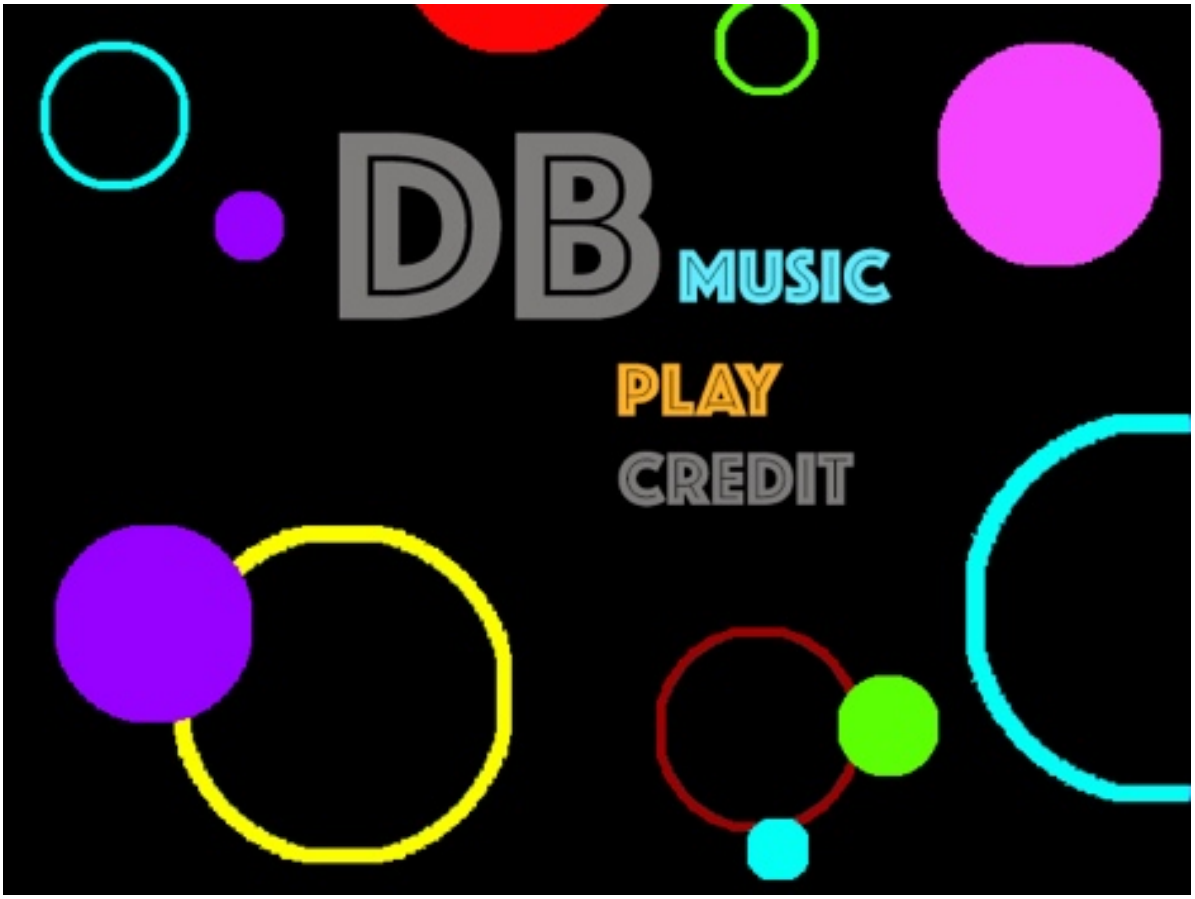
硬體實驗與設計
資工 103062161 張緣彩
資工 103062340 許博皓
8/1/2016

Final Project Report

組員：

張緣彩 103062161
許博皓 103062340

作品題目： DB music player



作品介紹：

我們都對音樂播放器很有興趣，所以就決定要做一個利用硬體來製作一個播放器，並且命名為 DB music。DB music 原本只是內存 3 首歌，也就是兩隻老虎，生日歌和 Maroon 5 的 PayPhone。也沒有 Main Menu，只

是一開始就直接等待播放。但是在實作過程，我們發現我們完成速度很快，所以我們就決定將 DB music 的功能升級，並製作了 Main Menu 以方便使用者很直覺的能使用。功能升級包括的加入第四首歌(Through the Arbor)，並且擁有雙聲道輸出功能，已達到合音效果。雖然我們的實作作品——DB music 的並沒有遊戲性質，但卻能享受音樂的美妙與欣賞精心設計的圖畫已達到放鬆心情。

板子上面硬體的使用方法：

我們先從輸入開始介紹，首先

- SWITCH：DB music 用到的 switch 包括，
  - Sw15 和 Sw14，可以控制音樂的 3 種速度，2'b00 是 0.5x，2'b01 是 1x 與 2'b11 是 2x。
  - Sw13 和 Sw12，可以控制音樂的 4 種音質，2'b00 是 25%，2'b01 是 50%，2'b10 是 75% 與 2'b11 是 100%。
  - Sw1，可以控制音樂是否循環播放（Repeat mode）。
  - Sw0，可以控制靜音模式（Mute mode）。
- BUTTON: DB music 用到的 button 包括，
  - BTNU 是 reset 訊號。
  - BTNC 是 play 和 pause 的訊號。
  - BTNL 是播放上一首歌（Previous）。
  - BTNR 是播放下一首歌（Next）。
- KEYBOARD：DB music 用到的 keyboard 按鈕包括，
  - ESC 是回到上一頁的 Main Menu，例如從 CREDIT 頁面回到 Main Menu。
  - ENTER 是進入 PLAY mode 還是 CREDIT mode。
  - UP／DOWN arrow 是用來選擇 PLAY mode 還是 CREDIT mode。
  - F key 是將音樂往前打快（Fast Forward）。
  - B key 是將音樂往回打快（Back Forward）。

接著就是輸出的介紹，首先

- VGA：VGA 的輸出最主要的就是顯示 Main Menu 和 MV。
- AUDIO：Audio 我們使用了 JB 和 JC 兩個 amplifier，在第一，二和三首音樂，兩個 amplifier 的輸出是一樣，但是在第四首音樂 JB 會輸出鋼琴譜的右手，JC 會輸出鋼琴譜的左手，所以會產生合音的效果。
- LED：會輸出與音調對應的音調數，例如 C 會輸出 7'b0000_001

- LD15～LD9 是與 JC amplifier 對應，例如 JC 現在輸出的音調是 D，LD15 到 LD11 會是 0，LD10 和 LD9 會是 1。
- LD0～LD6 是與 JB amplifier 對應，同上的解釋。
- 我們對音調的 decode 是
  - C 7'b0000_001
  - D 7'b0000_011
  - E 7'b0000_111
  - F 7'b0001_111
  - G 7'b0011_111
  - A 7'b0111_111
  - B 7'b1111_111
  - 沒調就是 7'b0000_000

• 7 SEGMENT DISPLAY：會現在是在的狀態的跑馬燈，例如 HELLO，PLAY 和 PAUSE
  - 剛開始的時候會顯示 HELLO，當音樂開始播放會顯示 PLAY，當音樂暫停會顯示 PAUSE

功能描述：

當已開啟板子，外接的螢幕就會顯示一下照片的 Main Menu。

圖一

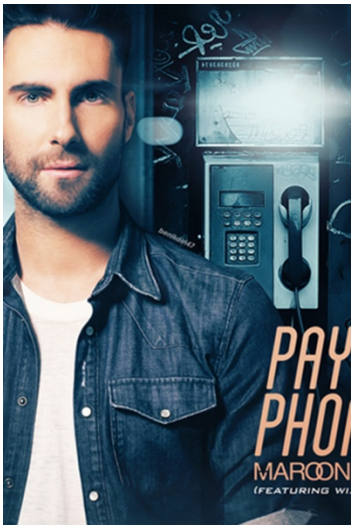
圖二

在照片裡，我們很清楚看到照片（圖一）裡的 PLAY 是黃色的，就代表我們現在選擇是 PLAY，如果按 ENTER 就會進入 PLAY mode。如果按 UP ARROW 或者是 DOWN ARROW，CREDIT 就會變成黃色，PLAY 就會變回灰色，照片（圖二）。如果在 CREDIT 按下 ENTER，就會進入 CREDIT mode。如果按 UP ARROW 或者是 DOWN ARROW，PLAY 又會變成黃，以此循環。此外，7 SEGMENT DISPLAY 會顯示"HELLO"字樣的跑馬燈。在 Main Menu，button 除了 reset 的按鈕外，其他按鈕皆為沒有功能（防呆）。接著我們就會介紹 PLAY mode 與 CREDIT mode 裡嗎有什麼。

PLAY mode：

圖一

圖二



圖三

圖四

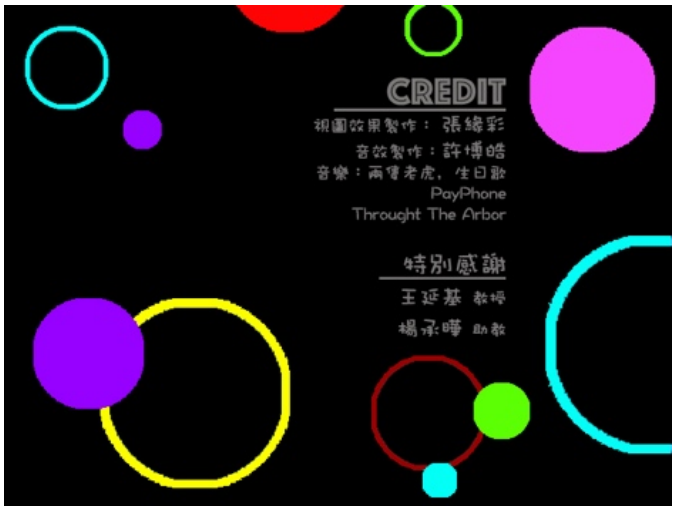
當進入 PLAY mode 時，就會顯示第一首音樂的照片（圖一），7 SEGMENT DISPLAY 還是一樣顯示“HELLO”的跑馬燈，當再此按下 PLAY / PAUSE 的 button，音樂就會開始播放，7 SEGMENT DISPLAY 就會顯示“PLAY”的跑馬燈。如果再次按下 PLAY / PAUSE 的 button，音樂就會停下來，7 SEGMENT DISPLAY 會顯示“PAUSE”的跑馬燈。如果第一首音樂播完，就會跳到下一首音樂繼續播放。如果第一首音樂正在播放，7 SEGMENT DISPLAY 顯示“PLAY”，這時按下 NEXT 或者是 PREV 的 button，就會馬上跳到下一首音樂活著回到上一首音樂繼續播放（從頭播放）。如果音樂是 PAUSE，7 SEGMENT DISPLAY 顯示“PAUSE”，這時按下 NEXT 或者是 PREV，就會馬上跳到下一首音樂或者回到上一首音樂等待 PLAY / PAUSE 的 button 按下才會開始播放音樂（從頭播放），當 PLAY / PAUSE 還沒被按下，7 SEGMENT DISPLAY 還是顯示“PAUSE”。

- 第一和第二首音樂只會顯示照片（圖一和圖二）
- 第三首音樂會往上捲動（圖三），在 PLAY 照片會捲動，在 PAUSE 照片則會與音樂一樣停止捲動
- 第四首音樂會往左捲動（圖四），在 PLAY 照片會捲動，在 PAUSE 照片則會與音樂一樣停止捲動
- 圖一和圖二是用 Photoshop 製作
- 圖三是上網下載

- 圖四是上網下載，Photoshop 修成對稱

CREDIT mode：

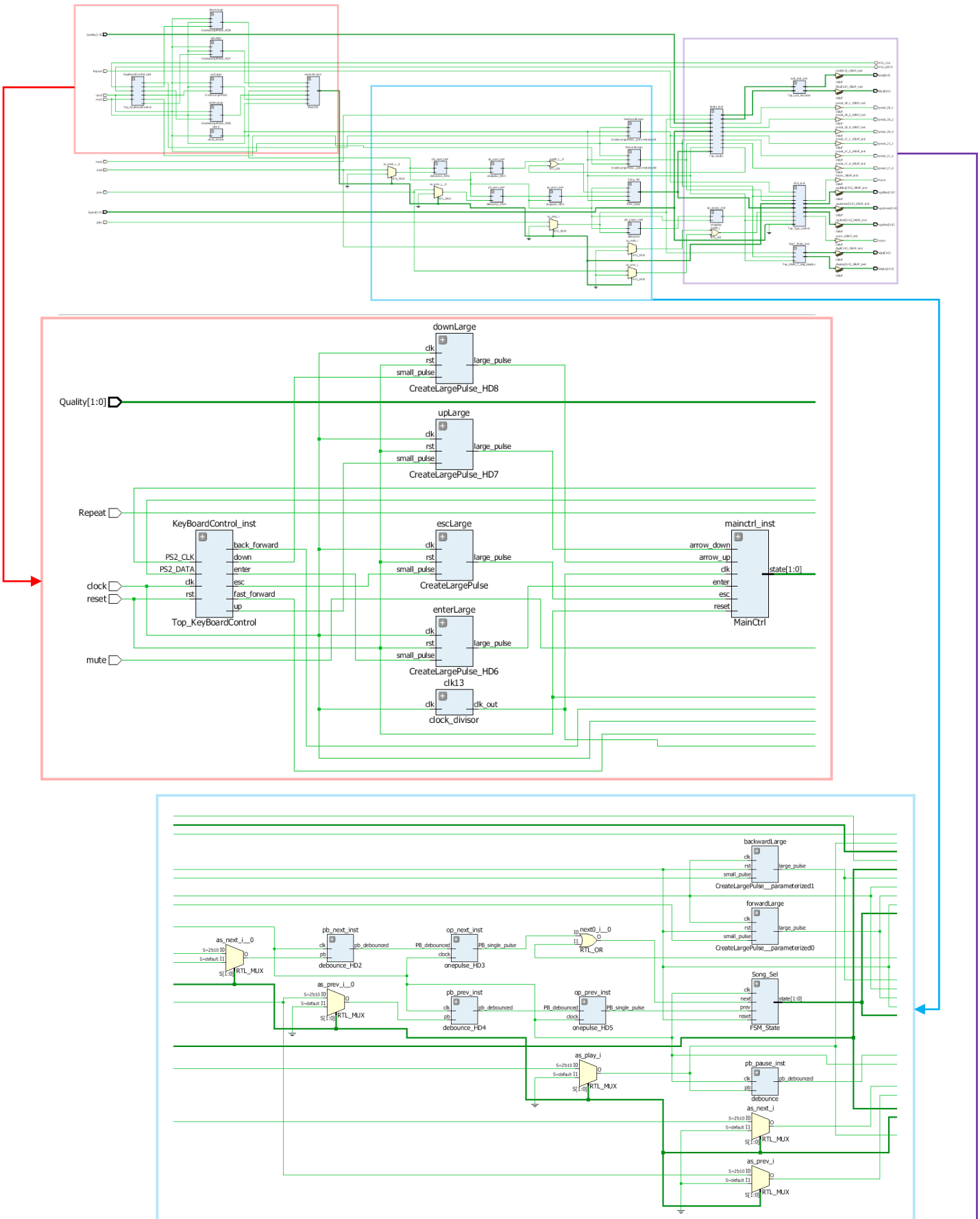
圖一

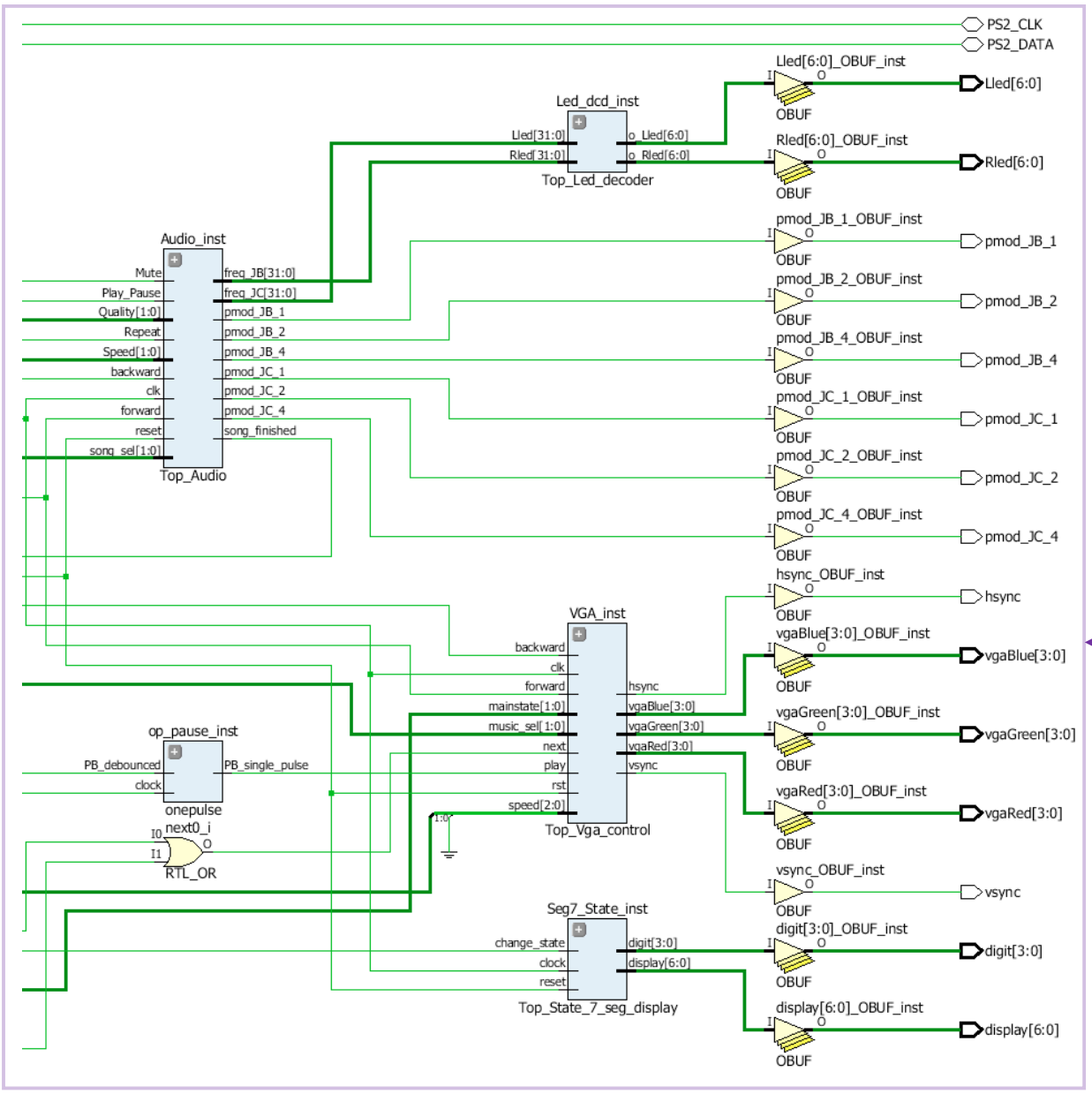


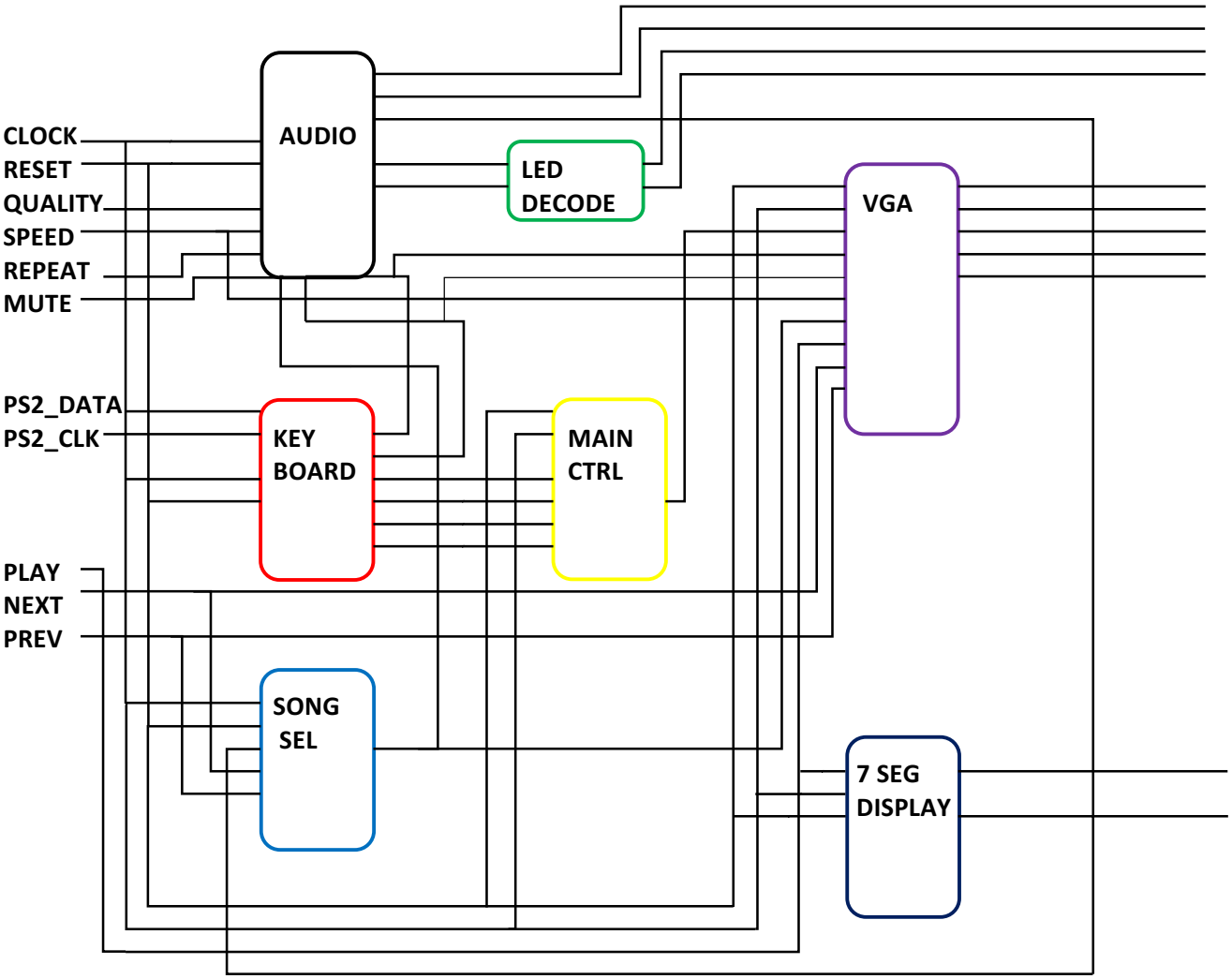
圖二

當進入 CREDIT mode，就會看到照片（圖二）裡面的 CREDIT 在往上捲動，而內容就是圖一的內容。圖一的照片也是使用 Photoshop 製作。在 CREDIT mode 時，如果要按 ESC 就會回到 Main Menu。

設計架構以及電路方塊圖：




PS2_CLK
PS2_DATA
Lled[6:0]_OBUF_inst
Lled[6:0]
OBUF
Led_dcd_inst
Lled[31:0]
Rled[31:0]
o_Lled[6:0]
o_Rled[6:0]
Top_Led_decoder
Rled[6:0]_OBUF_inst
Rled[6:0]
OBUF
Audio_inst
Mute
Play_Pause
Quality[1:0]
Repeat
Speed[1:0]
backward
clk
forward
reset
song_sel[1:0]
freq_JB[31:0]
freq_JC[31:0]
pmod_JB_1
pmod_JB_2
pmod_JB_4
pmod_JC_1
pmod_JC_2
pmod_JC_4
song_finished
Top_Audio
pmod_JB_1_OBUF_inst
pmod_JB_1
pmod_JB_2_OBUF_inst
pmod_JB_2
pmod_JB_4_OBUF_inst
pmod_JB_4
pmod_JC_1_OBUF_inst
pmod_JC_1
pmod_JC_2_OBUF_inst
pmod_JC_2
pmod_JC_4_OBUF_inst
pmod_JC_4
hsync_OBUF_inst
hsync
VGA_inst
backward
clk
forward
mainstate[1:0]
music_sel[1:0]
next
play
rst
speed[2:0]
hsync
vgaBlue[3:0]
vgaGreen[3:0]
vgaRed[3:0]
vsync
Top_Vga_control
vgaBlue[3:0]_OBUF_inst
vgaBlue[3:0]
vgaGreen[3:0]_OBUF_inst
vgaGreen[3:0]
vgaRed[3:0]_OBUF_inst
vgaRed[3:0]
vsync_OBUF_inst
vsync
op_pause_inst
PB_debounced
clock
PB_single_pulse
onepulse
next0_i
I0
I1
O
RTL_OR
Seg7_State_inst
change_state
clock
reset
digit[3:0]
display[6:0]
Top_State_7_seg_display
digit[3:0]_OBUF_inst
digit[3:0]
display[6:0]_OBUF_inst
display[6:0]



實作方法：

在關於實作方面，我們將 project break into part。分別是 Top_Main_Ctrl，Top_Song_Select，Top_VGA_output，Top_AUDIO_output，Top_LED_decoder，Top_7SEG_decoder 和 Top_Keyboard。我們在實作不同的 top module 先，然後在一起組合在一個大 Top final module。這樣的實作很容易 debug，也能很確定每個 module 能用了才組合，而且要升級 Final Project 的一些功能只要針對對應的 module 修改，容易升級。現在我們就一一介紹不同的 module 實作。

Top_Main_Ctrl：

- input（esc，enter，arrow up，arrow down，clock，reset）
- output（state）

這個 module 最主要的功能是判斷要進入 PLAY mode 還是 CREDIT mode。裡面有 4 個 state，分別是 MAIN_PLAY 2'd0 是要讓 VGA 知道現在要顯示的是 Main Menu 的 Play 是黃色，MAIN_CREDIT 2'd1 讓 VGA 顯示 Main Menu 的 Credit 是黃色，PLAYING 2'd2 進入 PLAY mode，其他 module 可以接受訊號，最後是 CREDIT 2'd3 進入 CREDIT mode 讓 VGA 顯示 CREDIT

的照片。MAIN_PLAY 和 MAIN_CREDIT 分別接受 enter 和 arrow up，arrow down。CREDIT 接受 esc。PLAYING 不接受任何訊號。如圖一。

```
24              `MAIN_PLAY: begin
25                  if(enter) next_state = `PLAYING;
26                  else if(arrow_up || arrow_down) next_state = `MAIN_CREDIT;
27                  else next_state = `MAIN_PLAY;
28              end
29              `MAIN_CREDIT: begin
30                  if(enter) next_state = `CREDIT;
31                  else if(arrow_up || arrow_down) next_state = `MAIN_PLAY;
32                  else next_state = `MAIN_CREDIT;
33              end
34              `PLAYING: begin
35                  next_state = `PLAYING;
36              end
37              `CREDIT: begin
38                  if(esc) next_state = `MAIN_PLAY;
39                  else next_state = `CREDIT;
40              end
41              default: begin
42                  next_state = `MAIN_PLAY;
43              end
```

圖一

Top_Song_Select：

- input（clock，reset，next，prev）
- output（song_sel）

這個 module 最重要是要告訴其它 module 現在是在那裡一首音樂，所以會接受 clock，reset，next 和 prev。next 和 prev 能切換音樂。這是一個 finite state。而 output 的 state 分別是 TIGER 2'd0，BIRTHDAY 2'd1，PAYPHONE 2'd2 和 ARBOR 2'd3。其他的 module 只要接受這個 state 並判斷是 0～3 的哪一個就能知道現在是那首音樂。如圖二。

```
23              `TIGER: begin
24                  if(next) next_state = `BIRTHDAY;
25                  else if(prev) next_state = `ARBOR;
26                  else next_state = state;
27              end
```

圖二

Top_LED_decoder：

- input（[31:0] Rled，[31:0] Lled）
- output（[6:0] o_Rled，[6:0] o_Lled）

這個 module 最要是接受從 Top_AUDIO 傳來的音調做比對，並輸出對應的 led 燈數。Decode 的結果分別是，

- C 7’b0000_001
- D 7’b0000_011
- E 7’b0000_111
- F 7’b0001_111
- G 7’b0011_111
- A 7’b0111_111
- B 7’b1111_111
- 沒調就是 7’b0000_000

Top_7SEG_decoder：

- input（clock，reset，play）
- output（digit，display）

這個 module 最最要是要在 7 SEGMENT DISPLAY 顯示現在對應的 state，也就是“HELLO”，“PLAY”和“PAUSE”。他的 input 訊號是 play，然後裡面是一個 finite state machine，output 就是直接顯示現在的 state 是什麼（“HELLO”，“PLAY”，“PAUSE”）。如圖三，圖四和圖五。

```
17    assign {BCD0,BCD1,BCD2,BCD3} = (state == `STOP)? {BCD0_hello, BCD1_hello, BCD2_hello, BCD3_hello} :
18                                   (state == `PLAY)? {BCD0_play, BCD1_play, BCD2_play, BCD3_play} :
19                                    {BCD0_pause, BCD1_pause, BCD2_pause, BCD3_pause};
```

圖三

```
55    assign DISPLAY = (value==`p) ? `P :
56                     (value==`l) ? `L :
57                     (value==`a) ? `A :
58                     (value==`y) ? `Y :
59                     (value==`u) ? `U :
60                     (value==`s) ? `S :
61                     (value==`e) ? `E :
62                     (value==`h) ? `H :
63                     (value==`o) ? `O : 7'b1111_111;
```

圖四

```
1  `define P 7'b0011_000
2  `define p 4'd0
3  `define L 7'b1110_001
4  `define l 4'd1
5  `define A 7'b0001_000
6  `define a 4'd2
7  `define Y 7'b1000_100
8  `define y 4'd3
9  `define U 7'b1000_001
10 `define u 4'd4
11 `define S 7'b0100_100
12 `define s 4'd5
13 `define E 7'b0110_000
14 `define e 4'd6
15 `define H 7'b1001_000
16 `define h 4'd7
17 `define O 7'b0000_001
18 `define o 4'd8
```

圖五

Top_Keyboard：

- input（PS2_DATA，PS2_CLK，reset，clock）
- output（fast_foward，back_forward，esc，enter，up arrow，down arrow）

這個 module 只是要接受 keyboard 傳出的訊號並 output 對應需要的訊號。

Top_VGA_output：

- input（clock，reset，play，next，prev，forward，backward，song select，speed，mainstate）
- output（vgaRED，vgaBLUE，vgaGREEN，hsync，vsync）

這個 module 最重要的是要在螢幕上顯示 Main Menu，Credit 或者是對應的音樂圖畫。從 input，我們能清楚看到 mainstate，這個 input 就是控制 VGA 現在要顯示 Main_Play，Main_Credit，Credit 還是 Playing。

- Main_Play 就是要顯示 MAIN menu，play 是黃色，credit 是灰色
- Main_Credit 就是要顯示 MAIN menu，play 是灰色，credit 是黃色
- Credit 就是要顯示 CREDIT 的畫面
- Playing 就是進入 PLAY mode，並且顯示現在對應的音樂照片

基本上，我把圖畫與背景分開。先看看以下的照片，我再一一解釋。

圖六

圖七

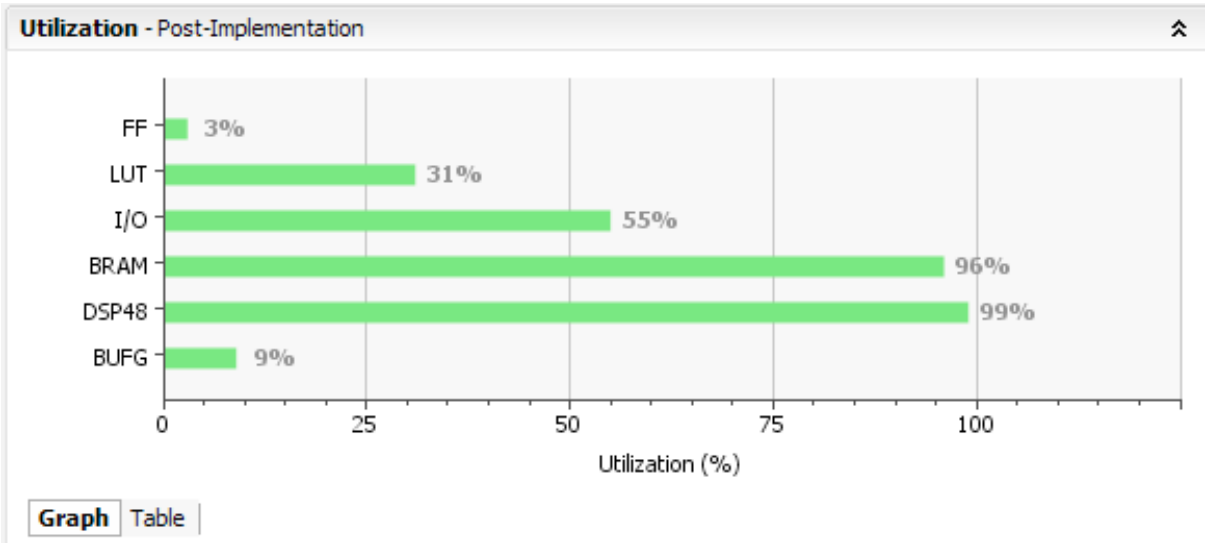
圖八

圖九

圖十

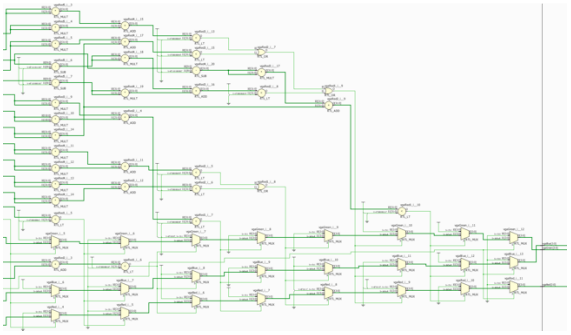
圖十一

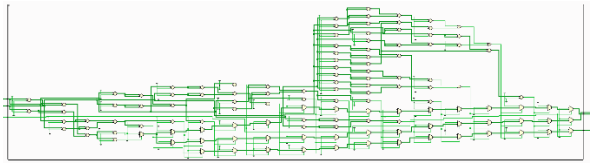
從圖六至圖十一就是將顯示在 MAIN menu 的照片，而至於背景很多的圓圓的球球就是用 x^2 + y^2 < r^2 的方法製作圓圓的圖案。如果要使圖案移動，只要將 x 換成（x－你要的位子），y 換成（y－你要的位子），這樣就能移位了。最後就是判斷（x－你要的位子）確保 x 一定要大於等於你要的位子。例如現在我要顯示圖十在（160，60）的地方，而且需要節省 memory block 的空間，我就會將原本圖十大小為原本 320x120 的圖換成 160x60 的圖，然後確定 160< x <480，x ＝（x－160）>> 1 的方法將圖移到 horizontal 為 160 的地方，並且放大為原圖，同要的方法實作在 vertical，就能節省 memory block 了。這次我使用了很多的 memory block，也完全的用盡了板子最大負荷。



圖十二

另外，從圖十二看，BRAM 是用了 96%，是因為我需要存 10 張圖案。而 DSP48 會是 99%是因為 background generator 佔了 70 到 80%。Background generator 最主要是要將背景顯示出來，裡面用了很多很多的 if else。從電路圖來看，就是像圖十三和圖十四。


圖十三

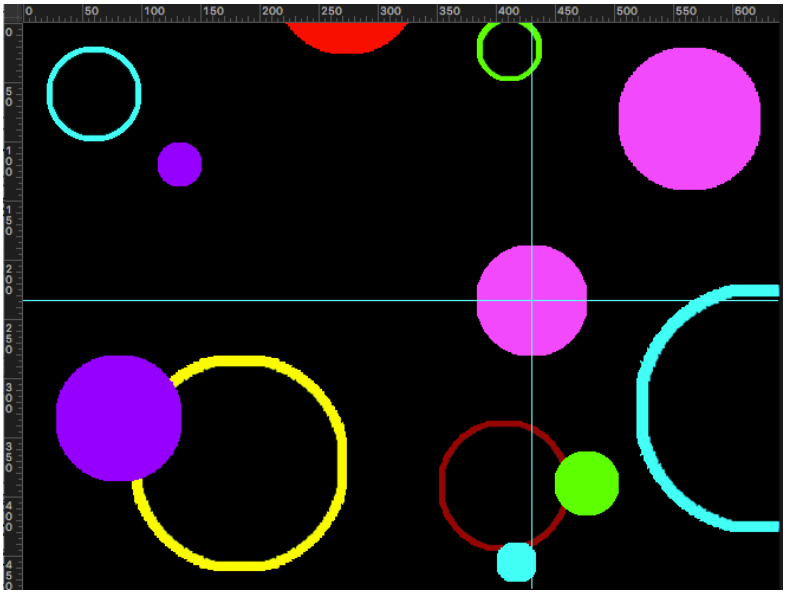

圖十四

而如果從 code 來看，就會像圖十五和圖十六。我是先用 photoshop 畫出我要的圖案，因為有 ruler 的關係，我能確定圓圈的中心的在哪，然後在用 x^2 + y^2 < r^2 的方法移動圓圈。圖十五是最後判斷現在的 h_cnt 和 v_cnt 是要從 memory block 拿 pixel 還是從 background generator 拿。圖十六是 background generator 如何 generate 與圓圈，圖十七是我如何利用 photoshop 找座標。

```
42    assign {vgaRed, vgaGreen, vgaBlue} = (valid == 1'b1 && mainstate == 2'd2)? (
43                                           (music_sel == 2'd0 && (h_cnt > 161) && (h_cnt < 480) && (v_cnt > 60) && (v_cnt < 180))? pixel_tiger:
44                                           (music_sel == 2'd1 && (h_cnt > 161) && (h_cnt < 480) && (v_cnt > 60) && (v_cnt < 180))? pixel_birthday:
45                                           (music_sel == 2'd2 && (h_cnt > 161) && (h_cnt < 480) && (v_cnt > 119) && (v_cnt < 360))? pixel_payphone:
46                                           (music_sel == 2'd3 && (h_cnt > 161) && (h_cnt < 480) && (v_cnt > 119) && (v_cnt < 360))? pixel_arbor : pixel_bg):
47                                         (valid == 1'b1 && mainstate == 2'd0)? (
48                                           ((h_cnt > 160) && (h_cnt < 480) && (v_cnt > 60) && (v_cnt < 180))? pixel_maintitle:
49                                           ((h_cnt > 320) && (h_cnt < 480) && (v_cnt > 188) && (v_cnt < 228))? pixel_mainplay_cover:
50                                           ((h_cnt > 320) && (h_cnt < 480) && (v_cnt > 236) && (v_cnt < 276))? pixel_maincredit_uncover: pixel_bg):
51                                         (valid == 1'b1 && mainstate == 2'd1)? (
52                                           ((h_cnt > 160) && (h_cnt < 480) && (v_cnt > 60) && (v_cnt < 180))? pixel_maintitle:
53                                           ((h_cnt > 320) && (h_cnt < 480) && (v_cnt > 188) && (v_cnt < 228))? pixel_mainplay_uncover:
54                                           ((h_cnt > 320) && (h_cnt < 480) && (v_cnt > 236) && (v_cnt < 276))? pixel_maincredit_cover: pixel_bg):
55                                         (valid == 1'b1 && mainstate == 2'd3)? (
56                                           ((h_cnt > 276) && (h_cnt < 494) && (v_cnt > 60) && (v_cnt < 325))? pixel_credit: pixel_bg): pixel_bg;
```
圖十五

```
10      always@(*) begin
11          if(!valid) begin
12              {vgaRed, vgaGreen, vgaBlue} = 12'h0;
13          end else if(((h_cnt - 417)*(h_cnt - 417)+(v_cnt - 455)*(v_cnt - 455)) < 289) begin
14              {vgaRed, vgaGreen, vgaBlue} = 12'h0ff;
15          end else if((((h_cnt - 609)*(h_cnt - 609)+(v_cnt - 165)*(v_cnt - 165)) < 169) ||
16                      (((h_cnt - 343)*(h_cnt - 343)+(v_cnt - 455)*(v_cnt - 455)) < 64) ||
17                      (((h_cnt - 477)*(h_cnt - 477)+(v_cnt - 389)*(v_cnt - 389)) < 729)) begin
18              {vgaRed, vgaGreen, vgaBlue} = 12'h6f0;
19          end else if((((h_cnt - 81)*(h_cnt - 81)+(v_cnt - 334)*(v_cnt - 334)) < 2809) ||
20                      (((h_cnt - 133)*(h_cnt - 133)+(v_cnt - 120)*(v_cnt - 120)) < 361)) begin
21              {vgaRed, vgaGreen, vgaBlue} = 12'h70f;
22          end else if((((h_cnt - 274)*(h_cnt - 274)+(v_cnt + 34)*(v_cnt + 34)) < 3721)) begin
23              {vgaRed, vgaGreen, vgaBlue} = 12'hf00;
24          end else if((((h_cnt - 564)*(h_cnt - 564)+(v_cnt - 81)*(v_cnt - 81)) < 3600)) begin
25              {vgaRed, vgaGreen, vgaBlue} = 12'hf4f;
26          end else if(((h_cnt - 60)*(h_cnt - 60)+(v_cnt - 60)*(v_cnt - 60)) < 2209) begin
27              {vgaRed, vgaGreen, vgaBlue} = 12'h0;
28          end else if(((h_cnt - 60)*(h_cnt - 60)+(v_cnt - 60)*(v_cnt - 60)) < 2500) begin
29              {vgaRed, vgaGreen, vgaBlue} = 12'h0ff;
30          end else if(((h_cnt - 407)*(h_cnt - 407)+(v_cnt - 391)*(v_cnt - 391)) < 2704) begin
31              {vgaRed, vgaGreen, vgaBlue} = 12'h0;
32          end else if(((h_cnt - 407)*(h_cnt - 407)+(v_cnt - 391)*(v_cnt - 391)) < 3025) begin
33              {vgaRed, vgaGreen, vgaBlue} = 12'hf00;
34          end else if(((h_cnt - 412)*(h_cnt - 412)+(v_cnt - 22)*(v_cnt - 22)) < 625) begin
35              {vgaRed, vgaGreen, vgaBlue} = 12'h0;
36          end else if(((h_cnt - 412)*(h_cnt - 412)+(v_cnt - 22)*(v_cnt - 22)) < 784) begin
37              {vgaRed, vgaGreen, vgaBlue} = 12'h6f0;
38          end else if(((h_cnt - 183)*(h_cnt - 183)+(v_cnt - 372)*(v_cnt - 372)) < 7744) begin
39              {vgaRed, vgaGreen, vgaBlue} = 12'h0;
40          end else if(((h_cnt - 183)*(h_cnt - 183)+(v_cnt - 372)*(v_cnt - 372)) < 8281) begin
41              {vgaRed, vgaGreen, vgaBlue} = 12'hff0;
42          end else if(((h_cnt - 625)*(h_cnt - 625)+(v_cnt - 326)*(v_cnt - 326)) < 10404) begin
43              {vgaRed, vgaGreen, vgaBlue} = 12'h0;
44          end else if(((h_cnt - 625)*(h_cnt - 625)+(v_cnt - 326)*(v_cnt - 326)) < 11025) begin
45              {vgaRed, vgaGreen, vgaBlue} = 12'h0ff;
46          end else begin
47              {vgaRed, vgaGreen, vgaBlue} = 12'h0;
48          end
49      end
```

圖十六



圖十七

Top_AUDIO_output：

- input（clk, reset, Play_Pause, Mute, Repeat, forward, backward, Quality, Speed, song_sel）
- output（pmod_JB_1, pmod_JB_2, pmod_JB_4, pmod_JC_1, pmod_JC_2, pmod_JC_4, song_finished, freq_JB, freq_JC）

Top_Audio 這部分 module 的雛型是參照 Lab8，然而我們的作品多出了許多功能。

首先，基本的音樂播放器一定擁有播放/暫停。這部分是直接從板子的 button 抓訊號，所以需要先經過 debounce 以及 onepulse，也用了 clock_divisor 降成 2^13 的 clk13。

第二，速度、音質對一首歌的構成來說是不可或缺的，這部分是由 PWM_gen 來控制。

```
50    PWM_gen btSpeedGen(.clk(clk), .reset(reset), .freq(speed), .duty(quality), .PWM(beatFreq));
module PWM_gen(
    input wire clk,
    input wire reset,
    input[31:0] freq,
    input[9:0] duty,
    output reg PWM
);

wire[31:0] count_max = 100_000_000 / freq;
wire[31:0] count_duty = count_max * duty / 1024;
reg[31:0] count;

always@(posedge clk, posedge reset) begin
    if(reset) begin
        count <= 0;
        PWM <=0;
    end else if(count < count_max) begin
        count <= count + 1;
        if(count < count_duty)
            PWM <= 1;
        else
            PWM <= 0;
    end else begin
        count <= 0;
        PWM <= 0;
    end
end

endmodule
```

其中 input[31:0] freq 所抓的訊號是 speed、input[9:0] duty 所抓的訊號是 quality，根據傳入的值不同，所產生的 count_max 及 count_duty 也不同，其作法跟 clock_divisor 類似 (上圖紅框部分)。

這邊比較特別的部分是 speed，我們的作品提供 3 種播放速度，0.5x、1x、2x。下圖是計算 speed 的方法，其中 Speed 是從板子抓的訊號有

2’b00 代表 0.5x、2’b01 代表 1x、2’b11 代表 2x。另外，可以看到下圖紅框內，我們所播放的 4 首歌原始速度皆不相同，而 BEAT_FREQ 則是先用 parameter 先設定好的一個值，乘上相對的比例即可。song_sel 是從 Top_Audio 傳入的值，並且會在下面做詳細的介紹。

```
assign speed = (Speed == 2'b00 && song_sel == 2'b00) ? BEAT_FREQ * 36 / 100: //0.5x
               (Speed == 2'b00 && song_sel == 2'b01) ? BEAT_FREQ * 44 / 100: //0.5x
               (Speed == 2'b00 && song_sel == 2'b10) ? BEAT_FREQ * 448 / 1000: //0.5x
               (Speed == 2'b00 && song_sel == 2'b11) ? BEAT_FREQ * 4 / 10: //0.5x
               (Speed == 2'b01 && song_sel == 2'b00) ? BEAT_FREQ * 72 / 100: //1x
               (Speed == 2'b01 && song_sel == 2'b01) ? BEAT_FREQ * 88 / 100: //1x
               (Speed == 2'b01 && song_sel == 2'b10) ? BEAT_FREQ * 896 / 1000: //1x
               (Speed == 2'b01 && song_sel == 2'b11) ? BEAT_FREQ * 8 / 10: //1x
               (Speed == 2'b11 && song_sel == 2'b00) ? BEAT_FREQ * 72 * 2 / 100: //2x
               (Speed == 2'b11 && song_sel == 2'b01) ? BEAT_FREQ * 88 * 2 / 100: //2x
               (Speed == 2'b11 && song_sel == 2'b10) ? BEAT_FREQ * 896 * 2 / 1000: //2x
               (Speed == 2'b11 && song_sel == 2'b11) ? BEAT_FREQ * 8 * 2 / 10: //2x
                                                       BEAT_FREQ * 72 / 100;
```

剛剛提到傳入 duty 的訊號是 quality。我們的作品提供 4 種音質供聆聽。

```
assign quality = (Quality == 2'b00) ? DUTY_BEST / 4 :
                 (Quality == 2'b01) ? DUTY_BEST / 2 :
                 (Quality == 2'b10) ? DUTY_BEST * 3 / 4 :
                 (Quality == 2'b11) ? DUTY_BEST :
                                      DUTY_BEST ;
```

會依照 Quality 的值來判斷，Quality 是從板子的 switch 輸入訊號，分別是 2’b00 時 25%、2’b01 時 50%、2’b10 時 75%、2’b11 時 100%。

一個完整的音樂播放器是絕對不只播放一首歌，因此，我們必須設計如何銜接歌與歌，以及該播放哪一首歌。

```
output song_finished,  input[1:0] song_sel,
```

在 Top_Audio 中 output song_finished 是用來告知其他部分的 module 目前播放的這首歌已經結束，例如，VGA 接收到後，則目前顯示的動畫會跳至下一首歌的動畫。這裡的 input[1:0] song_sel，是用來接收外部 module 選擇歌曲的訊號，因為我們的作品有 4 首歌，所以是 2 個 bit。

song_sel 在 Top_Audio 是非常重要的訊號之一，它與 Top_Audio 所包含的許多 module 都有連接。

```
PWM_gen toneGen_0(.clk(clk), .reset(reset), .freq(freq_JB), .duty(quality), .PWM(pmod_JB_1));
PWM_gen toneGen_1(.clk(clk), .reset(reset), .freq(freq_JC), .duty(quality), .PWM(pmod_JC_1));
```

先來介紹 tone 的輸出，我們的作品最大的特色之一是雙聲道，其所需要用到的兩個 amplifier，分別是 JB and JC。

```
output pmod_JB_1,  output pmod_JC_1,
output pmod_JB_2,  output pmod_JC_2,  output wire[31:0] freq_JB,
output pmod_JB_4,  output pmod_JC_4,  output wire[31:0] freq_JC
```

pomd_JB_1 and pmod_JC_1 由 PWM_gen 產生。

```
assign pmod_JB_2 = 1'd1;
assign pmod_JB_4 = (Mute == 1 || ispause == 1) ? 1'b0 : 1'b1;
assign pmod_JC_2 = 1'd1;
assign pmod_JC_4 = (Mute == 1 || ispause == 1) ? 1'b0 : 1'b1;
```

pmod_JB_2 以及 pmod_JC_2 皆預設為 1。最後，pmod_JB_4 和 pmod_JC_4 則需由板子傳入的 Mute 訊號以及現在是否為停播狀態來判斷其值，如果板子上連接 Mute 的 switch 往上播，Mute 值為 1 則表示要求靜音；另外，ispause 是由 PlayerCtrl 中的 FSM 產生值，其值為 1 的時候代表正處於停播狀態，為 0 則是播放中。

接下來我們來介紹跟 pmod_JB_1、pmod_JC_1 輸出相關的 freq_JB、freq_JC。

```
assign freq_JC = (song_sel == 2'b00) ? freq2 :
                 (song_sel == 2'b01) ? freq1 :
                 (song_sel == 2'b10) ? freq3 :
                 (song_sel == 2'b11) ? freq4_L :
                 (song_sel == 2'b00 && song_finished == 1) ? freq1 :
                 (song_sel == 2'b01 && song_finished == 1) ? freq3 :
                 (song_sel == 2'b10 && song_finished == 1) ? freq4_L :
                 (song_sel == 2'b11 && song_finished == 1) ? freq2 :
                                                             freq2;

assign freq_JB = (song_sel == 2'b00) ? freq2 :
                 (song_sel == 2'b01) ? freq1 :
                 (song_sel == 2'b10) ? freq3 :
                 (song_sel == 2'b11) ? freq4_R :
                 (song_sel == 2'b00 && song_finished == 1) ? freq1 :
                 (song_sel == 2'b01 && song_finished == 1) ? freq3 :
                 (song_sel == 2'b10 && song_finished == 1) ? freq4_R :
                 (song_sel == 2'b11 && song_finished == 1) ? freq2 :
                                                             freq2;
```

由上圖可以看見依據 song_sel 的值不同 assign 給 freq_JB 及 freq_JC 的值也不同，各有 8 種判斷情形，其中加入 song_finished 的就是剛剛提及如何銜接歌與歌。例如，原本正在播 song_sel==2'b00，應該是 assign 成 freq2，如果播放完畢的當下，由 PlayerCtrl 輸出 song_finished = 1 代表播放結束，則需要進入下一首，所以判斷的條件會變成 2 個 song_sel==2'b00 && song_finished == 1，我們可以從上圖看見這時候就不是 assign 成 freq2 而是 freq1 了。接著我們可以看見上圖紅框的部分，與前三首不同的是，對於 freq_JC 是 assign 成 freq4_L，freq_JB 是 assign 成 freq4_R，這就是我們與 proposal 不同，所新增的雙聲道，第四首歌是鋼琴演奏曲，樂譜有分左右手彈奏。

至於如何寫入樂譜的音則共呼叫 5 次 Music 的 module。

```
Music1 music01(.ibeatNum(ibeatNum), .tone(freq1));
Music2 music02(.ibeatNum(ibeatNum), .tone(freq2));
Music3 music03(.ibeatNum(ibeatNum), .tone(freq3));
Music4_R music04_R(.ibeatNum(ibeatNum), .tone(freq4_R));
Music4_L music04_L(.ibeatNum(ibeatNum), .tone(freq4_L));
```

以 music04_R 為例，

```
`define NM1_4 32'd277 //C#_freq
`define NM2_4 32'd294 //D_freq
`define NM3_4 32'd330 //E_freq
`define NM4_4 32'd370 //F#_freq
`define NM5_4 32'd392 //G_freq
`define NM6_4 32'd440 //A_freq
`define NM7_4 32'd494 //B_freq
`define NM0_4 32'd20000 //slience (over freq.)

module Music4_R(
    input [7:0] ibeatNum,
    output reg [31:0] tone
);
```

```
case (ibeatNum)      // 1/4 beat
    8'd0 : tone = `NM0_4;
    8'd1 : tone = `NM2_4 << 1;
    8'd2 : tone = `NM2_4 << 1;
    8'd3 : tone = `NM0_4;
    8'd4 : tone = `NM2_4 << 1;
    8'd5 : tone = `NM2_4 << 1;
    8'd6 : tone = `NM3_4 << 1;
    8'd7 : tone = `NM3_4 << 1;
    8'd8 : tone = `NM3_4 << 1;
    8'd9 : tone = `NM3_4 << 1;
    8'd10 : tone = `NM3_4 << 1;
    8'd11 : tone = `NM3_4 << 1;
    8'd12 : tone = `NM4_4 << 1;
    8'd13 : tone = `NM4_4 << 1;
    8'd14 : tone = `NM4_4 << 1;
    8'd15 : tone = `NM4_4 << 1;
```

ibeatNum 是 input，由於板子 memory 大小的問題牽扯到可以放多大的圖片，所以我們決定只寫 8 個 bit，共可以播放 256 個音。tone 則是 output，。可以看見上方右圖是寫入播放的頻率每一個 ibeatNum 對於音樂來說是 1/4beat，也就是一個四分音符相當於四個 ibeatNum，後面加給 tone 的值我們在一開始的`define 寫了預設值，這邊我們對於每個 Music 的 module 都擁有不同名稱，例如在 Music1 中是 NM1，在 Music2 中是 NM1_2，在 Music3 中是 NM1_3，在 Music4_R 中是 NM1_4，在 Music4_L 中是 NM1_5，為什麼要這麼做呢？因為最初我們每個 module 全部 define 相同名稱會產生 error。再來談談 tone 的寫法，對音樂來說同一個音域有 7 度音，如果需要在更高則稱為高 8 度，更低則稱為低 8 度，這邊我們的寫法是可以看見上方右圖 8'd1 : tone = `NM2_4 << 1，這邊的'<<'符號就用來升高 8 度音，反之，如需降低 8 度音則需寫成'>>'，例如

```
8'd48 : tone = `NM5_5 >> 1;
```

最後，我們來介紹 Top_Audio 中最關鍵的 Player_Ctrl。在這個 module 中 Repeat、forward、backward 這 3 個 input 是用來判斷的，Repeat 直接從板子的 switch 抓訊號，等於 1 時表示是重複播放，等於 0 則否。forward 及 backward 則是由 keyboard 抓訊號，一樣等於 1 時是往前打快/往回打快。

```
if(forward == 1 && backward == 0)begin
    if(ibeat > BEATLENGTH - 8)
        next_ibeat = BEATLENGTH;
    else begin
        next_ibeat = ibeat + 8;
        song_finished = 0;
    end
end
else if(forward == 0 && backward == 1)begin
    if(ibeat < 8)
        next_ibeat = 0;
    else begin
        next_ibeat = ibeat - 8;
        song_finished = 0;
    end
end
```

每首歌的長度不同，所以這個 module 也需傳入 song_sel 來判斷歌曲長度，例如，第一首歌長度為 128 個 ibeatNum，第二首長度為 216，第三首長度為 232，第四首長度為 256，與下圖中皆差 1 是因為 ibeatNum 從 0 開始算。

```
assign BEATLENGTH = (song_sel == 2'b00) ? 127 :
                    (song_sel == 2'b01) ? 215 :
                    (song_sel == 2'b10) ? 231 :
                    (song_sel == 2'b11) ? 255 :
                                          127;
```

```
parameter st  = 2'b00;
parameter play = 2'b01;
parameter pause = 2'b10;
```

Player_Ctrl 中有一個 FSM，包含 3 種情況。stop 為初始狀態，play 為正在播放，pause 為停播狀態。

當我們在 play 的 state 時，當 ibeat == BEATLENGTH 時來判斷 song_finished 的值。

```
if(ibeat == BEATLENGTH) begin
    next_ibeat = 0;
    if(Repeat == 1) begin
        next_state = play;
        ispause = 0;
        song_finished = 0;
    end
    else begin
        ispause = 0;
        next_state = play;
        song_finished = 1;
    end
end
```

心得：

張緣彩：對於這次的 final，我從中學會了很多，從不會到會，從完全不知道 VGA 是怎麼顯示畫面到完全掌握。現在我能隨心所欲，想再螢幕上的哪裡一個點顯示哪一張照片或者是印出圖案，我都能馬上實作處理。剛剛開始我們原本打算實作遊戲性質很高的跳舞機，但是因為怕時間不夠而最後決定做音樂播放器，直到實作完這作品後，我很確定能實作跳舞機，只

要 h_cnt 和 v_cnt 判斷對與 keyboard 的訊號輸出對應到就能算分。這樣就能完全實作跳舞機出來。如果還有下一次的實作機會，我想挑戰更難的。總而言之，經過這次的實作，我對 finite state machine 的使用完全不陌生，反而很親切，在這次的實作，我使用了很多的 finite state machine，包括 7 segment display，main control，song selection 和 vga。雖然這次我對 audio 方面沒有比 vga 來得熟悉，但是跟許博皓討論的時候也知道是怎麼回事。我們的作品從看起來很爛，慢慢的改，慢慢的包裝到最終看起來完成度 100%以及很滿意了。在這我也想感謝助教你們一直給我們問，在問的過程中，我們越來越清楚硬體是怎麼操作。另外，我真的是音樂白痴，我到現在為止還看不懂五線譜，所以 AUDIO 方面我真的要感謝我的好搭檔許博皓的 carry。最後，我是第一次使用中文打報告，如果有任何地方解釋的不清楚還是用詞不當，請手下留情 QQ。

許博皓：這次的 final 讓我對於整個學期所教的全都再複習過了一遍，也學到了很多新東西。一開始我跟我的 partner 討論 proposal 時想了很久，也想了很多方案，因為我們有共同的想法，就是用上全部教過的功能，只不過後來我們選擇了遊戲性較低，但實用性較高的音樂播放器。依照 proposal 上的設計圖，分配了各自所負責 module，然後再約時間組合。我跟我的 partner 合作的非常愉快，我們算是很早就開始動工的組，所以並沒有像許多同學在 demo 的前幾天待在電腦教室打通宵，彼此也做了一定程度的溝通，像是自己負責 module 需要怎麼樣子的 input，會 output 怎麼樣子的訊號。雖然我沒有負責 VGA 的部分，但有私下稍微玩了一下，以至於在討論的時候不會到聽不懂對方在做什麼。我覺得我們的 project 並沒有很多困難到無法執行的地方，而是有很多很麻煩的，例如 VGA 的 if、else if 判斷以及 Audio 把樂譜中的音符轉變成相對應的頻率。完成這次的 final 後，我認為我們的作品可以做很廣的延伸，如果硬體允許，可以貞的呈現一個類似影片的動畫；如果 amplifier 足夠，可以呈現一整個交響樂團。這次唯一沒做到 proposal 的功能就只有歌詞的部分，我們的構想是類似 KTV 的伴唱帶，只是最後因為時間跟執行困難度關係而取消，不過我想，如果給我們足夠的時間，我跟我 partner 一定可以做出來，畢竟這次的 final 讓我們對於 FPGA 及 verilog 更加的了解認識，有了理論基礎，實作成功只是遲早的事。

分工項目以及對 project 的貢獻百分比：

張緣彩：VGA，KEYBOARD，7-SEGMENT DISPLAY，BUTTON，MAIN CONTROL，SONG SELECT，LED DECODER，PROPOSAL，POWERPOINT，REPORT。對 Final 的貢獻度 50%。

許博皓：AUDIO，BUTTON，LED DECODER，PROPOSAL，POWERPOINT，REPORT。對 Final 的貢獻度 50%