ExaVerif

Exhaustive Verification for RISC-V Custom Instructions

SSCCS Foundation

2026-05-30

ExaVerif performs exhaustive verification of RISC-V custom instruction extensions. A YAML file describing a module's constraints is all that is required — ExaVerif generates every valid constraint combination, evaluates each one deterministically, and produces an auditable report with optional LLM-powered interpretation. The CLI is open-source and free to use. ExaVerif is an early-stage project: the core engine has been internally validated via x86-64 CI, and RISC-V target integration is in progress.

Code	References	References	References
Github	neXus	SSCCS Project	OpenHW

References	Other Formats
Space Computing	HTML

The Problem

RISC-V's defining advantage is its extensibility. Custom instruction extensions (XIF) enable domain-specific acceleration for AI inference, signal processing, cryptography, and embedded control. Every custom instruction must be verified across its full operational envelope — operand ranges, pipeline states, data alignment, hazard conditions, and physical constraints.

The current verification tooling faces two persistent obstacles:

1. **No automated constraint composition.** Random instruction generators produce statistical coverage but cannot enumerate the complete space of interacting constraints. When operand ranges, pipeline depth, and alignment rules combine, the number of valid configurations reaches into the thousands. Manual test writing cannot scale to cover this space exhaustively.
2. **Non-deterministic coverage.** Random-seed-based simulation cannot determinically prove that a specific edge case was tested. Reproducing a failure requires preserving the exact seed, environment state, and tool versions — a fragile chain that breaks across teams and over time.

What ExaVerif Does

ExaVerif treats a YAML description as a **Spec Space**: field domains define its axes, constraints carve admissible subspaces, and each combination is a point within this space. A single command enumerates and evaluates every point exhaustively.

```
ev verify --target cva6_xif_ref.xif.yaml
```

Internally, ExaVerif expands the full cartesian product of all field domains, then evaluates each combination against every declared constraint. The result is exact: every passing combination is a valid instruction encoding; every failing combination has a specific, recorded reason.

```
target: cva6_xif_ref
total:  262144
passed: 3072
failed: 259072

Failures:
  [FAIL] [0, 1, 0, 0, 0] - funct3 → funct7: {funct3=0 → [0]; funct3=1 → [0, 1, 2, 3, 32]}
  [FAIL] [2, 0, 0, 0, 0] - funct3   {0, 1}
```

The example above shows ev verifying the CVA6 CV-X-IF reference coprocessor against its actual encoding specification. The coprocessor accepts only two funct3 values with specific funct7 subfields, resulting in 3,072 valid encodings out of 262,144 possible. The remaining 259,072 are correctly identified as illegal — not by random sampling, but by exhaustive enumeration.

```
ev verify --target cva6_xif_ref.xif.yaml --json
```

Every result is also available as structured JSON for downstream consumption by Nexus — each combination becomes a Fact, each constraint a Hint, and the verification run itself an Intent.

```
{
  "target": "cva6_xif_ref",
  "total": 262144,
  "passed": 3072,
  "failed": 259072,
  "results": [...]
}
```

The Field Composition engine — a branchless, constant-time verification kernel validated via x86-64 CI — guarantees deterministic results. The same input always produces the same output, independent of random seeds or simulator state.

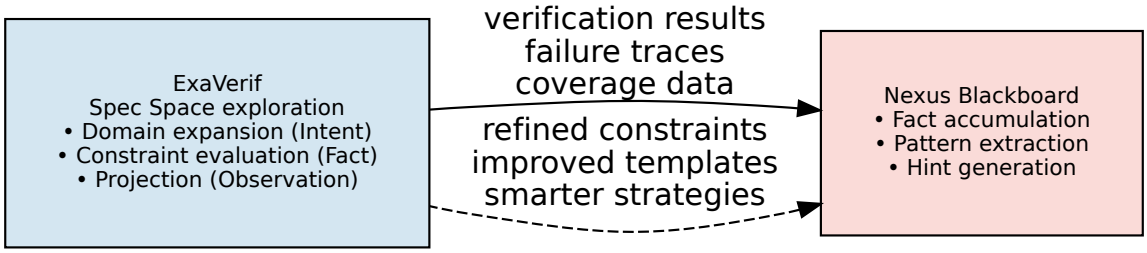

Figure 1: ExaVerif verification results flow into Nexus's knowledge base; the same FIH cycle operates inside ExaVerif itself.

ExaVerif's verification results are consumed by Nexus, the autonomous research infrastructure. The same FIH cycle operates inside ExaVerif itself at a different scale — domain expansion as Intent, each evaluated combination as a Fact, each constraint as a Hint — making ExaVerif a Nexus instance in its own right. Every constraint combination evaluated, every failure pattern discovered, and every coverage gap identified flows into Nexus's knowledge base, where it becomes training material for the agentic research loop and reusable knowledge for the entire RISC-V ecosystem.

How the Industry Verifies Custom Instructions

Before examining why ExaVerif is different, it is worth understanding how verification teams approach custom instruction verification today. Each approach carries its own strengths, and each leaves specific gaps unfilled.

Scenario-Based Integration Testing

Tools like Breker generate test scenarios using SystemVerilog, UVM, and PSS. Their modular FASTApps framework can target custom extensions as reusable verification components. These tools excel at complex SoC-level integration testing where real-world system behavior must be simulated.

The burden falls on setup. UVM environments are expensive to build and maintain. PSS requires specialized knowledge that many RISC-V teams lack. The randomness at the core of scenario generation means that deterministic edge cases can slip through — a particular operand combination that triggers a pipeline hazard might never be generated, and no log will report its absence.

Reference Model Comparison

ImperasDV takes a different approach: it runs the RTL design and a golden reference model in lockstep, comparing results cycle by cycle. Custom instruction support is a recognized strength — the reference model can be modified to match the extension being verified.

The limitation lies in the reference model itself. Updating it to match a new custom instruction takes time, and the verification remains simulation-bound. No matter how many cycles are compared,

exhaustive coverage of the constraint space is not the goal — correctness relative to the model is, and the model may contain blind spots of its own.

Formal Verification

OneSpin (Siemens EDA) applies mathematical proof techniques to achieve zero bug escapes. SystemVerilog Assertions (SVA) define properties that the design must satisfy, and the tool proves or disproves them exhaustively.

This approach catches bugs that simulation never would. The trade-off is complexity. Setup demands deep formal expertise. Large designs hit capacity limits where proofs become computationally infeasible. The scope is typically narrow — specific properties are verified, not the full constraint space of a custom instruction.

Random and Automated Test Generation

RiescueC and RISCOF represent the open-source end of the spectrum. A JSON configuration describes the extension, and random assembly test sequences are generated automatically. RISCOF compares results across simulators to detect discrepancies.

The barrier to entry is low, which makes these tools popular for ISA compliance testing. The shortcoming is the same randomness that makes them accessible: coverage is statistical, not deterministic. A test that passes today may fail tomorrow with a different seed. A corner case that matters may never be randomly selected.

The Pattern Across All Approaches

Every existing approach — whether commercial or open-source — shares one trait: **coverage is either statistical or scoped to specific properties.** None generates every valid constraint combination and evaluates each deterministically. This is the gap ExaVerif exists to fill.

Why ExaVerif Is Different

	Existing Tools	ExaVerif
Coverage	Random sampling, manual test writing	Exhaustive constraint combination
Evaluation	Seed-dependent, simulator-dependent	Deterministic
Setup	UVM/SystemVerilog environment, commercial simulators	Single YAML file, single binary
Interpretation	Manual log analysis	LLM-powered natural-language explanation

	Existing Tools	ExaVerif
Result reuse	One-time reports	All results accumulated as reusable knowledge

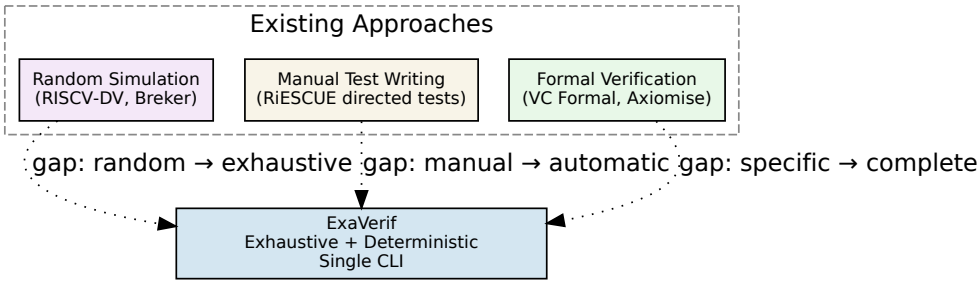

Figure 2: ExaVerif fills the unfilled quadrant: exhaustive coverage combined with zero-setup CLI accessibility.

Automated Issue Management

A verification result that is filed away after review becomes stale the moment the codebase changes. ExaVerif treats every result as a permanent, queryable record. When a constraint combination fails, the failure is not merely logged — it becomes part of a structured knowledge store that other projects can query without human intervention.

Existing tools produce reports that must be manually compared, triaged, and tracked across projects. ExaVerif records every pass and every failure alongside the exact constraint combination that produced it. When a developer runs `ev check` against a module, ExaVerif consults what the collective has already discovered about similar constraint patterns. A failure that was analysed once never needs to be diagnosed from scratch again.

This shifts verification from a series of disconnected runs to a compounding knowledge loop. A team validating a new custom instruction benefits from every failure pattern that every other team has already encountered, resolved, and deposited. The tool does not merely report issues — it remembers them, and that memory grows with every use.

Acknowledging the Limits of Exhaustive Verification

Exhaustive verification carries an inherent constraint: the number of combinations grows exponentially with the number of parameters. A simple integer XIF with two operands and a few constraints may produce thousands of combinations. A complex vector accelerator with pipeline interactions, multiple data types, and physical constraints may produce billions — far beyond practical evaluation time.

ExaVerif does not solve the combinatorial explosion problem. No tool can. What it provides is a structured approach to constraint composition that makes the scope of verification explicit. Rather

than relying on random sampling and hoping coverage is sufficient, ExaVerif shows exactly which combinations were evaluated and which were deferred.

For small to medium-sized modules — single-cycle integer XIFs, basic cryptographic accelerators, simple state machines — exhaustive coverage is achievable. For larger designs, ExaVerif serves as a complement to existing tools, systematically covering the constraint space that random approaches might miss. The scope is narrow, and that is by design.

Competitive Landscape

The RISC-V verification market contains several established approaches, each with a distinct gap that ExaVerif fills.

Category	Tool / Vendor	Approach	Limitation
Random simulation	RISCV-DV (Google)	Constrained-random instruction generation	Statistical coverage; misses deterministic edge cases
Random simulation	Breker SystemVIP	AI-assisted test synthesis	Random seed-based; commercial license required
Formal verification	OneSpin (Siemens EDA)	Mathematical property proving	Complex setup; verifies specific properties only
Formal verification	Axiomise (formalISA)	Formal property verification	Requires deep formal expertise to operate
Simulation DV	ImperasDV (Synopsys)	Simulation-based with RVVI standard	Requires commercial simulator; manual test writing
Compliance	RISC-V ACT	ISA compliance test suite	Limited to architectural conformance; no custom extension coverage
Cloud platform	Vyoma UpTickPro	Cloud-based integrated verification	Verification still random/manual at core
AI debugging	Vtool Cogita-PRO	AI-assisted debug	Post-silicon debug; does not prevent verification gaps

What no existing tool does: exhaustive constraint composition. Every tool either samples statistically, verifies only specific properties, or requires manual test authoring. ExaVerif generates every valid constraint combination from a single YAML description and evaluates each one deterministically.

Collaboration with the Ecosystem

ExaVerif is being developed as an independent product within the RISC-V ecosystem. It is not tied to any single organization, core, or workflow. The project welcomes collaboration with groups working on RISC-V verification — including OpenHW, PULP, and individual core developers — to validate the approach against real-world custom instruction extensions.

Collaboration is structured as a mutual exchange: partners provide domain expertise and access to real-world XIF designs for validation; the ExaVerif project provides the tool, methodology, and benchmarking infrastructure. All verification results generated through collaboration are deposited in the shared knowledge store, where they become reusable knowledge for the entire ecosystem.

Business Model

ExaVerif is an early-stage project. Our initial focus is not monetization but establishing the technical foundation that makes RISC-V verification more accessible. The CLI is open-source (Apache 2.0) and free to use. Paid tiers — if and when ecosystem validation confirms demand — will provide advanced interpretation and collaboration features. Certification services, if developed, would be offered as a separate layer built on top of the core engine.

Current State

The core constraint composition engine has been internally validated via x86-64 CI. RISC-V target integration (Spike) is in progress. No paid features exist. No certification service is operational.

Public Assets

These are published openly to build community trust and invite collaboration.

Asset	Status
ExaVerif CLI (`ev`)	Under development (Apache 2.0)
Technical concept document	Published
Shared knowledge store	In development

Future Monetization (Post-Validation)

Paid tiers are designed but **activated only after ecosystem partners provide positive validation**. No pricing has been set. Any future paid services — advanced LLM interpretation, custom constraint templates, CI/CD integration, formal certification — would be built on top of the free core engine, not required for basic use.

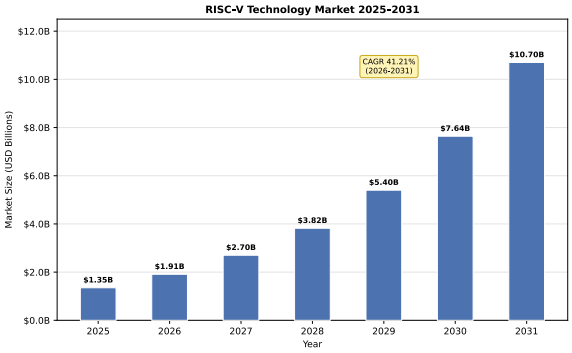

Figure 3: RISC-V Technology Market Size Forecast (2025–2031)

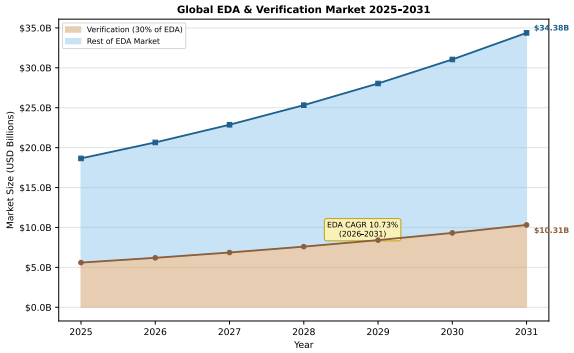

Figure 4: Global EDA and Verification Market Size Forecast (2025–2031)

Market Context

The RISC-V technology market is projected to grow from **$1.35B in 2025** to **$10.7B by 2031** at a **41.2% CAGR**. Verification accounts for roughly 30% of the broader EDA market, which reached **$19.22B in 2025**.

These figures describe the total addressable market. The serviceable market for ExaVerif — teams actively designing custom RISC-V extensions who would adopt a new verification methodology — is a narrow slice of this total. ExaVerif's near-term opportunity lies not in displacing established tools but in complementing them for specific use cases where exhaustive coverage adds demonstrable value.

Related Research: Radiation-Tolerant Computing

A parallel research track explores how the same XIF verification methodology extends to radiation-tolerant RISC-V computing for space environments. Custom instructions verified by ExaVerif at design time can carry fault-tolerance policies into deployment. This work is documented separately in the RISC-V Space Computing report.

What Makes This Possible Only on RISC-V

Dimension	Proprietary ISA (x86 / ARM)	Open ISA (RISC-V)
Simulators	Vendor-proprietary (Intel SDE, ARM Fast Models)	Fully open source (Spike, QEMU, Renode)
Extensibility	Vendor-permitted instructions only	Anyone can design custom instructions via XIF
Verification transparency	Black-box; internal tooling required	Anyone can inspect algorithms and results

Dimension	Proprietary ISA (x86 / ARM)	Open ISA (RISC-V)
Certification	Vendor-issued proprietary certs only	Third-party independent certification possible

Only RISC-V's open infrastructure enables a third-party certification authority to exist. ExaVerif is the engine that makes that certification meaningful.

Roadmap

Phase 0: Foundation (Present)

Goal: Validate that the problem ExaVerif solves is real, and that the solution is technically credible.

- Complete RISC-V Spike integration and publish benchmarks against a public CORE-V module
- Interview 10 RISC-V design teams to confirm demand for exhaustive constraint coverage
- Submit a collaboration proposal to the OpenHW Verification Task Group
- **Output:** Published benchmarks, a list of confirmed pain points, and at least one ecosystem partner engaged.

Phase 1: Working CLI + Reference Implementation (2026)

Goal: Release a functional open-source CLI with a publicly verifiable reference implementation.

- Release `ev check` as open source (Apache 2.0)
- Publish benchmarks against at least two public RISC-V cores
- Complete integration of the shared knowledge store for result persistence
- **Output:** A single binary that performs exhaustive verification with published, reproducible benchmarks.

Phase 2: Ecosystem Validation (2027)

Goal: Determine whether the tool solves a problem that teams will pay for.

- Secure at least two independent ecosystem partners actively using ExaVerif in their workflow
- If validation is positive, develop paid tiers (advanced interpretation, custom templates, CI/CD integration)
- Propose ExaVerif as a standard verification module within OpenHW's CORE-V-VERIF framework
- **Output:** Confirmed product-market fit or a decision to pivot.

Future Direction: Certification Services

If Phase 2 confirms demand, a formal certification layer may be developed as a separate service built on top of the core engine. This would involve:

- C2PA-compatible manifest generation and cryptographic signing
- Regulation-specific compliance templates (ISO 26262, DO-178C, ESA ESCC)
- On-chain verification proofs for auditability

These services are contingent on ecosystem validation. They are not part of the current roadmap. The analogy of "Let's Encrypt for RISC-V" describes a long-term vision, not a near-term deliverable.

Connection to the Substrate

ExaVerif is a neXus instance — a self-contained agentic neXus for the silicon verification domain. It does not merely "consume" neXus; it implements neXus internally. Its Spec Space exploration follows the same FIH cycle that governs every neXus entity: domain expansion is an Intent, each evaluated combination a Fact, each constraint a Hint. The Fact graph it builds over thousands of runs is its own independent, accumulated knowledge base — not a log, but a permanent, queryable, auditable body of verified knowledge.

At the same time, ExaVerif's Facts are consumed by other neXus instances. A gap in coverage detected here becomes an Intent for OpenROAD or Yosys on the same blackboard. A verified constraint combination becomes a Hint that guides hardware synthesis elsewhere. This is not data exchange between tools; it is a single FIH graph to which every instance contributes and from which every instance reads.

This dual nature — self-contained neXus internally, co-equal peer on a larger neXus externally — is the paradigm shift the industry has not yet recognized. Existing verification tools produce PASS/FAIL logs that are discarded at the end of a run. The result has no life beyond the terminal. ExaVerif, as a neXus instance, produces Facts that are immutable, content-addressed, and permanently embedded in a graph that spans every domain of the project. A verification result is not a log entry; it is a permanent contribution to a shared body of knowledge that compounds across runs, across tools, and across time.

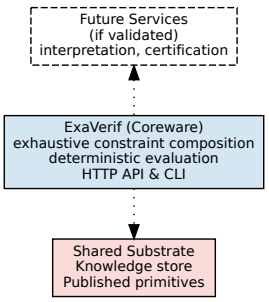

Figure 5: ExaVerif is the verification engine; future services would be layered on top of the shared substrate.

- Whitepaper: PDF / HTML DOI: 10.5281/zenodo.18759106 via CERN/Zenodo, indexed by OpenAIRE. Licensed under *CC BY-NC-ND 4.0*.
- Official repository: GitHub. Authenticated via GPG: BCCB196BADF50C99. Licensed under *Apache 2.0*.
- Governed by the Foundational Charter and Statute of the SSCCS Foundation (in formation).
- Provenance: Human-in-Command, AI-assisted. Aligns with ISO/IEC JTC 1/SC 42 and C2PA-certified. Full intellectual responsibility with author(s).