

# Lab 7

學號: 109062173

姓名: 葉昱揚

## A. Lab Implementation

### 1. Block Diagram of the design with explanation (10%)

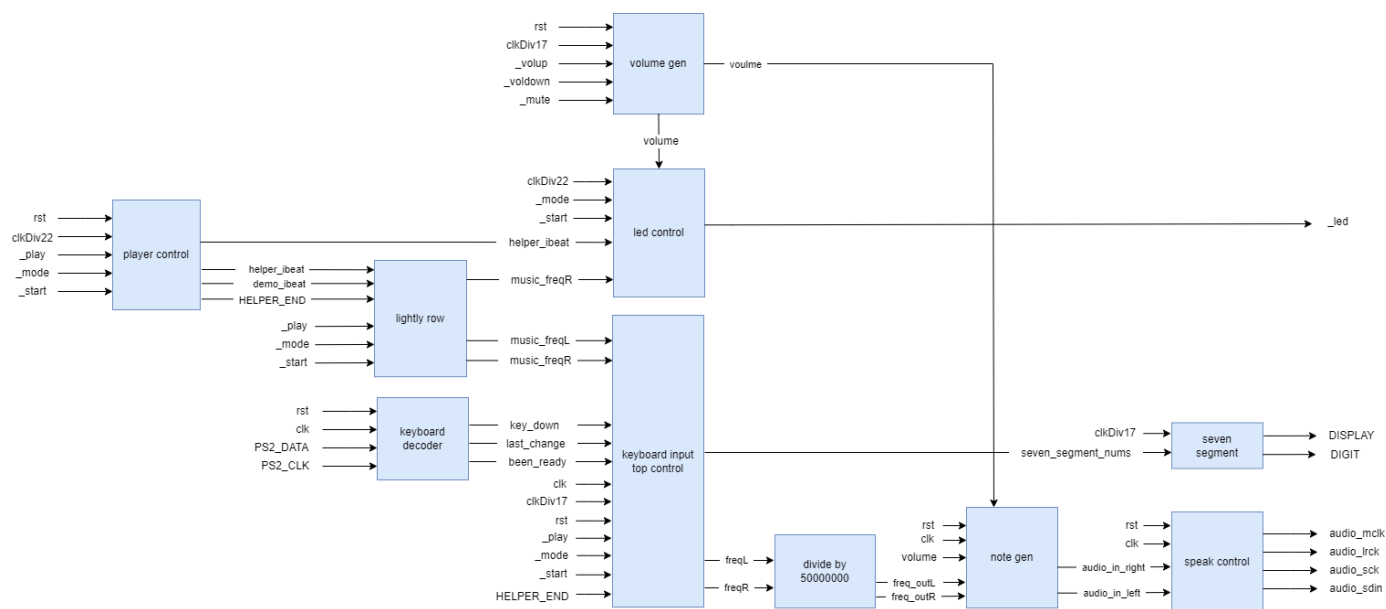


圖 1、Lab7 block diagram

上圖是 Lab7 的 block diagram。由左到右看也是產出 output 的流程圖

player control 是控制音符的 beat 的產生器:

demo_ibeat	控制 demonstrate mode 音符的 beat
helper_ibeat	控制 Play mode helper 音符的 beat
HELPER_END	判斷當前是否為 Play mode helper 完全結束

lightly row 是預先寫好的音譜，lightly row 即是譜名，會接收 player control 產生的 beat，產出對應的音頻:

music_freqL	音譜的左音頻
music_freqR	音譜的右音頻

volume gen 是控制音量大的 module，它會接收 voldown button、volup button 和 mute 的資訊，產出對應的音量層級:

volume	音量層級
--------	------

keyboard input control 是監控鍵盤輸入，並以 music\_freqL、music\_freqR、當前模式等等...為 input，產出 freqL、freqR、seven\_segment\_nums:

freqL	聲音的左音頻
freqR	聲音的右音頻
seven_segment_nums	七段顯示器的控制碼

note gen 吃進經過處理的 freqL 和 freqR，並配合 volume gen 的 output volume，產生 audio\_in\_left、audio\_left\_right。最後經過 speaker control 生產出我們耳朵聽到的聲音。

## 2. Partial code screenshot with the explanation (35%)

### Volume control



```
1  module volume_gen(  
2      input clk,  
3      input rst,  
4      input volup,  
5      input voldown,  
6      input isMute,  
7      output reg [2:0]volume  
8  );  
9  
10     reg [2:0] v;  
11     reg [2:0] next_v;  
12  
13     /* ----- */  
14     /*                               volume                               */  
15     /* ----- */  
16     always@(*) begin  
17         if(isMute) volume = 3'd0;  
18         else volume = v;  
19     end  
20  
21     /* ----- */  
22     /*                               v                               */  
23     /* ----- */  
24     always@(posedge clk,posedge rst) begin  
25         if(rst) v <= 3'd3;  
26         else v <= next_v;  
27     end  
28  
29     always@(*) begin  
30         if(volup && v < 3'd5) next_v = v + 1;  
31         else if(voldown && v > 3'd1) next_v = v - 1;  
32         else next_v = v;  
33     end  
34 endmodule
```

圖、設定音量大小

參考上圖，module volume\_gen 有 3 個重要的 input signal 和一個 output signal:

- (input) volup 音量放大一個 level
- (input) voldown 音量變小一個 level
- (input) isMute 靜音
- (output) volume 音量 level

控制音量大小時，先用其他線路(v、next\_v)處理：

如果按下 **volup button** 且 v 小於最大音量(level 5)，則音量放大一個 level， $v + 1$ ；

如果按下 **voldown button** 且 v 大於最低音量(level 1)，則音量變小一個 level， $v - 1$ ；

反之 v 不變。

最後把處理好的 v 連接給 output volume:

如果需要靜音(isMute)，則  $\text{volume} = 0$ ；

反之， $\text{volume} = v$ 。

這邊的小細節是 isMute 需要和 volup、voldown 分開處理。

原因是當 isMute 訊號拉起又放下後，volume 需要回歸到 isMute 拉起前的狀態。

用其他線路(v、next\_v)紀錄原本的音量大小，再分開處理 isMute 訊號，就可以避免當 isMute 拉起時洗刷掉原本音量的情況。

```

1
2 // Assign the amplitude of the note
3 // Volume is controlled here
4 /* ----- */
5 /*                          audio_left                          */
6 /* ----- */
7 always@(*) begin
8     if(note_div_left == 22'd1) audio_left = 16'h0000;
9     else begin
10         case(volume)
11             0: audio_left = 16'h0000;
12             1: audio_left = (b_clk == 1'b0) ? 16'h00C8 : 16'hFF38; // -200 < volume < 200
13             2: audio_left = (b_clk == 1'b0) ? 16'h03E8 : 16'hFC18; // -1000 < volume < 1000
14             3: audio_left = (b_clk == 1'b0) ? 16'h0BB8 : 16'hF448; // -3000 < volume < 3000
15             4: audio_left = (b_clk == 1'b0) ? 16'h1388 : 16'hEC78; // -5000 < volume < 5000
16             5: audio_left = (b_clk == 1'b0) ? 16'h2710 : 16'hD8F0; // -10000 < volume < 10000
17         endcase
18     end
19 end
20
21 /* ----- */
22 /*                          audio_right                         */
23 /* ----- */
24 always@(*) begin
25     if(note_div_right == 22'd1) audio_right = 16'h0000;
26     else begin
27         case(volume)
28             0: audio_right = 16'h0000;
29             1: audio_right = (c_clk == 1'b0) ? 16'h00C8 : 16'hFF38; // -200 < volume < 200
30             2: audio_right = (c_clk == 1'b0) ? 16'h03E8 : 16'hFC18; // -1000 < volume < 1000
31             3: audio_right = (c_clk == 1'b0) ? 16'h0BB8 : 16'hF448; // -3000 < volume < 3000
32             4: audio_right = (c_clk == 1'b0) ? 16'h1388 : 16'hEC78; // -5000 < volume < 5000
33             5: audio_right = (c_clk == 1'b0) ? 16'h2710 : 16'hD8F0; // -10000 < volume < 10000
34         endcase
35     end
36 end

```

圖、設定振幅(amplitude)大小

參考上圖，經過 volume\_gen 產出的 volume 會在兩個 always block 發揮作用( Line10~Line17、Line27~Line34 )。這邊依據 volume 的 level，分別把音量大小的數據賦給左右聲道。

其中 volume 的等級有 mute、level 1 ~ level 5 總共 6 個等級。  
它們實際代表的音量數據分別為：

- 第 0 級 (mute) ，音量 = 0
- 第 1 級，音量 = 200
- 第 2 級，音量 = 1000
- 第 3 級，音量 = 3000
- 第 4 級，音量 = 5000
- 第 5 級，音量 = 10000

## DEMONSTATE mode

```

1  module demo_beat_control (
2      input clkDiv22,
3      input rst,
4      input play,      // SW0: Play/Pause
5      input mode,      // SW15: Mode
6      output reg [11:0] demo_ibeat
7  );
8
9      parameter DEMONSTRATE_MODE = 1'b1;
10     parameter PLAY_MODE       = 1'b0;
11     parameter LEN = 4095;
12
13     /* ----- */
14     /*             demo_ibeat             */
15     /* ----- */
16     reg [8:0] next_demo_ibeat;
17
18     always @(posedge clkDiv22, posedge rst) begin
19         if (rst) begin
20             demo_ibeat <= 0;
21         end else begin
22             demo_ibeat <= next_demo_ibeat;
23         end
24     end
25
26     // next_beat
27     always @* begin
28         if(mode==DEMONSTRATE_MODE && play) next_demo_ibeat = (demo_ibeat + 1 < LEN) ? (demo_ibeat + 1) : 0;
29         else next_demo_ibeat = demo_ibeat;
30     end
31
32 endmodule

```

圖、DEMONSTRATE mode 的 beat control

上圖是控制 DEMONSTRATE mode 的 beat 的 module。

module demo\_beat\_control 有 2 個重要的 input signal 和一個 output signal:

(input)	play	播放 / 暫停
(input)	mode	DEMONSTRATE mode / Play mode
(output)	demo ibeat	控制 DEMONSTRATE mode 時播放的音符

在 DEMONSTRATE mode 有兩項目標需要達成。

實作在上圖 Line18~30，主要控制在 Line28、Line29:

1. 循環播放預先寫好的旋律

Line28: 只要 demo\_ibeat 沒超過歌曲長度就會不斷增加。  
若超過長度，則歸零，回到歌曲原點。

2. 若中途暫停/切到別的 mode，下次進入 DEMONSTRATE mode 需要從上次的離開點繼續播放。

Line28 判斷“如果現在是 DEMONSTRATE mode 且 play”才會更動 demo\_ibeat。  
因此中途暫停/切到別的 mode 時，Line29 會讓 demo\_ibeat 保持原樣不變。

```
1  /* ----- */
2  /*                               lightly_row                               */
3  /* ----- */
4  wire [31:0] music_freqL;
5  wire [31:0] music_freqR;
6  lightly_row music(
7      .demo_ibeat(demo_ibeat),
8      .helper_ibeat(helper_ibeat),
9      .MODE(_mode),
10     .PLAY(_play),
11     .START(_start),
12     .HELPER_END(HELPER_END),
13     .toneL(music_freqL),
14     .toneR(music_freqR)
15 );
```

圖、預先寫好的音樂 module

module lightly\_row 有 2 個重要的 input signal 和 2 個 output signal:

(input)	demo_ibeat	DEMOE mode 的 beat 位置
(input)	helper_ibeat	PLAYmode 的 helper 的 beat 位置
(output)	music_freqR	右聲道音符
(output)	music_freqL	左聲道音符

這個 module 吃進兩個 mode\_ibeat，並判斷現在是 demo mode OR play mode，把對應的音符輸出給 music\_freqR 和 music\_freqL

```

1
2  /* ----- */
3  /*           DEMONSTRATE_MODE           */
4  /* ----- */
5  always@(*) begin
6      case(MODE)
7          DEMONSTRATE_MODE:begin
8              next_freqL = music_freqL;
9              next_freqR = music_freqR;
10             next_seven_segment_nums[19:0] = {`seven_segment_DASH, `seven_segment_DASH, `seven_segment_DASH, `seven_segment_DASH};
11             if(PLAY) begin
12                 case(music_freqR)
13                     `lc: next_seven_segment_nums[9:0] = {`seven_segment_C, `seven_segment_3 };
14                     `ld: next_seven_segment_nums[9:0] = {`seven_segment_D, `seven_segment_3 };
15                     `le: next_seven_segment_nums[9:0] = {`seven_segment_E, `seven_segment_3 };
16                     `lf: next_seven_segment_nums[9:0] = {`seven_segment_F, `seven_segment_3 };
17                     `lg: next_seven_segment_nums[9:0] = {`seven_segment_G, `seven_segment_3 };
18                     `la: next_seven_segment_nums[9:0] = {`seven_segment_A, `seven_segment_3 };
19                     `lb: next_seven_segment_nums[9:0] = {`seven_segment_B, `seven_segment_3 };
20
21                     `c: next_seven_segment_nums[9:0] = {`seven_segment_C, `seven_segment_4 };
22                     `d: next_seven_segment_nums[9:0] = {`seven_segment_D, `seven_segment_4 };
23                     `e: next_seven_segment_nums[9:0] = {`seven_segment_E, `seven_segment_4 };
24                     `f: next_seven_segment_nums[9:0] = {`seven_segment_F, `seven_segment_4 };
25                     `g: next_seven_segment_nums[9:0] = {`seven_segment_G, `seven_segment_4 };
26                     `a: next_seven_segment_nums[9:0] = {`seven_segment_A, `seven_segment_4 };
27                     `b: next_seven_segment_nums[9:0] = {`seven_segment_B, `seven_segment_4 };
28
29                     `hc: next_seven_segment_nums[9:0] = {`seven_segment_C, `seven_segment_5 };
30                     `hd: next_seven_segment_nums[9:0] = {`seven_segment_D, `seven_segment_5 };
31                     `he: next_seven_segment_nums[9:0] = {`seven_segment_E, `seven_segment_5 };
32                     `hf: next_seven_segment_nums[9:0] = {`seven_segment_F, `seven_segment_5 };
33                     `hg: next_seven_segment_nums[9:0] = {`seven_segment_G, `seven_segment_5 };
34                     `ha: next_seven_segment_nums[9:0] = {`seven_segment_A, `seven_segment_5 };
35                     `hb: next_seven_segment_nums[9:0] = {`seven_segment_B, `seven_segment_5 };
36                     default: next_seven_segment_nums[9:0] = seven_segment_nums[9:0];
37                 endcase
38             end
39             else begin
40                 next_seven_segment_nums[9:0] = {`seven_segment_DASH, `seven_segment_DASH };
41             end
42         end
43     endcase
44 end

```

圖、DEMONSTRATE mode 的控制

參考上圖的 Line8 、 Line9 。

music\_freqL、music\_freqR 是預先寫好的音譜的音符。這個地方會把 music\_freqL、music\_freqR 賦給真正能發出聲音的左聲道 freqL、右聲道 freqR

參考上圖的 Line12 ~ Line 37 。

這個 case 判斷目前音樂的右聲道 music\_freqR 是甚麼音符並將結果顯示到七段顯示器上。

music\_freqR 有 21 種可能，分別為：

降 C (lc) 顯示 C3;	C(c) 顯示 C4;	升 C(hc) 顯示 C5;
降 D (ld) 顯示 D3;	D(d) 顯示 D4;	升 D(hd) 顯示 D5;
降 E (le) 顯示 E3;	E(e) 顯示 E4;	升 E(he) 顯示 E5;
降 F (lf) 顯示 F3;	F(f) 顯示 F4;	升 F(hf) 顯示 F5;
降 G (lg) 顯示 G3;	G(g) 顯示 G4;	升 G(hg) 顯示 G5;
降 A (la) 顯示 A3;	A(a) 顯示 A4;	升 A(ha) 顯示 A5;



降 B (lb) 顯示 B3;      B(b) 顯示 B4;      升 B(hb) 顯示 B5;

## PLAY mode – Piano

```

1  case(MODE)
2      PLAY_MODE:begin
3          next_freqL = `silence;
4          next_freqR = `silence;
5          next_seven_segment_nums[19:0] = {`seven_segment_DASH, `seven_segment_DASH, `seven_segment_DASH, `seven_segment_DASH};
6          if(!HELPER_END) begin
7              case(key_num)
8                  KEY_CODES_Q: begin
9                      next_freqL = `hc;
10                     next_freqR = `hc;
11                     next_seven_segment_nums[9:0] = { `seven_segment_C, `seven_segment_5 };
12                 end
13
14                 KEY_CODES_W: begin
15                     next_freqL = `hd;
16                     next_freqR = `hd;
17                     next_seven_segment_nums[9:0] = { `seven_segment_D, `seven_segment_5 };
18                 end
19
20                 KEY_CODES_E: begin
21                     next_freqL = `he;
22                     next_freqR = `he;
23                     next_seven_segment_nums[9:0] = { `seven_segment_E, `seven_segment_5 };
24                 end
25
26                 KEY_CODES_R: begin
27                     next_freqL = `hf;
28                     next_freqR = `hf;
29                     next_seven_segment_nums[9:0] = { `seven_segment_F, `seven_segment_5 };
30                 end
31
32                 KEY_CODES_T: begin
33                     next_freqL = `hg;
34                     next_freqR = `hg;
35                     next_seven_segment_nums[9:0] = { `seven_segment_G, `seven_segment_5 };
36                 end
37
38                 KEY_CODES_Y: begin
39                     next_freqL = `ha;
40                     next_freqR = `ha;
41                     next_seven_segment_nums[9:0] = { `seven_segment_A, `seven_segment_5 };
42                 end
43
44                 KEY_CODES_U: begin
45                     next_freqL = `hb;
46                     next_freqR = `hb;
47                     next_seven_segment_nums[9:0] = { `seven_segment_B, `seven_segment_5 };
48                 end
49
50                 /* ----- 剩下音符的寫法都差不多 ----- */
51             endcase
52         end
53     end
54 end
55 endcase

```

圖、Play mode Piano

在這個 mode 需要做的判斷是：

判斷目前按下哪一個案件，發出對應的音頻，並將音頻名顯示於七段顯示器上。

參考上圖的 case，key\_num 是按壓的鍵盤。

若沒有按壓任何按鍵，則左聲道 freqL、右聲道 freqR 都是靜音 silence，七段顯示器顯示 DASH;

如果按壓鍵盤 Q，Q 代表升 C(hc)，左聲道 freqL、右聲道 freqR 都是 hc，七段顯示器顯示 C5;

如果按壓鍵盤 W，W 代表升 D(hd)，左聲道 freqL、右聲道 freqR 都是 hd，七段顯示器顯示 D5;

剩下的按鍵操作同理。

只需要對照 SPEC 不同按鍵對應到不同的音頻，七段顯示器顯示不同的音頻。

因此上圖只列出 7 個鍵盤判斷，當作簡易的範例解釋，但實際上有 21 個判斷。

## PLAY mode – Helper

//Explain how to use the song as a demonstration to design the helper game.

```

1  module helper_beat_control (
2      input clkDiv22,
3      input rst,
4      input mode,          // SW15: Mode
5      input start,
6      output reg [11:0] helper_ibeat,
7      output reg HELPER_END
8  );
9
10     parameter DEMONSTRATE_MODE = 1'b1;
11     parameter PLAY_MODE        = 1'b0;
12     parameter LEN = 4095;
13
14     /* ----- */
15     /*             helper_ibeat             */
16     /* ----- */
17     reg initialized;
18     reg [8:0] next_helper_ibeat;
19
20     always @(posedge clkDiv22, posedge rst) begin
21         if (rst) begin
22             helper_ibeat <= 0;
23             initialized <= 0;
24         end
25         else begin
26             if (!initialized) begin
27                 helper_ibeat <= 0;
28                 initialized <= 1; //
29             end
30             else begin
31                 helper_ibeat <= next_helper_ibeat;
32                 initialized <= 1;
33             end
34
35             if (! (mode==PLAY_MODE && start)) initialized <= 0;
36         end
37     end
38
39     // next_helper_beat
40     always @* begin
41         if (mode == PLAY_MODE && start == 1 && !HELPER_END) begin
42             if (helper_ibeat < LEN) next_helper_ibeat = helper_ibeat + 1;
43             else next_helper_ibeat = LEN;
44         end
45         else next_helper_ibeat = helper_ibeat;
46     end
47
48     /* ----- */
49     /*             HELPER_END             */
50     /* ----- */
51     always @(*) begin
52         if (rst) HELPER_END = 0;
53         else if (helper_ibeat==LEN) HELPER_END = 1;
54         else HELPER_END = 0;
55     end
56
57 endmodule

```

圖、PLAY mode helper 的 beat control

上圖是控制 PLAY mode helper 的 beat 的 module。

module healper\_beat\_conotrol 有 2 個重要的 input signal 和 2 個 output signal:

(input)	star	開啟 helper / 關閉 helper
(input)	mode	DEMONSTRATE mode / Play mode
(output)	helper ibeat	控制 PLAY mode helper 播放的音符



(output) HELPER\_END 是否在 helper 模式下跑完整首旋律

在 Line20~Line37, helper ibeat 會不斷遞增, 當整首旋律結束時 helper ibeat 會停留在最後一個音符。值得注意的是, 每次開啟 helper 都要從頭開始跑旋律。因此設計一個 initialized:

如果 initialized 為 0 -> helper ibeat 初始化

如果 initialized 為 1 -> helper ibeat 不變

當 helper ibeat 停留在最後一個音符時, 代表旋律全部結束, 此時 HELPER\_END 會被設為 active。HELPER\_END 使用於七段顯示器後控制鍵盤輸入。

接下來分為兩個部分解釋 PLAY mode Helper 的聲音輸出、七段顯示器輸出、LED 燈輸出。

### helper 聲音輸出

helper 的聲音輸出方式與上方的 PLAY mode piano 一模一樣, 兩者的模式都是判斷按壓哪個鍵盤, 把對應個音頻輸出出去。PLAY mode piano 和 PLAY mode helper 聲音輸出的 code 是共用的。

### helper 七段顯示器輸出

helper 的七段顯示器需要顯示當前旋律的音頻, 而不是鍵盤對應的音頻。

從上圖的 module lightly\_row 得到當前旋律右聲道 music\_freqR、左聲道 music\_freqL, 這邊選擇顯示右聲道的音頻。

```

1  /* ----- helper 七段顯示器 ----- */
2  /* ----- helper 七段顯示器 ----- */
3  /* ----- helper 七段顯示器 ----- */
4  if(START) begin
5      next_seven_segment_nums[19:10] = seven_segment_nums[19:10];
6      case(music_freqR)
7          `lc: next_seven_segment_nums[9:0] = { `seven_segment_C, `seven_segment_3 };
8          `ld: next_seven_segment_nums[9:0] = { `seven_segment_D, `seven_segment_3 };
9          `le: next_seven_segment_nums[9:0] = { `seven_segment_E, `seven_segment_3 };
10         `lf: next_seven_segment_nums[9:0] = { `seven_segment_F, `seven_segment_3 };
11         `lg: next_seven_segment_nums[9:0] = { `seven_segment_G, `seven_segment_3 };
12         `la: next_seven_segment_nums[9:0] = { `seven_segment_A, `seven_segment_3 };
13         `lb: next_seven_segment_nums[9:0] = { `seven_segment_B, `seven_segment_3 };
14
15         `c: next_seven_segment_nums[9:0] = { `seven_segment_C, `seven_segment_4 };
16         `d: next_seven_segment_nums[9:0] = { `seven_segment_D, `seven_segment_4 };
17         `e: next_seven_segment_nums[9:0] = { `seven_segment_E, `seven_segment_4 };
18         `f: next_seven_segment_nums[9:0] = { `seven_segment_F, `seven_segment_4 };
19         `g: next_seven_segment_nums[9:0] = { `seven_segment_G, `seven_segment_4 };
20         `a: next_seven_segment_nums[9:0] = { `seven_segment_A, `seven_segment_4 };
21         `b: next_seven_segment_nums[9:0] = { `seven_segment_B, `seven_segment_4 };
22
23         `hc: next_seven_segment_nums[9:0] = { `seven_segment_C, `seven_segment_5 };
24         `hd: next_seven_segment_nums[9:0] = { `seven_segment_D, `seven_segment_5 };
25         `he: next_seven_segment_nums[9:0] = { `seven_segment_E, `seven_segment_5 };
26         `hf: next_seven_segment_nums[9:0] = { `seven_segment_F, `seven_segment_5 };
27         `hg: next_seven_segment_nums[9:0] = { `seven_segment_G, `seven_segment_5 };
28         `ha: next_seven_segment_nums[9:0] = { `seven_segment_A, `seven_segment_5 };
29         `hb: next_seven_segment_nums[9:0] = { `seven_segment_B, `seven_segment_5 };
30         default: next_seven_segment_nums[9:0] = seven_segment_nums[9:0];
31     endcase
32 end

```

圖、PLAY mode helper 的七段顯示器控制

參考上圖, 21 種音頻對應 21 條判斷式, 並將該 music\_freqR 音頻顯示於七段顯示器的右邊兩格。

Ex: 當前音符是降 C (lc) 在七段顯示器最右邊兩格顯示 C3。其他鍵盤概念雷同。

## helper LED 燈輸出

```

1  module Led_control(
2      input clk,
3      input MODE,
4      input START,
5      input [31:0] music_freqR,
6      input [11:0] helper_ibeat,
7      input [2:0] volume,
8      output reg [15:0] led
9  );
10
11  /* ----- */
12  /*                      led control                      */
13  /* ----- */
14  always@(*) begin
15      led = 16'b0;
16
17      /* ----- Play mode helper ----- */
18      if(MODE == 1'b0 && START) begin
19          case(music_freqR)
20              `lc,`c,`hc: led[15]=1;
21              `ld,`d,`hd: led[14]=1;
22              `le,`e,`he: led[13]=1;
23              `lf,`f,`hf: led[12]=1;
24              `lg,`g,`hg: led[11]=1;
25              `la,`a,`ha: led[10]=1;
26              `lb,`b,`hb: led[9] =1;
27          endcase
28          if(helper_ibeat==11'd511) led[15:9] = 7'b111_1111;
29      end
30      /* ----- */
31
32      /* ----- volume led ----- */
33      case(volume)
34          0: led[4:0] = 5'b0;
35          1: led[4:0] = {4'b0000,1'b1};
36          2: led[4:0] = {3'b000,2'b11};
37          3: led[4:0] = {2'b00,3'b111};
38          4: led[4:0] = {1'b0,4'b1111};
39          5: led[4:0] = {5'b1_1111};
40      endcase
41  end
42  /* ----- */
43  end
44
45  endmodule

```

圖、led control

上圖是控制 led 的 module led\_control。

PLAY mode helper	控制 led[15:9]
volume	控制 led[4: 0]

PLAY mode helper 的 led 燈需要顯示預先寫好的旋律的各個音頻代表的 led。

music\_freqR 即是預先寫好的旋律的各個音頻，因此在上圖 Line19 ~ Line28 可以看到：

如果音頻是 Do，則 led[15] 亮起，其他熄滅；  
 如果音頻是 RE，則 led[14] 亮起，其他熄滅；  
 如果音頻是 Mi，則 led[13] 亮起，其他熄滅；  
 如果音頻是 Fa，則 led[12] 亮起，其他熄滅；  
 如果音頻是 Sol，則 led[11] 亮起，其他熄滅

如果音頻是 La，則 led[10] 亮起，其他熄滅；  
如果音頻是 Ti，則 led[09] 亮起，其他熄滅；  
當整首旋律結束時，亮起 led[15:9]

## Score mechanism of the Helper

```
1  if(!HELPER_END && !inc) begin
2      case(music_freqR)
3          `lc,`c,`hc: begin
4              if(isKeyInput && (key_num==KEY_CODES_Q//key_num==KEY_CODES_A//key_num==KEY_CODES_Z)) begin
5                  next_score = score + 1;
6                  inc = 1;
7              end
8              else next_score = score;
9          end
10 end
```

圖、score control

上圖是控制 score 的 partial code。

因為所有鍵盤適用相同的邏輯，這邊只截圖降 C、C、升 C 以做解釋。

上圖的 Line4，判斷當前是否有鍵盤輸入(isKeyInput)，壓下的按鍵(key\_num)是否為 Q、A、。

若是，則 score + 1，並 active inc；

若否，則 score 不變

inc 會在 Line1 擋下鍵盤連續輸入，導致分數瞬間飆到 99 分的情況。

```
1  if(next_score > 7'd99) next_score = 7'd99;
2  next_seven_segment_nums[19:15] = next_score / 10;
3  next_seven_segment_nums[14:10] = next_score % 10;
4
5  if(!isKeyInput) inc = 0;
```

圖、讓 score 顯示在七段顯示器上

上圖是讓 score 顯示在七段顯示器上的 code。

Line1 控制分數最高為 99

Line2、Line3 分別把 score 的十位數、個位數存到七段顯示器的最左兩格

Line5 即是當放開鍵盤時，lazy inc，這樣下次按對按鍵時才可以加分。

## B. Questions and Discussions (40%)

A. If we want to provide two songs in the DEMONSTRATE mode by using an additional switch to select the song, how would you modify your design of Lab 7?

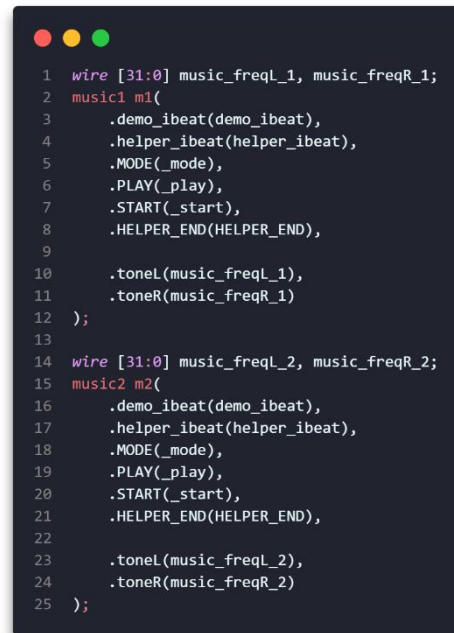


```

1  music m1(
2      .demo_ibeat(demo_ibeat),
3      .helper_ibeat(helper_ibeat),
4      .MODE(_mode),
5      .PLAY(_play),
6      .START(_start),
7      .HELPER_END(HELPER_END),
8
9      .toneL(music_freqL),
10     .toneR(music_freqR)
11 );

```

圖、Demo mode 只有一首歌



```

1  wire [31:0] music_freqL_1, music_freqR_1;
2  music1 m1(
3      .demo_ibeat(demo_ibeat),
4      .helper_ibeat(helper_ibeat),
5      .MODE(_mode),
6      .PLAY(_play),
7      .START(_start),
8      .HELPER_END(HELPER_END),
9
10     .toneL(music_freqL_1),
11     .toneR(music_freqR_1)
12 );
13
14 wire [31:0] music_freqL_2, music_freqR_2;
15 music2 m2(
16     .demo_ibeat(demo_ibeat),
17     .helper_ibeat(helper_ibeat),
18     .MODE(_mode),
19     .PLAY(_play),
20     .START(_start),
21     .HELPER_END(HELPER_END),
22
23     .toneL(music_freqL_2),
24     .toneR(music_freqR_2)
25 );

```

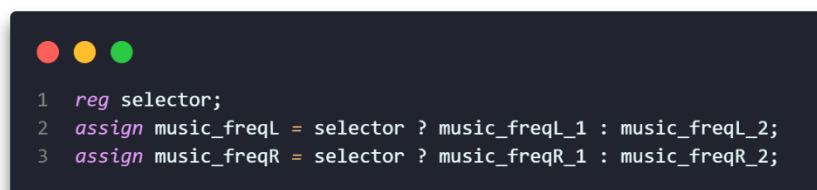
圖、Demo mode 有兩首歌

當 demo mode 只有一首歌的時候，預先寫好的音譜 module 如上左圖。

當有兩首歌可以在 demo mode 展示時，可以寫出上右圖的 module

第一首歌寫在 music1 module，產出 music\_freqL\_1、music\_freqR\_1，代表第一首歌的左右音頻。

第二首歌寫在 music2 module，產出 music\_freqL\_2、music\_freqR\_2，代表第二首歌的左右音頻。



```

1  reg selector;
2  assign music_freqL = selector ? music_freqL_1 : music_freqL_2;
3  assign music_freqR = selector ? music_freqR_1 : music_freqR_2;

```

圖、選擇其中一首 demo 歌

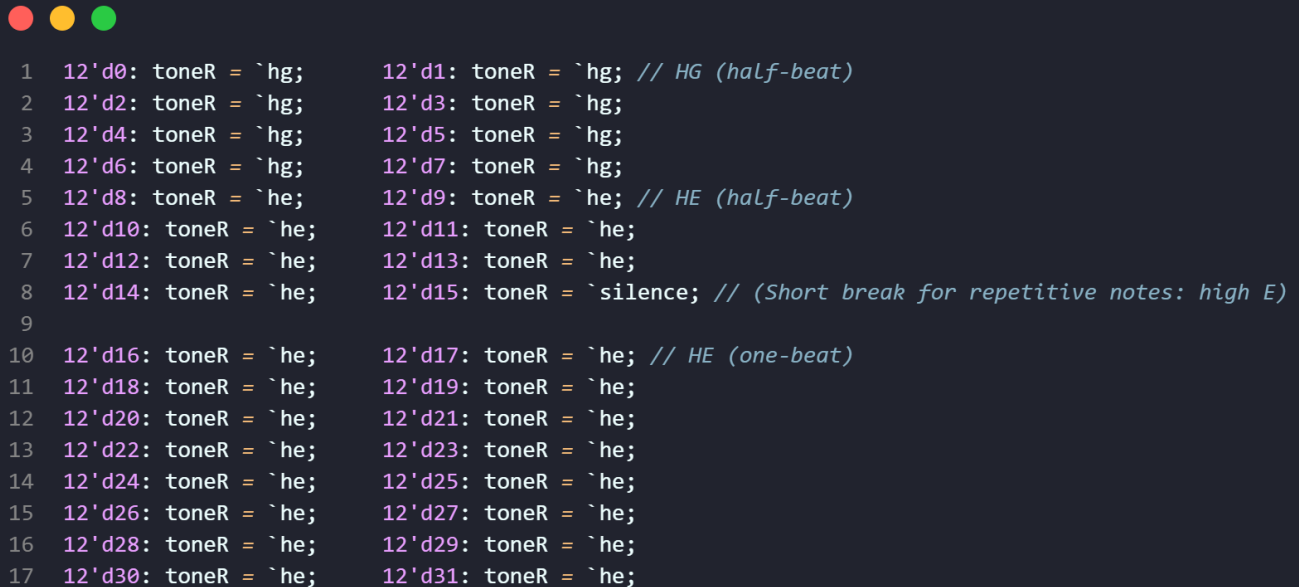
選擇的方法如上圖。真正要播放的音頻（music\_freqL、music\_freqR）用一個 selector 選擇它們的值。後面處理頻率時，處理 music\_freqL、music\_freqR 即可。

B. What happens when we change

next\_ibeat = (ibeat + 1 < LEN) ? (ibeat + 1) : LEN-1;

to

next\_ibeat = (ibeat + 2 < LEN) ? (ibeat + 2) : LEN-1; ?



```

1  12'd0: toneR = `hg;      12'd1: toneR = `hg; // HG (half-beat)
2  12'd2: toneR = `hg;      12'd3: toneR = `hg;
3  12'd4: toneR = `hg;      12'd5: toneR = `hg;
4  12'd6: toneR = `hg;      12'd7: toneR = `hg;
5  12'd8: toneR = `he;      12'd9: toneR = `he; // HE (half-beat)
6  12'd10: toneR = `he;     12'd11: toneR = `he;
7  12'd12: toneR = `he;     12'd13: toneR = `he;
8  12'd14: toneR = `he;     12'd15: toneR = `silence; // (Short break for repetitive notes: high E)
9
10 12'd16: toneR = `he;     12'd17: toneR = `he; // HE (one-beat)
11 12'd18: toneR = `he;     12'd19: toneR = `he;
12 12'd20: toneR = `he;     12'd21: toneR = `he;
13 12'd22: toneR = `he;     12'd23: toneR = `he;
14 12'd24: toneR = `he;     12'd25: toneR = `he;
15 12'd26: toneR = `he;     12'd27: toneR = `he;
16 12'd28: toneR = `he;     12'd29: toneR = `he;
17 12'd30: toneR = `he;     12'd31: toneR = `he;

```

$\text{next\_ibeat} = (\text{ibeat} + 1 < \text{LEN}) ? (\text{ibeat} + 1) : \text{LEN}-1$  會正常的跑完音譜，並停留在最後一個音符。

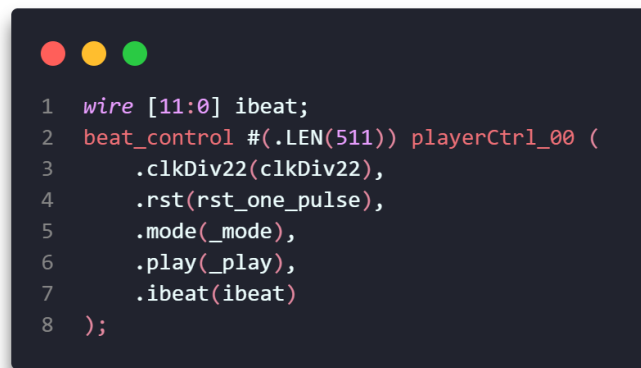
如上圖，toneR 會照著 0 -> 1 -> 2 -> 3 -> 4 -> ... -> LEN 的順序跑過每一個音符，聽覺上我們會聽套一首完整的音樂。

$\text{next\_ibeat} = (\text{ibeat} + 2 < \text{LEN}) ? (\text{ibeat} + 2) : \text{LEN}-1$  會跳過一半音譜，並停留在最後一個音符。

如上圖，toneR 會照著 0 -> 2 -> 4 -> 6 -> 8 -> ... -> LEN 的順序跑。

上圖的 beat 0 ~ beat 7 是 he，ibeat + 2 的寫法跳過 beat 0 ~ beat 7 中的 beat 1、beat 3、beat 5。其餘音符也是如此，每個 8 beat 音符會有一半的 beat 被跳過。

## C. Problem Encountered (10%)



```

1  wire [11:0] ibeat;
2  beat_control #(.LEN(511)) playerCtrl_00 (
3      .clkDiv22(clkDiv22),
4      .rst(rst_one_pulse),
5      .mode(_mode),
6      .play(_play),
7      .ibeat(ibeat)
8  );

```

圖、ibeat 最初的寫法

上圖是最開始 ibeat 的寫法，但這樣寫會讓 demo mode 和 play mode helper 的 beat 互相衝突。

會卡在 demo mode 的歌和 play mode helper 的歌互相影響，demo mode 的 beat 資料和 helper 的 beat 資料會互相洗掉對方。從 demo mode 切到 play mode helper 再切回 demo mode，無法從上次的斷點繼續播放。切到 play mode helper 時也無法從頭開始播放。



```

1  /* ----- */
2  /* ----- demo beat ----- */
3  /* ----- */
4  wire [11:0] demo_ibeat;
5  demo_beat_control #(.LEN(511)) playerCtrl_00 (
6      .clkDiv22(clkDiv22),
7      .rst(rst_one_pulse),
8      .mode(_mode),
9      .play(_play),
10     .demo_ibeat(demo_ibeat)
11 );
12
13
14 /* ----- */
15 /* ----- helper beat ----- */
16 /* ----- */
17 wire [11:0] helper_ibeat;
18 wire HELPER_END;
19 helper_beat_control #(.LEN(511)) playerCtrl_01 (
20     .clkDiv22(clkDiv22),
21     .rst(rst_one_pulse),
22     .mode(_mode),
23     .start(_start),
24     .helper_ibeat(helper_ibeat),
25     .HELPER_END(HELPER_END)
26 );

```

圖、設計兩個 ibeat

上圖是最終的解法，即是幫 demo mode 和 play mode helper 都設計自己的 ibeat。

在 demo mode 則改變 demo\_ibeat;

在 play mode helper 則改變 helper\_ibeat。

再根據 \_mode、\_start、\_play 等 SW 輔助判斷現在是 demo mode 或是 play mode helper，即可順利解決原本兩個 mode 共用 ibeat，互相影響的情況。



## D. Suggestions (5%)

