Comprehensive Verification of the RISC-V Memory Management Unit: Challenges and Solutions

Huda Sajjad, Muhammad Hammad Bashir, Yazan Hussnain[*], and Fatima Saleem

10xEngineers

Abstract

The Memory Management Unit (MMU), critical for virtual memory translation and protection, demands rigorous verification due to its inherent complexity. This work details a two-step methodology to ensure MMU compliance with RISC-V Privileged ISA specification. First, a test suite was developed using the RISCOF framework, leveraging RISC-V ISAC for coverage analysis. Second, the suite was executed on the OpenHW Core-V Wally processor, employing ImperasDV as a reference model and riscvISACOV for functional coverage development. This approach identified a critical architectural bug in Core-V Wally's MMU implementation, demonstrating the methodology's effectiveness in validating memory management units.

Introduction

The Memory Management Unit (MMU) is a vital component, responsible for virtual address translation, memory protection, and enabling efficient multitasking. Compliance with the RISC-V Privileged ISA specification[1] is essential to ensure interoperability and reliability across diverse implementations. However, the RISC-V MMU's configurability—supporting multiple paging schemes, with superpage address translations—introduces significant verification challenges, particularly for open-source cores where edge cases and specification ambiguities can lead to critical flaws.

Implementation

This work addresses these challenges through a systematic two-phased methodology as described below:

1. **RISCOF Framework**: In the first phase, a comprehensive test suite comprising 29 tests was developed using the RISCOF framework[2], leveraging the RISC-V Spike simulator and the RISC-V Sail Golden reference model for cross-verification. Functional coverage of the test suite was ensured through the RISC-V ISAC framework, employing YAML-based covergroups to track virtual memory behaviors rigorously. A total of 505 coverpoints were generated.
2. **Core-V Wally (CVW) Verification Workflow**: In the second phase, the test suite was executed on the Core-V Wally processor[3]. The riscvISACOV[4] tool facilitated functional coverage implementation, while ImperasDV acted as the reference model. A total of 182 coverpoints were developed for sv32, and 215 for both sv39 and sv48. This workflow ensured thorough verification and revealed a bug within the CVW implementation.

[*]Corresponding author: `hammad.bashir543@gmail.com`

Test Planning

When verifying any subsystem, the first and most essential step is the development of a design verification (DV) plan. This plan forms the foundation for both test creation and coverage development. Hence, a single, unified DV plan was developed to encompass every aspect of the MMU that needs to be tested and validated. This includes verifying the PTE permission bits, across all page table levels in both the supervisor and the user modes while also checking the functionality of global mappings and the satp register, which holds the root page table base address and the Address Space Identifier (ASID) used during context switching. Furthermore, it evaluates corner cases such as enabling virtualization in Machine mode, trapping otherwise permissible supervisor virtual memory management operations (by enabling TVM bit), allowing supervisor memory accesses to U-mode-accessible pages, and ensuring the functionality of MXR (Make eXecutable Readable). This plan was used as a foundation for the development of the test suite as well as coverage flow.

Test Suite development

The test suite was developed in the riscv-arch-test repository[5], to validate the functionality and compliance of virtual memory implementation with the RISC-V Privileged ISA specification[1]. For a sophisticated implementation we focused on creating test scenarios for each component of the memory management unit, particularly page table entry (PTE) configurations and fault generation. The test development process was further divided into four steps as detailed below:

1. **Test Case Design:**
   - Testing address translation for valid PTEs across all levels.
   - Saving expected behavior in the signature file.
   - Ensuring accurate fault generation for Load, Store, and Fetch accesses when the PTE lacks adequate permissions
2. **Page Table Configuration:** The page tables were dynamically configured using assembly macros:
   - *PTE_SETUP:* Sets leaf and non-leaf PTEs with custom permissions (e.g., PTE_V, PTE_R, PTE_W).
   - *SATP_SETUP:* Enable virtual memory by writing to the satp register.
3. **Modular Test Macros:** To simplify complex operations and reduce redundancy such as, configuration of Physical Memory Protection (PMP) entries to restrict or allow access and verification of RWX permissions at a given virtual address, custom macros were developed. These assisted in setting up memory regions and initializing PTEs by selecting different combinations of access permissions in different privilege modes.
4. **Signature-Based Verification:** Results for each test were written to a designated signature region in memory. After the test execution, results from both the DUT and reference are compared. Deviations indicate a failure in the virtual memory implementation.

Tracer and Coverage Flow

The Core-V Wally[3] processor, part of the OpenHW-Group, features a dedicated submodule **cvw-arch-verif**[6] designed for the architectural verification of cores conforming to the **RVA22S64** profile. This repository provided the foundation for extending the open-source Imperas **riscvISACOV**[4] platform to support VM verification. The coverage infrastructure was further enhanced to incorporate VM-specific functions, enabling more granular verification.

To achieve this, key **MMU micro-architectural signals**—such as Page Table Entries (PTEs), Physical Page Numbers (PPNs), Virtual Addresses, Physical Addresses, Access Types (RWX), and Page Levels—were exposed to the verification interface. These states, along with other architectural signals, including Control and Status Registers (CSRs), were sampled at the same pipeline stage by propagating the necessary signals. This ensured that the MMU behaved as expected under various system conditions.

Additionally, **SystemVerilog coverpoints** were implemented to leverage these functions for collecting functional coverage on VM tests. The CVW processor was used as the Design Under Test (DUT), while ImperasDV[7] served as the Reference Model for verification. The verification framework thoroughly tested all privilege modes, PMP permissions, and memory accesses, ensuring accurate permission handling and MMU functionality.

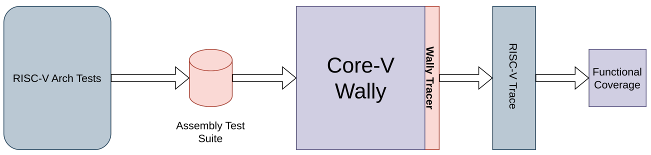

Figure 1: *Verification Flow*

Results

This work was implemented and validated on Core-V Wally[3], a 5-stage pipelined processor with configurations ranging from a minimal RV32E core to a fully featured RV64GC application processor. It validated the proposed test suite, uncovering a critical bug[8] in the memory management unit through the **reserved_pte_s_mode** test. The test revealed that Core-V Wally failed to trigger a page fault exception when accessing memory regions mapped by Page Table Entries (PTEs) with reserved encoding of RWX (i.e. pte.W=1 and pte.R=0), violating the RISC-V Privileged ISA specification[1].

References

[1] RISC-V Foundation. *RISC-V Privileged Architectures Manual, Version 1.12.* Accessed: 2025-01-27. 2021. URL: https://github.com/riscv/riscv-isa-manual.

[2] RISC-V Software Source. *RISCOF: RISC-V Architectural Compliance Framework.* Accessed: 2025-01-27. 2025. URL: https://github.com/riscv-software-src/riscof.

[3] OpenHW Group. *CVW: Core Verification Workflows.* Accessed: 2025-01-27. 2025. URL: https://github.com/openhwgroup/cvw.

[4] RISC-V Verification Group. *RISC-V ISA Coverage Analysis Tool (riscvISACOV).* Accessed: 2025-01-27. 2025. URL: https://github.com/riscv-verification/riscvISACOV.

[5] RISC-V Foundation. *RISC-V Architecture Test Framework.* Accessed: 2025-01-27. 2025. URL: https://github.com/riscv-non-isa/riscv-arch-test.

[6] OpenHW Group. *CVW Architectural Verification.* Accessed: 2025-01-27. 2025. URL: https://github.com/openhwgroup/cvw-arch-verif.

[7] Synopsys. *ImperasDV – RISC-V Processor Design Verification.* Accessed: 29-Jan-2025. 2025. URL: https://www.synopsys.com/verification/imperasdv.html.

[8] OpenHW Group. *GitHub Issue #1198: CVW Repository.* Accessed: 2025-01-27. 2025. URL: https://github.com/openhwgroup/cvw/issues/1198.