UNIVERSITY OF CALIFORNIA SAN DIEGO

IC Physical Design Methodologies for Advanced Process Nodes

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

in

Electrical Engineering (Computer Engineering)

by

Kwangsoo Han

Committee in charge:

Professor Andrew B. Kahng, Chair
Professor Chung-Kuan Cheng
Professor Rajesh Gupta
Professor Ryan Kastner
Professor Farinaz Koushanfar

2018

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

_____

_____

_____

_____

_____

Chair

University of California San Diego

2018

DEDICATION

To my family, without whose love, encouragement and sacrifice this thesis would not have been finished.

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. His enthusiasm and passion always stimulate me, and his attitude on research will remain as a priceless lesson in my life.

I would like to thank my fellow labmates in the UCSD VLSI CAD Laboratory (Hyein Lee, Lutong Wang, Bangqi Xu, Minsoo Kim, Uday Mallappa and Mateus Fogaca) and former lab members (Dr. Tuck-Boon Chan, Dr. Siddhartha Nath, Dr. Jiajia Li, Dr. Wei-Ting (Jonas) Chan, Dr. Ilgweon Kang, Mulong Luo, Yaping Sun, Ahmed Youssef, Tushar Shah and Sriram Venkatesh) for their assistance and enthusiastic discussions. I will remember the hard-working time and sleepless nights with them. Special thanks to Dr. Tuck-Boon Chan, Dr. Jiajia Li and Dr. Ilgweon Kang for their guidance at the beginning of my Ph.D. studies.

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

I would like to thank my industrial collaborators (especially Dr. Jae-gon Lee, Jongpil Lee, Dr. Praveen Raghavan, Peter Debacker, Samyoung Bang and Dr. Kun Young Chung) for their invaluable guidance and feedback in many of my research projects. I would like to thank my collegues (Dr. Charles J. Alpert, Dr. Zhuo Li and Dr. Wing-Kai Chow) for their thoughtful support and invaluable guidance.

Last but not least, I would like to thank my parents Yoonsuck Han and Eunhee Choi, my parents-in-law Jongsoon Kang and Sunhui Moon, and the rest of my family for their endless love and encouragement: my wife Jinsil Kang whose love and sacrifice allowed me to finish this journey, and my sons Jio Han and Teo Han who have made me always happy with their lovely smiles. This thesis is dedicated to them.

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

Chapter 2 contains reprints of 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; Kwangsoo Han, Andrew B. Kahng and Hyein Lee, “Scalable Detailed Placement Legalization for Complex Sub-14nm Constraints”, *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2015; and Kwangsoo Han, Andrew B. Kahng and Hyein Lee, “Evaluation of BEOL Design Rule Impacts Using an Optimal ILP-Based Detailed Router”, *Proc.*

ACM/ESDA/IEEE Design Automation Conference, 2015. The dissertation author is a main contributor to, and a primary author of, each of these papers.

Chapter 3 contains reprints of Kwangsoo Han, Andrew B. Kahng and Jiajia Li, "Improved Performance of 3DIC Implementations Through Inherent Awareness of Mix-and-Match Die Stacking", *Proc. Design, Automation and Test in Europe*, 2016; Kwangsoo Han, Andrew B. Kahng, Jongpil Lee, Jiajia Li and Siddhartha Nath, "A Global-Local Optimization Framework for Simultaneous Multi-Mode Multi-Corner Skew Variation Reduction", *Proc. ACM/ESDA/IEEE Design Automation Conference*, 2015; and Tuck-Boon Chan, Kwangsoo Han, Andrew B. Kahng, Jae-Gon Lee and Siddhartha Nath, "OCV-Aware Top-Level Clock Tree Optimization", *Proc. Great Lakes Symposium on Very Large Scale Integration*, 2014. The dissertation author is a main contributor to, and a primary author of, each of these papers.

Chapter 4 contains reprints of Kwangsoo Han, Andrew B. Kahng and Jiajia Li, "Optimal Generalized H-Tree Topology and Buffering for High-Performance and Low-Power Clock Distribution", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018 (to appear); Kwangsoo Han, Andrew B. Kahng, Christopher Moyes and Alex Zelikovsky, "A Study of Optimal Cost-Skew Tradeoff and Remaining Suboptimality in Interconnect Tree Constructions", *Proc. ACM International Workshop on System-Level Interconnect Prediction*, 2018; and Charles J. Alpert, Wing-Kai Chow, Kwangsoo Han, Andrew B. Kahng, Zhuo Li, Derong Liu and Sriram Venkatesh, "Prim-Dijkstra Revisited: Achieving Superior Timing-driven Routing Trees", *Proc. ACM International Symposium on Physical Design*, 2018. The dissertation author is a main contributor to, and a primary author of, each of these papers.

My coauthors (Dr. Charles J. Alpert, Dr. Tuck-Boon Chan, Dr. Wing-Kai Chow, Peter Debacker, Professor Andrew B. Kahng, Dr. Hyein Lee, Dr. Jae-Gon Lee, Jongpil Lee, Dr. Jiajia Li, Dr. Zhuo Li, Dr. Derong Liu, Christopher Moyes, Dr. Siddhartha Nath, Dr. Praveen Raghavan, Sriram Venkatesh, Lutong Wang and Professor Alex Zelikovsky, listed in alphabetical order) have all kindly approved the inclusion of the aforementioned publications in my thesis.

VITA

1987	Born, Eumseong, Chungcheongbuk-do, South Korea
2011	B.Sc., Media Communication Engineering, Hanyang University, Seoul, South Korea
2013	M.Sc., Electrical and Computer Engineering, Hanyang University, Seoul, South Korea
2016	C.Phil., Electrical Engineering (Computer Engineering), University of California, San Diego
2018	Ph.D., Electrical Engineering (Computer Engineering), University of California, San Diego

All papers co-authored with my advisor Prof. Andrew B. Kahng have authors listed in alphabetical order.

- **Kwangsoo Han**, Andrew B. Kahng and Jiajia Li, "Optimal Generalized H-Tree Topology and Buffering for High-Performance and Low-Power Clock Distribution", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* (2018), to appear.

- **Kwangsoo Han**, Andrew B. Kahng, Christopher Moyes and Alex Zelikovsky, "A Study of Optimal Cost-Skew Tradeoff and Remaining Suboptimality in Interconnect Tree Constructions", *Proc. ACM International Workshop on System-Level Interconnect Prediction*, 2018, pp. 2:1-2:8.

- Charles J. Alpert, Wing-Kai Chow, **Kwangsoo Han**, Andrew B. Kahng, Zhuo Li, Derong Liu and Sriram Venkatesh, "Prim-Dijkstra Revisited: Achieving Superior Timing-driven Routing Trees", *Proc. ACM International Symposium on Physical Design*, 2018, pp. 10-17.

- 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, pp. 667-674.

- 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), pp. 1075-1088.

- Dong-Ik Jeon, **Kwangsoo Han** and Ki-Seok Chung, "Low Power and High Performance Level-Up Shifters for Mobile Devices with Multi-VDD", *Journal of Semiconductor Technology and Science* 17(5) (2017), pp. 577-583.

- Tuck-Boon Chan, Puneet Gupta, **Kwangsoo Han**, Abde Ali Kagalwalla and Andrew B. Kahng, "Benchmarking of Mask Fracturing Heuristics", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 36(1) (2017), pp. 170-183.

- 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/ESDA/IEEE Design Automation Conference*, 2017, pp. 51:1-51:6.

- **Kwangsoo Han**, Andrew B. Kahng, Hyein Lee and Lutong 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.

- **Kwangsoo Han**, Andrew B. Kahng and Jiajia Li, "Improved Performance of 3DIC Implementations Through Inherent Awareness of Mix-and-Match Die Stacking", *Proc. Design, Automation and Test in Europe*, 2016, pp. 61-66.

- Samyoung Bang, **Kwangsoo Han**, Andrew B. Kahng and Mulong Luo, "Delay Uncertainty and Signal Criticality Driven Routing Channel Optimization for Advanced DRAM Products", *Proc. Asia and South Pacific Design Automation Conference*, 2016, pp. 697-704.

- **Kwangsoo Han**, Andrew B. Kahng and Hyein Lee, "Scalable Detailed Placement Legalization for Complex Sub-14nm Constraints", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2015, pp. 867-873.

- **Kwangsoo Han**, Andrew B. Kahng, Hyein Lee and Lutong Wang, "ILP-Based Co-Optimization of Cut-Mask Layout, Dummy Fill and Timing for Sub-14nm BEOL Technology", *Proc. SPIE/BACUS Symposium on Photomask Technology and Management*, 2015, pp. 96350E:1-96350E:14.

- **Kwangsoo Han**, Andrew B. Kahng and Hyein Lee, "Evaluation of BEOL Design Rule Impacts Using an Optimal ILP-Based Detailed Router", *Proc. ACM/ESDA/IEEE Design*

Automation Conference, 2015, pp. 1-6.

- **Kwangsoo Han**, Andrew B. Kahng, Jongpil Lee, Jiajia Li and Siddhartha Nath, "A Global-Local Optimization Framework for Simultaneous Multi-Mode Multi-Corner Skew Variation Reduction", *Proc. ACM/ESDA/IEEE Design Automation Conference*, 2015, pp. 26:1-26:6.

- Samyoung Bang, **Kwangsoo Han**, Andrew B. Kahng and Vaishnav Srinivas, "Clock Clustering and IO Optimization for 3D Integration", *Proc. ACM International Workshop on System-Level Interconnect Prediction*, 2015, pp. 1-8.

- Tuck-Boon Chan, Puneet Gupta, **Kwangsoo Han**, Abde Ali Kagalwalla, Andrew B. Kahng and Emile Sahouria, "Benchmarking of Mask Fracturing Heuristics", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2014, pp. 246-253.

- Tuck-Boon Chan, **Kwangsoo Han**, Andrew B. Kahng, Jae-Gon Lee and Siddhartha Nath, "OCV-Aware Top-Level Clock Tree Optimization", *Proc. Great Lakes Symposium on Very Large Scale Integration*, 2014, pp. 33-38.

- Dong-Ik Jeon, **Kwangsoo Han** and Ki-Seok Chung, "Novel Level-Up Shifters for High Performance and Low Power Mobile Devices", *Proc. IEEE International Conference on Consumer Electronics*, 2013, pp. 181-182.

- **Kwangsoo Han**, Dong-Ik Jeon and Ki-Seok Chung, "Ultra Low Power and High Speed FPGA Design with CNFET", *Proc. IEEE International Symposium on Communications and Information Technologies*, 2012, pp. 828-833.

- Dong-Ik Jeon, **Kwangsoo Han** and Ki-Seok Chung, "A High Performance Low Power Level-Up Shifter Design for Dual Supply Voltages", *Proc. International Technical Conference on Circuits/Systems, Computers and Communications*, 2012, pp. 421-423.

- **Kwangsoo Han**, Dong-Ik Jeon and Ki-Seok Chung, "Low Power and High Speed Level Shifter with CNFET", *Proc. International Technical Conference on Circuits/Systems, Computers and Communications*, 2012, pp. 213-215.

ABSTRACT OF THE DISSERTATION

IC Physical Design Methodologies for Advanced Process Nodes

by

Kwangsoo Han

Doctor of Philosophy in Electrical Engineering (Computer Engineering)

University of California San Diego, 2018

Professor Andrew B. Kahng, Chair

Next-generation applications in mobile, automotive, internet of things, robotic, artificial intelligence, etc. domains require the development and integration of advanced systems-on-chip (SOCs) that deliver ever-higher performance with much lower power. Thus, Moore's Law continues to be necessary, and innovations are needed beyond this law to help manage performance, power, area and cost (PPAC) for integrated-circuit (IC) design. Among the steps in the typical IC design flow, *physical design* implementation critically impacts PPAC. However, in concert with continuation of the Moore's Law trajectory, IC physical design encounters new challenges such as patterning restrictions due to manufacturing limits, severe process variation, and escalating interconnect RC delay. This thesis presents techniques to mitigate these challenges, grouped into three main thrusts: (i) manufacturing-aware design methodologies, (ii) process-aware design methodologies, and (iii) interconnect-aware design methodologies.

Multiple-patterning techniques play a key role in the quest to print ever-smaller features for continued technology scaling in advanced nodes. However, the use of multiple-patterning significantly raises the number of extra steps for patterning as well as layout constraints needed for patternability; this causes an explosion of design rules and a loss of achievable layout density. To manage the onslaught of complex design rules arising from multiple-patterning, the *manufacturing-aware design methodologies* thrust of this thesis proposes approaches to optimize 2D block mask layout for minimum timing degradation, perform detailed placement to fix complex front-end-of-line (FEOL) design rule violations, and evaluate complex back-end-of-line (BEOL) design rule impact.

Design variability due to manufacturing process variations has significant impact on the quality and yield of modern IC designs. Escalating process variation with new device architectures and manufacturing techniques (e.g., FinFET, multiple-patterning, etc.) required for node scaling results in the rapid increase of pessimism and overdesign. To mitigate the impact of severe process variation, the *process-aware design methodologies* thrust of this thesis presents approaches to optimize top-level clock tree for OCV minimization, reduce skew variation in the clock network, and perform partitioning in 3DIC that leverages a priori knowledge of inter-die variation.

In advanced technology nodes, interconnect RC delay becomes more and more dominant. The continuous rapid increase of interconnect RC leads to not only performance loss from interconnect delay increase, but circuit power and area degradation as well, due to exponential increase in the number of buffers and drivers. To mitigate the escalating interconnect RC delay, the *interconnect-aware design methodologies* thrust of this thesis proposes approaches to co-optimize wirelength and pathlength in routing, studies optimal wirelength-skew tradeoff and remaining suboptimality in interconnect tree constructions, and performs optimal generalized H-tree construction with buffering.

Chapter 1

Introduction

Over the past decades, "Moore's Law" has become well-understood as an indicator of successful advancement along the semiconductor industry roadmap. The consistent delivery of node-to-node density scaling has been a key enabler of market success and ever-deeper societal impact of integrated circuit-based products. However, due to the slowdown of density scaling, the semiconductor industry today faces two severe crises in IC design. The first crisis is cost: design in advanced nodes is too expensive. Figure 1.1 shows design cost and transistor count trends. The shortage of cost scaling makes designers unable to access the benefits of new technologies.

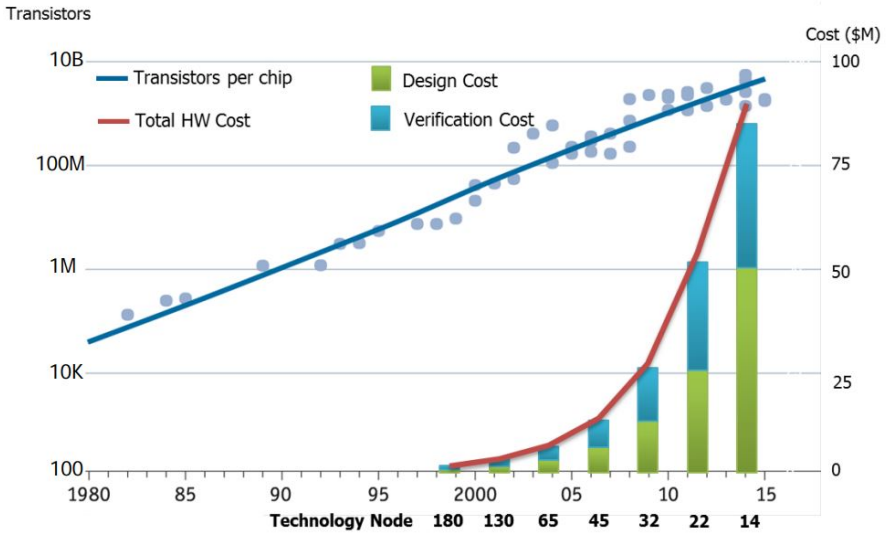


Figure 1.1: Design cost and transistor count trends [140].

The second crisis is quality: current IC design flows do not extract sufficient benefit (power, performance, area, cost, etc.) from new technology nodes. The 2013 ITRS roadmap [206] highlighted that even with continued (at least through the year 2013) "available" Moore's-Law density scaling (i.e., geometric pitch scaling) of 2× per technology node, the "realized" transistor density scaling in actual products has slowed down to around 1.6× per technology node since 2008 (see Figure 1.2). The *design capability gap* between "available" density scaling and "realizable" density scaling is caused by the failure to address new challenges – notably, growing limitations of manufacturing, increased process variations, and escalating interconnect RC delay – that arise in advanced nodes.

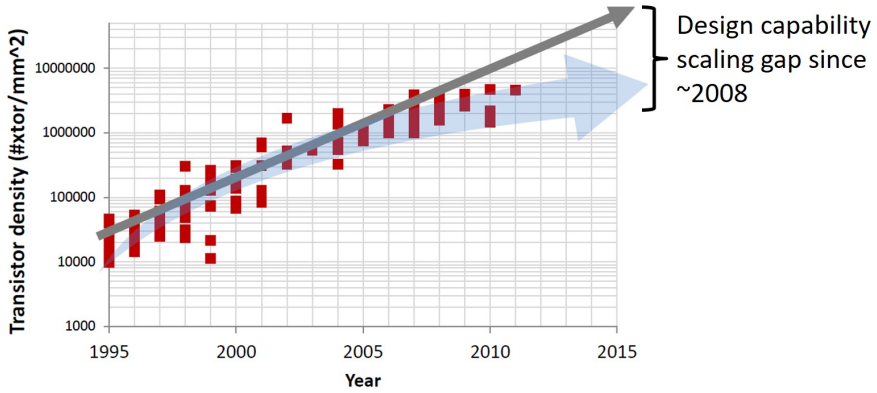


Figure 1.2: Design Capability Gap [102].

Despite the cost and quality crises facing IC design, next-generation applications in mobile, automotive, internet of things, robotic, artificial intelligence, etc. domains all require advanced SOCs that deliver ever-higher performance with much lower power. Thus, Moore's Law continues to be necessary, and innovations are needed beyond this law to help manage performance, power, area and cost (PPAC) for IC design. Among the several steps in today's typical IC design flow, *physical design* implementation has critical impacts on PPAC, and this thesis proposes several innovative physical design methodologies to achieve better design quality.

1.1 Challenges in Physical Design

Inevitably, continuation of the Moore's-Law scaling trajectory brings new challenges to IC physical design. Key challenges may be categorized as being within the contexts of (i) patterning, (ii) interconnect, (iii) transistor, (iv) process variation and (v) reliability. Figure 1.3 shows a roadmap of future technology. Notably, many innovations are essential in patterning, interconnect and transistor technologies, which implies that these three fields are the main bottlenecks for continued process node scaling.

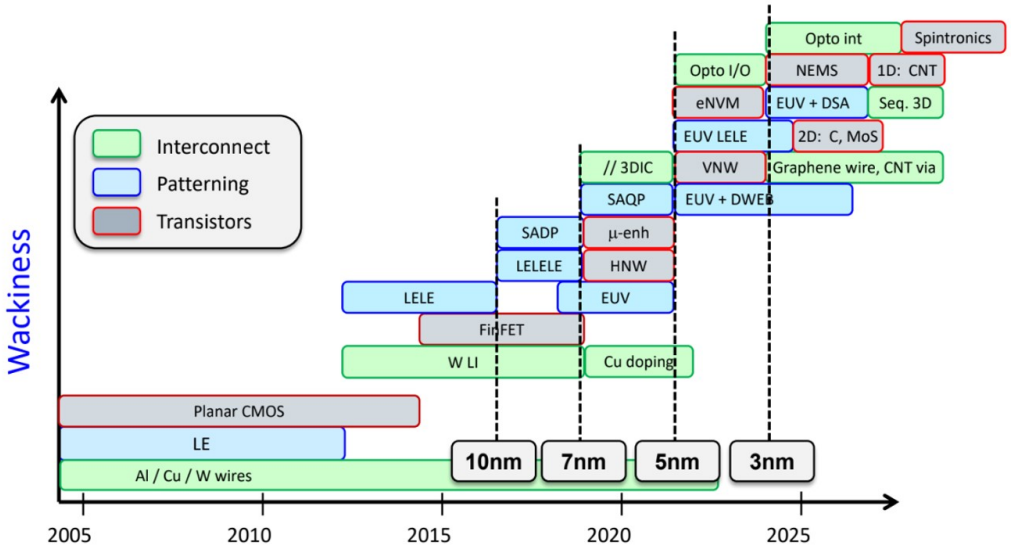


Figure 1.3: Roadmap of future technology [193].

The history of timing signoff shown in Figure 1.4 [101] provides additional perspective on how new issues or methodologies arise as process geometries continue to be scaled down. In particular, there has been a clear, node-to-node evolution of methodologies used to model process variation. For example, multi-corner-multi-mode (MCMM) is proposed at $65nm$, advanced OCV is proposed at $45nm$, and LVF is proposed at $20nm$. This indicates that new models are frequently required to capture process variation with sufficient accuracy. Further, several issues regarding reliability have become first-class concerns for IC product design, such as electromigration (EM), bias temperature instability (BTI), dynamic IR drop, etc.

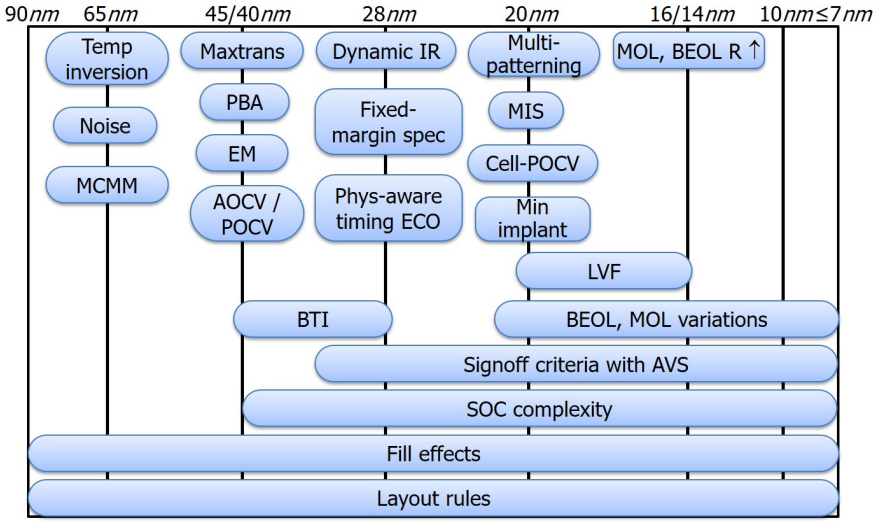


Figure 1.4: History of timing signoff [101].

Among the five categories of challenges mentioned above, this thesis focuses on three major challenges, namely, (i) limitation of manufacturing, (ii) severe process variation and (iii) escalating interconnect RC delay. The following subsections describe each challenge in detail.

1.1.1 Limitation of Manufacturing

As the semiconductor industry continues to follow Moore's Law, the need to print ever-smaller features continues. In the past, patterning resolution had been successfully scaled down by reducing the wavelength of the light source of the lithography equipment. This reached its limit for high-volume production with ArF ($193nm$) about a decade ago. That same light source has been continuously used since the $65nm$ technology node, with various innovative lithography technologies (immersion lithography, resolution enhancement techniques, multiple-patterning technology, etc.) helping to overcome inherent ArF resolution limits. Even as extreme ultraviolet (EUV) lithography with $13.5nm$ wavelength reaches mass production in foundry $7nm$ and $5nm$ processes, resolution enhancement and multiple-patterning techniques are still required for continuation of the patterning roadmap.

Notably, *multiple-patterning* lithography has played a key role in the successful continuation of Moore's Law scaling. However, the multiple-patterning technique causes two critical side effects. The first side effect is manufacturing cost: multiple-patterning escalates manufacturing cost (see Figure 1.5), due to the extra process steps required to do multiple masking, etching, etc.

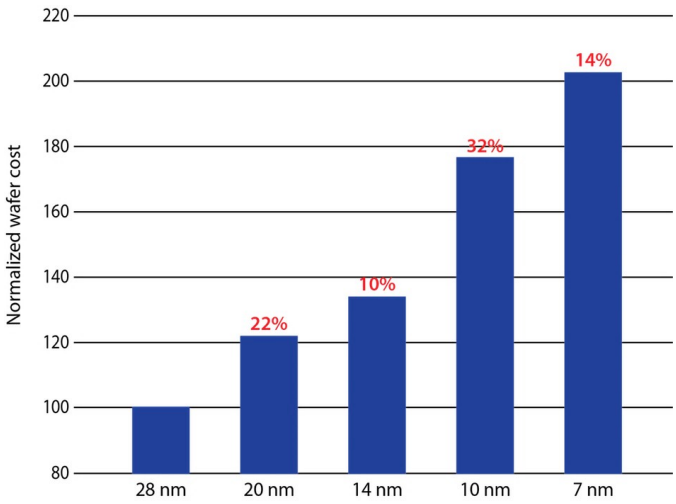


Figure 1.5: Wafer price trend [215].

The second side effect is design rule explosion: complex patterning steps, along with resolution enhancements and interactions between multiple mask layouts used to print a given layer, add a significant number of complex front-end-of-line (FEOL) and back-end-of-line (BEOL) design rules that have a high impact on design PPAC. Figure 1.6 shows how the number of design rules has grown rapidly along with process node scaling. The increase of the number of design rules in advanced nodes is clearly far beyond the historical growth trend. Moreover, new design rules are more complicated, such that design rule violations can cause other violations, leading to a string of problems that can limit yield and affect reliability. With this in mind, it is critical to understand design rules in physical design optimization. Powerful design methodology that effectively honors advanced-node design rules can achieve significant benefits: (i) cost reduction with less Engineering Change Order (ECO) fixing; (ii) yield improvement due to robust patterning; etc.

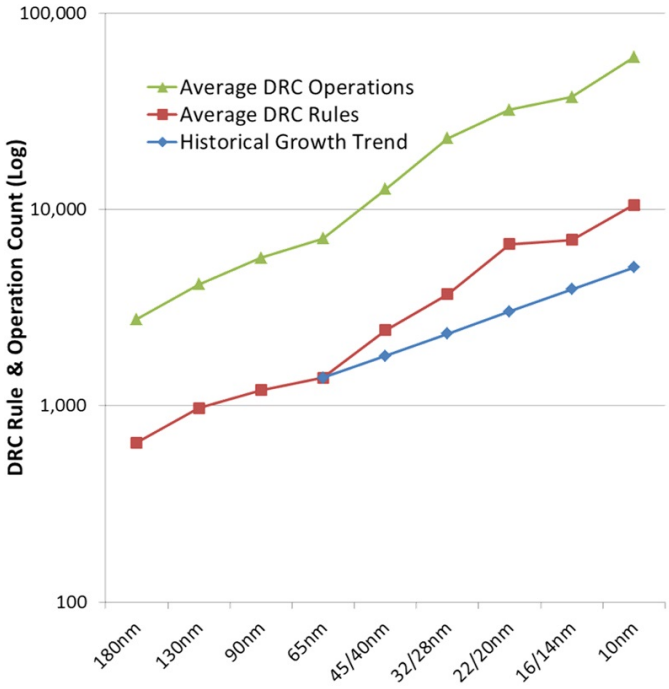


Figure 1.6: Design rules and operations explosion [214].

1.1.2 Severe Process Variation

Design variability due to process variations significantly impacts the quality and yield of advanced-node IC designs. It comes as no surprise that FEOL and BEOL process variations, along with multiple corners for voltages and temperatures (corresponding to the wide range of

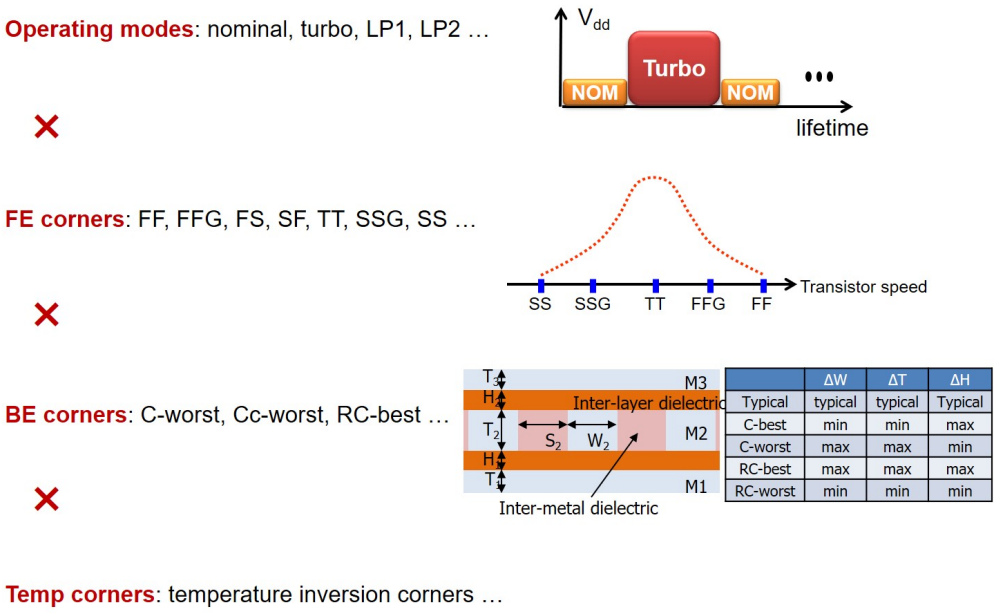


Figure 1.7: Timing corner explosion [101].

operating conditions and functional modes that modern SOCs must serve), result in corner explosion (Figure 1.7). The proliferation of design signoff corners has a multiplicative effect, e.g., when there are 3 FEOL corners, 5 BEOL corners, 2 temperatures and 3 voltages, 90 simulations (i.e, 90 = 3 x 5 x 2 x 3) are required for complete coverage. Further, the corner explosion is escalated by the rise of FEOL and BEOL layers including multiple-patterning layers. With such corner explosion, timing closure becomes harder and harder, resulting in the increase of design turnaround time (TAT) with more ECO fixing and cost. Thus, minimum selection of corners for full coverage has been a key to reduce TAT in the design closure and signoff stages of physical implementation.

Challenges to physical design and further scaling benefits arise not only from corner explosion, but also from the increased design margin needed to model severe variability that is due to new techniques (e.g., FinFET device architectures, multiple-patterning lithography, etc.) used for node scaling. The growth of design margins results in a rapid increase of pessimism and overdesign (Figure 1.8). In this regime, with a given set of signoff corners, fixing timing issues at one corner often leads to violations at other corners, and such "ping-pong" effect is mainly caused by delay variation of datapaths and clockpaths across corners. Such timing violations across corners are typically reduced by (hold and/or setup) buffer insertion, V_{th}-swapping and gate sizing on datapaths at later design stages, which can lead to potential costs of area, power and design TAT.

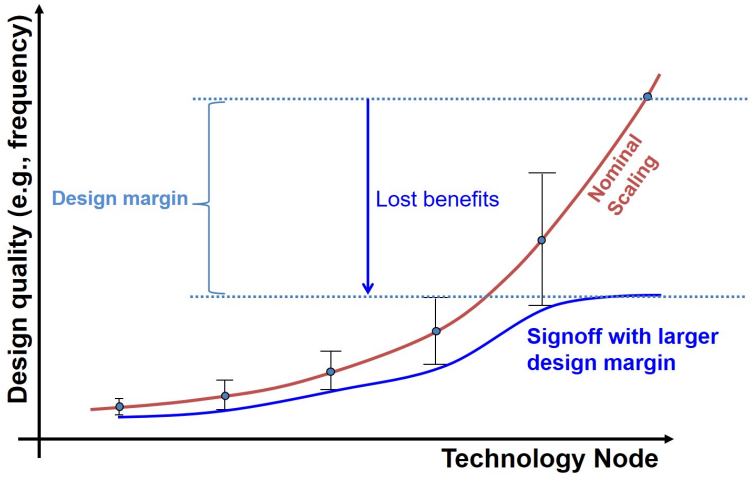


Figure 1.8: Cost of design margin.

Recently, three-dimensional IC (3DIC) technologies have emerged as a prominent option for "More-than-Moore" integration strategy, since 3DIC can provide substantial wirelength reduction and performance improvement along with a massive amount of bandwidth. However, 3DIC technologies add another dimension of variability (i.e., inter-die variation), which should be managed by physical design methodologies.

1.1.3 Escalating Interconnect RC Delay

In advanced technology nodes, the interconnect RC delay becomes more and more dominant. Reasons for this include: (i) the resistance of Cu interconnect has increased dramatically in sub-100nm nodes due to grain boundary and trench liner effects [98] and (ii) the scaling of effective dielectric constant has slowed in recent years, resulting in severely increased interconnect capacitance [206] and diminished performance benefits at new nodes.

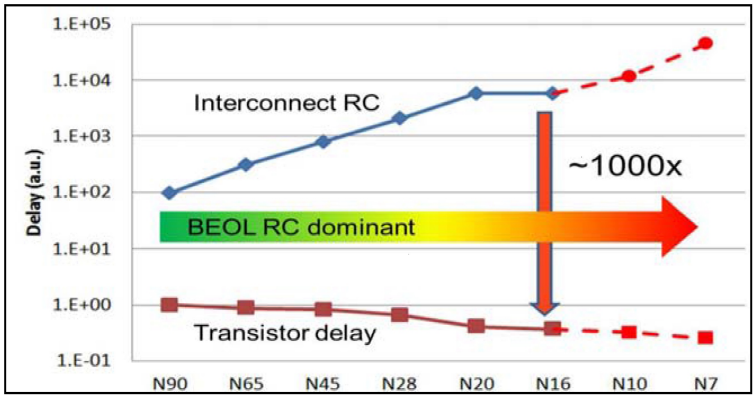


Figure 1.9: Interconnect RC increase as technology nodes scale down [44].

Figure 1.9 illustrates the escalating interconnect RC delay with node scaling, in which the gap between transistor and interconnect delays grows approximately 10× for every two technology nodes. The continuous rapid increase of interconnect RC leads to not only performance loss from interconnect delay increase, but circuit power and area degradation as well, due to the exponential increase in buffer and driver counts. With the continued performance divergence between transistors and interconnects, designs have become interconnect-limited.

1.2 This Thesis

To compensate for the design capability gap with technology scaling and to address the above-noted major challenges in physical design, this thesis proposes a number of innovative physical design methodologies. The physical design methodologies in this thesis can be grouped into three main thrusts (see Figure 1.10):

- Manufacturing-aware design methodologies;
- Process-aware design methodologies; and
- Interconnect-aware design methodologies.

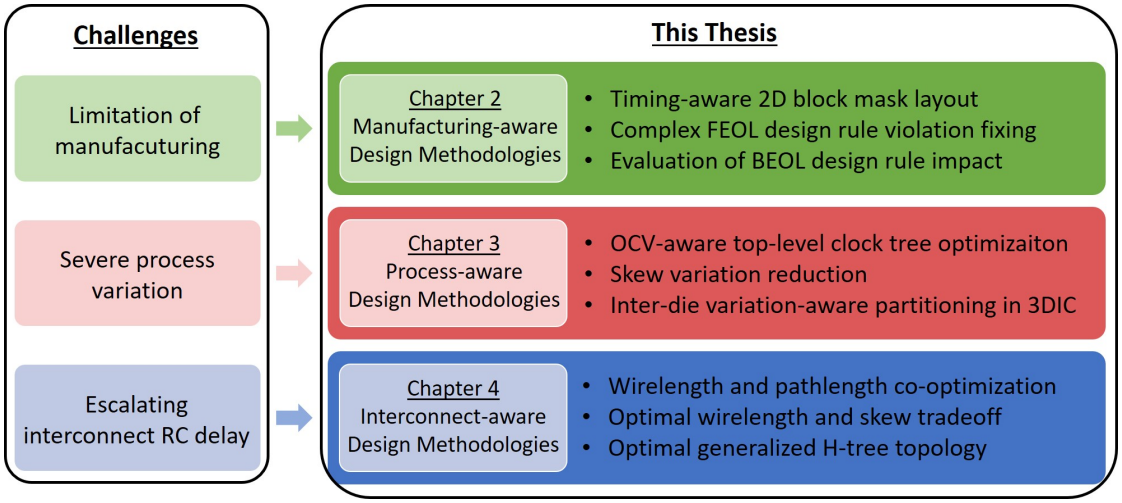


Figure 1.10: Scope and organization of this thesis.

To manage the explosion of complex design rules, e.g., with more pervasive use of multiple-patterning, the *manufacturing-aware design methodologies* thrust of this thesis proposes approaches to optimize 2D block mask layout for minimum timing degradation, perform

detailed placement to fix complex FEOL design rule violations, and evaluate complex BEOL design rule impact.

To mitigate the impact of severe process variation, the *process-aware design methodologies* thrust of this thesis presents approaches to optimize top-level clock tree for OCV minimization, reduce skew variation in the clock network, and perform partitioning in 3DIC that leverages knowledge of inter-die variation.

To mitigate the impacts of escalating interconnect RC delays, the *interconnect-aware design methodologies* thrust of this thesis proposes approaches to co-optimize wirelength and pathlength in routing, evaluate optimal wirelength-skew tradeoff and remaining suboptimality in interconnect tree constructions, and perform optimal generalized H-tree construction with buffering.

The remainder of this thesis is organized as follows.

- Chapter 2 presents three distinct studies in the context of manufacturability and new design rules. First, we address the timing-aware optimization of 2D block mask layouts under various sets of mask rules that are derived from mask patterning technology options (e.g, 193i, 193d) for foundry $7nm$ / $5nm$ (N7 / N5) BEOL. Our central contribution is a mixed integer linear programming (MILP) optimization that minimizes timing impact due to dummy metal segments while satisfying block mask rules and metal density constraints. Second, we propose a mixed integer-linear programming (MILP)-based placer, called DFPlacer, for final-phase design rule violation (DRV) fixing. DFPlacer finds DRV-free or near-DRV-free solutions considering various complex layout constraints including minimum implant width, drain-drain abutment, and oxide diffusion jogs. Third, we study the impacts of patterning technology choices and associated design rules on physical implementation density, with respect to cost-optimal design rule-correct detailed routing. A key contribution is an integer linear programming (ILP)-based optimal router (OptRouter) which considers complex design rules that arise in sub-$20nm$ process technologies. Using OptRouter, we assess wirelength and via count impacts of various design rules (implicitly, patterning technology choices) by analyzing optimal routing solutions of clips (i.e., switchbox instances) extracted from post-detailed route layouts in an advanced technology.

- Chapter 3 presents several distinct process-aware design methodologies. First, we present a new clock tree synthesis (CTS) methodology that optimizes clock logic cell placements and buffer insertions in the top level of a clock tree. Balancing the clock trees of mul-

tiple clocks is challenging because timing constraints depend on clock periods, as well as on the corresponding process, voltage and temperature (PVT) corners used for design signoff. Our work formulates the top-level clock tree optimization problem as a linear program that minimizes a weighted sum of timing slacks, clock uncertainty and wirelength. Second, we propose a novel framework encompassing both global and local clock network optimizations to minimize the sum of skew variations across different PVT corners between all sequentially adjacent sink pairs. The global optimization uses linear programming to guide buffer insertion, buffer removal and routing detours. The local optimization is based on machine learning-based predictors of latency change; these are used for iterative optimization with tree surgery, buffer sizing and buffer displacement operators. Third, we study performance improvements of 3DIC implementation that leverage a priori knowledge of mix-and-match die stacking during manufacturing. We propose partitioning methodologies to partition timing-critical paths across tiers to explicitly optimize the signed-off timing across the reduced set of corner combinations that can be produced by the stacked-die manufacturing. These include both an ILP-based methodology and a heuristic with novel maximum-cut partitioning, solved by semidefinite programming, and a signoff timing-aware Fiduccia-Mattheyses optimization. We also extend two existing 3DIC implementation flows to incorporate mix-and-match-aware partitioning and signoff, demonstrating the simplicity of adopting our techniques.

- Chapter 4 presents three distinct techniques for interconnect optimization. First, we propose a new PD-II construction which directly improves both wirelength and pathlength upon the original tree constructed by the Prim-Dijkstra (PD) method. The PD-II approach achieves improvement for both objectives, making it a clear win over PD for virtually zero additional runtime cost. PD-II is a spanning tree algorithm (which is useful for seeding global routing); however, since Steiner trees are needed for timing estimation, this work also includes a post-processing algorithm called DAS to convert PD-II trees into balanced Steiner trees. Second, we formulate the minimum-cost bounded skew spanning and Steiner tree problems as flow-based integer linear programs, and give the first-ever study of optimal cost-skew tradeoffs. We also assess the "suboptimality gap" seen in the leading heuristics (notably, Bounded-Skew DME (BST-DME), Steiner shallow-light tree (SALT), and Prim-Dijkstra (PD)) that are currently available for trading off cost and skew. Third, we study the concept of a generalized H-tree – a topologically balanced tree with an arbitrary sequence of branching factors – and propose a dynamic programming-based

method to determine optimal clock power, skew and latency, in the space of generalized H-tree solutions. Our method co-optimizes clock tree topology and buffering along branches according to fitted electrical models. We further propose a balanced K-means clustering and a linear programming-guided buffer placement approach to embed the generalized H-tree with respect to a given sink placement. We validate our solutions in commercial clock tree synthesis tool flows, in a commercial foundry's 28LP technology.

- Chapter 5 summarizes the contributions of this thesis toward new physical design methodologies in advanced nodes.

Chapter 2

Manufacturing-Aware Design Methodologies

Continued technology scaling with more pervasive use of multi-patterning has led to complex design rules and increased difficulty of maintaining high layout densities. Intuitively, emerging design rules such as block and cut mask rules, minimum implant width, unidirectional patterning, increased via spacing, etc. will decrease achievable density of the final place-and-route solution, worsening die area and product cost.

This chapter presents three distinct studies in the context of manufacturability and new design rules. First, we address the timing-aware optimization of 2D block mask layouts under various sets of mask rules that are derived from mask patterning technology options (e.g., 193i, 193d) for foundry $7nm$ / $5nm$ (N7 / N5) BEOL. Our central contribution is a mixed integer linear programming (MILP) optimization that minimizes timing impact due to dummy metal segments while satisfying block mask rules and metal density constraints. We also propose a distributed optimization flow to improve the scalability. With our optimizer, we recover up to 84% of the worst negative slack (WNS) impact from dummy segments, with up to 64% dummy removal rate. We further extend our MILP to a co-optimization of cut and block masks. Our work gives new insights into fundamental limits of benefit from emerging cut and block mask technology options. Second, we develop a mixed integer-linear programming (MILP)-based placer, called DFPlacer, for final-phase design rule violation (DRV) fixing. DFPlacer finds (near-)DRV-free solutions considering various complex layout constraints including minimum implant width, drain-drain abutment, and oxide diffusion jogs. To overcome the runtime limitation of MILP-based approaches, we implement a distributable optimization strategy based on partitioning of

the block layout into windows of cells that can be independently legalized. Using layouts in an abstracted $7nm$ library, we find that DFPlacer fixes 99% of DRVs on average with minimal impacts on area and timing. We also study an area-DRV tradeoff between two types of standard-cell library strategies, namely, with and without dummy poly gates. Third, we study the impacts of patterning technology choices and associated design rules on physical implementation density, with respect to cost-optimal design rule-correct detailed routing. A key contribution is an Integer Linear Programming (ILP)-based optimal router (OptRouter) which considers complex design rules that arise in sub-$20nm$ process technologies. Using OptRouter, we assess wirelength and via count impacts of various design rules (implicitly, patterning technology choices) by analyzing optimal routing solutions of clips (i.e., switchbox instances) extracted from post-detailed route layouts in an advanced technology.

2.1 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*[1] or *block*[2] approaches can be used. The work of [66] demonstrates that mask shapes used to keep signal wire segments (M2 pitch $= 32nm$ [150][171]) 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.[3]

Despite previous works [47][50][80][195] 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 node, which is our

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

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

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

focus in this section. [66] 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 2.1 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 2.1(a) and (b). After the cut process, Figure 2.1(c) shows one EOL extension and three non-functional dummy segments. 2D block mask application is shown in Figure 2.1(d), and Figure 2.1(e) shows the final layout with one EOL extension and one dummy segment. Compared to the layout in Figure 2.1(c), (e) is superior with smaller capacitance, lower power, and better timing, due to fewer dummy segments.

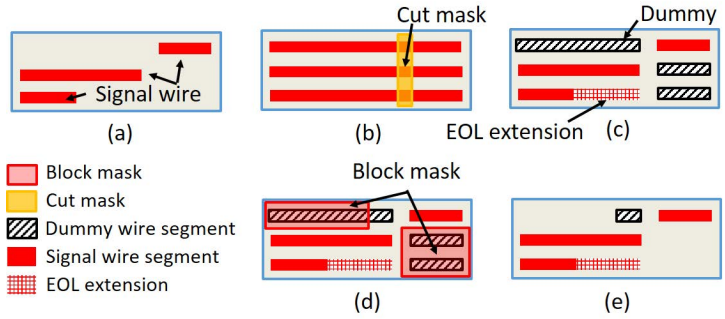


Figure 2.1: SAMP process: (a) post-route layout; (b) cut mask application; (c) layout after cut mask application; (d) block mask application; and (e) final layout after block mask application.

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 minimum pitch of 1D SAMP BEOL process. Thus, it is not possible to cover all dummy segments using one block mask. For example, in Figure 2.1(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 [204], and with support for a "selective"[4] variant of block-mask patterning technology.**

[4]I.e., a block mask approach that selectively removes metal lines according to the colors of metal. See Section 2.1.1 for the detailed description.

In advanced nodes, minimum metal density is crucial to chemical-mechanical polishing (CMP) [70]. In 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.[5] 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: *CORTEX 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, 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 [80] 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 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.

[5] Regarding the feasibility of the final pattern after dummy removal in terms of lithography, we note that [66] 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.

- 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 fundamental limits of benefit from emerging (cut and) block mask technology options.

In the remainder of this section, Section 2.1.1 provides background of cut and block mask technology, as well as related work. In Section 2.1.2, 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 2.1.3. Section 2.1.4 provides experimental results and analysis. We give conclusions and future research directions in Section 2.1.5.

2.1.1 Background

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.

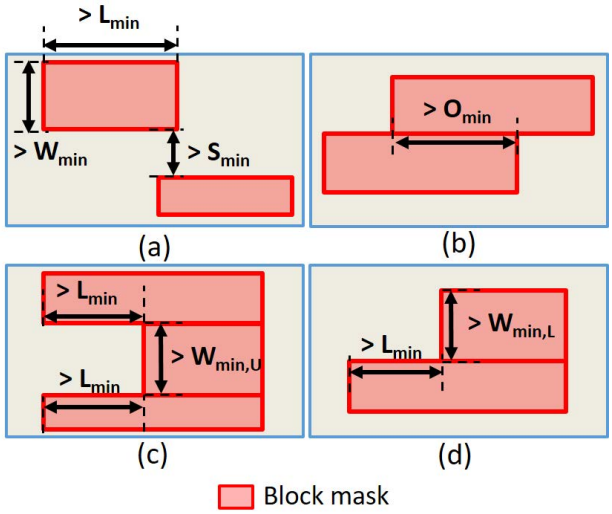


Figure 2.2: Block mask rules: (a) minimum width and length rules; (b) minimum overlap rule; (c) minimum U-shape rule; and (d) minimum L-shape rule.

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 2.2. For each rectilinear block mask shape, Figure 2.2(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 2.2(b), a minimum overlap rule applies. Figures 2.2(c) and (d) illustrate U-shape and L-shape constraints. Table 2.1 shows preliminary block mask rule sets (R1 - R8) for 193i and 193d patterning technologies.[6]

A *selective block approach* [122] 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 [122], 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 [82] is particularly applied to SADP/SAQP, where there are two types of wires that are created by mandrel and gap. Figure 2.3 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.[7] The final metal patterns are shown in blue color.

Figure 2.4 illustrates the difference between the selective block and non-selective block approaches. The red (resp. green) block mask in Figure 2.4(a) (resp. (b)) removes red (resp.

[6] 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 2.1 are from our collaborators at a leading technology development center / consortium.

[7] 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.

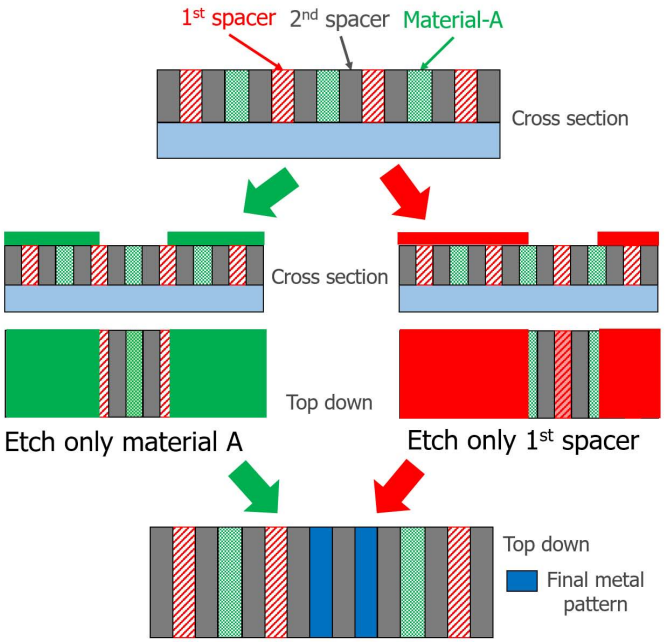


Figure 2.3: Illustration of the material selectivity-based block approach.

green) dummy segments, but acts as transparent to green (resp. red) segments. Note that without selectivity, the gray block mask shape becomes complex (Figure 2.4(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 2.4(a) and (b), block mask shapes can extend to non-target tracks, which is equivalent to doubling the metal pitch.

Table 2.1: 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

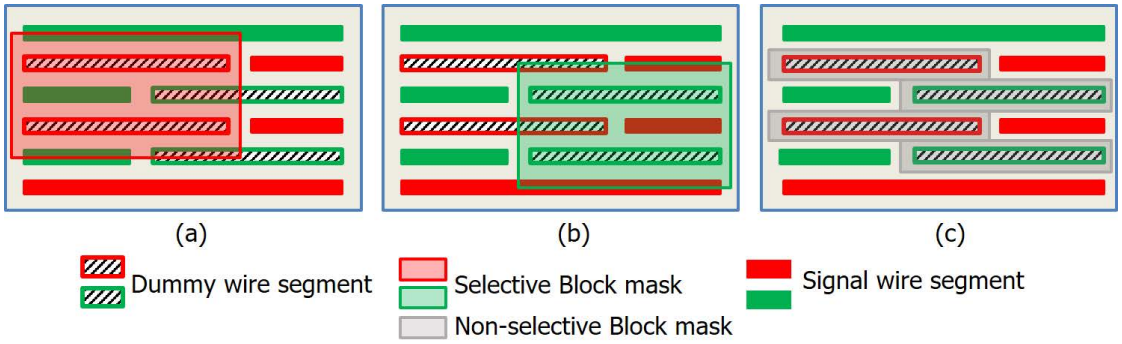


Figure 2.4: Comparison between selective block and non-selective block: (a) selective block mask in red removes only red segments; (b) selective block mask in green removes only green segments; and (c) a complex non-selective block mask is required to remove the same dummy segments.

Cut Mask Rules / Selectivity / LELE cut

Cut mask rules constrain shapes on the cut mask. As in [47][50][66][80][195], 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 2.5. Table 2.1 shows preliminary cut mask rule sets (R7, R8) for 193i patterning technologies.

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 2.6(a) and (b) illustrate the selective cut and LELE cut approaches, respectively. In Figure 2.6(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 2.6(b) shows LELE cuts. A minimum cut spacing rule is enforced separately for two green (resp. two red) cuts.

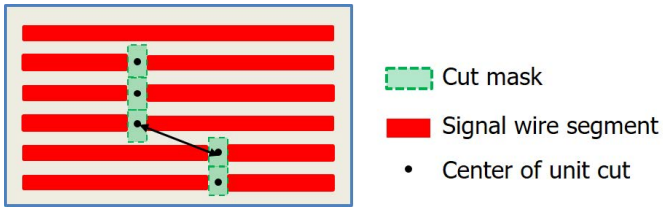


Figure 2.5: Cut mask rules: minimum spacing.

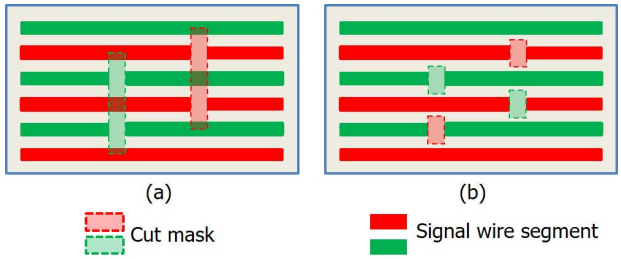


Figure 2.6: Comparison between selective cuts and non-selective LELE cuts. (a) Selective cut mask in red (resp. green) realizes EOL only for red (resp. green) segments, and is transparent to green (resp. red) segments. (b) Non-selective LELE cuts realize EOL for both colors.

Related Works

While selective block mask is a very recent concept [122], we may classify related works into three 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.

1D cut mask optimization. Zhang et al. [196] propose a shortest-path algorithm to resolve lithography hotspots in cut masks. Du et al. [50] propose an integer linear program to minimize total end-of-line (EOL) extension. Ding et al. [47] subsequently extend the methodology in [50] to reduce the runtime. Han et al. [80] extend the MILP formulation in [47] 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. [80] 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. [195] 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 [195] show a tradeoff between printability and wirelength increase, albeit without any hard design rule constraints. Ding et al. [47] propose an integer linear program formulation, with support of 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 timing impact of dummy fill patterns.

1D cut mask-aware routing optimization.[8] Su and Chang [174] propose a nanowire-aware router considering 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 [174] also suggest detailed routing with a cost function that considers cut sharing and EOL extension.

2D block mask-aware routing optimization. Fang [56] 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.

2.1.2 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 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.

[8] 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.

MILP Formulation for Block Mask Optimization

We now formulate the MILP problem for the block mask optimization problem. Table 2.2 shows notations that we use in our formulation.

Table 2.2: 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 2.7 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 2.7 since each shape may change according to the selected *block candidates*. 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.[9] 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 2.9(a).

Figure 2.7 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 2.7. 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.

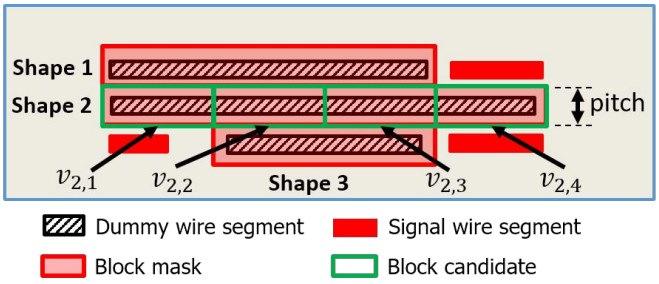


Figure 2.7: Shapes and block candidates for Shape 2.

[9] 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 2.1.4.

$$\text{Minimize:} \quad \sum_{p} m_p \tag{2.1}$$

Subject to:

$$\sum_{\substack{(i,j)\in B_{q,a} \\ (i',j')\in B_{q,b}}} v_{i,j} + (1 - v_{i',j'}) < |B_{q,a}| + |B_{q,b}|, \ \ \forall q \tag{2.2}$$

$$L + \sum_{i} (l_i - \sum_{j} r_{i,j} \cdot v_{i,j}) \geq d_{min} \tag{2.3}$$

$$\sum_{k\in K_p} \sum_{i,j} t^k_{i,j} \cdot (1 - v_{i,j}) \leq s_p + m_p, \ \ \forall p \tag{2.4}$$

$$m_p \geq 0, \quad \forall p \tag{2.5}$$

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* [200] (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.[10] 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., $m_p = \max(d_p - s_p, 0)$.[11] 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 (2.2) 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 form a violating block shape. In Constraint (2.2), $B_{q,a}$ (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 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 $B_{q,b}$.

[10] 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 *CORTEX M0*, *AES* and *JPEG*, respectively.

[11] 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 (2.4) and (2.5) 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.

We create a constraint to forbid each block mask pattern that forms a block mask rule violation. Figure 2.8 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:

$$v_{2,1} + v_{2,2} + (1 - v_{3,1}) + v_{3,2} + v_{4,2} + v_{4,3} < 6 \tag{2.6}$$

In Constraint (2.6), 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 (2.6). 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 sets B is in practice linear in the number of block candidates (shapes).[12]

Figure 2.8: Illustration of a U-shape block mask rule violation.

Constraints for local minimum metal density. Constraint (2.3) 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 2.1.3 below.

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

Constraints for timing-critical paths. Constraint (2.4) upper-bounds the timing degradation for timing-critical paths. Variable $t_{i,j}^k$ 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;[13] otherwise, $m_p > 0$. Note that we minimize m_p for all timing critical paths p by the objective. Constraint (2.5) limits m_p to be a non-negative number. We also note that Constraint (2.5) 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 (2.5), 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 (2.4). 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.

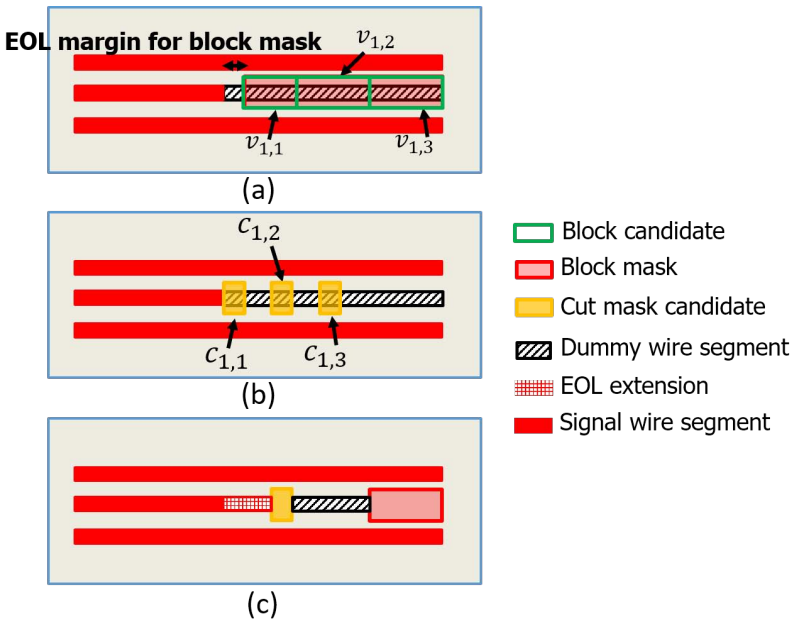


Figure 2.9: Cut and block mask co-optimization: (a) block candidates; (b) cut candidates; and (c) a possible final layout.

[13]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 11 above.)

MILP Formulation for Cut and Block Mask Co-Optimization

We extend the MILP in Section 2.1.2 by providing *cut candidates*. Figure 2.9 illustrates block and cut candidates with one possible final layout after cut and block mask application. Figures 2.9(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 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 2.9(c) shows the final layout when $v_{1,3}$ and $c_{1,2}$ are selected as the final block and cut candidate solutions, respectively.[14]

We now formulate the MILP problem for the cut and block mask co-optimization. Table 2.2 shows notations that we use in our formulation. Analogous to the block mask MILP in Section 2.1.2, 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 as in Section 2.1.2; 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 in Section 2.1.2.

Constraints for LELE cuts. In the case of non-selective LELE cuts, Constraint (2.8) 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 (2.8). 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 (2.10) and (2.11) prevent cut and block mask rule violations. Constraint (2.10) is the same as Constraint (2.2) in block mask optimization. Similar to Constraint (2.10) for block candidates, we enumerate sets of conflicting cut candidates and prevent them from co-existing with Constraint (2.11).

Constraints for EOL realization. Constraint (2.9) 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

[14]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.

signal segments, and must be split by cut of block to realize the EOL of the two signal segments. Thus, we enforce that at least one cut or block exists for shape i'.

$$\text{Minimize:} \quad \sum_{p} m_p \tag{2.7}$$

Subject to:

$$\sum_{f} c_{i,j}^{f} = c_{i,j_l}, \quad \forall i, j \tag{2.8}$$

$$\sum_{j} v_{i',j} + \sum_{j} c_{i',j} \geq 1, \quad \forall i' \tag{2.9}$$

$$\sum_{\substack{(i,j)\in B_{q,a} \\ (i',j')\in B_{q,b}}} v_{i,j} + (1 - v_{i',j'}) < |B_{q,a}| + |B_{q,b}|, \ \ \forall q \tag{2.10}$$

$$\sum_{\substack{(i,j)\in C_{q,a} \\ (i',j')\in C_{q,b}}} c_{i,j} + (1 - c_{i',j'}) < |C_{q,a}| + |C_{q,b}|, \ \ \forall q \tag{2.11}$$

$$e_{i,x_l(x_r)} \geq v_{i,j}, \quad \text{if } \ j_l(j_r) = x, \ \ \forall i \tag{2.12}$$

$$e_{i,x_l(x_r)} \geq c_{i,j}, \quad \text{if } \ j_l(j_r) = x, \ \ \forall i \tag{2.13}$$

$$e_{i,x_l(x_r)} \leq v_{i,j} + c_{i,j'}$$
$$\text{if } \ j_l(j_r) = x, \ \ j'_l(j'_r) = x, \ \ \forall i \tag{2.14}$$

$$e_{i,x_l} - \sum_{x'<x} e_{i,x'_l} - {e'}_{i,x_l} \leq 0,$$
$$e_{i,x_r} - \sum_{x'>x} e_{i,x'_r} - {e'}_{i,x_r} \leq 0, \ \ \forall i, \forall x \tag{2.15}$$

$${e'}_{i,x_l} \leq e_{i,x_l}, \quad {e'}_{i,x_r} \leq e_{i,x_r}, \ \ \forall i \tag{2.16}$$

$$\sum_{x_l} {e'}_{i,x_l} \leq 1, \quad \sum_{x_r} {e'}_{i,x_r} \leq 1, \ \ \forall i \tag{2.17}$$

$$L + \sum_{i} (l_i - \sum_{j} r_{i,j} \cdot v_{i,j}) \geq d_{min} \tag{2.18}$$

$$\sum_{k\in K_p} (\sum_{i,j} t^{k}_{i,j} \cdot (1 - v_{i,j}) + \sum_{i,x_l} {t'}^{k}_{i,x_l} \cdot e'_{i,x_l}$$
$$+ \sum_{i,x_r} {t'}^{k}_{i,x_r} \cdot e'_{i,x_r}) \leq s_p + m_p, \quad \forall p \tag{2.19}$$

$$m_p \geq 0, \quad \forall p \tag{2.20}$$

Constraints for EOL definition. Constraints (2.12) - (2.14) 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 (2.15) - (2.17) describe the methodology to find the leftmost (resp. rightmost) selected cut or block candidate. Constraints (2.15) - (2.16) 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 (2.15). Figure 2.10 demonstrates variable e'. In the figure, we assume that $c_{1,1} = 1$ and $v_{1,3} = 1$. e variables are computed in Constraints (2.21) by Constraints (2.12) - (2.14). Constraints (2.22) - (2.24) correspond to Constraint (2.15). Constraint (2.25) corresponds to Constraint (2.17). As a result, $e'_{1,2_l}$ becomes equal to one, which indicates that location $x = 2$ is the EOL, as shown in Figure 2.10.

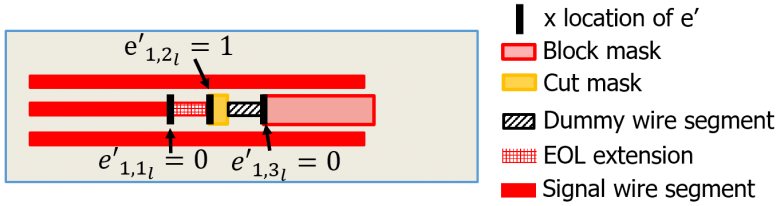


Figure 2.10: Illustration of binary variable e': cut candidate $c_{1,1}$ and block candidate $v_{1,3}$ are selected.

$$e_{1,1_l} = 0; \;\; e_{1,2_l} = 1; \;\; e_{1,3_l} = 1 \tag{2.21}$$

$$e_{1,1_l} - e'_{1,1_l} \leq 0 \tag{2.22}$$

$$e_{1,2_l} - e_{1,1_l} - e'_{1,2_l} \leq 0 \tag{2.23}$$

$$e_{1,3_l} - e_{1,1_l} - e_{1,2_l} - e'_{1,3_l} \leq 0 \tag{2.24}$$

$$e'_{1,1_l} + e'_{1,2_l} + e'_{1,3_l} \leq 1 \tag{2.25}$$

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, types of neighboring wire segments of the dummy / signal wires) [70][110]. 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 five 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; (iv) four tracks away; and (v) more than 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* [198] for parasitic RC extraction with *Cadence Quantus Extraction Solution v15.2* [199] techfiles provided by our collaborators at a leading technology consortium. Table 2.3 shows normalized capacitance increase per unit length for (grounded) EOL extension, and cases (i) - (iv) for (floating) dummy segments from Section 2.1.2.

Table 2.3: 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

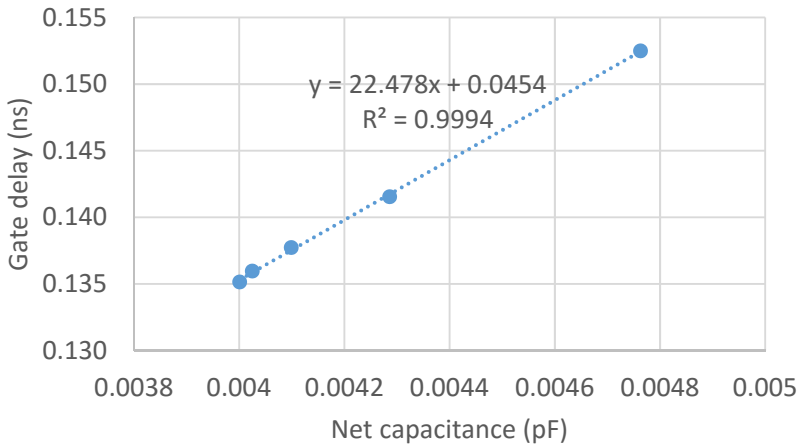


Figure 2.11: Gate delay vs. net capacitance for a specific gate instance.

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 caused by dummy wire segments, linear delay modeling shows good accuracy. We use *Cadence Tempus Timing Signoff Solution v15.2* [200] 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.[15] Figure 2.11 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* [200]. 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 2.12. The maximum errors are -4ps and -23ps (a negative value means optimistic) for stage delay and path delay, respectively. To compensate the errors, we add timing margin of 50ps in our ILP formulation for all studies reported below.

2.1.3 Overall Flow

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

Conflict List Enumeration

Minimum spacing violation. Algorithm 1 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 2.1.2). 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 candi-

[15] 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.

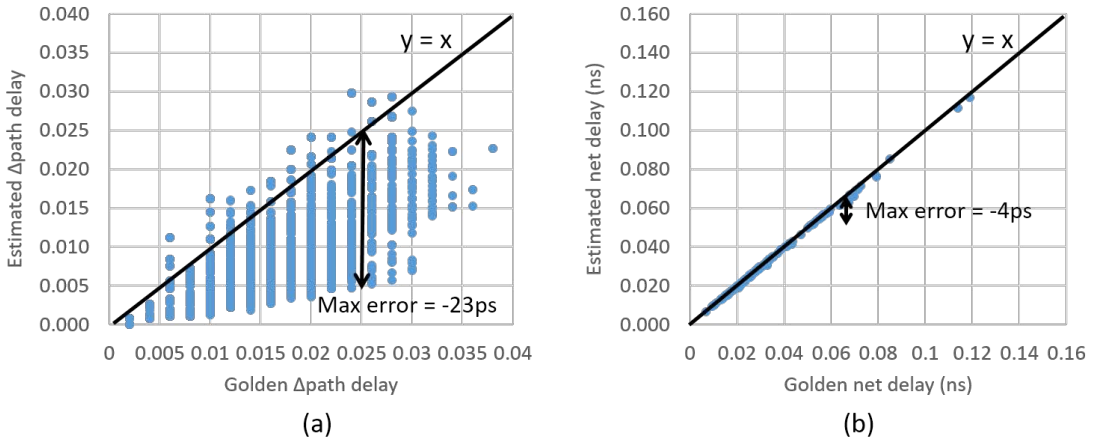


Figure 2.12: Comparison of timing results from Tempus (Golden) and our estimation (Estimated). (a) Path delay and (b) stage delay comparisons. The maximum errors are -4ps and -23ps for stage delay and path delay, respectively.

Algorithm 1 Enumeration for minimum spacing constraint.

```
for each block candidate pair (v_{i,j}, v_{i',j'}) ∈ V do
    if S(v_{i,j}, v_{i',j'}) < S_min then
        B_{q,a} ← {v_{i,j}, v_{i,j}};
        for each block candidate v_{k,l} located between v_{i,j} and v_{i',j'} do
            B_{q,b} ← B_{q,b} ∪ {v_{k,l}};
        end for
        q ← q + 1;
    end if
end for
```

date pair ($v_{i,j}$,$v_{i',j'}$) is separated. Figure 2.13 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 $B_{q,b} = \{v_{1,2}\}$.

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 [80].

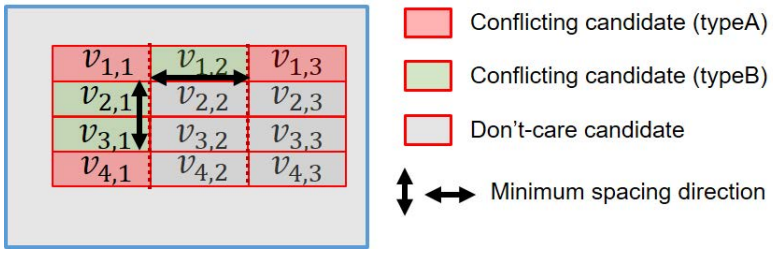


Figure 2.13: Illustration of conflict list enumeration for minimum spacing constraint, showing horizontally and vertically conflicting pairs.

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 2.14, 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 2.1.4.

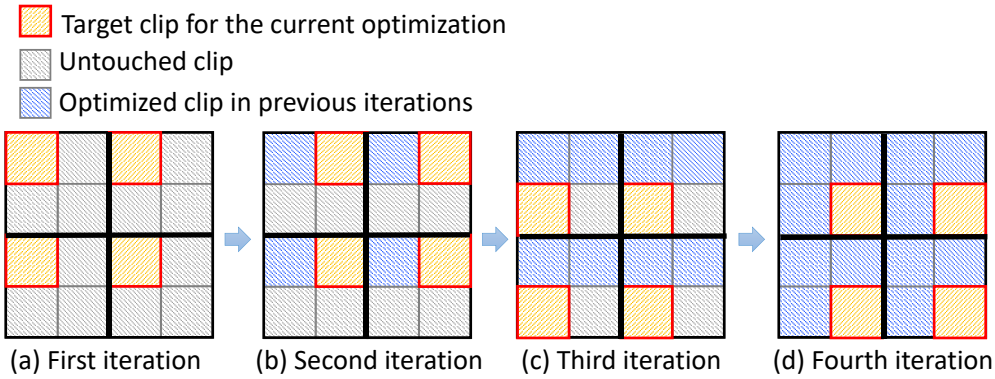


Figure 2.14: Distributed optimization: (a) - (d) respectively illustrate the first, second, third and fourth iteration in our approach. Since target clips (yellow) for an iteration do not share their boundaries with each other, each target is independently optimizable.

Overall Optimization Flow

Figure 2.15 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* [198] and *Cadence Tempus Timing Signoff Solution v15.2* [200].

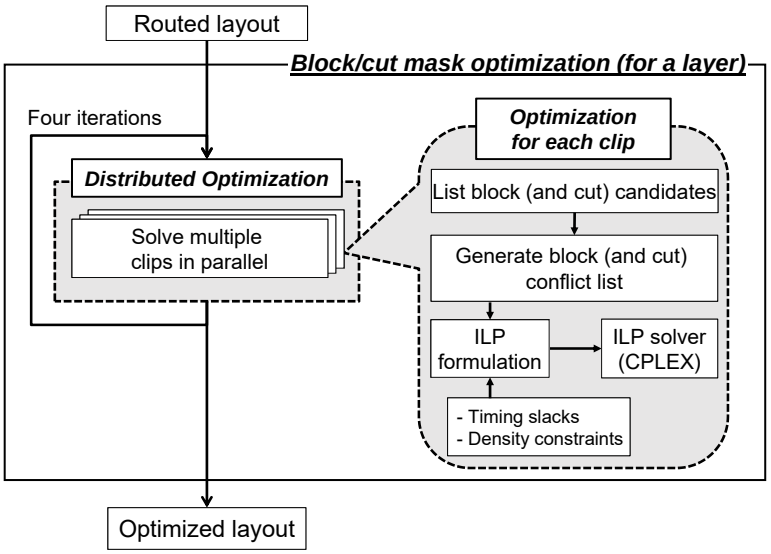


Figure 2.15: Overall optimization flow.

2.1.4 Experimental Setup and Results

Experimental Setup

We implement our optimizations in C++ with *OpenAccess 2.2.6* [217] to support LEF/DEF [209], and with *CPLEX 12.5.1* [203] as our MILP solver.[16] We evaluate our approach using two design blocks (*AES* and *JPEG*) from *OpenCores* [212], and an *Arm CORTEX M0 Core*

[16]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.

without memories. We synthesize the designs with *Synopsys Design Compiler vH-2013.03-SP3* [218] from RTL netlists and then perform placement and routing with *Cadence Innovus Implementation System v15.2* [198] using an IMEC N7 (i.e., $7nm$ foundry node) library [204]. 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.)

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.[17]

- **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.[18]

[17] 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 [195] 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 [196] 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.

[18] 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 2.2. 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 (2.4) in Section 2.1.2.

- **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 2.1). We use a loose 20% minimum metal density constraint to demonstrate the upper bound of performance impact from patterning technology.
- **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.[19]
- **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 approach.

The testcases are summarized in Table 2.4. Table 2.5 summarizes parameter settings for each type of experiment.

Table 2.4: Summary of testcases.

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

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

Table 2.5: 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 = *CORTEX 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 = *CORTEX 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 = *CORTEX 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	

Experimental Results

Table 2.7 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).[20] The best and worst cases serve as extreme, baseline data points for ExptB. Table 2.6 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 2.6: 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}
CORTEX 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 2.16(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 2.16(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.

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.

[20]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%.

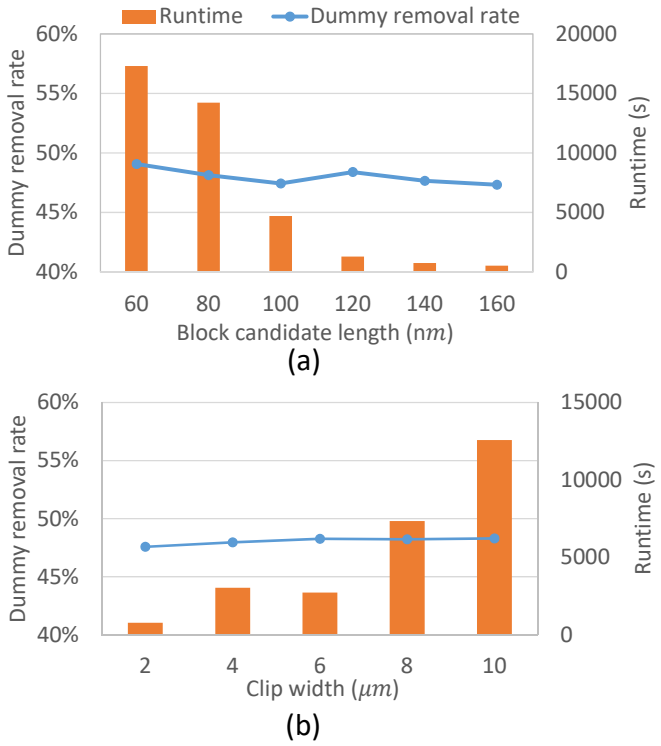


Figure 2.16: Sensitivity study results: sensitivity of dummy removal rate to (a) block candidate length and (b) clip size.

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 by 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.

Table 2.7: Overall experimental results. Values in parentheses denote percentage improvements (reductions) with respect to the worst case as described in Table 2.6. 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	*CORTEX 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	*CORTEX 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	*CORTEX 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	*CORTEX 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	*CORTEX 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	*CORTEX 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

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 power outcomes worsen.[21] 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% (resp. 52%, 38%) on average for a density constraint of 20% (resp. 30%, 40%). For *CORTEX 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 *CORTEX 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 compared to the LELE cut approach for *CORTEX M0* and *AES*. This is because selective cuts can be merged when they are aligned

[21] We see that for *CORTEX 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.

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.

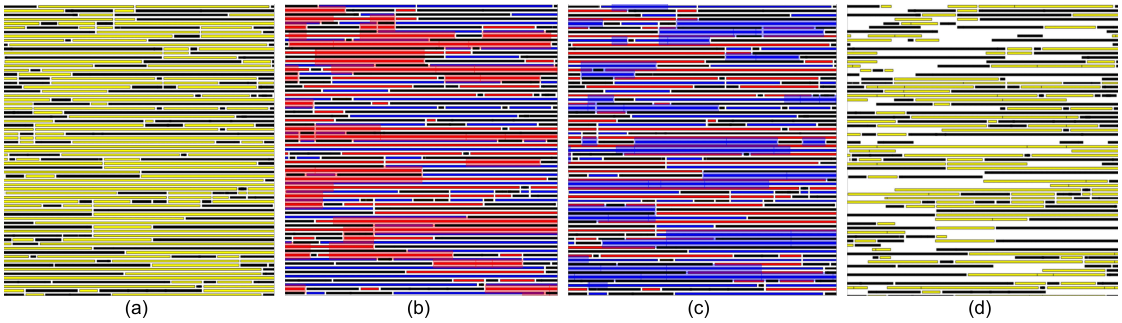


Figure 2.17: Layouts of M4 layer before and after dummy fill removal: (a) initial layout with dummy fill; (b) layout covered by the selective block mask (red); (c) layout covered by the selective block mask (blue); and (d) layout after timing-aware dummy fill removal with optimized selective block masks.

2.1.5 Conclusion

In this section, we first 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 further 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 and 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 furthermore 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.

2.2 Scalable Detailed Placement Legalization for Complex Sub-14nm Constraints

Continued technology scaling to the foundry $10nm$ node ($42nm$ minimum metal pitch, $36nm$ fin pitch) and below leads to more constraints in physical implementation. Not only do new metal-layer (back end of line, or BEOL) ground rules arise from multi-patterning techniques, but rules for device layers (front end of line, or FEOL) also become considerably more complex and restricted. For example, at the foundry $10nm$ node (henceforth referred to as **N10**), there are minimum width and area constraints for implant regions, as well as notch and jog width constraints for oxide diffusion (OD) regions. In older technology nodes, such layer rules were fairly benign: while of concern to the library cell designer, once the library cells were correctly designed, design rule violations (DRVs) could not occur during placement due to the correctness by construction of any non-overlapping cell placement.

Unfortunately, correctness by construction no longer holds for detailed placement at N10 and below. Cell sizes and minimum metal pitches have continued shrinking to stay on the Moore's Law density curve. However, patterning resolution in device (FEOL) layers has not kept pace due to challenges in device definition (e.g., ion implant) or lithographic variation (e.g., corner rounding). Thus, placing several 'legal' standard-cell layouts next to each other may cause violations of FEOL layer rules such as minimum implant width or area [103] rules. Such violations could in theory be prevented with larger cell area budgets (similar in spirit to how BEOL colorability, especially on the M1 layer, can be preserved) that permit correct-by-construction (or, more precisely, "composable-by-construction") cell layout styles. However, this runs counter to a core purpose of shrinking to the next node, and reduces the return on investment from enabling that node. Our present work proposes a new, final phase of detailed cell placement that can potentially maintain placement legality in the face of new N10 FEOL rules – without loss of density, routability or performance metrics.

2.2.1 N10 FEOL and Cell Placement Constraints

Figure 2.18 illustrates the layout of an inverter cell in the N10 node. The figure shows two fins each for PMOS and NMOS.[22] Source nodes of PMOS and NMOS are connected to M2 power/ground rails with M1. The input ***A*** is connected to the PMOS and NMOS gates using middle-of-line (MOL), a complementary metal layer below M1 that is used for intra-cell

[22] A more typical library in N10 might have 9-track (M2 tracks) cell height, and three fins each for PMOS and NMOS, with a gear ratio of M2:fin pitch anywhere from 7:6 to 4:3.

routing. The output ***Y*** is connected to the drain nodes of PMOS and NMOS. The FEOL layers which affect legal placement (i.e., in the context of other cells' placements) include implant layer, OD layer and poly, as follows.

- Implant layers, which indicate regions for ion implantation, decide the threshold (V_{th}) of transistors. Regions of the implant layer are typically aligned to the boundaries of standard cells.
- Oxide diffusion (OD) defines the active region of transistors.
- Dummy poly gates are inserted at the (vertical) standard cell boundaries to avoid edge device variability.

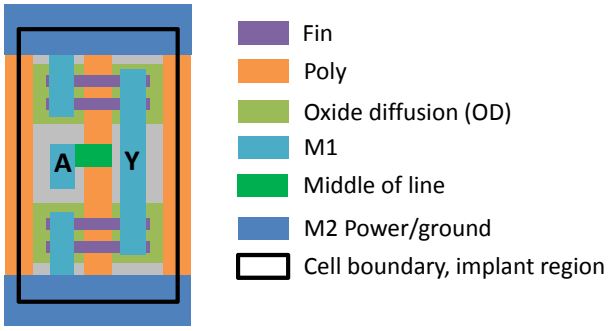


Figure 2.18: Illustration of inverter cell layout in N10 node.

Minimum implant width (IW) constraints. Minimum implant width (IW) constraints induce placement illegalities due to both inter- and intra-row IW violations, as shown in Figure 2.19(a). Below, we refer to the inter-row IW violation as being of type IW1. We refer to the intra-row IW violation as being of type IW2. An example of IW1 is shown in the figure, where two same-V_{th} cells are misaligned vertically and thus result in a narrow, "staircase" implant layer shape. IW2 occurs when a narrow cell is sandwiched between different-V_{th} cells, which results in a narrow implant region. Interestingly, the IW rules cause interactions between placement and sizing optimizations (e.g., V_{th}-swapping) that compromise the notion of, e.g., "post-route leakage optimization". This interaction has been recently studied in [103].

Minimum OD jog length (OW) constraints. Standard cells can have different oxide diffusion (OD) region heights according to functionality, drive strength, etc. When cells with different OD heights abut, OD jogs can result as shown in Figure 2.19(b). This is forbidden in N10 and below due to lithographic corner rounding and the consequent device performance variability, e.g., under misalignment. In N10, a minimum OD jog length rule is violated if the jog

length is less than a given minimum value. Introducing sufficient spacing between the violating cells can cure the OD jog violation.

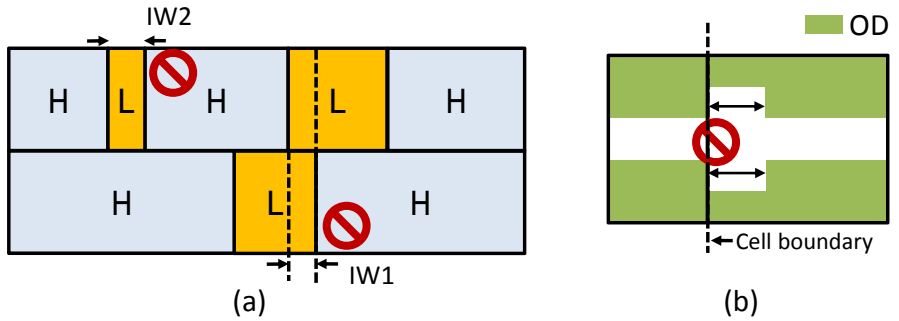


Figure 2.19: (a) Examples of minimum implant width violations [209]. (b) The design rule for OD jogs.

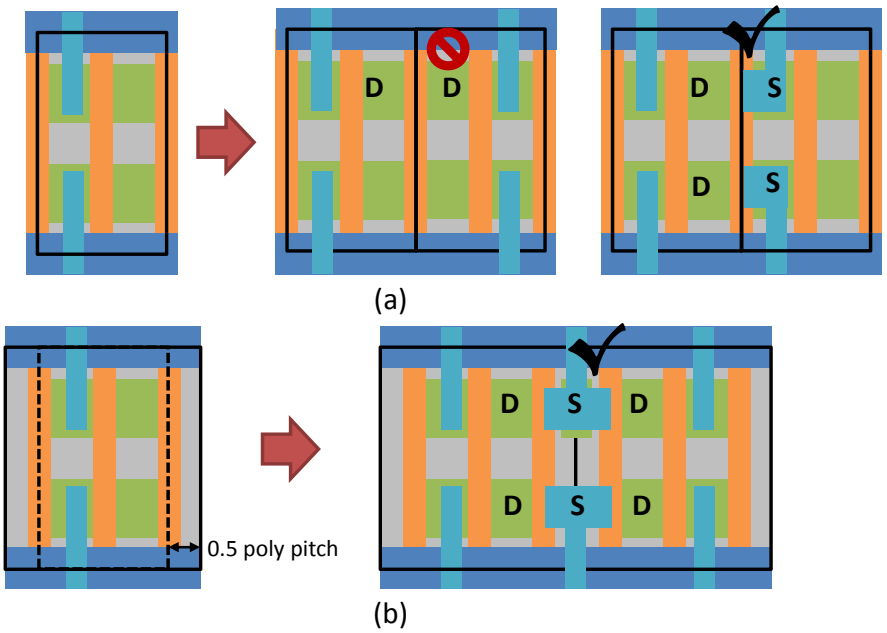


Figure 2.20: (a) Drain-drain abutment violation with an example standard cell layout. (b) Use of dummy poly gates in the library design style can avoid DDA violation in a correct-by-construction manner.

Drain-drain abutment (DDA) constraints. Dummy poly gates create extra dummy transistors connected to logic transistors within standard cells. The dummy transistors can induce leakage power and logic failure if they are not fully turned off. Hence, gate and source nodes of dummy transistors must be tied off to power/ground rails; in particular, if two drain nodes are abutted, an extra dummy poly gate is needed to create an additional source node to be tied up with power/ground rails. The recent work of Du and Wong [49] studies cell instance flipping as a way of mitigating this issue in detailed placement. Figure 2.20(a) depicts the DDA problem.

The leftmost diagram shows an example inverter layout, and the middle and right diagrams respectively show DDA and no-DDA cases. To avoid the DDA problem, we can consider two approaches [2]: (i) a smart detailed placement with comprehension of DDA; and (ii) standard cells with embedded dummy poly gates as shown in Figure 2.20(b). With approach (ii), the width overhead for each cell is one poly pitch. (We study the area-DRV tradeoff between approaches (i) and (ii) in Section 2.2.4 below.)

This Section

As noted above, [103] and [49] have respectively made initial studies of IW- and DDA-induced placement issues. However, to our knowledge, there is no existing work that addresses all the issues above simultaneously in detailed placement. Popular techniques used for conventional placement legalization, including graph-based, dynamic programming-based, etc. methods, appear ill-suited to handling of complex FEOL layer rules at N10 and below. For example, previous techniques have focused on removing overlaps between cells while maintaining the ordering of cells within a row, while minimizing half-perimeter wirelength or placement perturbation. Such previous works are largely single-row-based, and are applied row by row. Thus, they do not capture *inter-row* constraints such as IW1 that arise in N10. Furthermore, a number of implicit assumptions made by placement legalizers are broken when placement correctness by construction no longer holds, e.g., when more than two cells can interact and create DRVs. This challenges the use of dynamic programming frameworks, since decomposition into independent placement subproblems is no longer obvious. Finally, filler cell insertion has not previously been a concern of placement legalizers, but in N10 the filler cells can cause additional implant layer rule violations.

In this section, we propose a mixed integer-linear programming (MILP)-based placement legalization that considers complex N10 FEOL-layer design rules including minimum implant width, minimum oxide diffusion jogs and drain-drain abutment. We also propose a distributable optimization approach based on partitioning a given placement into many windows of cells, with each window being independently optimizable. The main contributions of our work are summarized as follows.

- We formulate as an MILP a placement problem that addresses new DRVs caused by complex N10 design rules. In contrast to previous approaches, our formulation captures new *inter-row* violation types. We further implement our solution approach in a prototype tool, **DFPlacer**.

- DFPlacer handles whitespace in the problem formulation and determines filler cell insertions to solve implant width constraint violations.
- We propose a distributable optimization based on partitioning of an input placement into windows of cells, and demonstrate that our optimization is scalable via this mechanism.
- We implement our proposed methods in C++ with OpenAccess 2.2.43 [217] and incorporate them into a commercial tool-based placement and routing (P&R) flow for evaluation.
- A further study provides insight into timing and area impacts of the dummy poly gate library cell strategy, using two kinds of libraries: (i) standard cells with dummy poly gates (drain-drain abutment violation-free) and (ii) standard cells without dummy poly gates.

The remainder of this section is organized as follows. In Section 2.2.2, we review relevant prior work. To address N10 rules, we formulate an MILP in Section 2.2.3 and describe our distributable optimization strategy in Section 2.2.3. Section 2.2.4 provides experimental results and analysis. We give conclusions and future research directions in Section 2.2.5.

2.2.2 Related Work

We now summarize relevant previous works on detailed placement and placement legalization.

Dynamic programming-based approaches. Dynamic programming (DP), typically for a single cell row, has been used by a number of authors. Kahng et al. [107] use DP to legalize placement of a single row with various minimization objectives: total perturbation, maximum perturbation, and wirelength. A shortest-path algorithm is applied to a directed acyclic graph constructed from the input ordering of cells. Gupta et al. [71] perform detailed placement optimization to enable sub-resolution assist feature insertion for improved manufacturability. A DP-based single row placement achieves this *assist-feature correctness* (AFCorr) while minimizing (timing criticality-weighted) perturbations of cell locations. Subsequent work addresses a 2D formulation that considers both horizontal and vertical interactions between adjacent cells [72]. The *2D AFCorr* approach uses DP in which vertical and horizontal costs are calculated with restricted perturbations. Hur and Lillis [94] propose *optimal interleaving* for intra-row optimization in detailed placement. Their work splits the cells of a single row into two groups with a given window size, and the two sequences are optimally interleaved via DP while preserving the initial relative ordering of cells in each group. At the global placement level, cells are assigned to bins and optimized via *relaxation-based local search*.

Integer Linear Programming (ILP)-based approaches. Another important class of previous methods is based on integer linear programming. Ramachandaran et al. [155] apply branch-and-price for improved scaling of the placement optimization. Li and Koh [125] propose ILP-based detailed placement approaches using *placement site variables*. Dantzig-Wolfe decomposition is applied to improve scalability, and single-cell-placement (SCP) variables enable grouping and mapping of placement site variables into patterns. The extension [126] supports mixed-size circuits and improves runtime by bounding solution spaces. In our present work, we begin with the MILP model of [125] [126], extending it to provide the first-ever comprehensive support (to our knowledge) of N10-relevant design rules such as minimum implant width, diffusion jogs and drain-drain abutment.

N10 design rules-aware placement. Du and Wong [49] address the abutment of source and drain in FinFET-based cell placement (i.e., for the foundry $14nm$ node onward), where the DDA constraint becomes prominent. The authors use cell flipping and adjacent-cell swapping as underlying operations for detailed placement perturbation that minimizes drain-drain abutments. As in [107], the authors of [49] apply a shortest-path algorithm with their proposed graph model, in which each operation and the violations are modeled as nodes and node/edge costs, respectively. However, the approach only swaps and flips cells within a single row, and does not handle interactions between placement rows. Hence, the optimization is made with respect to a highly restricted portion of the overall detailed placement solution space. Moreover, DDA-related optimization cannot be performed in isolation at the N10 node: many other neighborhood-related constraints (e.g., constraints for implant and oxide diffusion (OD) layers) have interactions with, and constrain, the drain-drain abutment solution. In our present work, we handle neighborhood-related constraints along with drain-drain abutment, with a larger solution space that includes multiple rows.

2.2.3 Our Approach

Problem Formulation

We now formulate a MILP for our detailed placement problem to address N10-related design rules including IW, DDA and OW in Section 2.2.1. Our notation is described in Table 2.8.

Table 2.8: Notations.

Notation	Meaning
C, R, Q	sets of cells, rows, columns
f_c	a binary indicator of whether cell c is flipped
$x(y)_{c,init}$	initial x (y) coordinate of cell c
s_{crq}	a binary indicator of whether cell c occupies site (r, q)
K_c	a set of candidate states of cell c
λ_c^k	a binary indicator of the k^{th} candidate state for cell c
x_c^k, y_c^k	x and y coordinates 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
m_{rq}	inter-row variable for IW1
h_{rq}	intra-row variable for IW2
W	minimum implant width (unit: site)

$$\text{Minimize:} \quad \sum_{c \in C} (|x_c - x_{c,init}| + |y_c - y_{c,init}|) \tag{2.26}$$

Subject to:

$$\sum_{k \in K_c} \lambda_c^k = 1, \qquad \forall c \in C, \;\; \lambda_c^k \in \{0, 1\} \tag{2.27}$$

$$f_c = \sum_{k \in K_c} f_c^k \lambda_c^k \tag{2.28}$$

$$x_c = \sum_{k \in K_c} x_c^k \lambda_c^k, \;\; y_c = \sum_{k \in K_c} y_c^k \lambda_c^k, \;\; \forall c \in C \tag{2.29}$$

$$s_{crq} = \sum_{k \in K_c} s_{crq}^k \lambda_c^k, \quad \forall c \in C \tag{2.30}$$

$$\sum_{c \in C} s_{crq} \leq 1, \qquad \forall q \in Q, r \in R \tag{2.31}$$

For a given input layout, our objective is to minimize the sum of cell displacements while achieving a legal placement with respect to given N10 design rules. We assume a given perturbation range for each cell g ($g.l$, $g.r$, $g.t$ and $g.b$ are the maximum allowed displacements of the cell in the left, right, top and bottom directions, respectively); a cell cannot move beyond its given perturbation range. Thus, we have a limited number of possible states (locations and orientations) within g, for each cell. To represent each candidate state for a cell, we adopt the single-cell-placement (SCP) model of [126]. The binary SCP variable λ_c^k represents a candidate state k for a cell c. The variable λ_c^k is associated with the location and orientation of cell c,

e.g., x_c^k, y_c^k, and f_c^k, which are pre-defined values. Also, s_{crq}^k, where $r \in |R|$ and $q \in |Q|$, is pre-defined for λ_c^k.

From Constraint (2.27), exactly one state is chosen for cell c among multiple candidate states $\lambda_c^k, k \in K_c$; this determines the location and orientation of c. Constraints (2.28), (2.29) and (2.30) determine the final x, y of cell c and s_{crq} for $r \in |R|, q \in |Q|$ from a selected candidate site for cell c. To ensure a legal placement (no overlap), Constraint (2.31) forces a site at (r, q) to be occupied by at most one cell. In addition to the basic formulation, we add extra constraints to address OW, DDA, IW1 and IW2 rules, as follows.

OW and DDA constraints. To handle OW and DDA constraints, we pre-characterize all adjacency conditions which violate OW and/or DDA for each library cell pair. We note that our pre-characterization considers the orientations of cells (i.e., the adjacency conditions change depending on the orientations of cells). We then generate a set P of forbidden pairs of λ_c^k. Based on P, we formulate Constraint (2.32) for every forbidden pair $(\lambda_{c_1}^i, \lambda_{c_2}^j)$.

$$\lambda_{c_1}^i + \lambda_{c_2}^j \leq 1 \qquad \text{where } c_1, c_2 \in C, (\lambda_{c_1}^i, \lambda_{c_2}^j) \in P \tag{2.32}$$

IW1 and IW2 constraints. IW1 violations occur across rows when vertically-adjacent same-V_{th} layers form a narrow staircase shape with width less than the minimum implant width (see Figure 2.19(a)). To handle IW1, we define a 0-1 *inter-row variable,* m_{rq}*,* that indicates whether the site at (r, q) (row r and column q) and the site at $(r + 1, q)$ have the same V_{th} ($m_{rq} = 1$) or not ($m_{rq} = 0$). Figure 2.21(a) illustrates the m_{rq} variables and IW1 constraints. As shown in the figure, if a 0-1 sequence of m values is found (e.g., m_{12}, m_{13}), the implant region has a staircase shape, and hence $(W - 1)$ consecutive m variables must be one. Thus, we formulate constraints that, if $m_{r(q-1)} = 0$ and $m_{rq} = 1$, force at least W consecutive inter-row variables $m_{rq} = \ldots = m_{r(q+W-1)} = 1$, so as to satisfy IW1 (e.g., $m_{13}, m_{14}, m_{15} = 1$ where $W = 3$, in Figure 2.21(a)).

IW2 violations occur when small-width cells are sandwiched in between different-V_{th} cells in the same row. Similar to how we handle IW1, we define a 0-1 *intra-row variable*, h_{rq}, that indicates whether the site at (r, q) and the site at $(r, q+1)$ have the same V_{th} ($h_{rq} = 1$) or not ($h_{rq} = 0$), as shown in Figure 2.21(b). If $h_{rq} = 0$, i.e., sites (r, q) and $(r, q + 1)$ have different V_{th}, we force $(W - 1)$ consecutive binary variables $h_{r(q+1)} = \ldots = h_{r(q+W-1)} = 1$, so as to have at least W consecutive same-V_{th} sites. Figure 2.21(b) shows the case when $h_{rq} = 0$, where $r = 1, q = 2, W = 3$.

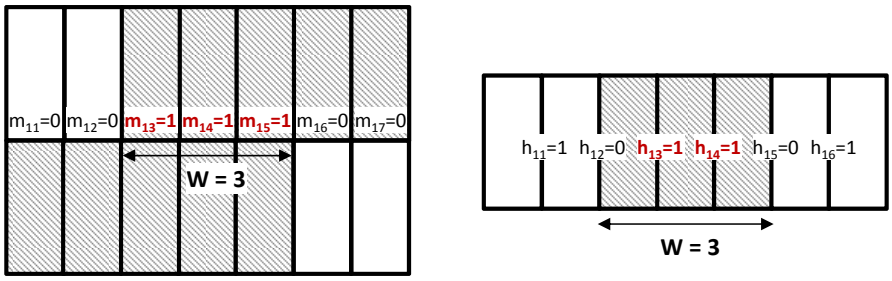


Figure 2.21: (a) Inter-row variable m_{rq} for IW1. (b) Intra-row variable h_{rq} for IW2. The color (gray and white) of regions indicates V_{th}.

The generalized constraints for IW1 and IW2 are as follows:

$$m_{r0} = 0, h_{r0} = 0 \qquad 1 \leq r < |R| \tag{2.33}$$

$$m_{rq} + (1 - m_{r(q+1)}) + y_{r(q+2)} \geq 1$$
$$0 \leq q < |Q| - W,\ 1 \leq r < |R| \tag{2.34}$$

$$y_{rq} \leq m_{r(q+w)}$$
$$2 \leq q \leq |Q| - 2,\ 1 \leq r < |R|,\ 0 \leq w < W - 1 \tag{2.35}$$

$$h_{rq} + z_{rq} \geq 1$$
$$0 \leq q < |Q| - W,\ 1 \leq r < |R| \tag{2.36}$$

$$z_{rq} \leq h_{r(q+1+w)}$$
$$2 \leq q \leq |Q| - 2,\ 1 \leq r < |R|,\ 0 \leq w < W - 1 \tag{2.37}$$

- Constraint (2.33) initializes the leftmost m and h variables where $q = 0$.
- Constraint (2.34) detects the condition of $m_{rq} = 0$ and $m_{r(q+1)} = 1$, and forces $y_{r(q+2)} = 1$.
- When $y_{r(q+2)} = 1$, Constraint (2.35) forces $(W - 1)$ consecutive binary variables $m_{r(q+2)} = \ldots = m_{r(q+2-(W-2))} = 1$.
- Constraint (2.36) detects the condition of $h_{rq} = 0$ and forces $z_{rq} = 1$.
- When $z_{rq} = 1$, Constraint (2.37) forces $(W - 1)$ consecutive binary variables $h_{r(q+1)} = \ldots = h_{r(q+(W-1))} = 1$.

We now describe our method of obtaining inter- and intra-row variables (m_{rq} and h_{rq}). We first set the V_{th} of cell c as the binary vector $\vec{k}_c$. The length of $\vec{k}_c$ is determined by $\lceil log_2(n_{V_{th}} + 1) \rceil$ where $n_{V_{th}}$ is the number of available V_{th} options. For example, if we have three V_{th} options, then $\vec{k}_c$ is $\{k_c^1\ k_c^2\}$. Concretely, the binary vectors $\{0\ 1\}$, $\{1\ 0\}$ and $\{1\ 1\}$ represent HVT, NVT and LVT, respectively. We then define the V_{th} *variable* $\vec{v}_{rq}$ as a binary vector variable $\{v_{rq}^1\ v_{rq}^2\}$ indicating the V_{th} of the site (r, q). Given that $m_{rq} = 1$ if $\vec{v}_{rq} = \vec{v}_{(r+1)q}$, we add the following constraint to obtain m_{rq}:

$$m_{rq} = \overline{(v_{rq}^1 \oplus v_{(r+1)q}^1) + (v_{rq}^2 \oplus v_{(r+1)q}^2)} \tag{2.38}$$

Constraint (2.38) is rewritten in our MILP formulation, using binary variables u_1, u_2 and $\overline{m_{rq}}$, as follows:

$$\begin{aligned}
&\overline{m_{rq}} + m_{rq} \leq 1; \\
&\overline{m_{rq}} \leq u_1 + u_2;\ \overline{m_{rq}} \geq u_1;\ \overline{m_{rq}} \geq u_2; \\
u_1 \leq v_{rq}^1 + v_{(r+1)q}^1;&\ u_1 \geq v_{rq}^1 - v_{(r+1)q}^1; \\
u_1 \geq v_{(r+1)q}^1 - v_{rq}^1;&\ u_1 \leq 2 - v_{rq}^1 - v_{(r+1)q}^1 \\
u_2 \leq v_{rq}^2 + v_{(r+1)q}^2;&\ u_2 \geq v_{rq}^2 - v_{(r+1)q}^2; \\
u_2 \geq v_{(r+1)q}^2 - v_{rq}^2;&\ u_2 \leq 2 - v_{rq}^2 - v_{(r+1)q}^2
\end{aligned} \tag{2.39}$$

Similarly, h_{rq} can be formulated as follows:

$$h_{rq} = \overline{(v_{rq}^1 \oplus v_{r(q+1)}^1) + (v_{rq}^2 \oplus v_{r(q+1)}^2)} \tag{2.40}$$

We also consider whitespace (empty sites), which can be filled with filler cells. We have the freedom to choose V_{th} of filler cells to satisfy IW1 and IW2 constraints. To exploit this flexibility, we define a binary vector variable $\vec{e}_{rq} = \{e_{rq}^1 e_{rq}^2\}$ which indicates V_{th} of the site at (r, q). From variables $\vec{k}_c$, s_{crq} and $\vec{e}_{rq}$, the binary vector variable $\vec{v}_{rq}$ is defined as follows:

$$\vec{v}_{rq} = \sum_{c \in C} \vec{k}_c \cdot s_{crq} + \vec{e}_{rq} \tag{2.41}$$

Thus, $\vec{v}_{rq}$ is determined by either $\sum_{c \in C} \vec{k}_c \cdot s_{crq}$ or $\vec{e}_{rq}$. We add a constraint below for $\vec{e}_{rq}$:

$$e^1_{rq} \leq 1 - \sum_{c \in C} s_{crq}; \;\; e^2_{rq} \leq 1 - \sum_{c \in C} s_{crq} \tag{2.42}$$

Constraint (2.42) states that if a site is occupied by any cell, $\vec{e}_{rq} = 0$. Then, Constraint (2.41) becomes independent of $\vec{e}_{rq}$. Otherwise, Constraint (2.41) becomes $\vec{v}_{rq} = \vec{e}_{rq}$.

Analysis of the number of variables and constraints. The number of variables and constraints depends on the number of sites in a target window ($|R| \cdot |Q|$), the number of instances ($|C|$) and the size of the perturbation range ($g.size = (g.l + g.r) \cdot (g.t + g.b)$).

- The number of variables s_{crq} is $|C| \cdot |R| \cdot |Q|$.
- The number of variables x_c, y_c is (each) $|C|$; the number of variables $x^k_c, y^k_c, \lambda^k_c$ is (each) $g.size \cdot |C|$.
- The number of inter-/intra-row variables m, h is (each) $|R| \cdot |Q|$.
- The number of variables v and e is (each) $n \cdot |R| \cdot |Q|$, where n is $\lceil log_2(n_{V_{th}} + 1) \rceil$.
- The numbers of Constraints (2.27), (2.29), (2.30) and (2.31) are $|C|$, $|C|$, $|C| \cdot |R| \cdot |Q|$ and $|R| \cdot |Q|$, respectively.
- The number of Constraint (2.32) is $g.size \cdot |C|^2$.
- The number of Constraints (2.34), (2.35), (2.36) and (2.37) is (each) $|R| \cdot |Q|$.

Overall Flow

We implement our flow in C++ with *OpenAccess 2.2.43* [217] to support LEF/DEF [209], and with *CPLEX 12.5.1* [203] as our MILP solver. Figure 2.22 shows the overall flow of our tool, which we call DFPlacer. DFPlacer has two optimization stages: global and local optimization. In the global optimization, we split the given routed layout T uniformly into a set of windows D and optimize each of the windows $d \in D$ in parallel. We use a fixed boundary margin b for each window to enable independent optimization among windows. In the local optimization, we generate a new window for each remaining violation $\gamma \in \Gamma$ such that the

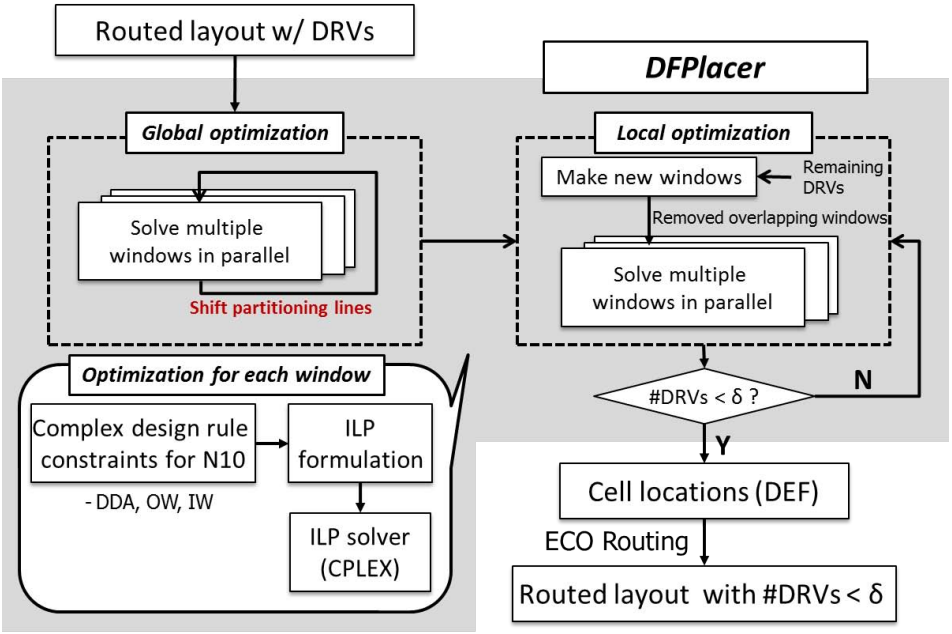


Figure 2.22: Overall flow of detailed placement legalization.

violation is located at the center of the window. We then remove overlapping windows so that no window affects another. With the new set of windows D', we optimize each window $d' \in D'$ again in parallel. The output cell location solution is saved in DEF file format, which can be fed into a commercial P&R tool. We perform ECO routing with the solution and finally obtain a new layout T_{opt} with number of violations $|\Gamma|$ less than the given target number δ.

Algorithm 2 gives further details of our optimization flow. In Lines 2-6, the global optimization phase solves D in parallel with a small perturbation range g (e.g., $g.l = 4$, $g.r = 4$, $g.t = 1$ and $g.b = 1$ sites) of cells. This distributable method overcomes the runtime limitation of MILP-based approaches and fixes more than 90% of initial $|\Gamma|$ (see Figure 2.24). In Line 3, we first partition a given routed T into D, and we solve each $d \in D$ in parallel using OpenMP [213] in Line 4. We set a window width $z.w$ as 47 sites, and a window height $z.h$ as nine cell rows in our experiments.[23] When running optimizations for the windows in parallel, we set the vertical (resp. horizontal) boundary margin $b.v$ (resp. $b.h$) so that the solution of one window can be isolated from the solutions of neighbor windows. We set $b.v$ as the minimum implant width W and $b.h$ as two cell row heights. Figure 2.23 shows the boundary margin in green color. We then update the MILP solutions to the layout T.

[23]Window size affects the tradeoff between the number of remaining violations $|\Gamma|$ after global optimization and the runtime of global optimization. Our studies of different window sizes (i.e., $z.w$ ranging from 40 sites to 55 sites and $z.h$ ranging from five cell row heights to 11 cell row heights) find that for a sample design (*JPEG*) a width of 47 sites and a height of nine cell row heights empirically achieves a good outcome ($< 10\%$ of initial $|\Gamma|$) with relatively small runtime ($< 30\ min$). We therefore use this window size in all of our reported experiments.

Algorithm 2 Overall flow of DFPlacer.

Procedure $DFPlacer(T, U, z, b, g, \delta)$
Input : Layout T, set of design rule constraints U, window size z, boundary margin b, perturbation range g, target number of DRVs δ
Output : Layout T_{opt} with $|\Gamma| < \delta$

// Global optimization
for $i = 1$ to 3 **do**
  A set of windows $D \leftarrow Partition(T, i, z, b, g)$;
  Solve all MILP instances for windows D in parallel;
  Update MILP solutions to T;
end for
// Local optimization
$\Gamma \leftarrow getDRV(T, U)$;
while $|\Gamma| < \delta$ **do**
  $D \leftarrow \emptyset$;
  for all $\gamma \in \Gamma$ **do**
    $d \leftarrow MakeNewWindows(T, \gamma, z, b, g)$;
    $D \leftarrow D \cup d$;
  end for
  $D' \leftarrow NonOverlapWindows(D)$
  Solve all MILP instances for windows D' in parallel;
  Update MILP solutions to T;
  $\Gamma \leftarrow getDRV(T, U)$;
  if $|\Gamma|$ is the same as $|\Gamma|$ in the previous iteration **then**
    $IncreaseWindow(z)$;
    $IncreasePerturb(g)$;
  end if
end while
$T_{opt} \leftarrow T$;
return T_{opt};

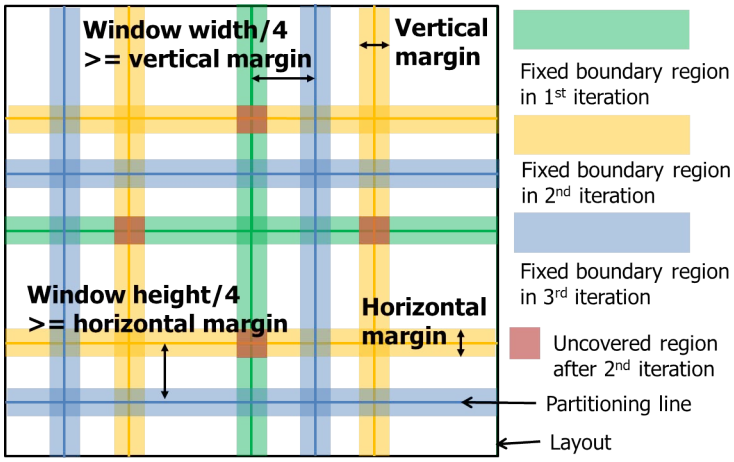


Figure 2.23: Partitioning of layout for parallel global optimization.

Since the fixed boundary cells corresponding to b of the first iteration can contain DRVs which are not fixed in the first iteration, we perform a second iteration with a new partitioning that is shifted by half of $z.w$ and $z.h$ in the x- and y-directions, respectively; these are shown in yellow color in Figure 2.23. We then partition the current T into a new D and solve the corresponding MILP instances to fix the violations Γ remaining from the first iteration. We then update the MILP solutions to T. Even after the first and second iterations, DRVs could still exist in the intersections of the fixed boundary regions (red color in Figure 2.23). To fix the Γ in the uncovered intersection region, we perform a third iteration that has new partitioning lines shifted by a quarter of $z.w$ and $z.h$ in x-direction and y-direction. Note that a quarter of $z.w$ and of $z.h$ should respectively be larger than or equal to $b.v$ and $b.h$. This ensures that the windows of the third iteration contain the uncovered regions, such that the fixed boundary region of the third iteration is not overlapped with the uncovered intersection region.

The small window size and perturbation range used in global optimization restricts the solution space, potentially leading to infeasible solutions for certain windows. To fix the remaining Γ, we perform the local optimization in Lines 8-24. For each DRV, the function $MakeNewWindows()$ creates a new window whose center is the DRV point. In Line 15, $NonOverlapWindows()$ picks a set of disjoint windows D' to process in parallel. We then update the solutions to the current T and check the remaining Γ (Lines 16-17). In Lines 19-22, if the current $|\Gamma|$ is the same as the $|\Gamma|$ of the previous iteration, we increase $z.w$ by 10 sites and $z.h$ by one cell height. For perturbation range, we increase $g.l$, $g.r$, $g.t$ and $g.b$ by 2, 2, 1 and 1 sites, respectively. When the current $|\Gamma|$ is less than the target number of DRVs δ, we save the current T as T_{opt} and terminate the optimization. In our experiment, we set δ as 1% of initial $|\Gamma|$.

2.2.4 Experimental Setup and Results

Experimental Setup

We evaluate *DFPlacer* using two open-source designs (*AES_**, *JPEG_**) [212], an *Arm CORTEX M0* without memories (*CORTEX M0_**) and a *3×CORTEX M0* without memories (*CORTEX M0×3_**). We synthesize these testcases from RTL, and perform P&R with an abstracted $7nm$ dual V_{th} library. Our RTL-to-layout flow uses *Synopsys Design Compiler vH-2013.03-SP3* [218] and *Cadence Encounter Digital Implementation System v13.1* [198] for logic synthesis and P&R, respectively. All experiments are performed with 40 threads on a $2.6GHz$ Intel Xeon E5-2690 dual-CPU server. In principle, the number of threads could be as large as the number of layout windows.

Table 2.9: Summary of testcases.

Design	#Inst	LVT (%)	Util. (%)	WL (μm)	Area (μm^2)	WSS (ps)	WHS (ps)
CORTEX M0_nd	8260	52	77	114685	7668	38	0
AES_nd	12147	54	78	142294	8894	90	0
CORTEX M0×3_nd	27248	56	80	392540	24463	126	0
JPEG_nd	47948	51	77	694624	49629	12	0
CORTEX M0_d	8238	51	77	116866	8668	93	1
AES_d	12491	54	80	150632	10596	58	0
CORTEX M0×3_d	26690	55	79	409579	27400	107	0
JPEG_d	48317	52	77	764738	55824	13	0

Libraries and design rules. We use a prototype $7nm$ standard-cell library from a leading IP provider. Since our design enablement for the $7nm$ technology is missing detailed BEOL technology information such as RC values and BEOL stack options, we scale the library to use a $28nm$ BEOL stack, following the methodology described in [78]. The site width and height are $0.136\mu m$ and $0.9\mu m$; these values correspond to placement site and cell row height parameters of the $28nm$ enablement. For design rules, we set the OW, IW1 and IW2 rules as four site widths. To check for DDA and OW violations, we pre-characterize all pairs of standard cells in the $7nm$ library. The library has 62 standard cells and the total number of pairs is 15376 (= 62×62×2×2), including cell flipping. For the standard cells without dummy poly gate, 7172 pairs out of the 15376 pairs violate the DDA constraint, and these pairs require at least one site space. Similarly, with the 4 site widths for OW, 280 out of the 15376 pairs violate the OW constraint, and such pairs also require one site space.[24]

[24]Based on our OW rule and library, all pairs of standard cells that violate OW constraints require only one site space. However, depending on the OW rule and library, some pairs of standard cells could require two or more site

Tradeoff between area/wirelength and DRVs. Table 2.9 shows the testcases used in our experiments. LVT, WSS, WHS and WL respectively indicate the portion of LVT cells, the worst setup and hold slacks, and wirelength. We assign V_{th} to cells uniformly to create more IW1 and IW2 violations. We use two kinds of libraries: (i) without dummy poly gates (CWOD) and (ii) with dummy poly gates (CWD). CWD is designed with dummy poly gates inserted to avoid interactions between cells which create DDA violations. For the CWD library, cell width is increased by one poly pitch compared to the CWOD library. The suffixes *_*d* and *_*nd* indicate that the designs are implemented with CWD and CWOD libraries, respectively. The same initial netlists are used for both *_*d* and *_*nd*. While comparing *_*d* and *_*nd* designs, we observe that the average wirelength and area overhead of designs implemented using libraries with dummy poly gates are 7% and 14%, respectively. In terms of DRVs, *_*nd* testcases have 134%∼176% more initial DRVs as reported in the second and fourth columns of Table 2.10.

Experimental Results

Table 2.10: Results with #violations, worst setup slack, worst hold slack, Δwirelength, maximum Δcell location, average Δcell location, #changed cells and runtime.

Design	IW #Vio.		DDA/OW #Vio.		WSS (ps)		WHS (ps)		ΔWL (%)	Max. Δloc. (μm)	Avg. Δloc. (μm)	#Changed cells (%)	CPU total (sec)	
	Init	Final	Init	Final	Init	Final	Init	Final					Global	Total
CORTEX M0_nd	926	11	1611	14	38	83	0	0	2.79	2.89	0.56	4489 (54%)	768	2820
AES_nd	1771	16	1900	18	90	71	0	-1	3.42	2.89	0.52	5939 (49%)	787	2992
CORTEX M0×3_nd	3514	17	4230	48	126	113	0	0	2.90	3.02	0.51	12752 (47%)	957	6897
JPEG_nd	4056	29	12024	135	12	22	0	0	2.30	8.99	0.70	24169 (50%)	1788	11983
CORTEX M0_d	988	10	0	0	93	85	1	0	3.04	2.89	0.57	2996 (36%)	161	434
AES_d	1566	11	0	0	58	80	0	0	3.10	2.89	0.54	3852 (31%)	425	1207
CORTEX M0×3_d	2810	27	0	0	107	105	0	-2	2.14	2.89	0.58	9340 (35%)	517	1336
JPEG_d	6296	43	0	0	13	81	0	0	-0.57	3.02	0.49	12244 (27%)	954	1401

Table 2.10 summarizes the number of DRVs, the worst setup slack, worst hold slack, Δwirelength, maximum Δlocation, average Δlocation, the number of moved cells and runtime. Our DFPlacer fixes more than 99% of initial violations in runtime that is reasonable for practical contexts. From a timing perspective, ΔWSS (i.e., final WSS − init WSS) ranges from -19ps to 68ps, but all final designs have no negative WSS. Similar to WSS, ΔWHS ranges from -2ps to 0ps. The timing impact is small since most of the cells are moved within a given small perturbation range. Some cells can be moved more than 20 sites (i.e., $0.136 \times 20 = 2.72\mu m$) from their initial locations due to the accumulated displacement in the iterative local optimization.

spaces.

However, those cells are less likely to be in the most critical path, which is how the WSS or WHS would worsen. Also, the positive ΔWSS implies that there is room to improve timing, and that we could potentially co-optimize the timing along with DRV fixing in detailed placement legalization. This is a direction of ongoing work.

On the other hand, DFPlacer increases wirelength by up to 3%. The accumulated displacements of cells and the limited pin access for the standard cells in N10 could be causes of this wirelength increase. Between *_nd and *_d testcases, the ΔWL% of *_nd cases is similar or slightly larger. We believe that this is because IW violations are harder to fix compared to the OW and DDA violations, since the constraints are more complex. The rate at which the number of IW violations reduces is slower than that for OW and DDA violations. Also, since the CWD library cells are larger than the CWOD library cells, the displacement of cells in *_d cases might have more impact on the wirelength increase. Therefore, % WL increase of *_d cases is smaller in general, but not necessarily always less than that of corresponding *_nd cases.

Columns *Max.* Δ*loc.* and *Avg.* Δ*loc.* show maximum and average cell displacement, respectively. We observe that the average cell displacement for all designs is up to $0.70\mu m$, which is $\sim$5 sites' width. The maximum displacement is up to $8.99\mu m$ for *JPEG_nd*. For other designs, the maximum displacement is similar to the half-perimeter of the perturbation range used in the global optimization ($2.888 = 0.9 \cdot 2 + 0.136 \cdot 8$ microns).

When we compare the results of designs with CWOD and CWD, we see a tradeoff between area and the number of DRVs (and runtime). We observe that the area overhead of using cells with dummy poly gate is 14% on average (up to 19%). However, the number of DRVs decreases by 61% on average (up to 64%). This affects the runtime of detailed placement legalization.

Figure 2.24 shows the remaining number of DRVs (%) versus runtime (sec). Each dot stands for an iteration of the optimization, and the third iteration points are marked with diamond-shaped markers. During the global optimization, which includes first, second and third iterations, the remaining violations drop quickly; $\sim$90% of DRVs are fixed in most of the designs during the global optimization. The runtime of the global optimization phase still increases with the problem size. However, with added computing resources to run windows of cells in parallel, the runtime can be further reduced. After the third iteration, when entering into local optimizations, the rate of decrease of the number of DRVs becomes much lower, implying that DFPlacer spends considerable time to fix the last few DRVs. This is because these last DRVs cannot be solved with small window sizes and perturbation ranges in the global optimization;

thus, DFPlacer tries to resolve them in the local optimization by increasing window sizes and perturbation ranges. The poor scaling of MILP solution versus instance size leads to the observed runtimes.

Figures 2.25(a) and 2.25(b) respectively show layout snapshots from the pre- and post-detailed placement legalization phases. In Figure 2.25(a), we highlight the cells that violate OW (green color), DDA (light green color), IW1 (yellow color) and IW2 (brown color) rules. Figure 2.25(b) shows the displacement of corresponding cells in post-detailed placement legalization. We observe that our DFPlacer fixes the DDA violation by flipping one of the violating cells; the IW1, IW2 and OW violations are all resolved by moving the violating cells or their neighbor cells within and/or across rows.

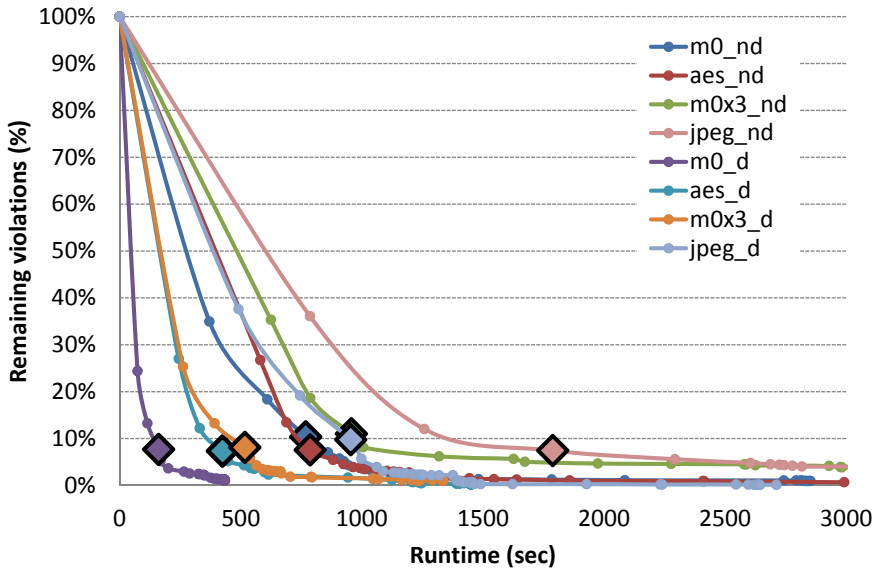


Figure 2.24: Remaining violations vs. runtime. Each dot indicates an iteration; after the third iteration, local optimization is performed. The diamond-shaped markers represent third-iteration points.

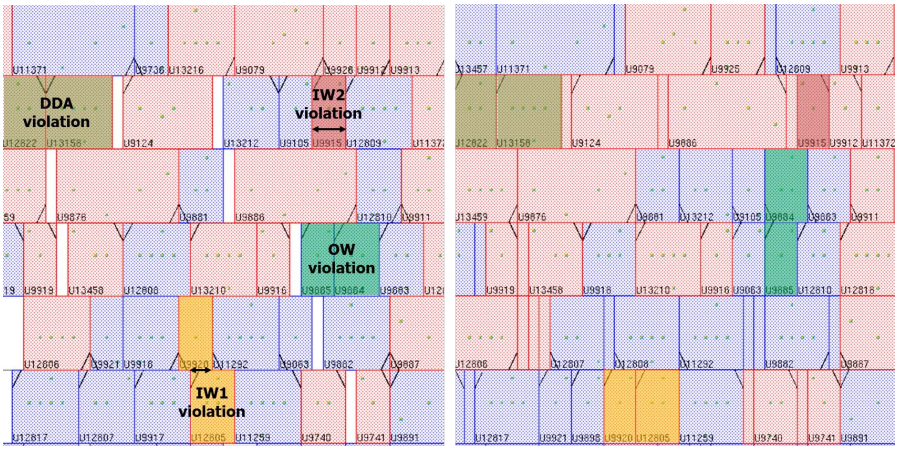


Figure 2.25: (a) Layout with DRVs before optimization. (b) Layout without DRVs after optimization.

2.2.5 Conclusion

In this section, we have proposed a scalable detailed placement legalization flow for complex FEOL constraints arising at the N10 foundry node. These include drain-drain abutment, minimum implant width, and minimum OD jogging rules. Given initial (timing-driven) placements, our $DFPlacer$ fixes 99% of DRVs with 3% increase in wirelength and minimal impact on timing. We feel that our use case of fixing all but a few tens of violations, with a highly parallelizable two-iteration strategy, is a good practical tradeoff between runtime complexity and DRV fixing. Further, the level of DRV fixing achieved by DFPlacer is encouraging, given that our default experimental configuration makes no attempt at "correctness by construction". Using OpenMP, we confirm that our flow is scalable via a distributed optimization strategy. Additionally, we study an area-DRV tradeoff between two types of standard-cell library strategies, namely, with and without dummy poly gates.

Our future work includes (i) timing and wirelength-driven placement legalization, which we believe can be enabled by more compact optimization formulations along with a more restricted perturbation range for each cell; (ii) a "smart ECO" method for the few DRVs that remain after global placement legalization; and (iii) further investigation of the scalability of our partitioning-based distributed optimization approach. Finally, we believe that our present placement-centered work may converge with such recent routing-centered works as [78], leading eventually to an "optimal detailed P&R" that can shield physical design teams from impacts of increasing ground rule complexity at N10 and beyond.

2.3 Evaluation of BEOL Design Rule Impacts Using an Optimal ILP-based Detailed Router

To scale semiconductor process nodes below the resolution limits of $193i$ optical lithography, multi-patterning techniques (e.g., litho-etch-litho-etch (LELE) and self-aligned double and quadruple patterning (SADP, SAQP) [127] have already been widely used in production. Multi-patterning is expected to be the basis of mainstream process offerings through the foundry $10nm$ and even $7nm$ nodes, and will persist even with deployment of extreme ultraviolet (EUV) lithography [208]. Although multi-patterning techniques are key enablers for advanced sub-$20nm$ process technologies, they can induce highly complex design rules which challenge both IC physical design tools and the development (and enablement) of IC physical implementation methodology. Tight design rules (e.g., via placement restrictions, unidirectional routing on Mx layers, etc.) lead to design wirelength and density overheads, to the point where benefits from technology scaling reduce or even disappear altogether. Assessing the real value of a prospective future technology is also difficult in FinFET nodes, where higher drive strengths enable smaller standard-cell footprints that further challenge pin access and routability [3].

Given the above considerations, as well as the enormous cost of technology development and design enablement for a new process node, it is critical for the industry to be able to assess the impact of design rules (implicitly, patterning technology choices) on physical implementation metrics. Such assessments should be made as early as possible, to permit correct choices among various technology options and to enable design-technology co-optimization. Unfortunately, there are two basic reasons why process technology developers cannot easily evaluate impacts of complex design rules on chip implementation metrics. First, EDA vendors often require prolonged, close co-development with customers to correctly support new advanced design rules. While the latest Library Exchange Format standard (LEF5.8) [209] supports advanced design rule descriptions, even for the rapidly approaching foundry $10nm$ node there is varying (and contradictory) support across the EDA industry today [163, 149]. Thus, it is practically difficult to study new "future" design rules with current EDA tools. Second, EDA tools apply many heuristics to perform efficient large-scale layout optimizations. This clouds evaluations of how new patterning technologies or design rules impact chip implementation metrics. In other words, the "chicken-egg" relationship between current EDA algorithms that are optimized for current design enablements (design rules, cell libraries, etc.) makes it difficult to assess true impacts of future design enablements. Wherever possible, we would like to reduce the "chicken-egg" obstacles to design rule and patterning technology assessment.

In this section, we provide a framework for evaluating how prospective sub-20nm design rules – as well as back-end-of-line (BEOL) stack choices – will affect chip implementation metrics such as density or wirelength. Our framework is based on *optimal* detailed routing that is correct with respect to advanced design rules. We describe *OptRouter*, an ILP-based optimal detailed router which considers various design rules and technology options especially for the coming 10nm/7nm process nodes. OptRouter computes optimal routing solutions for small switchboxes (approximately the size of a single gcell [104], similar to the recent work of [96]), and has the ability to consider routing direction (unidirectional or bidirectional), design rules induced by advanced patterning technology (e.g., SADP), via adjacency restrictions, and pin shapes. Our studies combine realistic testcases in multiple technologies (including testcases synthesized with a prototype 7nm cell library from a leading commercial IP provider) with cost-optimal detailed routing. It is this combination that enables new, quantitative assessment of design rule impact on detailed routing metrics. The key contributions of our work are summarized as follows.

- We formulate as an integer linear program (ILP) a minimum-cost switchbox routing problem that arises in advanced technology nodes (corresponding to clips from standard-cell place-and-route instances). In contrast to previous approaches, our formulation captures multi-pin net routing (i.e., Steiner routing), via shapes, via adjacency restrictions, pin shapes, layer uni-/bi-directionality, and SADP constraints that occur with sub-20nm patterning.

- We develop *OptRouter*, which extracts layout clips from place-and-route solutions and uses ILOG CPLEX v12.5.1 [203] to solve the corresponding ILP instances. The correctness and capability of OptRouter are validated against commercial router results with foundry 28nm 8- and 12-track and 7nm 9-track libraries.

- We apply OptRouter within a novel methodology to quantify and rank impacts of complex sub-20nm design rules on layout metrics (wirelength, vias, and routability). Our testbed notably includes a prototype 7nm PDK from a leading IP provider, as well as the aforementioned 28nm foundry libraries.

- Our comparisons of different design rules' impacts can potentially guide patterning technology choices and other basic design-technology co-optimization decisions.

In the remainder of this section, Section 2.3.1 briefly reviews relevant previous works, with an emphasis on the seminal recent work of Jia et al. [96]. In Section 2.3.2, we explain our

ILP formulation of multi-pin net routing with the consideration of various cell library and routing strategies (e.g., pin shapes and via shapes) and routing rules (e.g., via adjacency restrictions and SADP-specific rules) Section 2.3.3 describes our empirical studies: experimental questions, design of experiments, result and discussion. We offer conclusions and ongoing research directions in Section 2.3.4.

2.3.1 Related Work

Relevant previous works are found in two areas: (1) design rule evaluation frameworks, and (2) ILP-based global and detailed routers.

Design rule evaluation. The work of [73] exemplifies efforts to connect layout ground rules with layout area, electrical variability, and parametric yield implications. Specifically, the authors of [73] study the effect of a line-end extension rule on logic standard cell and SRAM bitcell layout area, and on leakage variability and parametric yield. Ghaida and Gupta [65] propose DRE, a platform that comprehensively connects design rule alternatives to the automated synthesis of standard-cell library cells, and then to the power-performance-area envelope of standard-cell based layouts of small blocks. Subsequently, [64] extends the DRE approach to chip-level analyses. Badr et al. [16] suggest a pattern matching-based design rule evaluation method, which is then applied to checking of routing within standard cells. A fundamental distinction between these previous works and our present work is that we provide a new capability to assess design rule and patterning technology choices *with cost-optimal detailed routing*.

ILP-based routers. ILP has been widely used for optimization problems due to its simplicity along with its ability to find optimal solutions up to some limit of tractable instance complexity. A number of works adopt ILP for global routing, often starting from a multi-commodity flow perspective. The early work of Carden and Cheng [25] uses column-generating techniques within a multi-commodity flow based global router. Cho et al. [35] propose a global router based on box expansion and progressive ILP. After decomposing nets into two-pin nets, ILP is used to choose a routing between two L-shaped candidate routings for each two-pin net within a box. The approach iteratively expands the box and solves new nets within the expanded new box, using progressive ILP and maze routing. Similarly, Hu et al. [92] use ILP for global routing; they enumerate two path candidates to connect two-pin nets after initial routing, and an ILP is formulated to select the better path between the two candidates.

An important recent work is that of Jia et al. [96], which proposes a *detailed* router based on multi-commodity flow. The authors of [96] formulate an ILP for detailed routing with all nets being two-pin nets. Pin shapes and basic design rules (side-to-side, tip-to-tip, cut-to-

cut) are considered. The proposed methods are demonstrated to reduce the number of Design Rule Check (DRC) violations in a $45nm$ technology without wirelength or via overheads. A fundamental distinction between the work of Jia et al. [96] and our present work is that [96], while using ILP, does not guarantee optimal routing since multi-pin nets are not handled in the formulation. Further, only basic design rules are considered. In particular, the ability to compute minimum-cost *optimal* routing solutions with SADP-specific rules and via shapes is unique to our present work.

2.3.2 Optimal Routing Formulation

We now describe our ILP-based formulation of the detailed routing problem for a netlist of multi-pin nets, with consideration of via adjacency restrictions, unidirectional routing, SADP-aware line end rules, pin shapes, and via shapes. Like previous works, our development adopts the well-known paradigm of multi-commodity flow. Table 2.11 gives the notations that we use.

Table 2.11: Notations.

Notation	Meaning
N	set of multi-pin nets
n_k	k^{th} multi-pin net
s_k	source of n_k
T_k	set of sinks of n_k
$t_{k,i}$	i^{th} sink of n_k
$G(V, A)$	routing graph
V	set of vertices (of the routing graph)
v_i	a vertex with the location (x_i, y_i, z_i)
A	set of directed arcs
$a_{i,j}$	a directed arc from v_i to v_j
$e^k_{i,j}$	0-1 indicator whether $a_{i,j}$ is used in the routing of n_k
$c^k_{i,j}$	cost for $a_{i,j}$ in the routing of n_k
$f^k_{i,j}$	flow variable for $a_{i,j}$ in the routing of n_k
$p^k_{r,i}(p^k_{l,i})$	0-1 indicator whether there are the flows connected to v_i coming from right (left) side, in the routing of n_k

General Routing Problem Formulation

We use a *routing graph* $G = (V, A)$ to represent available routing resources, e.g., metal tracks on multiple layers, and inter-layer vias. Each vertex $v_i \in V$ is associated with variables

that represent coordinates in the three-dimensional routing resources: horizontal metal track x_i, vertical metal track y_i and metal layer z_i. A directed arc $a_{i,j}$, where $x_j = x_i$, $y_j = y_i$ and $z_j = z_i \pm 1$, represents a via. We solve the optimization:

$$\text{Minimize: } \sum_{n_k \in N} \sum_{a_{i,j} \in A} c^k_{i,j} \cdot e^k_{i,j}$$

Subject to:

$$\sum_{n_k \in N} (e^k_{i,j} + e^k_{j,i}) \leq 1 \quad a_{i,j}, a_{j,i} \in A \tag{2.43}$$

$$e^k_{i,j} \geq \frac{f^k_{i,j}}{|T_k|} \quad a_{i,j} \in A, n_k \in N \tag{2.44}$$

$$e^k_{i,j} \leq f^k_{i,j} \quad a_{i,j} \in A, n_k \in N \tag{2.45}$$

$$\sum_{v_j : a_{i,j} \in A} f^k_{i,j} - \sum_{v_j : a_{j,i} \in A} f^k_{j,i} = \begin{cases} |T_k| & \text{if } v_i = s_k, n_k \in N \\ -1 & \text{else if } v_i \in T_k, n_k \in N \\ 0 & \text{otherwise} \end{cases} \tag{2.46}$$

The objective is to minimize the weighted sum of $e^k_{i,j}$, i.e., weighted total wirelength and the number of vias. Constraint (2.43) ensures that each arc is used by only one net. Constraints (2.44) and (2.45) pertain to the binary variable $e^k_{i,j}$, which indicates whether there is a flow through $e_{i,j}$. Constraint (2.46) ensures source-sink connectivities (flow conservation). The first and second terms respectively represent the sum of the flows exiting v_i (outflows of v_i) and the sum of the flows entering v_i (inflows of v_i). For any internal node that is not a source or a sink, the sum of the node's outflows must equal to the sum of the node's inflows. For a source s_k, the sum of outflows of s_k must be $|T_k|$ (the number of sinks) since there must be $|T_k|$ flows which connect between s_k and $|T_k|$ number of sinks in n_k, and the sum of inflows of s_k must be zero. On the other hand, for a sink $v_i \in T_k$, the sum of inflows must be one since a flow coming from s_k must reach each sink, and the sum of outflows must be zero.

Figure 2.26 shows a two-net example consisting of a three-pin net (n_1) and a two-pin net (n_2), along with its solution. Net n_1 has a source v_1 and two sinks, v_3 and v_4. Net n_2 has a source v_5 and a sink, v_6. According to Constraint (2.46), for n_1, the sum of outflows of the source node $v_1 = 2$ ($|T_1|$) and the sum of inflows of sink nodes v_3 and $v_4 = -1$. Similarly, for n_2, the sum of outflows of $v_5 = 1$ and the sum of inflows of $v_6 = 1$. For all other vertices, the sum of outflows is equal to the sum of inflows so that flows are conserved for each net. According to Constraint (2.43), $e^1_{1,2}$, $e^1_{2,3} = 1$, which connects between v_1 and v_3, since

$f^1_{1,2}$, $f^1_{2,3}$ are non-zero values. Constraint (2.43) forces $e^2_{1,2}$, $e^2_{2,1}$, $e^1_{2,1}$ to be zero so that the edge between v_1 and v_2 can be reserved only for n_1.

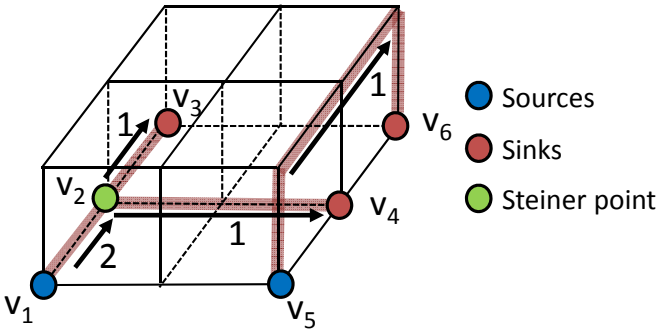


Figure 2.26: Example showing multi-pin nets and the routing solution.

Routing Rule Formulation

Via restrictions. As noted in [127], placement of vias next to each other is not allowed in advanced nodes. That is, as via pitches are larger (e.g., by a $\sqrt{2}$ factor) than metal pitches, placement of a via at a particular location blocks horizontally and vertically adjacent locations, and sometimes diagonally adjacent locations as well.

We use the following constraint so that any neighbor vertical arcs $a_{i',j'}$ of a vertical arc $a_{i,j}$ can be blocked if there is a via between v_i and v_j, where $x_{i'} = x_{j'} = x_i \pm 1$, $y_{i'} = y_{j'} = y_i \pm 1$, $z_{i'} = z_i$ and $z_{j'} = z_j$.

$$e^k_{i,j} + e^k_{j,i} + e^k_{i',j'} + e^k_{j',i'} \leq 1 \qquad \forall\ a_{i',j'}$$

In our study below, we consider two types of restrictions: (i) blocking of orthogonally adjacent locations (N, E, S, W neighbors), and (ii) blocking of both orthogonally and diagonally (NE, NW, SE, SW) adjacent locations.

Unidirectional routing. Patterning with severe restriction, as with one-pitch/one-orientation metal layers, is used because of better robustness, scalability and manufacturability – as well as fewer masking steps – compared to a standard LELE-patterned bidirectional metal layer. We trivially restrict routing on a given layer to be unidirectional by removing arcs that are not in the preferred direction. (See also our discussion of SADP constraints, below.)

Pin shape. In the above example of Figure 2.26, we assume that each source or sink has a single fixed location. However, in actual routing, a pin has multiple access points, which

means that the source or sink locations ultimately used in the routing solution can vary. Multiple access points for source and sink are captured by creating a *supersource* or *supersink* which is connected to all available access points in the corresponding pin. We note that the supersource and supersink are virtual vertices, which are not actually located but which nonetheless have flows. We also observe that each access point for source or sink becomes an internal node.[25]

Via shape. To trade off between manufacturability and routability, various via types with square or rectangular shapes may be instantiated. Some vias shapes, e.g., 2×2 size, are too large to be modeled as a single vertex in our routing graph. We model a via's shape by creating a *representative vertex* which is connected to all the vertices that belong to a via, according to that via's footprints on lower and upper layers.[26]

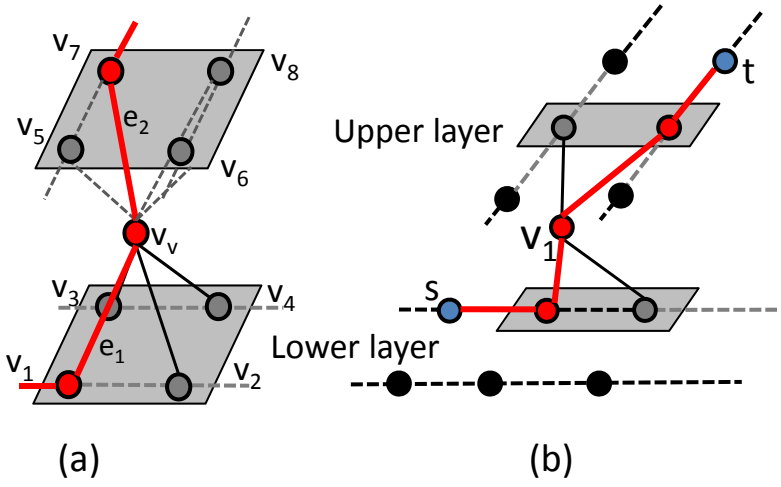


Figure 2.27: Via shape. (a) 2×2 square via. (b) 2×1 bar via.

Figure 2.27(a) shows a via and its vertices in a routing graph. v_v is a square via with size 2×2 (with respect to the number of metal tracks). With the flow conservation constraint (Constraint (2.45)), once a flow (routing) enters v_v, the flow goes through one of four vertices in the upper layer. Note that for each via type, vertices are created for all possible locations where the via can be placed. For example, if a 2×2 size square via type is added to the routing graph with three layers and 15×15 tracks ($15 \times 15 \times 3$), we will create 392 ($14 \times 14 \times 2 = (15 -$

[25]Pin shape is important in assessment of routing costs, e.g., smaller pin geometries with fewer access points in advanced FinFET nodes are a major challenge to detailed routing. In 7.5T or 7.25T library cells in FinFET nodes, power/ground rails, fin connections and other aspects of standard cell architectures must reconcile with pin shapes (access points). Strict tip-to-tip spacing (more than one contacted poly pitch (pin pitch)), diagonal via placement restriction as discussed above, and wider power rails also decrease the number of access points to a cell and potentially cause unroutability [128]. We study interactions of smaller pin shapes in $7nm$ (Figure 2.34(c)) and routing rules in Section 2.3.3.

[26]Doubled or redundant vias are also modelable with small modification of via shape formulation.

$1) \times (15-1) \times (3-1))$ vertices for the square via at all possible locations. Further, note that we use lower cost values for larger via shapes so that the optimization selects as many larger vias as possible to achieve better manufacturability.

In addition to the basic formulation with Constraints (2.43)–(2.45), vertices used by a via must be blocked and not be used by other nets. For example, in Figure 2.27(a), as v_7 is selected by e_2, all the gray edges connected to other vertices used by the via (v_5, v_6, v_8) must be disabled for other nets. A generalized formulation is given in Constraint (2.47) where i' are the vertices that are not used for the routing but are within the used via shape, and j' are the neighbor vertices of i':

$$\sum_{n_k \in N} (e^k_{v,i} + e^k_{i,v}) + \sum_{n_{k'} \in N, v_{i'}, v_{j'}: a_{i',v}, a_{j',i'} \in A} (e^{k'}_{i',j'} + e^{k'}_{j',i'}) \leq 1$$
$$\text{where } a_{i,v} \in A, k' \neq k, i' \neq i \tag{2.47}$$

Thus, Constraint (2.47) prevents any other nets from using the vertices i' or the edges connected to i'. Figure 2.27(b) shows an example of 2×1 size *bar via* shape. Vertices s and t are source and sink, respectively. The red lines are selected as routing from s to t. The gray dots in Figure 2.27(b) are disabled for other nets by Constraint (2.47), so that there is no overlap between the bar via and other nets.

SADP-aware rules. Xu et al. [192] propose SADP-specific design rules. Figure 2.28(a) illustrates how the end of line (EOL) of a wire segment is the key parameter to check with such rules.

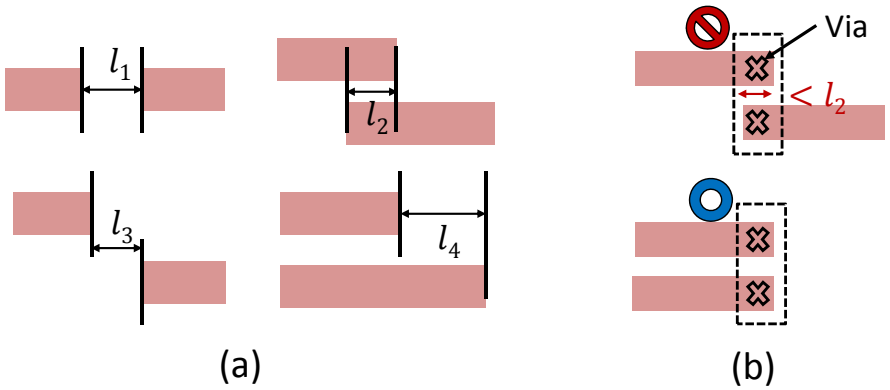


Figure 2.28: (a) SADP-specific design rules and (b) example showing that via location does not provide enough information to distinguish the upper and lower cases, i.e., to check SADP-aware rules.

In our ILP formulation, we use via locations to check the locations of EOL, but this is not enough to differentiate the two cases in Figure 2.28(b), where the upper case is an illegal routing while the lower case is legal with the same via placements. Therefore, binary variables $p^k_{r,i}$, $p^k_{l,i}$ that indicate whether there are flows connected to a vertex v_i that come from right or left direction, respectively, are defined for a net n_k to represent the directions of the EOL. Note that there are only two directions, since we assume unidirectional routing.

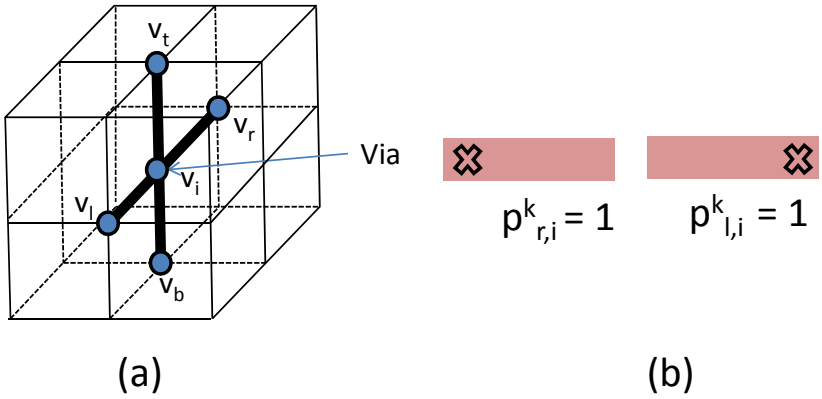


Figure 2.29: An example of a routing graph. (a) The p variable of a vertex v_i is determined by flow variables of edges with vertex v_i's neighbor vertices v_t, v_b, v_l, v_r. (b) Wire segment geometries that respectively result when $p^k_{r,i} = 1$ and $p^k_{l,i} = 1$.

Figure 2.29(a) shows a vertex v_i and its top, bottom, left, right neighbor vertices (v_t, v_b, v_l, v_r) in a routing graph, and Figure 2.29(b) enumerates the cases when each p variable $= 1$. For the left EOL at (x_i, y_i, z_i) in Figure 2.29(a), where $p^k_{r,i} = 1$, the right edge ($e^k_{r,i}$) connected to v_i must be used in routing and the left edge ($e^k_{l,i}$) connected to v_i must not be used. By Constraint (2.46), the expression $e^k_{r,i}$ && $\neg e^k_{l,i}$ is equal to the right-hand side of Constraint (2.49). The right EOL ($p^k_{l,i} = 1$) is formulated in the same manner, i.e., as Constraint (2.48).

$$p^k_{l,i} = (e^k_{l,i} * e^k_{i,t}) || (e^k_{l,i} * e^k_{i,b}) || (e^k_{i,l} * e^k_{t,i}) || (e^k_{i,l} * e^k_{b,i}) \tag{2.48}$$

$$p^k_{r,i} = (e^k_{r,i} * e^k_{i,t}) || (e^k_{r,i} * e^k_{i,b}) || (e^k_{i,r} * e^k_{t,i}) || (e^k_{i,r} * e^k_{b,i}) \tag{2.49}$$

As all the variables are binary, we can convert the quadratic constraints in Constraint (2.48) and (2.49) to linear constraints by using a simple technique as shown in (2.50). The $(a \leq b)$ && $(a \leq c)$ condition ensures that a is zero when either b or c is zero. The condition $(a \geq b + c - 1)$ makes $a = 1$ when both b and c are one.

$$a = b * c \qquad \Longleftrightarrow \ (a \leq b)\ \&\&\ (a \leq c)\ \&\&\ (a \geq b + c - 1) \tag{2.50}$$

We then convert the Constraint (2.48) to a set of linear constraints as shown in Constraint (2.51).

$$\begin{aligned}
p^k_{l,i} \geq p^k_{l,i,1}\ ;\ p^k_{l,i} \geq p^k_{l,i,2}\ ;\ p^k_{l,i} \geq p^k_{l,i,3}\ ;\ p^k_{l,i} \geq p^k_{l,i,4} \\
p^k_{l,i} \leq p^k_{l,i,1} + p^k_{l,i,2} + p^k_{l,i,3} + p^k_{l,i,4} \\
(p^k_{l,i,1} \leq e^k_{l,i})\ \&\&\ (p^k_{l,i,1} \leq e^k_{i,t})\ \&\&\ (p^k_{l,i,1} \geq e^k_{l,i} + e^k_{i,t} - 1) \\
(p^k_{l,i,2} \leq e^k_{l,i})\ \&\&\ (p^k_{l,i,2} \leq e^k_{i,b})\ \&\&\ (p^k_{l,i,2} \geq e^k_{l,i} + e^k_{i,b} - 1) \\
(p^k_{l,i,3} \leq e^k_{i,l})\ \&\&\ (p^k_{l,i,3} \leq e^k_{t,i})\ \&\&\ (p^k_{l,i,3} \geq e^k_{i,l} + e^k_{t,i} - 1) \\
(p^k_{l,i,4} \leq e^k_{i,l})\ \&\&\ (p^k_{l,i,4} \leq e^k_{b,i})\ \&\&\ (p^k_{l,i,4} \geq e^k_{i,l} + e^k_{b,i} - 1)
\end{aligned} \tag{2.51}$$

Here, $p^k_{*,*}$ is a net-specific variable. As SADP rules must be checked over all nets, we define global p variables as follows:

$$p_{l,i} = \sum_{n_k \in K} p^k_{l,i},\ \ p_{r,i} = \sum_{n_k \in K} p^k_{r,i} \tag{2.52}$$

Figure 2.30 shows how the p variables can be used to formulate SADP-aware rules for ILP. Figure 2.30(a) shows a wire segment, of which the EOL is located at vertex v_i with the wire coming from the right side; (b) and (c) show forbidden via locations for the other wire segments. The constraints shown in Figures 2.30(b) and (c) are formulated as Constraints (2.53) and (2.54), respectively.

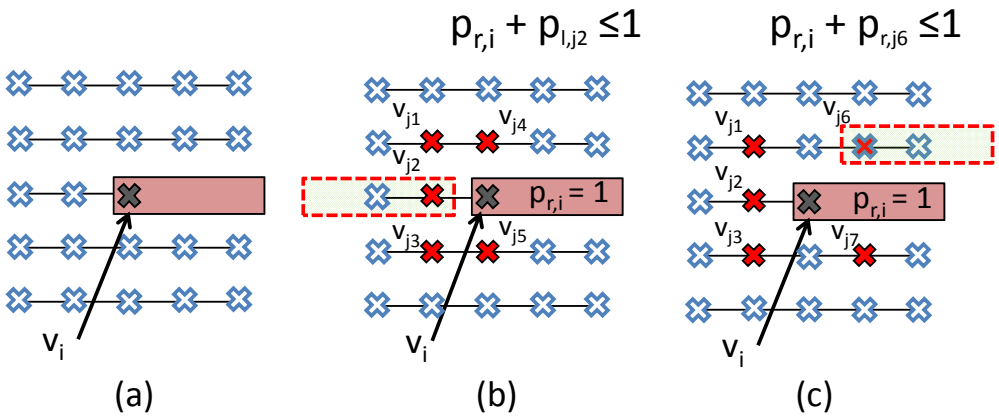


Figure 2.30: (a) A wire segment, of which the EOL is located at vertex v_i with the wire coming from the right side. (b) Forbidden via locations for other wire segments with $p_{l,j} = 1$. (c) Forbidden via locations for other wire segments with $p_{r,j} = 1$.

$$(p_{r,i} + p_{l,j_1} \leq 1) \;\&\&\; (p_{r,i} + p_{l,j_2} \leq 1) \;\&\&\; (p_{r,i} + p_{l,j_3} \leq 1)$$
$$\&\&\; (p_{r,i} + p_{l,j_4} \leq 1) \;\&\&\; (p_{r,i} + p_{l,j_5} \leq 1) \tag{2.53}$$
$$(p_{r,i} + p_{r,j_1} \leq 1) \;\&\&\; (p_{r,i} + p_{r,j_2} \leq 1) \;\&\&\; (p_{r,i} + p_{r,j_3} \leq 1)$$
$$\&\&\; (p_{r,i} + p_{r,j_6} \leq 1) \;\&\&\; (p_{r,i} + p_{r,j_7} \leq 1) \tag{2.54}$$

2.3.3 Empirical Studies

Our empirical studies seek to answer two basic questions:

- What are the design costs of various BEOL rules with respect to wirelength, # vias, and routability metrics?
- How much do impacts of design rules vary across different technologies and different-track cell architectures?

Overall flow of BEOL rule evaluation. We implement our experimental testbed in C++ code, with interface to support LEF/DEF [209] implemented via the *OpenAccess 2.6* [217] API. We use *CPLEX 12.5.1* [203] as our ILP solver. Figure 2.31 shows our overall BEOL rule evaluation flow. From a routed design, all possible routing clips are extracted, and evaluated according to our pin cost metric. The clips with highest pin cost are selected, and each clip (= switchbox instance) is converted to a routing graph based on available metal tracks, and then to

a corresponding ILP instance, for each routing rule configuration that we study. Our *OptRouter* then obtains optimal routing solutions by solving the ILP. From the solution, we report out wirelength, number of vias, and feasibility (routability) for each given clip with each given rule configuration. For the experiments that we report here, routing cost in the ILP is defined as *wirelength* + $4\times$ *number of vias*. We have separately observed that the ILP sensibly handles alternative routing cost definitions with different weighting of via count.

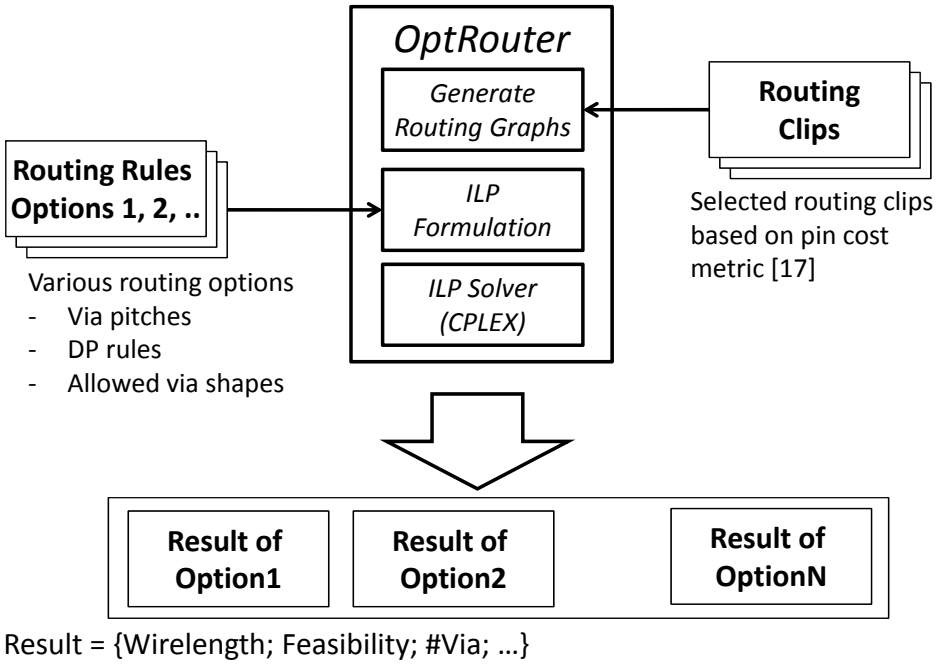


Figure 2.31: Overall flow of BEOL rule evaluation.

Physical implementation with advanced technology. We verify our methods using the open-source *AES* design [212] and an *Arm CORTEX M0*, implemented with three different technologies and standard-cell libraries: 8-track in $28nm$ FDSOI (N28-8T), 12-track in $28nm$ FDSOI (N28-12T), and 9-track in $7nm$ (N7-9T). We use *Synopsys Design Compiler vH-2013.03-SP3* [218] for synthesis and *Cadence Encounter Digital Implementation System v13.1* [198] for P&R. We implement each design multiple times, with a range of final utilizations. Table 2.12 summarizes benchmark design information.

For $7nm$ technology, we use $7nm$ standard-cell libraries (P&R, layout and timing views) from a leading IP provider; metal pitches on layers M1 to M6 and layers M7 to M8 are $40nm$ and $80nm$, respectively. In this technology, our design enablement is missing detailed BEOL technology information such as RC values and BEOL stack options. Thus, to obtain timing-closed P&R results we scale up the geometries of the $7nm$ 9-track cells by $2.5\times$ in the vertical

Table 2.12: Summary of testcases.

Tech.	Design	Period (ns)	#Inst.	Util. (%)
N28-12T	*AES*	1.2	13.5–14K	89–94
	CORTEX M0	2.2	9.2K	90–96
N28-8T	*AES*	2	12–12.7K	89–95
	CORTEX M0	2.5	9.3–9.5K	90–95
N7-9T	*AES*	0.6	13–15K	93–97
	CORTEX M0	1.2	9.7–11.4K	92–95

dimension (i.e., by the ratio of 1× metal pitch in 28nm horizontal layers (100nm) to 1× metal pitch in 7nm horizontal layers (40nm)). Then, the scaled 7nm cells fit into the 28nm BEOL stack with the same number of horizontal metal tracks, for which we use 100nm metal pitch in horizontal layers. To scale the widths of the 7nm standard cells, we scale by the ratio of the 28nm placement grid (vertical metal layer pitch of 136nm) to that of the 7nm placement grid (vertical metal layer pitch of 54nm), which is ∼2.5. We further adjust pin locations so that pins are on-grid, since simple scaling results in off-grid pins which affect routability.[27] To derive the missing 7nm wire RC information from 28nm RC values, we scale up R by 15× for 7nm wire R, and use the same wire C value. This follows methodology of, e.g., [28] to account for the rapid increase of resistivity in advanced nodes. Then, since we are using the scaled geometries to mimic a 7nm P&R flow, R and C per unit length are scaled down (in the P&R tool) by 2.5×. The end result is that 7nm R and C per unit length (R_{N7}, C_{N7}) are obtained from 28nm R and C per unit length (R_{N28}, C_{N28}) as $R_{N7} = 6 \times R_{N28}$ and $C_{N7} = C_{N28}/2.5$.

Extraction of routing clips. We use 1μm × 1μm routing clips extracted from the routed designs as input instances; these correspond to 7 vertical routing tracks × 10 horizontal routing tracks, with eight metal layers, for ***OptRouter***. Figure 2.32 shows example routing clips extracted from layouts with (a) N28-12T cells, (b) N28-8T cells, and (c) N7-9T cells.[28]

We select "difficult-to-route" clips based on pin cost metrics of Taghavi et al. [178], specifically, a pin existence cost (PEC), a pin-area cost ($PAC = \sum_{i=1}^{PEC} 2^{2-\frac{area(p_i)}{\theta}}$) and a

[27] In greater detail: the 28nm and 7nm placement grids are 136nm and 54nm, respectively, with ratio between the two being ∼2.519. It is not possible to obtain integer cell widths by simply scaling with this number. Thus, we scale up the 7nm cells by 2.5 so that all cell widths are a multiple of 135nm. We then increase each cell width by *scaled cell width* /135 in order to make it a multiple of 136nm, which is the foundry 28nm placement grid. Since scaling by 2.5× results in a pin pitch of 135nm, which is off-grid with respect to a 136nm grid, we perform a scripted movement of pin locations so that all pins are again on-grid (the pin x locations should be multiples of 136nm).

[28] By comparison, the recent work of [96] uses 1.26μm × 1.26μm clips in a 45nm technology; these correspond to 9 vertical routing tracks × 9 horizontal routing tracks.

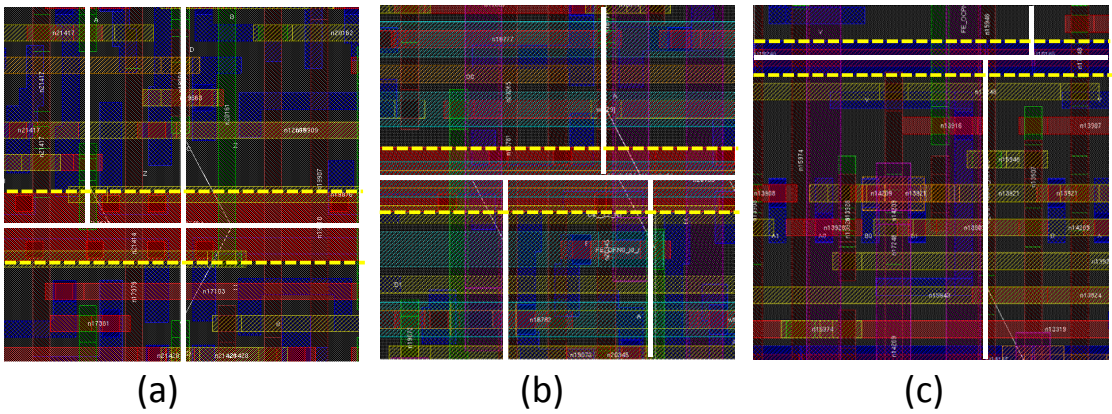


Figure 2.32: Routing clips from (a) N28-12T, (b) N28-9T and (c) N7-9T. Standard cell boundaries and power/ground rail are highlighted with white lines and yellow dashed lines, respectively.

pin-spacing cost ($PRC = \sum_{i=1}^{PEC-1} \sum_{j=i+1}^{PEC} 2^{2-\frac{spacing(p_i,p_j)}{3\theta}}$). We use $PEC+PAC+PRC$ as the *pin cost* for a routing clip, with $\theta = 500$ to obtain a reasonable range of costs.

We calculate the pin cost for every routing clip in the routed testcases listed in Table 2.12 ($\sim$10K clips per testcase). Figure 2.33 shows the top-100 pin cost ranges for several versions of *AES* and *CORTEX M0* design implementations in N7-9T with different utilizations. The utilizations of *AES_v1*, *AES_v2* and *AES_v3* are 93%, 95% and 97% respectively, and the utilizations of *CORTEX M0_v1*, *CORTEX M0_v2* and *CORTEX M0_v3* are 92%, 94% and 95%[29]. We observe that pin cost distributions do not change significantly with different utilizations, and that pin cost distributions are not design-specific: ranges of top-100 pin costs of both designs are similar (*AES*: 33$\sim$42, *CORTEX M0*: 30$\sim$41). Thus, in each technology we select top-100 clips from across all design implementations, according to the pin cost metric.

Design of Experiments

We evaluate various BEOL design rule configurations, each of which is a combination of via restrictions and mix of LELE/SADP BEOL layers. (All routing layers are unidirectional in our study.) Table 2.13 shows the BEOL design rule configurations, denoted as RULE1 - RULE11, used in the experiments. We test three via restriction cases (*0 neighbors blocked*; *4 neighbors blocked*; and *8 neighbors blocked*) and five LELE/SADP layer combinations (M2-M8 LELE layers (*No SADP*); M2-M8 SADP layers (*SADP* $\geq$ *M2*); M2 LELE + M3-M8 SADP

[29]We use high utilizations to obtain designs that are "difficult-to-route" and sensitive to design rules due to routing congestion.

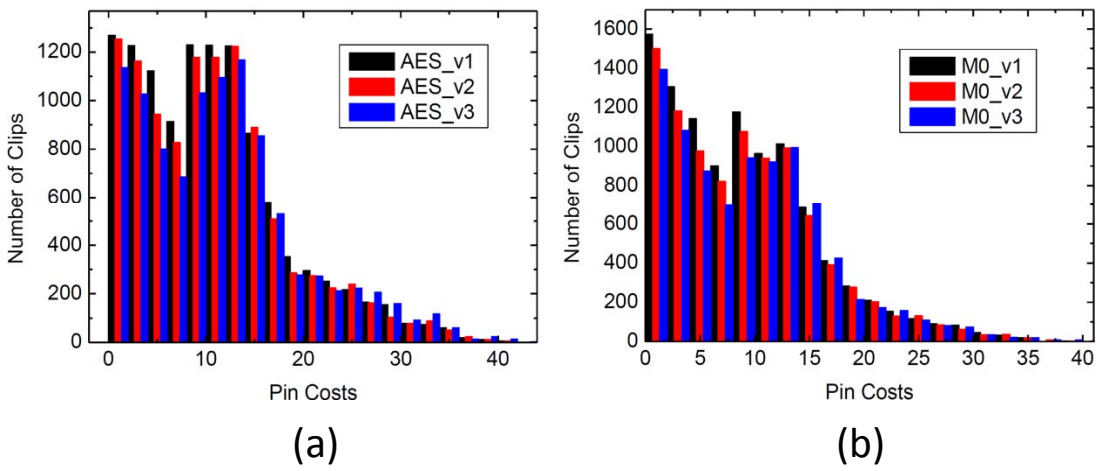


Figure 2.33: Pin cost distributions (per the *PEC* + *PAC* + *PRC* metrics in [178]) of (a) *AES* and (b) *CORTEX M0* with different utilizations.

layers (*SADP* $\geq$ *M3*); M2-M3 LELE + M4-M8 SADP layers (*SADP* $\geq$ *M4*); and M2-M4 LELE + M5-M8 SADP (*SADP* $\geq$ *M5*)). Via restrictions are applied to the V12 through V78 layers.

We select the top 100 routing clips according to pin costs across all designs in Table 2.12, for each of the three combinations of technology node and cell height, as discussed above. We then run *OptRouter* on each of the 100 routing clips to evaluate the impact of each given routing rule configuration. We obtain the Δcost of each rule configuration, relative to the routing cost of RULE1 (no constraints).[30] In the present study, we do not use M1 as a routing resource.

Table 2.13: BEOL design rule configurations.

Name	SADP rules	Blocked via sites
RULE1	No SADP	0 neighbors blocked
RULE2, 3, 4, 5	SADP $\geq$ {M2, M3, M4, M5}	
RULE6	No SADP	4 neighbors blocked
RULE7, 8	SADP $\geq$ {M2, M3}	
RULE9	No SADP	8 neighbors blocked
RULE10, 11	SADP $\geq$ {M2, M3}	

We have evaluated all of RULE1 to RULE11 for the N28-12T and N28-8T technologies. However, we do not test RULE2, RULE7, and RULE9 to RULE11 for N7-9T since the smaller

[30] Our separate studies support the claimed optimality of OptRouter. We have compared the results of OptRouter and those of the commercial routing tool, and have found that OptRouter always achieves non-positive Δcost with respect to the commercial tool's solution. Indeed, the average Δcost of -10$\sim$-15, relative to an average routing cost of $\sim$380, suggests the potential for using OptRouter for detailed routing improvement.

pin shapes in the $7nm$ standard cells do not permit the diagonal (adjacency in) via placement which is required for these rules. Figures 2.34(a), (b) and (c) show pin shapes in a NAND2X1 cell in N28-12T, N28-8T and N7-9T, respectively. In Figure 2.34(c), the input pin shapes have only two access points and the two pins are close to each other. With eight via sites blocked, it is impossible to connect to the two input pins without violations.

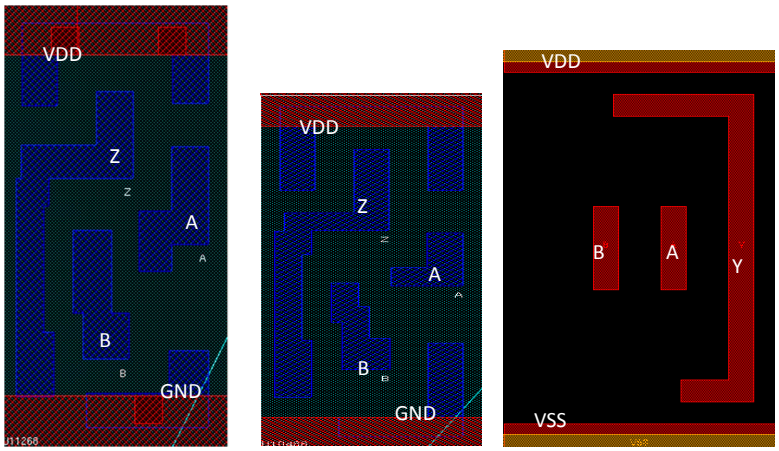


Figure 2.34: Pin shapes in NAND2X1: (a) N28-12T, (b) N28-8T and (c) scaled N7-9T.

Experimental Results and Discussion

Figures 2.35(a), (b) and (c) respectively show sorted Δcost per clip of each RULE, in N28-12T, N28-8T and N7-9T cell-based designs. The Δ is relative to costs with RULE1 (i.e., the minimum achievable routing cost with eight unidirectional LELE layers and no via restrictions). For unroutable clips, we arbitrarily set Δcost = 500 for convenience of plot generation.

In N28-12T (Figure 2.35(a)), we observe that SADP rules for upper metal layers above M3 do not significantly affect routing costs. Two kinds of via restrictions (4 or 8 neighbors blocked) show similar routing costs, suggesting that the orthogonal via restriction (4 neighbors blocked) is dominant. When SADP rules are applied (RULE4, RULE7, RULE2), routing costs vary across routing clips. By comparing the "crossing" traces for RULE2 and RULE6, we see that routing costs are higher with SADP layers than with via restriction rules, but that in terms of absolute feasibility, the via restriction appears to result in fewer feasible routings.

With N28-8T (Figure 2.35(b)), in contrast to the N28-12T case, there is higher sensitivity of Δcost to the number of SADP layers: we see a clear increasing cost trend across RULE2 through RULE5. With respect to via restriction, for the 8 LELE layer (no SADP) cases, having 4 and 8 neighbors blocked yields different pin cost distributions (RULE6 vs. RULE9), i.e.,

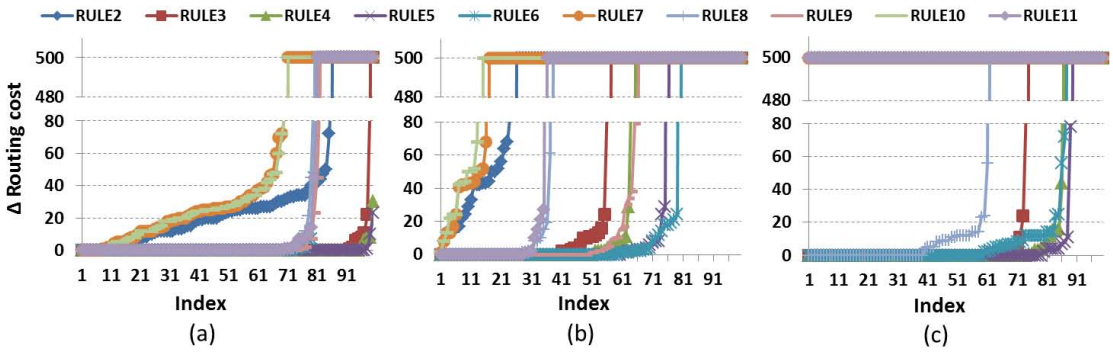


Figure 2.35: Δcost with different RULE* in (a) N28-12T, (b) N28-8T and (c) N7-9T.

the orthogonal via restriction is less dominant in this particular context. However, when via restriction is combined with SADP layers, the two forms of via restriction again show similar results (RULE7 vs. RULE10, RULE8 vs. RULE11).

With N7-9T (Figure 2.35(c)), SADP routing rules have less cost impact on layers above M4, as RULE4, RULE5 and RULE6 show similar Δcost distributions. When M3 is made into an SADP layer (RULE3), the vertical line (i.e., the clip index at which sorted Δcost goes to infinity (infeasible solution)) shifts left significantly. When the 4-neighbors via restriction is added to RULE3 (i.e., in RULE8), the vertical line shifts again. (RULE3, RULE4, RULE5, RULE6 and RULE8 respectively have 26, 14, 11, 13 and 39 infeasible clips out of 100.)

From the preceding discussions, we may tentatively form two general observations. (1) First, the via restriction and SADP routing rules show different trends, i.e., effects on the Δcost profile. Moreover, the sensitivities of Δcost to design rules and routing options vary with technology. For example, when SADP rules are applied to upper metal layers in N28-12T or N7-9T, the routing costs do not change significantly, which we interpret to mean that SADP rules do not affect routability significantly for these clips. This is different from what we observe in N28-8T. (2) Second, for design rules that are applied to upper metal layers ($>$ M3), almost half of routing clips show zero Δcost. This could imply that the pin cost metric of [178] cannot, by itself, accurately quantify the difficulty. In other words, there is a gap between pin accessibility metrics such as [178] and our switchbox-centric evaluation of routability.

Analysis of the number of variables and constraints. The number of directed arcs ($|A|$), the number of vertices ($|V|$) and the number of nets ($|N|$) determine the number of variables and constraints in the ILP. Without via restrictions and SADP rules (no restriction), the number of variables is $O(|A| \cdot |N|)$. With Constraints (2.43)–(2.46), the number of constraints is $O((|V| + 3 \cdot |A|) \cdot |N|)$. Regarding via restriction, when α neighbor sites are blocked, the

number of variable is the same as the basic case (no restriction), and the number of constraints is $O(\alpha \cdot |V| + (|V| + 3 \cdot |A|) \cdot |N|)$. With the SADP routing rules, the number of variables is $O((10 \cdot |V| + |A|) \cdot |N|)$ because of the additional binary indicator (p); the number of constraints is $O((34 \cdot |V| + 3 \cdot |A|) \cdot |N| + 10 \cdot |V|)$. Regarding via shapes, when a β size of via shape is considered, the number of variables is $O((\beta \cdot |V| + |A|) \cdot |N|)$ due to the creation of additional via edges, and the number of constraints is $O(\beta^2 \cdot |V| \cdot |N| + (\beta \cdot |V| + 3 \cdot |A|) \cdot |N|)$.

2.3.4 Conclusion

In this section, we have studied impacts of patterning technology choices and design rules on physical implementation metrics, with respect to *cost-optimal* design rule-correct detailed routing. We describe *OptRouter*, an ILP-based optimal detailed router that correctly handles multi-pin nets and various sub-20nm routing challenges including via restrictions, via shapes, and SADP patterning rules. OptRouter enables design rule evaluation using "difficult" routing clips (switchboxes) selected according to a pin cost metric. We study Δcost distributions for different design rules, relative to a RULE1 where all layers are LELE and there are no via restrictions. From the results, we observe that the sensitivities of Δcost to design rules and routing options vary with technology. Also, we observe that there is a gap between pin accessibility metrics such as [178] and our switchbox-centric evaluation of routability.

Future work includes speedup of OptRouter to gain insights into physical implementation impacts at larger granularity (switchbox size). Currently, OptRouter average runtime for a 7 track $\times$ 10 track switchbox ($1.0 \times 1.0 \mu m^2$ layout area in 28nm) is 1047 sec (single-threaded) with SADP and via restriction rules. Without such rules (as in [96]), average runtime is 842 sec.[31] As noted above, our results give insight into the degree of suboptimality in current routing tools, and open up the possibility of (massively distributed) local improvement of detailed routing solutions. Also, for better quantification of "difficult-to-route" clips, development of a metric beyond [178] to estimate routability in sub-20nm nodes will be an important aspect of our future work.

2.4 Acknowledgments

Chapter 2 contains reprints of Peter Debacker, Kwangsoo Han, Andrew B. Kahng, Hyein Lee, Praveen Raghavan and Lutong Wang, "MILP-Based Optimization of 2D Block

[31] OptRouter runtime for a 10 track $\times$ 10 track switchbox, with (resp. without) SADP and via restriction rules, is 1340 (resp. 925) sec.

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; Kwangsoo Han, Andrew B. Kahng and Hyein Lee, "Scalable Detailed Placement Legalization for Complex Sub-14nm Constraints", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2015; and Kwangsoo Han, Andrew B. Kahng and Hyein Lee, "Evaluation of BEOL Design Rule Impacts Using an Optimal ILP-Based Detailed Router", *Proc. ACM/ESDA/IEEE Design Automation Conference*, 2015. 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, Andrew B. Kahng, Hyein Lee, Praveen Raghavan and Lutong Wang.

Chapter 3

Process-Aware Design Methodologies

This chapter presents several distinct process-aware design methodologies, which includes OCV-aware top-level clock tree optimization, the mitigation of clock skew variation and die-to-die variation-aware partitioning in 3DIC. First, we present a new CTS methodology that optimizes clock logic cell placements and buffer insertions in the top level of a clock tree. Balancing the clock trees of multiple clocks is challenging because timing constraints depend on clock periods, and on the process, voltage and temperature (PVT) corners. In this study, we formulate the top-level clock tree optimization problem as a linear program that minimizes a weighted sum of timing slacks, clock uncertainty and wirelength. Experimental results in a commercial $28nm$ FDSOI technology show that our method can improve post-CTS worst negative slack across all modes/corners by up to $320ps$ compared to a leading commercial provider's CTS flow. Second, we propose a novel framework encompassing both global and local clock network optimizations to minimize the sum of skew variations across different PVT corners between all sequentially adjacent sink pairs. The global optimization uses linear programming to guide buffer insertion, buffer removal and routing detours. The local optimization is based on machine learning-based predictors of latency change; these are used for iterative optimization with tree surgery, buffer sizing and buffer displacement operators. Our optimization achieves up to 22% total skew variation reduction across multiple testcases implemented in foundry $28nm$ technology, as compared to a best-practices CTS solution using a leading commercial tool. Third, we study performance improvements of 3DIC implementation that leverage knowledge of mix-and-match die stacking during manufacturing. We propose partitioning methodologies to partition timing-critical paths across tiers to explicitly optimize the signed-off timing across the reduced set of corner combinations that can be produced by the stacked-die manufacturing. These include

both an ILP-based methodology and a heuristic with novel maximum-cut partitioning, solved by semidefinite programming, and a signoff timing-aware FM optimization. We also extend two existing 3DIC implementation flows to incorporate mix-and-match-aware partitioning and signoff, demonstrating the simplicity of adopting our techniques. Experimental results show that our optimization flow achieves up to 16% timing improvement as compared to the existing 3DIC implementation flow in the context of mix-and-match die stacking.

3.1 OCV-Aware Top-Level Clock Tree Optimization

In a modern SOC, *clock logic cells* (CLCs), such as clock gating cells (*CGC*s), multiplexers (*MUX*es) and dividers (*DIV*s), are required in the clock tree to achieve different performance and power saving requirements. To enable multi-mode operation and dynamic voltage frequency scaling (DVFS), large numbers of clocks are generated to drive flip-flops (FFs) in an SOC.[32] Balancing the clock trees of multiple clocks is challenging because timing constraints depend on clock periods, and on the process, voltage and temperature (PVT) corners. Furthermore, as on-chip variation (OCV) increases, *clock uncertainties* (derates) on the launch and capture paths can increase. Clock tree synthesis (CTS) must find optimal branching points in the clock tree to minimize clock uncertainties due to OCV on *non-common paths* [60][152][156]. Figure 3.1 (left) illustrates the clock balancing problem due to CLCs in a clock tree and the impact due to OCV. Due to the CLCs, the clock arrival times at flip-flop groups are skewed. Moreover, the clock tree splits near the clock source; this leads to long non-common paths between the flip-flop groups. As shown in Figure 3.1 (right), we can insert buffers to balance the clock, and optimize placement of the CLCs to reduce the non-common paths.

3.1.1 Motivation and Related Work

Given a clock tree, we represent the *top-level clock tree* as a hypergraph, $G_{top}(V_{top}, E_{top})$, in which V_{top} is a set of CLCs and the transitive fanin cells of the CLCs. E_{top} is a set of nets that connect the cells in V_{top}. Figure 3.2 shows a top-level clock tree with a CLC and three bottom-level buffered clock trees. In most cases, sophisticated EDA tools and CTS algorithms are able to achieve good solutions for the bottom-level clock trees. However, *achieving a good solution for the top-level clock tree can be problematic* when there are critical paths across the flip-flop groups between different bottom-level clock trees. The requirements

[32] Both synchronous and asynchronous clocks can exist in an SOC. Our work focuses on balancing synchronous clocks in an SOC.

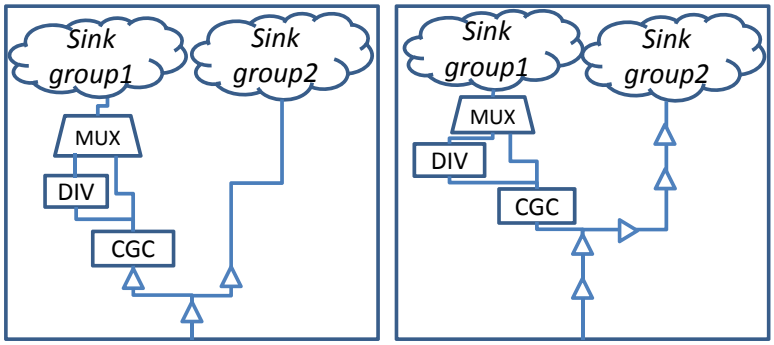


Figure 3.1: Clock tree synthesis problems.

to balance the top-level clock are not obvious due to the complex structure of the tree (see Figure 3.6). Fixing the critical paths across the FF groups can be difficult at the bottom-level clock trees due to tight timing constraints among FFs within the same group. To optimize timing across flip-flop groups, we propose to **balance the top-level clock tree while preserving the bottom-level clock trees**. For example, in Figure 3.2, if we increase the delay $d(1,2)$ on the net between *pins 1* and *2* from $2ns$ to $4ns$, we can change the skew between *flip-flop groups 1* and *2* from $2ns$ to $0ns$, thereby meeting the timing target of *critical path A* which has a clock period of $3ns$. Note that varying the delay on the top-level clock tree does not affect *critical path B* (but, the OCV derating on a longer top-level path will be larger), which has both its launch and capture FFs in the same group. Therefore, we only need to consider the requirements to balance clock across flip-flop groups, thereby simplifying the top-level clock tree optimization problem. Since problems arise in the top-level tree due to CLCs, our work focuses on optimizing the placement of CLCs and insertion of buffers in the *top-level clock tree*.

Related Work

Rajaram and Pan [152] propose CTS algorithms to optimize the chip-level clock tree across different PVT corners. They use quadratic programming to reallocate clock pins of IP blocks to reduce non-common paths in the chip-level clock tree. After clock pins are reallocated, buffers are inserted up to each pin, and subtrees are merged recursively in the same manner as the *deferred-merge embedding* (DME) algorithm [29]. The algorithm only inserts buffers that minimize the difference in clock latency among subtrees across PVT corners. Although the chip-level CTS work in [152] accounts for delay variation across PVT corners and timing penalty on non-common paths, it does not consider CLCs, timing between flip-flop groups, or wirelength, all of which make CTS a challenging task. As illustrated in Figure 3.1, the placement of CLCs should also be considered during CTS as it can significantly affect the non-common paths in

the tree. Other works [187][182] seek to minimize the effect of OCV during CTS, but do not address the issues of CTS with CLCs across multi-corner and multi-mode (MCMM) scenarios. Lung et al. [133] propose a linear programming (LP) based clock skew optimization [59] which accounts for delay variation across PVT corners. They also present a method to map the required delays obtained from the LP to actual circuits. While mapping delays, they use updated timing information to dynamically adjust buffer delays. Although this section addresses the MCMM clock skew minimization problem, it does not consider the effects of non-common paths and CLC placement. There are many previous works on buffer insertion for CTS (e.g., [8][31]), but they do not consider clock trees with CLCs which have different timing requirements depending on the operating modes and flip-flop groups. Papa et al. [145] minimize worst negative slack (WNS) at a single PVT corner by optimizing the placement and buffering of datapaths. They do not consider multiple PVT corners and they do not balance the top-level clock trees.

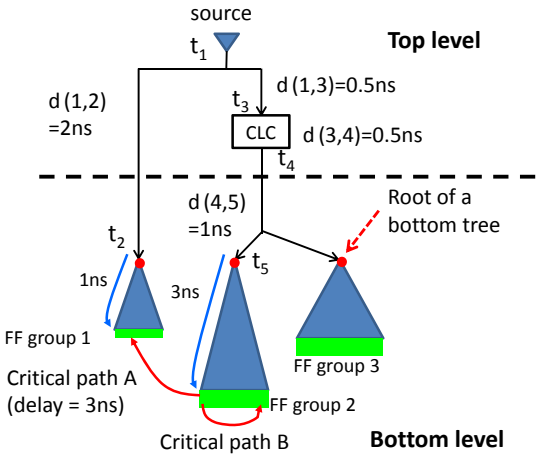


Figure 3.2: Example of balancing a clock tree by varying $d(i, j)$.

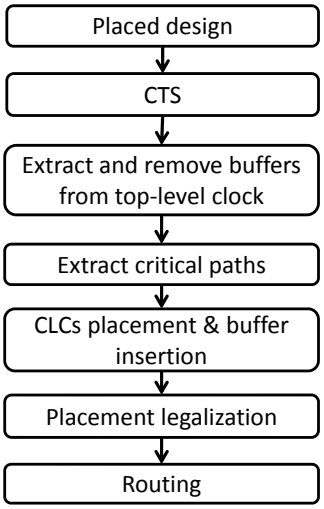


Figure 3.3: Overview of our CTS flow.

3.1.2 Our Approach

To address the top-level CTS problem mentioned above, we propose a *new* CTS flow that accounts for the effects of CLCs as well as delay variations due to MCMM and OCV. The basic idea of our approach is to *automatically* identify the requirements to balance clocks based on the timing critical paths and use them to drive the CTS. The flow shown in Figure 3.3 starts with a placed design and performs *conventional CTS* to obtain a clock tree. We then extract the top-level clock tree (see Algorithm 3) and remove buffers in the top-level clock tree.

Within the remaining (bottom-level) clock trees, we extract timing-critical FF-to-FF paths to identify the timing requirements for clock balancing. Based on these requirements, we construct a linear program (LP) to optimize the placement of CLCs and the delay on nets (achieved by inserting buffers) in the top-level clock tree. Unlike the routing algorithm proposed by Oh et al. [139] which minimizes the total wirelength of a routing tree, we include CLCs and Steiner point locations as variables in the LP, so that the LP-based optimization can account for the cost of non-common paths. With the physical locations of CLCs and Steiner points of the routes, we insert buffers in the top-level clock tree, legalize the placement and route the clock tree. The advantages of our methodology are as follows.

- Preserving the bottom-level clock trees affords more accurate timing information for the top-level clock tree optimization.[33]
- Since the top-level clock tree has many fewer instances, we can perform runtime-intensive optimizations which cannot be practically applied to the bottom-level clock tree.
- Introducing our new top-level clock tree placement optimization enables fixing of suboptimal CLC placements which have already been determined during the preceding placement stage.
- Buffer insertion and CLC placement optimization can achieve reductions of non-common path timing penalties, which are not achievable using local/incremental optimizations.

The key contributions of our work are summarized as follows.

- We propose a new automated clock tree synthesis methodology that optimizes the CLC placements and buffer insertion in the top-level clock tree.
- We propose an LP-based clock tree optimization method which accounts for routing resources (i.e., wirelength), circuit timing and the impact of non-common paths.
- Our method improves WNS by up to $320ps$, and reduce the top-level clock wirelength by up to 50% compared to a default CTS flow.
- As part of our validation process, we develop generators for testcases that represent clock tree structures typically found in high-speed IPs (e.g., graphics accelerators) and real-world SOCs.

In the remainder of this section, Section 3.1.3 describes our top-level clock tree optimization methodology. Section 3.1.4 describes experimental setup and our experimental results. In Section 3.1.5, we summarize our work and outline directions for future research.

[33] In this section, we optimize only the top-level clock tree. Joint optimization of the top- and bottom-level trees is a direction of ongoing work.

3.1.3 Clock Tree Optimization

We now explain the top-level clock tree optimization problem and our approach. In the following, we use *condition*, k, to denote that a timing value is specific to a PVT corner, clock group and timing analysis type (setup or hold). For example, with two PVT corners, two operating modes, two clock groups and two timing analysis types, k will range from $1, 2, ..., 16$.

Problem Statement

Formally, the top-level CTS problem is defined as follows.

Objective: Minimize the weighted sum of (i) worst negative slack, (ii) total negative slack (TNS), (iii) clock uncertainty and (iv) wirelength of a clock tree [152].

Input: Placed design; list of CLCs; timing constraints (SDC).

Output: An optimized placement of CLCs and clock buffers, clock routing of the top-level clock tree.

We model the cost of clock uncertainty $Z_k(a, b)$ on a critical path between flip-flops a and b as the sum of delays of the non-common launch and capture clock paths in the critical path. The non-common path delays are normalized to the clock period (CP) of the path using factor α_k.

$$\begin{aligned} Z_k(a,b) &= \alpha_k\{\sum_{i\in h_a, j\in h_b} d_k(i,j) - \sum_{i,j\in(h_a\cap h_b)} 2d_k(i,j)\} \\ \alpha_k &= 1/\text{CP at condition } k \end{aligned} \tag{3.1}$$

where h_a denotes a launch/capture path from a clock source to FF a, and $d_k(i, j)$ is the delay between pin i and j.

Our Approach

We formulate the top-level clock tree balancing problem as a linear program by assuming that we can vary (i) the delay $d_{ref}(i, j)$ from an output pin i to its fanout input pin j at a *reference condition*;[34] (ii) locations of CLCs; and (iii) Steiner points in the clock net (for a given topology). Although wire delay is normally nonlinear with respect to wirelength, we approximate $d_{ref}(i, j)$ as a linear function of distance between pin i and j assuming buffer insertion (as noted in, e.g., [145], the delay of a net with uniformly spaced buffers is linearly proportional to the number of stages).[35]

[34] The reference condition is {SS process corner, $0.85V, 125°C$}.

[35] A buffered net has relatively linear delay vs. distance even in advanced technology nodes. For example, the stage delay in a uniformly buffered-chain is almost the same except for the first few stages. Adding an additional stage will

The main objective of the LP is to minimize the weighted sum of worst negative slack S_{wns}, the total negative slack S_{tns}, non-common paths, $Z_k(a,b)$, and total wirelength $U(i,j)$.[36] Note that we weight the $Z_k(a,b)$ proportional to its original *negative* slack (i.e., $1 - s_k^0(a,b)$) such that the LP focuses on reducing the non-common path delay on timing paths. The critical paths and their original slacks $s_k^0(a,b)$ are extracted after the buffer removal step in Figure 3.3 by performing static timing analysis (STA).

To represent negative slack $s_k{}'(a,b)$ in the LP, we use Constraints (3.3) and (3.4) such that $s_k{}'(a,b) = 0$ when $s_k(a,b) > 0$. S_{wns} and S_{tns} are defined in Constraints (3.5) and (3.6), respectively. Since circuit designers may treat hold and setup slacks differently, we use a weight $\gamma_k \geq 0$ to set the ratio of importance (i.e., normalization ratio) of setup and hold slacks. The value of γ_k can be different for hold or setup analysis, as indicated by the condition k. We represent the timing slacks $s_k(a,b)$ for each timing-critical path between flip-flops a and b as a function of the original slack, original clock skew $\lambda_k(a,b)$, and the clock arrival times ($t_{ref}(a)$) in Constraint (3.7). Because delay and slack vary according to PVT corners and timing analysis type, we normalize the slacks across different conditions to a reference corner by using scaling factors η_k, following the approach in [133]. $\zeta = 1$ if the path is a setup-critical path and $\zeta = -1$ if the path is a hold-critical path. $t_{ref}(a)$ is the sum of delays along the path h_a (Constraint (3.8)).

increase the delay by a fixed amount. To account for the non-linearity within a single stage delay, our buffer insertion algorithm detour wires to match the required delay obtained from our LP.

[36] Our objective function is different from [152]. They do not consider wirelength and the timing between flip-flop groups.

Objective:

$$\text{Minimize } -w_{wns} \cdot S_{wns} - w_{tns} \cdot S_{tns} + w_{wl} \cdot \sum_{e(i,j) \in E_{top}} U(i,j)$$
$$+ w_{ncp} \cdot \sum_{k,a,b} (1 - s_k^0(a,b)) \cdot Z_k(a,b) + w_{dis} \cdot \sum_i M(i,i_0) \tag{3.2}$$

Subject to:

$$s_k{}'(a,b) \leq \alpha_k \cdot s_k(a,b), \ \forall a,b,k \tag{3.3}$$
$$s_k{}'(a,b) \leq 0, \qquad \forall a,b,k \tag{3.4}$$
$$S_{wns} \leq \gamma_k \cdot s_k{}'(a,b), \forall a,b,k \tag{3.5}$$
$$S_{tns} = \sum\nolimits_{a,b,k} \gamma_k \cdot s_k{}'(a,b) \tag{3.6}$$
$$\eta_k \cdot s_k(a,b) = \eta_k \cdot (s_k^o(a,b) - \lambda_k(a,b)) + \zeta(t_{ref}(a) - t_{ref}(b)) \tag{3.7}$$
$$t_{ref}(a) = \sum_{i,j \in h_a} d_{ref}(i,j) \tag{3.8}$$
$$d_{ref}(i,j) \geq \beta_{ref} \cdot U(i,j) \tag{3.9}$$
$$Z_k(a,b) = \alpha_k \{ \sum_{i \in h_a, j \in h_b} d_k(i,j) - \sum_{i,j \in (h_a \cap h_b)} 2 d_k(i,j) \} \tag{3.10}$$
$$M(i,j) = m_x(i,j) + m_y(i,j) \tag{3.11}$$
$$m_x(i,j) \geq (p_x(j) - p_x(i)), m_x(i,j) \geq 0 \tag{3.12}$$
$$m_y(i,j) \geq (p_y(j) - p_y(i)), m_y(i,j) \geq 0 \tag{3.13}$$
$$M(i,i_0) = m_x(i,i_0) + m_y(i,i_0) \tag{3.14}$$
$$m_x(i,i_0) \geq (p_x(i) - p_x(i_0)), m_x(i,i_0) \geq 0 \tag{3.15}$$
$$m_y(i,i_0) \geq (p_y(i) - p_y(i_0)), m_y(i,i_0) \geq 0 \tag{3.16}$$
$$0 \leq p_{\{x,y\}}(i) \leq F_{\{x,y\}} \tag{3.17}$$

The values of $\lambda_k(a,b)$ and the cell delays in $d_{ref}(i,j)$ are constants in the LP, and are extracted from STA reports after the buffer removal step in our flow. In Constraint (3.9), we model the delay $d_{ref}(i,j)$ between pins i and j as a linear function of the Manhattan distance $U(i,j)$ between the pins. β_{ref} is a conversion factor to convert the Manhattan distance to delay at the reference condition. We obtain the value of β_{ref} using the optimal repeater length method in [17]. The value of β_{ref} is $30ps$ per $100\mu m$ for a X8 buffer in the $28nm$ foundry FDSOI standard cell library that we use in our experiments. We calculate $Z_k(a,b)$ in Constraint (3.10). The Manhattan distances are calculated by using Constraints (3.11)–(3.13). The location of a pin i is specified by variables $p_x(i)$ and $p_y(i)$, which represent the x and y coordinates of the pin. The

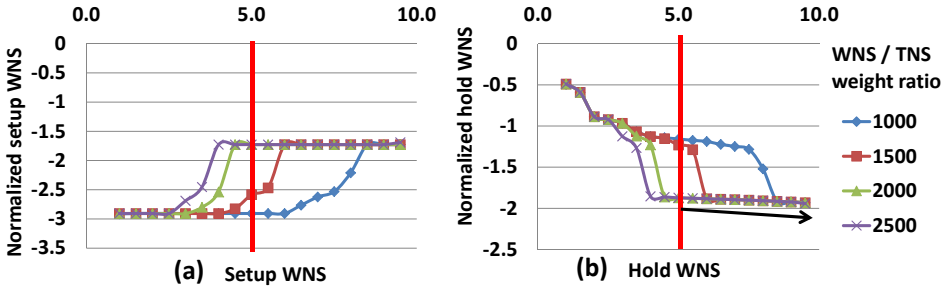


Figure 3.4: Normalized (a) setup WNS and (b) hold WNS obtained by solving the LP for different γ_k and w_{wns}/w_{tns}.

bounds for $p_x(i)$ and $p_y(i)$ are specified in Constraint (3.17). F_x and F_y are the upper bounds for the pin coordinates along the x and y axes, i.e., the dimensions of the design's floorplan.

To avoid unnecessary cell displacements, we add a *displacement cost* $M(i, i_0)$ in the objective function [145]. The displacement cost is defined as the sum of Manhattan distances between the original cell locations ($[p_x(i_0), p_y(i_0)]$) and their corresponding cell locations ($[p_x(i), p_y(i)]$) after optimization. $M(i, i_0)$ is calculated using Constraints (3.14)–(3.16). Since the displacement cost will force the LP to "pull" the cells to their original locations, we use a very small weighting factor ($w_{dis} = 0.001$) as the cell displacement cost. We apply uniform weights for TNS and non-common path delays, i.e., $w_{tns} = 1$, $w_{ncp} = 1$. Since the typical values of total wirelength in a top-level clock tree is much larger than the timing slacks we set $w_{wl} = 0.001$ such that the cost in the LP is not dominated by the wirelength.

Figures 3.4(a) and 3.4(b) respectively show the setup and hold WNS (both normalized to their corresponding clock periods) obtained by solving the LP for different values of γ_k. As we sweep γ_k from 1 to 10, the setup WNS obtained from the LP improves but the hold WNS worsens. When we sweep the w_{wns}/w_{tns} ratio, the setup and hold WNS are not affected when $\gamma_k \leq 3$. However, when $\gamma_k > 3$, the cost in the LP is dominated by the setup WNS and increasing the w_{wns}/w_{tns} ratio will improve the setup WNS. Since the hold time violations are relatively easy to fix by inserting buffers, we prioritize setup slacks when we select the γ_k and w_{wns}/w_{tns} weight ratios. In our experiments, we use $\gamma_k = 5$ and $w_{wns}/w_{tns} = 2000$ because we experimentally observe that by increasing γ_k further does not improve the setup WNS but makes hold WNS worse (black arrow in Figure 3.4(b)). We use the same values of the weighting factors across all testcases. It is also possible to apply different combinations of values of weighting factors, run the flows in parallel, and choose the best CTS solution.

Implementation Heuristics

Given a design with an initial clock tree, $G(V, E)$, and a subset of vertices $V_{CLC} \subseteq V$ corresponding to CLCs, we extract the top-level clock net using Algorithm 3.[37] First, we create a list V_{top} of all transitive fanin cells of the CLCs. In Lines 2–4, we remove all the clock routes connected to the fanin cells. In Lines 5–12, we check each cell in V_{top}, remove all the buffers and reconnect the nets accordingly.

Algorithm 3 Extract top-level clock tree.

Procedure *Extract_top*()
Input : $G(V, E)$, V_{CLC}
Output: $G(V_{top}, E_{top})$
1: $V_{top} \leftarrow$ transitive fanins of all $v \in V_{CLC}$;
2: **for** all $e(u, v) \in E$; $u, v \in V_{top}$ **do**
3: Remove clock routing for $e(u, v)$;
4: **end for**
5: $E_{top} \leftarrow \emptyset$
6: **for** $v \in V_{top}$ **do**
7: **if** v is a buffer **then**
8: $(v.parent).children \leftarrow v.children$;
9: $V_{top} \leftarrow V_{top} \setminus \{v\}$;
10: $E_{top} \leftarrow E_{top} \cup \{e(v.parent, v.children)\}$;
11: **end if**
12: **end for**
13: Return $G(V_{top}, E_{top})$;

In the top-level clock balancing problem, the LP optimizes the delays from an output pin to input pins in every net. For nets with more than one fanout, we modify the net into a binary tree by inserting Steiner points. The purpose of this step is to include the locations of the Steiner points as variables in the LP so as to optimize the non-common paths. Given a net, $G_{net}(V, E)$, and its driving pin, v_r, we apply Algorithm 4 to obtain a binary tree. In Lines 8–16, we find the pin pair that minimize the metric $\Delta L'$ which is defined as the sum of the difference in *sink latency*[38] and the delay due to the Manhattan distance between these pins.[39] In Lines 17–25, we merge the pin pair that has minimum $\Delta L'$ by creating a new Steiner point. We define the x and y coordinates of the new Steiner point as the average of the x and y coordinates of the merged pins (Lines 21–22). The sink latency of the Steiner point is defined as the maximum sink latency of the merged pins (Line 20). The procedure *split_net()* is invoked repeatedly until all driving pins have a single connection (to a Steiner point). Figure 3.5 illustrates our Steiner point insertion algorithm. In the first iteration, we merge pins j_2 and j_3 because they have the

[37] We obtain V_{CLC} by assuming all CLCs are in the top-level clock tree.
[38] The sink latency $L(u)$ of a pin u is the maximum latency from u to any flip-flop in the transitive fanout of u.
[39] We convert the Manhattan distance to delay by a conversion factor β_k at the reference condition.

smallest ΔL and Manhattan distance. Pins j_2 and j_3 are then connected to Steiner point $j_{2'}$ (red square). The location of $j_{2'}$ is defined by the average of the x and y coordinates of pins j_2 and j_3. In the second iteration, we merge pins j_1 with $j_{2'}$ because they have a smaller $\Delta L'$ even though the Manhattan distance between pins j_1 and $j_{2'}$ is larger than the Manhattan distance between pins j_4 and $j_{2'}$. In the last iteration, we merge j_4 and $j_{1'}$. Note that our algorithm selects the pins to merge based on the sum of Manhattan distance and the difference in sink latency. This is different from the algorithm in [52] which selects the pins based on Manhattan distance only. For example, the algorithm in [52] will merge j_2 and j_3, followed by j_4 and j_1. As shown in Figure 3.5 (the upper-right clock tree), the algorithm in [52] will lead to a clock tree that will require more buffers to be inserted (red arrows) to balance the clock latencies (green arrows) compared to the tree produced by our algorithm (the lower-right clock tree).

Algorithm 4 Create Steiner points.

Procedure *split_net*()
Input : $G_{net}(V, E), v_r \in V$
Output: $G'_{net}(V', E')$

$V' \leftarrow V$;
if ($|v_r.child| < 2$) **then**
  $E' \leftarrow E$;
else
  $E' \leftarrow \emptyset$;
  while ($|v_r.child| \geq 2$) **do**
    $min\_\Delta L' \leftarrow \infty$;
    for ($u_1, u_2 \in v_r.child$) **do**
      $\Delta L(u_1, u_2) \leftarrow |u_1.L - u_2.L|$;
      $\Delta L'(u_1, u_2) \leftarrow \beta_k \cdot M(u_1, u_2) + \Delta L(u_1, u_2)$;
      if ($\Delta L'(u_1, u_2) \leq min\_\Delta L'$) **then**
        $u_{min1} \leftarrow u_1$;
        $u_{min2} \leftarrow u_2$;
        $min\_\Delta L' \leftarrow \Delta L'(u_1, u_2)$;
      end if
    end for
    Create a new Steiner point $u' \notin V$;
    $v_r.child \leftarrow v_r.child \setminus \{u_{min1}, u_{min2}\}$;
    $u'.child \leftarrow \{u_{min1}, u_{min2}\}$;
    $u'.L \leftarrow max(u_{min1}.L, u_{min2}.L)$;
    $p_x(u') \leftarrow (p_x(u_{min1}) + p_x(u_{min2}))/2$;
    $p_y(u') \leftarrow (p_y(u_{min1}) + p_y(u_{min2}))/2$;
    $v_r.child \leftarrow v_r.child \cup \{u'\}$;
    $V' \leftarrow V' \cup \{u'\}$;
    $E' \leftarrow E' \cup \{e(u', u_{min1}), e(u', u_{min2})\}$;
  end while
  $E' \leftarrow E' \cup \{e(v_r, u')\}$;
end if
Return $G'_{net}(V', E')$;

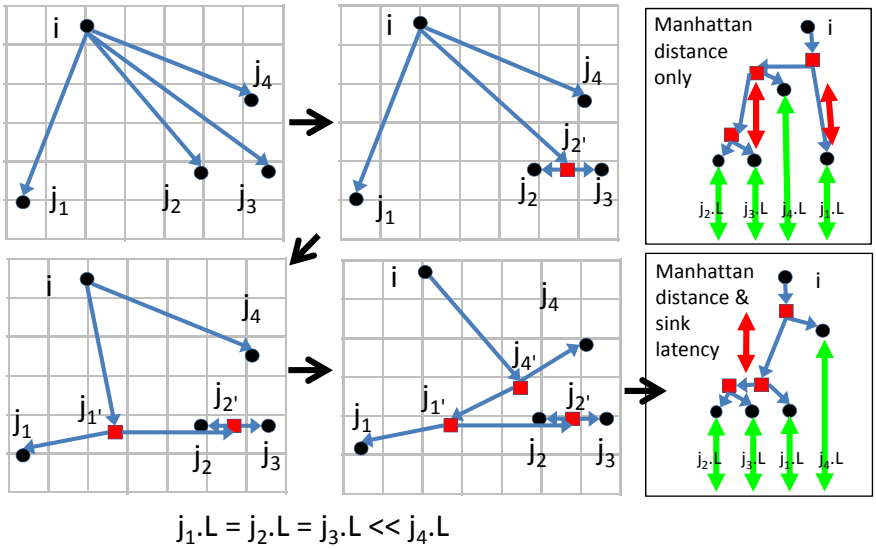


Figure 3.5: Steiner point creation. In each iteration, we find a pair of pins (black circles) or Steiner points with the minimum $\Delta L'$ (sum of scaled Manhattan distance and difference in sink latency) and connect them to a new Steiner point (red square).

By solving the LP, we obtain cell locations, clock routes (Steiner point locations) and net delays in the top-level clock tree. Next, we insert buffers in the top-level clock tree to guide clock routing and control clock skews. For each two-pin net in the optimized top-level clock tree, we insert buffers according to the steps described in Algorithm 5. In Line 1, we initialize the variable n, which indicates the number of inserted buffers, to 1. In Lines 2–14, we calculate the number of buffers required to meet the delay target as a function of net delays and buffer delays. M_{buf} is the minimum required spacing between two buffers.[40] The **while** loop exits when the sum of net and buffer delays (d_{est}) exceeds the required delay between the pins i and j (d_{req}). In Lines 15–21, we calculate the minimum wirelength required to insert n buffers. If this wirelength is less than or equal to the Manhattan distance between pins i and j, $M(i,j)$, we place the buffers in an L-shaped (y-axis first, followed by x-axis) manner. Otherwise, we place the buffers in a U-shaped manner because total wirelength is $> M(i,j)$. U-shaped placement is the general case, and L-shaped is a special case of U-shape when total wirelength is $\leq M(i,j)$.

3.1.4 Experimental Setup and Results

To test the effectiveness of our methodology, we require testcases with complex top-level clock trees. Since existing benchmarks [113][205] typically lack complex top-level clock trees, we generate testcases based on common clock tree structures typically found in high-speed

[40] We use M_{buf} = 5μm in our experiments.

Algorithm 5 Insert buffers.

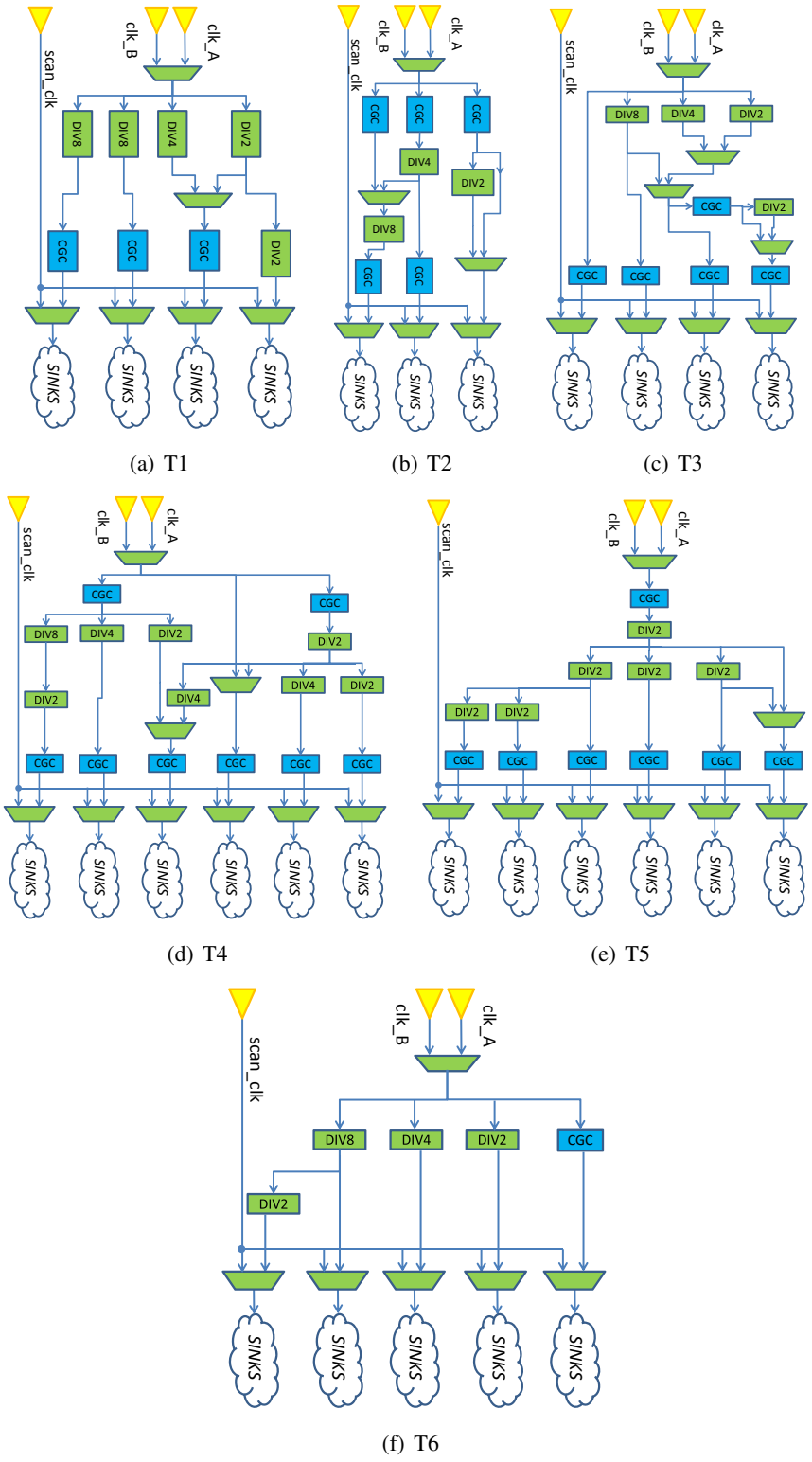
Procedure *insert_buffers*()
Input : pins i and j, $d_{req}(i,j)$
Output: inserted buffers

$n \leftarrow 1$;
// calculate number of buffers to meet required delay
while (1) **do**
  $l \leftarrow M(i,j)/(n+1)$;
  if ($l < M_{buf}$) **then**
    $l \leftarrow M_{buf}$;
  end if
  $d_{est} \leftarrow (n+1) \times d_w(l) + (n-1) \times d_g(c_{in\_buf} + c_w(l)) + d_g(c_{in}(j) + c_w(l))$;
  if ($d_{est} > d_{req}(i,j)$) **then**
    $n \leftarrow n-1$;
    break;
  else
    $n \leftarrow n+1$;
  end if
end while
if ($n > 0$) **then**
  if ($M_{buf} \times n > M(i,j)$) **then**
    Detour wire and place n buffers in U-shape;
  else
    Place n buffers in L-shape;
  end if
end if

SOCs and IPs [22][164]. The clock structures of our testcases are shown in Figures 3.6(a)–(f). We use dual-Vt $28nm$ foundry FDSOI libraries and implement each testcase at two operating modes – $\{1.25GHz$ at $0.95V\}$ and $\{1.667GHz$ at $1.20V\}$. We perform placement and routing (P&R) using a commercial tool and use *Synopsys PrimeTime vH-2013.06-SP2* [221] for timing analysis. Table 3.1 shows the timing analysis parameters in our experiments.

Testcase Description and Generation

Testcases from Tsay [186], Kahng and Tsao [113] and ISPD-2009/2010 [205] CNS contest benchmarks lack CLCs and are insufficient to create complex top-level clock hierarchies. Kahng et al. [105] improve CTS testcases by adding CLCs (Figures 3(a) and 3(b) in [105]) but two key elements ignored: (1) combinational logic between flip-flop groups and hence critical paths between flip-flop groups; and (2) multiple clock sources. The CTS problem becomes difficult when synchronous and asynchronous clocks need to be balanced across multiple flip-flop groups. We improve over [105] by (1) adding combinational logic with varying number of stages between flip-flop groups, (2) adding multiple synchronous and asynchronous clocks,



(a) T1 (b) T2 (c) T3

(d) T4 (e) T5

(f) T6

Figure 3.6: Clock structures of our testcases.

Table 3.1: Timing analysis setup.

Parameter	Value
PVT corner for setup analysis at the $1.250GHz$ mode	SS, $0.85V$, $125°C$
PVT corner for hold analysis at the $1.250GHz$ mode	FF, $1.05V$, $125°C$
PVT corner for setup analysis at the $1.667GHz$ mode	SS, $1.10V$, $125°C$
PVT corner for hold analysis at the $1.667GHz$ mode	FF, $1.30V$, $125°C$
Clock uncertainty	0.15 $\times$ clock period
Maximum transition for clock paths	$0.055ns$
Maximum transition for data paths	0.125 $\times$ clock period
Timing derate on net delay (early/late)	0.90 / 1.19
Timing derate on cell delay (early/late)	0.90 / 1.05
Timing derate on cell check (early/late)	1.10 / 1.10

(3) using CLCs at different hierarchies to make the clock balancing problem very complex, (4) creating multiple top-level clock hierarchies, and (5) performing CTS with MCMM and OCV constraints.

Figures 3.6(a)–(f) show the six testcases T1–T6 used in our experiments. These testcases use three clock sources typically seen in SOC designs [22] and can have large fanouts (e.g., $>$ 1000 flip-flops). The clock source *m_clk* is from the crystal oscillator, *clk* is the output of a PLL and *scan_clk* is the test clock. Clock sources *m_clk* and *clk* are used to implement low-power modes of operation, such as DVFS. The testcases use three kinds of dividers (*DIV2, DIV4, DIV8* in figures), a glitch-free clock MUX, and integrated clock gating cells (CGCs) as CLCs. Outputs of all dividers are sources of generated clocks; the generated clocks typically drive flip-flops for debug/tracing, IO and other peripheral logic.

To implement variable stages of combinational logic, we use *NetGen* [222] and vary #stages from 15 to 30. To model different critical paths, we connect flip-flips across groups as well as within the same group using these logic stages. To obtain floorplan dimensions that resemble SOCs, we use multiple instantiations of an interface logic module (ILM) of the *JPEG* design from *OpenCores* [212]. We create a netlist with the top module *5×JPEG*, in which we instantiate the *JPEG* design five times, perform SP&R and generate an ILM. Note that in this section, we do not optimize the bottom-level clock tree. Therefore, instantiation of the same *5×JPEG* multiple times (instead of using different modules) does not change the outcome of our experiments. We connect multiple instances of the ILM using combinational logic stages. For all CLCs, we implement custom netlists in the $28nm$ foundry FDSOI technology, and group flip-flops within the CLCs into their own skew groups so that these flip-flops do not affect global

skew and latencies. The path latencies of flip-flop groups are controlled by changing timing constraints and the number of stages of combinational logic between the groups. To allow a blockage-free placement region for the CLCs, we place ILM blocks (hard macros for the CTS tools) in an L-shaped manner along the periphery of the core as shown in Figure 3.7(a).

All testcases contain bidirectional paths, i.e., both launch and capture flip-flops appear in flip-flop groups that are driven by the fastest clock and other slower clocks. In addition, the fastest clock drives around 90% of the flip-flops that do not belong to the ILMs. Table 3.2 shows #CLCs, #cells, the flip-flops not in ILM, flip-flops in the ILM, flip-flops at the ILM boundary, and the area of each testcase (design in table). Testcases T2, T3 and T6 contain critical paths between flip-flops from two different clocks, one with large latency and the other with small latency. The CTS problem is complicated by the need to balance skew between these FF groups. Testcases T1–T4 contain multiple generated clocks and reconvergent paths between these clocks. These testcases make the CTS problem complex because skew needs to be balanced between fast and slow clocks. In testcases T3–T5, the control signals of CGCs are generated by *clk*, which makes the latency of the signal to the enable pin of the CGCs very critical. Besides balancing skews, CTS also needs to balance the critical path delays of the enable signal to the CGCs along with the clock latency. To report timing paths across clocks accurately, we set the path multiplier in the Synopsys Design Constraint (SDC) [18] file for paths between all clocks.

Table 3.2: Summary of testcases.

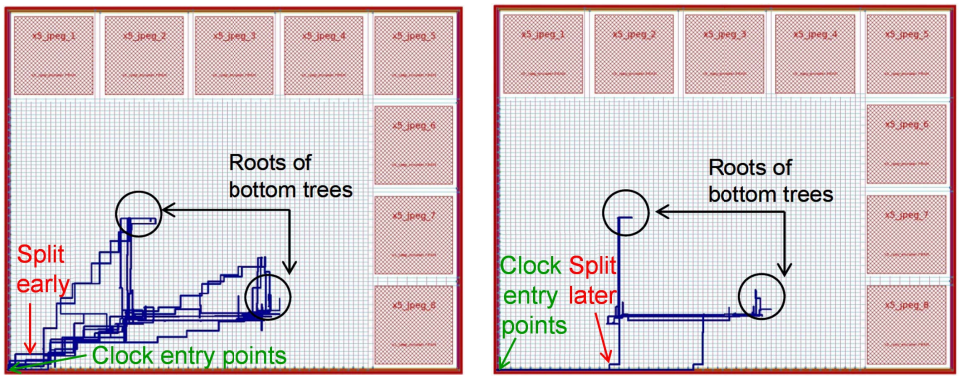
Design	**#CLCs**	**#Cells**	**#Flip-flops**			**Area**
			$\notin$ **ILM**	$\in$ **ILM**	**Boundary**	(mm^2)
T1	17	1.93M	10K	202.7K	1.7K	3.75×3.00
T2	12	1.93M	10K	202.7K	1.7K	3.75×3.00
T3	18	1.93M	12K	202.7K	1.7K	3.75×3.00
T4	24	1.93M	12K	202.7K	1.7K	3.75×3.00
T5	18	1.93M	8K	202.7K	1.7K	3.75×3.00
T6	13	1.92M	7K	202.7K	1.7K	3.75×3.00

Experimental Results

Table 3.3 summarizes the key metrics of the clock tree before (**I** = Initial, produced by a commercial tool) and after (**O** = Optimized) applying our top-level clock tree optimization. Rows 1–14 in Table 3.3 show the results at the post-CTS stage, while Rows 15-28 show the results at the end of the implementation flow (after datapath routing).[41]

[41] We apply the default clock tree optimization, routing and design optimization commands in the EDA tool after CTS. We do not compare our work with previous work as their algorithms cannot be applied to our testcases.

Post-CTS stage. Our optimization flow reduces the total wirelength of the top-level clock tree by 53% to 68% across all six testcases. Figure 3.7 shows that wirelength reduces because our flow clusters the CLCs such that the clock tree does not split near the clock entry points. The large wirelength reduction suggests that the initial CLC placements by EDA tools may not be aware of the CTS requirements. The smaller wirelength enables the optimized clock tree to also reduce the number of buffers. In testcases T4 and T5, the number of buffers is larger, as our optimization flow inserts more buffers in the clock tree to improve timing slack. To estimate switching power, we extract gate and wire capacitances of the top-level clock tree. Rows 5–6 in Table 3.3 show that our flow can reduce the switching power in the top-level clock tree by 12% to 40% for all testcases, including testcases T4 and T5, where the number of buffers increases.



(a) Initial clock tree. (b) Optimized clock tree.

Figure 3.7: (a) Initial and (b) optimized clock trees for testcase T6. Wiring of the top-level clock trees is shown in black. Our flow splits common paths farther from the clock root compared to the initial clock tree. As a result, the total wirelength in the top-level clock tree is reduced from 45mm to 22mm.

Our flow also improves the setup WNS and TNS by up to 550ps and 255ns, respectively (Rows 7–10). Hold WNS and TNS are also improved except for testcase T6, in which the hold WNS and TNS worsen by 110ps and 780ps, respectively (Rows 11–14). Our optimization flow can worsen hold WNS and TNS because we focus on improving the setup slacks ($\gamma_k = 5$). The tradeoff between setup and hold slacks is based on the following assumptions: (1) hold time violations are easier to fix in post-CTS implementation stages and (2) some of the hold time violations are fixed by the increased wire delays in the routing stage.

In Rows 30–34 of Table 3.3, we report runtimes of the main procedures in our optimiza-

tion flow. We spend most of the time to extract timing information and to formulate the LP.[42] CLC placement, buffer insertion, legalization and routing only take 10 minutes in total because there are not many cells in the top-level clock tree. The total runtime is 135 minutes on average. Testcase T3 has a higher runtime because it has more timing-critical paths than other testcases (Row 29).

Post-datapath routing stage. To study the benefits of our optimization flow, we also compare the post-routing results between the initial and the optimized clock trees. The results in Table 3.3 show that all designs with the optimized clock tree have the same or improved setup WNS compared to the designs with the initial clock tree (Rows 21–24). The improvement in setup WNS at the post-routing stage is up to $320ps$. Although some testcases with the optimized clock tree have worse hold slacks (i.e., testcases T4, T5 and T6), the differences are less than $100ps$. The results in Rows 15–16 show that our optimization flow reduces the total wirelength by 38% to 51% across all six testcases. The improvements are smaller as compared to the post-CTS stage because the total wirelength of the initial and optimized clock trees both increase at the post-routing stage due to wiring of the signal nets. Total number of buffers and switching power at the post-routing stage are similar to values seen at the post-CTS stage.

3.1.5 Conclusion

Designing a balanced top-level clock tree with multiple clock sources is very complex as we need to consider MCMM, OCV and timing constraints across flip-flop groups. We develop a CTS methodology that optimizes CLC placement and buffer insertion, and that minimizes non-common paths between flip-flop groups. We formulate the top-level CTS problem as the minimization of a weighted sum of WNS, TNS, clock uncertainty due to OCV and wirelength. We solve this problem using LP and develop heuristic flows to insert Steiner points and buffers, which are required elements of a top-level CTS solution. We also develop generators for testcases that resemble clock tree structures typically found in high-speed SOCs. We validate our optimization flow on testcases from our generators and achieve up to 51% reduction in wirelength for the top-level clock tree, and $320ps$ improvement in WNS, compared to a leading commercial CTS tool. Our future work includes (i) handling obstacles, (ii) accounting for *optimal* buffering solutions, (iii) creating testcases to capture other important SOC elements such as memory controller and multimedia blocks, and (iv) joint optimization of the top- and bottom-level clock trees.

[42] Solving the LP takes less than 30 sec.

Table 3.3: Post-CTS results. I: Initial, O: Optimized.

Testcase:			T1	T2	T3	T4	T5	T6
Post-CTS								
1	Top-level	I (μm)	18086	19261	41476	38830	34009	36052
2	wirelength	O (μm)	8442	8614	13193	14389	14186	15104
3	Total-level	I	163	210	361	298	322	253
4	buffers	O	152	167	242	301	421	226
5	Switching	I (μW)	875	1018	1639	1515	1557	1315
6	power	O (μW)	590	692	969	1210	1360	987
7	Worst	I (ns)	-0.05	-0.10	-0.37	-0.65	-0.55	-0.32
8	setup WNS	O (ns)	-0.04	0.00	-0.36	-0.55	0.00	-0.20
9	Total	I (ns)	-0.41	-0.25	-48.47	-1034.38	-8.39	-40.56
10	setup TNS	O (ns)	-0.17	0.00	-45.47	-779.46	0.00	-12.78
11	Worst	I (ns)	0.00	0.00	-0.40	-0.04	0.00	-0.04
12	hold WNS	O (ns)	0.00	0.00	-0.40	-0.01	0.00	-0.15
13	Total	I (ns)	0.00	0.00	-130.12	-0.21	0.00	-0.09
14	hold TNS	O (ns)	0.00	0.00	-128.23	-0.05	0.00	-0.87
Post-datapath routing								
15	Top-level	I (μm)	26261	30779	58223	50432	48761	44794
16	wirelength	O (μm)	15750	19097	33982	27342	28570	22051
17	Total-level	I	163	215	357	300	322	252
18	buffers	O	152	170	248	306	427	226
19	Switching	I (μW)	885	1100	1748	1592	1616	1337
20	power	O (μW)	638	729	1042	1220	1374	968
21	Worst	I (ns)	-0.03	0.00	-0.05	-0.58	-0.32	-0.19
22	setup WNS	O (ns)	0.00	0.00	0.00	-0.46	0.00	-0.18
23	Total	I (ns)	-0.05	0.00	-0.06	-883.50	-3.03	-10.81
24	setup TNS	O (ns)	0.00	0.00	0.00	-609.28	0.00	-1.10
25	Worst	I (ns)	0.00	0.00	-0.37	-0.04	0.00	0.00
26	hold WNS	O (ns)	0.00	0.00	-0.10	-0.11	-0.04	-0.05
27	Total	I (ns)	0.00	0.00	-19.82	-0.14	0.00	0.00
28	hold TNS	O (ns)	0.00	0.00	-5.46	-0.78	-0.08	-0.33
29	Total timing paths in LP		16K	20K	72K	40K	28K	11K
Runtime (minutes)								
30	Extract timing		45	37	176	71	71	25
31	Formulate LP		36	26	165	51	36	9
32	Place & legalization		8	4	6	5	6	5
33	Clock routing		7	4	5	4	5	5
34	Total		96	71	352	131	118	44

3.2 A Global-Local Optimization Framework for Simultaneous Multi-Mode Multi-Corner Clock Skew Variation Reduction

Modern SOCs typically exploit complex operating scenarios to maximize performance and reduce power consumption. For instance, techniques such as dynamic voltage and frequency scaling (DVFS), split rail power supply, etc. are widely applied in SOC designs to meet performance and power targets. However, these techniques increase the number of modes and corners used for timing closure, which will in turn lead to increased datapath delay variation and clock skew variation across corners. Such large timing variations increase area and power overheads, as well as design turnaround time (TAT) due to a "ping-pong" effect whereby fixing timing issues at one corner leads to violations at other corners. To solve this issue, we can minimize either datapath delay variation or *clock skew variation* across corners. Given that datapath optimization is a local optimization and is usually applied after the clock network optimization, what datapath delay variation minimization can accomplish is limited. In other words, datapath optimizations are practically less impactful than minimizing clock skew variations in most cases. This is why clock network optimization is a key first step during the physical implementation flow for timing closure. Further, clock skew variation can be achieved via both global and local optimizations of the clock network. Therefore, minimizing clock skew variation across corners is more effective for multi-corner timing closure. In this section, we minimize clock skew variation.

Moreover, timing violations due to clock skew variation across corners are typically reduced by (hold and/or setup) buffer insertion, V_{th}-swapping and gate sizing on datapaths at later design stages. Thus, clock skew variation between each pair of sequentially adjacent sinks can lead to potential costs of area, power and design TAT. We therefore minimize the sum of skew variations between all sink pairs to minimize the overall physical implementation costs (e.g., in area, power, TAT).

Although many commercial EDA tools are capable of multi-mode multi-corner clock network synthesis [176][220], our optimization framework can be applied as an incremental optimization for further reduction of skew variations in light of our robust interface to commercial P&R and STA tools. Moreover, experimental results show that our proposed optimization is able to achieve significant skew variation reduction on clock networks that have been synthesized with a leading commercial tool.

Contributions of our work are as follows.

1. We are the first in the literature to study the problem of minimizing the *sum of clock skew variations* across multiple PVT corners.

2. We propose a novel global-local framework for clock network optimizations to minimize the sum, over all pairs of PVT corners, of skew variation between all sequentially adjacent pairs.

3. We demonstrate that machine learning-based predictors of latency change can provide accurate guidance on the best moves to test during local optimization for minimization of skew variation across corners.

4. Our optimization framework has a robust interface to leading commercial P&R and STA tools and production PDKs/libraries, and can be generalized to other clock network optimization problems.

5. We achieve up to 22% reduction in the sum of skew variations of clock trees in testcases that reflect high-speed application processor and memory controller blocks.

3.2.1 Related Work

We classify previous works on clock skew optimization as (i) works that target skew and/or delay optimization at single or multiple corners and (ii) works that optimize skew variation across multiple PVT corners.

Several previous works optimize skew at one or more PVT corners, but do not address skew variation across corners. Cao et al. [24] minimize the worst skew in a clock tree by partitioning the tree into different skew groups. The authors then greedily minimize the worst skew in each skew group to minimize overall local skew. Cho et al. [34] perform clock tree optimization that is temperature-aware. The authors modify the deferred merge embedding (DME) algorithm to include *merging diamonds* for consideration of temperature variations to guide clock skew and wirelength minimization. Lung et al. [133] perform multi-mode multi-corner (MMMC) clock skew optimization by minimizing the worst skew across all corners. They propose a methodology to determine the *delay correlation factor* for clock buffers at $130nm$, $90nm$ and $65nm$ and conclude that the correlation across corners is linear. However, such an assumption might not be valid at $28nm$ and below. Lung et al. [134] perform chip-level as well as module-level clock skew optimizations with multiple voltage modes. The authors use power-mode-aware buffers for chip-level clock tree optimization. For the module-level optimization, they only consider the worst voltage corner.

Relatively fewer works exist that optimize skew variation across multiple PVT corners. Restle et al. [159] propose a two-dimensional nontree structure. They divide the nontree struc-

ture into two levels – leaf level (close to clock sinks) and top level (close to clock source). The top level is the same as the traditional clock tree structure, but the leaf level is a mesh structure such that each sink is connected to the nearest point on the mesh. Although this is a very effective way to minimize skew variation across corners, the mesh structure consumes enormous wire resources and power. Su and Sapatnekar [175] use mesh structures for the top-level tree which consumes less wire resource and power as compared to [159]. However, this consumes 59%-168% more wire resource than a tree structure. Further, the authors do not optimize skew variation which still exists in the bottom-level subtrees. Rajaram et al. [151][153] propose a non-tree construction method to insert crosslinks[43] in a clock tree by estimating subtree delays using Elmore delay model. The authors verify their method with SPICE-based Monte Carlo simulations and report skew variability reduction. However, the approach consumes excess additional wire and power due to crosslink insertions. Mittal and Koh [137] propose a greedy method to insert crosslinks to reduce skew variation.

To our knowledge, there has been no systematic framework for minimization of *clock skew variation* (across multiple signoff corners) for clock trees. Our work exploits both global and local iterative optimizations to minimize skew variations across different PVT corners which is very important for high-speed processor and multimedia blocks that operate at multiple modes and corners. Further, instead of minimizing the maximum skew or skew variation, we minimize the sum of skew variations over all sink pairs, which will reduce the potential costs of gate sizing and buffer insertion for multi-corner timing closure.

3.2.2 Problem Formulation

The notations we use in this section are given in Table 3.4.

For a corner pair (c_k, $c_{k'}$), we define the normalized skew variation between sink pair (f_i, $f_{i'}$) as

$$v_{i,i'}^{c_k,c_{k'}} = |\alpha_k \cdot skew_{i,i'}^{c_k} - \alpha_{k'} \cdot skew_{i,i'}^{c_{k'}}| \tag{3.18}$$

where skew ($skew_{i,i'}^{c_k}$) is defined as the latency difference between capture and launch clock paths at c_k. We emphasize that our optimization is local skew-aware, so that we only optimize skews between launch-capture sink pairs that have valid datapaths in between them (i.e., we avoid the pessimism that would result from use of global skew in the formulation). α_k is the normalization factor at corner c_k with respect to the nominal corner. Note that α_k is an input parameter and can be determined by technology information (e.g., ratio between buffer delays at

[43] A crosslink is an additional wire between any two nodes of a given clock tree.

Table 3.4: Notations.

Term	Meaning
c_k	operating corner, ($0 \leq k \leq K$; c_0 is the nominal corner)
α_k	normalization factor of corner c_k with respect to c_0
f_i	sink (e.g., flip-flop) in clock tree, ($1 \leq i \leq N$)
P_i	clock path from clock source to f_i
$skew_{i,i'}^{c_k}$	clock skew between sink pair (f_i, $f_{i'}$) at corner c_k
s_j	arc (i.e., tree segment without branching) in clock tree, ($1 \leq j \leq M$)
$D_j^{c_k}$	original arc delay at corner c_k
$\Delta_j^{c_k}$	delay change of arc s_j at corner c_k from optimization
$D_{max}^{c_k}$	maximum latency of a clock path at corner c_k
$v_{i,i'}^{c_k,c_{k'}}$	normalized skew variation across corner pair (c_k, $c_{k'}$) between (f_i, $f_{i'}$)
$V_{i,i'}$	worst normalized skew variation across all corner pairs between (f_i, $f_{i'}$)

c_k and c_0), clock tree properties (e.g., V_{th} and sizes of buffers in the tree), etc. Further, one can define specific α_k values for each sink pair. In our work, we define α_k as the average skew ratio between c_0 and c_k over all sink pairs.

We further define the maximum skew variation across corners, for each sink pair (f_i, $f_{i'}$) as

$$V_{i,i'} = \max_{\forall (c_k, c_{k'})} v_{i,i'}^{c_k,c_{k'}} \tag{3.19}$$

Based on the above, we address the following problem formulation:

Skew variation reduction problem. Given a routed clock tree, minimize the sum over all sink pairs of the maximum normalized skew variation across all corners.

$$\text{Minimize} \sum_{\forall (f_i, f_{i'})} V_{i,i'} \tag{3.20}$$

3.2.3 Optimization Framework

Figure 3.8 illustrates our optimization framework. We perform global and local optimizations to reduce skew variations. The global optimization constructs a linear program (LP) and uses it to guide buffer insertion, buffer removal, and routing detours. Local optimization is based on a machine learning-based predictor of latency changes. It iteratively minimizes skew variation via tree surgery (i.e., driver reassignment), buffer sizing, and buffer displacement. The iterative optimization continues until there is no further improvement or other stopping condition is reached.

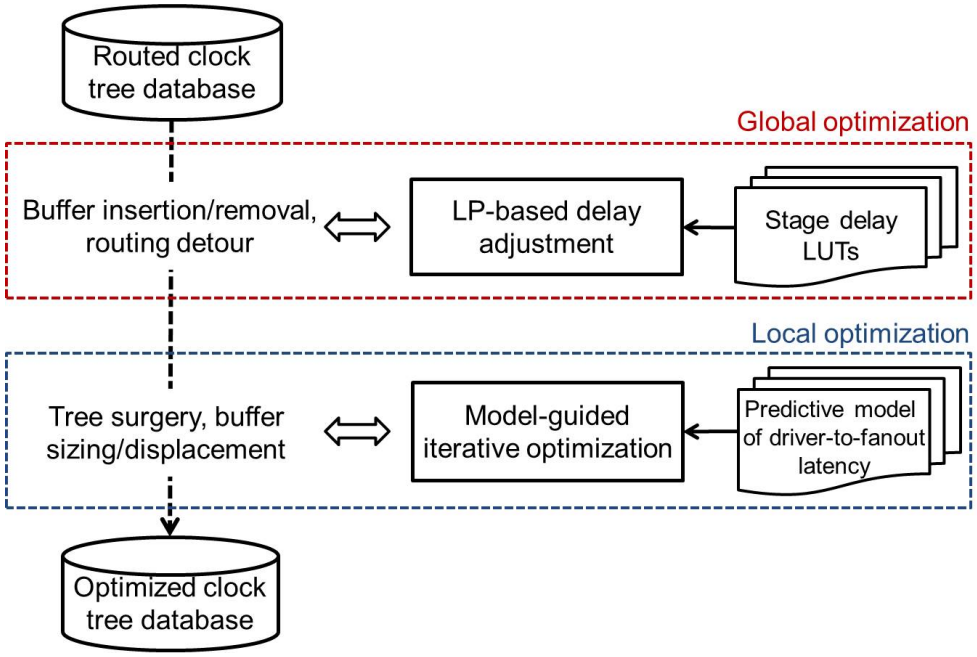


Figure 3.8: Overview of our optimization framework.

Global Optimization

We construct a linear program (LP) to reduce the sum of skew variations between all sink pairs in a clock tree. Based on the LP solution, we determine the desired delay changes of arcs at all corners and perform buffer insertion and removal, as well as routing detour, to accomplish the desired delay changes. We determine number of buffers, buffer size and length of routing detour based on lookup tables. However, the achievable delay values are discrete due to the limited number of buffer sizes. Further, placement legalization and routing congestion also lead to discrepancy between desired delay and actual delay after ECOs in the P&R tool. Therefore, to minimize the sum of skew variations as well as to increase the likelihood that the solution is practically implementable, we formulate the LP such that it minimizes the total amount of delay changes with respect to an upper bound on sum of skew variations. As a result, we implicitly minimize the number of ECO changes. We then sweep this upper bound to search for the achievable solution with minimum sum of skew variations. The objective function is:

$$\text{Minimize} \sum_{1 \le j \le M,\ 0 \le k \le K} |\Delta_j^{c_k}| \tag{3.21}$$

where $\Delta_j^{c_k}$ is the latency change on arc s_j at corner c_k.[44] The upper bound U on the sum of skew variations is specified as

$$\sum_{(f_i, f_{i'})} V_{i,i'} \leq U \tag{3.22}$$

where $V_{i,i'}$ is the maximum normalized skew variation for the sink pair (f_i, $f_{i'}$) over all corner pairs (c_k, $c_{k'}$), and is calculated based on the following constraint.

$$\begin{aligned}
V_{i,i'} \geq & \alpha_k \cdot (\sum_{s_{j'} \in P_{i'}} (D_{j'}^{c_k} + \Delta_{j'}^{c_k}) - \sum_{s_j \in P_i} (D_j^{c_k} + \Delta_j^{c_k}) \\
& - \alpha_{k'} \cdot (\sum_{s_{j'} \in P_{i'}} (D_{j'}^{c_{k'}} + \Delta_{j'}^{c_{k'}}) - \sum_{s_j \in P_i} (D_j^{c_{k'}} + \Delta_j^{c_{k'}})) \\
V_{i,i'} \geq & \alpha_{k'} \cdot (\sum_{s_{j'} \in P_{i'}} (D_{j'}^{c_{k'}} + \Delta_{j'}^{c_{k'}}) - \sum_{s_j \in P_i} (D_j^{c_{k'}} + \Delta_j^{c_{k'}})) \\
& - \alpha_k \cdot (\sum_{s_{j'} \in P_{i'}} (D_{j'}^{c_k} + \Delta_{j'}^{c_k}) - \sum_{s_j \in P_i} (D_j^{c_k} + \Delta_j^{c_k}))
\end{aligned} \tag{3.23}$$

We further constrain the optimization such that the solution returned does not degrade (i) local skew at any corner, nor (ii) the skew variation between corner pairs (c_k, c_0), for all arcs on clock paths at all non-nominal corners c_k.

$$\begin{aligned}
\sum_{s_{j'} \in P_{i'}} (D_{j'}^{c_k} + \Delta_{j'}^{c_k}) - \sum_{s_j \in P_i} (D_j^{c_k} + \Delta_j^{c_k}) \leq & |\sum_{s_{j'} \in P_{i'}} D_{j'}^{c_k} - \sum_{s_j \in P_i} D_j^{c_k}| \\
\sum_{s_j \in P_i} (D_j^{c_k} + \Delta_j^{c_k}) - \sum_{s_{j'} \in P_{i'}} (D_{j'}^{c_k} + \Delta_{j'}^{c_k}) \leq & |\sum_{s_{j'} \in P_{i'}} D_{j'}^{c_k} - \sum_{s_j \in P_i} D_j^{c_k}|
\end{aligned} \tag{3.24}$$

$$\begin{aligned}
& \alpha_k \cdot (\sum_{s_{j'} \in P_{i'}} (D_{j'}^{c_k} + \Delta_{j'}^{c_k}) - \sum_{s_j \in P_i} (D_j^{c_k} + \Delta_j^{c_k})) \\
& - \sum_{s_{j'} \in P_{i'}} (D_{j'}^{c_0} + \Delta_{j'}^{c_0}) - \sum_{s_j \in P_i} (D_j^{c_0} + \Delta_j^{c_0}) \\
& \leq |\alpha_k \cdot (\sum_{s_{j'} \in P_{i'}} D_{j'}^{c_k} - \sum_{s_j \in P_i} D_j^{c_k}) - (\sum_{s_{j'} \in P_{i'}} D_{j'}^{c_0} - \sum_{s_j \in P_i} D_j^{c_0})| \\
& \sum_{s_{j'} \in P_{i'}} (D_{j'}^{c_0} + \Delta_{j'}^{c_0}) - \sum_{s_j \in P_i} (D_j^{c_0} + \Delta_j^{c_0}) \\
& - \alpha_k \cdot (\sum_{s_{j'} \in P_{i'}} (D_{j'}^{c_k} + \Delta_{j'}^{c_k}) - \sum_{s_j \in P_i} (D_j^{c_k} + \Delta_j^{c_k})) \\
& \leq |\alpha_k \cdot (\sum_{s_{j'} \in P_{i'}} D_{j'}^{c_k} - \sum_{s_j \in P_i} D_j^{c_k}) - (\sum_{s_{j'} \in P_{i'}} D_{j'}^{c_0} - \sum_{s_j \in P_i} D_j^{c_0})|
\end{aligned} \tag{3.25}$$

[44] We formulate $\Delta_j^{c_k}$ as positive and negative components to handle the absolute values in our formulation.

We also bound the maximum latency for each clock path as follows.

$$\sum_{s_j \in P_i} (D_j^{c_k} + \Delta_j^{c_k}) \leq D_{max} \tag{3.26}$$

For each arc, we specify the upper and lower bounds on the latency change. The lower bound $D_{min\ j}^{c_k}$ is determined by the delay with optimal buffer insertion, without any routing detour. The upper bound of delay change is defined as β times of the original arc delay, in which β can be selected empirically (we assume β = 1.2 in this section).

$$D_{min\ j}^{c_k} \leq D_j^{c_k} + \Delta_j^{c_k} \leq \beta \cdot D_j^{c_k} \tag{3.27}$$

To increase the likelihood that the LP solution is practically implementable, we characterize lookup tables at each corner for stage delays of inverter pairs[45] with various gate sizes and routed wirelengths between consecutive inverters. We define the stage delay between inverter pairs as the sum of gate delays of the two inverters in a pair and the delays of their fanout nets (Figure 3.10). Based on the characterized lookup tables, we observe that for a given stage delay per unit distance at c_0 (i.e., the ratio between stage delay and routed wirelength for an inverter pair), the stage delay ratios between pairs of corners are limited by the buffer insertion solutions in lookup tables. Figure 3.9 shows the stage delay ratios between pairs of corners (c_0, c_1) and (c_0, c_2), respectively. In the plot, each circle represents an inverter pair with a particular gate size, routed wirelength between consecutive inverters, input slew and load capacitance. We use polynomial fit to determine upper ($W_{max}^{c_k,c_{k'}}$) and lower ($W_{min}^{c_k,c_{k'}}$) bounds of delay ratios for each pair of corners, which are shown as the red curves. Any delay ratio larger (smaller) than the upper (lower) bound is not achievable with available buffer insertion solutions in lookup tables. Furthermore, we assume that the delay per unit distance of an arc does not vary significantly in our optimization due to Constraints (3.24)–(3.27). Thus, we use delay per unit distance of an arc in the original clock tree to estimate upper and lower bounds of delay ratios ($W_{min,max}^{c_k,c_{k'}}$), and apply these bounds in Constraint (3.28) to avoid LP solutions that are not practically implementable by ECOs.

$$W_{min}^{c_k,c_{k'}} \leq \frac{D_j^{c_k} + \Delta_j^{c_k}}{D_j^{c_{k'}} + \Delta_j^{c_{k'}}} \leq W_{max}^{c_k,c_{k'}} \tag{3.28}$$

Complexity analysis. The LP formulation (Constraints (3.21)–(3.28)) has $O(M \cdot K)$ variables to indicate delay change on each arc at each corner ($\Delta_j^{c_k}$); there are also $O(N^2)$ (i.e.,

[45] In this section, we assume that the buffers used to construct the clock tree are comprised of inverter pairs. But, our methodologies apply to clock trees with both inverting and non-inverting buffers.

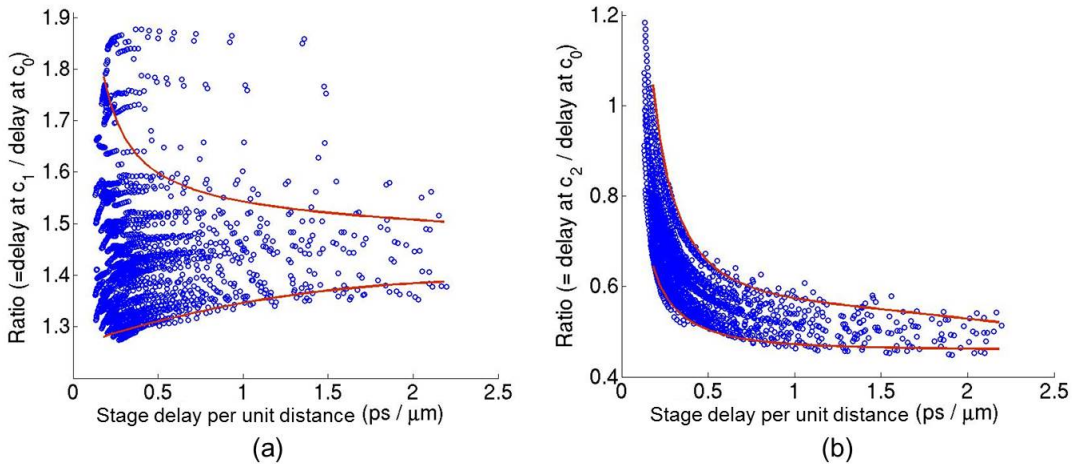


Figure 3.9: Delay ratios between (c_1, c_0) and (c_2, c_0), respectively. c_0 = (SS, 0.9V, -25°C, Cmax), c_1 = (SS, 0.75V, -25°C, Cmax) and c_2 = (FF, 1.1V, 125°C, Cmin).

the number of sink pairs) variables to indicate the maximum normalized skew variation across all corner pairs between each sink pair ($V_{i,i'}$). There are $C(K,2)$ constraints to force $V_{i,i'}$ to be no less than the maximum normalized skew variation between each sink pair (Constraint (3.23)); $(4 \cdot K)$ constraints to prevent local skew and skew variation degradations (Constraints (3.24)–(3.25)); N constraints to specify the maximum latency (Constraint (3.26)); $(2 \cdot M)$ constraints to bound arc delay changes (Constraint (3.27)); and $C(K,2)$ constraints to enhance ECO feasibility (Constraint (3.28)).

ECO implementation. We apply ECO changes to accomplish the desired arc delays at each corner, which are determined by LP solution. Given that the buffer insertion problem is NP-complete [172], although we apply several techniques to enhance ECO feasibility, the LP formulation still cannot guarantee an optimal solution that is practically implementable. Thus, our target is to minimize the discrepancy between the desired delays in the LP solution and those that actually result from ECOs. In our ECO implementation, we first remove all original inverter pairs on the arc. We then determine the solution (i.e., gate size and routed wirelengths between consecutive inverters) of inverter pair insertion based on the characterized lookup tables with stage delays. Note that in this section, we always use one gate size, and uniformly place inverter pairs, for each individual arc. We place inverter pairs in a "U" shape when routing detour is required. The lookup table contains stage delays with five inverter sizes and routed wirelengths between consecutive inverters varying from 10μm to 200μm with a step size of 5μm across different corners. Since these lookup tables are technology-dependent, we only perform the characterization once per technology. More specifically, we have two lookup tables: (i)

LUT_{detail} is characterized with different input slew and fanout load capacitance, and is applied for the first and last inverter pairs of a given arc, and (ii) $LUT_{uniform}$ is characterized based on average stage delay of inverter pairs in an arc, and is applied for the inverter pairs in the middle of an arc (Figure 3.10).

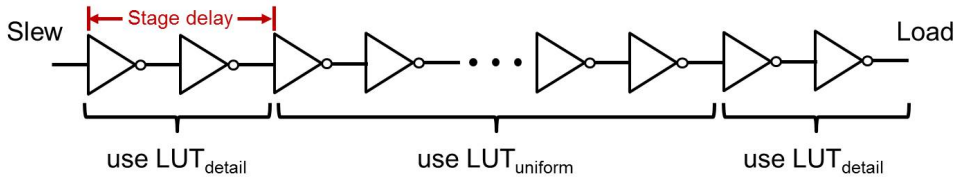


Figure 3.10: LUT_{detail} is characterized with various input slews and fanout load capacitances; $LUT_{uniform}$ contains average stage delay with particular gate size and routed wirelengths between consecutive inverters.

Algorithm 6 describes the flow to select solutions for inverter pair insertions. For each combination of gate size and routed wirelength between consecutive inverters, we estimate a range of desired number of inverter pairs (i.e., $[max(u_{est}-2, 0), u_{est}+2]$) based on the average stage delay in $LUT_{uniform}$ at corner c_0 (Lines 5-6). $D_{LP}^{c_k}$ is the required arc delay at corner c_k in the LP solution. We then assess error for each potential solution (i.e., a combination of gate size, routed wirelength between consecutive inverters and number of inverter pairs) and select the solution with minimum error (Lines 7-16). We use p and q to respectively index the gate size and the routed wirelength between consecutive inverters. $D_{est}^{c_k}$ is the estimated delay using LUTs. Last, we implement ECO changes based on the selected solution (Lines 19 and 21).

Local Optimization

We apply local iterative optimization to further minimize the sum of skew variations across corners. More specifically, we consider three types of local moves, which are illustrated in Figures 3.11(b)–(d) and in Table 3.5 – Type (I) buffer sizing and/or buffer displacement, Type (II) displacement of a buffer and gate sizing on one of its child buffers, and Type (III) tree surgery (i.e., reassignment of a (child) node to a different (parent) driver). However, performance of such iterative optimization is usually limited by its large turnaround time. For instance, each local move requires placement legalization, ECO routing, parasitic extraction, and timing analysis in the golden timer.[46] Given such large turnaround time, it is practically impossible to explore all possible local moves for a given design. Therefore, a fast and accurate model to predict

[46]In our experiments, the runtime for each local move on a testcase with 1.79M instances and 270K flip-flops, using one thread per analysis corner on a 2.5GHz Intel Xeon server, is around 70 minutes (i.e., 30 minutes for ECO and parasitic extraction, and 40 minutes for timing analysis).

Algorithm 6 LP-guided ECO flow.

for all s_j to be optimized **do**
  Remove current inverter pairs on s_j
  $err_{min} \leftarrow +\infty$; $sol \leftarrow \emptyset$
  for p := 1 to N_{size}, q := 1 to N_{WL} **do**
    $u_{est} \leftarrow$ round$(D_{LP}^{c_k} / d(LUT_{uniform})_{p,q}^{c_k})$
    for $u := max(u_{est} - 2, 0)$ to $u_{est} + 2$ **do**
      $err \leftarrow 0$
      for k := 0 to K **do**
        $err \leftarrow err + |D_{est}^{c_k} - D_{LP}^{c_k}|$
      end for
      for all corner pair $(c_k, c_{k'})$ **do**
        $err \leftarrow err + |(D_{est}^{c_k} - D_{est}^{c_{k'}}) - (D_{LP}^{c_k} - D_{LP}^{c_{k'}})|$
      end for
      if $err < err_{min}$ **then**
        $err_{min} \leftarrow err$; $sol \leftarrow (p, q, u)$
      end if
    end for
  end for
  Perform ECO inverter pair insertion based on sol
end for
Legalize all inserted inverters and perform ECO routing

the impact of local moves is necessary. Previous work [81] has demonstrated that machine learning-based models are quite accurate for delay and slew estimation. In our work, we apply a two-stage machine learning-based model for prediction of arc delay changes with local moves. The overarching goal is to be able to accurately predict *delta-latency*, i.e., the change in post-ECO routing source-sink delays that results from a given buffer's resizing and/or placement perturbation.

Machine learning-based model. To predict the impact of a local move, we first estimate new routing pattern (if the move contains displacement or tree surgery) by constructing two types of trees – FLUTE [37] tree and single-trunk Steiner tree. We approximate wire delays correspondingly using Elmore delay and D2M [6] models. We then update the delay and output slew of the driver based on the estimated wire capacitance and update pin capacitance (if the move sizes the child node) by performing interpolation in the Liberty table. Last, we perform slew propagation using PERI [116] and update gate delays one and two stages downstream based on Liberty tables.[47] However, as observed in [81], the interpolated delay values do not always match those from the golden timer's analysis. Further, the estimated routing pattern as well as wire delay can have discrepancy with respect to the commercial router's actual ECO solution.

[47] Our analyses show that the delay and slew change of buffers beyond two stages is $< 1ps$, so we do not update timings of buffers beyond two stages downstream.

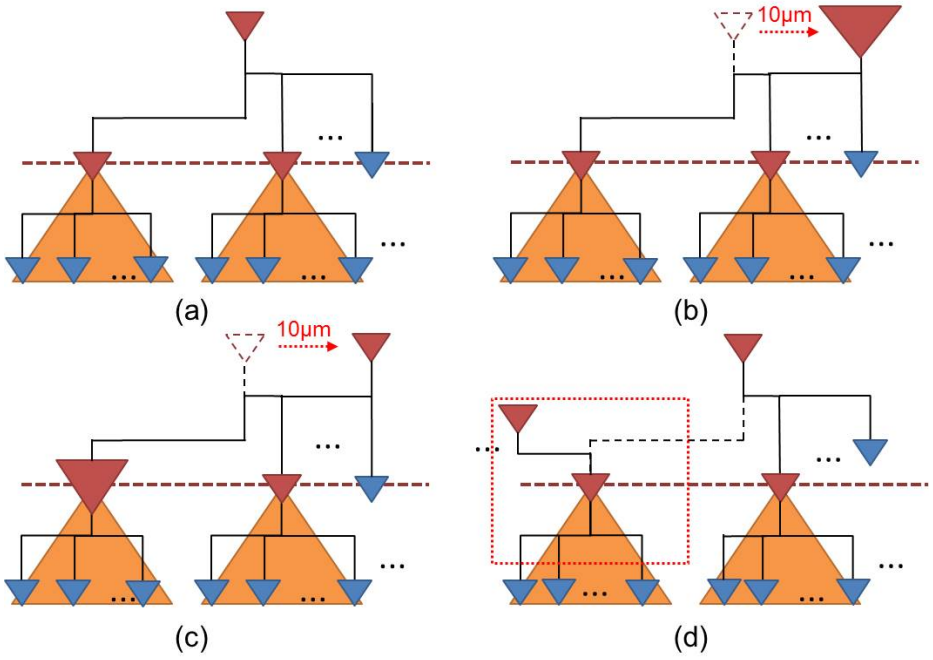


Figure 3.11: Local optimization moves used in our flow. (a) Initial subtree; (b) sizing and/or displacement; (c) displacement and sizing of child node; and (d) tree surgery, i.e., driver reassignment.

We therefore construct machine learning-based models to minimize such discrepancy. We use Artificial Neural Networks (ANN) [84], Support Vector Machines (SVM) with a Radial Basis Function (RBF) kernel [84], and Hybrid Surrogate Modeling (HSM) [106].[48] In addition to the estimated delays based on {FLUTE tree, single-trunk Steiner tree} $\times$ {Elmore delay, D2M}, the input parameters to the machine learning-based model also include the number of fanout cells, as well as the area and aspect ratio of the bounding box which contains driving pin and fanout cells. To generate training data, we construct artificial testcases (i.e., clock trees) that resemble real designs with fanout ranging from 1-5 (20-40 for last-stage buffers) and bounding box area and aspect ratio of the driven pins ranging from $1000\mu m^2$ to $8000\mu m^2$ and from 0.5 to 1, respectively. We then place fanout cells or sinks randomly within the bounding box. We generate 150 artificial testcases and perform 450 moves on average to each testcase (the runtime for one testcase is $\sim$1 hour). Note that we only construct one model for each corner, and that this model is applied to all designs.

We create one delta-latency model for each corner used in our experiments. Figure 3.12(a) shows the predicted vs. actual latencies that we compute from the predicted delta latencies by our model at corner c_3 in Table 3.6. Figure 3.12(b) shows the corresponding histogram

[48] Further details of the applied machine-learning techniques that we use may be found in [84] and [106].

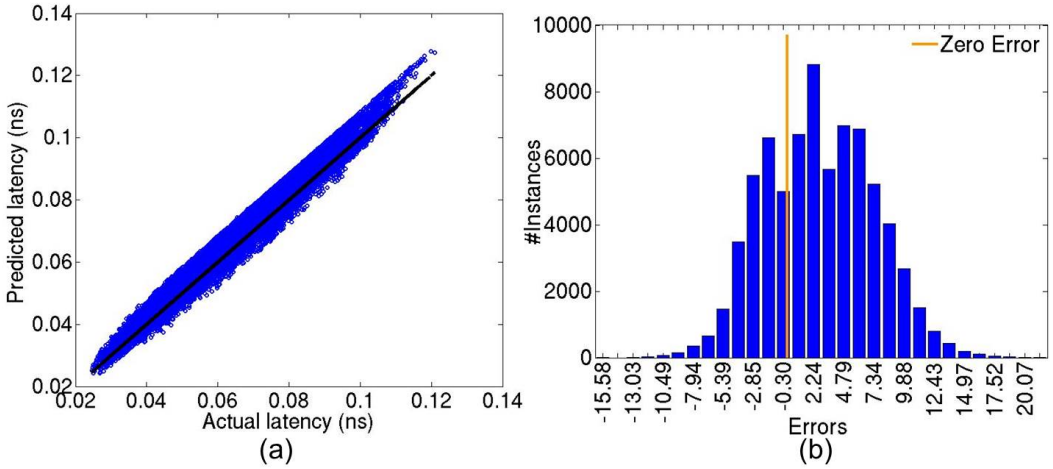


Figure 3.12: Examples of (a) predicted vs. actual latencies and (b) percentage error histograms from our model for the c_3 corner in Table 3.6.

of percentage errors. Across all the corners, our modeling error is 2.8% on average. The absolute of maximum and minimum errors are 21.98% and 16.21% respectively. The modeling for each corner using the artificial testcases is a one-time effort. On a $2.5GHz$ Intel Xeon server, the time to train a model for each corner is around 5 hours with four threads. Models for each corner can be trained in parallel, e.g., on a server with 24 threads, we can train six models in 5 hours. Our models generalize to different testcases because (i) our training dataset generated from the artificial testcases span ranges of parameters that are typically seen in clock trees in SOC application processors and memory controllers, and (ii) we prevent overfitting by performing cross-validation. Our experimental results indicate that our models are generalizable and accurate when applied to "unseen" testcases during the model training phase. Figure 3.2.3 shows the accuracy comparison between our learning-based model and analytical models. We observe that with fewer attempts, our learning-based model is able to identify the best move for more buffers.

Iterative optimization flow. Based on our model, we perform iterative local optimization flow illustrated in Algorithm 7. We first enumerate all candidate local moves and generate the input data to our model (Line 1). The moves we consider in this section are shown in Table 3.5. We predict the delta-latency resulting from each move based on our model (Line 2). We then estimate the skew variation reductions based on the predicted latency changes. Our experimental results show that we are able to evaluate the impacts of more than 160K moves at three corners in 17 minutes on a $2.5GHz$ Intel Xeon server with 15 threads. We sort the candidate moves in decreasing order of their predicted skew variation reductions, and pick the top R (i.e.,

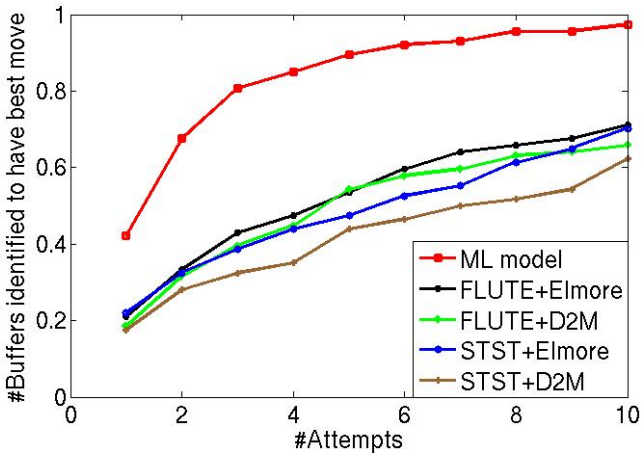


Figure 3.13: Accuracy comparison between our learning-based model and analytical models. An attempt is an ECO. There are 114 buffers, and each buffer has 45 candidate moves. In one attempt, the learning-based model (resp. analytical models) can identify best moves for 40% (resp. up to 20%) of the buffers.

R = 5 in this section) moves to implement in R individual threads (Line 3). Last, we perform timing analysis using the golden timer to assess the actual skew variation changes (Line 4). If there is skew variation reduction, we update the database with the minimum skew variation solution. Otherwise, we implement the next R moves (Lines 5-9). The iteration terminates when there is no move showing skew variation reduction according to our predictor.

Algorithm 7 Iterative optimization flow.

1: Enumerate all candidate moves and generate input data to model
2: Predict delta-latency and skew variation reductions
3: Implement R moves with maximum predicted skew variation reductions using R threads
4: Assess actual skew variation reductions with the golden timer
5: **if** there is skew variation reduction **then**
6: Update database with the minimum skew variation solution
7: **else**
8: Implement the next R moves and go to Line 4
9: **end if**

Table 3.5: Candidate moves in our optimization.

Type	Candidate moves
I	displace {N, S, E, W, NE, NW, SE, SW} by $10\mu m \times$ one-step up/down sizing
II	displace {N, S, E, W, NE, NW, SE, SW} by $10\mu m \times$ one-step up/down sizing on one child node
III	reassign to a new driver (i) at the same level as current driver, and (ii) within bounding box of $50\mu m \times 50\mu m$

3.2.4 Experimental Setup and Results

Our experiments are implemented in foundry $28nm$ LP technology. We construct the original clock tree and perform ECO optimizations using *Synopsys IC Compiler vI-2013.12-SP1* [220]. We use *Synopsys PrimeTime vH-2013.06-SP3* [221] and *Synopsys PrimeTime-PX vH-2013.06-SP3 PT-PX* [221] for timing and power analyses, respectively. We construct the machine learning-based model using *MATLAB vR2013a* [210]. The optimization flow is implemented using C++ and Tcl scripts.

Testcase Description

We develop two classes of testcase generators to validate our proposed optimization framework. Class *CLS1* corresponds to clock networks typically observed in high-speed application processors and graphics processors. Class *CLS2* corresponds to clock networks in memory controllers, which are typically used in SOCs to interface SOC components with DRAM/eDRAM. We implement our testcases at $28nm$ LP technology. The corners used in our experiments are shown in Table 3.6. We use the testcase generation methodology described in [26], and the top-level structures of the testcases T1 and T2 in [26]. We modify the floorplan and clock tree synthesis flow to develop two variants of *CLS1*, *CLS1v1* and *CLS1v2*. Each of *CLS1v1* and *CLS1v2* contains four identical $650\mu m \times 650\mu m$ interface logic modules (ILMs) to resemble four cores of an application processor. These are floorplanned in a rectangular block such that the utilization of standard cells is $\sim$60% before placement.[49] Figure 3.14(a) shows the floorplan of *CLS1v1*. We implement the *CLS1* class testcases at corners c_0, c_1 and c_3 as shown in Table 3.7. Corners c_0 and c_1 are setup-critical, and c_3 is hold-critical. Table 3.7 summarizes various post-synthesis metrics of these testcases.

Table 3.6: Description of corners.

Corner	Process	Voltage	Temperature	Back-end-of-line
c_0	SS	$0.90V$	-25°C	Cmax
c_1	SS	$0.75V$	-25°C	Cmax
c_2	FF	$1.10V$	125°C	Cmin
c_3	FF	$1.32V$	125°C	Cmin

[49] We understand from our industry collaborators that best-practices flows for high-speed and memory controller blocks start with 50%–60% utilization before placement [165].

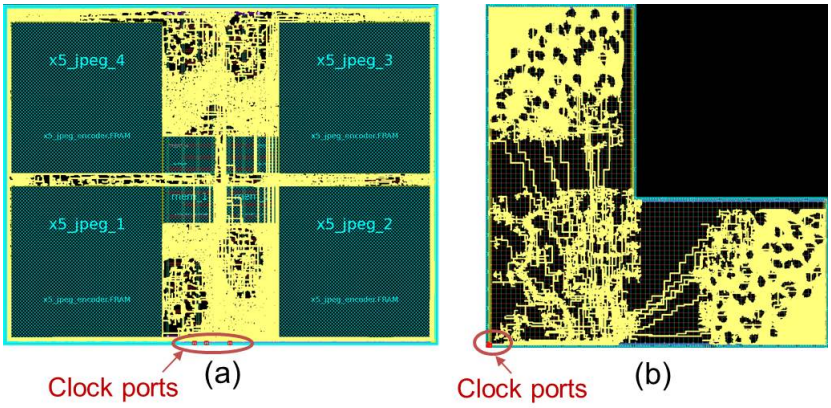


Figure 3.14: Floorplans of (a) *CLS1v1* and (b) *CLS2v1*. In yellow are routed clock nets.

Table 3.7: Summary of testcases.

Testcase	#Cells	#Flip-flops	Area	Util	Corners
CLS1v1	0.4M	36K	$3.3mm^2$	62%	c_0, c_1, c_3
CLS1v2	0.4M	35K	$3.4mm^2$	60%	c_0, c_1, c_3
CLS2v1	1.79M	270K	$4.5mm^2$	58%	c_0, c_1, c_2

We also study a testcase *CLS2v1* of class memory controller, which is new as compared to [26]. Table 3.7 summarizes the post-synthesis metrics of this testcase, and Figure 3.14(b) shows its floorplan. We use the methodology described in [26] to generate random logic and connect this logic to flip-flops; this includes datapaths across different clock groups. The memory controller is floorplanned in an L-shaped block with the controller at the center and the interface logic in each of the top and bottom arms of the L-shape. The interface logic has data and control signals across memory, processor and other blocks. The control signals are generated within the controller, and the flip-flops in the interface logic and controller are separated by large distances (e.g., $\sim 1mm$). The large distance between sequentially adjacent sinks leads to large clock skew, which the commercial tool tries to balance by inserting buffers. However, these clock buffers lead to skew variations across corners. We implement the *CLS2v1* testcase at corners c_0, c_1 and c_2 as shown in Table 3.7, where c_0 and c_1 are setup-critical and c_2 is hold-critical.

For implementations of all our testcases, we follow a production methodology [165]. We set the skew target as $0ps$ in the CTS tools, as our studies (with skew targets ranging from $0ps$ to $250ps$, in steps of $50ps$) indicate that a target skew of $0ps$ steers the tool to deliver the smallest skew at each corner. We perform clock tree optimizations with both multi-corner multi-mode (MCMM) scenario as well as multi-corner single-mode (MCSM) scenario at each mode. We then select the optimized clock tree solution with minimum skew variation as the input to our optimization.

Results

Table 3.8 shows the experimental results,[50] where **variation**, **skew**, **#cells**, **power** and **area** are respectively the sum of normalized skew variations over union of top 10K critical sink pairs (in terms of setup and hold timing slacks) at each corner,[51] local skew at each corner, total number of clock cells, clock tree power and total area of clock cells. In the experiments, we apply three optimization flows to each of the testcases: (i) *global* is the global optimization flow, (ii) *local* is the local iterative optimization flow, and (iii) *global-local* performs global and local optimizations in sequence. The global (local) optimization alone achieves up to 16% (5%) reduction on the sum of skew variations. Since local moves affect only a subset of sink pairs, they have smaller impact than that of the global optimization. By combining the two optimizations, we reduce the sum of skew variations by 22% with negligible area and power overhead. The results also show no degradation of local skews. Further, we observe that the local iterative optimization reduces skew variations more when applied after the global optimization, as compared to a standalone local skew optimization (e.g., for *CLS1v1*, local optimization achieves $13ns$ more reduction with a prior global optimization, as compared to the standalone local optimization).

Table 3.8: Experimental results.

Testcase	**Flow**	**Variation [norm]** (ns)	**Skew** (ps)			**#Cells**	**Power** (mW)	**Area** (μm^2)
			c_0	c_1	$c_{2,3}$			
CLS1v1	*orig*	**512** [1.00]	214	530	226	2515	0.355	3615
	global	431 [0.84]	179	395	188	2553	0.356	3705
	local	493 [0.96]	214	529	223	2515	0.355	3621
	global-local	**399** [0.78]	175	387	188	2553	0.356	3706
CLS1v2	*orig*	**585** [1.00]	272	594	259	2762	0.369	3968
	global	518 [0.89]	269	575	235	2762	0.369	3975
	local	557 [0.95]	258	545	259	2762	0.369	3970
	global-local	**510** [0.87]	265	564	235	2762	0.369	3975
CLS2v1	*orig*	**972** [1.00]	179	192	282	5568	0.865	8556
	global	888 [0.91]	175	192	232	5574	0.866	8577
	local	926 [0.95]	180	190	282	5568	0.865	8556
	global-local	**841** [0.87]	176	192	232	5574	0.866	8557

[50] Our optimization does not create any maximum transition or maximum capacitance violations.
[51] The number of optimized sink pairs for *CLS1v1*, *CLS1v2* and *CLS2v2* are respectively 15012, 14671 and 15142.

Figure 3.15 shows the skew variation reduction during the local iterative optimization. We observe that tree surgeries (type-I moves) are more effective than sizing and displacement moves (type-II and type-III moves), and are applied by our model in the early iterations. For *CLS1v1*, we also show the results with 10 random moves (dots in black), where the gap between random move and our optimization is $15ns$. This validates the benefits of our delta-latency model. The runtimes per iteration (with 15 threads) are 60 min, 80 min and 200 min for testcases *CLS1v1*, *CLS1v2* and *CLS2v1*, respectively.

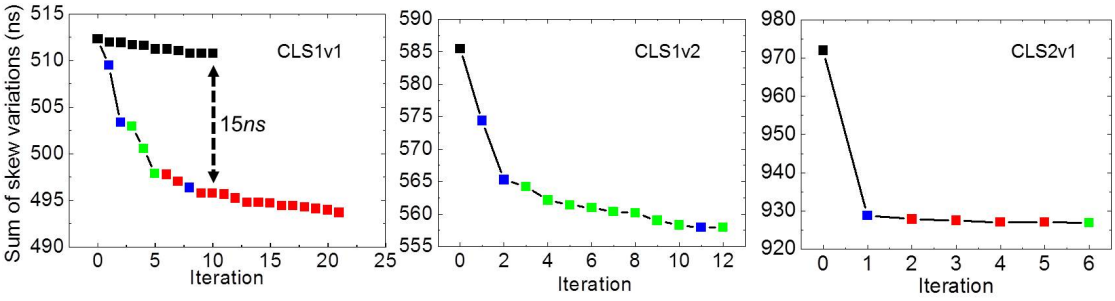


Figure 3.15: Sum of skew variations decreases during the local iterative optimization. In blue are type-I moves, in red are type-II moves, and in green are type-III moves.

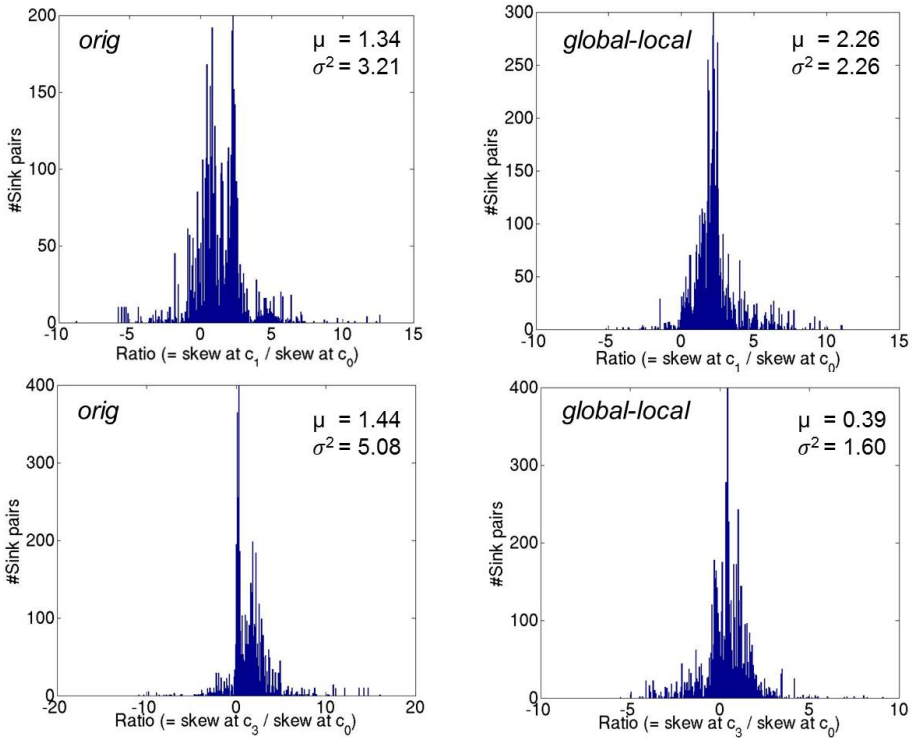


Figure 3.16: Distribution of skew ratios between (c_1, c_0) and (c_3, c_0) of (a) original clock tree, and (b) optimized clock tree for *CLS1v1*.

Figure 3.16 shows the distributions of skew ratios between corner pairs (c_1, c_0) and (c_3, c_0), over sink pairs, of the initial clock tree and the optimized clock tree. We observe that our optimization significantly reduces the variation and range of skew ratios between corner pairs.

3.2.5 Conclusion

In this section, we propose the first framework to minimize the sum of skew variations over all sequentially adjacent sink pairs, using both global and local optimizations. Our experimental results show that the proposed flow achieves up to 22% reduction of the sum of skew variations for testcases implemented in foundry $28nm$ technology, as compared to a leading commercial tool. In the global optimization, our LP formulation comprehends the ECO feasibility based on characterized lookup tables of stage delays. In the local optimization, we demonstrate that machine learning-based predictors of latency changes can provide accurate estimation of local move impacts.

Our future works include: (i) study of the resultant power and area benefits of reduced skew variation; (ii) development of models to predict a buffer location for minimum skew over a continuous range of possible buffer locations; (iii) explorations, motivated by our current results, of new library cells whose delay and slew are less sensitive to corner variation so as to enable fine-grained ECOs based on our LP solutions; and (iv) investigation of whether a worse initial start point (clock network with large skew variations) can enable us to achieve smaller skew variation across corners using our optimization flow.

3.3 Improved Performance of 3DIC Implementations Through Inherent Awareness of Mix-and-Match Die Stacking

Small footprint and high transistor density in three-dimensional integrated circuits (3DICs) make 3D logic-logic integration an important future lever for cost and density scaling. Specific to 3DICs, a number of works [27][57][63][99] have pointed out that "mix-and-match" of multiple stacked die, according to binning information, can improve overall product yield.[52] Without loss of generality, assuming that dies are classified into two process bins, SS and FF, the example in Figure 3.17 (where SS-SS, SS-FF and FF-SS respectively indicate SS Tier 0 + SS Tier 1, SS Tier 0 + FF Tier 1 and FF Tier 0 + SS Tier 1) shows that mix-and-match die stacking can offer 75ps timing improvement for a small 28nm FDSOI block as compared to the conventional worst-case analysis.[53] However, in the previous works each of the stacked die is independently designed, that is, there is no holistic "design for eventual stacking" of any of the die.

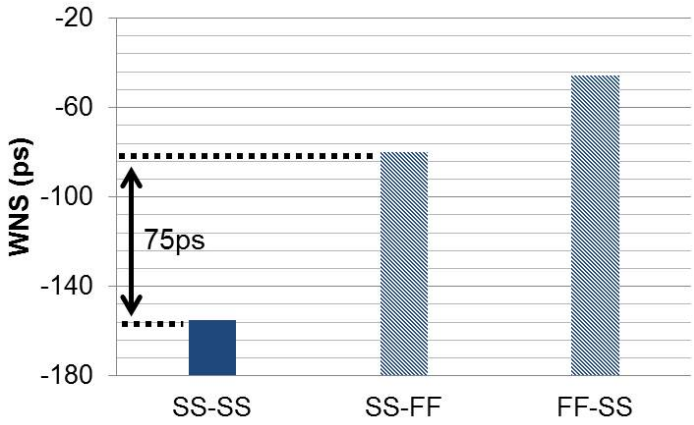


Figure 3.17: Worst negative slack (WNS) of design *AES* [212] in 28FDSOI technology. Clock period = 1.2ns. The *AES* implementation was simply bipartitioned for minimum net cut using MLPart [23][211].

Separately, many works [43][90][97][124][144][181] have suggested approaches for partitioning of logic into multiple die, e.g., to obtain the wirelength (hence, power and delay)

[52]The mix-and-match stacking optimization is also applicable to wafer-to-wafer bonding integration where SS wafers are integrated with FF wafers, and to monolithic 3D integration with adaptive adjustment of the top-tier process according to the bottom-tier process condition. For simplicity, we use "(die) stacking" to refer collectively to these multiple contexts.

[53]In the following discussions and our experiments, we assume that dies are classified into two process bins, SS and FF. However, given matched pairs of process bins based on die-level and/or wafer-level stacking optimization, our approaches can be extended to scenarios with > 2 process bins, e.g., additional combinations can be { SS Tier 0 + TT Tier 1, TT Tier 0 + SS Tier 1, FF Tier 0 + TT Tier 1, TT Tier 0 + FF Tier 1, TT Tier 0 + TT Tier 1 } when we also consider the TT process bin.

savings implied by implementing a 1 $\times$ 1 die area into two stacked 0.7 $\times$ 0.7 dies. However, the signoff criteria used to implement such a multi-die solution must necessarily validate timing correctness for all combinations of process conditions on the multiple die – e.g., the four combinations { SS Tier 0 + SS Tier 1, SS Tier 0 + FF Tier 1, FF Tier 0 + SS Tier 1, FF Tier 0 + FF Tier 1 }.[54] Satisfying this combinatorial number of signoff constraints induces area and power overheads as a result of the sizing and buffering operations needed to close timing.

To our knowledge, no previous work has examined the fundamental issue of *design partitioning and signoff specifically for mix-and-match die stacking*. In particular, if we know *a priori* that, say, SS Tier 0 and SS Tier 1 die will never be stacked together, or that FF Tier 0 and FF Tier 1 die will never be stacked together, this changes our signoff criteria. Even more, this *a priori* knowledge allows us to partition timing-critical paths across tiers to explicitly optimize the design's performance in the regime of mix-and-match stacking. The simple example in Figure 3.18 (where we assume that SS Tier 0 + FF Tier 1 and FF Tier 0 + SS Tier 1 are utilized for die stacking, the partitioning solution indicated by the blue dotted line has the maximum timing slack, while the partitioning solution indicated by the red solid line has the minimum timing slack) illustrates how the partitioning solution can impact design signoff timing in the regime of mix-and-match stacking.

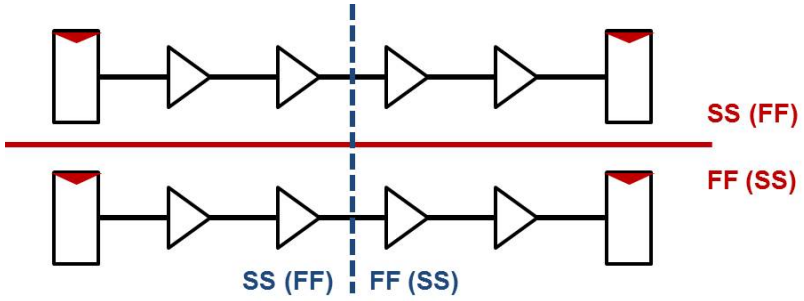


Figure 3.18: Partitioning solutions affect a design's performance in the regime of mix-and-match stacking.

In this section, we propose partitioning methodologies and signoff flows that are aware of mix-and-match die stacking to improve design timing (i.e., to improve worst negative slack (WNS)). However, 3D partitioning for mix-and-match die stacking is nontrivial. First, the optimal cut locations on one timing path might conflict with those on other timing paths. Thus, the partitioning optimization must trade off timing optimizations among timing paths. This can be quite challenging in a design with a large number of potentially critical paths and shared logic

[54] Here, a *tier* refers to one stacked die in a 3DIC. In a two-tier 3DIC, Tier 0 is the bottom tier and Tier 1 is the top tier.

cones among multiple pairs of timing startpoints-endpoints. Further, the partitioning optimization must comprehend the timing impact of *vertical interconnects* or *VIs* (i.e., the vertical electrical connections (vias) between tiers, such as through-silicon vias), and can no longer "freely" partition a timing path into segments. In addition, delay variations across different process conditions can be different for cells of different types (e.g., INV, NAND or NOR), sizes and V_{th} flavors. Last, asymmetric distribution of process bins (e.g., 3σ FF + 2σ SS) as discussed in [101] will also increase the difficulty of the partitioning optimization. Figure 3.19 shows a simple example with different optimal partitioning solutions that respectively minimize (a) delay of path A-C, (b) delay of path B-C, and (c) the worst case over the two paths. Moreover, the optimal partitioning solution changes with increased VI delay impact, as shown in Figure 3.19(d). In Figure 3.19, the red bars are VIs. We further assume the same stage delay ($30ps$ at SS, $10ps$ at FF) for every stage in the two paths, and that the timing analysis is aware of mix-and-match stacking (i.e., { SS Tier 0 + FF Tier 1, FF Tier 0 + SS Tier 1 }) and assumes ideal clock.

Our contributions in this section are as follows.

- We are the first to study design-stage optimization specifically for mix-and-match die stacking.
- We develop partitioning methodologies that are inherently aware of mix-and-match die stacking. Our approaches achieve up to 16% timing improvement as compared to a min-cut based partitioning approach.
- We extend the existing 3DIC implementation flows to incorporate mix-and-match-stacking-aware partitioning and signoff, demonstrating the simplicity of adopting our techniques.

3.3.1 Related Work

We classify related works into two categories: (i) mix-and-match die stacking optimization and (ii) 3D netlist partitioning.

Mix-and-match optimization. Several works propose approaches for mix-and-match die stacking optimization. Ferri et al. [57] propose methodologies to benefit from the flexibility of die-to-die and/or die-to-wafer 3D integration with awareness of the inter-die process variation. Their optimization improves performance and parametric yield of 3DICs with one CPU die and one L2 cache die. Garg et al. [63] formulate mathematical programs to improve the

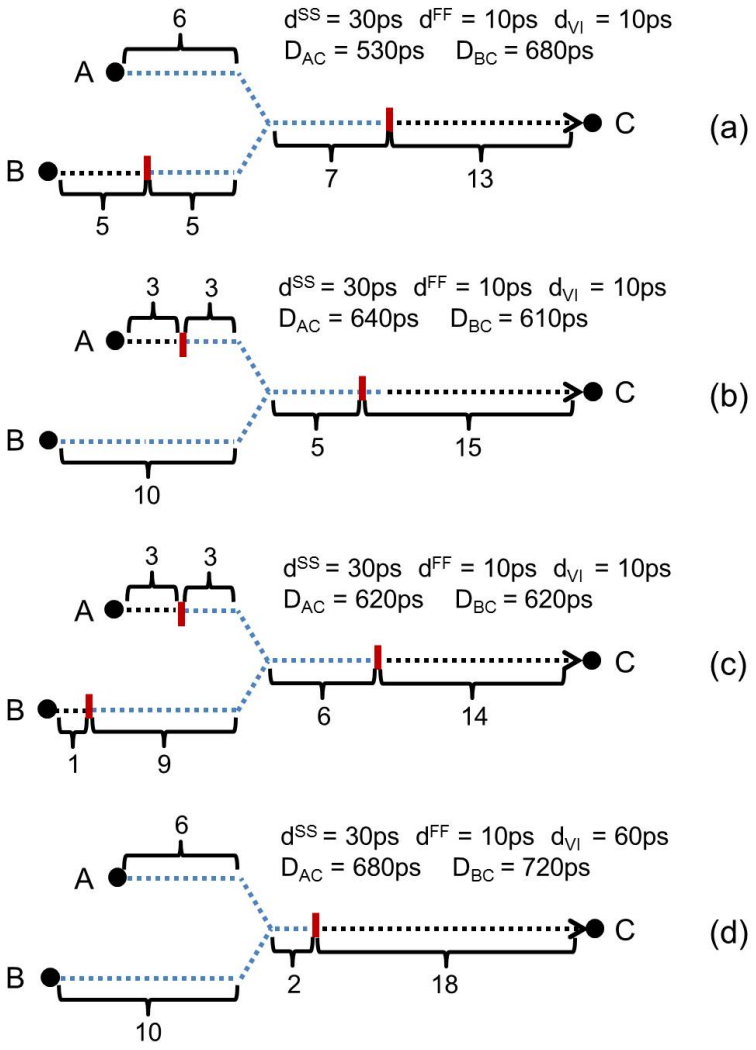


Figure 3.19: Area-balanced partitioning solutions on path A-C (26 stages) and path B-C (30 stages) which respectively minimize (a) delay of path A-C (D_{AC}), (b) delay of path B-C (D_{BC}), (c) worst-case delay over the two paths, and (d) worst-case delay over the two paths with large VI delay impact (d_{VI}).

performance yield of 3DICs via mix-and-match die stacking. Chan et al. [27] propose an integer linear programming-based method as well as a heuristic method to optimize reliability of 3DICs (i.e., to improve the mean time to failure). To avoid the large runtime of thermal simulation, Juan et al. [99] develop a learning-based model for temperature prediction in 3DICs. Based on the model, they perform thermal-aware matching and stacking of dies to improve thermal yield. These optimization approaches operate at die level or wafer level (essentially, post-manufacturing). By contrast, our work addresses design-stage optimization and signoff for mix-and-match die stacking.

3D netlist partitioning. As mentioned above, quite a few works study 3D partitioning. Li et al. [124] use a simulated annealing engine to partition blocks across tiers during the floorplanning stage to minimize wirelength. Several works cast 3D partitioning as a form of standard hypergraph partitioning. Thorolfsson et al. [181] use hMetis [115] to partition the design into balanced halves while minimizing the number of cuts. A multilevel partitioning methodology is proposed in [90], which first applies Hyperedge Coarsening (HEC) techniques to coarsen the netlist, then performs an FM-like K-way partitioning procedure to partition the netlist such that the number of VIs is minimized. An integer linear programming for 3D partitioning is formulated in [97], where the objective is to reduce the number of VIs subject to area balancing constraints. Partitioning methodologies based on an initial 2D implementation solution are also proposed in previous literatures. Cong et al. [43] assign cells to tiers through folding-based transformations of an initial 2D placement solution. Based on a 2D implementation solution with scaled dimension (i.e., 0.7×), Panth et al. [144] perform routability-driven partitioning to minimize the overall routing overflow; this can mitigate routing congestion and help minimize wirelength. Compared to these works, our work is again distinguished by being the first to inherently comprehend mix-and-match die stacking integration. In particular, unlike previous works, our partitioning methods directly maximize the design's timing slack in the mix-and-match regime.

3.3.2 Problem Formulation

We formulate the partitioning problem for mix-and-match die stacking as follows.

Given: post-synthesis netlist, Liberty files according to various process bins, vertical interconnect (VI) parasitics, timing constraints and area balancing criteria,

Perform: 3D partitioning to determine the tier index for each cell, such that the worst timing slack is maximized in the context of mix-and-match die stacking.[55]

In the next section, we describe an ILP-based partitioning methodology which is able to achieve near-optimal solutions. Section 3.3.4 then proposes a heuristic partitioning methodology in which we (i) perform *maximum-cut* partitioning on the subgraph of the sequential graph that is induced by timing-critical pairs of startpoints and endpoints, then (ii) apply a signoff timing-aware FM optimization for further slack improvement.

[55] In this section, we only consider partitioning into two-tier 3DICs. But, our formulation generalizes easily to larger numbers of tiers.

3.3.3 ILP-Based Partitioning Methodology

We now formulate an integer linear program (ILP) to partition the netlist into two tiers such that the worst timing slack, over the corner combinations that can be formed by mix-and-match stacking, is maximized. Table 3.9 summarizes our notations.

Table 3.9: Notations.

Term	Meaning
α_j	process condition (corner), ($1 \leq j \leq J$)
P	set of timing paths
p_k	k^{th} timing path ($p_k \in P$)
C	set of cells
c_i	i^{th} cell ($c_i \in C$)
a_i	area of cell c_i
y_i	binary indicator whether cell c_i is on Tier 0 (y_i = 0) or on Tier 1 (y_i = 1)
$\beta_{i,i'}$ ($\beta_{i',i}$)	binary indicator whether a cut (VI) exists between adjacent cells c_i and $c_{i'}$, where cell c_i is on Tier 0 (Tier 1) while cell $c_{i'}$ is on Tier 1 (Tier 0).
d_i^j	stage delay of cell c_i and its fanout wire at α_j
D_k	maximum delay of path p_k over all pairs of process corners
D_{max}	maximum delay over all paths among all pairs of process corners
d_{VI}	delay impact of VI insertion
θ	area balancing criterion

Minimize D_{max}

Subject to

$$\beta_{i,i'} \geq y_{i'} - y_i \qquad \forall \text{ adjacent cells } c_i, c_{i'} \in C \tag{3.29}$$

$$\beta_{i',i} \geq y_i - y_{i'} \qquad \forall \text{ adjacent cells } c_i, c_{i'} \in C \tag{3.30}$$

$$\beta_{i,i'} + \beta_{i',i} \leq 1 \qquad \forall \text{ adjacent cells } c_i, c_{i'} \in C \tag{3.31}$$

$$\sum_{c_i \in p_k} (d_i^j \cdot (1 - y_i) + d_i^{j'} \cdot y_i) + \sum_{adjacent\ c_i, c_{i'} \in p_k} (\Delta_{i'}^{j,j'} \cdot \beta_{i,i'} + \Delta_{i'}^{j',j} \cdot \beta_{i',i})$$

$$+ \sum_{adjacent\ c_i, c_{i'} \in p_k} (\beta_{i,i'} + \beta_{i',i}) \cdot d_{VI} \leq D_k \quad \forall (\alpha_j, \alpha_{j'}),\ p_k \in P \tag{3.32}$$

$$D_k \leq D_{max} \qquad \forall p_k \in P \tag{3.33}$$

$$\sum_{c_i \in C} a_i \cdot y_i - \sum_{c_i \in C} a_i \cdot (1 - y_i) \leq \theta \cdot \sum_{c_i \in C} a_i \tag{3.34}$$

$$\sum_{c_i \in C} a_i \cdot (1 - y_i) - \sum_{c_i \in C} a_i \cdot y_i \leq \theta \cdot \sum_{c_i \in C} a_i \tag{3.35}$$

Our objective is to minimize the maximum path delay D_{max} over all paths $p_k \in P$, across all relevant pairs of process corners in the context of mix-and-match die stacking. y_i is a binary indicator of cell c_i's tier assignment, with y_i = 0 (resp. y_i = 1) indicating that c_i is on Tier 0 (resp. Tier 1). For any pair of adjacent cells c_i and $c_{i'}$, we use Constraints (3.29) and (3.30) to force either $\beta_{i,i'}$ or $\beta_{i',i}$ to be one when cells c_i and $c_{i'}$ are on different tiers. In other words, $\beta_{i,i'}$ and $\beta_{i',i}$ are indicators of a cut (or VI) such that $\beta_{i',i} = 1$ (resp. $\beta_{i,i;} = 1$) when c_i is on Tier 0 (resp. Tier 1) while $c_{i'}$ is on Tier 1 (resp. Tier 0). Therefore, $\beta_{i,i'}$ and $\beta_{i',i}$ are mutually exclusive.

Constraint (3.32) defines the maximum delay D_k for each path $p_k \in P$ among all pairs of process corners with mix-and-match stacking. The first term on the left-hand side of Constraint (3.32) is the sum of stage delays along path p_k. We extract stage delays at a particular corner α_j based on the timing analysis assuming all cells are at α_j. However, such an assumption can lead to an inaccurate stage delay estimation because cells of different process corners output different slews, which affect the delays of downstream cells. For example, our assumption can be pessimistic for a cell at SS when its driver is at FF. This is because to estimate the stage delay at SS, our timing analysis assumes all cells (including its driver) are at SS, which results in pessimistic input slew estimation. To compensate for such inaccuracy, we pre-calculate

the delta stage delays (that is, the second term) between the case where the driver cell c_i and driven cell $c_{i'}$ are at different process corners (i.e., c_i is at α_j, and $c_{i'}$ is at $\alpha_{j'}$) versus the case where the c_i and $c_{i'}$ are at the same process corner.[56] We denote such delta stage delays as $\Delta_{i'}^{j,j'}$. Incorporating the second term, i.e., the sum of delta stage delays along path p_k, enables us to achieve a more accurate delay estimation.[57] The third term on the left-hand side of Constraint (3.32) accounts for VI delay impact along the path. Note that VI insertion at the output pin of a small-size cell can have quite large delay impact. However, such delay impact will be addressed with sizing/V_{th}-swapping optimization during the P&R (placement and routing) flow. Since no sizing/V_{th}-swapping optimization is involved during the partitioning stage, to avoid pessimism in estimation of VI delay impact, we simply use a constant value to estimate the delay impact of one VI insertion. In Constraint (3.33), we obtain the maximum delay D_{max} over all paths $p_k \in P$. Last, our formulation satisfies area balancing criteria which are indicated by θ in Constraints (3.34) and (3.35). We set θ as 5% in our experiments.

3.3.4 Heuristic Partitioning Methodology

Although the ILP-based methodology can achieve near-optimal partitioning solutions, its runtime can be large. Moreover, it is practically impossible to extract all timing paths for a large design.[58] We therefore propose a timing-aware FM partitioning methodology with better scalability. Our heuristic partitioning methodology contains two optimization stages – (i) the global optimization performs *maximum cut* on the timing-critical sequential graph (i.e., a partial sequential graph which contains only startpoints and endpoints of timing-critical paths) and (ii) the incremental optimization performs timing-aware multi-phase Fiduccia-Mattheyses (FM) optimization to achieve the final partitioning solution. Unlike previous works which minimize the number of cuts [115] or the number of paths passing across different partitions [114], we directly target the timing slack improvement during our partitioning optimization. Our objective is to minimize the maximum path delay (i.e., maximize the worst timing slack) for mix-and-match die stacking. Further, we show that a maximum-cut partitioning is more suitable than the traditional minimum-cut partitioning for 3DICs in the mix-and-match regime. To our knowledge,

[56] Our separate study shows that delay impact caused by cells more than one stage upstream of the current cell is negligible (i.e., $< 2ps$). We therefore only consider the slew change due to current cell's direct fanins.

[57] We note that since the partitioning optimization is performed before placement and routing, the wire delay and accurate wire load information are not available, which might lead to suboptimality in the partitioning solution.

[58] Slight suboptimality of the ILP comes from the estimations of stage delay and delay impact of VI insertions, which are inputs to the ILP. The runtime to extract timing path information and solve the ILP can be even larger if there are more process bins, which makes the ILP-based methodology infeasible. The runtime of the ILP on *AES* (with 11K instances and 254K timing paths) is $>$ 24 hours.

few if any previous works have applied a semidefinite program-based maximum cut optimization [67] to VLSI design.

Maximum-Cut Partitioning on Timing-Critical Sequential Graph

We first study the tradeoff between delay impact of VI insertions versus timing improvement from mix-and-match stacking. Without loss of generality, we assume a die stacking of { SS Tier 0 + FF Tier 1, FF Tier 0 + SS Tier 1 }. We denote the path delay of path p_k at SS (resp. FF) as D_k^{SS} (resp. D_k^{FF}), and the total number of stages along p_k as l_k. Approximating the path delay as a linear function of the stage number and assuming that there are l'_k stages on Tier 0, the corresponding path delay without considering delay impact of VI insertion can be estimated as

$$l'_k \cdot \frac{D_k^{SS}}{l_k} + (l_k - l'_k) \cdot \frac{D_k^{FF}}{l_k} \tag{3.36}$$

$$l'_k \cdot \frac{D_k^{FF}}{l_k} + (l_k - l'_k) \cdot \frac{D_k^{SS}}{l_k} \tag{3.37}$$

where (3.36) assumes the stacking of SS Tier 0 + FF Tier 1, and (3.37) assumes the stacking of FF Tier 0 + SS Tier 1. Maximizing the minimum value between (3.36) and (3.37) corresponds to having (3.36) = (3.37) and $l'_k = l_k/2$. We therefore estimate the timing improvement from mix-and-match stacking over the worst-case analysis (i.e., SS Tier 0 + SS Tier 1) as $(D_k^{SS} - D_k^{FF})/2$. Furthermore, we denote the worst slack of p_k among combinations of process conditions (i.e., { SS Tier 0 + FF Tier 1, FF Tier 0 + SS Tier 1 }) as s_k, and denote the delay increase due to an inserted VI as d_{VI}. Based on the above, we classify timing paths of a design into three categories:

1. Type I: Timing non-critical paths ($s_k \geq s_{th}$);
2. Type II: Timing-critical paths without tolerance of VI insertion ($s_k < s_{th}$ && $\frac{D_k^{SS} - D_k^{FF}}{2} \leq d_{VI} + s_{gb}$);
3. Type III: Timing-critical paths with tolerance of VI insertions ($s_k < s_{th}$ && $\frac{D_k^{SS} - D_k^{FF}}{2} > d_{VI} + s_{gb}$);

Here, s_{th} is the threshold of timing slack to define the timing-critical paths (i.e., s_{th} = 10% of clock period); and s_{gb} is the slack guardband to evaluate tradeoff between delay impact of VI insertions versus timing improvement from mix-and-match stacking.[59] We note that when

[59]The value of s_{th} needs to be empirically determined such that timing-critical paths are optimized. However, a too-large value of s_{th} can result in a large number of VI insertions and large runtime for timing analysis. Slack guardband s_{gb} is a flat timing margin, where the timing improvement from mix-and-match must exceed the VI delay impact by more than s_{gb}.

the delay of a VI insertion is so large that most of the timing-critical paths are Type-II paths, the timing benefits from mix-and-match die stacking will be limited.

Our optimization focuses on timing-critical paths (i.e., Type-II and Type-III paths). Our optimization ensures that startpoint and endpoint of a Type-II path are assigned to the same tier. Further, our optimization maximizes the number of Type-III paths being cut, so as to improve the potential timing benefits from mix-and-match die stacking. The procedure of our optimization is described in Algorithm 8. To construct the sequential graph, each startpoint or endpoint (e.g., register, PI or PO) becomes one vertex, and a directed edge is inserted between two vertices if there exists a (combinational) timing path between the vertices when they are taken as startpoint and endpoint. Note that in this optimization we only consider the maximum-delay path between any startpoint-endpoint pair. We use the algorithm in [67] for our maximum-cut optimization, in which the maximum-cut problem is relaxed to a semidefinite program (SDP). The SDP solution is then randomly rounded to achieve a partitioning solution. We use SDPA [216] as our semidefinite programming solver.

Algorithm 8 Partitioning of the sequential graph.

1: Extract restricted sequential graph G_0 that contains only Type-II and Type-III paths.
2: Collapse vertices connected with Type-II paths (edges) into one vertex to obtain a new graph G_1.
3: Perform maximum cut on G_1.

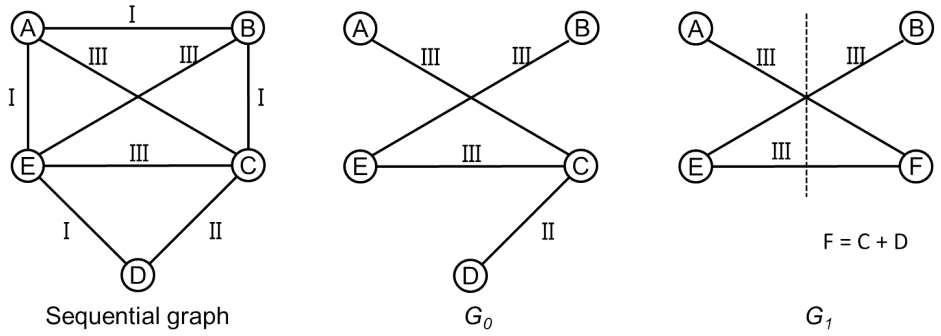


Figure 3.20: Example of maximum-cut partitioning of the sequential graph. Types of paths are shown in edge labels. The dotted line indicates the final maximum-cut solution. We assume the same weight for all edges.

Figure 3.20 illustrates Algorithm 8 with an example consisting of five vertices and eight edges. The figure shows each updated graph, and the dotted line indicates the final maximum-cut solution.

Timing-Aware Multi-Phase FM Partitioning

Based on the maximum-cut partitioning solution of a timing-critical sequential graph, we fix the tier assignments of flip-flops and then perform *timing-aware multi-phase partitioning* to achieve the final partitioning solution. At each phase of our optimization, we perform optimizations in parallel with multiple threads. Optimization in each thread first clusters cells such that the size of the cluster is within a given range (i.e., [N_{lb}, N_{ub}]). Based on the clustered netlist, each thread then performs Fiduccia-Mattheyses (FM) optimization [58] to improve the partitioning solution in terms of the worst timing slack in the context of mix-and-match stacking. We vary the range of cluster sizes across different threads during our optimization. At the end of each phase, we select the partitioning solution with the maximum timing slack as the input to the next phase.

In our FM optimization, the gain function of a cluster u is defined as

$$gain(u) = \frac{\Delta slack(u)}{slack(u) - WNS} \tag{3.38}$$

where $slack(u)$ is the worst slack of cluster u; $\Delta slack(u)$ is the slack change when moving u across tiers; and WNS is the worst negative slack of the entire design.

Clustering cells at each phase before the FM optimization not only reduces the runtime of FM optimization but more importantly also improves the solution quality. Figure 3.21 shows an example in which moving one cell with negative gain can eventually lead to slack improvement after moving its neighbor cells, where we assume that the difference between cell delays at SS and FF is 30ps, delay impact due to VI insertion is 50ps, and all cells along the path (only a segment of five cells is shown) are initially on Tier 0. We also assume that a stacking of SS Tier 0 + FF Tier 1 is applied. In the example, although moving one cell across tiers degrades the slack of the path due to VI insertions, moving its neighbor cells compensates for the delay impact of VI insertions and eventually improves the path timing for mix-and-match stacking. However, during the FM optimization, it is difficult and expensive (in terms of runtime) to "foresee" such slack benefits. In other words, to evaluate the gain function of one cell including its future impact, one must consider a large number of potential moves of its neighbor cells. The number of potential future move sequences can be large if only moving multiple stages of cells can compensate for the delay impact of VI insertions.[60] We therefore cluster cells such that timing improvement from moving a cluster can compensate for the delay impact of VI insertions. Further, since the goal of clustering and partitioning is to balance cell delays across tiers along

[60] We are aware of "lookahead" approaches, such as gain vectors, CLIP/CDIP and LIFO gain buckets, etc. [51][77][120]. However, these are cut-centric and not path-aware, hence inapplicable to our current problem.

each timing path, the desired cluster size highly depends on number of stages along the paths, fanout number at each stage, and netlist topology. Given that the number of stages along the path is limited by timing constraints, along with the maximum fanout constraint, a too-large cluster size might not help to balance delays across tiers along a timing path. We empirically set the cluster size to be no larger than 120 in our experiments.

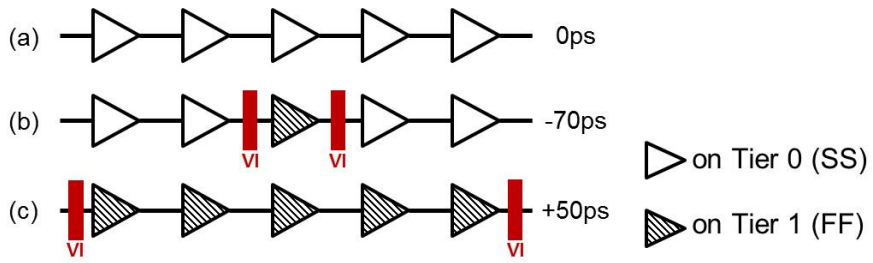


Figure 3.21: Example to optimize a cell with a negative gain value. (a) Initial path with zero slack. (b) Moving one cell to Tier 1 degrades the slack by $70ps$ due to VI insertions. (c) Further optimization on the shown segment improves the slack by $50ps$.

Algorithm 9 shows our clustering procedure. We first sort all cells in increasing order of their slacks (Line 1). We use topological order to break ties. We then select an unclustered cell from the ordered list as the starting point for clustering (Line 2). Based on the selected cell, we evaluate its slack changes due to moves (i.e., tier re-assignment) on its neighbor cells. If slack improves, we add the corresponding neighbor cell into the cluster (i.e., u), and further consider moves on neighbor cells of the new added cell (Lines 7-11, 15). However, when no move with slack improvement is available, we select the neighbor cell corresponding to the move with the minimum slack degradation and add it to the cluster (Lines 17-22, 27-30). The clustering procedure terminates when the cluster size meets the required range (i.e., $[N_{lb},\ N_{ub}]$) or there is no unclustered neighbor cell (Lines 12-14, 24-26).

Note that each cluster contains cells originally belonging to the same tier. The slack of a cluster (i.e., $slack(u)$) is defined as the worst slack of cells within the cluster. Further, the estimation of $slack(\{c, u\})$ comprehends mix-and-match stacking (i.e., worst case over SS Tier 0 + FF Tier 1 and FF Tier 0 + SS Tier 1). Moreover, our timing analysis takes into account the delay impact of VI insertions (Figure 3.22 shows one example). Assuming that the incremental timing analysis is performed in constant time,[61] the runtime complexity of our clustering algorithm is $O(|C|^3)$.

[61] In incremental timing analysis, we propagate slew and update cell delay through interpolation in Liberty lookup tables. Starting from the moved cell, we traverse the timing graph both forwards and backwards until there is no slack change. Given the maximum fanout constraints (e.g., 20) and limited number of stages to which "ripple effects" propagate (e.g., ∼2-3 stages at most), in practice there is a constant bound on the number of cells updated during the incremental timing analysis.

Algorithm 9 Clustering.

$cell\_list \leftarrow$ sort all cells in increasing order of their slacks
for all $c \in cell\_list$ that is not clustered **do**
  $queue.push\_front(c)$; $u \leftarrow \emptyset$ // initialize cluster u
  while $|u| < N_{ub}$ **do**
    $s' \leftarrow -\infty$; $c' \leftarrow \emptyset$; $queue' \leftarrow \emptyset$
    while $|queue| > 0$ **do**
      $c \leftarrow queue.pop\_front()$
      $s_u \leftarrow slack(u)$; $s_c \leftarrow slack(c)$
      move c to a different tier; incremental timing analysis
      if $|u| == 0$ || $slack(u) \geq s_u$ && $slack(c) \geq s_c$ **then**
        $u \leftarrow u \cup \{c\}$
        if $|u| \geq N_{ub}$ **then**
          break
        end if
        $queue.push\_back$(neighbors of c that are not clustered)
      else
        if $Min(slack(c), slack(u)) > s'$ **then**
          $s' \leftarrow min(slack(c), slack(u))$; $c' \leftarrow c$
        end if
        $queue'.push\_back(c)$
        recover c to its original tier; incremental timing analysis
      end if
    end while
    if $|u| \geq N_{ub}$ || $|queue'| == 0$ **then**
      break
    end if
    move c' to a different tier; incremental timing analysis
    $u \leftarrow u \cup \{c'\}$
    $queue.push\_back$(neighbors of c' that is not clustered)
    $queue.push\_back(queue')$; $queue' \leftarrow \emptyset$
  end while
end for

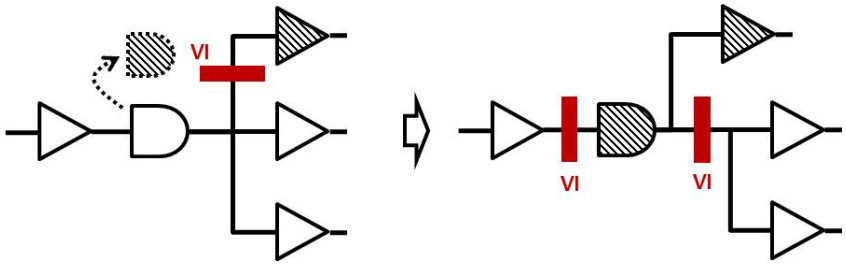


Figure 3.22: Example of VI insertion/removal due to cell movement across tiers. Shaded cells are on Tier 1 and the others are on Tier 0.

In each run of FM optimization, we iteratively select the cluster with the maximum gain value and move it across tiers. We lock the clusters (cells) that have been moved. After each move, we perform incremental timing analysis and update the gain values of the neighboring clusters of which the worst slack is changed. We empirically observe that the slack improvement

at the later stages of an FM run is small (e.g., shown in Figure 3.23, where cluster size ranges are [60, 70], [30, 40] and [15, 20] and each phase contains two runs of FM optimization shown as red and blue curves). Therefore, we terminate each FM iteration when 25% of clusters have been moved. Given that the initial partitioning solution is not area-balanced, in the first FM iteration we terminate the optimization when the area balancing criterion is met. Figure 3.23 shows an example of our FM optimization on design *AES*. The optimization has three phases, where each phase contains two runs of FM optimization. We observe that the worst slack improves from -200ps to -14ps in this example with ∼3000 moves.

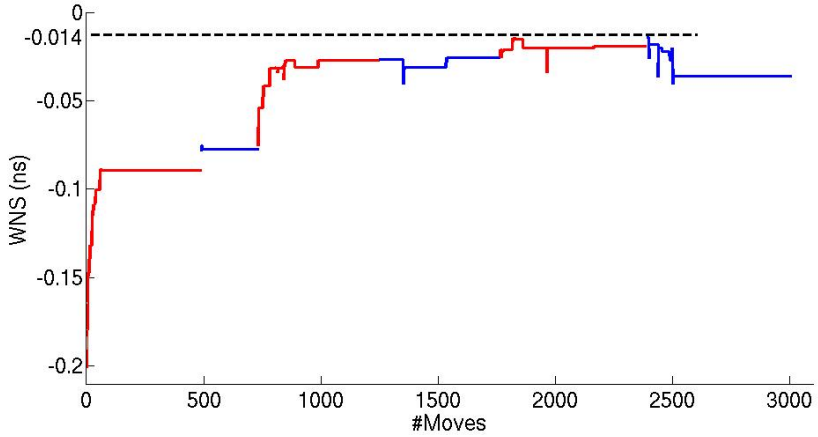


Figure 3.23: An example of our multi-phase FM optimization. Design: *AES*. Technology: 28FDSOI. WNS improves from -200ps to -14ps. Runtime = 565 seconds on a 2.5GHz Intel Xeon server.

3.3.5 Experimental Setup and Results

Experimental Setup

Our partitioning methodologies for mix-and-match stacking are implemented in C++. We use *CPLEX v12.5* [203] as our ILP solver and *SDPA* [216] as our semidefinite programming solver. Our SP&R (synthesis, placement and routing) flow uses *Synopsys Design Compiler vH-2013.03-SP3* [218], *Cadence Encounter Digital Implementation System v12.0* [198], *Synopsys PrimeTime vH-2013.06-SP2* [221] for logic synthesis, P&R, and timing and power analyses, respectively. Similarly to [143], we stitch SPEF files of Tier 0 and Tier 1, with annotated VI parasitics for timing and power analyses.

We use six open-source designs (*DMA*, *USB*, *AES*, *MPEG*, *JPEG*, *VGA*) [212] and an *Arm CORTEX M0* in our experiments. These testcases are generated with foundry 28nm FDSOI 12-track, dual-V_{th} libraries. We use a BEOL stack of six metal layers for routing.

Table 3.10: Summary of testcases.

Design	#Instances	Clock period (ns)
DMA	∼2K	0.6
USB	∼4K	0.8
CORTEX M0	∼9K	1.2
AES	∼11K	1.1
MPEG	∼13K	1.2
JPEG	∼36K	1.4
VGA	∼73K	1.0

We conduct three experiments to evaluate the performance of our partitioning methodologies. (i) We validate the solution quality of our heuristic partitioning optimization by comparing its solutions with those of the ILP-based method. Due to poor scalability of the ILP-based method, we perform experiments on two small testcases (*DMA* and *USB*). (ii) We assess the benefit from our heuristic partitioning method within a brute-force 3DIC implementation flow, which we refer to as GT2012 [100]. (iii) We further assess the benefit from our partitioning method within a state-of-the-art 3DIC implementation flow (Shrunk2D) [143]. In our experiments, we perform three-phase optimization; each phase contains two FM runs. The ranges we use for cluster sizes are [100, 120], [80, 90], [60, 70], [40, 50], [20, 30], [10, 20]. Thus, our optimization uses six threads.

3DIC Implementation Flows

Based on the conventional 2D implementation (P&R) flow, we study the GT2012 3DIC implementation as shown in Algorithm 10.[62] We first partition the netlist into two tiers (Line 1). After the partitioning, we place cells on Tier 0, and determine the VI locations based on that placement (Lines 2-3). With the fixed VI locations, we perform placement optimization on Tier 0 and Tier 1 separately (Line 4). We then insert a VI as the clock port on Tier 1. The clock VI location on Tier 1 is close to the clock port location on Tier 0 to minimize the cross-tier clock skew. We perform clock tree synthesis (CTS) on Tier 0 and Tier 1 separately (Lines 6-7). Last, we perform routing and routing optimization on each tier (Line 9). Note that we perform 3D timing analysis and update timing constraints for each tier after placement and CTS.

We also use the 3DIC implementation flow in [143] to validate our partitioning method. The flow first performs 2D implementation with scaled (i.e., 0.7×) cell sizes and floorplan.

[62] This 3DIC flow is similar to early flows that we have seen used, e.g., at U.S. Department of Energy laboratories.

Algorithm 10 GT2012 3DIC implementation flow.

1: Netlist partitioning (MLPart [211] or our partitioning method);
2: Initial placement on Tier 0;
3: VI insertion based on placement of Tier 0;
4: Placement optimization on Tier 0 and Tier 1;
5: Timing constraint update;
6: VI insertion for clock port on Tier 1;
7: Clock tree synthesis (CTS) on Tier 0 and Tier 1;
8: Timing constraints update;
9: Routing and routing optimization on Tier 0 and Tier 1;

Based on the shrunk 2D implementation, it partitions cells into two tiers. It further modifies the technology files so that BEOL stacks of two tiers (each has six layers) are connected as one (12-layer) BEOL stack and performs routing on both tiers to determine VI locations. Last, it performs routing and routing optimization on each tier separately. In the flow, all the clock cells are forced to be on Tier 0. Following [143], we refer to this flow as the Shrunk2D flow.

To be aware of mix-and-match die stacking, we extend both flows to perform a multi-view optimization after the netlist is partitioned, such that the die stacking of { SS Tier 0 + FF Tier 1, FF Tier 0 + SS Tier 1 } is captured during the P&R optimization. In addition, we assume face-to-face (F2F) die stacking in both flows.[63]

Experimental Results

Calibration of heuristic partitioning. We calibrate our heuristic partitioning method by comparing its solutions to those of the ILP-based method. We perform experiments on designs *DMA* and *USB*. We vary the VI insertion delay impact from $10ps$ to $50ps$. We also assume different combinations of process conditions (i.e., { 3σ SS + 3σ FF, 2σ SS + 3σ FF, 3σ SS + 2σ FF }). Comparison results in Figure 3.24 show that except for one outlier, the timing slack resulted from our heuristic method is always within $30ps$ difference compared to the solution of the ILP-based method, where the ILP-based solution is considered to be very close to the optimal solution. This confirms that our heuristic method is able to comprehend asymmetric distribution of process bins and VI delay impact. The outlier occurs with the setup of large VI delay impact, where the problem becomes more challenging.

[63] To maximize the timing benefit from mix-and-match die stacking, large number of VIs will be inserted. On the other hand, VI insertions will have area impact in a face-to-back stacking-based implementation. We therefore assume F2F stacking. We also note that F2F stacking and monolithic 3D integration are more preferable in the regime of mix-and-match die stacking due to their small VI area impact.

Table 3.11: Validation of our partitioning methodology on GT2012 and Shrunk2D flows.

Design	Flow	WNS (ps)	TNS (ns)	Area (μm^2)	Power (mW)	#Instances	Wirelength (μm)	#VIs	Utilization (bottom / top)
CORTEX M0	GT2012 (orig)	**-178**	-56.735	8451	6.701	8816	116966	304	77% / 69%
	GT2012 (opt)	**-23**	-0.173	8448	6.210	8780	136631	2744	70% / 76%
	Shrunk2D (orig)	**-89**	-11.040	9697	6.499	9855	83462	3715	83% / 86%
	Shrunk2D (opt)	**-13**	-0.080	10106	6.985	9982	93495	4490	86% / 90%
AES	GT2012 (orig)	**-181**	-26.113	8536	10.700	10964	129896	250	74% / 70%
	GT2012 (opt)	**-8**	-0.012	8554	9.351	10947	156716	4417	65% / 79%
	Shrunk2D (orig)	**-4**	0.000	9621	10.600	11302	113209	4787	78% / 81%
	Shrunk2D (opt)	**56**	0.000	9611	10.200	11356	116816	6304	75% / 83%
MPEG	GT2012 (orig)	**-68**	-2.043	18089	13.900	13152	227734	307	69% / 73%
	GT2012 (opt)	**73**	0.000	18125	14.100	13185	321866	4674	74% / 67%
	Shrunk2D (orig)	**20**	0.000	18620	14.800	13275	158386	4741	72% / 74%
	Shrunk2D (opt)	**79**	0.000	18691	15.400	13279	174804	7727	77% / 70%
JPEG	GT2012 (orig)	**-155**	-7.094	44758	32.100	36521	703770	1159	69% / 72%
	GT2012 (opt)	**-52**	-0.462	45094	31.800	36631	1007156	12571	76% / 67%
	Shrunk2D (orig)	**-115**	-1.760	54457	42.900	52824	520123	14075	85% / 88%
	Shrunk2D (opt)	**-82**	-1.210	54637	43.000	52947	562430	20635	88% / 85%
VGA	GT2012 (orig)	**-244**	-6.213	100143	113.300	72682	2201814	1546	76% / 70%
	GT2012 (opt)	**-80**	-0.251	102683	117.200	72731	3667133	15353	70% / 80%
	Shrunk2D (orig)	**-47**	-0.270	104525	90.000	73950	904742	27780	76% / 77%
	Shrunk2D (opt)	**11**	0.000	104008	86.800	74051	929942	35908	79% / 73%

Validation of our method on GT2012 flow. Table 3.11 shows the timing quality, total cell area, power, gate count, wirelength, number of VIs and post-routing utilization of implementations using the GT2012 flow and the GT2012 flow with our heuristic partitioning method. Note that the reported timing and power are the worst cases between SS Tier 0 + FF Tier 1 and FF Tier 0 + SS Tier 1. We observe that our partitioning approach leads to up to 16% timing improvement (i.e., on designs *AES* and *VGA*) compared to the GT2012 flow, which uses conventional min-cut partitioning [23][211], while achieving similar area and power. This is a significant improvement, considering that even 20% improvement in performance per new technology generation is now quite difficult to achieve. The larger wirelength is because of additional wires routed to the increased number of VIs.

Validation of our method on Shrunk2D flow. Table 3.11 shows design metrics of implementations using the original Shrunk2D flow [143] and its extension with our partitioning method. We observe that the extended flow with our partitioning approach achieves up to 7% timing improvement (i.e., on design *CORTEX M0*) with similar area, power and wirelength. Note that to maintain the solution of the 2D implementation in the scaled floorplan, we include

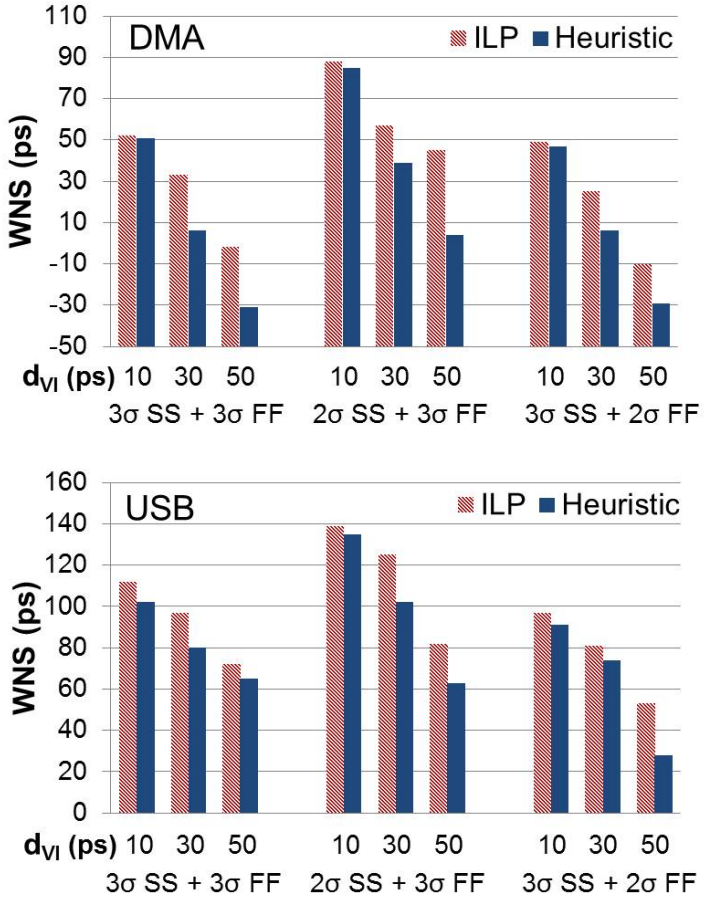


Figure 3.24: Comparison of solution qualities between the ILP-based method (which is near-optimal) and the heuristic method.

additional bin-based area balancing constraints such that we uniformly divide the core area into $N \times N$ bins and set area balancing criteria for each bin during the FM optimization. We use three bin sizes in our optimizations – $20\mu m \times 20\mu m$, $30\mu m \times 30\mu m$ and $50\mu m \times 50\mu m$ – and report the result with the maximum timing slack.

3.3.6 Conclusion

In this section, we study design-stage optimization for mix-and-match die stacking. Our motivating insight is that *a priori* knowledge of mix-and-match 3DIC integration should influence multi-die partitioning optimization and signoff. We propose an ILP-based partitioning methodology and a heuristic partitioning methodology that performs maximum cut on the timing-critical sequential graph followed by an iterative multi-phase FM optimization. We validate our partitioning optimization on two 3DIC implementation flows, each of which we have

extended to be aware of mix-and-match die stacking. Our optimization leads to up to 16% timing improvement, as compared to a flow with min-cut based partitioning solution, when measured by RC extraction and signoff timing at the post-routing stage. Our study also indicates that a gate-level 3D integration has more flexibility and thus larger timing benefits in the mix-and-match regime as compared to a block-level integration. Our ongoing works include (i) integration of design-stage optimization and die- and/or wafer-level optimization for mix-and-match die stacking; (ii) clock tree synthesis for mix-and-match stacking; (iii) including BEOL variation in our optimization; (iv) a new abstraction model for slack improvement with mix-and-match stacking, for faster calculation of gain functions in FM optimization; and (v) more general formulations of die-level mix-and-match optimizations. We will also seek to develop more detailed cost modeling for multi-die integration – e.g., to understand how testability or other considerations might affect our study and/or its conclusions.

3.4 Acknowledgments

Chapter 3 contains reprints of Kwangsoo Han, Andrew B. Kahng and Jiajia Li, "Improved Performance of 3DIC Implementations Through Inherent Awareness of Mix-and-Match Die Stacking", *Proc. Design, Automation and Test in Europe*, 2016; Kwangsoo Han, Andrew B. Kahng, Jongpil Lee, Jiajia Li and Siddhartha Nath, "A Global-Local Optimization Framework for Simultaneous Multi-Mode Multi-Corner Skew Variation Reduction", *Proc. ACM/ESDA/IEEE Design Automation Conference*, 2015; and Tuck-Boon Chan, Kwangsoo Han, Andrew B. Kahng, Jae-Gon Lee and Siddhartha Nath, "OCV-Aware Top-Level Clock Tree Optimization", *Proc. Great Lakes Symposium on Very Large Scale Integration*, 2014. The dissertation author is a main contributor to, and a primary author of, each of these papers.

I would like to thank my coauthors Tuck-Boon Chan, Andrew B. Kahng, Jae-Gon Lee, Jongpil Lee, Jiajia Li and Siddhartha Nath.

Chapter 4

Interconnect-Aware Design Methodologies

As interconnect geometry has shrunk in advanced technology nodes, the rapid increase of interconnect RC leads to not only performance loss from interconnect delay increase, but circuit power and area degradation as well. This chapter presents three distinct techniques for interconnect optimization. First, we propose a new PD-II construction which directly improves both wirelength and pathlength upon the original tree constructed by Prim-Dijkstra (PD) method. The PD-II approach achieves improvement for both objectives, making it a clear win over PD for virtually zero additional runtime cost. PD-II is a spanning tree algorithm (which is useful for seeding global routing); however, since Steiner trees are needed for timing estimation, this section also includes a post-processing algorithm called DAS to convert PD-II trees into balanced Steiner trees. Experimental results demonstrate that this construction outperforms the recent state-of-the-art academic tool, SALT [33], for high-fanout nets, achieving up to 36.46% PL improvement with similar WL on average for 20K nets of size $\geq$ 32 terminals from DAC 2012 contest benchmark designs [189]. Second, we formulate the minimum-cost bounded skew spanning and Steiner tree problems as flow-based integer linear programs, and give the first-ever study of optimal cost-skew tradeoffs. We also assess heuristics (notably, Bounded-Skew DME (BST-DME), Steiner shallow-light tree (SALT), and Prim-Dijkstra (PD)) that are currently available for trading off cost and skew. Experimental results demonstrate that BST-DME has suboptimality $\sim 10\%$ in cost at iso-skew and $\sim 50\%$ in skew at iso-cost. In addition, SALT and PD shows suboptimality in terms of skew by up to $\sim 3\times$. Third, we study the concept of a generalized H-tree – a topologically balanced tree with an arbitrary sequence of branching factors

– and propose a dynamic programming-based method to determine optimal clock power, skew and latency, in the space of generalized H-tree solutions. Our method co-optimizes clock tree topology and buffering along branches according to fitted electrical models. We further propose a balanced K-means clustering and a linear programming-guided buffer placement approach to embed the generalized H-tree with respect to a given sink placement. We validate our solutions in commercial clock tree synthesis tool flows, in a commercial foundry's 28LP technology. The results show up to 30% clock power reduction while achieving similar skew and maximum latency as CTS solutions from recent versions of leading commercial place-and-route tools. Our proposed approach also achieves up to 56% clock power reduction while achieving similar skew and maximum latency as compared to CTS solutions from a state-of-the-art academic tool.

4.1 Prim-Dijkstra Revisited: Achieving Superior Timing-driven Routing Trees

In recent technology nodes, wire capacitance has become a key challenge to design closure, and this problem only worsens with each successive technology node [207]. Today, a digital implementation flow cannot simply use minimum wirelength (WL) trees for routing estimates in placement and optimization, nor can they be used for timing-driven routing of critical nets. Routing an advanced-node design with minimum WL trees leads to untenable source-to-sink distances, yielding high delays for many nets. On the other hand, one cannot afford to use a shortest path tree which achieves optimal source-to-sink pathlength (PL) for each sink, due to the increased WL which degrades dynamic power and worsens routing congestion. For these reasons, timing-driven tree construction that trades off WL and PL becomes a critical technology for modern designs.

The Prim-Dijkstra (*PD*) [9] construction is generally regarded as the best available spanning tree algorithm for achieving this tradeoff and has the additional advantage of simplicity [4].[64] This algorithm has been used for over 20 years to construct high-performance routing trees in leading semiconductor design methodologies and electronic design automation (EDA) tools, as can be seen by related patents assigned to IBM, Synopsys, Cadence and other entities ([86][21][87][62][61][167] [10] [7]). Further, the authors of [11] performed an evaluation that compared *PD* to other spanning tree constructions such as BRBC [40], KRY [119], etc. in 2006;

[64] For global routing, spanning trees are often preferred to Steiner trees since global routing commonly decomposes multi-fanout nets into two-pin nets. A spanning tree provides the router with an obvious decomposition. However, Steiner trees are not well-suited for this because the Steiner points become unnecessary constraints that restrict the freedom of the router to resolve congestion.

they concluded that *PD* obtained the best tradeoff between WL and PL. That paper [11] argues that the *PD* wirelength cost is minimal enough to be practically free. However, this claim is now suspect because today's designs are significantly more power-sensitive than a decade ago: now, a 1% reduction in power is viewed as a big win for today's design teams performing physical implementation. Consequently, even a small WL saving with similar timing can have a high impact on value. A deeper discussion of prior art is given in Section 4.1.1.

The *PD* construction balances between WL and source-to-sink PL by blending the Prim and Dijkstra spanning tree constructions [148][46] via a weighting factor α. When $\alpha = 0$, *PD* is identical to Prim's algorithm [148] and constructs a minimum spanning tree (MST). As α increases, *PD* constructs a tree with higher WL but better PL; when $\alpha = 1$, *PD* is identical to Dijkstra's algorithm [46] and constructs a shortest-path tree (SPT). *PD* begins with a tree consisting just the source node, then iteratively adds the edge e_{ij} that minimizes $d_{ij} + \alpha \cdot l_i$, where node v_i is in the current tree and v_j is not in the current tree, d_{ij} is the distance between nodes v_i and v_j, and l_i is the PL from the source to v_i in the current tree.

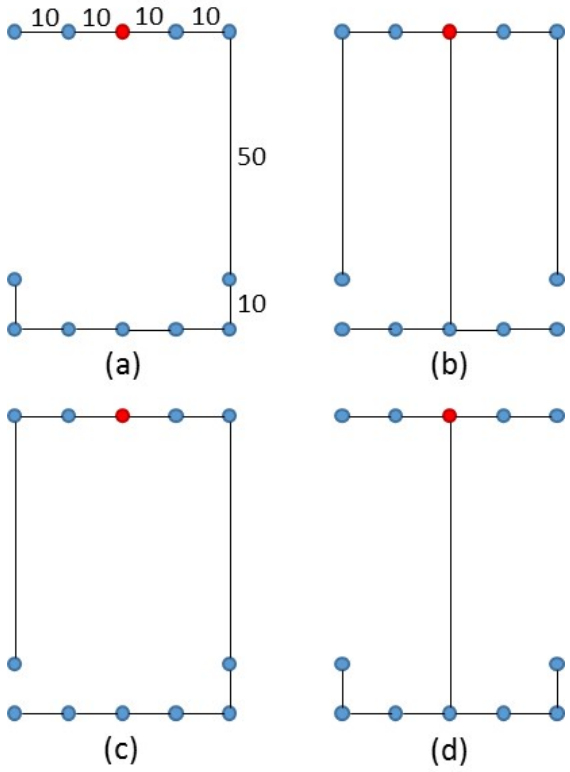


Figure 4.1: An example instance showing suboptimality of *PD*. The red node is the source. (a) shows the MST obtained when $\alpha = 0.2$, (b) shows the SPT obtained when $\alpha = 0.8$, and (c) shows the solution when $\alpha = 0.4$. The tradeoff in (c) is clearly suboptimal in both WL and PL, as compared to (d).

One problem with the *PD* algorithm is that it greedily adds edges, which becomes problematic with higher fanout trees. Once an edge is added, it is never removed from the final solution, making it impossible for *PD* to recover from a potentially poor choice. This can lead to trees that are suboptimal in both WL and maximum PL. Figure 4.1 shows such an example. When α is small (0.2), *PD* obtains the MST solution (a) with $WL = 150$ and $PL = 130$. When α is large (0.8), *PD* obtains the SPT solution (b) with $WL = 240$ and $PL = 80$. However, when $\alpha = 0.4$, *PD* obtains the solution (c) with suboptimal values of both WL and PL ($WL = 190$ and $PL = 120$). This solution (c) is inferior for both objectives than the solution (d) with $WL = 160$ and $PL = 90$. Thus, $\alpha = 0.4$ generates a poor solution for both WL and PL.

This section makes the following contributions:

- To fix the shortcomings in *PD*, one needs to directly optimize PL in the tree construction, which requires a new problem formulation. We propose incorporating total *detour cost*, the amount of suboptimal PL for each node, into the tradeoff. The correct formulation of the objective is paramount since it drives any optimization which follows. This section seeks to optimize the detour cost to all sinks instead of just the worst one, as proposed in prior works [33].

- Next, a new algorithm, which we call *PD-II*, is proposed. The idea is to recover the tree, that has any edges poorly chosen by *PD*, using an iterative improvement method according to the proposed objective function.

- Since Steiner trees are most commonly useful for timing prediction and physical synthesis, an algorithm for converting balanced spanning trees into balanced Steiner trees is proposed. The resulting *Detour-Aware Steinerization* (DAS) algorithm optimizes both WL and detour cost to achieve a tree with similar properties to those obtained by the *PD-II* spanning tree algorithm.

- Finally, three sets of experiments are presented. The first shows that *PD-II* is able to meaningfully shift the Pareto curve obtained by the *PD* algorithm, obtaining up to 18% improvement in PL for the same WL. The second experiment demonstrates the value of the *DAS* algorithm versus more standard Steinerization methods. The third experiment shows that the proposed Steiner construction outperforms those of SALT [33] for medium- and high-fanout nets, a recent state-of-the-art academic tool, achieving up to 36.48% PL improvement for similar WL.

The remainder of this section is organized as follows. Section 4.1.1 briefly reviews related works in the areas of spanning and Steiner tree constructions. Section 4.1.2 presents the proposed problem formulation that incorporates both WL and detour cost. Section 4.1.3 presents the *PD-II* heuristic for spanning tree optimization, and Section 4.1.4 presents the *DAS* heuristic for Steiner tree optimization. Section 4.1.5 reports our experimental results, and Section 4.1.6 concludes the section.

4.1.1 Related Work

There is a rich history on spanning tree and Steiner tree constructions. Many focus on minimizing WL or minimizing the longest PL. (Our present work studies constructions that consider both metrics.)

Spanning tree constructions. As discussed previously, the Prim and Dijkstra constructions achieve optimal WL and PL, respectively. Spanning tree algorithms that optimize both are called *shallow-light* constructions [40][117]; they seek to optimize WL and PL simultaneously to within constant factors of optimal. Shallow-light constructions have in many ways been a "holy grail" in VLSI CAD literature for over 25 years. The *PD* algorithm is "shallow-light in practice", but no such formal property has ever been established [9]. Cong et al. [41] give the Bounded Prim (BPRIM) extension of Prim's MST algorithm [148], which produces trees with low average WL and bounded PLs, but possibly unbounded WL. The BRBC algorithm of Cong et al. [40] produces a tree that has WL no greater than $1 + 2/\epsilon$ times that of an MST, and radius no greater than $1 + \epsilon$ times that of an SPT. Khuller et al. [117] contemporaneously develop a method similar to BRBC.

Minimum WL heuristic Steiner tree constructions. Several works describe heuristic algorithms for Steiner tree constructions with minimized WL. Kahng and Robins [108] give the iterated 1-Steiner (I1-S) heuristic which greedily constructs a Steiner tree through iterative Steiner point insertion, resulting in trees with close to optimal WL. Ho et al. [89] propose an algorithm (HVW) to optimally edge-overlap *separable* MSTs to obtain Steiner trees, while Borah et al. [20] present a greedy heuristic (BOI) to convert spanning trees to RSMTs with performance similar to the I1-S heuristic. Chu and Wong [38] propose FLUTE which uses pre-computed look-up tables for Steiner construction to find solutions more efficiently than the prior art.

Rectilinear Steiner arborescence (RSA) constructions. The NP-complete [173] *rectilinear Steiner arborescence* (RSA) problem seeks to find a minimum-WL tree in the Manhattan plane that achieves optimal PL for every sink. Rao et al. [158] present the first heuristic for the

RSA problem. Cong et al. [42] address the construction of RSAs with the A-tree algorithm, while Kahng and Robins [109] give a simple adaptation of their Iterated 1-Steiner algorithm to the RSA problem.

Steiner constructions that optimize WL and PL. Recently, Scheifele [169] has proposed a method to construct Steiner trees for which Elmore delays are bounded. Given an RMST solution (i.e., FLUTE), [169] iteratively finds the vertex that breaks its ϵ-based metric, and reroutes the vertex to the source via a shortest path, which indirectly balances between RMST and RSA. On the other hand, Elkin and Solomon [54] propose a more direct shallow-light Steiner tree construction method (ES). The main idea is to identify breakpoints and reconnect those breakpoints to the root directly by a Steiner SPT so that there is no detour from the root to the breakpoints. The authors of [54] build a Hamiltonian path and check the accumulated distance along the Hamiltonian path to find proper breakpoints, such that the final Steiner tree meets the given shallowness and lightness criteria. Recently, Chen et al. [33] present SALT, which further improves the ES method [54]. The key contributions are (i) tighter criteria to identify breakpoints, and (ii) using an MST instead of a Hamiltonian path. With some post-processing such as L-shape flipping, the method shows superior tradeoffs between pathlength and wirelength compared to any state-of-the-art spanning/Steiner tree construction methods. Comparisons to the method of [33] are included in Section 4.1.5 below.

4.1.2 Problem Formulation

A *signal net* $V = \{v_0, v_1, ..., v_{n-1}\}$ is a set of n terminals, with v_0 as the *source* and the remaining terminals as *sinks*. We define the underlying routing graph to be a connected weighted graph $G = (V, E)$, where each edge $e_{ij} \in E$ has a cost d_{ij}. We are concerned with the case where G is a complete graph with each e_{ij} having cost equal to the Manhattan distance d_{ij}. A *routing tree* $T = (V, E')$ is a spanning subgraph of G with $|E'| = n - 1$.[65] Given a routing tree T, the cost of the unique $v_0 - v_i$ path in T is l_i, the *radius* of T is $r(T) = max_{1 \le i \le n-1} l_i$, and the *wirelength* (WL) of T is $W_T = \sum_{e_{ij} \in T} d_{ij}$. All notations used in our work are listed in Table 4.1.

Initially, the tree consists only of v_0. The *PD* algorithm iteratively adds edge e_{ij} and sink v_i to T, where v_i and v_j are chosen to minimize

[65] Our use of G and T pertains to the spanning tree context. In the rectilinear Steiner tree context, the underlying routing graph would be the Hanan grid [83], and a Steiner routing tree would be a spanning tree over $\{V \cup S\}$, where S is a set of Steiner points taken from the Hanan grid. For simplicity, as long as meanings are obvious, we will use terms from the spanning tree context in the Steiner tree context as well.

Table 4.1: Notations.

Notation	Meaning
V	signal net, $V = \{v_0, v_1, ... v_{n-1}\}$ having $n-1$ sinks
G	routing graph in the spanning tree context
T	routing tree, which is a spanning subgraph of G
v_0	source node of the signal net V, which is the root of T
e_{ij}	edge from node v_i to v_j
$par(v_i)$	parent node of v_i
l_j	cost of the unique v_0 to v_j path in a tree, $v_0, v_j \in T$
d_{ij}	cost of the edge e_{ij}
m_{ij}	Manhattan distance from node v_i to v_j
W_T	total wirelength of a tree
Q_i	detour cost of node v_i, $Q_i = l_i - m_{i0}$
Q_T	detour cost of a tree, $= \sum_{n-1}(Q_i)$
C	weighted cost of a tree, $= \alpha \cdot Q_T + (1-\alpha) \cdot W_T$
$\Delta C_{e,e'}$	the change in the weighted cost that results from removing edge e and adding e', used in *PD-II*
α	weighting factor used in *PD* and *PD-II*
D	flipping distance used in *PD-II*
P_T	sum of pathlengths of a tree, $= \sum_{n-1}(l_j)$

$$(\alpha \cdot l_j) + d_{ij} \; s.t. \; v_j \in T, \; v_i \in V - T \tag{4.1}$$

The *PD* algorithm can result in trees with either large WL or PL, as shown in Figure 4.1. To alleviate this issue, conventional *shallow-light* tree constructions [40][41][33] focus on bounding the *shallowness* and *lightness* to optimize the tree cost. Lightness η means that the WL of a tree is at most η times of the MST WL. A tree has shallowness ζ if PL to each sink in the tree is at most ζ times the source-to-sink Manhattan distance (MD). However, shallowness alone does not adequately represent the quality of a routing tree. Figure 4.2 shows two examples that have the same shallowness and lightness. It is clear that Figure 4.2(b) is preferable to Figure 4.2(a) since the left sinks have shorter PLs, but shallowness does not capture the difference.

With the above in mind, we define a new *detour cost* metric as follows. Detour cost Q_i of a sink v_i is the difference between PL from v_0 to v_i in T and the Manhattan distance from v_0

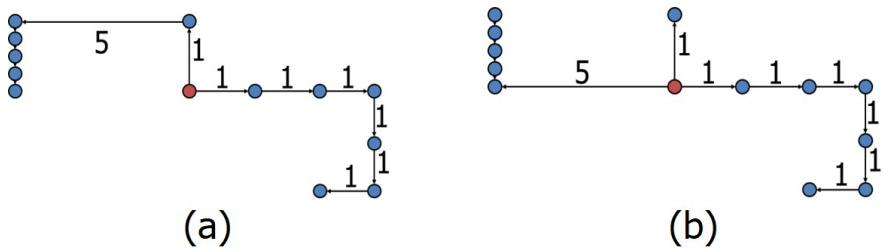


Figure 4.2: Two routing trees that have the same lightness and shallowness.

to v_i. The detour cost of the tree T, denoted by Q_T, is the sum of the detour cost values of all the sinks in the tree, i.e., $Q_T = \sum_{1 \leq i \leq n-1} Q_i$. Since *PD* iteratively adds edges and nodes to the growing tree, if a sink v_j close to the source incurs high detour, then all downstream sinks (descendants of v_j) will also have high detour and hence long PL. We therefore propose the following formulation to capture the problem of simultaneously reducing WL and detour cost of a spanning tree:

Simultaneous WL and detour cost reduction (SWDCR) problem. Given a spanning tree $T = (V, E)$, minimize the weighted sum of WL and detour cost of the tree.

$$\text{Minimize } \alpha \cdot \sum Q_i + (1 - \alpha) \cdot W_T \tag{4.2}$$

where $0 \leq \alpha \leq 1$. We present a heuristic algorithm *PD-II* in Section 4.1.3 for tackling the SWDCR problem.

Once the spanning tree construction is converted into a Steiner tree, there is a change in the tree topology. We propose and address the following formulation to further optimize the detour cost of a Steiner tree:

Detour cost reduction in Steiner trees (DCRST) problem. Given a Steiner tree, minimize the tree detour cost.

$$\begin{aligned} &\text{Minimize } Q_T \\ &\text{s.t. } W_{T,new} \leq W_{T,init} \\ &\qquad Q_{T,init} \geq Q_{T,new} \end{aligned} \tag{4.3}$$

To address the DCRST problem, we present our algorithm *DAS* below in Section 4.1.4.

4.1.3 The PD-II Spanning Tree Construction

This section presents the *PD-II* algorithm that performs iterative edge-swapping which simultaneously improves the detour cost and WL. The key idea of the *PD-II* algorithm is to start

with a spanning tree and swap edges to improve the tradeoff between detour cost and WL. The algorithm can take any spanning tree as input, but it makes sense to start with the *PD* solution since it should already be relatively strong for both objectives. We note that while *PD* can be quite slow for higher-fanout nets, it can be sped up significantly by using a sparsified nearest-neighbor graph instead of the complete graph.

We initially populate the *neighbors* of each node using the following method. We say that v_i is a *neighbor* of v_j if the smallest bounding box containing v_i and v_j contains no other nodes. The worst-case number of neighbor nodes for each node is $\Theta(n)$. For example, every red point in Figure 4.3 is a neighbor of every green point, and vice versa. However, Naamad et al. [138] show that the expected number of maximal empty boxes amidst n random points in a plane is bounded above by $O(n\ log\ n)$, so it is reasonable to expect the average number of neighbors per node to be $O(log\ n)$.

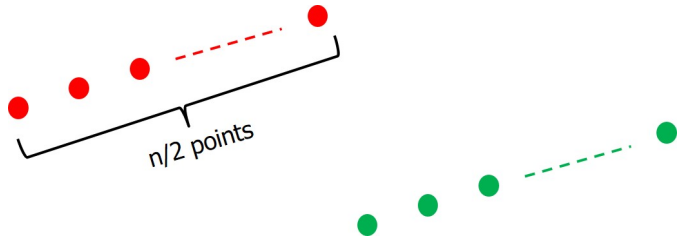


Figure 4.3: Example showing $\Theta(n^2)$ asymptotic worst-case complexity of the number of *neighbor* relationships. Each green node is a neighbor to each red node.

Analysis of random placements of net sinks show this to be true. The number of neighbors for 100K random point sets of size 16, 32, and 64 yields an average number of neighbors per node of 6.3, 8.7 and 11.3, respectively. Real placements should generally have even fewer neighbors, since cells tend to align horizontally or vertically. For the testcases described in Section 4.1.5, the average number of neighbors is 2.58, 4.27, 6.15 and 8.24 for small, medium, large and huge nets, respectively. Hence, in practice, runtime complexity of iterating through the neighbors of a node has logarithmic complexity.

An $O(n\ log\ n)$ runtime complexity can be obtained for *PD* using a binary heap implementation and an adaptation of Scheffer's MST code [168][69]. Since *PD* solutions are generally good, though sometimes suboptimal, it makes sense to post-process the *PD* solution to obtain a better one. The key technique for *PD-II* is *edge flipping*, whereby one edge is removed from the original tree and replaced with a new edge. Figure 4.4(a) shows an example tree, represented as a DAG, representing a topological ordering starting at the source. Figure 4.4(b) shows an example transform in which one edge is removed and replaced with a new red edge, thereby obtaining

a different tree. Note that one of the directed edges in the new (rooted) tree is reversed from its previous orientation in order to maintain a well-formed rooted tree. This approach recalls the iterative improvement operation used in BOI [20], but the application of *flipping* is more restricted to focus on WL vs. PL improvements.

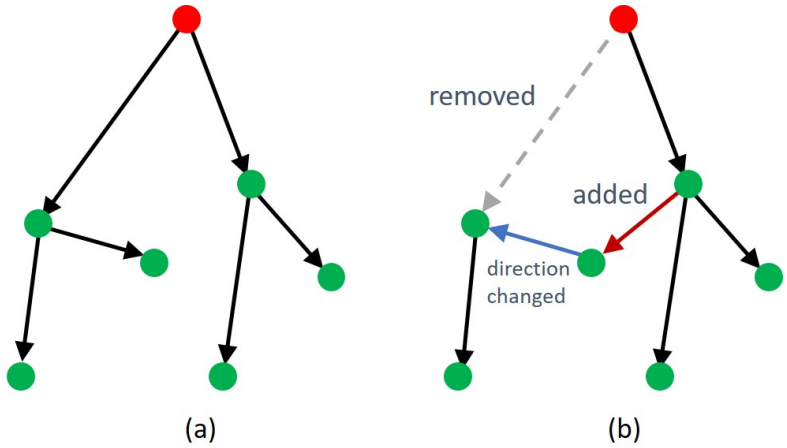


Figure 4.4: Illustration of *PD-II* edge *flipping*.

For each edge pair, we define the *flip cost* as the cost associated with edge flipping, i.e., the cost of removing edge e_{ij} and adding edge $e_{i'j'}$. *Flip cost* $\Delta C_{e,e'} = \alpha \cdot (Q_{T_{i'j'}} - Q_{T_{ij}}) + (1-\alpha) \cdot (d_{i'j'} - d_{ij})$, where α is a weighting factor;[66] $Q_{T_{ij}}$ and $Q_{T_{i'j'}}$ are the detour costs of the trees before and after edge flipping, respectively; and d_{ij} and $d_{i'j'}$ are the lengths of edges being removed and inserted, respectively.

Pseudocode for *PD-II*, Algorithm 11 is given below. Essentially, *PD-II* takes an input tree and searches for edge flips that improve flip cost.[67] If the flip cost improves, the swap is taken. Considering all pairs of possible swaps could be expensive, so we define the *flipping distance* D to be equal to the number of edges in the DAG that require a change in direction to preserve topological ordering, i.e., rooted orientation. For the swap in Figure 4.4, $D = 1$. In practice, using $D > 1$ has little benefit (but more runtime) compared to $D = 1$, so we use $D = 1$ for all experiments.

Line 3 of Algorithm 11 initializes the best flip cost to zero. Line 5 computes the set of candidate edges E_e that can be flipped with edge e, as restricted by the flipping distance D.

[66] The parameter α can be determined by the timing constraints. If a net is critical, a higher value of α can be used to achieve lower delays, but if arcs through the net have positive slacks, α can be small to save wirelength. Hence, α allows topology optimization and can be chosen to best satisfy the design specifications on a per-net basis.

[67] Flipping cannot be added into the original PD cost function since the flip cost objective cannot be correctly computed until an entire tree is constructed. Hence, we propose PD-II as a post-processing algorithm which improves a given spanning tree.

Algorithm 11 Algorithm *PD-II*.

Input: Spanning tree $T_{in} = (V, E_{in})$, with $E_{in} \subseteq E$
Output: Spanning tree $T_{out} = (V, E_{out})$, with $E_{out} \subseteq E$

1: Initialize $T_{out} \leftarrow T_{in}$
2: **repeat**
3: Initialize largest detour cost reduction, $\Delta C_{best} \leftarrow 0$
4: **for all** $e \in E_{out}$ **do**
5: $E_e \leftarrow candidateEdges(e,\ D)$
6: **for all** $e' \in E_e$ **do**
7: $\Delta C_{e,e'} \leftarrow flipCost(e, e')$
8: **if** $\Delta C_{e,e'} < \Delta C_{best}$ **then**
9: $\Delta C_{best} \leftarrow C_{e,e'}$
10: $e_{best} \leftarrow e$; $e'_{best} \leftarrow e'$
11: **end if**
12: **end for**
13: **end for**
14: **if** $\Delta C_{best} < 0$ **then**
15: Remove e_{best}, insert e'_{best} and change direction of associate edges
16: **end if**
17: **until** $\Delta C_{best} == 0$

For each candidate edge $e' \in E_e$, we calculate the flip cost for the edge pair (e, e') and find the edge pair (e_{best},e'_{best}) with lowest flip cost in Lines 6-12. These edges are swapped if the lowest flip cost is less than zero (Lines 14-16). The algorithm continues until no more flip-cost improvement is obtained (Line 17).

The number of candidates for edge flipping can be very large when D is unbounded. The worst-case number of edges is $(n/2)^2$, giving Algorithm *PD-II* a worst-case time complexity of $O(n^3)$, where n is the number of sinks. However, with the distance restriction, the complexity reduces to $O(D \cdot n^2)$, and in practice it converges rapidly. To show this, we take two large blocks from an industrial design and run a production Steiner package on an Intel Xeon $2.7GHz$ machine (CPU E5-2680), using RHEL5. The first design has 1.9 million datapath nets, and the total runtime for the Steiner package which uses *PD* for its spanning tree construction requires 59.3 seconds. Adding *PD-II* to the Steiner package increases the runtime to 62.7 seconds, for a net penalty of 3.4 seconds. The second design with 4.0M datapath nets requires 124.0 seconds for running the default Steiner package. Adding *PD-II* to the Steiner package increases the runtime from 124.0 seconds to 125.8 seconds, for a net penalty of 1.8 seconds. Consequently, the runtime cost of using *PD-II* is negligible, averaging less than one additional second of runtime per million nets.

4.1.4 The Detour-Aware Steinerization Algorithm (*DAS*)

For global routing, spanning tree constructions such as *PD-II* are sometimes preferred to Steiner trees since global routing commonly decomposes multi-pin nets into two-pin nets. However, for timing estimation, congestion prediction, or general physical synthesis optimization, a Steiner tree is required since spanning trees will have too much WL. The previous spanning tree formulation can easily be extended to Steiner trees; the definitions of WL and PL do not change. However, since finding the minimum wirelength Steiner is NP-complete, FLUTE WL is used as a proxy for minimum Steiner tree cost.

To transform a spanning tree into a Steiner tree, the linear-time algorithm of [89] is invoked. It maximizes edge-overlaps in the spanning tree by creating a Steiner node. We call the algorithm *HVW* after the algorithm's creators: Ho, Vijayan, and Wong. *HVW* traverses the tree from the leafs and iteratively maximize overlaps with the currently visited edge and its immediate children edges. However, this basic construction can be inefficient both in terms of WL and PL. Hence a new Steinerization algorithm, called *DAS* for Detour-Aware Steinerization is proposed below.

DAS has two phases of optimization (Algorithm 12). The first phase seeks to reduce WL while minimizing the detour cost penalty (Lines 1-14). This phase does a bottom-up tree traversal and makes edge swaps which reduce WL. For each edge e_{ji} in the Steiner tree, the edge e_{ji} is removed from the tree and replaced with e_{ki} where v_k is a nearest neighbor of v_i if the WL improves and the PL is not overly degraded. (i.e., $p_i \leq 0.5 \cdot p_T^{max}$).

After the first phase, since PL (or detour cost) is not targeted, there still may be room to improve for that dimension. Hence, a second phase (Lines 15-29) seeks to optimize detour cost Q_T without degrading WL. This second phase performs a *top-down* tree traversal to minimize Q_T. This is because the detour cost Q_i to a node v_i affects not only the PL to the node, but also the PL to the downstream nodes of v_i. Thus, more opportunity for large Q_T reductions exists in the edges near the source v_0. For each edge e_{ji} in the Steiner tree, the edge e_{ji} is removed and replaced with e_{ki}, where v_k is the possible parent among the nearest neighbors of v_i, to reduce Q_T without degrading WL. This process is repeated for all the nodes in the tree with non-zero detour cost.

Algorithm *DAS* has a worst-case time complexity of $O(n^2)$. However, with the sparsified nearest neighbor graph implementation described in Section 4.1.3, *DAS* runs much faster than $O(n^2)$ and is closer to $O(n\ log\ n)$ in practice. For 100K nets, *DAS* runs in 0.86 seconds for 16-terminal nets, 1.71 seconds for 32-terminal nets and 4.83 seconds for 64-terminal nets.

Algorithm 12 The Detour-Aware Steinerization Algorithm (*DAS*).

Input: Steiner tree $T_{St,in}$
Output: Improved Steiner tree $T_{St,out}$

//First phase: wire recovery at the cost of small additional PL
$p_T^{max} \leftarrow$ maximum PL of the Steiner tree
Do Breadth-First Search (BFS) from the leaf node
for all v_i **do**
  $v_j \leftarrow par(v_i)$; $d_{ji} \leftarrow$ edge length to v_i;
  $o_{ji} \leftarrow$ overlap length with other edges to v_i
  $\Delta d_{ji} \leftarrow d_{ji} - o_{ji}$
  for all v_k in {all *neighbors* of v_i} **do**
    $\Delta d_{ki} \leftarrow d_{ki} - o_{ji}$; $p_i \leftarrow$ PL to node v_i
    if ($\Delta d_{ki} < \Delta d_{ji}$ && ($p_i \leq 0.5 \cdot p_T^{max}$) **then**
      Disconnect v_i to v_j and reconnect v_i to v_k
    end if
  end for
end for
//Second phase: detour cost reduction with bounded WL
$W_{T,init} \leftarrow$ Init. Steiner tree WL; $Q_{T,init} \leftarrow$ Init. Steiner tree detour cost
Do Breadth-First Search (BFS) from the source node
for all v_i **do**
  $v_j \leftarrow par(v_i)$; $d_{ji} \leftarrow$ Initial edge length to v_i
  for all v_k in {all *neighbors* of v_i} **do**
    $e_{ki} \leftarrow$ Edge from v_k to v_i; $d_{ki} \leftarrow$ Edge length from v_k to v_i
    $W_{T,new} \leftarrow W_{T,init} + d_{ki} - d_{ji}$
    $Q_{T,new} \leftarrow$ detour cost tree with edge e_{ki}
    if ($W_{T,new} \leq W_{T,init}$) && ($Q_{T,new} < Q_{T,init}$) **then**
      Disconnect v_i to v_j and reconnect v_i to v_k
      $W_{T,init} \leftarrow W_{T,new}$; $Q_{T,init} \leftarrow Q_{T,new}$
    end if
  end for
end for

4.1.5 Experimental Setup and Results

Experimental Setup

The algorithms described above are implemented in C++. The following experiments are performed on a 2.7 GHz Intel Xeon server with 8 threads. Testcases are generated from the DAC 2012 contest benchmarks [189], with pin locations for each net are extracted from ePlace placement solutions [132]. Since finding a solution with optimal WL and PL is trivial for two- and three-pin nets, our experiments focus on nets with fanout larger than two. The roughly 749K total nets are divided into four groups (small, medium, large, huge) by their terminal count, as shown in Table 4.2.

While our algorithms optimize Q_T, Q_T itself does not adequately capture the quality of the tree. Instead, results are reported based on two normalized metrics, W_{Tnorm} (normalized

Table 4.2: Net statistics for superblue benchmark designs.

	Small	**Medium**	**Large**	**Huge**
$\|V\|$	$4-7$	$8-15$	$16-31$	$32+$
#Nets	533029	128463	46486	20853

WL) and P_{Tnorm} (normalized PL). W_{Tnorm} is defined as the ratio of the tree WL to the MST WL for spanning trees. P_{Tnorm} is defined as the ratio of sum of PLs of each node in the tree to the sum of Manhattan distances from source to each node. The optimal value any tree could have for either metric is one, which makes the corresponding Pareto curve more intuitive.

Experiment I - Spanning Tree Results

In the following results, *PD* and *PD-II* refer respectively to the *spanning trees* constructed using the *PD* and *PD-II* algorithms. Figure 4.5 shows normalized WL and PL tradeoff curves for *PD* and *PD-II*, for the 46486 large nets. Each point in the curves represents the average (W_{Tnorm}, P_{Tnorm}) over all the nets for a particular value of α. We sweep α from 0.05 to 0.95, in steps of 0.05, to obtain both the *PD* and *PD-II* curves. We observe that the blue *PD-II* Pareto curve is clearly better than the red *PD* curve.

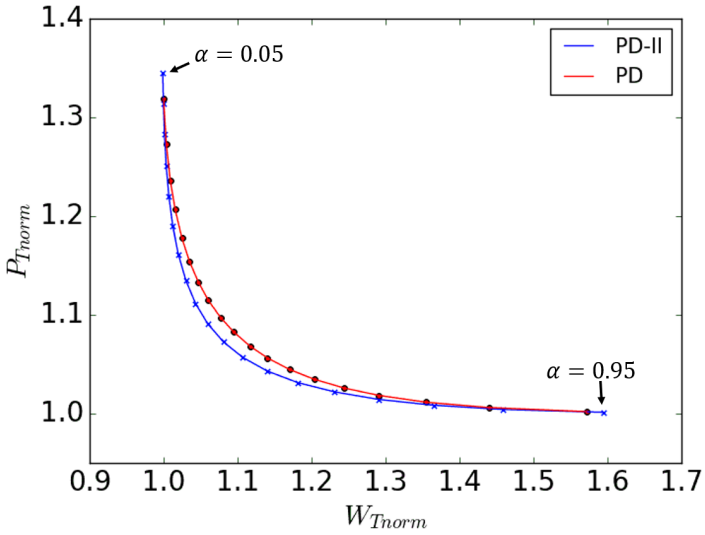


Figure 4.5: WL and PL tradeoff for various α.

The Pareto curve makes the improvement trend clear, but makes it difficult to measure the degree of improvement of *PD-II*. To compare the two algorithms more robustly, we analyze the results in the following way; (1) select different percentages of permissible WL degradation

with respect to MST WL (i.e., WL thresholds $= 1\%$, 2%, 4%, 7%, 10% and 15%), and (2) for each net, find the minimum P_{Tnorm} solution that meets the WL threshold across all solutions with different α. The results are averaged across all the nets and summarized in Table 4.3. Each entry in the table corresponds to the normalized PL P_{Tnorm}. To find the percentage improvement, one is subtracted from each value, since 1.0 is a lower bound. For example, a reduction from 1.15 to 1.12 results in an improvement of 20%, i.e., $(1-(1.12-1.0)/(1.15-1.0))\cdot 100\%$.

We observe the following:

- *PD-II* gives better results than *PD* for all classes of nets. This makes sense since it strictly improves upon an existing *PD* solution.

- Small nets obtain relatively small improvement, ranging from 0.26% to 1.63%; however, huge nets show significant improvements, ranging from 4.91% to 18.87%. Trends for medium and large nets lie in between. This is because the detour cost is close to optimal for smaller nets, but is much larger for bigger nets. For example, with a 1% WL threshold, the average normalized PL for *PD-II* is 1.097 for small nets but 1.376 for large nets.

- When the WL threshold is tight (such as 1% or 2%), the improvement of *PD-II* is much smaller as compared to looser constraints of 10% or 15%. This makes sense because a looser constraint gives the algorithms more freedom to reduce PL. A threshold of 1% means the topology cannot deviate much from the minimum-length spanning tree.

Table 4.3: Comparisons of the best P_{Tnorm} for *PD* and *PD-II* across different WL thresholds.

$\|V\|$	Method	WL threshold					
		1%	**2%**	**4%**	**7%**	**10%**	**15%**
Small	*PD*	1.0972	1.0927	1.0819	1.0680	1.0569	1.0427
	PD-II	1.0970	1.0923	1.0812	1.0672	1.0561	1.0420
	Imp. (%)	**0.26**	**0.42**	**0.78**	**1.15**	**1.36**	**1.63**
Medium	*PD*	1.1888	1.1746	1.1483	1.1189	1.0974	1.0723
	PD-II	1.1870	1.1706	1.1423	1.1122	1.0909	1.0668
	Imp. (%)	**0.93**	**2.33**	**4.07**	**5.66**	**6.62**	**7.68**
Large	*PD*	1.2981	1.2698	1.2216	1.1723	1.1390	1.1006
	PD-II	1.2895	1.2545	1.2025	1.1533	1.1219	1.0870
	Imp. (%)	**2.89**	**5.66**	**8.64**	**11.00**	**12.32**	**13.52**
Huge	*PD*	1.3952	1.3550	1.2873	1.2210	1.1777	1.1302
	PD-II	1.3758	1.3238	1.2526	1.1876	1.1488	1.1056
	Imp. (%)	**4.91**	**8.79**	**12.06**	**15.14**	**16.27**	**18.87**

Experiment II - Steiner Tree Results

Our next experiments compare (*PD* + *HVW* + *DAS*) and a baseline flow (PD + *HVW*) to show the value of *DAS*. *HVW* refers to the Steiner tree obtained after performing edge-overlapping as described by Ho et al. [89], and *DAS* refers to the Steiner tree after applying *DAS* algorithm to the *HVW* tree. Figure 4.6 shows the normalized WL and PL tradeoff comparison for the two flows for the set of large nets. Steiner tree W_{Tnorm} is defined as the ratio of total WL of the tree to the FLUTE WL[68] [38] and P_{Tnorm} is defined as the ratio of sum of PLs of all sinks in the tree to the sum of source-to-sink Manhattan distances. Each point in the curve represents the average (W_{Tnorm}, P_{Tnorm}) over all nets, for a particular value of α. It is clear that *DAS* adds significant value to the Steiner construction, pushing its Pareto curve further left and down compared to the one from the baseline.

Similarly to Table 4.3, Table 4.4 shows normalized PL across a range of permissible WL degradations for *HVW* versus *HVW+DAS*. We observe the following:

- DAS always obtains better results than HVW. Again, this makes sense since DAS starts with an HVW solution and further refines it to improve both WL and PL.
- Improvements for DAS can be quite significant, ranging from 8.36% to 83.67%.

[68]FLUTE constructs optimal RSMTs for nets with terminal sizes up to 9, and near-optimal RSMTs for nets with higher terminal counts.

- DAS improves results more significantly for smaller fanout nets than for larger ones. This may suggest there is still further room for improvement in Steinerization.
- Larger WL thresholds correspond to larger normalized PL improvements, which again is likely due to more freedom for the algorithm to find a solution that reduces detour cost.

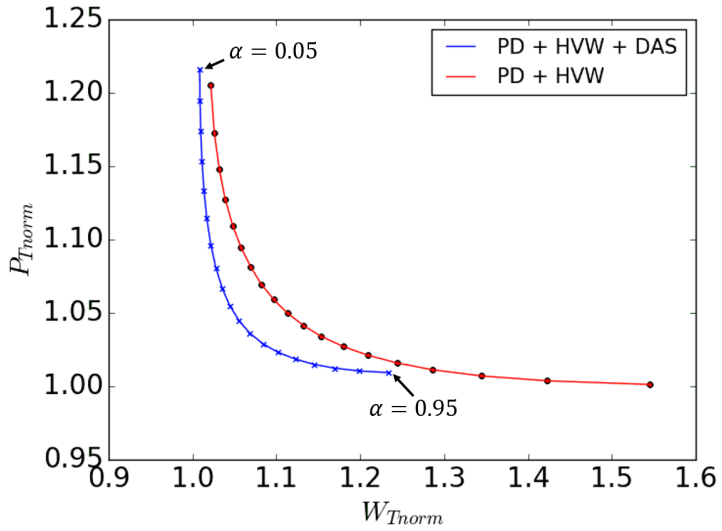


Figure 4.6: WL and PL tradeoff for Steiner tree constructions.

Table 4.4: Comparisons of the best P_{Tnorm} for (1) *PD + HVW* and (2) *PD + HVW + DAS* across different WL thresholds.

$\|V\|$	**Method**	**WL threshold**					
		1%	**2%**	**4%**	**7%**	**10%**	**15%**
Small	(1)	1.0233	1.0241	1.0250	1.0249	1.0236	1.0202
	(2)	1.0126	1.0115	1.0097	1.0073	1.0054	1.0033
	Imp. (%)	**46.14**	**52.31**	**61.15**	**70.85**	**77.30**	**83.67**
Medium	(1)	1.0786	1.0821	1.0828	1.0757	1.0649	1.0489
	(2)	1.0665	1.0629	1.0532	1.0385	1.0277	1.0168
	Imp. (%)	**15.43**	**23.30**	**35.78**	**49.07**	**57.24**	**65.58**
Large	(1)	1.1637	1.1644	1.1547	1.1275	1.1026	1.0728
	(2)	1.1440	1.1347	1.1087	1.0760	1.0553	1.0357
	Imp. (%)	**12.01**	**18.07**	**29.73**	**40.36**	**46.08**	**50.93**
Huge	(1)	1.2278	1.2091	1.1606	1.1107	1.0812	1.0538
	(2)	1.228	1.209	1.161	1.111	1.081	1.054
	Imp. (%)	**8.36**	**15.14**	**27.82**	**36.36**	**39.74**	**41.69**

Experiment III - Comparison with SALT [33]

Our final set of experiments compares the best combined flow (*PD-II + HVW + DAS*) with the results from the state-of-the-art academic Steiner tree construction, SALT [33]. SALT uses FLUTE [38] to generate its initial input and improves the initial construction to reduce PL. For nets with less than 10 terminals, FLUTE produces the optimal WL and may also produce excellent or even optimal PL, in which case running SALT is not even necessary. Hence, the cases for which FLUTE produces excellent PL are in some sense uninteresting. If FLUTE produces a good tradeoff curve, then SALT simply returns the FLUTE solution. Our approach can do something similar using the following simple metaheuristic: (1) run both FLUTE and (*PD-II + HVW + DAS*) in parallel; (2) if FLUTE is better than (*PD-II + HVW + DAS*) for both WL and PL, return the FLUTE solution, else return the (*PD-II + HVW + DAS*) solution. Essentially, the metaheuristic returns a solution identical to SALT's when the FLUTE solution is dominant. Note that for large and huge nets, the FLUTE solution almost never is dominant.

Figure 4.7 shows normalized WL and PL tradeoff curves for the metaheuristic flow and SALT for (a) small, (b) medium, (c) large and (d) huge nets. For small nets, SALT actually achieves better solutions than the metaheuristic until the normalized WL is about 2.3% higher than optimal.[69] However, for medium, large and huge nets, the Pareto curve for the metaheuristic outperforms the one from SALT, especially as nets increase in size. For huge nets, SALT achieves $W_{Tnorm} = 1.0370$, $P_{Tnorm} = 1.141$ for $\epsilon = 1.281$, which is its knee point in the tradeoff curve. The knee point in the metaheuristic's tradeoff curve corresponds to $W_{Tnorm} = 1.024$ and $P_{Tnorm} = 1.121$ at $\alpha = 0.35$, which achieves 35.13% WL and 14.18% PL improvements compared to SALT at its $\epsilon = 1.281$.

Since SALT optimizes shallowness and not detour cost, Figure 4.8 presents the same set of data but using SALT's proposed metrics. SALT dominates our method according to the shallowness metric. Thus, SALT is superior with respect to its proposed metric, while *PD-II + HVW + DAS* is superior with respect to its metric.

Finally, Table 4.5 compares our best recipe to SALT using the same methodology as Tables 4.3 and 4.4. Note that we use FLUTE WL as a lower bound. We observe the following:

- For small nets, and WL thresholds below 10%, SALT outperforms the proposed approach. SALT is also better on medium nets with WL thresholds below 2%. This makes sense since trees in this space will closely resemble FLUTE constructions. SALT starts with

[69] For {small, medium, large, huge} nets, FLUTE results for {55.6, 7.9, 0.03, 0}% of nets have smaller WL and PL than our results. As expected, FLUTE results are dominant for small nets, but our algorithm gives better PL for large and huge nets.

a FLUTE construction and iteratively improves it, so in the space where FLUTE obtains good trees for WL and PL, such an approach outperforms the algorithm proposed in this section. Note that the magnitude of the improvement is still small. For example, for small nets and a 1% threshold, SALT is 0.99% away from the optimal normalized path length, while our approach is 1.26% away.

- For large and huge nets, and for medium nets with thresholds larger than 2%, the proposed approach performs better, reaching a peak of 36.46% improvement for huge nets with a 10% threshold. This is the domain for which the optimal tradeoff can be considerably different from FLUTE. These arguably form the class of more interesting instances where the tradeoff between WL and PL becomes increasingly important.

- As WL threshold increases, the improvement of our approach vs. SALT improves too, especially around the 7% and 10% WL threshold ranges. However, for large and huge nets the improvement is somewhat less at the 15% threshold.

Table 4.5: Comparisons of the best P_{Tnorm} for (1) SALT and (2) *PD-II + HVW + DAS* across different WL thresholds.

$\|V\|$	**Method**	**WL threshold**					
		1%	**2%**	**4%**	**7%**	**10%**	**15%**
Small	(1)	1.0099	1.0093	1.0082	1.0067	1.0053	1.0036
	(2)	1.0126	1.0115	1.0097	1.0073	1.0054	1.0033
	Imp. (%)	**-27.29**	**-23.85**	**-17.98**	**-8.80**	**-0.86**	**7.90**
Medium	(1)	1.0652	1.0619	1.0547	1.0435	1.0337	1.0213
	(2)	1.0665	1.0629	1.0532	1.0385	1.0277	1.0168
	Imp. (%)	**-1.95**	**-1.66**	**2.76**	**11.32**	**17.63**	**21.15**
Large	(1)	1.1564	1.1475	1.1261	1.0961	1.0720	1.0432
	(2)	1.1440	1.1347	1.1087	1.0760	1.0553	1.0357
	Imp. (%)	**7.91**	**8.66**	**13.77**	**20.92**	**23.09**	**17.31**
Huge	(1)	1.2744	1.2574	1.2205	1.1688	1.1277	1.0763
	(2)	1.2278	1.2090	1.1606	1.1107	1.0811	1.0536
	Imp. (%)	**17.01**	**18.79**	**27.18**	**34.44**	**36.46**	**29.71**

Runtime. For the benchmarks studied, SALT's total runtime is 2762 seconds. By contrast, the *PD-II + HVW + DAS* algorithms, as implemented and optimized within a commercial EDA tool's code base, take 361 seconds in total. Thus, PD-II today runs more than 7 times faster than SALT.

Delay. Below, we show the impact of WL and PL improvement on delay. We estimate delays of nets produced by our algorithms and by SALT, based on the Elmore delay model with resistance of 37.318Ω per micron of wire, capacitance of $0.228fF$ per micron of wire, and $0.67fF$ pin capacitance per sink. For the solutions produced by our approach and SALT with WL threshold 2%, we calculate the sum of all sink delays for each net, and the average of this sum across all nets. For $\{$small, medium, large, huge$\}$ nets, the average sum of sink delays from PD-II is lower than the average sum of sink delays from SALT by $\{$-0.0005, 0.24, 1.54, 5.62$\}\%$. As seen with the WL and PL comparison, our algorithm has slightly larger delays for small nets and smaller delays for higher-fanout nets.

In summary, while our approach does not uniformly outperform SALT, it does provide a superior tradeoff for the most interesting class of nets that are far from optimal in terms of PL and WL.[70]

4.1.6 Conclusion

This section shows that the classic *PD* spanning tree algorithm that balances between Prim's and Dijkstra's algorithm can have a bad tradeoff that ends up with both WL and PL being highly suboptimal. A new spanning tree heuristic *PD-II* is demonstrated to significantly improve both WL and total detour cost compared to *PD*. Further, this section extends the construction to Steiner tree with the *DAS* algorithm that directly improves trees according to both objectives. The algorithms are shown to be fast and practical. They are also suitable for integration into existing commercial routers, and can be applied in conjunction with any existing spanning and Steiner tree constructions for simultaneous WL and PL improvements. Compared to the recent SALT algorithm, our construction generates clear improvements according to the proposed metrics, especially for medium-size and larger nets. Future research includes (i) revisiting the still-open question of worst-case detour from a *PD* construction; (ii) learning-based estimation of the best α for any given instance (i.e., set of pin locations of a signal net); and (iii) extending the detour cost objective to encompass sink criticality, "global" radius, and other additional criteria.

[70] The PD-II algorithm has been released as part of a leading commercial tool, with demonstrated improvements of timing and wirelength.

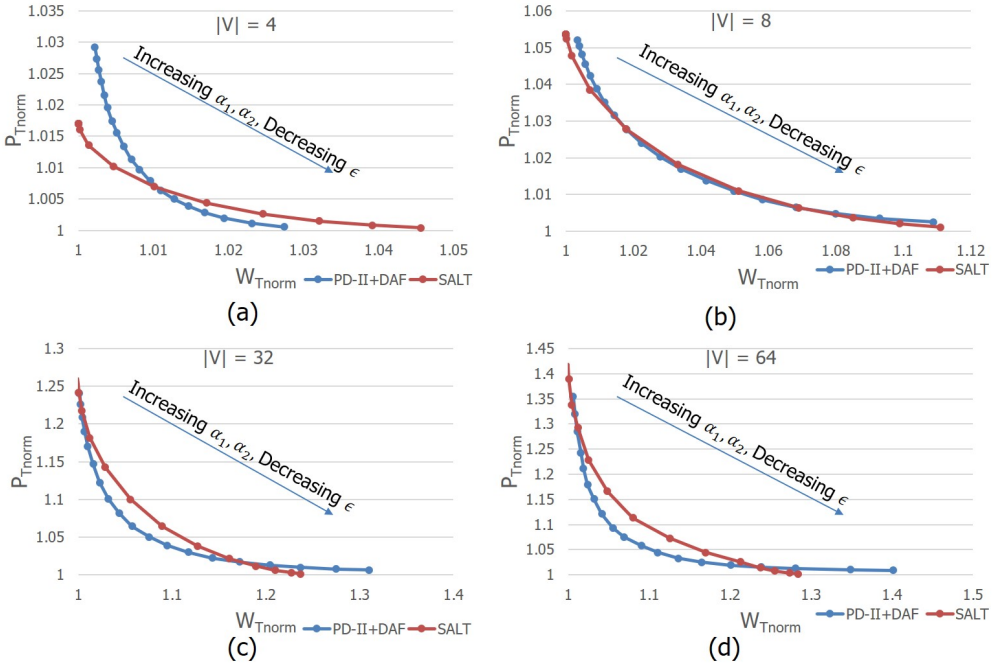


Figure 4.7: Normalized WL and PL for our metaheuristic and SALT on nets with $|V|$ = (a) 4 to 7, (b) 8 to 15, (c) 16 to 31, and (d) 32+.

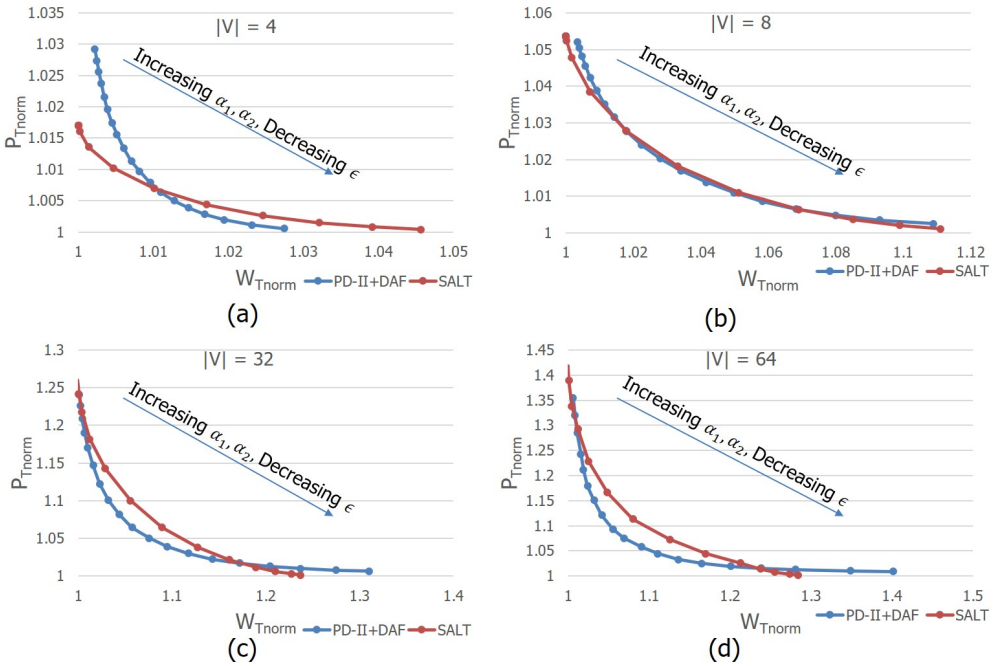


Figure 4.8: Average shallowness and lightness for our metaheuristic and SALT on nets with $|V|$ = (a) 4 to 7, (b) 8 to 15, (c) 16 to 31, and (d) 32+.

4.2 A Study of Optimal Cost-Skew Tradeoff and Remaining Suboptimality in Interconnect Tree Constructions

The difficulty of scaling integrated-circuit power efficiency, performance, area and cost (PPAC) in advanced technology nodes has been well-documented [207]. With the lack of new back-end-of-line interconnect materials, and consequent poor scaling of wire resistance and capacitance, there is increased pressure to improve the quality of interconnect layout. The recent paper of Alpert et al. [5] notes the power-sensitivity of modern (mobile, IoT, etc.) designs: "a 1% reduction in power is viewed as a big win for ... physical implementation", and "even a small WL savings with similar timing can have a high impact on value". In other words, there is renewed focus on the *cost* of interconnect trees in advanced VLSI.

Clock distribution has long been a crucial aspect of IC physical implementation since it strongly affects both power and performance. Clock routing brings together both cost and *skew* criteria: the cost-skew tradeoff [39] is particularly important in buffered clock tree construction, where clock subnets with ∼20 fanouts are a "sweet spot" for balancing of on-chip variation-aware analysis, skew, power and other factors. Future growth in the number of fanouts per clock buffer is unlikely, as fanout is limited by poor scaling of drive strengths relative to interconnect parasitics, increased use of multi-bit flip-flops to reduce clock wirelength and power, and increasing number of clocks in complex, low-power SOCs.[71]

Over the past decades, a number of works have studied the *bounded-skew routing tree* problem:

Bounded-skew routing tree (BST) problem. Given a set of terminals (points in the Manhattan plane) $P = \{p_1, p_2, \ldots, p_n\}$ with p_1 being the designated *root* (source) terminal, along with a skew bound B, construct a tree T with minimum cost $c(T)$ that contains all points of P and with root-terminal pathlength skew $\leq B$, i.e., $|d_T(1,i) - d_T(1,j)| \leq B$, for $2 \leq i < j \leq n$.

Here, the cost of an edge in T is its Manhattan length. The cost $c(T)$ of the tree T is the sum of its edge costs, and $d_T(1,i)$ denotes the sum of edge costs along the unique path in T from p_1 to p_i. The BST problem may be formulated in either the *spanning* or the *Steiner* contexts; we denote these respectively as the **BSSpanT** and the **BSSteinT** problems. Figure 4.9 illustrates

[71] A recent keynote address of TI's Anthony Hill [88] cites an IOT design with 200K instances, 200 distinct source clocks, an average of 12 clocks per register, and 1200 total clock domains.

the BST problem as well as the different nature of the BSSpanT and BSSteinT problems. Figure 4.9(a) shows the distribution of terminals for an example with $n = 8$. Figure 4.9(b) shows cost-skew tradeoff for this instance. Note that when $B = \infty$, the BSSpanT problem is the same as the rectilinear minimum spanning tree problem, and the BSSteinT problem is the same as the rectilinear Steiner minimum tree problem. Further, when $B = 0$ the BSSteinT problem becomes the exact zero-skew clock tree (ZST) problem studied in [19] [29] and many subsequent works. The unknown that we study in this section is the tradeoff between these extreme solutions.

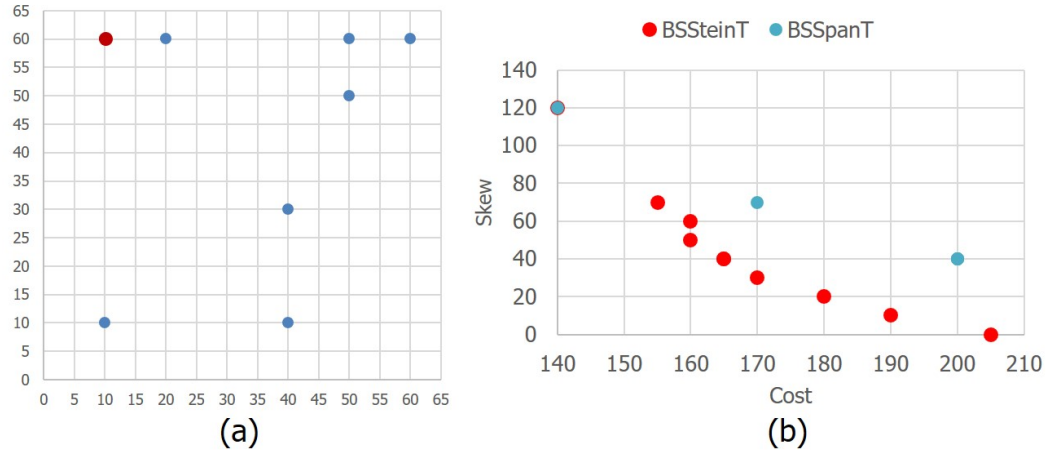


Figure 4.9: Illustration of the bounded-skew spanning and Steiner tree problems. (a) Distribution of terminals for an example with $n = 8$. (b) Minimum achievable BSSteinT cost decreases as the skew bound B increases. For the same instance and the same values of B, a BSSpanT may not always exist.

The VLSI CAD literature of the 1990s developed constructions for bounded-skew clock and Steiner routing trees [39][93] [112]. Additionally, cost-radius tradeoff methods such as shallow-light trees [40][117] (see also [41][119] [53] [54]) or Prim-Dijkstra trees [5][9] were applied in the bounded-skew context, since bounding the radius of a tree trivially also bounds the skew of a tree. During the 2000s, the discrete algorithms community addressed bounded-skew tree construction in works such as [30][194]. In recent advanced nodes, where both performance and power are critical to IC products, there is now intense focus on the challenge of minimizing clock distribution wirelength while controlling skew.

Our present work revisits the bounded-skew routing tree problem – in both the spanning tree and Steiner tree contexts – with an aim to determine "how much is left on the table". While VLSI interconnect trees are ultimately realized as Steiner trees, the spanning formulation is of distinct interest. Footnote 1 of [5] observes that "For global routing, spanning trees are often preferred to Steiner trees since global routing commonly decomposes multi-fanout nets into two-

pin nets. A spanning tree provides the router with an obvious decomposition. However, Steiner trees are not well-suited for this because the Steiner points become unnecessary constraints that restrict the freedom of the router to resolve congestion." Indeed, the spanning and Steiner formulations have different "behaviors": Elkin and Solomon [54] show that Steiner shallow-light trees can be exponentially lighter than their spanning counterparts (recall also Figure 4.9).

The key contributions of this section are as follows.

- We formulate the minimum-cost bounded skew spanning and Steiner tree problems as flow-based integer linear programs and give the first-ever study of *optimal* cost-skew tradeoffs.
- We evaluate the heuristics (Bounded-Skew DME and Prim-Dijkstra variants) that are currently the best available methods for trading off cost and skew, and quantify remaining suboptimality.

In the following, Section 4.2.1 summarizes related work on cost-delay tradeoffs and bounded-skew tree constructions. Section 4.2.2 describes flow-based ILP formulations for the spanning and Steiner BST problems. Section 4.2.3 experimentally demonstrates a surprisingly substantial "gap" between existing heuristic tree constructions and optimum bounded-skew trees. We conclude in Section 4.2.4 with ongoing and future directions.

4.2.1 Related Work

Many spanning and Steiner tree heuristics have been proposed for VLSI routing applications. These heuristics typically optimize or trade off the fundamental objectives of tree cost, delay and skew [109]. Types of tree constructions for VLSI that are related to our present work can be classified into three main categories: cost-delay tradeoffs, bounded-skew constructions, and optimal (integer programming-based) constructions.

Cost-delay tradeoffs have most famously been achieved by *shallow-light* constructions, which optimize cost (wirelength) and radius (maximum source-sink pathlength) simultaneously to within constant factors of optimal. For example, the BRBC algorithm [40] produces a tree that has wirelength no greater than $1 + 2/\epsilon$ times that of a minimum spanning tree (MST), and radius no greater than $1 + \epsilon$ times that of a shortest-paths tree (SPT). Over the ensuing 25+ years, numerous works ranging from [117] to [54][33] have continued to improve the basic approach. The SALT method of [33] is the most recent and strongest work in the shallow-light literature, incorporating additional techniques such as post-processing via edge flipping [89]. The *Prim-*

Djkstra algorithm [9] achieves in practice a high-quality tradeoff between tree cost and maximum source-terminal pathlength (i.e., radius), but has no provable shallow-light property. The very recent work of [5] improves the original Prim-Dijkstra method with topology and edge-flipping optimizations, and can produce tree solutions superior to [33]. Additional work has studied the *rectilinear Steiner arborescence* (RSA) problem, which seeks to find a minimum-cost tree that achieves optimal source-sink delay *at every sink*, i.e., a minimum-cost shortest-paths Steiner tree. Rao et al. [158] and Cong et al. [42] give heuristics for the RSA problem, which is known to be NP-complete [173]; an implementation of the A-Tree method of [42] is available at [85].

Bounded-skew tree (BST) constructions originally arose as extensions of deferred-merge embedding (DME) based zero-skew tree (ZST) constructions [19][29][111]. Notably, [39][93][112] all extend the DME algorithm to achieve BST routing. With a skew bound of $B = 0$, the BST problem reduces to the ZST problem. When $B = \infty$, the BST problem reduces to the rectilinear Steiner minimum tree (RSMT) problem. Tsao and Koh [185] improve the DME algorithm's bottom-up merging step to construct trees subject to general skew constraints. Empirical results show improvements over BST-DME with certain skew constraints. In the discrete algorithms literature, works of Charikar et al. [30] and of Zelikovsky and Mandoiu [194] propose ZST and BST heuristics with constant-factor error bounds; the latter work gives a realizable ZST construction based on the "rooted-Kruskal" approach which guarantees rectilinear BST cost within 9 times of optimal. Rajaram et al. [151] apply bounded-skew tree construction within low-cost (cross-link insertion-based) nontree routing. Below, we study cost-skew tradeoff performance of an updated version [180] of the open-source BST-DME implementation of Tsao [184].

A number of *optimal* tree constructions have been proposed as well. The well-known FLUTE method of Chu and Wong [38] uses topology pruning and look-up tables to find RSMT solutions extremely efficiently; FLUTE solutions are optimal for instances with up to $\sim$9 pins. The well-known GeoSteiner code of Warme et al. [202] can solve the RSMT problem optimally for instances with thousands of points. The work of Peyer et al. [147] exemplifies the use of integer linear programming (ILP) to solve the Steiner tree problem via the flow-based *directed Steiner tree* framework. Single-commodity and multi-commodity flow-based ILPs have been also used by Han et al. [78] to assess back-end-of-line design rule impacts on local routability, and by Jia et al. [96] within a detailed router. Aneja [14] applies a set-covering ILP to the construction of Steiner trees given a prescribed set of Steiner points. A "row generation" technique prevents an exponential number of constraints from arising. Oh et al. [139] use linear program-

ming to find Steiner routing trees with upper-and lower-bounded path delays within a prescribed topology; a method similar to BST-DME is used to embed the Steiner points in the Manhattan plane. No previous work that we are aware of optimally solves either the spanning or the Steiner form of the bounded-skew tree problem. Below, we experimentally study a flow-based ILP that solves both the spanning and Steiner BST formulations.

4.2.2 Flow-based ILP Formulation

We now formulate an ILP that can be generally applied to both the BSSpanT and BSSteinT problems. In this section, we first introduce our bounded-skew tree routing formulation. Second, we then explain constraints that detect and block any cycles, such that the ILP outputs a well-formed tree as its solution. Third, additional constraints to improve runtime are explained. Table 4.6 lists the notations that we use.

Table 4.6: Notations.

Notation	Meaning
p_i	i^{th} point ($p_i \in P$, p_1 is a root point)
v_i	i^{th} vertex ($v_i \in V$, $P \subseteq V$)
e_{jk}	a directed edge from vertex v_j to vertex v_k ($e_{jk} \in E$)
λ_{jk}	0-1 indicator of whether e_{jk} is in a tree T
c_{jk}	cost of edge e_{jk}
f^i_{jk}	0-1 indicator of whether the flow to p_i goes through e_{jk}
d^i_j	pathlength at v_j along unique v_1-p_i path
m_{ij}	Manhattan distance from v_i to v_j
B	skew bound
L	lower bound on pathlength from source v_1

Bounded-Skew Tree Routing

Given a set of terminals P, specified as (x, y) points in the Manhattan plane with x, y integers, we create a graph $G = (V, E)$ whose vertex set V contains P as well as additional points S (i.e., $V = P \cup S$). (Thus, each point in P is identified with some vertex in V.) For the BSSpanT problem, S is empty and E consists of all $|P| \cdot (|P| - 1)$ possible directed edges between pairs of terminals. Thus, $G = (V, E)$ is a complete graph in the BSSpanT problem. For the BSSteinT problem, S is a set of non-terminal points in a *half-integer grid* of P, and E is a set of directed edges between any neighboring vertices. Each vertex has up to four outgoing and four

incoming edges to/from neighbor vertices in the east, west, north and south directions. The half-integer grid consists of all points in the bounding box of P for which both x and y coordinates are multiples of $1/2$. By convention, we assume that (i) $v_1 = p_1$ is the *root* (i.e., source) terminal, (ii) $\{v_2, v_3, ..., v_{|P|}\} = \{p_2, p_3, ..., p_{|P|}\}$ are the *leaf* (i.e., sink) terminals, and (iii) other vertices $\{v_{|P|+1}, ...v_{|V|}\}$ are additional non-terminal vertices. We formulate the following integer linear program:

$$\text{Minimize:} \quad \sum_{j,k} \lambda_{jk} \cdot c_{jk}$$

Subject to:

$$\lambda_{jk} \geq f^i_{jk} \qquad \forall p_i \in P,\ e_{jk} \in E \tag{4.4}$$

$$\sum_j f^i_{jk} - \sum_j f^i_{kj} = \begin{cases} 1 & \text{if } v_k = v_1, \forall v_j \in V,\ p_i \in P,\ i \neq 1 \\ -1 & \text{else if } v_k = v_i \\ 0 & \text{otherwise} \end{cases} \tag{4.5}$$

$$\sum_{j,k} c_{jk} \cdot f^i_{jk} \geq L \quad \forall p_i \in P,\ i \neq 1 \tag{4.6}$$

$$\sum_{j,k} c_{jk} \cdot f^i_{jk} \leq L + B \quad \forall p_i \in P,\ i \neq 1 \tag{4.7}$$

Our objective is to minimize total cost, while satisfying a given skew bound B. We consider each path from p_1 to p_i as a separate flow. λ_{jk} is a global binary variable that indicates whether any flow goes through an edge e_{jk} in the tree solution T. Constraint (4.4) forces $\lambda_{jk} = 1$ when a flow exists in e_{jk}.

Flow conservation. Constraints (4.5) are for flow conservation. (A unit of flow from source v_1 to sink v_i will traverse a path from p_1 to p_i.) These constraints enforce that (i) there is one net outgoing unit of flow at a vertex v_k that is the vertex identified with the root terminal p_1; (ii) there is one net incoming unit of flow at a vertex v_k that is the vertex identified with the leaf terminal p_i, and (iii) otherwise, the sum of incoming and outgoing flow at vertex v_k must be zero. Since each flow for each path is considered exclusively, for the flow to the terminal p_i, other terminals' vertices (i.e., $v_2, ..., v_{|P|}$ except for v_i) are not considered as leaf terminals.

Skew bound constraints. Given L and B, Constraints (4.6) and (4.7) respectively bound the minimum and maximum pathlengths for all source-to-sink paths. However, with these constraints only, invalid solutions that contain cycles could arise. Next, we add more constraints to block the formation of cycles.

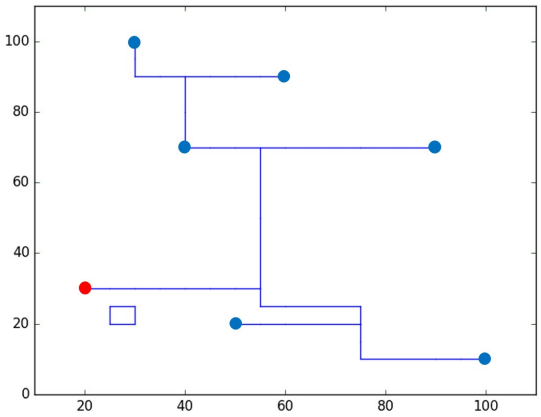


Figure 4.10: Example solution with a cycle on the lower-left corner. Red dot is a root and blue dots are leaf terminals.

Cycle Correction

Figure 4.10 shows an example solution with a cycle. This solution satisfies the above flow conservation constraints since the sum of incoming and outgoing flows are the same for each vertex in the cycle. This happens when the cost of creating the cycle is less than the cost of detouring to leaf terminals (that are close to the root) in order to satisfy a given skew bound. To prevent the cycle, we define a new d_j^i variable that represents the pathlength from p_1 to v_j for path p_i. The d_j^i should satisfy Constraint (4.8):

$$\begin{cases} d_j^i = 0 & \text{if } v_j = v_1, \forall\, p_i \in P,\ i \neq 1 \\ d_j^i \geq L & \text{else if } v_j = v_i, \forall\, p_i \in P,\ i \neq 1 \\ d_j^i \leq L + B & \text{otherwise } \forall\, p_i \in P,\ i \neq 1 \end{cases} \tag{4.8}$$

In other words, the pathlength from the root to any vertex v_j must lie between prescribed minimum and maximum pathlength bounds. To compute d_j^i, we add the following constraints.

$$d_j^i + U \cdot (1 - f_{kj}^i) \geq d_k^i + c_{kj} \quad \forall v_j, v_k \in V, j \neq k,\ p_i \in P \tag{4.9}$$

$$d_j^i - U \cdot (1 - f_{kj}^i) \leq d_k^i + c_{kj} \quad \forall v_j, v_k \in V, j \neq k,\ p_i \in P \tag{4.10}$$

When $f^i_{kj} = 1$, d^i_j should be equal to d^i_k. Otherwise, these constraints are always met with a large constant value U. With these constraints, d^i_j becomes infinite if there is a cycle.

Constraints for Runtime Improvement

It is well known that the chief drawback of using ILP is long, poorly-scaling runtime. We add further constraints to predetermine some variables according to what we might know before we run the ILP instance. The idea is to find flow variables that cannot exist, or combinations of variables that cannot coexist. This reduces the solution space for an ILP solver to explore.

$$\begin{aligned} &\text{if } m_{1j} + m_{ji} > L + B \;\; \forall v_j \in V,\; j \neq 1, ..., |P|,\; p_i \in P \\ &\quad f^i_{jk} = 0,\; f^i_{kj} = 0 \quad \forall e_{jk} \in E,\; e_{kj} \in E \end{aligned} \tag{4.11}$$

For any non-terminal vertex v_j, if the sum of Manhattan length from root terminal p_1 to v_j and from v_j to leaf terminal p_i is larger than the upper bound on pathlength $L + B$, all incoming and outgoing flows for p_i going through v_j should be zero.

Similarly, we can consider two non-terminal vertices v_j and $v_{j'}$:

$$\begin{aligned} &\text{if } m_{1j} + m_{jj'} + m_{j'i} > L + B \\ &\&\& \;\; m_{1j'} + m_{j'j} + m_{ji} > L + B \\ &\quad f^i_{jk} + f^i_{j'k'} = 1, \quad f^i_{kj} + f^i_{j'k'} = 1,\; \forall e_{jk} \in E,\; e_{kj} \in E \end{aligned} \tag{4.12}$$

$$f^i_{jk} + f^i_{k'j'} = 1, \quad f^i_{kj} + f^i_{k'j'} = 1,\; \forall e_{j'k'} \in E,\; e_{k'j'} \in E \tag{4.13}$$

For any two non-terminal vertices v_j and $v_{j'}$, if the sum of Manhattan lengths from p_1 going through v_j and $v_{j'}$ to terminal p_i is larger than the pathlength upper bound, then any combinations of incoming and outgoing flows going through v_j and $v_{j'}$ cannot coexist. Our ILP implementation applies such additional constraints to reduce ILP solver runtime.

Analysis of the Number of Variables and Constraints

The number of variables and constraints depends on the number of edges ($|E|$), vertices ($|V|$) and points ($|P|$).

- The number of variables λ_{jk} is $|E|$.
- The number of variables f_{jk}^{i} is $|E| \cdot |P|$.
- The number of variables d_{j}^{i} is $|V| \cdot |P|$.
- The number of Constraints (4.4) is $|E| \cdot |P|$.
- The numbers of Constraints (4.5), (4.8), (4.9), (4.10) are $|V| \cdot |P|$.
- The numbers of Constraints (4.6) and (4.7) are (each) $|P|$.

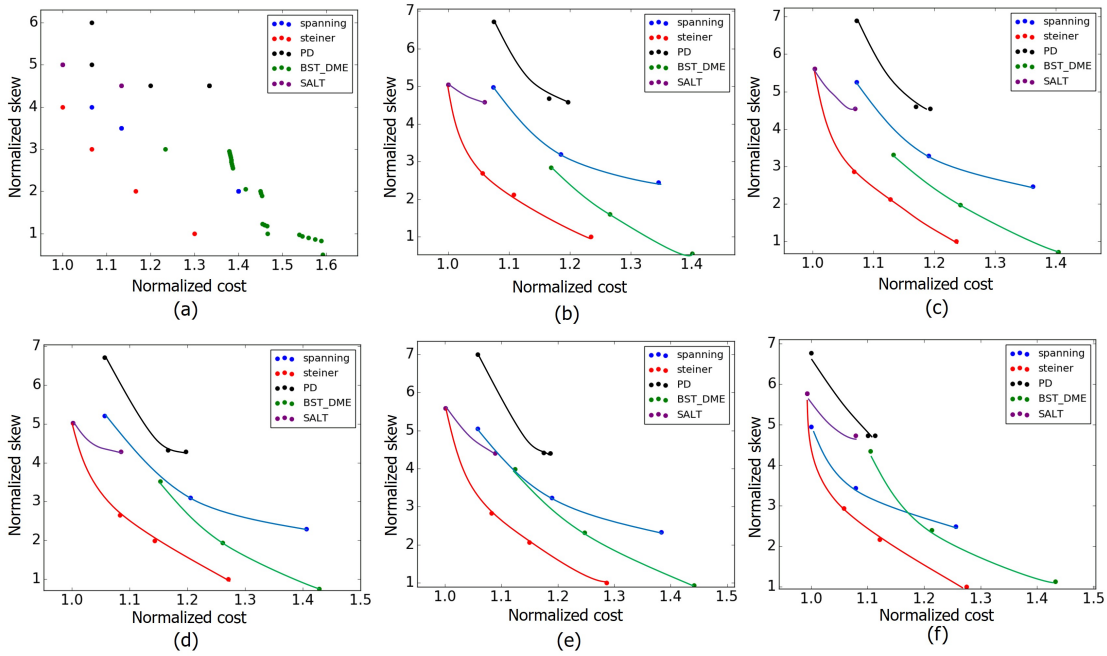


Figure 4.11: Illustration of cost-skew tradeoff: (a) cost-skew tradeoff curve for one 14-terminal instance, and (b)-(f) cost-skew tradeoff curves for all 8-, 10-, 12-, 14- and 16-terminal instances, respectively.

4.2.3 Experimental Setup and Results

Experimental Setup

We implement our tool in C++ and use *CPLEX v12.6* [203] as our ILP solver. The following experiments are performed on a 2.7 GHz Intel Xeon server with 32 threads.

Even with the additional constraints for runtime improvement, our flow-based ILP for Steiner tree only works for a limited condition (i.e., $|P| \leq\sim$16, $|V| \leq\sim$140, $|E| \leq\sim$550). Under this condition, we generate 50 testcases for each $|P| = \{8, 10, 12, 14, 16\}$. The terminals of each testcase are randomly distributed.

For the generated testcases, we run our flow-based ILP for Steiner tree, and obtain cost-skew tradeoff curves. For each instance, we run ILP with four different skew bounds $= M \cdot \{0.3,\ 0.5,\ 0.7,\ \infty\}$, where M is the maximum source-to-sink Manhattan length. We do not run our ILP with skew bound $B = 0$ due to the long runtime. We also set $L = M - B$ for a given B. When a fixed L is used, our ILP Steiner tree solution could end up with a suboptimal solution. The impact of a fixed L on suboptimality is discussed in Experimental Results below.

We also run several academic tools for evaluation; BST-DME [184], SALT [33] and PD [9]. For each tool, we sweep input parameters to obtain several solutions.

Experimental Results

Study of cost-skew tradeoff. We normalize the costs (resp. skews) of academic tools' solutions as well as our ILP-based spanning tree solutions by the minimum cost (resp. skew) achieved from ILP-based Steiner tree for each testcase. Figure 4.11(a) shows the results for one 14-terminal instance. Each data point is mapped to a solution from the corresponding tool. This figure clearly shows that our ILP-based Steiner tree solutions are dominating the other tools' solutions on both skew and cost. Some solutions from BST-DME achieve slightly better skew than our minimum skew from ILP-based Steiner tree solutions. This is because we do not have a run with $B = 0$. Figure 4.12 shows the plots of ILP-based Steiner tree solutions for this 14-terminal instance.

For a more comprehensive study across different instances, we propose the following way for comparison. (1) For each tool, we select three representative solutions: the minimum-skew solution, the minimum-cost solution and a "median" solution in between. (2) We then compute the average normalized cost (resp. skew) for the same set of solutions (e.g., minimum skew solutions) for all 50 instances.

Figures 4.11(b)-(f) show the cost-skew tradeoff curves for all $|P| = \{8, 10, 12, 14, 16\}$, respectively. From these figures, we observe that:

- ILP-based Steiner tree dominates all other tools in terms of both cost and skew across all terminal nets.
- Compared to the Steiner tradeoff curve, BST-DME is $\sim 10\%$ suboptimal in cost at iso-skew and $\sim 50\%$ suboptimal in skew at iso-cost. Solutions for different terminal nets show a similar trend.
- Both SALT and PD mostly generate large skew solutions, with up to $\sim 3\times$ suboptimality. This suggests that optimizing shallowness only, without awareness of skew, is insufficient to find solutions with good skew.
- In general, ILP-based spanning tree solutions have larger skew and cost than the ones from SALT and BST-DME. However, as the number of terminals increases, it generates comparable or better cost and skew solutions than SALT and BST-DME. This implies that the optimal spanning tree solutions could be good candidates for high-fanout nets.

Runtime. ILP for spanning tree runs very fast. Average runtime is 0.32 second and maximum runtime is 4 seconds. On the other hand, ILP for Steiner tree uses the half-integer grid to ensure that optimal solutions. Thus, as we increase the number of terminals, the number of vertices and edges increase and runtime goes up quickly. Runtime also increases as the skew bound is tightened. Table 4.7 shows the average runtime for different $|P|$ and skew bound.

Table 4.7: Average ILP runtime for Steiner tree.

Skew bound	$\|P\| = 8$	$\|P\| = 10$	$\|P\| = 12$	$\|P\| = 14$	$\|P\| = 16$
Unbounded	0.33	0.50	0.68	1.04	0.78
$0.7 \cdot M$	2.89	5.22	41.24	130.08	58.53
$0.5 \cdot M$	13.20	74.98	364.03	1522.25	892.50
$0.3 \cdot M$	320.77	662.01	2593.59	4664.06	3477.73

Possible suboptimality with fixed L. Due to runtime scaling and available computational resource, we have used a fixed L (computed as $M - B$; see Experimental Setup above) in our reported results for ILP-based bounded-skew Steiner trees. However, a fixed L can cause small suboptimality in the solutions. To study the impact of L on suboptimality, we select 10 sample instances (two instances from each testset with a given number of terminals) and vary

L from $M - B$ to M. We then compare the best cost found to the cost obtained using a fixed $L = M - B$. We find that one out of 10 nets is suboptimal due to the fixed L and the suboptimality is 2.8%.

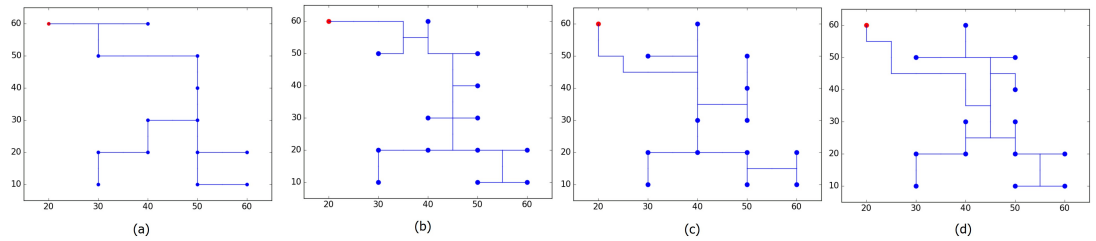


Figure 4.12: Illustration of ILP-based Steiner tree solutions for a 14-terminal instance with (a) $B = \inf$, (b) $B = 0.7 \cdot M$, (c) $B = 0.5 \cdot M$ and (d) $B = 0.3 \cdot M$.

4.2.4 Conclusion

In this section, we empirically study the minimum-cost bounded skew spanning and Steiner tree problems. We formulate and apply a flow-based ILP to find optimal cost-skew tradeoffs for generated testcases with number of terminals from 8 to 16. Based on the optimal cost-skew tradeoffs, we find significant remaining suboptimality of several state-of-art academic tools: (1) BST-DME, (2) SALT and (3) Prim-Dijkstra. Across our testcases, BST-DME has suboptimality $\sim 10\%$ in cost at iso-skew, and $\sim 50\%$ in skew at iso-cost. In addition, SALT and PD show suboptimality in terms of skew by up to $\sim 3\times$. This degree of suboptimality is very different from the near-optimality in practice of heuristics for the RSMT problem (e.g., FLUTE, 1-Steiner, etc.). Thus, our study motivates renewed attention to the cost-skew tradeoff.

Our future work includes (1) further scalability study and improvement of the flow-based ILP formulation; (2) extensions of our suboptimality study to the cost-radius tradeoff and well-studied variants (sink-specific radius bounds, critical-sink trees, required arrival time (RAT) trees, etc.), and (3) benchmark suite generation with known optimal solutions to various formulations.

4.3 Optimal Generalized H-Tree Topology and Buffering for High-Performance and Low-Power Clock Distribution

Physical implementation of clock distribution networks is increasingly critical to the success of high-performance, low-power IC product designs. Clock distribution takes up substantial routing and buffering resources as well as a significant portion of overall power consumption [146]. Power dissipation in the clock network has often been estimated to be one third of total IC power dissipation [131], or even half the total power in some designs. Further, the quality (skew and latency) of clock delivery strongly determines achievable performance of the design, particularly in advanced nodes. Skew is well known to affect datapath area and power, as well as the design schedule needed to achieve timing closure [45][153][159][170][175][190]. Maximum clock latency is another key metric of the clock distribution network in advanced nodes since skews are magnified by on-chip variation (OCV) deratings [26].

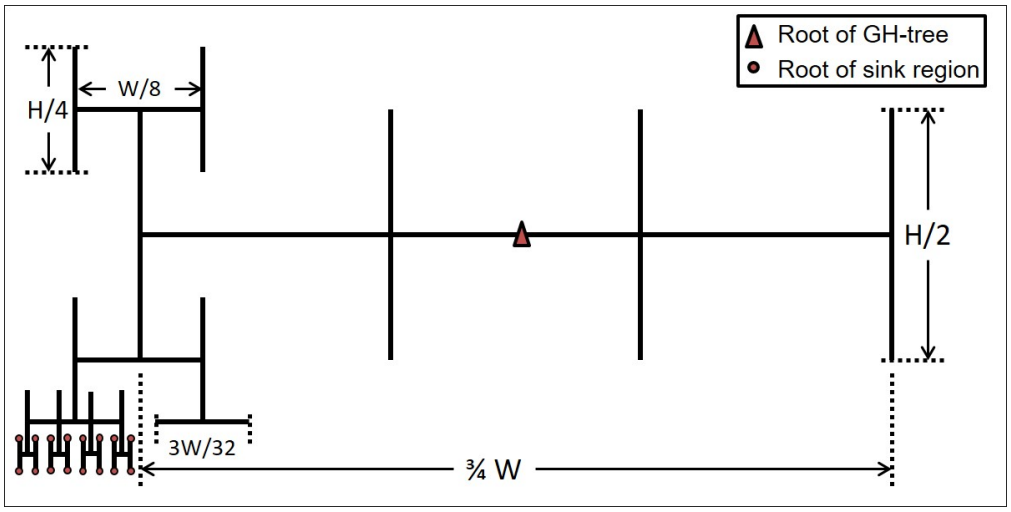


Figure 4.13: 8-level GH-tree with branching pattern (4, 2, 2, 2, 4, 2, 2, 2).

As reviewed below, there are several methods of clock distribution. Tree-based constructions are still dominant, and remain the default of commercial clock tree synthesis (CTS) tools. To reduce skew and increase robustness (e.g., in light of manufacturing variability or reliability mechanisms), mesh and other non-tree topologies (e.g., trees + cross-link insertion) have been used. Such non-tree methods typically have large overheads in terms of power, area, wirelength and signoff analysis complexity. Thus, clock trees are still of interest and great practical relevance for reasons of cost efficiency, flexibility and design flow complexity. Increasingly, *structured* approaches to clock tree design are seen in practice, since these offer benefits of predictability in the resource-versus-skew tradeoff, particularly in the upper levels of clock trees.

As a special case of structured clock trees, the highly regular, recursive *H-tree* embedding of a complete binary tree [17] offers minimum skew, but at the cost of larger wirelength and potentially larger clock power and latency. "Fishbone" clock tree topologies (e.g., [13]) with spines and ribs can be more cost-efficient in terms of latency, wirelength, area and clock power – but incur varying propagation delays (that is, of the clock signal to branching points along a given spine) which cause skew. To explore the tradeoff among skew, latency and (clock power, clock buffer area) cost, this section proposes the concept of a *generalized H-tree* (GH-tree), which is a balanced tree topology with arbitrary branching factor at each level. (Like the H-tree, the GH-tree is a multi-level topology; like the fishbone, it can have branching factor greater than two.) Figure 4.13 shows a GH-tree with depth $P = 8$ and branching factors (4, 2, 2, 2, 4, 2, 2, 2) at levels $p = 1, 2, ..., 8$. In the example, we assume there are 1024 (= $4 \cdot 2 \cdot 2 \cdot 2 \cdot 4 \cdot 2 \cdot 2 \cdot 2$) nodes (sinks) uniformly placed in the region. If the root of the tree is at the region center, each root-to-leaf path will contain horizontal segments of lengths $\frac{3 \cdot W}{4}$, $\frac{W}{8}$, $\frac{3 \cdot W}{32}$ and $\frac{W}{64}$, in that order. The lengths of the successive vertical segments in each root-leaf path are $\frac{H}{2}$, $\frac{H}{4}$, $\frac{H}{8}$ and $\frac{H}{16}$. We note that in our proposed GH-tree, we do not necessarily insert a buffer at each branching point. Further, we allow buffering that is internal to a given branch, that is, at any location along wiring of that given branch.

In this section, we study potential benefits of the generalized H-tree for low-power, low-skew, and low-latency clock distribution. We propose a dynamic programming (DP) algorithm that efficiently finds an *optimal*[72] GH-tree with minimum clock power for given latency and skew targets. This optimization uses calibrated clock buffer library and interconnect timing and power models, and co-optimizes the clock tree topology along with the buffering along branches. Furthermore, we also propose a clustering and linear programming-based heuristic to *embed* the GH-tree with respect to a given placement of clock sinks. Finally, our embedding of the GH-tree is blockage-aware. In a 28LP testbed with multi-corner timing constraints, our embedded GH-tree solutions provide significant clock power benefits (iso-skew and -latency) in comparison to commercial CTS solutions from the place-and-route tools of two leading EDA vendors as well as a state-of-the-art academic CTS tool. Our contributions are summarized as follows.

- We propose the concept of a *generalized H-tree*, which is a balanced tree topology that can have an arbitrary sequence of branching factors.
- We propose a DP-based method to co-optimize clock tree topology and buffering to

[72] We note that our claim of an *optimal* clock tree solution is in the regime of generalized H-tree solutions (i.e., the continuum between H-tree and spine). We do not claim that our optimal GH-tree is a globally optimal clock tree solution.

achieve an optimal GH-tree solution with respect to the tradeoffs among skew, latency and clock power.

- We propose a *balanced K-means clustering* and linear programming-based buffer placement to embed our GH-tree solution with respect to any given sink placement.
- We validate our GH-tree optimization based on sink placements from a leading commercial place-and-route (P&R) tool, which include testcases with high floorplan aspect ratio and existence of blockages.
- Our methodology and optimizations can easily be integrated with commercial P&R tools. Our experimental results in a foundry 28LP technology with multi-corner testcases show up to 30% clock power reductions compared to current CTS tool solutions from two leading EDA vendors and up to 56% clock power reduction compared to a state-of-the-art academic solution [118].
- Our optimizer is open-sourced as TritonCTS in GitHub (https://github.com/abk-openroad/TritonCTS).

4.3.1 Motivation and Related Work

In this section, we first review previous works on clock distribution, categorizing them as: (i) non-tree methods and (ii) tree-based methods. We then provide a motivating analysis of the GH-tree solution space.

Non-tree methods. Mesh topologies are commonly understood to provide robustness and small skew. Many works, such as [75][76][160][161], propose clock mesh designs for high-performance circuits that require robust clock networks. To reduce the cost (e.g., wirelength, power), hybrid clock distribution methods integrating both mesh and tree topologies have been proposed [159][175]. Several works [55][91][153][162] propose to insert cross-links for minimization of clock skew, based on an initial clock tree solution, with small power overhead. Rajaram et al. [153] propose an algorithm to recursively merge subtrees with backward slew propagation. Ewetz and Koh [55] propose systematic cross-link insertion methods to improve the robustness of a clock tree while minimizing its overheads. They propose a vertex reduction method to reduce the amount of redundancy in their non-tree structures. Although non-tree methods reduce clock skew and enhance robustness of clock networks, their intrinsic redundancy incurs additional cost (e.g., wirelength, power, effort of verification) as compared to tree-based

methods. Furthermore, non-tree clock distribution topologies such as meshes lack flexibility and tunability; this can block, e.g., useful skew optimizations.

Dolev et al. [48] propose a hexagonal grid-based clock topology (HEX), consisting of a hexagonal grid with intermediate nodes that control the clock signals in the grid and supply the clock signals to nearby functional units. Abdelhadi et al. [1] propose an algorithm to construct a variation-tolerant hybrid clock network based on a combination of non-uniform meshes and unbuffered trees. Their method selectively reduces clock skew variations on critical timing paths. Zhou et al. [197] propose an algorithm to determine tapping points for local buffers that drive a clock mesh with non-uniform load distribution in a tree-driven grid clock network. Their algorithm first calculates load for each node, then clusters the nodes. Tapping points are determined for each cluster based on the minimum and maximum latencies.

Recently, Y. Kim and T. Kim [118] have proposed a synthesis algorithm for clock spine networks that effectively optimizes the tradeoff between clock resource and variation tolerance. The key idea of their algorithm is to treat the clock spine allocation and placement problem as a slicing floorplan optimization problem. Clock tree solutions from the CTS algorithm of [118] are compared to our GH-tree solutions in Section 4.3.3.

Tree-based methods. Due to their cost efficiency, clock tree-based methods have been commonly used for clock distribution in low-power designs. Early works [39][29][111][185] propose clock tree constructions based on linear or Elmore delay models to minimize wirelength for a given skew target. However, delay and power impacts of buffers are ignored in these works. Approaches in [31][74][130][135][154][157][183][190] comprehend buffering impact and co-optimize clock tree construction (i.e., tree topology) with buffering. Vittal and Marek-Sadowska [190] give an early algorithm that co-optimizes tree topology and buffer insertion. Mehta et al. [135] propose a clustering algorithm to obtain approximately load-balanced clusters and construct clock trees so as to minimize skew. These previous approaches typically construct the clock tree in a bottom-up way with a greedy algorithm, and do not explore the skew vs. cost tradeoff. Furthermore, few works adapt their tree construction approaches to, and validate their solution quality with, commercial P&R tools and realistic design blocks. Other works [26][79] are ECO-based incremental optimizations based on an initial clock tree solution generated by commercial P&R tools. The objective functions of these works differ from ours. Chan et al. [26] minimize skew at the top-level, whereas Han et al. [79] minimize skew variation across corners.

A motivating analysis. To motivate our main studies below, we briefly illustrate the tradeoffs among clock tree wirelength, global skew and maximum clock latency seen across GH-

tree topologies with various branching patterns. Before doing so, we summarize in Table 4.8 the terminology and notation used in the remaining discussion.

Table 4.8: Notations.

Term	Meaning
r (R)	clock tree (set of clock trees)
ω (Ω)	global clock skew (maximum global skew constraint)
t (T)	clock latency (maximum clock latency constraint)
γ (Γ)	clock slew (maximum clock slew constraint)
C	maximum load capacitance constraint
L	clock tree wirelength
O	set of placement blockages
w (W)	width (width of given layout region)
h (H)	height (height of given layout region)
n (N)	number of sink regions (required number of sink regions)
p (P)	level index [p = 1 for clock source] (depth of clock tree)
b_p	branching factor at level p in clock tree
B	branching patterns (i.e., sequences of branching factors $\{b_1,\ b_2,\ ...,\ b_P\}$)
u_k	k^{th} sink cluster ($u_k \in U$)
s_i	i^{th} sink ($s_i \in S$)
$d_{k,i}$	Manhattan distance between sink s_i and root of cluster u_k
$\eta_{k,i}$	binary indicator of whether sink s_i belongs to cluster u_k
$\psi_{j,q}^{llx,lly,urx,ury}$	binary indicator whether j^{th} buffer is located outside the corresponding boundaries of q^{th} blockage
c_i	clock pin capacitance of sink s_i
x_i (y_i)	x-coordinate (y-coordinate) of sink s_i
$((x_k^{ll}, y_k^{ll}),\ (x_k^{ur}, y_k^{ur}))$	bounding box of sink cluster u_k

Given layout region area $W \times H$, we analyze the wirelength of a GH-tree with branching pattern (b_1, b_2, b_3, ..., b_P). We assume that (i) at any level, the region area is uniformly split into sink regions (i.e., regions that contain downstream sinks) according to the branching factor; (ii) the root of a sink region is located at the center of the sink region; (iii) branching factor b_p at any level p is always an even number; and (iv) the GH-tree always starts with a horizontal segment at the top level. Based on these assumptions, the wirelength of a horizontal (vertical) wire segment w_p (h_p) at level p is calculated as[73]

$$w_p = \frac{b_p - 1}{\Pi_{i=1}^{(p+1)/2} b_{2i-1}} \cdot W\ , \qquad h_p = \frac{b_p - 1}{\Pi_{i=1}^{p/2} b_{2i}} \cdot H \tag{4.14}$$

[73]Note that $(p+1)/2$ in Equation (4.14) is always an integer since we create horizontal segments (w_p) and vertical segments (h_p) in alternation, and always start with a horizontal segment from the top. Thus, all horizontal segments are created when p is an odd number.

The total wirelength is calculated as

$$L = \sum_{k=1}^{\lceil \frac{P}{2} \rceil} [\frac{b_{2k-1} - 1}{b_{2k-1}} \prod_{i=1}^{k-1} b_{2i}] \cdot W + \sum_{k=1}^{\lfloor \frac{P}{2} \rfloor} [\frac{b_{2k} - 1}{b_{2k}} \prod_{i=1}^{k} b_{2i-1}] \cdot H \tag{4.15}$$

Assuming a linear wire delay model and ignoring buffering, we derive the maximum and minimum (linear-delay) clock latency from clock source to any sink as

$$t_{max} = \frac{1}{2} \cdot [\sum_{k=1}^{\lceil \frac{P}{2} \rceil} (\frac{b_{2k-1} - 1}{\Pi_{i=1}^{k} b_{2i-1}}) \cdot W + \sum_{k=1}^{\lfloor \frac{P}{2} \rfloor} (\frac{b_{2k} - 1}{\Pi_{i=1}^{k} b_{2i}}) \cdot H] \tag{4.16}$$

$$t_{min} = \frac{1}{2} \cdot [\sum_{k=1}^{\lceil \frac{P}{2} \rceil} (\frac{1}{\Pi_{i=1}^{k} b_{2i-1}}) \cdot W + \sum_{k=1}^{\lfloor \frac{P}{2} \rfloor} (\frac{1}{\Pi_{i=1}^{k} b_{2i}}) \cdot H] \tag{4.17}$$

The maximum global skew ω is then defined as the difference between t_{max} and t_{min}.

$$\omega = \frac{1}{2} \cdot [\sum_{k=1}^{\lceil \frac{P}{2} \rceil} (\frac{b_{2k-1} - 2}{\Pi_{i=1}^{k} b_{2i-1}}) \cdot W + \sum_{k=1}^{\lfloor \frac{P}{2} \rfloor} (\frac{b_{2k} - 2}{\Pi_{i=1}^{k} b_{2i}}) \cdot H] \tag{4.18}$$

Figures 4.14(a) and 4.14(b) respectively show skew-wirelength and latency-wirelength tradeoffs in GH-trees with various branching patterns. We calculate skew and latency based on a linear delay model[74] according to Equations (4.16) and (4.18). The figures also show the Pareto frontier of non-dominated points of each tradeoff. Figures 4.14(c) and 4.14(d) respectively show skew-power and latency-power tradeoffs of *buffered* GH-trees with various branching patterns, as reported by a commercial static timing analysis tool with foundry 28LP technology and library models. The Pareto frontier of each tradeoff is shown as the black curve. We observe from the figures that different branching patterns lead to wide-ranging skew-power and/or latency-

[74]The linear wire delay model approximates clock latency with relatively low accuracy. However, we give this motivating analysis using the simple linear delay model to more intuitively illustrate the tradeoff among clock power, skew and latency with different GH-tree topologies. Below, we apply comprehensive buffer and wire delay modeling in our optimization to demonstrate the benefits of our proposed approach.

power tradeoffs. In this section, we explore branching patterns via dynamic programming to optimize tradeoffs among skew, latency and power. Figure 4.14(c) shows that with the same branching pattern (e.g., (10, 4, 2, 12)), different buffering solutions can lead to more than 20% skew difference with similar clock power. We therefore perform co-optimization of tree topology (i.e., branching pattern) and buffering to minimize clock power, skew and latency.

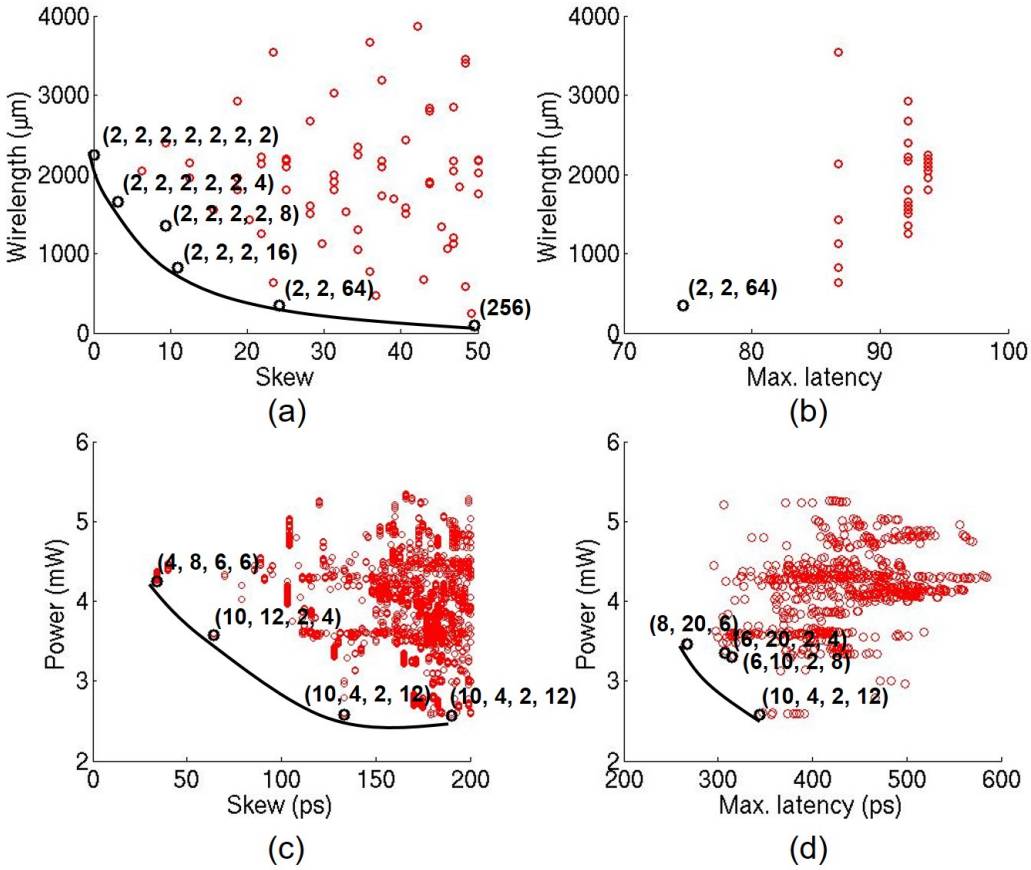


Figure 4.14: Study of motivating tradeoffs. (a) Linear delay skew vs. WL and (b) linear delay latency vs. WL with different branching patterns. (c) Skew vs. clock power and (d) maximum latency vs. clock power, in *buffered* GH-trees for a testcase with 17K sinks and region area = $380\mu m \times 380\mu m$.

4.3.2 Our Approach

We now describe our problem formulation and our approach. Based on the motivating examples shown in Figure 4.14, we construct GH-trees to explore the tradeoff among skew, latency and clock power. Our construction comprehends the delay and power impact of buffer insertion, sink placement and multiple constraints (e.g., maximum transition time and maximum

load capacitance). More specifically, we address the following **GH-tree construction problem:**
Given: a placement solution (i.e., a layout region $W \times H$ and placement of sinks), number of sink regions N such that each region contains < 40 sinks, timing library of clock buffers (.lib), maximum clock skew constraint Ω, maximum clock latency constraint T, maximum transition time constraint Γ (i.e., at both sinks and clock buffer input pins), and maximum load capacitance constraint C.
Perform: GH-tree construction with co-optimization of the clock tree topology (i.e., branching patterns) and buffering to **minimize** clock power, subject to the given constraints.

Figure 4.15 shows our overall GH-tree construction flow. An instance consists of a post-placement layout, the number of sink regions, and constraints, along with pre-characterized technology- and library-specific lookup tables (LUTs) containing power and delay information of candidate buffering solutions (i.e., segment length and buffer sizes). We perform the GH-tree construction primarily through two main steps: (i) according to the total sink capacitance and layout region area and aspect ratio, we first formulate a dynamic programming problem to co-optimize branching pattern and buffering; and (ii) we then perform balanced K-means clustering and formulate an integer linear programming problem to determine clock buffer placement (i.e., to embed our generalized GH-tree structure into the given sink placement). We note that the key step is the first step (i.e., DP-based co-optimization of clock topology and buffering) that systematically explores the continuum between H-tree and spine to achieve an optimal tradeoff among clock power, skew and latency (within this regime). Last, we realize our GH-tree solution in a commercial P&R tool and report metrics (e.g., skew, latency, power, etc.) to assess solution quality.

Although our approach systematically optimizes the tradeoff among clock power, latency and skew in the GH-tree regime, our method has two limitations that we highlight. First, we do not co-optimize tree topology and buffering together with sink placement, due to large runtime complexity. Second, our approach does not consider clock gating cells in a clock tree. Therefore, an improved resolution of the "chicken-and-egg" loop between (i) placement of roots of sink regions and (ii) top-level tree topology optimization, as well as consideration of clock gating cells during the optimization, remain as open research directions.

LUT Characterization

We characterize LUTs based on simulations using *Synopsys PrimeTime* [221] as inputs for our DP-based optimization. These LUTs contain power, input capacitance, slew propagation,

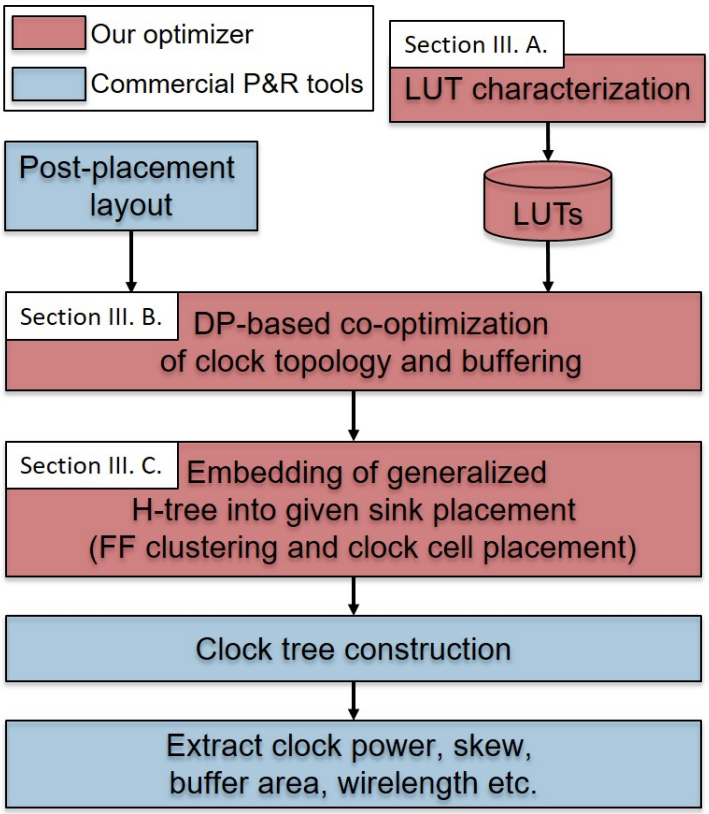


Figure 4.15: Overall flow of GH-tree construction.

and delay information of buffered and unbuffered wire segments. We use four types of buffers (X50, X67, X100, X134 from the 28LP libraries). In this technology, the gate area of a X134 buffer is $\sim 7\times$ the gate area of a minimum-size (X2) buffer. We also use "ganged buffers" (i.e., two, four or six X134 buffers with shorted inputs and shorted outputs) to achieve higher driving strengths. We create wire segments of lengths $15\mu m$, $30\mu m$, $45\mu m$, $60\mu m$, $75\mu m$, and $90\mu m$.[75] Along these wire segments, we enumerate all possible buffering solutions with the minimum granularity of $15\mu m$ (i.e., the minimum distance between two buffers is $15\mu m$).

The granularity of LUTs (e.g., number of buffer candidates, minimum wire segment length) will determine the tradeoff between optimization runtime and solution quality. In this section, we empirically select the granularity of our LUTs to achieve improved solution quality compared to two commercial tools, while using comparable runtime. For example, with a $45\mu m$ segment length, a minimum buffering distance of $15\mu m$ and seven buffer sizes, there are 8^3 (i.e.,

[75] We use LUTs based on multiple short wire segments to estimate the delay and slew propagation of a long wire segment. Since we match the output load, input capacitance as well as output and input slew values of two consecutive short segments to form a long segment, the estimation error is negligible. The small estimation error comes from discreteness of capacitance and slew values.

no buffer or exactly one of the seven buffer sizes, at each of the three buffering locations) distinct buffering solutions. Note that our LUTs include unbuffered solutions, i.e., pure-wire solutions.

We consider both 1W1S (single-width, single-spacing) and 2W2S (double-width, double-spacing – which we understand to be the common non-default routing rule (NDR) for clock distribution) wire segments in our characterization. We vary the input slew from $5ps$ to $60ps$ in steps of $5ps$ and we vary output load from $1fF$ to $150fF$ in steps of $1fF$ from $1fF$ to $5fF$, and in steps of $5fF$ from $5fF$ to $150fF$. For each 3-tuple of (distance, input slew, output load) we obtain the buffering solution (including input capacitance) and the output slew. From the large number of possible solutions, we prune solutions as follows. For each (distance, input capacitance, output slew, output load) 4-tuple, we select three delay values at the 10^{th}, 50^{th} and 90^{th} percentiles of the delay range, and then select the minimum-power solution for each of these three delay values. Figure 4.16 shows an example of our pruning on LUTs, in which we select minimum-power solutions with different output load values. Red (resp. blue) dots are the buffering solutions with output load = $75fF$ (resp. $35fF$). Cross (x) points are the selected buffering solutions. In practice, we have found that this pruning reduces the number of buffering solutions by $\sim$94% at the cost of only $\sim$5% solution quality (i.e., in terms of skew or latency) loss.

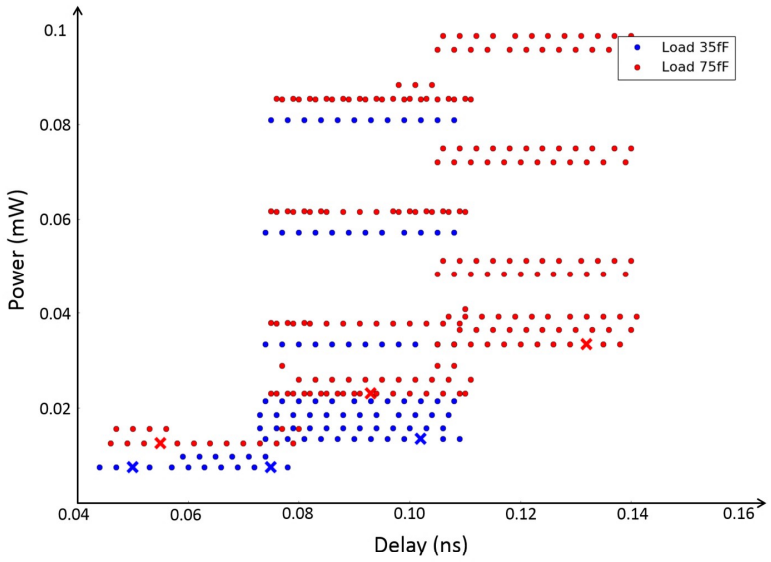


Figure 4.16: Example of pruning for buffering solutions with distance = $45\mu m$ and output slew = $35ps$.

DP-Based Co-optimization of Clock Topology and Buffering

Based on the characterized LUTs, we determine the optimal branching pattern along with the buffering solution for GH-tree using dynamic programming (DP). Other inputs to our optimization are layout region, placement of sinks, the number of sink regions, and the maximum skew and maximum latency constraints. A sink region typically contains < 40 sinks in our optimization. Our objective is to minimize the clock power while satisfying the given maximum skew and maximum latency constraints. As discussed, in this step, we assume that the sink regions are induced from a uniform placement of sinks, and that branching points are always located at the center of the corresponding sink region. We understand that the sink regions are typically not uniform for a real placement solution. However, due to high runtime complexity, it is practically infeasible for our current approach to consider sink placement during our DP-based optimization. We therefore assume uniform sink regions during our DP-based GH-tree construction. We then embed our DP-based solution (without solution quality degradation) into the given (real) sink placement based on sink clustering and branching point displacement. We formulate our DP in a high-dimensional solution space with seven dimensions (i.e., with respect to seven essential parameters of a clock tree optimization) and construct our GH-tree in a bottom-up way. The seven dimensions are $\{$clock tree depth (P), region area (width (w) and height (h)), number of sink regions (n), maximum and minimum clock latencies (t_{max} and t_{min}), and input capacitance (i.e., the load capacitance seen from the root)$\}$.

Algorithm 13 describes our optimization procedure; see also the illustration in Figure 4.17. We first construct GH-trees for the base case, that is, trees with depth $= 1$ over all different region areas (i.e., $w \times h$) and numbers of sink regions (i.e., n) (Line 1). As an example, Figure 4.17 shows the solutions at level p (i.e., subtree with depth $= 1$) with different region areas (i.e., 5×5 (red), 10×10 (green) and 15×15 (purple)). Procedure $build\_base\_trees(w, h, n, \emptyset)$ constructs GH-trees with depths of one (i.e., *spines* with different buffering solutions) within a $w \times h$ region and with a branching factor of b_p. Following [179], we use the term *spine* to denote one horizontal or vertical wire segment in the clock tree. Note that at the bottom level, $b_p = n$. As illustrated in Figure 4.17, we optimize the buffering solution along the spine based on characterized LUTs. Optimization of each tree segment along the spine generates a minimum-power Pareto surface in the high-dimensional space indexed by the LUT input parameters (e.g., maximum and minimum latencies, and input capacitance). The optimization eventually results in multiple subtree solutions. We store these subtree solutions in a set R indexed by tree depth, region area, number of sink regions, maximum and minimum clock latencies, and input capaci-

tance. In other words, R is a set of subtree solutions along with their depth, region area, number of sink regions, clock latency, input capacitance and power information corresponding to the minimum-power Pareto surface.

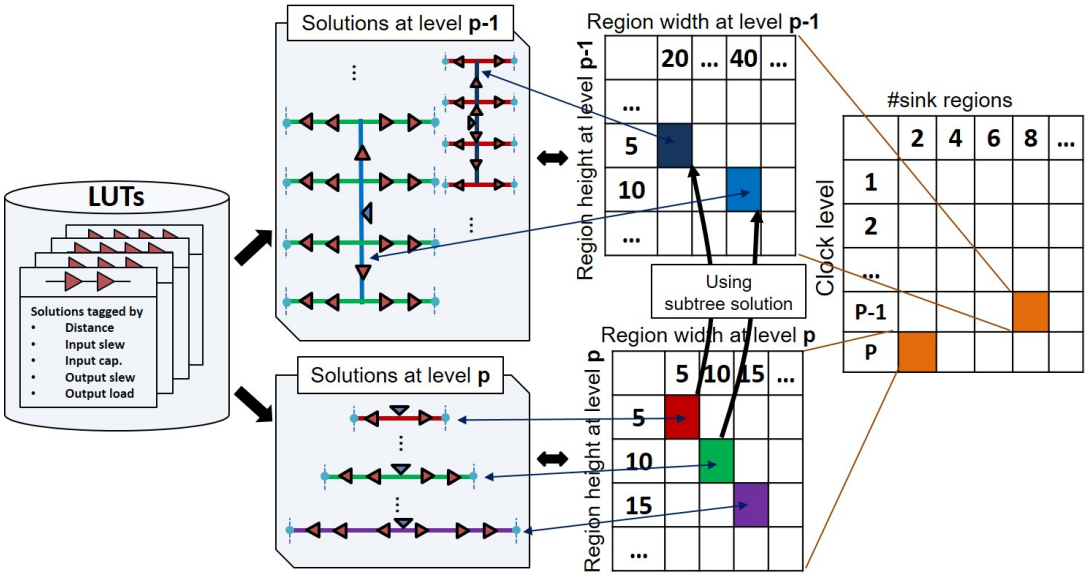


Figure 4.17: Co-optimization of GH-tree topology and buffering. The example illustrates construction of trees with depth $=$ 2 and eight sink regions, based on subtrees with depth $=$ 1 and two sink regions.

Next, we recursively search for the optimal (i.e., minimum-power) GH-tree solutions with depth $P > 1$, region area $w \times h$, and number of sink regions n. We increase P by one per iteration during the optimization, until $P = P_{max}$ (Lines 2–20). The maximum depth P_{max} for a given N is estimated based on the conventional H-tree (which has branching factor of two for all levels), i.e., $P_{max} = \lceil \log_2 N \rceil$. For each depth P, we perform buffering optimization along the tree segments at the topmost level, based on the stored subtree solutions at depth $(P - 1)$. In other words, we use existing solutions (i.e., subtree solutions from R) and add one more (topmost) level with optimized buffering to construct a new tree with depth increased by one. Figure 4.17 illustrates how our optimizer constructs solutions at level $(p - 1)$ (i.e., subtrees with depth $=$ 2) based on the solutions at level p (i.e., subtrees with depth $=$ 1). In this example, solutions for region area 20×5 (resp. 40×10) at level $(p - 1)$ are constructed based on four instantiated solutions for region area 5×5 (resp. 10×10) at level p.

Algorithm 13 DP-based GH-tree construction.

1: $R \leftarrow build\_base\_trees(w, h, n, \emptyset),\ \forall\, w, h, n$
$s.t.\ 0 < w \leq W,\ 0 < h \leq H,\ 2 \leq n \leq N,\ n$ is an even number
2: **for** $P := 2$ to P_{max} **do**
3: **for** $w := 0$ to W **do**
4: **for** $h := 0$ to H **do**
5: **for** $n := 2$ to $(N \cdot w \cdot h)/(W \cdot H)$ **do**
6: $R \leftarrow retrieve\_subtrees(P_l, w_l, h_l, n_l)$
7: **for all** $(r_l) \in R$ **do**
8: $r \leftarrow build\_tree(w, h, n, r_l)$
9: $r' \leftarrow retrieve\_tree(R, P, w, h, n, r.t_{max}, r.t_{min})$
10: **if** r' = **null then**
11: $R \leftarrow R \cup \{r\}$
12: **else if** $r.power < r'.power$ **then**
13: remove r' from R
14: $R \leftarrow R \cup \{r\}$
15: **end if**
16: **end for**
17: **end for**
18: **end for**
19: **end for**
20: **end for**
21: $r_{opt}.power \leftarrow \infty$
22: **for all** $r' \in R$ s.t. $r'.w = W,\ r'.h = H,\ r'.n \geq N$ **do**
23: **if** $r'.t_{max} - r'.t_{min} \leq \Omega$ && $r'.t_{max} \leq T$ && $r'.power < r_{opt}.power$ **then**
24: $r_{opt} \leftarrow r'$
25: **end if**
26: **end for**
27: **return** r_{opt}

For each (P, w, h, n) tuple, we construct our optimization solutions based on the set of all stored subtrees (r_l) (from R) that satisfy

$$P_l = P - 1;\ w_l = h;\ h_l = w/b_t;\ n_l = n/b_t \tag{4.19}$$

where P_l is the depth of r_l; $w \times h$ is the dimension of the layout region; $w_l \times h_l$ is the dimension of a sink region (i.e., layout region for subtree r_l); b_t is the branching factor at the topmost level of the current tree; and n_l is the number of sink regions of r_l. Lines 3-4 and Line 5 respectively enumerate possible dimensions and numbers of sink regions for subtree solutions. In Line 6 of Algorithm 13, we find all subtree solutions, which we have optimized in previous iterations, with branching factor b_t such that $2 \leq b_t \leq n/2$.

Procedure $build\_tree(w, h, n, r_l)$ then builds trees r with depth = P, using copies of the collected subtrees $r_l \in R$ (which have depth = $(P - 1)$) as its lower-level subtrees (Line 8).

In other words, we build the tree segment with optimized buffering at the topmost level, and at each sink of the topmost segment, we use (i.e., instantiate) the same subtree r_l to build lower levels of the tree r. To reduce the runtime complexity, our current approach assumes that at any level, the subtrees are identical. As shown in Section 4.3.3 below, this does not preclude strong final solution quality. Among all the constructed trees with depth = P and the same maximum and minimum latency, we select the solution with minimum power and add it to the solution set R (Lines 7–16). Procedure $retrieve\_tree(R, P, w, h, n, r.t_{max}, r.t_{min})$ in Line 9 retrieves a previously stored solution r' from the set R that satisfies the conditions depth = P, width = w, height = h and number of sink regions = n, and has maximum and minimum latencies equal to specified t_{max} and t_{min}, where t is clock latency. Finally, we select the solution with minimum power that satisfies the maximum skew constraint (Ω) and the maximum latency constraint (T) from set R with number of sinks N and region area $W \times H$ (Lines 21–27).

We note that the slews are propagated from top to bottom in a tree. However, our optimization performs bottom-to-top GH-tree construction. We propagate slew bottom-up to accurately capture the slew degradation and avoid maximum transition violations. We first assume several slew values (e.g., 25ps, 30ps, 35ps, 40ps) at the root of each sink region. For each of the assumed slew values, we propagate slew bottom-up, based on LUTs (note that our LUTs contain output and input slews for each buffering and/or wiring solution). During the DP-based optimization, we only select solutions which ensure that slew values throughout the slew propagation are always within the range of [5ps, 60ps], where 60ps is the maximum slew constraint in our experiments, and 5ps is the minimum achievable slew in practice in our experiments. We also note that buffer locations are determined by the selected LUT solution with bottom-up slew propagation. Thus, buffers are not necessarily inserted in all branching points.

The runtime complexity of proposed algorithm is $O(P^{max} \cdot \frac{W \cdot H}{w_{int} \cdot h_{int}} \cdot N^2 \cdot \frac{\gamma_{max}}{\gamma_{int}})$, where w_{int}, h_{int} and γ_{int} are respectively the minimum distance and timing intervals for discretization of the original continuous solution space to formulate the dynamic programming; γ_{max} is the specified maximum clock latency constraint. We set w_{int}, $h_{int} < 5\mu m$ and $\gamma_{int} = 1ps$ in our experiments. To reduce the runtime, we apply the following pruning techniques.

- **Pruning with number of leaf regions.** For a given sub-region of size $w \times h$ we prune solutions with number of leaf regions greater than $\frac{N \cdot w \cdot h}{W \cdot H}$.
- **Pruning with skew/latency constraints.** We prune solutions that have skew larger than the maximum skew constraint or maximum latency larger than the latency upper bound.

- **Pruning with maximum fanout constraint.** We prune solutions that have branching factor larger than the maximum fanout constraint.[76]

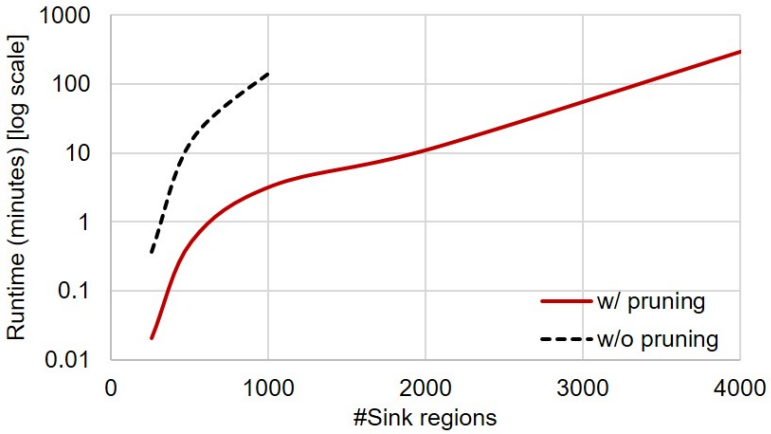


Figure 4.18: Runtime of our DP-based optimization method with and without pruning techniques across different numbers of sink regions.

Figure 4.18 shows DP-based optimization runtime with the number of sink regions ranging from 200 to 4000, where each sink region contains $\sim$25 flip-flops. The maximum skew constraint used in the experiment is $30ps$. Results show that with pruning, the DP-based optimization can optimize a design with more than 4K sink regions (or 100K flip-flops) within six hours. Assuming that the flip-flop count to total instance count ratio is typically 10% to 25%, our approach can optimize a design with 1M instances within six hours. Our studies show that runtime and memory usage increase significantly if we do not apply the proposed pruning techniques, due to the large number of intermediate solutions (such that we are not able to optimize beyond a design with 1K sink regions due to excessive memory usage). Moreover, we observe same solution quality between the runs with and without pruning.

Embedding of GH-Tree Into a Sink Placement

The clock tree topology and buffering solution from our DP-based optimization assumes a uniform sink (region) distribution (whereby branching points are at the centers of regions). However, given a (realistic) non-uniform sink (flip-flop) placement, we must cluster flip-flops with balanced load across different clusters to avoid skew and latency increase. In other words, we should assign clusters of flip-flops to sinks of the GH-tree, based on the actual sink flip-flop placement. To adapt our optimized GH-tree to the given sink (i.e., flip-flop) placement,

[76] In our experiments, we set the maximum fanout constraint to 40 based on guidance from an industrial collaborator [95].

we perform a balanced K-means clustering of sinks and adapt buffer placements based on the clustering solution.[77] We note that our approach is different from conventional top-down clock tree construction methods (e.g., Planar-DME [111]), in that (i) we embed an optimized clock tree topology with buffering solutions to a layout region with given sink placements, and (ii) we balance load capacitance among sink regions. By maintaining the distances between consecutive branching points and buffers at each clock level as well as balancing the load capacitance among sink regions, we preserve the solution quality (i.e., skew and latency) of the GH-tree solution obtained by the DP-based optimization. Furthermore, we understand that the optimal GH-tree topology and buffering solution can vary across different sink placements. With this in mind, we keep the best M solutions from our DP-based optimization and select the minimum-power solution for the given (actual) sink placement as our final solution. Based on our preliminary experiments, we empirically use $M = 5$ to generate the results reported below, where increasing M beyond 5 will not improve the solution quality significantly.

The remainder of this subsection describes two mathematical programs (ILPs) that perform sink clustering (i.e., to assign flip-flops to sinks of the constructed GH-tree) and place buffers of our GH-tree (one example is shown in Figure 4.19). The two ILPs respectively act at global and local clustering, as we describe below. We perform the clustering and branching point placement top-down, level-by-level. Our clustering optimization balances the load capacitance across different clusters (each cluster is assigned to a sink of our GH-tree) to minimize the discrepancy between our DP solution (which assumes a uniform sink placement) and the final solution (with the given actual sink placement). Note that this implies that we must consider wire capacitance at the bottom-most level, where the routing is achieved by a commercial P&R tool.[78] However, any constructive approach to this wire capacitance estimation is inaccurate and can dramatically increase runtime complexity during top-level optimization where each branching point can have many sink flip-flops. We therefore divide the GH-tree into global and local clustering, based on the total number of downstream sinks (i.e., flip-flops), and only consider wire capacitance during local clustering.

Algorithm 14 describes our clustering procedure. For each level p, we iteratively apply either global or local clustering followed by LP-based branching point and clock buffer placement. Starting from level 1 (the topmost level) of the tree, we cluster sinks based on initial locations of the branching points (i.e., the light blue dots in the top-left figure of Figure 4.19, which

[77] Conventional K-means clock tree synthesis [45] cannot be applied to our problem as it does not comprehend the load capacitance balancing criterion.

[78] Different routing tools can have different clock routing solutions for the bottom-level clock tree. However, the difference is very small. Based on our experimental results, skew from bottom-level clock routing is $< 1ps$.

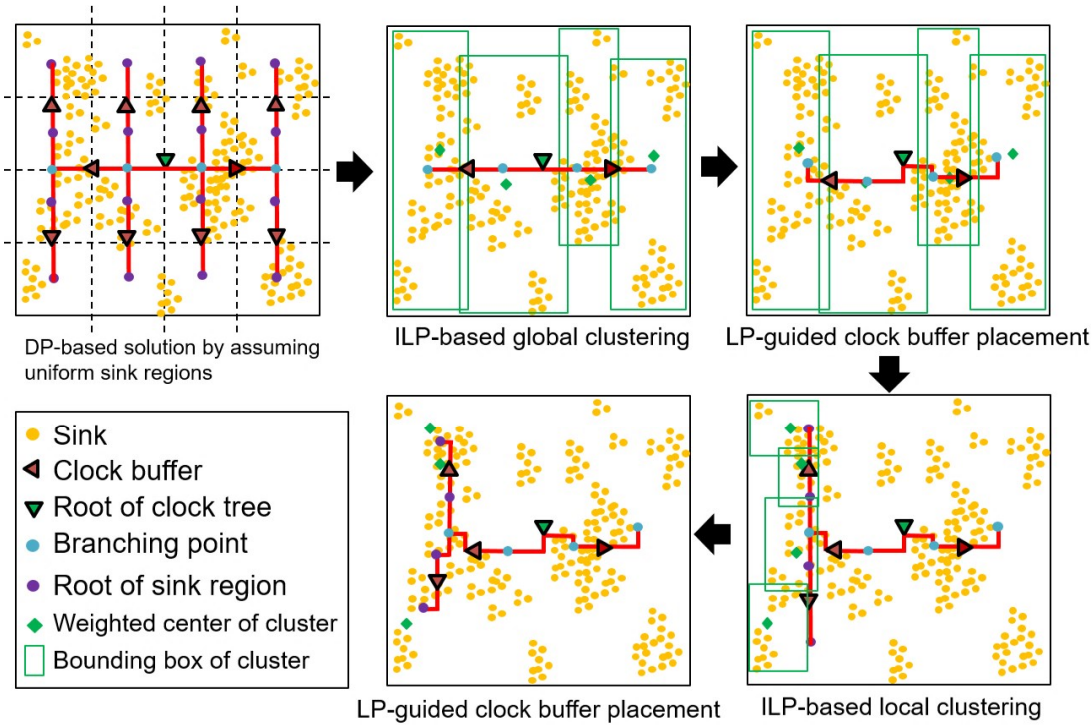


Figure 4.19: Example of sink clustering and clock buffer placement. In this example, we only show the local clustering for the leftmost branch of the first-level clock tree.

are at the centers of uniform sink regions) of the DP-based GH-tree solution (Algorithm 13). The number of global clusters is the same as the number of branching points. For an example in Figure 4.19, since $p = 1$ and $b_1 = 4$, our ILP will generate four global clusters that have similar load capacitance. We then formulate an LP to determine the exact branching point locations as well as buffer locations in each global cluster. When the number of sinks in each cluster is smaller than a threshold value (i.e., $|S|/|U| < Q_{th}$), we apply our ILP-based local clustering, and refine the branching point locations in each local cluster using our LP. Note that the "K" in the K-means clustering is determined by the number of branching points in the GH-tree.

ILP formulation for global clustering. We pre-calculate distance $d_{k,i}$ between the branching point of cluster u_k and the sink s_i within the bounding box of the region that will be clustered. The initial locations of branching points are assumed to be at the centers of uniformly-sized regions corresponding to the branching factor. The blue dots in the top-left figure of Figure 4.19 show an example of initial branching point locations when the branching factor $b_1 = 4$.

Algorithm 14 Embedding of GH-Tree into a sink placement.

```
for p := 1 to P do
    if |S|/|U| ≤ Q_th then
        global_clustering()
    else if r.power < r'.power then
        local_clustering()
    end if
    branching_point_and_buffer_placement()
end for
```

$$\text{Minimize:} \quad \sum_{s_i \in S} d_i + \alpha \cdot d^{max}$$

Subject to:

$$\sum_{u_k \in U} \eta_{k,i} = 1 \qquad \forall s_i \in S \tag{4.20}$$

$$d_i = \sum_{u_k \in U} d_{k,i} \cdot \eta_{k,i} \qquad \forall s_i \in S \tag{4.21}$$

$$d^{max} \geq d_i \qquad \forall s_i \in S \tag{4.22}$$

$$C^{pin} \cdot (1 - \Delta) \leq \sum_{s_i \in S} c_i \cdot \eta_{k,i} \leq C^{pin} \cdot (1 + \Delta) \qquad \forall u_k \in U \tag{4.23}$$

We define d_i as the distance between the sink s_i and the branching point of the cluster that includes the sink s_i; d^{max} denotes the maximum distance among the distances d_i; and α is a weighting factor.[79] Our objective is to minimize the sum of all distances d_i and weighted d^{max}. Constraint (4.20) ensures that each sink can only belong to exactly one cluster. In Constraints (4.21) and (4.22), we obtain d_i for each sink and d^{max}, respectively. Constraint (4.23) ensures that the total pin capacitance of each cluster satisfies specified lower and upper bounds. The lower and upper bound capacitances are determined by C^{pin}, which is estimated as the total pin capacitance covered by the current region divided by the number of clusters, along with the margin Δ. Since the capacitance cannot be always balanced between clusters, we add margin Δ to ensure that there is a feasible solution of the ILP.

[79] Based on our preliminary studies, we empirically use $\alpha = 8$ in our experiments.

ILP formulation for local clustering. Although the ILP for global clustering finds a balanced pin capacitance solution over all sink regions, it ignores wire capacitance. Therefore, we formulate a second ILP and apply it to local clusters that have smaller regions.

$$\text{Minimize:} \quad \sum_{s_i \in S} d_i + \sum_{u_k \in U} \frac{\alpha}{|U|} \cdot (x_k^{ur} - x_k^{ll} + y_k^{ur} - y_k^{ll})$$

Subject to:

$$x_k^{ur} \geq x_i \cdot \eta_{k,i} \qquad \forall s_i \in S,\ u_k \in U \tag{4.24}$$

$$y_k^{ur} \geq y_i \cdot \eta_{k,i} \qquad \forall s_i \in S,\ u_k \in U \tag{4.25}$$

$$x_k^{ll} \leq x_i + \lambda \cdot (1 - \eta_{k,i}) \qquad \forall s_i \in S,\ u_k \in U \tag{4.26}$$

$$y_k^{ll} \leq y_i + \lambda \cdot (1 - \eta_{k,i}) \qquad \forall s_i \in S,\ u_k \in U \tag{4.27}$$

$$C^{pin+wire} \cdot (1 - \Delta) \leq \sum_{s_i \in S} (c_i + \zeta) \cdot \eta_{k,i} + \beta \cdot (x_k^{ur} - x_k^{ll} + y_k^{ur} - y_k^{ll}) \tag{4.28}$$

$$C^{pin+wire} \cdot (1 + \Delta) \geq \sum_{s_i \in S} (c_i + \zeta) \cdot \eta_{k,i} + \beta \cdot (x_k^{ur} - x_k^{ll} + y_k^{ur} - y_k^{ll}) \tag{4.29}$$

+ Constraints (4.20) and (4.21)

The objective of this second ILP is to minimize the sum of all distances d_i, of which the definition is the same as above, plus the weighted sum of half-perimeter wirelength (HPWL) of all clusters' bounding boxes. We use HPWL and the number of sinks within a cluster to model the wire capacitance of the sink region. In Constraints (4.24)–(4.27), we obtain the lower-left and upper-right corner locations of each cluster's bounding box. λ is a large positive integer. Constraints (4.28) and (4.29) ensure that total (i.e., pin and wire) capacitance of each cluster satisfy given lower and upper bounds. ζ and β are respectively coefficients for the number of sinks and the cluster's bounding box HPWL in our linear wire capacitance estimation model.[80]

[80] We determine ζ and β by fitting a linear model to wire capacitances extracted for all our testcases. We perform least-squares regression as follows: wire capacitance = ζ· #sinks + β· HPWL. In our $28nm$ foundry enablement, $\zeta = 0.28$ and $\beta = 2.93$.

LP formulation for branching point location. Based on the two ILPs described above, we obtain the clustering solution at a given clock level. However, the initial assumption of branching locations to be at the center of a region may cause large skew if the sinks within a cluster are placed non-uniformly. To address this issue, we formulate a linear program (LP) to place each branching point close to the weighted center of each cluster, as follows.

Minimize: d^{Δ}_{max}

Subject to:

$$|x^b_{k+1} - x^b_k| + |y^b_{k+1} - y^b_k| = d^b \tag{4.30}$$

$$x^{\Delta}_k + x^b_k - x^w_k \geq 0, \;\; x^{\Delta}_k - x^b_k + x^w_k \geq 0 \tag{4.31}$$

$$y^{\Delta}_k + y^b_k - y^w_k \geq 0, \;\; y^{\Delta}_k - y^b_k + y^w_k \geq 0 \tag{4.32}$$

$$x^{\Delta}_{max} \geq x^{\Delta}_k, \;\; y^{\Delta}_{max} \geq y^{\Delta}_k \tag{4.33}$$

$$d^{\Delta}_{max} = x^{\Delta}_{max} + y^{\Delta}_{max} \tag{4.34}$$

Here, x^b_k and y^b_k denote the x-/y-coordinates of the k^{th} branching point, and x^w_k and y^w_k denote the x-/y-coordinates of the weighted center of the k^{th} cluster.[81] The objective is to minimize the sum of the maximum x- and y-distances between any branching point and its corresponding weighted center. Constraint (4.30) ensures a fixed distance d^b between any two consecutive branching points. In Constraints (4.31) and (4.32), variables x^{Δ}_k and y^{Δ}_k respectively denote the x- and y-distances between the branching point and the weighted center of the k^{th} cluster. We then obtain the sum of the maximum x- and y-distances from Constraints (4.33) and (4.34).

Placement blockage. We further show a simple extension of our LP formulation to an ILP formulation to comprehend *blockage-aware* buffer placement. In this extension, we adjust buffer placement to address the existence of blockages. A caveat is that this method may not work well for designs with a large number of blockages and/or a complex floorplan, since our DP-based tree topology and buffering solutions are not aware of blockages. We assume that there are O rectangular blockages. The index of a blockage is denoted by q. Each blockage is defined by its lower-left corner (x^{ll}_q, y^{ll}_q) and upper-right corner (x^{ur}_q, y^{ur}_q). When there are placement blockages in the floorplan, we define the following constraints.

[81] We calculate the x- and y-coordinates of each weighted center as the respective means of all x- and y-coordinates of placed sinks in the corresponding cluster.

$$
\begin{aligned}
&\psi_{j,q}^{llx} = 1 \Leftrightarrow x_j^f < x_q^{ll} \\
&\psi_{j,q}^{urx} = 1 \Leftrightarrow x_j^f > x_q^{ur} \\
&\psi_{j,q}^{lly} = 1 \Leftrightarrow y_j^f < y_q^{ll} \\
&\psi_{j,q}^{ury} = 1 \Leftrightarrow y_j^f > y_q^{ur} \\
&\psi_{j,q}^{llx} + \psi_{j,q}^{urx} + \psi_{j,q}^{lly} + \psi_{j,q}^{ury} \geq 1
\end{aligned}
\tag{4.35}
$$

Here, $(x_j^f,\ y_j^f)$ is the location of the j^{th} buffer at a given clock level. $\psi_{j,q}^{\{llx,lly,urx,ury\}}$ are binary indicator variables which indicate whether the j^{th} buffer is located outside the corresponding boundaries of the blockages. The last inequality in Constraints (4.35) defines the constraint that at least one of the indicator variables must be true. Satisfying this constraint implies that the j^{th} buffer is not in the bounding box of the q^{th} blockage. Note that with the Constraints (4.35), the problem becomes an integer linear program (ILP).

4.3.3 Experimental Setup and Results

We conduct our experiments in a commercial foundry's $28nm$ LP technology, with a dual-Vt, 12-track standard-cell library. The input placement solutions (including clock sink placements) are generated using *Cadence Innovus Implementation System v15.2* [198]. Our optimization flow is implemented using C++ and Tcl scripts. We use *CPLEX v12.6* [203] as both ILP solver and LP solver, along with *OpenMP* [213] to enable multi-threaded execution. We execute all our experiments by using up to 40 threads on a $2.6GHz$ Intel Xeon E5-2690 server. We construct reference clock tree solutions using the latest releases available to us of two leading-edge commercial P&R tools (i.e., Tool1 and Tool2) as well as a state-of-the-art academic tool [118], and report attributes of solutions from these tools along with those of our GH-tree solutions.[82]

We evaluate our optimizer using four designs: *JPEG* from *OpenCores* [212], and *B19*, *VGA* and *LEON3MP* from the ISPD-2012 contest [141]. We use the real design (*JPEG*) and the testcases from the ISPD-2012 contest (*B19*, *VGA*, *LEON3MP*) since they contain datapaths (in contrast to testcases from the ISPD-2010 contest, which do not contain datapath information), thus enabling comparison versus commercial tools. We synthesize these testcases using *Synopsys Design Compiler H-2013.03-SP3* [218]. Table 4.9 summarizes the instance count, number of clock sinks, placement utilization and timing constraints for our testcases. To study the im-

[82]The tool flows used in our work are based on latest versions of leading commercial EDA tools available through the respective vendors' university programs. More specific identification of tools and vendors is not permitted by the vendors.

pact of maximum skew and latency constraints on power, we run our optimizer with different maximum skew and latency constraints and collect discrete solution points showing skew-power and latency-power tradeoff. We obtain reference clock solutions with multi-corner multi-mode (MCMM) optimization; we define our mode and corner settings in Table 4.10. We also apply late and early deratings of 1.1 and 0.9 to model OCV effects. We use *Synopsys HSPICE G-2012.06-SP1* [219] to perform timing and power analysis.

Table 4.9: Summary of testcases.

Testcases	**#Instances**	**#Flip-flops**	**Utilization (%)**	**Max transition time (ps)**	**Max capacitance (fF)**
B19	39788	3086	73	60	80
JPEG	46937	4712	73	60	80
VGA	66226	17057	76	60	80
LEON3MP	463104	108817	74	60	80
VGA_blockage	65891	17057	61	60	80
VGV_high_AR	65124	17057	75	60	80

Table 4.10: MCMM settings and clock periods (ns) for our testcases.

	C1 = {SS, 0.9V, -40°C}	**C2 = {FF, 1.1V, -40°C}**
B19	1.5	1.0
JPEG	1.2	1.0
VGA	1.4	1.0
LEON3MP	1.6	1.0
VGA_blockage	1.4	1.0
VGA_high_AR	1.4	1.0

Comparison with Trees from State-of-the-art CTS

Figure 4.20 and Figure 4.21 respectively compare skew and clock power, and maximum latency and clock power, of GH-tree solutions to those from the two commercial tools and one academic flow [118]. Table 4.11 compares #buffers, buffer area, max (insertion) delay across corners, and wirelength among clock tree solutions as well as optimization runtime. All flows use the same sink placement solution as input. We apply the same setups (i.e., clock buffer cells (X50, X67, X100, X134 and ganged buffers), BEOL layers (M3 and M4), maximum transition constraints (60ps)) to our GH-tree construction and to both commercial and academic tool flows.

We sweep the maximum skew and maximum latency constraints on the four designs for the GH-tree constructions. Blue curves in the figures are skew-power and latency-power Pareto curves of our GH-tree solutions.[83]

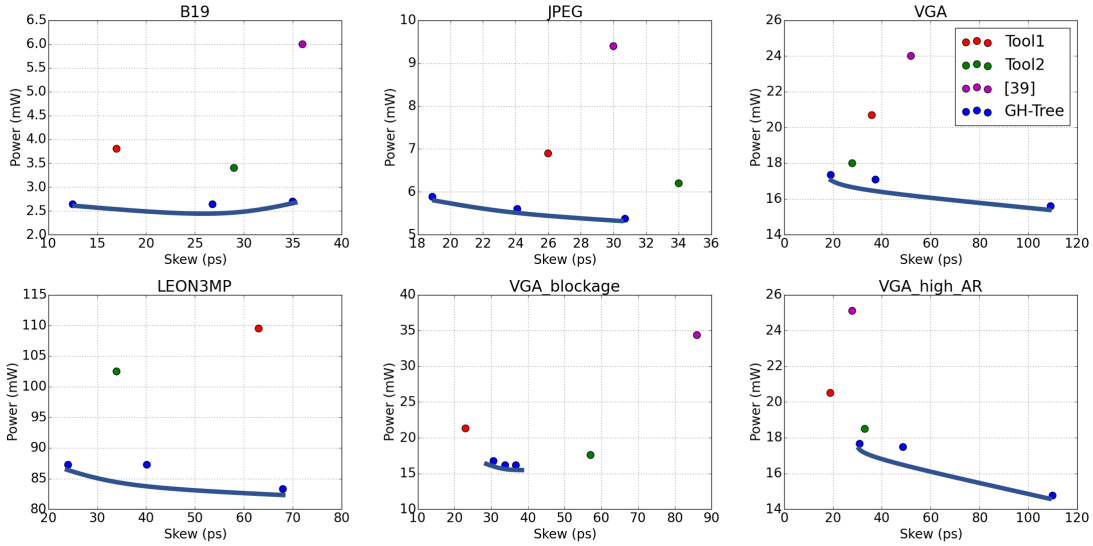


Figure 4.20: Power and skew comparisons among GH-tree, Tool1, Tool2 and [118] for four testcases.

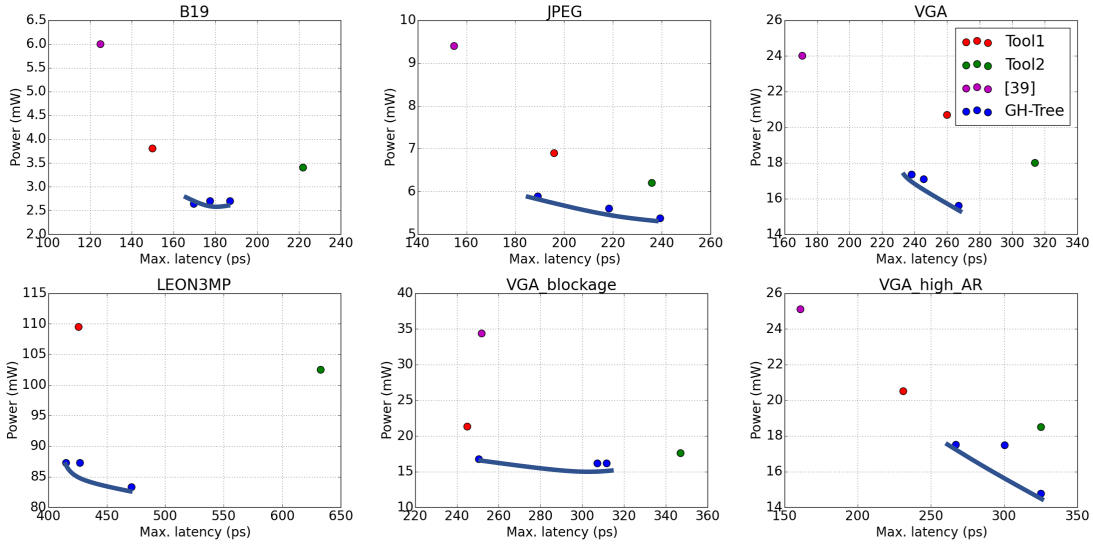


Figure 4.21: Power and maximum latency comparisons among GH-tree, Tool1, Tool2 and [118] for four testcases.

[83] We estimate the Pareto curve based on discrete solution points due to limited computing resources. In addition, since the tradeoff between skew/max latency versus clock power is monotone, we feel that three solution points can provide a useful estimation of the tradeoff.

Overall analysis. Our results show that our GH-tree solutions achieve significant clock power reduction with similar or reduced skew and latency values as compared to the solutions from both commercial and academic tools. We also include the conventional "strict" H-tree solution as a comparison. Note that we use the same methodology to determine the buffer locations and sizes in "strict" H-tree and GH-tree constructions. In other words, it is unnecessary to add a buffer at each branching point. For example, we achieve power reductions of 30% on *B19* and 20% on *LEON3MP* in Table 4.11 compared to commercial tools' results. Moreover, we observe that due to the symmetric topology of our GH-tree, our GH-tree solution is typically more robust against skew variation across different corners as compared to the clock trees from other tools. As an example, skew of the clock tree solution from Tool2 increases by 137% on design *VGA_blockage* between two corners; by contrast, our solutions generally have similar skew values across corners. The conventional ("strict") H-tree achieves the minimum clock skew but at the cost of larger power, buffer area and wirelength. As an example, for *LEON3MP*, H-tree has depth $P = 12$, but GH-tree has $P = 10$ (i.e., branching factor = (2, 2, 2, 2, 2, 4, 4, 2, 2, 2)). The shallower depth of GH-tree significantly reduces the number of buffers and wirelength. We also validate our GH-tree optimization on design *VGA* with high floorplan aspect ratio and existence of placement blockages (shown in Figure 4.27), where we observe similar clock power and latency, but larger skew, compared to the case without blockage and high floorplan aspect ratio. Compared to the results of [118], our GH-tree achieves much smaller power (i.e., up to 55% on *B19*) and skew, but at the cost of larger maximum latency.

Power analysis. We also observe that our GH-tree solutions have smaller number of buffers and clock wirelength as compared to solutions from commercial and academic tools. Figure 4.22 further shows a histogram of clock buffer power (i.e., sum of internal, leakage and dynamic power) values of our GH-tree versus corresponding values from a commercial tool's solution. We observe that our DP-based optimization, which can select its buffering solution "optimally" based on the characterized LUTs, achieves smaller buffer power values for most of the clock buffers. In other words, our GH-tree optimization is able to achieve optimized load capacitance of buffers as well as slew propagation for reduced clock power. The power information from our LUTs enables power-awareness in our DP-based GH-tree construction.

Runtime analysis. Results in Table 4.11 show that although the naive worst-case time complexity of our (DP- and ILP-based) optimization is high (cf. the nested **for** loops in Algorithm 13), the pruning techniques and empirically selected granularity of our LUTs (e.g., $15\mu m$ for distance, $5ps$ for slew, and $5fF$ for capacitance) make the runtime of our optimization com-

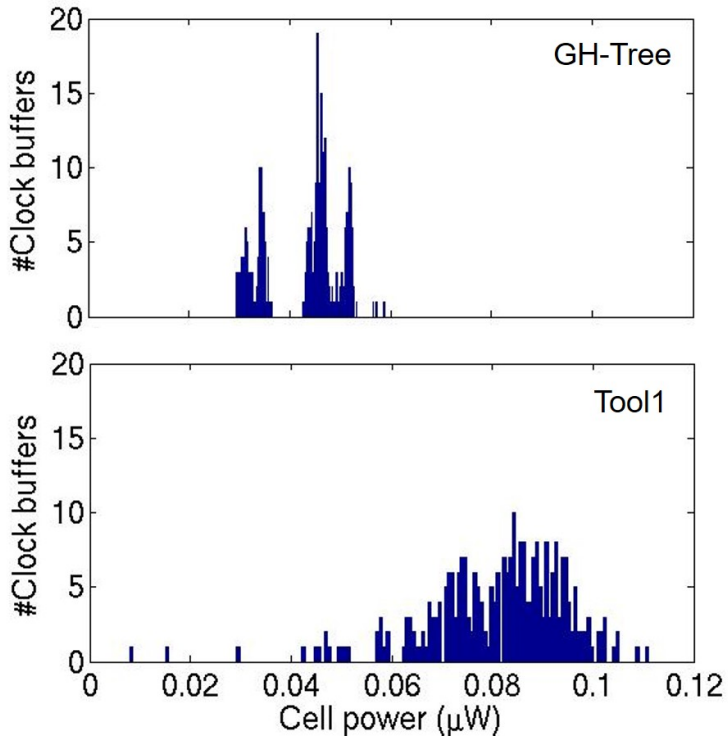


Figure 4.22: Distribution of clock buffer power values from GH-tree and commercial tool solutions. Design: *VGA*.

parable to those of commercial tools. The large runtime of design *LEON3MP* mainly comes from bottom-level tree construction ($\sim$40 minutes) and ECO routing according to our GH-tree solution using OpenAccess [217] ($\sim$15 minutes). The actual GH-tree construction runtime (i.e., DP + ILP runtime) for design *LEON3MP* is only $\sim$25 minutes. We understand that such runtime is very acceptable in light of the potential clock power benefits from our approach.

Robustness analysis. We further perform Monte Carlo simulation on our GH-tree solution and those from commercial tools, and compare the resultant variation in clock skew and power. Figure 4.23 shows that our GH-tree solution exhibits relatively smaller variation in clock skew (i.e., $\sim 35ps$) compared to commercial tools' solutions (i.e., $\sim 40ps$). Furthermore, all of Tool1, Tool2 and GH-tree solutions have small power variation.

Impact of NDR. We now summarize observed impacts of non-default rules (NDRs) on clock tree solution quality. We generate GH-trees with various NDR options: (i) 1W1S only, (ii) 2W2S only, and (iii) the combination of 1W1S and 2W2S. We set the maximum skew constraint to $200ps$ and compare Pareto curves of the latency versus clock power tradeoffs. Figure 4.24 shows the comparison for the *JPEG* testcase at 28LP technology. Due to better slew propagation, solutions with 2W2S have fewer clock buffers as compared to the 1W1S solutions

(i.e., the average numbers of clock buffers are respectively 83 and 111 in 2W2S- and 1W1S-only solutions). Further, solutions that permit either 1W1S or 2W2S at each level (of clock subnets) are able to achieve a better tradeoff between latency and power.

Table 4.11: Comparison between clock tree solutions from [118], Tool1 and Tool2 versus our GH-trees. Technology: 28LP.

Testcase	**Flow**	**Corner = *C1***			**Corner = *C2***			**#Buffers**	**Buf area**	**Clk WL**	**Runtime**
		Max laten. (ps)	**Skew** (ps)	**Clk power** (mW)	**Max laten.** (ps)	**Skew** (ps)	**Clk power** (mW)		(μm^2)	(mm)	(min)
B19	Tool1	**150**	**17**	**3.8**	110	**27**	9.3	104	227	15688	15
	Tool2	**222**	**29**	**3.4**	129	**5**	8.3	84	245	12996	11
	[118] (min_pwr)	**111**	**40**	**5.6**	63	**40**	12.1	211	338	N/A	N/A
	[118] (min_skew)	**125**	**36**	**6.0**	78	**38**	13.0	228	413	N/A	N/A
	GH-tree	**170**	**12.5**	**2.6**	116	**25.8**	6.4	41	106	12242	15
	H-tree	**166**	**7**	**3.1**	106	**11**	7.6	147	227	13941	16
JPEG	Tool1	**196**	**26**	**6.9**	131	**26**	16.8	160	345	20967	16
	Tool2	**236**	**34**	**6.2**	141	**18**	15.2	120	352	18432	14
	[118] (min_pwr)	**179**	**65**	**9.2**	103	**73**	19.7	340	651	N/A	N/A
	[118] (min_skew)	**155**	**30**	**9.4**	92	**36**	20.4	353	676	N/A	N/A
	GH-tree	**201**	**19**	**5.9**	129	**17**	14.5	147	296	20009	17
	H-tree	**229**	**12**	**6.6**	150	**16**	16.3	169	456	20064	14
VGA	Tool1	**260**	**36**	**20.7**	152	**7**	55.3	464	1119	57678	16
	Tool2	**314**	**28**	**18.0**	201	**11**	48.5	369	1047	56305	22
	[118] (min_pwr)	**171**	**52**	**24.0**	114	**73**	52.1	911	1651	N/A	N/A
	[118] (min_skew)	**171**	**52**	**24.0**	114	**73**	52.1	911	1651	N/A	N/A
	GH-tree	**238**	**19**	**17.4**	174	**41**	44.2	331	1036	57404	21
	H-tree	**253**	**16**	**20.4**	162	**19**	55.0	597	1682	62957	16
LEON3MP	Tool1	**426**	**63**	**109.5**	276	**25**	195.9	2661	6654	369737	54
	Tool2	**633**	**34**	**102.5**	421	**58**	184.2	2509	7225	367854	37
	[118] (min_pwr)	**N/A**	**N/A**	**N/A**	N/A	**N/A**	N/A	N/A	N/A	N/A	N/A
	[118] (min_skew)	**N/A**	**N/A**	**N/A**	N/A	**N/A**	N/A	N/A	N/A	N/A	N/A
	GH-tree	**415**	**24**	**87.2**	285	**35**	157.3	1331	4154	374568	101
	H-tree	**415**	**22**	**96.0**	287	**24**	173.0	2399	6741	393582	99
VGA_blockage	Tool1	**245**	**23**	**21.3**	174	**36**	56.7	475	1148	66323	14
	Tool2	**347**	**57**	**17.6**	212	**24**	47.5	401	1127	64640	24
	[118] (min_pwr)	**298**	**133**	**29.9**	208	**145**	54.5	1118	2154	N/A	N/A
	[118] (min_skew)	**252**	**86**	**34.4**	157	**93**	74.3	1291	2387	N/A	N/A
	GH-tree	**239**	**36**	**16.9**	163	**31**	45.4	293	815	68635	19
	H-tree	**282**	**22**	**20.8**	174	**21**	56.0	599	1685	72636	16
VGA_high_AR	Tool1	**231**	**19**	**20.5**	164	**27**	54.7	456	1094	59506	14
	Tool2	**325**	**33**	**18.5**	211	**43**	49.8	395	1120	58114	21
	[118] (min_pwr)	**161**	**28**	**25.1**	105	**74**	54.5	956	1679	N/A	N/A
	[118] (min_pwr)	**161**	**28**	**25.1**	157	**93**	54.5	956	1679	N/A	N/A
	GH-tree	**265**	**33**	**17.5**	187	**30**	47.3	299	1039	57855	21
	H-tree	**271**	**15**	**20.4**	169	**12**	54.8	661	1669	65181	22

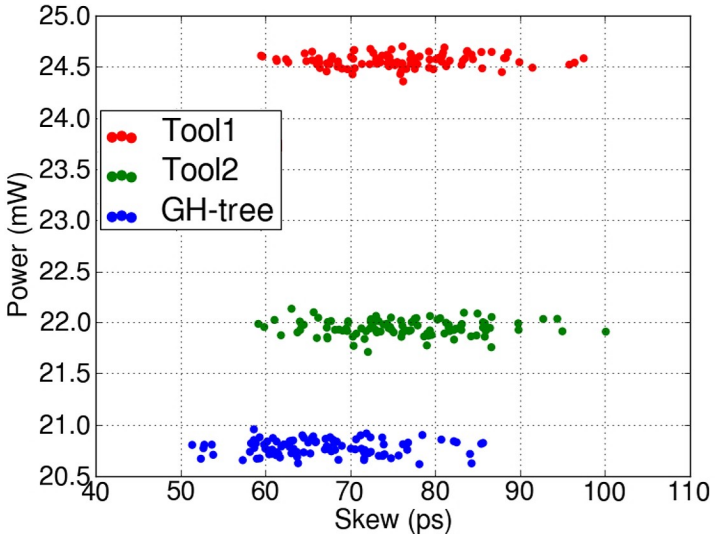


Figure 4.23: Clock skew and power comparison among GH-tree, Tool1 and Tool2 through Monte Carlo simulation. Design: *VGA*.

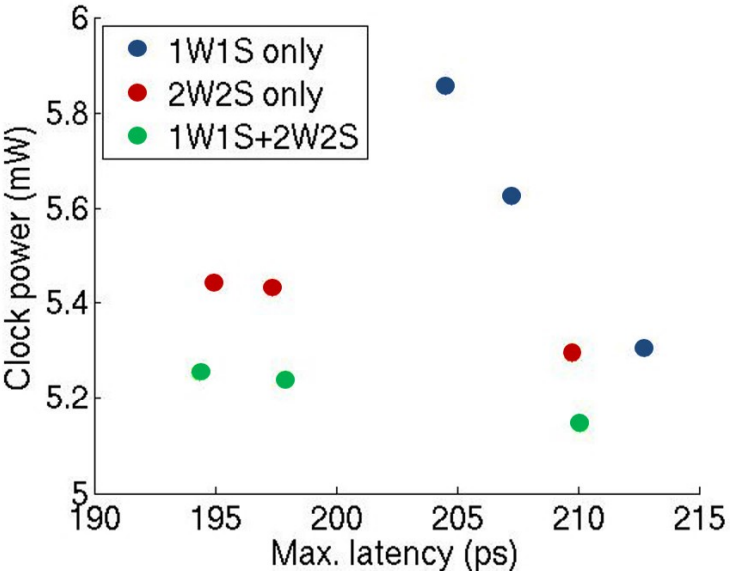


Figure 4.24: GH-tree optimization with various NDR options.

Study of "Skew Budgeting" across Clock Levels

How to optimally budget skew across clock tree levels has been an open problem for over two decades. Interestingly, our DP approach may provide new insights into how to budget skew for minimum clock power. To study the skew budgeting across clock levels, we run our optimizer multiple times with target skews from $5ps$ to $40ps$ in steps of $5ps$ on testcase *VGA*. In each run, we find the minimum-power GH-tree solution that satisfies the given target skew. Figure 4.25

shows the normalized skew at each clock level of the minimum-power GH-tree solutions across different target skews . When the skew constraint is tight (i.e., $\leq 15ps$), most of the skew occurs in the bottom levels (i.e., levels 5, 6 and 7) of the clock tree. However, when the skew constraint is relaxed (i.e., $\geq 20ps$), most of the skew occurs in the top levels (i.e., levels 1 and 2) of the clock tree. We show the normalized clock power for each target skew at the top of each bar in Figure 4.25. For example, minimum-power GH-tree solution for target skew $30ps$ consumes 74% power of minimum-power GH-tree solution for target skew $5ps$. To achieve $\sim$26% clock power reduction by changing target skew from $5ps$ to $30ps$, the clock tree must have $\sim$80% of the skew in levels 1 and 2. We emphasize that, as noted above, in our GH-tree a level does not necessarily imply the insertion of a buffer. In other words, a higher number of levels does not necessarily result in larger latency. Rather, we observe from our GH-tree solutions (which are on Pareto frontiers with respect to tradeoffs among skew, latency and clock power) that skew and latency are typically correlated with each other.

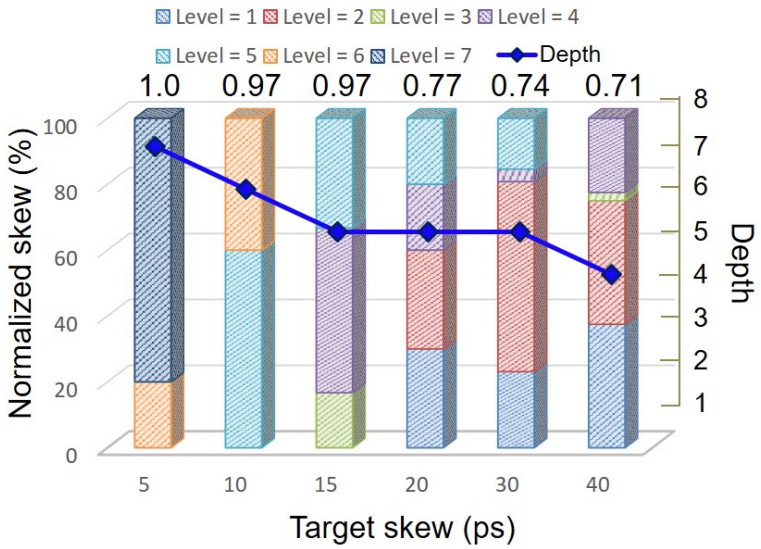


Figure 4.25: Skew budgeting, normalized clock power and the number of levels (depth) of the clock tree for different target skews. Nearly 80% of skew occurs in the bottom levels of the tree when the target skew is $\leq 15ps$, but this shifts to the top levels of the tree when the target skew is $\geq 20ps$.

Figure 4.26: Layout example of GH-tree on *VGA*. In red is clock routing of the top four levels in the GH-tree with branching pattern (4, 4, 2, 6). The left figure shows clock routing (top and bottom levels) and the right figure shows the sink clustering solution.

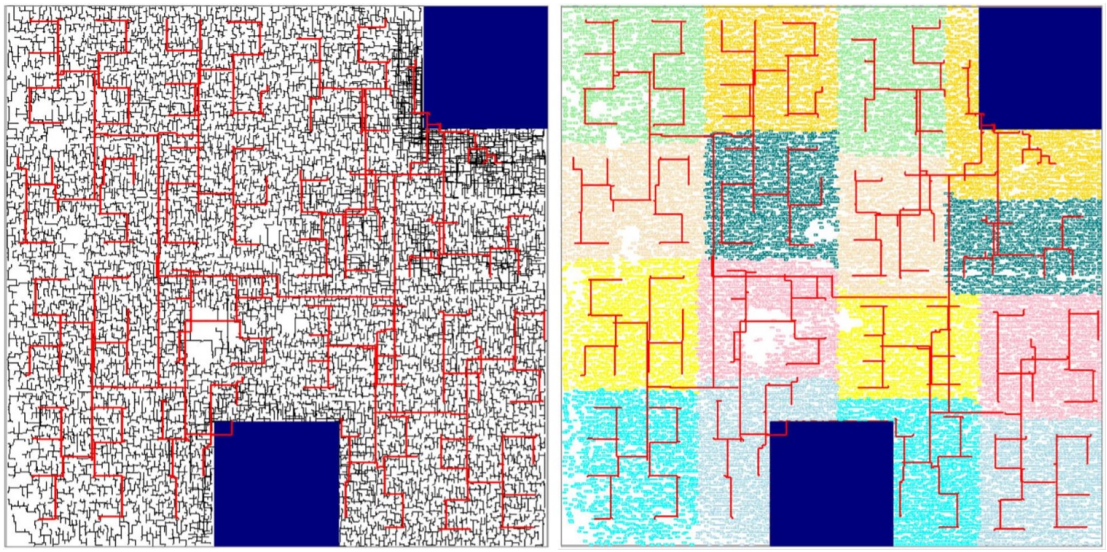


Figure 4.27: Layout example of GH-tree on *VGA_blockage*. In red is clock routing of the top six levels in the GH-tree with branching pattern (2, 2, 2, 2, 2, 4). The left figure shows clock routing (top and bottom levels) and the right figure shows the sink clustering solution.

4.3.4 Conclusion

In this section, we propose the concept of a *generalized H-tree*, which is a balanced tree topology with an arbitrary sequence of branching factors at each level. Our DP-based method provides an *optimal* GH-tree that has minimum clock power for a given skew and maximum latency targets. Our DP solutions are constructed using clock buffers (with ganging) along with

interconnect timing and power models from a 28LP foundry design enablement; we co-optimize the clock tree topology along with the buffering along branches. We furthermore propose a clustering- and linear programming-based heuristic to embed the GH-tree with respect to the given placement of clock sinks. We validate our solutions in commercial P&R tool flows in a 28LP foundry technology. The results show up to 30% clock power reduction while achieving similar skew and latency as CTS solutions from recent versions of leading commercial P&R tools. Our proposed approach also achieves up to 56% clock power reduction compared to a state-of-the-art academic tool [118]. Compared to "strict" H-tree, our results achieve better tradeoffs such that power is significantly reduced at the cost of small skew increase. Our ongoing and future work includes (i) co-optimization of sink placement and clock tree construction; (ii) budgeting of skew and latency across levels; (iii) application of useful skew in GH-tree; (iv) application of GH-tree construction in hierarchical designs (that require hierarchical CTS); (v) co-optimization of datapath placement and GH-tree construction; (vi) clock gate- and logic cells-aware GH-tree construction; and (vii) blockage-aware DP-based clock tree topology and buffering.

4.4 Acknowledgments

Chapter 4 contains reprints of Kwangsoo Han, Andrew B. Kahng and Jiajia Li, "Optimal Generalized H-Tree Topology and Buffering for High-Performance and Low-Power Clock Distribution", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2018 (to appear); Kwangsoo Han, Andrew B. Kahng, Christopher Moyes and Alex Zelikovsky, "A Study of Optimal Cost-Skew Tradeoff and Remaining Suboptimality in Interconnect Tree Constructions", *Proc. ACM International Workshop on System-Level Interconnect Prediction*, 2018; and Charles J. Alpert, Wing-Kai Chow, Kwangsoo Han, Andrew B. Kahng, Zhuo Li, Derong Lin and Sriram Venkatesh, "Prim-Dijkstra Revisited: Achieving Superior Timing-driven Routing Trees", *Proc. ACM International Symposium on Physical Design*, 2018. The dissertation author is a main contributor to, and a primary author of, each of these papers.

I would like to thank my coauthors Charles J. Alpert, Wing-Kai Chow, Andrew B. Kahng, Jiajia Li, Zhuo Li, Derong Liu, Christopher Moyes, Sriram Venkatesh and Alex Zelikovsky.

Chapter 5

Conclusion

This thesis has presented novel physical design optimization methodologies to resolve critical challenges faced by today's complex SOC design teams. The proposed methods compensate the slowdown of density scaling and the existing "design capability gap" with better design techniques and more accurate modeling.

Chapter 2 presents three distinct studies in the context of manufacturability and new design rules. (1) The chapter first presents a scalable MILP-based optimization of 2D block masks that considers block mask rules, minimum metal density constraints, and timing impact of dummy fills. Further, an improved timing impact model is proposed 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. This study shows up to 84% ΔWNS recovery and 85% ΔTNS recovery, and up to 56% Δswitching power recovery, along with up to 62% dummy removal rate. This enablement of a timing-aware optimization shows promising product-level benefits from use of 2D block masks, and further sheds light on the merits of various block mask optimization objectives. We furthermore study 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) Chapter 2 also proposes a scalable detailed placement legalization flow for complex FEOL constraints arising at the N10 foundry node. These include drain-drain abutment, minimum implant width, and minimum OD jogging rules. Given initial (timing-driven) placements, our $DFPlacer$ fixes 99% of DRVs with 3% increase in wirelength and minimal impact on timing. We feel that our use case of fixing all but a few tens of violations, with a highly parallelizable two-iteration strategy, is a

good practical tradeoff between runtime complexity and DRV fixing. Further, the level of DRV fixing achieved by DFPlacer is encouraging, given that our default experimental configuration makes no attempt at "correctness by construction". Using OpenMP, this flow is scalable via a distributed optimization strategy. Additionally, our study shows an area-DRV tradeoff between two types of standard-cell library strategies, namely, with and without dummy poly gates. (3) Last, Chapter 2 studies impacts of patterning technology choices and design rules on physical implementation metrics, with respect to *cost-optimal* design rule-correct detailed routing. We describe *OptRouter*, an ILP-based optimal detailed router that correctly handles multi-pin nets and various sub-20nm routing challenges including via restrictions, via shapes, and SADP patterning rules. OptRouter enables design rule evaluation using "difficult" routing clips (switchboxes) selected according to a pin cost metric. We study Δcost distributions for different design rules, relative to a RULE1 where all layers are LELE and there are no via restrictions. From the results, we observe that the sensitivities of Δcost to design rules and routing options vary with technology. Also, we observe that there is a gap between pin accessibility metrics such as [178] and our switchbox-centric evaluation of routability.

Chapter 3 presents several process-aware distinct design methodologies, which mitigate clock skew variation, die-to-die variation in 3DIC. (1) The chapter first presents a CTS methodology that optimizes CLC placement and buffer insertion, and that minimizes non-common paths between FF groups. We formulate the top-level CTS problem as the minimization of a weighted sum of WNS, TNS, clock uncertainty due to OCV, and wirelength. We solve this problem using LP and develop heuristic flows to insert Steiner points and buffers, which are required elements of a top-level CTS solution. We also develop generators for testcases that resemble clock tree structures typically found in high-speed SOCs. We validate our optimization flow on testcases from our generators and achieve up to 51% reduction in wirelength for the top-level clock tree, and 320ps improvement in WNS, compared to a leading commercial CTS tool. (2) Chapter 3 also proposes the first framework to minimize the sum of skew variations over all sequentially adjacent sink pairs, using both global and local optimizations. Our experimental results show that the proposed flow achieves up to 22% reduction of the sum of skew variations for testcases implemented in foundry 28nm technology, as compared to a leading commercial tool. In the global optimization, our LP formulation comprehends the ECO feasibility based on characterized lookup tables of stage delays. In the local optimization, we demonstrate that machine learning-based predictors of latency changes can provide accurate estimation of local move impacts. (3) Last, Chapter 3 proposes design-stage optimization for mix-and-match die stacking.

Our motivating insight is that *a priori* knowledge of mix-and-match 3DIC integration should influence multi-die partitioning optimization and signoff. We propose an ILP-based partitioning methodology and a heuristic partitioning methodology that performs maximum cut on the timing-critical sequential graph, followed by an iterative multi-phase FM optimization. We validate our partitioning optimization on two 3DIC implementation flows, each of which we have extended to be aware of mix-and-match die stacking. Our optimization shows up to 16% timing improvement, as compared to a flow based on min-cut based partitioning, when measured by RC extraction and signoff timing at the post-routing stage. Our study also indicates that a gate-level 3D integration has more flexibility and thus larger timing benefits in the mix-and-match regime as compared to a block-level integration.

Chapter 4 presents three distinct techniques for interconnect optimization. (1) The chapter first proposes a new spanning tree heuristic *PD-II*, which is demonstrated to significantly improve both WL and total detour cost compared to *PD*. Further, this work extends the construction to the Steiner tree regime, via the *DAS* algorithm that directly improves trees according to both the WL and detour cost objectives. The algorithms are shown to be fast and practical. They are also suitable for integration into existing commercial routers, and can be applied in conjunction with any existing spanning and Steiner tree constructions for simultaneous WL and PL improvements. Compared to the recent SALT algorithm, our construction generates clear improvements according to the proposed metrics, especially for medium-size and larger nets. (2) Chapter 4 also studies the minimum-cost bounded skew spanning and Steiner tree problems. We formulate and apply a flow-based ILP to find optimal cost-skew tradeoffs for generated testcases with number of terminals from 8 to 16. Based on the optimal cost-skew tradeoffs, we find significant remaining suboptimality of several state-of-art academic tools: (1) BST-DME, (2) SALT and (3) Prim-Dijkstra. Across our testcases, BST-DME has suboptimality $\sim 10\%$ in cost at iso-skew, and $\sim 50\%$ in skew at iso-cost. In addition, SALT and PD show suboptimality in terms of skew by up to $\sim 3\times$. This degree of suboptimality is very different from the near-optimality in practice of heuristics for the RSMT problem (e.g., FLUTE, 1-Steiner, etc.). Thus, our study motivates renewed attention to the cost-skew tradeoff. (3) Last, Chapter 4 proposes the concept of a *generalized H-tree*, which is a balanced tree topology with an arbitrary sequence of branching factors at each level. Our DP-based method provides an *optimal* GH-tree that has minimum clock power for a given skew and maximum latency targets. Our DP solutions are constructed using clock buffers (with ganging) along with interconnect timing and power models that are pre-chracterized from the foundry design enablement. We co-optimize the clock tree

topology along with the buffering along branches. We furthermore propose a clustering- and linear programming-based heuristic to embed the GH-tree with respect to the given placement of clock sinks. We validate our solutions in commercial P&R tool flows in a 28LP foundry technology. The results show up to 30% clock power reduction while achieving similar skew and latency as CTS solutions from recent versions of leading commercial P&R tools. Our proposed approach also achieves up to 56% clock power reduction compared to a state-of-the-art academic tool [118]. Compared to "strict" H-tree, our results achieve improved tradeoffs such that power is significantly reduced at the cost of small skew increase.

Bibliography

[1] A. Abdelhadi, R. Ginosar, A. Kolodny and E. G. Friedman, "Timing-Driven Variation-Aware Synthesis of Hybrid Mesh/Tree Clock Distribution Networks", *Integration, the VLSI Journal* 46(4) (2013), pp. 382-391.

[2] R. Aitken, *personal communication*, March 2015.

[3] R. Aitken, G. Yeric, B. Cline, S. Sinha, L. Shifren, I. Iqbal and V. Chandra, "Physical Design and FinFETs", *Proc. ACM International Symposium on Physical Design*, 2014, pp. 65-68.

[4] C. J. Alpert, *Personal Communication*, Nov. 2016.

[5] C. J. Alpert, W.-K. Chow, K. Han, A. B. Kahng, Z. Li, D. Liu and S. Venkatesh, "Prim-Dijkstra Revisited: Achieving Superior Timing-driven Routing Trees" *Proc. ACM International Symposium on Physical Design*, 2018, pp. 10-17.

[6] C. J. Alpert, A. Devgan and C. Kashyap, "A Two Moment RC Delay Metric for Performance Optimization", *Proc. ACM International Symposium on Physical Design*, 2000, pp. 73-78.

[7] C. J. Alpert, R. G. Gandham, J. Hu, S. T. Quay and A. J. Sullivan, "Apparatus and Method for Determining Buffered Steiner Trees for Complex Circuits", *U.S. Patent 6,591,411*, July 2003.

[8] C. J. Alpert, M. Hrkic, J. Hu, A. B. Kahng, J. Lillis, B. Liu, S. T. Quay, S. S. Sapatnekar, A. J. Sullivan and P. Villarrubia, "Buffered Steiner Trees for Difficult Instances", *Proc. ACM International Symposium on Physical Design*, 2001, pp. 4-9.

[9] C. J. Alpert, T. C. Hu, J. H. Huang, A. B. Kahng and D. Karger, "Prim-Dijkstra Tradeoffs for Improved Performance-driven Routing Tree Design", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 14(7) (1995), pp. 890-896.

[10] C. J. Alpert, J. Hu and P. H. Villarrubia, "Practical Methodology for Early Buffer and Wire Resource Allocation", *U.S. Patent 6,996,512*, Feb. 2006.

[11] C. J. Alpert, A. B. Kahng, C. N. Sze and Q. Wang, "Timing-driven Steiner Trees are (Practically) Free", *Proc. ACM/ESDA/IEEE Design Automation Conference*, 2006, pp. 389-392.

[12] 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* 8,954,912, May 2014.

[13] A. Andreev, A. Nikishin, S. Gribok, P.-C. Tan and C.-H. Choo, "Clock Network Fishbone Architecture for a Structured ASIC Manufactured on a 28 NM CMOS Process Lithographic Node", *U.S. Patent* 8,629,548, January 2014.

[14] Y. P. Aneja, "An Integer Linear Programming Approach to the Steiner Problem in Graphs", *Networks* 10(2), 1980, pp. 167-178.

[15] M. Badaroglu, "More Moore Scaling: Opportunities and Inflection Points", *ITRS Rebooting Computing Workshop*, 2015.

[16] Y. Badr, K.-W. Ma and P. Gupta, "Layout Pattern-Driven Design Rule Evaluation", *SPIE Journal of Micro/Nanolithography, MEMS, and MOEMS* 13(4) (2014), p. 043018.

[17] H. B. Bakoglu, *Circuits, Interconnections and Packaging for VLSI*, Reading, MA, Addison-Wesley, 1990.

[18] J. Bhasker and R. Chadha, *Static Timing Analysis for Nanometer Designs: A Practical Approach*, Springer, 2009.

[19] K. Boese and A. B. Kahng, "Zero-Skew Clock Routing Trees With Minimum Wirelength", *Proc. IEEE International Conference on ASIC*, 1992, pp. 1.1.1 - 1.1.5.

[20] M. Borah, R. M. Owens and M. J. Irwin, "An Edge-based Heuristic for Steiner Routing", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 13(12) (1994), pp. 1563-1568.

[21] S. Bose, "Methods and Systems for Placement and Routing", *U.S. Patent 8,332,793*, Dec. 2012.

[22] Broadcom Corporation (networking infrastructure physical design principal engineer), *personal communication*, November 2013.

[23] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Improved Algorithms for Hypergraph Bipartitioning", *Proc. Asia and South Pacific Design Automation Conference*, 2000, pp. 661-666.

[24] A. Cao, S.-M. Chang and D.-C. Yuan, "Local Clock Skew Optimization", *U.S. Patent* 8,635,579, 2014.

[25] R. Carden and C.K. Cheng, "A Global Router with a Theoretical Bound on the Optimal Solution", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 15(2) (1996), pp. 208-216.

[26] T.-B. Chan, K. Han, A. B. Kahng, J.-G. Lee and S. Nath, "OCV-Aware Top-Level Clock Tree Optimization", *Proc. Great Lakes Symposium on Very Large Scale Integration*, 2014, pp. 33-38.

[27] T.-B. Chan, A. B. Kahng and J. Li, "Reliability-Constrained Die Stacking Order in 3DICs under Manufacturing Variability", *Proc. International Symposium on Quality Electronic Design*, 2013, pp. 16-23.

[28] T.-B. Chan, A. Kahng and J. Li, "Toward Quantifying the IC Design Value of Interconnect Technology Improvements", *Proc. ACM International Workshop on System-Level Interconnect Prediction*, 2013, pp. 1-6.

[29] T. H. Chao, Y. C. Hsu, J. M. Ho, K. D. Boese and A. B. Kahng, "Zero Skew Clock Routing With Minimum Wirelength", *IEEE Transactions On Circuits and Systems* 39(11) (1992), pp. 799-814.

[30] M. Charikar, J. Kleinberg, R. Kumar, S. Rajagopalan, A. Sahai and A. Tomkins, "Minimizing Wirelength in Zero and Bounded Skew Clock Trees", *SIAM Journal on Discrete Mathematics* 17(4) (2004), pp. 582-595.

[31] Y.-Y. Chen, C. Dong and D. Chen, "Clock Tree Synthesis Under Aggressive Buffer Insertion", *Proc. ACM/ESDA/IEEE Design Automation Conference*, 2010, pp. 86-89.

[32] C. Chen, C. Kang and M. Sarrafzadeh, "Activity-Sensitive Clock Tree Construction for Low Power", *Proc. International Symposium on Low Power Electronic Design*, 2002, pp. 279-282.

[33] G. Chen, P. Tu and E. F. Y. Young, "SALT: Provably Good Routing Topology by a Novel Steiner Shallow-Light Tree Algorithm", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2017, pp. 569-576.

[34] M. Cho, S. Ahmed and D. Z. Pan, "TACO: Temperature Aware Clock-tree Optimization", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2005, pp. 582-587.

[35] M. Cho and D. Z. Pan, "BoxRouter: A New Global Router Based on Box Expansion and Progressive ILP", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 26(12) (2007), pp. 2130-2143.

[36] H.-M. Chou, H. Yu and S.-C. Chang, "Useful-Skew Clock Optimization for Multi-Power Mode Designs", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2011, pp. 647-650.

[37] C. Chu, "FLUTE: Fast Lookup Table Based Wirelength Estimation Technique", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2004, pp. 696-701.

[38] C. Chu and Y.C. Wong, "FLUTE: Fast Lookup Table Based Rectilinear Steiner Minimal Tree Algorithm for VLSI Design", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27(1) (2008), pp.70-83.

[39] J. Cong, A. B. Kahng, C. K. Koh and C.-W. A. Tsao, "Bounded-Skew Clock and Steiner Routing", *ACM Transactions on Design Automation of Electronic Systems* 3(3) (1998), pp. 341-388.

[40] J. Cong, A. B. Kahng, G. Robins and M. Sarrafzadeh, "Provably Good Performance-driven Global Routing", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 11(6) (1992), pp. 739-752.

[41] J. Cong, A. B. Kahng, G. Robins, M. Sarrafzadeh and C.K. Wong, "Performance-driven Global Routing for Cell Based ICs", *Proc. IEEE International Conference on Computer Design*, 1991, pp. 170-173.

[42] J. Cong, K. S. Leung and D. Zhou, "Performance-driven Interconnect Design Based on Distributed RC Delay Model", *Proc. ACM/ESDA/IEEE Design Automation Conference*, 1993, pp. 606-611.

[43] J. Cong, G. Luo, J. Wei and Y. Zhang, "Thermal-Aware 3D IC Placement Via Transformation", *Proc. Asia and South Pacific Design Automation Conference*, 2007, pp. 780-785.

[44] M. Darmi, L. Cherif, J. Benallal, R. Elgouri and N. Hmina, "Integrated Circuit Conception: A Wire Optimization Technic Reducing Interconnection Delay in Advanced Technology Nodes", *Eletronics* 6(4) (2017), p. 78.

[45] 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.

[46] E. W. Dijkstra, "A Note on Two Problems in Connexion with Graphs", *Numerische Mathematik* 1 (1959), pp. 269-271.

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

[48] D. Dolev, M. Fugger, C. Lenzen, M. Perner and U. Schmid, "HEX: Scaling Honeycombs is Easier Than Scaling Clock Trees", *Proc. ACM Symposium on Parallelism in Algorithms and Architectures*, 2013, pp. 164-175.

[49] 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.

[50] 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.

[51] S. Dutt and W. Deng, "VLSI Circuit Partitioning by Cluster-removal Using Iterative Improvement Techniques", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 1996, pp. 194-200.

[52] M. Edahiro, "A Clustering-Based Optimization Algorithm in Zero-Skew Routings", *Proc. ACM/ESDA/IEEE Design Automation Conference*, 1993, pp. 612-616.

[53] M. Elkin and S. Solomon, "Narrow-shallow-low-light Trees With and Without Steiner Points", *SIAM Journal on Discrete Mathematics* 25(1) (2011), pp. 181-210.

[54] M. Elkin and S. Solomon, "Steiner Shallow-light Trees are Exponentially Lighter than Spanning Ones", *SIAM Journal on Computing* 44(4) (2015), pp. 996-1025.

[55] R. Ewetz and C.-K. Koh, "Cost-Effective Robustness in Clock Networks Using Near-Tree Structures", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34(4) (2015), pp. 515-528.

[56] 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.

[57] C. Ferri, S. Reda and R. I. Bahar, "Parametric Yield Management for 3D ICs: Models and Strategies for Improvement", *ACM Journal on Emerging Technologies in Computing Systems* 4(4) (2008), pp. 19:1-19:22.

[58] C. M. Fiduccia and R. M. Mattheyses, "Linear Time Heuristic for Improving Network Partitions", *Proc. ACM/ESDA/IEEE Design Automation Conference*, 1982, pp. 175-181.

[59] J. P. Fishburn, "Clock Skew Optimization", *IEEE Transactions on Computers* 39(7) (1990), pp. 945-951.

[60] E. G. Friedman, "Clock Distribution Networks in Synchronous Digital Integrated Circuits", *Proc. of IEEE*, 89(5) (2001), pp. 665-692.

[61] G. M. Furnish, M. J. LeBrun and S. Bose, "Node Spreading Via Artificial Density Enhancement to Reduce Routing Congestion", *U.S. Patent 7,921,392*, Apr. 2011.

[62] G. M. Furnish, M. J. LeBrun and S. Bose, "Tunneling as a Boundary Congestion Relief Mechanism", *U.S. Patent 7,921,393*, Apr. 2011.

[63] S. Garg and D. Marculescu, "Mitigating the Impact of Process Variation on the Performance of 3-D Integrated Circuits", *IEEE Transactions on Very Large Scale Integration Systems* 21(10) (2013), pp. 1903-1914.

[64] R. S. Ghaida, Y. Badr, M. Gupta, N. Jin and P. Gupta, "Comprehensive Die-Level Assessment of Design Rules and Layouts", *Proc. Asia and South Pacific Design Automation Conference*, 2014, pp. 61-66.

[65] R. S. Ghaida and P. Gupta, "DRE: A Framework for Early Co-Evaluation of Design Rules, Technology Choices, and Layout Methodologies", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31(9) (2012), pp. 1379-1392.

[66] 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", *SPIE Advanced Lithography*, 2015, p. 942709.

[67] M. X. Goemans and D. P. Williamson, "Improved Approximation Algorithms for Maximum Cut and Satisfiability Problems Using Semidefinite Programming", *Journal of the ACM* 42(6) (1995), pp. 1115-1145.

[68] J. Griffith, G. Robins, J. S. Salowe and T. Zhang, "Closing the Gap: Near-optimal Steiner Trees in Polynomial Time", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 1994, pp. 1351-1365.

[69] L. J. Guibas and J. Stolfi, "On Computing All Northeast Nearest Neighbors in the L1 Metric", *Information Processing Letters* 17 (1983), pp. 219-223.

[70] 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.

[71] 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.

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

[73] P. Gupta, K. Jeong, A. B. Kahng and C.-H. Park, "Electrical Assessment of Lithographic Gate Line-End Patterning", *SPIE Journal of Microlithography, Microfabrication and Microsystems* 9(2) (2010), pp. 023014-1–023014-19.

[74] M. R. Guthaus, D. Sylvester and R. B. Brown, "Clock Buffer and Wire Sizing using Sequential Programming", *Proc. ACM/ESDA/IEEE Design Automation Conference*, 2006, pp. 1041-1046.

[75] M. R. Guthaus, G. Wilke and R. Reis, "Non-uniform Clock Mesh Optimization with Linear Programming Buffer Insertion", *Proc. ACM/ESDA/IEEE Design Automation Conference*, 2010, pp. 74-79.

[76] M. R. Guthaus, G. Wilke and R. Reis, "Revisiting Automated Physical Synthesis of High-performance Clock Networks", *ACM Transactions on Design Automation of Electronic Systems* 18(2) (2013), pp. 31:1-31:27.

[77] L. W. Hagen, D. J.-H Huang and A. B. Kahng, "On Implementation Choices for Iterative Improvement Partitioning Algorithms", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 16(10) (1997), pp. 1199-1205.

[78] K. Han, A. B. Kahng and H. Lee, "Evaluation of BEOL Design Rule Impacts Using an Optimal ILP-based Detailed Router", *Proc. ACM/ESDA/IEEE Design Automation Conference*, 2015, pp. 1-6.

[79] K. Han, A. B. Kahng, J. Lee, J. Li and S. Nath, "A Global-Local Optimization Framework for Simultaneous Multi-Mode Multi-Corner Skew Variation Reduction", *Proc. ACM/ESDA/IEEE Design Automation Conference*, 2015, pp. 26:1-26:6.

[80] 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/BACUS Symposium on Photomask Technology and Management*, 2015, pp. 1-14.

[81] S. S. Han, A. B. Kahng, S. Nath and A. Vydyanathan, "A Deep Learning Methodology to Proliferate Golden Signoff Timing", *Proc. Design, Automation and Test in Europe*, 2014, pp. 1-6.

[82] 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 Lithographic Technologies VII*, 2016, p. 977718.

[83] M. Hanan, "On Steiner's Problem with Rectilinear Distance", *SIAM Journal on Applied Mathematics* 14(2) (1966), pp. 255-265.

[84] T. Hastie, R. Tibshirani and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer, 2009.

[85] L. He, C.-K. Koh, D. Z. Pan and X. Yuan, *TRIO release B 1.0*. `http://vlsicad.eecs.umich.edu/BK/Slots/cache/cadlab.cs.ucla.edu/software\_release/trio/htdocs`

[86] L. He, S. Yao, W. Deng, J. Chen and L. Chao, "Interconnect Routing Methods of Integrated Circuit Designs", *U.S. Patent 8,386,984*, Feb. 2013.

[87] R. F. Hentschke, M. de Oliveira Johann, J. Narasimhan and R. A. de Luz Reis, "Methods and Apparatus for Providing Flexible Timing-driven Routing Trees", *U.S. Patent 8,095,904*, Jan. 2012.

[88] A. Hill, keynote address, *Proc. ACM International Symposium on Physical Design*, 2018. `http://www.ispd.cc/slides/2018/k1.pdf` (Slide 17).

[89] J. M. Ho, G. Vijayan and C. K. Wong, "New Algorithms for the Rectilinear Steiner Tree Problem", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 9(2) (1990), pp. 185-193.

[90] Y. C. Hu, Y. L. Chung and M. C. Chi, "A Multilevel Multilayer Partitioning Algorithm for Three Dimensional Integrated Circuits", *Proc. International Symposium on Quality Electronic Design*, 2010, pp. 483-487.

[91] J. Hu, A. B. Kahng, B. Liu, G. Venkataraman and X. Xu, "A Global Minimum Clock Distribution Network Augmentation Algorithm for Guaranteed Clock Skew Yield", *Proc. Asia and South Pacific Design Automation Conference*, 2007, pp. 24-31.

[92] J. Hu, J. A. Roy and I. L. Markov, "Sidewinder: A Scalable ILP-based Router", *Proc. ACM International Workshop on System-Level Interconnect Prediction*, 2008, pp. 73-80.

[93] J.-H. Huang, A. B. Kahng and C.-W. A Tsao, "On the Bounded-Skew Clock and Steiner Routing Problems", *Proc. ACM/ESDA/IEEE Design Automation Conference*, 1995, pp. 508-513.

[94] 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.

[95] S. Jang, Samsung Electronics, *personal communication*, May 2015.

[96] X. Jia, Y. Cai, Q. Zhou, G. Chen, Z. Li and Z. Li, "MCFRoute: A Detailed Router Based on Multi-Commodity Flow Method", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2014, pp. 397-404.

[97] I. H.-R. Jiang, "Generic Integer Linear Programming Formulation for 3D IC Partitioning", *Proc. IEEE International SOC Conference*, 2009, pp. 321-324.

[98] Q.-T. Jiang, M.-H. Tsai and R. H. Havemann, "Line Width Dependence of Copper Resistivity", *Proc. IEEE International Interconnect Technology Conference*, 2001, pp. 227-229.

[99] D.-C. Juan, S. Garg and D. Marculescu, "Statistical Peak Temperature Prediction and Thermal Yield Improvement for 3D Chip Multiprocessors", *ACM Transactions on Design Automation of Electronic Systems* 19(4) (2014), pp. 39:1-39:23.

[100] M. Jung, *personal communication*, 2013.

[101] A. B. Kahng, "New Game, New Goal Posts: A Recent History of Timing Closure", *Proc. ACM/ESDA/IEEE Design Automation Conference*, 2015, pp. 1-6.

[102] A. B. Kahng, "The ITRS Design Technology and System Drivers Roadmap: Process and Status", *Proc. ACM/ESDA/IEEE Design Automation Conference*, 2013, pp. 34-39.

[103] A. B. Kahng and H. Lee, "Minimum Implant Area-Aware Gate Sizing and Placement", *Proc. Great Lakes Symposium on Very Large Scale Integration*, 2014, pp. 57-62.

[104] A. B. Kahng, J. Lienig, I. L. Markov and J. Hu, *VLSI Physical Design: From Graph Partitioning to Timing Closure*, Springer, 2011.

[105] A. B. Kahng, B. Lin and S. Nath, "High-Dimensional Metamodeling for Prediction of Clock Tree Synthesis Outcomes", *Proc. ACM International Workshop on System-Level Interconnect Prediction*, 2013, pp. 1-7.

[106] A. B. Kahng, B. Lin and S. Nath, "Enhanced Metamodeling Techniques for High-Dimensional IC Design Estimation Problems", *Proc. Design, Automation and Test in Europe*, 2013, pp. 1861-1866.

[107] A. B. Kahng, I. L. Markov and S. Reda, "On Legalization of Row-Based Placements", *Proc. Great Lakes Symposium on Very Large Scale Integration*, 2004, pp. 214-219.

[108] A. B. Kahng and G. Robins, "A New Class of Iterative Steiner Tree Heuristics with Good Performance", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 11(7) (1992), pp. 893-902.

[109] A. B. Kahng and G. Robins, *On Optimal Interconnects for VLSI*, Kluwer Academic Publishers, 1995.

[110] 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.

[111] A. B. Kahng and C.-W. A. Tsao, "Planar-DME: A Single-Layer Zero-Skew Clock Tree Router", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 15(1) (1996), pp. 8-19.

[112] A. B. Kahng and C.-W. A. Tsao, "Practical Bounded-Skew Clock Routing", *Journal of VLSI Signal Processing* 16 (1997), pp. 199-215.

[113] A. B. Kahng and C.-W. A. Tsao, "VLSI CAD Software Bookshelf: Bounded-Skew Clock Tree Routing", Version 1.0, 2000. `http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/BST/`

[114] A. B. Kahng and X. Xu, "Local Unidirectional Bias for Smooth Cutsize-Delay Tradeoff in Performance-Driven Bipartitioning", *Proc. ACM International Symposium on Physical Design*, 2003, pp. 81-86.

[115] G. Karypis and V. Kumar, "Multilevel K-Way Hypergraph Partitioning", *Proc. ACM/ESDA/IEEE Design Automation Conference*, 1999, pp. 343-348.

[116] C. V. Kashyap, C. J. Alpert, F. Liu and A. Devgan, "PERI: A Technique for Extending Delay and Slew Metrics to Ramp Inputs", *Proc. ACM International Workshop on Timing Issues in the Specification and Synthesis of Digital Systems*, 2002, pp. 57-62.

[117] S. Khuller, B. Raghavachari and N. Young, "Balancing Minimum Spanning Trees and Shortest-path Trees", *Proc. SIAM Symposium on Discrete Algorithms*, 1993, pp. 243-250.

[118] Y. Kim and T. Kim, "Algorithm for Synthesis and Exploration of Clock Spines", *Proc. Asia and South Pacific Design Automation Conference*, 2017, pp. 263-268.

[119] G. Kortsarz and D. Peleg, "Approximating Shallow-light Trees", *Proc. SIAM Symposium on Discrete Algorithms*, 1997, pp. 103-110.

[120] B. Krishnamurthy, "An Improved Min-Cut Algorithm for Partitioning VLSI Networks", *IEEE Transactions on Computers* 33(5) (1984), pp. 438-446.

[121] J. B. Kruskal Jr., "On the Shortest Spanning Subtree of a Graph and the Traveling Salesman Problem", *Proc. American Mathematical society* 7(1) (1956), pp. 48-50.

[122] C. Y. Lee, C.-Y. Ting and J.-H. Shieh, "Method of Patterning for a Semiconductor Device", *U.S. Patent* 8,697,537, April 15, 2014.

[123] K.-S. Leung and J. Cong, "Fast Optimal Algorithms for the Minimum Rectilinear Steiner Arborescence Problem", *Proc. IEEE International Symposium on Circuits and Systems*, 1997, pp. 1568-1571.

[124] Z. Li, X. Hong, Q. Zhou, Y. Cai, J. Bian, H. H. Yang, V. Pitchumani, C.-K. Cheng, "Hierarchical 3-D Floorplanning Algorithm for Wirelength Optimization", *IEEE Transactions on Circuits and Systems I* 53(12) (2006), pp. 2637-2646.

[125] S. Li and C.-K. Koh, "Mixed Integer Programming Models for Detailed Placement", *Proc. ACM International Symposium on Physical Design*, 2012, pp. 87-94.

[126] 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.

[127] L. Liebmann, V. Gerousis, P. Gutwin, M. Zhang, G. Han and B. Cline, "Demonstrating Production Quality Multiple Exposure Patterning Aware Routing for the 10NM Node", *Proc. SPIE Design-Process-Technology Co-optimization for Manufacturability VIII*, 2014, p. 905309.

[128] L. Liebmann and D. Pietromonaco, "The Increasing Pain of Scaling with 193i: Where Does it Hurt? How Much More Can We Endure?", *tutorial*, *SPIE Advanced Lithography*, 2013.

[129] A. Lim, S.-W. Cheng and C.-T. Wu, "Performance Oriented Rectilinear Steiner Trees", *Proc. ACM/ESDA/IEEE Design Automation Conference*, 1993, pp. 171-175.

[130] I.-M. Liu, T.-L. Chou, A. Aziz and D. F. Wong, "Zero-Skew Clock Tree Construction by Simultaneous Routing, Wire Sizing and Buffer Insertion", *Proc. ACM International Symposium on Physical Design*, 2000, pp. 33-38.

[131] D. Liu and C. Svensson, "Power Consumption Estimation in CMOS VLSI Circuits", *IEEE Journal of Solid State Circuits* 29(6) (1994), pp. 663-670.

[132] J. Lu, H. Zhuang, P. Chen, H. Chang, C.-C. Chang, Y.-C. Wong, L. Sha, D. Huang, Y. Luo, C.-C. Teng and C.-K. Cheng, "ePlace-MS: Electrostatics based Placement for Mixed-Size Circuits", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34(5) (2015), pp. 685-698.

[133] C.-L. Lung, H.-C. Hsiao, Z.-Y. Zeng and S.-Y. Chang, "LP-Based Multi-Mode Multi-Corner Clock Skew Optimization", *Proc. International Symposium on VLSI Design, Automation and Test*, 2010, pp. 335-338.

[134] C.-L. Lung, Z.-Y. Zeng, C.-H. Chou and S.-Y. Chang, "Clock Skew Optimization Considering Complicated Power Modes", *Proc. Design, Automation and Test in Europe*, 2010, pp. 1474-1479.

[135] 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.

[136] F. Minami and M. Takano, "Clock Tree Synthesis Based on RC Delay Balancing", *Proc. IEEE Custom Integrated Circuits Conference*, 1992, pp. 28-3.1-28.3.4.

[137] T. Mittal and C.-K. Koh, "Cross Link Insertion for Improving Tolerance to Variations in Clock Network Synthesis", *Proc. ACM International Symposium on Physical Design*, 2011, pp. 29-36.

[138] A. Naamad, D. T. Lee and W.-L. Hsu, "On the Maximum Empty Rectangle Problem", *Discrete Applied Mathematics* 8 (1984), pp. 267-277.

[139] J. Oh, I. Pyo and M. Pedram, "Constructing Lower and Upper Bounded Delay Routing Trees Using Linear Programming", *Proc. ACM/ESDA/IEEE Design Automation Conference*, 1996, pp. 401-404.

[140] A. Olofsson, "Silicon Compilers - Version 2.0", *keynote*, *Proc. ACM International Symposium on Physical Design*, 2018. `http://www.ispd.cc/slides/2018/k2.pdf`

[141] M. M. Ozdal, C. Amin, A. Ayupov, S. M. Burns, G. R. Wilke and C. Zhuo, "ISPD-2012 Discrete Cell Sizing Contest and Benchmark Suite", *Proc. ACM International Symposium on Physical Design*, 2012, pp. 161-164. `http://archive.sigda.org/ispd/contests/12/ispd2012_contest.html`.

[142] U. Padmanabhan, J. M. Wang and J. Hu, "Robust Clock Tree Routing in the Presence of Process Variations", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27(8) (2008), pp. 1385-1397.

[143] S. Panth, K. Samadi, Y. Du and S. K. Lim, "Design and CAD Methodologies for Low Power Gate-level Monolithic 3D ICs", *Proc. International Symposium on Low Power Electronic Design*, 2014, pp. 171-176.

[144] S. Panth, K. Samadi, Y. Du and S. K. Lim, "Placement-Driven Partitioning for Congestion Mitigation in Monolithic 3D IC Designs", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34(4) (2015), pp. 540-553.

[145] D. A. Papa, T. Luo, M. D. Moffitt, C. N. Sze, Z. Li, G.-J. Nam, C. J. Alpert and I. L. Markov, "RUMBLE: An Incremental Timing-Driven Physical-Synthesis Optimization Algorithm", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27(12) (2008), pp. 2156-2168.

[146] M. Pedram, "Power Minimization in IC Design: Principles and Applications" , *ACM Transactions on Design Automation of Electronic Systems* 1(1) (1996), pp. 3-56.

[147] S. Peyer, M. Zachariasen and D. G. Jorgensen, "Delay-related Secondary Objectives for Rectilinear Steiner Minimum Trees", *Discrete Applied Mathematics* 136(2-3) (2004), pp. 271-298.

[148] R. C. Prim, "Shortest Connecting Networks and Some Generalizations", *Bell System Technical Journal* 36 (1957), pp. 1389-1401.

[149] Qualcomm, Inc. VLSI technology principal engineer, *personal communication*, July 2014.

[150] 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.

[151] A. Rajaram, J. Hu, and R. Mahapatra, "Reducing Clock Skew Variability via Crosslinks", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 25(6) 2006, pp. 1176-1182.

[152] A. Rajaram and D. Z. Pan, "Robust Chip-Level Clock Tree Synthesis", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 30(6) (2011), pp. 877-890.

[153] A. Rajaram and D. Z. Pan, "Variation Tolerant Buffered Clock Network Synthesis with Cross Links", *Proc. ACM International Symposium on Physical Design*, 2006, pp. 157-164.

[154] L. Rakai, A. Farshidi, L. Behjat and D. Westwick, "Buffer Sizing for Clock Networks using Robust Geometric Programming Considering Variations in Buffer Sizes", *Proc. ACM International Symposium on Physical Design*, 2013, pp. 154-161.

[155] P. Ramachandaran, A. R. Agnihotri, S. Ono, P. Damodaran, K. Srihari and P. H. Madden, "Optimal Placement by Branch-and-Price", *Proc. Asia and South Pacific Design Automation Conference*, 2005, pp. 337-342.

[156] V. Ramachandran, "Construction of Minimal Functional Skew Clock Trees", *Proc. ACM International Symposium on Physical Design*, 2012, pp. 119-120.

[157] R. R. Rao, D. Blaauw, D. Sylvester, C. J. Alpert and S. Nassif, "An Efficient Surface-Based Low-Power Buffer Insertion Algorithm", *Proc. ACM International Symposium on Physical Design*, 2005, pp. 86-93.

[158] S. K. Rao, P. Sadayappan, F. K. Hwang and P. W. Shor, "The Rectilinear Steiner Arborescence Problem", *Algorithmica* 7(2) (1992), pp. 277-288.

[159] P. J. Restle, T. G. McNamara, D. A. Webber, P. J. Camporese, K. F. Eng, K. A. Jenkins, D. H. Allen, M. J. Rohn, M. P. Quaranta, D. W. Boerstler, C. J. Alpert, C. A. Carter, R. N. Bailey, J. G. Petrovick, B. L. Krauter, and B. D. McCredie, "A Clock Distribution Network for Microprocessors", *IEEE Journal of Solid State Circuits* 36(5) (2001), pp. 792-799.

[160] J. Reuben, H. M. Kittur and M. Shoaib, "A Novel Clock Generation Algorithm for System-on-Chip Based on Least Common Multiple", *Computers and Electrical Engineering* 40(7) (2014), pp. 2113-2125.

[161] J. Reuben, V. M. Zackriya, S. Nashit and H. M. Kittur, "Capacitance Driven Clock Mesh Synthesis to Minimize Skew and Power Dissipation", *IEICE Electronics Express* 10(24) (2013), pp. 1-12.

[162] R. Samanta, J. Hu and P. Li, "Discrete Buffer and Wire Sizing for Link-Based Non-Tree Clock Networks", *IEEE Transactions on Very Large Scale Integration Systems* 18(7) (2010), pp. 1025-1035.

[163] Samsung Electronics Corp. CAE principal engineer, *personal communication*, September 2014.

[164] Samsung Electronics Corporation (System LSI application processor principal engineer), *personal communication*, November 2013.

[165] Samsung Electronics Corporation (System LSI application processor principal engineer), *personal communication*, July 2014.

[166] V. Sathe, S. Arekapudi, A. Ishii, C. Ouyang, M. Papaefthymiou and S. Naffziger, "Resonant Clock Design for a Power-efficient, High-volume x86-64 Microprocessor", *Proc. International Solid State Circuits Conference*, 2012, pp. 140-149.

[167] P. Saxena, V. Khandelwal, C. Qiao, P-H. Ho, J. C. Lin and M. A. Iyer, "Interconnect-Driven Physical Synthesis using Persistent Virtual Routing", *U.S. Patent 7,853,915*, December 2010.

[168] L. Scheffer, Bookshelf RMST code, `http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/RSMT/RMST`.

[169] R. Scheifele, "Steiner Trees with Bounded RC-delay", *Algorithmica* 78(1) (2017), pp. 86-109.

[170] H. Seo, J. Kim, M. Kang and T. Kim, "Synthesis for Power-Aware Clock Spines", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2015, pp. 126-131.

[171] 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.

[172] W. Shi, Z. Li and C. Alpert, "Complexity Analysis and Speedup Techniques for Optimal Buffer Insertion with Minimum Cost", *Proc. Asia and South Pacific Design Automation Conference*, 2004, pp. 609-614.

[173] W. Shi and C. Su, "The Rectilinear Steiner Arborescence Problem is NP-complete", *SIAM Journal on Computing* 35(3) (2006), pp. 729-740.

[174] Y.-H. Su and Y.-W. Chang, "Nanowire-Aware Routing Considering High Cut-Mask Complexity", *Proc. ACM/ESDA/IEEE Design Automation Conference*, 2015, pp. 1-6.

[175] H. Su and S. S. Sapatnekar, "Hybrid Structured Clock Network Construction", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2001, pp. 333-336.

[176] S. Sunder and K. Scholtman, "Multi-Mode Multi-Corner Clocktree Synthesis", *U.S. Patent* No. US20090217225 A1, 2009.

[177] C. N. Sze, "ISPD 2010 High Performance Clock Network Synthesis Contest: Benchmark Suite and Results", *Proc. ACM International Symposium on Physical Design*, 2010, pp. 143-143.

[178] T. Taghavi, C. Alpert, A. Huber, Z. Li, G.-J. Nam, S. Ramji, "New Placement Prediction and Mitigation Techniques for Local Routing Congestion", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2010, pp.621-624.

[179] S. Tam, "Modern Clock Distribution Systems" in *Clocking in Modern VLSI Systems*, New York, Springer, 2009.

[180] B. Taskin, *personal communication*, Oct. 2012.

[181] T. Thorolfsson, G. Luo, J. Cong and P. D. Franzon, "Logic-on-logic 3D Integration and Placement", *Proc. 3D Systems Integration Conference*, 2010, pp. 1-4.

[182] J.-L. Tsai, "Clock Tree Synthesis for Timing Convergence and Timing Yield Improvement in Nanometer Technologies", *Ph.D. Thesis*, Electrical and Computer Engineering, University of Wisconsin-Madison, 2005.

[183] J.-L. Tsai, T.-H. Chen and C. C.-P. Chen, "Zero Skew Clock-Tree Optimization with Buffer Insertion/Sizing and Wire Sizing", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 23(4) (2004), pp. 565-572.

[184] C.-W. A. Tsao, *BST-DME source code*, MARCO GSRC Bookshelf, 2002. `https://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/BST/download/`

[185] C.-W. A. Tsao and C.-K. Koh, "UST/DME: A Clock Tree Router for General Skew Constraints", *ACM Transactions on Design Automation of Electronic Systems*, 7(3) (2002), pp. 359-379.

[186] R.-S. Tsay, "Exact Zero-Skew", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 1991, pp. 336-339.

[187] D. Velenis, M. C. Papaefthymiou and E. G. Friedman, "Reduced Delay Uncertainty in High Performance Clock Distribution Networks", *Proc. Design, Automation and Test in Europe*, 2003, pp. 68-73.

[188] G. Venkataraman, N. Jayakumar, J. Hu, P. Li and S. Khatri, "Practical Techniques to Reduce Skew and Its Variations in Buffered Clock Networks", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2005, pp. 592-596.

[189] N. Viswanathan, C. J. Alpert, C. C. N. Sze, Z. Li and Y. Wei, "The DAC 2012 Routability-driven Placement Contest and Benchmark Suite", *Proc. ACM/ESDA/IEEE Design Automation Conference*, 2012, pp. 774-782.

[190] A. Vittal and M. Marek-Sadowska, "Low-Power Buffered Clock Tree Design", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 16(9) (1997), pp. 965-975.

[191] D. M. Warme, P. Winter and M. Zachariasen, "Exact Algorithms for Plane Steiner Tree Problems: A Computational Study", in D.Z. Du, J.M. Smith and J.H. Rubinstein (Eds.) *Advances in Steiner Trees*, Kluwer Academic Publishers, 2000, pp. 81-116.

[192] X. Xu, B. Cline, G. Yeric, B. Yu and D. Z. Pan, "Self-Aligned Double Patterning Aware Pin Access and Standard Cell Layout Co-Optimization", *Proc. ACM International Symposium on Physical Design*, 2014, pp. 101-108.

[193] G. Yeric, *personal communication*, 2018.

[194] A. Z. Zelikovsky and I. I. Mandoiu, "Practical Approximation Algorithms for Zero- and Bounded-skew Trees, *SIAM Journal on Discrete Mathematics* 15(1) (2002), pp. 97-111.

[195] 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.

[196] 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.

[197] N. Y. Zhou, P. Restle, J. Palumbo, J. Kozhaya, H. Qian, Z. Li, C. J. Alpert and C. Sze, "PACMAN: Driving Nonuniform Clock Grid Loads for Low-Skew Robust Clock Network", *Proc. ACM International Workshop on System-Level Interconnect Prediction*, 2014, pp. 1-5.

[198] Cadence Innovus User Guide. `http://www.cadence.com`

[199] Cadence Quantus QRC Extraction Solution. `http://www.cadence.com`

[200] Cadence Tempus Timing Signoff Solution. `http://www.cadence.com`

[201] CAD/CAM/CAE Wallchart. `http://www.garysmitheda.com/wp-content/uploads/2015/05/All\_WC-15.pdf`

[202] `http://www.geosteiner.com/`

[203] IBM ILOG CPLEX. `www.ilog.com/products/cplex/`

[204] IMEC. `www2.imec.be`

[205] ISPD CNS Contest. `http://ispd.cc/contests/09/ispd09cts.html`

[206] International Technology Roadmap for Semiconductors. `http://www.itrs2.net/itrs-reports.html`

[207] ITRS 2013 Edition Report - Interconnect. `https://www.semiconductors.org/clientuploads/Research\_Technology/ITRS/2013/2013Interconnect.pdf`

[208] International Technology Roadmap for Semiconductors, Lithography Chapter, 2013. `http://www.itrs.net/`

[209] LEF DEF reference. `http://www.si2.org/openeda.si2.org/projects/lefdef`

[210] MATLAB. `http://www.mathworks.com/products/matlab/`

[211] MLPart. `http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/Partitioning/MLPart`

[212] OpenCores: Open Source IP-Cores. `http://www.opencores.org`

[213] OpenMP. `http://www.openmp.org/`.

[214] Are You (Really) Ready for Your Next Node? https://blogs.mentor.com/calibre/blog/2017/01/11/are-you-really-ready-for-your-next-node/

[215] Assessing The True Cost of Node Transitions. http://www.techdesignforums.com/practice/technique/assessing-the-true-cost-of-node-transitions/

[216] SDPA Official Page. http://sdpa.sourceforge.net/

[217] Si2 OpenAccess. http://www.si2.org/?page=69

[218] Synopsys Design Compiler User Guide. http://www.synopsys.com

[219] Synopsys HSPICE User Guide. http://www.synopsys.com

[220] Synopsys IC Compiler User Guide. http://www.synopsys.com

[221] Synopsys PrimeTime User Guide. http://www.synopsys.com

[222] UC Benchmark Suite for Gate Sizing. http://vlsicad.ucsd.edu/SIZING/bench/artificial.html