兰州大学信息科学与工程学院
计算机组成原理课程设计实验报告

一、 实验目的

1. 熟悉 MIPS 指令集中的运算指令，对这些指令进行归纳和分析
2. 了解 MIPS 指令结构
3. 熟悉并掌握 ALU 的原理、功能和设计
4. 进一步加强运用 Verilog 语言进行电路设计的能力
5. 为后续的 CPU 设计实验打下基础

二、 实验器材与设备

2.1 装有 Xilinx vivado 的计算机一台

2.2 LS-CPU-EXB-002 教学系统实验箱一套

三、 实验分析与设计

3.1 实验原理

1. 本次实验设计的指令分析

本次实验共涉及 12 条 ALU 运算相关操作如下：

ALU 操作	相关指令
加法	add addu addi
减法	sub subu subi
有符号比较	slt slti
无符号比较	sltu sltiu

按位与	and andi
按位或非	nor nori
按位或	or ori
按位异或	xor xori
逻辑左移	sll
逻辑右移	srl
算数右移	sra
高位加载	lui

表 1 ALU 涉及的运算种类和对应的相关指令

而设计的 ALU 就应该要实现上述的十二种运算

2. ALU 的原理、功能和设计

ALU，全称位算数逻辑单元，是整个 CPU 中负责处理运算的模块，它可以简化为一个输入输出模型，输入两个操作数，一个操作码，给出一个结果，如下图所示：

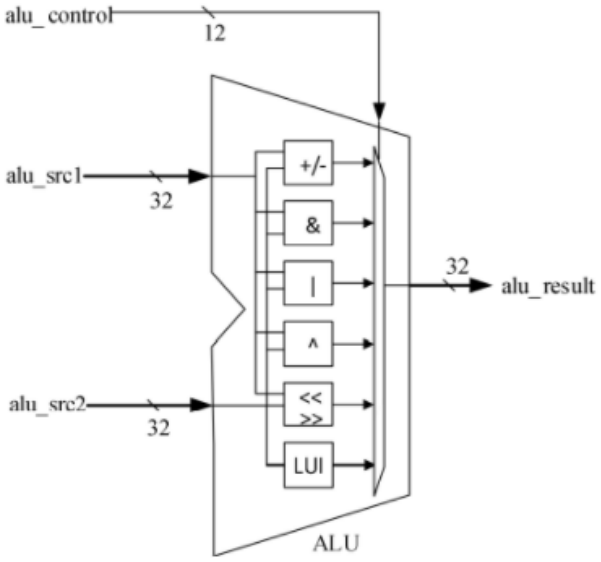

图 5 ALU 的原理图

3.2 端口设计

对于 ALU：

输入：12 位 OP 指令、32 位操作数 1、32 位操作数 2

输出：32 位运算结果

对于显示屏的交互：

SW18、19：00 表示输出 OP 指令；10 表示输入操作数 1；11 表示输入操作数 2；

显示屏中显示：

SRC_1：第一个操作数对应的 32 位数据

SRC_2：第二个操作数对应的 32 位数据

CONTR：12 位 OP 对应的数据

RESUL：32 位结果输出对应的数据

3.3 设计框图

1. 总体框架的设计框图如下：

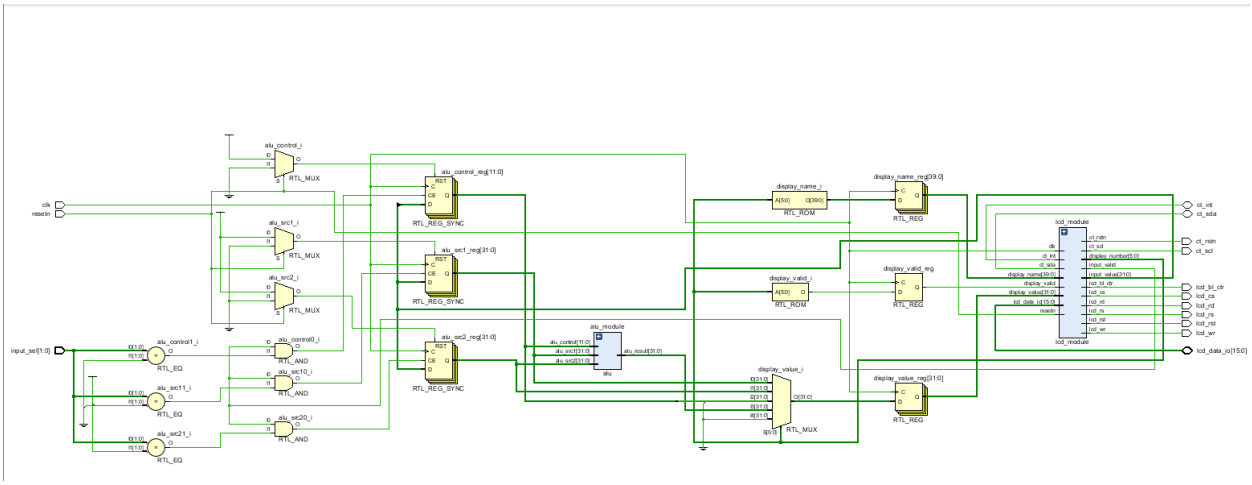

图 7 整体设计的电路图

2. ALU 的设计框图如下：

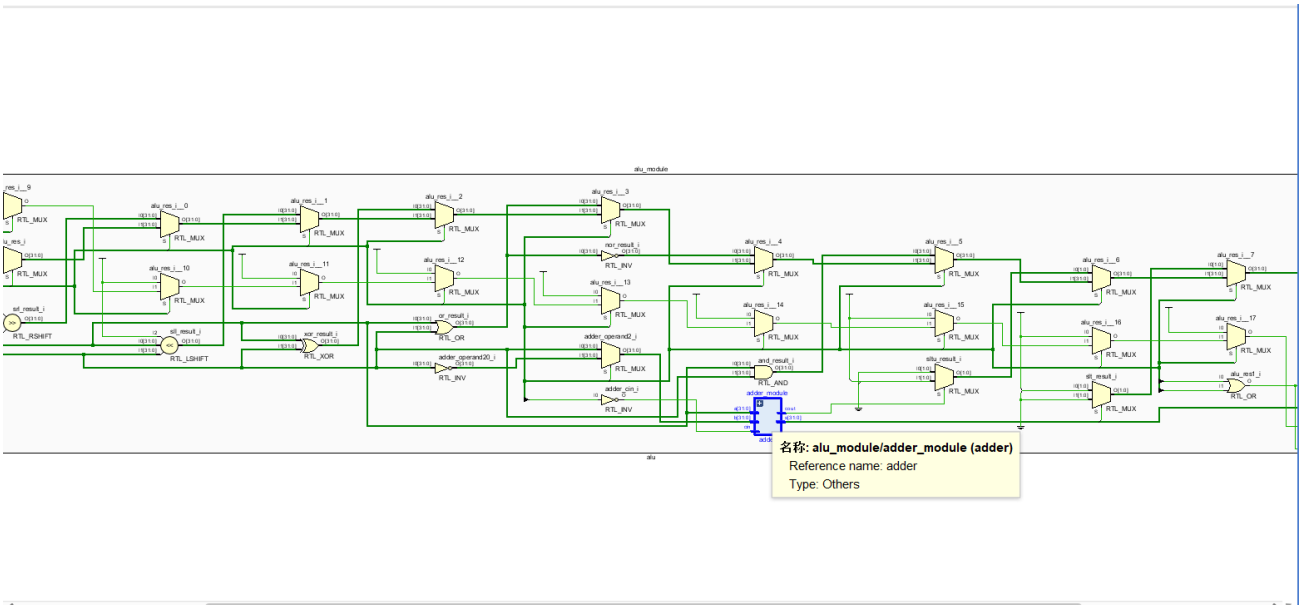

图 8 ALU 的设计框架(复用了加法器模块)

四、 实验步骤

4.1 代码的实现

为写好的 ALU 代码编写测试文件 tb.v，再其中为 12 种运算进行测试，然后加上外围的 alu_display.v 与显示屏实现交互，在 alu_display.v 中实例化 ALU，并编写对应拨码的对应输入，和显示屏的对应输出。最后加上实验所给的 lcd_module 和 XDC 管脚约束文件，便可以进行仿真综合烧录的工作了。

4.2 仿真与综合

首先对于 ALU 进行仿真测试，验证十二种运算的正确性。本人设计了如下所示的验证表来验证十二种运算的正确性：

表中的数据均为十六进制

ALU 操作	操作数 1	操作数 2	指令代码	结果
加法	00000011	0000222		
减法	00000400	0000028		
有符号比较	00000400	0000028		
无符号比较	00000028	0000040		
按位与	0000000A	000000C		
按位或非	0000000A	000000C		
按位或	0000000A	000000C		
按位异或	0000000A	000000C		
逻辑左移	00000001	0000004		
逻辑右移	00000100	0000008		
算数右移	80000010	0000001		
高位加载	00000000	0000BA1F		

表 2 十二种运算的验证表

通过 run simulation 得到的仿真波形图如下：

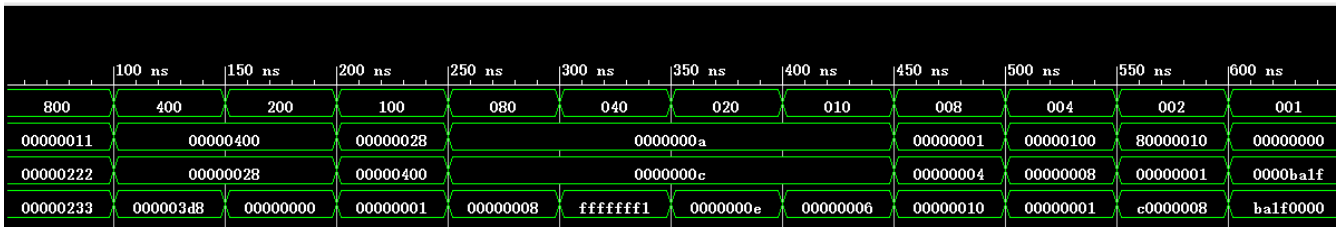

图 9 仿真波形图

对照来看结果完全正确，仿真正确

然后是综合 synthesis 的时延 report：

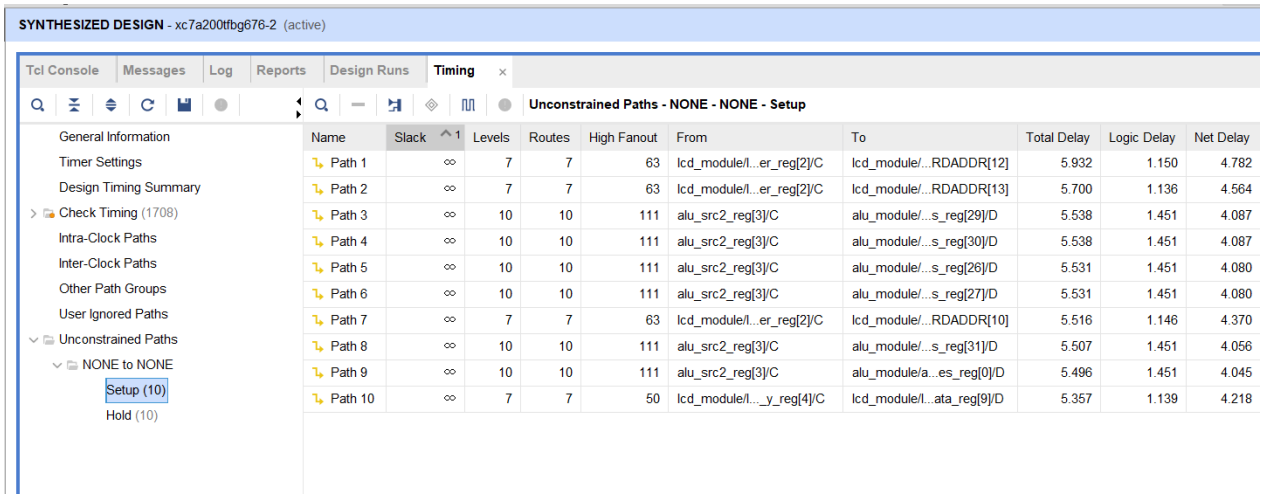

图 10 synthesis 综合的时延报告

然后是 implement 的 delay report：

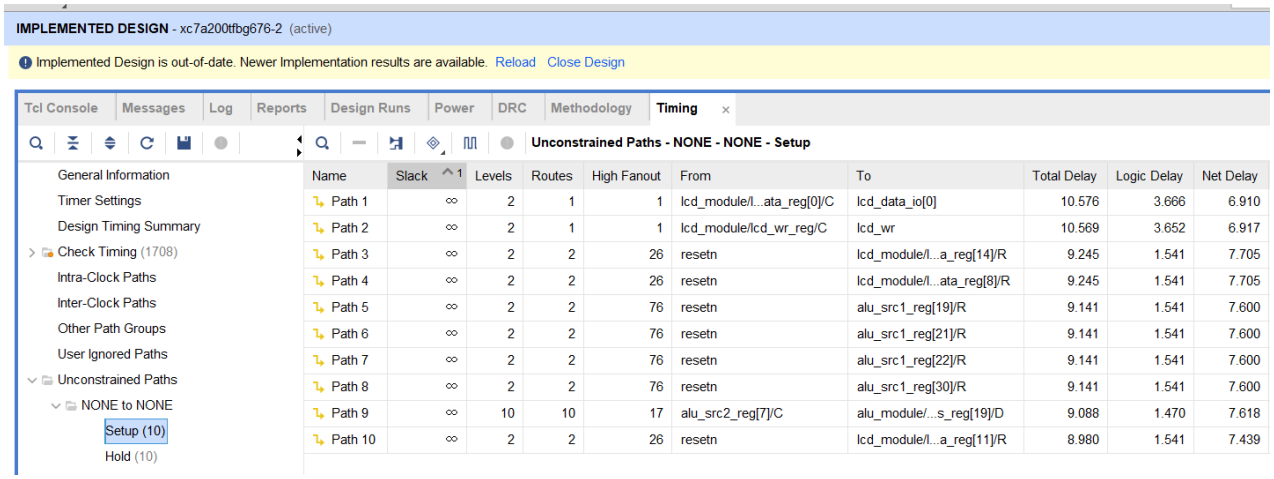

图 11 implement 实现的时延报告

可以发现综合和实现的 delay 差距还是非常大的

4. 3 上板验证与结果展示

最后将测试好的程序烧入龙芯教学实验平台中，通过显示屏的交互同样测试上述的 12 组验证实验，检测上板后的验证是否正确

五、 问题回答

5.1 基础问题

ALU 的结构如何，由哪些部分组成？不同的运算器之间是什么关系？结合具体指令，使用代码和仿真结果详细讲解 ALU 的原理和实现。

5.2 进阶问题

在本次实验中，有些指令的实现并没有它的含义看上去那么直接。另外，ALU 只负责指令中运算部分的执行，单看 ALU，并不能看到指令的完整实现。结合代码和仿真结果详细讲解 SLT 指令、位移系列指令的原理和实现，并结合具体指令谈谈对上述两句话的理解。

答：对于 SLT 比较的实现，其原理为两个数进行减法，再对结果进行处理得到比较的结果。具体实现的时候对于无符号的比较只需要查看减法结果的最高位即可，1 则说明操作数 1 小；0 则说明操作数 1 大；对于有符号的比较原理上是对符号先进行比较，符号不同则仅根据符号比较即可，符号相同则比较绝对值。具体实现的时候采用了高位补 0——也就是类似双符号位的方法，将两个操作数进行减法，得到的进位结果取反就得到比较的结果。

对于位移系列指令的实现则是直接调用已有的运算符>>、<<、>>>进行实现

对于第一句“有些指令的实现并有没有它的含义看上去那么直接”我的理解是：比如 SLT 比较的实现并不是直接使用了>或<的比较符号来实现，而是借助了加法器的减法运算来实现，再比如一些跳转指令，实际上的实现是通过拼接和位移运算来实现的。所以在真正考虑指令的实现的时候，不能仅从指令的意义去思考设计，我们还要从底层硬件的实现角度来考虑，才能更好的完成指令的实现。

对于第二句话“ALU 只负责指令中运算部分的执行，单看 ALU，并不能看到指令的完整实现”有些指令 ALU 仅负责了其中的部分语义，完整的实现还需要其他 CPU 的部件一同完成，比如读写操作 lw 和 sw，ALU 仅仅负责了寄存器地址的运算部分——也就是进行了加法，如果单看 ALU，好像只是完成了一个加法没有实际意义，但是从整个指令来看，该加法是在计算读写的寄存器地址，这才是加法的实际意义。

六、收获与体会

1. 熟悉了 ALU 的结构、原理和设计
2. 熟悉了 Verilog 语言的编写
3. 熟悉了 MIPS 指令集的架构和类型
4. 体会到指令的完成不仅仅需要 ALU 的运作，还需要整个 CPU 各个部分模块的协同工作才能正常完成
5. ALU 设计还有改进的空间，比如如何根据指令来实现不需要将

规定的运算都计算一遍后再进行最终结果的选择；如何减少计算的次数；如何做到并行运算等等。