Bridging RISC-V Core Verification and PSS : A Portable-Stimulus Stress-Testing Approach

Darshan Shivsingh Patel

Swapnil Amol Sindhur

Problem Statement/Introduction

The expanding RISC-V ecosystem spans from tiny microcontrollers to complex superscalar CPUs, creating diverse verification challenges.

ISA compliance tests ensure instruction legality but miss critical microarchitectural corner cases—data hazards, branch prediction faults, and resource contention—that cause silicon failures.

Traditional verification limitations:

- ➢ Limited functional coverage for complex cores
- ➢ Non-reusable stress-test scenarios across platforms

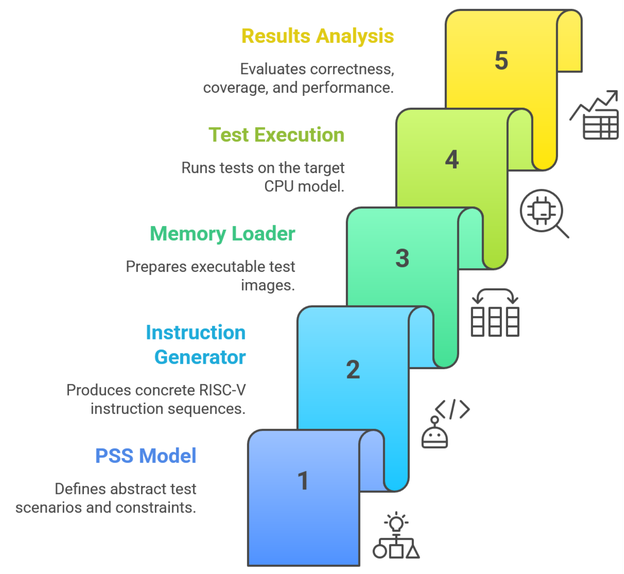
The Portable Stimulus Standard (PSS) enables single-source, constraint-driven tests reusable from simulation to silicon, yet remains underutilized for CPU stress testing.

Our contribution: A PSS-based methodology targeting critical microarchitectural behaviors, demonstrated on a RISC-V core with superior coverage and bug detection compared to traditional approaches.

Proposed Methodology/Advantages

PSS-Based RISC-V Stress Testing Framework

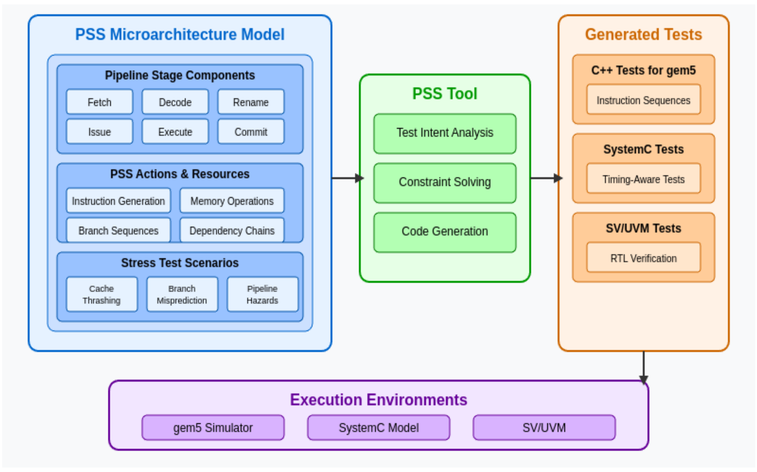
Our methodology leverages PSS to generate sophisticated instruction scenarios targeting microarchitectural stress conditions through a systematic five-stage flow



Key Advantages:

- ✓ Single-Source Portability: Tests reusable across simulation, emulation, and silicon
- ✓ Targeted Stress Generation: Creates arithmetic-heavy sequences, branch-intensive patterns, and hazard-prone combinations
- ✓ Architecture Scalability: Adapts from simple in-order to complex superscalar designs

Implementation Details



A single PSS microarchitecture model defines pipeline stages, actions, and stress scenarios. The PSS tool uses this to generate targeted, reusable test cases for diverse environments (gem5, SystemC, SV/UVM). These tests are executed in multiple simulators or verification setups, enabling thorough RISC-V core validation with one unified, portable flow that boosts coverage and uncovers subtle bugs.

Proof-of-Concept Implementation

- Targets a single-cycle in-order RISC-V core implementing the RV32I base instruction set with basic CSR support.
- Chosen for clear visibility of instruction execution behavior by eliminating pipeline hazards and out-of-order complexities.
- Focuses on validating core methodology principles via:
  - ✓ Instruction mix distribution testing stressing decoder logic under extreme skew conditions
  - ✓ Branch-intensive patterns targeting basic prediction mechanisms
  - ✓ Dependency-rich sequences validating forwarding paths within single-cycle execution
- Demonstrates methodology scalability by reusing the same Portable Stimulus Standard (PSS) model structure for more complex cores with only constraint parameter adjustments.

```
component pss_top {

    riscv_core_c riscv_core;

    action riscv_microarch_stress_test_a {

        rand int max_instruction;

        riscv_core_c::basic_test_a basic_test;
        riscv_core_c::mixed_stress_a mixed_stress;
        riscv_core_c::war_hazard_a war_hazard;
        riscv_core_c::arithmetic_stress_a arithmetic_stress;
        riscv_core_c::branch_stress_a branch_stress;
        riscv_core_c::memory_stress_a memory_stress;
        riscv_core_c::raw_hazard_a raw_hazard;
        riscv_core_c::waw_hazard_a waw_hazard;

        activity {
            sequence {
                    basic_test with {
                        basic_test.max_instruction == this.max_instruction;
                     };
                    mixed_stress with {
                        mixed_stress.max_instruction == this.max_instruction;
                     };
                    war_hazard with {
                        war_hazard.max_instruction == this.max_instruction;
                     };
                    arithmetic_stress with {
                        arithmetic_stress.max_instruction == this.max_instruction;
                     };
                    branch_stress with {
                        branch_stress.max_instruction == this.max_instruction;
                     };
                    memory_stress with {
                        memory_stress.max_instruction == this.max_instruction;
                     };
                    raw_hazard with {
                        raw_hazard.max_instruction == this.max_instruction;
                     };
                    waw_hazard with {
                        waw_hazard.max_instruction == this.max_instruction;
                     };
                };
            }
        }
    };
```

A sample PSS test specification for RISC-V microarchitecture validation

Results Table

CPU Pipeline Visualization

Filter by Sequence #: Enter sequence number | Filter by PC: Enter PC value | Filter by Disassembly: Enter instruction text | Status: All Instructions | Apply Filters | Reset

Table View | Gantt Chart

Committed | Not Committed | f = fetch | d = decode | n = rename | p = dispatch | i = issue | c = complete | r = retire

Seq #	Status	PC.uPC	Disassembly	Pipeline Activity	Fetch Time	Retire Time	Latency
1	Committed	0x000100b0.0	add s7, a6, s8			117000	7.00
2	Committed	0x000100b4.0	or s10, a7, s4			117000	7.00
3	Committed	0x000100b8.0	add s8, a7, a0			117000	7.00
4	Committed	0x000100bc.0	or a2, a4, s9			117000	7.00
5	Committed	0x000100c0.0	xor s4, t3, a2			212000	7.00
6	Committed	0x000100c4.0	sub s8, s4, s7			213000	8.00
7	Committed	0x000100c8.0	or a4, ra, s0			213000	8.00
8	Committed	0x000100cc.0	add a2, a7, s9			213000	8.00
9	Committed	0x000100d0.0	sub ra, s2, ra			213000	8.00
10	Committed	0x000100d4.0	sub gp, s5, sp			213000	8.00
11	Committed	0x000100d8.0	and t1, a3, s9			213000	8.00
12	Committed	0x000100dc.0	and s11, s2, a7			213000	8.00
13	Committed	0x000100e0.0	and a5, t5, a4			213000	7.00
14	Committed	0x000100e4.0	xor s2, t0, s6			214000	8.00
15	Committed	0x000100e8.0	xor t2, t5, s10			214000	8.00
16	Committed	0x000100ec.0	or a6, s3, s7			214000	8.00
17	Committed	0x000100f0.0	or t0, s8, a3			214000	8.00
18	Committed	0x000100f4.0	add a3, a6, tp			214000	8.00

Conclusion

- ✓ We demonstrated a PSS-based microarchitectural stress-testing methodology as a viable, scalable approach for RISC-V processor verification.
- ✓ The proof-of-concept on a single-cycle in-order core validated the methodology's ability to generate diverse, targeted scenarios and reveal subtle microarchitectural behaviors.
- ✓ Modular, abstraction-driven design enables reuse across multiple CPU implementations and verification platforms, bridging high-level test intent with concrete stress testing.
- ✓ The approach is readily scalable to complex out-of-order superscalar architectures with advanced pipeline, hazard, and cache-hierarchy modeling.
- ✓ Future work will incorporate ML-guided constraint tuning, formal verification integration, and standardized PSS scenario libraries to further enhance coverage and efficiency.
- ✓ This work positions PSS as a valuable tool for improving design quality, coverage, and verification productivity in next-generation RISC-V designs.

Takeaway:

The presented PSS-based microarchitectural stress-testing methodology not only advances the rigor and portability of RISC-V verification but also establishes a scalable, efficient foundation that empowers verification teams to confidently tackle increasingly complex CPU designs - ushering in a new era of robust, scalable processor validation.

References

[1] A. Waterman and K. Asanović, The RISC-V Instruction Set Manual, Volume I: User-Level ISA, RISC-V Foundation, 2019.
[2] S. Ahmadi-Pour, V. Herdt, and R. Drechsler, "Constrained Random Verification for RISC-V: Overview, Evaluation and Discussion," in Proc. Methods and Description Languages for Modeling and Verification of Circuits and Systems, 2021, pp. 1-8.
[3] R. Hafner, S. Weiss, and D. Große, "Survey of Verification of RISC-V Processors," Springer Journal of Electronic Testing, vol. 41, no. 2, pp. 127-149, May 2025.
[4] Accellera Systems Initiative, "Portable Test and Stimulus Standard Version 3.0," Aug. 2024.
[5] T. Anderson et al., "User experiences with the portable stimulus standard," in Proc. DVCon, San Jose, CA, USA, 2023.
[6] J. Bergeron, E. Cerny, A. Hunter, and A. Nightingale, Verification Methodology Manual for SystemVerilog. New York: Springer, 2006
[7] A. Hunter et al., "Automatic formal verification of RISC-V pipelined microprocessors," in Advances in Information Security, Springer, 2023, ch. 11, pp. 201-219