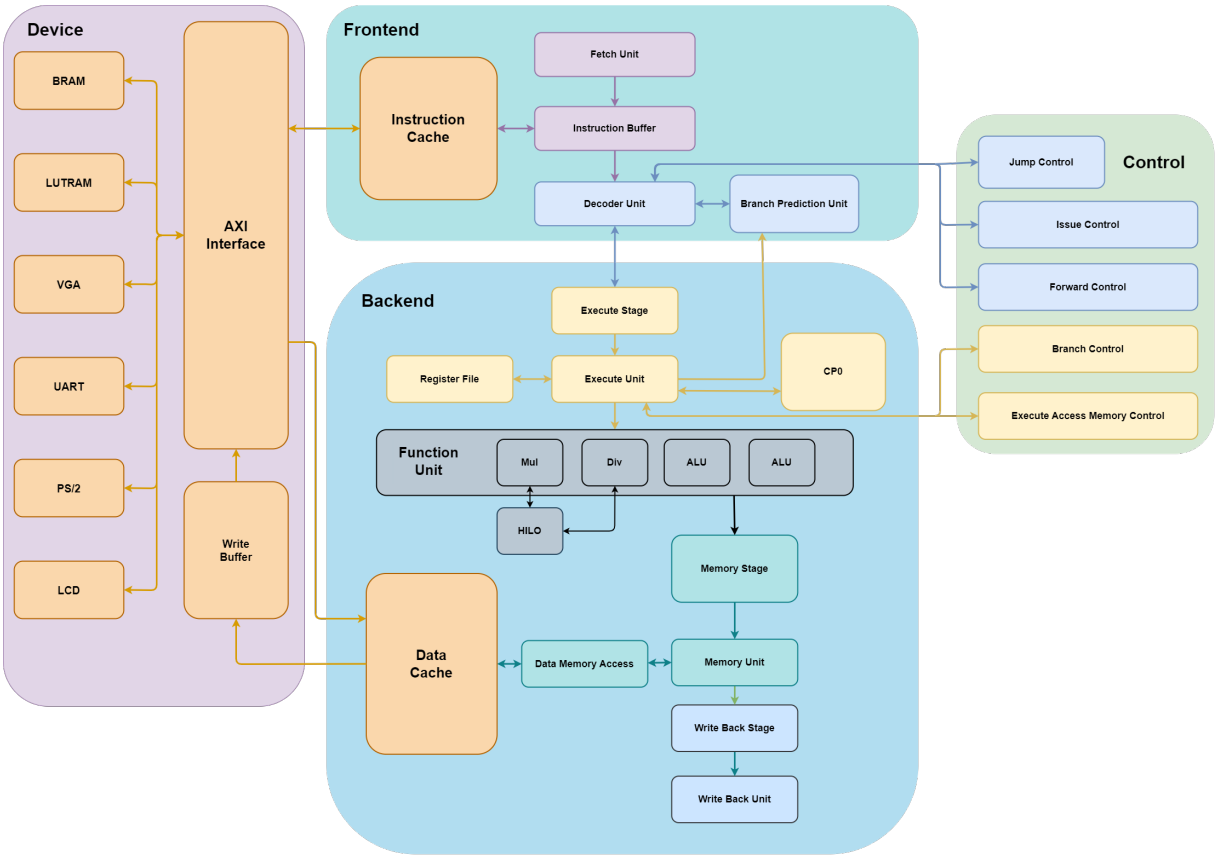
NSCSCC2023

杭州电子科技大学 PUA-MIPS 队

初赛设计报告



叶剑豪 奚力丰 胡致尧

目录

1 概述

我们基于Chisel语言设计并实现了一个MIPS32指令集架构的处理器:PUA-MIPS(Powerful Ultra Architecture MIPS)处理器。能够使用龙芯FPGA实验平台上的大部分外设，并且成功通过了龙芯杯官方框架提供的功能测试、性能测试和系统测试。该处理器可以运行PMON，U-Boot引导程序，uCore操作系统和Linux操作系统。

1.1 处理器亮点

- 顺序双发射六级流水线处理器PUA-MIPS(Powerful Ultra Architecture MIPS)。
- 处理器参数
  - IPC 1.1
  - 极限频率 82M
  - 顺序执行
  - 双发射
  - 六级动态流水（1取指，1解码，1~4执行，2访存，1写回）
  - 55%双发率
  - 8KB大容量Cache
  - 2bit分支预测
- 处理器设计
  - 指令集：MIPS32 Release1的除BranchLikely和浮点指令以外的所有指令。
  - 流水线结构：除法2-4级流水，两级访存。非对称双发射分为Path0和Path1，在遇到一些特殊指令时Path1暂停，Path0继续执行。
  - 异常处理：实现MIPS32 Release1中提到的所有异常和中断机制。
  - CP0：实现MIPS32 Release1中提到的所有CP0寄存器行为。
  - 缓存
    - ICache为两路组相连8KB设计，支持四指令读取窗口，状态机直接控制AXI交互。
    - DCache为两路组相连8KB设计，WriteBuffer使得写入不再制约处理器性能。
- Soc
  - CPU：PUA-MIPS CPU（含Cache）
  - DRAM支持：使用板载 DDR3 SDRAM作为主存
  - Flash：支持Flash读写
  - 串口：实现串口控制器以调整波特率
  - 以太网：使用IP核构建以太网控制器
  - GPIO：使用confreg组件控制LED，数码管等组件
  - 图像输出：实现VGA传输协议
  - PS/2：支持PS/2 键盘/鼠标输入
- OS
  - 支持PMON，UBoot引导程序，支持uCore操作系统。
  - 支持最新Linux主线v6.5-rc4版本。
  - 支持OS驱动所有外设。
- Chisel语言
  - 参数化设计，支持流水线长度、取指宽度、访存宽度等参数的配置。
  - KISS设计理念加持，单个文件大小不超过200行。
  - 面向对象带来更高的自由度，同一种算法，四种不同实现随意组合。

1.2 参考资料

- NSCSCC往届处理器设计
  - nontrivial-mips 给了我们如何启动Linux操作系统的良好示范。
  - TrivialMIPS 中有许多顺序处理器设计的技巧。
  - amadeus-mips 帮助我们更好的使用Chisel语言。
  - UltraMIPS 给了我们Cache设计的良好示范。
  - CDIM 为我们提供了高效的测试框架。
  - GenshinCPU 有许多频率优化的设计方法。
  - zenCove 在高级语言的使用上启发了我们。
- 开源处理器
  - 香山处理器 给了我们很多架构设计上的启发。
  - 果壳处理器 一个优秀处理器的示范。
  - dinocpu 一个以Chisel语言为基础开设的处理器设计课程，使用基于Scala的交互式调试框架给了我们很多帮助。
- 书籍
  - 计算机组成与设计：硬件/软件接口
  - 计算机体系结构：量化研究方法(第6版)
  - CPU设计实战
  - 超标量处理器设计
  - 自己动手写CPU

2 处理器详细设计

PUA-MIPS是顺序双发射结构设计，支持MIPS32 Release1扩展，PUA-MIPS处理器前端流水级包括分支选择单元、取指单元、指令缓冲等单元，顺序取指。后端包括译码、分支预测、运算单元、寄存器堆、访存控制和写回等单元。顶端有一个控制单元，负责控制各个流水级之间的同步和冲突。缓存分为ICache、DCache、TLB等模块。

2.1 前端

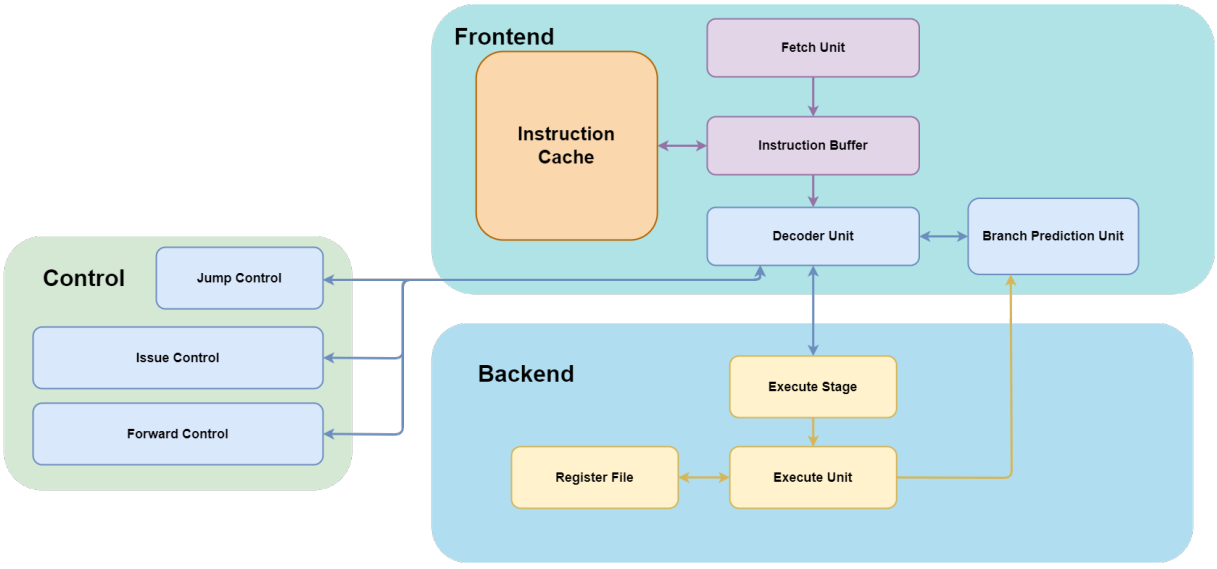


图 2: Frontend Design

2.1.1 取指

前端流水级包括分支选择单元、取指单元、指令缓冲等单元。我们通过取指队列（InstBuffe）实现前端和后端的充分解耦，使得开发过程能够尽量独立。为了保证后端流水能全速运行，前端必须保证取指队列内一直有足够的指令，因此我们采用了四宽度取指。取指单元（FetchUnit）根据后端的信息决定是否跳转并送给指令缓存（Instruction Cache）接下来四条指令的地址，指令缓存每行8条指令，当取指的起始地址为后三条指令时取至行尾。每条指令都附带指令的有效位。

2.1.2 译码

译码阶段中进行双路译码。从取指队列中取出两条指令，根据指令产生控制信号。同时为了减少分支跳转指令产生的对流水线刷新的影响，我们添加了静态分支预测单元，使用传统2bit分支预测器对PC进行预测。译码会得到指令在执行阶段的阻塞情况，这部分信息会反馈给取指队列以控制取指队列的出队。随后，通过四端口读两端口写寄存器堆，读出寄存器对应的值并发送给执行阶段。

2.2 后端

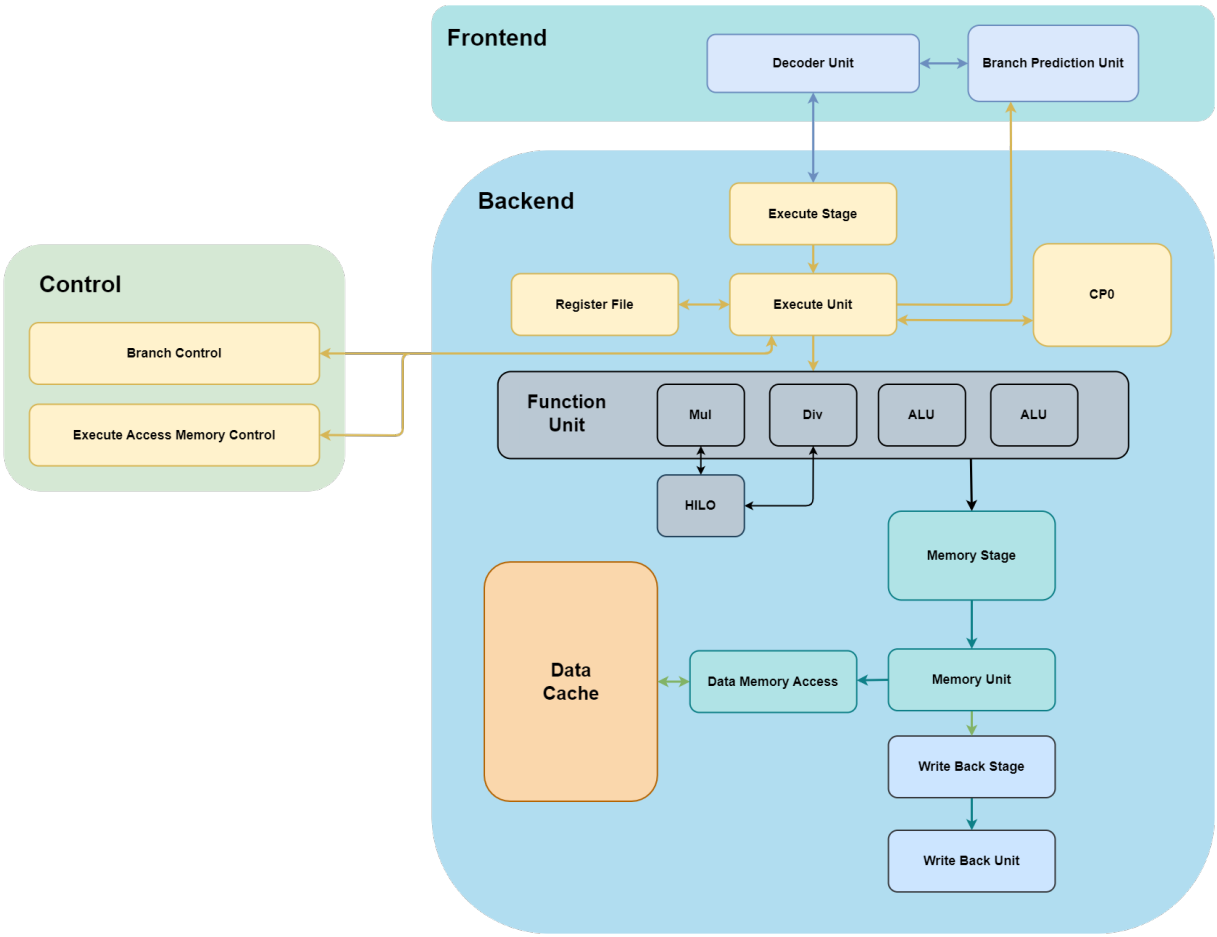


图 3： Backend Design

2.2.1 执行

执行阶段中对根据译码阶段的信号对数据进行运算。我们使用IP核进行乘除法的运算，其中乘法两周期，除法四周期。通过握手信号与执行单元进行通信，同时在功能单元（Function Unit）中对两条路径上的运算进行仲裁。同时，针对分支指令，我们需要在该阶段判断分支指令是否成功，并将判断的结果返回分支预测单元。针对访存指令，由于访存需要两个周期，因此我们在执

行级同时需要向数据缓存（Data Cache）发送访存信号。在这一阶段，我们还需要进行TLB的第一级地址转换来减少TLB的时延。这一阶段还要对异常信号再CP0模块中进行集中处理。

2.2.2 访存

访存阶段将完整的信息传递给数据缓存，以避免数据缓存将错误的地址传输给AXI总线。在下一拍，访存级获得数据，并将数据分配给Path0以及Path1。

2.2.3 写回

写回阶段进行双发射写回，若遇到了读写冲突则进行前递操作。

2.3 缓存

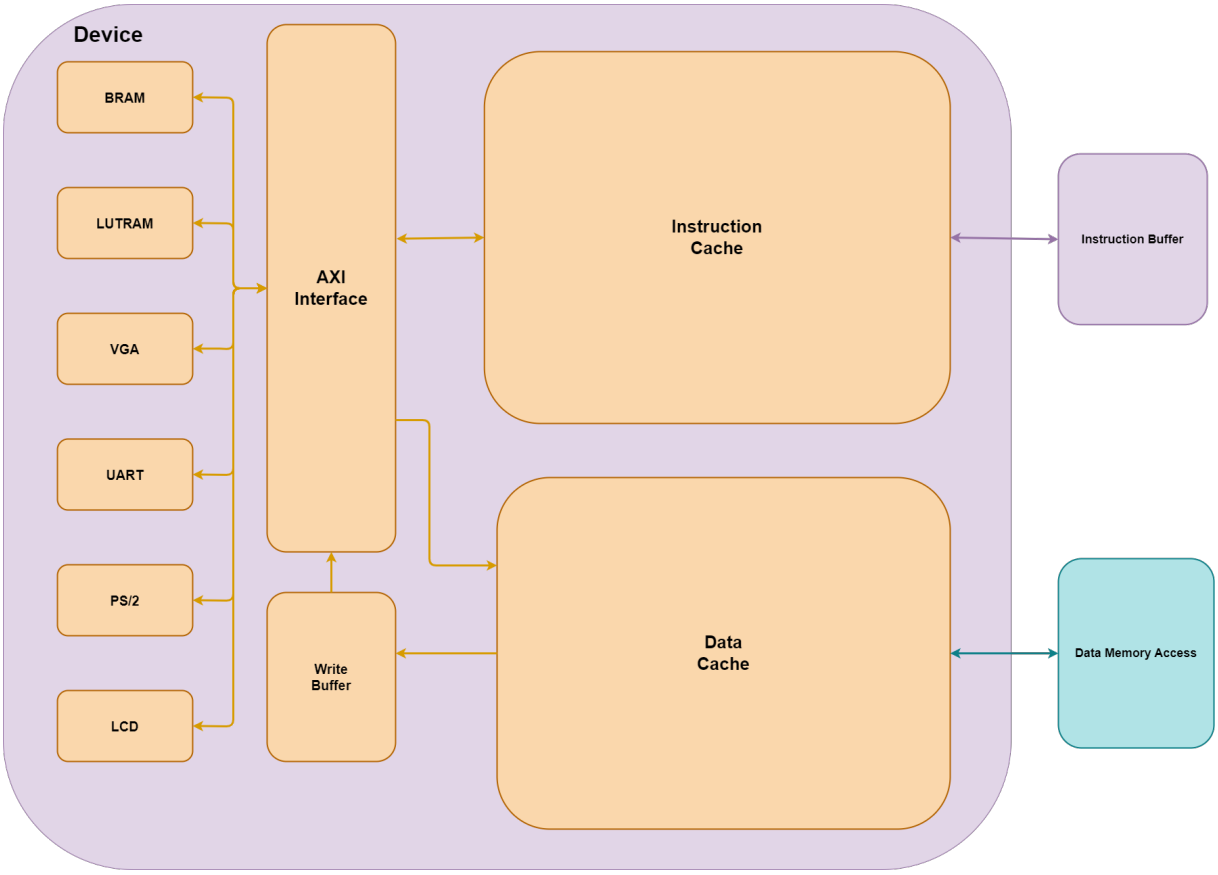


图 4: Cache Design

在PUA-MIPS中，设计实现了指令缓存和数据缓存。

- VIPT
- AXI突发传输
  - 十六字突发
- 两级TLB平衡延迟
- 两路8KB大缓存
- 大小可配置
- 指令缓存
  - 四宽度取指
  - 单行八条指令
- 数据缓存
  - 写队列解耦读写带来的巨大延迟

2.4 TLB

我们设计了两级TLB以解决起操作系统时TLB带来的巨大延迟。

3 模拟器设计

我们从NEMU（NJU Emulator）和CEMU（CQU Emulator）中获得启发，设计了我们自己的模拟器HEMU（HDU Emulator）。该模拟器基于Rust语言的强大生态，兼有NEMU的优雅设计模式和CEMU对比赛测试的强大支持，可以做到对指令运行状态的ISA级仿真，支持QEMU、NEMU、Xiangshan作为DIFFTEST的Reference。支持RISCV64，RISCV32，MIPS32指令集架构。我们使用HEMU来对指令执行、Cache策略进行统计发现可能的优化方向，并且对我们的处理器进行差分测试。

基于现有的多种模拟器，我们开发了HDU BOARD作为一种抽象的主板。这个主板可以集成任意的SOC，并且能够像真实的主板那样插拔不同的处理器。