Sail-RISC-V and Spike for RISC-V Vector: Toward Consistent Golden Reference Behavior

Manfred Schlägl, Katharina Ruep and Daniel Große

Institute for Complex Systems, Johannes Kepler University Linz, Austria
manfred.schlaegl@jku.at, katharina.ruep@jku.at, daniel.grosse@jku.at

Abstract

In recent years, the executable specification generated from Sail-RISC-V has increasingly been considered as a successor to the widely used Spike ISA Simulator as golden reference for RISC-V, including the complex and highly configurable RISC-V Vector Extension (RVV). In this paper, we compare the RVV behavior of Sail-RISC-V against Spike using the automated testing framework RVVTS. While Sail-RISC-V largely matches Spike under positive testing (0.23% deviations), negative testing reveals substantially more deviations (3.73%), highlighting remaining issues in Sail-RISC-V's RVV instruction validity checking under dynamic configurations.

Introduction

Since the early days of RISC-V, the *Spike* ISA Simulator [1] has served as the de-facto golden reference model and has been widely used in verification frameworks and architectural validation flows [2, 3, 4]. In recent years, the executable specification generated from the *Sail-RISC-V* model [5] has increasingly emerged as a candidate for the next-generation golden reference due to its formally grounded specification approach. Both models include the ratified and highly relevant *RISC-V Vector Extension* (RVV) 1.0, which introduces powerful data-level parallelism for RISC-V. However, creating and verifying RVV implementations is challenging not only because of the large number of specified instructions, but also because their behavior depends heavily on dynamic architectural parameters such as vector length, element width, masking, and register grouping. Consequently, the effective state and parameter space of RVV instructions is orders of magnitude larger than that of scalar instructions. Beyond conventional positive testing with valid instruction sequences, robust verification of RVV must therefore also include *negative testing* [6] to ensure correct handling of invalid instructions and configurations. In RVV, such invalid cases arise for example when a vector instruction uses an illegal register group or an unsupported element type under the current configuration. The recently presented open-source *RVVTS* framework addresses these challenges by combining coverage-guided test generation targeting positive and negative testing with automated test execution, failure code minimization and single-instruction isolation [7].

Using *RVVTS*, extended with support for *Sail-RISC-V* as a *Design Under Test* (DUT) (not covered in this paper), we compare the behavior of RVV in *Sail-RISC-V* against the former golden reference *Spike*. Negative testing with invalid instruction sequences reveals 3.73% deviations, showing that the validation of RVV instruction legality under certain dynamic configurations still requires further attention in *Sail-RISC-V*. Additionally our results demonstrate the effectiveness of *RVVTS* for systematically detecting, minimizing, and analyzing deviations in RVV implementations. To support the alignment and improvement of reference-model behavior, we provide *RVVTS* with *Sail-RISC-V* support, the generated RVV test sets, the generated test reports, and the minimized test cases as open source on GitHub[1].

Spike vs. *Sail-RISC-V*

First, we use the coverage-guided generator in *RVVTS* to produce four compact, high-coverage test sets for RV32 and RV64, targeting both positive and negative testing of RVV (with `VLEN` = 128 bit). Table 1 summarizes their configuration, the number of test cases and RVV instructions, and the achieved functional coverage. The *Invalid+Valid Sequences* (IVS) sets contain trap-triggering code sequences for negative testing, whereas *Valid Sequences* (VS) sets contain only non-trapping sequences for positive testing. Each set comprises tens of thousands of test cases, hundreds of thousands of RVV instructions, and achieves >94% functional coverage.

Next, we use *RVVTS* to run the generated test sets on *Sail-RISC-V* by executing each case on *Spike* and *Sail-RISC-V* and comparing the resulting *Machine States* (registers, CSRs, #traps, etc.). Any deviation of the *Machine States* is flagged as a potential failure, for which *RVVTS* automatically isolates the triggering instruction and produces a minimized test. The far-right of Table 1 summarizes the detected deviations. The deviation rates for RV32 and RV64 are of the same order of magnitude and are therefore consolidated in the last column. Although *Spike* and *Sail-RISC-V* behave very similarly in positive testing (VS), with deviations of 0.23%, they differ notably in negative testing (IVS), with deviations of 3.73%.

[1] https://github.com/ics-jku/RVVTS_SailRV_Spike

Table 1: *Test Sets pre-generated with* RVVTS *and applied on* Sail-RISC-V *(git hash a33475aeb8)*

Test Set	RISC-V Config	#Test Cases	#RVV Instr.	Functional Coverage (*riscvOVPsim* RVV) Points	Percent	Deteted Deviations (% w.r.t. #Test Cases)	
Invalid+Valid Sequences **(IVS)**	RV32	36,016	289,512	30,606 / 31,894	95.96	1,232 (3.42%)	**2,868 (3.73%)**
	RV64	40,939	329,145	31,765 / 33,076	96.04	1,636 (4.00%)	
Valid Sequences **(VS)**	RV32	34,578	253,309	30,126 / 31,894	94.46	95 (0.27%)	**194 (0.23%)**
	RV64	50,760	372,041	31,179 / 33,076	94.26	99 (0.20%)	

Table 2: *Deviation Categories on* Sail-RISC-V

Deviation Category	Detected Deviations IVS (% w.r.t. SUM)	VS (% w.r.t. SUM)
① ***Invalid Accept*** (Missing trap)	**1,803** 62.87%	0 0.00%
② ***Assertion: Valid Reject*** (*Sail-RISC-V* terminated)	**1,038** 36.19%	0 0.00%
③ ***Deviation in Results*** (e.g. registers, CSRs, ...)	16 0.56%	**191** 98.45%
④ ***Assertion: Invalid Reject*** (*Sail-RISC-V* terminated)	**11** 0.38%	3 1.55%
SUM (100.00%)	2,868	194

Using the comprehensive reports automatically generated by *RVVTS*, we analyze the detected deviations in detail. Table 2 lists four main categories revealed by IVS and VS, ordered by frequency. For didactic clarity, we discuss ①, ②, and ④ first, followed by ③.

① covers deviations where *Spike* rejects an instruction (invalid-instruction trap), while the instruction is executed by *Sail-RISC-V* (no trap). By construction, these deviations are detected only by IVS, which is reflected in the results (no cases for VS). Given that *Spike* includes an older and therefore more mature RVV implementation, we hypothesize that *Sail-RISC-V* does not yet fully model certain prohibited RVV instruction cases under specific configurations (e.g., element type, grouping, etc.).

② covers deviations where *Spike* rejects an instruction, whereas the *Sail-RISC-V* model terminates on an assertion. These deviations are also only detected by IVS. Although *Sail-RISC-V* detects these invalid cases, they are not rejected properly within the model via a trap. Instead, the cases are caught deep in the implementation by assertions, leading to a termination, which is clearly incorrect behavior. This confirms the hypothesis above that *Sail-RISC-V* has not yet properly covered certain prohibited cases.

④ covers deviations where *Spike* accepts an instruction, whereas the *Sail-RISC-V* model terminates on an assertion. In contrast to ②, these deviations indicate cases in which assertions in *Sail-RISC-V* are too strict. For all $11+3=15$ cases, *RVVTS* automatically isolates the instruction `vrgatherei16.vv`. All assertions relate to the use of supposedly invalid vector register groups. After studying the RVV specification, we can confirm that *Spike* is correct and that these are indeed bugs in *Sail-RISC-V*.

③ includes all deviations where only results (e.g., register and CSR values) differ. These deviations are mostly detected by VS. For all $16+191=207$ cases, *RVVTS* automatically isolates four instructions, namely `vf(w)redusum.vs` and `vssubu.vv|vx`. For the former, the deviation can be explained by different order of floating-point element processing and from the fact that floating-point arithmetic is not distributive. For the latter, a closer inspection shows that *Sail-RISC-V* does not set the saturation bit `vcsr.vxsat` for subtractions where the second operand is greater than the first. After investigating the *Sail-RISC-V* code, we can confirm this as a bug.

Conclusions

Overall, our results show that RVV in *Sail-RISC-V* mostly corresponds to the former golden reference *Spike* under positive testing with valid code sequences (VS), with only **0.23%** deviations. However, negative testing with invalid code sequences (IVS) reveals substantial deviations of **3.73%** relative to *Spike*. In particular, the validation of RVV instruction legality under specific configurations (e.g., element type, grouping, etc.) in *Sail-RISC-V* requires further attention. These findings also underscore the value of *RVVTS*– its comprehensive support for positive and negative RVV testing and its automated code minimization with instruction isolation.

RVVTS with support for *Sail-RISC-V*, the generated RVV test sets, the generated test reports, and the minimized test cases are available as open source on GitHub.

Acknowledgments

This work has partially been supported by the LIT Secure and Correct Systems Lab funded by the State of Upper Austria.

References

[1] *Spike RISC-V ISA Simulator*. https://github.com/riscv/riscv-isa-sim. 2026.

[2] *RISC-V Vector Tests Generator*. https://github.com/chipsalliance/riscv-vector-tests. 2026.

[3] *RISCV-DV*. https://github.com/google/riscv-dv. 2026.

[4] Alexandre Joannou et al. "Randomized Testing of RISC-V CPUs Using Direct Instruction Injection". In: *IEEE Design and Test* 41.1 (2024), pp. 40–49. DOI: 10.1109/MDAT.2023.3262741.

[5] *Sail RISC-V: Formal Specification of the RISC-V ISA*. https://github.com/riscv/sail-riscv. 2026.

[6] Vladimir Herdt, Daniel Große, and Rolf Drechsler. "Closing the RISC-V Compliance Gap: Looking from the Negative Testing Side". In: *DAC*. 2020, pp. 1–6. DOI: 10.1109/DAC18072.2020.9218629.

[7] Manfred Schlägl and Daniel Große. "Single Instruction Isolation for RISC-V Vector Test Failures". In: *ICCAD*. 2024, 156:1–156:9. DOI: 10.1145/3676536.3676755.