UNIVERSITY OF CALIFORNIA SAN DIEGO

Improved Physical Design for Manufacturing Awareness and Advanced VLSI

A dissertation submitted in partial satisfaction of the
requirements for the degree
Doctor of Philosophy

in

Electrical Engineering (Computer Engineering)

by

Lutong Wang

Committee in charge:

Professor Andrew B. Kahng, Chair
Professor Chung-Kuan Cheng
Professor Puneet Gupta
Professor Rajesh K. Gupta
Professor Farinaz Koushanfar
Professor Bill Lin

2020

The dissertation of Lutong Wang is approved, and it is acceptable in quality and form for publication on microfilm and electronically:

_____

_____

_____

_____

_____

_____

Chair

University of California San Diego

2020

DEDICATION

To my family.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

Foremost, I would like to thank my advisor Professor Andrew B. Kahng for his continuous guidance and support throughout my Ph.D. study. I have learned a lot from his enormous amount of knowledge, passion and responsibility in the research process.

I would like to thank my fellow labmates in the UCSD VLSI CAD Laboratory (Minsoo Kim, Hsin-Yu Liu, Zhiang Wang, Mingyu Woo and Bangqi Xu) and former lab members (Dr. Wei-Ting (Jonas) Chan, Ahmed Taha Elthakeb, Dr. Kwangsoo Han, Chia-Tung Ho, Dr. Hyein Lee, Dr. Jiajia Li, Mulong Luo, Uday Mallappa, Dr. Siddhartha Nath, Tushar Shah, Dr. Vaishnav Srinivas, Yaping Sun and Sriram Venkatesh) for their assistance and enthusiastic discussions. I would also like to thank Dr. Tuck-Boon Chan and Dr. Ilgweon Kang for their guidance through countless discussions.

I would also like to thank my industrial collaborators (Peter Debacker, Changho Han, Sun ik Heo, Soowan Hong, Chunghee Kim and Dr. Praveen Raghavan) for their invaluable guidance and feedback in many of my research projects.

My sincere thanks go to my thesis committee members Professor Chung-Kuan Cheng, Professor Puneet Gupta, Professor Rajesh Gupta, Professor Farinaz Koushanfar and Professor Bill Lin for their time, encouragement and insightful comments.

Last, but not least, I would like to thank my family. This journey would not have been made possible without their continuous support and sacrifice.

The material in this thesis is based on the following publications.

Chapter 2 contains the reprints of Andrew B. Kahng, Jiajia Li and Lutong Wang, “Improved Flop Tray-Based Design Implementation for Power Reduction”, *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2016; and Peter Debacker, Kwangsoo Han, Andrew B. Kahng, Hyein Lee, Praveen Raghavan and Lutong Wang, “MILP-Based Optimization of 2D Block Masks for Timing-Aware Dummy Segment Removal in Self-Aligned Multiple Patterning Layouts”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36(7), 2017. The dissertation author is a main contributor to, and a primary author of, the second paper.

I would like to thank my coauthors Peter Debacker, Kwangsoo Han, Andrew B. Kahng, Hyein Lee, Jiajia Li and Praveen Raghavan for their support and work.

Chapter 3 contains reprints of Changho Han, Kwangsoo Han, Andrew B. Kahng, Hyein Lee, Lutong Wang and Bangqi Xu, "Optimal Multi-Row Detailed Placement for Yield and Model-Hardware Correlation Improvements in Sub-10nm VLSI", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2017; Changho Han, Andrew B. Kahng, Lutong Wang and Bangqi Xu, "Enhanced Optimal Multi-Row Detailed Placement for Neighbor Diffusion Effect Mitigation in Sub-10nm VLSI", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38(9), 2019; and Peter Debacker, Kwangsoo Han, Andrew B. Kahng, Hyein Lee, Praveen Raghavan and Lutong Wang, "Vertical M1 Routing-Aware Detailed Placement for Congestion and Wirelength Reduction in Sub-10nm Nodes", *Proc. ACM/EDAC/IEEE Design Automation Conference*, 2017. The dissertation author is a main contributor to, and a primary author of, each of these papers.

I would like to thank my coauthors Peter Debacker, Changho Han, Kwangsoo Han, Andrew B. Kahng, Hyein Lee, Praveen Raghavan and Bangqi Xu, as well as the research support from Samsung Electronics.

Chapter 4 contains reprints of Andrew B. Kahng, Lutong Wang and Bangqi Xu, "The Tao of PAO: Anatomy of a Pin Access Oracle for Detailed Routing", *Proc. ACM/IEEE Design Automation Conference*, 2020. Chapter 4 also contains the draft of Andrew B. Kahng, Lutong Wang and Bangqi Xu, "TritonRoute: The Open Source Detailed Router", in submission to *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020. The dissertation author is a main contributor to, and a primary author of, each of these papers.

I would like to thank my coauthors Andrew B. Kahng and Bangqi Xu for their support and work.

My coauthors (Mr. Peter Debacker, Mr. Changho Han, Dr. Kwangsoo Han, Professor Andrew B. Kahng, Dr. Hyein Lee, Dr. Jiajia Li, Dr. Praveen Raghavan and Mr. Bangqi Xu, listed in alphabetical order) have all kindly approved the inclusion of the aforementioned publications in my thesis.

VITA

2014	B. Eng., Microelectronics, Tsinghua University, Beijing, China
2015	M. S., Electrical Engineering (Computer Engineering), University of California, San Diego
2018	C. Phil., Electrical Engineering (Computer Engineering), University of California, San Diego
2020	Ph. D., Electrical Engineering (Computer Engineering), University of California, San Diego

All papers coauthored with my advisor Professor Andrew B. Kahng have authors listed in alphabetical order.

PUBLICATIONS

Andrew B. Kahng, **L. Wang** and B. Xu, "The Tao of PAO: Anatomy of a Pin Access Oracle for Detailed Routing", *Proc. ACM/IEEE Design Automation Conference*, 2020, to appear.

C.-K. 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* 38(9) (2019), pp. 1717-1730.

C. Han, A. B. Kahng, **L. Wang** and B. Xu, "Enhanced Optimal Multi-Row Detailed Placement for Neighbor Diffusion Effect Mitigation in Sub-10nm VLSI", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38(9) (2019), pp. 1703-1716.

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", *Proc. ACM/IEEE Design Automation Conference*, 2019, pp. 76:1-76:4.

S. Heo, A. B. Kahng, M. Kim, **L. Wang** and C. Yang, "Detailed Placement for IR Drop Mitigation by Power Staple Insertion in Sub-10nm VLSI", *Proc. Design, Automation and Test in Europe*, 2019, pp. 824-829.

M. Fogaça, A. B. Kahng, R. Reis and **L. Wang**, "Finding Placement-Relevant Clusters With Fast Modularity-Based Clustering", *Proc. Asia and South Pacific Design Automation Conference*, 2019, pp. 569-576.

S. Heo, A. B. Kahng, M. Kim and **L. Wang**, "Diffusion Break-Aware Leakage Power Optimization and Detailed Placement in Sub-10nm VLSI", *Proc. Asia and South Pacific Design Automation Conference*, 2019, pp. 550-556.

A. B. Kahng, **L. Wang** and B. Xu, "TritonRoute: An Initial Detailed Router for Advanced VLSI Technologies", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2018, pp. 81:1-81:8.

A. B. Kahng, C. Moyes, S. Venkatesh and **L. Wang**, "Wot the L: Analysis of Real versus Random Placed Nets, and Implications for Steiner Tree Heuristics", *Proc. ACM International Symposium on Physical Design*, 2018, pp. 2-9.

C. Han, K. Han, A. B. Kahng, H. Lee, **L. Wang** and B. Xu, "Optimal Multi-Row Detailed Placement for Yield and Model-Hardware Correlation Improvements in Sub-10nm VLSI", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2017, pp. 667-674.

P. Debacker, K. Han, A. B. Kahng, H. Lee, P. Raghavan and **L. Wang**, "MILP-Based Optimization of 2D Block Masks for Timing-Aware Dummy Segment Removal in Self-Aligned Multiple Patterning Layouts", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36(7) (2017), pp. 1075-1088.

P. Debacker, K. Han, A. B. Kahng, H. Lee, P. Raghavan and **L. Wang**, "Vertical M1 Routing-Aware Detailed Placement for Congestion and Wirelength Reduction in Sub-10nm Nodes", *Proc. ACM/IEEE Design Automation Conference*, 2017, pp. 1-6.

K. Han, A. B. Kahng, H. Lee and **L. Wang**, "Performance- and Energy-Aware Optimization of BEOL Interconnect Stack Geometry in Advanced Technology Nodes", *Proc. International Symposium on Quality Electronic Design*, 2017, pp.104-110.

A. B. Kahng, J. Li and **L. Wang**, "Improved Flop Tray-Based Design Implementation for Power Reduction", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2016, pp. 20:1-20:8.

K. Han, A. B. Kahng, H. Lee and **L. Wang**, "ILP-Based Co-Optimization of Cut-Mask Layout, Dummy Fill and Timing for Sub-14nm BEOL Technology", *Proc. SPIE Photomask Technology*, 2015, pp. 96350E:1-96350E:14.

ABSTRACT OF THE DISSERTATION

Improved Physical Design for Manufacturing Awareness and Advanced VLSI

by

Lutong Wang

Doctor of Philosophy in Electrical Engineering (Computer Engineering)

University of California San Diego, 2020

Professor Andrew B. Kahng, Chair

Increasing challenges arise with each new semiconductor technology node, especially in advanced nodes, where the industry tries to extract every ounce of benefit as it approaches the limits of physics, through manufacturing-aware design technology co-optimization and design-based equivalent scaling. The increasing complexity of design and process technologies, and ever-more complex design rules, also become hurdles for academic researchers, separating academic researchers from the most up-to-date technical issues.

This thesis presents innovative methodologies and optimizations to address the above challenges. There are three directions in this thesis: (i) manufacturing-aware design technology co-optimization; (ii) advanced node design-based equivalent scaling; and (iii) an open source academic detailed routing flow.

To realize manufacturing-aware design technology co-optimization, this thesis presents two works: (i) a multi-row detailed placement optimization for neighbor diffusion effect mitigation between neighboring standard cells; and (ii) a post-routing optimization to generate 2D block mask layout for dummy segment removal in self-aligned multiple patterning.

To achieve advanced node design-based equivalent scaling, this thesis presents two improved physical design methodologies: (i) a post-placement flop tray generation approach for clock power reduction; and (ii) a detailed placement approach to exploit inter-row M1 routing for congestion and wirelength reduction.

To address the increasing gap between academia and industry, this thesis presents two works toward an open source academic detailed routing flow: (i) a complete, robust, scalable and design rule-aware dynamic programming-based pin access analysis framework; and (ii) TritonRoute – the open source detailed router that is capable of delivering DRC-clean detailed routing solutions in advanced nodes.

This thesis concludes with a summary of its contributions and open directions for future research.

Chapter 1

Introduction

The past decade has seen tremendous changes in information technology, such as (i) rapid transition from 2G/3G to 4G/5G cellular network technologies; (ii) explosion of the Internet of Things (IoT); and (iii) massive deployment of cloud computing and fast networking. The semiconductor industry sits at the heart of technology. Even though Moore's Law scaling has become more costly and difficult, major players have all been keen to race toward the limits of physics, at the cost of increasing complexity in both design and manufacturing. To compensate for the slowdown of Moore's Law, extra efforts are made to extract the last drop of benefit from new technologies, making design and manufacturing even more complicated. Given such a scenario, this thesis presents several physical design methodologies and optimizations to address existing and future challenges in advanced VLSI.

1.1 New Challenges

Increasing challenges arise with each new semiconductor technology node, especially in advanced nodes, where the industry tries to extract every ounce of benefit as it approaches the limits of physics, through manufacturing-aware design technology co-optimization and design-based equivalent scaling. The increasing complexity of design and process technologies, and ever-more complex design rules, also become hurdles for academic researchers, separating academic researchers from the most up-to-date technical issues.

1.1.1 Manufacturing-Aware Design Technology Co-Optimization

In advanced technology nodes, aggressive device scaling, lithography limitations and process complexity bring new challenges in the physical design implementation flow. Figure 1.1 shows a roadmap of recent and future technology advancements.

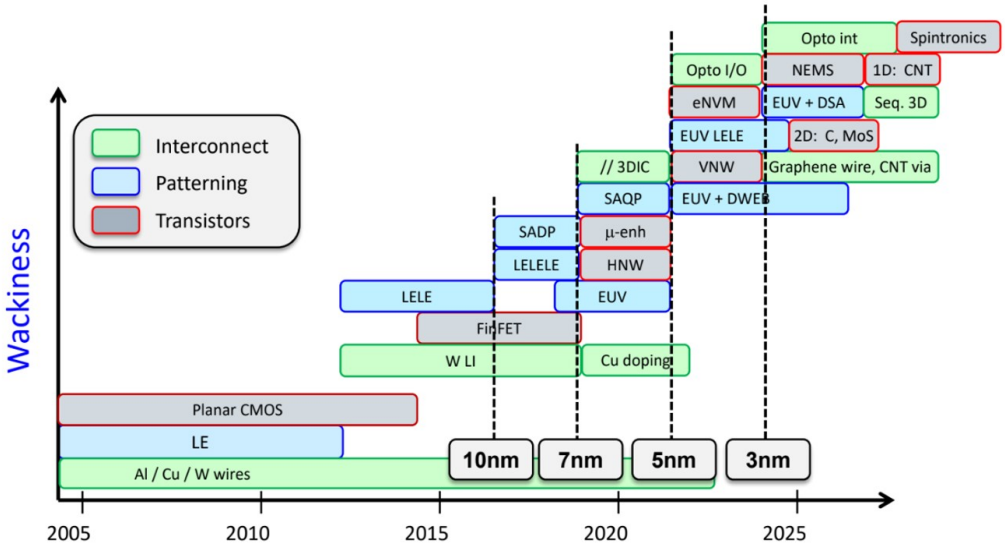


Figure 1.1: Roadmap of future technology [37].

In the front end, devices have made a transition from planar to FinFET and nanowire structures. The shrinking dimension approaches the fundamental limits of physics. Therefore, device behavior no longer depends on pre-characterized, independent geometrical parameters. Layout-dependent effect (LDE) arises from the proximity of devices, and significantly affects device performance. Pre-characterized libraries have difficulty considering such effects, causing model-hardware miscorrelation issues and resulting in yield loss. Improved physical design methodologies that consider and reduce the variability of front-end devices are critical in claiming the potential product quality and cost benefits at each new technology node.

In the back end, lithography limitations significantly complicate the manufacturing process. The industry has transitioned from single patterning (LE), and double patterning (LELE) to self-aligned double patterning (SADP) and self-aligned quadruple patterning (SAQP). New and complicated interconnects result in more gridded and limited layout patterns, leading to increased capacitance (resistance). To achieve

the full node scaling benefit, additional manufacturing steps are introduced to redistribute wire cuts, and to remove redundant metals. New physical design methodologies for the additional manufacturing steps are critical to achieve a better quality of result.

1.1.2 Advanced Node Design-Based Equivalent Scaling

Power, performance and area (PPA) are always the ultimate goals of the semiconductor industry. For decades, due to Moore's Law scaling, the semiconductor industry has enjoyed all of power, performance and area benefits without the need to trade off one for another. However, as billions and tens of billions of transistors are packed onto a tiny die, the 2013 ITRS roadmap [117] notes an increasing gap of design capability, as shown in Figure 1.2. While Moore's Law continued (at least until the year 2013) to deliver "available" scaling (i.e., geometric pitch scaling) of 2× per technology node, designers were only able to actually "achieve" a transistor density scaling of 1.6× since ∼2008. In addition to the design capability gap, there is an intrinsic trend of slowdown in geometric pitch scaling, and each new node provides only a limited amount of PPA improvements. Therefore, there is an increasing need and practice to extract more benefits from each technology node by *design-based equivalent scaling*. Design-based equivalent scaling refers to better physical design optimization methodologies, such as exploration of new cell architecture, and additional stages in the flow, etc. – without reliance on any change to the device, interconnect or manufacturing technologies that underlie the design enablement.

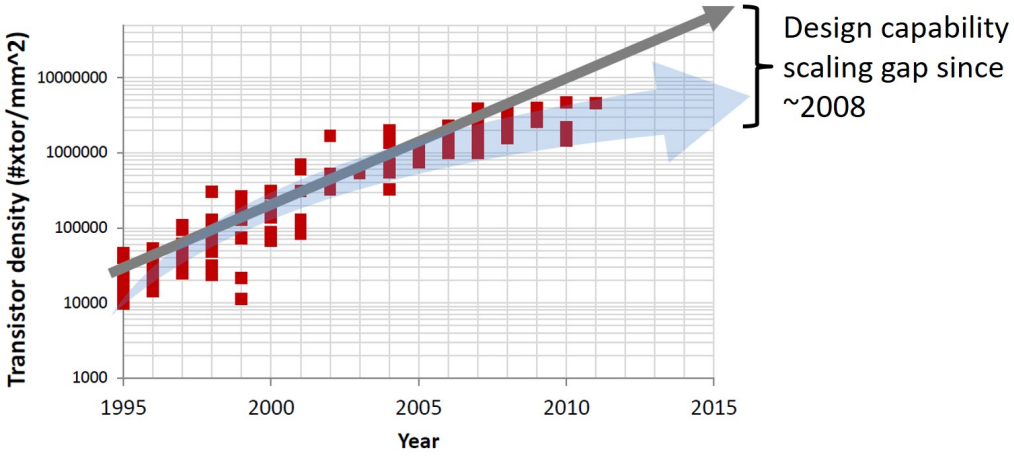


Figure 1.2: Design capability gap [49].

One of the major challenges is to directly reduce power consumption. Low-power design methodologies are vital to enable reduced power consumption at the same performance, or increased performance at the same power. In advanced nodes, physical scaling cannot provide the anticipated full scale of power benefits: (i) VDD and threshold voltage (V_{th}) essentially stop to scale; and (ii) leakage becomes worse with the shrinking channel length. Various types of flow optimizations – gate sizing and swapping, clock and power gating, etc. – have already been widely adopted. However, demands for mobile and IoT (Internet of Things) continue to drive the growth of integrated circuits (IC), creating more and stricter requirements for physical design that require innovative solutions.

One example is clock network power reduction, since the clock network typically has large power consumption due to its high switching activity, multi-level buffering and long wirelength. An application of flop trays (i.e., multi-bit flip-flops) can significantly reduce the number of sinks in a clock network, thus reducing the number of clock buffers, clock wirelength, and clock network power. Shared inverters within flop trays also reduce power at the flip-flop level. Further, careful design of the internal routing within a flop tray prevents hold buffer insertion between flops within the tray, especially along scan chains. This reduces the number of hold buffers, DFT (Design for Test) overheads, and potential placement congestion. However, large-size flop trays typically induce placement and routing congestion, and impose additional placement constraints on their fanin/fanout logic cones; this results in power overheads on datapaths. The "chicken-and-egg" loop between flop tray generation and placement optimization is a further challenge to flop tray-based design.

Another major challenge is to maintain sufficient density scaling by area shrinking. In advanced nodes, geometric scaling encounters the limitations of physics – neither devices nor the back-end-of-line stack can scale linearly with Moore's Law. To address these challenges, the industry has seen rapid innovation in standard-cell architecture starting at the foundry $10nm$ (N10) node, and accelerating into the N7/N5 enablement. As examples of cell architecture evolution, metal layers below M1 are used for internal routing within a standard cell, or horizontal M1 power/ground pins are removed to gain additional routing resources for inter-cell routing. These new cell architectures, wherein inter-row M1 routing is allowed, force new consideration of vertical alignment of cells.

1.1.3 The Widening Academia – Industry Gap

New technology nodes come with smaller feature sizes, while fundamental physical (lithographic patterning, CMP, reliability, variability, etc.) and circuit (crosstalk, delay, etc.) limitations remain. As a result, ever-more complex design rules must be comprehended and satisfied at various stages of the physical implementation flow.

One major challenge is detailed routing. Detailed routing is a dead-or-alive critical element of advanced node enablement, but only a few academic works even attempt to present an end-to-end detailed routing flow, and almost no works make claims to viability in the real-world IC physical design (P&R) context. Therefore, most detailed routing research works focus on incremental improvements, such as crosstalk or a specific part of new-technology contexts. Also, comparison between these works is difficult, since there is a lack of any common platform that each work can be based on. Further, lack of a basic platform results in near-impossibility of the direct application of academic codes, especially given that commercial tools and industrial designs satisfy far more, and more complex, design rules than any academic tool.

Given the above, there is a widening gap between academic research and industry. This widening gap, in turn, prevents academic researchers from making future practical innovations.

1.2 This Thesis

This thesis presents innovative optimizations and design methodologies to address various challenges in physical design. Figure 1.3 illustrates the scope and organization of this thesis.

To realize manufacturing-aware design technology co-optimization, this thesis presents two works: (i) a multi-row detailed placement optimization for neighbor diffusion effect mitigation between neighboring standard cells; and (ii) a post-routing optimization to generate 2D block mask layout for dummy segment removal in self-aligned multiple patterning.

To achieve advanced node design-based equivalent scaling, this thesis presents two improved physical design methodologies: (i) a post-placement flop tray generation approach for clock power reduction; and (ii) a detailed placement approach to exploit inter-row M1 routing for congestion and wirelength reduction.

To address the increasing gap between academia and industry, this thesis presents two works toward an open source academic detailed routing flow: (i) a complete, robust, scalable and design rule-aware dynamic programming-based pin access analysis framework; and (ii) TritonRoute – the open source detailed router that is capable of delivering DRC-clean detailed routing solutions in advanced nodes.

The remainder of this thesis is organized as follows.

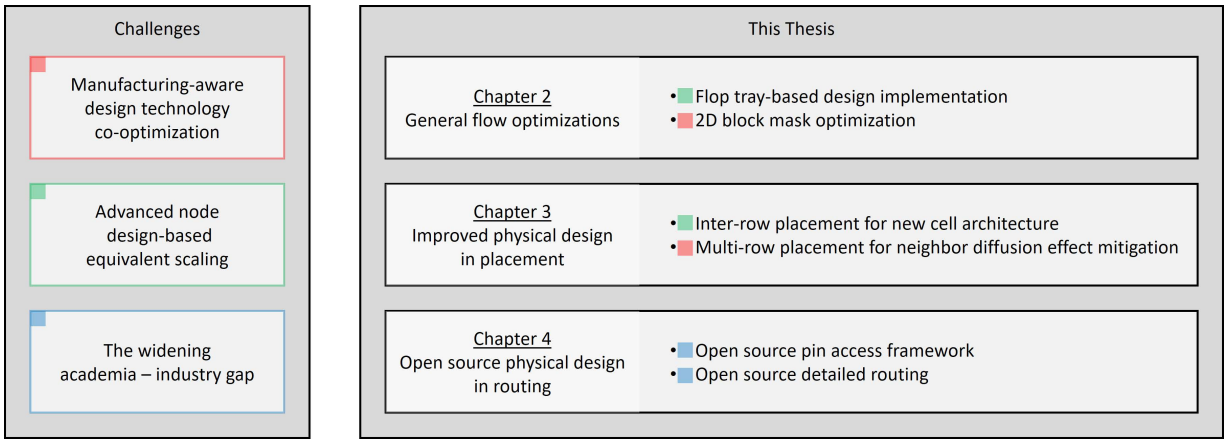


Figure 1.3: Scope and organization of this thesis.

- Chapter 2 presents two physical design methodologies that can be incorporated into the conventional place-and-route flow. First, we present a post-placement flop tray generation approach for clock power reduction. Our approach consists of a capacitated K-means iterative optimization and a *Silhouette*-based flop clustering evaluation and selection method. The capacitated K-means clustering includes a min-cost flow clustering, and a linear programming-based placement optimization, that considers flop tray aspect ratios and relative location displacement of timing-critical start-end pairs. Our optimization is able to convert more single-bit flops into flop trays, with smaller datapath power overhead as compared to a logical clustering flow implemented with commercial tools, and is aware of useful skew. We achieve up to 32% and 90% reductions of total block power and clock power as compared to implementations using only single-bit flops; and up to 16% and 40% reductions of total block power and clock power as compared to a commercial tool-based flow with logical clustering. Our optimization also achieves 13% clock power reduction on average as compared to previous works. Second, we present a post-routing optimization to generate 2D block mask layout

for dummy segment removal in self-aligned multiple patterning. We develop a mixed integer linear programming-based methodology to optimize 2D block mask layout that considers realistic block mask rules, timing impact of dummy fills and metal density constraints. Our optimization includes a timing model to evaluate the performance impact on a per-segment basis, and a co-optimization technique for both cut and block masks. We perform experiments using different sets of block mask rules and verify our optimization using different clip sizes. We further perform experiments with different metal density constraints and show the performance impact.

- Chapter 3 presents two methodologies targeted to the placement stage of physical design. First, we present a detailed placement methodology for neighbor diffusion effect mitigation and better model-hardware correlation. Our methodology consists of optimal dynamic programming-based single-row/double-row and multi-row detailed placement optimizations that considers displacement and HPWL. Our optimization supports movable and fully-reorderable multi-height cells, including reordering between multi-height cells and inter-row cell movements. Our optimization maximizes the *diffusion step* reduction to mitigate the neighbor diffusion effect in order to reduce model-hardware miscorrelation and yield loss, with up to 98% inter-cell *diffusion step* reduction. Our formulation is further extended for a potential timing-aware optimization that leads to 6$\times$ increase in *intentional steps* around timing-critical cells. Second, we present a detailed placement methodology to reduce congestion and wirelength. Our methodology consists of a mixed integer linear programming-based optimization for two cell architectures that are relevant in sub-10nm process nodes, and considers and exploits inter-row M1 routing. We adopt a distributed, window-based optimization to overcome the runtime limitation, achieving up to 6.4% total routed wirelength reduction, and up to 14.4% #via12 reductions, with no adverse timing impact.

- Chapter 4 presents two works towards a complete, end-to-end academic detailed routing flow targeting advanced nodes. First, we present a multi-level, standard cell- and instance-based, complete, robust, scalable and design rule-aware pin access analysis framework. The proposed framework includes pin-based access point generation, boundary conflict-aware access pattern generation and cluster-based access pattern selection based on dynamic programming. The work achieves 100%

DRC-clean pin access and demonstrates a superior final detailed routing solution as compared to the best known results using the ISPD-2018 initial detailed routing benchmark suite. Second, we present a complete, end-to-end academic detailed router, TritonRoute. Our router is capable of comprehending connectivity and design rule constraints using industry-standard formats. Our router consists of an in-memory router database that complies with the LEF/DEF data models, a pin access analysis engine, a track assignment engine, a detailed routing engine, and a design rule checking engine. The detailed routing engine includes a ripup-and-reroute-based path search engine, capable of avoiding potential design rule violations, as well as working around existing design rule violation markers. The router is evaluated using the official ISPD-2018 contest benchmark suite, demonstrating an extremely low level of DRCs. Overall, TritonRoute improves wirelength by up to 0.8% (avg. 0.4%), via count by up to 16.1% (avg. 9.3%) and DRCs by up to 100% (avg. 92.0%) as compared to the known best detailed routing solutions.

- Chapter 5 concludes the thesis and gives future directions in physical design methodologies.

Chapter 2

General Flow Optimizations

This chapter presents two physical design methodologies that can be incorporated into the conventional place-and-route flow. First, we present a post-placement flop tray generation approach for clock power reduction. Our approach consists of a capacitated K-means iterative optimization and a *Silhouette*-based flop clustering evaluation and selection method. The capacitated K-means clustering includes a min-cost flow clustering, and a linear programming-based placement optimization, that considers flop tray aspect ratios and relative location displacement of timing-critical start-end pairs. Our optimization is able to convert more single-bit flops into flop trays, with smaller datapath power overhead as compared to a logical clustering flow implemented with commercial tools, and is aware of useful skew. We achieve up to 32% and 90% reductions of total block power and clock power as compared to implementations using only single-bit flops; and up to 16% and 40% reductions of total block power and clock power as compared to a commercial tool-based flow with logical clustering. Our optimization also achieves 13% clock power reduction on average as compared to previous works. Second, we present a post-routing optimization to generate 2D block mask layout for dummy segment removal in self-aligned multiple patterning. We develop a mixed integer linear programming-based methodology to optimize 2D block mask layout that considers realistic block mask rules, timing impact of dummy fills and metal density constraints. Our optimization includes a timing model to evaluate the performance impact on a per-segment basis, and a co-optimization technique for both cut and block masks. We perform experiments using different sets of

block mask rules and verify our optimization using different clip sizes. We further perform experiments with different metal density constraints and show the performance impact.

2.1 Improved Flop Tray-Based Design Implementation for Power Reduction

Clock network optimization is critical in modern SoC designs due to the following reasons: (i) clock network typically has large power due to its high switching activity; (ii) clock skew and latency (with on-chip variation) have significant impact on design performance; and (iii) clock network routing consumes routing resources and can cause routing congestion. In this work, we study design optimization with *flop trays*[1] (i.e., macro cells of multi-bit flip-flops), where the application of flop trays can significantly reduce the number of sinks (similar to [4]) and thus can result in an improved clock network. Further, careful design of the internal routing within a flop tray prevents hold buffer insertion between flops within the tray, especially along scan chains. This reduces the number of hold buffers, DFT (Design for Test) overheads, and potential placement congestion.

Flop tray potential benefits. It is intuitively reasonable that more clock power reduction can be achieved by using larger sizes (i.e., greater number of bits) of flop trays. As a motivating "thought experiment", consider a clock tree with N sinks and fanout of f at each level: the total number of (internal) clock buffers between the clock root and the clock pins of sinks (i.e., flops, flop trays) is $\approx \frac{N-1}{f-1}$. If we could replace all single-bit flops with K-bit flop trays, the number of clock buffers would reduce to only $\approx \frac{N/K-1}{f-1}$ (e.g., using 64-bit flop trays to replace single-bit flops could reduce the number of clock buffers by up to 98.4% (= $\frac{N-N/64}{N-1} \approx \frac{63}{64}$)). Furthermore, Figure 2.1 illustrates how inverters for clock signals can be shared among flops in a flop tray, resulting in power and area reduction as compared to multiple single-bit flops. These power and area reductions would also increase with flop tray sizes.

Current approaches and their limitations. Flop tray-based implementation is very challenging due to the following reasons. (1) In advanced nodes, flops (including single-bit flops and flop trays)

[1]Terminology: A flop tray is synonymous with a multi-bit flip-flop (MBFF); we use "flop" as a synonym for "flip-flop".

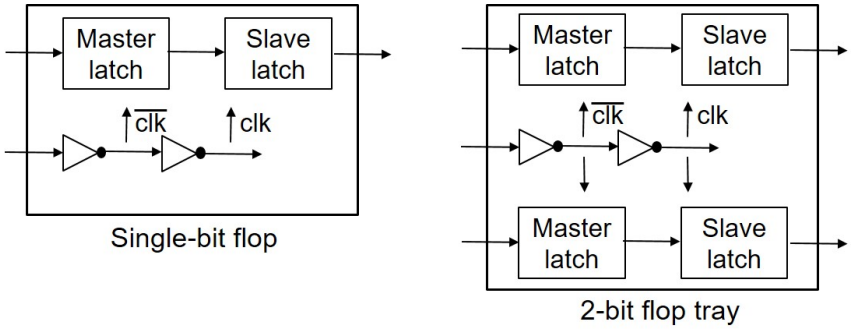


Figure 2.1: Two inverters for the clock signal are shared between the two flops in a 2-bit flop tray.

typically occupy a large portion of the entire block area due to their large sizes.[2] Moreover, flop trays can have high aspect ratios (e.g., a 64-bit flop tray may be implemented as a 4×16 array of flops, with much greater width than height); flop tray size and shape have been ignored by previous literature on multi-bit flop optimization [65][70][96] and flop clustering [13][82]. Flop trays with large area and high aspect ratio make placement optimization very difficult [17][76]. (2) Clustering of flops imposes additional placement constraints on their fanin and fanout logic cones, which is highly likely to degrade the placement solution quality [76]. (3) Usage of flop trays can easily cause routing congestion. (4) Clustering of single-bit flops into flop trays has a large impact on timing and limits the application of useful skew optimization. Most previous works study small-size flop trays, and do not fully address the above challenges in their optimization approaches. Crucially, further achievable benefits of using large-size flop trays are not exploited by previous works. To maximize obtained benefits from flop tray deployment, our present work proposes a flop tray-based optimization that comprehends arbitrary flop tray sizes. (Below, we show results with flop tray size up to 64 bits.)

A common practice for flop tray-based implementation is to cluster flops during the synthesis stage based on logic functions of the design, along with clock domain and clock gating information. We refer to this as *logical clustering* in the following discussion. However, flop tray generation without physical information can result in placement and routing congestion and degrade place-and-route (P&R) solution qualities. Figure 2.2 shows examples where flop tray-based implementations with logical clustering during

[2] As an example, a minimum-size inverter occupies two placement sites; a single-bit flop occupies 18 sites; and a 64-bit flop tray can occupy 244 sites in width and four cell rows in height. Due to their large sizes, flops and flop trays can consume a substantial fraction of overall cell area (e.g., *VGA* from *OpenCores* [124] has 30% of its instances as flops, which accounts for 51% of the total cell area).

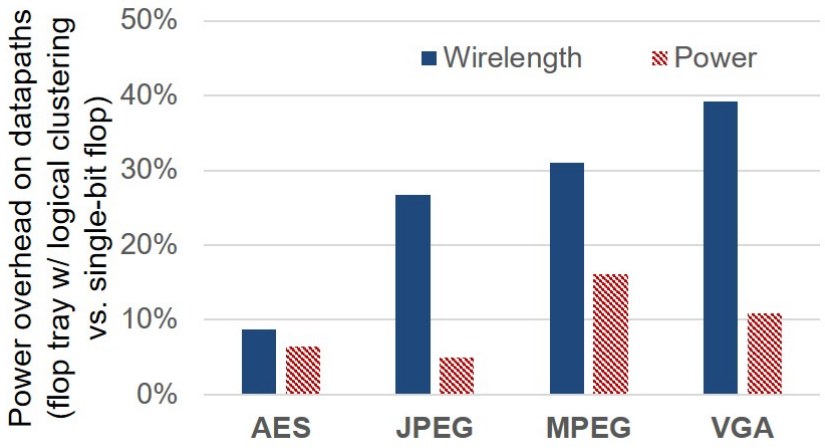


Figure 2.2: Wirelength and power overheads on datapaths due to flop tray-based implementations compared to implementations using only single-bit flops. Technology: 28FDSOI. Designs are from *OpenCores* [124].

synthesis stage can result in 8% – 39% wirelength overhead and 5% – 16% power overhead on datapaths after detailed routing even at a low conversion ratio from single-bit flops to flop trays. (In the example, numbers of flops and flop trays in flop tray-based implementations, as percentages of flop numbers in implementations with single-bit flops, are 43%, 37%, 41% and 45% for *AES*, *JPEG*, *MPEG* and *VGA*, respectively.) This degrades power benefits from flop tray deployment. Therefore, feedback loops and iterations are required between early-stage flop clustering and P&R optimization, which can significantly increase design time [17]. Furthermore, although splitting large flop trays into smaller trays or single-bit flops during placement and/or routing can mitigate the congestion and power penalty, benefits of applying flop trays then become limited. In addition, the capability of logical clustering to realize flop tray benefits can be limited according to attributes of the given design. Designs with few multi-bit signals may not derive substantial benefits from flop tray deployment. On the other hand, designs with many multi-bit signals might use flop trays aggressively, with large-size flop trays in particular causing placement and routing congestion.

Our approach. In this work, we focus on post-placement flop tray optimization.[3] We first place the design with all single-bit flops, where the placement solution is considered to give *ideal* locations

[3]Other low-power clocking styles and methodologies (e.g., pulsed-latch, register arrays, and rotary clock) are not the focus of this work.

of individual flops and combinational cells (given that there are no additional constraints induced by flop clustering). We then cluster flops based on the placement solution. In this way, we resolve the "chicken-and-egg" loop between early-stage flop tray generation and placement optimization of flop trays. However, post-placement flop tray generation such as ours must carefully comprehend different flop tray sizes and aspect ratios; it must also minimize perturbation on datapath placement and timing degradation (otherwise, the assumption of "ideal" combinational cell placement does not hold).

To maximize the benefits of applying flop trays while minimizing the perturbation on the initial placement solution, we propose a *capacitated K-means optimization* which iteratively executes min-cost flow to cluster single-bit flops into flop trays, and a linear programming-based optimization to place flop trays. Based on the proposed capacitated K-means optimization, we achieve a solution (including flop clustering and flop tray placement) for each given flop tray size and AR. We then formulate an integer linear program (ILP) to select the best combination of flop tray solutions. In addition to minimization of displacement of flops (i.e., from the initial single-bit flop location to the flop location in a flop tray), our optimization is also aware of timing-critical start-end flop pairs. Specifically, we minimize the *relative location* displacement of timing-critical start-end pairs to minimize the timing impact from flop tray insertion.

The contributions of this work are as follows.

- We propose a capacitated K-means iterative optimization that applies (i) min-cost flow based clustering, and (ii) LP-based placement optimization to generate flop trays with various sizes (e.g., 4-bit, 16-bit and 64-bit) at the post-placement stage.
- Our optimization is aware of flop tray aspect ratios and *relative location* displacement of timing-critical start-end pairs.
- We apply a new *Silhouette*-based metric in addition to displacement distance to evaluate flop clustering solutions.
- Our optimization is able to convert more single-bit flops into flop trays, but with smaller datapath power overhead, as compared to a logical clustering flow implemented with commercial tools.

- We achieve up to 32% and 90% reductions of total block power and clock power as compared to implementations using only single-bit flops; and up to 16% and 40% reductions of total block power and clock power as compared to a commercial tool-based flow with logical clustering. We also achieve 13% clock power reduction on average as compared to the previous work in [48].

- We evaluate the benefit (i.e., leakage reduction) of useful skew optimization on flop tray-based design and propose a useful skew-aware clustering to maximize such benefit.

The remainder of this section is organized as follows. Section 2.1.1 reviews related works on flop tray optimization. Section 2.1.2 describes our capacitated K-means optimization flow. In Section 2.1.3, we describe our experimental setup and results. Section 2.1.4 concludes and gives directions for ongoing work.

2.1.1 Related Work

In this section, we review flop clustering and flop tray (multi-bit flop) generation approaches proposed in previous works. We classify these approaches into two categories: (i) early-stage flop tray generation, and (ii) flop tray generation during and/or after placement.

Several early works propose flop tray generation at early design stages. Kretchmer et al. [56] propose register banking during logic synthesis. They create Liberty models of flop trays, which can be used by logic synthesis tools. But, flop tray generation during synthesis has only logic topology as its main lever, and the lack of physical information can result in a sub-optimal clustering solution, with degraded timing and larger power. To address this, Hou et al. [45] further propose register banking removal based on routing congestion and timing information. However, such a "(flop) clustering at early stage and (flop tray) removal at late stage" flow is not able to effectively exploit the benefits of flop tray usage. Thus, many other works propose flop tray generation during and/or after placement.

Yan et al. [103] generate flop trays at the post-placement stage. They first construct an intersection graph based on routing length and congestion constraints derived from an initial placement solution with single-bit flops. They then perform minimum-clique partitioning to reduce the number of flop trays. Lin et al. [64] use progressive window-based optimization to improve the methodology proposed in [103] by

considering given flop tray sizes. They solve the clustering problem by finding K-cliques and maximum independent sets in a merging graph constructed based on feasible-location regions of flops. Similarly, Wang et al. [96] use clique partitioning to identify a set of non-conflicting cliques. Jiang et al. [48] propose an efficient post-placement flop tray generation technique using interval graphs and a pair of linearized sequences. Liu et al. [70] also propose flop clustering based on an intersection graph. In addition to reducing the number of flop trays, they apply agglomerative clustering to minimize displacements of flops, wirelength and clock power. More recently, Lin et al. [65] develop a clock tree-aware in-placement flop tray generation technique. They build an intersection graph considering clock latency, wirelength and timing, then iteratively perform flop tray generation and timing-driven incremental placement. Xu et al. [101] propose an analytical clustering score for flop tray generation, permitting seamless integration with the traditional wirelength objective. Tsai et al. [94] propose to generate flop trays during placement. During analytical global placement, they guide placement of flops (to enable flop tray generation) with additional bonding force (resembling ionic bonds in chemistry). Other works optimize flop trays with awareness of crosstalk [46], clock gating [71], etc.

In addition to flop tray-based design, flop and/or latch clustering optimizations have been widely applied in previous works for clock tree and latch placement optimization. Mehta et al. [74] propose a clustering algorithm to obtain approximately load-balanced clusters and construct clock trees so as to minimize skew. Papa et al. [82] apply K-means clustering algorithm to minimize latch displacement during a physical synthesis optimization. Deng et al. [13] propose a register clustering methodology in generating the leaf-level topology of the clock tree to reduce clock power consumption.

We summarize our algorithmic and methodological improvements, as compared to previous works, as follows.

- None of the previous in-placement and post-placement approaches study flop tray optimization with large-size flop trays (e.g., 64-bit flop trays). The ARs of flop trays are ignored (indeed, many previous works treat flop trays essentially as points in their optimizations). By contrast, our optimization considers arbitrary flop tray sizes and is aware of flop tray ARs.

- Most previous works assume a feasible displacement region for each flop. However, such an assumption does not comprehend the movements of fanin/fanout flops, which can be either pessimistic or optimistic. In addition, such an assumption essentially precludes exploiting benefits of useful skew. By contrast, our approach considers timing path-aware timing impact of flop displacement; specifically, we minimize the *relative location* displacement of timing-critical start-end pairs. We also propose a useful skew-aware optimization flow to maximize such benefit.

- Previous works use local search to cluster flops into flop trays. However, due to capacity constraints of flop trays, such local search can result in outliers with large displacement distances. By contrast, in this work we apply a more globally-aware optimization based on (i) a capacitated K-means formulation (with iterative min-cost flow-based clustering and LP-based placement optimization), and (ii) a practically scalable ILP-based matching and selection of flop tray solutions to globally optimize flop clustering with given capacity constraints (i.e., flop tray sizes).[4]

2.1.2 Methodology

We now describe our optimization methodology for flop tray generation and placement. Figure 2.3 illustrates our overall optimization flow, where we integrate our flop tray optimization (steps in blue boxes) into a conventional SP&R (synthesis, place, and route) flow. To address the "chicken-and-egg" loop between flop tray generation and placement optimization, we first perform an initial placement with only single-bit flops, where the placement is considered to be "optimal" with no placement constraints induced by flop clustering. We note that since the initial placement is timing- and congestion-aware, minimizing subsequent perturbations can mitigate potential congestion due to flop trays, as well as minimize timing impacts. Further, to comprehend multiple flop tray sizes and ARs, we perform flop tray optimization for each flop tray choice (i.e., a $\{$size, AR$\}$ combination). Last, we perform an integer linear programming (ILP)-based optimization to select the optimal combination of flop trays and their placement solutions.[5]

[4] Our ILP runtime (*CPLEX 12.6*) is less than one minute on the *VGA* testcase [124] (with 17K flops and 1000 timing-critical paths) with five candidate flop tray sizes studied in Section 2.1.2 and Section 2.1.3 below, using 20 threads on a $2.5GHz$ Intel Xeon server.

[5] Our separate study shows that due to high runtime complexity, it is practically infeasible for our current approach to optimize flop clustering and flop tray placement considering all possible flop tray candidate sizes simultaneously. We therefore perform a two-step optimization in this work.

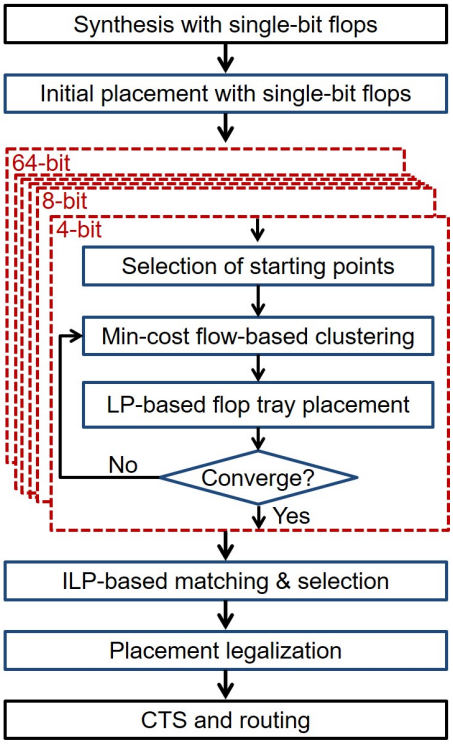


Figure 2.3: Overall optimization flow of flop tray generation.

We state our post-placement flop tray generation problem as: **Given** an initial placement solution with only single-bit flops, flop tray choices, and timing constraints, **cluster** single-bit flops into flop trays and **determine** the placement location of each flop tray, **such that** total block power (including clock power and power of sequential cells (i.e., flops and flop trays) and combinational cells) is minimized after routing.

The following subsections describe our capacitated K-means clustering and our ILP-based selection of flop tray solutions. Table 2.1 lists the notations used in our discussion.

Capacitated K-Means Clustering

We first address the following, narrower problem: **Given** an initial placement solution with all single-bit flops (i.e., N single-bit flops), and $\lceil N/K \rceil$ K-bit flop trays with fixed AR, **cluster** the single-bit flops into flop trays and **determine** the placement location of each flop tray, **such that** the total displacement of flops is minimized.

Table 2.1: Description of notations used in our formulation.

Term	Meaning
t_i	i^{th} flop tray
e_i	binary indicator whether t_i is used
w_i	cost of using tray t_i
f_{ij}	j^{th} flop of t_i
h_l	l^{th} single-bit flop
$b_{l,ij}$	binary indicator whether h_l is matched to f_{ij}
(X_i, Y_i)	center location of t_i
(x'_{ij}, y'_{ij})	relative center location of f_{ij} w.r.t. the center of t_i
(x_l, y_l)	optimal location of h_l
$(d_{l,ij}, d_{l,ij})$	Manhattan distance between h_l and f_{ij}

To address this problem, we propose a capacitated K-means algorithm [55]. (As noted above, K-means clustering algorithms have also been applied to flop (or latch) clustering in previous works [13][82].) There are two steps in a standard K-means algorithm: (i) clustering, and (ii) updating the center location of each cluster. We associate these two steps with: (i) matching of single-bit flops to flop slots in flop-trays, and (ii) updating the locations of flop trays. We propose a min-cost flow to address (i), and a linear programming (LP)-based optimization to address (ii). We iterate between these two steps until convergence (i.e., no further displacement reduction can be achieved, or a maximum number of iterations (= 35 in our experiments below) is reached).

In our capacitated K-means clustering, we use an algorithm that is similar to K-means++ [5] to select the starting points. Selection of $\lceil N/K \rceil$ starting points for clustering is described in Algorithm 1. In Algorithm 1 we calculate center-to-center distances between single-bit flops. To comprehend the aspect ratio of flop trays, we scale the horizontal distance by (1/AR) (= height/width) of the given flop tray.

Algorithm 1 Selection of starting points.

1: Randomly select one flop among single-bit flops
2: For each flop h_l, calculate the total Manhattan distance (d_l) from h_l to all selected flops
3: Randomly select one new flop with probability d_l
4: Repeat Steps 2 and 3 until $\lceil N/K \rceil$ flops are selected

These selected starting points serve as initial locations of flop trays. We then apply a min-cost flow to achieve *capacitated clustering* of flops. Our min-cost flow is illustrated in Figure 2.4. To construct

the flow instance, we create a node for each single-bit flop h_l. For each flop tray t_i, we further create K nodes for its K slots, $f_{i1} \ldots f_{iK}$. For each edge between a pair of h_l and f_{ij}, we set its capacity as 1 and its cost as the Manhattan distance between h_l and f_{ij}. Here, we directly calculate the Manhattan distance between single-bit flops and flop slots without any scaling. Finally, we create one source and one sink, and assign edges connected to them with capacity as 1 and cost as 0, as illustrated in Figure 2.4. Notice that by considering the distances between the locations of single-bit flops and flop slots in flop trays, our min-cost flow optimization is explicitly aware of physical information (in particular, dimensions and ARs) of the given flop trays.

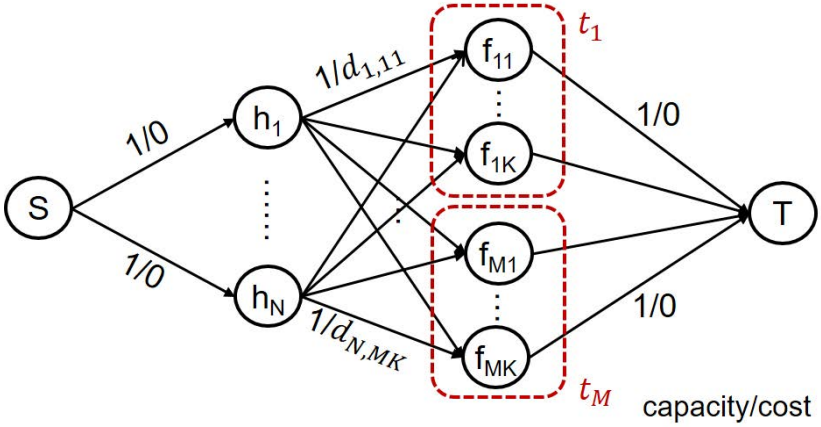


Figure 2.4: Example of min-cost flow with K-bit flop trays.

Based on the capacitated K-means clustering solution from the min-cost flow, we formulate a linear program (shown as follows) to determine the flop tray locations that achieve minimum total displacement of flops. These placement locations of flop trays will serve as starting points for the next iteration of clustering.

$$\text{Minimize} \quad D \tag{2.1}$$

$$\text{Subject to } |X_i + x'_{ij} - x_l| + |Y_i + y'_{ij} - y_l| = d_l \quad \forall h_l \tag{2.2}$$

$$\sum_l d_l = D \tag{2.3}$$

Constraint (2.2) calculates the displacement for each flop (d_l), and the objective (2.3) seeks to minimize the total displacement over all flops.

We iterate between the min-cost flow-based clustering and the LP-based flop tray placement until no further displacement reduction is achievable (i.e., no flop trays move between two consecutive iterations).

To confirm benefits from awareness of flop tray ARs, we show in Figure 2.5 representative clustering solutions from (i) the classic K-means approach, which treats each flop tray as a point, and (ii) our min-cost flow-based clustering, which is aware of flop tray ARs. We observe that our clustering solution more closely matches the AR of given flop trays. Further, classic K-means without awareness of flop tray AR can result in 2× increase in average displacement from the "ideal" single-bit flop placement; this is likelier to incur datapath power and timing overheads.

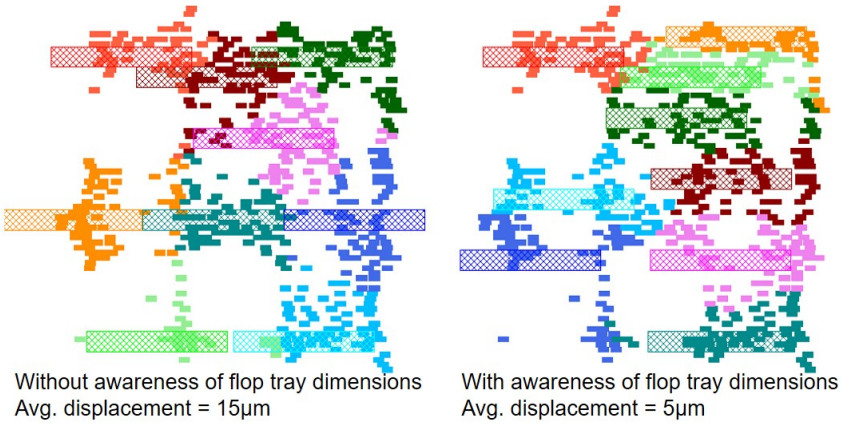


Figure 2.5: Clustering solutions into 64-bit flop trays (i) without awareness of flop tray aspect ratio and dimensions, and (ii) with awareness of flop tray aspect ratio and dimensions. Design: *AES* (530 single-bit flops). Technology: 28FDSOI.

In our capacitated K-means algorithm, as with K-means approaches in general, the selection of starting points has a strong impact on the final solution quality. We adapt the Silhouette metric [86] and use Equation (2.4) to evaluate the solution quality of generated starting points.[6]

$$func(h_l) = \frac{min_{i' \neq i,j'}(d_{l,i'j'}) - d_{l,ij}}{max(d_{l,ij},\ min_{i' \neq i,j'}(d_{l,i'j}))} \tag{2.4}$$

[6]As presented in [86], the Silhouette value is a measure of how similar an object is to its own cluster, compared to other clusters. A general Silhouette value is defined as $s(i) = \frac{b(i)-a(i)}{max(a(i),b(i))}$, where $a(i)$ is the average dissimilarity (e.g., average distance) of i with all other data within the same cluster, and $b(i)$ is the lowest average dissimilarity (e.g., minimum average distance) of i to the data in any other cluster other than its own. By definition, $-1 \leq s(i) \leq 1$, and a larger Silhouette value indicates a better clustering solution. In this work, data are slots of flop trays, and dissimilarities are measured by distances.

where h_l is matched to f_{ij}. The dissimilarity within a cluster is measured by the displacements of each of the cluster's assigned flops h_l. The dissimilarity between a given cluster and other clusters is measured by the distances between assigned flops h_l and the nearest flop-tray slot in another cluster to which h_l is not assigned.

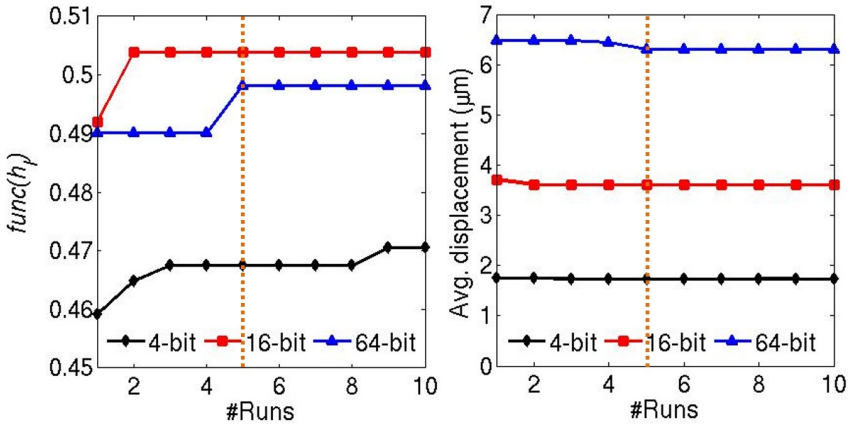


Figure 2.6: Best clustering solution (i.e., $func(h_l)$ (left) and displacement (right)) with multiple runs (numbers of runs are shown in the x-axis).

We apply a multistart strategy to improve the selection of starting points. Multiple runs (five in our experiments) of the procedure in Algorithm 1 are each followed by a small number (15 in our experiments) of iterations between the min-cost flow and LP-based placement optimization. We then select the solution with the highest average $func(h_l)$ value and proceed with capacitated K-means iterations until convergence. Figure 2.6 shows a typical improvement of the average value of $func(h_l)$ (left) and the average displacement (right) with increased number of runs. In our studies, the improvement of $func(h_l)$ and displacement typically saturates after five runs. Thus, the experiments reported below apply five multistarts to mitigate the impact of starting point selection.

ILP-Based Matching Optimization

The next step of our optimization approach addresses the following problem: **Given** candidate flop trays with various capacities, each with a fixed placement location, **select** the optimal subset of the candidate flop trays, and **determine** a mapping of single-bit flops into slots of selected candidate flop trays, **such that** (i) every single-bit flop is mapped to a slot of a selected flop tray (including flop trays with one

bit, i.e., no clustering), and (ii) a weighted sum of the total displacement of flops, relative displacement of timing-critical start-end pairs, and total flop tray costs is minimized.

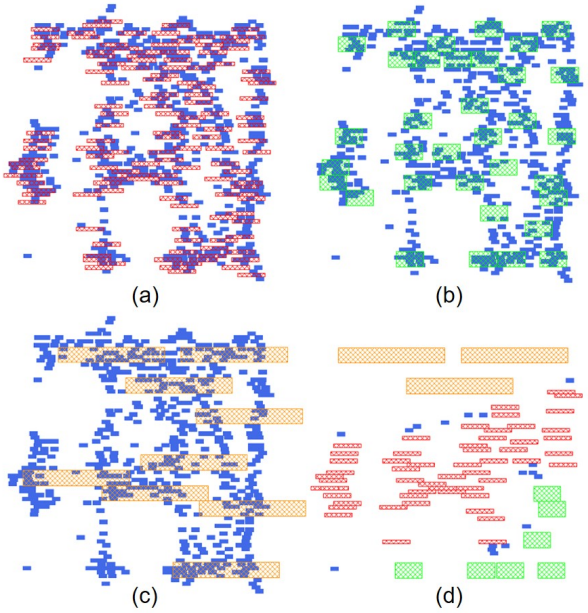


Figure 2.7: Example of our ILP-based optimization.

As discussed in Section 2.1.2, we run capacitated K-means clustering with different flop tray sizes and ARs, and use these flop trays together with their optimized placement locations as inputs ("candidates") for an ILP-based matching optimization. Our ILP-based optimization selects an optimal subset of candidate flop trays with various flop tray sizes as our final solution. As an example, Figures 2.7(a) – (c) show solutions of flop trays with fixed sizes and ARs on the *AES* testcase. Specifically, Figures 2.7(a) – (c) respectively show solutions with only 4-bit flop trays (flop trays are in red, #flop trays = 133, average displacement = $2\mu m$), only 16-bit flop trays (flop trays are in green, #flop trays = 34, average displacement = $3\mu m$), and only 64-bit flop trays (flop trays are in orange, #flop trays = 9, average displacement = $5\mu m$). Figure 2.7(d) shows the final solution, i.e., solution with a combination of single-bit flops and 4-bit, 16-bit and 64-bit flop trays (#flops + #flop trays = 81, average displacement = $2\mu m$). Our objective is to minimize a weighted sum of total displacement of flops, relative displacement of timing-critical start-end flop pairs, and total flop tray cost. Relative displacement of a timing-critical start-end flop pair is illustrated in Figure 2.8. As an improvement to previous approaches, we comprehend timing impact of flop tray

generation considering timing-critical paths (i.e., start-end pairs). Specifically, if the flop tray generation moves two flops towards each other, combinational cells in the logic cone between the flops are forced to be placed in a more compact region, which results in congestion and distortion of the placement and routing. Alternatively, if the flop tray generation moves two flops away from each other, timing paths between the two flops will tend to have longer wirelength, degrading timing. We therefore seek to minimize the *relative displacement* of flops that are timing-critical start-end pairs.

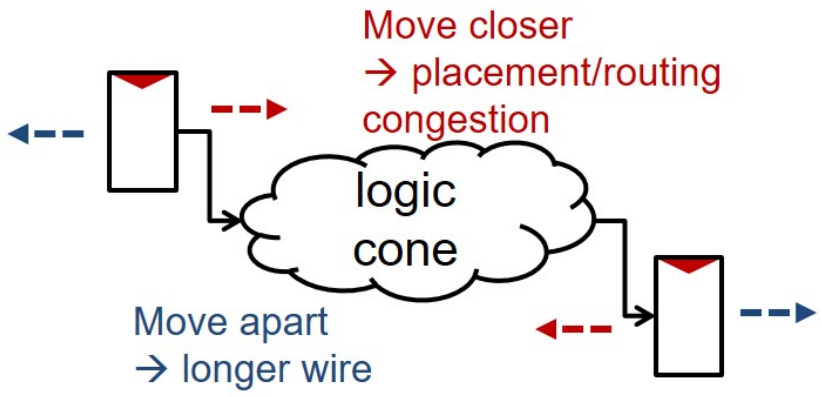


Figure 2.8: Illustration of the timing impact due to relative displacement between timing-critical start-end flop pairs.

Our ILP to select the optimal combination of flop tray solutions with various sizes and ARs is given below.[7]

[7]Note that our ILP can be extended to be aware of clock gating, clock domain and useful skew optimization, etc. with additional constraints. Section 2.1.3 briefly describes a useful skew-aware extension and corresponding benefits.

$$\text{Minimize } \alpha \cdot W + D + \beta \cdot Z \tag{2.5}$$

$$\text{Subject to } |\sum_{ij} (X_i + x'_{ij} - x_l) \cdot b_{l,ij}|$$

$$+|\sum_{ij} (Y_i + y'_{ij} - y_l) \cdot b_{l,ij}| = d_l \quad \forall l \tag{2.6}$$

$$\sum_{l} d_l = D \tag{2.7}$$

$$d_l \leq d_{max} \quad \forall l \tag{2.8}$$

$$|\sum_{ij} (X_i + x'_{ij} - x_l) \cdot b_{l,ij} - \sum_{i'j'} (X_{i'} + x'_{i'j'} - x_{l'}) \cdot b_{l',i'j'}|$$

$$+|\sum_{ij} (Y_i + y'_{ij} - y_l) \cdot b_{l,ij} - \sum_{i'j'} (Y_{i'} + y'_{i'j'} - y_{l'}) \cdot b_{l',i'j'}|$$

$$= z_{ll'} \qquad \forall (h_l,\ h_{l'}) \in \text{timing-critical paths} \tag{2.9}$$

$$\sum_{(h_l,h_{l'}) \in cri\_paths} z_{ll'} = Z \tag{2.10}$$

$$z_{ll'} \leq d_{max} \quad \forall (h_l,\ h_{l'}) \in \text{timing-critical paths} \tag{2.11}$$

$$b_{l,ij} \leq e_i \qquad \forall l,\ j \tag{2.12}$$

$$e_i \leq \sum_{lj} b_{l,ij} \quad \forall i \tag{2.13}$$

$$\sum_{i} w_i \cdot e_i = W \tag{2.14}$$

$$\sum_{l} b_{l,ij} \leq 1 \quad \forall j \tag{2.15}$$

$$\sum_{i,j} b_{l,ij} = 1 \quad \forall i \tag{2.16}$$

Here, W is the total cost of selected flop trays, which is determined based on their power consumption and sizes (i.e., number of bits); D is the total displacement over all flops; Z is the total relative displacement over all timing-critical start-end flop pairs; and α and β are weighting parameters. Constraints (2.6) and (2.7) calculate the total displacement of all flops. Constraint (2.8) bounds the

maximum displacement of each flop. Constraints (2.9) and (2.10) calculate the total relative displacement of timing-critical start-end flop pairs (i.e., (h_l, $h_{l'}$)). Constraint (2.11) bounds the maximum relative displacement of each timing-critical start-end flop pair. Constraints (2.12) and (2.13) force the binary indicator variable e_i to be 1 if the corresponding flop tray is used, and 0 otherwise. Constraint (2.14) calculates the total cost of selected flop trays. Constraints (2.15) and (2.16) ensure that each flop is matched to exactly one slot, and that each slot is matched to at most one flop. We note that additional mutual exclusion constraints can avoid placement overlaps between pairs of flop trays (e.g., $e_i + e_j \leq 1$ if there is overlap between the i^{th} and j^{th} flop trays). However, such mutual exclusion constraints might limit the solution space and thus degrade the solution quality. We therefore perform placement legalization in the commercial P&R tool to remove overlaps among flop trays.[8] We also note that although an ILP-based optimization typically has large runtime, in our formulation, the number of binary variables is only $O(N \cdot Q)$, where N is the number of flops and Q is the number of candidate flop tray choices (i.e., sizes and dimensions). In practice, our method exhibits practical and reasonable runtimes (see Footnote 4 above).

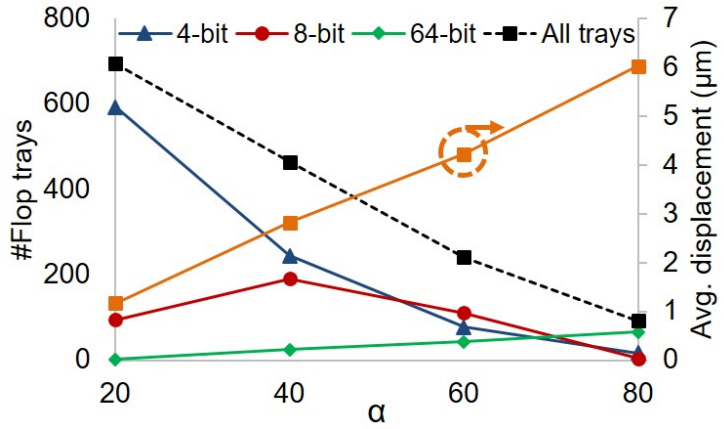


Figure 2.9: Number of flop trays and average displacement of flops change with different α values. Design: *JPEG*. Technology: 28FDSOI.

To give an understanding of how the weighting parameters α and β affect solution quality, Figure 2.9 shows the number of flop trays and the average flop displacement resulting from optimization with various α values. In the figure, each column is an implementation with corresponding α. The black-dotted curve indicates the total number of flops and flop trays. The orange curve indicates the

[8] Our experimental results show no more than three sites displacement on average per flop tray during the placement legalization.

average displacement over all flops. (Small) numbers of 16- and 32-bit flop trays are omitted for figure clarity. We observe that more large-size flop trays are selected with an increased value of α, so as to minimize the total tray costs. Such selection of large-size flop trays will reduce power of flop trays as well as the clock power. However, the average flop displacement increases with the value of α, and this can incur datapath power overhead. Therefore, the choice of α determines a tradeoff point between (i) clock power reduction and power reduction of flop trays, versus (ii) the power overhead on datapaths. In our experiments, we empirically set α = 20, 40, 60 and 80. We then select the solution with the minimum total block power from these four runs.

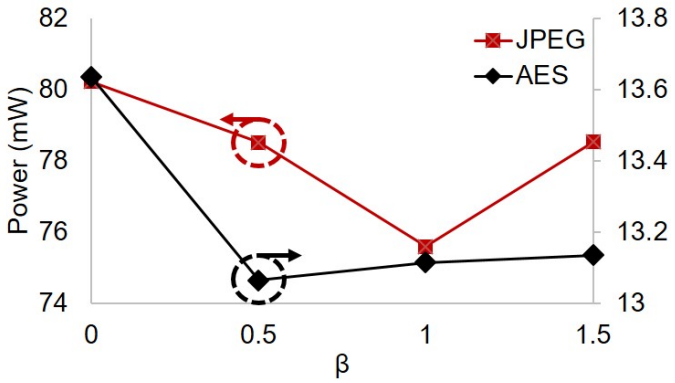


Figure 2.10: Power change with various β values. Designs: *AES*, *JPEG*. Technology: 28FDSOI.

To evaluate the impact of β, we uniformly place flop trays within the block area and fix their locations. The number of flop trays is determined by the number of flops; no flop tray can be empty, which eliminates the impact of W in our objective function. We then perform an ILP-based matching optimization to cluster flops into flop trays. Figure 2.10 shows the total block power of the *AES* and *JPEG* testcases implemented with various β values. We observe reduced block power with $\beta > 0$, where our optimization minimizes the relative displacement between timing-critical start-end flop pairs. This confirms the benefits of minimizing the relative displacement between timing-critical start-end flop pairs. We also observe increased block power with a large β value. This is because with a large β value, relative displacements between timing-critical start-end flop pairs dominate our objective function. The resultant large displacements of non-timing critical flops incur datapath power penalty. We empirically use β = 1 in our experiments.

2.1.3 Experiments

We perform experiments in a $28nm$ FDSOI foundry technology with dual-V_{th} libraries. We use four design blocks (*AES*, *JPEG*, *MPEG*, *VGA*) from the *OpenCores* website [124] as our testcases. Parameters of these four testcases are shown in Table 2.2. We scale flop tray power and area based on the ratios shown in Table 2.3. Layout ARs of flop trays are also shown in Table 2.3. We synthesize designs using *Synopsys Design Compiler vI-2013.12-SP3* [129] and then place and route using *Cadence Innovus Implementation System v15.2* [112]. We set the placement density at the floorplan stage as 70%. We also perform timing and power analyses using *Cadence Innovus Implementation System v15.2*. We perform vectorless power simulation with a default switching activity of 10% at primary inputs. Our optimization flow is implemented in C++. We use *CPLEX v12.6* [115] as our ILP solver and LEMON [121] as our min-cost flow solver. Functions used in P&R tools are implemented in Tcl. We conduct our experiments on a 2.5GHz Intel Xeon server.

Table 2.2: Testcase parameters.

Design	#Inst	#Flops	Clock period
AES	∼12K	530	600ps
JPEG	∼47K	4512	600ps
MPEG	∼13K	3181	500ps
VGA	∼56K	17053	700ps

Table 2.3: Normalized flop tray area and power, and layout AR.

Tray size	4-bit	8-bit	16-bit	32-bit	64-bit
Norm. area/power per bit	0.875	0.854	0.854	0.844	0.844
AR (#rows×#columns)	1×4	2×4	4×4	4×8	4×16
AR (#rows×#sites)	1×63	2×62	4×62	4×122	4×244

Comparison to Logical Clustering

To evaluate the performance of our proposed methodology, we compare our solutions to three reference flows: (i) the conventional implementation flow with only single-bit flops (*ref_1b*), (ii) a flop

tray-based implementation flow which generates flop trays during commercial synthesis based on logical clustering, followed by conventional commercial P&R optimization (*ref_mb1*), and (iii) a flop tray-based implementation flow which generates flop trays at the post-placement stage using the method proposed in [48], followed by clock tree synthesis and routing (*ref_mb2*). No value judgment or "benchmarking" regarding any commercial tool is intended by, or should be inferred from, our present discussion.

Table 2.4 shows the results evaluated at the post-routing stage. Figure 2.11 shows the layouts of placement solutions with single-bit flops and optimized flop trays. We observe that our proposed optimization (*opt_mb*) is able to significantly reduce the number of sinks with application of flop trays (e.g., we reduce the number of sinks by 98% on the *VGA* testcase as compared to the implementation using only single-bit flops). The reduction in number of sinks results in smaller clock power: our optimization reduces clock power by up to 90% and 40% compared to implementations with single-bit flops and flop trays generated by logical clustering, respectively. Our flop tray generation also results in reduced power on flops. Moreover, we observe that although our optimization has a large conversion ratio from single-bit flops to flop trays, the incurred datapath power and wirelength penalties are small as compared to the implementation with logical clustering. This strongly suggests that our approach of optimization with minimum perturbation from a "good" initial placement solution forestalls placement and routing congestion while also minimizing the datapath power penalty from application of flop trays. For the *MPEG* testcase, our optimization actually results in smaller datapath power as compared to the "ideal" implementation with single-bit flops; we believe this is likely due to a reduced placement density (i.e., usage of flop trays reduces the total area of flops).

Our optimization (*opt_mb*) also achieves up to 7% total block power reduction as compared to the previous work [48] (*ref_mb2*). Since *ref_mb2* only uses up to 8-bit flop trays, we limit the flop tray options to 4-bit and 8-bit flop trays in *opt_mb* for a fair comparison. Table 2.4 shows that with the same set of flop tray options, our optimization achieves 13% clock power reduction on average compared to *opt_mb'*, along with smaller datapath power for most of the testcases (the exception is the *JPEG* testcase with $< 1\%$ power overhead).

Table 2.4: Experimental results.

Design	Flow	Power (mW)				#Flops						#Clk	WNS	Area	WL	#Inst
		comb	seq	clk	sum (norm)	1	4	8	16	32	64	bufs	(ps)	(μm^2)	(μm)	
AES	*ref_1b*	**8.11**	**4.37**	**1.53**	**14.02 (1.00)**	530	0	0	0	0	0	11	-11	10362	140	12002
	ref_mb1	**8.64**	**4.00**	**0.72**	**13.35 (0.95)**	198	5	19	2	2	1	0	9	10606	153	11730
	ref_mb2	**8.14**	**4.05**	**0.43**	**12.62 (0.90)**	34	56	34	0	0	0	3	-4	10122	140	11595
	opt_mb	**8.15**	**3.94**	**0.46**	**12.56 (0.90)**	59	22	46	1	0	0	4	-5	10171	139	11619
	opt_mb'	**8.09**	**3.98**	**0.54**	**12.60 (0.90)**	80	41	36	0	0	0	4	-2	10160	137	11598
JPEG	*ref_1b*	**35.13**	**36.04**	**13.37**	**84.54 (1.00)**	4512	0	0	0	0	0	115	1	47595	420	47567
	ref_mb1	**36.88**	**33.21**	**6.10**	**76.20 (0.90)**	1388	109	84	70	0	14	59	0	46374	531	44246
	ref_mb2	**35.45**	**32.06**	**4.56**	**72.07 (0.85)**	308	457	297	0	0	0	40	-1	45888	437	44094
	opt_mb	**35.68**	**31.28**	**2.28**	**69.24 (0.82)**	274	77	110	2	9	43	25	1	45535	460	43545
	opt_mb'	**35.64**	**31.85**	**3.12**	**70.62 (0.84)**	83	37	537	0	0	0	28	1	45898	428	43607
MPEG	*ref_1b*	**5.88**	**28.93**	**10.72**	**45.53 (1.00)**	3181	0	0	0	0	0	92	-17	18169	149	12291
	ref_mb1	**6.52**	**26.99**	**5.19**	**38.70 (0.85)**	1225	27	17	15	18	14	53	-34	17757	195	10079
	ref_mb2	**6.03**	**25.62**	**3.30**	**34.95 (0.77)**	161	381	187	0	0	0	29	-11	17136	159	9849
	opt_mb	**5.66**	**25.12**	**0.98**	**31.76 (0.70)**	120	9	2	3	1	46	15	-3	16666	176	9183
	opt_mb'	**5.65**	**25.33**	**2.24**	**33.22 (0.73)**	77	16	382	0	0	0	21	-23	16780	149	9531
VGA	*ref_1b*	**14.32**	**108.34**	**42.19**	**164.84 (1.00)**	17053	0	0	0	0	0	361	-5	88015	960	56039
	ref_mb1	**16.63**	**101.63**	**20.73**	**138.99 (0.84)**	7325	42	77	75	50	96	215	-2	84537	1337	45793
	ref_mb2	**14.60**	**94.51**	**10.24**	**119.35 (0.73)**	129	1299	1466	0	0	0	110	-2	80710	1032	41656
	opt_mb	**15.29**	**93.99**	**2.04**	**111.32 (0.68)**	33	1	6	0	2	266	28	3	80083	1132	39129
	opt_mb'	**14.33**	**94.29**	**8.41**	**117.03 (0.71)**	56	51	2114	0	0	0	89	-13	80538	1001	40909

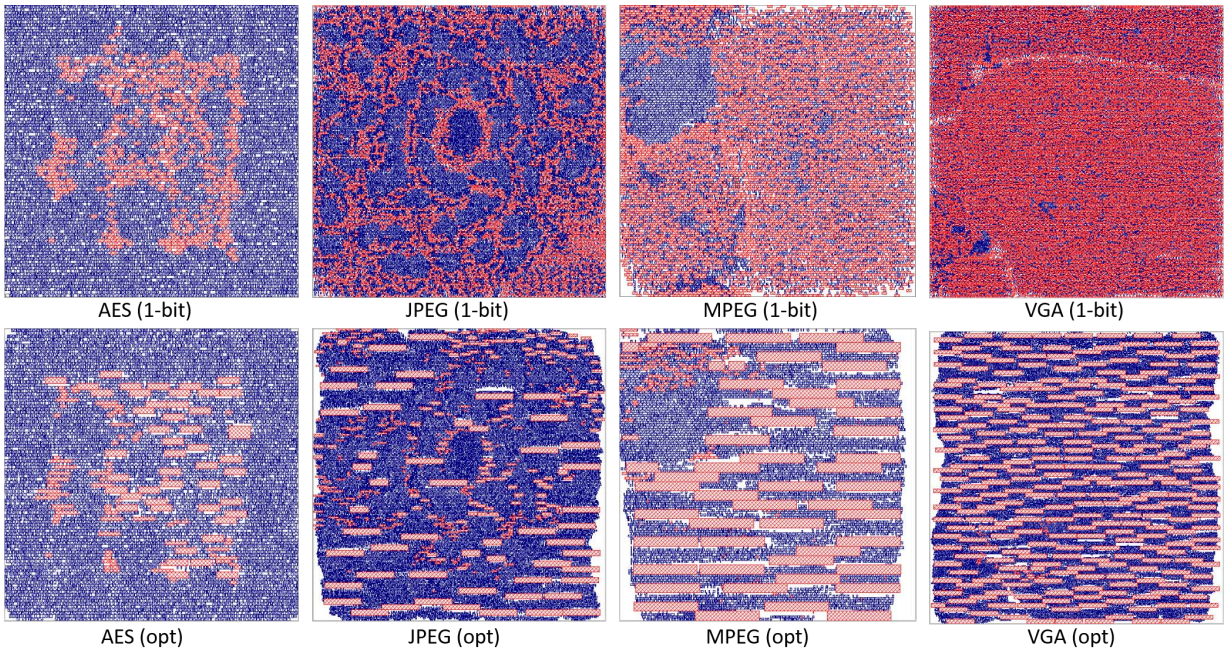


Figure 2.11: Layout comparison between implementations with only single-bit flops and with optimized flop trays. In the flop tray-based solutions, the candidate flop tray sizes are 4-bit, 8-bit, 16-bit, 32-bit and 64-bit.

Optimization with Various Flop Tray Sizes

We further perform flop tray optimization with various combinations of flop tray sizes. More specifically, we implement designs with (i) single-bit flops only, (ii) {4-bit} flop trays, (iii) {4-bit, 8-bit} flop trays, (iv) {4-bit, 8-bit, 16-bit} flop trays, and (v) {4-bit, 8-bit, 16-bit, 32-bit, 64-bit} flop trays with various α values (i.e., 20, 40, 60, 80). We note that setups (ii) – (v) can also use single-bit flops. For each setup, we select the minimum total block power solution with $< 5\%$ power penalty on datapaths as compared to the case with only single-bit flops. Figure 2.12 shows flop power and clock power, normalized to implementations using only single-bit flops. We observe that with only 4-bit flop trays, our optimization achieves $> 7\%$ power reduction on flops and flop trays. However, including larger flop trays does not afford much further reduction of flop power. (This may be due to our conservative assumptions regarding power-per-bit in larger flop trays, as shown in Table 2.3). On the other hand, application of large-size flop trays can effectively reduce clock power. For example, optimizations with {16-bit, 32-bit, 64-bit} flop trays achieve 11% more clock power reduction on average as compared to the cases with only {4-bit, 8-bit} flop trays.

Study of Useful Skew Optimization with Flop Trays

Last, we evaluate the benefits of useful skew optimization in terms of leakage power reduction on (i) designs with only single-bit flops (*reflocal_1b*), and (ii) flop tray-based designs (*opt_mb* as shown in Figure 2.13.[9] Based on the approach proposed in [2], we formulate the useful skew optimization as a maximum mean weight cycle problem and apply iterative shortest path search to maximize the average endpoint slack. We then perform leakage power optimization using a commercial tool [112], i.e., we exploit the increased timing slacks for leakage power reduction. We observe from Figure 2.13 that due to the clustering of endpoints, flop tray-based designs have 9% less leakage power reduction on average across four designs as compared to cases with only single-bit flops. To reduce the impact of flop tray generation on benefits from useful skew optimization, we study skew-aware flop tray generation that only allows clustering of flops with desired skew less than θ (we use $\theta = 20ps$ in our experiments). Figure 2.13

[9] In the technology we use, we do not observe significant dynamic power benefits from useful skew optimization. We therefore study leakage power reduction from useful skew optimization in this experiment.

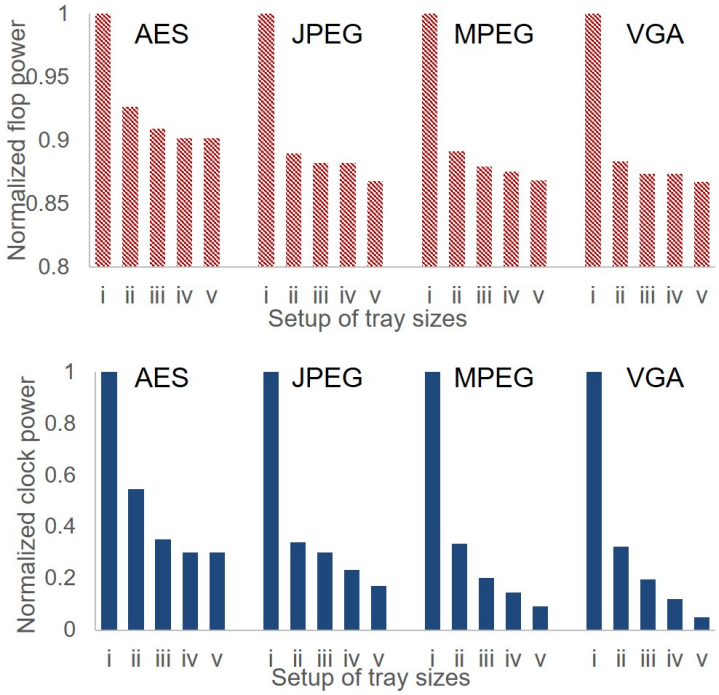
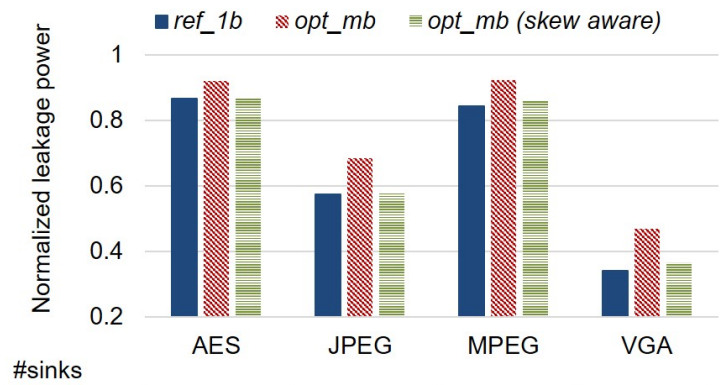


Figure 2.12: Flop (tray) power and clock power of designs with various flop tray sizes. Candidate tray sizes are 4-bit, 8-bit, 16-bit, 32-bit and 64-bit.

shows that the skew-aware clustering (*opt_mb (skew aware)*) can achieve similar leakage power reduction as compared to the cases with only single-bit flops (green vs. blue bars), but at the cost of more sinks (i.e., an average of 21% less reduction in number of sinks than *opt_mb*).

2.1.4 Conclusion

In this work, we present a novel flop tray-based optimization for improved design power reduction. We propose a capacitated K-means algorithm which iteratively applies a min-cost flow-based clustering and a LP-based flop tray placement. We also propose an ILP-based matching optimization to generate flop trays while minimizing the perturbation to the initial placement solution. Our work achieves several improvements as compared to previous works: (i) awareness of flop tray aspect ratio and (large) size; (ii) explicit minimization of relative displacement of timing-critical start-end flop pairs; and (iii) global optimization instead of local search. The proposed techniques allow us to achieve up to 32% total block power reduction as compared to designs with only single-bit flops, and up to 16% total block power reduction over designs with flop trays generated by logical clustering during synthesis. We also achieve



#sinks	AES	JPEG	MPEG	VGA
ref_1b	530	4512	3181	17053
opt_mb	128	515	181	308
opt_mb (skew aware)	392	1830	205	1245

Figure 2.13: Datapath leakage power results, normalized to implementations with only single-bit flops.

13% clock power reduction on average as compared to the previous work in [48]. We further study the impact of flop tray sizes on optimization solution quality, as well as the useful skew optimization in the context of our flop tray-based designs.

2.2 MILP-Based Optimization of 2D Block Masks for Timing-Aware Dummy Segment Removal in Self-Aligned Multiple Patterning Layouts

Self-aligned multiple patterning (SAMP), due to its low overlay error, has emerged as the leading option for 1D gridded back-end-of-line (BEOL) layers in sub-14nm nodes. To form actual routing patterns from a uniform "sea of wires", *keep*[10] or *block*[11] approaches can be used. The work of [28] demonstrates that mask shapes used to keep signal wire segments (M2 pitch $=$ 32nm [85][87]) are not patternable with single-exposure lithography, even if we assume aggressive optical proximity correction (OPC). To address this problem, the block approach is used, wherein both 1D *cut masks* and 2D *block masks* are required. 1D *cut masks* are needed for line-end cutting or realization of space between routing segments, resulting in end-of-line (EOL) extensions and non-functional (i.e. dummy fill) patterns.[12]

[10] *Keep* refers to a mask to keep signal wire segments.

[11] *Block* refers to a mask to erase dummy wire segments.

[12] In terms of layout patterns, cut mask and block mask would act the same since both masks remove unnecessary metal patterns. Indeed, the terms cut and block are used interchangeably in many previous works. In this section, we use the term cut mask to

Despite previous works [14][22][40][109] proposing cut mask optimizations to minimize the EOL extension, such effects as increased capacitance, degraded timing and power are inevitable due to dummy fill patterns. Therefore, extra 2D *block masks* can be used to remove dummy fill patterns. However, using only 2D block masks cannot realize line ends due to required complex shapes, particularly with metal pitch $\leq 32nm$ in N7 / N5 nodes, which is our focus in this section. [28] shows that 2D block mask shapes fail to realize $\leq 80nm$ tip-to-tip spacing between line ends while a 1D cut mask strategy can realize $56nm$ tip-to-tip spacing. Thus, 1D cut masks are needed to define clean line ends with small tip-to-tip spacing. In this section, we assume that the cut mask is used to define EOL, and the block mask is used to remove dummy fill patterns or define EOL with a margin.

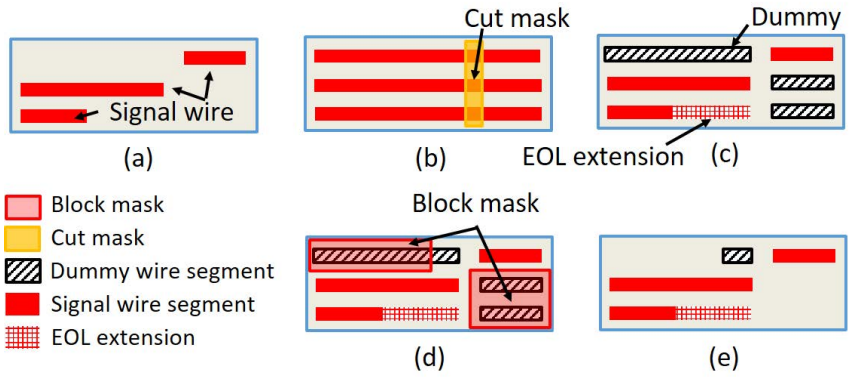


Figure 2.14: SAMP process: (a) post-route layout; (b) cut mask application; (c) layout after cut mask application; (d) block mask application; and (e) final layout after block mask application.

Figure 2.14 illustrates 1D SAMP patterning with cut and block masks. For a given post-route layout, a "sea of wires" is generated and line ends are defined by a cut mask, as shown in Figures 2.14(a) and (b). After the cut process, Figure 2.14(c) shows one EOL extension and three non-functional dummy segments. 2D block mask application is shown in Figure 2.14(d), and Figure 2.14(e) shows the final layout with one EOL extension and one dummy segment. Compared to the layout in Figure 2.14(c), (e) is superior with smaller capacitance, lower power, and better timing, due to fewer dummy segments.

For printability, 2D block masks must satisfy given design rules from a particular patterning technology. Possible patterning technology options include single-exposure (SE) 193i, SE 193d, and EUV. Except in the case of EUV, the critical dimension (CD) for block mask shapes is $\sim 2\times$ larger than the

refer to a 1D shaped mask, and the term block mask to refer to a 2D shaped mask.

minimum pitch of the 1D SAMP BEOL process. Thus, it is not possible to cover all dummy segments using one block mask. For example, in Figure 2.14(e), the two dummy segments in the final layout cannot be removed because of (i) the minimum spacing constraint between individual block mask shapes; and (ii) the L-shape constraint. The first main contribution of this section is that we formulate and optimally solve for 2D block mask shapes based on realistic design rules of SE 193i and SE 193d patterning technology from industry [118], and with support for a "selective"[13] variant of block-mask patterning technology.

In advanced nodes, minimum metal density is crucial to chemical-mechanical polishing (CMP) [32]. In the 1D SAMP manufacturing process, metal fills are generated intrinsically by the "sea-of-wires" with cut process, and partially removed by the block mask, as opposed to a dedicated post-routing metal fill process in the traditional physical design flow. Thus, block mask optimization must be metal density-aware.[14] Another contribution of this section is that we consider the local minimum metal density constraint.

From a performance perspective, maximizing the block mask usage (dummy removal) is not equivalent to minimizing timing impact of dummy fill patterns. In our preliminary study, a timing-oblivious block mask optimization that simply maximizes dummy removal (design: *M0*) can only recover 14% of the WNS degradation caused by non-functional dummy fill patterns. At the same time, timing-aware block mask optimization can run much faster than timing-oblivious optimization since we do not need to optimize non-functional dummy fills. Further, given minimum metal density constraints, a smart dummy removal method is required to maximize timing recovery. Thus, it is important to capture timing impact of dummy segments in block mask optimization. The second main contribution of this section is that we incorporate into our optimization a timing model to evaluate dummy fill performance impact. Together with our first main contribution, this enables quantified assessment of performance benefits from selective block mask technology.

Lastly, we extend our MILP to a *co-optimization* of cut and block masks, opening up a broader solution space. Compared to a sequential cut [40] and block mask optimization, where line-end realization is performed with cut mask only, a cut and block mask co-optimization seeks to use both cut and block

[13] I.e., a block mask approach that selectively removes metal lines according to the colors of metal. See Section 2.2.1 for the detailed description.

[14] Regarding the feasibility of the final pattern after dummy removal in terms of lithography, we note that [28] validates the cut and block mask approach with lithography simulation. We also note that CMP effects from pattern density occur at relatively large length scales compared to feature and pitch dimensions in N7/N5 Mx layer.

masks for realization of line ends: the block mask can complement the cut mask when a cut-only solution may result in excessive EOL extensions.

To summarize, in this section we propose an MILP-based optimization for 2D block mask with timing-aware dummy segment removal, while satisfying a given set of block mask rules (including for selective block mask technology) and metal density constraints. We further provide what we believe to be the first co-optimization of cut and block mask patterns. Our key contributions are as follows.

- To our knowledge, our work is the first to optimize 2D block mask layout considering realistic block mask rules, timing impact of dummy fills and metal density constraints.
- We develop a timing model to evaluate performance impact on a per-segment basis.
- We develop a co-optimization of cut and block mask layout.
- We study the impacts of timing-awareness and patterning technology on optimization outcomes, and we furthermore quantify the power and timing benefits of the "selective" approach.
- Our MILP formulation gives new insights into the fundamental limits of the benefits from emerging (cut and) block mask technology options.

In the remainder of this section, Section 2.2.1 provides background of cut and block mask technology, as well as related work. In Section 2.2.2, we describe our MILP-based optimization of 2D block masks and our cut and block mask co-optimization. We also explain our model to capture the timing impact of dummy segments. We describe our conflict list generation techniques, distributed optimization strategy and overall flow in Section 2.2.3. Section 2.2.4 provides experimental results and analysis. We give conclusions in Section 2.2.5.

2.2.1 Related Work and Preliminaries

In this section, we first describe block mask rules and the "selective" block approach. We then review cut mask rules, the selective cut approach, and LELE cuts. Last, we review relevant related works.

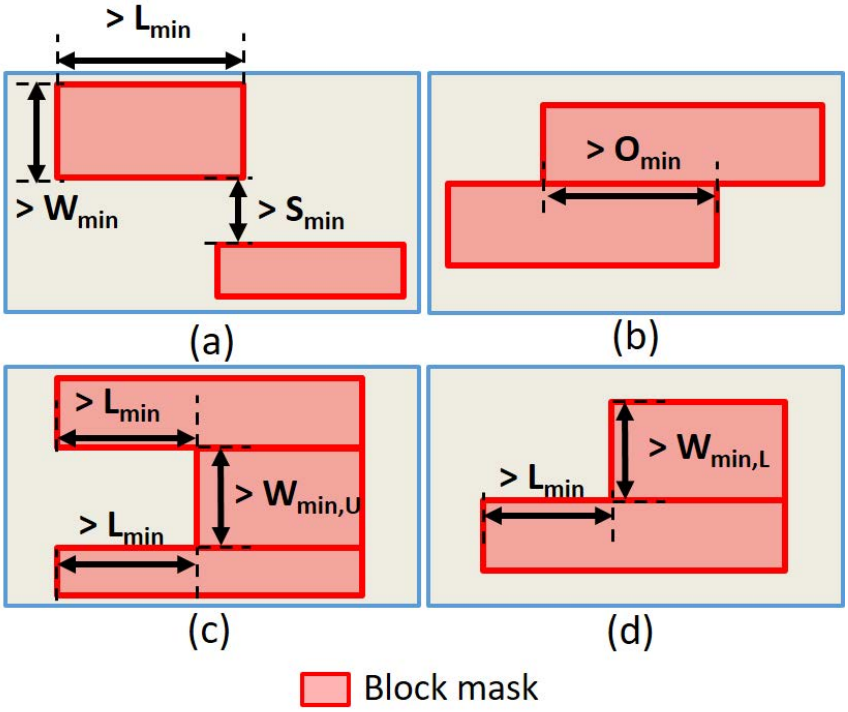


Figure 2.15: Block mask rules: (a) minimum width and length rules; (b) minimum overlap rule; (c) minimum U-shape rule; and (d) minimum L-shape rule.

Block Mask Rules / Selectivity

Block mask rules constrain each individual shape on the block mask, as well as sets of adjacent block mask shapes. A set of essential rules for block mask shapes is shown in Figure 2.15. For each rectilinear block mask shape, Figure 2.15(a) illustrates minimum width, minimum length, and minimum spacing constraints. For a given rectilinear shape, we use "length" to refer to the extent (length) of edges along the direction of metal lines, and "width" refer to the length of edges perpendicular to the direction of metal lines. When two rectilinear shapes abut each other but are not perfectly aligned, as shown in Figure 2.15(b), a minimum overlap rule applies. Figures 2.15(c) and (d) illustrate U-shape and L-shape constraints. Table 2.5 shows preliminary block mask rule sets (R1 - R8) for 193i and 193d patterning technologies.[15]

[15]We use the term "preliminary" since plan-of-record patterning strategies for mass production at N7 / N5 did not yet exist at the time this research was performed. Values in Table 2.5 are from our collaborators at a leading technology development center / consortium.

Table 2.5: Preliminary cut and block mask rules.

Rule	Notation	Meaning	Values (nm)	
			193i	**193d**
R1	W_{min}	minimum width	60	120
R2	S_{min}	minimum spacing	240	480
R3	L_{min}	minimum length	120	240
R4	O_{min}	minimum overlap	240	480
R5	$W_{min,U}$	minimum width (U-shape)	120	240
R6	$W_{min,L}$	minimum width (L-shape)	60	120
R7	C_{min}	minimum cut spacing	80	N/A
R8	C_w	cut width	20	N/A

A *selective block approach* [58] allows removal of some, but not all, segments covered by the block mask. More precisely, similar to multiple patterning technology, the selective block approach selectively removes dummy segments according to the *color* of the wire segment. There are two methodologies that realize selectivity for block mask: (i) order selectivity and (ii) material selectivity. In [58], the selective blocks for metal color A and metal color B are processed sequentially. In other words, the block A for metal color A is processed right after the patterning of metal color A; then, metal B is patterned followed by block color B. Given the process order, block A only blocks metal A, and block B only blocks metal B, due to the order in which the process is assembled. By contrast, the material selectivity-based approach [41] is particularly applied to SADP/SAQP, where there are two types of wires that are created by mandrel and gap. Figure 2.16 illustrates the process of the material selectivity-based approach for SAQP. In this SAQP process, spacer-is-dielectric is assumed. After 1st and 2nd spacers are generated, the region between spacers is filled with material A. Then, two types of block masks are introduced: one for material A, and the other for 1st spacers. The two block masks are used to perform the etch process which is selective to material A or to 1st spacer.[16] The final metal patterns are shown in blue color.

Figure 2.17 illustrates the difference between the selective block and non-selective block approaches. The red (resp. green) block mask in Figure 2.17(a) (resp. (b)) removes red (resp. green) dummy segments, but acts as transparent to green (resp. red) segments. Note that without selectivity, the gray

[16] Indeed, there are 3 colors where each color defines the first spacer, the second spacer and the gap. However, after the second spacer formation, the first spacer is already excavated on the hardmask, and there are only the second spacer and gap as the two materials. Thus, the same color contrast that is used in SADP (e.g., two colors) can be used in SAQP as well.

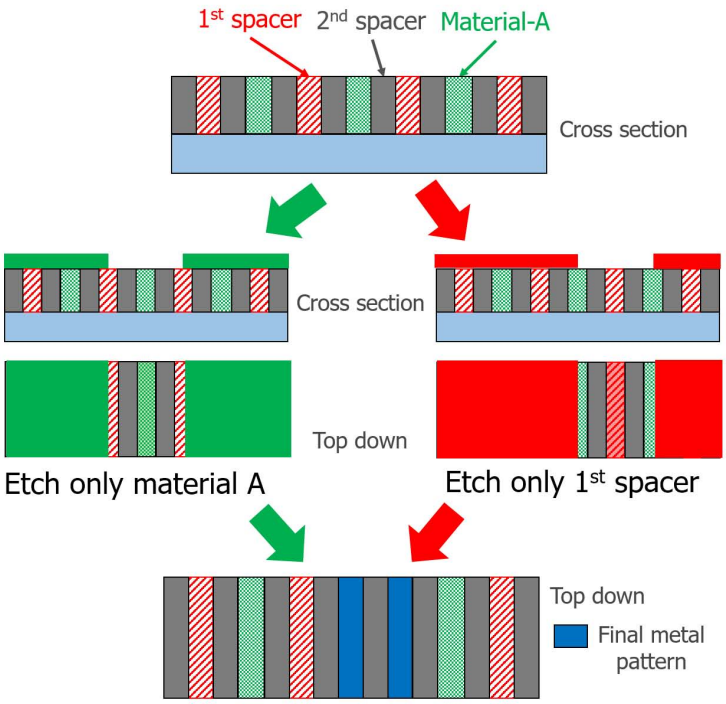


Figure 2.16: Illustration of the material selectivity-based block approach.

block mask shape becomes complex (Figure 2.17(c)) and may not be patternable with single-exposure (SE) in 193i/193d. Since the color of wire segments is assigned alternatively track by track, selective block mask applies separately to odd and even tracks. With selectivity, as shown in Figures 2.17(a) and (b), block mask shapes can extend to non-target tracks, which is equivalent to doubling the metal pitch.

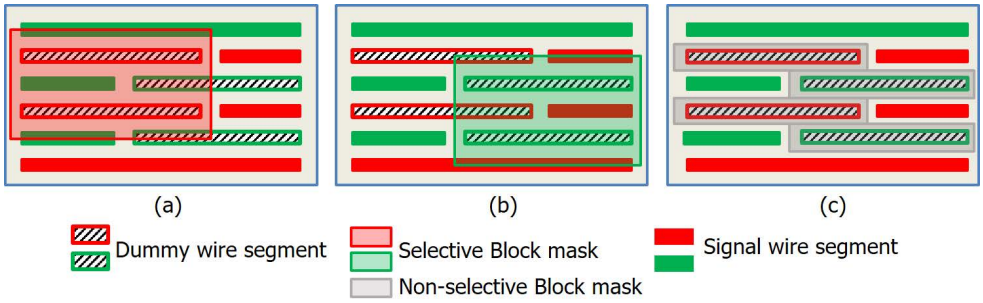


Figure 2.17: Comparison between selective block and non-selective block: (a) selective block mask in red removes only red segments; (b) selective block mask in green removes only green segments; and (c) a complex non-selective block mask is required to remove the same dummy segments.

Cut Mask Rules / Selectivity / LELE cut

Cut mask rules constrain shapes on the cut mask. As in [14][22][28][40][109], we assume that cut mask shapes are unit-size rectangular cuts, with width equal to the *cut width*. A cut mask must satisfy a minimum cut spacing constraint, which is the center-to-center distance between two disjoint cuts. Two cuts are exempt from the minimum cut spacing rule if they abut and are fully aligned. For two aligned merged cuts, the minimum spacing rule is applied between each pair of unit-size cuts so that the edge-to-edge distance is always guaranteed to be above a lower bound, as shown in Figure 2.18. Table 2.5 shows preliminary cut mask rule sets (R7, R8) for 193i patterning technologies.

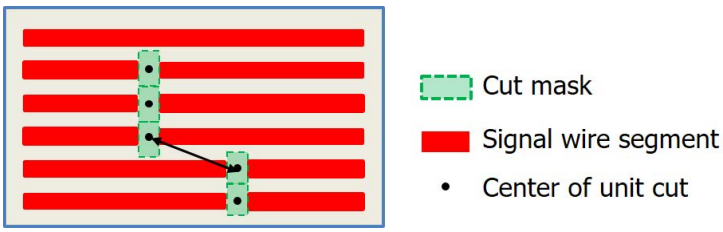


Figure 2.18: Cut mask rules: minimum spacing.

Similar to selective block, the *selective cut approach* realizes EOL only for the corresponding color of wire segments. As another option, the non-selective *LELE cut approach* uses two cut masks to realize EOL, regardless of the color of wire segments. Minimum cut spacing is checked within each cut mask, because two cut masks do not interfere with each other. Figures 2.19(a) and (b) illustrate the selective cut and LELE cut approaches, respectively. In Figure 2.19(a), similar to selective block, cuts can extend to non-target tracks while not affecting segments of a different color. Thus, two green (resp. two red) cuts are aligned and there is no need to check minimum cut spacing since the colors of the cuts are different. Figure 2.19(b) shows LELE cuts. A minimum cut spacing rule is enforced separately for two green (resp. two red) cuts.

Related Works

While selective block mask is a very recent concept [58], we may classify related works into four categories: (i) 1D cut mask optimization, (ii) 2D block mask optimization, (iii) 1D cut mask-aware routing optimization and (iv) 2D block mask-aware routing optimization.

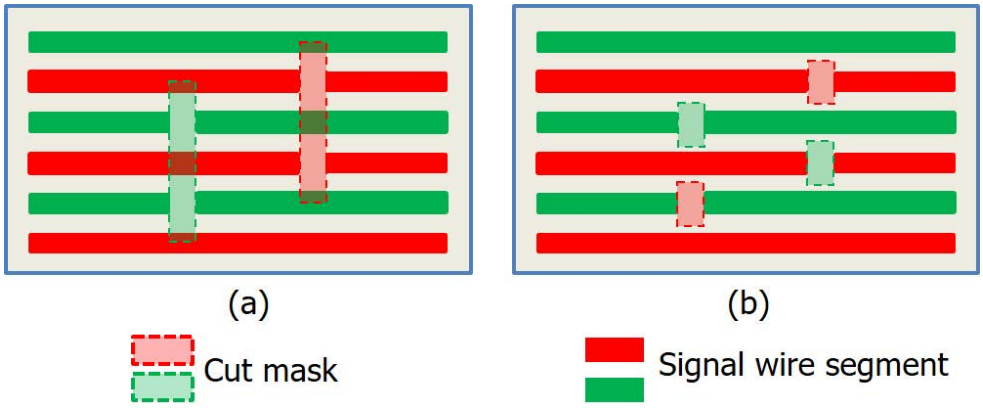


Figure 2.19: Comparison between selective cuts and non-selective LELE cuts. (a) Selective cut mask in red (resp. green) realizes EOL only for red (resp. green) segments, and is transparent to green (resp. red) segments. (b) Non-selective LELE cuts realize EOL for both colors.

1D cut mask optimization. Zhang et al. [110] propose a shortest-path algorithm to resolve lithography hotspots in cut masks. Du et al. [22] propose an integer linear program to minimize total end-of-line (EOL) extension. Ding et al. [14] subsequently extend the methodology in [22] to reduce the runtime. Han et al. [40] extend the MILP formulation in [14] and propose co-optimization of cut mask layout, dummy fill and timing. Their objective incorporates awareness of design timing in minimizing a weighted sum of EOL extensions, with weights determined by a grouping of timing slacks. [40] also proposes a post-MILP optimization that iteratively removes dummy segments near timing-critical nets while satisfying density and uniformity constraints. However, 2D block mask optimization is not supported, and the grouping-based weights that are employed to achieve a timing-aware optimization may not be accurate.

2D block mask optimization. Zhang et al. [109] propose a constrained shortest-path algorithm to improve the printability of 2D block masks. Printability is assumed to be a function of the number of polygon edges in the block mask. The authors of [109] show a tradeoff between printability and wirelength increase, albeit without any hard design rule constraints. Ding et al. [14] propose an integer linear program formulation, with support for limited design rules. By contrast, our formulation supports flexible design rules, and we use recent, realistic design rules from collaborators from a leading technology consortium. We also incorporate a more accurate model to minimize the timing impact of dummy fill patterns.

1D cut mask-aware routing optimization.[17] Su and Chang [90] propose a nanowire-aware router that considers cut mask complexity. They first estimate the line-end probability cost for each global routing tile based on a pre-evaluation of line-end counts using minimum spanning trees. They then perform global routing while minimizing the routing bends and considering hotspots with respect to the line-end costs. After that, force-driven layer and track assignments are performed. At this stage, an attractive force is established for wires that can share a cut. The authors of [90] also suggest detailed routing with a cost function that considers cut sharing and EOL extension.

2D block mask-aware routing optimization. Fang [23] proposes an ILP-based wire planning approach that considers block masks. The proposed ILP minimizes the generation of single track/wire segments during track routing. She then performs detailed routing, which is based on A* search routing with block mask-aware routing costs.

2.2.2 MILP-based 2D Block Mask Optimization

We now present our problem statement, our MILP formulation, as well as the timing model used for each of our two optimizations: (1) 2D block mask optimization, and (2) cut and block mask co-optimization.

Problem Statement

2D block mask optimization. Given a post-route layout with EOL extensions and legal EOL cuts, timing information, minimum metal density constraint, and technology options (i.e., block mask rules, selectivity), **perform** 2D block mask optimization considering block mask rules and metal density constraints, such that timing impact of dummy segments is minimized.

Cut and block mask co-optimization. Given a post-route layout, timing information, minimum metal density constraint, and technology options (i.e., cut mask rules, block mask rules, selectivity), **perform** co-optimization of cut and block masks considering cut mask rules, block mask rules, and metal

[17] The co-optimization with routing is beyond the scope of our present work. We understand that a co-optimization of routing, cut and block mask should result in the best performance. However, integration of a custom router and a commercial tool flow with full N7 / N5 design rule support is extremely hard (and, not accessible to us); "hacks" possible for us in the academic setting usually result in degraded performance.

density constraints, such that EOL of signal segments is realized by cut or block mask, and the timing impact of EOL extension and dummy segments is minimized.

MILP Formulation for Block Mask Optimization

We now formulate the MILP problem for the block mask optimization problem. Table 2.6 shows the notations that we use in our formulation.

Table 2.6: Notations. The notations from the twelfth row to the eighteenth row (i.e., beginning with $c^f_{i,j}$) are used for cut and block co-optimization.

Notation	Meaning
$v_{i,j}$	(0-1) indicator of whether the block candidate j of shape i is used
$t^k_{i,j}$	delay increase due to dummy segments for net k if $v_{i,j} = 0$
l_i	original dummy segment length of shape i
$r_{i,j}$	removed dummy segment length if $v_{i,j} = 1$
L	total length of signal wires
K_p	set of nets in path p
$B_{q,a}(B_{q,b})$	q^{th} set of typeA (typeB) conflicting block candidates
d_{min}	minimum metal density constraint
s_p	initial timing slack of path p
m_p	timing degradation of path p
$c^f_{i,j}$	(0-1) indicator of whether cut candidate j of shape i is on cut mask f
$c_{i,j}$	(0-1) indicator of whether the cut candidate j of shape i is used
$C_{q,a}(C_{q,b})$	q^{th} set of typeA (typeB) conflicting cut candidates
j_l (j_r)	location of left (right) edge of cut or block candidate j
e_{i,x_l} (e_{i,x_r})	(0-1) indicator of whether location x is the left (right) edge of any selected cut or block candidate of shape i
e'_{i,x_l} (e'_{i,x_r})	(0-1) indicator of whether location x is the leftmost (resp. rightmost) of shape i
t'^k_{i,x_l} (t'^k_{i,x_r})	delay increase due to EOL extension for net k if $e'_{i,x_l} = 1$ (resp. if $e'_{i,x_r} = 1$)

Block candidates. We begin by describing how a block mask layout is represented within our MILP formulation. In the block mask layout, we create a dedicated rectangular *shape* for every dummy

wire segment between signal segments. Figure 2.20 shows an example with three dummy wire segments, covered by three rectangular block mask shapes in the block mask layout. The final block mask layout may vary from the ones shown in Figure 2.20 since each shape may change according to the selected *block candidate*. We define *block candidates* as subsegments of a rectangular block mask shape for a dummy segment. We provide several *block candidates* for each rectangular shape. We do this by slicing each rectangular shape according to a user-specified input length ($120nm$, in all results reported below) into several subsegments that define block candidates.[18] Because block mask cannot realize EOL with small tip-to-tip spacing, for leftmost (or rightmost) block candidates, we add "EOL margin" between the boundary of candidates and the signal EOL. The EOL margin is illustrated in Figure 2.22(a).

Figure 2.20 illustrates four block candidates $v_{2,1}$, $v_{2,2}$, $v_{2,3}$, $v_{2,4}$ for Shape 2. The block candidates are indexed in ascending (resp. descending) order of x coordinate. The final block mask layout for Shape 2 is determined by selected block candidates. The height of the shape is determined by the metal pitch, as shown in Figure 2.20. For the "selective" block approach, shapes can extend to the non-target tracks, equivalent to doubling the metal pitch. The following MILP optimally selects block candidates of each rectangular shape, while satisfying block mask rules.

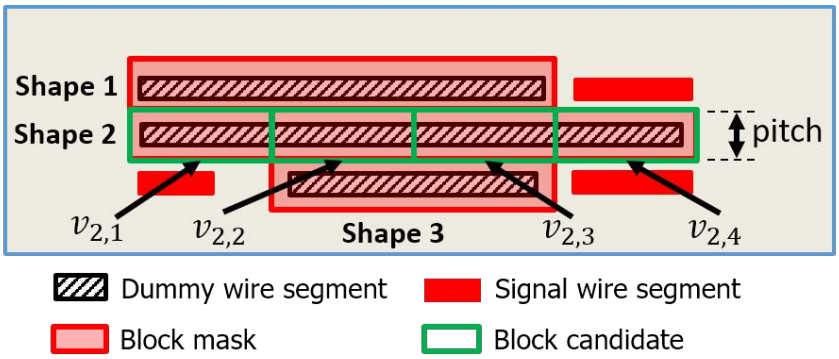


Figure 2.20: Shapes and block candidates for Shape 2.

[18] We note that there is a tradeoff between solution quality and runtime depending on the user-specified input length, which determines fine-grained or coarse-grained block candidate generation. Experimental results for various block candidate lengths are reported in Section 2.2.4.

$$\text{Minimize:} \quad \sum_{p} m_p \tag{2.17}$$

Subject to:

$$\sum_{\substack{(i,j)\in B_{q,a} \\ (i',j')\in B_{q,b}}} v_{i,j} + (1 - v_{i',j'}) < |B_{q,a}| + |B_{q,b}|, \ \ \forall q \tag{2.18}$$

$$L + \sum_{i} (l_i - \sum_{j} r_{i,j} \cdot v_{i,j}) \geq d_{min} \tag{2.19}$$

$$\sum_{k\in K_p} \sum_{i,j} t^k_{i,j} \cdot (1 - v_{i,j}) \leq s_p + m_p, \ \ \forall p \tag{2.20}$$

$$m_p \geq 0, \quad \forall p \tag{2.21}$$

The objective is to minimize the total timing degradation arising from the final dummy fill patterns for timing-critical paths. We extract (setup) timing-critical paths using *Cadence Tempus Timing Signoff Solution v15.2* [114] (dummy segments and EOL extensions do not worsen hold, as we do not touch the clock distribution). A path is considered to be timing-critical if its slack is less than a prescribed threshold for timing-criticality.[19] For path p, the timing degradation m_p is defined as the delay increase d_p (induced by dummy fills that affect path p) that exceeds the initial timing slack s_p, i.e., $m_p = \max(d_p - s_p, 0)$.[20] In this way, we only count timing degradation that causes a negative timing slack. The value m_p is calculated from the sum of delay increases along path p, subtracted by the initial timing slack s_p.

Constraints for block mask rule violation. Constraint (2.18) prevents block mask rule violations. Given a set of close-by block candidates from neighboring shapes, we enumerate conflict sets where selection (removal) of each block candidate in any given conflict set forms a violating block shape. In Constraint (2.18), $B_{q,a}$ (resp.$B_{q,b}$) represents conflict set q, which stores a (minimal) set of block candidates that cannot be "selected" (resp. "removed") simultaneously. More specifically, we define typeA candidates

[19] We use +200ps as the threshold for timing-criticality in our experiments. The numbers of timing-critical paths for initial implementations are 8K, 0.9K and 18K for *M0*, *AES* and *JPEG*, respectively.

[20] For example, if $s_p = 10ps$, and $d_p = 5ps$, then $m_p = 0$. If $s_p = -10ps$, and $d_p = 5ps$, then $m_p = 15ps$. Constraints (2.20) and (2.21) enforce $m_p = \max(d_p - s_p, 0)$. We note that we do not optimize for the timing degradation within positive slacks. However, our formulation can be easily adapted by designers to preserve a given amount of positive slack (i.e., timing guardband) by decreasing s_p.

such that the inclusion of the candidates forms the violating shape, and store the candidates in $B_{q,a}$. Similarly, we define typeB candidates such that the exclusion of the candidates forms the violating shape, and store the candidates in $B_{q,b}$.

We create a constraint to forbid each block mask pattern that forms a block mask rule violation. Figure 2.21 illustrates an example minimum width U-shape block mask rule violation on the right boundary of $v_{3,1}$. The figure shows typeA and typeB candidates that define a violating U-shape, with don't-care candidates that do not directly contribute to the formation of the U-shape violation. In this example, we prevent the U-shape rule violation with the following constraint:

$$v_{2,1} + v_{2,2} + (1 - v_{3,1}) + v_{3,2} + v_{4,2} + v_{4,3} < 6 \tag{2.22}$$

In Constraint (2.22), if any candidate in the typeA candidate set (e.g., $v_{2,1}$, $v_{2,2}$, $v_{3,2}$, $v_{4,2}$, $v_{4,3}$) is zero or any candidate in the typeB candidate set (e.g., $v_{3,1}$) is one, the violating U-shape does not exist anymore. In this case, the constraint is automatically satisfied. We note that we only enumerate "minimal" sets of typeA and typeB candidates. For example, the inclusion of candidates $v_{1,1}$, $v_{1,2}$ and $v_{4,1}$ in addition to the typeA set above (and with the exclusion of the typeB set) forms an additional violating U-shape. However, this case is forbidden by Constraint (2.22). Thus, $v_{1,1}$, $v_{1,2}$ and $v_{4,1}$ are don't-care candidates. In light of this, we find that for the block mask rules that we have studied, relevant combinations will exist within very small neighborhoods of any given block candidates. Thus, the complexity of enumeration of block candidate combinations to determine set B is in practice linear to the number of block candidates.[21]

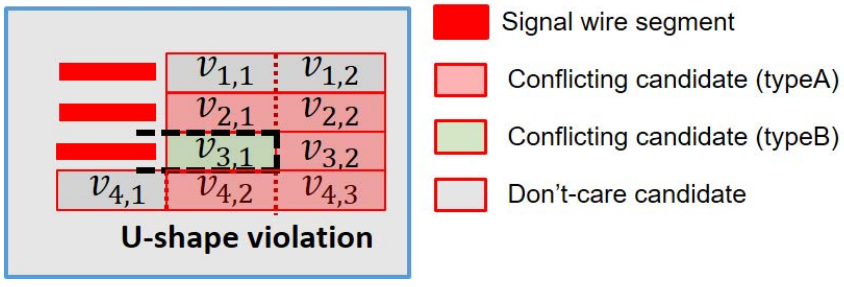


Figure 2.21: Illustration of a U-shape block mask rule violation.

[21] In our experiments, the total runtimes of conflict lists generation for *M0* and *JPEG* are 36 and 184 seconds, respectively. The number of segments (shapes) in *JPEG* is 257K, and the number of shapes in *M0* is 63K.

Constraints for local minimum metal density. Constraint (2.19) enforces the local minimum metal density. We obtain the total signal wire length L from the routed layout. Variable $l_{i,j}$ is the removed dummy segment length if $v_{i,j} = 1$ for shape i. $\sum_i l_i - \sum_j r_{i,j} \cdot v_{i,j}$ calculates the total dummy wire segment length. $r_{i,j}$ is the length of block candidate $v_{i,j}$. The minimum metal density is enforced locally within each clip; this is described in Section 2.2.3 below.

Constraints for timing-critical paths. Constraint (2.20) upper-bounds the timing degradation for timing-critical paths. Variable $t^k_{i,j}$ is the delay increase for net k caused by the remaining dummy segment if $v_{i,j} = 0$. We sum up the delay increase of every stage (gate and wire) on timing-critical path p and force this sum to be smaller than $s_p + m_p$. The initial path slack s_p is calculated from an initial design with no dummy segments. For each timing-critical path p, $m_p = 0$ indicates that the delay increase is not larger than the initial path slack s_p and thus design WNS will not worsen;[22] otherwise, $m_p > 0$. Note that we minimize m_p for all timing critical paths p by the objective. Constraint (2.21) limits m_p to be a non-negative number. We also note that Constraint (2.21) is necessary to optimize WNS as well as TNS. If we do not have such a constraint, the algorithm might keep removing dummy segments that are associated with "less" timing critical paths instead of focusing on the most timing critical path. For example, let us suppose that there are two paths with slacks $s_1 = 10$ and $s_2 = 0$. With Constraint (2.21), we optimize m_2 rather than m_1 since constraints for m_2 is tighter (i.e., the second path is more critical), and it is not necessary to optimize m_1 until the lower bound of m_1 becomes negative in Constraint (2.20). However, if we allow m_1 to be negative, the algorithm could trade off m_2 for a negative m_1 to minimize the sum of m_1 and m_2.

MILP Formulation for Cut and Block Mask Co-Optimization

We extend the MILP in Section 2.2.2 by providing *cut candidates*. Figure 2.22 illustrates block and cut candidates with one possible final layout after cut and block mask application. Figures 2.22(a) and (b) show block candidates and cut candidates, respectively. We note that the leftmost block candidate $v_{1,1}$ is generated considering a given "EOL margin" to allow block mask to realize the EOL of signal wire

[22] Here, we assume the initial "WNS" is negative. For designs with positive WNS (i.e., worst slack), we can easily shift the "zero slack" threshold to establish a guardband that preserves the worst slack of the original design. (See also Footnote 20 above.)

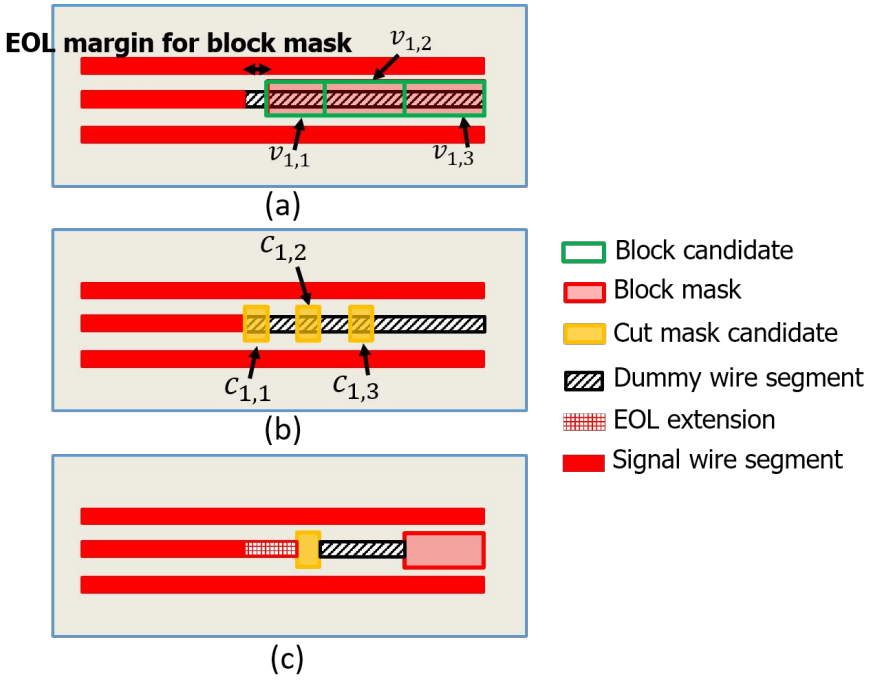


Figure 2.22: Cut and block mask co-optimization: (a) block candidates; (b) cut candidates; and (c) a possible final layout.

segment. We use $10nm$ as the EOL margin in our experiments. To realize the EOL of the signal wire next to the block mask, we must select at least one cut or block candidate from among the cut and block candidates. Figure 2.22(c) shows the final layout when $v_{1,3}$ and $c_{1,2}$ are selected as the final block and cut candidate solutions, respectively.[23]

We now formulate the MILP problem for the cut and block mask co-optimization. Table 2.6 again shows notations that we use in our formulation. Analogous to the block mask MILP above, the objective is to minimize the total timing degradation arising from EOL extensions and final dummy fill patterns for timing-critical paths. s_p and m_p are calculated in the same way. However, for the delay increase d_p, we now consider the impact from both the EOL extensions as well as the dummy fills that affect path p.

We now describe constraints in our cut and block mask co-optimization with the exception of the minimum metal density and timing constraints since these two constraints are the same as in the block mask optimization.

[23]We note that a block candidate cannot replace a cut candidate due to the larger EOL margin for block mask shapes. I.e., cut (resp. block) candidates cannot be replaced by block (resp. cut) candidates even though they might share their locations.

Constraints for LELE cuts. In the case of non-selective LELE cuts, Constraint (2.24) enforces cut uniqueness. Binary variable $c_{i,j}^{f}$ indicates whether the cut candidate j for shape i on cut mask f is selected, as shown in Constraint (2.24). For non-selective LELE, we assume that two cut masks are available, i.e., $f = 1, 2$. For the selective cut approach, we assume only one cut mask is available, i.e., $f = 1$.

Constraints for cut and block mask rule violation. Constraints (2.26) and (2.27) prevent cut and block mask rule violations. Constraint (2.26) is the same as Constraint (2.18) in block mask optimization. Similar to Constraint (2.26) for block candidates, we enumerate sets of conflicting cut candidates and prevent them from co-existing with Constraint (2.27).

Constraints for EOL realization. Constraint (2.25) enforces EOL realization. We use index i' to indicate a shape which is the only existing shape between any two horizontally adjacent signal segments. In other words, shape i' is a dummy shape that connects two neighboring signal segments, and must be split by a cut or a block to realize the EOL of the two signal segments. Thus, we enforce the rule that at least one cut or block exists for shape i'.

Minimize: $$\sum_{p} m_p \tag{2.23}$$

Subject to:

$$\sum_{f} c^{f}_{i,j} = c_{i,j_l}, \quad \forall i,j \tag{2.24}$$

$$\sum_{j} v_{i',j} + \sum_{j} c_{i',j} \geq 1, \quad \forall i' \tag{2.25}$$

$$\sum_{\substack{(i,j)\in B_{q,a} \\ (i',j')\in B_{q,b}}} v_{i,j} + (1 - v_{i',j'}) < |B_{q,a}| + |B_{q,b}|, \ \forall q \tag{2.26}$$

$$\sum_{\substack{(i,j)\in C_{q,a} \\ (i',j')\in C_{q,b}}} c_{i,j} + (1 - c_{i',j'}) < |C_{q,a}| + |C_{q,b}|, \ \forall q \tag{2.27}$$

$$e_{i,x_l(x_r)} \geq v_{i,j}, \quad \text{if } \ j_l(j_r) = x, \ \forall i \tag{2.28}$$

$$e_{i,x_l(x_r)} \geq c_{i,j}, \quad \text{if } \ j_l(j_r) = x, \ \forall i \tag{2.29}$$

$$e_{i,x_l(x_r)} \leq v_{i,j} + c_{i,j'}$$

$$\text{if } \ j_l(j_r) = x, \ j'_l(j'_r) = x, \ \forall i \tag{2.30}$$

$$e_{i,x_l} - \sum_{x'<x} e_{i,x'_l} - {e'}_{i,x_l} \leq 0,$$

$$e_{i,x_r} - \sum_{x'>x} e_{i,x'_r} - {e'}_{i,x_r} \leq 0, \ \forall i, \ \forall x \tag{2.31}$$

$${e'}_{i,x_l} \leq e_{i,x_l}, \quad {e'}_{i,x_r} \leq e_{i,x_r}, \ \forall i \tag{2.32}$$

$$\sum_{x_l} {e'}_{i,x_l} \leq 1, \quad \sum_{x_r} {e'}_{i,x_r} \leq 1, \ \forall i \tag{2.33}$$

$$L + \sum_{i} (l_i - \sum_{j} r_{i,j} \cdot v_{i,j}) \geq d_{min} \tag{2.34}$$

$$\sum_{k\in K_p} (\sum_{i,j} t^{k}_{i,j} \cdot (1 - v_{i,j}) + \sum_{i,x_l} {t'}^{k}_{i,x_l} \cdot e'_{i,x_l}$$

$$+ \sum_{i,x_r} {t'}^{k}_{i,x_r} \cdot e'_{i,x_r}) \leq s_p + m_p, \quad \forall p \tag{2.35}$$

$$m_p \geq 0, \quad \forall p \tag{2.36}$$

Constraints for EOL definition. Constraints (2.28) – (2.30) find the leftmost (resp. rightmost) edge for shape i from a selected cut or block candidate, since this candidate determines EOL for the signal wire segment on its left (resp. right). Binary variable e_{i,x_l} (resp. e_{i,x_r}) indicates whether location x is the left (resp. right) edge of any selected cut or block candidates for shape i. Constraints (2.31) - (2.33) describe the methodology to find the leftmost (resp. rightmost) selected cut or block candidate. Constraints (2.31) – (2.32) ensure that $e'_{i,x_l(x_r)} = 1$ if $e_{i,x_l(x_r)} = 1$ and x is the location of leftmost (rightmost) edge for shape i. Otherwise, $e'_{i,x_l(x_r)} = 0$ is forced by checking whether e variables that are associated with x' are equal to one, where $x' < x$ ($x' > x$) for e'_{i,x_l} (e'_{i,x_r}) in Constraint (2.31). Figure 2.23 illustrates variable e'. In the figure, we assume that $c_{1,1} = 1$ and $v_{1,3} = 1$. e variables are computed in Constraints (2.37) by Constraints (2.28) – (2.30). Constraints (2.38) – (2.40) correspond to Constraint (2.31). Constraint (2.41) corresponds to Constraint (2.33). As a result, $e'_{1,2_l}$ becomes equal to one, which indicates that location $x = 2$ is the EOL, as shown in Figure 2.23.

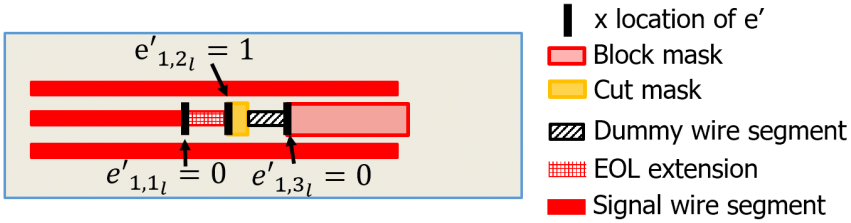


Figure 2.23: Illustration of binary variable e': cut candidate $c_{1,1}$ and block candidate $v_{1,3}$ are selected.

$$e_{1,1_l} = 0; \;\; e_{1,2_l} = 1; \;\; e_{1,3_l} = 1 \tag{2.37}$$

$$e_{1,1_l} - e'_{1,1_l} \leq 0 \tag{2.38}$$

$$e_{1,2_l} - e_{1,1_l} - e'_{1,2_l} \leq 0 \tag{2.39}$$

$$e_{1,3_l} - e_{1,1_l} - e_{1,2_l} - e'_{1,3_l} \leq 0 \tag{2.40}$$

$$e'_{1,1_l} + e'_{1,2_l} + e'_{1,3_l} \leq 1 \tag{2.41}$$

Timing Model for Dummy Wire Segments

Dummy wire segments cause net capacitance increase (Δcapacitance), and hence gate and wire delay increase. This timing impact of dummy wire segments should be minimized so that the performance and robustness of designs with dummy wire segments can be consistent with (or, better than) designers' expectations at signoff. We now describe how we model Δcapacitance, along with resulting changes to gate and wire delays, to capture timing impact of dummy wire segments in our optimization flow.

Capacitance model. To model the timing impact of floating dummy wire segments, we first characterize capacitance increase of signal nets due to neighboring dummy segments. Fill-aware capacitance extraction must comprehend various situations (e.g., upper / lower layers, and types of neighboring wire segments of the dummy / signal wires) [32][52]. However, to obtain linear expressions that we can incorporate into our MILP formulation, we study the impact of a dummy wire segment on capacitance of a signal wire in four simplified situations (cases) according to the distance between a signal wire and a dummy segment: (i) one track away (the dummy segment is on a neighboring track of the signal segment); (ii) two tracks away; (iii) three tracks away; and (iv) four tracks away. For each case, we experiment with different parallel run lengths of the dummy wire segment to a signal wire, and measure the capacitance of the signal wire to extract the coefficients. We use *Cadence Innovus Implementation System v15.2* [112] for parasitic RC extraction with *Cadence Quantus Extraction Solution v15.2* [113] techfiles provided by our collaborators at a leading technology consortium. Table 2.7 shows normalized capacitance increase per unit length for (grounded) EOL extension, and cases (i) - (iv) for (floating) dummy segments from this section above.

Table 2.7: Normalized capacitance increase for (grounded) EOL extension and (floating) dummy fill, using a Cadence Innovus-based extraction flow provided by our collaborators at a leading technology consortium.

Case	**EOL**	**(i)**	**(ii)**	**(iii)**	**(iv)**
Δcap	1270	342	53	5	1

Gate and wire delay model. We use linear gate and wire delay models. The linear delay models are fast and easy to incorporate into an MILP formulation. Also, for the very small Δcapacitance values

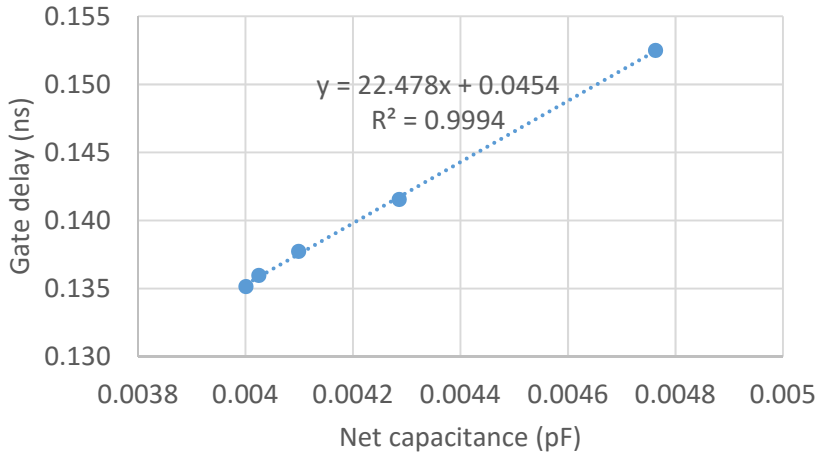


Figure 2.24: Gate delay vs. net capacitance for a specific gate instance.

caused by dummy wire segments, linear delay modeling shows good accuracy. We use *Cadence Tempus Timing Signoff Solution v15.2* [114] to extract delays for each gate and net given extracted SPEF files of (i) layout design with dummy wire segments for only clock nets, and (ii) layout design with dummy wire segments for all nets. We then use the linear delay model to extract coefficients. Timing coefficient extraction is performed for each gate instance and driving net.[24] Figure 2.24 shows an example of extracted coefficients (i.e., determining a linear equation for gate delay vs. capacitance) of a specific gate instance.

Validation of our timing model. We validate our timing model by comparing with timing results obtained from *Cadence Tempus Timing Signoff Solution v15.2* [114]. We report stage and timing path delays calculated based on our model (and, which are used in our ILP formulation) and compare them with timing results from *Tempus*. We observe that estimated values and golden values from *Tempus* are quite similar, as shown in Figure 2.25. The maximum errors are -4ps and -23ps (a negative value means optimistic) for stage delay and path delay, respectively. To compensate for the errors, we add timing margin of 50ps in our ILP formulation for all studies reported below.

[24]We note that for different instances of the same library cell (master), the coefficients are not the same since the instances' output nets have different load capacitances according to the circuit structure. We do not separately model slew (transition time) changes that are due to the Δcapacitance changes. This is because (i) we already achieve high accuracy by modeling each gate and net separately, and (ii) fill-induced slew changes are very small, since the associated capacitance and delay changes are small. Our implementation takes 20 minutes to extract coefficients for every gate in the *JPEG* testcase, using a single thread.

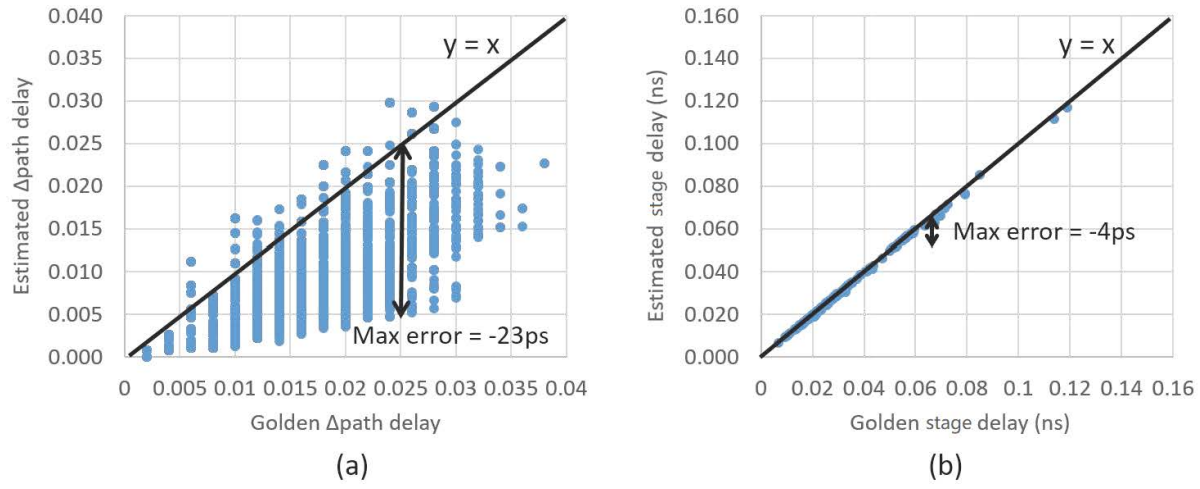


Figure 2.25: Comparison of timing results from *Tempus* (Golden) and our estimation (Estimated). (a) Path delay and (b) stage delay comparisons. The maximum errors are -4ps and -23ps for stage delay and path delay, respectively.

2.2.3 Overall Flow

We now describe the overall flow of our optimizations, including conflict list enumeration and distributed optimization.

Conflict List Enumeration

Algorithm 2 Enumeration for minimum spacing constraint.

1: **for** each block candidate pair $(v_{i,j}, v_{i',j'}) \in V$ **do**
2: **if** $S(v_{i,j}, v_{i',j'}) < S_{min}$ **then**
3: $B_{q,a} \leftarrow \{v_{i,j}, v_{i,j}\}$;
4: **for** each block candidate $v_{k,l}$ located between $v_{i,j}$ and $v_{i',j'}$ **do**
5: $B_{q,b} \leftarrow B_{q,b} \cup \{v_{k,l}\}$;
6: **end for**
7: $q \leftarrow q + 1$;
8: **end if**
9: **end for**

Minimum spacing violation. Algorithm 2 describes the enumeration for minimum spacing constraint. For each pair of block candidates ($v_{i,j}$,$v_{i',j'}$) within minimum spacing, we add the candidate pair to $B_{q,a}$ (Line 3). They are typeA candidates, where the inclusion of each candidate (on the block mask) results in a violation (see Section 2.2.2). We then enumerate all block candidates that are located between

$v_{i,j}$ and $v_{i',j'}$ and add them to $B_{q,b}$ (Lines 4-6). These candidates are typeB candidates, where the exclusion of each candidate ensures that the candidate pair ($v_{i,j}$,$v_{i',j'}$) is separated. Figure 2.26 shows horizontal and vertical minimum spacing violations. For the ($v_{1,1}$,$v_{4,1}$) pair, let us assume that the vertical spacing between $v_{1,1}$ and $v_{4,1}$ is less than the minimum spacing. Then, $B_{q,a} = \{v_{1,1}, v_{4,1}\}$, and $B_{q,b} = \{v_{2,1}, v_{3,1}\}$, since $v_{2,1}$ and $v_{3,1}$ are located between $v_{1,1}$ and $v_{4,1}$. As an another example, for the ($v_{1,1}$,$v_{1,3}$) pair, let us assume that the horizontal spacing between $v_{1,1}$ and $v_{1,3}$ is less than the minimum spacing. Then, $B_{q,a} = \{v_{1,1}, v_{1,3}\}$, and $B_{q,b} = \{v_{1,2}\}$.

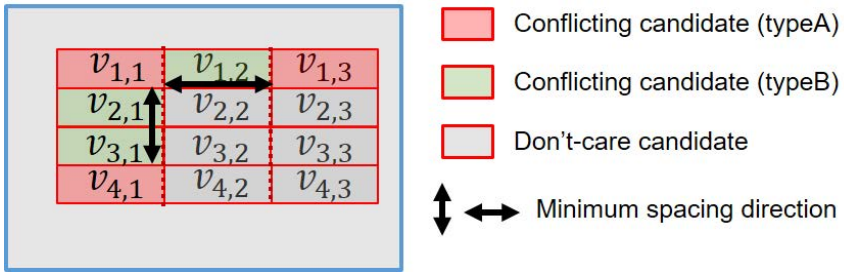


Figure 2.26: Illustration of conflict list enumeration for minimum spacing constraint, showing horizontally and vertically conflicting pairs.

Other design rules. The enumeration of conflict lists for other rules can be applied similarly by collecting all typeA and typeB candidates.

Distributed Optimization

The most critical limitation of the MILP-based approach in practice is runtime. To achieve a scalable approach, we adopt the distributable optimization approach that has been previously proposed by Han et al. in [40].

We first partition the layout into small clips and optimize in four iterations. In each iteration, we select clips that are not adjacent to each other and optimize the clips in parallel. For example, we optimize all clips in the following sequence in our four iterations: (i) clips in odd rows and odd columns in the first iteration; (ii) clips in odd rows and even columns in the second iteration; (iii) clips in even rows and odd columns in the third iteration; and (iv) clips in even rows and even columns in the fourth iteration. With this approach, as shown in Figure 2.27, the target clips (yellow) do not share their boundaries with each other. Thus, each target clip can be optimized without creating any interference between clips. After each

iteration, we save block/cut solutions for optimized clips. The solutions are used in the following iterations as boundary conditions. In our implementation, we set the clip size to be $8\times8\mu m^2$ and the boundary width to be $0.6\mu m$. The local minimum metal density constraint is enforced within each clip. Note that with this approach, speedup is effectively linear in compute resources. We report the results of our scalability test in Section 2.2.4.

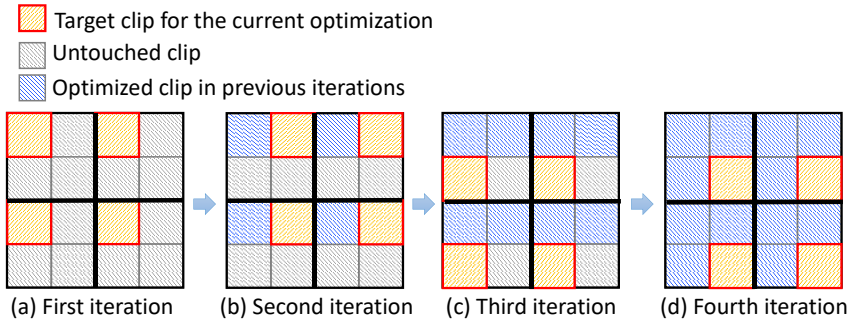


Figure 2.27: Distributed optimization: (a) – (d) respectively illustrate the first, second, third and fourth iteration in our approach. Since target clips (yellow) for an iteration do not share their boundaries with each other, each target is independently optimizable.

Overall Optimization Flow

Figure 2.28 shows our overall optimization flow. We start from a routed design and candidate block (and cut) shapes that cover dummy segments. We then optimize in four iterations per metal layer. In each iteration, we optimize small clips that are independently optimizable in parallel. In an iteration, we (i) generate block (and cut) candidates for each shape, (ii) generate sets of conflict candidates with our block (and cut) mask rule checker, and (iii) formulate and solve our MILP with pre-characterized timing coefficients and local minimum metal density constraints. After four iterations, we obtain the optimized block/cut mask layout and perform timing/power/capacitance evaluations with *Cadence Innovus Implementation System v15.2* [112] and *Cadence Tempus Timing Signoff Solution v15.2* [114].

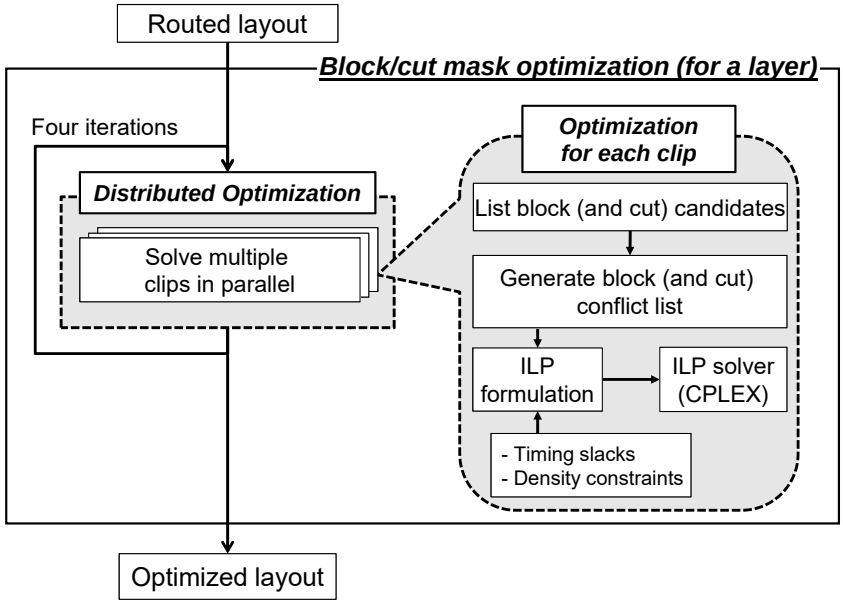


Figure 2.28: Overall optimization flow.

2.2.4 Experiments

Experimental Setup

We implement our optimizations in C++ with *OpenAccess 2.2.6* [128] to support LEF/DEF [119], and with *CPLEX 12.5.1* [115] as our MILP solver.[25] We evaluate our approach using two design blocks (*AES* and *JPEG*) from *OpenCores* [124], and an Arm Cortex-M0 core (*M0*) without memories. We synthesize the designs with *Synopsys Design Compiler vH-2013.03-SP3* [129] from RTL netlists and then perform placement and routing with *Cadence Innovus Implementation System v15.2* [112] using an IMEC N7 (i.e., $7nm$ foundry node) library [118]. All experiments are performed with 24 threads on a 2.6GHz Intel Xeon dual-CPU server. (As noted above, runtimes will generally see linear speedup with added compute resources.)

[25] We use one thread for each CPLEX instance. Based on our experiments, solving multiple MILP instances in a serial fashion with CPLEX parallel optimization takes longer time than solving multiple MILP instances together with a single thread for each instance. For *JPEG* design with the same total 24 threads, the runtime with CPLEX parallel optimization is 9010 sec, but the runtime with our optimization method is 4146 sec.

Design of Experiments

We perform three types of experiments: **ExptA** studies the tradeoff between solution quality and runtime. **ExptB** studies 2D block mask optimization. And **ExptC** studies cut and block mask co-optimization. In ExptA, we apply our optimizer to layouts with various numbers of dummy segments and clip sizes to show the tradeoff between solution quality and runtime. (We use the results to determine the best setting for input parameters.) For ExptB on 2D block mask optimization, we use a cut mask-aware post-route layout with EOL extension already defined by a commercial tool. For ExptC on cut and block co-optimization, we perform cut and block optimization to define EOL and dummy removal using our software. We describe details of our design of experiments as follows.[26]

- **ExptA-1**: Sensitivity study on the effect of block candidates. We trade off dummy removal rate and runtime for different block candidate lengths. We vary the block candidate length from $40nm$ (1.2X minimum metal pitch) to $160nm$ (5X minimum metal pitch) in steps of $20nm$.

- **ExptA-2**: Sensitivity study on the effect of clip size. We trade off dummy removal rate and runtime for different clip sizes. We vary the clip sizes from $2\mu m \times 2\mu m$ to $10\mu m \times 10\mu m$. In both experiments A-1 and A-2, we use non-timing-aware (i.e., "timing-oblivious") optimization, which is achieved by simply maximizing the removal of dummy fill.[27]

- **ExptB-1**: Comparison of timing-aware and non-timing-aware optimizations.

- **ExptB-2**: Comparison of the performance impact of 193i and 193d block mask rules (summarized in Table 2.5). We use a loose 20% minimum metal density constraint to demonstrate the upper bound of performance impact from patterning technology.

[26] We note that it is hard to make an apples-to-apples comparison between our work and previous works since the objectives of our work and previous works are fundamentally different. The algorithms proposed in previous works are dedicated to solving the problem formulations posed in those works; they are difficult to extend and adapt to handle our complex design rules. For example, the work [109] simply minimizes the number of edges of each polygon of block mask patterns, and is not based on explicit design rules. Additionally, timing constraints are not considered. Similarly, the work [110] applies very limited and simple design rules, which gives a very different context from the detailed rules (obtained from our collaborators at a large industry consortium) that we use in our work.

[27] Specifically, the non-timing-aware objective is to minimize $\sum_i (l_i - \sum_j r_{i,j} \cdot v_{i,j})$, with notations as defined in Table 2.6. In other words, the objective is to minimize Δarea of final block mask shapes, compared to a block mask layout covering all dummy segments. Note that we disable timing-awareness by removing Constraint (2.20) in Section 2.2.2.

- **ExptB-3**: Comparison of the performance difference with selective and non-selective block approaches. We again use a loose 20% minimum metal density constraint to demonstrate the upper bound of performance impact from patterning technology.

- **ExptB-4**: Comparison of the impact of metal density constraints. We study 20%, 30% and 40% minimum metal densities.[28]

- **ExptC-1**: Comparison of cut and block mask co-optimization to a sequential cut and block mask optimization. A cut mask only optimization is enabled without generating block shape candidates.

- **ExptC-2**: Comparison of selective cut and LELE cut approaches.

The testcases are summarized in Table 2.8. Table 2.9 summarizes parameter settings for each type of experiment.

Table 2.8: Summary of testcases.

Expt type	Design	#Inst	#Nets
A, B	*M0*	11194	11457
B	*AES*	10010	10066
	JPEG	52753	52778
C	*M0*	9884	9951
	AES	13381	13656
	JPEG	54012	54155

Experimental Results

Table 2.11 shows the experimental results of ExptB and ExptC. For ExptB, the **Timing Impact Recovery** column shows timing improvements. The timing impact recovery is measured in ns against a design with no dummy segments removed (worst case). The percentage shown indicates how closely our optimizations can approach a design that assumes all dummy segments are removed (best, or ideal, case).[29]

[28] Without block mask, a SADP/SAQP-based uni-directional design implies ∼50% metal density, assuming metal width equal to spacing.

[29] For example, if WNS is 0.000ns (resp. -0.100ns) for the best (resp. worst) case, and we achieve -0.030ns in WNS after block mask optimization, we recover 0.070ns in WNS, with a recovery percentage of 70%.

Table 2.9: Parameter settings for the experiments.

ExptA				
Expt	**Timing/non-timing**	**Layers**	**Clip width (μm)**	**Block candidate length (nm)**
A-1	non-timing	M3	2	60 - 160
A-2	non-timing	M3	2 - 10	120
Default setup		design = *M0* density LB = 0% non-selective block mask		

ExptB				
Expt	**Timing/non-timing**	**193i/193d**	**selective/non-selective**	**Density LB (%)**
B-1	both	193i	selective	40
B-2	timing	both	selective	20
B-3	timing	193i	both	20
B-4	timing	193i	selective	20, 30, 40
Default setup		design = *M0*, *AES*, *JPEG* layers = M2, M3, M4, M5 clip size = $4\mu m \times 4\mu m$ block candidate length = $120nm$		

ExptC		
Expt	**co-optimization/sequential**	**selective/LELE cut**
C-1	both	selective cut
C-2	co-optimization	both
Default setup	design = *M0*, *AES*, *JPEG* layers = M2, M3, M4, M5 clip size = $4\mu m \times 4\mu m$ block candidate length = $120nm$ density LB = 20% 193i mask, selective block mask	

The best and worst cases serve as extreme, baseline data points for ExptB. Table 2.10 shows WNS, total negative slack (TNS) and switching power (P_{sw}) of the best and worst cases, At the worst case, WNS (resp. TNS) degradation is up to $0.114ns$ (resp. $47.853ns$) for testcase *JPEG*. The switching power is increased by up to 3.4%. *Dummy removal rate* is calculated as the removed dummy segment length over the sum of removed and remaining dummy segment length.

Table 2.10: Timing and switching power of best and worst cases for ExptA. The units are ns, ns and μW for WNS, TNS and P_{sw}, respectively.

Design	**Best case**			**Worst case**		
	WNS	**TNS**	P_{sw}	**WNS**	**TNS**	P_{sw}
M0	-0.030	-1.737	4.06	-0.092	-23.86	4.17
AES	-0.037	-1.417	10.77	-0.069	-5.827	11.08
JPEG	-0.047	-9.583	39.18	-0.161	-57.436	40.53

ExptA-1: Sensitivity study on the effect of block candidates. Figure 2.29(a) shows dummy removal rate and runtime results for various block candidate lengths. In the range of $60nm$ to $160nm$, we see that the block candidate length does not significantly affect the dummy removal rate. However, the runtime increases proportionally to the block candidate length.

ExptA-2: Sensitivity study on the effect of clip size. Figure 2.29(b) shows dummy removal rate and runtime results for various clip sizes. In the range of $2\mu m$ to $10\mu m$, we see that the clip size does not significantly affect the dummy removal rate. However, the runtime increases as the clip size increases.

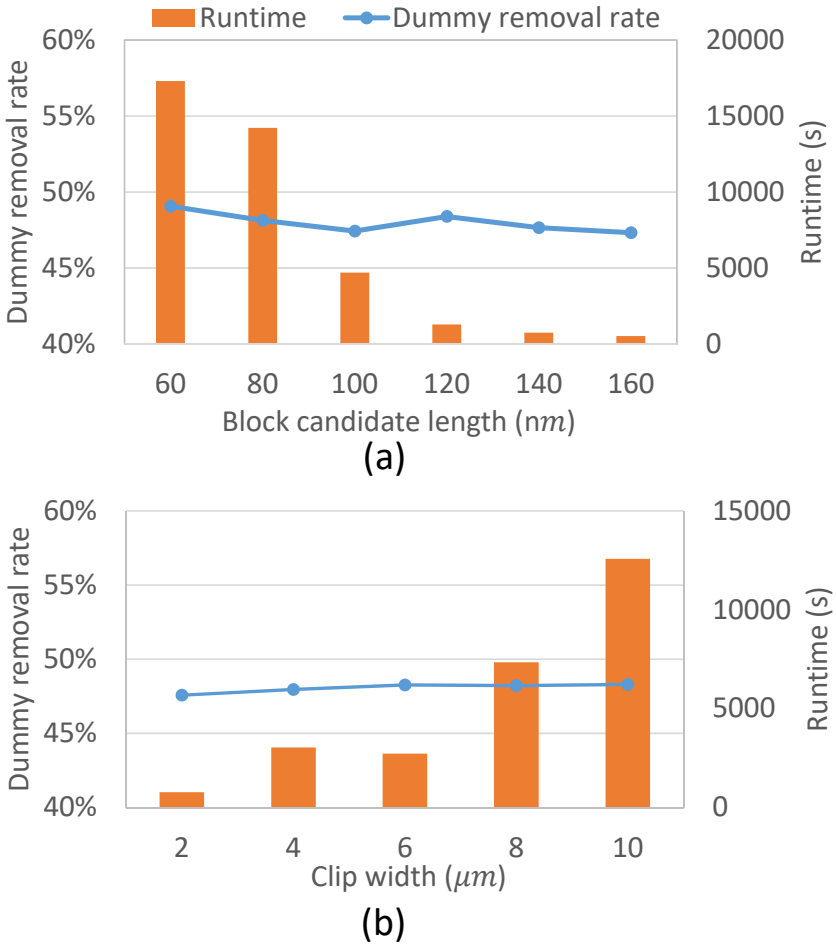


Figure 2.29: Sensitivity study results: sensitivity of dummy removal rate to (a) block candidate length and (b) clip size.

ExptB-1: Comparison of timing-aware and non-timing-aware optimizations. We observe that non-timing-aware optimization results in higher dummy removal rates than timing-aware. However,

timing-aware optimizations shows better timing impact recovery. Averaged over all three designs, timing-aware optimization recovers 57% (resp. 69%) of ΔWNS (resp. ΔTNS), compared to 32% (resp. 35%) recovered by non-timing-aware optimization. The results demonstrate that our timing-aware optimization helps recover timing with less dummy removal. We also see that the runtime of timing-aware optimization is 76% smaller on average than non-timing-aware.

ExptB-2: Comparison of 193i and 193d selective block mask rules. This experiment shows the impact of patterning options. On average, application of 193i selective block mask recovers 75% (resp. 81%) of ΔWNS (resp. ΔTNS), while application of 193d selective block mask recovers 36% (resp. 48%) of ΔWNS (resp. ΔTNS). For switching power, application of 193i selective block mask recovers 53%, compared to 27% for 193d, on average. For dummy removal rate, 193i selective block mask improves by up to 43% over 193d (*JPEG*, metal layer M4, 62% vs. 19%), with an average improvement of 21%.

ExptB-3: Comparison of selective and non-selective approaches. The selective block mask approach affords better control of dummy removal, since the minimum width of a block mask shape for a dummy segment is twice as large in the selective block mask case as in the non-selective block mask case. This results in much greater overlay margin in the selective block mask case. The results show that the selective block mask approach recovers up to 84% and on average 75% of ΔWNS, while the non-selective block mask approach recovers up to 39% and 25% on average of ΔWNS. For ΔTNS, the selective block mask approach recovers up to 86%, and 81% on average; the non-selective block mask approach recovers up to 42% and 33% on average. Regarding ΔP_{sw}, the average recovery rates are 53% and 18% for selective and non-selective mask approaches, respectively. The timing and power benefits of the selective block approach come from high dummy removal rates; we see that the dummy removal rates are larger for the selective block approach in all designs.

ExptB-4: Comparison of different metal density constraints. As metal density lower bounds increase, dummy segment removal becomes more restricted. We observe that the dummy removal rates drop by up to 36% (*JPEG*, M4, 51% vs. 26%) with higher density constraints. With respect to timing and power, our experimental results show the expected tradeoff between timing/power and density constraints. We see that with higher density constraints, as dummy removal is more restricted, the final timing and

Table 2.11: Overall experimental results. Values in parentheses denote percentage improvements (reductions) with respect to the worst case as described in Table 2.10. Note that ExptA and ExptB use cut-aware (from commercial tool) and cut-unaware post-route layout, respectively.

Experiment	Design	Option	Timing Impact Recovery (ns)		$\Delta\ P_{sw}$ (μW)	Dummy removal rate (%)				Runtime (s)
			ΔWNS	ΔTNS		M2	M3	M4	M5	
B-1	*M0*	Timing-aware	0.035 (56%)	17.307 (78%)	-0.041 (36%)	24	32	31	22	823
		Non-timing-aware	0.022 (35%)	8.945 (40%)	-0.039 (34%)	59	36	33	27	4451
	AES	Timing-aware	0.014 (43%)	2.516 (57%)	-0.129 (42%)	30	40	38	29	716
		Non-timing-aware	0.010 (31%)	1.569 (35%)	-0.116 (37%)	60	43	41	30	4231
	JPEG	Timing-aware	0.080 (70%)	34.194 (71%)	-0.458 (33%)	28	29	26	15	4150
		Non-timing-aware	0.033 (28%)	14.401 (30%)	-0.372 (27%)	56	29	27	23	11773
B-2	*M0*	193i	0.039 (62%)	17.681 (79%)	-0.052 (46%)	24	39	40	25	956
		193d	0.035 (56%)	13.097 (59%)	-0.034 (30%)	6	23	31	25	1963
	AES	193i	0.025 (78%)	3.482 (78%)	-0.170 (55%)	31	47	51	49	643
		193d	0.008 (25%)	1.755 (39%)	-0.095 (30%)	6	21	30	42	1307
	JPEG	193i	0.096 (84%)	40.960 (85%)	-0.759 (56%)	22	56	62	33	4146
		193d	0.030 (26%)	21.143 (44%)	-0.247 (18%)	4	16	19	10	6751
B-3	*M0*	Selective	0.039 (62%)	17.681 (79%)	-0.052 (46%)	24	39	40	25	956
		Non-selective	0.024 (38%)	9.280 (41%)	-0.023 (20%)	12	17	21	14	2992
	AES	Selective	0.025 (78%)	3.482 (78%)	-0.170 (55%)	31	47	51	49	643
		Non-selective	0.007 (21%)	1.414 (32%)	-0.076 (24%)	12	21	26	29	1319
	JPEG	Selective	0.096 (84%)	40.960 (85%)	-0.759 (56%)	22	56	62	33	4146
		Non-selective	0.018 (15%)	11.390 (23%)	-0.121 (8%)	7	8	10	5	6347
B-4	*M0*	Density LB 20%	0.039 (62%)	17.681 (79%)	-0.052 (46%)	24	39	40	25	956
		Density LB 30%	0.041 (66%)	17.577 (79%)	-0.054 (48%)	24	37	41	28	1005
		Density LB 40%	0.035 (56%)	17.307 (78%)	-0.041 (36%)	24	32	31	22	823
	AES	Density LB 20%	0.025 (78%)	3.482 (78%)	-0.170 (55%)	31	47	51	49	643
		Density LB 30%	0.025 (78%)	3.487 (79%)	-0.169 (55%)	31	46	51	49	748
		Density LB 40%	0.014 (43%)	2.516 (57%)	-0.129 (42%)	30	40	38	29	716
	JPEG	Density LB 20%	0.096 (84%)	40.960 (85%)	-0.759 (56%)	22	56	62	33	4146
		Density LB 30%	0.092 (80%)	39.868 (83%)	-0.702 (52%)	22	56	51	27	4375
		Density LB 40%	0.080 (70%)	34.194 (71%)	-0.458 (33%)	28	29	26	15	4150

Experiment	Design	Option	Timing (ns)		P_{sw} (μW)	Removal rate (%)				Runtime (s)
			WNS	TNS		M2	M3	M4	M5	
C-1	*M0*	Co-optimization	-0.139	-39.844	3.537	29	36	30	24	1947
		Sequential	-0.284	-136.515	3.815	12	21	18	20	1748
	AES	Co-optimization	-0.107	-21.567	18.685	29	37	35	24	1691
		Sequential	-0.132	-34.452	20.103	13	12	17	21	1460
	JPEG	Co-optimization	-0.014	-0.071	74.609	20	18	14	9	8015
		Sequential	-0.042	-0.404	70.772	9	12	12	11	8972
C-2	*M0*	LELE cut	-0.139	-39.844	3.537	29	36	30	24	1947
		Selective cut	-0.103	-20.482	3.475	35	37	28	20	1180
	AES	LELE cut	-0.107	-21.567	18.685	29	37	35	24	1691
		Selective cut	-0.08	-17.515	18.341	32	37	33	22	1165
	JPEG	LELE cut	-0.014	-0.071	74.609	20	18	14	9	8015
		Selective cut	-0.067	-1.293	74.669	18	18	10	7	5730

power outcomes worsen.[30] The average percentage recovery of ΔWNS is 75% (resp. 75%, 57%) for a density lower bound of 20% (resp. 30%, 40%). The average percentage recovery of ΔTNS is 81% (resp. 81%, 69%) for a density lower bound of 20% (resp. 30%, 40%). And, the recovery of P_{sw} impact is 53%

[30] We see that for *M0*, this trend is reversed between the 20% and 30% density lower bounds. The reason might be that the 20% and 30% density lower bounds are already too loose for this design, such that the lower bounds do not constrain dummy removal. Similarly, we do not see much difference in timing and power for the *AES* design.

(resp. 52%, 38%) on average for a density constraint of 20% (resp. 30%, 40%). For *M0*, we see that the dummy removal rate for M4 and M5 at 20% density is slightly lower than at 30% density. This is because different density constraints lead to different solutions for each iteration (clip), and our timing-aware optimization does not target maximum dummy removal rate.

C-1: Comparison of co-optimization and sequential optimization. We observe that WNS from co-optimization shows up to $0.146ns$ improvement compared to WNS from sequential optimization. For TNS, we observe $96.671ns$ (71%) improvement for *M0*, $12.885ns$ (37%) for *AES*, and $0.333ns$ (82%) for *JPEG*. We also achieve improved (reduced) switching power with our co-optimization. This is because in the sequential approach, the EOL of all signal wire segments must be defined using only cut masks, which increases EOL extensions. On the other hand, the co-optimization approach has more flexibility with cut and block masks for the EOL realization of signal wire segments. Thus, better timing and power are achieved with smaller EOL extensions. For dummy removal rate, we also observe higher removal rate for the co-optimization, indicating that our co-optimization enables a broader solution space than the sequential cut and block approach. We emphasize to the reader that the "removal rate" for ExptC is different from "dummy removal rate" in ExptB. Removal rate is calculated as the quotient of (removed dummy segment length) divided by (sum of EOL extension length, removed dummy segment length, and remaining dummy segment length), since EOL extension is generated in ExptB.

C-2: Comparison of selective cut approach and LELE cuts. Our results indicate that the selective cut approach achieves up to $36ps$ better WNS as compared to the LELE cut approach for *M0* and *AES*. This is because selective cuts can be merged when they are aligned on non-adjacent tracks that are adjacent in the given color (e.g., cuts on first and third tracks) although signal segments exist in between, while LELE cuts in the same color will violate the minimum spacing rule. However, for *JPEG*, the LELE cut approach shows better WNS. We believe that the results can be highly dependent on the routing pattern (e.g., if we have more alignment opportunity on neighboring tracks, LELE could align more cuts with the same color cut). Therefore, it is very important for the router to understand the patterning technology for the cut. Power and TNS follow the trend of WNS.

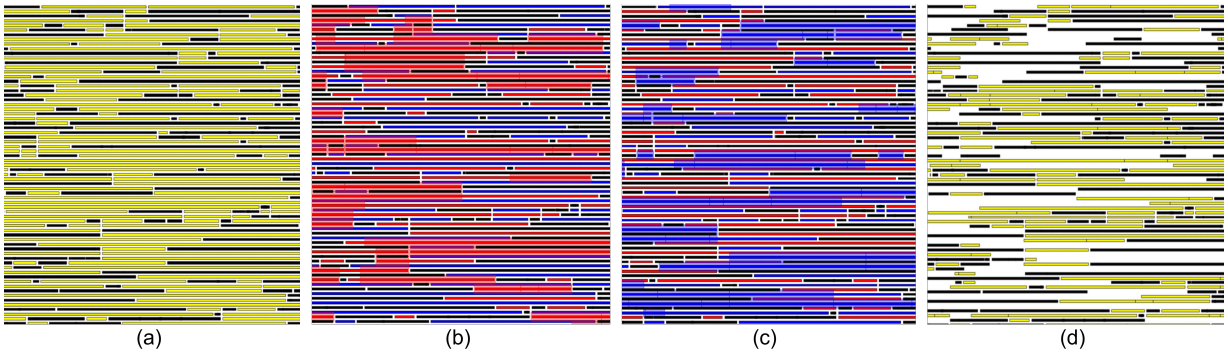


Figure 2.30: Layouts of M4 layer before and after dummy fill removal: (a) initial layout with dummy fill; (b) layout covered by the selective block mask (red); (c) layout covered by the selective block mask (blue); and (d) layout after timing-aware dummy fill removal with optimized selective block masks.

2.2.5 Conclusion

In this work, we present a scalable MILP-based optimization of 2D block masks that considers block mask rules, minimum metal density constraints, and timing impact of dummy fills. We propose an improved timing impact model for use in our MILP formulation. A distributed optimization flow enables application of the MILP-based optimization to large design layouts. We evaluate our approach across timing-awareness, different patterning technologies, and different minimum metal density constraints. Our study shows up to 84% ΔWNS recovery, up to 85% ΔTNS recovery, and up to 56% Δswitching power recovery, along with up to 62% dummy removal rate. We believe that our enablement of a timing-aware optimization shows promising product-level benefits from use of 2D block masks, and furthermore sheds light on the merits of various block mask optimization objectives. We have also studied the *co-optimization* of cut and block masks. Our cut and block co-optimization opens up a broader solution space, with more flexibility in EOL realization and attendant design quality benefits.

2.3 Acknowledgments

Chapter 2 contains the reprints of Andrew B. Kahng, Jiajia Li and Lutong Wang, “Improved Flop Tray-Based Design Implementation for Power Reduction”, *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2016; and Peter Debacker, Kwangsoo Han, Andrew B. Kahng, Hyein Lee, Praveen Raghavan and Lutong Wang, “MILP-Based Optimization of 2D Block Masks for Timing-Aware

Dummy Segment Removal in Self-Aligned Multiple Patterning Layouts", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36(7), 2017. The dissertation author is a main contributor to, and a primary author of, the second paper.

I would like to thank my coauthors Peter Debacker, Kwangsoo Han, Andrew B. Kahng, Hyein Lee, Jiajia Li and Praveen Raghavan for their support and work.

Chapter 3

Improved Physical Design Methodologies in Placement

This chapter presents two methodologies targeted to the placement stage of physical design. First, we present a detailed placement methodology for neighbor diffusion effect mitigation and better model-hardware correlation. Our methodology consists of optimal dynamic programming-based single-row/double-row and multi-row detailed placement optimizations that considers displacement and HPWL. Our optimization supports movable and fully-reorderable multi-height cells, including reordering between multi-height cells and inter-row cell movements. Our optimization maximizes the *diffusion step* reduction to mitigate the neighbor diffusion effect in order to reduce model-hardware miscorrelation and yield loss, with up to 98% inter-cell *diffusion step* reduction. Our formulation is further extended for a potential timing-aware optimization that leads to 6$\times$ increase in *intentional steps* around timing-critical cells. Second, we present a detailed placement methodology to reduce congestion and wirelength. Our methodology consists of a mixed integer linear programming-based optimization for two cell architectures that are relevant in sub-10nm process nodes, and considers and exploits inter-row M1 routing. We adopt a distributed, window-based optimization to overcome the runtime limitation, achieving up to 6.4% total routed wirelength reduction, and up to 14.4% #via12 reductions, with no adverse timing impact.

3.1 Enhanced Optimal Multi-Row Detailed Placement for Neighbor Diffusion Effect Mitigation in Sub-10nm VLSI

In advanced technology nodes, device behavior no longer depends on independent geometrical parameters [24]. Due to aggressive device scaling, lithography limitations and process complexity, *layout-dependent effect* (LDE) arises from the proximity of devices, and significantly affects device performance. An important type of LDE is *neighbor diffusion effect* (NDE) [8], where the horizontal spacing between diffusion regions changes the performance of transistors. Figure 3.1(a) illustrates different diffusion spacing caused by diffusion height changes between four transistors. If the heights of neighboring diffusion regions are different, there is a diffusion *step*, e.g., transistor *T2* has a diffusion *step* to each of *T1* and *T3*.

More specifically, the drive strength (i.e., I_{on}) and the leakage power (i.e., I_{off}) of a transistor fin is a function of the horizontal spacing to the adjacent diffusion regions of the transistor fin. Since NDE changes the electrical characteristics of transistors, it affects the power, performance and area of designs [8]. For example, Figure 3.1(a) shows the transistor fins A and B with the spacings to their neighboring diffusion area, i.e., d_A and d_B, respectively. As d_A and d_B are different, I_{on} and I_{off} of the two transistor fins are different (e.g., $I_{off}(A) = f(d_A) \neq I_{off}(B) = f(d_B)$) due to the change in V_{th} [8]. For example, given a single inverter with a diffusion *step* next to the PFET and a diffusion *step* next to the NFET, the impacts to the two devices in combination result in higher leakage.

In this work, we use a bimodal assumption to simplify the NDE problem: for a given transistor, either of two leakage values holds, depending on whether the diffusion region on the nearest neighboring site of the transistor has *full height* (that is, same or larger height), or *less height*, compared to the transistor's diffusion height. The leakage difference for the above two cases is linear with #*steps*, e.g., a diffusion height difference of two steps results in $2\times$ leakage difference compared to that of one step. In a conventional place-and-route flow, intra-cell NDE (i.e., NDE effect within a standard cell) is captured by library characterization since the diffusion shapes within a cell are pre-determined. However, it is difficult to capture inter-cell NDE since neighboring diffusion shapes are determined by detailed placement. Thus, in general, library characterization always assumes existence of a full-height neighboring diffusion region on standard cell boundaries, which causes miscorrelation between the model (i.e., library) and the

hardware (i.e., actual diffusion shapes at standard cell boundaries and their device leakage impacts) in a design. Minimizing diffusion *steps* in detailed placement is a key idea toward reduction of model-hardware miscorrelation.

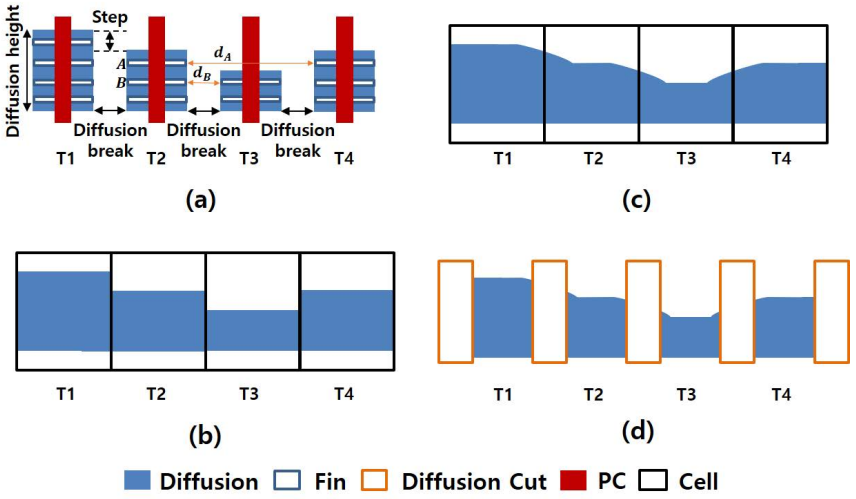


Figure 3.1: (a) Diffusion *step* and fin spacing, (b) desired pattern, (c) actual diffusion region showing corner rounding, and (d) diffusion breaks (after diffusion cuts applied).

With aggressive device scaling, the diffusion *step* not only causes NDE, but also induces a increase in the process complexity due to the limited resolution of conventional 193i lithography. In advanced nodes, the diffusion shapes of transistors are merged and patterned as a single polygon; the transistors are then separated by using diffusion breaks (which are achieved by applying diffusion cuts) [99], as shown in Figure 3.1(a). Figure 3.1(b) illustrates the desired pattern of a single polygon to generate the diffusion regions of four transistors. The actual pattern of the polygon (showing corner rounding in lithography) is shown in Figure 3.1(c). Figure 3.1(d) illustrates the final printed diffusion layout with diffusion cuts. At the boundaries of diffusion where diffusion *steps* exist, fin shapes and diffusion shapes are distorted due to the corner rounding phenomena. A distorted and/or sharp-angled end of a fin may cause an increase in electrical field, resulting in gate oxide breakdown [92]. Further, such distorted diffusion shapes change the diffusion height and fin length, which can cause dramatic shifts in threshold voltage (V_{th}), or even device failure in sub-10*nm* nodes.[31] This V_{th} shift has negative impact on design performance and quality. For example, V_{th} variation can cause setup time and/or hold time violations in a design. As a result, the

[31] According to our collaborator [75], there can be $> 150mV$ V_{th} shift in the 10LPE node.

maximum frequency that the design can achieve is reduced, or the design can even fail with hold time violations due to ultra-low V_{th} which cannot be recovered.

For a motivating study, we define a (inter-cell NDE-induced) *cell failure* to occur if the boundary transistor has a $> 100mV$ V_{th} shift compared to the average V_{th} for all transistors. According to [75], the failure rate of a transistor with a diffusion *step* is twice as high as a transistor without a diffusion *step* (base failure rate). The solid lines in Figure 3.2 show the yield vs. (initial) number of diffusion *steps* ($\sim$ #cells) with different base failure rates. We assume #$steps$ is approximately proportional to #cells, which holds for testcases in Section 3.1.5. The dashed lines in Figure 3.2 show the projected yield for the same chip if we can reduce 90% of diffusion *steps*. In our preliminary study, more than 60% of standard cells (cell-boundary transistors) have inter-cell diffusion *steps*. For a relatively small design block *VGA* (85% utilization in an N7 (foundry $7nm$) design enablement, 69K cells and 50K diffusion *steps* initially), we assume a base failure rate of 1ppm and can achieve 3.6% yield improvement by removing 90% of diffusion *steps*. For a commercial design with multiple hundreds of millions of cells and diffusion *steps*, if we assume a more realistic 1ppb base failure rate, then we can achieve $\sim 3\%$ yield improvement by removing 90% of diffusion *steps*.[32] In light of this, minimizing diffusion *steps* helps to recover the yield of designs by reducing V_{th} (and thus speed) variation of transistors.

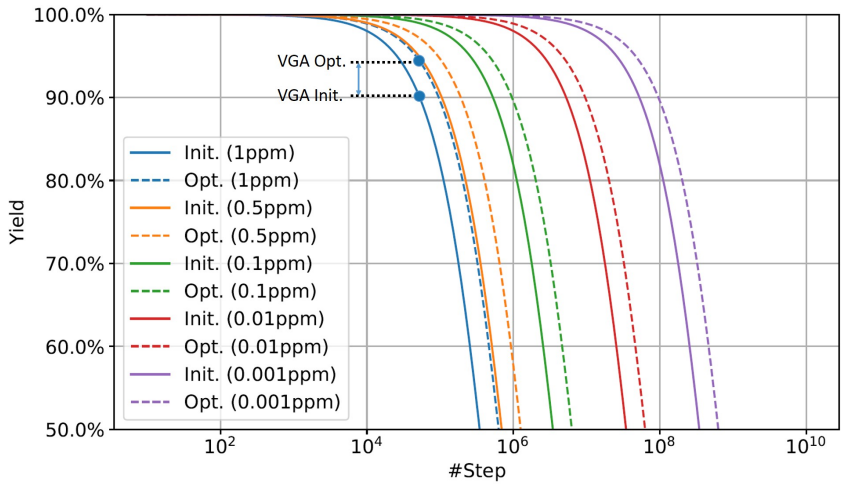


Figure 3.2: Initial (Init.) and projected (Opt.) yield assuming 90% inter-cell $step$ reduction for various base failure rates.

[32] Based on guidance from our collaborator [75], after scaling to account for our small testcase sizes, we assume a base failure rate of 1ppm for each *step* in our experiments with small design blocks. See Table 3.4 in Section 3.1.5.

Current limitations and our approach. In order to reduce diffusion *steps*, special non-functional *filler cells* are instantiated between functional cells [79] as we elaborate in Section 3.1.2 below. However, opportunities for *step*-reducing filler cell insertion are limited given a fixed layout, and this approach (effectively similar to cell padding) is expensive in terms of area. Other works [21] [67][93][105] propose graph-algorithmic or dynamic programming methods to resolve complex design rules in advanced nodes. However, the solution spaces considered are typically limited due to the assumption of (ordered)-single-row placement.[33] Recent works [66][98] on multi-row detailed placement involve heuristic approaches, and no advanced-node rules are considered. Han et al. [36] propose an optimal single-row and double-row dynamic programming for detailed placement optimization, allowing cell reordering with support of double-height cells.

In this work, we extend our previous single-row and double-row detailed placement framework [36] with HPWL-awareness and with multi-row detailed placement optimization. Our main contributions are summarized as follows.

- We extend the optimal single-row dynamic programming-based approach [36] to an HPWL-aware version. The proposed approach minimizes and balances diffusion *steps* and HPWL cost. Our proposed algorithm is capable of all types of cell movements – i.e., cell variants, relocating, and reordering (specifically, *P-reordering* with $P > 2$).

- We propose a new multi-row dynamic programming, with support of movable, and fully-reorderable, multi-height cells, including reordering between multi-height cells. Inter-row cell moving within each optimization window (in multiple of rows) is intrinsically supported, and further improves solution quality.

- We propose metaheuristics to use both single-row HPWL-aware optimization and multi-row optimization to achieve better solution quality.

- We extend our formulation to a potential timing-aware optimization that leads to $6\times$ increase in *intentional* steps around timing-critical cells to improve the timing performance.

[33] Lin et al. [67] propose a *P-reordering* problem. However, only *2-reordering* (i.e., neighbor cell switching) is presented. We describe our methodology to handle the *P-reordering* problem in Section 3.1.2.

- We improve the solution quality over [36] by achieving up to 98% inter-cell diffusion *step* reduction compared to 90% achieved in [36], while consuming similar runtime.

The remainder of this work is organized as follows. Section 3.1.1 reviews related works. Section 3.1.2 describes the problem formulation and dynamic programming-based single-row detailed placement methodology. Section 3.1.3 describes the double-row detailed placement flow. Section 3.1.4 describes the multi-row detailed placement flow. In Section 3.1.5, we describe our experimental setup and results. Section 3.1.6 gives conclusions and directions for ongoing work.

3.1.1 Related Work

We classify relevant previous works on detailed placement into three categories: (i) detailed placement for advanced nodes, (ii) mixed cell-height placement, and (iii) NDE-aware detailed placement.

Detailed placement for advanced nodes. To support complex design rules introduced in advanced nodes, the objectives of detailed placement have changed from classical objectives (e.g., wirelength reduction [44][47][50][53][61][81]) in recent years. The works of [67][93][105] resolve triple-patterning issues. Yu et al. [105] propose shortest path and dynamic programming algorithms to solve the ordered single row (OSR) placement. Tian et al. [93] develop a weighted partial MAX SAT approach to solve the OSR problem. Lin et al. [67] propose a local reordered single row refinement (LRSR) and implement a 2-reordering (i.e., neighboring cell switching) approach using a unified graph model. Du and Wong [21] apply a shortest-path algorithm supporting flipping and 2-reordering to address the drain-drain abutment problem in FinFET-based cell placement. The works of [12][39] propose mixed integer linear programming (MILP)-based methods to comply with drain-drain abutment, minimum implant area and minimum oxide jog length rules, and to increase vertical M1 connections.

Mixed cell-height placement. Wu et al. [98] propose a pairing technique to handle double-height cells for detailed placement. Their method simply groups or inflates cells so that all cells become double-height cells, after which a conventional detailed placer can be used. Recently, Lin et al. [66] have proposed a *chain move* scheme along with a nested dynamic programming-based approach to support multiple cell-height placement. They first perform chain moves to save wirelength cost. On top of this, dynamic

programming is applied to solve the nested shortest path problem. Other techniques [18] are developed to support non-integer-ratio (e.g., mixture of 8T and 12T cells) mixed cell-height placement.

NDE-aware placement. Ou et al. [80] perform NDE-aware analog placement by modifying and integrating a compact model for NDE into an existing analog placement algorithm. Oh et al. [79] develop special filler cells to mitigate NDE.

Han et al. [36] (which this work builds on) propose to resolve the NDE problem in the detailed placement stage. Inter-cell diffusion *steps* are minimized by trying to match the diffusion heights of neighboring cells. If two neighboring cells have different diffusion heights, special filler cells can be inserted to reduce diffusion *steps*. [36] proposes single-row and double-row dynamic programming optimizations that support cell relocating, reordering and flipping as well as double-height cells. They support reordering between single-height cells, and between a single-height cell and a double-height cell, but not between two double-height cells.

In summary, many works such as [21][67][93][105] propose graph or dynamic programming models to resolve complex design rules in advanced nodes. However, their solution spaces are limited by the assumption of (ordered)-single-row placement. Two recent works [66][98] on multi-row detailed placement give heuristic approaches, but no advanced node rules are considered. Our previous work [36] proposes dynamic programming-based methods to optimize single-row and double-row placements, systematically supporting cell reordering and double-height cells. However, the dynamic programming formulation cannot be extended to support more than two rows, and the formulation cannot support reordering between two double-height cells. Notably, our present work advances over [36], and is distinguished from previous approaches, in several ways. (i) We formulate an optimal (HPWL-aware) single-row and multi-row dynamic programming-based approach to minimize a cost function that includes diffusion *steps*. (ii) We support a richer set of cell movements than in previous works – i.e., flipping, relocating *and* reordering – via a systematic methodology to handle *P-reordering* with $P > 2$. Specifically, our multi-row approach intrinsically supports *inter-row* cell relocation. (iii) Our formulation supports multi-height cells with movable, and fully-reorderable, multi-height cells.

3.1.2 Single-Row Optimization

In this section, we describe the problem statement and our dynamic programming formulation for single-row detailed placement.

Single-Row Optimization Problem. *Given an initial legalized single-row placement, perturb the placement to minimize inter-cell diffusion* steps.

Inputs: A legalized single-row placement, available cell variants, and cost function of a diffusion *step*.

Output: Optimized single-row detailed placement with minimized overall cost (including inter-cell diffusion *steps*).

Constraints: Maximum displacement range, maximum reordering range, availability of cell flipping.

Filler Cell and Step Costs

Table 3.1: Cost for one diffusion *step*.

Spacing (sites)	0	1	2	3	4+
Cost	1	$+\infty$	1	1	0

Table 3.1 describes inter-cell diffusion *step* cost. For each pair of adjacent cells, if there are zero, two or three empty sites in between, the cost is equal to the number of inter-cell diffusion *steps*; if there are at least four empty sites in between, the cost is always zero. That is, with four or more empty sites we can always assume proper filler cell insertions resulting in no inter-cell diffusion *steps*. Figure 3.3 shows an example of filler cell insertion between two functional cells that have different diffusion heights at edges that face each other. If the two functional cells have fewer than four empty sites in between, filler cells can only match one of the diffusion heights. As a result, there always exists at least one diffusion *step* that affects one of the two functional cells. However, with a spacing of four or more sites, a legal diffusion height transition can always be achieved by one or more contiguous filler cell(s). Thus, the filler cell(s) can match both the diffusion heights of the two functional cells. In a relevant advanced technology, the minimum filler cell width is two placement sites due to process limitations. Therefore, adjacent functional cells must abut, or have at least two empty sites between them, in order to insert a

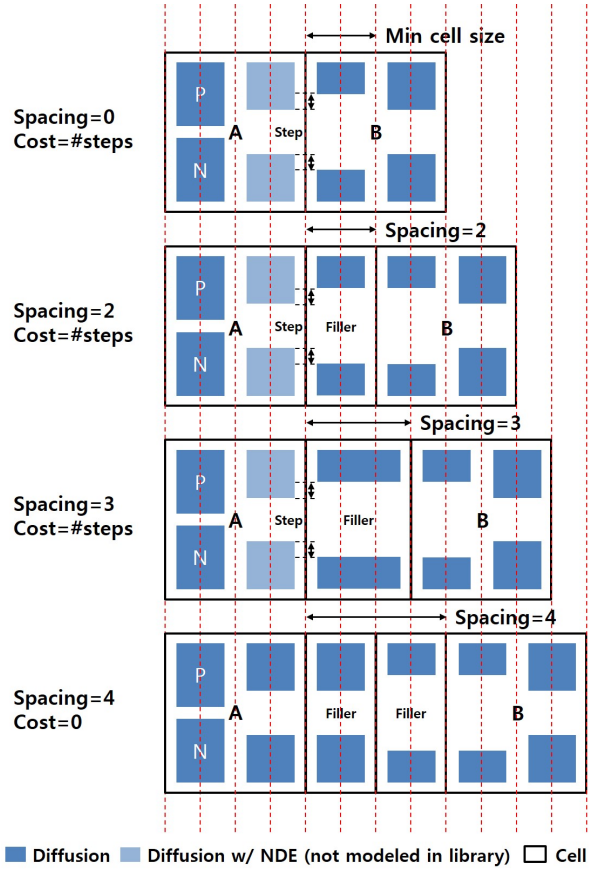


Figure 3.3: Filler insertion between cell A and B, given different spacings.

filler cell [75]. In our implementation, we avoid single-site spacings by assigning infinite cost to such scenarios, as indicated in Table 3.1. Even though our optimization does not explicitly allocate white space, the dynamic programming (presented later in Sections 3.1.2, 3.1.3 and 3.1.4) itself can utilize/change the local white space distribution by cell relocating and cell reordering within specified ranges. Filler cell cost is explicitly included in our dynamic programming cost calculation, such that our optimization is aware of both whitespace and filler insertion as it trades off between (i) abutting two cells without filler insertion at the cost of diffusion *steps*, and (ii) leaving four placement sites for a proper filler insertion in an effort to minimize the diffusion *steps* between neighboring cells.

Table 3.2: Notations.

Notation	Meaning
C	set of cells in a window of initial placement
c_k	k^{th} cell in the left-to-right ordered initial placement, i.e., k is the cell index
v	a cell variant
$w_{k,v}$	width of c_k with a variant v
$[-x_\Delta, x_\Delta]$	horizontal displacement range
x_k	absolute x coordinate of c_k in the initial placement, in units of placement sites
l	displacement from the initial placement, in units of placement sites
$[-r, r]$	reordering range
i	number of placed cells
j	position shift from the initial placement
s	placement status array
$d[i][j][v][l][s]$	minimum cost when i cells are placed with case (j,v,l,s)
The notations below apply only to multi-row optimization	
$[-y_\Delta, y_\Delta]$	vertical displacement range
y_k	absolute y coordinate of c_k in the initial placement, in units of rows
m	number of rows in an optimization window
b	row index in an optimization window
d_b	for the b^{th} row, d_b is the distance between the rightmost boundary of b^{th} row, and the rightmost boundary of all rows in the optimization window
D	distance array of d_b in an optimization window (i.e. $[d_0...d_{m-1}]$)
t_b	for the b^{th} row, type of the rightmost cell (e.g., 2-fin, 3-fin or 4-fin)
T	type array of t_b in an optimization window (i.e., $[t_0...t_{m-1}]$)
$\{D, T\}$	boundary condition
$[D][T]$	forming boundary condition $\{D, T\}$
$d[i][j][v][l][s][D][T]$	minimum cost when i cells are placed, forming boundary condition $\{D, T\}$

Notations

Table 3.2 shows notations used in our formulation. For each cell c_k, **cell index** k is its (left-to-right) sequentially ordered position in the initial placement. Given a set of cells (C) in a row of an initial placement, the leftmost cell is c_1, and the rightmost cell is $c_{|C|}$.

For each c_k, we define **cell variants** (v) which correspond to different cell orientations and cell layouts with the same functionality. To minimize #diffusion *steps*, we can use several variants of a cell with the same functionality, for which layouts have different diffusion heights. In our experiments below, $v = 0$ indicates the cell orientation in the initial placement, and $v = 1$ indicates the flipped (i.e., mirrored

about the y-axis) cell orientation. $w_{k,v}$ is the width of cell c_k with variant v, in units of placement sites. Flipping a cell does not change the set of sites that the cell occupies.

We define the **displacement range** $[-x_\Delta, x_\Delta]$ as the constraint that a cell cannot move more than x_Δ sites from its initial placement. We use x_k to denote the initial right x coordinate of c_k, in units of placement sites. Thus, c_k can be placed with its right x coordinate in the interval $[x_k - x_\Delta, x_k + x_\Delta]$. We use l to denote the **displacement** (in sites) from the initial cell placement (i.e., $l \in [-x_\Delta, x_\Delta]$). For the cells on the boundary of the die, we make sure that the displacement range will not extend beyond the die boundary.

We support cell reordering with a **reordering range** $[-r, r]$, i.e., given r, in the placement solution c_k can have a new sequentially ordered position within the range $k - r, k - r + 1, \ldots, k + r$.

In our dynamic programming, we place one cell at a time from left to right, and the index i is used to indicate that i cells have been placed. Given a cell reordering range $[-r, r]$, cells c_k with $k < i - r$ are placed; those with $i - r \leq k \leq i + r$ may or may not be placed; and those with $k > i + r$ are not placed. For the $2r + 1$ cells such that $i - r \leq k \leq i + r$, we use a binary array s to denote the *placement status* of each cell. Here, s is a binary array of size $(2r + 1)$, i.e., $s \in \{0, 1\}^{2r+1}$. Each bit in the array indicates whether the corresponding cell is placed or not. For example, if we have six cells c_1 to c_6, $i = 4$ and $r = 1$, then s captures the placement status of the $(2 \cdot 1 + 1 = 3)$ cells c_3, c_4 and c_5. $s = [0, 1, 1]$ means that c_3 is not placed, while c_4 and c_5 are placed. Figure 3.4 illustrates six placement solutions with three legal states when $i = 4$. In this example, c_1 and c_2 must be placed and c_6 must not be placed. We note that the indices of s correspond to k (position in the initial placement), but not the final position. For example, $s[0]$ always represents the status for c_3, and $s[2]$ always represents the status for c_5, regardless of the actual sequence of positions, as shown in Figure 3.4(b). Also, when we have placed i cells, since cells with index $k < i - r$ must be placed, we must have placed $i - (i - r - 1) = r + 1$ cells that have cell index $i - r \leq k \leq i + r$. Thus, at all times, a legal status array s has exactly $r + 1$ elements equal to 1. In the above example, s always has $1 + 1 = 2$ elements equal to 1.

Given i, to identify the last placed cell c_k (that is, the i^{th} cell to have been placed), we define the **position shift** as j, where $k = i + j$. For example, in Figure 3.4(c), given $i = 4$, the position shift $j = -1$ indicates that the last placed cell is c_3, since $3 = 4 + (-1)$.

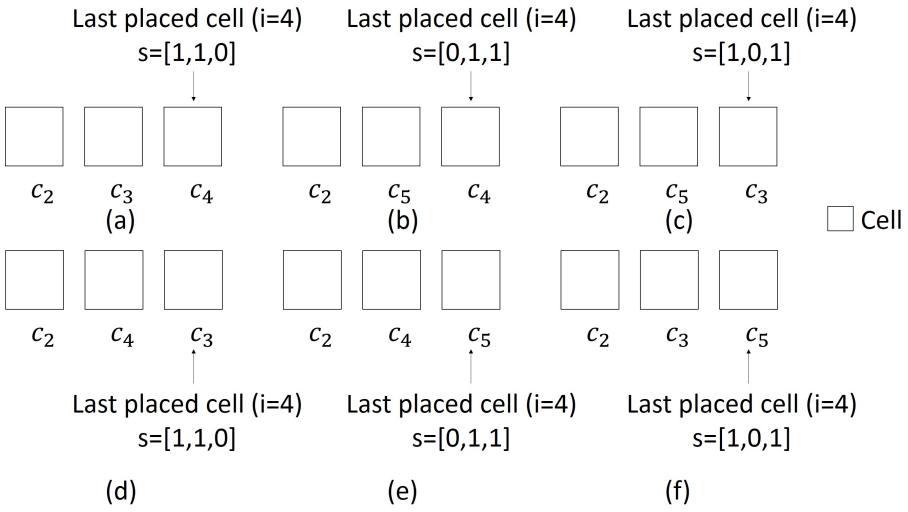


Figure 3.4: Illustration of six placement solutions with three legal states given $i = 4$ and $r = 1$.

At the heart of our dynamic programming recurrence, we use $d[i][j][v][l][D][T][s]$ to represent the minimum cost when i cells have been placed. Note that in single-row case, the dimensions of D and T are both zero. Therefore, the dynamic programming array can be reduced to $d[i][j][v][l][s]$. From this array, we can obtain the last placed cell c_k, where $k = i + j$. We can also tell the variant v in use, the displacement l, and the status s for cell c_k. We define the above as *case* (j, v, l, s), with i implicitly given, for simplicity. Therefore, we complete the row placement once we reach $i = |C|$, and we obtain the optimal solution by finding the minimum cost among all cases of $i = |C|$. In our implementation, we store a pointer for each entry in the DP array so that the optimized placement can be traced back from $d[|C|][j][v][l][s]$ all the way to $d[0][j][v][l][s]$.

Dynamic Programming Formulation

Algorithm 3 describes our dynamic programming (DP) procedure for single-row placement in detail. Line 2 initializes the DP solution array. Lines 3 – 13 describe the main algorithm. Starting with placing the first cell, the algorithm incrementally adds (places) cells next to the current partial placement solution. Procedure $getNext()$ returns a list of legal $next$ cells and the respective status of each of these cells. Along with legal (j', s') from Line 5, Line 6 checks all possible cases (v', l') considering placement legality and displacement constraints, as shown in Equation (3.1). Lines 7 – 9 update the minimum cost for

the case (j', v', l', s') when we place the $i' = (i+1)^{st}$ cell. In Lines 14 – 17, we obtain the minimum cost among all legal cases when $i = |C|$, and Line 18 returns the minimum cost for the current row.

$$x_{i+j} + l + w_{i+j,v} \leq x_{i'+j'} + l' \tag{3.1}$$

The function $cost(^{i',j',v',l'}_{i,j,v,l})$ calculates the cost as a weighted sum of (i) diffusion *step* cost, (ii) displacement cost, and (iii) cell variant cost, as shown in Equation (3.2). The diffusion *step* cost is calculated as total #inter-cell diffusion *steps* between the i^{th} and $(i')^{th}$ placed cells. The displacement cost is equal to the absolute value of l'. In this work, we assume that the given initial placement solution has adequate quality in terms of various metrics, including but not limited to pin accessibility, global routability, etc. Thus, we simplify other optimization objectives as one "displacement minimization" objective. As noted above, in this work we assume two cell variants: original orientation and flipped orientation. We set the variant cost to one if a cell is flipped ($v' = 1$), and zero otherwise. Two weighting factors α and β (β can be seen as supplementing α by capturing an equivalence between cell flipping and displacement) are used to balance the three cost terms. We describe experiments regarding the impact of weighting factors in Section 3.1.5.

$$cost(^{i',j',v',l'}_{i,j,v,l}) = cost_{step} + \alpha \cdot cost_{disp} + \alpha \cdot \beta \cdot cost_{var} \tag{3.2}$$

Algorithm 4 details our methodology to obtain $next$ status. That is, given the binary status array for i, we construct the status array for $i' = i+1$. Line 2 initializes the list of next available $(cellIndex, status)$ combinations. In Line 3, we first shift the status array for i one bit to the left to obtain the cell placement status for $i' = i + 1$. Then, Lines 4 – 9 check whether cell $c_{i'-r}$ must be placed as the $(i')^{th}$ cell. If we do not place $c_{i'-r}$ as the $(i')^{th}$ cell, then cell $c_{i'-r}$ will be placed out of its reordering range. Thus, we set $s[-r] = 1$ and return so that we make sure to choose $c_{i'-r}$ as the $(i')^{th}$ cell. Lines 10 – 16 check whether any binary indicator $s[m]$ is equal to zero. If so, $c_{i'+m}$ could be the next legally placed cell. In such a case, we add $(m, nextStatus)$ to the list.

Algorithm 3 Dynamic programming (single-row)

Initialize for all legal cases (j, v, l, s)
  $d[0][j][v][l][s] \leftarrow 0$, $d[i][j][v][l][s] \leftarrow +\infty, (0 < i \leq |C|)$
for all $i = 0$ to $|C| - 1$ **do**
  for all $d[i][j][v][l][s] \neq +\infty$ **do**
    for all $(j', s') \in$ getNext(s) **do**
      for all (v', l') **do**
        $i' = i + 1$
        $t \leftarrow d[i][j][v][l][s] + cost(^{i',j',v',l'}_{i\ ,j\ ,v\ ,l})$
        $d[i'][j'][v'][l'][s'] \leftarrow \min(d[i'][j'][v'][l'][s'], t)$
      end for
    end for
  end for
end for
$finalCost \leftarrow \infty$
for all (j, v, l, s), $i = |C|$ **do**
  $finalCost \leftarrow \min(d[|C|][j][v][l][s], finalCost)$
end for
Return $finalCost$

HPWL-Aware Optimization

We mitigate the wirelength impact of single-row *step* optimization by modifying the cost function. Specifically, we add a ΔHPWL cost component to the function $cost(^{i',j',v',l'}_{i\ ,j\ ,v\ ,l})$, as shown in Equation (3.3).

$$cost(^{i',j',v',l'}_{i,j,v,l}) = cost_{step} + \alpha \cdot cost_{disp} + \alpha \cdot \beta \cdot cost_{var} + \gamma \cdot cost_{\Delta HPWL} \tag{3.3}$$

We calculate the $cost_{\Delta HPWL}$ by summing up the ΔHPWL contribution of cell c_k over all nets incident to c_k, in the same way as in [53]. $cost_{\Delta HPWL}$ captures the impact of a cell's placement on bounding box sizes of incident nets. We use a new weighting factor γ to balance the four cost terms. We describe experiments regarding the impact of weighting factors in Section 3.1.5.

3.1.3 Double-Row Optimization

In this section, we describe the problem statement and the dynamic programming approach for *double-row detailed placement considering double-height cells as well as reordering, flipping and available cell variants*.

Algorithm 4 Procedure $getNext$ (single-row)

Inputs: s
Initialize $nextList \leftarrow \emptyset$
$s \leftarrow$ shiftLeft1Bit(s)
if $s[-r] = 0$ **then**
  $s[-r] \leftarrow 1$
  $nextStatus \leftarrow s$
  $nextList \leftarrow nextList \cup \{(-r, nextStatus)\}$
  Return $nextList$
end if
for all $m \in [-r, r]$ **do**
  if $s[m] = 0$ **then**
    $nextStatus \leftarrow s$
    $nextStatus[m] \leftarrow 1$
    $nextList \leftarrow nextList \cup \{(m, nextStatus)\}$
  end if
end for
Return $nextList$

Double-Row Optimization Problem. *Given an initial legalized double-row placement with double-height cells, perturb the placement within each row to minimize inter-cell diffusion* steps.

Inputs: Legalized double-row placement, available cell variants, and cost function of a diffusion *step*.

Output: Optimized double-row detailed placement with minimized overall cost (including inter-cell diffusion *steps*).

Constraints: Maximum displacement range, maximum reordering range, availability of cell flipping.

Assumptions

We make the following assumptions with respect to this problem statement.

Assumption 1. *Cell rows can be fully separated from each other every two consecutive rows.* In the case of placement rows that contain only single-height cells, the assumption is correct by definition. However, for any cell row, a double-height cell that occupies sites in the row must span to either the upper neighboring row or the lower neighboring row, but not both. Figure 3.5(a) shows such separable pairs of cell rows, where rows 1 and 2 (with double-height cells A and B) do not interfere with rows 3 and 4 (with double-height cells C and D). By contrast, in Figure 3.5(b), row 2 has double-height cells E and F which interfere with both row 1 and row 3, violating our assumption. Given the interleaving of VDD/VSS power

rails in modern libraries, our assumption is normally satisfied. In other words, all double-height cells in the current technology node tend to have the same power rail configuration. (In Figure 3.5(b), cell F has a different type of power rail design (VDD-VSS-VDD) than the other double-height cells (VSS-VDD-VSS).) We do not have such double-height library cells in the current technology node.[34]

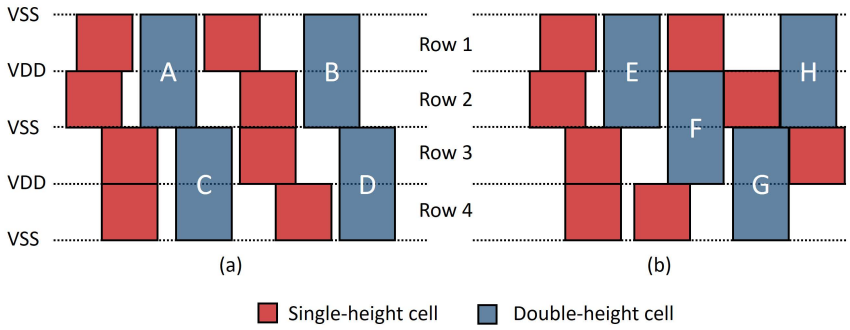


Figure 3.5: Illustrations of double-height cells in placement rows. (a) Separable pairs of cell rows, reflecting power rail design of double-height cells in current N10 libraries. (b) Non-separable pairs of cell rows.

Assumption 2. *The relative positions among double-height cells are fixed.*[35] For two double-height cells A and B, if A is initially to the left of B ($x_A < x_B$), then we require that in our final placement, c_A remains to the left of c_B. We note that we still allow reordering between a single-height cell and a double-height cell (thus, the double-height cells are *partially reorderable*) so as to maximize the *steps* reduction.

Formulation

Given the above assumptions, our approach can provide optimal placement solutions for two consecutive rows sharing common double-height cells as in Figure 3.5(a). Overall, double-row optimization uses single-row optimization as a basic building block. From each double-height cell, we invoke separate single-row optimizations that progress left-to-right in each of the two rows, and merge the solutions once

[34]Our collaborator [75] at a major advanced foundry indicates that all double-height cells have only one power rail configuration in the 10LPE node. Cells with height of four or more rows account for less than 1% of all instances, and thus our formulation can be easily adopted if we just assume that these very large (height $\geq$ four rows) cells are fixed.

[35]The double-height cell effectively breaks the two rows into separate optimization regions, wherein we invoke single-row optimization separately for the two rows. The prerequisite is that we know exactly what instance is the "next double-height cell", which requires that relative positions be unchanged for double-height cells. This assumption will be lifted below in Section 3.1.4.

Algorithm 5 Dynamic programming (double-row)

1: **Initialize** $DHCellList \leftarrow getOrigDHOrdering()$
2: **Initialize costs for all legal CASES** $(v, l, j_0, s_0, j_1, s_1)$
3: $D[0][v][l][j_0][s_0][j_1][s_1] \leftarrow 0$
$D[I][v][l][j_0][s_0][j_1][s_1] \leftarrow +\infty, (0 < I \leq |DHCellList| + 1)$
4: **for all** $I = 0$ to $|DHCellList|$ **do**
5: **for all** $D[I][v][l][j_0][s_0][j_1][s_1] \neq +\infty$ **do**
6: **for all** legal $(v', l', j_0', s_0', j_1', s_1')$ **do**
7: $I' = I + 1$
8: $t \leftarrow D[I][v][l][j_0][s_0][j_1][s_1] + Cost(^{I',v',l',j_0',s_0',j_1',s_1'}_{I\ ,v\ ,l\ ,j_0,s_0,j_1,s_1})$
9: $D[I'][v'][l'][j_0'][s_0'][j_1'][s_1'] \leftarrow$
$\min(D[I'][v'][l'][j_0'][s_0'][j_1'][s_1'], t)$
10: **end for**
11: **end for**
12: **end for**
13: **for all** $(v, l, j_0, s_0, j_1, s_1)$ **when** $I = |DHCellList|$ **do**
14: $sol \leftarrow \min(D[I][v][l][j_0][s_0][j_1][s_1], sol)$
15: **end for**
16: **Return** sol

we encounter the next double-height cell. The merging is designed to preserve all optimal candidates, while enabling movable and partially reorderable double-height cells. Our development is similar to that of Algorithm 3, where we saw that given the minimum costs of all *cases* (j, v, l, s) for i, we could derive the minimum costs of all *cases* (j', v', l', s') for $i' = i + 1$. Now, let us extend the definition of $case$ to support double-row placement when double-height cells span the two rows, row 0 and row 1. We define $CASE$ $(v, l, j_0, s_0, j_1, s_1)$ given I, where I is the number of placed *double-height* cells. Subscripts 0 and 1 refer to row 0 and row 1, respectively. In Algorithm 3, we obtain the last placed cell c_k from i and j. Here, in double-row optimization, we know exactly the last placed double-height cell because of **Assumption 2**, and we would like to obtain i_0 and i_1 (number of cells placed in row 0 and row 1, respectively) since the formulation allows reordering between a single-height cell and a double-height cell, i.e., a single-height cell may be relocated to the left of the double-height cell even if that single-height cell was originally to the right of the double-height cell. Given the double-height cell's initial position k_0 in row 0 and k_1 in row 1, we have $i_0 = k_0 - j_0$ and $i_1 = k_1 - j_1$. The values of v, l, s_0 and s_1 can be obtained directly from *CASE*.

We give a precise description of our double-row dynamic programming in Algorithm 5. Line 1 obtains the double-height cell sequence from the initial (i.e., input) two-row placement. We note that two

Algorithm 6 $Cost$ (double-row)

1: **Inputs:** $I, v, l, j_0, s_0, j_0, s_0, I', v', l', j'_0, s'_0, j'_1, s'_1$
2: $k_0 \leftarrow getK(I, 0)$, $k_1 \leftarrow getK(I, 1)$
3: $k'_0 \leftarrow getK(I', 0)$, $k'_1 \leftarrow getK(I', 1)$
4: $i_0 \leftarrow k_0 + j_0$, $i_1 \leftarrow k_1 + j_1$
5: $i'_0 \leftarrow k'_0 + j'_0$, $i'_1 \leftarrow k'_1 + j'_1$
6: $d_0 \leftarrow optSR_0(^{i'_0,j'_0,v',l',s'_0}_{i_0,j_0,v,l,s_0})$
7: $d_1 \leftarrow optSR_1(^{i'_1,j'_1,v',l',s'_1}_{i_1,j_1,v,l,s_1})$
8: $totCost \leftarrow d_0 + d_1$
9: **Return** $totCost$

virtual double-height cells are added to "pad" the input at the start and at the end of the placement rows, respectively. Lines 2 – 3 initialize the DP solution array. The array only has entries for double-height cells, and records all solutions (costs) $D[I]$ when we have placed the I^{th} double-height cell. Lines 4 – 12 are the heart of the algorithm. Starting with the (left) virtual double-height cell, the algorithm incrementally places double-height cells and updates minimum costs from all *CASES* in $D[I]$ to all *CASES* in $D[I + 1]$ assuming we have placed I double-height cells. In Lines 13 – 15, we obtain the minimum cost among all legal *CASES* when we reach the ending (right) virtual cell ($I = |DHCellList|$), and Line 16 returns the minimum cost for the two rows.

Algorithm 6 describes the cost function in our double-row DP. Line 2 retrieves the double-height cell position in the initial placement for each of the rows. Line 3 gets the next double-height cell similarly. Line 4 obtains the numbers of cells (i_0 and i_1) that have been placed for the two rows. And, Line 5 obtains the numbers of cells (i'_0 and i'_1) that we must place by the time we reach the next double-height cell. For example, for row 0, we need to place cells starting from the *case* (j_0, v, l, s_0) with i_0, until we reach the *case* (j'_0, v', l', s'_0) with i'_0. The above can be achieved by $optSR$ – a modified version of the single-row dynamic programming. In $optSR$, we make sure that we do not place any double-height cells other than $c_{i'_0}$. Thus, Assumption 2 is maintained. In Lines 8 and 9, we return the two-row sum of costs.

We highlight the fact that in our implementation, given the starting *case* (j, v, l, s) with i, $optSR$ calculates all minimum costs of *case* (j', v', l', s') with i', where $k' = i' + j'$, within one functional call to our single-row DP. With this, the number of calls to single-row DP is proportional only to #*cases*, rather than to #*CASES*.

3.1.4 Multi-Row Optimization

In this section, we generalize from the single-row dynamic programming, and describe our approach for *multi-row detailed placement*, with support of fully-reorderable multi-height cells and inter-row cell relocating.

Multi-Row Optimization Problem. *Given an initial legalized multi-row placement, perturb the placement across the multiple rows to minimize inter-cell diffusion* steps.

Inputs: Legalized multi-row placement, available cell variants, and yield cost function.

Output: Optimized multi-row detailed placement with minimized overall cost (including inter-cell diffusion *steps*).

Constraints: Maximum horizontal displacement range, maximum vertical displacement range, maximum reordering range and availability of cell flipping.

Preliminaries

Similar to double-row optimization, we optimize m consecutive rows together (as a single optimization *window*) in multi-row optimization. In an optimization window, we move the cells according to our algorithm assuming that cells outside the window are fixed. Different windows are optimized separately. However, compared to the double-row optimization in Section 3.1.3, we do not require the relative positions among double-height cells to be fixed. Instead, a double-height cell can be reordered with another double-height cell as long as they are within the reordering range. Moreover, in contrast to Section 3.1.3's double-row optimization, where a cell cannot move outside its original cell row, here we allow a cell to move freely within a given vertical displacement range (in units of placement rows), enabling a larger solution space to minimize diffusion *steps*.

In single-row and double-row optimization, where only intra-row relocating and reordering are allowed, the initial cell ordering (c_k in Table 3.2) is defined within each row from the initial (input) placement. To enable a unified multi-row reordering range, with support of inter-row relocating and reordering, we redefine the original cell ordering as follows:

Definition. *Given an m-row initial (input) placement, cells in all m rows are left-to-right ordered according to their rightmost boundary, in a unified one-dimensional array, e.g., c_1, c_2, ..., c_k. If cells in the initial placement have the same x coordinate for their right boundary, we break ties using the y coordinate of their lower boundary.*

Figure 3.6 shows an example of sequential cell ordering for a two-row initial placement. We note that cells c_4 and c_5 could have their positions exchanged in the ordering, regardless of their left boundary. However, as mentioned, in our implementation tie-breaking is by descending order of y coordinate.

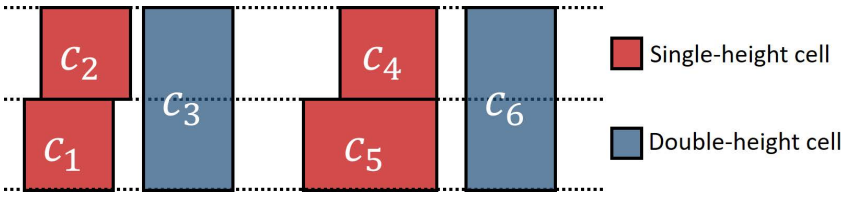


Figure 3.6: An example of multi-row cell ordering. Cells are sequentially ordered (c_1 to c_6) according to the x coordinate of their right boundary. Cells c_4 and c_5 have the same right boundary x coordinate, and thus could be switched in the ordering.

With the above redefined cell ordering, reordering range works the same way as in Section 3.1.2. The new sequentially ordered position is determined by the new x coordinate (in the final solution) of the right boundary of each cell. The difference between the original and the new sequentially ordered position should be always within the reordering range. In the multi-row optimization, given the above redefined cell ordering, our dynamic programming still seeks to place one cell at a time, from left to right. The left-to-right placement procedure then induces the following assumption:

Assumption. *The x coordinate of the right boundary of the $(i+1)^{st}$ cell must be greater than or equal to the right boundary of the partial placement consisting of i cells (i.e., placement boundary).*

Given the definition, the assumption does not reduce the solution space. For example, in Figure 3.7, assuming a partial placement of c_2 and c_1, if the 3^{rd} cell to be placed is c_3, and we would like its right boundary to be to the left of the placement boundary, then we can always get to such a partial placement solution from a partial placement of c_2 and c_3, followed by placement of c_1.

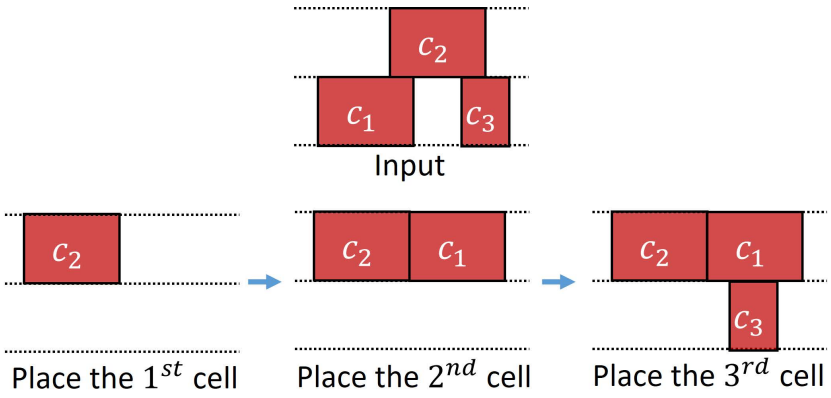


Figure 3.7: Illustration of the **Assumption**.

Formulation

Given the above assumption, our approach will find an optimal placement solution for a given optimization window of m rows containing multi-height cells. We illustrate the multi-row dynamic programming-based detailed placement in Figure 3.8(a). We use type array $T = \{t_0, ..., t_{m-1}\}$ to describe the type, i.e., 2-fin, 3-fin or 4-fin configuration, of the rightmost cell in each row. Initially, each entry of T is an initial virtual cell, indicating that the placement boundary for all rows is the left boundary of the die, and that there will be no diffusion *step* penalty applied to any type of cell immediately to the right of this boundary. We also use distance array $D = \{d_0, ..., d_{m-1}\}$ to describe the shape of the placeable region as shown in Figure 3.8(b). The subproblems solved in the DP are of form: place $|C| - i$ cells in the placeable region defined by a partial placement with i cells.

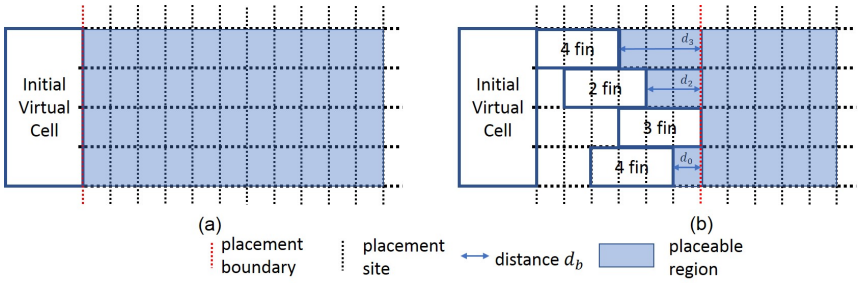


Figure 3.8: Illustration of DP in multi-row placement with $m = 4$.

We give a precise description of our multi-row dynamic programming in Algorithm 7. Note that the numbers of entries of distance array D and cell type array T are both $m - 1$ because the distance

from the last placed cell to the placement boundary is always zero, and the cell type of the last placed cell can be retrieved by cell variant v. Lines 1 – 3 initialize the DP solution array. Lines 4 – 15 describe the main algorithm. Compared to single-row dynamic programming, we have one more iteration over all placement rows in an optimization window, subject to the maximum vertical displacement range constraint. Effectively, the multi-row DP array is different from single-row DP array in that it is capable of storing multiple intermediate placement solutions given the same cell ordering and horizontal displacement, as long as these solutions have different type (T) or distance (D) arrays. Also, Line 11 updates distance array D and cell type array T according to the choice of placement row b'. Lines 16 – 19 obtain the optimal solution among all legal cases when $i = |C|$, and Line 20 returns the optimal solution for the current optimization window.

Algorithm 7 Dynamic programming (multi-row)

1: **Initialize costs for all legal cases** (j, v, l, b, D, T, s)
2: $d[0][j][v][l][b][D][T][s] \leftarrow 0$,
3: $d[i][j][v][l][b][D][T][s] \leftarrow +\infty, (0 < i \leq |C|)$
4: **for all** $i = 0$ to $|C| - 1$ **do**
5: **for all** $d[i][j][v][l][b][D][T][s] \neq \infty$ **do**
6: **for all** $(j', s') \in$ getNext(s) **do**
7: **for all** (v', l', b') **do**
8: $i' = i + 1$
9: $t \leftarrow d[i][j][v][l][b][D][T][s] + cost(^{j',v',l',b'}_{i,j,v,l,b,D,T})$
10: $d[i'][j'][v'][l'][b'][D'][T'][s'] \leftarrow$ $\min\left(d[i'][j'][v'][l'][b'][D'][T'][s'], t\right)$
11: $Update(D, T)$
12: **end for**
13: **end for**
14: **end for**
15: **end for**
16: $finalCost \leftarrow \infty$
17: **for all** (j, v, l, b, D, T, s) **when** $i = |C|$ **do**
18: $finalCost \leftarrow \min\left(d[|C|][j][v][l][b][D][T][s], finalCost\right)$
19: **end for**
20: **Return** $finalCost$

Multi-row optimization is not capable of being aware of HPWL change in y direction and across different optimization windows. Therefore, to prevent HPWL degradation, we add additional displacement costs if a cell is moved out of the original HPWL bounding box, with penalty coefficient $\gamma_{penalty}$, as

shown in Equation (3.4).[36] The term $cost_{hpwl}$ is calculated as the distance between the current cell and the original HPWL bounding box, in units of placement sites.

$$cost = cost_{step} + \alpha \cdot cost_{disp} + \gamma_{penalty} \cdot cost_{hpwl} + \alpha \cdot \beta \cdot cost_{var} \tag{3.4}$$

3.1.5 Experiments

We implement our dynamic programming in C++ with OpenAccess 2.2.43 [128] to support LEF/DEF [119], and with OpenMP [125] to enable thread-level parallelism. We perform experiments in an N7 FinFET technology with multi-height triple-V_{th} libraries from a leading technology consortium. The fin height information is not disclosed in our enablement. Therefore, following guidance from [75], we randomly assign fin heights (2, 3, or 4 fins) to each cell with 1:3:6 ratio for 2, 3 and 4 fins, respectively, as our default fin height assignment methodology to match industrial designs at advanced nodes. For example, a double-height cell will have four random fin heights, i.e., for its left and right boundaries on the first row, and its left and right boundaries on the second row. Section 3.1.5 further discusses the impact of alternative fin height assignment methods.

We generate the bimodal leakage values from the NDE-oblivious standard-cell Liberty file as follows [75]. Since NDE only affects the boundary transistors for each cell, given a leakage value of each standard cell from the Liberty file, we first approximate the boundary transistor leakage value by dividing the state-independent cell leakage by the cell width (in units of contacted-poly pitch), e.g., if a cell (width = 3) has a leakage value of three, then the boundary transistors have a leakage value of one. Then, for each diffusion *step*, 52% of boundary transistor leakage value is added to the cell leakage. In the above example, the cell has a new leakage value of 3.52 (resp. 4.04) when there exists one *step* (resp. two *steps*).

We apply our detailed placement optimization to an Arm Cortex-M0 core (*M0*) and four design blocks (*AES*, *JPEG*, *VGA* and *MPEG*) from OpenCores [124]. Design information is summarized in Table 3.3. We synthesize designs using *Synopsys Design Compiler L-2016.03-SP4* [129], and perform

[36] We pre-calculate all net bounding boxes (one-time effort) and only apply the HPWL penalty if a cell is placed outside of its nets' bounding boxes.

place-and-route using *Cadence Innovus Implementation System v15.2* [112]. We also apply our detailed placement optimization to winning solutions from the ICCAD-2017 multi-deck standard cell legalization contest [11]. All experiments are performed with 8 threads on a $2.6GHz$ Intel Xeon server.

In the following, we show (i) the scalability and sensitivity, i.e., impact of cell displacement range x_Δ, reordering range r, enabling of cell flipping f, and #rows per window m for the multi-row implementation on runtime and quality of results (QoR in terms of *step* reduction); (ii) impact of the weighting factors, i.e., weighting factor α for cell displacement, weighting factor β for cell flipping, and weighting factor γ for HPWL on QoR; (iii) metaheuristics by combining single-row HPWL-aware and multi-row optimization; (iv) our main results with single-row, double-row and multi-row optimization for five design blocks and three fin height assignment methodologies; (v) performance improvement using *intentional steps*; and (vi) our results with multi-row optimization for ICCAD-2017 benchmarks [11].

Table 3.3: Design information.

Design	#Inst	Clock period
AES	$\sim$12K	500ps
M0	$\sim$10K	500ps
JPEG	$\sim$54K	500ps
VGA	$\sim$69K	500ps
MPEG	$\sim$14K	500ps

Scalability/Sensitivity Study

In this subsection, we compare the impact of reordering range and displacement range on the single-row (SR), double-row (DR) and multi-row (MR) optimization. By default, we use $m = 2$ in MR optimization (see Figure 3.10 and discussion below). Following results of [36], cell flipping is enabled by default for maximum *step* reduction.

To assess the scalability of our approach, we sweep (x_Δ, r), i.e., maximum allowed cell displacement x_Δ (in placement sites) and maximum allowed one-sided reordering r, and study the impact on runtime. In this experiment, we sweep x_Δ from 0 to 15, and r from 0 to 2. A cell can freely move across 31 placement sites, and can have up to 5 different positions in a placement window, if we set $x_\Delta = 15$

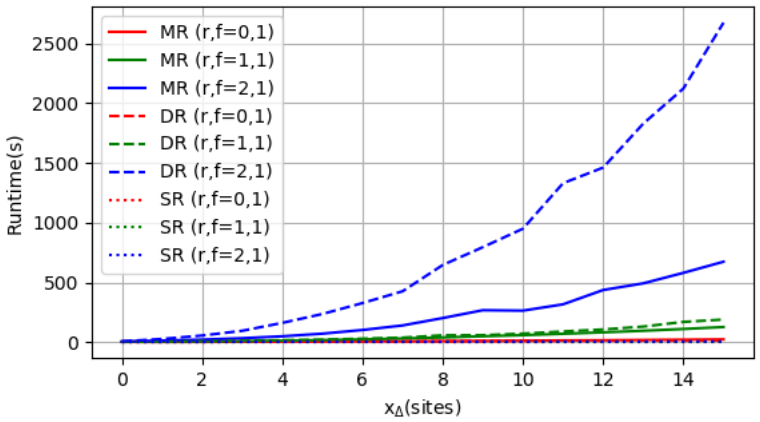


Figure 3.9: Sensitivity of runtime to (x_Δ, r, f) parameters.

and $r = 2$. We set $(\alpha, \beta) = (0, 0)$ as these parameters do not have any impact on the complexity of our formulation. We use design block *AES* for this study.[37]

Our study results are shown in Figure 3.9. We find that the runtime generally grows quadratically with the number of available placement sites per each cell. However, for cell reordering, there is a dramatic increase in runtime as r goes up, e.g., we observe $12\times$ runtime increase going from $r = 1$ to $r = 2$.

Also, compared to DR [36], our new MR implementation with $m = 2$ rows per window is much more efficient in terms of runtime. To investigate the impact of m (#rows in a window) in MR, we compare the sensitivity of #*steps* in Figure 3.10 for $m = 2$ and $m = 3$. Runs with $m = 4$ are not feasible due to much larger memory consumption. We find that $m = 2$ actually gives better #*steps* than $m = 3$ using our N7 library, because all multi-height cells have VSS power rails for their cell boundaries, such that all multi-height cells are aligned per two cell rows. Given the above observation, we use $m = 2$ for MR in all of the following experiments.

To assess the sensitivity to (x_Δ, r), Figures 3.11, 3.12 and 3.13 show #diffusion *steps*, HPWL and RWL respectively, as we sweep (x_Δ, r). Since our algorithm only optimizes #diffusion *steps* when $(\alpha, \beta) = (0, 0)$, here we see HPWL and RWL that correspond to a best-case (minimized) #*steps* normalized to initial design.

[37] To investigate the stability of our sensitivity studies and observations, we also use (i) an alternative *AES* design implementation with slightly different layout, and (ii) design block *M0*. Results for (i) and (ii) are consistent with the results that we report here.

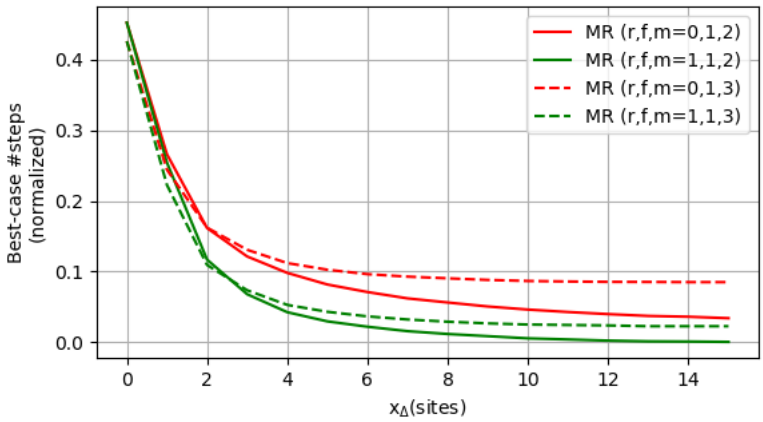


Figure 3.10: Sensitivity of #*steps* to m in MR optimization.

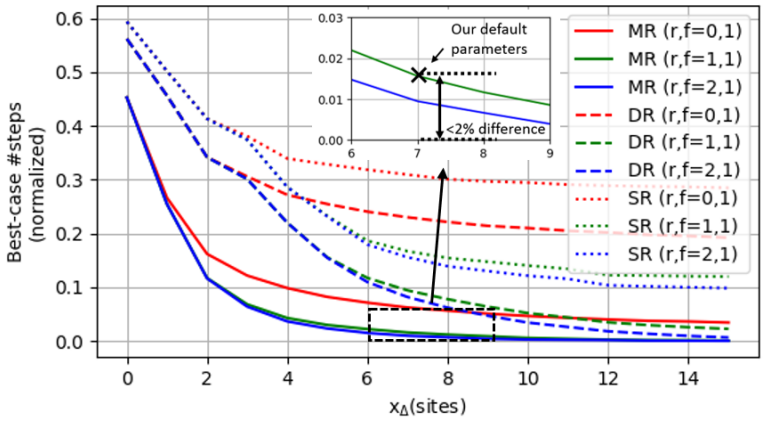


Figure 3.11: Sensitivity of #*steps* to (x_Δ, r, f) parameters.

We see from Figure 3.11 that SR can only reduce #*steps* by up to 80%, while DR and MR are able to reduce #*steps* by up to 99% given larger displacement range. Also, MR is consistently better than DR, especially given a smaller displacement range. Along with the runtime benefit of MR, we believe that the new MR implementation surpasses both the solution quality and the runtime efficiency of DR [36].

Moreover, for $f = 1$, there is only $\sim 0.6\%$ benefit of using $r = 2$ over $r = 1$, at the cost of $12\times$ the runtime; this suggests that $r \geq 2$ may not offer significant benefit in reducing #*steps*. In Figure 3.12 and Figure 3.13, HPWL and RWL increase linearly as x_Δ goes up. Based on these studies, to balance solution quality and runtime we apply $(x_\Delta, r) = (7, 1)$ in all of the following experiments.

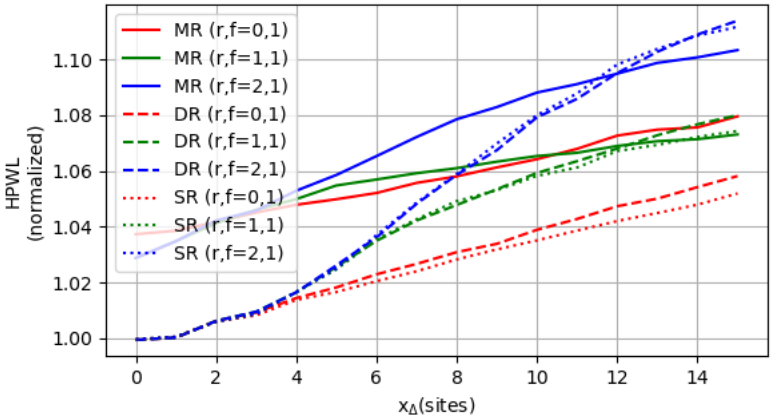


Figure 3.12: Sensitivity of HPWL to (x_Δ, r, f) parameters.

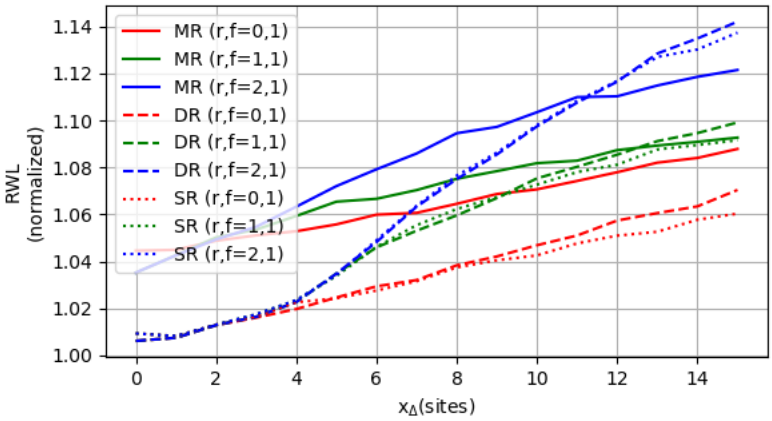


Figure 3.13: Sensitivity of RWL to (x_Δ, r, f) parameters.

Study of Weighting Factors

In the following subsection, our default flow is MR optimization, with two rows per window. We investigate impacts of the weighting factors $(\alpha, \gamma_{penalty})$ for cell displacement and HPWL penalty $(\gamma_{penalty})$ on HPWL and #*steps*. We sweep α and $\gamma_{penalty}$ from 0 to 1. We perform this experiment using design block *AES*. The results are shown in Figure 3.14. We can see that a non-zero displacement weight (α) and a non-zero HPWL penalty $(\gamma_{penalty})$ save HPWL while preserving most of the *step* reduction benefits. Therefore, we apply $\alpha = 0.01$ and $\gamma_{penalty} = 0.00001$ in all following experiments.

For the single-row optimization, we also study the impact of the HPWL weighting factor γ on HPWL and #*steps*. We sweep γ from 0.00001 to 1 with a step size of 10×. We perform this experiment using design block *AES*, with results shown in Figure 3.15. The tradeoff between HPWL and #*steps* is clear when γ is in the range of $[0.00001, 0.01]$. We use $\gamma = 0.0001$ for the HPWL-aware single-row optimization.

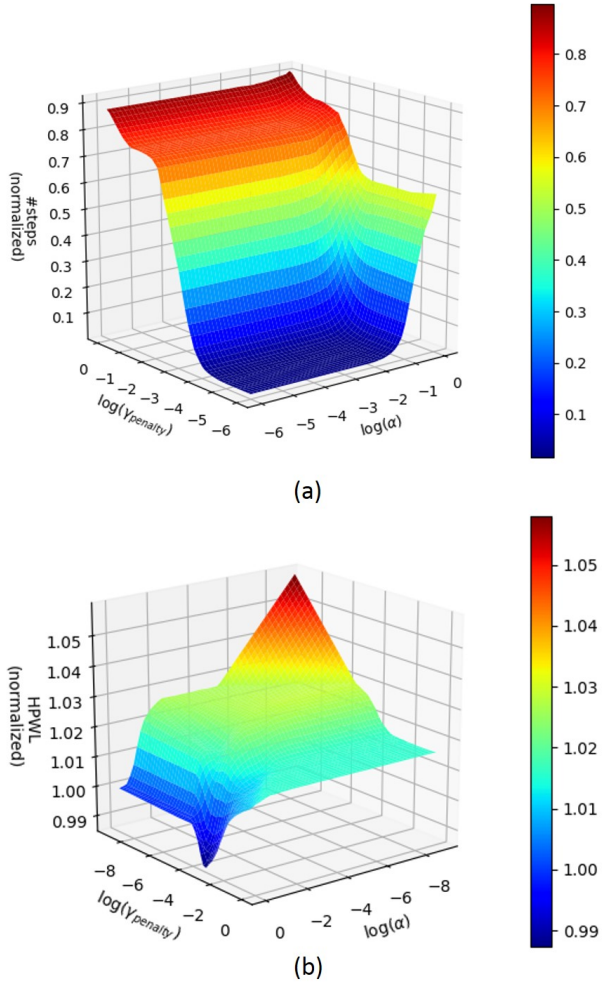


Figure 3.14: Impacts of weighting factors $(\alpha, \gamma_{penalty})$ on the tradeoff between HPWL and #*steps*.

Main Results

We apply our multi-row dynamic programming-based optimization to all our design blocks using the aforementioned parameter settings, i.e., $(x_\Delta, r, f) = (7, 1, 1)$ and $(\alpha, \beta) = (0.01, 1)$. Table 3.4 shows

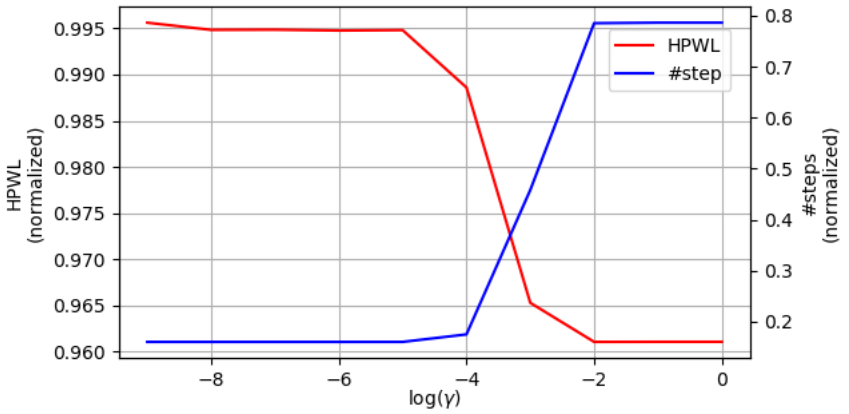


Figure 3.15: Impact of weighting factor γ on the tradeoff between HPWL and #*steps*.

the *step* reduction, runtime and estimated yield improvement for all five design blocks using multi-row optimization. We also report the impact on other metrics, i.e., routed wirelength (RWL), worst negative slack (WNS) and leakage power as reported by the place-and-route tool [112].

Table 3.4: Experimental results for all design blocks using multi-row optimization.

Design	Type	Fin Height Distribution			#*Steps*		RWL (μm)		WNS (ns)		Leakage (mW)		Runtime	Est. Yield
		2 fin%	3 fin%	4 fin%	Init	Final (Δ%)	Init	Final (Δ%)	Init	Final	Init	Final (Δ%)	(s)	Impr. %
AES	rand	10.0	30.4	59.6	7973	152 (-98.1%)	31873	32995 (+3.5%)	-0.013	-0.021	16.1	15.8 (-2.1%)	162.1	+0.71
	Vt	48.3	47.8	3.9	6816	143 (-97.9%)	31874	32944 (+3.4%)	-0.013	-0.020	16.6	15.8 (-4.9%)	81.5	+0.66
	drive	47.5	46.6	5.9	7215	236 (-96.7%)	31874	32888 (+3.2%)	-0.013	-0.018	16.1	15.8 (-2.0%)	109.9	+0.69
M0	rand	10.1	30.4	59.4	6588	243 (-96.3%)	27670	28728 (+3.8%)	-0.043	-0.070	18.9	18.6 (-1.9%)	174.4	+0.22
	Vt	49.3	48.6	2.1	5379	152 (-97.2%)	27674	28588 (+3.3%)	-0.043	-0.111	19.5	18.6 (-4.5%)	74.1	+0.52
	drive	46.3	45.6	8.0	6211	398 (-93.6%)	27669	28718 (+3.8%)	-0.043	-0.051	19.1	18.6 (-2.6%)	64.5	+0.58
JPEG	rand	10.0	30.0	60.0	34760	656 (-98.1%)	101000	107699 (+6.6%)	-0.319	-0.278	96.3	94.3 (-2.1%)	776.5	+3.50
	Vt	48.2	48.6	3.3	29452	387 (-98.7%)	100997	106972 (+5.9%)	-0.319	-0.274	98.8	94.4 (-4.4%)	403.2	+2.78
	drive	44.0	44.5	11.5	36173	1291 (-96.4%)	101003	108103 (+7.0%)	-0.323	-0.290	97.2	94.4 (-2.9%)	398.2	+3.30
VGA	rand	10.0	30.1	60.0	50766	6179 (-87.8%)	208155	217492 (+4.5%)	-0.137	-0.080	208.3	205.1 (-1.5%)	713.3	+4.56
	Vt	48.8	49.6	1.6	40743	3685 (-91.0%)	208155	216603 (+4.1%)	-0.137	-0.069	213.4	205.5 (-3.7%)	536.8	+3.48
	drive	42.1	42.8	15.1	57273	10871 (-81.0%)	208155	217664 (+4.6%)	-0.137	-0.129	208.2	205.1 (-1.5%)	491.1	+4.24
MPEG	rand	9.9	30.5	59.6	9994	1367 (-86.3%)	38896	40594 (+4.4%)	-0.005	-0.018	33.2	33.1 (-0.2%)	137.3	+0.87
	Vt	49.6	49.4	1.0	7824	753 (-90.4%)	38882	40383 (+3.9%)	-0.011	-0.026	33.2	33.1 (-0.3%)	68.6	+0.70
	drive	43.1	43.1	13.8	10931	2145 (-80.4%)	38901	40649 (+4.5%)	-0.005	-0.030	33.2	33.1 (-0.3%)	99.5	+0.86

We also investigate the impact of fin height assignment methodologies. We apply three methodologies – (i) *rand* randomly assigns fin heights according to probability ratio 1:3:6 for 2, 3, and 4 fins, respectively (see Section 3.1.5 above); (ii) *Vt* assigns fin heights according to their V_{th} property, with HVT (resp. NVT and LVT) cells having probability ratio 1:1:0 (resp. 1:1:1 and 0:1:1) for 2, 3, and 4 fins; (iii) *drive* assigns fin height according to their drive strength, with X0 (resp. X1 and others) cells having probability ratio 1:1:0 (resp. 1:1:1 and 0:1:1) for 2, 3, and 4 fins. The three methodologies generate different fin height distributions, and thus help confirm the robustness of our optimization in broader

scenarios. The results are shown in Table 3.4. For all designs with the default (*rand*) random fin height distribution, we achieve up to 98.1% reduction in #*steps* at the cost of around 3.5% RWL increase. The results also show that our optimization has negligible impact on WNS and that we can slightly improve the leakage. In addition, we perform a preliminary yield estimation assuming 2ppm failure rate for each *step*, and 1ppm failure rate after we remove the *step* (recall Footnote 32). Based on this assumption, we can see a yield improvement of up to 4.56% for a design block of 69K instances. We note that the yield improvement is expected to grow markedly with the die size. A larger design with many millions of instances may see more benefits.

For V_{th} and drive distribution, the results show similar *step* reduction percentage, demonstrating the robustness of our optimization. Figure 3.16 shows the layouts of placements before and after MR optimization.

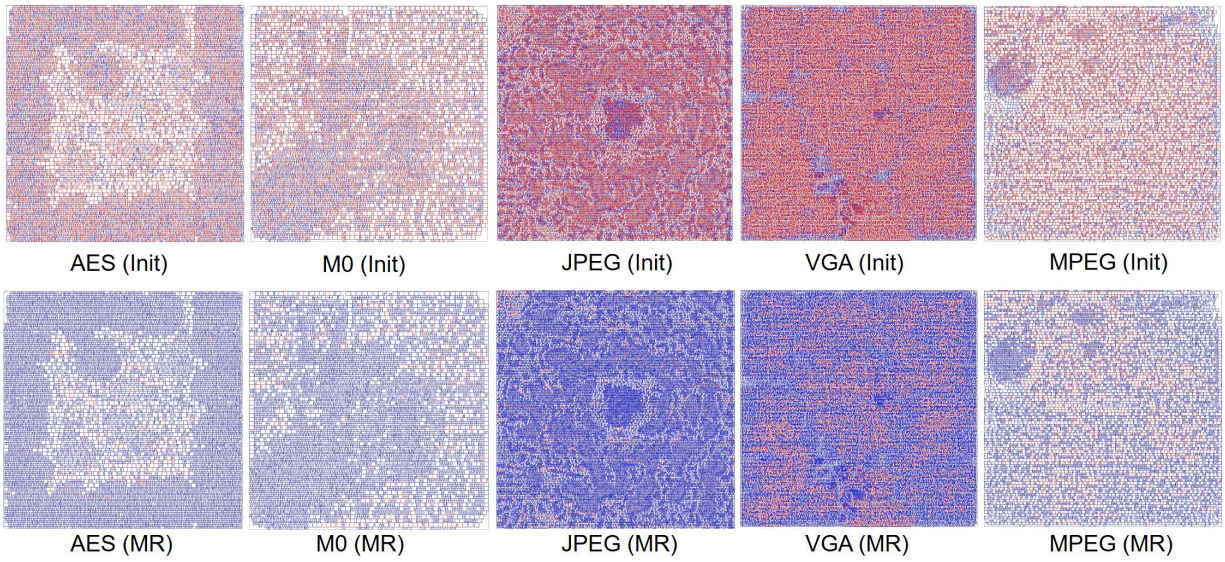


Figure 3.16: Layouts of placements before (Init) and after (MR) our MR optimization. Red color indicates cell instances with diffusion *steps* and blue color indicates cell instances without diffusion *steps*.

We also investigate the improvement achieved by our multi-row optimization over single-row, double-row optimization and previous works. We compare multi-row (MR) optimization to (i) single-row (SR) optimization (also to match [21][67]), (ii) ordered double-row (ODR) optimization (to match [66]), and (iii) double-row (DR) optimization. For (i), we use the proposed methodology in Section 3.1.2 and fix the locations of all multi-height cells. We note that our SR implementation is equivalent to [21][67],

supporting neighboring cell swapping and cell flipping with the adaptation of NDE. In SR, we use the same displacement range and reordering range as in DR, while using the default HPWL weighting factor $\gamma = 0.0001$ (HPWL weighting factor is not considered in the work of [36]). For (ii), we simply run our DR optimization with zero reordering range to achieve an ODR equivalent to [66]. For (iii), we use the proposed methodology in Section 3.1.3. The comparisons of #*steps*, routed wirelength (RWL) and runtime are shown in Tables 3.5, 3.6 and 3.7, respectively. For design blocks with fewer double-height cells, SR performance is competitive with that of ODR. However, for design blocks with more double-height cells, ODR is significantly better (up to 21% more *step* reduction) than SR due to movable double-height cells. The results show that DR effectively reduces the diffusion *steps* by around half compared to SR, and by around 40% compared to ODR. On average, DR has 11.6% more *step* reduction than ODR, and 17.7% more than SR, with respect to the initial number of diffusion *steps*. This suggests the importance of supporting movable and reorderable double-height cells, as there will be substantial benefits.

Table 3.5: Comparison of diffusion *steps* with SR (to match [21][67]), ODR (to match [66]) DR, MR and metaheuristics (Meta). DH% = % of double-height cells.

Design	DH%	Init	**SR (to match [21][67])**	**ODR (to match [66])**	**DR**	**MR**	**Meta**
AES	4.3%	7973	1395 (-82.5%)	1869 (-76.6%)	750 (-90.6%)	152 (-98.1%)	131 (-98.4%)
M0	8.4%	6588	1672 (-74.6%)	1742 (-73.6%)	842 (-87.2%)	243 (-96.3%)	179 (-97.3%)
JPEG	8.3%	34760	9731 (-72.0%)	8341 (-76.0%)	4555 (-86.9%)	656 (-98.1%)	473 (-98.6%)
VGA	24.8%	50766	27170 (-46.5%)	16405 (-67.7%)	11816 (-76.7%)	6179 (-87.8%)	5652 (-88.9%)
MPEG	23.0%	9994	5101 (-49.0%)	3444 (-65.5%)	2402 (-76.0%)	1367 (-86.3%)	1215 (-87.8%)
Avg.	–	-0.00%	-64.9%	-71.9%	-83.5%	-93.3%	-94.2%

Table 3.6: Comparison of routed wirelength (RWL) with SR, ODR, DR, MR and metaheuristics (Meta).

Design	Init	**SR**	**ODR**	**DR**	**MR**	**Meta**
AES	31873	32517 (+2.02%)	32637 (+2.40%)	32898 (+3.22%)	32995 (+3.52%)	33065 (+3.74%)
M0	27670	28201 (+1.92%)	28271 (+2.17%)	28470 (+2.89%)	28728 (+3.82%)	28805 (+4.10%)
JPEG	101000	104562 (+3.53%)	104657 (+3.62%)	105550 (+4.50%)	107699 (+6.63%)	108173 (+7.10%)
VGA	208155	212186 (+1.94%)	212905 (+2.28%)	214169 (+2.89%)	217492 (+4.49%)	216856 (+4.18%)
MPEG	38896	39640 (+1.91%)	39799 (+2.32%)	39950 (+2.71%)	40594 (+4.37%)	40512 (+4.15%)
Avg.	+0.00%	+2.26%	+2.56%	+3.24%	+4.57%	+4.66%

Table 3.7: Comparison of runtime (seconds) with SR, ODR, DR, MR and metaheuristics (Meta).

Design	SR	ODR	DR	MR	Meta
AES	32	8	59	162	348
M0	22	8	51	174	214
JPEG	325	50	344	776	2153
VGA	493	51	386	713	1658
MPEG	30	11	86	137	234

Metaheuristics

We have also explored several metaheuristics to assess (i) the *step* reduction achievable by invoking multiple optimization iterations, as well as (ii) potential improved tradeoffs between *step* reduction and degradation from initial placement (in terms of HPWL). First, we investigate the maximum *step* reduction versus the number of iterations. To explore the maximum benefits of *step* reduction, we invoke the multi-row optimization several times. Since the multi-row optimization is for every two rows, e.g., row 1 and 2 in a window, row 3 and 4 in the next window, etc., we can shift the window by one row and run again if we can further improve the solution quality. In our experiments, we alternatively align/unalign the optimization window with double-height cells, with aligned window in the first iteration to encourage the movement of double-height cells. We show the normalized number of *diffusion steps* and HPWL versus the number of optimization iterations (up to 8) in Figure 3.17. Compared to one iteration, the second iteration removes 45 out of 152 remaining steps after the first iteration, while the remaining six iterations only reduce 13 more *steps*, at the cost of increased HPWL.

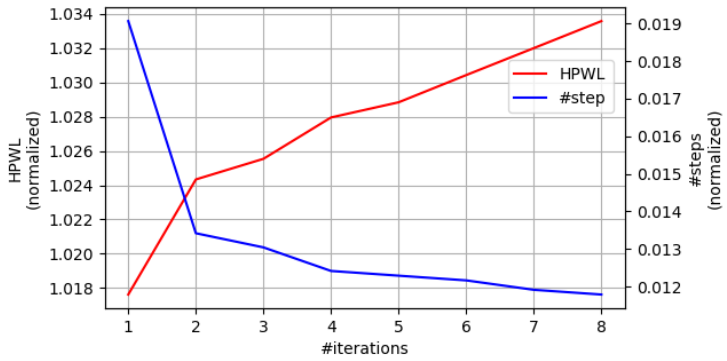


Figure 3.17: *#steps* (normalized) and HPWL (normalized) vs. #iterations in metaheuristic optimization.

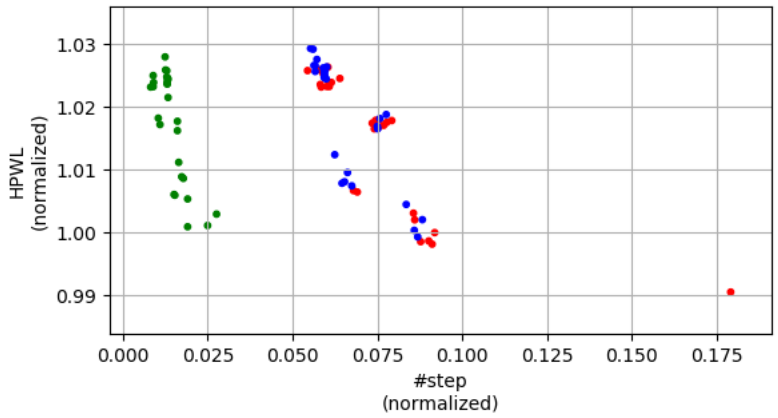


Figure 3.18: #steps vs. HPWL in metaheuristic optimization. Red (resp. green and blue) dots represent metaheuristic iterations that start with configuration A (resp. configuration B and configuration C).

Given the above observation, we seek to obtain a better tradeoff between *step* reduction and HPWL. Since our multi-row optimization is not HPWL-aware, we propose to invoke both single-row and multi-row optimization with a total "budget" of four iterations, to find the best four-iteration sequence. We explore all possible optimization sequences comprised of the following three configurations – (A) single-row HPWL-aware; (B) multi-row aligned with double-height cells; and (C) multi-row unaligned with double-height cells. We report the optimized number of *steps*, along with HPWL, in Figure 3.18. We can see that the configuration for the first iteration largely determines the optimized number of *steps*. The first iteration should be (B) to obtain better *step* reduction. Also, the optimization should finish with (A) for better HPWL. We report the metaheuristic results in Tables 3.5, 3.6 and 3.7.

Performance Improvement Using Intentional Steps

Similar in spirit to [51], we explore the possibility of improving design performance with *intentional steps* – i.e., using filler cells that create an *intentional step* to the neighboring timing-critical functional cell so as to improve the timing of that functional cell.[38] In the cost function, we use a third

[38] An *intentional inter-cell step* may increase/decrease the drive strength of the function cell. E.g., a *step* adjacent to a PFET may decrease the drive strength while a *step* adjacent to an NFET may increase the drive strength. Here, instead of using a filler cell to match diffusion heights for both the NFET and the PFET of the function cell (to reduce #*steps*), we create a filler-induced *intentional step* by matching the diffusion height for only the PFET, thus increasing the drive strength for the NFET. We note that exact timing and power impacts and tradeoffs will vary with STI processes.

weighting factor δ to represent the benefit of an *intentional step* to a timing-critical cell. We sweep δ from 0 to -2 with a step size of -0.1. We select 5% of all cells as timing-critical cells and perform optimization using all design blocks. The results are shown in Figure 3.19. We use $orig.opt$ to represent the results with $\delta = 0$, and $time.opt$ to represent the results with $\delta = -0.3$. Compared to $\delta = 0$, we achieve up to 5× increase in *#filler-induced steps* incident to timing-critical cells when $\delta = -0.3$, at the cost of slightly increased *#non-filler-induced steps* to non-timing-critical cells. This translates to up to 2.13 *steps* per timing-critical cell after $time.opt$, compared to 0.42 *steps* after $orig.opt$. Overall, we can still decrease total *steps* by more than 70%, showing the effectiveness of our algorithm. We note that as we add more *intentional steps* to timing-critical cells, we leave a smaller solution space for non-timing-critical cells. Thus, $time.opt$ generates more *steps* to non-timing-critical cells. We furthermore observe that as δ decreases, the *#intentional steps* that we can achieve approaches a limit, as shown in Figure 3.20. This may help set expectations for benefits that might be derived from a more comprehensive, timing-aware flow (which we leave for future work).

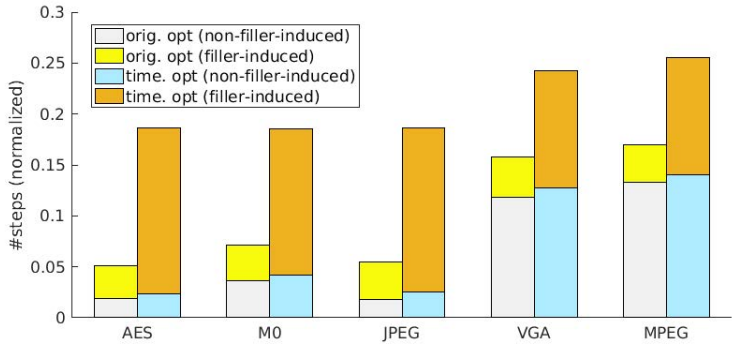


Figure 3.19: Comparison of #filler-induced *steps* and total *#steps* for all design blocks before ($orig.opt$, $\delta = 0$) and after ($time.opt$, $\delta = -0.3$) using *intentional steps*.

ICCAD-2017 Benchmark Results

We apply our multi-row dynamic programming-based optimization to winning solutions from the ICCAD-2017 contest [11] only considering row and site alignments, but not considering constraints, including maximum cell movement, cell edge spacing, pin access, pin shorts and fence regions from the contest. The input legalized placements for all benchmark testcases are from the first-place team's

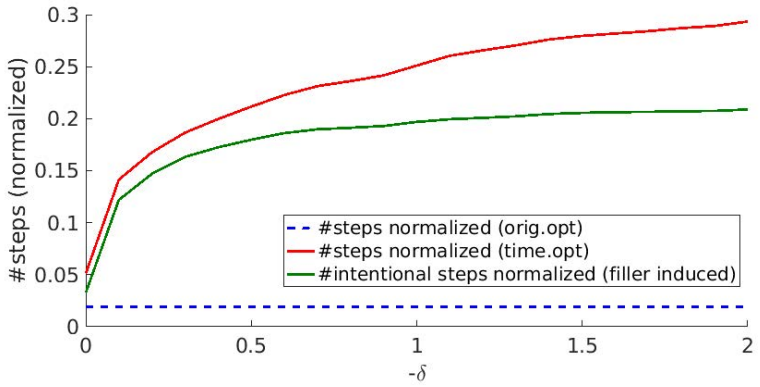


Figure 3.20: Sensitivity of filler-induced *steps* to δ. Testcase: *AES*.

solutions in ICCAD-2017 contest, except *pci_bridge32_a_md1* and *pci_bridge32_a_md2*, for which we use the second-place team's solutions (because the first-place team's solutions for these two testcases have cells placed outside of the die boundary). We keep the same P/G alignment as in the input placement. We apply *rand* fin height assignment methodology with the above-mentioned 1:3:6 ratio for 2, 3 and 4 fins, respectively. The results are shown in Table 3.8. For all ICCAD-2017 benchmark testcases, we achieve up to 96.8% reduction in #*steps*.

3.1.6 Conclusion

In this work, we present an *optimal* dynamic programming-based single-/double-row detailed placement methodology to minimize diffusion *steps* in sub-10nm VLSI, for improved yield and mitigation of NDE. Our work achieves several improvements as compared to previous works: (i) optimal dynamic programming with support of a richer set of cell movements, i.e., flipping, relocating and enhanced reordering; (ii) optimal double-row dynamic programming *with support of movable and reorderable double-height cells*; and (iii) a novel performance improvement technique using *intentional steps*. The proposed techniques achieve up to 98% reduction of inter-cell diffusion *steps*, with scalable runtime and high die utilization in an N7 node enablement.

Table 3.8: Design information and experiment results for ICCAD-2017 benchmark [11]. Distribution of single-height, double-height, triple-height and quadruple-height cells are shown in columns 1×H, 2×H, 3×H and 4×H, respectively.

Design	#Inst	Cell types %				*#Steps*			Runtime
		1×H	2×H	3×H	4×H	Init	Final	(Δ%)	(s)
des_perf_b_md1	∼11K	94.80	5.20	0.00	0.00	57806	3781	(-93.46%)	361.3
des_perf_b_md2	∼11K	90.47	6.02	2.01	1.50	70733	7494	(-89.41%)	232.8
edit_dist_1_md1	∼13K	90.31	6.12	2.04	1.53	74351	6019	(-91.90%)	420.9
edit_dist_a_md2	∼13K	90.31	6.12	2.04	1.53	76657	8074	(-89.47%)	417.8
fft_2_md2	∼ 3K	89.62	6.56	2.18	1.64	22040	3789	(-82.81%)	53.2
fft_a_md2	∼ 3K	89.57	6.59	2.19	1.65	10960	606	(-94.47%)	136.4
fft_a_md3	∼ 3K	93.42	2.19	2.19	2.19	11631	372	(-96.80%)	78.1
pci_bridge32_a_md1	∼ 3K	90.39	6.07	2.02	1.52	17284	1429	(-91.73%)	83.8
des_perf_1	∼11K	100.00	0.00	0.00	0.00	73202	3516	(-95.20%)	488.7
des_perf_a_md1	∼11K	95.66	4.34	0.00	0.00	64624	3060	(-95.26%)	307.3
des_perf_a_md2	∼11K	96.99	1.00	1.00	1.00	64346	4793	(-92.55%)	315.9
edit_dist_a_md3	∼13K	93.88	2.04	2.04	2.04	78560	11100	(-85.87%)	258.9
pci_bridge32_a_md2	∼ 3K	85.51	7.08	4.05	3.37	21435	6235	(-70.91%)	71.2
pci_bridge32_b_md1	∼ 3K	90.39	6.07	2.02	1.52	14988	1070	(-92.86%)	68.1
pci_bridge32_b_md2	∼ 3K	96.97	1.01	1.01	1.01	13812	488	(-96.47%)	135.0
pci_bridge32_b_md3	∼ 3K	94.94	1.01	2.02	2.02	14929	1193	(-92.01%)	84.2

3.2 Vertical M1 Routing-Aware Detailed Placement for Congestion and Wirelength Reduction in Sub-10nm Nodes

In tandem with aggressive pitch scaling in sub-$10nm$ technology nodes, the detailed routing problem has become extremely challenging. Routing today must deal with large numbers of complex design rules that are driven by patterning technologies – notably, self-aligned multiple patterning and line-end cut on minimum-pitch metal layers, as well as contact- and via-layer patterning. The quest to scale "PPAC" (power, performance, area, cost) has led to a very delicate balancing act among power delivery, routing resource, and resistivity in middle-of-line (MOL) and local metal layers.

To address these challenges, the industry has seen rapid innovation in standard-cell architecture starting at the foundry $10nm$ (N10) node, and accelerating into the N7/N5 enablement. As examples of cell architecture evolution, metal layers below M1 are used for internal routing within a standard cell, or horizontal M1 power/ground pins are removed to gain additional routing resources for inter-cell routing.

These new cell architectures, wherein *inter-row M1 routing is allowed*, force new consideration of vertical alignment of cells.

New Cell Architectures in Sub-10nm

Figure 3.21 illustrates inverter (INV) layout in three types of cell architectures: (a) conventional 12-track, (b) *ClosedM1* 7.5-track, and (c) *OpenM1* 7.5-track. The conventional 12-track INV has power/ground (VDD/VSS) in M1, which prevents use of vertical M1 routing for pin access. In other words, with the conventional cell architecture, pin access is available only with M2 routing. However, in sub-10nm nodes, where metal layers below M1 are used for internal cell routing, the M1 layer can be used for pin access as well as for routing with both the *ClosedM1* and *OpenM1* cell architectures.

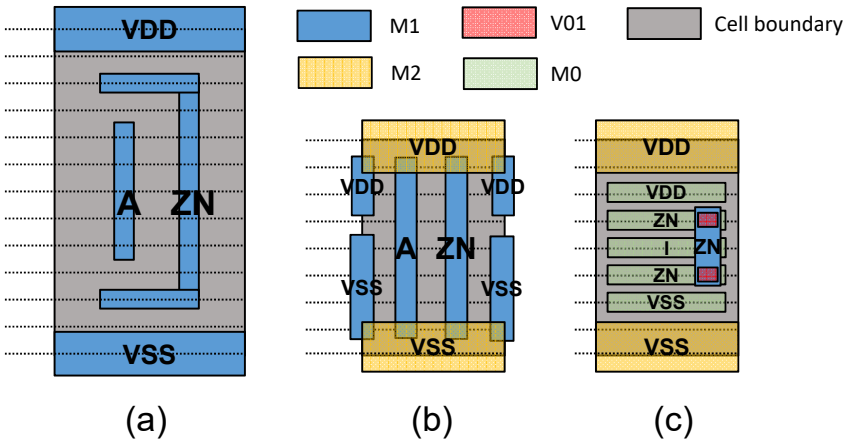


Figure 3.21: New cell architectures to gain additional routing resources. (a) Conventional 12-track INV; (b) *ClosedM1* 7.5-track INV; (c) *OpenM1* 7.5-track INV.

***ClosedM1* standard cell architecture.** A *ClosedM1* standard cell has 1D vertical M1 pins, including VDD/VSS pins, as shown in Figure 3.21(b). The M1 VDD/VSS pins at the left and right boundaries of the cell are connected to M2 VDD/VSS pins at the top and bottom boundaries by using via V12. In this way, VDD/VSS pins do not block inter-row M1 routing. Also, due to the design rules for self-aligned multiple patterning (SAMP), the M1 pins in *ClosedM1* have 1D shapes and are regularly placed with a fixed pitch. In particular, the *ClosedM1* cell library that we use in this work has M1 pitch equal to the width of a placement site. Therefore, if we vertically align pins of given net, these pins can be connected by a small M1 segment with negligible routing cost or overheads. Figure 3.22(a) illustrates

an example of direct vertical M1 routing (dM1) between two INVs. Here we define a direct vertical M1 routing as a (sub)net routing using only one M1 routing segment. Importantly, even though the *ClosedM1* cell architecture enables inter-row M1 routing, the realized power/performance/area (PPA) benefit from M1 routing may not be significant unless a router can effectively exploit the availability of direct vertical M1 routing. This is because M1 routing tracks are blocked by M1 pins, and the inter-row M1 routing can be used only when two pins are sufficiently aligned. Thus, *both* the detailed placer and the router must comprehend vertical alignment in order to maximally exploit direct vertical M1 routing for *ClosedM1*-based designs.

***OpenM1* standard cell architecture.** At sub-10nm nodes, the *OpenM1* standard cell architecture is introduced to enable more M1 routing resource than with the *ClosedM1* architecture. For *OpenM1* cells, M1 routing is "open" since most of the pins are on the M0 layer, which is a complementary layer below the M1 layer. As shown in Figure 3.21(c), the I, ZN, VDD, VSS pins have horizontal M0 segments, and an M1 segment connects two M0 segments for the ZN pin. We note that there is no connection between M0 and M2 segments for VDD/VSS pins. Thus, M1 routing for VDD/VSS pins must be accomplished with a special structure for the power distribution network.[39] In terms of signal routing, if two pins are overlapped horizontally (i.e., their projections onto the x-axis intersect), direct vertical M1 routing can be used to connect them. Figure 3.22(b) shows a direct vertical M1 routing between the ZN pin of the upper INV and the I pin of the lower INV. As long as the ZN and I pins are overlapped horizontally, the two pins can be connected using a single vertical M1 segment along with two V01 vias.

Compared to both the conventional and the *ClosedM1* cell architectures, *OpenM1* effectively enables an additional metal layer for routing, which can have considerable routability benefits. Furthermore, unlike with the sub-10nm *ClosedM1* architecture, conventional P&R tools can easily find benefits from *OpenM1* without any special optimization to maximize M1 routing. This being said, below we explore the question of whether there might still be room (beyond the current state of the art in commercial P&R tooling) to optimize for better pin accessibility in *OpenM1*-based designs, given that pins are horizontal. For instance, by maximizing "overlap" between pins in a net, we might induce a router to use more direct vertical M1 routing between pins, which would reduce usage (blockage) and detouring on upper layers

[39] For example, vertical M1 segments must be inserted with a fixed pitch to staple M2 and M0 VDD/VSS pins.

(M2, M3, etc.). In Section 3.2.4, we report experimental results with and without a detailed placement optimization that maximizes pin overlaps for *OpenM1*.

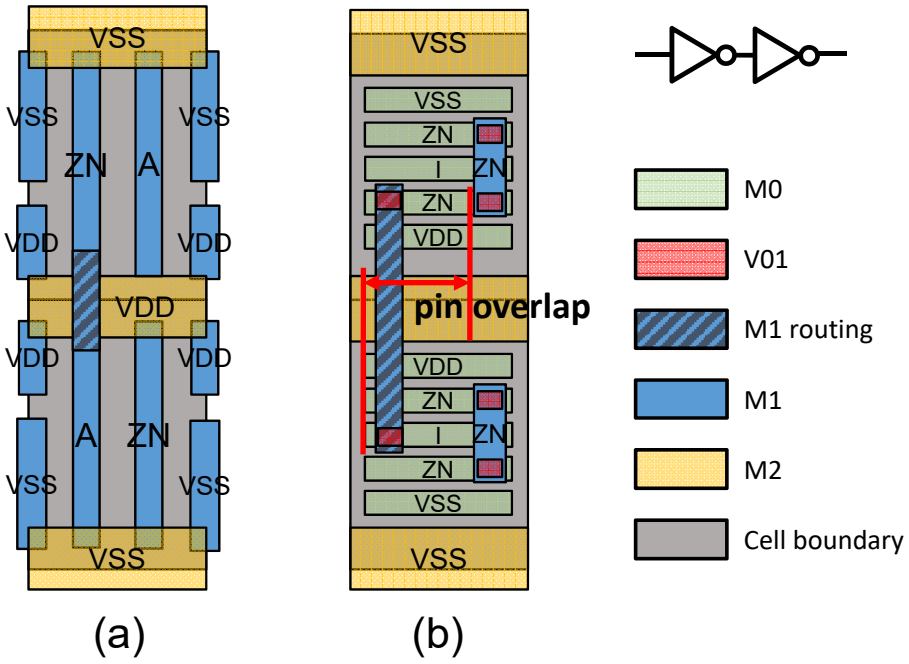


Figure 3.22: Direct vertical M1 routing examples: (a) *ClosedM1* and (b) *OpenM1*.

This Work

In this work, we propose a vertical M1 routing-aware detailed placement optimization based on mixed-integer linear programming (MILP) for two new sub-10nm cell architectures, i.e., *OpenM1* and *ClosedM1*. We note that the vertical M1 routing-aware detailed placement is a completely different problem from traditional wirelength-driven detailed placement, in the sense that the routing cost is non-monotonic due to vertical M1 routing, which is almost "free". Our MILP formulation enables exploration of the tradeoff between minimization of the traditional half-perimeter wirelength (HPWL) objective and maximization of the number of vertical pin alignments (= potential direct pin-pin routings using vertical M1) via a weighting factor (α). Below, we specifically study the impact of α on routed wirelength. The main contributions of our work are summarized as follows.[40]

[40]The MILP formulation will differ according to the standard cell template and layer directionality. However, our distributable optimization and exploration of metaheuristic configurations can apply with any technology.

- We propose an MILP-based detailed placement optimization for two cell architectures that are relevant in sub-$10nm$ process nodes, to consider and exploit (direct vertical) inter-row M1 routing.
- We propose a distributable window-based optimization to overcome the runtime limitation of the MILP-based approach.
- We implement our proposed approach in C++ with OpenAccess 2.2.43 [128] and incorporate it into a commercial tool-based placement and routing (P&R) flow. The results from our approach are evaluated using a commercial tool flow.
- We explore various metaheuristic configurations (optimization degrees of freedom, window size, iteration strategy, etc.) and study impacts on runtime and solution quality.

The remainder of this work is organized as follows. Section 3.2.1 reviews related previous works. In Section 3.2.2, we describe our MILP formulations for detailed placement optimization considering direct vertical M1 routing. In Section 3.2.3, we explain our overall optimization metaheuristic, centered around a distributable window-based optimization. Section 3.2.4 provides experimental results and analysis. We give conclusions in Section 3.2.5.

3.2.1 Related Work

We classify relevant previous works on detailed placement and placement legalization into three categories: (i) dynamic programming-based approaches, (ii) graph model-based approaches, and (iii) MILP-based approaches. Our present work is most closely related to the third category.

Dynamic programming-based approaches. Dynamic programming (DP) has been a popular framework, particularly for row-based detailed placement, for many years. Kahng et al. [53] propose an HPWL-driven ordered single-row detailed placement with free sites. Gupta et al. [33] propose a DP-based single-row placement optimization to enable sub-resolution assist feature insertion for improved manufacturability. Subsequent work addresses a 2D formulation [34], using DP in which vertical and horizontal costs are calculated with restricted perturbations. Hur and Lillis [47] propose a DP-based *optimal interleaving* for intra-row optimization in detailed placement. For double-patterning-aware

detailed placement, Gupta et al. [31] propose a DP-based algorithm that solves coloring conflicts while minimizing the displacement of timing-critical cells.

Graph-based approaches. A literature of graph model-based approaches typically formulates placement optimization as a shortest-path computation in an appropriate directed graph. Kahng et al. [50] legalize placement of a single row with various minimization objectives, by calculating a shortest path in a directed acyclic graph constructed from the input ordering of cells. The work of [106] proposes a triple-patterning-aware detailed placement using a graph model. The authors formulate a graph to determine cell locations as well as coloring solutions for a single row placement. Du and Wong [21] address the abutment of source and drain in FinFET-based cell placement. The authors propose a graph model that captures cell flipping and adjacent-cell swapping as underlying operations for detailed placement perturbation. A shortest-path algorithm then minimizes the cost induced by fixing the placement with respect to the source-drain abutment. Lin et al. [67] propose a graph-based detailed placement to resolve inter-row middle-of-line conflicts. Similar to [21], a graph is constructed to handle cell flipping, swapping and shifting operation for *local reordered single row refinement*.

MILP-based approaches. While DP-based and graph model-based approaches are efficient for single-row placement, it is not easy to handle multiple-row placement optimizations (specifically, in the context of this work, vertical M1 routing-aware placement) with these approaches due to interaction between vertically adjacent cells. However, several mixed integer-linear programming (MILP)-based approaches have been proposed which handle both single-row and multiple-row placement. Lin and Chu [63] formulate a MILP for triple-patterning-aware detailed placement. The MILP is used to assign a coloring solution for each standard cell and determine the location of each cell in a single row, while minimizing placement perturbation and coloring conflicts. Li and Koh [61] propose MILP-based detailed placement approaches using *single-cell-placement (SCP) variables*. The SCP variables correspond to locations, orientations as well as placement sites of each cell. The MILP determines the best SCP variable for each cell. The same authors' extension [62] supports mixed-size circuits and improves runtime by bounding solution spaces. Han et al. [39] adopt the MILP model of [61][62] and extend it to support N10-relevant design rules. Further, a distributable optimization is proposed based on partitioning of the layout into windows that can be independently legalized. In our present work, we use a similar strategy as

the work of [39], extending it to handle vertical M1 routing for new cell architectures in sub-10nm. Overall, our work is distinguished from previous (MILP-based) approaches in that (i) we formulate inter-row cell alignment to maximize direct vertical M1 routing, which has not been addressed in previous works, and (ii) we improve the distributable optimization of [39] by a smart selection of target windows along with a metaheuristic strategy.

3.2.2 MILP-based Optimization

In this section, we give our problem statement, followed by MILP formulations for vertical M1 routing-aware detailed placement optimization with two sub-10nm cell architectures, *ClosedM1* and *OpenM1*.

Vertical M1 Detailed Placement

Given: a post-routed placement, and per-cell placement perturbation range.

Perform: Perturb the input placement to optimize a weighted sum of (minimized) HPWL and (maximized) inter-row pin alignments, while satisfying cell location perturbation bounds and placement legality constraints.

MILP Formulation for ClosedM1

We formulate an MILP for our detailed placement problem for the *ClosedM1* cell architecture. In the following, we use notation as described in Table 3.9. For a given input layout, our objective is to minimize the weighted sum of HPWL of all nets subtracted by the total number of pin alignments for direct vertical M1 routing, while achieving a legal placement (no overlap of cells).

Minimize: $$-\alpha \cdot \sum d_{pq} + \sum_{n \in N} \beta_n \cdot w_n \tag{3.5}$$

Subject to:

$$w_n = x_{max,n} - x_{min,n} + y_{max,n} - y_{min,n}, \ \ \forall n \in N \tag{3.6}$$

$$x_{max,n} \geq x_c + x_p, \ \ x_{min,n} \leq x_c + x_p$$

$$y_{max,n} \geq y_c + y_p, \ \ y_{min,n} \leq y_c + x_p$$

$$\forall p \in P_n, \text{ where } c \text{ is the owner cell of pin } p \tag{3.7}$$

$$(x_c + x_p) - (x_{c'} + x_q) \leq G(1 - d_{pq})$$

$$(x_c + x_p) - (x_{c'} + x_q) \geq -G(1 - d_{pq})$$

$$(y_c + y_p) - (y_{c'} + y_q) \leq G(1 - d_{pq}) + H$$

$$(y_c + y_p) - (y_{c'} + y_q) \geq -G(1 - d_{pq}) - H$$

$$\forall (p, q) \text{ in } n, \text{ where } c, c' \text{ are owners of pins } p, q \tag{3.8}$$

$$\sum_{k \in K_c} \lambda_c^k = 1, \ \ \forall c \in C \tag{3.9}$$

$$f_c = \sum_{k \in K_c} f_c^k \lambda_c^k, \ \ \forall c \in C \tag{3.10}$$

$$x_c = \sum_{k \in K_c} x_c^k \lambda_c^k, \ \ y_c = \sum_{k \in K_c} y_c^k \lambda_c^k, \ \ \forall c \in C \tag{3.11}$$

$$s_{crq} = \sum_{k \in K_c} s_{crq}^k \lambda_c^k, \ \ \forall c \in C \tag{3.12}$$

$$\sum_{c \in C} s_{crq} \leq 1, \ \ \forall q \in Q, r \in R \tag{3.13}$$

HPWL calculation. Constraint (3.6) calculates the HPWL for each net n, where HPWL as usual corresponds to the half-perimeter of the minimum bounding box that contains all pins of n. The maximum and minimum x, y coordinates of pins of the net n are obtained by Constraint (3.7). The absolute coordinates of pin p are determined by adding the coordinates (x_c, y_c) of p's owner cell c to (x_p, y_p).

Table 3.9: Notations.

Notation	Meaning
d_{pq}	a binary indicator of whether pins p and q are aligned (*ClosedM1*) or overlapped (*OpenM1*)
w_n	half-perimeter wirelength (HPWL) of net n
α	a weighting factor for direct vertical M1 routing
β_n	a weighting factor for HPWL of net n
C, R, Q	sets of cells, rows, columns (placement sites)
N	set of nets
$x(y)_{min,n}$ $x(y)_{max,n}$	minimum x (y) and maximum x (y) coordinates of net n
P_n	set of pins in net n
G	a large positive constant number
H	placement row height
$x_c(y_c)$	x (y) coordinate of the center of cell c
$x_p(y_p)$	relative x (y) coordinate of pin p to its owner cell's x (y) coordinate
$x_{min,p}$ ($x_{max,p}$)	minimum (maximum) x coordinate of pin p relative to its owner cell's x coordinate
f_c	a binary indicator of whether cell c is flipped
s_{crq}	a binary indicator of whether cell c occupies site (r, q)
K_c	a set of candidates of cell c
λ_c^k	a binary indicator of whether candidate k for cell c is selected
$x_c^k(y_c^k)$	x (y) coordinate corresponding to λ_c^k
f_c^k	f_c corresponding to λ_c^k
s_{crq}^k	s_{crq} corresponding to λ_c^k
γ	maximum allowed length for a direct vertical M1 routing (unit: number of placement rows)
v_{pq}	a binary indicator of whether pins p and q are within a given range (γ) in y direction
o_{pq}	length of overlap between pins p and q
δ	minimum required overlap length for direct vertical M1 routing
ϵ	a weighting factor for the sum of overlap lengths (o_{pq})

Checking pin alignment. Constraint (3.8) checks whether pins p, q are aligned, by comparing their absolute coordinates. If the (absolute) x coordinates of p, q are not the same, $d_{pq} = 0$. Otherwise, the left side of the first and second constraints in Constraint (3.8) becomes zero, which makes $d_{pq} = 1$ allowed. In our implementation, we always ensure $d_{pq} = d_{qp}$.

Placement of each cell. Similar to [39], we assume that a perturbation range is given for each cell c, and that a cell cannot move beyond its given perturbation range. As in [39], we adopt the single-cell-placement (SCP) model of [62] to represent each candidate location and orientation for a cell. The binary variable λ_c^k represents a candidate k for a cell c, including the coordinates (x_c^k, y_c^k), the orientation (f_c^k), and whether placement site (r, q) is occupied (s_{crq}^k). These relations are handled by Constraints

(3.10), (3.11) and (3.12). Constraint (3.9) ensures that exactly one candidate is chosen for cell c among all $\lambda_c^k, k \in K_c$. Constraint (3.13) ensures a legal placement.

MILP Formulation for OpenM1

To maximize direct vertical M1 routing for the *OpenM1* cell architecture, we must maximize "overlap" between target pins, which is different from the objective for *ClosedM1*. In addition to maximizing the number of overlapping pin pairs, we also maximize the sum of overlap lengths of each pin-to-pin (sub)net so as to increase the probability that the router completes the direct vertical M1 routing. The *OpenM1* objective is:

$$\textbf{Minimize:} \quad -\alpha \cdot \sum d_{pq} - \epsilon \cdot \sum o_{pq} + \sum_{n \in N} \beta_n \cdot w_n \tag{3.14}$$

To support *OpenM1*, we slightly modify the previous MILP formulation for *ClosedM1* by introducing extra variables. In this case, d_{pq} becomes a binary indicator of whether pins p and q are "overlapped", and Constraint (3.8) is replaced with Constraints (3.15) – (3.17). Our notation is again as described in Table 3.9.

$$\begin{aligned} &a \geq x_c + x_{min,p}, \;\; a \geq x_{c'} + x_{min,q} \\ &b \leq x_c + x_{max,p}, \;\; b \leq x_{c'} + x_{max,q} \\ &\qquad \forall p, q, \text{ where } c, c' \text{ are the owner cells of pins } p, q \end{aligned} \tag{3.15}$$

$$\begin{aligned} &(y_c + y_p) - (y_{c'} + y_q) \leq G \cdot v_{pq} + \gamma \cdot H \\ &(y_c + y_p) - (y_{c'} + y_q) \geq -G \cdot v_{pq} - \gamma \cdot H \\ &\qquad \forall p, q, \text{ where } c, c' \text{ are the owner cells of pins } p, q \end{aligned} \tag{3.16}$$

$$a \geq x_c + x_{min,p}, \;\; a \geq x_{c'} + x_{min,q}$$

$$o_{pq} \leq b - a - \delta + G(1 - d_{pq}), \;\; o_{pq} \leq G \cdot d_{pq}$$

$$o_{pq} \geq -G(1 - d_{pq})$$

$$\forall (p, q) \text{ pin pairs in net } n, \forall n \tag{3.17}$$

$$d_{pq} + v_{pq} \leq 1, \;\; \forall p, q \tag{3.18}$$

Checking pin overlaps. Constraint (3.15) calculates the length of overlap in x direction between pins p and q. It first identifies the left side (a) and the right side (b) of the overlap between pins p and q. The overlap length o_{pq} is determined by a and b in Constraint (3.17). Constraint (3.16) checks whether the absolute difference of y coordinates of pins p and q is *larger* than γH and, if so, forces $v_{pq} = 1$. γ is a user-defined value for the maximum allowed vertical span of a direct vertical M1 routing.[41]

We use $\gamma = 3$, which means that a direct vertical M1 routing can cross three placement rows. For the case $v_{pq} = 1$, we do not need to make overlaps in the x direction since pins are multiple rows apart vertically; in such cases, it is difficult (i.e., highly improbable) to make a direct vertical M1 routing across multiple rows. Thus, Constraint (3.18) forces $d_{pq} = 0$ if $v_{pq} = 1$ so that the optimization does not make unnecessary overlaps. Constraint (3.17) forces $d_{pq} = 1$ if $b - a$ is larger than a predefined δ, which is the minimum required overlap length. Then, the o_{pq} is bounded by $b - a - \delta$. Otherwise, o_{pq} is bounded by zero.

3.2.3 Overall Flow

We now describe the overall flow of our optimization.

Distributable Optimization

In practice, the most critical limitation of the MILP-based approach is runtime. To overcome the runtime limitation, we adopt the distributable optimization proposed in [39].

[41] For example, $\gamma = 1$ means that direct vertical M1 routing can traverse only between two adjacent cell rows, and $\gamma = 2$ (resp. 3) means that direct vertical M1 routing can go through at most one (resp. two) intervening cell row(s).

We partition the layout into small windows, each with width b_w, and height b_h, and optimize these windows in several iterations. In each iteration, we select a subset of windows that are independently optimizable, and optimize them in parallel. More specifically, we select windows that do not have any horizontal or vertical overlap (i.e., have disjoint projections onto the x-axis and onto the y-axis). For example, as shown in Figure 3.23, windows that are diagonally adjacent can be selected and optimized in parallel. This is because a given window's optimization is unaware of cell displacements concurrently being made outside of the window; if windows share projections onto the x- or y-axis, the impact of solutions on HPWL from each window cannot be accurately captured.

Figure 3.24 illustrates two example cases of (a) target windows with intersecting projections (on the y-axis) and (b) target windows with disjoint projections. Since the target windows are optimized in parallel, the optimizer calculates $\Delta HPWL1$ for the displacement of p in $w1$ without knowing pin q's displacement, and vice versa ($\Delta HPWL2$ for q in $w2$). However, according to the final locations of p and q, the pins that determine the bounding box corresponding to HPWL can change, as shown in the figure. In the (a) case, this results in a discrepancy between the total $\Delta HPWL$ and the sum of $\Delta HPWL$ from each window. In the (b) case, since p and q always determine the top-left point and the bottom-right point of the bounding box, the sum of $\Delta HPWL$ from each window is equal to the total $\Delta HPWL$.

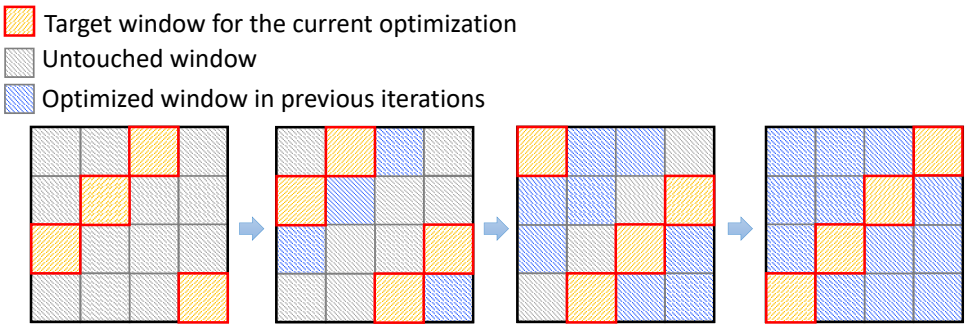


Figure 3.23: Illustration of distributable optimization.

Overall Flow

Algorithm 8 ($VM1Opt()$) gives the metaheuristic outer loop of our detailed placement optimization considering direct vertical M1 routing. The inputs include a routed layout T, a weighting factor α,

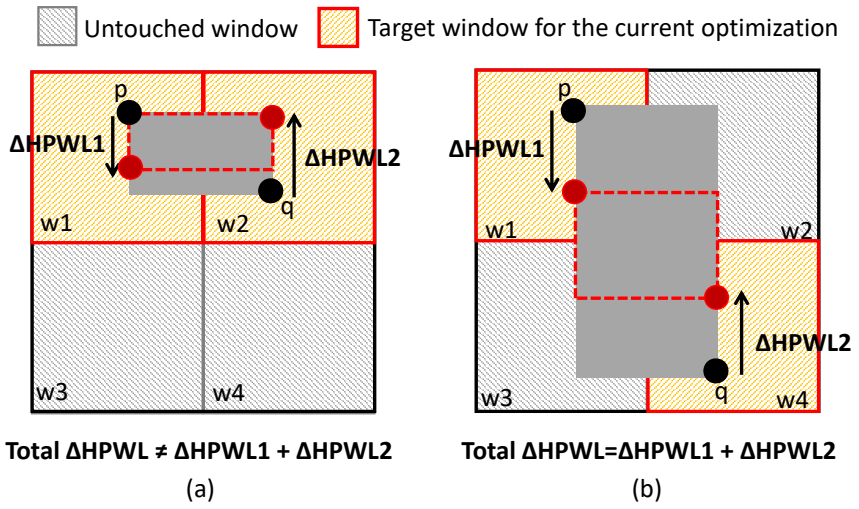


Figure 3.24: HPWL calculation for two cases. (a) Target windows with intersecting projections on the y-axis. (b) Windows with disjoint projections. In the case of (a), the total $\Delta HPWL$ is not equal to the sum of $\Delta HPWL$ values that are calculated from each window.

and a sequence (queue) of input parameter sets U. Each parameter set in U includes window width (b_w), window height (b_h), maximum x displacement for cells (l_x), and maximum y displacement of cells (l_y). The sequence U is determined empirically based on experimental results (see Section 3.2.4). The output is an optimized layout T_{opt} with a heuristically minimized objective value Obj.

In Line 2, we obtain the first input parameter set u in the current U. In Lines 3 – 11, we iteratively run $DistOpt()$ with u until the normalized improvement (ΔObj) of the objective with respect to Obj of the previous iteration is less than a threshold θ. We use $\theta = 1\%$ as the threshold. In Line 4, we first store the previous Obj value as $preObj$. In Lines 5 – 6, we then perform $DistOpt()$ with window size and perturbation range defined in u (i.e., $u.b_w$, $u.b_h$, $u.l_x$, $u.l_y$) but without allowing the flip operation ($f = 0$). After that, $DistOpt()$ is performed again in Lines 7 – 8, with allowing of the flip operation ($f = 1$) but without allowing perturbation. Empirically, we observe that a sequential optimization that performs perturbation and flipping serially is faster than an optimization that performs perturbation and flipping simultaneously, while both optimizations give similar solution quality. In Line 9, we update the x and y shift values for windows (t_x, t_y). Although we avoid interference between windows by selecting diagonally-adjacent windows (recall Figure 3.23) for parallel optimization, cells at the boundary (i.e., cells that overlap two windows simultaneously) cannot be optimized. Thus, similar to the method of [39], we

shift the windows to handle the unoptimized boundary region of the previous iteration. If ΔObj is less than θ (Line 3), we change u to the next input parameter set in U (Line 2). We iterate the optimization until we reach the last input parameter set in U.

Algorithm 9 describes details of $DistOpt()$. According to the given input parameters, we partition the layout into small windows (Line 1). We then select target windows that are independently optimizable and store them in D (Line 3) as explained above. Since we select target windows such that windows do not have any vertical or horizontal overlaps, the parallel optimization has $k = \sqrt{|W|}$ iterations, where $|W|$ is the total number of windows. In Lines 5 – 6, all windows $d \in D$ are optimized in parallel. For each window, we list candidates for each cell according to a given perturbation range (i.e., l_x and l_y, the maximum displacement of x and y, respectively). Along with input parameters α, β_n, γ and δ, we formulate the MILP instance for the window and use *CPLEX* to solve the MILP instance. The solution is updated for each window, and is then used as a boundary condition for the target windows in the next iteration.

Algorithm 8 Overall flow of *VM1Opt*

Procedure *VM1Opt*(T, α, U)
Input : Layout T, weighting factor α, queue of parameter sets U
Output : Layout T_{opt}

1: **while** $U \neq \emptyset$ **do**
2: $u \leftarrow U.pop()$; $\Delta Obj \leftarrow \infty$;
3: **while** $\Delta Obj \geq \theta$ **do**
4: $preObj \leftarrow Obj$;
5: $l_x \leftarrow u.l_x$; $l_y \leftarrow u.l_y$; $f \leftarrow 0$;
6: $(T, Obj) \leftarrow DistOpt(T, t_x, t_y, u.b_w, u.b_h, l_x, l_y, f, \alpha)$;
7: $l_x \leftarrow 0$; $l_y \leftarrow 0$; $f \leftarrow 1$;
8: $(T, Obj) \leftarrow DistOpt(T, t_x, t_y, u.b_w, u.b_h, l_x, l_y, f, \alpha)$;
9: Update t_x, t_y
10: $\Delta Obj \leftarrow (preObj - Obj)/preObj$;
11: **end while**
12: **end while**
13: $T_{opt} \leftarrow T$;
14: return T_{opt};

Algorithm 9 Procedure *DistOpt*

Procedure $DistOpt(T, t_x, t_y, b_w, b_h, l_x, l_y, f, \alpha)$
Input : Horizontal (vertical) offset t_x (t_y), width (height) of window b_w (b_h), perturbation range in x (y) l_x (l_y), binary indicator of whether flip operation is allowed f, weighting factor α
Output : Updated layout T_{opt}, objective value Obj

A set of windows $W \leftarrow Partition(T, t_x, t_y, b_w, b_h)$;
for $i = 1$ to $\sqrt{|W|}$ **do**
  $D \leftarrow$ set of current target windows;
  // parallel optimization
  $MILPFormulation(d, l_x, l_y, f, \alpha)$ for $\forall d \in D$;
  Solve MILP and update MILP solutions to T;
  // parallel optimization ends
end for
$T_{opt} \leftarrow T$
$Obj \leftarrow CalculateObj(T_{opt})$;
return T_{opt};

3.2.4 Experiments

Experimental Setup

We implement our flow in C++ with *OpenAccess 2.2.43* [128] to support LEF/DEF [119], and with *IBM ILOG CPLEX Optimization Studio v12.6.3* [115] as our MILP solver. We apply our detailed placement optimization flow to an ARM Cortex-M0 core (*M0*) and three designs (*AES*, *JPEG* and *VGA*) from the *OpenCores* website [124]. The design information is summarized in Table 3.10. The four designs are implemented with $7nm$ *OpenM1* and *ClosedM1* triple-V_{th} libraries from a leading technology consortium. We synthesize the testcases using *Synopsys Design Compiler K-2015.06-SP4* [129], and then perform placement and routing using *Cadence Innovus v16.1* [112]. The experiments are performed with 8 threads on a $2.6GHz$ Intel Xeon dual-CPU server. We note that with flexible computing resources, the number of usable threads could be as large as the number of layout windows that are independently optimizable (i.e., $\sqrt{|W|}$) to reduce runtime for larger designs.

Experimental Results

We have conducted two basic types of experiments. **Expt1** experiments seek to optimize our overall flow by finding input parameters and optimization sequences that give dominating runtime versus solution

quality tradeoffs. The *AES* design with *ClosedM1* is used for **Expt1** experiments. **Expt2** experiments apply our flow to both *ClosedM1*-based and *OpenM1*-based designs. For all experiments, we use $\beta = 1$ so that our MILP formulation minimizes pure HPWL.

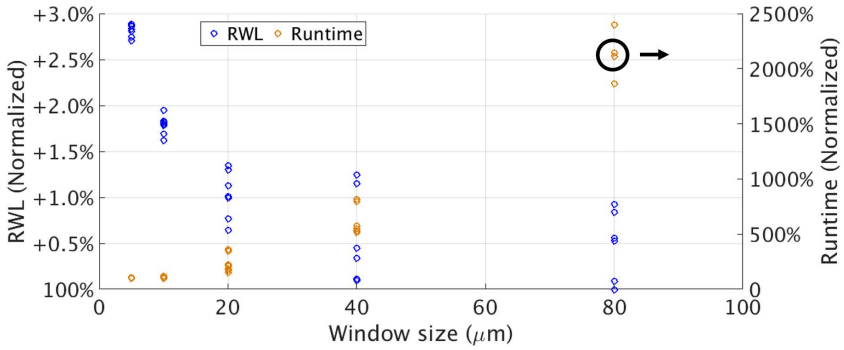


Figure 3.25: Scalability test with various window sizes and perturbation ranges.

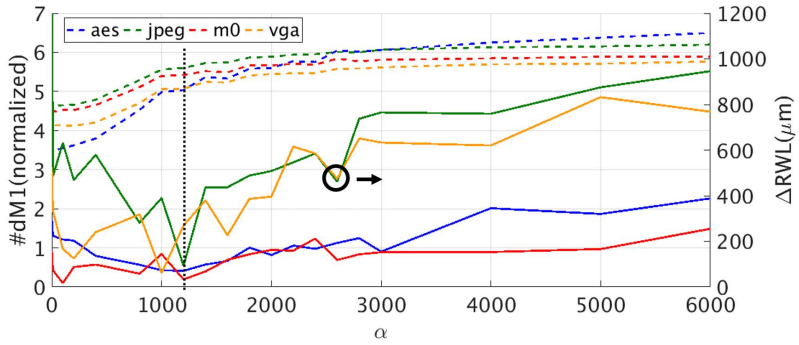


Figure 3.26: Sensitivity of total routed wirelength (RWL) and the number of direct vertical M1 routings (#dM1) to α.

Expt1-1: Scalability study on window size and perturbation range. We sweep the window size and the perturbation range to study the tradeoff between solution quality and runtime. We assume square windows and vary $b_w = b_h$ from $5\mu m$ to $80\mu m$. For the perturbation range, we try $l_x \in \{2, 3, 4, 5\}$, $l_y \in \{0, 1\}$. In this experiment, we only run one iteration in Algorithm 8 (i.e., one pair of $DistOpt()$). Figure 3.25 shows the normalized routed wirelength (RWL) and runtime versus the window size. As the window size increases, the routed wirelength decreases, as expected. However, we observe huge runtime increases, e.g., 5× runtime increase with $b_w = b_h = 40\mu m$. To balance between runtime overhead and solution quality, we select the option with shortest runtime that gives $\leq 1\%$ total routed wirelength increase compared to the minimum routed wirelength; this is $b_w = b_h = 20\mu m$, $l_x = 4$, and $l_y = 1$.

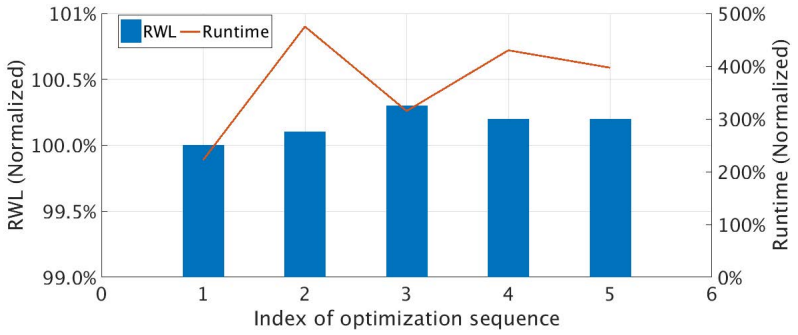


Figure 3.27: Results of various optimization sequences.

Expt1-2: Sensitivity study for α. We sweep α values and study the impact of α on the number of direct vertical M1 routings (#dM1) and the routed wirelength (RWL). We vary α from 0 to 6000 – e.g., for *ClosedM1*-based design, the objective with $\alpha = 10$ prefers one more aligned pin pair at the cost of at most 10 units increase in HPWL. Figure 3.26 shows total routed wirelength (RWL) and the number of direct vertical M1 routings (#dM1) versus α. As α increases, the number of direct vertical M1 routings increases. However, maximizing the number of direct vertical M1 routings does not always reduce routed wirelength, Based on our studies, we select $\alpha = 1200$ for *ClosedM1*. Similarly, we experiment on *OpenM1*-based designs and select $\alpha = 1000$.

Expt1-3: Sequence of optimization. We explore various sequences of input parameter sets ($b_w = b_h$, l_x, l_y) to optimize our overall flow. We illustrate this with five example optimization sequences: (1) (20, 4, 1); (2) (10, 3, 1) $\rightarrow$ (10, 4, 0) $\rightarrow$ (20, 4, 0) ; (3) (10, 3, 1) $\rightarrow$ (20, 3, 1) $\rightarrow$ (20, 3, 0); (4) (10, 3, 1) $\rightarrow$ (20, 3, 0) ; and (5) (10, 3, 1) $\rightarrow$ (10, 3, 0) $\rightarrow$ (20, 3, 1) $\rightarrow$ (20, 3, 0). Figure 3.27 shows RWL and runtime for these optimization sequences. We observe that optimization sequences 1 and 2 with $l_x = 4$ give better solution quality (in terms of RWL). However, optimization sequence 2 consumes twice the runtime of optimization sequence 1. Therefore, $(20, 4, 1)$ would be a preferred choice of sequence.

Table 3.10: Results of Expt2.

Design	#Inst	Util (%)	α	#dM1		M1 WL (μm)		#via12		HPWL (μm)		RWL (μm)		WNS (ns)		Power (mW)		Runtime (s)
				Init	Final (Δ%)	Init	Final (Δ%)	Init	Final (Δ%)	Init	Final (Δ%)	Init	Final (Δ%)	Init	Final	Init	Final (Δ%)	
***ClosedM1*-based designs**																		
M0	9922	75%	1200	545	2955 (442.2)	676	629 (-7.0)	35766	31932 (-10.7)	22850	23760 (4.0)	27636	26833 (-2.9)	0.000	0.000	2.444	2.431 (-0.5)	344
AES	12345	75%	1200	631	3177 (403.5)	970	710 (-26.8)	43248	38631 (-14.4)	30420	28890 (-5.0)	32560	30471 (-6.4)	0.000	0.000	3.240	3.212 (-0.9)	711
JPEG	54570	75%	1200	3694	20688 (460.0)	3605	3329 (-7.7)	179315	153500 (- 5.7)	91030	88900 (-2.3)	96621	90593 (-6.2)	0.000	0.000	28.592	28.399 (-0.7)	1216
VGA	68606	75%	1200	2460	12473 (407.0)	5973	5428 (-9.1)	270930	255466 (-10.7)	169200	169800 (0.4)	206558	204269 (-1.1)	0.000	-0.002	53.614	53.542 (-0.1)	561
***OpenM1*-based designs**																		
M0	9891	75%	1000	1183	1931 (63.2)	3681	3790 (3.0)	35099	34336 (-1.7)	24790	24570 (-0.9)	29884	29575 (-1.0)	-0.003	0.000	2.475	2.468 (-0.3)	298
AES	12348	75%	1000	1341	1975 (47.3)	4646	4620 (-0.5)	43004	42269 (-4.1)	30670	29980 (-2.2)	34338	33592 (-2.2)	0.000	0.000	3.273	3.263 (-0.3)	325
JPEG	54689	75%	1000	8391	13763 (64.0)	18709	19244 (2.8)	173622	166411 (-3.8)	92100	91110 (-1.1)	103257	101463 (-1.7)	0.000	-0.001	29.024	28.957 (-0.2)	1026
VGA	68729	75%	1000	7714	13132 (70.2)	26912	26823 (-0.3)	261424	251558 (-2.2)	170000	168700 (-0.8)	215218	213598 (-0.8)	0.000	-0.002	53.805	53.730 (-0.1)	515

Expt2-1: Detailed placement optimization for *ClosedM1*-based designs. Table 3.10 shows overall results for our detailed placement optimization. Our optimizer increases the number of direct vertical M1 routings by more than 4× compared to the initial post-routing solution, while decreasing overall M1 wirelength. This means that we remove long vertical M1 routings that are not used for direct vertical routing, while generating many short, direct vertical M1 routes; this results in smaller M1 wirelength and a larger number of M1 routing segments. Along with the increase in the number of direct vertical M1 routings, we achieve up to 6.4% routed wirelength (RWL) reduction and up to 14.4% #via12 reduction without design rule violations (DRCs).[42] Total power also decreases by up to 0.9%. For half of the designs, HPWL increases in favor of more dM1 to further reduce routed wirelength.

To study the impact of direct M1 routing on congestion reduction, we increase the initial utilization on the *AES* design so as to induce congestion hotspots, which lead to design rule violations. In Figure 3.28, we show that our optimizer has the added benefit of avoiding a substantial fraction of DRCs (#DRCs orig versus opt in the figure). We note that even though our optimization consistently decreases #DRCs, routing QoR is ultimately determined by the initial placement quality. Notably, placement QoR with utilization 83% from the commercial tool is worse than placement with utilization 84% in terms of DRCs. The cause of this phenomenon is beyond our present scope.

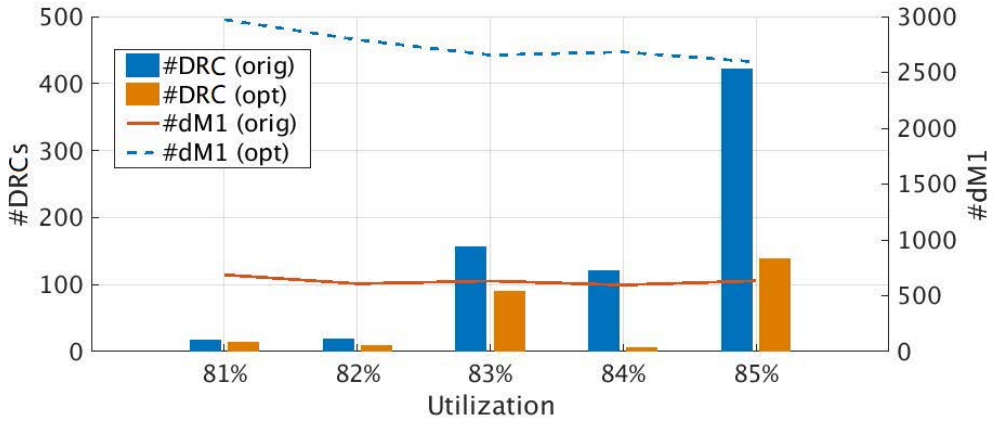


Figure 3.28: #DRCs after optimization for *AES* design with various utilizations. Also shown: the number of direct vertical M1 routings.

[42]Here we refer to routing DRCs. In this work, we do not consider advanced node placement rules (e.g., drain-drain abutment, minimum implant area, etc.). However, our framework is fully compatible, and can be easily integrated, with the work of [39] and complex sub-14nm rules.

Expt2-2: Detailed placement optimization for *OpenM1*-based designs. Our optimizer increases the number of direct vertical M1 routings by around 60% compared to the initial post-routing solution. We observe that the increase of the number of direct vertical M1 routings for *OpenM1*-based designs is much smaller than that for *ClosedM1*-based designs. This small increase of the number of direct vertical M1 routings results in only up to 2.2% routed wirelength reduction, and up to 4.1% #via12 reduction, without design rule violations. There can be several reasons for the lesser improvement seen for *OpenM1*-based designs. Our current hypothesis is that P&R for *OpenM1* is very similar to traditional P&R in terms of pin access. In traditional P&R flows with conventional libraries, where most pins are on M1, the M2 layer is used to access the pins. Similarly, *OpenM1* cells also have pins (on or) below M1, and M1 can be used for pin access. Thus, P&R for *OpenM1* can be seen as a variant of the conventional P&R flow, where the bottom routing layer is shifted down to M1. Indeed, in *OpenM1*-based designs, direct vertical M1 routing can block access to other pins, which limits the wirelength reduction. On the other hand, in *ClosedM1*-based designs, direct vertical M1 routing does not block any pin access, and is thus "free" in terms of routing resource. Compared to *ClosedM1*, where routed wirelength can be reduced even at the cost of HPWL increase, *OpenM1*-based designs prefer smaller α to reduce HPWL. However, given our use of a black-box commercial router, it is difficult to identify root causes of the improvement difference between *OpenM1* and *ClosedM1*. This is the subject of one of our ongoing studies.

3.2.5 Conclusion

In this work, we present a vertical M1 routing-aware detailed placement optimization based on mixed-integer linear programming (MILP) for two new cell architectures in sub-10nm nodes, i.e., *ClosedM1* and *OpenM1*. With our optimization, up to 6.4% (resp. 2.2%) total routed wirelength reductions and 14.4% (resp. 4.1%) #via12 reductions are achieved for *ClosedM1*-based (resp. *OpenM1*-based) designs, with no adverse timing impact.

We note that the model for *ClosedM1* library cells might need to change since the vertical M1 routings might affect cells' library model (change in gate capacitance, etc.). However, according to our study with an INV cell in *ASAP ASU 7nm PDK* [111], the timing impact is negligible ($\leq 0.1ps$).[43]

[43]We modify pin shapes (increase the pin length by 32nm) in a cell layout, run parasitic extraction with *Calibre xRC*

3.3 Acknowledgments

Chapter 3 contains reprints of Changho Han, Kwangsoo Han, Andrew B. Kahng, Hyein Lee, Lutong Wang and Bangqi Xu, "Optimal Multi-Row Detailed Placement for Yield and Model-Hardware Correlation Improvements in Sub-10nm VLSI", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2017; Changho Han, Andrew B. Kahng, Lutong Wang and Bangqi Xu, "Enhanced Optimal Multi-Row Detailed Placement for Neighbor Diffusion Effect Mitigation in Sub-10nm VLSI", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 38(9), 2019; and Peter Debacker, Kwangsoo Han, Andrew B. Kahng, Hyein Lee, Praveen Raghavan and Lutong Wang, "Vertical M1 Routing-Aware Detailed Placement for Congestion and Wirelength Reduction in Sub-10nm Nodes", *Proc. ACM/EDAC/IEEE Design Automation Conference*, 2017. The dissertation author is a main contributor to, and a primary author of, each of these papers.

I would like to thank my coauthors Peter Debacker, Changho Han, Kwangsoo Han, Andrew B. Kahng, Hyein Lee, Praveen Raghavan and Bangqi Xu, as well as the research support from Samsung Electronics.

v2016.1_31.21 [123], and measure cell delay and slew with *HSPICE I-2013.12* [130]. We observe that the delay and slew impacts of the pin modifications are negligible ($\leq 0.1ps$). Further, there are only a small number of possible uses of vertical M1 incident to a cell (this number is a function of the number of pins, and of upward versus downward alignments). In a regime where these delay and slew changes must be modeled, each of these contexts could be characterized.

Chapter 4

Open Source Physical Design Methodologies in Routing

This chapter presents two works towards a complete, end-to-end academic detailed routing flow targeting advanced nodes. First, we present a multi-level, standard cell- and instance-based, complete, robust, scalable and design rule-aware pin access analysis framework. The proposed framework includes pin-based access point generation, boundary conflict-aware access pattern generation and cluster-based access pattern selection based on dynamic programming. The work achieves 100% DRC-clean pin access and demonstrates a superior final detailed routing solution as compared to the best known results using the ISPD-2018 initial detailed routing benchmark suite. Second, we present a complete, end-to-end academic detailed router, TritonRoute. Our router is capable of comprehending connectivity and design rule constraints using industry-standard formats. Our router consists of an in-memory router database that complies with the LEF/DEF data models, a pin access analysis engine, a track assignment engine, a detailed routing engine, and a design rule checking engine. The detailed routing engine includes a ripup-and-reroute-based path search engine, capable of avoiding potential design rule violations, as well as working around existing design rule violation markers. The router is evaluated using the official ISPD-2018 contest benchmark suite, demonstrating an extremely low level of DRCs. Overall, TritonRoute improves wirelength by up to 0.8% (avg. 0.4%), via count by up to 16.1% (avg. 9.3%) and DRCs by up to 100% (avg. 92.0%) as compared to the known best detailed routing solutions.

4.1 Pin Access Analysis Framework for Detailed Routing

Pin accessibility has been one of the major crucial issues [3][84] in advanced node enablement. Various related topics have been widely studied in recent works, ranging from detailed placement optimization, standard cell layout optimization and new design rule-aware access model. (See Section 4.1.1 below for our definition.)

The works of [69][107] perform detailed placement optimization using a global routing solution as guidance, with pin accessibility modeled only in the form of pin density. Ding [15] develops a dynamic programming and linear programming-based detailed placement optimization considering pin access per instance pin. Ye [104] proposes an integer linear programming formulation to solve the unidirectional cell layout optimization under middle-of-line structure. However, the above models are over-simplified with assumptions of 1D gridded design and distance-based cost function, with no precise awareness of design rules. Recently, Xu [100][102] develops a series of pin access planning and regular routing techniques for self-aligned double patterning. These works, still under the assumption of 1D gridded design, are the first open literature trying to address both cell-level and instance-level pin accessibility. However, the methodology has a few drawbacks: (i) there is no robust flow to generate "hit points" given any 1D/2D, gridded/non-gridded design, with or without specific (e.g., self-aligned double patterning) design rules; (ii) the flow is unrealistic in that the number of "hit point combinations" is far too large, resulting in a complex lookup table that is impractical to use; and (iii) the benchmark suite is not public and includes testcases only up to 12K cells. These small testcases nevertheless consume as much as 800 seconds of wall time in multithreaded mode, which is a prohibitive runtime cost for real industry testcases and use contexts.

To our knowledge, no works present a complete, fully defined pin access analysis flow, or demonstrate robustness with a real detailed routing contest benchmark suite. In this work, we present a real, robust, scalable and design rule-aware dynamic programming-based pin access analysis framework that performs both standard cell-based and instance-based pin access analysis. With the integration to the open source TritonRoute [54][126], we demonstrate superior solution quality over the best known results [60] using the official ISPD-2018 benchmark suite [73]. Our main contributions are summarized as follows.

- We propose a multi-level, standard cell-based and instance-based pin access analysis framework with intra-cell and inter-cell pin accessibility awareness.
- We propose a robust and design rule-aware pin access point generation methodology for unique instances, supporting both planar and via access, and both on-track and off-track access.
- To achieve **intra-cell** pin compatibility, we propose a dynamic programming-based, design rule and boundary conflict-aware access pattern generation methodology for unique instances.
- We propose a dynamic programming-based access pattern selection methodology for standard cell instance clusters, which minimizes **inter-cell** pin access conflicts. To the best of our knowledge, this proposed framework is the only scalable solution in the open literature.
- We improve the pin access over the open-source TritonRoute v0.0.6.0 [127] (the latest release as of this writing), achieving design rule check (DRC)-clean via access for all of ISPD-2018 benchmark suite testcases. With the integration to TritonRoute, we demonstrate superior solution quality over the best known results using the official ISPD-2018 benchmark suite.

The remainder of this work is organized as follows. Section 4.1.1 provides background information for pin access. Section 4.1.2 describes our pin access methodology. Section 4.1.3 presents our experimental setup and results. Section 4.1.4 gives conclusions and directions for future work.

4.1.1 Preliminaries

In this section, we describe fundamental concepts that underlie pin access analysis: *unique instance*, *access point*, *access pattern*, and *coordinate types*.

Unique Instance

A **unique instance** is defined by a *signature*, which consists of (i) the cell master of the instance (e.g., NANDX1, NORX4, etc.); (ii) the orientation of the instance (e.g., R0, R180, MX, MY); and (iii) offsets to all track patterns that exist in the design DEF. Two instances having different signatures require separate intra-cell pin access analysis flows. Figures 4.1(a) and (b) illustrate two different unique instances.

Although the two instances share the same cell master and orientation, they are considered as different unique instances because they have different offsets to routing track patterns, resulting in different on-track, off-track conditions for the same pin access location (relative to the origin of the cell master). Thus, these instances require separate intra-cell pin access analyses. By contrast, two instances having the same signature would have exactly the same intra-cell pin access analysis result. Thus, we only need to perform intra-cell pin access analysis once for each unique instance.

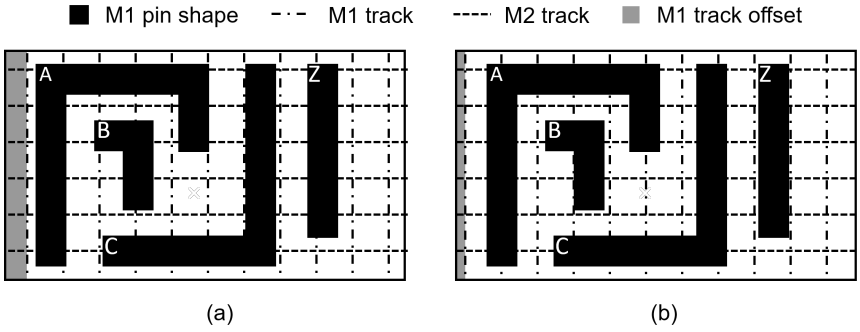


Figure 4.1: Illustration of two different unique instances that have the same cell master and orientation, but different offsets to track patterns.

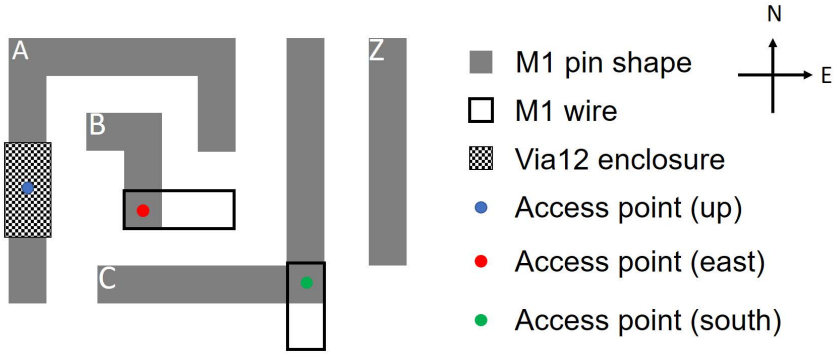


Figure 4.2: Illustration of access points.

Access Point

For each pin, an access point is an (x, y) coordinate on a metal layer where the detailed router ends routing. Each access point stores from which direction the router can access the pin. For example, in Figure 4.2, pin A has an access point indicating the up direction. We use a via12 enclosure to show that

an up-via (i.e., a via connecting the pin to the upper metal layer) is valid to escape from this access point. Similarly, pin B (resp. C) has an access point indicating that routing to the east (resp. south) is valid. In our implementation, Each access point may indicate multiple valid access directions. For the up direction, we also store which vias are valid to use, among which one via is primary (preferred to use). The access point must be on the pin shape.

Access Pattern

For each unique instance, an access pattern consists of one access point per pin, so that the primary vias from these access points are compatible (i.e., DRC-clean) with each other.

Coordinate Type

To accommodate a broad range of technology nodes, we define four coordinate types (and respective *cost* values, given in parentheses) as follows.

- An **on-track (0)** coordinate is on a preferred or non-preferred routing track. We always use the upper-layer preferred direction routing tracks as the non-preferred direction routing tracks for the current metal layer so that the on-track up-via access aligns to both the current and its immediately above metal layers.
- A **half-track (1)** coordinate is at the midpoint between two neighboring routing tracks.
- A **shape-center (2)** coordinate is at the midpoint between the left and right (or top and bottom) coordinates of a rectangular pin shape. If the pin consists of polygon(s), we generate the maximum rectangles of the polygon(s) (all overlapping rectangles that are maximal in area) to obtain shape-center coordinate(s). We skip the shape-center x (resp. y) coordinate if the x-span (resp. y-span) of the rectangle touches at least two tracks; we do this to reduce the occurrence of unique, off-track coordinates.
- An **enclosure boundary (3)** coordinate satisfies the via-in-pin requirement for an up-via access and the via enclosure alignment with the pin shape boundary.

Figure 4.3 illustrates examples of the coordinate types for a horizontal preferred direction. In Figures 4.3(a) and (b), we see that up-vias at the on-track and half-track coordinates cause minimum step DRCs. In such cases, we need shape center or enclosure boundary access points although they are off-track as illustrated in Figures 4.3(c) and (d). The above four types of coordinates are concise, while satisfying a broad range of technology nodes – from mature nodes where 2D, off-track pin access is required, to advanced nodes where 1D, on-track pin access is required. The cost serves as the priority (the lower, the better) when we loop through different types of coordinates to generate access points (cf. Lines 3 and 4 in Algorithm 10, in Section 4.1.2 below).

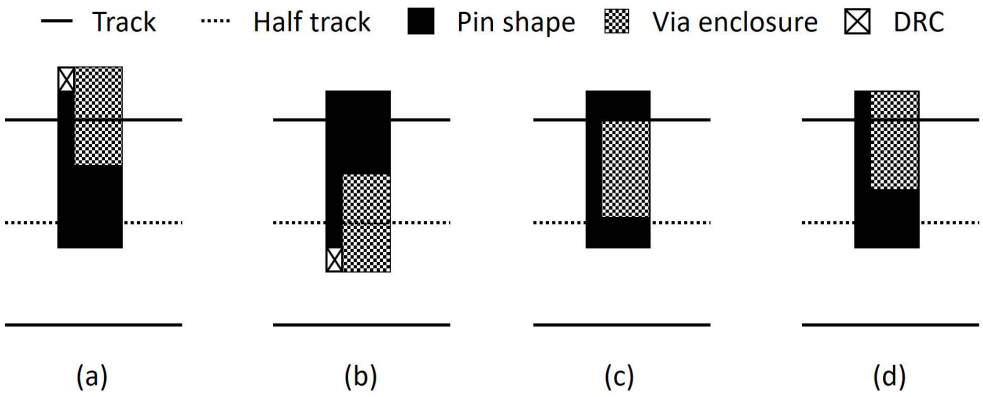


Figure 4.3: Illustration of four y-coordinate types, overlaid with same-layer up-via enclosure at the access point: (a) on-track; (b) half-track; (c) shape-center; and (d) enclosure boundary. Only (c) and (d) are DRC-clean.

4.1.2 Methodology

In this section, we describe our methodology to analyze pin accessibility for detailed routing. We perform three analyses in a multi-level sequence of three steps: (i) **pin-based access point generation**; (ii) **unique instance-based access pattern generation**; and (iii) **cluster-based access pattern selection**. The first step enumerates valid *access points* per unique instance, **without** consideration of intra-cell or inter-cell pin access compatibility. The second step picks good access points per pin within a given unique instance, forming an *access pattern*, within which **intra-cell** pin accesses are mutually compatible. The third step selects the best access pattern for all instances in the design, with awareness of **inter-cell** pin compatibility.

Step 1: Pin-Based Access Point Generation

Although we could enumerate all coordinate types to generate every access point per pin, in a reasonable detailed routing-driven pin access analysis framework the number of generated access points per pin should be neither too small nor too large. Too small a number of access points will overly restrict the solution space in detailed routing, resulting in degraded solution quality. On the other hand, given the heuristic, cost-based nature of modern detailed routing [60][126], too large a number of access points will provide excessive options (e.g., many off-track access points) for the detailed router, again resulting in degraded solution quality. Thus, the access point generation flow must be robustly designed to generate a proper amount of access points. In our flow, for example, to generate an access point at (x, y) on Metal1, where the preferred routing direction is horizontal, we consider all four coordinate types for the y coordinate (corresponding to the preferred direction), but only consider the first three coordinate types for the x coordinate (corresponding to the non-preferred direction) to reduce unique, off-track coordinates. We explain below the determination of "proper amount" after the description of Algorithm 10.

Algorithm 10 Pin-based access point generation

```
1:  Inputs: pin, track patterns tps, viadefs vias
2:  Output: valid access points aps
3:  for all nonPreferredDirCoordType t1 ∈ {0, 1, 2} do
4:      for all preferredDirCoordType t0 ∈ {0, 1, 2, 3} do
5:          tmpAps ← genAccessPoint(pin, tps, vias, t0, t1)
6:          for all ap ∈ tmpAps do
7:              if isValid(ap) then
8:                  aps += ap
9:              end if
10:         end for
11:         if |aps| ≥ k then
12:             return
13:         end if
14:     end for
15: end for
```

Algorithm 10 describes the pin-based access point generation. In Lines 3 – 4, we loop through different combinations of x and y coordinates sequentially according to their cost. For example, we first generate all (on-track, on-track) points, then (off-track, on-track) points, etc. In Line 5, for each type of coordinates, we first generate all access points. Then in Lines 6 – 10, we add all valid access points to the

output. An access point is valid if a via can be dropped DRC-free to access the pin. We use an accurate DRC engine similar to the one used in [126] to perform the design rule check, considering all design rules existing in the specific design. Next, in Lines 11 – 13, we check whether we have generated enough access points for a pin, and early-terminate the procedure once the number of generated access points is equal to or greater than our required number k. Given the above, all access points of given coordinate types are generated, DRC-checked and added before we try to early-terminate the procedure. Therefore, the number of access points generated may be slightly larger than k. This behavior allows more access points to be generated when we are given a large pin shape, while also reducing the occurrence of unique, off-track coordinates. In our implementation, $k = 3$ for both standard-cell and macro-cell pins.

Step 2: Unique Instance-Based Access Pattern Generation

For each unique instance, we now describe how to pick a good access point per pin to form an *access pattern* in which the chosen access points are compatible with each other. Figure 4.4 illustrates our unique instance-based access pattern generation flow. The access pattern generation mainly consists of (i) pin ordering, (ii) graph construction, and (iii) dynamic programming-based pattern generation.

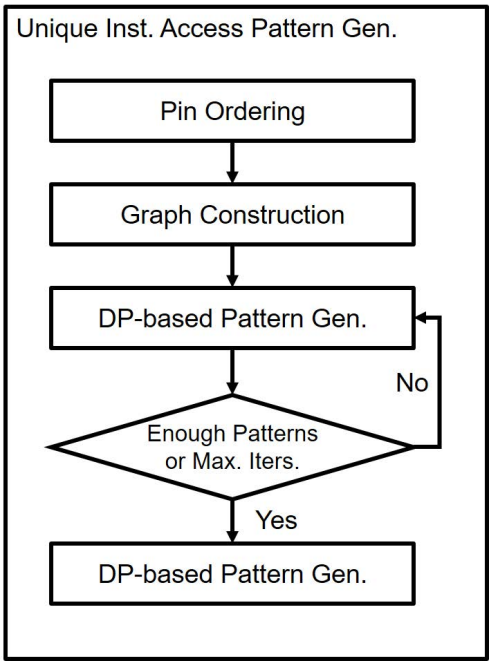


Figure 4.4: Iterative access pattern generation flow.

Pin ordering. Pin ordering is a preparation step for graph construction and dynamic programming-based pattern generation. Given a unique instance and an ordering of the pins in the unique instance, we assume only the neighboring ordered two-pin pairs might have conflicting access points (i.e., the two access points cause DRCs). For example, if we have a pin order of <A, B, C, Z>, then our assumption is that only <A, B>, <B, C> and <C, Z> could have conflicting access points, while <A, C>, <A, Z> and <B, Z> should not have conflicting access points. In this way, the access patterns can be generated within reasonable amount of time, without the need to perform design rule check among all two-pin pairs. For corner cases where non-neighboring two-pin pairs have conflicting access points, we can still avoid such cases by a post-processing method, described at the end of the discussion below of DP-based access pattern generation. As shown in Section 4.1.3, this method works well in all ISPD-2018 benchmark suite testcases.

For a pin, if the averaged coordinates of all its access points are (x_{avg}, y_{avg}), then given a unique instance, we sort the pins according to $(x_{avg} + \alpha \cdot y_{avg})$. Figure 4.5 illustrates an example of unique instances with four pins. If $\alpha = 0$, then the pin ordering is equivalent to the ordering of x_{avg}. Thus, we obtain a pin order of <A, B, C, Z>. The first and last pin according to the pin order are **boundary pins**, which receive special treatment in access pattern generation as described below. Generally, given a reasonably small α ($\alpha < 1$), the first and last pins are the leftmost and the rightmost pins in the unique instance, respectively. In our implementation, we use $\alpha = 0.3$.

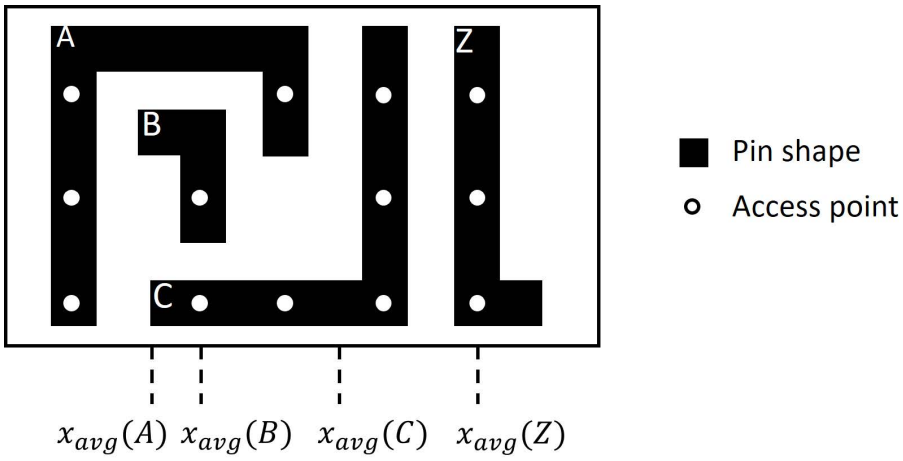


Figure 4.5: Pin ordering.

Graph construction. We build a graph for dynamic programming. Figure 4.6 shows the directed graph corresponding to the unique instance shown in Figure 4.5, assuming $\alpha = 0$. All edges are directed from left to right in the figure. The leftmost (resp. rightmost) vertex in the graph is the (virtual) starting (resp. ending) vertex, which serves as the starting (resp. ending) point in the dynamic programming that we describe below. Vertices between the starting and ending vertices represent access points; these are grouped by the owner pin of the access point, and ordered sequentially following the aforementioned pin order. We build complete bipartite graphs over neighboring groups' respective vertex sets. A path from the starting vertex to the ending vertex visits one access point vertex per pin. The visited access points represent an access pattern.

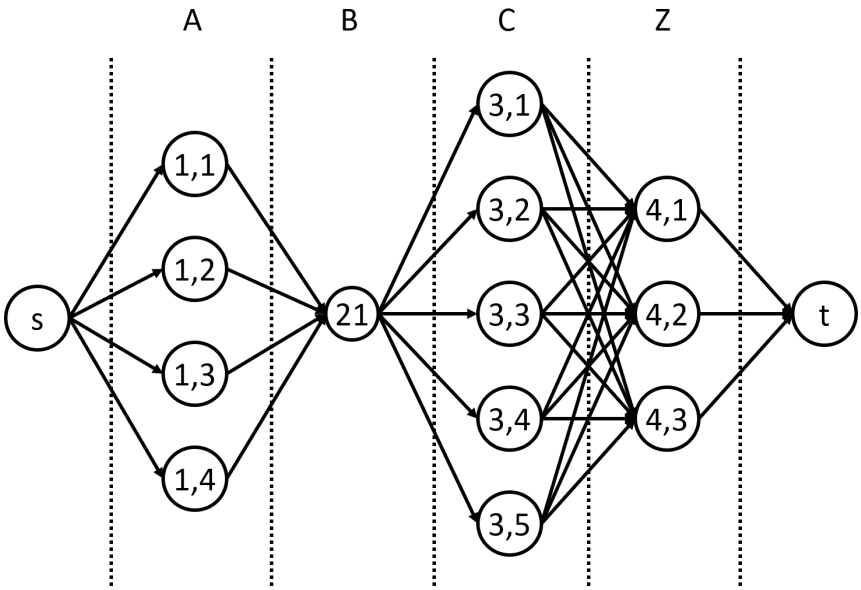


Figure 4.6: Graph for dynamic programming-based access pattern generation.

DP-based access pattern generation. Algorithm 11 describes the procedure of dynamic programming-based access pattern generation. The input is the graph. We describe all access points according to the pin index (m) and access point index (n). For example, access point $\{3,2\}$ in Figure 4.6 is the second access point ($n = 2$) of the third pin ($m = 3$). Line 3 initializes the dynamic programming array dp. The array stores the minimum cost up to the current vertex, and its previous vertex. The minimum cost is initialized to infinity for every vertex except for the source. In Lines 4 – 17, we loop through all vertices (access points) of the current pin. For each vertex of the current pin, we find one vertex from the previous pin,

from which the total path cost is minimized. Line 9 gets the edge cost from one previous access point vertex to the current access point vertex. Line 10 gets the total cost. The total path cost equals the previous path cost plus the edge cost. In Lines 11 – 14, we update the path cost up to the current vertex if the path cost is smaller than the existing path cost stored in the vertex. We also update the previous vertex, from which the path comes from, so that we can trace back the path to obtain the access pattern solution. Line 18 traces back the dp array and returns the access pattern with the lowest cost. We perform Algorithm 11 several times to generate up to three access patterns. Each time, the edge costs are slightly different so as to obtain different access patterns.

Algorithm 11 Access pattern generation

```
1:  Inputs: graph G(V, E)
2:  Output: access patterns APs
3:  Initialize array dp[m][n] G(V, E)
4:  for all currPinIdx m do
5:     for all currApIdx n do
6:        for all prevApIdx n' do
7:           prev ← aps[m − 1][n']
8:           curr ← aps[m][n]
9:           edgeCost ← getEdgeCost(prev, curr)
10:          pathCost ← prev.cost + edgeCost
11:          if pathCost < curr.cost then
12:             curr.cost ← pathCost
13:             curr.prev ← prev
14:          end if
15:       end for
16:    end for
17: end for
18: APs ← traceBack()
19: return APs
```

Algorithm 12 details the edge cost calculation. The edge cost calculation is **boundary conflict-aware (BCA)**. In Lines 3 – 6, we assign a penalty cost to the boundary pin (the first and last pins according to the pin order) access points that have been selected in existing access patterns. This helps to generate access patterns with different boundary pin access points. Thus, two neighboring instances have more flexibility choosing compatible access patterns, as described in Section 4.1.2. Lines 7 – 8 check whether the two access points have design rule violations, and apply design rule violation cost if two access points are not compatible. Lines 9 – 10 further look back one more pin, and check whether the two access points

(indexed $prev - 1$ and $curr$) have design rule violations. This step generates a history-based cost to avoid DRCs between non-neighboring access points. We call this step **history-aware optimization**. We note that since there can only be one intermediate solution when we reach node $curr$, the nodes $prev$ and $prev - 1$ are always deterministic, and thus the cost of each edge is still fixed. Line 12 calculates the edge cost according to the quality metric of the two access patterns if neither the penalty nor the violation cost applies.

Algorithm 12 Edge cost calculation

```
1:  Inputs: previous dp array vertex prev current dp array vertex curr
2:  Output: edge cost cost
3:  if isUsed(prev) and prev ∈ boundaryAp then
4:      edgeCost = penaltyCost
5:  else if isUsed(curr) and curr ∈ boundaryAp then
6:      edgeCost = penaltyCost
7:  else if isDRCClean(prev, curr) then
8:      edgeCost = drcCost
9:  else if isDRCClean(prev-1, curr) then
10:     edgeCost = drcCost
11: else
12:     edgeCost = apCost(prev) + apCost(curr)
13: end if
14: return edgeCost
```

Finally, for all the access patterns that we generate, we use a DRC engine similar to the one used in [126] to validate whether there exist unseen DRCs, i.e., between non-neighboring groups of access points, or between multiple objects. To accelerate the access pattern generation, only up-vias are included for DRC.

Step 3: Cluster-Based Access Pattern Selection

Given access patterns per unique instance, we select the best access patterns per instance so that the access patterns of neighboring instances are compatible. Our cluster-based access pattern selection is performed on a continuous chunk of instances. We first group all instances according to their rows, and each continuous chunk of instances (no empty site in between) forms a cluster. We only consider the access pattern compatibility within a cluster while assuming that the neighboring clusters within or across rows always allow compatible access patterns. The cluster-based access pattern selection works similarly

to the access pattern generation. The pin ordering step, in Algorithm 11, is now replaced with the instance ordering step, which naturally follows the left-to-right instance ordering. The graph construction works the same way except that now each vertex represents an access pattern of an instance. Finally, the dynamic programming-based optimization selects the best access pattern per instance to minimize the total cost. To accelerate the procedure, only up-vias of boundary access points (pin A and pin Z of each instance in Figure 4.7(a)) are included for DRC.

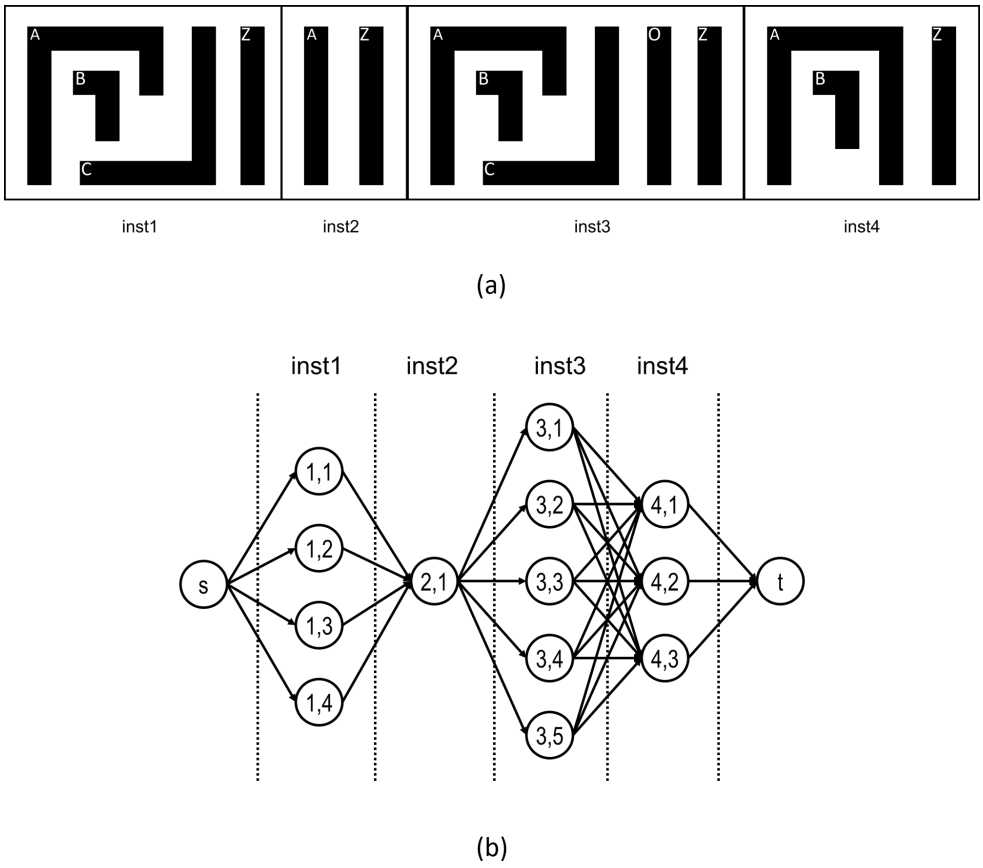


Figure 4.7: Illustration of (a) ordered cell instances and (b) corresponding graph.

4.1.3 Experiments

In this section, we present our experimental setup and results.

Table 4.1: Testcase information [73].

Benchmark	#Standard cell	#Macro cell	#Net	#IO pin	#Layer	Die size	Tech. node
ispd18_test1	8879	0	3153	0	9	$0.20\times0.19mm^2$	$45nm$
ispd18_test2	35913	0	36834	1211	9	$0.65\times0.57mm^2$	$45nm$
ispd18_test3	35973	4	36700	1211	9	$0.99\times0.70mm^2$	$45nm$
ispd18_test4	72094	0	72401	1211	9	$0.89\times0.61mm^2$	$32nm$
ispd18_test5	71954	0	72394	1211	9	$0.93\times0.92mm^2$	$32nm$
ispd18_test6	107919	0	107701	1211	9	$0.86\times0.53mm^2$	$32nm$
ispd18_test7	179865	16	179863	1211	9	$1.36\times1.33mm^2$	$32nm$
ispd18_test8	191987	16	179863	1211	9	$1.36\times1.33mm^2$	$32nm$
ispd18_test9	192911	0	178857	1211	9	$0.91\times0.78mm^2$	$32nm$
ispd18_test10	290386	0	182000	1211	9	$0.91\times0.87mm^2$	$32nm$

Experimental Setup

We implement our pin access analysis in C++ and integrate our framework with the open-source TritonRoute [126]. We perform all our experiments using the official ISPD-2018 initial detailed routing contest benchmark suite [73]. Table 4.1 summarizes the testcase information. These testcases are real industry designs with up to 290K standard cells in two technology nodes. We note that these testcases use real industry LEF-based design rule syntax, which is much more realistic than the testcases used in previous works [100][102]. Currently, no pin access framework targets the ISPD-2018 benchmark suite. To our best knowledge, no pin access framework has ever demonstrated enough robustness and scalability in publicly accessible, large benchmark testcases. Thus, we compare our work with the pin access framework from the latest release of the open-source TritonRoute v0.0.6.0 [127]. Furthermore, to enable a broader horizontal comparison to other frameworks, we also make necessary improvements to TritonRoute in addition to the integration of pin access analysis. We compare final routed designs to the best known academic detailed router – Dr. CU 2.0 [60]. All our experiments are performed using a Xeon $2.6GHz$ server in single-threaded mode. We perform three experiments.

- **Experiment 1**: We compare the quality of access points for all unique instance pins (without consideration of intra-cell or inter-cell pin access compatibility) from this work with that from TritonRoute v0.0.6.0.

Table 4.2: Results for Experiment 1: comparison between the original TritonRoute (TrRte) and our pin access analysis framework (PAAF) for all unique instance pins (without considering intra-cell or inter-cell pin access compatibility) in terms of total #access points generated (Total #APs), #access points with DRCs (#Dirty APs), and runtime.

Benchmark	**#Unique Inst**	**Total #APs**		**#Dirty APs**		**Runtime** (s)	
		TrRte	PAAF	TrRte	PAAF	TrRte	PAAF
ispd18_test1	182	2320	**3102**	0	**0**	4	2
ispd18_test2	222	3638	**4867**	1	**0**	8	4
ispd18_test3	227	3672	**4970**	1	**0**	8	4
ispd18_test4	2725	98220	**99356**	416	**0**	120	63
ispd18_test5	2733	76290	**80027**	385	**0**	142	71
ispd18_test6	2886	84012	**87876**	469	**0**	163	78
ispd18_test7	148	3982	**4152**	4	**0**	7	3
ispd18_test8	414	11814	**12316**	10	**0**	20	12
ispd18_test9	404	11832	**12342**	12	**0**	21	11
ispd18_test10	426	11749	**12254**	12	**0**	20	13

- **Experiment 2**: We compare the quality of access points for all instance pins (with consideration of intra-cell and inter-cell pin compatibility) from this work with that from TritonRoute v0.0.6.0.
- **Experiment 3**: By integrating our framework with the open-source TritonRoute and making additional improvements, we enable a preliminary comparison of pin accesses from the final routed design, and also of the final routed #DRCs, between the original TritonRoute, the best known published result from Dr. CU 2.0 [60][116], and our pin access analysis framework. We further demonstrate the capability to extend our PAAF into $14nm$ and below nodes.

Experimental Results

Experiment 1. Table 4.2 shows the experimental results of the quality of access points for all unique instance pins, between the original TritonRoute (TrRte) and our pin access analysis framework (PAAF). This experiment only evaluates the quality of each access point, but does not consider intra-cell or inter-cell pin access compatibility. Total #APs means the total number of access points generated. #Dirty APs means #access points with DRCs. Ideally, a robust pin access point generation methodology should not generate any access points with DRCs. In *ispd18_test6*, with nearly 3K unique instances, our method

generates 90K access points, all DRC-clean, within 80 seconds in single-threaded mode. Overall, our method generates only DRC-clean access points, while the original TritonRoute produces several hundreds of dirty access points. Also, our method generates more access points, while consuming less runtime.

Table 4.3: Results for Experiment 2: comparison between the original TritonRoute (TrRte) and our pin access analysis framework (PAAF) for all instance pins (considering intra-cell and inter-cell pin access compatibility) in terms of #pins without a DRC-clean access point (#Failed Pins), and runtime. Total #pins means the total number of all instance pins (with net attached).

Benchmark	**Total #Pins**	**#Failed Pins**			**Runtime** (s)		
		TrRte	PAAF		TrRte	PAAF	
			w/o BCA	w/ BCA		w/o BCA	w/ BCA
ispd18_test1	17203	31	0	**0**	4	3	5
ispd18_test2	157990	665	0	**0**	7	5	8
ispd18_test3	158110	663	0	**0**	7	5	7
ispd18_test4	316652	1305	0	**0**	95	84	94
ispd18_test5	316220	2529	80	**0**	107	85	98
ispd18_test6	474300	4048	0	**0**	113	96	121
ispd18_test7	790550	7816	0	**0**	8	7	23
ispd18_test8	790550	7816	0	**0**	20	17	39
ispd18_test9	790550	7816	0	**0**	20	17	38
ispd18_test10	790550	7816	0	**0**	21	18	49

Experiment 2. Table 4.3 shows the experimental results of the quality of access points for all instance pins, between the original TritonRoute (TrRte) and our pin access analysis framework (PAAF). We have two setups for PAAF. The first setup is "without BCA" (w/o BCA): we generate only one access pattern per unique instance, hence the access pattern is not boundary conflict-aware and there could be inter-cell pin accessibility issues. The second setup is "with BCA" (w/ BCA): we generate up to three access patterns per unique instance. Total #pins means the total number of all instance pins (with net attached). Since all of these pins must be connected in detailed routing, we need a good (i.e., DRC-clean) access point per pin. #Failed pins means the number of pins without a DRC-clean access point. We can see that the original TritonRoute fails to provide legal pin access for thousands of instance pins, while our PAAF can generate intra-cell and inter-cell DRC-clean pin access. For up to 790K instance pins, PAAF takes less than a minute of runtime in single-threaded mode. Note that runtime is one of the most important aspects of a pin access analysis framework in physical design, especially for support of

placement optimizations (i.e., detailed placement, sizing, buffering), where frequent changes in placement require a tremendous amount of inter-cell pin access analysis.

Experiment 3. By integrating our framework with the open-source TritonRoute v0.0.6.0 [127] (the latest release as of this writing) and making additional improvements, we show a preliminary result of pin accesses from the final routed design, and also of the #DRCs for the final routed design, for testcase *ispd18_test5*. Figure 4.8 compares two pin accesses from the final routed design, between Dr. CU 2.0 and our PAAF. As noted above, PAAF is capable of generating DRC-clean pin access for all instance pins. By using our robust PAAF, we surpass the best known academic detailed routing result in terms of #DRCs. The current best known result comes from Dr. CU 2.0 [60][116], with 755 DRCs. By contrast, we complete detailed routing with only two DRCs, and with no pin access issues remaining.

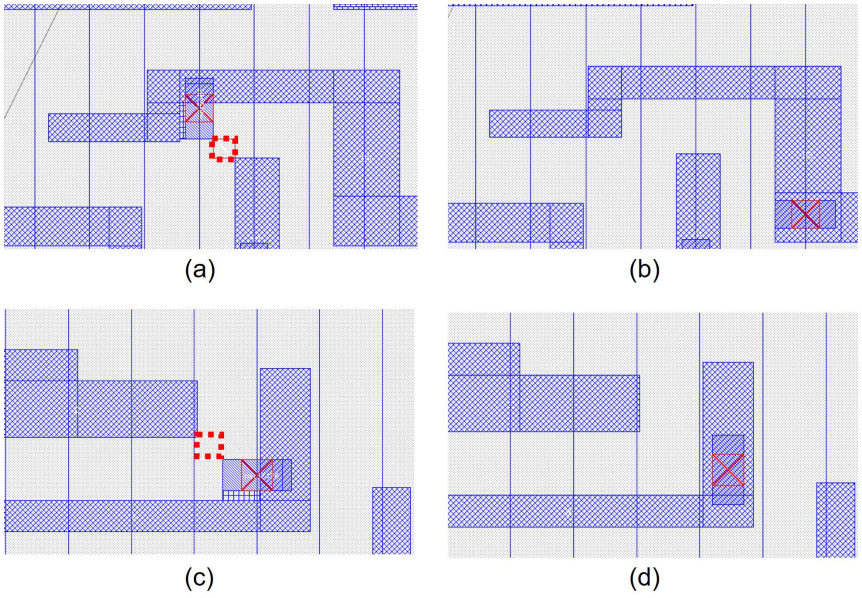


Figure 4.8: Comparison of pin access between Dr. CU 2.0 and PAAF: (a) Dr. CU 2.0 (Case 1), (b) PAAF (Case 1), (c) Dr. CU 2.0 (Case 2), and (d) PAAF (Case 2). Dashed red boxes are DRCs. Testcase: *ispd18_test5*.

We also perform a preliminary study on pin accessibility using a commercial $14nm$ library. We perform our experiments using the AES testcase from OpenCores [124] (20K instances, 779 unique instances). Our preliminary study shows that our PAAF successfully generates and selects DRC-clean access points for all 57K instance pins in a runtime of 9 seconds. An example of standard cell pin access is shown in Figure 4.9.

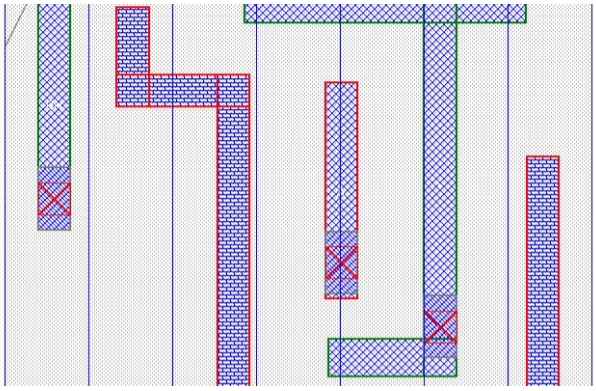


Figure 4.9: Illustration of pin accesses in $14nm$. Note that off-track pin access is enabled automatically in PAAF.

4.1.4 Conclusion

In this work, we present a multi-level, standard cell- and instance-based, complete, robust, scalable and design rule-aware pin access analysis framework. We describe our robust pin-based access point generation, boundary conflict-aware access pattern generation and cluster-based access pattern selection based on dynamic programming. We achieve 100% DRC-clean pin access and demonstrate a superior final detailed routing solution as compared to the best known results using the ISPD-2018 initial detailed routing benchmark suite.

4.2 TritonRoute: The Open Source Detailed Router

Detailed routing is a dead-or-alive critical element of advanced node enablement. New technology nodes come with smaller feature sizes, while fundamental physical (lithographic patterning, CMP, reliability, variability, etc.) and circuit (crosstalk, delay, etc.) limitations remain. As a result, ever-more complex design rules must be comprehended and satisfied at the detailed routing stage, greatly challenging routability as well as the architecture and strategy of the detailed router itself.

Due to the high complexity and enormous solution space for the VLSI routing problem, the routing is typically split into global routing and detailed routing stages. In global routing, the routing region is divided into rectangular grid cells and represented using a coarse-grained 3D routing graph. Capacities and various constraints are assigned to the edges and vertices in this 3D routing graph so that overall routing

topology and layer assignment can be optimized considering routability, timing, crosstalk, power, etc. The ensuing detailed routing stage attempts to realize the segments and vias according to the global routing solution, while minimizing design rule violations.

The detailed routing problem has been extensively studied for more than five decades. The fundamental algorithms (e.g., Lee's algorithm, unidirectional and bidirectional A* search, ripup-and-reroute paradigm, etc.) and problem formulations (e.g., channel routing and switchbox routing) have largely remained intact in commercial tools for several decades; see [7] for a thorough review. These algorithms and formulations are elaborated to meet real-world requirements (design-rule correctness, quality of result, scalability, and turnaround time) and widely deployed in today's commercial tools that support foundry N7, N5 or even N3 nodes.

However, only a few academic works [27] even attempt to present an end-to-end detailed routing flow, and almost no works make claims to viability in the real-world IC physical design (P&R) context. Since most detailed routing research focus on different objectives, such as crosstalk or new-technology contexts, comparison between these works is difficult. Further, direct application of academic codes to modern industrial benchmarks has many hurdles, especially given that commercial tools and industrial designs satisfy far more, and more complex, design rules than any academic tools.

Given the above, it is a highly significant milestone for the field that the ISPD-2018 contest, on the subject of initial detailed routing, has recently exposed industrial detailed routing challenges and benchmarks to the academic community [73][122]. The ISPD-2018 benchmark suite provides 10 testcases in $45nm$ and $32nm$ nodes, with up to 290K standard cells and 182K nets. These designs are industrial benchmarks – including large memory cells, off-track pin access, IO ports, and power and macro blockages – with realistic design rules offered in industry-standard input/output formats while keeping problem complexity tractable to academic researchers within the four-month contest timespan. However, even two full years after the initial release of the ISPD-2018 contest, there are only a few works [9][10][30][54][60][91] capable of delivering any kind of result; these results have nearly a thousand, if not thousands, of design rule check violations (DRCs) for nearly every testcase. Up until now, no work has come close to approaching the solution quality we expect from commercial detailed routers, although almost every work utilizes a variant of the five-decades-old path search algorithm.

Based on the ISPD-2018 Initial Detailed Routing contest, the present work describes TritonRoute, an open source detailed router for advanced VLSI technologies. Our main contribution is an end-to-end (i.e., complete, and with collaterals visible in a permissively open-sourced repository) detailed routing framework that aims and achieves beyond all existing academic detailed routers. Highlights of our work are summarized as follows.

- We propose an end-to-end detailed routing scheme. Our proposed scheme is capable of comprehending connectivity constraints (i.e., opens and shorts) and design rule constraints (i.e., spacing tables, end-of-line (EOL) spacing, minimum area and cut spacing).
- We build an in-memory router database that complies with LEF/DEF data models. This non-contest-driven code infrastructure enables future development and leverage of our open-source code towards deeper core optimization, more complete design rule support, and other enhancements.
- We present a number of key ideas in addition to the well-known A*-based path search. Transparency of our descriptions is aided by all implementation source codes being released under a permissive open source license.
- We evaluate our router using the official ISPD-2018 benchmark suite, and show that we reach an unprecedented, extremely low level of DRCs (< 20) in seven of 10 testcases, which is a 99.3% reduction of DRCs on average as compared to the known best detailed routing solutions from all published academic detailed routers. For the remaining three testcases, we reduce DRCs by 75.1% on average, and by 60.0% at a minimum. Overall, compared to the known best detailed routing solutions, TritonRoute improves wirelength by up to 0.8% (avg. 0.4%), via count by up to 16.1% (avg. 9.3%), and DRCs by up to 100% (avg. 92.0%).
- To the best of our knowledge, we are the first and the only open source gridded detailed router which is capable of delivering a DRC-clean detailed routing solution in sub-65nm technology nodes.

The remainder of this work is organized as follows. Section 4.2.1 provides a brief overview of previous works in the open literature. As noted above, such literature is sparse as far as it gives insight into

industry routing tools and how they address modern routing challenges. Section 4.2.2 presents our router database. Section 4.2.3 details our overall detailed routing flow. Section 4.2.4 presents our detailed routing methodology. Section 4.2.5 presents our experimental results using the official ISPD-2018 benchmark suite. Section 4.2.6 gives conclusions.

4.2.1 Related Work

As surveyed in [7], previous works on detailed routing can be categorized into fundamental and conventional algorithms, and recent developments. Further, we summarize the recent works targeting the ISPD-2018 initial detailed routing contest.

Fundamental and conventional algorithms. Lee [57] proposed the first maze routing algorithm, i.e., a breadth-first search that guarantees to find a minimum-cost path between two terminals if a path exists. Use of "best-first search", also known as A* search [78], sometimes in its bidirectional [83] form, enables maze-based search to focus itself toward desired targets, and reduces effort needed to find a minimum-cost feasible path. Hadlock [35] and Soukup [89] applied speedups to Lee's algorithm and others applied the line-search paradigm [43] to improve time and space efficiency as compared to Lee's and A* algorithms. Hetzel [42] developed a sequential routing approach using a shortest path algorithm with respect to euclidean distance. Specialized contexts such as channel routing [25] and switchbox routing [72], along with general frameworks such as multicommodity flow [88] and ripup-and-reroute [95], have respective sub-literatures and remain as fundamental building blocks of the detailed router today (cf. [27]).

Recent developments. More recent academic works on detailed routing focus on certain aspects of the modern routing challenge, mainly to address issues arising with advanced nodes. [59] gives an excellent summary of the academia-industry gap for detailed routing as of 2003; much of this gap remains today. Examples of focused recent works include Nieberg [77], which proposes techniques for gridless pin access in detailed routing. Xu [102] proposes pin-access planning and regular routing for self-aligned double patterning (SADP). The works of [16][20][26][68] address the detailed routing problem in an SADP process context. MANA [6] introduces an end-end separation and minimum wire length-aware shortest path algorithm. Han [38] develops a framework to reduce various DRCs in advanced nodes using

multicommodity flow-based integer-linear programming. BonnRoute [1][27] and RegularRoute [108] are two works prominent in the recent literature that present more complete portraits of overall detailed routing solutions.

ISPD contest-based works. Recently, a few works in the open literature attempt to address the gap between modern industrial designs and academic detailed routing flows, based on the ISPD-2018 initial detailed routing contest [73]. Sun [91] presents a multi-stage ripup-and-reroute flow for detailed routing. Kahng [54] proposes an integer linear programming (ILP)-based parallel intra-layer and sequential inter-layer routing flow. Chen [9][10] and Li [60] propose a detailed routing flow using min-area-captured path search on a sparse grid graph. Gonçalves et al. [29][30] propose a tunnel-aware A* lower bound, and a design-rule-aware path search algorithm for detailed routing. Although most recent works use correct-by-construction or safe-by-construction approaches to prevent DRCs, none of them is capable of delivering decent solution quality (that is, in a practical sense) due to the complexity of developing the necessary router infrastructure.

4.2.2 Database

Table 4.4: Database objects from LEF.

Object	**LEF Keyword**	**Meaning**
tech		back-end-of-line metal stacks
layer	LAYER	metal or cut layers
viadef	VIA	via definitions
constraint	WIDTH	default routing width
	AREA	minimum area rule
	SPACING	spacing rule
	SPACINGTABLE	spacing table rule
	MINIMUMCUT	minimum cut rule
	MINWIDTH	minimum width rule
	MINSTEP	minimum step rule
block	MACRO	standard or macro cells
term	PIN	standard or macro cell pin
blockage	OBS	standard or macro cell blockage
pin	PORT	physical pin
rect	RECT	rectangle
polygon	POLYGON	polygon

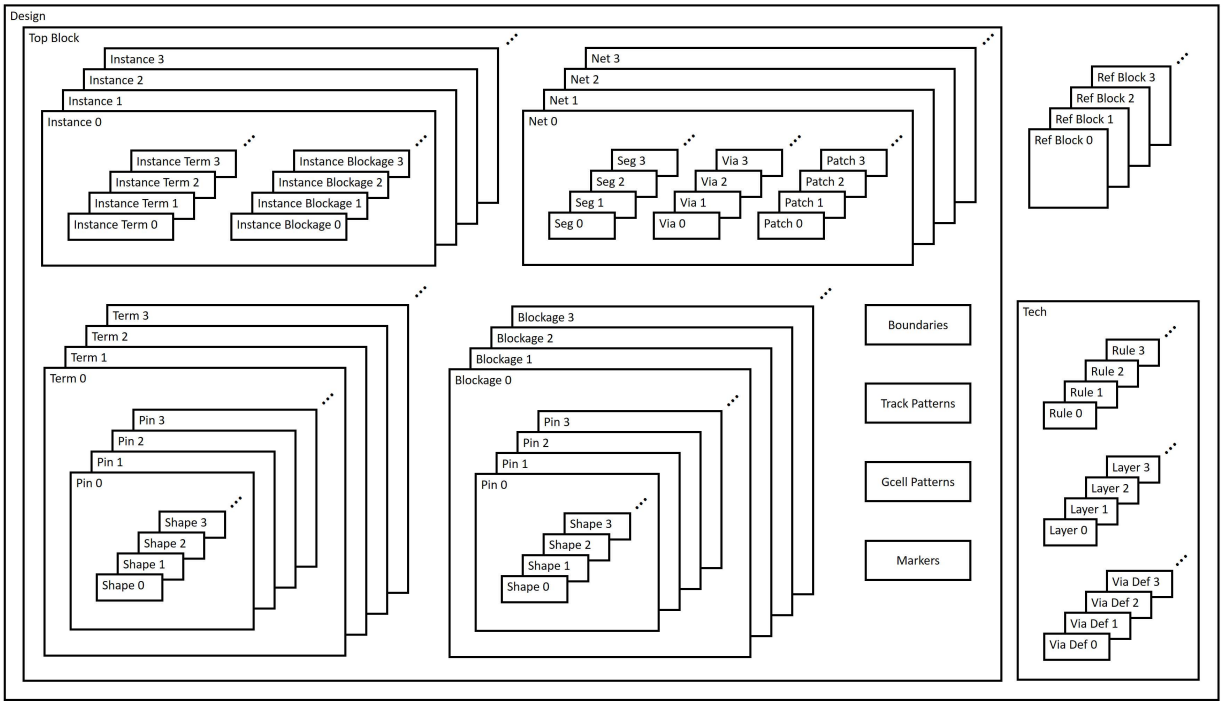


Figure 4.10: Major database structures.

In this section, we list all major objects and structures in the routing database. In building this database, we follow the LEF/DEF [119] data model, and reuse the naming convention from OpenAccess [128] as much as possible. The objects from LEF are summarized in Table 4.4, and the objects from DEF are summarized in Table 4.5. The structure of the database is described in Figure 4.10. The database is an in-memory, flattened physical design database. In the top level, the database consists of a **technology library**, a **top block** and several **reference blocks**.

Technology library

Technology library stores all metal and cut **layers**, **viadefs**, and design rule **constraints**. A back-of-end-stack layer consists of basic layer information, i.e., type, direction, pitch, offset, as well as all its applied design rule constraints. A viadef holds one or more **shapes** (rectangles or polygons) on two consecutive metal layers with shape(s) in the middle cut layer, realizing physical connection between neighboring metal layers at the same x-y coordinate. We summarize the design rules that we support in Table 4.6. For definitions, examples, and detailed handling methodology of each rule, please refer to [120].

Table 4.5: Database objects from DEF.

Object	DEF Keyword	Meaning
block	DESIGN	block-level design
inst	COMPONENTS	instance of standard or macro cell
term	PINS	block-level IO pin
blockage	BLOCKAGES	block-level blockage
net	SPECIALNETS	special net
	NETS	regular net
instTerm		points to a term
instBlockage		points to a blockage
pathSeg		routing segment
via		routing via
patchMetal		routing patch rectangle

Block

The top block describes the flattened logical and physical connections, following the DEF model. There are four major types of objects: **term**, **blockage**, **instance** and **net**. A reference block is a standard or macro cell from LEF, having the same data structure as the top block, except that only terms and blockages are populated.

Terms are IO pins for the top block, and standard or macro cell pins for the reference blocks. Each term consists of one or more physical **pins**. Each pin consists of one or more physical shapes across one or more metal and cut layers.[44]

Blockages are user-defined routing blockages from DEF BLOCKAGES for the top block, and are from LEF OBS statement for reference blocks. We reuse the pin object to hold physical shapes of the blockages.

Instances are from DEF COMPONENTS. Each instance is an instantiation of either a standard cell or a macro block, holding zero or more **instance terms** and **instance blockages**. An instance term points to the related term from its reference block. An instance blockage points to the related blockages from its reference block.

[44] A term including more than one pin with "MUSTJOIN" keyword indicates that the two pins should be physical connected in detailed routing. In this work, we assume that each term holds one physical pin to simplify the description.

Table 4.6: Design rules.

```
// metal layer
WIDTH defaultWidth ;
[MINWIDTH minWidth ;]
SPACINGTABLE
 PARALLELRUNLENGTH {length} ...
  {WIDTH width  {spacing} ...} ...  ;
[SPACING minSpacing SAMENET [PGONLY] ;]
[MINSTEP minStepLength [MAXEDGES maxEdges] ;]
[SPACING eolSpacing ENDOFLINE eolWidth WITHIN eolWithin
 [PARALLELEDGE parSpace WITHIN parWithin [TWOEDGES] ;] ...
// cut layer
{SPACING cutSpacing [CENTERTOCENTER]
 [  ADJACENTCUTS numCuts WITHIN cutWithin [EXCEPTSAMEPGNET]
  | PARALLELOVERLAP
  | AREA cutArea] ;}...
[SPACING cutSpacingSN [CENTERTOCENTER] SAMENET ;]
```

Nets are from DEF NETS and SPECIALNETS. A net stores its logical connections, and its physical connections, i.e., **pathSegs**, **vias** and **patchMetals**. A pathSeg is a point to point routing wire on a specific layer, defined with the start and end points, width and extensions. A via is an instantiation of viadef at a specific coordinate. A patchMetal is a patching rectangular metal used to satisfy various design rules.

Other types of objects in a block include **boundary**, **trackPattern**, **gcellPattern**, **marker**, etc. The gcellPattern object defines the global routing cells (GCells) [19] in 2D grids;[45] and marker object represents a design rule check (DRC) violation, including the bounding box, layer, violation type and source objects. In our implementation, we also build several assisting objects and structures. Some of the procedures are described in Section 4.2.3. A complete picture and details of the database implementation are visible at [126].

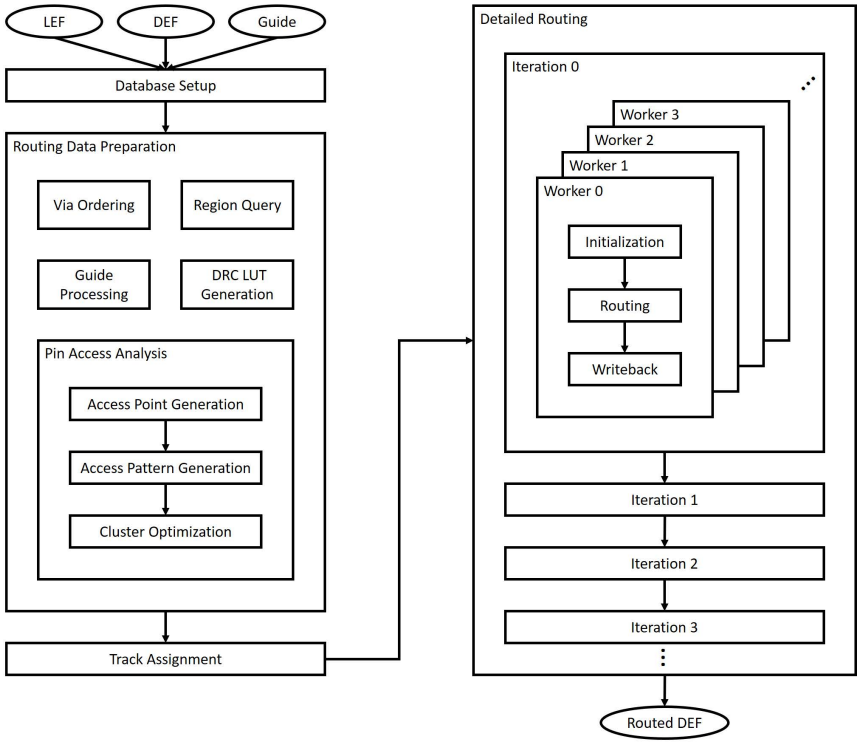


Figure 4.11: Overall flow.

4.2.3 Flow

In this section, we describe the detailed routing flow. As shown in Figure 4.11, the inputs to the router are LEF, DEF and guide files. LEF and DEF files are industry-standard formats. The route guide file serves as the global routing solution. Given the inputs, we first set up the design database. Next, we take several data preparation steps. Then, we perform track assignment, multiple iterations of detailed routing and output a routed DEF.

Data preparation

The data preparation step processes the design database to generate assisting structures, including via ordering, guide processing, region query, DRC LUT generation and pin access analysis.

Via ordering is the step to select default viadef(s) used for pin access and detailed routing. We sort all viadefs according to (i) number of cuts; (ii) default via property; (iii) enclosure direction; (iv)

[45] In our work, we derive the GCell size based on global routing solution, in the "route guide" format of ISPD18, ISPD19 and ICCAD19 contests. GR solutions in practice (to our knowledge) commonly use $\sim$15 M2 tracks as a typical GCell dimension.

enclosure area; and (v) enclosure width. In detailed routing, we only use the minimal-enclosed default single-cut viadef, with both lower and upper-layer enclosure along the preferred routing direction. In pin access analysis, in addition to the viadef we use in detail routing, we also use the minimal-enclosed default single-cut viadef, with the lower-layer enclosure orthogonal to the preferred routing direction, and the upper-layer enclosure along the preferred routing direction. Overall, we select one of two viadefs to access the pin, and only use one viadef for all other connections. Figure 4.12 illustrates the ordered viadefs for detailed routing, additional viadef for pin access analysis, and a non-preferred viadef.[46]

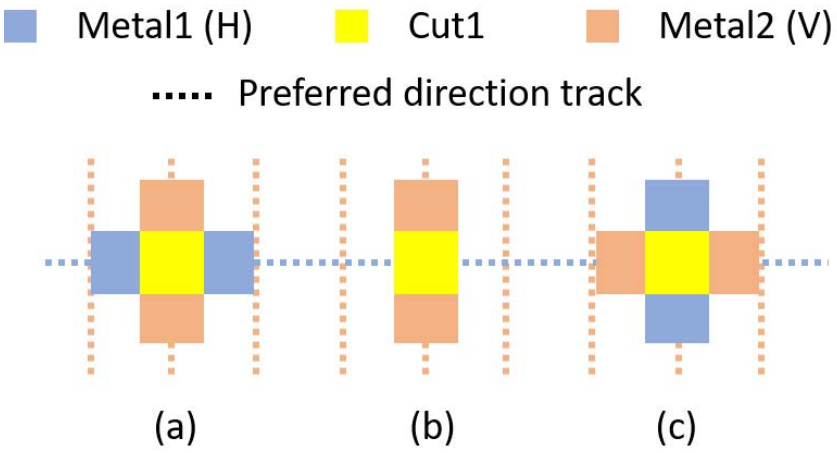


Figure 4.12: Illustrations of ordered viadefs: (a) preferred viadef for detail routing; (b) additional viadef for pin access analysis; and (c) non-preferred viadef.

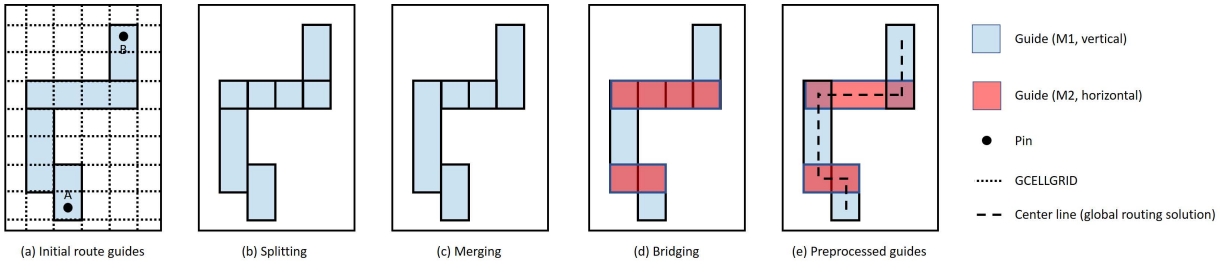


Figure 4.13: Preprocessing: (a) initial route guides; (b) splitting; (c) merging; (d) bridging; and (e) preprocessed guides. The preferred direction for M1 is vertical, and for M2 is horizontal.

[46] Ultimately, the via ordering step should be replaced with a more robust via generation and LEF matching strategy in a future work.

Guide processing [19] [54] is the step to transform a set of input route guides into a standardized tree-like global routing solution.[47] A route guide specifies a rectangular region on a specific metal layer. A global routing solution for a net may contain several route guides on some or all of the metal layers. If we abstract the guide by drawing a center line for each guide along the preferred routing direction, we take the center lines to form a connected graph, as shown in Figure 4.13(e).

To standardize on a guide dimension that is conducive to form a trimmed tree-like global routing solution, we first extract the most common offset and width of all guides to form GCELLGRIDS [19], then process all route guides with **splitting**, **merging** and **bridging** techniques. Given the input guides in Figure 4.13(a), we first split the guide according to the GCELLGRID along the preferred routing direction for each metal layer, as shown in Figure 4.13(b); then merge touching guides along the preferred routing direction, as shown in Figure 4.13(c). Last, for abutting guides along the non-preferred routing direction, we bridge them by creating upper-layer (or, otherwise, lower-layer) guides, as shown in Figure 4.13(d).

The above procedures guarantee a connected global routing solution as long as the input guides satisfy the assumption described in [19]. To remove redundant edges (i.e., loops) in a global routing solution, we further perform A* search from any pin to all other pins through the processed guides. All off-path guides are removed.

Region query is the data structure for fast shape queries. The inputs to the region query engine is a bounding box on a specific layer. The outputs are all intersecting shapes, in the form of {bbox, owner} pairs. For polygon shapes, we decompose the polygon into rectangles to be used in the region query engine. The owner belongs to one of the following types: term, instTerm, blockage, instBlockage, pathSeg, via or patchMetal.

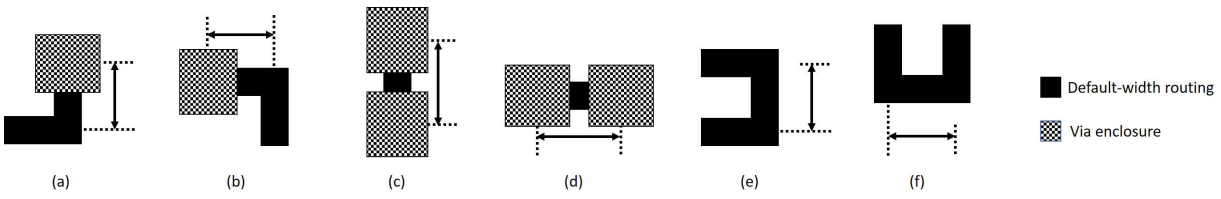


Figure 4.14: DRC LUT: (a) via to jog (vertical); (b) via to jog (horizontal); (c) via to via (vertical); (d) via to via (horizontal); (e) jog to jog (vertical); and (f) jog to jog (horizontal).

[47] Ultimately, the solution quality of detailed routing may be improved with an input of a better global routing solution that satisfies our guide processing behavior in a future work.

LUT generation is the step to construct assisting data structure to avoid same-net design rule check violations. In grid-based path search, we use object cost (described in Section 4.2.4) to avoid potential DRCs to existing objects. To prevent DRCs within the current path, i.e., same-net violation, we characterize the minimum default-width routing length between any two-object pair of an up via, a down via and a jog, on all metal layers, and in all directions. Figure 4.14 illustrates three types of minimum length requirement: via to jog, via to via, and jog to jog, in both x and y directions. In our implementation, we characterize separately for the up via and down via. In grid-based path search, we apply additional cost if the minimum length between vias and/or jogs is not satisfied.

The pin access analysis framework is described in Section 4.1.

Track assignment

We adopt a simplified version of greedy track assignment [97]. To reduce the problem size and lay a foundation for future parallel implementation, we perform the track assignment every 50 GCell panels. Each GCell panel has length along the preferred routing direction and spans 50 GCell heights. The initial track assignment is applied once on all horizontal layers, then on all vertical layers. According to [97], we then perform one iteration of track reassignment to optimize the solution quality.

Detailed Routing

Given the track assignment result, we perform multiple iterations of detailed routing. In each iteration, we partition the design into 7×7, non-overlapping GCell-aligned clips, and create one **detailed routing worker** for each clip. Each detailed routing worker first initializes its own data structures (worker database) from the global database, then performs routing and design rule checking, all without touching the global database. Last, each worker commits the changes by writing back to the global database. In alternate iterations, we shift the partitioning of 7×7 clips with an offset of 0 and -4 to enable optimization at clip boundaries. We describe the detailed routing flow inside the detailed routing worker in Section 4.2.4.

In the construction of a detailed routing worker, each clip comes with three bounding boxes: **standard**, **DRC** and **extended box**. The standard box is the above-mentioned 7×7, non-overlapping GCell-aligned clip. The detailed routing worker can only modify objects with their center lines on or

within the standard box. The DRC box is slightly larger than the standard box, enclosing the bounding box of all modifiable objects. We only count and writeback those markers intersecting with the DRC box. The extended box is slightly larger than the DRC box, allowing design rule check across the DRC box. In the detailed routing worker database, all objects within the extended box are constructed locally. Only the objects that are on or within the standard box are modifiable, while other objects are fixed. The fixed objects are used for cost calculation and design rule checking.

4.2.4 Detailed Routing Worker

In this section, we describe the methodology to perform gridded, A*-based detailed routing inside the detailed routing worker. We first describe the grid graph structure and various types of costs. Then, we describe the overall ripup-and-reroute flow of a detailed routing worker. Last, we detail the methodology to route one net.

Grid graph

The grid graph is an essential part of detailed routing because the path search algorithm works directly on the grid graph, and various costs and properties are associated with the grid vertices and edges in the grid graph. In TritonRoute, we build a non-regular-spaced 3D grid graph supporting **irregular tracks** and **off-track routing**.

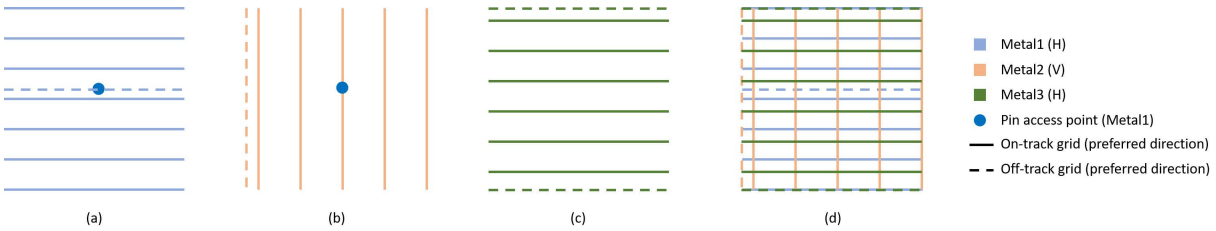


Figure 4.15: Grid graph: (a) preferred-direction grid lines on Metal1; (b) preferred-direction grid lines on Metal2; (c) preferred-direction grid lines on Metal3; and (d) overlay of grid lines (3D grid graph projected onto the x-y plane).

Construction. We now describe how to generate the preferred-direction grid lines on each metal layer. We first form all grid lines that are on-track – i.e., align with the DEF TRACKS definitions. Then we form all grid lines that are off-track – i.e., the center lines along the preferred direction for any existing pathSegs, vias and pin access points. We also form the grid lines on the boundary. We do not generate the grid lines in the non-preferred direction. However, bi-directional routing is still available as described later.

Figure 4.15 shows how we form the grid lines. Figure 4.15(a) shows horizontal Metal1, with 7 regular-spaced tracks from DEF. The Metal1 pin has an access point with an off-track y-coordinate. Thus, we create an off-track grid line according to the pin access point location. Figure 4.15(b) shows vertical Metal2, with 5 regular-spaced tracks from DEF. We additionally create an off-track grid on the left boundary. By always creating grid lines along the boundaries of the routing region, we make sure that at least one path exists in the grid graph in any direction, in the case that no on- and off-track grid lines exist (e.g., given a small routing region). Since the center line of the Metal1 pin access point aligns with a Metal2 track, we do not build additional off-track grid lines on Metal2. Similarly, we build grid lines on Metal3. Note that Metal1 and Metal3 grid lines do not necessarily align.

In Figure 4.15(d), we show the overlay of x- and y-direction grid lines. The grid vertices are formed by intersecting all x- and y-direction grid lines, and repeating $|Z|$ times along the z-direction. Each vertex has six neighbors (except the boundary vertices) – west, east, south, north, down and up; this is the 3D grid (projected onto the *x*-*y* plane) that we use in TritonRoute.

Table 4.7: Edge properties.

Type	**Name**	**Meaning**
boolean	isEnable	whether the edge exists in path search
boolean	isOnTrack	whether the edge is on track
boolean	isOnPrefDir	whether the edge is on the preferred direction
viadef	specialVia	special via
int	objCost	object cost
int	markerCost	marker cost

Edge. The edge properties are summarized in Table 4.7. As shown in Figure 4.15, not every grid line exists in every metal layer. We use *isEnable* to show whether the edge exists in the path search. A

planar edge in the preferred direction is enabled if it is on a current layer grid line. A planar edge in the non-preferred direction is enabled if it is on an upper-layer grid line (if any, otherwise lower-layer). Via edges are enabled between any two preferred-direction grid lines on neighboring metal layers. For each edge, we use *isOnTrack* to show whether the edge is on track; we use *isOnPrefDir* to show whether the edge is on the preferred direction. For a via edge, *specialVia* indicates whether the router should choose a special via instead of the default via. Only pin access points may have this special via property. We preprocess and mark relevant via edges for all up-via pin accesses (using non-default via). There are two types of costs associated with each edge, object cost and marker cost. We describe these costs in Section 4.2.4.

Table 4.8: Vertex properties.

Type	**Name**	**Meaning**
enum	prevDir	incoming direction
boolean	isSrc	whether the vertex is the source
boolean	isDst	whether the vertex is the destination

Vertex. The vertex properties are summarized in Table 4.8. In A*-based path search, after a path is found, we only know the ending vertex. We use *prevDir* to indicate the incoming direction of the current vertex so that we are able to trace back the path. We use *isSrc* (resp. *isDst*) to indicate whether the vertex is a source (resp. destination).

Routing Cost

We use two types of costs: **object cost**, and **marker cost**. Overall, object cost is applied around an existing shape. This cost preemptively guides the path search to go around existing objects to avoid potential DRCs. The marker cost is applied around an existing DRC marker. In the ripup-and-reroute scheme, this cost helps the nets to be routed avoiding the DRC hotspots given the history of DRC data.

Object cost is the cost originated from an object, and stored in neighboring edges to the object. We modify this cost whenever the worker database adds or removes an object, e.g., at the time of database initialization, after net ripup, or after routing of one net. We use the object cost to prevent

potential design rule check violations. The evaluation of object cost is non-precise but quick, and does not invoke the DRC engine.[48] We support three types of spacing rules for object cost: (i) SPACINGTABLE PARALLELRUNLENGTH; (ii) SPACING ENDOFLINE; and (iii) SPACING (cut).

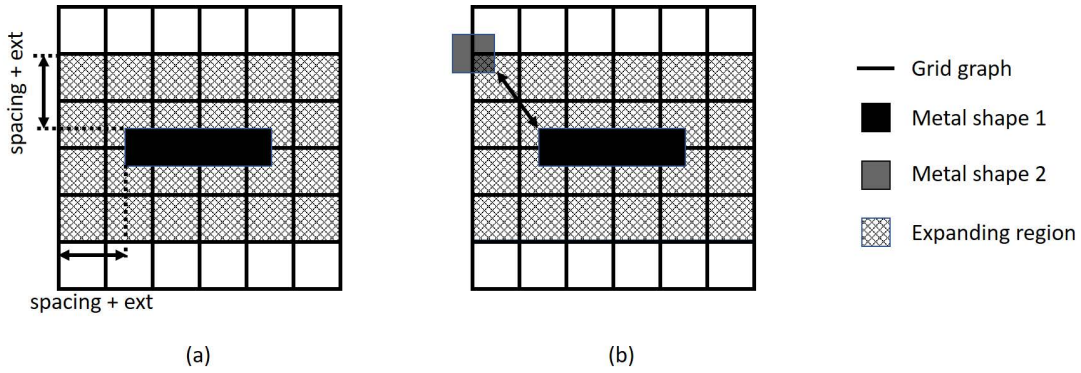


Figure 4.16: Object cost from parallel run length spacing: (a) expanding region; and (b) shadow object.

For parallel run length spacing, given a **target object**, we first draw an expanding region in which objects on the intersecting edges may cause DRCs, as shown in Figure 4.16(a). The expanding region extends beyond the target object up to the maximum required spacing plus half the default width for planar edges, and half the via enclosure for via edges. We then assume a **shadow object** (either a default-width pathSeg or a via) on each of the neighboring planar and via edges, and check against the target object, as shown in Figure 4.16(b). For a pathSeg on a planar edge, since the exact length of the shadow object can be arbitrarily longer than the edge length, we add pessimism by assuming maximum parallel run length between the two objects to accelerate convergence. The maximum parallel run length is the length of the target object regardless of the actual parallel run length. For each via edge, we assume a default via, or the special via stored with the edge, and check the via enclosure against the target object. The parallel run length between a shadow via enclosure and the target object is calculated by their actual parallel run length. We modify the cost of the edge if there is a violation. Here, the modification of the costs also helps to avoid short violations since the expansion region implicitly includes those edges that may have potential short violations with the target object.

[48] We do not have a metric for "precision" of object cost evaluation. The goals of the quick object cost evaluation, in decreasing priority order, are: (i) quickness, and (ii) help avoidance of repeated cycles of violations (e.g., arising due to DRC marker cost in A* search). In practice, we see that our use of quick object cost evaluation – which naturally must be pessimistic – helps avoid cycling.

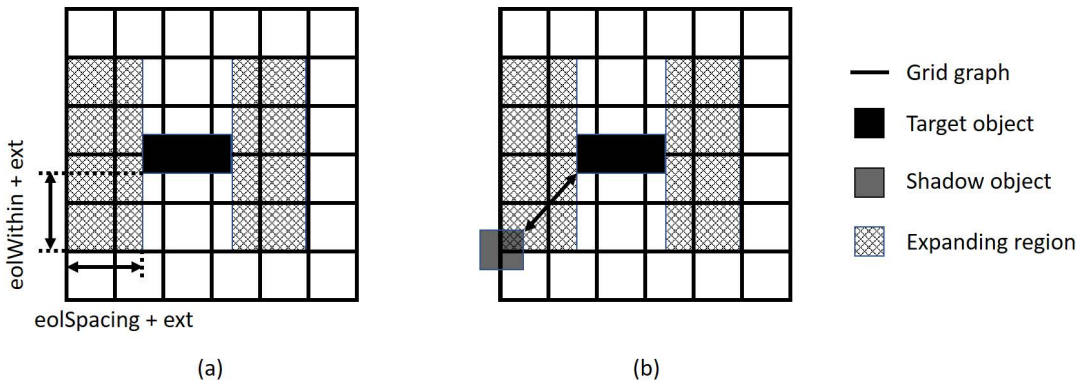


Figure 4.17: Object cost from end-of-line spacing: (a) expanding region; and (b) shadow object. The preferred routing direction is horizontal.

For end-of-line spacing, we only check the target object if it is a via, and the spacing is only checked along the preferred routing direction of the metal layer. Spacing orthogonal to the preferred routing direction is not checked to avoid pessimism since almost all jogs end with a preferred-direction routing or a default via, making the line end a non-end-of-line edge. Figure 4.17 illustrates the procedure.

For cut spacing, given a target via, we check all neighboring via edges which could potentially cause a cut spacing violation. For each via edge, we assume a default via (or the special via stored with the edge) and check against the target via. We modify the cost of the via edge if there is a violation.

The object cost has no history. For example, an object cost is added to the neighboring edges of the target object after the object is created, and subtracted from the neighboring edges of the target object after the object is removed. The object cost calculation supports same-net overriding, blockage spacing overriding and other exceptions. For more details pertaining to this and other parts of our discussion, please refer to [126].

Marker cost is the cost applied according to the DRC markers after each call to the DRC engine. For each marker, we get all objects touching the marker, and add costs to the nearest edge(s) that are used to form the objects. The marker cost has history within the detailed routing worker. For example, a marker cost is added to an edge and decayed over time ($currIter$ in Algorithm 13), but is never subtracted due to the removal of a specific marker. Here, marker cost history only persists within the detailed routing worker. There is no history between detailed routing iterations shown in Figure 4.11.

Routing flow

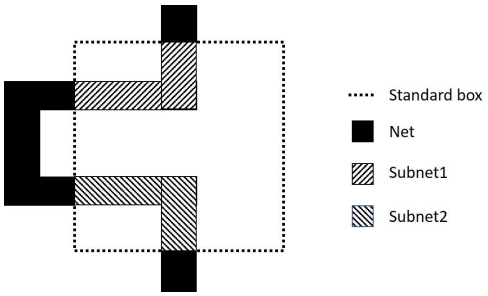


Figure 4.18: Local netlist construction: two disjoint subnets constructed in the detailed routing worker from one global net.

Now we describe the routing flow inside a detailed routing worker. In Algorithm 13, Line 2 first initializes the worker database from the global database. In this step, we construct a local netlist from the connectivity of routing objects. Figure 4.18 shows an example, where a single net passes through the standard box twice, with two parts disjoint. In this case, we construct two subnets so that ripup-and-reroute does not change the connectivity of the net.

In Lines 3 – 20, we perform up to *maxIter* iterations of ripup-and-rerouting.[49] In each iteration, we ripup the problematic nets and reroute each one sequentially. Line 4 adds the marker cost according to all existing markers. Line 5 gets all nets that are associated with markers. We order the nets according to their distance to the nearest marker and route them sequentially. Line 6 rips up those nets and Line 7 subtracts the object cost from the ripped-up objects. Here, the boundary objects outside the standard box are not removed and their object costs remain. Since nets are routed sequentially, according to the net ordering, we would like to avoid the i^{th} net blocking the pin access of the $j^{th}(j > i)$ net. In Line 8, we reserve the pin access of all unrouted nets (ripped-up nets) by adding the object cost of their preferred pin access (an up via) as if those pin access points are used.

In Lines 9 – 15, we route each net once according to the net ordering. Before routing, Line 10 unreserves the pin access for the current net by subtracting the corresponding object cost of the preferred pin access (up via). Line 11 subtracts the object cost for the boundary objects outside the standard box to

[49]Note that this number of iterations is different from the number of "outer" iterations in Figure 4.11. For the results that we report in this work, we perform seven (outer) iterations. The *maxIter* number of iterations in Algorithm 13 defines the maximum number of ripup-and-reroute iterations a net inside a DRWorker can undergo. In the current implementation/results represented in this work, we use (1, 4, 4, 4, 4, 4, 4) as the maxIter (for ripup-and-reroute) for each net in the seven "outer" iterations, respectively.

avoid unnecessary costs when we connect the net to the boundary pin. Line 12 routes the current net. Line 13 adds the object cost for all the newly routed objects. Line 14 adds back the object cost for boundary objects to prevent design rule violations between these objects to the remaining unrouted nets. Lines 16 – 19 perform design rule checking, and terminates the ripup-and-reroute flow once the clip is clean.

Line 21 commits the worker database back to the global database.

Algorithm 13 Routing flow

```
1:  Input: worker database, worker markers markers
2:  WorkerDBInit()
3:  while currIter < maxIter do
4:      addMarkerCost(markers)
5:      nets ← getMarkeredNets(markers)
6:      ripupNets(nets)
7:      subObjCost(nets)
8:      reservePA(nets)
9:      for all net ∈ nets do
10:         unreservePinAccess(net)
11:         subBoundCost(net)
12:         routeOneNet(net)
13:         addObjCost(net)
14:         addBoundCost(net)
15:     end for
16:     DRC(nets)
17:     if numMarkers = 0 then
18:         break
19:     end if
20: end while
21: DBCommit()
```

Routing one net

Flow. We now describe the methodology to route one net in a detailed routing worker. In our current implementation, in the standard box, a net is either fully routed or unrouted, but not partial routed. Algorithm 14 describes the methodology to route one net. Line 2 gets all unconnected pins, including standard box boundary pins and pins from instTerm and term. Line 3 holds the set of visited grid vertices, and we initialize the set to be empty. Lines 4 and 5 select the source pin to perform path search and remove it from the unconnected pins. To select the source pin, we first calculate the center of gravity for all pins in

the x-y plane, then select the pin furthest away from the center of gravity as the source. Line 6 performs the initialization described later. In Lines 7 – 11, we perform the path search as long as there are still unconnected pins. After path search, we update the grid graph in preparation for the next round of path search. The writeDB function backtraces the path to create the routing objects according to the path.

Algorithm 14 Route one net

```
1:  Input: net n, grid graph G
2:  unConnPins ← allPins(n)
3:  visitedGrids ← ∅
4:  srcPin ← selectSrcPin(unConnPins)
5:  unConnPins.removePin(srcPin)
6:  init(n, srcPin, unConnPins, visitedGrids, G)
7:  while not isEmpty(unConnPins) do
8:      path ← search(visitedGrid, G)
9:      update(n, path, unConnPins, visitedGrids, G)
10:     writeDB(n, path)
11: end while
```

During backtracing, we calculate the total metal area and add necessary patch metals to satisfy the minimum area rule. The patch metals are always created with default routing width along the preferred routing direction. In our implementation, we also build assisting structures to calculate necessary patch metal area for objects connected to the boundary pin. Figure 4.19 gives two examples of patch metal addition. We assume the preferred routing direction is horizontal. We do not allow the patch metal to exceed the standard box. If there are more than one patch metal choices, e.g., adding to the left or to the right of a routing object, we choose the one with smaller object cost. The path search is completed once all pins are connected. The path search algorithm is described in Algorithm 16. The update function is described in Algorithm 17.

Initialization. Algorithm 15 describes the initialization procedure. In Line 2, we first reset the previous direction flag for each grid vertex. In Lines 3 – 6, we set the source flag for all vertices on the access points of the source pin, and add the vertices to the visited grids. In Lines 7 – 11, we set the destination flag for all vertices on the access points of all destination pins. After initialization of the grid graph, the core path search algorithm does not need to look for objects and properties of the net, which is beneficial to the runtime.

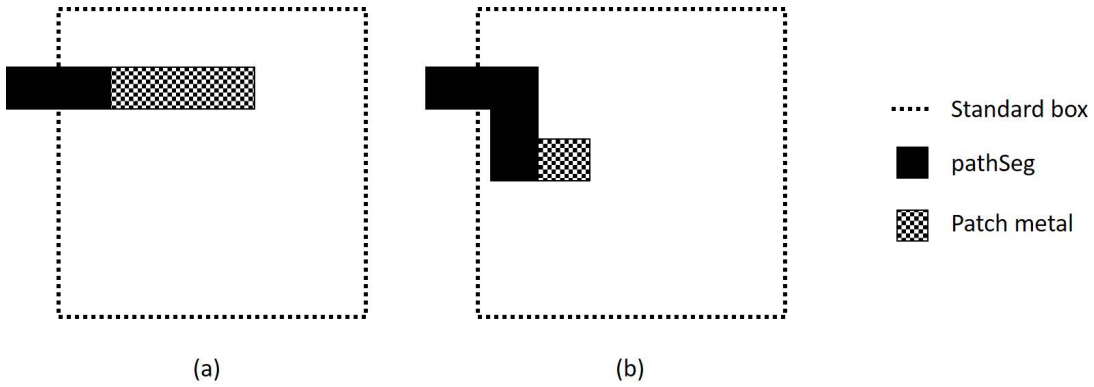


Figure 4.19: Minimum area patch metal: (a) patch metal considering area outside of standard box; and (b) patch metal always along the preferred routing direction even if the routing ends in the non-preferred direction.

Algorithm 15 Initialization

```
1:  Input: n, srcPin, unConnPins, visitedGrids, G
2:  G.resetPrevDir()
3:  for all grid ∈ srcPin do
4:      G.setSrc(grid)
5:      visitedGrids.add(grid)
6:  end for
7:  for all dstPin ∈ unConnPins do
8:      for all grid ∈ dstPin do
9:          G.setDst(grid)
10:     end for
11: end for
```

Path search. Algorithm 16 details the path search. The A*-based path search is based on a priority queue. Each element in the priority queue is an element of the search's wavefront, representing that a path exists from the source up to the wavefront grid vertex. In Lines 3 – 5, we first push all visited grids (source) to the queue as the initial wavefront vertices. Then in Lines 6 – 16, we pop the wavefront vertex with the least cost. We use the previous direction to indicate whether the wavefront vertex has been visited before. Lines 9 – 11 skip the wavefront vertex if it has been visited before. In Lines 12 – 14, we check whether the wavefront vertex is the destination, and return the path when reaching the destination. Otherwise, we expand the wavefront vertex by pushing its neighbors into the priority queue (with proper cost) as new wavefront vertices.

Algorithm 16 Search

```
1: Input: visitedGrids, G
2: Initialize wf
3: for all grid∈visitedGrids do
4:    wf.push(grid)
5: end for
6: while not isEmpty(wf) do
7:    currGrid ← wf.top()
8:    wf.pop()
9:    if hasPrevDir(currGrid) then
10:      continue
11:   end if
12:   if isDst(currGrid) then
13:      return path
14:   else
15:      expand(currGrid)
16:   end if
17: end while
```

Here, the cost in the priority queue is the A* cost, consisting of an existing path cost and an estimated future cost, as shown in Equation (4.1). Whenever we expand from a wavefront vertex to its neighboring vertex, the existing cost is the cost from the wavefront vertex plus the cost to its neighbor, as shown in Equation (4.2). The cost is the sum of edge length, plus 8× edge length if the edge has a non-zero object cost, and 64× edge length if the edge has a non-zero marker cost. In addition, we apply a penalty p if any match to the DRC LUT is found. The estimated future cost is the Manhattan distance to a pre-determined destination, as shown in Equation (4.3). If there are more than one unconnected pins to be connected, the pre-determined destination is the bounding box of the unconnected pin that is the closest to the bounding box of all visited grids. The Manhattan distance in z-direction (between two neighboring metal layers) is calculated as 4× the lower metal layer pitch.

$$cost_{tot} = cost_{wf'} + cost_{est} \tag{4.1}$$

$$cost_{wf'} = cost_{wf} + len_e + objCost_e + markerCost_e + p \tag{4.2}$$

$$cost_{est} = dist_{wf',dst} \tag{4.3}$$

As described in Lines 9 – 11, we avoid expanding an already-visited vertex by checking its previous directional flag. In an ideal A*-based path search with a consistent path cost and a lower-bounded estimated future cost, each vertex only needs at most one visit to get the minimum cost path. However, considering the inconsistent nature of the penalty applied from the DRC LUT, the worst-case complexity of A*-based path search becomes $O(n^2)$. To balance the tradeoff between runtime and solution quality, we write the previous direction to a vertex only after two more wavefront expansions are performed from that vertex.

Update. Algorithm 17 describes the methodology to update the grid graph. In Line 2, we reset the previous direction flag for every grid vertex in preparation of the next path search. In Lines 3 – 6, we set the source flag for every grid vertex along the path. We then add these grid vertices to the visited grids. Here the source flag and the visited grids serve the same purpose as they both identify the new sources for the next round of path search. However, visited grids are stored in a vector-like container to allow us to initialize the wavefront for the next path search in batches. In Lines 7 – 15, we identify the destination pin that we route to in the current round of path search, remove it from the unconnected pins, and reset the destination flag on all access points of the destination pin.

We now describe two special cases for **pin feedthrough**. Pin feedthrough describes a scenario where two (or multiple) parts of the net are connected to different access points of the same pin. We can either enable, or disable pin feedthrough. Disabling pin feedthrough forces that only one access point per pin can be used.

In case of enabling feedthrough, all access points of the destination pin, even those we do not route to, now become new sources for the next round of path search, as shown in Lines 12 –14.

In case of disabling feedthrough, special handling methodology is needed for the first source pin of the net, described in Lines 17 – 24. Recall that in Line 4 of Algorithm 15, we set the source flag on all access points of the source pin. Given feedthrough disabled, we must reset the source flag on all unused access points of the source pin once the first path search completes.

4.2.5 Experiments

In this section, we present experimental setup and results.

Algorithm 17 Update

```
1: Input: n, path, unConnPins, visitedGrids, G
2: G.resetPrevDir()
3: for all grid ∈ path do
4:     setSrc(grid)
5:     visitedGrids ← add(grid)
6: end for
7: endGrid ← path.end()
8: currDstPin ← findPin(endGrid)
9: unConnPins.removePin(currDstPin)
10: for all grid ∈ currDstPin do
11:     G.resetDst(grid)
12:     if isAllowPinAsFeedThrough() then
13:         G.setSrc(grid)
14:     end if
15: end for
16: beginGrid ← path.begin()
17: if not isAllowPinAsFeedThrough() then
18:     if findPin(beginGrid) then
19:         currSrcPin ← findPin(beginGrid)
20:         for all grid ≠ beginGrid ∈ currSrcPin do
21:             G.resetSrc(g)
22:         end for
23:     end if
24: end if
```

Setup

We implement our router in C++ with LEF/DEF parser [119] and Boost C++ libraries. We perform experiments using the ISPD-2018 benchmark suite [73]. The ISPD-2018 benchmark suite provides 10 testcases in $45nm$ and $32nm$ nodes, with up to 290K standard cells and 182K nets. These designs are industrial benchmarks – including large memory cells, off-track pin access, IO ports, and power and macro blockages – with realistic design rules offered in industry-standard input/output formats. ISPD-2018 benchmark information is summarized in Table 4.1.

The ISPD-2018 contest evaluation metrics consist of three components: (i) routing, including wirelength and via count; (ii) guides and tracks obedience, including out-of-guide wire and vias, off-track wire and vias, and wrong-way wire; and (iii) DRCs, including area of metal shorts, number of minimum

area violations and number of spacing violations. However, in the experimental results below, we do not report (ii), and make several improvements to (iii) according to the following.

- We do not strictly obey the guides since TritonRoute is not targeting the ISPD-2018 contest. According to the contest organizers, strict guide obedience was never their initial intention although all participating teams and the following published papers all strictly follow the route guides.
- We do not report the off-track and wrong-way routing although they are already considered throughout the routing flow. In all our reported testcases, such off-track and wrong-way routing account for 0.68% of the total wirelength on average.
- We report all types of DRCs, including all ISPD-2018 centric DRCs plus (number of) metal short, non-sufficient metal overlap and minimum width. The number of metal short is a good indicator of the strength of the detailed router. Non-sufficient metal overlap and minimum width are two design rules existing in the input, but not considered in the contest evaluation. We believe that the reporting of all types of DRCs effectively forbids any optimization targeting the contest metric.

Among all recently published academic detailed routers [30][60][91] that are capable of delivering ISPD-2018 contest solutions, Dr. CU dominates the solution quality for all ten testcases in terms of DRCs. Thus, we compare our TritonRoute to Dr. CU 2.0. All experiments are performed using a single thread on an Intel Xeon server.

Results

Experimental results are shown in Table 4.9 and Table 4.10. Table 4.9 gives wirelength, via count, memory consumption and runtime; Table 4.10 gives the details of DRCs.

As a prerequisite, a routing solution is valid only if there are no open nets. All of our reported solutions meet the connectivity requirement. Furthermore, our solution guarantees a loop-free and dangling wire-free solution (except the minimum area patch metals).

We achieve DRC-clean solution for *ispd18_test1*, and reach an unprecedented, extremely low level of DRCs (< 20) in seven of 10 testcases while consuming substantially reduced memory, with similar

single-threaded runtime. This translates to a 99.3% reduction of DRCs as compared to known best detailed routing solutions from all published academic detailed routers. For the remaining three testcases, we reduce DRCs by 75.1% on average, and by 60.0% at a minimum. Overall, compared to the known best detailed routing solutions, TritonRoute improves wirelength by up to 0.8% (avg. 0.4%), via count by up to 16.1% (avg. 9.3%), and DRCs by up to 100% (avg. 92.0%). TritonRoute completes routing with smaller wirelength and smaller via count, and leaves only a fraction of DRCs compared to all other academic detailed routers.

Table 4.9: Comparison of total wirelength, total via count, memory usage and runtime between TritonRoute (column A) and Dr. CU (column B).

Benchmark	**Wirelength** (μm)		**Via count**		**Memory** (GB)		**Runtime** (s)	
	TR	CU	TR	CU	TR	CU	TR	CU
ispd18_test1	**86025**	86709	32912	**32402**	**0.08**	0.21	61	**40**
ispd18_test2	1570651	**1566537**	**319855**	325684	**0.43**	1.39	614	**578**
ispd18_test3	1750028	**1743561**	319456	**318309**	**0.47**	1.51	824	**788**
ispd18_test4	**2620890**	2641860	**695901**	729312	**1.09**	5.72	**1866**	3422
ispd18_test5	**2763186**	2780130	**831775**	965544	**1.29**	4.61	**1722**	2383
ispd18_test6	**3557744**	3570351	**1241673**	1480617	**1.71**	5.72	**2682**	3357
ispd18_test7	**6482066**	6517341	**2041794**	2402543	**3.07**	9.87	**5023**	5847
ispd18_test8	**6513278**	6546908	**2062997**	2412121	**3.11**	10.47	**4916**	5932
ispd18_test9	**5442527**	5476029	**2049839**	2410790	**2.71**	10.11	**4378**	4910
ispd18_test10	**6769942**	6809019	**2226243**	2594386	**3.09**	10.58	10129	**9380**

We have also performed a case-study experiment using different standard box sizes to analyze the tradeoff between runtime and final DRC count. We sweep the standard box size from 3×3 to 11×11 with a step size of 2 on the *ispd18_test3* testcase. The specific testcase that we choose has relatively high *#violation-to-#instance* ratio, which indicates that *ispd18_test3* is a difficult and congested design among the ISPD18 contest benchmarks. Figure 4.20 illustrates the tradeoff between runtime and final DRC count with different standard box sizes. We observe that a larger standard box provides a larger solution space for ripup-and-reroute for DRC fixing at the cost of longer runtime for A* search. A standard box with size of 7×7 GCells can achieve a decent tradeoff between runtime and final DRC count, especially for difficult designs.

Table 4.10: Comparison of number of minimum width (MinWid), non-sufficient-metal overlap (NSMet), minimum area (MAR), metal short (Short), cut short (CShort), metal parallel run length spacing (MetSpc), metal end-of-line spacing (EOLSpc), cut spacing (CutSpc) and total design rule violations between TritonRoute (TR) and Dr. CU (CU).

Benchmark	Design rule violations																	
	#MinWid		#NSMet		#MAR		#Short		#CShort		#MetSpc		#EOLSpc		#CutSpc		#Total	
	TR	CU	TR	CU	TR	CU	TR	CU	TR	CU	TR	CU	TR	CU	TR	CU	TR	CU
ispd18_test1	**0**	0	**0**	1716	**0**	0	**0**	1	**0**	0	**0**	1	**0**	1	**0**	0	**0**	1719
ispd18_test2	**0**	0	**0**	20048	**0**	0	**1**	1	**0**	0	**7**	49	**9**	9	**0**	0	**17**	20107
ispd18_test3	**0**	0	**0**	21224	**0**	0	**112**	219	1	**0**	**17**	86	10	**9**	2	**0**	**142**	21538
ispd18_test4	**0**	10	**2**	17	**0**	32	**190**	287	**0**	0	**132**	289	**2**	164	**0**	142	**326**	941
ispd18_test5	**0**	7	**0**	19	**0**	48	**2**	342	**0**	0	**0**	309	**0**	36	**0**	20	**2**	781
ispd18_test6	**0**	8	**0**	44	**3**	92	**1**	36	**0**	0	**2**	489	**2**	21	**0**	30	**8**	720
ispd18_test7	**0**	0	**0**	11	**5**	127	**4**	604	**0**	0	**4**	129	**0**	7	**0**	60	**13**	938
ispd18_test8	**0**	0	**0**	19	**3**	138	**2**	625	**0**	0	**1**	118	**0**	15	**0**	59	**6**	974
ispd18_test9	**0**	0	**0**	16	**4**	185	**1**	39	**0**	0	**0**	49	**0**	7	**0**	54	**5**	350
ispd18_test10	**0**	0	**0**	26	**4**	228	**1103**	3180	5	**1**	**425**	742	144	**73**	**33**	100	**1714**	4350

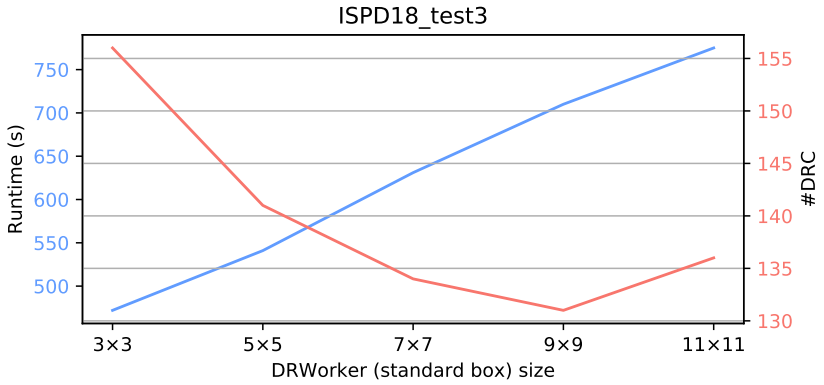


Figure 4.20: Illustration of tradeoff between runtime and final DRC count with various DRWorker standard box sizes in unit of GCell.

4.2.6 Conclusion

In this work, we present TritonRoute, an open source detailed router. We describe an in-memory router database, and an end-to-end detailed routing scheme. We evaluate our router using the official ISPD-2018 benchmark suite, and show that we reach an unprecedented, extremely low level of DRCs (< 20) in seven of 10 testcases, a 99.3% reduction of DRCs on average as compared to the known best detailed routing solutions from all published academic detailed routers. Overall, compared to the known best detailed routing solutions, TritonRoute improves wirelength by up to 0.8% (avg. 0.4%), via count by up to 16.1% (avg. 9.3%), and DRCs by up to 100% (avg. 92.0%).

4.3 Acknowledgments

Chapter 4 contains reprints of Andrew B. Kahng, Lutong Wang and Bangqi Xu, “The Tao of PAO: Anatomy of a Pin Access Oracle for Detailed Routing”, *Proc. ACM/IEEE Design Automation Conference*, 2020. Chapter 4 also contains the draft of Andrew B. Kahng, Lutong Wang and Bangqi Xu, “TritonRoute: The Open Source Detailed Router”, in submission to *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2020. The dissertation author is a main contributor to, and a primary author of, each of these papers.

I would like to thank my coauthors Andrew B. Kahng and Bangqi Xu for their support and work.

Chapter 5

Conclusion

This thesis has presented several physical design methodologies and optimizations to address existing and future challenges in advanced VLSI. The presented methodologies and optimizations help to realize manufacturing-aware design technology co-optimizations, advanced node design-based equivalent scaling, and an open-source academic detailed routing flow.

Chapter 2 has presented two general flow optimizations in physical design. First, we have presented a novel flop tray-based optimization for improved design power reduction. We propose a capacitated K-means algorithm which iteratively applies a min-cost flow-based clustering and a LP-based flop tray placement. We also propose an ILP-based matching optimization to generate flop trays while minimizing the perturbation to the initial placement solution. Our work achieves several improvements as compared to previous works: (i) awareness of flop tray aspect ratio and (large) size; (ii) explicit minimization of relative displacement of timing-critical start-end flop pairs; and (iii) global optimization instead of local search. The proposed techniques allow us to achieve up to 32% total block power reduction as compared to designs with only single-bit flops, and up to 16% total block power reduction over designs with flop trays generated by logical clustering during synthesis. We also achieve 13% clock power reduction on average as compared to the previous work in [48]. We further study the impact of flop tray sizes on optimization solution quality, as well as the useful skew optimization in the context of our flop tray-based designs. Second, we have presented a scalable MILP-based optimization of 2D block masks that considers block mask rules,

minimum metal density constraints, and timing impact of dummy fills. We propose an improved timing impact model for use in our MILP formulation. A distributed optimization flow enables application of the MILP-based optimization to large design layouts. We evaluate our approach across timing-awareness, different patterning technologies, and different minimum metal density constraints. Our study shows up to 84% ΔWNS recovery, up to 85% ΔTNS recovery, and up to 56% Δswitching power recovery, along with up to 62% dummy removal rate. We believe that our enablement of a timing-aware optimization shows promising product-level benefits from use of 2D block masks, and furthermore sheds light on the merits of various block mask optimization objectives. We have also studied the *co-optimization* of cut and block masks. Our cut and block co-optimization opens up a broader solution space, with more flexibility in EOL realization and attendant design quality benefits.

Chapter 3 has presented two improved physical design methodologies in placement. First, we have presented an *optimal* dynamic programming-based single-/double-row detailed placement methodology to minimize diffusion *steps* in sub-10nm VLSI, for improved yield and mitigation of NDE. Our work achieves several improvements as compared to previous works: (i) optimal dynamic programming with support of a richer set of cell movements, i.e., flipping, relocating and enhanced reordering; (ii) optimal double-row dynamic programming *with support of movable and reorderable double-height cells*; and (iii) a novel performance improvement technique using *intentional steps*. The proposed techniques achieve up to 98% reduction of inter-cell diffusion *steps*, with scalable runtime and high die utilization in an N7 node enablement. Second, we have presented a vertical M1 routing-aware detailed placement optimization based on mixed-integer linear programming (MILP) for two new cell architectures in sub-10nm nodes, i.e., *ClosedM1* and *OpenM1*. With our optimization, up to 6.4% (resp. 2.2%) total routed wirelength reductions and 14.4% (resp. 4.1%) #via12 reductions are achieved for *ClosedM1*-based (resp. *OpenM1*-based) designs, with no adverse timing impact.

Chapter 4 has presented two works towards an open source detailed router. First, we have presented a multi-level, standard cell- and instance-based, complete, robust, scalable and design rule-aware pin access analysis framework. We describe our robust pin-based access point generation, boundary conflict-aware access pattern generation and cluster-based access pattern selection based on dynamic programming. We achieve 100% DRC-clean pin access and demonstrate a superior final detailed routing solution as compared

to the best known results using the ISPD-2018 initial detailed routing benchmark suite. Second, we have presented TritonRoute, an open source detailed router. We describe an in-memory router database, and an end-to-end detailed routing scheme. We evaluate our router using the official ISPD-2018 benchmark suite, and show that we reach an unprecedented, extremely low level of DRCs (< 20) in seven of 10 testcases, a 99.3% reduction of DRCs on average as compared to the known best detailed routing solutions from all published academic detailed routers. Overall, compared to the known best detailed routing solutions, TritonRoute improves wirelength by up to 0.8% (avg. 0.4%), via count by up to 16.1% (avg. 9.3%), and DRCs by up to 100% (avg. 92.0%).

The methodologies and optimizations presented in this thesis are only tips of an iceberg to address critical challenges in advanced VLSI. Beyond this thesis, future directions and ongoing works include, but are not limited to, the following.

- Better PPAC requires an even tighter integration in physical design. In the past decade, we have seen tight integration of different stages, e.g., global placement considering routing congestion, detailed placement considering pin access, etc. Such optimizations are still in the form of look-ahead, with limitations due to various practical and engineering reasons. One example is between detailed placement and detailed routing pin access. On the one hand, the detailed placement engine does not "understand" how the pin is accessed; while on the other hand, the detailed routing engine does not "understand" how to best move a cell. In this situation, solution space is lost during routing while a single move of a cell might save several DRCs. In-route cell movement, or even a unified metric supporting both cell movement and "real" detailed routing pin access, is preferred.

- If tighter integration is to break the wall between different physical design stages, to achieve ultimate PPAC we must also build a better wall in layout – a seamless, synthesis, place-and-route full flow on any clip of the layout. From a flow perspective, look-ahead cannot solve all the problems. Every time a problem occurs, designers cannot afford to loop through several previous stages to solve the problem on a whole layout scale. Instead, a localized layout can be generated and a full-flow optimization can be run on a small clip of the layout. Even though existing physical design tools may have such capability to some extent, an automatic and seamless framework is still largely

missing. Especially for academic researchers, there are several valuable high-level decisions worth investigating, such as where to generate the localized clip, and which optimization stages to loop through.

- Several other research topics are also among the key interests of physical design engineers and EDA R&D. One such problem is that of detailed routing convergence. In mature technology nodes, the opportunistic search and repair heuristic works well with fast convergence. However, in the $5nm$ node and below, more restricted design rules and denser designs result in much slower DRC convergence. The solution quality largely depends on fine tuning dozens of parameters, without an analytical, or even a systematic, understanding. A detailed analysis and explanation of each impacting factor to the underlying path search algorithm, or the proposal of new metric, is of high value to both academia and industry.

- Last, beyond physical design, innovative "More-than-Moore" approaches are one of a few ways to go beyond conventional PPAC tradeoffs. For accuracy-tolerant tasks, we have seen active academic research and development using approximate computing and analog computing, achieving both high speed and low power. Recently, in-memory computing shows great potential of power reduction compared to conventional designs for machine learning workloads. With other fundamental breakthroughs, such as in materials science and quantum computing, these open up a new era of computing. In turn, these innovations can also provide abundant opportunities to further optimize physical design, by utilizing new hardware and new algorithms.

Bibliography

[1] M. Ahrens, M. Gester, N. Klewinghaus, D. Müller, S. Peyer, C. Schulte and G. Téllez, "Detailed Routing Algorithms for Advanced Technology Nodes", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34(4) (2015), pp. 563-576.

[2] C. Albrecht, B. Korte, J. Schietke and J. Vygen, "Maximum Mean Weight Cycle in a Digraph and Minimizing Cycle Time of a Logic Chip", *Discrete Applied Mathematics* 123(1-3) (2002), pp. 103-127.

[3] C. J. Alpert, Z. Li, M. D. Moffitt, G.-J. Nam, J. A. Roy and G. Tellez, "What Makes a Design Difficult to Route", *Proc. ACM International Symposium on Physical Design*, 2010, pp. 7-12.

[4] C. J. Alpert, Z. Li, G.-J. Nam, S. Ramji, C. N. Sze, P. G. Villarubia and N. Viswanathan, "Structured Placement of Latches/Flip-Flops to Minimize Clock Power in High-Performance Designs", *U.S. Patent* No. US8954912B2, February 2015.

[5] D. Arthur and S. Vassilvitskii, "K-Means++: The Advantages of Careful Seeding", *Proc. ACM-SIAM Symposium on Discrete Algorithms*, 2007, pp. 1027-1035.

[6] F.-Y. Chang, R.-S. Tsay, W.-K. Mak and S.-H. Chen, "MANA: A Shortest Path Maze Algorithm Under Separation and Minimum Length NAnometer Rules", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 32(10) (2013), pp. 1557-1568.

[7] H.-Y. Chen and Y.-W. Chang, "Global and Detailed Routing", Chapter 12 in Wang, Chang and Cheng (Eds.), *Electronic Design Automation: Synthesis, Verification, and Test*, Morgan Kaufmann, 2009, pp. 687-749. `http://cc.ee.ntu.edu.tw/~ywchang/Courses/PD_Source/EDA_routing.pdf`

[8] D. C. Chen, G. S. Lin, T. H. Lee, R. Lee, Y. C. Liu, M. F. Wang, Y. C. Cheng and D. Y. Wu, "Compact Modeling Solution of Layout Dependent Effect for FinFET Technology", *Proc. International Conference on Microelectronics Test Structures*, 2015, pp. 110-115.

[9] G. Chen, C.-W. Pui, H. Li, J. Chen, B. Jiang and E. F. Y. Young, "Detailed Routing by Sparse Grid Graph and Minimum-Area-Captured Path Search", *Proc. Asia and South Pacific Design Automation Conference*, 2019, pp. 754-760.

[10] G. Chen, C.-W. Pui, H. Li and E. F. Y. Young, "Dr. CU: Detailed Routing by Sparse Grid Graph and Minimum-Area-Captured Path Search", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, to appear. DOI:10.1109/TCAD.2019.2927542

[11] N. K. Darav, I. S. Bustany, A. Kennings and R. Mamidi, "ICCAD-2017 CAD Contest in Multi-Deck Standard Cell Legalization and Benchmarks", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2017, pp. 867-871.

[12] P. Debacker, K. Han, A. B. Kahng, H. Lee, P. Raghavan and L. Wang, "Vertical M1 Routing-Aware Detailed Placement for Congestion and Wirelength Reduction in Sub-10nm Nodes", *Proc. ACM/EDAC/IEEE Design Automation Conference*, 2017, pp. 51:1-51:6.

[13] C. Deng, Y.-C. Cai and Q. Zhou, "Register Clustering Methodology for Low Power Clock Tree Synthesis", *Journal of Computer Science and Technology* 30(2) (2015), pp. 391-403.

[14] Y. Ding, C. Chu and W.-K. Mak, "Throughput Optimization for SADP and E-beam Based Manufacturing of 1D Layout", *Proc. ACM/EDAC/IEEE Design Automation Conference*, 2014, pp. 1-6.

[15] Y. Ding, C. Chu and W.-K. Mak, "Pin Accessibility-Driven Detailed Placement Refinement", *Proc. ACM International Symposium on Physical Design*, 2017, pp. 133-140

[16] Y. Ding, C. Chu and W.-K. Mak, "Self-Aligned Double Patterning Lithography Aware Detailed Routing with Color Preassignment", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36(8) (2017), pp. 1381-1394.

[17] S. Dobre, Qualcomm CDMA Technologies, Inc., *personal communication*, April 2016.

[18] S. Dobre, A. B. Kahng and J. Li, "Mixed Cell-Height Implementation for Improved Design Quality in Advanced Nodes", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2015, pp. 854-860.

[19] S. Dolgov, A. Volkov, L. Wang and B. Xu, "2019 CAD Contest: LEF/DEF Based Global Routing", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2019, pp. 1-4.

[20] Y. Du, Q. Ma, H. Song, J. Shiely, G. Luk-Pat, A. Miloslavsky and M. D. F. Wong, "Spacer-is-Dielectric-Compliant Detailed Routing for Self-Aligned Double Patterning Lithography", *Proc. ACM/EDAC/IEEE Design Automation Conference*, 2013, pp. 1-6.

[21] Y. Du and M. D. F. Wong, "Optimization of Standard Cell Based Detailed Placement for 16nm FinFET Process", *Proc. Design, Automation and Test in Europe*, 2014, pp. 1-6.

[22] Y. Du, H. Zhang, M. D. F. Wong and K.-Y. Chao, "Hybrid Lithography Optimization with E-beam and Immersion Processes for 16nm 1D Gridded Design", *Proc. Asia and South Pacific Design Automation Conference*, 2012, pp. 707-712.

[23] S.-Y. Fang, "Cut Mask Optimization with Wire Planning in Self-Aligned Multiple Patterning Full-Chip Routing", *Proc. Asia and South Pacific Design Automation Conference*, 2015, pp. 396-402.

[24] J. V. Faricelli, "Layout-Dependent Proximity Effects in Deep Nanoscale CMOS", *Proc. IEEE Custom Integrated Circuits Conference*, 2010, pp. 1-8.

[25] A. Feller, "Automatic Layout of Low-Cost Quick-Turnaround Random-Logic Custom LSI Devices", *Proc. ACM/IEEE Design Automation Conference*, 1976, pp. 79-85.

[26] G.-R. Gao and D. Z. Pan, "Flexible Self-Aligned Double Patterning Aware Detailed Routing with Prescribed Layout Planning", *Proc. ACM International Symposium on Physical Design*, 2012, pp. 25-32.

[27] M. Gester, D. Muller, T. Nieberg, C. Panten, C. Schulte and J. Vygen, "BonnRoute: Algorithms and Data Structures for Fast and Good VLSI Routing", *ACM Transactions on Design Automation of Electronic Systems* 18(2) (2013), pp. 32:1-32:24.

[28] W. Gillijns, S. M. Y. Sherazi, D. Trivkovic, B. Chava, B. Vandewalle, V. Gerousis, P. Raghavan, J. Ryckaert, K. Mercha, D. Verkest, G. McIntyre and K. Ronse, "Impact of a SADP Flow on the Design and Process for N10/N7 Metal Layers", *Proc. SPIE Design-Process-Technology Co-Optimization for Manufacturability IX*, 2015, pp. 942709:1-942709:9.

[29] S. M. M. Gonçalves, L. S. Rosa and F. S. Marques, "An Improved Heuristic Function for A*-Based Path Search in Detailed Routing", *Proc. IEEE International Symposium on Circuits and Systems*, 2019, pp. 1-5.

[30] S. M. M. Gonçalves, L. S. Rosa and F. S. Marques, "DRAPS: A Design Rule Aware Path Search Algorithm for Detailed Routing", *IEEE Transactions on Circuits and Systems II: Express Briefs*, to appear. DOI:10.1109/TCSII.2019.2937893

[31] M. Gupta, K. Jeong and A. B. Kahng, "Timing Yield-Aware Color Reassignment and Detailed Placement Perturbation for Bimodal CD Distribution in Double Patterning Lithography", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 29(8) (2010), pp. 1129-1242.

[32] P. Gupta, A. B. Kahng, O. S. Nakagawa and K. Samadi, "Closing the Loop in Interconnect Analyses and Optimization: CMP Fill, Lithography and Timing", *Proc. International VLSI/ULSI Multilevel Interconnection Conference*, 2005, pp. 352-363.

[33] P. Gupta, A. B. Kahng and C.-H. Park, "Detailed Placement for Improved Depth of Focus and CD Control", *Proc. Asia and South Pacific Design Automation Conference*, 2005, pp. 343-348.

[34] P. Gupta, A. B. Kahng and C.-H. Park, "Manufacturing-Aware Design Methodology for Assist Feature Correctness", *Proc. SPIE Design and Process Integration for Microelectronic Manufacturing III*, 2005, pp. 131-140.

[35] F. O. Hadlock, "A Shortest Path Algorithm for Grid Graphs", *Networks* 7(4) (1977), pp. 323-334.

[36] C. Han, K. Han, A. B. Kahng, H. Lee, L. Wang and B. Xu, "Optimal Multi-Row Detailed Placement for Yield and Model-Hardware Correlation Improvements in Sub-10nm VLSI", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2017, pp. 667-674.

[37] K. Han, "IC Physical Design Methodologies for Advanced Process Nodes", *Ph.D. Thesis*, Electrical and Computer Engineering, University of California, San Diego, 2018.

[38] K. Han, A. B. Kahng and H. Lee, "Evaluation of BEOL Design Rule Impacts Using an Optimal ILP-Based Detailed Router", *Proc. ACM/EDAC/IEEE Design Automation Conference*, 2015, pp. 68:1-68:6.

[39] K. Han, A. B. Kahng and H. Lee, "Scalable Detailed Placement Legalization for Complex Sub-14nm Constraints", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2015, pp. 867-873.

[40] K. Han, A. B. Kahng, H. Lee and L. Wang, "ILP-Based Co-Optimization of Cut-Mask Layout, Dummy Fill and Timing for Sub-14nm BEOL Technology", *Proc. SPIE Photomask Technology*, 2015, pp. 96350E:1-96350E:14.

[41] T. Han, H. Liu and Y. Chen, "A Paradigm Shift in Patterning Foundation from Frequency Multiplication to Edge-Placement Accuracy: A Novel Processing Solution by Selective Etching and Alternating-Material Self-Aligned Multiple Patterning", *Proc. SPIE Alternative Lithography Technologies VIII*, 2016, pp. 977718:1-977718:16.

[42] A. Hetzel, "A Sequential Detailed Router for Huge Grid Graphs", *Proc. Design, Automation and Test in Europe*, 1998, pp. 332-339.

[43] D. W. Hightower, "A Solution to Line-Routing Problems on the Continuous Plane", *Proc. ACM/IEEE Design Automation Conference*, 1969, pp. 1-24.

[44] D. Hill, "Method and System for High Speed Detailed Placement of Cells Within an Integrated Circuit Design", *US Patent* No. US6370673B1, April 2002.

[45] W. Hou, D. Liu and P.-H. Ho, "Automatic Register Banking for Low-Power Clock Trees", *Proc. International Symposium on Quality Electronic Design*, 2009, pp. 647-652.

[46] C.-C. Hsu, Y.-T. Chang and M. P.-H. Lin, "Crosstalk-Aware Power Optimization with Multi-Bit Flip-Flops", *Proc. Asia and South Pacific Design Automation Conference*, 2012, pp. 431-436.

[47] S.-W. Hur and J. Lillis, "Mongrel: Hybrid Techniques for Standard Cell Placement", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2000, pp. 165-170.

[48] I. H.-R. Jiang, C. L. Chang and Y. M. Yang, "INTEGRA: Fast Multibit Flip-Flop Clustering for Clock Power Saving", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31(2) (2012), pp. 192-204.

[49] A. B. Kahng, "The ITRS Design Technology and System Drivers Roadmap: Process and Status", *Proc. ACM/EDAC/IEEE Design Automation Conference*, 2013, pp. 34-39.

[50] A. B. Kahng, I. L. Markov and S. Reda, "On Legalization of Row-Based Placements", *Proc. ACM Great Lakes Symposium on Very Large Scale Integration*, 2004, pp. 214-219.

[51] A. B. Kahng, P. Sharma and R. O. Topaloglu, "Exploiting STI Stress for Performance", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2007, pp. 83-90.

[52] A. B. Kahng and R. O. Topaloglu, "A DOE Set for Normalization-Based Extraction of Fill Impact on Capacitances", *Proc. International Symposium on Quality Electronic Design*, 2007, pp. 467-474.

[53] A. B. Kahng, P. Tucker and A. Zelikovsky, "Optimization of Linear Placements for Wirelength Minimization with Free Sites", *Proc. Asia and South Pacific Design Automation Conference*, 1999, pp. 241-244.

[54] A. B. Kahng, L. Wang and B. Xu, "TritonRoute: An Initial Detailed Router for Advanced VLSI Technologies", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2018, pp. 81:1-81:8.

[55] S. Khuller and Y. J. Sussmann, "The Capacitated K-Center Problem", *SIAM Journal on Discrete Mathematics* 13(3) (2000), pp. 403-418.

[56] Y. Kretchmer, "Using Multi-Bit Register Inference to Save Area and Power: The Good, The Bad, and The Ugly", *EE Times Asia*, 2001.

[57] C. Y. Lee, "An Algorithm for Path Connections and Its Applications", *IRE Transactions on Electronic Computers* 10(3) (1961), pp. 346-365.

[58] C. Y. Lee, C.-Y. Ting and J.-H. Shieh, "Method of Patterning for a Semiconductor Device", *US Patent* No. US8697537B2, April 2014.

[59] H. K.-S. Leung, "Advanced Routing in Changing Technology Landscape", *Proc. ACM International Symposium on Physical Design*, 2003, pp. 118-121.

[60] H. Li, G. Chen, B. Jiang, J. Chen and E. F. Y. Young, "Dr. CU 2.0: A Scalable Detailed Routing Framework with Correct-by-Construction Design Rule Satisfaction", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2019, pp. 1-7.

[61] S. Li and C.-K. Koh, "Mixed Integer Programming Models for Detailed Placement", *Proc. ACM International Symposium on Physical Design*, 2012, pp. 87-94.

[62] S. Li and C.-K. Koh, "MIP-based Detailed Placer for Mixed-size Circuits", *Proc. ACM International Symposium on Physical Design*, 2014, pp. 11-18.

[63] T. Lin and C. Chu, "TPL-Aware Displacement-driven Detailed Placement Refinement with Coloring Constraints", *Proc. ACM International Symposium on Physical Design*, 2015, pp. 75-80.

[64] M. P.-H. Lin, C.-C. Hsu and Y.-T. Chang, "Post-Placement Power Optimization with Multi-Bit Flip-Flops", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 30(12) (2011), pp. 1870-1882.

[65] M. P.-H. Lin, C.-C. Hsu and Y.-C. Chen, "Clock-Tree Aware Multibit Flip-Flop Generation During Placement for Power Optimization", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34(2) (2015), pp. 280-292.

[66] Y. Lin, B. Yu, X. Xu, J.-R. Gao, N. Viswanathan, W.-H. Liu, Z. Li, C. J. Alpert and D. Z. Pan, "MrDP: Multiple-row Detailed Placement of Heterogeneous-sized Cells for Advanced Nodes", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2016, pp. 1-8.

[67] Y. Lin, B. Yu, B. Xu and D. Z. Pan, "Triple Patterning Aware Detailed Placement Toward Zero Cross-Row Middle-of-Line Conflict", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2015, pp. 396-403.

[68] I.-J. Liu, S.-Y. Fang and Y.-W. Chang, "Overlay-Aware Detailed Routing for Self-Aligned Double Patterning Lithography Using the Cut Process", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35(9) (2016), pp. 1519-1531.

[69] W.-H. Liu, C.-K. Koh and Y.-L. Li, "Optimization of Placement Solutions for Routability", *Proc. ACM/EDAC/IEEE Design Automation Conference*, 2013, pp.153:1-153:9.

[70] S. S.-Y. Liu, W.-T. Lo, C.-J. Lee and H.-M. Chen, "Agglomerative-Based Flip-Flop Merging and Relocation for Signal Wirelength and Clock Tree Optimization", *ACM Transactions on Design Automation of Electronic Systems* 18(3) (2013), pp. 40:1-40:20.

[71] S.-C. Lo, C.-C. Hsu and M. P.-H. Lin, "Power Optimization for Clock Network with Clock Gate Cloning and Flip-Flop Merging", *Proc. ACM International Symposium on Physical Design*, 2014, pp. 77-84.

[72] W. K. Luk, "A Greedy Switch-Box Router", *Integration* 3(2) (1985), pp. 129-149.

[73] S. Mantik, G. Posser, W.-K. Chow, Y. Ding and W.-H. Liu, "ISPD 2018 Initial Detailed Routing Contest and Benchmarks", *Proc. ACM International Symposium on Physical Design*, 2018, pp. 140-143.

[74] A. D. Mehta, Y.-P. Chen, N. Menezes, D. F. Wong and L. T. Pileggi, "Clustering and Load Balancing for Buffered Clock Tree Synthesis", *Proc. IEEE International Conference on Computer Design*, 1997, pp. 217-223.

[75] Model-Hardware Correlation Team, *Samsung Electronics Co., Ltd.*, November 2016.

[76] G.-J. Nam, IBM, *personal communication*, March 2016.

[77] T. Nieberg, "Gridless Pin Access in Detailed Routing", *Proc. ACM/EDAC/IEEE Design Automation Conference*, 2011, pp. 170-175.

[78] N. J. Nilsson, "State-Space Search Methods", in *Problem-Solving Methods in Artificial Intelligence*, McGraw-Hill Book Co., 1971, pp. 43-79.

[79] S.-K. Oh, S.-H. Baek, S.-Y. Lee and T.-J. Song, "Standard Cell Library, Method of Using the Same, and Method of Designing Semiconductor Integrated Circuit", *US Patent* No. US9830415B2, November 2017.

[80] H.-C. Ou, K.-H. Tseng, J.-Y. Liu, I.-P. Wu and Y.-W. Chang, "Layout-Dependent-Effects-Aware Analytical Analog Placement", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35(8) (2016), pp. 1243-1254.

[81] M. Pan, N. Viswanathan and C. Chu, "An Efficient and Effective Detailed Placement Algorithm", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2005, pp. 48-55.

[82] D. Papa, N. Viswanathan, C. Sze, Z. Li, G.-J. Nam, C. Alpert and I. L. Markov, "Physical Synthesis with Clock-Network Optimization for Large Systems on Chips", *IEEE Micro* 31(4) (2011), pp. 51-62.

[83] I. Pohl, "Bi-Directional Search", *Machine Intelligence* (1971), pp. 127-140.

[84] X. Qiu and M. Marek-Sadowska, "Can Pin Access Limit the Footprint Scaling?", *Proc. ACM/EDAC/IEEE Design Automation Conference*, 2012, pp. 1100-1106.

[85] P. Raghavan, M. G. Bardon, D. Jang, P. Schuddinck, D. Yakimets, J. Ryckaert, A. Mercha, N. Horiguchi, N. Collaert, A. Mocuta, D. Mocuta, Z. Tokei, D. Verkest, A. Thean and A. Steegen, "Holistic Device Exploration for 7nm Node", *Proc. IEEE Custom Integrated Circuits Conference*, 2015, pp. 1-5.

[86] P. J. Rousseeuw, "Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis", *Journal of Computational and Applied Mathematics* 20 (1987), pp. 53-65.

[87] S. M. Y. Sherazi, B. Chava, P. Debacker, M. G. Bardon, P. Schuddinck, F. Firouzi, P. Raghavan, A. Mercha, D. Verkest and J. Ryckaert, "Architectural Strategies in Standard-Cell Design for the 7nm and beyond Technology Node", *SPIE Journal of Micro/Nanolithography, MEMS, and MOEMS* 15(1) (2016), pp. 1-11.

[88] E. Shragowitz and S. Keel, "A Global Router Based on a Multicommodity Flow Model", *Integration* 5(1) (1987), pp. 3-16.

[89] J. Soukup, "Fast Maze Router", *Proc. ACM/IEEE Design Automation Conference*, 1978, pp. 100-102.

[90] Y.-H. Su and Y.-W. Chang, "Nanowire-Aware Routing Considering High Cut-Mask Complexity", *Proc. ACM/EDAC/IEEE Design Automation Conference*, 2015, pp. 1-6.

[91] F.-K. Sun, H. Chen, C.-Y. Chen, C.-H. Hsu and Y.-W. Chang, "A Multithreaded Initial Detailed Routing Algorithm Considering Global Routing Guides", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2018, pp. 82:1-82:7.

[92] M. Tarabbia, A. Mittal and N. Hindawy, "Forming FinFET Cell with Fin Tip and Resulting Device", *US Patent* No. US9059093B2, June 2015.

[93] H. Tian, Y. Du, H. Zhang, Z. Xiao and M. D. F. Wong, "Triple Patterning Aware Detailed Placement with Constrained Pattern Assignment", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2014, pp. 116-123.

[94] C.-C. Tsai, Y. Shi, G. Luo and I. H.-R. Jiang, "FF-bond: Multi-Bit Flip-Flop Bonding at Placement", *Proc. ACM International Symposium on Physical Design*, 2013, pp. 147-153.

[95] P.-S. Tzeng and C. H. Sequin, "Codar: A Congestion-Directed General Area Router", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 1988, pp. 30-33.

[96] S.-H. Wang, Y.-Y. Liang, T.-Y. Kuo and W.-K. Mak, "Power-Driven Flip-Flop Merging and Relocation", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31(2) (2012), pp. 180-191.

[97] M.-P. Wong, W.-H. Liu and T.-C. Wang, "Negotiation-Based Track Assignment Considering Local Nets", *Proc. Asia and South Pacific Design Automation Conference*, 2016, pp. 378-383.

[98] G. Wu and C. Chu, "Detailed Placement Algorithm for VLSI Design with Double-Row Height Standard Cells", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 35(8) (2016), pp. 1569-1573.

[99] R. Xie, K.-Y. Lim, M. G. Sung and R. R.-H. Kim, "Methods of Forming Single and Double Diffusion Breaks on Integrated Circuit Products Comprised of FinFET Devices and The Resulting Products", *US Patent* No. US9412616B1, August 2016.

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

[101] C. Xu, P. Li, G. Luo, Y. Shi and I. H.-R. Jiang, "Analytical Clustering Score with Application to Post-Placement Multi-Bit Flip-Flop Merging", *Proc. ACM International Symposium on Physical Design*, 2015, pp. 93-100.

[102] X. Xu, B. Yu, J.-R. Gao, C.-L. Hsu and D. Z. Pan, "PARR: Pin Access Planning and Regular Routing for Self-Aligned Double Patterning", *ACM Transactions on Design Automation of Electronic Systems* 21(3) (2016), article 42.

[103] J.-T. Yan and Z.-W. Chen, "Construction of Constrained Multi-Bit Flip-Flops for Clock Power Reduction", *Proc. International Conference on Green Circuits and Systems* 2010, pp. 675-678.

[104] W. Ye, B. Yu, D. Z. Pan, Y.-C. Ban and L. Liebmann, "Standard Cell Layout Regularity and Pin Access Optimization Considering Middle-of-Line", *Proc. ACM Great Lakes Symposium on Very Large Scale Integration*, 2015, pp. 289-294.

[105] B. Yu, X. Xu, J.-R. Gao, Y. Lin, Z. Lee, C. J. Alpert and D. Z. Pan, "Methodology for Standard Cell Compliance and Detailed Placement for Triple Patterning Lithography", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34(5) (2015), pp. 726-739.

[106] B. Yu, X. Xu, J.-R. Gao and D. Z. Pan, "Methodology for Standard Cell Compliance and Detailed Placement for Triple Patterning Lithography", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2013, pp. 349-356.

[107] Y. Zhang and C. Chu, "CROP: Fast and Effective Congestion Refinement of Placement", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2009, pp. 344-350.

[108] Y. Zhang and C. Chu, "RegularRoute: An Efficient Detailed Router Applying Regular Routing Patterns", *IEEE Transactions on Very Large Scale Integration Systems* 21(9) (2013), pp. 1655-1668.

[109] H. Zhang, Y. Du, M. D. F. Wong and K.-Y. Chao, "Mask Cost Reduction with Circuit Performance Consideration for Self-Aligned Double Patterning", *Proc. Asia and South Pacific Design Automation Conference*, 2011, pp. 787-792.

[110] H. Zhang, Y. Du, M. D. F. Wong and K.-Y. Chao, "Lithography-Aware Layout Modification Considering Performance Impact", *Proc. International Symposium on Quality Electronic Design*, 2011, pp. 1-5.

[111] ASAP ASU 7nm PDK. `http://asap.asu.edu/asap/`

[112] Cadence Innovus Implementation System. `https://www.cadence.com/content/cadence-www/global/en_US/home/tools/digital-design-and-signoff/soc-implementation-and-floorplanning/innovus-implementation-system.html`

[113] Cadence Quantus QRC Extraction Solution.
https://www.cadence.com/content/cadence-www/global/en_US/home/tools/digital-design-and-signoff/silicon-signoff/quantus-qrc-extraction-solution.html

[114] Cadence Tempus Timing Signoff Solution.
https://www.cadence.com/content/cadence-www/global/en_US/home/tools/digital-design-and-signoff/silicon-signoff/tempus-timing-signoff-solution.html

[115] IBM ILOG CPLEX Optimization Studio.
https://www.ibm.com/products/ilog-cplex-optimization-studio

[116] Dr. CU 2.0, https://github.com/cuhk-eda/dr-cu/releases/tag/v4.1.1

[117] International Technology Roadmap for Semiconductors.
http://www.itrs2.net/itrs-reports.html

[118] imec. https://www.imec-int.com/

[119] LEF/DEF 5.7 reference.
http://projects.si2.org/openeda.si2.org/projects/lefdef

[120] LEF/DEF Language Reference. http://www.ispd.cc/contests/18/lefdefref.pdf

[121] LEMON (Library for Efficient Modeling and Optimization in Networks).
http://lemon.cs.elte.hu/trac/lemon

[122] W.-H. Liu, "ISPD 2018 Initial Detailed Routing Contest and Benchmarks" *presentation slides*, http://www.ispd.cc/slides/2018/s7_3.pdf

[123] Mentor Graphics Calibre.
https://www.mentor.com/products/ic_nanometer_design/verification-signoff/physical-verification/

[124] OpenCores: Open Source IP-Cores. http://www.opencores.org

[125] OpenMP Architecture Review Board, "OpenMP Application Program Interface, Version 4.0".

[126] The-OpenROAD-Project/TritonRoute: UCSD Detailed Router, https://github.com/The-OpenROAD-Project/TritonRoute

[127] TritonRoute Version 0.0.6.0, https://github.com/The-OpenROAD-Project/TritonRoute/releases/tag/0.0.6.0

[128] Si2 OpenAccess. http://projects.si2.org/?page=69

[129] Synopsys Design Compiler.
https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/dc-ultra.html

[130] Synopsys HSPICE.
https://www.synopsys.com/verification/ams-verification/hspice.html