ASPEN: LLM-Guided E-Graph Rewriting for RTL Datapath Optimization

Niansong Zhang[†*], Chenhui Deng[§], Johannes Maximilian Kuehn[§], Chia-Tung Ho[§], Cunxi Yu[‡], Zhiru Zhang[†], Haoxing Ren[§]
[†]Cornell University, [‡]University of Maryland, College Park, [§]NVIDIA Corporation
{nz264, zhiruz}@cornell.edu, cunxiyu@umd.edu, {cdeng, jkuhn, chiatungh, haoxingr}@nvidia.com

***Abstract*—Datapath RTL optimization is a challenging multi-objective task that balances power, performance, and area (PPA). Traditionally, this process is done manually due to the complexity of balancing conflicting objectives. Recent approaches have explored rule-based rewriting using equality saturation with e-graphs, which solves the phase ordering problem but relies on manually designed rewrite rules and proxy PPA cost models. Meanwhile, large language models (LLMs) have been applied to RTL code generation and optimization due to their reasoning and programming capabilities. However, existing LLM-based methods lack a structured approach to represent and explore Pareto-optimal design points while ensuring equivalence in transformed RTL. We propose ASPEN, a system that leverages LLMs to guide e-graph rewriting while incorporating accurate, detailed feedback from EDA tools. Our approach employs an agentic system to propose and select rewrite rules, using real PPA feedback for extraction rather than proxy models. This reduces the need for handcrafted rewrite rules and eliminates the need for designing proxy-based cost models. ASPEN achieves average improvements of 16.51% in area and 6.65% in delay over existing e-graph-based RTL optimization approaches across a diverse set of benchmarks. Our results demonstrate that combining LLM-driven e-graph rewrite with real PPA feedback enables scalable and effective datapath RTL optimization.**

***Index Terms*—Register-Transfer-Level implementation, language models, optimization**

I. Introduction

RTL designs typically undergo a series of transformations to explore trade-offs between power, performance, and area (PPA) while preserving functional correctness. However, the order in which these transformations are applied significantly impacts the final PPA outcome, a challenge known as the phase ordering problem. Furthermore, RTL optimization involves an enormous design space. For example, an N-bit prefix adder circuit has a design space of $\mathcal{O}(2^{N^2})$ [1], and a 2048-bit multiplier, when transformed with 25 rewrite rules, results in 1.59×10^{6179} possible expressions [2].

To address the phase ordering problem and efficiently represent large sets of functionally equivalent designs, e-graphs have been applied to RTL optimization. Figure 1 illustrates the Verilog optimization workflow using e-graphs, as employed in works like ROVER [3]. An e-graph is a graph-based data structure constructed iteratively by applying rewrite rules. Once saturation is reached or a specified timeout occurs, the resulting e-graph compactly encodes a vast set of equivalent expressions, enabling the extraction of an optimized design based on a cost model. However, this approach has limitations: the rewrite rule set and cost model must be manually defined by designers, which is non-scalable. Maintaining an extensive rewrite rule set for all possible RTL designs is also impractical, as excessive rules can slow down saturation and complicate the extraction of high-quality design points. Furthermore, designing and customizing new rewrite rules relies on human expertise, making it labor-intensive and time-consuming. Additionally, existing e-graph-based RTL optimization methods rely on PPA proxy cost models, which may lack accuracy and fidelity.

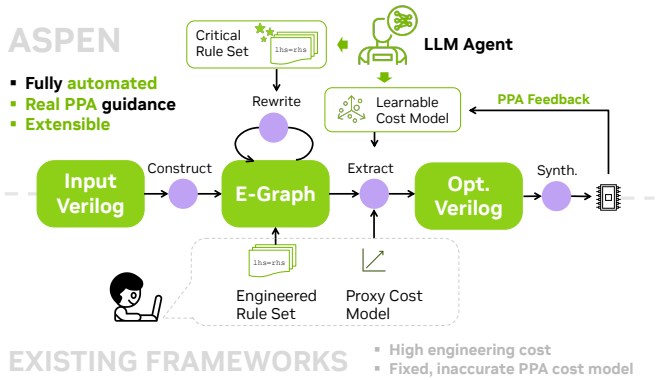


Fig. 1. Comparing ASPEN with existing RTL rewrite optimization methods.

Meanwhile, large language models (LLMs) have been applied to RTL tasks such as code completion, repair, and generation [4]–[8]. Agent-based approaches incorporate LLMs into automated and iterative workflows, as seen in RTLFixer [9] and AutoChip [10]. However, LLMs lack formal guarantees for preserving RTL functionality and do not provide a structured method for representing and exploring Pareto-optimal design points. These limitations can be addressed by integrating e-graphs, while LLM agents can propose, select rewrite rules and extract PPA feedback from EDA tools to guide e-graph rewriting and extraction.

We propose a system that combines the strengths of both approaches while mitigating their respective weaknesses. LLMs automate proposal and selection of critical design-specific rewrite rules and integrate PPA feedback from EDA tools. E-graphs preserve equivalence and encode functionally equivalent design points. This PPA feedback loop guides rule proposal, selection, and e-graph extraction to enhance the quality and scalability of RTL optimization. Our key contributions are:

- **LLM-guided e-graph rewriting**: We use LLM agents to propose new rewrite rules and select critical ones, reducing the manual effort for rule design and improving scalability.
- **Accurate PPA feedback integration**: Instead of relying on manually designed proxy cost models, we leverage EDA tools' PPA feedback to guide rule selection and e-graph extraction.
- **General and extensible PPA feedback extraction**: We introduce a graph-partitioning method to extract PPA metrics from synthesized netlists using LLMs. Our approach is tool-agnostic and demonstrated with area and delay feedback from a commercial

∗ This work was conducted during an internship at NVIDIA.

979-8-3315-3762-3/25/$31.00 ©2025 IEEE

synthesizer, while being readily extensible to other metrics such as power.

- **Equivalence-preserving design space exploration**: We combine e-graphs with LLM-guided rewrite proposals, each automatically verified by a theorem prover, to ensure functional correctness while efficiently exploring the Pareto frontier of RTL designs.

II. Related Work

A. RTL PPA Optimizations

RTL PPA optimization has traditionally relied on manual efforts or heuristic-based methods. Experienced designers manually refine RTL to improve timing, area, and power efficiency by applying techniques such as pipelining for higher frequency operation, eliminating redundant computations, and selecting efficient arithmetic implementations (e.g., Booth multipliers vs. Wallace trees). Heuristic-based transformations have also been widely used in ASIC synthesis tools, which apply optimizations such as constant propagation, strength reduction, and algebraic simplifications. For example, BOOM [11] employs an implicant generation paradigm to efficiently handle functions with numerous input variables and sparse care terms, improving upon earlier bottom-up approaches. Similarly, Childerhose and Liu [12] developed a heuristic method based on Karnaugh mapping to eliminate redundant and selective prime implicants, simplifying sum-of-products Boolean expressions with lower computational complexity than traditional techniques like the Quine-McCluskey method.

More recently, machine learning approaches have been explored to enhance RTL optimization. MapTune [13] leverages RL to make design-specific cell selection decisions during technology mapping, thereby reducing the search space and improving mapping quality. PrefixRL [1] leverages reinforcement learning (RL) to optimize parallel prefix circuits, including adders and priority encoders. By incorporating synthesis into the training loop, PrefixRL obtains accurate PPA rewards, enabling more effective design space exploration.

B. Equality Saturation and E-Graphs

E-Graphs are graph-based data structures constructed through equality saturation, where rewrite rules are iteratively applied until reaching a fixed point or timeout [14], [15]. These rewrite rules can represent compiler transformations or arithmetic decompositions, such as integer multiplication breakdown. With the state-of-the-art equality saturation tool `egg` [14] and `egglog` [15], this technique has been successfully applied to various domains, including linear algebra [16], tensor computation, DSP compilation [17], logic synthesis [2], [18], [19], Boolean reasoning [20], high-level synthesis [21], [22], formal verification [23], and RTL PPA optimization [3], [24], [25].

Within an e-graph, functionally equivalent terms are grouped into equivalence classes, known as **e-classes**. Nodes within an e-class, representing values or operators, are called **e-nodes**. The edges in e-graphs are directed, connecting e-nodes to their child e-classes to capture dependencies between operators and operands. After saturation or timeout, a final extraction step selects the e-nodes that minimize a user-defined cost function [14], [26].

Applying e-graphs to RTL optimization, ROVER [3] represents RTL designs as data-flow graphs and leverages equality saturation for efficient design space exploration. By defining a set of rewrite rules based on bitwidth parameters, their approach automates optimization while achieving performance comparable to experienced hardware engineers. Additionally, they analyze the noise floor in logic synthesis to evaluate the accuracy of cost metrics. Their work demonstrates the potential of automated rewriting to generate diverse architectures reflecting bitwidth-dependent trade-offs. Subsequent work [24], [25] further extends e-graph-based RTL optimization by enabling constraint-aware transformations, expanding the applicability of equality saturation to a broader range of datapath designs.

Algorithm 1: Two-level optimization with LLM-guided rewriting and PPA feedback

foreach *rule_select_iter* **do**
  $rules \leftarrow$ LLM_Select$(past_ppa, pool)$;
  $egraph \leftarrow$ Egglog_Saturate$(program, rules)$;
  $costs \leftarrow$ All_One();
  foreach *ppa_feedback_iter* **do**
    $expr \leftarrow$ EGraph_Extract$(egraph, costs)$;
    $vprogram \leftarrow$ Verilog_Translate$(expr)$;
    $(ppa, netlist) \leftarrow$ Synthesizer$(vprogram)$;
    $critical\_nodes\_inc, critical\_nodes\_dec \leftarrow \emptyset, \emptyset$;
    foreach $part \in$ *Partition*$(netlist)$ **do**
      $(inc, dec) \leftarrow$ LLM_Select$(vprogram, part, ppa)$;
      $critical\_nodes\_inc \leftarrow critical\_nodes\_inc \cup inc$;
      $critical\_nodes\_dec \leftarrow critical\_nodes\_dec \cup dec$;
    foreach $c\_node \in critical\_nodes\_inc$ **do**
      $costs[c\_node] \leftarrow costs[c\_node] + \Delta_{\text{inc}}$;
    foreach $c\_node \in critical\_nodes\_dec$ **do**
      $costs[c\_node] \leftarrow costs[c\_node] - \Delta_{\text{dec}}$;

C. LLMs for RTL Design

Large language models (LLMs) have been increasingly applied to RTL design tasks such as code completion, repair, and generation [4]–[8], [27]–[38]. Beyond single-shot generation, agent-based approaches integrate LLMs into automated and iterative workflows, enabling adaptive refinements and constraint-aware modifications, as demonstrated by RTLFixer [9] and AutoChip [10]. However, LLM-based methods lack formal guarantees for preserving RTL correctness and do not inherently provide a structured framework for exploring Pareto-optimal design trade-offs. These limitations can be mitigated by incorporating e-graphs, which offer a principled representation of equivalences and systematic design space exploration. By extracting PPA feedback and tracing from EDA tools, LLM agents can guide e-graph rewriting and extraction, enabling more effective and structured RTL optimization.

III. ASPEN Framework

We present ASPEN, a framework that leverages LLM agents to propose rewrite rules and select a set of critical rules based on the input design and exploration history. It guides e-graph extraction by adjusting e-node costs using real PPA feedback.

A. Overview

Figure 2 illustrates the end-to-end flow of ASPEN. The input consists of a Verilog datapath design and a set of initial rewrite rules. ASPEN features three key agents: a rewrite rule proposal agent, a rewrite rule selection agent, and a PPA feedback agent. The rewrite rule proposal agent generates new, design-specific rules and verifies their correctness. This proposal phase occurs once, prior to entering the optimization loop, and is based on the initial rule set and the input Verilog datapath design. The rewrite rule selection agent chooses a critical set of rewrite rules based on the input design and prior PPA results. The critical set may either be a subset of the rewrite rules or include all rules from the rewrite rule pool. Using `egglog`, we perform equality saturation on the extracted expression of the

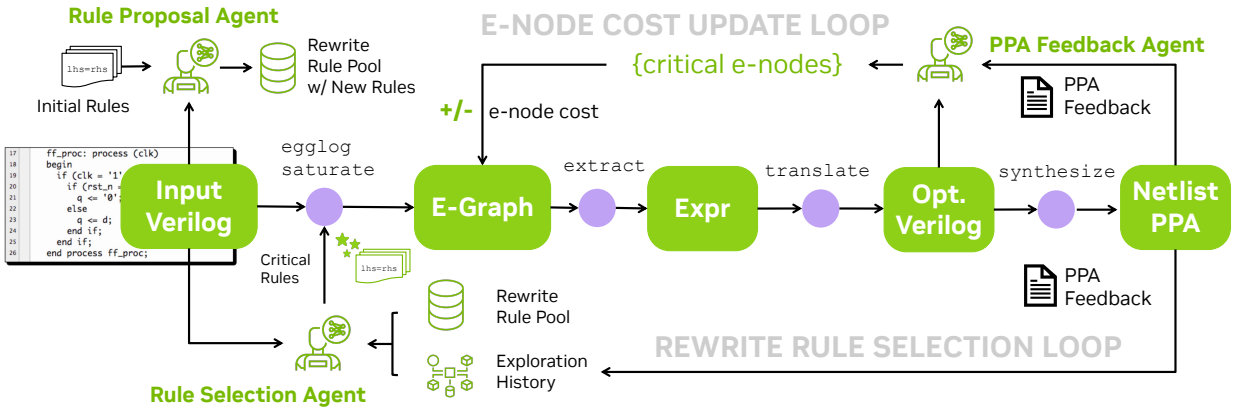


Fig. 2. **ASPEN workflow overview** – The system takes as input a Verilog datapath design and an initial set of rewrite rules. A rewrite rule proposal agent first generates new, design-specific rules based on the input design and initial rule set, verifying their correctness. A rewrite rule selection agent then identifies a relevant subset of rules, which are applied via equality saturation using `egglog`. The resulting e-graph is extracted to produce an optimized RTL design, which is synthesized to obtain PPA metrics. A PPA feedback agent analyzes synthesis results and adjusts e-node weights to guide further extractions. This forms a feedback loop that enables iterative refinement of rewrite selections and e-graph extractions.

input datapath design. The resulting saturated e-graph is extracted to obtain an expression, which is translated into an optimized Verilog design. This design is then synthesized to generate PPA feedback. The PPA feedback agent analyzes the optimized Verilog, the synthesized netlist, and synthesis reports to identify relevant e-nodes and adjust their costs accordingly—either increasing or decreasing them to guide subsequent extractions.

Algorithm 1 presents the two-level optimization loop of ASPEN. The outer loop explores different subsets of rewrite rules, while the inner loop uses PPA feedback to adjust e-node costs and guide the extraction process from the e-graph.

B. Rewrite Rule Proposal

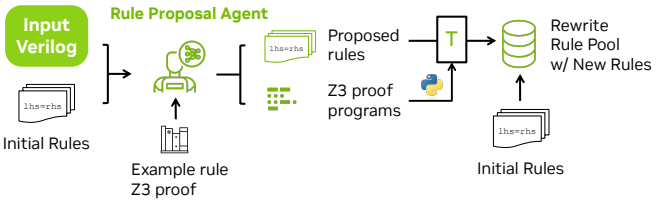


Fig. 3. **Rewrite rule proposal agent** – starting from the input Verilog design, the initial rule set, and example proofs, the agent generates candidate rewrite rules together with Z3 proof scripts. A candidate rule is added to the rule pool only after its proof is successfully verified.

Figure 3 illustrates the rewrite rule proposal agent. It takes as input the Verilog design, an initial set of rewrite rules listed in Table I, and example proof programs for existing rules. The initial rewrite rules and corresponding Z3 proofs serve as examples for in-context learning. The input Verilog design allows the LLM to analyze design patterns and propose design-specific rules. The agent is tasked with proposing new rewrite rule candidates aimed at PPA optimizations, along with corresponding proof programs. Each proof program is then evaluated to check whether the proposed rule preserves functional equivalence. If the rule is verified, it is added to the rewrite rule pool. This rewrite rule proposal phase is executed only once at the beginning of the workflow.

TABLE I
INITIAL REWRITE RULES IN ASPEN

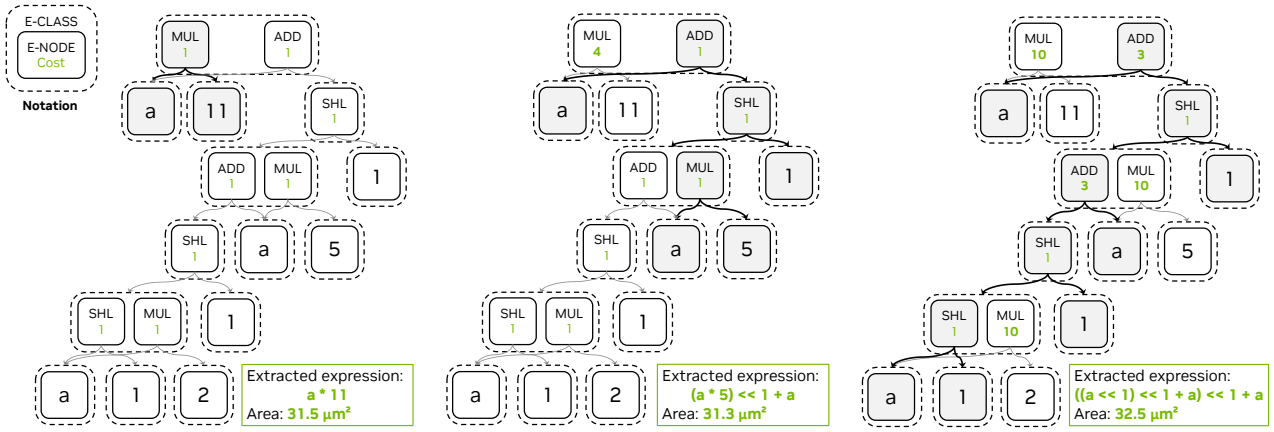
Class	Rewrite Rules
Commutativity	$a + b \Leftrightarrow b + a$ $a \cdot b \Leftrightarrow b \cdot a$
Associativity	$a + (b + c) \Leftrightarrow (a + b) + c$ $a \cdot (b \cdot c) \Leftrightarrow (a \cdot b) \cdot c$
Identity	$0 + a \Rightarrow a$ $0 \cdot a \Rightarrow 0$ $1 \cdot a \Rightarrow a$
Distributivity	$a \cdot b + a \cdot c \Rightarrow a \cdot (b + c)$
Power-of-Two Multiplication	$c \cdot x \Rightarrow x \ll \log_2 c$ if $c = 2^k$
Odd Constant Multiplication	$c \cdot x \Rightarrow 2 \cdot (\lfloor c/2 \rfloor \cdot x) + (c \bmod 2) \cdot x$ if c is odd and $c > 1$
Even Constant Multiplication	$c \cdot x \Rightarrow (c \gg 1) \cdot 2 \cdot x$ if c is even and $c > 0$
Shift Simplification	$(x \ll c) \ll 1 \Rightarrow x \ll (c + 1)$
Estrin's Transformation	$x^2 \cdot a + x^2 \cdot b \Rightarrow x^2 \cdot (a + b)$
Addition Balancing	$(a + b) + (c + d) \Rightarrow (a + c) + (b + d)$
MUXAR Simplification	$a \cdot b + c \cdot \neg b \Rightarrow \text{MUXAR}(b, a, c)$

C. Critical Rule Set Selection

A key observation in our work is that while rewrite rules are the foundation of improvement in e-graph-based RTL PPA optimization, using too many or too few can degrade effectiveness. An excessive number of rewrite rules can cause the e-graph to grow excessively, leading to longer runtimes or failure to reach saturation, ultimately making it difficult to extract a high-quality design. To address this, we propose selecting a *critical* set of rewrite rules based on the characteristics of the input design and feedback from synthesis results. The critical rule selection agent receives three types of input: (1) the input Verilog datapath design, which provides design context; (2) the rewrite rule pool with newly proposed rewrite rules; and (3) the exploration history, including previously selected rule subsets and their best PPA results, which guides the agent based on past performance.

D. Incorporating PPA Feedback

Each e-node in the e-graph is associated with a cost. During extraction, a cost model is used to select a sub-graph from the



(a) Uniform cost. (b) PPA-aware cost. (c) Fixed cost, no PPA feedback.

Fig. 4. **E-Graph extraction example with different e-node costs** – The saturated e-graph is produced by applying rewrite rules for power-of-two multiplication, odd constant multiplication, and even-cost multiplication to the expression `a×11`. The extracted subgraph is highlighted in gray. Leaf nodes `a`, `1` are duplicated to enhance visualization. (a) When all operation costs are set to 1. (b) Based on the PPA result of the initial design point, the cost of the root `MUL` e-node is increased, resulting in a partial unfolding. (c) Without PPA feedback, the cost of all `MUL` e-nodes is uniformly set to a high value, leading to complete unfolding into additions and shifts.

saturated e-graph with the lowest total cost, typically the sum of e-node costs. As such, setting appropriate e-node costs is critical.

Existing e-graph-based RTL PPA optimization methods often rely on heuristics to assign e-node costs based on synthesis feedback. However, this approach has several limitations. First, heuristic cost assignments may lack accuracy or fidelity—-they often fail to reflect the true relative impact of delay or area in the synthesized design. Second, these costs are typically fixed and global, making them unable to adapt dynamically to PPA feedback or apply fine-grained, localized adjustments.

1) Motivating Example: Figure 4 illustrates how e-node costs influence post-synthesis PPA. The target scenario is fixed delay, with the goal of minimizing area. Area information is obtained by synthesizing the design using a commercial synthesis tool with the ASAP7 PDK [39]. The optimization target is the expression `a×11`, and the saturated e-graph is generated by applying rewrite rules for power-of-two multiplication, odd constant multiplication, and even-cost multiplication listed in Table I.

These rules enable the constant multiplication to be recursively unfolded into additions and shifts. Since multipliers typically consume more area than adders or shifters, a natural assumption is to assign a higher cost to multiplication. In Fig. 4c, this is modeled by setting a uniformly high cost for all `mul` e-nodes, which forces full unfolding into additions and shifts. However, this actually results in a larger area than the baseline case shown in Fig. 4a, where all e-nodes are assigned a uniform cost of one.

By incorporating PPA feedback, we can more effectively balance trade-offs between multipliers and the combination of adders and shifters. As shown in Fig. 4b, non-uniform e-node costs—adjusted based on synthesis feedback—lead to an extracted expression that mixes multiplication, shift, and addition operations, resulting in a smaller area than either uniform-cost baseline.

This example highlights that assigning the same fixed cost to all e-nodes of a given type does not necessarily lead to real PPA improvement. Instead, localized e-node cost adjustment guided by PPA feedback can yield more efficient designs.

2) Tracing PPA Results to e-nodes: A key challenge in using PPA results to guide optimization lies in mapping individual operations to components in the synthesized netlist. While commercial synthesis tools provide symbol tracing from Verilog to netlist, these traces are often incomplete or imprecise. We find that LLMs' pattern matching can reliably map netlist-level PPA metrics back to e-nodes by exploiting naming conventions preserved through synthesis (e.g., appended prefixes or suffixes). This enables correlation between e-node groups and netlist subgraphs, even in the presence of structural transformations like adder trees or partial-product networks. Two major challenges remain in applying LLMs for this tracing task:

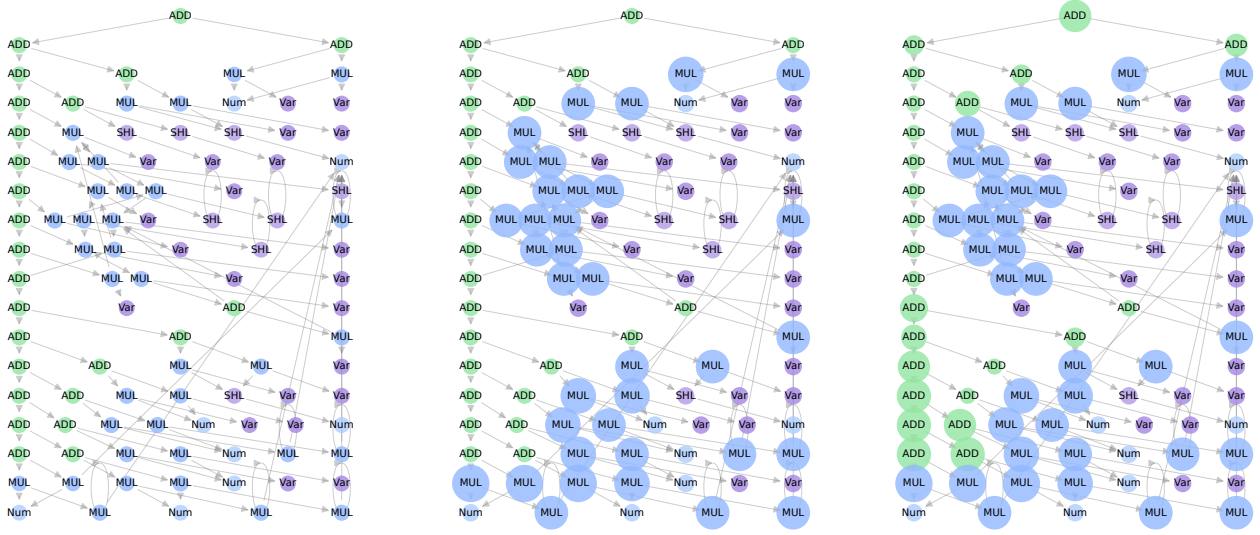
1. Context window limitations. Synthesized netlists can easily contain hundreds of thousands to millions of instances, far exceeding the context window of even the latest LLMs. Even when fitting within the window, the latency of the prefill stage would be prohibitively high. To address this, we apply K-way graph partitioning to divide the netlist into subgraphs. Each subgraph can then be processed independently by the LLM.

2. Graph representation mismatch. LLMs are inherently better suited for processing sequential data than graph-structured data. To make the data more amenable to LLM processing, we serialize each subgraph into a text format that includes instance names along with input and output signal names. These serialized representations capture the structural and naming patterns necessary for effective subgraph-to-e-node-group matching.

3) History Backtracking: To improve design space exploration, our system allows the agent to backtrack e-node cost changes that lead to degraded PPA outcomes. The prompt records recent cost updates and corresponding feedback, enabling the agent to reason about past decisions and undo unfavorable ones. This helps avoid local minima and better navigate non-monotonic optimization landscapes.

E. Verification

To ensure that optimizations preserve the original functionality, we apply an RTL equivalence checker between the input Verilog and each optimized Verilog design. This step verifies that the transformed design remains semantically equivalent to the original. In our implementation, we use a commercial equivalence checking tool to validate all final design points on the Pareto frontier.



(a) Iteration 1. (b) Iteration 5. (c) Iteration 9.

Fig. 5. **Iterative E-node Cost Update in FIR Design using PPA Feedback** – Sequence of saturated e-graphs from a Finite Impulse Response (FIR) filter design, illustrating the update of e-node costs over successive iterations based on Power, Performance, and Area (PPA) feedback. The relative cost of each e-node is visualized by its size. (a) Iteration 1: All e-nodes are assigned an initial, uniform cost. (b) Iteration 5: Following PPA analysis, the costs associated with all `MUL` (multiplication) e-nodes are increased. (c) Iteration 9: Further PPA feedback results in an increased cost for `ADD` (addition) e-nodes, particularly those positioned closer to the leaf nodes in the e-graph.

TABLE II

END-TO-END COMPARISON OF ASPEN WITH ROVER AND THE COMMERCIAL SYNTHESIZER BASELINE ACROSS AREA AND DELAY METRICS. THE TABLE REPORTS ABSOLUTE AREA (IN μm^2) AND DELAY (IN PS) FOR EACH BENCHMARK, ALONG WITH PERCENTAGE IMPROVEMENTS OF ASPEN OVER BOTH BASELINES. THE AREA AND DELAY VALUES FOR ASPEN ARE SELECTED INDEPENDENTLY AS THE BEST (I.E., MINIMAL) VALUES FROM THE PARETO FRONTIER.

Benchmarks	**Commercial Baseline**		**ROVER**		**ASPEN**		**Improv. over Baseline**		**Improv. over ROVER**	
	area(μm^2)	delay (ps)	area (μm^2)	delay (ps)	area (μm^2)	delay (ps)	area (%)	delay (%)	area (%)	delay (%)
FIR	240.31	933.75	194.10	933.75	169.52	804.30	29.46	13.86	12.66	13.86
DCT	3805.31	1355.33	3715.26	1326.31	3666.00	1272.26	3.66	6.13	1.33	4.08
Polynomial	685.41	2492.13	569.99	2486.84	544.65	2486.79	20.54	0.21	4.45	0.00
Watermark	30.66	485.84	28.23	442.18	28.23	442.18	7.94	8.99	0.00	0.00
MCM(3, 7, 21)	32.99	366.17	31.43	332.58	22.83	332.58	30.80	9.17	27.37	0.00
MCM(7, 19, 31)	44.64	462.33	44.64	405.34	25.11	316.12	43.76	31.62	43.76	22.01
MCM(5, 93)	30.53	431.52	27.47	383.62	20.32	358.20	33.43	16.99	26.01	6.63
Average	-	-	-	-	-	-	24.23	12.43	16.51	6.65

IV. EVALUATION

A. Experimental Setup

We evaluate ASPEN on a suite of RTL datapath design benchmarks listed in Table III. All seven designs are open-source. For equality saturation, we use `egglog` version 0.4.0 and employ the fast greedy and fast bottom-up extractors provided by `extraction-gym` [1]. We use a commercial RTL design synthesis tool with advanced technology mapping optimization features. In all experiments, the area, power, and delay optimization efforts are configured to "high" within the synthesis tool to ensure aggressive optimization. We use OpenAI's `gpt-4.1-20250414`[2] to power the rule selection and PPA feedback agents, and `o3-20250416` for the rule proposal agent. We synthesize each unoptimized design, sweeping clock frequencies from 250 MHz to 5 GHz, and select the target frequency at which area begins to climb sharply to meet tighter timing; these values are listed in Table III.

[1]https://github.com/egraphs-good/extraction-gym

[2]https://platform.openai.com/docs/models

TABLE III

BENCHMARKS AND THEIR TARGET CLOCK FREQUENCIES.

Benchmarks	**Description**	**Target Clock Freq.**
FIR	Finite Impulse Response filter	1 GHz
DCT	8-point Discrete Cosine Transform	1 GHz
Polynomial	4-degree polynomial evaluator	400 MHz
Watermark	Bitplane image encryption	2 GHz
MCM(3, 7, 21)	Multiple constant multiplication 3, 7, 21	2 GHz
MCM(7, 19, 31)	Multiple constant multiplication 7, 19, 31	2 GHz
MCM(5, 93)	Multiple constant multiplication 5, 93	2 GHz

B. End-to-end Evaluation

Table II presents an end-to-end comparison of ASPEN against two baselines, ROVER and a Commercial Synthesis Tool, across a diverse set of benchmarks, reporting absolute area and delay values, as well as percentage improvements. The commercial synthesis baseline refers to the base area and delay achieved by a commercial synthesis tool, serving as a reference to understand the effectiveness of PPA optimization during synthesis and technology mapping. ROVER denotes our reproduction using the rewrite rules in Table I and fixed

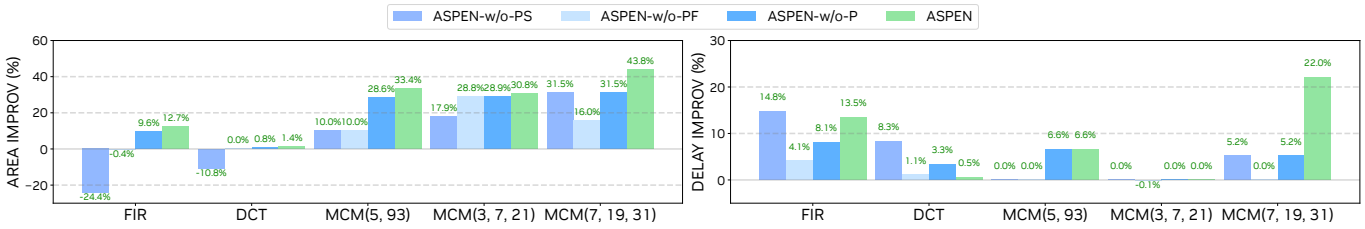


Fig. 6. **Ablation study** — Comparison of area and delay improvements (%) over the commercial baseline across different variants of ASPEN. **ASPEN** is the full-featured system with rule proposal, rule selection, and PPA feedback. **ASPEN-w/o-PS** removes rule proposal and selection, using initial rules and random rule selection. **ASPEN-w/o-PF** removes rule proposal and PPA feedback, using initial rules and fixed e-node costs. **ASPEN-w/o-P** removes rule proposal but retains agent-based selection and PPA feedback.

e-node costs. ASPEN results are the best (minimal) area and delay from the Pareto frontier. ASPEN outperforms both baselines: up to 43.76% area and 31.62% delay reductions on MCM(7,19,31) and 33.43% area and 16.99% delay on MCM(5,93). It delivers a 13.86% delay gain on FIR and matches or slightly exceeds ROVER on Watermark and Polynomial. On average, ASPEN cuts area and delay by 24.23% and 12.43% versus commercial tools, and by 16.51% and 6.65% versus ROVER. By combining targeted rewrites with equality saturation, ASPEN overcomes phase-ordering limits to uncover richer transformations and superior PPA improvements.

C. Case Study: Proposed and Selected Rewrite Rules

The rules proposed by the agent that pass equivalence checking are listed in Appendix A. They fall into two categories: (1) design-specific rules (e.g., multiply-by-12, -41, -21 for FIR; -93, -5 for MCM(5,93)) that limit e-node growth vs. stepwise unfolding; and (2) deeper-level rewrites (e.g., three-level nested multiplier factoring in DCT) that extend existing rules for more aggressive optimizations.

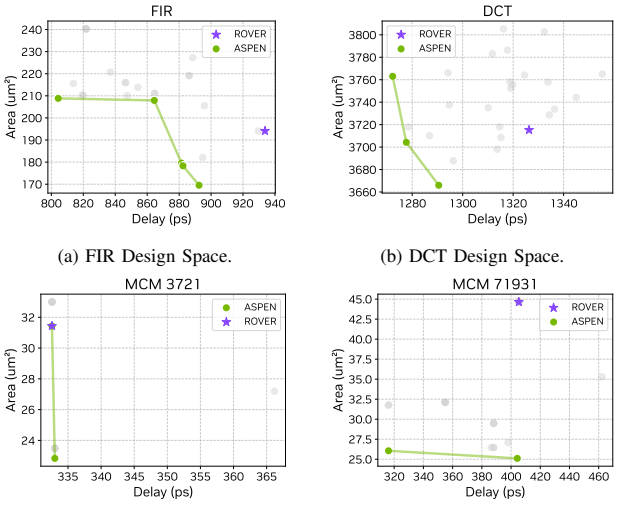
Selected critical rules are in Appendix B. A core set of algebraic laws applies across all benchmarks, augmented by a few custom rewrites (e.g., constant-coefficient multipliers for FIR and nested factorizations for DCT). This mix of general and targeted rules highlights the agent's balance of broad applicability and specialization, driving efficient e-graph exploration and high-quality extraction.

D. Case Study: E-Node Cost Learning via PPA Feedback

Figure 5 illustrates how ASPEN iteratively refines e-node costs within the saturated e-graph of a FIR design using feedback from PPA analysis. Initially, all e-nodes are assigned a uniform cost, resulting in a uniform visual size across the graph (Fig. 5a). After several iterations, the cost model incorporates feedback indicating that `MUL` operations are more expensive, which is visualized as an increase in the size of all `MUL` e-nodes (Fig. 5b). As iterations progress, ASPEN further adjusts the cost model based on PPA feedback, this time increasing the cost of `ADD` nodes—especially those closer to the leaf level of the graph (Fig. 5c). This example highlights ASPEN's ability to dynamically adjust the internal cost model according to PPA results.

E. Case Study: Pareto-Optimal Designs

Figure 7 shows design space exploration for the FIR, DCT, MCM(3, 7, 21), and MCM(7, 19, 31) benchmarks. ASPEN evaluates a wide range of candidate designs (grey dots) and successfully identifies a Pareto frontier capturing the area-delay trade-off (green line). In contrast, ROVER yields only a single design point (purple star), which lies on the Pareto frontier for MCM(3, 7, 21) but is otherwise dominated by ASPEN's solutions. This highlights ASPEN's ability to provide a broader and more efficient set of design choices.



(a) FIR Design Space. (b) DCT Design Space.

(a) MCM(3, 7, 21) Design Space. (b) MCM(7, 19, 31) Design Space.

Fig. 7. **Design points explored by ASPEN for FIR, DCT, and MCMs** – grey points represent design candidates evaluated by ASPEN. ASPEN identifies a Pareto frontier, whereas ROVER produces only a single design point.

F. Ablation Study

The ablation results in Fig. 6 highlight the contribution of each component in ASPEN. Removing rule proposal (**ASPEN-w/o-P**) consistently reduces performance, particularly for MCM(7, 19, 31), where full ASPEN achieves a 43.8% area and 22.0% delay improvement, compared to 31.5% and 5.2% without proposal. Excluding PPA feedback (**ASPEN-w/o-PF**) leads to further degradation, especially on FIR and DCT, underscoring the importance of dynamic cost guidance. The weakest variant, **ASPEN-w/o-PS**, which lacks both proposal and learned selection, occasionally performs worse than the commercial baseline (e.g., –24.4% area on FIR). Overall, these results demonstrate that all three components are critical to ASPEN's effectiveness.

V. Conclusion

We introduced ASPEN, an agentic system that combines LLM-driven rule generation with e-graph rewriting and real PPA feedback from commercial EDA tools to automate equivalence-preserving datapath RTL optimization. By eliminating manual rule design and proxy metrics, ASPEN enables principled, practical exploration of the RTL design space.

ACKNOWLEDGEMENTS – This work is supported in part by NSF Awards #2019306, #2403135, CCF2403134, CCF2349670, CNS2349461, and CNS2229562.

REFERENCES

[1] R. Roy, J. Raiman, N. Kant, I. Elkin, R. Kirby, M. Siu, S. Oberman, S. Godil, and B. Catanzaro, "Prefixrl: Optimization of parallel prefix circuits using deep reinforcement learning," in *2021 58th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2021, pp. 853–858.

[2] E. Ustun, C. Yu, and Z. Zhang, "Equality saturation for datapath synthesis: A pathway to pareto optimality," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–2.

[3] S. Coward, T. Drane, and G. A. Constantinides, "Rover: Rtl optimization via verified e-graph rewriting," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[4] M. Liu, N. Pinckney, B. Khailany, and H. Ren, "Verilogeval: Evaluating large language models for verilog code generation," in *2023 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2023, pp. 1–8.

[5] Y. Lu, S. Liu, Q. Zhang, and Z. Xie, "Rtllm: An open-source benchmark for design rtl generation with large language model," in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*. IEEE, 2024, pp. 722–727.

[6] S. Thakur, B. Ahmad, Z. Fan, H. Pearce, B. Tan, R. Karri, B. Dolan-Gavitt, and S. Garg, "Benchmarking large language models for automated verilog rtl code generation," in *2023 Design, Automation & Test in Europe Conference & Exhibition (DATE)*. IEEE, 2023, pp. 1–6.

[7] S. Thakur, B. Ahmad, H. Pearce, B. Tan, B. Dolan-Gavitt, R. Karri, and S. Garg, "Verigen: A large language model for verilog code generation," *ACM Transactions on Design Automation of Electronic Systems*, vol. 29, no. 3, pp. 1–31, 2024.

[8] S. Liu, W. Fang, Y. Lu, Q. Zhang, H. Zhang, and Z. Xie, "Rtlcoder: Outperforming gpt-3.5 in design rtl generation with our open-source dataset and lightweight solution," in *2024 IEEE LLM Aided Design Workshop (LAD)*. IEEE, 2024, pp. 1–5.

[9] Y. Tsai, M. Liu, and H. Ren, "Rtlfixer: Automatically fixing rtl syntax errors with large language model," in *Proceedings of the 61st ACM/IEEE Design Automation Conference*, 2024, pp. 1–6.

[10] S. Thakur, J. Blocklove, H. Pearce, B. Tan, S. Garg, and R. Karri, "Autochip: Automating hdl generation using llm feedback," *arXiv preprint arXiv:2311.04887*, 2023.

[11] J. Hlavicka and P. Fiser, "Boom: A heuristic boolean minimizer," in *IEEE/ACM International Conference on Computer-Aided Design. ICCAD 2001. IEEE/ACM Digest of Technical Papers (Cat. No. 01CH37281)*. IEEE, 2001, pp. 439–442.

[12] E. L. Childerhose and J. Liu, "A heuristic approach to two-level boolean minimization derived from karnaugh mapping," *arXiv preprint arXiv:2008.09307*, 2020.

[13] M. Liu, D. Robinson, Y. Li, and C. Yu, "Maptune: Advancing asic technology mapping via reinforcement learning guided library tuning," *arXiv preprint arXiv:2407.18110*, 2024.

[14] M. Willsey, C. Nandi, Y. R. Wang, O. Flatt, Z. Tatlock, and P. Panchekha, "egg: Fast and extensible equality saturation," *Proceedings of the ACM on Programming Languages*, vol. 5, no. POPL, pp. 1–29, 2021.

[15] Y. Zhang, Y. R. Wang, O. Flatt, D. Cao, P. Zucker, E. Rosenthal, Z. Tatlock, and M. Willsey, "Better together: Unifying datalog and equality saturation," *Proceedings of the ACM on Programming Languages*, vol. 7, no. PLDI, pp. 468–492, 2023.

[16] Y. R. Wang, S. Hutchison, J. Leang, B. Howe, and D. Suciu, "Spores: Sum-product optimization via relational equality saturation for large-scale linear algebra," *arXiv preprint arXiv:2002.07951*, 2020.

[17] A. VanHattum, R. Nigam, V. T. Lee, J. Bornholt, and A. Sampson, "Vectorization for digital signal processors via equality saturation," in *Proceedings of the 26th ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2021, pp. 874–886.

[18] C. Chen, G. Hu, C. Yu, Y. Ma, and H. Zhang, "E-morphic: Scalable equality saturation for structural exploration in logic synthesis," *Proceedings of the 62nd ACM/IEEE Design Automation Conference (DAC)*, 2025.

[19] C. Chen, G. Hu, D. Zuo, C. Yu, Y. Ma, and H. Zhang, "E-syn: E-graph rewriting with technology-aware cost functions for logic synthesis," in *Proceedings of the 61st ACM/IEEE Design Automation Conference (DAC)*, 2024, pp. 1–6.

[20] J. Yin, Z. Song, C. Chen, Q. Hu, and C. Yu, "Boole: Exact symbolic reasoning via boolean equality saturation," *Proceedings of the 62nd ACM/IEEE Design Automation Conference (DAC)*, 2025.

[21] J. Cheng, S. Coward, L. Chelini, R. Barbalho, and T. Drane, "Seer: Super-optimization explorer for high-level synthesis using e-graph rewriting," in *Proceedings of the 29th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 2*, 2024, pp. 1029–1044.

[22] E. Ustun, I. San, J. Yin, C. Yu, and Z. Zhang, "Impress: Large integer multiplication expression rewriting for fpga hls," in *2022 IEEE 30th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2022, pp. 1–10.

[23] J. Yin, Z. Song, N. B. Agostini, A. Tumeo, and C. Yu, "Hec: Equivalence verification checking for code transformation via equality saturation," *USENIX Annual Technical Conference (ATC)*, 2025.

[24] S. Coward, G. A. Constantinides, and T. Drane, "Automating constraint-aware datapath optimization using e-graphs," in *2023 60th ACM/IEEE Design Automation Conference (DAC)*. IEEE, 2023, pp. 1–6.

[25] S. Coward, T. Drane, and G. A. Constantinides, "Constraint-aware e-graph rewriting for hardware performance optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2024.

[26] Y. Cai, K. Yang, C. Deng, C. Yu, and Z. Zhang, "Smoothe: Differentiable e-graph extraction," in *Proceedings of the 30th ACM International Conference on Architectural Support for Programming Languages and Operating Systems, Volume 1*, 2025, pp. 1020–1034.

[27] H. Huang, Z. Lin, Z. Wang, X. Chen, K. Ding, and J. Zhao, "Towards llm-powered verilog rtl assistant: Self-verification and self-correction," *arXiv preprint arXiv:2406.00115*, 2024.

[28] M. DeLorenzo, A. B. Chowdhury, V. Gohil, S. Thakur, R. Karri, S. Garg, and J. Rajendran, "Make every move count: Llm-based high-quality rtl code generation using mcts," *arXiv preprint arXiv:2402.03289*, 2024.

[29] S. Liu, Y. Lu, W. Fang, M. Li, and Z. Xie, "Openllm-rtl: Open dataset and benchmark for llm-aided design rtl generation," in *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, 2024, pp. 1–9.

[30] M. Akyash, K. Azar, and H. Kamali, "Rtl++: Graph-enhanced llm for rtl code generation," *arXiv preprint arXiv:2505.13479*, 2025.

[31] A. Wei, H. Tan, T. Suresh, D. Mendoza, T. S. Teixeira, K. Wang, C. Trippel, and A. Aiken, "Vericoder: Enhancing llm-based rtl code generation through functional correctness validation," *arXiv preprint arXiv:2504.15659*, 2025.

[32] S. Sandal and I. Akturk, "Zero-shot rtl code generation with attention sink augmented large language models," *arXiv preprint arXiv:2401.08683*, 2024.

[33] Y. Wang, W. Ye, P. Guo, Y. He, Z. Wang, B. Tian, S. He, G. Sun, Z. Shen, S. Chen *et al.*, "Symrtlo: Enhancing rtl code optimization with llms and neuron-inspired symbolic reasoning," *arXiv preprint arXiv:2504.10369*, 2025.

[34] H. Sami, P.-E. Gaillardon, V. Tenace *et al.*, "Eda-aware rtl generation with large language models," *arXiv preprint arXiv:2412.04485*, 2024.

[35] V. K. Gupta, A. Yadav, M. Fujita, and B. Kumar, "Llm-aided front-end design framework for early development of verified rtls," in *2024 IEEE 33rd Asian Test Symposium (ATS)*. IEEE, 2024, pp. 1–6.

[36] Y. Zhao, H. Zhang, H. Huang, Z. Yu, and J. Zhao, "Mage: A multi-agent engine for automated rtl code generation," *arXiv preprint arXiv:2412.07822*, 2024.

[37] Y. Hu, J. Ye, K. Xu, J. Sun, S. Zhang, X. Jiao, D. Pan, J. Zhou, N. Wang, W. Shan *et al.*, "Uvllm: An automated universal rtl verification framework using llms," *arXiv preprint arXiv:2411.16238*, 2024.

[38] S. Ranga, R. Mao, D. Bhattacharjee, E. Cambria, and A. Chattopadhyay, "Rtl agent: An agent-based approach for functionally correct hdl generation via llms," in *2024 IEEE 33rd Asian Test Symposium (ATS)*. IEEE, 2024, pp. 1–6.

[39] L. T. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "ASAP7: A 7-nm FinFET Predictive Process Design Kit," *Microelectronics Journal*, vol. 53, pp. 105–115, Jul. 2016.

TABLE IV
REWRITE RULES PROPOSED BY AGENT THAT PASSED VERIFICATION

Design Name	Class	Rewrite Rules
FIR	Double add to shift	$x + x \Rightarrow x \ll 1$
	Multiply by 12	$12 \cdot x \Rightarrow (x \ll 3) + (x \ll 2)$
	Factor common shift	$(a \ll c) + (b \ll c) \Rightarrow (a + b) \ll c$
	Multiply by 41	$41 \cdot x \Rightarrow ((x \ll 5) + (x \ll 3)) + x$
	Multiply by 21	$21 \cdot x \Rightarrow ((x \ll 4) + (x \ll 2)) + x$
DCT	Even constant multiply via shift	$c \cdot x \Rightarrow ((c \gg 1) \cdot x) \ll 1$ if $c \neq 0 \wedge (c \ \& \ 1) = 0$
	Factor common shift	$(a \ll c) + (b \ll c) \Rightarrow (a + b) \ll c$
	Merge constant multipliers	$(c_1 \cdot x) + (c_2 \cdot x) \Rightarrow (c_1 + c_2) \cdot x$
	Factor nested multipliers	$(a \cdot b) + ((a \cdot c) + e) \Rightarrow (a \cdot (b + c)) + e$
	Factor common constant multiplier	$(c \cdot x) + (c \cdot y) \Rightarrow c \cdot (x + y)$
Polynomial	Merge nested constant multipliers	$(c_1 \cdot (c_2 \cdot x)) \Rightarrow (c_1 \cdot c_2) \cdot x$
	Quartic reassociation	$((x \cdot x) \cdot x) \cdot x \Rightarrow (x \cdot x) \cdot (x \cdot x)$
	Move shift outside multiplication	$(x \ll c) \cdot y \Rightarrow (x \cdot y) \ll c$
	Distributive law (right)	$(a \cdot c) + (b \cdot c) \Rightarrow (a + b) \cdot c$
	Move shift from right operand	$a \cdot (b \ll c) \Rightarrow (a \cdot b) \ll c$
Watermark	Factor common shift from add	$(a \ll n) + (b \ll n) \Rightarrow (a + b) \ll n$
	Multiply boolean to MUX	$a \cdot (b \wedge 1) \Rightarrow \text{MUXAR}(b \wedge 1, a, 0)$
	Convert and-or to MUX	$(a \wedge (\neg s)) \vee (b \wedge s) \Rightarrow \text{MUXAR}(s, b, a)$
	Simplify redundant and-or	$(a \wedge b) \vee (a \wedge (\neg b)) \Rightarrow a$
	Symmetric MUXAR	$(a \cdot (\neg b)) + (c \cdot b) \Rightarrow \text{MUXAR}(b, c, a)$
MCM(3, 7, 21)	Constant shift then multiply	$(c \ll k) \cdot x \Rightarrow (c \cdot x) \ll k$
	Optimized multiply by 21	$21 \cdot x \Rightarrow (x \ll 4) + ((x \ll 2) + x)$
	Multiply 7 via 3	$7 \cdot x \Rightarrow ((3 \cdot x) \ll 1) + x$
	Multiply 21 via 7	$21 \cdot x \Rightarrow ((7 \cdot x) \ll 1) + (7 \cdot x)$
	Merge shifts	$(x \ll c) \ll k \Rightarrow x \ll (c + k)$
MCM(7, 19, 31)	Alternate $2^n - 1$ multiply form	$c \cdot x \Rightarrow (x \ll \log_2(c + 1)) - x$ if $c \neq 0 \wedge c \neq 1 \wedge ((c + 1) \ \& \ c) = 0$
	Multiply by 19	$19 \cdot x \Rightarrow (x \ll 4) + ((x \ll 1) + x)$
	Multiply by 7	$7 \cdot x \Rightarrow (x \ll 3) - x$
	Double add to shift	$a + a \Rightarrow a \ll 1$
	Multiply by 31	$31 \cdot x \Rightarrow (x \ll 5) - x$
MCM(5, 93)	Push shift into constant	$(k \ll 1) \cdot x \Rightarrow (k \cdot x) \ll 1$
	Multiply by 93	$93 \cdot x \Rightarrow ((((x \ll 6) + (x \ll 4)) + (x \ll 3)) + (x \ll 2)) + x$
	Multiply by 5	$5 \cdot x \Rightarrow (x \ll 2) + x$

APPENDIX A
AGENT-GENERATED REWRITE RULES

Table IV presents the set of rewrite rules proposed by our agent that successfully passed equivalence verification across different RTL designs. The rules span a wide range of algebraic and bitwise optimizations, including constant multiplication decomposition, reassociation, common subexpression factoring, and conditional expression restructuring. These rules reflect a blend of human-like insight (e.g., factoring shared shifts or constants) and less intuitive but correct transformations (e.g., alternate forms of multiplying by constants such as $2^n - 1$). Notably, the agent consistently discovered canonical shift-add forms for constant multiplication in both general datapaths (e.g., `FIR`, `Polynomial`) and structured domains like multiple constant multiplication (MCM). For instance, in the `MCM(7, 19, 31)` design, the agent proposed five distinct rules capturing optimized decompositions of constants into shift and add/subtract forms. Across all benchmarks, the agent demonstrated an ability to generate reusable, design-specialized rules, several of which generalize common hardware design idioms. The diversity and correctness of these rewrites demonstrate the system's capability to automate high-quality rule discovery that would otherwise require manual design effort.

TABLE V
AGENT-SELECTED REWRITE RULES FOR PARETO-OPTIMAL DESIGN POINTS FOR EACH DESIGN

Rule Class	**FIR**	**DCT**	**Polynomial**	**Watermark**	**MCM(3, 7, 21)**	**MCM(7, 19, 31)**	**MCM(5, 93)**
Associativity	✓	✓		✓	✓	✓	✓
Commutativity	✓	✓		✓	✓	✓	✓
Constant shift then multiply					✓		
Distributivity	✓	✓	✓	✓	✓	✓	✓
Even Constant Multiplication	✓	✓	✓	✓	✓	✓	✓
Identity	✓	✓	✓	✓	✓	✓	✓
Optimized multiply by 21					✓		
Odd Constant Multiplication	✓	✓	✓	✓	✓	✓	✓
Power-of-Two Multiplication	✓	✓	✓	✓	✓	✓	✓
Merge shifts					✓		✓
Shift Simplification	✓	✓	✓	✓	✓	✓	✓
Merge nested constant multipliers			✓				
Quartic reassociation			✓				
Distributive law (right)			✓				
Estrin's Transformation			✓				
Move shift outside multiplication			✓				
Even constant multiply via shift		✓					
Merge constant multipliers		✓					
Factor nested multipliers		✓					
Factor common constant multiplier		✓					
Factor common shift	✓	✓		✓			
Convert and-or to MUXAR				✓			
Simplify redundant and-or				✓			
MUXAR Simplification				✓			
Multiply boolean to MUX				✓			
Symmetric MUXAR				✓			
Alternate 2^n-1 multiply form						✓	
Multiply 21 via 7					✓		
Multiply 7 via 3					✓		
Multiply by 19						✓	
Multiply by 7						✓	
Multiply by 31						✓	
Multiply by 12	✓						
Multiply by 21	✓						
Multiply by 41	✓						
Multiply by 5							✓
Multiply by 93							✓
Push shift into constant							✓

APPENDIX B
AGENT-SELECTED REWRITE RULES

In Table V, we report the critical set of rewrite rules that the agent chose for each design to reach Pareto-optimal frontier. The algebraic laws, namely distributivity, even and odd constant multiplication, identity elimination, power-of-two multiplication, and shift simplification, are universally beneficial and therefore selected across all seven benchmarks. Associativity and commutativity likewise appear in every design except the polynomial kernel, reflecting their broad utility in rearranging operations without changing semantics. Beyond these general rules, each design exhibits a handful of bespoke transformations. For FIR, the agent specializes in constant-coefficient multipliers ($\times$12, $\times$21, $\times$41), while the MCM instances (e.g., MCM(3,7,21) and MCM(7,19,31)) employ targeted rules such as "constant shift then multiply," "optimized multiply by 21," and chain-factoring multipliers (e.g., multiply 21 via 7 and multiply 7 via 3). The DCT design leverages deeper-level factorizations—quartic reassociation and nesting constant multipliers—whereas the watermark benchmark relies on Boolean-to-MUX conversions (e.g., Convert and–or to MUXAR) to simplify logical expressions. This pattern—core algebraic laws augmented by a small number of design-specific rewrites—demonstrates how our agent balances generality and specialization to drive efficient e-graph growth and high-quality extraction.