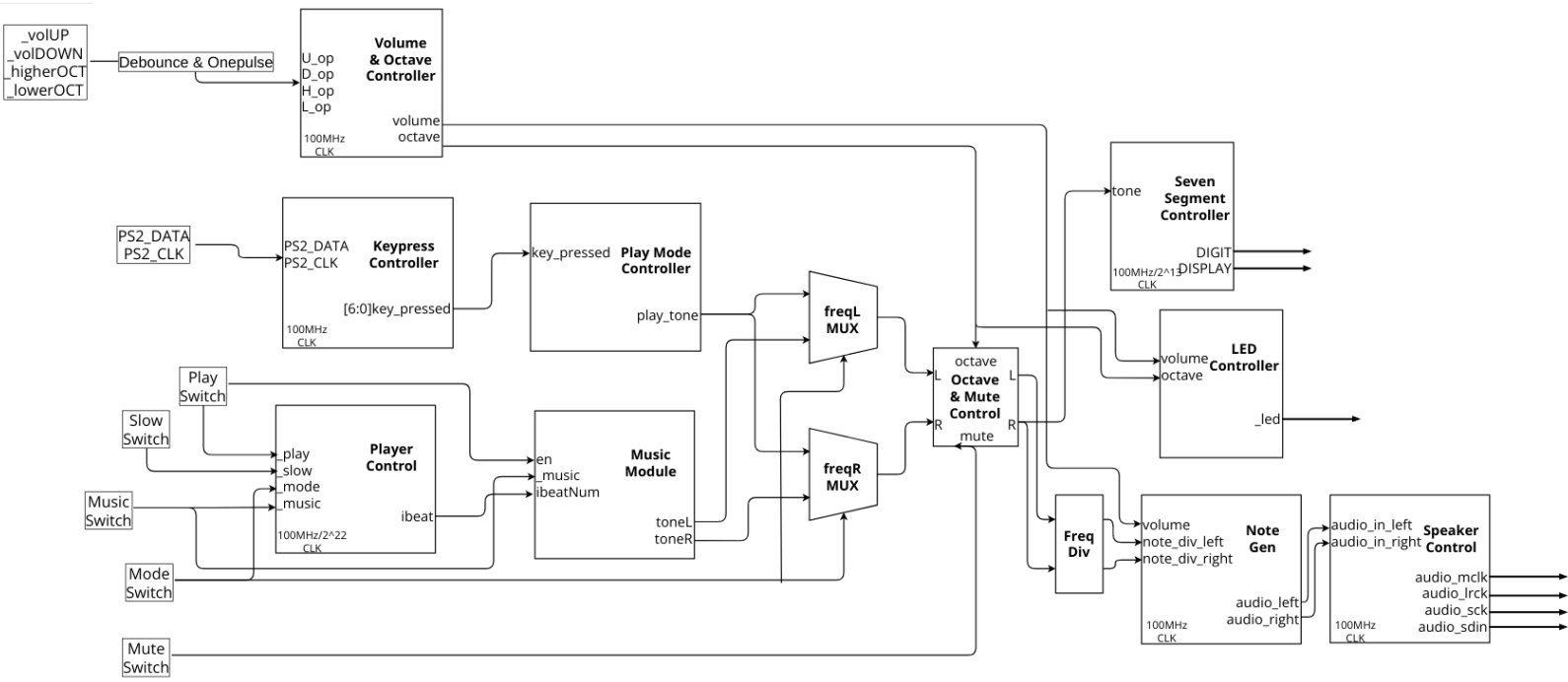
Lab 8

學號: 109062302 姓名: 郭品毅

1. 實作過程



- PLAY Mode:
  - 由 Keypress Controller 及 Play Mode Controller 負責計算及產生輸出聲音頻率

- Keypress Controller:

```
module keypress_controller (
    input clk,
    input rst,
    inout PS2_DATA,
    inout PS2_CLK,
    output reg [6:0] key_pressed
);

    wire [511:0] key_down;
    wire [8:0] last_change;
    wire been_ready;

    KeyboardDecoder key_de (
        .key_down(key_down),
        .last_change(last_change),
        .key_valid(been_ready),
        .PS2_DATA(PS2_DATA),
        .PS2_CLK(PS2_CLK),
        .rst(rst),
        .clk(clk)
    );

    always @(posedge clk, posedge rst) begin
        if (rst) begin
            key_pressed = 0;
        end else begin
            if (been_ready) begin
                case (last_change)
                    {1'b0, 8'h1c}: // L Shift
                        key_pressed[0] = key_down[last_change];
                    {1'b0, 8'h1b}: // [
                        key_pressed[1] = key_down[last_change];
                    {1'b0, 8'h23}: // ]
                        key_pressed[2] = key_down[last_change];
                    {1'b0, 8'h2b}: // K
                        key_pressed[3] = key_down[last_change];
                    {1'b0, 8'h34}: // L
                        key_pressed[4] = key_down[last_change];
                    {1'b0, 8'h33}: // ;
                        key_pressed[5] = key_down[last_change];
                    {1'b0, 8'h3b}: // ;
                        key_pressed[6] = key_down[last_change];
                endcase
            end
        end
    end
endmodule
```

- 負責控制鍵盤訊號，並轉換為一個類似 push button 的輸出
  - 使用 FF 儲存按鍵狀態
  - 當按鍵按下時變為 1, 直到放開才變為 0

- Play Mode Controller

```
module play_mode_controller (
    input [6:0] key_pressed,
    output reg [31:0] play_tone
);

    always @* begin
        if (key_pressed[0])
            play_tone = `C4;
        else if (key_pressed[1])
            play_tone = `D4;
        else if (key_pressed[2])
            play_tone = `E4;
        else if (key_pressed[3])
            play_tone = `F4;
        else if (key_pressed[4])
            play_tone = `G4;
        else if (key_pressed[5])
            play_tone = `A4;
        else if (key_pressed[6])
            play_tone = `B4;
        else
            play_tone = `sil;
    end

endmodule
```

- 負責透過現在鍵盤按鍵的狀態，輸出要播放聲音的頻率
- 有 priority, 如果有多個按鍵同時按，會輸出最低的頻率

○ DEMONSTRATION Mode

- 由 Player Control 及 Music Module 負責計算及產生輸出聲音頻率
- Player Control

```
module player_control (
    input clk,
    input reset,
    input _play,
    input _slow,
    input _mode,
    input _music,
    output reg [11:0] ibeat
);
    parameter LEN = 512;
    reg [11:0] next_ibeat;
    reg slow_flag, next_slow_flag;
    reg cur_music, next_cur_music;

    always @(posedge clk, posedge reset) begin
        if (reset) begin
            ibeat <= 0;
            slow_flag <= 0;
            cur_music <= 0;
        end else if (_play && _mode == 1) begin
            ibeat <= next_ibeat;
            slow_flag <= next_slow_flag;
            cur_music <= next_cur_music;
        end
    end

    always @* begin
        if (_music != cur_music) begin
            next_ibeat = 0;
            next_slow_flag = 0;
            next_cur_music <= _music;
        end else begin
            next_cur_music <= cur_music;
            if (_slow && !slow_flag) begin
                next_ibeat = ibeat;
                next_slow_flag = 1;
            end else begin
                next_ibeat = (ibeat + 1 != LEN) ? (ibeat + 1) : 0;
                next_slow_flag = 0;
            end
        end
    end

endmodule
```

- 儲存了 3 個狀態, 且只有在 _play (非暫停) 且 _mode==1 (Demo Mode) 時才會改變、運作
  - ibeat: 目前播放的聲音位置 (第幾拍)
    - 當位置超過歌曲長度 (512), 即回到 0, 產生重複播放的效果
  - slow_flag: slow 模式時，一拍要多停留一個 cycle
    - 第一個 cycle 時，將 show_flag 設為 1, 第二個 cycle 才跳到下一拍
  - cur_music: 記錄現在播放的曲目
    - 如果切換曲目 (cur_music != _music), 就從頭開始

- Music Module
  - ```
    module music_example (
        input [11:0] ibeatNum,
        input en,
        input _music,
        output reg [31:0] toneL,
        output reg [31:0] toneR
    );

        always @* begin
            if (_music == 0) begin
                if(en == 1) begin
                    case(ibeatNum) ...
                    endcase
                end else begin
                    toneR = `sil;
                end
            end else begin
                if(en == 1) begin
                    case(ibeatNum) ...
                    endcase
                end else begin
                    toneR = `sil;
                end
            end
        end

        always @(*) begin
            if (_music == 0) begin
                if(en == 1)begin
                    case(ibeatNum) ...
                    endcase
                end else begin
                    toneL = `sil;
                end
            end else begin
                if(en == 1)begin
                    case(ibeatNum) ...
                    endcase
                end else begin
                    toneL = `sil;
                end
            end
        end
    endmodule
    ```
  - 負責從目前的時間位置 (ibeat) 找到對應的音符，並輸出頻率
  - 將 _play (是否暫停) 接為 en，暫停的時候才不會一直發出聲音
- freqL, freqR MUX
  - ```
    assign prefreqL = (_mode == 0)? play_tone: demo_freqL;
    assign prefreqR = (_mode == 0)? play_tone: demo_freqR;
    ```
  - 根據現在的模式選擇採用 Play Mode 還是 Demo Mode 的頻率輸出
- Octave & Mute Control
  - ```
    always @(posedge clkDiv13, posedge rst) begin
        if (rst) begin
            freqL <= `sil;
            freqR <= `sil;
        end else begin
            freqL <= (_mute || prefreqL == `sil)? `sil: (octave == 1)? (prefreqL / 2): (octave == 3)? (prefreqL * 2): prefreqL;
            freqR <= (_mute || prefreqR == `sil)? `sil: (octave == 1)? (prefreqR / 2): (octave == 3)? (prefreqR * 2): prefreqR;
        end
    end
    ```
  - 根據 mute 狀態選擇是否靜音
  - 根據 octave 狀態調整音高
  - 使用 assign 可能會造成 Combinational Circuit 比較複雜，電路不穩定，會產生奇怪的聲音，因此在輸出接上 FF，用來穩定輸出，這樣就改善很多了
    ```

- Volume & Octave Controller

```
always @(posedge clk, posedge rst) begin
    if (rst) begin
        volume <= 3;
        octave <= 2;
    end else begin
        if (U_op && volume != 5)
            volume <= volume + 1;
        if (D_op && volume != 1)
            volume <= volume - 1;
        if (H_op && octave != 3)
            octave <= octave + 1;
        if (L_op && octave != 1)
            octave <= octave - 1;
    end
end
```

  -
  - 負責記錄和改變 volume 及 octave 的狀態
- NoteGen
  - 負責將頻率轉為方波
    - 先用 50000000 / 頻率, 計算切換的週期
    - 根據週期變換輸出的 high 和 low
    - 透過改變 high 和 low 的幅度來控制音量

```
always @* begin
    case (volume)
        5: begin
            next_high = 16'h4000;
            next_low = 16'hc000;
        end
        4: begin
            next_high = 16'h1000;
            next_low = 16'hf000;
        end
        2: begin
            next_high = 16'h0100;
            next_low = 16'hff00;
        end
        1: begin
            next_high = 16'h0020;
            next_low = 16'hffe0;
        end
        default: begin // 3
            next_high = 16'h0400;
            next_low = 16'hfc00;
        end
    endcase
end

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

    -
    - 由於直接使用 Combinational Circuit 的輸出也會讓訊號不穩定產生雜音，因此也將 high 及 low 接上 FF 來穩定輸出

- Seven Segment Controller
  - ```
    module seven_seg_controller (
        input clk_display,
        input [31:0] tone,
        output reg [3:0] DIGIT,
        output reg [6:0] DISPLAY
    );

        reg [3:0] note;
        reg [3:0] notation;

        parameter sharp = 7;
        parameter flat = 8;
        parameter c = 0;
        parameter d = 1;
        parameter e = 2;
        parameter f = 3;
        parameter g = 4;
        parameter a = 5;
        parameter b = 6;
        parameter nothing = 10;

        always @* begin
            notation = nothing;
            note = nothing;
            case (tone)
                `C5, `C3, `C4, `C2:
                    note = c;
                `D5, `D3, `D4:
                    note = d;
                `Ef3, `E4, `Ef5, `Ef4:
                    note = e;
                `F3, `F4, `F2, `F5:
                    note = f;
                `G3, `G5, `G4, `G2:
                    note = g;
                `A4, `Af2, `A2, `Af4, `Af3:
                    note = a;
                `Bf4, `B4, `B2, `Bf2, `Bf3:
                    note = b;
    ```
  - ```
            endcase
            case (tone)
                `Ef4, `Ef5, `Bf2, `Af4, `Af2, `Ef3, `Bf3, `Bf4, `Af3:
                    notation = flat;
            endcase
        end

        reg [3:0] value;

        always @(posedge clk_display) begin
            case (DIGIT)
                4'b1110: begin
                    value = notation;
                    DIGIT = 4'b1101;
                end
                4'b1101: begin
                    value = nothing;
                    DIGIT = 4'b1011;
                end
                4'b1011: begin
                    value = nothing;
                    DIGIT = 4'b0111;
                end
                4'b0111: begin
                    value = note;
                    DIGIT = 4'b1110;
                end
                default: begin
                    value = note;
                    DIGIT = 4'b1110;
                end
            endcase
        end

        always @* begin
            case (value)
                4'd0: DISPLAY = 7'b0100111; // c
                4'd1: DISPLAY = 7'b0100001; // d
                4'd2: DISPLAY = 7'b0000110; // E
                4'd3: DISPLAY = 7'b0001110; // F
                4'd4: DISPLAY = 7'b1000010; // G
                4'd5: DISPLAY = 7'b0100000; // a
                4'd6: DISPLAY = 7'b0000011; // b
                4'd7: DISPLAY = 7'b0011100; // #
                4'd8: DISPLAY = 7'b0000011; // b
                default: DISPLAY = 7'b0111111; // -
            endcase
        end

    endmodule
    ```
  - 根據右聲道要發出的頻率來決定顯示的 note 及 notation

- LED Controller

```
always @* begin
    _led = 0;
    if (!_mute) begin
        _led[0] = 1;
        _led[1] = volume != 1;
        _led[2] = volume != 1 && volume != 2;
        _led[3] = volume == 4 || volume == 5;
        _led[4] = volume == 5;
    end

    _led[16-octave] = 1;
end
```

  - 根據 volume 及 octave 計算那些 led 要亮

- 其他：譜面 code 產生程式
  - 因為要自己打每個音的長度太麻煩了，因此希望能找到更簡單產生 code
  - 我發現打譜軟體 (如 MuseScore) 有一個通用的格式叫 MusicXML, 打開來看發現需要的資訊 (音符/休止符、音調、長度) 都在裡面了，這樣自己用程式就很好 parse 了

```
<note default-x="81.69" default-y="-25.00">
  <pitch>
    <step>C</step>
    <octave>3</octave>
    </pitch>
  <duration>1</duration>
  <voice>1</voice>
  <type>quarter</type>
  <stem>up</stem>
  </note>
<note default-x="105.71" default-y="-5.00">
  <pitch>
    <step>G</step>
    <octave>3</octave>
    </pitch>
  <duration>1</duration>
  <voice>1</voice>
  <type>quarter</type>
  <stem>down</stem>
  </note>
<note default-x="129.74" default-y="-40.00">
  <pitch>
    <step>G</step>
    <octave>2</octave>
    </pitch>
  <duration>1</duration>
  <voice>1</voice>
  <type>quarter</type>
  <stem>up</stem>
  </note>
```

  - 而且又發現 Python 已經有 library 叫 music21 可以直接處理 MusicXML, 又更方便了
  - 因此寫了簡單的 Python Script 將 MusicXML 轉換為我們要用的 code:
  - 這樣只要用 Musescore 把譜打上去，匯出 MusicXML, 就可以轉成 code 了
  - 還可以上網找譜，很快的轉成 code！

2. 學到的東西與遇到的困難
   - 這次學會了聲音模組的使用方式！
     - 基本上只要給出要發出的頻率，剩下的 code 都可以重複利用
   - 遇到了電路輸出不穩導致發出雜音
     - 之前老師就已經常常舉例輸出訊號不穩導致產生奇怪行為的例子了
     - 這次我終於也發生了
     - 一直發出雜音，找了很久都找不到原因，然後才發現有一些輸出是從 Combinational Circuit 接出來，最後把這些輸出都加上 Flip Flops 雜音就消失了！
3. 想對老師或助教說的話
   - 謝謝老師和助教～
   - 感謝助教出了那麼多有趣的題目！