AI-based RTL (Register Transfer Level) Code Analysis and Optimization

1st Yuming Wang
Hochschule Hamm-Lippstadt
Lippstadt, Germany
https://orcid.org/0009-0003-3313-0485

***Abstract*—In today's technology-driven world, chips have become the fundamental building blocks of almost every electronic system — from mobile devices and automobiles to high-performance data centers and AI accelerators. As the complexity of modern integrated circuits continues to grow, chip design has evolved into an increasingly crucial and challenging discipline. Designers must not only ensure functional correctness but also strive for maximum performance, minimal power consumption, and optimal area utilization. Among various stages of the chip design process, Register Transfer Level (RTL) design plays a crucial role as it bridges high-level architectural concepts and low-level hardware implementation. Optimization at the RTL stage can significantly impact the overall performance and efficiency of the final chip. Therefore, achieving effective RTL optimization has become one of the core objectives and toughest challenges in today's semiconductor industry.**

***Index Terms*—RTL-Design, AI, Optimization**

I. Introduction

In modern digital circuit design, RTL design is one of the most essential phases. At this abstraction level, designers describe how data is transferred between registers and how logical operations are performed on that data using hardware description languages such as Verilog or VHDL. RTL design serves as the bridge between high-level architecture and gate-level implementation, defining both the functional behavior and the potential performance characteristics of the final chip[1] As the system complexity increases, achieving optimized timing,power, and performance behaviors has become a challenging goal in RTL design. Traditional RTL optimization methods such as logic restructuring[2] usually depend on the engineer's experience and are very time-consuming. In recent years, artificial intelligence (AI) and machine learning (ML) have drawn people's attention when it comes to the RTL-level optimization.[3] This paper provides an overview of optimization approaches in RTL design, focusing on both traditional and AI-driven methods. It highlights key optimization targets such as timing, area, and power, and discusses how machine learning enables faster and more intelligent RTL optimization. The remainder of this paper is organized as follows: Section II outlines the key optimization targets in RTL design, including timing, area, and power considerations. Section III discusses traditional RTL optimization techniques that have been widely used in industry. Section IV explores recent advancements in artificial intelligence (AI) and machine learning (ML) applications for RTL code analysis and optimization. Section V provides an overview of available tools that support both conventional and AI-assisted optimization workflows. Finally, Section VI concludes the paper by summarizing the key findings and discussing the benefits and challenges of integrating AI into RTL design optimization.

II. RTL optimization targets

Optimization at the RTL is aimed at achieving efficient digital circuit design. At this stage, design structures strongly affect Power, Performance(timing), and Area (PPA) objectives of electronic design automation [4]. Performance determines achievable highest frequency, area reflects silicon resource utilization, and power defines the energy efficiency. As noted by [5], optimizing PPA early at the RTL stage allows designers to explore trade-offs effectively and avoid costly late-stage iterations. Consequently, timing, area, and power form the foundation of all RTL optimization efforts. In this case, PPA are known as three primary goals for RTL opmization.

III. Traditional RTL Optimization Techniques

Many optimization techniques have been proposed to realize the PPA goals of chip design.

A. Hierarchical Flattening Optimization

In [6], a structural optimization technique called **hierarchical flattening** is introduced to enhance the optimization process in RTL design. Compared with the traditional hierarchical design, hierarchical flattening allows the synthesis tool to optimize across module boundaries, meaning that the original submodules are rebuilt into a series of sub-logic modules. One simple example of this technique can be seen in Fig. 1. In this example, the original hierarchical design of the *Top*, *ALU*, and *Memory_Controller* modules is flattened into corresponding logic modules, making further optimization steps more efficient.

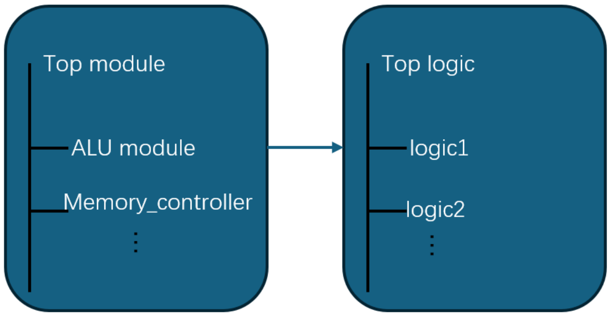


Fig. 1. Transfer from hierarchical design to flattened design

S. Anu *et al.* analyzed the impact of hierarchical flattening on FPGA-based automotive Electronic Control Units (ECUs) using Xilinx Vivado. By comparing hierarchical and flattened versions of the same design, the authors demonstrated that full flattening improved timing performance by up to 25%, reduced area utilization by 15%, and lowered total power consumption by approximately 10%. However, they also noted that as design complexity increases, debugging and functional verification become more challenging, since behavioral hierarchies are transformed into low-level logic modules.

B. *CDC-Aware RTL Synthesis Optimization*

Compared to the hierarchical flattening technique, which removes module boundaries to perform global logic optimization, the **Clock Domain Crossing (CDC)-aware RTL synthesis** method focuses on safe structural optimization under multi-clock conditions. While flattening can significantly improve timing and area by enabling cross-boundary optimizations, it may inadvertently alter synchronization structures between different clock domains, causing metastability or timing violations.

To address this, Li [7] introduced a CDC-aware synthesis flow using the Synopsys Fusion Compiler. The flow begins by analyzing the RTL with Synopsys SpyGlass to identify all clock domain crossings and verify proper synchronization. Based on the user-defined SDC file, SpyGlass generates a binary CDC restriction file that records protected synchronization structures. This file is then imported into Synopsys Fusion Compiler, ensuring that flattening, ungrouping, and retiming are disabled for CDC regions while optimization proceeds elsewhere. As a result, the synthesis achieves improved timing and area efficiency without introducing CDC violations or breaking synchronization logic. The flow chart of this method is shown in Fig. 2.

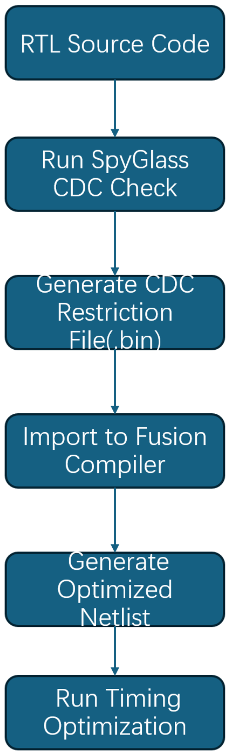

RTL Source Code
Run SpyGlass CDC Check
Generate CDC Restriction File(.bin)
Import to Fusion Compiler
Generate Optimized Netlist
Run Timing Optimization

Fig. 2. Flow chart of the CDC-aware synthesis method

By applying the CDC-aware synthesis flow to an arithmetic block in a commercial display chip, the authors achieved notable improvements in design quality. The total cell area was reduced from 4.92×10^6 μm^2 to 4.68×10^6 μm^2, and the placement density decreased from 75.9% to 62.3%, easing routing congestion. The worst negative slack (WNS) improved to less than 100 ps, with total timing violations reduced to fewer than one thousand paths. These results demonstrate that the proposed flow enables aggressive optimization while maintaining CDC safety and design reliability.

IV. AI and ML Applications for RTL Optimizations

As the optimization of the RTL design is becoming more and more crucial, researchers start exploring more and more solutions toward achieving PPA goals. However, traditional RTL optimization techniques are usually very time-consuming and highly rely on full-fledged commercial electronic design automation tools. Recently, the investigation on ML-based prediction toward RTL optimization has attracted significant attention. By learning the relationship between RTL structures and post-synthesis performance, machine learning models can provide fast and accurate estimation of PPA aspects, enabling early design guidance.

A. *Path-Level Timing Prediction using ML-based Framework*

For timing prediction at the path level, Sengupta et al. propose a machine learning-based framework for early timing analysis at the RTL stage [3]. In chip design, it is very crucial to identify timing-sensitive components at the early stage to prevent timing issues and improve the performance of the designed chips. For this goal, the authors introduce a machine-learning-based method that can directly predict timing path delays at a granular level directly from RTL design.

The technique proposed depends on two main components: the **Abstract Syntax Tree (AST)** and the **XGBoost model**. AST is a structured, hierarchical representation of source code that captures the syntactic and semantic relationships between operations, variables, and expressions. In RTL analysis, the AST helps transform Verilog or VHDL code into a machine-readable graph form, enabling automated feature extraction for timing or power modeling. XGBoost is a powerful machine learning algorithm based on gradient-boosted decision trees. It iteratively builds an ensemble of trees to minimize prediction error, offering high accuracy, fast training speed, and robustness to noise—making it well suited for modeling complex RTL-to-delay relationships.

The process of training the prediction model can be seen in Fig. 3. The workflow begins with a collection of benchmark RTL designs written in Verilog. Each RTL source is first parsed using Verilator, which converts the code into an AST representing the syntactic and structural hierarchy of the design. The AST is then analyzed to extract signal-level relationships, forming a design graph where nodes correspond to registers, logic operations, and module connections. From this graph, the framework automatically extracts multiple register-to-register, input-to-register, and register-to-output paths, and computes a set of path-level features such as logic depth, number of arithmetic and logical operations, bit widths, and fan-in/fan-out information. These feature vectors serve as the input samples for the machine learning model.

In parallel, the same RTL designs are synthesized, placed, and routed using commercial EDA tools to obtain accurate post-placement path delay values, which serve as the ground truth labels. Finally, these feature–label pairs are used to train an XGBoost classifier that learns the correlation between RTL-level structural features and actual timing delays. The trained model is then capable of predicting the delay class of unseen RTL paths without requiring time-consuming synthesis or layout.

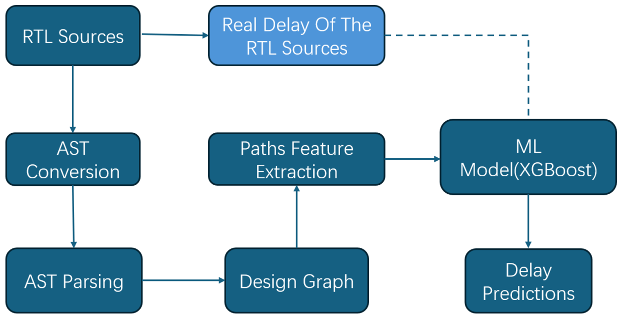


Fig. 3. ML Model Training Flowchart

The proposed framework was evaluated on 26 open-source Verilog designs from IWLS 2005 and OpenCores benchmarks. Each design was synthesized and analyzed using commercial EDA tools to obtain post-placement timing data as ground truth. The trained XGBoost model achieved an average classification accuracy of 91% and an F1-score of 87%, while reducing analysis runtime by nearly 40× compared to traditional static timing analysis, effectively identifying timing-critical RTL paths with high reliability.

B. Design-Level PPA Prediction using MasterRTL Framework

Another AI-based optimization technique called the **MasterRTL framework** was proposed in [5]. It works in a different way compared to the hierarchical flattening method, which removes module boundaries to allow cross-module logic optimization during synthesis. MasterRTL focuses on pre-synthesis prediction and guidance rather than direct structural transformation. Traditional optimization methods are usually implemented after the synthesis or place-and-route process, whereas the proposed MasterRTL focuses on RTL-stage prediction toward PPA goals.

The framework mainly addresses two key questions: (1) how to represent RTL in a machine-learning-friendly format, and (2) how to capture the essential features to accurately estimate timing, power, and area across different RTL designs. For question (1), the authors propose a new data format called the **Simple Operator Graph (SOG)**. SOG is a bit-level RTL representation proposed in the MasterRTL framework. It decomposes all multi-bit operations into fundamental single-bit logic operators (AND, OR, XOR, NOT, and MUX) and represents the design as a directed acyclic graph. Compared with AST-based representations, SOG provides a more unified and ML-friendly format that closely resembles gate-level netlists, enabling accurate cross-design learning and efficient PPA prediction.

For question (2), MasterRTL customizes different learning models for each PPA goal, since the underlying mechanisms behind these objectives are fundamentally different. For timing estimation, the framework extracts register-to-register paths from the SOG and learns to predict critical path delays. For power prediction, it incorporates toggle rate information to model both vector-based and vector-less power. For area estimation, it counts logic and register nodes to approximate the total gate area. These customized models collectively enable MasterRTL to achieve accurate cross-design PPA prediction without invoking time-consuming synthesis or place-and-route processes.

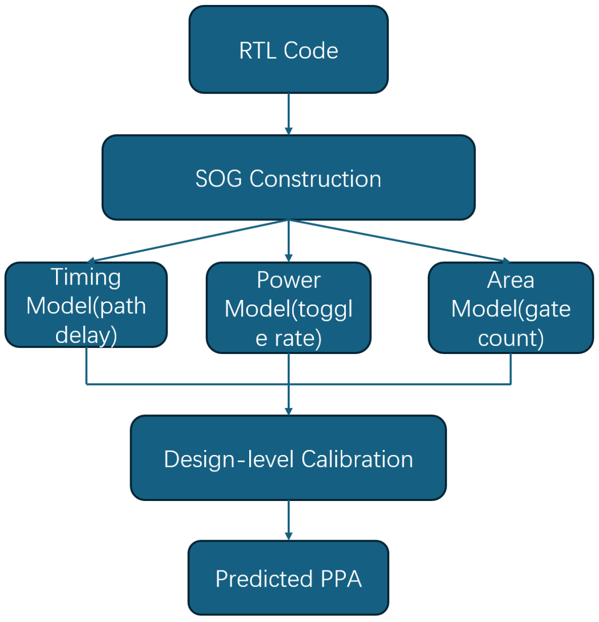


Fig. 4. MasterRTL framework flowchart

The MasterRTL framework takes RTL code as input and converts it into a bit-level representation called the Simple Operator Graph (SOG). Based on the SOG, three separate machine learning models are developed to estimate timing, power, and area using features such as path delay, toggle rate, and gate count. Their outputs are then calibrated at the design level to generate accurate pre-synthesis PPA predictions.

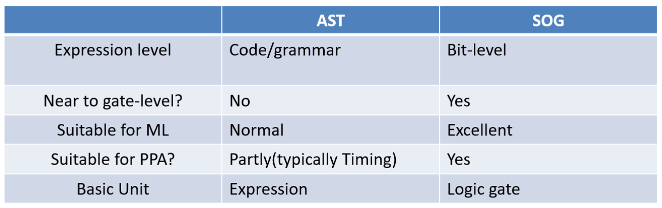
Finally, the MasterRTL framework was evaluated on 90 RTL designs synthesized with Synopsys Design Compiler using the NanGate 45 nm library. Compared with previous ML-based RTL PPA estimation approaches that rely on AST representations or random path sampling, MasterRTL achieved much higher accuracy, with correlation R values of 0.96 for TNS, 0.93 for WNS, 0.89 for power, and 0.98 for area. The results demonstrate that the SOG-based representation and customized models enable accurate and efficient pre-synthesis PPA prediction, requiring only about 5% of synthesis runtime.

C. *Description about AST and SOG*

AST is more a RTL representation based on expression, on the other hand, SOG is more a bit-level RTL representation, it directly shows the gate-level expression of RTL code. For Machine Learning models, SOG is more suitable and can conduct more dimensions estimation in Power,Performance,and Area. A figure showing the comparison can be seen in the following figure.

V. Available Tools

To realize the optimization goals of the RTL design, there are many existed EDA tools available already, in this section, some Electronic Design Automation (EDA) tools are introduced.



	AST	SOG
Expression level	Code/grammar	Bit-level
Near to gate-level?	No	Yes
Suitable for ML	Normal	Excellent
Suitable for PPA?	Partly(typically Timing)	Yes
Basic Unit	Expression	Logic gate

Fig. 5. Comparison Table for AST and SOG

A. *Commercial EDA Tools*

Commercial EDA tools play a crucial role in modern RTL design and optimization workflows. These tools provide robust capabilities for logic synthesis, simulation, static timing analysis, and verification, enabling designers to efficiently achieve power, performance, and area (PPA) objectives. Developed and maintained by leading semiconductor tool vendors such as **Synopsys**, **Cadence**, commercial EDA tools offer high accuracy, scalability, and integration across the entire design flow—from RTL coding to physical implementation.
Synopsys Design Compiler is a leading RTL synthesis tool designed by Synopsys[8] that performs concurrent optimization of timing, area, power, and test. It integrates topographical technology to predict post-layout timing and area within 10% accuracy, significantly reducing iterations between synthesis and physical implementation. The tool features scalable multi-core performance with up to 2× faster runtime and supports advanced design capabilities such as multi-voltage synthesis, congestion prediction, and cross-probing between RTL and timing reports. As the core of **Synopsys**' synthesis solution, **Design Compiler** works seamlessly with **Power Compiler**, **PrimeTime**, and **DFTMAX** to deliver high-quality, physically-aware synthesis results.
Cadence Genus Synthesis Solution is one advanced synthesis tool developed by **Cadence**[9]to address the increasing complexity of advanced-node SoC designs. Built on a massively parallel architecture, Genus provides 3–5× faster synthesis runtimes while maintaining scalability beyond 10 million instances. The tool achieves tight correlation with the **Cadence Innovus Implementation System**, ensuring timing and wirelength correlation within 5% and frequency correlation within 2%, thereby significantly reducing iterations between synthesis and place-and-route (P&R) stages. Genus introduces a proprietary **global analytical architecture optimization** algorithm that evaluates multiple microarchitectures simultaneously, achieving up to 20% datapath area reduction without performance degradation. Additionally, it supports **unit-level timing and physical context extraction**, allowing RTL designers to perform physically aware synthesis and reduce block-level iteration effort by over 2×. With unified engines for placement, routing, and timing signoff shared with **Innovus** and **Tempus**, Genus provides a highly correlated, high-capacity synthesis solution optimized for advanced SoC design productivity.

In summary, commercial EDA tools such as **Synopsys Design Compiler** and **Cadence Genus Synthesis Solution** represent the state-of-the-art in RTL synthesis and optimization. These tools combine advanced algorithms, parallel computing architectures, and tight integration with physical design environments to deliver highly accurate, timing- and power-aware results. Their physically correlated synthesis capabilities significantly shorten design iterations and improve design productivity for complex, advanced-node SoCs.

B. Open Source Tools

In addition to commercial EDA solutions, open-source tools have gained significant attention in both academia and industry for RTL design analysis and optimization. These tools provide cost-effective, transparent, and highly customizable alternatives for research and early-stage design exploration. Open-source frameworks such as **Yosys**, **Verilator**, and **OpenROAD** enable logic synthesis, simulation, timing analysis, and physical design implementation without relying on proprietary software, making them ideal for academic research, prototyping, and machine learning-based RTL studies. **yosys** is an open-source RTL synthesis framework that supports Verilog designs and provides basic logic optimization, technology mapping, and structural analysis capabilities[10].While it performs essential gate-level and Boolean optimizations, it lacks the timing-driven and power-aware optimization features of commercial tools such as Synopsys Design Compiler or Cadence Genus.

Verilator is an open-source SystemVerilog and Verilog simulation and analysis tool that can compile HDL code into systemC or C++ models for high-performance simulation. One special about this tool is it can be used for RTL structure analysis while cooperating with AI driven or machine-learning models. The output of this tool is AST and netlists information, which is very helpful for the training of machine-learning models.[11]

OpenROAD is a tool developed by **OpenROAD** is a tool developed by the University of California, San Diego, as part of the DARPA-funded IDEA program to enable fully autonomous digital design implementation from RTL to GDSII. It integrates multiple open-source engines for logic synthesis, placement, routing, and static timing analysis, forming a complete and physically aware design flow. The framework aims to eliminate manual intervention in the backend design process by providing automated timing closure, congestion optimization, and power reduction. According to the OpenROAD documentation[12], the tool achieves scalable performance on advanced-node designs through a multi-threaded architecture and shared optimization engines, contributing significantly to the open-source EDA ecosystem.

Compared to commercial EDA tools, which provide highly integrated, timing-driven, and power-aware optimization with advanced physical correlation, open-source tools mainly focus on functional correctness, structural analysis, and early-stage design exploration. While tools such as **Yosys**, **Verilator**, and **OpenROAD** offer transparency and extensibility suitable for research and prototyping, commercial solutions like **Synopsys Design Compiler** and **Cadence Genus** deliver superior PPA optimization, scalability, and signoff accuracy required for industrial-scale designs.

VI. Conclusion

This paper presented an overview of optimization techniques for Register Transfer Level (RTL) design, highlighting both traditional and AI-driven approaches.Traditional techniques such as **hierachical flattening** and **CDC-aware synthesis** focus on focus on structural and timing improvements through logic-level transformations, but they often require iterative synthesis and rely heavily on commercial EDA tools. In contrast, recent advances in artificial intelligence and machine learning, such as ML-based path delay prediction and the MasterRTL framework, enable early-stage performance, power, and area estimation directly from RTL code, significantly reducing design turnaround time. Furthermore, the paper reviewed both commercial and open-source EDA tools that support RTL optimization workflows. Commercial tools like **Synopsys Design Compiler** and **Cadence Genus Synthesis Solution** provide highly integrated, timing- and power-driven optimization with strong correlation to physical implementation, while open-source tools such as **Yosys**, **Verilator**, and **OpenROAD** offer transparent, flexible platforms for research and prototyping.

As AI and ML continue to evolve, their integration into EDA workflows is expected to accelerate the automation and intelligence of RTL optimization. Future research will focus on improving model generalization, combining AI predictions with physical-aware synthesis, and bridging the gap between early-stage RTL design and final silicon performance.

References

[1] S. Palnitkar, Verilog HDL: A Guide to Digital Design and Synthesis, 2nd ed. Upper Saddle River, NJ, USA: Prentice Hall, 2003.

[2] J. Wang, H. You, Z. Wang, M. Liu, Y. Su and Y. Zhang, ”Optimization of Redundant Logical Units in RTL Logic Synthesis,” 2023 International Symposium of Electronics Design Automation (ISEDA), Nanjing, China, 2023, pp. 137-141, doi: 10.1109/ISEDA59274.2023.10218581.

[3] P. Sengupta, A. Tyagi, Y. Chen and J. Hu, ”Early Identification of Timing Critical RTL Components using ML based Path Delay Prediction,” 2023 ACM/IEEE 5th Workshop on Machine Learning for CAD (MLCAD), Snowbird, UT, USA, 2023, pp. 1-6, doi: 10.1109/MLCAD58807.2023.10299879.

[4] A. B. Kahng, “Machine learning applications in physical design: Recent results and directions,” in Proc. Int. Symp. on Physical Design (ISPD), New York, NY, USA: ACM, 2018, pp. 68–73, doi: 10.1145/3177540.3177554.

[5] W. Fang et al., ”MasterRTL: A Pre-Synthesis PPA Estimation Framework for Any RTL Design,” 2023 IEEE/ACM International Conference on Computer Aided Design (ICCAD), San Francisco, CA, USA, 2023, pp. 1-9, doi: 10.1109/ICCAD57390.2023.10323951.

[6] P. Mahesh, M. S. Chinnapamthy, M. Parthiban, R. Rajarajavigraman, S. Akthar and S. Yeswanth, ”Optimizing RTL-Based FPGA Designs for ECU Performance Enhancement: Impact of Hierarchical Flattening Using Xilinx Vivado,” 2025 8th International Conference on Circuit, Power and Computing Technologies (ICCPCT), Kollam, India, 2025, pp. 142-147, doi: 10.1109/ICCPCT65132.2025.11176538.

[7] BolunLi, W. Cao and B. Wu, ”Clock Domain Crossing Aware RTL Synthesis using Fusion Compiler in Commercial Display Chip Design,” 2024 4th International Symposium on Artificial Intelligence and Intelligent Manufacturing (AIIM), Chengdu, China, 2024, pp. 523-526, doi: 10.1109/AIIM64537.2024.10934572.

[8] Synopsys, Inc., “Design Compiler: Industry Standard for RTL Synthesis,” [Online]. Available: https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/design-compiler.html

[9] Cadence Design Systems, Inc., “Genus Synthesis Solution Product Brief,” [Online]. Available: https://www.cadence.com/en_US/home/resources/product-briefs/genus-synthesis-solution-pb.html

[10] C. Wolf, “Yosys Open SYnthesis Suite,” YosysHQ, Documentation Version 0.33, 2025. [Online]. Available: https://yosyshq.readthedocs.io/_/downloads/yosys/en/0.33/pdf/

[11] W. Snyder,“Verilator:Open-Source SystemVerilog Simulator User Guide,” Veripool, [Online]. Available: https://www.veripool.org/ftp/verilator_doc.pdf

[12] OpenROAD Project, “OpenROAD Documentation: Foundations and Realization of Open, Accessible Design,” [Online]. Available: https://openroad-test.readthedocs.io/_/downloads/en/stable/pdf/