CS207Project

12210401 龚浚哲　　12210403 王赵融杰
12210232 倪汇智

December 2023

1　任务分工与主要贡献

成员及贡献比：

成员	贡献比
龚浚哲	1/3
王赵融杰	1/3
倪汇智	1/3

具体分工：

- 整体框架设计和模块划分：王赵融杰 & 倪汇智 & 龚浚哲
- 基础自由模式, 高低八度设置：倪汇智
- 歌曲存储: 倪汇智
- 基础自由演奏模式：王赵融杰
- led 灯实现：倪汇智 & 龚浚哲 & 王赵融杰
- 七段数码管：倪汇智 & 龚浚哲 & 王赵融杰
- 学习模式实现：龚浚哲
- VGA 实现：王赵融杰

2 开发日志

12 周：进行项目选择与基本分工安排完成情况：每个人都根据自己的理解写出了基本的自由模式实现和模块划分。

13 周：根据分工完成每个人负责的基本模式和部分标准功能完成情况：所有人都完成了基本模式以及大部分标准功能，进行了第一次合代码和整体调试。

14 周：完成所有标准功能以及部分附加创意，开始报告撰写；完成情况：完成了所有标准功能，实现了附加创意的音符节奏变化，学习模式评分细化，自由模式存歌功能，基础 VGA。

15 周（答辩前）：检查代码规范性，合并所有代码。完成情况：实现了对所有代码的检查和调试，并完善了 bonus 部分的储存歌曲和 VGA 实现。

15 周（答辩后）：进行了视频的录制，完成报告。

3 系统功能列表

系统功能列表

模式	基础功能	标准功能	附加创意
自由模式	随意按键，播放相应音符	增加调准八度音符按键	实现将自由模式的录音长期存入乐库
自动演奏模式	自动演奏歌曲	切换歌曲 用灯光指示用户演奏位置和持续时间 按键实现乐库翻页 在七段数码管显示曲目名称及编号	实现播放自由模式的录音 每个音符的持续时间可以不一致 利用VGA显示歌曲名称，使用钢琴键动效提示音符播放的时机（四个模式都接入）
练习模式	根据顺序和时长亮灯，引导用户正确演奏	不评级，只供选手为竞赛练习	每个音符的持续时间可以不一致
竞赛模式		根据用户的演奏状态，在七段数码管上实现了用户演奏水平的评级（实时和最终结果均有） 为不同用户创建账户，并实现更新账户的演奏评级	根据用户按键的时间与标准时间的差异提供实时变化的评分 每个音符的持续时间可以不一致

图 1: 系统功能列表

4 系统使用说明

系统使用说明

系统的输入：

1 bit clk　连接开发板时钟信号

1 bit reset　连接 rst 按键

1 bit start　连接中间按键

1 bit user　连接中间小开关

1 bit write_on　连接最右边大开关

2 bit octave　连接左边三个小按键

2 bit song_select　连接上下两个按钮

2 bit speed_select　连接左右两个按钮

4 bit mode　连接右边三个小开关

7 bit keys　连接左边七个大开关

系统的输出：

1 bit speaker　连接音频输出

1 bit test　连接蜂鸣器使用

7 bit led　七个连接 led 灯

8 bit light seg　连接七段数码管

8 bit light_left　选择左边四个七段数码管

4 bit an　控制左边的七段数码管

4 bit an_right　控制最右边的七段数码管

1 bit hsync　水平同步

1 bit vsync　垂直同步

12 bit vga_rgb　控制 vga 的颜色

5　系统结构说明

系统结构说明

我们在主模块 MiniPiano 下实例化了以下四个子模块：

-Buzzer

-Controller

-Light_seg
-vga_colorbar

在 Controller 模块下实例化了以下四个模式子模块：
-Mode_free
-Mode_auto
-Mode_learn
-Mode_competition
在 Mode_auto 模块下实例化了 lib 模块。
在 Mode_learn 模块下实例化了 lib 模块。
在 Mode_competition 模块下实例化了 lib 模

6 子模块功能说明

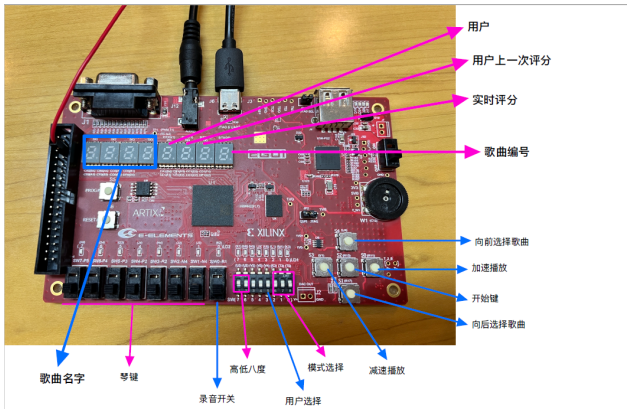

图 2: 输入输出图

Buzzer 模块：
输入输出如下： input wire clk, // Clock signal
input wire [3:0] note, // Note 代表要选的音
output wire speaker, // 输出的音频信号
input wire [2:0] mode, // 模式选择，根据不同模式 buzzer 需要做调整

input wire [1:0] octave, // Octave selection: 01 for lower, 10 for higher, else standard，高低八度，来自自由模式

input wire [1:0] octave_auto 高低八度，来自自动演奏模式或者学习模式

此模块负责接收需要发出的声音选择信号，并根据内部逻辑实现音频信号输出。

debounce 模块: 负责将按键进行消抖。

Light_seg:

input wire user,//显示用户的输入

input wire [3:0] score, 显示用户评分

input wire [3:0] score_user, 显示用户上次评分

input wire [3:0] num, number // 4-bit input representing the song

input wire clk, // Clock signal

input wire reset, // Reset signal

input wire [2:0] mode,

output reg [7:0] seg1, // show right score, speed, song

output reg [7:0] seg, //show name

output reg [3:0] an, // control the left seg

output reg [3:0] an_right 控制右边四个七段数码管

该模块接收一些需要在七段数码管上显示的参数输入，同时在内部通过利用人眼视觉残留，快速变化不同位置来实现每一个七段数码管都能显示。

mode_auto 模块:

input wire clk, // Clock signal

input wire [1:0] song_select, // choose song

input wire reset, 复位信号

output reg [3:0] note_to_play, // out_put to buzzer

output reg [6:0] led_out,led 灯输出

output reg [1:0] octave_auto , 曲库里的高低八度

output wire[3:0] num, 现在正在播放的歌曲，传入七段数码管

input wire [1:0] speed_select, 选择歌曲播放的速度

input wire play_state 用来判断是否开始播放/暂停

该模块为自动演奏模块，实现了自动演奏功能，从我们预先设置好的 lib 曲库里同时获得三个参数：歌曲音符，高低八度，每个音符持续时间。然后判断当 play_state 信号为高电平有效时进行歌曲的播放，同时，我们还实现了速度的选择，用户可以根据左右两个按钮实现对播放的歌曲速度的选择。

mode_competition 模块: input wire clk, // Clock signal

input wire [1:0] song_select, // Input to choose song

input wire [6:0] switches, // Input from 7 switches

input wire reset, // Reset signal

input wire [1:0] octave_competition, // Input to choose the proper octave

output reg [3:0] note_to_play, // Output to buzzer

output reg [6:0] led_out, // LED output

output reg [1:0] octave_out, // Output the octave value

output wire [3:0] num, // Number output

input wire [1:0] speed_select, // Input to select speed

output reg [3:0] score, // Output for the score

input wire play_state, // Play state signal

input wire user, // User input signal

output reg [3:0] score_A, // Score for player A

output reg [3:0] score_B // Score for player B

该模式为学习模式的扩展，歌曲会保持播放状态，评分会根据用户有没有在音符播放时间内按下按钮来进行。

mode_free 模块: input wire clk,

input wire reset,

input wire write_on,

input wire storeRecord,

input wire [6:0] keys,

input wire [1:0] octave,

input wire [3:0]selectSong,

output reg [6:0] led_out1,

output reg [6:0] led_out2,

output reg[3:0] note_to_play1,

output reg [1:0]octave_out1,

output reg[3:0] note_to_play2,

output reg [1:0]octave_out2

该模式同时实现了自由演奏以及存储歌曲，具体实现见 bonus 部分描述。

mode_learn 模块: input wire clk, // Clock signal

input wire [1:0] song_select, // Input to choose song

input wire [6:0] switches, // Input from 7 switches

input wire reset, // Reset signal

input wire [1:0] octave_learn, // Input to choose the proper octave

output reg [3:0] note_to_play, // Output to buzzer

output reg [6:0] led_out, // Output for LEDs

output wire [3:0] num, // Output for displaying numbers

output reg [1:0] octave_out // Output the octave value

该模式为基础的学习模式模块，用户可以自由选择歌曲进行学习，该模式下，只有当用户按下正确的按键，才会进入下一个音符。

Controller 模块: 负责协调各个模式，输入输出都已在之前的各个模式出现过，此处就不再赘述。该模块的功能则是根据顶层 minipiano 传入的模式选择信号来切换到不同模式。input wire clk,

input wire write_on,

input wire [6:0] keys,

input wire [1:0]octave, //choose the proper octave

input wire [2:0] mode,// mode 100 free ; 010 auto; 001 learn //check constrain

input wire reset,

input wire [1:0] song_select,

output reg[3:0] num, //show the number of song

output reg [3:0] note_out, //output the note

output reg [6:0] led_out ,

output reg [1:0] octave_out, //octave from auto mode or learn mode

input wire [1:0] speed_select,

input wire start,

output reg [3:0] score_out,

input wire user,

output reg [3:0] score_user

lib 模块: 存放音乐信息，学习模式和自动模式都会从里面获取歌曲的音符，八度，持续时间

wire clk, // Clock signal

input wire [3:0] song_num, // on behalf of song

output reg [223:0] song_packed, // out_put song

output reg [223:0] time_continue, // each note time

output reg [111:0] octave_packed, // out_put ocative

output reg [3:0]num

7 bonus 实现说明

7.1 歌曲存储实现

7.2 评分实现

我们在竞赛模式当中，实现了存储两个用户的上一次演奏的评分，以及显示某个用户在演奏过程中的实时评分，具体评分是我们会记录在某个 LED 灯亮起的这段时间内，用户能不能正确打开对应的开关，如果可以，正确数加一；在 LED 灯熄灭的这段时间内，用户能不能及时的把所有开关关闭，如果可以，正确数加一。在每个时钟上升沿，我们会根据已经播放的音符数量和正确数的比例，显示评分，当二者相等，评分为 S；当正确数是已播放数的三分之二及以上，评分为 A；以此类推还有 B，C。当一首歌演奏完，我们会将这个最终的评分记录进用户上一次演奏的评分，覆盖之前的评分。

7.3 音乐节奏实现

我们在播放音乐时会从曲库里获取每个音符的持续时间，我们以 0.1s 为基准时间（实际上是根据 clk 来设置的时钟上升沿次数），区库里存储的是一个倍数，在播放时我们会将倍数乘以基准时间，然后让判断在这么多个时钟上升沿下，音符信号都是持续传出的。

7.4 VGA 实现

首先要对 csdn 博主七十二骑士进行感谢，该模式参考

https://blog.csdn.net/qq_43796199/article/details/119916049 博客，但在此基础上进行了很大程度的修改，这里先附上原博主的实现来跟我们的进行区分。

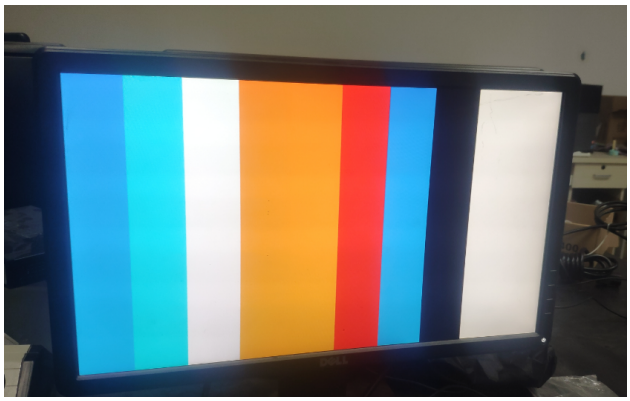

图 3: 原博客效果图

原博客只实现了固定的颜色显示，但我们从中得到了一些启发能使得我们的 VGA 实现更有趣味性。

VGA 功能主要有三个模块实现:

vga_colobar：该模块为顶层模块，主要接受系统时钟和复位信号，输出 12 位 rgb，因为 EGO1 的 VGA 接口是 444 类型的，所以总共是 12 位，该模块还输出 hsync 和 vsync 信号

clk_gen：该模块输入的是系统时钟，还有 areset 信号，输出 clk_out 和 locked。该模块实现了分频，将系统时钟 100MHz 分频为 25MHz，并通过 clk_out 输出，这里我们用了自定义的 ip 核来实现时钟切换

vga_ctrl：该模块是 vga 驱动模块，在 25MHz 的工作时钟下，产生横纵坐标信号 (pix_x,pix_y)−−−−>(639,479), 并且将该横纵坐标信号输出给 vga_pic 模块，生成行场、同步信号，再把 pix_data 像素点信息传输到 rgb 输出端口

vga_pic：根据输入的时钟信号，复位信号，以及坐标信号，产生 pix_data 像素信息

先附上我们的实现效果，由于动态结果无法掩饰，最终重现可以参考我们的 bonus 演示视频。

图 4: VGA 实现效果

根据原博主的启发，我们发现这种一个一个的条纹很像我们的钢琴键，因此我们保留了左边 7 条条纹，并将底色设置成白色。

同时，为了更像钢琴，我们在中间加入了黑色细条纹来区分每个琴键。并在右边预留了一大块区域来放置歌曲名字。

为了使演奏更加有趣味，我们将 led 信号与左边的琴键进行关联，在原有的逻辑上 led 灯显示的同时左边的对应琴键会变成黑色，实现动态更新，同时，更新的持续时间与音符持续时间相关联。

针对字符的显示在 VGA 上交有难度，我们利用了在线取模软件对想要的字进行了像素取模，在播放对应歌曲的时候会在我们设置的像素点的位置变成黑色，最终实现看起来显示字的效果。同时，我们还能对歌曲进行切换，切换的逻辑与七段数码管切换逻辑一致，字模参数存在我们的参数

文件中。

8 项目总结

本次项目进展总体来说较为顺利，一开始我们每个人都做好了明确的分工，并且一开始就对整体框架进行了讨论，所以我们每个人都是框架的修改者，没有分出专门的成员来进行顶层模块的实时更新，整体也没有出现什么冲突。

同时，我们利用 git 进行协作，但在合并代码时出现了几次冲突修改错误，最终不小心删除了队友部分代码的情况，这点在后期我们进行了调整，重新协调并学习了 git 使用及规范，在后期就没有出现这样的问题。

解决了上述问题，在进行总体测试时较为顺利，这是因为我们在之前讨论了几个可能出现的问题，比如说按键的使用，reset 信号的高低电平设置，顶层模块参数名字的同一定义。除了有些时候约束文件可能会落掉一些新加的信号，整体上还是比较顺利。

在进行 bonus 部分实现时，我们进行了比较集中的几次讨论，这是因为有些信号会关联到多个模块，比如说自动演奏模式和自由模式的存歌，我们根据分工来进行对队友需要部分的修改。

整体团队合作较为顺利，我们每个人各司其职，平均分工进行协作，讨论和沟通也及时有效，实现了 1+1+1>3 的效果。

9 project 设计

street-Figther 街头霸王

项目描述：祝大是 sust 一名街头霸王爱好者，由于正在学习数字逻辑课程，他希望用 EGO1 开发板实现对街头霸王的操作。

可能需要类似原神厨房项目在电脑上实现一些基础的显示。

可能的一些实现：

利用五个按键实现人物的跳跃，下蹲，移动，中间按钮也许可以作为开始。

左边八个按钮也许可以作为攻击键，轻拳，腿，中拳，腿等等

也许可以实现自由模式，玩家可以任意练习，实现基础搓招。

学习模式：玩家需要根据指示进行连招练习，连招要先存在开发板内，可以根据 led 灯指示来进行训练

自动攻击模式，利用开发板内存好的招数进行自动攻击。

bonus 部分：1、存入一键连招操作，允许用户实时存入连招到一个按键，拨下按键时会自动发出一套连招。2、为你的连招设计一些炫酷的音效，并使用蜂鸣器，在出招的时候发出这些存储进去的音效。