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

Raphael Huang
Argonne National Laboratory
Lemont, IL, USA
rhuang72@ucsc.edu

Kazutomo Yoshii
Argonne National Laboratory
Lemont, IL, USA
kazutomo@anl.gov

I. Introduction

Custom domain-specific accelerators are receiving increasing attention for the speedups they deliver on specialized workloads: scientific instruments utilize in-house accelerators to sustain extreme data rates — X-ray detectors are approaching the terabyte-per-second range [1], [2] — while accelerators for molecular dynamics have reached orders-of-magnitude gains over general-purpose systems [3]. But their growing complexity and specialization make verification hard, increasing the risk that design flaws corrupt results. This risk sits in tension with development agility: experimental requirements shift quickly, forcing frequent redesigns of protocol- and datapath-heavy accelerators and demanding a verification flow that keeps pace.

Conventional simulation-based verification may execute relatively quickly, but a testbench cannot catch bugs it didn't check. Formal verification (FV) fills this gap by combining a property φ and a model $\mathcal{M}$ of the design and feeding them into a model-checking engine, which either proves or rejects whether $\mathcal{M}$ *models* φ. Yet FV is far from easy: good properties are hard to write [4] and exhaustive proof scales poorly as the reachable state space grows exponentially with design size [5].

Recently, LLM-based agentic EDA systems started rapidly generating designs and specifications from prompts [6]–[11], making the *origin* of a specification a primary concern. When $\mathcal{M}$ and φ derive from the same ambiguous natural language description, an LLM can insert a bug in the design and mirror the bug in the property meant to catch it: a *common-mode failure*. A trustworthy flow must ground φ independently of $\mathcal{M}$ and preserve that grounding as the design changes.

Open-source FV tooling has evolved in two directions to support AI for EDA:

1) Writing good formal properties is hard, so engineers often avoid FV [4]. Tools like AutoSVA [4], [12] **render properties from annotations** instead.
2) Design-specification drift occurs when signals are renamed/repurposed. **Utilizing alternatives to Verilog** (e.g. Chisel, SpinalHDL [13]) **so coherence is maintained at compile-time** fixes this.

Yet, to our knowledge, no existing flow unites the two directions, and most query one engine per property, rendering them unable to rank proof strength or flag vacuous passes.

TABLE I
Feature comparison in annotation- and HCL-driven formal property generation tools.

	AutoSVA(2) [4], [12]	**ChiselFV** [14]	**Mallet (this work)**
Auto. property generation	✓	✗	✓
Spec/design coherence	✗	✓	✓
Unbounded liveness	✓	✗	✗
Fully open-source flow	✗	✗	✓

We present Mallet[1], an open-source FV flow for Chisel that unifies the two directions and adds a third (Table I). Mallet renders design annotations into properties and issues them to a portfolio of model-checking engines, producing an adjudication matrix to support agile verification of hardware accelerators.

TABLE II
Design annotations in a MalletSpec for an AXI4-Lite MAC.

```
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 --
    property("result_not_dropped") { dut.io.out.
    fire |=> dutValidReg }
    done()
}
```

II. Methodology

A. Annotations

Mallet centers around annotations on a Chisel design $\mathcal{M}$, which produce φ in three tiers. *Protocol* annotations on

[1]Open-source repository available at https://github.com/rphlhuang/mallet.

TABLE III
EXAMPLE ADJUDICATION MATRIX FOR THE MULTIPLY-ACCUMULATE DUT.

Property	RCH[1]	btormc	rIC3	pono	Verdict
axi_r_val_stable	✓	NOCEX[2]	PROOF	PROOF	PROVEN
axi_b_solicited	✓	NOCEX[2]	PROOF	T/O[3]	PROVEN
result_no_drop	✓	CEX@12	CEX@13	T/O	REFUTED
vacuity_test	?	–	–	–	VACUOUS
unreachable_test	✗	NOCEX[2]	PROOF	PROOF	VACUOUS

[1] Antecedent reachability. vacuity_test was removed before reachability was determined; unreachable_test has an unreachable antecedent, so Mallet marks it VACUOUS despite three PROVEN engine results.
[2] No counterexample after k steps; in this run $k = 20$.
[3] Terminated by a user-defined timeout.

typed interfaces (e.g. AXI4-Lite) generate spec-conformance properties. *Memory-map* annotations give each address an access mode (e.g. RO) or a role, which pairs an access mode with a relation (e.g. Result gatedBy status_r). *Manual* properties are hand-written à la Chisel AssertProperty/AssumeProperty statements.

Only manual properties may reference signals inside $\mathcal{M}$; the other tiers only probe its protocol-level interface, so φ stays grounded in human-approved truth.

B. Custom Grammar

Mallet encodes each property in three linked formats. A natural language (English) description aids human and LLM understanding; a Chisel assertion ensures properties are always syntactically correct (solving a common issue of LLM-generated SVA); and an AST representation allows us to reduce redundancy before dispatching properties to the backend.

C. Adjudication matrix

Mallet runs every property p through every engine e, creating $p \times e$ parallel, independent threads. The engines are complementary: btormc performs bounded model checking, while rIC3 and pono perform unbounded proof. The resulting matrix (Table III) reports the outputs of each thread and one verdict per property that emphasizes the strongest available result, ranking an unbounded proof (φ holds in all reachable states) above a bounded one (φ holds up to depth k). Through this we display not just whether a property holds but how conclusively it was proven.

III. EVALUATION

We tested Mallet on five AXI4-Lite DUTs — a register file, a Fletcher checksum, a floating-point adder, a multiply-accumulate, and a streaming top-k selector — generating 25-33 properties each from across the three tiers. Then, to show annotations correctly compile to the intended checks, we injected several faults into the AXI-wrapped multiply-accumulate and found all produced the intended verdict, with vacuity caught statically and the dataflow and protocol bugs refuted by counterexample (Table IV).

TABLE IV
FAULT INJECTIONS ON AXI-WRAPPED MAC; INTENDED VERDICTS WERE PRODUCED

Origin[1]	Class	Injected fault	Verdict
manual	vacuity	tautology[2]	VACUOUS
manual	vacuity	unreachable antecedent[3]	VACUOUS
memmap	access	status write accepted	REFUTED
memmap	access	operand readable	REFUTED
memmap	access	status read refused	REFUTED
memmap	relation	status stuck ready	REFUTED
memmap	relation	early result read	REFUTED
memmap	relation	ready result refused	REFUTED
memmap	decode	read address alias	REFUTED
memmap	decode	write address alias	REFUTED
protocol	handshake	bvalid withdrawal	REFUTED

[1] Origin of the property: memmap from memory-map annotations, protocol from the AXI4-Lite contract, manual hand-written properties.
[2] Removed by the CIRCT compiler.
[3] Caught by Mallet.

IV. LIMITATIONS AND FUTURE WORK

Mallet has no agentic harness, but agents could earn their keep in two spots: 1) agent-proposed, human-approved annotations, and 2) design iterations against read-only annotations. CHIA [15] offers an open-source agentic architecture for integration. Only one protocol (AXI4-Lite) is currently supported, but Scala allows us to easily extend support to others. Further, unbounded liveness properties are not supported (Table I) as open-source tooling currently lacks support. Finally, like all FV, Mallet may scale poorly; large hierarchical designs require formal decomposition, which is outside this work's scope.

V. ACKNOWLEDGMENTS

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.

REFERENCES

[1] S. Strempfer, K. Yoshii, M. Hammer, D. Bycul, and A. Miceli, "Designing a streaming data coalescing architecture for scientific detector asics with variable data velocity," in *2021 3rd Annual Workshop on Extreme-scale Experiment-in-the-Loop Computing (XLOOP)*, 2021.

[2] M. Hammer, K. Yoshii, and A. Miceli, "Strategies for on-chip digital data compression for x-ray pixel detectors," *Journal of Instrumentation*, vol. 16, no. 01, 2021. [Online]. Available: http://dx.doi.org/10.1088/1748-0221/16/01/P01025

[3] D. E. Shaw, M. M. Deneroff, R. O. Dror, J. S. Kuskin, R. H. Larson, J. K. Salmon, C. Young, B. Batson, K. J. Bowers, J. C. Chao, M. P. Eastwood, J. Gagliardo, J. P. Grossman, C. R. Ho, D. J. Ierardi, I. Kolossváry, J. L. Klepeis, T. Layman, C. McLeavey, M. A. Moraes, R. Mueller, E. C. Priest, Y. Shan, J. Spengler, M. Theobald, B. Towles, and S. C. Wang, "Anton, a special-purpose machine for molecular dynamics simulation," *Commun. ACM*, vol. 51, no. 7, p. 91–97, Jul. 2008. [Online]. Available: https://doi.org/10.1145/1364782.1364802

[4] M. Orenes-Vera, A. Manocha, D. Wentzlaff, and M. Martonosi, "AutoSVA: Democratizing Formal Verification of RTL Module Interactions," Apr. 2021, arXiv:2104.04003 [cs.AR]. [Online]. Available: http://arxiv.org/abs/2104.04003

[5] E. M. Clarke, W. Klieber, M. Nováček, and P. Zuliani, "Model checking and the state explosion problem," in *Tools for Practical Software Verification (LASER 2011)*, ser. Lecture Notes in Computer Science. Springer, 2012, vol. 7682.

[6] Z. Zang, Y. Song, A. Wang, B. W.-K. Ling, Q. Sun, Z. Lei, F. Yang, C. Zhuo, and J. Luo, "The Dawn of Agentic EDA: A Survey of Autonomous Digital Chip Design," 2025, version Number: 2. [Online]. Available: https://arxiv.org/abs/2512.23189

[7] C. Yu, C. Deng, N. Pinckney, and B. Khailany, "Agentic Hardware Design as Repository-Level Code Evolution," Jun. 2026, arXiv:2606.28279 [cs.AR]. [Online]. Available: http://arxiv.org/abs/2606.28279

[8] Y. Wu, W. Fang, J. Wang, W. Li, Z. Guo, and Z. Xie, "AssertLLM2: A Comprehensive LLM Benchmark for Assertion Generation from Design Specifications," May 2026, arXiv:2605.27472 [cs.AR]. [Online]. Available: http://arxiv.org/abs/2605.27472

[9] Y.-A. Shih, A. Lin, A. Gupta, and S. Malik, "FLAG: Formal and LLM-assisted SVA Generation for Formal Specifications of On-Chip Communication Protocols," Apr. 2025, arXiv:2504.17226 [cs.AR]. [Online]. Available: http://arxiv.org/abs/2504.17226

[10] L. Muller, C. K. Jha, B. Arlt, M. Hassan, and R. Drechsler, "Modular End-to-End Pipeline for Formal Property Verification using Large Language Models."

[11] S. Rajabi, C. Yang, and S. Patnaik, "STELLAR: Structure-guided LLM Assertion Retrieval and Generation for Formal Verification," Mar. 2026, arXiv:2601.19903 [cs.AR]. [Online]. Available: http://arxiv.org/abs/2601.19903

[12] M. Orenes-Vera, M. Martonosi, and D. Wentzlaff, "Using LLMs to Facilitate Formal Verification of RTL."

[13] C. Papon and Y. Xiao, "SpinalHDL." [Online]. Available: https://github.com/SpinalHDL/SpinalHDL

[14] M. Xiang, Y. Li, and Y. Zhao, "Chiselfv: A formal verification framework for chisel," in *2023 Design, Automation Test in Europe Conference Exhibition*, 2023.

[15] 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," Jul. 2026, arXiv:2606.27350 [cs.AR]. [Online]. Available: http://arxiv.org/abs/2606.27350