硕士学位论文

（专业学位论文）

面向物联网应用的 RISC-V 架构 SoC 设计与实现

DESIGN AND IMPLEMENTATION OF RISC-V ARCHITECTURE SOC FOR INTERNET OF THINGS APPLICATIONS

李孟晗

哈尔滨工业大学

2023 年 5 月

国内图书分类号：TN4　　学校代码：10213
国际图书分类号：621.3　　密级：公开

硕士学位论文

面向物联网应用的 RISC-V 架构 SoC 设计与实现

硕士研究生：李孟晗
导　　师：王新胜 副教授
申请学位：专业学位
学　　科：集成电路工程
所在单位：信息科学与工程学院
答辩日期：2023 年 5 月
授予学位单位：哈尔滨工业大学

Classified Index: TN4
U.D.C: 621.3

Dissertation for the Master Degree in Engineering

DESIGN AND IMPLEMENTATION OF RISC-V ARCHITECTURE SOC FOR INTERNET OF THINGS APPLICATIONS

Candidate： Li Menghan
Supervisor： Associate Prof. Wang Xinsheng
Degree Applied for： Professional Degrees
Speciality： Integrated Circuit Engineering
Affiliation： School of Information Science and Engineering
Date of Defence： May, 2023
Degree-Conferring-Institution： Harbin Institute of Technology

摘　要

随着物联网技术的迅速发展，物联网应用已经涵盖了智能家居、智能医疗、智能交通等多个领域，推动着芯片技术的不断进步和发展。片上系统（System on Chip，SoC）技术能够实现高性能、低功耗、高可靠性的系统级集成，广泛应用于各种领域的电子设备中。

然而，在物联网应用领域中，SoC 的功耗是制约其发展的一个重要限制因素。为了降低功耗，现代 SoC 设计采用动态电压频率调节（Dynamic Voltage and Frequency Scaling，DVFS）技术来实现功耗优化，但 DVFS 技术的调度策略一直是该技术领域中的设计挑战之一，其调度策略的优化对于提高系统的能效比具有重要意义，已经成为领域内的研究热点。与此同时，RISC-V 作为一种新型的指令集架构，以其低功耗、开放性和灵活性等特点在 SoC 的设计开发中得到广泛应用。因此，本文将 DVFS 技术和 RISC-V 架构相结合，提出了基于内核负载监测的 DVFS 调度方案，应用 DVFS 技术完成了面向物联网应用的 RISC-V SoC 设计开发。

本文首先研究了 RISC-V 指令集体系结构，开展了 RISC-V 处理器的建模仿真和设计实现的研究。确定了 RISC-V 处理器的总体架构，根据处理器的应用需求选择了合适的 RV32IMAC 指令集，实现了处理器的基本功能。在确定了总体架构之后，在体系结构模拟器 gem5 中进行了 RISC-V 处理器的系统级模型构建，并将该处理器模型应用于 DVFS 系统仿真，确定了针对本文低功耗 SoC 设计的 DVFS 实现方案。最后完成了三级流水线结构的 RISC-V 处理器核的 RTL 设计。

基于本文设计完成的 RISC-V 处理器核，开发了包括系统总线、外设接口等组成部分的 RISC-V SoC。接着，本文重点研究了 RISC-V SoC 的低功耗设计，对 RISC-V SoC 电源域与时钟域进行了划分，并设计了一种高效的 DVFS 控制器，该控制器以 RISC-V 处理器核中的控制状态寄存器为接口，实时监测系统负载变化情况，并通过控制时钟分频产生模块与电源模块为 SoC 提供对应的时钟频率与工作电压，实现了对整个 SoC 的动态电压频率调节。

在完成了 RISC-V 架构 SoC 的 RTL 设计后，进行了 SoC 的仿真验证与物理实现。完成了 riscv-tests 指令集测试，成功验证了处理器核的各项功能。搭建了 FPGA 原型验证平台，进行了物联网应用性能测试，RISC-V 处理器在 50MHz 运行主频下运行 CoreMark 跑分程序测试得分为 2.35 CoreMark/MHz。完成了 RISC-

V SoC 的物理实现和功耗评估，最终本文设计的 RISC-V 架构 SoC 在 DBH 180nm 的工艺尺寸下获得了最大为 37.9%的功耗收益，能够满足物联网应用对性能和功耗的需求。

关键词：物联网；SoC；RISC-V 指令集；低功耗；动态电压频率调节

Abstract

With the rapid development of IoT technology, IoT applications have covered multiple fields such as smart homes, medical care, and transportation, driving the continuous progress and development of chip technology. System on Chip (SoC) technology can achieve high-performance, low-power consumption, and high-reliability system-level integration, and is widely used in various electronic devices.

However, in the field of IoT applications, the power consumption of SoCs is a crucial limiting factor for their development. To reduce power consumption, modern SoC designs adopt the Dynamic Voltage and Frequency Scaling (DVFS) technique for power optimization. Nevertheless, the scheduling strategy of DVFS remains a challenging design issue in this field. The optimization of DVFS scheduling strategy is of significant importance for improving the energy efficiency of systems and has become a research hotspot in the field. At the same time, RISC-V, as a new type of instruction set architecture, has been widely used in the design and development of SoCs due to its characteristics such as low power consumption, openness, and flexibility. Therefore, this paper combines DVFS technology with the RISC-V architecture and proposes a DVFS scheduling scheme based on kernel load monitoring. The proposed scheme is applied in the design and development of RISC-V SoC for IoT applications using DVFS technology.

This paper first studied the RISC-V instruction set architecture, and conducted research on the modeling, simulation, and design implementation of RISC-V processors. The overall architecture of the RISC-V processor was determined, and a suitable RV32IMAC instruction set was selected based on the application requirements of the processor, achieving the basic functions of the processor. After determining the overall architecture, a system-level model of the RISC-V processor was constructed in the architecture simulator gem5, and the processor model was applied to the simulation of the Dynamic Voltage Frequency Scaling (DVFS) system, determining the DVFS implementation scheme for low-power SoC design in this article. Finally, the RTL design of the RISC-V processor core with a three-stage pipeline structure was completed.

Based on the RISC-V processor core designed in this study, a RISC-V SoC was developed which includes components such as system bus and peripheral interfaces.

Subsequently, this article focused on the low-power design of the RISC-V SoC, dividing the power domain and clock domain of the RISC-V SoC and designing an efficient DVFS controller. The controller uses the CSR register in the RISC-V processor core as an interface, real-time monitors the system load changes, and provides corresponding clock frequency and operating voltage to the SoC by controlling clock divider module and power supply module, achieving dynamic voltage frequency scaling for the entire SoC.

After completing the RTL design of the RISC-V architecture SoC, simulation validation and physical implementation were performed. The riscv-tests instruction set test was completed, successfully verifying the functionality of the processor core. An FPGA prototype verification platform was built and performance testing was conducted for IoT applications, with the RISC-V processor achieving a score of 2.35 CoreMark/MHz when running the CoreMark benchmark program at a clock frequency of 50MHz. The physical implementation and power evaluation of the RISC-V SoC were completed, and the final design achieved a maximum power savings of 37.9% in the DBH 180nm process size, meeting the requirements for performance and power consumption of IoT applications.

Key words: Internet of Things, SoC, RISC-V, low power consumption, dynamic voltage frequency scaling

目　录

第1章 绪　论

1.1 课题的研究背景和意义

随着物联网技术的快速发展，物联网设备的数量和种类飞速增长。这些设备通常需要长时间运行，并具有强大的计算、存储和通信能力。为了满足这些要求，需要高效的电子芯片来支持这些设备。然而，传统的单一功能芯片已经无法满足这些要求，需要更加复杂、集成化的芯片设计来支持物联网应用的各种需求。

在这种背景下，SoC 技术已经成为了一种重要的设计方案。SoC 可以将多个功能模块集成到一个芯片中，从而实现高度集成化的设计。这种技术可以帮助降低功耗、提高性能和可靠性，并且更加灵活地满足各种应用需求，广泛应用于各种领域的电子设备中。

然而，在物联网应用中，功耗成为了一个关键的限制因素。在物联网系统中，需要在多种位置和场景下进行信息的收集、传递和交换，这对于物联网应用对象的功耗和寿命有着较高要求。例如，在一种植入型人体健康监测系统中，通过可穿戴式传感器和植入人体设备可以实时监测用户的健康状况，但这种场景下难以更换植入设备的电池，因此需要设备在低功耗模式下进行工作[1]。物联网技术一般需要重点考虑小尺寸、低成本和低功耗三点因素，其中低功耗设计更是物联网技术面临的关键问题[2]，针对低功耗技术的研究一直是物联网应用领域的重要发展方向[3]。因此，需要设计低功耗的 SoC，以确保物联网设备的续航能力和能源消耗。

在 SoC 平台的搭建过程中，处理器核的设计是决定整个系统性能的关键部分。RISC-V 作为一种新型的指令集架构，以其低功耗、开放性和灵活性等特点在 SoC 的设计开发中得到广泛应用[4]。随着物联网应用的快速发展，RISC-V 架构也越来越受到物联网 SoC 设计者的青睐。在物联网应用中，因为不同的设备对处理器的要求不同，需要根据具体需求来设计不同的 SoC，而 RISC-V 架构因其开放式的设计使得各种厂商和组织都可以自由地开发和定制其自己的 SoC，满足各种不同的应用需求，同时也可以更好地支持开源软件和硬件生态系统的发展。因此，本文采用 RISC-V 架构来设计 SoC 中的微处理器。

综上所述，进行面向物联网应用的 RISC-V 架构 SoC 设计，其研究意义不言而喻。

1.2 国内外研究现状

本文主要开展面向物联网应用的 RISC-V 架构 SoC 设计，其中 RISC-V 处理器核及 SoC 的设计开发和 SoC 设计中的低功耗方法是需要重点考虑的两方面问题，因此本节将从以上这两个方面分别对国内外研究领域中的研究成果以及设计挑战进行分析和总结。

1.2.1 RISC-V 处理器核及 SoC 研究现状

RISC-V 架构于 2010 年诞生于美国加州大学伯克利分校的研究小组，该项目组的目标是设计一种免费的、开放的、可扩展的指令集架构，以支持各种体系结构的应用。RISC-V 架构正是以此为设计思想，逐步发展并走向成熟。其发展历程如图 1-1 所示，RISC-V 架构经历了不断地演进和完善，凭借其特有的优势广泛应用于各项科研领域。

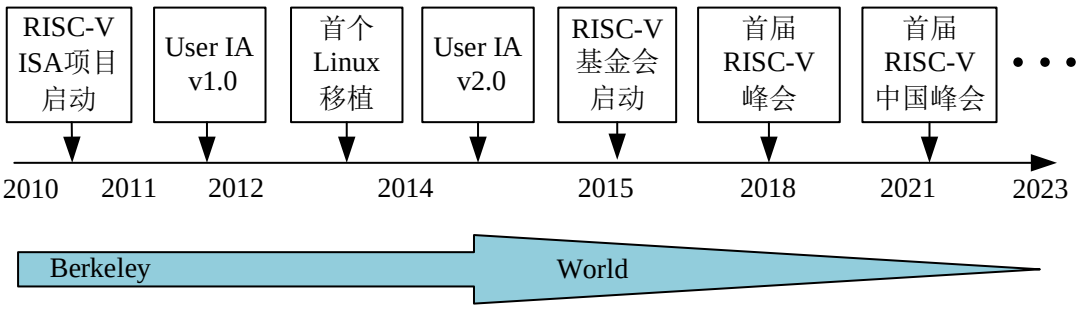

图 1-1 RISC-V 的发展历程图

迄今为止，基于 RISC-V 架构的处理器核与 SoC 设计已经发展出了许多优秀的设计。Rocket Core 是一款基于 RISC-V 指令集架构的开源处理器核，采用现代处理器微架构设计技术，采用 5 级流水线设计和多功能单元，实现高效的指令执行，并且支持可配置的扩展以适应不同的应用场景和需求，具有较高的性能表现[5]。此外，伯克利的科研人员还开发了另一款处理器核 BOOM Core[6]，此处理器核采用了乱序执行的设计，支持多发射技术，可以同时发射多条指令到流水线中，以提高指令吞吐量。它还支持多线程，可同时执行多条指令，提升了处理器的并行处理能力，其工作频率为 1.5GHz，达到了更高的性能指标。Rocket Core 和 BOOM Core 都是通过伯克利大学设计的一款 SoC 配置工具 Rocket Chip 配置生成的，目前已经广泛应用于学术研究与商业开发。

RISC-V 处理器核的研究人员不仅追求高性能的设计目标，在低功耗设计领

域也有许多发展突破。PULPinol 是由苏黎世瑞士联邦理工学院(ETH Zurich)研发的低功耗并行处理器架构，旨在为嵌入式应用提供高性能和低功耗的解决方案[7,8]，在多个层面体现了低功耗的设计理念。同时其研究人员还在此架构基础上设计了 RI5CY、Zero-riscy 和 Micro-riscy 等多款 RISC-V 处理器核。最早开发的 RI5CY 为一款实用级处理器，具有四级流水线结构，采用 32 位按序调度的设计方式[9]，在超低功耗信号处理领域有良好的应用效果。之后开发的 Zero-riscy 处理器核使用了指令压缩、动态时钟门控等低功耗设计方法[10]，支持标准的 RV32I 指令子集，由于其在 FPGA 和 ASIC 实现方面的灵活性，广泛应用于物联网、嵌入式系统和数字信号处理等领域。Micro-riscy 前两款处理不同，它支持 RV32EC 架构，仅包含 16 个通用寄存器[11]，大幅减少了硬件面积，其功耗也相对较低。

SiFive 公司是 RISC-V 开源指令集生态系统的重要成员之一[12]，该公司设计开发了 Freedom E310 SoC，该 SoC 通过设置低功耗模式和外设优化等多种技术手段有效降低了芯片功耗，适用于嵌入式系统、物联网设备、传感器网络等多个应用场景。SiFive 公司近年来在车用处理器领域也做出了许多贡献，其在 2022 年推出的 64 位处理器核 S7-A 具有极高的灵活性和可兼容性[13]，能够满足现代车用 SoC 对高性能安全岛的需求。

国内对于 RISC-V 开源处理器核与 SoC 设计的研究起步较晚，关键技术往往受到西方国家的限制和制约，这给我国自主开发芯片设计带来了巨大的挑战和压力。但近年来随着国家政策的大力支持和市场需求的不断激励，越来越多的国内企业开始关注和投入到 RISC-V 处理器的研发和设计中，积极探索和推动 RISC-V 处理器在国内的应用和推广。

蜂鸟 E200 系列是由国内设计人员开发的一款开源 RISC-V 处理器，其采用两级流水线结构，主要适用于极低功耗的应用场景。与同级的 ARM 系列产品相比，蜂鸟系列处理器的功耗和面积都要更优，实现了极高的能效比。蜂鸟 E203 处理器核完全开源，并提供了完整的配套 SoC，其设计者还出版了基于蜂鸟 E203 处理器的相关书籍，便于国内 RISC-V 初学者进行学习研究[14]。

2019 年，兆易创新公司开发了国内第一款自主知识产权的 RISC-V MCU GD32VF103[15,16]，该 Bumblebee 处理器内核支持多种低功耗模式，包括待机模式、休眠模式和停止模式，可以大幅降低系统功耗，在物联网技术及其它超低功耗应用场景中广泛应用。同时，设计人员在该处理器在性能提升方面也做出了许多贡献。RISC-V 指令集中的乘除法指令一般需要多个周期才能完成，且需占用较多的硬件面积，而 Bumblebee 处理器核中设计的硬件乘除法器以及加速

单元在解决此问题方面展现了良好的效果，有效地平衡了性能与功耗。

香山处理器项目于 2020 年 6 月启动，旨在开发一款开源高性能 RISC-V 处理器。第一代“雁栖湖”微架构已经成功流片，回片性能符合预期；第二代“南湖”微架构已进入最后的优化迭代阶段，仿真评估结果显示，“南湖”微架构 SPECCPU2006 分值约为 10.42 分/GHz，同频性能达到 ARM CortexA76 水平，单核性能为当前开源处理器之最[17]；第三代“昆明湖”微架构将目前也已建立联合开发团队[18]，已经启动了架构设计工作。

阿里巴巴旗下公司平头哥于 2021 年推出了玄铁 E907 处理器，该处理器在已经推出的 E902、E906、E910 等产品基础上，兼顾高性能与低功耗的特点，采用五级流水线设计，最高工作频率超过 1GHz[19]。2022 年 11 月，平头哥又推出了新款玄铁 C908 处理器，在处理器能效方面实现了大幅提升，比前一代产品提升 30%以上，在 AI 测试的图像分类任务中也展现出了超过 2 倍的性能提升，处于国内领先地位，可广泛用于穿戴式设备、智能家居、工业自动化、AI 计算等应用场景[20]。

RISC-V 架构处理器经过多年的发展，已经在学术科研与商业制造的多个领域得到广泛应用，表 1-1 展示了国内外部分研发机构开发的 RISC-V 处理器及其对应平台。

表 1-1 国内外部分 RISC-V 处理器及对应平台

平台名称	发布机构	处理器
Rocket Chip	Berkeley	Rocket、BOOM
PULPinol	苏黎世瑞士联邦理工学院	RI5CY、Zero-riscy Micro-riscy
Freedom E310	SiFive	Rocket Core
HBird-E200	芯来科技	蜂鸟 E203
GD32VF103	兆易创新	Bumblebee
玄铁 C908	平头哥	RV64GCB

总体来看，关于 RISC-V 处理器的研究主要聚焦于处理器性能与功耗两个关键指标，针对不同应用领域开展具有方向性的设计研究。而 RISC-V 指令集架构凭借其模块化可配置的指令集、简洁的运算指令、自定制指令拓展等多方面优点，在计算效率和功耗上均显示出独特优势。采用 RISC-V 指令集架构进行处理器设计，可以实现自主开发，降低依赖于外部技术和软件生态的风险，

从而提高芯片的可控性和安全性，更好地适应我国市场的需求。并且，通过采用 RISC-V 架构进行低功耗、高性能的嵌入式处理器研究，可以有效地提高处理器的能效比和性能，为我国芯片技术的发展提供更多的选择和支持。因此，本项目采取 RISC-V 架构处理器，并将充分发挥其在低功耗设计中的开发潜力，以满足面向物联网应用的设计需求。

1.2.2 SoC 设计中的低功耗方法研究现状

现代 SoC 往往从多个层级来进行低功耗设计，系统级设计往往能对整个 SoC 的功耗产生最为明显的优化效果，也是低功耗设计中最为关键的一环。DVFS 技术能够根据芯片运行需要动态调节芯片的运行频率和电压[21]，具有良好的节能效果。Liu 等人开发的一款自供电的物联网 SoC 芯片，具有一个电源管理单元（Power Management Unit，PMU）,支持事件驱动的细粒度 DVFS 技术，实现了 SoC 的最小 194nW 功耗和 PMU 的 5.2nW 静态功耗，峰值效率为 92.6%[22]。Hassan 等人对 DVFS 技术在热感知任务调度方面的方法进行了优化，提出了一种整数线性规划模型，包括对于温度和片上资源的相关约束，与比传统的热感知调度方法相比，该技术可减少 46%的测试时间[23]。

Safari 等人将一种基于功率感知的任务列表调度算法与 DVFS 技术相结合[24]，对需要消耗不同资源的任务配置适当改变处理器的电压和频率，在不违反任务优先级约束的前提下重新分配处理器的空闲间隙，实现了分布式环境中的高效任务调度，同时能够显著降低功耗，测试数据表明应用此技术的系统可节省 38%以上的能量。

优秀的低功耗 SoC 设计不仅要在系统级层面力求节能，在其他层次也同样需要降低功耗。国外不少学者在多阈值 CMOS 技术方面做出了诸多贡献[25,26]。对于 CMOS 电路来说，增大晶体管的阈值电压将减少电路单元的漏电流，而漏电流是电路产生静态功耗的重要原因，降低漏电流可以有效降低静态功耗。Agrawal 等人提出了一种改进的多阈值 8×8 SRAM 结构，与标准的 6T SRAM 结构相比可降低 21%的功耗[27]，该结构为处理器缓存低功耗设计提供了结构基础。门控时钟通过在时钟网络上增加开关门控，减少了不必要的开关功耗，具有降低功耗、优化时序、提高可靠性和灵活性等优点[28]，广泛应用于低功耗数字电路的设计中。

国内研究人员对于 SoC 低功耗设计方法的研究与上述国外学者研究的领域类似。在系统级低功耗设计方面，陈道品提出了一种基于多电源域和自适应调压的 SoC 低功耗技术[29]，其研究开发了一种基于 DVFS 技术的调压电路，可对

电路内部的电压数据进行收集整合，并通过测算分析电源使用裕度来调节供电电压，其在不同工作模式下对功耗的优化效果可达 90%～99.6%。邵胜芒采用双电源多模式的方式设计了一种适用于低功耗芯片的电源架构[30]，其电压域可通过 DVFS 调压的方式进行快速调整，并通过 SMIC55 的 CMOS 工艺进行流片测试，最终得到在室温下最大 58.3%的功耗收益。

在 DVFS 技术中，电路在不同电压工作点直接的切换会造成一定程度的能耗损失。为解决此问题，李航设计了一种结合片内电源轨切换和片外 DC-DC 调压的混合 DVFS 调节策略[31]，该切换方法能够在 $V_{DD}/2$～V_{DD} 的宽电压范围下达到 80ns/V～120ns/V 的切换速度，有效降低了由电压切换造成的能耗损失。

国内研究人员在系统级以外其他层级的低功耗设计领域也取得了许多成果。陈力颖等人基于门控时钟技术，提出了一种降低时钟树功耗的方法[32]，其通过修改综合工具的内部逻辑，使其插入的门控单元更靠近时钟树的根节点，从而降低时钟网络的翻转率，进而降低内部功耗，测试结果表明该方法可以降低约 12.5%的动态功耗。

综上所述，降低 SoC 的功耗可以从系统级、电路级等多个层面开展研究，其中系统级低功耗设计对整体芯片功耗的优化效果极为显著。而在系统级低功耗设计中，能够直接调节影响芯片功耗的电压和频率这两大关键因素的 DVFS 技术越来越受到设计者的重视。

在 DVFS 技术中，频率电压表的设计裕度、有限个电压点的权衡设置、工作模式的切换代价等问题都是 DVFS 技术的设计挑战。而如何能够正确衡量处理器性能需求，并根据此需求调节系统的目标电压与时钟频率，即 DVFS 的调度策略，一直是研究人员重点关注的问题。在系统实际工作过程中，过早或过晚地调整电压和时钟频率，会对造成处理器性能的损失。良好的 DVFS 调度策略应能够在尽量少损失运算性能的前提下最大程度地优化功率，有必要进行进一步的研究。

1.3 本文主要研究内容

本文面向物联网技术的应用，主要研究低功耗 RISC-V 架构的 SoC 设计与实现，整体研究按照图 1-2 中所示的顺序开展，具体研究内容如下：

(1) 开展面向物联网应用的低功耗 RISC-V 架构的 SoC 系统级设计，确立 RISC-V 处理器的整体架构及各模块设计方案，并采用基于 CPU 负载监测的 DVFS 技术进行系统级低功耗设计；

(2) 根据已经建立的低功耗处理器设计方案，在体系结构模拟器中构建

CPU 模型与功耗评估模型，并开展对 DVFS 技术的研究，通过进行 DVFS 仿真测试，实时统计系统功耗，分析比较 DVFS 策略的能耗效率，确定 DVFS 实现方案；

(3) 在模拟器仿真效果达到预期后，进行低功耗 RISC-V 架构 SoC 的 RTL 设计与实现，并进行系统级低功耗设计；

(4) 在 SoC 设计完成后，开展 RISC-V SoC 的仿真验证，并搭建 FPGA 原型验证平台，进行系统板级模拟验证，验证结束后进行整个 RISC-V 架构 SoC 的物理实现，最终进行功耗分析与评估。

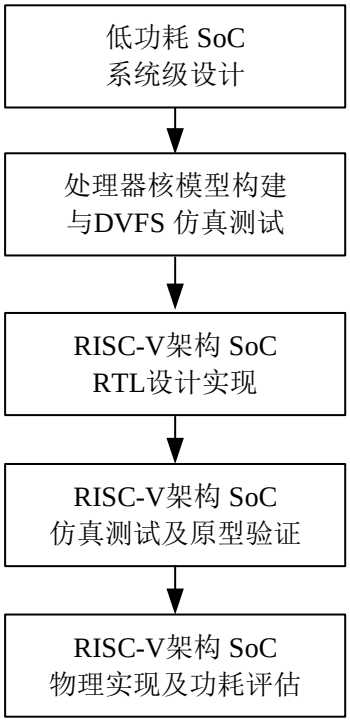

图 1-2 研究内容整体框图

1.4 论文结构安排

本文共有五个章节，具体的结构安排如下：

第 1 章绪论。分析了课题的研究背景与意义，研究了国内外关于 RISC-V 架构 SoC 与低功耗设计方法的研究现状，最后给出了本文的主要研究内容以及结构安排。

第 2 章 RISC-V 架构 SoC 设计方法研究。分析了 RISC-V 架构 SoC 的设计方法，由此引出了 RISC-V 架构处理器设计和低功耗 SoC 平台设计两部分主要内容。首先分析了 RISC-V 指令集体系结构特点和处理器设计的相关技术，重

点对处理器流水线设计开发的相关技术进行了探究。之后分析了低功耗 SoC 的设计方法，重点研究了系统级低功耗设计中的 DVFS 技术原理与框架。

第 3 章 RISC-V 处理器建模与设计实现。首先进行了 RISC-V 处理器的总体架构设计，以低功耗为设计理念与实现目标，从指令集的选取、流水线的设计、内部模块结构等多方面对 RISC-V 处理器系统架构进行设计。之后在体系结构模拟器中构建了 RISC-V 处理器系统模型，并进行了 DVFS 系统仿真，实时统计系统功耗，分析比较 DVFS 策略的能耗效率，确定 DVFS 实现方案。最后进行了 RISC-V 处理器的 RTL 设计，完成了三级流水线结构的处理器内核设计。

第 4 章 RISC-V 架构 SoC 的设计与实现。确定了 RISC-V 架构 SoC 的整体结构，将本文设计开发的低功耗 RISC-V 处理器核连接外部中断异常处理模块、调试模块以及各外设模块，构建整体 RISC-V SoC。之后进行了 SoC 低功耗设计，对整个 SoC 的电源域和时钟域进行了划分，并设计了 DVFS 控制器，应用 DVFS 技术实现了系统级低功耗设计。

第 5 章 RISC-V 架构 SoC 的仿真验证与物理实现。搭建了 RISC-V 架构 SoC 的验证平台，对 RISC-V 处理器进行了指令集验证。之后使用 OEM Zynq-7000 SoC XC7Z2020 开发板，将构建的 RISC-V 架构 SoC 用 FPGA 进行了原型验证。最后完成了整个 SoC 的物理实现，并对整个系统产生的功耗进行了分析与评估。

第2章 RISC-V 架构 SoC 设计方法研究

本章围绕着 RISC-V 架构 SoC 的设计方法展开，开展对 RISC-V 指令集体系结构和处理器设计技术的研究，之后深入研究了低功耗 SoC 设计方法，为后续低功耗 RISC-V 架构 SoC 平台的设计提供了理论基础。

2.1 RISC-V 指令集体系结构研究

RISC-V 指令集开发人员通过将指令设计规整化，使得指令集具有非常清晰的结构和规律性。图 2-1 展示了本文中使用的部分指令集格式，对于普通的非压缩 RISC-V 指令而言，其指令长度都为 32 位。

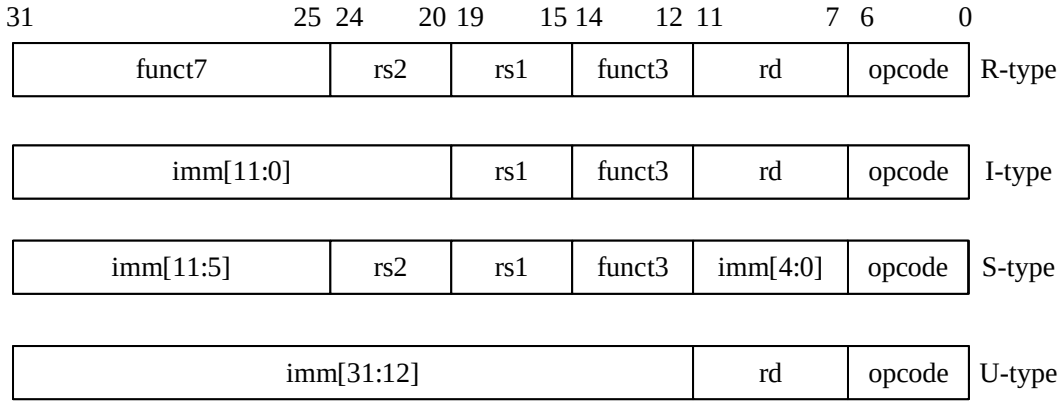

图 2-1 RISC-V 规整的指令格式图

每条 RISC-V 指令都可以根据内部的寄存器索引被划分为若干个指令段，各指令段的含义如表 2-1 所示。

表 2-1 RISC-V 指令编码

指令段代号	指令段名称	功能描述
opcode	操作码	区分不同类型的指令
funct3	3 位功能码	区分指令的具体功能
funct7	7 位功能码	
imm	立即数	指令运算所需的立即数
rs1	源寄存器 1	指令运算所需的操作数 1
rs2	源寄存器 2	指令运算所需的操作数 2
rd	目的寄存器	指令运算的运算结果

RISC-V 的指令集与其他架构不同，其中实现同类功能的指令组成了可由英文字母指代的模块，不同模块可以自由地进行组合连接。具体的模块化指令子集如表 2-2 所示，其中只有字母 I 表示的基本整数指令子集强制要求实现，除此之外的其余子集均可自由选择配置，方便设计开发人员针对不同应用场景的需要进行设计。此外，RISC-V 架构还支持用户自定义指令，从而为处理器的开发和设计提供了更大的灵活性和创新空间。CPU 架构开发人员通过灵活配置各指令子集，可以有效地优化处理器在特定应用领域中的性能，并且由 RISC-V 架构设计开发的处理器的功耗明显低于同类型的 ARM 处理器[33]。

表 2-2 RISC-V 模块化指令集

RISC-V 指令集	名称	指令数	描述
基本指令集	RV32I	47	整数指令，32 位寻址空间
	RV32E	47	RV32I 子集，仅支持 16 个通用寄存器
	RV64I	59	整数指令，包含一部分 32 位整数指令，64 位寻址空间
	RV128I	71	整数指令，128 位寻址空间
扩展指令集	M	8	整数乘除法指令
	A	11	存储器原子指令、互斥读、互斥写指令
	F	26	单精度（32bit）浮点指令
	D	26	双精度（64bit）浮点指令
	C	46	压缩指令，指令长度为 16 位

考虑到本设计面向物联网技术的低功耗应用场景，本文采用 RV32IMAC 的指令集架构进行开发设计，在满足一定性能需求的前提下追求低功耗的设计指标。

2.2 处理器设计相关技术研究

2.2.1 处理器流水线功能需求

流水线结构是现代处理器体系结构中的重要组成部分，处理器的流水线设计也是进行处理器开发的一项核心技术，在现代计算机体系结构中广泛应用[34-37]。流水线技术是一种时间重叠技术，通过把对指令操作过程中的几个阶段并行化执行，最大化利用硬件资源，从而提高了吞吐量和运行效率，有助于提高处理器性能。

一条 RISC-V 指令的执行一般要经历以下五个阶段：

(1) 取指

指令取指为流水线的第一级，指的是从指令存储器中将指令读出的过程。

(2) 译码

指令译码过程是对已取出的指令进行翻译，处理器解码指令确定指令需要执行的操作类型，并从通用寄存器组中将所需操作数读出。

(3) 执行

指令执行就是根据指令的操作类型对操作数执行操作的阶段。例如，算术逻辑部件运算器（Arithmetic Logical Unit，ALU）在执行指令时可以执行加、减、乘、除、位移和位运算等操作。

(4) 访存

指令访存是指的是与存储器相关的指令访问存储器的过程，即从存储器中取出数据或写入数据至存储器的过程。

(5) 写回

指令写回是将指令执行结果写回到目的操作数寄存器或者存储器的过程。根据指令类型的不同，其写回的结果值可能来自执行阶段的计算结果，或是来自访存阶段从存储器读取的数据。

指令经历的五个阶段如图 2-2 所示，假设每个阶段都需要 t 时间完成，如果采用单周期的五级流水线去执行 3 条指令，总共需要花费 $7t$ 的时间，如果不采用流水线设计，那 3 条指令执行将总共花费 $15t$。由此可见，使用流水线技术可以有效地提升处理器的计算效率。

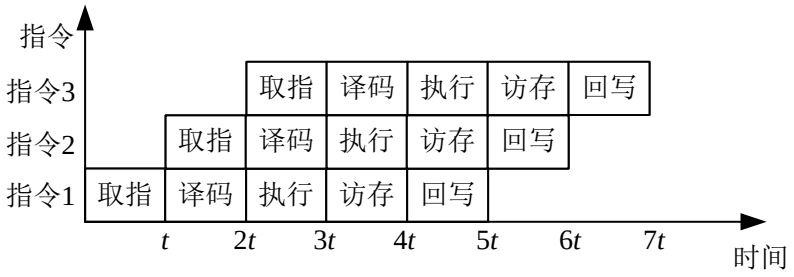

图 2-2 经典五级流水线执行过程图

2.2.2 处理器流水线冲突分析

处理器流水线技术可以在同一时间内并行地执行多条指令，从而提高处理器的效率。但是，在处理器流水线的执行过程中，由于指令之间存在依赖关系

和硬件资源的竞争，可能会对流水线执行过程造成停顿或延误。在处理器指令的实际执行过程中，存在以下三种冲突或冒险：

(1) 资源冲突

在 RISC-V 处理器中，指令的执行涉及到多种硬件资源，例如 ALU、乘法器、除法器、存储器等。这些硬件资源的数量是有限的，因此在执行指令时可能会出现资源冲突。资源冲突指的是在一个指令尚未完成时，另一个指令需要使用相同的硬件资源，从而导致后者无法按时完成。资源冲突可能会导致流水线停顿或延迟，从而降低处理器的效率和性能。现代的处理器一般采用复制硬件资源或停顿流水线的方式来解决资源冲突。

在本设计中，采用模块间握手信号来解决资源冲突。本设计中的各模块接口均采用严谨的 valid-ready 握手接口，若在某个模块内出现了资源冲突，则其向其他模块输出的 ready 信号为低，无法与其他模块完成握手，需要等待至资源冲突被解除后方可实现信号传递。

(2) 数据冲突

在流水线中，如果两条指令需要访问相同的数据，并且它们在时间上存在重叠，就会产生数据冲突，一般有以下三种形式：

写后写冲突（Write After Write，WAW）指的是两个指令先后写入同一寄存器，后面的指令覆盖了前面的指令的结果，导致前面的指令的结果被丢失；读后写冲突（Write After Read，WAR）指的是一个指令在写入一个寄存器后，后面紧跟着另一个指令需要读取该寄存器中的值，但由于前面的指令还未完成写入，导致后面的指令读取到的值不是最新值，从而引发错误；写后读冲突（Read After Write，RAW）指的是一个指令在读取某个寄存器的值后，紧接着又需要对该寄存器进行写操作，由于两个操作之间存在数据相关关系，写操作会覆盖读操作的结果，导致运算出现错误。

WAW 和 WAR 相关性可以通过寄存器重命名的方法进行解决，这种技术将每个物理寄存器都映射到多个逻辑寄存器中，从而消除了 WAW 和 WAR 冲突。而 RAW 属于真数据相关，后序指令必须使用前序指令完成的结果，从而造成流水线停顿。虽然可以通过数据旁路传播等方法尽可能减少流水线停顿的性能损失，但此类方法都会消耗额外的面积和功耗。

(3) 分支冒险

在 RISC-V 架构中，带条件的跳转指令是常见的指令类型之一，由于该类指令在取指单元无法确定是否要发生跳转，因此需要在该指令真正执行完成后才能给出实际的跳转结果。这种延迟可能会导致处理器的流水线出现停顿，从

而降低了处理器的性能。分支预测技术可以对指令的跳转结果作出预估，从而尽可能地避免出现流水线停顿的情况[38]。

目前已经有许多较为成熟的分支预测策略，一般分为静态分支预测与动态分支预测两种。静态分支预测是一种基于分支指令静态特征进行预测的方法，只根据指令的特征进行预测。其中，BTFN（Backward Taken, Forward Not Taken）是一种较为简单的静态分支预测方法，其基本思想是：如果当前分支指令是跳转指令且被跳转的指令位于当前指令之后，那么当前分支指令就被预测为跳转。否则，当前分支指令被预测为不跳转。BTFN 方法的优势在于它非常简单且易于实现，并且由于其不需要额外的硬件单元或者复杂的算法，具有较低的硬件成本，广泛应用于嵌入式应用场景中。

动态分支预测指凭借当前分支指令和已经执行过的指令的历史信息综合考虑来进行预测的方法。常见的方法有通过计数器来记录上次预测方向的方法，和将多个预测器组织成预测器表格的方法。这些方法能实现较高的分支预测正确率，但会造成极大的硬件开销，一般用于高性能处理器设计。本设计作为面向低功耗场景的处理器，采用 BTFN 的静态预测方案。

2.2.3 处理器流水线结构分析

在处理器的设计中，流水线级数的选择是一个需要仔细考虑的问题。较深的流水线级数可以降低相邻两级流水线之间的硬件逻辑量，从而能够使处理器达到更高的运行主频和更高的流水线吞吐率，进而提升处理器性能。然而，流水线级数的增加必然导致了芯片内部器件数量和电路面积的增加。此外，由于每个流水线级都需要握手协议，最后一个流水线级可能会反压到最前面的流水线级，导致严重的时序问题。较深的流水线级在遇到分支预测失败时也会导致更多的性能损失和功耗浪费，不适用于低功耗处理器设计。相反，合并流水段可以减少流水线级数和芯片面积和功耗开销，并且更容易解决流水线上的冲突问题。然而，合并流水段也会降低流水线上同时处理的指令数量，从而降低流水线吞吐率，可能会导致性能下降。因此，在 CPU 核的设计中，需要根据实际应用场景进行折衷，选择适当的流水线级数。

为了优化处理器的流水线效率，现代处理器常采用三级流水线结构，将“执行”、“访存”和“写回”三个操作阶段放在同一级流水线中完成[39]。这种结构能够利用不同指令执行、访存和写回所需的时间差异，从而在最短的时间内完成指令的处理[40]。相比于其它流水线结构，这种设计能够提高处理器的效率和性能，但同时也需要考虑寄存器和面积等方面的成本。

2.3 低功耗 SoC 设计方法研究

2.3.1 低功耗设计层次

随着嵌入式芯片对低功耗技术日益增长的设计需求，业内广大科研人员一直在研究设计各种能够平衡性能与功耗的 SoC。目前的低功耗 SoC 开发人员已经将低功耗设计思想贯穿设计的全过程[41]。按照数字电路的低功耗设计抽象层次的不同，可以将其分为系统级、行为级、寄存器传输级、逻辑门级、版图级以及电路级[42]。并且，在各个层级使用的低功耗方法也各不相同[43]。

不同抽象层次产生的功耗优化效果有较大区别，一般来说高层级的优化方法达到的优化效果会更加显著，如图 2-3 所示，系统级的优化效果可以达到 70%甚至以上，而电路级的优化效果则仅有 5%~10%。

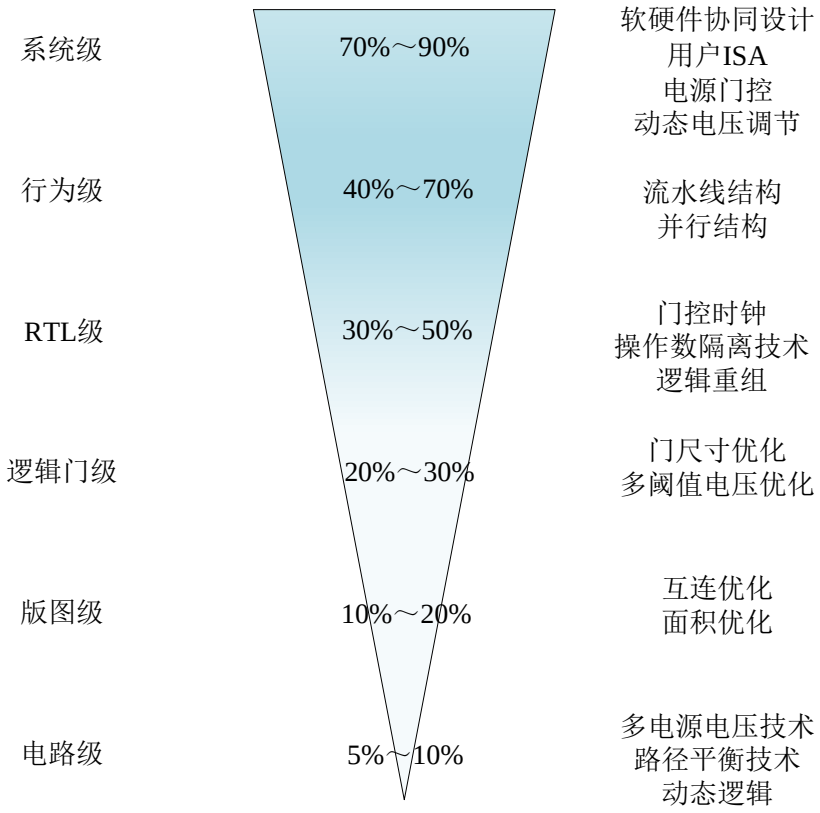

图 2-3 各层次优化方法及效果图

系统级设计对每一款 SoC 的开发来说都是最为关键的环节，而系统级低功耗设计同样在功耗优化方面至关重要。在这一层次上，可以采用软硬件协同设计、电源门控技术、动态电压调节技术等多种方式。行为级低功耗设计思想主要从电路的运行体系结构出发，采用流水线结构或并行结构等方式来降低功耗。寄存器传输级主要是通过降低电路中关键路径上晶体管的翻转率来降低芯片的

功耗，主要包括门控时钟技术、操作数隔离技术等。逻辑门级主要通过使用多阈值电压优化技术来降低功耗。版图级低功耗设计主要在后端布局布线的过程中实现，电路级低功耗设计则主要包括多电源电压技术、路径平衡技术等[44]。

综合以上分析，本文重点对系统级低功耗设计方法中的 DVFS 技术进行研究，应用 DVFS 技术进行低功耗 RISC-V 架构 SoC 设计。

2.3.2 DVFS 基本原理

DVFS 技术根据不同的应用场景和系统负载动态地改变电路系统的工作电压和时钟频率，实现了功耗和性能之间的可调权衡。在移动设备、物联网和其它嵌入式系统中，DVFS 技术已经成为实现功耗和性能优化的关键技术之一。

CMOS 电路的功耗分为静态功耗与动态功耗，总体计算方式如公式 (2-1) 所示。

$$P = P_{\mathrm{s}} + P_{\mathrm{d}} = V_{\mathrm{dd}} I_{\mathrm{Q}} + \alpha C V_{\mathrm{dd}}^2 f_{\mathrm{c}} \tag{2-1}$$

式中 P_{s}、P_{d}——分别为静态功耗和动态功耗(W)；

V_{dd}——电路工作电压(V)；

I_{Q}——漏电流(A)；

α——电路开关活动性；

C——负载电容(F)；

f_{c}——电路开关频率(Hz)。

静态功耗是由半导体自身工艺所带来的必要功耗，功耗大小主要取决于 CMOS 电路的电源电压、电流以及电路本身的晶体管结构，与电路的工作频率无关。动态功耗是指在电路切换过程中由电流瞬间流过导体引起的功率损耗，它是集成电路中主要的功率消耗来源之一。动态功耗的影响因素主要包括电容、电压和频率三个方面。电路工作的电压和频率越高，电路的切换速度也就越快，每个晶体管的充放电所需的能量也就越多，因此消耗的功耗也会增多。DVFS 技术正是以电压和频率这两个最主要的功耗影响因素为研究对象，通过合理地调整处理器的工作电压和频率来优化功耗。

DVFS 技术虽然能够显著降低系统功耗，但也带来了一些负面影响。CMOS 电路延迟与工作电压的关系如公式 (2-2) 所示。

$$delay = K V_{\mathrm{dd}} / \left(V_{\mathrm{dd}} - V_{\mathrm{t}}\right)^2 \tag{2-2}$$

式中 K——与工艺相关的常数；

V_t——阈值电压(V)。

由公式 (2-2) 可以看出，降低工作电压必然增加电路的延迟时间，从而导致电路翻转速率降低，进而延长系统任务的完成时间。因此，DVFS 技术为了实现系统功耗的降低，需要牺牲一部分系统响应能力。CMOS 电路中的开关频率主要受工作电压的影响，具体关系如公式 (2-3) 所示。

$$f_c = K\frac{(V_{dd}-V_t)^{\varepsilon}}{V_{dd}} \tag{2-3}$$

式中 ε——与工艺相关的常数。

由公式 (2-1) 以及公式 (2-3) 可知，电路的动态功耗$P \propto V_{dd}(V_{dd}\text{-}V_t)^2$，可见电路工作电压对系统功耗的影响十分显著。而在不同的工作电压下，电路系统执行任务所需的时间也不相同，由公式 (2-2) 可以推断出：

$$T(V_{dd1}) = \left[(V_{dd2}-V_t)^2 / V_{dd2}\right] \times \left[V_{dd1} / (V_{dd1}-V_t)^2\right] \times T(V_{dd2}) \tag{2-4}$$

式中 $T(V_{dd1})$、$T(V_{dd2})$——电路工作在两种工作电压V_{dd1}和V_{dd2}下时任务执行时间。

假设$V_{dd} \gg V_t$，则化简公式 (2-4) 为：

$$T(V_{dd1}) = T(V_{dd2}) \times V_{dd2} / V_{dd1} \tag{2-5}$$

由公式 (2-5) 可知，系统工作电压的降低会导致任务执行时间的延长，从而降低系统工作效率。因此，如何平衡系统响应能力与消耗功率之间的矛盾，实时优化系统工作状态是 DVFS 技术关注的一项重要问题。本文利用处理器系统负载率的变化来判断处理器的性能需求，从而确定系统工作电压与时钟频率，在满足各系统任务需求的同时实现系统功耗最小化。

2.3.3 DVFS 技术框架

为了实现 DVFS 技术，需要对电路系统设计额外的硬件电路来完成处理器的性能需求判断并调节系统工作电压与时钟频率。DVFS 控制器根据处理器的负载信息对电路工作电压与时钟频率进行实时控制与调节，主要结构可以分为两类：开环 DVFS 控制器与闭环 DVFS 控制器[45]。开环 DVFS 控制器需要预先设定好频率电压表，该频率电压中设定了系统支持的若干工作电压与时钟频率点。DVFS 控制器根据电路系统工作状况从频率电压表中选择具体的工作电压与时钟频率，从而实现动态调节。闭环 DVFS 控制器则通过工艺-电源-温度（Process-Voltage-Temperature，PVT）监测模块来对系统电路进行感知，进而预

测系统工作电压与频率。与开环 DVFS 技术相比，闭环 DVFS 技术可以有效降低频率电压表的设计裕量，但其需要额外添加精准可靠的 PVT 监测模块，增加了设计成本与复杂度。综合上述分析，本文以开环 DVFS 控制器作为主要研究对象。

DVFS 控制器的工作流程如图 2-4 所示，首先实时监测并收集处理器系统的负载信息，以此作为性能需求预测的基础；之后根据系统当前工作状态，结合系统负载状况来对系统性能需求进行预测；完成预测后在频率电压表中确定对应的工作电压与时钟频率，并对电路的各个模块进行配置；最后整个系统达到稳定状态，以最优的工作电压和时钟频率运行系统任务。此外，在整个工作流程中，DVFS 控制器都对系统负载情况进行实时监控，保证系统在负载情况发生变化时能够迅速准确地作出反应，从而最大限度地优化系统整体运行功耗。

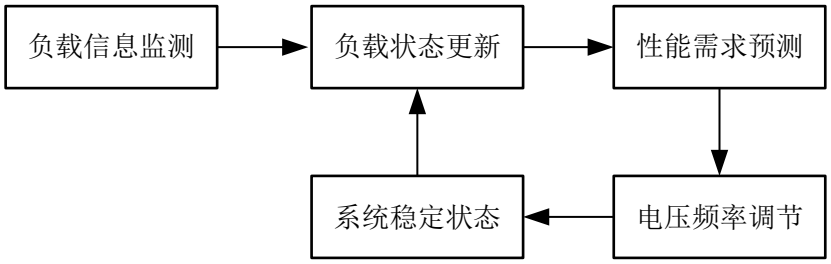

图 2-4 DVFS 工作流程示意图

2.4 本章小结

本章首先研究了 RISC-V 指令集体系结构，探讨了指令集特点与处理器设计相关技术之间的关系。之后本章深入分析了处理器设计相关技术，重点研究了处理器流水线需要完成的功能以及其中可能出现的冲突，并对由流水线结构不同导致的功耗和性能差异进行了探究。此外，本章还研究了低功耗 SoC 的设计方法，包括低功耗设计层次、DVFS 技术的基本原理和框架。这些理论分析与方法研究为低功耗 RISC-V 架构 SoC 平台的设计提供了完整的框架和指导，对于实现更加高效、可靠的 SoC 设计具有重要的参考意义。

第3章 RISC-V 处理器建模与设计实现

处理器的设计是整个 SoC 设计的核心部分，本文首先根据物联网应用对性能和功耗等方面的需求确定处理器核设计方案，之后在体系结构模拟器中进行建模，并在此模型上应用 DVFS 技术，通过仿真测试的形式对处理器系统功耗进行评估，以确定最佳的 DVFS 调度方案，最后进行三级流水线结构的 RISC-V 处理器核的 RTL 设计实现。

3.1 RISC-V 处理器总体架构

根据本文面向物联网应用的需求，主要以低功耗为设计理念与实现目标，从指令集的选取、流水线的设计、内部模块结构等多方面对 RISC-V 处理器系统架构进行设计。考虑到本设计面向的物联网技术的低功耗应用场景，采用 RV32IMAC 的指令集架构进行开发设计，使用多周期的乘除法单元，而不支持浮点运算，在满足一定性能需求的前提下追求低功耗的设计指标。

处理器的流水线是另一个需要重点考虑的问题。根据第二章的分析，三级流水线结构能够有效地提高处理器的效率，且消耗的硬件资源较少，适用于低功耗处理器的设计开发。综合上述分析，本设计中的低功耗 RISC-V CPU 采用三级流水线结构，分别为取指（Instruction Fetch，IF）、译码（Instruction Decode，ID）和执行（Execute，EX），具体结构如图 3-1 所示。

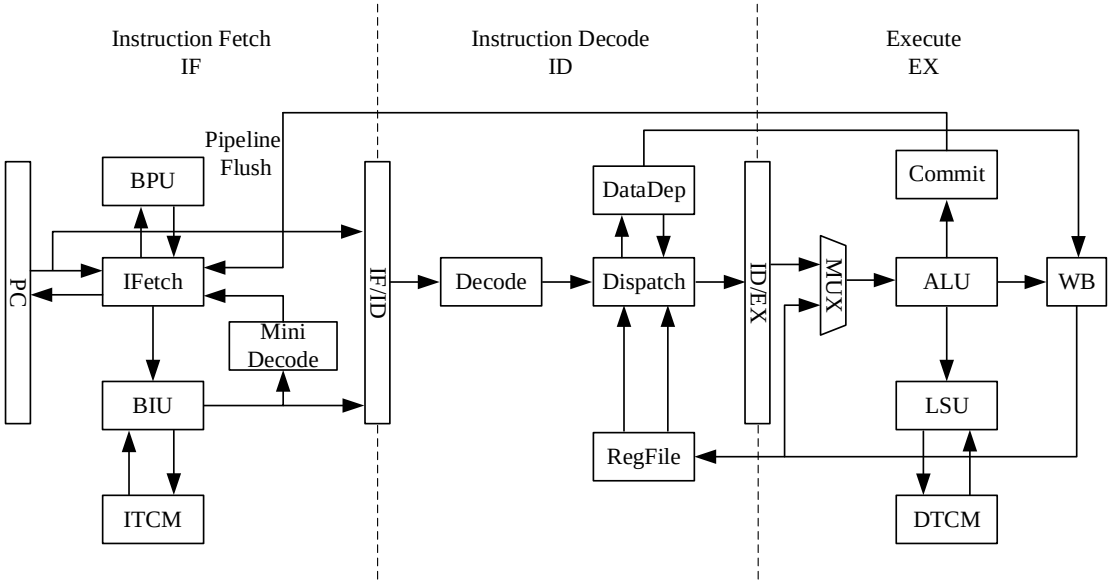

图 3-1 RISC-V CPU 结构示意图

3.2 RISC-V 处理器系统级模型构建

体系结构模拟器，简称为模拟器，可用于对处理器微结构进行性能评估，量化系统的性能，并在生成实际硬件之前发现并解决问题，从而降低开发成本，缩短开发周期。此外，模拟器可以在不同的应用场景下模拟处理器核的运行，评估不同的处理器设计方案，为后续的优化和改进提供有价值的信息，具有重要的实际应用价值。上节中已对处理器核设计方案进行了初步拟定，之后将进行处理器的建模与 DVFS 仿真。本节主要开展对体系结构模拟器建模方法与 RISC-V 处理器系统建模的研究。

本设计采用 gem5 模拟器来对 RISC-V 处理器进行模型构建。gem5 是一个具有高度可扩展性和灵活性的开源体系结构模拟器，支持多种指令集体系结构（Industry Subversive Alliance，ISA），可模拟多种处理器架构和系统级组件的行为。gem5 模拟器基于 C++语言来创建具有严谨继承关系的模拟对象，并通过脚本文件将各个模拟对象通过虚拟的总线和设备连接在一起，构建出一个完整的模拟环境。gem5 中提供了四种 CPU 模型，分别为 AtomicSimple，TimingSimple，Minor 和 O3（Out-of-Order）。四种模型的内部结构与设计理念各不相同，并可以由开发人员根据设计需要进行自由配置与修改。考虑到本设计中的处理器采用三级流水线结构，本设计选取与之最为相似的 Minor CPU 模型为基础，研究其具体结构组成，采用重构 Minor CPU 模型的方式对低功耗 RISC-V 处理器核进行建模。

本设计中的构建的 CPU 模型如图 3-2 所示，主要包含三级流水线结构。

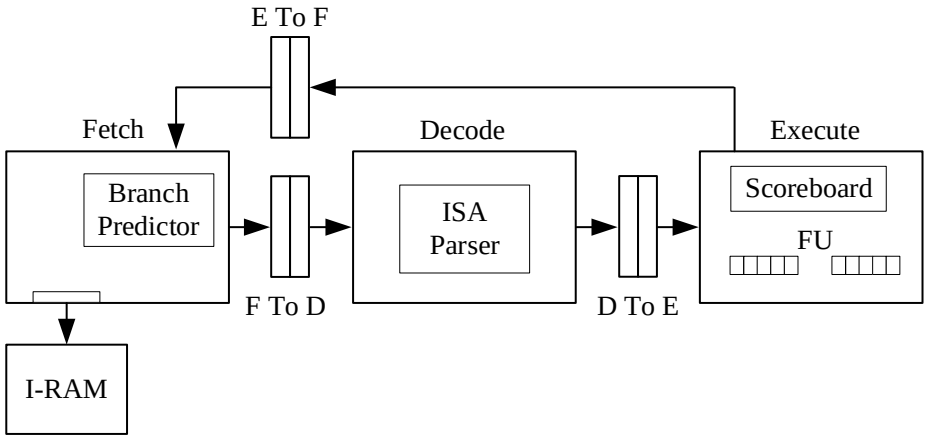

图 3-2 gem5 模拟器中的 CPU 模型示意图

第一级 Fetch 模块从指令 RAM 里取出指令，并将指令解码为宏指令后打包发送给 Decode 模块前的输入缓冲区。本模型中不同阶段间都具有输入缓冲区，

在每个周期都以接收到的相同形式存储数据，如果下一阶段尚未准备好处理该数据，则可以进行保存。Fetch 模块中还包括一个分支预测器（Branch Preditcor），并采用 BTFN 的静态预测方案。

第二级 Decode 模块接收来自 Fetch 模块的指令输入，并将这些指令解码成微指令，最后进行指令打包输出。其中具体的解码操作由指令集译码器（ISA Parser）实现，该译码器由 gem5 官方提供，对 RISC-V 指令集中的各指令进行具体解码。

第三级 Execute 模块执行所有的指令操作和存储器访问操作。从 Decode 模块传来的指令会先进入等待队列中，即模拟指令的派遣过程。之后再从等待队列中取出指令进行真正的运算执行，即模拟指令的发射过程。Execute 模块中包含了许多运算单元（Functional Unit，FU），模块中指令的派遣、发射、交付等操作都是在这些 FU 中完成的。指令记分牌（Scoreboard）对正在运行的指令数量进行计数，并统计用于后续功耗评估的相关信息。同时，该阶段会将指令实际执行的分支跳转结果反馈到 Fetch 模块中的分支预测器，用于判断是否需要更新指令流。

在完成 CPU 模型构建后，进一步构建 CPU 与 ISA 的交互模型。gem5 中的 ISA 模型与 CPU 模型是解耦的，内部包含 RISC-V、ARM 和 x86 等多种指令集的指令信息。ISA 模型与 CPU 模型之间通过动态指令（Dynamic Instruction）模型来实现交互，具体交互模型如图 3-3 所示。当模拟系统开始运行程序时，CPU 模型将指令中的寄存器或立即数的值传递给动态指令模型，存入动态指令模型中的源寄存器（Src Register）；之后 ISA 模型从动态指令模型中读取源操作数信息，并经过计算后将结果返回至动态指令模型；在指令的写回阶段，CPU 模型从动态指令模型中读取目的寄存器（Dest Register）的值，并写回内部寄存器组，从而实现指令完整周期的模拟。

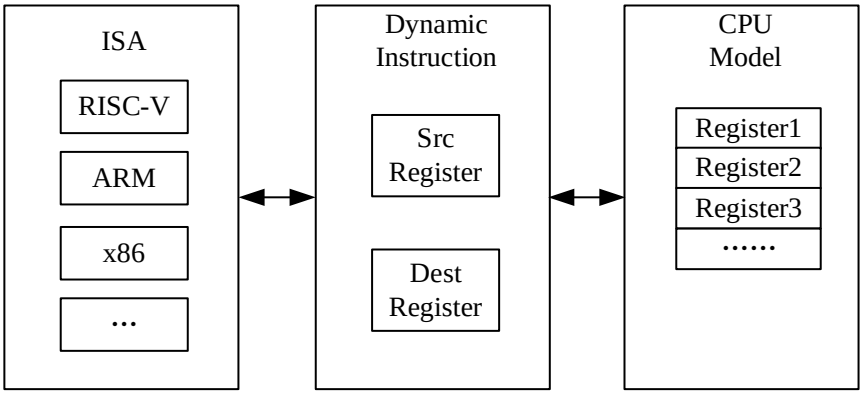

图 3-3 ISA 与 CPU 交互模型示意图

3.3 DVFS 系统仿真

在 gem5 模拟器中完成低功耗 RISC-V 处理器模型构建后，进行 DVFS 系统仿真。本设计中的 DVFS 系统仿真架构如图 3-4 所示。

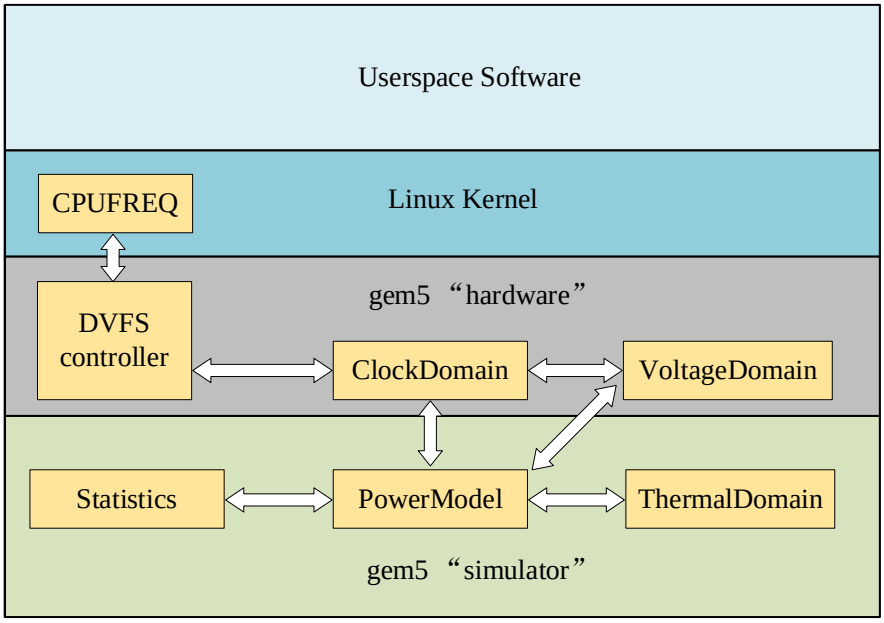

图 3-4 gem5 中的 DVFS 架构示意图

为了实现系统功耗的统计评估，首先构建 CPU 功耗估计模型。本设计使用 gem5 提供的 MathExprPowerModel 类，该类可以用数学公式表达某种电源状态下各模块的动静态功耗，并利用各模块的统计信息来进行具体计算。本设计中使用的动态功耗计算公式如公式 (3-1) 所示，静态功耗计算公式如公式 (3-2) 所示。

$$P_{\mathrm{dyn}} = V_{\mathrm{DD}}^{2} fA \tag{3-1}$$

$$P_{\mathrm{st}} = 4 \times Temp \tag{3-2}$$

式中 P_{dyn}——动态功耗(W)；

P_{st}——静态功耗(W)；

V_{DD}——电源电压(V)；

f——工作频率(Hz)；

A——每周期运行指令数；

$Temp$——当前工作温度(K)。

功耗评估模型中还需对电压与时钟域进行设置，根据本文的设计指标，将系统工作主频设为 50MHz，并配置了四种电压-频率对，分别为（1.8V，50MHz）、（1.8V，25MHz）、（1.8V，12.5MHz）、（1.6V，32.768KHz），可以根据不同工作场景进行动态调节。

在处理器功耗评估模型建立完成后，本设计采用 gem5 中的全系统（Full System，FS）仿真模式进行仿真，该模式将在模拟器上模拟整个 Linux 系统的启动，并通过仿真数据来验证系统设计和实现的正确性和可靠性。在完成配置 gem5 运行环境、指定系统内核镜像和 rootfs 文件系统、设置仿真平台的相关参数等步骤后，启动 RISC-V 全系统仿真。如图 3-5 所示，本设计将构建好的处理器模型应用于 gem5 中的 RISC-V 全系统仿真平台[46]，使用开源的 UCanLinux 内核进行 FS 仿真。

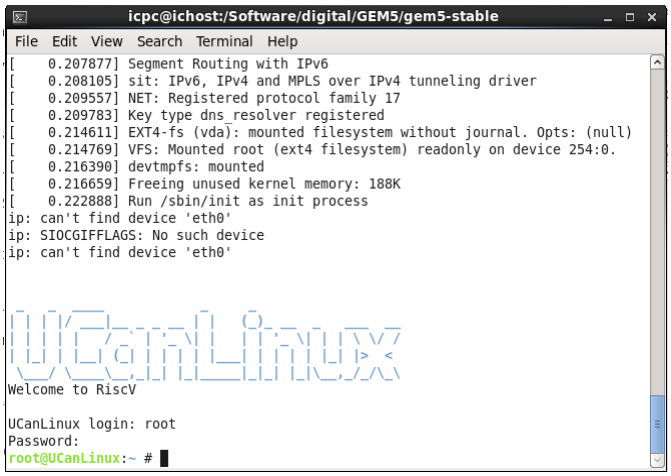

图 3-5 RISC-V 全系统仿真界面图

系统部署完成后，使用 CPUFREQ 软件接口进行 DVFS 仿真。该程序可以实时监测 CPU 的工作负载，并提供了几种不同的 CPU 调节策略，具体如表 3-1 所示。

表 3-1 CPU 调频策略

策略名	策略说明
performance	始终以最高频率运行
ondemand	平时以低速方式运行，当系统负载提高时自动提高频率
conservative	与 ondemand 类似，但在提高频率时渐进式提高

本设计对以上三种策略进行仿真模拟，以 STAMP 基准测试程序作为处理器运行负载[47]，观察不同调度策略对处理器性能和功耗的影响，以确定最佳调度方案。如图 3-6 所示，gem5 模拟器可以将仿真对象的一些量化指标如当前仿真时间、当前工作频率、工作电压、每周期运行指令数、系统功耗等数据实时记录并打印输出，本设计中通过观察分析这些统计数据来实现对不同调频策略的比较评估。

```
14      clk_domain.clock          20000     # Clock period in ticks (Tick)
24      cpus.ipc                  0.572198  # IPC: instructions per cycle ((Count/Cycle))
805     cpus.dynamicPower         0.0097    # Dynamic power for this object (Watts) (Watt)
1214    voltage_domain.voltage    1.8000    # Voltage in Volts (Volt)

1433    clk_domain.clock          40000     # Clock period in ticks (Tick)
1443    cpus.ipc                  0.617277  # IPC: instructions per cycle ((Count/Cycle))
2173    cpus.dynamicPower         0.0016    # Dynamic power for this object (Watts) (Watt)
2550    voltage_domain.voltage    1.6000    # Voltage in Volts (Volt)
```

图 3-6 gem5 部分原始统计数据图

本设计对待测 CPU 系统在相同负载下、启用不同调频策略时完成测试程序的所用时间和运行功耗进行了对比，具体如图 3-7 和图 3-8 所示。从测试结果可以看出，performance 调频策略完成任务用时最短。这是由于当系统工作在 performance 调频策略下时，系统始终以最高工作频率与工作电压运行，因此该调频策略下消耗的平均功耗也明显高于其他两种调频策略。

ondemand 调频策略比较激进，当检测到系统中负载变化时，系统会先对适合的工作电压和频率作出预测，之后直接将频率与电压调整至对应档位。当 CPU 处于轻负载状态时，频率会降低以降低功耗，而当 CPU 处于重负载状态时，频率会增加以提高性能。ondemand 策略下的状态切换是极其灵敏迅捷的，系统负载变化时会立即作出反应，向目标状态进行切换。例如，当系统从空闲状态转换至高负载状态时，频率与电压可能直接从最低档升至最高档。结果表明系统在此调频策略下的完成任务耗时仅比 performance 调频策略下略长一些，平均功耗则有显著降低。

conservative 调频策略与 ondemand 策略类似，但在变化时更为保守，只会在临近的电压-频率对之间进行切换。这意味着在系统繁忙时，conservative 调频策略需要更多的时间才能将系统调整至最高工作频率，需要比 ondemand 策略花费更多的时间开销。从测试结果也可看出，该策略下系统完成任务耗时最长，所用平均功耗也略高于 ondemand 策略。

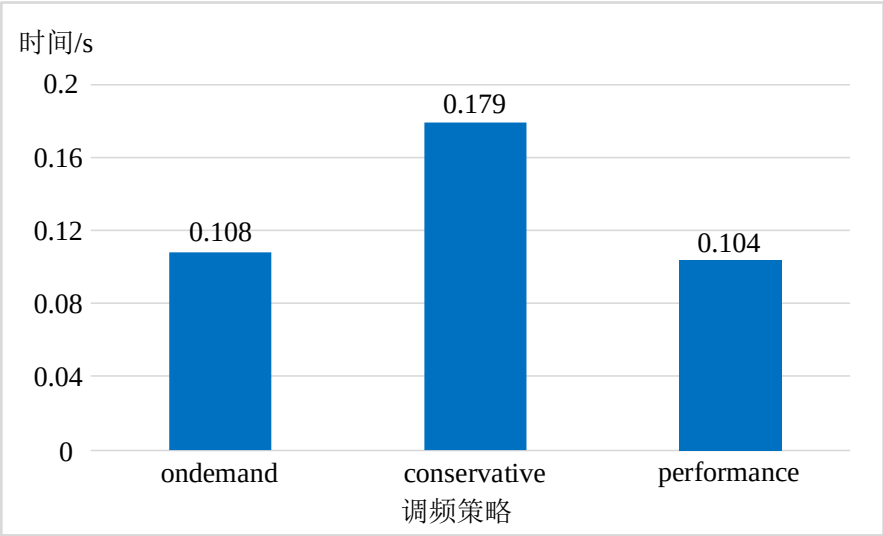

图 3-7 相同负载下不同调频策略运行时间比较结果图

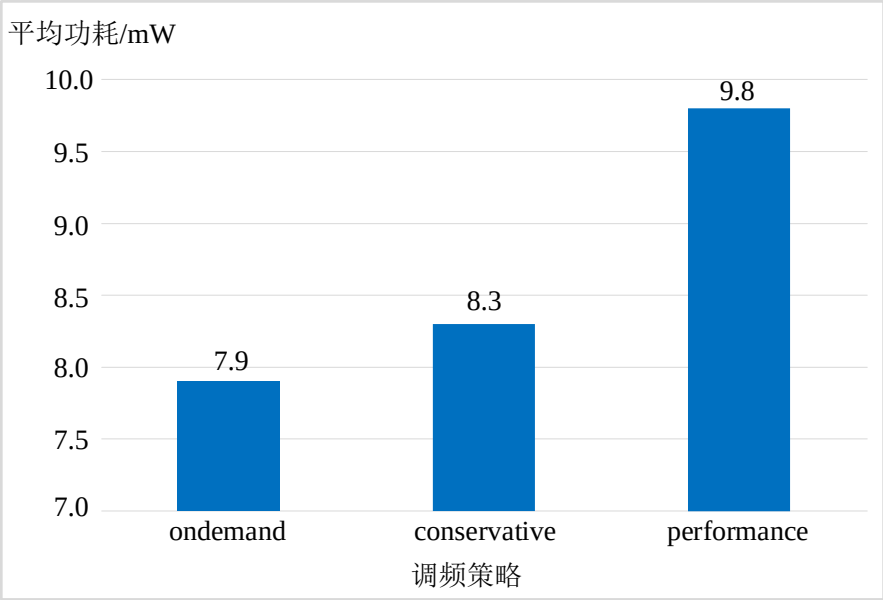

图 3-8 相同负载下不同调频策略平均运行功耗比较结果图

综合上述分析，调节方式更为激进的 ondemand 调频策略可以以较小的切换时间为代价，显著地减少系统运行的平均功耗，更适用于本系统。

3.4 RISC-V 处理器的设计与实现

在完成模拟器中的 RISC-V 处理器建模以及 DVFS 系统仿真研究工作后，进行 RISC-V 处理器核的 RTL 设计。本设计中的低功耗 RISC-V 处理器总体架构已在 3.1 节中给出，本节将具体说明各单元的设计内容。

3.4.1 取值单元设计

取指单元需要以最快的速度从存储器中连续不断地取出指令以提供给处理器核执行。本设计中没有采用延迟不确定的指令缓存，而是采用低延迟的内嵌指令紧耦合存储器（Instruction Tightly Coupled Memory，ITCM）来保证取指的高速度和实时性。

取指单元的效率直接影响着整个 CPU 的性能表现，本设计中的取指单元如图 3-9 所示，取指单元通过总线接口单元 BIU 从 ITCM 中连续不断地取出指令，同时生成下一条待取指令的 PC 值并更新流水线寄存器。此外，为了能够预测分支指令的跳转地址，取指单元还对当前指令进行部分译码，并判断其是否为分支跳转指令。如果当前指令是分支跳转指令，取指单元需要将该指令送到 BPU 模块进行 BTFN 分支预测，并将分支预测的跳转地址作为下一条待取指令的 PC 值；如果当前指令不是分支预测指令，则将该指令 PC 与指令自增值相加作为下一条待取指令的 PC 值。本设计中，16 位指令的指令自增值为 2，32 位指令的指令自增值为 4。如果发生了流水线冲刷，则使用执行单元传回的新 PC 值作为待取指令 PC 值。取指单元在每个时钟周期都会完成上述判断流程，有序生成待取指令，保证取指过程的连续不断，从而提高处理器性能。

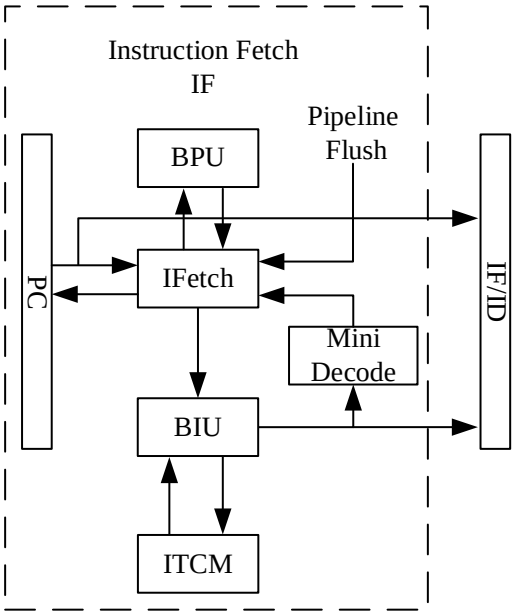

图 3-9 取指单元结构示意图

本设计支持 RV32IMAC 指令集，而其中的压缩指令子集中的指令宽度为 16 位，其余指令均为 32 位，因此 IF 单元在取指过程中需要处理 32 位指令与 16 位指令混合的指令流。IF 单元每次从指令存储器中取指的宽度固定为 32 位，因

此每次取指都可能遇到非对齐指令，需要对非对齐取指的情况进行分类处理，具体如图 3-10 所示，共有五种形式。

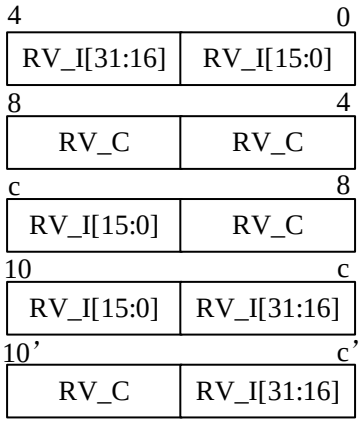

图 3-10 混合指令存储组合图

第一种为直接取到一条 32 位的完整指令，这种情况最为简单，无需进行额外操作。第二种为取到连续两条完整的 16 位指令，这种情况下指令的地址边界仍是对齐的，将依次执行两条完整的压缩指令。第三种情况下，指令存储空间的低 16 位存储的是一条完整的 16 位压缩指令，高 16 位存储的是另一条 32 位指令的低 16 位，这种情况下发生了指令地址边界的非对齐。第四种情况中，指令存储空间的低 16 位存储的是某一条 32 位指令的高 16 位，而高 16 位存储的是另一条 32 位指令的低 16 位。第五种情况与第四种相似，此时与该 32 位指令的高 16 位在存储空间中相连的为另一条 16 位压缩指令。

为了解决上述情况中的非对齐指令取指，本设计采用剩余缓存技术来从混合指令存储空间中取指。如果在取指过程中检测到了非对齐指令取指，则将取到指令的高 16 位存入剩余缓存中，在下一次取指过程中再将剩余缓存中的指令与新取到指令的低 16 位进行拼接，从而得到完整的 32 位指令。这种方法可以处理上述全部的五种情形，并且都可等效为在一个取指周期内取回了一条 32 位完整指令，不会造成性能的损失。

3.4.2 译码单元设计

译码单元主要负责对指令中包含的具体信息进行解析，以识别出该指令的类型和需要使用的寄存器等相关信息，之后从通用寄存器组（RegFile）中将所需操作数读出，并将指令派遣（Dispatch）到运算单元。本设计中的译码单元结构如图 3-11 所示。

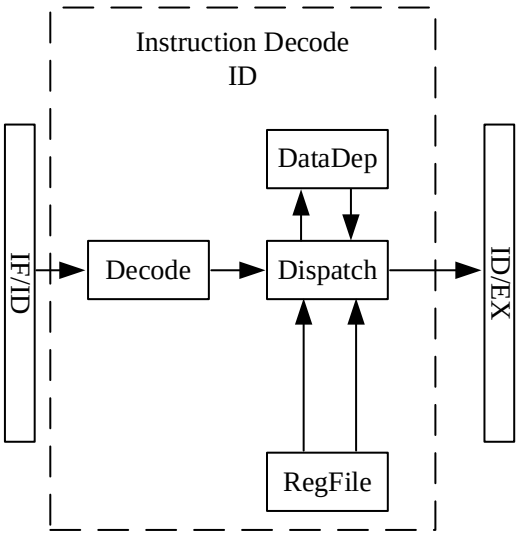

图 3-11 译码单元结构示意图

Decode 模块为由组合逻辑电路组成的 RISC-V 译码器，按照 RISC-V 指令集的编码格式进行翻译。依照 RISC-V 架构规整的指令格式，Decode 模块可以取出 32 位指令或 16 位指令中的关键编码段，极大地简化了硬件电路结构。在得到各指令的编码信息后，将各指令信息打包成为信息总线，传递给 Dispatch 模块派遣到各运算单元。

本设计中的 RegFile 模块结构图如图 3-12 所示，其中定义了 32 个整数通用寄存器，并通过例化专门的 D 触发器实现。当输入端收到来自译码器输出的源操作数寄存器地址 addr_op1 与 addr_op2，寄存器组内的对应寄存器将需要读出的操作数 op_data1 与 op_data2 输出。当输入端的写回信号 wen 使能有效时，向寄存器组内对应寄存器写入需要写回的数据 wdata。

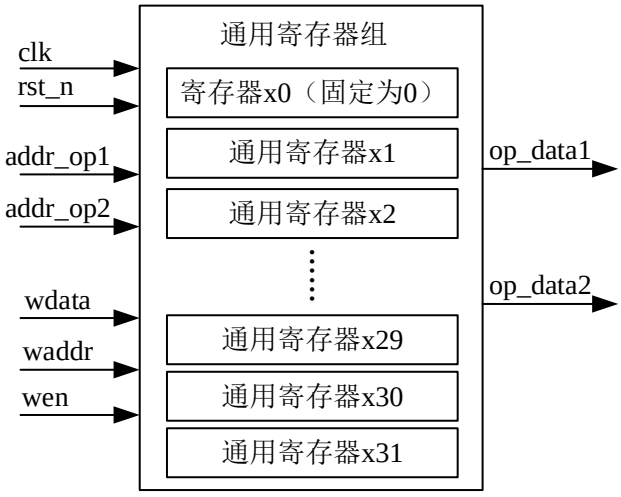

图 3-12 RegFile 微结构图

3.4.3 执行单元设计

执行单元根据指令的操作类别将指令发送给不同的运算单元进行执行，并完成指令的交付、访存和写回。本设计中的执行单元如图 3-13 所示。EX 单元完成指令的执行，其中 ALU 模块执行算术逻辑运算，LSU 模块处理存储器访问指令，Commit 模块完成指令的交付，WB 模块完成指令的写回。ALU 模块包含多个功能单元，执行不同类型的逻辑运算指令。Commit 模块中会对分支预测指令进行解析，若该指令的预测结果与实际情况不符，即出现分支预测错误，则向 IF 单元发出流水线冲刷的请求。WB 模块作为整个流水线结构的最后一级，将指令执行结果写回通用寄存器组。

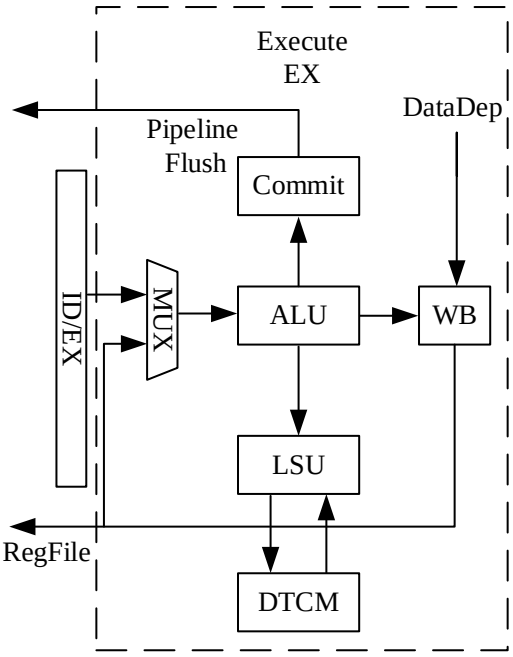

图 3-13 执行单元结构示意图

本设计中的 ALU 单元的微结构如图 3-14 所示，共有五个功能单元。普通 ALU 运算（Regular-ALU）单元主要处理 RISC-V 指令集中的一般逻辑运算、加减法计算和移位操作等指令；多周期乘除法器单元中没有设计独立的运算单元，与 ALU 模块中的其它模块共享一个数据通路，经过多个周期完成乘除法指令，以节省面积开销；地址生成单元（Address Generation Unit）负责处理读写指令并计算访存地址，同样通过复用 ALU 的加法器实现；分支预测解析（Branch and Jump resolve）单元负责处理 RISC-V 指令集中的六种带条件跳转指令；CSR 读写控制单元负责 CSR 读写指令的执行。这五个功能单元本身都不存在独立的运算电路，它们只产生对共享运算通路的操作请求与运算控制，通过这种资源复

用的方式节省了面积开销。

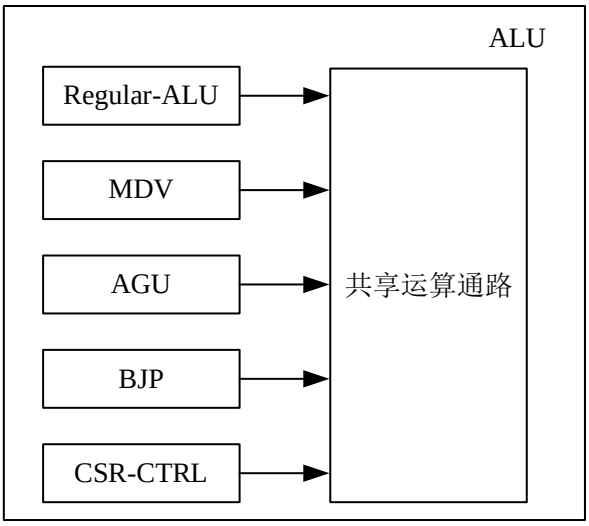

图 3-14 ALU 微结构图

指令在 ALU 中执行完成后，需要进行交付。若指令本身没有发生分支预测错误，且没有产生中断和异常，即可直接完成交付。而对于分支预测指令，需要在 Commit 模块中判断分支预测结果与真实跳转结果，若二者相符，则不需要进行流水线冲刷，指令成功交付；若二者不符，则需要冲刷流水线，根据跳转结果重新生成下一条指令的 PC 值。

指令执行的最终阶段为指令的写回，需要将各运算单元的执行结果写回至 Regfile 中。WB 模块中负责对写回指令的优先级进行仲裁，在本设计中，需要多周期执行的长指令比当前 ALU 中的指令更早执行，因此其写回的优先级也高于单周期指令。此外，写回单元中还需要维护并解决流水线中的数据冲突。根据第二章中的分析，流水线中可能出现的数据冲突主要有 WAR、WAW 和 RAW 三种，需要逐个分析解决。

本设计中采用了一种混合的写回策略，以解决数据冲突，同时在平衡性能和功耗的矛盾方面也具有良好的效果。对于单周期指令，本设计采用顺序派遣、顺序执行、顺序写回的策略，由指令派遣和写回的顺序可知不可能出现 WAR 和 WAW 相关性。在检测 RAW 相关性方面，本设计采用了数据前递的方法，将 RAW 相关性信息发送给运算模块，若发生冲突则暂停流水线直到相关性解除。对于多周期指令，采用顺序派遣、乱序执行、乱序写回的策略，同样根据派遣和写回的顺序可确保不会出现 WAW 相关性，RAW 和 WAR 相关性则由 DataDep 模块负责维护。这种混合的写回策略为指令执行提供了更加高效和可靠的解决方案，同时避免了指令执行过程中的数据冲突问题。

对于多周期指令，本文设计了一个 DataDep 模块，如图 3-15 所示，DataDep 模块内部包含一个 FIFO，当派遣多周期指令时，DataDep 模块会将该指令的 rd、rs1 和 rs2 等关键寄存器索引存入该 FIFO 的一个表项中，这个表项会随着长指令被派遣到运算单元中，并与其一起写回。只有在这条长指令被成功写回之后，此表项才会被移除。每条指令在派遣时，都会与 DataDep 中存储的寄存器索引进行相关性比较，若出现数据相关性，则阻塞流水线至相关性解除，以此保证不同长指令的派遣与写回顺序保持一致。

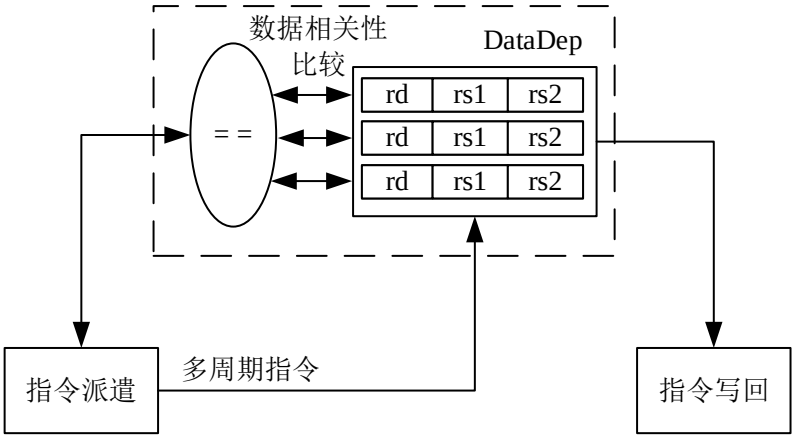

图 3-15 DataDep 微架构示意图

3.5 本章小结

本章开展了 RISC-V 处理器的建模仿真和设计实现的研究，其中包括了 RISC-V 处理器的总体架构、系统级模型构建、DVFS 系统仿真以及 RISC-V 处理器的 RTL 设计几部分内容。

首先根据本设计面向物联网应用的实际需求，确定了本设计中 RISC-V 处理器的总体架构，并根据此总体架构在体系结构模拟器 gem5 中进行了模型构建。之后将该处理器模型应用于 DVFS 系统仿真，并对几种 DVFS 调频策略进行了对比，确定了 DVFS 实现方案，为后续低功耗 SoC 中的 DVFS 技术实现提供了合理依据。最后完成了三级流水线结构的 RISC-V 处理器核设计。

第4章 RISC-V 架构 SoC 的设计与实现

处理器核需要配套的 SoC 才能在实际应用场景中实现全部功能。本文针对物联网应用场景对低功耗系统级芯片的需求，在完成了 RISC-V 处理器的设计后，提出了面向物联网应用的 RISC-V SoC 设计方案，将处理器核和外设集成到一起，实现了系统级设计。在此基础上，本文开展了针对 RISC-V SoC 的低功耗设计，合理划分了电源域与时钟域，采用了 DVFS 技术对系统进行优化，通过动态地调整电压和频率来实现低功耗的系统设计，满足物联网应用场景对低功耗系统级芯片的需要。

4.1 RISC-V 架构 SoC 整体结构

本设计实现的 SoC 整体结构如图 4-1 所示。该 SoC 使用的处理器核心为本课题设计开发的低功耗 RISC-V 处理器核，片上系统总线采用开源的 ICB 总线，外设部分包括异步收发器 UART、串行外设总线 SPI、通用输入/输出 GPIO、存储器 ROM、看门狗定时器 WDT 和实时计数器 RTC。

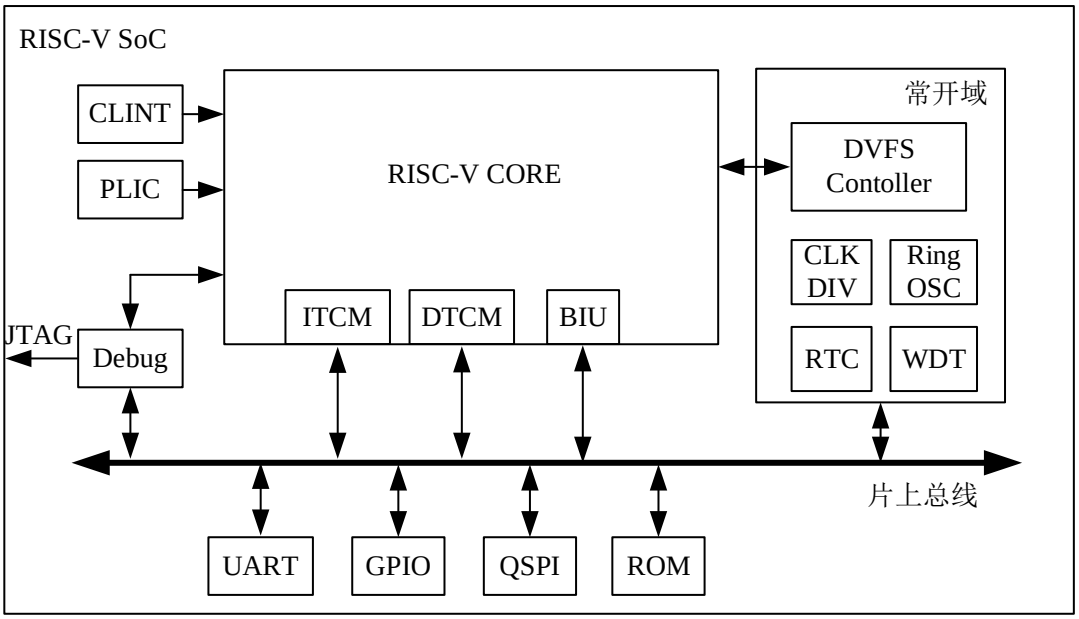

图 4-1 RISC-V SoC 整体结构示意图

本 SoC 的总线地址分配如表 4-1 所示，为 SoC 中的各部分外设分配了地址空间，同时预留了部分空闲地址空间，便于后续扩展开发。

表 4-1 RISC-V SoC 地址分配表

模块名称	地址区间	描述
调试模块	0x0000_0000-0x0000_FFFF	用于调试器使用
CLINT	0x0200_0000-0x0200_FFFF	CLINT 模块寄存器地址空间
PLIC	0x0C00_0000-0x0C00_FFFF	PLIC 模块寄存器地址空间
ITCM	0x8000_0000-0x8000_FFFF	ITCM 地址区间
DTCM	0x9000_0000-0x9000_FFFF	DTCM 地址区间
常开域	0x1000_0000-0x1000_FFFF	包括 RTC、WDT 等外设
GPIO	0x1001_0000-0x1001_FFFF	GPIO 地址区间
UART	0x1002_0000-0x1002_FFFF	UART 地址区间
QSPI	0x1003_0000-0x1003_FFFF	QSPI 地址区间
ROM	0x1004_0000-0x1004_FFFF	ROM 地址区间

4.2 RISC-V SoC 中断异常模块设计

中断和异常是 RISC-V 指令集架构中非常重要的一部分。控制和状态寄存器（Control and Status Registers，CSR）是由 RISC-V 指令集架构规定的处理器核内部的一系列寄存器，如表 4-2 所示，这些 CSR 寄存器位于执行单元中，严格按照 RISC-V 架构规范来定义各个 CSR 寄存器的具体功能，这些 CSR 寄存器将在异常处理过程中发挥作用。

表 4-2 中断和异常相关寄存器

寄存器名称	寄存器全称	描述
mtvec	机器模式异常入口基地址寄存器	进入异常的程序 PC 地址
mcause	机器模式异常原因寄存器	进入异常的原因
mtval	机器模式异常值寄存器	进入异常的信息
mepc	机器模式异常 PC 寄存器	保持异常的返回地址
mstatus	机器模式状态寄存器	该寄存器中的 MIE 域和 MPIE 域反映全局中断使能
mie	机器模式中断使能寄存器	控制不同类型中断的局部使能
mip	机器模式中断等待寄存器	反映不同类型中断的等待状态

当发生异常时，处理器需要立即停止当前程序的执行，并跳转到预设的异常服务程序地址执行，以处理异常事件。为了实现这一功能，处理器内部需要

实现异常处理机制。本设计中采用的异常处理过程如图 4-2 所示。

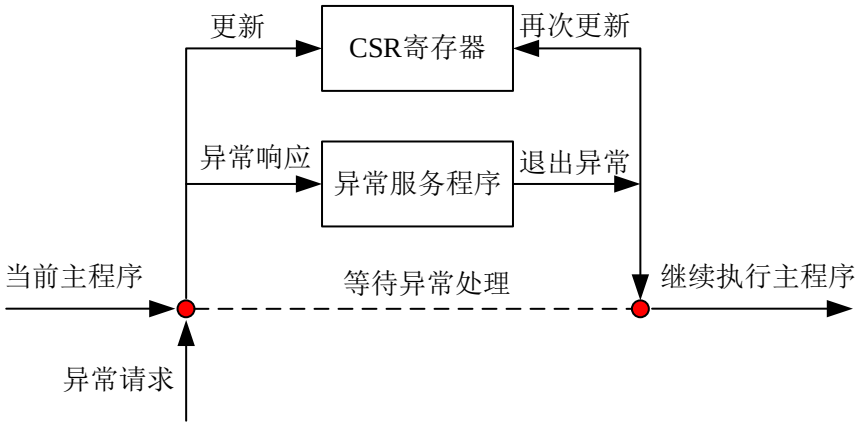

图 4-2 异常处理过程示意图

当异常发生时，处理器会首先根据 mtvec 寄存器中的值跳转到对应的异常服务程序地址，执行异常服务程序，并根据异常类型更新相应的 mcause、mepc、mtval 和 mstatus 等 CSR 寄存器。异常服务程序会根据异常的具体类型，选择对应的异常处理程序进行处理。完成异常处理后，处理器会执行专门的 MRET 指令返回到之前的程序执行位置，并再次更新 mstatus，恢复之前的状态。

在 RISC-V 处理器中，中断的管理由两个部分完成：处理器核局部中断控制器（Core Local Interrupts Controller，CLINT）和平台级别中断控制器（Platform Level Interrupt Controller，PLIC）。CLINT 是一个处理器内部的模块，负责处理本地软件中断和计时器中断，它包含多个定时器，可以产生周期性中断事件。PLIC 位于处理器外，负责处理外部中断，它可以连接 GPIO、UART 等多个外设，并对这些外部中断源产生的中断进行管理。当外设产生中断事件时，PLIC 会向处理器内部发送中断请求信号，处理器根据优先级和中断类型响应相应的中断事件。

本设计中的处理器核与中断模块的交互关系如图 4-3 所示，处理器核接收到来自 CLINT、PLIC 及调试模块的四种中断信号请求后，将其作为一种异步异常来处理。在中断响应阶段，处理器核首先识别中断的来源，保存当前处理器状态，然后跳转到异常处理程序开始执行。在异常处理阶段，处理器核根据中断来源执行相应的中断处理程序。例如，当来自 CLINT 的定时器中断发生时，处理器核将跳转到处理定时器中断的中断处理程序中，并执行相应的操作以处理定时器中断。在处理程序执行完毕后，使用 MRET 指令将控制权返回给原始程序，并将处理器状态恢复到中断前的状态，从而使主程序继续执行。

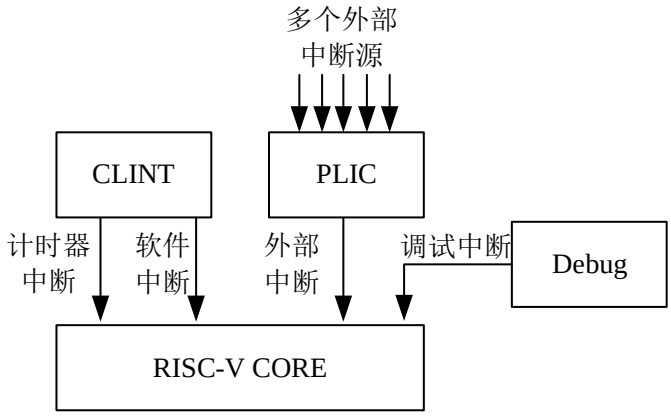

图 4-3 中断模块与处理器核的交互示意图

4.3 RISC-V SoC 调试模块设计

调试功能对于处理器系统来说极其关键，而调试机制及其软硬件实现都较为复杂。本设计采用 RISC-V 基金会官网上公布的 SiFive 公司的调试方案，其具体方案如图 4-4 所示。

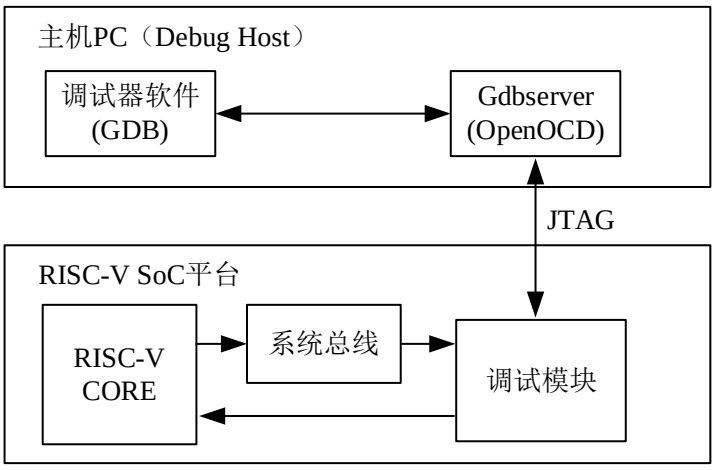

图 4-4 RISC-V 系统调试方案示意图

本方案使用主机 PC 作为整个平台中的调试上位机，在 PC 中运行调试软件（GNU Project Debugger，GDB），并使用 OpenOCD 软件作为 Gdbserver 来进行通信。之后通过 JTAG 接口连接到 SoC 平台上的调试模块，实现调试主机与 RISC-V SoC 硬件平台的通信交互。

其中，调试模块主要实现两部分功能。一部分是将从调试接口传入的 JTAG 标准信号解析成系统内的调试信号，另一部分是向处理器核发送调试中断。一

旦处理器核接收到调试中断信号请求，即会立即发送中断当前流水线的请求信号，取消所有后续指令，并从预先指定的地址 0x800 开始取指令。在进入调试模式后，处理器核将当前执行指令的 PC 值保存在调试 PC 值寄存器 dpc 中，并将触发调试的原因记录在调试原因寄存器 dcsr 中。一旦完成调试任务，通过执行 dret 指令退出调试模式。在退出调试模式时，处理器核直接跳转至 dpc 寄存器中存储的 PC 值，同时清除 dcsr 寄存器中的相关值，以便程序能够回到进入调试模式之前的程序断点并继续执行。

4.4 RISC-V SoC 低功耗设计

根据第 2 章中对于低功耗设计层次的分析，设计层次越高，功耗优化的效果越佳。因此本设计重点从系统级角度出发，进行 RISC-V SoC 的低功耗设计，主要包括整个 SoC 的电源域与时钟域划分和 DVFS 控制器设计两部分。

4.4.1 RISC-V SoC 电源域与时钟域设计

本设计中电源域与时钟域划分如图 4-5 所示，其中 RISC-V 处理器核、片上总线及主要外设等都位于主域中，DVFS 控制器、环路振荡器、看门狗定时器 WDT 和实时计数器 RTC 位于常开域中。

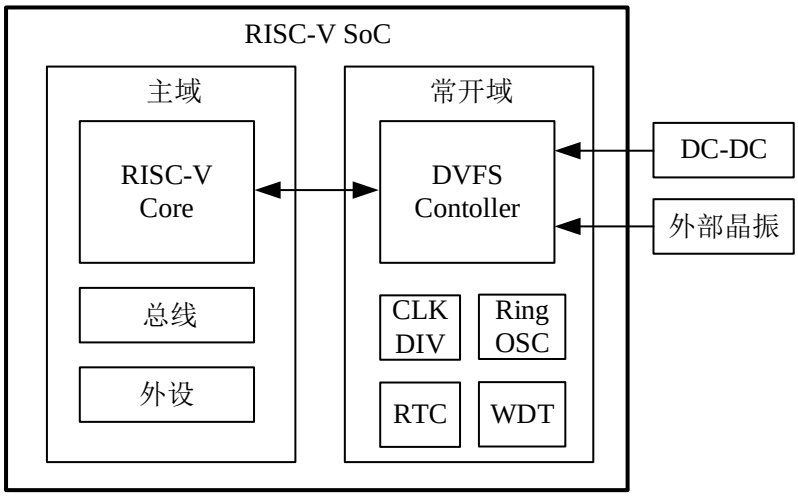

图 4-5 RISC-V SoC 电源域与时钟域划分图

主域的电源电压与时钟频率均可变，在实际工作过程中受到 DVFS 控制器的控制，根据系统工作状态动态调节工作电压与工作频率。常开域中的电源电压与时钟频率均为恒定值，以保证环路振荡器及定时器等模块能够稳定工作。

本设计中的电源电压来自片外 DC-DC 芯片的电源输入，并由 DVFS 控制器来实现电源轨切换。时钟信号有两个输入来源，一是外部晶振产生的低频

32.768KHz 时钟，主要负责给看门狗定时器和实时计数器提供不间断的低速时钟信号；二是片内环路振荡器产生的 50MHz 高频时钟，提供系统满负载状态下能够运行的最高频率。此外，本设计中还包括一个时钟分频模块，该模块根据 DVFS 控制器的控制信号对高频时钟进行分频，得到系统工作所需的 25MHz 与 12.5MHz 频率的时钟信号。

4.4.2 RISC-V SoC DVFS 控制器设计

本设计通过一个 DVFS 控制器来实现对整个 SoC 的动态电压频率调节，其与处理器系统的交互关系如图 4-6 所示。DVFS 控制器直接与 RISC-V 处理器核交互，并控制时钟分频产生模块与片外 DC-DC 电源模块，使其为 SoC 提供对应的时钟频率与工作电压。

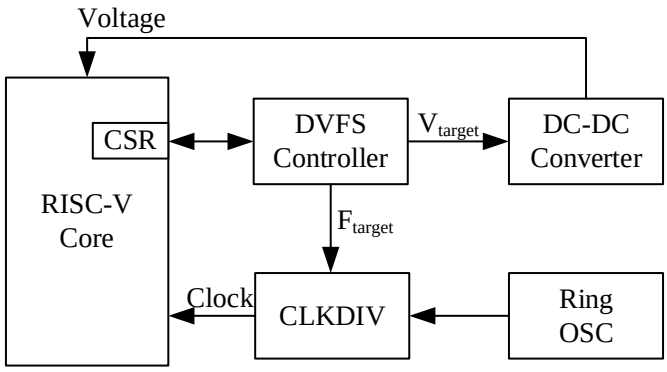

图 4-6 DVFS 控制器与处理器系统交互图

本设计以 RISC-V 处理器核中的 CSR 寄存器为 DVFS 控制器的接口，在处理器核的 CSR 地址空间内定义了与 DVFS 控制相关的寄存器，具体如表 4-3 所示。

表 4-3 DVFS 相关的控制寄存器

寄存器地址	寄存器名称	有效位宽	功能
0xBE0~0xBE3	dvfs_vftabx	[2:0]	频率电压表
0xBED	dvfs_vmode	[0:0]	当前工作电压档位
0xBEE	dvfs_fmode	[1:0]	当前工作频率档位

地址为 0xBE0-0xBE3 的 4 个寄存器为频率电压表 dvfs_vftab，其最高位代表电压档位，1’b1 表示 1.8V 电压，1’b0 表示 1.6V 电压；低两位表示四种频率档位，2’b11 表示最高档 50MHz，2’b10 表示次高档 25MHz，2’b01 表示次低档

12.5MHz，2’b00 表示最低档 32.768kHz。两者组合表示本设计中可用的四种电压-频率对。并且，各档位对应的具体数值用宏来表示，可由 cfg 文件进行配置。地址为 0xBED 的寄存器作为当前工作电压档位的指示位，地址为 0xBEE 的寄存器作为当前工作频率档位的指示位，DVFS 控制器可根据此指示位来判断系统当前工作状态。

DVFS 控制器通过实时读取处理器核工作负载状况，并结合系统当前工作状态，对系统的未来工作状态作出预测。具体来说，本设计中处理器核流水线内的译码、执行、访存等关键模块中都设置了活跃状态指示信号，这些信号将在其对应模块工作时使能，在空闲时清零。这些信号直接输入至 DVFS 控制器中，DVFS 控制器根据流水线中各模块的工作情况和当前工作状态，作出对系统目标工作状态的预测判断，若目标状态与当前状态不符，则会修改对应控制寄存器的值，并向时钟分频模块和片外电源接口输出对应的控制信号，使系统调整至目标工作状态。此外，在 DVFS 技术中，为了避免时序违例，调节电压和频率时需要保证两者变化的正确顺序。当电压请求升高时，需要保证当新的高电平电压稳定后再进行时钟频率的切换，而当电压请求降低时，需要在时钟频率降低之后才能降低电压。

在本设计中的 DVFS 控制器中，通过一个有限状态机来实现上述的状态转换功能，具体转换过程如图 4-7 所示。

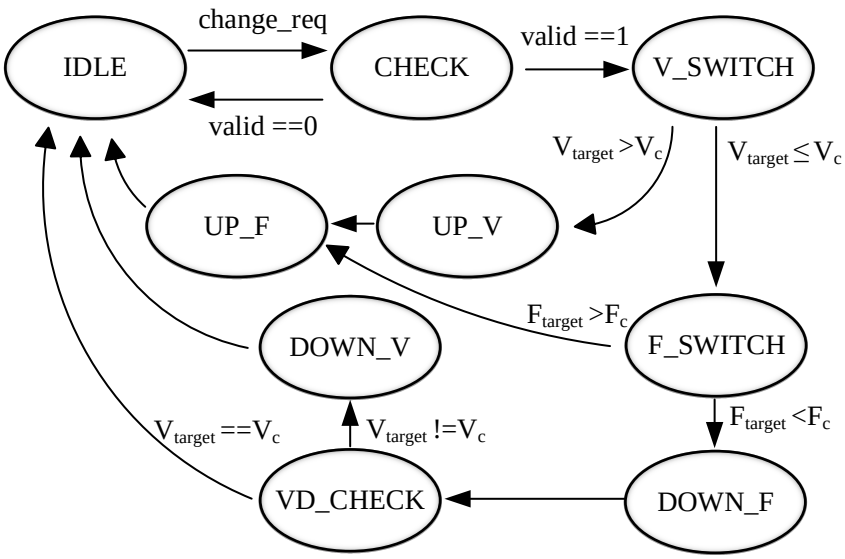

图 4-7 DVFS 控制器状态转换图

IDLE 状态表示 DVFS 控制器处于空闲状态，无需对 SoC 系统进行状态转换检测。当处理器核流水线中的活跃状态指示信号发生变化时，对 DVFS 控制

器发起状态转换请求信号 change_valid，DVFS 控制器进入 CHECK 状态，进行状态转换的检测，如果目标状态与当前状态不符则会进入下一状态，否则返回至空闲状态。当判断需要进行状态转换时，首先对系统电压值进行判断，进入 V_SWITCH 状态，若目标电压值大于当前电压值，则先后进入 UP_V 升压状态和 UP_F 升频状态，完成转换后回归至空闲态，继续等待系统负载变化。

若在 V_SWITCH 状态中判断目标电压值小于或等于当前电压值，则进一步进入 F_SWITCH 状态来进行时钟频率转换的检测。如果预测频率大于当前频率，则说明系统工作状态正在随负载增加而上调，因此无需进行是否下调电压的判断，直接进入 UP_F 升频状态；如果预测频率小于当前频率，则先进入 DOWN_F 状态下调频率，之后再判断电压值是否需要下调，若预测电压小于当前电压值，则进入 DOWN_V 降压状态，否则直接回归至空闲状态。经过这样的状态转换判断，能够在保证电压和频率的调节顺序正确的前提下，实现对系统工作状态的动态调节。

在状态转换的具体实现方面，本设计中的 DVFS 控制器将系统目标工作状态输出至时钟分频模块和片外调压模块。时钟分频模块根据此目标工作状态，对由片内环路振荡器产生的高频信号进行分频，产生系统所需频率的时钟信号；片外调压模块根据 DVFS 控制器的输出信号来调节输出电压。图 4-8 为系统从初始上电状态至正常工作状态的一段系统工作状态转换仿真图，可以看出系统状态从初始空闲状态逐步切换至高频高压工作状态。状态①为系统的初始上电状态，CPU 处于空闲状态，时钟频率为 32.768kHz，工作电压为 1.6V；状态②为系统开始运行程序，CPU 开始处理指令，状态由空闲状态开始向上切换，主域时钟频率也随之上升，工作电压上调；经过一段时间后，系统稳定在状态③，CPU 工作在高频工作状态，时钟处于最高频率 50MHz，工作电压保持在 1.8V；当系统负载减轻时，DVFS 控制器调整系统工作在状态④，此状态为系统工作的次高态，工作频率为 25MHz；如果此时负载再次加重，则系统回调至高频工作状态⑤。

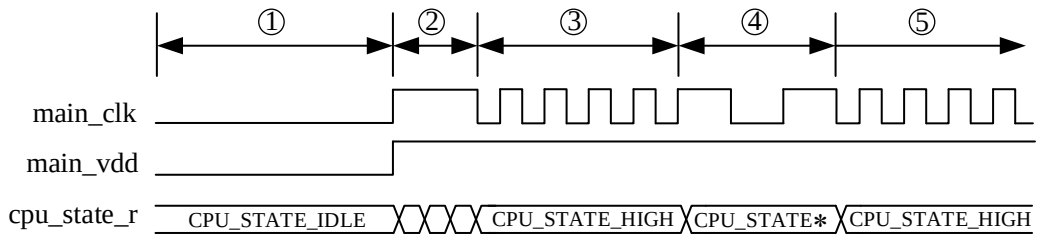

图 4-8 系统工作状态转换仿真图

4.5 本章小结

本章开展了对 RISC-V 架构 SoC 的 RTL 设计的研究，主要包括 SoC 整体结构、中断异常模块设计、调试模块设计、低功耗设计等方面的内容。首先确定了 RISC-V 架构 SoC 的整体结构，包括 RISC-V 处理器核、系统总线、外设接口等组成部分，结合中断异常处理模块和调试模块，共同实现了 SoC 的各种功能。

接下来本章重点开展了对 RISC-V SoC 低功耗设计的研究工作，主要包括电源域与时钟域的划分和 DVFS 控制器的设计。合理地划分电源域与时钟域，使得整个 SoC 各内部模块都能够工作在适宜状态，并最大限度地节省了功耗。DVFS 控制器通过与 RISC-V 处理器核交互，并控制时钟分频产生模块与片外 DC-DC 电源模块，使其为 SoC 提供对应的时钟频率与工作电压，实现了对整个 SoC 的动态电压频率调节。

第5章 RISC-V 架构 SoC 的仿真验证与物理实现

SoC 的设计往往具有较为复杂的功能和时序，因此对于 SoC 的开发设计而言，完善有效的仿真验证是 SoC 研发工作中极为重要的一环。本设计中，采用软硬件结合的验证方式，先在软件层面搭建验证平台测试整个系统的功能，之后再在 FPGA 开发板上进行板级验证，最后进行整个 RISC-V SoC 的物理实现与功耗评估。

5.1 验证平台搭建

仿真验证首先需要进行验证平台的搭建。本文对 RISC-V SoC 的搭建了编译环境和验证平台，整个仿真测试的工作流程如图 5-1 所示。

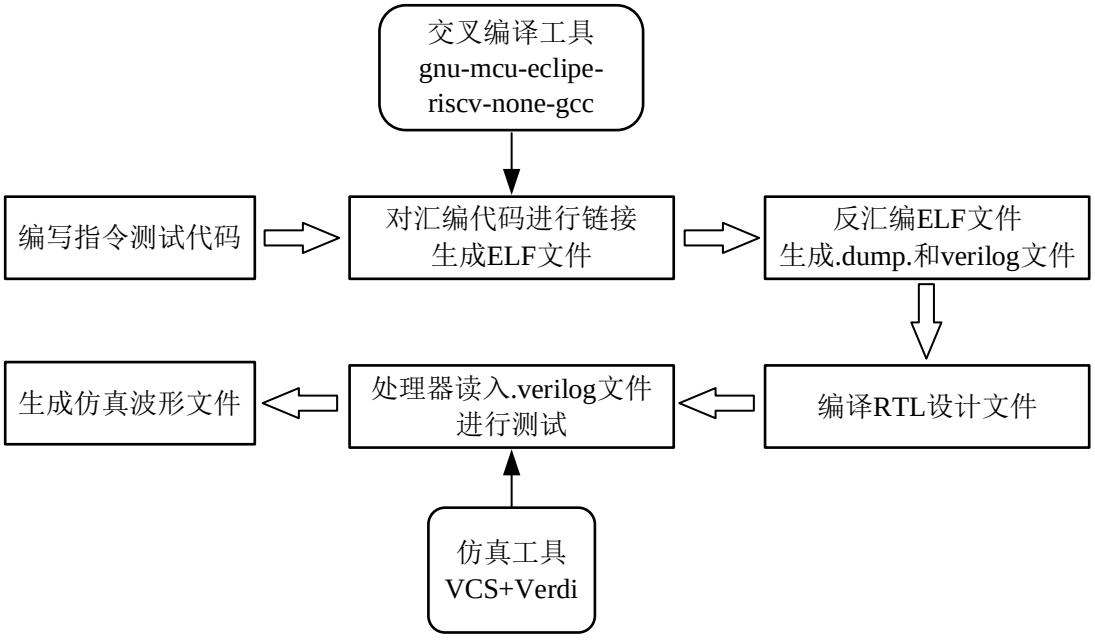

图 5-1 RISC-V 指令集测试流程示意图

首先编写用于指令集测试的汇编代码，但是这些由汇编语言编写的测试程序不能直接在处理器上运行，需要借助其他工具进行编译转换。本设计使用 RISC-V 架构提供的一个 GNU 工具链来实现编译和转换功能，该工具链包括交叉编译工具 gnu-mcu-eclipe-riscv-none-gcc，可将汇编代码转换为 RISC-V 架构所支持的指令集，最终编译完成的测试程序会生成一系列文件，其中包括格式为 Verilog 的文件，可以被测试平台读入进行后续仿真。之后使用 VCS 软件对 SoC 代码进行编译，处理器核将测试代码读入至 ITCM 中进行仿真，并联合 Verdi

工具查看仿真波形。

在上述测试流程中，还需对 SoC 进行仿真测试平台的开发。本文通过通过开发 TestBench 来实现 RISC-V SoC 的功能测试。在本设计的 TestBench 中，将 SoC 中的各个待测模块实例化为待测电路（Design Under Test，DUT），并开发测试用例对待测电路输入测试激励。同时，在 TestBench 中设置输出监测模块，以便观察待测电路的输出信号是否符号预期。本设计搭建的测试平台如图 5-2 所示。

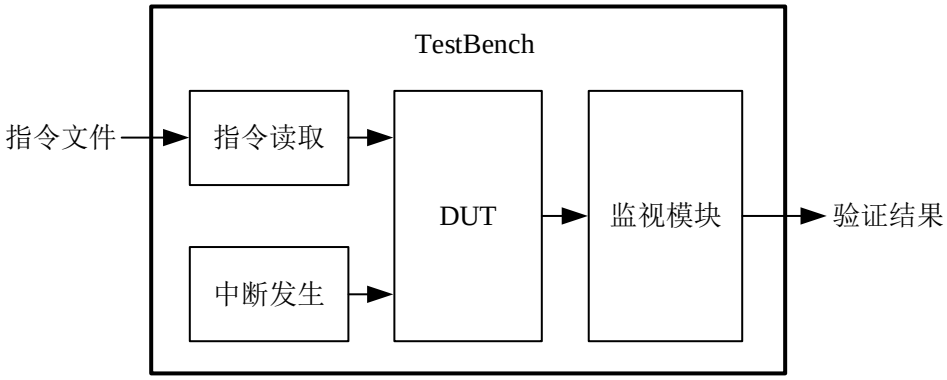

图 5-2 测试平台示意图

整个测试平台包括四个组成部分。指令读取模块通过 Verilog 语言中的 readmemh 函数读入指令文件，并作为激励输入至待测 SoC 的 ITCM 中。中断发生模块将随机地产生计时器中断、软件中断、外部中断及总线访问错误等模拟信号，用于测试待测 SoC 是否能够处理相关的中断和异常。监视模块将对整个测试过程进行实时监控，通过观察各寄存器中的值来判断验证过程是否成功，并将验证结果打印输出。这四个组成部分的协同工作，保证测试平台能够对待测 SoC 进行全面的验证，确保其符合设计要求。

5.2 RISC-V 指令集测试

riscv-tests 是一个针对 RISC-V 架构的开源测试框架，旨在提供全面、可靠的指令集和体系结构验证，以确保符合 RISC-V 指令集架构规范。该测试框架由 RISC-V 基金会维护，包含指令集测试集等多个子测试集。riscv-tests 测试用例由汇编语言编写，通过执行各种指令序列来验证 CPU 是否正确执行指令，确保 RISC-V 处理器实现的正确性和符合性[48]。

riscv-tests 包含的测试用例涵盖了各种不同的指令执行模式、流程控制和中断处理等，以确保处理器在不同情况下的稳定性和兼容性。例如，sub.S 测试程序主要对 RV32I 指令子集中的减法指令的进行了测试，其测试程序的部分代码

如图 5-3 所示。

```
#-------------------------------------------------------------
# Arithmetic tests
#-------------------------------------------------------------

TEST_RR_OP( 2,  sub, 0x0000000000000000, 0x0000000000000000, 0x0000000000000000 );
TEST_RR_OP( 3,  sub, 0x0000000000000000, 0x0000000000000001, 0x0000000000000001 );
TEST_RR_OP( 4,  sub, 0xfffffffffffffffc, 0x0000000000000003, 0x0000000000000007 );

TEST_RR_OP( 5,  sub, 0x0000000000008000, 0x0000000000000000, 0xffffffffffff8000 );
TEST_RR_OP( 6,  sub, 0xffffffff80000000, 0xffffffff80000000, 0x0000000000000000 );
TEST_RR_OP( 7,  sub, 0xffffffff80008000, 0xffffffff80000000, 0xffffffffffff8000 );
```

图 5-3 rv32ui-p-sub 测试程序节选图

该测试程序包含多组测试，每组测试都将进行一次减法运算，完成减法操作后将得到的计算值与参考值进行比较，例如在图 5-3 中的 4 号测试用例表示将数据 0x0003 减去 0x0007，并设定理论结果为 0xfffc，若两者不等则直接跳转至 fail 程序，若两者相等则继续进行下一项测试。在完成所有的测试用例后，将 x3 寄存器的值赋为 1。若进入 fail 程序，则将 x3 寄存器的值赋为 0。在测试程序运行结束前，验证平台检测 x3 寄存器的值是否为 1，并以此判断是否通过测试。

完成测试程序的编写后使用 gnu-mcu-eclipe-riscv-none-gcc 工具将测试程序编译为可被 TestBench 读入的二进制文件。继续以 sub 指令测试中的 4 号测试用例为例，其生成的反汇编文件如图 5-4 所示。测试开始时先使用 RISC-V 指令集中定义的伪指令 li（load immediate）将立即数 0x0003 存入通用寄存器 ra 中，再将立即数 0x0007 存入 sp 寄存器中，使用 sub 指令将 ra 寄存器中的值与 sp 寄存器中的值相减，并将结果存入 t5 寄存器中。接着，将理论结果-4 和测试向量编号 4 分别存入寄存器 t4 和 gp 中。最后使用 bne 指令将 t4 和 t5 寄存器中的值进行比较，若不等则跳转至 fail 程序段。

```
80000282 <test_4>:
80000282:       408d            li      ra,3
80000284:       411d            li      sp,7
80000286:       40208f33        sub     t5,ra,sp
8000028a:       ffc00e93        li      t4,-4
8000028e:       4191            li      gp,4
80000290:       35df1063        bne     t5,t4,800005d0 <fail>
```

图 5-4 rv32ui-p-sub 反汇编文件节选图

最后生成的可执行的二进制文件如图 5-5 所示。本设计中的 TestBench 使

用 readmemh 函数读入该二进制指令测试文件，并将其作为激励输入至待测 SoC，即可执行对应的指令测试程序。

```
@00000000
0D AA 01 00 73 10 05 34 17 25 00 00 13 05 85 FF
23 20 E5 01 23 22 F5 01 73 2F 20 34 63 41 0F 04
A1 4F 63 02 FF 07 A5 4F 63 0F FF 05 AD 4F 63 0C
FF 05 85 4F 63 05 FF 0B 95 4F 63 00 FF 0D 9D 4F
63 0B FF 0D 17 0F 00 80 13 0F CF FB 63 03 0F 00
02 8F 73 2F 20 34 63 53 0F 00 09 A0 1D A0 B7 0F
00 80 93 8F 3F 00 63 04 FF 05 B7 0F 00 80 93 8F
7F 00 63 0A FF 05 B7 0F 00 80 93 8F BF 00 63 0C
```

图 5-5 rv32ui-p-sub 二进制文件节选图

在测试程序运行结束前，TestBench 对 x3 寄存器的值进行检测，以此判断是否能够成功完成测试。若成功完成全部测试则输出“PASS”字符。经过仿真验证，指令集测试的全部测试用例执行完毕的结果示例如图 5-6 所示，可以看到本设计成功完成了本设计所支持的 RV32IMAC 指令集的全部测试。

```
PASS ../vsim/run/rv32um-p-mulh/rv32um-p-mulh.log
PASS ../vsim/run/rv32ui-p-bgeu/rv32ui-p-bgeu.log
PASS ../vsim/run/rv32mi-p-scall/rv32mi-p-scall.log
PASS ../vsim/run/rv32ui-p-add/rv32ui-p-add.log
PASS ../vsim/run/rv32ui-p-sltu/rv32ui-p-sltu.log
......
PASS ../vsim/run/rv32ui-p-beq/rv32ui-p-beq.log
PASS ../vsim/run/rv32ua-p-amominu_w/rv32ua-p-amominu_w.log
PASS ../vsim/run/rv32ua-p-lrsc/rv32ua-p-lrsc.log
PASS ../vsim/run/rv32um-p-rem/rv32um-p-rem.log
PASS ../vsim/run/rv32ui-p-xori/rv32ui-p-xori.log
ALL PASS，TEST OVER!
```

图 5-6 指令集测试结果汇总图

5.3 RISC-V SoC 的 FPGA 平台验证

FPGA 的英文全称为 Field Programmable Gate Array，作为一种半定制的集成电路，广泛用于芯片设计流程中的系统模拟验证。现代 EDA 软件可以方便地将电路设计进行综合并生成比特流文件，烧录进 FPGA 中后即可在 FPGA 板上模拟芯片的实际功能，并且可以快捷便利地进行修改，帮助开发人员在进行流片之前检测设计中的功能和实现问题，有助于减少集成电路设计开发的周期和成本。本节主要开展对 RISC-V SoC 的 FPGA 原型验证平台的搭建以及运行软件示例测试的研究。

5.3.1 FPGA 原型验证平台搭建

本文使用的 FPGA 开发平台为由 AVNET 公司开发的 ZedBoard 开发板，如图 5-7 所示。ZedBoard 作为一款基于 Xilinx Zynq-7000 系列 SoC 的开发板，具有高性能的处理能力和强大的可编程逻辑资源、丰富的可扩展性和可定制性，能够满足本设计需要。ZedBoard 板上搭载着 XC7Z020-1CLG484C 型号的芯片，具有超过 80000 个可编程逻辑片，还配备了大小为 512MB 的 DDR3 内存。此外，ZedBorad 上的外设和接口资源十分丰富，本设计主要使用的是 12V 电源接口、USB-JTAG 程序下载接口、按钮开关和 Pmod 接口。其中的 Pmod 接口是 Digilent 公司定义的一种扩展接口，本设计中使用 Pmod 接口作为 RISC-V SoC 的主要外设的输入和输出接口。

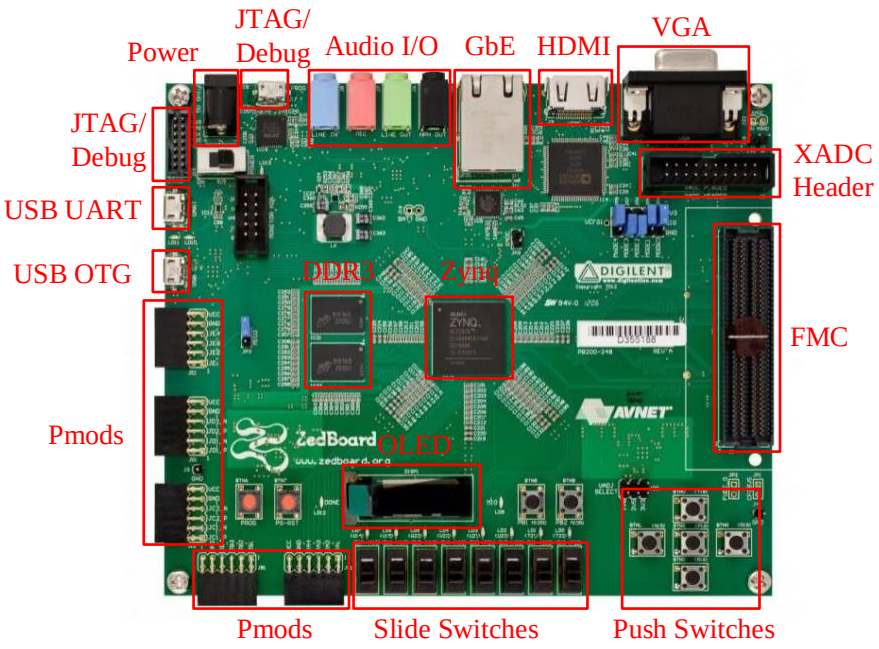

图 5-7 ZedBoard 实物图

为了能够正确地下载并调试 RISC-V SoC 中运行的程序，还需要在验证平台中加入调试器工具。本设计采用的是由 Sipeed 公司为 RISC-V 架构处理器开发的 Sipeed 调试器，如图 5-8 所示。

图 5-8 Sipeed 调试器实物图

该调试器通过杜邦线与 RISC-V SoC 中的 JTAG 接口连接，具体连接关系如图 5-9 所示。

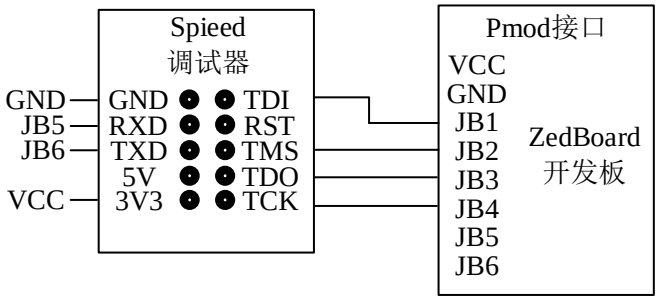

图 5-9 调试器与开发板连接关系

在 RISC-V SoC 顶层设计中，通过约束文件将 JTAG 接口连接至 FPGA 板上 Pmod 接口中的 JB1~JB4 引脚上，再与调试器上的标准 4 线接口连接。调试器的另一端连接 PC 上位机。完成 JTAG 接口的连接后，使用 Sipeed 调试器将 PC 端中已完成编译的测试程序下载至 RISC-V 处理器核中运行。此外，该调试器中包含一组 UART 接口，本设计同样通过约束文件将 SoC 顶层输出的 uart_rx 和 uart_tx 信号约束至 Pmod 接口中的 JB5 和 JB6 引脚，再通过调试器与 PC 相连，即可通过串口与上位机通信，并打印测试信息。最终搭建完成的 FPGA 原型验证平台如图 5-10 所示。

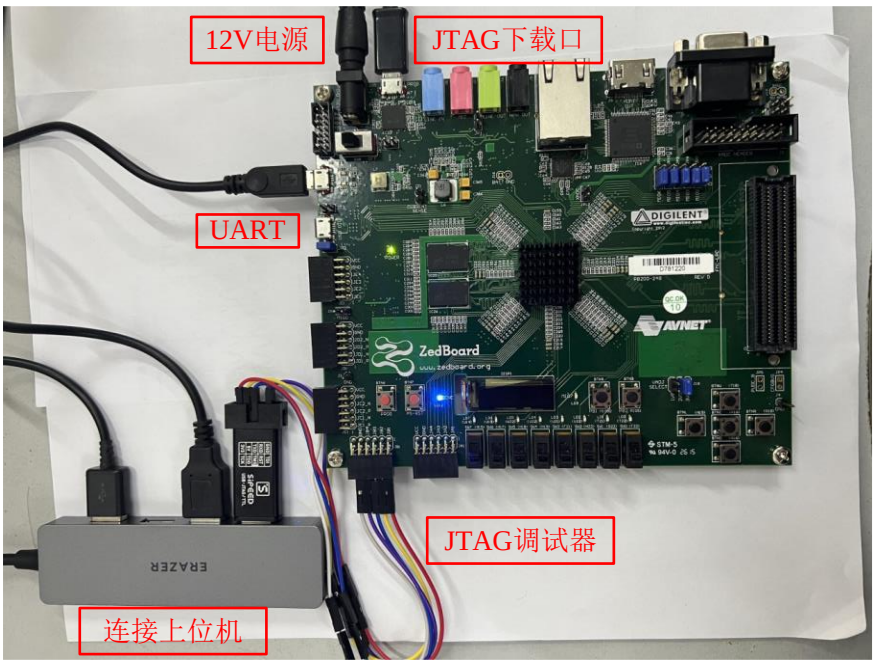

图 5-10 FPGA 原型验证平台实物图

5.3.2 运行软件测试用例

在完成 FPGA 原型验证平台的搭建后，编写软件测试程序，并在 RISC-V 架

构 SoC 中运行，以此来验证 SoC 的具体功能。本文使用 Vivado 软件将 RISC-V SoC 的 RTL 代码进行编译。由于 ZedBoard 板上的时钟晶振为 100MHz，而本设计工作的主频为 50MHz，因此额外使用了 Xilinx 提供的 PLL ip 产生本设计所需的工作时钟。完成整个项目工程的编译后，根据上节中定义的硬件连接关系编写 xdc 约束文件，为 SoC 的各输入输出接口分配对应的管脚，之后进行综合优化以及布局布线，并生成 bitstream 文件，通过 JTAG 下载口烧录至 ZedBoard FPGA 板中。

接下来首先进行 SoC 系统功能的测试，本节以 UART 测试为例进行说明。本设计使用 C 语言创建测试程序，控制 UART 模块连续发送字符串至上位机。之后使用 RISC-V 架构的 GNU 工具链进行编译，并通过调试器下载至 FPGA 板。在完成 SoC 的烧录以及测试程序的下载后，首先按下 FPGA 开发板上的 RESET 按钮对整个 SoC 进行复位，之后开始执行测试程序，完成后在 PC 上位机中通过串口接收终端查看信息。如图 5-11 所示，RISC-V SoC 的 UART 模块成功发送了“Hello! From My Low_power_SoC!!!”信息，并通过调试器上的 UART 接口传回至上位机。

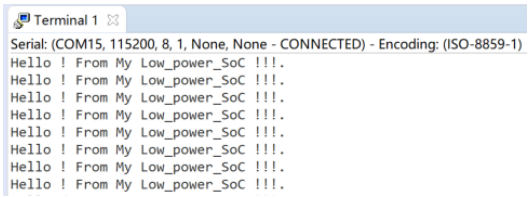

图 5-11 UART 程序成功测试结果图

在完成 RISC-V SoC 的功能测试后，进一步进行处理核在物联网应用中的性能测试。衡量处理器性能的重要指标之一就是运行跑分程序（Benchmarks）后的得分情况，分数越高代表处理器性能也相对更优。本设计中采用的是由 C 语言编写的开源跑分程序 CoreMark，CoreMark 测试是一种广泛应用于嵌入式系统和处理器性能评估的测试基准，它涵盖了一系列在物联网应用中常用的操作，例如列表处理、矩阵操作和循环冗余校验等，具有较高的实际价值[49]。因此，通过使用 CoreMark 测试，我们可以对该处理器在物联网应用中的性能进行全面的评估。

CoreMark 程序主要通过计算处理器每秒执行测试程序的次数来衡量处理器性能，该程序的运行流程如图 5-12 所示。测试程序首先会根据系统的配置参数来估算出系统在 1 秒钟内能够完成的迭代次数，这个估算值会被用来控制后续迭代的数量。之后进行实际的迭代计算，每一次迭代都会执行预定的测试程序。在完成迭代之后会进行获取本次迭代所耗时间，之后根据迭代次数与处理

器的运行频率计算出最终的跑分结果。

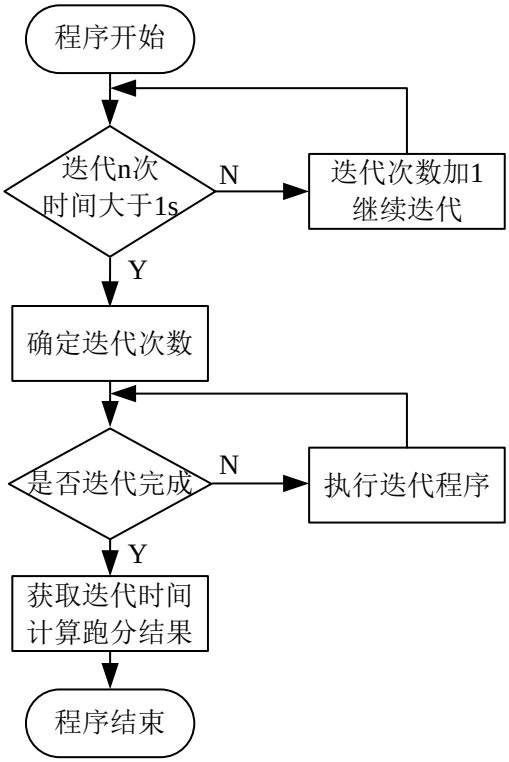

图 5-12 CoreMark 程序运行流程图

在 FPGA 原型验证平台中，成功地编译并下载了 CoreMark 程序至 ZedBoard 板上的 RISC-V SoC 平台中，并通过串口在监视器上打印的测试程序的运行结果信息。如图 5-13 所示，本文开发的 RISC-V 处理器核完成了 1000 次 CoreMark 程序的迭代执行，在 50MHz 运行主频下每秒可执行约 117 次，最终得分为 2.35 CoreMark/MHz。本文设计的 RISC-V 处理器核在性能上与在物联网领域中广泛应用的 ARM Cortex-M0 处理器基本持平，能够满足物联网应用对处理器核的性能需求。

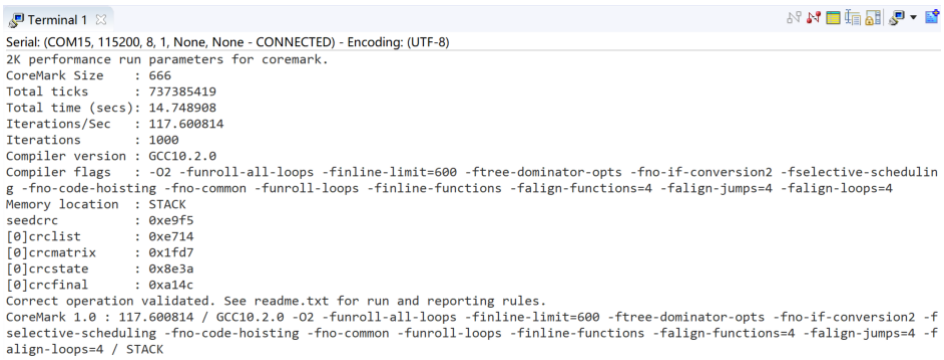

图 5-13 CoreMark 跑分程序打印得分信息图

5.4 RISC-V SoC 的物理实现与功耗评估

在完成 RISC-V 架构 SoC 的功能验证后，对整个 SoC 的后端物理设计。首先经过逻辑综合，得到门级网表和时序约束文件，之后进行布局规划、时钟树综合和物理布线等步骤，最终得到芯片版图。此外，本设计在逻辑综合和布局布线阶段都加入了功耗意图描述文件，保证本系统中的低功耗设计能够完整且正确地实现。

本设计基于东部高科的 180nm 工艺库，使用 Design Compiler、Innovus 和 PrimeTime 等 EDA 工具，完成了 RISC-V 架构 SoC 的物理实现与功耗评估。

5.4.1 功耗意图设计

在进行实际的物理实现之前，首先进行 RISC-V 架构 SoC 的功耗意图设计。本设计采用统一功耗格式（Unified Power Format，UPF）来实现 SoC 的多电压域设计。UPF 由多家芯片设计公司共同推广和支持，旨在提高芯片设计中功耗意图描述和电源管理方面的标准化程度，降低设计成本和设计复杂度，提高设计效率和设计质量。UPF 支持多种设计工具和芯片设计流程，如逻辑综合、时序约束、布局布线、功耗分析和电源管理等，可以实现对芯片设计的全面管理和优化。本设计使用的 UPF 文件如图 5-14 所示。

```
create_power_domain PD_TOP -include_scope
create_power_domain PD_MAIN -elements u_lpcpu_subsys_top/u_lpcpu_subsys_main
create_power_domain PD_AON -elements {u_lpcpu_subsys_top/u_sirv_aon_top
u_lpcpu_subsys_top/u_sirv_debug_module}

create_supply_port VDD_HIGH
create_supply_port VDD_MAIN
create_supply_port VSS
……
```

图 5-14 upf 约束文件电源域部分节选图

首先根据 SoC 的电源架构对整体设计进行电源域的划分，共包括了三个电源域 PD_TOP、PD_MAIN 和 PD_AON。PD_TOP 为顶层电源域，用于连接外部电源与内部电流模块。PD_MAIN 为电源主域，放置 RISC-V 处理器核、片上总线及主要外设，为可变电压域。PD_AON 为常开恒压域，包括 DVFS 控制器模块等常开模块。针对这三个电源域，本设计定义了三个电源接口，分别为 VDD_HIGH、VDD_MAIN 和 VSS。其中 VDD_HIGH 向 PD_MAIN 和 PD_AON 电压域提供 1.8V 电压，VDD_MAIN 向 PD_MAIN 电压域提供 1.6V 电压，VSS

则恒定接地。

此外还需对电路的功耗状态进行定义，不同功耗状态下可以使能不同的电源接口，考虑到本设计所支持的电压值，共设计了三种功耗状态，具体如表 5-1 所示。MAIN_HIGH 与 MAIN_LOW 为芯片正常工作的两种工作状态，分别使用 LV_HIGH 和 LV_LOW 两种电压值为 PD_MAIN 电压域供电，可根据系统负载情况由 DVFS 控制器动态切换。SLEEP 状态为 SoC 待机时进入的休眠状态，此状态下打开常开域所使用的 VDD_HIGH 电压，暂时关闭主域使用的 VDD_MAIN 电压以节省系统功耗。

表 5-1 SoC 功耗状态表

功耗状态	VDD_HIGH	VDD_MAIN	VSS
MAIN_HIGH	VDD_HV	LV_HIGH	VSS_ON
MAIN_LOW	VDD_HV	LV_LOW	VSS_ON
SLEEP	VDD_HV	OFF	VSS_ON

最终完成的 SoC 电压域设计如图 5-15 所示，VDD_HIGH、VDD_MAIN 和 VSS 三个电源接口从顶层电源域 PD_TOP 连接至内部电源域 PD_AON 和 PD_MAIN，向两个电源域供电。

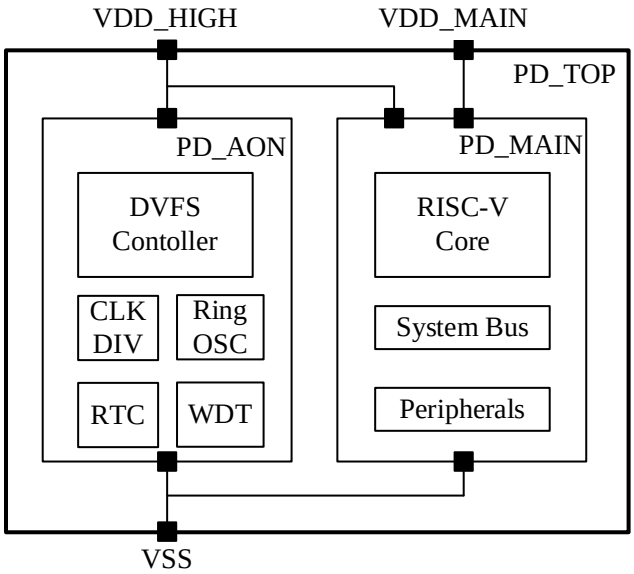

图 5-15 RISC-V SoC 电源域设计示意图

5.4.2 逻辑综合与物理版图设计

RISC-V SoC 的 Verilog 代码需要进行逻辑综合才能将设计映射到所选的工

艺库中，并生成与目标芯片实现相适应的电路，本文使用的综合工具是 Synopsys 公司的 Design Compiler，综合的主要流程如图 5-16 所示。

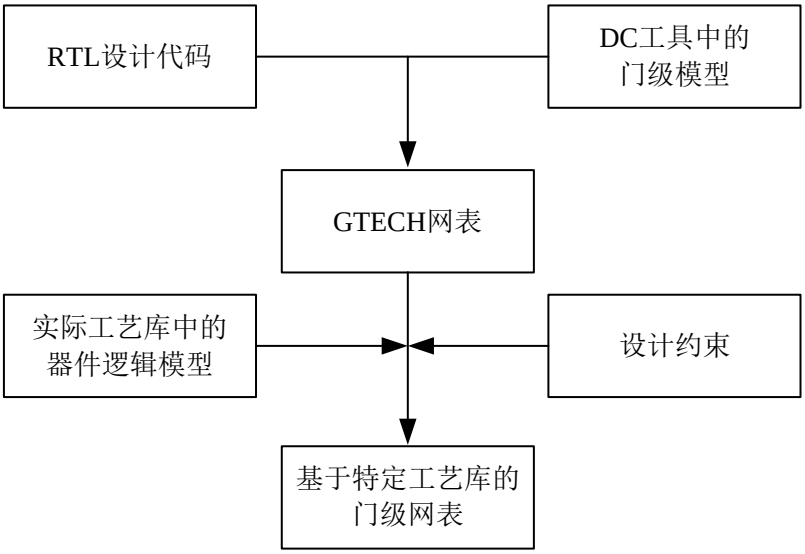

图 5-16 DC 综合流程示意图

首先将 RISC-V SoC 设计的 RTL 代码替换为 DC 工具中的门级模型，生成与具体工艺无关的通用网表（Generic Technology，GTECH）；之后对 GTECH 网表进行典型电路和一些逻辑运算进行优化，节省硬件资源；接着进行 GTECH 网表的门级映射，将网表中的各个逻辑门换成相应代工厂内的标准门；最后进行时序优化，在 DC 工具中添加对于时序、端口、面积等方面的约束，最终得到满足各项设计约束的电路网表。

约束的合理设定是 DC 综合中的重要环节，直接关系到最终芯片电路的性能。本设计将建立时间不确定度设置为 0.2ns，保持时间不确定度设置为 0.1ns 输入延迟和输出延迟都设置为 1.2ns；之后将输入驱动设置为工艺库中二输入与非门 ISO_ND2D4 的输出端，输出驱动设置为工艺库中四输入与非门 ISO_ND4D8 的输入端，并将最大扇出系数约束为 20；最后基于不同工作状态下的工艺库设定工作环境，DC 工具使用 SS 工艺角下的时序库分析建立时间，使用 FF 工艺角下的时序库分析保持时间，保证各工艺角的时序均能满足。此外，由于本设计中进行了多电压域设计，还需要读入 upf 功耗意图描述文件。文件的具体设计内容在上节中已经进行了说明，这里不再重复。在 DC 综合的流程中，只需使用 load_upf 操作将 upf 文件读入至 DC 工具，同时设定好本设计中几种电源接口的电压值，即可完成功耗意图文件的读入。

完成工艺库的设定、RTL 设计读入和约束设定等步骤后，对 RISC-V SoC 进行逻辑综合，生成时序检查报告和各项违例报告，并输出电路网表文件和时序

约束文件。在各时序路径都已满足要求后，最终得到的面积和时序参数如表 5-2 所示。

表 5-2 综合报告中的主要参数

综合报告结果	数据
Clock_frequency	50MHz
Number of ports	64800/个
Number of nets	116118/个
Number of cells	56730/个
Number of combinational cells	44081/个
Number of sequential cells	11597/个
Number of buf/inv	6136/个
Number of references	3/个
Combinational area	1179861.810139/μm^2
Buf/Inv area	94799.432339/μm^2
Noncombinational area	1179935.113739/μm^2
Total cell area	2359796.923878/μm^2
Total area	2359796.923878/μm^2

综上所述，RISC-V SoC 的综合工作全部完成，最终的硬件实现面积为 2359796.923878μm^2。需要特别指出的是，本设计在综合阶段并未使用 DC 工具对系统功耗进行分析，此部分研究工作将在下节中使用其他更为专业准确的功耗评估工具进行。接下来进行 RISC-V SoC 的物理版图设计。

本设计使用 Innovus 软件进行整体电路的布局布线。首先读入 DBH 工艺库的标准单元库和工艺库文件，以及 DC 综合输出的电路网表文件，完成电路基本结构的初始化。之后还需进行时序分析设置，Innovus 软件中提供了多模式多工艺角（Multi-Mode Multi-Corner，MMMC）分析方法。本设计在初始化阶段导入了用于时序分析的 MMMC 文件，此 MMMC 文件包含了由代工厂提供的时序库文件、寄生参数查找表文件和由 DC 综合输出的时序约束文件，并定义不同的视图来对建立时间和保持时间进行分析。此外，本设计还添加了 I/O PAD 的排列文件和功耗意图描述文件，完成了设计的初始化。

读入设计后对整个芯片进行布局规划与电源网络的设计。布局规划的质量对芯片电路的整体时序和功耗等都具有重大影响，而电源网络的设计能够为芯片提供稳定的电源供电，从而有效地提高芯片的可靠性和稳定性。本设计中使

用的工艺共有六层金属层，由于高层金属的寄生电阻相对较小，用作电源线可以减少电源路径上的电阻和电容，从而减小功耗噪声的传播和影响，因此本设计采用高层金属 MET5 与 MET6 来构成电源网络。本设计中共有三种电源信号，即为芯片主域和常开域供电的 VDD_HIGH 和 VDD_MAIN，以及地线 VSS。三种电源线与芯片电源域的连接关系在设计初始化时已由功耗意图描述文件进行了定义。在定义全局电源网络名称及其连接关系后，开始布置电源环与电源带。根据芯片综合报告给出的面积信息以及实际布局规划，在芯片内核和 I/O PAD 之间预留 200μm 布置电源环。然后设置连接电源环的电源带来为芯片内核部分供电，垂直方向使用金属 MET5，水平方向上使用金属 MET6，完成电源网络的设计的芯片版图如图 5-17 所示。

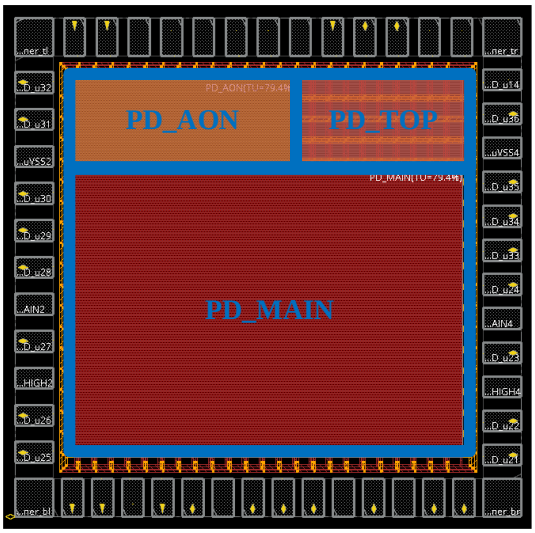

图 5-17 完成电源网络设计的芯片版图

完成电源网络的设置后，进行标准单元的物理放置。在进行放置操作后进行 preCTS 优化，即进行时钟树综合之前的优化，通过优化时序约束，如缩短时序路径、调整时序偏差等，可以进一步优化芯片的布局，以满足时序要求。在时序优化完成后，开始时钟树综合，确定时钟线的路径、方向和长度，以满足时钟树的时延和功耗约束，最终生成一个低时延、低抖动、低功耗的时钟树。若此阶段的时序检查的结果不理想，则继续进行时钟树综合之后的优化，即 postCTS 优化，进一步改善时序。

下一步进行标准单元的布线工作，将经过物理布局的电路元件之间的连线进行细化和最优化。在布线的过程中，自动布线工具会对时序要求和信号串扰等约束进行检查和优化，以保证芯片的性能和可靠性。最后在完成布线后，进行芯片版图的填充单元插入。在芯片内部电路以及 I/O PAD 中插入一些没有功

能的单元，以填补布局中留下的空白区域，从而使布局更加紧凑。最终得到的芯片版图如图 5-18 所示。

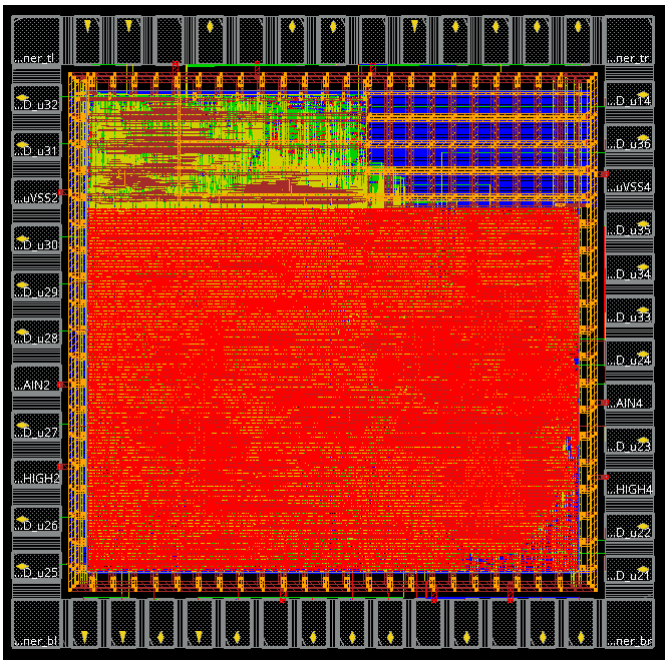

图 5-18 RISC-V SoC 芯片物理版图

整体电路版图面积为 3321787.320μm^2，其中位于版图内偏上方的区域为 SoC 常开域，主要包括 DVFS 控制器电路、环路振荡器及定时器等模块；而位于下方面积较大的区域为 SoC 主域，包括 RISC-V 处理器核及主要外设。完成芯片的版图绘制后，输出并保存电路网表，用于后续的系统功耗分析。

5.4.3 功耗分析与评估

在完成 RISC-V SoC 的物理版图实现后，进行功耗统计分析。本设计使用 PrimeTime PX 工具对 SoC 产生的功耗进行分析，并对有无 DVFS 设计的系统能耗变化进行比对分析。

PrimeTime PX 工具有两种功耗分析模式：平均功耗分析模式（Averaged Power Analysis）和基于时间的功耗分析模式（Time-Based Power Analysis）。平均功耗分析模式一般用于计算芯片或模块的平均功耗，主要通过确定各电路元件的功耗贡献，并针对贡献较大的元件进行优化，以达到功耗优化的目的。但此模式下计算的功耗信息相对粗略，不适合作为衡量芯片实际工作功耗的指标。基于时间的功耗分析模式与平均功耗分析模式不同，它通过对每个时钟周期内电路的瞬态功率进行计算，得出更加精细的芯片功耗情况，可以提供较为接近芯片实际工作消耗的功耗数据结果，在低功耗芯片设计中广泛应用。

综合上述分析，本设计采用基于时间的功耗分析模式来对 RISC-V SoC 进行功耗分析。在此模式下，PrimeTime PX 工具需要读入电路网表的仿真文件，才能对系统工作期间的各个时钟周期的瞬时功耗进行分析。因此，本设计首先在 VCS 工具中启动带有功耗意图信息 UPF 文件的仿真，需要在开始 VCS 仿真时加入 UPF 编译选项，之后在 TestBench 顶层模块中导入 UPF 库并设置各电源接口的电压值，为各电源域供电，即可进行带有电源信息的仿真。仿真时输出的日志文件中记录了各电源域及电源接口的状态信息，为了更加直观地体现电源状态的变化情况，在 Verdi 工具中查看仿真波形，如图 5-19 所示。

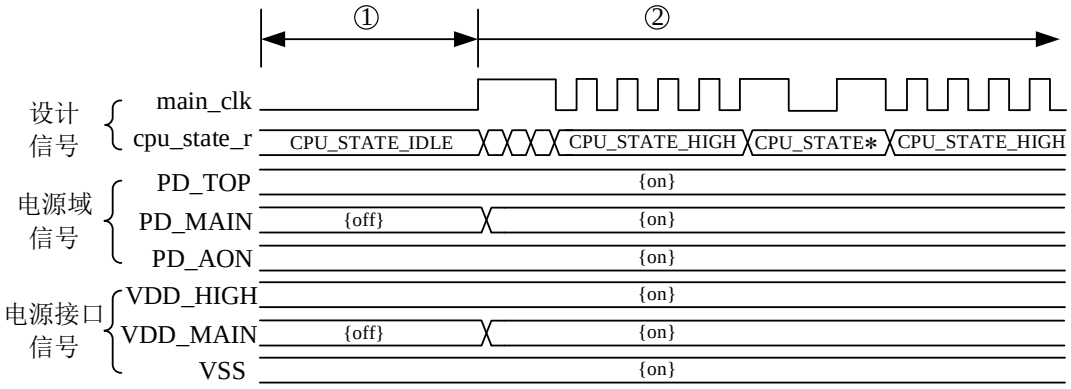

图 5-19 SoC 电源信息仿真波形图

此波形中包含来自待测设计中的主域时钟信号和 CPU 工作状态指示信号，还增加了描述电源信息的电源域信号和电源接口信号。波形图 5-19 展示了 SoC 系统从初始上电状态转换正常工作状态的变化过程，状态①为系统的初始上电状态，此时顶层电源域 PD_TOP 和常开电源域 PD_AON 为打开状态，其对应的 VDD_HIGH 和 VSS 电源接口信号也为打开状态；主域 PD_MAIN 为关闭状态，为其供电的 VDD_MAIN 接口也为关闭。当系统转化为状态②时，CPU 根据负载情况受 DVFS 控制器控制，在不同工作状态之间切换。由于此阶段内系统负载始终没有完全清空，意味着处理器始终处于工作状态，所以此时系统主域及其他电源域均处于打开状态，各电源接口也正常供电。由此可见，本设计中的 SoC 可以正确地按照所设计的功耗意图来控制电源域和电源接口的信号变化，基本达到了预期效果。

在完成 SoC 电源域的仿真后，生成带有电源信息的仿真波形 VCD 文件，该文件描述了每个电路节点的翻转率情况，将其应用在 PrimeTime PX 工具中作为开关行为表示文件进行功耗分析。在依次完成工艺库设置、电路网表读入、功耗分析模式设置、功耗意图文件读入以及开关行为文件读入等步骤后，得到最终的功耗数据分析报告，在 DBH 180nm 工艺库、TT 工艺角下运行测试程序

后得到的功耗数据报告如表 5-3 所示，系统整体总功耗约 37.4mW。

表 5-3 功耗数据分析报告

实例名	内部功耗 (W)	翻转功耗 (W)	漏电功耗 (W)	总功耗 (W)
lpcpu_soc_top (Total)	3.42e-02	3.14e-03	5.98e-06	3.74e-02
u_lpcpu_aon_top	1.71e-03	2.13e-03	6.13e-07	3.84e-03
u_lpcpu_aon_clkdiv	1.55e-04	2.78e-05	8.81e-09	1.83e-04
u_lpcpu_aon_wrapper	2.05e-06	1.72e-07	4.73e-07	2.70e-06
u_lpcpu_aon	2.01e-06	1.56e-07	4.65e-07	2.63e-06
u_dvfs_controller	1.48e-05	4.27e07	1.45e-09	1.52e-05
u_lpcpu_subsys_main	3.07e-02	8.57e-04	5.07e-06	3.15e-02
u_lpcpu_subsys_clint	6.45e-04	4.09e-08	8.55e-08	6.46e-04
u_lpcpu_subsys_plic	1.85e-03	8.04e-09	2.60e-07	1.85e-03
u_lpcpu_cpu_top	9.87e-03	8.56e-04	2.48e-06	1.07e-02
u_lpcpu_cpu_ifu	8.51e-04	8.03e-05	2.14e-07	9.31e-04
u_lpcpu_cpu_dec	4.51e-03	3.86e-04	6.61e-07	4.89e-03
u_lpcpu_cpu_exu	3.41e-03	2.94e-04	6.55e-07	3.71e-03
u_lpcpu_cpu_biu	5.91e-04	4.88e-05	8.43e-08	6.40e-04
u_lpcpu_cpu_wbck	2.46e-07	5.16e-07	4.08e-09	7.66e-07
u_lpcpu_cpu_lsu	3.60e-04	3.60e-05	7.34e-08	3.91e-04
u_lpcpu_clk_ctrl	7.17e-05	1.11e-05	5.90e-09	8.28e-05
u_lpcpu_dbg_module	1.85e-03	1.49e-04	2.94e-07	2.00e-03
u_lpcpu_subsys_perips	1.54e-02	7.08e-07	1.92e-06	1.54e-02
u_lpcpu_subsys_mems	2.85e-03	6.53e-10	3.14e-07	2.85e-03
u_lpcpu_subsys_srams	2.89e-05	4.08e-06	6.37e-07	3.36e-05

为了进行功耗收益的分析评估，本文统计了 RISC-V SoC 工作在不同工艺角下产生的系统功耗，同时以主域电压与工作频率保持恒定状态下的系统作为参照，比较得出本文 SoC 低功耗设计的功耗收益。经过仿真验证，本设计的功耗收益分析图如图 5-20 所示。其中未使能 DVFS 调节的 SoC 作为参照组，工作在 TT 工艺角、1.8V 电压、25℃的典型工作状态。与参照组相比，进行 DVFS 调节的电路系统在 FF、TT 和 SS 工艺角下分别节省了 25.9%、31.2%和 37.9%的系统功耗。

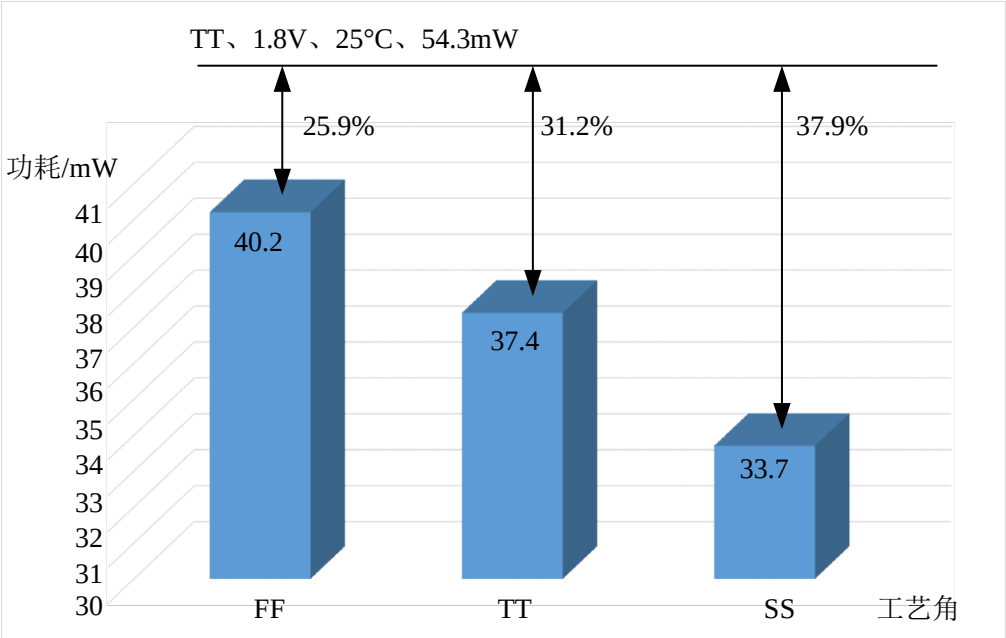

图 5-20 不同工艺角下的功耗收益结果图

本文将本设计产生的功耗收益与业内其他研究机构的同类研究进行了对比，比较结果如表 5-4 所示。

表 5-4 功耗收益对比

研究机构	验证系统	工艺	最大功耗收益	性能
Samsung Electronics (ICCE2022)[50]	微处理器	5nm	15.98%	2.9GHz
AMD(VLSI2016)[51]	微处理器	28nm	20%	1.6GHz
MediaTek (ISSCC2021)[52]	多媒体 SoC	7nm	13%	2.8GHz
Intel(ISSCC2017)[53]	图像处理器	22nm	33%	800MHz
本文	SoC	180nm	37.9%	50MHz

其中，Samsung Electronics 通过精确计算系统工作时能量消耗最小的频率，获得了最大 15.98%的功耗收益；AMD 使用关键路径累加器技术来补偿局部电压调节时的电压偏差，能够在最高 1.6GHz 的工作频率下节省 20%的功耗；MediaTek 和 Intel 的研究人员都在减少由电压调节造成的功耗损失方面作出了贡献，分别达到了 13%和 33%的最大功耗收益。本文主要采取基于负载检测的 DVFS 调节方式，在 DBH 180nm 的工艺尺寸下获得了最大为 37.9%的功耗收益。

5.5 本章小结

本章主要开展了针对RISC-V架构SoC的仿真验证与物理实现的相关研究。首先搭建了RISC-V SoC验证平台，对RISC-V指令集进行了测试，通过了riscv-tests的全部测试用例。之后开展了RISC-V SoC在FPGA上的原型验证工作，搭建了FPGA原型验证平台，并完成了物联网应用性能测试，RISC-V处理器在50MHz运行主频下运行CoreMark跑分程序测试得分为2.35 CoreMark/MHz。接着，对RISC-V SoC进行了物理实现和功耗评估，包括功耗意图设计、逻辑综合和物理版图设计，完成版图绘制后对SoC整体进行了功耗分析和评估。最终本文设计的RISC-V架构SoC在DBH 180nm的工艺尺寸下获得了最大为37.9%的功耗收益，能够满足物联网应用对性能和功耗的需求。

结　论

随着物联网应用的不断普及和发展，传统的电子芯片设计已经难以满足其多样化、复杂化、高度集成化的需求。因此，采用 SoC 技术设计物联网电子芯片已经成为发展趋势。SoC 可以将多个功能模块集成到一个芯片中，从而实现高度集成化的设计。这种技术可以帮助降低功耗、提高性能和可靠性，并且更加灵活地满足各种应用需求。在物联网应用领域，物联网设备往往需要长时间连续运行，对电池续航能力和能源消耗有着极高的要求。因此，设计低功耗的 SoC 成为了一个关键的挑战。本文基于 DBH 180nm 工艺，应用 DVFS 技术，开展了有关面向物联网应用的 RISC-V SoC 的研究，完成了从设计到验证再到物理实现的全部过程，具体研究成果如下：

(1) 在充分分析处理器设计方法以及 RISC-V 指令集特点的基础上，结合物联网领域的应用需求，设计了 RISC-V 架构处理器的三级流水线系统结构，并在 gem5 体系结构模拟器中采用模型重构的方式构建了 RISC-V 处理器系统模型，采用 gem5 中的全系统仿真模式对处理器模型进行了 DVFS 系统仿真，对比了不同 DVFS 调频策略对系统功耗的影响，确定了本系统采用 DVFS 技术的实现方案。

(2) 采用 Verilog 语言完成了 RISC-V 处理器的三级流水线的 RTL 设计。实现了 RV32IMAC 的指令集架构，根据流水线功能需求和 RISC-V 指令特点，设计了包含指令获取、指令微解码、分支调整指令预测等模块的取指单元，设计了包含指令译码、通用寄存器组、指令派遣等模块的指令译码单元，设计了包含 ALU 模块、指令交付、指令访存、指令写回等模块的指令执行单元。同时通过复用数据计算通路、并行化指令处理等方式，完成了 RISC-V 架构微处理器的低功耗设计。

(3) 在完成处理器内核的设计研究工作后，进一步设计开发了 RISC-V 架构 SoC。首先确定了 RISC-V 架构 SoC 的整体结构，包括 RISC-V 处理器核、系统总线、外设接口等组成部分，结合中断异常处理模块和调试模块，共同实现了 SoC 的各种功能。接下来重点开展了 RISC-V SoC 低功耗设计的研究工作。完成了 SoC 电源域与时钟域的划分，使得整个 SoC 各内部模块都能够工作在适宜状态，并最大限度地节省了功耗。设计了 DVFS 控制器，该控制器以 RISC-V 处理器核中的 CSR 寄存器为接口，实时监测系统负载变化情况，并通过控制时钟分频产生模块与片外 DC-DC 电源模块为 SoC 提供对应的时钟频率与工作电压，

实现了对整个 SoC 的动态电压频率调节。

(4) 完成了 RISC-V 架构 SoC 的仿真验证与物理实现。搭建了 RISC-V SoC 验证平台，进行了 RISC-V 指令集测试，通过了 riscv-tests 的全部测试用例。开展了 RISC-V SoC 在 FPGA 上的原型验证工作，搭建了 FPGA 原型验证平台并成功运行软件测试用例，RISC-V 处理器在 50MHz 运行主频下运行 CoreMark 跑分程序测试得分为 2.35 CoreMark/MHz，在物联网应用测试中表现了较好的性能。完成了 RISC-V 架构 SoC 的物理实现和功耗评估。采用统一功耗格式实现了 SoC 的多电压域设计，并进一步完成了功耗意图设计。在 DC 工具中完成了 RISC-V SoC 的逻辑综合，在 Innovus 中完成了从布局规划至导出电路网表的完整版图绘制流程，最终得到的版图总面积为 3321787.320μm^2。采用 Primetime PX 工具对 SoC 整体进行了功耗分析和评估，最终本文设计的 RISC-V 架构 SoC 在 DBH 180nm 的工艺尺寸下获得了最大为 37.9%的功耗收益。因此，本文设计开发的 RISC-V SoC 在性能和功耗上都能够满足物联网应用的需求，基本实现了预期目标。

工作展望：

本文设计完成的 RISC-V SoC 虽然已经达到课题的预期结果，但在很多方面还有待改进和提高，需要开展进一步的优化设计研究，具体如下：

(1) 在 RISC-V 处理器核的验证工作方面，本文主要通过运行 riscv-tests 测试用例的方式进行验证，未采用 UVM 验证方法学等更为专业的方式进行可靠性验证，还需进行补充优化；

(2) 在 RISC-V SoC 的开发方面，本文设计开发的 SoC 搭载的外设模块种类相对单一，在实际应用中可以进一步添加所需功能的外设模块；

(3) 在功耗评估方面，本文并未对实际芯片进行功耗测试与评估，需要在完成流片后进行进一步的测试分析，以验证本设计的功耗优化效果。

参考文献

[1] MUKHOPADHYAY S C. Wearable sensors for human activity monitoring:a review[J]. IEEE Sensors Journal, 2015, 15(3): 1321-1330.

[2] WANG P H P, JIANG H, GAO L, et al.A near-zero-power wake-up receiver achieving -69dBm sensitivity[J]. IEEE Journal of Solid-State Circuits, 2018, 53(6): 1640-1652.

[3] 黄凯杰. 一种精简高效的 FCU 控制子系统平台[D]. 杭州: 浙江大学硕士学位论文, 2014: 34-35.

[4] A WATERMAN, Y LEE, D A PATTERSON, et al. The RISC-V in-struction set manual, Volume I: User-Level ISA[J]. Eecs Department, 2011, 7(9): 475.

[5] ASANOVIC K, AVIZIENIS R, BACHRACH J, et al. The rocket chip generator[J]. EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2016-17, 2016, 4: 3-10.

[6] CELIO, CHRISTOPHER, PATTERSON, et al. The Berkeley Out-of-Order Machine(BOOM): An industry-competitive, syn-thesizable, parameterized RISC-V Processor[R]//EECS De-partment, University of California, Berkeley, 2015.

[7] TRABER A, ZARUBA F, STUCKI S, et al. PULPino: A small single-core RISC-V SoC[C]//3rd RISCV Workshop. 2016: 1-3.

[8] GAROFALO ANGELO, RUSCI MANUELE, CONTI FRANCESCO, et al. PULP-NN: accelerating quantized neural networks on parallel ultra-low-power RISC-V processors[J]. Philosophical transactions. Series A, Mathematical, physical, and engineering sci-ences, 2020, 378(2164): 11-16.

[9] OTTAVI G, GAROFALO A, TAGLIAVINI G, et al. A mixed-precision RISC-V processor for extreme-edge DNN inference[C]//2020 IEEE Computer Society Annual Symposium on VLSI (ISVLSI). IEEE, 2020: 512-517.

[10] CHENG Y, HUANG L, CUI Y, et al. Efficient multiple-ISA embedded processor

core design based on RISC-V[C]//Workshop on Computer Architecture Research with RISC-V (CARRV). 2020: 1-3.

[11] PINYOTRAKOOL K, SUPMONCHAI B. Design of a low power processor for embedded system applications[C]//2020 8th International Electrical Engineering Congress (IEECON). IEEE, 2020: 1-4.

[12] 邓亚威. 引领芯片定制化革命——SiFive 2018 上海技术研讨会圆满召开[J]. 中国集成电路, 2018, 27(06): 12-13+21.

[13] POPOVICI C A, STAN A. Extending a RISC-V core with a CAN-FD communication unit[C]//2022 26th International Conference on System Theory, Control and Computing (ICSTCC). IEEE, 2022: 31-36.

[14] 胡振波. 手把手教你设计 CPU--RISC-V 处理器[M]. 北京: 人民邮电出版社, 2018: 25-26.

[15] 王莹. 兆易创新推出基于 RISC-V 的通用 MCU[J]. 电子产品世界, 2019, 26(9): 43.

[16] 单祥茹. 兆易创新大胆启用双赛道策略, 率先实现 RISC-V 通用 MCU 商用落地[J]. 中国电子商情（基础电子）, 2019, 9: 16-18.

[17] 王凯帆, 徐易难, 余子濠等. 香山开源高性能 RISC-V 处理器设计与实现[J]. 计算机研究与发展, 2023, 60(03): 476-493.

[18] 王凯帆, 徐易难, 何伟等. 香山开源高性能 RISC-V 处理器敏捷设计实践[J]. 单片机与嵌入式系统应用, 2022, 22(12): 4-6+36.

[19] 平头哥发布全新 RISC-V 处理器[J]. 中国集成电路, 2021, 30(06): 21.

[20] 许子皓. 从新兴到主流 阿里平头哥引领 RISC-V 端云一体“芯”生态[N]. 中国电子报, 2022-11-04(005).

[21] IBRO M, MARINOVA G. DVFS technique on a Zynq SoC-based system for low power consumption[C]//2020 International Conference on Broadband Communications for Next Generation Networks and Multimedia Applications (CoBCom). IEEE, 2020: 1-5.

[22] LIU X, KAMINENI S, BREIHOLZ J, et al. A 194nW energy-performance-aware loT SoC employing a 5.2 nW 92.6% peak efficiency power management unit for system performance scaling, fast dvfs and energy minimization[C]//2022 IEEE International Solid-State Circuits Conference (ISSCC). IEEE, 2022, 65: 1-3.

[23] HASSAN H. Test scheduling of SoC by using Dynamic Voltage Frequency Scaling (DVFS) Technique[J]. Emerging Advances in Integrated Technology, 2022, 3(1): 1-8.

[24] SAFARI M, KHORSAND R. PL-DVFS: combining power-aware list-based scheduling algorithm with DVFS technique for real-time tasks in cloud computing[J]. The Journal of Supercomputing, 2018, 74(10): 5578-5600.

[25] SAINI S, SINGH B, PAHUJA H, et al. Design and analysis of priority encoder with low power MTCMOS technique[C]//2018 8th International Conference on Cloud Computing, Data Science & Engineering (Confluence). IEEE, 2018: 14-15.

[26] KHALEEQ M M, PENSHANWAR K, ANANIAH DURAI S, et al. Design of low power 8T SRAM array with enhanced RNM[J]. International Journal of Engineering and Advanced Technology (IJEAT) ISSN: 2249, 2019, 8958: 1-6.

[27] AGRAWAL R, GOYAL V. Analysis of MTCMOS cache memory architecture for processor[C]//Proceedings of International Conference on Communication and Artificial Intelligence. Springer, Singapore, 2021: 81-91.

[28] LEE J G, JEON H, CHOI Y, et al. Fully automated hardware-driven clock-gating achitecture with complete clock coverage for 5nm exynos Mobile SoC[C]//2022 IEEE International Solid-State Circuits Conference (ISSCC). IEEE, 2022, 65: 216-218.

[29] 陈道品, 武利会, 罗春风等. 基于多电源域和自适应调压的 SoC 低功耗研究[J]. 能源与环保, 2022, 44(01): 183-188+195.

[30] 邵胜芒. 基于多电源域和自适应电压调节技术的低功耗微控制器[D]. 杭州:

浙江大学硕士学位论文, 2020: 55-58.

[31] 李航. 面向宽电压的 DVFS 控制器设计与实现[D]. 上海: 上海交通大学硕士学位论文, 2020: 53-54.

[32] 陈力颖, 胥世俊, 吕英杰. 基于门控时钟技术的时钟树低功耗研究[J]. 南开大学学报(自然科学版), 2022, 55(03): 21-25.

[33] LEE Y, WATERMAN A, AVIZIENIS R, et al. A 45nm 1.3 GHz 16.7 double-precision GFLOPS/W RISC-V processor with vector accelerators[C]//ESSCIRC 2014-40th European Solid State Circuits Conference (ESSCIRC). IEEE, 2014: 199-202.

[34] GOOSSENS B. Building a RISC-V processor[M]//Guide to Computer Processor Architecture: A RISC-V Approach, with High-Level Synthesis. Cham: Springer International Publishing, 2023: 183-200.

[35] SERRANO R, SARMIENTO M, DURAN C, et al. A low-power low-area SoC based in RISC-V processor for IoT applications[C]//2021 18th International SoC Design Conference (ISOCC). IEEE, 2021: 375-376.

[36] KÜKNER H, KAPLAYAN G, EFE A, et al. RISC-V processor trace encoder with multiple instructions retirement support[C]//2022 IFIP/IEEE 30th International Conference on Very Large Scale Integration (VLSI-SoC). IEEE, 2022: 1-6.

[37] DEEPIKA R, SM G P, ANAND V. Microarchitecture based RISC-V instruction set architecture for low power application[J]. Journal of Pharmaceutical Negative Results, 2022: 362-371.

[38] VISHNU S N S, GANDLURU A, RAMESH S R. 32-Bit RISC processor using vedicMultiplier[C]//2022 3rd International Conference for Emerging Technology (INCET). IEEE, 2022: 1-5.

[39] ZHENG T, CAI G, HUANG Z. A soft RISC-V processor IP with high-performance and low-resource consumption for FPGA[C]//2022 IEEE International Symposium on Circuits and Systems (ISCAS). IEEE, 2022: 2538-2541.

[40] 袁攀. 基于嵌入式 RISC-V 微处理器的流水线研究与设计[D]. 长沙: 长沙理工大学硕士学位论文, 2021: 10-11.

[41] 王亚军. 基于超深亚微米工艺的嵌入式 SoC 低功耗设计与优化[D]. 无锡: 江南大学硕士学位论文, 2018: 2-3.

[42] 王晓凤. 一款双核 SoC 芯片的低功耗设计与验证[D]. 长沙: 国防科学技术大学硕士学位论文, 2015: 10-11.

[43] 谢辉. 基于物联网应用的 SoC 超低功耗芯片设计[J]. 集成电路应用, 2023, 40(01): 1-3.

[44] 孙鹏涛. 一种显示器接口控制器的低功耗设计与评估[D]. 西安: 西安电子科技大学硕士学位论文, 2018: 17-19.

[45] BASIREDDY K R, SINGH A K, AL-HASHIMI B M, et al. AdaMD: Adaptive mapping and DVFS for energy-efficient heterogeneous multicores[J]. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, 2019, 39(10): 2206-2217.

[46] HIN P Y H, LIAO X, CUI J, et al. Supporting RISC-V full system simulation in gem5[C]//Proc. Workshop Comput. Architect. Res. RISC-V. 2021: 2-4.

[47] MINH C C, CHUNG J W, KOZYRAKIS C, et al. STAMP: Stanford transactional applications for multi-processing[C]//2008 IEEE International Symposium on Workload Characterization. IEEE, 2008: 35-46.

[48] 顾梦杰. 基于 RISC-V 指令集架构的通用微控制器设计[D]. 上海: 东华大学硕士学位论文, 2022: 54-56.

[49] 毛斌杰. 用于计量的嵌入式 RISC-V 处理器设计及 MCU 实现[D]. 杭州: 浙江大学硕士学位论文, 2022: 67-68.

[50] PARK C, PARK J, LEE Y, et al. A new DVFS algorithm to minimize energy consumption on system-on-chip architecture and electrical characteristics[C]//2022 IEEE International Conference on Consumer Electronics (ICCE). IEEE, 2022: 1-4.

[51] SUNDARAM S, SAMABMURTHY S, AUSTIN M, et al. Adaptive voltage frequency scaling using critical path accumulator implemented in 28nm cpu[C]//2016 29th International Conference on VLSI Design and 2016 15th International Conference on Embedded Systems (VLSID). IEEE, 2016: 565-566.

[52] HUANG B J, FANG E J W, HSUEH S S Y, et al. 35.1 an octa-core 2.8/2ghz dual-gear sensor-assisted high-speed and power-efficient cpu in 7nm finfet 5g smartphone soc[C]//2021 IEEE International Solid-State Circuits Conference (ISSCC). IEEE, 2021, 64: 490-492.

[53] CHO M, KIM S T, et al. Postsilicon voltage guard-band reduction in a 22 nm graphics execution core using adaptive voltage scaling and dynamic power gating[J]. IEEE Journal of Solid-State Circuits, 2016, 52(1): 50-63.