Mallet: An Open-source Formal Verification Flow for Agile Domain-Specific Accelerator Development

Argonne NATIONAL LABORATORY

Raphael Huang
rhuang72@ucsc.edu

Kazutomo Yoshii
kazutomo@anl.gov

Why formal verification for accelerators?

- Scientific instruments need custom accelerators to placate extreme data rates
- **Precision** is critical for data accuracy, since fails are silent
- **Development speed** is similarly critical: data requirements shift fast
- We can address both at once with **formal verification (FV)**

Modern X-ray detectors are approaching data rates on the terabyte-per-second scale, and high-energy physics experiments, the petabytes-per-second range. Dynamic verification — driving inputs and checking outputs — is intuitive and therefore commonplace but lacks mathematical rigor; a testbench cannot find edge cases it didn't check. Formal verification fills this gap by feeding a property φ and a formal model $\mathcal{M}$ of the design to a SAT/SMT engine, which either proves or disproves with a counterexample whether $\mathcal{M} \vDash \varphi$ ($\mathcal{M}$ models φ).

What are current trends in AI for EDA?

- LLMs and **agentic EDA workflows → rapidly generated RTL**; more than ever before, **verification is the bottleneck**
- Inherent stochasticity of LLMs → risk of *common-mode failure*: $\mathcal{M}$ and φ derive from the same ambiguous NL description, and a bug in $\mathcal{M}$ isn't checked by φ
- **Solution**: FV with φ grounded independently of whatever process produced $\mathcal{M}$

How have FV tools responded?

Problem 1: Writing good formal properties/assertions is hard.
Solution 1: RTL I/O annotations automatically generate properties (e.g. AutoSVA).

Problem 2: Design-specification drift occurs, with the design leading the spec
Solution 2: Use hardware construction languages (e.g. Chisel, SpinalHDL [3]) that check coherence at compile-time.

The remaining problem: no tools currently solve both at once while being integrated with a fully open-source toolchain

We introduce: Mallet

An open-source formal verification workflow for the Chisel language stack with annotation-based property generation and design-spec drift protections.

	AutoSVA(2) [1], [2]	ChiselFV [4]	Mallet (this work)
Auto. property generation	✓	✗	✓
Spec/design coherence	✗	✓	✓
Unbounded liveness	✓	✗	✗
Fully open-source flow	✗	✗	✓

Open-source GitHub repository available at: https://github.com/rphlhuang/mallet

Methodology

```
class MacSpec(p: MacModuleParams) extends Axi4LiteMac(p) with MalletSpec
{
  // — memory-map roles ————————————————————————————————
  p.a_w            is WO
  p.b_w            is WO
  p.push_w         is Commit requiring (p.a_w, p.b_w)
  p.status_r       is Status setBy p.push_w
  p.result_r       is Result gatedBy p.status_r
  p.soft_reset_rw is WO
  // — protocol contract ———————————————————————————————
  S.AXI conformsTo AxiLite32Slave

  // — manual properties ———————————————————————————————
  private val srPulse = B(softResetPulseReg, "softResetPulseReg")
  property("soft_reset_is_pulse") { srPulse |=> !srPulse }
  property("result_not_dropped") { dut.io.out.fire |=> dutValidReg }

  done()
}
```

Figure 1: A code snippet with possible Mallet annotations for an AXI4-Lite wrapped multiply-accumulate (MAC) module.

Mallet Annotations: Mallet starts with a Chisel design and annotations, which come in three tiers. *Protocol* annotations define interface types (currently AXI4-Lite). *Memory-map* annotations give each bus address an access mode (e.g. `RO`) or a role, which pairs a mode with a relation (e.g. `Result gatedBy status_r`). *Manual* properties are hand-written. Note: only manual properties can reference signals inside the design; the other tiers only probe the protocol interface, so φ persists through redesigns even as $\mathcal{M}$ shifts.

Formal Properties: Each candidate property has 3 representations — natural language (English), Chisel assertions, and an abstract syntax tree format — so redundant or vacuous properties filter out before lowering to btor2. Two passes are executed in generation: once to generate the properties themselves, and once to evaluate reachability, which culls unreachable antecedents before the property set lowers.

Adjudication matrix: Every surviving property is checked in parallel by three model checkers, which each represent a different solving technique:

- btormc → Bounded Model Checking
- IC3 → bit-level Property Driven Reachability (PDR)
- IC3-IA → word-level PDR

Each engine's results are then combined into a matrix that reports all engine outcomes and a unified verdict. Possible verdicts are:

- `PROVEN`, an unbounded proof
- `NOCEX`, a bounded check passed but PDR didn't terminate
- `REFUTED`, a counterexample was found and generated a waveform
- `VACUOUS`, labelled for honesty

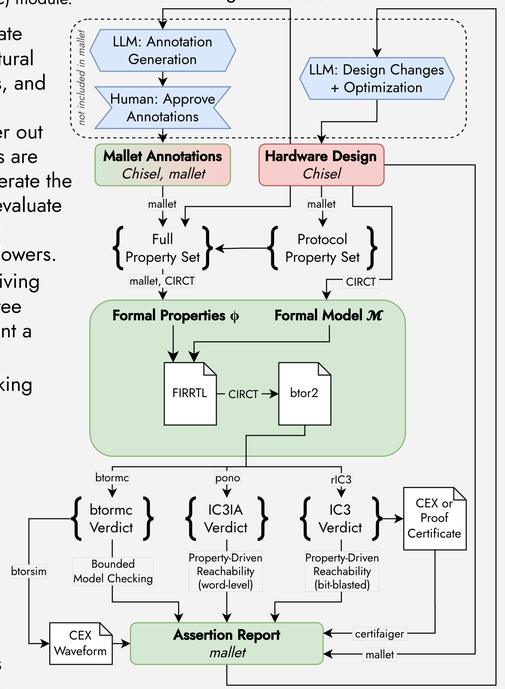


Figure 2 (above): An architectural diagram of Mallet, showing the dataflow, key tool invocations, and artifacts.

```
-------------------------------------------------------------------------------
MODULE   PROPERTY              RCH  btormc   rIC3     pono      VERDICT   ENGLISH
MacSpec  axi_r_valid_stable    yes  NOCEX-20 PROVEN   PROVEN    PROVEN    Whenever rvalid was high and rready was low on the previous cycle, rvalid is high.
MacSpec  axi_b_solicited       yes  NOCEX-20 PROVEN   PROVEN    PROVEN    Whenever bvalid is high, awAccepted is high and wAccepted is high.
MacSpec  result_not_dropped    yes  CEX@11   CEX@12   TIMEOUT   REFUTED   Whenever dut.io.out.fire was high on the previous cycle, dutValidReg is high.
Canaries vacuity_canary         ?   -        -        -         VACUOUS   At all times, dutDataReg equals dutDataReg.
Canaries reach_canary          no   NOCEX-20 PROVEN   PROVEN    VACUOUS   Whenever reachCtr equals 7, dutValidReg is high.
--- MacSpec / result_not_dropped  [refuted] CEX@12 ---
  btormc CEX@12; rIC3 CEX@12; pono TIMEOUT
  waveform: generated/mallet/btor2/MacSpec.p21.cex.vcd
```

Figure 3: A code snippet from the log file that is generated after each run. The originating module, each engine's results, the verdict, and the property's natural language representation are kept coherent by a JSON sidecar. *Note: this snippet was simplified for presentation purposes, but is representative of the original file.*

Results

1) Role to Property Generation

To probe utility and transferability across multiple module types, Mallet was tested on five AXI-Lite wrapped DUTs: a register, an integer Fletcher checksum, an IEEE-754 adder, a multiply-accumulate, and a streaming top-k selector.

DUT	Compute core	Mem. map	AXI	Manual	Total
`CmdSpec`	register file	8	16	1	25
`FletcherSpec`	Fletcher checksum (int)	12	16	2	30
`FPAddSpec`	FP adder	13	16	2	31
`MacSpec`	FP MAC	13	16	2	31
`TopKSpec`	streaming top-k	16	16	1	33

Protocol properties are fixed by the interface, so every DUT receives the same 16, while memory-map properties grow with the size of the address map. Together these two tiers produced 93–97% of each DUT's properties from just 5 to 8 lines of annotation; only 1 to 2 properties per DUT were hand-written.

2) Fault Injection

To check that annotations generate their intended checks, we built 11 faulty variants of the AXI-wrapped MAC, isolating one behavior each, including:

- 2 vacuous properties (a tautology, an unreachable antecedent),
- 8 memory-map faults (e.g. a write to the read-only status register answered OKAY, a result readable before it is ready), and
- 1 AXI handshake violation (`bvalid` withdrawn before acceptance).

Every variant produced its intended verdict: the vacuous properties reported `VACUOUS` (tautology eliminated by CIRCT, unreachable antecedent by Mallet), and every other fault was REFUTED with a counterexample waveform.

Limitations and Future Work

- Agentic harness and LLM queries not yet implemented
  ↳ Agents can attach at two points: the agent can suggest Mallet annotations for human approval (co-creating φ), and can optimize the design using the adjudication matrix results while keeping annotations read-only.
  ↳ Integration with CHIA [5] could support a holistic open-source arch.
- No unbounded liveness, due to lack of support in open-source tools
  ↳ Future CIRCT (Circuit IR Compilers and Tools) contributions add support
- Only supports one protocol (AMBA AXI4-Lite), but could be easily extended
- Poor scaling w.r.t. design complexity and increasing levels of hierarchy
  ↳ Large hierarchical designs need formal decomposition, an intriguing approach but ultimately outside this work's scope

References:

[1] M. Orenes-Vera, A. Manocha, D. Wentzlaff, and M. Martonosi, "AutoSVA: Democratizing Formal Verification of RTL Module Interactions," Apr. 2021.
[2] M. Orenes-Vera, M. Martonosi, and D. Wentzlaff, "Using LLMs to Facilitate Formal Verification of RTL," Sept. 2023
[3] C. Papon and Y. Xiao, "SpinalHDL." [Online]. Available: https://github.com/SpinalHDL/SpinalHDL
[4] M. Xiang, Y. Li, and Y. Zhao, "ChiselFV: A formal verification framework for chisel," in 2023 Design, Automation Test in Europe Conference Exhibition.
[5] A. Cui, F. Hermida-Rivera, J. Toubes, R. Gupta, J. Fang, C. L. Zhang, E. Schwarz, J. Kim, Y. S. Shao, B. Nikolic, C. W. Fletcher, and S. Karandikar, "CHIA: An open-source framework for principled, agentic AI-driven hardware/software co-design research".

Acknowledgements:

This work was primarily supported by the XSCOPE project jointly funded by the U.S. Department of Energy, Office of Science, Advanced Scientific Computing Research (ASCR) and Basic Energy Sciences (BES), under Contract No. DE-AC02-06CH11357, and was also supported in part by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research's Computer Science Competitive Portfolios program under Contract No. AC02-05CH11231.
We are grateful for the resources provided by the Joint Laboratory for System Evaluation (JLSE) at Argonne National Laboratory.