Enhancing Privileged Architecture Support in RISC-V ISAC

Muhammad Hammad Bashir[1,3], Umer Shahid*[1,3], Allen Baum[2]and Pawan Kumar Sanjaya[4]

[1]10xEngineers
[2]Esperanto Technologies
[3]Department of Electrical Engineering, U.E.T Lahore
[4]Department of Computer Science, University of Toronto

Abstract

RISCOF, a Python-based framework, ensures the compliance of RISC-V processor implementations with established instruction set simulators like Spike and Sail. This compliance testing framework offers both manual and automated test suite generation through RISC-V CTG, while coverage analysis is performed using RISC-V ISAC. However, automated test generation face challenges when handling privileged architecture tests due to the complexities of exception handling, which often require manual composition. To overcome the limitation for the coverage analysis, we introduce tailored features in RISC-V ISAC specifically designed for privileged architecture support. Additionally, a more efficient approach to writing coverpoints is proposed, aiming to reduce redundancy and simplify the process for users. These advancements aim to facilitate more comprehensive compliance testing.

Introduction

With the growing adoption of RISC-V processors across various domains, ensuring the compliance of processor implementations with the RISC-V ISA specifications becomes imperative. Compliance testing involves verifying that a processor correctly executes instructions as specified by the RISC-V ISA and behaves according to the architectural requirements. RISCOF [1], a Python based compliance framework, developed by InCoreSemi[1] and currently maintained by RISC-V International[2], serves the purpose of evaluating RISC-V target implementations against standard RISC-V golden/pseudo-golden reference models such as SAIL [2] and Spike [3] utilizing RISC-V ACTs (Architecture Compatibility Tests) [4]. RISCOF has gained global recognition owing to its open-source nature. The Figure 1, available at the RISCOF documentation captures the overall flow of RISCOF and its components.

While automated test generation via RISC-V CTG (Compliance Test Generator) [5] streamlines the testing process for most cases, it encounters challenges when dealing with privileged architecture tests. The complexities associated with exception handling in privileged architecture tests make automated generation less suitable, necessitating manual composition of these tests to ensure thorough coverage. RISC-V ISAC (RISC-V ISA Coverage) [6] is a tool designed to provide instruction level coverage checking using the CGFs (Cover Group Format). The flow adopted by

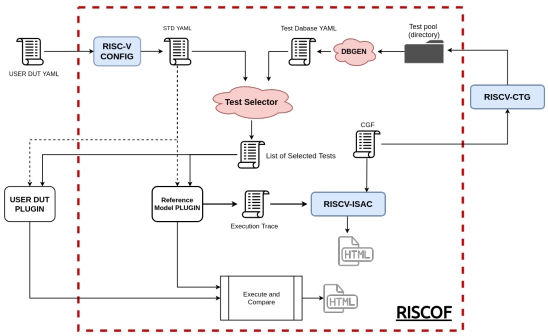

Figure 1: *RISCOF Flow*

the RISC-V ISAC[3] is shown in Figure 2. The CGF is structured as a dictionary where each node represents a collection of coverpoints termed as covergroups. These covergroups are defined based on specific ISA configurations and instructions. For unprivileged architecture, CGFs have been developed and are available at the RISC-V CTG [5] and can be used to generate ACTs. However, for the privileged architecture, there is no such support in RISC-V CTG or RISC-V ISAC. In this paper, we present features which enable enhanced coverage analysis of privileged ACTs in RISC-V ISAC. We also present a new format to write the coverpoints in more intuitive way to avoid redundancy on the user end.

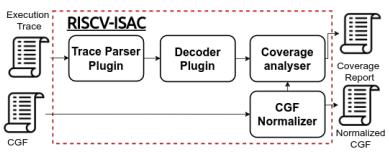

Figure 2: *RISCV-ISAC Tool flow*

*Corresponding author: umer.shahid@10xengineers.ai
[1] https://incoresemi.com/
[2] https://riscv.org/
[3] Figures have been used after permissions from the author

Table 1: *Features updated in RISC-V ISAC*

Feature	Coverpoint Format	Support
mode flag	`mode == 'M'`	new
get_addr()	`get_addr('label_name')`	new
mem_val()	`mem_val(address in hex/dec, number_of_bytes_to_fetch)`	new
mode_change	`mode_change == "S to M"`	new
rd_val	`rd_val == "value in hex/dec"`	new
mcause/mcause	`mcause == "exception_number"`	updated
mtval/stval	`mtval == "tval"`	updated

Methodologies

Features in RISC-V ISAC

To address the issue of incomplete coverage for the privilege architecture tests , we have introduced new variables and features, which are detailed in Table 1. Additionally, features to track address (virtual and physical) for both instructions and data were added. These features also track the implicit accesses during page table walks, as depicted in Table 2.

Table 2: *Features for Page Table Walk*

Feature	Coverpoint Format
pte property check	`get_pte_prop(prop_name, pa, pte_addr, pgtb_addr)`
virtual and physical address variables	`iepa, ieva, depa, deva`
ptw variables	`iptw0a ... iptw4a, iptw0cont ... iptw4cont`

Enhanced Coverpoint Format

We introduce a more concise format to define coverpoints, utilizing a variety of rules and operations, including ranges, macros, placeholders, loops, enumeration with operations, and advanced range enumeration to remove redundancy in user-defined coverpoints. This new format, similar to *abstract_comb* [1], is versatile and applicable to any label. The updated tool flow for the RISC-V ISAC is given in Figure 3.

The first coverpoint under the *mnemonics* label expands to 8 distinct coverpoints in Listing 1. The macro ${*custom_cov*} can be used across multiple coverpoints to reduce redundancy, and its value is defined in the *macros.yaml* file. The expression {{0...7} >> 2} in Line 8 is evaluated, resulting in two sequences, one $\{0, 1, 2, 3..., 7\}$ for the $1 placeholder and other $\{0, 0, 0, 0, 1, 1, 1, 1\}$ for $2, generating 8 coverpoints in total. The syntax {}[] in the second coverpoint refers to a list.

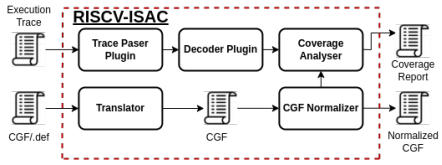

Figure 3: *Updated RISCV-ISAC Tool flow*

Listing 1: *Coverpoint format example for the Translator*

```
pmp_cover:                                                          1
  config:                                                           2
    - check ISA:=regex(.*I.*Zicsr.*)                                3
  mnemonics:                                                        4
    "{lw,sw,${custom_cov},csrrw}": 0                                5
  csr_comb:                                                         6
    # Braces and placeholder features                               7
    (pmpcfg{{0 ... 7}>>2} & 0x80 == 0x80) and (old(                 8
        "pmpaddr$1")) ^ (pmpaddr$2) == 0x00:0
    # List and more braces features                                 9
        # Example for understanding purposes                        10
    (pmpcfg{0 ... 3} & 0x80 == 0x80) and pmpcfg{4                   11
        ... 7}[$1] ^ (pmpaddr$2) == 0x00:0
```

Conclusions and Future Work

This paper presents two key advancements: features support for privileged architecture in RISC-V ISAC and a efficient method of writing coverpoints. Our future plans involve expanding this support to hypervisor extension and interrupt testing, thereby improving the comprehensiveness of the coverage analysis tool.

References

[1] Pawan Kumar S et al. "Automating Generation and Maintenance of a High-Quality Architectural Test Suite for RISC-V". In: *Proceedings of the Sixth Workshop on Computer Architecture Research with RISC-V, Co-located with ISCA* (2022). URL: https://carrv.github.io/2022/papers/CARRV2022_paper_2_Kumar.pdf.

[2] RISC-V. *Sail RISC-V model*. https://github.com/riscv/sail-riscv. 2024.

[3] RISC-V International. *Spike, a RISC-V ISA Simulator*. https://github.com/riscv/riscv-isa-sim. 2024.

[4] RISC-V International. *RISC-V Architectural Tests*. https://github.com/riscv/riscv-arch-test. 2024.

[5] RISC-V Software Source. *RISC-V CTG, Compliance Test Generator*. https://github.com/riscv-software-src/riscv-ctg. 2024.

[6] RISC-V Software Source. *RISC-V ISAC, a coverage analyser*. https://github.com/riscv-software-src/riscv-isac. 2024.