基于 RISC-V 处理器的软硬件联合验证平台设计与实现

钟戴元[1] 曾庆立[1] 周佳凯[1] 薛 浪[1] 唐瑞东[1]
ZHONG Daiyuan ZENG Qingli ZHOU Jiakai XUE Lang TANG Ruidong

摘 要 针对目前 RISC-V 处理器业界不成熟的验证思想和单一验证方法的欠合理性，提出一种软硬件联合验证平台。将原有处理器级的行为验证升级成 SoC 系统级的行为验证，并以此为基础，改进原有的单一软件验证和硬件验证模式，联合形成一种结构合理、内容清晰和便于移植的验证方案。并将提出的验证方案使用高级软件编程语言 C++ 和验证描述语言 SV HDL，运用 verilator 的软件验证和 FPGA 的硬件验证，实现基于 RISC-V 处理器的软硬件联合验证平台设计。实验结果表明，所设计的平台能够有效地提高验证的一般性和全面性，使用 C 语言进行验证降低了验证的门槛，增强了验证与实际应用的关联性，模块化设计和可移植性使其能够适应不同的设计需求和应用场景，为 RISC-V 处理器的进一步研究和开发提供了强有力的支持。

关键词 RISC-V；verilator；SoC；FPGA；软硬件联合验证

doi：10.3969/j.issn.1672-9528.2024.11.005

0 引言

随着半导体技术的革新发展和实际应用需求的日益增长，对处理器的性能和效率提出了更高的标准，这要求人们应不断对处理器进行创新和升级 [1]。目前，在处理器架构领域，可以看到两种主要的架构类型：一种是闭源的架构，如 X86 和 ARM 等广为人知的成熟架构；另一种是开源的架构，如 RISC-V 等新兴的自由架构 [2-3]。

RISC-V 指令集架构因其开源的特性、开放的设计理念、无授权成本、可控性以及精简且灵活的模块化设计，成为学术界和工业界的焦点 [4]。这些特点不仅降低了进入门槛，还促进了广泛的创新和定制化开发。因此，RISC-V 吸引了包括专业学者和科技企业在内的众多研究者和开发者的关注，并持续推动着其发展和应用的扩展 [5]。

RISC-V 的应用成就现已遍布多个关键领域。例如，在物联网 [6] 领域，中国移动推出的 RISC-V 基于 LTE-Cat.1 芯片和蜂窝物联网通信芯片，彰显了 RISC-V 在能效和性能方面的优势；同时，在人工智能领域，RISC-V 的高性能处理器和向量运算功能，为机器学习算法提供了高效的计算支持。此外，在边缘计算中，RISC-V 的高效设计满足了对实时数据处理的需求。此外，移动设备和服务器市场也证明了 RISC-V 在节能、性能优化和定制化方面的显著优势 [7]。

RISC-V 架构的这些优势，使其在处理器设计领域具有巨大的潜力，不断激发新的研究思路和应用场景，为半导体行业的发展贡献了新的动力。随着 RISC-V 生态系统的不断成熟，可以预见它将在未来的计算设备中扮演越来越重要的角色 [8]。

面对 RISC-V 处理器不断更新迭代的大趋势，本文提出一种基于 RISC-V 处理器的软硬件联合验证平台设计 [9]。采用 SoC 系统级的验证方法代替传统处理器级的验证方法，提升系统验证的综合性、广泛性和真实性 [10]；采用软硬件联合验证的方法取代传统单一的软件验证或硬件验证，赋予系统验证的多元化、多层次化和综合化等特点。

1 平台设计框架介绍

本设计采用的是一种层次化、条理化的项目文件管理系统，将验证平台设计归结为三个文件子类设计，如表 1 所示。

表 1 文件子类的表述

文件子类	描述
processor	专用于存储与处理器核心相关的 SV 文件，这些文件是硬件描述和设计逻辑的基础，对硬件开发和仿真至关重要。
simulator	包含了软件仿真所需的全部文件，涵盖 xbar、uart_sim、sd_sim 等仿真组件及配置文件，它们支持在 Verilator 环境下对处理器核心进行全面测试和验证。
emulator	用于存储硬件上板验证所涉及的文件，包括 AXI 和 APB 总线、uart 等 System Verilog 文件以及用于 Vivado 项目的 Tcl 脚本。

使用 Venn 图表示各文件子类的包含关系，如图 1 所示。

1. 吉首大学通信与电子工程学院 湖南吉首 416000

整个验证平台围绕着 RISC-V 处理器设计开发，使用结构化策略确保开发过程中各阶段的资源得到有效组织和管理，提升了项目的可维护性和开发效率。

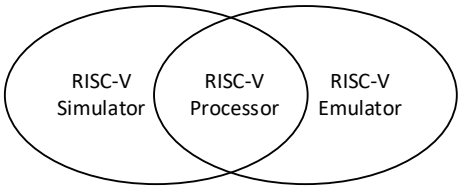


图 1 文件子类 Venn 图

2 软硬件联合验证

联合验证流程在硬件和软件的结合测试中通常分为两个主要步骤。首先是实施软件的前期逻辑验证，这一步骤主要是在仿真环境中对软件逻辑进行测试，确保软件代码在理论上的正确性；其次是进行 FPGA 平台的功能验证，这一步骤将软件与硬件设计相结合，在实际的硬件平台上测试其功能。

这两个阶段的主要差异体现为：前者主要负责验证 RISC-V 设计逻辑的正确性，后者主要负责依据软件验证的指导，验证实际硬件逻辑的正确性。这两个阶段是相互依赖且逐步深入的，它们共同构成了一个互补和连续的验证过程。

2.1 准备阶段

本文提出的基于 RISC-V 的软硬件验证平台工作在 Linux 系统环境下，使用基本工具 make、flex 和 bison，使用 riscv-gcc 的交叉编译器编译 C 语言程序，使用 verilator 和 vivado 进行软硬件平台开发。具体内容如表 2 所示。

表 2 环境及工具准备

名称	版本号	描述
Linux	20.04	系统配置成 Ubuntu20.04 或在 WSL 环境下开发
verilator	5.008	实现软件验证，目标 pc 机
vivado	2019.1	实现硬件验证，目标 fpga 板
make	—	处理 Makefile 操作，用于项目管理和自动化编译
flex 和 bison	—	用于文本处理和解析

2.2 软件验证

此处采用纯软件的方式构建 SoC 系统基础架构。整体上，使用 C++ 程序描述 SoC 系统 top 层的运行流程进行表述。细节上，对使用到的 uart、sd 卡等外设终端设备采用 SV HDL 描述成 verilator 软件支持的 uart_sim.sv 和 sd_sim.sv。之后，根据 SV HDL 描述的结构整合成完整的 SoC 系统。展示 SoC 系统 top 层的结构运行流程图，如图 2 所示。

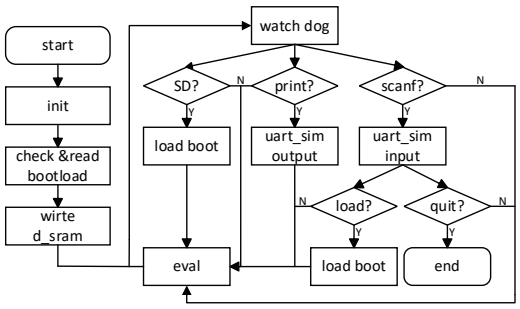


图 2 verilator top structure

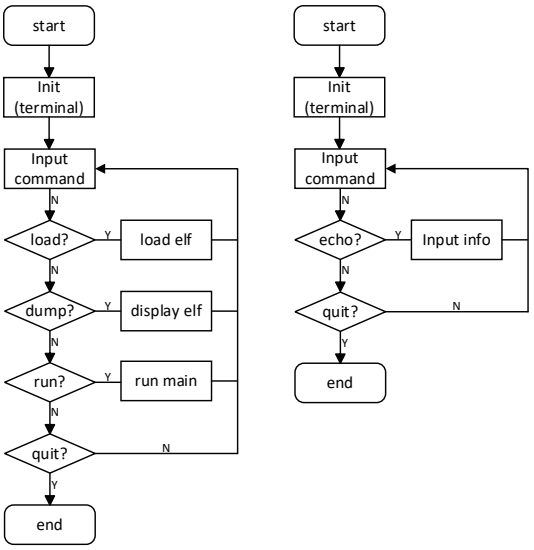
C++ 程序所描述的 top 层中主要负责 SoC 系统模型的 clock 驱动，记录 SoC 系统的 time step，并可以访问每个 time step 下的 register 内的实时信息。其次负责实现 SoC 系统的终端通信功能。本文的 SoC 系统的 C++ 程序是通过实现 XMODEM 的串口通信协议程序实现基础的人机交互。后续负责外部文件的检查、读写功能，全面还原 bootload 导入 SoC 系统的过程。

Bootload（Bootload.c）实现了 SoC 系统模型启动方式的描述。本文设计的系统上电提供了三个可使用的操作命令。命令及其内容如表 3 所示。

表 3 bootload 命令描述

Command	Format	Description
load	load	加载 main（riscv-gcc 编译 main.c）
dump	dump	检查 main 文件（hex）
run	run *	运行 main。* 指定运行 SD 卡时代入 SD 名

Bootload 通过这三个可执行操作命令，引导 SoC 系统模型到指定的程序入口（main）中执行。Bootload 程序框图，如图 3（a）所示。本文执行一个简单的 main 验证程序，实现一个基本的 echo 回传功能。Main 程序框图如图 3（b）所示。



（a）boot code （b）main code

图 3 boot code and main code

终端显示方面，整个 SoC 系统为 uart 配置了物理地址，如下所示。

#define UART_LITE1 ((volatile struct uart_lite *) 0x9a100000)

依据 uart 波特率计算公式，根据需要的 Baud 的大小调整 div 系数。

$$\text{Baud} = \frac{f_{\text{clk}}}{\text{div} \times (\text{data} + \text{stop} + \text{parity})} \quad (1)$$

本文项目的具体实验参数为：f_{clk}=100 MHz，data=8 bit，stop=1 bit，parity=0 bit。

最后，展示软件验证结果。

```
串口装载程序及其运行结果：
Running ...
[TRACER] Output filename is: trace_core.log
boot loader started.
neo-terminal-> load
XMODEM receive succeeded.
neo-terminal-> dump
size: 580
7f 45 4c 46 01 01 01 00 00 00 00 00 00 00 00 00
02 00 f3 00 01 00 00 00 00 00 08 80 34 00 00 00
40 04 00 00 01 00 00 00 34 00 20 00 02 00 28 00……
neo-terminal-> run
starting from entry point: 80080000
Hello World!
> echo Hello world!
 Hello world!
> exit

SD 卡装载程序及其运行结果：
Running ...
[TRACER] Output filename is: trace_core.log
 boot loader started.
neo-terminal-> run neo
starting from entry point: 80080000
Hello World!
> echo Hi!How do you do?
 Hi!How do you do?
> exit
```

2.3 硬件验证

FPGA 上板测试阶段，本文选用 xc7k325tffg676-2 型号开发板来进行实验。通过设计 SoC 系统框架来完成该 RISC-V Processor 的验证。该系统框架主要由 region、xbar 和外设 peripheral 三部分组成，如表 4 所示。其中，region 部分引入了 core2axi_intf 模块，为整个验证平台的移植提供了指导方案。

表 4 文件子类的表述

Model	Include
region	RISC-V core、core2axi_intf 等
xbar	axi_node、master_bus 和 slave_bus 等
peripheral	apb、uart 等

该 SoC 系统采用模块化的 RISC-V 处理器作为其核心，配备指令只读存储器（Instr_ROM）和数据只读存储器（Data_ROM）以存储代码和数据。同时，还包括一个启动只读存储器（Boot_ROM）用于初始化程序，以及通过 AXI 和 APB 总线进行系统内部通信的机制。此外，系统还集成了 UART 接口，用于与外部设备进行串行通信。宏观结构，如图 4 所示。

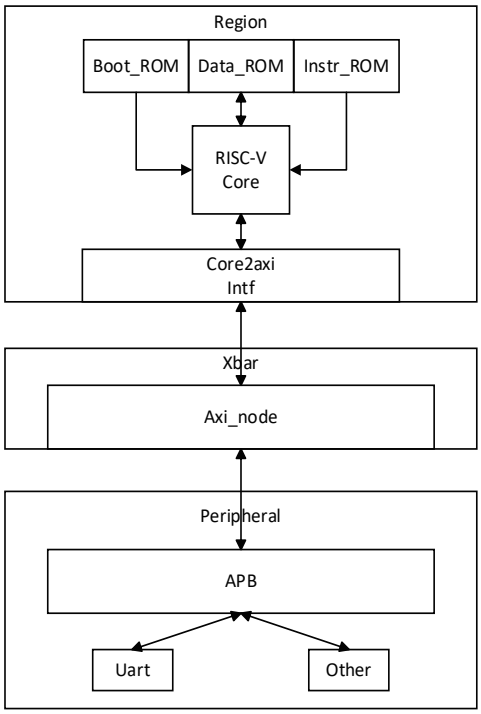


图 4 硬件验证的 SoC 系统图

将整个 SoC 系统放入 vivado 进行逻辑综合，得到各资源模块占用及功耗情况，如图 5 所示。

```
+-----------------------------------+------+
|Module                             |Cells |
+-----------------------------------+------+
|top                                | 22279|
+-----------------------------------+------+
|neoino_top                         | 22131|
|axi_node_intf_wrap                 |  1140|
|clk_rst_gen                        |     7|
|core_region                        | 10075|
|peripherals                        | 10909|
+-----------------------------------+------+
|apb_uart                           |  2463|
|axi2apb_wrap                       |  2077|
|periph_bus_wrap                    |    72|
+-----------------------------------+------+
```

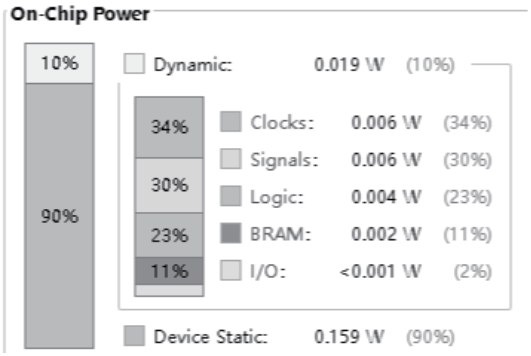


图 5 各资源模块占用及功耗情况

通过分析，可以清晰了解到该 RISC-V 构成的 SoC 系统资源占用量和功耗比例，为综合考量 RISC-V 处理器的性能提供了指南。

结果显示使用的是 CuteCom 串口调试工具，通过 PC 端捕获并打印了 SoC 系统通过串口发送的数据信息。串口调试工具现场打印的结果如图 6 所示。

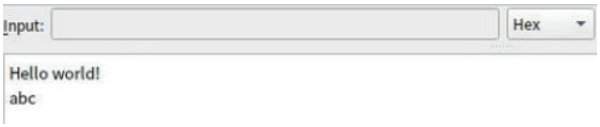


图 6 CuteCom Display

3 结语

本文围绕 RISC-V 架构处理器，构建了一个集成的软硬件联合测试框架，并详细阐述了联合验证的基本原则、实施策略，以及该框架的构建流程和在 FPGA 上的实际应用演示。实验结果表明，该联合验证框架不仅显著提高了验证过程的一般性和准确性，还引入 C 语言程序进行测试，降低了 RISC-V 验证门槛并强化了与实际工作环境的联系，增强了对 RISC-V 处理器在实际 SoC 环境中性能和可靠性的全面评估。通过软硬件联合验证的方法，能够更深入地理解处理器与系统其他组件的交互，确保了设计的一致性和协调性。此外，SoC 系统级验证方法的应用，使得能够在更接近真实应用场景的条件下测试和优化处理器，从而减少了从设计到部署的风险性和不确定性。

此外，通过软硬件联合的验证技术，可以对系统架构的准确性进行全面的模拟验证和上板验证。利用 FPGA 进行的原型验证能够在物理硬件环境中进行深入测试，这不仅克服了纯软件验证的局限性，还有助于减少芯片验证的时间和整体开发周期，显著提升芯片在系统级应用中的首次成功率。

通过本研究，期望为 RISC-V 处理器的设计提供一个全面、高效的验证解决方案，推动 RISC-V 处理器的快速发展，为电子设备的运行提供更强大的计算能力。

参考文献：

[1]VLADIMIR H, ROLF D. Advanced virtual prototyping for cyber-physical systems using RISC-V: implementation, verification and challenges[J].Science China information sciences, 2021, 65(1):110201.

[2] 陈龙震 , 徐康民 , 徐天骅 , 等 . RISC-V 架构的交叉调试系统设计 [J]. 单片机与嵌入式系统应用 ,2023,23(12):15-18+22.

[3] 周丹雅 .RISC-V 势如破竹启航算力新未来 : 专访算能高级副总裁张力 [J]. 中国安防 ,2023(12):73-76.

[4] 田林琳 . 美国第五代精简指令集技术产业发展现状 [J]. 中国科技人才 , 2023(6):75-80.

[5] 刘旸 , 丁涛杰 , 汤明宏 , 等 . 一种 RISC-V 验证系统的设计与实现 [J]. 电子技术与软件工程 ,2021(13):107-108.

[6]TOKER O. A high-level synthesis approach for a RISC-V RV32I-based system on chip and its FPGA implementations†[J]. Engineering proceedings, 2023,58(1):72.

[7] 王铜柱 , 张宇 , 罗云鹏 . 国产化 RISC-V 芯片的健康管理工程设计及实践 [J]. 单片机与嵌入式系统应用 , 2023, 23(11): 25-28+62.

[8] 杜岚 , 王裕 , 刘向峰 , 等 . 一种基于 RISC-V 架构的高性能嵌入式处理器设计 [J]. 小型微型计算机系统 , 2023, 44(12): 2865-2871.

[9] 邢明杰 , 宋威 , 张科 , 等 .RISC-V 技术及生态专题前言 [J]. 计算机系统应用 , 2023,32(11):1-2.

[10] 谭飞鸿 , 苏成悦 . 基于 Chipyard 的 RISC-V 处理器设计与实现 [J]. 现代计算机 ,2023,29(17):68-73.

【作者简介】

钟戴元（2000—），男，湖南怀化人，硕士研究生，研究方向：SoC FPGA 系统、RISC-V 处理器系统结构。

曾庆立（1975—），通信作者（email: 002095@jsu.edu.cn），男，湖南永州人，博士，高级实验师，硕士生导师，研究方向：FPGA 综合应用、RISC-V SoC 系统生态。

周佳凯（2000—），男，湖南永州人，硕士研究生，研究方向：FPGA IP 核、AXI 总线。

薛浪（1999—），女，贵州六盘水人，硕士研究生，研究方向：FPGA、数字信号处理。

唐瑞东（2000—），男，湖南娄底人，硕士研究生，研究方向：FPGA、图像信号处理。

（收稿日期：2024-07-24）