《Digital Design》Project Design Report

Name： 叶璨铭　　SID：12011404

目录

> 本文档从我的电子琴project的功能出发，到系统流程的设计、模块与类的设计、工具模块与工具类的选择， 自顶向下地介绍了本project的设计流程、设计思想、设计方法。

Topic: Design and finish an Electronic Piano that can play music on EGO1.

System Function

基本功能

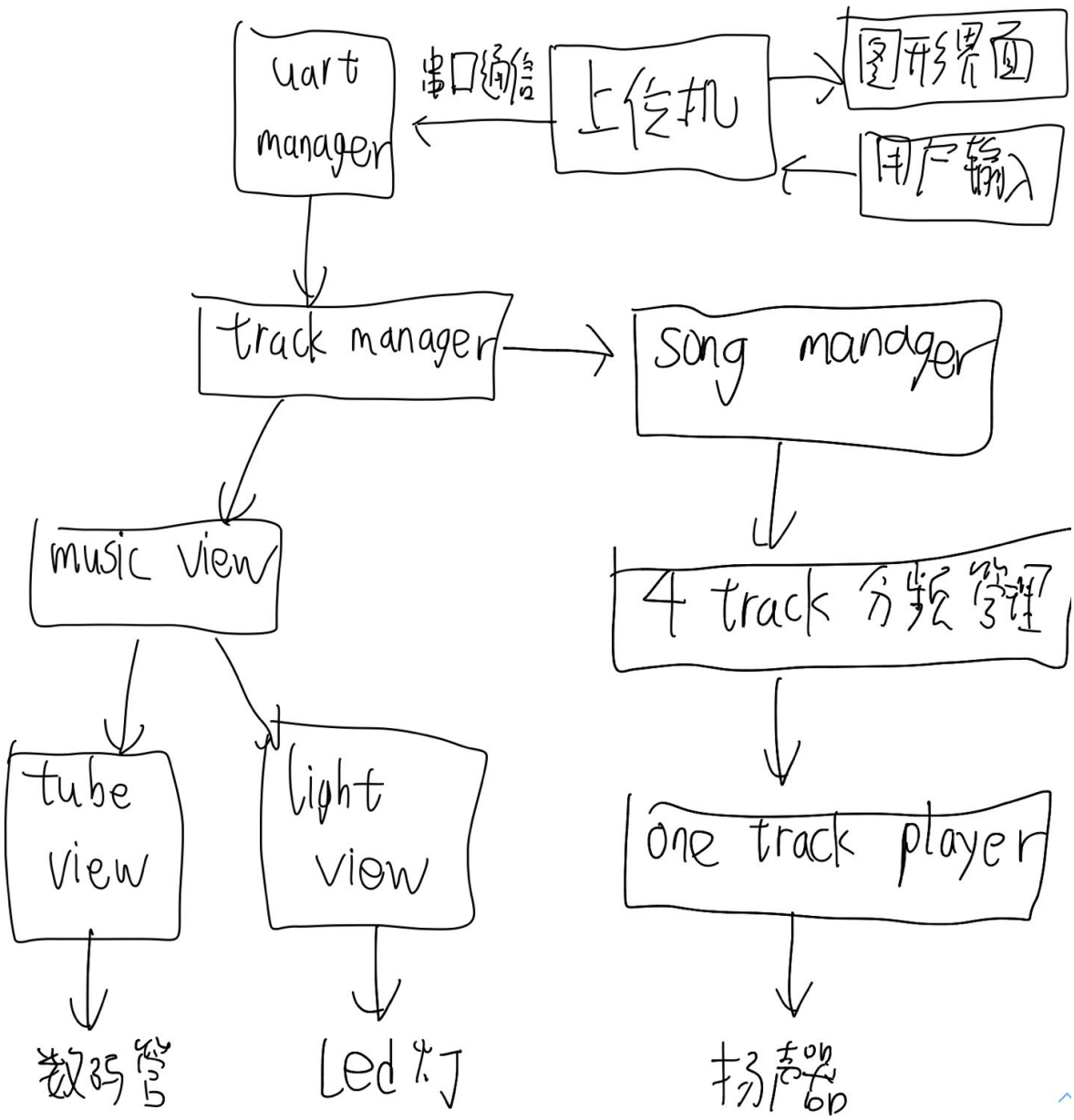
- 使用**键盘**作为输入单元。当按下指定键位的时候，EGO1开发板的**扬声器**像一台钢琴一样，演奏出优美的乐曲。
- EGO1开发版的**七段数码管**和**led灯**显示音符信息。

亮点功能

- 众所周知，**钢琴可以同时演奏多个音符**，而不像长笛、小提琴、黑管、萨克斯...等单音符演奏乐器那样同时只能演奏一个音符。既然我们模拟的是电子琴，而不是电子长笛、电子小提琴...，所以本project**尝试实现可以同时演奏多个音符的电子琴。**
- 钢琴之所以被称为乐器之王，是因为其音域宽广，**有黑白键。**本project除了支持高中低三个八度21个音符之外，还支持其中的半音，**总计3个八度36个半音被支持。**
    - 同时，设计合理的键位，让这些音符能够被准确、容易地被弹奏。
    - 因此，**七段数码管**和**led灯**的音符显示需要更加复杂。
- 建立**上位机-下位机串口通信**.
    - 上位机使用Java设计一个电子琴小游戏，通过usb串口连接EGO1开发版。
    - 使得任何连接笔记本电脑的键盘， 以及鼠标点击、网络联机传输的信息等**多种渠道，都可以控制EGO1开发版发出音乐。**
    - 让电子琴不仅仅是电子琴，还可以通过在**电脑端存储串口指令，形成音乐存档**，让电子琴变成音乐播放器，覆盖另一个project可以实现的所有功能
        - 可以通过上位机串口指令发送的速度来快进、减速、暂停。
        - 还可以把音乐存档发送给朋友一起分享你的演奏。不局限于上一首、下一首。
    - 更重要的是，网上有大量的**电子乐器通用乐谱格式(midi)**文件。
        - 即使project展示的时候不会弹钢琴， 也可以通过扩展上位机的程序，
        - 让上位机解析midi文件，形成串口指令，发送给下位机。
        - 因此本project具有**很好的扩展性。**
    - 如果不是EGO1开发版，而是其他开发版或者发声设备。
        - 本project设计的上位机和通信编码依然适用
        - 下位机只需要将Verilog代码稍作修改，就可以变成另一个开发版的电子琴下位机电路代码。
        - 因此**具有一定的可移植性与跨平台性。**

System Design——系统设计

系统流程图



下位机的Verilog设计层次：

1.顶层模块

输入

- EGO1_Clock, EGO1_Reset 系统时钟、系统重置（低有效）
- EGO1_Uart_fromPC 串口输入

输出

- EGO1_Uart_toPC 串口输出
- EGO1_Audio_SD， EGO1_Audio_PWM 音频输出
- EGO1_DigitalTubes_Enable [7:0] 七段数码管组的位选
- EGO1_DigitalTube[15:0] 七段数码管的左右段选。

功能概述

- 例化**音轨管理器模块**， 获得4个音轨临时变量。
- 例化**声音驱动模块**， 根据获得的音轨，转化为声音信号输出。
- 例化**音符显示模块**， 根据获得音轨的信息，转化为七段数码管的信号输出。
- 对**变量名进行重定向**，以让xdc约束文件可以映射变量名到管脚。

1.1 音轨管理器模块

输入

- EGO1_Clock, EGO1_Reset 系统时钟、系统重置（低有效）
- EGO1_Uart_fromPC 串口输入

输出

- EGO1_Uart_toPC 串口输出
- [5:0] track0、track1、track2、track3 四个音轨的音符信息

功能概述

- 例化**串口通信模块**
- **解算串口信息，形成音轨信息。**

1.2 声音驱动模块

输入

- [5:0] track0、track1、track2、track3 四个音轨的音符信息

输出

- speaker 扬声器输出

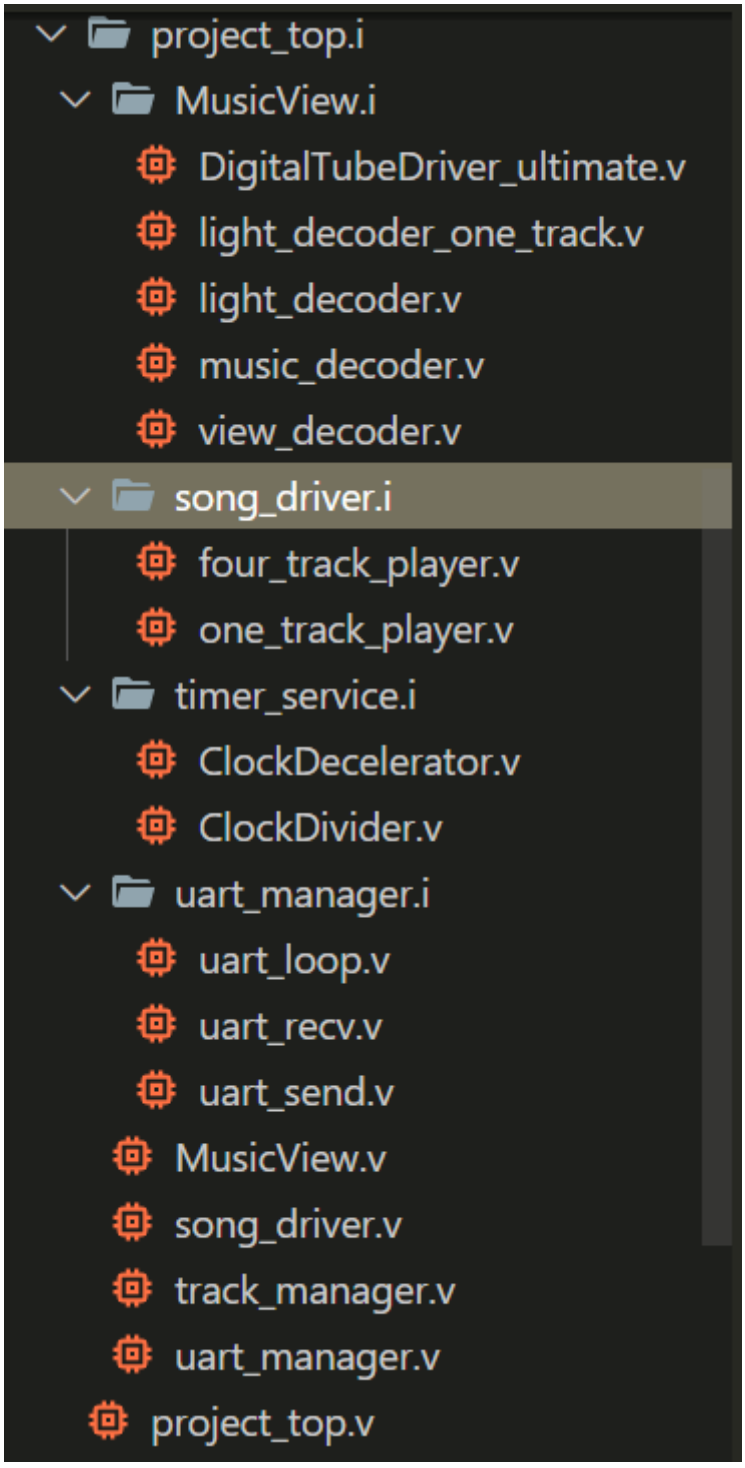
功能概述

- 例化**单音驱动模块**
    - 单音驱动模块能够在其分配到的时间片内，根据声音来形成频率方波信号。
- 根据音轨，进行分频，为单音驱动模块分配时间片
- 我们根据乐理了解到，要完成大部分乐曲，至少需要支持16分音符。
    - 因此，我们分频出一个16Hz的时钟，然后再16Hz的基础上，进行混音分频。

1.3音符显示模块

- 四个音轨分别显示两位

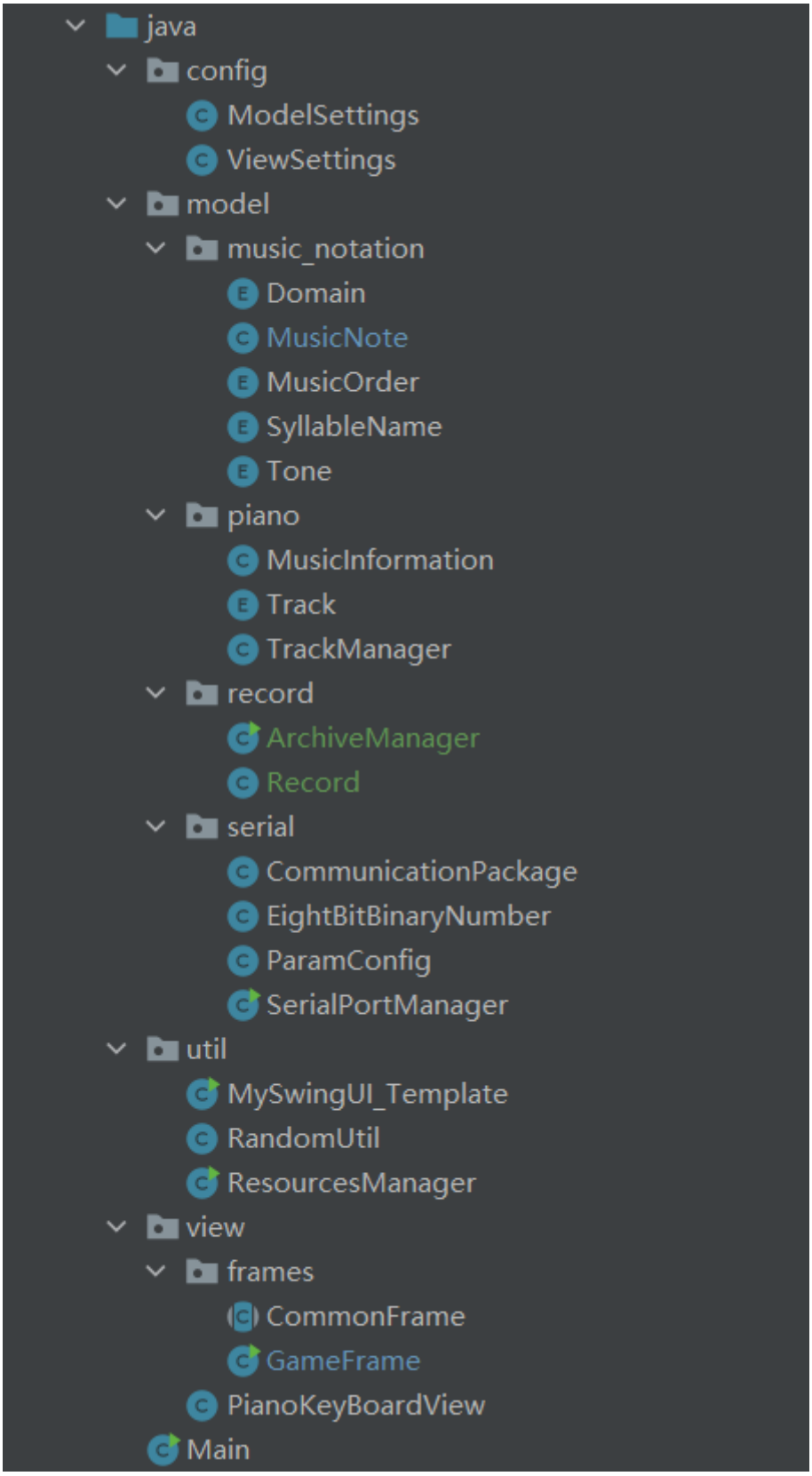
下位机项目结构图



其中，.v文件时一个Verilog模块，而和.v文件在一起的，还有一个.i文件夹。而.i文件夹中又有其他的.v文件。

他们的关系是：.v文件依赖于.i文件才能运行，表示依赖关系。

上位机的Java代码设计



使用经典的model、view设计方法，利用计算机程序设计基础A课程中我积累的swing游戏开发代码、工具类，对游戏进行快速开发。

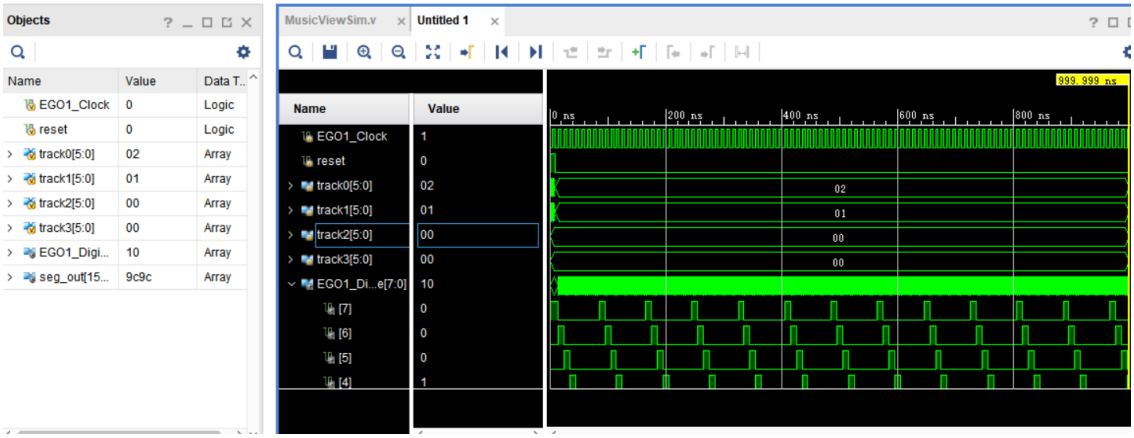
针对这次project，

- 首先，我们设计了**串口通信类**，和父类**串口通信包。**通过继承父类的串口通信包，我们定义了音乐发送信息类。
- 其次，根据乐理，我们**对音乐进行了抽象。**形成了
    - Domain 高中低音
    - MusicOrder 根据十二平均律对音符的位置进行抽象

    - Tone 音调，升调、降调、不变。
    - SyllableName 音名，表示7音。
    - MusicNote：音轨中存储的音乐信息，根据Domain、SyllableName和Tone进行唯一确定（构造）。
- 有了乐理类的抽象之后，我们**通过java自动计算分频比和预置数，自动形成Verilog代码**：
    -
```
class MusicNote_verilog {
    public static void main(String[] args) {
        for (Domain d:Domain.values()) {
            for (int i = 1; i <= 12; i++) {
                final MusicNote musicNote = new MusicNote(d, MusicOrder.musicOfOrder(i));
                System.out.println("\t\t\t"+musicNote.getBinaryCode()+":\torigin <= "+
                        Math.round((16383 - 3000000/musicNote.getFrequency()))+";"
                );
            }
        }
    }
}
```

System Design——系统仿真测试

- 对MusicView的仿真测试



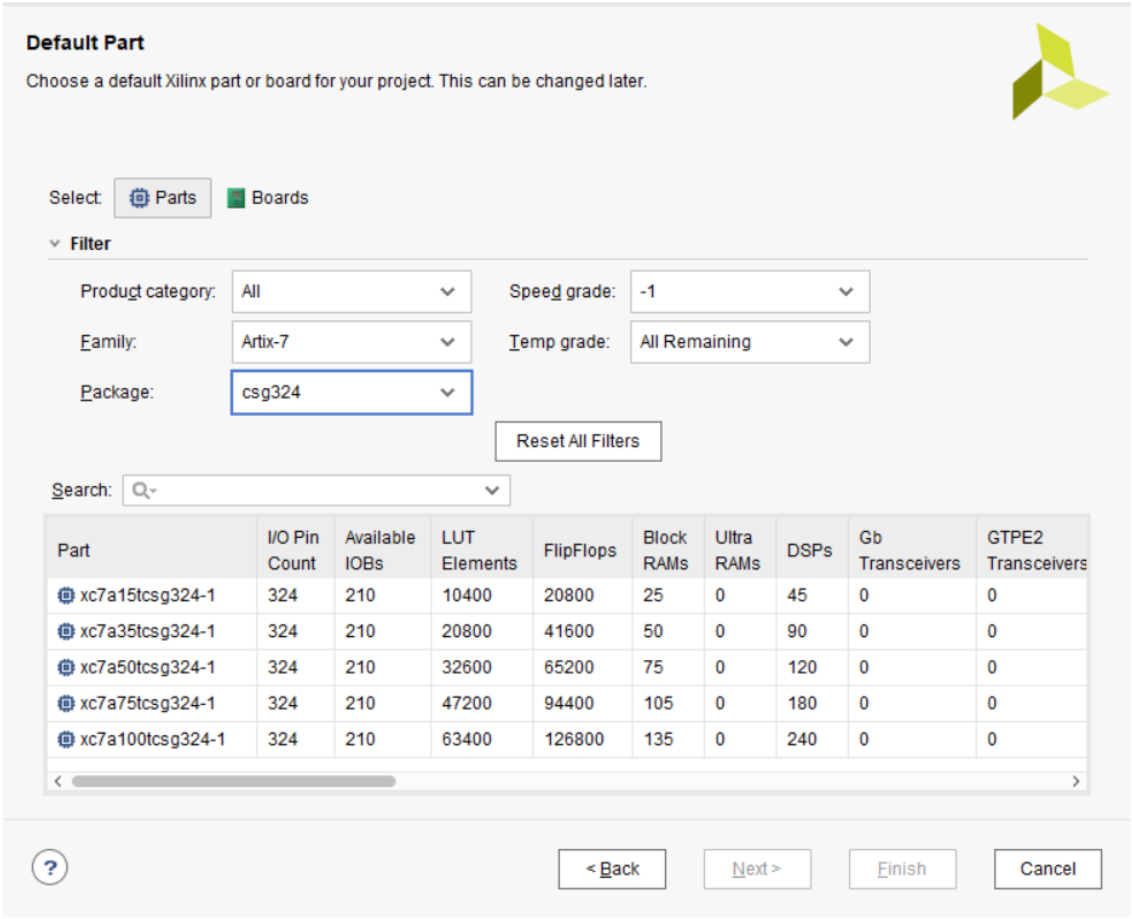
- 由于MusicView出现了问题，我们执行仿真调试，经过波形，发现音符编码解算没有错误（虽然代码比较复杂），而分频有问题。
- 其他模块demo是对的，而且和其他模块无关，因此不需要测试，只要保持代码不变即可。

Problems Encountered and Solutions

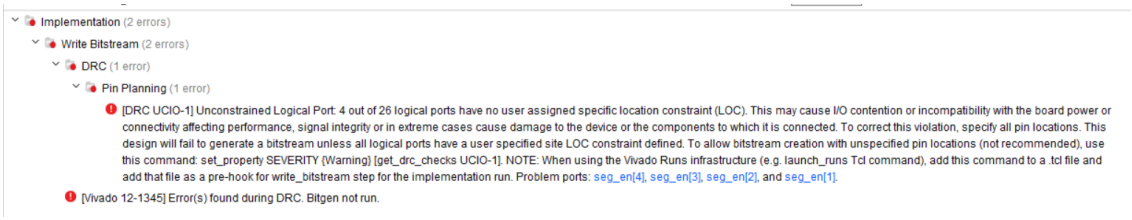
1.如何建立舒适的模块化设计方法论?

使用Vivado软件配合Verilog设计，是数字逻辑这门课程实验部分的基本方法。然而，在lab课的实践当中，出现以下问题：

- 创建项目时，每次都要手动选择开发版的信息，有时候还会选错。

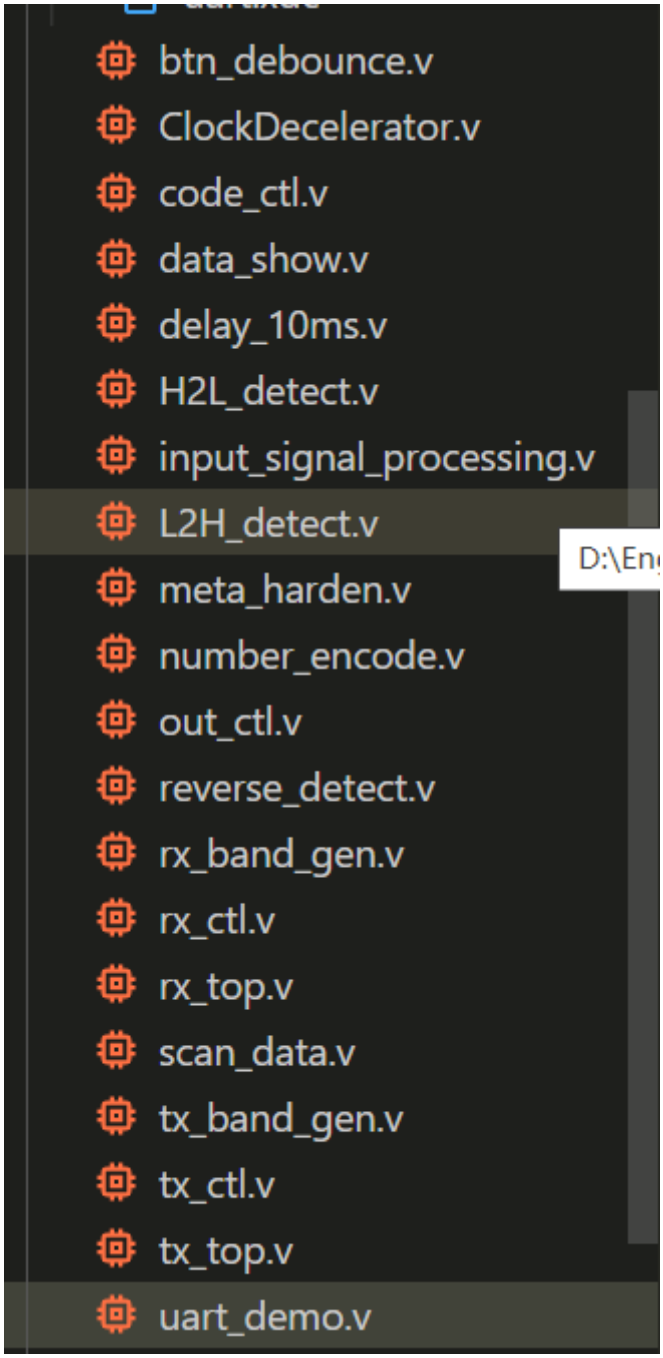


- 每新增一个变量，就需要重新绑定xdc文件，而xdc文件到底是如何描述的、局部修改的影响、变量是否能够绑定成功（比如clk），这些行为都是不确定的。导致经常出现编译了很久，但是还是失败了。



- 当一个代码文件出现很多模块、很多always时，难以理解代码的功能。局部的修改可能导致整个功能崩溃，而且不知道是为什么而崩溃。
- 即使是多个文件，如果文件模块之间的依赖关系过于复杂，虽然vivado可以显示出其树状结构，但是用VSCode编辑时，文件没有树状结构，会导致无法区分文件的功能层次。

  例如，这是EGO1学生开发包中，串口通信的示例代码。

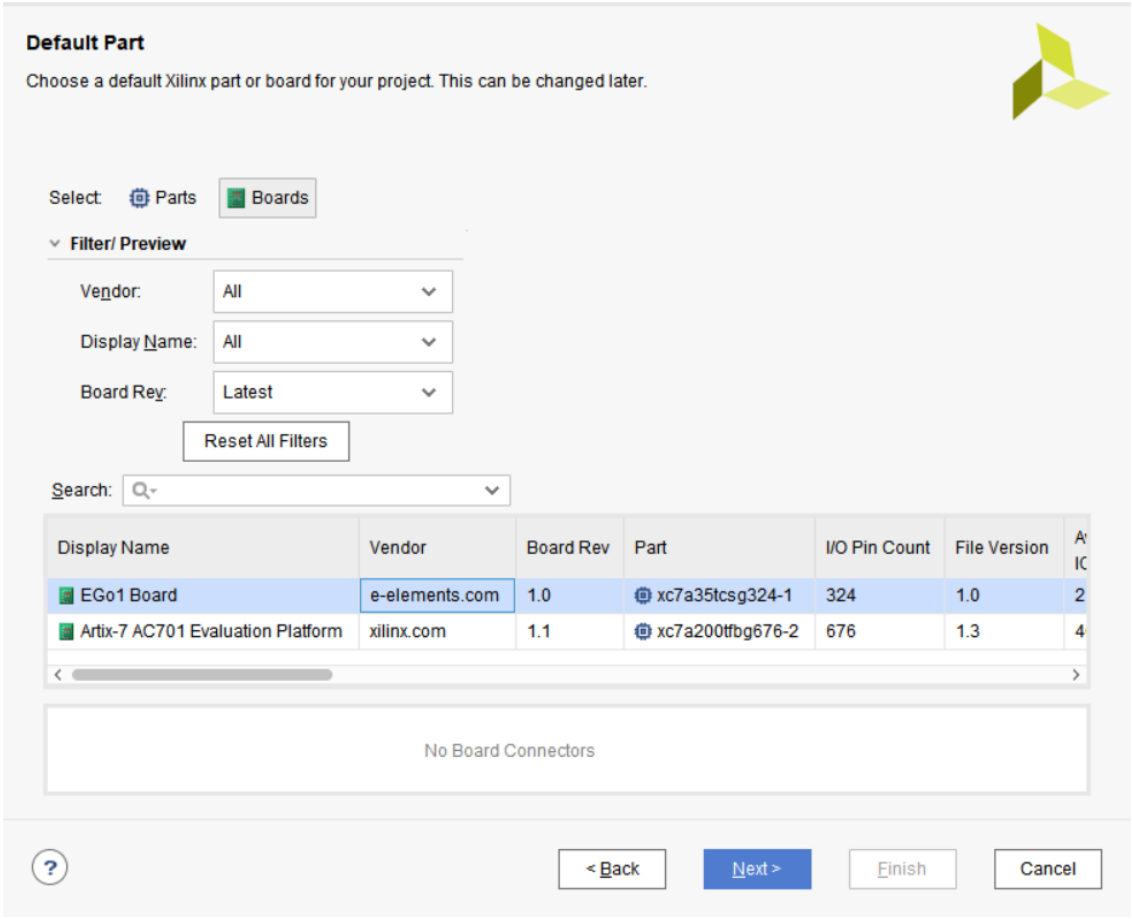


可以看到，该代码复杂难懂，层次非常复杂，没有对外提供清晰的接口。

为了让project的开发更加高效，解决之前lab课中写课堂作业出现的以上问题，对于代码规范进行规定，对工具进行升级。如下：

- 使用EGO1学生开发包中提供的板卡文件，加入到vivado中。

效果：



- 解决xdc问题。
  - 经过研究，vivado对于一个项目，可以有很多组xdc文件夹，但是只能有一组的top的状态。
    - 而每组xdc文件夹，可以有多个文件。因此，我们应该按照不同的绑定功能，把xdc约束语句分到不同名字的文件。
  - 经过研究，xdc有一个**变量名映射端口与电压的字典语法**，比如

```
1  ////////////////////////////串口//////////////////////////////////////////
2  set_property -dict {PACKAGE_PIN N5 IOSTANDARD LVCMOS33} [get_ports EGO1_Uart_fromPC]
3  set_property -dict {PACKAGE_PIN T4 IOSTANDARD LVCMOS33} [get_ports EGO1_Uart_toPC]
```

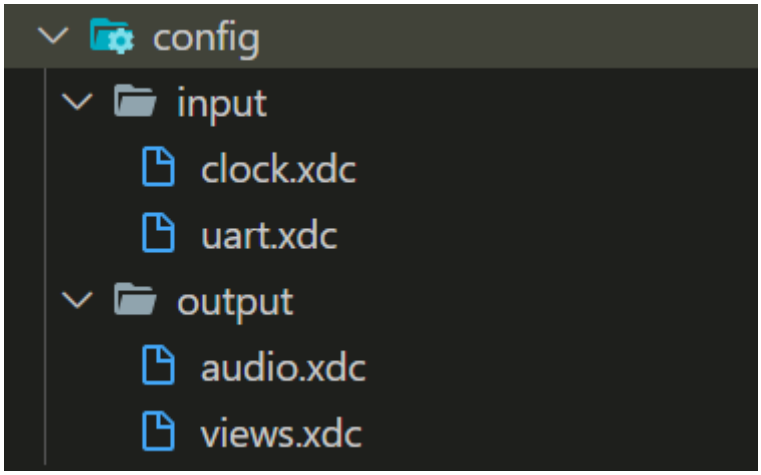
    这样，**当我们在程序中用变量名 "EGO1_Uart_fromPC"时，xdc就能自动帮我们绑定端口**，

    我们只要提前把这个文件写好，就能避免反复、无效、无聊的端口绑定与电压选择工作。

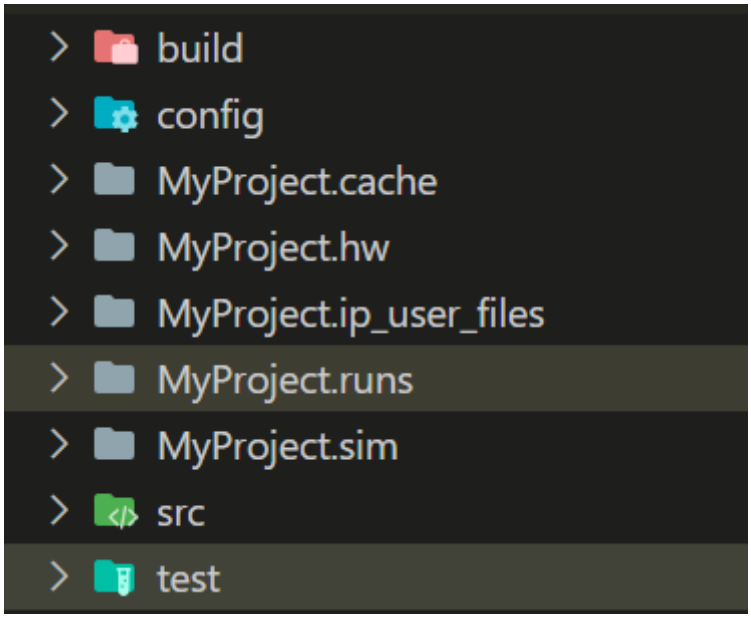
    因为管脚绑定从思想上就是只和一个开发版的信息有关，而与代码的变化、项目的变化无关。

    只要开发版确定了，开发版什么管脚是什么功能，就必然是确定的，而不需要反复的确认。

    如果我们写新的项目，都重新绑定，假如这个项目不是电子琴，而是更加复杂的项目，比如CPU，那么我们的手动绑定不仅时间长、而且出错了不好找bug。
  - 根据以上研究，建立NoXDC工具包，将EGO1所有端口分类绑定。
  - 最终，根据本项目中使用到的EGO1功能，我们从工具包取出我们需要的xdc文件：

- 

- 项目结构的设计：
  - 经过研究，我们拒绝使用vivado的create file功能，而是在项目文件的外边建立以下四个文件夹



    其中，build是生成的.bit文件，config是xdc约束文件，src是Verilog源码， test是Verilog仿真源码
  - 这样，我们就不会把源码和Vivado为了合成、仿真而生成的脚本文件、配置文件搞混淆，而且我们可以自己定义源码的结构
  - 当我们需要Vivado进行合成的时候，我们只需要使用add Directory功能，将这四个文件夹加入到vivado的项目认知当中，vivado就能自动识别。

2.如何实现串口通信?

下位机遇到的问题

- 首先，我们尝试使用EGO1开发包中串口控制器代码
  -

    - demo成功运行。
    - 那么，如何把官方给的串口代码转换成我的音轨控制代码呢?
    - 经过代码分析，官方给的示例非常复杂，但是功能与我的期望不符
        - 官方代码让ego1通过按钮来接受一次串口信息，然而，我们需要实时接收上位机的音符信息。
        - 但是不能简单地把按钮信号改为时钟信号，因为
        - 官方代码为了按钮，设计减震、信号处理等复杂的流程，严重干扰了我抽取功能的能力。
        - 官方代码为了让按钮能按照时序收到信息，设计了缓存，对缓存的管理、对信号的分析，官方代码异常复杂，难以找到最初串口得到的到底是什么信息，是用什么做的。
- 因此我们决定弃用官方代码。
    - 经过了解，领航者ZYNQ的开发版文档成熟、教程详细。
    - 我们下载了其文档，果然，里面对串口通信有着详细的介绍。

送到 uart_send 模块，并通过 send_en 接口给出一个上升沿，以启动发送过程。

在编写代码之前，我们首先要确定串口通信的数据格式及波特率。在这里我们选择串口比较常用的一种模式，数据位为 8 位，停止位为 1 位，无校验位，波特率为 115200bps。则传输一帧数据的时序图如下图所示：

图 14.4.3 串口通信时序图

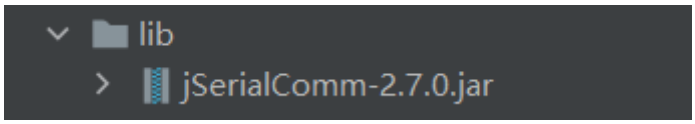
顶层模块的代码如下：

```
1  module uart_loopback_top(
2      input           sys_clk,          //外部 50M 时钟
3      input           sys_rst_n,        //外部复位信号，低有效
4
5      input           uart_rxd,         //UART 接收端口
6      output          uart_txd          //UART 发送端口
7      );
8
9  //parameter define
10 parameter  CLK_FREQ = 50000000;       //定义系统时钟频率
11 parameter  UART_BPS = 115200;         //定义串口波特率
12
13 //wire define
14 wire       uart_recv_done;           //UART 接收完成
15 wire [7:0] uart_recv_data;           //UART 接收数据
16 wire       uart_send_en;             //UART 发送使能
17 wire [7:0] uart_send_data;           //UART 发送数据
```

292

- 该代码高内聚、低耦合，文档详细，容易修改
- 虽然这个开发版和我的EGO1开发版不一样，比如时钟是50M的
- 但是我很容易就能把它的代码改成我的代码，因为parameter定义清晰、明白。输入输出和EGO1的输入输出完全类似。
- 最终，我们解决了问题。我的下位机成功接受了串口调试工具的信息。

上位机遇到的问题

- 经过百度搜索得到，RXTX是一个常用的Java串口通信jar包。
  - 然而，经过实际运行，我确认了RXTX无法在Win10上运行。RXTX的开发者已经停止了对其的更新支持。
- 经过询问对串口通信比较熟悉的同学，我找到了另外一个Java工具jar包，JSerialComm。 运行了简单的Demo之后，成功连接了EGO1开发版。
- 

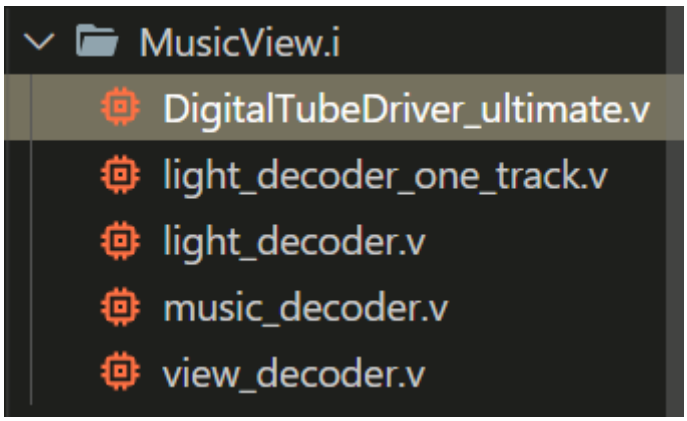
3. 如何设计上下位机通信编码

由于串口通信的特点是每次发送一个字节（byte），八个位（bit），我进行了如下设计

- 我们需要支持36个音符（包括半音），所以，我们至少需要6位来描述音符。
- 还剩两位，因此，我们决定支持四个音轨，用前两位表示音轨的编号。
- 注意，自动分配音轨是一件较为复杂的计算任务，因此我们在上位机计算完成后再传递给下位机，
- 这样，下位机只需要考虑如何把四个音轨的音符，物理实现播放成功。

4.MusicView无法在数码管上正确显示。

- 一开始，我把以前lab中数码管的代码复制到MusicView上面，同时MusicView负责音符解算任务。
  - 然而，片选有问题，只能显示一个。
- 于是我把代码分开成几个更加单元化的模块，每个模块单独测试。
  -



- DigitalTubeDriver_ultimate.v

```
module DigitalTubeDriver_ultimate ( //右边开始编号
    input EGO1_Clock, reset,
    output[7:0] EGO1_DigitalTubes_Enable,
    output[15:0] EGO1_DigitalTube,

    input[7:0] text7,
    input[7:0] text6,
    input[7:0] text5,
    input[7:0] text4,
    input[7:0] text3,
    input[7:0] text2,
    input[7:0] text1,
    input[7:0] text0
);
```

七段数码管的终极解决方案，只要输入8个管的字型，就能够让8个管分别显示自己该显示的东西。

- music_decoder.v

  音乐解码器，把音符信息计算为音符符号编码

  （比如，高音C和低音C，虽然音符信息不一样，但是要编码成C这个符号）

- view_decoder.v

  字型解码器，把音符编码计算为字型。

  比如，字母C计算为8'b1001_1100

  升调符号#难以显示，我们用-|代替，8'b0110_0010

- 经过分模块测试，我们确认了问题在DigitalTubeDriver（通过仿真波形，和实际测试）
  - 原来，分频例化时，传入的时钟参数错误。
  - 之前reset的方向也是错误的。
- 修正错误后，我们把一切分频、计数操作封好，命名为ultimate版本，从此只需要这个模块即可。

5.java如何绑定键盘到一个串口信息

键盘监听问题

- 一开始我打算使用java swing的addKeyListener方法，然而不能工作。
- 了解了gui的知识之后，原来是因为焦点在别的组件，而java的这个方法只能监听某一个组件的键盘，不能监听所有键盘
- 因此，我们使用java键盘映射方法，将绑定级别设置为WHEN_IN_FOCUSED_WINDOW
- 这样只需要在窗口中按下按键，就可以

键盘码问题

- java提供了抽象，只需要电脑中输入了一个字符，就可以知道其键盘码。
- 然而，这个抽象对我们的project是无效的。
- 我们需要在键盘上弹奏，只要我们的按键按下了，音符就应该确定，而与输入法、大小写无关
- 那么，我们应该使用通用的键盘码进行绑定，而不是根据字符输入来绑定。
- 但是java的抽象还有一点是对我们有效的，就是Ctrl Mask， Shift Mask
    - 因为我们需要设计ctrl降音调，shift升音调。

按下与释放问题

- 一开始，我认为，只要按下了就播放，不按下就不播放，不需要检测键盘的断码
- 然而，实际project出现了长按一个音，但是音先出现、断开、再出现的情况
- 了解了键盘原理之后，我反现，原来长按时，需要判断是否有一段时间没有释放，才会连续认为键盘按下
- 因此，有必要通过检测断码。

最终我们形成了的代码绑定设置十分简单

```
registerPianoBoardListener( keyName: "3",          Domain.HIGH, SyllableName.Mi);
registerPianoBoardListener( keyName: "P",          Domain.HIGH, SyllableName.Mi);
registerPianoBoardListener( keyName: "ADD",        Domain.HIGH, SyllableName.Mi);

registerPianoBoardListener( keyName: "4",          Domain.HIGH, SyllableName.Fa);
registerPianoBoardListener( keyCode: 219,         Domain.HIGH, SyllableName.Fa);
registerPianoBoardListener( keyName: "NUM_LOCK",   Domain.HIGH, SyllableName.Fa);

registerPianoBoardListener( keyName: "5",          Domain.HIGH, SyllableName.So);
registerPianoBoardListener( keyCode: 221,         Domain.HIGH, SyllableName.So);
registerPianoBoardListener( keyName: "DIVIDE",  Domain.HIGH, SyllableName.So);
```

只要运行registerPianoBoardListener，就能自动绑定通码、断码、升调、降调。

Summary

- 经过本次project，熟练了Verilog硬件开发的基本方法，总结了EGO1开发的常用工具模块。
- 初步了解了串口通信的原理，熟练掌握Verilog、Java收发串口信息的方法。
- 初步理解音乐信号的本质、扬声器发音的本质。虽然最后未能实现完美的混音，但是在这个过程中听闻了傅里叶变换、信号与处理、方波信号、正弦波信号等技术的名称，让我对以后相关课程的学习充满了期待。

参考文献

- EGO1音乐播放器.pptx
- EGO1学生开发包

- 领航者ZYNQ之FPGA开发指南_V1.3.pdf