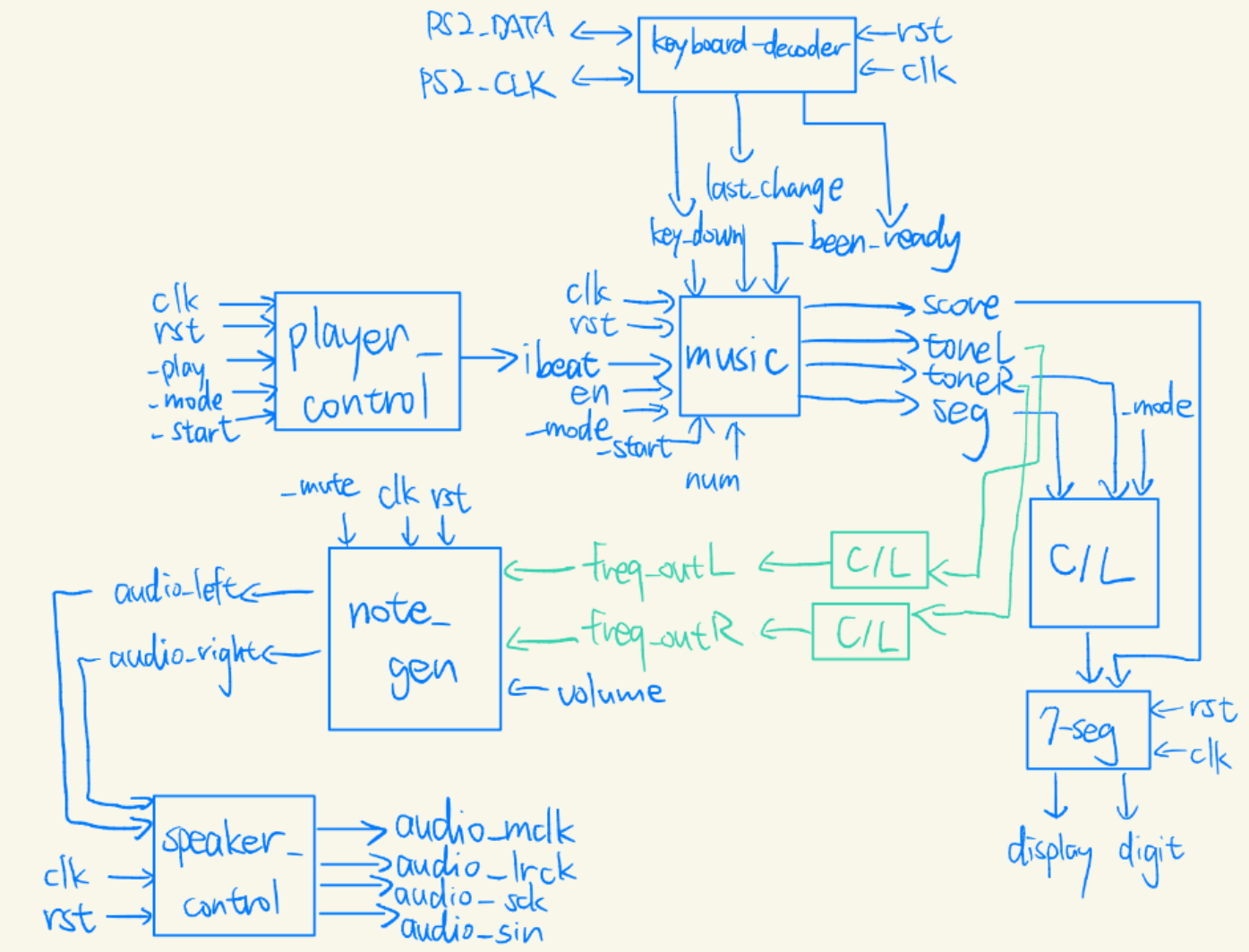
Lab 7

學號: 109021115 姓名: 吳嘉濬

A. Lab Implementation

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



player_control 決定下一個要發出的聲音(note)的編號(ibeatNum)，這 depend on 現在在哪個 mode，如果是 demonstrate mode，音樂會自動播放下去並循環播放；如果是 play mode – piano 和 play mode - helper，則不自動發出聲音，只有在玩家按下對應鍵盤按鍵時才會發出聲音；而在 play mode - helper，歌曲不循環播放且每次進入這個 mode 時，歌曲都會重頭開始顯示(於 7-seg)。

music module 在 input ibeatNum 以及對應的 mode 當中 output 出 toneL, toneR 還有 score, seg(used in helper mode)，其中 toneL, toneR 代表欲發出聲音之左右聲道頻率。

利用 toneR(in demonstrate mode and piano mode)以及 seg(in helper mode)，對應顯示七段顯示器右邊 2 個 digit；score(in helper mode)對應顯示左邊 2 個 digit。

經過 C/L 運算：

```
assign freq_outL = 50000000 / freqL;
assign freq_outR = 50000000 / freqR;
```

處理好的資料成為 module note_gen 的 input，再搭配 input volume 訊號，可以 output 出雙

聲道的 pos/neg edge frequency(決定音頻, do re mi …)和振幅(決定音量)。
再經過相對底層的 module speaker_control 做進一步的處理，最終成功產生想要發出的聲音。

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

a. Volume control

在 note_gen module 底下，利用 input _mute 和 volume 便足以處理 volume control：

```
wire [2:0] vo;
assign vo = (_mute) ? 3'd0 : volume  ;

assign audio_left = (note_div_left == 22'd1) ? 16'h0000 :
                                    (b_clk == 1'b0) ? min : max;
assign audio_right = (note_div_right == 22'd1) ? 16'h0000 :
                                    (c_clk == 1'b0) ? min : max;
```

```
always@* begin
    case(vo)  //volume
    3'd0: begin
        max=16'h0000;
        min=16'h0000;
    end
    3'd1: begin
        max=16'h0400;
        min=16'hfc00;
    end
    3'd2: begin
        max=16'h0800;
        min=16'hf800;
    end
    3'd3: begin
        max=16'h1000;
        min=16'hf000;
    end
    3'd4: begin
        max=16'h2000;
        min=16'he000;
    end
    3'd5: begin
        max=16'h4000;
        min=16'hc000;
    end
    endcase
end
```

`sil 被定義為 500000，經過 C/L 處理(可參考最上面 code segment 截圖)之後 note_div 會等於 1，所以 audio_left/right 會被 assign 成 0；非`sil 的 note 音量大小取決於 case(vo) statement 下的判斷，volume 從 1 到 5，彼此量值成等比數列(2 倍)。如果_mute==1'b1，assign vo = 3'd0。在 case(vo) statement 當中讓 max 和 min 皆等於 0 以達到靜音效果。

b. DEMONSTATE mode

以下反白區域是 module player control 在 demonstrate mode 下會使用到的 code segment：

```
always @* begin
    next_ibeat1=ibeat1;
    next_ibeat0=ibeat0;

    if(!_start) next_ibeat0=0;
    if(_mode) begin
        if(_play) next_ibeat1 = (ibeat1 + 1 < LEN) ? (ibeat1 + 1) : 0;  //!!
        else next_ibeat1 = ibeat1;
    end else begin
        if(_start) begin
            next_ibeat0 = (ibeat0 + 1 < LEN) ? (ibeat0 + 1) : LEN;  //ibeat0==LEN i.e. `sil
        end else begin
            next_ibeat0 = 0;  //piano
        end
    end
end
```

如果_play==1'b1，音樂會持續播放，如果到底了 i.e. ibeat1 + 1 < LEN 不成立時，讓 next_ibeat1 = 0，達到循環播放的效果。

```
always@* begin
    if(_mode) begin
        if(en) begin
        case(ibeatNum)
            // --- Measure 1 ---
                12'd0: toneR = `hg;      12'd1: toneR = `hg; // HG (half-beat)
                12'd2: toneR = `hg;      12'd3: toneR = `hg;
                12'd4: toneR = `hg;      12'd5: toneR = `hg;
                12'd6: toneR = `hg;      12'd7: toneR = `hg;
                12'd8: toneR = `he;      12'd9: toneR = `he; // HE (half-beat)
                12'd10: toneR = `he;     12'd11: toneR = `he;
                12'd12: toneR = `he;     12'd13: toneR = `he;
                12'd14: toneR = `he;     12'd15: toneR = `sil; // (Short break for repetitive notes: high E)

                12'd16: toneR = `he;     12'd17: toneR = `he; // HE (one-beat)
                12'd18: toneR = `he;     12'd19: toneR = `he;
                12'd20: toneR = `he;     12'd21: toneR = `he;
                12'd22: toneR = `he;     12'd23: toneR = `he;
                12'd24: toneR = `he;     12'd25: toneR = `he;
                12'd26: toneR = `he;     12'd27: toneR = `he;
                12'd28: toneR = `he;     12'd29: toneR = `he;
                12'd30: toneR = `he;     12'd31: toneR = `he;
```

ibeat1 的值在 module music 底下被存於 ibeatNum，對應出欲發出聲音之頻率。
之後再經過 note_gen 和 speaker_control 的處理，以完成 demonstrate mode 的實作。

c. PLAY mode – Piano

在 play mode - piano 當中，不會主動發出聲音，對應到下面 code segment 倒數第 4 行：

```
always @* begin
    next_ibeat1=ibeat1;
    next_ibeat0=ibeat0;

    if(!_start) next_ibeat0=0;
    if(_mode) begin
        if(_play) next_ibeat1 = (ibeat1 + 1 < LEN) ? (ibeat1 + 1) : 0;  //!!
        else next_ibeat1 = ibeat1;
    end else begin
        if(_start) begin
            next_ibeat0 = (ibeat0 + 1 < LEN) ? (ibeat0 + 1) : LEN;  //ibeat0==LEN i.e. `sil
        end else begin
            next_ibeat0 = 0;  //piano
        end
    end
end
```

我讓按鍵盤發出聲音的機制設計在 module music，如下圖：

```
end else begin  //_mode==0
        toneR = `sil;
        for(i=0;i<21;i=i+1) begin
            if(key_down[KEY_CODES[i]]) begin
                case(i)
                0: toneR = `hc;
                1: toneR = `hd;
                2: toneR = `he;
                3: toneR = `hf;
                4: toneR = `hg;
                5: toneR = `ha;
                6: toneR = `hb;
                7: toneR = `c;
                8: toneR = `d;
                9: toneR = `e;
                10: toneR = `f;
                11: toneR = `g;
                12: toneR = `a;
                13: toneR = `b;
                14: toneR = `lc;
                15: toneR = `ld;
                16: toneR = `le;
                17: toneR = `lf;
                18: toneR = `lg;
                19: toneR = `la;
                20: toneR = `lb;
                default: toneR = `sil;
                endcase
            end
        end
```

```
parameter [8:0] KEY_CODES [0:20] = {
        9'b0_0001_0101, // Q => 15
        9'b0_0001_1101, // W => 1D
        9'b0_0010_0100, // E => 24
        9'b0_0010_1101, // R => 2D
        9'b0_0010_1100, // T => 2C
        9'b0_0011_0101, // Y => 35
        9'b0_0011_1100, // U => 3C
        9'b0_0001_1100, // A => 1C
        9'b0_0001_1011, // S => 1B
        9'b0_0010_0011, // D => 23
        9'b0_0010_1011, // F => 2B
        9'b0_0011_0100, // G => 34
        9'b0_0011_0011, // H => 33
        9'b0_0011_1011, // J => 3B
        9'b0_0001_1010, // Z => 1A
        9'b0_0010_0010, // X => 22
        9'b0_0010_0001, // C => 21
        9'b0_0010_1010, // V => 2A
        9'b0_0011_0010, // B => 32
        9'b0_0011_0001, // N => 31
        9'b0_0011_1010  // M => 3A
    };
```

按下按鍵 output 出對應的 toneR, toneL 值以發出正確的聲音。

d. PLAY mode – Helper

先前兩個 mode 的需求允許我可以讓 toneR 的值和 7-seg 要顯示的畫面綁在一起，但是在 helper mode 當中七段顯示器的值和發出的聲音不再同步了，所以我必須另外設計 reg 去各自實作出相對應的功能。詳細作法我把它擺在 C part: problem encounter 那邊。

至於我如何在 demonstrate mode 和 helper mode 來回切換時還能儲存 demonstrate mode 的"進度”，我是利用兩個不同的 ibeat 去各自存取/更新進度，如下圖：

```
always @(posedge clk, posedge reset) begin
    if (reset) begin
        ibeat1 <= 0;
        ibeat0 <= 0;
    end else begin
        ibeat1 <= next_ibeat1;
        ibeat0 <= next_ibeat0;
    end
end

always @* begin
    next_ibeat1=ibeat1;
    next_ibeat0=ibeat0;

    if(!_start) next_ibeat0=0;
    if(_mode) begin
        if(_play) next_ibeat1 = (ibeat1 + 1 < LEN) ? (ibeat1 + 1) : 0;  //!!
        else next_ibeat1 = ibeat1;
    end else begin
        if(_start) begin
            next_ibeat0 = (ibeat0 + 1 < LEN) ? (ibeat0 + 1) : LEN;  //ibeat0==LEN i.e. `sil
        end else begin
            next_ibeat0 = 0;  //piano
        end
    end
end
```

ibeat1 存 demonstrate mode 的 ibeat；ibeat0 存 play mode - helper 的 ibeat。
helper mode 在 led 燈上提示 node 位置的機制，設計於以下 code：

```
always@* begin
    next_nled=nled;

    if(_mode) next_nled=7'b0;
    else if(!_mode && !_start) next_nled=7'b0;
    else begin
        case(seg)
            `hc, `c, `lc: next_nled=7'b1000000;
            `hd, `d, `ld: next_nled=7'b0100000;
            `he, `e, `le: next_nled=7'b0010000;
            `hf, `f, `lf: next_nled=7'b0001000;
            `hg, `g, `lg: next_nled=7'b0000100;
            `ha, `a, `la: next_nled=7'b0000010;
            `hb, `b, `lb: next_nled=7'b0000001;
            `silence    : next_nled=7'b0000000;

            default     : next_nled=7'b0000000;
        endcase
    end
    if(ibeatNum==512) next_nled=7'b1111111;
end
```

利用 seg 的值去判斷 led 需要顯示的位置。(seg 類似於 toneR，詳情請看 part C)
七段顯示器右邊兩個 digit 的顯示方式也是依據 seg 的值，如下圖：

```
end else begin  //helper mode
    next_nums=nums;

    next_nums[15:12]=score/10;
    next_nums[11:8]=score%10;
    case(seg)
    `hc: begin  //C5
        next_nums[7:4]=4'd11;
        next_nums[3:0]=4'd5;
    end
    `hd: begin
        next_nums[7:4]=4'd12;
        next_nums[3:0]=4'd5;
    end
    `he: begin
        next_nums[7:4]=4'd13;
        next_nums[3:0]=4'd5;
    end
```

e. Score mechanism of the Helper

判斷得分的機制如下圖：

```
always@* begin
    next_score=score;

    if(!_mode && _start && ibeatNum==0) next_score=0;
    if(been_ready && toneR==seg && toneR!=`sil) next_score= (score<99) ? score+1 : score ;  //!!
end
```

這一段 code 在 module music 底下。如果 toneR==seg，也就是 speaker 發出的聲音(note)等於當下 ibeatNum 對應出的 preset note 的值時，並且不是`sil note，則判斷為得分。

七段顯示器左邊兩個 digit 顯示 helper mode 時的得分狀況，其機制如下：

```
end else begin  //helper mode
    next_nums=nums;

    next_nums[15:12]=score/10;
    next_nums[11:8]=score%10;
```

我在 module music 中 output score 值，並在 top module 中做運算，讓最左邊的 7-seg digit 顯示十位數的分數(in base 10)，第二個 7-seg digit 顯示個位數的分數(in base 10)。

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?

在 module music 當中新增 input select (module music 是 output toneL, toneR 的 module)，利用 if(select)判斷式去 branch 出兩大類(兩首不同曲子)，之後再根據 ibeatNum 去得到對應的 toneL, toneR。如果考慮歌曲播放當中會來回切換歌曲，並希望曲子可以在斷掉的地方繼續播放下去，那麼就必須再新增 input [11:0] ibeatNum2，個別記錄兩個曲子播放的進度以決定下一個要 output 的 toneL, toneR。

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;

?

在 clk 頻率不變的情況下，相當於 ibeat 每經過一個 clock cycle 便增加 2，整個 ibeat array 被 traverse 的速度變成原本 2 倍，也就是整首歌的速度會變成原本的 2 倍。需要注意的是，在這樣的情況下 ibeat 會一直保持偶數(或奇數)，所以如果遇到連續但分開的同音頻聲音，原本 1/16 拍的`sil 必須改成連續兩個`sil 以避免被跳過，造成聲音連在一起的情況。

C. Problem Encountered (10%)

實作 demonstrate mode 和 play mode - piano 時，我利用 module music output 的 toneL 和 toneR 去判斷 7-seg 要顯示的對應畫面，如下圖：

```
always@* begin
    next_nums=nums;
    if(!_play) begin
        next_nums={4'd10, 4'd10, 4'd10, 4'd10};
    end
    if(!_mode && _start && ibeatNum==0) begin
        next_nums={4'd0, 4'd0, 4'd10, 4'd10};  //at the start of helper mode: 00--
        //score=6'd0;
    end
    if(_mode || !_start) begin
        next_nums[15:8]={4'd10, 4'd10};
    case(freqR)
        `hc: begin  //C5
            next_nums[7:4]=4'd11;
            next_nums[3:0]=4'd5;
        end
        `hd: begin
            next_nums[7:4]=4'd12;
            next_nums[3:0]=4'd5;
        end
        `he: begin
            next_nums[7:4]=4'd13;
            next_nums[3:0]=4'd5;
        end
        `hf: begin
            next_nums[7:4]=4'd14;
            next_nums[3:0]=4'd5;
        end
        `hg: begin
            next_nums[7:4]=4'd9;
            next_nums[3:0]=4'd5;
        end
```

在 top module 當中對應 toneL, toneR 的 wire signal 是 freqL, freqR，我利用 freqR 的值，也就是 melody segment(主旋律)的 note 去顯示對應的 7-seg。

但是這個方法不適用於設計 play mode - helper，因為 freqR 的值一旦被設為非`sil，其對應的 note 就會發出相對應的聲音，所以我在 helper 模式當中必須使用別的變數(signal)去決定 7-seg 要顯示的值，而非前兩個模式所使用的 freqR。
所以我另外在 module music 當中設新的 reg [13:0] seg，他被 assign 的值和 toneR 是一模一樣的，但是 seg 卻不會直接讓 speaker 發出聲音(seg 只影響 7-seg 的顯示，但 toneR i.e. freqR 會直接讓聲音發出)，所以我在 top module 利用一樣的方式去決定 7-seg 要顯示的畫面，如下圖：

```
end else begin  //helper mode
    next_nums=nums;

    next_nums[15:12]=score/10;
    next_nums[11:8]=score%10;
    case(seg)
    `hc: begin  //C5
        next_nums[7:4]=4'd11;
        next_nums[3:0]=4'd5;
    end
    `hd: begin
        next_nums[7:4]=4'd12;
        next_nums[3:0]=4'd5;
    end
    `he: begin
        next_nums[7:4]=4'd13;
        next_nums[3:0]=4'd5;
    end
    `hf: begin
        next_nums[7:4]=4'd14;
        next_nums[3:0]=4'd5;
    end
    `hg: begin
        next_nums[7:4]=4'd9;
        next_nums[3:0]=4'd5;
    end
    `ha: begin
        next_nums[7:4]=4'd15;
        next_nums[3:0]=4'd5;
    end
    `hb: begin
        next_nums[7:4]=4'd6;
        next_nums[3:0]=4'd5;
    end
```

如此一來，便解決發出的聲音和對應 7-seg 顯示的值被互相綁住的困境。

D. Suggestions (5%)

感謝教授以及助教提供我實作 lab 時的協助。