2020 DESIGN AND VERIFICATION™ DVCON CONFERENCE AND EXHIBITION EUROPE

A Methodology to Verify Functionality, Security, and Trust for RISC-V Cores

W. W. Chen, OneSpin Solutions, Munich, Germany (*weiwei.chen@onespin.com*)

N. Tusinschi, OneSpin Solutions, Munich, Germany (*nicolae.tusinschi@onespin.com*)

T. L. Anderson, OneSpin Solutions, San Jose, CA, USA (*tom.anderson@onespin.com*)

***Abstract*—Modern processor designs present some of the toughest hardware verification challenges. These challenges are especially acute for RISC-V processor core designs, with a wide range of variations and implementations available from a plethora of sources. This paper describes a verification methodology available to both RISC-V core providers and system-on-chip (SoC) teams integrating these cores. It spans functional correctness, including compliance, detection of security vulnerabilities, and trust verification that no malicious logic has been inserted. Detailed examples of design bugs found in actual RISC-V core implementations are included.**

Keywords—RISC-V; ISA; processor; formal; verification; compliance; security; trust

I. Introduction

Modern processor designs present some of the toughest hardware verification challenges. Such advanced features as multi-stage pipelines, multi-level caches, out-of-order execution, branch prediction, speculative execution, and pre-fetching are hard to design and even harder to verify. There are many combinations of behaviors and corner-case conditions unlikely to be verified by constrained-random simulation testbenches. Further, full confidence in the processor design requires formal verification that the hardware not only does what it is supposed to do, but also that it does not do what it is not supposed to do. This ensures that a design is functionally correct, secure, and trusted. Verification is particularly challenging for RISC-V processor core designs, with many providers and many variations of implementation [1]. A common verification methodology available to both RISC-V core providers and system-on-chip (SoC) teams integrating these cores is required.

II. RISC-V Verification Challenges

The RISC-V project started in 2010 at the University of California at Berkeley to define and implement a fifth-generation reduced instruction set computer (RISC) processor. Its goals included support for a wide variety of diverse implementations suitable for different end applications. This flexibility resulted in three aspects of RISC-V that make it especially difficult to verify. First, the defined instruction set architecture (ISA) has many optional features and possible variations [2, 3]. The processor has 32 registers that can be 32-bit, 64-bit, or 128-bit. The baseline "I" instruction set has an optional "E" version that supports only 16 32-bit registers for embedded applications. Additional instructions that may be supported include:

- "M" extension for integer multiplication/division
- "A" extension for atomic read-modify-write memory accesses
- "F" extension for single-precision (32-bit) floating point
- "D" extension for double-precision (64-bit) floating point
- "Q" extension for quad-precision (128-bit) floating point
- "C" extension for compressed (16-bit) instructions

The optional floating-point instructions add 32 more registers of appropriate width. The RISC-V ISA defines three privilege levels, nine exceptions associated with privileged instructions, and 4096 control and status registers (CSRs). Clearly just checking for compliance to the ISA is a significant challenge. But full verification of a RISC-

V core for functional correctness goes beyond compliance. The ISA allows the definition of custom instructions, and these must be verified to ensure that they work correctly without breaking compliance of the standard instructions. Further, the RISC-V ISA was designed to map to many different microarchitectures, from small controllers to multi-core implementations with the most advanced processor features. A core provider must be able to verify the entire design, including the microarchitectural details, not just ISA compliance. A core integrator may wish to repeat some or all of this verification as an acceptance test. Finally, any register-transfer-level (RTL) design must be statically analyzed to eliminate common coding errors.

III. A RISC-V Verification Methodology

A methodology meeting all the requirements for verification of RISC-V cores must be based on formal technology. Any approach based on simulation or emulation can explore only a tiny percentage of design functionality. There is no way to be sure that constrained-random or hand-crafted tests find all the hardware bugs; only formal can provide this guarantee. Similarly, only formal verification can prove the absence of something, in this case parts of the design that might pose risks for end applications with high security or trust requirements. Such a formal-based methodology has been developed and deployed on multiple RISC-V core designs. This methodology spans functional correctness, security, and trust for RISC-V designs, and can be run by both core providers and core integrators.

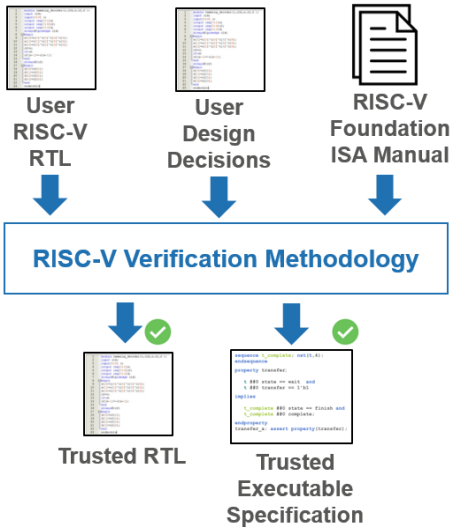

Figure 1. RISC-V verification flow

Inputs to the process includes the core RTL, the ISA compliance rules captured formally, and input from the core provider on design decisions such as the number of pipeline stages, which enables verification of the microarchitecture. Thus, the methodology supports the full range of RISC-V verification and meets all the challenges outlined. Although beyond the scope of this paper, the methodology extends to the full-SoC level, addressing additional chip-level verification challenges such as functional safety and proper core integration.

IV. Functional Correctness

The heart of the methodology is the formalization of the RISC-V ISA as a set of SystemVerilog Assertions (SVA) using a novel "Operational Assertion" library [3, 4]. This approach defines high-level transactions concisely, similar to timing diagrams, and is ideal for capturing the expected results for processor instructions. Each instruction in the RISC-V ISA is captured in a single Operational Assertion that applies to any microarchitectural implementation of the instruction. Formal engines verify that the processor state at the end of each instruction's execution matches the specification. Since this approach is formal, it finds all bugs related to functional correctness and then proves that no further bugs exist. It is flexible enough to handle user extensions beyond the ISA.

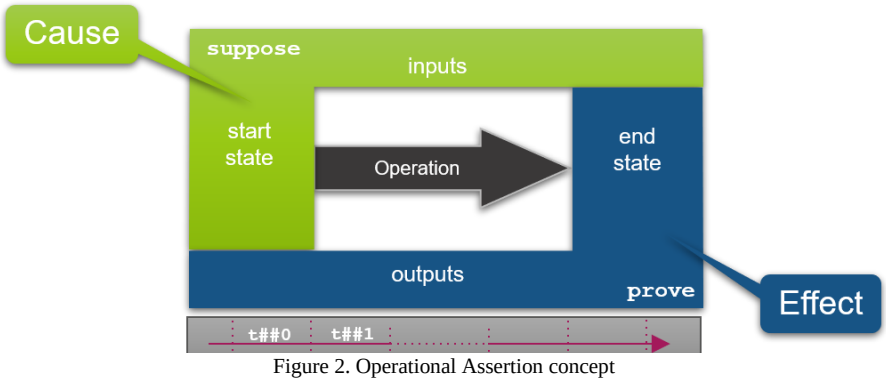

Figure 2. Operational Assertion concept

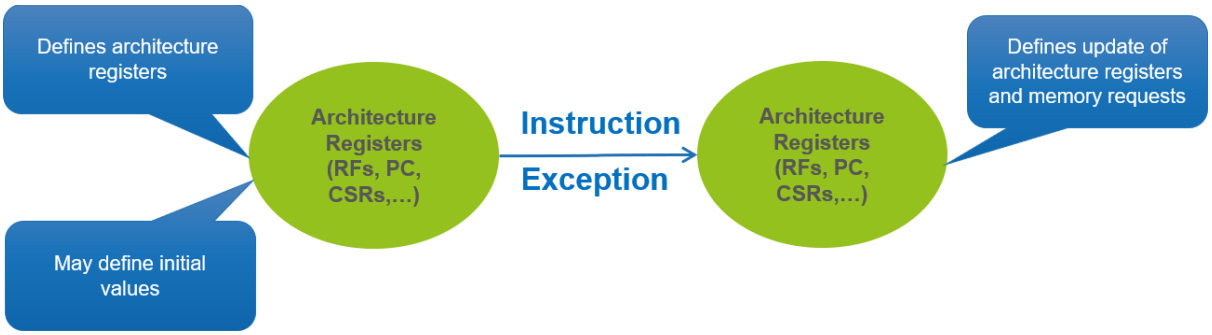

Figure 3. Operational Assertions applied to RISC-V ISA

The formal engines verify the complete RISC-V core, beyond ISA compliance, including the microarchitecture. The methodology also calls for running static analysis on the core to find common design errors. These range from simple syntax mistakes and typographical errors in the RTL to weaknesses that could compromise the final chip. This process is fully automated, so many design teams set it to run on every attempt to check RTL code into a revision-control system.

V. Security Verification

Some applications for RISC-V processors have strict security requirements so that malicious agents cannot exploit vulnerabilities present in the design. Security verification must include a rigorous process to detect all bugs and flaws, establishing precisely what can and cannot happen in the design. The formal-based RISC-V verification methodology includes running a wide range of automated checks on the core RTL design, rapidly eliminating many classes of common coding and design errors. These checks include:

- Structural analysis: syntactic and semantic analysis of source code
- Safety checks: exhaustive verification of the absence of common sequential design operation issues
- Activation checks: proof that all design functions can be executed and are not blocked due to unreachable code

Some of the problems uncovered by these checks represent security risks that could be exploited. For example, an incomplete case statement leaves unspecified what will happen if an unexpected value occurs. Such issues can be found and fixed automatically, early in the design process.

VI. Trust Verification

Inadvertent vulnerabilities are worrisome, but deliberately inserted malicious code (hardware Trojans) are an even bigger concern for trust-critical applications such as autonomous vehicles and nuclear power plants. The development process for cores and SoCs is complex, often with multiple opportunities for introduction of Trojans. Problems may creep in during logic synthesis, either by deliberate action or tool issues. Integration of cores such as RISC-V processors presents extra risk since the SoC team does not know the details of those designs. Tool bugs or intentional netlist modifications can occur during the place-and-route (P&R) process. The RISC-V verification methodology includes two steps to ensure trust in a processor core and extend to any SoC integrating it.

Trust in the design itself relies on formal verification's ability to detect when a design can do something that it is not supposed to do. Beyond verifying compliance to the RISC-V ISA, the methodology ensure that the set of assertions covers the entire core design. This requires the novel GapFreeVerification technology, which detects and reports any specification omissions and errors, holes in the verification plan, and unverified RTL functions. Any unexpected functionality is flagged, since it could potentially be a hardware Trojan inserted during the design process. Core providers want their products to be trusted, and core integrators may want to verify that trust themselves.

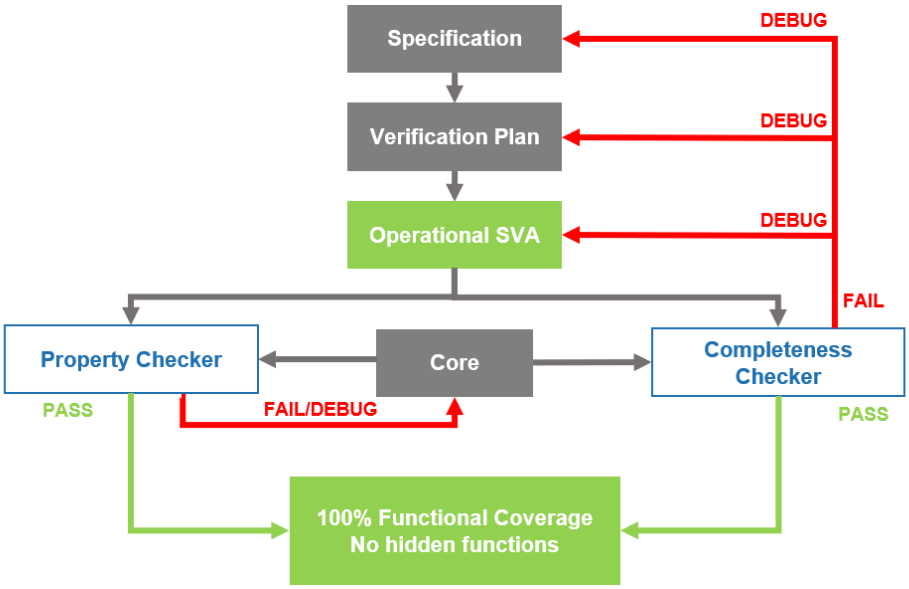

Figure 4. GapFreeVerification flow

For the remainder of the development flow beyond the RTL stage, the RISC-V verification methodology uses formal equivalence checking to ensure that no new logic is introduced during logic synthesis or place and route. This process also ensures that the aggressive optimizations available during implementation are appropriate for the design and have not altered it in some unexpected way. Verifying post-synthesis and post-route netlists against the input RTL specification proven correct ensures trust in every stage of the core and SoC development process.

VII. Real-World Results

The verification methodology described has been deployed at multiple sites and run on multiple RISC-V processor core implementations. Results for two cores available from open-source repositories can be shared publicly. The first is RI5CY, a 32-bit implementation with a four-stage in-order pipeline [6]. It supports the "IMFC" instruction sets plus customer instruction-set extensions intended for signal processing. The RISC-V verification methodology has identified 13 issues as of the date of this paper. All of these have been reported back to the RI5CY developers, who have confirmed four issues as bugs and fixed three. The remainder are still under investigation. The issues found include:

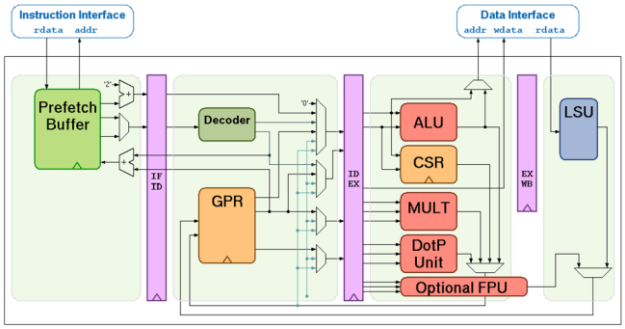

Figure 5. RI5CY block diagram

- Incorrect value written to a CSR
- Two violations of exception handling rules
- Six cases where exceptions should have raised but were now
- Incorrect floating-point dynamic rounding mode
- Wrong result from floating-point calculation
- Trap return handling violation
- CSRs updated while in debug mode

The second design verified by the methodology is Rocket Core, a 64-bit implementation of RISC-V with a five-stage, single-issue, in-order pipeline [7]. It has a sophisticated microarchitecture with branch prediction, instruction replay, and out-of-order completion for long-latency instructions such as division. Five issues were found and reported to the Rocket Core developers:

- Jump instructions store different return program counter (PC)
- Illegal opcodes are replayed (generating memory accesses)
- DIV (divide) result not written to register file
- Return from debug mode is executable outside of debug mode
- Core contains undocumented non-standard instruction

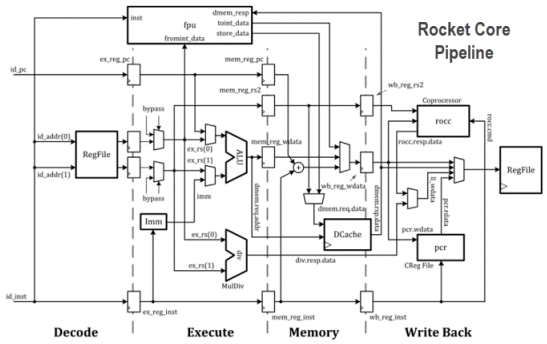

Figure 6. Rock Core block diagram

The first issue was not deemed a bug since the instruction fetch unit is responsible to prevent it from happening, and the second is still under investigation. The remaining three issues have been confirmed as bugs and fixed by the developers. The final bug is especially interesting. The formal engines reported that the core contained an unexpected instruction, which could very well have been a hardware Trojan. It turned out that the development team had added a new instruction but had not documented it yet in the specification that the verification team used to develop Operational Assertions for the custom instructions. Any integration team that downloaded the same version of the core would have had unknown and undocumented functionality in the design, and only GapFreeVerification could have detected and reported this.

VIII. ACKNOWLEDGEMENTS

Verification of both cores was performed using the methodology and tools provided by OneSpin Solutions, which includes the DV-Verify formal platform, the Floating-Point Unit App, the RISC-V Verification App, and the EC-ASIC and EC-FPGA equivalence checkers [8].

IX. CONCLUSION

RISC-V is transforming the processor and SoC industry, but it presents verification challenges for both core providers and SoC integrators. Verification must extend beyond ISA compliance to cover microarchitecture, customizations, security, and trust. This paper has described a methodology that spans this full range. Its value has been shown with the results of verifying two popular open-source RISC-V core. Both have been verified and taped out multiple times, yet the methodology found lingering design bugs verified by the core developers. There is no need for core providers or core integrators to find bugs in silicon. A third-party solution usable by both parties, enabling RISC-V designs that meet the highest standards for functionality, security, and trust, is available today.

REFERENCES

[1] www.risc-v.org.

[2] The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 20190608-Base-Ratified.

[3] The RISC-V Instruction Set Manual, Volume II: Privileged Architecture, Doc-ument Version 20190608-Priv-MSU-Ratified.

[4] R. Baranowski and M. Trunzer, “Complete formal verification of a family of automotive DSPs”, DVCon Europe, 2016.

[5] J. Bormann, S. Beyer, A. Maggiore, M. Siegel, S. Skalberg, T. Blackmore, and F. Bruno, “Complete formal verification of TriCore2 and other processors”, DVCon, 2007.

[6] github.com/pulp-platform/riscv

[7] github.com/chipsalliance/rocket-chip

[8] www.onespin.com/products/360-dv-verify/