The open RISC-V processor architecture is shaking up both worlds – that of intellectual property (IP) and system-on-chip (SoC). There's much industry activity and interest, with RISC-V cores already finding their way in new designs. However, successful RISC-V core providers must verify all aspects of integrity for those designs, including functional correctness, safety, security and trust. Core-level verification steps may need to be re-run, plus some additional tasks performed, to ensure that the cores have been integrated properly. Verification of RISC-V designs is especially challenging because of optional features, implementation flexibility and provisions for customer extensions.

Verifying open-source cores
Two industry associations have formed expressly to support RISC-V evolution and deployment. Many RISC-V cores and some of the SoCs built around them are available as open-source, with some of these already commercially available with software titles ported into designs. It might seem that nothing can stop the ascension of this new Instruction Set Architecture (ISA), but no evolution happens without challenges. For RISC-V, one of the biggest questions is how to verify the integrity of the cores and the SoCs that contain them. Open source may mean easy access to cores, but how can their implementation be confirmed as correct and acceptable for designs?

The attraction of the new architecture is clear: anyone can design anything based on RISC-V with no license fees or royalty payments, since the ISA is open and available to all. Further, implementations are available from many sources, enabling comparative evaluations by SoC teams and establishing a stronger position for negotiation if choosing a commercial solution. This is in sharp contrast to most established processor architectures, available only from a single vendor that controls the ISA and charges for its licensing. On the other hand, established processor families offer huge ecosystems and years of proven silicon. SoC developers give little thought to re-verifying a licensed core, trusting the vendor to perform thorough verification of functional correctness, and other aspects of integrity.

To build a viable business, RISC-V core vendors must compete successfully with the proven established options. Open-source implementations must be vetted so that SoC developers are comfortable integrating them into designs, which comes back to verification. There is no one central team designing and verifying RISC-V cores. For the RISC-V ecosystem to thrive, core suppliers need an independent verification solution to ensure that their designs are compatible with one another, free from bugs and compliant with the ISA specification. Similarly, SoC developers must ensure that the cores they license are fully verified and ISA compliant. For applications with safety, security and trust requirements, other aspects of design integrity are also critical.
such as ensuring the cores do what they are supposed to but also don’t do what they are not supposed to.

One of the biggest challenges of verifying RISC-V cores is the many options allowed by this highly configurable ISA; user extensions are allowed into the architecture, including custom instructions. Any verification solution must be flexible enough to expand to verify such extensions whilst also checking that no extension has violated any aspect of the baseline ISA. In addition, verification must include the core’s microarchitecture. RISC-V was developed to map onto a wide range of implementations, including different chip technologies, a wide range of PPA tradeoffs, deep processor pipelines and out-of-order execution of instructions. The verification approach must be general enough to span all these variations; see Figure 1.

**Good design hygiene**

Verification begins with automatic code inspection applicable to the RISC-V cores, the SoCs that contain them and any register transfer level (RTL) design code. This process eliminates many classes of common coding and design errors prior to functional verification and logic synthesis, but may take some time. Using the right automated solution can speed up this code inspection process and debug, making it easy to fix any violations discovered. Ideally, three perspectives should be considered:

- **Structural analysis** that focuses on syntactic and semantic analysis of source code;
- **Safety checks** to perform exhaustive verification of the absence of common sequential design operation issues;
- **Activation checks** to prove that specific design functions can be executed and are not blocked due to unreachable code.

**Functional correctness**

When it comes to verification, simulation is often the first thing that comes to mind. However, 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 behaviours. 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, which could trigger hidden design bugs. Arithmetic operations are especially prone to this problem. A single mistyped array index in an RTL description can yield a design where unexpected and non-intuitive operands produce wrong output. 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. A formal methodology that uses operational SystemVerilog Assertions (SVA) enables high-level, non-overlapping assertions that capture end-to-end transactions and requirements in a concise way:

- Capture functional requirements in a formal and simulation executable format;
- Capture entire circuit transactions in a concise way, similar to timing diagrams;
- Achieve 100% functional coverage with high-level, easy-to-review assertions;
- Adopt a consistent assertion style that’s applicable to a wide range of applications and can deliver optimal performances for both simulators and formal tools;
- Cleanly separate implementation-specific supporting verification code from reusable specification-level code.

Each instruction in the RISC-V ISA is captured in a single Operational Assertion. These assertions capture the high-level operational view, and map to sequential or pipelined implementation, out-of-order execution and other possible options in the RTL core. A solution like the RISC-V Verification App from OneSpin includes privileged ISA, Control and Status Registers (CSRs), an exception mechanism and other extensions. Its verification framework splits the specification side from mapping to implementation, to enable full SVA re-use. Regardless of the approach used, it should be flexible enough to ensure that nothing is broken if custom extensions are added, and powerful enough to verify the new functionality.

The Operational Assertions can be used for automatic detection of specification omissions and errors, holes in the verification plan and unverified RTL functions. Verification must be performed to confirm that the set of assertions is sufficient to cover the RISC-V core design, and to ensure there’s no unverified RTL functionality.
Verifying safety, security and trust

Some RISC-V cores are used in safety-critical applications with industry standards that require functional safety analysis and higher diagnostic coverage to meet certification requirements. The automotive market, governed by the ISO 26262 standard, demands a particularly rigorous development methodology. This requires using specific verification techniques and a well-defined, thorough verification process. What is needed is an automated functional safety analysis that enables higher diagnostic coverage. Whilst these requirements may not apply to a RISC-V processor core, they will arise at the SoC level for relevant application domains and DO-254 standards.

Some applications for RISC-V have rigid security requirements to prevent malicious agents exploiting vulnerabilities in the design. Effective hardware security to guard against adversary attacks requires innovation beyond traditional functional verification, which focuses on target use cases. Bugs that break these use cases are fixed, but bugs exposed in artificial use cases are often dismissed. Hardware bugs with software workaround may also remain in the design. These shortcuts are unacceptable when security is a concern. Security verification must include a rigorous process to detect all bugs and flaws, establishing precisely what can and can’t happen in the design.

The development process for cores and SoCs is complex, often involving multiple teams in multiple locations, with a mix of reused design components and third-party IP (3PIP) blocks such as RISC-V cores. There may be multiple opportunities for introduction of hardware Trojans or other malicious agents. The original specification might include issues that could lead to design errors or unintended and exploitable features. Problems may creep in during RTL synthesis, either by deliberate action or via the tools. Integration of 3PPs 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. FPGA designs are especially vulnerable because of the extensive optimisations performed during synthesis/P&R and because changes may be made when generating the bitstream to program the device.

A trusted design flow must ensure that no hardware Trojans are inserted at any point; see Figure 2. Many applications, from autonomous vehicles and financial data servers to defence/aerospace and nuclear power plants, must be protected from Trojans since the consequences of adversary attacks could be severe.

Great potential

RISC-V has great potential in the industry, but it must meet significant verification challenges to succeed against entrenched processor competition. This means verifying functional correctness, ensuring proper operation in safety-critical applications, checking that designs are secure and trusted, free of unintentional or deliberate vulnerabilities, and establishing proof of compliance to the ISA. SoC teams evaluating RISC-V cores should demand that their providers run complete integrity verification and document the results. This process must include automated code inspection, RISC-V verification and any other applicable formal method to prove functional correctness. Verification of safety, security and trust must also be performed when the requirements of the end application demand it. SoC teams should be able to re-run all these verification steps themselves as part of their acceptance criteria for RISC-V cores.