

CS214 期末大作业说明

CS214 期末大作业说明

GIT

CPU 特性 (按照文档要求)

ISA

基础ISA指令

寻址空间设计:

哈佛结构, 内存地址和指令地址分布于2个模块中

指令空间:

数据空间:

外设IO寻址范围:

CPI:

CPU接口:

CPU内部结构

CPU各子模块的模型

cpuDecoder

InstructionMemory

DataMemory

CpuController

InstructionFetcher

CpuExecutor

下面是添加的关于IO的管理和一些外设driver模块

MemOrIO

输入外设SwitchDriver

输入外设UartDriver

输出外设LightDriver

输出外设 PianoDriver

输出外设TubeDriver

测试说明

vivado模拟测试

asm和mips测试项目 (上板测试)

mips测试文件都进行保留, 位于test路径的mips文件夹下

测试状态

主要特色 (讲解可参照BONUS视频)

1. 基于华为大规模逻辑设计指导书和高通绝密VERILOG 编码规范进行代码重命名和优化, 并且总结我们一套的命名规范
2. 最新CPU架构的串口通信的巧妙设计
3. 电子琴 的巧妙设计

开发过程的问题及总结

1.vivado和vscode 和GIT 结合的多人合作的方法论

2.硬件编写的主要事项

叶璨铭 测试场景2， CPU全面升级，电子琴，代码规范；

王睿 UART，基础CPU组装，测试场景1，2调试，七段数码管；

张力宇 测试场景1，基础CPU组装，基础功能仿真；

贡献比平分

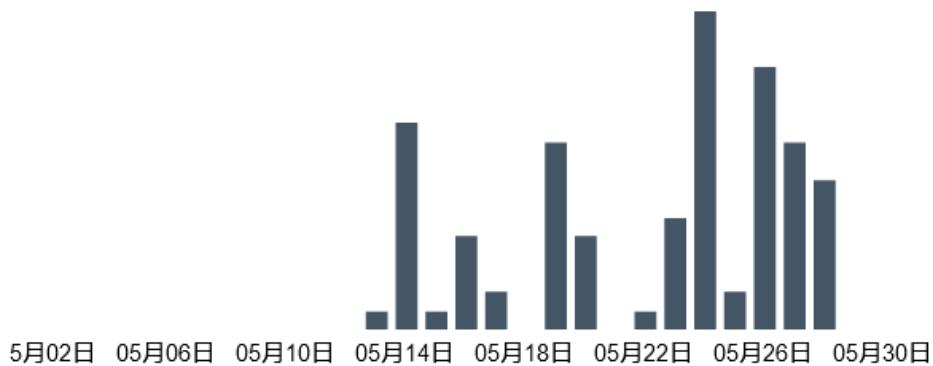
GIT

GITEE连接

采用gitee进行版本控制，主要是控制设计和仿真文件，约束xdc文件，ip核xci文件，asm和mips文件。其中每一个文件都放在相对应的文件夹内，在vivado中导入即可快速复现工程。

develop ▾

最近 4 周 提交动态:



提交总数: 103

当前分支 的所有文件数: 89

贡献者: 2

仓库大小: 8 MB

测试环境有效隔离可以更快复现工程，以及多人协作

利用git标签控制，发布release版本，方便进行版本回退

<p>v1.0-mips 2022-05-30 11:06</p>	<h2>测试场景1、2 Mips程序16进制文本机器码文件发布！</h2> <p>yeccanming</p> <p>这是两个16进制文本机器码文件，由改编后的MARS编译生成。 可以经由串口工具的16进制文件发送功能发送到CPU上。 完整地实现了https://gitee.com/yeccanming/SUSTech-CS202_214-Computer_Organization-Project/blob/master/dev-doc/architect/Computer%20Organization%E5%A4%A7%E4%BD%9C%E4%B8%9A-cs214%20(1).pdf中的要求。</p> <p>最后提交信息为： I2 基于已完成的Mips Cpu， 实现测试场景1、2并且通过测试</p>
<p>v1.0 2022-05-29 14:23</p>	<h2>下载</h2> <p>situation1_test1.coe situation2_integral.coe 下载 Source code (zip) 下载 Source code (tar.gz)</p>
	<h2>MIPS CPU 1.0</h2> <p>yeccanming</p> <ul style="list-style-type: none">• 描述： 稳定版， 可以稳定运行MIPS指令。• 更改：

CPU 特性 (按照文档要求)

ISA

基础ISA指令

| 我们实现的基础指令集是课件上指出的subset

Minisys - A subset of MIPS32								
Type	Name	funC(ins[5:0])	Type	Name	opC(ins[31:26])	Type	Name	opC(ins[31:26])
R	sll	00_0000	I	beq	00_0100		jump	00_0010
	srl	00_0010		bne	00_0101	J	jal	00_0011
	sllv	00_0100		lw	10_0011			
	srlv	00_0110		sw	10_1011			
	sra	00_0011						
	sraw	00_0111						
	jr	00_1000						
	add	10_0000		addi	00_1000			
	addu	10_0001		addiu	00_1001			
	sub	10_0010		slti	00_1010			
	subu	10_0011		sltiu	00_1011			
	and	10_0100		andi	00_1100			
	or	10_0101		ori	00_1101			
	xor	10_0110		xori	00_1110			
	nor	10_0111		lui	00_1111			
	slt	10_1010						
	situ	10_1011						

MIPS_Green_Sheet.pdf

NOTE:

Minisys is a subset of MIPS32.

The **opC** of **R-Type** instruction is **6'b00_0000**

BASIC INSTRUCTION FORMATS

R	opcode	rs	rt	rd	shamt	funct	
	31	26 25	21 20	16 15	11 10	6 5	0
I	opcode	rs	rt	immediate			
	31	26 25	21 20	16 15			
J	opcode	address					
	31	26 25	0				

调用方法是按照汇编调用

寻址空间设计：

哈佛结构，内存地址和指令地址分布于2个模块中

指令空间：

[0x0000_0000-0xFFFF_FFFF]

数据空间：

[0x0000_0000-0xFFFF_FC00)

外设IO寻址范围：

设备编号	外设名称	外设类型	约定内存地址1	数据类型/行为描述	备注
0	左灯光(8)	输出设备	0xFFFFFC62	输出的32位数的低比特 8位点亮左边的8盏灯	不允许 sw
0	右灯光 (16)	输出设备	0xFFFFFC60	输出的32位数的低比特 16位点亮右边的16盏灯	
1	左开关(8)	输入设备	0xFFFFFC72	输入的32位数的低比特 8位为左边的8个开关	不允许 lw
1	右开关 (16)	输入设备	0xFFFFFC70	输入的32位数的低比特 16位为右边的8个开关	
2	左七段数码管(4)	输出设备	0xFFFFFC82	输出的32位数的低比特 16位以十六进制显示在 左数码管	
2	右七段数码管(4)	输出设备	0xFFFFFC80	输出的32位数的低比特 16位以十六进制显示在 右数码管	
3	串口输出(8位)	输出设备	0xFFFFFC92	输入的32位数的低比特 8位发送到上位机	未实现

设备编号	外设名称	外设类型	约定内存地址1	数据类型/行为描述	备注
3	串口输入(8位)	输入设备	0xFFFFFC90	输入的32位数的低比特 8位为最新的串口数据	
4	电子琴输出(8位)	输出设备	0xFFFFFCA2	输出的32位数的低比特 8位按照 2catycm标准电子琴格式 解析为音乐并播放。	支持36阶半音符，4音轨。

CPI:

CPI = 1

单周期CPU

CPU接口：

时钟采用开发板自带的时钟接口

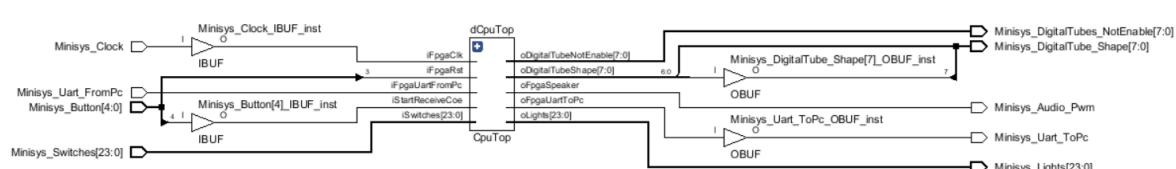
复位按钮是S2

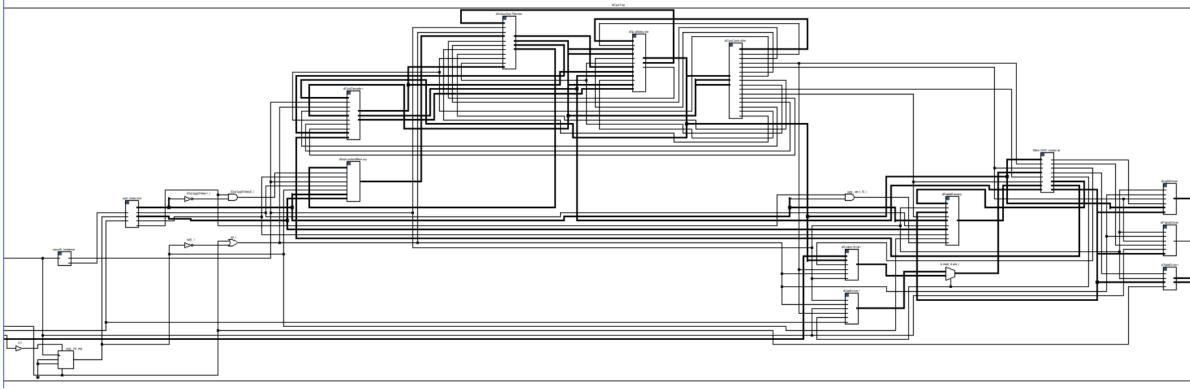
UART接口，UART模式切换接口按钮S4

数码管LED，24位拨码开关，24位LED灯

CPU内部结构

CPU外部的接口的设置

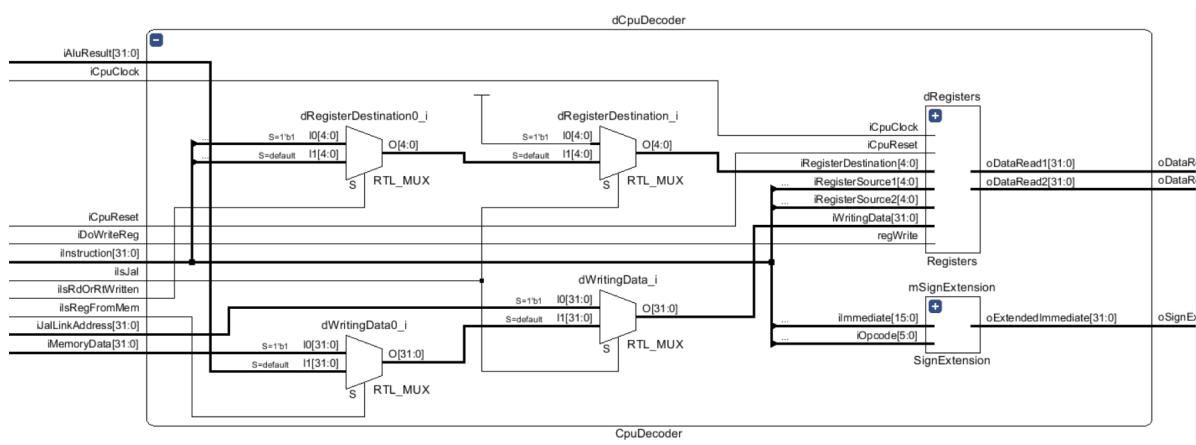




CPU各子模块的模型

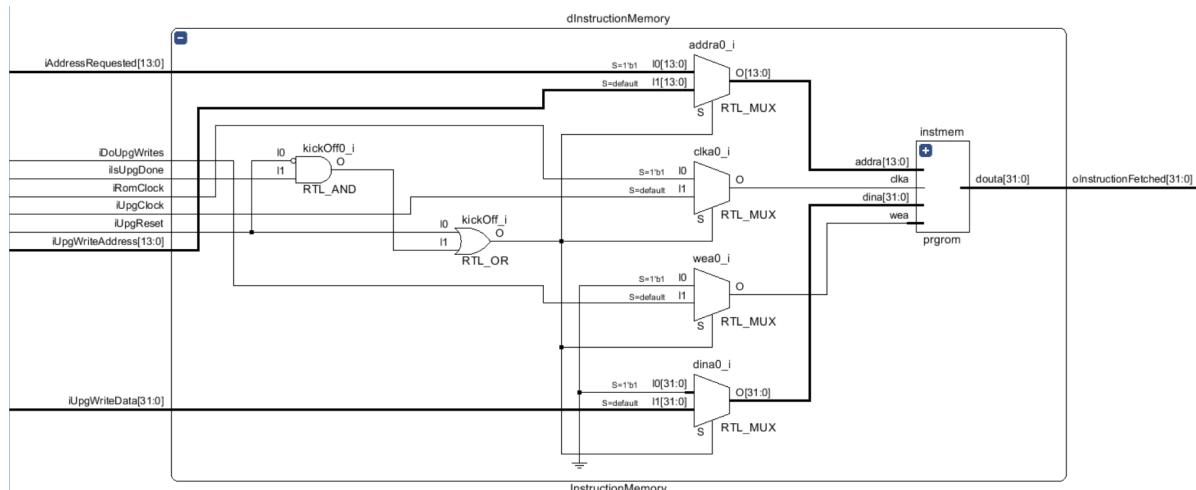
首先是6个核心CPU模块

cpuDecoder



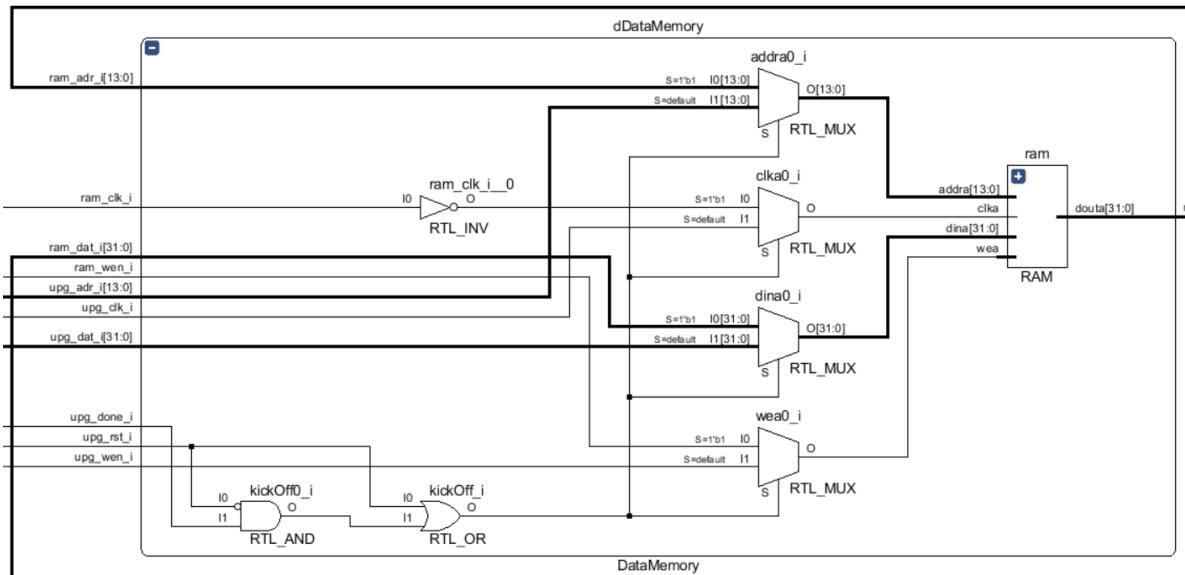
cpuDecoder模块主要用来解析指令，输出操作数，扩展的立即数等。

InstructionMemory



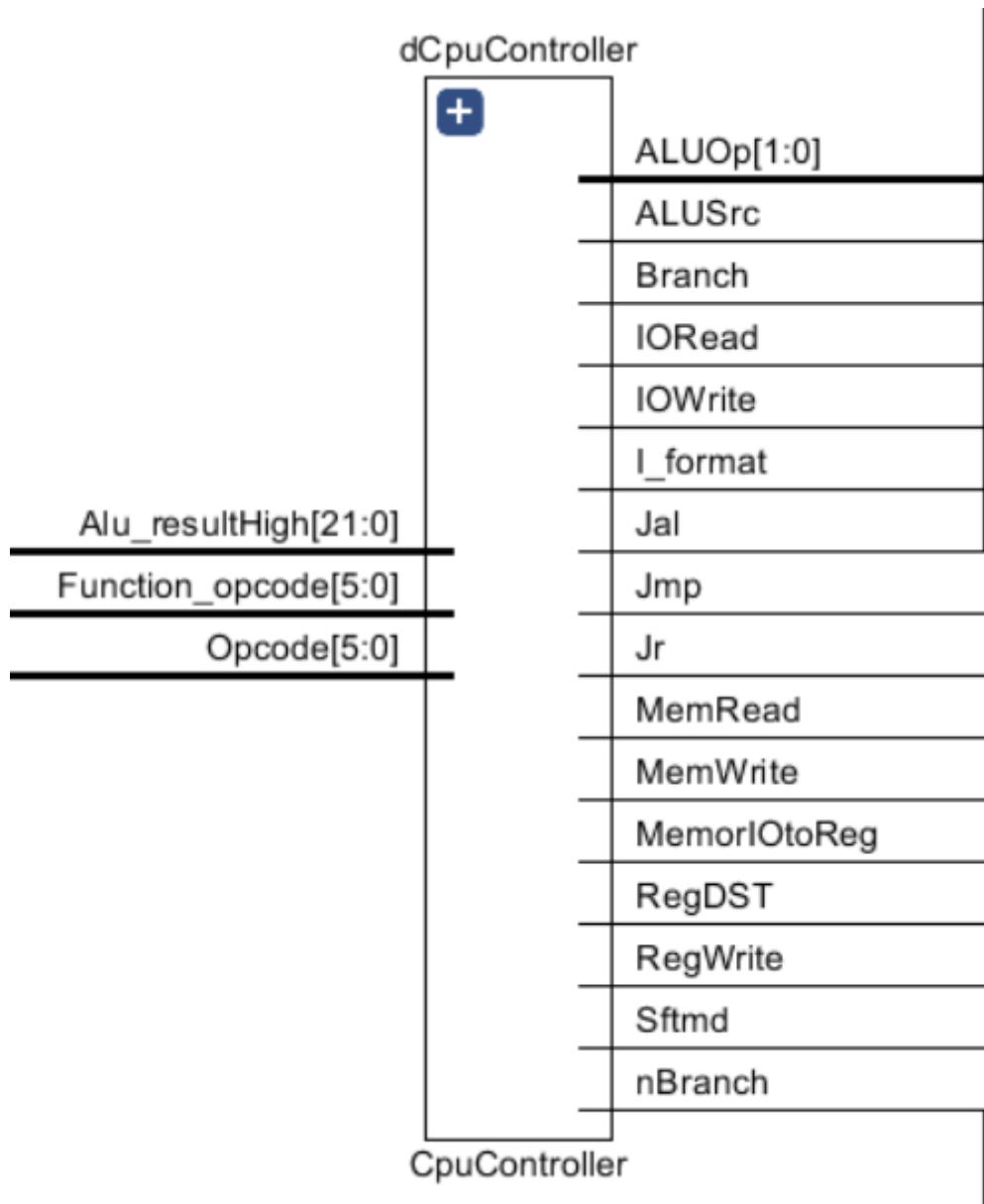
InstructionMemory模块主要用来存储指令，也可和uart接口做交互。

DataMemory



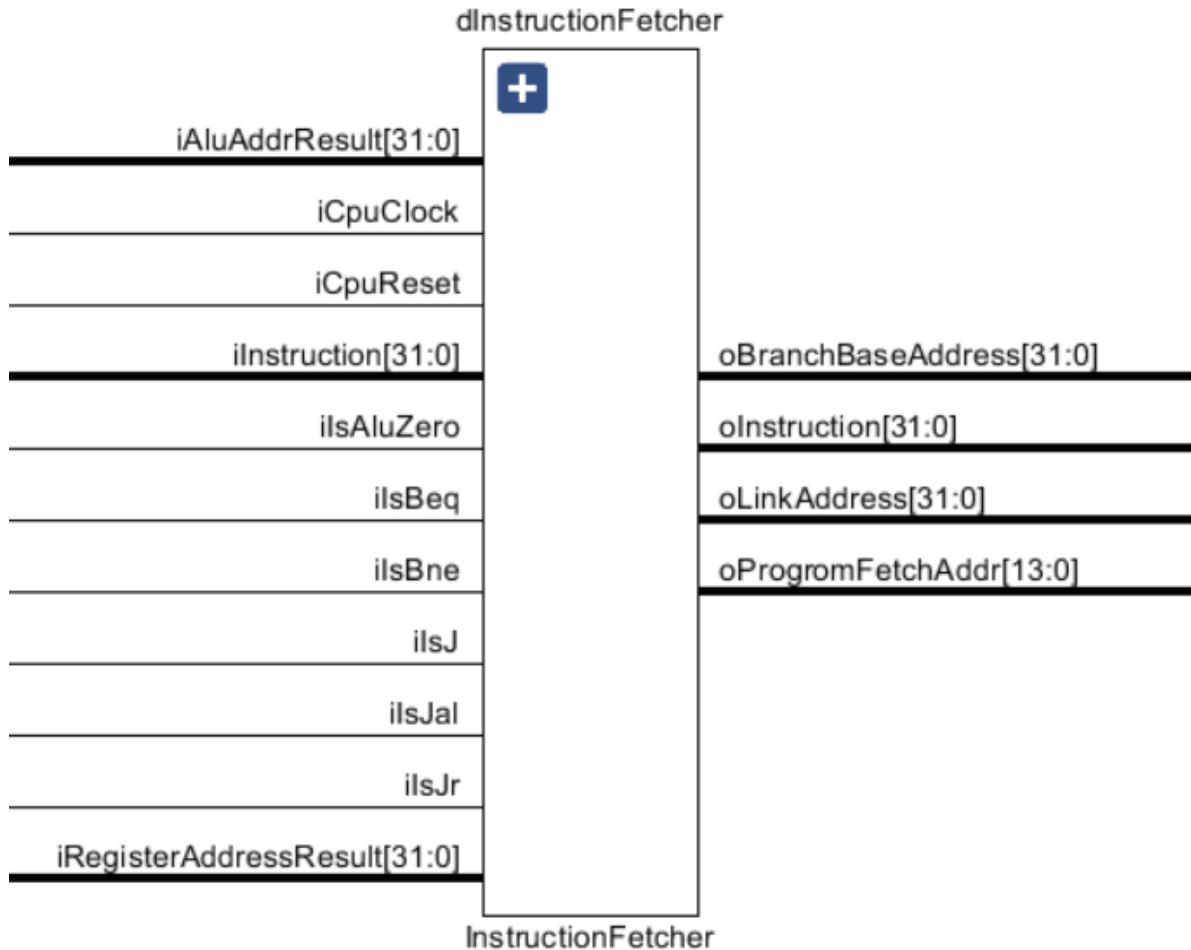
DataMemory模块主要用来存储数据，也可和uart接口做交互。

CpuController



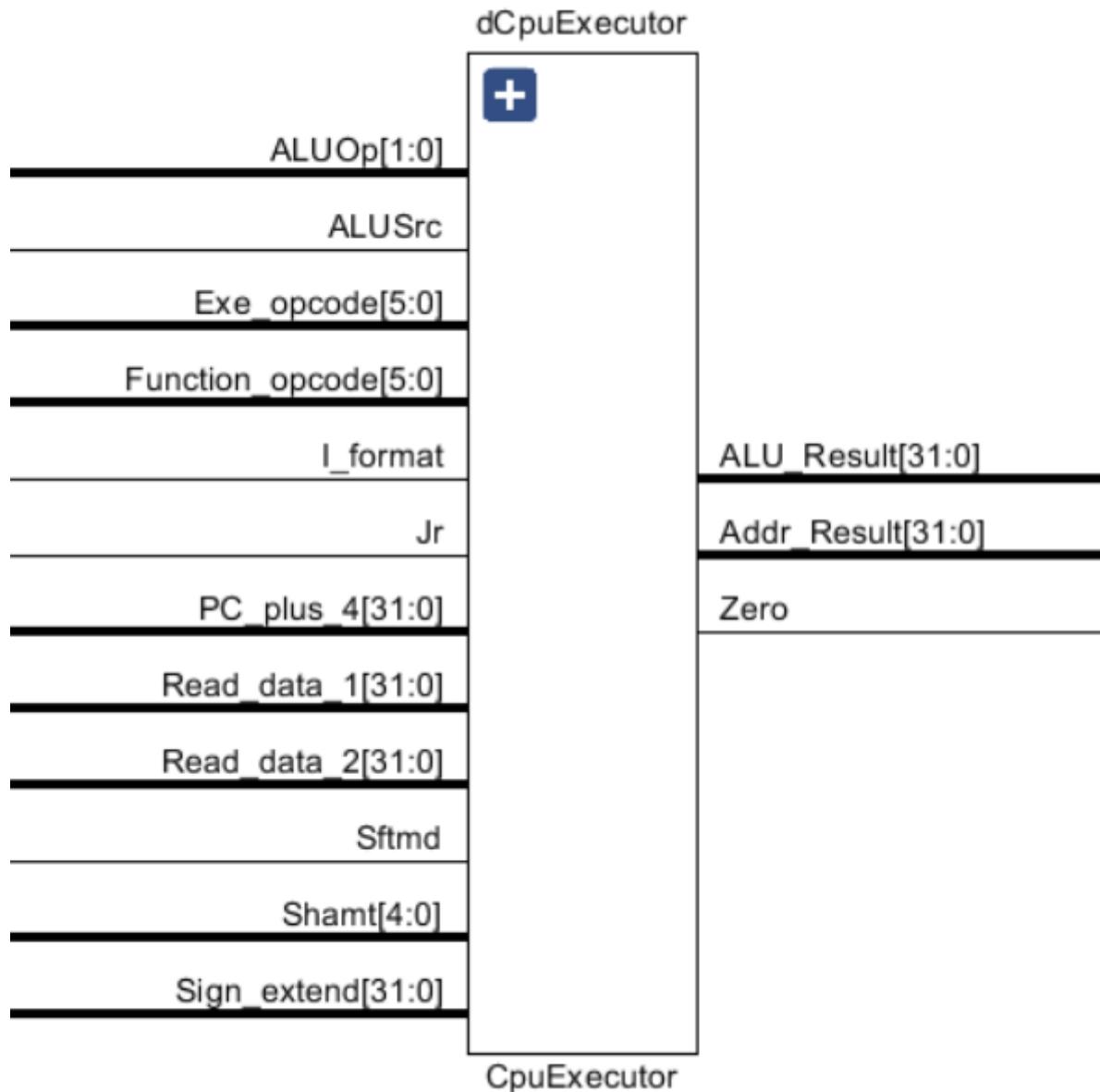
CpuController模块主要根据指令中的Function_opcode和Opcode来输出各种控制信号给到其他模块。

InstructionFetcher



InstructionFetcher主要根据时钟更新PC寄存器的值来取出相应的指令。注意执行jal指令时要先算出PC+4的值作保留，然后再更新PC的值。

CpuExecutor



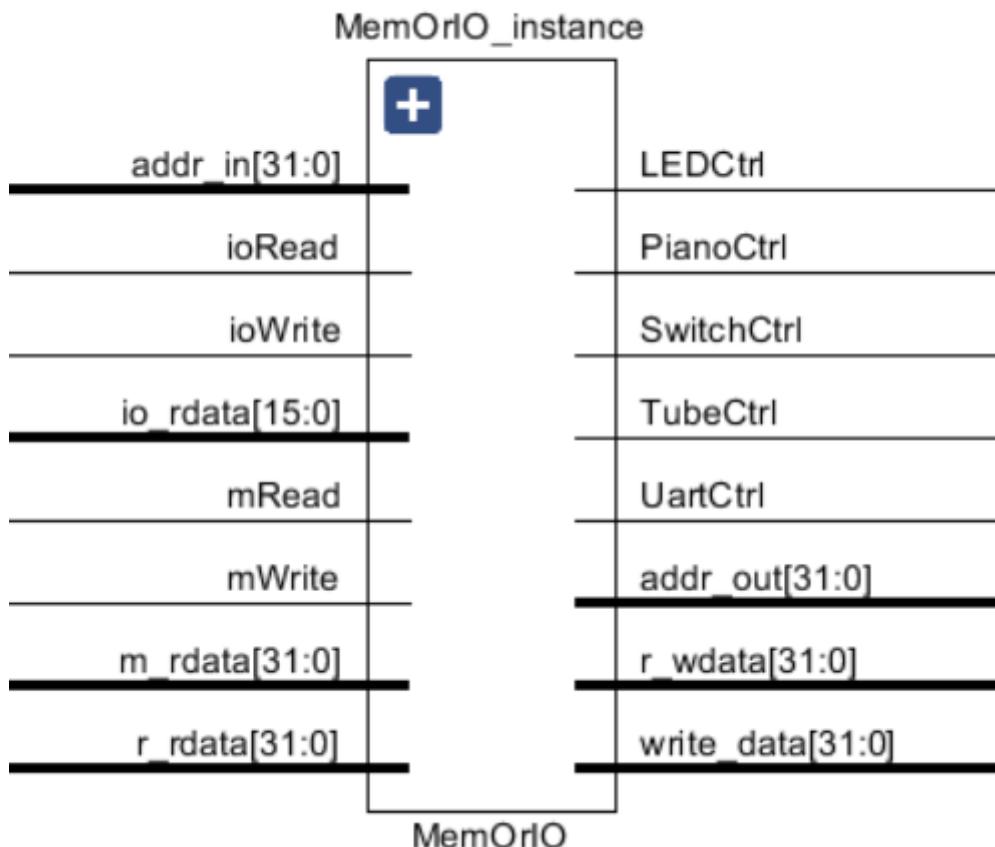
Executor其实是CPU的ALU模块，其主要是计算加减法和移位操作，然后把输出值给出去，ALU_Result主要是给到Memory去取出地址，是针对SW,LW这类，或者针对各类的加减shift操作是给到register的。

Addr_Result主要可能是给到PC寄存器。Zero主要是辅助进行判断是否跳转。

这个模块设计巧妙的是beq或者bne都是通过和减法是同一个ALU_ctrl模块，达到精简设计。

下面是添加的关于IO的管理和一些外设driver模块

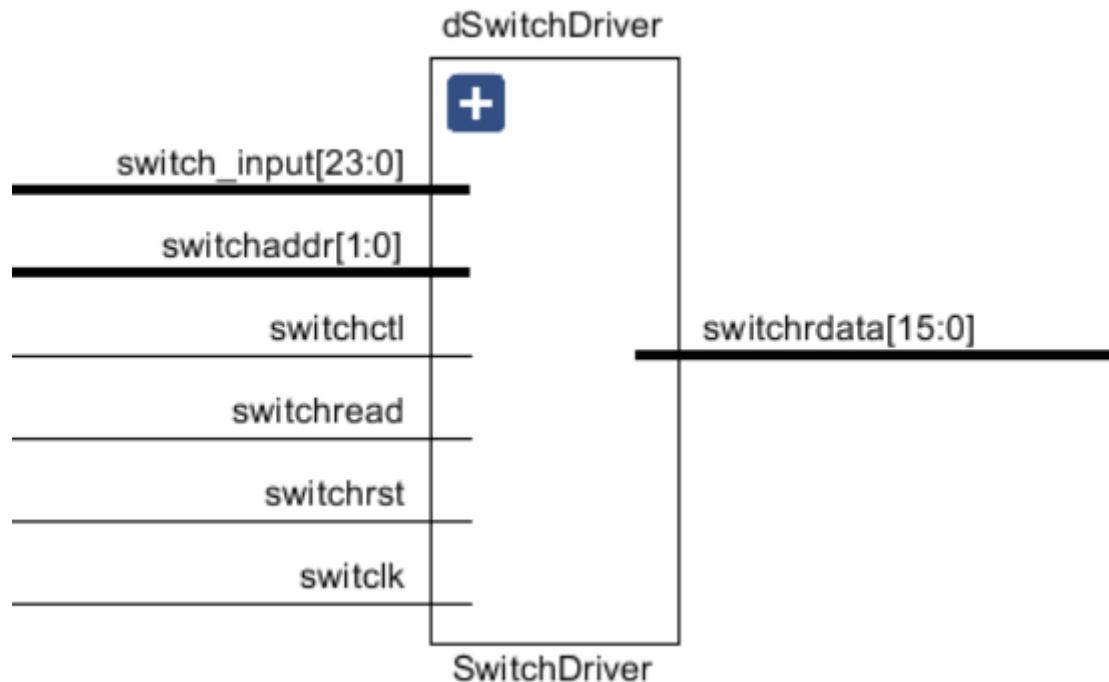
MemOrI0



这个模块核心就是为了控制和外设之间的交互，我们在完全重构之后的CPU是把这个模块添加到IOManger

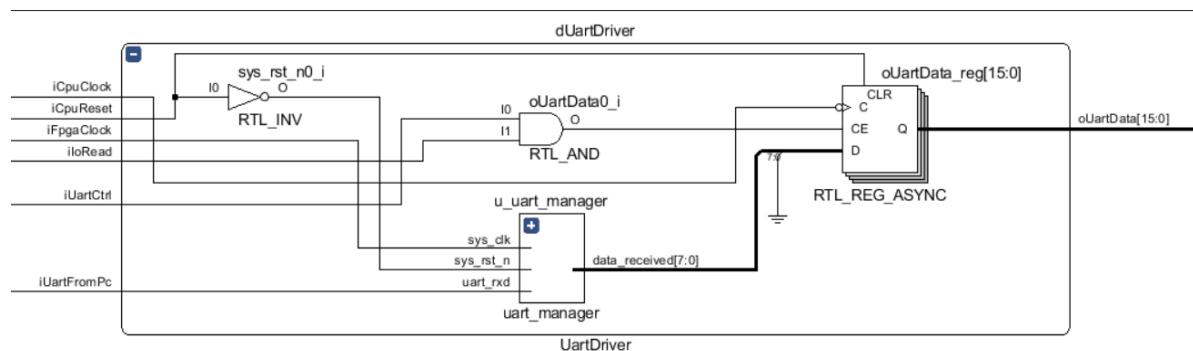
现在这里主要是提供了几个Driver的control 信号的输出，r_wdata能统一提供该输出的地址。

输入外设SwitchDriver



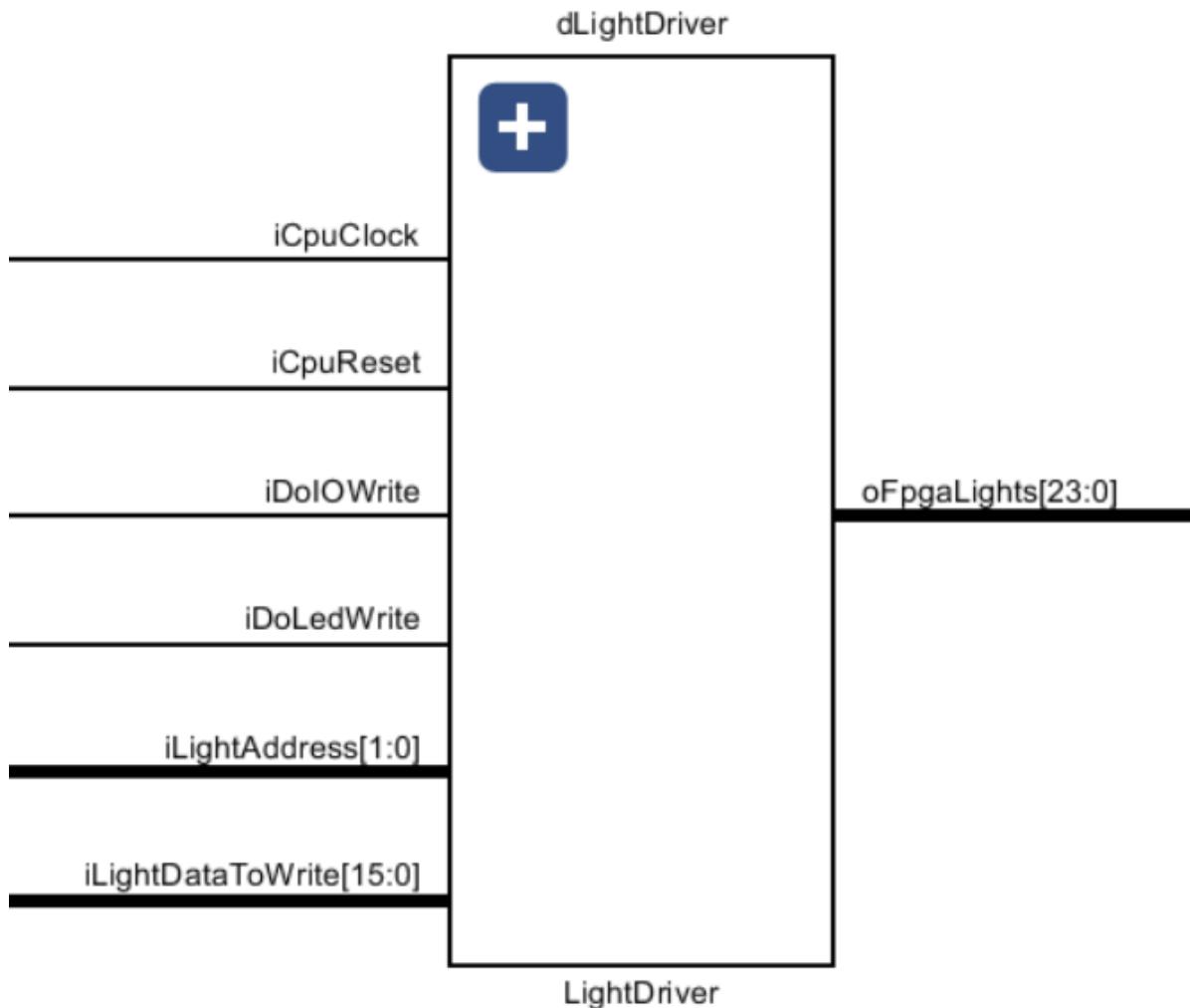
这个模块主要是读入24个拨码开关，然后传入的2位地址主要是判断是否合法（是否应该lw的标志），传出的switchrdata 是我们认为16位，因为一次lw我们就认为最多读取16位的数据，也就是在测试场景1的时候会有读入16位的数据。

输入外设UartDriver



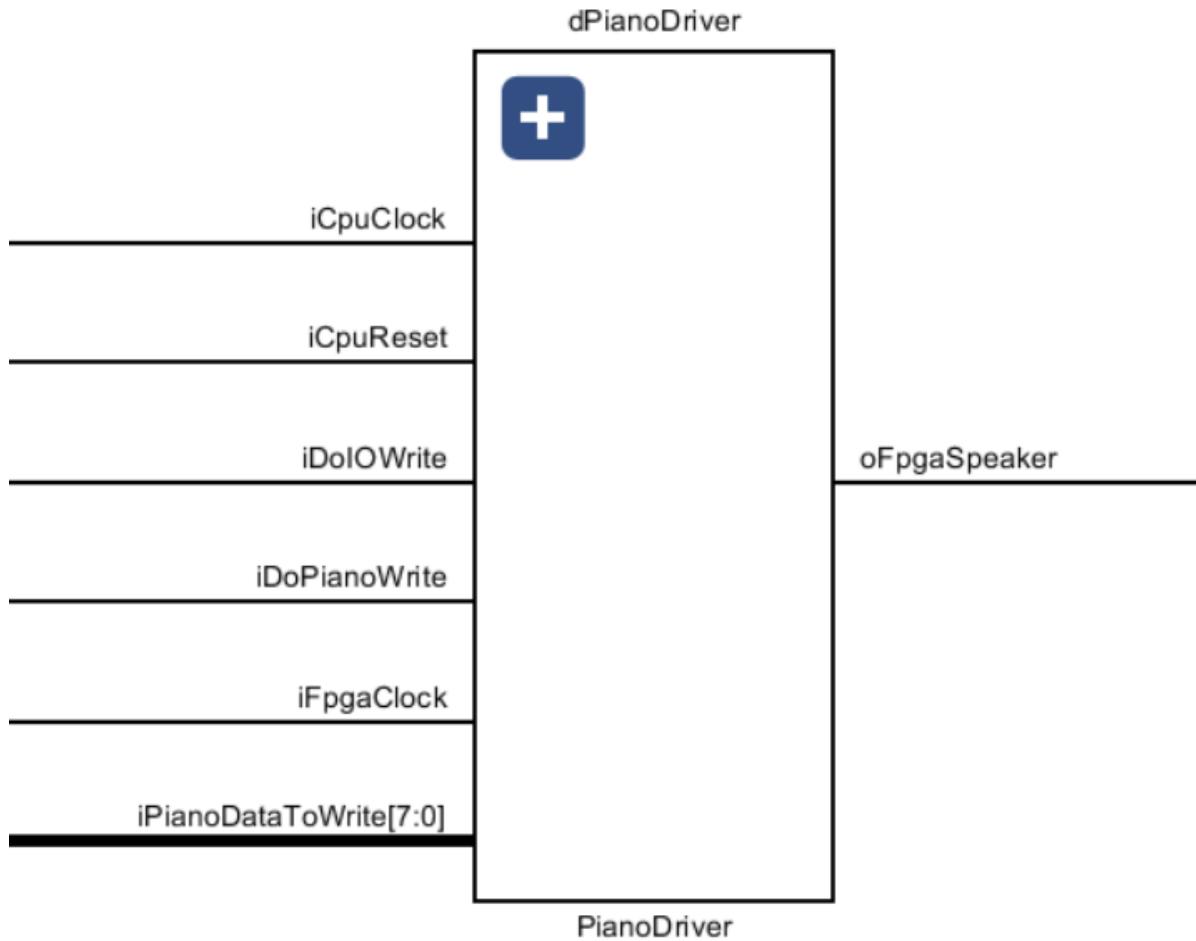
这个模块主要是进行串口的通信

输出外设LightDriver

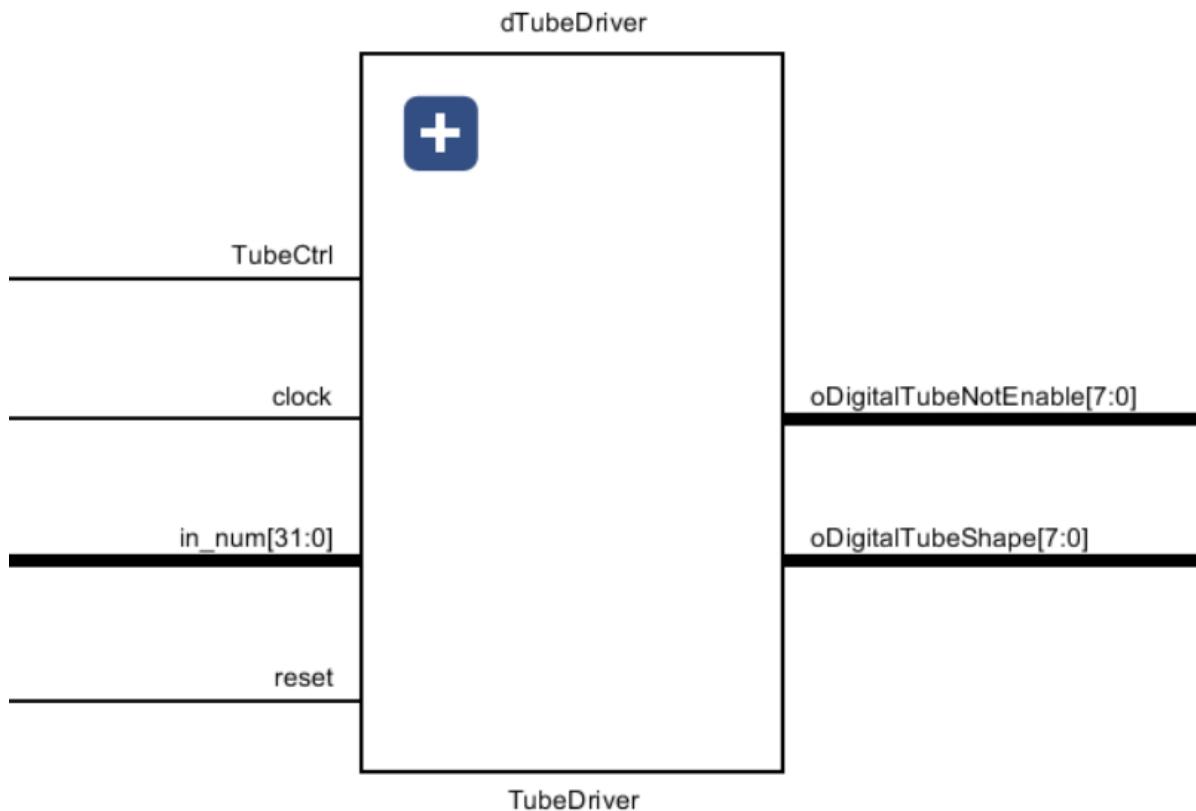


这个模块主要是控制LED灯，每次iLightAddress 主要是判断是写左边8个灯还是写右边16个灯。这种分开的设计比较方便写ASM文件，也是课件中给出的指导方法。

输出外设 PianoDriver



输出外设TubeDriver



这个是数码管模块，输入是ALU的结果write_data 的32位，这是因为正好8个数码管，然后正好可以显示32的数字。然后输入的clk主要记得要变成50HZ左右，这个是数码管的CLK的应该有的频率。然后Minisys和EGO1不同的是，这些显示都是低电平有效，所以输出要取反。

做这个模块的时候主要要把in_num存起来，不然会反复更新，导致无法有稳定的输出。

测试说明

vivado模拟测试

测试方法	测试类型	测试样例描述	测试结果	测试结论
仿真	单元	对OJ上分别写的模块进行测试	通过	各子模块功能没有问题
仿真	集成	测试场景1 testSituation1.v (在test路径下verilog文件夹)	通过	基础的指令没有问题
仿真	集成	测试场景1 testNewSituation1.v (在test路径下verilog文件夹)	通过	asm文件没有问题

asm和mips测试项目（上板测试）

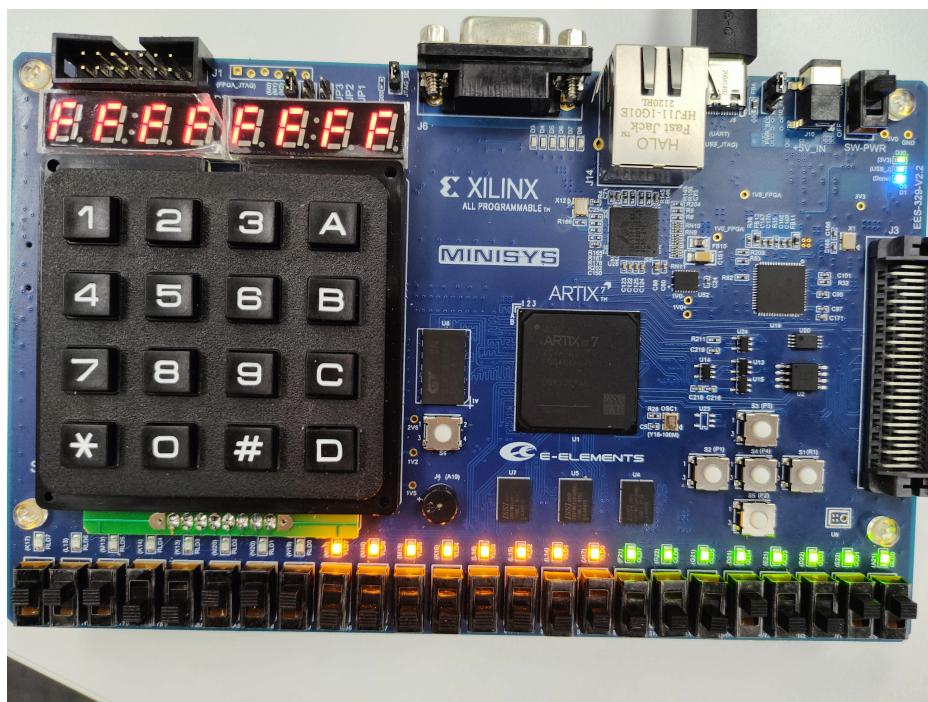
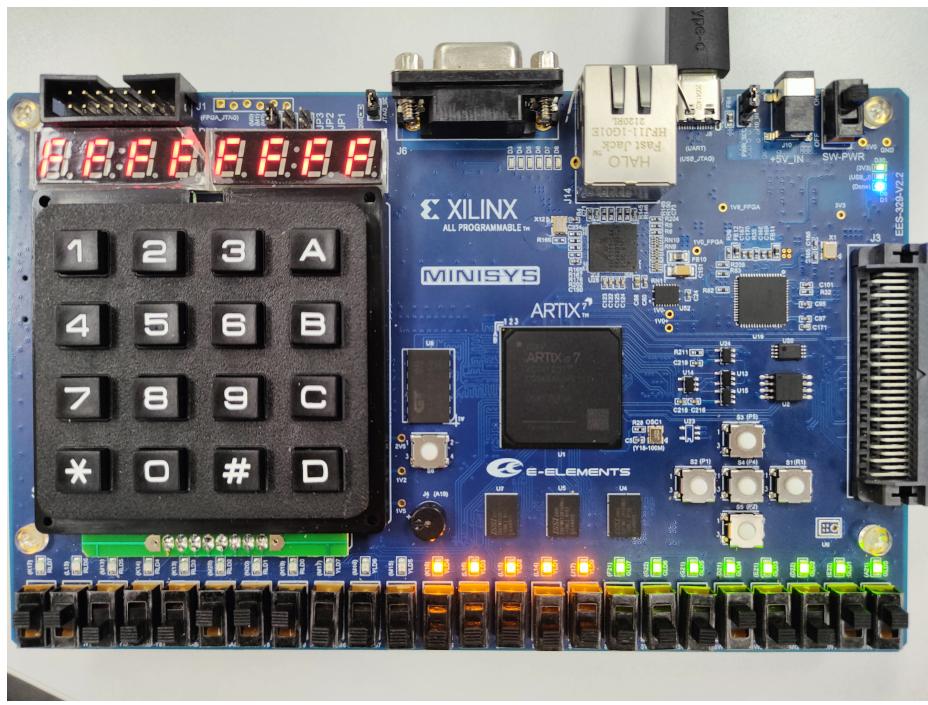
mips测试文件都进行保留，位于test路径的mips文件夹下

- demo_read_switich_write_led.mips
 - 期望行为：这是最简单的测试场景，当导入到我们的CPU上后，用户按什么按键就会亮什么灯。
 - 验证的硬件功能：lw, sw, j的支持；MemOrIOn模块。
- jal_test.mips
 - 期望行为：右边的16盏灯全亮起来
 - 验证的硬件功能：jal, jr, j, sw的支持；
- demo_flow.mips

- 期望行为：左边的八个灯每隔1s轮流亮起来，形成流水灯。
- 验证的硬件功能：jal, jr, j, sw, lw的支持；
- 验证的软件功能：commons/stdio_minisys.mips 的 sleep, write_control函数。
- situation1.asm
 - 000状态下, 可以及时确定数码管及是否是回文数, 001状态下可以保存2个数, 保存的数字不会被刷新。具体测试样例：A是1000_0000_0001_1111, B 是0000_0000_0001_0011, 值得说明的是110的情况下是0001_1111_1111_1111, 然后111情况下是算术右移, 结果是1111_1111_1111_1111.
- new_situation1.mips
 - 与situation1asm的效果几乎一样, 但是是通过后期调用的宏进行重构。
- situation2.mips
 - 在此做一些说明,
 - 比如输入1000_0011(-3, -125),1000_0100(-4,-124), 0000_0000(0, 0),0000_0001(1, 1),括号里面的值都是原码, 然后我们可以看出比如是 101情景, 数据集3 (补码) , 下标0, 则会输出二进制形式1000_0011 (补码最小) .如果101场景, 把这个数据集换成1 (无符号数) , 下标3, 则会输出二进制形式1000_0100 (无符号数最大)
 -

测试状态

- demo_read_swttich_write_led.mips
 - 通过
- jal_test.mips
 - 通过
- demo_flow.mips
 - 通过
- situation1.asm
 - 通过



- new_situation1.mips
 - 通过。
- situation2.mips

- 通过。

主要特色（讲解可参照BONUS视频）

1. 基于华为大规模逻辑设计指导书和高通绝密VERILOG 编码规范进行代码重命名和优化，并且总结我们一套的命名规范

课件中的代码命名相对比较难debug，在网上找到华为和高通内部的设计书之后决定对线路命名和线路连线进行完全意义上的重新设计。包括但不限于一下部分：

1. 按照驼峰命名法，对于所有input添加i前缀，对于所有的output添加o前缀，这个input和output是对于内部模块来说的。对于模块的例化始终在模块名字之前添加d进行区分。对于连接外设的设备统一添加driver，对于内部设备则尽量以表达明白含义的方式进行命名。
2. 一般能不使用中间变量的时候就不使用中间变量，在管脚绑定的时候能通过以下比较简洁的方式书写就通过更加简洁的方式书写。比如如下这种管脚绑定的方式

```
RAM ram (
    .c1ka (kickoff ? ram_c1k : upg_c1k_i),
    .wea (kickoff ? ram_wen_i : upg_wen_i),
    .addr_a (kickoff ? ram_addr_i : upg_addr_i),
    .dina (kickoff ? ram_dat_i : upg_dat_i),
    .douta (ram_dat_o)
);
```

3. 以上仅仅是列举的一小部分规范，具体的规范请参考dev-dov/style 路径下的 Verilog style.md，里面有更加详细的规范说明

2.最新CPU架构的串口通信的巧妙设计

这部分主要由叶璨铭同学在视频中介绍

3.电子琴 的巧妙设计

这部分主要由叶璨铭同学在视频中介绍

开发过程的问题及总结

1.vivado和vscode 和GIT 结合的多人合作的方法论

首先要明白的是vivado的源代码文件是可以批量的导入源代码的，对于v文件，肯定是没什么问题的。xdc约束文件也是可以放在一个文件夹里面，然后添加约束文件夹，也可以一键导入。

然后还有比较核心的就是xci的IP核文件，这个也是可以直接导入的，但是注意不同IP核之间要放在不同的路径下。

也正是因为这些一键导入，可以支持vivado和vscode的协作，vscode可以调用第三方插件进行自动生成testbench，或者检查一些低级的错误。

然后vscode还是和GIT的图形化界面的联系IDE。通过VS CODE才可以大家更加方便使用GIT。

在做一些冒险的开发的时候，才可以直接切换一个分支，然后等到功能稳定之后，再把分支切换回来，或者开发不了就永远的把那个分支留在那。

GIT在实操的过程中也会出现因为不熟悉带来的不方便的地方，但是从长远来说，GIT的使用肯定更加方便项目管理。

Vivado建议大家的字体都设置成一样，不同的字体的编码方式是不一样的，在后期我们组出现了中文编码乱码的问题，而且很难切换回去。

可以按照JAVA项目那样，分成main和test，main底下分成mips和verilog，test底下也分成这样。

2.硬件编写的主要事项

时钟一定要想清楚要用多少HZ的，不然会有很多奇怪的BUG。

还有就是写模拟文件尽量还是要满足真实的拨按钮的时间线，模拟调整的太快，实际上有些时候会有很多BUG。