EECS 2070 02 Digital Design Labs 2019 Lab 8
學號：104021215　姓名：熊磊

1. 實作過程
   - 先利用 clock_divider 做出想要的頻率，然後將 input 中的 volumeUP, volumeDOWN, rst 做 debounce 及 one pulse 的處理。

   - 音量是用 volume 這個 register 來儲存。調整 volume 的方式是用 always block 來處理，再送到 note_gen 這個 module 來處理。音量總共分 5 個 level，選擇上沒有特別設計，單純用聽的來決定音量大小。ampH 和 ampL 互為 2's complement。

```
always@(posedge clkDiv13, posedge rst_d)begin
    if(rst_d)
        volume = 3;
    else if(v_up_1pulse)
        volume = (volume == 5) ? 5 : volume + 1;
    else if(v_down_1pulse)
        volume = (volume == 1) ? 1 : volume - 1;
    else
        volume = (volume>0) ? volume : 3;
end

assign _led_vol = (volume == 1) ? 5'b0_0001 :
                  (volume == 2) ? 5'b0_0011 :
                  (volume == 3) ? 5'b0_0111 :
                  (volume == 4) ? 5'b0_1111 :
                  (volume == 5) ? 5'b1_1111 : 5'b0_0111;
```

```
always @*
    case(volume)
        1: begin
            ampH <= 16'h0040;
            ampL <= 16'hFFC0;
        end
        2: begin
            ampH <= 16'h00A0;
            ampL <= 16'hFF60;
        end
        3: begin
            ampH <= 16'h0A00;
            ampL <= 16'hF600;
        end
        4: begin
            ampH <= 16'h4000;
            ampL <= 16'hC000;
        end
        5: begin
            ampH <= 16'h7000;
            ampL <= 16'h9000;
        end
        default: begin
            ampH <= 16'h00C0;
            ampL <= 16'hFFC0;
        end
    endcase
```

   - 靜音則是在送進 note_gent 前，就先判斷_mute 是否開啟，走得根據 _music 把對應的音樂 signal 送到 note_gen 產生。

```
// Note gen makes no sound, if freq_out = 50000000 / `silence = 1
assign freq_outL = 50000000 / (_mute ? `silence : (_music) ? freqL1 : freqL0);
assign freq_outR = 50000000 / (_mute ? `silence : (_music) ? freqR1 : freqR0);
```

   - 使用 player_control 來控制音樂進度、循環、播放、暫停，_repeat 在 player_control 用來判斷當播放進度結束時(ibeat>= LEN)，是否要重新播放。若是，則將 ibeat 歸零，否則設為 12'd511。_play 用來控制 next_ibeat 是否要+1。

```
always @* begin
    next_ibeat = (_play) ? (ibeat + 1 < LEN) ? (ibeat + 1) : (_repeat) ? 12'd0 : 12'd511 : ibeat;
end
```

   - 而為了方便產生對應的樂譜頻率，我額外寫了一個 python 程式來把簡譜轉成 verilog signal。

In [2]:
```python
n = 0
for i in s:
    if '.-' in i:
        M = i.split('.-')
        for c in range(11):
            print(f'12\'d{n+c}: toneR = `{M[0]};')
        print(f'12\'d{n+11}: toneR = `sil;')
        n = n+12
        for c in range(3):
            print(f'12\'d{n+c}: toneR = `{M[1]};')
        print(f'12\'d{n+3}: toneR = `sil;')
        n = n+4

    elif '-' in i:
        M = i.split('-')
        prev = M[0]
        start = 0
        for j in M:
            if prev == j and start!=0:
                print(f'12\'d{n}: toneR = `sil;')
                n = n+1
            elif start==0:
                start = 1
            else:
                print(f'12\'d{n}: toneR = `{prev};')
                n = n+1
            for c in range(7):
                print(f'12\'d{n+c}: toneR = `{j};')
            n = n+7
            prev = j
        print(f'12\'d{n}: toneR = `sil;')
        n = n+1

    elif '--' in i:
        M = i.split('--')
        for c in range(31):
            print(f'12\'d{n+c}: toneR = `{M[0]};')
        print(f'12\'d{n+31}: toneR = `sil;')
        n = n+32
    else:
        for c in range(15):
            print(f'12\'d{n+c}: toneR = `{i};')
        print(f'12\'d{n+15}: toneR = `sil;')
        n = n+16
```

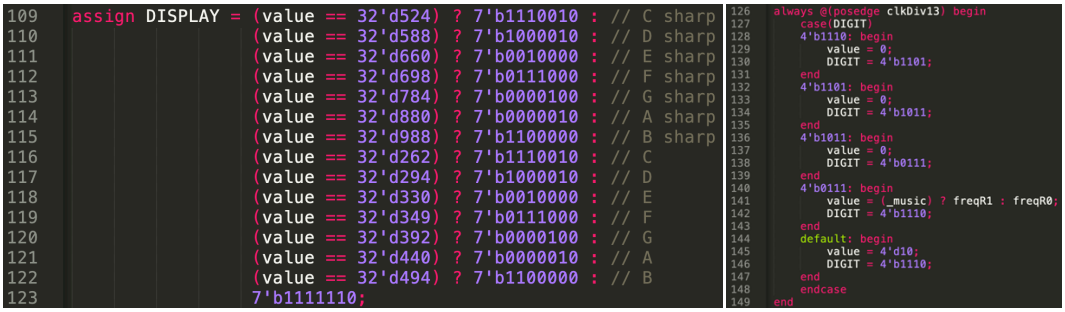
- 兩首音樂實現的方式是，直接寫兩個 module，然後在送進 note_gen 實再決定要拿哪一個 signal。

```
154  // Music module
155  music music_data ( .ibeatNum(ibeatNum), .en(~_play), .toneL(freqL0), .toneR(freqR0));
156  music1 music_data1 ( .ibeatNum(ibeatNum), .en(~_play), .toneL(freqL1), .toneR(freqR1));
```

- 另外用 FSM 來控制音樂狀態。如果再從 music0 切換到 music1，player_control 要做 reset，因此分了 3 個 states。

```
78   always@(*) begin
79       case(state)
80           `INIT: begin
81               reset1 = 1;
82               next_state = (_music) ? `MUSIC1 : `MUSIC0;
83           end
84           `MUSIC0: begin
85               reset1 = 0;
86               next_state = (_music) ? `INIT : `MUSIC0;
87           end
88           `MUSIC1: begin
89               reset1 = 0;
90               next_state = (_music) ? `MUSIC1 : `INIT;
91           end
92           default:  begin
93               reset1 = 0;
94               next_state = `INIT;
95           end
96       endcase
97   end
98   always@(posedge clkDiv13, posedge rst_d)
99   begin
100      if(rst_d==1) begin
101          state = `INIT;
102      end
103      else begin
104          state = next_state;
105      end
106  end
```

- 七段顯示器就依照目前_music 的狀態來顯示右聲道 toneR 播放的音符。



```
assign DISPLAY = (value == 32'd524) ? 7'b1110010 : // C sharp
                 (value == 32'd588) ? 7'b1000010 : // D sharp
                 (value == 32'd660) ? 7'b0010000 : // E sharp
                 (value == 32'd698) ? 7'b0111000 : // F sharp
                 (value == 32'd784) ? 7'b0000100 : // G sharp
                 (value == 32'd880) ? 7'b0000010 : // A sharp
                 (value == 32'd988) ? 7'b1100000 : // B sharp
                 (value == 32'd262) ? 7'b1110010 : // C
                 (value == 32'd294) ? 7'b1000010 : // D
                 (value == 32'd330) ? 7'b0010000 : // E
                 (value == 32'd349) ? 7'b0111000 : // F
                 (value == 32'd392) ? 7'b0000100 : // G
                 (value == 32'd440) ? 7'b0000010 : // A
                 (value == 32'd494) ? 7'b1100000 : // B
                 7'b1111110;
```

```
always @(posedge clkDiv13) begin
    case(DIGIT)
    4'b1110: begin
        value = 0;
        DIGIT = 4'b1101;
    end
    4'b1101: begin
        value = 0;
        DIGIT = 4'b1011;
    end
    4'b1011: begin
        value = 0;
        DIGIT = 4'b0111;
    end
    4'b0111: begin
        value = (_music) ? freqR1 : freqR0;
        DIGIT = 4'b1110;
    end
    default: begin
        value = 4'd10;
        DIGIT = 4'b1110;
    end
    endcase
end
```

2. 學到的東西與遇到的困難

   這次 lab 一開始沒想到要用 finite state machine，所以花了很多力氣在想要怎麼換歌的時候重新播放，後來決定直接用 FSM 寫。在產生音樂訊號的地方因為太長了，果斷寫一個 python 來產生 XD。

3. 想對老師或助教說的話

   謝謝老師與助教!