重庆大学课程设计报告

课程设计题目:	MIPS SOC设计与性能优化
学　　院:	计算机学院
专 业 班 级:	计算机科学与技术01班
年　　级:	2019
学　　生:	杨小川　　何鑫
学　　号:	20174179　　20172375
完 成 时 间:	2017年　1月　1日
成　　绩:	
指 导 教 师:	钟将

重庆大学教务处制

项目	分值	优秀 $100 > x \geq 90$	良好 $90 > x \geq 70$	中等 $80 > x \geq 70$	及格 $70 > x \geq 60$	不及格 $x < 60$	评分
		参考标准					
学习态度	15	学习态度认真，科学作风严谨，严格保证设计时间并按任务书中规定的进度开展各项工作	学习态度比较认真，科学作风良好，能按期圆满完成任务书规定的任务	学习态度尚好，遵守组织纪律，基本保证设计时间，按期完成各项工作	学习态度尚可，能遵守组织纪律，能按期完成任务	学习马虎，纪律涣散，工作作风不严谨,不能保证设计时间和进度	
技术水平与实际能力	25	设计合理、理论分析与计算正确，实验数据准确，有很强的实际动手能力、经济分析能力和计算机应用能力，文献查阅能力强、引用合理、调查调研非常合理、可信	设计合理、理论分析与计算正确，实验数据比较准确，有较强的实际动手能力、经济分析能力和计算机应用能力，文献引用、调查调研比较合理、可信	设计合理，理论分析与计算基本正确，实验数据比较准确，有一定的实际动手能力，主要文献引用、调查调研比较可信	设计基本合理，理论分析与计算无大错，实验数据无大错	设计不合理，理论分析与计算有原则错误，实验数据不可靠，实际动手能力差，文献引用、调查调研有较大的问题	
创新	10	有重大改进或独特见解，有一定实用价值	有较大改进或新颖的见解，实用性尚可	有一定改进或新的见解	有一定见解	观念陈旧	
论文(计算书、图纸)撰写质量	50	结构严谨，逻辑性强，层次清晰，语言准确，文字流畅，完全符合规范化要求，书写工整或用计算机打印成文；图纸非常工整、清晰	结构合理，符合逻辑，文章层次分明，语言准确，文字流畅，符合规范化要求，书写工整或用计算机打印成文；图纸工整、清晰	结构合理，层次较为分明，文理通顺，基本达到规范化要求，书写比较工整；图纸比较工整、清晰	结构基本合理，逻辑基本清楚，文字尚通顺，勉强达到规范化要求；图纸比较工整	内容空泛，结构混乱，文字表达不清，错别字较多，达不到规范化要求；图纸不工整或不清晰	

指导教师评定成绩：

指导教师签名：

MIPS SOC设计报告

杨小川、何鑫

1 设计简介

MIPS指令集流水线CPU的Verilog实现，支持57条指令的AXI总线、52条指令Cache的运行。其中57条指令的axi总线(通过89个测试点)，并支持指令Cache以及数据cache。

1.1 小组分工说明

- 杨小川：负责逻辑运算、算术运算指令扩展、异常指令与特权指令、SOC外设。
- 何鑫：负责逻辑运算、算术运算指令扩展、连接axi总线、Cache设计。

2 设计方案

2.1 总体设计思路

前52条指令设计：

1、按照原有的数据通路设计扩展完成逻辑运算指令、移动指令。

2、增加Hilo模块，以存储乘除法结果。为避免数据冒险，Hilo寄存器的位置未ALU旁，写信号只可时钟上升沿有效，读为组合逻辑，随时可读，且接入ALU。

3、扩展完成所有的算数运算指令。

4、改写compare模块，使其变为广义比较器，并扩展完成分支跳转指令。

5、增加memsel模块，并扩展完成访存指令。memsel为选择模块，由于写内存指令中有按字写、按半字写和按字节写，故需要四位写使能信号控制写入的具体位置（我们设计为小端）。

后5条指令设计（内陷指令与特权指令）：

1.增加cp0寄存器存储并处理异常信息。由于取指、译码、执行阶段和访存阶段都有可能出现异常，故方案为每个阶段收集异常信息，直到M阶段（访存）再处理异常。故cp0寄存器放在M阶段，写使能在上升沿有效，读为组合逻辑。

2.将所有的异常信息（包括特权指令eret）传入exception模块，传出为exceptiontype（异常类型，按照异常优先级判断），将异常类型传入cp0模块，并进行相应处理。

2.2 Hilo模块设计

Hilo寄存器位置：E阶段，ALU旁。

模块实现功能：写使能仅在上升沿有效，读使能始终有效。当前位置不涉及数据前推。

2.3 memsel模块设计

sel模块位置：M阶段

实现功能：按照指令进行四位写使能信号的输出，实现按字写入，按半字写入以及按字节写入的功能；在当前指令为读内存时四位信号全部为1，在数据按字读出后再根据指令分配字节写回寄存器堆。为组合逻辑故不需要处理数据前推。

2.4 cp0模块设计

cp0寄存器位置：M阶段

实现功能：

1.写使能在上升沿有效，读数据时为组合逻辑。

2.当发生异常或者特权指令eret时，将exceptiontype传入冒险处理模块hazard，并由hazard传给pc新的地址。

3.数据冒险处理：（与lw指令的处理方式类似）

4.当mfc0指令后遇到一个运算指令（如add）并且同时要用到mfc0取出的值时，需要将cp0取出的数据前推至M阶段。

5.当mfc0指令后遇到分支跳转指令（如bne）并且需要用到mfc0取出的数据时，需要先将F、D阶段暂停一个时钟周期，再将取出的数据前推至D阶段。

3 设计过程

3.1 设计流水账

12月10日至12月12日15:30，共同添加完除了乘除法52条指令，通过各个部分的测试集；
12月13日13:08共同添加除法指令，把袁福焱的除法器添加在E阶段后，测试失败；
12月13日13:55共同将除法模块添加完毕，至此52条指令添加完毕并测试func_part1_2；
12月13日17:21共同将文件加入了soc测试文件，只能跑过第一个测试点；
12月16日1:00吕昱峰协助debug，改完后soc可以跑到43个测试点;
12月16日15:20至12月17日5:00小川添加特权内陷指并且57条指令添加完毕；

12月18日15:23通过89个测试点；
12月19日至12月28日何鑫添加完axi并且跑通89个测试点；
12月31日16:32何鑫添加完cache，但只能跑过前64个测试点。

3.2 错误记录

3.2.1 错误1

(1) 错误现象:除法结果正确但是无法写入hilo寄存器;
(2) 分析定位过程:
1.把hilo寄存器的输入hi_in与lo_in调出来后没有值，但是除法结果是正确的;
2.查看div_stall与div_valid，前者落后半个周期也正确;
3.找hilo寄存器的冒险问题，但是我们的hilo寄存器在E阶段因此没有冒险;
4.最后在hilo源代码中找到错误。
(3) 错误原因:hilo需要上升沿写入而源码时下降沿;
(4) 修正效果:通过52条指令测试;
(5) 归纳总结:给的代码需要查看并纠正，并不一定都符合自己的通路。

3.2.2 错误2

(1) 错误现象:soc在一条lw指令错误，没有从存储器取出值;
(2) 分析定位过程:
1.经过询问，连接soc之后取数据的addr若为bfaf改成efaf;
2.更改后仍然无法取出数据，查找存储器该地址但没有数据;
3.怀疑是soc本身的错误，取数据的指令地址时钟都没有错;
4.最终在吕学长的帮助下找到了错误并通过此处;
(3) 错误原因:soc外设的桥时钟边沿对不齐，导致无法取出数据;
(4) 修正效果:更改了时钟边沿后可以取出数据;
(5) 归纳总结:不能妄自更改外设的代码(最好不要)。

3.2.3 错误3

(1) 错误现象:特权指令mfc0后接beq时pc错误;
(2) 分析定位过程:
1.mfc0后接beq时没有跳转，但reference是跳转，导致了pc错误;
2.对比compare模块中的两个输入，发现不相同，但按照reference应当相同;
3.发现mfc0取出的值在beq指令马上要用到;
(3) 错误原因:未将cp0的数据前推至D阶段;

(4) 修正效果:在mfc0指令后的分支跳转指令均能正常执行。

3.2.4 错误4

(1) 错误现象:特权指令mfc0需要调用badaddr,但取出值有错;
(2) 分析定位过程:
1.寻找传入的badaddr指令的pc,找到目标位置;
2.引出sel模块里的badaddr(pc值),代码逻辑是没错的;
3.找到badaddr的相关值,即input的pc,发现只有一bit;
(3) 错误原因:原本32bit的badaddr的pc手误打成1bit;
(4) 修正效果:取出badaddr并且跑过89个测试点。

3.2.5 错误5

(1) 错误现象:AXI无法在lw指令取出数据;
(2) 分析定位过程:
1.d_ready信号在lw指令时变为1,说明信号是已经发出去了;
2.readdata信号却没有取出值,从stall_from_mem查找原因;
3.由于取数据暂停所有阶段,取指令暂停前两个阶段,所以确定这里错了;
4.D阶段memenD在前一条指令指令译码,导致暂停过程中没有取数据;
5.尝试将D阶段memenD信号暂停。
(3) 错误原因:存储器使能信号没有在取数据时没有作用,需stallD;
(4) 修正效果:可以通过AXI存数取数指令。

3.2.6 错误6

(1) 错误现象:连接AXI时43个点除法循环执行;
(2) 分析定位过程:
1.循环执行某条指令说明循环交替暂停;
2.查看div_stall信号与stall_from_if交替出现了;
3.这说明D阶段的除法流到了E阶段,导致除法和取值交替进行;
4.因此可以尝试div_stall只暂停E阶段;
(3) 错误原因:D阶段的除法流到了E阶段,导致除法和取值交替进行;
(4) 修正效果:AXI测试除法通过。

3.2.7 错误7

(1) 错误现象:连接AXI时异常指令咱取指令时刷新;

(2) 分析定位过程:

1.except_flush信号没有在执行指令时刷新;

2.F阶段取出的指令继续执行下去，结果必然是错误的;

3.尝试写状态机来延迟刷新，但是过于繁琐;

4.经过考虑决定在D阶段所有信号都disable，包括alucontrolD;

5.这样一来除法就仍然暂停前三个阶段，统一起来添加cache更方便;

(3) 错误原因:except_flush提前在取指时刷新，没有达到异常刷新的效果;

(4) 修正效果:AXI测试通过。

4 设计结果

4.1 设计交付物说明

4.1.1 目录结构

(1) 总体目录结构：

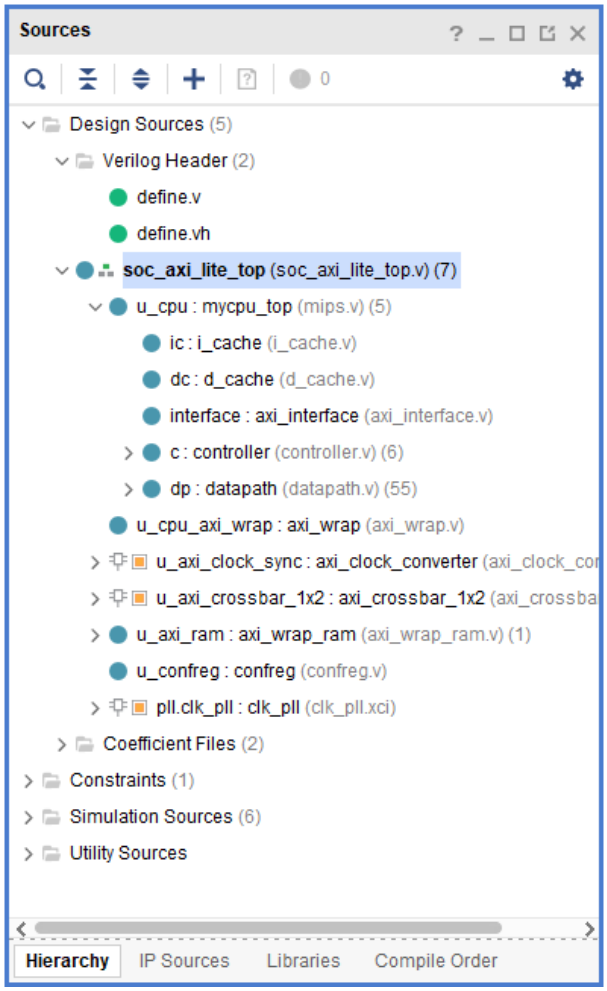

图 1: 项目结构示意图

(2) define.v是57条指令的宏译码定义,define.vh是异常处理模块的宏定义;

(3) mycpu_top是处理器的顶层模块，包含datapath、controller、i_cache与d_cache;

(4) controller中没有区分main_dec与alu_dec，指令统一译码。指令在D阶段时需要通过controller的译码，并且发出相应的控制信号(控制信号在D阶段暂停时无效)。下图中，当D阶段暂停时controls信号为0。并且给出了R型指令的译码示例;

```
assign {
    regwriteD , regdstD, alusrcD, branchD, memwriteD,
    memtoregD, jumpD, jalD, jrD, balD, memenD,//ctrl
    alucontrolD, //sig
    invalidD//instruction invalid
        } = controls;

always @(*)
begin
    controls<=0;
    if(~request_stall)begin
    case(instr_opD)
        `RTYPE:
            case(instr_funD)
                // arithmetic
                `add:  controls <= { `ctrl_Rtype, `sig_add,  1'b0 };
                `addu: controls <= { `ctrl_Rtype, `sig_addu, 1'b0 };
                `sub:  controls <= { `ctrl_Rtype, `sig_sub,  1'b0 };
                `subu: controls <= { `ctrl_Rtype, `sig_subu, 1'b0 };
                `slt:  controls <= { `ctrl_Rtype, `sig_slt,  1'b0 };
                `sltu: controls <= { `ctrl_Rtype, `sig_sltu, 1'b0 };
```

图 2: controller示例

(5) datapath为数据通路，其中包括FDEMW五个流水阶段、hazard、alu运算器、除法模块、hilo寄存器、异常处理模块以及选择器等;

(6) hilo寄存器存放的是乘除法的64位结果，将它安置在E阶段。原因如下：由于乘除法都在E阶段算出结果所以可直接将结果写入无需等待流水线。除法执行完无效的时刻(上升沿)写入，同时在异常处理时不写入。且不存在数据冒险的问题。

```
// hilo register
hilo_reg   hilo ( .clk     (clk        ),
                  .rst     (rst        ),
                  .we      (hilo_writeE && ~div_stallE && except_enM),
                  .hi      (hi_inE     ),
                  .lo      (lo_inE     ),
                  .hi_o    (hi_alu_inE ),
                  .lo_o    (lo_alu_inE ) );
```

图 3: hilo寄存器与除法模块

(7) hazard部分是数据通路非常重要的部分，它掌控着数据通路的暂停与刷新信号。其中包括数据冒险的前推，axi总线访存的暂停信号以及刷新信号等等。

(8) 其余部分为外部AXI总线设备。

```
assign stallF = lwstall | branchstall | inst_stall | mfc0stall | data_stall|div_stallE;
assign stallD = lwstall | branchstall | inst_stall | mfc0stall | data_stall|div_stallE;
assign stallE = div_stallE | data_stall;
assign stallM = data_stall;
assign stallW = 0;

assign flush_except = (excepttypeM != 32'b0);
assign flushF = flush_except;
assign flushD = flush_except;
assign flushE = lwstall | flush_except | branchstall | mfc0stall;
assign flushM = flush_except;
assign flushW = flush_except | data_stall;
```

图 4: hazard暂停与刷新信号

4.1.2 仿真、综合、上板演示的必要操作提示步骤

(1) 仿真步骤：在仿真文件中更改trace文件路径、u_axi_ram中更改相应的coe文件。

4.2 设计演示结果

1.添加完所有指令并连接axi总线后成功通过所有测试点(89个)。

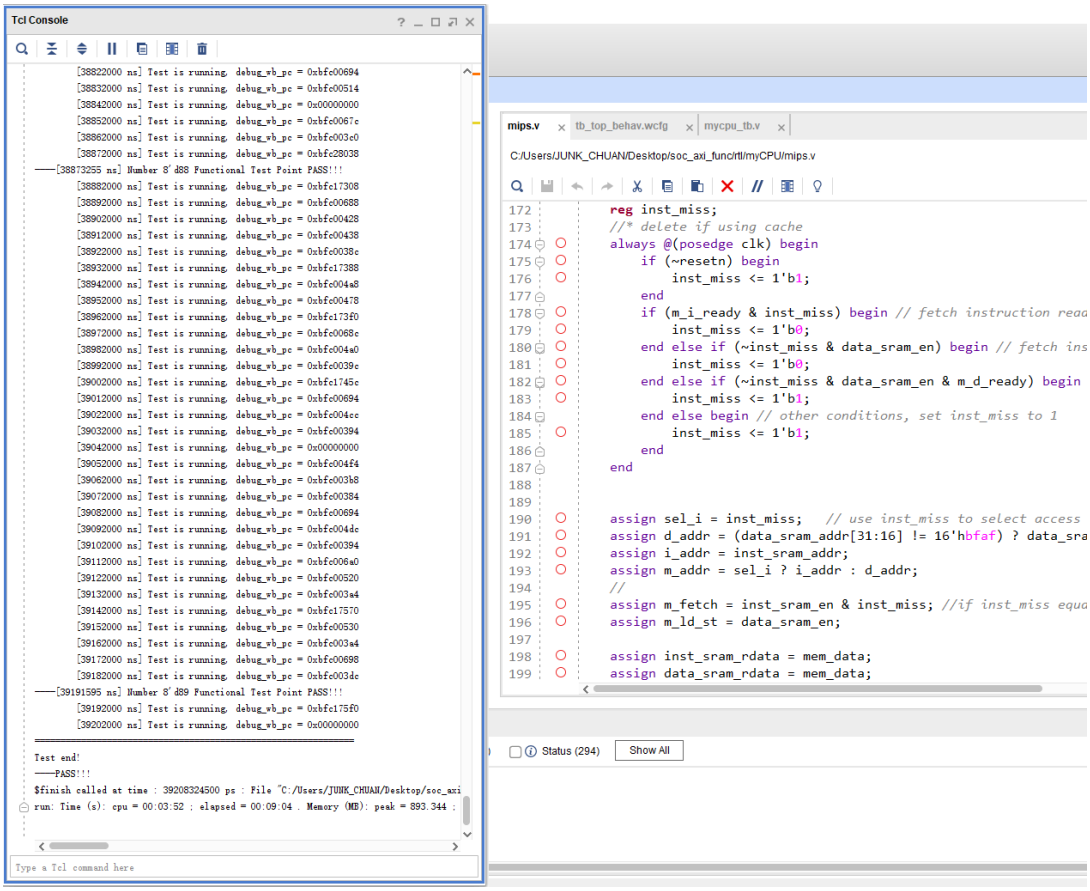

图 5: axi仿真结果：成功通过所有测试点

2.增加cache后成功通过64个测试点(实际上已通过75个测试点，出现在异常syscall的错误与吕学长描述的一样，可以认为cache已添加完毕)。

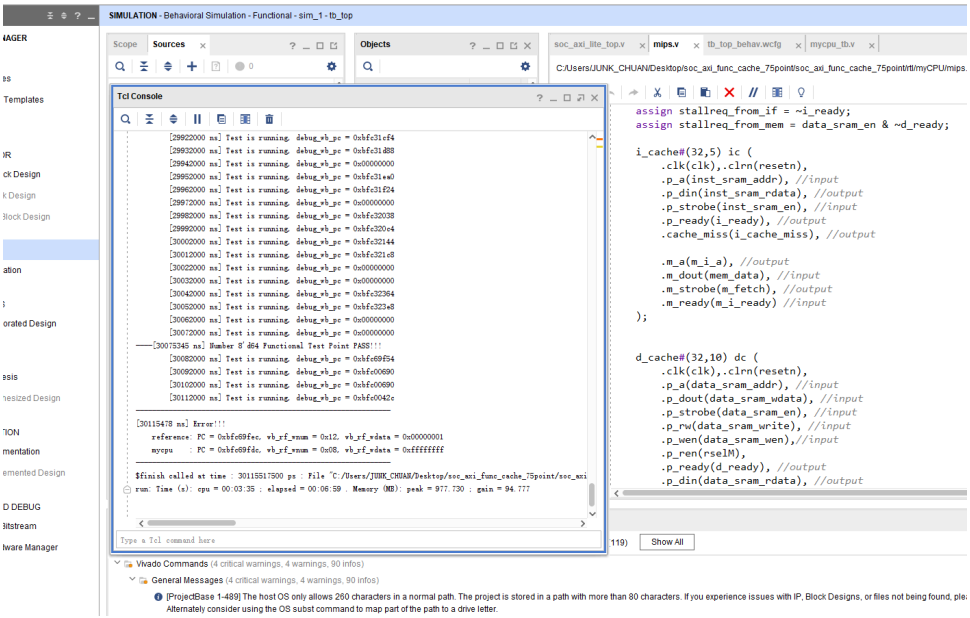

图 6: 加cache后仿真结果：成功通过64个测试点(即前52条指令)

3.处理器主频率测试

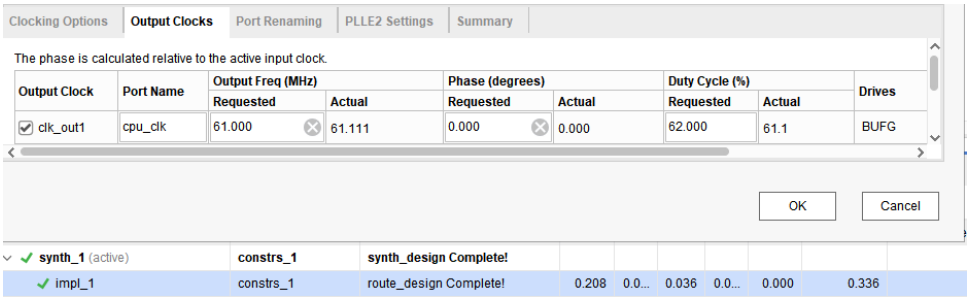

图 7: 实际主频率为62MHz

5 参考设计说明

1、袁福焱的除法器模块。

2、吕昱峰学长的axi接口模块。

6 总结

动手添加指令：

在课上学习CPU时只是很快速地过一遍ppt，对理论一知半解，在自己动手写了cpu后对每一条指令的执行步骤都有了十分详细的了解，并对每个指令的数据通路、CPU内部总线

都有了很好的掌握，以及清晰地了解数据冒险的处理方法。从sram到axi总线让我们知道了在不同硬件条件的支持下CPU与其他模块的协作。在sram条件下访存的速度极快，但同时造价也极高，当使用速度慢，价格较低一些的存储器结构时，由于访存需要的时间较长，故需要通过发送request请求和接受ready回应来实现握手的机制。

通过硬综达到境界的提升：

除此之外，这次硬件综合设计让我们熟悉了各种异常、中断的处理、执行以及CPU和外部系统总线的连接。令我们对计算机的整个体系结构有了进一步的了解。相较于上课时学习的理论知识，直接动手来的更为直接有效，虽然难度提升了不少，但我们对计算机系统组成的理解也更为深刻，对指令的执行理解也更为深刻，这些都是理论知识无法带给我们的。

共同感悟：

这次的硬件综合设计难度比较高，具有挑战性。为此和队友经历了数天的熬夜和通宵。虽然最后cache只跑通了前52条指令，在系统调用时出现了异常没有找到解决方法，心中有所遗憾，但我们还是对自己的付出和回报比较满意。也很感谢学长坚持课改、组织这次硬件综合设计，并且直播给大家培训相关知识。总而言之，这次的硬综还是让我们收货颇丰的。

7　供同学们吐槽之用。有什么问题都可以直接写在这。

可以完善一下axi的接口连接文档和cache模块的设计，我们自己填了一些坑(上面已写)。希望下一届学弟学妹们人均axi总线，cache卓越班都能加完！早日达到人均东部985水平