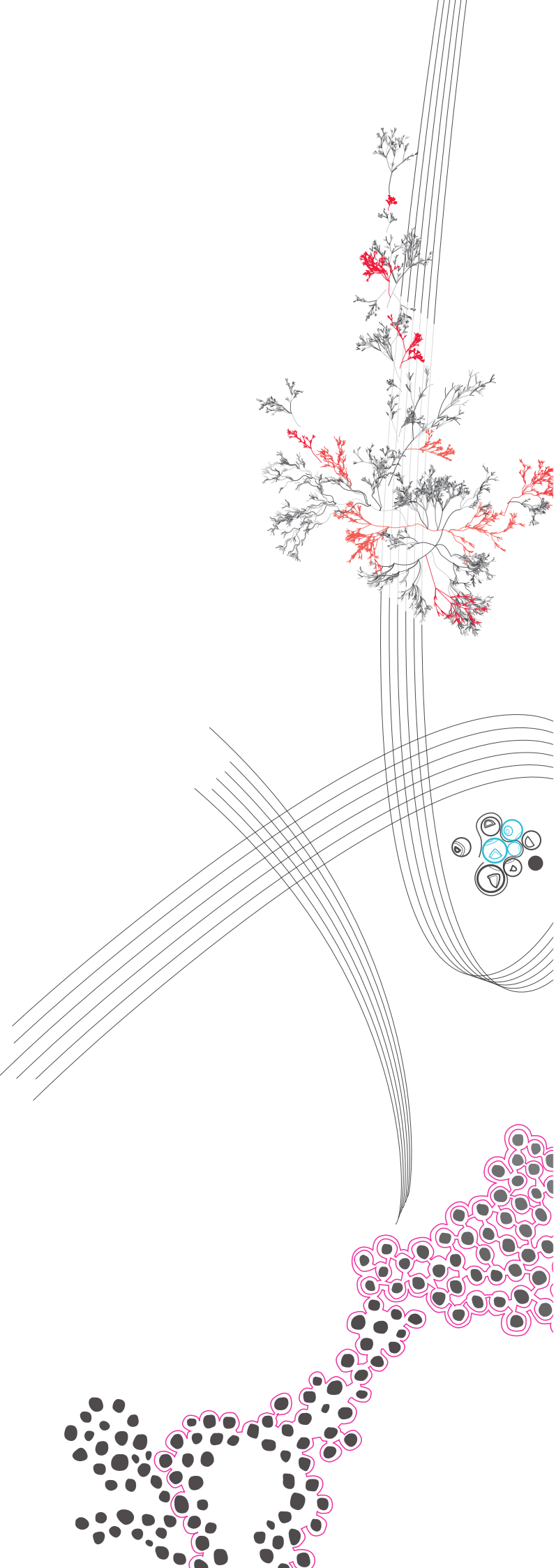
UNIVERSITY OF TWENTE.



How does post-synthesis timing analysis differ from post-layout timing analysis in a RISC-V processor implementation using open-source toolchains?

Bachelor Thesis

Barto Visser

Supervisors:
dr. ir. Marco Ottavi
dr. ir. Ronan van der Zee
Madiha Sheikh, MSc

Computer Architecture for Embedded Systems
Electrical Engineering
University of Twente
Netherlands
June 2025

Abstract

The semiconductor industry's shift towards Domain-Specific Architectures (DSAs), largely enabled by the open RISC-V instruction set architecture, has complicated the choice between mature, proprietary electronic design automation toolchains and emerging open-source alternatives. A critical challenge in any physical design flow is optimization of Performance, Power, and Area (PPA). This challenge is further complicated by the "predictive gap"–the discrepancy between post-synthesis estimates and final post-layout PPA results. This thesis investigates the significance of this gap in an open-source toolchain and compares synthesis quality when implementing an identical RISC-V System on a chip using the open-source Yosys and the proprietary Cadence Genus synthesis tools. Both flows targeted the IHP 130 nm PDK. Post-synthesis analysis revealed that the proprietary tool produced a more efficient netlist, at the cost of a longer runtime. Critically, post-synthesis timing proved to be an unreliable predictor of final performance in the open-source flow; a design that met timing after synthesis exhibited significant timing violations after physical implementation. This was accompanied by an increase in power and area due to buffer insertion for the clock tree and hold-time fixing. The results highlight the trade-offs between the rapid iteration speed of current open-source tools and the refined optimization of mature, proprietary ones, while underscoring that the predictive gap remains a significant challenge.

Contents

1 Introduction

The semiconductor industry is at a pivotal juncture. For half a century, its progress was reliably charted by Moore's Law and Dennard scaling, principles that guaranteed exponentially more powerful and efficient processors with each generation. The breakdown of these foundational laws due to physical limitations has forced a fundamental rethink of processor design. This introduction outlines this paradigm shift, details the critical implementation challenges that have emerged, and establishes the research objectives of this thesis, which investigates the predictability of modern design flows in this new landscape.

1.1 The End of an Era and the Rise of Specialization

The semiconductor industry is undergoing a foundational paradigm shift, as the two principles that governed its progress for half a century—Moore's Law and Dennard scaling—have reached their physical limits. For decades, Moore's Law described the exponential increase in transistor density on integrated circuits, serving as the primary driver of computational advancement. This was complemented by Dennard scaling, which states that as transistors shrink, their power density remains constant, yielding proportional gains in energy efficiency [1]. However, since the mid-2000s, physical constraints, such as rising sub-threshold leakage currents, have effectively ended Dennard scaling, even as transistor counts continue to grow. This has created a significant "power wall," capping the practical clock speeds of general-purpose processors and ending the era of automatic performance gains [2].

In response, the focus of processor innovation has pivoted from monolithic, general-purpose CPUs to a new era of Domain-Specific Architectures (DSAs). This shift, heralded as a "new golden age for computer architecture," emphasizes custom-engineered hardware accelerators tailored for specific workloads, such as artificial intelligence, network processing, or cryptography [3]. By aligning hardware structure with algorithmic requirements, DSAs can deliver orders-of-magnitude improvements in performance and energy efficiency for the specific tasks they are designed for, compared to their general-purpose counterparts.

1.2 The EDA Implementation Challenge

To realize these specialized designs, sophisticated Electronic Design Automation (EDA) toolchains are required. These complex software suites automate the process of transforming a human-written hardware description into a production-ready physical layout.

1.2.1 Open-source vs. Proprietary Toolchains

Engineers face a critical choice between two competing implementation philosophies: the established proprietary ecosystem and the emerging open-source movement. The proprietary path, dominated by vendors like Cadence and Synopsys, offers mature, silicon-proven EDA tools that are considered the industry standard for designing complex, high-performance systems on advanced technology nodes. In contrast, the open-source movement promotes accessibility and collaborative innovation. This ecosystem is anchored by the free and open RISC-V instruction set architecture, which allows for architectural customization and doesn't require any licensing fees [4]. This architectural freedom is now complemented by a complete, end-to-end open-source EDA toolchain, the OpenROAD Project, which has demonstrated fully autonomous, "no-human-in-the-loop" layout generation from register-transfer level (RTL) to final (GDSII) manufacturing files [5]. This open path significantly lowers the barrier to custom silicon development, fostering innovation across academia and startups.

1.2.2 Post-Synthesis vs. Post-Layout

Regardless of the chosen toolchain, the central engineering challenge is the multi-objective optimization of Power, Performance, and Area (PPA). To understand this challenge, it is essential to recognize the two fundamental stages of digital design implementation. The first is logic synthesis, a process where the high-level hardware description (RTL code) is translated into a "logical blueprint"—a gate-level netlist that specifies which standard logic cells are needed and how they are logically connected. The second stage is physical design, often called place-and-route (P&R), which takes this logical blueprint and creates a "physical blueprint." It determines the precise X-Y coordinates for every logic cell on the silicon die and then routes the millions of microscopic metal wires that physically connect them.

The critical difficulty in this two-stage process is the "predictive gap"—the often significant disagreement between PPA estimates made after synthesis and the final, measured results after the physical design is complete. A smaller predictive gap indicates a more reliable and efficient design flow, reducing the risk of costly late-stage redesigns caused by inaccurate estimations.

The core of this predictive gap lies in the fundamental differences between the analysis performed at each stage, as detailed in Table 1. Post-synthesis analysis provides the first PPA estimate. It is performed on the logical gate-level netlist, before the cells have been physically placed and routed. At this point, the actual length and complexity of the interconnecting wires are unknown. Consequently, the analysis must rely on statistical Wire Load Models (WLMs) or other estimations to approximate interconnect delay, which dominates modern designs [6]. While this process is computationally fast and suitable for early feasibility checks, its accuracy is inherently limited.

In contrast, post-layout analysis occurs after the full physical implementation, including placement, Clock Tree Synthesis (CTS), and routing. Its input is a physically-aware netlist enriched with detailed parasitic data extracted from the actual, routed layout presented in a SPEF file. This allows for the calculation of delays based on real RC (resistance-capacitance) values of the interconnects, providing high-quality results. This final verification is computationally intensive but essential, as it performs comprehensive setup and hold time checks across various Process, Voltage, and Temperature (PVT) corners to account for the true physical effects. The discrepancy between the fast, estimated results of the former and the slow, accurate results of the latter define the predictive gap this thesis investigates.

Table 1: Comparison of Post-Synthesis and Post-Layout Static Timing Analysis

Attribute	**Post-Synthesis Timing Analysis**	**Post-Layout Timing Analysis**
Design Stage	After logic synthesis, before physical placement & routing (P&R).	After P&R, including clock tree synthesis (CTS).
Input Data	Gate-level netlist, library timing models (.lib), estimated wire loads.	Placed/Routed netlist (DEF), library models (.lib), extracted parasitics (SPEF/SDF).
Delay Models	Wire Load Models (WLMs) or statistical estimates (no physical layout).	Based on RC parasitics extracted from actual physical layout.
Accuracy	Lower accuracy; primarily gate delays + estimated interconnect.	High accuracy; includes actual gate and interconnect delays.
Checks Performed	Primarily setup time; hold time checks less useful.	Comprehensive setup and hold time checks across corners.
Computational Cost	Relatively fast; suitable for iterative synthesis loops.	Slower, especially parasitic extraction; run less frequently.
Purpose	Early feasibility check, guide synthesis optimization, initial P&R constraints.	Final timing verification, determine max frequency.

1.3 The Croc SoC

To investigate the PPA predictive gap and the capabilities of open-source EDA tools, this thesis employs the Croc System on a Chip (SoC) as the design under test. This investigation was performed on the latest release of the SoC (version 1.1). The Croc SoC, a platform developed within the PULP (Parallel Ultra-Low Power) ecosystem, is based on the CV32E40P RISC-V core (an implementation of the Ibex core) and utilizes the Open Bus Interface (OBI) for on-chip communication [7].

The Croc SoC provides a platform for evaluating EDA flows. As illustrated in Figure 1, its architecture is divided into several key domains and components. The primary computational element is the ‘core_wrap‘ module, which instantiates the Ibex CV32E40P RISC-V processor core. This core interacts with memory and peripherals via an OBI Crossbar. The ‘croc_domain‘ encapsulates the core, the OBI crossbar, an OBI demultiplexer for peripheral access, dedicated SRAM (static random-access memory) memory banks, and essential peripherals including SoC control registers, a UART (universal asynchronous

receiver-transmitter), a timer unit, and GPIO (general-purpose input/output). Debug capabilities are provided through a JTAG interface connected to a debug module, which also interfaces with the OBI crossbar. A user domain is provisioned for potential custom logic integration, though this was not modified for the baseline experiments in this work.

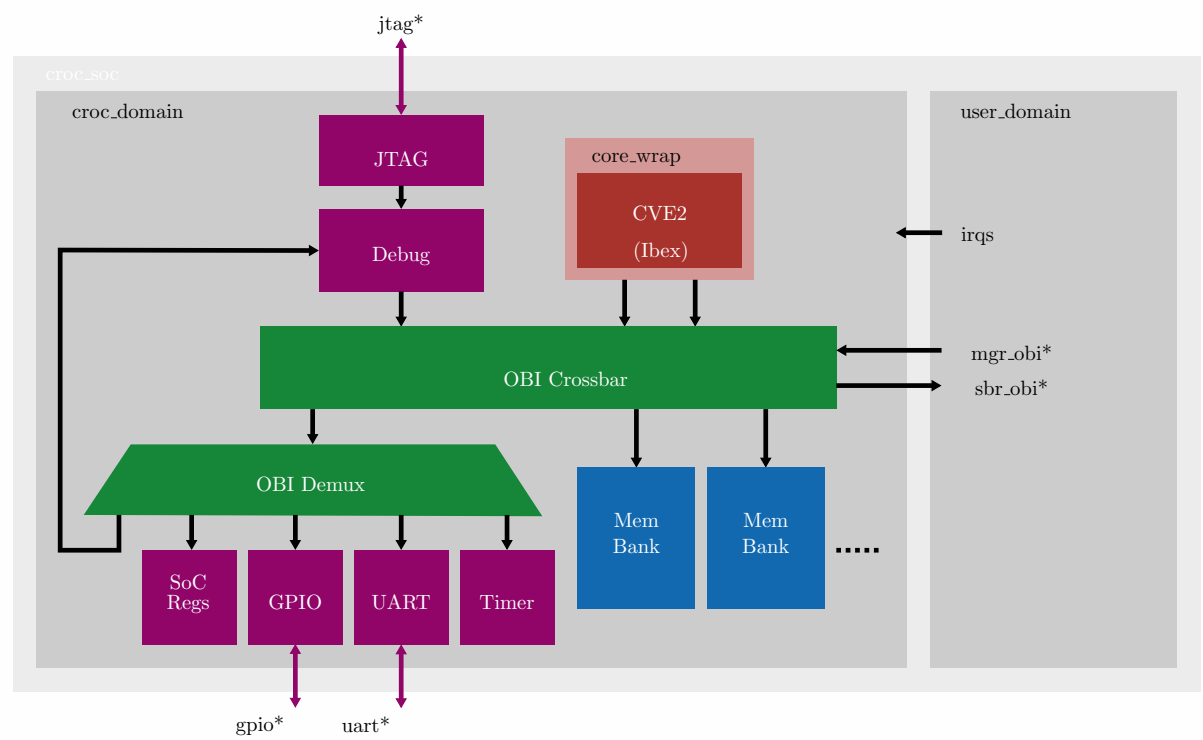


Figure 1: Croc SoC architecture [8]

1.4 Research Goal

This thesis presents an investigation of the predictive gap in an open-source flow and compares this to an industry standard, proprietary flow. It seeks to answer the following research questions:

- How does post-synthesis timing analysis differ from post-layout timing analysis in a RISC-V processor implementation using open-source toolchains?
- How does synthesis quality of the OpenROAD flow differ from the quality offered by an industry standard, proprietary flow?

To ensure a controlled and equitable comparison, the Croc SoC will serve as the design under test. This platform was selected as it is a complete, well-documented system representative of modern, open-source RISC-V-based designs, featuring a standard core, peripherals, and memory hierarchy. As it is the default for the Croc SoC, both implementation flows will target the same IHP 130 nm Process Design Kit (PDK). By analyzing the PPA reports from both flows, this work will provide a comprehensive comparison of not only the final achievable results but, more critically, the predictability of the journey to silicon, offering valuable insights for designers evaluating these two distinct paths.

2 Experimental Methodology

This section details the experimental methodology used to investigate the PPA predictive gap and compare the synthesis quality of open-source and proprietary EDA toolchains. It begins by introducing the two competing implementation flows at the heart of this study. It then outlines the experimental conditions, including the design under test, target technology, and Process, Voltage, and Temperature (PVT) corners. Finally, it describes the step-by-step procedures for design implementation and PPA analysis.

2.1 Introducing the Toolchains

The core of this investigation involves comparing two distinct EDA implementation philosophies, each represented by a specific toolchain. Both toolchains take the same RTL input and target the same IHP 130 nm PDK, allowing for a direct comparison of their optimization quality and predictive capabilities.

2.1.1 The Open-Source Toolchain

This path represents the emerging ecosystem focused on accessibility and collaborative development. For this work, the OpenROAD Project, comprised of multiple tools, is analyzed. It uses Yosys for synthesis, several other tools for physical design (place-and-route), and OpenSTA for Static Timing Analysis (STA). Yosys is a flexible synthesis suite that transforms RTL code into a technology-mapped gate-level netlist. OpenROAD then takes this netlist and performs the full physical implementation flow—from floorplanning to final routing, to produce a manufacturable layout. OpenSTA, a core component of OpenROAD, is used for timing verification. Together, they provide a complete, end-to-end, open-source RTL-to-GDSII flow.

2.1.2 The Proprietary Toolchain

This path represents the established, industry-standard approach, known for its maturity and silicon-proven results. For the scope of this thesis, which focuses heavily on the synthesis stage, the Cadence Genus Synthesis Solution is used. Genus is a leading commercial tool for logic synthesis and optimization, responsible for translating RTL into a highly optimized gate-level netlist based on PPA constraints. By comparing the netlist generated by Genus against the one from Yosys, this work can assess the relative synthesis quality.

2.2 Process, Voltage, and Temperature

To ensure a silicon design is robust, it must function correctly not just under ideal conditions, but across a range of possible manufacturing and operating scenarios. These operational extremes are known as PVT corners. The performance of transistors and interconnects varies significantly based on these three key factors:

- Process (P): Due to inconsistencies in fabrication, transistors on a wafer can be inherently fast (higher than normal carrier mobility) or slow (lower than normal carrier mobility). Timing models are provided for these process variations.
- Voltage (V): The supply voltage delivered to the chip can fluctuate. Higher voltages generally allow transistors to switch faster, where lower voltages slow them down.
- Temperature (T): The chip's operating temperature affects carrier mobility and leakage. In modern technologies, higher temperatures typically make transistors slower.

STA is performed at these corners to guarantee functionality. Setup time checks, which ensure data arrives before a clock edge, are most challenging at the slow corner (e.g., slow process, low voltage, high temperature) where signal delays are longest. Hold time checks, which ensure data remains stable after a clock edge, are most critical at the fast corner (e.g., fast process, high voltage, low temperature) where signals travel quickest. Verifying the design at these opposing extremes builds confidence that the design will function reliably under all conditions.

All corners specified by the ihp13 library are shown in Table 2. As documentation is currently being developed, these details are taken directly from the filenames and file contents [9]. The original flow considers different corners at different stages. For synthesis, it considers only the typical (LV) corner, denoted as tt, while the P&R procedure also takes into account the fast (LV) corner, denoted as ff. This same structure will be used for this thesis to maintain consistency.

Table 2: ihp13 Library Corner Specifications

Corner Designation	Component	Voltage (V)	Temperature (°C)
Slow (LV)	SRAM	1.08	125
	Standard Cell	1.08	125
	I/O Cell	1.08 (Vdd) & 3.0 (ioVdd)	125
Slow (HV)	SRAM	1.08	125
	Standard Cell	1.35	125
	I/O Cell	1.35 (Vdd) & 3.0 (ioVdd)	125
Typical (LV)	SRAM	1.20	25
	Standard Cell	1.20	25
	I/O Cell	1.20 (Vdd) & 3.3 (ioVdd)	25
Typical (HV)	SRAM	1.20	25
	Standard Cell	1.50	25
	I/O Cell	1.5 (Vdd) & 3.3 (ioVdd)	25
Fast (LV)	SRAM	1.32	−55
	Standard Cell	1.32	−40
	I/O Cell	1.32 (Vdd) & 3.6 (ioVdd)	−40
Fast (HV)	SRAM	1.32	−55
	Standard Cell	1.65	−40
	I/O Cell	1.65 (Vdd) & 1.6 (ioVdd)	−40

2.3 Croc SoC Implementation with OpenROAD

The entire implementation and analysis flow was conducted using exclusively open-source EDA tools, executed within a containerized environment to ensure consistency and manage software dependencies. In this case, version 2025.03 of the hpretl/iic-osic-tools Docker image, based on the IIC-OSIC-TOOLS environment, was used [10], [11]. The implementation of the Croc SoC followed the standard automated flow provided by the Croc repository, which is documented in the README [8]. Running the flow as prescribed generates the post-synthesis and post-layout netlists, which are then analyzed as outlined in later sections.

2.4 Croc SoC Synthesis with Cadence Genus

To establish a basis for comparison against the open-source toolchain, a parallel implementation flow was developed using the proprietary Cadence EDA suite. As no predefined Cadence flow existed for the Croc SoC, a standard synthesis and implementation methodology was constructed, using the Cadence Genus Synthesis Solution – specifically version 23.33 – for logic synthesis. A tcl script was created to orchestrate this process. For initial testing, this script uses the default Genus settings. It simply reads the required files and synthesizes them without further configuration. After that, a comparison will be made between different settings of some synthesis effort variables. These variables, their possible values and the default value bold are shown below:

- syn_generic_effort {none low **medium** high}
- syn_map_effort {none low medium **high**}
- syn_opt_effort {none low medium **high** extreme}
- design_power_effort {**none** low high}

For this thesis, three sets of settings will be compared. Namely, all defaults, setting everything to none, and setting all settings to high. To enable a full comparison, an additional data point, namely synthesis runtime, will be tracked for each configuration. This will be tracked using the Linux time command, which runs the command and tracks: real time, user CPU time, and system CPU time.

The Cadence flow begins with a comprehensive setup phase to define the design environment. This includes specifying paths to the technology libraries, which, for this work, is the same PDK library used in the OpenROAD flow to ensure an equitable comparison. The script targets the same typical (LV) corner as the Yosys script from the original flow.

Logic synthesis commences within Cadence Genus. It reads and parses all the same System Verilog files as the original flow. This ensures that the logical input to both the open-source and proprietary synthesis engines is identical. An essential step at this stage is the elaboration of the design, where Genus builds an in-memory representation of the design hierarchy. Following elaboration, timing constraints are applied by sourcing the same constraints file used in the original flow. This file is taken from the P&R stage of the Croc flow, as Yosys is not timing-driven.

With the design elaborated and constrained, the core synthesis process, referred to as syn_generic, is initiated. This step performs high-level, technology-independent optimizations, focusing on architectural and logical transformations. Subsequently, the syn_map command maps the generic logic to the specific standard cells available in the IHP SG13G2 technology library. This technology mapping is a critical optimization step where the tool makes decisions to optimize PPA. The final synthesis command, syn_opt, performs post-mapping optimizations, including incremental logic restructuring, to further refine the PPA results.

The script concludes by generating comprehensive reports using built in tools, namely:

- report_gates
- report_qor
- report_timing
- report_area
- report_power

These artifacts constitute the native post-synthesis PPA metrics for the proprietary flow. In addition to comparing the quality of the netlists, this methodology allows for the Genus-synthesized netlist to be analyzed by the open-source OpenSTA tool. This facilitates a direct comparison between the timing and power analysis engines of the proprietary and open-source toolchains. Finally, the technology-mapped gate-level netlist is written to disk.

2.5 Post-Synthesis Analysis

As Yosys cannot perform timing analysis and to compare the analysis tools provided by both flows, OpenSTA, another tool from the OpenROAD toolchain, was used for timing analysis. Yosys excels at logic synthesis and technology mapping but does not include a built-in static timing analysis engine for path-based timing verification against SDC constraints. While its internal show and stat commands can provide structural and area information, and the logic mapping step is timing-driven based only on a target period, a dedicated STA tool is required to perform comprehensive setup and hold checks on the final synthesized netlist. Using OpenSTA allows for a consistent analysis methodology across different netlists and provides detailed timing path reports that are essential for evaluating the performance of the synthesized design. OpenSTA can also perform timing analysis on the Genus generated netlist, allowing for a direct comparison between the timing engines.

Preliminary analysis using OpenSTA was performed using a basic script, which does the following:

- Read in the library (.lib) files corresponding to the tt corner.
- Read in the library (.lef) files and the bond pad (.lef) file
- Read either the Yosys or Genus synthesized netlist
- Source the helper reports.tcl script included with the Croc flow
- Call report_metrics to generate a report

2.6 Post-Layout Analysis

The evaluation of the final, physically-implemented design relies on the comprehensive reports and output files generated by the automated OpenROAD flow. For this thesis, the primary source of post-layout data is the set of final reports generated by the make openroad command, specifically the 07_croc.final.rpt file. This report is generated at the final stage of the make openroad command. To create a full comparison between both netlists they are sent through the OpenROAD P&R stage. This will reveal if any differences observed post-synthesis also carry over to the layout stage.

2.7 Experimental limitations

It is important to note the scope and limitations of this experimental setup. The primary comparison in this work focuses on the logic synthesis stage and the full physical implementation of the open-source flow. A complete, end-to-end proprietary flow, which would involve place-and-route of the Genus-synthesized netlist using a tool like Cadence Innovus, was not performed as the required proprietary libraries were inaccessible. Consequently, a direct PPA comparison of a fully proprietary layout against a fully open-source layout is not presented. Additionally, all power analyses were conducted without simulation-based activity data, meaning the results are based on tool-default statistical estimates rather than realistic workload-driven switching activity.

3 Results & Discussion

This section presents the results of the comparative analysis between the open-source and proprietary EDA flows. The investigation proceeds in stages, beginning with a post-synthesis comparison of the netlists generated by Yosys and Genus. It then quantifies the predictive gap by analyzing the final post-layout results of the open-source flow. Finally, it provides a definitive post-layout comparison of both netlists after physical implementation in OpenROAD and examines the impact of different effort settings within the proprietary tool.

3.1 Post-Synthesis Quality of Results (QoR) Comparison

The initial analysis focused on the gate-level netlists produced by Yosys and Cadence Genus to establish a baseline for synthesis quality before physical implementation.

3.1.1 Initial Analysis and Identification of Estimation Artifacts

Initial analysis was performed by running both the Yosys and Genus-synthesized netlists through the OpenSTA tool to establish a direct, tool-for-tool comparison of the synthesis quality. Additionally, the Genus netlist was analyzed using its native timing engine. The PPA results from these three scenarios, using the default constraints.sdc file from the original flow, are summarized in Table 3.

The "Yosys" column, corresponding to the Yosys netlist shows nearly 5000 paths with setup violations, with the worst setup slack on the order of 60 ns. As the Croc SoC targets 80 MHz, this is significant. However, upon further analysis it turns out that this is caused by a single, unbuffered clock tree an extremely high fan-out. It is important to note that this specific issue, along with the numerous reported hold violations, is resolved during the physical design stage, as the OpenROAD tool inserts buffers to drive the high fan-out net and to fix short path delays. This buffering optimization is only performed during layout, as the tool can then leverage estimated physical placement to make informed decisions. As this one unoptimized net skewed the entire post-synthesis setup analysis, the netlist will also be analyzed with these paths marked as false paths in a refined SDC file. All subsequent comparative analyses presented in this thesis utilize this refined set of constraints to focus on the timing performance of the actual logic paths.

The next column "Genus (OR)" depicts the results for the Genus netlist, analyzed in OpenROAD using the same script. It shows a similarly large worst setup slack, also caused by an unbuffered high fan-out instance, confirming that both synthesis tools deferred the optimization of this net to the P&R stage. The hold violation count is comparable, reinforcing the observation that short path delays are not a primary focus during logic synthesis.

Interestingly, the final column, "Genus (Cadence)", which shows the results from Genus's internal timing analysis engine, paints a different picture. It reports only a single setup violation with a WNS (worst negative slack) of -1.641 ns and does not report any hold violations (noted as 'N/R' or Not Reported). This suggests that the Genus engine, by default, automatically identifies the high fan-out clock-gating net as an ideal net that will be buffered by the P&R tool and therefore excludes it from timing. This behavior leads to a significantly less pessimistic—and arguably more realistic—pre-layout timing estimate compared to the analysis performed by OpenSTA on the same netlist.

Table 3: Post-Synthesis PPA Metrics for Yosys vs. Genus

Metric	Category	Yosys	Genus (OR)	Genus (Cadence)
Performance	Worst Setup Slack	-63.97 ns	-58.36 ns	-1.641 ns
	Total Negative Setup Slack	-308,935.12 ns	-185,613.41 ns	-1.641 ns
	Worst Hold Slack	-0.23 ns	-0.41 ns	N/R
	Setup Violation Count	4,920	4,737	1
	Hold Violation Count	157	144	N/R
Power	Internal Power	3.90e-02 W	2.02e-02 W	1.76e-02 W
	Switching Power	7.18e-03 W	6.05e-03 W	1.39e-02 W
	Leakage Power	7.04e-06 W	6.13e-06 W	6.32e-06 W
	Total Power	**4.62e-02 W**	**2.63e-02 W**	**3.15e-02 W**
Area	Standard Cell Area	575,547.6 μm^2	457,867.5 μm^2	457,867.5 μm^2

Beyond timing, the PPA metrics in Table 3 reveal other key differences. The Genus-synthesized design achieves a notably smaller standard cell area (458k μm^2) compared to the Yosys design (576k μm^2), indicating a more area-efficient optimization by the Cadence tool. The power reports also show a discrepancy, however here it is critical to note that no switching activity file was provided to either toolchain. This analysis was therefore limited to using the tools' default statistical power models. A difference in these default models is a likely contributor to the variance in reported power, particularly the switching power component.

3.1.2 Synthesis Runtime Comparison

Another crucial dimension of comparison between the toolchains is the synthesis runtime. The Cadence Genus synthesis runs were executed on a server using 16 threads on dual AMD EPYC 9354 and took between 23 and 27 minutes of real time. The open-source Yosys flow was executed on a laptop also using 16 threads on an AMD Ryzen 7 5800H and completed in 2 minutes and 48 seconds of real time (2m43s user CPU time). Although the execution environments differ, this order-of-magnitude difference in runtime highlights a significant advantage of the Yosys flow for rapid design iteration. The results suggest a fundamental trade-off: the proprietary Cadence toolchain, with its advanced optimization heuristics, achieves superior area results at the cost of a substantially longer runtime, whereas the open-source Yosys tool provides a much faster turnaround time, which is highly beneficial for early-stage design exploration and debugging, albeit with less optimized results.

3.2 Timing Analysis with Refined Constraints

Using a refined SDC file that marks the high fan-out net as a false path, the analysis was re-run on both netlists using OpenSTA. The results are summarized in Table 4.

Table 4: Post-Synthesis Timing Comparison with Refined Constraints

Metric	Category	Yosys	Genus
Setup Timing	Worst Setup Slack	+1.32 ns	-1.59 ns
	Total Negative Setup Slack	0.00 ns	-1.59 ns
	Setup Violation Count	0	1
Hold Timing	Worst Hold Slack	-0.23 ns	-0.41 ns
	Hold Violation Count	157	144

A stark difference emerges from this refined analysis. The Yosys-synthesized netlist now reports zero setup violations, with a worst-case positive slack of +1.32 ns. This indicates that once the artifact of the unbuffered clock net is removed, the logical paths synthesized by Yosys comfortably meet the 80 MHz timing target at the typical corner.

In contrast, the Genus-synthesized netlist still exhibits a setup violation, albeit a single failing path with WNS of -1.59 ns. The critical path originates from a flip-flop within the timer unit and terminates at the status output port. This suggests that while Genus produced a more area-efficient design (as shown in the area reports), its default optimization effort resulted in a slightly slower logic path to this particular output port compared to the Yosys result. The -1.59 ns violation indicates that the design, as synthesized by Genus with default settings, would not meet the 80 MHz target without further optimization or manual intervention. The violating path in the Genus-synthesized netlist is the same failing path failing in the last column of Table 3. The difference in the reported slack value for this same path highlights that even when analyzing an identical netlist, different STA tools can produce slightly different results due to their unique internal timing models.

Both netlists continue to show a significant number of hold violations (157 for Yosys, 144 for Genus), with the Genus netlist having a slightly worse WNS of -0.41 ns compared to Yosys's -0.23 ns. Since neither synthesis tool performs physical placement, they do not attempt to insert delay buffers to fix these short paths, leading to hold violations being a common artifact of this design stage. The similarity in the number of violations suggests that both tools produce structurally similar short paths that will need to be addressed by the P&R tool.

3.3 The Predictive Gap in the Open-Source Flow

To evaluate the predictive accuracy of the post-synthesis analysis, a full physical implementation was performed using OpenROAD. This post-layout stage incorporates crucial physical effects, such as the actual delays of the synthesized clock tree and the parasitic resistance and capacitance of the routed interconnect, providing a signoff-quality assessment of the design's performance. The final PPA metrics, generated by the OpenROAD flow, are presented in Table 5 alongside the pre-layout results for direct comparison.

The results clearly demonstrate the significant impact of physical implementation on timing closure. As hypothesized, all 157 hold violations reported in the post-synthesis analysis were successfully resolved by the OpenROAD tool during the P&R process, resulting in zero hold violations in the final design. This is achieved through the strategic insertion of delay buffers into short logic paths, validating the common industry practice of deprioritizing post-synthesis hold fixes.

Conversely, the setup timing picture becomes more complex. While the initial -63.97 ns setup violation caused by the high fan-out net was resolved by the insertion of a proper buffer tree, the final design still exhibits 2,027 setup violations with a WNS of -2.449 ns. This indicates that while the most egregious pre-layout issue was fixed, the introduction of realistic interconnect delays from the SPEF file revealed thousands of other paths that fail to meet the 80 MHz target. This discrepancy highlights the "predictive gap" inherent in post-synthesis analysis; even after correcting for obvious estimation artifacts (the high fan-out net), the pre-layout analysis (with a WNS of +1.32 ns) provided an overly optimistic view of performance. The final design, with a WNS of -2.449 ns, would require further optimization loops or a reduction in target frequency to achieve timing closure.

The area and power metrics also shift significantly. The final standard cell area increased from 576k μm^2 to 690k μm^2, a rise of nearly 20%. This growth is primarily due to the addition of buffers during CTS and for hold/slew fixing. Consequently, the total power consumption also increased from 4.62e-02 W to 6.86e-02 W, driven by the increased cell area (higher leakage and internal power) and the additional switching power of the clock tree buffers. As noted in communication with the design maintainers, the remaining setup violations may be attributable to pessimistic modeling within the OpenROAD parasitic extraction (RCX) module, which may require further calibration.

Table 5: Post-Synthesis vs. Post-Layout PPA Summary (Typical Corner)

Metric	Category	Yosys	Yosys (False Paths)	OpenROAD
Performance	Worst Setup Slack	-63.97 ns	+1.32 ns	-2.449 ns
	Total Negative Setup Slack	-308,935.12 ns	0	-1987.24 ns
	Setup Violation Count	4,920	0	2027
	Hold Violation Count	157	157	0
Power	Internal Power	3.90e-02 W	3.90e-02 W	5.29e-02 W
	Switching Power	7.18e-03 W	7.18e-03 W	1.58e-02 W
	Leakage Power	7.04e-06 W	7.04e-06 W	9.91e-06 W
	Total Power	**4.62e-02 W**	**4.62e-02 W**	**6.86e-02 W**
Area	Standard Cell Area	575,547.6 μm^2	575,547.6 μm^2	689,891.1 μm^2

3.4 Post-Layout PPA Comparison: Yosys vs. Genus Netlists

The final comparison involved running both the Yosys and Genus netlists through the same OpenROAD P&R flow. The results, shown in Table 6, provide the most definitive assessment of netlist quality. The results reveal that the initial area and power advantages of the Genus-synthesized design carry over after layout using the OpenROAD P&R tool. The final standard cell area of the Genus-synthesized netlist after routing is roughly 18.5% smaller than the original flow. At the same time, the Genus design consumes approximately 25% less total power, with the most significant savings in internal power.

The performance metrics present a more nuanced picture. Both designs fail to meet the 80 MHz target, which may be attributable to pessimistic modeling in the RCX module as discussed in subsection 3.3. The Yosys-based design reports a better WNS of 2.449 ns, compared to -3.73 ns for the Genus design. This is the same path that was shown to have negative slack post-synthesis in Table 4. The violating path

count tells a different story: 2027 violating paths for the Yosys design compared to 151 violations for the Genus design. This implies that, although Genus has a singular path with WNS, the design overall should perform better.

Table 6: Yosys vs. Genus Netlist Post-Layout PPA Summary (Typical Corner)

Metric	Category	Yosys	Genus
Performance	Worst Setup Slack	-2.449 ns	-3.73 ns
	Total Negative Setup Slack	-1987.24 ns	-64.68 ns
	Setup Violation Count	2027	151
	Hold Violation Count	0	0
Power	Internal Power	5.29e-02 W	3.61e-02 W
	Switching Power	1.58e-02 W	1.51e-02 W
	Leakage Power	9.91e-06 W	8.79e-06 W
	Total Power	**6.86e-02 W**	**5.12e-02W**
Area	Standard Cell Area	689,891.1 μm^2	562,308.0 μm^2

3.5 Impact of Synthesis Effort on Proprietary Tool QoR

To assess the optimization capabilities of the proprietary toolchain, the Croc SoC was synthesized using Cadence Genus under three distinct effort level configurations: none, high, and the tool's defaults. The PPA results for each configuration, analyzed within the Genus timing engine, are presented in Table 7.

The data illustrates a clear correlation between synthesis effort and QoR. The high effort configuration yields the most favorable results across nearly all metrics when compared to the none configuration. It achieves a better setup slack, a lower total power consumption, and a smaller standard cell area by more than 1,000 μm^2. This demonstrates the effectiveness of the advanced logic restructuring and optimization algorithms employed by Genus when given sufficient runtime to explore the design space.

The comparison between high and default settings reveals that the default configuration achieves the best setup slack (-1.641 ns) and the smallest standard cell area (457.9k μm^2), marginally outperforming the high effort run. However, the high effort configuration reports slightly lower total power consumption. This is an expected outcome, as the high effort test case explicitly sets design_power_effort to high, enabling more aggressive power-saving optimizations, whereas this setting is none by default. This analysis highlights the complex, multi-objective nature of synthesis, where different effort settings result in distinct PPA trade-offs.

Furthermore, the synthesis time, captured using the Linux time command, reveals the computational cost of these optimizations. As expected, the high effort synthesis takes significantly longer (26m45s real time) than the none effort run (23m39s). Interestingly, the default configuration, despite achieving superior area and timing results, had a comparable runtime to the high effort configuration. This suggests that the default settings for Genus represent a well-balanced heuristic that delivers excellent QoR without an excessive runtime penalty compared to the all-high setting. These results underscore the ability for designers to tune the synthesis tool's behavior to meet specific project goals, whether they prioritize maximum performance, minimum area, or faster design iteration cycles.

Table 7: Different settings for Genus

Metric	Category	Genus (none)	Genus (high)	Genus (defaults)
Performance	Worst Setup Slack	-1.849 ns	-1.671 ns	-1.641 ns
	Setup Violation Count	1	1	1
Power	Internal Power	1.78e-02	1.76e-02	1.76e-02
	Switching Power	1.40e-02	1.38e-02	1.39e-02
	Leakage Power	6.28e-06	6.28e-06	6.32e-06
	Total Power	**3.18e-02 W**	**3.14e-02 W**	**3.15e-02 W**
Area	Standard Cell Area	459,787.1 μm^2	458,747.5μm^2	457,867.5μm^2
Synthesis Time	Real time	23m39s	26m45s	27m14s
	User CPU	21m07s	23m04	23m05s
	System CPU	1m17s	2m00	1m28s

4 Conclusion and Future Perspectives

This thesis conducted a comparative study of EDA methodologies by investigating the synthesis quality of the open-source Yosys toolchain versus the proprietary Cadence Genus, and by quantifying the "predictive gap" within the end-to-end open-source OpenROAD flow. Using the Croc RISC-V SoC and the IHP 130 nm PDK as a common baseline, this work provides direct, data-driven insights into the trade-offs between these two design philosophies.

At the synthesis level, the proprietary Cadence Genus tool produced a netlist of superior quality, achieving 1.75 times better power efficiency and a significantly smaller standard cell area (458k μm^2 vs. 576k μm^2). This advanced optimization came at the cost of a substantially longer runtime of approximately 25 minutes, in stark contrast to the 3-minute synthesis time of Yosys, which highlights the open-source tool's clear advantage for rapid design iteration and debugging.

Post-synthesis timing analysis proved to be an unreliable predictor of final performance. A design that was timing-clean at the logical level ultimately failed timing closure after physical implementation, accumulating over 2,000 setup violations with a WNS of -2.449 ns. This performance degradation was a direct consequence of introducing realistic physical effects, with buffer insertion for the clock tree and hold-time fixing causing a 20% increase in total standard cell area. This clearly illustrates that post-synthesis metrics are insufficient for performance signoff and that full post-layout analysis is non-negotiable.

The findings of this thesis have broader implications for the adoption of open-source EDA within the semiconductor industry. The rapid iteration speed of Yosys confirms the value of the open-source ecosystem for academic research, startups, and early-stage design exploration, where quick feedback cycles are paramount. However, the significant predictive gap and lower optimization quality observed highlight a critical maturity hurdle. For open-source tools to gain trust for high-volume or performance-critical commercial tapeouts, future development must focus on improving correlation between pre- and post-layout analysis and advancing optimization algorithms to rival their proprietary counterparts. Until then, the industry is likely to maintain a hybrid approach, leveraging open-source tools for initial development and proprietary tools for final signoff verification and optimization.

Future work should involve a complete place-and-route of the Genus-synthesized netlist using a tool like Cadence Innovus to determine if its superior synthesis performance translates to a superior final PPA. Completing the proprietary flow would also serve as a crucial reference for validating the OpenROAD RCX engine's accuracy in parasitic extraction, which currently represents another significant area of uncertainty within the open-source ecosystem. Furthermore, incorporating simulation-based activity files would enable a more precise power comparison. In conclusion, this work affirms the trade-offs inherent in modern EDA tool selection. While the open-source ecosystem offers a fast and accessible path to silicon, proprietary tools provide more refined optimization. The significant predictive gap observed demonstrates that regardless of the toolchain, the journey from RTL to a physically-realized design is fraught with challenges that can only be identified and addressed through comprehensive post-layout analysis.

References

[1] R. Dennard, F. Gaensslen, H.-N. Yu, V. Rideout, E. Bassous, and A. LeBlanc, "Design of ion-implanted mosfet's with very small physical dimensions," *IEEE Journal of Solid-State Circuits*, vol. 9, no. 5, p. 256–268, oct 1974.

[2] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam, and D. Burger, "Dark silicon and the end of multicore scaling," in *2011 38th Annual International Symposium on Computer Architecture (ISCA)*, 2011, pp. 365–376.

[3] J. L. Hennessy and D. A. Patterson, "A new golden age for computer architecture," *Communications of the ACM*, vol. 62, no. 2, p. 48–60, Jan. 2019.

[4] D. Patterson and A. Waterman, *The RISC-V Reader: An Open Architecture Atlas*, 1st ed. Strawberry Canyon, 2017.

[5] D. B. T. Ajayi, "Openroad: Toward a self-driving, open-source digital layout implementation tool chain," *Proceedings of Government Microcircuit Applications and Critical Technology Conference*, 2019. [Online]. Available: https://par.nsf.gov/biblio/10171024

[6] EDA Association, Ed., *Design, Automation, and Test in Europe: Proceedings, February 23-26, 1998, Paris, France*. IEEE Computer Society, 1998.

[7] P. Sauter, T. Benz, P. Scheffler, H. Pochert, L. Wüthrich, M. Povišer, B. Muheim, F. K. Gürkaynak, and L. Benini, "Croc: An end-to-end open-source extensible risc-v mcu platform to democratize silicon," 2025.

[8] PULP Platform Contributors, "Croc system-on-chip." [Online]. Available: https://github.com/pulp-platform/croc

[9] IHP-GmbH, "Ihp open source pdk." [Online]. Available: https://github.com/IHP-GmbH/IHP-Open-PDK/

[10] H. Pretl. [Online]. Available: https://hub.docker.com/r/hpretl/iic-osic-tools/

[11] H. Pretl and G. Zachl, "GitHub repository of the IIC-OSIC-TOOLS." [Online]. Available: https://github.com/iic-jku/IIC-OSIC-TOOLS