

# 数字逻辑电子钢琴项目文档

小组成员：

姓名	负责工作	项目贡献度
白浚言	总体模块连接、学习模式、自动播放模式、蜂鸣器、数码管	33.3%
林易成	VGA显示模块、录音模式	33.3%
李容川	用户输入、自由模式、键位调整	33.3%
































## 开发计划日程安排和实施情况

日期	日程安排	实施情况
12.1 - 12.3	项目文档阅读，小组成员确定分工	按计划完成
12.4 - 12.11	完成基本功能模块，如 <b>蜂鸣器</b> ， <b>数码管</b> ， <b>用户输入</b> 等，方便后续开发使用	提前至12.7完成
12.12 - 12.15	完成 <b>自由模式</b> 和 <b>自动播放模式</b>	按计划完成
12.16 - 12.23	同步进行 <b>学习模式</b> 、 <b>VGA显示</b> 、 <b>键盘调整</b> 的开发	推迟至12.24完成
12.24 - 12.26	完成 <b>录音模式</b> ，连接所有模块并最终上板测试	按计划完成
12.27	项目 <b>答辩</b>	按计划完成
12.27 - 12.31	完成项目文档，录制说明视频	按计划完成

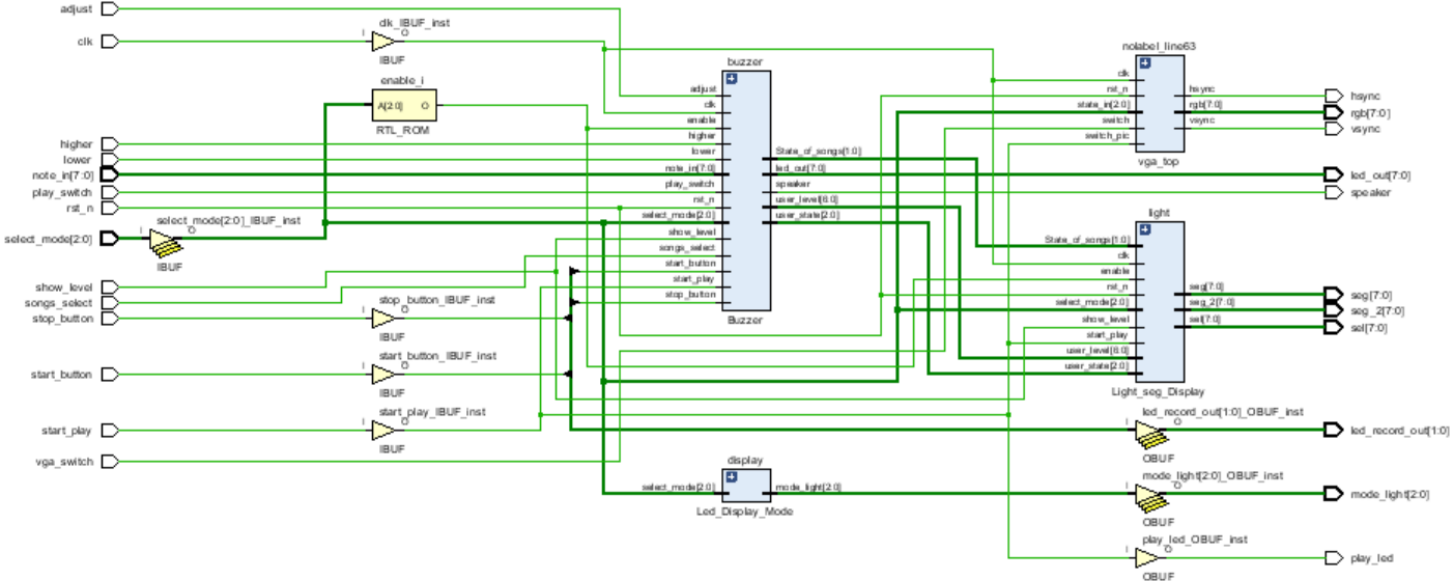
# 系统功能列表

系统功能列表	
基本框架	1. 能在自由模式、自动播放模式和学习模式之间切换
	2. 切换时有数码管、VGA 显示当前所在模式
自由模式	1. 能在按下键盘（开关）时播放出 7 个音符
	2. 能在按下升高音调以及降低音调键以后播放对应八度的音调
	3. 能够打开 VGA 显示中的自动下落乐谱进行歌曲小星星的练习
	4. 能够根据用户习惯，调整用户弹奏的键位位置
自动播放模式	1. 能够自动播放小星星、生日歌、HisTheme 三首歌曲，可以来回切换
	2. 播放时数码管上显示当前歌曲的名字
	3. 此模式下 VGA 显示一张包含所有歌曲名的曲谱图片
	4. 播放时可以调整播放速度，可以加快或减慢
学习模式	1. 有两个用户可以切换，拥有不同的弹奏评分
	2. 有演奏前的亮灯提示，提示用户应该按下哪个键，直到用户按对了播放下一个键
	3. 可以用 LED 灯进行用户演奏时的引导，引导速度可以改变
	4. 用户在弹奏时根据弹奏情况匹配情况有相应的评分（A ~ G），评分可实时更新
	5. 能记录用户弹奏的曲子，并可以播放出之前记录下的曲子。录音记录可以更新



- ▼  **MiniPiano** (MiniPiano.v) (4)
  - ▼  **buzzer : Buzzer** (Buzzer.v) (8)
    -  **u1 : Speed\_Control** (Speed\_Control.v)
    - ▼  **Key\_Adjustment : Key\_Adjustment** (Key\_Adjustment.v) (2)
      - ▼  **com\_pro : button** (button.v) (1)
        -  **clk\_div : clock\_div** (Key\_Adjustment.v)
        -  **k1 : Keyboard** (Keyboard.v)
    - ▼  **b1 : Record** (Record.v) (3)
      - >  **u\_ram1 : blk\_mem\_gen\_0** (blk\_mem\_gen\_0.xci)
      - >  **u\_ram2 : blk\_mem\_gen\_1** (blk\_mem\_gen\_1.xci)
      -  **k\_record : Keyboard** (Keyboard.v)
    - ▼  **u2 : Library** (Library.v) (2)
      - ▼  **b1 : button** (button.v) (1)
        -  **clk\_div : clock\_div** (Key\_Adjustment.v)
        -  **cd1 : clk\_div** (clk\_div.v)
      -  **lm1 : Learning\_Mode** (Learning\_Mode.v)
      -  **u3 : Frequency\_Divider** (Frequency\_Divider.v)
      -  **u4 : Wave\_Generator** (Wave\_Generator.v)
      -  **l1 : Led** (Led.v)
    - ▼  **light : Light\_seg\_Display** (Light\_seg\_Display.v) (1)
      -  **calculate : Level\_Calculate** (Level\_Calculate.v)
      -  **display : Led\_Display\_Mode** (Led\_Display\_Mode.v)
    - ▼  **vga\_top : vga\_top** (vga\_top.v) (3)
      - >  **u\_character : vga\_character** (vga\_character.v) (2)
      - >  **u\_game : vga\_music\_game** (vga\_music\_game.v) (2)
      - >  **u\_pic : vga\_pic\_littlestar** (vga\_pic\_littlestar.v) (3)
  - ▼  **Library** (Library.v) (2)
    - ▼  **b1 : button** (button.v) (1)
      -  **clk\_div : clock\_div** (Key\_Adjustment.v)
      -  **cd1 : clk\_div** (clk\_div.v)
    -  **Button\_Debounce** (Button\_Debounce.v)
  - >  **clk\_wiz\_0** (clk\_wiz\_0.xci)

RTL图



模块间信号线关系

顶层模块 MiniPiano 输入/输出信号:

```

input [7:0] note_in, //用户输入信号，对应开发板上八个开关
input higher, //调高一个八度
input lower, //调低一个八度
input clk, //内置时钟信号
input rst_n, //系统复位信号
input [2:0] select_mode, //选择钢琴模式：011自由模式，010自动播放模式，101学习模式，001练习模式
input start_play, //控制自由模式开始播放
input songs_select, //歌曲选择信号
input start_button, //录音模式开始
input stop_button, //录音结束信号
input play_switch, //播放录音信号
input show_level, //展示用户评分信号
input adjust, //进入按键调整模式的开关
//input commit, //按键调整时确定的按钮
input vga_switch, //切换vga状态
output speaker, //蜂鸣器输出
output [7:0] led_out, //8个开关上方灯光指引信号
output [7:0] sel, //位选信号
output [7:0] seg, //前四个段选信号
output [7:0] seg_2, //后四个数码管段选信号
output play_led, //灯光指引信号
output [2:0] mode_light,
output [1:0] led_record_out,
output wire hsync, //行信号
output wire vsync, //列信号
output wire [7:0] rgb //颜色信号

```

项目中的信号线主要分为四种：

- 数据信号：用户的输入、曲谱的输出等
- 控制信号：状态切换信号，如切换歌曲的信号等
- 时钟信号：全局时钟信号，用于时序逻辑的实现
- 状态信号：系统所处状态信号，如所处模式、选择的用户等

各模块对应的信号如下：

- MiniPiano：顶层模块，传入用户的输入信号，传出相应的数据信号。
- Buzzer：接受来自 MiniPiano 的蜂鸣器相关数据和控制信号，通过控制信号选择模式，来控制蜂鸣器的输出。
- Speed\_Control：歌曲播放速度的控制模块，接受全局时钟信号和模式的状态信号，当处于自动播放、学习模式需要播放内置歌曲的时候，开启计时器传出相应的时钟信号至 Library。

- Library : 接受 Buzzer 的状态信号和 Speed\_Control 的时钟信号, 根据当前模式输出相应的音符数据信号至 Frequency\_Divider 。
- Frequency\_Divider : 接受来自 Keyboard\_Adjustment 、 Library 和 Record 模块的音符数据信号, 通过 Buzzer 传入的状态信号根据当前所处的模式输出相应的分频器数据信号至 Wave\_Generator 。
- Wave\_Generator : 接受来自 Frequency\_Divider 的数据信号和 Buzzer 的时钟信号, 根据分频输出蜂鸣器的音符数据信号, 通过 Buzzer 传递至顶层模块使蜂鸣器发声。
- Key\_Adjustment : 接受来自 Buzzer 的用户输入信号, 传出对应的音符数据信号 (包括音阶) 至 Frequency\_Divider 进行选择。
- Keyboard : Keyboard\_Adjustment 的子模块, 通过处理将用户输入转化为音符数据信号传出至 Key\_Adjustment 。
- Learning\_Mode : 接受来自 Buzzer 的控制信号和状态信号, 当学习机处于学习模式时, 将音符输出信号连接至 Frequency\_Divider 。
- Led : 接受来自 Keyboard 的数据信号, 转化为灯光的输出传递至 Buzzer 。
- Light\_seg\_Display : 接受来自 MiniPiano 的状态信号和控制信号, 并在全局时钟的控制下输出8段数码管的段选信号和位选信号。
- button : 接受按钮的控制信号和全局的时钟信号, 获得一个较慢的时钟信号后生成新的消抖后的控制信号传递至 Key\_Adjustment 。
- Record : 接受来自 Buzzer 的控制信号和时钟信号, 以及来自 Key\_Adjustment 的用户输入信号, 传递出音符信号至 Frequency\_Divider 。
- VGA\_top : 接受来自 MiniPiano 的控制信号进行VGA显示信号的输出。

## 子模块功能说明

同名的输入输出端口的功能相同, 便不做赘述, 且主要对主要模块进行说明。

- Buzzer : **蜂鸣器**控制顶层模块, 通过子模块的实例化完成**自由模式、学习模式、播放模式和练习模式**的切换, 并连接至蜂鸣器输出。

```

module Buzzer (
    input [7:0] note_in, //用户的输入
    input higher,lower, //升调, 降调信号
    input clk, //全局时钟信号
    input rst_n, //复位信号
    input [2:0] select_mode, //选择钢琴模式: 011自由模式, 010自动播放模式, 101学习模式, 001
    input start_play, //控制自由模式和学习模式开始播放
    input songs_select, //歌曲选择和用户切换
    input enable, //全局控制信号, 当不处于四个模式时, 让蜂鸣器不能发声
    input start_button, //录音开始按钮
    input stop_button, //停止录音
    input play_switch, //录音模式的开关, 播放模式的开
    input show_level, //数码管显示用户当前的评分记录
    input adjust, //开始键盘调整模式
    output speaker, //蜂鸣器输出
    output [7:0] led_out, //指示灯信号
    output [1:0] State_of_songs, //正在播放哪一首歌曲
    output [2:0] user_state, //正处于哪一个用户
    output [6:0] user_level, //用户当前评级
    output [1:0] state //录音模式的状态: 00, 01, 10, 11分别对应均初始, 录音, 待播放, 播放
);

```

- Speed\_Control : 内置曲库播放的控制模块, 通过生成较慢的时钟控制 cnt 信号的自增, 不同的 cnt 指向 Library 中不同的音符, 进而完成对曲谱的“自动播放”。该模块还可以通过 higher 和 lower 的按钮进行播放速度的控制。

```

module Speed_Control (
    input clk,
    input rst_n,
    input [2:0] select_mode,
    input start_play, //开关信号
    input enable, //添加全局enable信号让时钟重置为0, 不让蜂鸣器播放
    input higher,
    input lower,
    output reg [6:0] cnt
);

```

- Library : 该模块内部存储了学习机可以自动播放的三首歌曲, 并有以下功能:
  - 通过 select\_mode 信号完成模式的切换。
  - 自动播放模式下, 通过输入信号 songs\_select 通过状态机进行歌曲的切换, 并将输出的信号 music 转化成小灯泡的控制信号完成对用户的指引。



- 学习模式下，通过接受来自 Keyboard 的用户输入信号与曲库内部存储的音符信号比对，控制播放的计数器，当用户按对音符后才能让计数器变化，完成学习模式的功能。
- 练习模式下，通过生成的新时钟完成对用户输入的定时采样，与曲库存储的音符对比完成评分的实时更新。
- 练习模式时，通过 songs\_select 进行用户的切换。

```
module Library (
    input clk,
    input rst_n,
    input [6:0] cnt, //存储曲子中的所有音符的计数器
    input [2:0] select_mode,
    input songs_select, //应该用一位实现时序逻辑,并利用按钮实现。
    input [6:0] keyboard_input, //键盘输入用于与曲库中的音符进行比较
    input [7:0] note, //键盘输入，用来检测用户输入是否有变化
    input start_play,
    input show_level,
    output reg [6:0] music, //存储各个音高的音符
    output reg [1:0] control_group,
    output reg [7:0] led_playmode,
    output reg [1:0] State_of_songs, //将歌曲状态输出，方便数码管判定是哪首歌
    output reg [6:0] user_level,
    output reg [2:0] user_state
);

parameter Little_Star = 2'd0, Happy_Birthday = 2'd1, His_Theme = 2'd2; //对应当前的歌曲
parameter User_0 = 3'd0, User_1 = 3'd1, User_2 = 3'd2;
reg [1:0] next_State_of_songs;
reg [6:0] cnt_learning_mode; //学习模式的cnt，实现用户输入匹配曲库内音符才能播放下一个音符的操作
reg [6:0] real_cnt; //切换后的计数器
reg [6:0] input_cnt; //记录用户输入的计数器
reg [6:0] input_music; //用户输入所对应的音符
reg [6:0] level [2:0]; //记录用户的评级
wire clk_out; //新的时钟
reg button_select;
reg [2:0] user_next_state;
wire songs_select_button_out;
```

- Frequency\_Divider：分频器模块，保证蜂鸣器能够输出对应高低的音符。
  - 能够根据模式选择对应模式的 music 的输出控制 divider 分频器。

```

module Frequency_Divider (
    input clk,
    input rst_n,
    input [2:0] select_mode,
    input [6:0] music_freemode, //自由模式的音符信号
    input [6:0] music_playmode, //自动播放模式的音符信号
    input [6:0] music_recordmode, //录音模式的音符信号
    output reg [31:0] divider, //分频器输出信号
    input [1:0] group,
    input [1:0] control_group,
    input [1:0] record_group,
    input [1:0] state //表示是否录音的状态
);
parameter S0 = 2'b00, S1 = 2'b01, S2 = 2'b10, S3 = 2'b11; // 均初始, 录音, 待播放, 播放,
reg [6:0] music_select;
reg [1:0] change_group;

```

- Wave\_Generator：音频输出模块，根据从分频器模块得到的频率输出对应的音符。

```

module Wave_Generator (
    input clk,
    input rst_n,
    input [31:0] divider,
    input [2:0] select_mode,
    output reg speaker
);
    reg [31:0] cnt;//内置计数器，控制蜂鸣器在一个时钟周期内响的时间，进而完成不同的音符输出。

```

- Key\_Adjustment：键盘包装模块
  - 键位调整状态：在键位调整进行至某音符时播放对应音符，并通过led提示用户设置按键，用户选定按键并按下“确认”按钮后开始下一个音符的键位调整。
  - 演奏状态：根据用户设置的键位(未设置或重置后则为初始键位，从左向右依次为C、D、E、F、G、A、B)，读取键盘状态，通过内置的 Keyboard 模块处理并输出目标音频。

```

module Key_Adjustment(
`include "Constants.vh"
input enable,
input [7:0] keys,
output reg [6:0] music,
input trigger,//开启这一模式
input rst,clk,
input commit, //确定当前键位对应的音符
input higher,
input lower,
output wire [1:0] group,
output wire [7:0] state_out,
//debug用
output wire [6:0] music_keyboard,
output commit_out
);
wire commit_pro;
assign commit_out = commit_pro;

//reg [7:0] notes;
//wire [6:0] music_keyboard;
reg ifFinish = 1'b0;
parameter
0 = `0,
C = `C,
D = `D,
E = `E,
F = `F,
G = `G,
A = `A,
B = `B; //对应音符
reg [7:0] convert [8'b11111111:0];
reg [7:0] state = 0,next_state = 0;

```

- Keyboard：初始键盘模块，输入为note模拟键盘状态数据、higher升调信号、lower降调信号；输出为note\_out音符数据、group八度数据。功能为根据传入的模拟键盘状态数据输出指定音符，同时可通过升/降调实现。

```

module Keyboard(
input [7:0] note,
input higher,
input lower,
input rst,
input clk,
input [2:0] select_mode,
input enable,//添加全局enable信号
output reg [6:0] note_out,
output reg [1:0] group
);
    reg [1:0] group_next;

```

- Light\_seg\_Display：数码管显示模块，实现数码管的动态显示。

```

module Light_seg_Display (
    input clk,
    input rst_n,
    input [2:0] select_mode,
    input [1:0] State_of_songs,//传入歌曲选择信号
    input start_play,
    input enable,
    input [6:0] user_level,//从Library模块中传来的评分信号，需要在该模块中转换成能够显示的字
    input [2:0] user_state,//当前的用户
    input show_level,
    output reg [7:0] sel, //位选信号，决定8段数码管哪根亮。
    output reg [7:0] seg,//段选信号，四根数码管的位选信号是公共的，只能同时显示。决定单根数码管
    output reg [7:0] seg_2
);

```

- Led：灯泡显示模块，根据音符对应灯泡控制。

```

module Led(
input clk,
input rst_n,
    input [7:0] led_in_playmode,//自由模式的灯泡信号
    input [7:0] led_in_learning_mode,//学习模式的灯泡信号
    input [7:0] led_keyboard_adjust,//键盘调整模式的灯泡信号
input [2:0] select_mode,
    output reg [7:0] led_out //最终灯泡输出
);
    reg [7:0] led;

```

- vga\_top : VGA顶层模块。

```
module vga_top(
input wire clk,
input wire rst_n,
input wire [2:0] state_in, //转换状态
input wire switch, //调整vga输出为下落方块
input wire switch_pic, //调整输出为图片
output reg hsync, //行信号
output reg vsync, //列信号
output reg [7:0] rgb
);
```

- vga\_character : VGA静态显示模块。

```
module vga_character(
input wire clk,
input wire rst_n,
input wire [2:0] input_state, //转换状态
output wire hsync,
output wire vsync,
output reg [7:0] rgb
);
wire clk_25M;
gen_clk25 u_clk25(
    // Clock in ports
    .clk(clk), // IN
    // Clock out ports
    .clk_25(clk_25M));
wire [9:0] hsync_cnt;
wire [9:0] vsync_cnt;

reg [2:0] state;
parameter S0 = 3'b011, S1 = 3'b010, S2 = 3'b101, S3 = 3'b001; //表示自由, 自动, 学
```

- vga\_driver : VGA驱动模块。

```

module vga_driver
#(
    //分辨率640*480@60Hz
    parameter H_CNT_BIT_WIDTH = 10'd10 , //行扫描计数器位宽
               H_SYNC          = 10'd96 , //行同步
               H_BACK          = 10'd40 , //行时序后沿
               H_LEFT          = 10'd8  , //行时序左边框
               H_VALID         = 10'd640 , //行有效数据
               H_RIGHT         = 10'd8  , //行时序右边框
               H_FRONT         = 10'd8  , //行时序前沿
               H_TOTAL         = 10'd800 , //行扫描周期

               V_CNT_BIT_WIDTH = 10'd10 , //场扫描计数器位宽
               V_SYNC          = 10'd2  , //场同步
               V_BACK          = 10'd25 , //场时序后沿
               V_TOP           = 10'd8  , //场时序上边框
               V_VALID         = 10'd480 , //场有效数据
               V_BOTTOM        = 10'd8  , //场时序下边框
               V_FRONT         = 10'd2  , //场时序前沿
               V_TOTAL         = 10'd525 ) //场扫描周期

(
    input wire clk, //分辨率640*480@60Hz对应于25MHz的时钟
    input wire rst_n,
    output reg hsync,
    output reg vsync,
    output reg [H_CNT_BIT_WIDTH-1'b1 : 0] hsync_cnt,
    output reg [V_CNT_BIT_WIDTH-1'b1 : 0] vsync_cnt
);

```

- game\_reg\_out : VGA动态显示模块。

```

module game_rgb_out(
input wire clk,    //分辨率640*480@60Hz对应于25MHz的时钟
input wire rst_n,
output wire hsync,
output wire vsync,
output reg [7:0] rgb
);
parameter H_CNT_BIT_WIDTH = 10'd10, V_CNT_BIT_WIDTH = 10'd10,
          H_TOTAL          = 10'd800,    //行扫描周期
          V_TOTAL          = 10'd525,
          h_length_8 = 10'd80, h_length_4 = 10'd160, h_length_2 = 10'd320,
          h_left = 10'd144, h_up = 10'd35, h_height_4 = 10'd120; //场扫描周期

reg frame_flag;                //帧结束标志
reg [H_CNT_BIT_WIDTH-1 : 0] x; //位置变量x,这里位宽和hsync_cnt计数器一致,是

reg [V_CNT_BIT_WIDTH-1 : 0] y; //位置变量y,这里位宽和vsync_cnt计数器一致,是

wire [H_CNT_BIT_WIDTH-1 : 0] hsync_cnt;
wire [V_CNT_BIT_WIDTH-1 : 0] vsync_cnt;

```

## Bonus实现说明

已实现Bonus：

- 键位调整
- VGA静态显示、动态显示、图片显示
- 评分实时更新
- 音乐节奏变化

## 键位调整

利用KeyAdjustment模块，用户可以进入键位调整模式，根据个人习惯调整设备的键位。

该模式下模块的主要输入为keys实际键盘输入数据、trigger模式开关、commit确认信号；主要输出为music音频信号。模块内包括8个状态，对应更改7个音符的习惯按键和更改完成状态。

在该模式下，设备会依次进入状态C、D、E、F、G、A、B（对应do、re、……、xi），并在每个状态下播放相应的音符。

```

always @(negedge rst,posedge commit_pro,posedge trigger)begin
    if(~rst&trigger)
        {ifFinish,next_state}<={1'b0,C};
    else if(commit_pro)
        case(state)
            C:next_state<=D;
            D:next_state<=E;
            E:next_state<=F;
            F:next_state<=G;
            G:next_state<=A;
            A:next_state<=B;
            B:{next_state,ifFinish}<={0,1'b1};
        endcase
    else next_state<=state;
end

```

用户可以根据自己的喜好和习惯，设置该音符的键位，然后按下确认键。

此时，用户的键位设置（记作x）会作为数组下标，存储当前状态对应的音符，同时设备会进入下一个状态。

```

always @(negedge rst,posedge commit_pro)begin
    if(~rst) {convert[0],convert[C],convert[D],convert[E],convert[F],convert[G],conver
    else if(commit_pro) convert[keys]<=state;
end

```

当所有音符的键位调整完成后，设备会进入状态O，此时用户可以通过调整后的按键设置来演奏对应的音符。

此时，键盘输入数据会作为数组下标，模块内置的Keyboard键盘演奏模块会读取相应数组元素，获取该键位对应的音符并进行演奏。

如果用户想要重置已保存的键位习惯，可以使用reset按钮来清空已有设置。

## VGA

VGA实现原理：

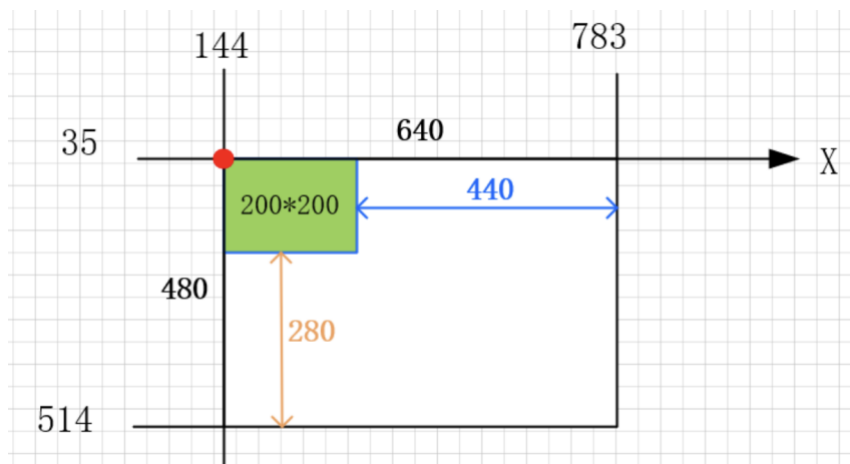
VGA（Video Graphics Array）是一种用于连接计算机和显示器的图形标准。计算机中通常有一个图形缓冲区，用于存储图像的像素数据。每个像素通常由颜色信息组成。计算机内有VGA控制器这个硬件组件，负责生成图像信号并将其发送到显示器。它与图形缓冲区交互，按照特定的时序生成图像。



显示器的水平和垂直扫描时需要VGA时序进行控制。扫描时先进行一行的扫描，再进行行回扫，再进行下一行的扫描。所有的行扫描完以后需要进行整个场的垂直回扫，在回扫的过程中需要进行消隐。控制扫描的信号包括前沿同步、后沿同步、水平同步、垂直同步等信号（由VGA接口传出），以确保显示器能够按照正确的方式扫描像素并显示图像。

在像素输出时，VGA控制器读取像素数据（由VGA接口传出），并将其转换成电信号输出。每个像素的颜色信息通过红、绿、蓝三个通道的电压表示。

在verilog代码中，vga\_driver 实现了VGA时序内的各种信号的变化，传递出的hsync\_cnt与vsync\_cnt信号代表当前扫描到的位置，再根据当前所在位置进行当前像素点rgb值的赋值。本项目采用的显示模式是640\*480@60，对应一个固定的时钟以及行时序场时序值。对应的图片如下：



在显示文字时，先用相应工具将汉字转换为像素点的储存格式，再通过行场信号赋值；在显示图片时则是将图片存储为一个coe文件，再存储入一个ROM中，通过data\_drive.v进行图片信息的读取，再输出为当前像素点的rgb值；对于下落滑块的实现，先根据每个音符再曲谱中的位置以及音调给每个滑块的初始位置赋值，在按下rst键后每扫描一帧滑块的像素位置会增加2，由于扫描的频率是60Hz，人的肉眼看到的滑块是连续移动的。

文字部分主要代码为显示像素点位置判断，大体如下：

```
if(hsync_cnt >= h_left - char_length_2 + h_length_2 && hsync_cnt <= h_left + char_length_2)
    char_bit <= 10'd63 - (hsync_cnt - (h_left - char_length_2 + h_length_2));
    case(vsync_cnt)
        h_height_4 + h_up: if(char1_line0[char_bit]) rgb <= 'b111_111_11; else rgb <= 'b000_000_00;
        h_height_4 + h_up + 9'd1: if(char1_line1[char_bit]) rgb <= 'b111_111_11; else rgb <= 'b000_000_00;
        h_height_4 + h_up + 9'd2: if(char1_line2[char_bit]) rgb <= 'b111_111_11; else rgb <= 'b000_000_00;
        h_height_4 + h_up + 9'd3: if(char1_line3[char_bit]) rgb <= 'b111_111_11; else rgb <= 'b000_000_00;
        //省略中间
        h_height_4 + h_up + 9'd15: if(char1_line0f[char_bit]) rgb <= 'b111_111_11; else rgb <= 'b000_000_00;
```

图片部分主要代码为显示ROM文件读取，大体如下：

```

always @(posedge clk_25 or negedge rst_n) begin
    if(~rst_n)begin
        rgb <= 11'b0;
    end
    else if ( flag_visible ) begin //判断是否为图片显示区域
        rgb <= rom_data;
    end
    else begin
        rgb <= 11'b0;
    end
end

assign flag_clear_address = (rom_address == length * width - 1); //是否读到最后一个地址
assign flag_visible_h = hsync >= ( (visible_length - length)/2 + 10'd144 ) && hsync <= (visible_length - length)/2 + 10'd144;
assign flag_visible_v = vsync >= ( (visible_width - width)/2 + 10'd35 ) && vsync <= (visible_width - width)/2 + 10'd35;
assign flag_visible = flag_visible_h & flag_visible_v;

```

图片移动的主要代码如下：

```

always @(posedge clk, negedge rst_n) begin
    if(~rst_n) begin
        // 以显示的底边 + 最长条块长度作为基准线
        for (i = 0; i < 8; i = i + 1) begin
            bar_bottom[i] <= bar_length*2*i + 16'd480 + 16'd240; //小星星中，每个音符间
        end
    end
    else begin
        if(frame_flag) begin //这个帧结束
            for (i = 0; i < 8; i = i + 1) begin
                if(bar_bottom[i] == 16'd0)
                    bar_bottom[i] <= 16'd0; //如果到底了则一直在底端
                else
                    bar_bottom[i] <= bar_bottom[i] - 2; //如果没到底下降两个像素
            end
        end
    end
end
end
end

```

## 评分实时更新

主要代码如下：

```

//使用新的时间周期每隔1/4秒检测用户的输入，如果输入与对应的曲谱音符匹配，就让评分增加。注意更新的是相应F
clk_div cd1 (.clk(clk),.rst_n(rst_n),.select_mode(select_mode),.start_play(start_pl
always @(posedge clk_out,negedge rst_n) begin
    if(~rst_n) begin level[user_state] <= 7'd0; end
    else if(keyboard_input == music) level[user_state] <= level[user_state] + 1'b1;
    else level[user_state] <= level[user_state];
end

//当按下show_level开关，展示相应用户的评分,将user_level传给display模块。
always @(posedge clk_out) begin
    if(show_level) user_level <= level[user_state];
    else user_level <= 7'd0;
end

```

其中 clk\_div 生成了相较于自动播放速度更慢的时钟，每次间隔1/4秒对用户的输入进行采样并于音符进行比较，如果相同则让评分增加，然后通过 Level\_Calculate 模块对用户的评级进行实时更新。进而传给数码管显示对应的评级。

```

module Level_Calculate(
input [6:0] user_level,
output reg [2:0] level
);
always @(*) begin
if(user_level >= 0 && user_level <= 20) level = 3'd0;
else if(user_level > 20 && user_level <= 30) level = 3'd1;
else if(user_level > 30 && user_level <= 40) level = 3'd2;
else if(user_level > 40 && user_level <= 60) level = 3'd3;
else if(user_level > 60 && user_level <= 70) level = 3'd4;
else if(user_level > 70 && user_level <= 90) level = 3'd5;
else level = 3'd6;
end
endmodule

```

## 音乐节奏调整

在 Speed\_Control 模块中实现，通过调整时钟分频器的时钟来调整对应音符指针 cnt 的改变速度，进而改变播放速度，主要是通过状态机进行 T\_final 的赋值，代码如下：

```

parameter T_1000ms = 32'd1_0000_0000, T_500ms = 32'd5000_0000, T_2000ms = 32'd2_0000_000
reg [31:0] T_final;
reg [31:0] T_final_next;
// parameter T_125ms = 1;
reg [31:0] count;
wire is_Reaching_125ms; //判断是否到达一个1/8的周期

//状态机, 切换自动播放速度
always @ (posedge clk, negedge rst_n) begin
    if(~rst_n) T_final <= T_1000ms;
    else T_final <= T_final_next;
end

always @(posedge clk, posedge higher, posedge lower) begin
    case(T_final)
        T_500ms: if(higher) T_final_next <= T_500ms; else if(lower) T_final_next <= T_1
        T_1000ms: if(higher) T_final_next <= T_500ms; else if(lower) T_final_next <= T_
        T_2000ms: if(higher) T_final_next <= T_1000ms; else if(lower) T_final_next <= T_

    endcase
end

```

## 项目总结

在整个项目的开发过程中，我们按照阶段性的计划和目标，有条不紊地进行了工作，取得了一系列阶段性的成果。

- 第一阶段，我们成功地认领任务，并通过任务分工迅速展开工作。首先完成了蜂鸣器相关模块的开发，同时，成功地将曲库整合到该模块中，实现了《小星星》的播放，初步完成自动演奏的模式的基本功能。随后转向键盘输入模块的开发，使用户可以通过键盘输入演奏音符，从而实现了自由模式的基本功能。其中加入升/降调功能，用户可以在三个八度内进行自由演奏。这一阶段的测试相对简单，我们能够直接在板上进行所有功能的测试，确保项目的初步可行性。-
- 第二阶段，我们着手进行各种子模块的开发，包括七段数码管、LED、VGA等功能。同时，我们开始综合子模块功能，初步完成了学习模式、键位调整、录音等综合性功能的开发。由于模块功能尚未合并，我们采用了testbench仿真和单独上板测试某一模块的方式，确保每个功能都能够独立正常运行。这一阶段的工作为项目的后续发展做准备。
- 第三阶段，我们进行了模块的合并和功能的完善。成功实现了学习模式中的账户与用户评级等综合性较强的功能，并进一步完善了不同模式的切换设置。这一阶段的测试主要集中在在板上装载总模块，分别测试不同模式的功能。这为我们确保整个系统的协调运行提供了充分的保障。

- 第四阶段，以对项目进行规范化为主。我们着重于对语法和数值常量的使用进行规范，以提高代码的可读性和维护性。通过规范化的工作，我们使项目更加规范、清晰，减少了潜在的错误，确保了项目的质量和可维护性。

## 新项目设计： 基于Fpga的"Sobel"边缘检测

### 项目任务

用FPGA实现边缘检测的算法（sobel），要求将一张自选图片经过边缘检测以后通过VGA接口显示在显示屏上

### 实现原理

1. Sobel算法：我们知道边缘点其实就是图像中灰度跳变剧烈的点，所以先计算梯度图像，然后将梯度图像中较亮的那一部分提取出来就是简单的边缘部分。Sobel算子用了一个3\*3的滤波器来对图像进行滤波从而得到梯度图像，这个算子一般给出一个定值。
2. Vga显示：要显示整个图像，一行扫描完成后要进行回扫，然后开始下一行的扫描。所有行扫描完成后，需要进行垂直回扫完成扫描过程。垂直扫描的周期长，完成整个屏幕的显示，也称为场。回扫的过程中会要行消隐，因此有一定的时间延迟。利用行扫描信号以及场扫描两个信号，可以完成在整个屏幕的扫描并且通过RGB属性输出得到当前位置的颜色。

### 基本功能

1. 能够显示原先的图片
2. 能够在拨动开关以后显示边缘检测以后的图片
3. 能够实现多种边缘检测效果，并且可以互相切换，同时在数码管上显示相关信息

### Bonus

1. 学习canny边缘检测并且进行比较
2. 研究方法，将检测效果变得更加精细化

### 结果示例

