UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO
CURSO DE GRADUAÇÃO EM CIÊNCIAS DA COMPUTAÇÃO

Rafael Moresco Vieira

Evaluating the Impact of Pin Assignment Order in VLSI Circuit Floorplanning Outcomes

Florianópolis
2024

Rafael Moresco Vieira

Evaluating the Impact of Pin Assignment Order in VLSI Circuit Floorplanning Outcomes

Trabalho de Conclusão de Curso do Curso de Graduação em Ciências da Computação do Centro Tecnológico da Universidade Federal de Santa Catarina para a obtenção do título de Bacharel em Ciências da Computação.
Orientador: Prof. José Luís Almada Güntzel, Dr.

Florianópolis
2024

Ficha catalográfica gerada por meio de sistema automatizado gerenciado pela BU/UFSC.
Dados inseridos pelo próprio autor.

Vieira, Rafael Moresco
Evaluating the Impact of Pin Assignment Order in VLSI Circuit Floorplanning Outcomes / Rafael Moresco Vieira ; orientador, José Luís Almada Güntzel, 2024.
72 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Ciências da Computação, Florianópolis, 2024.

Inclui referências.

1. Ciências da Computação. 2. Physical Design. 3. Floorplanning. 4. Electronic Design Automation. I. Güntzel, José Luís Almada. II. Universidade Federal de Santa Catarina. Graduação em Ciências da Computação. III. Título.

Rafael Moresco Vieira

Evaluating the Impact of Pin Assignment Order in VLSI Circuit Floorplanning Outcomes

Este Trabalho de Conclusão de Curso foi julgado adequado para obtenção do Título de "Bacharel em Ciências da Computação" e aprovado em sua forma final pelo Curso de Graduação em Ciências da Computação.

Florianópolis, 6 de dezembro de 2024.

Banca Examinadora:

Prof. José Luís Almada Güntzel, Dr.
Universidade Federal de Santa Catarina

Prof. Cristina Meinhardt, Dra.
Universidade Federal de Santa Catarina

Renan Oliveira Netto, Dr.
Universidade Federal de Santa Catarina

Este trabalho é dedicado a minha família, que sempre esteve ao meu lado.

ACKNOWLEDGEMENTS

I would like to express my deepest gratitude to those who have supported and guided me throughout the journey of completing this thesis. First and foremost, I am forever grateful to my family for their love, encouragement, and sacrifices. Your constant support has been my foundation and inspiration to persevere. I thank all my friends that where with my during my entire academic journey, some of them since our days of high school.

I am sincerely thankful to the Universidade Federal de Santa Catarina for providing a great learning environment, the great professors that guided me in this journey, and specially the Embedded Computing Lab (ECL). To all the colleagues from ECL, that where always present to support my work, and helped to introduce me to the world of Electronic Design Automation. Our weekly meetings always brought me new knowledge, and our long discussions of our respective works lead to a great intellectual development.

Lastly, I thank all the OpenROAD contributors, that not only work to bring free software to the public, but are also very receptive and were willing to help during the development of my work.

"Quem não aguenta o
trote não monta o burro."
(Getúlio Vargas)

ABSTRACT

The early stages of the physical design of VLSI (Very-Large-Scale Integration) circuits, referred to as floorplanning, are critical for achieving quality layouts as they directly affect subsequent design stages. A typical floorplanning flow consists of pin assignment, macro placement, and power planning. However, pin assignment and macro placement are interdependent steps, and their execution order significantly influences floorplanning outcomes, which is why they are often alternated. For example, the standard floorplanning flow in the open-source OpenROAD platform performs an initial random pin assignment followed by macro and global placement, concluding with an extra pin assignment step. In this approach, pin assignment does not directly influence the macro and global placement results, potentially missing optimization opportunities. This work explores new enhanced floorplanning flows within integrated circuit (IC) synthesis using OpenROAD by introducing an additional non-random pin assignment step. The proposed flows are tested with two macro placers available in OpenROAD. Experimental results using FreePDK45 test circuits demonstrated that the proposed flows achieved average reductions in wirelength and via count of 1.25% and 0.37%, respectively, with TritonMP, and 1.44% and 1.43%, respectively, with Hierarchical RTL-MP. Specific circuits showed reductions in wirelength and via count of up to 9.34% and 7.69%, respectively. These results underscore the potential for further optimizations during the floorplanning stage, highlighting the importance of addressing pin assignment.

Keywords: Floorplanning, Physical Design, Open Source, Placement.

RESUMO

As etapas iniciais do projeto físico de circuitos VLSI (do inglês Very-Large-Scale Integration), conhecidas como *floorplanning* (planejamento de planta baixa), são críticas para a obtenção de layouts de qualidade, pois afetam diretamente as fases subsequentes do projeto. Um fluxo típico de *floorplanning* consiste em assinalamento de pinos, posicionamento de macros e planejamento da distribuição da alimentação. No entanto, a atribuição de pinos e o posicionamento de macros são etapas interdependentes, e sua ordem de execução influencia significativamente os resultados do *floorplanning*, razão pela qual essas etapas são frequentemente alternadas. Por exemplo, o fluxo padrão de floorplanning na plataforma de código aberto OpenROAD realiza uma atribuição inicial aleatória de pinos, seguida pelo posicionamento de macros e posicionamento global, concluindo com uma etapa extra de atribuição de pinos. Em tal abordagem, a atribuição de pinos não influencia diretamente os resultados do posicionamento de macros e global, potencialmente perdendo oportunidades de otimização. Este trabalho explora novos fluxos de *floorplanning* aprimorados no contexto da síntese de circuitos integrados (IC) utilizando a OpenROAD, por meio da introdução de uma etapa adicional de atribuição de pinos não aleatória. Os fluxos propostos foram testados com dois posicionadores de macros disponíveis no OpenROAD. Os resultados experimentais, utilizando circuitos de teste do FreePDK45, demonstraram que o fluxo proposto alcança reduções médias no comprimento total dos fios e na contagem de vias de 1,25% e 0,37%, respectivamente, com o TritonMP, e de 1,44% e 1,43%, respectivamente, com o Hierarchical RTL-MP. Circuitos específicos apresentaram reduções no comprimento dos fios e na contagem de vias de até 9,34% e 7,69%, respectivamente. Esses resultados destacam o potencial de otimizações adicionais durante a etapa de floorplanning, enfatizando a importância de abordar a atribuição de pinos.

Keywords: Floorplanning, Physical Design, Open Source, Placement.

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS AND ACRONYMS

ASIC	*Application Specific Integrated Circuit*
DEF	*Design Exchange Format*
DRC	*Design Rule Check*
DRV	*Design Rule Violation*
EDA	*Electronic Design Automation*
HDL	*Hardware Description Language*
HPWL	*Half Perimeter Wirelength*
ICs	*Integrated Circuits*
IP	*Intellectual Property*
LEF	*Library Exchange Format*
LVS	*Layout-Versus-Schematic*
MPSoC	*Multiprocessor System-on-Chip*
OOP	*Object-Oriented Programming*
PPA	*Power-Performance-Area*
RL	*Reinforcement Learning*
RTL	*Register Transfer Level*
SoC	*System-on-Chip*
TNS	*Total Negative Slack*
UI	*User Interface*
VLSI	*Very-Large-Scale Integration*

CONTENTS

1 INTRODUCTION

The technological advancements have allowed the fabrication of *Integrated Circuits* (ICs) featuring over a billion transistors. The rising complexity of these designs calls for the extensive use of hierarchical design, functional blocks and an increasing use of *Intellectual Property* (IP) blocks. This trend has raised the importance of floorplanning in determining the quality of a *Very-Large-Scale Integration* (VLSI) design (LAUNG-TERNG WANG; CHENG, 2008). With the continuous reduction of technology nodes size, there is also an increasing demand to improve placement and routing, as the ever so smaller transistors allow for a greater circuit density that combined with more metal layers in more recent technology nodes make the total wirelength resistance and capacitance the main cause for delays and a major part of switching power loss (WESTE; HARRIS, 2011). Due to those issues, there is a prominent need for good quality floorplan that could lead to high quality placement and routing solutions.

In recent years many new approaches for VLSI optimization have been introduced in academia, ranging from an increase in AI-based design optimization for all stages of the physical synthesis (KAHNG, 2024) to the introduction of changes to the traditional steps. This latter approach was subject of ICCAD CAD Contests 2020 (HU; YANG, et al., 2020) and 2021 (HU; YU, et al., 2021) and highlighted that there are many optimization opportunities arriving from modifying the synthesis flow, either adding extra steps or by altering the behavior of the existing ones through the integration of new features.

With the growing demand from the *Electronic Design Automation* (EDA) industry, there has been a rise in open-source movements led by academia aimed at disseminating physical design knowledge. Over the years, several tools specific to each step of the synthesis flow have been developed. To support the rise of those tools, open-source libraries such as the Ophidian library developed at the Federal University of Santa Catarina (NETTO et al., 2018) appeared, until more comprehensive projects covering all stages began to emerge. Among these projects, OpenROAD stands out, a global initiative supported by several universities and companies, headquartered at UC San Diego, California (KAHNG, 2022), which serves as the foundational infrastructure for the development of this work.

This work investigates how the addition of an extra step of pin assignment impacts the resulting layouts of physical synthesis. To perform this investigation, the open-source EDA tool-kit OpenROAD (KAHNG, 2022) was chosen as our test case. We explain how the standard OpenROAD flow works, presenting its two variations due to the existence of two available macro placers, TrintonMP and Hierarchical RTL-MP (KAHNG; VARADARAJAN; WANG, 2023), and compare with three proposed investigative flows using test designs available within OpenROAD.

1.1 MOTIVATION AND OBJECTIVES

With modern IC designs growing increasingly complex, featuring billions of transistors and densely packed functional blocks, even slight inefficiencies in floorplanning can significantly impact performance, power consumption, and design feasibility. Considering how recent works add new steps in the global routing stage executing functions associated with global placement (FONTANA et al., 2021), there is still much to explore in the interactions between synthesis steps. This work, therefore, seeks to explore the potential of an altered floorplanning flow by adding an extra pin assignment step to improve routing efficiency and layout compactness. Using the OpenROAD platform as a test case, this work aims to identify and quantify the benefits of structured pin assignment, contributing to more optimized and accessible VLSI design flows, especially within the growing field of open-source EDA tools.

General Objectives

The general objective of this work is to investigate and introduce a new VLSI floorplanning flow that considers the dependency relation between the steps of macro placement, pin assignment and global placement in the physical synthesis process, utilizing the addition of extra pin assignment steps in order to enhance layout quality, utilizing the open-source toolkit OpenROAD as a test case. The experimental results will be compared with those generated by the standard flows avaliable within OpenROAD.

Specific Objectives

- Analyze the impact of pin assignment in relation to macro placement on the floorplanning outcomes by introducing additional, pin assignment steps within the OpenROAD flow;
- Compare the effectiveness of the proposed pin assignment modifications with different macro placers, TritonMP and Hierarchical RTL-MP, to determine improvements in layout efficiency, focusing on metrics like wirelength reduction and via minimization;
- Measure the synthesis runtime implications of the proposed flows and assess the trade-offs between layout quality improvements and computational resource demands;
- Conduct experimental evaluations on diverse OpenROAD test circuits, quantifying the benefits and limitations of the proposed flows and validating the feasibility of these approaches for integration in open-source VLSI design workflows.

2 BACKGROUND

This chapter presents the necessary theoretical knowledge to understand the current work.

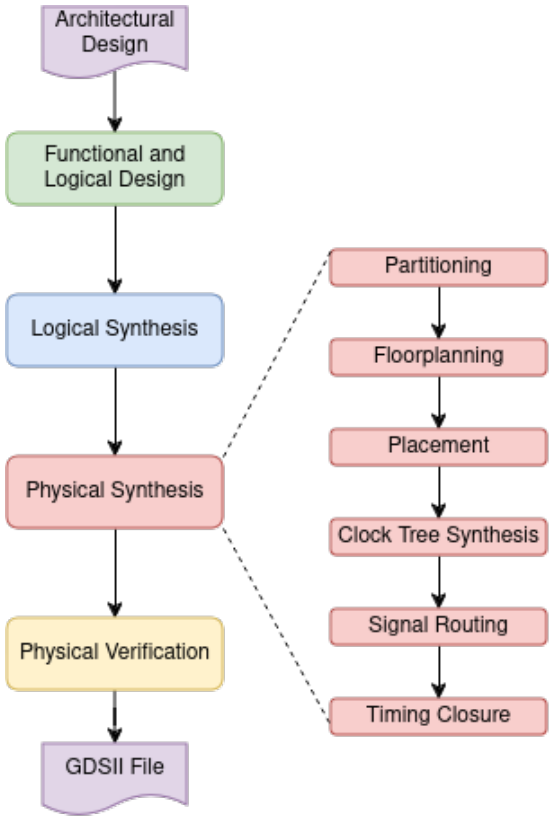
2.1 PHYSICAL SYNTHESIS

The RTL-to-GDSII synthesis flow is a critical process in VLSI circuit design, serving as the bridge from abstract functional descriptions to the detailed physical layouts necessary for manufacturing. The Fig. 1 shows the complete RTL-to-GDSII flow, highlighting the presence of the Physical Synthesis steps with red colored blocks. This flow begins at the *Register Transfer Level* (RTL), where designers use *Hardware Description Language* (HDL) like Verilog or VHDL to define the circuit's functional and timing behavior at a high level. From here, the synthesis process translates this high-level design into a gate-level netlist, mapping RTL logic to specific gates using standard cells from a given technology library (WESTE; HARRIS, 2011). This synthesis step optimizes the design to meet predefined constraints, such as minimizing area or power and ensuring timing closure, which is crucial for high-speed circuits.

Following synthesis, the flow advances to physical design, where the gate-level netlist undergoes a series of steps to create a manufacturable layout. Physical design includes placement, where cells are arranged on the chip 2D surface so as to minimize both area usage and interconnect delay. After placement, global and detailed routing phases establish the precise connections between cells, with a focus on minimizing wire length, reducing crosstalk, and ensuring signal integrity. According to Kahng, Lienig, et al. (2011), this stage requires iterative optimization to ensure timing closure, particularly as device sizes shrink and timing constraints grow more stringent. Tools are employed to refine the layout until timing closure is achieved, ensuring that all paths meet their designated timing requirements.

Timing analysis and iterative optimization steps follow placement and routing to ensure that the design meets timing requirements, known as *timing closure*. Modern EDA tools employ sophisticated algorithms to achieve this, as timing constraints become increasingly challenging in smaller technology nodes. The final stage of the RTL-to-GDSII flow includes *Design Rule Check* (DRC) and *Layout-Versus-Schematic* (LVS) verification, which confirm that the layout complies with manufacturing requirements and matches the intended design specifications. The output, the GDSII file, contains all the geometric and layer information needed for photolithographic fabrication, making it the final product of the digital design flow (WESTE; HARRIS, 2011).

Figure 1 – Academic RTL-to-GDSII Synthesis Flow.



Source: Adapted from Kahng, Lienig, et al. (2011).

2.2 FLOORPLAN

Floorplanning plays a crucial role in chip layout, specially in the hierarchical approach to module-based design. It offers preliminary feedback to assess architectural choices, predict chip areas, and estimate delays and congestion due to interconnections. As fabrication technology evolves, design complexity increases as more transistors are integrated in a single chip. To address such growing complexity, hierarchical design practices and IP modules are extensively employed. Consequently, floorplanning has become more important than ever for ensuring the quality of VLSI designs (LAUNG-TERNG WANG; CHENG, 2008).

Traditionally, the floorplanning step is further divided into macro placement, pin assignment, and power planning (KAHNG; LIENIG, et al., 2011). Among these, macro placement is particularly vital because it lays the foundation for the overall layout quality and directly influences key design metrics such as wirelength, congestion, and timing closure. The process of macro placement ensures that each block generated during the partitioning stage is assigned a position and shape within the floorplan, aiming to minimize wirelength and balance the area usage, which is crucial for efficient routing and timing optimization. In modern VLSI designs, where circuits consist of hundreds to thousands

of macros, effective placement becomes even more important. Poor macro placement can lead to long interconnects, resulting in increased signal delay and higher parasitic capacitances, which degrade performance and power efficiency. Furthermore, congestion in certain regions due to inefficient placement can create bottlenecks during routing, making it harder to meet design closure.

Although many advanced algorithms and tools have been developed to automate macro placement, most designers still rely on handcrafted placements for high-performance chips to fine-tune critical regions of the design (KAHNG; VARADARAJAN; WANG, 2023). However, current *Multiprocessor System-on-Chip* (MPSoC) may have thousands of macros and IP blocks, rendering manual macro placing unfeasible. In such scenario, macro placement can exploit design hierarchy in such a way that larger macros or modules are placed before smaller ones, ensuring that major components like memory blocks, processing units, or high-speed interfaces are optimally positioned.

During the pin assignment step, each incoming and outgoing signal is assigned to a specific pin location, looking to enhance the overall performance of the design. The primary goal of pin assignment is to optimize the placement of I/O pins, as it directly affects the efficiency of signal routing across the chip. By carefully assigning pins, designers can improve routability, minimize the total wirelength, and reduce the number of vias, which are critical factors in achieving a more compact and efficient design. Poor pin placement can lead to long, convoluted routing paths that increase resistance, capacitance, and signal delay, potentially causing timing violations and negatively impacting the overall chip performance. According to (KAHNG; LIENIG, et al., 2011) the ideal moment for the pin assignment is before the macro placement, with the locations being updated during and after macro placement. In the power planning step, the ground and power nets are routed in dedicated metal layers, usually the upper ones.

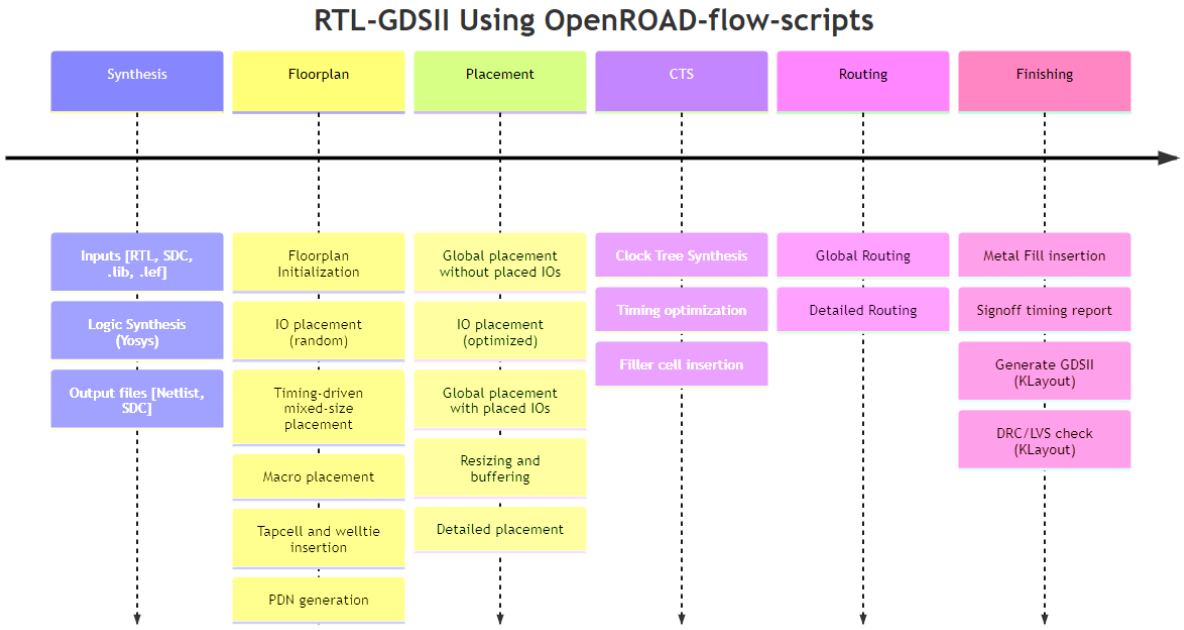
2.3 OPENROAD

With the growing demand in the EDA industry, there has been a surge of open-source movements led by academia aimed at disseminating knowledge in this field. Among these projects, OpenROAD stands out, an initiative supported by various universities and companies worldwide, headquartered at UC San Diego, California (KAHNG, 2022), which serves as the foundational infrastructure for the development of this work.

OpenROAD, launched in 2019, has the mission to develop a fully open platform, where the entire synthesis process would be completed within 24 hours without human intervention (KAHNG, 2022). To achieve this goal, OpenROAD is available in two versions: the standalone OpenROAD tool, where all commands are manually applied, designed to support the development of new tools, and a version called OpenROAD Flow Scripts, which consists of a set of scripts that automatically execute all steps of the tool, made for ICs designers. As shown in Fig. 2, the flow implemented in OpenROAD resembles

the theoretical flow identified by (KAHNG; LIENIG, et al., 2011), but it presents certain modifications and different nomenclatures.

Figure 2 – OpenROAD's Synthesis Flow.



Source: Adapted from Project (2024a).

Performing the entire RTL-to-GDSII synthesis flow, OpenROAD (KAHNG, 2022) is composed of several individual tools with a common interface and data types. For the floorplan step, there are four essential tools: Pin Placer, TritonMP, Hierarchical RTL-MP and RePlAce. Introduced by Bandeira et al. (2020), the Pin Placer employs a divide-and-conquer strategy combined with Hungarian matching (KUHN, 1955) to achieve efficient pin assignment. This method divides the I/O pin assignment problem into smaller, manageable subproblems, which reduces computational complexity and allows for parallel processing, resulting in faster runtime and scalability for large designs. The Hungarian matching algorithm, capable of solving assignment problems optimally (KUHN, 1955), is then used within each subregion to assign pins while minimizing total wirelength. The tool also offers alternative assignment methods, such as simulated annealing, which is useful for exploring a wider solution space at the cost of increased computation time. By balancing precision with computational efficiency, the Pin Placer tool aims to provide a flexible and scalable solution for pin assignment in modern, complex integrated circuits.

The macroplacer TritonMP was developed utilizing an implementation of ParquetFP, an open-source floorplanning tool introduced by Markov and Adya (2003). ParquetFP primarily focuses on fixed-outline floorplanning, an approach that is especially relevant for hierarchical *Application Specific Integrated Circuit* (ASIC) and *System-on-Chip* (SoC) designs. Unlike classical floorplanning methods, which minimize area and

wirelength without specific layout boundaries, fixed-outline floorplanning mandates that the layout conforms to a predetermined outline, making it more applicable to real-world designs where chip dimensions are constrained. To address the increased complexity of fixed-outline constraints, ParquetFP incorporates advanced objective functions within its simulated annealing framework. It uses wirelength minimization based on the *Half Perimeter Wirelength* (HPWL) metric and aspect ratio adjustments to handle varying block shapes effectively. ParquetFP also introduces slack-based moves, which allow for local adjustments to minimize wirelength while maintaining critical path constraints. This combination of techniques makes ParquetFP highly effective for both outline-free and fixed-outline contexts, providing scalable, high-quality floorplanning solutions suitable for the hierarchical design methodologies employed in modern VLSI layouts

Hierarchical RTL-MP is a sophisticated hierarchical macro placer developed for OpenROAD to handle the increasing complexity and scale of VLSI designs (KAHNG; VARADARAJAN; WANG, 2023). With the rise of auto-generated RTL, particularly in areas like machine learning accelerators, the number of macros can reach several hundred in a single design, making traditional peripheral placement methods unfeasible. Unlike previous macro placers that often arranged macros along the periphery, Hierarchical RTL-MP can place macros within the core of the layout, accommodating large macro numbers and better maintaining design dataflow. The tool employs a multi-level hierarchical approach, transforming logical hierarchies from the RTL into physical hierarchies through a novel autoclustering technique. This technique groups macros into clusters based on design hierarchy and dataflow, creating physical clusters that mimic the logical relationships. Additionally, Hierarchical RTL-MP uses a shaping engine to determine allowable cluster shapes, which it refines through a bottom-up and top-down process to optimize floorplan utilization and routability. This enables more efficient macro placements that align with critical timing paths, support power grid generation, and minimize wirelength. Empirical tests by Hierarchical RTL-MP authors have shown that Hierarchical RTL-MP outperforms prior placements by reducing timing violations and runtime, making it a valuable addition to the OpenROAD toolkit for complex IP blocks.

RePlAce is a mixed-size placer developed to enhance solution quality and address routability challenges in global placement (CHENG et al., 2019). It builds upon ePlace (LU et al., 2014), a previous analytical placer utilizing an eletrostatic model, by implementing new techniques that improve both placement quality and routability validation. RePlAce leverages a density function that incorporates local area overflow, allowing it to address congestion at a finer granularity, per placement bin. This approach enables localized adjustments rather than globally applied density penalties, thus preserving overall wirelength while effectively managing high-density areas.

To further optimize placement quality, RePlAce integrates a dynamic step size adaptation method that adjusts optimization effort based on the design's placement

state, improving efficiency without increasing runtime significantly. Additionally, RePlAce includes a routability-driven component, which estimates congestion early in the flow and performs cell inflation in congested regions to avoid hotspots. This helps in producing layouts with minimal routing congestion. RePlAce achieves notable improvements in HPWL and routability across various benchmarks, making it a robust tool for tackling the challenges of modern, large-scale VLSI designs.

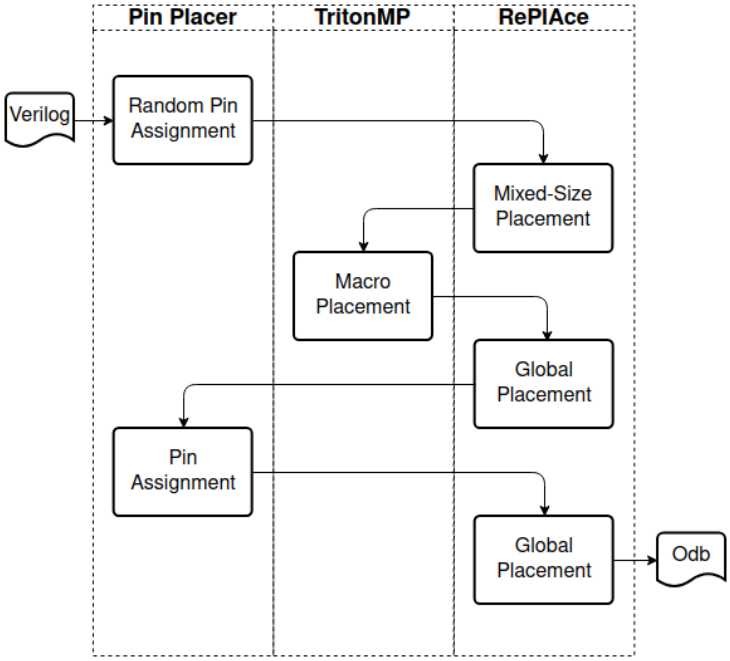
The input to the floorplanning step in OpenROAD is a verilog netlist generated by the third party logic synthesis tool Yosys (YOSYS HQ, 2024). This file is loaded and converted into an internal file type. Then, floorplanning starts. The final output of the floorplanning is a proprietary file called ODB, utilized to transfer information from OpenROAD's internal database.

Fig. 3 shows a simplified flowchart of the OpenROAD floorplanning flow using TritonMP as macro placer, while Fig. 4 shows a similar flowchart using the newer Hierarchical RTL-MP as macro placer, hereinafter referred to as flow1_TMP and flow1_hier, respectively. Each column in the flow chart identifies the tool that executes the step.

The first step of flow1_TMP corresponds to randomly assigning the I/O pins. The second step is a timing driven mixed-size placement using RePlAce, where the macro blocks get an initial placement, followed by the refining macro placement with TritonMP. After the macro placement, tapcells and wellties are inserted, and the power delivery network is routed. Although OpenROAD classifies this as the end of the floorplanning, the pin assignment is not finished. After floorplanning, OpenROAD invokes a global placement ignoring the I/O pins, as they were previously randomly placed. Being classified as part of placement, the definitive pin assignment is made after the previous global placement step, and to refine the results, a new global placement step is performed, but now considering the pin assignment.

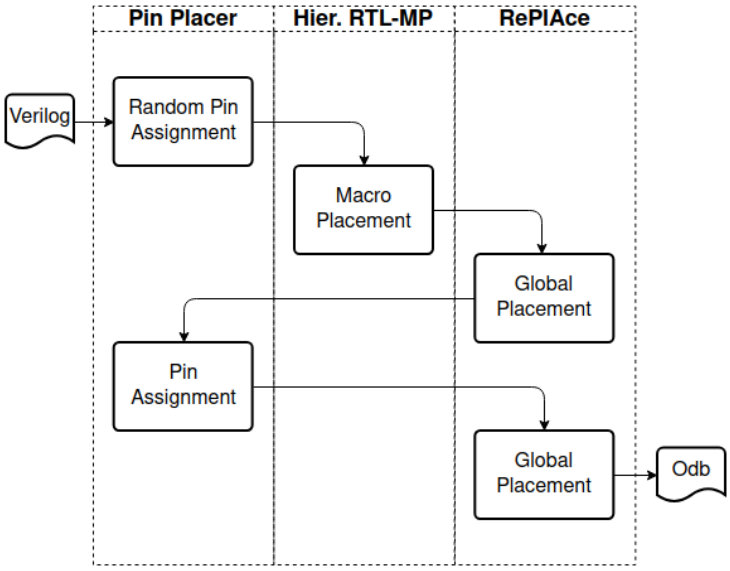
The flow using Hierarchical RTL-MP as macro placer, flow1_hier, starts by randomly assigning the I/O pins. But differently from flow1_TMP, in flow1_hier there is no mixed-size placement using RePlAce. Instead, it goes directly to the macro placement step with Hierarchical RTL-MP. The rest of flow1_hier undergoes the same steps as flow1_TMP, with a global placement ignoring the randomly placed I/O pins positions being the next step, followed by a definitive I/O placement and a final global placement.

Figure 3 – Original Floorplanning Flow flow1_TMP.



Source: The Author.

Figure 4 – Original Floorplanning Flow flow1_hier.



Source: The Author.

3 RELATED WORK

The following sections present the most recent works on floorplanning techniques.

3.1 DREAMPLACE

The DREAMPlace framework, presented by Lin et al. (2021), is a GPU-accelerated tool designed to enhance the speed and scalability of the VLSI placement process by leveraging deep learning techniques. Traditional analytical placement methods, though capable of high-quality results, often suffer from extended runtimes due to the computational complexity involved in optimizing large-scale designs. DREAMPlace addresses this issue by framing placement as a neural network training problem, which enables efficient gradient calculations for wirelength and density through GPU acceleration. Utilizing PyTorch, the framework achieves up to 30× speed improvement over multi-threaded CPU counterparts like RePlAce, without compromising placement quality. This method not only accelerates global placement tasks but also ensures legal placement with minimized overlaps, making it particularly suitable for designs with millions of cells. DREAMPlace has paved the way for integrating advanced AI tools in EDA, setting a strong foundation for further GPU-accelerated VLSI design methodologies .

3.2 AUTODMP

The work by Agnesina et al. (2023) explores recent advances in VLSI macro placement by leveraging the DREAMPlace (LIN et al., 2021) GPU-accelerated placer combined with machine learning-driven optimization strategies, introducing a new macro placer called AutoDMP. Prior works in macro placement have traditionally divided the problem into floorplanning and placement of standard cells and macros separately, which can hinder performance due to limited coordination between these elements. Classical floorplanning methods include simulated annealing techniques and partitioning-based approaches, while more recent methods have applied reinforcement learning to achieve competitive placements. However, these approaches either suffer from scalability issues or require significant computational resources.

AutoDMP advances the field by integrating Bayesian optimization with DREAMPlace, allowing simultaneous placement of macros and standard cells in a mixed-size environment. Unlike sequential flows, which may struggle to optimize placement quality when faced with complex multi-objective goals such as timing, wirelength, and area, the AutoDMP methodology achieves a Pareto-optimal balance across these competing factors. By utilizing a multi-objective Tree-Structured Parzen Estimator (MOTPE), AutoDMP refines parameters to deliver high-quality placements with improved *Power-Performance-Area* (PPA) metrics on benchmarks. Additionally, DREAMPlace is extended with features

like macro halo adjustments and density constraints, which facilitate effective placement legalization while reducing overlaps. This method has demonstrated comparable or superior performance to industry-standard EDA tools, marking a significant step toward scalable and efficient VLSI design automation.

3.3 INCREMACRO

IncreMacro, developed by Pu et al. (2024), builds upon DREAMPlace's analytical placement capabilities by introducing a targeted approach to incrementally refine macro placement. Recognizing that placing macros centrally can lead to routing congestion and timing issues, IncreMacro employs a three-stage methodology: it diagnoses poorly placed macros using a KD-tree, shifts them towards the periphery through gradient-based optimization, and finalizes positions using constraint-graph-based linear programming. This structured approach preserves the relative positioning of macros, thus maintaining wirelength optimization from the original placement, while eliminating central blockages that hinder routing. In evaluations on RISC-V benchmarks, IncreMacro shows significant improvements in PPA metrics compared to DREAMPlace alone, with reductions in wirelength and power consumption, making it a valuable refinement tool in GPU-accelerated placement flows.

3.4 GOOGLE'S REINFORCEMENT LEARNING

The work by Mirhoseini et al. (2021) propose a novel approach to chip floorplanning using deep *Reinforcement Learning* (RL). This method reformulates the floorplanning problem as a Markov decision process, leveraging an edge-based graph neural network for learning effective representations of chip netlists. Their approach optimizes key metrics such as PPA while considering constraints like routing congestion and density. Notably, the method produces manufacturable floorplans within six hours, claiming to surpass or equal human expert designs across metrics. By enabling generalization and transfer learning, the proposed RL framework not only accelerates the chip design process but also has the potential to transform other placement optimization tasks beyond chip design. The methodology was applied in the design of Google's TPU accelerators.

This work was heavily criticized for its lack of reproducibility, flawed methodologies, and exaggerated claims. A recent work by Markov (2024) reviews and meta-analyses the work by Mirhoseini et al. (2021), highlighting several critical shortcomings: the omission of necessary details for reproducing experiments, reliance on proprietary data, and poorly documented baselines. Independent evaluations demonstrated that standard techniques like simulated annealing and commercial EDA tools outperformed the proposed RL approach. The RL methodology was further questioned for its use of a flawed proxy cost function, which correlated poorly with actual chip metrics, and for overstating results with-

out proper statistical analysis. Allegations of research misconduct, including cherry-picking favorable results, were also reported, raising doubts about the study's integrity. Despite the claimed innovation, subsequent investigations revealed that the methodology failed to surpass state-of-the-art approaches in both performance and efficiency (MARKOV, 2024).

3.5 HIDAP

In the work by Vidal-Obiols et al. (2021), the authors present a novel approach to macro placement in VLSI design, emphasizing the use of RTL dataflow information. This method, named HiDaP, stands out from traditional approaches by integrating dataflow-driven techniques with a hierarchical, multilevel optimization strategy. By leveraging RTL structural details, such as hierarchy and pipeline register stages, HiDaP calculates dataflow affinity between blocks based on factors like bitwidth and latency, which aids in maintaining timing and minimizing wirelength. Unlike typical analytic and partitioning-based placement methods, this approach emphasizes the interactions between macros and their surrounding standard cells, optimizing not only for placement quality but also for ease in meeting timing constraints with minimal manual adjustments. HiDaP uses a recursive, top-down placement model, which provides a robust structure for managing large designs with diverse macro requirements and varying block sizes. It is shown to achieve results that closely rival or even exceed those of handcrafted placements in terms of timing and wirelength performance, positioning HiDaP as an efficient alternative in industrial design flows.

3.6 CONCLUSION

The reviewed methodologies address critical aspects of macro placement, the most computationally intensive part of floorplanning, with unique strengths and limitations, but do not approach floorplanning as a whole. DREAMPlace establishes a foundation with GPU-accelerated global placement, achieving substantial speedups without quality degradation (LIN et al., 2021). However, its focus on general analytical placement leaves room for specialized improvements in macro handling and specific design constraints.

AutoDMP builds on DREAMPlace, introducing machine learning-driven multi-objective optimization to tackle mixed-size placement challenges (AGNESINA et al., 2023). Its integration of Bayesian optimization delivers Pareto-optimal placements across PPA metrics, though its reliance on extensive computational resources may limit accessibility for iterative design flows.

IncreMacro refines macro placement incrementally, preserving relative positional relationships and reducing central blockages that degrade routing (PU et al., 2024). By employing KD-tree-based diagnosis and gradient-based adjustments, it enhances routabil-

ity and timing while minimizing disruption to established layouts. However, its benefits are constrained by reliance on initial placement quality from analytical tools.

The controversial work responsible for Google's Reinforcement Learning technique proposed a novel use of RL for macro placement, achieving results comparable to human experts on proprietary TPU designs (MIRHOSEINI et al., 2021). However, subsequent critiques (MARKOV, 2024) revealed significant flaws in reproducibility, unverified claims of superiority over standard methods, and reliance on a proxy cost function poorly correlated with actual chip metrics.

HiDaP introduces a dataflow-driven, hierarchical approach, leveraging RTL information to optimize timing and wirelength while integrating macros and standard cells (VIDAL-OBIOLS et al., 2021). Its structured top-down process balances scalability with design-specific adaptability. Despite these advantages, its hierarchical dependency may complicate integration with flat placement frameworks.

Table 2 lists and compares the main properties of the related work. In the first column, the reference of the work is listed. In the following columns, it is indicated what kind of strategy the work implements.

Table 2 – Related work comparison.

Work	Macro Placer	Machine Learning	Bayesian Network	Linear Programming	Markov Decision	Dataflow Driven
(LIN et al., 2021)		X				
(AGNESINA et al., 2023)	X	X	X			
(PU et al., 2024)	X	X		X		
(MIRHOSEINI et al., 2021)	X	X			X	
(VIDAL-OBIOLS et al., 2021)	X					X

In conclusion, while DREAMPlace and AutoDMP excel in computational efficiency and mixed-size optimization, IncreMacro and HiDaP target placement refinement and structural awareness, respectively. In this work, instead of focusing only on a part of it, the general floorplanning flow will be approached, independently of the macro placer used.

4 METHODOLOGY

To elaborate a new VLSI floorplanning flow a profound study of the standard flow and the properties of each tool in the floorplan is required. This section describes the main characteristics of the general structure and the used macro placers available in the OpenROAD toolkit (PROJECT, 2024a).

4.1 OPENROAD STRUCTURE

Utilizing *Object-Oriented Programming* (OOP), the OpenROAD toolkit is built in a modular architecture, where every individual tool has its own separate module and namespace, and connected together with a top module. This top module, aptly named *ord*, creates an *User Interface* (UI) where its possible to individually call all the public commands created by the individual tool modules. For the industrial use of the toolkit, instead of utilizing the central *ord* module, every step of the RTL-GDSII flow is separately executed, initiating a specific instance of the toolkit for every module to be used. The toolkit is mostly written in the C++ language, utilizing a few Python scripts for linking support to the compiler.

The flow is implemented utilizing the GNU Make tool, where every circuit has its own *config.mk* file defining environment variables, source HDL description and technology node used. These files are imported in a central *Makefile*, where several different *Tcl* scripts are executed utilizing the imported information. Every step in the flows described by figures 4 and 3 is a different *Tcl* script. To connect this step by step execution approach, there is an unifying database structure with a proprietary file, the OpenDB module with its respective *.odb* files.

The OpenDB module is a comprehensive design database used to support tools for physical chip design. It is structured to handle various aspects of chip design, including layout, routing, and technology information. The module follows a design pattern that separates the interface from the implementation, resulting in public and private classes for each database object. The odb module uses a system of object tables and pages to manage database objects efficiently. Each object has a unique identifier (OID) that is persistent across save/restores, allowing for consistent referencing. The database supports hierarchical design by allowing blocks to contain other blocks, instances, and modules. This is managed through various hash tables and vectors that store references to these objects. The database has support classes that allow event-driven programming by providing hooks that can be implemented to respond to changes in the database, such as the creation or destruction of instances and nets. The database is designed to be saved and restored with exact fidelity, ensuring that the layout and state of the design are preserved across sessions.

Besides operating with the *.odb* files, OpenDB also manages the input and output for

the standard industry file formats *Library Exchange Format* (LEF) and *Design Exchange Format* (DEF), used to describe the physical aspects of ICs during various design stages. LEF describes the physical characteristics and constraints of standard cells, macros, and IP blocks in a technology library, while DEF represents the design of an entire chip or a specific block in terms of placement, routing, and connectivity.

4.2 MACRO PLACERS

TritonMP, as implemented in the MacroPlacer class with the *mpl* namespece, is the first implemented macro placer in OpenROAD, being replaced by the newer Hierarchical RTL-MP. It is designed to place macros or blocks in a circuit layout while considering various constraints such as halos, channels, and snapping to cell rows. TritonMP offers two main placement strategies: corner_min_wl and corner_max_wl. These strategies determine whether the tool should minimize or maximize the wire lengths of connections between macros, with the maximization strategy forcing the macros to the corners of the layout. In the standard flow utilizing TritonMP (flow1_TMP), the maximization strategy is the default. The tool uses a ParquetFP (MARKOV; ADYA, 2003) based annealing engine to optimize the placement of macros, aiming to minimize or maximize wire lengths based on the chosen strategy. The code in algorithm 1 shows the internal flow of the maximum wirelenth strategy.

The function begins by checking if the *MacroPlacer* is properly initialized in line 1. If not, it returns immediately. Line 4 calculates the initial weighted wire length of the macros and logs this information. A Layout object is created to represent the area where macros will be placed in line 5. A Partition object is initialized with all macros, representing the entire layout area in line 6 to 7. In lines 8 and 9 *MacroPartMap* is created to map macros to their respective partitions, and this map is filled using the *makeMacroPartMap* function. If the placement is connection-driven, the netlist table is filled with connection weights using the *fillNetlistTable* function as seen in line 10 and 11. The layout is divided into partitions using cut lines, and each partition is annealed using the ParquetFP engine to explore different macro placements. The function iterates over all possible partition sets, annealing each one and updating macro locations based on the best solution found, as seen in lines 17 to 33. After annealing, the function evaluates the solutions based on the weighted wire length and selects the best one. If a better solution is found, it updates the macro locations in the database. The function updates the database with the final macro placements. Both strategies follow this same flow, only changing the values in line 14 and 28, either choosing the smallest value possible or like seen in the code choosing the largest value possible.

The *fillNetlistTable* function in the Partition class executed in line 12 is responsible for populating a table (*net_tbl_*) that represents the connectivity and weights between macros and core edges within a partition. This table is used to guide the macro placement

Algorithm 1: MaxWL

```
Input  : odb = Internal data structure
Output : odb = Internal data structure
if not initialized then
    return
end
initial_wl ← calculateInitialWeightedWireLength()
layout ← createLayout(lx, ly, ux, uy)
partition ← createPartition(ALL, lx, ly, ux - lx, uy - ly)
partition.macros ← macros
macroPartMap ← createMacroPartMap()
makeMacroPartMap(partition, macroPartMap)
if connection_driven then
    partition.fillNetlistTable(macroPartMap)
end
partitionSets ← getPartitions(layout, partition)
bestWwl ← −∞
bestSetIdx ← 0
foundBest ← False
foreach partitionSet in partitionSets do
    if partitionSet.size() = 1 then
        continue
    end
    foreach curPart in partitionSet do
        success ← curPart.anneal()
        if not success then
            break
        end
    end
    curWwl ← calculateWeightedWireLength()
    if curWwl > bestWwl then
        bestWwl ← curWwl
        bestSetIdx ← indexOf(partitionSet)
        foundBest ← True
    end
end
if foundBest then
    updateMacroLocations(bestSet)
end
```

process by providing information about the relative importance of different connections. This function utilizes a data structure called Core Edges. Instead of considering the position of the block terminals (BTerm), OpenDB structure for I/O pins, TritonMP only considers in which of the edges of the circuit the I/O is located. The function begins by calculating the total number of macro and core edge connections (*macro_edge_count*) and resizing the *net_tbl_* to accommodate all possible connections. The function then fills the *net_tbl_* with weights for all pairs of macros and core edges.

The Hierarchical RTL-MP is created by the HierRTLMP class. It operates within the *mpl2* namespace and is part of a framework that supports multi-level clustering and timing-driven macro placement. The class is structured to handle various stages of macro placement, from initialization to final placement and orientation improvement, each addressing different aspects of macro placement:

- Multilevel Autoclustering: Converts the logical hierarchy into a physical hierarchy, setting the stage for subsequent placement steps.
- Coarse Shaping: Determines rough shapes for clusters, focusing on macro sizes and ignoring standard-cell clusters.
- Fine Shaping: Refines cluster shapes based on parent cluster outlines and locations.
- Hierarchical Macro Placement: Places clusters and macros in a top-down approach, considering both cluster and macro levels.
- Boundary Pushing: Adjusts macro clusters to design boundaries, ensuring no overlap with IO blockages or other macros.
- Orientation Improvement: Optimizes macro orientation to improve wirelength reduction.

This organization can be seen by the code described in algorithm 2.

In line 1 the multilevel autocluster is called. Line 2 verifies if the hand made macro placement flag is active, returning if it is. If there are already cells in the macro placer data structure, it is cleared as seen in lines 5 and 6. In line 8 the Coarse shaping is called. The function *runHierarchicalMacroPlacement* in line 9 is responsible for both the macro placement and the fine shaping. To realize the boundary pushing, it is created a pusher object, that receives the circuit area and the blockages, and a method of this object is called, as seen in line 10 and 11. In line 12, the positioned macros are placed in the odb structure. For the orientation improvement, fake cell placements are made in line 13, and utilized to change the macro orientations in line 14. The oriented macros are updated on the data structure. In line 16 a file writer is called, used to generate and odb file with the macro placement. Lastly, all the constraints and internal structures are cleaned in line 17.

Algorithm 2: Run_Hier

Input : odb = Internal data structure
Output : odb = Internal data structure,
def = Design placement

```
1  runMultilevelAutoclustering()
2  if skip_macro_placement then
3  |  return
4  end
5  if not tree.has_std_cells then
6  |  resetSAParameters()
7  end
8  runCoarseShaping()
9  runHierarchicalMacroPlacement()
10 pusher ← createPusher(tree.root, block, boundary_to_io_blockage)
11 pusher.pushMacrosToCoreBoundaries()
12 updateMacrosOnDb()
13 generateTemporaryStdCellsPlacement(tree.root)
14 correctAllMacrosOrientation()
15 commitMacroPlacementToDb()
16 writeMacroPlacement(macro_placement_file)
17 clear()
```

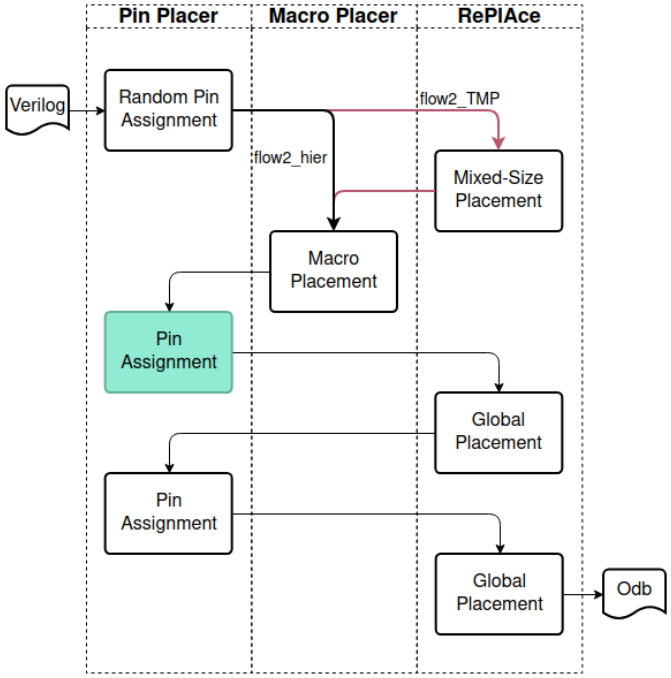
There are two instances of interactions between the circuit I/Os during the execution of the Hierarchical RTL-MP. The first one happens during the *runMultilevelAutoclustering* function, where the new hierarchy is created. In this function, the I/O are considered not by their position, but their connectivity with circuit pins and nets, generating the new hierarchical blocks. The second interaction within the *runCoarseShaping* function, where the placed I/Os are used to created area blockages for the macro placer. This blockage is used during the *runHierarchicalMacroPlacement* and the *pushMacrosToCoreBoundaries* functions.

The conclusion from this macro placer analysis is that both TritonMP and Hierarchical RTL-MP do not fully consider the position of the circuit I/Os. TritonMP has a stronger connection with the I/O locations, as it considers in what position of the circuit they are for the weight calculation. This justifies the first step of pin placement in the standard flow (flow1_TMP and flow1_hier) being random, as the following macro placement step will not consider the exact I/O positions. This disconnection from the pin placement and macro placement will be addressed in the following section, with the introduction of new flows.

5 PROPOSED FLOWS

After analyzing OpenROAD's macro placers properties described in the previous section, it was noticed that the macro placement has a certain independence from the pin assignment, more prevalent in the newer Hierarchical RTL-MP due to not only this macro placer consider the I/Os placement just as blockage areas, but also the lack of the initial mixed-size placement before the macro placement step present in the TritonMP flow. The mixed-size placement made by the RePlAce tool considers the randomly assigned I/Os, causing the following macro placement to indirectly consider the I/Os trough the positioned standard cells. Although the older macro TritonMP considers more of the I/O positions with the core edges, there is still a degree of independence from the exact I/O placement. With this in mind, two initial flows were devised, one for each macro placer.

Figure 5 – Proposed Floorplanning Flows flow2_TMP and flow2_hier.



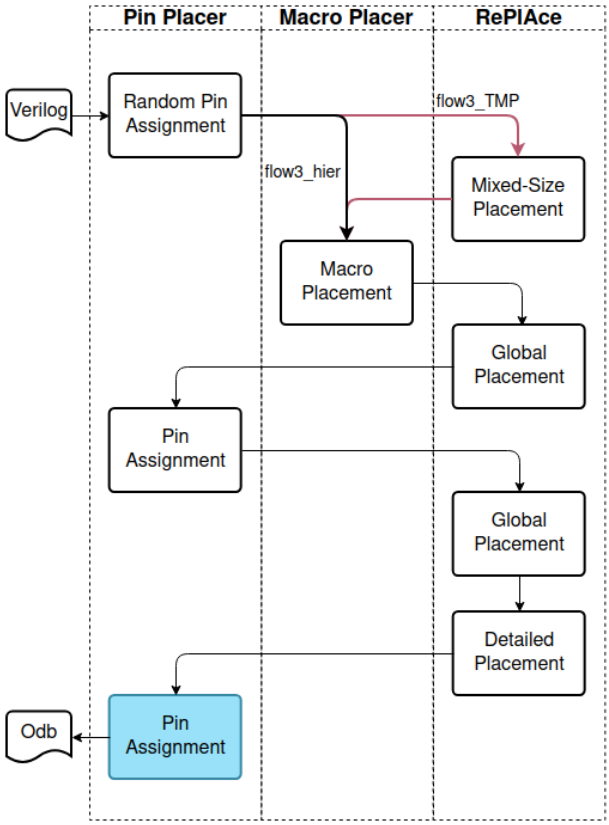
Source: The Author.

The first flow to be examined is called flow2_TMP. The first three steps of flow2_TMP are the same as flow1_TMP. However, an extra call of the Pin Placer is added just after the macro placement so as to perform a non random pin assignment using hungarian matching. After this new step, flow2_TMP follows the same remaining steps of flow1_TMP, continuing with the global placement using RePlAce, but now considering the I/O pins in its execution, as seen in Fig. 5, where the new step is colored green. Flow2_TMP is represented by the red flow line in the start, before converging with flow2_hier.

The proposed flow, flow2_hier, follows the same changes as the ones made on flow2_TMP, i.e., an extra non random pin assignment step is added after the macro placement, and hence the following global placement step is altered so as to consider the I/O pins positions during its execution, as it can be seen in Fig. 5, with the new step is indicated by the green color and flow2_hier being represented by the black flow line before converging with flow2_TMP.

To explore the optimal timing of pin assignment within the floorplanning and placement sequence a third set of flows was created. By adding a pin assignment step after detailed placement made with the RePlAce tool, this third flow tests whether aligning I/O pin locations closer to the final stages of placement can improve routing efficiency and performance. These flows aim to determine if late-stage pin assignment offers better adaptability to the final layout conditions, or if early-stage pin assignment provides more foundational guidance for placement and routing optimization.

Figure 6 – Proposed Floorplanning Flows flow3_TMP and flow3_hier.



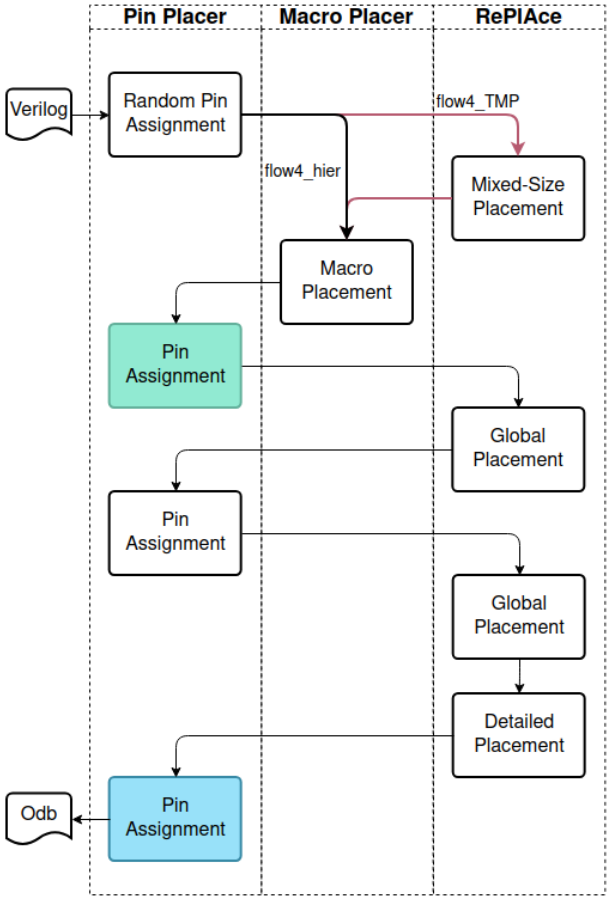
Source: The Author.

The first of these new flows is flow3_TMP and the second is flow3_hier, as seen in Fig. 6. The flowchart is expanded to include the detailed placement, the step following the global placement shown in all previous flowcharts, as well as the new late-stage pin assignment step, colored in blue.

As all of the previously proposed flows add steps in different parts of the floorplan, they do not conflict with each other, allowing for the creation of a final set of flows, flow4_TMP and flow4_hier, combining together the changes of flow2_TMP and flow2_hier with flow3_TMP and flow3_hier. With these new flows lies in the potential for progressively refined pin alignment throughout the placement stages. By incorporating an initial pin assignment after macro placement, the flow provides an early anchor allowing global placement to adjust component positioning relative to these key access points.

Following this with a second pin assignment after detailed placement allows for further optimization, ensuring pin locations align with the nearly finalized layout, thereby reducing routing complexity and enhancing overall design efficiency. This dual-assignment approach aims to capitalize on the benefits of early guidance from pin locations while still providing flexibility to fine-tune these assignments as the layout crystallizes, which could result in a more optimized, adaptable floorplan. These new flows can be seen with Fig. 7, where the new steps are painted green and blue.

Figure 7 – Proposed Floorplanning Flows flow4_TMP and flow4_hier.



Source: The Author.

The summary of the flows can be seen in Table 3, where the first column indicates the flow, and the subcolums of the second column indicate where is the extra Pin

Assignment in the determined flow.

Table 3 – Summary of flow characteristics.

Extra Pin Placement	**Flow1**	**Flow2**	**Flow3**	**Flow4**
After Macro Placement		X		X
After Detailed Placement			X	X

5.1 EXPERIMENTAL RESULTS

The experimental evaluation of the proposed flows used eight different designs available from OpenROAD listed in Table 4. In this table, column 1 gives the names of the circuits, whereas columns 2, 3 and 4 bring the number of cells, the number of macros and the number of I/Os, respectively. The circuits use the FreeDPK45-based open-source NanGate45 enablement. Each circuit was synthesized for each of the eight flows (flow1_TMP, flow2_TMP, flow3_TMP, flow4_TMP, flow1_hier, flow2_hier, flow3_hier, flow4_hier) and successively underwent the remaining steps i.e., detailed placement, global routing and detailed routing. Thus, a total of 64 syntheses were carried out; All executions used an Ubuntu 22.04.4LTS workstation with an Hexacore Intel® Core® i7 8750H CPU and 16GB RAM at 3200MHz. The OpenROAD Flow Scripts version used in the experiments was the commit labeled 9f67f4a (PROJECT, 2024b), and the OpenROAD toolkit version was the commit labeled 57be191. Total Wirelength, Number of Vias and Synthesis Runtime information was gathered from the OpenROAD reports.

The difference between the results obtained by comparing both macro placers in flow1_TMP and flow1_hier was not investigated in this work, for it is known that the new Hierarchical RTL-MP macro placer is still under development and its focused on state-of-the-art circuits with hundreds of macros and a starting hierarchy defined to be further altered and improved, something not present in the test circuits used in this work.

Table 4 – Main Statistics of the Test Circuits.

Circuit	**# cells**	**# macros**	**# I/Os**
ariane136	175K	136	495
ariane133	167K	133	495
swerv_wrapper	96K	28	1416
black_parrot	302K	24	1198
bp_multi	137K	26	1453
bp_be	50K	11	3029
bp_fe	33K	10	2511
tinyRocket	25K	2	269

Table 5 shows the obtained results for the TritonMP flows circuit analysis. Column 1 gives the name of the circuits, Columns 2 to 5 bring the total wirelength results, in the internal OpenROAD generic unit of measurement, Columns 6 to 9 show the results of

numbers of vias, and Columns 10 to 13 display the time, in seconds, taken to run the complete physical synthesis up to finishing the detailed routing. All results are given as percentages of those of flow1. Negative percentages correspond to improvements, whereas positive percentages indicate worsens.

Table 5 – Circuit analysis results for TritonMP flows: flow2, flow3 and flow4 results expressed as flow1 percentages.

Circuit	Total Wirelength				Number of Vias				Synthesis Runtime			
	flow1_TMP	flow2_TMP	flow3_TMP	flow4_TMP	flow1_TMP	flow2_TMP	flow3_TMP	flow4_TMP	flow1_TMP	flow2_TMP	flow3_TMP	flow4_TMP
ariane136	7073k	0.22%	0.00%	0.21%	1337k	-0.09%	0.02%	-0.06%	5530	71.90%	1.73%	10.51%
ariane133	Failed	-	-	-	Failed	-	-	-	Failed	-	-	-
swerv_wrapper	4304k	2.70%	-0.01%	2.71%	931k	0.93%	-0.02%	0.95%	4397	8.99%	0.08%	7.16%
black_parrot	7266k	-2.28%	-0.02%	-2.29%	1304k	0.14%	-0.08%	0.10%	2769	18.71%	0.76%	18.33%
bp_multi	4125k	-1.67%	0.00%	-1.68%	757k	-0.19%	0.04%	-0.13%	1597	16.67%	0.00%	17.24%
bp_be	2673k	-6.17%	0.09%	-6.00%	431k	-1.40%	-0.01%	-1.28%	1152	21.50%	0.14%	15.83%
bp_fe	2010k	-8.42%	0.10%	-8.35%	294k	-3.11%	-0.13%	-2.65%	636	21.18%	-0.87%	21.95%
tinyRocket	646k	6.88%	-0.05%	6.94%	190k	1.16%	0.03%	1.26%	872	-11.06%	1.26%	-14.44%
Average	**-**	**-1.25%**	**0.02%**	**-1.21%**	**-**	**-0.37%**	**-0.02%**	**-0.26%**	**-**	**21.13%**	**0.44%**	**10.94%**
Median	**-**	**-1.67%**	**0.00%**	**-1.68%**	**-**	**-0.09%**	**-0.01%**	**-0.06%**	**-**	**18.72%**	**0.14%**	**15.83%**

Regarding total wirelength, flow2_TMP and flow4_TMP achieve reductions of 1.25% and 1.21%, respectively, while flow3_TMP remains nearly neutral with a minor 0.02% increase. These reductions highlight the benefits of the individual and combined techniques in optimizing interconnect length. For via count, both flow2_TMP and flow4_TMP also show modest improvements, with reductions of 0.37% and 0.26%, respectively, whereas flow3_TMP has a minor impact, only reducing by -0.02%. This suggests that flow2_TMP's approach is slightly more effective in improving layout metrics.

The synthesis runtime analysis offers a contrasting perspective. Flow2_TMP incurs a significant average increase of 21.13% in runtime, indicating a trade-off for its wirelength and via reductions. Flow4_TMP, which combines the methods of flow2_TMP and flow3_TMP, achieves a more balanced runtime overhead of 10.94%. In contrast, flow3_TMP demonstrates the lowest runtime increase at only 0.44%, showcasing its efficiency but with less pronounced improvements in physical metrics.

Outliers like the *tinyRocket* circuit, where flow2_TMP significantly increases wirelength (6.88%), but drastically reduces the runtime (-11.06%), emphasize the variability in flow performance depending on circuit characteristics. Conversely, circuits like *bp_be* and *bp_fe* highlight substantial wirelength reductions (-6.17% and -8.42%, respectively), validating the effectiveness of flow2_TMP.

Overall, while flow4_TMP offers the best balance between optimization and runtime, these results underscore the importance of evaluating trade-offs for specific circuit designs.

Table 6 shows the obtained results for the TritonMP flows circuit analysis. As in the previous table, Column 1 gives the name of the circuits, Columns 2 to 5 bring the total wirelength results, in generic units, Columns 6 to 9 show the results of numbers of vias, and Columns 10 to 13 display the synthesis time, with all results given as percentages of those of flow1.

The results of the hierarchical RTL-MP flows reveal notable differences compared

Table 6 – Circuit analysis results results for Hierarchical RTL-MP flows: flow2, flow3 and flow4 results expressed as flow1 percentages.

Circuit	Total Wirelength				Number of Vias				Synthesis Runtime			
	flow1_hier	flow2_hier	flow3_hier	flow4_hier	flow1_hier	flow2_hier	flow3_hier	flow4_hier	flow1_hier	flow2_hier	flow3_hier	flow4_hier
ariane136	8232k	-1.21%	-0.01%	-1.19%	1549k	-0.10%	0.05%	-0.07%	6714	8.83%	1.77%	9.80%
ariane133	7725k	-1.03%	-0.01%	-1.03%	1473k	-0.43%	0.02%	-0.44%	7401	-7.64%	3.39%	-7.00%
swerv_wrapper	4353k	-1.24%	-0.21%	-1.30%	996k	-0.09%	-0.20%	-0.14%	5372	9.27%	-3.58%	16.50%
black_parrot	9010	2.02%	0.01%	1.98%	2185k	0.08%	0.04%	0.04%	4484	23.37%	0.48%	23.52%
bp_multi	4820k	-0.23%	0.02%	-0.20%	1094k	-0.08%	0.01%	-0.10%	2329	21.52%	1.49%	21.84%
bp_be	3085k	-0.60%	-1.72%	-0.15%	548k	-3.02%	-5.77%	-2.95%	2000	16.24%	-11.57%	26.14%
bp_fe	2332k	-9.34%	-0.37%	-9.23%	353k	-7.66%	-0.34%	-7.69%	1170	-23.05%	-0.49%	-25.38%
tinyRocket	691k	0.11%	-0.07%	0.10%	193k	0.07%	0.07%	-0.05%	746	9.96%	3.69%	10.60%
Average	**-**	**-1.44%**	**-0.30%**	**-1.38%**	**-**	**-1.40%**	**-0.77%**	**-1.43%**	**-**	**7.31%**	**-0.60%**	**9.50%**
Median	**-**	**-0.81%**	**-0.05%**	**-0.62%**	**-**	**-0.09%**	**0.01%**	**-0.12%**	**-**	**9.62%**	**0.99%**	**13.55%**

to the initial flow1_hier, with all flows demonstrating distinct advantages depending on the evaluated metrics. For total wirelength, flow2_hier and flow4_hier achieve average reductions of 1.44% and 1.38%, respectively, underscoring their effectiveness in minimizing routing costs, while flow3_hier exhibits a slight increase of 0.30%. These reductions indicate that flow2_hier and flow4_hier, which focus on optimizing pin assignment in conjunction with hierarchical placement strategies, are well-suited for improving interconnect efficiency.

In terms of via count, flow4_hier achieves the largest average reduction of 1.43%, followed closely by flow2_hier at 1.40%, while flow3_hier again has a smaller impact with a reduction of only 0.77%. This consistency across wirelength and via count suggests that flow4_hier effectively combines the strengths of the other two flows while maintaining minimal design complexity, particularly benefiting circuits such as *bp_fe*, which shows a remarkable via count reduction of 7.69%.

The synthesis runtime metric presents a different story, with flow2_hier incurring an average increase of 7.31%, flow4_hier at 9.50%, and flow3_hier demonstrating a rare reduction of 0.60%. Notably, the outlier *bp_fe* achieves significant runtime improvements across all hierarchical flows, with reductions as high as 25.38% for flow4_hier, emphasizing the role of specific circuit characteristics in runtime performance. In contrast, circuits like *black_parrot* experience runtime increases of over 23%, which could be attributed to the complexity of balancing macro placement and pin optimization.

Overall, the hierarchical flows validate the benefits of combining macro placement strategies with advanced pin assignment techniques. Flow4_hier consistently emerges as the most balanced option, achieving substantial physical optimizations at a moderate runtime cost.

Table 7 shows the obtained results for the TritonMP flows timing analysis. Column 1 gives the name of the circuits, Columns 2 to 5 bring the worst slack, Columns 6 to 9 show the results of *Total Negative Slack* (TNS), and Columns 10 to 13 displays the number of *Design Rule Violation* (DRV) errors during the synthesis process. All results are given as percentages of those of flow1. Negative percentages correspond to improvements, whereas positive percentages indicate worsens. The circuit *ariane_136* is not shown in this table, as it had no negative slack and no *DRV*.

Table 7 – Timing analysis results for TritonMP flows: flow2, flow3 and flow4 results expressed as flow1 percentages.

Circuit	Worst Slack				Total Negative Slack				DRV			
	flow1_TMP	flow2_TMP	flow3_TMP	flow4_TMP	flow1_TMP	flow2_TMP	flow3_TMP	flow4_TMP	flow1_TMP	flow2_TMP	flow3_TMP	flow4_TMP
ariane133	Failed	-	-	-	Failed	-	-	-	Failed	-	-	-
swerv_wrapper	-0.287	21.53%	-3.00%	28.77%	-91.497	116.53%	5.76%	115.76%	955	9.95%	-3.56%	10.37%
black_parrot	-2.988	-10.81%	0.11%	-10.77%	-2.988	-10.81%	0.11%	-10.77%	1	0%	0%	0%
bp_multi	-3.796	-4.95%	-0.08%	-5.20%	-3.796	-4.95%	-0.08%	-5.20%	1	0%	0%	0%
bp_be	-0.078	112.89%	43.03%	181.85%	-6.521	108.58%	48.38%	191.72%	102	5.88%	4.90%	6.86%
bp_fe	-0.014	-74.51%	108.39%	171.63%	-0.027	-86.57%	112.95%	468.70%	2	-50.00%	50.00%	300.00%
tinyRocket	-0.354	-15.30%	-0.08%	-14.08%	-156.529	-6.84%	-1.29%	-5.82%	616	-4.71%	0.16%	-5.03%
Average	**-**	**4.81%**	**24.74%**	**58.70%**	**-**	**19.32%**	**27.64%**	**125.73%**	**-**	**-6.48%**	**8.58%**	**52.03%**
Median	**-**	**-7.88%**	**0.02%**	**11.79%**	**-**	**-5.89%**	**2.94%**	**55.28%**	**-**	**0.00%**	**0.08%**	**3.43%**

For the worst slack metric, flow2_TMP demonstrates a modest average worsening of 4.81%, whereas flow3_TMP and flow4_TMP show substantial degradations of 24.74% and 58.70%, respectively. The significant worst slack deterioration in flow4_TMP suggests that its combined optimizations for wirelength and via count may come at the cost of reduced timing margins. The main cause for this elevated value is the circuit *bp_be*, where the worse slack increased by 112.89%, 43.08% and 181.85% with flow2_TMP, flow3_TMP and flow4_TMP respectively, but considering the initial worst slack in flow1_TMP for this circuit is low, the increase in time is not as expressive as the 4.95% reduction in *bp_multi* and 10.81% reduction in *black_parrot*, both with flow2_TMP.

TNS results reveal a more negative. Flow2_TMP achieves a 19.32% average worsening, and flow3_TMP a slightly higher average worsening of 27.64%. However, flow4_TMP exhibits a 125.73% increase, indicating timing degradation. Considering the original values for TNS, *swerv_wrapper* shows the worst performance, increasing in all flows, while *tinyRocket* shows a considerable reduction of 6.84% from the largest TNS. The DRV analysis highlights that flow2_TMP and flow3_TMP offer reductions of 6.48% and 8.58%, respectively, whereas flow4_TMP has a significant DRV increase of 52.03%.

Outlier circuits, such as bp_fe, reveal interesting behaviors. While flow2_TMP achieves significant improvements in worst slack and TNS (-74.51% and -86.57%, respectively), flow4_TMP suffers from major degradations (171.63% and 468.70%, respectively).

Table 8 shows the obtained results for the TritonMP flows timing analysis. As in the previous table, Column 1 gives the name of the circuits, Columns 2 to 5 bring the worst slack, Columns 6 to 9 show the results of TNS, and Columns 10 to 13 displays the number of DRV errors during the synthesis process, with all results given as percentages of those of flow1. Like in the previous table, circuit *ariane_136* is not shown in the table due to having no negative slack nor *DRV*.

For worst slack, flow2_hier achieves the most notable improvement, with an average reduction of 26.45%, followed by flow4_hier at 18.43%. Flow3_hier shows the smallest average reduction of 7.57%. While most of the circuits show improvements, the circuit with the largest worst slack with Hierarchical RTL-MP, *bp_multi*, shows a 2.94% increase of its worst slack with flow2_TMP, having the worst performance in this circuit compared to the other flows.

Table 8 – Timing analysis results for Hierarchical RTL-MP flows: flow2, flow3 and flow4 results expressed as flow1 percentages.

Circuit	Worst Slack				Total Negative Slack				DRV			
	flow1_hier	flow2_hier	flow3_hier	flow4_hier	flow1_hier	flow2_hier	flow3_hier	flow4_hier	flow1_hier	flow2_hier	flow3_hier	flow4_hier
ariane133	-0.093	-23.47%	14.06%	14.22%	-38.343	-48.99%	33.32%	41.01%	952	-23.53%	17.54%	26.68%
swerv_wrapper	-0.648	14.05%	3.98%	20.38%	-601.177	3.97%	-4.32%	22.36%	1423	15.25%	-2.74%	-0.63%
black_parrot	-0.647	-16.95%	0.51%	-16.86%	-0.647	-16.95%	0.51%	-16.86%	1	0%	0%	0%
bp_multi	-1.928	2.94%	0.22%	1.37%	-1.928	2.94%	0.22%	1.37%	1	0%	0%	0%
bp_be	-0.871	-61.06%	-53.25%	-52.31%	-89.903	-61.12%	-56.93%	53.25%	144	-11.81%	-22.92%	-18.06%
bp_fe	-0.314	-92.47%	-14.34%	-89.37%	-12.498	-99.07%	-6.26%	-99.12%	177	-94.29%	0.56%	-95.48%
tinyRocket	-0.273	-8.20%	-4.16%	-6.44%	-112.223	-3.62%	-2.16%	-2.11%	658	-6.08%	-0.15%	-5.93%
Average	**-**	**-26.45%**	**-7.57%**	**-18.43%**	**-**	**-31.83%**	**-5.09%**	**-15.23%**	**-**	**-17.30%**	**-1.10%**	**-13.34%**
Median	**-**	**-16.95%**	**0.22%**	**-6.44%**	**-**	**-16.95%**	**-2.16%**	**-2.11%**	**-**	**-6.08%**	**0.00%**	**-0.63%**

TNS results are more significative with Hierarchical RTL-MP. Flow2_hier achieves the most significant improvement, with a reduction of 31.83%, followed by flow4_hier (-15.23%). Flow3_hier offers a smaller reduction of 5.09%, which still indicates an improvement over the baseline flow1_hier. For *swerv_wrapper*, the circuit with the largest TNS, flow3_hier is the only to offer a reduction, reducing by 4.32%, while flow2_hier increases the value by 3.97% and flow4_hier increases by 22.36%.

The DRV analysis reveals consistent reductions across all hierarchical flows, with flow2_hier achieving the largest reduction (-17.30%), followed by flow4_hier (-13.34%). Flow3_hier shows the smallest improvement (-1.10%), but its the only flow to reduce the number of DRV in the circuit with the most of them, *swerv_wrapper*. Overall, flow2_hier demonstrates to be better performing in most circuits, followed by flow4_hier, but few specific circuits benefit more in the timing aspect utilizing flow3_hier, like *bp_multi* and *swerv_wrapper*.

6 CONCLUSION

In this work, it was investigated alternative floorplanning flows for VLSI physical design using the OpenROAD toolkit, focusing on optimizing macro placement and improving pin assignment through an additional non-random placement step. By comparing the proposed flows (flow2_TMP, flow3_TMP, and flow4_TMP) and their hierarchical versions (flow2_hier, flow3_hier, and flow4_hier) to their respective original versions (flow1_TMP and flow1_hier) across various test circuits, we demonstrated the potential benefits of these enhanced methods. In the TritonMP-based flows, flow2_TMP and flow4_TMP reduced total wirelength by 1.25% and 1.21%, respectively, while maintaining small reductions in via counts. However, flow3_TMP provided minimal improvements, emphasizing the importance of flow design in achieving consistent gains. Similarly, in the hierarchical flows, flow2_hier and flow4_hier achieved average wirelength reductions of 1.44% and 1.38%, respectively, and also exhibited via reductions, particularly benefiting circuits like *bp_fe*, which saw a 7.69% decrease in via count.

The hierarchical flows demonstrated slightly lower runtime overheads compared to their TritonMP counterparts, with flow2_hier incurring an average runtime increase of 7.31%, whereas flow2_TMP increased runtime by 21.13%. Notably, flow3_hier showed minimal runtime increase of 0.60%, albeit with less significant physical optimizations. Outliers, such as the tinyRocket and *bp_fe* circuits, highlighted the variability in flow performance depending on circuit characteristics, with runtime reductions of up to 25.38% observed for flow4_hier on *bp_fe*.

These results underscore the trade-off between improved layout quality and increased computational effort. These findings validate the effectiveness of incorporating non-random pin assignment and hierarchical placement strategies in enhancing the floorplanning process. They also highlight the adaptability of open-source EDA tools like OpenROAD for advancing VLSI design methodologies while addressing specific design challenges.

6.1 FUTURE WORK

Future research could focus on developing an advanced macro placer that integrates pin assignment directly into its optimization steps, enabling simultaneous optimization of macro placement and interconnect routing. By embedding pin assignment within the macro placement process, such a tool could better address the interdependencies between these tasks, potentially leading to more significant reductions in wirelength and via count without the need for a separate pin assignment step. This integrated approach could also streamline the overall design flow, achieving higher-quality layouts.

REFERENCES

AGNESINA, Anthony et al. AutoDMP: Automated DREAMPlace-based Macro Placement. In: PROCEEDINGS of the 2023 International Symposium on Physical Design. Virtual Event, USA: Association for Computing Machinery, 2023. (ISPD '23), p. 149–157. DOI: 10.1145/3569052.3578923. Disponível em: https://doi.org/10.1145/3569052.3578923.

BANDEIRA, Vitor et al. Fast and Scalable I/O Pin Assignment with Divide-and-Conquer and Hungarian Matching. In: 2020 18th IEEE International New Circuits and Systems Conference (NEWCAS). [S.l.: s.n.], 2020. P. 74–77. DOI: 10.1109/NEWCAS49341.2020.9159791.

CHENG, Chung-Kuan et al. RePlAce: Advancing Solution Quality and Routability Validation in Global Placement. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 38, n. 9, p. 1717–1730, 2019. DOI: 10.1109/TCAD.2018.2859220.

FONTANA, Tiago Augusto et al. ILP-Based Global Routing Optimization with Cell Movements. In: 2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI). [S.l.: s.n.], July 2021. P. 25–30. DOI: 10.1109/ISVLSI51109.2021.00016.

HU, Kai-Shun; YANG, Ming-Jen, et al. ICCAD-2020 CAD contest in routing with cell movement. In: 2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD). [S.l.: s.n.], 2020. P. 1–4. DOI: 10.1145/3400302.3415738.

HU, Kai-Shun; YU, Tao-Chun, et al. 2021 ICCAD CAD Contest Problem B: Routing with Cell Movement Advanced: Invited Paper. In: 2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD). [S.l.: s.n.], 2021. P. 1–5. DOI: 10.1109/ICCAD51958.2021.9643568.

KAHNG, A.; LIENIG, J., et al. **"VLSI Physical Design: From Graph Partitioning to Timing Closure"**. [S.l.]: Springer Netherlands, 2011. DOI: 10.1007/978-90-481-9591-6.

KAHNG, Andrew B; VARADARAJAN, Ravi; WANG, Zhiang. Hier-RTLMP: A hierarchical automatic macro placer for large-scale complex IP blocks. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, IEEE, 2023.

KAHNG, Andrew B. Leveling Up: A Trajectory of OpenROAD, TILOS and Beyond. In: PROCEEDINGS of the 2022 International Symposium on Physical Design. Virtual Event, Canada: Association for Computing Machinery, 2022. (ISPD '22), p. 73–79. DOI: 10.1145/3505170.3511479.

_____. Solvers, Engines, Tools and Flows: The Next Wave for AI/ML in Physical Design. In: PROCEEDINGS of the 2024 International Symposium on Physical Design. Taipei, Taiwan: Association for Computing Machinery, 2024. (ISPD '24), p. 117–124. DOI: 10.1145/3626184.3635277.

KUHN, H. W. The Hungarian method for the assignment problem. **Naval Research Logistics Quarterly**, v. 2, n. 1-2, p. 83–97, 1955. DOI: 10.1002/nav.3800020109.

LAUNG-TERNG WANG, Yao-Wen Chang; CHENG, Kwang-Ting (Tim). **Electronic Design Automation: Synthesis, Verification, and Test**. [S.l.]: Morgan Kaufmann, 2008.

LIN, Yibo et al. DREAMPlace: Deep Learning Toolkit-Enabled GPU Acceleration for Modern VLSI Placement. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 40, n. 4, p. 748–761, 2021. DOI: 10.1109/TCAD.2020.3003843.

LU, Jingwei et al. ePlace: Electrostatics based placement using Nesterov's method. In: 2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC). [S.l.: s.n.], 2014. P. 1–6. DOI: 10.1145/2593069.2593133.

MARKOV, Igor L. Reevaluating Google's Reinforcement Learning for IC Macro Placement. **Commun. ACM**, Association for Computing Machinery, New York, NY, USA, v. 67, n. 11, p. 60–71, Oct. 2024. ISSN 0001-0782. DOI: 10.1145/3676845.

MARKOV, Igor L.; ADYA, Saurabh N. Fixed-Outline Floorplanning: Enabling Hierarchical Design. **IEEE Trans. on VLSI Systems**, 2003.

MIRHOSEINI, Azalia et al. A graph placement methodology for fast chipdesign. **Nature**, v. 594, n. 7862, p. 207–212, June 2021. ISSN 1476-4687. DOI: 10.1038/s41586-021-03544-w.

NETTO, Renan et al. Ophidian: an Open-Source Library for Physical Design Research and Teaching. In: FIRST Workshop on Open-Source EDA Technology. [S.l.: s.n.], 2018. Disponível em: https://woset-workshop.github.io/PDFs/2018/a25.pdf.

PROJECT, The OpenROAD. **OpenROAD**. [S.l.]: GitHub, 2024. https://github.com/The-OpenROAD-Project/OpenROAD.

_____. **OpenROAD Flow Scripts**. [S.l.]: GitHub, 2024. https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts.

PU, Yuan et al. IncreMacro: Incremental Macro Placement Refinement. In: PROCEEDINGS of the 2024 International Symposium on Physical Design. Taipei, Taiwan: Association for Computing Machinery, 2024. (ISPD '24), p. 169–176. DOI: 10.1145/3626184.3633321. Disponível em: https://doi.org/10.1145/3626184.3633321.

VIDAL-OBIOLS, Alex et al. Multilevel Dataflow-Driven Macro Placement Guided by RTL Structure and Analytical Methods. **IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems**, v. 40, n. 12, p. 2542–2555, 2021. DOI: 10.1109/TCAD.2020.3047724.

WESTE, Neil H.; HARRIS, David Money. **CMOS VLSI Design a Circuit and Systems Perspective**. [S.l.]: Addison-Wesley, 2011.

YOSYS HQ. **Yosys Open SYnthesis Suite**. [S.l.]: GitHub, 2024. https://github.com/YosysHQ/yosys.

Appendix

APPENDIX A – FLOWS SOURCE CODE

The flows proposed and utilized for this work are available in the following open repository: https://github.com/rafaelmoresco/ScriptsTCC

APPENDIX B – SBC PAPER

The following pages have an alternative version of this work for publication following the model of the Sociedade Brasileira de Computação.

Evaluating the Impact of Pin Assignment Order in VLSI Circuit Floorplanning Outcomes

Rafael Moresco VIeira[1], **José Luís Almada Güntzel**[1]

[1]Universidade Federal de Santa Catarina – Departamento de Informática e Estatística

Abstract. *The early stages of the physical design of VLSI (Very-Large-Scale Integration) circuits, referred to as floorplanning, are critical for achieving quality layouts as they directly affect subsequent design stages. A typical floorplanning flow consists of pin assignment, macro placement, and power planning. However, pin assignment and macro placement are interdependent steps, and their execution order significantly influences floorplanning outcomes, which is why they are often alternated. For example, the standard floorplanning flow in the open-source OpenROAD platform performs an initial random pin assignment followed by macro and global placement, concluding with an extra pin assignment step. In this approach, pin assignment does not directly influence the macro and global placement results, potentially missing optimization opportunities. This work explores new enhanced floorplanning flows within integrated circuit (IC) synthesis using OpenROAD by introducing an additional non-random pin assignment step. The proposed flows are tested with two macro placers available in OpenROAD. Experimental results using FreePDK45 test circuits demonstrated that the proposed flows achieved average reductions in wirelength and via count of 1.25% and 0.37%, respectively, with TritonMP, and 1.44% and 1.43%, respectively, with Hierarchical RTL-MP. Specific circuits showed reductions in wirelength and via count of up to 9.34% and 7.69%, respectively. These results underscore the potential for further optimizations during the floorplanning stage, highlighting the importance of addressing pin assignment.*

1. Introduction

The technological advancements have allowed the fabrication of Integrated Circuits (ICs) featuring over a billion transistors. The rising complexity of these designs calls for the extensive use of hierarchical design, functional blocks and an increasing use of Intellectual Property (IP) blocks. This trend has raised the importance of floorplanning in determining the quality of a Very-Large-Scale Integration (VLSI) design [Laung-Terng Wang and Cheng 2008]. With the continuous reduction of technology nodes size, there is also an increasing demand to improve placement and routing, as the ever so smaller transistors allow for a greater circuit density that combined with more metal layers in more recent technology nodes make the total wirelength resistance and capacitance the main cause for delays and a major part of switching power loss [Weste and Harris 2011]. Due to those issues, there is a prominent need for good quality floorplan that could lead to high quality placement and routing solutions.

In recent years many new approaches for VLSI optimization have been introduced in academia, ranging from an increase in AI-based design optimization for all stages of the physical synthesis to the introduction of changes to the traditional steps. This

latter approach was subject of ICCAD CAD Contests 2020 [Hu et al. 2020] and 2021 [Hu et al. 2021] and highlighted that there are many optimization opportunities arriving from modifying the synthesis flow, either adding extra steps or by altering the behavior of the existing ones through the integration of new features.

With the growing demand from the Electronic Design Automation (EDA) industry, there has been a rise in open-source movements led by academia aimed at disseminating physical design knowledge. Over the years, several tools specific to each step of the synthesis flow have been developed. To support the rise of those tools, open-source libraries such as the Ophidian library developed at the Federal University of Santa Catarina [Netto et al. 2018] appeared, until more comprehensive projects covering all stages began to emerge. Among these projects, OpenROAD stands out, a global initiative supported by several universities and companies, headquartered at UC San Diego, California [Kahng 2022], which serves as the foundational infrastructure for the development of this work.

This work investigates how the addition of an extra step of pin assignment impacts the resulting layouts of physical synthesis. To perform this investigation, the open-source EDA tool-kit OpenROAD [Kahng 2022] was chosen as our test case. We explain how the standard OpenROAD flow works, presenting its two variations due to the existence of two available macro placers, TrintonMP and Hierarchical RTL-MP [Kahng et al. 2023], and compare with three proposed investigative flows using test designs available within OpenROAD.

1.1. Motivation and Objectives

With modern IC designs growing increasingly complex, featuring billions of transistors and densely packed functional blocks, even slight inefficiencies in floorplanning can significantly impact performance, power consumption, and design feasibility. Considering how recent works add new steps in the global routing stage executing functions associated with global placement [Fontana et al. 2021], there is still much to explore in the interactions between synthesis steps. This work, therefore, seeks to explore the potential of an altered floorplanning flow by adding an extra pin assignment step to improve routing efficiency and layout compactness. Using the OpenROAD platform as a test case, this work aims to identify and quantify the benefits of structured pin assignment, contributing to more optimized and accessible VLSI design flows, especially within the growing field of open-source EDA tools.

General Objectives

The general objective of this work is to investigate and introduce a new VLSI floorplanning flow that considers the dependency relation between the steps of macro placement, pin assignment and global placement in the physical synthesis process, utilizing the addition of extra pin assignment steps in order to enhance layout quality, utilizing the open-source toolkit OpenROAD as a test case. The experimental results will be compared with those generated by the standard flows avaliable within OpenROAD.

Specific Objectives

- Analyze the impact of pin assignment in relation to macro placement on the floorplanning outcomes by introducing additional, pin assignment steps within the OpenROAD flow;

- Compare the effectiveness of the proposed pin assignment modifications with different macro placers, TritonMP and Hierarchical RTL-MP, to determine improvements in layout efficiency, focusing on metrics like wirelength reduction and via minimization;
- Measure the synthesis runtime implications of the proposed flows and assess the trade-offs between layout quality improvements and computational resource demands;
- Conduct experimental evaluations on diverse OpenROAD test circuits, quantifying the benefits and limitations of the proposed flows and validating the feasibility of these approaches for integration in open-source VLSI design workflows.

2. Background

This section presents the necessary theoretical knowledge to understand the current work.

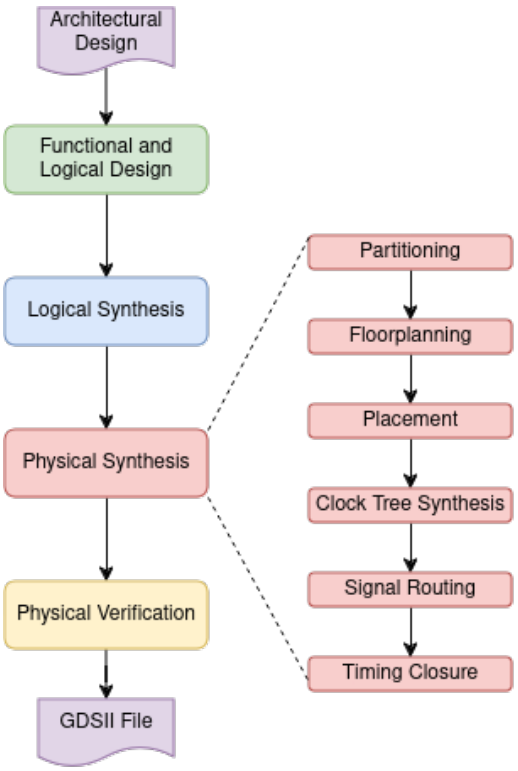
2.1. Physical Synthesis

The RTL-to-GDSII synthesis flow is a critical process in VLSI circuit design, serving as the bridge from abstract functional descriptions to the detailed physical layouts necessary for manufacturing. The Fig. 1 shows the complete RTL-to-GDSII flow, highlighting the presence of the Physical Synthesis steps with red colored blocks. This flow begins at the Register Transfer Level (RTL), where designers use Hardware Description Language (HDL) like Verilog or VHDL to define the circuit's functional and timing behavior at a high level. From here, the synthesis process translates this high-level design into a gate-level netlist, mapping RTL logic to specific gates using standard cells from a given technology library [Weste and Harris 2011]. This synthesis step optimizes the design to meet predefined constraints, such as minimizing area or power and ensuring timing closure, which is crucial for high-speed circuits.

Following synthesis, the flow advances to physical design, where the gate-level netlist undergoes a series of steps to create a manufacturable layout. Physical design includes placement, where cells are arranged on the chip 2D surface so as to minimize both area usage and interconnect delay. After placement, global and detailed routing phases establish the precise connections between cells, with a focus on minimizing wire length, reducing crosstalk, and ensuring signal integrity. According to [Kahng et al. 2011], this stage requires iterative optimization to ensure timing closure, particularly as device sizes shrink and timing constraints grow more stringent. Tools are employed to refine the layout until timing closure is achieved, ensuring that all paths meet their designated timing requirements.

Timing analysis and iterative optimization steps follow placement and routing to ensure that the design meets timing requirements, known as *timing closure*. Modern EDA tools employ sophisticated algorithms to achieve this, as timing constraints become increasingly challenging in smaller technology nodes. The final stage of the RTL-to-GDSII flow includes Design Rule Check (DRC) and Layout-Versus-Schematic (LVS) verification, which confirm that the layout complies with manufacturing requirements and matches the intended design specifications. The output, the GDSII file, contains all the geometric and layer information needed for photolithographic fabrication, making it the final product of the digital design flow [Weste and Harris 2011].

Figure 1. Academic RTL-to-GDSII Synthesis Flow.



2.2. Floorplan

Floorplanning plays a crucial role in chip layout, specially in the hierarchical approach to module-based design. It offers preliminary feedback to assess architectural choices, predict chip areas, and estimate delays and congestion due to interconnections. As fabrication technology evolves, design complexity increases as more transistors are integrated in a single chip. To address such growing complexity, hierarchical design practices and IP modules are extensively employed. Consequently, floorplanning has become more important than ever for ensuring the quality of VLSI designs [Laung-Terng Wang and Cheng 2008].

Traditionally, the floorplanning step is further divided into macro placement, pin assignment, and power planning [Kahng et al. 2011]. Among these, macro placement is particularly vital because it lays the foundation for the overall layout quality and directly influences key design metrics such as wirelength, congestion, and timing closure. The process of macro placement ensures that each block generated during the partitioning stage is assigned a position and shape within the floorplan, aiming to minimize wirelength and balance the area usage, which is crucial for efficient routing and timing optimization. In modern VLSI designs, where circuits consist of hundreds to thousands of macros, effective placement becomes even more important. Poor macro placement can lead to long interconnects, resulting in increased signal delay and higher parasitic capacitances, which degrade performance and power efficiency. Furthermore, congestion in certain regions due to inefficient placement can create bottlenecks during routing, making it harder to meet design closure.

Although many advanced algorithms and tools have been developed to automate macro placement, most designers still rely on handcrafted placements for high-performance chips to fine-tune critical regions of the design [Kahng et al. 2023]. However, current Multiprocessor System-on-Chip (MPSoC) may have thousands of macros and IP blocks, rendering manual macro placing unfeasible. In such scenario, macro placement can exploit design hierarchy in such a way that larger macros or modules are placed before smaller ones, ensuring that major components like memory blocks, processing units, or high-speed interfaces are optimally positioned.

During the pin assignment step, each incoming and outgoing signal is assigned to a specific pin location, looking to enhance the overall performance of the design. The primary goal of pin assignment is to optimize the placement of I/O pins, as it directly affects the efficiency of signal routing across the chip. By carefully assigning pins, designers can improve routability, minimize the total wirelength, and reduce the number of vias, which are critical factors in achieving a more compact and efficient design. Poor pin placement can lead to long, convoluted routing paths that increase resistance, capacitance, and signal delay, potentially causing timing violations and negatively impacting the overall chip performance. According to [Kahng et al. 2011] the ideal moment for the pin assignment is before the macro placement, with the locations being updated during and after macro placement. In the power planning step, the ground and power nets are routed in dedicated metal layers, usually the upper ones.

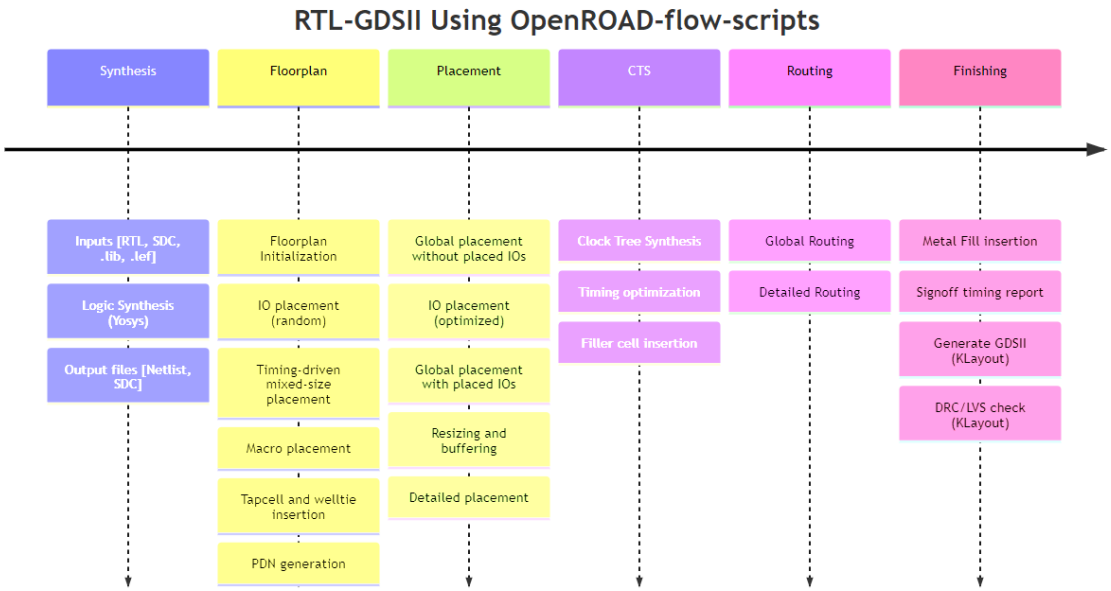
2.3. OpenROAD

With the growing demand in the EDA industry, there has been a surge of open-source movements led by academia aimed at disseminating knowledge in this field. Among these projects, OpenROAD stands out, an initiative supported by various universities and companies worldwide, headquartered at UC San Diego, California [Kahng 2022], which serves as the foundational infrastructure for the development of this work.

OpenROAD, launched in 2019, has the mission to develop a fully open platform, where the entire synthesis process would be completed within 24 hours without human intervention [Kahng 2022]. To achieve this goal, OpenROAD is available in two versions: the standalone OpenROAD tool, where all commands are manually applied, designed to support the development of new tools, and a version called OpenROAD Flow Scripts, which consists of a set of scripts that automatically execute all steps of the tool, made for ICs designers. As shown in Fig. 2, the flow implemented in OpenROAD resembles the theoretical flow identified by [Kahng et al. 2011], but it presents certain modifications and different nomenclatures.

Performing the entire RTL-to-GDSII synthesis flow, OpenROAD [Kahng 2022] is composed of several individual tools with a common interface and data types. For the floorplan step, there are four essential tools: Pin Placer, TritonMP, Hierarchical RTL-MP and RePlAce. Introduced by [Bandeira et al. 2020], the Pin Placer employs a divide-and-conquer strategy combined with Hungarian matching [Kuhn 1955] to achieve efficient pin assignment. This method divides the I/O pin assignment problem into smaller, manageable subproblems, which reduces computational complexity and allows for parallel processing, resulting in faster runtime and scalability for large designs. The Hungarian matching algorithm, capable of solving assignment problems optimally [Kuhn 1955], is

Figure 2. OpenROAD's Synthesis Flow.



then used within each subregion to assign pins while minimizing total wirelength. The tool also offers alternative assignment methods, such as simulated annealing, which is useful for exploring a wider solution space at the cost of increased computation time. By balancing precision with computational efficiency, the Pin Placer tool aims to provide a flexible and scalable solution for pin assignment in modern, complex integrated circuits.

The macroplacer TritonMP was developed utilizing an implementation of ParquetFP, an open-source floorplanning tool introduced by [Markov and Adya 2003]. ParquetFP primarily focuses on fixed-outline floorplanning, an approach that is especially relevant for hierarchical Application Specific Integrated Circuit (ASIC) and System-on-Chip (SoC) designs. Unlike classical floorplanning methods, which minimize area and wirelength without specific layout boundaries, fixed-outline floorplanning mandates that the layout conforms to a predetermined outline, making it more applicable to real-world designs where chip dimensions are constrained. To address the increased complexity of fixed-outline constraints, ParquetFP incorporates advanced objective functions within its simulated annealing framework. It uses wirelength minimization based on the Half Perimeter Wirelength (HPWL) metric and aspect ratio adjustments to handle varying block shapes effectively. ParquetFP also introduces slack-based moves, which allow for local adjustments to minimize wirelength while maintaining critical path constraints. This combination of techniques makes ParquetFP highly effective for both outline-free and fixed-outline contexts, providing scalable, high-quality floorplanning solutions suitable for the hierarchical design methodologies employed in modern VLSI layouts

Hierarchical RTL-MP is a sophisticated hierarchical macro placer developed for OpenROAD to handle the increasing complexity and scale of VLSI designs [Kahng et al. 2023]. With the rise of auto-generated RTL, particularly in areas like machine learning accelerators, the number of macros can reach several hundred in a single design, making traditional peripheral placement methods unfeasible. Unlike previous

macro placers that often arranged macros along the periphery, Hierarchical RTL-MP can place macros within the core of the layout, accommodating large macro numbers and better maintaining design dataflow. The tool employs a multi-level hierarchical approach, transforming logical hierarchies from the RTL into physical hierarchies through a novel autoclustering technique. This technique groups macros into clusters based on design hierarchy and dataflow, creating physical clusters that mimic the logical relationships. Additionally, Hierarchical RTL-MP uses a shaping engine to determine allowable cluster shapes, which it refines through a bottom-up and top-down process to optimize floorplan utilization and routability. This enables more efficient macro placements that align with critical timing paths, support power grid generation, and minimize wirelength. Empirical tests by Hierarchical RTL-MP authors have shown that Hierarchical RTL-MP outperforms prior placements by reducing timing violations and runtime, making it a valuable addition to the OpenROAD toolkit for complex IP blocks.

RePlAce is a mixed-size placer developed to enhance solution quality and address routability challenges in global placement [Cheng et al. 2019]. It builds upon ePlace [Lu et al. 2014], a previous analytical placer utilizing an eletrostatic model, by implementing new techniques that improve both placement quality and routability validation. RePlAce leverages a density function that incorporates local area overflow, allowing it to address congestion at a finer granularity, per placement bin. This approach enables localized adjustments rather than globally applied density penalties, thus preserving overall wirelength while effectively managing high-density areas.

To further optimize placement quality, RePlAce integrates a dynamic step size adaptation method that adjusts optimization effort based on the design's placement state, improving efficiency without increasing runtime significantly. Additionally, RePlAce includes a routability-driven component, which estimates congestion early in the flow and performs cell inflation in congested regions to avoid hotspots. This helps in producing layouts with minimal routing congestion. RePlAce achieves notable improvements in HPWL and routability across various benchmarks, making it a robust tool for tackling the challenges of modern, large-scale VLSI designs.

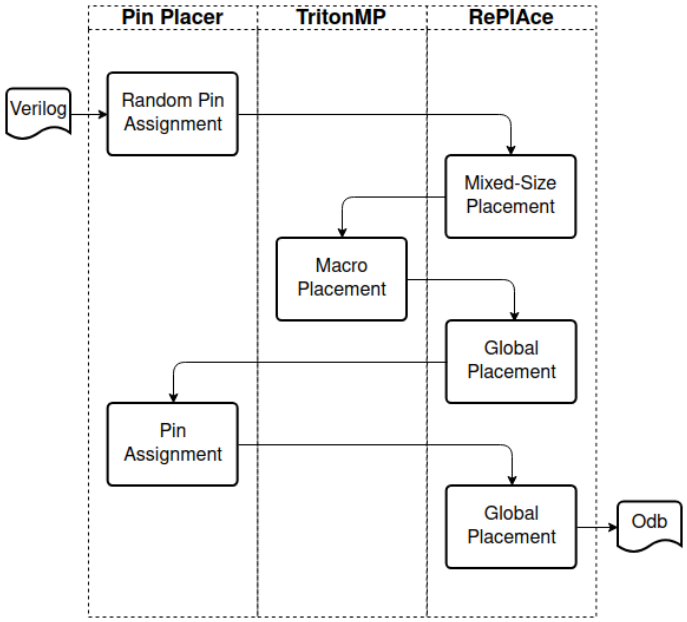
The input to the floorplanning step in OpenROAD is a verilog netlist generated by the third party logic synthesis tool Yosys [Yosys HQ 2024]. This file is loaded and converted into an internal file type. Then, floorplanning starts. The final output of the floorplanning is a proprietary file called ODB, utilized to transfer information from OpenROAD's internal database.

Fig. 3 shows a simplified flowchart of the OpenROAD floorplanning flow using TritonMP as macro placer, while Fig. 4 shows a similar flowchart using the newer Hierarchical RTL-MP as macro placer, hereinafter referred to as flow1_TMP and flow1_hier, respectively. Each column in the flow chart identifies the tool that executes the step.

The first step of flow1_TMP corresponds to randomly assigning the I/O pins. The second step is a timing driven mixed-size placement using RePlAce, where the macro blocks get an initial placement, followed by the refining macro placement with TritonMP. After the macro placement, tapcells and wellties are inserted, and the power delivery network is routed. Although OpenROAD classifies this as the end of the floorplanning, the pin assignment is not finished. After floorplanning, OpenROAD invokes a global place-

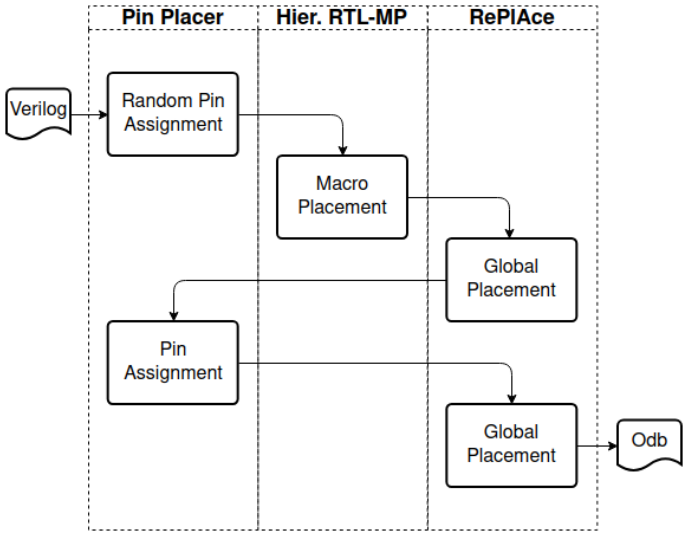
ment ignoring the I/O pins, as they were previously randomly placed. Being classified as part of placement, the definitive pin assignment is made after the previous global placement step, and to refine the results, a new global placement step is performed, but now considering the pin assignment.

Figure 3. Original Floorplanning Flow flow1_TMP.



The flow using Hierarchical RTL-MP as macro placer, flow1_hier, starts by randomly assigning the I/O pins. But differently from flow1_TMP, in flow1_hier there is no mixed-size placement using RePlAce. Instead, it goes directly to the macro placement step with Hierarchical RTL-MP. The rest of flow1_hier undergoes the same steps as flow1_TMP, with a global placement ignoring the randomly placed I/O pins positions being the next step, followed by a definitive I/O placement and a final global placement.

Figure 4. Original Floorplanning Flow flow1_hier.



3. Related Work

This section present the most recent works on floorplanning techniques.

3.1. DREAMPlace

The DREAMPlace framework, presented by [Lin et al. 2021], is a GPU-accelerated tool designed to enhance the speed and scalability of the VLSI placement process by leveraging deep learning techniques. Traditional analytical placement methods, though capable of high-quality results, often suffer from extended runtimes due to the computational complexity involved in optimizing large-scale designs. DREAMPlace addresses this issue by framing placement as a neural network training problem, which enables efficient gradient calculations for wirelength and density through GPU acceleration. Utilizing PyTorch, the framework achieves up to 30× speed improvement over multi-threaded CPU counterparts like RePlAce, without compromising placement quality. This method not only accelerates global placement tasks but also ensures legal placement with minimized overlaps, making it particularly suitable for designs with millions of cells. DREAMPlace has paved the way for integrating advanced AI tools in EDA, setting a strong foundation for further GPU-accelerated VLSI design methodologies .

3.2. AutoDMP

The work by [Agnesina et al. 2023] explores recent advances in VLSI macro placement by leveraging the DREAMPlace [Lin et al. 2021] GPU-accelerated placer combined with machine learning-driven optimization strategies, introducing a new macro placer called AutoDMP. Prior works in macro placement have traditionally divided the problem into floorplanning and placement of standard cells and macros separately, which can hinder performance due to limited coordination between these elements. Classical floorplanning methods include simulated annealing techniques and partitioning-based approaches, while more recent methods have applied reinforcement learning to achieve competitive placements. However, these approaches either suffer from scalability issues or require significant computational resources.

AutoDMP advances the field by integrating Bayesian optimization with DREAMPlace, allowing simultaneous placement of macros and standard cells in a mixed-size environment. Unlike sequential flows, which may struggle to optimize placement quality when faced with complex multi-objective goals such as timing, wirelength, and area, the AutoDMP methodology achieves a Pareto-optimal balance across these competing factors. By utilizing a multi-objective Tree-Structured Parzen Estimator (MOTPE), AutoDMP refines parameters to deliver high-quality placements with improved Power-Performance-Area (PPA) metrics on benchmarks. Additionally, DREAMPlace is extended with features like macro halo adjustments and density constraints, which facilitate effective placement legalization while reducing overlaps. This method has demonstrated comparable or superior performance to industry-standard EDA tools, marking a significant step toward scalable and efficient VLSI design automation.

3.3. IncreMacro

IncreMacro, developed by [Pu et al. 2024], builds upon DREAMPlace's analytical placement capabilities by introducing a targeted approach to incrementally refine macro placement. Recognizing that placing macros centrally can lead to routing congestion and timing issues, IncreMacro employs a three-stage methodology: it diagnoses poorly placed

macros using a KD-tree, shifts them towards the periphery through gradient-based optimization, and finalizes positions using constraint-graph-based linear programming. This structured approach preserves the relative positioning of macros, thus maintaining wirelength optimization from the original placement, while eliminating central blockages that hinder routing. In evaluations on RISC-V benchmarks, IncreMacro shows significant improvements in PPA metrics compared to DREAMPlace alone, with reductions in wirelength and power consumption, making it a valuable refinement tool in GPU-accelerated placement flows.

3.4. Google's Reinforcement Learning

The work by [Mirhoseini et al. 2021] propose a novel approach to chip floorplanning using deep Reinforcement Learning (RL). This method reformulates the floorplanning problem as a Markov decision process, leveraging an edge-based graph neural network for learning effective representations of chip netlists. Their approach optimizes key metrics such as PPA while considering constraints like routing congestion and density. Notably, the method produces manufacturable floorplans within six hours, claiming to surpass or equal human expert designs across metrics. By enabling generalization and transfer learning, the proposed RL framework not only accelerates the chip design process but also has the potential to transform other placement optimization tasks beyond chip design. The methodology was applied in the design of Google's TPU accelerators.

This work was heavily criticized for its lack of reproducibility, flawed methodologies, and exaggerated claims. A recent work by [Markov 2024] reviews and meta-analyses the work by [Mirhoseini et al. 2021], highlighting several critical shortcomings: the omission of necessary details for reproducing experiments, reliance on proprietary data, and poorly documented baselines. Independent evaluations demonstrated that standard techniques like simulated annealing and commercial EDA tools outperformed the proposed RL approach. The RL methodology was further questioned for its use of a flawed proxy cost function, which correlated poorly with actual chip metrics, and for overstating results without proper statistical analysis. Allegations of research misconduct, including cherry-picking favorable results, were also reported, raising doubts about the study's integrity. Despite the claimed innovation, subsequent investigations revealed that the methodology failed to surpass state-of-the-art approaches in both performance and efficiency [Markov 2024].

3.5. HiDaP

In the work by [Vidal-Obiols et al. 2021], the authors present a novel approach to macro placement in VLSI design, emphasizing the use of RTL dataflow information. This method, named HiDaP, stands out from traditional approaches by integrating dataflow-driven techniques with a hierarchical, multilevel optimization strategy. By leveraging RTL structural details, such as hierarchy and pipeline register stages, HiDaP calculates dataflow affinity between blocks based on factors like bitwidth and latency, which aids in maintaining timing and minimizing wirelength. Unlike typical analytic and partitioning-based placement methods, this approach emphasizes the interactions between macros and their surrounding standard cells, optimizing not only for placement quality but also for ease in meeting timing constraints with minimal manual adjustments. HiDaP uses a recursive, top-down placement model, which provides a robust structure for managing large

designs with diverse macro requirements and varying block sizes. It is shown to achieve results that closely rival or even exceed those of handcrafted placements in terms of timing and wirelength performance, positioning HiDaP as an efficient alternative in industrial design flows.

3.6. Conclusion

The reviewed methodologies address critical aspects of macro placement, the most computationally intensive part of floorplanning, with unique strengths and limitations, but do not approach floorplanning as a whole. DREAMPlace establishes a foundation with GPU-accelerated global placement, achieving substantial speedups without quality degradation [Lin et al. 2021]. However, its focus on general analytical placement leaves room for specialized improvements in macro handling and specific design constraints.

AutoDMP builds on DREAMPlace, introducing machine learning-driven multi-objective optimization to tackle mixed-size placement challenges [Agnesina et al. 2023]. Its integration of Bayesian optimization delivers Pareto-optimal placements across PPA metrics, though its reliance on extensive computational resources may limit accessibility for iterative design flows.

IncreMacro refines macro placement incrementally, preserving relative positional relationships and reducing central blockages that degrade routing [Pu et al. 2024]. By employing KD-tree-based diagnosis and gradient-based adjustments, it enhances routability and timing while minimizing disruption to established layouts. However, its benefits are constrained by reliance on initial placement quality from analytical tools.

The controversial work responsible for Google's Reinforcement Learning technique proposed a novel use of RL for macro placement, achieving results comparable to human experts on proprietary TPU designs [Mirhoseini et al. 2021]. However, subsequent critiques [Markov 2024] revealed significant flaws in reproducibility, unverified claims of superiority over standard methods, and reliance on a proxy cost function poorly correlated with actual chip metrics.

HiDaP introduces a dataflow-driven, hierarchical approach, leveraging RTL information to optimize timing and wirelength while integrating macros and standard cells [Vidal-Obiols et al. 2021]. Its structured top-down process balances scalability with design-specific adaptability. Despite these advantages, its hierarchical dependency may complicate integration with flat placement frameworks.

Table 1 lists and compares the main properties of the related work. In the first column, the reference of the work is listed. In the following columns, it is indicated what kind of strategy the work implements.

Table 1. Related work comparison.

Work	Macro Placer	Machine Learning	Bayesian Network	Linear Programming	Markov Decision	Dataflow Driven
[Lin et al. 2021]		X				
[Agnesina et al. 2023]	X	X	X			
[Pu et al. 2024]	X	X		X		
[Mirhoseini et al. 2021]	X	X			X	
[Vidal-Obiols et al. 2021]	X					X

In conclusion, while DREAMPlace and AutoDMP excel in computational efficiency and mixed-size optimization, IncreMacro and HiDaP target placement refinement

and structural awareness, respectively. In this work, instead of focusing only on a part of it, the general floorplanning flow will be approached, independent of the macro placer used.

4. Methodology

To elaborate a new VLSI floorplanning flow a profound study of the standard flow and the properties of each tool in the floorplan is required. This section describes the main characteristics of the general structure and the used macro placers available in the OpenROAD toolkit [The OpenROAD Project 2024a].

4.1. OpenROAD Structure

Utilizing Object Oriented Programming (OOP), the OpenROAD toolkit is built in a modular architecture, where every individual tool has its own separate module and namespace, and connected together with a top module. This top module, aptly named *ord*, creates an User Interface (UI) where its possible to individually call all the public commands created by the individual tool modules. For the industrial use of the toolkit, instead of utilizing the central *ord* module, every step of the RTL-GDSII flow is separately executed, initiating a specific instance of the toolkit for every module to be used. The toolkit is mostly written in the C++ language, utilizing a few Python scripts for linking support to the compiler.

The flow is implemented utilizing the GNU Make tool, where every circuit has its own *config.mk* file defining environment variables, source HDL description and technology node used. These files are imported in a central *Makefile*, where several different *Tcl* scripts are executed utilizing the imported information. Every step in the flows described by figures 4 and 3 is a different *Tcl* script. To connect this step by step execution approach, there is an unifying database structure with a proprietary file, the OpenDB module with its respective *.odb* files.

The OpenDB module is a comprehensive design database used to support tools for physical chip design. It is structured to handle various aspects of chip design, including layout, routing, and technology information. The module follows a design pattern that separates the interface from the implementation, resulting in public and private classes for each database object. The odb module uses a system of object tables and pages to manage database objects efficiently. Each object has a unique identifier (OID) that is persistent across save/restores, allowing for consistent referencing. The database supports hierarchical design by allowing blocks to contain other blocks, instances, and modules. This is managed through various hash tables and vectors that store references to these objects. The database has support classes that allow event-driven programming by providing hooks that can be implemented to respond to changes in the database, such as the creation or destruction of instances and nets. The database is designed to be saved and restored with exact fidelity, ensuring that the layout and state of the design are preserved across sessions.

Besides operating with the *.odb* files, OpenDB also manages the input and output for the standard industry file formats Library Exchange Format(LEF) and Design Exchange Format (DEF), used to describe the physical aspects of ICs during various design stages. LEF describes the physical characteristics and constraints of standard cells, macros, and IP blocks in a technology library, while DEF represents the design of an entire chip or a specific block in terms of placement, routing, and connectivity.

4.2. Macro Placers

TritonMP, as implemented in the MacroPlacer class with the *mpl* namespece, is the first implemented macro placer in OpenROAD, being replaced by the newer Hierarchical RTL-MP. It is designed to place macros or blocks in a circuit layout while considering various constraints such as halos, channels, and snapping to cell rows. TritonMP offers two main placement strategies: corner_min_wl and corner_max_wl. These strategies determine whether the tool should minimize or maximize the wire lengths of connections between macros, with the maximization strategy forcing the macros to the corners of the layout. In the standard flow utilizing TritonMP (flow1_TMP), the maximization strategy is the default. The tool uses a ParquetFP [Markov and Adya 2003] based annealing engine to optimize the placement of macros, aiming to minimize or maximize wire lengths based on the chosen strategy. The code in algorithm 1 shows the internal flow of the maximum wirelenth strategy.

The function begins by checking if the *MacroPlacer* is properly initialized in line 1. If not, it returns immediately. Line 4 calculates the initial weighted wire length of the macros and logs this information. A Layout object is created to represent the area where macros will be placed in line 5. A Partition object is initialized with all macros, representing the entire layout area in line 6 to 7. In lines 8 and 9 *MacroPartMap* is created to map macros to their respective partitions, and this map is filled using the *makeMacroPartMap* function. If the placement is connection-driven, the netlist table is filled with connection weights using the *fillNetlistTable* function as seen in line 10 and 11. The layout is divided into partitions using cut lines, and each partition is annealed using the ParquetFP engine to explore different macro placements. The function iterates over all possible partition sets, annealing each one and updating macro locations based on the best solution found, as seen in lines 17 to 33. After annealing, the function evaluates the solutions based on the weighted wire length and selects the best one. If a better solution is found, it updates the macro locations in the database. The function updates the database with the final macro placements. Both strategies follow this same flow, only changing the values in line 14 and 28, either choosing the smallest value possible or like seen in the code choosing the largest value possible.

The *fillNetlistTable* function in the Partition class executed in line 12 is responsible for populating a table (*net_tbl_*) that represents the connectivity and weights between macros and core edges within a partition. This table is used to guide the macro placement process by providing information about the relative importance of different connections. This function utilizes a data structure called Core Edges. Instead of considering the position of the block terminals (BTerm), OpenDB structure for I/O pins, TritonMP only considers in which of the edges of the circuit the I/O is located. The function begins by calculating the total number of macro and core edge connections (*macro_edge_count*) and resizing the *net_tbl_* to accommodate all possible connections. The function then fills the *net_tbl_* with weights for all pairs of macros and core edges.

The Hierarchical RTL-MP is created by the HierRTLMP class. It operates within the *mpl2* namespace and is part of a framework that supports multi-level clustering and timing-driven macro placement. The class is structured to handle various stages of macro placement, from initialization to final placement and orientation improvement, each addressing different aspects of macro placement:

Algorithm 1: MaxWL

```
1  if not initialized then
2      return
3  end
4  initial_wl ← calculateInitialWeightedWireLength()
5  layout ← createLayout(lx, ly, ux, uy)
6  partition ← createPartition(ALL, lx, ly, ux - lx, uy - ly)
7  partition.macros ← macros
8  macroPartMap ← createMacroPartMap()
9  makeMacroPartMap(partition, macroPartMap)
10 if connection_driven then
11     partition.fillNetlistTable(macroPartMap)
12 end
13 partitionSets ← getPartitions(layout, partition)
14 bestWwl ← −∞
15 bestSetIdx ← 0
16 foundBest ← False
17 foreach partitionSet in partitionSets do
18     if partitionSet.size() = 1 then
19         continue
20     end
21     foreach curPart in partitionSet do
22         success ← curPart.anneal()
23         if not success then
24             break
25         end
26     end
27     curWwl ← calculateWeightedWireLength()
28     if curWwl > bestWwl then
29         bestWwl ← curWwl
30         bestSetIdx ← indexOf(partitionSet)
31         foundBest ← True
32     end
33 end
34 if foundBest then
35     updateMacroLocations(bestSet)
36 end
```

- Multilevel Autoclustering: Converts the logical hierarchy into a physical hierarchy, setting the stage for subsequent placement steps.
- Coarse Shaping: Determines rough shapes for clusters, focusing on macro sizes and ignoring standard-cell clusters.
- Fine Shaping: Refines cluster shapes based on parent cluster outlines and locations.
- Hierarchical Macro Placement: Places clusters and macros in a top-down approach, considering both cluster and macro levels.
- Boundary Pushing: Adjusts macro clusters to design boundaries, ensuring no overlap with IO blockages or other macros.
- Orientation Improvement: Optimizes macro orientation to improve wirelength reduction.

This organization can be seen by the code described in algorithm 2.

Algorithm 2: Run_Hier

```
1  runMultilevelAutoclustering()
2  if skip_macro_placement then
3  |   return
4  end
5  if not tree.has_std_cells then
6  |   resetSAParameters()
7  end
8  runCoarseShaping()
9  runHierarchicalMacroPlacement()
10 pusher ← createPusher(tree.root, block, boundary_to_io_blockage)
11 pusher.pushMacrosToCoreBoundaries()
12 updateMacrosOnDb()
13 generateTemporaryStdCellsPlacement(tree.root)
14 correctAllMacrosOrientation()
15 commitMacroPlacementToDb()
16 writeMacroPlacement(macro_placement_file)
17 clear()
```

In line 1 the multilevel autocluster is called. Line 2 verifies if the hand made macro placement flag is active, returning if it is. If there are already cells in the macro placer data structure, it is cleared as seen in lines 5 and 6. In line 8 the Coarse shaping is called. The function *runHierarchicalMacroPlacement* in line 9 is responsible for both the macro placement and the fine shaping. To realize the boundary pushing, it is created a pusher object, that receives the circuit area and the blockages, and a method of this object is called, as seen in line 10 and 11. In line 12, the positioned macros are placed in the odb structure. For the orientation improvement, fake cell placements are made in line 13, and utilized to change the macro orientations in line 14. The oriented macros are updated on the data structure. In line 16 a file writer is called, used to generate and odb file with the macro placement. Lastly, all the constraints and internal structures are cleaned in line 17.

There are two instances of interactions between the circuit I/Os during the execution of the Hierarchical RTL-MP. The first one happens during the *runMultilevelAu-*

toclustering function, where the new hierarchy is created. In this function, the I/O are considered not by their position, but their connectivity with circuit pins and nets, generating the new hierarchical blocks. The second interaction within the *runCoarseShaping* function, where the placed I/Os are used to created area blockages for the macro placer. This blockage is used during the *runHierarchicalMacroPlacement* and the *pushMacrosToCoreBoundaries* functions.

The conclusion from this macro placer analysis is that both TritonMP and Hierarchical RTL-MP do not fully consider the position of the circuit I/Os. TritonMP has a stronger connection with the I/O locations, as it considers in what position of the circuit they are for the weight calculation. This justifies the first step of pin placement in the standard flow (flow1_TMP and flow1_hier) being random, as the following macro placement step will not consider the exact I/O positions. This disconnection from the pin placement and macro placement will be addressed in the following section, with the introduction of new flows.

5. Proposed Flows

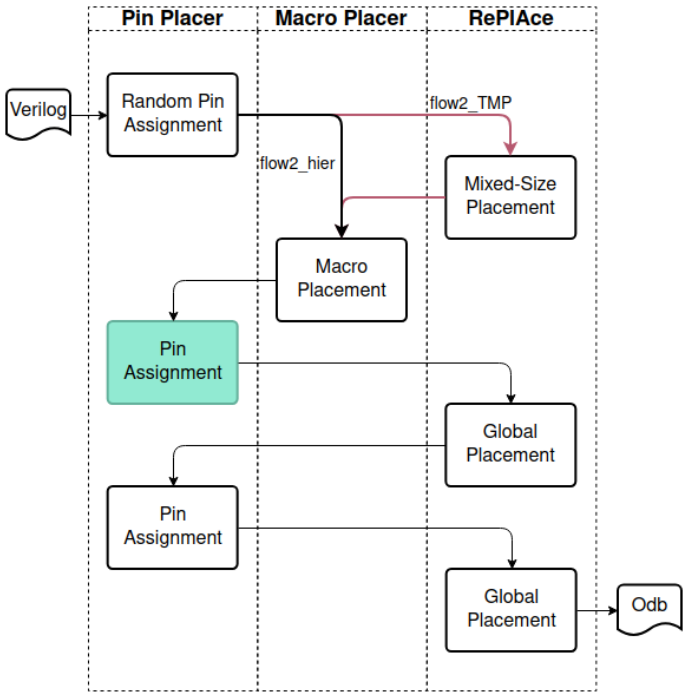
After analyzing OpenROAD's macro placers properties described in the previous section, it was noticed that the macro placement has a certain independence from the pin assignment, more prevalent in the newer Hierarchical RTL-MP due to not only this macro placer consider the I/Os placement just as blockage areas, but also the lack of the initial mixed-size placement before the macro placement step present in the TritonMP flow. The mixed-size placement made by the RePlAce tool considers the randomly assigned I/Os, causing the following macro placement to indirectly consider the I/Os trough the positioned standard cells. Although the older macro TritonMP considers more of the I/O positions with the core edges, there is still a degree of independence from the exact I/O placement. With this in mind, two initial flows were devised, one for each macro placer.

The first flow to be examined is called flow2_TMP. The first three steps of flow2_TMP are the same as flow1_TMP. However, an extra call of the Pin Placer is added just after the macro placement so as to perform a non random pin assignment using hungarian matching. After this new step, flow2_TMP follows the same remaining steps of flow1_TMP, continuing with the global placement using RePlAce, but now considering the I/O pins in its execution, as seen in Fig. 5, where the new step is colored green. Flow2_TMP is represented by the red flow line in the start, before converging with flow2_hier.

The proposed flow, flow2_hier, follows the same changes as the ones made on flow2_TMP, i.e., an extra non random pin assignment step is added after the macro placement, and hence the following global placement step is altered so as to consider the I/O pins positions during its execution, as it can be seen in Fig. 5, with the new step is indicated by the green color and flow2_hier being represented by the black flow line before converging with flow2_TMP.

To explore the optimal timing of pin assignment within the floorplanning and placement sequence a third set of flows was created. By adding a pin assignment step after detailed placement made with the RePlAce tool, this third flow tests whether aligning I/O pin locations closer to the final stages of placement can improve routing efficiency and performance. These flows aim to determine if late-stage pin assignment offers better

Figure 5. Proposed Floorplanning Flows flow2_TMP and flow2_hier.



adaptability to the final layout conditions, or if early-stage pin assignment provides more foundational guidance for placement and routing optimization.

The first of these new flows is flow3_TMP and the second is flow3_hier, as seen in Fig. 6. The flowchart is expanded to include the detailed placement, the step following the global placement shown in all previous flowcharts, as well as the new late-stage pin assignment step, colored in blue.

As all of the previously proposed flows add steps in different parts of the floorplan, they do not conflict with each other, allowing for the creation of a final set of flows, flow4_TMP and flow4_hier, combining together the changes of flow2_TMP and flow2_hier with flow3_TMP and flow3_hier. With these new flows lies in the potential for progressively refined pin alignment throughout the placement stages. By incorporating an initial pin assignment after macro placement, the flow provides an early anchor allowing global placement to adjust component positioning relative to these key access points.

Following this with a second pin assignment after detailed placement allows for further optimization, ensuring pin locations align with the nearly finalized layout, thereby reducing routing complexity and enhancing overall design efficiency. This dual-assignment approach aims to capitalize on the benefits of early guidance from pin locations while still providing flexibility to fine-tune these assignments as the layout crystallizes, which could result in a more optimized, adaptable floorplan. These new flows can be seen with Fig. 7, where the new steps are painted green and blue.

The summary of the flows can be seen in Table 2, where the first column indicates the flow, and the subcolums of the second column indicate where is the extra Pin Assignment in the determined flow.

Figure 6. Proposed Floorplanning Flows flow3_TMP and flow3_hier.

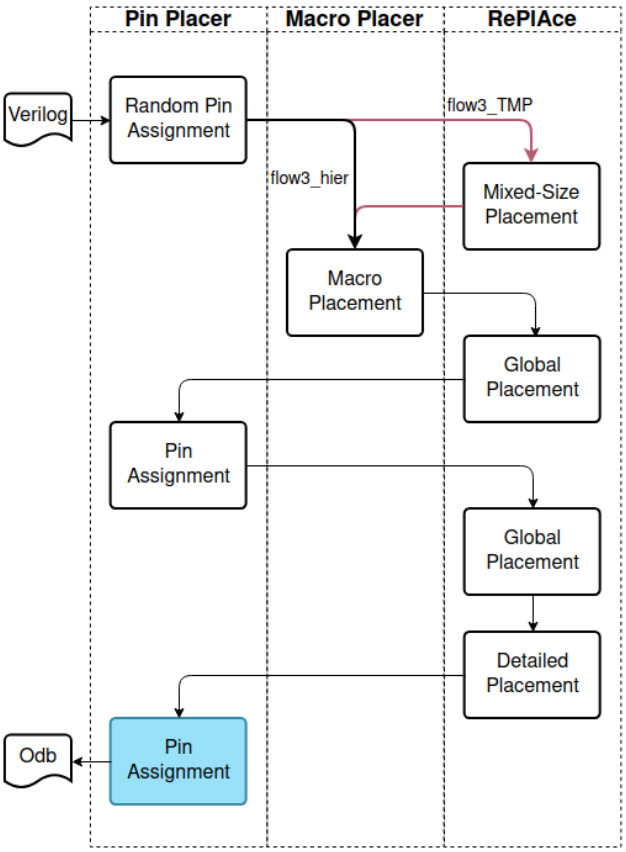


Table 2. Summary of flow characteristics.

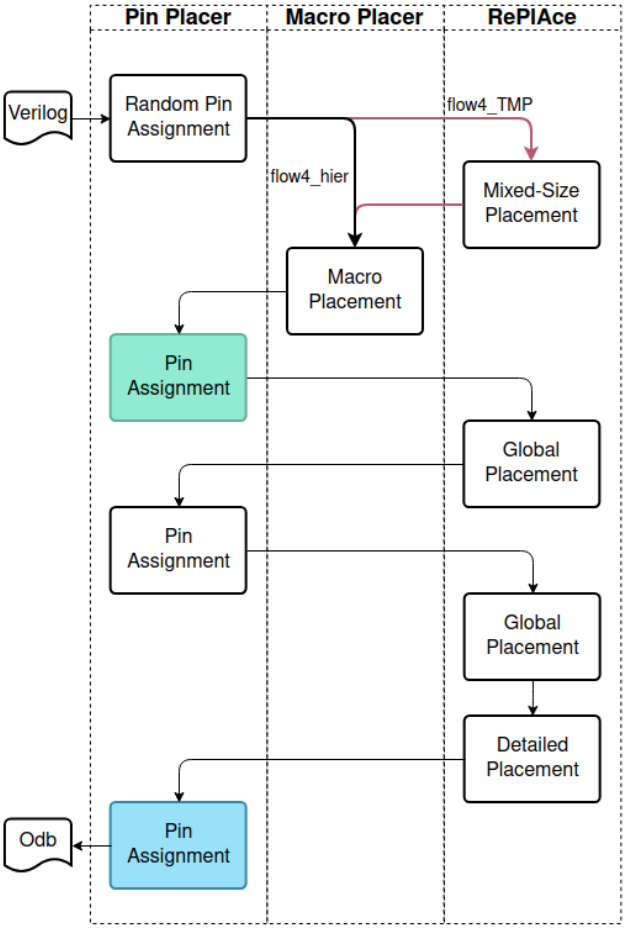
Extra Pin Placement	Flow1	Flow2	Flow3	Flow4
After Macro Placement		X		X
After Detailed Placement			X	X

5.1. Experimental Results

The experimental evaluation of the proposed flows used eight different designs available from OpenROAD listed in Table 3. In this table, column 1 gives the names of the circuits, whereas columns 2, 3 and 4 bring the number of cells, the number of macros and the number of I/Os, respectively. The circuits use the FreeDPK45-based open-source NanGate45 enablement. Each circuit was synthesized for each of the eight flows (flow1_TMP, flow2_TMP, flow3_TMP, flow4_TMP, flow1_hier, flow2_hier, flow3_hier, flow4_hier) and successively underwent the remaining steps i.e., detailed placement, global routing and detailed routing. Thus, a total of 64 syntheses were carried out; All executions used an Ubuntu 22.04.4LTS workstation with an Hexacore Intel® Core® i7 8750H CPU and 16GB RAM at 3200MHz. The OpenROAD Flow Scripts version used in the experiments was the commit labeled 9f67f4a [The OpenROAD Project 2024b], and the OpenROAD toolkit version was the commit labeled 57be191. Total Wirelength, Number of Vias and Synthesis Runtime information was gathered from the OpenROAD reports.

The difference between the results obtained by comparing both macro placers in flow1_TMP and flow1_hier was not investigated in this work, for it is known that the new

Figure 7. Proposed Floorplanning Flows flow4_TMP and flow4_hier.



Hierarchical RTL-MP macro placer is still under development and its focused on state-of-the-art circuits with hundreds of macros and a starting hierarchy defined to be further altered and improved, something not present in the test circuits used in this work.

Table 4 shows the obtained results for the TritonMP flows circuit analysis. Column 1 gives the name of the circuits, Columns 2 to 5 bring the total wirelength results, in the internal OpenROAD generic unit of measurement, Columns 6 to 9 show the results of numbers of vias, and Columns 10 to 13 display the time, in seconds, taken to run the complete physical synthesis up to finishing the detailed routing. All results are given as percentages of those of flow1. Negative percentages correspond to improvements, whereas positive percentages indicate worsens.

Regarding total wirelength, flow2_TMP and flow4_TMP achieve reductions of 1.25% and 1.21%, respectively, while flow3_TMP remains nearly neutral with a minor 0.02% increase. These reductions highlight the benefits of the individual and combined techniques in optimizing interconnect length. For via count, both flow2_TMP and flow4_TMP also show modest improvements, with reductions of 0.37% and 0.26%, respectively, whereas flow3_TMP has a minor impact, only reducing by -0.02%. This suggests that flow2_TMP's approach is slightly more effective in improving layout metrics.

The synthesis runtime analysis offers a contrasting perspective. Flow2_TMP in-

Table 3. Main Statistics of the Test Circuits.

Circuit	# cells	# macros	# I/Os
ariane136	175K	136	495
ariane133	167K	133	495
swerv_wrapper	96K	28	1416
black_parrot	302K	24	1198
bp_multi	137K	26	1453
bp_be	50K	11	3029
bp_fe	33K	10	2511
tinyRocket	25K	2	269

Table 4. Circuit analysis results for TritonMP flows: flow2, flow3 and flow4 results expressed as flow1 percentages.

Circuit	Total Wirelength				Number of Vias				Synthesis Runtime			
	flow1_TMP	flow2_TMP	flow3_TMP	flow4_TMP	flow1_TMP	flow2_TMP	flow3_TMP	flow4_TMP	flow1_TMP	flow2_TMP	flow3_TMP	flow4_TMP
ariane136	7073k	0.22%	0.00%	0.21%	1337k	-0.09%	0.02%	-0.06%	5530	71.90%	1.73%	10.51%
ariane133	Failed	-	-	-	Failed	-	-	-	Failed	-	-	-
swerv_wrapper	4304k	2.70%	-0.01%	2.71%	931k	0.93%	-0.02%	0.95%	4397	8.99%	0.08%	7.16%
black_parrot	7266k	-2.28%	-0.02%	-2.29%	1304k	0.14%	-0.08%	0.10%	2769	18.71%	0.76%	18.33%
bp_multi	4125k	-1.67%	0.00%	-1.68%	757k	-0.19%	0.04%	-0.13%	1597	16.67%	0.00%	17.24%
bp_be	2673k	-6.17%	0.09%	-6.00%	431k	-1.40%	-0.01%	-1.28%	1152	21.50%	0.14%	15.83%
bp_fe	2010k	-8.42%	0.10%	-8.35%	294k	-3.11%	-0.13%	-2.65%	636	21.18%	-0.87%	21.95%
tinyRocket	646k	6.88%	-0.05%	6.94%	190k	1.16%	0.03%	1.26%	872	-11.06%	1.26%	-14.44%
Average	**-**	**-1.25%**	**0.02%**	**-1.21%**	**-**	**-0.37%**	**-0.02%**	**-0.26%**	**-**	**21.13%**	**0.44%**	**10.94%**
Median	**-**	**-1.67%**	**0.00%**	**-1.68%**	**-**	**-0.09%**	**-0.01%**	**-0.06%**	**-**	**18.72%**	**0.14%**	**15.83%**

curs a significant average increase of 21.13% in runtime, indicating a trade-off for its wirelength and via reductions. Flow4_TMP, which combines the methods of flow2_TMP and flow3_TMP, achieves a more balanced runtime overhead of 10.94%. In contrast, flow3_TMP demonstrates the lowest runtime increase at only 0.44%, showcasing its efficiency but with less pronounced improvements in physical metrics.

Outliers like the *tinyRocket* circuit, where flow2_TMP significantly increases wirelength (6.88%), but drastically reduces the runtime (-11.06%), emphasize the variability in flow performance depending on circuit characteristics. Conversely, circuits like *bp_be* and *bp_fe* highlight substantial wirelength reductions (-6.17% and -8.42%, respectively), validating the effectiveness of flow2_TMP.

Overall, while flow4_TMP offers the best balance between optimization and runtime, these results underscore the importance of evaluating trade-offs for specific circuit designs.

Table 5 shows the obtained results for the TritonMP flows circuit analysis. As in the previous table, Column 1 gives the name of the circuits, Columns 2 to 5 bring the total wirelength results, in generic units, Columns 6 to 9 show the results of numbers of vias, and Columns 10 to 13 display the synthesis time, with all results given as percentages of those of flow1.

The results of the hierarchical RTL-MP flows reveal notable differences compared to the initial flow1_hier, with all flows demonstrating distinct advantages depending on the evaluated metrics. For total wirelength, flow2_hier and flow4_hier achieve average reductions of 1.44% and 1.38%, respectively, underscoring their effectiveness in minimizing routing costs, while flow3_hier exhibits a slight increase of 0.30%. These reductions indicate that flow2_hier and flow4_hier, which focus on optimizing pin assignment in con-

Table 5. Circuit analysis results results for Hierarchical RTL-MP flows: flow2, flow3 and flow4 results expressed as flow1 percentages.

Circuit	Total Wirelength				Number of Vias				Synthesis Runtime			
	flow1_hier	flow2_hier	flow3_hier	flow4_hier	flow1_hier	flow2_hier	flow3_hier	flow4_hier	flow1_hier	flow2_hier	flow3_hier	flow4_hier
ariane136	8232k	-1.21%	-0.01%	-1.19%	1549k	-0.10%	0.05%	-0.07%	6714	8.83%	1.77%	9.80%
ariane133	7725k	-1.03%	-0.01%	-1.03%	1473k	-0.43%	0.02%	-0.44%	7401	-7.64%	3.39%	-7.00%
swerv_wrapper	4353k	-1.24%	-0.21%	-1.30%	996k	-0.09%	-0.20%	-0.14%	5372	9.27%	-3.58%	16.50%
black_parrot	9010	2.02%	0.01%	1.98%	2185k	0.08%	0.04%	0.04%	4484	23.37%	0.48%	23.52%
bp_multi	4820k	-0.23%	0.02%	-0.20%	1094k	-0.08%	0.01%	-0.10%	2329	21.52%	1.49%	21.84%
bp_be	3085k	-0.60%	-1.72%	-0.15%	548k	-3.02%	-5.77%	-2.95%	2000	16.24%	-11.57%	26.14%
bp_fe	2332k	-9.34%	-0.37%	-9.23%	353k	-7.66%	-0.34%	-7.69%	1170	-23.05%	-0.49%	-25.38%
tinyRocket	691k	0.11%	-0.07%	0.10%	193k	0.07%	0.07%	-0.05%	746	9.96%	3.69%	10.60%
Average	**-**	**-1.44%**	**-0.30%**	**-1.38%**	**-**	**-1.40%**	**-0.77%**	**-1.43%**	**-**	**7.31%**	**-0.60%**	**9.50%**
Median	**-**	**-0.81%**	**-0.05%**	**-0.62%**	**-**	**-0.09%**	**0.01%**	**-0.12%**	**-**	**9.62%**	**0.99%**	**13.55%**

junction with hierarchical placement strategies, are well-suited for improving interconnect efficiency.

In terms of via count, flow4_hier achieves the largest average reduction of 1.43%, followed closely by flow2_hier at 1.40%, while flow3_hier again has a smaller impact with a reduction of only 0.77%. This consistency across wirelength and via count suggests that flow4_hier effectively combines the strengths of the other two flows while maintaining minimal design complexity, particularly benefiting circuits such as *bp_fe*, which shows a remarkable via count reduction of 7.69%.

The synthesis runtime metric presents a different story, with flow2_hier incurring an average increase of 7.31%, flow4_hier at 9.50%, and flow3_hier demonstrating a rare reduction of 0.60%. Notably, the outlier *bp_fe* achieves significant runtime improvements across all hierarchical flows, with reductions as high as 25.38% for flow4_hier, emphasizing the role of specific circuit characteristics in runtime performance. In contrast, circuits like *black_parrot* experience runtime increases of over 23%, which could be attributed to the complexity of balancing macro placement and pin optimization.

Overall, the hierarchical flows validate the benefits of combining macro placement strategies with advanced pin assignment techniques. Flow4_hier consistently emerges as the most balanced option, achieving substantial physical optimizations at a moderate runtime cost.

Table 6 shows the obtained results for the TritonMP flows timing analysis. Column 1 gives the name of the circuits, Columns 2 to 5 bring the worst slack, Columns 6 to 9 show the results of Total Negative Slack (TNS), and Columns 10 to 13 displays the number of Design Rule Violation (DRV) errors during the synthesis process. All results are given as percentages of those of flow1. Negative percentages correspond to improvements, whereas positive percentages indicate worsens. The circuit *ariane_136* is not shown in this table, as it had no negative slack and no *DRV*.

Table 6. Timing analysis results for TritonMP flows: flow2, flow3 and flow4 results expressed as flow1 percentages.

Circuit	Worst Slack				Total Negative Slack				DRV			
	flow1_TMP	flow2_TMP	flow3_TMP	flow4_TMP	flow1_TMP	flow2_TMP	flow3_TMP	flow4_TMP	flow1_TMP	flow2_TMP	flow3_TMP	flow4_TMP
ariane133	Failed	-	-	-	Failed	-	-	-	Failed	-	-	-
swerv_wrapper	-0.287	21.53%	-3.00%	28.77%	-91.497	116.53%	5.76%	115.76%	955	9.95%	-3.56%	10.37%
black_parrot	-2.988	-10.81%	0.11%	-10.77%	-2.988	-10.81%	0.11%	-10.77%	1	0%	0%	0%
bp_multi	-3.796	-4.95%	-0.08%	-5.20%	-3.796	-4.95%	-0.08%	-5.20%	1	0%	0%	0%
bp_be	-0.078	112.89%	43.03%	181.85%	-6.521	108.58%	48.38%	191.72%	102	5.88%	4.90%	6.86%
bp_fe	-0.014	-74.51%	108.39%	171.63%	-0.027	-86.57%	112.95%	468.70%	2	-50.00%	50.00%	300.00%
tinyRocket	-0.354	-15.30%	-0.08%	-14.08%	-156.529	-6.84%	-1.29%	-5.82%	616	-4.71%	0.16%	-5.03%
Average	**-**	**4.81%**	**24.74%**	**58.70%**	**-**	**19.32%**	**27.64%**	**125.73%**	**-**	**-6.48%**	**8.58%**	**52.03%**
Median	**-**	**-7.88%**	**0.02%**	**11.79%**	**-**	**-5.89%**	**2.94%**	**55.28%**	**-**	**0.00%**	**0.08%**	**3.43%**

For the worst slack metric, flow2_TMP demonstrates a modest average worsening of 4.81%, whereas flow3_TMP and flow4_TMP show substantial degradations of 24.74% and 58.70%, respectively. The significant worst slack deterioration in flow4_TMP suggests that its combined optimizations for wirelength and via count may come at the cost of reduced timing margins. The main cause for this elevated value is the circuit *bp_be*, where the worse slack increased by 112.89%, 43.08% and 181.85% with flow2_TMP, flow3_TMP and flow4_TMP respectively, but considering the initial worst slack in flow1_TMP for this circuit is low, the increase in time is not as expressive as the 4.95% reduction in *bp_multi* and 10.81% reduction in *black_parrot*, both with flow2_TMP.

TNS results reveal a more negative. Flow2_TMP achieves a 19.32% average worsening, and flow3_TMP a slightly higher average worsening of 27.64%. However, flow4_TMP exhibits a 125.73% increase, indicating timing degradation. Considering the original values for TNS, *swerv_wrapper* shows the worst performance, increasing in all flows, while *tinyRocket* shows a considerable reduction of 6.84% from the largest TNS. The DRV analysis highlights that flow2_TMP and flow3_TMP offer reductions of 6.48% and 8.58%, respectively, whereas flow4_TMP has a significant DRV increase of 52.03%.

Outlier circuits, such as bp_fe, reveal interesting behaviors. While flow2_TMP achieves significant improvements in worst slack and TNS (-74.51% and -86.57%, respectively), flow4_TMP suffers from major degradations (171.63% and 468.70%, respectively).

Table 7 shows the obtained results for the TritonMP flows timing analysis. As in the previous table, Column 1 gives the name of the circuits, Columns 2 to 5 bring the worst slack, Columns 6 to 9 show the results of TNS, and Columns 10 to 13 displays the number of DRV errors during the synthesis process, with all results given as percentages of those of flow1. Like in the previous table, circuit *ariane_136* is not shown in the table due to having no negative slack nor *DRV*.

Table 7. Timing analysis results for Hierarchical RTL-MP flows: flow2, flow3 and flow4 results expressed as flow1 percentages.

Circuit	Worst Slack				Total Negative Slack				DRV			
	flow1_hier	flow2_hier	flow3_hier	flow4_hier	flow1_hier	flow2_hier	flow3_hier	flow4_hier	flow1_hier	flow2_hier	flow3_hier	flow4_hier
ariane133	-0.093	-23.47%	14.06%	14.22%	-38.343	-48.99%	33.32%	41.01%	952	-23.53%	17.54%	26.68%
swerv_wrapper	-0.648	14.05%	3.98%	20.38%	-601.177	3.97%	-4.32%	22.36%	1423	15.25%	-2.74%	-0.63%
black_parrot	-0.647	-16.95%	0.51%	-16.86%	-0.647	-16.95%	0.51%	-16.86%	1	0%	0%	0%
bp_multi	-1.928	2.94%	0.22%	1.37%	-1.928	2.94%	0.22%	1.37%	1	0%	0%	0%
bp_be	-0.871	-61.06%	-53.25%	-52.31%	-89.903	-61.12%	-56.93%	53.25%	144	-11.81%	-22.92%	-18.06%
bp_fe	-0.314	-92.47%	-14.34%	-89.37%	-12.498	-99.07%	-6.26%	-99.12%	177	-94.29%	0.56%	-95.48%
tinyRocket	-0.273	-8.20%	-4.16%	-6.44%	-112.223	-3.62%	-2.16%	-2.11%	658	-6.08%	-0.15%	-5.93%
Average	**-**	**-26.45%**	**-7.57%**	**-18.43%**	**-**	**-31.83%**	**-5.09%**	**-15.23%**	**-**	**-17.30%**	**-1.10%**	**-13.34%**
Median	**-**	**-16.95%**	**0.22%**	**-6.44%**	**-**	**-16.95%**	**-2.16%**	**-2.11%**	**-**	**-6.08%**	**0.00%**	**-0.63%**

For worst slack, flow2_hier achieves the most notable improvement, with an average reduction of 26.45%, followed by flow4_hier at 18.43%. Flow3_hier shows the smallest average reduction of 7.57%. While most of the circuits show improvements, the circuit with the largest worst slack with Hierarchical RTL-MP, *bp_multi*, shows a 2.94% increase of its worst slack with flow2_TMP, having the worst performance in this circuit compared to the other flows.

TNS results are more significative with Hierarchical RTL-MP. Flow2_hier achieves the most significant improvement, with a reduction of 31.83%, followed by flow4_hier (-15.23%). Flow3_hier offers a smaller reduction of 5.09%, which still in-

dicates an improvement over the baseline flow1_hier. For *swerv_wrapper*, the circuit with the largest TNS, flow3_hier is the only to offer a reduction, reducing by 4.32%, while flow2_hier increases the value by 3.97% and flow4_hier increases by 22.36%.

The DRV analysis reveals consistent reductions across all hierarchical flows, with flow2_hier achieving the largest reduction (-17.30%), followed by flow4_hier (-13.34%). Flow3_hier shows the smallest improvement (-1.10%), but its the only flow to reduce the number of DRV in the circuit with the most of them, *swerv_wrapper*. Overall, flow2_hier demonstrates to be better performing in most circuits, followed by flow4_hier, but few specific circuits benefit more in the timing aspect utilizing flow3_hier, like *bp_multi* and *swerv_wrapper*.

6. Conclusion

In this work, it was investigated alternative floorplanning flows for VLSI physical design using the OpenROAD toolkit, focusing on optimizing macro placement and improving pin assignment through an additional non-random placement step. By comparing the proposed flows (flow2_TMP, flow3_TMP, and flow4_TMP) and their hierarchical versions (flow2_hier, flow3_hier, and flow4_hier) to their respective original versions (flow1_TMP and flow1_hier) across various test circuits, we demonstrated the potential benefits of these enhanced methods. In the TritonMP-based flows, flow2_TMP and flow4_TMP reduced total wirelength by 1.25% and 1.21%, respectively, while maintaining small reductions in via counts. However, flow3_TMP provided minimal improvements, emphasizing the importance of flow design in achieving consistent gains. Similarly, in the hierarchical flows, flow2_hier and flow4_hier achieved average wirelength reductions of 1.44% and 1.38%, respectively, and also exhibited via reductions, particularly benefiting circuits like *bp_fe*, which saw a 7.69% decrease in via count.

The hierarchical flows demonstrated slightly lower runtime overheads compared to their TritonMP counterparts, with flow2_hier incurring an average runtime increase of 7.31%, whereas flow2_TMP increased runtime by 21.13%. Notably, flow3_hier showed minimal runtime increase of 0.60%, albeit with less significant physical optimizations. Outliers, such as the tinyRocket and *bp_fe* circuits, highlighted the variability in flow performance depending on circuit characteristics, with runtime reductions of up to 25.38% observed for flow4_hier on *bp_fe*.

These results underscore the trade-off between improved layout quality and increased computational effort. These findings validate the effectiveness of incorporating non-random pin assignment and hierarchical placement strategies in enhancing the floorplanning process. They also highlight the adaptability of open-source EDA tools like OpenROAD for advancing VLSI design methodologies while addressing specific design challenges.

6.1. Future Work

Future research could focus on developing an advanced macro placer that integrates pin assignment directly into its optimization steps, enabling simultaneous optimization of macro placement and interconnect routing. By embedding pin assignment within the macro placement process, such a tool could better address the interdependencies between these tasks, potentially leading to more significant reductions in wirelength and via count

without the need for a separate pin assignment step. This integrated approach could also streamline the overall design flow, achieving higher-quality layouts.

References

Agnesina, A., Rajvanshi, P., Yang, T., Pradipta, G., Jiao, A., Keller, B., Khailany, B., and Ren, H. (2023). Autodmp: Automated dreamplace-based macro placement. In *Proceedings of the 2023 International Symposium on Physical Design*, ISPD '23, page 149–157, New York, NY, USA. Association for Computing Machinery.

Bandeira, V., Fogaça, M., Monteiro, E. M., Oliveira, I., Woo, M., and Reis, R. (2020). Fast and scalable i/o pin assignment with divide-and-conquer and hungarian matching. In *2020 18th IEEE International New Circuits and Systems Conference (NEWCAS)*, pages 74–77.

Cheng, C.-K., Kahng, A. B., Kang, I., and Wang, L. (2019). Replace: Advancing solution quality and routability validation in global placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 38(9):1717–1730.

Fontana, T. A., Aghaeekiasaraee, E., Netto, R., Almeida, S. F., Gandh, U., Tabrizi, A. F., Westwick, D., Behjat, L., and Güntzel, J. L. (2021). Ilp-based global routing optimization with cell movements. In *2021 IEEE Computer Society Annual Symposium on VLSI (ISVLSI)*, pages 25–30.

Hu, K.-S., Yang, M.-J., Yu, T.-C., and Chen, G.-C. (2020). Iccad-2020 cad contest in routing with cell movement. In *2020 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–4.

Hu, K.-S., Yu, T.-C., Yang, M.-J., and Shen, C.-F. C. (2021). 2021 iccad cad contest problem b: Routing with cell movement advanced: Invited paper. In *2021 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, pages 1–5.

Kahng, A., Lienig, J., Markov, I., and Hu, J. (2011). *"VLSI Physical Design: From Graph Partitioning to Timing Closure"*. Springer Netherlands.

Kahng, A. B. (2022). Leveling up: A trajectory of openroad, tilos and beyond. In *Proceedings of the 2022 International Symposium on Physical Design*, ISPD '22, page 73–79, New York, NY, USA. Association for Computing Machinery.

Kahng, A. B., Varadarajan, R., and Wang, Z. (2023). Hier-rtlmp: A hierarchical automatic macro placer for large-scale complex ip blocks. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*.

Kuhn, H. W. (1955). The hungarian method for the assignment problem. *Naval Research Logistics Quarterly*, 2(1-2):83–97.

Laung-Terng Wang, Y.-W. C. and Cheng, K.-T. T. (2008). *Electronic Design Automation: Synthesis, Verification, and Test*. Morgan Kaufmann.

Lin, Y., Jiang, Z., Gu, J., Li, W., Dhar, S., Ren, H., Khailany, B., and Pan, D. Z. (2021). Dreamplace: Deep learning toolkit-enabled gpu acceleration for modern vlsi placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(4):748–761.

Lu, J., Chen, P., Chang, C.-C., Sha, L., Huang, D. J.-H., Teng, C.-C., and Cheng, C.-K. (2014). eplace: Electrostatics based placement using nesterov's method. In *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6.

Markov, I. L. (2024). Reevaluating google's reinforcement learning for ic macro placement. *Commun. ACM*, 67(11):60–71.

Markov, I. L. and Adya, S. N. (2003). Fixed-outline floorplanning: Enabling hierarchical design. *IEEE Trans. on VLSI Systems*.

Mirhoseini, A., Goldie, A., Yazgan, M., Jiang, J. W., Songhori, E., Wang, S., Lee, Y.-J., Johnson, E., Pathak, O., Nova, A., Pak, J., Tong, A., Srinivasa, K., Hang, W., Tuncer, E., Le, Q. V., Laudon, J., Ho, R., Carpenter, R., and Dean, J. (2021). A graph placement methodology for fast chipdesign. *Nature*, 594(7862):207–212.

Netto, R., Fontana, T. A., Fabre, S., Ferrari, B., Livramento, V., Barbato, T., Souto, J., Guth, C., Pilla, L., and Güntzel, J. L. (2018). Ophidian: an open-source library for physical design research and teaching. In *First Workshop on Open-Source EDA Technology*.

Pu, Y., Chen, T., He, Z., Bai, C., Zheng, H., Lin, Y., and Yu, B. (2024). Incremacro: Incremental macro placement refinement. In *Proceedings of the 2024 International Symposium on Physical Design*, ISPD '24, page 169–176, New York, NY, USA. Association for Computing Machinery.

The OpenROAD Project (2024a). Openroad. `https://github.com/The-OpenROAD-Project/OpenROAD`.

The OpenROAD Project (2024b). Openroad flow scripts. `https://github.com/The-OpenROAD-Project/OpenROAD-flow-scripts`.

Vidal-Obiols, A., Cortadella, J., Petit, J., Galceran-Oms, M., and Martorell, F. (2021). Multilevel dataflow-driven macro placement guided by rtl structure and analytical methods. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 40(12):2542–2555.

Weste, N. H. and Harris, D. M. (2011). *CMOS VLSI Design a Circuit and Systems Perspective*. Addison-Wesley.

Yosys HQ (2024). Yosys Open SYnthesis Suite. `https://github.com/YosysHQ/yosys`.

Annex

ANNEX A – DECLARAÇÃO PADRÃO PARA EMPRESA OU LABORATÓRIO

DECLARAÇÃO DE CONCORDÂNCIA COM AS CONDIÇÕES PARA O DESENVOLVIMENTO DO TCC NA INSTITUIÇÃO

Declaro estar ciente das premissas para a realização de Trabalhos de Conclusão de Curso (TCC) de Ciência da Computação e Sistema de Informações da UFSC, particularmente da necessidade de que se o TCC envolver o desenvolvimento de um software ou produto específico (ex: um protocolo, um método computacional, etc.) o código fonte e/ou documentação completa correspondente deverá ser entregue integralmente, como parte integrante do relatório final do TCC.

Ciente dessa condição básica, declaro estar de acordo com a realização do TCC identificado pelos dados apresentados a seguir.

Instituição	ECL/INE/CTC
Nome do Responsável	José Luís A. Güntzel
Cargo/Função	Prof. INE/CTC
Fone de Contato	(48) 37216466 / (53) 999711982
Acadêmico	Rafael Moresco Vieira
Título do trabalho	Evaluating the Impact of Pin Assignment Order in VLSI Circuit Floorplanning Outcomes
Curso	Ciências da Computação/INE/UFSC

Florianópolis, December 18, 2024.

Professor Responsável
Prof. Dr. José Luís Almada Güntzel