ZenCove 初赛设计报告

崔轶锴 张为 王拓为

August 5, 2022

1 项目概述

1.1 项目背景

本项目是第六届“龙芯杯”全国大学生计算机系统能力培养大赛（NSCSCC 2022）的参赛作品。

项目实现了基于 MIPS32 指令集的乱序多发射 CPU ——ZenCove，并以比赛提供的 FPGA 实验平台为基础，实现了完整的 CPU 微架构。经测试，可以通过全部功能测试、性能测试，并运行监控程序

1.2 项目概述

1.2.1 硬件语言

本项目使用非传统硬件描述语言——基于 Scala 构建的硬件描述语言 SpinalHDL 编写。

1.2.2 设计模式

ZenCove 借鉴了 Vexriscv 的设计模式，将包括 ALU、寄存器文件、PC 组件等几乎所有的模块作为插件 (Plugins) 在流水线中添加或移除。得益于此，各部分功能代码不再分散在各个文件中，而是位于单独的插件文件中，避免了多个代码文件间的耦合，便于协同开发，也实现了 CPU 架构的高度可配置性。

1.2.3 CPU 架构

ZenCove 采用了乱序多发射的基本架构，在标准参数下为 5 发射。流水线结构可以大致分为前端和后端两部分，前端包括取指令 1、取指令 2、指令译码、寄存器重命名、指令派发，后端包括发射、读寄存器、执行、写回。

ZenCove 实现了 92 条 MIPS 指令，基本涵盖了单核 MIPS32 release 1 处理器除浮点外所需的指令。同时实现了 23 个 CP0 寄存器以提供对于操作系统所必须的功能支持。

ZenCove 支持基于 TLB 的虚拟地址转换，同时实现了指令和数据缓存以加快取指和访存。

ZenCove 支持硬件和软件中断，同时实现了精确异常。

1.3 开发平台

1.3.1 硬件平台

本项目使用的硬件平台是比赛提供的龙芯体系结构教学实验箱（Artix-7），其核心是一块基于 FPGA 芯片的嵌入式系统开发板，型号为 XC7A200T-FBG676。此外，平台包含了 DDR3、SRAM、NAND、Flash 等丰富的外设资源。

1.3.2 软件平台

本项目的开发共分为两个部分。

第一部分是基于 mill 0.10 版本构建的 scala 项目，在 vscode/JetBrain Intellij IDEA 中进行开发，用于生成 verilog 代码

第二部分使用 Xilinx Vivado 2019.2 Web HL Edition 作为开发 IDE 平台。

两部分共同使用 GitLab-CI 进行自动化集成。

1.4 参考资料

本项目的设计和开发过程中参考和借鉴了包括但不限于以下资料：

- 超标量处理器设计. 姚永斌
- The Berkeley Out-of-Order Machine (BOOM)

  文档：https://docs.boom-core.org/en/latest/sections/intro-overview/boom.html

  仓库：https://github.com/riscv-boom/riscv-boom
- SpinalHDL 文档：https://spinalhdl.github.io/SpinalDoc-RTD/master/index.html
- Vexriscv 仓库：https://github.com/SpinalHDL/VexRiscv
- NonTrivial-MIPS 仓库：https://github.com/trivialmips/nontrivial-mips
- LLCL-MIPS 仓库：https://github.com/huang-jl/LLCL-MIPS
- MIPS® Architecture For Programmers I, II, III. Imagination Technologies LTD.
- 各外设使用手册. 相关厂商

2 CPU 架构

2.1 基本架构

ZenCove 采用乱序多发射的基本架构，发射数量为可配置的参数。在标准参数下为 5 发射。乱序多发射流水线大致可以分为前端流水线与后端流水线，以重排序缓存 (ROB) 和发射队列为分隔。为了满足精确异常的要求，在前端与后端执行完指令后，由 ROB 异步地将指令提交。

2.1.1 前端

前端流水线共五级，包括两级取指令、指令译码、寄存器重命名、指令派发。

取指令 1 这一阶段进行 PC 的 TLB 查询与地址翻译，并进行 I-Cache 查询的第一阶段。分支预测在此阶段查询分支目标缓存 (BTB) 与预测历史表 (PHT) 。

取指令 2 这一阶段进行 I-Cache 查询的第二阶段，并对 Cache 缺失的情况进行填充。分支预测在此阶段形成预测结果，预测目标 PC。这一阶段之后将指令装入**取指令缓冲 (Fetch buffer)** ，这样可以平衡取指令速度与指令派发速度，也将取指令速度“削峰填谷”，减少气泡与暂停的出现。

指令译码 指令译码阶段取出取指令缓冲中的若干条指令（宽度为可配置参数），进行译码，翻译成微码 (Micro op, μop) 。由于 MIPS 为 RISC 架构，我们保证每条指令只生成一个 μop。

寄存器重命名 ZenCove 采用显式重命名，即只采用物理寄存器重命名逻辑寄存器，将寄存器映射存储在重命名映射表 (Rename alias table, RAT) 中。由于精确异常回滚的需要，重命名映射表需要存储推测性的和架构性的两份。采用 FIFO 管理的空闲寄存器列表 (free list) 管理可以被分配的物理寄存器，在此阶段给需要写寄存器的指令分配物理寄存器。若需要分配而 free list 已空，则需要暂停此阶段。

指令派发 此阶段根据指令类型将指令装入不同的发射队列。ROB 的装入可以在此阶段，也可以在寄存器重命名阶段。

2.1.2 后端

后端流水线数量等于发射宽度。在默认配置下，有三条不完全对称的整数流水线，一条乘除法流水线，一条访存流水线，组成 5 发射。所有后端流水线都由发射、读寄存器、执行、写回 4 个主要阶段组成，其中访存流水线的执行由于 D-Cache 访问的需要扩展成了两个阶段。

发射 ZenCove 采用分布式的发射队列，有整数发射队列、乘除法发射队列、访存发射队列 3 个发射队列。整数发射队列有三条整数流水线共同取指令发射，乘除法与访存各自面向一条流水线。整数发射采用纯乱序的方式，三条流水线共同选择避免重复发射；乘除法与访存目前均采用纯顺序的方式，因此它们都是 FIFO 队列，但是不同发射队列的指令之间仍然构成了乱序。整数流水线在此阶段唤醒整数发射队列中的指令，使得在前传下整数指令写后读 (RAW) 相关可以背靠背执行。

读寄存器 此阶段读物理寄存器堆，并进行寄存器的前传。整数流水线在此阶段唤醒与它 RAW 相关的后续指令，使得所有指令均可以间隔一个气泡读到整数指令的结果。

执行 每一条整数流水线均包括相同的算术逻辑单元 (ALU)，而仅有一条整数流水线包括比较器，读协处理器 0 (CP0) 的功能。乘除法流水线在此阶段计算乘法、除法，以及处理 Hi/Lo 寄存器的读写。访存流水线的执行第一阶段进行 TLB 查询与地址翻译，并进行 D-Cache 查询的第一阶段。第二阶段进行 D-Cache 查询的第二阶段，并对 Cache 读缺失的情况进行填充。同时对于 store 指令和 uncached 地址段的 load 指令，由于执行会对处理器状态造成不可恢复的改变，装入 **store buffer** 等待提交后执行。从 store buffer 发射的指令，在此阶段进行 D-Cache 的写，进行写缺失填充，或对 uncached 情况进行直接设备访问。

写回 此阶段将执行结果分别写入物理寄存器和 ROB。

2.1.3 指令提交

ROB 读取未提交的指令，判断指令是否可以提交，按顺序提交至多提交宽度条指令，并对分支预测错误、异常与中断、处理器状态回滚、store buffer 激活等需要提交时处理的情况进行处理。

为了降低硬件复杂度同时简化特殊情况的处理，我们限制了只有 0 号提交口可以处理特殊的指令（如 Conditional Move, Uncached Load 等）和特殊的情况（如异常，分支预测错误等需要重置处理器状态的情况）。

2.2 指令支持

ZenCove 共实现了 92 条 MIPS 指令，基本涵盖了单核 MIPS32 release 1 处理器除浮点外所需的指令，按照功能划分如下：

- 算术指令：ADD, ADDI, ADDIU, ADDU, CLO, CLZ, DIV, DIVU, MADD, MADDU, MSUB, MSUBU, MUL, MULT, MULTU, SUB, SUBU
- 逻辑指令：AND, ANDI, NOR, OR, ORI, SLL, SLLV, SRA, SRAV, SRL, SRLV, XOR, XORI
- 访存指令：LB, LBU, LH, LHU, LW, LWL, LWR, SB, SH, SW, SWL, SWR
- 分支指令：BEQ, BGEZ, BGEZAL, BGTZ, BLEZ, BLTZ, BLTZAL, BNE, J, JAL, JALR, JALR.HB, JR, JR.HB
- 自陷指令：TEQ, TEQI, TGE, TGEI, TGEIU, TGEU, TLT, TLTI, TLTIU, TLTU, TNE, TNEI
- 移动指令：LUI, MFHI, MFLO, MOVN, MOVZ, MTHI, MTLO, SLT, SLTI, SLTU, SLTIU
- 特权指令：BREAK, CACHE, ERET, MFC0, MTC0, SYSCALL, TLBP, TLBR, TLBWI, TLBWR, WAIT
- 其他指令：PERF, SYNC

由于 axi 总线不支持以 3 byte 为访问粒度进行读写，故 unaligned load/store 指令（LWL/LWR/SWL/SWR）不支持以 uncached 方式进行读写。

2.3 协处理器 0

ZenCove 共实现了 23 个 CP0 寄存器以提供对于操作系统所必须的功能支持，按照地址排序如下：

名称	地址	名称	地址
Index	0, 0	IntCtl	12, 1
Random	1, 0	SRSCtl	12, 2
EntryLo0	2, 0	Cause	13, 0
EntryLo1	3, 0	EPC	14, 0
Context	4, 0	PRId	15, 0
PageMask	5, 0	EBase	15, 1
Wired	6, 0	Config	16, 0
BadVaddr	8, 0	Config1	16, 1
Count	9, 0	Config2	16, 2
EntryHi	10, 0	Config3	16, 3
Compare	11, 0	ErrorEPC	30, 0
Status	12, 0		

2.4 内存管理

ZenCove 支持基于 TLB 的虚拟地址转换，TLB 项数可配置。依据 MIPS 标准，各对应段的映射关系和访存行为如下：

名称	地址段	User 态可访问?	固定映射?	可 cache?
kuseg (ERL=1)	[0x0000_0000, 0x7FFF_FFFF]	N/A	是	否
useg/kuseg	[0x0000_0000, 0x7FFF_FFFF]	是	否	根据 TLB
kseg0	[0x8000_0000, 0x9FFF_FFFF]	否	是	K0=1 则是
kseg1	[0xA000_0000, 0xBFFF_FFFF]	否	是	否
sseg	[0xC000_0000, 0xDFFF_FFFF]	否	否	根据 TLB
kseg3	[0xE000_0000, 0xFFFF_FFFF]	否	否	根据 TLB

注意，kseg0 段与 kseg1 段映射到相同的物理地址段。在 SoC 上，这一段部分为 MMIO 寄存器，因此使得一些 kseg0 地址的访问错误地以 Cached 方式访问了 MMIO。虽然程序正常执行不会出现这样的访问，但是乱序处理器的前瞻执行会导致访问随机的虚拟地址。因此我们在实际实现时根据 SoC 的实际地址分配对 kseg0 一部分进行了屏蔽，避免了 MMIO 被以 cached 方式映射到。

2.5 异常中断

ZenCove 共实现了 12 个异常编号、共 17 种异常的处理，按照编号排序如下：

名称	编号
Interrupt	0
TLB modification	1
TLB invalid/refill (load or instruction fetch)	2
TLB invalid/refill (store)	3
Address error (load or instruction fetch)	4
Address error (store)	5
Syscall	8
Breakpoint	9
Reserved instruction	10
Coprocessor Unusable	11
Arithmetic Overflow	12
Trap	13

2.6 缓存设计

ZenCove 采用可配置组相连缓存，默认配置下 I-Cache 大小 8KB，2 路，Cache line 为 32B。D-Cache 大小 8KB，2 路，Cache line 为 64B。均为两级流水线访问，I-Cache 单周期默认配置取 16B，满足取出同一 Cache line 至多 4 条指令的要求。Cache 采用 VIPT，无重名。均采用 LRU 作为替换策略，D-Cache 为写回 cache。

2.7 分支预测器设计

分支预测器一共三部分，分支目标缓冲 (BTB)，分支方向预测器和基于返回地址栈的调用返回预测器。

同时，由于插件的设计模式的便利性，只需要简单的修改就可以切换到不同的分支方向预测器以及与其匹配的分支目标缓冲。在实际实验中我们尝试了两位饱和计数器局部预测器，Gshare、Gselect 全局预测器，以及局部预测和全局预测的混合预测器。最终采用 Gselect 全局预测器，可以平衡分支指令间的冲突和全局历史的长度。

ZenCove 的 BTB 大小和具体参数可配置，默认配置下可存储 1024 项，直接映射，每周期可以任意取连续 4 项，对应连续 4 个 PC 的分支信息。

默认配置下每项存储以下信息：

- tag: 用于判断信息是否与地址匹配
- target：目标地址
- isCall: 是否为函数调用指令
- isRet: 是否为函数返回指令

2.7.1 Gselect 分支方向预测器

默认参数下，Gselect PHT 共有 8192 个表项，每个表项对应 2 位饱和计数器。记录 5 位长度的全局历史，与 PC 的 9 2 位拼接查询表项，形成预测结果。

分支错误信息在 ROB 对应指令提交进行更新，在初始化时需要区分条件分支 (branch) 和无条件分支 (jump) 指令。

2.7.2 调用返回预测器

ZenCove 的返回地址栈 (RAS) 大小可配置，默认为 8 项。

对于 jr/jalr/jal/bal 等指令，若其写回寄存器为 ra 寄存器，则我们认为其为函数调用指令。若其跳转目标为 ra 寄存器的内容，则我们认为其为调用返回指令。

在前端进行分支预测时，若其遇到对应的函数调用/返回指令，则将预测跳转地址压栈/弹出作为预测跳转地址。为了让后端在提交分支预测错误信息时可以恢复 RAS 的指针信息，我们将该信息同指令的 μop 一起传到 ROB 中，在后端提交时进行恢复。

若函数调用层数超过 RAS 项数，则循环覆盖，该操作只会造成特别底层的函数调用返回时产生分支地址的预测错误。

若函数返回弹出 RAS 项时，弹出了被覆盖的项，也只会导致分支地址错误，不会产生更多的副作用。

2.8 微架构优化

ZenCove 微架构在设计时，为了进一步提高指令执行效率，加入了推测唤醒和流水线前传两个执行方法。

这两个方法可以有效减少指令在各个执行流水线中冲突时暂停的周期，从而有效降低 CPI。

2.8.1 推测唤醒

Load 指令与需要读 load 结果的后续指令的 RAW 依赖是程序执行中常见的依赖，即 load-use 依赖。

由于 load 指令执行周期数不确定，尤其是在 cache 缺失时，因此只能在访存流水线的执行 2 (MEM2) 阶段进行唤醒。这导致 load-use 要空 3 个气泡。即使是乱序执行也通常找不到这么多不相关指令，导致 load 经常导致气泡出现。

而 cache 命中时 load 执行周期时确定的，因此在 MEM1 阶段即可唤醒，那么 cache 命中时 load-use 只需要空 2 个气泡，通常就可以找到其它不相关指令进行填充。

但在 cache 缺失时这一唤醒就是错误的，后续例如整数指令就不能及时读到正确的寄存器结果。

因此我们在 cache 缺失时简单地将所有流水线暂停。考虑到 cache 缺失应当是小概率事件，并且 cache 缺失本就不可避免地导致处理器执行暂停，因此总体而言还是能够做到性能提升。

2.8.2 流水线前传

由于发射宽度较宽，本处理器没有实现所有流水线之间的前传，但对最常用的整数流水线之间实现了完全的前传。在没有前传的情况下，整数指令与其它指令之间的 RAW 依赖需要空一个气泡，整数流水线的前传使得整数指令之间的 RAW 依赖可以背靠背执行，达到了最高的执行效率。

2.9 外部接口

CPU 采用 AXI4 总线与外部相连，流水线直接访问 Instruction bus，Cached data bus，Uncached data bus 三条总线，对于需要暴露一条总线的，采用 AXI crossbar IP 做 3*1 crossbar。

按照 MIPS 手册，CPU 支持 6 个外部中断。