Spec-First Formal Verification of RISC-V Vector and Scalar Instructions

From Sail ISA Specification to RTL Equivalence

Yi-De Wu

ICCAD 2026

National Taiwan University

Two Problems Everyone Knows About

The Vector Gap

RVV 1.0 ratified 2021. Five years later:

- riscv-formal: "vector not included yet" (since 2018)
- chiRVFormal (SETTA'24): scalar only
- Commercial tools: none publicly known

Why? State explosion:

- VLEN=256 $\Rightarrow$ $\sim$16,456 bits of state
- Search space: $2^{16,456}$ — **intractable**
- Ara (VLEN=4096): $>$150,000 bits

The Trust Gap

Every existing tool relies on hand-written oracles:

- riscv-formal: hand-written RVFI properties *per core*
- chiRVFormal: hand-translates Sail $\rightarrow$ Chisel reference model

> *"Hand-written reference models reintroduce the very errors formal verification aims to eliminate."*
> — Gao & Melham, FMCAD'21

Meanwhile, **Isla** (CAV 2021) auto-extracts SMT from Sail — available but unused by these tools

As-Is: How Scalar Verification Works (and Why Vector Doesn't)

Scalar: ADD instruction

RTL (Verilog) → Yosys → SMT-LIB2 → Z3 ✓ **UNSAT**

Target	Instruction	State bits	Z3 time	Result
Scalar (RV64I)	`ADD`	422	milliseconds	✓ tractable
Vector (RVV)	`vadd.vv`, VLEN=256	16,456	>10 min	× timeout
Vector (Ara)	`vadd.vv`, VLEN=4096	$>150{,}000$	—	× impossible

Where do 16,456 bits come from?

- `vs2`, `vs1`, `vd`: $3 \times 256 = 768$ bits
- `v0` mask register: 256 bits
- `vl`, `vtype`, SEW/LMUL: $\sim$18 bits
- 32 parallel 8-bit adders: 256 bits
- Crossbar, mux, forwarding: $\sim$15,000 bits

The structural problem:

Yosys flattens all 32 lanes into one bitvector. Z3 cannot see that lanes are independent — it bit-blasts *all* 16,456 bits.

Lane independence is **destroyed** by RTL flatten.

Key Insight: Sail Gives You SMT for Free

Sail = RISC-V's official ISA spec language

- 30,000 lines, community-maintained
- Bitvector-typed DSL — every op maps to SMT
- Covers scalar AND vector instructions

Isla (CAV 2021): symbolic execution of Sail

- Input: instruction encoding (hex)
- Output: SMT-LIB2 bitvector formula
- Automatic — no hand-writing

TCB: only Isla + Z3
(not hand-written properties)

Concrete example:

```
$ isla-footprint -i "vadd.vv"
```

↓ produces per-element formula:

```
result[i] = bvadd(vs2[i],
vs1[i])
```

This is the **oracle**:

- Machine-checkable
- Auto-extracted from Sail
- Works for every instruction
- Same oracle reused across *all* cores

The Oracle Hub: L3 Fans Out to Every Layer

L7: Model correctness	
L6: Quantization error	
L5: Compiler IR (MLIR/LLVM)	39 proofs, all UNSAT
L4: Kernel correctness	13 kernels verified, 344 ms
L3: ISA Specification (Sail/Isla)	← **Oracle hub (43 traces)**
L2: RTL implementation	← **main contribution**
L1: Silicon fabrication	

Same oracle, multiple targets. Drop a `bridge.toml` → auto-verify → auto-dashboard.
L3 fans out: **downward** to RTL (L2), **upward** to kernels (L4) and compiler IR (L5).

To-Be: The Same `vadd.vv`, Our Way

Step 1: Isla auto-extracts per-element formula from Sail

`result[i] = bvadd(vs2[i], vs1[i])` for each 8-bit element

Step 2: Per-element decomposition: 16,456 bits $\rightarrow$ 32 $\times$ 24 bits (two 8-bit inputs + 8-bit output per lane)

Step 3: Z3 checks each 8-bit equivalence in $<$1 ms

Result: 32 elements $\times$ $<$1 ms = 18 ms total, UNSAT ✓

	As-Is (S3)	Our Method
Oracle	hand-written per core	auto from Sail
State bits	16,456	24 per element
Z3 time	$>$10 min (timeout)	18 ms
Vector support	$\times$	✓
Cores supported	1 (rewrite per core)	any (same oracle)

We Catch Real Bugs — Zero Hand-Written Properties

hack@DAC competition: 130+ bugs across 3 editions (2018, 2019, 2021).
9 bugs (7%) at ISA level — exactly our target.

Bug #14: Incomplete ALU case

- Missing case in CVA6 ALU
- Certain ops silently return `0x0`
- ISA spec says `0xFFFFFFFF`
- Z3: SAT in 5 ms
- Counterexample: `a=0xFFFF...`, `b=0x0`

Bug #3: CSR privilege escalation

- User-mode accesses M-mode CSRs
- Isla trace encodes privilege field
- 6 of 11 M-mode CSRs exploitable
- Z3: SAT in 2 ms per CSR
- Zero manual privilege modeling

CSR mutation injection (5 hack@DAC-inspired bugs):

Bug	Source	Mutation	Time	Result
Privilege bypass	hack@DAC'18 #3	CSRRS writes all-ones	<5 ms	SAT
Counter access	hack@DAC'18 #25	CSRRC clears all bits	<5 ms	SAT
Read/write mismatch	hack@DAC'19 #1	CSRRW writes ~rs1	<5 ms	SAT

Dashboard: Always Demo-Ready

Plug-and-play workflow:

1. Drop a `bridge.toml` for new core
2. `make verify-all` runs pipeline
3. `report.json` auto-generated
4. Dashboard auto-populates

Current coverage:

- 12 bridges: 9 cores + Vector + CSR + Branch
- 141 RTL proofs, all UNSAT
- 7 bugs detected
- 200+ total proofs (L2–L5)

Multi-backend redundancy:

Backend	Proofs	Time
Z3 (SMT)	141/141	8.3 s
EBMC (BMC)	40/40	1.65 s
rIC3 (BTOR2)	190/190	—

Cross-core differential:

9 cores pairwise consistency
$\Rightarrow$ all UNSAT

Open source:

`https://github.com/yiidtw/rvs`