

**本 科 毕 业 设 计（论文）**



**题目: 基于 RISC-V P 扩展的脉冲神经网络加速处理器模块以及扩展指令集设计**

**姓 名 王九龙**

**学 院 电子工程学院**

**专 业 电子科学与技术**

**班 级 2019211209**

**学 号 2019211138**

**指导教师 漆渊**

**2023 年 5 月**

**基于 RISC-V P 扩展的脉冲神经网络加速处理器基于 RISC-V P 扩展的脉冲神经网络加速处理器**

**摘 要**

随着人工智能的不断发展，人工神经网络作为其重要技术手段受到广泛的关注。脉冲神经网络作为第三代人工神经网络在低功耗方面有着巨大的优势，可以应用于许多低功耗场景，如物联网终端、通信系统、边缘计算等领域。面向脉冲神经网络计算的加速器也应运而生，但是这样的系统需要加速器配合控制核进行计算，容易出现指令架构差异、访存速度瓶颈等问题。针对以上问题，本毕设基于开源指令集RISC-V中的P扩展指令，通过集成脉冲神经网络扩展指令与加速模块到一个双发射顺序处理器中，使得处理器支持并加速脉冲神经网络相关计算、单指令多数据（SIMD）计算、16 bit定点数指数函数（EXP）计算以及求和计算。

本毕设的主要工作为：第一，基于RISC-V指令集P扩展指令设计脉冲神经网络扩展指令，包括神经元计算相关指令、突触及通用计算指令和脉冲神经网络配置指令。第二，使用Chisel HDL设计并实现加速模块的硬件设计。第三，搭建软硬件测试环境，验证指令集和硬件设计的功能正确性。第四，通过在FPGA上部署本设计，得到功耗、硬件资源使用以及计算速度的相关数据。最后得到结论：在相同硬件平台上相较于传统“加速器+控制核”方法，本设计的硬件资源使用与功耗上有显著提升；同时，相较于C语言实现的脉冲神经网络计算相关的程序，在相同运行平台上，调用脉冲神经网络扩展指令能够将计算性能提升74.7%-91.0%。

**关键词** 脉冲神经网络 指令集扩展 计算加速

**The Design of Spiking Neural Network Accelerate Module and Extension Instructions based on RISC-V P-Extension**

**ABSTRACT**

With the continuous development of artificial intelligence (AI), artificial neural network has received extensive attention as an important technique of AI. The third-generation artificial neural network, spiking neural network (SNN), has great advantages in low power consumption and can be applied to many low-power consumption scenarios, such as Internet of Things terminals, communication systems, edge computing and other fields. Accelerators for SNN calculations have also emerged as the times require, but such systems require accelerators to cooperate with control cores for calculations, which are prone to problems such as differences in instruction architectures and memory access speed bottlenecks. In response to the above problems, this project designs the SNN instruction extensions and accelerating module based on the RISC-V P instruction extensions, and integrates them into a dual-issue sequential processor, so that the processor supports and accelerates the SNN related calculations , Single Instruction Multiple Data (SIMD) calculation, 16 bit fixed-point exponential function (EXP) calculation and summation calculation.

The main jobs of this project are as follows: First, the spiking neural network extension instructions are designed based on the RISC-V instruction set P extension instructions, including neuron computing related instructions, synapse and general computing instructions, and spiking neural network configuration instructions. Second, this project uses Chisel HDL to design and implements the hardware design of the acceleration module. Third, this project builds both software and hardware testing environment to verify the functional correctness of the instruction set and hardware design. Fourth, by deploying this design on the FPGA, relevant data on power consumption, hardware resource usage, and computing speed are obtained. Finally, it is concluded that compared with the traditional "accelerator + control core" method on the same hardware platform, the hardware resource usage and power consumption of this design are significantly improved; For the program performance, on the same platform, calling spiking neural network extension instructions can improve computing performance by 74.7%-91.0%.

**KEY WORDS** Spiking Neural Network, Instruction Extensions, Computing Acceleration

**目 录**

[第一章 绪论 1](#_Toc134806966)

[1.1 研究背景与研究意义 1](#_Toc134806967)

[1.2国内外研究现状 2](#_Toc134806968)

[1.2.1脉冲神经网络加速硬件实现的研究 2](#_Toc134806969)

[1.2.2 RISC-V 开源指令集与指令集扩展的研究 4](#_Toc134806970)

[1.3 开发工具介绍 5](#_Toc134806971)

[1.3.1 硬件敏捷开发语言:Chisel HDL 5](#_Toc134806972)

[1.3.2 开源Verilog仿真器和裸机运行时环境 6](#_Toc134806973)

[1.4 论文主要研究内容与章节安排 6](#_Toc134806974)

[1.4.1 论文主要研究内容 6](#_Toc134806975)

[1.4.2 论文章节安排 6](#_Toc134806976)

[第二章 脉冲神经网络模型以及RISC-V指令集架构介绍 8](#_Toc134806977)

[2.1 脉冲神经网络模型 8](#_Toc134806979)

[2.1.1脉冲神经网络的神经元模型 8](#_Toc134806980)

[2.1.2 脉冲神经网络的突触可塑性 9](#_Toc134806981)

[2.1.2.1 脉冲时间相关的突触可塑性学习规则 9](#_Toc134806982)

[2.1.2.2 脉冲驱动的突触可塑性学习规则 11](#_Toc134806983)

[2.1.2.3反向传播的脉冲时间相关的突触可塑性学习规则 11](#_Toc134806984)

[2.1.3 神经网络的拓扑结构 13](#_Toc134806985)

[2.2 RISC-V 指令集架构介绍 14](#_Toc134806986)

[2.2.1 RISC-V 指令集的组成 14](#_Toc134806987)

[2.2.2 RISC-V 指令格式 15](#_Toc134806988)

[2.2.3 RISC-V中的P指令集扩展 15](#_Toc134806989)

[2.3 本章小结 16](#_Toc134806990)

[第三章 脉冲神经网络扩展指令设计与加速模块的硬件实现 17](#_Toc134806991)

[3.1 脉冲神经网络扩展指令设计 17](#_Toc134806992)

[3.1.1 神经元更新指令 17](#_Toc134806993)

[3.1.2 突触学习与通用运算指令 18](#_Toc134806994)

[3.1.3 脉冲神经网络配置指令 19](#_Toc134806995)

[3.2 脉冲神经网络加速模块整体设计 21](#_Toc134806996)

[3.3 发射单元设计 21](#_Toc134806997)

[3.4 LIF神经元更新单元设计 22](#_Toc134806998)

[3.5 突触与通用计算单元设计 23](#_Toc134806999)

[3.6 本章小结 26](#_Toc134807000)

[第四章 功能验证与性能分析 27](#_Toc134807001)

[4.1 功能验证 27](#_Toc134807002)

[4.1.1 软件测试平台功能验证 27](#_Toc134807003)

[4.1.2 硬件FPGA测试平台的功能验证 29](#_Toc134807004)

[4.2性能与加速结果分析 31](#_Toc134807005)

[4.3 本章小结 32](#_Toc134807006)

[第五章 全文总结与展望 33](#_Toc134807007)

[参考文献 34](#_Toc134807008)

# 第一章 绪论

本章将介绍本设计在当今人工智能发展下的研究背景，进一步阐述本设计的研究意义。同时将概述国内外神经形态计算硬件实现以及扩展指令相关研究的现状，并对开发工具进行介绍。最后将说明本设计的研究内容以及本文的行文结构。

## 1.1 研究背景与研究意义

近年来，随着人工智能（AI，artificial intelligence）的快速发展，人工神经网络（ANN，artificial neural network）被应用于现在的很多场景中。例如在物联网终端中使用预训练好的神经网络模型进行推理，并完成图像处理或语音处理；在边缘计算中使用人工智能进行实时的数据处理，不必担心由于通信带宽带来的传输效率的瓶颈。同时由人工神经网络训练的预训练 Transformer AI 大模型[1]也在最近引起广泛的关注，这进一步刷新了人们对 AI 的认识。这些丰富的应用场景下 AI 的蓬勃发展伴随而来的是研究人员对极致性能的追求，对神经网络加速计算的研究也逐渐称为了焦点。

在这样的背景下，使用“终端-云服务器”的结构来进行部署人工神经网络称为了主流的硬件部署解决方案。但是这样的方案无疑对通信技术以及服务器规模有着很高的要求。因此，利用分布式将人工神经网络的计算核心部署在终端就是其中一个可能的解决方案。基于脉冲神经网络（Spiking Neural Network, SNN）的神经形态计算由于其高能效的特点成为低功耗人工智能实现的最佳选择。这样的神经形态计算区别于通用处理器所进行的计算，具有稀疏性以及专用性的特点。针对于 SNN 的加速计算，设计专用的加速器或加速模块比使用传统的通用处理器在硬件性能上更有优势。

与此同时，开源指令集——RISC-V 指令集由于其开源、简洁、规整以及易扩展等特性在硬件加速、嵌入式开发等方面展现出上述特性带来的优越性。对于加速计算，RISC-V 指令集提供 P 指令集扩展（RISC-V Packed-SIMD DSP instruction set extension）以及 V指令集扩展（RISC-V Vector instruction set extension）来进行加速。其中 P 扩展主要是应用于小型 SIMD（Single-Instruction-Multiple-Data）加速计算，主要利用通用处理器的通用寄存器来进行并行化计算；V 扩展需要再引入向量寄存器以及向量计算部件，从而实现并行计算。对于低功耗设备，P 扩展的能力足以覆盖需求，因此基于 P 扩展的研究也成为了在小型设备上实现并行化计算方案的一个重要研究方向。

目前在神经形态计算的硬件实现上，神经形态加速器成为了最近优化 SNN 加速计算的主要研究方向。现有的主流硬件加速方案是由一个通用处理器控制核（CPU）和SNN 专用神经形态计算加速器组成的计算系统，由 CPU 负责搬运数据，加速器负责计算数据。在加速器中集成了神经元模型用于训练和推理，还集成了突触可塑性学习算法用于训练网络。

但是，上述现有对脉冲神经网络的加速计算解决方案有以下问题：1）加速器需要控制核作为整个计算系统的组成部分，而数据搬运就出现了大部分能耗，这就会导致脉冲神经网络加速器低功耗特点无法完全发挥；2）加速器有自己独特的指令编码，当与控制核不属于一个架构时，需要在控制核与加速器之间使用如 SPI Flash 这样的储存介质以缓冲需要执行的指令，这对指令的高效执行产生很大的困难；3）将突触可塑性学习算法集成到硬件上使得系统的灵活性与可编程性变差，且大多专用于 SNN 的突触可塑性学习算法不适用于更深层的学习网络。

因此，讨论新的神经形态计算处理器的硬件结构以及加速方案就充满挑战与意义。针对上述三个问题，本设计拟基于 RISC-V P 扩展设计实现脉冲神经网络加速处理器模块以及扩展指令集。主要内容包括：采用紧耦合加速方案，在一个顺序双发射 RISC-V核心执行级中增加 SNN 加速单元，并采用 SNN 专用指令集扩展的方式来调用运算单元，不再需要中间介质（如上述提到的 SPI Flash）作为指令调用的缓冲，直接通过 SNN专用指令调用计算部件进行计算；基于 RISC-V 指令集 P 扩展的硬件实现方式，对神经元和突触更新进行并行化处理，增加其运算能力；支持 STDP 与 BP-STDP 更新算法，对多层网络训练有较好的适应性，并对特殊运算提供支持（exp 指数运算），也提高了算法实现的灵活性。由于 RISC-V 指令集的统一、简洁、易扩展特性，使得实现SNN 专用指令集具有可行性，这样的扩展指令在低功耗软硬件协同开发上具有较大优势。

## 1.2国内外研究现状

本研究的主题主要涉及两个方面：脉冲神经网络计算加速硬件实现与 RISC-V 开源指令集及其扩展。目前，随着对低功耗与高算力的需求，神经形态计算的研究逐渐成为加速神经形态计算以及控制功耗的研究焦点。同时，开源指令集 RISC-V 的可扩展性与灵活性为紧耦合神经形态计算加速提供了技术支持。下面将针对这两个领域的国内外研究现状进行详细的介绍。

### 1.2.1脉冲神经网络加速硬件实现的研究

随着对生物神经系统的不断了解，脉冲神经网络被称为下一代神经网络[3]。目前，脉冲神经网络的具体实施方向主要由软件和硬件组成。由于脉冲神经网络的异步式、稀疏式的计算特点，基于通用处理器实现的脉冲神经网络会因为处理器的性能被限制运算速度。因而为了从根本上解决这样的性能墙问题，国内外对于脉冲神经网络硬件化的研究展现出密切的关注。

目前对于脉冲神经网络硬件的实现有两个主要的方向，分别是利用忆阻器在电压操作下可以阻态转换而实现的存算一体化芯片与完全采用纯数字电路设计的神经形态计算芯片。

**1. 基于忆阻器的存算一体芯片**

忆阻器具有的纳米级尺寸及非易失性，能够在模拟神经突触时实现突触权值的不断变化，实现存算一体化。2010 年，通过操控忆阻器中的离子迁移过程从而精密调控器件的电阻转变，密歇根大学首先使用忆阻器件实现了突触权重的调节以及脉冲时序依赖可塑性[4]，从而开启了世界上对在模拟电路上使用忆阻器件实现神经形态计算的研究。近年来，基于忆阻器件的存算一体加速器发展迅速。2019 年，第一款基于忆阻器交叉阵列的存算一体化芯片实现了 3 种人工智能算法[5]。中国台湾清华大学采用 68 nm CMOS 工艺实现了一个 1Mb 的忆阻器随机存储阵列[6]，利用了忆阻器的多值突触特性以及二值阻变特性实现脉冲神经网络的模拟计算。

一方面，基于忆阻器的内存加速器将计算与存储紧密结合，从而省去传统的冯·诺依曼体系结构的中心处理器和内存之间的数据传输，进而提升整体系统的性能并节省大部分的系统能耗。另一方面，通过在忆阻器阵列外部加入一些功能单元，阵列能在几乎一个读操作的延迟内完成一次矩阵乘加计算。

**2. 纯数字电路神经形态计算芯片**

与此同时，纯数字电路的神经形态计算芯片也正随着物联网与边缘计算的发展而受到关注。2014 年，IBM 设计并生产了“TrueNorth”芯片[7]。“TrueNorth”芯片本身是一个多核处理器，但是每个核心中包含了 256 个可编程设计的模拟神经元。整个芯片含有4096 个核心，因此总计支持的神经元约为一百万个。每个神经元之间都有可编程的传递信号的突触，其数量为 228 条。“TrueNorth”芯片在运行时功耗约为 70 毫瓦，是传统微处理器的万分之一。英特尔公司在 2017 年设计并生产了能够自我学习神经形态芯片“Loihi”[8]，采用英特尔 14 nm 制程，共包含了 131072 个神经元，约 1.3 亿条突触。罗格斯大学的研究人员[9]使用“Loihi”展示了其在解决同时地图构建与定位问题时的高效能源效率。2021 年，因特尔公司发布了“Loihi”芯片的升级版本——“Loihi 2”[10]，支持更加广泛的神经元模型以及可编程的突触学习算法。例如用于计算短时傅里叶变换（STFT，Short Time Fourier Transform）的共振发射 (RA，Resonate-and-Fire) 神经元，其计算复杂度与传统的 STFT 类似，但是输出带宽少 47 倍。

在国内研究中，清华大学发布的“天机（Tianjic）”芯片采用 ANN 与 SNN 异构融合的设计[10]，实现了四万神经元和千万级突触，同时在使用 SNN 的模式时能效可以达到649 GSOPS/W[[1]](#footnote-1)。2022 年，由中国科学院计算技术研究所开发的“文曲星 22A”芯片[11]采用了紧耦合设计，将 SNN 模块集成为通用处理器的一部分，并实现了二值权重的脉冲神经网络模型。

在纯数字电路神经形态计算设计中，基于脉冲神经网络的 ASIC 加速器设计也十分突出。其中最具有代表性的是由 Frenkel 提出的在线学习加速器 ODIN[12]。ODIN 集成了 256 个神经元以及约 64000 条突触。ODIN 中神经元可配置为标准 LIF 模型（leaky integrate-and-fire model）以及模仿 20 种 Izh 模型[13]行为（Izhikevich behaviors）的自定义神经元。在突触更新上采用了脉冲驱动突触可塑性（spike-driven synaptic plasticity，SDSP）学习规则对 4 bit 突触进行调整，并实现了在线学习的功能（Online Learning）。其每突触操作最小功耗为 12.7pJ。然而与 ODIN 类似的加速器硬件设计在实现上需要一个控制核负责数据搬运以及指令的发送[14]，以及需要一个中间层如 SPI flash 来作为核与加速器之间的通信桥梁。

值得一提的是，2013 年开始的由欧洲工程与物理科学研究委员会（Engineering and Physical Sciences Research Council）资助的人类大脑计划项目发布了由曼彻斯特大学主导研究的“SpiNNaker”芯片（The SpiNNaker Chip）[15]。这款芯片采用了片上网络技术，集成了千万级基于 ARM 架构的嵌入式核心来实现实时类脑计算。

### 1.2.2 RISC-V 开源指令集与指令集扩展的研究

**1. RISC-V 开源指令的研究**

RISC-V 指令集架构是由美国加州大学伯克利分校 2010 年发布[16]的一种新型开源精简指令集架构。这个架构具有模块化设计、可扩展性、可定制性以等特点，在嵌入式系统场景下被广泛的应用，特别是在物联网、边缘计算、可穿戴设备等领域。在近年来，RISC-V 的研究在学术界和工业界成为了一个焦点，并出现了许多显著的成果。2011 年，加州大学伯克利分校成功流片第一个 RISC-V 嵌入式微处理器“Raven-1”[17]。2013 年，使用 Chisel HDL 设计的 RISC-V 架构顺序处理器“Rocket”发布[18]，其工作频率可达到1 GHz。2018 年，同样使用 Chisel HDL 设计的 RISC-V 架构乱序处理器“Boom”[19]发布并流片。

国内对于 RISC-V 架构的研究同样关注。阿里巴巴公司设计发布了“玄铁-910”[20]芯片。中国科学院计算技术研究所发布了基于 RISC-V 架构的高性能开源处理器“香山”[21] 芯片, 在 28 nm 工艺节点下达到 1.3GHz 的频率。

**2. RISC-V 指令集扩展的研究**

RISC-V 标准组织为这套指令集架构（ISA，instruction set architecture）定义了一系列除基础指令集之外的指令集扩展，包括 P 扩展、V 扩展等。实际上，除了 RISC-V 标准组织制定的 ISA 规范外，还有各种新颖的研究提出了 RISC-V 对许多场景的定制扩展，包括物联网、人工智能、通信、图形计算、后量子加密、高性能计算、虚拟化和安全。

在物联网场景下，有一种新兴趋势是根据物联网应用程序对计算精度和能耗的要求来调整浮点格式。然而，目前的 RISC-V 官方 ISA 规范只提供了 binary32、binary64 和binary128 指令集扩展，缺乏对较低精度浮点格式的支持。Tagliavini 等[22]提出了用于RISC-V 的小于 32 位的浮点（smallFloat）的指令集扩展。

在人工智能的场景下，卷积神经网络（CNN, converlutional neural network）等新的人工智能方法层出不穷。然而，人工智能方法对巨大的计算和存储资源的需求限制了其性能的提升和更广泛的应用。因此，各类针对优化人工智能算法的 RISC-V 指令集扩展被提出。Cococcioni 等人[23]提出了针对深度神经网络计算优化的指令集扩展；Wang 等人[24]实现了对卷积神经网络的自定义指令集扩展；Jiao 等人[25]针对大模型训练 Transformer提出了 RISC-VTF 扩展，用于加速优化 Transformer 的算法实现。

在通信领域，第六代无线通信技术（6G，6-Generation）将比其前身第五代无线通信技术（5-Generation）更加强调低延迟、高吞吐量和智能化。由于这一趋势，对无线通信基础设施的适应性和可扩展性有着更严格的要求。专用指令集处理器（ASIP，Application-specific instruction-set processors）在灵活性、可负担性和效率之间取得了很好的平衡，有望成为通信处理硬件的使能技术之一。RISC-V 指令集架构为设计人员提供了一种替代方案，允许在通信 ASIP 中进行定制和创新。在当前的研究中，针对于信号处理领域有Razilov 等人[26]使用 RISC-V 向量扩展进行计算；针对纠错编码，Tourres 等人[27]提出自定义的扩展指令来实现算法；对无线电资源管理也有各种指令集扩展来帮助实现复杂通信算法[28][29]。

随着现代信息技术逐渐向软硬件协同设计以及面向需求的精致化定制的方向发展，现有国内外对于 RISC-V 指令集架构以及其指令集扩展的研究越来越受到重视。并随着开源思想的逐步深入发展，RISC-V 这一开源指令集架构的特点能够发挥开放系统的独特优势。

总的来说，对 SNN 的硬件加速方案有着模拟电路与数字电路两个不同的方向。模拟电路实现使用了忆阻器，通过存算一体化来提高运算效率；数字电路实现上则是集成可神经元或突触，并通过并行化提高运算效率。同时 RISC-V ISA 也为神经形态计算处理器的开发提供了一个高效的指令集架构。本设计正是基于 RISC-V ISA 设计了自定义的 SNN 专用扩展指令，并完成 SNN 加速模块的数字带你来设计，将其集成到一个双发射顺序处理器中，让它具备神经形态计算能力。

## 1.3 开发工具介绍

### 1.3.1 硬件敏捷开发语言:Chisel HDL

本研究使用 Chisel 硬件描述语言（Chisel HDL，Chisel hardware description language）作为开发语言。Chisel 是由加州大学伯克利分校[30]进行开发的，是一种基于 Scala 语言的硬件描述语言。

在目前的数字电路设计中，绝大多数项目采用的设计以及测试语言为 Verilog HDL或 VHDL。然而，在上述的硬件描述语言中，有很大一部分语义并不适用于硬件综合。很多结构在如何映射到硬件上并不直观，这样就造成了混乱的模型规范。与此同时，虽然 Verilog 或 VHDL 包含一些用于生成电路的原始结构，但是他们缺乏现代编程语言所具备的强大敏捷开发功能，如面向对象以及函数式编程等。

与使用 Verilog 直接描述要生成的电路不同，使用 Chisel 设计电路更多的是在设计一个硬件的生成器。这些硬件生成器是对一系列具有相同功能与结构的电路抽象的结果。由于 Chisel 的设计理念是将硬件构造原语嵌入到现有语言（Scala）中，所以 Chisel可以继承到现代编程语言的强大特性。在设计电路的同时，采用面向对象与函数式编程的思想将电路高度抽象，实现一类电路的设计，并最终形成硬件生成器。

使用 Chisel 的设计，首先通过 Chisel 编译器生成抽象语法树数据。随后再通过 firrtl编译器编译为 firrtl 文件。这些 firrtl 文件作为一种中间交换模式，这也是一种将高级编程语言转化为 Verilog/VHDL 的一个媒介。随后再次经过 firrtl 编译器将 firrtl 文件转化为 Verilog 代码。整个过程如图1-1所示。由于当前的环境下没有直接支持 Chisel 的 EDA（Electronic DesignAutomation）工具，所以最后 Chisel 没有直接生成电路，而还是生成了 Verilog 代码。



图1-1 chisel代码文件编译为Verilog代码文件的过程

### 1.3.2 开源Verilog仿真器和裸机运行时环境

Verilator 是由 Wilson Snyder 开发[31]的一个高性能的开源 Verilog 仿真器。其主要功能是将 Verilog 转变为 C++ 或 SystemC 中的周期精确模型。由于这些生成的模型是周期精确的，因此 Verilator 通常比使用更广泛的事件驱动的仿真器有着更高的性能。Verilator现在广泛用于学术研究、开源项目和商业半导体开发，并逐渐成为开源 EDA 的一部分。

由于本毕设是指令集与处理器设计，属于软硬件协同设计，在最终的验证上需要一个可以解释并运行我们自定义指令的运行时环境来运行对应的特殊指令。为此本设计使用了由南京大学蒋岩炎老师设计的裸机运行时环境[32]：Abstract Machine（AM）。简单来说，AM 就是一个面向程序运行时所提供的库，根据程序的需求 AM 把库划分成以下模块：

* TRM(Turing Machine) - 图灵机, 最简单的运行时环境, 为程序提供基本的计算能力；
* IOE(I/O Extension) - 输入输出扩展, 为程序提供输出输入的能力;
* CTE(Context Extension) - 上下文扩展, 为程序提供上下文管理的能力;
* VME(Virtual Memory Extension) - 虚存扩展, 为程序提供虚存管理的能力;
* MPE(Multi-Processor Extension) - 多处理器扩展, 为程序提供多处理器通信的能力.

在程序被编译为可执行的二进制文件并传输进入硬件模拟器执行前，AM便提供了一个完整的裸机时运行的环境，以便于硬件模拟器能够实现设计所需要的功能。

## 1.4 论文主要研究内容与章节安排

### 1.4.1 论文主要研究内容

本设计主要根据脉冲神经网络的模型以及计算特点，基于 RISC-V P 指令集扩展的设计思路设计脉冲神经网络加速扩展指令（RV-SNN）；然后根据RV-SNN的指令集定义对应实现执行指令的脉冲神经网络加速扩展模块，并最终将模块嵌入到支持 RISC-V P 扩展的双发射顺序处理器中；完成硬件设计后使用Verilator以及AM进行软件仿真，验证硬件设计的功能正确性；最后将处理器部署到 FPGA 上进行验证，获得最终的性能指标。

### 1.4.2 论文章节安排

本文主要包含五个章节，每个章节的安排如下：

第一章，绪论。首先介绍脉冲神经网络加速计算硬件设计的研究背景与研究意义。随后对当前脉冲神经网络加速计算的硬件设计以及 RISC-V 的指令集扩展在国内外的研究现状进行了介绍。最后介绍了本设计使用的开发工具，并概括了本设计的研究内容和本文的章节安排。

第二章，脉冲神经网络模型以及开源指令集 RISC-V ISA 介绍。本章节主要介绍了对本设计十分重要的前置概念。首先通过详细列举说明脉冲神经网络模型中的神经元模型与脉冲神经网络学习规则，介绍了脉冲神经网络模型的组成部分，并对脉冲神经网络模型的整体结构进行了介绍。随后详细介绍开源指令集架构 RISC-V 的组成，说明了RISC-V 指令集定义的规范。同时也特别介绍了 RISC-V 指令集架构中的 P 指令集扩展，为随后的指令集设计实现 SIMD 提供了理论基础。

第三章，脉冲神经网络指令集设计与硬件实现。本章由设计一套指令集的流程为引入，按照其不同功能属性将指令集分为了神经元更新指令、突触学习指令、通用计算指令以及脉冲神经网络配置指令四个部分，分门别类地介绍了每部分所包含指令的定义、编码和功能。同时本章自顶向下介绍脉冲神经网络加速计算模块的电路设计。其中，按照模块化设计，将整个加速模块分为了发射单元、LIF神经元更新单元和突触及基本运算单元三个部分，分别介绍硬件的微架构设计，流水线处理等。

第四章，功能验证和性能分析。本章首先介绍本设计中使用到的软硬件测试平台的搭建，以及基于测试平台的功能验证过程。随后再根据功能验证中得到的性能数据进行详细分析，并与其他脉冲神经网络加速有关的硬件设计进行对比。

第五章，全文总结与展望。本章节对本文的工作做出总结，并对未来的改进方向与研究思路进行展望。

# 第二章 脉冲神经网络模型以及RISC-V指令集架构介绍

本章将介绍脉冲神经网络模型以及RISC-V指令集架构（ISA，instruction set architecture）。脉冲神经网络模型是本设计主要研究的对象，模型的特征正是实现一系列脉冲神经网络加速计算的理论基础。而RISC-V指令集架构是本设计的技术基础，脉冲神经网络加速指令正是根据RISC-V ISA规范而定义并实现的。在本设计中实现的SIMD计算也是参考了RISC-V指令集提供的P指令集扩展而设计的，因此对RISC-V P扩展的介绍也十分重要。



## 2.1 脉冲神经网络模型

在脉冲神经网络模型中，神经元模型作为网络计算的基本处理单元，突触可塑性作为网络记忆学习的基本理论，二者与网络的拓扑结构共同构成了脉冲神经网络模型中的三个要素。本节将分别介绍这三个重要的要素。

### 2.1.1脉冲神经网络的神经元模型

根据生物学上对神经元的定义，一个神经元的结构包括了树突，胞体和轴突三个部分，其中树突负责收集来自其他神经元的输入信号，胞体积累膜电位到阈值产生脉冲，轴突负责神经脉冲的发放。在这三个部分中，胞体可以看作一个神经元中主要负责运算的部分。针对这样的特点，神经生理学家建立了多种神经元模型，其中对脉冲神经网络实现类脑计算影响较大的模型有 H-H（Hodgkin–Huxley）模型以及 LIF（Leaky integrate-and-fire）模型等。

1）Hodgkin–Huxley 模型：在 1952 年 Hodgkin 和 Huxley 在对乌贼轴突的电位变化研究[33]中提出了神经元的电位活动机制理论模型，随后成为了多种不同生理结构的神经元模型的雏形。根据其实验和数据，Hodgkin 和 Huxley 得到了如的微分方程：





其中：为膜电流总密度；为膜电位；为膜单位面积电容；、、分别表示钾离子、其他离子和钠离子的电导密度；、、分别为对应离子通道的反转电位；、、分别为假设的几种与离子运转相关的粒子浓度；、分别表示膜内、膜外离子移动速度。

在H-H模型中，动作电位与各类离子的移动被精确地建模了出来。可以说，H-H模型是整个脉冲神经网络的神经元模型的起点。然而，即使H-H模型的生物可靠性很高，但对于人工网络中的神经元实现来说还是过于复杂。

2）LIF模型：在1907年，Lapicque提出了IF（Integrate-and-fire）模型。但由于当时对于神经元如何产生动作电位的了解不足，神经元发放脉冲的过程就被简化为了神经元收到激励累积膜电位，到达阈值电压后输出脉冲并落回静息电位。事实上，神经元的发放过程包含了三个关键的特征，分别是累积、泄露和发放。2010 年，Indiveri 等人提出的 LIF（leaky integrate-and-fire）模型[34]则简化了 H-H 模型的工作电位产生过程，保留了上述关键的三个特征。LIF 模型的数学描述可以用表示



其中: 是膜时间常数；表示静息电位；、分别为膜阻抗和输入电流。当膜电位超过阈值电压时，神经元会发放脉冲并将膜电位重置为静息电位后进入一段不应期。在神经网络中对神经元的计算就是这一过程的不断循环。由于LIF模型的实现简单，且保留了神经元的生物学特征，所以LIF模型也被广泛用于脉冲神经网络的设计。

在进一步的研究中，为了提高 LIF 模型的生物可信度，研究人员提出了更多的变体，如指数 LIF 模型[35]、二阶 LIF 模型[36]等。

由于 LIF 模型对硬件实现较为简单且具有足够的生物学特征，因此本设计将采用 LIF 模型作为神经网络中的基本神经元模型。

### 2.1.2 脉冲神经网络的突触可塑性

最初人们对于大脑学习机制的认识十分有限，但是在突触这一概念尚未提出时，Bain[37]就提出了细胞间的物理连接正是支持学习和记忆的基础。随着突触以及突触可塑性概念的提出以及对神经学的深入研究，人们对于神经元之间的联系有了进一步认识。这对于大脑如何实现学习和记忆产生了深远影响。

### 2.1.2.1 脉冲时间相关的突触可塑性学习规则

在 1949 年，Hebb[38]指出了早期对突触可塑性的研究结果。这些研究成果形成了一个影响巨大的假设: 当一个细胞（A）的激活重复地参与另一个细胞（B）的激活，那这两个细胞将会产生某种相互依赖的生长过程。换言之，细胞 A 将随着这样的生长过程更高效地参与到细胞 B 的激活与发放中，这两个同时激活的细胞将彼此建立连接。随后的一系列研究中[39]，通过激发前后神经单元诱导了长时程增强（LTP，long-term potentiation）与长时程抑制（LTD，long-term depression）现象的产生。这进一步为 Hebb 的假说提供了支撑。

记忆是与时间相关的，因此在对突触可塑性的研究中需要引入时间的概念。Markram等[40][41]的研究中，进一步证明了长时程增强与长时程抑制是与时间密切相关的。在相同的时间差异下，前后两个神经元的发放顺序会导致不同的结果：前神经元先于后神经元发放，会导致 LTP；反之则会导致 LTD。这也说明了同时发放了两个神经元并不一定会产生联系。这样的突触可塑性也被称为了脉冲时间相关的突触可塑性（STDP，spike-timing dependent plasticity）。



图2-1 脉冲时间相关的突触可塑性示意图

如图 2-1 所示，、分别为前神经元和后神经元的发放时刻；为突触权重的变化量。图中前神经元先于后神经元发放，因此导致连接两神经元的突触发生长时程增强现象（图中的LTP）；若发放顺序相反则引发长时程抑制（图中LTD）。

在人工神经网络模型中，适用于脉冲神经网络学习的STDP数学模型由 Cassidy 等人提出[42]，基本 STDP的数学模型表达如所示。



其中，为突触权重改变量，为如图2-1中所示的STDP函数（或称之为学习窗口）；参数与取决于当前突触权重；与都是时间常数。

STDP学习规则有着良好的局部特征学习能力，是脉冲神经网络中无监督算法的重要模拟对象。在此基础上，研究人员对基本STDP进行了不同的变化，使其更准确或更适应于监督学习算法。

### 2.1.2.2 脉冲驱动的突触可塑性学习规则

脉冲驱动的突触可塑性学习规则（SDSP，spike driven synaptic plasticity）是基于基本的 STDP 学习规则的变体，由 Brader 等人提出[43]。在每一次脉冲发生时，都会引发SDSP 学习规则对权重的更新。SDSP 对突触权重的增加和减少遵循如所示的数学表达，并且只取决于脉冲发生前后的神经元状态（膜电位）。



其中，表示膜电位，与表示的膜阈值电压进行比较；表示钙离子浓度，与、、和表示的离子浓度阈值进行比较。同时，钙浓度也代表了突触后放电活动的图像。SDSP的学习示意图如图2-2所示。



图2-2 脉冲驱动的突触可塑性示意图

### 2.1.2.3反向传播的脉冲时间相关的突触可塑性学习规则

在现有的人工神经网络实现中，反向传播是训练深度学习网络的一个重要基础。网络通过链式法则对损失函数进行梯度下降的操作，从而对网络中的参数进行调整，使得整个网络的损失达到极小。随着网络层数的增加，脉冲神经网络中通过突触可塑性训练得到的模型与传统人工神经网络使用反向传播训练得到的模型性能上出现明显的差距。在这样的情况下，研究人员开始探寻在脉冲神经网络中使用反向传播的可能性。

1）基于梯度替代的反向传播算法。要将反向传播应用于脉冲神经网络，一个重要的问题在于脉冲的正向传播是不可导的。在传统的人工神经网络中，神经元的计算输出是由一个连续函数产生的。在训练的过程中，这个连续函数就可以参与到对整个损失的链式求导中。然而脉冲神经网络模型中，描述神经元放电过程是通过一个 Heaviside 阶跃函数来实现的。而这个函数如果参与到链式求导的过程中会导致整个网络出现梯度爆炸或消失。因此，为了能够参与反向传播中，许多研究提出了各种梯度替代法（SG method，Surrogate gradients method）[44][45][46]。其主要思想是，在网络正向传播时，仍使用阶跃函数来进行神经元放电；在反向传播中使用链式法则进行参数调整时就采用替代的发放函数。

基于脉冲时间相关的突触可塑性的反向传播算法。同样时为了在脉冲神经网络中实现反向传播，基于脉冲时间相关的突触可塑性的反向传播算法（BP-STDP, backpropagation STDP）[47]在 2019 年被提出。这个算法与上述梯度替代方法不同之处在于，BP-STDP是根据基本的 STDP 规则进行的近似。换言之，BP-STDP 集合了反向传播对损失的敏感以及 STDP 中神经元学习与记忆的生物学解释。BP-STDP 学习规则中，错误（损失）从输出开始向前传播，参与到每一层突触权重的改变中。假设一个脉冲神经网络包含三层结构，分别为输入层、隐藏层和输出层，那么输出层与隐藏层之间权重变化量可以用进行表示。



其中表示学习率；是由输出层错误（损失）决定的变量，它同时决定了此时对突触的学习是增强、抑制还是不变；表示隐藏层神经元在时间内的脉冲总和。



隐藏层与输入层之间的权重变化由中的数学表达式来描述。与输出层-隐藏层之间权重变化类似，表示学习率；是由输出层错误（损失）决定的变量；为输出-隐藏层神经元之间连接的权重；表示输入层神经元在时间内的脉冲总和。如果隐藏层增加，对应的数学表达应该与相同，只是将输入层替换为另一个隐藏层。

BP-STDP在结合了脉冲神经网络以及反向传播的同时，保留了STDP中神经元的发放顺序对突触可塑性变化的影响。同时，这也是一种在硬件实现上相较于梯度替代方法更友好的学习规则，可以为脉冲神经网络的加速计算提供支持。

### 2.1.3 神经网络的拓扑结构

在现有的神经网络的研究中，脉冲神经网络的拓扑结构与传统人工神经网络一样被分为两种结构：前馈型脉冲神经网络以及递归型脉冲神经网络。

在前馈型脉冲神经网络中，两个神经元之间可以拥有多个突触连接，每个突触的延时不同，且包含了可改变的突触权重。前馈型脉冲神经网络示意如图 2-3 所示。多突触的不同延时使得突触前神经元输入的脉冲能够在更长的时间范围对突触后神经元的脉冲发放产生影响。突触前神经元传递的多个脉冲再根据突触权值的大小产生不同的突触后电位。



图 2-3 前馈型脉冲神经网络示意图

递归型脉冲神经网络中含有反馈路径，网络中的神经元发放脉冲是根据以前时间节点上的网络状态进行更新的。递归型脉冲神经网络可以很好地模拟时间序列，用来完成预测、控制等工作。递归型脉冲神经网络如图2-4所示，其中为*t*时刻输入层输入，为*t*时刻经过第*l*层突触传递后的乘以权重的值，为*t*时刻第*l*层神经元输出。



图 2-4 递归型脉冲神经网络示意图

## 2.2 RISC-V 指令集架构介绍

指令集架构是计算机系统中不可或缺的一部分。可以说，指令集的定义就是计算机硬件实现的标准。正是由于指令集定义了各类有关运算器和寄存器的操作，才有了用这些指令组合形成的程序。当前主流的指令集架构主要为由英特尔公司制定的X86架构、ARM公司制定的ARM架构以及逐渐引起学术界和工业界关注的RISC-V架构。本节将对RISC-V架构的组成以及其中的指令集扩展进行详细的介绍。

### 2.2.1 RISC-V 指令集的组成

RISC-V标准化组织将RISC-V指令集架构以**模块化**的形式进行制定，分为了一个基本指令集架构和各类可选的指令集扩展。RISC-V架构的核心是一个名为RV32I（32位整数指令集）的基础ISA，这使得完整的软件可以被正常运行。根据不同的场景，就可以选择不同的指令集搭配方式。

对于嵌入式场景来说，根据应用程序的需要，在硬件实现时就可以选择包含或不包含某些扩展。例如使用RV32IMFD将单精度浮点（RV32F）和双精度浮点（RV32D）的扩展添加到了基础指令集（RV32I）中。一般来说，在表示使用的RISC-V架构中包含了哪些指令集扩展，习惯上要把代表扩展的字母附加到指令集名称之后作为指示。而对于需要启动操作系统的场景，硬件就必须要包含一些必须的指令集扩展。启动操作系统的最低标准要使用RV32GC[[2]](#footnote-2)标准。

表2-1展示了RISC-V中常见的指令集扩展。

表2-1 RISC-V指令集架构中常见的指令集扩展

|  |  |  |
| --- | --- | --- |
| RISC-V基础指令集（Base） | | |
| 名称 | 描述 | 指令数量 |
| RV32I | 32位整数指令集 | 40 |
| RV32E | 32位整数指令集（嵌入式），32位，16寄存器 | 40 |
| RV64I | 64位整数指令集 | 15 |
| RISC-V指令集扩展（Extension） | | |
| M | 标准整数乘除指令集扩展 | 8(RV32)  13(RV64) |
| A | 标准原子（Atomic）指令集扩展 | 11(RV32)  22(RV64) |
| F | 标准单精度浮点指令集扩展 | 26 (RV32)  30 (RV64) |
| D | 标准双精度浮点指令集扩展 | 26 (RV32)  32 (RV64) |
| Zicsr | 控制和状态寄存器 (CSR，Control and Status Register) 指令 | 6 |
| Zifencei | 指令获取栅（Instruction-Fetch Fence）扩展 | 1 |
| C | 标准压缩指令集扩展 | 40 |

### 2.2.2 RISC-V 指令格式

如图2-5所示，RISC-V指令集包含6中基本指令格式，包括关于寄存器-寄存器操作的R型指令、用于短立即数以及访存load操作的I型指令、用于访存store操作的S型指令、用于条件跳转的B型指令、用于长立即数的U型指令以及用于无条件跳转操作的J型指令。为了方便程序员调试，所有32位都为0的指令是非法的，这有助于对一些异常的调试。



图2-5 RISC-V指令集中6种指令格式定义

图中，funct7、funct3是分别是指令的功能码；rs1、rs2分别是源寄存器1和2；rd为目标寄存器；opcode是操作码；imm是立即数。

可以看出，RISC-V指令对于指令的格式十分规整。相比于类似于X86架构的边长指令，RISC-V这样规整的指令更有利于译码工作。这也简化了处理器设计中对译码的要求，只需要判断指令是什么类型，就可以按照对应的比特去读取需要的数据。对于如何区别是什么类型的指令，就需要对操作码（opcode）进行判断。表2-2列举了opcode的不同维度对应的不同含义，其中就包含了区别压缩指令与其他指令、区别标准指令和扩展指令以及自定义指令和标准指令。

表2-2 RISC-V指令操作码维度,inst[1:0]=11[48]

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Inst[4:2] | 000 | 001 | 010 | 011 | 100 | 101 | 110 | 111 |
| Inst[6:5] | （>32b） |
| 00 | LOAD | LOAD-FP | custom-0 | MISC-MEM | OP-IMM | AUIPC | OP-IMM-32 | 48b |
| 01 | STORE | STORE-FP | custom-1 | AMO | OP | LUI | OP-32 | 64b |
| 10 | MADD | MSUB | NMSUB | NMADD | OP-FP | reserved | custom-2/rv128 | 48b |
| 11 | BRANCH | JALR | reserved | JAL | SYSTEM | reserved | custom-3/rv128 | *≥* 80b |

### 2.2.3 RISC-V中的P指令集扩展

数字信号处理 (DSP，Digital Signal Processing) 已成为现代电子系统的一项重要技术。 广泛的现代应用采用 DSP 算法来解决其特定领域的问题，包括传感器融合、伺服电机控制、音频解码/编码、语音合成和编码、MPEG4 解码、医学成像、计算机视觉、嵌入式控制、人机接口等。为了在DSP算法中提高运算效率，单指令多数据 (SIMD，Single instruction, multiple data) 技术被广泛应用。SIMD是 Flynn 分类法[49]中的一种并行处理。 SIMD 可以是硬件设计的一部分，可以通过指令集架构 (ISA) 直接访问。 SIMD 描述了具有多个处理元素的计算机，这些处理元素同时对多个数据点执行相同的操作。



图2-6 RISC-V 标准整数指令集与P指令集扩展运算模式对比

RISC-V P指令集扩展[50]（简称RVP）是处于草案阶段的RISC-V指令集扩展，尚未并入标准RISC-V扩展指令。RVP将RVI指定的通用寄存器复用。如图2-6所示，这些通用寄存器被分为8bit数据组、16bit数据组或32bit数据组。 这样，RVP就可以让cpu在一条指令中处理多组数据，提高并行计算性能。P 指令集扩展增加了RISC-V 架构产品的 DSP 算法处理能力。RISC-V CPU 现在可以以更低的功耗和更高的性能运行这些各种 DSP 应用程序。

本设计的SNN专用指令集就借鉴了RVP指令中SIMD的思想，同样将通用64位寄存器分为16bit、8bit数据组，以实现一条指令处理多条数据。

## 2.3 本章小结

本章对本设计的两个重要理论基础进行了详细的介绍。首先介绍了脉冲神经网络的三个基本要素：神经元模型、突触可塑性以及脉冲神经网络的拓扑结构。其中神经元模型与突触可塑性是设计SNN专用指令集扩展的主要研究对象，也是自定义指令中主要的运算的理论基础。然后本章介绍了RISC-V指令集的相关重要概念，从整体的角度介绍了指令集的组成和指令定义，阐述了RISC-V指令集架构模块化、规整化的设计理念。最后对RISC-V P指令集扩展的功能与原理做了概括，其思想正是本设计SNN扩展指令设计的基础。

# 第三章 脉冲神经网络扩展指令设计与加速模块的硬件实现

本章将主要从指令集和硬件两方面的设计来介绍脉冲神经网络扩展指令的设计和实现。这两个部分相互依赖，相互适应。指令集主要考虑脉冲神经网络中神经元模型突触可塑性算法的计算，定义了LIF模型更新、突触更新和参数配置的基本指令，并在设计时考虑硬件的可实现性；同时硬件也参考了与指令之间的联动，定义数据规范以提高运行效率。

## 3.1 脉冲神经网络扩展指令设计

脉冲神经网络扩展指令主要包含三个部分：神经元更新指令，用于更新基于LIF模型的神经元；突触学习指令，用于计算STDP以及BP-STDP突触可塑性；脉冲神经网络配置指令，对神经元的重置电位、膜时间常数、突触的学习率以及一个累加寄存器进行配置初始值。

本设计采用的操作码为表2-2中的custom-0，即opcode为0001011。根据不同的指令采用不同的Funct3和Funct7来指示。

### 3.1.1 神经元更新指令

根据第二章中LIF模型数学描述，可以得到计算神经元膜电位变化量的公式为。



考虑实际计算脉冲神经网络时与近似于固定值，因此可以继续将进行简化为的形式。



其中，为输入脉冲*S*乘以突触权重*w*后总的激励，为固定神经网络时间常数。

神经元更新指令（NUP，neuron update）格式如图3-1所示。



图3-1 神经元更新指令设计

NUP指令为R型指令。指令[31:25]为funct7，其中Funct7[7:1]为保留位，用全0表示；funct7[0]为时间戳指示位，表示该神经元模型是否带有时间戳（TS，time stamp）；指令[24:20]为源寄存器2（rs2），该寄存器内应为传入的中输入神经元的总激励；指令[19:15]为源寄存器1（rs1），该寄存器传入的是神经元的当前膜电位*V*；指令[14:12]为funct3，用于指示这是NUP指令。

当时间戳指示位为0时，表示当前更新神经元不包含时间戳部分，即64bit寄存器中包含了4个16bit待更新神经元的膜电位，且膜电位为定点数；当时间戳指示位为1时，表示当前更新神经元带时间戳，64bit寄存器4个神经元，每个神经元前8bit为时间戳位，后8bit为待更新神经元膜电位。

### 3.1.2 突触学习与通用运算指令

1）时间依赖规则指令（TDR，time dependence rule）。

该指令用于计算两个带有时间戳神经元数据的发放时间差，并同时计算8个神经元之间的时间戳。指令定义如图3-2所示。



图3-2 时间依赖规则指令设计

TDR指令为R型指令，指令中funct7保留，作全0处理；源寄存器rs2和源寄存器rs1分别载入了4个16bit的神经元数据，每个16bit的神经元数据高8位为时间戳数据，低八位为神经元膜电位数据；funct3指示TDR指令。

通过TDR指令计算出的结果可以用作中。与RISC-V P扩展中SUB8指令不同的是，TDR指令只会对神经元数据中时间戳部分进行读取，且最后得到的结果只包含了通过时间戳计算出的结果。

2）反向传播输出方向参数计算指令（BPO，back-propagation output direction parameter）。

BPO指令用于加速BP-STDP计算（）中对突触更新的方向参数的计算过程。一条BPO指令更新四条突触更新的方向参数。指令定义如图3-3所示。



图3-3 反向传播输出方向参数计算指令设计

指令中funct7为0000001，与funct3共同指示该指令为BPO指令；源寄存器rs2载入掩码，用于指示目标神经元位置；源寄存器rs1载入了4个神经元发放数据。根据rs1与rs2以及，计算出每个神经元对应的。

3）指数函数计算指令（EXP，exponential function calculation）。

在本毕设的设计中，支持16bit定点数的指数函数求解。一条EXP指令将同时计算4个不同的指数函数求解。指令定义如图3-4所示。



图3-4 指数函数计算指令设计

指令中funct7、源寄存器rs2保留，做全0处理；rs1载入了4个16bit的定点数，用于计算对应的指数函数值；funct3指示该指令为EXP指令。

在计算EXP时，采用了cordic算法来作为硬件实现，并对该算法的收敛域进行了扩展。理论上在计算指数函数时，调用EXP指令相对于调用库函数中的exp计算函数要减少周期数5-10倍。但是与此同时缺点是精度并不如调用库函数使用浮点数。具体的硬件实现方法会在3.5节详细讨论。

4）16bit求和指令（SUM16，16bit sum）。

SUM指令用于计算一个寄存器内4个16bit值的和，并根据对应的掩码进行选择。该指令的定义如图3-5所示。



图3-5 16bit求和指令设计

该指令的funct7[7:1]为保留位，funct7[0]为累加寄存器指示位acc\_flag：当acc\_flag为1时，计算时需要根据源寄存器rs2中的掩码选出源寄存器rs1中的4个16bit数参与计算的数据，将其累加后再加上累加寄存器中的值作为结果，并存回累加寄存器；若为0，则只累加源根据源寄存器rs2中的掩码选出源寄存器rs1中的4个16bit数参与计算的数据，并返回结果，不保存到累加寄存器中。指令中funct3指示该指令为SUM16指令。

SUM16指令不光可以单独对16bit数据组进行求和，也可以参与到突触可塑性算法的计算中。根据与，都需要对相应数据进行累加，因此在保证基础运算的情况下也能支持突触可塑性算法的计算。

### 3.1.3 脉冲神经网络配置指令

在脉冲神经网络中神经元与突触可塑性的计算中有很多重复使用的参数，根据、以及，可以总结出以下三个主要的参数：神经元重置膜电位；神经元更新时间常数；突触可塑性学习率。因此可以在脉冲神经网络计算单元中定义寄存器组，分别保存对应的神经网络参数。为了能初始化神经网络参数寄存器，本设计定义了以下脉冲神经网络配置指令。

1）网络时间常数设置指令（STAU，set tau）。

STAU指令用于设置神经元更新时使用的时间常数寄存器初始值。根据，在神经元更新时会从储存有时间常数的寄存器中读取参数并使用。时间常数为一个16bit定点数。该指令的指令定义如图3-6所示。



图3-6 网络时间常数设置指令设计

指令中funct7与funct3共同指示该指令为STAU指令；源寄存器rs2位保留，全0处理；源寄存器rs1中载入的是网络时间常数；目标寄存器默认都是SNN运算单元的SNN专用寄存器组。

2）突触可塑性学习率设置指令（SLR，set learning rate）。

SLR用于设置突触可塑性学习率寄存器初始值。根据与，使用BP-STDP突触可塑性算法时会从储存有学习率的寄存器中国读取参数并使用。突触可塑性学习率是一个16bit定点数。指令定义如图3-7所示。



图3-7 突触可塑性学习率设置指令设计

指令中funct7与funct3共同指示该指令为SLR指令；源寄存器rs2位保留，全0处理；源寄存器rs1中载入的是突触可塑性学习率 ；目标寄存器默认都是SNN运算单元的SNN专用寄存器组。

3）神经元重置膜电位设置指令（SVR，set reset voltage）。

SVR用于设置神经元重置膜电位寄存器初始值。根据，在神经元更新时会从储存有神经元重置膜电位的寄存器中读取参数并使用。神经元重置膜电位为一个16bit定点数。指令定义如图3-8所示。



图3-8 神经元重置膜电位设置指令设计

指令中funct7与funct3共同指示该指令为SVR指令；源寄存器rs2位保留，全0处理；源寄存器rs1中载入的是神经元重置膜电位 ；目标寄存器默认都是SNN运算单元的SNN专用寄存器组。

4）累加寄存器初始值设置指令（SACC，set accumulation register）。

SACC指令用于设置SNN计算单元中累加寄存器的初始值。指令设计如图3-9所示。



图3-9

指令中funct7与funct3共同指示该指令为SACC指令；源寄存器rs2位保留，全0处理；源寄存器rs1中载入的是累加寄存器初始值；目标寄存器默认都是SNN运算单元的SNN专用寄存器组。

累加寄存器主要用于和累加指令SUM16配合计算。SUM16中如果acc\_flag设置为1，则将累加寄存器的值作为第三操作数参与到累加计算中。

## 3.2 脉冲神经网络加速模块整体设计

本设计中的脉冲神经网络加速模块（SNNU，spiking neural network unit）是作为一个执行级的运算单元嵌入到一个基于RISC-V指令集架构的双发射顺序处理器“Polaris”[51]中，运算单元整体设计如图3-10所示。



图3-10 脉冲神经网络加速模块整体设计

SNNU可以分为3个部分。其中各个模块为：SNN发射单元（SISU， snn issue unit）负责对传入的指令进行进一步解码，根据指令分类与需要发射到对应的运算单元；LIF神经元更新单元（LNU，LIF neuron unit），负责LIF神经元的更新；突触计算单元（SU，synapse unit），负责突触可塑性相关计算过程，包括累加、求自然指数函数以及BP-STDP相关计算。

SNNU被作为一个运算器增加到了处理器流水线中。为了不阻塞流水线，不影响除了SNN扩展指令以外的其他指令工作，SNNU也采用了流水线设计。整个部分分为两级流水，首先由ISU作为第一级流水线，将收到的译码信息以及指令进一步分析，按照对应功能发送到相应的部件；流水线第二级由LNU以及SU组成，在完成运算后将结果经过一个选择器输出到处理器流水线的提交（commit）级。

## 3.3 发射单元设计

发射单元负责对收到的译码信息以及指令进一步分析，并按照功能区分将指令送往不同的计算模块。ISU的模块设计如图3-11所示。



图3-11 发射模块设计

发射模块由一个操作查找控制模块，SNN寄存器组（SNNRF，SNN register file）以及一个信号整合模块组成。指令信息进入ISU后，先进入操作查找控制模块匹配对应的操作，指令中神经元相关指令（3.1.1 神经元更新指令）匹配LNU模块；突触及通用计算指令（3.1.2 突触学习与通用运算指令）匹配SU模块；SNNRF配置相关指令（3.1.3 脉冲神经网络配置指令）直接匹配ISU。匹配成功后产生信号，如果是匹配的ISU操作，则流水仅由ISU完成，不再进入流水线第二级；若匹配LNU或SU，则匹配后信号进入信号整合模块，与SNNRF中数据一并组合后通过SCtrl总线进入流水线第二级。

信号整合模块将传入SNNU的操作数按照4个16bit的形式对源操作数进行划分，随后将从操作查找模块发出的匹配信号以及SNNRF的数据一并打包后通过SCtrl总线传出。SNNRF的数据只作用于SNNU模块内部，不参与除SNN扩展指令以外的其他指令运算；SCtrl总线是一类在SNNU中各个部件之间通信的专用总线，包含了每条指令的操作类型信号、匹配信号、向量化的操作数、寄存器访问值以及累加值的反馈。

## 3.4 LIF神经元更新单元设计

LIF神经元更新单元接受通过SCtrl总线传入的信号，并根据对4个LIF模型神经元同时进行更新。LNU的硬件设计如图3-12所示。



图3-12 LIF神经元更新单元设计

图3-12中NUx[t]为t时刻第x个神经元的膜电位数据；vr为重置电位寄存器中的值；swx为第x个输入神经元的激励值；Tau为网络时间常数寄存器的值。

经过图3-12中的计算流程后，得到每个神经元更新后的膜电位，并根据是否带有时间戳组合成为最后的结果：如果带有时间戳，则将原本的时间戳数据与更新后神经元膜电位组合后形成64bit的结果；如果不带时间戳，则直接组合神经元膜电位得到64bit结果。

因为超过两个操作数的指令在硬件实现上比较困难，大多RISC-V架构的指令只支持至多2个源操作数。因此本设计在指令设计中仍采用两个操作时，硬件设计时通过提取出计算过程中常用的不变量，在SNNU内部设立寄存器组来储存这些不变量，从而达到在一个指令中隐式地使用多个操作数。

## 3.5 突触与通用计算单元设计

突触与通用计算单元主要用于参与突触可塑性相关计算，也可以通过调用SNN扩展指令中EXP指令和SUM16指令实现只进行部分计算。SU模块架构如图3-13所示。当指令与SU匹配，数据会经过第一级流水ISU后通过SCtrl传入SU模块。在SU模块中，主要包括了BPO指令计算模块、累加指令计算模块以及向前传播计算模块。其中，向前传播计算模块用于计算STDP突触可塑性算法，包括四个计算指数函数的计算单元Exp和一个时间戳计算单元。每条指令的计算结果都会以64bit的数据输出到处理器的下一级流水。

下面将分别对每个部分进行介绍。



图3-13 突触与通用计算单元设计

1）BPO计算模块：BPO计算模块中，主要应用了中对的计算逻辑：当目标神经元未在规定的发放窗口中发放，则对应输出神经元与前一层神经元之间的突触可塑性方向参数；当非目标神经元在窗口中发放，则；若为其他情况，则。

2）SUM计算模块：Sum计算模块根据源操作数中的掩码对指定的数据进行累加操作，同时根据指令中累加寄存器指示位acc\_flag来判断是否需要SNNRF中ACC寄存器中的值，若需要，则结果不光输出到处理器下一流水，还要更新SNNRF中ACC寄存器的值；若不需要，则直接输出到处理器下一流水中。

3）向前传播计算模块：在向前传播计算中，时间戳计算单元仅对传入的神经元时间戳数据进行计算，得到时间间隔，用于后续突触可塑性的计算，不会计算神经元膜电位数据。

指数函数计算单元通过使用坐标旋转数字计算机算法[52]（CORDIC，Coordinate Rotation Digital Computer）计算16bit定点数的指数函数值。CORDIC算法拥有多种坐标系统以及计算模式，其中圆周系统可用于计算三角函数、向量旋转等；双曲系统可用于计算双曲函数、指数函数等。

在SU中指数函数单元使用了CORDIC算法的双曲系统。CORDIC算法根据极坐标下不断进行矢量旋转逼近目标角度来实现近似求解。如图3-14所示，假设向量与*x*轴夹角为，P点坐标可以表示为





图3-14 CORDIC双曲系统矢量旋转图

将向量旋转到向量，此时与坐标*x*轴夹角为，故Q点坐标公式为



将带入后再进行化简可以得到



其中可以分解为一系列的线性组合：



其中，*i*从1开始。结合、可以得到如的CORDIC的n次迭代输出公式



根据可以看出，每一次CORDIC算法的迭代都将其中的乘法转变为了移位计算，因此可以设计出高效的数字电路来实现这一算法。

但是，根据也可以得到CORDIC算法的收敛范围为所示



换言之，当时，CORDIC的结果将无法被正确的表示。因此，SU模块的EXP单元使用了基于区间压缩法的收敛域扩展技术[53]。该技术将求解关于的指数函数时，将其转化为如所示的形式



这样可以先计算关于*y*的指数函数（*y* < 1.1182），再将其结果左移|Q|位得到最终关于的指数函数的值。

在本设计中，指数函数计算单元使用了16bit定点数，通过流水线与查找表技术，选取适当的初始值，先求解sinh与cosh双曲函数，再根据来计算指数函数的值。

## 3.6 本章小结

本章首先详细介绍了脉冲神经网络扩展指令的设计。通过将指令集分为三个部分，分门别类地介绍了功能不同的自定义指令。每一条指令的编码都参考了RISC-V指令集的定义规范。并为了防止一条指令计算过长导致占用流水线，本设计根据神经元与突触计算的数学表达提取出重要的计算步骤作为指令，以提高处理器的吞吐量以及灵活性。随后根据指令的设计继续介绍了对应功能的硬件实现，从脉冲神经网络加速模块的整体到各个部分的设计自顶向下说明了每一个部分的功能与实现架构。其中重点说明了如何在突触单元中使用CORDIC算法和基于区间压缩法的收敛域扩展技术实现指数函数计算单元。

指令设计与硬件是相辅相成的。在指令集设计的过程中需要考虑硬件实现的难度以及运算效率，同时设计好的指令是硬件设计的技术标准与功能参考。在本设计的指令设计中充分考虑了硬件实现的可行性、指令运算的效率与指令使用的灵活性，实现了软硬件协同设计。

# 第四章 功能验证与性能分析

本章将主要讨论实现RTL（Register Transfer Level）代码后如何对设计进行功能验证与性能分析。其中功能验证包括软件模拟器平台验证和硬件FPGA验证。随后将对通过硬件验证后的数据进行加速性能的结果分析。

## 4.1 功能验证

在实现了RTL级描述的架构设计后，需要对每一条指令的正确性进行测试。首先需要在软件仿真环境下验证每条指令功能的正确性，然后再将设计部署到FPGA上进行硬件真实环境的验证。

### 4.1.1 软件测试平台功能验证

**1）软件测试平台概述**

软件平台测试的主要过程如图4-1所示。



图4-1 软件平台测试过程

在待验证设计一侧，有以下几个特点：1）Verilator可以将RTL级设计编译为基于C++实现的周期精确模型。通过使用Verilator编译后的模型可以被C++程序引用，最后被g++编译器编译为可执行模拟器emu。2）为验证指令在处理器中执行的正确性，在AM环境C语言编写的测试程序通过内联汇编引入脉冲神经网络扩展指令。因为内联汇编不必修改编译器，所以被广泛用于自定义指令扩展的软件实现中。图4-2展示了本设计中神经网络配置指令嵌入到C程序中的具体表现形式。3）由于需要对通用寄存器中数据进行分组（64bit分为4\*16bit），为了方便程序中数据声明，头文件定义了两类向量类型定义，如图4-2中所示。4）使用随机数值进行测试。在参考模型一侧，使用C语言实现精确的、包含有扩展指令对应功能的参考模型，使用GCC编译后形成可执行文件，输入与待验证一侧相同输入后得到可供参考的输出。



图4-2 测试程序内联汇编定义与汇编代码中的SNN扩展指令

**2）测试结果**

最终部分软件平台测试结果（指令NUP）如图4-3所示。与此同时，为了验证本设计对原处理器基本RISC-V指令没有影响，软件还测试了一系列基本指令测试[[3]](#footnote-3)以及启动操作系统的测试，结果如图4-4与图4-5所示。

文本

描述已自动生成

图4-3 软件功能测试结果图

 

图4-4 处理器成功启动Linux内核

 文本

描述已自动生成

图4-5处理器成功启动Linux内核与RTthread OS

总结来说，为了验证脉冲神经网络的功能的正确性，本设计首先使用Verilator开源仿真器将处理器设计编译为一个可执行的模拟器，再通过内联汇编的方式将扩展指令引入程序中，通过与参考模型的对比，实现了对其功能正确性验证。此外，软件平台验证还保证了脉冲神经网络扩展指令没有对其他处理器功能产生影响，这也是对除扩展指令之外的其他指令的功能正确性验证。

### 4.1.2 硬件FPGA测试平台的功能验证

本文使用的FPGA平台为Alveo U250 Xilinx UltraScale+ FPGA，包含1728K查找表（LUT，look up table）以及3456K寄存器资源，DDR总容量为64G，最大带宽为77GB/s。为了在FPGA上运行测试程序，需要一个合适的集成环境以便于控制。本设计使用的集成环境[56]包括了：DDR4 DRAM用于处理器访问和储存需要运行的程序；JTAG调试接口，用于传输程序；System ILA（Integrated Logic Analyzer）直接在vivado工程中查看信号与波形；UART串口设备，用于输出屏幕信息；时钟模块，产生300MHz的时钟频率；虚拟复位，用于复位处理器。以上介绍的集成环境如图4-6所示。



图4-6 FPGA平台集成开发环境结构示意图

在实际使用过程中，需要先在本地设置好生成处理器的配置（如设备地址、复位向量等），生成对应的Verilog文件并加入到vivado工程中。随后在本地编译好的程序使用JTAG接口传输到DDR4 DRAM中。在FPGA平台的逻辑设计中，包括了jtag\_axi的IP，因此可以通过图4-6中AXI接口向DRAM请求写事务。最后在调试终端打开串口，设置波特率为115200，并通过vivado向虚拟复位设备发送复位请求，串口即可输出信息，如图4-7所示的为NUP指令测试程序。

文本

中度可信度描述已自动生成

图4-7 在FPGA运行与软件平台相同的NUP指令测试程序

## 4.2性能与加速结果分析

在4.1.2节中详细介绍了FPGA平台的搭建，通过使用FPGA平台以及搭配的Vivado工程可以得到处理器性能的相关数据。如图4-8所示，图中高亮部分为本设计加速模块的硬件资源占用量，查找表（LUT）资源使用为4370，寄存器资源使用1936。图4-9展示了本设计加速模块的功耗。

表格, 日历

中度可信度描述已自动生成

图4-8 加速模块在Vivado中显示的资源使用情况

表格

描述已自动生成

图4-9 加速模块在Vivado中显示的功耗情况

为了展示本设计相较于“加速器+控制核解决”方案的优势，我们选择了“‘ODIN[12]’加速器+‘Polaris’处理器[51]”的方案，用ODIN实现脉冲神经网络计算，“Polaris”处理器作为控制核。得到如表4-1的资源使用数据对比。从表中数据可以得到，本设计加速模块相较于ODIN使用的查找表资源减少了46.8%，寄存器资源减少了67.5%。

表4-1 本设计与加速器+控制核解决方案的资源使用对比

|  |  |  |  |
| --- | --- | --- | --- |
| 名称 | 平台 | LUT | FF |
| ODIN | Alveo U250 | 8212 | 5962 |
| 本设计（snnu） | Alveo U250 | 4370 | 1936 |

在功耗上，我们对比两种解决方案的整体功耗，可以得到表4-2。从表中也可以看出，本设计提出的扩展指令解决方案可以大幅度减小功耗。其中总功耗仅为加速器+控制核心解决方案功耗的21.1%，动态功耗为加速器+控制核心解决方案的11.1%。

表4-2 本设计与加速器+控制核解决方案的功耗对比

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 名称 | 平台 | 总功耗 | 静态功耗 | 动态功耗 |
| Polaris + ODIN | Alveo U250 | 25.949 W | 3.488 W | 22.461 W |
| 本设计 | Alveo U250 | 5.477 W | 2.992 W | 2.484 W |

在运算性能上，主要对运算提升的指令为NUP、SUM16、EXP、TDR和BPO。这些指令与对应功能的C程序实现性能对比如表4-3所示。

表4-3 主要加速指令与对应C语言实现计算性能对比

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| 指令 | 理论周期数 | 对应C实现指令数 | 对应C实现周期数 | 性能提升 |
| SUM16 | 2 | 12 | 12-20 | 83.3%-90% |
| EXP | 10 | 58 | 60-240 | 83.3%-95.8% |
| NUP | 2 | 13 | 13-52 | 84.6%-96.1% |
| TDR | 3 | 4 | 4-12 | 25%-75% |
| BPO | 1 | 35 | 35-60 | 97.1%-98.3% |
| 平均性能提升 |  |  |  | 74.7%-91.0% |

从表中可以看出，有了扩展指令后，实现其对应的功能所带来的新跟那个提升平均为74.7%到91.0%，这使得处理器在脉冲神经网络计算场景下的计算性能得到了大幅的增强。

从上述硬件资源利用率、功耗以及计算表现来看，通过集成脉冲神经网络扩展指令与加速模块到一个双发射顺序处理器中，在相同硬件平台上相较于传统“加速器+控制核”的硬件资源使用与功耗上有显著提升；同时，相较于C语言实现的脉冲神经网络计算相关的程序，在相同运行平台上，调用脉冲神经网络扩展指令能够将计算性能提升74.7%-91.0%。

## 4.3 本章小结

本章主要阐述了如何对本设计的最终电路进行基于软硬件平台的功能验证。其中基于软件平台的功能验证主要使用了Verilator仿真器对从Chisel转化而来的Verilog文件进行编译并得到可运行的模拟器文件，随后再基于AM平台交叉编译可执行文件，使用内联汇编的方式引用扩展指令，最后将使用相同输入得到的模拟器输出与使用C语言构建的标准功能模型进行比对从而实现功能验证。同时为保证扩展指令与加速模块在处理器中并未影响其他指令的使用，在软件平台上同时也进行了多种标准指令的基准测试，包括Coremark、Microbench、Linux内核以及RTthread OS的启动。在基于硬件平台的功能验证中，本设计利用了包含集成开发环境的Vivado工程来进行功能验证，将在软件平台上运行的程序放入FPGA的DRAM中，使用处理器来访问并运行，得到与软件平台上一致的结果。同时，在硬件实际测试时得到了关于本设计的一系列性能数据，包括脉冲神经网络加速模块FPGA硬件资源使用情况，功耗以及计算性能。并得到了最终结论：通过集成脉冲神经网络扩展指令与加速模块到一个双发射顺序处理器中，在相同硬件平台上相较于传统“加速器+控制核”的硬件资源使用与功耗上有显著提升；且相较于C语言实现的脉冲神经网络计算相关的程序，在相同运行平台上，调用脉冲神经网络扩展指令能够将计算性能提升74.7%-91.0%。

# 第五章 全文总结与展望

随着人工智能的不断发展，人工智能的应用场景越来越广泛。人工神经网络作为人工智能重要的部分也在时代的浪潮下被提出更高的要求。越来越多的场景需要人工神经网络展现出追求高算力、低功耗的能力。脉冲神经网络作为第三代人工神经网络正具备了这样的能力。通过模拟生物大脑的工作模式，利用稀疏性脉冲作为神经网络中的信息传递载体，实现了低功耗的特性；通过高度并行化的计算，脉冲神经网络同其他神经网络一样可以实现高算力的特性。

目前主流的脉冲神经网络加速方案是由一个脉冲神经网络加速器外加一个控制核组成计算系统，但是这样的系统由于存在的数据搬运、指令架构差异等问题无法完全发挥脉冲神经网络的高算力和低功耗特性。因此，本设计通过设计基于RISC-V P扩展指令集的脉冲神经网络扩展指令，将一个可以被此扩展指令调用的脉冲神经网络加速模块加入到了RISC-V双发射顺序处理器中，通过紧耦合的方式解决了“加速器+控制核”方案存在的部分问题。

本设计的主要工作可以概括为以下三点：

1）在对扩展指令设计的过程中，充分考虑扩展指令在计算时的效率问题，在兼顾指令完备性的前提下，提取出在脉冲神经网络计算中高度重复的计算过程将其转化为指令，并提供对特殊计算的加速（指数函数与元素求和），从而在处理器中实现SIMD计算，提升了处理器的运算效率。

2）在硬件实现的过程中，本设计使用了敏捷开发语言Chisel HDL，利用其高级语言的敏捷特性（函数式编程与面向对象的特性），大大简化了硬件开发的难度；对硬件进行仿真验证的过程中，使用了开源硬件仿真器Verilator编译出可运行的模拟器文件，并使用AM平台交叉编译可以在此模拟器上运行的程序，其结果与C语言实现的参考模型进行对比，实现基于软件的功能验证。

3）在硬件平台Alveo U250 Xilinx UltraScale+ FPGA上，通过一个集成开发环境来实现了FPGA上的功能测试，可以通过简单的操作对处理器进行调试并获得相关数据。

综上所述，根据在硬件平台上实际测试得到的性能指标，本文可以得到如下结论：通过集成脉冲神经网络扩展指令与加速模块到一个双发射顺序处理器中，在相同硬件平台上相较于传统“加速器+控制核”的硬件资源使用与功耗上有显著提升；且相较于C语言实现的脉冲神经网络计算相关的程序，在相同运行平台上，调用脉冲神经网络扩展指令能够将计算性能提升74.7%-91.0%。

在本设计的工作中存在值得改进的地方。首先，对于将通用寄存器划分为几个数据段打包为一组数据从而实现SIMD计算，可以进一步改进为加入向量寄存器，增加数据的吞吐量，且与通用寄存器隔离开来。其次，在设计验证上，本设计没有进行后端综合，仅仅是Vivado工程中的资源利用率数据很难估算出更精确的芯片面积及功耗指标。最后，本设计中的处理器单核性能是有限的，可以通过使用片上网络技术（NoC，network on chip）来实现多核设计，提升处理器的计算理论极限。

# 参考文献

1. Radford Alec, Narasimhan Karthik, Salimans Tim et al. Improving language understanding by generative pre-training [J]. 2018.
2. Maass Wolfgang. Networks of spiking neurons: the third generation of neural network models [J].Neural networks. 10 (9). 1997: 1659–1671.
3. Jo Sung Hyun, Chang Ting, Ebong Idongesit et al. Nanoscale memristor device as synapse in neuromorphic systems [J]. Nano letters. 10 (4). 2010: 1297–1301.
4. Cai Fuxi, Correll Justin M, Lee Seung Hwan et al. A fully integrated reprogrammable memristor–CMOS system for efficient multiply–accumulate operations [J]. Nature Electronics. 2 (7). 2019: 290–299.
5. Chen Wei-Hao, Dou Chunmeng, Li Kai-Xiang et al. CMOS-integrated memristive non-volatile computing-in-memory for AI edge processors [J]. Nature Electronics. 2 (9). 2019: 420–428.
6. Nurse Ewan, Mashford Benjamin S, Yepes Antonio Jimeno et al. Decoding EEG and LFP signals using deep learning: heading TrueNorth [C]. In Proceedings of the ACM international conference on computing frontiers. 2016 : 259–266.
7. Davies Mike, Srinivasa Narayan, Lin Tsung-Han et al. Loihi: A neuromorphic manycore processor with on-chip learning [J]. Ieee Micro. 38 (1). 2018: 82–99.
8. Tang Guangzhi, Shah Arpit, Michmizos Konstantinos P. Spiking neural network on neuromorphic hardware for energy-efficient unidimensional slam [C]. In 2019 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS). 2019 : 4176–4181.
9. Orchard Garrick, Frady E Paxon, Rubin Daniel Ben Dayan et al. Efficient neuromorphic signal processing with loihi 2 [C]. In 2021 IEEE Workshop on Signal Processing Systems (SiPS). 2021 : 254–259.
10. Pei Jing, Deng Lei, Song Sen et al. Towards artificial general intelligence with hybrid Tianjic chip architecture [J]. Nature. 572 (7767). 2019: 106–111.
11. Wang Jiulong, Wu Ruopu, Chen Guokai et al. RISC-V Toolchain and Agile Development based Opensource Neuromorphic Processor [J]. arXiv preprint arXiv:2210.00562. 2022.
12. Frenkel Charlotte, Lefebvre Martin, Legat Jean-Didier et al. A 0.086-mm ^212.7-pJ/SOP 64k-synapse 256-neuron online-learning digital spiking neuromorphic processor in 28-nm CMOS [J]. IEEE transactions on biomedical circuits and systems. 13 (1). 2018: 145–158.
13. Izhikevich Eugene M. Simple model of spiking neurons [J]. IEEE Transactions on neural networks. 14 (6). 2003: 1569–1572.
14. Spitale Andrea. Interfacing a Neuromorphic Coprocessor with a RISC-V Architecture [Dissertation]. Politecnico di Torino, 2021
15. Mayr Christian, Hoeppner Sebastian, Furber Steve. Spinnaker 2: A 10 million core processor system for brain simulation and machine learning [J]. arXiv preprint arXiv:1911.02385. 2019.
16. Waterman Andrew, Lee Yunsup, Patterson David A et al. The risc-v instruction set manual. volume 1: User-level isa, version 2.0 [R]. 2014.
17. Lee Yunsup, Zimmer Brian, Waterman Andrew et al. Raven: A 28nm RISC-V vector processor with integrated switched-capacitor DC-DC converters and adaptive clocking [C]. In 2015 IEEE Hot Chips 27 Symposium (HCS). 2015 : 1–45.
18. CHIPS Alliance. Rocket Chip Generator [EB/OL]. <https://github.com/chipsalliance/rocket-chip>.
19. Celio Christopher Patrick. A highly productive implementation of an out-of-order processor generator [M]. University of California, Berkeley, 2017.
20. Chen Chen, Xiang Xiaoyan, Liu Chang et al. Xuantie-910: A commercial multi-core 12-stage pipeline out-of-order 64-bit high performance RISC-V processor with vector extension: Industrial product [C]. In 2020 ACM/IEEE 47th Annual International Symposium on Computer Architecture (ISCA). 2020 :52–64.
21. Xu Yinan, Yu Zihao, Tang Dan et al. Towards Developing High Performance RISC-V Processors Using Agile Methodology [C]. In 2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO). 2022 : 1178–1199.
22. Tagliavini Giuseppe, Mach Stefan, Rossi Davide et al. Design and evaluation of SmallFloat SIMD extensions to the RISC-V ISA [C]. In 2019 Design, Automation & Test in Europe Conference & Exhibition (DATE). 2019 : 654–657.
23. Cococcioni Marco, Rossi Federico, Ruffaldi Emanuele et al. A lightweight posit processing unit for risc-v processors in deep neural network applications [J]. IEEE Transactions on Emerging Topics in Computing. 10 (4). 2021: 1898–1908.
24. Wang Shihang, Zhu Jianghan, Wang Qi et al. Customized instruction on risc-v for winograd-based convolution acceleration [C]. In 2021 IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP). 2021 : 65–68.
25. Jiao Qiang, Hu Wei, Liu Fang et al. RISC-VTF: RISC-V Based Extended Instruction Set for Transformer [C]. In 2021 IEEE International Conference on Systems, Man, and Cybernetics (SMC). 2021 : 1565–1570.
26. Razilov Viktor, Matúš Emil, Fettweis Gerhard. Communications Signal Processing Using RISC-V Vector Extension [C]. In 2022 International Wireless Communications and Mobile Computing (IWCMC).2022 : 690–695.
27. Tourres Mael, Chavet Cyrille, Le Gal Bertrand et al. Extended RISC-V hardware architecture for future digital communication systems [C]. In 2021 IEEE 4th 5G World Forum (5GWF). 2021 : 224–229.
28. Andri Renzo, Henriksson Tomas, Benini Luca. Extending the RISC-V ISA for efficient RNN-based 5G radio resource management [C]. In 2020 57th ACM/IEEE Design Automation Conference (DAC). 2020 : 1–6.
29. Paulin Gianna, Andri Renzo, Conti Francesco et al. RNN-based radio resource management on multicore RISC-V accelerator architectures [J]. IEEE Transactions on Very Large Scale Integration (VLSI) Systems. 29 (9). 2021: 1624–1637.
30. Bachrach Jonathan, Vo Huy, Richards Brian et al. Chisel: constructing hardware in a scala embedded language [C]. In Proceedings of the 49th Annual Design Automation Conference. 2012 : 1216–1225.
31. Snyder Wilson. Verilator and systemperl [C]. In North American SystemC Users’Group, Design Automation Conference. 2004 .
32. NJU-projectN. Abstract Machine: A minimal, modularized, and machine-independent abstraction layer of the computer hardware [EB/OL]. <https://github.com/NJU-ProjectN/abstract-machine>.
33. Hodgkin Alan L, Huxley Andrew F. A quantitative description of membrane current and its application to conduction and excitation in nerve [J]. The Journal of physiology. 117 (4). 1952: 500.
34. Indiveri Giacomo, Stefanini Fabio, Chicca Elisabetta. Spike-based learning with a generalized integrate and fire silicon neuron [C]. In Proceedings of 2010 IEEE International Symposium on Circuits and Systems. 2010 : 1951–1954.
35. Fourcaud-Trocmé Nicolas, Hansel David, Van Vreeswijk Carl et al. How spike generation mechanisms determine the neuronal response to fluctuating inputs [J]. Journal of neuroscience. 23 (37). 2003: 11628–11640.
36. Brunel Nicolas, Latham Peter E. Firing rate of the noisy quadratic integrate-and-fire neuron [J]. Neural computation. 15 (10). 2003: 2281–2306.
37. Bain Alexander. Mind and body: The theories of their relation [M]. Henry S. King & Company, 1873.
38. Hebb Donald Olding. The organization of behavior: A neuropsychological theory [M]. Psychology press, 2005.
39. Markram Henry, Gerstner Wulfram, Sjöström Per Jesper. A history of spike-timing-dependent plasticity [J]. Frontiers in synaptic neuroscience. 3. 2011: 4.
40. Kress Geraldine J, Mennerick Steven. Action potential initiation and propagation: upstream influences on neurotransmission [J]. Neuroscience. 158 (1). 2009: 211–222.
41. Markram Henry, Lübke Joachim, Frotscher Michael et al. Regulation of synaptic efficacy by coincidence of postsynaptic APs and EPSPs [J]. Science. 275 (5297). 1997: 213–215.
42. Cassidy Andrew, Andreou Andreas G, Georgiou Julius. A combinational digital logic approach to STDP [C]. In 2011 IEEE international Symposium of Circuits and Systems (ISCAS). 2011 : 673–676.
43. Brader Joseph M, Senn Walter, Fusi Stefano. Learning real-world stimuli in a neural network with spike-driven synaptic dynamics [J]. Neural computation. 19 (11). 2007: 2881–2912.
44. Bohte Sander M. Error-backpropagation in networks of fractionally predictive spiking neurons [C]. In Artificial Neural Networks and Machine Learning–ICANN 2011: 21st International Conference on Artificial Neural Networks, Espoo, Finland, June 14-17, 2011, Proceedings, Part I 21. 2011 : 60–68.
45. Zenke Friedemann, Ganguli Surya. Superspike: Supervised learning in multilayer spiking neural networks [J]. Neural computation. 30 (6). 2018: 1514–1541.
46. Neftci Emre O, Mostafa Hesham, Zenke Friedemann. Surrogate gradient learning in spiking neural networks: Bringing the power of gradient-based optimization to spiking neural networks [J]. IEEE Signal Processing Magazine. 36 (6). 2019: 51–63.
47. Tavanaei Amirhossein, Maida Anthony. BP-STDP: Approximating backpropagation using spike timing dependent plasticity [J]. Neurocomputing. 330. 2019: 39–47.
48. Waterman A, Lee Y, Patterson D A, et al. The risc-v instruction set manual. volume 1: User-level isa, version 2.0[R]. California Univ Berkeley Dept of Electrical Engineering and Computer Sciences, 2014.
49. Flynn Michael J. Some computer organizations and their effectiveness [J]. IEEE transactions on computers. 100 (9). 1972: 948–960.
50. Yang Chun-Chieh, Chen Yi-Ru, Liao Hui-Hsin et al. Auto-tuning Fixed-point Precision with TVM on RISC-V Packed SIMD Extension [J]. ACM Transactions on Design Automation of Electronic Systems. 28 (3). 2023: 1–21.
51. Jixiang Zong. Polaris22A: RISC-V SIMD Superscalar Dual-Issue Processor [EB/OL]. <https://github.com/ByeBeihai/Polaris>.
52. Volder Jack. The CORDIC computing technique [C]. In Papers presented at the the March 3-5, 1959, western joint computer conference. 1959 : 257–261.
53. 唐文明, 刘桂雄. 指数函数 CORDIC 算法的 FPGA 定点化技术 [J]. 华南理工大学学报: 自然科学版. 44 (7). 2016: 9–14.
54. Ebedded Microprocessor Benchmark Consortium. CoreMark [EB/OL]. <https://github.com/eembc/coremark>.
55. RT-Thread. RT-Thread: an open source IoT real-time operating system (RTOS) [EB/OL]. <https://github.com/RT-Thread/rt-thread>.
56. Chen Xuhao. NutShell\_U25 [EB/OL]. <https://github.com/ssdfghhhhhhh/NutShell_U250>.

1. GSOPS/W，Giga Synaptic OptionS per Watt，每瓦特突触操作 [↑](#footnote-ref-1)
2. G 表示包括I、M、A、F、D、Zicsr、Zifencei基础指令；C表示压缩指令 [↑](#footnote-ref-2)
3. 测试包括coremark[54]和microbench。 [↑](#footnote-ref-3)