CS207_Digital logic_Project_Mini Piano

项目A 电子琴学习机

陈彦妤 贾适萌 杨若谷

1 团队分工

1.1 成员工作列表

陈彦妤：负责自由模式，完成高低八度、按键调整等功能；

贾适萌：负责自动演奏模式，完成读秒器、音乐库调整、显示曲目名称等功能；

杨若谷：负责学习模式，完成实时调整评级、学霸模式等功能。

1.2 贡献百分比

三位成员贡献比一致，均为33.3%。

2 开发计划进度和实施状态

2.1 项目开发计划

2023.12.15 - 12.17 开始理解和和设计项目架构；

2023.12.17之前完成项目的分工；

2023.12.20 - 12.23 实现基础模式中的自由模式和自动演奏；

2023.12.20 - 12.25 实现学习模式；

2023.12.26 - 2024.1.1 实现bonus中的按键调整，VGA显示曲目名称；

2024.1.1 - 1.3　报告撰写以及代码整合；

2.2 实施状态

2023.12.2 第一次小组讨论，确认选题为电子琴；

2023.12.8 学习常量文件的导入；

2023.12.16 第二次小组讨论，彼此了解成员进度；

2023.12.18 完成自动演奏模式的小星星和欢乐颂；

2023.12.19 完成自由模式；

2023.12.20 完成高低八度的调整；

2023.12.21 完成buzzer的调整以及自动演奏显示歌曲序号；

2023.12.23 自动演奏音乐库添加到四首；

2023.12.25 基本实现学习模式；

2023.12.26 自动演奏显示歌曲名称，开始进行代码整合，初步实现top文件；

2023.12.29 学习模式实现评级和用户；

2023.1.1 实现按键调整功能；

2024.1.3 第三次小组讨论，整合代码；

2024.1.4 实现了学霸模式，准备最后的答辩；

3 系统功能列表

3.1 基本功能

3.1.1 自由模式

```
`include "header.vh"
module free_mode(//buzzer play certain note when pressed,the correspongding
light work as well
input do,re,mi,fa,so,la,si,
input wire clk,rst_n,
input [1:0] mode,
output reg l_1,l_2,l_3,l_4,l_5,l_6,l_7,
output speaker
    );
    reg [3:0] note;
    initial begin
    note = 3'b000;
    l_1 = 1'b0; l_3 = 1'b0;
    l_2 = 1'b0; l_4 = 1'b0;
    l_5 = 1'b0; l_6 = 1'b0;
    l_7 = 1'b0;
    end
    always @(*) begin
    case({do,re,mi,fa,so,la,si})
    7'b0000001:begin  note = 4'b0001; l_7 = 1'b1; end
    7'b0000010:begin  note = 4'b0010; l_6 = 1'b1; end
    7'b0000100:begin  note = 4'b0011; l_5 = 1'b1; end
    7'b0001000:begin  note = 4'b0100; l_4 = 1'b1; end
    7'b0010000:begin  note = 4'b0101; l_3 = 1'b1; end
    7'b0100000:begin  note = 4'b0110; l_2 = 1'b1; end
    7'b1000000:begin  note = 4'b0111; l_1 = 1'b1; end
    default:begin note = 4'b0000; l_1 = 1'b0; l_3 = 1'b0;
        l_2 = 1'b0; l_4 = 1'b0;
        l_5 = 1'b0; l_6 = 1'b0;
        l_7 = 1'b0; end
    endcase
    end
    Buzzer b(clk,mode,note,speaker);
endmodule```
```

使用组合逻辑，通过输入改变note，调用蜂鸣器。

3.1.2 自动演奏模式

在010状态下，学习机可以自动演奏歌曲，进入自动演奏模式后，在开关状态为10010状态下，自动演奏“小星星”。

```
include "header.vh"
module happyTry(
input clk,//clk
input rst_n,//rst
input wire[4:0] mode,//song mode 4
output reg music = 0, //speaker
output [3:0] tub_sel1, //signal
output tub_sel2,//signal
output wire [7:0]tub_sel1_ctrl,
output wire [7:0]tub_sel2_ctrl,
output [6:0] led //led 7
);```
```

自动演奏歌曲的原理如下：

count1表示一个音符内的持续时间，freq表示当前的状态，若freq大于count1则此个音符演奏完毕。ip的长度为一个间隔的长度加一个音符的长度，所以当count2大于ip时意味着该演奏下一个音符了，此时index切换到下一个音符，以此类推。

```
if(count1 >= freq) begin
       //move to the next
        count1 = 0;
        //turn over the signal
        music = ~music;
        end
else count1 = count1 + 1;//after a clock cycle

if(count2 <= `gap) begin
    judge = 1;// turm to period
end

if(`gap < count2 && count2 <= ip) begin
    judge = 0; //turn to music
end

if(count2 > ip) begin
    count2 = 0;
    index = index + 1; //turn yo next mu

    if(index > curr) begin
        index = 0; //clearing 0
    end
end
count2 = count2 + 1; //count2++
end
```

将歌曲的谱子存在Melody数组中，每个音符从后往前读，每次读取5个bit，真正有实际意义的后4个bit。每个音符皆为5个bit，例：so的音符的二进制形式为01100，do的音符的二进制形式为01000，依次向上递增即可。

```
    always @* begin
    if (judge)
        freq = silence;
    else begin
        case (melody[index * 5 + 4 -: 5])//from hou to qian every five number
            5'd0 : begin
                freq = silence;
                light = `led0;
            end
            5'd8 : begin
                freq = `do;
                light = `led1;
```

3.1.3 学习模式

六个key按键对应六个灯，当对应灯亮起时，需要推动对应的按键，才能进行下一步的学习

```
module learning(
    input           clk,                    // 时钟
    input     [6:0] key,                    // 0: do, 1: re, 2: mi, 3: fa, 4:
sol, 5: la, 6: si
    input     [2:0] mode,
    output          speaker,                // 扬声器
    output reg [6:0] led,                    // LED
    output reg       finished,               // 歌曲是否播放完成
    output reg [40:0] score                  // 分数
);
```

采用对音符进行计数的方式决定下一个应该学习的音符

```
reg  [10:0] cnt_note;
reg [10:0] note_num;                    // 歌曲中音符的计数
reg  [6:0]  current;                     // 当前音符
```

每次弹对一个音符，cnt_note进行加1的操作

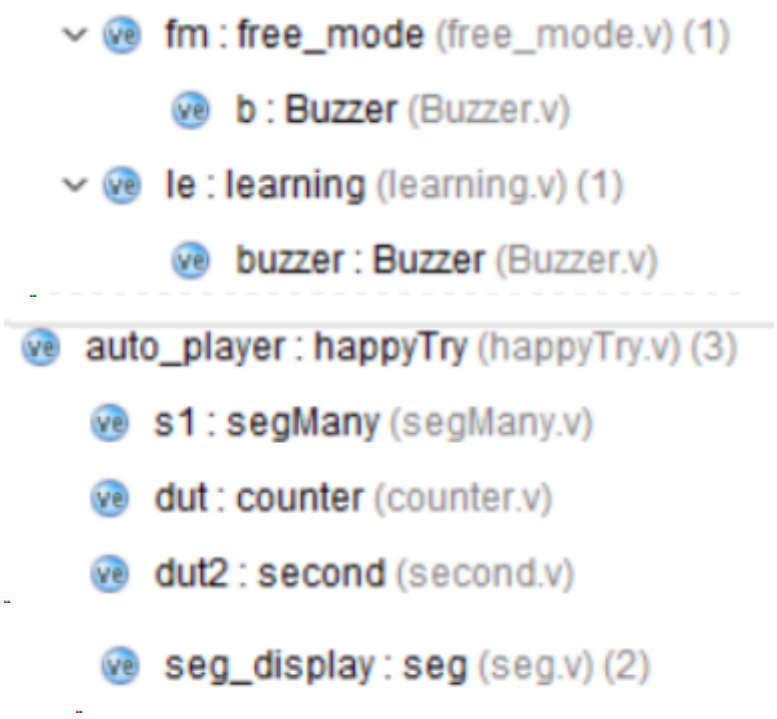
```
  if (cnt_note == note_num) begin
      finished <= 1;
      cnt_note <= 1;
  end else cnt_note <= cnt_note + 1;
```

```
  always @(cnt_note) begin
        case (cnt_note)
            6'd1:  begin current = `led5; led = `led5; note_len = 1; end
            6'd2:  begin current = `led3; led = `led3; note_len = 1; end
            6'd3:  begin current = `led5; led = `led5; note_len = 1; end
            6'd4:  begin current = `led3; led = `led3; note_len = 1; end
            6'd5:  begin current = `led5; led = `led5; note_len = 1; end
```

通过一个current来记录下一个应该弹什么

3.1.4 模块化设计

每一个模式都有属于自己的子模块，但共用buzzer



同时在顶层调用七段数码管的显示

```
seg seg_display (.clk(clk),
                 .rst_n(rst_n),
                 .p0(p0), .p1(p1), .p2(p2), .p3(p3),
                 .seg_en(temp3),.seg_out0(tub_sel1_ctrl_reg2),
.seg_out1(tub_sel1_ctrl_reg4));
```

3.2 标准功能

3.2.1 自由模式-高低八度

Buzzer：通过模式按钮输入确定演奏音区，可以实现高低八度

```
`include "header.vh"
module Buzzer(
input wire clk,
input wire [1:0] mode,
input wire [3:0] note,
output wire speaker
    );
    reg [31:0] notes [7:0];
    reg [31:0] counter;
    reg pwm;
    always@* begin
    case(mode)
    2'b00:begin
     notes[1] = 381680;
     notes[2] = 340136;
     notes[3] = 303030;
     notes[4] = 285714;
     notes[5] = 255102;
     notes[6] = 227273;
     notes[7] = 202429;
     end
     2'b01:begin
```

```
        notes[1] = `high_do;
        notes[2] = `high_re;
        notes[3] = `high_mi;
        notes[4] = `high_fa;
        notes[5] = `high_so;
        notes[6] = `high_la;
        notes[7] = `high_si;
      end
      2'b10:begin
        notes[1] = `low_do;
        notes[2] = `low_re;
        notes[3] = `low_mi;
        notes[4] = `low_fa;
        notes[5] = `low_so;
        notes[6] = `low_la;
        notes[7] = `low_si;
      end
      default:;
      endcase
      end
    initial
    begin
         pwm = 0;
    end
    always @(posedge clk) begin
        if(counter < notes[note] || note == 1'b0)
        begin
            counter <= counter + 1'b1;
        end
        else begin
             pwm = ~pwm;
             counter <= 0;
        end
    end
    assign speaker = pwm;
endmodule
```

3.2.2 自动演奏模式

学习机在增加音乐库的基础上自动演奏歌曲。

\1. 在七数码管上显示曲目编号：在showGe模块里可以根据选择的mode分配当前的曲目编号（在曲目标题之后此模块暂时不使用，实例化后即可实现）

```
`include "header.vh"
module showGe(// to show the num of song >useless uu
input [4:0] song,
output seg_out,
output reg [7:0] seg_ctrl);
    assign seg_out = 1'b1;
    always@* begin
    case(song)
      5'b10001: seg_ctrl = `sed1;
      5'b10010: seg_ctrl = `sed2;
      5'b10100: seg_ctrl = `sed3;
      5'b11000: seg_ctrl = `sed4;
```

```
      default: seg_ctrl = `sed0;
    endcase
    end

endmodule
```

\2. 显示歌曲名称：

在segMany模块里可以根据选择的mode分配当前的曲目的歌曲名称：

10001 Ring 铃儿响叮当

10010 Song 欢乐颂

10100 Star 小星星

11000 daub 粉刷匠

为了使四个字母同时亮起，需要写一个400Hz的新时钟，在always模块里实现四个字母分别亮起，只是无限缩短时间，使人眼无法分辨。

```
module segMany(
    input  clk, rst_n,  // clock, reset
    input [4:0] song, // input the song
    output reg [3:0] selLeft,   // pianxuan
    output reg [7:0] seg_ctrl  // 7-segment
    );
reg clk2; // creat a new clock
reg [31:0] count;// Counter for the 400Hz clock
reg [1:0] counts;// quicker signal
parameter circle = 250000;       //  the 400Hz clock

     always @(*) begin
    case (counts)//for every circle
        2'b00: begin
        selLeft = 4'b1000; //1
        case(song)
        5'b10001:seg_ctrl = `sedR;//R
        5'b10010:seg_ctrl = `sedS;//S
        5'b10100:seg_ctrl = `sedS;//S
        5'b11000:seg_ctrl = `sedd;//d
        default:seg_ctrl = `sed0;//0
        endcase
         end
        2'b01: begin
        selLeft = 4'b0100; //2
         case(song)
                    5'b10001:seg_ctrl = `sedi;//i
                    5'b10010:seg_ctrl = `sedo;//o
                    5'b10100:seg_ctrl = `sedt;//t
                    5'b11000:seg_ctrl = `seda;//a
                    default: seg_ctrl = `sed0; //0
                    endcase
       end
        2'b10: begin
        selLeft = 4'b0010; //3
        case(song)
```

```
                    5'b10001:seg_ctrl = `sedn;//n
                    5'b10010:seg_ctrl = `sedn;//n
                    5'b10100:seg_ctrl = `seda;//a
                    5'b11000:seg_ctrl = `sedu;//u
                    default:seg_ctrl = `sed0;//0
                    endcase
        end
        2'b11: begin
        selLeft= 4'b0001; //4
         case(song)
                    5'b10001:seg_ctrl = `sedg; //g
                    5'b10010:seg_ctrl = `sedg;//g
                    5'b10100:seg_ctrl = `sedr;//r
                    5'b11000:seg_ctrl = `sedb;//b
                    default:seg_ctrl = `sed0;//0
                    endcase
         end
        default: selLeft = 4'b0000;//nothing
    endcase
end
// Generate the 400Hz clock
//quickly just eyes disappear

always @(posedge clk, negedge rst_n) begin
    if (!rst_n) begin
        clk2 <= 0;
        count <= 0;
        //restart
    end
    else begin
        if (count == (circle >> 1) - 1) begin
        //over turn the clock
        //count zero clearing
            clk2 <= ~clk2;
            count <= 32'd0;
        end else begin
            count <= count + 1;//count++
        end
    end
end

// Generate the scan signal
always @(posedge clk2, negedge rst_n) begin
    if (!rst_n) begin
        counts <= 0;
         //counts zero clearing
    end else begin
        if (counts == 2'd3) begin
            counts <= 0;
            // over the circle 00 01 10 11
        end else begin
            counts <= counts + 1;//counts++
        end
    end
end
endmodule
```

\3. 进入自动模式后，按下按钮实现音乐库的切换，可以实现四首歌曲的切换：

在自动演奏时，使用灯光指示用户演奏位置和持续时间：

按下按钮确认曲目后，学习机开始自动演奏，当音符出现时从音符上方点亮灯光，音符演奏后熄灭灯光，直到歌曲结束。

3.2.3 学习模式

1.确定演奏的时间间隔

新建一个名为duration的寄存器用于存储音符持续时间,同时定义音符周期以及音符的持续时间。

```
reg  [28:0] duration;                    // 持续时间计数器
parameter   period   = `CLK;             // 音符周期
parameter   note_dur = 11 * period / 10; // 音符持续时间
```

在歌谱中对时长进行存储

```
 always @(cnt_note) begin
        case (cnt_note)
            6'd1:  begin current = `led5; led = `led5; note_len = 1; end
            6'd2:  begin current = `led3; led = `led3; note_len = 1; end
            6'd3:  begin current = `led5; led = `led5; note_len = 1; end
            6'd4:  begin current = `led3; led = `led3; note_len = 1; end
            6'd5:  begin current = `led5; led = `led5; note_len = 1; end
            6'd6:  begin current = `led3; led = `led3; note_len = 1; end
```

只有时长达到了要求的时间长短，才会变到下一个音符，否则维持在原始音符

```
 if (duration == note_dur / note_len) begin
            if (cnt_note == note_num) begin
                finished <= 1;
                cnt_note <= 1;
            end else cnt_note <= cnt_note + 1;//next note
            duration <= 0;
            end else if (key == current) begin
            duration <= duration + 1;//remain the same note, let the duration
increase
```

2. 增加用户评级

采用时间递增的方法对分数进行计算，如果单位时间没能完成乐曲，则评级会发生相应变化

首先在score这个module中确定分数

```
module score(
input [40:0]  score,
output reg [4:0] p0,p1);
    always @(score) begin

      if(score > 40'd0)begin
   {p1,p0} = `Go;  // Good
    end
      if(score > 40'd6_000_000_000)begin
    {p1,p0} = `no;  // normal
     end
```

```
        if(score >= 40'd10_000_000_000) begin
          {p1,p0} = `ba;  // Bad
        end
end
endmodule
```

在learning模块中，让score随着时间进行增加

```
if (finished == 1) begin
     score <= score;
     end
else score <= score + 1;
```

3. 增加用户选择

   在top模块新建一个user的输入，同时建立一个record数组

```
input [1:0]user;
reg  [40:0] record [3:0]; // record of the user
```

实例化传入参数的时候采用传入record，让七段数码管的显示可以随时变换

```
score sc(.score(record[user]),.p0(p4),.p1(p5));
```

让record作为参数，记入每一个user的分数,七段数码管就可以根据不同的用户来更改评级的显示

```
p2= {3'b000, user};
record[user]=score1;
```

3.3 附加创意（Bonus）

3.3.1 按键调整

```
module modify(
input en,
input clk,rst,
input l1,l2,l3,l4,l5,l6,l7,
input pick,
output speaker,
output reg [3:0] note,
output write
    );
    reg [1:0] mode;
    wire clk_div;
    reg write_en;
    reg pick_prev; //
    reg [3:0] block_m [6:0]; // use a reg array to save input
    reg [3:0] cnt,cnt_next; //the note currently play
    parameter S1 = 4'b0001,S2 = 4'b010,S3 = 4'b0011,S4 = 4'b0100,S5 = 4'b0101,
    S6 = 4'b0110,S7 = 4'b0111,S8=4'b1000,S0 = 4'b0000;
```

```
counter c(clk,rst,clk_div);//use a clk with lower frequency to modify the pick
button
always@(posedge clk_div, negedge rst) begin
if(~rst) begin
cnt <= S1;
write_en <= 1'b1;
mode <= 2'b00;
end
else begin
cnt <= cnt_next;
end
end
//tag bit,if the 7 notes are picked or not, if all picked, write_en = 0;
always@(posedge clk_div) begin
if(cnt == S8) write_en <= 1'b0;
end
```

每个clk_div的上升沿，cnt到下个状态

```
always@(cnt,pick) begin//use fsa to control cnt
    case(cnt)
    S1: if(pick) cnt_next = S2; else cnt_next = S1;
    S2: if(pick) cnt_next = S3; else cnt_next = S2;
    S3: if(pick) cnt_next = S4; else cnt_next = S3;
    S4: if(pick) cnt_next = S5; else cnt_next = S4;
    S5: if(pick) cnt_next = S6; else cnt_next = S5;
    S6: if(pick) cnt_next = S7; else cnt_next = S6;
    S7: if(pick) cnt_next = S8; else cnt_next = S7;
    S8: if(pick) cnt_next = S8; else cnt_next = S8;//when all 7 notes are
picked,cnt stay at state8 to play the notes
    endcase
    end
```

状态变化

```
always@(cnt,{l1,l2,l3,l4,l5,l6,l7},pick) begin
      if(cnt == S8) begin
         case({l1,l2,l3,l4,l5,l6,l7})
        `l1 : note = block_m[0];
        `l2 : note = block_m[1];
        `l3 : note = block_m[2];
        `l4 : note = block_m[3];
        `l5 : note = block_m[4];
        `l6 : note = block_m[5];
        `l7 : note = block_m[6];
         endcase
         end
         else begin
          note = cnt;
          if(pick == 1'b1) begin
          case({l1,l2,l3,l4,l5,l6,l7})
           `l1: block_m[0] = cnt - 1'b1;
           `l2: block_m[1] = cnt - 1'b1;
           `l3: block_m[2] = cnt - 1'b1;
           `l4: block_m[3] = cnt - 1'b1;
           `l5: block_m[4] = cnt - 1'b1;
```

```
            `16: block_m[5] = cnt - 1'b1;
            `17: block_m[6] = cnt - 1'b1;
          endcase
          end
          end
          end
    assign write = write_en;
    Buzzer b (clk,mode,note,speaker);
endmodule
```

通过改变note和开关的对应关系实现换状态。

3.3.2显示演奏时间-秒计时器

有一个秒计时器来检测当前的演奏时间，按rst可归零。

```
always @(posedge clk or negedge rst_n) begin
        if(!rst_n)begin
        seg_ctrl <= `sed0;
        end
        else begin
        if (clk_bps)
        //use new clock to update every second
            case (seg_ctrl)
                `sed0: seg_ctrl <= `sed1;
                `sed1: seg_ctrl <= `sed2;
                `sed2: seg_ctrl <= `sed3;
                `sed3: seg_ctrl <= `sed4;
                `sed4: seg_ctrl <= `sed5;
                `sed5: seg_ctrl <= `sed6;
                `sed6: seg_ctrl <= `sed7;
                `sed7: seg_ctrl <= `sed8;
                `sed8: seg_ctrl <= `sed9;
                `sed9: seg_ctrl <= `sed0;
            endcase
end
```

```
 if(!rst_n)
    cnt_first <= 14'd0;
    else if(cnt_first == 14'd10000)
    cnt_first <= 14'd0;
    else
    cnt_first <= cnt_first + 1'b1;
always@(posedge clk, negedge rst_n)
    if(!rst_n)
    cnt_second <= 14'd0;
    else if(cnt_second == 14'd10000)
    cnt_second <= 14'd0;
    else if(cnt_first == 14'd10000)
         cnt_second <= cnt_second + 1'b1;
    else
    cnt_second <= cnt_second;

    assign clk_bps =  cnt_second == 14'd10000;
```

构造一个状态机根据改变频率的clk_bps时钟来改变每一个时间的状态，以达成显示演奏时间的效果，每更新曲目，rst归0重新开始。

3.3.3 学霸模式-更灵活的评分

为学习模式进行难度上的提高，如果弹错或者在单位时间内没有完成弹奏，则全部重新开始

采用一个一秒的慢时钟，每一秒进行一次检测，检测玩家是否进行正确演奏

```
reg [31:0] slow_clk_counter;
parameter SLOW_CLK_PERIOD = 100_000_000;

slow_clk_counter <= slow_clk_counter + 1;
if (slow_clk_counter == SLOW_CLK_PERIOD) begin
// Reset the counter
slow_clk_counter <= 0;
```

增加一个right的output来告诉用户是否在单位时间内演奏正确

```
output reg right
```

评分上也将更加细致，如果弹错一个音符，就会进行罚分

```
if (key == current ) begin
   cnt_note <= cnt_note + 1;
   right<=1;
   score<=score;
end else begin
   cnt_note <=1;//restart
   right<=0;
   score<=score+40'd1_000_000_000;//punishment
end
```

4 系统使用说明

4.1 系统的输入和输出端口说明

clk 系统时钟 P17

```
set_property PACKAGE_PIN P17 [get_ports clk]
```

Led 7个led灯 分别为F6 G4 G3 J4 H4 J3 J2

```
set_property PACKAGE_PIN J2 [get_ports {led[0]}]

set_property PACKAGE_PIN J3 [get_ports {led[1]}]

set_property PACKAGE_PIN H4 [get_ports {led[2]}]

set_property PACKAGE_PIN J4 [get_ports {led[3]}]

set_property PACKAGE_PIN G3 [get_ports {led[4]}]

set_property PACKAGE_PIN G4 [get_ports {led[5]}]

set_property PACKAGE_PIN F6 [get_ports {led[6]}]
```

rst_n 复位按钮 P15

```
set_property PACKAGE_PIN P15 [get_ports rst_n]
```

change8 高低八度 U3为低八度 U2为高八度

```
set_property PACKAGE_PIN U3 [get_ports {change8[1]}]

set_property PACKAGE_PIN U2 [get_ports {change8[0]}]
```

modeAll 模式调整 T5为自由模式 T3为自动演奏模式 R3为学习模式

```
set_property PACKAGE_PIN T5 [get_ports {modeAll[0]}]

set_property PACKAGE_PIN T3 [get_ports {modeAll[1]}]

set_property PACKAGE_PIN R3 [get_ports {modeAll[2]}]
```

speaker 音乐 H17 T1

```
set_property PACKAGE_PIN H17 [get_ports speaker]

set_property PACKAGE_PIN T1 [get_ports pwm_ctrl]
```

do re me fa so la si 分别对应N4 M4 R2 P2 P3 P4 P5

```
set_property PACKAGE_PIN N4 [get_ports do]

set_property PACKAGE_PIN M4 [get_ports re]

set_property PACKAGE_PIN R2 [get_ports me]

set_property PACKAGE_PIN P2 [get_ports fa]

set_property PACKAGE_PIN P3 [get_ports so]

set_property PACKAGE_PIN P4 [get_ports la]

set_property PACKAGE_PIN P5 [get_ports si]
```

```
set_property PACKAGE_PIN R1 [get_ports kong]
```

tub_sel1_ctrl 左边的四个七段数码管

```
set_property PACKAGE_PIN B1 [get_ports {tub_sel1_ctrl[4]}]

set_property PACKAGE_PIN A3 [get_ports {tub_sel1_ctrl[5]}]

set_property PACKAGE_PIN A4 [get_ports {tub_sel1_ctrl[6]}]

set_property PACKAGE_PIN B4 [get_ports {tub_sel1_ctrl[7]}]

set_property PACKAGE_PIN A1 [get_ports {tub_sel1_ctrl[3]}]

set_property PACKAGE_PIN B3 [get_ports {tub_sel1_ctrl[2]}]

set_property PACKAGE_PIN B2 [get_ports {tub_sel1_ctrl[1]}]

set_property PACKAGE_PIN D5 [get_ports {tub_sel1_ctrl[0]}]
```

tub_sel2_ctrl右边的四个七段数码管

```
set_property PACKAGE_PIN D4 [get_ports {tub_sel2_ctrl[7]}]

set_property PACKAGE_PIN E3 [get_ports {tub_sel2_ctrl[6]}]

set_property PACKAGE_PIN D3 [get_ports {tub_sel2_ctrl[5]}]

set_property PACKAGE_PIN F4 [get_ports {tub_sel2_ctrl[4]}]

set_property PACKAGE_PIN F3 [get_ports {tub_sel2_ctrl[3]}]

set_property PACKAGE_PIN E2 [get_ports {tub_sel2_ctrl[2]}]

set_property PACKAGE_PIN D2 [get_ports {tub_sel2_ctrl[1]}]

set_property PACKAGE_PIN H2 [get_ports {tub_sel2_ctrl[0]}]
```

tub_sel2控制最右

```
set_property PACKAGE_PIN G6 [get_ports tub_sel2]
```

tub_sel1控制最左

```
set_property PACKAGE_PIN G2 [get_ports {tub_sel1[3]}]

set_property PACKAGE_PIN C2 [get_ports {tub_sel1[2]}]

set_property PACKAGE_PIN C1 [get_ports {tub_sel1[1]}]

set_property PACKAGE_PIN H1 [get_ports {tub_sel1[0]}]
```

finish 学习结束

```
set_property PACKAGE_PIN J5 [get_ports finish]
```

user 用户V5 V2

set_property PACKAGE_PIN V2 [get_ports {user[1]}]

set_property PACKAGE_PIN V5 [get_ports {user[0]}]

finish1 学习结束1

set_property PACKAGE_PIN K1 [get_ports finish1]

pick 切换调节音符R17

set_property PACKAGE_PIN R17 [get_ports pick]

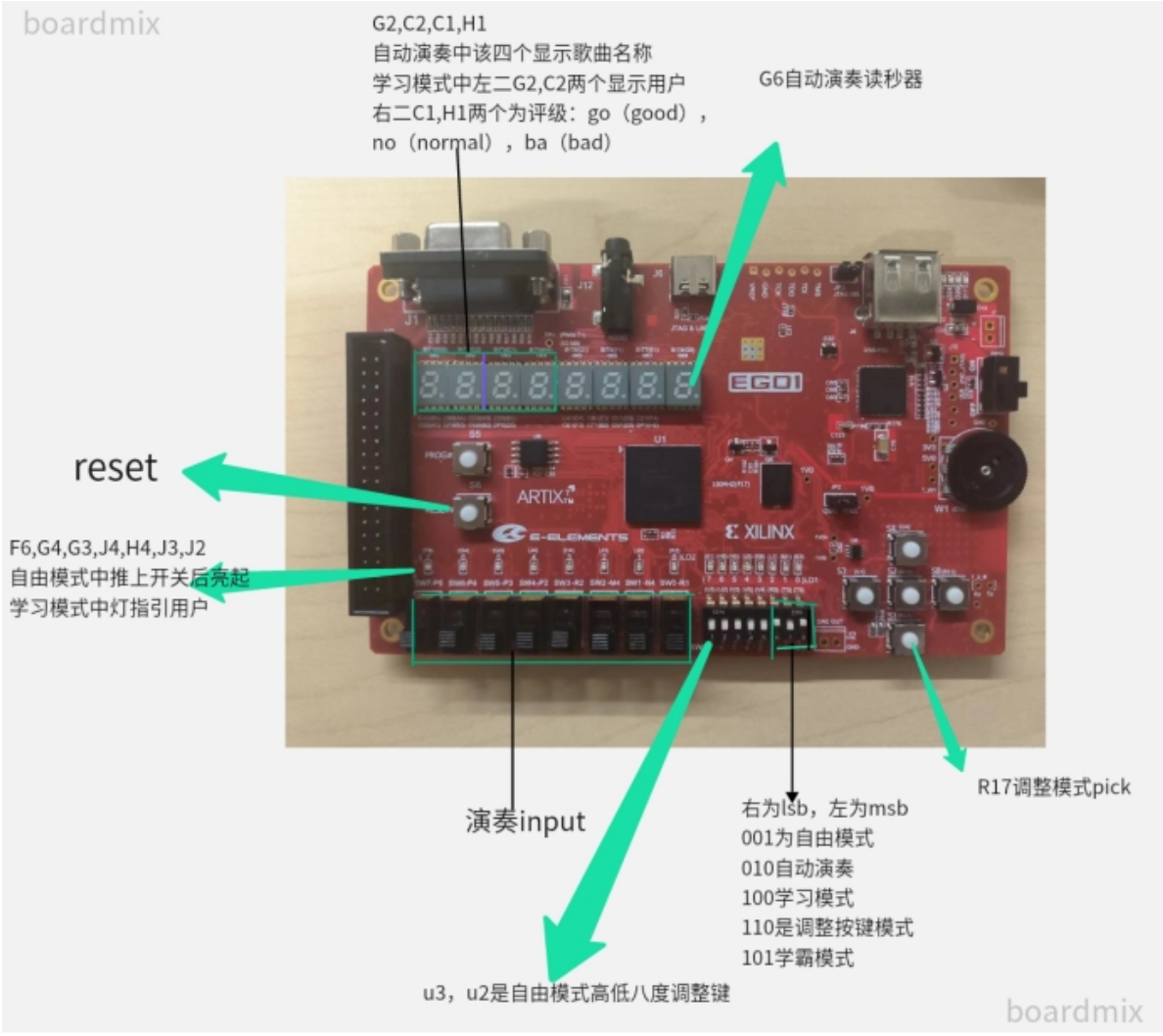
en enable V1

set_property PACKAGE_PIN V1 [get_ports en]

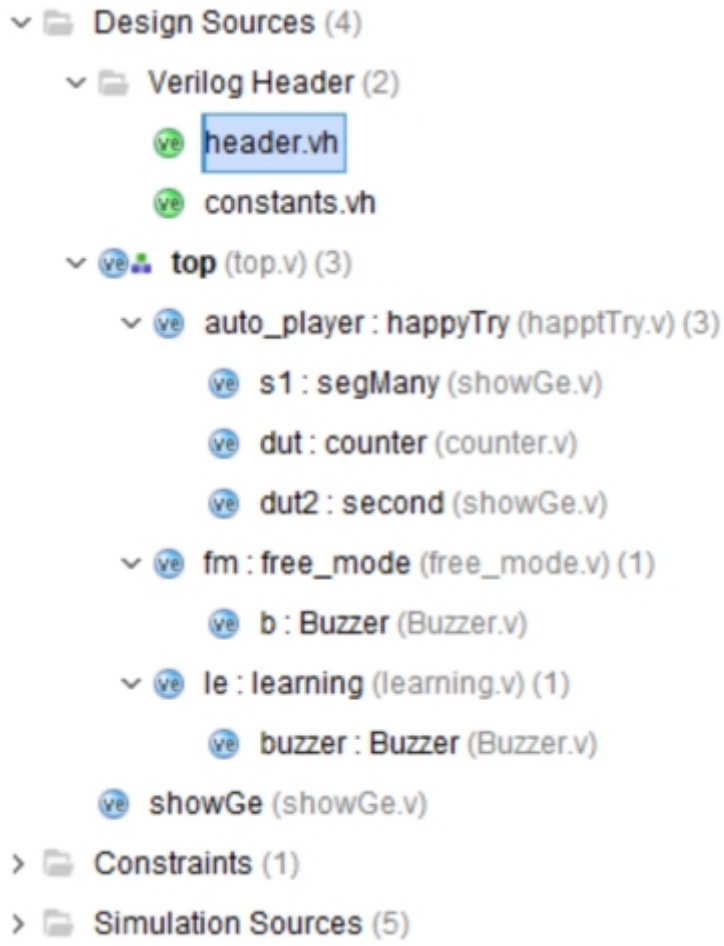
right 学霸模式演奏正确性L1

set_property PACKAGE_PIN L1 [get_ports right]

4.2 IO端口描述演示



5.1 内部模块之间的关系以及模块之间的信号线

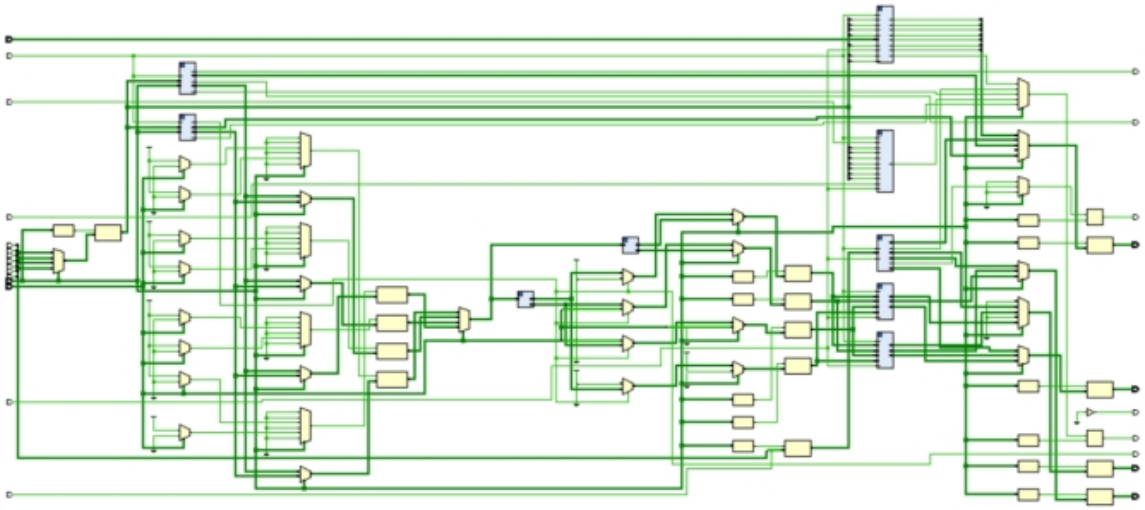


在top模块里，分别实例化三个功能模块，free_mode为自由模式，happyTry为自动演奏模式，learning为学习模式。

在happyTry里实例化三个模块，分别为segMany七段数码管、counter来实现秒时钟，second实现读秒的数码管。在learning和free_mode中实例化buzzer。

```
happyTry auto_player (
.clk(clk),.rst_n(rst_n),
.mode(mode),
.music(speaker2),
.led(led2),
.tub_sel1(temp1),
.tub_sel2(temp2),
.tub_sel1_ctrl(tub_sel1_ctrl),
.tub_sel2_ctrl(tub_sel2_ctrl)
);
free_mode fm(
      .do(key[0]), .re(key[1]), .mi(key[2]),.fa(key[3]),
      .so(key[4]),.la(key[5]), .si(key[6]),
      .clk(clk), .rst_n(rst_n),
      .mode(change8),
      .speaker(speaker1),
      .l_1(led1[6]),.l_2(led1[5]),.l_3(led1[4]),.l_4(led1[3]),.l_5(led1[2]),.l_6(led1[1]),.l_7(led1[0])
            );
learning le(.clk(clk),.key(key),.speaker(speaker3),
    .led(led3),.finished(finish), .score(score));
```

5.2 输入和输出端口描述及子模块功能描述



6 bonus实施说明

具体原理及代码见3.3附加创意（bonus）。

7 项目总结

7.1 团队合作

团队合作是项目成功的关键之一。通过三次小组讨论，我们明确了每个阶段的任务和责任，制定了合理的时间计划。在分工合作的过程中，团队成员充分发挥各自专业优势，共同解决遇到的问题。这种协作精神不仅提高了工作效率，也促进了团队成员之间的技能互补和共同成长。

7.2 开发和测试工作

在项目的开发和测试阶段，我们遇到了一些挑战，如连接不灵敏、系统文件不兼容、上板失败、multiple drivers等问题。然而，通过不断的测试和调整，我们成功克服了这些问题。在解决过程中，我们积累了更丰富的经验，学到了更多的调试技巧和优化方法。项目的整体完成时间相对较为集中，但在团队的共同努力下，取得了较为顺利的进展。

总体而言，我们的小组项目的成功完成得益于团队成员之间的协作和努力。在项目中遇到的各种问题都成为我们成长的机会，让我们更加熟悉了数字逻辑的开发和测试流程。通过这次项目，我们不仅取得了实际的成果，也积累了宝贵的团队协作经验，为今后的学习和工作奠定了基础。

8 基于EGO1的一些项目设置的建议

8.1 打地鼠

出于钢琴中学习模式的灵感，可以以相同原理开发“ 打地鼠 ”游戏，让灯随机亮起，推动开关实现打地鼠的操作，并进行记分和评级等。

8.2 计算器

可以实现一个有计算器功能的项目，通过推动开关来实现数字的读取，点击不同的按钮来实现加减乘除等操作，在七段数码管上显示结果。可以添加多个模式，比如答题模式：让玩家来计算，计算器需要实现判断玩家计算正确与否的操作，用led灯来显示。

8.3 打字机

可以实现一个打字机功能的键盘，七个开关分别代表七段数码管的七个信号，从而确定相应的字母，显示在七段数码管上，确认保存之后可以输入下一个字母，以此实现打字的效果，一个词语打字结束后可以进行存储，之后打下一个词语，在打n个词语之后可以进行循环地播放，具有实用性