Risc-V design requirements

To increase understanding of how verification and the associated tools and techniques should be applied to achieve complete verification, **Rob van Blommestein** offers some advice

It is safe to say that Risc-V is enjoying a surge in popularity among the design community. Not only does the open source nature make it a cost-effective alternative to Cisc, but the instruction set architecture (ISA) is designed for flexibility.

It can map to many implementations and micro-architectures and contains numerous optional instructions and features. It also allows for the development of custom instructions and features, and supports a wide range of end applications.

However, with this flexibility come verification challenges and questions about how best to tackle them.

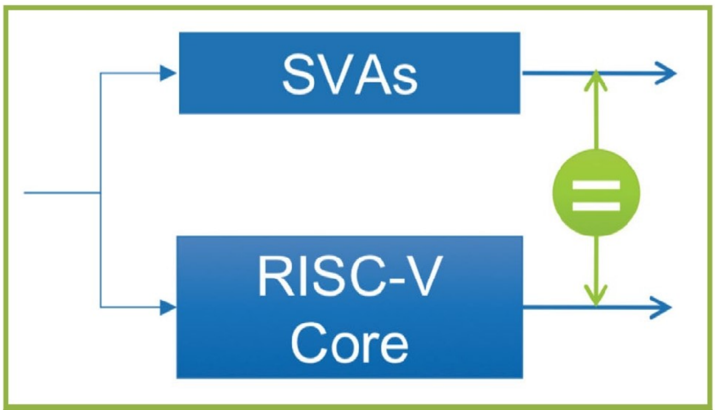

Figure 1: One method of property checking is to use reusable SystemVerilog assertions (SVAs) to achieve an unbounded proof

A question of trust

There are a number of requirements that must be met when developing and using Risc-V cores. One of the biggest is that of trust in the IP implementation.

Have all the features been implemented?

Have they been implemented correctly?

Are there any bugs that are hiding in the implementation?

Are there any Trojans inside this IP?

Simulation has had a difficult time answering these questions, and this has led to an increasing use of formal verification (Formal) to answer these questions. The exhaustive nature of Formal makes it an ideal choice to assure confidence in the core.

To provide a deeper understanding of how to effectively verify Risc-V, we asked the community for their specific questions and have provided some answers.

Is there a special Formal technique for Risc-V only?

There isn't anything specific to Risc-V. However, you need to have a fairly detailed understanding of the Risc-V ISA in order to create the correct set of properties and to make sure the property set is complete. A Formal technique that should be used to create the properties is interval property checking. These are reusable SystemVerilog assertions (SVAs) to achieve an unbounded proof (*Figure 1*).

There are several important aspects of these assertions. They do not start from reset but from a generic valid state. There is a limited number of cycles to reach the generic valid state. It is crucial to de-couple the ISA specification requirements from the micro-architectural details.

These operational SystemVerilog assertions enable high level, non overlapping assertions that capture end to end transactions and requirements in a concise, elegant way.

They capture functional requirements in a Formal and simulation executable format and capture entire circuit transactions in a concise and elegant way, similar to timing diagrams. They also achieve 100% functional coverage with high level, easy-to-review assertions.

Other advantages are that they adopt a consistent assertion style that is applicable to a wide range of applications and able to deliver optimal performances for both simulators and Formal tools.

Finally, they cleanly separate implementation-specific supporting verification code from reusable specification-level code.

Many properties will need to be constructed to fully verify the Risc-V design. This could take many months to complete and will require detailed knowledge of the Formal tool in order to reach full convergence. There are solutions on the market that can dramatically reduce this time.

Since Risc-V is a huge design; how can it be verified and what are the initial assumptions?

It is difficult to answer what the initial assumptions should be as these should vary according to the specific design. However, when tackling the verification of a huge design like a Risc-V core, the best way is to 'divide and conquer'.

Focusing on the core should be the first step. This includes making sure the protocols are not violated, verifying the caches separately to isolate associated complexities, looking at the interconnects between different blocks and units and performing x-propagation.

It's important to note that x-propagation should be done at the block and unit level and not on the entire core during this step. Another technique is to use "assume guarantee" to validate the interfaces.

It is worth mentioning that these methods are not specific to Risc-V and can be applied to any processor verification. However, the complexities of Risc-V may mean that it takes many months to perform a complete and exhaustive verification of the core.

There are solutions available, designed with these complexities in mind, to generate reusable assertion IP and achieve full verification in a fraction of the time.

What are the benefits of using formal verification instead of simulation for companies wanting to certify a Risc-V implementation against safety-critical standards?

One of the main benefits that Formal has over simulation is the exhaustiveness that it provides. This does not just apply to Risc-V, but to any processor.

What is key is that as complexity increases, there is a point where simulation is no longer enough. For today's designs, that complexity threshold is rather low. As it relates to Risc-V, exhaustive verification can be achieved with the correct knowledge of the ISA, the complete property set, and the right formal engines.

Formal proofs are much more powerful than simply passing a traditional compliance test suite running in simulation. Simulation, no matter how extensive, can exercise only an infinitesimal portion of possible design behaviour. Even the most carefully crafted compliance suite leaves gaps in design coverage. There are countless cases where specific operand values or corner-case conditions are untested, and some of these may trigger hidden design bugs.

Arithmetic operations are especially subject to this problem. A single mis-typed array index in an RTL description can yield a design where unexpected and non-intuitive operands produce the wrong answer.

The very nature of simulation, as well as acceleration and emulation, makes it impossible to try every possible case. Formal tools do not iterate through test cases; they mathematically analyse a design in its entirety.

When it comes to safety, knowing if vulnerabilities can be avoided, what and where any and all bugs exist, and if all paths are correct and are vital. Simulation has a difficult time providing a complete picture. There is a point of diminishing returns with simulation runs, since continued runs fail to find all the answers. Formal is the only technology that can conclusively prove the absence of something (such as a bug).

There are safety standards that require the use of specific verification techniques, as well as a well-defined, thorough verification process. That means verification has to be much more rigorous. If safety is a concern, then simulation alone is not sufficient and Formal must be a part of the verification plan.

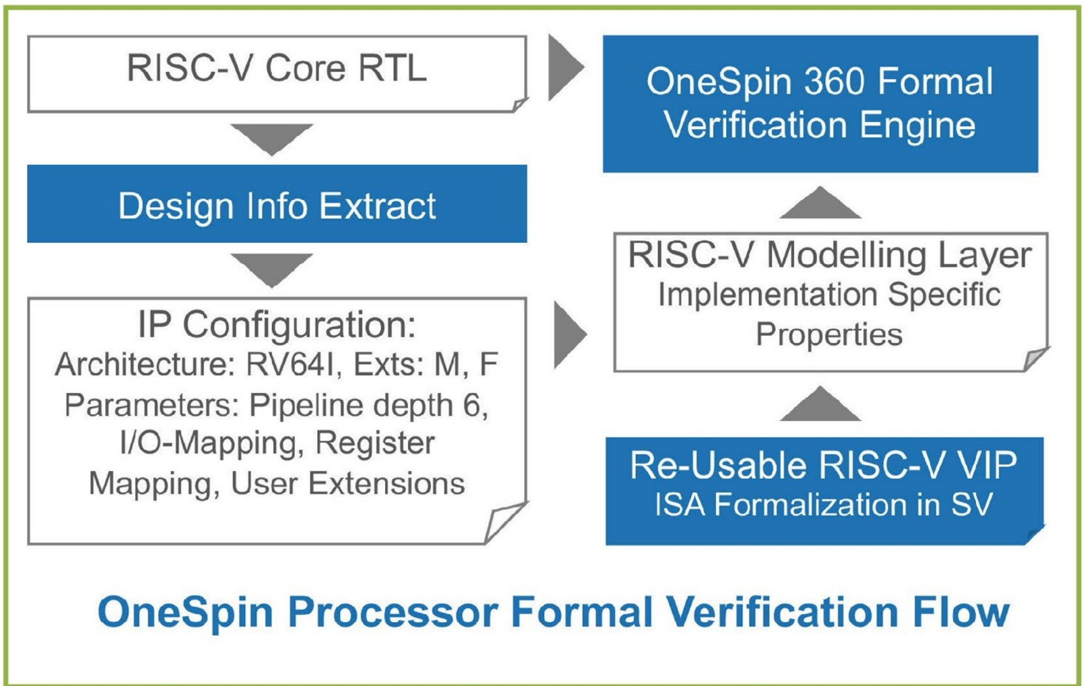
RISC-V Core RTL
Design Info Extract
IP Configuration: Architecture: RV64I, Exts: M, F Parameters: Pipeline depth 6, I/O-Mapping, Register Mapping, User Extensions
OneSpin 360 Formal Verification Engine
RISC-V Modelling Layer Implementation Specific Properties
Re-Usable RISC-V VIP ISA Formalization in SV
OneSpin Processor Formal Verification Flow

Figure 2: The OneSpin processor verification flow

I've heard that the Risc-V core is written in Chisel. How does Formal deal with that?
There are a few open source cores, like those from Berkeley, that are written in Chisel. However, most Formal solutions do not directly support this language.

There are, however, many methods that can easily convert Chisel to Verilog, which is widely supported by Formal tools on the market. There are, of course, many Risc-V cores that are directly written in Verilog so converting is not an issue for those cores.

Do I need to verify that the Risc-V core received from the provider is bug-free?
The short answer is yes. The core provider should run a complete integrity verification of the core and provide documentation of the results. For confidence in the integrity of the core, Formal verification should be a part of their verification flow. Just because the core has been formally verified though, does not mean that its integration into the design will be clean and free of issues. All verification steps should be re-run when introducing the Risc-V core into the final design.

> "Understanding how to overcome verification challenges is critical for Risc-V designers

How do I verify that a Risc-V core is integrated properly into the design?
Using a Formal solution geared toward the complexities of Risc-V is recommended. The process must include automated code inspection to rapidly eliminate many classes of common coding and design errors, before functional verification and logic synthesis.

Verification of the RTL should be done from three different perspectives. The first is structural analysis – focused syntactic and semantic analysis of source code.

The second is safety checks, exhaustive verification of the absence of common sequential design operation issues. The third perspective is activation checks – proof that specific design functions can be executed and are not blocked due to unreachable code.

Interval property checking should also be used as in the response to the question "Is there a special Formal technique for Risc-V only?"

I have added a custom instruction. Do I need to re-verify the entire Risc-V core?
When anything is done to modify the custom instruction, it means functionality has been added or changed.

To ensure that the design operates as intended and does not do anything unintended, a full re-verification is strongly recommended.

I need a bug-free Risc-V core. What vendors or open source core should I use?
Look for a vendor that provides open source cores that have been exhaustively verified using Formal methods. This will dramatically reduce the chances of bug escapes into the final design.

Be sure to ask for documentation of the vendor's verification process so that you are sure that Formal has been used. A good place to start for formally verified open-source cores is to check with the OpenHW Group (www.openhwgroup.org). The core development includes a verification plan with Formal in mind.

Risc-V offers the benefits of flexibility and customisation in addition to reducing the processor cost barrier. However, the benefits increase the complexity and therefore may complicate the verification process.

Understanding the verification challenges and how to overcome them is critical for any designer interested in Risc-V. ❏

About the author
Rob van Blommestein is vice-president of marketing at OneSpin