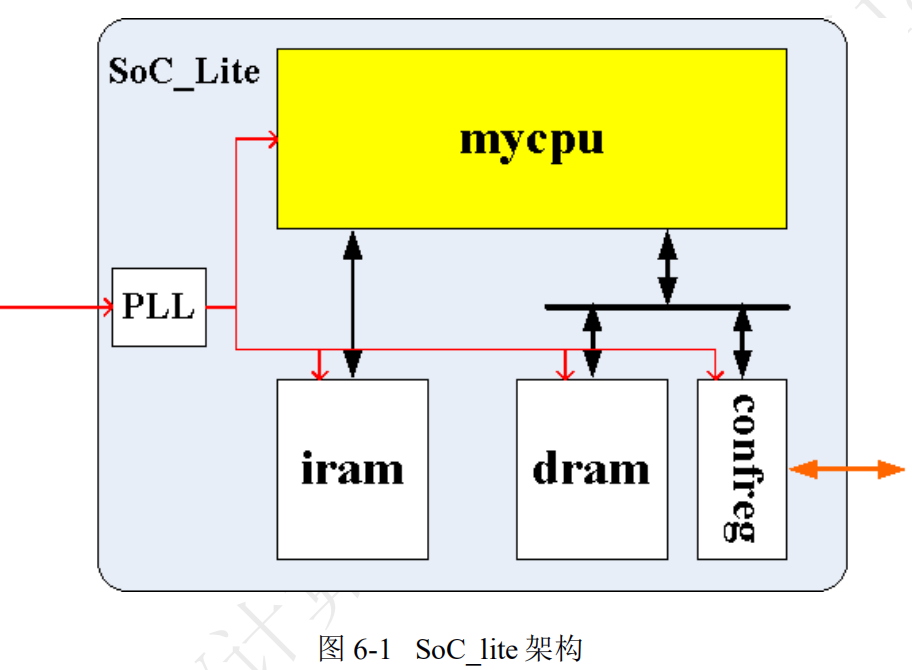
**实验1报告**

组员：张丽玮

一、实验任务

替换myCPU中的龙芯三级流水代码，改为计组实验时的多周期代码。更改顶层的接口定义，使得代码融入SoC\_lite框架。之后在vivado中调试运行，根据龙芯三级流水的测试样例结果，run all自动测试寻找error，pass之后上板测试。

二、实验设计



实验环境的架构，而本次实验的设计在于mycpu这一块。

实验目录结构——

|-cpu132\_gettrace/ 目录，验证平台之生成参考 trace 部分。

| |--rtl/ 目录，SoC\_lite 的源码。

| | |--soc\_lite\_top.v SoC\_lite 的顶层。

| | |--CPU\_gs132/ 目录，龙芯开源 gs132 源码，对其顶层接口做了修改。

| | |--CONFREG/ 目录，confreg 模块，连接 CPU 与开发板上数码管、拨码开关等 GPIO 类设

备。

| | |--BRIDGE/ 目 录 ， bridge\_1x2 模 块 ， CPU 的 data sram 接 口 分 流 去 往 confreg 和

59

data\_ram。

| | |--xilinx\_ip/ 目录，Xilinx IP，包含 clk\_pll、inst\_ram、data\_ram。

| |--testbench/ 目录，仿真文件。

| | |--tb\_top.v 仿真顶层，该模块会抓取 debug 信息生成到 trace\_ref.txt 中。

| |--run\_vivado/ 目录，运行 Vivado 工程。

| | |--soc\_lite.xdc Vivado 工程设计的约束文件

| | |--cpu132\_gettrace/ 目录，创建的 Vivado 工程，名字就叫 cpu132\_gettrace

| | | |--

cpu132\_gettrace.xpr 创建的 Vivado 工程

| |--trace\_ref.txt 该验证平台运行测试 func 生成的参考 trace。发布包已包含该文件。

|

|-func/ 目录，验证平台之测试 func 交叉编译部分。发布包已包含编译结果。

| |--include/ 目录，mips 编译所有头文件

| | |--asm.h MIPS 汇编需用到的一个宏定义的头文件，比如 LEAF(x)。

| | |--regdef.h MIPS 汇编 32 个通用寄存器的助记符定义。

| |--inst/ 目录，各条指令的验证汇编程序

| | |--Makefile 子目录里的 Makefile，会被上一层次的 Makefile 调用。

| | |--inst\_test.h 各条指令的验证程序使用的宏定义头文件

| | |--n\*.S 各条指令的验证程序，汇编语言编写。

| |--obj/ 目录，func 编译结果

| | |--sim\_test\_ram/ 目录，用于仿真的编译结果

| | |--syn\_test\_ram/ 目录，用于综合实现的编译结果

| |--Makefile 编译脚本 Makefile

| |--start.S func 的主函数

| |--bin.lds.S 交叉编译的链接脚本

| |--convert.c 生成 coe 和 mif 文件的本地执行程序源码。

| |--rules.make 子编译脚本，被 Makefile 调用

|

|-mycpu\_verify/ 目录，自实现 CPU 的验证环境

| |--rtl/ 目录， SoC\_lite 的源码。

| | |--soc\_lite\_top.v SoC\_lite 的顶层。

| | |--myCPU / 目录，自实现 CPU 源码。

| | |--CONFREG/ 目录，confreg 模块，连接 CPU 与开发板上数码管、拨码开关等 GPIO 类设

备。

| | |--BRIDGE/ 目 录 ， bridge\_1x2 模 块 ， CPU 的 data sram 接 口 分 流 去 往 confreg 和

data\_ram。

| | |--xilinx\_ip/ 目录，Xilinx IP，包含 clk\_pll、inst\_ram、data\_ram。

| |--testbench/ 目录，仿真文件。

| | |--mycpu\_tb.v 仿真顶层，该模块会抓取 debug 信息与 trace\_ref.txt 进行比对。

| |--run\_vivado/ 目录，运行 Vivado 工程。

| | |--soc\_lite.xdc Vivado 工程设计的约束文件

| | |--mycpu/ 目录，创建的 Vivado 工程，名字就叫 mycpu

| | | |--mycpu.xpr 创建的 Vivado 工程

（一）设计方案

1、总体设计思路

在原来多周期处理器的基础上，加一个顶层模块mycpu\_top，从而对接相应接口，没有赋值的input默认赋值为1。而多周期处理器主要分为四个模块，regfile的文件读写，alu的运算器，还有control控制器控制多周期各个控制信号，以及状态机表现多周期的状态转换。

2、模块1设计(mycpu\_top)

（1）工作原理

定义接口，调用mips\_cpu模块，从而实现接口的一一对应。

（2）接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | IN | 1 | 时钟信号，来自 clk\_pll 的输出时钟 |
| resetn | OUT | 1 | 复位信号，低电平同步复位 |
| inst\_sram\_en | OUT | 1 | ram 使能信号，高电平有效 |
| inst\_sram\_wen | OUT | 4 | ram 字节写使能信号，高电平有效 |
| inst\_sram\_add | OUT | 32 | ram 读写地址，字节寻址 |
| inst\_sram\_wdata | OUT | 32 | ram 写数据 |
| inst\_sram\_rdata | IN | 32 | ram 读数据 |
| data\_sram\_en | OUT | 1 | ram 使能信号，高电平有效 |
| data\_sram\_wen | OUT | 4 | ram 字节写使能信号，高电平有效 |
| data\_sram\_addr | OUT | 32 | ram 读写地址，字节寻址 |
| data\_sram\_wdata | OUT | 32 | ram 写数据 |
| data\_sram\_rdata | IN | 32 | ram 读数据 |
| debug\_wb\_pc | OUT | 32 | 写回级（多周期最后一级）的 PC，需要 mycpu 里将 PC 一路带  到写回级 |
| debug\_wb\_rf\_wen | OUT | 4 | 写回级写寄存器堆(regfiles)的写使能，为字节写使能，如果  mycpu 写 regfiles 为单字节写使能，则将写使能扩展成 4 位即  可 |
| debug\_wb\_rf\_wnum | OUT | 5 | 写回级写 regfiles 的目的寄存器号 |
| debug\_wb\_rf\_wdata | OUT | 32 | 写回级写 regfiles 的写数据 |

（3）功能描述

两部分，一部分为赋值，另一部分为调用mips\_cpu模块。

3、模块2设计(mips\_cpu）

（1）工作原理

定义接口，调用mips\_cpu模块，从而实现接口的一一对应。

（2）接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | IN | 1 | 时钟信号，来自 clk\_pll 的输出时钟 |
| rst | OUT | 1 | 复位信号，低电平同步复位 |
| Inst\_Req\_Valid | OUT | 1 | 数据高电平有效 |
| Inst\_Req\_Ack | OUT | 1 | 握手信号，高电平有效 |
| PC | OUT | 32 | ram 读写地址，字节寻址 |
| Adress | OUT | 32 | 地址 |
| Memwrite | OUT | 1 | 写数据信号 |
| Memread | OUT | 1 | 读数据信号 |
| Write\_data | OUT | 32 | 写数据 |
| Inst\_Ack | OUT | 1 | 握手信号，高电平有效 |
| Inst\_Valid | IN | 1 | 指令是否有效 |
| Instruction | IN | 32 | ram 读数据 |
| Mem\_Req\_Ack | IN | 1 | Ram的握手信号，高电平有效 |
| Read\_data | IN | 32 | 读入的数据 |
| Read\_data\_valid | IN | 1 | 读入数据是否有效，高电平有效 |
| Read\_data\_Ack | OUT | 1 | 读入数据的握手信号，高电平有效 |

（3）功能描述

主要实现多周期状态机处理，以及regfile、alu之后对于数据的处理，control模块的调用。

4、模块3设计(control）

（1）工作原理

从mips\_cpu中获得相应指令，从而判断是哪一种类型的指令，输出进行后续处理。

（2）接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| IN | IN | 6 | 指令输入 |
| FU | IN | 6 | 相应指令的function输入 |
| jump等 | OUT | 1 | 使能信号，表示是否是哪一类信号 |
| Regwrite | OUT | 1 | regfile读信号 |
| ALUop | OUT | 3 | 运算器指令信号 |

（3）功能描述

实现根据输入的指令，判断是何种指令，并判断应该执行运算器的哪种处理方式。

5、模块4设计(reg\_file）

（1）工作原理

寄存器，支持2读1写，同步读，异步写。add r1, r1, r1：新r1 = 旧r1 + 旧r1

 raddr1 = raddr2 = waddr = 1：读出旧值；算加法；加法结果在下一个clk上升沿到来时写入

（2）接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| clk | IN | 1 | 时钟信号 |
| rst | IN | 1 | 复位信号 |
| waddr | IN | 5 | 写口地址 |
| raddr1 | IN | 5 | 读口地址1 |
| raddr2 | IN | 5 | 读口地址2 |
| wen | IN | 1 | 写使能 |
| wdata | IN | 5 | 写数据 |
| rdata1 | OUT | 32 | 读数据1 |
| rdata2 | OUT | 32 | 读数据2 |

（3）功能描述

实现2读1写，同步读，异步写。

6、模块5设计(alu）

（1）工作原理

读入数据与指令，根据指令进行相应逻辑处理。

（2）接口定义

| **名称** | **方向** | **位宽** | **功能描述** |
| --- | --- | --- | --- |
| A | IN | 32 | 读入数据1 |
| B | IN | 32 | 读入数据2 |
| ALUop | IN | 3 | 读入运算指令 |
| Overflow | OUT | 1 | 溢出标志 |
| CarryOut | OUT | 1 | 进借位标志 |
| Zero | OUT | 1 | 零标志 |
| Result | OUT | 32 | 输出结果 |
| Xor | OUT | 32 | 异或数据 |

（3）功能描述

支持逻辑按位“与”（AND）、逻辑按位“或”（OR）、算术加法（Add）、算术减法（Subtract）、有符号整数比较（Seton less than，Slt）、移位等ALU运算功能。

（二）验证方案

1、总体验证思路

首先对比一下接口是否对应，之后运行仿真能否出现波形，验证无语法问题。在run all测试和龙芯测试数据进行比对，如果pass表示都一致，否则会finish，可以根据差异修改代码。最后上板测试，观察数码管显示结果。

2、验证环境

（1）仿真验证：可以选择看tcl是error还是pass，与龙芯测试数据对比。也可以观察波形判断。

（2）FPGA上板测试：开始，单色 LED 全灭，双色 LED 灯一红一绿，数码管显示全 0；执行过程中，单色 LED 全灭，双色 LED 灯一红一绿，数码管高 8 位和低 8 位同步累加；结束时，单色 LED 全灭，双色 LED 灯亮两绿，数码管高 8 位和低 8 位数值相同，对应测试功能点数目。

3、验证计划

具体说明要做哪些验证工作，要验证哪些功能点。功能点以下列表格形式列出：

| **编号** | **功能点描述** | **考核标准** |
| --- | --- | --- |
| 1 | 仿真测试 | 显示PASS |
| 2 | FPGA硬件测试 | LED等正确显示 |

三、实验实现

（一）实现交付说明

四个文件，由mycpu\_top顶层模块接入，mips\_cpu中还涵盖了control模块。另外两个是reg\_file寄存器和ALU运算器模块。

（二）实现说明

四个附件对应myCPU，mycpu\_top对应原top文件，为顶层接口。

四、实验测试

（一）测试过程

大部分时间花在理解题意上，由于一开始mycpu文件夹不是空的而误以为要在龙芯代码上进行删改。之前计组是在文件中写了//TO DO 或者 // TO CODE，比较明确。一开始对应接口的时候发现几个wen接口无法对应，由于顶层设计不同，而另外定义，用变量全部替换的方法。写完之后debug全线飘红，由于resetn之前是高电平复位，没有进行更改而导致始终没有有效PC。这之后虽然有了PC，但PC并没有在运行始终是复位值，最后比对发现是wen信号赋值错误。并且借此机会直接改为设计顶层top模块而不是在原本的mips\_cpu上进行更改。PC可以正常运行之后，run all发现所有的reference测试数据都是XXXX，发现我的trace文件是空的，从同学那里另存为替换之后最终得到PASS结果。上板也没有过多问题，自从重装了电脑之后一切顺利。

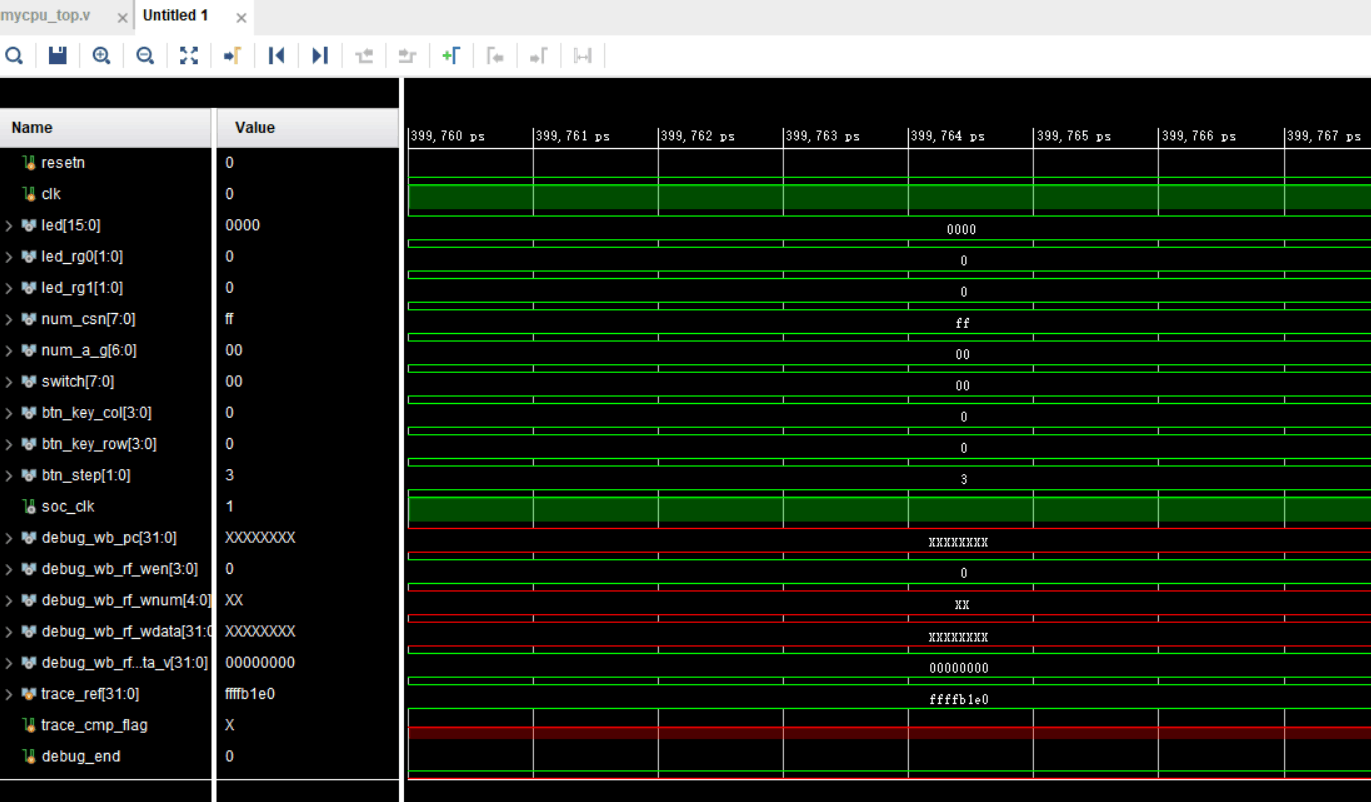
（二）测试结果

仿真测试全部PASS，上板测试LED灯亮暗正确。

1. 实验bug说明

（一）bug-1：resetn复位粗偶

1、错误现象

2、分析原因

应该是低电平复位，而原本代码是高电平复位。

3、解决方案

顶层模块直接~resetn调用。

六、成员分工

单人实验，没有分工。

七、实验总结

（一）组员：张丽玮

需要非常仔细看讲义，讲义上基本没有废话。很多时候我遇到问题去找同学询问，最后得到的答案都是“你看讲义的XXX条”。对讲义的描述内容有问题要及时询问，只有理解题意通读讲义才能顺利完成。

重装电脑系统带来全新世界，有时候确实需要有从头来过的勇气。

八、参考文献

计算机体系结构研讨课 讲义18-19 秋季v1.0