

数字电子技术开放性实验报告

**题 目 射频电子琴设计**

**电气工程 学院 电气工程及其自动化 系**

**学生姓名 王浩宇 班级 电气813 学号 2186114150**

**学生姓名 聂永欣 班级 电气810 学号 2186113564**

**指导教师 金印斌**

2020年12月

**实验题目：射频电子琴设计**

**学生姓名：王浩宇、聂永欣**

**指导教师：金印斌**

摘 要

**FPGA**（Field－Programmable Gate Array），即现场可编程门阵列，是在PAL、GAL、CPLD等可编程器件的基础上进一步发展的产物。作为专用[集成电路](https://baike.so.com/doc/647555-685412.html)（ASIC）领域中的一种半定制电路，它的出现既解决了定制电路的不足，又克服了原有可编程器件门电路数有限的缺点。而**EGO1**是一种基于FPGA电路的新式开发板，使用Verilog语言进行编译，功能强大。

**电子琴**又称作电子键盘，属于[电子乐器](https://baike.so.com/doc/6025623-6238621.html)(区别于电声乐器)，发音音量可以自由调节。音域较宽，和声丰富，甚至可以演奏出一个[管弦乐队](https://baike.so.com/doc/2377717-2514092.html)的效果，表现力极其丰富。它还可模仿多种[音色](https://baike.so.com/doc/1834850-1940380.html)，甚至可以奏出常规乐器所无法发出的声音(如合唱声，风雨声，宇宙声等)。另外，电子琴在独奏时，还可随意配上类似打击乐音响的节拍伴奏，适合于演奏节奏性较强的现代音乐。另外，电子琴还安装有效果器，如混响、回声、延音，震音轮和调制轮等多项功能装置，表达各种情绪时运用自如。

在对Verilog语言较为熟悉、能够实现EGO1基础功能、了解电子琴工作原理及功能的前提下，应用EGO1开发板设计了一款射频电子琴，较为全面且规范得完成了射频电子琴的设计要求。

本程序应用**顶层模块**与**功能模块**相结合的方法，将电子琴的功能进行分解、并分区实现，各区联动，共同构成一个完整的番茄种程序。首先，顶层**TOP**起到了调配输入和输出，联结各子模块，完成模块间互动与通讯的功能。将输入与输出变量合理命名，统一放置，易于程序的修改和纠错；同时具有修改所使用键盘为ps2键盘还是矩阵键盘，以及切换自动播放内置歌曲和手动输入音符的功能。模块**clk1**功能为产生一个100Hz脉冲并输出，同时可选择**3**种内置歌曲进行播放；模块**k1**功能为矩阵键盘的输入，具体包括检测矩阵键盘的输入信号，对输入信号进行消抖，将输入信号转化为键值输出信号；模块**ps2\_1**功能为ps2键盘的输入，具体包括检测ps2键盘的输入信号，对输入信号进行消抖，将输入信号转化为键值输出信号等；同时，模块**a1**主体功能为放音，具体包括接收音调指令，将从**k1**或**ps2\_1**来源的键值输入信号转化为对应振动频率的音调，并输出音调；模块**auto1**为内置歌曲功能模块，其功能为通过**TOP**层的播放内置歌曲按钮控制内置的三首歌：天空之城、、的播放；模块**d1**为数码管模块，主要将输入的音符信号转换为可视的晶体管灯，可通过晶体管显示弹奏音调的高低。

**关 键 词**：FPGA；EGO1；电子琴；音乐自动播放；ps2键盘

目 录

[1 前 言 4](#_Toc60239436)

[1.1 设计背景 4](#_Toc60239437)

[1.2 主要实现功能 4](#_Toc60239438)

[1.3 设计思路 4](#_Toc60239439)

[1.4 设计难点与亮点 5](#_Toc60239440)

[1.4.1 设计亮点 5](#_Toc60239441)

[1.4.2 设计难点 5](#_Toc60239442)

[2 系统设计 6](#_Toc60239443)

[2.1 总体结构及功能设计 6](#_Toc60239444)

[2.1.1 实验平台 6](#_Toc60239445)

[2.1.2 板载音频接口 6](#_Toc60239446)

[2.1.3 总体结构及功能设计 7](#_Toc60239447)

[2.1.4 输入输出设计 8](#_Toc60239448)

[2.1.5 子模块划分 8](#_Toc60239449)

[2.2 RTL顶层逻辑图 9](#_Toc60239450)

[3 系统调试 10](#_Toc60239451)

[4 功能 11](#_Toc60239452)

[4.1 Ps2键盘输入模块 11](#_Toc60239453)

[4.1.1 Ps2键盘输入模块（ps2\_1:ps2\_kyboard） 11](#_Toc60239454)

[4.2 音调状态机转化为音调模块 11](#_Toc60239455)

[4.2.1 音调状态机转化为音调模块（a1:audio\_port） 11](#_Toc60239456)

[5 总结 13](#_Toc60239457)

[5.1 所做工作 13](#_Toc60239458)

[5.2 课程收获 13](#_Toc60239459)

[5.3 课程意见 13](#_Toc60239460)

[6 附录 14](#_Toc60239461)

[6.1 源文件 14](#_Toc60239462)

[6.1.1 顶层代码——TOP.v 14](#_Toc60239463)

[6.1.2 时钟分频模块 15](#_Toc60239464)

[6.1.3 矩阵键盘模块 16](#_Toc60239465)

[6.1.4 ps2键盘 18](#_Toc60239466)

[6.1.5 放音模块 22](#_Toc60239467)

[6.1.6 自动放音模块 23](#_Toc60239468)

[6.1.7 数字显示模块 36](#_Toc60239469)

[6.2 约束文件 41](#_Toc60239470)

[6.2.1 top1.xdc 41](#_Toc60239471)

[6.2.2 port\_constrains.xdc 43](#_Toc60239472)

# 前 言

## 设计背景

电子琴是一种电子键盘乐器，属于电子合成器。它采用大规模集成电路，大多配置声音记忆存储器（波表）。用于存放各类乐器的真实声音波形并在演奏的时候输出。常用的电子琴有编曲键盘（带自动伴奏）和合成器（无自动伴奏）两大类，广义上的电子琴包括电子钢琴（数码钢琴，区别于电声钢琴），多使用五线谱，多为高低音双行记谱。有时也用中音谱和简谱、吉他谱。一般用于摇滚乐。

## 主要实现功能

本文主要进行了射频电子琴的功能设计、编程、实现可视化的工作，分别设计实现了以下功能：

(1) 基础功能：使用ps2键盘输入，实现了音域为三个大组跨度，21个音符的准确音高任意弹奏；

(2) 提高功能：内置多首流行歌曲，可通过按键自由选择播放；通过设置按键可选择静音或放音；通过设置按键实现了播放内置歌曲和自由弹奏的模式切换。

## 设计思路

对于总体功能的设计，有限状态机是本系统实现的核心思想。首先，设计输入阶段创建FPGA工程，添加源文件等；然后是设计综合阶段，FPGA开发工具的综合引擎将编译整个设计，并将HDL源文件转译为特定结构的设计网表；接下来是约束输入，通过指定番茄钟时序以及布线布局；这些过程中穿插设计仿真，对工程进行功能或时序验证。再之后是设计实现，将逻辑设计进一步转译为目标器件中的特定物理文件格式，即生成bitstream文件；然后对结果进行分析，并对资源占用率，结果功耗等进行分析；接着提出设计优化，进行修改，不断优化结果，重新设计综合，约束输入，再一次实现设计，最后板级调试。

对应本工程而言。首先，编码出一个顶层文件，并命名为**TOP**，思考并列举出所需要实现的功能个数，并创建出以功能命名的具有序列的多个设计文件，排列整齐等待实现。初步思考需要输入和输出的变量个数，并以功能和逻辑命名，做到条理清楚，易于修改和添加。

对于基础功能的设计，分模块进行编程和联动。要实现ps2键盘输入，音域为三个大组跨度，21个音符的准确音高任意弹奏，需要ps2键盘协议部分、ps2键值输出模块、放音模块、音符振动频率定义模块，需要生成秒脉冲模块等；对于自动播放模式和自由弹奏模式，需要添加状态机进行状态切换。

对于提高功能的设计，可运用基础功能的一些模块已经写好的功能，进行重命名和改编。为了实现添加多首内置歌曲并播放的功能，在网络上下载学习了多首歌曲的简谱，并将简谱翻译成为verilog程序；对于设置按键静音或者播放，通过使用状态机，设置audio模块正常工作或停止工作。

## 设计难点与亮点

### 设计亮点

在以上功能的基础上，我们通过FPGA的音频接口，实现了一个非常灵敏的可多音同时弹奏的，具有个性化内置歌曲的射频电子琴。使用者可以根据自己的喜好更改内置歌曲，弹奏自己喜爱的曲子，甚至进行乐理的教学。

### 设计难点

1. 如何通过变成手段体现不同高低音符的特征；
2. 如何用较少的代码量实现自动播放歌曲，即将歌曲的曲调写入程序之中，并保证代码的可维护性。

# 系统设计

## 总体结构及功能设计

### 实验平台

FPGA取自Field Programmable Gate Array这四个英文单词，直译为“现场可编程逻辑阵列”，就是可反复变成的逻辑器件（可编程数字电路）。是在PAL、GAL等可编程器件的基础上进一步发展的产物。它是作为专用集成电路（ASIC）领域中的一种半定制电路而出现的，既解决了定制电路的不足，又克服了原有可编程器件门电路数有限的缺点。

FPGA 器件属于专用集成电路中的一种半定制电路，是可编程的逻辑列阵，能够有效的解决原有的器件门电路数较少的问题。FPGA的基本结构包括可编程输入输出单元，可配置逻辑块，数字时钟管理模块，嵌入式块RAM，布线资源，内嵌专用硬核，底层内嵌功能单元。

FPGA基本的组成有三大部分：

1. 可配置逻辑块（Configurable Logic Blocks，CLB）；
2. 输入输出模块（Input Output Blocks，IOB）；
3. 内部互联资源（Interconnector）。

而本实验采用的是由依元素科技基于Xilinx Artix-7 FPGA研发的便携式数模混合基础教学实验平台。EGO1配备的FPGA具有大容量高性能的特点，能实现较复杂的数字逻辑。该平台拥有丰富的外设以及灵活的通用拓展接口。

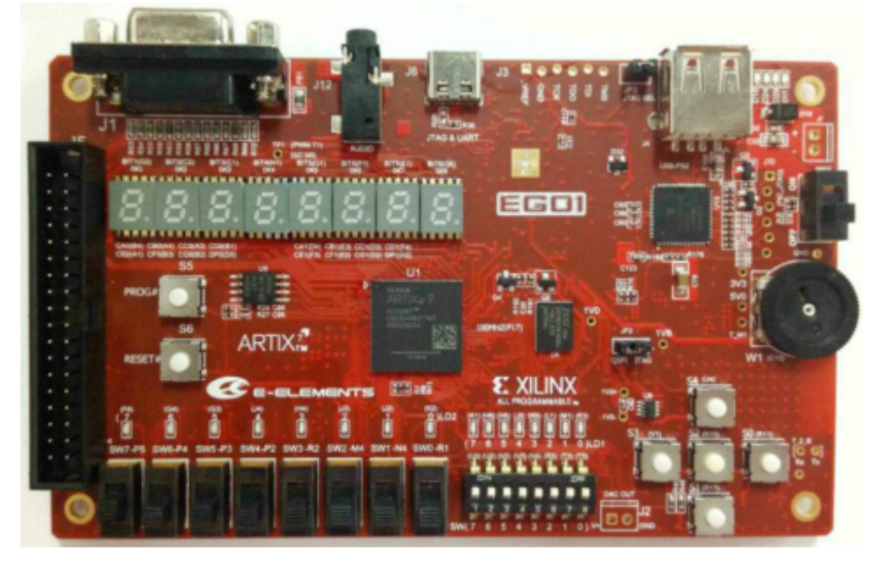


图2.1 EGO1开发板

### 板载音频接口

1. **音频接口简介**

EGO1 上的单声道音频输出接口（J12） 由下图所示的低通滤波器电路驱动。滤波器的输入信号（AUDIO\_PWM）是由 FPGA 产生的脉冲宽度调制信号（PWM）或脉冲密度调制信号（PDM）。 低通滤波器将输入的数字信号转化为模拟电压信号输出到音频插孔上。

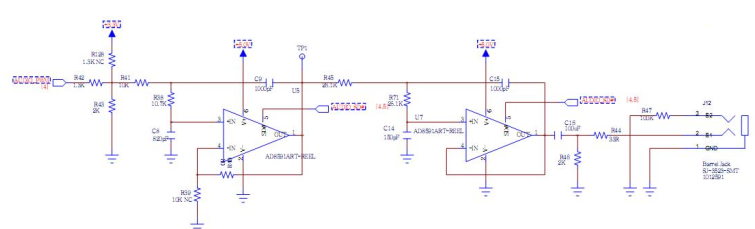


图2.2 音频接口电路图

1. **PWM脉冲宽度调制产生声音信号**

脉冲宽度调制信号是一连串频率固定的脉冲信号， 每个脉冲的宽度都可能不同。 这种数字信号在通过一个简单的低通滤波器后，被转化为模拟电压信号，电压的大小跟一定区间内的平均脉冲宽度成正比。这个区间由低通滤波器的3dB截止频率和脉冲频率共同决定。例如，脉冲为高电平的时间占有效脉冲周期的10%的话，滤波电路产生的模拟电压值就是 Vdd 电压的十分之一。下图是一个简单的PWM信号波形：

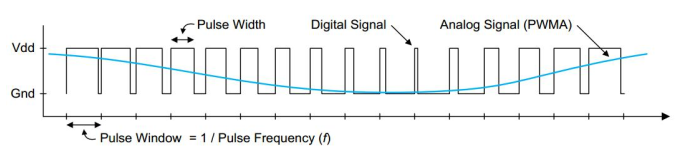


图2.3 PWM信号波形

低通滤波器3dB频率要比PWM信号频率低一个数量级，这样PWM频率上的信号能量才能从输入信号中过滤出来。例如，要得到一个最高频率为5KHz的音频信号，那么PWM信号的频率至少为50KHz或者更高。通常，考虑到模拟信号的保真度，PWM信号的频率越高越好。下图是PWM信号整合之后输出模拟电压的过程示意图，可以看到滤波器输出信号幅度与Vdd的比值等于PWM信号的占空比。

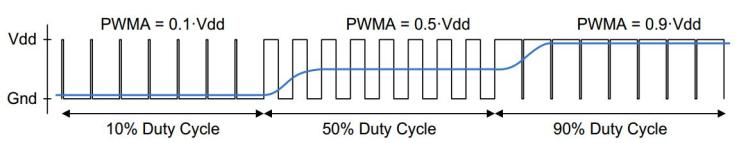


图2.3 PWM信号整合后输出模拟电压

通过控制PWM信号占空比的方式，我们进而控制模拟电压的幅值进而控制输出音频的音调高低以产生不同的音符，进而实现电子琴的基础弹奏功能。

### 总体结构及功能设计

1. 顶层模块：**TOP.v。**起到了统筹各模块，实现模块间互联的作用。
2. 分区模块：**clk1：clk\_control、k1:keyboard、ps2\_1:ps2\_keyboard、a1:audio\_port、auto1:auto\_display、d1:scan\_led。**
3. 本程序使用ps2键盘输入，实现了音域为三个大组跨度，21个音符的准确音高任意弹奏，内置多首流行歌曲，可通过按键自由选择播放；通过设置按键可选择静音或放音；通过设置按键实现了播放内置歌曲和自由弹奏的模式切换，因此本程序实现的电子琴具有很强的实用性。

### 输入输出设计

输入变量名及功能如表1-1所示：

表2-1 总体输入变量功能

|  |  |
| --- | --- |
| 输入变量 | 功能 |
|  | 产生100MHz的脉冲 |
|  | 复位 |
|  | 选择键盘类型 |
|  | PS2键盘时钟 |
|  | PS2键盘的数据输入 |
|  | 矩阵键盘输入 |
|  | 自动放音使能 |
|  | 歌曲选择 |
|  | 低通滤波器使能 |

输出变量名及功能如表1-2所示：

表2-2 总体输出变量功能

|  |  |
| --- | --- |
| 输入变量 | 功能 |
|  | 低通滤波器使能 |
|  | 音频输出 |
|  | 矩阵键盘扫描 |
|  | 数码管显示 |
|  | 数码管片选 |
|  | 数码管段选 |

### 子模块划分

1. clk1：clk\_control：产生一个100Hz脉冲并输出，同时可选择**3**种内置歌曲进行播放。
2. k1:keyboard：矩阵键盘的输入，具体包括检测矩阵键盘的输入信号，对输入信号进行消抖，将输入信号转化为键值输出信号。
3. ps2\_1:ps2\_keyboard：ps2键盘的输入，具体包括检测ps2键盘的输入信号，对输入信号进行消抖，将输入信号转化为键值输出信号等。
4. a1:audio\_port：功能为放音，具体包括接收音调指令，将从**k1**或**ps2\_1**来源的键值输入信号转化为对应振动频率的音调，并输出音调。
5. auto1:auto\_display：内置歌曲功能模块，其功能为通过**TOP**层的播放内置歌曲按钮控制内置的三首歌：天空之城、、的播放。
6. d1:scan\_led：数码管模块，主要将输入的音符信号转换为可视的晶体管灯，可通过晶体管显示弹奏音调的高低。

## RTL顶层逻辑图

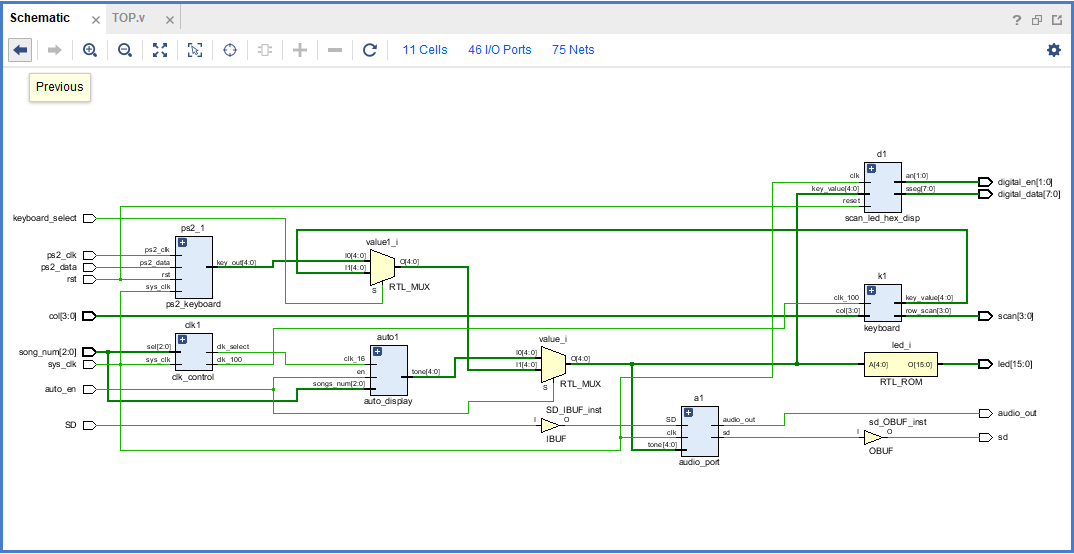


图2.4 总程序RTL分析

如图所示，分析电路图可清晰得出，ps2键盘的输入被clk迅速检测，同时转化为键值输出信号key\_out输出，紧接着又被keyboard\_select进行选择；选择输出后，进入放音audio模块，但放音之前还要被auto\_en检测，如果此时用户选择的是手动弹奏，则他通过检测，顺利进入放音模块audio，进而成功输出声音。而对于自动播放内置歌曲的song\_num键，可通过类似以上的分析，进而得出其工作特性。

如图所示，按键控制状态转移图：

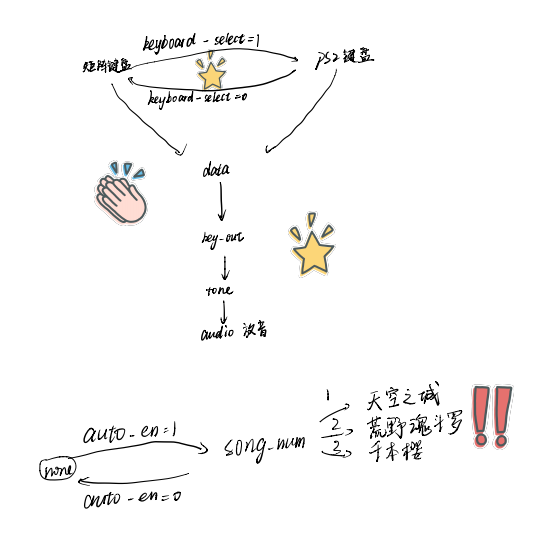


图2.5 状态转换图

# 系统调试

系统调试是整个系统功能否实现的关键步骤，我们将整个调试过程分为三大部分：硬件调试、软件调试和综合调试。

## 硬件调试

首先根据设计原理图，将开发板的PWM信号输出引脚与蜂鸣器输入端相接，然后将PWM信号输入引脚与PS2键盘相接。然后，通过一个简单的程序测试蜂鸣器是否能发出声音，进而测试连线是否正确。之后，写一个矩阵键盘控制LED灯与数码管的程序，测试矩阵键盘是否实现功能。测试过程中发现，矩阵键盘按键和设计的返回的键值不一样，经过检查发现是程序接线顺序出错。改正后运行良好。

## 软件调试

调试主要方法和技巧：通常一个调试程序应该具备至少四种性能：跟踪、断点、查看变量、更改数值。整个程序是一个主程序调用各个子程序实现功能的过程，要使主程序和整个程序都能平稳运行，各个模块的子程序的正确与平稳运行必不可少，所以在软件调试的最初阶段就是把各个子程序模块进行分别调试。

## 综合调试

将程序下载到开发板中，按下按键检查是否实现设计的功能。测试过程中发现的问题有：

（1）多个按键发出的声音频率是一样的，有的按键按下没有声音。检查程序发现判断键值时候，GPIOA端口判断值写错。

（2）按键按下，蜂鸣器声音不够清脆，有杂音。经测试，发现输出PWM方波的延时时间不够，增大延时时间后，声音变得清脆，没有杂音。

（3）某些按键在按下后没有声音，发现是PS2按键没有消抖，而按键过多不可能一个个消抖，只能通过统一消抖的方式对键盘按键消抖。

# 功能

## PS2键盘输入模块

### PS2键盘输入模块（ps2\_1:ps2\_kyboard）

设置了四个输入，分别是复位键、键盘时钟检测、键盘数据传送、以及FPGA时钟检测，同时设置了一个输出：键值输出。

使用always语句检测键盘时钟，完成数据滤波：当检测到FPGA上升沿或者复位下降沿时，检测ps2的时钟、ps2输入数据，并储存在reg形式的存储器filter中，并且将输入信号通过创建脉冲的方式除去毛刺；紧接着，将输入的去除毛刺的有效值转化为输出key\_out，即音调的状态机，因为我们设计的音域跨度为三个组21个音调，因此创建了21个key\_out的状态机。



图4.1 PS2键盘

## 矩阵键盘输入模块

首先，将寄存器初始化为无效键值，通过对键盘信号以100Hz的频率进行扫描，分别扫描列信号和行信号，并进行消抖，将所得键值进行译码存入key\_value中输出。

## 音调状态机转化为音调模块

### 音调状态机转化为音调模块（a1:audio\_port）

查阅蜂鸣器——音频对照表，并且结合乐理知识，我们选择了从C3~B5三个大组跨度，21个音符作为我们弹奏的音域。首先因为EGO1自带时钟频率为100MHz，C3的频率查表可知为523Hz，因此，对于C3状态，设置计数器上限为100MHz/523Hz，取整，其他音调同理。这样我们便把输入键盘信号转化成为了音符信号，达到了不同按键弹奏不同音高的效果。

### 自动放音模块（auto1:auto\_display.v）

通过上述模块的key\_value的值，改变数码管和LED的使能信号，进而使的按键结果在数码上显示。

## 数码管显示模块（d1:scan\_led\_hex\_disp.v）

首先，将21个音符标记为A1~A7、B1~B7、C1~C7，分别代表三个大组，音调从A至C依次增高；接着，调用数码管显示模块，将21个音符标记对应的数码管显示位一一写入，譬如：当输入A1，第一个七段数码管为A，第二个七段数码管从a至g依次为1111110，这样便把A1显示出来；当输入其他音符时以此类推。

# 总结

## 所做工作

本程序完成了射频电子琴的弹奏、播放、提示、静音功能，功能全面且完备，设计新颖，别具一格。

## 课程收获

经过8周的学习，我们终于结束了《数字电子技术》这门课的学习，而随后的几周，我们又完成了《数字电子技术》的相关实验及其相关课程。

通过学习《数字电子技术》这门课以及设计开放性实验，我学到了很多，不仅学到了充实的理论知识，为我之后进一步的专业课的学习奠定了基础，更掌握了Verilog HDL这门硬件编程语言，为我们以后的科研和学习提供了一种新的工具。

在实验过程中，我们需要将基础知识与硬件编程结合起来，在这个过程中，我们进一步提升了对基本数字电子技术知识的理解，同时，还需要查阅用户手册、数据手册，在这个过程中也极大地提高了我的自学能力。

作为一个开发板小白，之前从未接触过开发板的编程，因此这项开放性实验对我和我的搭档而言还是很有难度的。但是每当独立完成相应功能，看到开发板亮起来的时候就会觉得付出是值得的。此课程相当于一位引路人，将我引上开发板编程的道路，有了对FPGA编程的经验，相信以后在面对其他开发板编程时也会有所裨益。

## 课程意见

此次开放性实验课程总体来讲我认为任务难度比较大，需要自学和自行查阅资料的内容非常多，并且有几个模块我们的数电实验并未涉及到，需要自行学习与熟悉。但是按时完成试验后，我们队伍的收获也非常多，相信下次遇到类似的开发板问题时我们能够更加迅速准确有效地解决。

# 附录

## 源文件

### 顶层代码——TOP.v

`timescale 1ns / 1ps

//顶层模块

module TOP(

input sys\_clk,//100MHZ

input rst,//复位键

input keyboard\_select,//键盘选择矩阵键盘orPS2键盘

input ps2\_clk,//ps2键盘时钟

input ps2\_data, //ps2键盘数据线

input [3:0] col,//矩阵键盘列输入

input auto\_en,//自动放音使能

input [2:0] song\_num,//歌曲选择

input SD,//低通滤波器使能

output sd,//低通滤波器使能

output audio\_out,//音频输出

output [3:0] scan,//矩阵键盘行扫描

output reg [15:0] led,//LED输出

output [1:0] digital\_en,//数码管片选

output [7:0] digital\_data//数码管段选

);

wire clk\_100;//100HZ

wire clk\_sel;//节拍时钟

wire [4:0] value1;//最终键盘键值

wire [4:0] value2;//曲谱生成模块输出键值

wire [4:0] value;//最终输出键值

wire [4:0] key1;//矩阵键盘键值

wire [4:0] key2;//ps2键盘键值

assign value1=keyboard\_select?key2:key1;//键盘使能控制

assign value=auto\_en?value2:value1;//人工or自动模式控制

//实例化

clk\_control clk1(sys\_clk,song\_num,clk\_100,clk\_sel);

keyboard k1(clk\_100,col,scan,key1);

ps2\_keyboard ps2\_1(rst,ps2\_clk,ps2\_data,sys\_clk,key2);

audio\_port a1(sys\_clk,SD,value,sd,audio\_out);

auto\_display auto1(clk\_sel,auto\_en,song\_num,value2);

scan\_led\_hex\_disp d1(sys\_clk,rst,value,digital\_en,digital\_data);

//LED控制

always @(value)

begin

case(value)

5'd0:led<=16'b1;

5'd1:led<=16'b10;

5'd2:led<=16'b100;

5'd3:led<=16'b1000;

5'd4:led<=16'b10000;

5'd5:led<=16'b100000;

5'd6:led<=16'b1000000;

5'd7:led<=16'b10000000;

5'd8:led<=16'b100000000;

5'd9:led<=16'b1000000000;

5'd10:led<=16'b10000000000;

5'd11:led<=16'b100000000000;

5'd12:led<=16'b1000000000000;

5'd13:led<=16'b10000000000000;

5'd14:led<=16'b100000000000000;

5'd15:led<=16'b1000000000000000;

default:led<=16'b0;

endcase

end

endmodule

### 时钟分频模块

`timescale 1ns / 1ps

//时钟分频模块

module clk\_control(

input sys\_clk,//100MHZ

input [2:0] sel,//歌曲选择

output reg clk\_100,//矩阵键盘时钟100HZ

output reg clk\_select//节拍时钟，不同歌曲可能不同

);

reg [19:0] count1;//键盘时钟分频计数变量

reg [25:0] count2;//节拍时钟分频计数变量

reg [25:0] COUNT;//节拍分频预置数变量

//寄存器类型变量初始化

initial

begin

clk\_100<=1'b0;

clk\_select<=1'b0;

count1<=1'b0;

count2<=1'b0;

COUNT<=1'b0;

end

//100HZ时钟分频

always @(posedge sys\_clk)

begin

if(count1>=20'd500000)

begin

count1<=20'b0;

clk\_100<=~clk\_100;

end

else

begin

count1<=count1+1'b1;

end

end

//歌曲节拍选择

always @(sel)

begin

case(sel)

3'b100:COUNT<=26'd3125000;

3'b010:COUNT<=26'd12500000;

default:COUNT<=26'd12500000;

endcase

end

//节拍时钟分频

always @(posedge sys\_clk)

begin

if(count2>=COUNT)

begin

count2<=1'b0;

clk\_select<=~clk\_select;

end

else

begin

count2<=count2+1'b1;

end

end

endmodule

### 矩阵键盘模块

`timescale 1ns / 1ps

//矩阵键盘模块

module keyboard(

input clk\_100,//键盘时钟100HZ

input [3:0] col,//列信号

output reg [3:0] row\_scan,//行扫描信号

output reg [4:0] key\_value//键值输出限号

);

wire key\_pressed;//0为有键按下

wire [7:0] data;//键盘行列状态编码

reg [3:0] col\_filter;//去抖后的列信号

assign data={row\_scan[3:0],col\_filter[3:0]};

assign key\_pressed=col\_filter[3]&col\_filter[2]&col\_filter[1]&col\_filter[0];//列信号的归与

//寄存器变量的初始化

initial

begin

col\_filter<=4'b1111;//初始化为无键按下

row\_scan<=4'b1111;//初始化不扫描

key\_value<=5'd22;//初始化为无效键值

end

//键盘扫描模块(去抖)

//列信号去抖

always @(posedge clk\_100)

begin

col\_filter<=col;

end

//行扫描

always @(negedge clk\_100)

begin

if(key\_pressed)

begin

case(row\_scan)

4'b1110:row\_scan<=4'b1101;

4'b1101:row\_scan<=4'b1011;

4'b1011:row\_scan<=4'b0111;

4'b0111:row\_scan<=4'b1110;

default:row\_scan<=4'b1110;

endcase

end

end

//键值译码模块

always @(posedge clk\_100)

begin

case(data)

8'b1110\_1110:key\_value<=5'd0;

8'b1110\_1101:key\_value<=5'd1;

8'b1110\_1011:key\_value<=5'd2;

8'b1110\_0111:key\_value<=5'd3;

8'b1101\_1110:key\_value<=5'd4;

8'b1101\_1101:key\_value<=5'd5;

8'b1101\_1011:key\_value<=5'd6;

8'b1101\_0111:key\_value<=5'd7;

8'b1011\_1110:key\_value<=5'd8;

8'b1011\_1101:key\_value<=5'd9;

8'b1011\_1011:key\_value<=5'd10;

8'b1011\_0111:key\_value<=5'd11;

8'b0111\_1110:key\_value<=5'd12;

8'b0111\_1101:key\_value<=5'd13;

8'b0111\_1011:key\_value<=5'd14;

8'b0111\_0111:key\_value<=5'd15;

default:key\_value<=5'd22;

endcase

end

endmodule

### ps2键盘

`timescale 1ns / 1ps

//ps2键盘

module ps2\_keyboard(

input rst,//复位键

input ps2\_clk,//键盘时钟线

input ps2\_data,//键盘数据线

input sys\_clk,//FPGA时钟

output reg[4:0] key\_out//键值输出

);

reg [1:0] key\_loosen;//1、2表示待按键松开，0表示待按下

reg PS2CF,PS2DF;//滤波后的键盘时钟和数据

reg [7:0] ps2c\_filter,ps2d\_filter;//键盘时钟和数据滤波器

reg [7:0] data\_temp;//当前接受的数据

reg [3:0] num;//移位控制

reg [7:0] key\_value;//有效键值

reg [1:0] error;//1、2、3为键盘发送数据乱码

wire flag;//1为键盘时钟上升沿标记

assign flag=(ps2c\_filter[1])&(~ps2c\_filter[0]);

//寄存器类型变量初始化

initial

begin

key\_out<=5'd22;//初始化为无效键值

key\_loosen<=1'b0;//初始化为待键按下

PS2CF<=1'b1;

PS2DF<=1'b1;

ps2c\_filter<=8'b11111111;

ps2d\_filter<=8'b11111111;

data\_temp<=8'b00000000;

num<=4'b0;

key\_value<=5'd0;

error<=1'b0;//初始化为无数据错误

end

//键盘时钟和数据滤波

always @(posedge sys\_clk or posedge rst)

begin

if(rst==0)

begin

ps2c\_filter<=8'b11111111;

ps2d\_filter<=8'b11111111;

PS2CF<=0;

PS2DF<=0;

end

else

begin

ps2c\_filter[7]<=ps2\_clk;

ps2c\_filter[6:0]<=ps2c\_filter[7:1];

ps2d\_filter[7]<=ps2\_data;

ps2d\_filter[6:0]<=ps2d\_filter[7:1];

if(ps2c\_filter==8'b11111111)

begin

PS2CF<=1;//去时钟毛刺

end

else if(ps2c\_filter==8'b00000000)

begin

PS2CF<=0;

end

if(ps2d\_filter==8'b11111111)

begin

PS2DF<=1;//去数据毛刺

end

else if(ps2d\_filter==8'b00000000)

begin

PS2DF<=0;

end

end

end

//数据接收

always @(negedge PS2CF or posedge rst)

begin

if(rst==0)

begin

num<=4'd0;

data\_temp<=8'd0;

end

else

begin

if(num==0)

begin

num<=num+1'b1;//跳过起始位

end

else if(num<=8)//数据位赋值

begin

data\_temp[num-1]<=PS2DF;

num<=num+1'b1;

end

else if(num==9)

begin

num<=num+1'b1;//跳过校验位

end

else

begin

num<=4'd0; //停止位清零

end

end

end

//按键处理（按下，松开），键值输出

always @(posedge sys\_clk or posedge rst)

begin

if(rst==0)

begin

key\_loosen<=2'b0;

end

else if(num==4'd10)//每接受到完整的数据包

begin

if(data\_temp==8'hF0)//判断是否为断码

begin

if(error==1)

begin

error<=2'd2;

end

else if(error==3)

begin

error<=2'd1;

end

else

begin

key\_loosen<=2'b1;//断码标识符

end

end

else

begin

if(key\_loosen==2'b1)//判断是否松开响应按键

begin

if(data\_temp==key\_value&&flag==1)

begin

key\_loosen<=2'b0;

key\_out<=5'd22;

end

end

else if(key\_loosen==2'b0&&flag==1&&(error==2'b00||error==2'b11))

begin

case(data\_temp)//判断是否有效键值

8'h15:error=0;

8'h1D:error=0;

8'h24:error=0;

8'h2D:error=0;

8'h2C:error=0;

8'h35:error=0;

8'h3C:error=0;

8'h1C:error=0;

8'h1B:error=0;

8'h23:error=0;

8'h2B:error=0;

8'h34:error=0;

8'h33:error=0;

8'h3B:error=0;

8'h1A:error=0;

8'h22:error=0;

8'h21:error=0;

8'h2A:error=0;

8'h32:error=0;

8'h31:error=0;

8'h3A:error=0;

default:error=2'b01;

endcase

if(error==0)//有效键值转换输出

begin

key\_value=data\_temp;

case(key\_value)

8'h15:key\_out<=5'd0;

8'h1D:key\_out<=5'd1;

8'h24:key\_out<=5'd2;

8'h2D:key\_out<=5'd3;

8'h2C:key\_out<=5'd4;

8'h35:key\_out<=5'd5;

8'h3C:key\_out<=5'd6;

8'h1C:key\_out<=5'd7;

8'h1B:key\_out<=5'd8;

8'h23:key\_out<=5'd9;

8'h2B:key\_out<=5'd10;

8'h34:key\_out<=5'd11;

8'h33:key\_out<=5'd12;

8'h3B:key\_out<=5'd13;

8'h1A:key\_out<=5'd14;

8'h22:key\_out<=5'd15;

8'h21:key\_out<=5'd16;

8'h2A:key\_out<=5'd17;

8'h32:key\_out<=5'd18;

8'h31:key\_out<=5'd19;

8'h3A:key\_out<=5'd20;

default:key\_out<=5'd22;

endcase

end

end

else if(error==2'd2)

begin

error<=2'd3;

end

end

end

end

endmodule

### 放音模块

module audio\_port(

input clk,//100MHZ时钟

input SD,//低通滤波器使能

input [4:0] tone,//音调指令接收

output sd,//低通滤波器使能

output reg audio\_out//音调输出

);

reg [20:0] count,COUNT;//分频计数器和预置数变量

reg [20:0] buffer[20:0];//保存相应频率的预置数

assign sd=SD;

//寄存器类型变量初始化

initial

begin

buffer[0]=21'd191110;

buffer[1]=21'd170259;

buffer[2]=21'd151685;

buffer[3]=21'd143172;

buffer[4]=21'd127554;

buffer[5]=21'd113636;

buffer[6]=21'd101239;

buffer[7]=21'd95557;

buffer[8]=21'd85131;

buffer[9]=21'd75844;

buffer[10]=21'd71689;

buffer[11]=21'd63776;

buffer[12]=21'd56818;

buffer[13]=21'd50620;

buffer[14]=21'd47778;

buffer[15]=21'd42566;

buffer[16]=21'd37951;

buffer[17]=21'd35793;

buffer[18]=21'd31888;

buffer[19]=21'd28409;

buffer[20]=21'd25310;

end

//分频输出

always @ (posedge clk)

begin

if(count>=COUNT&&COUNT!=1)

begin

count<=0;

audio\_out<=~audio\_out;

end

else if(COUNT==1)

audio\_out<=0;

else

count<=count+1'b1;

end

//音调指令接收，预置数变化

always @(tone)

if(tone>=0&&tone<=20)

COUNT=buffer[tone];

else

COUNT=1;

Endmodule

### 自动放音模块

`timescale 1ns / 1ps

//自动放音模块

module auto\_display(

input clk\_16,//节拍时钟

input en,//1为使能

input [2:0] songs\_num,//歌曲选择

output reg [4:0] tone//键值输出

);

reg [19:0] count;//时间计数器

//寄存器类型变量初始化

initial

begin

tone<=5'd22;//初始化为无效键值

count<=1'b0;

end

//内置曲谱并按节拍读取输出

always @(posedge clk\_16)

begin

if(en==1'b1)

begin

case(songs\_num)

3'b100://千本樱

begin

if(count>=192)

begin

count<=1'b0;

end

else

begin

count<=count+1'b1;

end

case(count)

20'd0:tone<=5'd12;

20'd1:tone<=5'd12;

20'd2:tone<=5'd12;

20'd3:tone<=5'd12;

20'd4:tone<=5'd12;

20'd5:tone<=5'd12;

20'd6:tone<=5'd11;

20'd7:tone<=5'd11;

20'd8:tone<=5'd12;

20'd9:tone<=5'd12;

20'd10:tone<=5'd12;

20'd11:tone<=5'd12;

20'd12:tone<=5'd12;

20'd13:tone<=5'd12;

20'd14:tone<=5'd11;

20'd15:tone<=5'd11;

20'd16:tone<=5'd12;

20'd17:tone<=5'd12;

20'd18:tone<=5'd12;

20'd19:tone<=5'd12;

20'd20:tone<=5'd11;

20'd21:tone<=5'd11;

20'd22:tone<=5'd11;

20'd23:tone<=5'd11;

20'd24:tone<=5'd12;

20'd25:tone<=5'd12;

20'd26:tone<=5'd15;

20'd27:tone<=5'd15;

20'd28:tone<=5'd15;

20'd29:tone<=5'd15;

20'd30:tone<=5'd12;

20'd31:tone<=5'd12;

20'd32:tone<=5'd12;

20'd33:tone<=5'd12;

20'd34:tone<=5'd12;

20'd35:tone<=5'd12;

20'd36:tone<=5'd11;

20'd37:tone<=5'd11;

20'd38:tone<=5'd12;

20'd39:tone<=5'd12;

20'd40:tone<=5'd12;

20'd41:tone<=5'd12;

20'd42:tone<=5'd12;

20'd43:tone<=5'd12;

20'd44:tone<=5'd11;

20'd45:tone<=5'd11;

20'd46:tone<=5'd12;

20'd47:tone<=5'd12;

20'd48:tone<=5'd12;

20'd49:tone<=5'd12;

20'd50:tone<=5'd14;

20'd51:tone<=5'd14;

20'd52:tone<=5'd14;

20'd53:tone<=5'd14;

20'd54:tone<=5'd15;

20'd55:tone<=5'd15;

20'd56:tone<=5'd15;

20'd57:tone<=5'd15;

20'd58:tone<=5'd16;

20'd59:tone<=5'd16;

20'd60:tone<=5'd16;

20'd61:tone<=5'd16;

20'd62:tone<=5'd8;

20'd63:tone<=5'd8;

20'd64:tone<=5'd9;

20'd65:tone<=5'd9;

20'd66:tone<=5'd5;

20'd67:tone<=5'd4;

20'd68:tone<=5'd5;

20'd69:tone<=5'd4;

20'd70:tone<=5'd8;

20'd71:tone<=5'd8;

20'd72:tone<=5'd9;

20'd73:tone<=5'd9;

20'd74:tone<=5'd5;

20'd75:tone<=5'd4;

20'd76:tone<=5'd5;

20'd77:tone<=5'd4;

20'd78:tone<=5'd8;

20'd79:tone<=5'd8;

20'd80:tone<=5'd9;

20'd81:tone<=5'd9;

20'd82:tone<=5'd5;

20'd83:tone<=5'd4;

20'd84:tone<=5'd5;

20'd85:tone<=5'd4;

20'd86:tone<=5'd14;

20'd87:tone<=5'd14;

20'd88:tone<=5'd13;

20'd89:tone<=5'd13;

20'd90:tone<=5'd5;

20'd91:tone<=5'd4;

20'd92:tone<=5'd8;

20'd93:tone<=5'd8;

20'd94:tone<=5'd9;

20'd95:tone<=5'd9;

20'd96:tone<=5'd5;

20'd97:tone<=5'd4;

20'd98:tone<=5'd5;

20'd99:tone<=5'd4;

20'd100:tone<=5'd8;

20'd101:tone<=5'd8;

20'd102:tone<=5'd9;

20'd103:tone<=5'd9;

20'd104:tone<=5'd5;

20'd105:tone<=5'd4;

20'd106:tone<=5'd5;

20'd107:tone<=5'd4;

20'd108:tone<=5'd8;

20'd109:tone<=5'd8;

20'd110:tone<=5'd9;

20'd111:tone<=5'd9;

20'd112:tone<=5'd11;

20'd113:tone<=5'd11;

20'd114:tone<=5'd14;

20'd115:tone<=5'd14;

20'd116:tone<=5'd13;

20'd117:tone<=5'd13;

20'd118:tone<=5'd14;

20'd119:tone<=5'd14;

20'd120:tone<=5'd13;

20'd121:tone<=5'd13;

20'd122:tone<=5'd12;

20'd123:tone<=5'd12;

20'd124:tone<=5'd11;

20'd125:tone<=5'd11;

20'd126:tone<=5'd9;

20'd127:tone<=5'd9;

20'd128:tone<=5'd8;

20'd129:tone<=5'd8;

20'd130:tone<=5'd9;

20'd131:tone<=5'd9;

20'd132:tone<=5'd5;

20'd133:tone<=5'd4;

20'd134:tone<=5'd5;

20'd135:tone<=5'd4;

20'd136:tone<=5'd8;

20'd137:tone<=5'd8;

20'd138:tone<=5'd9;

20'd139:tone<=5'd9;

20'd140:tone<=5'd5;

20'd141:tone<=5'd4;

20'd142:tone<=5'd5;

20'd143:tone<=5'd4;

20'd144:tone<=5'd8;

20'd145:tone<=5'd8;

20'd146:tone<=5'd9;

20'd147:tone<=5'd9;

20'd148:tone<=5'd5;

20'd149:tone<=5'd4;

20'd150:tone<=5'd5;

20'd151:tone<=5'd4;

20'd152:tone<=5'd14;

20'd153:tone<=5'd14;

20'd154:tone<=5'd13;

20'd155:tone<=5'd13;

20'd156:tone<=5'd5;

20'd157:tone<=5'd4;

20'd158:tone<=5'd5;

20'd159:tone<=5'd5;

20'd160:tone<=5'd4;

20'd161:tone<=5'd5;

20'd162:tone<=5'd7;

20'd163:tone<=5'd7;

20'd164:tone<=5'd5;

20'd165:tone<=5'd7;

20'd166:tone<=5'd8;

20'd167:tone<=5'd8;

20'd168:tone<=5'd8;

20'd169:tone<=5'd9;

20'd170:tone<=5'd11;

20'd171:tone<=5'd14;

20'd172:tone<=5'd9;

20'd173:tone<=5'd11;

20'd174:tone<=5'd14;

20'd175:tone<=5'd14;

20'd176:tone<=5'd14;

20'd177:tone<=5'd14;

20'd178:tone<=5'd13;

20'd179:tone<=5'd14;

20'd180:tone<=5'd13;

20'd181:tone<=5'd12;

20'd182:tone<=5'd12;

20'd183:tone<=5'd11;

20'd184:tone<=5'd11;

20'd185:tone<=5'd12;

20'd186:tone<=5'd12;

20'd187:tone<=5'd12;

20'd188:tone<=5'd12;

20'd189:tone<=5'd12;

20'd190:tone<=5'd12;

20'd191:tone<=5'd15;

20'd192:tone<=5'd15;

endcase

end

3'b010://天空之城

begin

if(count>=195)

begin

count<=1'b0;

end

else

begin

count<=count+1'b1;

end

case(count)

20'd0:tone<=5'd12;

20'd1:tone<=5'd13;

20'd2:tone<=5'd14;

20'd3:tone<=5'd14;

20'd4:tone<=5'd14;

20'd5:tone<=5'd13;

20'd6:tone<=5'd14;

20'd7:tone<=5'd14;

20'd8:tone<=5'd16;

20'd9:tone<=5'd16;

20'd10:tone<=5'd13;

20'd11:tone<=5'd13;

20'd12:tone<=5'd13;

20'd13:tone<=5'd13;

20'd14:tone<=5'd13;

20'd15:tone<=5'd13;

20'd16:tone<=5'd9;

20'd17:tone<=5'd9;

20'd18:tone<=5'd12;

20'd19:tone<=5'd12;

20'd20:tone<=5'd12;

20'd21:tone<=5'd11;

20'd22:tone<=5'd12;

20'd23:tone<=5'd12;

20'd24:tone<=5'd7;

20'd25:tone<=5'd7;

20'd26:tone<=5'd11;

20'd27:tone<=5'd11;

20'd28:tone<=5'd11;

20'd29:tone<=5'd11;

20'd30:tone<=5'd11;

20'd31:tone<=5'd11;

20'd32:tone<=5'd9;

20'd33:tone<=5'd9;

20'd34:tone<=5'd10;

20'd35:tone<=5'd10;

20'd36:tone<=5'd10;

20'd37:tone<=5'd9;

20'd38:tone<=5'd10;

20'd39:tone<=5'd14;

20'd40:tone<=5'd14;

20'd41:tone<=5'd14;

20'd42:tone<=5'd9;

20'd43:tone<=5'd9;

20'd44:tone<=5'd9;

20'd45:tone<=5'd14;

20'd46:tone<=5'd14;

20'd47:tone<=5'd14;

20'd48:tone<=5'd13;

20'd49:tone<=5'd13;

20'd50:tone<=5'd13;

20'd51:tone<=5'd10;

20'd52:tone<=5'd10;

20'd53:tone<=5'd10;

20'd54:tone<=5'd13;

20'd55:tone<=5'd13;

20'd56:tone<=5'd13;

20'd57:tone<=5'd13;

20'd58:tone<=5'd13;

20'd59:tone<=5'd13;

20'd60:tone<=5'd13;

20'd61:tone<=5'd13;

20'd62:tone<=5'd12;

20'd63:tone<=5'd13;

20'd64:tone<=5'd14;

20'd65:tone<=5'd14;

20'd66:tone<=5'd14;

20'd67:tone<=5'd20;

20'd68:tone<=5'd14;

20'd69:tone<=5'd14;

20'd70:tone<=5'd16;

20'd71:tone<=5'd16;

20'd72:tone<=5'd13;

20'd73:tone<=5'd13;

20'd74:tone<=5'd13;

20'd75:tone<=5'd9;

20'd76:tone<=5'd9;

20'd77:tone<=5'd12;

20'd78:tone<=5'd12;

20'd79:tone<=5'd12;

20'd80:tone<=5'd11;

20'd81:tone<=5'd12;

20'd82:tone<=5'd12;

20'd83:tone<=5'd7;

20'd84:tone<=5'd7;

20'd85:tone<=5'd11;

20'd86:tone<=5'd11;

20'd87:tone<=5'd11;

20'd88:tone<=5'd11;

20'd89:tone<=5'd11;

20'd90:tone<=5'd11;

20'd91:tone<=5'd9;

20'd92:tone<=5'd9;

20'd93:tone<=5'd10;

20'd94:tone<=5'd10;

20'd95:tone<=5'd14;

20'd96:tone<=5'd13;

20'd97:tone<=5'd13;

20'd98:tone<=5'd13;

20'd99:tone<=5'd14;

20'd100:tone<=5'd14;

20'd101:tone<=5'd15;

20'd102:tone<=5'd15;

20'd103:tone<=5'd16;

20'd104:tone<=5'd14;

20'd105:tone<=5'd14;

20'd106:tone<=5'd14;

20'd107:tone<=5'd14;

20'd108:tone<=5'd14;

20'd109:tone<=5'd14;

20'd110:tone<=5'd13;

20'd111:tone<=5'd12;

20'd112:tone<=5'd12;

20'd113:tone<=5'd13;

20'd114:tone<=5'd13;

20'd115:tone<=5'd11;

20'd116:tone<=5'd11;

20'd117:tone<=5'd12;

20'd118:tone<=5'd12;

20'd119:tone<=5'd12;

20'd120:tone<=5'd12;

20'd121:tone<=5'd12;

20'd122:tone<=5'd12;

20'd123:tone<=5'd7;

20'd124:tone<=5'd7;

20'd125:tone<=5'd15;

20'd126:tone<=5'd16;

20'd127:tone<=5'd16;

20'd128:tone<=5'd16;

20'd129:tone<=5'd15;

20'd130:tone<=5'd16;

20'd131:tone<=5'd16;

20'd132:tone<=5'd18;

20'd133:tone<=5'd18;

20'd134:tone<=5'd15;

20'd135:tone<=5'd15;

20'd136:tone<=5'd15;

20'd137:tone<=5'd15;

20'd138:tone<=5'd15;

20'd139:tone<=5'd15;

20'd140:tone<=5'd11;

20'd141:tone<=5'd11;

20'd142:tone<=5'd14;

20'd143:tone<=5'd14;

20'd144:tone<=5'd14;

20'd145:tone<=5'd13;

20'd146:tone<=5'd14;

20'd147:tone<=5'd14;

20'd148:tone<=5'd16;

20'd149:tone<=5'd16;

20'd150:tone<=5'd16;

20'd151:tone<=5'd16;

20'd152:tone<=5'd16;

20'd153:tone<=5'd16;

20'd154:tone<=5'd16;

20'd155:tone<=5'd16;

20'd156:tone<=5'd12;

20'd157:tone<=5'd12;

20'd158:tone<=5'd13;

20'd159:tone<=5'd13;

20'd160:tone<=5'd14;

20'd161:tone<=5'd14;

20'd162:tone<=5'd14;

20'd163:tone<=5'd14;

20'd164:tone<=5'd13;

20'd165:tone<=5'd13;

20'd166:tone<=5'd14;

20'd167:tone<=5'd14;

20'd168:tone<=5'd15;

20'd169:tone<=5'd15;

20'd170:tone<=5'd14;

20'd171:tone<=5'd14;

20'd172:tone<=5'd14;

20'd173:tone<=5'd11;

20'd174:tone<=5'd11;

20'd175:tone<=5'd11;

20'd176:tone<=5'd11;

20'd177:tone<=5'd11;

20'd178:tone<=5'd11;

20'd179:tone<=5'd17;

20'd180:tone<=5'd17;

20'd181:tone<=5'd16;

20'd182:tone<=5'd16;

20'd183:tone<=5'd15;

20'd184:tone<=5'd15;

20'd185:tone<=5'd15;

20'd186:tone<=5'd14;

20'd187:tone<=5'd14;

20'd188:tone<=5'd16;

20'd189:tone<=5'd16;

20'd190:tone<=5'd16;

20'd191:tone<=5'd16;

20'd192:tone<=5'd16;

20'd193:tone<=5'd16;

20'd194:tone<=5'd16;

20'd195:tone<=5'd16;

endcase

end

// 3'b001:

default:

begin

count<=1'b0;

tone<=5'd22;

end

endcase

end

else

begin

count<=1'b0;

tone<=5'd22;

end

end

endmodule

### 数字显示模块

`timescale 1ns / 1ps

module scan\_led\_hex\_disp(

input clk,

input reset,

input [4:0] key\_value,

// input [1:0] hex1,hex0,

// input [1:0] dp\_in,

output reg [1:0] an,

output reg [7:0] sseg

);

reg [4:0] hex1,hex0;

reg [1:0] dp\_in;

reg [17:0] regN;

reg [4:0] hex\_in;

reg dp;

always @(posedge clk)

begin

case(key\_value)

5'd0:

begin

hex1<=4'ha;

hex0<=4'd1;

dp\_in<=2'b10;

end

5'd1:

begin

hex1<=4'ha;

hex0<=4'd2;

dp\_in<=2'b10;

end

5'd2:

begin

hex1<=4'ha;

hex0<=4'd3;

dp\_in<=2'b10;

end

5'd3:

begin

hex1<=4'ha;

hex0<=4'd4;

dp\_in<=2'b10;

end

5'd4:

begin

hex1<=4'ha;

hex0<=4'd5;

dp\_in<=2'b10;

end

5'd5:

begin

hex1<=4'ha;

hex0<=4'd6;

dp\_in<=2'b10;

end

5'd6:

begin

hex1<=4'ha;

hex0<=4'd7;

dp\_in<=2'b10;

end

5'd7:

begin

hex1<=4'hb;

hex0<=4'd1;

dp\_in<=2'b10;

end

5'd8:

begin

hex1<=4'hb;

hex0<=4'd2;

dp\_in<=2'b10;

end

5'd9:

begin

hex1<=4'hb;

hex0<=4'd3;

dp\_in<=2'b10;

end

5'd10:

begin

hex1<=4'hb;

hex0<=4'd4;

dp\_in<=2'b10;

end

5'd11:

begin

hex1<=4'hb;

hex0<=4'd5;

dp\_in<=2'b10;

end

5'd12:

begin

hex1<=4'hb;

hex0<=4'd6;

dp\_in<=2'b10;

end

5'd13:

begin

hex1<=4'hb;

hex0<=4'd7;

dp\_in<=2'b10;

end

5'd14:

begin

hex1<=4'hc;

hex0<=4'd1;

dp\_in<=2'b10;

end

5'd15:

begin

hex1<=4'hc;

hex0<=4'd2;

dp\_in<=2'b10;

end

5'd16:

begin

hex1<=4'hc;

hex0<=4'd3;

dp\_in<=2'b10;

end

5'd17:

begin

hex1<=4'hc;

hex0<=4'd4;

dp\_in<=2'b10;

end

5'd18:

begin

hex1<=4'hc;

hex0<=4'd5;

dp\_in<=2'b10;

end

5'd19:

begin

hex1<=4'hc;

hex0<=4'd6;

dp\_in<=2'b10;

end

5'd20:

begin

hex1<=4'hc;

hex0<=4'd7;

dp\_in<=2'b10;

end

default:

begin

hex1<=5'd22;

hex0<=5'd22;

dp\_in<=2'b00;

end

endcase

end

always @(posedge clk or posedge reset)

if(reset==0)

regN <= 0;

else

regN <= regN +1'b1;

always @ (posedge clk)

begin

case(regN[16])

1'b0 :

begin

an <= 2'b01;

hex\_in <= hex0;

dp<=dp\_in[0];

end

default :

begin

an <= 2'b10;

hex\_in <= hex1;

dp<=dp\_in[1];

end

endcase

end

always @ (posedge clk)

begin

case(hex\_in)

5'h0:sseg[6:0] <= 7'b1111110;

5'h1:sseg[6:0] <= 7'b0110000;

5'h2:sseg[6:0] <= 7'b1101101;

5'h3:sseg[6:0] <= 7'b1111001;

5'h4:sseg[6:0] <= 7'b0110011;

5'h5:sseg[6:0] <= 7'b1011011;

5'h6:sseg[6:0] <= 7'b1011111;

5'h7:sseg[6:0] <= 7'b1110000;

5'h8:sseg[6:0] <= 7'b1111111;

5'h9:sseg[6:0] <= 7'b1111011;

5'ha:sseg[6:0] <= 7'b1110111;

5'hb:sseg[6:0] <= 7'b0011111;

5'hc:sseg[6:0] <= 7'b1001110;

5'hd:sseg[6:0] <= 7'b0111101;

5'he:sseg[6:0] <= 7'b1001111;

5'hf:sseg[6:0] <= 7'b1000111;

default:sseg[6:0] <= 7'b0000000;

endcase

sseg[7] <= dp;

end

endmodule

## 约束文件

### top1.xdc

set\_property IOSTANDARD LVCMOS33 [get\_ports {col[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {col[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {col[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {col[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led[15]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led[14]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led[13]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led[12]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led[11]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led[10]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led[9]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led[8]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led[7]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led[6]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led[5]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led[4]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {led[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {scan[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {scan[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {scan[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {scan[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports sys\_clk]

set\_property IOSTANDARD LVCMOS33 [get\_ports audio\_out]

set\_property IOSTANDARD LVCMOS33 [get\_ports SD]

set\_property IOSTANDARD LVCMOS33 [get\_ports sd]

set\_property PULLUP true [get\_ports {col[3]}]

set\_property PULLUP true [get\_ports {col[2]}]

set\_property PULLUP true [get\_ports {col[1]}]

set\_property PULLUP true [get\_ports {col[0]}]

set\_property PACKAGE\_PIN G14 [get\_ports {col[0]}]

set\_property PACKAGE\_PIN D17 [get\_ports {col[1]}]

set\_property PACKAGE\_PIN J13 [get\_ports {col[2]}]

set\_property PACKAGE\_PIN G17 [get\_ports {col[3]}]

set\_property PACKAGE\_PIN F6 [get\_ports {led[0]}]

set\_property PACKAGE\_PIN G4 [get\_ports {led[1]}]

set\_property PACKAGE\_PIN G3 [get\_ports {led[2]}]

set\_property PACKAGE\_PIN J4 [get\_ports {led[3]}]

set\_property PACKAGE\_PIN H4 [get\_ports {led[4]}]

set\_property PACKAGE\_PIN J3 [get\_ports {led[5]}]

set\_property PACKAGE\_PIN J2 [get\_ports {led[6]}]

set\_property PACKAGE\_PIN K2 [get\_ports {led[7]}]

set\_property PACKAGE\_PIN K1 [get\_ports {led[8]}]

set\_property PACKAGE\_PIN H6 [get\_ports {led[9]}]

set\_property PACKAGE\_PIN H5 [get\_ports {led[10]}]

set\_property PACKAGE\_PIN J5 [get\_ports {led[11]}]

set\_property PACKAGE\_PIN K6 [get\_ports {led[12]}]

set\_property PACKAGE\_PIN L1 [get\_ports {led[13]}]

set\_property PACKAGE\_PIN M1 [get\_ports {led[14]}]

set\_property PACKAGE\_PIN K3 [get\_ports {led[15]}]

set\_property PACKAGE\_PIN E16 [get\_ports {scan[0]}]

set\_property PACKAGE\_PIN C15 [get\_ports {scan[1]}]

set\_property PACKAGE\_PIN G16 [get\_ports {scan[2]}]

set\_property PACKAGE\_PIN F16 [get\_ports {scan[3]}]

set\_property PACKAGE\_PIN P17 [get\_ports sys\_clk]

set\_property PACKAGE\_PIN T1 [get\_ports audio\_out]

set\_property PACKAGE\_PIN T5 [get\_ports SD]

set\_property PACKAGE\_PIN M6 [get\_ports sd]

set\_property IOSTANDARD LVCMOS33 [get\_ports {song\_num[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {song\_num[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {song\_num[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports auto\_en]

set\_property PACKAGE\_PIN R1 [get\_ports auto\_en]

set\_property PACKAGE\_PIN P5 [get\_ports {song\_num[2]}]

set\_property PACKAGE\_PIN P4 [get\_ports {song\_num[1]}]

set\_property PACKAGE\_PIN P3 [get\_ports {song\_num[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports keyboard\_select]

set\_property PACKAGE\_PIN U3 [get\_ports keyboard\_select]

set\_property PULLUP true [get\_ports ps2\_clk]

set\_property PULLUP true [get\_ports ps2\_data]

set\_property IOSTANDARD LVCMOS33 [get\_ports {digital\_en[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {digital\_en[0]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {digital\_data[7]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {digital\_data[6]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {digital\_data[5]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {digital\_data[4]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {digital\_data[3]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {digital\_data[2]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {digital\_data[1]}]

set\_property IOSTANDARD LVCMOS33 [get\_ports {digital\_data[0]}]

set\_property PACKAGE\_PIN G1 [get\_ports {digital\_en[1]}]

set\_property PACKAGE\_PIN H2 [get\_ports {digital\_data[7]}]

set\_property PACKAGE\_PIN D4 [get\_ports {digital\_data[6]}]

set\_property PACKAGE\_PIN E3 [get\_ports {digital\_data[5]}]

set\_property PACKAGE\_PIN D3 [get\_ports {digital\_data[4]}]

set\_property PACKAGE\_PIN F4 [get\_ports {digital\_data[3]}]

set\_property PACKAGE\_PIN F3 [get\_ports {digital\_data[2]}]

set\_property PACKAGE\_PIN E2 [get\_ports {digital\_data[1]}]

set\_property PACKAGE\_PIN D2 [get\_ports {digital\_data[0]}]

set\_property PACKAGE\_PIN F1 [get\_ports {digital\_en[0]}]

### port\_constrains.xdc

set\_property IOSTANDARD LVCMOS33 [get\_ports audio\_out]

set\_property IOSTANDARD LVCMOS33 [get\_ports clk]

set\_property IOSTANDARD LVCMOS33 [get\_ports ps2\_clk]

set\_property IOSTANDARD LVCMOS33 [get\_ports ps2\_data]

set\_property IOSTANDARD LVCMOS33 [get\_ports rst]

set\_property IOSTANDARD LVCMOS33 [get\_ports SD]

set\_property IOSTANDARD LVCMOS33 [get\_ports sd]

set\_property PACKAGE\_PIN T1 [get\_ports audio\_out]

set\_property PACKAGE\_PIN P17 [get\_ports clk]

set\_property PACKAGE\_PIN K5 [get\_ports ps2\_clk]

set\_property PACKAGE\_PIN L4 [get\_ports ps2\_data]

set\_property PACKAGE\_PIN P15 [get\_ports rst]

set\_property PACKAGE\_PIN T5 [get\_ports SD]

set\_property PACKAGE\_PIN M6 [get\_ports sd]