Late Breaking Results: Float Fight - Verifying Floating-Point Behavior in RISC-V Simulators

Katharina Ruep Manfred Schlägl Daniel Große
Institute for Complex Systems, Johannes Kepler University Linz, Austria
katharina.ruep@jku.at manfred.schlaegl@jku.at daniel.grosse@jku.at

Abstract—**In this paper, we enhance *RVVTS*, an open-source framework for testing RISC-V vector instructions, to enable comprehensive floating-point (FP) verification across various RISC-V simulators and FP libraries. Our enhanced *RVVTS*, referred to as *FP-RVVTS*, adds support for the RISC-V FP extensions (F, D, Zfh) through a novel context-free grammar specification with annotations, strengthened automatic single-instruction isolation, and improved failure cause analysis.**

In the experiments we show that *FP-RVVTS* generates FP test sets achieving over 95% functional coverage, reveals critical bugs in several RISC-V simulators, and, using isolated instructions, supports to narrow down the causes of failures.

I. Introduction

Floating-point (FP) computation is pervasive in modern systems, from scientific simulation and signal processing to computer graphics, machine learning, and safety-critical applications (see e.g. [1]–[3]). Incorrect FP behavior in these domains can cause subtle numerical errors that are hard to detect yet may compromise system reliability. At the same time, FP arithmetic is challenging to implement and verify: it approximates real numbers with finite precision, supports special values such as *Not a Number*s (NaNs), infinities, and subnormals, and offers multiple rounding modes and exception flags, leading to many corner cases that are rarely exercised by conventional testing [4], [5].

The IEEE 754 standard [6], which specifies FP semantics, has been revised and extended over time, adding operations and tightening requirements. This ongoing evolution further increases the verification effort, as hardware implementations, simulators, and software libraries must remain compliant with a changing specification while still meeting performance and area constraints. These challenges are particularly relevant for open *Instruction Set Architecture*s (ISAs) such as RISC-V [7], [8], where a diverse set of processors, simulators, and toolchains coexist and evolve in parallel.

In this work, we examine the FP backbone of modern RISC-V simulators: the *SoftFloat* (SF) [9] and the *FloppyFloat* (FF) [10] libraries on which they rely. SF is a widely used, bit-accurate IEEE 754 reference library from UC Berkeley, whereas FF is a hybrid library designed to offer SF-compatible semantics at higher performance [11]. Concretely, we consider the simulators *RISC-V VP++* [12] (in variants using SF and FF), *Spike FF* [10] (a performance-tuned version of the official RISC-V golden reference simulator *Spike* from RISC-V International), and *QEMU* [13], which also employs SF. **With the proposed *FP-RVVTS*, we systematically verify the FP behavior of these RISC-V simulators,** effectively staging a direct "float fight" between their underlying SF- and FF-based FP implementations.

There is substantial work on general RISC-V instruction generation and processor verification (e.g. [14]–[22]), as well as open-source constrained-random generators such as Google's RISCV-DV [23]. These approaches mainly target overall ISA correctness and compliance, but are not specialized for high-coverage verification of the RISC-V FP extensions or the interplay of different FP libraries. For example, [21] fuzzes RISC-V processors with long, valid programs and has also uncovered FP bugs, yet it still treats FP as just one aspect of overall processor correctness and does not focus on systematic FP-centric verification across different simulators or FP libraries. RISCV-DV initializes FP registers only once at the beginning of a test with special values, which limits operand diversity and thus FP coverage, as also discussed in [24]. To the best of our knowledge, the only RISC-V-specific work that directly targets FP instruction verification is [24], which proposes coverage and constraint-based test generation for RV32F on one concrete RTL processor. However, their flow is not open-source, is limited to 32-bit single precision, and does not address off-the-shelf RISC-V simulators, alternative FP libraries, or techniques to improve failure analysis.

II. FP-RVVTS

We introduce *FP-RVVTS*, an enhanced version of *RVVTS* [18], an open-source framework for positive and negative testing [25] of RISC-V vector instructions, to enable comprehensive FP verification across various RISC-V simulators and FP libraries. *FP-RVVTS* incorporates a context-free FP grammar to cover the RISC-V FP extensions and enriches it with dependency annotations to enable more effective reductions. As in *RVVTS*, the generated tests are automatically executed on both a reference simulator and *Design Under Test* (DUT), where differences in the resulting architectural states are detected and used as input for isolation and minimization of failing instructions.

Floating-point Grammar: To generate valid FP assembly, the *Instruction Sequence Generator* (ISG) is augmented with a RISC-V FP grammar. We cover the F, D, and Zfh extensions for both RV32 and RV64, configured according to the target architecture. The grammar reflects these configurations, in particular for conversion instructions (e.g., `fcvt.s.d` and `fcvt.l.s`). In RISC-V, the rounding mode can be set globally via the *Control and Status Register* (CSR) field `frm` or per

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

TABLE I: Overview of Test Sets generated by *FP-RVVTS*

TS	Coverage	Float Ratio	Tests
pos	95.44%	51.80% (42,187/81,447)	5,353
neg	96.73%	53.48% (43,530/81,394)	5,340

instruction via a suffix (e.g., `fsqrt.d f1, f2, rne`). We support both mechanisms, individually and in combination. The grammar enforces architectural support (unsupported conversions do not compile), configures `frm`, and allows independent per-instruction rounding-mode suffixes.

Dependency Annotation: In case of a failure, *FP-RVVTS* improves test case reduction by using dependency annotations in the grammar. While *RVVTS* can pinpoint a failure to a single instruction, *FP-RVVTS* also produces a minimal snapshot of the system state immediately before that instruction – containing only the relevant registers and configuration CSRs – thereby easing debugging significantly.

III. Experiments

In our experiments we consider 8 *test setups*, a combination of 4 DUTs and 2 test sets. *FP-RVVTS* and the DUTs have been configured for *RV64IFD_Zfh*, providing the broadest FP coverage supported across all tested simulators. The two test sets are generated using *FP-RVVTS*, one targeting *positive testing* (pos) and one targeting *negative testing* (neg). For both test sets, Table I lists, from left to right, the functional coverage achieved[1], the ratio of FP to total instructions, and the number of tests generated. As can be seen, *FP-RVVTS* achieves more than 95% functional coverage for both of them.

Table II summarizes the test results for the 8 test setups consisting of *ID*, *DUT*, and *TS* (Test Set). Recall that *FP-RVVTS* uses *Spike* (with SF) as golden reference model, i.e., the official RISC-V reference simulator maintained by RISC-V International. As DUTs, we use *RISC-V VP++* [12] with either FF [11] or SF [9] as the FP library, *Spike FF* [10], and QEMU [13]. The column *Fails* shows the number of failures reported by *FP-RVVTS*, and the last column provides the corresponding isolated instructions. As can be seen, our approach found a huge number of failures for *RISC-V VP++* independent of the used FP library (ID 1–4). For *Spike FF* also 2 failures have been determined (ID 5–6). In contrast, for *QEMU* no failures have been identified. What also becomes evident from Table II is the importance of negative testing as in most cases the number of failures is higher in comparison to the corresponding positive testing. In the following, we discuss how *FP-RVVTS* helps in narrowing down the cause of the failure. For this analysis, we consider the isolated instructions shown in the rightmost column of Table II, which are automatically determined by our approach leveraging the strengthened single instruction isolation technique. For example, the instruction `fmax.d` is isolated (among others) in ID 3 and 4 with *RISC-V VP++ SF* as DUT. Since the reference model *Spike* also uses SF, this indicates that the failure is caused by an incorrect use of the SF library in *RISC-V VP++*.

By analyzing other isolated instructions and their DUTs, including associated test sets, we can analogously derive specific

[1]We use riscvOVPsim [26] measuring only F and D extensions of RISC-V.

TABLE II: Reported Failures for different *Test Setups*

ID	DUT	TS	**Fails**	Isolated Instructions
1	RISC-V VP++ FF	pos	**852**	`fcvt.wu.d/s`
2	RISC-V VP++ FF	neg	**1858**	`fld/w/h`, `fsd/w/h`, `fmv.x.h`, `fcvt.wu.d/s`, `fcvt.lu.d`
3	RISC-V VP++ SF	pos	**373**	`fmax.d/s`, `fmin.d/s`
4	RISC-V VP++ SF	neg	**1758**	`fld/w/h`, `fsd/w/h`, `fmax.d/s/h`, `fmin.d/s`, `fmv.x.h`
5	Spike FF	pos	**1**	`fnmadd.d`
6	Spike FF	neg	**1**	`fdiv.s`
7	QEMU	pos	**0**	-
8	QEMU	neg	**0**	-

Golden reference: Spike (with SF) from RISC-V International

TABLE III: Cause Analysis of Isolated Instructions (Selection)

Instruction	**Location**	**Bug**
`fmax.d`	RISC-V VP++ SF	NaN handling
`fld`	RISC-V VP++	FP enable check
`fcvt.wu.d`	RISC-V VP++ FF	NaN handling
`fdiv.s`	Spike FF	flag generation
`fmv.x.h`	RISC-V VP++	precision mismatch handling

locations. Sometimes it is even possible to narrow down the problem further (cf. discussion of `fld` below). An overview for the cause analysis (selection) are given in Table III. Here, *Location* represents the result of the previous analysis, whereas *Bug* describes the cause of the problem. Let's examine the bug for the four colored isolated instructions:

`fmax.d`: This instruction receives two FP values and returns their maximum. In all failing cases, *Spike* and the DUT (*RISC-V VP++ SF*) disagree only when at least one input is NaN. Spike returns the non-NaN operand, whereas the DUT returns NaN. Per the RISC-V specification [27, Sec. 20.6], the semantics changed in v2.2: before v2.2 the result was NaN; since v2.2 it is the non-NaN operand. Thus, the bug stems from *RISC-V VP++* implementing the pre-v2.2 behavior.

`fld`: This instruction loads an FP value from memory into an FP register. Failures only occur in the negative tests (ID 2 and ID 4) with both FP libraries, indicating a *RISC-V VP++* bug rather than a library issue: when the FP extension is disabled, *Spike* correctly raises an exception, but *RISC-V VP++* executes the instruction because its load logic omits the required FP-extension-enabled check.

`fcvt.wu.d`: *RISC-V VP++ FF* correctly computes the NaN result as $2^{32} - 1$ but then incorrectly zero-extends the 32-bit value to 64 bits instead of sign-extending it as required by the RISC-V specification.

`fdiv.s`: This is an arithmetic instruction with a single failing case. The mismatch concerns the underflow exception flag: *Spike* sets this flag as intended, whereas *Spike FF* does not, indicating a bug in *Spike FF*.

To summarize, *FP-RVVTS* systematically verified the FP backbones of modern RISC-V simulators and, in the "float fight", their SF/FF libraries, exposing a diverse set of previously unknown bugs – from outdated NaN semantics and missing FP-enable checks to incorrect sign extension and exception-flag handling. *FP-RVVTS* precisely narrows down each failure, making these issues easy to pinpoint and act upon, and thus may emerge as a practical open-source approach to "harden" the floating-point ecosystem of RISC-V.

References

[1] D. Delmas, E. Goubault, S. Putot, J. Souyris, K. Tekkal, and F. Védrine, "Towards an industrial use of FLUCTUAT on safety-critical avionics software," in *Formal Methods for Industrial Critical Systems (FMICS 2009)*, 2009, pp. 53–69.

[2] M. Cococcioni, E. Ruffaldi, and S. Saponara, "Exploiting posit arithmetic for deep neural networks in autonomous driving applications," in *International Conference of Electrical and Electronic Technologies for Automotive (AUTOMOTIVE 2018)*, 2018, pp. 1–6.

[3] J. S. Kole and F. J. Beekman, "Evaluation of accelerated iterative x-ray CT image reconstruction using floating point graphics hardware," *Physics in Medicine and Biology*, vol. 51, no. 4, pp. 875–889, 2006.

[4] D. Monniaux, "The pitfalls of verifying floating-point computations," *ACM Transactions on Programming Languages and Systems*, vol. 30, no. 3, pp. 12:1–12:41, May 2008.

[5] J.-M. Muller, N. Brunie, F. de Dinechin, C.-P. Jeannerod, M. Joldes, V. Lefèvre, G. Melquiond, N. Revol, and S. Torres, *Handbook of Floating-Point Arithmetic*, 2nd ed. Birkhäuser, 2018.

[6] *IEEE Standard for Binary Floating-Point Arithmetic*, IEEE Computer Society Std. IEEE Std 754-1985, 1985.

[7] A. Waterman and K. Asanović, *The RISC-V Instruction Set Manual; Volume I: Unprivileged ISA*, SiFive Inc. and CS Division, EECS Department, University of California, Berkeley, 2019.

[8] ——, *The RISC-V Instruction Set Manual; Volume II: Privileged Architecture*, SiFive Inc. and CS Division, EECS Department, University of California, Berkeley, 2019.

[9] J. R. Hauser, "Berkeley SoftFloat," https://github.com/ucb-bar/berkeley-softfloat-3.

[10] N. Zurstraßen, "Modified Spike using FloppyFloat," https://github.com/not-chciken/riscv-isa-sim, 2025.

[11] N. Zurstraßen, N. Bosbach, and R. Leupers, "FloppyFloat: An open source floating point library for instruction set simulators," in *Design, Automation and Test in Europe Conference (DATE)*, 2025, pp. 1–6.

[12] M. Schlägl, C. Hazott, and D. Große, "RISC-V VP++: Next generation open-source virtual prototype," in *Workshop on Open-Source Design Automation (OSDA)*, 2024.

[13] "QEMU a generic and open source machine emulator and virtualizer," https://www.qemu.org, 2025.

[14] V. Herdt, D. Große, and R. Drechsler, "Towards specification and testing of RISC-V ISA compliance," in *Design, Automation and Test in Europe Conference (DATE)*, 2020, pp. 995–998.

[15] V. Herdt, D. Große, E. Jentzsch, and R. Drechsler, "Efficient cross-level testing for processor verification: A RISC-V case-study," in *Forum on Specification and Design Languages (FDL)*, 2020, pp. 1–7.

[16] L. Klemmer and D. Große, "EPEX: processor verification by equivalent program execution," in *ACM Great Lakes Symposium on VLSI (GLSVLSI)*, 2021, pp. 33–38.

[17] N. Bruns, V. Herdt, E. Jentzsch, and R. Drechsler, "Cross-level processor verification via endless randomized instruction stream generation with coverage-guided aging," in *Design, Automation and Test in Europe Conference (DATE)*, 2022, pp. 1123–1126.

[18] M. Schlägl and D. Große, "Single instruction isolation for RISC-V vector test failures," in *IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2024, pp. 156:1–156:9.

[19] J. Xu, Y. Liu, S. He, H. Lin, Y. Zhou, and C. Wang, "MorFuzz: Fuzzing processor via runtime instruction morphing enhanced synchronizable co-simulation," in *USENIX Security Symposium (USENIX)*, 2023, pp. 1307–1324.

[20] P. Liu, W. Hu, D. Liu, X. Han, and Y. Liu, "A RISC-V test sequences generation method based on instruction generation constraints," *Journal of Electronics & Information Technology*, vol. 45, no. 9, pp. 3141–3149, 2023.

[21] F. Solt, K. Ceesay-Seitz, and K. Razavi, "Cascade: CPU fuzzing via intricate program generation," in *USENIX Security Symposium (USENIX)*, 2024, pp. 5341–5358.

[22] A. Joannou, P. Rugg, J. Woodruff, F. A. Fuchs, M. van der Maas, M. Naylor, M. Roe, R. N. M. Watson, P. G. Neumann, and S. W. Moore, "Randomized testing of RISC-V CPUs using direct instruction injection," *IEEE Design & Test*, vol. 41, no. 1, pp. 40–49, 2024.

[23] "RISCV-DV," https://github.com/google/riscv-dv, 2024.

[24] T. Lu, A. Liu, B. Xia, and P. Liu, "Comprehensive RISC-V floating-point verification: Efficient coverage models and constraint-based test generation," in *Design, Automation and Test in Europe Conference (DATE)*, 2025, pp. 1–7.

[25] V. Herdt, D. Große, and R. Drechsler, "Closing the RISC-V compliance gap: Looking from the negative testing side," in *Design Automation Conference (DAC)*, 2020, pp. 1–6.

[26] Imperas, "riscvovpsim," https://github.com/riscv-ovpsim/imperas-riscv-tests, 2025.

[27] "The RISC-V Instruction Set Manual, Volume I: User-Level ISA," RISC-V Foundation, Nov. 2025, document version 20251126.