Efficient debug and trace of RISC-V systems: a hardware/software co-design approach

Oana Lazar[1*], Henrique Mendes[1] and Angelo Maldonado-Liu[2]

[1]Tessent Embedded Analytics, Siemens Electronic Design Automation, UK
[2]Tessent Embedded Analytics, Siemens Electronic Design Automation, USA

Abstract

The development risks of novel RISC-V solutions are exacerbated by bugs which are increasingly complicated to root-cause in ever-larger systems. We propose an end-to-end solution comprised of both hardware and software components for minimally intrusive tracing, logging, and run-stop-step debugging of RISC-V systems.

Introduction

With the rise in adoption of the RISC-V ecosystem comes an increase in complexity of both hardware and software, with use cases including high-performance computing, generative AI, and multi-chiplet technologies. This is both a blessing and a curse: solutions are increasingly taking advantage of RISC-V's customisability, yet also increasing the complexity of issues such as Heisenbugs and Silent Data Corruption, which are expensive to reproduce and solve. This is skyrocketing demands for specialists able to rapidly debug issues to ensure timely product deliveries, all while a "lack of skilled talent is the biggest issue facing the [semiconductor] industry over the next three years" [1].

Software-only solutions such as OpenOCD and GDB address some of these needs with run-stop-step debug control, but this is intrusive and lacks the depth of visibility needed to root-cause complex and timing-sensitive bugs. More insight is needed from the system to allow engineers to identify bugs more efficiently, requiring a more comprehensive approach that can not only provide run control and processor trace, but also provide multiple debug methods and insights into a given RISC-V system.

This shows a demand for solutions that address the need for minimally intrusive debugging whilst being familiar enough to use out-of-the-box to address the skills shortage.

We will discuss the importance of an approach for RISC-V that combines the benefits of both hardware and software, introducing a complete and customisable end-to-end solution. This offers benefits such as multi-core run-stop-step debug control, highly compressed instruction tracing of program execution, and numerous hardware-assisted enhancements. Building upon industry-standard freeware and open-source tools familiar to users, such as GDB, OpenOCD and VSCode, also minimises the ramp-up in learning required to start benefiting from such a solution.

* Corresponding author : oana.lazar@siemens.com

Highly efficient trace of custom instructions

A key benefit of the RISC-V ecosystem is the customisability of its ISA. To verify the correct functioning of custom instructions in a novel hart, more in-depth visibility over program execution is required than can be provided with run-stop-step debuggers, but without the debuggers affecting program execution order or timing.

The proposed approach provides an Instruction Trace Encoder with out-of-the-box custom instruction support, enhancing the use of GDB to non-intrusively and efficiently trace any RISC-V program's execution path over time.

The trace encoder used in this approach is fully compliant with the ratified "Efficient Trace for RISC-V (E-Trace)" specification [2]. It provides highly compressed instruction trace, benefiting large systems with heavy workloads, where bandwidth may be limited. The approach's "enhanced" trace encoder boasts an average compression rate of 0.2317 bits per instruction based on preliminary tests with Embench™ benchmarks, with no extensions enabled. Enabling optional extensions such as the Call Counter, Branch Prediction Mode, and Jump Target Cache [2] improves the compression rate by ~40%. These extensions allow for more efficient tracing of program execution, to pinpoint the start of unexpected behaviours and solve complex bugs.

Minimally intrusive logging of program flow

As debuggers such as GDB can potentially interfere with program execution order, developers still fall back on "printf debugging", which is still popular due to its ease of use [3]. However, "printf" interferes with program execution timing, making it difficult to catch timing-sensitive Heisenbugs.

This approach uses a hardware Static Instrumentation (SI) module for hardware-assisted and minimally intrusive logging of program flow, via an intuitive "printf"-like API. FPGA implementations of SI demonstrate that its logs only take 0.2-0.3% of the processing time that their "printf" equivalents take to run, using just a fraction of the number of instructions used for "printf". This minimises timing interference, allowing for timing-sensitive bugs to be caught.

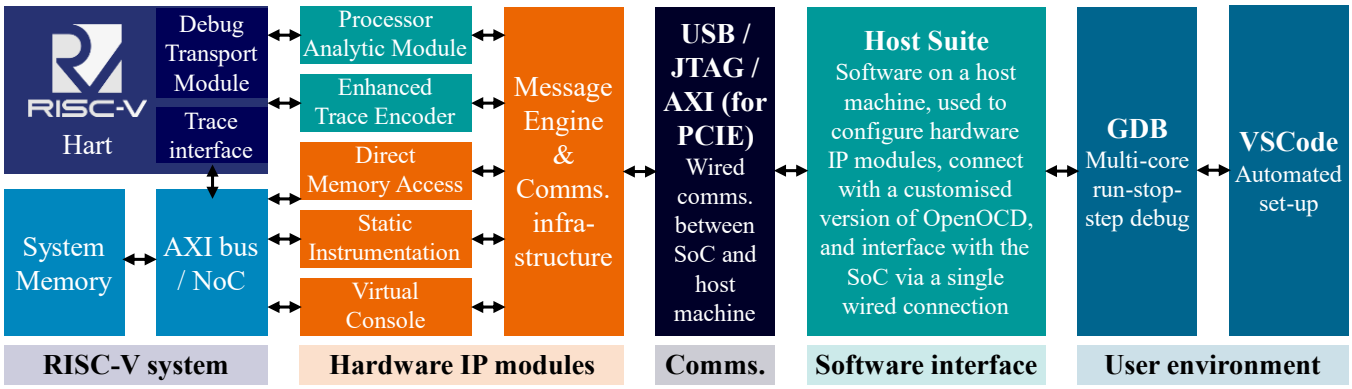

Figure 1: The RISC-V hart (top left), and the hardware (left) and software (right) components of the UltraSight-V solution.

Harnessing hardware/software trade-offs

There exists a trade-off between hardware and software components: hardware is faster at completing a specific task than software, with the cost of using expensive silicon area.

The approach includes hardware IP with low gate counts, such as the Direct Memory Access module. In FPGA tests, this module offers a 200x speed-up in on-chip ELF file loads over software-only GDB, accelerating debug iteration loops.

Inversely, the approach uses a Virtual Console hardware module to leverage the benefits of software multiplexing. This enables the use of multiple "virtual" bi-directional off-chip communication channels, without needing additional pins as in a hardware-only solution. Likewise, a Processor Analytic Module allows the RISC-V hart to be debugged via a single wired connection, be it JTAG or USB.

Maximising benefits of both hardware and software leads to faster debug iterations, with minimal silicon overhead.

Verifying system integration using UVM

Over 42% of FPGA design engineers' time is reported to be spent on verification instead of design [4]. This reveals a demand for robust and automated verification solutions that are delivered along with the hardware IP modules.

The approach thus includes a UVM environment for driving stimuli to a system's communicators, to easily verify the integration of hardware modules within the RISC-V system. This prepackaged solution allows engineers to refocus their time on designing rather than verification.

An end-to-end debug and trace solution

The proposed solution in Figure 1 uses the above hardware modules within a given FPGA or SoC, connected to a RISC-V hart via its existing interfaces. The hardware modules communicate using a message-passing fabric. Message Engine modules pass data off-chip through a single wired interface connected to a host machine. The Host Suite software on the host handles communication with OpenOCD and GDB, for an end-to-end debug and trace solution taking advantage of both hardware and software capabilities.

Conclusion

Through FPGA and SoC implementations, we demonstrate that an approach using both hardware and software to their full extent can provide efficient and minimally intrusive RISC-V debug and trace. UltraSight-V implements this approach in a complete end-to-end solution for debug, trace, and testing of any RISC-V system, maximising developers' usage of extensions and customisations out-of-the-box.

Author biographies

O. Lazar is an Embedded Software Engineer at Tessent Embedded Analytics and holds an MEng in Electronic Engineering from the University of Southampton.

H. Mendes is a Product Engineer at Tessent Embedded Analytics and holds an MSc in Electrical and Computer Engineering from the University of Lisbon.

A. Maldonado-Liu is an Engineer at Tessent Embedded Analytics and holds a B.S. from Portland State University.

References

[1] KPMG LLP. and Global Semiconductor Alliance (GSA), "2024 KPMG Global Semiconductor Industry Outlook", Aug. 28, 2023. https://www.gsaglobal.org/global-semiconductor-industry-outlook-2024/

[2] G. Panesar and I. Robertson, "Efficient Trace for RISC-V," Siemens, ver. 2.0.3, Apr. 19, 2024. [Online]. Available: https://github.com/riscv-non-isa/riscv-trace-spec/releases/download/v2.0.3/riscv-trace-spec-asciidoc.pdf

[3] M. Beller, N. Spruit, D. Spinellis and A. Zaidman, "On the Dichotomy of Debugging Behavior Among Programmers," 2018 IEEE/ACM 40th International Conference on Software Engineering (ICSE), Gothenburg, Sweden, 2018, pp. 572-583, doi: 10.1145/3180155.3180175

[4] "Part 4: The 2022 Wilson Research Group Functional Verification Study - Verification Horizons," Verification Horizons, Nov. 06, 2022. https://blogs.sw.siemens.com/verificationhorizons/2022/11/06/part-4-the-2022-wilson-research-group-functional-verification-study/