南京大学二队龙芯杯 CPU 设计报告

2018 年 8 月 20 日

队伍名称: 南京大学 2 队

成员: 刘志刚

欧先飞

沈明杰

邱子豪

指导老师: 李俊

蒋炎岩

目录

1 设计简介

1.1 项目结构简述

由于 verilog 语言设计本身的缺陷，其自身抽象能力不足，而且对类型系统的支持完全是 0，这使得我们在使用 verilog 进行 cpu 的编写的时候，麻烦程度不亚于手写汇编。所以在几经考量之下，我们决定用更高级的硬件描述语言 chisel 完成我们的开发流程。

chisel 是一门由伯克利大学开发的基于 scala 的硬件描述语言，由于起源自元编程语言 scala，chisel 拥有相当丰富的语法特性，同时也给我们的开发流程带来诸多便利，因此我们的项目也都是建立在 scala 之上，项目的目录布置也近似于 scala 的官方缺省方案。

项目的布局如下：

- build.sbt，这个是我们整个项目中 scala 源码文件的编译入口，在这个文件中定义了一些搜索规则，和 chisel 的支持包的下载站点，同时做了一些文件的时间戳检查，防止与 Makefile 做了重复的编译工作，文件中语法细节可参考https://www.scala-sbt.org/1.x/docs/index.html
- Makefile，由于 chisel 编写的 core 仅仅是项目的一小部分，除此之外还有很多的 uncore 部分需要处理，Makefile 具体干了哪些事，这里不展开细讲，如果想了解整个项目的编译流程，请详细阅读 Makefile，对于 Makefile 里面的一些特殊用法，可以参阅官方文档https://www.gnu.org/software/make/manual/make.pdf
- src/，这个是我们整个 cpu 的核心源码目录，包括 cpu 的 icache、dcache、l2 cache、流水段设置、cp0、分支预测器、协议转化桥、arbiter、crossbar，具体语言细节请参阅 scala 的官方手册http://people.cs.ksu.edu/~schmidt/705a/Scala/scala_tutorial.pdf，和 chisel 的官方手册https://chisel.eecs.berkeley.edu/api/latest/index.html
- emulator/，这是我们的模拟执行目录，在 src 目录下的所有源码文件最终会与 emulator 下的所有 verilog 文件及 cpp 文件一起编译链接，最终生成模拟器，模拟执行 cpu，并据此进行核内主要功能调试
- scripts/，这里包含一些脚本和工具，用于回归测试和差分测试
- uncore/，由于我们的 cpu 对多种板子均有支持，所以这个目录的存在用于多板子之间的解耦
- zynq_sw/，这个是用于快速上板所存在的目录，包括用于描述板上硬件的 dts 文件，用于加载程序的 loader，和用于引导 os 的 u-boot
- nscscc/，由于龙芯比赛规定，我们暂时无法修改龙芯比赛提供的项目文件和结构，因此我们没有将龙芯板子加到我们的 uncore 里面。而这个目录的存在是专门用于龙芯的上板：

nscscc/cpu_design.bd 用于生成我们的 cpu 在龙芯板子上的 uncore

nscscc/func-prj.tcl 用于生成符合龙芯比赛提交要求的 soc_axi_func 项目目录

nscscc/perf-prj.tcl 用于生成符合龙芯比赛提交要求的 soc_axi_perf 项目目录

nscscc/cpu_top.v 用于将我们使用到的 block design 包装成符合龙芯比赛接口的顶层 cpu

- nemu-mips32/，由我们组编写的 mips32 虚拟机，用于模拟执行 mips32 架构下的可执行文件，同时用于我们 cpu 的调试对比，尚未完工，所以暂时不会开源。

1.2 项目编译说明

我们已经将编译好的 verilog 文件放置在 soc 对应的目录下，无需特地在对项目进行编译。如果的确需要编译我们的源码，需要先安装一些工具：

- scala，scala 语言的编译器
- sbt，用于 scala 的项目管理工具
- verilator，用于编译 verilog 的 verilog 执行工具
- java，ubuntu 缺省的即可

以及一些需要设置的环境变量：

- AM_HOME，指向 AM 的目录，如果不需要运行差分测试和回归测试，可以不设置
- NPC_HOME，用于指向 noop-lo 所在目录的绝对路径
- NEMU_MIPS32_HOME，用于差分的 mips32 模拟器，如果不需要运行测试则不需要指定
- NSCSCC_HOME，用于指定大赛资源包所在的目录，支持第四版

在依赖包安装完毕之后可以通过 make submit 来生成功能测试模块所需的 verilog 文件和项目工程文件，生成完毕可在 outout/nscscc-submit 目录下找到对应的 vivado 项目目录，性能测试同理。如果需要仿真综合，还需要运行 make submit-soft 来构建仿真综合环境。

Makefile 中除了上述用于生成龙芯比赛的伪目标外，还定义了一些其他的伪目标，大概列举如下：

- emu，编译 core 生成模拟器模拟执行
- xsim，调用 vivado 的仿真模块进行仿真

- vivado，生成 zedboard 对应的项目文件并打开
- bit，综合生成 zedboard 的 bitstream 文件
- diff-insttest，用一份可确保正确的 mips32 虚拟机去差分模拟器，测试样例移植自今年龙芯比赛发布的资源。
- diff-cputests，对 cputests 进行差分，cputests 移植自我们 ICS 课程的测试程序，全部由 C 代码构成
- submit，生成符合龙芯杯比赛提交要求的项目目录
- loongson，用于生成符合龙芯比赛要求的 verilog 文件

2 设计方案

2.1 模块设计图纸

我们的 cpu 整体设计为 6 段流水线，如图1所示。由于初赛数据集较小，大部分数据集都可以直接装在 l1 cache，l2 cache 的设置在初赛中反而在某些数据集上起反效果，所以在最终提交的源码中，我们移除了这一组件。以及图中并未显式画出 flush 信号，这一信号用于分支跳转以及中断异常的冲刷。

图中所有的执行单元和写回单元 (wbu) 均有一条虚线连接到 ISU，这条虚线的含义是 bypass 信号，由于实际数据写回至少会晚上 3 个周期，为了能够更快的获取到寄存器数据，防止 ISU 因为等待数据就绪而空耗时间，项目添加了如图中虚线所示的 bypass 信号。

除了 core，为了能够对接到龙芯的开发板，我们加入 block design 用于处理外围引线，其中 block design 的设计如图2所示：

2.2 core 模块简介

如图1所示，图中列出了主要的 core 设计所涉及的模块，其中各个模块的含义大致描述如下：

a. IFU 为取指单元，其通过握手信号向 ICache 发送取指信号，为了防止照成关键路径，ICache 内部数据存储使用的是 block ram，所以 ICache 在收到数据的下一周期才能返回数据，同时 ICache 的内部实现了流水化，所以迟一个周期返回数据几乎没有影响。

b. IDU 为译码单元，用于对 IFU 取得的指令进行译码，以取得这条指令的操作数寻找模式，操作数扩展方式，访问的寄存器编号，写回的寄存器编号，所需发往的执行单元编号。

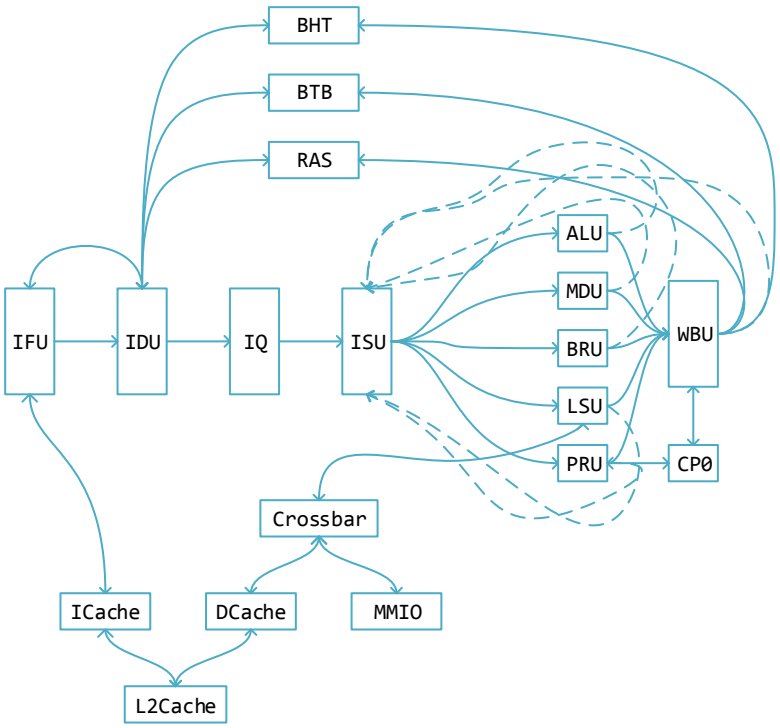
BHT
BTB
RAS
IFU
IDU
IQ
ISU
ALU
MDU
BRU
LSU
PRU
WBU
CP0
Crossbar
ICache
DCache
MMIO
L2Cache

图 1: CPU 的核内设计

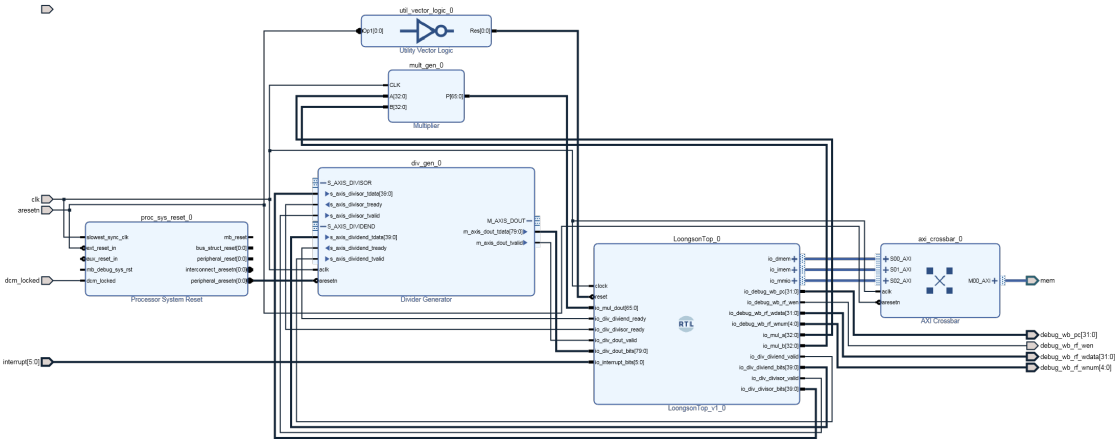

图 2: CPU 的核外布线

c. IQ 为指令队列，用于存储所有已经译码成功的指令，为了防止后端执行周期过长阻塞前端。

d. ISU 为分发单元，通过译码所得的结果读取寄存器，同时在寄存器读写约束不满足的时候进行必要的等待，索引转发结果。

e. ALU 为算术运算单元，用于完成加法、减法、移位等可在一个周期内出结果的运算指令

f. MDU 为乘除法单元，用于完成乘法和除法的运算，这一模块主体执行功能放在了核外的 IP 核上，而 MDU 内部逻辑主要是等待乘除法器的执行结果。

g. BRU 为分支跳转单元，通过对 ISU 传入的操作数进行计算，以获取跳转目标，同时对于条件跳转指令还需要计算一下是否跳转

h. LSU 为访存单元，用于 load store 指令的执行，若 dcache miss，还需要额外的周期等待访存结果

i. PRU 为特权指令单元，非特权指令不允许发送到这个单元，需要发送到这个单元的指令有 MFC0、MTC0、SYSCALL、ERET，同时对于取指异常的指令和译码异常的指令，也需要发送到这个单元用以下拉异常信号

j. CP0 为 0 号协处理器，用于完成中断异常主要计算功能

k. WBU 为写回单元，用于将执行单元获取到的操作数写回到目标寄存器

l. BHT、BTB、RAS 为分支预测单元，由于延迟槽的存在，整体的分支预测请求由 IDU 发出，并用于重定位 IFU，wbu 在写回寄存器时，也需要将 BRU 的计算结果更新到分支预测单元

3 设计结果

3.1 设计交付物说明

交付物目录结构基本符合大赛规定,其中 soc_axi_func 为功能测试的源码目录,soc_axi_perf 为性能测试的源码目录，score.xls 为性能测试的跑分结果。除此之外还有一个 noop-lo 目录，其中存放着我们整个 cpu 设计的实际源码。

在 soc 目录下的 rtl/myCPU 下总计有三个文件，其中 cpu_design.bd 是用于设计 cpu 外围结构的 block design 文件，LoonsgonTop.v 是由 noop-lo 目录下的源码编译生成的 verilog 文件，cpu_top.v 是一层对 block design 的 wrapper，用以将接口包装到符合大赛规定的接口。

3.2 设计演示结果

具体细节可以参见 score.xls，我们的 cpu 最高主频能够达到 73Mhz，跑分 42.09，但由于 vivado 的电路优化算法带有一定的随机性（本质上是随机算法），所以在每次综合实现的时候，并不总能到达 73Mhz，大部分情况下会因为时序关系不满足而导致负的 wns 值，因此我们将频率降到稳定的 71Mhz，经过多次综合实现确认，在这个频率下 vivado 的综合结果总能满足时序要求，但由于频率低了 2Mhz，跑分也相对应的低了 1 分，在 71Mhz 下，cpu 的跑分为 41.13 分。但为了稳定考量，我们选择将频率固定在 71Mhz，同时由于我们组所有人的电脑均为 ubuntu 系统，在 windows 下使用 vivado 综合实现是否能满足 71Mhz 的时序要求我们并未测试。

4 参考设计说明

在整个 CPU 的设计实现过程中，我们参考了 riscv 的 rocket-chip 的一些设计理念，如布线方式，对 scala 高级特性的使用。由于二者指令集并不一致，因而除了一些设计理念之外，只有一些架构无关的部分可以稍作参考。

在 uncore 之中，cpu 使用了总计三个 IP 核，由 vivado 提供的乘除法器 IP 核，用于将 cpu 出口的访存请求综合起来的 vivado 提供的 crossbar 模块。由于除法器需要特殊的方式进行 reset，所以我们还额外使用了 vivado 提供的 reset 模块用来初始化除法器。