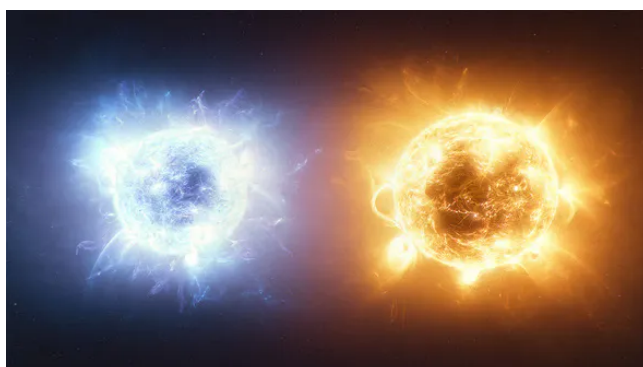


“龙芯杯”第六届全国大学生计算机系统能力培养大赛

重庆大学“所以延迟槽会消失对不队”队

CDIM 项目 决赛设计报告



陈泱宇

cyy@cyysf.name

李燕琴

maxpicca@qq.com

王梓宇

925136384@qq.com

张翀

20194159@cqu.edu.cn

2022 年 8 月

目录

第 1 章 概述	1
1.1 项目背景	1
1.2 名词解释	1
1.3 项目概述	1
第 2 章 CPU 设计方案	2
2.1 指令集支持	2
2.2 双发策略	2
2.3 数据通路设计	3
2.4 冲突处理	5
2.5 CP0 寄存器	5
2.6 中断和异常	6
2.7 缓存设计	6
第 3 章 仿真与差分测试框架	10
3.1 框架介绍	10
3.2 SoC-Simualtor	10
3.3 CEMU	10
第 4 章 操作系统支持	15
4.1 虚拟内存支持	15
4.2 Cache 指令支持	15
4.3 U-Boot	15
4.4 uCore	16
4.5 Linux	18
第 5 章 附录	20
5.1 系统软件移植仓库	20
5.2 差分测试工具链	20

第 1 章 概述

1.1 项目背景

本项目依托于龙芯杯提供的 FPGA 实验平台、Soc 工程环境以及基准测试程序，设计并实现了一个部分兼容 MIPS32 体系结构的小端序 CPU，名为 CDIM（CQU Dual Issue Machine），其能成功通过龙芯杯提供的功能测试、性能测试、系统测试，具有较完善的运算处理、AXI 访问、异常处理、中断响应等功能，并能够运行 PMON、U-Boot 引导程序、uCore 操作系统和 Linux 操作系统等。

1.2 名词解释

本项目中可能用到的一些名词缩写及其解释如表 1.1 所示。

表 1.1: 名词缩写和解释

名词缩写	全称	解释
MIPS	Microprocessor without Interlocked Pipeline Stages	无内部互锁流水级的微处理器
SOC	System On a Chip	片上系统
CPU	Central Processing Unit	中央处理器
ALU	Arithmetic Logic Unit	算术逻辑单元
GPR	General Purpose Register	通用寄存器
CP0	Co-Processor 0	协处理器 0
BRAM	Block Random Access Memory	块随机访问存储器
FIFO	First In First Out	先进先出
RAW	Read After Write	写后读
WAW	Write After Write	写后写
WAR	Write After Read	读后写
ASID	Address Space Identification	地址空间标识

1.3 项目概述

本项目，首先设计了一款双发射五级顺序流水线 CPU，CDIM（CQU Dual Issue Machine），其支持指令 FIFO、分支预测、指令缓存和数据缓存、Store Buffer 等特殊单元，以提升系统性能，最高 CPU 主频可达 100Mhz，IPC 比值达 36.227。其中，五级顺序流水线由取指（Instruction Fetch）、译码（Instruction Decode）、执行（Excute）、访存（Memory access）、写回（Write Back）五个阶段组成；双发射采用对称双发逻辑以充分保证双发率；指令 FIFO 可以隔离取指阶段和后续阶段，以实现高效取指的作用；分支预测采用静态分支预测单元以减少跳转带来流水线刷新数，利用 PC 低位记录跳转数据，利用传统 2bit 策略更新跳转数据的记录；指令缓存和数据缓存均采用二路组相联和突发传输的设计，单路均为 4KB 以满足伪 VIPT 对虚拟内存最小页面的要求，其中指令缓存的数据行为 64bit 以适应双发取指要求，数据缓存的数据行为 32bit；Store Buffer 可缓存 Uncached 的写 AXI 请求，以降低写 AXI 带来的流水线阻塞数。CDIM 还支持 U-Boot 引导程序，并基于该引导程序，成功运行 uCore 和 Linux 操作系统。

此外，为了方便仿真调试，本项目还基于 Verilator 和 GTKWave，开发了支持功能测试、性能测试、系统测试、运行操作系统等系列测试的差分测试工具链，在逐步提交对比时生成对应的波形图。该工具链同时支持指定记录 Trace 的开始时间，以避免生成的 Trace 过大；亦支持双发率、Cache 命中率、分支预测成功率的统计，以进行针对性的优化。相较于之前 Vivado 仿真调试或 FPGA 上板调试，我们开发的工具链可极大缩短调试时间。

第 2 章 CPU 设计方案

CDIM 采用对称双发射顺序执行, 共有 5 级流水。CDIM 的数据通路示意图如 2.1 所示, 其中红线部分为 Master Path, 蓝线部分为 Slave Path。在对称双发射中, Master 和 Slave 的主要区别在于前者先于后者执行, 在处理异常、提交等事宜时会被优先处理。当然, 上述的“对称”双发射, 只是相对于只支持 ALU 指令双发的非对称逻辑而言, CDIM 的设计中 Slave Path 支持的功能和 Master Path 旗鼓相当。但严格来讲, 并不是绝对对称, 在处理跳转指令、例外指令、写 TLB 指令等这类可能刷新流水线的指令时, 需要优先在 Master 处理, 这在后续的双发策略中会详细说明。

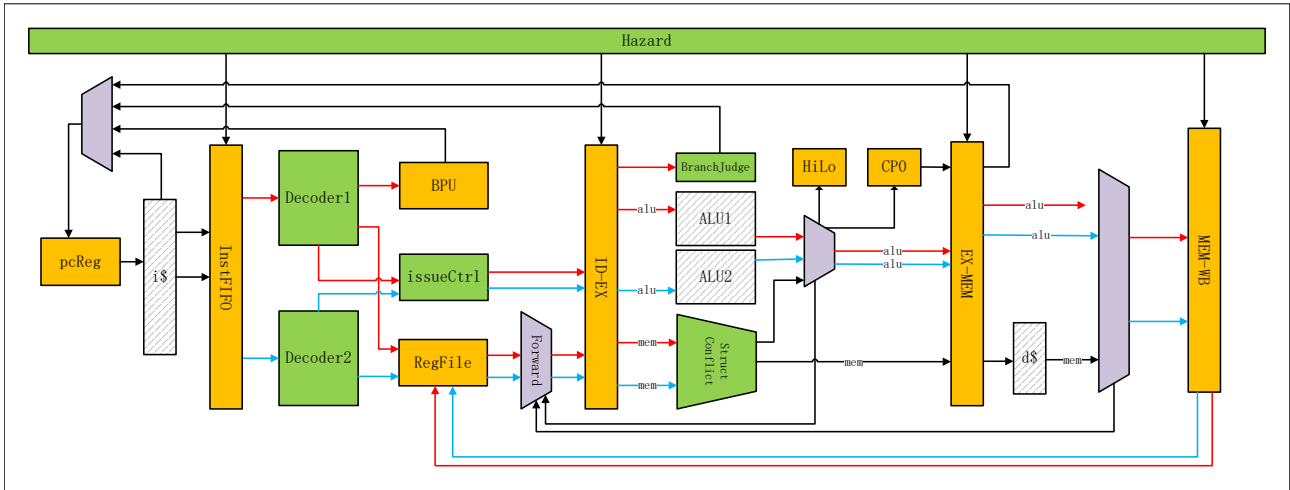


图 2.1: 数据通路设计图

2.1 指令集支持

为了满足大赛提供的功能测试、性能测试、系统测试的要求以及运行操作系统需要的基本指令要求, 我们实现了除 Branch likely 指令和浮点指令的所有 MIPS Release 1 指令集, 按照功能分类如下。

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

2.2 双发策略

在双发射的 CPU 设计中, 双发策略会影响整个数据通路的设计, 故我们先来介绍一下 CDIM 的双发策略, 即分情况讨论 Slave 是否发射。在提供的代码中, issueCtrl 模块解释了 CDIM 的整个双发策略。

1. **only_Master 类指令**: 即只能放在 Master 发射, 且 Slave 不发射指令。对于可能刷新整个流水线的指令 (如 MTC0、SYSCALL、ERET、BREAK、TLBWI、TLBWR 等指令), 如果在 Slave 中检测到这类指令, 则暂停 Slave 发射, 将其延迟到下一周期的 Master 发射; 如果在 Master 中检测到这类指令, 也会暂停 Slave 发射, 避免不必要的流水线刷新。
2. **only_in_Master 类指令**: 即只能放在 Master 发射, 且 Slave 可发射指令。对于跳转指令, 在一定条件下会跳转到其他地址, 并刷新整个流水线。和 only_Master 不同的是, 跳转指令会涉及到延迟槽的处理。故如果在 Slave 中检测到这类指令, 则暂停 Slave 发射, 将其延迟到下一周期的 Master 发射; 如果在 Master 中检测到这类指令, 若满足 Slave 发射条件, 则发射该条指令 (延迟槽), 若不满足, 则在下一周期的 Master 中发射延迟槽数据。
3. **数据冲突**
 - (a). **RAW 冲突**: 检查 Master 和 Slave 之间是否存在 HiLo、CP0、GPR 等寄存器的 RAW 情况, 若存在, 则 Slave 不发射。
 - (b). **Load To Use 冲突**: 检查 Slave 读到的寄存器是否是当前在 E 阶段执行的读存指令的结果, 若是, 则 Slave 不发射。
4. **结构冲突**: 由于设计的乘除单元、访存单元只有一个, 故二者会产生结构冲突。若 Master 和 Slave 同时需要占用这类单元, 则 Slave 不发射。
5. **数据有效性**: Slave 发射的前提是 Master 发射或当前第二条指令有效。若遇到 Master 暂停发射或指令 FIFO 为空的情况下, Slave 不发射。

2.3 数据通路设计

2.3.1 取指阶段

为了保证双发射能够正常执行, 第一要素便是“一次取指能够取回多条指令”, 才能保证传到 D 阶段时, 至少有两条指令可以准备用于双发射。指令 Cache 数据行为 64bit 的设计, 也正适应了双发逻辑中取指的特点, 使得 CDIM 传入指令 Cache 一个 PC 地址, 可获得该 PC 及 PC+4 对应的两条指令。值得注意的是, 为了简化 Cache 命中逻辑 (即不存在跨 Cache Line 访问 Cache 的情况), 传回数据中, 并不是所有 PC 都会传回 2 条指令, 故我们添加了 inst_data_ok 信号表示取回的指令是否有效。CDIM 的设计中, 数据行共 2 字, 如果 PC 索引到该行的最后一个字, Cache 只传回 1 条指令, inst_data_ok2 置为 0; 除此之外, 传回两条数据, inst_data_ok1 和 inst_data_ok2 均置为 1。

出于双发策略中, Slave 并不是每次都会发射, 故取回的数据需要缓存下来, 以用于下次使用, 节省 Cache 访问时间。CDIM 中, 使用指令 FIFO, 将传回的有效指令缓存下来。使用指令 FIFO 还有一个好处是可以分割取指和后续阶段, 即取指不受后续阶段产生的阻塞影响 (但为了保证结果的正确执行并简化控制逻辑, 后续阶段仍然会受到指令 Cache 产生的 stall 影响)。其中, 如果指令 FIFO 放满, 则暂停前段取指; 如果指令 FIFO 为空, 则暂停后续发射。

2.3.2 译码阶段

Instruction Decode 阶段中, 主要完成以下操作:

- **译码**: 从 FIFO 中读出指令, 立即进入译码模块。译码模块中, 会分析该指令的功能, 并分配控制信号。
- **分支预测阶段**: 为了减少分支跳转带来的流水线刷新数量, 我们添加了静态分支预测单元, 利用 PC 低位记录跳转数据, 利用传统 2bit 策略更新跳转数据的记录。在 D 阶段获取到指令后, 立即根据该指令所在 PC 进行预测, 其中为了减少译码带来的延迟, 分支预测单元内部单独进行分支译码, 若是分支指令, 即可取出预测结果并更新取指阶段的 PC。

- **发射判断：**译码结束后，需要进入 issueCtrl 模块进行发射判断，发射判断结果会返回到指令 FIFO 中，控制 FIFO 的读指针的增量。
- **数据准备：**读取 RegFile 中对应的 GPR 数据，准备好数据后进入 Excute 阶段执行（在 CDIM 中，RegFile 具有 4 个读端口和 2 个写端口）。其中为了在执行阶段的开头即可直接使用准确的数据，我们将数据前推单元放在了译码阶段的末尾。值得注意的是，为了减少前推带来的延迟，译码阶段中需要 GPR 数据的模块，均使用 GPR 直接读出的数据，而不是前推数据（如 JR 类指令目标地址的计算），若此时存在 RAW 冲突，则需要推迟到执行阶段进行计算。

2.3.3 执行阶段

Excute 阶段中，主要完成以下操作：

- **跳转处理：**在译码阶段中，Branch 指令需要在执行阶段中判断跳转是否成功，并将判断结果送回分支预测单元进行数据更新，更新此时取值阶段的 PC 并刷新流水线。同理，如果是译码阶段遇到数据冲突的 JR 类指令，其在本阶段获取到准确的目标地址后，更新此时取值阶段的 PC 并刷新流水线。
- **ALU 计算：**可以处理单周期运算指令和多周期运算指令。其中，多周期运算指令包括乘除法指令。CDIM 中乘法指令需要 2 周期、除法指令需要 35 周期，且在运算完成前会产生 stall 信号（流水线阻塞信号），阻塞 D、E、M、W 四个阶段。其中，为减少资源占用，流水线中只提供一个乘法器和除法器，故 Master 和 Slave 在使用时需要进行仲裁。同时获取到乘除结果后，即在执行阶段写 HiLo 寄存器，以期在访存阶段即可获取新的 HiLo 数据值，避免了 HiLo 数据的前推。
- **访存仲裁及其地址计算：**由于 Data Cache 是 BRAM 结构，需要一个周期取出数据，我们需要在 E 阶段将访存信号传递给 Data Cache。因为 Master 和 Slave 都支持访存（但不会同时访存），故在传递访存前，需要用 StructConflict 模块进行仲裁；并保证访存阶段的刷新信号置 0 且使能信号置为 1，以使访存信号能由 E 阶段正常传到 M 阶段，保证访存阶段数据的正常传回。同时，为了减少访存地址计算经过的路径，我们分离了执行阶段中的 ALU 路径和 MEM 路径（如图 2.1 所示），单独利用 $\text{base(rs value)} + \text{offset(immediate value)}$ 计算访存地址。同时根据访存地址及其比特要求对传给 Data Cache 的地址进行地址错例外判断（AdEl 和 AdEs 判断）。
- **TLB 地址转换：**在上一个小点中，能够获取到访存虚拟地址，此时将访问 L1 TLB（TLB 访问逻辑见章节 4.1）进行初步的映射地址翻译，以便于提前访存。
- **异常汇总：**由上述可以看出，在执行阶段，可以获取从任何指令涉及到的所有异常信号，故我们在执行阶段汇总异常信号，并更新 CP0 寄存器，以期在访存阶段即可获取新的 CP0 数据值，避免了 CP0 数据的前推。

2.3.4 访存阶段

Memory Access 阶段，主要完成以下操作：

- **访存控制：**虽然在 E 阶段传递给了 Data Cache 基本的访存信号，告知 Data Cache 需要准备对应地址的数据；但是并没有告知需要取出的数据的比特数。E 阶段汇总时若发现异常，则传到 M 阶段后，会将 Data Cache 的握手信号 data_sram_enM（M 阶段的访存使能信号）将会置为 0，阻止 Data Cache 将错误的地址传到 AXI 总线上。反之，则正常发送 data_sram_enM 信号，并得到传回的数据，再通过 StructConflict 模块，将数据交付到对应 Path（Master 或 Slave）上。
- **写回数据选择：**Master 和 Slave 两条路径上的计算结果（alu）和访存结果（mem）的选择。

2.3.5 写回阶段

Write Back 阶段，主要执行写回 GPR 请求。若此时读取 GPR 的地址，恰好是写 GPR 的地址，需要进行写前递操作。

2.4 冲突处理

2.4.1 数据冲突

- **RAW 冲突:** 当 Master 或 Slave 需要读取的数据是已经发射且进入后续流水阶段的指令, 会产生 RAW 冲突。本设计将需要的数据均前推到 D 阶段, 前推后立即进入触发器后再进入 E 阶段使用, 这样可以避免执行数据的路径过长。其中, 若需要读取的数据是当前 E 阶段的访存数据, 则需要阻塞以等待访存数据在 M 阶段返回。若需要读取的数据是当前 E 阶段的计算结果、或 M 阶段的计算结果、或 M 阶段的访存结果, 则通过数据前推模块前推到 D 阶段。
- **WAW 冲突:** 当同周期的 Master 和 Slave 写入寄存器的地址一致时, 会产生 WAW 冲突。因为 Slave 是 Master 后一条指令, 故 RegFile 更新时, 会优先写入 Slave 的结果。
- **WAR 冲突:** 由于本设计为顺序流水线, 故不存在 WAR 冲突。

2.4.2 结构冲突

当 Master 和 Slave 同时需要访存或占用乘法器或占用除法器时, 会产生结构冲突。本设计中, 结构冲突的指令只发射 Master 一条, 保证 CPU 正确执行。

2.4.3 控制冲突

当遇到跳转指令时, 会产生控制冲突。MIPS 指令系统中, 跳转指令包含延迟槽。本设计中, 若产生跳转 (译码阶段或执行阶段), 则根据发射情况判断当前 Path 中的第一条读数据是否是延迟槽数据。如果是, 则需要判断当时是否有阻塞信号, 若没有则直接执行该第一条读数据, 若有则需要保留该数据到指定 `delayslot_data` 寄存器, 以备阻塞取消后使用; 此外跳转会引起流水线的刷新, 但需要保证延迟槽数据能够不被刷新并正常流入下一阶段执行。如果不是, 则正常刷新流水线即可。

2.5 CP0 寄存器

在自定义 CPU 中扮演着小型操作系统角色的 CP0, 具有异常处理、内存管理、系统控制等功能。为了支持 MIPS 指令系统的正常功能运行, 我们设计的 CP0 处理器包含的基本寄存器如表 2.1 所示。

表 2.1: 基本的 CP0 寄存器

编号	选择	名称	功能定义
8	0	BadVAddr	记录最新地址相关例外的出错地址
9	0	Count	处理器内部计数器
11	0	Compare	计时中断控制器
12	0	Status	处理器状态与控制寄存器
13	0	Cause	存放上一次例外原因
14	0	EPC	存放上一次发生例外指令的 PC
15	0	PRId	处理器版本和标识符
15	1	EBase	中断向量基地址寄存器 (MIPS Release2 必需)
16	0	Config0	处理器配置
30	0	ErrorEPC	上一次发生例外的计数器数值

在添加 TLB 时, 需要涉及内存管理相关的 CP0 寄存器, 如表 2.2 所示。

表 2.2: MMU 相关的 CP0 寄存器

编号	选择	名称	功能定义
0	0	Index	TLB 数组的索引
1	0	Random	随机数
2	0	EntryLo0	TLB 项的低位
3	0	EntryLo1	TLB 项的低位
4	0	Context	指向内存中页表入口的指针
5	0	PageMask	控制 TLB 的虚拟页大小
6	0	Wired	控制 TLB 中固定的页数
10	0	EntryHi	TLB 项的高位
28	even	TagLo	缓存标签 Tag 的低位
29	even	TagHi	缓存标签 Tag 的高位

2.6 中断和异常

为使处理器的功能完整，CDIM 支持异常处理。根据 MIPS 规范要求，CDIM 支持精确异常，即出现异常后，准确记录发生异常的指令地址，并存放在 EPC 寄存器中以待异常返回时使用；且保证发生异常前的所有指令正常提交；发生异常的指令及其之后的指令不提交，并跳转至异常处理程序入口进行异常处理。CDIM 中支持的指令优先级顺序如下：

1. **中断例外**：包括硬件中断、软件中断和计时器中断。
2. **地址转换异常**：访问 TLB 时，TLB 表中没有有效的转换对应项可用时，会发生地址转换异常，包括 TLB 重填异常、TLB 无效异常（有 load 页无效、store 页无效、modify 页无效等）等。
3. **地址错例外（取指）**：PC 地址未对齐四字节。
4. **保留指令例外**：当执行一条未实现的指令时，触发保留指令例外。
5. **自陷例外、系统调用例外**：执行到 Syscall、Break 等自陷指令时。
6. **整型溢出例外**：执行 ADD，SUB，ADDI，SUBI 等指令发生溢出时。
7. **地址错例外（数据访问）**：访问数据的地址未对齐，包括 AdEl、AdEs 两类错误。

其中中断例外为异步例外，其他异常例外为同步例外。CDIM 的设计中，某时刻发生的中断，将被绑定在 D 阶段执行的指令及其 PC 上。

2.7 缓存设计

CDIM 中，设计实现了指令 Cache 和数据 Cache，且均为虚拟索引物理标签（VIPT）记录，二者的突发传输数量、大小、组相联数量均为可配置。其中数据 Cache 支持写缓冲，以减少写回的阻塞数量。

2.7.1 内存管理

MIPS 指令系统对操作模式和存储访问类型进行了严格的限定，将整个 4 GB 大小的虚地址空间划分为 5 段，如图 2.2 所示。

虚拟地址到物理地址的翻译上，共有两种方式：一是直接地址翻译，对应图中 Unmapped 段，翻译较快，延迟较少；二是映射地址翻译，对应图中 Mapped 段，需要访问 TLB 表项，延迟较大。在 CDIM 的缓存设计中，通过多级 TLB 的方式，减少映射地址翻译模式的时延和缺失率，以在实际地址翻译中，可以综合使用直接地址翻译和映射地址翻译。

缓存使用上，指令 Cache 和数据 Cache 均支持 Uncached 段的单独处理，这样发送 AXI 请求时，只需要做指令和数据请求的两路仲裁即可，极大简化仲裁逻辑。

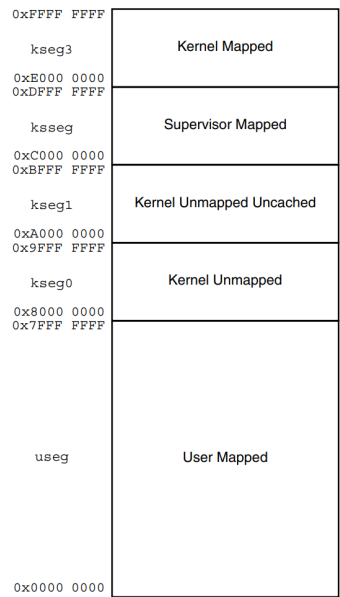


图 2.2: 虚实地址映射关系图

2.7.2 缓存结构

CDIM 的缓存结构和资源使用如图2.3所示，指令 Cache 默认为二路组相联且一路 4KB，一行 data 共 2 字 64bit，一行 tag 对应 8 行 data；数据 Cache 默认为二路组相联且一路 4KB、一行 data 共 1 字 32bit，一行 tag 对应 16 行 data。因为 Bram 读取数据时需要一周后返回数据，故我们提前发送了下一个访存地址（指令 Cache 对应 pc_next，数据 Cache 对应 E 阶段的访存地址）进行提前查询。当到达对应的访存阶段时（指令 Cache 对应 F 阶段，数据 Cache 对应 M 阶段），在根据上一周期的命中情况判断能否同周期获取对应数据，以及是否发送 AXI 请求。在具体访问 AXI 时，Cached 请求支持突发传输，默认为 16 字突发（即一个 tag 对应的总字数）；Uncached 请求默认不支持突发传输。

2.7.3 CACHE 指令

在 MIPS 指令系统中，CACHE 指令要求清除特定地址或索引位置的数据缓存。由于清除“索引位置”已经包含了清除“特定地址”，故 CDIM 的设计中，Data Cache 操作全部实现为 Index Writeback Invalid，Instruction Cache 操作全部实现为 Index Invalid，这使我们能够在满足运行操作系统所必备执行修改后的指令以及刷新 DMA 缓冲区的情况下，简化 Cache 的逻辑，使 CPU 能够实现更高的频率。

2.7.4 状态机

Cache 的控制实现上，我们采用状态机控制的方式，状态转换的情况如图2.4所示，其中，Data Cache 包含写回操作，故比 Inst Cache 多一个“CACHE_WRITEBACK”状态，即图中绿色部分。

各个状态的含义如下。

- **IDLE**: Cache 空闲状态，进行数据提前读取、地址 L1TLB 查询或直接翻译。
- **TLB REFILL**: L1TLB 缺失时，查找 L2TLB、处理可能遇到的 TLB 重填异常或 TLB 无效异常、L2TLB 命中后更新 L1TLB。
- **UNCACHE**: 向 AXI 发送读请求或写请求（无突发传输），并等待 AXI 访存结束。
- **CACHE REPLACE**: cache 缺失后，若没有脏位，需要重新向 AXI 发送读请求（突发传输），访存并替换对应路的缓存行。其中替换原则采用 1bit 替换方式。

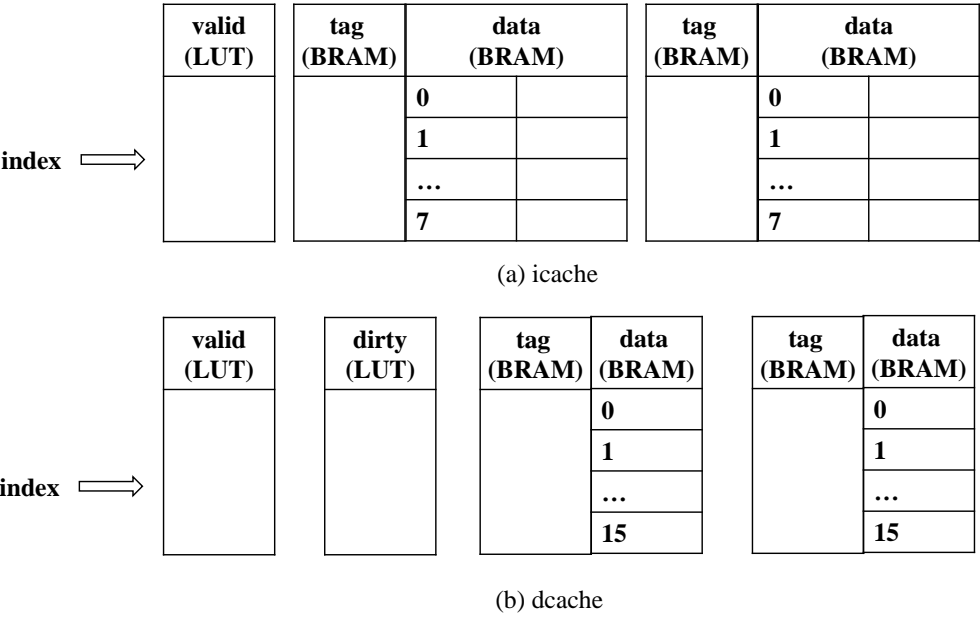


图 2.3: CDIM 缓存结构示意图

- **CACHE WRITEBACK:** 当需要清除或替换缓存行时，若存在脏位，则需要将脏数据写回，即向 AXI 发送写请求（突发传输）。
- **SAVE RESULT:** 当 Cache 因为 TLB 缺失或 Uncached 导致 Cache 暂时无法正常访问数据时的保留状态。

2.7.5 写缓存

在 CDIM 的设计中，对于 Uncached 的 AXI 写请求，我们设置了 Store Buffer，即一个写请求队列，将 AXI 写请求数据缓存下来，再逐个写回，该期间如果没有 Cached 的访问 AXI 请求，则流水线不会因为写回 Store Buffer 数据而阻塞，提高了流水线的执行效率。

值得注意的是，为了保证读写数据的准确，当 Cached 数据需要访问 AXI 时，若这个时候 Store Buffer 仍然在占用 AXI 通道，则需要等待 Store Buffer 写完后，高速缓存才能去处理这个 Cached 数据请求，这个阶段流水线会因为该 Cached 数据请求得不到响应而阻塞。

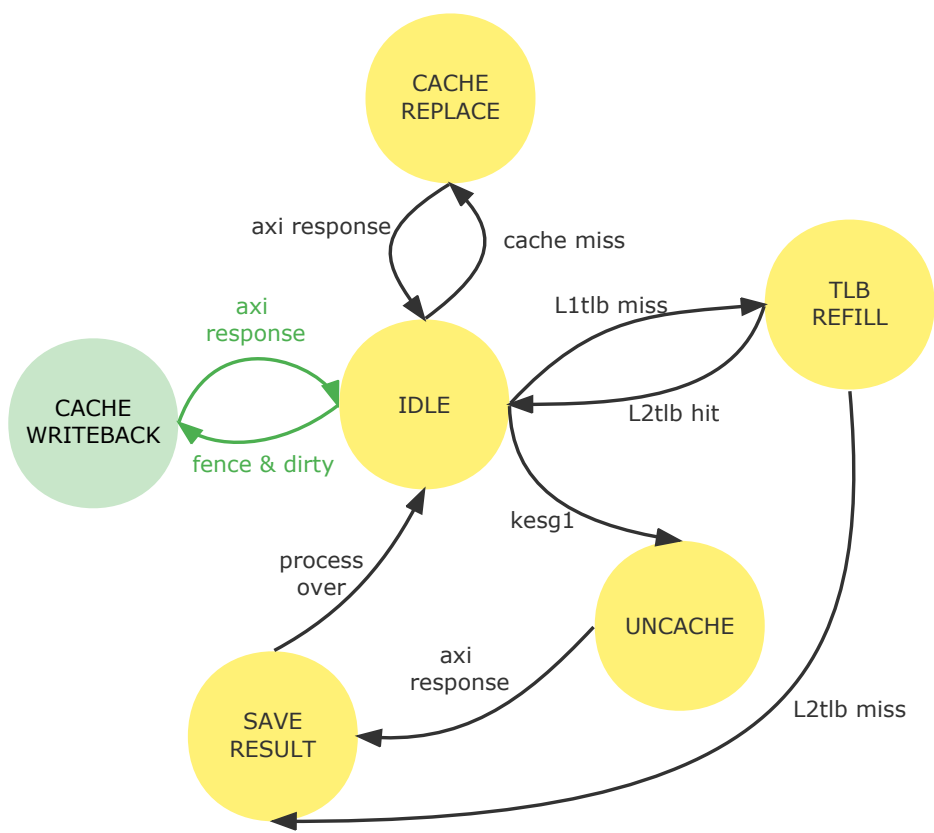


图 2.4: CDIM 缓存状态机示意图

第 3 章 仿真与差分测试框架

3.1 框架介绍

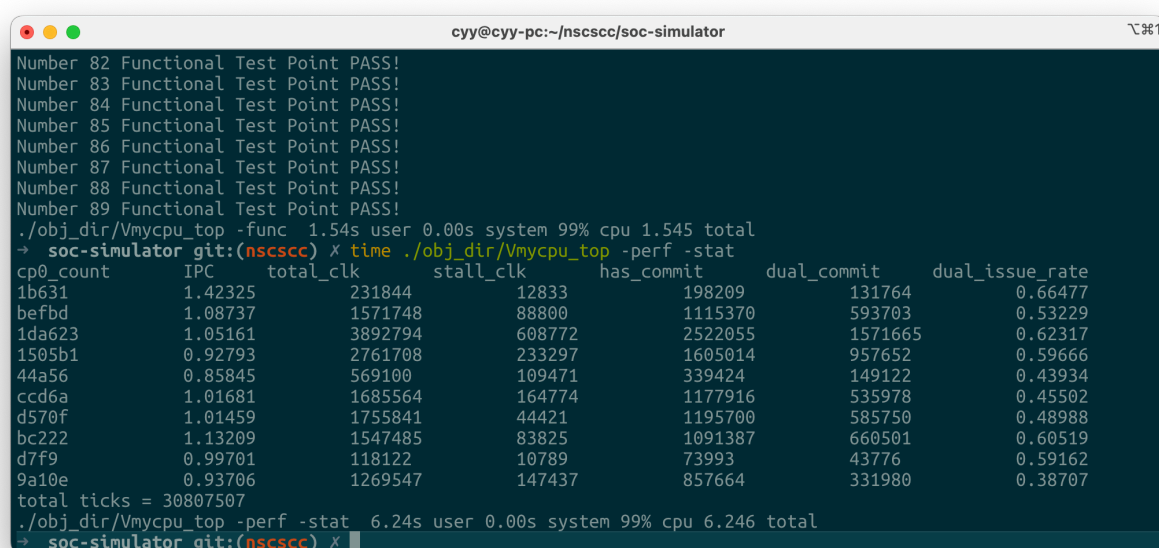
为了提高处理器开发效率，我们自己开发了仿真与差分测试框架，分为 SoC-Simulator 与 CEMU（CQU Emulator）两个部分。

3.2 SoC-Simualtor

SoC-Simulator 是我们开发的一个基于 Verilator 的 SoC 仿真框架。它使用 C++ 语言编写，通过软件实现了 AXI Slave 设备功能。其内置了 MMIO Crossbar、UARTLite、UART8250、NSCSCC confreg 等设备，能够满足运行龙芯杯功能测试、性能测试、系统测试，启动 u-boot、uCore 和 Linux 的需求。

此外，SoC-Simulator 还能够支持 AXI 延迟的自定义，在我们开发处理器期间经过了多次测试，在 AXI 延迟设置为 23 周期的情况下，SoC-Simulator 仿真的性能得分与上板真实成绩误差在 1 分以内。

同时得益于 Verilator 的仿真效率，以我们的 CPU 核 CDIM 为例，在当前主流的台式机（AMD Ryzen 7 5800X）上实现了高速测试，2 秒完成功能测试（含 Trace 比对），7 秒钟完成性能测试并输出 CP0 count 寄存器的结果。同时，在 SoC-Simulator 上运行 uCore 可以 10 秒内进入 shell，运行 Linux v5.19 配合我们基于 defconfig 修改的配置大约 4 分钟开始执行 init 进程，相比于改一行 RTL 综合上板 10 分钟以上的时间，这大大方便了我们修改 RTL 后及时得到 IPC 反馈与问题定位与调试工作。



```
cyy@cyy-pc:~/nscsc/soc-simulator
Number 82 Functional Test Point PASS!
Number 83 Functional Test Point PASS!
Number 84 Functional Test Point PASS!
Number 85 Functional Test Point PASS!
Number 86 Functional Test Point PASS!
Number 87 Functional Test Point PASS!
Number 88 Functional Test Point PASS!
Number 89 Functional Test Point PASS!
./obj_dir/Vmycpu_top -func 1.54s user 0.00s system 99% cpu 1.545 total
→ soc-simulator git:(nscsc) x time ./obj_dir/Vmycpu_top -perf -stat
cp0_count      IPC      total_clk      stall_clk      has_commit      dual_commit      dual_issue_rate
1b631          1.42325      231844          12833          198209          131764          0.66477
befbd          1.08737      1571748         88800          1115370         593703          0.53229
1da623         1.05161      3892794         608772         2522055         1571665         0.62317
1505b1         0.92793      2761708         233297         1605014         957652          0.59666
44a56          0.85845      569100          109471         339424          149122          0.43934
ccd6a          1.01681      1685564         164774         1177916         535978          0.45502
d570f          1.01459      1755841         44421          1195700         585750          0.48988
bc222          1.13209      1547485         83825          1091387         660501          0.60519
d7f9           0.99701      118122          10789          73993           43776           0.59162
9a10e          0.93706      1269547         147437         857664          331980          0.38707
total ticks = 30807507
./obj_dir/Vmycpu_top -perf -stat 6.24s user 0.00s system 99% cpu 6.246 total
→ soc-simulator git:(nscsc) x
```

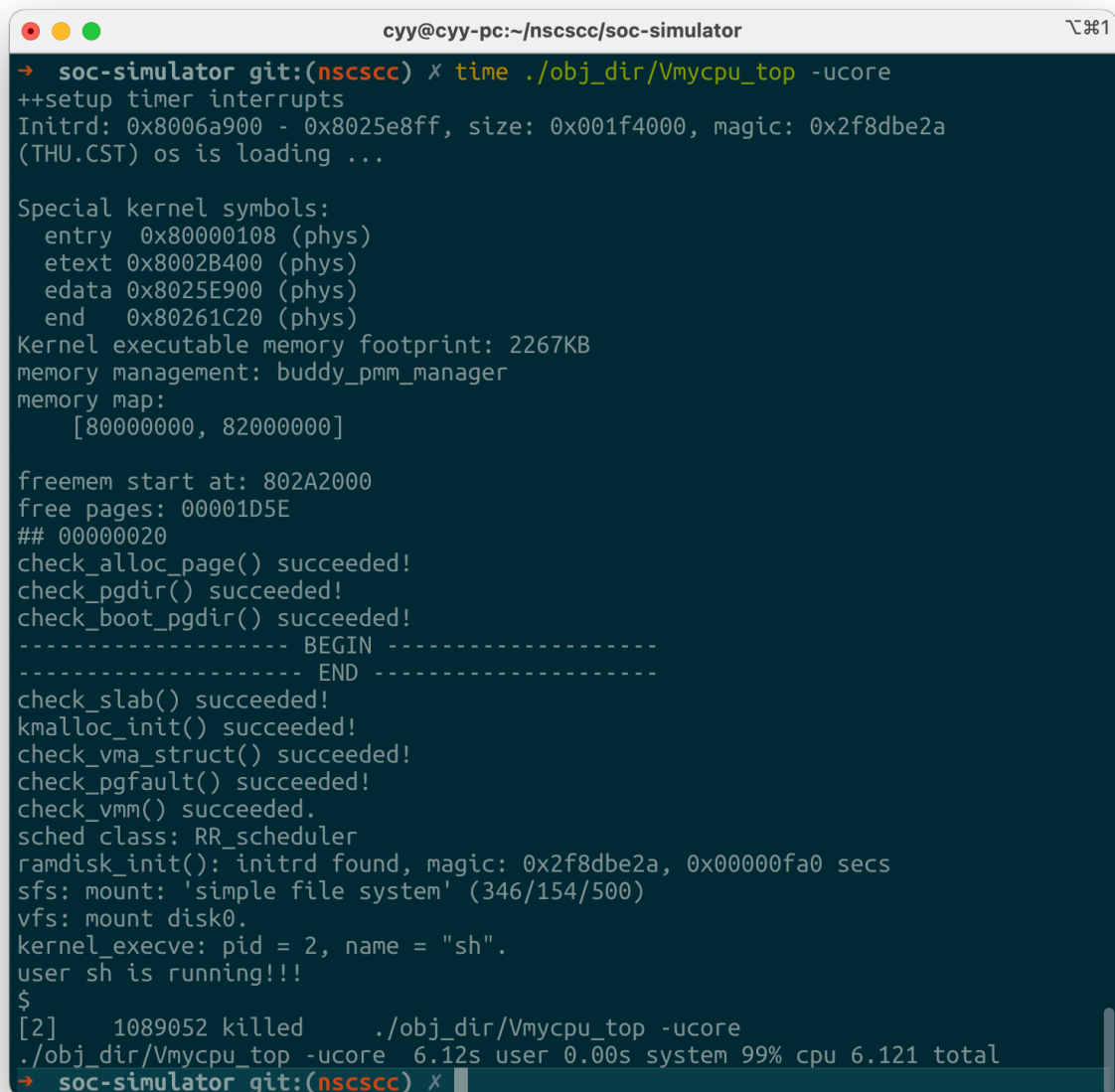
图 3.1: SoC-Simulator 运行功能测试与性能测试

3.3 CEMU

CEMU(CQU Emulator) 是我们开发的一个 CPU 模拟器，目前支持了 RV64IMA 与 MIPS Release 1 指令集，LoongArch32 的支持正在实现中，我们使用它与 CDIM 进行差分测试，

由于 CEMU 与 SoC-Simulator 都是我们自行开发的，因而采用了完全相同的外设接口（C++ 的抽象类），从而 CEMU 与 SoC-Simulator 能够运行在完全相同的 SoC 环境中，并通过处理器 RTL 输出的 CP0 Count、CP0 Cause、CP0 Random、是否中断等信号对 CEMU 的各寄存器状态进行同步，这帮助我们在 CDIM 调通 uCore 后仅 2 天时间就完成了 Linux 的调试。

在 CDIM 调试过程中，我们也不断完善了 CEMU 与 SoC-Simulator 的 difftest 功能。例如添加了内存写回时 AXI 传入数据与 CEMU 内存的对比功能，以及条件输出波形图等功能（避免调试 Linux 时波形图输出消耗大量存储资源）。并在调试 Linux 期间为了调试 Linux Kernel 自身写入的指令（如 EBASE 段的异常处理，这部分不在 vmlinux 的反汇编中）的执行过程，在差分测试错误时能够将 CEMU 的整个内存转储出来，再通过 objdump 反汇编即可知道动态写入的指令，便于通过指令序列观察处理器执行可能存在的问题（例如常见的延迟槽出现先前测试未覆盖的指令）。



```
cyy@cyy-pc:~/nscsc/soc-simulator
→ soc-simulator git:(nscsc) x time ./obj_dir/Vmycpu_top -ucore
++setup timer interrupts
Initrd: 0x8006a900 - 0x8025e8ff, size: 0x001f4000, magic: 0x2f8dbe2a
(THU.CST) os is loading ...

Special kernel symbols:
  entry 0x80000108 (phys)
  etext 0x8002B400 (phys)
  edata 0x8025E900 (phys)
  end    0x80261C20 (phys)
Kernel executable memory footprint: 2267KB
memory management: buddy_pmm_manager
memory map:
  [80000000, 82000000]

freemem start at: 802A2000
free pages: 00001D5E
## 00000020
check_alloc_page() succeeded!
check_pgdir() succeeded!
check_boot_pgdir() succeeded!
----- BEGIN -----
----- END -----
check_slab() succeeded!
kmalloc_init() succeeded!
check_vma_struct() succeeded!
check_pgfault() succeeded!
check_vmm() succeeded.
sched class: RR_scheduler
ramdisk_init(): initrd found, magic: 0x2f8dbe2a, 0x00000fa0 secs
sfs: mount: 'simple file system' (346/154/500)
vfs: mount disk0.
kernel_execve: pid = 2, name = "sh".
user sh is running!!!
$
[2] 1089052 killed ./obj_dir/Vmycpu_top -ucore
./obj_dir/Vmycpu_top -ucore 6.12s user 0.00s system 99% cpu 6.121 total
→ soc-simulator git:(nscsc) x
```

图 3.2: SoC-Simulator 运行 uCore

```

cyy@cyy-pc:~/nscsc/soc-simulator
➔ soc-simulator git:(nscsc) x time ./obj_dir/Vmycpu_top -linux
[ 0.000000] Linux version 5.19.0-00006-gd23d4ca1c01e-dirty (cyy@cyy-pc) (mipsel-linux-gnu-gcc (Debian 11.2.0-18) 11.2
.0, GNU ld (GNU Binutils for Debian) 2.38) #3 Sat Aug 13 20:18:54 CST 2022
[ 0.000000] printk: bootconsole [early0] enabled
[ 0.000000] CPU0 revision is: 00018003 (MIPS 4Kc)
[ 0.000000] MIPS: machine is MegaSoC,Demo
[ 0.000000] Malformed early option 'earlycon'
[ 0.000000] Initrd not found or empty - disabling initrd
[ 0.000000] Primary instruction cache 8kB, VIPT, 2-way, linesize 64 bytes.
[ 0.000000] Primary data cache 8kB, 2-way, VIPT, no aliases, linesize 64 bytes
[ 0.000000] Zone ranges:
[ 0.000000] Normal [mem 0x0000000000000000-0x0000000003ffffff]
[ 0.000000] Movable zone start for each node
[ 0.000000] Early memory node ranges
[ 0.000000] node 0: [mem 0x0000000000000000-0x0000000003ffffff]
[ 0.000000] Initmem setup node 0 [mem 0x0000000000000000-0x0000000003ffffff]
[ 0.000000] Built 1 zonelists, mobility grouping on. Total pages: 16256
[ 0.000000] Kernel command line: console=ttyS0,230400n8 root=/dev/mmcblk0p2 rootfstype=ext4 rw rootwait earlycon rdin
it=/sbin/init
[ 0.000000] Dentry cache hash table entries: 8192 (order: 3, 32768 bytes, linear)
[ 0.000000] Inode-cache hash table entries: 4096 (order: 2, 16384 bytes, linear)
[ 0.000000] mem auto-init: stack:off, heap alloc:off, heap free:off
[ 0.000000] Memory: 54456K/65536K available (6412K kernel code, 605K rdata, 1096K rodata, 2116K init, 220K bss, 1108
0K reserved, 0K cma-reserved)
[ 0.000000] SLUB: HWalign=32, Order=0-3, MinObjects=0, CPUs=1, Nodes=1
[ 0.000000] NR_IRQS: 256
[ 0.000000] arch_init_irq with ebase: 0x80000000
[ 0.000000] irq-xilinx: /soc/interrupt-controller@1fb00000: num_irq=8, edge=0x1
[ 0.000000] clocksource: MIPS: mask: 0xffffffff max_cycles: 0xffffffff, max_idle_ns: 57337813970 ns
[ 0.000012] sched_clock: 32 bits at 33MHz, resolution 30ns, wraps every 64424509936ns
[ 0.001587] Console: colour dummy device 80x25
[ 0.002001] Calibrating delay loop... 66.17 BogoMIPS (lpj=132352)
[ 0.040724] pid_max: default: 32768 minimum: 301
[ 0.042783] Mount-cache hash table entries: 1024 (order: 0, 4096 bytes, linear)
[ 0.043119] Mountpoint-cache hash table entries: 1024 (order: 0, 4096 bytes, linear)
[ 0.061191] devtmpfs: initialized
[ 0.085152] clocksource: jiffies: mask: 0xffffffff max_cycles: 0xffffffff, max_idle_ns: 7645041785100000 ns
[ 0.085489] futex hash table entries: 256 (order: -1, 3072 bytes, linear)
[ 0.099150] NET: Registered PF_NETLINK/PF_ROUTE protocol family
[2] 1089592 killed ./obj_dir/Vmycpu_top -linux
./obj_dir/Vmycpu_top -linux 5.31s user 0.01s system 99% cpu 5.326 total
➔ soc-simulator git:(nscsc) x

```

图 3.3: SoC-Simulator 运行 Linux


```

cyy@cyy-pc:~/nscscc/soc-simulator
→ soc-simulator git:(nscscc) x time ./obj_dir/Vmycpu_top -linuxdiff
[ 0.000000] Linux version 5.19.0-00007-g6f6e5ede4bed-dirty (cyy@cyy-pc) (mipsel-linux-gnu-gcc (Debian 11.2.0-18) 11.2
.0, GNU ld (GNU Binutils for Debian) 2.38) #9 Tue Aug 16 02:04:49 CST 2022
[ 0.000000] printk: bootconsole [early0] enabled
[ 0.000000] CPU0 revision is: 00018003 (MIPS 4Kc)
[ 0.000000] MIPS: machine is MegaSoC,CDIM
[ 0.000000] Malformed early option 'earlycon'
[ 0.000000] Initrd not found or empty - disabling initrd
[ 0.000000] Primary instruction cache 8kB, VIPT, 2-way, linesize 64 bytes.
[ 0.000000] Primary data cache 8kB, 2-way, VIPT, no aliases, linesize 64 bytes
[ 0.000000] Zone ranges:
[ 0.000000]   Normal [mem 0x0000000000000000-0x0000000007fffffff]
[ 0.000000] Movable zone start for each node
[ 0.000000] Early memory node ranges
[ 0.000000]   node 0: [mem 0x0000000000000000-0x0000000007fffffff]
[ 0.000000] Initmem setup node 0 [mem 0x0000000000000000-0x0000000007fffffff]
[ 0.000000] Built 1 zonelists, mobility grouping on. Total pages: 32512
[ 0.000000] Kernel command line: console=ttyS0,230400n8 rdinit=/sbin/init earlycon
[ 0.000000] Dentry cache hash table entries: 16384 (order: 4, 65536 bytes, linear)
[ 0.000000] Inode-cache hash table entries: 8192 (order: 3, 32768 bytes, linear)
[ 0.000000] mem auto-init: stack:off, heap alloc:off, heap free:off
[ 0.000000] Memory: 118984K/131072K available (6774K kernel code, 609K rwdata, 1128K rodata, 2168K init, 224K bss, 12
088K reserved, 0K cma-reserved)
[ 0.000000] SLUB: HWalign=32, Order=0-3, MinObjects=0, CPUs=1, Nodes=1
[ 0.000000] NR_IRQS: 256
[ 0.000000] arch_init_irq with ebase: 0x80000000
[ 0.000000] irq-xilinx: /soc/interrupt-controller@1fb00000: num_irq=8, edge=0x1
Error at 12809808 ps!
last      : PC = 0x803dcb7c
reference: PC = 0x80000180, wb_rf_wnum = 0x1b, wb_rf_wdata = 0x8000001c
mycpu     : PC = 0x80966be8, wb_rf_wnum = 0x04, wb_rf_wdata = 0xbfb00000
cemu memory dumped!
cemu pc history:
803dcb74
803dcb78
803dcb7c
80000180
stop at 12809809 ns.
./obj_dir/Vmycpu_top -linuxdiff 2.93s user 0.08s system 99% cpu 3.027 total
→ soc-simulator git:(nscscc) x

```

图 3.4: SoC-Simulator 与 CEMU 差分测试错误提示

第 4 章 操作系统支持

4.1 虚拟内存支持

CDIM 支持了 MIPS 规范中基于 TLB 的虚拟地址转换功能。由于 MIPS 指令集架构中，TLB 采用全相连的结构，匹配需要大量的电路延迟，为了保证运行操作系统时的频率，我们将 TLB 拆为两级。

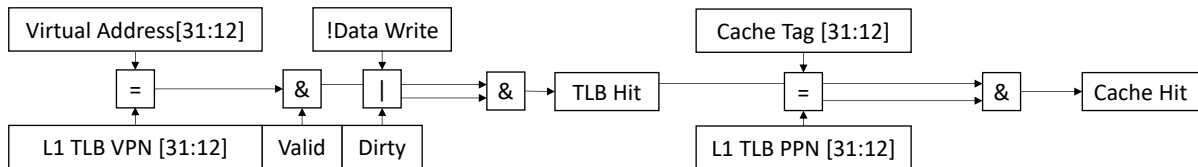


图 4.1: 一级 TLB 的设计

其中，一级 TLB 分为一级指令 TLB 与一级数据 TLB，各只有一项，位于指令 Cache 与数据 Cache 中。由于我们的 Cache 单路恰好为 4KB，恰好为我们处理器支持的最小页面大小，因此采用伪 VIPT（Virtually Indexed Physically Tagged）方式，比较方式如图 4.1 所示。

在我们的设计中，一级 TLB 与 Cache 状态机深度整合，且在一级 TLB 命中的情况下不会带来额外的流水线停顿，由此兼顾了 IPC 与频率。

二级 TLB 采用全相连结构，有 8 项，位于 CP0 中，提供了 3 个可同时访问的查找接口分别用于 TLBP 指令、一级指令 TLB、一级数据 TLB。当一级 TLB 缺失时，会向 CP0 中的 L2 TLB 发出缺失的 VPN2 地址（虚拟地址的 [31:27] 部分）进行 TLB 查找，并在下一周期得到对应的 TLB 表项以及是否命中成功。当出现 L2 TLB 查找失败、TLB 无效、写只读页等情况时，则根据当前状态发出对应的异常，跳转到异常处理地址交给系统内核完成 TLB 的回填操作。

由于我们 TLB 做了两级，需要解决 L1 TLB 与 L2 TLB 间的一致性问题，这里需要考虑两种情况，一种是 ASID 切换，另一种是 TLB 写入。我们采用的设计是在 L1 TLB 中添加一个清空 TLB 的信号，并在执行 MTC0、TLBWI、TLBWR 三条指令时都拉高清空信号，以解决虚拟地址重名问题。

4.2 Cache 指令支持

由于操作系统中存在写入指令后执行，与 DMA 前后写回或失效 Cache 的操作，这需要 Cache 指令的支持。这一部分已在 Cache 章节有所介绍。且经过我们测试发现 Linux 不会使用 Index Store Tag 这样特殊的 Cache 指令，因此这样简单的实现就满足了运行 Linux 的要求。

4.3 U-Boot

我们继承了前辈 NSCSCC 2019 清华大学编程是一件很危险的事情与 NSCSCC 2021 北京航空航天大学一队的工作，为我们的 CPU 移植的 U-Boot。主要工作包括编写设备树、修改配置文件、修改串口时钟频率。最终我们成功运行起了 U-Boot 并能够通过 tftp 载入 uCore 与 Linux 操作系统。

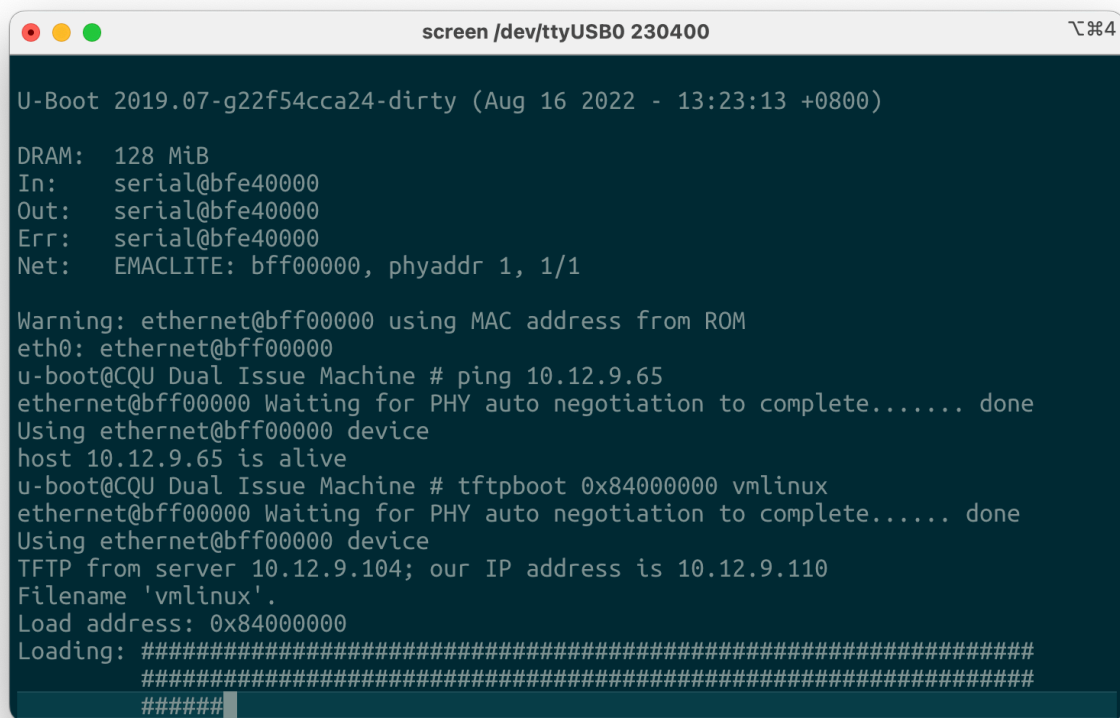
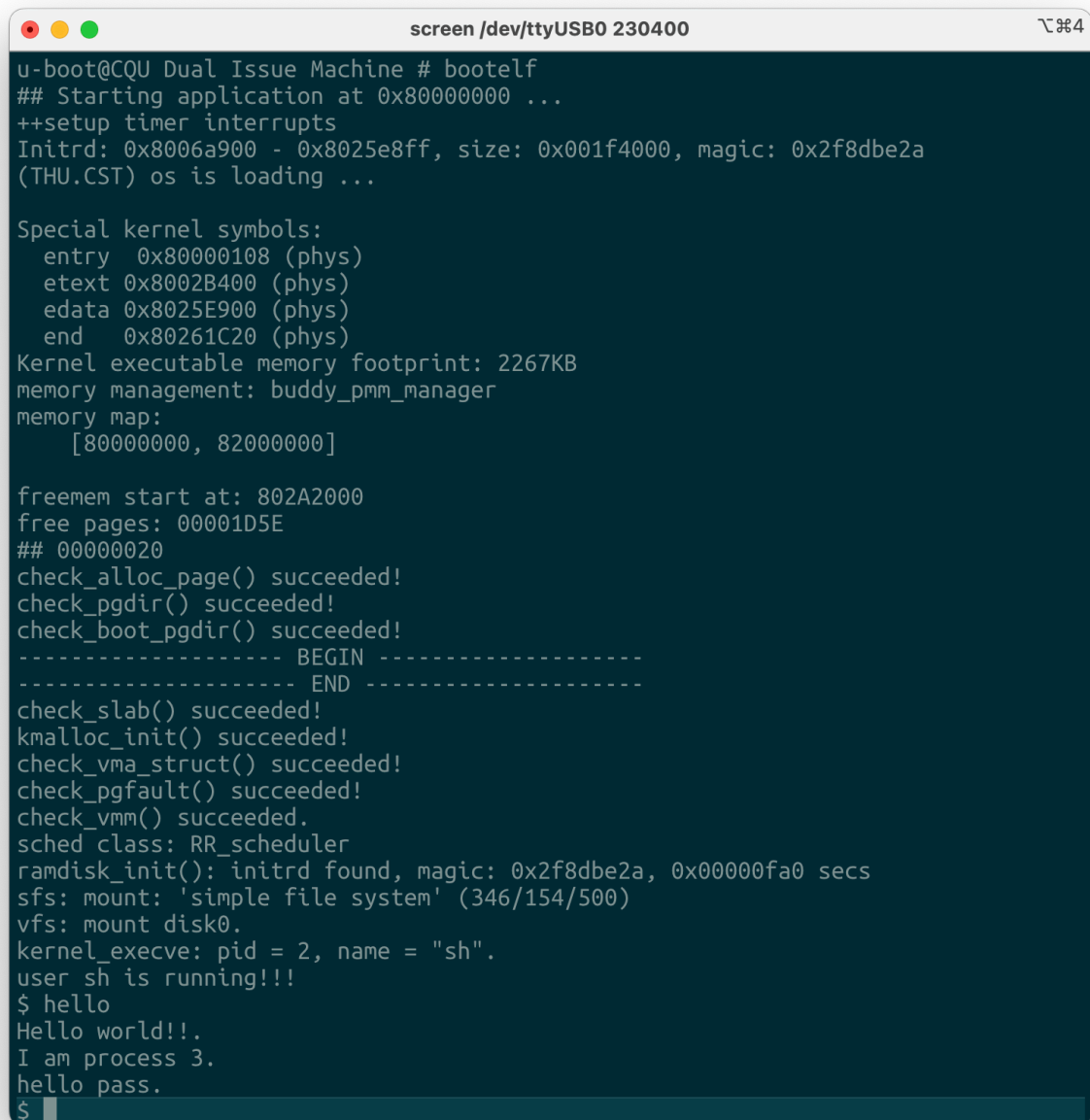
A screenshot of a terminal window titled 'screen /dev/ttyUSB0 230400'. The terminal shows the U-Boot boot process. It starts with a timestamp 'U-Boot 2019.07-g22f54cca24-dirty (Aug 16 2022 - 13:23:13 +0800)'. It then displays hardware information: 'DRAM: 128 MiB', 'In: serial@bfe40000', 'Out: serial@bfe40000', 'Err: serial@bfe40000', and 'Net: EMACLite: bff00000, phyaddr 1, 1/1'. A warning message follows: 'Warning: ethernet@bff00000 using MAC address from ROM'. The terminal then shows the initialization of 'eth0: ethernet@bff00000'. A user prompt 'u-boot@CQU Dual Issue Machine # ping 10.12.9.65' is shown, followed by the response 'ethernet@bff00000 Waiting for PHY auto negotiation to complete..... done' and 'Using ethernet@bff00000 device'. The user then enters 'tftpboot 0x84000000 vmlinux', and the terminal shows 'ethernet@bff00000 Waiting for PHY auto negotiation to complete..... done' and 'Using ethernet@bff00000 device'. The next line is 'TFTP from server 10.12.9.104; our IP address is 10.12.9.110'. The filename 'vmlinux' is confirmed. The load address is set to '0x84000000'. The loading process is indicated by a series of hash characters '#####' and a progress bar.

图 4.2: U-Boot 启动截图

4.4 uCore

我们的处理器能够运行清华大学的教学操作系统 uCore，并进入用户 Shell 执行程序。



```
screen /dev/ttyUSB0 230400
u-boot@CQU Dual Issue Machine # bootelf
## Starting application at 0x80000000 ...
++setup timer interrupts
Initrd: 0x8006a900 - 0x8025e8ff, size: 0x001f4000, magic: 0x2f8dbe2a
(THU.CST) os is loading ...

Special kernel symbols:
  entry 0x80000108 (phys)
  etext 0x8002B400 (phys)
  edata 0x8025E900 (phys)
  end    0x80261C20 (phys)
Kernel executable memory footprint: 2267KB
memory management: buddy_pmm_manager
memory map:
  [80000000, 82000000]


freemem start at: 802A2000
free pages: 00001D5E
## 00000020
check_alloc_page() succeeded!
check_pgdir() succeeded!
check_boot_pgdir() succeeded!
----- BEGIN -----
----- END -----
check_slab() succeeded!
kmalloc_init() succeeded!
check_vma_struct() succeeded!
check_pgfault() succeeded!
check_vmm() succeeded.
sched class: RR_scheduler
ramdisk_init(): initrd found, magic: 0x2f8dbe2a, 0x00000fa0 secs
sfs: mount: 'simple file system' (346/154/500)
vfs: mount disk0.
kernel_execve: pid = 2, name = "sh".
user sh is running!!!
$ hello
Hello world!!.
I am process 3.
hello pass.
$
```

图 4.3: uCore 运行截图

4.5 Linux

我们的处理器能够在编译器关闭 Branch-Likely 指令的情况下运行 Linux v5.19，并自己编译了 MIPS Release 1 关闭浮点支持的 Busybox，在 Busybox 中能够正常使用 Shell。

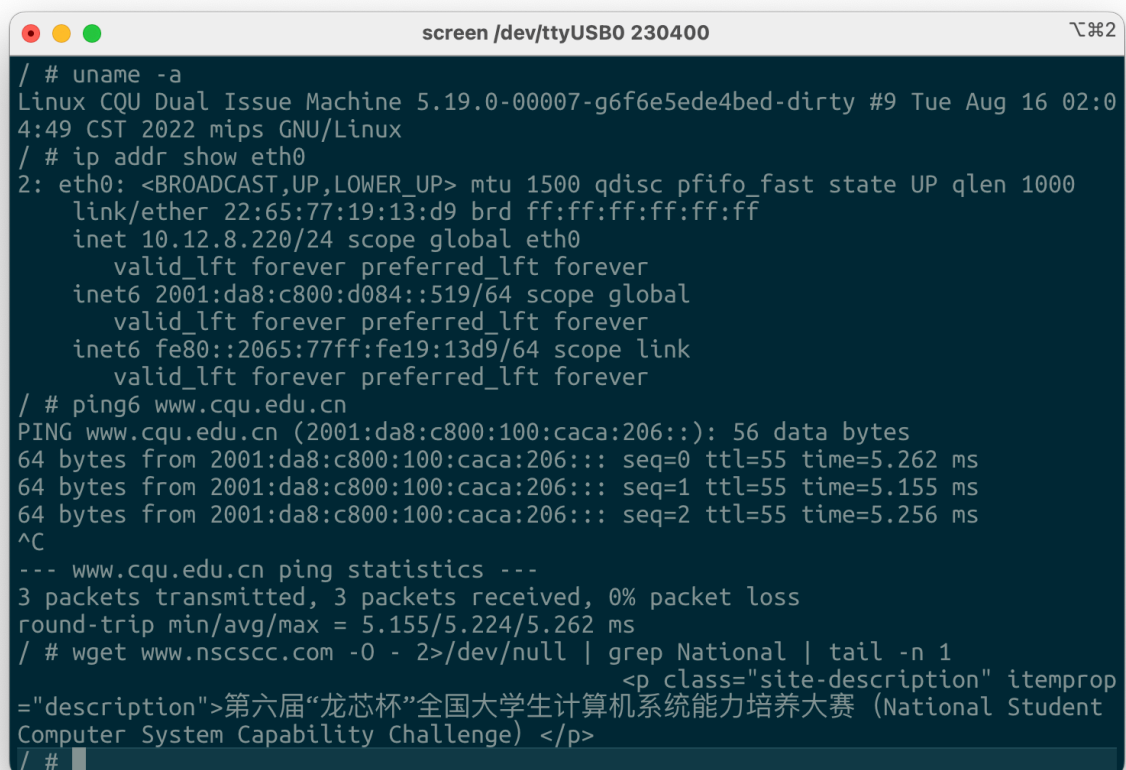
在仿真 SoC-Simulator 与上板 MegaSoC SoC 上均测试成功，且能够使用 SoC 上的 AXI Ethernet Lite IP 核连接网络，能够使用 wget 访问 www.nscscc.com 网站并提取部分文字。



```
screen /dev/ttyUSB0 230400
:13:d9
[ 4.678850] xilinx_emaclite 1ff00000.ethernet: Xilinx EmacLite at 0x1FF00000
mapped to 0x(ptrval), irq=9
[ 4.689550] usbcore: registered new interface driver usb-storage
[ 4.696512] i2c_dev: i2c /dev entries driver
[ 4.708933] usbcore: registered new interface driver usbhid
[ 4.712288] usbhid: USB HID core driver
[ 4.720704] usbcore: registered new interface driver snd-usb-audio
[ 4.738691] NET: Registered PF_INET6 protocol family
[ 4.760748] Segment Routing with IPv6
[ 4.764265] In-situ OAM (IOAM) with IPv6
[ 4.767742] sit: IPv6, IPv4 and MPLS over IPv4 tunneling driver
[ 4.783392] NET: Registered PF_PACKET protocol family
[ 4.787108] Key type dns_resolver registered
[ 4.824104] ALSA device list:
[ 4.826110]   No soundcards found.
[ 4.874985] Freeing unused kernel image (initmem) memory: 2168K
[ 4.878524] This architecture does not have kernel memory protection.
[ 4.882306] Run /sbin/init as init process

Please press Enter to activate this console.
/ # uname -a
Linux CQU Dual Issue Machine 5.19.0-00007-g6f6e5ede4bed-dirty #9 Tue Aug 16 02:04:49 CST 2022 mips GNU/Linux
/ #
```

图 4.4: Linux 启动截图



```
screen /dev/ttyUSB0 230400
/ # uname -a
Linux CQU Dual Issue Machine 5.19.0-00007-g6f6e5ede4bed-dirty #9 Tue Aug 16 02:0
4:49 CST 2022 mips GNU/Linux
/ # ip addr show eth0
2: eth0: <BROADCAST,UP,LOWER_UP> mtu 1500 qdisc pfifo_fast state UP qlen 1000
    link/ether 22:65:77:19:13:d9 brd ff:ff:ff:ff:ff:ff
    inet 10.12.8.220/24 scope global eth0
        valid_lft forever preferred_lft forever
    inet6 2001:da8:c800:d084::519/64 scope global
        valid_lft forever preferred_lft forever
    inet6 fe80::2065:77ff:fe19:13d9/64 scope link
        valid_lft forever preferred_lft forever
/ # ping6 www.cqu.edu.cn
PING www.cqu.edu.cn (2001:da8:c800:100:caca:206::): 56 data bytes
64 bytes from 2001:da8:c800:100:caca:206::: seq=0 ttl=55 time=5.262 ms
64 bytes from 2001:da8:c800:100:caca:206::: seq=1 ttl=55 time=5.155 ms
64 bytes from 2001:da8:c800:100:caca:206::: seq=2 ttl=55 time=5.256 ms
^C
--- www.cqu.edu.cn ping statistics ---
3 packets transmitted, 3 packets received, 0% packet loss
round-trip min/avg/max = 5.155/5.224/5.262 ms
/ # wget www.nscscc.com -O - 2>/dev/null | grep National | tail -n 1
        <p class="site-description" itemprop
="description">第六届“龙芯杯”全国大学生计算机系统能力培养大赛 (National Student
Computer System Capability Challenge) </p>
/ #
```

图 4.5: Linux 上网截图

第 5 章 附录

5.1 系统软件移植仓库

表 5.1: 系统软件移植仓库

软件名称	代码仓库
U-Boot	https://github.com/cyysself/u-boot/commits/cdim_soc
uCore	https://github.com/cyysself/ucore-thumips/tree/cdim_soc
Linux	https://github.com/cyysself/linux/tree/cdim_soc

5.2 差分测试工具链

表 5.2: 差分测试工具链

工具名称	代码仓库
SoC-Simulator	https://github.com/cyysself/soc-simulator/tree/nscsc
CEMU	https://github.com/cyysself/cemu/tree/mips32-nscsc

参考资料

- [1] MIPS® Architecture For Programmers I, II, III. Imagination Technologies LTD.
- [2] 计算机组成与设计: 硬件/软件接口. David A.Patterson
- [3] Sirius 设计文档. 于海鑫, 尹思维
- [4] 自己动手写 CPU. 雷思磊
- [5] MegaSoC <https://github.com/MegaSoC>