中国科学技术大学

专业硕士学位论文

（专业学位类型）

RISC-V 指令集模拟器的设计与实现

作者姓名： 王昊

专业领域： 软件工程

校内导师： 汪增福 教授

企业导师： 侯锐 研究员

完成时间： 二〇二二年五月十八日

University of Science and Technology of China

A dissertation for master's degree

(Professional degree type)

Design and implementation of RISC-V instruction set simulator

Author:	Wang Hao
Speciality:	Software Engineering
Supervisor:	Prof. Wang Zengfu
Advisor:	Prof. Hou Rui
Finished time:	May 18, 2022

中国科学技术大学学位论文原创性声明

本人声明所呈交的学位论文，是本人在导师指导下进行研究工作所取得的成果。除已特别加以标注和致谢的地方外，论文中不包含任何他人已经发表或撰写过的研究成果。与我一同工作的同志对本研究所做的贡献均已在论文中作了明确的说明。

作者签名：_____ 签字日期：_____

中国科学技术大学学位论文授权使用声明

作为申请学位的条件之一，学位论文著作权拥有者授权中国科学技术大学拥有学位论文的部分使用权，即：学校有权按有关规定向国家有关部门或机构送交论文的复印件和电子版，允许论文被查阅和借阅，可以将学位论文编入《中国学位论文全文数据库》等有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存、汇编学位论文。本人提交的电子文档的内容和纸质论文的内容相一致。

控阅的学位论文在解密后也遵守此规定。

☑ 公开 ☐ 控阅（____ 年）

作者签名：_____ 导师签名：_____

签字日期：_____ 签字日期：_____

摘　要

众所周知, 芯片产业是投入极高、回报极慢的领域, 而 RISC-V 提供了免费开源、开发周期较短的解决方案。面对国外芯片的生态和专利壁垒,RISC-V 有望成为我国自主研制处理器的极好的选择。而在集成度越来越高的今天, 面对数千万乃至上亿晶体管的规模, 那种“设计硬件原型-实现-评估-改进-再实现”的模式已经无法满足现代设计应用的需求, 因此在芯片开发项目中实现一款体系结构模拟器有着重要的现实意义。

本文设计并实现了一款 RISC-V 指令集模拟器, 能够完成对 RISC-V 架构处理器的功能模拟, 包括指令集功能模拟,CPU 和总线模拟, 平台级中断控制器模拟等, 并且提供了 UI 调试窗口, 可以直接对 RISC-V 架构目标程序进行可视化的调试。本模拟器采用了基于解释型的指令集模拟策略, 能够提供指令级别的仿真, 并且采用了信号与槽机制进行对象间通信, 能够以接近宿主机的速度进行跨平台程序开发和测试。该模拟器的主要功能是脱离硬件平台进行系统软件的移植开发工作, 能够极大地缩短芯片开发后期的软件适配过程, 另外也能够辅助进行处理器验证。

测试表明, 在本模拟器上的软件开发, 测试, 迭代周期, 相较于在 RISC-V 硬件仿真平台上, 缩短了 90% 以上, 并且还能够提供丰富的调试手段, 极大地降低了调试难度。本模拟器已在实际的芯片开发过程中承担了系统软件的前期移植工作, 在流片之前完成了 Linux 内核的适配移植, 并辅助进行了部分外设的驱动程序开发工作。

关键词：RISC-V; 模拟器; 指令集; 中断控制器; 调试

ABSTRACT

As we all know, the chip industry is an area with extremely high investment and extremely slow returns, and RISC-V provides a solution that is free and open source, with a short development cycle. In the face of the ecosystem and the patent barriers of foreign chips, RISC-V is expected to become an excellent choice for China's independent development of processor. In today's high level of integration, the "design hardware prototype - implementation - evaluation - improvement - re-implementation" model can no longer meet the needs of modern designs, so it is of great practical significance to implement an architecture simulator in chip development projects.

This dissertation designs and implements a RISC-V instruction set simulator, which can complete the functional simulation of the RISC-V architecture processor, including instruction set function simulation, CPU and bus simulation, platform-level interrupt controller simulation, etc., and provides a UI debugging window, which can directly visualize the debugging of the RISC-V architecture target program. This simulator adopts an interpreted instruction set simulation strategy, which can provide instruction-level simulation, and adopts signal and slot mechanism for inter-object communication, which can develop and test cross-platform programs at a speed close to the host machine. The main function of the simulator is to carry out the porting and development of system software from the hardware platform, which can greatly shorten the software adaptation process in the later stage of chip development, and can also assist in processor verification.

The test shows that the software development, testing, and iteration cycle on this simulator is more than 90% shorter than that developed on the RISC-V hardware simulation platform, and it can also provide a wealth of debugging methods, which greatly reduces the debugging difficulty. This simulator has undertaken the pre-migration of the system software in the actual chip development process, completed the adaptation and porting of the Linux kernel before the tape-out, and assisted in the driver development of some peripherals.

Key Words: RISC-V; Simulator; Instruction Set Architecture; Interrupt Controller; Debug

目　　录

第 1 章　绪　　论

1.1　系统开发背景

半个世纪以来, 集成电路产业伴随着摩尔定律一路高歌猛进, 为人类进入信息化时代提供了强大的算力支撑, 芯片俨然已经成为信息时代的原油。在如今逆全球化的浪潮中, 芯片行业首当其冲, 成为了大国博弈的前沿战场。我国的集成电路产业经过了几十年的发展, 在某些应用领域也取得了不错的成绩, 但在高性能处理器以及芯片设计的关键技术中, 仍然面临着巨大的挑战[1-2]。不管是 2018 年中美贸易战中” 中兴事件” 和” 华为事件” 的发生, 还是其他国家对我国实行的禁运政策, 都让我国在市场竞争中处于下风, 除了光刻机等硬件生产设备, 在工业软件乃至指令集架构等领域, 都存在” 卡脖子” 的现象。

作为软硬件之间至关重要的接口, 指令集架构是处理器的灵魂。长期以来, 主流的指令级架构作为少数大公司的知识产权, 几乎垄断了全球的芯片设计与制造行业, 成为绕不开的技术壁垒。不过近年来随着部分开源的指令级架构的诞生, 这一局面有望被打破。RISC-V 作为一款完全开源的指令集架构, 在众多开源指令集架构中脱颖而出, 不仅是因为其设计理念先进, 采用模块化设计的方案, 而且因为它开放, 包容, 提供了高度灵活的配置空间, 在积极拥抱开源软件的今天,RISC-V 已经成为未来的主流架构, 随着 RISC-V 开源社区的日益壮大, 更多的芯片设计厂商选择 RISC-V 作为其指令集架构, 指导芯片的设计生产[3]。

随着摩尔定律的减缓, 在如今的集成度面前, 传统的“设计硬件原型-实现-评估-改进-再实现”的芯片开发模式已然落伍[4]。有研究表明, 在实际的芯片开发项目中, 如果在前期方案论证阶段没有体系结构模拟器的支持, 将很难及时发现设计的缺陷和性能瓶颈, 导致后期的工作变得异常困难, 且会伴随高昂的代价[5]。除此之外, 在芯片开发后期, 软硬件适配工作作为测试的重点, 往往也需要模拟器环境的支持, 厂商可以根据自身产品特性, 设计相应的性能或功能模拟器, 并在此基础之上进行软件移植工作和前期软硬件设配工作, 以此来提高芯片验证与测试工作的效率。从经济成本和时间成本的角度来看, 体系结构模拟器都可以为芯片设计与验证工作提供极大的便利。相较于一些主流指令集架构, 当前 RISC-V 的软件生态尚不完善, 各类软件的 RISC-V 移植工作正在开源社区如火如荼地进行中, 对于芯片设计和开发厂商, 系统软件的开发移植更是重中之重。因此开发一款符合自身产品特性的 RISC-V 模拟器将成为贯穿芯片开发周期的一项重要的软件工程。

1.2　国内外发展现状

龙芯中科技术有限公司在龙芯 2 号处理器研发过程中, 开发了模拟器 ICT-Godson[6], 对硬件进行了细粒度的模拟, 能够精确地模拟芯片开发过程中的所有硬件信号和行为, 但是由于模拟精度过高, 导致模拟器执行速度太慢, 难以高效地进行辅助工作。因此, 基于开源模拟器 Simple-Scalar[7], 龙芯中科公司重新实现了 Sim-Godson 模拟器[8], 该模拟器摒弃了细粒度的硬件仿真, 采用执行驱动的组织方式, 以及模块化的设计, 可以支持时序模拟和功能模拟[8], 相较于前一版的模拟器性能得到了极大的提升。使用该模拟器可以直接加载目标程序的二进制可执行文件, 通过功能模拟对目标机器进行微结构性能探索[9]。

Gem5 是一款开源的系统级体系结构模拟器, 它结合了 CPU 模拟框架 M5 和内存时序模拟器 GEMS 的特点, 是一款具有多种执行模式的事件驱动模拟器[10],Gem5 能够模拟多种不同架构的处理器, 包括 Alpha、ARM、X86、SPARC、MIPS 等, 近年来也支持了 RISC-V 架构, 借助于其开源的强大合作能力,Gem5 不断完善自身, 在工业界和学术界都极受欢迎。Gem5 的灵活性依赖于其面向对象的设计,85% 的 Gem5 源码由 C++ 编写, 在 Gem5 中可以选择不同的指令集架构和 CPU 模型。对于部分指令集架构,Gem5 支持全系统模式 (Full System) 运行, 在该模式下,Gem5 可以运行操作系统, 这样也就可以进行中断、异常、特权级别、I/O 等的模拟。

IBM 在处理器开发的不同时期, 使用不同类型的体系结构模拟器进行辅助。在前期设计与规划时期, 使用 Mambo[11]模拟器的时钟精确模式进行微结构探索和粗粒度微结构定义, 该阶段体系结构模拟器采用踪迹驱动方式[12], 主要运行和研究用户态应用, 对处理器的产品竞争力进行横向比较研究[13]。在微结构设计实现期间,IBM 使用基于公司内部专用“T”语言[14-15]编写的时钟精准模拟器 M1 进行详细模拟[13], 如图 1.1所示。M1 是以 Mambo 模拟器或者硬件上抓取的踪迹作为输入, 并且可以收集非常详细的微结构数据进行性能评估[16]。

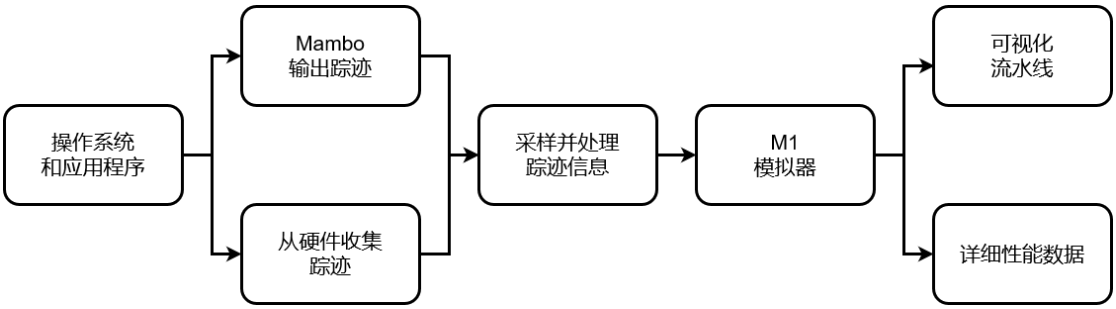

图 1.1　IBM 体系结构模拟器框架

目前 RISC-V 开源社区的体系结构模拟器有 Spike, Gem5, Qemu[17]等, 这一类模拟器能够对部分 RISC-V 开源架构如 BOOM[18]进行模拟, 对于新手熟悉

RISC-V 有很大的帮助。Spike 是一款基于解释型的 RISC-V 指令集模拟器, 能够提供指令级别的仿真, 跟开源社区工具链中的 pk(proxy kernel) 和 fesrv(front-end server) 结合能够完成全系统的模拟, 并且能够提供命令行的调试模块。Qemu 是一款基于编译型的模拟器[19], 直接对 RISC-V 二进制流进行翻译, 模拟速度接近宿主机。在此类 RISC-V 模拟器上也能够进行软件移植工作, 主要是基于 Linux 内核的应用软件移植, 下面重点介绍下指令集模拟器 Spike。

Spike 模拟器属于非时钟精确模型, 使用 C++ 实现, 在 Spike 模拟器上可以运行以下三种类型的 RISC-V 目标代码: (1) bare metal;(2) 使用 newlib;(3) 使用 glibc。

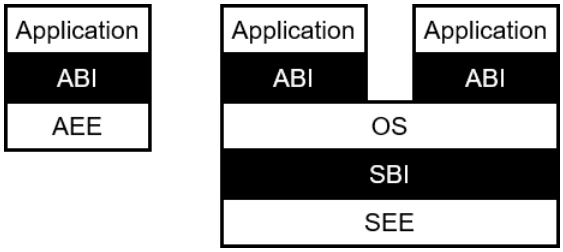

图 1.2　RISC-V 支持的软件栈

裸机代码可以直接在 RISC-V 硬件上执行,Spike 运行裸机程序无需加载 pk, 如图1.2, 根据 RISC-V 软件栈[20-21]的定义, 此时并不存在应用程序执行环境 (Application Execution Environment, AEE), 目标程序也不能够直接和宿主机进行 I/O 交互。pk 是一个轻量级内核, 预定义了主存的基地址等信息, 并且绑定了一些基本的库 (newlib), 是软件栈中 AEE 的位置, 通过 pk 和 fesvr 可以进行 I/O 系统调用, 与宿主机交互。glibc 提供了 OS 的接口, 此时应用程序二进制接口 (Application Binary Interface, ABI) 访问的是 OS。

Spike 模拟器配合 pk、fesrv 完成系统模拟的原理图如图1.3所示。

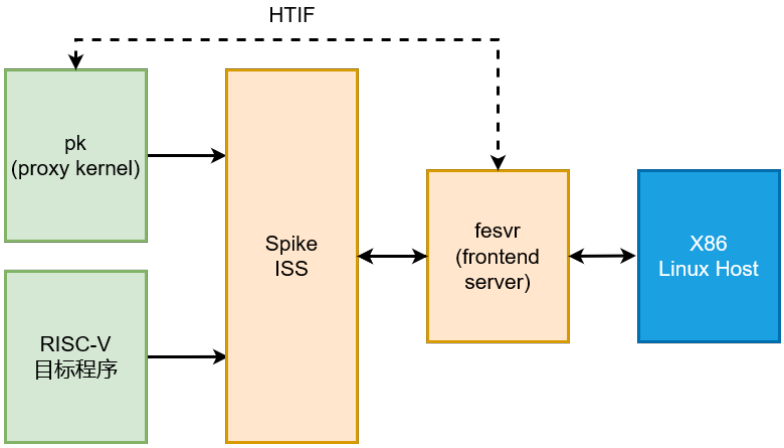

图 1.3　spike 系统模拟原理图

RISC-V 目标程序通过 pk 提供的接口在模拟器程序上运行, 对应的 RISC-V 系统调用经过 fesvr 转换成宿主机上的 X86 系统调用, 以此来实现目标机与宿主

机的交互。

Spike 模拟器由 RISC-V 官方进行维护, 能够提供最新的指令集版本模拟, 基于 Spike 能够方便地进行 RISC-V 应用程序的移植和开发, 但是也存在以下瓶颈:

1) 不提供平台级中断控制器模拟, 因此也不能够方便地进行外设驱动程序的适配工作。

2) 不提供可视化的调试模块, 对于实际的 RISC-V 芯片开发团队来说, 易用性不够。

1.3　本文主要工作

为了加快芯片开发项目中的系统软件开发过程, 以及辅助进行处理器验证, 本文从 RISC-V 芯片开发项目工作流程入手, 设计并实现了一款针对 RISC-V 体系结构的指令集功能模拟器, 可以使得芯片开发团队脱离硬件平台并行地进行系统软件移植, 开发和测试工作, 本模拟器提供对真实硬件的功能模拟, 以及丰富的调试手段, 缩短软件开发周期, 辅助处理器验证。主要完成的工作如下:

(1) 参照 RISC-V 用户手册, 特权级架构文档, 以及实际的硬件配置方案, 对 RISC-V 架构处理器进行功能建模, 对 RISC-V 标准拓展指令集共 196 条指令进行 C++ 功能函数模拟, 结合系统软件开发人员的需求, 确定指令集模拟器的模拟策略, 规划具体功能模块的边界, 采用面向对象的方法进行模块设计。

(2) 实现 RISC-V 指令集模拟器前后端模块, 包括预加载模块,CPU 和总线模块, 中断控制器模块, 调试和 UI 模块。设计并实现平台级中断控制器和部分外设模拟, 可以直接在模拟器平台上进行外设驱动的适配；结合实际项目需求, 实现 UI 可视化界面进行交互调试, 进一步加强模拟器的易用性。在实际的芯片开发项目中帮助团队进行系统软件开发工作, 并对处理器进行辅助验证。

本人负责的主要工作有模拟器前后端整体框架的搭建；CPU 和总线模块的设计与实现；平台级中断控制器的功能建模与实现；调试模块的 UI 界面设计；测试模拟器是否符合设计需要。

1.4　论文的组织结构

本论文总共分为 7 章, 各章节内容安排如下：

第一章：绪论。本章首先简要介绍了 RISC-V 指令集架构和体系结构模拟器的背景, 然后对国内外主要体系结构模拟器进行举例分析, 进而确定了本模拟器的设计思路, 最后阐述了本文的主要工作以及论文的组织结构。

第二章：相关技术分析。本章首先介绍了指令集架构的基本知识, 重点分析

了 RISC-V 架构的特点, 然后对体系结构模拟器的相关技术进行分析, 包括体系结构模拟器的类别和作用, 以及不同指令集模拟策略的相关技术, 然后针对 RISC-V 指令集架构特点进行技术选型, 确定了基于解释型的模拟策略作为本模拟器的设计方案。

第三章：系统需求分析。本章主要是采用面向对象分析的方法对模拟器用户的需求进行分析与建模, 确定模拟器的主要业务流程, 给出模拟器功能需求和非功能性需求, 生成系统需求规格说明书。

第四章：系统概要设计。本章主要是对 RISC-V 指令集模拟器的总体框架进行设计与描述, 包括系统的静态软件结构、动态运行流程, 确立了模拟器的四个功能模块：预加载模块,CPU 和总线模块, 中断控制器模块以及调试和 UI 模块。

第五章：系统详细设计与实现。本章主要是对 RISC-V 指令集模拟器的各个功能模块的具体实现进行阐述, 确定每个模块的功能边界, 数据结构定义和接口细节, 并进行具体实现。

第六章：系统测试。本章主要是对 RISC-V 指令集模拟器进行单元测试、配置项测试和系统测试, 并通过测试结果分析模拟器功能和性能的实现情况。

第七章：结论与期望。本章对 RISC-V 指令集模拟器的设计与实现过程进行总结, 对模拟器的实际使用情况进行介绍, 最后分析了模拟器设计存在的缺陷, 以及实际使用过程中存在的瓶颈, 提出了后续可以改进的方向。

第 2 章　相关技术分析

从第一章的分析可以看出, 当前体系结构模拟器的研究主要是基于主流的指令集架构, 从处理器功能和性能两个不同的角度对模拟器进行设计。所以本章首先分析主流指令集架构和 RISC-V 架构的特点, 然后介绍体系结构模拟器的相关技术, 最后结合本系统的设计目标－－为 RISC-V 架构的系统软件提供移植环境并辅助进行处理器验证, 提出了基于解释型的指令集模拟策略, 对 RISC-V 架构处理器进行功能模拟。

2.1　指令集架构概述

指令集架构（Instruction Set Architecture, ISA）是计算机体系结构中定义的软硬件接口规范, 是信息技术生态的原始起点[22]。根据同一指令集架构, 不同的厂商可以设计出性能各异的处理器, 虽然生产的芯片会有成本, 功耗, 性能上的区别, 但是它们都可以运行基于同一指令集架构开发的软件。指令集架构主要分为复杂指令集（Complex Instruction Set Computer, CISC）和精简指令集（Reduced Instruction Set Computer,RISC）, 前者使用不定长的指令设计方案, 单条指令的功能实现有相当大的空间, 但是由于保留了许多复杂指令, 使得 CPU 设计变得异常繁杂, 大大增加了硬件设计的时间成本和面积开销[23]；而后者采用定长的指令设计方案, 只保留处理器常用指令, 因此指令结构较为精简, 相应的硬件设计也会比较简洁。这两种指令集架构的代表分别是 X86 架构和 ARM 架构。其中 X86 架构因为其向下兼容的特点在 PC 领域占有巨大的市场份额, 而 ARM 架构因为低功耗的特点被广泛用于嵌入式领域。这两种指令集架构的共同特点是闭源, 并且授权费高昂, 当其他拿到授权的公司生产的芯片使得 Intel 和 ARM 公司受到威胁时,Intel 和 ARM 完全可以拿起专利的大棒停止授权[1]。

2.2　RISC-V 架构

第五代精简指令集 (RISC-V, Reduced Instruction Set Computer - Five) 于 2010 年诞生于加州大学伯克利分校[24], 由 David.Paterson 教授团队设计, 由于其开放性和包容性, 以及充分发挥的后发优势, 如今的 RISC-V 已经成为行业实施的标准开源指令集架构,RISC-V 开源采用宽松的 BSD 协议, 企业可以完全自由免费使用, 同时也容许企业添加自有指令集拓展而不必开放共享以实现差异化发展。RISC-V 提供了详细的特权级指令规范和用户级指令规范, 指令集使用模块化的方式进行组织, 试图通过一套统一的架构满足各种不同的应用场景, 包含标准拓

展的 RISC-V 指令集组成如表 2.1所示。

表 2.1　RISC-V 指令集模块

指令集类型	类型简写	指令数	说明
基本指令集	RV32I	47	基本整数指令集, 包含算数指令、访存指令、环境调用等指令
	RV32E	47	RV32I 指令集简化版本, 专为嵌入式设计
	RV64I	59	整数指令
	RV128I	71	整数指令
扩展指令集	M	8	乘除扩展
	A	11	原子扩展
	F	26	单精度浮点扩展
	D	26	双精度浮点扩展
	C	46	压缩指令扩展

其中, RV32 和 RV64 表示寄存器的位宽, 决定了处理器寻址范围的大小。基本整数指令集是所有 RISC-V 架构设计方案都必须实现的最小子集, 其余的统称为拓展指令集, 是可选的。

RISC-V 指令集架构所使用的定点通用寄存器如表2.2所示, 浮点通用寄存器如表2.3所示。

表 2.2　RISC-V 定点通用寄存器

寄存器	助记符	描述	调用与被调用
x0	zero	硬编码为 0	-
x1	ra	返回地址寄存器	调用者
x2	sp	堆栈指针寄存器	被调用者
x3	gp	全局指针寄存器	-
x4	tp	线程指针寄存器	-
x5	t0	临时/备用链接寄存器	调用者
x6-7	t1-2	临时寄存器	调用者
x8	s0/fp	保存的寄存器/帧指针	被调用者
x9	s1	保存的寄存器	被调用者
x10-11	a0-1	函数参数/返回值寄存器	调用者
x12-17	a2-7	函数参数寄存器	调用者
x18-27	s2-11	保存的寄存器	被调用者
x28-31	t3-6	临时寄存器	调用者

表 2.3　RISC-V 浮点通用寄存器

寄存器	助记符	描述	调用与被调用
f0-7	ft0-7	浮点临时寄存器	调用者
f8-9	fs0-1	浮点保存寄存器	被调用者
f10-11	fa0-1	浮点参数/返回值寄存器	调用者
f12-17	fa2-7	浮点参数寄存器	调用者
f18-27	fs2-11	浮点保存寄存器	被调用者
f28-31	ft8-11	浮点临时寄存器	调用者

RISC-V 指令集有以下特点：

(1) 完全开源, 使用 RISC-V 架构无需获取授权。

(2) 模块化的指令集设计, 针对不同的应用场景, 可以选择支持不同组合的指令集模块, 有很好的灵活性。

(3) 提供可配置的通用寄存器组。

(4) 提供规整的指令编码格式, 在指令格式中寄存器索引位置对齐, 方便译码器硬件实现。

(5) 极简的设计哲学, 摒弃了分支延迟槽, 条件码执行等特性, 极大地简化了硬件电路的布局。

(6) 方便用户自定义, 用户可以对中断系统进行定制, 并且还可以实现自定义的拓展指令。

RISC-V 架构为软件开发人员提供了开源的指令集规范, 也为硬件设计人员提供的便利的指令集模块支持, 厂家可以根据自身产品特性选择适合自己的组合模块。这样的指令级架构演化模式充分调动了软硬件开发者的热情, 也吸引了众多商业芯片厂商的目光, 同时极大地降低了处理器的设计门槛, 能够方便研究者进行零成本的创新实践。按照如今 RISC-V 开源社区的发展态势, 有望在新兴的 AI 与 IoT 领域中对 ARM 的统治地位形成挑战[25], 这同时也是我国能在芯片设计领域打破技术壁垒的良好契机。

2.3　体系结构模拟器

模拟器是体系结构量化分析的重要手段, 对架构设计、芯片开发有重要的指导作用。基于模拟器辅助进行集成电路设计可以追溯到 1980 年代[26], 自此模拟器便一直是处理器设计过程中不可或缺的工具。在芯片开发项目中, 体系结构模拟器的具体作用如图 2.1所示：

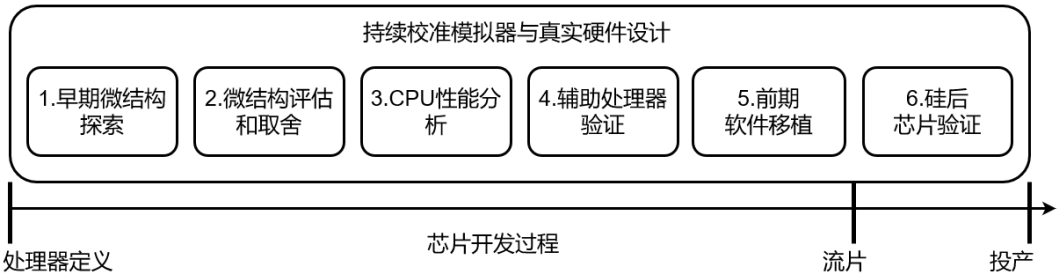

图 2.1　体系结构模拟器在 CPU 开发流程中的作用

1) 在芯片开发早期, 基于体系结构模拟器可以进行微结构探索和粗粒度微结构定义, 此时模拟器的开发抽象层次较高。

2) 随着处理器设计的不断推进和模拟器的不断完善, 基于模拟器可以持续对芯片微结构进行评估、修改和取舍。

3) 当模拟器趋于成熟, 可以对微结构、多核互联系统、一致性协议等进行详细性能分析, 基于分析结果对微结构进行微调。

4) 在对处理器逻辑设计进行验证的阶段, 模拟器可以作为参考模型辅助进行验证, 可以快速定位逻辑设计错误。

5) 在未流片之前基于模拟器就可以开展系统软件开发和适配工作, 在芯片流片结束后以最快速度启动系统软件。

6) 流片结束后, 基于模拟器可以辅助进行芯片硅后验证环境的搭建以及测试用例编写工作[27]。为了保证模拟器可以顺利辅助进行处理器设计, 在整个芯片开发过程中, 需要持续对模拟器进行校准, 通过持续对比模拟器和寄存器传输层 (Register-Transfer Level, RTL) 之间的差别, 可以互相校准并发现模拟器或者 RTL 的设计错误[28]。

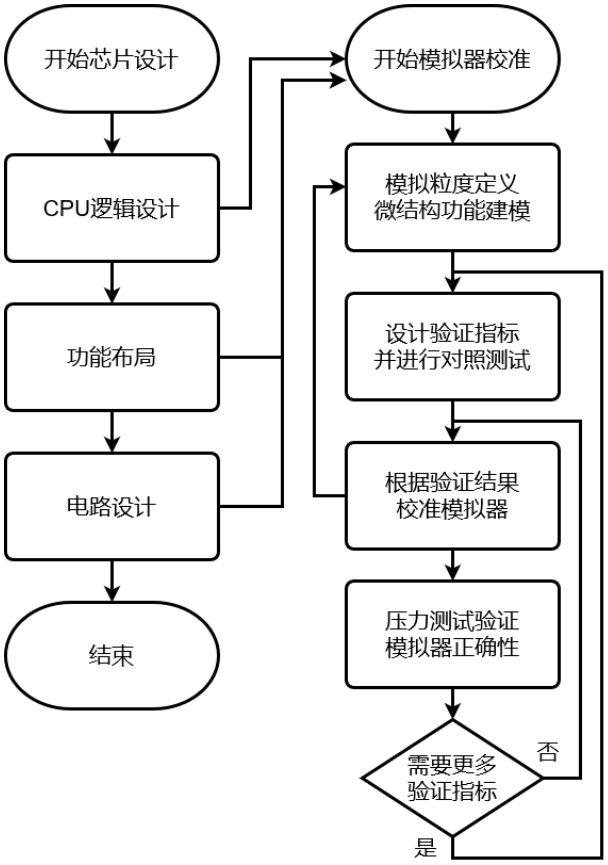

图 2.2　体系结构模拟器开发流程图

使用体系结构模拟器进行辅助开发的基本流程如图 2.2所示。首先, 需要对模拟的粗细粒度进行定义, 是功能模拟还是性能模拟, 然后需要对实际的硬件设计进行建模, 将无需模拟的部分进行抽象, 使用软件手段封装成黑箱, 这部分不必过分关心硬件的实现细节, 只需对输入输出作明确的定义。为了达到所需的模拟精度, 需要对模拟器和真实硬件进行行为比对, 此外, 建模过程中所选的具体算法需要达到对模拟器的性能要求。确定好模拟方案, 完成建模, 选定合适的实现算

法后, 进行程序设计。等确定了程序的正确性后, 就可以使用该模拟器进行后续的工作, 如对处理器系统软件的预开发移植, 或者是对模型本身进行评价或研究, 也可以是对模拟的目标系统性能作出评价。

根据模拟粒度可以将体系结构模拟器分为两类, 功能模拟器（指令集模拟器 Instruction Set Simulator, ISS）和性能模拟器（时钟周期精确模拟器）[29]。ISS 只模拟目标系统的指令集体系结构, 比如寄存器状态、指令语义、存储器状态等功能特性；性能模拟器除了模拟功能特性外, 还模拟目标系统的微体系结构, 比如流水线、分支预测等[30]。

对于指令集模拟器 ISS, 在设计过程中, 需要充分考虑指令集模拟策略以及模拟器驱动方式。指令集模拟策略是 ISS 设计的基础, 它决定了模拟器的性能, 同时也会影响模拟器调试模块的功能实现。指令集模拟策略分为两种：基于解释型和基于编译型。下面将详细介绍这两种模拟策略。

2.3.1　解释型指令集模拟

解释型指令集模拟器的最大的特点在于直接将硬件行为映射到软件, 从而模拟出真实的硬件环境, 由于其对指令进行逐条翻译, 使得指令流程控制很容易实现[31-32]。ISS 的指令控制流程通常是取指－译码－执行的循环, 如图 2.3所示。

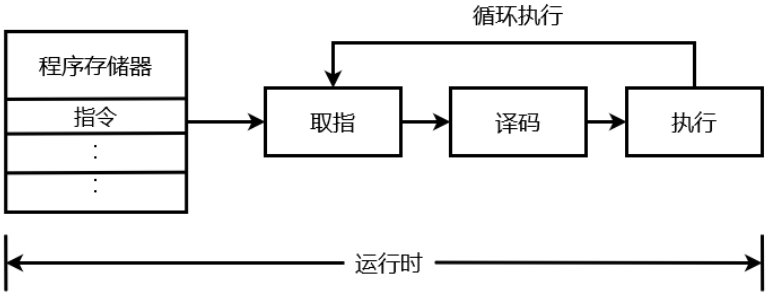

图 2.3　解释型 ISS 工作流程

1) 取指：模拟器取出目标程序的单条指令。

2) 译码：模拟器对目标指令进行翻译, 得到指令对应的功能函数。

3) 执行：模拟器执行目标指令所对应的功能函数, 完成功能函数中定义的软硬件行为。

解释型指令集模拟器的工作流程使得模拟器设计比较简单, 易于建模和实现, 且灵活性较好, 模拟精度高。但是由于模拟器通过软件行为对指令进行逐条译码, 相较于真实硬件电路效率太低[33], 所以解释型 ISS 的模拟速度一般不是很高。

2.3.2　编译型指令集模拟

基于编译型的 ISS 通过对译码过程的改进, 大幅度提升了模拟的效率, 编译型 ISS 采用二进制翻译技术[34]将目标指令翻译成宿主机可识别的指令来完成对目标机状态的模拟。根据译码过程处于编译还是运行时, 又可分为静态编译型指令集模拟器（Static Compiled ISS）和动态编译型指令集模拟器（Dynamic Compiled ISS）[35]。由 Zhu and Gajski[36]给出的静态编译型指令集模拟器将本处于运行时的指令译码过程转移至编译时, 如图 2.4所示。首先使用交叉编译器将目标程序编译成目标机架构的二进制文件, 接着使用代码生成器进行优化, 最终翻译为宿主机二进制文件, 由宿主机直接运行。这种基于二进制翻译的指令集模拟策略拥有很高的执行效率, 运行速度接近宿主机。

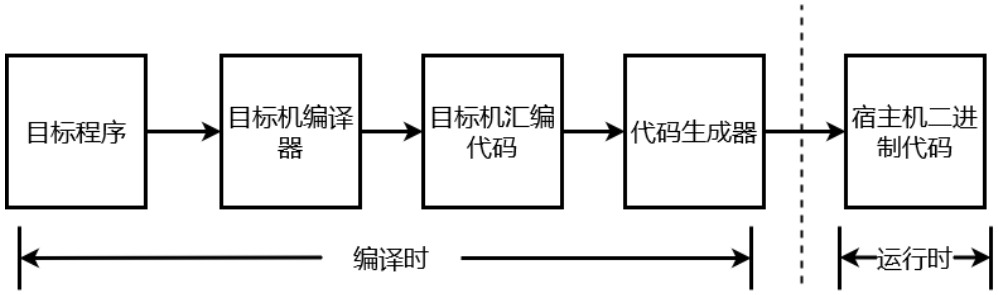

图 2.4　静态编译型 ISS 工作流程

动态编译型 ISS 的典型代表为 Embra[37]及 Shade[38], 其工作流程如图 2.5所示：

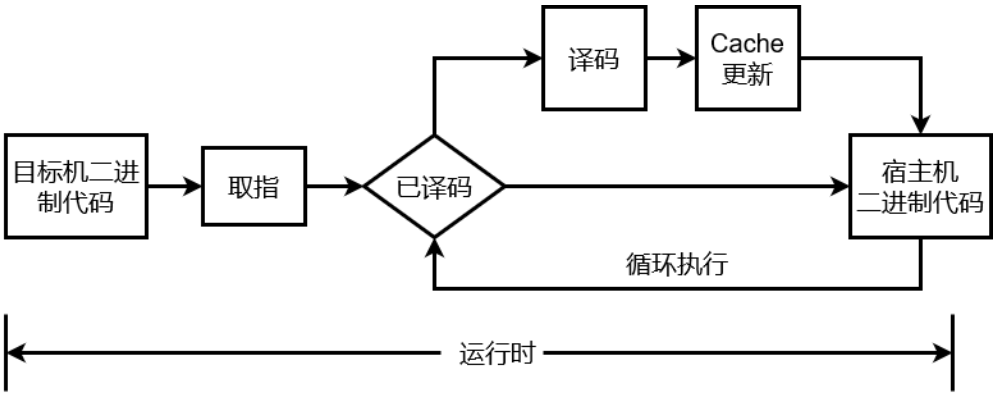

图 2.5　动态编译型 ISS 工作流程

2.3.3　模拟器技术选型

本模拟器的开发目标是对 RISC-V 架构处理器进行功能模拟, 模拟的粒度是单条 RISC-V 汇编指令, 属于指令集功能模拟器, 用于系统软件的移植开发和测试工作。为了兼顾开发和执行效率, 本模拟器采用面向对象的设计方法, 使用 C++

实现前后端代码框架, 采用解释型的指令集模拟策略, 最终完成了 RISC-V 指令集功能模拟器的开发。

2.4　本章小结

本章节主要介绍了 RISC-V 指令集架构的特点以及体系结构模拟器的相关技术。对体系结构模拟器的功能和开发流程做了介绍, 并对几种常见的模拟器类型作了分析对比, 其中详细介绍了指令集模拟器 ISS, 对指令集模拟器的模拟流程以及两种指令集模拟策略的优缺点进行了分析, 最终结合实际芯片开发项目需求, 以及 RISC-V 指令集架构的特点, 拟定了本次模拟器的设计方案：采用基于解释性的指令集模拟策略, 方便进行调试功能的实现, 并且结合了动态编译型 ISS 的译码优化策略, 提高模拟器执行效率。

第 3 章　系统需求分析

需求分析是软件生产周期中的一个重要环节, 本章将采用面向对象分析的方法对体系结构模拟器的需求进行具体分析与建模。明确模拟器所需实现的功能性需求和非功能性需求。

3.1　需求导出

在芯片设计及验证的流程中, 对于基础系统软件尤其是操作系统, 设备驱动等的适配和验证往往是反馈硬件设计缺陷较频繁的部分, 这部分的工作不仅是对于前期硬件设计的重要测试, 也是后续用户态程序开发的基础。对于系统软件的移植和适配工作, 有两种主流方式, 一种是在模拟芯片硬件特性的可编程逻辑门阵列 (Field Programmable Gate Array, FPGA) 平台上仿真, 另一种是通过软件模拟。两种方法各有利弊,FPGA 开发板更加接近真实硬件环境, 能够获取精确的仿真信号, 但是能够提供的调试手段有限, 并且软件迭代过程将花费大量的时间用于存储器烧写和刻录。而模拟器环境下的软件开发, 其运行速度接近宿主机, 并且调试手段丰富, 虽然信号精度与真实硬件有差异, 但是能够在软件移植的前期反馈大部分的缺陷并及时进行迭代。所以实际的开发和测试流程一般是先使用模拟器验证, 再上 FPGA 平台仿真, 这样既能够提高开发效率, 又不失精度。

随着 RISC-V 开源社区的日益壮大, 更多的芯片设计厂商选择 RISC-V 作为其指令集架构, 但是由于 RISC-V 架构的软件生态尚不成熟, 有大量的系统软件还未得到 RISC-V 支持, 因此芯片厂商将花费更多的时间在软硬件设配工作上, 而这部分工作将更多地依赖模拟器环境。当前 RISC-V 开源社区提供的模拟器作为教学工具可以帮助初学者快速上手, 但在实际的芯片开发流程中很难发挥作用, 一方面是因为开源模拟器的模拟对象与实际的处理器架构设计有很大差异, 不能够提供足够的设备模拟, 另一方面是因为开源模拟器不提供或是只提供少数的调试手段, 而在实际的 RISC-V 软件移植开发过程中非常依赖调试功能。所以对于 RISC-V 芯片开发团队来说, 设计一款符合自身硬件架构的模拟器, 同时提供强大的调试工具, 具有非常重要的现实意义。

通过在实际的 RISC-V 芯片开发项目中总结的经验, 以及对系统软件移植工作过程的痛点分析, 最终明确了 RISC-V 指令集模拟器的业务需求和工作流程, 并在此基础上建立了系统需求模型。

3.2　分析建模

指令集模拟器的主要参与者是进行系统软件开发和移植的程序员, 本模拟器主要用于芯片开发项目后期的验证工作和系统软件移植工作, 核心业务流程如图3.1所示。

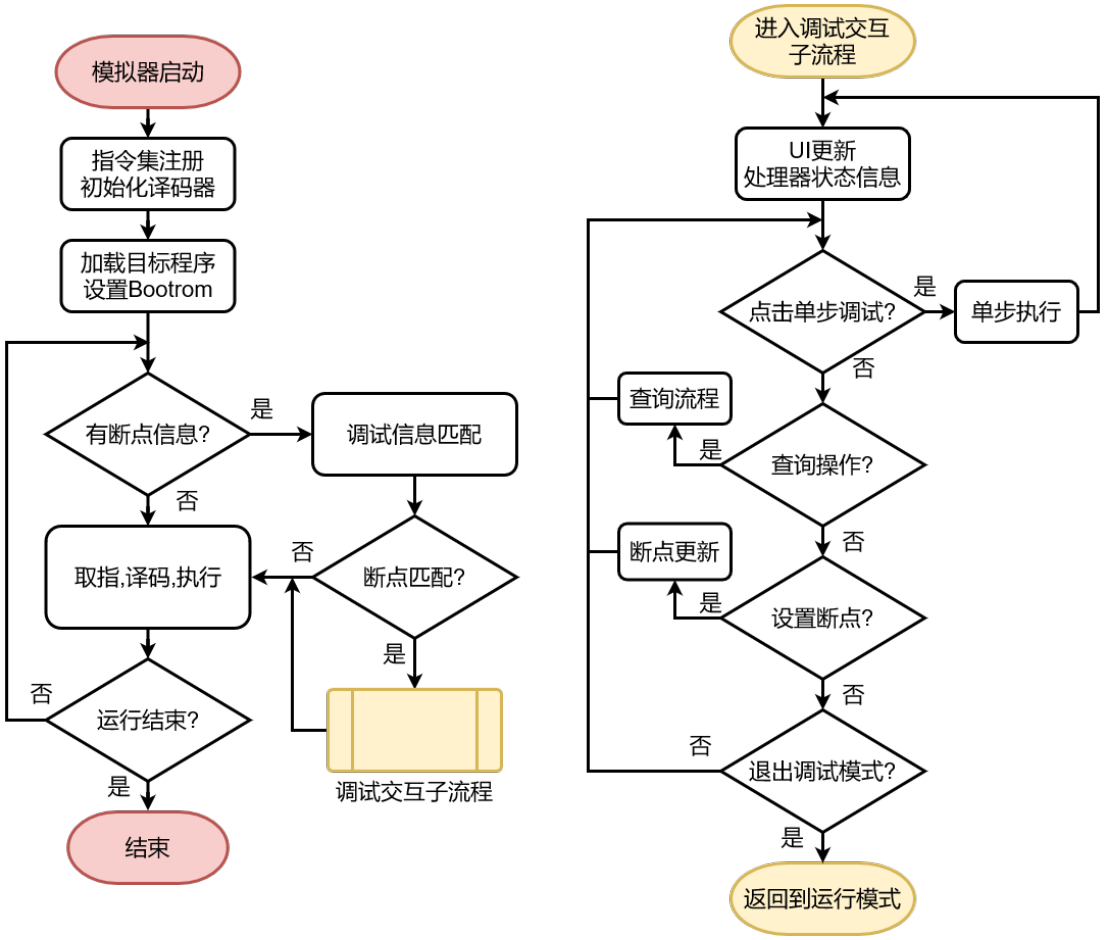

图 3.1　模拟器业务流程图

用户首先使用 RISC-V 交叉编译工具链将目标程序编译为 RISC-V 架构的 ELF 文件, 然后模拟器解析该 ELF 文件, 将对应的指令流搬运到内存起始位置, 模拟器在配置启动后为处理器注册指令集, 初始化译码器, 逐条进行取指、译码、执行。指令译码器完成包括操作数在内的指令信息提取, 找到该条指令注册时对应的功能函数, 执行该功能函数, 然后将更新后的寄存器状态信息, 内存状态信息同步到前端 UI 显示模块。在模拟器运行的过程中, 用户还可以通过前端交互调试窗口来切换模拟器运行模式, 设置断点触发条件, 进行单步调试, 状态查询等操作。通过对实际芯片开发验证过程的分析和归纳, 得出模拟器所需要的主要功能有:

1) 设置模拟器启动配置, 包括 ELF 文件路径添加, 指令集模块注册, 运行模式选择等。

2) 模拟器执行流程控制。包括正常运行模式下的 UART 串口交互, 暂停执行进入调试模式, 模拟器重启等。

3) 调试功能。在调试模式下, 进行断点设置, 内存查询, 历史指令查询, 单步执行等。

4) 模拟外部中断信号发送。

综上所述, 用户需求描述表如表 3.1所示。

表 3.1　用户需求描述表

名称	参与者	说明
模拟器配置并启动	系统软件开发人员	设置模拟器启动参数并运行
运行模式切换	系统软件开发人员	模拟器由运行模式切换为调试模式, 由调试模式切换为运行模式
重启模拟器	系统软件开发人员	重新加载当前配置项并运行
断点设置	系统软件开发人员	调试模式下进行断点添加/移除
内存查询	系统软件开发人员	调试模式下对虚拟地址/物理地址内容查询
中断信号发送	系统软件开发人员	点击外部中断源按钮, 发送对应的外部中断到平台级中断控制器

根据用例描述表可以得出系统软件开发人员的用例图如图 3.2所示。

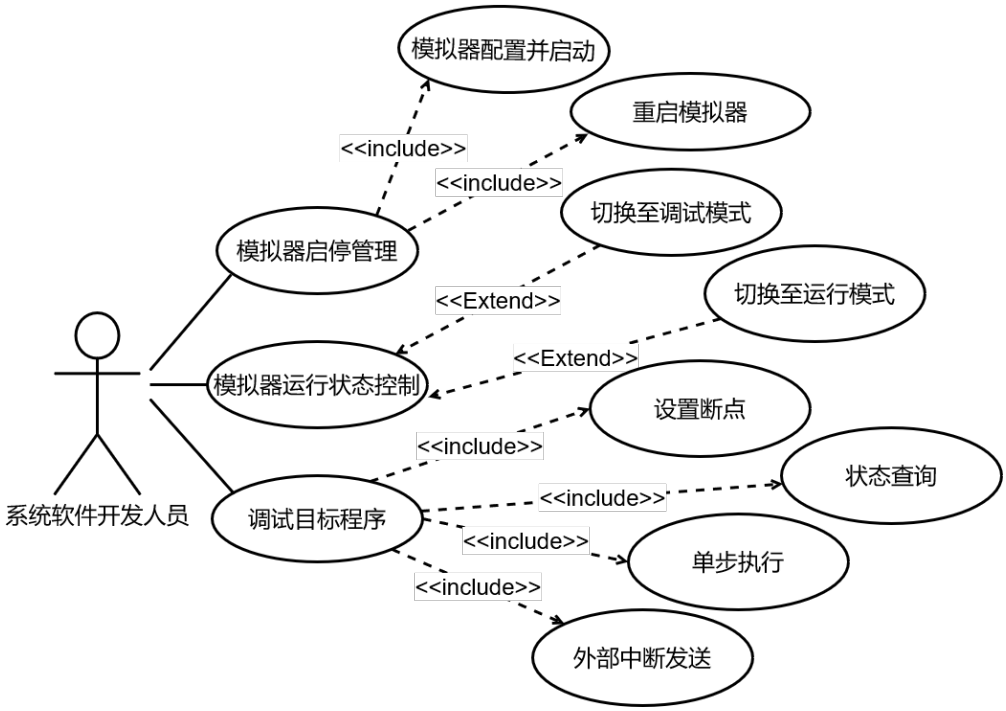

图 3.2　系统软件开发人员用例图

其中模拟器运行状态控制和调试目标程序部分是本模拟器的主体需求, 依赖于处理器核的具体模拟精度, 下面将重点分析 RISC-V 架构处理器的软硬件行为模拟需求。

3.2.1　CPU 和总线模拟需求分析

本模拟器属于指令集功能模拟器, 对于单条汇编指令涉及的软硬件行为进行模拟, 提供指令级别的仿真, 对于一些细粒度的硬件行为, 如流水线, 分支预测等不进行模拟, 因此需要对处理器核心进行抽象, 对其中的必要硬件行为进行建模。根据 RISC-V 指令集手册, 总结了处理器硬件行为分类, 包括寄存器读写、内存读写、MMU 行为、缓存行为、内存映射的 IO(Memory-Mapped I/O, MMIO) 行为、中断行为。

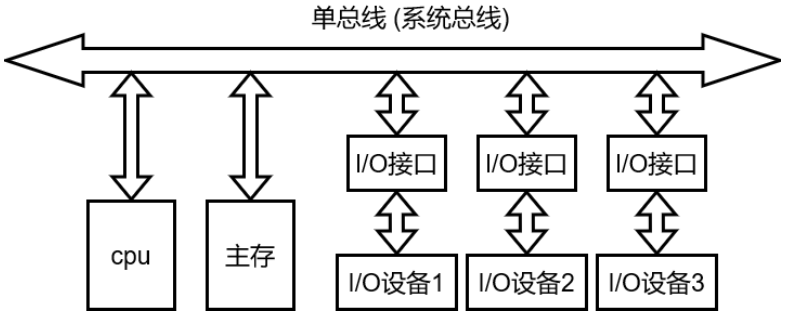

图 3.3　模拟器单总线结构图

其中 I/O 的模拟需要通过总线设备, 本模拟器模拟如图3.3所示的单总线结构, 但在实际的设计中, 内存的读写行为无需经过总线设备, 这样做可以更好的进行职责分工, 使得总线设备只需要处理 MMIO 请求。缓存涉及到性能模拟, 在软件模拟中无法做到硬件加速, 因此只使用哈希表模拟 iCache, 用来提高取指效率。MMU 的模拟需要支持 SV39 和 Mbare 模式的虚拟化方案, 参照 RISC-V 官方文档进行功能模拟。RISC-V 特权架构文档中对于中断行为做了规范, 本模拟器对于中断行为的模拟需要满足特权架构文档的要求, 具体的实现将在系统详细设计章节进行阐述。

3.2.2　中断控制器模拟需求分析

RISC-V 支持用户对中断系统的定制, 平台级中断控制器 (Platform-Level Interrupt Controller, PLIC) 作为外部中断源的核心控制部件, 是 RISC-V 架构处理器和外设进行交互的桥梁, 因此 PLIC 也是本模拟器的重点模拟对象之一, 该部分需要完成对外部中断源的管理, 包括闸口控制、中断源裁决、外部中断信号发起、优先级门限配置等。本模拟器的平台级中断控制器部分需要满足 SiFive 公司 PLIC 规范文档中的功能需求, 需要在解析设备树的时候完成对应外设的中断源配置, 注册对应的设备驱动程序。具体的需求有以下几点:

1) 根据规范文档的内存映射, 配置相应的寄存器组。

2) 实现 PLIC 的外部中断源管理逻辑, 包括中断裁决逻辑。

3) 根据规范文档对 claim/complete 寄存器的功能描述, 实现外部中断信号分发、接受、完成逻辑。

根据 SiFive 公司的 PLIC 规范, 单个中断控制器最多可以管理 1024 个外部中断源, 因此模拟器的 PLIC 实现也需要能够同时管理多个外部中断源。

3.2.3　调试模块需求分析

调试模块是本模拟器的核心业务模块, 在芯片开发过程中, 基于 FPGA 平台的系统软件调试过程如图3.4所示。可以看出, 调试的过程是跨机器的, 系统软件需要通过 vivado 对 FPGA 平台进行烧录, 该过程非常耗时, 此外硬件平台的资源有限, 需要更多地用于性能测试以及处理器验证工作, 因此系统软件的前期适配工作需要在模拟器上完成。

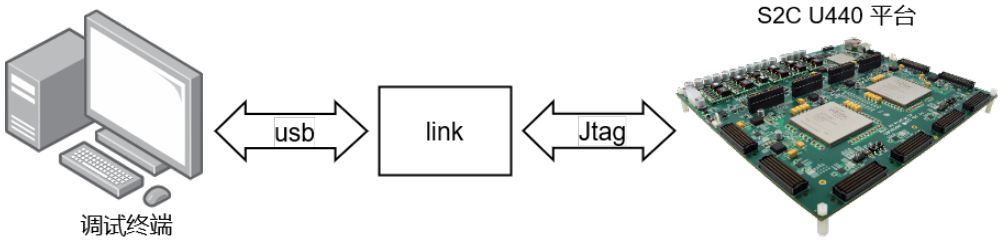

图 3.4　FPGA 平台上的调试手段

调试模块的功能需求建立在上述的硬件设备功能模拟的基础之上, 需要提供可视化的窗口供系统软件开发人员进行调试交互, 具体的需求有如下几点:

1) 提供可视化调试窗口, 状态查询窗口, 实现串口控制器模拟, 与目标程序进行交互。

2) 需要提供多种断点触发条件, 包括寄存器触发、内存触发、异常类型触发等。

3) 需要提供状态查询功能, 包括监管模式下的内存查询、对应的地址翻译过程、寄存器组状态查询、程序计数器历史查询等。

4) 需要提供多种外部中断信号的模拟, 如 mailbox、spi 控制器等, 并且能够异步地产生外部中断信号, 发送到后端模拟器对象。

5) 单步执行需要立刻更新处理器状态信息, 当匹配到断点信息时需要给出相应的匹配提示信息。

3.2.4　测试用例描述

根据用例图描述和上述各模块的需求分析, 分别对系统软件开发人员的五个主要用例进行详细描述。

模拟器配置启动的用例描述如表 3.2所示。

表 3.2　模拟器配置启动用例描述

用例名称	模拟器配置启动
用例描述	设置模拟器启动参数并运行
触发条件	勾选模拟器配置选项, 输入 ELF 文件路径
后置条件	模拟器解析配置参数, 启动程序
基本事件流	(1) 输入 ELF 文件路径 (2) 选择启动模式是否为调试模式 (3) 其他参数勾选, 包括核心数, 模拟外设路径等
异常事件流	配置参数错误, 启动失败

模拟器切换调试模式用例描述如表 3.3所示。

表 3.3　切换调试模式用例描述

用例名称	切换调试模式
用例描述	模拟器从运行模式切换为调试模式
触发条件	点击 run/stop 按键
后置条件	模拟器暂停指令流程, 进入调试模式
基本事件流	在运行模式下点击 run/stop 按键
异常事件流	运行模式下点击调试窗口按键, 窗口提示该行为无效

模拟器断点设置的用例描述如表 3.4所示。

表 3.4　断点设置用例描述

用例名称	断点设置
用例描述	调试模式下进行断点添加/移除
触发条件	在调试窗口勾选断点类型, 输入断点条件, 点击”应用”
后置条件	点击 run/stop 进入运行模式, 模拟器运行至断点条件触发调试中断, 进入调试模式
基本事件流	(1) 调试窗口添加/移除断点 (2) 程序运行, 触发断点 (3) 进入调试模式, 打印断点信息
异常事件流	断点信息填写错误导致无效断点条件

模拟器内存查询的用例描述如表 3.5所示。

表 3.5　内存查询用例描述

用例名称	内存查询
用例描述	调试模式下对虚拟地址/物理地址内容查询
触发条件	查询窗口输入内存地址, 点击查询
后置条件	输出地址对应的主存内容
基本事件流	(1) 选择地址类型为虚拟地址/物理地址 (2) 虚拟地址需要指定核心 ID (3) 输入 16 进制地址, 点击查询
异常事件流	输入无效地址导致访存失败

模拟器 UART 中断信号发送的用例描述如表 3.6所示。

表 3.6　UART 中断信号发送用例描述

用例名称	UART 中断信号发送
用例描述	在交互窗口键入, 发送 UART 外部中断到平台级中断控制器
触发条件	在交互窗口键入
后置条件	触发 UART 外部中断, 处理器读取 UART 的 FIFO 队列, 输出相应的键入信息
基本事件流	(1) 交互窗口键入 (2) 发起 UART 外部中断信号到平台级中断控制器 (3) 系统接受外部中断, 执行 UART 串口驱动程序, 显示键入信息
异常事件流	UART 外部中断优先级低于门限, 无法发起外部中断

3.3　非功能性需求

该指令集模拟器的非功能性需求有:

(1) 准确性: 体系结构模拟器的首要需求就是准确性, 只有准确模拟出真实硬件的行为, 才能在模拟器上进行后续的软件开发和移植工作。本模拟器属于指令集功能模拟器, 因此要求模拟器要和真实硬件在寄存器级别完全一致。

(2) 可靠性: 可靠性要求模拟器要能够在使用过程中持续稳定运行, 不会因为宿主机上模拟器程序的设计缺陷导致模拟过程发生崩溃。如果遇到异常情况, 模拟器需要能够在不修改启动配置的情况下重启成功, 且模拟过程是可复现的。

(3) 实时性: 作为一个指令集功能模拟器, 本身的设计初衷不是为了测试 CPU 性能, 但是模拟器运行速度至少需要达到 10MIPS, 以满足开发需求。

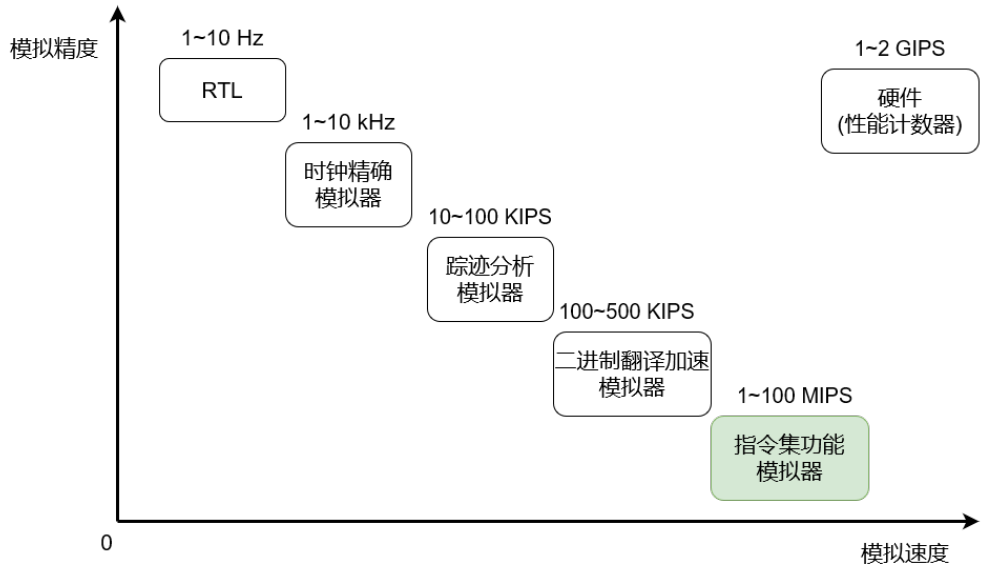

图 3.5　不同功能的模拟器运行速度对比

(4) 友好性: 模拟器需要提供一个结构清晰的可视化界面, 调试模式下需要能够及时更新处理器状态, 包括寄存器值, 历史汇编指令列表等。需要提供便捷的内存查询功能, 模拟外部中断信号发送等。

3.4　本章小结

本章主要是采用面向对象分析的方法, 在实际的芯片开发项目过程中, 对系统软件移植、开发、测试流程进行剖析, 给出了模拟器应满足的功能性需求和非功能性需求, 确定了指令集模拟器的主要业务流程, 最终形成了系统的需求规格说明书。

第 4 章　系统概要设计

4.1　系统概述

本章在需求分析的基础上, 对 RISC-V 指令集模拟器的整体架构设计进行阐述, 并对各功能模块进行分析建模, 以本章节为基础, 可以从整体上了解 RISC-V 指令集模拟器的设计思路, 为后续的具体实现提供指导。

4.2　系统静态结构

本系统是针对 RISC-V 芯片开发团队在系统软件开发和移植过程中使用的体系结构模拟器。用户将编译好的 RISC-V 架构可执行程序加载到模拟器上运行, 控制执行流程, 观察执行结果, 能够脱离实际硬件平台进行系统软件的调试。该 RISC-V 指令集模拟器的整体功能模块如图4.1所示, 主要包含四个功能模块：预加载模块,CPU 和总线模块, 中断控制器模块, 以及调试和 UI 模块。其中, 预加载模块包括模拟器参数配置, 指令集注册, 加载 ELF 文件功能；CPU 和总线模块包括了核心的处理器模拟, 内存模拟, 总线模拟等功能, 是模拟器的主体功能模块；中断控制器模块包括 PLIC 模拟和部分外设模拟; 调试模块包含了断点设置, 内存查询, 模拟中断信号发送功能, 和 UI 显示模块结合在一起, 提供包括目标程序执行窗口, 调试窗口在内的整体可视化界面。

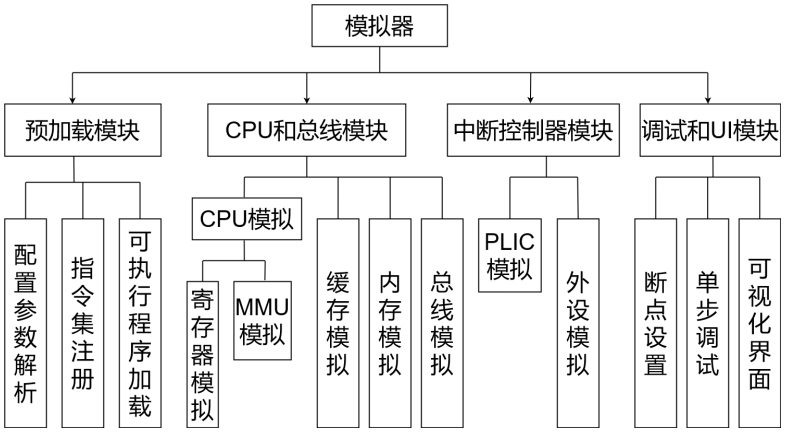

图 4.1　系统功能模块图

根据模拟器各个功能模块的特点和业务需求, 确定了 RISC-V 指令集模拟器的整体层次架构如图4.2所示, 共分为 5 层。硬件层主要是 X86 架构的宿主机环境。数据层主要包括 RISC-V 目标程序二进制文件, 和注册过相应指令集模块的译码器。在模拟器运行时还包括动态的缓存及主存模拟数据。逻辑层包含了模拟器的业务处理逻辑, 由预加载模块从数据层提取可执行代码, 通过译码器进行翻

译,CPU 和总线模块执行相应的功能函数, 最后将目标程序执行结果以及硬件状态变更同步到表现层。表现层主要由调试窗口, 查询窗口和目标程序交互窗口组成。用于展示 RISC-V 目标程序执行结果, 查询模拟器状态信息, 以及提供交互调试界面。最终在用户层向用户提供完整的 RISC-V 指令集功能模拟器。

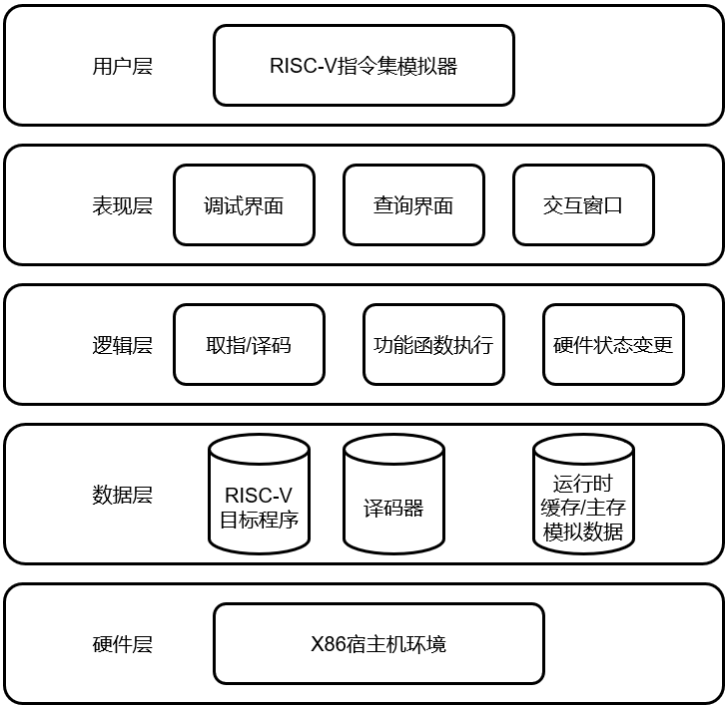

图 4.2　系统整体架构图

CPU 和总线模块是模拟器的主体功能模块, 该模块模拟了单条指令执行过程中的主要硬件行为, 包括寄存器,MMU, 高速缓存, 主存, 总线等。模拟出的 RISC-V CPU 整体逻辑架构图如图4.3所示。

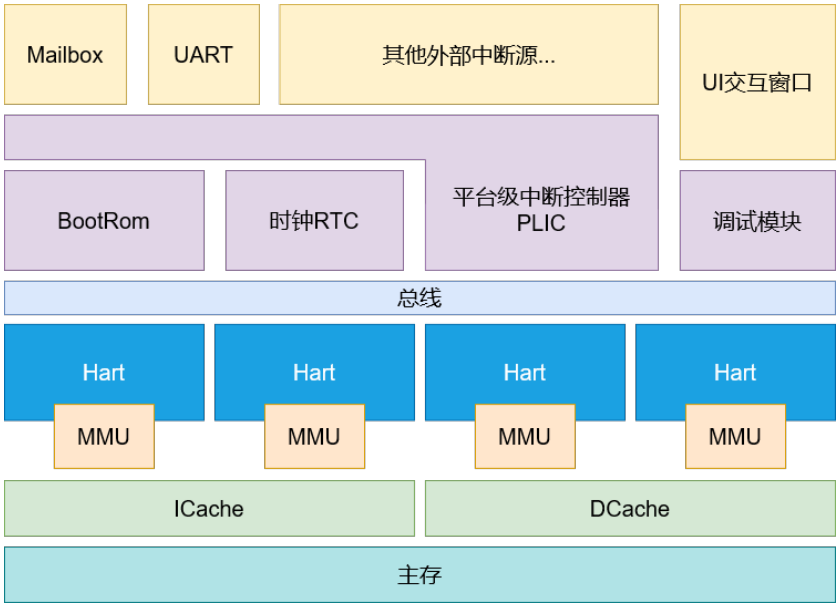

图 4.3　处理器逻辑架构图

模拟器以单个处理器 (在 RISC-V 架构中称为硬件线程 Hart) 为核心模拟对象, 每个 Hart 都有独立的寄存器组, 内存管理单元, 所有处理器共享同一个 iCache,dCache 和主存。核内的数据通信不经过总线, 直接在处理器复合类对象内部进行通信, 具体的类图见下一章。所有 MMIO 都需要经过总线, 包括时钟,Bootrom, 平台级中断控制器以及调试模块。

中断系统以平台级中断控制器 (Platform-Level Interrupt Controller, PLIC) 为核心,PLIC 对外部中断源进行优先级裁决, 对内表现为黑盒, 处理器不需要关心具体的外部中断源情况, 在接受外部中断信号时才会读取相应的外部中断 ID, 执行对应的中断处理程序。PLIC 与处理器核的关系如图 4.4所示。

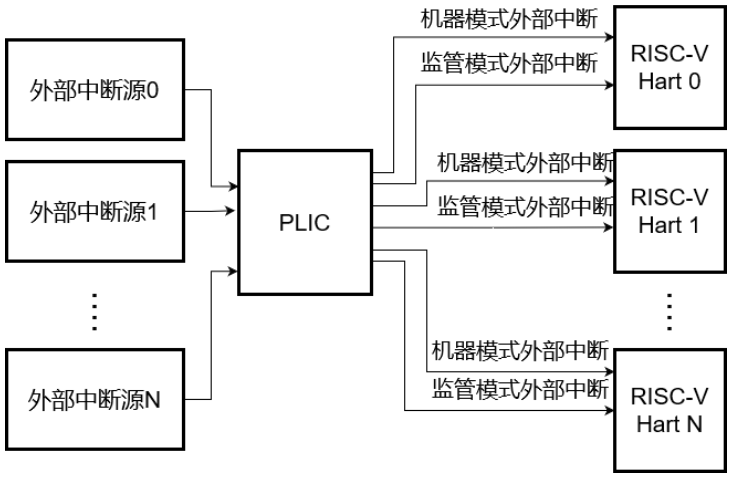

图 4.4　PLIC 逻辑架构图

在实际的硬件设计中,PLIC 与每个处理器核都有两条硬连线, 分别对应机器模式外部中断信号和监管模式外部中断信号。该模块还包含了部分外部设备的模拟, 是本模拟器的特色之一, 区别于 RISC-V 开源社区的 Spike 模拟器, 能够以 PLIC 为核心在模拟器上对设备驱动程序进行前期的适配工作, 使得团队能够在流片后以最快的速度启动系统软件。

调试和 UI 模块的设计主要包含前端窗口设计, 以及前后端对象间通信设计。前端窗口对象包含交互窗口, 调试窗口和状态查询窗口, 其中交互窗口需要模拟 UART 串口控制器, 将交互窗口的键入复制到 UART 的 FIFO 队列, 然后发起 UART 外部中断, 通过 PLIC 和总线与处理器进行通信。该部分的设计使用了 QWidget 提供的" 信号与槽" 机制, 处理对象间的通信。只需要绑定自定义的信号函数和槽函数, 就能方便地实现对象间的异步通信。详细的实现将在下一章进行阐述。

4.3　系统动态结构

本模拟器是提供指令级别仿真的功能模拟器, 模拟器单条指令运行的活动图如图 4.5所示。

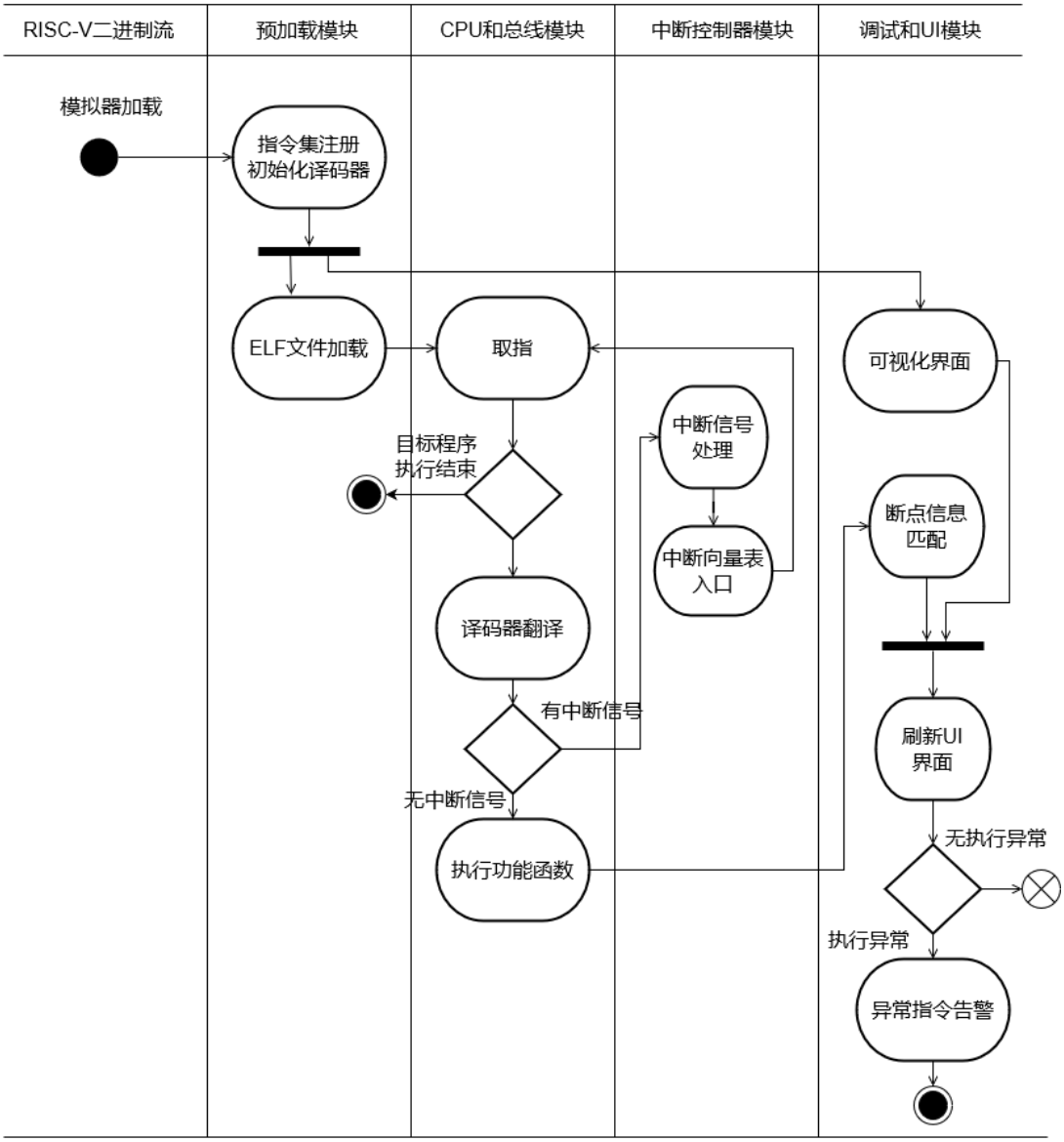

图 4.5　模拟器执行单条指令活动图

首先使用 RISC-V 交叉编译工具链将目标程序编译为 RISC-V 架构的 ELF 文件, 然后模拟器解析该二进制文件, 将对应的指令流搬运到内存起始位置, 模拟器在配置启动后为处理器注册指令集, 绑定解码器, 逐条进行译码, 执行。指令译码器完成包括操作数在内的指令信息提取, 找到该条指令注册时对应的功能函数, 执行该功能函数, 然后将更新后的寄存器状态信息, 内存状态信息同步到前端 UI 显示模块。在模拟器运行的过程中, 用户还可以通过前端交互调试窗口来切换模拟器运行模式, 设置断点触发条件, 进行单步调试, 状态查询等操作。

4.3.1　指令集注册和译码器初始化

本模拟器实现了 RISC-V 特权指令集 V1.9 版本, 和用户指令集 V2.1 版本的标准拓展指令集共 196 条指令的模拟。该部分模拟的核心是 RISC-V 汇编指令对应的功能函数, 在实际的工程项目中, 所有功能函数的实现除了参照指令集文档, 还需要参照硬件设计团队的 HDL 代码, 该项目使用了 Chisel 高级硬件设计语言进行芯片设计, 无须对冗长复杂的 Verilog 代码进行抽象, 能够较方便地进行指令功能函数的编写, 这样的做法是为了确保模拟器和真实硬件能够有一致的指令行为, 辅助进行后续的处理器验证工作, 但是需要硬件设计团队的协助, 开发过程费时费力。本论文指令集功能函数的实现相较于实际工程中使用的模拟器做了简化, 只根据 RISC-V 指令集文档进行功能函数的实现, 不涉及到具体的硬件设计。

模拟器启动后首先根据配置参数解析需要加载的指令集模块, 然后遍历各个模块的指令列表, 将指令操作码格式和对应的功能函数注册到译码器当中, 作为模拟目标机器的完备指令集, 后续的目标函数执行过程均不超过当前注册的指令集范围, 否则将会产生非法指令的异常导致模拟目标程序崩溃。

4.3.2　指令流程控制

在 CPU 运行过程中, 存在两种指令流程, 一种是常规的逻辑控制流, 包括顺序的指令流和分支跳转; 另一种称为异常控制流, 用来响应处理器状态的某些变化, 表现为中断或异常。CPU 指令控制流程的模拟包括了响应中断的逻辑,CPU 在取指之前检查当前是否有中断信号, 根据控制与状态寄存器判断是否响应中断, 进入异常控制流逻辑。本节只讨论逻辑控制流的设计, 异常控制流程设计将在下一节的中断控制中详细说明。

逻辑控制流的指令周期包括取指、译码、执行三个步骤。对于单条指令, 在逻辑上这三个步骤是顺序的, 同步的。所以对于功能模拟器, 仍然可以把实际的流水线设计看作是单周期的 CPU。处理器核的功能模拟主要包括寄存器和内存的读写、存储管理单元 MMU 行为、缓存行为和 MMIO 行为, 其中寄存器,MMU 和缓存部分都作为处理器核对象的私有成员, 模拟器对象管理公有的内存部分, 译码器, 以及总线和外设。

MMU 的功能模拟主要体现在处理器各个运行状态下的地址翻译功能。在真实的芯片设计中, 缓存和 MMU 是两个独立的硬件模块, 但是在功能模拟器中, 为了实现的方便, 可以将高速缓存模块放到 MMU 功能模块中, 在逻辑上仍然属于两个独立的功能模块, 这样做对于处理器行为的模拟没有影响。由于模拟器对于缓存的模拟只能记录缓存的命中率等信息, 并不能够真正起到硬件加速的效果, 此外本次设计的模拟器作为功能模拟器并不需要对处理器性能指标进行模拟, 所以直接舍弃 L2Cache 的模拟, 只在 MMU 模块中设置 iCache 和 dCache, 并

且使用哈希的方式进行优化。CPU 访存过程的 MMU 行为以及处理器存储层次如图 4.6所示。具体的 MMU 和缓存逻辑实现将在下一章详细实际中进行阐述。

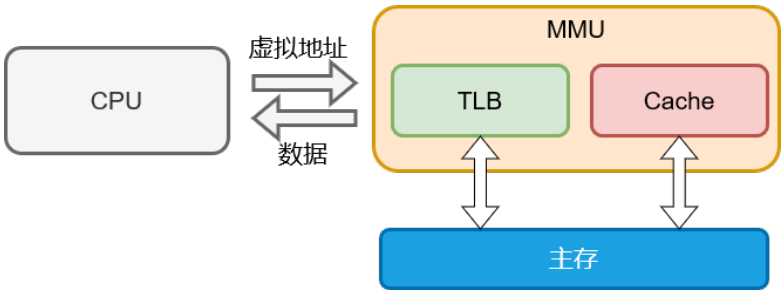

图 4.6　模拟器访存过程和存储器层次

寄存器组的模拟需要参照汇编指令具体功能进行设计。单从数据层面上讲，寄存器组只是处理器内部的一组可随机存取的数据单元，实现上使用数组就可以模拟，但是对寄存器的操作是和汇编指令功能密切相关的，RISC-V 指令级架构的设计充分发挥了后发优势，将寄存器 ID 映射到指令操作码的固定位置，从硬件设计上来讲能够极大地简化电路设计，从软件模拟的角度讲，也更加方便于接口设计，使得后续的指令集功能函数实现能够逻辑清晰，代码也更加简洁。

对于总线设备的模拟，可以忽略实际硬件设计中需要经过总线的主存访问行为，这样做可以更好的进行职责分工，降低模块间的耦合程度，使得总线设备只需要处理 MMIO 请求，如图4.7。

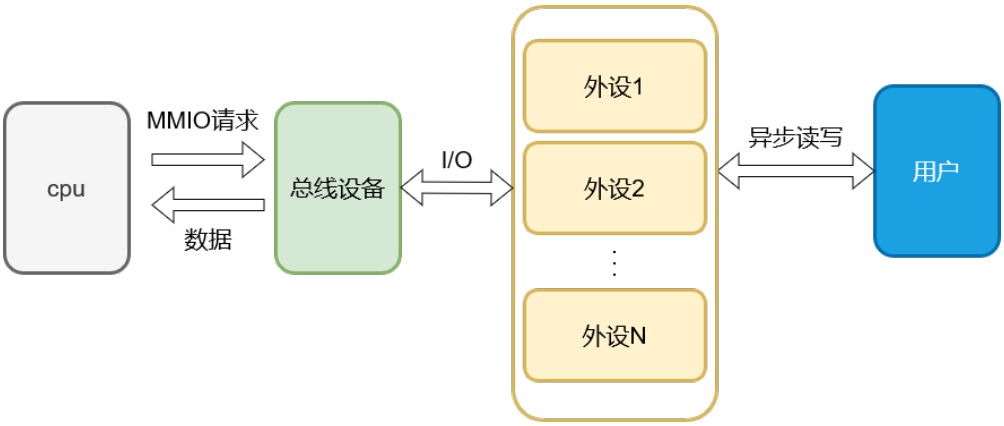

图 4.7　总线设备 MMIO 逻辑

设计上可以将总线设备抽象成统一的通信中转信箱，由全局的模拟器对象维护，忽略实际总线的电气特性细节，对处理器核表现为统一的外设控制器，处理包括中断控制器,RTC,Bootrom 固件等以内存映射方式寻址的外部设备的通信。

4.3.3　中断控制器

上一小节介绍了 CPU 指令流程控制的设计, 主要是取值、译码、执行的逻辑控制流程, 而异常控制流, 也就是中断逻辑, 需要对中断控制器进行模拟, 下面将介绍中断控制器模块的整体设计。本模拟器参照了 SiFive 公司的 PLIC 规范文档进行设计, 通过设备树挂载 PLIC 设备, 通过对内存映射的寄存器进行读写来配置 PLIC。图 4.8是 SiFive 公司给出的 PLIC 规范框架图。

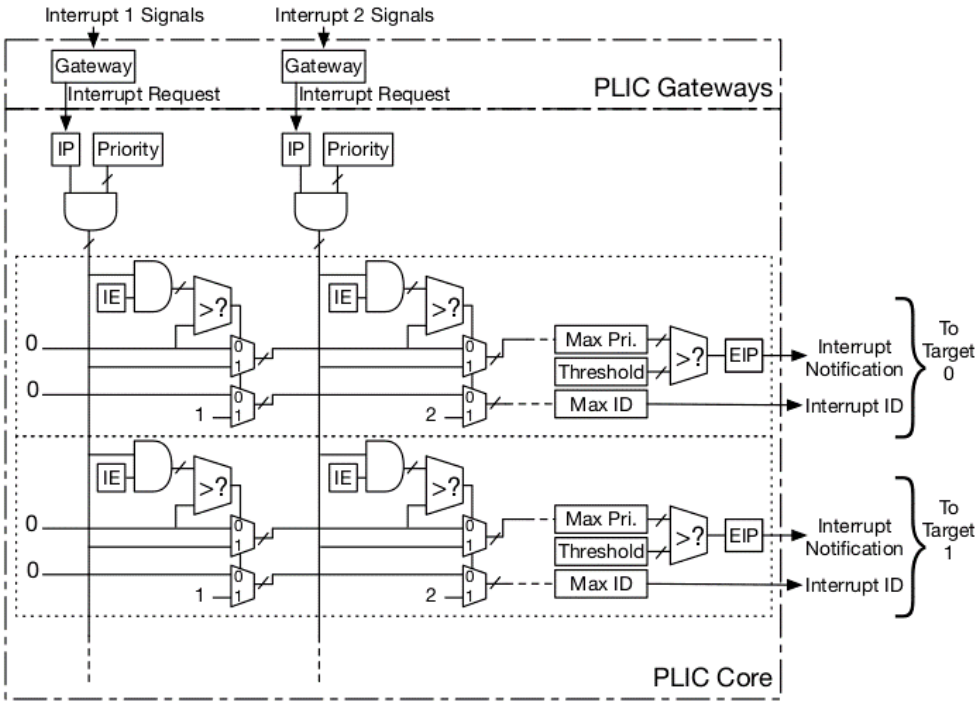

图 4.8　SiFive 公司的 PLIC 框架

中断源 (Interrupt Sources) 是挂载在 PLIC 上的设备的统称, 一般是 I/O 设备。PLIC 理论上支持任意多个中断源, 根据 SiFive 公司给出的 PLIC 规范, 每个中断源分配有如下的信息:

1) 闸口 (Gateway) 和 IP(Interrupt Pending)。闸口控制中断源发送中断请求。IP 标识当前中断源是否有请求。IP 拉高时, 闸口不再接受该中断源的请求。

2) 设备编号 ID。PLIC 为每个中断源分配的唯一编号 ID。该 ID 也作为在多个中断源具有相同优先级时的选择条件, 数值较小优先级更高。

3) 优先级 (Priority)。每个中断源都有一个内存映射的 Priority 寄存器, 表示该中断源的优先级, 软件可以通过配置 (设备树) 将其设置成不同的优先级。优先级的数字越大, 表示优先级越高。

4) 中断使能 (Interrupt Enabled, IE)。每个中断源均分配了一个内存映射的中断使能寄存器, 和优先级一样也可以通过设备树进行配置。IE 寄存器配置为 0 表示该中断源被中断目标屏蔽, 反之则是使能。

中断目标 (Interrupt Targets) 对应为 RISC-V 处理器核心的各个特权级模式。PLIC 产生的外部中断请求会分别标示在处理器的 mip 寄存器的 meip/seip 位, 对应于机器模式和监管模式。每个中断目标都有对应的内存映射的优先级门限 (Priority Threshold) 寄存器, 只有中断源优先级高于该门限,PLIC 才会将中断源 ID 发送给对应中断目标。

PLIC Core 负责所有中断请求的仲裁和分发。图 4.9展示了 PLIC 处理外部中断的流程。

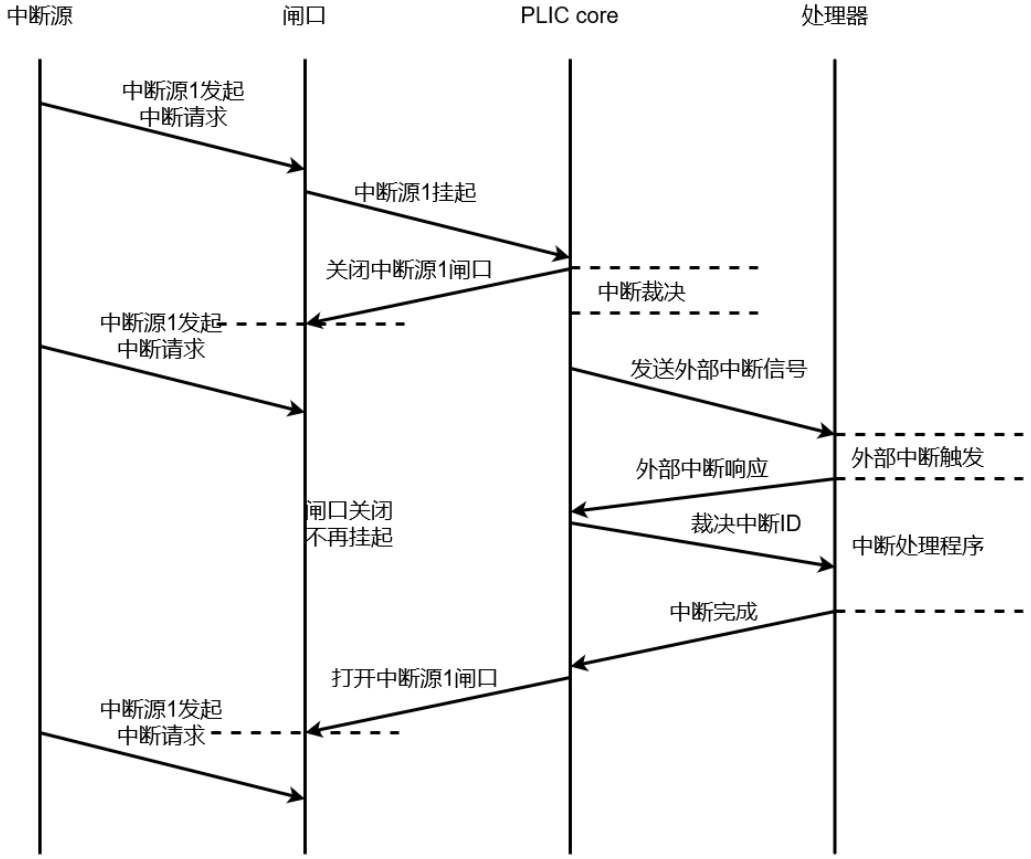

图 4.9　PLIC 中断处理序列图

首先, 对于中断模块的被动响应方, 处理器核对于中断信号的响应需要在单个指令执行周期内完成, 所以模拟器在单条汇编指令的取值动作之前将对处理器核的状态寄存器进行检查, 判断当前是否有待处理的中断信号, 然后完成相应的硬件动作; 其次, 对于外部中断的主动发起方,PLIC 需要进行多个外部中断源的优先级裁决, 控制中断源的使能情况, 并对所有处理器核的优先级门限进行筛查, 对符合条件的所有中断目标发起外部中断请求。PLIC 连接到模拟器的所有处理器核心, 对内表现为黑盒, 处理器不关心中断控制器的内部逻辑, 只针对相应的外部中断信号, 以及最高优先级中断源 ID, 进行对应的中断响应。本模拟器按照 SiFive 公司给出的 PLIC 规范文档进行设计, 具体的实现将在下一章介绍。该硬件模块

对处理器核表现为黑盒, 只提供固定功能的接口, 整体作为外部设备通过总线与处理器进行通信。

4.3.4　交互调试模块

本模拟器的主要功能就是进行目标程序的调试工作, 包括 Bootloader,Linux 内核, 驱动程序等。因此本模拟器的设计不仅需要提供丰富的调试手段, 还要能够提供友好的可视化界面, 方便进行目标程序调试, 缩短系统软件的开发, 测试, 迭代周期。调试模块的功能集成在上述的处理器指令流程控制模块以及前端 UI 模块之中。其中,UI 模块的可视化界面提供调试信息的设置, 查询功能, 后端模拟器指令执行流程中将对设置的调试信息进行匹配, 将断点匹配结果, 调试查询结果反馈到前端 UI 模块, 与开发人员进行调试交互。该模块需要提供断点设置, 单步执行, 寄存器/内存查询等功能。UI 显示界面主要包含了断点设置窗口, 查询窗口和执行交互窗口。模拟器前后端整体的工作流程如图4.10所示。

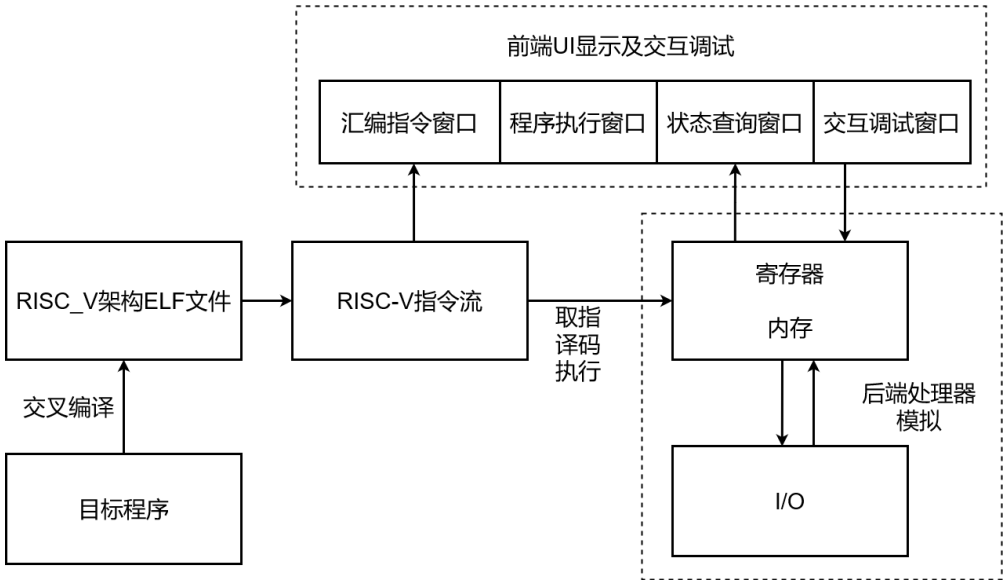

图 4.10　模拟器前后端工作流程示意图

4.4　本章小结

本章主要是根据系统的需求规格说明书, 对 RISC-V 指令集模拟器进行概要设计, 确定了本模拟器的四个主要功能模块: 预加载模块,CPU 和总线模块, 中断控制器模块, 以及调试和 UI 模块。描述了模拟器整体运行流程, 并对各模块的功能逻辑进行设计, 明确了 RISC-V 指令集模拟器的总体框架, 动态流程以及各模块设计方案。

第 5 章　系统详细设计与实现

本章节基于系统概要设计部分, 分别对模拟器四个主要功能模块的具体设计和实现细节进行阐述和说明。

5.1　预加载模块的设计与实现

预加载模块主要实现了两个功能, 指令集功能函数的翻译, 以及解码器对指令列表的解析和注册。

对应单条汇编指令的取值, 译码, 执行过程, 模拟器从 PC 地址读取一条 32 位的汇编指令, 通过解码器进行解码, 找到对应的功能函数, 然后执行该功能函数。核心的数据结构就是指令类 insn_t, 除了包含 32 位的 uint 数据成员来保存指令内容, 还定义了一系列的接口, 方便读取 RISC-V 指令格式所定义的指令码, 寄存器, 立即数等的位域信息。下面详细介绍模拟器对于指令类数据结构的定义和实现。

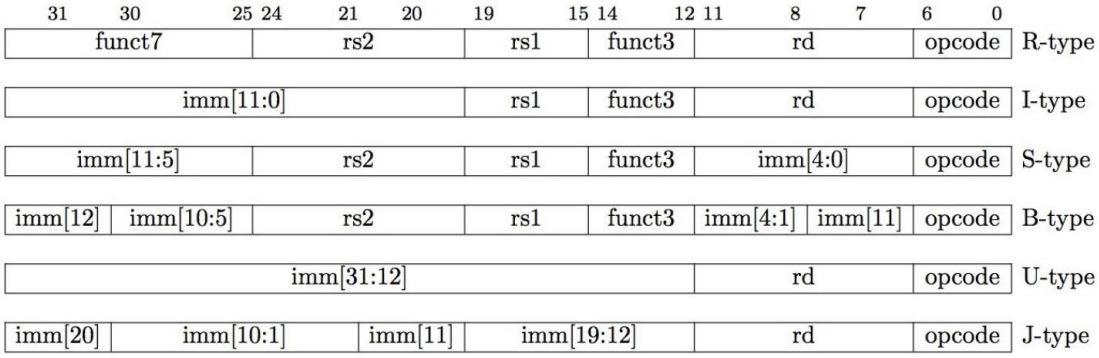

图 5.1　RISC-V32I 指令格式

由前面的章节可以知道,RISC-V 汇编指令的格式是非常明晰的, 图 5.1展示了 RISC-V32I 所包含的全部六种指令类型的格式。在实际编码过程中, 编码位置的安排都是有意义的。例如 3 个寄存器索引号在不同指令格式中的编码位置是永远不变的,Rd 在 7-11bit 位,Rs1 在 15-19bit 位,Rs2 在 20-24bit 位。即使有些指令中可能没有用到部分寄存器, 比如第二个指令类型 I-type 中没有 Rs2, 但是 Rs1 和 Rd 的索引号也在对应的位置上。又例如在 S-type 里 funct3 在 12-14bit 位, 与在 R-type 中的位置一致。操作码是所有指令格式都有的, 而且位置不变, 永远都是 0-6bit 位。得益于 RISC-V 指令类型的优秀设计理念, 很容易抽象出指令类 insn_t, 该类包含一个 32 位无符号数作为指令数据, 另外定义统一的接口来获取 RISC-V 指令中的位域信息。结合读写寄存器的宏定义, 可以方便地编写指令功能函数。

该部分的实现较为简单, 针对不同的位域信息, 通过位运算获取相应的内容, 主要为功能函数的实现提供接口。

区别于其他指令集架构的设计,RISC-V 的译码过程是比对操作码和 func 位域信息, 通过 RISCV-OPCODES 工具生成的头文件包含了所有标准指令集模块的指令格式信息, 每条汇编指令都包含一对掩码 (MASK) 和匹配 (MATCH) 信息, 译码过程是将指令内容与 MASK 取位与运算, 得到的结果和 MATCH 一致表示是该条汇编指令。在模拟器实现过程中只需要为解码器开辟一块内存空间, 存放 (MATCH,MASK,insn_t) 的三元组即可, 为了加快译码速度, 在模拟器设计中, 采用了哈希表的数据结构进行存储。综合上述接口定义, 译码器取值, 翻译, 执行逻辑可以用伪代码表示为:

```
match = insn->GET_MATCH();              //获取指令MATCH,与掩码位与
func = disassembler->GET_FUNC(match);//译码器指令翻译
func(insn);                             //功能函数执行
```

预加载模块注册指令集, 以及初始化解码器的流程如图 5.2所示。

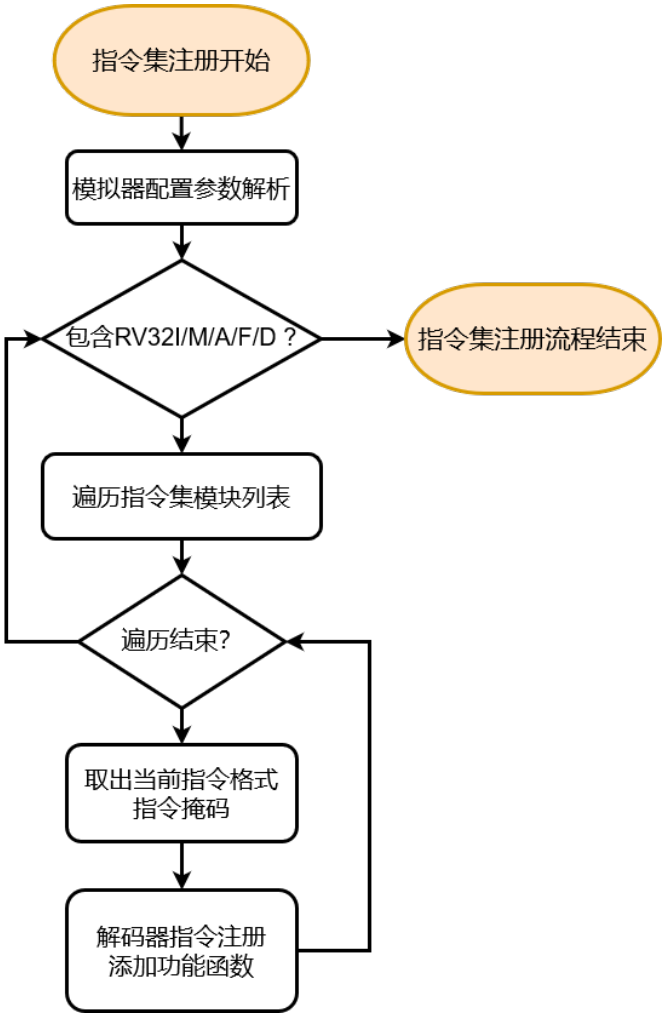

图 5.2　指令集模块注册流程图

定义了指令数据结构和解码器之后, 取指, 译码的过程就完成了, 接下来介绍

指令集功能函数的实现。理论上说, 汇编指令的功能函数需要和实际的硬件设计一一对应, 由于硬件设计所参考的指令集架构版本已经定义了各个汇编指令的功能和具体行为, 所以对于指令集的功能模拟只需要参照相应的指令集手册。指令集功能函数的实现大量依赖于寄存器操作, 下一小节将结合寄存器组的实现策略给出的部分指令的功能函数设计。

5.2　CPU 和总线模块的设计与实现

上一节介绍了指令集模块的实现, 从一个较高的抽象层次对模拟器的功能进行了定义, 本节将详细介绍硬件的模拟细节, 主要包含 CPU 内部的各个功能部件, 以及 CPU 与片外通信的桥梁－－总线的设计与实现。

5.2.1　寄存器模拟

寄存器是 CPU 内部用来存储数据的预定义单元, 汇编指令的执行过程主要就是对寄存器和其他存储器的读写过程, 因此对于寄存器的模拟需要做到精确且高效, 为功能函数的编写提供良好的接口。

RISC-V 体系结构中, 定义了两类寄存器, 整数和浮点寄存器 XPR/FPR; 控制与状态寄存器 CSR(control and status register, CSR)。寄存器组的类图如图5.3所示。

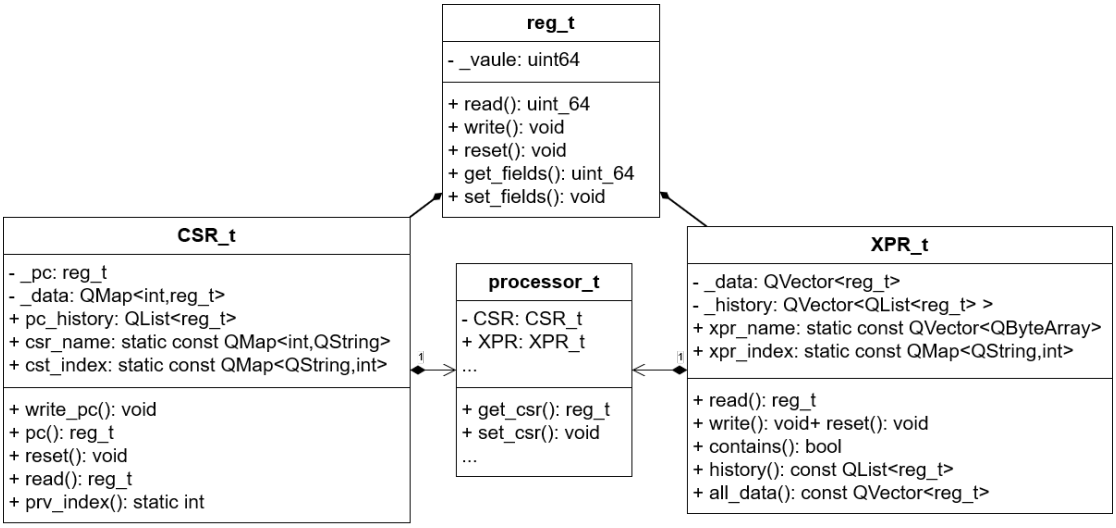

图 5.3　寄存器类图

XPR/FPR 均属于通用数据寄存器, 在用户模式和机器模式下的访问方式一致, 因此只需要将其定义为处理器类内部的公有成员, 可以直接由处理器对象获取并修改。CSR 寄存器主要由特权集指令进行位操作, 表示 CPU 状态的改变, 后续的调试模块也需要重点关注 CSR 的状态, 因此在模拟实现上需要提供统一的接口, 一方面为了简化指令集功能函数的实现, 另一方面也可以节省参数传递等

过程带来的性能损失。对于寄存器组的模拟, 除了保存当前的运行时数据, 还需要为后续的调试模块保存部分历史状态信息, 比如程序计数器 PC, 状态寄存器 MSTATUS 等。总体设计上, 寄存器组对象均封装在 processor_t 类内部, 其中通用寄存器组可以被处理器类对象直接调用, 进行读写操作;CSR 寄存器组被声明为类私有成员, 对外提供 get_csr(),set_csr() 统一接口。将 CSR 寄存器访问权限, 状态寄存器位域信息获取等操作封装在接口内部, 这样就可以使指令集功能函数的实现不必关心具体的寄存器实现细节。根据 RISC-V 特权级架构文档,CSR 寄存器接口的实现主要需要考虑三个因素:

1) 检查寄存器组所需的指令集模块支持。

2) 处理器当前权限模式检查。

3) CSR 寄存器写入格式的检查。

以状态寄存器 CSR_MSTATUS 为例。该寄存器格式如图 5.4所示。状态寄存器持续跟踪和控制硬件线程的当前操作状态。在读写状态寄存器的过程中, 需要检查 VM、MPP、MPRV、PUM、MXR 位是否有变化, 如果上述的位域发生改变, 表示处理器状态发生改变, 需要在寄存器读写之前判断当前特权等级。

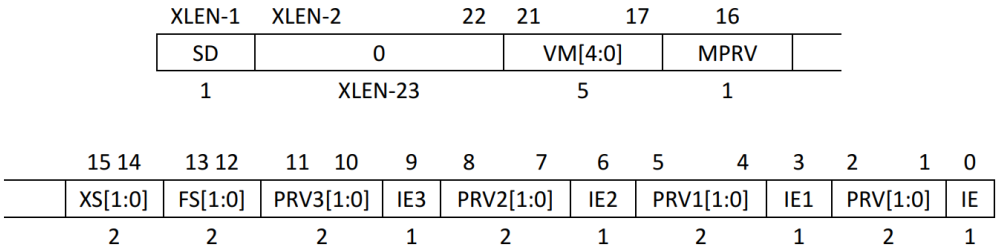

图 5.4　状态寄存器位域信息

机器模式异常返回指令 mret, 用于从机器模式异常处理程序返回, 是 RISC-V 架构中断流程的最后一条机器模式指令。该指令功能包含了上述的全部三种检查条件。下面以该指令为例, 用伪代码详细说明寄存器模拟实现和指令功能函数的具体实现方式。mret 的功能包括将 PC 还原为中断前的下一条指令 (保存在 mepc), 将特权级设置成 MPP 位保存的值, 将中断使能位 MIE 还原为 MPIE 位保存的值, 最后将 MPIE 设置为 1, 完成从机器模式到用户模式的跳转。

```
require_privilege(PRV_M);                    //宏定义，用于检查当前状态寄存器特权模式
set_pc(p->get_state().mepc);                 //通过set_pc()接口设置 PC 值
reg_t s = STATE.mstatus;
reg_t prev_prv = get_field(s, MSTATUS_MPP);
s = set_field(s, MSTATUS_UIE << prev_prv, get_field(s, MSTATUS_MPIE));
s = set_field(s, MSTATUS_MPIE, 1);
s = set_field(s, MSTATUS_MPP, PRV_U);        //寄存器类 set_fields() 接口置位
p->set_privilege(prev_prv);
p->set_csr(CSR_MSTATUS, s);               //处理器类 set_csr() 接口写 CSR 寄存器
```

在 set_csr() 接口的实现中, 需要对于 CSR 的读写进行检查, 一方面是为了确保读写时所处的特权级模式是否支持读写, 另一方面需要针对状态寄存器的改变做出相应的动作, 比如 tlb 的清除等。

除了上述的寄存器, 每个处理器对象都维护私有的 PC 程序计数器, 初始化过程中 PC 被初始化为 Bootrom 地址。

5.2.2　MMU 和缓存模拟

内存管理单元 (Memory Management Unit, MMU) 是一种负责处理 CPU 内存访问请求的计算机硬件。它的主要功能是进行虚拟地址到物理地址的转换（即虚拟内存管理）。

值	缩写	所需要的模式	描述
0	Mbare	M	没有翻译或者保护
1	Mbb	M，U	单个基址和边界
2	Mbbib	M，U	分离的指令和数据基址和边界
3-7	*保留*		
8	Sv32	M，S，U	基于页面的 32 位虚拟寻址
9	Sv39	M，S，U	基于页面的 39 位虚拟寻址
10	Sv48	M，S，U	基于页面的 48 位虚拟寻址
11	Sv57	M，S，U	保留给基于页面的 57 位虚拟寻址
12	Sv64	M，S，U	保留给基于页面的 64 位虚拟寻址
13-31	*保留*		

图 5.5　RISC-V 虚拟化方案

在 RISC-V 体系结构中, 与 MMU 有关的 CSR 寄存器主要有控制与状态寄存器 MSTATUS 和页表基址寄存器 SPTBR。MSTATUS 的 VM 位域记录了当前的虚拟化方案, 图 5.5给出了 RISC-V 架构定义的所有虚拟化方案。对于一个 RISC-V 硬件实现, 只有 Mbare 模式是强制要求的, 该模式下地址是直接映射, 理论上不需要经过 MMU 进行地址翻译,Sv39 和 Sv48 分别表示 39 位和 48 位的虚拟地址空间, 本模拟器支持上述三种虚拟化方案, 但是为了实现方便, 所有的主存访问请求都需要经过 MMU, 通过 MMU 模块的统一接口进行访存, 当 MSTATUS 寄存器 VM 位为 0 时, 无需进行地址翻译。以 Linux 内核加载过程中从物理地址向虚拟地址过渡的逻辑可以看出, 当内核支持 MMU 时, 会进入到 relocate 代码段进行页表基地址寄存器的初始化, 内核通过 setup_vm() 函数进行了首级页表的加载, 然后在 relocate 段计算了首级页表基地址, 写入 SPTBR 寄存器, 然后通过一条 mret 指令跳出机器模式, 以后首级页表会常驻内存, 接下来就全是加载虚拟地址了,MMU 开始工作。以下列出了 Linux V5.10 版本的 relocate 代码段, 由于 Linux 内核对于 RISC-V 架构的支持依据的是 RISC-V 官方给出的最新指令集版本, 所以对于芯片设计厂商的具体硬件通常不能够直接适配, 经常需要对软件进行微

调, 以适应硬件架构设计。

```
#ifdef CONFIG_MMU
relocate:
    li a1, PAGE_OFFSET
    la a2, _start
    sub a1, a1, a2
    add ra, ra, a1
    la a2, 1f
    add a2, a2, a1
    csrw CSR_TVEC, a2
    srl a2, a0, PAGE_SHIFT
    li a1, SATP_MODE
    or a2, a2, a1
    la a0, trampoline_pg_dir
    srl a0, a0, PAGE_SHIFT
    or a0, a0, a1
    sfence.vma
    csrw sptbr, a0
.align 2
1:
    /* Set trap vector to spin forever to help debug */
    la a0, .Lsecondary_park
    csrw CSR_TVEC, a0
    /* Reload the global pointer */
.option push
.option norelax
    la gp, __global_pointer$
.option pop
    csrw sptbr, a2
    sfence.vma
    ret
#endif /* CONFIG_MMU */
```

在本模拟器的实现中,MMU 模块包含了快表 TLB, 加速地址翻译, 另外, 将 iCache,dCache 的功能也一并放到 MMU 模块中, 不再实现单独的缓存硬件模块, 缓存采用直写的方式。这样的设计和真实硬件的差异很大, 会导致缓存模拟的不准确。考虑到本模拟器并不进行缓存相关的性能模拟, 所以可以忽略这部分的差别, 在功能模拟上没有影响。MMU 的类图如图5.6所示。

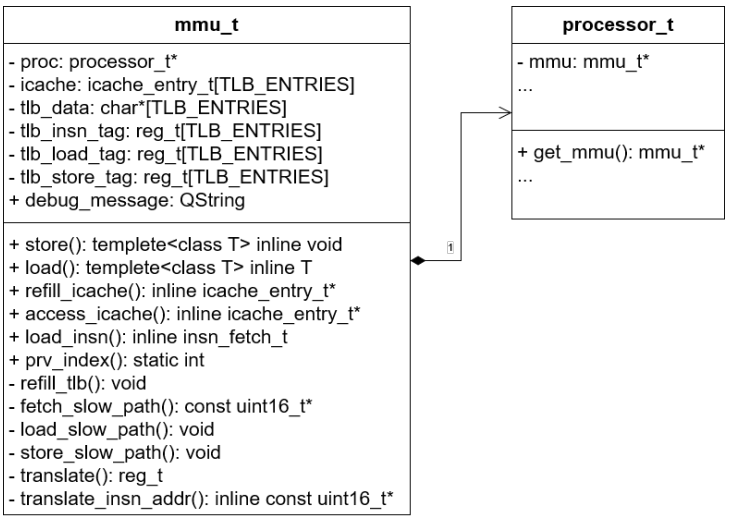

图 5.6　MMU 类图

MMU 模块对取指做特殊处理, 取指首先会查找 iCache, 当 iCache 未命中时, 退化为其余类型的访存。访存请求通过模板函数提供的统一的接口 load/store 进行请求, 通过模板可以忽略具体的数据类型, 在功能函数的实现中也可以更加方便的使用, 直接调用 MMU 的接口对存储单元进行操作。

MMU 接收到访存请求后, 首先会检查地址对齐, 然后通过页大小计算出虚拟页号,TLB 的具体实现是一个哈希表, 采用直接映射, 如果 TLB 未命中, 则进入 load_slow_path() 逻辑, 需要查找页表, 这部分的地址翻译过程封装在 MMU 类中, 在翻译之前首先确认地址范围是否是 MMIO 地址范围, 如果是, 就将访存任务转移给总线处理, 后续将详细说明 MMIO 的流程。综上, 整个 MMU 和缓存模块的功能实现都封装到 MMU 类内部, 访存请求的活动图如图 5.7所示。

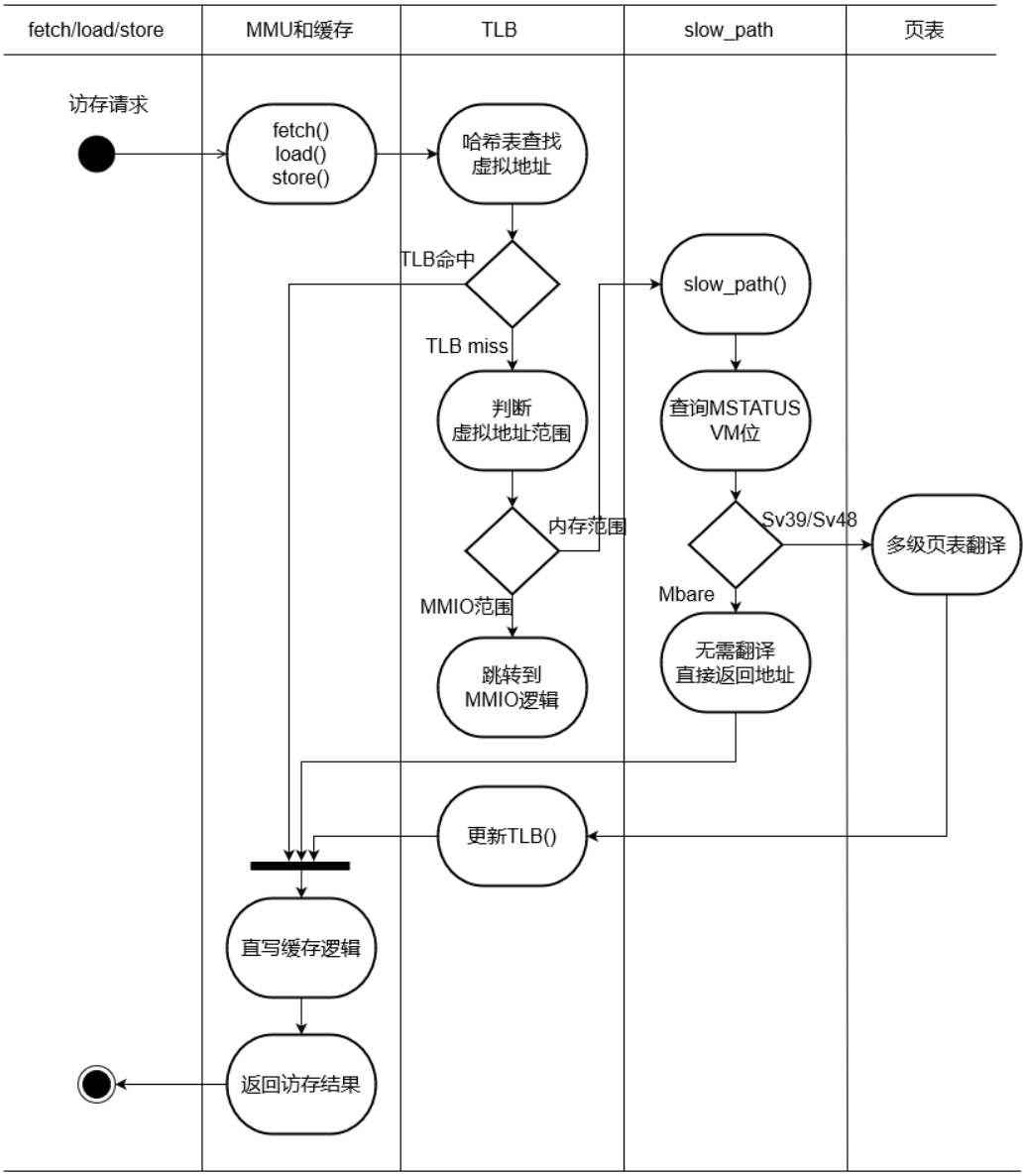

图 5.7　访存请求活动图

5.2.3　总线和 I/O 模拟

总线是 CPU 与外部设备进行数据交换的桥梁, 按照功能有不同的总线类型划分, 本模拟器对总线设备进行了功能抽象, 将总线设计为某一块物理地址区间内的 IO 控制器, 提供统一接口, 处理 CPU 和内存映射的 IO 设备之间的通信。

总线在模拟器初始化的过程中会根据配置挂载设备, 在总线类 bus_t 中维护一个物理地址和设备类对象的字典, 模拟器在接收到访存请求时, 首先检查物理地址是否是主存地址范围, 不在主存地址区间内的 I/O 请求都是 MMIO 请求, 模拟器将调用总线设备的 I/O 接口, 对内存映射的外设进行读写操作。

Base	Top	Description	
0x0000_0000	0x0000_00FF	*Reserved*	Debug (4 KiB)
0x0000_0100		Clear debug interrupt to component	
0x0000_0104		Set debug interrupt to component	
0x0000_0108		Clear halt notification from component	
0x0000_010C		Set halt notification from component	
0x0000_0110	0x0000_03FF	*Reserved*	
0x0000_0400	0x0000_07FF	Debug RAM (≤ 1 KiB)	
0x0000_0800	0x0000_0FFF	Debug ROM (≤ 2 KiB)	
0x0000_1000		Reset	Mask ROM (≤124 KiB)
0x0000_1004		*Reserved*	
0x0000_1008		*Reserved*	
0x0000_100C		Configuration string address	
0x0000_1010	0x0001_FFFF	User ROM	
0x0002_0000	0x0003_FFFF	On-chip OTP read port	OTP read (≤128 KiB)
0x0004_0000	0x00FF_FFFF	On-chip eFlash read port	eFlash read (<16 MiB)
0x0100_0000	0x01FF_FFFF	Scratchpad RAMs	Scratchpads (≤16 MiB)
0x0200_0000	0x0200_FFFF	Coreplex-Local Interrupts (CLINT) (≤64 KiB)	On-Coreplex Devices (224 MiB)
0x0201_0000	0x0BFF_FFFF	Additional Devices (<160 MiB)	
0x0C00_0000	0x0FFF_FFFF	Platform-Level Interrupt Control (PLIC) (64 MiB)	
0x1000_0000	0x1000_7FFF	Always-On (AON) (≤32 KiB)	Off-Coreplex I/O (1.75 GiB)
0x1000_8000	0x1000_FFFF	Power, Reset, Clock, Interrupts (PRCI) (≤32 KiB)	
0x1001_0000	0x1FFF_FFFF	Off-Coreplex Devices (<256 MiB)	
0x2000_0000	0x7FFF_FFFF	I/O, Flash, RAM (1.5 GiB)	
0x8000_0000	0xFFFF_FFFF	RAM	Main Memory (2 GiB)

图 5.8　SiFive 公司的内存映射方案

模拟器根据物理地址划分为主存, 和内存映射的 IO 设备, 图 5.8是 SiFive 公司提供的内存映射参考。本模拟器的总线设备需要处理的内存映射空间就是 0x00000000-0x80000000, 提供这块内存区间上的 IO 模拟。总线设备可以挂载多种通过内存映射的 IO 设备, 某些具有特殊用途的寄存器也能够挂载在总线, 比如 TIMECMP 寄存器,IPI 寄存器。

下面以实时时钟 (Real-Time Clock, RTC) 设备为例介绍总线设备类的实现方式, 以及 MMIO 请求的流程。总线设备的类图如图5.9所示。所有设备都继承自抽象基类 abstract_device_t, 在总线类 bus_t 中维护一个设备数组, 统一管理挂载到总线的 MMIO 设备, 比如图中的 RTC 设备, 此类设备的 IO 均需要通过总线对象的 MMIO 接口与处理器进行通信。可以看到外部设备类对象中会维护一些以内存映射方式寻址的寄存器, 例如图中 RTC 设备的 timer 寄存器, 通过宿主机的精

确时钟, 模拟器对象会异步地对 timer 寄存器进行写操作, 用来模拟时钟的发生, 并在条件满足时产生时钟中断, 而处理器对象对 timecmp 寄存器的读写只能通过 MMIO 请求方式进行。类似的外部中断源设备也以同样的方式连接到总线, 通过 MMIO 的方式和处理器对象通信。

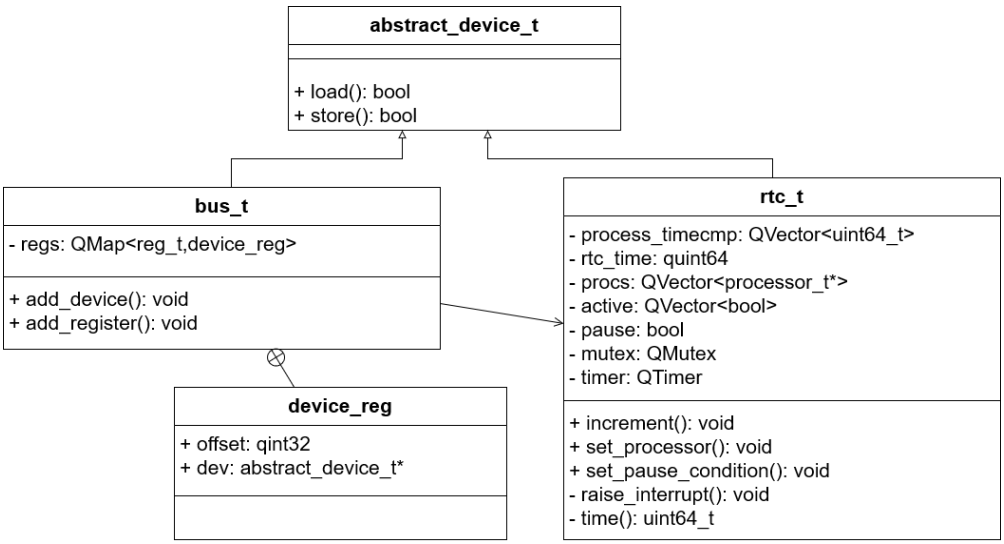

图 5.9　总线设备类图

mmio 的通信流程如图 5.10所示。

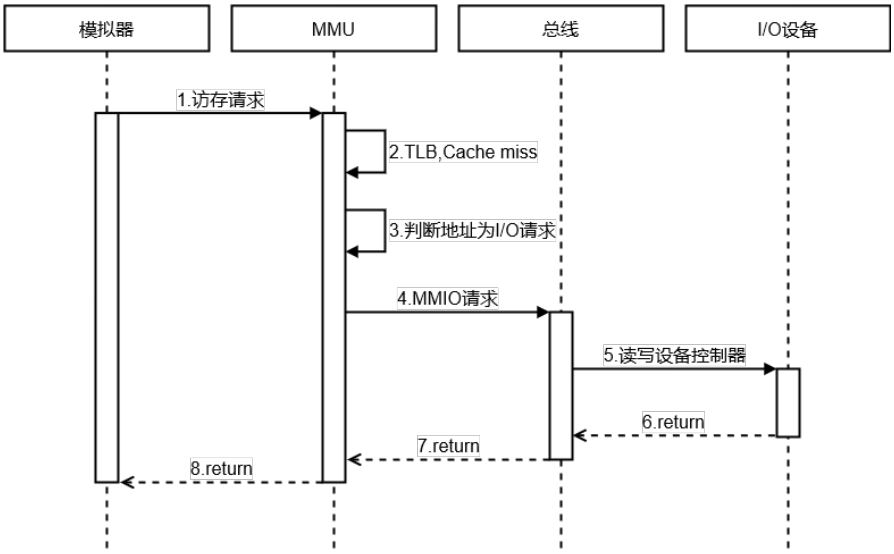

图 5.10　MMIO 请求序列图

所有支持安全启动的 CPU 都会有一个 Bootrom 固件,CPU 在通电之后执行的第一条指令就在 Bootrom 的入口, 因此 Bootrom 拥有最高的执行权限, 也就是机器模式 M-mode。它将初始化 Secure Boot 安全机制, 加载 Secure Boot Key 等密钥、从存储器加载并验证第一阶段加载程序 (First Stage Bootloader, FSBL), 最后跳转进 FSBL 中。本次芯片设计项目也包含了该部分的设计, 但是由于安全启动过程涉及到部分保密信息, 本文将不涉及 FSBL 的加载和验证过程,Bootrom 固

件程序仅仅用来跳转至目标程序入口。Bootrom 设备具体的实现方式包括以下的步骤, 首先在配置中读取 Bootrom 的地址, 默认 0x1000, 然后初始化 Bootrom 设备, 填写 Bootrom 固件内容, 总线对象使用 add_device() 接口将物理地址 0x1000 和 Bootrom 设备对象的映射保存在字典中。本模拟器的 Bootrom 程序内容如图所示, 包含两条汇编指令:"auipc t0, 0x7ffff" 和"jr t0", 模拟器启动后,PC 初始化为 Bootrom 固件地址, 即 0x1000,MMU 检测到该地址在 MMIO 区间内, 将访存请求转移给总线, 总线对象使用 MMIO 接口找到对应的设备对象 Bootrom, 读取到 Bootrom 的两条指令, 处理器在机器模式执行上述两条指令, 跳转到主存起始位置 0x80000000, 即目标程序的入口地址。接下来的一系列指令执行由目标程序决定, 主要包括 Bootloader 的 reset vector 以及加载 Linux 内核的过程。

5.3　中断控制器模块的设计与实现

之前的章节已将介绍了 CPU 指令控制流程的模拟, 其中包括了响应中断的逻辑,CPU 在取值之前检查当前是否有中断信号, 根据控制与状态寄存器判断是否响应中断, 进入异常控制流逻辑。本节将详细阐述模拟器中断系统的实现。包括平台级中断控制器 PLIC 以及部分中断源的模拟。

RISC-V 一共有两大类的中断类型：局部中断（Local Interrupts）以及全局中断（Global Inerrupts）。局部中断是指直接与硬件线程相连的中断, 可以直接通过 CSRs 当中的 xcause（mcause、scause、ucause）中的值获取中断的类型。在局部中断当中, 只有两种标准的中断类型：计时中断（timer）以及软件中断（software）。全局中断实际上就是外部中断（External Interrupts）, 与平台级中断控制器 PLIC 相连。实际上全局中断在多个硬件线程的情况下最为常用,PLIC 用于对外部中断进行仲裁, 然后再将仲裁的结果送入核内的中断控制器。

RISC-V 架构中规定了一些硬件行为来实现异常事件的响应和处理, 这些行为通过 CSR 寄存器来反应异常事件信息。涉及到如下几个寄存器, 见表 5.1。

表 5.1　中断相关的寄存器列表

寄存器	功能
mstatus	状态寄存器, 保存全局中断使能等信息。
mie	外部中断使能寄存器
mtvec	异常入口基地址寄存器
mscratch	上下文切换时用于保存当前的指针信息
mepc	异常 PC 寄存器, 现场保护时保存 PC 值
mcause	异常原因寄存器
mip	中断悬挂寄存器, 标识中断信号 pending
mbadaddr	异常原因辅助信息寄存器

本模拟器实现了这部分的硬件行为, 包括中断源的产生, 和中断响应, 下面进行详细的介绍。在处理器类内定义了两个 uint64 型数据, 分别用来表示时钟中断和外部中断的 pending, 总线上挂载的设备可以直接对这两个 pending 数据进行设置, 用来模拟外设”异步”的中断请求发起。处理器在下一个取值周期之前, 使用 try-catch 语句来处理可能的中断响应, 如果处理器决定响应中断, 就会抛出一个 trap_t 类型的异常, 进入到响应中断的异步逻辑。trap_t 的定义如下:

```
class trap_t
{
public:
    trap_t(){}
    trap_t(reg_t type):cause(type),addr_valid(false){}
    trap_t(reg_t type,quint64 addr):cause(type),addr_valid(true),addr(addr){}
    reg_t cause;
    bool addr_valid;
    quint64 addr;
    static QMap<reg_t,QString> names;
    static quint64 delegable_exceptions;
    QString &name(){return names[cause];}
};
```

抛出异常后, 处理器将根据状态寄存器以及异常信息决定是否响应中断,CPU 异常处理的流程如图 5.11所示。

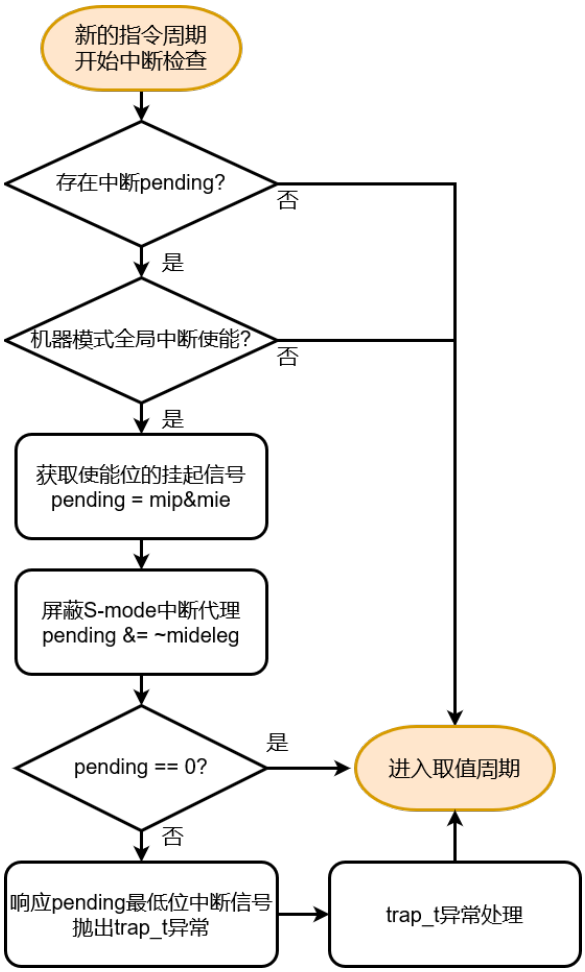

图 5.11　处理器响应中断流程图

模拟器抛出异常后, 根据捕获到的 trap_t, 进入中断处理流程。RISC-V 特权架构对这部分的处理器行为做了一定规范, 本模拟器的实现也参照了特权架构定义。

1) 根据处理的中断类型将信号源编号记录到 mcause 寄存器中;

2) 地址不对齐或者发生访问异常, 将导致错误的指令部分保存到 mbadaddr;

3) 更新状态寄存器 mstatus, 记录中断处理前的状态;

4) 保存当前 PC 到 mepc 寄存器中, 以便于处理完中断后返回;

5) 停止当前执行程序流, 设置 PC 为 mtvec 中断向量表入口地址并开始执行。

以上就是中断产生和响应过程对应的处理器硬件行为模拟, 接下来将着重阐述平台级中断控制器 PLIC 的实现。

5.3.1　PLIC 模拟

PLIC 的功能是接受外部中断源发出的中断信号, 对中断请求进行裁决, 将裁决结果和外部中断信号发送给处理器。本模拟器参照了 SiFive 公司的 PLIC 规范文档进行设计, 通过设备树挂载 PLIC 设备, 通过对内存映射的寄存器进行读写来配置 PLIC。根据上一章的 PLIC 概要设计, 将中断源和中断目标分别抽象成 plic_dev_info_t 类和 plic_core_info_t 类, 整体的 PLIC 中断控制器的设备类图如图 5.12所示。

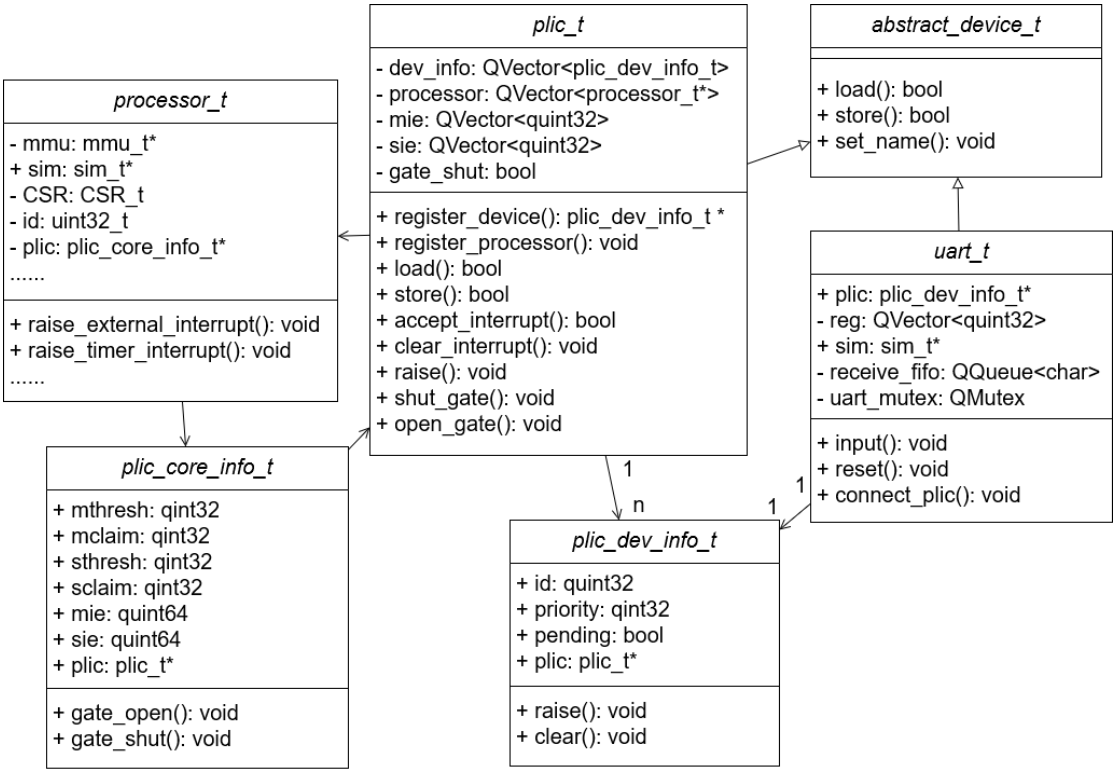

图 5.12　PLIC 设备类图

以串口控制器类 uart_t 为例, 在外设的模拟实现中, 通过 connect_plic() 接口初始化设备类中的 plic_dev_info_t 对象, 并将自身挂载到 PLIC 设备上, 后续设备的 IO 请求可以通过 plic_dev_info_t::raise() 接口发送到 PLIC 设备, 模拟了闸口的功能。PLIC 设备类型 plic_t 中定义了 accept_interrupt() 接口用来拉高对应设备的 IP 寄存器,plic_t::raise() 接口实现了中断信号裁决, 将裁决后的中断 ID 写入 mclaim 和 sclaim 寄存器, 并且向处理器发送外部中断信号, 当处理器响应中断时, 会读取这两个寄存器的值, 然后进入中断向量表, 查找对应的中断处理函数, 通常会调用定义在内核 drives 目录下的驱动程序。中断处理结束后, 处理器会给 PLIC core 发送中断完成信息, 具体的动作就是对 PLIC 中断源指定位置 claim 寄存器发送一条 store 指令, 根据 SiFive 公司给出的规范, 中断源 ID 为 n 的的 claim/complete 寄存器内存映射位置等于 (PLIC 基地址 +0x200004+0x1000*n), 因此该位置进行 store 请求时直接调用处理器对应 plic_core_info_t 对象的 gate_open() 接口来打开闸口。此时 PLIC 完成一次外部中断请求周期, 可以发起下一次的中断请求。

5.3.2　RTC 模拟

RISC-V 架构还定义了硬件线程内的局部中断, 局部中断控制器 (Core-Local Interruptor, CLINT) 是一个存储器地址映射模块,CLINT 只负责处理软件中断和时钟中断, 因为这两个中断是 RISC-V 架构中定义的, 经过 CLINT 不需要进行任何的仲裁, 直接将中断信号写入对应的寄存器内即可。软件中断只需要向 CLINT 的 MSIP0 或者 SSIP0 寄存器的最高位写 1 即可, 处理完中断后, 将其置为 0, 这样就能够清除掉软件中断的标志位。计时器中断作为 RISC-V 内核特有的中断, 其用法就是往 MTIMECMP 或者 STIMECMP 中写特定的值, 当 mtime 达到该值时产生中断, 此时继续填写特定的 tick 就可以继续产生下个中断, 反复如此, 便可产生周期性的时钟中断。想要模拟时钟中断, 需要宿主机产生时钟源, 这部分的实现使用了 Qt 类库中的 QTimer 类对象, 周期性地调用 RTC 类的 increment() 接口来模拟时钟的发生, 当到达时钟中断条件时, 通过 MMIO 接口更新 RTC 设备的 TIMECMP 寄存器, 并调用 raise_interrupt() 接口抛出异常, 拉高时钟中断 pending 位, 模拟时钟中断的发生。在下一指令周期, 处理器接收时钟中断信号, 进入中断处理流程。

5.4　调试和 UI 模块的设计与实现

调试模块能够提供断点设置, 单步执行, 寄存器/内存查询等功能。UI 显示界面主要包含了断点设置窗口, 查询窗口和执行交互窗口。本模块使用 Qt 的 UI Designer 工具进行开发, 通过拖拽摆放各种窗口控件并进行属性设置, 观察界面

整体效果, 可以方便地进行可视化界面的设计。另外 QtWidget 库还提供了信号 (Signals) 和槽 (Slots) 机制, 用于对象间通信, 只需要在对应窗口子类化 widgets 来添加自定义信号, 然后实现自定义的槽函数, 连接到该信号即可。在本模拟器的实现中, 主要的对象间通信就是前端窗口对象和后端模拟器对象间的通信, 因此只需要定义一组双向的信号和槽就能够满足通信需求。前端 UI 窗口和后端模拟器之间的信号定义如下:

```
enum sim_cmd
{
    sim_cmd_pause_sim,
    sim_cmd_run_sim,
    sim_cmd_step_sim,
    sim_cmd_reset_sim,
    sim_cmd_access_memory,
    sim_cmd_set_breaks,
    sim_cmd_key_input,
    sim_cmd_mailbox_input
};
enum window_cmd
{
    window_cmd_update_reg,
    window_cmd_update_mem,
    window_cmd_sim_output,
    window_cmd_gst_output,
    window_cmd_pause_sim,
    window_cmd_update_mailbox
};
```

其中 sim_cmd_key_input 和 sim_cmd_mailbox_input 信号分别用来模拟 UART 和 mailbox 外部中断源信号。其余信号用于断点和查询信息的交互, 以及模拟器单步运行等流程的控制。

通过 UI Designer 设计的模拟器前端整体界面如图 5.13所示。

图 5.13　模拟器前端整体界面

断点设置窗口下可以添加任意多个断点, 断点匹配类型分为如下几种:PC、寄

存器、内存、中断类型。当模拟器包含多核时需要指定相应的核心 ID 号。当模拟器处于调试模式时, 中断指令流程, 此时可以进行断点设置, 内存查询等操作, 点击运行按键, 模拟器恢复至正常运行模式, 在每一次指令执行周期完成后模拟器将对当前断点信息进行检查, 一旦匹配上任意断点, 将发送 window_cmd_sim_output 信号用于告知触发断点信息, 并接着发送 window_cmd_pause_sim 信号, 中断指令流程, 模拟器进入调试模式。每发送一次单步执行信号, 模拟器都需要将寄存器更新信号发送给 UI 前端, 刷新寄存器窗口。

调试模式下, 模拟器正常工作的界面如图 5.14所示。

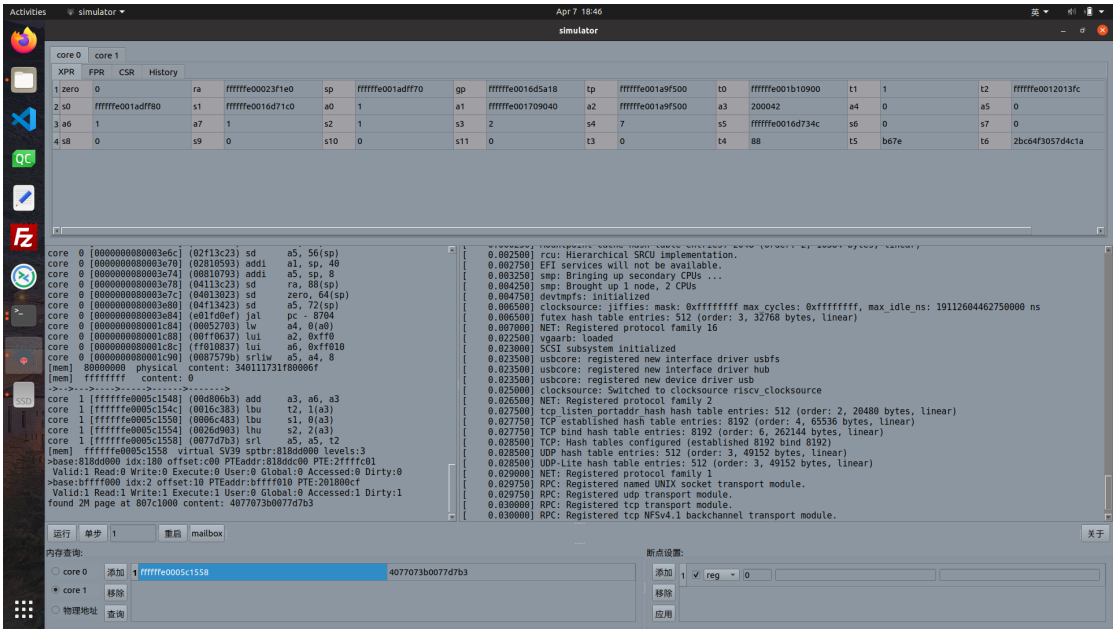

图 5.14　调试模式下的模拟器运行界面

5.5　本章小结

本章主要在概要设计的基础上对 RISC-V 指令集模拟器四个功能模块的详细设计和实现细节进行了阐述, 并根据实际工程项目中使用的设备和具体型号 IP 文档, 进行了部分外设的功能模拟, 验证了模拟器的可用性, 最后完成了前后端所有功能模块的代码实现, 并给出了相应的演示效果图。

第 6 章　系 统 测 试

6.1　测试概要

系统测试是系统开发过程中必不可少的重要阶段, 是确保系统开发质量的关键, 它贯穿于系统开发的整个过程, 尽可能地发掘系统中存在的问题和错误, 可以及时地进行纠错, 促进系统开发工作高效有质量地进行。

6.1.1　测试环境

RISC-V 指令集模拟器的测试环境分为软件模拟环境和硬件对照环境, 软件模拟环境是系统设计使用到的第三方库和模拟器的宿主机环境, 硬件对照环境主要是芯片开发过程中的 FPGA 开发板 S2C Single U440 平台以及预流片后的真实芯片环境。具体的模拟器环境如表 6.1所示。

表 6.1　系统测试环境表

名称	说明
Ubuntu20.04	模拟器的宿主机环境
Qt5	Qt5 的 UI 类库
BBL(Berkele-BootLoader)	RISC-V 官方的 BootLoader
Linux-v5.10	Linux 内核 5.10 版本目标程序
Busybox-1.32.1	Linux 命令集合

6.1.2　测试方案概述

本模拟器的测试采用测试驱动开发方式进行, 主要过程如下:

(1) 根据需求分析中确定的芯片开发团队成员需求和系统概要设计中划分的四个功能模块, 采用黑盒法设计对应的测试用例, 当各个功能模块代码实现后立刻开始测试。

(2) 根据需求分析中的需求规格说明, 使用场景法对模拟器进行配置项测试, 测试模拟器的业务功能是否满足需求。

(3) 最后进行系统测试, 重点测试 Linux 内核移植过程中的 MMU 启动, 挂载 PLIC 外设, 以及调试功能是否满足后续的系统软件移植开发需求。

6.2　测试分析

测试分析主要对模拟器的测试需求进行分析, 包含可以直接获取的显式功能性需求和系统中隐含的隐性需求比如模拟器运行速度。在系统开发的不同阶段, 测试的需求也有所不同, 本节所分析的测试需求有: 模块测试需求, 配置项测试需

求和系统测试需求。下面对主要的测试需求展开分析。

6.2.1　模块测试需求获取

根据系统概要设计, 本模拟器分为四个功能模块, 下面对各个功能模块的测试需求进行分析:

(1) 预加载模块: 该模块解析模拟器启动参数, 获取指令集配置, 对相应的指令集模块进行注册, 为后续的指令译码, 以及功能函数调用提供所需的数据, 依据概要设计的要求, 主要测试该模块能否将所需指令集全部注册进解码器, 并且在后续译码执行过程中调用正确的功能函数。

(2) CPU 和总线模块: 该模块用于模拟处理器执行过程中的硬件行为, 依据概要设计的要求, 主要测试该模块的寄存器读写过程; 访存请求涉及到的 MMU 行为, 包括快表查询,iCache 查询, 通过页表的地址翻译过程, 以及经过总线的 MMIO 请求过程。主要测试其正确性。

(3) 中断控制器模块: 该模块用于模拟中断控制器的软硬件行为, 依据概要设计的要求, 主要测试通过 PLIC 挂载设备的中断请求过程能够被处理器响应, 平台级中断控制器 PLIC 能够做到 SiFive 公司规范文档的功能要求。

(3) 调试和 UI 模块: 该模块提供前端窗口交互功能和相应的调试功能, 是本系统的重点测试部分, 依据概要设计的要求, 主要测试窗口对象和后端模拟器对象间的通信情况, 包括断点信息的设置, 处理器状态查询, 调试模式下的 UI 更新情况等, 既要测试其正确性, 也要测试调试模式下的性能是否满足规范文档的要求。

6.2.2　配置项测试需求获取

根据 RISC-V 指令集模拟器的需求规格说明书, 对于系统软件开发测试人员的需求进行配置项测试, 主要是对模拟器加载目标程序的流程测试, 以及后续目标程序的流程控制测试, 包括设置断点, 单步调试, 寄存器/内存查询, 外部中断信号发送。系统软件开发测试人员通过对目标程序的流程控制, 可以方便地进行软件移植工作, 加快系统软件的开发迭代。

6.2.3　系统测试需求获取

根据需求规格说明书, 本模拟器需要进行系统软件的移植测试。包括了 Bootload 和 Linux 内核, 在此移植过程中, 对模拟器的整体功能部件进行测试, 主要有 Linux 加载过程中的 MMU 启动,PLIC 挂载外设, 并将中断源通过设备注册到操作系统。

6.3　测试用例设计

针对系统开发的各个阶段, 测试用例的设计和采用的设计方法都有所不同, 下面对各个阶段的测试用例进行设计。

6.3.1　模块测试用例设计

依据模块测试的需求, 对 RISC-V 指令集模拟器的四个功能模块进行测试, 各个模块的测试用例设计如下:

(1) 预加载模块: 根据测试需求, 该模块的功能是根据模拟器启动参数加载对应的指令集模块, 对指令列表进行注册, 将指令格式和对应的功能函数绑定起来, 完成解码器的初始化工作, 该模块检测到未定义指令模块, 会发出告警信息, 提示未定义行为。具体的测试用例设计如表 6.2所示。

表 6.2　预加载模块测试用例表

序号	输入	预期输出	说明
1	指令集标准拓展 rv64imafdc	模拟器注册指令列表的全部内容, 解码器保存指令格式和功能函数的映射	验证功能模块的有效性
2	未定义指令集拓展 rv64jkl	提示未定义的指令集拓展	验证功能模块的健壮性
3	未定义功能函数的自定义指令	解码器提示功能函数未定义	验证功能模块的健壮性

(2) CPU 和总线模块: 根据测试需求, 该模块的功能是模拟指令执行过程中的硬件行为, 包括寄存器,MMU, 缓存, 内存,IO 控制器等, 测试目的是验证该模块的硬件行为模拟和真实硬件行为一致。该模块的输入总是已注册的指令, 不会有其他未定义输入, 主要检测是否达到预期输出, 具体的测试用例设计如表 6.3所示。

表 6.3　CPU 和总线模块测试用例表

序号	输入	预期输出	说明
4	汇编指令 ld t0 1000	通用数据寄存器 t0 内容 0x1000	验证功能模块的有效性
5	汇编指令 csrw mtevc, t0	CSR 寄存器 mtevc 被写入 t0 寄存器的值	验证功能模块的有效性
6	机器模式下的内存查询请求	MMU 显示 Mbare 模式, 不进行地址翻译	验证功能模块的有效性
7	监管模式下的 sv39 内存查询请求	MMU 查询页表, 输出地址翻译过程	验证功能模块的有效性
8	和序号 7 同一地址的监管模式下的 sv39 内存查询请求	MMU 查询快表,TLB 命中, 输出查询内容	验证功能模块的有效性

(3) 中断控制器模块: 根据测试需求, 该模块的功能是将内存映射的 I/O 设备挂载到中断控制器, 和处理器进行通信。测试目的为了验证中断控制器能够对外部中断信号源进行有效裁决, 并配合处理器完成外部中断的流程, 以及测试当外部中断源优先级低于处理器门限寄存器时能否屏蔽该中断源。具体的测试用例设计如表 6.4所示。

表 6.4　中断控制器模块测试用例表

序号	输入	预期输出	说明
9	对 UART 的 mmio 请求	前端窗口输出请求回复内容	验证功能模块的有效性
10	UART 以优先级 2 挂载到 PLIC, 处理器机器模式和监管模式的门限优先级设置为 3	处理器不再响应 UART 中断源的外部中断请求	验证功能模块的健壮性

(4) 调试模块和 UI: 根据测试需求, 该模块的功能是对目标程序运行流程进行控制, 切换至调试模式进行断点设置, 寄存器/内存查询, 单步调试等功能, 测试目的是验证该模块通过 UI 将调试信号发送给后端模拟器, 并进行断点匹配的过程, 还有验证寄存器/内存查询的正确性, 以及查询无效内存地址给出告警信息的过程。具体测试用例设计如表 6.5所示。

表 6.5　调试模块和 UI 模块测试用例表

序号	输入	预期输出	说明
11	调试模式下单步执行信号	模拟器执行完一条指令后发出更新信号, 前端 UI 刷新寄存器状态	验证功能模块的有效性
12	断点信息 pc core0 0x80000000	核 0 在 PC=0x80000000 处匹配断点, 进入调试模式, 输出断点信息	验证功能模块的有效性
13	内存地址 0x80000000 查询信号	0x1f80006f 对应汇编指令 j pc+504 表示跳转到 reset_vector	验证功能模块的有效性
14	内存地址 0xffffffff 查询信号	发出警告信息, 无效内存地址	验证功能模块的健壮性

6.3.2　配置项测试用例设计

依据配置项测试需求, 采用基于实际业务的场景设计法对 RISC-V 指令集模拟器的 RISC-V 架构目标程序调试功能进行测试用例设计。该功能设计的基本流有:1. 设置断点信息, 点击应用后, 调试窗口发送断点信号到模拟器后端, 添加新的断点信息到处理器断点检测列表; 2. 删除处理器断点检测列表中的某一项; 3. 点击单步执行, 模拟器进行一次取值, 译码, 执行周期, 更新寄存器状态窗口, 输出

当前指令到指令历史窗口; 4. 输入待查询内存地址, 点击查询, 模拟器进入 MMU 访存逻辑, 查询当前地址内容; 5. 点击运行, 模拟器进入运行模式, 将交互信息输出到交互窗口; 该功能的备选流有: 1. 设置断点信息错误, 导致处理器断点检测列表添加失败; 2. 输入无效地址导致内存查询失败, 发出告警信息。基本流和备选流可以组合成各个场景, 进而对每个场景设计测试用例, 具体的测试用例设计如表 6.6所示。

表 6.6　配置项测试用例表

序号	操作	预期输出	说明
15	设置断点信息, 点击应用	处理器断点检测列表添加成功	基本流 1, 备选流 1
16	删除某一断点信息	删除成功	基本流 2
17	点击单步执行	寄存器状态更新, 指令历史窗口更新	基本流 3
18	输入内存地址, 点击查询	对应内存地址的内容	基本流 4, 备选流 2
19	调试模式下点击运行	更新目标程序的交互信息	基本流 5

6.3.3　系统测试用例设计

依据系统测试需求, 主要对 RISC-V 指令集模拟器进行稳定性测试, 容错性测试和性能测试, 下面分别对各个测试进行测试用例设计:

(1) 稳定性测试: 依据系统测试需求, 模拟器需要在加载给定的目标程序后不间断地稳定运行。因此稳定性测试设计为模拟器加载 linux 内核运行 top 程序三天, 预期结果是模拟器可以一直持续运行, 并且实时检测模拟 CPU 的进程状态。

(2) 容错性测试: 依据系统测试需求, 从一下两个方面进行测试用例设计: 加载含有未定义指令的程序, 检测程序能否陷入对应的异常 (illegal instruction); 设置错误的断点信息, 检测模拟器在断点检测逻辑是否会发生因设计错误导致的程序崩溃。

(3) 性能测试: 依据系统测试需求, 记录模拟器在断点检测列表为空的状态下, 和含有断点检测列表的状态下的 MIPS(Million instruction per-second), 测试模拟器模拟速度。另外从模拟器进行系统软件调试的功能考虑, 记录在模拟器和在真实硬件平台上的调试周期, 进行对照测试, 分析模拟器在软件移植、开发、测试迭代过程中的性能提升。

6.4　测试结果及分析

使用各个测试阶段所设计的测试用例, 对 RISC-V 指令集模拟器展开了单元测试, 配置项测试和系统测试, 具体测试结果如下:

(1) 在单元测试中对模拟器的四个功能模块都进行了测试, 经过五次回归实验后, 各个功能模块实际输出与预期输出达成一致, 各个模块的功能和健壮性都得到了验证, 达到了需求规格说明书中的需求。在调试模式设置断点如图6.1所示：程序计数器 PC 的值为 0x80000000 时匹配断点。

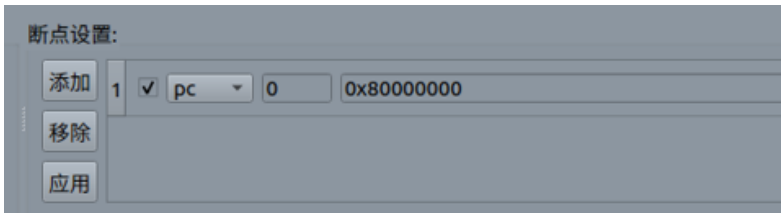

图 6.1　断点设置测试

断点信息设置完成后点击运行, 模拟器进入运行模式, 在断点位置触发调试中断, 重新进入调试模式。如图6.2：模拟器输出断点信息, 点击单步执行, 进入到 Bootloader 的 reset vector 程序段。

```
break conditions set
->-->--->---->----->------>------->
break: core 0 pc = 80000000
core   0 [0000000080000000] (1f80006f) j       pc + 504
core   0 [00000000800001f8] (00000093) li      ra, 0
core   0 [00000000800001fc] (00000113) li      sp, 0
core   0 [0000000080000200] (00000193) li      gp, 0
core   0 [0000000080000204] (00000213) li      tp, 0
```

图 6.2　断点匹配进入调试模式测试

(2) 在配置项测试中对模拟器的目标程序调试功能进行测试, 各个测试用例的实际输出与预期输出一致, 所测业务功能满足设计需求。如图6.3为 UART 设备挂载 PLIC 中断控制器的状态信息, 可以看出设备树中的 UART 配置中断源 ID 为 2, 成功挂载到 PLIC, 并且能够以外部中断的方式进行串口交互。

```
 # pwd
/
/ # ls
bin        etc        lib        mtd-utils  sbin       tmp
dev        init       linuxrc    proc       sys        usr
/ # uname -a
Linux (none) 5.10.0-00005-g62d4d9b73996-dirty #234 SMP Thu Apr 7 18:40:58 CST 2022 riscv64 GNU/Linux
/ # cat /proc/interrupts
           CPU0       CPU1
  1:       1144          0  SiFive PLIC   2  ttyS0
  5:        142         88  RISC-V INTC   5  riscv-timer
IPI0:        12         31  Rescheduling interrupts
IPI1:       609        664  Function call interrupts
IPI2:         0          0  CPU stop interrupts
IPI3:         0          0  IRQ work interrupts
/ #
```

图 6.3　UART 设备挂载 PLIC 测试

调试模式下进行内存信息查询情况如图6.4所示, 在运行模式下随机点击” 单

步” 按键, 中断模拟器运行, 进入调试模式, 查询上一条指令虚拟地址内容, 可以看到对应的 MMU 翻译过程, 以及查询结果的对照。

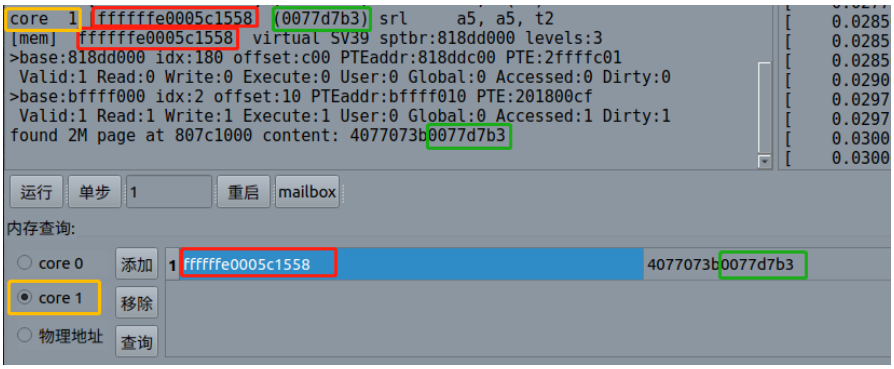

图 6.4　调试模式下内存查询测试

(3) 在系统测试中对模拟器进行稳定性测试、容错性测试和性能测试。通过对模拟器程序计数器的统计, 以宿主机时钟为基准, 测得本模拟器在无断点设置的运行模式下的运行速度为 37.9MIPS, 在不同的断点设置情况下速度略有不同, 平均值为 34.7MIPS, 达到了模拟器设计需求。稳定性测试时系统正常运行情况如图 6.5所示, 容错性测试达到了预测的结果。从测试结果可以看出 RISC-V 指令集模拟器的可靠性满足需求规格说明书中的需求。通过和真实硬件平台的对照, 在系统软件移植开发流程中, 模拟器的软件迭代周期要明显比硬件平台效率更高, 性能要求也达到了预期的结果。

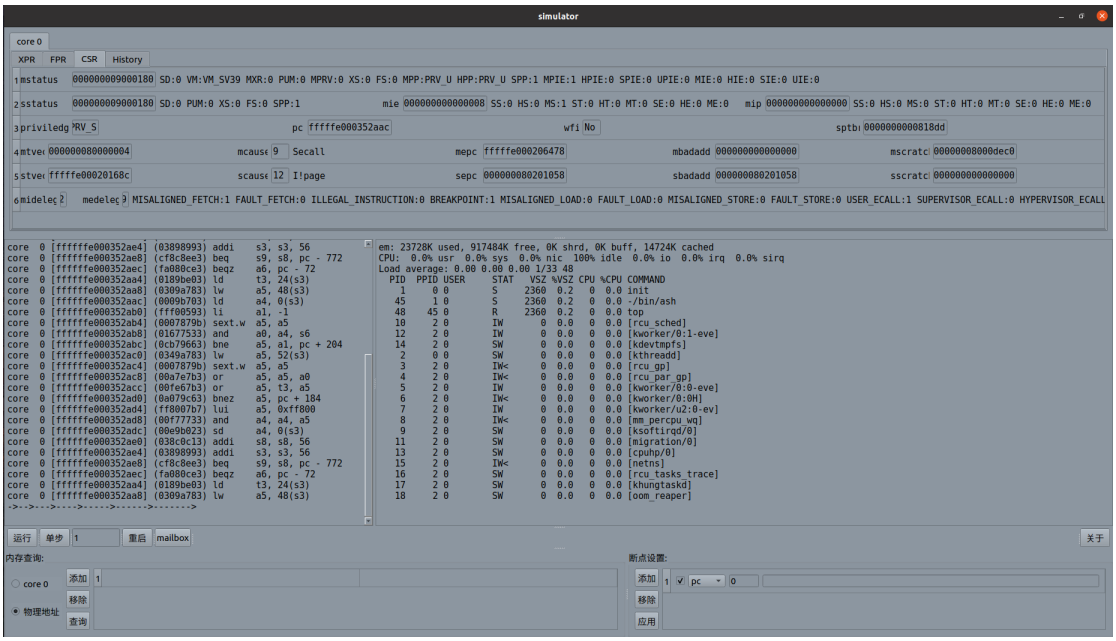

图 6.5　模拟器稳定性测试运行界面

通过比对模拟器和 FPGA 硬件平台的软件开发周期, 如表 6.7所示, 基于模拟器平台的软件迭代主要在编译和程序启动过程产生耗时, 而在硬件平台上迭代一次就需要产生 15 20 分钟的耗时,RISC-V 指令集模拟器带来的效率提升非常明显。

表 6.7　模拟器和硬件平台对照表

软件调试流程	模拟器平台	FPGA 平台	流片的 S2C 平台
目标程序交叉编译	<1min	<1min	<1min
粘贴 fsbl	-	-	5min
vivado 平台烧录	-	10min	-
flash 烧录	-	-	10min
启动程序	<1min	5min	5min
调试手段	断点/单步/查询	Jtag 单步调试	硬件波形

6.5　测试结论

经过一系列的测试,RISC-V 指令集模拟器能够长时间稳定运行, 进行目标程序的跨平台模拟执行。对于可能的目标程序异常行为, 模拟器提供了丰富的调试手段, 方便芯片开发团队进行问题排查, 针对发现的硬件缺陷或者软件错误, 及时进行修正, 进行下一轮的测试迭代。本模拟器一方面能够脱离硬件平台进行系统软件移植和开发工作, 极大地缩短了软硬件适配流程, 另一方面也可以为硅后处理器验证工作提供辅助。

但是在实际的使用过程中, 对于后续的驱动程序开发工作, 本模拟器经常需要对各种外设进行模拟, 这部分的模拟主要参照各个设备厂商 IP, 对于模拟过程可能产生的错误很难及时发现。此外在芯片开发过程中, 需要不断对模拟器进行迭代, 及时修正与真实芯片的功能差异, 这就需要软件开发人员和硬件设计人员及时有效地沟通, 很难做到职责分离, 因此本模拟器还是存在较大的改进空间。

第 7 章　结论与展望

随着 RISC-V 指令集架构的不断完善,RISC-V 软件生态的不断蓬勃发展, 未来的芯片设计与开发行业将越来越多地看到 RISC-V 的身影。本文依托 RISC-V 芯片设计与开发项目, 实现了 RISC-V 指令集模拟器, 用于 RISC-V 架构软件的移植和开发工作, 并辅助进行处理器验证。本模拟器完成了既定的功能需求, 使得芯片开发团队可以脱离硬件平台进行系统软件移植, 开发和测试工作, 并且提供了丰富的调试手段, 极大地缩短了芯片开发过程中的软硬件适配过程。主要完成的工作如下:

(1) 参照 RISC-V 用户手册, 特权级架构文档, 以及实际的硬件配置方案, 对 RISC-V 架构处理器进行功能建模, 对 RISC-V 标准拓展指令集共 196 条指令进行 C++ 功能函数模拟, 结合系统软件开发人员的需求, 确定指令集模拟器的模拟策略, 规划具体功能模块的边界, 采用面向对象的方法进行模块设计。

(2) 实现 RISC-V 指令集模拟器前后端模块, 包括预加载模块,CPU 和总线模块, 中断控制器模块, 调试和 UI 模块。设计并实现平台级中断控制器和部分外设模拟, 可以直接在模拟器平台上进行外设驱动的适配；结合实际项目需求, 实现 UI 可视化界面进行交互调试, 进一步加强模拟器的易用性。在实际的芯片开发项目中帮助团队进行系统软件开发工作, 并对处理器进行辅助验证。

本文开发的 RISC-V 指令集模拟器基本能够满足芯片开发团队的需求, 但是后续的测试以及日常的使用过程中, 发现本模拟器还是存在一些不足, 需要在未来不断地进行改进和优化, 主要包括以下两点:

(1) 本模拟器没有提供保存快照的功能, 有时调试过程中出现了异常的状态, 因为随机的因素较难复现, 所以保存快照的功能还是比较重要的。

(2) 本模拟器的一大特色就是能够通过 PLIC 对各种外设进行模拟, 并进行驱动程序的前期适配, 这就经常需要对外设进行模拟, 这部分的模拟主要参照各个设备厂商 IP, 对于模拟过程可能产生的错误很难及时发现, 对于新设备的模拟也很难在之前的设备模拟基础上进行拓展。这部分的实现有待进一步抽象提炼, 以满足易拓展的需求。

参 考 文 献

[1] 胡振波. 手把手教你设计 CPU——RISC-V 处理器篇[M]. 1 版. 北京: 人民邮电出版社, 2018.

[2] 胡振波. RISC-V 架构与嵌入式开发快速入门[M]. 1 版. 北京: 人民邮电出版社, 2019.

[3] 包云岗, 孙凝晖. 开源芯片生态技术体系构建面临的机遇与挑战[J]. 中国科学院院刊, 2022(1): 24-29.

[4] 金立忠, 窦勇. 微处理器体系结构模拟器 SimpleScalar 分析与优化[J]. 计算机应用研究, 2006(8): 197-198.

[5] 何锐. GPGPU 多核流体系结构与功耗模拟研究[D]. 国防科学技术大学, 2010.

[6] GAO X, ZHANG F X, TANG Y, et al. Simos-goodson: A goodson-processor based multi-core full-system simulator.[J]. Ruan Jian Xue Bao(Journal of Software), 2007, 18(4): 1047-1055.

[7] AUSTIN T, LARSON E, ERNST D. Simplescalar: An infrastructure for computer system modeling[J]. Computer, 2002, 35(2): 59-67.

[8] ZHANG F X, ZHANG L B, HU W W. Sim-godson: A godson processor simulator based on simplescalar[J]. CHINESE JOURNAL OF COMPUTERS-CHINESE EDITION-, 2007, 30 (1): 68.

[9] DESIKAN R, BURGER D, KECKLER S W, et al. Sim-alpha: a validated execution driven alpha 21264 simulator[R]. Technical Report TR-01-23, Department of Computer Sciences, University of …, 2001.

[10] 许鹏. 一种应用于嵌入式系统中断控制 IP 核的研究[J]. 信息技术, 2006, 30(11): 121-123.

[11] BOHRER P, PETERSON J, ELNOZAHY M. Mambo: a full system simulator for the powerpc architecture[J]. ACM SIGMETRICS Performance Evaluation Review, 2004, 4(31): 8-12.

[12] CEZE L, STRAUSS K, ALMASI G, et al. Full circle: Simulating linux clusters on linux clusters[C]//Proceedings of the Fourth LCI International Conference on Linux Clusters: The HPC Revolution 2003. 2003.

[13] KISTLER M. Experiences in simulation at ibm[Z].

[14] ASAAD S, BELLOFATTO R, BREZZO B, et al. A cycle-accurate, cycle-reproducible multi-fpga system for accelerating multi-core processor simulation[C]//Proceedings of the ACM/SIGDA international symposium on Field Programmable Gate Arrays. 2012: 153-162.

[15] CHAIX F, IOANNOU A, KOSSIFIDIS N, et al. Implementation and impact of an ultra-compact multi-fpga board for large system prototyping[C]//2019 IEEE/ACM International Workshop on Heterogeneous High-performance Reconfigurable Computing (H2RC). IEEE, 2019: 34-41.

[16] MAGNUSSON P S, CHRISTENSSON M, ESKILSON J, et al. Simics: A full system simulation platform[J]. Computer, 2002, 35(2): 50-58.

[17] BELLARD F. Qemu, a fast and portable dynamic translator.[C]//USENIX annual technical conference, FREENIX Track: volume 41. Califor-nia, USA, 2005: 10-5555.

[18] CELIO C, PATTERSON D, ASANOVIĆ K. The berkeley out-of-order machine (boom) design specification[J]. University of California, Berkeley, 2016.

[19] VARGA A. Omnet++[M]//Modeling and tools for network simulation. Springer, 2010: 35-59.

[20] 黄聪会, 陈靖, 张黎, 等. 软件移植理论与技术研究[J]. 计算机应用研究, 2012, 29(6): 2024-2027.

[21] BUTKO A, GARIBOTTI R, OST L, et al. Accuracy evaluation of gem5 simulator system [C]//7th International workshop on reconfigurable and communication-centric systems-on-chip (ReCoSoC). IEEE, 2012: 1-7.

[22] 刘畅, 武延军, 吴敬征, 等. RISC-V 指令集架构研究综述[J]. 软件学报, 2021, 32(12): 3992-4024.

[23] 王雅婕. 用于能量计量的 RISC-V 处理器设计及 MCU 实现[D]. 浙江大学, 2020.

[24] 余振波. 基于 SystemC 的粗粒度可重构通用浮点处理器设计[D]. 合肥工业大学, 2020.

[25] 邓紫珊. 基于 RISC-V 的 SoC 设计及其 RTOS 移植[D]. 电子科技大学, 2020.

[26] MUKHERJEE S S, ADVE S V, AUSTIN T, et al. Performance simulation tools[J]. Computer, 2002, 35(02): 38-39.

[27] BROOKS D, TIWARI V, MARTONOSI M. Wattch: A framework for architectural-level power analysis and optimizations[J]. ACM SIGARCH Computer Architecture News, 2000, 28(2): 83-94.

[28] 张乾龙, 侯锐, 杨思博, 等. 体系结构模拟器在处理器设计过程中的作用[J]. 计算机研究与发展, 2019, 56(12): 2702.

[29] 单磊. 大规模并行片上系统的分布式并行模拟关键技术研究[D]. 国防科学技术大学, 2012.

[30] 喻之斌, 金海, 邹南海. 计算机体系结构软件模拟技术[J]. 软件学报, 2008(4): 1051-1067.

[31] S.DWARKADAS, J.R.JUMP. Execution-driven simulation of mu1tiprocessors：address and timing analysis[J]. ACM Transactions on Modeling and Computer Simulation, 1994, 4(4): 314-338.

[32] 蔡启先, 郭森, 裴锋, 等. MIPS64 指令集模拟器的细化动态翻译技术[J]. 广西工学院学报, 2010, 21(4): 1-4.

[33] GUTIERREZ A, PUSDESRIS J, DRESLINSKI R G, et al. Sources of error in full-system simulation[C]//2014 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS). IEEE, 2014: 13-22.

[34] 李剑慧, 马湘宁, 朱传琪. 动态二进制翻译与优化技术研究[J]. 计算机研究与发展, 2007, 44(1): 161.

[35] 刘晓燕. 一种 RISC 处理器指令集模拟器的设计与实现[D]. 国防科学技术大学, 2014.

[36] ZHU J, GAJSKI D D. A retargetable, ultra-fast instruction set simulator[C]//Proceedings of the conference on Design, automation and test in Europe. 1999: 62-es.

[37] WITCHEL E, ROSENBLUM M. Embra: Fast and flexible machine simulation[C]// Proceedings of the 1996 ACM SIGMETRICS international conference on Measurement and modeling of computer systems. 1996: 68-79.

[38] CMELIK B, KEPPEL D. Shade: A fast instruction-set simulator for execution profiling[M]// Fast simulation of computer architectures. Springer, 1995: 5-46.

致　谢

三年的研究生生涯就要过去了, 首先要感谢我的导师汪增福教授给予的耐心指导和宝贵意见, 让我对论文的不足之处有了深入的理解, 并最终完成了毕业论文的写作, 还要感谢信息安全国家重点实验室的老师和师兄们在我实习期间给予的指导。最后还要感谢研究生三年认识的所有同学, 我们一起经历了一段难忘的岁月, 有汗水, 有迷茫, 也有互相的鼓励和帮助, 希望这段经历能为我们日后的工作带来力量。

路漫漫其修远兮, 吾将上下而求索。