韶关学院

毕 业 设 计

题　　目： 基于 Chisel 的精简指令集微处理器设计

学生姓名： 荀阳霖

学　　号： 19125042040

二级学院： 信息工程学院

专　　业： 物联网工程

班　　级： 20 物联网 02

指导教师： 王娜　　职　　称　　讲师

起止时间 2024 年 03 月 10 日

基于 Chisel 的精简指令集微处理器设计

摘要： 写完再摘

高效抽象的处理器芯片 硬件设计语言与工具.快速迭代的芯片设计与 系统级原型验证方法(敏捷开发)让开发硬件像开发软件那 么容易

关键词： 我；就是；测试用；关键词

Chisel implementation of a RISC microprocessor

ABSTRACT: English abstract

Key words: Dummy; Keywords; Here; It Is

目　　录

绪论

1.1 研究工作的背景与意义

世界正在加速转型进入数字经济时代。而做为数字化引擎的芯片，则是这一切的基石。虽然信息领域市场巨大，竞争强烈 但十几年来却已经形成了“赢家通吃”

我国的计算机专业人才培养面临着较大的结构问题。主要体现在顶层应用开发者过多，而下层软硬件研发人员缺乏。特别是计算机处理器芯片设计人才严重不足。在 2008 到 2017 年间，芯片架构研究优秀人才 85% 选择在美国就业，仅有 4% 在中国就业，差距巨大。这与当前芯片设计门槛过高，导致中国大学无法开展芯片相关教学与研究密切相关[1]。

而随着一种中国企业和高校被美国政府列入“实体清单”【再想想这里怎么写，参考一下 00icci 那篇，参考一下 基于 Verilog 和 chisel 的乱序超标量….的开头】 并且，从数量上来看【我忘记那个数量了，得去再看看报告】[2]

晶体管性能高速提升的时代即将结束，半导体行业即将进入后摩尔时代。提升芯片性能与能效的压力逐渐转移到了架构设计师与电路设计师上[3]

目前，由于芯片开发周期长，验证、流片成本高。【参考《问题与挑战》的引文 1 举点例子出来】

【这里不用开小标题】

【介绍 RiscV 指令集, 参考吕的文章, 参考包老师的诸文章, bytefield 可以派上用场】

处理器架构与敏捷开发

2.1 指令集架构

2.1.1 复杂指令集与精简指令集

指令集发明于 1960 年代，当时 IBM 为了解决该公司不同计算机产品线的程序无法通用的情况。 推出了完全独立于硬件架构的指令集架构这一概念。

在当时，程序员通常直接使用汇编语言进行开发。因此市场普遍认为指令集应当更加丰富，每条指令本身应该能实现更多功能。 因此，当前一些高级语言中常见的概念，如函数调用、循环、数组访问都有直接对应的机器指令。这使得早期的指令集均为复杂指令集（CISC）。

1980 年开始, 随着存储器成本大幅下降、指令数目持续增加、高级语言的普及, CISC 的一些缺陷开始暴露。 指令利用率低、控制电路趋于复杂、研发周期过长、验证难度增加等[4] 催生了对于精简指令集（RISC）的研究。

与 CISC 不同, RISC 强调每条指令的功能单一化。通过指令的组合来完成复杂操作。由于指令复杂程度下降，编译器更容易进行代码生成。同时节约的电路面积可以用于缓存或流水线等高级结构，RISC 在成本与性能上均好于 CISC

2.1.2 物联网时代与精简指令集

这段要不就不要了

2.2 RISC-V 指令集架构

2010 年，David Patterson 等人出于教学目的，在仔细评估了市面上各类指令集后，发现它们都存在设计上的不足。 决定重头开始，吸取指令集架构领域 25 年期间的经验教训。设计了一款新的指令集架构—— RISC-V。

2.2.1 RISC-V 指令集架构的优势

模块化,

2.2.2 RISV-V 指令集架构的格式

2.3 敏捷开发

传统上的软件开发通常会对整个项目制定一个规划，严格分阶段推进开发过程，称为瀑布式开发。上一阶段成功完成后，才会移至下一阶段，每个阶段只和自己的上下游对接，其他各阶段之间缺乏业务交流，这有可能导致细节疏漏、理解偏差，进而发展为项目延期，客户需求无法满足，甚至项目失败。

为了解决这个问题，一群软件工程师于 2001 年提出《敏捷开发宣言》[5]。重视小型、密集的多次迭代，以及对变化和新工具的使用 在实践上，敏捷开发通过以多周期、短增量的垂直形式完成工作。编码、测试和质量验证都必须在每一个周期进行一次。将大项目拆解成小项目，从一个简单的原型开始，不断迭代开发，直至功能完备。

2.3.1 敏捷开发的优点

传统流程的芯片开发存在 “代码冻结” 阶段，在此阶段后，除非设计出现严重问题，否则前端架构团队传递给后端物理团队的代码不能发生任何变化。这就导致了后端团队的反馈不能及时有效地帮助上游团队改进设计。

而使用敏捷思想指导开发的硬件，在完成每一个功能点的编写后，前后端团队都会一起进行测试与验证。为快速改进设计提供了可能。

基于生成器的设计促进了模块在项目之间的重用。敏捷开发可大幅降低处理器芯片的设计周期，并在市场需求变化时立即做出响应。通过敏捷开发，伯克利大学的研究团队成功在五年内开发并生产了 11 种不同的处理器芯片[3]。

2.3.2 敏捷开发的意义

与传统互联网时代相比，智能物联网时代（AIoT）的设备成本功耗受限，专用性强。传统上追求性能和通用的计算芯片不再适用。需要根据细分领域的需求，软硬件件协同优化，深度定制领域专用处理器（XPU）。以满足特定场景对于芯片的成本，性能与能效需求。[6]

这将使不同 AIoT 设备对芯片的需求差异化、专一化、碎片化。如果继续沿用传统的瀑布式开发流程，过长的芯片研发和上市时间显然无法满足海量 AIoT 设备的快速定制需要。

2.3.3 芯片行业的敏捷开发

如 图 2.1 所示。芯片行业的敏捷开发通常由高级语言写成的软件模拟器开始。软件仿真器的灵活性允许工程师快速尝试各种设计方案。方案完成后，设计被迁移到现场可编程门阵列（FPGA）上。FPGA 可以以高速和高准确性允许工程师在这一步运行完整的操作系统与测试程序。并与客户探讨设计是否满足需求。工程师可以快速在软件模拟器上根据客户反馈调整设计，并再次进入 FPGA 流程。

设计完成后，可以通过电子设计自动化（EDA）工具对芯片电路的面积、功耗和时序进行细致的分析。 出现指标不满足时，前后端团队可以及时沟通，继续优化设计。

当芯片电路设计敲定后，就可以与晶圆厂对接，准备进行投片生产。

整个设计流程的前三步都可以通过敏捷开发大幅提高效率与质量。

图 2.1 芯片开发流程

2.4 Chisel

目前，最流行的两种硬件定义语言（HDL）是 Verilog 和 VHDL[7]。硬件工程师通过 HDL 描述硬件电路的具体行为与结构。Verilog 与 VHDL 最初都是为硬件仿真而创建的，后来才被用于综合。这些语言缺乏抽象能力，使得组件难以在项目之间重复使用。

Chisel[8] 是一种基于高级编程语言 Scala[9] 的新型硬件构造语言（HCL）。由伯克利大学的研究团队于 2012 年推出。Chisel 在 Scala 中提供了各类硬件原语的抽象，使得开发者可以用 Scala 类型描述电路接口，以 Scala 函数操作电路组件。这种元编程可实现表现力强、可靠、类型安全的电路生成器，从而提高逻辑设计的效率和稳健性。

需要说明的是，虽然 Chisel 具有许多传统硬件描述语言不具备的高级特性，但其还是一门硬件构造语言，而不是高层次综合。Chisel 代码在运行后会生成对应硬件的形式化描述，以 FIRRTL（Flexible Intermediate Representation for RTL）语言记录[10]。

FIRRTL 随后被输入硬件编译框架(Hardware Compiler Framework) CIRCT[11]。经过多次递降变换（lowering transformations）[12]，逐级去除高层次的抽象，最终输出优化后的底层 RTL 代码（如 Verilog）。使用 HCF 的好处之一是其内置了大量的优化器，可进行常量传播、共用表达式合并、不可达代码消除或向 ASIC 与 FPGA 提供针对性的优化[13]。这使得工程师可以将更多精力放在逻辑功能上，进一步提升了开发效率和代码质量。

2.4.1 Chisel 的优点

开发高效

Chisel 语言的高效主要体现在四个方面：这里要么就稍微减少一点，然后把标签化的表 4 放进来

- 信号接口整体连接： Chisel 丰富的类型系统允许开发者将多条线缆“打包”并做为整体声明、连接。使用 Verilog 语言开发时，若要对总线接口的成员进行改动，开发者必须对所有出现该接口的模块逐一手动修改，不仅复杂而且容易出错。SystemVerilog 虽然提供类似的功能，但其端口定义不可嵌套，仍然无法达到良好的抽象程度。
- 元编程：Chisel 可以抽象出多份相似模块的共用部分,通过使用参数化的硬件模板，工程师可以创建出可复用的硬件库。 有效减少代码冗余，实现全局性的统一配置。SystemVerilog 虽然提供类似的功能，但仅用于验证[14]，属于不可综合代码。
- 面向对象: 通过继承, 工程师可以把 Chisel 中多个电路模块的共性抽提成一个父类。在实现具体模块时，只需通过继承的方式，就可以让模块自动拥有父类的所有特征。从而减少代码冗余，便于分层次设计。

- 函数式：Chisel 将多种电路元素包装为自函子。通过使用 map 或 zip 算子进行批量级联操作，工程师可以简洁地描述数据流。如 图 2.2 所示，级联的算子操作可以直接对应生成后的电路。与 Verilog 的数据流建模相比，工程师不再需要关心运算符优先级。

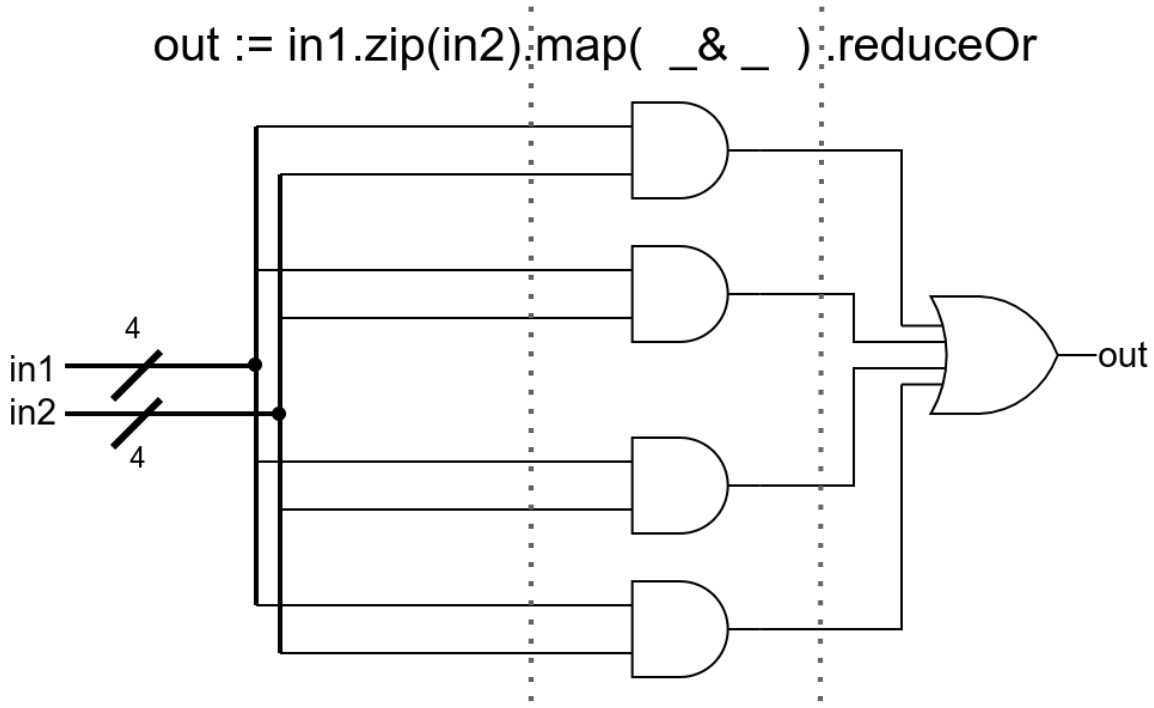

图 2.2 函数式操作很好可以很好地与电路对应

代码质量更优

余子濠.等的研究[15] 指出。 在 FPGA 上，与经验丰富的专职工程师使用 Verilog 相比，经训练的本科生使用 Chisel 的高级特性开发同样功能一致的二级缓存模块。无论是能效、频率还是资源占用，均优于 Verilog 实现。

表 2.1 资源占用与性能对比

类型	Verilog	Chisel
最高频率/MHz	135	154
功耗/W	0.77	0.74
LUT 逻辑	5676	2594
LUT RAM	1796	1492
触发器	4266	747
代码长度	618	155

在开发速度方面，专职工程师使用 Verilog，消耗至少 6 周完成开发。而另一位有 Chisel 经验的本科生则在 3 天内完成了所有工作。

而在处理器整体设计方面，同一个人分别使用两种语言开发同一微结构的 RISC-V 处理器。Verilog 的时间消耗是 Chisel 的 3 倍[16]。

处理器微架构设计与实现

本文将首先实现一个单发射，非流水线，顺序执行的基本 RISC-V 32E【格式是这样吗?】处理器微架构。 在逻辑上，为了方便将来流水线化，本处理器分为以下几个基础单元：取指单元，译码单元，运算单元，访存单元，写回单元【这里放一张图,】【可能要先介绍 Ready/Valid 接口】【必须要介绍反压控制代替集中控制。否则后面没法展开】

3.1 取指单元

取指单元的总体架构如图 3.1

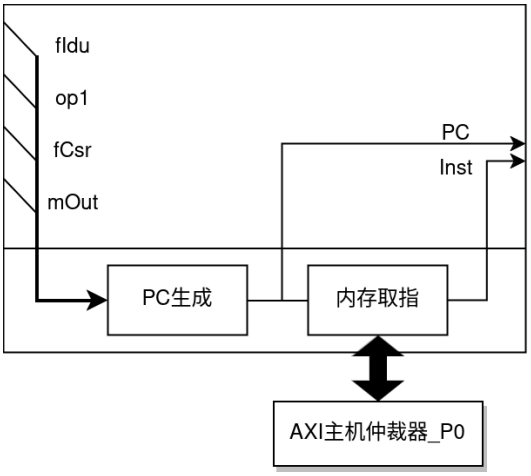

图 3.1 取指单元总体架构

其输入来自上一条指令的译码、执行结果。整个取值单元分成 PC 生成单元及指令访存单元。

3.1.1 PC(程序计数器)生成单元

PC 生成单元将根据当前处理器的状态以及指令来确定下一执行周期的程序计数器的值。 目前，程序计数器生成单元支持以下行为：

1. 顺序执行
2. 普通跳转（不切换特权级）
3. 暂停执行
4. 自陷指令——陷入
5. 自陷指令——返回

RISCV 架构支持比较两个通用寄存器的值并根据比较结果进行分支跳转（B 系列指令）。无条件跳转（J 系指令）， 也支持在异常发生时进入异常处理程序并返回。PC 寄存器的结构如图 3.2 所示,其行为如表 3.1 所示。

表 3.1 PC 生成表

类型	add1	add2	next_pc	PC 寄存器
顺序执行	pc	4	sum_next_pc	允许写入
无条件跳转	pc	立即数	sum_next_pc	允许写入
条件跳转	pc	据结果选择	sum_next_pc	允许写入
异常陷入	无关	无关	mtVec	允许写入
异常返回	无关	无关	mePC	允许写入
反压暂停	无关	无关	无关	禁止写入

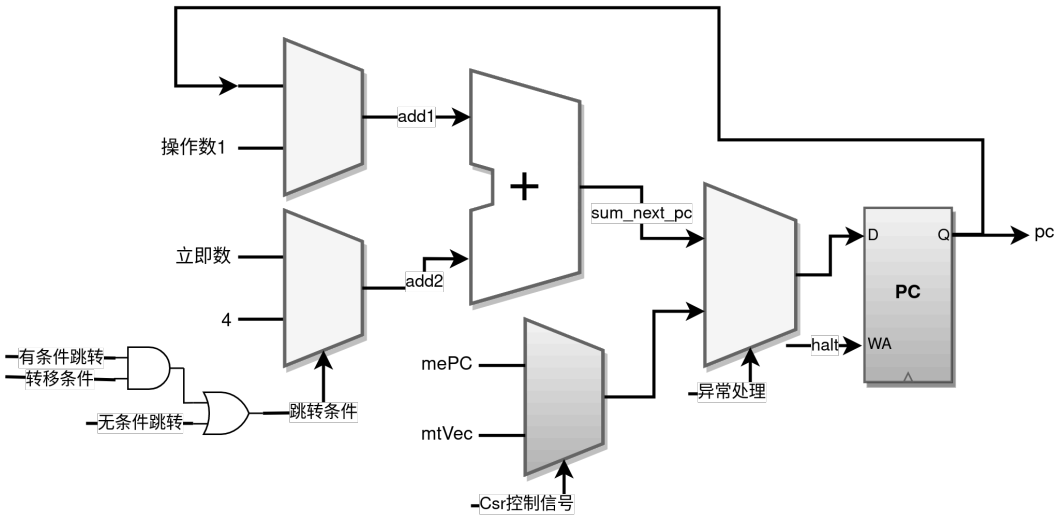

图 3.2 PC 单元结构

3.1.2 指令访存单元

指令访存单元的任务是根据当前 PC，向处理器核内的 AXI[插入引用？]主机仲裁单元发起访存请求。其结构如图 3.3 所示。在复位后，指令访存控制器【就是这个状态机】在 `idle` 状态等待有效的访存事务发生。在发生后，按照表 3.2 的行为对取值单元的输出端口进行操作。需要注意的是，该状态机的输出均来自寄存器。

由于 AXI 协议规定在读事务完成时不再保持数据有效信号。在读取完成后，指令执行中的若干周期，指令有效信号 将由状态机提供【是这样的吗。我要确认一下】

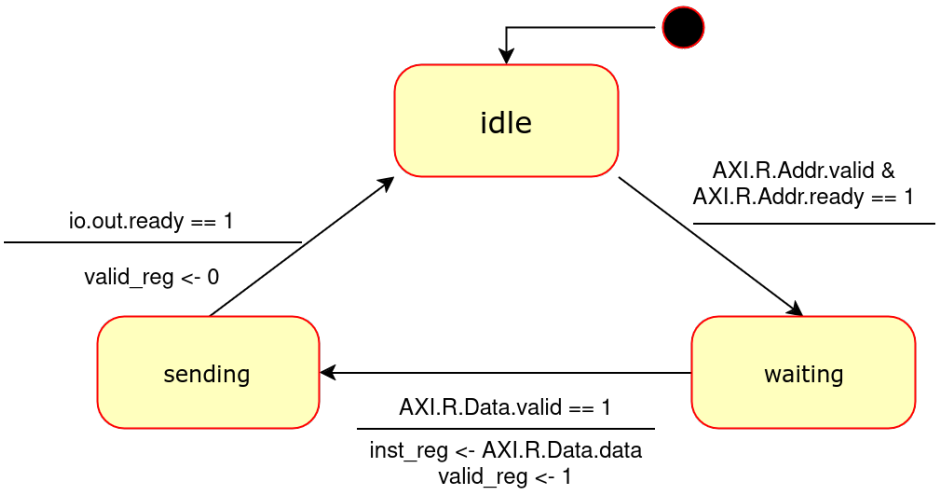

图 3.3 取指单元状态机

表 3.2 取指单元状态机转移说明

状态名	说明	离开条件
idle	当前未在访问指令存储器	存储器 AXI 读握手成功
waiting	读请求已发出。等待指令存储器回复	存储器回复数据
sending	向 CPU 下游部件提供指令码	下游部件汇报准备完成

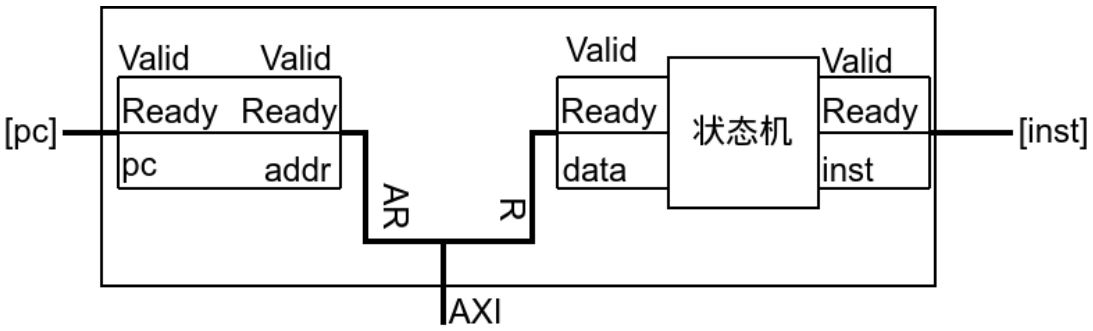

图 3.4 取指单元结构

3.2 译码单元

3.3 运算单元

[异常返回], [未定义], [未定义], [mePC] ,[允许写入], == 访存单元

3.4 写回单元

3.5 dasd

引用表 3.3,引用表 3.4，以及引用图表时，表格、图片和代码分别需要加上 `tbl:`、`fig:` 和 `lst:` 前缀才能正常显示编号。以及这里使用 `fig` 函数替代原生 `figure` 函数以支持将 `tablex` 作为表格来识别。

表 3.3 三线表 1

药品	规格
浓氨水	分析纯 AR
盐酸	分析纯 AR
钛酸四丁酯	≥99.0%

表 3.4 三线表 - 着色

t	1	2	3
y	3	4	9
3	3	17	0

3.6 插图

插图必须精心制作，线条均匀，图面整洁。插图位于正文中引用该插图字段的后面。每幅插图应有图序和图题，图序和图题应放在图位下方居中处

3.7 数学公式

可以像 Markdown 一样写行内公式 $x + y$，以及带编号的行间公式：

$$\phi := \frac{1+\sqrt{5}}{2} \tag{3.1}$$

引用数学公式需要加上 `eqt:` 前缀，则由公式 (3.1)，我们有：

$$F_n = \left\lfloor \frac{1}{\sqrt{5}} \phi^n \right\rfloor \tag{3.2}$$

图表和公式后的段落要用 `#indent` 手动缩进。同时，我们也可以通过 `<->` 标签来标识该行间公式不需要编号

$$y = \int_1^2 x^2 \,\mathrm{d}x$$

而后续数学公式仍然能正常编号。

$$F_n = \left\lfloor \frac{1}{\sqrt{5}} \phi^n \right\rfloor \tag{3.3}$$

3.8 参考文献

可以像这样引用参考文献：[17]

3.9 代码块

```
def add(x, y):
  return x + y
```

正文

4.1 正文子标题

4.1.1 正文子子标题

正文内容

参考文献

[1] 包云岗, 孙凝晖, 张科. 处理器芯片开源设计与敏捷开发方法思考与实践[J]. 中国计算机学会通讯, 2019, 15(10): 42-48.

[2] 雨前顾问, 安谋科技. 2023 年中国大陆集成电路产业人才供需报告[R/OL]. http://www.by-consulting.cn/home/projects/detail/id/10.

[3] LEE Y, WATERMAN A, COOK H, 等. An Agile Approach to Building RISC-V Microprocessors[J/OL]. IEEE Micro, 2016, 36(2): 8-20. DOI:10.1109/MM.2016.11.

[4] PATTERSON D A, DITZEL D R. The case for the reduced instruction set computer[J]. ACM SIGARCH Computer Architecture News, 1980, 8(6): 25-33.

[5] FOWLER M, HIGHSMITH J, OTHERS. The agile manifesto[J]. Software development, 2001, 9(8): 28-35.

[6] 包云岗, 常轶松, 韩银和, 等. 处理器芯片敏捷设计方法：问题与挑战[J]. 计算机研究与发展, 2021, 58(6): 1131-1145.

[7] FERDJALLAH M. Introduction to digital systems: modeling, synthesis, and simulation using VHDL[M]. John Wiley & Sons, 2011.

[8] BACHRACH J, VO H, RICHARDS B, 等. Chisel: Constructing Hardware in a Scala Embedded Language[C/OL]//Proceedings of the 49th Annual Design Automation Conference. Association for Computing Machinery, 2012: 1216-1225. DOI: 10.1145/2228360.2228584.

[9] ODERSKY M, ALTHERR P, CREMET V, 等. An overview of the Scala programming language[J/OL]. 2004. http://infoscience.epfl.ch/record/52656.

[10] LI P S, IZRAELEVITZ A M, BACHRACH J. Specification for the FIRRTL Language[R/OL]. (2016-02). http://www2.eecs.berkeley.edu/Pubs/TechRpts/2016/EECS-2016-9.html.

[11] ELDRIDGE S, BARUA P, CHAPYZHENKA A, 等. MLIR as hardware compiler infrastructure[C]//Workshop on Open-Source EDA Technology (WOSET). 2021.

[12] LATTNER C, AMINI M, BONDHUGULA U, 等. MLIR: Scaling compiler infrastructure for domain specific computation[C]//2021 IEEE/ACM International Symposium on Code Generation and Optimization (CGO). 2021: 2-14.

[13] IZRAELEVITZ A, KOENIG J, LI P, 等. Reusability is FIRRTL ground: Hardware construction languages, compiler frameworks, and transformations[C/OL]//2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD): 卷 0. 2017: 209-216. DOI:10.1109/ICCAD.2017.8203780.

[14] What SystemVerilog features should be avoided in synthesis?[EB/OL]. https://stackoverflow.com/questions/20312810/what-systemverilog-features-should-be-avoided-in-synthesis.

[15] 余子濠, 刘志刚, 李一苇, 等. 芯片敏捷开发实践:标签化 RISC-V[J]. 计算机研究与发展, 2019, 56(1): 35-48.

[16] 吕治宽. 基于 verilog 和 chisel 的乱序超标量 RISC-V 处理器设计比较[D]. 2022.

[17] 王晓华, 闫其涛, 程智强, 等. 科技论文中文摘要写作要点分析[J]. 编辑学报, 2010(S1): 53-55.

附录

A.1 附录子标题

A.1.1 附录子子标题

附录内容，这里也可以加入图片，例如图 A.1。

图 A.1 图片测试

致　　谢

感谢 NJU-LUG，提供 NJUThesis Typst 模板。