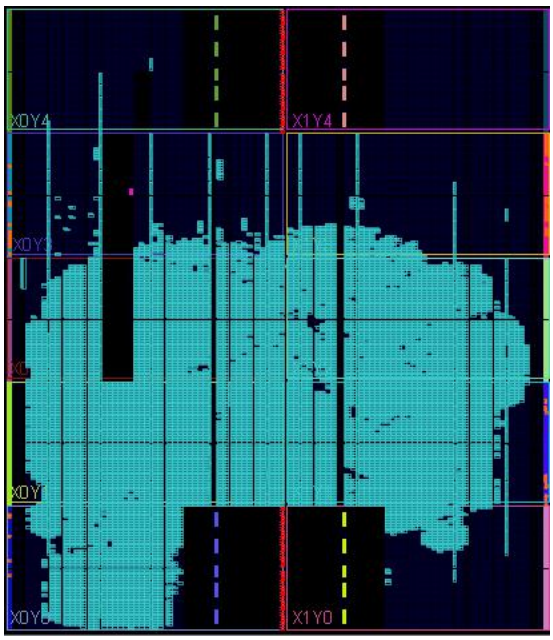
“龙芯杯”第七届全国大学生计算机系统能力培养大赛

哈尔滨工业大学二队

HelloMIPS 项目
决赛设计报告



胡光辉

罗　腾

史子琦

徐一夫

2023 年 8 月

目 录

第 1 章　项目概述

1.1　项目背景

本项目是第七届“龙芯杯”全国大学生计算机系统能力培养大赛（NSCSCC 2023）的参赛作品。项目实现了基于 MIPS32 指令集的乱序多发射 CPU——HelloMIPS，并以比赛提供的 FPGA 实验平台为基础，利用周边硬件，实现了完整的 CPU 微架构。经测试，可以通过全部的功能测试、性能测试，并运行监控程序、U-Boot 引导程序和 Linux 操作系统等。

1.2　项目概述

1.2.1　硬件语言

本项目使用**基于 Scala 构建的硬件描述语言 Chisel3.6.0** 编写。该语言发布于 2023.4.15，集成了许多强大而方便的的特性，比如 decoder、ChiselEnum，自带具有更快更强的编译器 CIRCT。此外 scala 作为一门静态类型的高级语言，能同时支持面向对象编程和函数式编程，这在编写 RTL 中减少了 Verilog 中许多不必要的代码，在可读性、可维护性、可参数化上都十分优秀，节省了许多编码时间。

1.2.2　设计模式

本项目借鉴了开源 RISC-V CPU 项目 NutShell 的设计模式。由统一的接口定义文件和配置文件，将各个分离的模块组合在一起。这既使得 CPU 具有了高度的可配置性，也为协同开发提供了便利。通过编写抽象模块和工具，比如流水线连接、多如多出的队列等，大量减少硬件设计中的重复代码。

1.2.3　CPU 架构

本项目采用了乱序多发射的基本架构，在标准配置下为 3 发射。其中，流水线结构可以大致分为前端取指译码和后端执行提交两部分。具体地，前端部分包括取指令 1、取指令 2、指令缓冲、解码重命名派发 4 个阶段，后端部分包括发射、读寄存器、执行和写回 4 个阶段。本项目支持基于 TLB 的虚拟地址转换，并实现了指令和数据缓存以加快取指和访存。在指令支持方面，基于 MIPS32 Release 1 指令集，我们实现了单核处理器运行 Linux 所需的 79 条指令，支持规范中必须实现的 23 个 CP0 寄存器和 12 种异常类型，实现了精确异常。

1.2.4　SoC 设计

龙芯官方提供的实验箱上有丰富的外设资源，为了更好的进行功能展示，我们以项目 CPU 为核心，搭建了一个 SoC，截止目前我们的片上系统支持：

- CPU：搭载本项目所构建的 CPU，对外暴露一个 AXI 接口
- DRAM：支持板载 128MiB DDR3 颗粒 SDRAM 作为内存
- Flash：支持识别板载可插拔 FLASH 芯片 EN25F80
- 串口：支持识别板载 UART 16550 串口
- 以太网：能够通过网络加载操作系统到指定内存区域、传输演示程序等
- Confreg：复用大赛官方组件，可以控制 LED、数码管、读取开关、按键等
- IIC：控制 TI DAC5571 驱动外部的模拟信号器件，如 LED 灯
- LCD：支持使用实验箱外置 NT35510 LCD 绘制图像
- 触摸屏：支持实验箱外置 NT35510 LCD 触屏反馈
- VGA：支持板载 VGA 接口文本输出
- USB：支持板载 USB 接口识别可移动存储设备以及 HID 设备

1.2.5　系统软件

为了验证 CPU 实现正确性、方便引导操作系统、进行更丰富的功能演示，我们支持以下多个系统软件：

- 清华 MIPS 32 位版本监控程序
- U-Boot 引导程序 v2023.07
- Linux 操作系统 v6.3.0

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部分。第一部分是使用 Chisel3.6.0 构建的 CPU 项目，用于生成可综合的 Verilog 代码。Verilog 代码的调试是使用 Cpp 语言搭建的差分测试系统，可仿真操作系统，大大加快 debug 速度。第二部分使用 Xilinx Vivado 2019.2 Web HL Edition 作为开发 IDE 平台。

自编写的hitd是基于南京大学的 NEMU、重庆大学的 CEMU 和 soc-simulator 差分测试系统，支持如下功能

- NSCSCC 功能测试、性能测试、系统测试
- uboot、linux 仿真至进入 shell
- 比对通用寄存器、HILO 寄存器、CP0 寄存器
- 比对内存数据
- 生成波形文件，支持 fst 和 vcd 格式
- 生成 log 文件，自动分析 AXI 协议
- （性能导向优化）profile guided optimization
- 死循环检测
- linux 内核源码的 C 调试器

该差分测试是 Linux 系统下的 cpp 工程，依赖 verilator 编译 verilog 生成 cpp 语言的电路仿真代码。仿真速度远大于 Xilinx 的 EDA，大大减少了 debug 的时间。此外，该测试系统具有可高度自定义的数据统计方式，极大便利了性能优化的工作，能够更高效的找出性能瓶颈。

第 2 章　CPU 设计

2.1　基本架构

本设计是基于 MIPS 32 架构的具有指令缓冲和数据缓冲的乱序多发射 CPU。CPU 实现 MIPS 32 Rev 1 指令集的一个较完整子集，覆盖了本次大赛初赛所要求的全部指令，以及运行 Linux 操作系统所需的指令。支持 CP0 协处理器和页表缓冲（TLB）的虚拟内存。整个流水线的控制模式为流水线逐级互锁机制，前端 4 级流水，后端为 5 到 7 级，根据不同功能单元有不同的级数。最终达到了 109MHZ 的时钟频率和 51.165 的 IPC 比值。整体的架构如图 2.1所示。

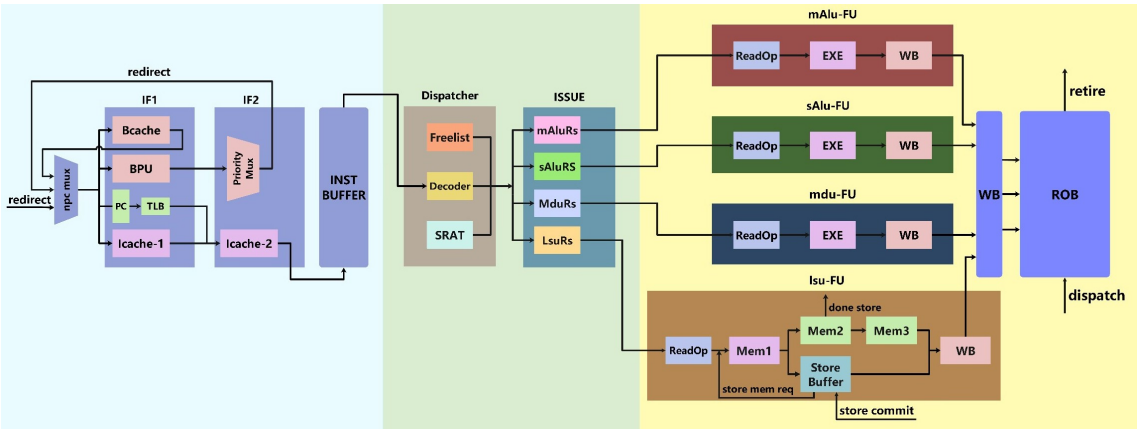


图 2.1　整体架构示意图

2.1.1　前端

前端流水线共 4 级，包括两级取指令、指令缓冲级、寄存器重命名派发

取指 1 段　使用 PC 高位进行 TLB 的查询，低位进行 I-Cache 第一阶段查询。同一周期得到第一级分支预测结果，和后端发来的刷新请求生成 NPC。I-Cache 一次取指 4 条指令，对传入地址不做任何对齐约束，如存在地址边界的情况，需要 CPU 自行丢弃。

取指 2 段　进行 I-Cache 第二阶段，按照命中情况进行阻塞填充。得到第二级分支预测结果，和第一级预测结果如不同则需要对取指 1 段行刷新。最后，本周期拉起 InstrBuffer 的写使能写入指令和 PC。

指令队列　封装个人实现的多入多出队列，大小为 8，如果该周期无法装入 4 条指令则堵塞取指段。出队阶段有微解码，得到指令需要被分配到的功能单元类型、源操作数和目的操作数，用于下一周期重命名和派发。

重命名和派发　从指令缓冲中读取多个指令进入该段进行重命名，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中（Rename Alias Table）。由于精确异常回滚的需要，重命名映射表包括推测性（SRAT）和架构性（ARAT）两份，在需要恢复的时候将 ARAT 复制给 SRAT。采用多入多出队列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。此阶段还需根据指令类型将指令派发到不同的发射队列。若需要分配而空闲寄存器列表空、重定向缓冲已满、产生结构相关，则需要暂停此阶段。此阶段需要并行的进行真正的解码，用于放入 RS 在执行阶段使用。

2.1.2　后端

后端实现了 4 条功能单元流水线：两个非对称的算术逻辑单元（ALU），一个访存单元（LSU）负责访存指令和 Cache 指令，一个乘除单元（MDU）负责乘除指令和特权指令。所有功能单元都包含发射、读操作数、执行写回三个阶段。

发射　保留站采取分布式的设计方式，为四个功能单元分别配置了一个，负责选择合适的就绪指令发射。其中，MDU 和 LSU 两个保留站采取完全顺序的发射方式，两个 ALU 采取的是部分乱序发射的方式，分支指令需要顺序发射。保留站需要监听 SRAT 的写回信号和功能执行单元的唤醒信号，便于将相应指令置为就绪态。

读操作数　由于采用显示重命名，因此需要配合使用非数据捕捉的方式获取操作数。该阶段，需要从物理寄存器堆中读取操作数。对于 ALU 和 LSU，还要在前递数据和寄存器堆数据中进行选择。

执行　执行单元有 4 个，两个一段流水的算术逻辑单元（ALU），一个三段流水的访存单元（LSU），一个多周期的乘除单元（MDU）。

ALU　两条整数流水线分为主从流水线，都包含了基本的算数逻辑单元。主流水线还包含了分支处理的功能，需要得到分支指令正确的跳转方向，结合上一

级带下来的实际跳转地址对分支预测结果进行判断，对 BPU 进行相应更新。若预测错误，需要向派发级发送信号，指示其完成前端重定向。还需将预测错误的标记写回 ROB 中，便于后续刷新流水线后端。

MDU　乘除特权流水线包含乘除指令、clz、HILO 读写指令以及特权指令的处理。写 HI/LO 相关指令需在这一阶段写入 Speculative HILO。此外，对于涉及到写 HI/LO/CP0 的指令，在这一阶段将写数据保存置一个队列，用于提交时改变处理器状态以及恢复 Speculative HILO。

LSU　包含地址计算，TLB 查询，D-Cache 查询和 StoreQeueu 写入/查询。

- 第一段查询 TLB，load 指令需要进行 D-Cache 的第一阶段查询。store 指令在退休后会让 StoreQueue 把存请求发送给第一段进行 D-Cache 访问。
- 第二段 store 指令装入 StoreQueue，并发起写回信号，等待提交后执行。load 指令会进行 D-Cache 的第二阶段查询，以及 StoreQueue 的查询。
- 第三段 load 指令对查询结果进行合并选择，最后写回提交。

写回　此阶段将执行结果分别写入物理寄存器和 ROB，并更改推测重命名表（Speculative RAT）。

指令退休　提交部分的 ROB 是一个多入多出的队列，会读取已写回的指令，并按顺序进行提交。正常情况下，退休指令可能会更改 Architecture RAT 并释放 Freelist 中借用的物理寄存器号；特殊情况包括同周期限定只能提交单条的指令，以及异常和分支预测失误等指令导致的流水线刷新。

2.2　分支预测

我们实现了两级分支预测和竞争分支预测。

两级分支预测　完整分支预测包含如下常见的分支预测器

- 分支预测目的缓冲（Branch Target Buffer），记录 PC 对应的指令类型和目标跳转地址
- 两位饱和计数器（Two-bit Predictors），预测简单的分支指令跳转模式
- 局部历史表（Pattern History Table），使用局部历史预测模式复杂的分支指令
- 返回地址栈（Return Address Stack），预测 jr ra 指令的跳转目的

因为前端一次取四条指令，每个周期需要预测四个 PC，得到四个预测结果。如果想要得到精确的预测结果，需要如下过程。

- 由于四个 PC 不可能相同，所以的 BTB、TBP、PHT 均由四个存储体构成，一次读四个
- 由于对 PC 没有对齐约束，所以在读取存储体之后需要进行四选一使得 PC 和结果匹配
- 由于可能存在地址边界，所以需要找出第一条有效的跳转分支目的作为预测结果。

考虑到该路径长度过大，因此将读存储体和四选一放在取指一段，优先级选择和重定向放在取指二段。2 个周期的完整分支预测开销会导致取指较慢，于是实现了第一级分支预测，也叫分支预测缓冲（B-Cache）。输入是四条取指 PC 的第一条，输出为第二级分支预测的结果。大小为 32 项的分支预测缓冲，预测命中率和优化效果如表 2.1所示

表 2.1　B-Cache 命中率和运行时间

程序名	命中率	使用前运行 ticks	使用后运行 ticks	减少比
bitcount	90.7276%	21178	20545	2.99%
bubble sort	83.2921%	121182	120981	0.17%
coremark	90.1726%	389289	379613	2.49%
crc32	98.1965%	185744	138347	25.54%
dhrystone	89.0955%	58202	37509	35.60%
quick sort	91.2439%	170375	169841	0.31%
select sort	97.0352%	125847	103873	17.44%
sha	99.0750%	124664	111292	10.70%
string copy	96.5502%	14176	13686	03.46%
string search	87.7337%	126079	89649	28.91%

竞争分支预测　分支指令的是否跳转是由局部历史表和两位饱和计数器竞争预测的。局部历史表的表项大小为 32，使用长度为 9 的历史（即 512 个两位饱和计数器），且带有 tag 和 4 位长度的频度计数器。每执行一条分支需要更新历史和对应的两位饱和计数器，此外若表项 tag 命中，需要对频度计数器自增，否则自减。更新时，若频度计数器冲突到 0 时需要更换 PHT 的表项。预测时，频度计数器达到 14 时使用 PHT 而不是 TBP 的预测结果。表 2.2展示了整个分支预测器对于不同类

型指令的命率。

表 2.2　竞争分支预测系统对不同类型分支指令的命中率

程序名	br	jcall	jret	jmp	jr
bitcount	93.08%	89.94%	90.98%	77.27%	33.33%
bubble sort	90.29%	97.56%	98.25%	77.77%	80.00%
coremark	87.54%	97.44%	91.46%	97.77%	40.66%
crc32	97.35%	96.16%	92.03%	99.78%	95.73%
dhrystone	93.94%	99.16%	98.34%	97.08%	69.39%
quick sort	80.72%	96.87%	95.62%	97.45%	60.00%
select sort	95.36%	98.90%	96.51%	99.91%	80.00%
sha	98.31%	96.82%	94.89%	94.18%	56.66%
string copy	97.41%	98.86%	97.89%	98.94%	40.00%
string search	94.97%	91.98%	98.71%	98.19%	77.07%

br 是所有的分支指令，jcall 是所有的 jal 和 jalr，jret 是 jr ra，jmp 是 j 指令，jr 是除了 jr ra 以外的 jr 指令

2.3　指令派发

在重命名派发段一次从指令队列中取出三条指令，分配到对应的功能单元中。MDU 和 LSU 较为简单，只需要派发第一条对应类型的指令。MainAlu 和 SubAlu 因为有指令交叠而需要特殊的派发方案。其中可派发到 SubAlu 的指令一定可以被派发到 MainAlu。

- 方案 1：MainAlu 派发第一条 Main 指令，SubAlu 派发第一条 Sub 指令
- 方案 2：MainAlu 派发第一条 Sub 指令，SubAlu 派发第二条 Sub 指令

初期实现是存在一条 MainAlu 指令的时候采用方案 1，否则采用方案 2，在后期实现中发现两个可优化点。

SUB SUB MAIN　当需派发的指令序列是 Sub、Sub、Main，此时采用方案 1，只会派发第二条指令。需要在派发逻辑中加入特殊条件的判断，如果遇到”SSM” 的情况需要采用方案 2，性能提升如表 2.3所示。

SUB FIRST　由于 MainAlu 较为繁忙，在采用方案二的情况下往往因为 MainAlu 阻塞而等待。因此把方案 2 改为 SubAlu 派发第一条 Sub，MainAlu 派发第二条 Sub

有较好的优化效果，性能提升如表 2.3所示。

表 2.3　不同派发优化方案的运行时间减少比

程序名	SMM	SFirst	叠加作用
bitcount	0.99%	0.13%	0.71%
bubble sort	0.019%	2.56%	2.80%
coremark	-0.32%	2.78%	4.24%
crc32	-0.43%	0.54%	0.42%
dhrystone	-1.26%	2.27%	2.76%
quick sort	2.76%	3.76%	5.10%
select sort	4.14%	3.76%	7.87%
sha	0.0%	1.74%	1.82%
string copy	0.89%	-0.014%	0.87%
string search	-0.95%	-0.68%	0.18%

2.4　旁路设计

设计中的旁路包括：两条主从整数流水线本身及相互之间的旁路，访存流水线到其本身及整数流水线的旁路。整数和访存流水线分别用执行段和访存 3 段产生的结果进行前递，且都在读操作数阶段获取前递结果。

2.5　提前唤醒

为了最大限度降低数据相关性导致的性能损失，我们将某些尚未写回的目的寄存器编号广播至保留站中，相关的指令会被设为就绪状态。唤醒源包括：访存 1 段的 load 类指令，被仲裁电路选中即将发射、以及确保在后一周期能够来到执行级的整数流水线指令。由于 load 类指令存在 cache 不命中等特殊情况，有可能无法及时地给需要的指令前递数据，因此需维护保留站中相应的标记位。被设为就绪态的指令会从保留站中发射，在读操作数阶段，需要前递数据而未得到的指令将会被阻塞。

初期实现时，我们采用的方案是：两个整数保留站在发射时进行组内和组间唤醒，在读操作数阶段唤醒 LSU 和 MDU；后期观察到部分程序中 LOAD 指令往往处于指令数据相关性的顶端，因此我们加上了 LOAD 指令对访存和整数保留站的唤醒，获取了较大的性能提升，不同唤醒方案运行时间减少比如表 2.4所示。

表 2.4　不同唤醒方案运行时间减少比

程序名	整数指令相关唤醒	LOAD 相关唤醒	叠加作用
bitcount	29.11%	1.25%	29.93%
bubble sort	6.13%	16.23%	22.38%
coremark	10.50%	11.38%	22.85%
crc32	38.10%	2.03%	40.16%
dhrystone	9.14%	9.09%	18.20%
quick sort	15.84%	10.39%	27.12%
select sort	14.13%	14.33%	21.62%
sha	24.35%	8.82%	33.31%
string copy	3.77%	13.48%	19.36%
string search	10.69%	9.69%	20.50%

2.6　访存索引竞争预测

我们实现了一个竞争预测模块，预测 D-Cache 访问所使用的 Indedx，该模块是频率和 IPC 的一个折中。

在读取操作数之后进入 LSU 第一段，需要得到访存地址用于访问 D-Cache。由于读取操作数需要进行较长的 64 选 1，计算地址又至少需要 12 位的加法，导致路径较长。如果单独增加一计算地址的流水，会导致 IPC 如附录中表 A.1所示的大幅度下降。考虑到实际线长是来自于访存地址到 D-Cache 的 BRAM，而不是到 Mem1 段的段间寄存器。可以通过使用源操作数 Index 部分来预测地址 Index 部分而不加上符号扩展的立即数。如果在 Mem1 段发现了不同，则需要暂停一拍。

考虑到大部分程序的预测命中率有待提升，尝试实现一个 PC 到 Index 的直接映射缓冲进行预测。单独使用该预测器会对大部分的程序有较大提升，除了 bubble sort 有 30.9% 的大幅度下降。附录统计数据 A.2显示，bubble sort 程序 47.87% 的预测失误来自于直接映射缓冲，于是实现一个使用两位饱和计数器竞争的访存索引预测器（Memory Index Predictor）。这有利于实现全面的性能提升，最后达到的效果如表 2.5所示。

2.7　写缓冲

LSU 中实现了一个 StoreQueue 用于实现 store 指令的回滚。如图 2.2所示，其本质是一个有四个指针的单写入读出的队列。

表 2.5　各种访存索引预测器命中率

程序名	源操作数预测	直接映射预测	竞争预测
bitcount	89.6528%	93.6692%	93.7103%
bubble sort	88.8426%	61.3900%[a]	88.3303%
coremark	84.7390%	96.6992%	93.7282%
crc32	84.7013%	91.4235%	88.1177%
dhrystone	67.7626%	89.2350%	85.7862%
quick sort	64.4898%	96.5355%	91.5337%
select sort	98.5381%	99.2707%	99.1512%
sha	82.3747%	89.6988%	88.3988%
string copy	88.3117%	92.0658%	91.5116%
string search	74.1802%	87.6655%	85.4556%

[a] 直接映射预测对 bubble sort 效果较差，需要竞争预测

- enqPtr：入队，store 指令将 TLB 翻译结果、写数据等信息写入 StoreQueue
- retPtr：指示 store 指令是否退休，实现同一周期退休多条 store 指令
- reqPtr：指示退休的 store 指令是否已经发起 D-Cache 请求，用于流水出队
- deqPtr：指示队尾，在 D-Cache 返回的时候加一

load 指令在进入 LSU 二段的时候需要查询 StoreQueue，使用 StoreQueue “最新” 的数据，合并上 D-Cache 的数据一并写回。查询 StoreQueue 最新的数据需以 Byte 为单位，从 enqPtr 到 deqPtr 查询地址是否匹配，这是一个复杂且长的优先级选择，将其放在和 D-Cache 同一段是不错的选择。

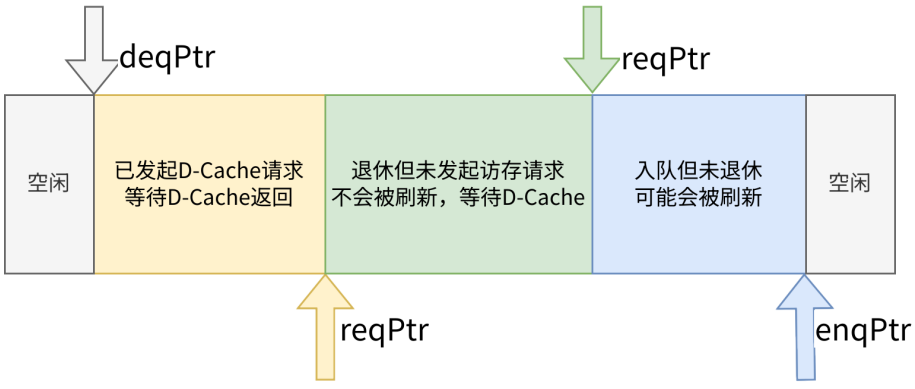


图 2.2　StoreQueue 示意图

2.8　退休逻辑

退休宽度为 3。出于时序考虑，退休阶段拆分为两级流水，且维护了一个自动状态机用于处理特殊情况：在第一级，至多 3 条指令会从 ROB 中弹出，会根据异常、分支预测失误等特殊情况决定下一级能够退休的指令和自动机状态的改变。在第二级，指令真正退休，正常的指令将完成 ARAT 的修改和物理寄存器编号的归还；特殊的指令会在对应的自动机状态下进行处理，需要刷新流水线的指令会在该指令退休的下一周期进行刷新。出于时序考虑和设计需求，有如下限定：

- 异常指令只能在 0 号提交口提交，提交后的下一周期进行重定向
- 在分支预测错误指令及其延迟槽指令退休的下一周期进行后端的刷新
- 特殊指令（写 HI、LO）限定同一周期只能提交一条
- store 指令需要激活 StoreBuffer
- cache、mtc0、TLB 相关指令以及非分支的分支预测错误指令，需要根据具体情况进行重定向

2.9　缓存设计

采用可配置组相联缓存，默认配置下 I-Cache 大小 8KB，2 路，Cache line 为 64B。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。均采用 LRU 作为替换策略，选路部分参考开源项目 rocket-chip[1]。D-Cache 为写回 Cache。

2.10　实现指令集

共实现了 79 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, CLZ, DIV, DIVU, MADD, MADDU, MSUB, 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，JR, JR.HB

自陷指令 TEQ, TNE

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

2.11　特权架构

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，按照地址排序如下表所示

表 2.6　CP0 实现

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

2.12　内存管理

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：注意，kseg0 段与 kseg1 段映射到相同的物理地址

表 2.7　内存地址分配

名称	地址段	User 态可访问	固定映射	可 Cache
kuseg	[0x00000000, 0x7FFFFFFF]	N/A	是	否
useg/kuseg	[0x00000000, 0x7FFFFFFF]	是	否	根据 TLB
kseg0	[0x80000000, 0x9FFFFFFF]	否	是	K0=1 则是
kseg1	[0xA0000000, 0xBFFFFFFF]	否	是	否

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 以 Cached 方式被映射到。

2.13　异常中断

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

表 2.8　异常实现

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

第 3 章　SoC 设计

3.1　总体结构

SoC 设计如下图 3.1所示。

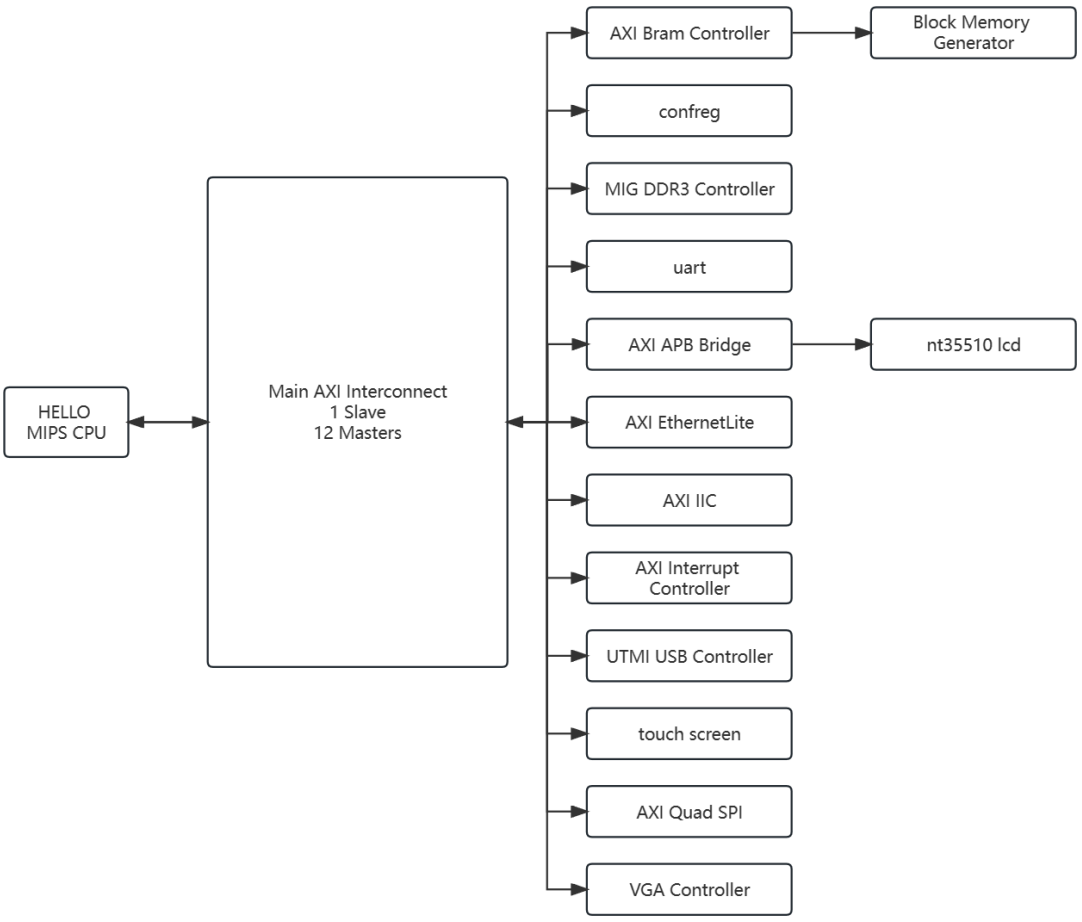


图 3.1　SoC 结构

为了更好的进行功能展示，我们使用 Vivado 的 Block Design 搭建了一个较为简易但是可用的 SoC，并将 Block Design 生成的 wrapper 作为顶级模块。

SoC 所使用的外设模块除 nt35510 lcd 外均提供标准的 AXI 接口，nt35510 lcd 模块提供的 APB 接口可以使用 Xilinx 提供的 AXI APB Bridge IP 核将其转换为标准的 AXI 接口。

为了让 CPU 能访问众多模块，我们使用 Xilinx 提供的 AXI Interconnect IP 核来实现一个总线。

我们使用 Xilinx 提供的 MIG DDR3 Controller IP 核实现了 DDR3 控制器，从

而控制板载 DDR3 内存颗粒作为 SoC 的内存。正确配置后，其上电后会自动进行初始化，为 SoC 提供连续的存储空间。我们使用了 TrivalMIPS 项目团队实现的 reset_synchronizer[2] 对 DDR3 控制器的复位信号进行处理，在此表示感谢。

3.2　地址映射

SoC 中各个设备的物理地址空间分配如下表 3.1所示。

表 3.1　外设物理地址分配

名称	起始地址	结束地址	有效大小	类型
DDR3 控制器	0x00000000	0x07FFFFFF	128M	存储
SPI Flash 控制器	0x1F700000	0x1F700FFF	4K	寄存器
USB 控制器	0x1F800000	0x1F800FFF	4K	寄存器
触摸屏控制器	0x1F900000	0x1F90FFFF	64K	寄存器
VGA 控制器	0x1FA20000	0x1FA2FFFF	64K	寄存器
LCD 控制器	0x1FA30000	0x1FA30FFF	4K	寄存器
GPIO 控制器	0x1FAF0000	0X1FAFFFFF	64K	寄存器
中断控制器	0x1FB00000	0x1FB00FFF	4K	寄存器
BRAM 控制器	0x1FC00000	0x1FCFFFFF	1M	存储
IIC 控制器	0x1FD00000	0x1FD0FFFF	64K	寄存器
串口控制器	0x1FE40000	0x1FE4FFFF	64K	寄存器
以太网控制器	0x1FF00000	0x1FFFFFFF	1M	寄存器

3.3　中断连接

一些外设需要通过中断告知 CPU 其状态。MIPS 规范中 CPU 至多支持 6 个外部中断（编号为 2 到 7，但 7 号通常作为内置定时器中断）。由于 Xilinx EthernetLite 使用了不受 CPU 直接支持的边沿触发的中断，我们引入 Xilinx Interrupt Controller IP 核来管理中断。

SoC 所使用中断如下表 3.2所示。

3.4　硬件支持

FPGA 板载了较多的物理接口和硬件资源，为了充分利用，达到较好的展示效果，我们均对它们进行了适配。

表 3.2　中断连接关系

名称	中断类型	接收方	中断编号
以太网控制器	上升沿	中断控制器	0
IIC 控制器	高电平	中断控制器	1
USB 控制器	高电平	中断控制器	2
SPI FLASH 控制器	上升沿	中断控制器	3
中断控制器	高电平	CPU	2
串口控制器	高电平	CPU	3

3.4.1　串口控制器

龙芯 FPGA 实验箱搭载的串口为 RS-232 接口。我们使用龙芯资料包所提供的串口模块，并学习了 CDIM 项目团队[3] 的移植经验，成功在我们的 SoC 上实现了串口控制器。

3.4.2　以太网控制器

龙芯 FPGA 实验箱上搭载了一个 10/100 Mbps 的以太网接口，并板载 DM 9161AEP 以太网 PHY，引出了以太网协议中规定的标准的 MII 和 MDIO 接口。我们使用 Xilinx 提供的 AXI Ethernet Lite IP 作为数据链路层设备，该控制器在 UBoot 和 Linux 中都有驱动支持。由于该 IP 仅提供上升沿触发的中断，我们使用 AXI 中断控制器将其中断类型改为电平中断。在实际测试中，速度最高可达到 2MB/s 左右。

3.4.3　SPI Flash 控制器

龙芯 FPGA 实验箱上共有两片 SPI NOR Flash 芯片，其中一片为 FPGA 设计固化专用，不可插拔更换 Flash 芯片，另一片是普通 SPI I/O 引脚，可以插拔更换 Flash 芯片。由于不可插拔芯片具有特殊用途，我们的 SoC 实现了对可插拔芯片的控制。我们使用 Xilinx 提供的 AXI Quad SPI IP 核的标准模式来进行控制。在 U-Boot 与 Linux 中都有 SPI 协议和 Flash 读写、MTD 设备支持。然而，由于可插拔芯片上搭载的 EN25F80 Flash 芯片并不在 Linux 内核支持的 Flash 芯片列表中，需要在 Linux 内核增加 EN25F80 Flash 芯片的相关信息，这将在后文驱动移植部分进行较为详细的说明。

3.4.4　GPIO 控制器

龙芯 FPGA 实验箱上有较多直连 FPGA 的数字信号驱动的器件，包括拨码开关、按键、数码管、单双色 LED 等。我们使用龙芯资料包所提供 Confreg 模块进行管理。

3.4.5　LCD 控制器

龙芯 FPGA 实验箱搭载了一块 NT35510 LCD 屏幕，分辨率为 800 ×480。感谢 TrivalMIPS 项目团队[4] 先前的移植适配工作，我们从 TrivalMIPS 项目中移植了一个 NT35510 控制器模块，该模块通过状态机，实现了 APB 总线到 LCD 屏幕的通信。通过该模块以及配套驱动，我们实现了对 LCD 屏幕的控制。驱动部分将在后文驱动移植部分进行说明。

3.4.6　触摸屏控制器

龙芯 FPGA 实验箱搭载了一块与 LCD 屏幕相同大小的触摸屏，对外使用 I2C 接口。感谢 Hypothetic 项目团队[5] 先前的移植适配工作，我们从 Hypothetic 项目的 GPU 模块中移植了触摸屏相关的子模块。该触摸屏子模块采用类似于 DMA 的方式，使用状态机对 IIC 接口进行初始化和读写控制，并将读到的结果存储在寄存器中，供总线访问。通过该模块，我们实现了对触摸屏的控制。

3.4.7　IIC 控制器

龙芯 FPGA 实验箱上板载德州仪器 DAC5571 数字/模拟信号转换芯片，使用 IIC 接口进行控制。我们使用 Xilinx 提供的 AXI IIC IP 核作为 IIC 总线控制器，控制 TI DAC5571 驱动外部的模拟信号器件，如 LED 灯。在 Linux 中，包含 Xilinx AXI IIC 总线控制器的驱动。为了便于使用一类涉及模拟信号的小外设，Linux 引入了软件层面 Industrial I/O 接口，即 iio 接口，为访问此类设备提供了标准化的用户态接口支持。Linux 也包含对 DAC5571 的 iio 支持。

3.4.8　USB 控制器

龙芯 FPGA 实验箱上搭载的 USB PHY 为 Microchip USB 3500 ，提供 UTMI+ Level 3.0 接口，支持 Host/Device/OTG 三种模式下的 USB 2.0 协议。感谢 TrivalMIPS 项目团队[6] 先前的移植适配工作，他们以开源的 UltraEmbedded USB 1.1 Host Controller IP 为基础进行修改，增加了对 UTMI 接口以及 FIFO 缓冲器的复位操作。通过该模块以及配套的驱动，我们实现了对 USB 接口的控制。驱动部分将在后文驱动移植部分进行说明。

3.4.9　VGA 控制器

龙芯 FPGA 实验箱搭载了一个标准的 VGA 接口。我们实现了一个较为简易的 VGA 控制器模块。该模块实现了在 1024*768 分辨率下的 64*24 文本输出。在该模块中，我们实例化了一个 Distributed Memory Generator IP 核作为字体 ROM 以及一个 Block Memory Generator IP 核作为文本存储器。模块地址空间分为屏幕光标寄存器和 VGA 文本缓冲区两个部分，分别存储光标位置和文本数据。在 Linux 中，我们还需要编写一个简易的驱动来实现对 VGA 的控制。驱动部分将在后文驱动移植部分进行说明。感谢哈尔滨工业大学 1 队在该模块实现上所提供的帮助。

第 4 章　系统软件

4.1　监控程序

监控程序源码由清华大学提供，团队赛要求开启监控程序的中断功能。与原生版本相比，团队赛对监控程序作了一定的改造。由于监控程序仅涵盖 20 多条指令，我们使用它验证基本的交互功能，并且它也是比赛要求运行的系统测试，故在此不再赘述。

4.2　引导程序 UBoot

4.2.1　背景

U-Boot 是一个主要用于嵌入式系统的引导加载程序，它的最终目的就是引导内核到内存，启动内核。由于我们编译得到的 U-boot 镜像较小，我们将其转换为 COE 文件，放到了 SoC 上由 AXI BRAM Controller IP 核控制的 BRAM 中。实验箱上电以后，会自动执行 UBoot ，它会完成硬件初始化等工作，便于我们进行对内核的引导。

4.2.2　编译方法

我们通过 crosstool-NG 项目制作交叉编译工具链，用于编译 UBoot。

UBoot 具体编译方法如下:

```
make ARCH=mips CROSS_COMPILE=mipsel-unknown-linux-gnu- \
chit_defconfig
make ARCH=mips CROSS_COMPILE=mipsel-unknown-linux-gnu- \
```

4.2.3　移植内容

我们感谢 CDIM 项目团队[7] 先前的移植适配工作，我们在此基础上更新了设备树，使其支持我们的 SoC 。

4.3　操作系统程序 Linux

4.3.1　背景

Linux 是著名的开源操作系统，可以运行在多种硬件平台上，如 x86、arm、SPARC、Alpha、MIPS 等。我们选择 Linux 6.3.0 作为基线进行移植。

4.3.2　硬件适配

为了使 Linux 能顺利运行在我们的 SoC 上，我们需要对内核代码进行一定的修改, 我们学习了先前参赛队伍的移植适配经验[8]，成功在我们的 SoC 上启动了 Linux。

4.3.2.1　CPU 适配

我们在内核目录 arch/mips/include/asm/mach-nscscc 下的 cpu-feature-overrides.h 文件中通过宏定义的方式去除了一些特定的指令。除此之外，由于 HELLO MIPS CPU 基本实现了所有 MIPS32 Rev1 必须的 CP0 寄存器以及指令，因此基本不需要修改内核代码，就能够正确运行 Linux 操作系统。

4.3.2.2　设备树编写

在 Linux 中，设备树用于在操作系统和硬件之间传递配置和描述信息，以实现跨平台的灵活性和可移植性。我们在内核目录 /arch/mips/boot/dts/nscscc 下存放了我们编写的设备树文件 chit.dts ，其中描述了板载所有设备的信息，包括型号、寄存器地址分配、中断号与连接关系等。

4.3.2.3　内核配置

在 Linux 中，需要对内核进行配置，加载必要的驱动来实现对硬件的支持，我们在内核目录 arch/mips/configs 下存放了我们编写的内核配置文件 CHIT_defconfig ，通过加载配置文件可以实现对实验箱上硬件的支持。

4.3.3　驱动移植

为了更好的支持实验箱上硬件，我们需要对部分驱动进行移植或更改。

4.3.3.1　SPI NOR FLASH 驱动更改

在 Linux 初始化时，会获取 FLASH 芯片的 ID 与内核中存储的 FLASH 芯片 ID 信息进行匹配，从而获得 FLASH 芯片的页面大小、擦除块数量等相关信息。由

于 Linux6.3.0 中并未存储有关 EN25F80 Flash 芯片的信息，我们需要在内核目录 drivers/mtd/spi-nor 下的 eon.c 文件中添加有关 EN25F80 Flash 芯片的信息。添加完成后，Linux 便可以成功识别板上 Flash 芯片。

4.3.3.2　USB 控制器驱动移植

USB 控制器的驱动移植来自于 TrivalMIPS 项目。我们在内核目录 drivers/usb/host 下存放了驱动文件 ue11-hcd.c ，并修改了相关的 makefile 使其能成功编译进内核。该驱动能够将自己注册为 Linux 的标准 USB 2.0 Full Speed Host Controller。经过测试，该控制器驱动可以成功识别 U 盘、USB 键盘、USB 鼠标等 USB 设备。

4.3.3.3　NT35510 LCD 驱动移植

LCD 驱动移植来自于 TrivalMIPS 项目。我们在内核目录 drivers/char 下存放了驱动文件 nt35510.c ，并修改了相关的 makefile 使其能成功编译进内核。该驱动会在 /dev 中创建名为 nt35510 的字符设备，只需向其写入 RGB565 格式的原始像素数据即可显示在屏幕上。同时，它还支持 seek 操作，可以修改屏幕的某个偏移处的像素数据，而无需全部刷新。除此之外，我们还可以在用户程序中调用 write 和 lseek 函数来控制屏幕显示。

4.3.3.4　VGA 驱动移植

我们实现了一个简易的 VGA 驱动。我们在内核目录 drivers/char 下存放了驱动文件 vga.c ，并修改了相关的 makefile 使其能成功编译进内核。该驱动实现了对 VGA 的文本数据输出，除此之外，该驱动将 VGA 注册为控制台，可以在 VGA 屏幕上显示 Linux 内核信息。

4.3.4　编译方法

Linux 编译与 UBoot 方法类似，均使用 crosstool-NG 项目制作的交叉编译工具链。

Linux 具体编译方法如下:

```
make ARCH=mips CROSS_COMPILE=mipsel-unknown-linux-gnu- \
CHIT_defconfig
make ARCH=mips CROSS_COMPILE=mipsel-unknown-linux-gnu- \
```

4.4　根文件系统

4.4.1　背景

我们使用 Buildroot 来制作根文件系统。Buildroot 是一个开源项目, 旨在简化嵌入式 Linux 系统的构建流程。使用 Buildroot 可以简单高效地构建出一个功能完善的根文件系统。

4.4.2　适配

我们在 Buildroot 目录 configs 下存放了配置文件 csystem_defconfig，其中包含了根文件系统配置以及一些所需软件包的配置，通过加载该配置文件，可以简单高效地构建出一个功能完善的根文件系统。并且，我们还将龙芯提供的交叉编译工具链 gcc-4.3-ls232 的动态链接库存放进了根文件系统中，这样我们便可以在启动的 Linux 系统中运行由 gcc-4.3-ls232 编译的可执行文件。除此之外，我们还优化了根文件系统部分配置，实现了 U 盘自动挂载以及终端路径显示等功能。

4.4.3　编译方法

根文件系统具体编译方法如下:

```
sudo make csystem_defconfig
sudo make all
```

4.5　演示程序

4.5.1　VGA 演示程序

由于 VGA 驱动没有实现自动创建设备节点的功能，所以首先我们需要在 dev 文件夹下创建 VGA 设备节点，之后我们在演示程序中通过 open 函数打开创建好的节点，通过 write 函数便可以实现面向 VGA 的文本输出。

VGA 屏幕输出如下图 4.1所示。

4.5.2　python 演示程序

Python 是一种高级编程语言，广泛用于开发各种类型的应用程序。MicroPython 则是针对嵌入式系统和微控制器的一种精简版本的 Python 实现。它专门为资源受限的环境设计，可以在资源有限的硬件上运行。我们在根文件系统中加入了 MicroPython 相关运行库，可以流畅运行 python 程序。

在演示程序中，我们实现了一个简易的计算器，可以实现加减乘除四种运算。

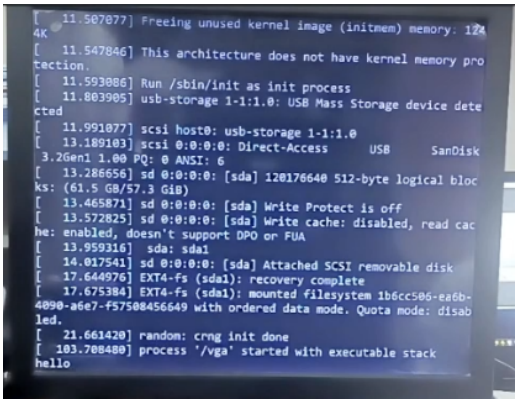


图 4.1　VGA 输出

4.5.3　幻灯片播放演示程序

我们利用实验箱上的 LCD 屏幕、触摸屏以及数码管实现了一个简易的幻灯片播放程序，可以通过触摸切换显示的幻灯片。

由于 LCD 驱动实现了自动创建节点的功能，我们在演示程序中可以直接通过 open 函数打开已经创建好的节点，通过 write 函数可以成功让 LCD 屏幕显示 RGB565 格式的图像。在切换图片时，我们可以通过 lseek 函数指定图像写入位置，再通过 write 函数写入新图像，从而实现图像切换。

对于触摸屏以及数码管，我们通过 dev 文件夹下的 mem 设备节点进行控制。mem 是 Linux 提供的一个非常特殊的设备，代表整个系统的物理地址空间而非某个真正的设备。在演示程序中，我们利用 open 函数打开 mem，通过 mmap 系统调用可以实现往自己的页表中填入映射到指定物理地址的页表项，这样就可以直接访问触摸屏以及数码管的寄存器和存储区。通过读取相应的寄存器和存储区，我们实现了对触摸屏状态的获取以及对数码管显示数字的更改。

LCD 屏幕输出如下图 4.2所示。

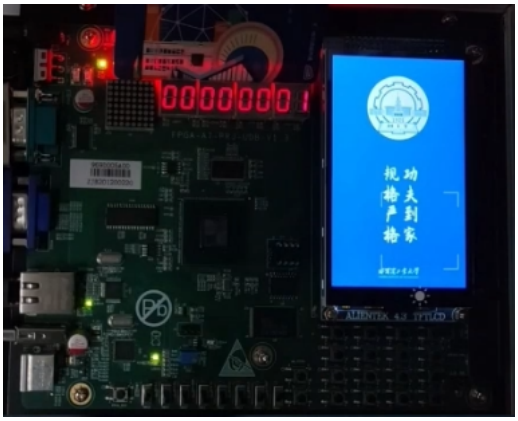


图 4.2　LCD 输出

第 5 章　测试系统

5.1　差分测试

自编写的差分测试系统 hitd 是一个仿真 verilog 代码 testbench 工程，能够迅速准确的定位大型程序中的 CPU 错误。

通过使用 verilator 将 V 代码的 CPU 编译成 cpp 文件，再和工程中仿真的内存，串口等外设编译链接，生成可执行 elf。该 elf 会仿真 V 代码的 CPU，将其指令提交结果与 CPU 模拟器 nemu 执行结果进行比对，从而达到实时 Assert 的效果，保证 CPU 的正确性。

除了能够在错误发生时迅速停止，该系统还提供了丰富的信息记录（log）帮助查找错误。

- ITRACE：记录模拟器指令执行
- MTRACE：记录模拟器访存踪迹
- ETRACE：记录模拟终端异常器
- TTRACE：记录 CPU 的 AXI 端口事务

特别是 MTRACE 和 TTRACE，极大便利了访存相关错误的查找。定位一个导致错误 load 的 store，可能在时间和波形上十分遥远。但是通过查找 MTRACE 的地址，可以迅速定位到错误的 store。图 5.1展示了使用 MTRACE 查找到 load 对应的遥远的 store。TTRACE 更是把开发者从看波形这一繁重的工作中将解脱出来，能够清楚的展示不同请求事务的握手、数据传递。图 5.2展示了使用 TTRACE 观察 AXI 事务。

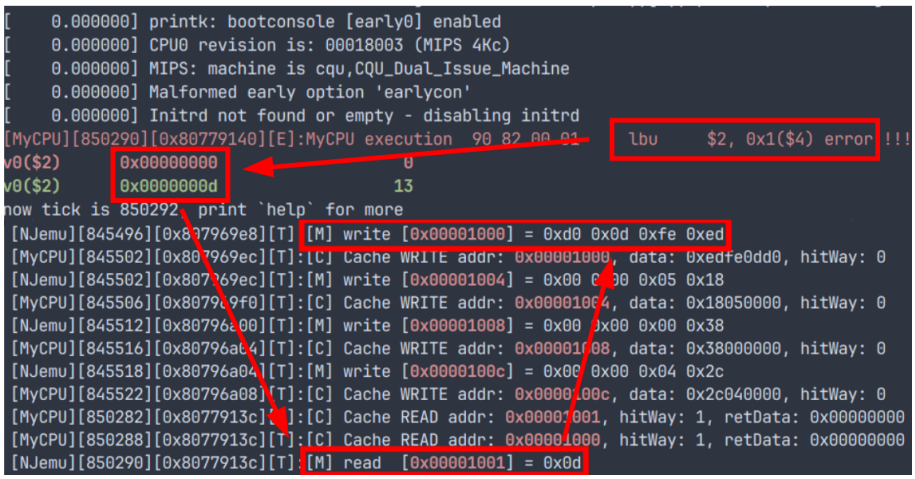


图 5.1　仿真 Linux 中 MTRACE 找出 load 错误对应的 store

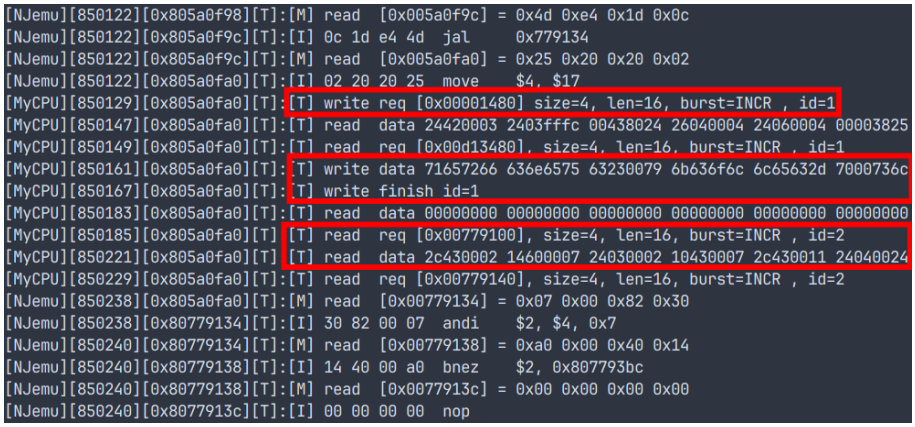


图 5.2　AXI 事务与模拟器 TRACE 交错

此外，该系统还支持添加自定义记录和自定义检查，在本次开发中，我们添加了 D-Cache 记录、BPU 记录、物理寄存器记录等。这些针对性强的记录能够清除的展示功能模块的行为，将错误快速定位到波形的时刻。自定义检查则会在指定的时刻检查功能单元的输入和输出，甚至可以编写模块级别的差分测试。在本次开发中实现了如下模块级别的差分测试:

- 测试分支预测器是否正确的读出和写入
- 测试物理寄存器的总数守恒
- 测试写缓冲入队数目等于队列中元素个数加上出队个数
- 测试访存索引预测器是否正确读出和写入

图 5.3展示了 D-Cache 的出错的 CTRACE，标出部分地址在同一行，然而第一次命中，第二次不命中。图 5.4展示了由于 Free List 实现错误导致的物理寄存器的总数不守恒报错，记录表示 48 和 49 号寄存器从 ROB 消失。

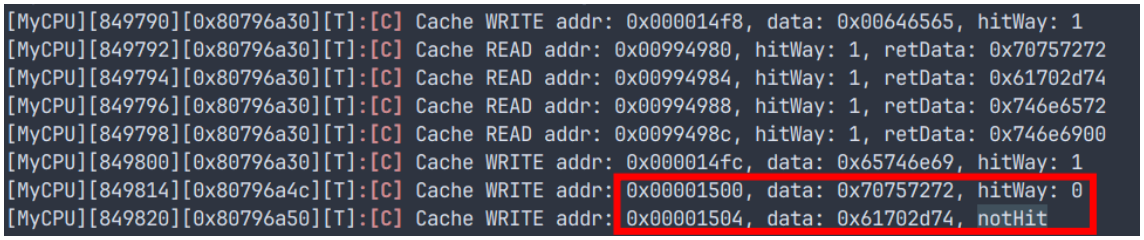


图 5.3　ctrace 指示明显错误

5.2　性能导向优化

hitd 不仅是一个测试系统，也是一个优化系统。该系统能够记录每个基本块、每条指令的执行时间，生成 bin 文件，可编写脚本分析 bin 文件的数据，报告中所有的分析数据均来自于该系统。下面是一个利用该系统进行性能导向优化的简单

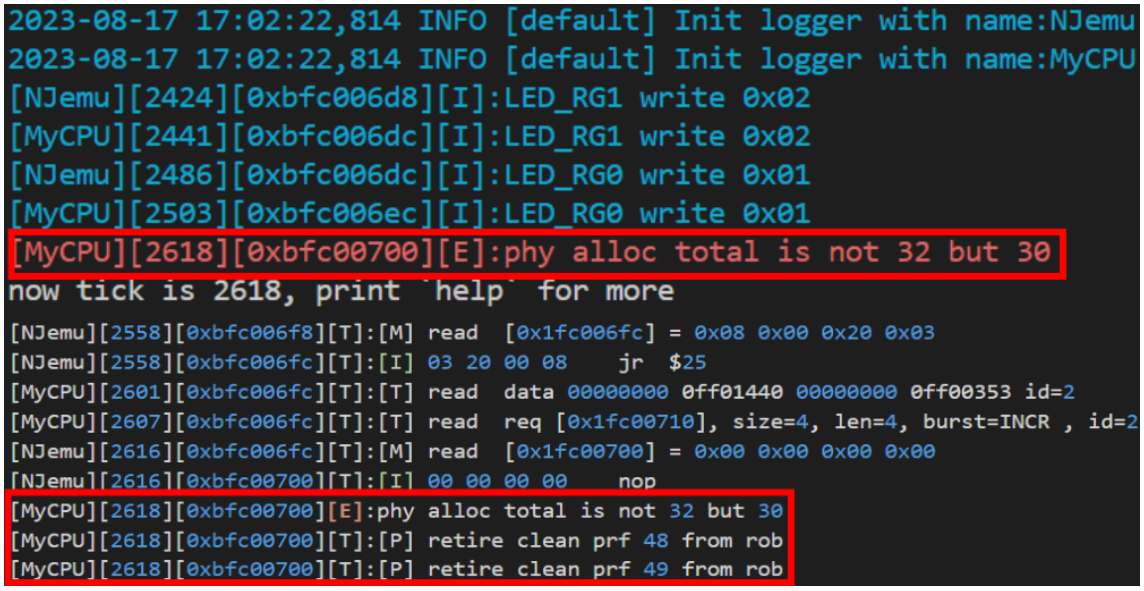


图 5.4　物理寄存器个数不守恒

例子:

图 5.5展示执行 select sort 程序的基本块和指令消耗的时间数据。从左到右的蓝色柱高代表基本块的运行总时间，紫色虚线用于分割各个基本块，红色点和绿色虚线表示该基本块各个指令的 IPC。其中执行时间占比第 2 高的基本块的指令序列和执行时间如表 5.1所示。尽管 slt 与 bnez 之间存在数据依赖，但在本设计中，bnez 和 addiu 指令可以同时执行提交，IPC 明显存在问题。观察波形可发现该问题可通过优化派发段来解决，最终实现了优化后的 IPC。

表 5.1　select sort 运行时间第二长的基本块

指令	优化前 IPC	优化后 IPC
slt t0,a1,t2	0.4459	1.1002
bnez t0,9fc0f30c	0.9982	0.5123
addiu a3,a3,12	0.9957	0.5123

由此可见，hitd 的优化系统为发者展示主要基本块的运行细节和统计数据，为性能优化提供分析来源。

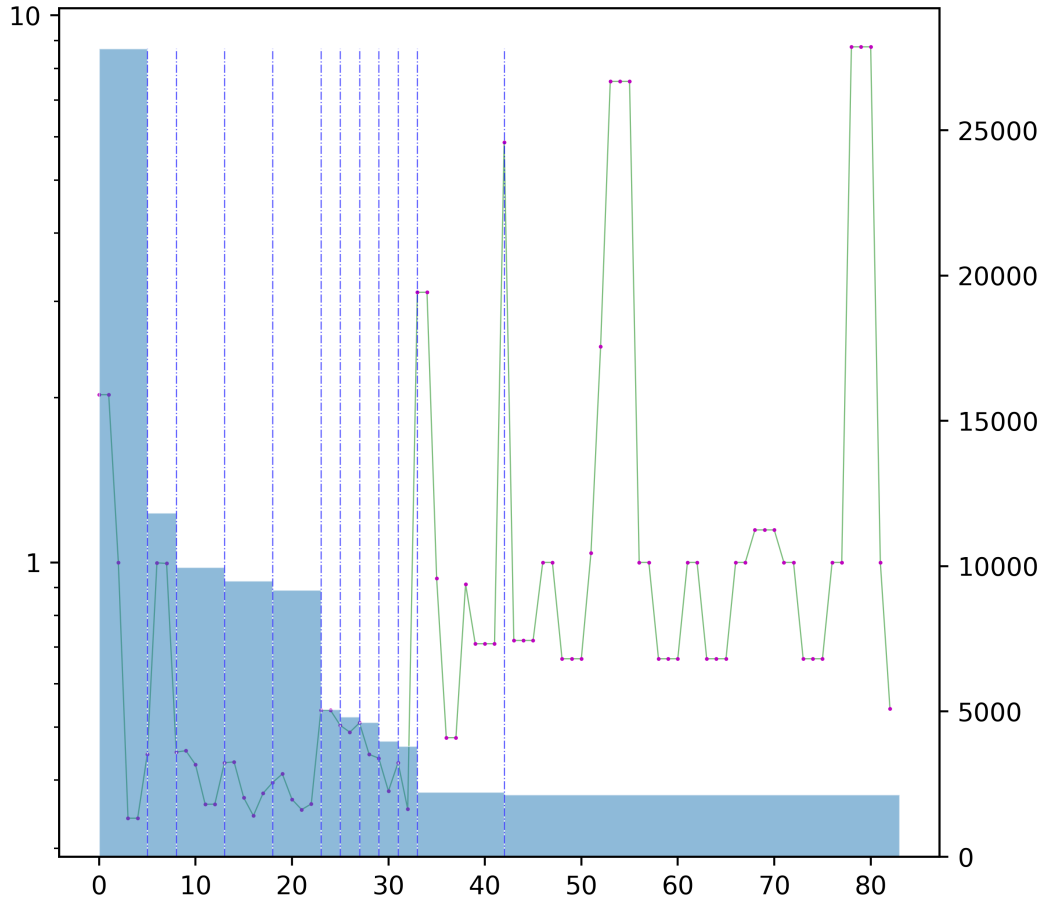


图 5.5　select sort 基本块执行时长和指令 IPC

插图和附表清单

参考文献

[1] chipsalliance. rocket-chip cache replacement[EB/OL]. 2023. https://github.com/chipsalliance/rocket-chip/blob/master/src/main/scala/util/Replacement.scala.

[2] 陈晟祺, 周聿浩, 刘晓义, 等. reset_synchronizer in Nontrivial-MIPS[EB/OL]. 2023. https://github.com/trivialmips/nontrivial-mips/tree/master/src/utils.

[3] 陈泱宇, 李燕琴, 王梓宇, 等. URT in CDIM_SoC[EB/OL]. 2023. https://github.com/cyyself/CDIM-SoC/tree/master/CDIM-SoC.srcs/sources_1/imports/APB_DEV/URT.

[4] 陈晟祺, 周聿浩, 刘晓义, 等. nt35510 controller in Nontrivial-MIPS[EB/OL]. 2023. https://github.com/trivialmips/nontrivial-mips/tree/master/vivado/ip_repo/nt35510_controller.

[5] 柴可、戴天宇、胡起、胡森. Hypo_SoC[EB/OL]. 2023. https://github.com/RickyTino/Hypo_SoC.

[6] 陈晟祺, 周聿浩, 刘晓义, 等. usb host controller in Nontrivial-MIPS[EB/OL]. 2023. https://github.com/trivialmips/nontrivial-mips/tree/master/vivado/ip_repo/usb_host_controller.

[7] 陈泱宇. cdim in u-boot[EB/OL]. 2023. https://github.com/cyyself/u-boot/tree/cdim_soc.

[8] 陈泱宇. cdim in linux[EB/OL]. 2023. https://github.com/cyyself/linux/tree/cdim_soc.

附录 A　补充内容

补充正文中提到的推论的支持数据，均使用差分测试 hitd 运行一次程序得到。

A.1　访存索引竞争预测

增加地址计算流水　如表 A.1直接增加一段用于地址计算的流水的会导致较大的时间开销。特别是对访存有较多数据依赖的程序，string copy 增加了近 10%。

表 A.1　程序总运行时间

程序名	三段（icks）	四段（ticks）	增加比 (%)
bitcount	20621	20817	0.95%
bubble sort	123717	137004	1.07%
coremark	394364	425753	7.99%
crc32	141039	143683	1.87%
dhrystone	38916	41068	5.51%
quick sort	184248	198716	7.96%
sort	108004	116325	7.70%
sha	109694	117128	6.77%
string copy	13718	15079	9.88%
string search	93659	99301	6.03%

直接映射预测失误　直接映射预测对大部分程序有较好的提升作用，除了 bubble sort 测试程序。表 A.2显示，近一半的失误来自于直接命中缓冲。对于该程序，源操作数的预测效果更好。

表 A.2　直接缓冲在 bubble sort 中的命中失误情况

情况	缓冲找到次数	总次数	比例
命中	11284	43919	25.69%
失误	13222	27622	47.87%