UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Departament de Ciències de la Computació

Thesis presented in partial fulfillment
of the requirements for the Degree of
Ph.D. in Computing

Algorithmic Techniques for Physical Design: Macro Placement and Under-the-Cell Routing

Alex Vidal-Obiols

Advisors: Jordi Cortadella Fortuny, Jordi Petit Silvestre
Computer Science Department
Universitat Politècnica de Catalunya

Barcelona, November 2019

Abstract

With the increase of chip component density and new manufacturability constraints imposed by modern technology nodes, the role of algorithms for electronic design automation is key to the successful implementation of integrated circuits. Two of the critical steps in the physical design flows are macro placement and ensuring all design rules are honored after timing closure. This thesis proposes contributions to help in these stages, easing time-consuming manual steps and helping physical design engineers to obtain better layouts in reduced turnaround time.

The first contribution is under-the-cell routing, a proposal to systematically connect standard cell components via lateral pins in the lower metal layers. The aim is to reduce congestion in the upper metal layers caused by extra metal and vias, decreasing the number of design rule violations. To allow cells to connect by abutment, a standard cell library is enriched with instances containing lateral pins in a pre-selected sharing track. Algorithms are proposed to maximize the numbers of connections via lateral connection by mapping placed cell instances to layouts with lateral pins, and proposing local placement modifications to increase the opportunities for such connections. Experimental results show a significant decrease in the number of pins, vias, and in number of design rule violations, with negligible impact on wirelength and timing.

The second contribution, done in collaboration with eSilicon (a leading ASIC design company), is the creation of HiDaP, a macro placement tool for modern industrial designs. The proposed approach follows a multilevel scheme to floorplan hierarchical blocks, composed of macros and standard cells. By exploiting RTL information available in the netlist, the dataflow affinity between these blocks is modeled and minimized to find a macro placement with good wirelength and timing properties. The approach is further extended to allow additional engineer input, such as preferred macro locations, and also spectral and force methods to guide the floorplanning search.

Experimental results show that the layouts generated by HiDaP outperform those obtained by a state-of-the-art EDA physical design software, with similar wirelength and better timing when compared to manually designed tape-out ready macro placements. Layouts obtained by HiDaP have successfully been brought to near timing closure with one to two rounds of small modifications by physical design engineers. HiDaP has been fully integrated in the design flows of the company and its development remains an ongoing effort.

Resum

A causa de l'increment de la densitat de components en els xip i les noves restriccions de disseny imposades pels últims nodes de fabricació, el rol de l'algorísmia en l'automatització del disseny electrònic ha esdevingut clau per poder implementar circuits integrats. Dos dels passos crucials en el procés de disseny físic és el placement de macros i assegurar la correcció de les regles de disseny un cop les restriccions de timing del circuit són satisfetes. Aquesta tesi proposa contribucions per ajudar en aquests dos reptes, facilitant laboriosos passos manuals en el procés i ajudant als enginyers de disseny físic a obtenir millors resultats en menys temps.

La primera contribució és el routing "under-the-cell", una proposta per connectar standard cells usant pins laterals en les capes de metall inferior de manera sistemàtica. L'objectiu és reduir la congestió en les capes de metall superior causades per l'ús de metall i vies, i així disminuir el nombre de violacions de regles de disseny. Per permetre la connexió lateral de standard cells, n'estenem una llibreria amb dissenys que incorporen connexions laterals. També proposem modificacions locals al placement per permetre explotar aquest tipus de connexions més sovint. Els resultats experimentals mostren una reducció significativa en el nombre de pins, vies i nombre de violacions de regles de disseny, amb un impacte negligible en wirelength i timing.

La segona contribució, desenvolupada conjuntament amb eSilicon (una empresa capdavantera en disseny ASIC), és el desenvolupament de HiDaP, una eina de macro placement per a dissenys industrials actuals. La proposta segueix un procés multinivell per fer el floorplan de blocks jeràrquics, formats per macros i standard cells. Mitjançant la informació RTL disponible en la netlist, l'afinitat de dataflow entre els mòduls es modela i minimitza per trobar macro placements amb bones propietats de wirelength i timing. La proposta també incorpora la possibilitat de rebre suggeriments de les posicions de les macros. Finalment, també usa mètodes espectrals i de forçes per guiar la cerca de floorplans.

Els resultats experimentals mostren que els dissenys generats amb HiDaP són millors que els obtinguts per eines comercials capdavanteres de EDA. Els resultats també mostren que els dissenys presentats poden obtenir un wirelength similar i millor timing que macro placements obtinguts manualment, usats per fabricació. Alguns dissenys obtinguts per HiDaP s'han dut fins a timing-closure en una o dues rondes de modificacions incrementals per part d'enginyers de disseny físic. L'eina s'ha integrat en el procés de disseny de eSilicon i el seu desenvolupament continua més enllà de les aportacions a aquesta tesi.

"It is not down in any map;
true places never are."

Ishmael, in **Moby Dick**
(Herman Melville)

Festina lente.

Hasten slowly.

Preface

The work presented in this thesis proposes two contributions in the field of algorithms for the physical design of integrated circuits. This is a challenging research area where both academia and industry strive to provide tools that enable the design and fabrication of digital circuits in future technology nodes. This preface aims to clarify the circumstances under which the presented research efforts were undertaken.

In chronological order, the first contribution was *under-the-cell routing*, developed from November 2015 to December 2016 at the Computer Science department of the Universitat Politècnica de Catalunya (UPC) under the guidance of my thesis advisors. The SAT-based standard cell router created at the department [21] allowed the development of a proposal for the systematic exploitation of internal cell routing layers by maximizing the opportunities for standard cell connection via abutment. The available resources limited the scope of the validation, for which a more ambitious industrial testing program would be have been desirable. Experiments were run and integrated with a state-of-the-art physical design tool, for which a single license was available at the moment. Although the execution was carried out in an academic environment, it sufficed to show the potential gains of the approach.

The second contribution, on *macro placement*, began on January 2017 and is still an ongoing effort. It is a joint work with my advisors and eSilicon, a leading ASIC company. The synergies among all parties allowed for the development of our macro placement tool HiDaP, combining the algorithmic expertise at the Computer Science department of UPC with the knowledge on physical design workflows of Ferran Martorell, Marc Galceran and many others at eSilicon.

From the very beginning, the tool has been developed and integrated into the eSilicon physical design flow ecosystem. The effort to build such a tool from scratch has involved two and a half years of development and tens of thousands of code lines. The first task was building prototype models for the circuits at different abstraction levels and showing the po-

tential value of the approach, by doing first floorplanning prototypes and developing layout and graph visualization tools aimed at the engineering teams. Having the tool being usable in an industrial context since the beginning helped a lot obtaining feedback, but on the other hand introduced usability and maintenance overheads which are barely reflected in this thesis due to its academic research nature. In the end, as it ensures the utility of HiDaP to tackle the real problems faced by engineers in the most recent FinFET nodes, all this effort has been rewarded.

The collaboration with eSilicon has also allowed us to test the tool with industrial designs under a real design flow. The access to the company's computational resources and physical design tools has enabled us to perform an evaluation which would have simply not been possible otherwise. When working with blocks of millions of cells, placement takes hours, up to a day, and the clock and routing stages can take between one or two days, at least. Massive experiment parallelism, using thousands of CPU hours, allowed us to try our algorithms with several parameters and variations and prove the utility of our approach. Given the very positive obtained results and the utility of the tool, the project is considered a success both from an academic and industrial point of view, and its development is set to continue beyond the scope of this thesis.

The contributions of this thesis have been published in the following conference papers:

- Alex Vidal-Obiols, Jordi Cortadella, Jordi Petit. Under-the-Cell Routing to Improve Manufacturability. In *Proceedings of the ACM Great Lakes Symposium on VLSI 2017* (GLSVLSI), pages 125-130, presented at Lake Louise (Canada), May 2017.

- Alex Vidal-Obiols, Jordi Cortadella, Jordi Petit, Ferran Martorell, Marc Galceran-Oms. RTL-Aware Dataflow-Driven Macro Placement. In *Proceedings of the Design, Automation and Test in Europe 2019* (DATE), pages 186-191, presented at Florence (Italy), March 2019.

An article extending a previous publication has been submitted to a journal and is under the review process:

- Alex Vidal-Obiols, Jordi Cortadella, Jordi Petit, Ferran Martorell, Marc Galceran-Oms. Multilevel Dataflow-Driven Macro Placement guided by RTL Structure and Analytical Methods.

Another contributions was also presented at a workshop:

- Alex Vidal-Obiols, Jordi Cortadella, Jordi Petit, Ferran Martorell, Marc Galceran-Oms. Design Mapper: Dataflow Analysis for Better Floorplans. In *6th Workshop on Design Automation for Understanding Hardware Designs* (DUHDE6), informal proceedings, Florence (Italy), March 2019.

Acknowledgments

The voyage to the limits of what is known can only be successfully undertaken in good company. I would like to express my deepest gratitude to my advisors, Prof. Jordi Cortadella and Prof. Jordi Petit, for their patient guidance and support during all the work leading to the elaboration of this thesis. Such an ambitious project would not have been possible without their constant motivation, expertise and the passion they transmitted to me for intellectual inquiry, problem solving and, of course, graphs. I would like to extend my gratefulness to all professors and mentors who, through my life, have stimulated my curiosity and thirst for knowledge. They are partly to blame for this thesis, too.

I would also like to thank the people at eSilicon for the opportunity to work with them. This project would have not been possible without the guidance and insight of Ferran Martorell and Marc Galceran: working with them has been a privilege. My gratitude also goes to Jordi Perez, Jonàs Casanova, *Nico* Chauveau, Vera Liu and Mónica Casares in the Barcelona office, for all their help, insights and discussions around the bowl of peanuts. It also goes to eSilicon engineers around the world (Anh Dinh Tuan, Hanh Pham Thi My and many others) who provided enormous help during this project. I hope you will find HiDaP of use.

I want to thank all the fellow grad students I had the privilege to meet during these years at the university. First of all, thanks to Alberto Moreno, with whom I have shared most of my adventures in the academic path. I would also like to thank Javier de San Pedro and Lucas Machado, for the many discussions on EDA related topics (and not). The experience during the PhD would not have been the same without the mutual support and debates with the basement-dwellers at the Omega building: Tuomas Hakoniemi, Josep Sanchez, Daniel Alonso, Jorge Muñoz, Josep Lluís Berral, Eva Martinez, Carles Creus, Albert Vilamala and many others. The repeated editions of the Doctoral Coffee Symposium with Alberto *Fubu* Gutierrez and Martí Anglada have also been a source of inspiration during all these years.

Finally I would like to thank my family, specially my parents, for always being there and constantly encouraging me to follow my path. I am also deeply grateful to all the friends who have supported me through all these years. You all gave me the courage, joy and imagination required to finish this project: it would not have been possible without all of you. Special thanks go to Alan Waller and Sergi Ruiz, who have always been there to listen during the bad moments, and celebrate during the good.

This thesis has been partially supported by funds from the Spanish Ministry for Economy and Competitiveness and the European Union (FEDER funds, under grant TIN2013-46181-C2-1-R, TIN2017-86727-C2-1-R and FPI2014-BES-2014-069118), the Generalitat de Catalunya (under grants 2014 SGR 1034, 2017 SGR 786 and FI-DGR 2015) and a grant from eSilicon Corporation.

Contents

List of Acronyms

ASIC application specific integrated circuit

CAD computer-aided design

DAC design automation conference
DATE design, automation and test in europe conference
DRC design rule checking

EDA electronic design automation
EUVL extreme ultra-violet light

FPGA field-programmable gate array

GUI graphical user interface

HDL hardware description language
HiDaP hierarchical dataflow placer
HLS high-level synthesis

IC integrated circuit

PD physical design

QoR quality of results

RTL register-transfer level

SADP self-aligned double patterning
SoC system-on-chip
STA static timing analysis

TNS total negative slack

VLSI very-large-scale integration

WL wirelength
WNS worst negative slack

List of Figures

List of Tables

Chapter 1

Introduction

The complexity of IC design and fabrication has been growing year after year since their introduction. According to Moore's Law [59], the density of transistors on a chip doubles approximately every two years. This tendency has been followed for the last 40 years but, of course, such a fast-paced evolution of the number of transistors comes with a lot of challenges at many levels, such as technology, design and tools. The process that allows combining millions of transistors in a single chip such as a microprocessor is VLSI, a field that has been constantly evolving, trying to build faster chips and integrate more transistors generation after generation. As the number and density per area unit of transistors has dramatically increased over the years, the complexity of circuits has also increased enormously; and with it, the challenges associated to the design of such circuits.

The design of VLSI circuits is therefore a very complex process that requires automation. To the rescue came EDA, a category of software tools for designing electronic systems such as ICs. This aid has been evolving together with the needs of VLSI design since the mid-70s. Nowadays, given the level of complexity that VLSI design has reached, EDA tools play a very important role in the fabrication of ICs, and current industrial chip design workflows would not be possible without the help of CAD tools.

1.1 The IC Design Flow

Current workflows for the design of chips are very modular: an overview of the process can be found in Fig. 1.1. When designing a chip, the first step is to obtain a *design specification* with the details of what needs to be

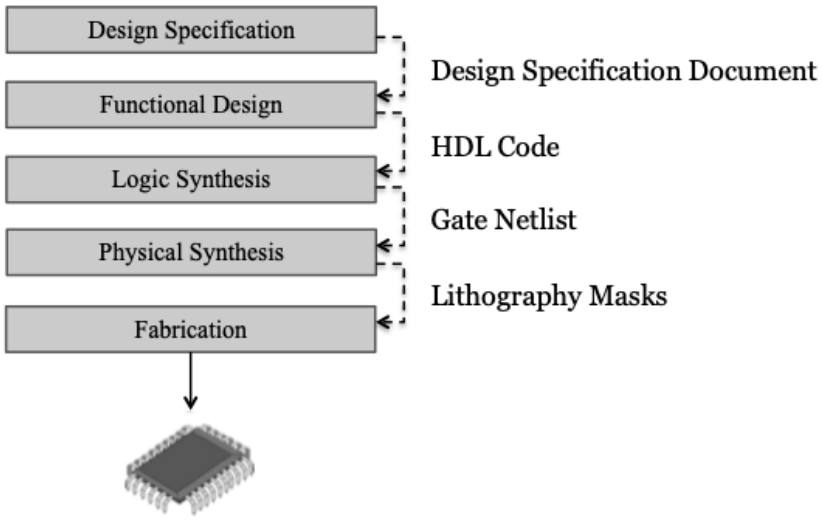


Figure 1.1: VLSI design flow.

built: how the system should behave, under which conditions, etc. This is a high-level description of the system and produces a *design specification document*.

The following step is *functional design*, in which the design described in the specification document is modeled in the RTL design abstraction, which focuses on the flow of digital signals between hardware registers using a HDL such as Verilog or VHDL. This design is done mostly by hand or with the help of HLS, a process that allows to generate RTL from code written using a higher level of abstraction (such as SystemC). The output of this step is HDL code describing the functionality of the circuit.

Next comes *logic synthesis*, which is a fully automated step done by the EDA tools that consists in taking the RTL description of the circuit and converting it into boolean expressions. After several logical optimization steps, the resulting expressions are mapped into physical components of a specific fabrication technology and the *gate netlist*, which describes the connectivity of the electronic circuit components, is produced.

The process continues with *physical synthesis* (or *physical design*), the step which takes the intended circuit design and decides the final physical layout of the circuit, a geometric description of the location of its components and connections that is ready for fabrication. In modern technology nodes, this is a semi-manual process in which the PD engineers use the algorithms provided by EDA vendors to automatically handle the millions of components involved, but still are expected to provide guidance to the flow by partitioning the circuit, adding extra constraints, manually

placing the bigger components and solving DRC violations at the end of the flow. *The contributions of this thesis focus in this particular step.* The physical synthesis flow is presented in detail in Section 2.2. Its outputs are the masks that will be used by photolithography, as shown in Section 2.6.

The last step in the process is *fabrication*, which takes place in the fabrication plants. The fabrication technology imposes a set of design rules that must be respected by the design so that it can be produced. This includes minimum spacing rules, forced component directions and others. As we move into smaller technology nodes, the constraints imposed by design rules become harder to satisfy, adding manufacturability-awareness to the list of desirable properties of new EDA algorithms.

A lot of steps are omitted and simplified in this explanation, including the increasingly expensive verification phases that take place between every pair of steps in the process to ensure no malfunctions are introduced and the behavior of the current circuit abstraction is as intended.

1.2 Motivation

It is important to remark that IC design is a complex engineering problem: given a high-level specification of intended behavior and constraints, multiple final circuits can be considered valid. Many optimization procedures must be applied in order to generate a good circuit design according to the desired goals: low area, low power consumption, failure tolerance... Optimizing a single requirement already leads to several instances of hard optimization problems across the whole design process. Moreover, there is no magical recipe to combine all of these requirements at the same time, and trade-offs between optimization objectives may be needed to achieve the minimum requirements.

In fact, the whole design flow process is not lineal, but rather iterative and repetitive: if it is found that it is unfeasible to implement the desired behavior under the chosen constraints at a given stage, the design must go back to the previous stage and be modified, up to changing the design specification if needed. As these steps are usually performed by different teams or even different companies, it is desirable to keep them at a minimum in order to reduce turnaround time. For all these reasons, the EDA field has become an exciting environment for academia, industry, and even the open source initiatives [3] to propose new, ever evolving methodologies to help engineers achieve the designs of the future.

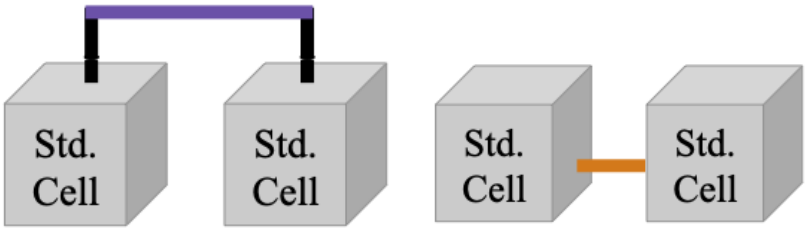
The steps where most optimization effort is devoted are the functional design and physical synthesis steps, which are informally known as the front-end and back-end of the design chain. In order to achieve good QoR, the HDL code of the most critical components is carefully tuned to optimize target objective areas. It is not straightforward to decide how the engineer can interact with the design at the physical design level, as its objective is to find the location for millions of components and connections. Modern tools provided by EDA vendors offer black boxed algorithms that cover basic functionality of physical design and graphical interfaces through which the engineers can see the results and manually interact with the layout. If physical synthesis is unsuccessful, the HDL code must be reworked until an implementable circuit is found.

Although the algorithms provided by EDA vendors may prove sufficient to process basic circuits, the constant push for performance and optimization makes human intervention necessary for the most complex and demanding cases, not only to reduce cycle turnaround time but also to enable designs which could not be produced otherwise. The most time-consuming human interaction to influence the physical design process happens at the beginning and at the end of physical design stage:

1. The first step is to decide the location of relatively big components in the circuit, usually called *macro components* (in contrast of the smaller elements, called *standard cells*). Current automated solutions are fast but results are usually not good enough, demanding manual intervention and understanding of the circuit to ensure the design will meet timing constraints. A lot of effort is involved since bad macro placements usually lead to designs whose timing can not be closed.

2. The last step is to ensure the design complies with all the constraints imposed by the fabrication technology process. These are a set of design rules on the layout objects for, among others, preventing short-circuits or ensuring the design can be manufactured with the target lithography process. EDA algorithms try to minimize them as another of their optimization goals, but they are not optional: the design must move into fabrication with no design rule check violations, and they must be manually fixed if present.

The goal of this thesis is to explore new algorithmic techniques to ease the time-consuming manual steps of physical synthesis and find better physical realizations of the gate netlist after logic synthesis.



(a) Over-the-cell routing. (b) Under-the-cell routing.

Figure 1.2: Connection using horizontal pins.

1.3 Thesis Contributions

The contributions presented in this thesis focus on the physical design stage, the first by systematically connecting cells using horizontal pins to reduce the number of design rule violations at the end of the physical flow, and the second by presenting algorithms to produce good macro floorplans at the beginning of the beginning of the flow. Both of the contributions are implemented and tested in the context of current industrial design flows, and the presented results prove the applicability of the techniques. These contributions are described in detail in the following sections.

1.3.1 Under-the-Cell Routing

The first contribution of this thesis proposes a systematic approach to allow the connection of standard cells via abutment, instead of connecting them using wires going over them, as illustrated in Fig. 1.2. By using cell generation tools for regular layouts, a cell library is enriched with cell instances that have lateral pins and allow under-the-cell connections between adjacent cells. Dynamic programming and graph methods are exploited to maximize the number of under-the-cell connections in a given circuit. The approach is integrated with an industrial EDA tool to prove its effectiveness. Experimental results show a reduction in the number of design rule check violations, pin counts and via counts with a negligible impact on timing. This work has been published in the following paper:

- Alex Vidal-Obiols, Jordi Cortadella, Jordi Petit. Under-the-Cell Routing to Improve Manufacturability. In *Proceedings of the ACM Great Lakes Symposium on VLSI 2017* (GLSVLSI), pages 125-130, presented at Lake Louise (Canada), May 2017.

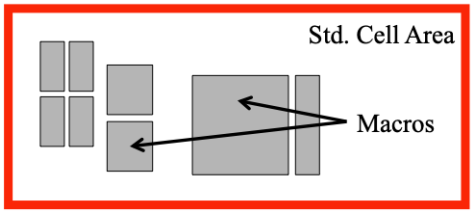


Figure 1.3: Blocks contain macros and standard cell area.

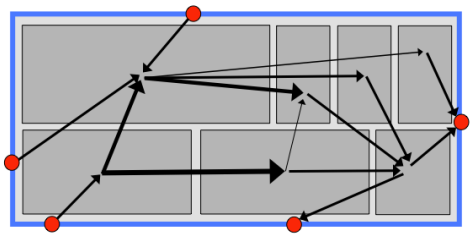


Figure 1.4: Floorplan of blocks, with their dataflow relations.

1.3.2 Macro Placement

The second contribution of this thesis has been conducted and deployed in collaboration with eSilicon, a leading chip design company. It presents a modern macro placer, pushing the state-of-the-art to handle the complex requirements of floorplanning. One of its key ideas is to obtain a macro placement via the floorplanning of blocks, which contain a set of macros and area, allowing them to have a flexible shape (Fig. 1.3). RTL-level information present on the name components in the netlist (such as hierarchy and arrays) is extracted and used to understand the structure of the circuit. The proposed multi-level hierarchical considers multi-cycle macros connections and locations of standard cell groups to find macro placements with good wirelength and timing properties (Fig. 1.4). The dataflow graph abstraction used to model the problem is also useful for engineers to easily understand circuit components and relations. After showing the effectiveness of the approach, it is extended with an adaptive multiobjective cost function. It allows the engineers to suggest macro locations to the tool, or to use algorithmic placement methods to find optimal locations for macro and guide the floorplan search.

Extensive experimentation proves that the obtained layouts after placement can reach better timing results than those of tape-out ready macro placements obtained by expert physical design engineers. Some of the layouts were brought to routing and passed to engineers who, after one or two iterations of manual modifications, managed to virtually close timing with few remaining DRC violations (Fig. 1.5). This work has been published in the following papers:

- Alex Vidal-Obiols, Jordi Cortadella, Jordi Petit, Ferran Martorell, Marc Galceran-Oms. RTL-Aware Dataflow-Driven Macro Placement. In *Proceedings of the Design, Automation and Test in Europe 2019* (DATE), pages 186-191, presented at Florence (Italy), March 2019. *(Corresponding to Chapter 5)*

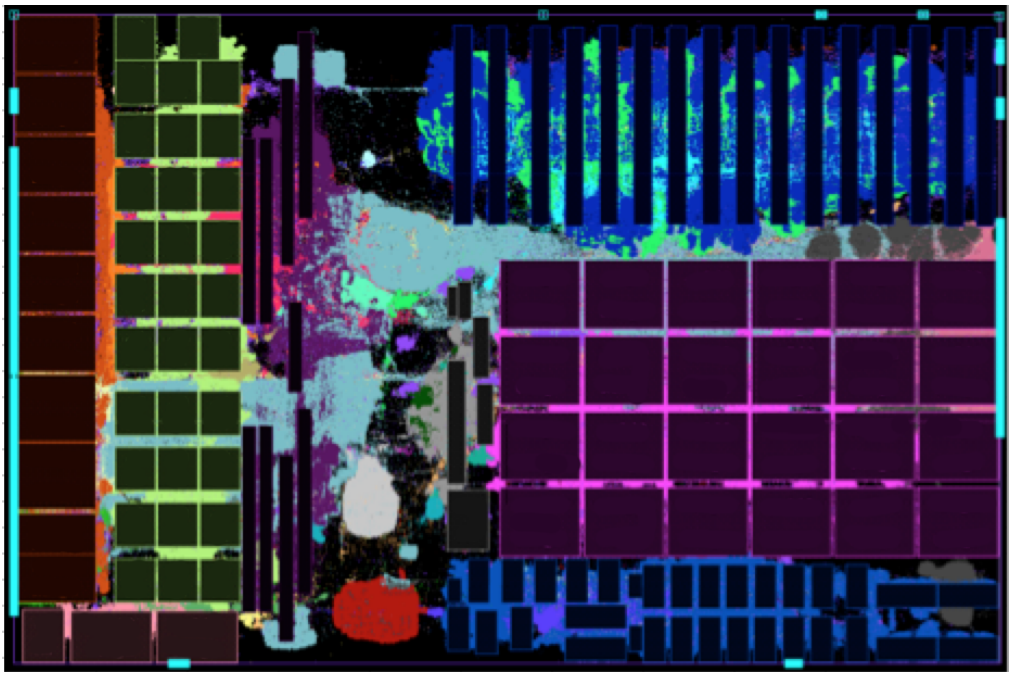


Figure 1.5: A nearly timing-closed layout generated by our tool.

- Alex Vidal-Obiols, Jordi Cortadella, Jordi Petit, Ferran Martorell, Marc Galceran-Oms. Design Mapper: Dataflow Analysis for Better Floorplans. In *6th Workshop on Design Automation for Understanding Hardware Designs* (DUHDE6), informal proceedings, Florence (Italy), March 2019. *(Corresponding to Chapter 5)*

An article extending a previous publication has been submitted to a journal and is under the review process:

- Alex Vidal-Obiols, Jordi Cortadella, Jordi Petit, Ferran Martorell, Marc Galceran-Oms. Multilevel Dataflow-Driven Macro Placement guided by RTL Structure and Analytical Methods. *(Corresponding to Chapters 5 and 6)*

1.4 Manuscript Organization

A summary of the organization of the main matter of the thesis is shown in Fig. 1.6. Chapter 2 introduces the reader to the context of the physical design flow, with its main steps and characteristics, and basic notions on semi-custom design and design for manufacturability. Chapter 3 contains

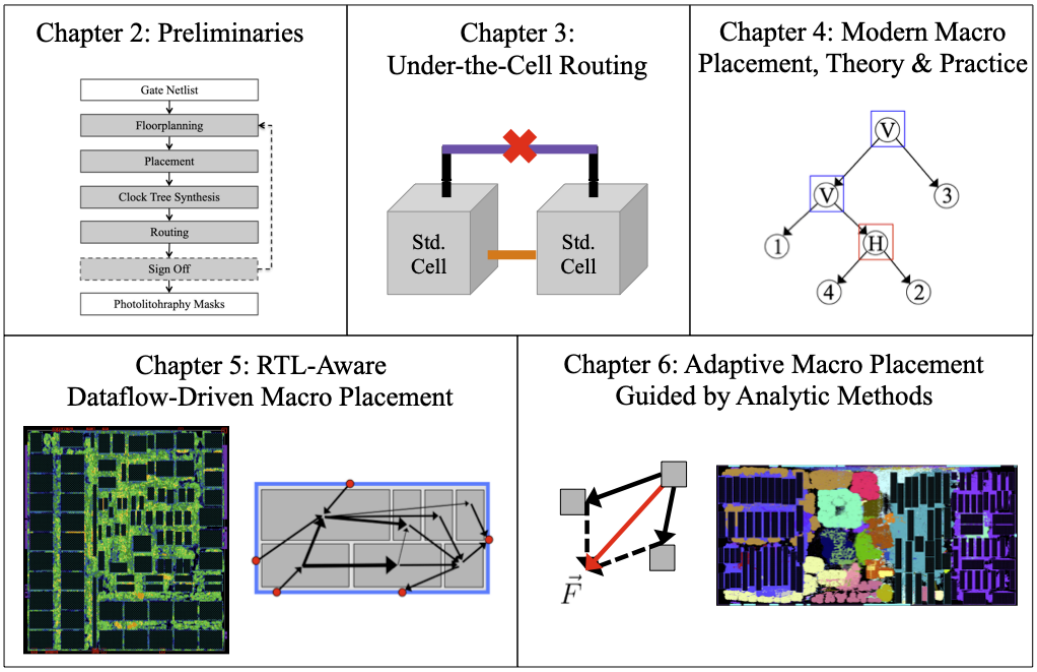


Figure 1.6: Thesis chapter organization, main matter.

the contribution on under-the-cell routing, with its motivation and related work, proposed approach, developed algorithms and results.

The rest of the thesis is devoted to the contribution on macro placement. It begins with Chapter 4, which contains an introduction to floorplanning concepts used in later chapters such as slicing trees, shape curves and simulated annealing. The second part of the chapter is devoted to a study of the state of the art of macro placers and industrial perspective on the problem, highlighting its needs and connections to academic work. The presented material motivates the development of HiDaP, a new macro placement tool which is presented in the next two chapters. Chapter 5 focuses on the overall flow of its algorithm (use of RTL information, dataflow inference and layout generation) and reports a first set of positive results. Chapter 6 adds to it by proposing an adaptive multiobjective cost function and using spectral and force placement methods to guide the floorplanner. It also features more exhaustive experimental results and layouts obtained for all circuits in the benchmark. Finally, Chapter 7 presents the global conclusions and some future lines of work.

Chapter 2

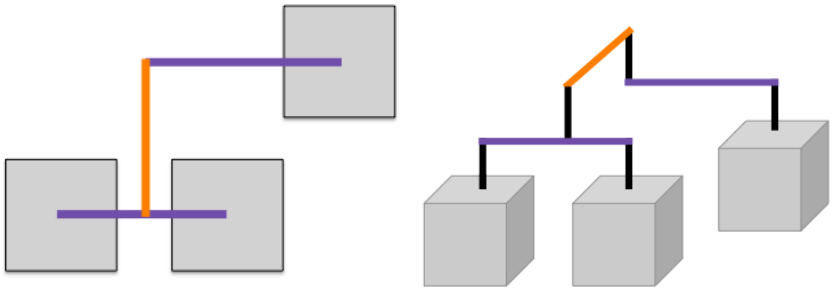
Preliminaries

This chapter provides the context to understand the scope and basis of the contributions of this thesis. First, an introduction to the semi-custom design methodology and its role as an enabler in current IC design is presented. It is followed by an overview of the physical design flow in modern technology nodes, with a brief description of the stages more closely related to the work presented in this thesis: floorplanning, placement and routing. More information on floorplanning techniques and state-of-the-art is exposed in Chapter 4. Finally, the chapter closes with an overview of the photolithographic process and how its increasing complexity has brought the need to take manufacturability into consideration.

2.1 Semi-Custom Design

The previous chapter showed that the design process for integrated circuits presents many challenges that are nowadays tackled with the help of design automation. As the number of transistors involved in the design grows, so does the complexity of chip design. The initial design methodology was full-custom design, in which the layout and interconnections of each individual transistor are considered. This process can potentially optimize the desired metrics of the circuit, such as area or performance, to the limit. However, as it is extremely expensive and labor-intensive, it is now used only for the most critical parts of circuits.

The most widely used design style, which draws on hierarchical properties to tackle the complexity induced by having millions of transistors, is *semi-custom* design. The methodology consists of using pre-designed layouts for small logic functions called *standard cells*, which normally have from 2 to 5 inputs, as basic bricks for the design of the circuit.



(a) 2D view. (b) 3D view.

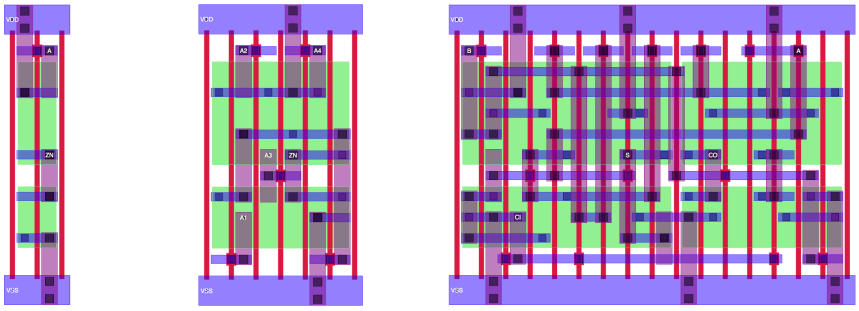
Figure 2.1: Standard cell connection.

Standard cells are small circuits in themselves, over which the rest of the circuit is constructed. A standard cell library is a set of standard cells that can be used for circuit design, which includes a full specification of each cell, with schematics, layout, physical characterization and others. All cells from a given standard cell library have a fixed height, thus converting the general layout of a chip in a set of rows of standard cells.

As in other design technologies, due to the increase in the amount of connections, the wiring takes place over the components. This is illustrated in Fig. 2.1, with the left showing a typical two dimensional layout view of three standard cells with their connections. The right figure shows the same connections in three dimensions, revealing that the connection uses two levels over the components: one for the horizontal connections (purple) and another for the vertical (orange). Each of these levels is called a *routing layer* or *metal layer*, and modern technology flows may require over 13 metal layers to route all connections. Normally, the first two metal layers are reserved for the routing inside the standard cells themselves.

For example, Figure 2.2 presents the physical layout of several standard cells. Their height is fixed at 13 rows, which are called tracks. Every cell has a superior wire connected to power, and an inferior connected to ground. They have been routed with two metal layers, the first for horizontal connections and the second for vertical connections, leading to a set of wires that can be seen to be very regular.

Designing with standard cells is divided in two steps, normally done by different teams and even different companies: cell generation and circuit synthesis. Cell generation is the process of creating the standard cell library itself. A factory or provider creates these standard cell libraries that are used by circuit designers to implement their integrated circuits.



(a) INV_X1 cell. (b) AND2_X1 cell. (c) FA_X1 cell.

Figure 2.2: Physical layout of several standard cells.

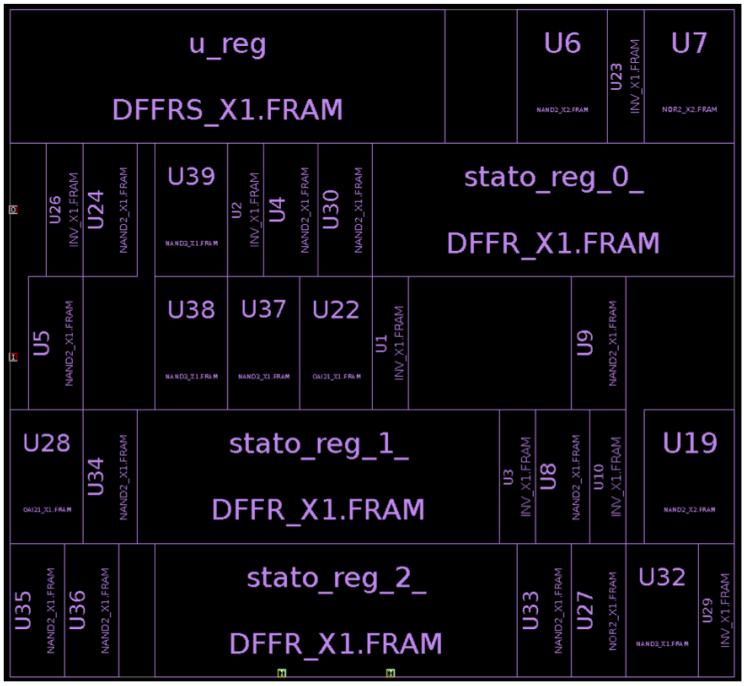
Each cell is full-custom and has extremely optimized placement and routings, given that they are determined once in their lifetime and will appear repeated all through many different circuits.

The physical synthesis process in the semi-custom design flow consists of creating a chip from the standard cell library by using the cells as basic constructions bricks, and is presented in more detail in the following section. Cells are first placed in the chip area, and given their height is fixed, standard cells are normally placed in rows. In a later stage (routing), standard cells need to be connected among them. Both processes, the generation of a standard cell library and the physical synthesis, are of interest to this thesis.

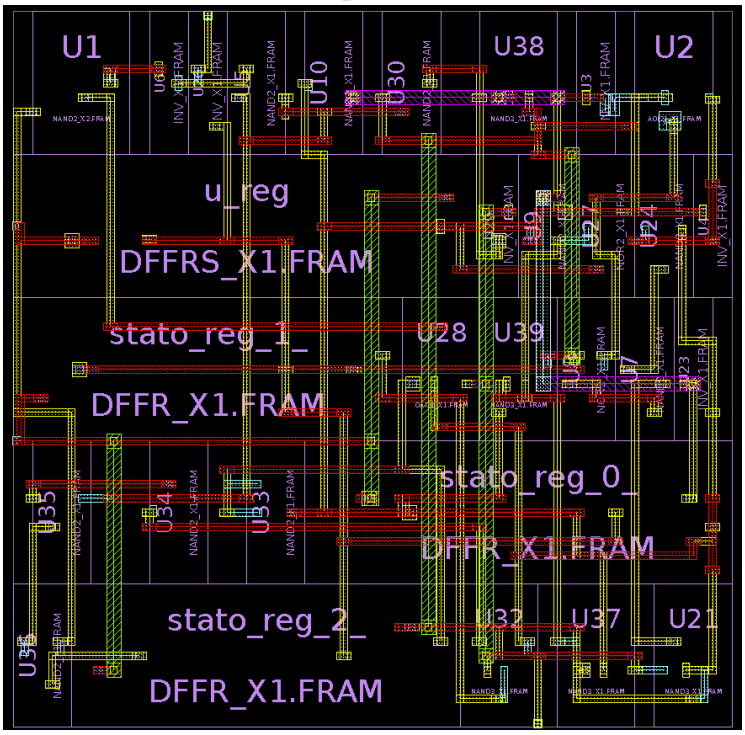
Figure 2.3a shows a small circuit that has been placed. The circuit comprises 29 standard cells, represented by the pink boxes with names, that have been arranged in five standard cell rows. Another similar example is shown after routing (Fig 2.3b), when later in the physical design flow, showing the wires connecting the components over the cells. The color reflects the metal layer the wire belongs to: blue for metal 1, yellow for metal 2, red for metal 3, green for metal 4 and pink for metal 5.

2.2 Physical Synthesis

Physical synthesis is the process that takes a circuit description as a gate netlist and defines the physical layout of the design, that is, the final coordinates of each transistor and connection metal segment. Figure 2.4 shows the most important steps during physical design in modern industrial VLSI flows. The closer to sign off the design is, the more accurate become its reported metrics (in terms of wirelength, timing or power consumption).



(a) Small placed circuit.



(b) Small routed circuit.

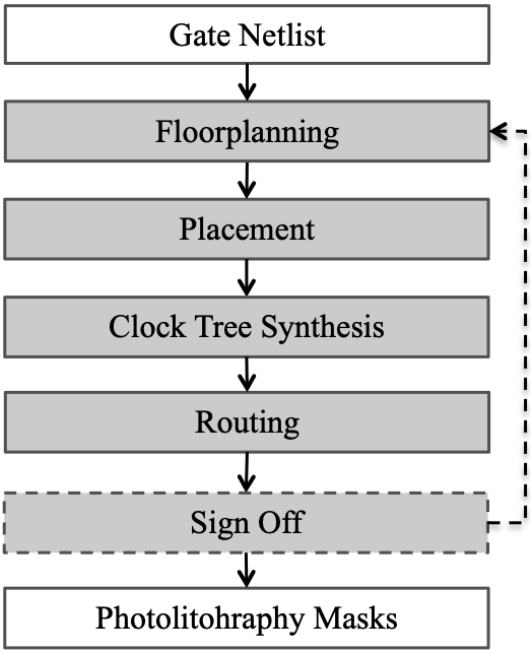


Figure 2.4: VLSI physical synthesis flow.

Floorplanning

The first step of the physical design process consists in prototyping the placement of blocks and fixing the location of macro components, black box circuits which are orders of magnitude bigger than standard cells. This is a critical, time-consuming process done manually by expert engineers by analyzing circuit connectivity and exploiting knowledge of the circuit structure given by the RTL designers if available.

Placement

During placement, the location of each standard cell is fixed. This is a fully automated process targeting at wirelength and timing minimization. The engineer may indicate placement regions or relative placement constraints before it, or may manually alter positions after it is completed.

Clock Tree Synthesis

The clock tree synthesis consists in the routing of the clock signal across the circuit. This step is taken before general routing due to its criticality. The process can be partially guided. Timing results become much more accurate after this stage.

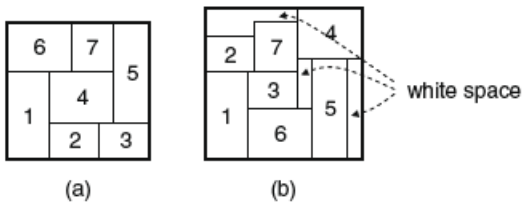


Figure 2.5: Optimal and non-optimal area floorplan. [9]

Routing
The routing process determines the precise paths for nets on the chip layout. The main aim is to complete all required connections on the layout without inducing design rule violations and meeting timing requirements. This step is also fully automated.

Sign Off
The final sign off stage consists of a series of tests and simulations to ensure that the design is ready for tape out, that is, to be sent to fabrication. It also includes static timing analysis, used to ensure all timing requirements are met. If problems are found they are solved manually but, if there are too many, the process begins again by starting from a different floorplan.

The work on this thesis focuses on easing the manual effort on floorplan, by providing tools with better macro placement capabilities, and at sign off, by proposing placement techniques to reduce the number of DRC violations at that stage. The following sections provide more background on the floorplanning, placement and routing stages as they are the ones involved in the contributions.

2.3 Floorplanning

Floorplanning is the first step in the physical design flow. It consists in identifying structures that should be placed close together, capturing relative positions but not necessarily always fixed coordinates. It can be considered a generalization of placement, a first draft on how the components will be allocated in the chip, allowing transformations of the components such as rotations and modifying their shapes. Simulated annealing and slicing structures [63] are widely used in this area. A floorplan can be optimized for metrics such as area, wirelength, routability

and others. Figure 2.5, taken from [9], shows two different floorplans for a given set of components. The floorplan on the left is optimal in area, but the one in the right introduces white spaces.

2.4 Placement

Placement consists in assigning cells to positions in the chip according to some cost functions while preserving legality (for example, with no overlapping). The inputs is the netlist and the goal is to find the best position for each cell considering wirelength, routability density, power and other metrics. Many placement styles exist depending on the design methodology it is integrated with (such as building blocks, standard cells or gate arrays). This step is tightly related to the next phase, routing. Some placement paradigms are:

- Constructive algorithms, such that when the position of a cell is fixed, it is not modified anymore. Some examples are cluster growth, min-cut [23], or quadratic-placement algorithms (such as Hall placement [29], the first analytical placer).

- Iterative algorithms, where intermediate placements are modified in order to improve some cost function. This would include analytical methods such as force-directed placement. Figure 2.6, taken from [9], shows several phases of the placement in a force-driven algorithm. The elements approach their final position iteration after iteration.

- Nondeterministic approaches, including metaheuristics like simulated annealing [71] and genetic algorithms [19].

All these methods can be combined to obtain a more accurate result. Additionally, other methods can be considered, for example a flow consisting of a global placement and legalization phase followed by detailed placement step. There are many interesting research directions in placement such as manufacturability-aware placement, but probably the most interesting for our project would be mixed-size placement, which considers the placement of both macros and standard cells.

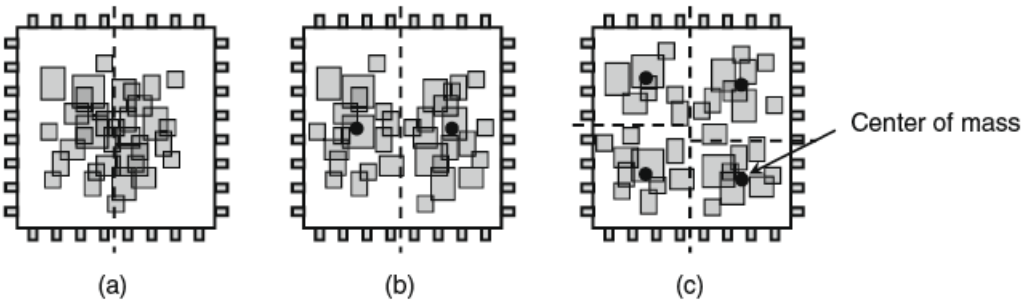


Figure 2.6: Placement of a chip. [9]

2.5 Routing

Routing is one of the multiple steps that take place in the physical design process. For the latest years many algorithmic techniques have been explored to address the complex problem of determining how the components of circuits should be interconnected. As the number of transistors per chip grows, the increasing complexity of the design becomes a challenge for the routing stage.

The main aim of the routing problem is to find a valid interconnection of terminals that honors a set of design rules. When routing, two kinds of constraints appear: performance constraints and design-rule constraints. The objective of the performance constraints is to make connections meet the performance specifications provided by the chip designers. Design rules impose restrictions on, for example, the minimum width of the wires or the wire-to-wire spacing. Another example of design rules is related to the layer models, which can be either *reserved* or *unreserved*. In the first case each layer is allowed only one routing direction, whereas the placement of wires with any direction is permitted in the other one. Most of the routers, however, use the reserved model because it has lower complexity and is much easier for implementation; as we will see, manufacturability has a great impact on many of the decisions taken during the physical design flow:

An important classification of routing algorithms is whether they aim at doing global or detailed placement. In the first case can be considered the coarse case of routing, in which we define big routing regions and propose paths for the signals, but without detailing the exact location of the wires. The task is left to detailed routing, which can work at a wire-segment level and decides the final position of the wires. Another interesting division exists between sequential and concurrent routers.

Sequential routing
In this schema we select a specific net order and then route nets sequentially according to that order. The quality of the solution greatly depends on the ordering given that an already routed net might block the routing of subsequent nets. Often, a *rip-up and re-route* heuristic is used to refine the solution. It basically consists in ripping-up some already connected nets and then re-route the ripped-up connections. It usually performs iteratively until all nets are routed, a time limit is exceeded or no gain is obtained.

Concurrent routing
Concurrent global routing tries to establish all connections at the same time. Therefore, whether or not a solution is found does not depend on any net ordering. One of the most popular approaches is to model the layout as a graph and then use *0-1 integer linear programming*. However, given its prohibitive running time, another approach would be to solve the continuous *linear programming* relaxation and the transform the fractional solution to integer solutions through a rounding scheme such as randomized rounding. In practice, such techniques are embedded into larger global routing frameworks that use a hierarchical, divide-and-conquer strategy.

The contributions presented in Chapter 3 includes the use of a detailed router for standard cells with traits from both concurrent and sequential routing. This means that it finds concrete routes for the wires at a physical level, and that the algorithm uses concurrent routing to find an initial solution, but later takes on a sequential routing strategy in order to improve the result of the initial solution with rip-up and re-route.

2.6 Design for Manufacturability

As we have seen in Section 2.2, the final result of the design and synthesis processes are masks to be used during the photolithographic process, illustrated in Figure 2.7. A source of light emits monochrome light that traverses the mask in which the circuit patterns have been engraved. The die, covered with a photosensitive layer, reacts to the exposition to light. The parts that have been illuminated remain whereas all the others are eliminated after an etching phase. Layers grow one above the other until all masks have been applied. The patterns on the masks (each corresponding to one of the metal layers) are the final product that the physical design produces for a given circuit abstraction.

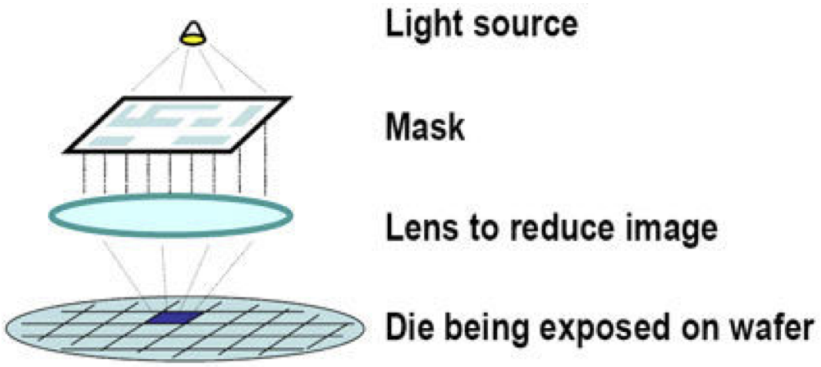


Figure 2.7: Lithographic process.

Over the past few years, additional problems have been adding up to the challenges of circuit design itself. As the size of transistors decreases, the manufacturing process has become increasingly complex. Previously, the standard way of scaling down the size of technologies was to reduce the wavelength of the source of light. However, for recent technology nodes in which the size feature is smaller than the minimum wavelength ($193\,nm$), this has become a source of variability and error during the photolithographic process [64]. This gap, called the lithography gap, is illustrated in Figure 2.8. Technological solutions such as extreme ultra-violet light (EUVL), electron beam lithography and others have been constantly delayed, prompting the design and manufacturing engineers to find extra solutions for the problems caused meanwhile.

Litho-friendly layout techniques must be considered to deal with all this increasing complexity. As we get into newer technology nodes, the amount of design rules increases enormously, resulting in more challenges during the design stage. Manufacturability awareness during the design phases of the circuit has become a must in order to produce good yields at the end of the fabrication process. For example, one of the techniques used during the lithography process is double patterning (DP), which consists on dividing what normally would be a single mask in two masks as shown in Figure 2.9. The masks would be exposed one after the other during the manufacturing process.

By using this method, the effective pitch can double, improving lithography resolution. However, masks must be assigned to the components, which is something that must be done during the design process. The basic rule is that two components that are too close cannot share the same mask, and the problem does only get more complex when moving to triple or multiple mask patterning, the more general case, as feature size

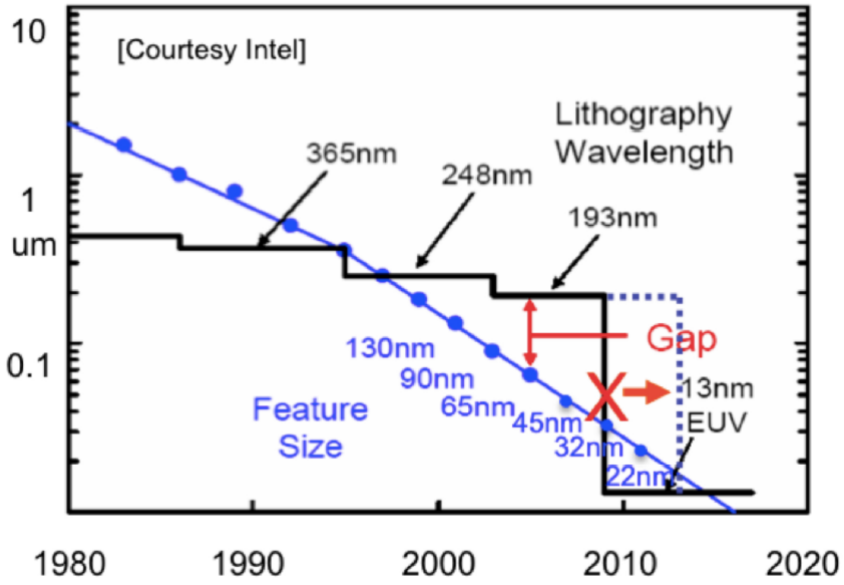


Figure 2.8: Lithography gap. Source: [64]

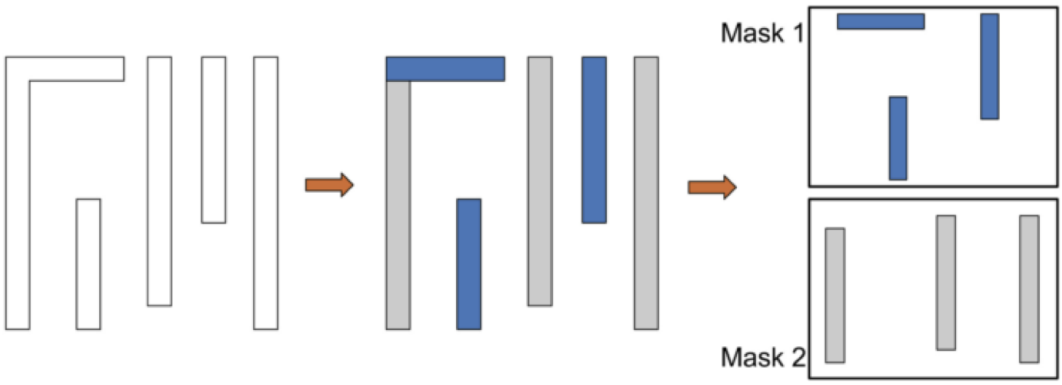


Figure 2.9: Double patterning example. Source: [64]

decreases. Even if in the past the design and manufacturing stages might have been relatively isolated, this makes for a perfect example of how manufacturability issues are becoming more and more important during design, and considering the delays in technology development this trend will only continue to grow in the future to come.

One of the strategies that has been increasing in popularity in the recent years is the regularity of the designs. In regular designs, we want to

use repetitive layout patterns and blocks, which require simplified masks and allow for a better process control [56]. This can be achieved by using simplified design models and restrictions, such as gridded models, which a priori would restrict the possible designs, but on the other hand allow for new families of algorithms to be used during the physical design phases. Additionally, double patterning and optical correction techniques are not as effective on complex layouts with arbitrary shapes, but are most useful when using regular layouts.

2.7 Conclusions

This chapter presents some preliminaries on the VLSI design flow, focusing on semi-custom design, the physical design flow and design for manufacturability. It is complemented by Chapter 4, which takes a deeper look into floorplanning and macro placement. The concepts introduced in these pages provide the foundational framework to motivate and develop the contributions that are presented in the following chapters.

Chapter 3

Under-the-Cell Routing

The impact of routing congestion on metrics such as performance, area and yield poses a major obstacle that must be addressed. This chapter proposes proposes under-the-cell routing, a method to aid the routing stage to improve manufacturability while preserving quality of results.

The presented approach proposes to systematically exploit the connection of adjacent circuit components using lateral connections in the internal routing metal layers. Standard cells are designed in a way that they can be treated as black boxes during physical design. However, this abstraction often prevents an efficient use of its internal free resources. White space in those layers can be used for routing in current design flows, but to a fraction of the potential use that could be achieved by exploiting it systematically. Relying on the regularity imposed by technology constraints, standard cell libraries can incorporate several similar cell layouts for each logical cell. These layouts are enriched with lateral pins that can be abutted against neighboring cells for direct connection avoiding the high level metal layers. Cell modularity allows the design tools to pick the most suitable cell layout after the placement stage.

In this chapter, an approach to generate cells with regular layouts and lateral pins is proposed, together with algorithms to maximize the impact of under-the-cell routing. The techniques are integrated with industrial design automation tools. This method decreases both pin count and congestion on the upper metal layers, resulting in a reduction of the number of design rule check violations with negligible impact on timing. The approach is technology independent and can be used in any fixed-height standard-cell technology node.

3.1 Motivation

The progressive miniaturization of technology and the unequal scalability of component and interconnect sizes aggravate the routing congestion problem and have a negative impact on manufacturability. Routing is becoming an increasingly challenging task as on-chip component density grows [66]. Given that component sizes are decreasing faster in comparison to metal connection sizes, pin accessibility in new technology nodes is becoming a critical problem. Over the last years, there has been a growing interest in preventing hard-to-route designs, focusing in the interaction between the placement and routing stages [81] [82]. Routing congestion, as well as pin density, are some of the metrics considered during placement in modern chip placers.

Moreover, the chip design community is presented with the challenging task of finding new ways to improve manufacturability in the latest technology nodes. Some lithography processes have added coloring requirements for wires [64], and printed layout quality is becoming increasingly dependent on details such as line-end distribution [101]. The profusion of new and complex design rules has posed additional constraints to pin placement and accessibility, creating even more congested zones in the most pin-dense areas.

The increasing congestion and extra constraints imposed by the manufacturability process translate to harder-to-satisfy design rule checks. Having no design rule violations is mandatory at the end of the physical design process, and sometimes DRC violation hotspots can not be solved with small routing incremental changes, and may demand going back to modifying macro placement. In this context, the aim of the work presented in this chapter is to reduce the amount of DRC violations by systematically connecting cells with lateral pins using the lower metal layers, reducing the number of vias and connections in the high metal layers, thus reducing congestion and induced DRC violations.

3.2 Related Work

The increase in routing congestion is generally being addressed in a per-technology basis. One example is self-aligned double patterning (SADP), one of the technologies used for multi-patterning at $10nm$ nodes [64]. The concern for pin accessibility, ultimately resulting in routing congestion, can be seen in several proposals that consider constraints imposed by this fabrication process to the synthesis flow. The work presented in [89]

considers the impact of SADP, as well as other technology constraints, to decide how to access I/O pins and offer maximized flexibility to the routing stage. Another interesting example is presented in [90], with a router that explicitly considers pin accessibility constraints induced by the SADP process. However, these methods are only valid for specific processes, and additional technology-independent solutions would be desirable.

From a circuit design point of view, a specific kind of placement and routing interaction consists on performing incremental routing-aware placement after a traditionally good placement has been obtained. Reference [68] analyzes a placement and performs inflation and spreading in the zones with higher congestion and pin density. Reference [102] also applies iterative spreading and congestion-aware detailed placement. Whereas these two approaches use a global router to determine the congestion of a placement, the authors of [76] proposes to use pin accessibility to identify hard-to-route cells and determine local congestion across the design. Later, cell movement among placement regions, inflation / legalization and position within the standard cell row are determined using congestion metrics. Recent work incorporates machine learning techniques to predict routability [6] [75], and even as a guide for a placement optimization process to reduce DRC violations [7].

From a standard cell design point of view, automation tools are key to the generation of manufacturability-friendly cells [22]. In particular, the routing of standard cells poses a particularly difficult challenge, as many optimization objectives must be satisfied at the same time. The optimization of detailed routing has also been addressed in a technology dependent basis [72], whereas other approaches such as [21] present technology-independent algorithms that can be parametrized to satisfy particulars requirements, including lithography-induced design rules. Recent approaches explicitly targeted pin accessibility [69].

The idea of abutting cells so that they connect horizontally has already been suggested in a patent by Cadence, a leading EDA company [52]. Their proposal triggers the abutment of standard cells which are overlapping or at a predetermined proximity, and focuses on the physical implications of the abutment: some pins would need shortening, swapping or merging. Our method builds on this approach to systematically exploit horizontal connections by extending a standard cell library to support lateral pins and proposing algorithms to maximize the occurrence of abutment connections in placed layouts.

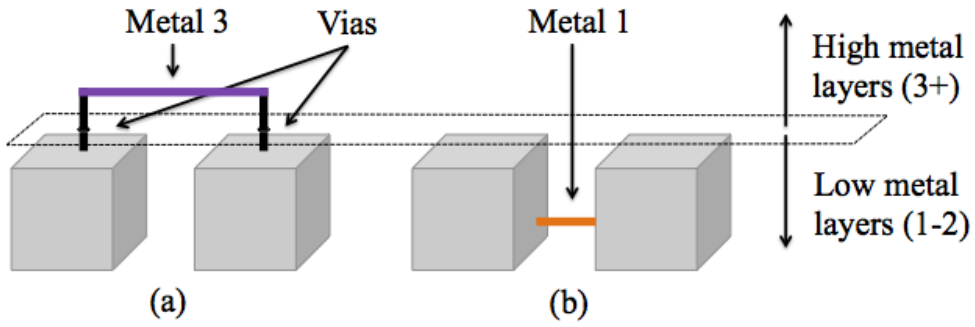


Figure 3.1: Symbolic cell interconnection: over-the-cell and under-the-cell routing.

3.3 Contributions

Whereas standard cells in traditional synthesis flows only offer I/O pins on their top, the aim of our method is to take advantage of cells whose I/O pins can also be on their sides, reducing pin count and enabling better manufacturability.

Lateral Pins.

Fig. 3.1 conveys this idea by representing two pairs of adjacent standard cells that must be connected assuming that metals 1 and 2 are used for internal cell connections. The cells in Fig. 3.1(a) show the traditional connection scheme: High metal layers (at least metal 3 and 4, depending on metal direction) must be used to connect the two pins of the cells. The cells in Fig. 3.1(b) illustrate the new connection scheme: They can be connected directly using the lower level metal layers by abutting one cell to the other without reaching higher metal layers via I/O pins. We refer to a net that can be connected via lateral pins without accessing the high metal layers as a *buried net*.

More specifically, Fig. 3.2 shows two views of a small circuit consisting of four NAND gates. The dotted wires in the schematic view represent connections that are routed using under-the-cell routing. The lower part shows the layout implementation of the circuit using gates with lateral pins. Each *NAND* cell layout presents a different lateral pin interface: C_1 has no pin, C_2 has an output lateral pin to the right, C_3 has an input lateral pin to the left and an output lateral pin to the right, and C_4 has an input lateral pin to the left. To simplify the design flow, the lateral pins are enforced to be in the same track (marked in red in the example), and each cell is allowed to have at most one input and one output lateral pin. The rationale for these decisions is discussed in Section 3.4.

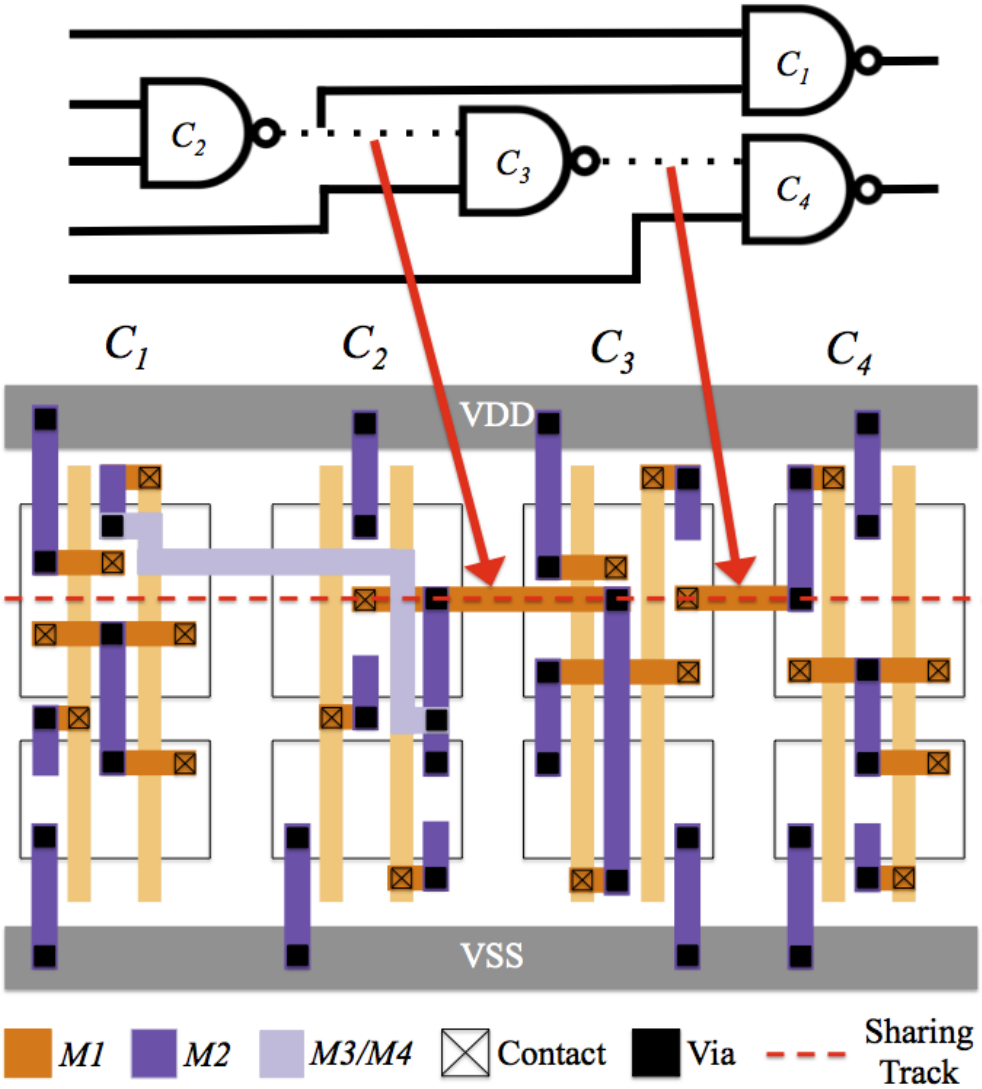


Figure 3.2: Small circuit consisting of four NAND gates.

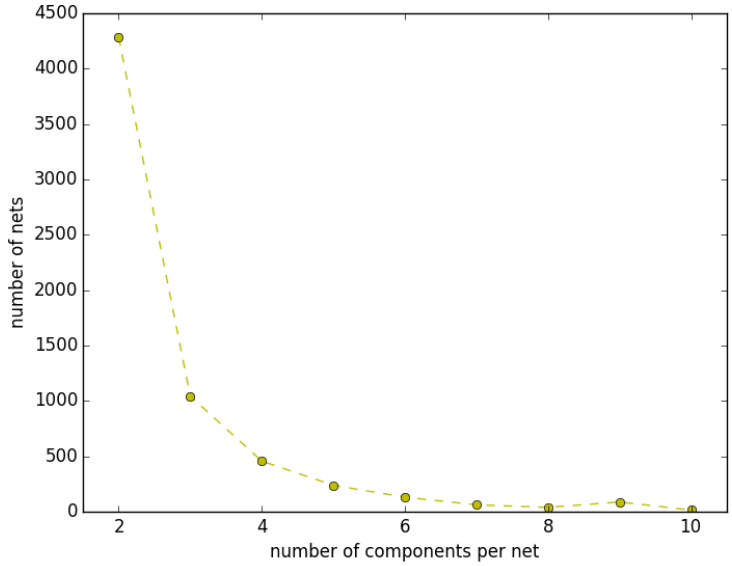


Figure 3.3: Number of nets per number of net components.

In the example, the output of C_2 is routed to one of the inputs of C_3 by a straight metal-1 wire extension, and the same happens with the output of C_3 being connected with one of the inputs in C_4. The remaining connection between C_2 and C_1 is done via high-metal layers. This circuit would normally require five I/O pins for its internal connections, but given that we have routed two of the three nets using under-the-cell routing, only two I/O pins are needed. Notice that the connection between C_2 and C_3 does not allow us to also connect C_2 and C_1, given that each cell can have at most one output lateral pin. The process of mapping cells to layouts becomes an interesting optimization problem.

Potential Gains.

If we want to remove the I/O pins of a net we need to connect all its components using lateral connections. Statistical analysis shows that the majority of the nets of our circuits are composed of 2 and 3 components, as shown in Fig. 3.3. If those frequent nets could be completely routed using subways we would be able to have less congestion on the upper levels of metal layer and either reduce area by augmenting core utilization or even route the same circuit using one less metal layer.

In order to assess the potential impact of our method, in Table 3.1 we consider several metrics obtained from the synthesis of two circuits by IC Compiler [35] using the Nangate ($45nm$) library [61]. The optimizations

Table 3.1: Initial exploration.

Benchmark	# Cells	Cells for 90% covering	Buried nets	Buried connections
b17	17344	18.4%	18.4%	17.9%
b18	36589	16.0%	19.8%	18.6%

90% cell cover indicates the percentage of cells of the library needed to cover the 90% of the cell instances in the circuit. Buried nets indicates the percentage of 2-pin nets that could be buried under the optimistic assumption that all pairs of adjacent cells can be connected through lateral pins. Buried connections represents the percentage of connections between general n-pins nets that could be potentially buried under the same assumption.

to maximize the number of lateral connections presented in later sections are not applied in these initial experiments. These preliminary results suggest that, 1) only a few cells must be adapted to allow lateral connections in a substantial part of the circuit and that, 2) a significant number of connections could be buried to low-metal layers.

To fully take advantage of this novel technique, two major challenges appear:

- The library of standard cells must be extended with new layouts for each cell that has lateral pins. In particular, re-routing the cells is necessary to use their versions with lateral pins.

- Traditional EDA flows must be enhanced with a lateral pin-aware placement flow in order to take maximum advantage of the possibilities offered by under-the-cell routing. In particular, one wishes to maximize the number of laterally connected adjacent cells.

Key Contributions.

The collaboration of both cell library providers and EDA vendors is required for the proposed flow to be applied at industrial level. This chapter proposes an approach to address the two previous challenges, along with experimental results to show the validity of the approach. The main contributions of this chapter are:

1. The use of lateral connections to reduce the complexity of higher metal layer routing, improving circuit manufacturability.

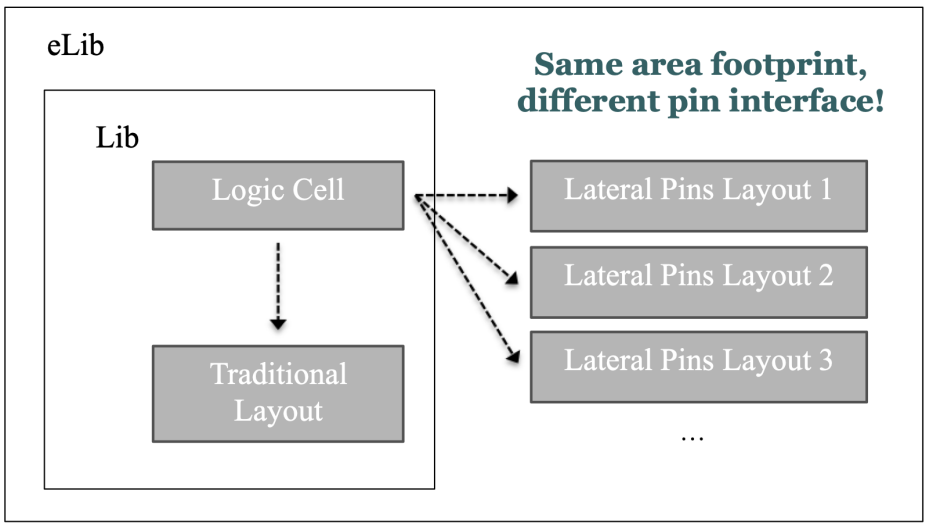


Figure 3.4: Extended cell library *eLib*.

2. A systematic approach that exploits regularity to generate standard cell layouts enriched with lateral pins.

3. A dynamic programming and a graph-based algorithm to maximize the number of under-the-cell connections in a given circuit while preserving timing.

4. Experimental results showing a reduction in the number of design rule check violations, pins and vias with negligible timing degradation.

3.4 Standard Cells with Lateral Pins

Cell library providers must enrich their libraries incorporating cell layouts with lateral pins. Given an original standard cell library *Lib*, an extended version *eLib* is created as shown in Fig. 3.4. In *Lib*, each logical cell is associated with a layout without lateral pins, whereas in *eLib*, each logical cell is mapped to the corresponding layout on *Lib* and to several other layouts with a set of lateral pins. The interface of these lateral pins is different in each one of the additional layouts, but all of them have the same area footprint.

As the number of cells in *eLib* is significantly larger than *Lib*, the first steps of the chip synthesis flow (e.g. technology mapping) will use only

Lib to avoid an increase in algorithm complexity. The use of *eLib* is postponed until placement has been completed, at which point the final cell layout for each cell instance is determined.

Creating the extended library *eLib* requires some important decisions to be taken:

- Which pins must be provided as lateral connections?
- Are traditional pins needed in cells with lateral pins?
- At which cell tracks must the lateral pins be accessible?
- How to generate the cells with such features?

3.4.1 Defining the I/O Interface

Potentially, any pin could be made accessible at any track if the routability of the cell would allow it. However, generating cell instances with all possibles subsets of pins accessible at any track would make the exploration of solutions unaffordable and the size of the library unmanageable. Based on empirical experimentation, a pragmatical approach has been taken by imposing the following two constraints:

One-Input/One-output sharing. Cells are only allowed to share (at most) one input and one output lateral pin, each one at a different side of the cell. The rationale behind this decision lies on the fact that most buried nets have 2 pins, thus requiring a balanced number of input and output pins. Moreover, most cells only have one output pin. In the case of sequential elements, the lateral pins are reserved for the `D` and `Q` (or $\bar{\texttt{Q}}$) pins.

Unique sharing track. One of the tracks is chosen to be the sharing track *for all cells*, thus unifying the position for lateral pins. This greatly simplifies routing as lateral pins are always connected via abutment, or with a straight metal segment if there is white space between the cells.

The fact that the final cell layout is decided for every cell after placement allows to synthesize cells without the full set of traditional I/O pins. In general, cell layouts with an input lateral pin can skip its corresponding traditional pin, given that they have a unique driver. However, cells with an output lateral pin might need to keep the corresponding traditional pin depending on whether they are driving only their neighbor cell

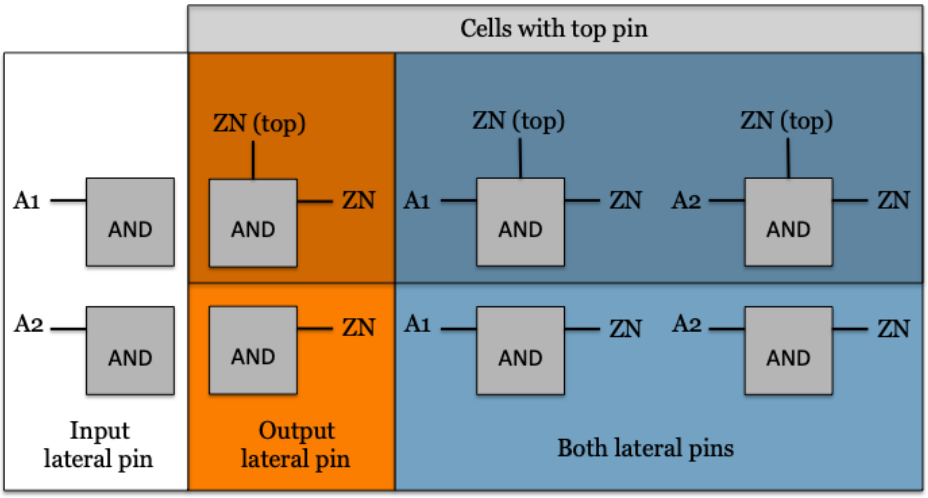


Figure 3.5: *eLib* layouts of a 2-input, 1-output AND cell.

or also other cells of the circuit. The top I/O pins removed from layouts with lateral pins are called *buried pins*.

Given the constraints mentioned above, three kinds of cells with lateral pins exist: cells with an input lateral pin, an output lateral pin, or both. For every cell in the last two groups we can have two versions, one keeping the I/O pin for the output and one without it. An example of the generated cells is shown for a 2-input, 1-output AND cell in Fig. 3.5. Since a large majority of cells have only one output, it is easy to see that the number of layouts in *eLib* will be about three times the number of *input pins* in *Lib*.

3.4.2 Generating the Cells

When synthesizing different instances of the same standard cell, transistor placement can be kept identical in all the layouts. However, the internal routing of each cell instance must be recomputed to adjust it to its interface, as shown in Fig. 3.2.

In order to automatically generate the routing for each extended cell, we advocate for the use of a Boolean rule-based approach such as the one described in [21]. This approach is efficient, technology-independent and parameterizable for different fabrics and design rules, including support for multiple-patterning lithography. Given the gridded transistor placement of a cell, such router generates a Boolean formula that encodes the

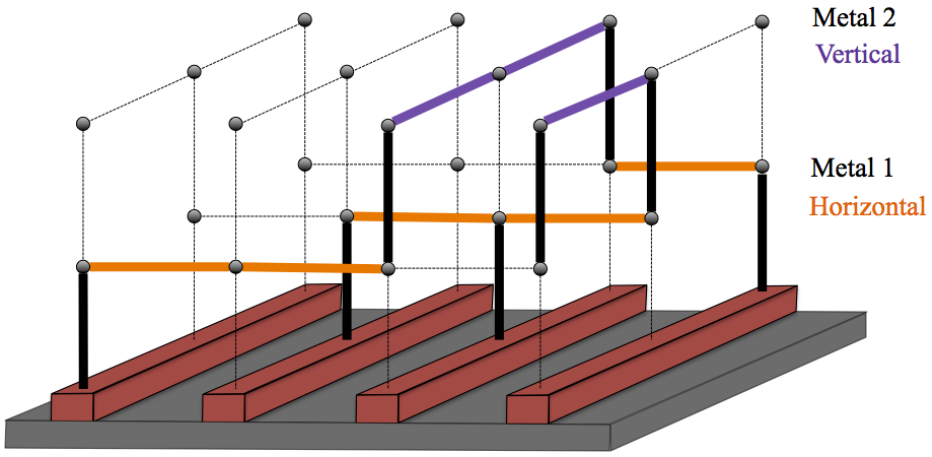


Figure 3.6: Grid representation for internal cell routing.

routing problem and feeds it to a SAT solver. The returned model is translated into a valid routing.

The underlaying representation for a cell is a discrete 3D grid, whose edges correspond to potential metal segments; see Fig. 3.6. For each edge in the grid, a set of Boolean variables describe whether certain nets of the cell use or not that edge and, therefore, ultimately define whether or not a segment of wire must be laid out on that edge. Using these variables, the routing problem is represented by a Boolean formula

$$F \equiv C \wedge DR \wedge R$$

where C defines some consistency constraints between the variables, DR imposes the design-rule constraints (including multiple patterning lithography), and R represents the routability constraints. In particular, the routability constraints encode that each net gets fully connected without short circuiting with other nets. Because of Euler's graph theory, these routing constraints can be specified requiring that each wire segment must have exactly two neighbors, except the end-points of the nets, which must have exactly one neighbor.

In particular, routability constraints guarantee that all pins of each net are connected. Such formulation can handle end-points whose position is either fixed at some grid point or selected from a specified set of grid points. In our context, lateral pins must be aligned with the sharing track at one side of the cell, whereas conventional pins can be located at any grid point of the top layer of the cell.

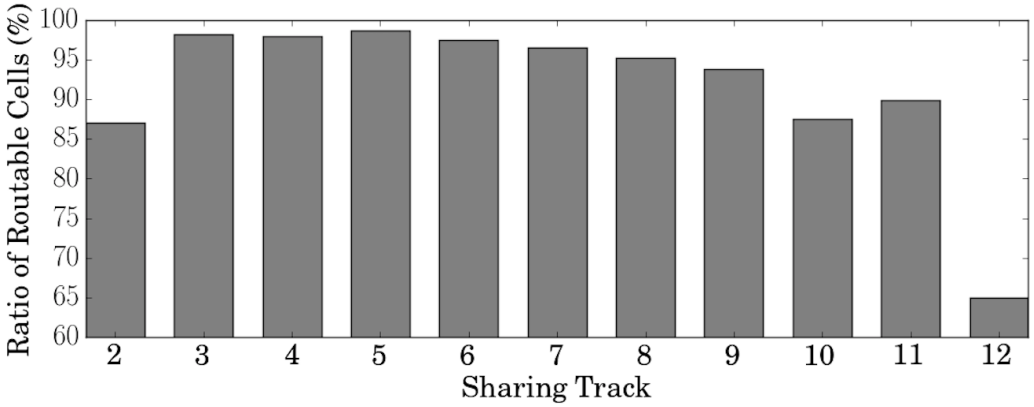


Figure 3.7: Routable cells per position of sharing track.

3.4.3 Selecting the Sharing Track

In order to find the sharing track that yielded more routable cells, we have undertaken the task of routing all cells in the Nangate $45nm$ cell library [61] using the routing approach discussed in Section 3.4.2 and honoring a commercial set of design rules (including rules for double patterning) on placements with minimal area. The template uses 13 tracks as shown in Fig. 3.2: track 1 is reserved for VDD, track 13 for VSS, and tracks 2 to 12 for signal routing. Using the criteria for external pins described in the previous subsection, the extended library demands a total of 1576 layouts for the 125 cells.

The exploration began by obtaining routings with lateral pins on a reduced set of hard cells. The results of this exploration are shown in Fig. 3.7. The vertical axis represents the percentage of cells that have been routed using the track indicated in the horizontal axis as sharing track. On the basis of this experiment, we have chosen track 5 (with a success ratio of 98.7%) to be the sharing track of the extended cell library. Then, using this sharing track, a valid routing was found for all cell instances except nine of them, leading to a routing ratio of 99.4% our *eLib*. In later experiments it was observed that virtually the 100% of the demanded layouts with lateral pins were among those successfully routed. Example of layouts obtained by the regular routing using these techniques are shown in Fig 3.2.

Fig. 3.8 shows two versions of the AND2_X1 cell. The one of the left has been routed with a special constraint imposing that the position of the 5th track at the left boundary should be occupied by the input A1, the pink signal. The one on the right requires the input A2, blue signal, to

be in the position of the 5th track in the left boundary. Depending on the final placement of the circuit either cell, or another of the copies, will take the final place of any given AND2_X1 cell, depending on the adjacent cells and their potential connections in every different case.

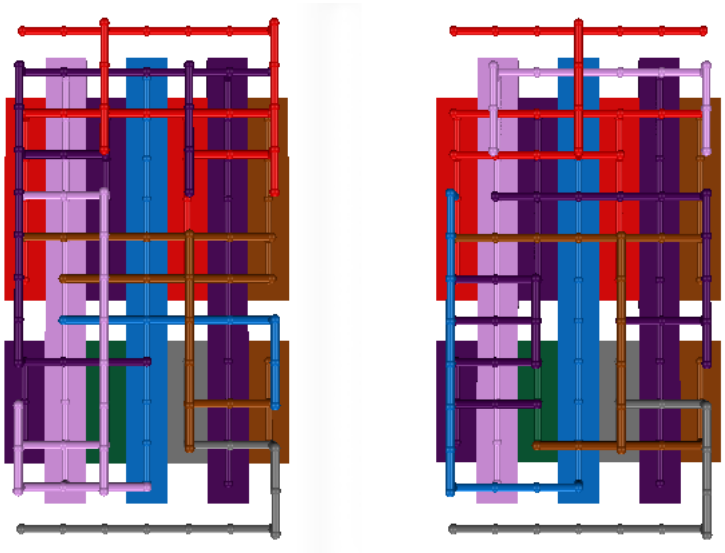


Figure 3.8: Two possible grid routings for AND2_X1 (using [21]).

3.5 Placement and Routing

The physical synthesis tools must be enhanced to take advantage of under-the-cell routing, in particular by mapping cells from *Lib* to *eLib* and modifying the placement to increase the opportunities for lateral connections.

3.5.1 Synthesis Flow

Under-the-cell routing requires new steps to be added between the placement and routing stages of the physical design flow, as shown in Fig. 3.9. The light boxes represent the usual steps in physical synthesis, whereas the dark boxes represent the newly introduced steps. The proposed process is transparent to the physical design flow before the cell assignment step, after which *eLib* is used.

The first new step is a *microplacement stage* which is performed after the tool has completed the placement. One of the conditions of lateral connections is that both cells to be connected must be placed side

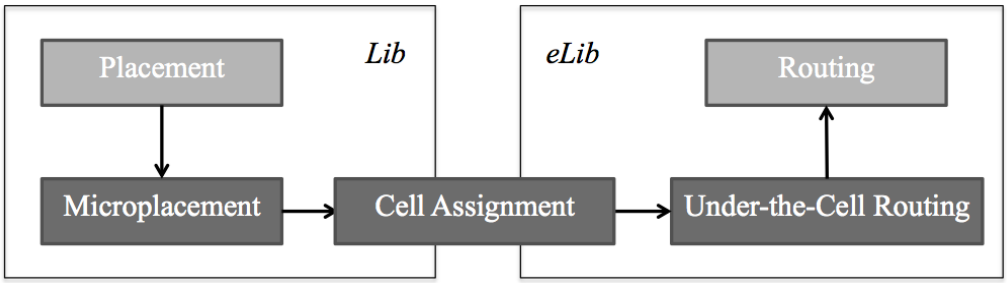
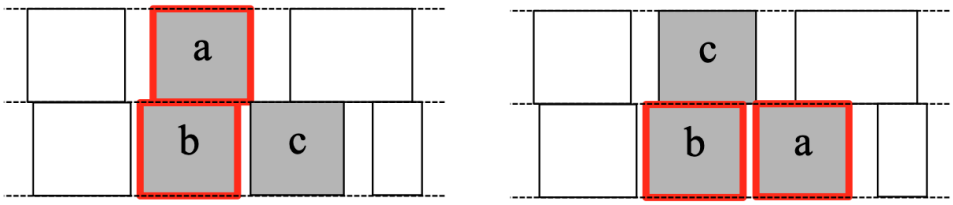


Figure 3.9: Proposed design flow modifications.



(a) Before microplacement. (b) After microplacement.

Figure 3.10: Role of microplacement.

by side. Current placement tools optimize very complex functions that do not pay particular attention to abut cells that need to be connected (Fig. 3.10a). The microplacement stage presents the challenging problem of introducing small modifications to the original placement where opportunities to exploit adjacent connections are detected while preserving timing (Fig. 3.10b).

The second step is *cell assignment*. Each placed cell from *Lib* can be potentially substituted by one of the cell layouts in *eLib*. The enriched library provides several viable candidates, and the assignment of one cell to a layout with lateral pins affects the assignment of its neighboring cells. Cell assignment also impacts the amount of I/O pins that can be saved by using lateral pins. Finally, the *under-the-cell routing stage* adds horizontal wires for buried nets of cells which are not immediately adjacent.

3.5.2 Microplacement

The microplacement algorithm takes a valid placement and detects cells that could be connected using lateral pins. The entire flow is shown in Algorithm 1. The input is a placement P of cells from *Lib* generated by an generic placer, that is first partitioned into several regions r. Then, a greedy method is used to maximize the number of lateral connections. It

generates relative placement constraints, which impose that some group of cells must be adjacent, for every region of the circuit. Finally, the generic tool performs an incremental placement on P with the newly added relative placement constraints, trying to improve the number of buried nets and pins.

Algorithm 1 Microplacement stage.

1: **Input:** *Placement P and slack tolerance ε*
2: **Output:** *Placement P'*
3: $RPC \leftarrow \emptyset$
4: $R \leftarrow$ *partition_in_regions(P)*
5: **for each** *region* $r \in R$ **do**
6: $G \leftarrow$ *build_graph(r)*
7: $V' \leftarrow$ *maximum_independent_set(G, ε)*
8: $RPC \leftarrow RPC \cup$ *relative_place_constraints(V')*
9: $P' \leftarrow$ *incremental_placement(P, RPC)*

Two mechanisms are applied to preserve the quality of the original placement.

Region partition: Lateral connections maximization is applied to exclusive regions of a fixed size, to preserve global placement.

Slack control: Relative placement constraints are forbidden on cells with slack lesser than a factor ε to avoid significant timing alterations.

Maximizing Lateral Connections

Consider a set of placed cells $C = \{c_1, ..., c_n\}$. Each cell c_i has a set of input and output signals, $I(c_i)$ and $O(c_i)$, respectively. Each pair of cells, c_i and c_j defines a set of potential connections $S_{i,j} = \{s_{i,j,k} \mid k \in O(c_i) \cap I(c_j)\}$, where $s_{i,j,k}$ means that signal k is an output of c_i and an input of c_j. The union of all potential connections is

$$S = \bigcup_{\substack{1 \leq i,j \leq n \\ i \neq j}} S_{i,j}$$

Because of the restrictions explained in Section 3.4, not all potential lateral connections can coexist in the final placement. The method considers such restrictions only for every pair of potential lateral connections. Fig. 3.11 shows the form of these incompatibilities for a given pair $s_{i,j,k}, s_{i',j',k'}$:

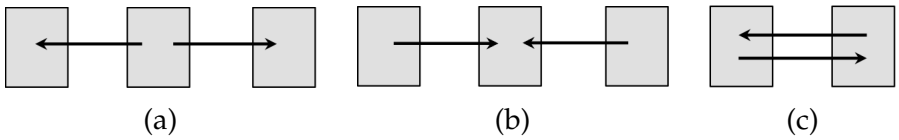


Figure 3.11: Microplacement incompatibilities.

- The case $i = i'$ implies a cell with two output lateral pins (Fig. 3.11a).
- The case $j = j'$ implies a cell with two input lateral pins (Fig. 3.11b).
- The case $i = j' \wedge j = i'$ implies lateral connections in both directions between two cells (Fig. 3.11c).

We can now define a graph $G = (S, E)$ as the graph where each vertex represents a potential connection and each edge an incompatibility:

$$E = \{(s_{i,j,k}, s_{i',j',k'}) \mid i = i' \vee j = j' \vee (i = j' \wedge j = i')\}.$$

By construction, the problem of maximizing the number of lateral connections of the cells C corresponds to finding a maximum independent set (MIS) of G, that is, the largest set of vertices such that no pair of them is adjacent. Each vertex can then be translated to a relative placement constraint that imposes both of the related cells to be placed one besides the other.

As the maximum independent set is a well-known NP-hard problem, a greedy approach is used as an approximation. A score is assigned to each potential lateral connection (represented by a node in our graph) and the algorithm iteratively picks the one with the highest score, incrementally propagating the incompatibilities on the graph. This score is obtained by performing a normal routing of the circuit and comparing the estimated wirelength and the final route of the nets, prioritizing nets that have taken a long detour in order to be routed.

3.5.3 Cell Assignment

Cell assignment substitutes each cell in *Lib* by its version in *eLib* maximizing the amount of lateral connections and buried pins.

The entire cell assignment flow is shown in Algorithm 2. The input is a placement for all the cells and the output is an equivalent placement in which some of the cells from *Lib* have been substituted by one of their layouts with lateral pins in *eLib*. For each row, *lateral_connections* applies

the dynamic programming technique explained below to obtain the set of lateral connections that maximizes the number of buried pins. For each lateral connection, the original cells in P and the proposed layouts with lateral pins are obtained via *original_cells* and *enhanced_cells*. The *substitution* finally changes them in the placement. As the mapping to lateral pins has already been decided, the routing stages that take place after cell assignment use *eLib*.

Algorithm 2 Cell assignment stage.

```
1: Input: Placement P
2: Output: Placement P'
3: P' ← P
4: R ← partition_in_rows(P)
5: for each standard cell row r ∈ R do
6:    LC ← maximize_lateral_connections(R)
7:    for each lateral connection lc ∈ LC do
8:       Lib_cells ← original_cells(lc)
9:       eLib_cells ← enhanced_cells(lc)
10:      substitution(P', Lib_cells, eLib_cells)
```

A Dynamic Programming Solution

The goal of the function *maximize_lateral_connections* is to determine the lateral connection to be used between each pair of cells of a row, $\{c_1,...,c_n\}$, while maximizing the number of buried pins. The problem can be solved for each row independently, given that there are no incompatibilities among lateral connections between different rows. Interestingly, the problem for cells $\{c_1,...,c_{i-1}\}$ is nested in the problem for $\{c_1,...,c_i\}$, and thus solving the problem until c_{i-1} is helpful to solve it for c_i. We propose a dynamic programming approach exploiting the optimal substructure of the problem.

The following nomenclature is illustrated in Fig. 3.12. Assume a set of the best solutions has been computed for the darker cells $\{c_1,...,c_{i-1}\}$, and now these solutions must be extended up to c_i. Let k_i be the number of possible lateral connections between c_i and c_{i+1}. At each step between two cells, either one among k_i possible lateral connections is fixed, or none is. Given $i \in \{1..n\}$ and $j \in \{0..k_i\}$, let $buried\_pins(i,j)$ be the number of pins the local connection will bury. It takes value 0 for $j = 0$, representing that no lateral connection is taken, and either 1 or 2 for any other valid connection $1 \leq j \leq k_i$. Let $compatibles(i,j)$ be all the indices of connections

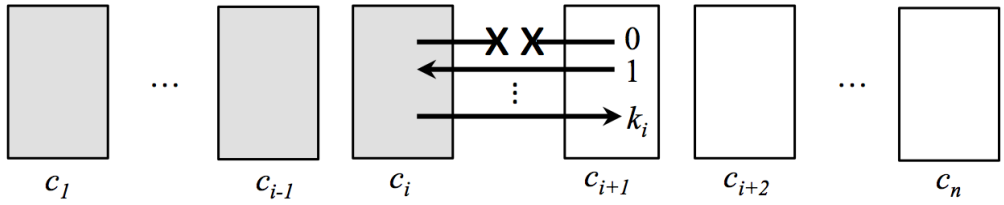


Figure 3.12: Dynamic programming nomenclature.

from c_{i-i} to c_i such that they are compatible with connection j from c_i to c_{i+1}. The function f that counts the optimal number of buried pins from c_1 to c_i for a given connection from c_i to c_{i+1} (indexed for $1 \leq i \leq n$ and $0 \leq j \leq k_i$) is recursively defined by

$$f(i,j) = \begin{cases} buried(i,j) & \text{if } i = 1, \\ \max\{\, f(i-1,j') + buried(i,j) \mid & \\ \qquad j' \in compatibles(i,j) \,\} & \text{otherwise.} \end{cases}$$

The case $i = 1$ represents the first cell, which has no lateral connection to its left side. In the general case, the optimal number of pins buried up to cell i using connection j depends on the pins j can bury at cell c_i and the best solutions of compatible lateral connections in the previous step. The value $f(n,0)$ provides the optimal lateral connection assignment that maximizes the number of lateral pins for the whole row of standard cells.

Using a bottom-up approach, the algorithm sweeps the standard cell row from left to right and at each cell c_i the best solution for any lateral connection that can be taken to the right is kept. When the best solution for the final cell c_n is fixed, it suffices to trace back the cell row to fix the solution for each of the lateral connections. The complexity is $O(kn)$, where n is the number of cells and $k = \max\{k_i \mid i = 1..n\}$. As k is low, the algorithm is essentially linear in time and space with respect to the number of cells in the row.

3.6 Experimental Results

We carried out the experiments using the 6 largest circuits of the itc99 benchmark [36], with a period of about 2ns. They were placed and routed with Synopsys IC Compiler [35] using the Nangate 45nm standard cell library [61]. The three synthesis flows that were used are depicted in Fig. 3.13:

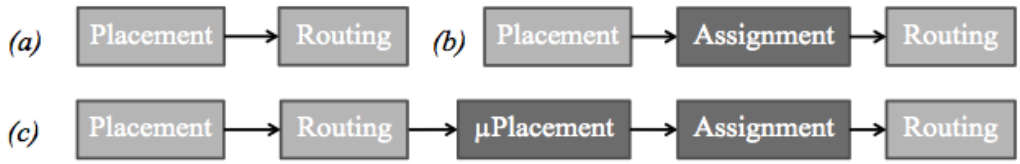


Figure 3.13: Synthesis flows analyzed in the experiments.

(*a*) Original: Placement and routing with DRC reduction. Gives the base number of DRC violations.

(*b*) Assignment: Placement, cell mapping into *eLib* and routing of the unburied nets with DRC reduction. This flow exploits lateral pins and shows the basic benefits of the approach.

(*c*) μPlacement: μPlacement is added before assignment to maximize the number of lateral connections.

The original Nangate library was used for physical synthesis (placement and routing) under the assumption that the *eLib* instances with lateral pins were available with identical timing characteristics. This assumption was made to avoid the complete timing characterization of *eLib*, which is an arduous task out of scope of this thesis and without which the validity of our results is interesting in its own right. We believe this is a conservative approach, since timing with lateral pins could slightly improve. The reason is that routing of the I/O pins to the upper metal levels could be avoided in many cases, freeing up connection resources leading to potentially reduced timing.

Table 3.2 shows the reported number of DRC violations and variations on worst negative slack (in percentage of circuit period, positive means better WNS), vias and pins across the selected circuits obtained after using the three routing flows. The different area utilization and metal layers (indicated using M under the circuit name) are chosen to show the point at which a circuit begins to present DRC violations. Approximate cell count is also shown under the benchmark's name.

Applying the assignment synthesis flow obtains fewer DRC violations than the original flow in almost all circuits. The results show that the timing of the circuit can be preserved when using these techniques. Reduction for vias and pins is obtained in all cases. When comparing the two flows using lateral pins, μPlacement tends to perform better on close to all of the circuits in terms of DRC violations. Timing is improved except in one case (-1.0% in b19). Additionally, μPlacement achieves a

Table 3.2: DRC violations and congestion.

		DRC violations			ΔWNS	ΔVias	ΔPins
	Util.	65%	70%	75%	70%		
b17	Orig.	53	21	140			
M4	Assign.	26	10	37	–0.6%	–4.5%	–7.0%
18k	μPla.	4	0	50	+4.9%	–2.5%	–7.7%
b18	Orig.	105	53	184			
M5	Assign.	129	49	179	–1.1%	–4.9%	–7.2%
36k	μPla.	34	31	132	+2.1%	–4.9%	–10.3%
b19	Orig.	43	79	263			
M5	Assign.	19	53	180	–1.5%	–5.5%	–7.6%
79k	μPla.	12	59	245	–1.0%	–8.0%	–15.7%
b20	Orig.	0	42	147			
M4	Assign.	0	34	105	–0.7%	–6.4%	–8.7%
18k	μPla.	5	0	80	+1.2%	–8.1%	–11.9%
b21	Orig.	44	109	137			
M4	Assign.	5	15	47	–0.4%	–6.3%	–8.7%
20k	μPla.	2	0	23	+0.5%	–9.9%	–13.0%
b22	Orig.	58	107	82			
M4	Assign.	16	49	46	–3.5%	–5.3%	–8.1%
12k	μPla.	0	10	46	+0.1%	–5.6%	–10.4%

Table 3.3: Summary of results (average).

Flow	# lat. wires	Buried 2-pin nets	ΔPins	ΔVias	ΔWL
Assignment	1×	18.1%	–7.7%	–5.4%	–1.1%
μPlacement	2.1×	33.1%	–11.5%	–6.7%	+0.3%

better reduction on vias in almost all cases and performs specially well with I/O pins, achieving a reduction of up to a 15.7%.

Table 3.3 shows more statistics comparing lateral pin metrics between the assignment and μPlacement flows. # lat. wires reports the number of adjacent cells that can be connected using lateral pins. Buried 2-pin nets reports the percentage of 2-pin nets that are connected using lateral pins. ΔPins reports the percentage of cell I/O pins that are removed when using their lateral pin counterpart. ΔVias reports the reduction of vias when the circuit is routed with lateral pins. ΔWL represents the wirelength variation.

One immediate result of the μPlacement stage is the increase on the number of lateral connections, which roughly doubles in all of the cases. However, the wirelength increases slightly after using the μPlacement flow, canceling part of its benefits and leading to examples in which the final DRC violation reduction is lower than when only assignment is used.

It is important to note that the experiments were obtained on a 45nm technology. The impact in buried pins would probably be more noticeable in modern technology nodes, where the different scaling between transistors and wires has increased the criticality of cell pins. At these nodes, the direct reduction on the number of pins would have a heavier impact on routing congestion and could overcome the placement degradation induced by the μPlacement stage.

3.7 Conclusions

This chapter presents under-the-cell routing, a novel and effective way of exploiting internal routing resources and routing adjacent standard cells using lateral pins. The systematic extension of a standard cell library to support lateral pins has been discussed in detail. Algorithms that apply and maximize the lateral connections are proposed and integrated in an industrial circuit design flow. The experiments show the viability of the approach and its potential to improve manufacturability by obtain-

ing fewer design rule violations with negligible timing degradation. The presented techniques require the collaboration of cell library providers to enhance libraries with lateral pins and EDA companies to implement assignment and μPlacement in the circuit synthesis tools. Future lines of work include the refinement of μPlacement algorithms to maximize the number of lateral connections while preserving all quality metrics of the circuit.

Chapter 4

Modern Macro Placement: Theory and Practice

The macro placement problem lies at the juncture between the floorplan and placement problems. Floorplanning is classically considered as the problem of determining layout topology in terms of relative positions of modules on chip, considering area and wirelength estimations, whereas placement consists of deciding the coordinates of a set of components with fixed sizes and shapes. In turn, they are usually sequential steps: first floorplanning, later placement.

In modern physical design flows, the partitioning of a chip in blocks is generally decided manually, and the internal physical design of each block is done separately. Macro placement is then the first step to be solved, and is critical to close timing with no DRC violations. The final coordinates of all elements is needed, but the location of macros in particular has a very powerful impact on the timing closure of modern designs. Given their importance, an array of techniques from the floorplanning and placement algorithm tool box have been used over the years to find the best ways to automate their placement.

This chapter aims to provide an overview on floorplanning foundations, latest macro placement academic advances (including packing-tree based placers), and its practice in the industrial flow to contextualize the work presented in Chapters 5 and 6. These chapters propose methods for macro placement that build on the presented material and rely on the floorplanning of hierarchical blocks. For more information on introductory floorplaning and placement, please refer to [9] [4]. For a detailed view on recent development on physical design automation, see [10].

4.1 Floorplanning Foundations

This section introduces the basic notation and ideas behind slicing-tree simulated annealing (introduced by Wong-Liu [85]), as well as how they relate with our proposed macro placement approach. In the classic floorplanning problem we are given a set of modules with a set of possible shapes for each, and are asked to find a distribution of the modules that minimizes the area of their bounding box, distances between certain modules or other properties.

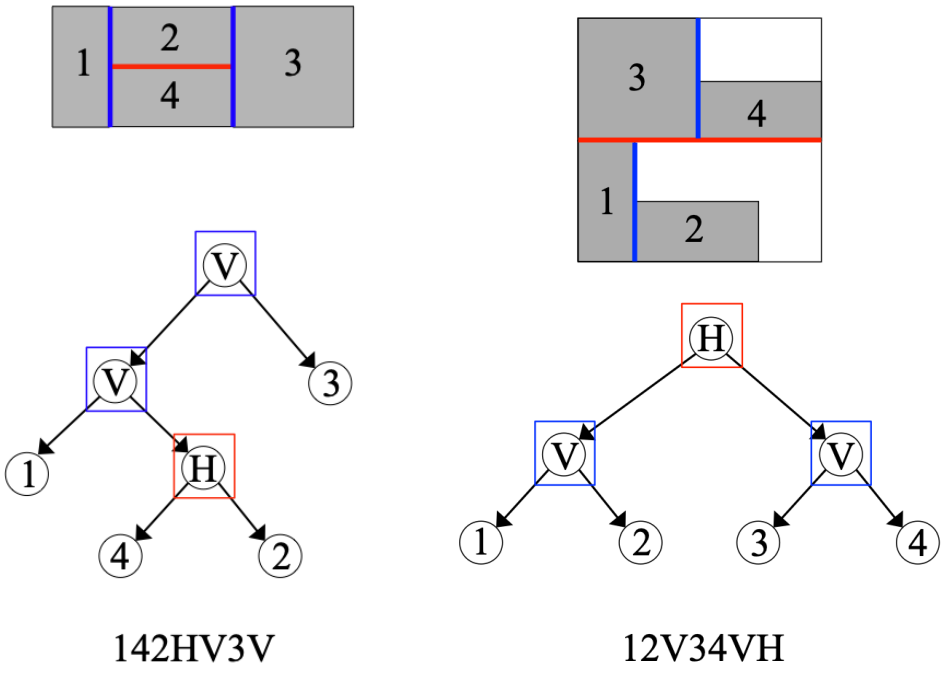
4.1.1 Slicing Structures

An important family of floorplans are *slicing floorplans*, which can be obtained by recursively cutting a rectangle using vertical and horizontal lines. Such floorplans can be represented using a *slicing tree*, which is a rooted binary tree in which leaves represent modules and internal nodes represent the relative location of the modules in the slicing floorplan. Given a slicing tree, the corresponding *Polish expression* is obtained by traversing the tree in post-order.

Fig. 4.1a illustrates a slicing tree, along with its generated layout and polish expression. Modules 1 to 4 have sizes 1x2, 2x1, 2x2 and 2x1 respectively. At each operand node of the slicing tree, the corresponding layout is the composition of the layouts of its children according to the corresponding operator (V for vertical line, H for horizontal line). The horizontal composition H of modules A and B, with sizes A_x, A_y and B_x, B_y, is a module with sizes $A_x + B_x$, $max(A_y, B_y)$, and a similar reasoning applies for the vertical composition. The final layout corresponds to the root of the tree. Fig. 4.1b shows another possible slicing tree for our set of modules which, when converted to a layout, results in a less compact result.

Slicing structures allow for efficient exploration via the simulated annealing methods working on the polish notation (as presented later) at the cost of the loss of some solution flexibility, as not all floorplans can be represented as a slicing structure. However, it has been shown they can produce tight module packings when modules are allowed shape flexibility [93], and represent non-slicing placements by adding a simple compaction procedure [47]. Slicing structures can consider:

Boundary constraints [95] Having particular modules being close to some boundary of the block.



(a) (b)

Figure 4.1: Slicing floorplan, slicing tree and polish expression examples.

Range and preplaced module constraints [96] [94] Having modules being placed in particular placement regions.

Abutment and clustering constraints [97] [98] Having pairs, or groups of blocks, to be abutted or grouped in the layout.

The work presented in later chapters is based on slicing trees as a means of floorplan representation. It is a natural choice given our approach to recursively divide the available rectangle into blocks depending on their area requirements and penalize those layouts which do not respect minimum rectangle sizes to allocate macro components adequately.

4.1.2 Shape Curves

A shape curve describes the possible shapes of a module, and can also be generalized to a subfloorplan corresponding to the internal node of a slicing tree. In general, it suffices to represent them with a piece-wise linear function containing the corner points of the shape curve: a set of Pareto points where each represents a possible shape for the module, that is, a minimal rectangle such that all module components fit in its area.

The shape curve of a node in the slicing tree can be computed with a generalization of the previously described composition for nodes in the slicing tree. Thus, given a slicing tree and the shape curve of each of the modules at the leaves, the shape curve of its root can be computed in a bottom-up fashion. The algorithm basically procees by, at each internal node, having every pair of points at each shape curve of its children be composed, and keeping only the Pareto optimal points. This naive algorithm can be speed-up, by keeping these Pareto points ordered in a dimension in a procedure akin to the merging of two sorted lists [73] or using other methods [63], resulting in linear complexity.

An example is shown in Fig. 4.2. It presents a slicing tree of three modules, 1 to 3. Their possible shapes are 1x2 or 2x1 for modules 1 and 2, and 2x2 or 1x3 for module 3. Their slicing tree and shape curves or the modules and internal nodes are shown in Fig. 4.2a, and the possible shapes of each leaf and intermediate nodes are also shown in Fig. 4.2b. Each point in the shape curve corresponds to a possible position for the nodes. Green points in the shape curve represent floorplans which are not Pareto optimal: there exists some point which is equal or better in all dimensions and still allows all components to be in its area (for example, point (2,2) dominates (2,3) in the shape curve of 12H).

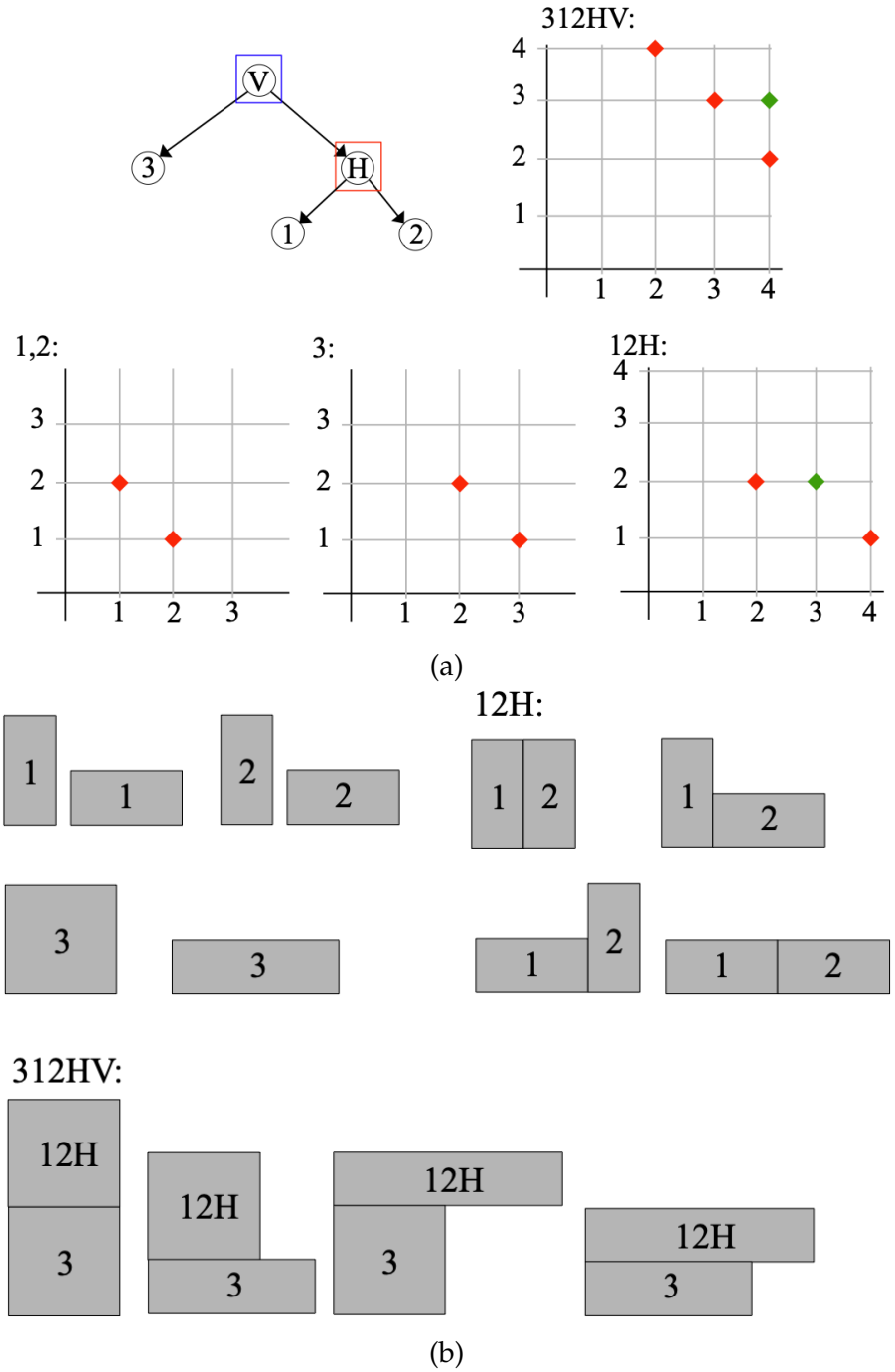


Figure 4.2: Example of shape curves composition.

In conclusion, a shape curve defined by some Pareto points represents the minimum dimensions such that all modules beneath can fit under a slicing tree. Picking a particular shape point in the root, the tree can be traced to obtain the final shape of each module, allowing modules to take different shapes from a closed set depending on the desired shape for the entire circuit. In the work presented in later sections, shape curves are used to store the minimum dimensions of floorplans of given sets of components in modern designs with macros. Because their slicing tree is not be known *a priori*, an optimization process generates slicing trees, and minimum valid shapes are obtained from there. Our use shows the versatility of the structure and how new strategies can be devised using classical floorplan data structures.

4.1.3 Simulated Annealing using Slicing Tree

Simulated annealing [70] is a meta-heuristic that aims to replicate the behaviour of the annealing process. It consists in melting a metal and letting it cool down until solidification. If the process is done slowly, the atomic structure becomes a regular lattice, otherwise it becomes disordered. The basic idea of combining a slicing tree and simulated annealing is to perturb the Polish expression, compute its derived layout and measure its quality. If it is better than before, the perturbation is kept for sure; otherwise it is only kept based on a probability depending on a temperature variable that slowly descends through the execution. The higher the temperature, the more likely a worsening perturbation is kept. The idea of the mechanism is to allow the search to escape local minima while the temperature is high. The final solution would be the global minimum if the cooling process had an infinite amount of time.

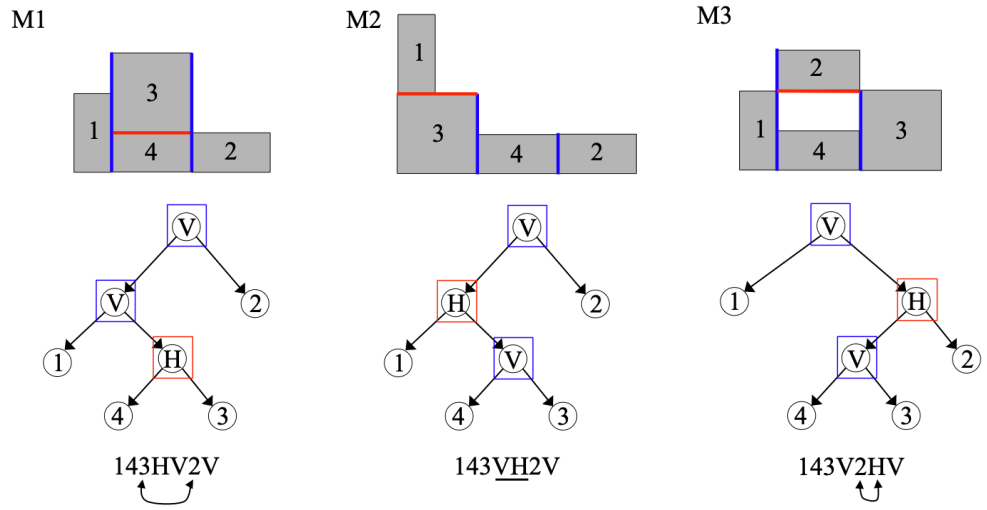
Algorithm 3 presents the method applied to a slicing tree represented in by Polish expression, with the aim of obtaining the layout with the least possible energy. The inputs are execution control parameters governing how fast temperature decays and how many perturbations are tested at each temperature step. First, the Polish expression P receives some initial solution and its energy is computed. From it, an initial temperature is decided and, while the number of rejections among all movements at the current temperature is low and the temperature has not dropped too much, the algorithm executes a round of optimization, with a temperature decrease at the end. Values 100 at line 7 and 0.95 at line 9 represent other configuration parameters that determine the termination conditions of the algorithm and which can be tuned if needed.

Algorithm 3 Simulated annealing using slicing trees

1: **Input:** Movs. per round (k), Temperature decay (ΔT)
2: **Output:** Layout L
3: $P \leftarrow initial\_solution()$ {*Trivial solution or other.*}
4: $L \leftarrow polish\_to\_layout(P)$
5: $E \leftarrow energy(L)$ {*Evaluation. Lower energy = better layout.*}
6: $T \leftarrow init\_temp(E)$
7: $T_{min} \leftarrow temp/100$
8: $reject \leftarrow 0$
9: **while** $reject/k < 0.95$ and $T \leq T_{min}$ **do**
10: **for** $i = 1$ **to** k **do**
11: $P' \leftarrow perturb(P)$ {*See 3 possible perturbations.*}
12: $L' \leftarrow polish\_to\_layout(P')$
13: $E' \leftarrow energy(L)$
14: $\Delta E = E' - E$
15: **if** $\Delta E < 0$ or $random(0,1) < e^{-\Delta E/T}$ **then**
16: $P = P'$
17: **else**
18: $reject \leftarrow reject + 1$
19: $T \leftarrow \Delta T \times T$
20: **return** $polish\_to\_layout(P)$

During each round of optimization (where temperature T is fixed, lines 9 to 19), in a loop which is executed k times, the Polish expression P is perturbed to P', its layout is derived and its energy (or cost) is computed. The functions to go from a Polish expression to a layout and to compute its energy are key to obtain good layouts at the end of the process and account for most of the runtime. In particular, the *energy* function typically measures the packing of the modules in the layout, the wirelength between its components and other optimization metrics. When the new proposed energy is known, if it is lesser than the previous energy or the execution is at high temperature, the current solution P is updated, otherwise the *reject* counter is increased.

The method usually considers picking at random one of three operations to perturb the polish expression. When performing these operations the new layout is evaluated and kept if certain conditions are met. Simulated annealing allows solutions that lead to worse layouts to be accepted because they may bring, after further perturbation, to even better solutions. The perturbations are illustrated in Fig. 4.3, in which the slicing tree in Fig. 4.1a is sequentially perturbed using the following three operations:



(a) Leaf swap (b) Operator inversion (c) Leaf-operator swap

Figure 4.3: Polish expression perturbations.

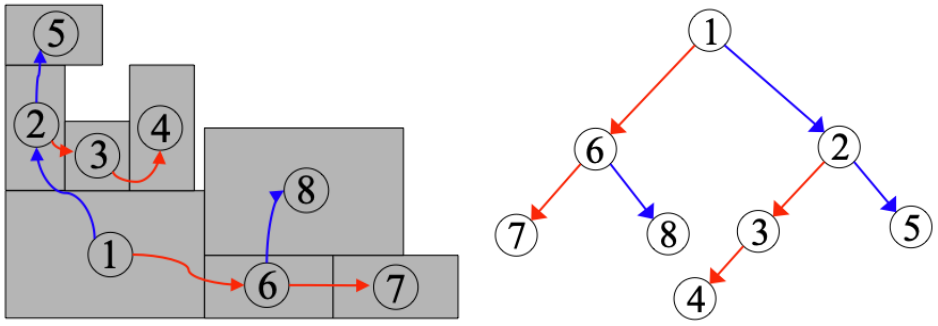
M1 : Swap consecutive leafs in the Polish expression (with possible operands in between).

M2 : Invert a chain of adjacent operators (HVH to VHV, for example).

M3 : Swap an adjacent pair of leaf and operator.

At the end of the process, the layout corresponding to the current Polish expression is returned. Additional mechanisms to keep the best solution can also be added, given that it is not guaranteed that the best visited solution is the one kept by the end. Notice ΔT is a number smaller than 1, typically around 0.9. The higher it is, the longer the search becomes, but better quality is usually obtained as the solution space can be explored with a slower temperature cooldown.

The approach presented in Chapter 5 relies on a simulated annealing optimization process as the one presented in this section. It is innovative in how the blocks and connections between them are modeled, and on how the *polish_to_layout* method allows for blocks with both standard cell and macros to be used to obtain spread macro placements. The function *energy* computes the product of block distances and their dataflow affinity, and also the overlap between various macros, aiming to find floorplans in which related components are close and with no overlaps.



(a) Example layout (b) B*-tree

Figure 4.4: B*-tree and layout example.

4.1.4 Other Floorplan Representations

Besides slicing structures, a myriad of other floorplan representations have been proposed in the literature. One popular class are *packing structures*, a general form of floorplan representation that can model empty spaces. This category includes structures such as the *sequence-pair* (SP) [60], consisting of two permutations of the blocks representing their horizontal and vertical relations. Simple algorithms go from the SP to a layout in quadratic time, while algorithms have also been proposed in $O(n \log \log n)$ [86] [87]. It is the chosen structure in one of the initial papers addressing modern macro placement by combining floorplan and placements techniques (also used on simulated annealing techniques) [2]. Some floorplan representations have been proved to be equivalent to others, for example sequence-pairs with respect to *transitive closure graphs* TCG [37]. Still they are distinguished by their neighborhood structures and operations, critical to finding good floorplans.

Another interesting subfamily of packing structures are *compacted structures*, where modules are compacted to a particular corner of the layout. Their solution spaces are reduced when compared to general packing structures, but their operations can be performed more efficiently. One of its most known representatives, and base for MP-tree floorplanners to be discussed in the next section, is the *B*-tree* [100]. It is an ordered binary tree whose root corresponds to the node at the bottom-left corner. Given a node, its left subree are nodes over it, and right subtree nodes to its right. Given such tree and module sizes, a packing layout can be constructed efficiently. An example is shown in Fig. 4.4. These structures are specially effective for packing, but in macro placement may produce solutions which lack spreading and can induce congestion.

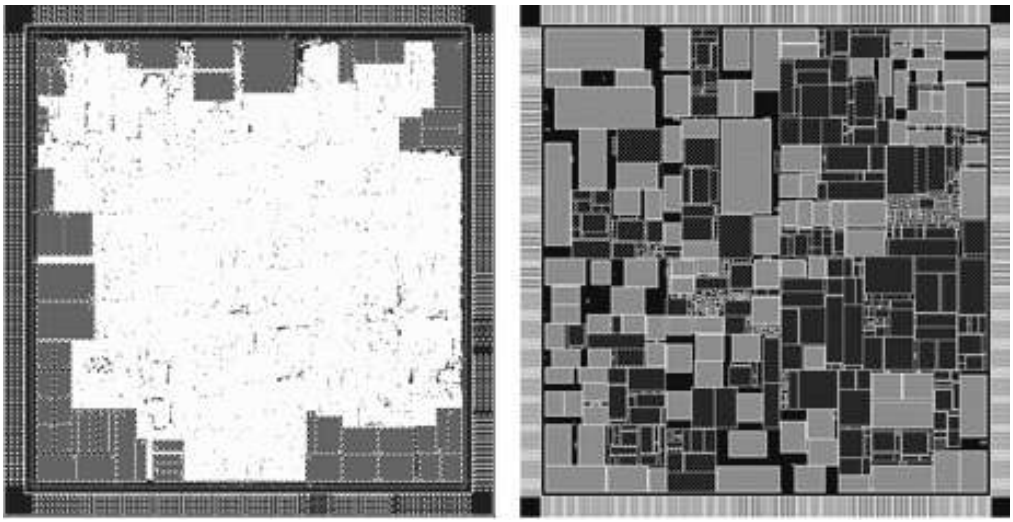


Figure 4.5: Example of "sea of cells" vs. "sea of hard macros". [24]

Most of these representations rely on a simulated annealing process to explore the neighborhood of a solution and find a good floorplan in terms of a given cost function. A huge body of work exists proposing additional constraints to make these structures usable in real instances such as the ones presented in the previous section: boundary, range, abutment... Some structures may offer different options than others, for example rectilinear module constraints for B*-trees, or different sets of perturbation operations, varying in efficiency and result of transformation.

The best structure for an implementation depends on the particular characteristics of the floorplan instance. As hinted before, the macro placement problem uses floorplan structures and techniques, but these can also be applied to other flavors of the floorplanning problem. For example, structures have been proposed for 3D floorplanning in the context of FPGA scheduling, extending sequence-pairs [91], B*-trees [65] and TCGs [99]. Proving the re-usability of such structures, these efforts are being revisited again thanks to recent interest in 3D IC for future technology nodes, further research is ongoing as proved by recent work on corner links and partial order [41]. Other variants of the floorplanning problem consider the use of symmetry constraints for analog designs [95], and the floorplanning of wafers at a manufacturing level [39] [40] [27].

4.2 Modern Macro Placement Automation

Up to the end of the XXth century, the main aim of general floorplanning had been to pack a set of rectangles minimizing area and wirelength between them. The problem would usually be applied to blocks, but not strictly macros: the need to automate their placement had not been there for very long. According to customer data of *Monterey Design Systems*, and confirmed by a user survey over 175 design groups in DAC 2004, the average number of macros per system on chip was under 20 in 2001, and was to reach almost 200 by 2004 [24]. They presented Fig. 4.5 to illustrate how IC would move from being a "sea of cells" with a few macros on the walls to a "sea of macros" with a few areas of standard cells connecting them. The authors already alerted that not enough attention was given to macro placement and about the criticality of the problem. Nowadays, as the number of macros per block follows an increasing trend as they did for the entire chip years ago, the problem becomes increasingly crucial. This section focuses on the advances spurred by this need in the particular field of macro placement and its relations to floorplanning and placement techniques.

When preparing the physical design of a circuit, the current practice is to divide its total area and components into several blocks, which are given to independent designers who work in parallel to close timing (and optimize other metrics such as power). The first step in the physical design of a block is macro placement, deciding the location of macros in the circuit. The step is critical and is usually an iterative manual process in which the designer tries to understand circuit structure and fixes macro locations and orientations. The block is then pushed down the flow and returned to the beginning if its results suggests a macro reorganization is needed.

4.2.1 Macro Placer Taxonomy

Modern macro placement needs were highlighted in a paper by Khang [38], which proposed several limitations of classical floorplanning approaches and noted that floorplanning should be a *fixed-die* (or *fixed-outline*) problem, with a packing which simultaneously achieved *zero whitespace* and *zero overlap* given the fixed die. Much effort has been devoted from that moment to propose floorplan techniques which go in this direction and note the similarity with the placement problem, mostly coalescing around the idea of *mixed-size placement*: the placement of both macro and standard cells. A survey on VLSI placement [57] classifies such algo-

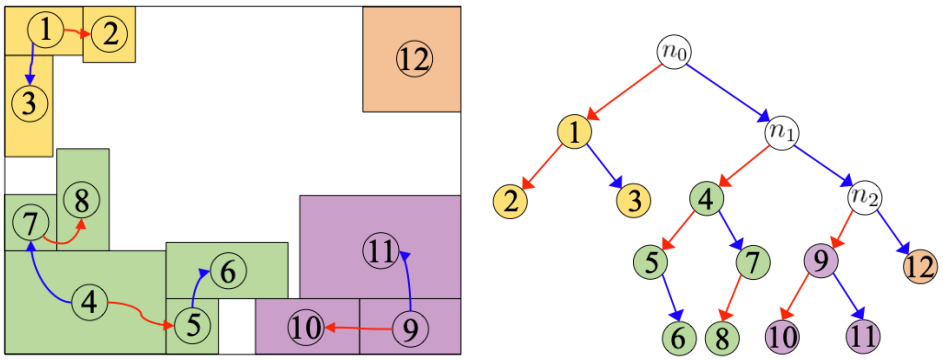
rithms in three groups depending on their order of action: simultaneous cell and macro placement, sequential placement (first macros, then cells), and postprocessing methods which remove overlap between macros and standard cells by floorplan repair.

Simultaneous Macro and Standard Cell Placement.

Under this kind of flows, the placement of macros and standard cells is not separated in different stages, but is done simultaneously. The employed methods are very close to placement techniques, if not placers extended with macro handling capabilities. A key challenge is that components in simultaneous mixed-sized have very different sizes, and they must be fully legalized (no overlaps). To unify component sizes, normally either cells are clustered to bigger components or macros shredded into smaller ones. The latter is the approach followed by [2], [67] and [43], where macros are divided into smaller components with pseudo-nets, and POLAR, a force-directed placer [53] where no extra pseudo-nets are used.

Most of the modern placers are based on analytic techniques. These began with quadratic and force-directed approaches and have become very sophisticated in the later years. Their usual approach is to run analytic placements, which tries to converge to some component coordinates according to analytic models and legalize the layout to remove overlaps. Some placers that consider macros and move them using variants of cell shifting as in FastPlace [83] are NTUPlace [32] and MAPLE [44]. There is also a group of electostatics-based placers [55] [14], where after doing general global placement, particular macro standard cell legalization stages are employed. Other approaches not based on analytic but constructive methods is FLOP, by first doing floorplan and then incremental movements on its results [92].

As highlighted in [92], the optimized functions are usually approximations and the legalization steps which are usually needed in analytical frameworks (and most of the simultaneous flows) to remove overlaps tend to degrade solution quality significantly. Finding new ways to handle both macros and standard cells and solve the problem of finding not only the best locations for components, but also to spread for a good use of routability resources, is still an open challenge. There are some sequential macro placers in which lightweight versions of simultaneous placers are used to propose coordinates that guide the macro placement search. In Chapter 6, a placer based on analytic methods (spectral and forces) with some macro legalization capabilities generates a global view capable of guiding the floorplanner to better macro placements.



(a) Example layout (b) MP-tree

Figure 4.6: MP-tree and layout example.

Sequential Macro and Standard Cell Placement.

In order to allow for a greater flexibility and its modular incorporation in an industrial physical design flow, our method advocates for performing macro placement prior to cell placement. One of the main challenges is that by choosing to place macros first, even though placement prototyping may be used, standard cell locations are not known.

A very prolific family of macro placers considers cell area implicitly by having macros close to circuit corners, which is the *de facto* chosen approach for some industrial floorplanning tools. The method was initially proposed by [12] and relied on packing macros at the corners of the circuit using MP-trees. The structure can be thought of as a tree of packing trees, with each packing tree seating at one corner of the layout. An example is shown in Fig. 4.6, with an example layout to the left and its corresponding MP-tree to the right. Each chidren of the n_i nodes represents a packing tree to one of the corners, and their internal structure is the same as presented in B*-trees. The proposed neighborhood exploration considers moving nodes and swapping nodes or subtrees, among others, and aims to minimize macro placement area, wirelength, displacement to macro coordinates suggested by a placer and overlaps between different packing subtrees in the MP-tree (as they are not avoided by the data structure directly). The resulting layouts look similar to the "sea of cells" of Fig. 4.5.

It was extended by [13] for routability and blockage-awareness. In [16], three major drawbacks to the use of these previous approaches in modern flows (related to overlap avoidance, pre-placed macros, and area utilization) were presented and *CP-trees* were proposed, aiming also to optimize the shape and area of the region for standard cells. However,

the approach lacked scalability, and pre-placed macros must be abuted to chip boundaries.

The most recent advances in this direction propose a multilevel framework for scalability [8], extensions to consider macro and standard cell interaction [11] and domain planning with macro placement [54]. In this last paper, the authors note that although MP-trees and CP-trees can pack macros efficiently, they may result in long wirelength for big designs and present the *MDP-tree*, a divide-and-conquer tree structure in which first the design is divided in "domains" using a slicing structure, and each holds an MP-tree inside (which effectively can be thought as a slicing tree of MP-trees). The approach is scalable and helps reducing wirelength when compared to the previous, but still restricts macro positions to the corners of their given regions, suggesting there would be more room for improvement if such constraint was not enforced by construction.

In contrast to all these packing methods, our approach advocates for directly modeling cells during macro placement to avoid forcing the macros to the walls, where they might be far away from their corresponding standard cells. This allows our layouts to be closer to the "sea of macros" (Fig. 4.5). Another reason to do so is that it is not clear how to model the wirelength between macros without considering possible locations for standard cells. By considering standard cell blocks it is possible to directly model wirelength through abstractions of the netlist, instead of simply relying on hierarchy-based pseudo-nets as do the approaches based on [12]. On the other hand, the blocks with standard cells also take space needed for their placement, instead of forcing them to be in the middle of the design, allowing a less biased and more complete design space exploration.

Although detailed placement of standard cells is out of the scope of this work, floorplanning standard cell groups is key to find satisfying macro placements. Whereas other works consider floorplanning two kinds of blocks, hard for macros with a fixed shape, and soft for standard cell groups with a flexible shape, e.g., using sequence-pairs [2] or slicing trees with shapes curves [92], our approach proposes to model blocks with characteristics of both: having some area requirements, but also knowing some minimal shapes to place macros inside them.

4.2.2 Other Considerations

Besides the general classification of macro placers according to the relation of their macro and standard cell placement phases (if they offer

capabilities for both), other criteria can be used to classify the works depending on the kind of techniques they are using or their intended optimization objectives.

Multilevel, hierarchical approaches.

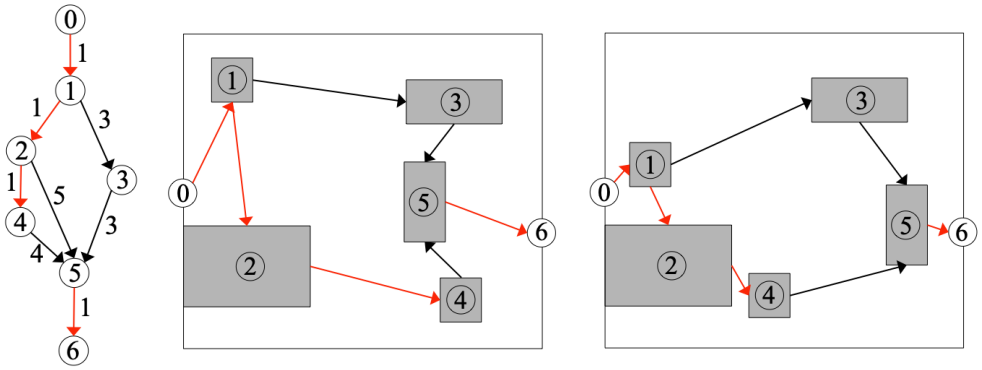
RTL-level information present in the netlist is often used to help obtain better results during physical design. One of the ICCAD 2012 CAD contest problems was on *Design hierarchy aware routability-driven placement* [82]. It has been mentioned that [12] and the deriving approaches use hierarchy information to create pseudo-nets to model the wirelength between macros. It is also used in [18] to obtain better routability, and to guide placement in [31] [30]. More recently, the approach has been used in particular for better component clustering [49] [50].

The natural way in industrial flows to tackle the complexity of working with macros and cells is to use a hierarchical (in a divide-and-conquer sense) and multilevel approach. Some proposals [17] [1] [20] divide the layout in subregions and solve their placements independently, but not explicitly exploiting the hierarchy information of the circuit. The use of multilevel techniques, which follow either a cluster-decluster or partition-merge scheme, have been employed in other contributions [15] [79], and has lately been used to overcome the scalability issues of CP-trees in [8]. Their use is critical for modern design use, where circuits can have millions of cells and hundreds of macros.

Our proposed method combines RTL hierarchy information with multilevel optimization to reduce problem size in macro placement: hierarchy information is considered a clustering of all netlist components. Our process consists on declustering considering area to identify blocks to work with, compute the affinity between such blocks and find a layout for them, and then recursively floorplan each block until macros have been placed.

Optimization objectives.

One of the key questions all macro placers try to answer is: *What makes a good macro placement?* From an industrial point of view, the goal of the physical design process is to close timing without design rule violations. The usual focus is rectangle packing with no overlaps and wirelength minimization. A recent example [26] proposes an algorithm to obtain optimal wirelength. Although it suffers from lack of scalability and produces too tightly packed macro designs, it presents a refreshing direction of work in contrast with the usual heuristic approaches. Under the fixed-die floorplanning paradigm, most macro placers try to primarily minimize some abstraction of wirelength and additional metrics. Many floorplanners aim to optimize several objectives like reducing



(a) Model (b) Wirelength floorplan (c) Pipeline floorplan

Figure 4.7: Pipeline aware floorplanning example.

overlap and displacement from desired positions [12] [54], have continuous cell area [8], improve routability [13] [51] or increase regularity [49]. Some of these approaches minimize complex objective functions in which the weight of components is tuned via user-defined parameters. In order to make the tool useful in general flows and to avoid manual tuning in our optimization function, our work implements an adaptive scheme in the vein of the one proposed in [44] to automatically decide parameter values depending on the current instance execution.

Interconnect estimation.

The usual physical design flow when pipelined connections are present in the block is an iterative process in which the designer tries to work on a floorplan to close timing and, when not possible, the pipelining is modified at the RTL level until a valid floorplan is found. The need to consider pipeline lengths is illustrated in Fig. 4.7. Fig. 4.7a presents a graph where each node is a port (1, 6) or a macro (2 to 5), and edges represent the number of registers in the pipelined connection between such components (red edges are critical). Floorplanners that ignore this information and only use wirelength proxies to guide the search may result in bad solutions from a timing perspective (Fig. 4.7b), as paths that are at short register distance are placed too far away in the layout. Considering register pipelining in the solution cost or as a constraint can lead to better floorplans in this regard (Fig. 4.7c).

Some works in the literature dealt with a similar kind of constraints under the name of microarchitecture-aware floorplanning [62] and illustrate the situation with a simple sentence: "the addition of latencies on some wires can have a large impact on the overall performance while

other wires are relatively insensitive to additional latencies". However, it was applied to processor stage pipelining, and not internal connections between ports and macros in blocks. In these approaches, the location of modules in the layout fix the latencies between them, thus impacting the IPC of the design. In other cases, the location of buses in the chip is particularly modeled [78] [42]. The particular challenge of the pipelined connections problem in macro placements is that these connections must be identified from the netlist.

Another recent work also identifies the need to understand indirect connectivity between macros [50], but not the timing problems induced by register pipelines. Instead, they consider elements that connect to several macros to add to a clustering function, having those macros be clustered and placed closer. The work presented in this thesis is the first macro placement method that considers in-block multi-cycle connections and exploits array information to understand circuit connections and model dataflow with the aim of reducing both circuit wirelength and timing in modern macro placement.

4.3 Macro Placement in the Industrial Flow

This section reviews macro placement from an industrial point of view, why it is critical and what is being done to address it. The presence of such discussion in an academic work such as a PhD thesis is justified given the apparent disconnection between academic approaches and industry practices with regard to the macro placement problem. The topics presented here motivate our research and some of the design choices taken in the design of the macro placement tool HiDaP, with an explicit orientation of being usable in the context of industrial practice.

A crucial step.

Previous sections have shown that macro placement is the first step in the physical design process and has enormous impact on project turnaround time. The placement of these relatively big components determines if any possible placement and routing after it can close timing. The whole physical design flow can take at least a week, and given the iterative nature of the chip design process, reducing the floorplan-to-sign-off iterations, and even avoiding having to go back to review the RTL (which is usually done by another team or company) because pipelines must be adjusted or blocks re-partitioned, could save enormous amounts of time and effort.

At each stage of the physical design flow, a model of the design is built and algorithms try to obtain close to optimal solutions for said model. However, the only result that counts in the end is whether the layouts makes it to sign-off or not. EDA tools tend to be conservative: a floorplan layout that looks good after placement can happen to be very bad later on, but if it is already dubious at floorplan time, it will hardly make it to routing. This motivates us to find good macro placements early, using timing (i.e., register pipeline connections) and area models which are not perfectly accurate but that can make a difference later in the flow.

The role of EDA.

Academic EDA tools are generally focused on generating an automated tool that reduces area and wirelength. However, in the industrial setting, the main goals are timing and DRC/congestion to ensure that the circuit can move to sign-off, besides many other concerns that are usually not directly addressed such as power planning, buffering, clock tree design... All these steps are eased when using commercial EDA tools, which also offer better support for the indispensable manual engineer interaction.

It would be natural to think that the critical process of macro placement has been well tended to by industrial EDA software. And yet, even if usually fast, the support of macro placement algorithms for modern chip designs does not provide good enough solutions. Some of the tools take the approach of placing macros besides walls, but this is unsustainable for designs with tens or hundreds of macros. Experimental results presented in Chapter 5 and Chapter 6 confirm the gap in quality between a particular tool and the quality of results achieved by both our approach and designs obtained manually by expert back-end engineers.

As the number of macros per block increases, enhanced macro placement and circuit analysis capabilities become a necessity, not a luxury. Pioneering companies such as MAXEDA [58] are returning the focus to the floorplanning and placement of macro instances, but it is still not general practice across the industry. To bring a circuit from RTL to working silicon within a given time budget, it is essential to understand the floorplanning implications of the RTL code early by using a new generation of tools that can floorplan modern designs with many macros and provide insight on the connectivity and data flow of circuit components before it is too late. If fast and reliable macro placers were available, they could be use extensively in combination with macro selection techniques for architectural prototyping.

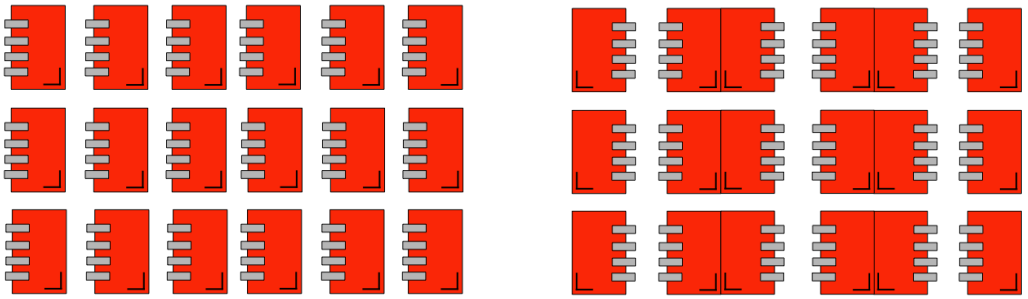
4.3.1 Some Rules for Macro Placement

In general, doing manual designs is the only way to obtain enough solution quality to close timing with no design rule violations. The process becomes an artisanal endavour in which engineers read RTL code (if available) and explore the netlist and hierarchy through the EDA tool GUIs to try to make sense of the connectivity of the components. In order to understand how this is done, the following is a set of basic rules of thumb which were spontaneously provided by an expert physical designer.

1. Go by flyline connections, which translate to wirelength, which needs to be kept at minimum.
2. Check congestion to remove hotspots. Some people use incremental placement iterations to fix those issues, but it's bad as more cells are added and timing constraints may become harder to meet.
3. Place macros from the same hierarchy together.
4. Put memories against block sides and place std cell logic in the middle (but this is only a "soft" rule). Avoid macros blocking dense IO-pins at the boundaries.
5. Abut very small macros together and leave channels for big macros. There is no magic channel width number for all the technologies.
6. If there are many macros, or even not so many, the ideal standard cell region shape is a circle.
7. Place macros back-to-back, with pins facing each other (may or may not be helpful).

They represent a basic set of recommendations, and of course much deeper analysis and understanding of the block is needed in each particular instance. Several rules (4, 7) explicitly mention they will not always help, and is in discerning the different use cases where the true expertise makes a difference. Let us take a slow look and compare them to the current algorithmic state of the art and the work presented in this thesis.

Rules 1 and 2 refer to the optimization objectives at the stage, which mostly consists on reducing wirelength and congestion hotspots. Timing estimates at the floorplan stage are not very reliable, given the low quality of the cell placement at the stage. However, some basic idea of where high cell density is can be quickly obtained. Wirelength remains



(a) Without back to back (b) With back to back

Figure 4.8: Back to back macro placement strategy.

the basic optimization metric, as has been in most modern macro placement approaches, but congestion was not considered an objective until recent years [13] [11]. Some effort has been devoted to doing routability prediction using machine learning [88] to guide an automated macro placer [33], showing growing interest in addressing the issue.

Rule 3 and 4 refer to the global placement of macros. The first insight relates to the use of hierarchical information to guide the placement. As discussed before, this was not much used before the 2012 ICCAD contest on hierarchy-aware placement [82], but is one of the basic guiding principles for human designers. The second advises to use a "sea of gates" model, although not in every situation. Sequential macro placers based on MP-tree [12] follow these two rules by estimating hierarchy-based pseudo-nets to represent wirelength between macros and forcing them to pack to module corners. The proposal presented in Chapter 5 uses hierarchy to guide the multilevel process (macros in the same hierarchy are placed in fixed regions), but allows the search to not be restricted to boundary-packing solutions to be able to find the most adequate solution to each particular instance, as simultaneous mixed-size placers can do.

Finally, rules 5 to 7 refer to a more detailed macro placement. They are concerned not so much about macro placement, but about "channel placement": the inverse problem to our own! These deal with local effects of congestion and routability which have been seldom modeled by macro placers. Some experimental quantification of the impact of inter-macro channel variation has been published [46]. The idea of creating big continuous areas for cell placement was addressed in [16], but more in the spirit of rule 3. Rule 7 refers to a macro array placement strategy illustrated in Fig. 4.8: instead of having all macros with the pins facing at the same side (Fig. 4.8a), they are grouped so that sides without pins abut, creating wider channels for zones with ports (Fig. 4.8b). This approach

has received no algorithmic attention, but might in the future considering some approaches are already working with the concept of macro arrays [49].

Although these rules are far from being a comprehensive set, they can be consider as directions for future work. It has been shown that later contributions to academic research have been going into directions pointed by these indications, and probably more effort would be needed to clearly identify new directions and opportunities for novel macro placement algorithms.

4.3.2 Redefining Engineer Interaction

But the role of EDA nowadays does not finish with pure design automation, but also has to enable engineers to keep imagining and realizing layouts which our tools are not (yet?) capable of producing. As the final drive of high-performance chips is optimizing them much as possible, new support tools and approaches will always be needed at the last technology nodes.

From the industrial practice point of view, one of the main roles of the floorplanning stage is diagnostics: not only to find a good macro placement, but ensure the RTL is good and has no issue that will prevent it from being timing-closeable. This involves physically understanding the implications of RTL-level code. As front-end engineers may not be familiar with physical layouts, and back-end engineers do not always have RTL expertise, floorplanning tools are a meeting point that should make this process as simple as possible through easy to understand tools and abstractions.

These needs call for novel abstractions to capture the main physical objects and their relations without the need of having a floorplan, so that front-end engineers understand the implications of their code before it moves down to physical design, and back-end engineers understand the structure and intent of the code without fighting against sometimes obscure, undocumented netlists. Interestingly, some effort in this direction has been devoted in the hardware reverse engineering field, although they have different assumptions [48] [74] [5]. However, placement-centric variations of their techniques may be of use to the problem in the future.

It is with this idea of obtaining high-level representations of a given netlist that the dataflow graph (presented in Chapter 5) is conceived. The motivation to generate the dataflow graph is to automatically obtain an easy-to-grasp representation of a circuit that can either be used to guide

our automated placement tool or to help engineers in understand their designs, enabling a new abstraction at the RTL-floorplan boundary for fast future bottleneck diagnosis.

Moreover, there is a need to have better user-interfaces that can work at higher levels of abstraction, switching from netlist, to graph, to layout, so that the implications of each to each other can be easily seen and explored. Although out of the scope of academic study, some contributions were made in this direction during the elaboration of the project. They are described briefly in Sect. 5.A.

4.4 Conclusions

This chapter provides an introduction to floorplanning through slicing structures, shape curves and other basic notions, which form the basis of the work developed in Chapter 5 and Chapter 6. It is followed by a retrospective on modern macro placement automation. It includes a description of the two main mixed-placement flows and its main representatives, and other techniques and ways of modeling the problem, with comments on the motivation of various design choices that have been taken for out approach. Finally, a last section has contextualized macro placement in the industrial flow, highlighting its criticality and needs, current approaches and relation with some of the last academic developments. The next chapters present HiDaP, a macro placement automation approach which has been integrated into an industrial physical design flow and managed to obtain better results similar to those of manual layouts after routing, with the aim of becoming a valuable addition to the tool set available to physical design engineers when facing complex designs in modern technology nodes.

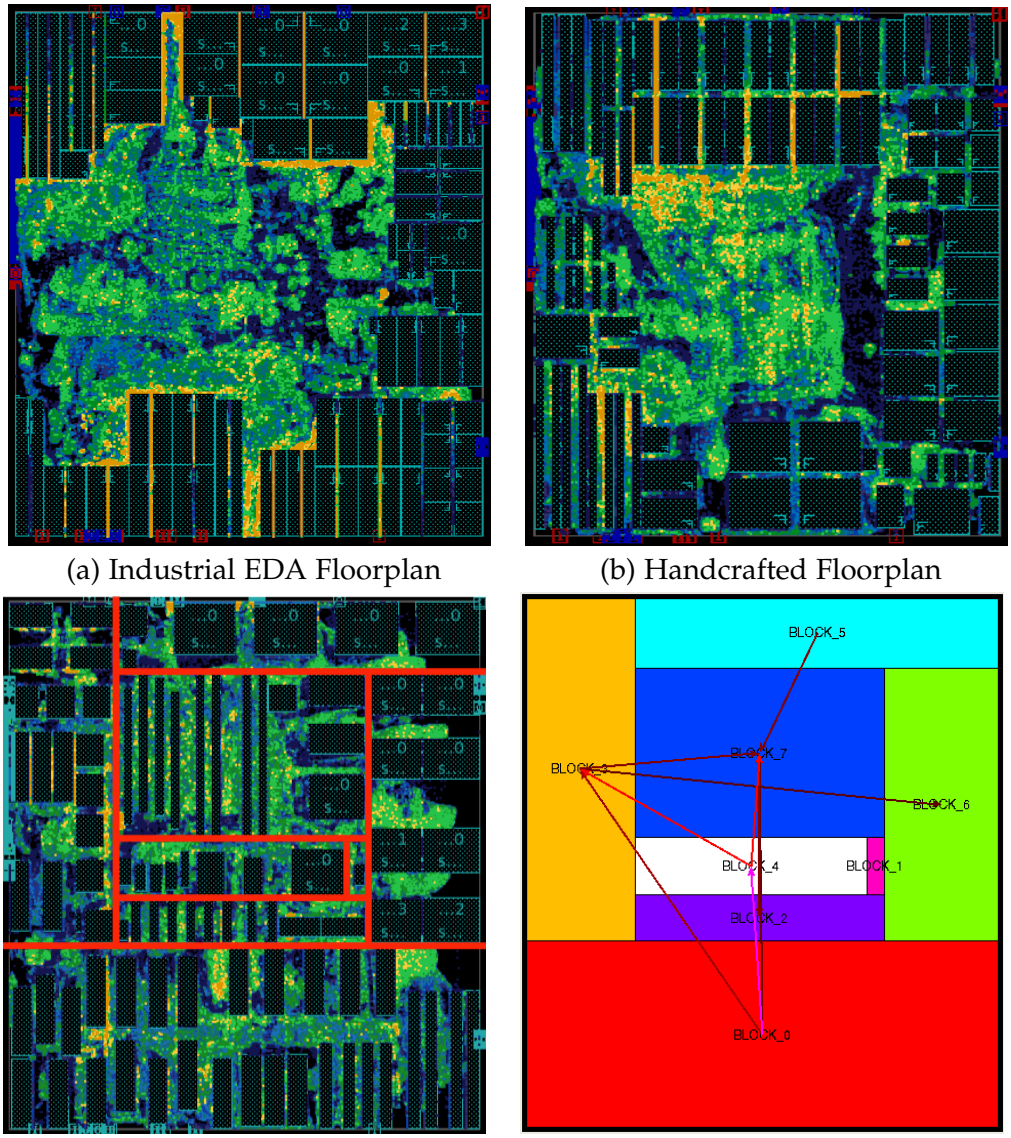
Chapter 5

RTL-Aware Dataflow-Driven Macro Placement

This chapter presents HiDaP, a modern macro placer to help finding better macro placements in an industrial physical design flow. Circuit partitions and their relations in terms of dataflow are derived by using the available RTL-stage information present in the netlist. Images of macro placements obtained using current state of the art techniques (industrial EDA tools and manually handcrafted) and obtained using our approach are shown in Fig. 5.1. Our floorplans are obtained by using a novel multi-level algorithm that starts by placing the most relevant blocks and then proceeds recursively. The top level placement of Fig. 5.1c is shown in Fig. 5.1d. The proposed approach outperforms the commercial tool and produces solutions with similar quality to the best handcrafted floorplans. Therefore, the generated floorplans provide an excellent starting point for the physical design iterations and can contribute to reduce turn-around time significantly. Additionally, Sect. 5.A presents a system to help engineers visualize and understand complex hardware designs.

5.1 Motivation

Industrial EDA floorplanning solutions are fast but often do not produce good enough macro placements. Further iterations by physical designers, which take significant effort, are needed. For circuits with more than 200 macros, it usually takes two to four weeks for the floorplan to reach the desired quality of results. As decisions taken at the floorplan stage have a critical impact on timing and power, considering structural properties during macro placement helps reduce design turn-around time.



(a) Industrial EDA Floorplan (b) Handcrafted Floorplan

(c) Our Approach (d) Block Floorplan of Fig. 5.1c

Figure 5.1: Macro placements obtained using the state of the art methods and our approach.

A key challenge in advanced technology nodes is that connection between components inside a same block can take more than one cycle. This is considered during RTL design, when wires are pipelined (based on RTL designer expertise or previous versions of the design) to meet timing constraints. Awareness of this structure when deciding the location of macros is key to obtain floorplans which can close timing without excessive congestion. The *dataflow* metric proposed in this chapter aims to capture the relation between cell groups, ports and macros by considering the bitwidth and number of stages in their pipeline connections.

The dataflow analysis of a given design is achieved by exploiting RTL-stage information, such as hierarchy and arrays. Many algorithms devote high computational effort to infer structural properties that were known in previous stages of the design and have been destroyed or ignored. The hierarchical abstraction used by humans provides a logic vision of the relations between circuit components. Identifying array components (registers, ports...) is key to understanding the relations between macro components (and the standard cells) and providing floorplans that are more attuned to the structure of the system.

5.2 Contributions

This chapter proposes HiDaP, a multilevel floorplanner to find macro placement and orientation by exploiting RTL-stage structure information. This is done based on the floorplanning of circuit blocks (groups of macros and standard cells), by modeling them and their dataflow analysis relations and using simulated annealing as an optimization process. The aim of our method is to reduce wirelength and timing without macro overlap so that the circuit can reach timing closure without DRC violations with minimal manual intervention.

Multilevel Approach.

HiDaP is based in a multilevel process, as illustrated in Fig. 5.2. It shows a small example deciding the location of the 16 macros of the design from the hierarchy tree shown in Fig. 5.2a (only for macro components). The first level (Fig. 5.2b) identifies and floorplans three key blocks, two containing 8 macros each (dark gray, */A* and */B*), and a third with cells connecting them. The process is repeated for the blocks with macros and each is again partitioned into blocks with macros and cells and blocks with only cells, first */A* (Figs. 5.2c to 5.2e) and then */B* (Figs. 5.2f to 5.2i). At the end of the process, coordinates for each one of the 16 macros are fixed and space has been left for their related standard cells.

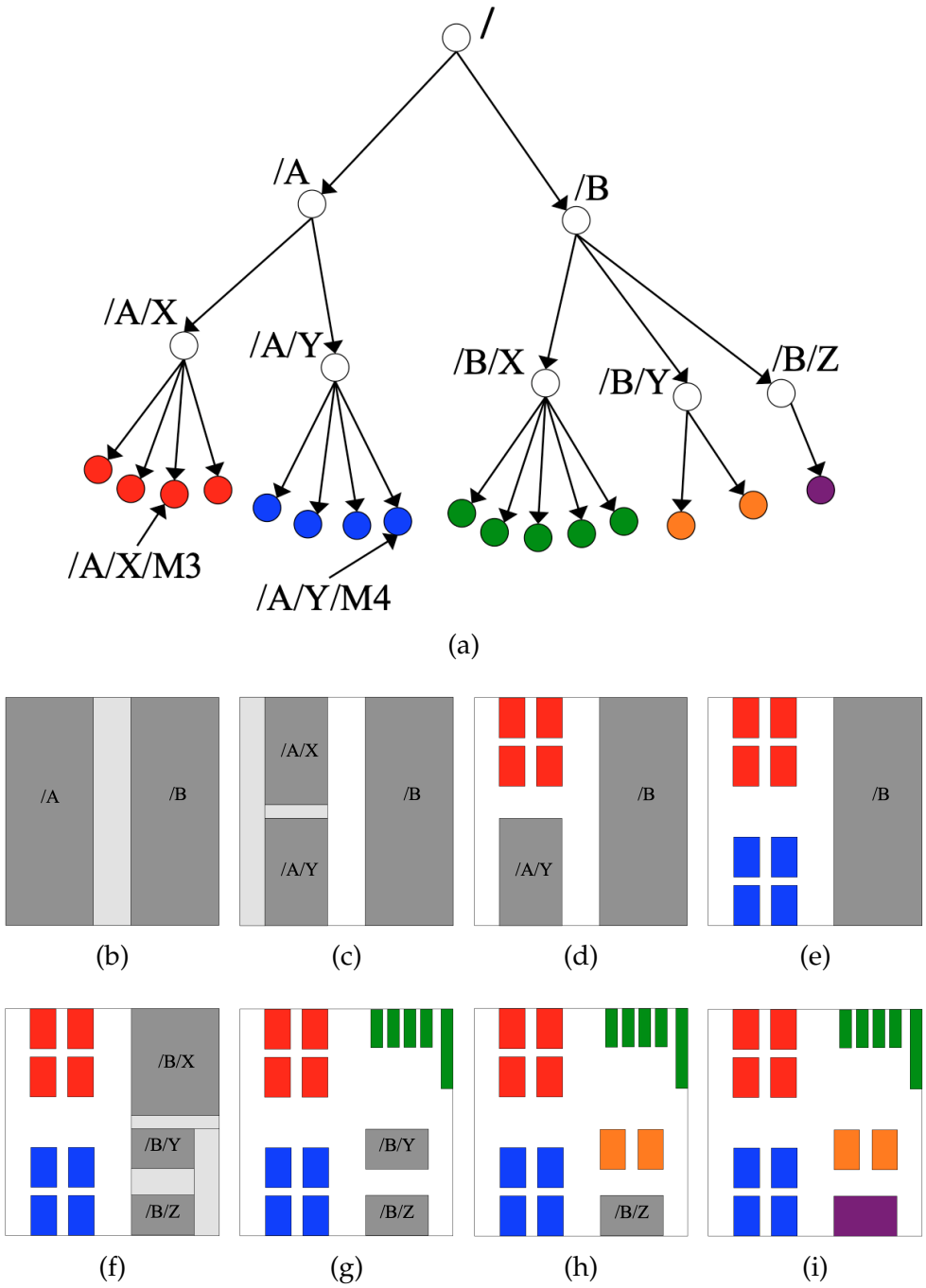
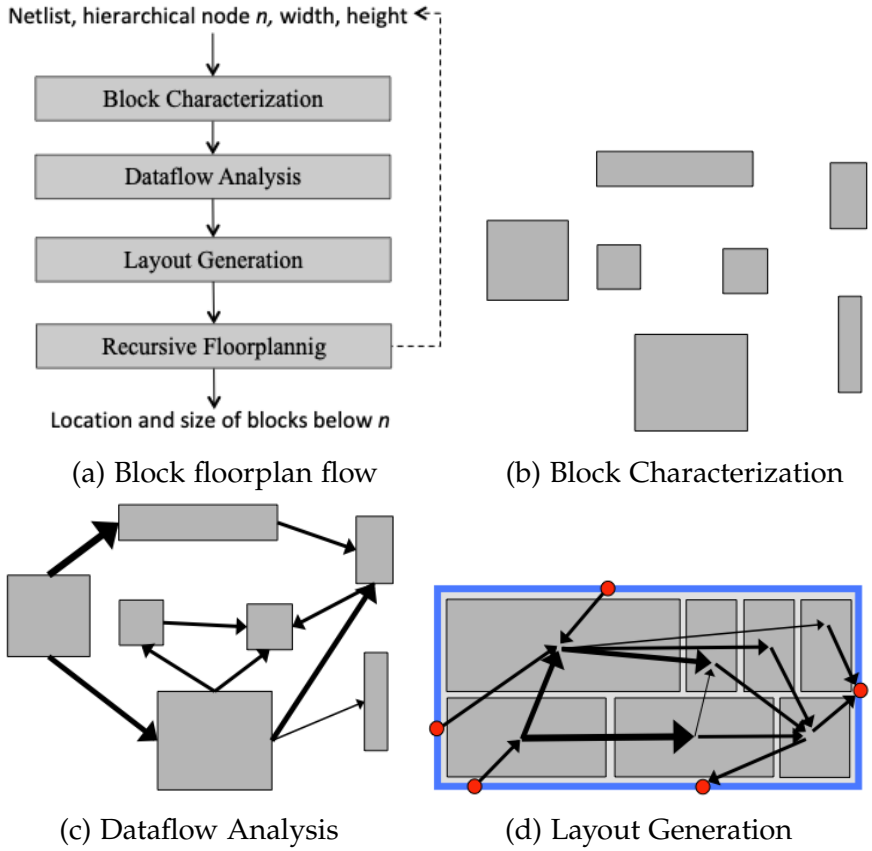


Figure 5.2: Multi-level block floorplan of a 16 macro design.



(a) Block floorplan flow (b) Block Characterization

(c) Dataflow Analysis (d) Layout Generation

Figure 5.3: Macro placement algorithm main stages

Dataflow-Driven Floorplanning Flow.

The main algorithmic steps executed at each floorplanning instance are shown in Fig. 5.3a. The goal is to place the elements below a hierarchical node n with some given width and height constraints. After examining the circuit hierarchy, our tool creates a partition into blocks containing both macros and standard cells groups (Fig. 5.3b) and analyses circuit array information to estimate their dataflow affinity, based on component bit counts and pipeline connection stages (Fig. 5.3c). Finally as seen in Fig. 5.3d, the layout generation step decides the coordinates of each block minimizing dataflow affinity distances. The layout of blocks is decided using slicing structures and simulated annealing, with a novel top-down layout generation algorithm designed to handle *hybrid* blocks with both hard and soft properties, not just hard properties as in most approaches derived from [85]. The approach considers the fixed die area to be a budget and distributes it among the blocks, implicitly dividing whitespace and avoiding overlap when possible.

5.3 Preliminaries

The input of macro placement is a netlist N with hierarchy and array information derived from the RTL stage and fixed area width and height. The outputs are the coordinates and orientation of each macro. The techniques presented in this work explicitly aim to obtain floorplans that show:

1. No overlaps between macro components.
2. Least possible wirelength, worst negative slack and total negative slack.

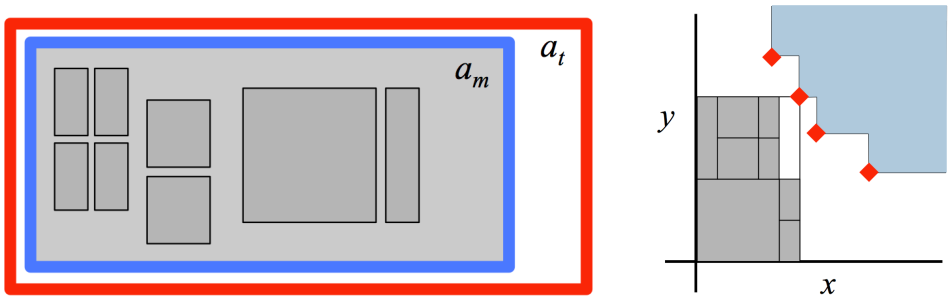
The first item guarantees that the produced layout can be advanced in the physical design, and is considered a priority in our approach. The second includes the primary metrics that define the global quality of the solution. The introduction of dataflow affinity between blocks aims to capture connection bitwidth and flop pipeline latency to minimize wirelenght and obtain good timing. Other metrics such as congestion are also reported in the experiments section but are not directly considered for optimization.

5.3.1 Block Representation

Each *block* represents the standard cells and macros under a node of the hierarchy tree, and has properties of both kinds of components. Given that our hierarchical top-down approach first decides block locations and later the position of the elements inside each block, cell area requirements and the possible slicing floorplans of the macros in a block are key to ensure that sufficient space is allocated in both dimensions when fixing the block area.

Formally, a block is characterized characterized by the triple $\langle \Gamma, a_m, a_t \rangle$. Γ is a shape curve [63] that contains a set of pairs (*width*, *height*) representing the smallest bounding boxes that can hold slicing placements of the macros in the block (standard cells are ignored for Γ). a_m (minimum area) is the sum of the area of all macros and standard cells under the hierarchy level. a_t is the *target area* for the block, i.e., a_m plus some extra area associated to the block (see Sect. 5.5.3).

Fig. 5.4 provides a visual reference of the block representation. Fig. 5.4a shows an abstraction of a block. The darker boxes represent macros. The blue rectangle (a_m) represents the minimum area of the block, whereas



(a) Minimum and target areas. (b) Shape curve (Γ).

Figure 5.4: Area model for a block.

the red rectangle (a_t) represents the target area. Fig. 5.4b shows its shape curve, defined by a set of Pareto points (in red). Given a rectangle for the block of dimensions x_r and y_r, there is a possible placement of our macros in it if the point (x_r, y_r) falls in the blue area. This area represents the space of rectangles that could hold a placement of the eight macros of the block.

Notice that two slicing curves of blocks A, B can be composed vertically to represent minimal slicing floorplans of a block composed of A being vertically adjacent to B (the same applies for the horizontal direction). As explained in Chapter 4, this operation arises naturally in slicing trees where, given the shape curves of its leaves, the shape curves representing floorplans of the macros of its intermediate nodes up to the root can be computed.

5.3.2 Dataflow Affinity

For every pair of blocks A and B, the *dataflow affinity* metric captures the "information flow proximity" between them. This concept is combined with physical distances to provide a layout quality metric. Dataflow affinity increases with the number of bits and flop pipeline stages between them, following the relation

$$Dataflow\_Affinity(A,B) \propto \frac{Information\_Flow(A,B)}{Latency(A,B)}.$$

The motivation is that blocks with a high information flow have high dataflow affinity if their latency is small. Array width information at RTL-stage is used to model information flow (bit count), and the number of sequential elements in paths between the blocks to model latency (number of flops from one block to the other). Blocks with high dataflow affinity need to be close in the layout.

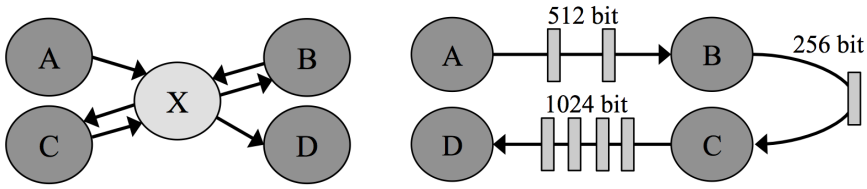


Figure 5.5: Block flow and macro flow for a small system.

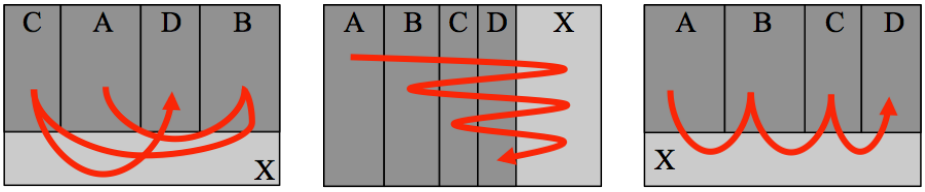


Figure 5.6: Possible layouts for the system in Fig. 5.5.

Two kinds of dataflow between blocks are proposed to better model designs with macros. *Block flow* considers the dataflow affinity between blocks and *macro flow* considers the dataflow affinity between macros inside blocks. Whereas the first models more accurately the physical connections present in the netlist, the second provides a more global insight of the data flow between blocks.

Consider a system with 4 blocks with macros (A to D) communicating through a standard cell block X. A block flow analysis of the netlist would reveal a connection pattern such as Fig. 5.5a, whereas inferring macro flow as in Fig. 5.5b would generate an alternative view on the circuit.

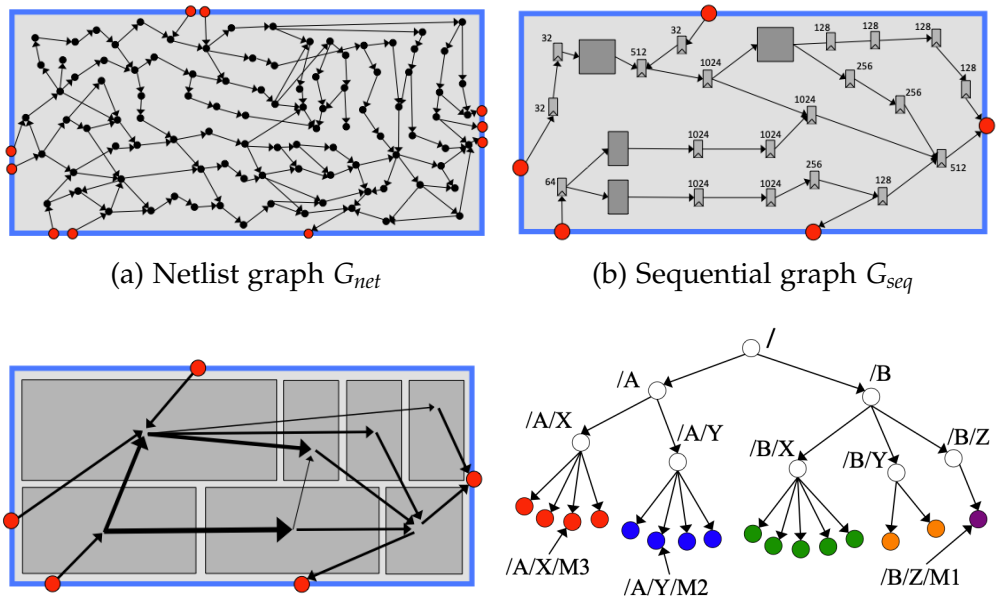
Three layouts for this system, with their communications marked in red, are shown in Fig. 5.6. With only block flow analysis, blocks A to D are close to X (as seen in Fig. 5.6a) without considering their relative positions. Fig. 5.6b shows a layout where only macro flow analysis is used. Its blocks are placed following the macro dataflow, but since X has no macros, it can end up anywhere in the circuit (Fig. 5.6b). Using a combination of both flows generates Fig. 5.6c.

5.3.3 Circuit Abstractions

Three graphs and a tree are used to model circuit connectivity (see Table 5.1 and Fig. 5.7). The netlist graph G_{net} contains the lowest level of abstraction of the circuit (standard cell level). G_{seq} and G_{df} are subsequently derived from G_{net} in order to model other view of circuit consid-

Table 5.1: Data structures for different circuit abstractions.

Graph	Size	Type of vertices	Description
HT	–	Hierarchical blocks.	Circuit hierarchy representation.
G_{net}	$\sim 10^7$	Macros, ports, sequential and combinational cells.	Bit-level complete netlist connectivity.
G_{seq}	$\sim 10^5$	Macros, multi-bit ports and registers.	Multi-bit sequential connectivity.
G_{df}	$\sim 10^2$	Blocks and multi-bit ports.	Dataflow affinity among blocks and ports.



(a) Netlist graph G_{net}

(b) Sequential graph G_{seq}

(c) Dataflow graph G_{df}

(d) Hierarchical Tree HT

Figure 5.7: View of different circuit abstractions.

ering multi-bit sequential elements (medium abstraction, Fig. 5.7b) and hierarchical block information (highest abstraction, Fig. 5.7c). The detail of the obtention of G_{seq} and G_{df} from G_{net} is presented in Sect. 5.5.4. The hierarchical tree HT (Fig. 5.7d, with some sample block and component names) is derived from the component names in the input hierarchical netlist N.

More formally, the graph $G_{net} = (V_{net}, E_{net})$ represents the original netlist, with $V_{net} = M \cup P \cup F \cup C$, respectively being macros, ports, sequential cells (flops) and combinational cells.

The sequential graph $G_{seq} = (V_{seq}, E_{seq})$ is a directed graph with weighted nodes. Its edges capture communication latency and array width between sequential components and ports. V_{seq} represents macros, multi-bit registers and ports.

The dataflow graph $G_{df} = (V_{df}, E_{df})$ is a directed graph with weighted nodes and edges which used to derive dataflow affinity. The node set is $V_{df} = B \cup P_{mb}$. Every node in B represents a block and is formed by a subset of nodes in V_{seq}. The edges in E_{df} hold information on the width and latency of the paths between these subsets of nodes in V_{seq}.

In $HT = (V_{ht}, E_{ht})$, every node represents a level in the hierarchy, and edges represent subhierarchy relations. A hierarchy cut HC with regard to a node $n \in V_{ht}$ is defined as a set of nodes in its subtree such that each path between n and the leaves of its subtree crosses exactly one node in HC.

5.4 Algorithmic Overview

As illustrated in Fig. 5.2, HiDaP considers the hierarchy as a clustering of the design and uses a decluster-and-floorplan scheme: successively undoing part of the clustering and floorplanning until macro objects have been placed. Building on the flow presented in Fig. 5.3a, Algorithm 4 provides a high-level view of the top flow of the tool. The input is the hierarchical netlist N and the dimensions of the circuit. The first step is the generation of the hierarchical tree from the netlist. *shape_curve_generation* constructs the set of shape curves S_Γ, used to ensure that a block layout fixed at a given hierarchical level can accommodate its macros under slicing constraints. It contains, for each $n_h \in V_{ht}$, a shape curve with the minimal sizes such that its macros can be placed. The next steps are finding macro location by calling *recursive_block_floorplan* and the orientation with *memory_flipping*, with a flipping post-process for wirelength reduction based on computing the dataflow affinity between macro corners.

Algorithm 4 Top tool flow

1: **Input:** *hierarchical netlist (N), width (w), height (h)*
2: **Output:** *location and orientation of the macro cells*
3: $HT \leftarrow$ *obtain_hierarchy_tree(N)*
4: $S_\Gamma \leftarrow$ *shape_curves_generation(HT)* {Sect. 5.5.1}
5: *macro_loc* $\leftarrow$ *recursive_block_floorplan(N, w, h, root(HT),* S_Γ*)*
6: *macro_orient* $\leftarrow$ *macro_flipping(N, macro_loc)*

Algorithm 5 Recursive block floorplan

1: **Input:** *hier. netlist (N), width (w), height (h),* $n_h \in V_{ht}$, S_Γ
2: **Output:** *location of the macros below* n_h
3: $B_{\Gamma,a_m} \leftarrow$ *hierarchical_declustering(*n_h, S_Γ*)* {Sect. 5.5.2}
4: $B_{\Gamma,a_m,a_t} \leftarrow$ *target_area_assignment(*B_{Γ,a_m}, *N)* {Sect. 5.5.3}
5: $M_{aff} \leftarrow$ *dataflow_inference(*B_{Γ,a_m,a_t}, *N)* {Sect. 5.5.4}
6: *coords* $\leftarrow$ *fp_gen(w, h,* B_{Γ,a_m,a_t}, M_{aff}, $coords_0$*)* {Sect. 5.5.5}
7: **for all** $b \in B_{\Gamma,a_m,a_t}$ **do**
8: **if** *macro_count(b)* > 1 **then**
9: w_b, $h_b \leftarrow$ *block_size(coords, b)*
10: *recursive_block_floorplan(N,* w_b, h_b, *b,* S_Γ*)* {Recursion}
11: **else** *fix_position(b,* w_b, h_b*)*

The recursive floorplanning function (Algorithm 5) takes as an input the netlist N, a hierarchical node $n_h \in V_{ht}$, whose subtree must be placed in a given space (w, h), and the shape curves for all hierarchical levels in S_Γ. The aim is to place the macros under n_h considering dataflow between themselves and standard cell logic under n_h, but also crucially with macros outside n_h and ports, to ensure a global view of the problem is kept at each floorplanning instance.

The first step is *hierarchical_declustering*, which identifies the blocks to be considered for floorplanning and partially characterizes them in the set B_{Γ,a_m}, with the minimum area a_m and shape curve Γ of each block. In *target_area_assignment*, the target area a_t of the blocks is stored in B_{Γ,a_m,a_t}, a set containing the full characterization $\langle \Gamma, a_m, a_t \rangle$ of each block.

In *dataflow_inference*, the dataflow affinity between each pair of blocks is derived and stored in the affinity matrix M_{aff}. During *fp_gen* (floorplan generation), slicing structures are used to model the floorplan and, given suggested coordinates (from spectral placement or user suggested) and M_{aff}, an optimization process (simulated annealing) is used to minimize three components: the product of the distance between blocks in the layout times their dataflow affinity, macro overlaps and distance to

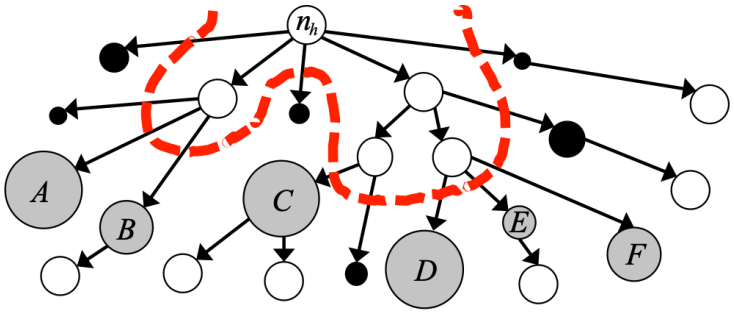


Figure 5.8: Hierarchical declustering to find HC_B and HC_G.

the suggested locations. After the coordinates of each block are fixed in *coords*, the process is recursively called if the block has more than a single macro. Otherwise, its macro position is fixed in the corner of the available area that minimizes wirelength.

5.5 Algorithmic Details

5.5.1 Shape Curves Generation

Our hierarchical flow first decides the area alloted for a given hierarchical sub-block, and then proceeds to find the locations of its macros recursively. It is necessary to ensure at each hierarchical level that enough space is reserved to fit a slicing floorplan of all its macros, or overlaps would be introduced (more detail is presented in Sect. 5.5.5). This initial step computes such feasible slicing floorplans for each hierarhy level and stores them in shape curves to be reused through the rest of the execution.

The input of this step is the hierarchical tree HT, and the output is a set of shapes curves S_Γ where each Γ is associated to a node in V_{ht} describing possible sizes for the floorplan of the macros under its subtree. The subtree shape curves are calculated in a bottom-up fashion up to the root of the hierarchical tree. At the leaf nodes of HT, the associated Γ contains the possible shapes of its macro. At each intermediate node of the hierarchy, it is not possible to compose the shapes of their children (the tree is not slicing). Since there is no general mode to compose the shape curves of a non-binary tree, area-optimization floorplanning using simulated annealing generates a set of shape combinations with small area which are valid for the node.

5.5.2 Hierarchical Declustering

The purpose of hierarchical declustering is to find the set of blocks to be considered during the floorplan of a given hierarchy level n_h, and to characterize their shape curve Γ and minimum area a_m. The idea is represented in Fig. 5.8, which shows the hierarchy tree under a node n. A possible hierarchical cut HC with respect to that node (defined in Sec. 5.3.3), marked by a red line, generates block candidates for floorplanning. The nodes are divided in two sets, depending on the number of macros and area in their subtrees, and those with big area or with macros are selected (grey).

Formally, consider functions *area(n)* and *macro_count(n)*, returning the sum of the area and macro count of the subtree rooted at $n \in V_{ht}$. Given a parameter defining a minimum area, the nodes in any hierarchy tree cut HC can be divided in two sets, HC_B and HC_G:

$$\begin{aligned} HC_B &= \{n \in HC | area(n) > min\_area \vee macro\_count(n) > 0\} \\ HC_G &= HC \setminus HC_B \qquad \text{(small nodes with glue logic)} \end{aligned}$$

Nodes in HC_B represent hierarchy levels with relatively big area or containing macros, whereas all others (HC_G) are small nodes with glue logic. Each node in HC_B is modeled as a block during layout generation. Area of nodes in HC_G is integrated with nodes in HC_B during later steps, as it represents small unstructured components.

Algorithm 6 Hierarchical declustering.

```
1:  Input: n_h ∈ V_ht, open_area, min_area ∈ ℝ
2:  Output: HC_B, HC_G
3:  HC_B ← ∅; HC_G ← ∅
4:  queue.insert(n_h)
5:  while not queue.empty() do
6:      m ← queue.pop_front()
7:      if area(m) > open_area then
8:          for each child c ∈ m.children() do
9:              queue.insert(c)
10:     else if area(m) > min_area or macro_count(m) > 0 then
11:         HC_B ← HC_B ∪ {m}
12:     else
13:         HC_G ← HC_G ∪ {m}
```

The strategy used to find sets HC_B and HC_G for a given node n is shown in Algorithm 6. The parameter *open_area* is an area amount that

controls how deep the tree will be explored. Both *min_area* and *open_area* are a fraction of $area(n_h)$, 40% and 1% in the experiments.

The process is illustrated in Fig. 5.9. The left shows the hierarchical tree, and the right the relative area of the nodes to the left. After processing the root, two nodes have very big area, while the other three are relatively small (Fig. 5.9a). Since the area of the colored nodes is over the defined threshold, they are opened (Fig. 5.9b) until all nodes are smaller than the threshold as seen in Fig. 5.9c (the red line represents the HC of the root). In the end (Fig. 5.9d), the colored nodes are considered during floorplan and are added in HC_B, and all other are incorporated to HC_G.

5.5.3 Target Area Assignment

Since small and unstructured blocks in HC_G are not directly considered for floorplanning, the goal of target area assignment is to incorporate their area to the target area a_t of blocks in HC_B. In G_{net}, a multi-source breadth-first search [77] finds the shortest paths from all nodes in HC_B to any cell in HC_G. Fig. 5.10 shows three blocks in HC_B with fully painted nodes, connected by glue logic components (elements in HC_G blocks) which are incorporated to their closest blocks as they are reached during the search. After this process, the sum of the area of HC_B blocks represents the whole area of the floorplanning instance and the triplet $\langle \Gamma, a_m, a_t \rangle$ for each block is characterized.

5.5.4 Dataflow Inference

Dataflow inference generates an affinity matrix M_{aff} estimating the dataflow affinity between each pair of blocks, ports and other macros in the circuit. M_{aff} is derived from the dataflow graph G_{df}, which is obtained from G_{seq} and G_{net}. First G_{net} is transformed into G_{seq} in the following steps:

1. C nodes representing combinational cells are removed by connecting their predecessors to their successors

2. Nodes in P and F are clustered using component names to find array structures (`name[`n`]`, `name_`n).

3. Edges between sequential components are inferred by analyzing their transitive fanin/fanout in G_{net} and discovering their paths.

4. To reduce graph size but keep relatively big components, elements with fewer bits than a threshold are discarded.

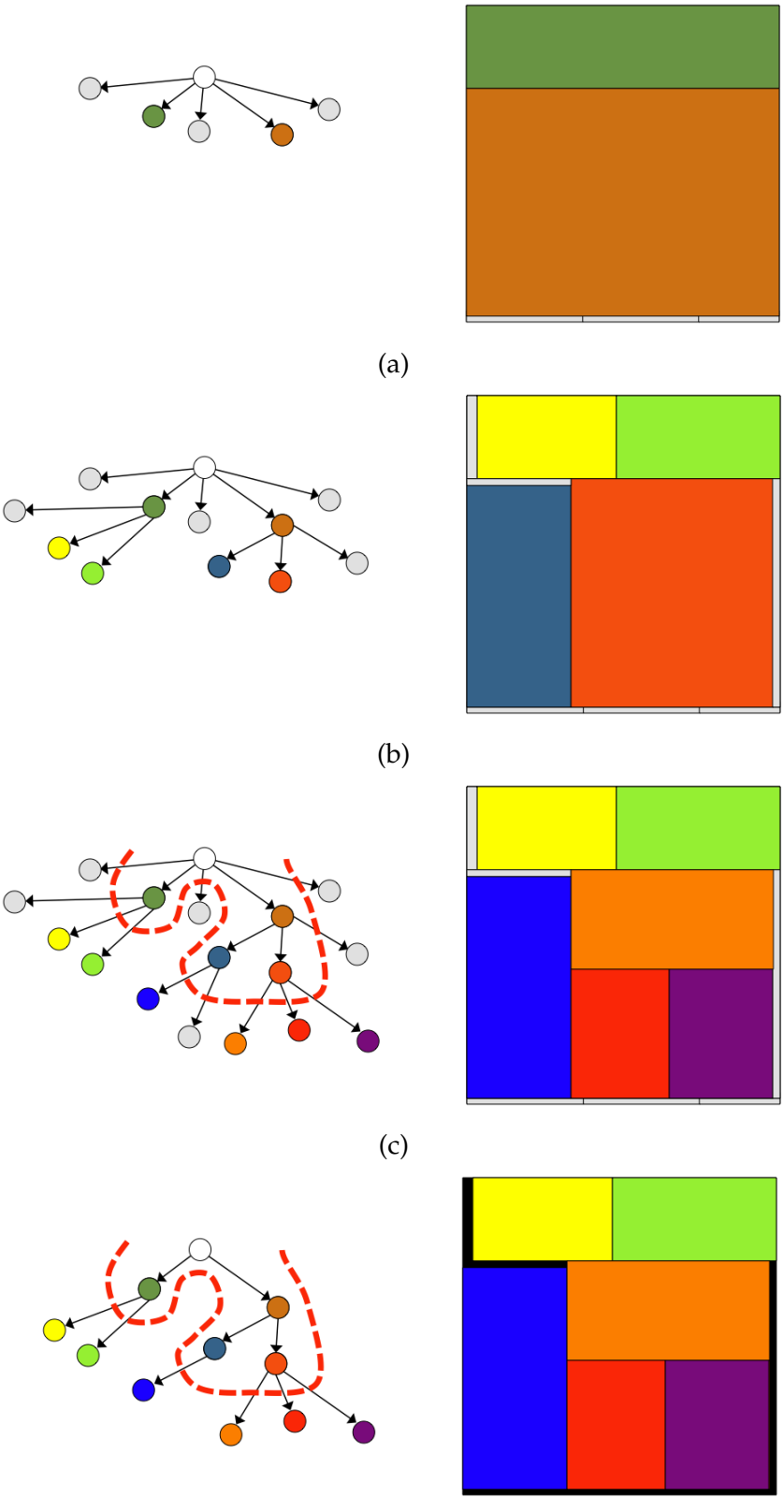


Figure 5.9: Example of the hierarchical declustering process.

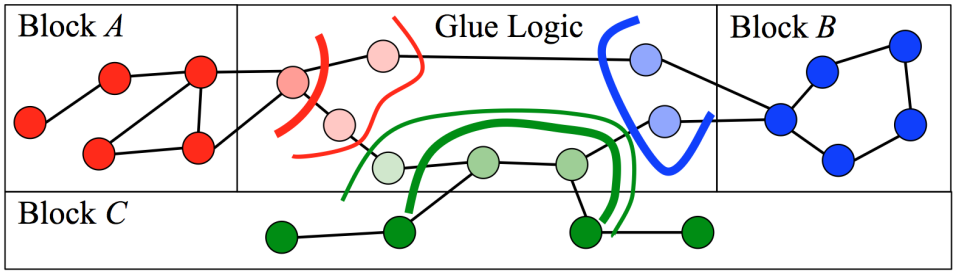
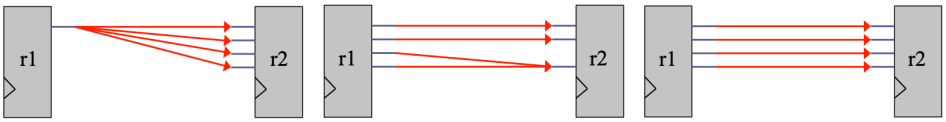


Figure 5.10: Assigning HC_G area to HC_B blocks.

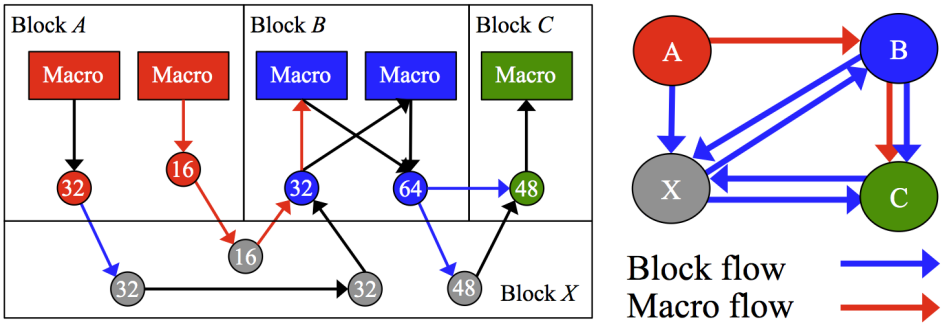


(a) $w_e = 1$ (b) $w_e = 3$ (c) $w_e = 4$

Figure 5.11: Sequential edge weights.

The edge weight between sequential elements in G_{seq} (for example, two registers r_1, r_2) is computed by doing graph traversals in G_{net} and counting the number of input bits in r_2 reachable from output bits in r_1, and output bits in r_1 reachable from input bits in r_2. The minimum value is kept as the weight. Some examples are shown in Fig. 5.11, with the resulting edge weight w_e between the registers. This conservative approach allows us to focus on the main datapath buses and reducing graph size for better tractability when removing edges with low weight.

The next step is constructing the dataflow graph G_{df} from G_{seq}. Each node in V_{df} represents a hierarchical block, which is associated to a set of nodes from V_{seq}. Each edge summarizes two auxiliar edges, one which represents block flow E^b_{df} and one which represents macro flow E^m_{df}. A



(a) G_{seq} graph. (b) G_{df} graph.

Figure 5.12: Dataflow inference example.

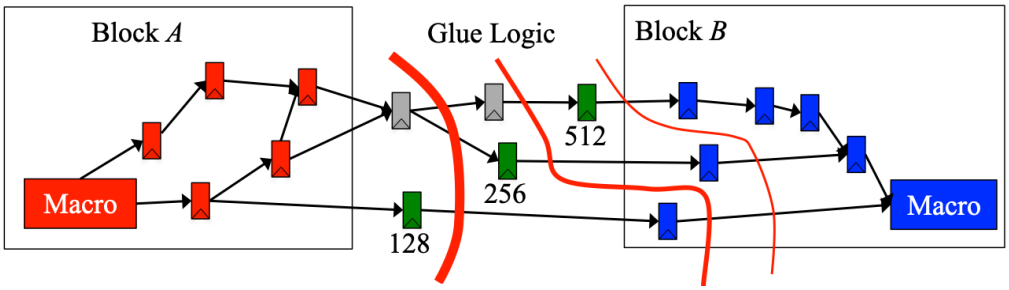


Figure 5.13: Finding flop pipeline stages between two blocks.

graph G_{seq} is shown in Fig 5.12a, where round nodes represent multi-bit registers with their bitwidth. Painted edges represent some paths that generate edges in G_{df}, shown in Fig 5.12b: blue represents block flow E^b_{df} edges, and red represents macro flow E^m_{df} edges.

Connectivity information of edge $e_{i,j}$ in E^b_{df} or E^m_{df} takes the form of a histogram, where bins represent latency and their height represents number of bits. In the case of block flow edges E^b_{df}, a breadth-first search at G_{seq} starts simultaneously from all components of block i traversing only outgoing edges through glue logic. When a component of block j is reached, the bitwidth of its predecessor in the path is added to the bin corresponding to the path length (blue paths in Fig 5.12a). To obtain macro flow edges in E^m_{df}, a similar process finds paths between macros in blocks, allowing the search to cross all nodes in V_{seq} (except macros) instead of only glue logic (red paths in Fig 5.12a). Fig. 5.13 illustrates the computation of the edge weights in G_{df} from a G_{seq}. It depicts two blocks, A and B, with multibit registers and macros, connecting through some glue logic. The red lines represent the register latency limits. The green registers contribute to the dataflow between the blocks A and B with 128 bits at latency 1, 256 at latency 2 and 512 bits at latency 3.

In order to condense the dataflow information of the histogram between two blocks, the weight of their edge is computed using $score(h, k)$ for an integer k according to the following formula, where i indexes bins in the histogram:

$$\sum_i \#bits_i \ / \ flop\_stages_i^k$$

The formula is based in the relation presented in Sec. 5.3.2. Parameter k controls the exponential decay impact of latency. For $k = 2$, the block dataflow between A and B in Fig. 5.13 is 384.

Given the edge sets E^b_{df} and E^m_{df} and scoring their histograms, the score of dataflow affinity edges in E_{df} is obtained using a parametric formula

defined as $\lambda \times score(e^b_{i,j}) + (1 - \lambda) \times score(e^m_{i,j})$. The value of λ balances block and macro flow and allows the generation of layouts with different emphasis on block or macro connectivity.

5.5.5 Layout Generation

The last step in the recursive floorplanning algorithm takes as input the rectangle in which the blocks at a given hierarchical level must be placed. After their characterization in terms of area/macros and their dataflow affinity relation has been computed (Sect. 5.5.2 to Sect. 5.5.4), the goal of layout generation is to find the coordinates for each block in the given area minimizing wirelength and timing, without overlaps and minimizing the distance between macros and their desired locations.

The layout is represented as a slicing tree with a node per block. This structure allows to work with shape curve compositions and naturally use a top-down algorithm for layout generation. The solution space is explored using simulated annealing. The structure is perturbed with equal probability with one of three operations: operand swap, operator inversion or operand-operator swap (similar to [85]).

Objective Function

The objective function to be minimized to optimize wirelength and timing is:

$$\left(\sum_{b_i, b_j} distance(b_i, b_j) \times dataflow\_affinity(b_i, b_j) \right) \times \text{OVERLAP}$$

Minimizing the sum of the products of dataflow affinity and distance ensures that blocks that have large array connections and relatively small latency are close in the layout, thus reducing wirelength. The position of ports and macros outside the subtree are considered a fixed point. The computation of this product consumes most run-time of each simulated annealing iteration, since the number of products is quadratic with respect to G_{df} nodes. The OVERLAP factor represents the sum of overlapping areas in the layout multiplied by a constant that increases with the severity of each illegality. This mechanism is used to ensure macro overlaps penalize more than standard cell area overlaps.

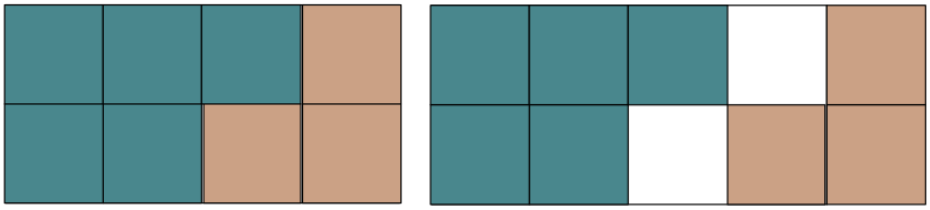
Layout Representation and Search

This section describes the algorithm to decide the coordinates and sizes of each block from a given slicing tree. There is a well-known method to obtain a layout from a slicing tree which consists of composing, at each node, the shape curves of its two children according to its operator. It has been widely studied in the context of non-fixed outline floorplans (when area minimization is one of the objectives for the formulation). For fixed-outline floorplanning, the approach presents two limitations:

- If the sum of the whole floorplan is divided among all placeable components, it is hard to find a valid floorplan that fits exactly in the alloted size. Extra legalization steps are necessary to push elements back into the floorplanning region. If not all available area is used, an extra spreading step might be needed to redistribute white space.
- Module shapes must be discretized and decided a priori. Blocks with standard cells must choose from a set of aspect ratios to be composed up to the root of the slicing tree. Too few result in bad solution space exploration, too many increase computation time greatly.

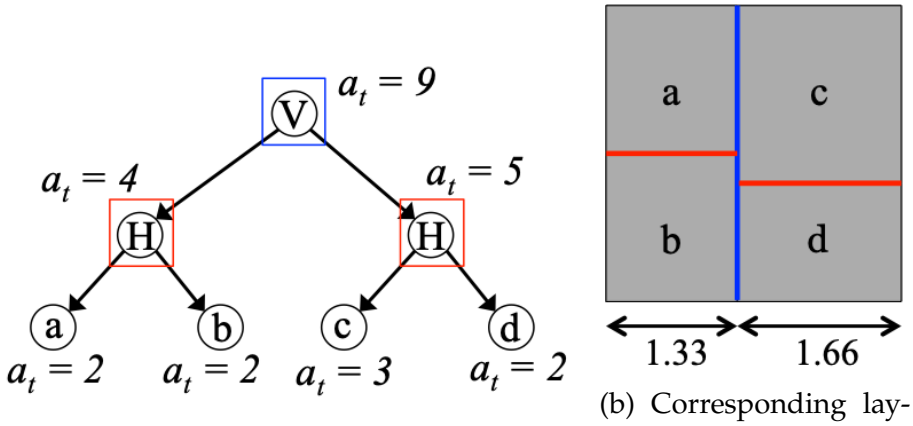
To overcome the first, our approach considers the allowed floorplan dimensions to be a budget: the layout always takes exactly the area it has been assigned. To overcome the second, and in combination with this notion, a top-down approach is proposed to determine the coordinates of each block. Instead of building the floorplan from the leaves to the root, the algorithm begins at the root and distributes the available area between its two children, depending on the node operator and area of its subtrees, and then proceeds recursively. Sometimes the approach requires to make space for macros by moving area from one child to the other and increase the cost function OVERLAP depending on the kind of area that was yielded (a_t, a_m or macro area, from least to most severe).

As exposed in Sect. 5.5.1, the floorplanning problem is being solved in a hierarchical flow: each block to be floorplanned represents a hierarchy level in the circuit. When fixing the size of a hierarchical block in a layout, it must be checked that the macros under that hierarchy fit using a slicing structure in the alloted area. The motivation is illustrated with a basic example in Fig. 5.14. Assume a block with 8 1x1 sized macros is alloted a 4x2 area in the layout. The would fit in the alloted area, so the rectangle size is strictly legal. However, when floorplanning the two sub-hierarchies of our 8-macro block at a latter step, suppose 5 of the



(a) Non-slicing. (b) Overlapping.

Figure 5.14: Not ensuring enough space for a hierarchical slicing placement of sub-blocks generates illegal situations.



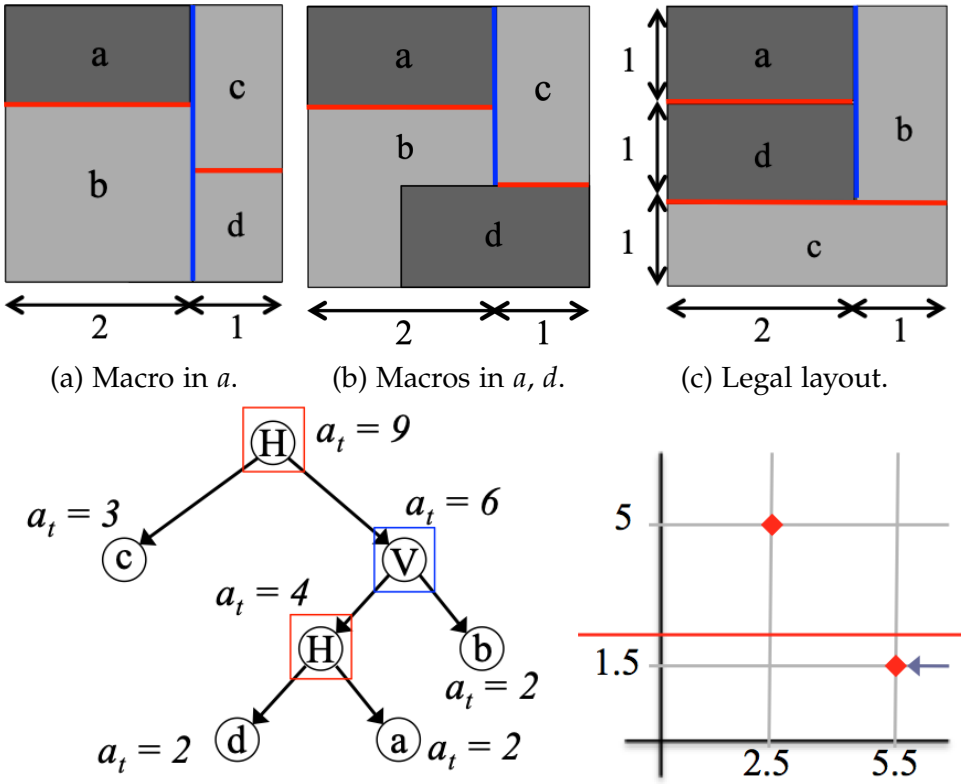
(a) Slicing Tree representation. (b) Corresponding layout.

Figure 5.15: Recursive layout generation.

macros belong to a sub-hierarchy and 3 to another. The kind of solution presented in Fig. 5.14a is not reachable, as it is not slicing. The one in Fig. 5.14b is slicing but oversized, resulting in OVERLAP penalization.

The following example illustrated the algorithm in the case of standard cell blocks only. Let Γ^n, a_m^n and a_t^n for a given slicing tree node n characterize its subtree as a block. They are computed at the beginning of the layout generation from the blocks at the leaves by composing shape curves and adding areas up to the root, similarly to the classical slicing tree to layout algorithm. The top-down process begins at the root, when the entire area is available. At each node, the available area is partitioned vertically or horizontally (depending on the node operator), according to the target area a_t sum of its subtrees. The process continues until the leaves are reached, when the block is assigned a rectangle in the layout.

Fig. 5.15 shows a slicing tree where each leaf has target area a_t, and its layout considering a budget of 3x3 area units. Beginning from the leaves, the a_t of all nodes are computed up to the root. The first partition, the vertical blue line from the V cut in the root, divides the horizontal budget



(a) Macro in a. (b) Macros in a, d. (c) Legal layout.

(d) Slicing Tree for (c). (e) Block shape curve.

Figure 5.16: Handling of macros in blocks.

among its children, with a_t 4 and 5. The process continues recursively with the H cuts until the leaves are reached and the rectangle on the layout for each block has been decided.

Slicing Tree to Layout, Detailed View

This technique guarantees the target area a_t demands of all blocks are met, but some layouts are illegal when considering macros. It would be the case if node a contained a macro with $w = 2$ and $h = 1$, since such macro would not fit in the alloted space in the example. The situation is shown in Fig. 5.16a. It presents the same case shown in Fig. 5.15, but now the Γ of block a indicates it contains a macro with size 2x1. Given the minimum width is 2 for the block, the idea to solve the situation is to push the cut that before was done at 1.33 (according to the a_t of the subtrees) to 2, thus satisfying the macro sizing constraints of the design, but at the cost of stealing area of blocks c and d, slightly increasing the

OVERLAP factor of the cost function. If block d also has a macro with the same size, the sum of both macro widths in this cut is 4, and both macro sizing requirements can not be satisfied at the same time. In this case, the cut is done according to one of them, and the OVERLAP increases severely. A possible solution without OVERLAP penalization is shown in Fig. 5.16c, deriving from the alternative slicing shown in Fig. 5.16d.

Algorithm 7 Slicing tree to layout: vertical cut.

```
1:  Input: width (w), height (h), slicing_node (p)
2:  Output: cuts set S_C, OVERLAP ∈ ℝ
3:  n, m ← left_child(p), right_child(p)
4:  target_area ← a_t^n + a_t^m
5:  w_cut ← w · (a_t^n / target_area)
6:  macro_w_n, macro_w_m ← min_w_for_h(Γ^n, h), min_w_for_h(Γ^m, h)
7:  if macro_w_n + macro_w_m > w then
8:     OVERLAP ← OVERLAP + C_1 × overlapping_area
9:  else if macro_w_n > w_cut then
10:    w_cut ← macro_w_n
11:    stolen_w ← w_cut − w
12:    extra_w_m ← (a_t^m − a_m^m) / h
13:    if stolen_w ≤ extra_w_m then
14:       OVERLAP ← OVERLAP + C_3 × stolen_w
15:    else
16:       OVERLAP ← OVERLAP + C_2 × h × (stolen_w − extra_w_m)
17:       OVERLAP ← OVERLAP + C_3 × h × extra_w_m
18: else if macro_w_m > w − w_cut then
19:    ...                                   {Same code for other child.}
20: S_C ← w_cut
21: recursive_call(w_cut, h, n); recursive_call(w − w_cut, h, m)
```

A fragment of the process to generate a layout from a slicing tree and handle its illegality through the OVERLAP factor is shown in Algorithm 7. The input is a w and h budget where to place the components under the slicing node p. Let Γ^n, a_m^n and a_t^n for a given slicing tree node n characterize its subtree as a block as described previously. A division of the area in the form of a vertical cut (x_cut) is proposed according to the target area of the subtrees (a_t^n,a_t^m) and the budget in the given dimension (in this case, w). Same applies for horizontal cuts. Line 6 computes the minimum width of the macros in the subtrees n and m. This is done by examining their shape curve Γ. Given that the maximum h for the macros is known, the tool finds the minimum w such that there exists a point (x,y) in the shape curve with $x \leq w$ and $y \leq h$, guaranteeing all macros

for the block fit in said space. Consider the example Γ shown in Fig. 5.16e. This block has two possible macro shapes, (2.5, 5) and (5.5, 5), but given for example a fixed at 2, the min width of its macros is to be considered 5.5.

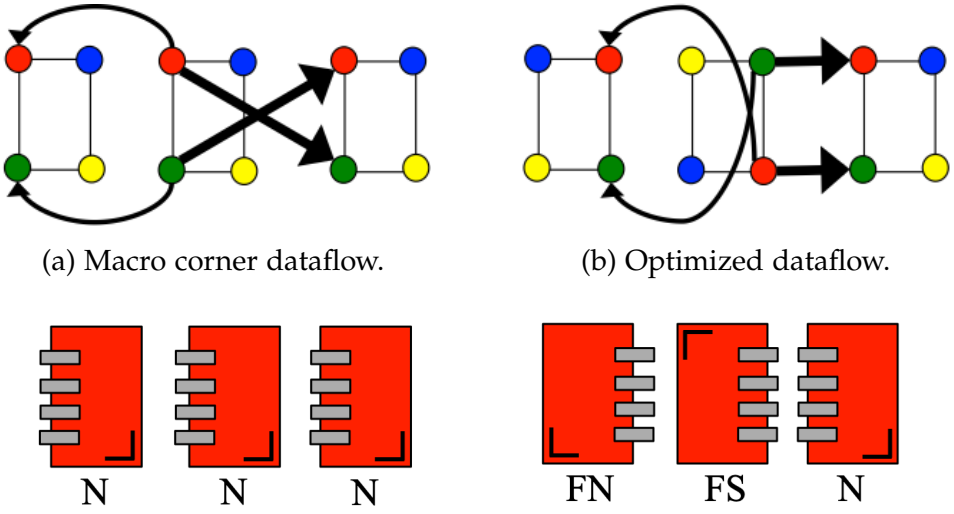
If the sum of the minimum w of both sides is greater than w, the macros overlap and the OVERLAP component increases by a constant C_1 times the overlapping area. Otherwise, if the left block needs more space, the increase in OVERLAP depends on the amount of stolen area and whether it belongs to the a_m (*medium*) or the a_t (*small*) of the right block. Note $C_1 > C_2 > C_3$ to penalize macro illegalities over area transfers between blocks. Experimental results show that setting C_2 or C_3 to 0 generates layouts which are not well spread, later resulting in increased routing congestion. After line 15, the process is repeated for the case of the right block needing more space for its macros, and then the function is recursively called until the leaves are reached.

5.5.6 Macro Orientation

The macro orientation step takes place outside the hierarchical flow, after the location of all macros has been decided. Its aim is to find the best possible orientation of macros to ease the communication between their ports. A macro flipping heuristic aimed at minimizing wirelength and timing is proposed. During macro floorplanning, communication between macros has been assumed to pass through their center. At this stage, communication is assumed to take take place from macro corners to macro corners, and the optimization process flips macros to reduce the dataflow affinity between them.

An example is shown in Fig. 5.17. In Fig. 5.17a, the dataflow affinity between the corners of 3 macros is shown. The width of the arrows between the colored nodes representing corners indicates the amount of dataflow affinity. In the previous stages of the algorithm, it would be known that the middle macro has more affinity with the macro to the right, but there would be no incentive to have the ports looking in that direction (resulting in the default macro orientation shown in Fig. 5.17c). However, by only flipping the macros, the distance between the critically connected corners can be reduced (Fig. 5.17b) to provide a better orientation from a dataflow affinity point of view (Fig. 5.17d).

The macro flipping process consists of two steps. First, modified versions of G_{seq} and G_{df} for the whole circuit are computed, where each macro is divided in four nodes (representing its corners). The dataflow



(a) Macro corner dataflow.

(b) Optimized dataflow.

(c) Default orientation.

(d) Optimized orientation.

Figure 5.17: Macro flipping for reduced wirelength.

from and to these corners is computed according to the algorithms shown in Sect. 5.5.4. A simulated annealing process is done for the horizontal and vertical dimensions. The solution space is a bit vector indicating whether or not each macro has been flipped in the dimension, and its perturbation is to flip one of the bits. A flip in the horizontal dimension causes the coordinates of the left corners to swap, and same for the right corners. The function to minimize is analogous to the DATAFLOW component in Sect. 5.5.5.

The process is repeated several times for each dimension. It is efficient, given that the perturbation is fast and the cost function can be computed incrementally, and allows pins to face the most urgent direction according to dataflow affinity, ensuring that very connected clusters of macros have their ports facing each other.

5.6 Experimental Results

Array information for dataflow analysis is essential for the quality of results in our method. Unfortunately, this information is not available in open benchmarks available to academia such as the ICCAD'12 benchmarks [34]. For this reason, a set of 8 real industrial examples of challenging circuits were used to validate our approach. The approach could not be compared to other academic efforts due to the partially industrial nature of our project (given their licenses only allow academic use, but

Table 5.2: Average WL, WNS and effort for the three flows.

	WL	WNS	TNS	Effort
IndEDA	1.143	-39.1%	-1810	10-30 mins (CPU)
HiDaP	1.013	-24.6%	-1059	0.5-2 hours (CPU)
handFP	1.000	-17.9%	-835	2-4 weeks (engineers + CPU)

had to run on company servers). The final handcrafted layouts obtained by expert backend engineers were available for comparison. The following three floorplan flows are compared:

Industrial EDA (IndEDA) Floorplan obtained with a state-of-the-art industrial tool using high effort settings.

Hierarchical Dataflow Placement (HiDaP) Floorplan by HiDaP, best WL of three ($\lambda = 0.2, 0.5, 0.8$).

Handcrafted floorplan (handFP) Floorplan manually obtained by expert backend engineers at the company.

Metrics are taken after placement of standard cells using the same tool as **IndEDA**. Wirelength averages are shown using the geometric mean to reduce sensitivity to extreme values. The more positive the worst negative slack and total negative slack metrics are, the better (0 means timing closed).

Table 5.2 shows summarized flow results, with average WL relative to **handFP**, WNS in clock cycle percentage, TNS in ns and the effort to create the solutions. While **IndEDA** execution takes from 10 to 30 minutes, our approach **HiDaP** takes from 30 minutes to 2 hours and obtains results approaching those of the **handFP** flow, where floorplans have taken weeks of iterations by the physical design engineers.

Table 5.3 shows the detailed metrics. Rows represent a circuit and floorplan flow, and columns give information on wirelength (in meters, and normalized with respect to **handFP**), congestion (global routing overflow percentage) and timing information (worst negative slack, in percentage of the clock period, and total negative slack). When comparing the **HiDaP** and **IndEDA** flows, wirelength is smaller using HiDaP in all but one case. Congestion is similar, with two cases with noticeably less (*c6*, *c7*). Our flow always obtains lower WNS, and also lower TNS in most cases.

The mean wirelength in **IndEDA** is 14.3% higher than **handFP** while obtaining generally worse congestion and timing. When comparing **HiDaP** and **handFP**, the mean WL increase is only 1.3%, and the manual flow obtains generally better results in all metrics with 2 exceptions. In *c3*, our

Table 5.3: Metrics after placement using the three flows.

		Wirelength		Cong.	Timing	
	Flow	WL	norm.	GRC%	WNS%	TNS
c1	**IndEDA**	13.19	1.029	6.51	0.0	0
520k cells	**HiDaP**	13.40	1.046	7.83	0.3	0
32 macros	**handFP**	12.81	1.000	7.36	-0.2	0
c2	**IndEDA**	46.01	1.180	12.99	-44.5	-931
3.95M cells	**HiDaP**	40.72	1.045	13.00	-19.0	-329
100 macros	**handFP**	38.97	1.000	9.33	-11.2	-213
c3	**IndEDA**	44.83	1.175	10.09	-75.5	-553
3.78M cells	**HiDaP**	35.02	0.918	8.29	-17.5	-260
94 macros	**handFP**	38.16	1.000	9.15	-17.8	-317
c4	**IndEDA**	45.03	1.174	7.24	-54.4	-2167
4.81M cells	**HiDaP**	40.43	1.054	4.94	-31.2	-2686
122 macros	**handFP**	38.35	1.000	3.33	-22.8	-1736
c5	**IndEDA**	44.25	1.162	2.02	-30.8	-1940
1.39M cells	**HiDaP**	39.51	1.038	4.72	-25.1	-1149
133 macros	**handFP**	38.06	1.000	3.42	-39.8	-1017
c6	**IndEDA**	96.42	1.288	9.95	-70.0	-15341
2.87M cells	**HiDaP**	79.20	1.058	2.22	-37.0	-5051
90 macros	**handFP**	74.87	1.000	1.63	-27.3	-3688
c7	**IndEDA**	41.44	1.174	38.56	-34.9	-1060
1.67M cells	**HiDaP**	35.52	1.007	6.47	-29.9	-1059
108 macros	**handFP**	35.29	1.000	4.61	-20.4	-774
c8	**IndEDA**	24.85	0.987	1.02	-3.4	-44
2.20M cells	**HiDaP**	23.75	0.944	1.37	0.0	0
37 macros	**handFP**	25.17	1.000	0.93	-3.9	-24

flow obtains 8.2% less wirelength with reduced congestion and timing, and in *c8*, HiDaP beats the other flows in wirelength and closes timing while maintaining similar congestion.

5.7 Conclusions

This chapter proposed HiDaP, a tool that exploits RTL information for macro floorplanning using a hierarchical multi-level flow. The key contributions presented in this chapter include:

1. Multi-level macro placement based on recursive block floorplanning with hierarchy-aware declustering.

2. Multiple graph-based data structures to model the circuit and estimate dataflow affinity between blocks, considering register pipeline latency and width.

3. Block area modeling considering macros and standard cells, with a top-down area-budgeting strategy aiming to avoid overlaps.

It has been tested on a suite of large industrial designs and compared against a state-of-the-art industrial floorplanner and against handcrafted macro placements. HiDaP improves on the industrial floorplan tool, and the obtained results are close to handcrafted designs with a fraction of the effort involved, providing a more promising start for the physical design iteration process. After validating the approach in this chapter and receiving feedback from a team of physical designers, the following chapter builds on the presented methods to obtain macro placements with better timing properties and user-friendly constraints, such as the capability to receive preferred macro locations as additional input.

5.A Design Mapper

This appendix presents the integration of the presented dataflow analysis algorithms into the Design Mapper system, to help engineers visualize and understand the structure and floorplans of their blocks.

5.A.1 Motivation

As presented in previous sections, floorplanning has a huge impact on the final quality of results of a circuit layout. Although current EDA tools offer some macro placement capabilities, their final locations and orientations are decided manually by expert physical design teams. They usually receive the netlist from an RTL team, with possibly a high-level idea of the main structures and communications in the design. However, when this description is inaccurate, obsolete or not available, the physical design team is left with the task of reverse-engineering the design communications and guessing how to best place the macros. Recovering high-level information from the netlist has been attracting attention lately [58]. Hierarchy information has already been used in GUIs as an exploration tool for designers, but the array information present at the RTL has not been sufficiently explored. In particular, exploiting array structures, such as multibit registers and ports, can help understand the

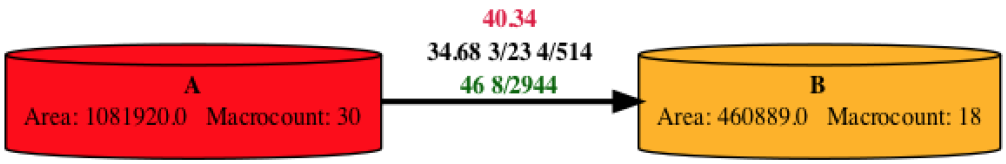


Figure 5.18: Nodes and dataflow edge example

data flow relations of blocks in a design, which is an essential part to finding a good floorplan in as few iterations as possible.

Our proposal

Design Mapper is a design analysis system with the goal of easing the process of understanding circuit structure by automatically processing its netlist and producing human-readable visualizations. The system is composed of three tools:

Design Analyzer Corresponding to the presented HiDaP, back-end tool that takes a netlist as an input and, from a graph modeling the circuit, extracts the dataflow affinity relations between blocks of the design.

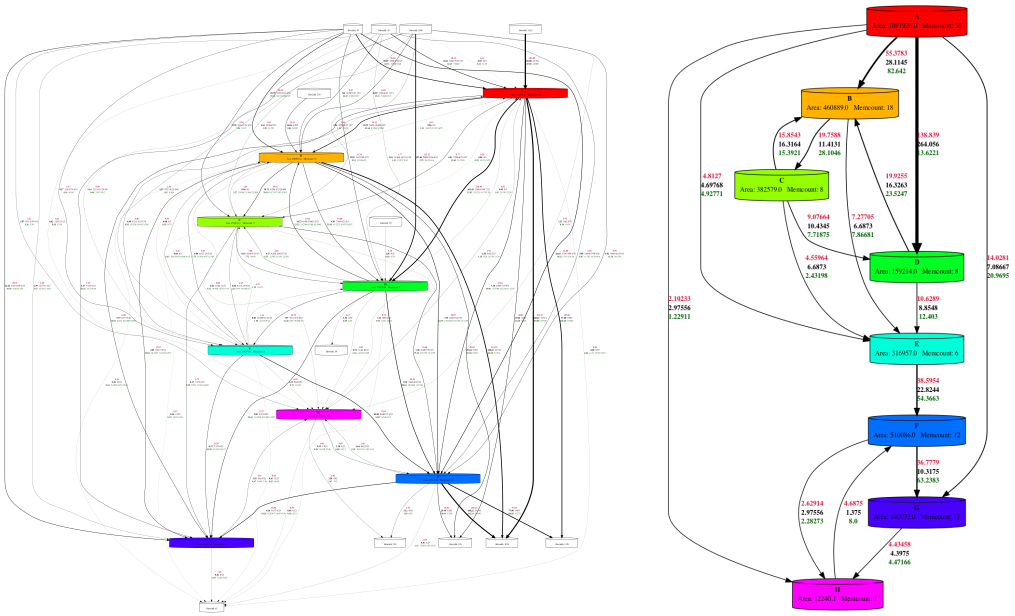
Dataflow Viewer renders graphs showing the structural information discovered by Design Analyzer. Useful to explore design structure and dataflow relations.

Layout Explorer receives a macro or block placement and shows a layout viewer enriched with dataflow affinity information. It can be used to understand the relation between particular block configurations and their dataflow, becoming a great help for layout validation.

5.A.2 Dataflow Viewer

Dataflow Viewer is the interface to visualize the dataflow graph, which is rendered using GraphViz [28]. Nodes are represented by colored cylinders with block name, area and macro count. The edges between them have three rows, offering average dataflow affinity, block dataflow and macro dataflow information.

In the example shown in Fig. 5.18, the second row (block dataflow) indicates there are 23 bits at flop latency 3 and 514 at flop latency 4 going from block *A* to block *B*. To assign an overall dataflow affinity score to



(a) Full graph (b) Simplified graph.

Figure 5.19: Views of a dataflow affinity graph

each row, we use $\sum_i$ *#bits*$_i$ / *latency*$_i^k$ as proposed in previous sections, based in the dataflow affinity relation. i iterates over all pairs *#bits*/*latency* and parameter k controls the exponential decay impact of latency. In this case, the block dataflow between the blocks is 34.68. On the other hand, there are 2944 bits at flop latency 8 going from block A to block B through other blocks, with a macro dataflow of 46. The resulting 40.34 dataflow affinity is the average of both.

These graphs can have in the order of tens of nodes and hundreds of edges. Fig. 5.19a offers the view of the full graph for a real circuit with 94 macros and 3.78 million cells. In order to provide basic insight on circuit structure at a glance, the size of the edges is proportional to the amount of dataflow between the blocks. To allow for an easier exploration to designers, a GUI tool has been developed to filter the graph according to several criteria: removing edges with low block or macro affinity, removing ports with low bitcount or focusing on particular parts of the hierarchy or blocks. Fig.5.19b shows a view of the same graph after the less relevant edges are removed, revealing a pipeline structure between the blocks.

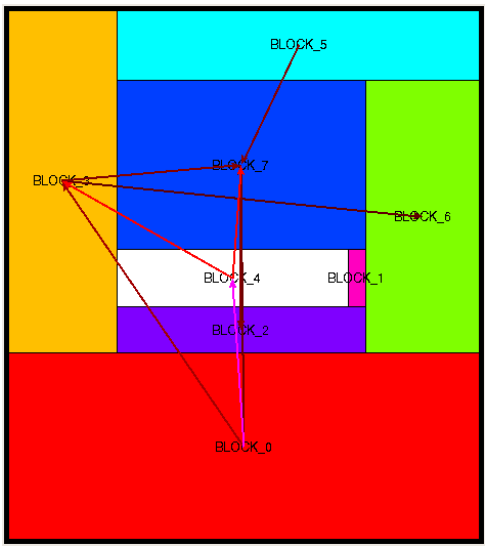
5.A.3 Layout Explorer

Layout Explorer is an interactive GUI that combines information from the dataflow affinity file and design layout. A first floorplan quality assessment can be conducted using Layout Explorer in block mode as shown in Fig. 5.20a. Position of blocks can be inspected together with dataflow affinity information. By hovering and selecting a block (white focus), the related edges are highlighted and additional information is provided.

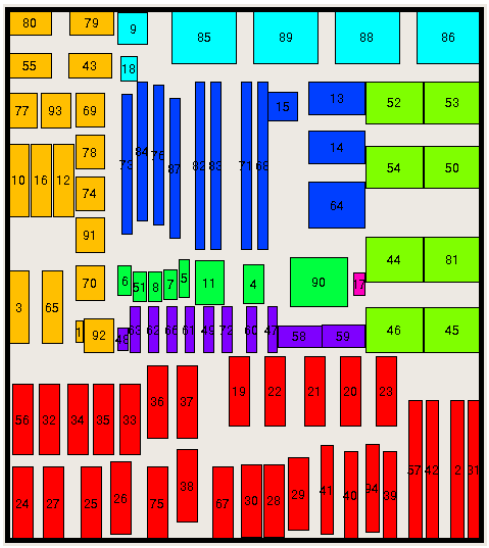
If a full macro placement is available, more information at macro level is shown as seen in Fig. 5.20b, where the macro placement corresponding to the block layout are seen. Again, the dataflow affinity information is printed in the form of edges between components, which can be filtered according to their weights, ports according to bit count, etc. Fig. 5.20c shows the same macro placement with additional array and hierarchy information. A view of the circuit by a state-of-the-art industrial physical design tool is shown in Fig. 5.20d, with red lines marking the proposed block coordinates by a floorplanner based on the dataflow affinity analysis.

Conclusions

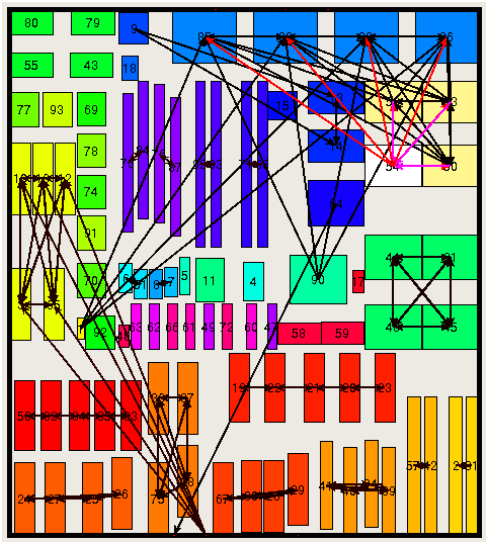
Design Mapper offers a global view of current complex designs by analyzing and displaying the dataflow relations between circuit components discovered by HiDaP. The information is helpful during both floorplanning exploration and validation. Our system contributes to closing the gap between the engineers and EDA tools, and it has been deployed in current design flows to help physical designers find better floorplans in less time.



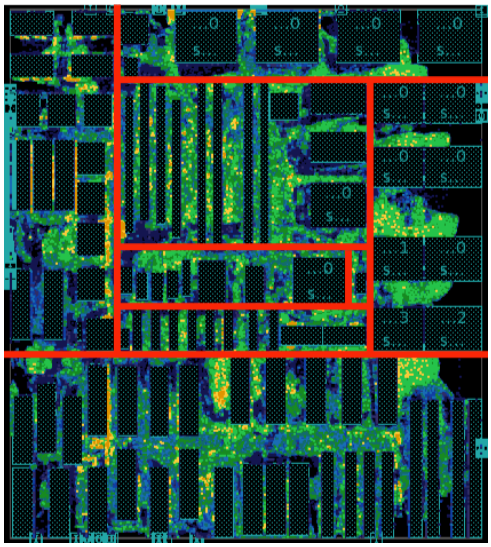
(a) Block view



(b) Macros view



(c) Detailed macros view



(d) Commercial tool viewer

Figure 5.20: Multiple design views

Chapter 6

Adaptive Macro Placement Guided by Analytic Methods

This chapter extends the macro placement approach presented in the previous chapter with the aim of obtaining layouts with better quality and enhancing usability from the point of view of the engineers. The key improvement is to allow the simulated annealing search to minimize macro distances to suggested locations, which can either be provided by the engineer or obtained via automated methods. In order to do so, the objective function becomes a multi-objective cost function with self-adaptive parameters to minimize dataflow, overlaps and distance to preferred positions. Additionally, methods to suggest macro locations based on spectral and force methods are proposed. Experimental results show our floorplans can obtain, after placement, better timing (44% reduction in TNS and 7.7% of period in WNS) with a reduction of about a 1% in terms of wirelength when compared to handcrafted macro placements, and can be brought to almost timing closure with few DRC violations after minimum manual modification.

6.1 Motivation

The results of the approach presented in the previous chapter showed how HiDaP managed to find layouts that were close in quality to tapeout-ready macro placements with a fraction of time involved in manual design. In particular, WL increase was close to a mere 1%, but timing results were still generally worse in terms of WNS and TNS.

On the other hard, HiDaP was integrated in the physical design flow of eSilicon and tested by several teams of physical designers. Their main

concern with respect to the usability of the tool came from a certain lack of interaction. In particular they asked for:

Macro location control For the tool to be most usable in the flow, the engineers asked for a method to be able to suggest macro locations to HiDaP. Given their domain knowledge of particular blocks or design constraints, they might decide that certain macros should be in particular locations.

Reduced solution variability The engineers observed that several layouts with similar dataflow scoring can have substantial differences with regards to block relative locations.

Keepout sizing control Modern technology nodes impose minimum distances between macros, but often they are not enough to ensure good routability. Although the problem of deciding good channel sizes is complex and has not been automated, our proposed first step is to allow the configuration of macro and block walls keepouts to avoid trivially solvable congestion-induced DRC violations hotspots.

Dealing with keepout sizing is relatively easy by modifying component and block sizes. The first and second demands require a more in-depth modification of the algorithm to be realized given the stochastic nature of the algorithm. However, it also presents potential synergy with our aim of obtaining better quality in our layouts. All these issues are addressed by adding the minimization of distance to preferred block locations into the layout generation phase. These preferred block locations can be a human input to the algorithm or can be generated by methods which do not necessarily need to provide non-overlapping solutions, but provide rough estimations of good macro locations.

A multi-objective cost function is introduced to model the problem of minimizing wirelength and timing (dataflow), distances to preferred locations and overlaps. The weight of each component is parametrized using constants that must be introduced by the user. However, their value must adapt to the current execution conditions to obtain good results, because different circuits and even different floorplanning subproblems inside a same circuit can have very different value ranges in the cost function components. An adaptive scheme updates these parameters so that they always represent a percentage of the global cost function, automating their adjustment and easing user interaction.

After describing the contributions related to both these items to obtain better quality and address engineer feedback, extensive experimental results are presented. The central question these experiments answer is whether or not HiDaP can be useful for engineers. Other concerns such as the impact of particular algorithms options and configurations are also addressed. The results prove the success of the approach and suggest future lines of work for solving the macro placement problem in modern circuit designs and technologies.

6.2 Contributions

The algorithms presented in Chapter 5 are extended to incorporate the new desired functionality and quality improvement. In particular, the problem definition is updated (in bold) to reflect these requirements. The new aim is to obtain floorplans that show:

1. No overlaps between macro components.
2. Least possible wirelength, worst negative slack and total negative slack.
3. **Macros being close to predefined preferred locations.**

The last item provides additional control of the result of the algorithm to the physical engineers: extra input can be provided, proposing location for a set of the macros in the design. Our approach is ready to take them into consideration during floorplan generation by minimizing the distances in the solution to these proposed distances. Preferred locations can also be generated algorithmically by our spectral and force-directed methods, or any other third party method such as an industrial coarse placer. Other extensions to support additional engineer input are incorporated, such as macro and block keepout configurations.

Adaptive Multi-objective Cost Function.

The multi-objective cost function allows the simultaneous minimization of dataflow, distances to preferred locations and overlap. In line with our concern for the usability of our tool and to let the user find the most suitable macro placement for their needs, the engineers can decide the relative weight of each optimization objective in the novel adaptive multi-objective cost function, and modify the criteria to decide the best solution. More details can be found in Sect. 6.3.

Automated Macro Placement Suggestions.

The source for the preferred macro locations can be another algorithm. Our proposal is to use spectral and force methods to compute these locations automatically. These are methods that are good at finding locations for point objects and can provide an idea of where macros want to go. In particular, spectral methods alone can be used to determine the location of each block in a particular simulated annealing instance by running spectral placement of its dataflow graph. On the other hand, force-directed methods can be added for extra spreading and allow the method to be applied once at the beginning of the HiDaP execution on the sequential graph, providing a global view of the macro preferred locations.

6.3 Multi-Objective Cost Function

As exposed in Sect. 6.2, our method proposes to find layouts that minimizes several metrics at the same time: wirelenght and timing (modeled by dataflow), distances to preferred locations and overlaps. The proposed cost function to minimize is:

$$\min \mu_1 \times \text{DATAFLOW} + \mu_2 \times \text{DISTANCE} + \mu_3 \times \text{OVERLAP}$$

The parameters μ_1, μ_2, and μ_3 adjust the weights of the components and are automatically updated during the simulated annealing execution as expained in the next section. DATAFLOW represents, for each pair of blocks b_i, b_j, the sum of the products of their dataflow affinity and distance,

$$\text{DATAFLOW} = \sum_{b_i, b_j} \mathit{distance}(b_i, b_j) \times \mathit{dataflow\_affinity}(b_i, b_j)$$

The minimization of the DATAFLOW component ensures that blocks that have large array connections and relatively small latency are close in the layout, aiming at obtaining good properties in terms of wirelength and timing. The position of ports and macros outside the current placement instance are considered a fixed points during the computation. The second component DISTANCE represents the displacement between the block in the layout and its preferred location (see Sect. 6.4 for details). Finally, the OVERLAP factor represents the sum of overlapping areas in the layout multiplied by a constant that increases with the severity of each illegality. This mechanism is used to ensure macro overlaps penalize more than standard cell area overlaps.

6.3.1 Adaptive Parameters

The adaptive parameters μ_1, μ_2, and μ_3 automatically adjust the relative weight of the components in the objective function during the exploration. This approach presents two benefits:

1. The algorithm adjusts the objective function weights automatically for all possible designs: if the weights were fixed, they would need to be hand-tuned for each new instance according to the relative value of each component in the function.

2. The objective function automatically becomes smoother with regards to the OVERLAP factor. At high temperature, illegal layouts should not be penalized severely to allow the heuristic to explore the solution space, but by the end of the exploration they should be avoided.

In order to control the multiparameter optimization, the user does not directly indicate μ_1, μ_2, and μ_3, but the desired relative weights for each one of the components of the formula: α_1, α_2, and α_3 (with $\alpha_i \in [0,1]$, and $\alpha_1 + \alpha_2 + \alpha_3 = 1$). At each temperature change in the simulated annealing, the values of μ_1, μ_2, and μ_3 are updated to ensure their components have the chosen relative weight in the aggregate function.

Algorithm 8 Update of objective function weights.

1: **Input:** $\mu_1, \mu_2, \mu_3, \alpha_1, \alpha_2, \alpha_3$
2: **Output:** μ_1, μ_2, μ_3
3: $max\_var \leftarrow \beta \times (1 - DT)$
4: $\mu'_1 \leftarrow (\alpha_1 \times mean\_total\_cost) / mean\_\text{DATAFLOW}\_cost$
5: $\mu'_2 \leftarrow (\alpha_2 \times mean\_total\_cost) / mean\_\text{DISTANCE}\_cost$
6: $\mu'_3 \leftarrow (\alpha_3 \times mean\_total\_cost) / mean\_\text{OVERLAP}\_cost$
7: **if** $\mu'_1 \geq \mu_1$ **then** $\mu_1 = min(\mu'_1, \mu_1 \times (1 + max\_var))$
8: **else** $\mu_1 = max(\mu'_1, \mu_1 \times (1 - max\_var))$ {Same for μ_2, μ_3}

The update method is shown in Algorithm 8. In line 3, DT is the temperature decay factor of the simulated annealing. When multiplied by a user-defined parameter β (2 in all our experiments), it is stored in *max_diff* as the maximum variation (as a percentage) allowed to μ_1, μ_2, μ_3. In lines 4 to 6, values for the multipliers are proposed to that each component has the relative weights $\alpha_1, \alpha_2, \alpha_3$. Lines 7-8 show how the increase is reduced if it is bigger than *max_var*. This mechanism, along with using the mean values of the cost components of the layouts found during the

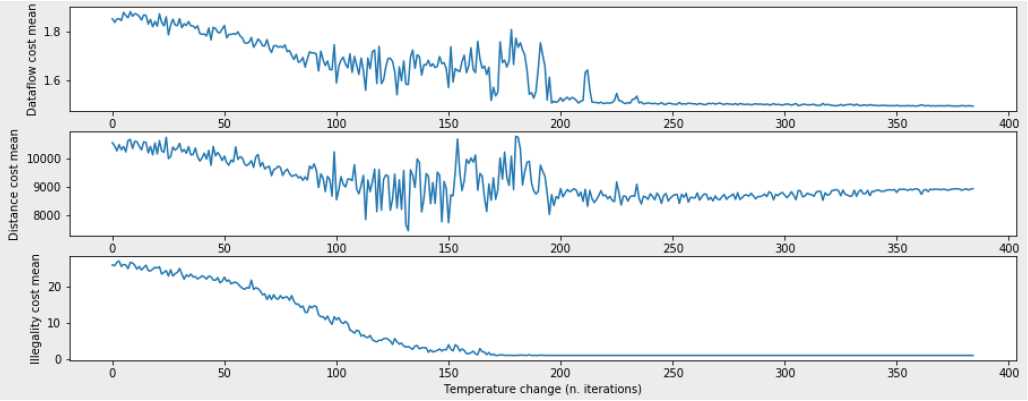


Figure 6.1: Evolution of separate cost components.

last temperature, ensure the multipliers do not change sharply, trapping the current layout in a early local minimum.

Fig. 6.1 illustrates the evolution of the objective function components during a real execution. The graphics from top to down represent DATAFLOW, DISTANCE and OVERLAP. The x axis counts how many times temperature has decreased (up to 400 temperature changes). At the beginning of the search, all three components get reduced. While exploring through illegal solutions is allowed, the OVERLAP component steadily declines until it becomes close to zero. At this point, its μ_3 becomes a high constant, and the search proceeds to optimize with the remaining μ_1, μ_2 until convergence in a final layout.

6.3.2 Keeping the Best Solution

HiDaP includes a mechanism to keep the best solution visited during the search. The final solution at the end of the search shown in Fig. 6.1 has no OVERLAP and reduced DATAFLOW, but not the lowest DISTANCE when compared to previously visited solutions. The example reveals it is not straightforward to define what *the best* solution is. More legal solutions are always considered better, and a trade-off is used to balance the final solution in terms of DATAFLOW and DISTANCE.

The proposed solution consists of keeping a set of good layouts found during the exploration. They are stored as a set of Pareto points in two dimensions (DATAFLOW and DISTANCE of the solution). The points are only stored if their OVERLAP value is equal to the lowest found until the point, ensuring the best solution is always the most legal. At the end of the search, given the lowest found DISTANCE score min_d, the

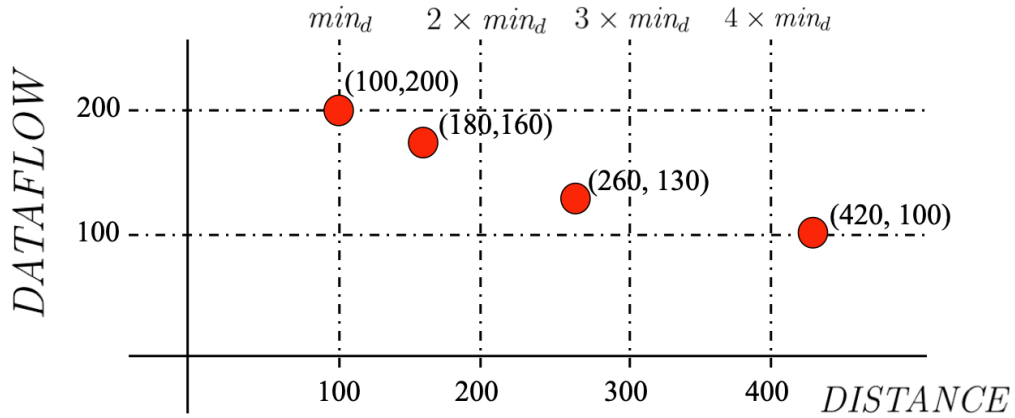


Figure 6.2: Example of a Pareto-optimal set of solutions.

solution with a DISTANCE equal or lesser to $\gamma \times min_d$ with the lowest DATAFLOW is considered the best solution ($\gamma \geq 1$ is a user-defined parameter).

The mechanism is illustrated in Fig. 6.2. It depicts a set of candidate layouts where the x and y axis represent DISTANCE and DATAFLOW, and each point is one of the most legal Pareto optimal solution at the end of a simulated annealing search. If the user-configurable parameter γ is 1, the best solution is the one where DISTANCE = 100, DATAFLOW = 200. But if more flexibility is allowed in DISTANCE minimization (γ is 2), a sligthly better solution in terms of DATAFLOW can be found (180, 160). Finally if a much greater γ is allowed, the solution that best minimizes DATAFLOW is (420, 100).

6.4 Preferred Macro Locations

Preferred macro locations allow for extra engineer control and solution quality. Through this section, it is assumed that the provided preferred locations have good properties, either because they have been obtained using good placement algorithms (point placement techniques give optimal solutions assuming the elements are dots) or provided by expert engineers. Our approach helps in three regards:

Reduce solution variability

Biasing the search towards some preferred macro locations can help to more thoroughly explore the solution neighborhood around the preferred locations and reducing solution variability.

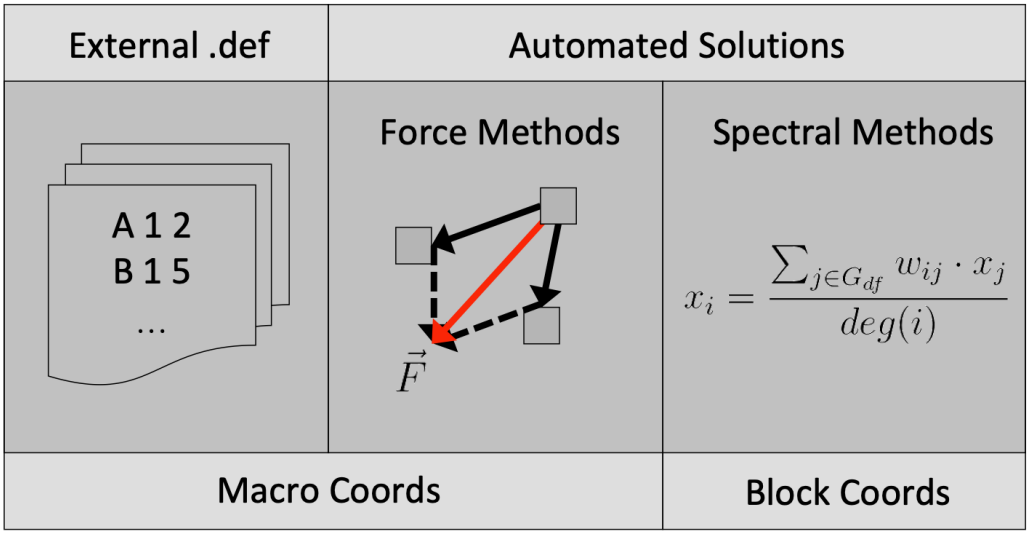


Figure 6.3: Taxonomy of preferred location sources.

Increase solution quality
In case of several solutions with similar dataflow, preferred locations with good properties (for example, minimizing WL or other metrics) should break the tie towards better solutions.

Increase engineer interactivity
Engineers can provide locations to guide the search and use the parameters described in the previous sections to decide their level of influence.

Fig. 6.3 shows the three proposed methods to provide preferred coordinates to the floorplanner. On one hand, they can be given at two levels: macro locations or block locations. The simulated annealing algorithm always works at a block level, but these blocks contain macros. If block coordinates are provided, they are directly used during the optimization problem (as shown in Sect. 6.3). If macro coordinates are provided, the center of gravity of the location of all macros inside a given block is computed to find the preferred block location.

The source of these coordinates can an external .def file, force-directed methods or spectral methods. The first two correspond to macro coordinates and propose a global suggestion for the entire macro placement problem, whereas the last provides block coordinates directly. Succinctly,

1. Spectral methods are executed at each multi-level instance of simulated annealing and find the coordinates for each block based on the dataflow graph G_{df} of that particular instance.

2. Force-directed methods rely on placing the sequential graph G_{seq} at the beginning of HiDaP execution, by obtaining an initial solution using spectral methods but refining it using force-directed algorithms with special forces to avoid macro overlap.

6.4.1 Spectral Dataflow Placement

The input to the spectral dataflow placement is the placement area (a rectangle) and the dataflow graph G_{df}, where each node is a block to be floorplanned and edge weights represent the dataflow affinity between them. The output is the location for each block inside the rectangle. Coordinates of ports and macros are also provided to consider their location (possibly outside the placement region) during the spectral dataflow placement. The algorithm that has been implemented is a modified version of the spectral graph drawing method proposed by Koren [45], which is computationally fast and could be extended to handle fixed ports. The following introduction to spectral drawing can be found in more detail in Koren's paper.

Spectral Drawing Basics.

Given a weighted graph $G(V,E)$ with n nodes, its adjacency matrix A:

$$A_{i,j} = \begin{cases} 0, & i = j \\ w_{ij}, & i \neq j \end{cases}$$

and let its Laplacian matrix L be:

$$L_{i,j} = \begin{cases} deg(i), & i = j \\ -w_{ij}, & i \neq j \end{cases}$$

where w_ij represents the weight of the edge between nodes i,j and $deg(i) = \sum_j w_{ij}$ for every j neighbor of i. The paper denotes the 1-D layout of a graph G by $x \in \mathbb{R}$ where $x(i)$ is the location of node i. This x is the solution of the constrained minimization problem:

$$\min_x \quad E(x) = \sum_{\langle i,j \rangle \in E} w_{ij}(x(i) - x(j))^2$$

$$st. \quad Var(x) = 1.$$

This amount can be understood as an energy that strives to make edge lengths short, and specially edges with larger w_{ij}, which is desirable from a placement point of view. $Var(x) = 1$ forces nodes in the drawing area

and prevents them overcrowding the same point. However, $E(x)$ can be rewritten as $x^T Lx = \sum_{\langle i,j \rangle \in E} w_{ij}(x(i) - x(j))^2$, and the desired 1-D layout can be described as the solution of the constrained minimization problem

$$\begin{aligned} \min_x \quad & x^T Lx \\ st. \quad & x^T x = 1. \\ \textit{in the subspace:} \quad & x^T \cdot 1_n = 0. \end{aligned}$$

The solution of this problem is proven to be $x = v_2$, the second smallest eigenvector of L, with an optimal energy value of λ_2, its related eigenvalue. To make a 2-D drawing, an additional vector y is computed. As there must be no correlation between x and y to obtain them maximum new information, the problem is further constrained,

$$\begin{aligned} \min_y \quad & y^T Ly \\ st. \quad & y^T y = 1. \\ \textit{in the subspace:} \quad & y^T \cdot 1_n = 0, \quad y^T \cdot v_2 = 0. \end{aligned}$$

The process could continue for other dimensions, but our focus is two dimensional drawing to get a placement of our components.

Drawing using degree-normalized eigenvectors.

The innovation in the method presented in [45] comes with the proposal of the use of degree-normalized eigenvectors for graph drawing, and a fast optimization method to compute them that we adapt for our problem. Let us define the *degrees matrix* as an $n \times n$ diagonal matrix D with $D_{ii} = deg(i)$. Taking the original minimization problem and weighting sums according to masses produces the problem

$$\begin{aligned} \min_x \quad & x^T Lx \\ st. \quad & x^T Dx = 1 \\ \textit{in the subspace:} \quad & x^T D1_n = 0. \end{aligned}$$

The solution $x = u_2$ is the second smallest generalized eigenvector of (L, D), whose eigenvectors are called the degree-normalized eigenvectors. These are shown to coincide with the eigenvectors of the matrix $D^{-1}A$, the transition matrix of a random walk on graph G.

Using degree-normalized eigenvectors avoids the typical situation in which dense clusters are drawn very densely and the rest of the area is

not well used. This is a situation that arises when minimizing wirelength without considering whitespace distribution, as all cells want to be close to each other (for example using techniques such as quadratic placement). This approach adjusts edge weights to reflect relative importance to the local scale.

Computing degree-normalized eigenvectors with fixed nodes.

Given a node i, differentiating $E(x)$ with respect to $x(i)$, equating the result to zero and isolating $x(i)$ gives

$$x(i) = \frac{\sum_{\langle i,j\rangle \in E} w_{ij} x(j)}{deg(i)}.$$

That is, the optimal position for node i to minimize $E(x)$ is the weighted centroid of i's neighbors. But putting each node at the weighted centroid of its neighbors is the same as multiplying x by $D^{-1}A$. Iterating this process gives the *dominant* eigenvector of $D^{-1}A$, and can be modified to ensure convergence to u_2. The algorithm to compute degree-normalized eigenvectors relies on this fact to iteratively perform this product with other small adjustments until convergence is reached.

Our extension of the original algorithm consists in the incorporation of fixed nodes to the formulation. After each product $x_{i+1} = D^{-1}Ax_i$, the location of the fixed nodes is updated with its known positions. All nodes which are not fixed begin at a random position and converge to their ideal position.

Algorithm 9 Spectral placement with fixed nodes.

Input: *Dataflow Graph* (G), *convergence criteria* (ε)
Output: *Coordinates* (X, Y)
for *dim* = 2 **to** 3 **do**
  for $i = 1$ **to** N **do**
    if *fixed*(i) **then** $u_{dim}(i) = known\_location(dim, i)$ {*Assign initial locations.*}
    else $u_{dim}(i) = random(0,1)$
  while *not converged*(ε) **do**
    if *dim* = 3 **then** $orthogonalize(u_{dim}, u_1)$ {*Orth. to previous eigenvector.*}
    for $i = 1$ **to** N **do**
      $u_{dim}(i) \leftarrow \left(u_{dim}(i) + \frac{\sum_{\langle i,j\rangle \in E} w_{ij} x(j)}{deg(i)} \right)$ {*Power iteration.*}
    for $i = 1$ **to** N **do**
      if *fixed*(i) **then** $u_{dim}(i) = known\_location(dim, i)$ {*Fixed nodes to place.*}
    $u_{dim} \leftarrow \frac{u_{dim}}{||u_{dim}||}$ {*Normalize vectors.*}
return u_2, u_3 {*Return X, Y coordinates.*}

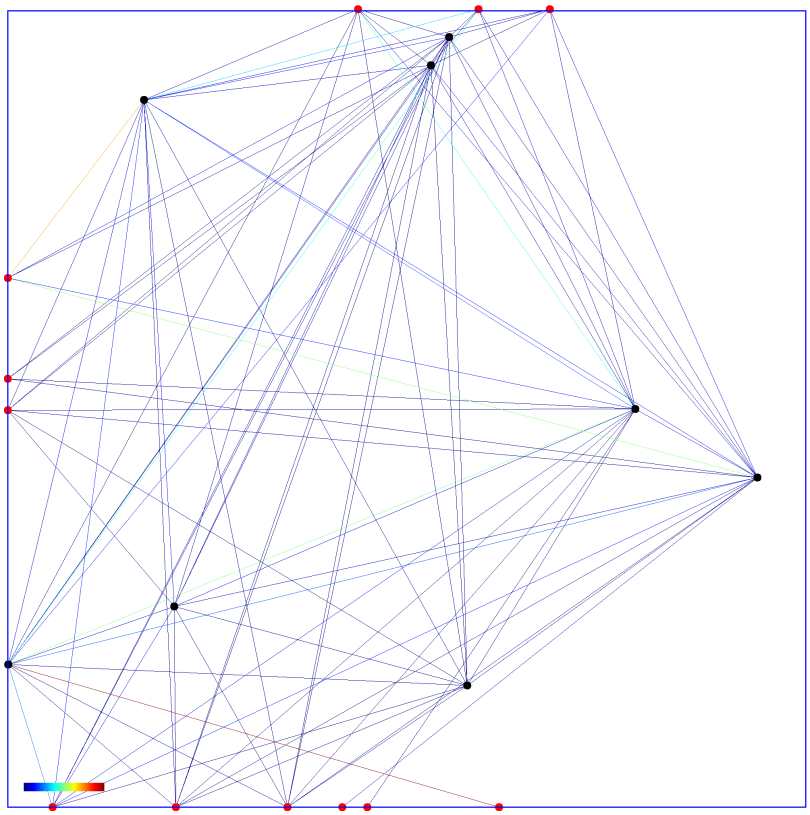
Our modified algorithm based on the proposal in [45] is shown in Algorithm 9. The input is the graph and a convergence control parameter ε, and the output are the X, Y coordinates of all vectors, from the eigenvectors u_2, u_3. As explained, first the X coordinates are computed. First the initial position for the dimension in u_{dim} is decided, to its known location if the node is fixed, or to a random position otherwise. Then until convergence, the vector is orthogonalized to the X vector (if computing Y), and the optimal position for each element is computed. Finally fixed nodes return to place and the positions are normalized. The resulting vectors u_2, u_3 contain the second and third degree-normalized eigenvectors and their values are interpreted as the desired coordinates for the blocks in our floorplanning problem.

In our use case, the number of nodes is small, usually less than 30. Some examples of resulting placements are shown in Fig. 6.4. In Fig. 6.4a, the spectral placement of the top simulated annealing problem of a design is shown. Black dots represent blocks and red dots represent fixed elements. Fig. 6.4b shows the spectral placement of an intermediate simulated annealing subproblem in the multilevel execution. The placement area is restricted with respect to the available space for the particular problem, but block ports and macros outside (in red) are still taken into account during spectral placement.

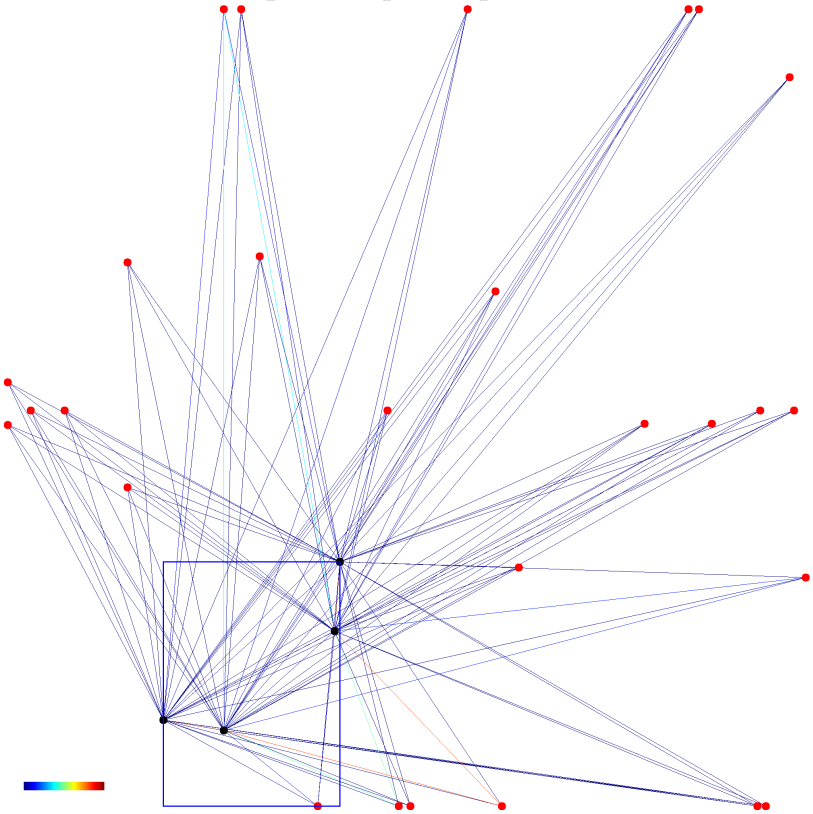
6.4.2 Force-directed Sequential Placement

The other analytic flow proposed in this chapter is the force-directed sequential placement, which is applied to the sequential graph G_{seq} instead of the dataflow graph G_{df}. The G_{seq} models macros, registers and ports, where edges represents amount of bits going from one to the other as seen in Chapter 5. It has tens of thousands of nodes, and the spectral method does not produce enough spreading, nor does provide good mechanisms to avoid macro overlap. The force-directed method is applied once at the beginning of the flow and macro positions are later used in the corresponding block floorplanning problems during the whole HiDaP execution.

The initial positions are determined by the spectral method, and later refined by graph-drawing force-directed algorithms based in well-known spring methods [25] [84]. The basic idea of these approaches is that edges in the graph represent springs. The nodes are placed in a two-dimensional layout and the system is released, so that springs move the nodes to find the minimal energy state where they are as close to their



(a) Top block spectral placement.



(b) Intermediate block spectral placement.

Figure 6.4: Example results of spectral placements.

desired length as possible. The corresponding spring force also depends on its edge weight, and the final effect is that nodes with higher weight end up being closer in the layout.

To avoid having all nodes being too clustered in these approaches, a repulsion force is added that pushes nodes away from all other nodes. However, in circuit placement uses, the presence of ports at the boundary of the drawing zone already has a graph-spreading effect. To suit our particular placement needs and for efficiency reasons, instead of this general repulsion force, a macro overlap removal force has been added, also providing some extra spreading to the register nodes.

Algorithm 10 Force-directed sequential placement, top view.

1: **Input:** *Coordinates* (X,Y), *sizes* (S), *termination criteria (max_iters,* ε*)*, *force parameters* (cte_a, cte_{oa})
2: **Output:** *Coordinates* (X',Y')
3: **for** $i = 1$ **to** N **do**
4: $P_i \leftarrow (X_i, Y_i)$ {*Assign locations from spectral.*}
5: $iter \leftarrow 1$
6: **while** $iter \leq max\_iters$ *and not converged*(ε) **do**
7: **for** $i \in mobile\_nodes$ **do**
8: $P \leftarrow attraction\_forces(i, P, cte_a)$ {*Macros and registers.*}
9: **for** $i \in macro\_nodes$ **do**
10: $P \leftarrow avoid\_overlap\_forces(i, P, cte_{ao})$ {*Macros only.*}
11: $(X',Y') \leftarrow P$
12: **return** X', Y' {*Return X, Y coordinates.*}

The top view of our force-directed sequential placement algorithm is shown in Algorithm 10. The input are the vectors with X, Y, S coordinates from spectral placement and component sizes, termination criteria parameters and user-defined force parameters cte_a, cte_{oa} to control the impact of the attraction and *overlap avoidance* forces. First, the position vector for each component is created. While the number of iterations has not exceeded *max_iters* and the system is not stable, all mobile nodes (registers and macros) are moved sequentially according to their attraction forces (ie. edges to other components), and all macros are moved to try to reduce overlap. At the end of the process, the new X', Y' coordinates for all components are returned. The following sections explains the details of the attraction and overlap avoidance forces.

Attraction forces computation.

The basic idea for the action of attractive forces is shown in Fig. 6.5. Each node i (gray boxes) is attracted to all its neighbors j with a force

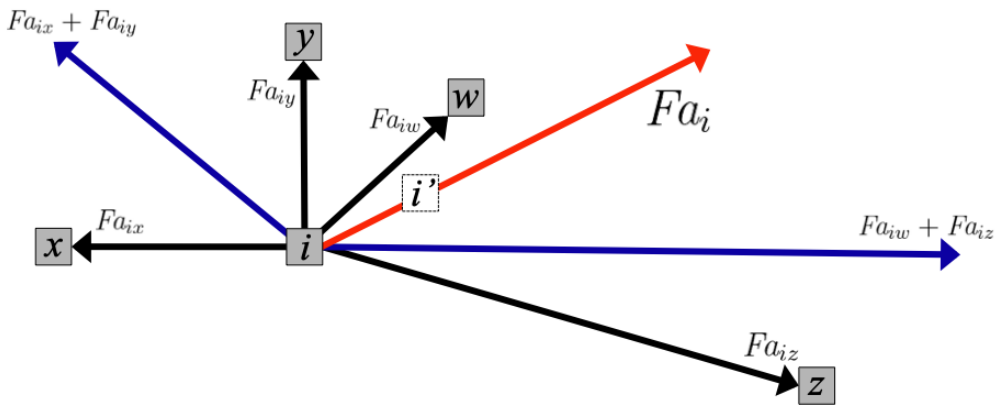


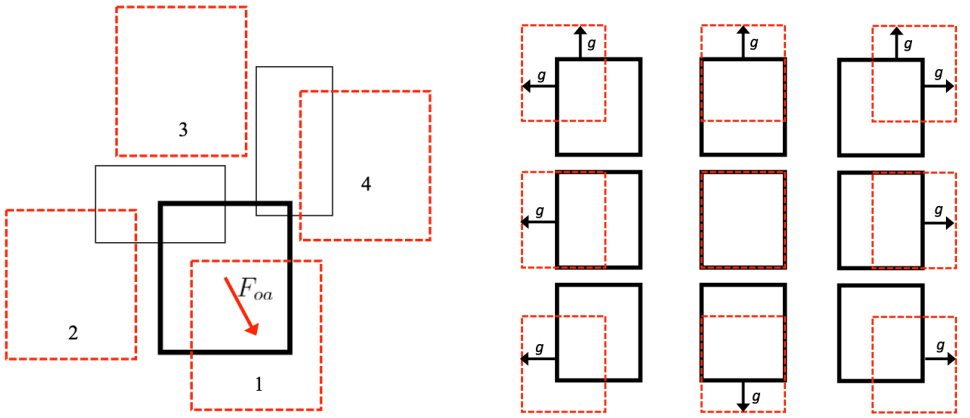
Figure 6.5: Example of attraction forces.

Fa_{ij} depending on their distance and edge weight w_{ij} (black arrows). All of these attractive forces are added (blue arrows) in the resultant force Fa_i (red arrow), and the node i moves a certain distance in its direction (new position is marked by the white box). Movement in any direction is halted if the node is to leave the placement area.

Algorithm 11 Force-directed sequential placement, attraction forces.

Input: *Node* (i), *positions* (P), *force parameters* (cte_a)
Output: *Positions* (P)
$Fa_i = (0,0)$
for $j \in neighbors(i)$ **do**
  $\delta_{ij} \leftarrow P_i - P_j$
  $direction_{ij} \leftarrow \frac{\delta_{ij}}{\|\delta_{i,j}\|}$
  $magnitude_{ij} \leftarrow \frac{\|\delta_{i,j}\|^2}{k} \times w_{ij}$
  $Fa_{ij} \leftarrow direction_{ij} \times magnitude_{ij}$
  $Fa_i \leftarrow Fa_i + Fa_{ij}$ {*Accumulate individual forces.*}
$P_i \leftarrow P_i + Fa_i \times cte_a$
return P {*Return updated coordinates.*}

The detailed algorithm to modify the position of node i according to the attraction forces of its neighbors is shown in Algorithm 11. The resultant force is accumulated in Fa_i. For each neighbor, their displacement vector δ_{ij} is computed. To form the attraction force Fa_{ij}, the direction is obtained by normalizing the displacement vector. Following the cited methods [25] [84], the magnitude is proportional to both the square of the norm of the displacement $\|\delta_{i,j}\|^2$ and the weight of the edge between the



(a) Move towards less overlap. (b) Scan in grid around macro.

Figure 6.6: Overlap avoidance force.

nodes w_{ij}, and decreases with the ideal spring length constant $k = \sqrt{1/N}$. Finally, the forces are accumulated in Fa_i. At the end of the process, the position of the node P_i moves according to Fa_i multiplied by a constant user parameter cte_a which governs the overall impact of the attraction forces.

Overlap avoidance forces computation.

The motivation of overlap avoidance forces is to attract macros to positions where they overlap less with other macros. The basic idea is that, for each overlap, neighboring locations are scanned and its overlap with other macros is computed. The overlap avoidance force moves the macro in the direction of the position with least overlap and closest to its own location. If the macro has no overlaps, it has no motivation to move. The algorithm is essentially quadratic (worst case compares overlap of each pair of macros), but is feasible in reasonable time given that it is only applied to macro objects (which are in the several hundreds at most), and only considers a restricted local area. The idea is illustrated in Fig. 6.6a. The black boxes represent macros (with some overlaps), and the process focuses on the one with wider border. Several candidate locations are considered (1 to 4, in red), and their amount of overlap against other macros is computed. Finally the F_{oa} force goes towards 1, which has no overlap with other macros and is closer than 3.

In order to find candidate positions to reduce overlap, a grid scan is executed centered at the current macro location. Given a *window_size* and *grid_unit* parameter, the considered positions are a quadratic search around the macro, with x position from $x - window\_size$ to $x + window\_size$, step *grid_unit*, and simultaneously $y - window\_size$ to $y + window\_size$,

step *grid_unit*. An example is shown in Fig. 6.6b with some $grid\_unit = g$ and $window\_size = 1$. In practice, *grid_unit* is a fraction (tuned by user parameter *grid_size_factor*) of the area of the smallest macro, ignoring aspect ratios,

$$\min_{i \in macros} \sqrt{\frac{w_i \times h_i}{grid\_size\_factor}}$$

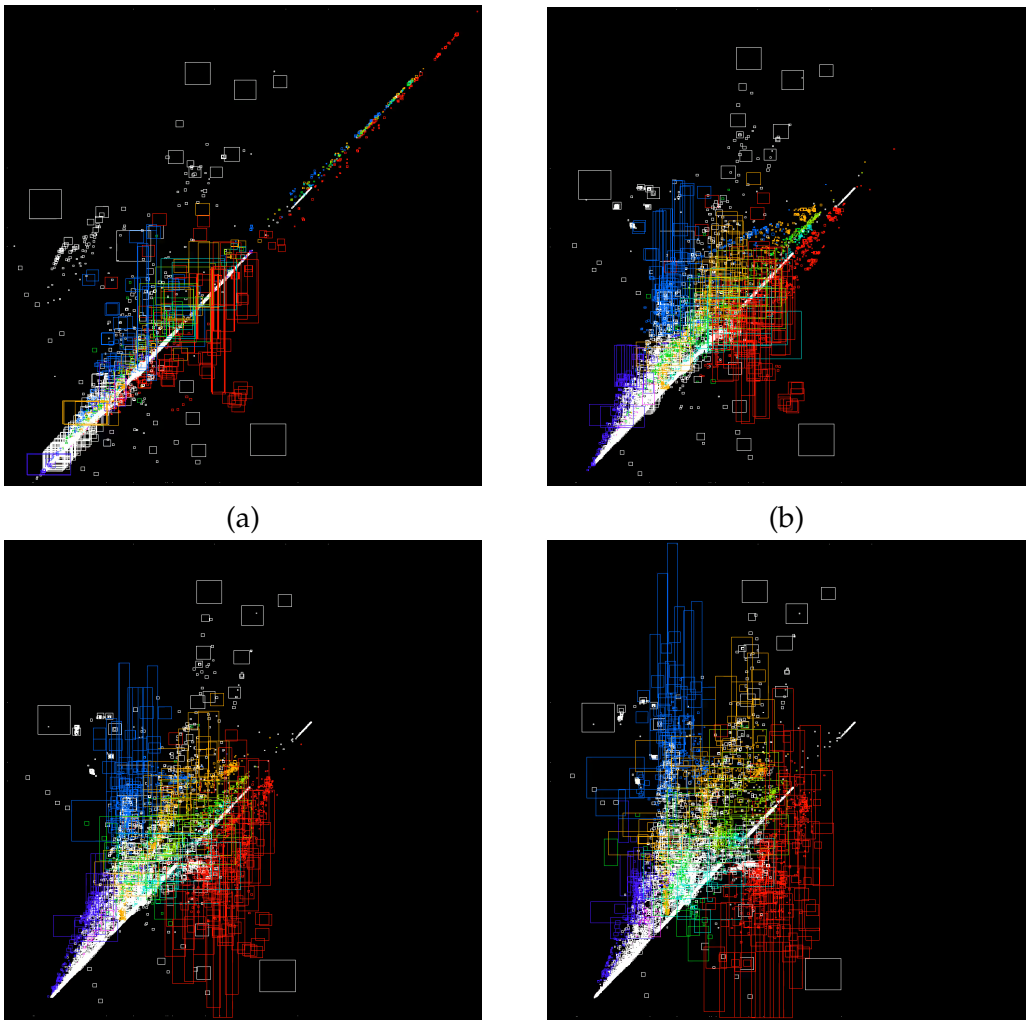
Additionally, to make the process happen smoothly, macro size scales as iteration count advances, from points at $iter = 1$ to their full size at $iter = max\_iters$, and overlapping with canvas borders or outside implies infinite overlap.

The computation of the overlap avoidance force Fao_i for a given macro i is illustrated in Algorithm 12. The input to the process is the node i, the positions P and sizes S of all components, the current iteration count *iter* to control macro scaling and the parameters *grid_unit*, *window_size* to control the grid exploration. First, to increase efficiency, the set of macros that overlap with the whole grid exploration zone is computed in *conflicting_macros*. The overlap function computes the amount of overlapping area, considering a rectangle or the point, size and scale of a macro. Then the grid scan begins, and for each candidate position, the sum of its overlap with conflicting macros (and block borders) is computed. If it is less than the minimum found, or equal but its position is closer to the current position, the best point is updated. Finally, a force from the current point to the best position is applied using the methods shown for attraction forces. The impact of the overlap avoidance forces is controlled by cte_{oa}, which can be several orders of magnitude bigger than cte_a to compensate the fact that there is no weight coming from the graph multiplying its magnitude.

An example of the evolution of the execution of the force-directed algorithm is shown in Fig. 6.7. Rectangles represent macro and registers, and the color depend of their top hierarchical block. At the beginning (Fig. 6.7a) the positions come from spectral placement (which had certain spreading properties). When forces begin to act and macros are still small, the design tends to contract (Fig. 6.7b). Later by Fig. 6.7c, hierarchical groups have more or less gotten together and begin to spread thanks to the effect of macros growing and trying to avoid overlap. By the end of the iteration count (Fig. 6.7d), the design is much more spread and macros are near their ideal positions (if they were points) according to the connections modeled in the sequential graph. Notice the final goal of our force-directed placement is to obtain preferred macro locations for

Algorithm 12 Force-directed sequential placement, overlap avoid forces.

Input: *Node* (i), *positions* (P), *sizes* (S), *algorithm parameters* ($iters, grid\_unit, window\_size$), *force parameters* ($cte_{oa}$)
Output: *Positions* (P)
$scale \leftarrow iter / max\_iters$
$exploration\_area \leftarrow find\_exploration\_area(P_i, grid\_unit, window\_size)$
$conflicting\_macros \leftarrow \emptyset$ {*Prune for efficiency.*}
for $j \in macro\_nodes$ **do**
  if $overlap(exploration\_area, P_j, S_j, scale)$ **then** $conflicting\_macros.insert(j)$
$best\_point = (0,0)$ {*Keep best candidate.*}
$min\_overlap \leftarrow \infty$
$associated\_distance \leftarrow \infty$
for $x \leftarrow -window\_size$ **to** $window\_size$ **do**
  for $y \leftarrow -window\_size$ **to** $window\_size$ **do**
    $p' \leftarrow p_i + (x \times grid\_unit, y \times grid\_unit)$ {*Grid scan.*}
    $sum\_overlap \leftarrow 0$
    for $j \in conflicting\_macros$ **do**
      $sum\_overlap \leftarrow overlap(p', S_i, P_j, S_j, scale)$
    $dist \leftarrow distance(P_i, p')$
    if $sum\_overlap < min\_overlap \vee (sum\_overlap = min\_overlap \wedge dist < associated\_distance)$ **then**
      $best\_point \leftarrow p'$
      $min\_overlap \leftarrow sum\_overlap$
      $associated\_distance \leftarrow distance(P_i, p')$
$\delta \leftarrow P_i - p'$ {*Apply overlap avoidance force.*}
$direction \leftarrow \frac{\delta}{\|\delta\|}$
$magnitude \leftarrow \frac{\|\delta\|^2}{k}$
$Fao_i \leftarrow direction \times magnitude$
$P_i = P_i + Fao_i \times cte_{ao}$
return P {*Return updated coordinates.*}



(a) (b)

(c) (d)

Figure 6.7: Evolution of the forces-directed process.

Table 6.1: Benchmark cell and macro count.

Bench.	Cells	Macros	Bench.	Cells	Macros
c1	520×10^3	32	*c5*	1.39×10^6	133
c2	3.95×10^6	100	*c6*	2.87×10^6	90
c3	3.78×10^6	94	*c7*	1.67×10^6	108
c4	4.81×10^6	122	*c8*	2.20×10^6	37

the floorplanning algorithm of HiDaP: it is not to distribute the objects perfectly evenly and with zero overlap. The runtime for the biggest circuits is of about 5 minutes.

6.5 Experimental Results

The previous chapter presented the initial results on the work on macro placement, showing promise in the method to produce layouts with good wirelength and timing properties which could help to reduce turn-around time. This section shows more in depth experimental results, and its aim is to answer the following questions:

- Do the spectral/force-directed methods help to obtain better macro layouts?
  - Subsections 6.5.1 to 6.5.3.
- What is the effect of individual features and parameters?
  - Subsections 6.5.2 to 6.5.5.
- Does the adaptive multiparameter approach help minimize distance to engineer-provided desired locations?
  - Subsection 6.5.6.
- What quality do our layouts show after routing? Can they be brought to timing-closure with close to no DRC violations?
  - Subsection 6.5.7 to 6.5.8.

As exposed in Chapter 5, array information for dataflow analysis is essential to our approach but is not available in open benchmarks available to academia such as the ICCAD'12 benchmarks [34]. The same set

of 8 real industrial examples of challenging circuits are used to validate our approach (see Table 6.1). This time the comparison is done against the industrial floorplanning tool, tape-out ready final handcrafted layouts and the previously reported results. The following four floorplan flows are compared:

Industrial EDA (IndEDA) Floorplan obtained with a state-of-the-art industrial tool using high effort settings.

Handcrafted floorplan (handFP) Floorplan manually obtained by expert backend engineers at the company.

DATE results (DATE) Results reported in Chapter 5 [80]. Best of three generated floorplans.

Our results (HiDaP) Results obtained using the approaches presented in this chapter.

Unless stated otherwise, metrics are taken after placement of standard cells using the same tool as **IndEDA**. Wirelength and total negative slack averages are shown using the geometric mean to reduce sensitivity to extreme values. The more positive the and total negative slack metrics are, the better (0 means timing closed). To understand the behaviour of our tool and given the stochastic nature of its optimization algorithm, the **HiDaP** flow is normally run 5 times, and the presented results are their average or their best, as indicated.

6.5.1 Results After Placement

The plots in Fig. 6.8 compare the average results across our whole benchmark. Wirelength is shown in meters, the is given in positive and in percentage of the clock period, and TNS is shown in positive and in ns (less is better). The reference values include the manual floorplans (**handFP**), industrial (**IndEDA**) and results from Chapter 5 (**DATE**) to the left.

Spectral average (HiDaP-SA) Using spectral methods on G_{df} and keeping the average of 5.

Spectral minimum (HiDaP-SM) Using spectral methods on G_{df} and keeping the minimum of 5.

Forces average (HiDaP-FA) Using force-directed methods on G_{seq} and keeping the average of 5.

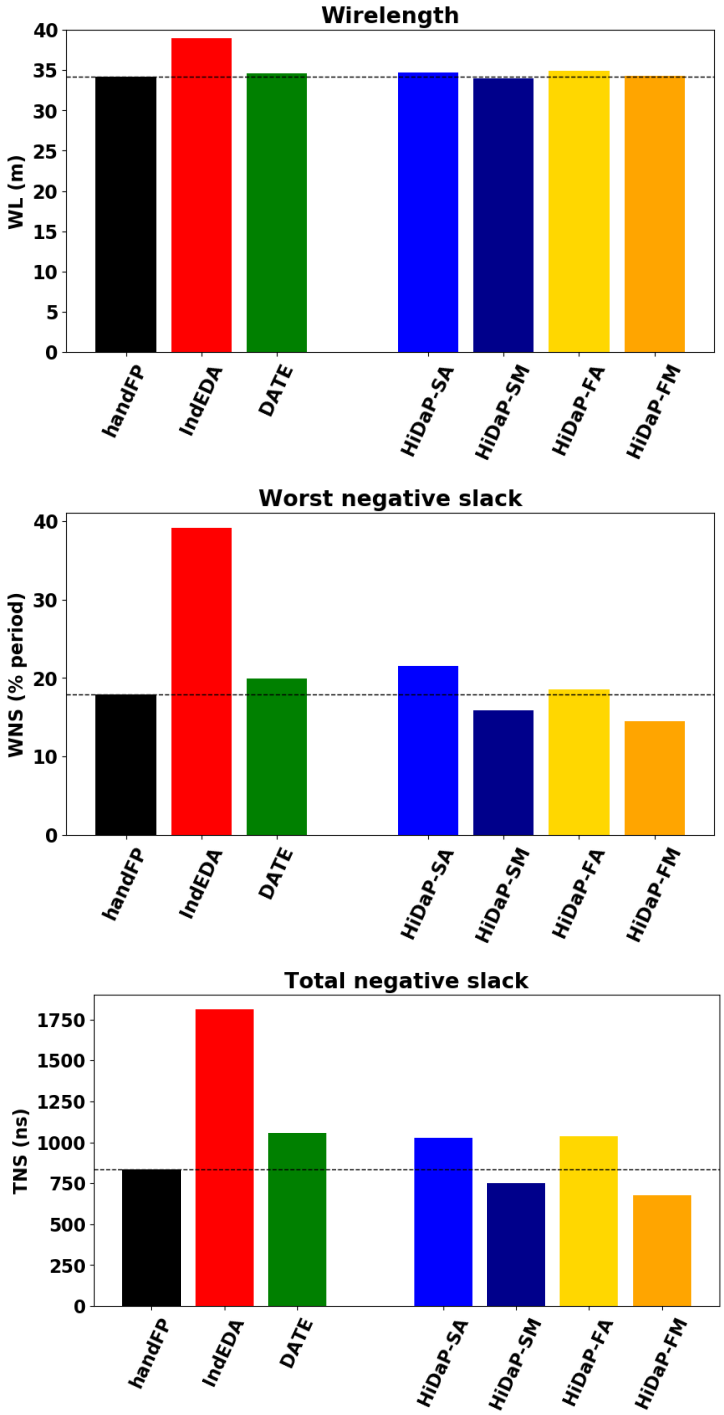


Figure 6.8: Comparison of HiDaP to other approaches. To the left, using layouts obtained manually (**handFP**), using industrial EDA tool (**IndEDA**) and presented in the previous chapter (**DATE**). To the right, our approach using spectral methods, average of 5 (**HiDaP-SA**) and minimum of 5 (**HiDaP-SM**), and using forces methods, average of 5 (**HiDaP-FA**) and minimum of 5 (**HiDaP-FM**).

Table 6.2: Results comparison with **HiDaP**-SA, **HiDaP**-SM.

	Wirelength		Cong.	Timing			
Flow	WL	Norm.	%GRC	WNS	Incr.	TNS	Norm.
handFP	34.13	1.000	3.87	-17.96	0%	-835	1.00
IndEDA	39.02	1.143	6.86	-39.07	21.1%	-1810	2.17
DATE	34.56	1.012	4.99	-19.93	2.0%	-1059	1.27
HiDaP-SA	34.67	1.016	4.78	-21.48	3.5%	-1029	1.23
HiDaP-SM	34.01	0.996	3.92	-15.93	-2.0%	-750	0.90

Forces minimum (HiDaP-FM) Using force-directed methods on G_{seq} and keeping the minimum of 5.

When comparing the results of automated flows against the manual **handFP** flow, current state of the art tool **IndEDA** shows the least promising results, with remarkably worse results in all areas (up to 14% WL increase, and near double congestion, WNS and TNS). The results reported in Chapter 5 are much closer to the ones from the handcrafted macro floorplans, obtaining slightly worse results in all areas. The WL when using spectral and force-directed methods remains very similar to the manual results and obtained with **DATE** in Chapter 5. However from a timing perspective, using spectral and force-directed methods clearly gives better results, obtaining similar results to those of **DATE** when taking the average of 5 (**HiDaP**-SA, **HiDaP**-FA), and obtaining lower TNS and WNS when taking the min of 5 runs per circuit (**HiDaP**-SM, **HiDaP**-FM).

Table 6.2 shows concrete numbers on an example of our basic experiment for a particular configuration of the tool using spectral methods (corresponding to columns **HiDaP**-SA/**HiDaP**-SM in the previous plots). Rows represent average results of each floorplan flow across all circuits, and columns give information on wirelength (in meters, and normalized with respect to **handFP**), congestion (global routing overflow percentage) and timing information. The is given in percentage of the clock period, with its increase over the handcrafted result. Its TNS is shown in ns and normalized. Numbers with the previous approaches (**IndEDA**, **handFP**, **DATE**) are reported for one floorplan, and for the current approach, the average (**HiDaP**-SA) and minimum results (**HiDaP**-SM) over the 5 generated macro placements are presented. The spectral flow on average across 5 floorplans, **HiDaP**-SA, manages to obtain similar results to Chapter 5, which took the best of 3 floorplans. When taking the best results according to the approach presented in this paper (**HiDaP**-SM), the results

show better WL (-0.4%) when compared to the manual flow **handFP**, along with a noticeable improvement in WNS (2% of clock period and 10% of the TNS), while reporting only a slight increase in congestion.

The detailed results for each of the 5 flows seen on Table 6.2 are presented in Table 6.3. When comparing the Chapter 5 results to the current, although the circuits where it did a good job got similar results (*c3, c8*), significant breakthroughs were observed in other circuits like *c7*. One the other hand, circuits like *c6* remain challenging from a timing perspective at this stage. This experiment suggests that the results from our approach **HiDaP** at placement prove to be generally competitive to manually obtained floorplans from **handFP**. They can be obtained at a fraction of the time and in parallel, creating opportunities for engineers to use our tool to prototype and advance promising macro placements through the physical design flow.

6.5.2 Parameter Exploration

The variety of algorithm recipes and parameters, in combination with each particular design, makes it difficult to establish the default running mode of the tool. Table 6.4 is an example of the experiments that have been run to determine them, and it explores the impact of the use of spectral and force-directed methods to prototype block desired coordinates before the simulated annealing stage and the use of macro affinity dataflow (λ). Experiments were run 5 times on each circuit for 8 tool configurations, with $\lambda = \{1, 0.8\}$ (not using and using macro dataflow) and turning spectral and force-directed methods on and off. The multiobjective alpha weight are, for $\alpha_1, \alpha_2, \alpha_3$, 0.4, 0.4, 0.2 (for spectral experiments) and 0.7, 0.1, 0.2 (for forces experiments), with $\gamma = \infty$ (always keep best solution with best dataflow). This generated a total of 40 floorplans per benchmark, which were brought to placement and evaluated. The results of **handFP**, **IndEDA** and **DATE** are also shown for reference.

When comparing the use of spectral and forces versus not using them, the results show that on the average of 5 cases they do not seem to help much in WL, but it is always desirable to use them from a timing point of view. For example with $\lambda = 1$, average TNS results are better than the ones obtained for **DATE** (which was a best of 3), and the average WNS when using forces is only 0.6% of period worse than with the manual tape-out layouts **handFP**. When looking at the best of 5 results, timing is almost always better than the reported for **handFP** with similar WL quality. The only case in which using spectral/forces does not seem to

Table 6.3: Metrics after placement for a configuration.

Case	Wirelength		Cong.	Timing			
c1	WL	Norm.	%GRC	WNS	Incr.	TNS	Norm.
handFP	12.81	1.000	7.36	-0.2	0%	-0.01	1.00
IndEDA	13.19	1.029	6.51	0.0	-0.2%	-0.01	1.00
DATE	13.40	1.046	7.83	0.3	-0.5%	0	0.00
HiDaP-SA	13.93	1.088	9.62	0.23	-0.4%	0	0.00
HiDaP-SM	13.66	1.067	8.06	0.79	-1.0%	0	0.00
c2	WL	Norm.	%GRC	WNS	Incr.	TNS	Norm.
handFP	38.97	1.000	9.33	-11.24	0%	-213	1.00
IndEDA	46.01	1.181	12.99	-44.53	33.28%	-932	4.37
DATE	40.72	1.045	13.00	-19.00	7.76%	-329	1.55
HiDaP-SA	40.94	1.050	14.40	-22.13	10.89%	-281	1.32
HiDaP-SM	40.29	1.034	13.71	-16.63	5.38%	-213	0.99
c3	WL	Norm.	%GRC	WNS	Incr.	TNS	Norm.
handFP	38.15	1.000	9.15	-17.78	0%	-317	1.00
IndEDA	44.83	1.175	10.09	-75.52	57.7%	-553	1.74
DATE	35.02	0.918	8.29	-17.50	-0.3%	-260	0.82
HiDaP-SA	36.01	0.944	8.70	-19.76	2.0%	-350	1.103
HiDaP-SM	35.56	0.932	7.10	-16.78	-1.0%	-257	0.81
c4	WL	Norm.	%GRC	WNS	Incr.	TNS	Norm.
handFP	38.34	1.000	3.33	-22.81	0%	-1736	1.00
IndEDA	45.03	1.174	7.24	-54.43	31.6%	-2167	1.25
DATE	40.43	1.054	4.94	-31.20	8.4%	-2686	1.55
HiDaP-SA	39.59	1.032	4.76	-26.21	3.4%	-2755	1.59
HiDaP-SM	39.24	1.023	4.21	-21.20	-1.6%	-1956	1.13
c5	WL	Norm.	%GRC	WNS	Incr.	TNS	Norm.
handFP	38.06	1.000	3.42	-39.82	0%	-1017	1.00
IndEDA	44.25	1.162	2.02	-30.80	-9.0%	-1940	1.90
DATE	39.51	1.038	4.72	-25.10	-14.7%	-1149	1.13
HiDaP-SA	39.49	1.037	3.95	-22.79	-17.0%	-1357	1.33
HiDaP-SM	38.03	0.999	3.20	-18.50	-21.3%	-858	0.84
c6	WL	Norm.	%GRC	WNS	Incr.	TNS	Norm.
handFP	74.86	1.000	1.63	-27.28	0%	-3688	1.00
IndEDA	96.42	1.288	9.95	-68.87	41.7%	-15341	4.16
DATE	79.20	1.058	2.22	-37.00	9.7%	-5051	1.37
HiDaP-SA	77.72	1.038	2.56	-61.60	34.3%	-6400	1.74
HiDaP-SM	75.54	1.009	2.17	-41.72	14.4%	-5448	1.48
c7	WL	Norm.	%GRC	WNS	Incr.	TNS	Norm.
handFP	35.29	1.000	4.61	-20.44	0%	-774	1.00
IndEDA	41.44	1.174	38.56	-34.89	14.4%	-1060	1.37
DATE	35.52	1.007	6.47	-29.90	9.4%	-1059	1.37
HiDaP-SA	34.79	0.986	4.48	-17.63	-2.8%	-505	0.65
HiDaP-SM	33.85	0.959	3.63	-13.06	-7.4%	-358	0.46
c8	WL	Norm.	%GRC	WNS	Incr.	TNS	Norm.
handFP	25.17	1.000	0.93	-4.17	0%	-24	1.00
IndEDA	24.85	0.987	1.02	-3.42	-0.8%	-44	1.80
DATE	23.75	0.944	1.37	0.0	-4.2%	0	0.00
HiDaP-SA	24.26	0.964	1.04	-1.96	-2.2%	-23	0.9
HiDaP-SM	23.95	0.951	0.67	-0.35	-3.8%	0	0.00

Table 6.4: Parameter exploration.

		Best of 5			Average of 5		
Flow	Mode	WL	TNS	WNS	WL	TNS	WNS
handFP	-	34.14	-835	-18.0%	only one run		
IndEDA	-	39.03	-1810	-39.2%	only one run		
DATE	-	34.57	-1059	-19.9%	only one run		
HiDaP	No Forces	34.07	-797	-14.8%	34.97	-1441	-22.0%
$\lambda = 1$	Forces	34.32	-678	-14.5%	34.91	-1039	-18.6%
HiDaP	No Forces	34.24	-882	-16.3%	35.00	-1422	-25.4%
$\lambda = 0.8$	Forces	34.48	-780	-15.0%	35.16	-1319	-23.4%
HiDaP	No Spectral	34.19	-670	-13.0%	34.86	-1267	-22.0%
$\lambda = 1$	Spectral	34.01	-750	-15.9%	34.68	-1029	-21.5%
HiDaP	No Spectral	34.26	-850	-16.9%	35.38	-1326	-21.5%
$\lambda = 0.8$	Spectral	34.30	-740	-15.2%	35.00	-1262	-20.9%

Table 6.5: Results with best layouts after placement.

	Wirelength		Cong.	Timing			
Flow	WL	Norm.	%GRC	WNS	Incr.	TNS	Norm.
handFP	34.13	1.000	3.87	-17.96	0%	-835	1.00
IndEDA	39.02	1.143	6.86	-39.07	21.1%	-1810	2.17
DATE	34.56	1.012	4.99	-19.93	2.0%	-1059	1.27
HiDaP-WL	33.52	0.981	3.53	-13.40	-4.6%	-625	0.75
HiDaP-TNS	34.10	0.998	4.06	-10.29	-7.7%	-463	0.56

improve is in best results for $\lambda = 1$, but this does not translate to the average experimental results. In general, it can also be said that the results for $\lambda = 1$ are better than $\lambda = 0.8$. This is not the case in all circuits, and reveals further investigation is required to understand how to best tune all parameters given a particular benchmark.

6.5.3 Best Layouts After Placement

Table 6.5 shows the average metrics picking the best layout per circuit (among the 8 layouts that were found during the experiment in the previous section, the ones with best WL and TNS respectively), with a chart view in Fig. 6.9. Reference results are also shown for comparison. The WNS is given in positive and in percentage of the clock period, and TNS is shown in positive and in ns (less is better). When focusing on the best

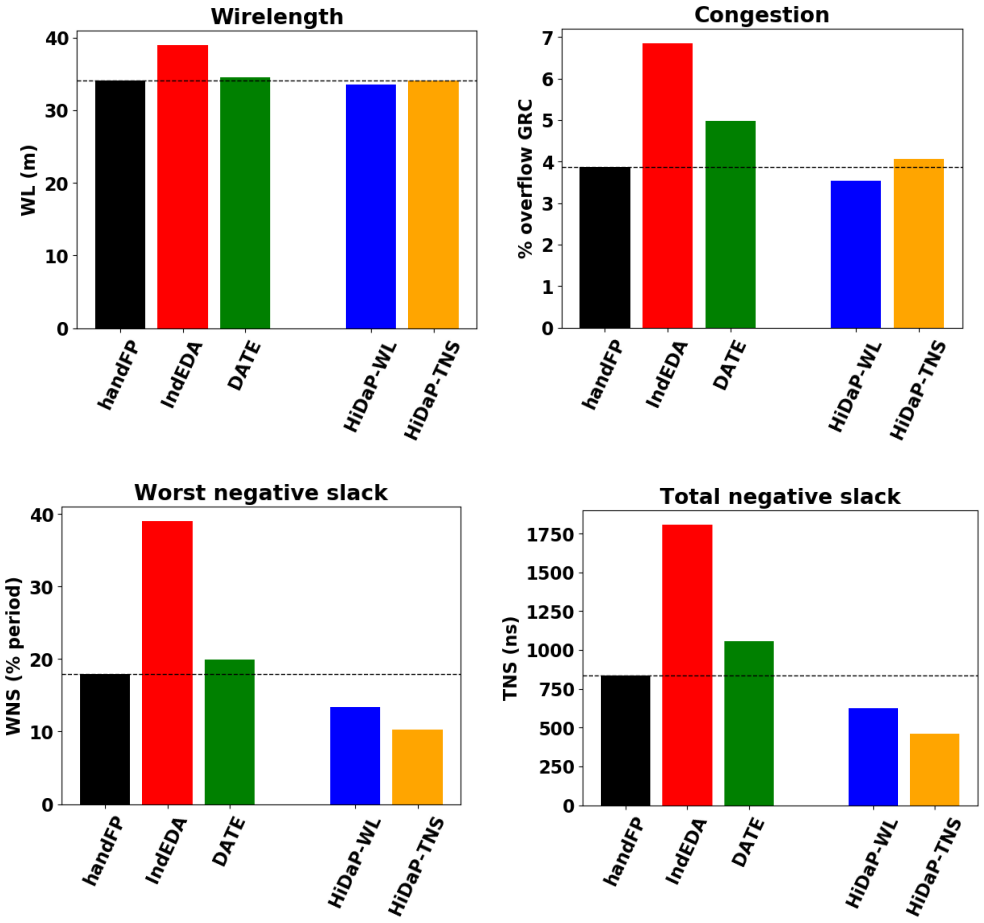


Figure 6.9: Placement results, charts of Table 6.5. To the left, using layouts obtained manually (**handFP**), using industrial EDA tool (**IndEDA**) and presented in the previous chapter (**DATE**). To the right, our approach keeping the best placement out of 40 in terms of wirelength (**HiDaP-WL**) and total negative slack (**HiDaP-TNS**).

Table 6.6: Detailed results with best TNS layouts.

Case	Wirelength		Cong.	Timing			
c1	WL	Norm.	%GRC	WNS	Incr.	TNS	Norm.
handFP	12.81	1.000	7.36	-0.2	0%	-0.01	1.00
HiDaP	13.40	1.046	9.99	0.3	-0.5%	0	0.00
c2	WL	Norm.	%GRC	WNS	Incr.	TNS	Norm.
handFP	38.97	1.000	9.33	-11.24	0%	-213	1.00
HiDaP	40.47	1.038	12.81	-15.23	3.4%	-170	0.79
c3	WL	Norm.	%GRC	WNS	Incr.	TNS	Norm.
handFP	38.15	1.000	9.15	-17.78	0%	-317	1.00
HiDaP	34.71	0.910	8.09	-9.67	-8.1%	-168	0.53
c4	WL	Norm.	%GRC	WNS	Incr.	TNS	Norm.
handFP	38.34	1.000	3.33	-22.81	0%	-1736	1.00
HiDaP	38.54	1.005	4.38	-17.34	-5.5%	-1725	0.99
c5	WL	Norm.	%GRC	WNS	Incr.	TNS	Norm.
handFP	38.06	1.000	3.42	-39.82	0%	-1017	1.00
HiDaP	39.18	1.029	4.36	-11.89	-27.9%	-633	0.63
c6	WL	Norm.	%GRC	WNS	Incr.	TNS	Norm.
handFP	74.86	1.000	1.63	-27.3	0%	-3688	1.00
HiDaP	74.56	0.996	1.36	-16.5	-10.73%	-1189	0.32
c7	WL	Norm.	%GRC	WNS	Incr.	TNS	Norm.
handFP	35.29	1.000	4.61	-20.44	0%	-774	1.00
HiDaP	35.24	0.999	3.07	-11.98	-8.5%	-267	0.35
c8	WL	Norm.	%GRC	WNS	Incr.	TNS	Norm.
handFP	25.17	1.000	0.93	-4.17	0%	-24	1.00
HiDaP	24.46	0.972	1.23	0.02	-4.2%	0	0.00

WL reduction, WL drops by a 2% while WNS is reduced in close to a 5% of cycle time and TNS is reduced on a 25% compared to **handFP**. When focusing on timing, WNS is reduced close to a 8% of cycle time when compared to the **handFP** results, and the TNS is reduced up to a 44%. On the other hand, congestion is kept at similar levels when compared to the manual designs. This numbers reveal the potential of the tool to find very good layouts in terms of quality at the price of (parallelizable) CPU time.

The detailed view for the results comparing the layouts with best TNS against the manual layouts **handFP** is shown in Table 6.6, and in chart form (including **IndEDA** results) in Fig. 6.10. When using the **HiDaP** flow, two of the circuits get fully timing closed (*c1*, *c8*). Whereas a very good candidate for the circuit with more TNS (*c6*) in terms of timing is found, the one with second worst TNS (*c4*) proves to be much harder for

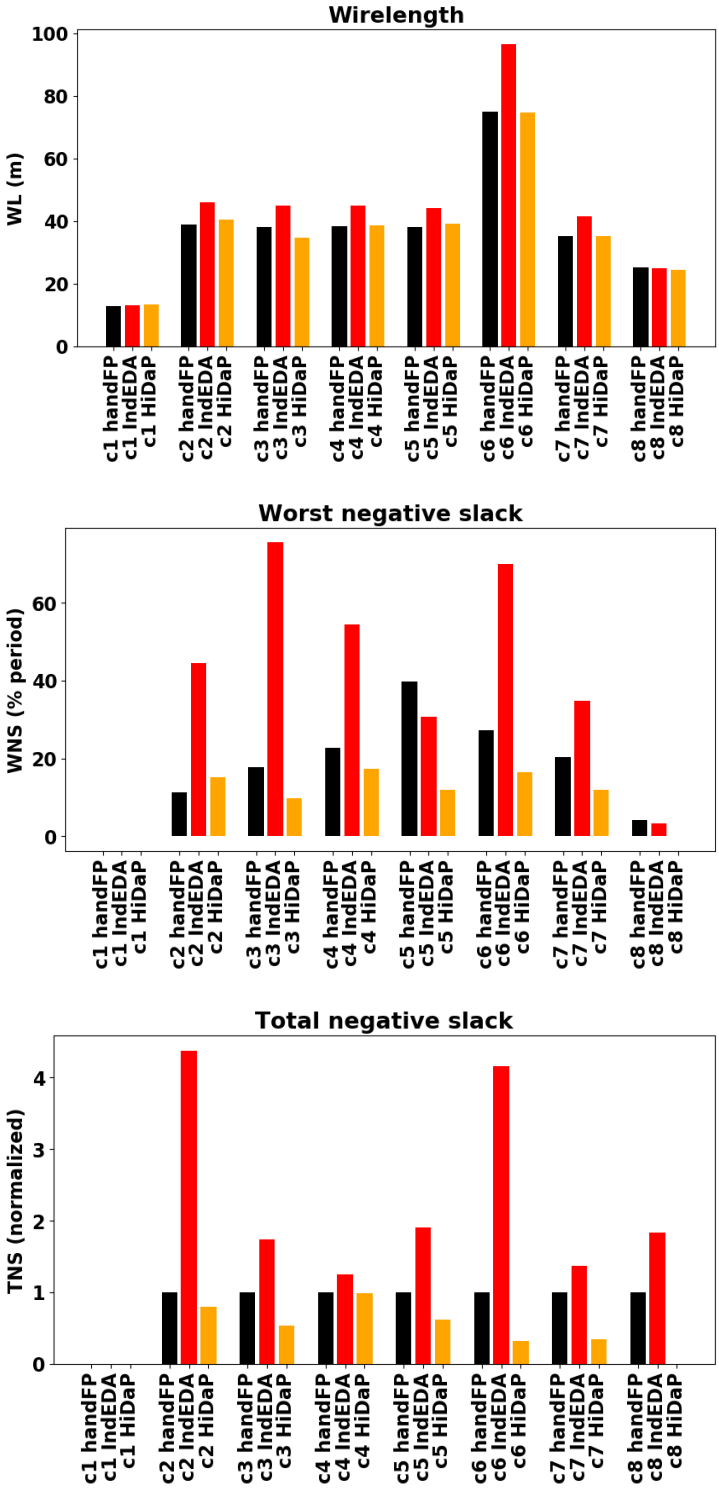


Figure 6.10: Placement results, charts of Table 6.6. Detailed results for each circuit, comparing the results after placement for several metrics of the manual layout (**handFP**), industrial EDA layout (**IndEDA**) and the best layout of our approach after the parameter exploration conducted in the previous section (**HiDaP**).

Table 6.7: Effect of latency decay factor.

		Wirelength		Timing			
Flow	k	WL	Norm.	WNS	Incr.	TNS	Norm.
handFP	-	34.13	1.000	-17.96	0%	-835	1.00
HiDaP	0	35.04	1.026	-28.22	10.26%	-1393	1.67
	1	34.57	1.013	-23.38	5.42%	-1254	1.50
	2	34.86	1.021	-21.95	3.99%	-1266	1.52

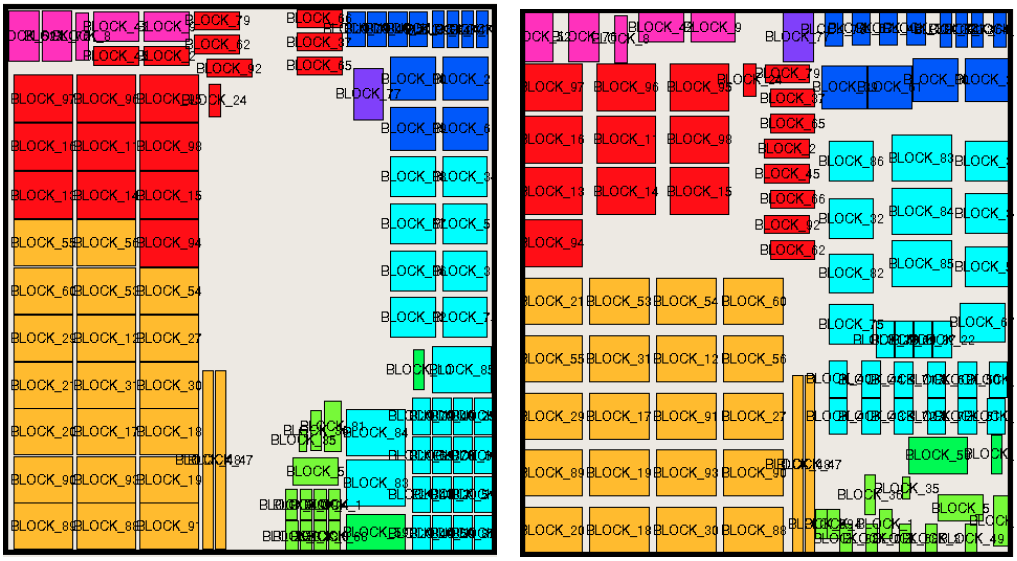
Table 6.8: Effect of macro orientation.

	Wirelength		Timing			
Flow	WL	Norm.	WNS	Incr.	TNS	Norm.
handFP	34.13	1.000	-17.96	0%	-835	1.00
HiDaP-D	35.29	1.034	-24.67	6.71%	-1405	1.68
HiDaP-C	35.13	1.029	-24.91	6.95%	-1486	1.77
HiDaP-F	34.99	1.025	-23.31	5.35%	-1382	1.65

HiDaP. Overall, the results indicate layouts with better properties than manually obtained layouts can be found when evaluated at the placement stage, enabling the tool to be successfully used in an exploratory manner for highly critical blocks. A picture of the HiDaP layouts presented in this table for each circuit can be found in Appendix 6.A at the end of this chapter.

6.5.4 Effect of Latency Awareness

The effect of the variation of the latency decay parameter k (introduced in Sect. 5.5.4), affecting the computation of the edges weights in the dataflow graph, is shown in Table 6.7. A value of $k = 0$ means considering all latencies to be 1, whereas $k = 2$ means considering them squared. All results are worse when ignoreing flop pipeline stages, specially WNS, which is a 10% worse than **handFP** in percentage of cycle time. When increasing k, the best point for WL and TNS seems to be at $k = 1$, although a significant further reduction in WNS can be achieved with $k = 2$. This reinforces the idea that considering flop latencies between components is a key element to achieving floorplans which can get timing closed.



(a) Preferred macro locations. (b) Constrained floorplan.

Figure 6.11: Guiding HiDaP using known macro locations.

Table 6.9: DATAFLOW and DISTANCE optimization tradeoff

	$\gamma = \infty$		$\gamma = 1$	
$\alpha_1, \alpha_2, \alpha_3$	(1)	(2)	(2)	(3)
DATAFLOW	1.00	1.03	1.30	1.38
DISTANCE	4.73	3.64	1.13	1.00

6.5.5 Effect of Macro Orientation

Table 6.8 studies the impact of macro orientation by comparing the results using three approaches: default orientation (**HiDaP**-D, all orientations set to *N*), having macro sides with most ports looking to the center (**HiDaP**-C, suggested in works such as [12]) and flipping SA (**HiDaP**-F, our approach). Compared with the default, **HiDaP**-C obtains slightly better WL, but worse WNS and TNS. In this case, our approach proves to beat the default in all three areas by reducing WL by a 1% and WNS by a 1.5% of cycle time while also maintaining lower TNS.

6.5.6 DATAFLOW-DISTANCE Tradeoff

When the engineer suggests preferred locations to the tool, it is often the case that dataflow reduction is negatively impacted in order to reduce distances to such locations. Table 6.9 shows the impact of each compo-

Table 6.10: Geo. mean over all circuits after routing. The first two rows average over the eight circuits, the next three over the only six that **IndEDA** managed to route.

	Wirelength		Legal?	Timing			
Flow	WL	Norm.	#DRCv	WNS	Norm.	TNS	Norm.
handFP	34.27	1.000	1447	-23.0	1.00	-298	1.00
HiDaP-M	34.71	1.013	1561	-16.5	0.71	-92	0.31
handFP	33.42	1.000	1546	-20.0	1.00	-73	1.00
IndEDA	38.16	1.142	55288	-19.5	0.98	-28	0.38
HiDaP-M	34.05	1.019	1720	-15.7	0.78	-21	0.30

nent of the cost function in average across all circuits, 5 runs per each, when using the **handFP** solution to guide the search of the **HiDaP** flow. Columns represent 4 parameters recipes. Recall when $\gamma = \infty$, the result with lowest DATAFLOW is kept, and when $\gamma = 1$, the result with lowest DISTANCE is kept. The alphas are, for $\alpha_1, \alpha_2, \alpha_3$, $(1) = 0.8, 0.0, 0.2$ (optimize only dataflow and overlap), $(2) = 0.4, 0.4, 0.2$ (optimize all) and $(3) = 0.0, 0.8, 0.2$ (optimize only distances and overlap).

When both the optimization effort and best solution are set to minimize dataflow ($\gamma = \infty$, (1)), the distance to desired positions is 4.73 times the minimum distance possible. When minimizing distances to positions is prioritized, the DATAFLOW component is 38% higher. This trade-off does not always ocur: in *c5*, both components can be simultaneously minimized as the handmade placement also minimized dataflow affinity. Fig. 6.11 shows the effect of giving macro location preferences visually: the layout to the left is the **handFP** layout for *c2*, whereas the layout to the right is the one obtained by **HiDaP** when given the **handFP** layout as a preferred locations and enforcing distance minimization.

6.5.7 Results After Routing

All circuits used in the spectral flow for our experiment in Sect. 6.5.1 were also brought to routing in order to check the effects of congestion after placement and ensure layout quality was preserved. Table 6.10 shows a summary of the results, also in chart form in Fig. 6.12. For each one of the 5 layouts generated by our tool for each circuit, the reported results are shown for the best layout according to a team of physical designers, who have also done manual iterations to bring these 8 best layouts to virtual timing closure and DRC clearness.

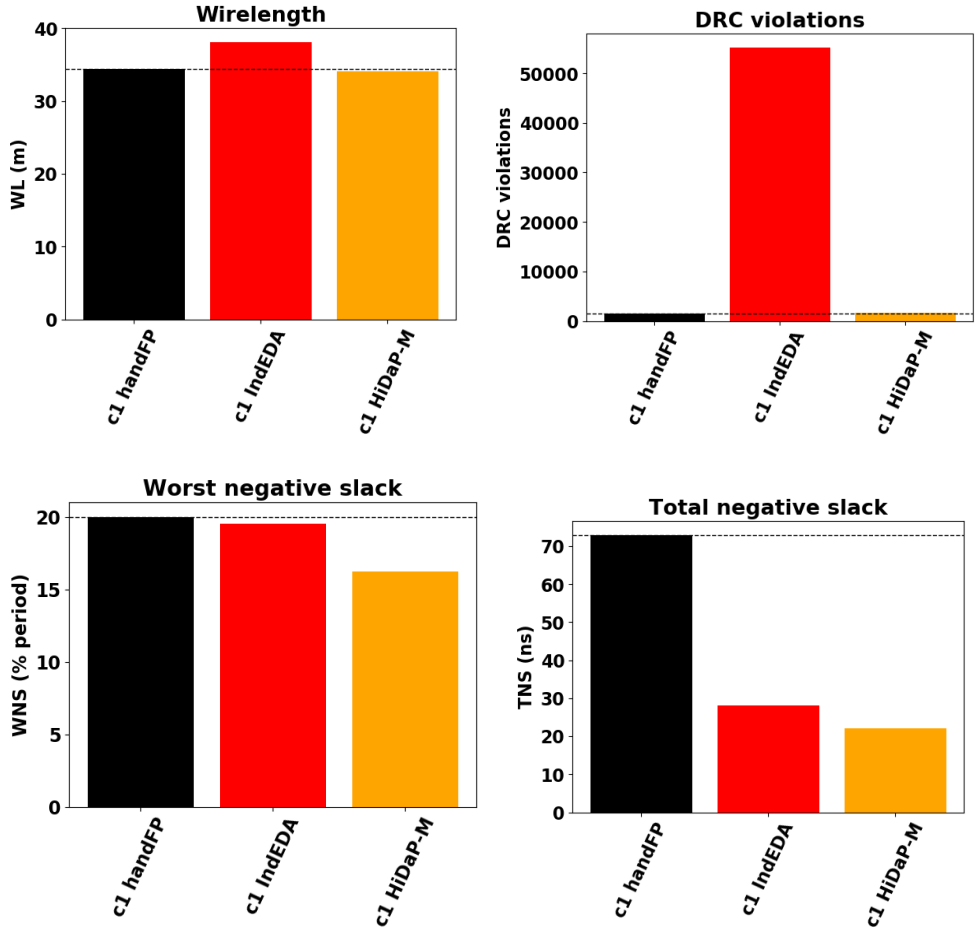


Figure 6.12: Routing results, charts of Table 6.10, showing the average results accross all circuits after routing, using layouts obtained manually (**handFP**), using industrial EDA tool (**IndEDA**) and the best of 5 with our method (**HiDaP-M**).

Each line in Table 6.10 represents the average of the best layouts for each circuit for a given flow. The first two lines average over all circuits, and the last three over the ones that **IndEDA** could finish routing of (6 circuits of 8, since it was no able to obtain routing results for *c4* and *c7* in reasonable time). The results show how WL reduction results observed in placement hold after routing, with timing showing even better reduction (-29% WNS, -70% TNS), while showing a similar number of DRC violations. When focusing only on the set of circuits that **IndEDA** was able to fully route, the results by the industrial tool confirm again the important gap in WL. The timing results **IndEDA** shows are better than the manual designs, but the layouts present an enormous number of DRC violations.

Table 6.11: Metrics after routing of layouts in Sec. 6.5.1.

Case	Wirelength		Legal?	Timing			
c1	WL	Norm.	#DRCv	WNS	Norm.	TNS	Norm.
handFP	12.97	1.000	4	0	0%	-0.1	1.00
IndEDA	14.98	1.155	1	-1.4	1.4%	-0.2	2.00
HiDaP-M	14.29	1.102	2	-0.8	0.8%	-0.1	1.00
c2	WL	Norm.	#DRCv	WNS	Incr.	TNS	Norm.
handFP	39.40	1.000	58	-51.7	0%	-1760	1.00
IndEDA	45.12	1.145	$> 10^5$	-7.8	-43.9%	-4	0.00
HiDaP-M	41.14	1.044	14	-4.9	-46.8%	-3	0.00
c3	WL	Norm.	#DRCv	WNS	Incr.	TNS	Norm.
handFP	38.27	1.000	30	-5.8	0%	-8	1.00
IndEDA	42.02	1.098	$> 10^4$	-12.7	6.82%	-7	0.94
HiDaP-M	36.34	0.950	37	-15.0	9.2%	-13	1.66
c4	WL	Norm.	#DRCv	WNS	Incr.	TNS	Norm.
handFP	38.34	1.000	2253	-31.7	0%	-998	1.00
IndEDA	Could not finish routing						
HiDaP-M	38.70	1.009	2269	-24.6	-4.0%	-1259	1.26
c5	WL	Norm.	#DRCv	WNS	Incr.	TNS	Norm.
handFP	38.72	1.000	518	-10.0	0%	-48	1.00
IndEDA	44.87	1.159	$> 10^4$	-20.5	10.53%	-220	4.58
HiDaP-M	39.126	1.014	633	-12.4	2.4%	-28	0.58
c6	WL	Norm.	#DRCv	WNS	Incr.	TNS	Norm.
handFP	74.21	1.000	3169	-35.8	0%	-1978	1.00
IndEDA	99.14	1.336	$> 10^5$	-40.5	4.6%	-389	0.20
HiDaP-M	75.33	1.015	2869	-51.0	15.2%	-4409	2.23
c7	WL	Norm.	#DRCv	WNS	Incr.	TNS	Norm.
handFP	35.64	1.000	49	-32.5	0%	-215	1.00
IndEDA	Could not finish routing						
HiDaP-M	34.28	0.962	62	-12.7	-19.8%	-33	0.15
c8	WL	Norm.	#DRCv	WNS	Incr.	TNS	Norm.
handFP	24.79	1.000	5498	-16.4	0%	-774	1.00
IndEDA	24.85	0.987	5457	-34.1	17.8%	-1014	1.36
HiDaP-M	23.95	0.951	6338	-16.9	0.5%	-547	0.77

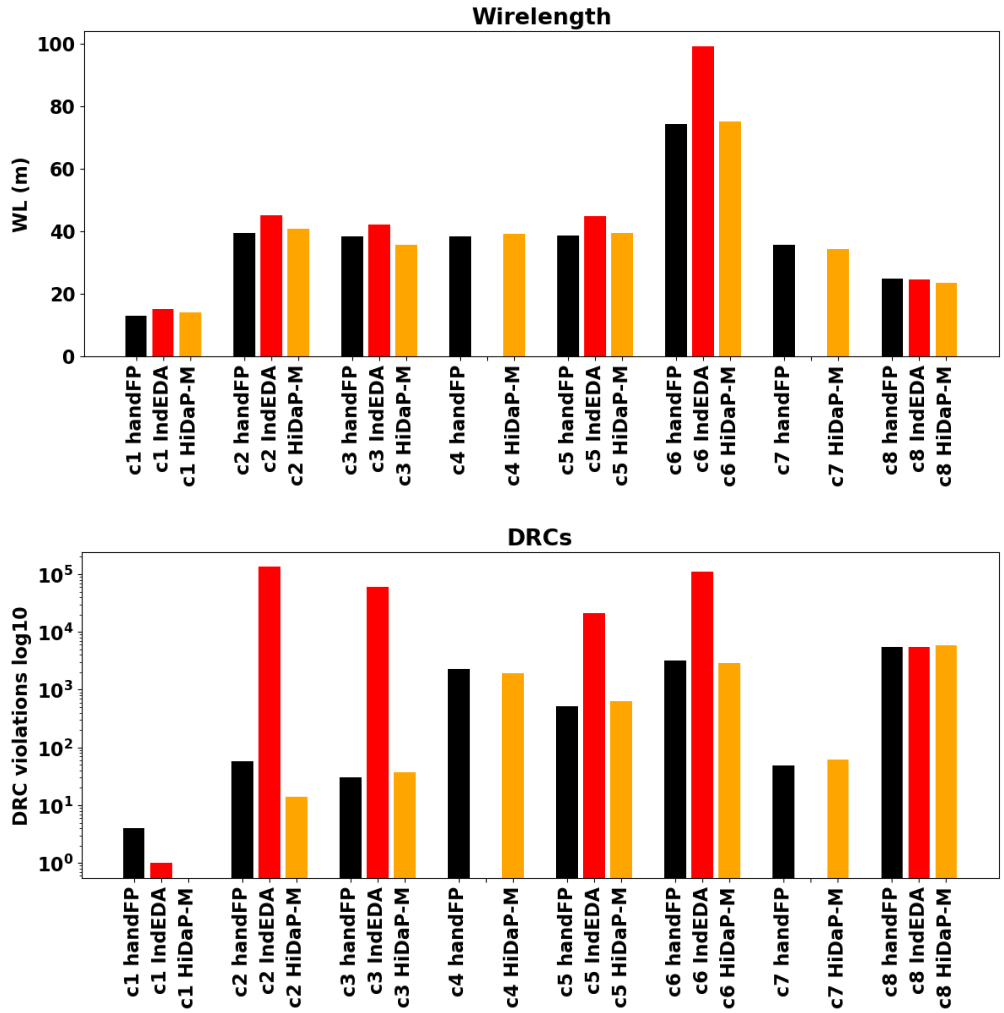


Figure 6.13: Routing WL and DRC results, charts of Table 6.11, comparing the manual layout (**handFP**), industrial EDA tool layout (**IndEDA**) and the best out of 5 for our tool (**HiDaP-M**) accross all 8 circuits in our benchmark.

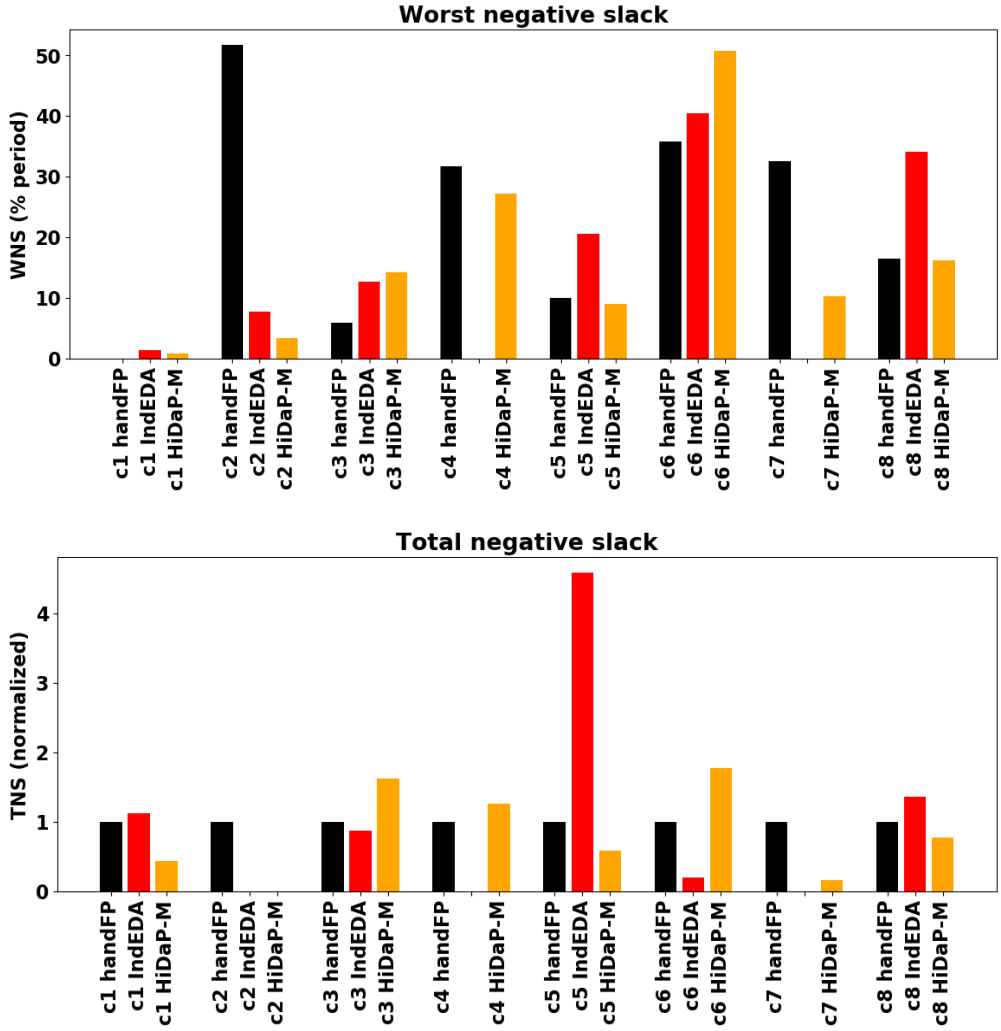


Figure 6.14: Routing WNS and TNS results, charts of Table 6.11, comparing the manual layout (**handFP**), industrial EDA tool layout (**IndEDA**) and the best out of 5 for our tool (**HiDaP-M**) accross all 8 circuits in our benchmark.

Table 6.12: Timing-closure results after manual layout modification.

Case	#FEP	WNS	TNS	DRC
c5	2632	-9.1%	-28.7	633
c5 + manual	112	-3.3%	-0.65	34
c7	1528	-12.5%	-32.84	62
c7 + manual	479	-2.5%	-1.54	22

Detailed results by circuit are shown in Table 6.11, and in chart form in Fig. 6.13 and Fig. 6.14. The only circuits where the **IndEDA** macro placements managed to finish with reasonable DRC violations are *c1* and *c8*, which are the ones with less macros (32, 37 respectively). Our approach **HiDaP-M** obtained results comparable to **handFP** in all circuits, specially in *c2*, where it nearly close timing, and *c7*, with a reduction of close to 4% in WL, -20% of clock cycle in WNS and 85% of the TNS. On the other hand, the manual floorplans still achieved better results in circuits *c3* and *c6*, specially considering timing results.

6.5.8 Post-Routing Timing Closure

In order to check if the floorplans can serve as initial proposals for industrial block design, a team of engineers took the **HiDaP-M** layouts at routing stage and tried to close timing without DRC violations. They pushed the macro placements through the full production physical design flow. Four of the circuits (*c1, c3, c4, c7*) were practically brought to sign-off (residual WNS/TNS and DRC violations) with little macro adjustment (aligning and small moves): that is, at most one iteration of macro refinement. The other circuit were also brought to completion after 2 macro refinement iterations.

Table 6.12 shows detailed results for the process in *c5* and *c7*. Whereas the *c5* and *c7* rows show the values for the automated layouts after passing the whole physical design process, the same layouts after being manually modified by our engineers are respectively *c5 + manual* and *c7 + manual*. Columns represent number of failing endpoints #FEP, WNS (% period), TNS (ns) and number of design rule check violations. Notice how the results after small manual modification are almost timing closed and DRC clear.

Fig. 6.15 shows the automatic macro placement generated by **HiDaP** of the *c5* circuit after routing, and Fig. 6.16 after manual modifications, which in this case include rearranging the macros to the left of the central

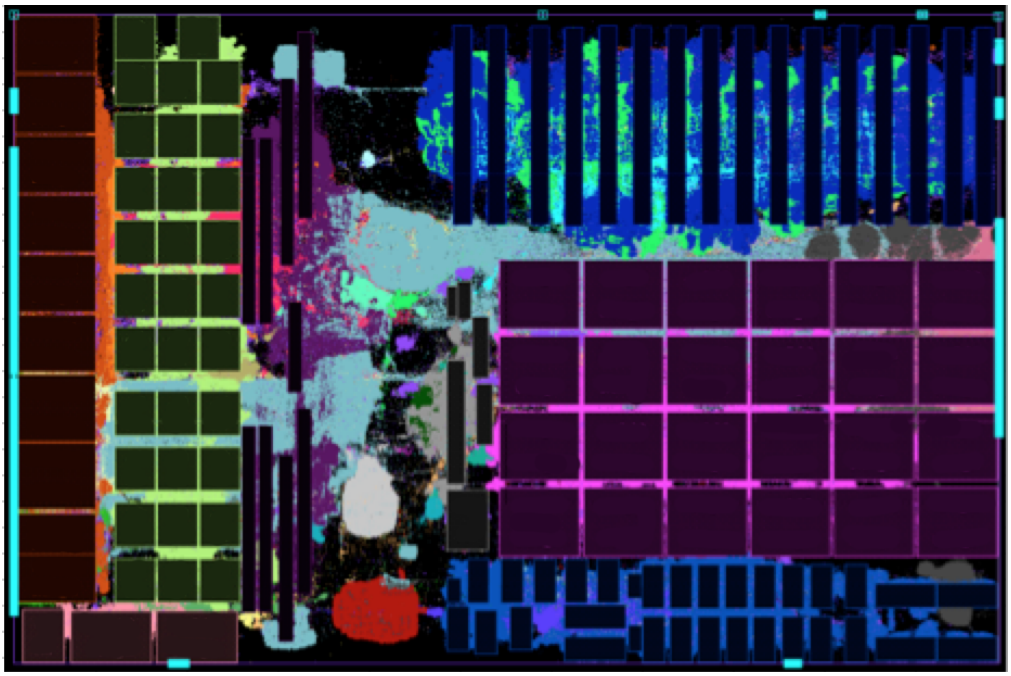


Figure 6.15: *c5* automatic macro placement.

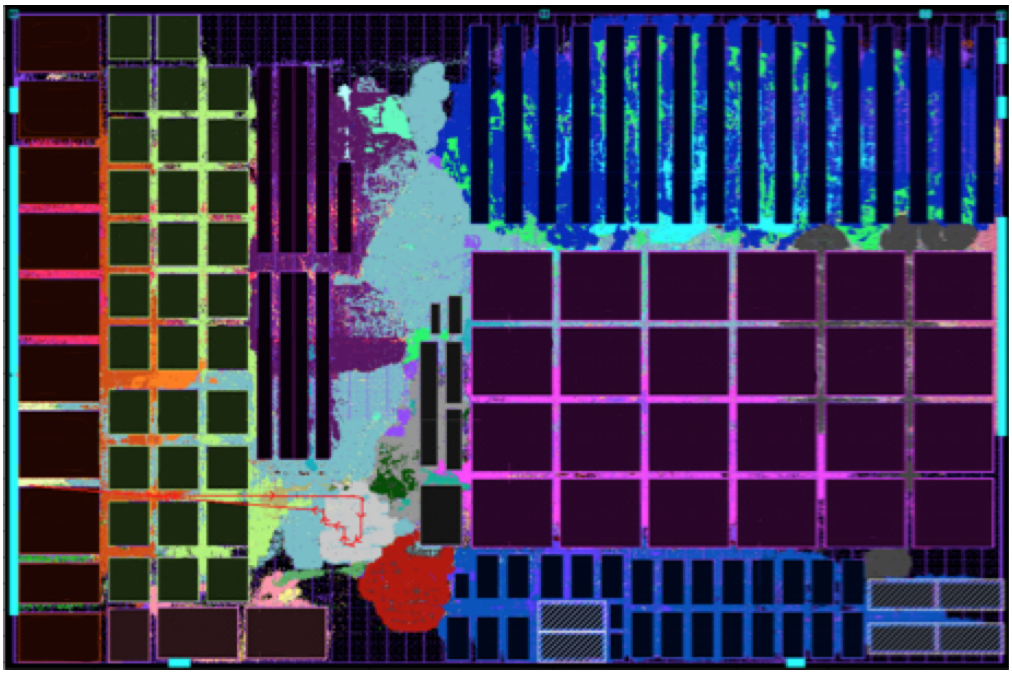


Figure 6.16: *c5* after light manual editing.

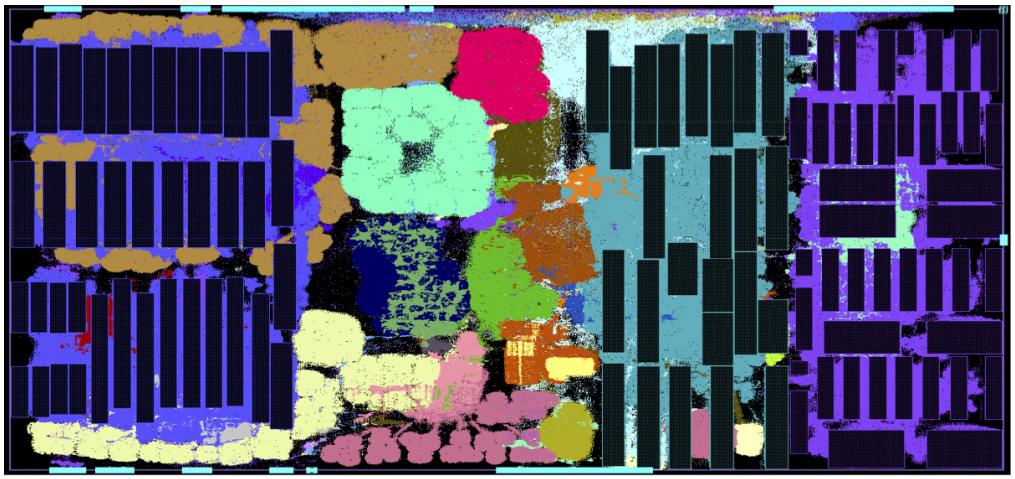


Figure 6.17: *c7* automatic macro placement.

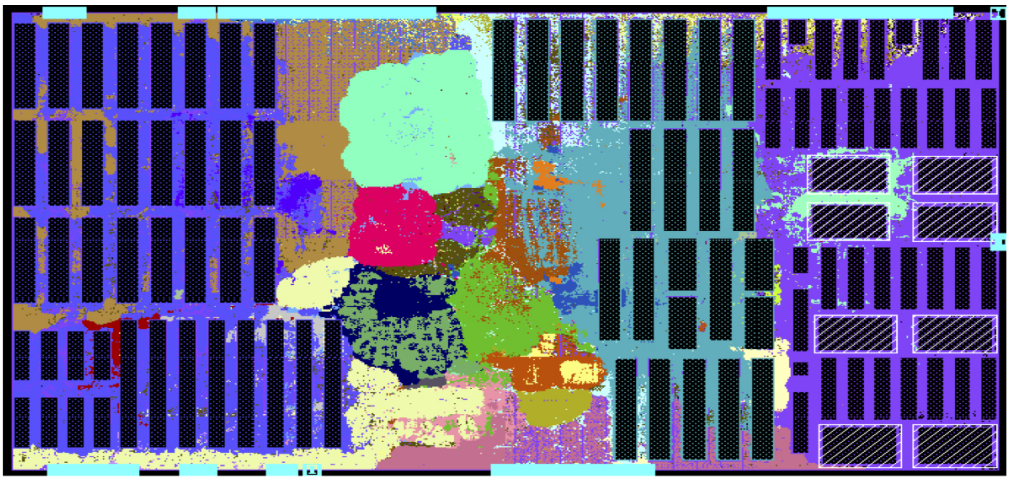


Figure 6.18: *c7* after light manual editing.

gap, reordering the macros to the top right and adding blockages to the bottom-right. The same kind of modifications have been applied to the layout of *c7*, as shown in Fig. 6.17 and Fig. 6.18. These results show that our tool has been able to produce competitive macro placements which could be brought to timing closure in a fraction of the manual engineering effort that would have been needed otherwise.

6.6 Conclusions

This chapter built on the previous to provide macro floorplans with enhanced quality and add designer-friendly capabilities such as suggested macro location and keepouts. Better layouts have been achieved by adding spectral and force placement methods to automatically generate desired macro locations, and these are taken into account during the simulated annealing algorithm by minimizing an adaptive multi-objective cost function. A new round of comprehensive experiments shows the resulting macro placements have quality close to handcrafted floorplans in terms of WL and are generally better in terms of timing. The best layouts at the routing stage have been brought to almost sign-off with little manual modification, showing our tool is capable of providing good starting points for engineers to find a final layout in reduced turnaround time.

Future Work.

After gathering feedback from the physical design engineers, it is clear there are many possible future lines of work for the project. On one hand, some feature requests involve making the tool aware of yet more reality-driven enhancements such as multiple clock domains, support for notches in the block size (rectilinear blocks) or support for macro placement blockages. More effort could also be devoted to increase channel and congestion awareness of our tool, as well as to improve the regularity of the resulting layouts.

On the other hand, the tool has been integrated into an industrial physical design flow, and ongoing experimentation is running to find the best way to integrate the designed algorithms with the designer workflow, and on trying the tool in new benchmarks with different design styles and characteristics, and preparing it for future technology node constraints (7nm and beyond).

6.A HiDaP layouts

This appendix includes standard cell density views of the best TNS layouts shown in Sect. 6.5.3. These images can give an idea of the amount of macro area and their shapes in our benchmark. Contrary to other packing-based approaches, HiDaP can distribute macros across the whole area of the block (*c2, c3*). Space is nonetheless reserved for standard cell logic if they appear reflected on the hierarchy (*c1, c4* and others).

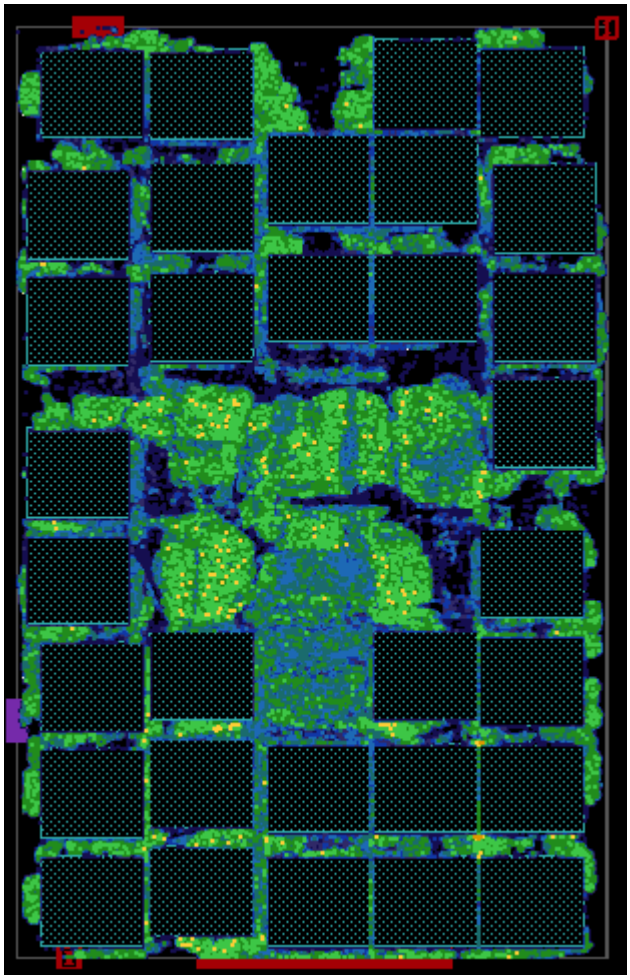


Figure 6.19: *c1* layout.

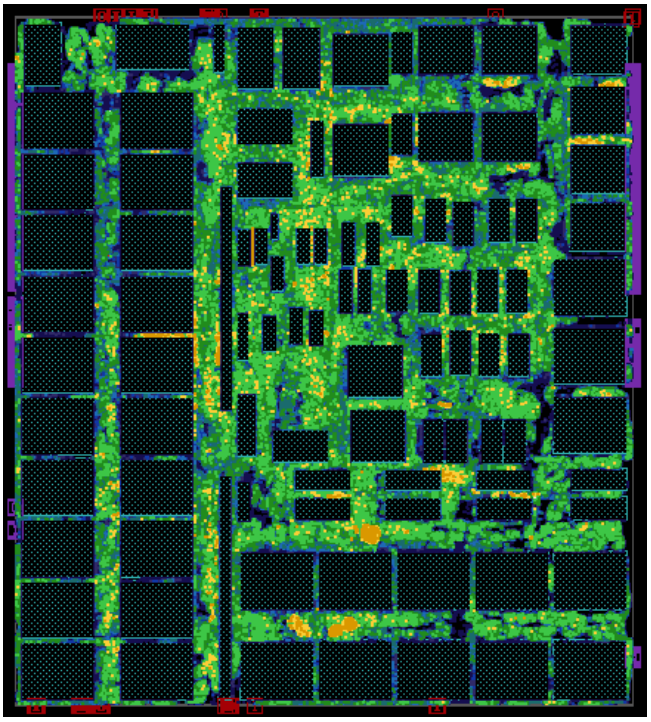


Figure 6.20: *c2* layout.

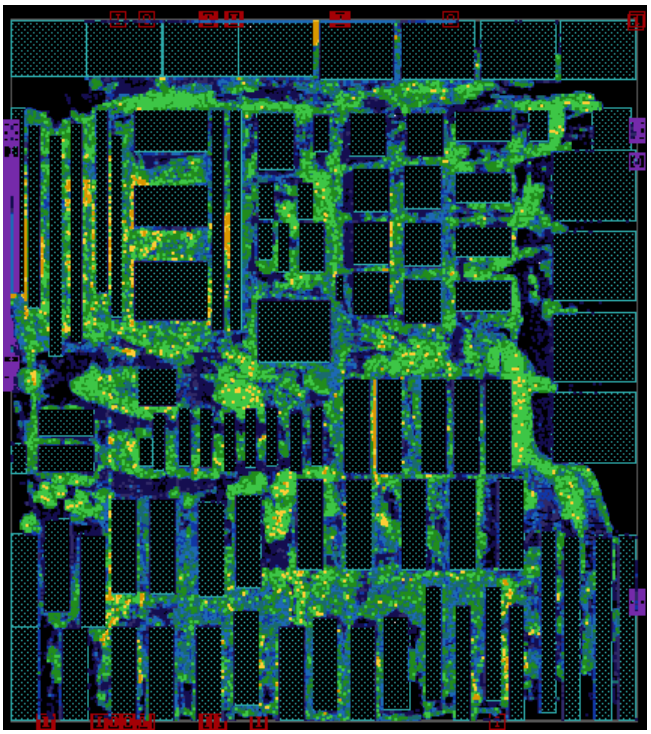


Figure 6.21: *c3* layout.

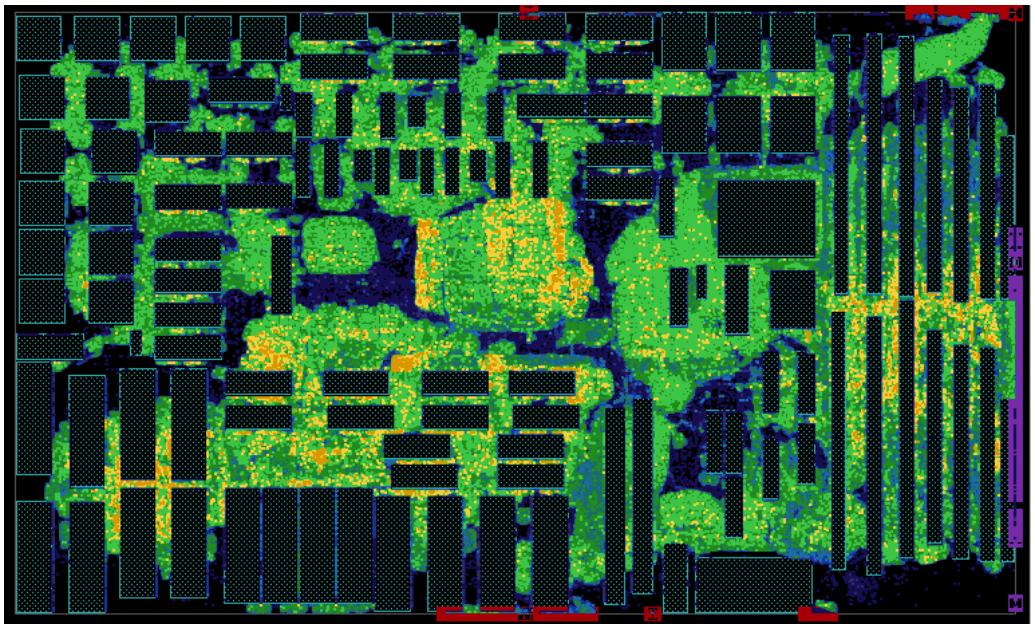


Figure 6.22: *c4* layout.

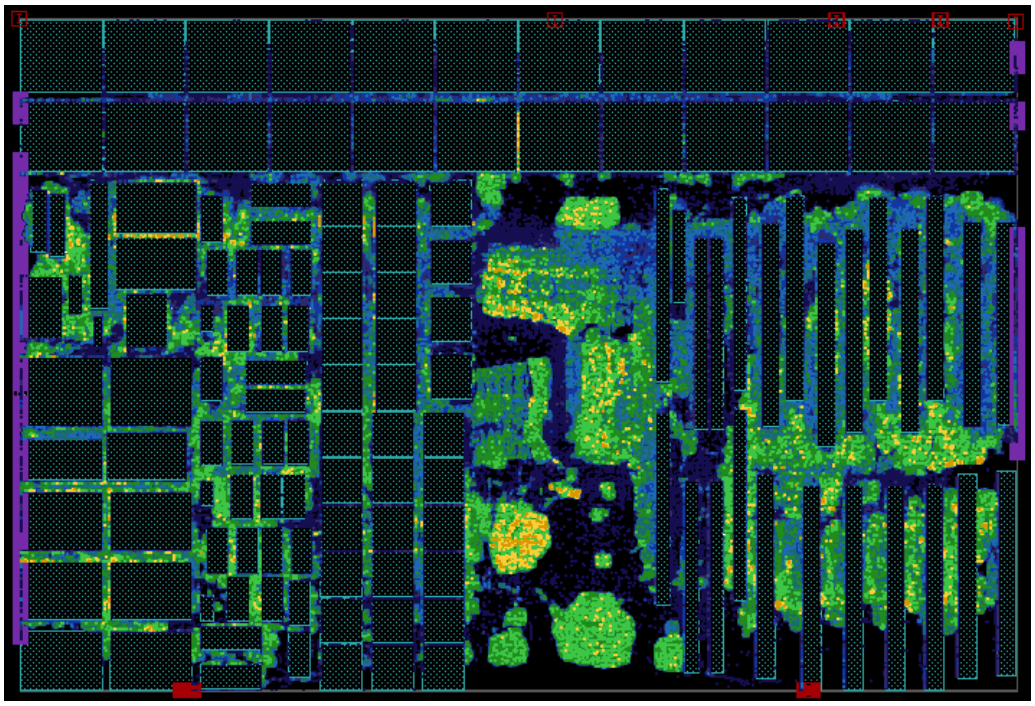


Figure 6.23: *c5* layout.

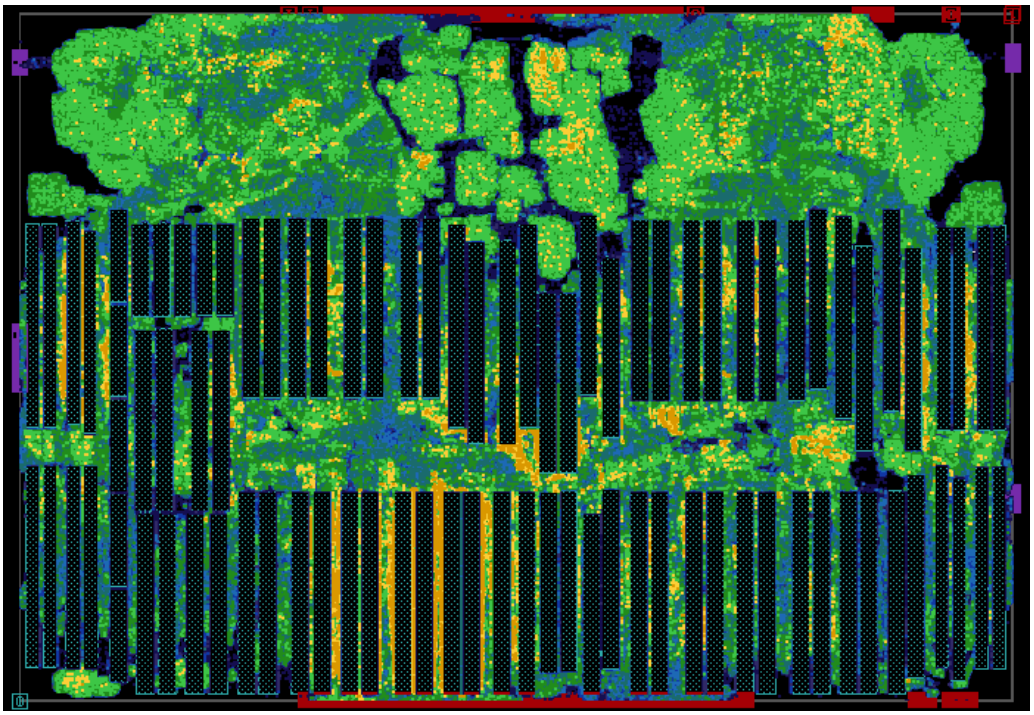


Figure 6.24: *c6* layout.

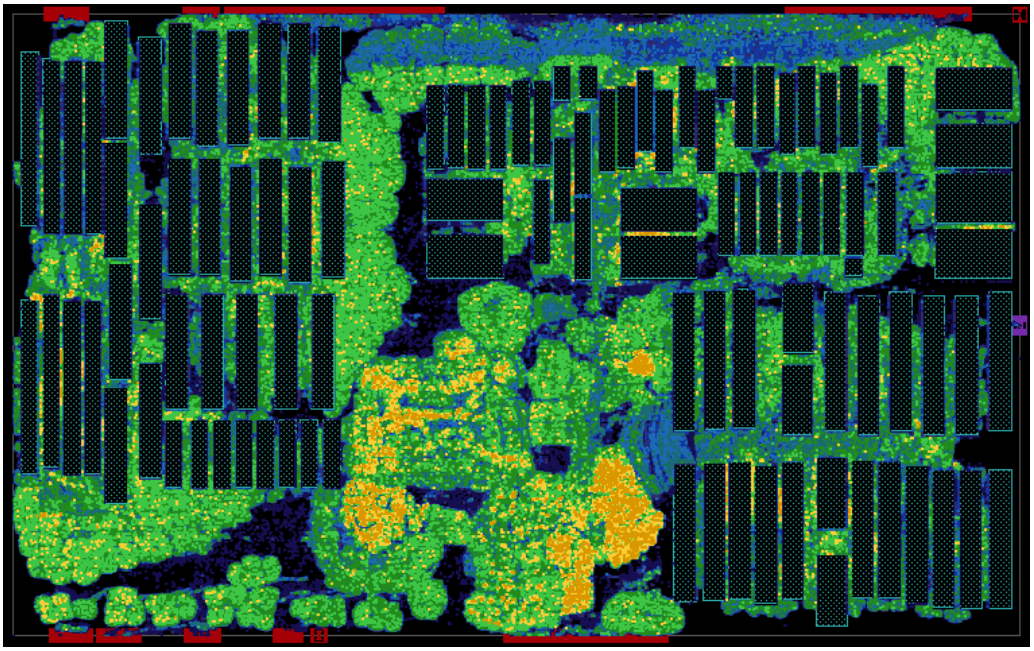


Figure 6.25: *c7* layout.

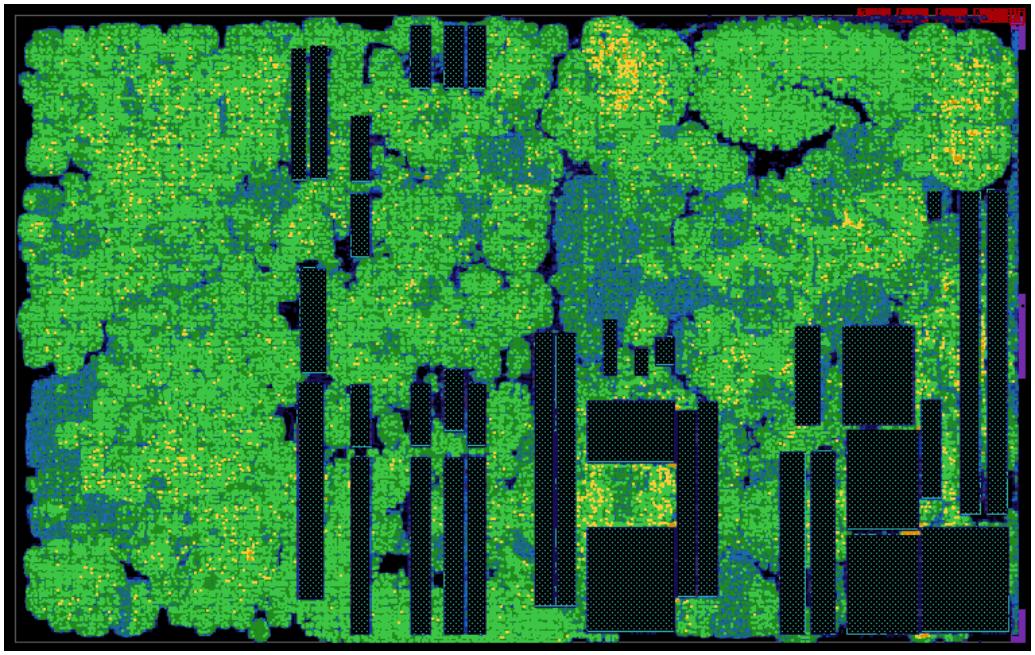


Figure 6.26: *c8* layout.

Chapter 7

Conclusion and Future Work

The goal of the thesis has been to explore new algorithmic techniques to ease the time-consuming manual steps of physical synthesis and find better physical realizations of the gate netlist after logic synthesis. The presented approaches and results have proved to advance the state-of-the-art in several points of the physical design process, by identifying some of its needs and proposing novel algorithmic methods to address them. This chapter finishes the thesis by summarizing the conclusions that have been drawn during its development, most of which have already been mentioned in other parts of this thesis. Additionally, several lines of future work are given.

7.1 Under-the-Cell Routing

The first contribution on under-the-cell routing has shown that potential gains can be obtained by systematically exploiting the use of lateral pins for standard cell abutment connections. Academic tools have been extended to generate layouts enriched with such lateral pins.

Experiments have shown that using microplacement techniques to maximize the opportunity for lateral connections, over a 30% of 2-pin nets can be routed without going to upper metal layers and about 11.5% of pins and 6.7% of the circuit vias can be removed. The approach proved to remove DRC violations in nearly all the tests, resulting in layouts which are easier to bring to the sign off stage.

It is important to remember that, given the limited resources available, the experiments were obtained on a $45nm$ technology, and the impact of such reductions can only be expected to grow in FinFET technology nodes and beyond. To further validate the approach, the collaboration

with standard cell providers and EDA companies would be desirable. On one side, full modeling and validation of the generated layouts for the standard cells and their direct use in the EDA flow would help show the value of our approach. In that direction, another line of work would be to find the minimum set of standard cells with lateral pins that would cover most of the lateral connections appearing in real cases. The questions being of the type: is it enough to enable lateral wires in only the *inverter*, *and* and *or* gates? On the other hand, directly extending existing placements algorithms to consider horizontal abutability to be an optimization constraint would directly incorporate the problem to already known formulations and methods. The final goal would be a full integration of all these pieces into a transparent flow to the user. The proposal is ambitious, but once adopted could easily become a standard practice in industry to help pushing designs down to the newest, most challenging technology nodes.

7.2 Macro Placement

The second contribution of the thesis contains a state-of-the-art study of current macro placement practices, both academic and industrial, to identify key areas for improvement. The HiDaP tool has been written to address some of these needs by exploiting RTL information from the netlist to group components and identify the dataflow relations between them. This information has both been used to guide an automated macro placer and to generate graph representations to help designers in the macro placement stage. Following the feedback of physical designers, the approach has been enriched with methods to suggest macro locations to the tool, either provided by them or by spectral and force based placement algorithms such as the ones proposed in the last chapter.

The experimental results on the last version of the tool shows that, after placement, the generated layouts can have up to less 40% worst negative slack and total negative slack when compared to tape-out-ready designs done by expert back-end engineers, while obtaining similar wirelength and congestion results. The best design among five for each circuit was given to a physical designer, tasked with bringing the layout to timing closure with close to no design rule violations. They succeeded in their task in only one to two rounds of manual macro layout modifications, proving the applicability of the tool in a real design flow as a prototyping helper for physical design engineers.

In fact, given its promising results, the development of HiDaP is expected to continue in the future. Some lines of work were already given in Chapter 6, including block and notch support and other reality-driven enhancements, and well as further testing with other benchmarks with different design styles and characteristics in new technology nodes. Looking back at Chapter 4, some other lines of work emerge. For instance,

1. Floorplanners are still not modeling detailed macro placement, including notions such as regularity, channel sizes and techniques such as back to back placement. These might have an important effect in final DRC count, specially in new technology nodes.

2. The interaction between macro placement and standard cell placement is not well solved, as most macro placers do not scale enough, and most cell placers have problems during the legalization phase. Although our approach has tried to work in the direction of placing macros considering standard cells, we acknowledge more effort is needed in this direction so that our tools efficiently explore the solution space.

3. Our approach relies on the RTL information left in the netlist, which is assumed to be a good guide for the physical design process, but what would happen if this information was not physically-aware or created pathological problems with the algorithm? Methods to decide whether a hierarchy is usable and bypass or modify it when it is not adequate for the physical design process may help to alleviate this issue.

Another topic that was discussed in Chapter 5 is on obtaining better netlist abstractions at the RTL-floorplan boundary for fast diagnosis. The work in this thesis has provided a method to automatically generate graphs which describe the main circuit components and their relations. Other more general questions remain yet to be addressed, and the most urgent is: how do we know a macro placement is good? Of course, without having to run it through expensive placement methods. A future line of work would be to develop methods centered in this question to be able to assess macro placement quality fast. If this existed, new uses for such tool in the flow could appear, for example combining a macro placer with a macro selection tool to do architectural prototyping, that is, picking the macros that will lead to a better macro placement (instead of considering them an fixed input to the problem). As we move into a sea-of-macros

paradigm inside each block, re-adapting placement techniques or the discovery of new ones is a must to handle the increasing complexity and challenges in future technology nodes.

Bibliography

[1] S. N. Adya, S. Chaturvedi, J. A. Roy, D. A. Papa, and I. L. Markov. Unification of Partitioning, Placement and Floorplanning. In *IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004.*, pages 550–557, Nov. 2004.

[2] S. N. Adya and I. L. Markov. Combinatorial Techniques for Mixed-Size Placement. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 10(1):58–90, 2005.

[3] T. Ajayi, V. A. Chhabria, M. Fogaça, S. Hashemi, A. Hosny, A. B. Kahng, M. Kim, J. Lee, U. Mallappa, M. Neseem, G. Pradipta, S. Reda, M. Saligane, S. S. Sapatnekar, C. Sechen, M. Shalan, W. Swartz, L. Wang, Z. Wang, M. Woo, and B. Xu. Toward an Open-Source Digital Flow: First Learnings from the OpenROAD Project. In *Proceedings of the 56th Design Automation Conference*, pages 76–79. ACM, June 2019.

[4] I. L. M. J. H. Andrew B. Kahng, Jens Lienig. *VLSI Physical Design: From Graph partitioning to Timing Closure*. Springer, 2011.

[5] B. Cakir and S. Malik. Reverse Engineering Digital ICs through Geometric Embedding of Circuit Graphs. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 23(4):50, July 2018.

[6] W. J. Chan, Y. Du, A. B. Kahng, S. Nath, and K. Samadi. BEOL Stack-Aware Routability Prediction From Placement Using Data Mining Techniques. In *2016 IEEE 34th International Conference on Computer Design (ICCD)*, pages 41–48, Oct. 2016.

[7] W.-T. J. Chan, P.-H. Ho, A. B. Kahng, and P. Saxena. Routability Optimization for Industrial Designs at Sub-14nm Process Nodes Using Machine Learning. In *Proceedings of the 2017 ACM on International Symposium on Physical Design*, pages 15–21. ACM, Mar. 2017.

[8] C. H. Chang, Y. W. Chang, and T. C. Chen. A Novel Damped-Wave Framework for Macro Placement. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 504–511, 2017.

[9] Y.-W. Chang and K.-T. Cheng, editors. *Electronic Design Automation: Synthesis, Verification, and Test*. Morgan Kaufmann, 2009.

[10] D. P. M. Charles J. Alpert and S. S. Sapatnekar, editors. *Handbook of Algorithms for Physical Design Automation*. CRC Press, 2008.

[11] S. T. Chen, Y. W. Chang, and T. C. Chen. An Integrated-Spreading-Based Macro-Refining Algorithm for Large-Scale Mixed-Size Circuit Designs. In *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 496–503, 2017.

[12] T. C. Chen, P. H. Yuh, Y. W. Chang, F. J. Huang, and T. Y. Liu. MP-Trees: A Packing-Based Macro Placement Algorithm for Modern Mixed-Size Designs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(9):1621–1634, 2008.

[13] Y. F. Chen, C. C. Huang, C. H. Chiou, Y. W. Chang, and C. J. Wang. Routability-Driven Blockage-Aware Macro Placement. In *2014 51st ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2014.

[14] C. Cheng, A. B. Kahng, I. Kang, and L. Wang. RePlAce: Advancing Solution Quality and Routability Validation in Global Placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, pages 1717–1730, 2018.

[15] Ching-Chung Hu, De-Sheng chen, and Yi-Wen Wang. Fast Multi-level Floorplanning for Large Scale Modules. In *2004 IEEE International Symposium on Circuits and Systems (IEEE Cat. No.04CH37512)*, volume 5, May 2004.

[16] C. H. Chiou, C. H. Chang, S. T. Chen, and Y. W. Chang. Circular-Contour-Based Obstacle-Aware Macro Placement. In *2016 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 172–177, 2016.

[17] W. Choi and K. Bazargan. Hierarchical Global Floorplacement using Simulated Annealing and Network Flow Area Migration. In *Design, Automation and Test in Europe Conference and Exhibition, 2003*, pages 1104–1105. IEEE, 2003.

[18] Y. L. Chuang et al. Design-Hierarchy Aware Mixed-Size Placement for Routability Optimization. In *Proceedings of the IEEE/ACM International Conference on Computer Aided Design, 2010.*, pages 663–668, 2010.

[19] J. P. Cohoon and W. Paris. Genetic Placement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 6(6):956–964, Nov. 1987.

[20] J. Cong, M. Romesis, and J. R. Shinnerl. Fast Floorplanning by Look-ahead Enabled Recursive Bipartitioning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(9):1719–1732, Sept. 2006.

[21] J. Cortadella, J. Petit, S. Gómez, and F. Moll. A Boolean Rule-Based Approach for Manufacturability-Aware Cell Routing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(3):409–422, Mar. 2014.

[22] P. Cremer, S. Hougardy, J. Schneider, and J. Silvanus. Automatic Cell Layout in the 7nm Era. In *Proceedings of the 2017 ACM on International Symposium on Physical Design*, pages 99–106. ACM, Mar. 2017.

[23] A. E. Dunlop and B. W. Kernighan. A Procedure for Placement of Standard-Cell VLSI Circuit. *IEEE Transactions of Circuits and Systems*, 4(1):92–98, 1985.

[24] Hard Macros will Revolutionize SoC Design. Last accessed november 2019, https://www.eetimes.com/document.asp?doc_id=1217863.

[25] T. M. J. Fruchterman and E. M. Reingold. Graph Drawing by Force-Directed Placement. *Software - Practice & Experience*, 21(11):1129–1164, Nov. 1991.

[26] J. Funke, S. Hougardy, and J. Schneider. An Exact Algorithm for Wirelength Optimal Placements in VLSI Design. *Integration, the VLSI Journal*, 52:355–366, Jan. 2016.

[27] Gang Xu, Ruiqi Tian, D. Z. Pan, and M. D. F. Wong. CMP Aware Shuttle Mask Floorplanning. In *Proceedings of the ASP-DAC 2005. Asia and South Pacific Design Automation Conference, 2005.*, volume 2, pages 1111–1114 Vol. 2, Jan. 2005.

[28] Graphviz. Last accessed november 2019, http://www.graphviz.org/.

[29] K. M. Hall. An r-Dimensional Quadratic Placement Algorithm. *Management Science*, 17(3):219–229, 1970.

[30] M. Hsu, Y. Chen, C. Huang, S. Chou, T. Lin, T. Chen, and Y. Chang. NTUplace4h: A Novel Routability-Driven Placement Algorithm for Hierarchical Mixed-Size Circuit Designs. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 33(12):1914–1927, Dec. 2014.

[31] M. K. Hsu et al. Routability-Driven Placement for Hierarchical Mixed-Size Circuit Designs. In *Proceedings of the 50th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, 2013.

[32] C. C. Huang, H. Y. Lee, B. Q. Lin, S. W. Yang, C. H. Chang, S. T. Chen, Y. W. Chang, T. C. Chen, and I. Bustany. NTUplace4dr: A Detailed-Routing-Driven Placer for Mixed-Size Circuit Designs With Technology and Region Constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 37(3):669–681, Mar. 2018.

[33] Y.-H. Huang, Z. Xie, G.-Q. Fang, T.-C. Yu, H. Ren, S.-Y. Fang, Y. Chen, and Jiang Hu. Routability-Driven Macro Placement with Embedded CNN-Based Prediction Model. In *Design, Automation and Test in Europe Conference and Exhibition, 2019*, Mar. 2019.

[34] ICCAD 2012 Hierarchy-Aware Placement Contest. Last accessed november 2019, http://cad-contest.cs.nctu.edu.tw/CAD-contest-at-ICCAD2012/.

[35] Synopsys IC Compiler. Last accessed november 2019, www.synopsys.com/Tools/Implementation/PhysicalImplementation/Pages/ICCompiler.aspx.

[36] itc99 Benchmark Homepage. Last accessed november 2019, https://github.com/squillero/itc99-poli.

[37] Jai-Ming Lin and Yao-Wen Chang. TCG: a Transitive Closure Graph-based Representation for Non-slicing Floorplans. In *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*, pages 764–769, June 2001.

[38] A. B. Kahng. Classical Floorplanning Harmful? In *Proceedings of the 2000 international symposium on Physical design*, pages 207–213. ACM, May 2000.

[39] A. B. Kahng, I. Măndoiu, Q. Wang, X. Xu, and A. Z. Zelikovsky. Multi-project Reticle Floorplanning and Wafer Dicing. In *Proceedings of the 2004 International Symposium on Physical Design*, pages 70–77. ACM, Apr. 2004.

[40] A. B. Kahng and S. Reda. Reticle Floorplanning with Guaranteed Yield for Multi-project Wafers. In *IEEE International Conference on Computer Design: VLSI in Computers and Processors, 2004.*, pages 106–110, Oct. 2004.

[41] I. Kang, F. Qiao, D. Park, D. Kane, E. F. Y. Young, C.-K. Cheng, and R. Graham. Three-dimensional Floorplan Representations by Using Corner Links and Partial Order. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 24(1):13, Jan. 2019.

[42] D. H. Kim and S. K. Lim. Bus-Aware Microarchitectural Floorplanning. In *Proceedings of the 12th Asia and South Pacific Design Automation Conference*, pages 204–208. IEEE Computer Society Press, Jan. 2008.

[43] M. Kim and I. L. Markov. ComPLx: A Competitive Primal-dual Lagrange Optimization for Global Placement. In *DAC Design Automation Conference 2012*, pages 747–755, June 2012.

[44] M.-C. Kim, N. Viswanathan, C. J. Alpert, I. L. Markov, and S. Ramji. MAPLE: Multilevel Adaptive Placement for Mixed-Size Designs. In *Proceedings of the 2012 ACM international symposium on International Symposium on Physical Design*, pages 193–200. ACM, 2012.

[45] Y. Koren. On Spectral Graph Drawing. In *Proceedings of the 9th annual International Conference on Computing and Combinatorics*, pages 496–508. Springer-Verlag, July 2003.

[46] P. Kumar, A. Fell, and S. Mathur. Automated, inter-macro channel space adjustment and optimization for faster design closure. In *2017 30th IEEE International System-on-Chip Conference (SOCC)*, pages 74–79, Sept. 2017.

[47] M. Lai and D. F. Wong. Slicing Tree is a Complete Floorplan Representation. In *Proceedings Design, Automation and Test in Europe. Conference and Exhibition 2001*, pages 228–232, Mar. 2001.

[48] W. Li, A. Gascon, P. Subramanyan, W. Y. Tan, A. Tiwari, S. Malik, N. Shankar, and S. A. Seshia. WordRev: Finding Word-level Structures in a Sea of Bit-level Gates. In *2013 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 67–74, June 2013.

[49] J. Lin, Y. Deng, S. Li, B. Yu, L. Chang, and T. Peng. Regularity-Aware Routability-Driven Macro Placement Methodology for Mixed-Size Circuits With Obstacles. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, pages 1–12, 2018.

[50] J.-M. Lin, S.-T. Li, and Y.-T. Wang. Routability-driven Mixed-size Placement Prototyping Approach Considering Design Hierarchy and Indirect Connectivity Between Macros. In *Proceedings of the 56th Annual Design Automation Conference 2019*, page 119. ACM, June 2019.

[51] J. M. Lin, B. H. Yu, and L. Y. Chang. Regularity-Aware Routability-Driven Placement Prototyping Algorithm for Hierarchical Mixed-Size Circuits. In *2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 438–443, Jan. 2017.

[52] M.-C. Lin, K. Ferguson, M. Y. Fang, and S.-P. Ko. Automatic Abutment for Devices with Horizontal Pins, US patent US9348963B1, May 2016.

[53] T. Lin, C. Chu, J. R. Shinnerl, I. Bustany, and I. Nedelchev. POLAR: A High Performance Mixed-Size Wirelengh-Driven Placer With Density Constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(3):447–459, Mar. 2015.

[54] Y.-C. Liu, T.-C. Chen, Y.-W. Chang, and S.-Y. Kuo. MDP-trees: multi-domain macro placement for ultra large-scale mixed-size designs. In *Proceedings of the 24th Asia and South Pacific Design Automation Conference*, pages 557–562. ACM, Jan. 2019.

[55] J. Lu, H. Zhuang, P. Chen, H. Chang, C.-C. Chang, Y.-C. Wong, L. Sha, D. Huang, Y. Luo, C.-C. Teng, and C.-K. Cheng. ePlace-MS:

Electrostatics-Based Placement for Mixed-Size Circuits. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(5):685–698, May 2015.

[56] Marc Pons Solé. *Layout Regularity for Design and Manufacturability*. PhD thesis, Polytechnic University of Catalonia, July 2012.

[57] I. L. Markov, J. Hu, and M.-C. Kim. Progress and Challenges in VLSI Placement Research. *Procs. of the IEEE*, 103(11):1985–2003, 2015.

[58] Maxeda. Last accessed november 2019, http://www.maxeda.tech/product.html.

[59] G. E. Moore. Cramming More Components onto Integrated Circuits. *Electronics (magazine)*, 38(8), 1965.

[60] H. Murata, K. Fujiyoshi, S. Nakatake, and Y. Kajitani. VLSI Module Placement Based on Rectangle-packing by the Sequence-pair. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 15(12):1518–1524, Dec. 1996.

[61] Nangate 45nm Open Cell Library. Last accessed november 2019, https://www.silvaco.com/products/nangate/FreePDK45_Open_Cell_Library.

[62] V. Nookala, Ying Chen, D. J. Lilja, and S. S. Sapatnekar. Microarchitecture-Aware Floorplanning Using a Statistical Design of Experiments Approach. In *Proceedings. 42nd Design Automation Conference, 2005.*, pages 579–584, June 2005.

[63] R. H. Otten. Efficient Floorplan Optimization. In *Proceedings of International Conference on Computer Design*, pages 499–502, 1983.

[64] D. Pan, B. Yu, and J.-R. Gao. Design for Manufacturing With Emerging Nanolithography. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 32(10):1453–1472, Oct. 2013.

[65] Ping-Hung Yuh, Chia-Lin Yang, and Yao-Wen Chang. Temporal Floorplanning Using the T-tree Formulation. In *IEEE/ACM International Conference on Computer Aided Design, 2004. ICCAD-2004.*, pages 300–305, Nov. 2004.

[66] R. S. S. Prashant Saxena and S. S. Sapatnekar. *Routing Congestion in VLSI Circuits - Estimation and Optimization*. Series on Integrated Circuits and Systems. Springer, 2007.

[67] J. Roy, S. Adya, D. Papa, and I. Markov. Min-cut Floorplacement. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(7):1313–1326, July 2006.

[68] J. Roy, N. Viswanathan, G.-J. Nam, C. Alpert, and I. Markov. CRISP: Congestion Reduction by Iterated Spreading During Placement. In *IEEE/ACM International Conference on Computer-Aided Design - Digest of Technical Papers, 2009. ICCAD 2009*, pages 357–362, Nov. 2009.

[69] N. Ryzhenko, S. Burns, A. Sorokin, and M. Talalay. Pin Access-Driven Design Rule Clean and DFM Optimized Routing of Standard Cells under Boolean Constraints. In *Proceedings of the 2019 International Symposium on Physical Design - ISPD '19*, pages 41–47, San Francisco, CA, USA, 2019. ACM Press.

[70] C. D. G. S. Kirkpatrick and M. P.Vecchi. Optimization by Simulated Annealing. *Science*, 220:671–680, 1983.

[71] C. Sechen and A. Sangiovanni-Vincentelli. The TimberWolf Placement and Routing Package. *IEEE Journal of Solid-State Circuits*, 20(2):510–522, 1985.

[72] D. Shi and A. Davoodi. Improving Detailed Routability and Pin Access with 3d Monolithic Standard Cells. In *Proceedings of the 2017 ACM on International Symposium on Physical Design*, pages 107–112. ACM, Mar. 2017.

[73] L. Stockmeyer. Optimal Orientations of Cells in Slicing Floorplan Designs. *Information and Control*, 57(2):91–101, May 1983.

[74] P. Subramanyan, N. Tsiskaridze, W. Li, A. Gascón, W. Y. Tan, A. Tiwari, N. Shankar, S. A. Seshia, and S. Malik. Reverse Engineering Digital Circuits Using Structural and Functional Analyses. *IEEE Transactions on Emerging Topics in Computing*, 2(1):63–80, Mar. 2014.

[75] A. F. Tabrizi, N. K. Darav, S. Xu, L. Rakai, I. Bustany, A. Kennings, and L. Behjat. A machine learning framework to identify detailed routing short violations from a placed netlist. In *Proceedings of the 55th Annual Design Automation Conference*, page 48. ACM, June 2018.

[76] T. Taghavi, C. Alpert, A. Huber, Z. Li, G.-J. Nam, and S. Ramji. New Placement Prediction and Mitigation Techniques for Local Routing Congestion. In *2010 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 621–624, Nov. 2010.

[77] M. Then, M. Kaufmann, F. Chirigati, T.-A. Hoang-Vu, K. Pham, A. Kemper, T. Neumann, and H. T. Vo. The more the merrier: efficient multi-source graph traversal. *Proceedings of the VLDB Endowment*, 8(4):449–460, Dec. 2014.

[78] Tung-Chieh Chen and Yao-Wen Chang. Modern Floorplanning Based on B*-tree and Fast Simulated Annealing. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(4):637–650, Apr. 2006.

[79] Tung-Chieh Chen, Yao-Wen Chang, and Shyh-Chang Lin. A New Multilevel Framework for Large-Scale Interconnect-Driven Floorplanning. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(2):286–294, Feb. 2008.

[80] A. Vidal-Obiols, J. Cortadella, J. Petit, M. Galceran-Oms, and F. Martorell. RTL-Aware Dataflow-Driven Macro Placement. In *2019 Design, Automation Test in Europe Conference Exhibition (DATE)*, pages 186–191, Mar. 2019.

[81] N. Viswanathan, C. Alpert, C. Sze, Z. Li, and Y. Wei. The DAC 2012 Routability-driven Placement Contest and Benchmark Suite. In *2012 49th ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 774–782, June 2012.

[82] N. Viswanathan, C. Alpert, C. Sze, Z. Li, and Y. Wei. ICCAD-2012 CAD Contest in Design Hierarchy Aware Routability-driven Placement and Benchmark Suite. In *2012 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 345–348, Nov. 2012.

[83] N. Viswanathan, M. Pan, and C. Chu. FastPlace 3.0: A Fast Multilevel Quadratic Placement Algorithm with Placement Congestion Control. In *Proceedings of the 2007 Asia and South Pacific Design Automation Conference*, pages 135–140. IEEE Computer Society, Jan. 2007.

[84] C. Walshaw. A Multilevel Algorithm for Force-Directed Graph Drawing. In *Proceedings of the 8th International Symposium on Graph Drawing*, pages 171–182. Springer-Verlag, Sept. 2000.

[85] D. F. Wong and C. L. Liu. A New Algorithm for Floorplan Design. In *23rd ACM/IEEE Design Automation Conference*, pages 101–107, June 1986.

[86] Xiaoping Tang, Ruiqi Tian, and D. F. Wong. Fast evaluation of sequence pair in block placement by longest common subsequence computation. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(12):1406–1413, Dec. 2001.

[87] Xiaoping Tang and D. F. Wong. FAST-SP: a Fast Algorithm for Block Placement Based on Sequence Pair. In *Proceedings of the ASP-DAC 2001. Asia and South Pacific Design Automation Conference 2001 (Cat. No.01EX455)*, pages 521–526, Feb. 2001.

[88] Z. Xie, Y. Huang, G. Fang, H. Ren, S. Fang, Y. Chen, and J. Hu. RouteNet: Routability prediction for Mixed-Size Designs Using Convolutional Neural Network. In *2018 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, pages 1–8, Nov. 2018.

[89] X. Xu, B. Cline, G. Yeric, B. Yu, and D. Pan. Self-Aligned Double Patterning Aware Pin Access and Standard Cell Layout Co-Optimization. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(5):699–712, May 2015.

[90] X. Xu, B. Yu, J.-R. Gao, C.-L. Hsu, and D. Pan. PARR: Pin Access Planning and Regular Routing for Self-aligned Double Patterning. In *2015 52nd ACM/EDAC/IEEE Design Automation Conference (DAC)*, pages 1–6, June 2015.

[91] H. Yamazaki, K. Sakanushi, N. Shigetoshi, and K. Yoji. The 3D-packing by Meta Data Structure and Packing Heuristics. *IEICE Trans. Fundamentals of Electronics, Communications and Computer Sciences*, E83-A(4):639–645, Apr. 2000.

[92] J. Z. Yan, N. Viswanathan, and C. Chu. Handling complexities in modern large-scale mixed-size placement. In *2009 46th ACM/IEEE Design Automation Conference*, pages 436–441, July 2009.

[93] F. Y. Young and D. F. Wong. How Good are Slicing Floorplans? *Integration, the VLSI Journal*, 23(1):61–73, Oct. 1997.

[94] F. Y. Young and D. F. Wong. Slicing Floorplans with Pre-placed Modules. In *Proceedings of the 1998 IEEE/ACM international conference on Computer-aided design*, pages 252–258. ACM, Nov. 1998.

[95] F. Y. Young, D. F. Wong, and H. H. Yang. Slicing Floorplans with Boundary Constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 18(9):1385–1389, Sept. 1999.

[96] F. Y. Young, D. F. Wong, and H. H. Yang. Slicing floorplans with range constraint. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 19(2):272–278, Feb. 2000.

[97] F. Y. Young, M. D. F. Wong, and H. H. Yang. On Extending slicing Floorplan to Handle L/T-shaped Modules and Abutment Constraints. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 20(6):800–807, June 2001.

[98] W. S. Yuen and F. Y. Young. Slicing Floorplan with Clustering Constraints. In *Proceedings of the ASP-DAC 2001. Asia and South Pacific Design Automation Conference 2001 (Cat. No.01EX455)*, pages 503–508, Feb. 2001.

[99] P.-H. Yuh, C.-L. Yang, and Y.-W. Chang. Temporal Floorplanning Using the Three-dimensional Transitive Closure subGraph. *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, 12(4):37, Sept. 2007.

[100] Yun-Chih Chang, Yao-Wen Chang, Guang-Ming Wu, and Shu-Wei Wu. B*-trees: a New Representation for Non-slicing Floorplans. In *Proceedings 37th Design Automation Conference*, pages 458–463, June 2000.

[101] H. Zhang, M. Wong, and K.-Y. Chao. On Process-aware 1-D Standard Cell Design. In *Design Automation Conference (ASP-DAC), 2010 15th Asia and South Pacific*, pages 838–842, Jan. 2010.

[102] Y. Zhang and C. Chu. CROP: Fast and Effective Congestion Refinement of Placement. In *IEEE/ACM International Conference on Computer-Aided Design - Digest of Technical Papers, 2009. ICCAD 2009*, pages 344–350, Nov. 2009.