= win_position_v && v_cnt <= win_position_v + 60 && h_cnt >= n_position_h && h_cnt <= n_position_h + 55)begin

    if(v_cnt >= win_position_v && v_cnt <= win_position_v + 60 && h_cnt >= n_position_h && h_cnt <= n_position_h + 7)
        {vgaRed, vgaGreen, vgaBlue} = 12'h000;
    else if(v_cnt >= win_position_v && v_cnt <= win_position_v + 60 && h_cnt >= n_position_h + 48 && h_cnt <= n_position_h + 55)
        {vgaRed, vgaGreen, vgaBlue} = 12'h000;
    else if(v_cnt >= win_position_v && v_cnt <= win_position_v + 60 && h_cnt >= n_position_h + 8 && h_cnt <= n_position_h + 55) be
        if(h_cnt >= (v_cnt - win_position_v) + n_position_h + 8 && h_cnt <= (v_cnt - win_position_v) + n_position_h + 15)
            {vgaRed, vgaGreen, vgaBlue} = 12'h000;
        else
            {vgaRed, vgaGreen, vgaBlue} = 12'h0ff;
    end
    else
        {vgaRed, vgaGreen, vgaBlue} = 12'h0ff;
end
```

4。state 轉換:

R1 R2 B1 B2 分別代表 player1 的紅色藍色 player2 的紅色藍色，而 start_beatmap 是我要用來開始譜面的信號，start_music 則是音樂開始的信號，end_song 是音樂結束的信號。

首先在 MENU(主畫面時)，如果按下藍色就會進入簡單模式，按下紅色就會進入困難模式。

```
always @(*) begin
    if(state == MENU) begin
        start_beatmap = 2'b00;
        if(B1 || B2) begin
            next_state = EASY_PLAY;
        end
        else if(R1 || R2) begin
            next_state = TRICKY_PLAY;
        end
        else
            next_state = state;
    end
```

在困難或是簡單模式中，將自己所對應的 start_beatmap 輸出 1，且只要 end_song 了，就進入 WIN

```
else if(state == EASY_PLAY) begin
    start_beatmap = 2'b01;
    if(end_song) begin
        next_state = WIN;
    end
    else begin
        next_state = state;
    end
end
else if(state == TRICKY_PLAY) begin
    start_beatmap = 2'b10;
    if(end_song) begin
        next_state = WIN;
    end
    else begin
        next_state = state;
    end
end
```

過關畫面時，只要再敲一下就會進入主畫面，設計時因為每次敲擊都會出

現，跳過主畫面，直接進入打擊畫面的狀況，所以我這邊設了會先進入 3'b111 的 state，等到 count111 到達定值才會回到 MENU。

```
else if(state == WIN) begin
    start_beatmap = 2'b00;
    if(B1 || B2 || R1 || R2)
        next_state = 3'b111;
    else
        next_state = state;
end
else if(state == 3'b111) begin
    if(count111 == 7'd100) begin
        next_state = MENU;
        next_count111 = 0;
        end
    else begin
        next_count111 = count111 + 1;
        next_state = state;
        end
```

5。譜面設計:

本次作業花最久時間設計的地方，因為參考過真實太鼓達人的譜面設計，因此決定用類似的方法應該會比較彈性，首先，我在音樂模組中複製一份音樂的譜，並將輸出 tone 改成輸出 beatmap，如圖:(ibeat_beatmap_easy 就像音樂模組中的 ibeatNum，用來決定該輸出甚麼顏色的音符，參數 N B R 代表 None Red Blue)

```
always @(*) begin
    if(state == 3'b010)begin
    case (ibeat_beatmap_easy)      // 1/4 beat
        9'd0 : beatmap = N; //3

        9'd1 : beatmap = R;
        9'd2 : beatmap = N;
        9'd3 : beatmap = N;
        9'd4 : beatmap = N; //1
        9'd5 : beatmap = R;
        9'd6 : beatmap = N;
        9'd7 : beatmap = N;
        9'd8 : beatmap = N; //2
        9'd9 : beatmap = R;
        9'd10 : beatmap = N;
        9'd11 : beatmap = N;
        9'd12 : beatmap = N;    //6-
        9'd13 : beatmap = R;
        9'd14 : beatmap = N;
        9'd15 : beatmap = N;
        9'd16 : beatmap = N;

        9'd17 : beatmap = R;
        9'd18 : beatmap = N;
        9'd19 : beatmap = N;
        9'd20 : beatmap = N;
        9'd21 : beatmap = R;
        9'd22 : beatmap = N;
        9'd23 : beatmap = N;
```

操控 ibeat_beatmap 是跟音樂放在一起的，變換的快慢與音樂一樣，state 一轉換就會開始，與音樂不同步，start_beatmap 是從

state_transition 接來的。

```
always @(posedge clk, posedge reset) begin
    if (reset) begin
        ibeat_beatmap_easy <= 0;
    end
    else if (start_beatmap[0] && ibeat_beatmap_easy < BEATLEAGTH) begin
        ibeat_beatmap_easy <= ibeat_beatmap_easy + 1;
    end
    else if (start_beatmap[0] && ibeat_beatmap_easy == BEATLEAGTH) begin
    end
    else begin
        ibeat_beatmap_easy <= 0;
    end
end
```

而這個 beatmap 我們會接到 pixel_gen 中做處理。（以下應該可以拉出來寫一個 moudle 比較好）

在 pixel_gen 中，我設計了四個 661bits 的陣列，用來儲存 p1 紅色(position_R1)、藍色(position_B1)、p2 紅色(position_R2)、藍色(position_B2)的音符位置，例如:position_R1[200]==1 代表 p1 在 200 這個位置有紅色音符。

接著，我將它們每次 clk 都往左 shift 一個 bit，這樣就會有音符往左的效果，接著再去設定判定，假如敲擊的訊號進來，我就看判定範圍內是否有音符，如果有的話，根據它所在的區域來決定分數增加以及音符的位置消失。

從 code 來看，下圖是判斷在中心左方 21~30 格是否有紅色音符，有的話，hit_R_bad = 2’b10，意思是這個地方的判定是 bad，用 2bits 來計是因為 10 是代表偏左的 bad，01 的話則是偏右的 bad。

```
module check_in_range(
    input [660:0] position_B,
    input [660:0] position_R,
    output reg [1:0] hit_R_bad,
    output reg [1:0] hit_B_bad,
    output reg [1:0] hit_R_good,
    output reg [1:0] hit_B_good,
    output reg hit_R_perfect,
    output reg hit_B_perfect,
    output reg miss
);

parameter Center = 100;


always @* begin
    if (position_R[Center-21] || position_R[Center-22] || position_R[Center-23] || position_R[Center-24] || position_R[Center-25]
        || position_R[Center-26] || position_R[Center-27] || position_R[Center-28] || position_R[Center-29] || position_R[Center-30] )
        begin
            hit_R_bad = 2'b10;
            hit_B_bad = 2'b00;
            hit_R_good = 2'b00;
            hit_B_good = 2'b00;
            hit_R_perfect = 1'b0;
            hit_B_perfect = 1'b0;
        end
```

在來到真正的譜面移動中，我們得到 R1(p1 的 Red)輸入後先從偏右方的 bad(R1_bad[0])判斷，如果有的話，combo 歸零，bad1(p1 的 bad)計次+1，並將 position_R1 的右 bad 位置清除，其餘位置繼續往右 shift 一格，其他還有(左右)good、perfect 也是一樣，那如果遇到藍色呢?那就全部往右 shift，將 combe 歸 0，然後 p2 也是做一樣的事就可以了!

```
//check Red hit
else if(R1) begin
  //check Red
  if(R1_bad[0]) begin
    combo1 <= 0;
    bad1 <= bad1 + 1'b1;
    position_R1[129:120] <= 10'b0;
    position_B1[129:120] <= 10'b0;
    position_R1[660:130] <= position_R1[660:130]>>1;
    position_B1[660:130] <= position_B1[660:130]>>1;
  end
  else if (R1_bad[1]) begin
    combo1 <= 1'b0;
    bad1 <= bad1 + 1'b1;
    position_R1[78:69] <= 10'b0;
    position_B1[78:69] <= 10'b0;
    position_R1[660:79] <= position_R1[660:79]>>1;
    position_B1[660:79] <= position_B1[660:79]>>1;
  end
```

```
//check Blue
else if(B1_bad[0]) begin
    combo1 <= 1'b0;
    position_R1 <= position_R1>>1;
    position_B1 <= position_B1>>1;
end
else if (B1_bad[1]) begin
    combo1 <= 1'b0;
    position_R1 <= position_R1>>1;
    position_B1 <= position_B1>>1;
end
else if (B1_good[0]) begin
    combo1 <= 1'b0;
```

再來是將譜輸入至 position 中，我用一個 counter 來計時，讓它跟音樂的節奏等速，接著只要在每次 counter 數到一定的值就將譜輸入至 position[650]中，當時忘記其他位置也要繼續移動，不然每次讀譜都會停頓那麼一小下，不過可能是 clk 很快所以看不太出來

```
always @* begin
    if(beatmap == 2'b01) begin
        next_position_R = 1'b1;
        next_position_B = 1'b0;
    end
    else if(beatmap == 2'b10) begin
        next_position_R = 1'b0;
        next_position_B = 1'b1;
    end
    else begin
        next_position_R = 1'b0;
        next_position_B = 1'b0;
    end
end
```

```
else begin
    if(count == 10'd30)begin
        position_R2[650] <= next_position_R;
        position_B2[650] <= next_position_B;
    end
```

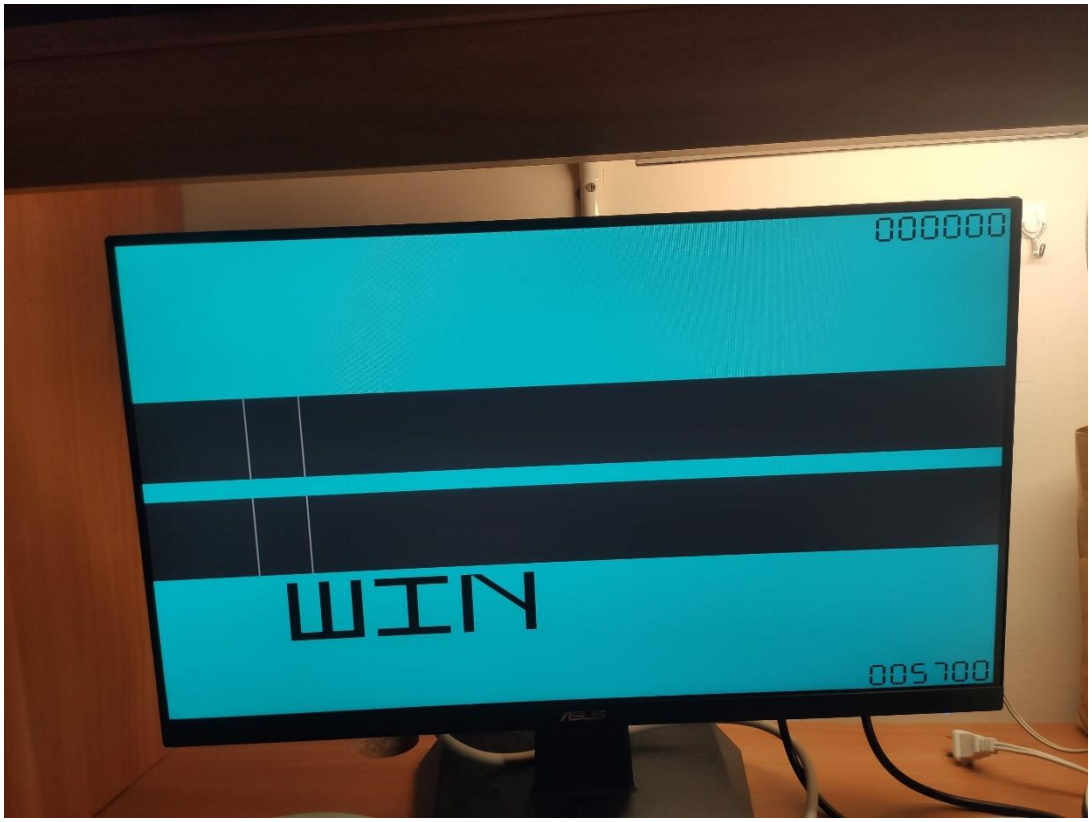
最後是延長敲擊輸入的 pulse，因為原本的 one pulse 信號時間太短，導致有時敲擊會沒有反應，所以我做一個延長信號的工具，當信號(hit_R1)輸入時，將 R1 設成 1，並在 counter 到達某值後才回來 0。

```
always @(posedge clk_25MHz) begin
    if(hit_R1) begin
        R1 <= 1'b1;
        count50_1 <= 0;
    end
    else if(hit_B1) begin
        B1 <= 1'b1;
        count50_1 <= 0;
    end
    else if(count50_1 < 20'd500000 ) begin
        count50_1 <= count50_1 + 1;
    end
    else begin
        R1 <= 1'b0;
        B1 <= 1'b0;
    end
```

Experimental result

初始畫面基本上沒有太大的問題，除了 Tricky 拼錯以外。

遊戲的過程也沒有問題，分數也正常顯示，可惜沒有加上 combo，combo 分數加成也沒用呀!



獲勝畫面沒有問題，但是這次 project 最大的錯誤就是，原本設計是敲一下會回到主畫面，結果卻會直接進入下一首歌，當時判斷可能是敲擊的 pulse 太長，但是就算延遲進入主畫面，還是會馬上進入下一首。是這次的 bug 之一。

Conclusion

這算是我們第一個從購買硬體、設計軟體，一直到解決各種問題的project，以下是我們學習到的收穫：

1. 學習如何掌控預算選擇有實際幫助的材料
2. 學習優化寫法減少資源利用
3. 學習在實體方面遇到困難要如何解決、如導線斷掉
4. 了解 coding style 的好處
5. 了解成員間的溝通討論比自己埋頭苦幹實作更有效率

改進的空間：

1. 把 Tricky 版本實作進去遊戲
2. 增加鼓種如連打(黃色方塊)
3. 分數顯示可放大一點並放在明顯的位置
4. 增加更多歌曲
5. 控制照片大小,增強美術部分
6. 更好的演算法, 解決空間不足的問題
7. 獲勝畫面顯示各種判定打擊次數
8. 增加過關計分條
9. 更快的 coding 能力,不然要做到上面這麼多東西一定會花更多時間