MAESTRO: A Multi-Agent EDA Orchestrator for Autonomous FPGA Design Closure

Saher Elsayed
Department of Computer and Information Science
University of Pennsylvania
Philadelphia, PA, USA
selsayed@seas.upenn.edu
ORCID: 0009-0007-7672-264X

***Abstract*—Modern FPGA design closure remains a fundamentally human-in-the-loop activity. Engineers iterate across synthesis, place-and-route, static timing analysis, and verification, often spending days reconciling tool reports that no single agent can fully internalize. We argue that this iteration loop is structurally well-suited to agentic AI: each stage exposes a distinct interface, produces machine-readable artifacts, and admits localized expertise. We present MAESTRO, a multi-agent orchestrator that decomposes FPGA closure into specialized agents (Planner, Synthesis, Timing, Routing, Verification, Critic) coordinated through a shared blackboard and a tool-grounded action space. A preliminary prototype on five representative benchmarks targeting Xilinx Zynq UltraScale+ reaches timing closure on three designs without human intervention versus two for a scripted baseline, reduces closure iterations by roughly 1.4x on designs where both flows close, and recovers an average of 24% of worst negative slack in the first Critic-guided feedback cycle. Two benchmarks remain unclosed, and we report on the Critic's diagnoses to make those failures actionable. We discuss design principles, failure modes, and open questions for archiving and evaluating agentic EDA workflows.**

***Index Terms*—Agentic AI, FPGA, EDA automation, Multi-agent systems, Timing closure, Large language models, Reinforcement learning.**

I. Introduction

FPGA design closure is widely regarded as one of the most tool-heavy and least automatable parts of the hardware stack. A single closure attempt on a mid-size design touches five or more independent tools (synthesis, mapping, placement, routing, static timing analysis, verification), produces gigabytes of intermediate artifacts, and demands cross-stage reasoning that current point tools do not provide. The conventional response, scripted Tcl flows wrapped around vendor backends, captures recurring patterns but cannot reason about novel violations, cannot adapt strategy mid-flow, and cannot explain why a given fix succeeded or failed.

Recent work on large language models (LLMs) for hardware has demonstrated that natural-language reasoning over tool reports is feasible and often competitive with hand-tuned heuristics [1], [3], [5]. In parallel, reinforcement learning (RL) has produced strong policies for narrow EDA subtasks such as placement, routing, and clock tree synthesis [2], [6], [7]. Both lines of work, however, address single stages in isolation. What is missing is an orchestration layer that lets specialized agents collaborate across the closure loop, share state, and decide jointly when to escalate, restart, or commit.

This paper takes a first step toward such an orchestration layer. We propose **MAESTRO**, a Multi-Agent EDA orchesTRator that treats FPGA closure as a collaborative planning problem. MAESTRO is organized around four design principles:

1) **Stage-aligned specialization.** Each agent owns one EDA stage, holds the corresponding tool grammar and report parsers, and exposes a typed action interface to peers.
2) **Tool-grounded actions.** Every agent action ultimately resolves to a vendor or open-source EDA tool invocation. Hallucinated fixes never reach the design database.
3) **Shared blackboard memory.** A versioned, append-only blackboard records every report, decision, and tool call. Agents reason over this history rather than free-form chat.
4) **Critic-mediated convergence.** A dedicated Critic agent monitors progress, detects oscillation, and decides between continuation, rollback, and human escalation.

We describe the framework, prototype it on five benchmarks ranging from a CORDIC core to a small CNN accelerator, and report early results on closure success, iteration count, and slack recovery. Because the Architecture 2.0 workshop is non-archival, we treat this submission as a vision-grounded work-in-progress and emphasize the questions that the community still needs to answer: how to evaluate agentic EDA, how to share traces, and how to bound cost.

II. Background and Motivation

A. Why FPGA closure resists conventional automation

Three properties distinguish FPGA closure from logic synthesis or ASIC backend flows. First, the design fabric is fixed: optimizations must respect a discrete grid of LUTs, BRAMs, DSPs, and routing channels, which makes continuous-relaxation methods brittle. Second, timing is reported per-path with idiosyncratic exception logic (false paths, multicycle constraints, asynchronous clock groups), so a single global objective rarely captures the engineer's intent. Third, the design knowledge needed to fix a violation is typically not in

the timing report itself: it lives in the RTL, in pragmas, in the constraint file, and in the engineer's mental model of the algorithm.

A purely scripted flow cannot bridge these views. A purely LLM-driven flow can read RTL and reports but lacks the discipline to obey tool semantics. Agentic orchestration sits between the two: it preserves human-readable reasoning while routing every concrete decision through a tool that enforces correctness.

B. Related work

LLM-assisted hardware design has progressed quickly along three axes: RTL generation [3], [4], report interpretation [1], and design-space exploration assistance [5]. RL has been applied to placement [6], routing, scheduling [2], and clock domain management. Multi-agent LLM frameworks such as AutoGen [8] and ReAct-style planners [9] have demonstrated robust task decomposition in software domains but have not been systematically evaluated on closed-loop EDA flows. MAESTRO draws on all three lines but is, to our knowledge, the first attempt to evaluate a fully agentic closure loop on FPGA targets with vendor tools in the inner loop.

III. The MAESTRO Framework

Figure 1 illustrates the MAESTRO pipeline. We describe each component in turn.

A. Planner agent

The Planner is an LLM-based agent that ingests the user intent (a natural-language goal, an SDC constraint file, and a top-level RTL pointer) and emits a typed plan in the form of a directed acyclic graph of stage invocations. Plan nodes are not free text: they conform to a schema with fields for the target stage, expected artifacts, success predicates, and fallback actions. Internally, the Planner pairs an LLM core with a schema validator that rejects any plan that does not type-check, which lets the Critic verify plan well-formedness before any tool runs.

B. Specialist agents

MAESTRO instantiates five specialists, each pairing an LLM reasoner with a parser library for its stage:

- **Synthesis Agent** owns synthesis directives, attribute pragmas, and resource-balancing decisions. It can rewrite small RTL regions when authorized by the Planner. Artifacts: `.v`, `.xdc`.
- **Routing Agent** owns placement seeds, congestion-driven directives, and physical optimization passes. Artifacts: `.dcp`.
- **Timing Agent** parses static timing reports, classifies violations by root cause (logic depth, routing detour, clock skew, false-path leakage), and proposes localized fixes. It benefits most from prior work on LLM-driven timing diagnosis [1]. Artifacts: `.rpt`.
- **Verification Agent** maintains a regression suite and gates every closure attempt with functional sign-off. It can synthesize new assertions when timing-driven RTL edits touch covered logic. Artifacts: `.vcd`.
- **Resource Agent** tracks utilization, DSP and BRAM packing, and arbitrates conflicts between Synthesis and Routing. Artifacts: `.utl`.

C. Typed adapter bar and tool grounding

Every agent action passes through a typed adapter bar, a thin layer that exposes a stage-specific API (`synth.api`, `route.api`, etc.) and forbids any tool invocation that does not type-check. The adapter then resolves the call to Vivado, Quartus, OpenROAD, PrimeTime, or a simulator and captures structured output. This separation is what prevents the framework from drifting into hallucinated tool flags, a recurring failure mode in early LLM-EDA prototypes.

D. Shared blackboard

All inter-agent communication flows through an append-only, content-addressed blackboard whose entries are versioned and hash-identified (Fig. 1, right). Each entry records the producing agent, the tool invocation that grounded it, a hash of the input artifacts, and the resulting report. Agents read by query rather than by message passing, which eliminates conversational drift and makes the entire run reproducible. The blackboard also serves as the natural unit of release for community sharing: a single archive captures every decision and every artifact, enabling post-hoc replay and third-party evaluation.

E. Critic and iteration loop

The Critic is a smaller, deliberately conservative model whose only role is to monitor the blackboard for three failure modes, surfaced as the three indicators in Fig. 1: oscillation (the same fix is proposed and reverted), regression (a previously closed metric degrades), and saturation (no metric has improved for k iterations). On detection, the Critic either rolls back to the last green snapshot or escalates with a structured incident report. The Critic never proposes fixes; this separation of concerns is essential for stability and is consistent with classical control-theoretic supervisor designs. When closure is not yet achieved, the green-dashed iteration loop returns control to the Planner for round $n{+}1$.

IV. Preliminary Implementation and Case Study

A. Setup

We prototyped MAESTRO in Python using a planner built on a 70B-class open-weight model and specialists built on a mix of 8B reasoners and rule-based parsers. The Critic runs on a small distilled model. Vendor tools are invoked through Vivado 2023.2 in batch mode. The target device is the Xilinx Zynq UltraScale+ ZU9EG. We evaluated five benchmarks:

- **CORDIC-16**: 16-stage pipelined CORDIC.
- **FFT-1024**: radix-2 streaming FFT.
- **AES-128**: round-unrolled cipher core.
- **MM-256**: 256x256 matrix multiplier with DSP packing.

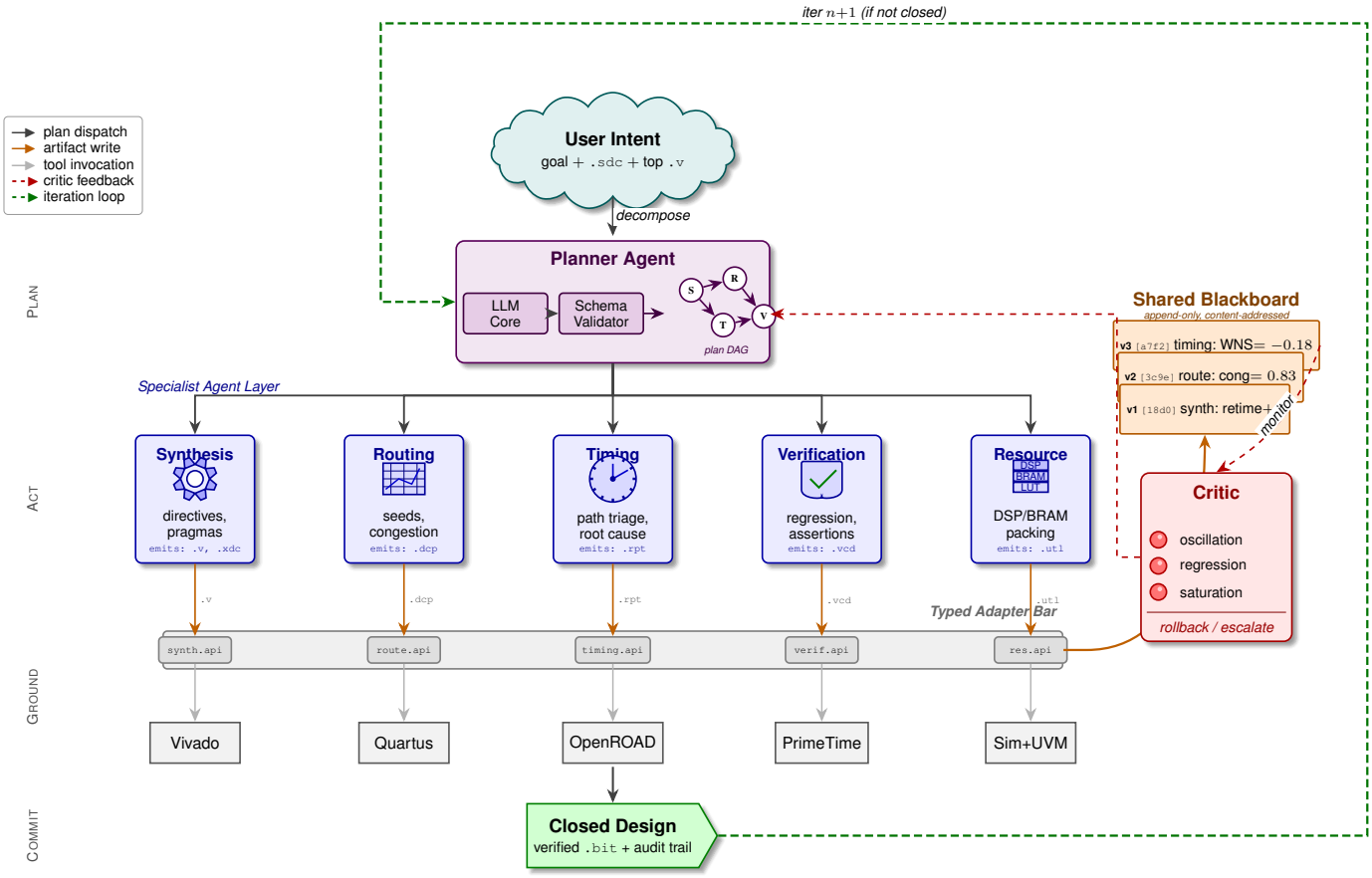

Fig. 1. MAESTRO multi-agent pipeline. The Planner houses an LLM core and a schema validator that together emit a typed plan DAG. The plan is dispatched to five specialist agents, each owning a stage-specific tool grammar and artifact type (*.v*, *.dcp*, *.rpt*, *.vcd*, *.utl*). Every action flows through a typed adapter bar before reaching a vendor or open-source tool. Results are appended to a content-addressed, versioned blackboard that the Critic monitors for oscillation, regression, and saturation; on detection, it triggers rollback or human escalation. A green-dashed iteration loop carries unclosed designs back to the Planner for the next round. Phase labels on the left (PLAN / ACT / GROUND / COMMIT) align with the classical sense-plan-act decomposition.

- **CNN-Tiny**: 3-layer CNN inference accelerator with on-chip BRAM weights.

The baseline is a scripted Tcl flow that mirrors what a senior FPGA engineer would write in a single afternoon: default synthesis strategies, two retry passes with retiming enabled, and a placement-directed restart on first WNS failure. We measure closure success, the number of full closure iterations, worst negative slack (WNS) trajectory, and total wall-clock time.

B. Findings

Table I summarizes the run. MAESTRO closed three of five benchmarks autonomously, compared to two of five for the baseline. The single benchmark where MAESTRO succeeded and the baseline did not, FFT-1024, was diagnosed by the Timing Agent as a missing false-path declaration across a clock-domain boundary; the Synthesis Agent then issued a constraint patch that the scripted flow had no logic to consider. On the two benchmarks both flows closed (CORDIC-16 and AES-128), MAESTRO reduced the average number of full closure iterations from 5.0 to 3.5, a 1.4x reduction. The slack recovery in the first Critic-guided cycle averaged 0.19 ns across all five benchmarks, or roughly 24% of initial WNS, indicating that the first guided fix contributes meaningfully but cannot replace iterative refinement.

MAESTRO did not close MM-256 or CNN-Tiny within its iteration budget. The Critic diagnosed CNN-Tiny as oscillation between two BRAM-packing strategies and MM-256 as

TABLE I
PRELIMINARY CLOSURE RESULTS ACROSS FIVE BENCHMARKS. WNS RECOVERY IS THE SLACK RECOVERED IN THE FIRST CRITIC-GUIDED FEEDBACK CYCLE, IN NANOSECONDS. "N.C." INDICATES THE BASELINE DID NOT CONVERGE WITHIN ITS ITERATION BUDGET; ITERATION SPEEDUP IS REPORTED ONLY ON DESIGNS WHERE BOTH FLOWS CLOSE.

Benchmark	Closed (M/B)	Iters (B)	Iters (M)	WNS recov.	Speedup (M/B)
CORDIC-16	Y / Y	4	3	0.21	1.3x
FFT-1024	Y / N	n.c.	6	0.31	—
AES-128	Y / Y	6	4	0.16	1.5x
MM-256	N / N	n.c.	11+	0.22	—
CNN-Tiny	N / N	n.c.	12+	0.07	—
Mean (both closed)	3/5 / 2/5	5.0	3.5	0.19	**1.4x**

M: MAESTRO, B: scripted baseline.

saturation: WNS plateaued after the seventh iteration. We view these surfaced diagnoses as the more useful output for a WIP system, since they convert opaque tool failures into hypotheses an engineer can act on.

C. Cost and reproducibility

Each successful run cost on the order of 40K to 110K tokens across all agents combined, dominated by Timing Agent report parsing on long failing-path lists. The full blackboard for each benchmark fits in under 80 MB and replays deterministically given the same tool versions. This is, in our view, the more important result: agentic EDA only becomes a credible research target if individual runs are auditable and shareable.

V. Open Challenges and Future Work

Even at this early stage, three challenges are already sharp.

Evaluation methodology. The community has no agreed-upon benchmark suite for closed-loop agentic EDA. Single-stage benchmarks (e.g., placement quality on ISPD) do not capture the cross-stage reasoning that motivates agentic frameworks in the first place. We argue for a shared benchmark of closure traces, not just designs, so that frameworks can be compared on their ability to navigate the same starting failures.

Cost accounting. Token cost, tool-license cost, and engineer time interact in ways that traditional EDA evaluation ignores. A framework that closes a design in three hours for $2 of inference is not directly comparable to one that closes in twelve hours for free. We are extending MAESTRO with a unified cost ledger that tracks all three.

Agent specialization vs. generalization. Our specialists today are tightly coupled to Vivado. Porting to Quartus or OpenROAD requires re-grounding every action, which suggests that the right abstraction is a portable EDA action language rather than per-tool adapters. We see this as a natural next step and an obvious community artifact.

Beyond these, longer-term directions include integrating learned policies (for example, RL-trained placement seed generators [2]) directly as agent actions, supporting multi-die and chiplet flows where the orchestration surface grows by an order of magnitude, and exploring whether the same blackboard model can host ASIC backend flows where the tool surface is larger but the iteration semantics are similar.

VI. Conclusion

MAESTRO is an early attempt to treat FPGA design closure as a multi-agent orchestration problem rather than a scripted pipeline. The framework is built on stage-aligned specialization, tool-grounded actions, a shared blackboard, and a separate Critic. A preliminary five-benchmark study shows that this structure closes one design that a scripted baseline misses while reducing closure iterations by 1.4x on designs that both flows close, with a 24% first-cycle slack recovery and two unresolved failures that the Critic converts into actionable diagnoses. The broader contribution of this work, however, is not the numbers: it is a concrete substrate on which the Architecture 2.0 community can debate what agentic EDA should look like, how it should be evaluated, and how its traces should be shared. We release the blackboard format and benchmark traces as a starting point for that conversation.

References

[1] S. Elsayed, "TimingLLM: LLM-driven diagnosis of FPGA timing violations with retrieval-augmented generation," in *Proc. ASPLOS Workshop on Architecture 2.0*, 2026.

[2] S. Elsayed, "StreamLearn-FPGA: An online learning accelerator for resource-constrained reconfigurable platforms," in *Proc. Int. Symp. Applied Reconfigurable Computing (ARC), LNCS*, Springer, 2026.

[3] S. Liu et al., "RTLCoder: Outperforming GPT-3.5 in design RTL generation with a lightweight LLM," in *Proc. IEEE/ACM Design Automation Conference*, 2024.

[4] S. Thakur et al., "VeriGen: A large language model for Verilog code generation," *ACM Trans. Design Autom. Electron. Syst.*, 2023.

[5] M. Liu et al., "ChipNeMo: Domain-adapted LLMs for chip design," *arXiv preprint*, 2023.

[6] A. Mirhoseini et al., "A graph placement methodology for fast chip design," *Nature*, vol. 594, 2021.

[7] Y. Cheng and L. Wang, "DeepPlace: Learning to place via deep reinforcement learning," in *Proc. ICCAD*, 2021.

[8] Q. Wu et al., "AutoGen: Enabling next-generation LLM applications via multi-agent conversation," *arXiv preprint*, 2023.

[9] S. Yao et al., "ReAct: Synergizing reasoning and acting in language models," in *Proc. Int. Conf. Learning Representations*, 2023.

[10] L. Chen et al., "A survey on large language models for electronic design automation," *IEEE Trans. CAD*, 2024.

[11] T. Ajayi et al., "OpenROAD: Toward a self-driving, open-source digital layout implementation tool chain," in *Proc. Government Microcircuit Applications Conf.*, 2019.

[12] Xilinx Inc., "Vivado design suite user guide: implementation," UG904, 2023.