UNIVERSITY OF CALIFORNIA, SAN DIEGO

# Improved Physical Design and Signoff Methodologies for Better Integrated Circuit Design Quality

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

in

Electrical Engineering (Computer Engineering)

by

Jiajia Li

Committee in charge:

Professor Andrew B. Kahng, Chair
Professor Chung-Kuan Cheng
Professor Rajesh Gupta
Professor Patrick Mercier
Professor Tajana Rosing
Professor Steven Swanson

2017

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

_____

_____

_____

_____

_____

_____

Chair

University of California, San Diego

2017

DEDICATION

*I dedicate this thesis to my loving parents. Without their encouragement and support 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 support and generous advice throughout my Ph.D. study. I have truly learned a lot from his immense knowledge, serious and responsible work attitude, and passion for research.

I would like to thank my father Biao Li, my mother Yanhua Jiang, and my girlfriend Yun Sheng for their selfless support and encouragement. This journey would not have been possible without their sacrifice.

I would like to thank my fellow labmates in the UCSD VLSI CAD Laboratory (Wei-Ting (Jonas) Chan, Hyein Lee, Kwangsoo Han, Lutong Wang, Bangqi Xu, Ahmed Youssef, Tushar Shah, Sriram Venkatesh) and former lab members (Professor Seokhyeong Kang, Dr. Tuck-Boon Chan, Dr. Siddhartha Nath, Ilgweon Kang, Mulong Luo, Yaping Sun) for their assistance and enthusiastic discussions. I will remember the hard-working time and sleepless nights with them. Special thanks to Professor Seokhyeong Kang, Dr. Tuck-Boon Chan and Dr. Siddhartha Nath 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 Patrick Mercier, Professor Tajana Rosing and Professor Steven Swanson for their time, encouragement and insightful comments.

Last, but not least, I would like to thank my industrial collaborators (especially Dr. Bongil Park, Professor José Pineda de Gyvez, Dr. Hamed Fatemi, Jongpil Lee, Sorin Dobre, Nancy MacDonald, Dr. Kun Young Chung, Ajay Kapoor and Kristof Blutman) for their invaluable guidance and feedback in many of my research projects.

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

Chapter 2 contains reprints of Kun Young Chung, Andrew B. Kahng and Jiajia Li, "Comprehensive Optimization of Scan Chain Timing During Late-Stage IC Implementation", *Proc. ACM/IEEE Design Automation Conference*, 2016; Tuck-Boon Chan, Andrew B. Kahng, Jiajia Li, Siddhartha Nath and Bongil Park, "Optimization of Overdrive Signoff in High-Performance and Low-Power ICs", *IEEE Transactions on Very Large Scale Integration Systems* 23(8), 2015; 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/IEEE Design Automation Conference*, 2015; and Tuck-Boon Chan, Andrew B. Kahng, Jiajia Li and Siddhartha Nath, "Optimization of Overdrive Signoff", *Proc. Asia and South Pacific Design Automation Conference*, 2013. The dissertation author is the primary author of the papers.

Chapter 3 contains reprints of Kristof Blutman, Hamed Fatemi, Andrew B. Kahng, Ajay Kapoor, Jiajia Li, and José Pineda de Gyvez, "Floorplan and Placement Methodology for Improved Energy Reduction in Stacked Power-Domain Design", *Proc. Asia and South Pacific Design Automation Conference*, 2017; Andrew B. Kahng, Jiajia Li and Lutong Wang, "Improved Flop Tray-Based Design Implementation for Power Reduction", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2016; Andrew B. Kahng, Seokhyeong Kang, Jiajia Li and José Pineda de Gyvez, "An Improved Methodology for Resilient Design Implementation", *ACM Transactions on Design Automation of Electronic Systems* 20(4), 2015; and Andrew B. Kahng, Seokhyeong Kang and Jiajia Li, "A New Methodology for Reduced Cost of Resilience", *Proc. Great Lakes Symposium on Very Large Scale Integration*, 2014. Chapter 3 also contains the draft submitted to *IEEE Transactions on Very Large Scale Integration Systems*, Kristof Blutman, Hamed Fatemi, Andrew B. Kahng, Ajay Kapoor, Jiajia Li and Jose Pineda de Gyvez, "Logic Design Partitioning for Stacked Power Domains", 2017. The dissertation author is the primary author of the papers and the submitted draft.

Chapter 4 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; Sorin Dobre, Andrew B. Kahng and Jiajia Li, "Mixed Cell-Height Implementation for Improved Design Quality in Advanced Nodes", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2015; Tuck-Boon Chan, Andrew B. Kahng and Jiajia Li, "NOLO: A No-Loop, Predictive Useful Skew Methodology for Improved Timing in IC Implementation", *Proc. International Symposium on Quality Electronic Design*, 2014; and Tuck-Boon Chan, Andrew B. Kahng and Jiajia Li, "Reliability-Constrained Die Stacking Order in 3DICs under Manufacturing Variability", *Proc. International Symposium on Quality Electronic Design*, 2013. Chapter 4 also contains the draft submitted to *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Sorin Dobre, Andrew B. Kahng and Jiajia Li, "Design Implementation with Non-Integer Multiple-Height Cells for Improved Design Quality in Advanced Nodes". The dissertation author is the primary author of the papers and the submitted draft.

My coauthors (Kristof Blutman, Dr. Tuck-Boon Chan, Dr. Kun Young Chung, Sorin Dobre, Dr. Hamed Fatemi, Professor José Pineda de Gyvez, Kwangsoo Han, Professor Andrew B. Kahng, Professor Seokhyeong Kang, Ajay Kapoor, Jongpil Lee, Dr. Siddhartha Nath, Dr. Bongil Park and Lutong Wang listed in alphabetical order) have all kindly approved the inclusion of the aforementioned publications in my thesis.

VITA

| | |
|---|---|
| 1989 | Born, Taiyuan, Shanxi, China |
| 2011 | B.Sc., Software Engineering,<br>Shenzhen University, Shenzhen, Guangdong, China |
| 2013 | M.Sc., Electrical Engineering (Computer Engineering),<br>University of California, San Diego |
| 2015 | C.Phil., Electrical Engineering (Computer Engineering),<br>University of California, San Diego |
| 2017 | Ph.D., Electrical Engineering (Computer Engineering),<br>University of California, San Diego |

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

- Wei-Ting J. Chan, Andrew B. Kahng and **Jiajia Li**, "Revisiting 3DIC Benefit with Multiple Tiers", *Integration, the VLSI Journal*, 2017, to appear.
- Armin Alaghi, Wei-Ting J. Chan, John P. Hayes, Andrew B. Kahng and **Jiajia Li**, "Trading Accuracy for Energy in Stochastic Circuit Design", *ACM Journal on Emerging Technologies in Computing Systems*, 2017, to appear.
- Kristof Blutman, Hamed Fatemi, Andrew B. Kahng, Ajay Kapoor, **Jiajia Li**, and José Pineda de Gyvez, "Floorplan and Placement Methodology for Improved Energy Reduction in Stacked Power-Domain Design", *Proc. Asia and South Pacific Design Automation Conference*, 2017, pp. 444-449.
- Andrew B. Kahng, Hyein Lee and **Jiajia Li**, "Measuring Progress and Value of IC Implementation Technology", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2016, pp. 27:1-27:8.
- Andrew B. Kahng, **Jiajia Li** and Lutong Wang, "Improved Flop Tray-Based Design Implementation for Power Reduction", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2016, pp. 20:1-20:8.
- Kun Young Chung, Andrew B. Kahng and **Jiajia Li**, "Comprehensive Optimization of Scan Chain Timing During Late-Stage IC Implementation", *Proc. ACM/IEEE Design Automation Conference*, 2016, pp. 61:1-61:6.

- Wei-Ting Jonas Chan, Andrew B. Kahng and **Jiajia Li**, "Revisiting 3DIC Benefit with Multiple Tiers", *Proc. ACM International Workshop on System-Level Interconnect Prediction*, 2016, pp. 6:1-6:8.

- 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.

- Armin Alaghi, Wei-Ting J. Chan, John P. Hayes, Andrew B. Kahng and **Jiajia Li**, "Optimizing Stochastic Circuits for Accuracy-Energy Tradeoffs", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2015, pp. 178-185.

- Sorin Dobre, Andrew B. Kahng and **Jiajia Li**, "Mixed Cell-Height Implementation for Improved Design Quality in Advanced Nodes", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2015, pp. 854-860.

- Myung-Chul Kim, Jin Hu, **Jiajia Li** and Natarajan Viswanathan, "ICCAD-2015 CAD Contest in Incremental Timing-driven Placement and Benchmark Suite", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2015, pp. 921-926.

- Andrew B. Kahng, Seokhyeong Kang, **Jiajia Li** and José Pineda de Gyvez, "An Improved Methodology for Resilient Design Implementation", *ACM Transactions on Design Automation of Electronic Systems* 20(4) (2015), pp. 66:1-66:26.

- Tuck-Boon Chan, Andrew B. Kahng, **Jiajia Li**, Siddhartha Nath and Bongil Park, "Optimization of Overdrive Signoff in High-Performance and Low-Power ICs", *IEEE Transactions on Very Large Scale Integration Systems* 23(8) (2015), pp. 1552-1556.

- 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/IEEE Design Automation Conference*, 2015, pp. 26:1-26:6.

- Andrew B. Kahng, Hyein Lee and **Jiajia Li**, "Horizontal Benchmark Extension for Improved Assessment of Physical CAD Research", *Proc. Great Lakes Symposium on Very Large Scale Integration*, 2014, pp. 27-32.

- Andrew B. Kahng, Seokhyeong Kang and **Jiajia Li**, "A New Methodology for Reduced Cost of Resilience", *Proc. Great Lakes Symposium on Very Large Scale Integration*, 2014, pp. 157-162.

- Tuck-Boon Chan, Andrew B. Kahng and **Jiajia Li**, "NOLO: A No-Loop, Predictive Useful Skew Methodology for Improved Timing in IC Implementation", *Proc. International Symposium on Quality Electronic Design*, 2014, pp. 504-509.

- Tuck-Boon Chan, Andrew B. Kahng and **Jiajia Li**, "Toward Quantifying the IC Design Value of Interconnect Technology Improvements", *Proc. ACM International Workshop on System-Level Interconnect Prediction*, 2013, pp. 1-6.

- Tuck-Boon Chan, Andrew B. Kahng and **Jiajia Li**, "Reliability-Constrained Die Stacking Order in 3DICs under Manufacturing Variability", *Proc. International Symposium on Quality Electronic Design*, 2013, pp. 16-23.

- Tuck-Boon Chan, Andrew B. Kahng, **Jiajia Li** and Siddhartha Nath, "Optimization of Overdrive Signoff", *Proc. Asia and South Pacific Design Automation Conference*, 2013, pp. 344-349.

ABSTRACT OF THE DISSERTATION

**Improved Physical Design and Signoff Methodologies for Better Integrated Circuit Design Quality**

by

Jiajia Li

Doctor of Philosophy in Electrical Engineering (Computer Engineering)

University of California, San Diego, 2017

Professor Andrew B. Kahng, Chair

In the late CMOS era, integrated-circuit physical design and signoff face three major challenges – (i) complex operating conditions, (ii) low-power demand, and (iii) growing design margin. Future scaling of designs and the continuation of Moore's Law itself require better physical design optimization and signoff methodologies. Toward this end, this thesis presents novel optimization techniques and signoff methodologies to respectively address these challenges in three main thrusts.

In modern SoC implementations, multi-mode design is commonly used to achieve better circuit performance and power across voltage scaling, "turbo" and other operating modes. Furthermore, PVT variations result in a large number of corners for circuit design and signoff. To mitigate the impact of complex operating conditions and corner explosion, in the *multi-mode multi-corner optimization* thrust, this thesis presents approaches to optimize signoff corner se-

lection, reduce skew variation in clock network, and perform scan timing optimization without causing any QoR degradation in functional mode.

Energy and battery lifetime constraints induce new and critical challenges to IC designs, especially for mobile and "Internet of Things" (IoT) applications. To achieve power autonomy in the era of a slowing Moore's law, low-power techniques must be exploited. To minimize design power and energy, in the *low-power optimization* thrust, this thesis presents a stacked power domain scheme to align SoC power domain voltages with battery voltages for power delivery efficiency and battery lifetime improvements, a novel flop tray generation technique for clock power reduction, and a low-cost resilient design flow to enable better than worst-case design for energy savings.

The 2013 ITRS update of system driver models reveal a "scaling gap" since 2008. One root cause of the density scaling slowdown is the growing design margins due to variability, reliability, etc. To reduce the design margins and to pursue design-based equivalent scaling [31], in the *mixed-fabric optimization* thrust, this thesis describes the concept of "mixed-fabric optimization" and presents several novel optimization techniques for improved design performance, power, area, reliability and turnaround time. First, we propose an optimization flow for implementation of design blocks with mixed non-integer multiple cell heights, achieving an improved tradeoff of performance, power and area. Second, we exploit the dual-$V_{th}$ libraries and propose a "no-loop" predictive useful skew optimization flow. Last, we integrate dies with different process conditions in a 3DIC, and apply mix-and-match-aware design optimization to improve performance and reliability of 3DICs.

# Chapter 1

# Introduction

Due to the slowdown of density scaling and the existence of a "design capability gap" as highlighted in the 2013 ITRS (International Technology Roadmap for Semiconductors) System Drivers Chapter analyses [225], the ability of designers to leverage the "available scaling" (e.g., of design quality) afforded by process and device has become weaker. There is now an urgent need for *design-based equivalent scaling* [31] that will compensate for the scaling gap through better design techniques, more accurate modeling, and other improvements. As key steps in IC design implementation, *physical design* and *signoff* have significant impacts on the performance, power, area and cost (PPAC) of designs. However, existing physical design and signoff methodologies are severely challenged by increased design complexity, corner explosion, low-power demand, advanced design rules, reliability issues, etc. Inability to address these challenges will result in degraded IC design quality. Thus, future scaling of designs and the continuation of Moore's Law itself require better physical design optimization and signoff. Towards this goal, this thesis presents improved physical design and signoff methodologies to address existing and future challenges, and to enable future design-based equivalent scaling.

## 1.1 Slowdown of Density Scaling and Need for Design-Based Equivalent Scaling

Over the past decades, IC designers have continued to extract value from Moore's Law by riding the density scaling "wave". However, as noted in the 2013 edition of ITRS [225], even as patterning technologies have continued to deliver (at least until the year 2013) "available" Moore's Law scaling (i.e., geometric pitch scaling), the "realized" transistor density scaling in

actual products has slowed down from the traditional Moore's-Law density scaling of 2× per technology node to around 1.6× per technology node since ∼2008 (see Figure 1.1). The slowdown of density scaling indicates that the benefits from technology scaling are significantly less than what has been expected according to historical trends. As a result, it is more challenging to achieve the "20% speed, 20% power and 20% density" that comprise the traditionally-understood threshold for necessary benefits from a new technology node [101]. Such inconsistency between the "available" density scaling and "realizable" density scaling in actual products may be attributed to a *design capability gap* caused by failure to address process variation, reliability requirements, patterning constraints, and other challenges.

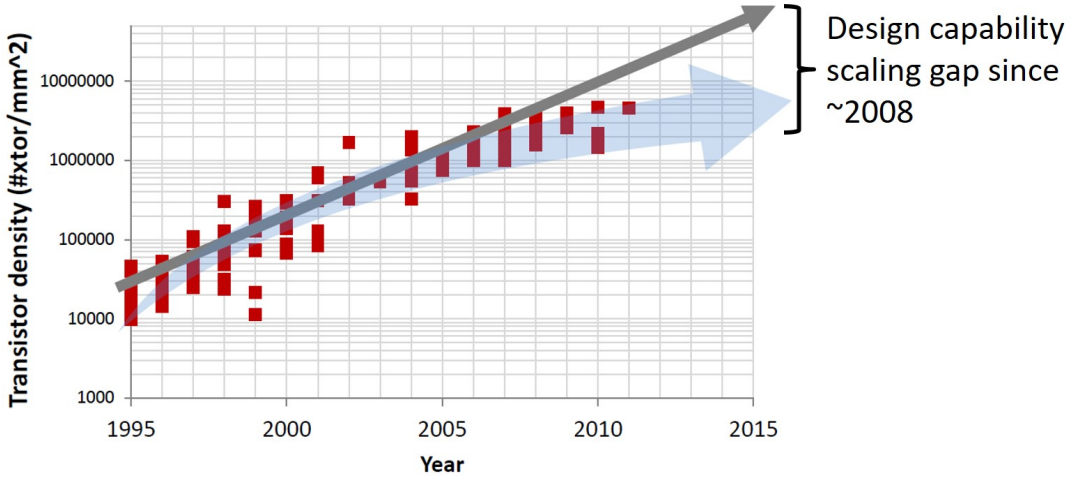



**Figure 1.1**: Gap between "available" density scaling (gray arrow) and "realizable" density scaling in MPU products (red squares), adapted from [100].

The most viable remaining path to mitigating the design capability gap is through *design-based equivalent scaling*, which encompasses better design methodology, optimized signoff constraints, more accurate modeling to improve model-hardware correlation, and other improvements to design technology and design enablement. The 2013 edition of ITRS [225] projects that for server and desktop processors (MPU), design-based equivalent scaling will need to recover one entire node from 2013 to 2019, and one node of scaling from 2013 to 2020 for system-on-chips (SoCs) [100]. Therefore, physical design and signoff, which are key steps in IC design and have significant impact on design QoR, need to be improved to enable design-based equivalent scaling.

Evidence of the urgent need for design-based equivalent scaling and better physical design and signoff is seen in the recent DARPA "Circuit Realization at Faster Timescales (CRAFT)" program, which aims to reduce the design time of complex mixed-signal SoCs in sub-20$nm$

technologies from years to months without any loss of "performance at power" (PAP) [219]. Figure 1.2 (adapted from [103]) shows hypothesized steps towards the goal, which include a number of physical design and signoff improvements. Given that Moore's Law classically corresponds to "one week = one percent", the targeted 100- week reduction of design time (iso-PAP) is an enormously valuable "moonshot" [110]. Works reported in this thesis (e.g., "optimization of signoff mode selection" in Chapter 2, "predictive, one-pass useful skew flow" in Chapter 3, and "mixed cell-height placement" in Chapter 4) address various aspects of #1, #2 and #3 in Figure 1.2.

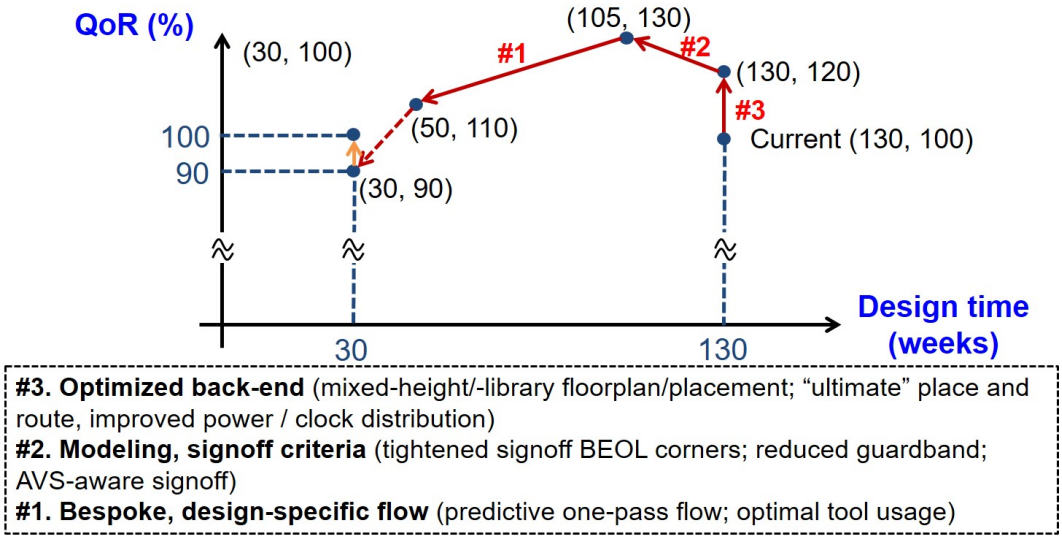



**Figure 1.2**: Three hypothesized steps to reduce the design time of complex mixed-signal SoCs in sub-20$nm$ technologies from 130 weeks to 30 weeks [219].

## 1.2 Challenges in Physical Design and Signoff

Besides the urgent need for better physical design and signoff to achieve design-based equivalent scaling, the physical design and signoff tasks inherently face many challenges due to increased design complexity, process variation, and number of operating conditions. The following three subsections summarize three major challenges to physical design and signoff.

### 1.2.1 Complex Operating Conditions and Corner Explosion

Modern SoCs typically exploit complex operating scenarios to maximize performance and reduce power consumption. As illustrated in Figure 1.3, high-performance, low-power designs can use frequency overdrive at elevated supply voltages (i.e., overdrive mode) to obtain

better performance, and operates at a lower supply voltage for power saving in nominal mode. However, multi-mode operation requires multi-mode design and signoff, that is, timing must be closed at multiple modes and power optimization must comprehend the duty cycle of each mode.

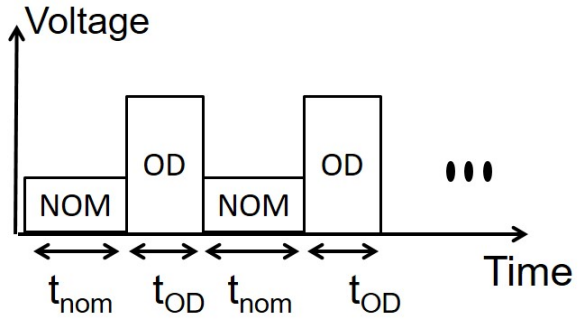



**Figure 1.3**: Example of multi-mode operation. OD = overdrive mode. NOM = nominal mode.

Further challenges arise from the industry's need for low-cost, high-quality test strategies for complex, high-performance SoCs. IC designs typically have test modes (e.g., scan, at-speed, BIST) for which timing and power optimization can also be critical. Optimizations in test modes risk degradation of design quality (e.g., timing and power) in function mode. Therefore, new test-mode optimizations are needed that comprehend timing and power impact in function mode.

Moreover, process, voltage and temperature (PVT) variations result in a large number of corners for design and signoff. For instance, according to the carrier mobilities (e.g., typical (T), fast (F) and slow (S)) of NMOS and PMOS devices, the front-end-of-line (FEOL) has FF, SS, FS, TT, etc. process corners. According to the parasitic variation in the back-end-of-line (BEOL), there are Cbest, Cworst, RCbest, etc. process corners. Taking into account different combinations of function modes and test modes, along with the many PVT corners, a complex SoC will have a very large number of views for signoff (i.e., a "corner explosion"). The temperature reversal effect – where the device becomes slower at a low temperature when the supply voltage is lower than a threshold – further increases the number of signoff corners.

The corner explosion significantly increases the difficulty of physical design and signoff. Given that the selection of signoff modes and corners has significant impact on design quality, one critical problem is how to optimize the selection of signoff modes and corners to avoid over design. Another major challenge in multi-mode multi-corner design is the "ping-pong" effect, that is, fixing timing issues at one corner leads to violations at other corners. The "ping-pong" effect can induce large area and power overheads and severely increase design turnaround time (TAT), and is mainly caused by delay variation in datapath and clock paths across corners. New

optimization techniques are needed to address the "ping-pong" effect for better design quality and shorter design time.

### 1.2.2 Demand for Low-Power Designs

Power and energy are well-understood as the ultimate "Grand Challenge" for the semiconductor industry, as explicitly noted in the ITRS since the early- to mid-2000s. Thus, power reduction has long been one of the primary goals for IC design. Today, especially for mobile and IoT ("Internet of Things") applications, energy/power and battery lifetime constraints present extremely difficult challenges to IC designers. Traditional physical design applies gate sizing, $V_{th}$ swapping, clock gating, power gating, and multiple power domains to achieve power reduction while satisfying performance requirements. (Additional levers for low-power design can be found in the Low-Power Design Technology Innovation Roadmap provided by the 2011 ITRS Design Chapter [224].) As the requirement for power reduction has become more critical to product competitiveness, more low-power optimization techniques have been introduced. However, these low-power techniques increase complexity for physical design, which can incur area and power overheads. Therefore, new low-power techniques must ensure that their power benefits outweigh their costs.

As an example, the clock network typically consumes a large portion of SoC power due to its high switching activity. Usage of flop trays (also known as multi-bit flip-flops) can significantly reduce the number of sinks in a clock tree, and thus the number of buffers as well as clock power. However, clustering of flip-flops induces additional placement constraints on their fanin/fanout datapaths, which can cause power and area overheads. We therefore must ensure that the clock power reduction is larger than the power overhead on datapaths. As another example, resilient flip-flops, which are error-tolerant, enable better than worst-case (BTWC) design and power reduction on datapaths. However, the power of a resilient flip-flop is higher than that of a conventional flip-flop. Therefore, the resilient design optimization must ensure that the power reduction on datapaths outweighs the power penalties from resilient flip-flops.

Furthermore, to achieve power autonomy in the era of a slowing Moore's law, new low-power techniques must be exploited. While many existing low-power techniques have concentrated on the circuit side of system design, power management techniques have received growing attention due to the importance of power efficiency. Notably, the misalignment of battery voltages compared to scaled core voltages causes inefficiencies that present significant opportunities for power saving.

### 1.2.3 Growing Design Margins

To account for variability (e.g., process variation, dynamic voltage drop) and reliability (e.g., electromigration (EM), negative-bias temperature instability (NBTI)), designers tend to insert margins (i.e., overdesign) in the signoff analysis to ensure correctness of operation. As design complexities and process variations increase, design margins also grow. Figure 1.4 illustrates how growing design margins counteract the benefits of technology scaling. If the potential benefits from technology scaling are already small, then the design margins become even more costly to apply.

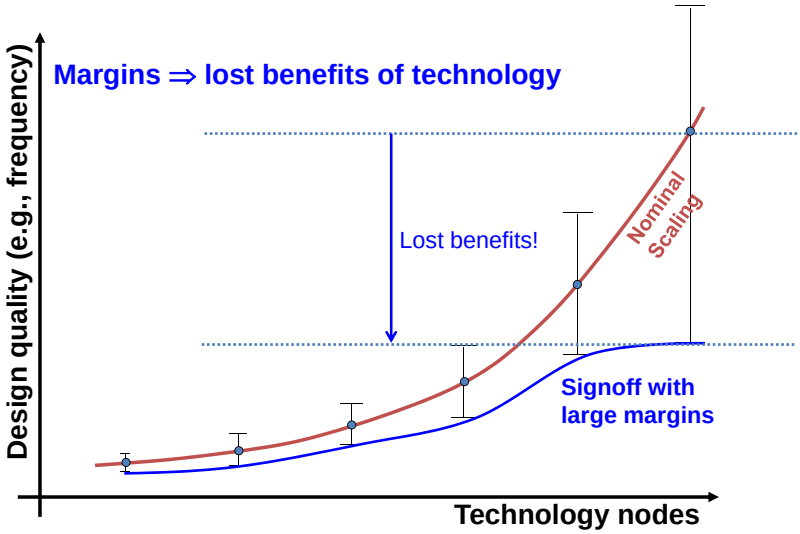



**Figure 1.4**: Growing design margins degrade the benefits of technology scaling.

## 1.3 This Thesis

To pursue design-based equivalent scaling and to address the existing challenges in modern SoC physical design and signoff, this thesis presents innovative optimization techniques and signoff methodologies. Figure 1.5 illustrates the scope and organization of this thesis, in which three main thrusts respectively address three major challenges:

- Multi-mode multi-corner optimization;
- Low-power optimization; and
- Mixed-fabric optimization.

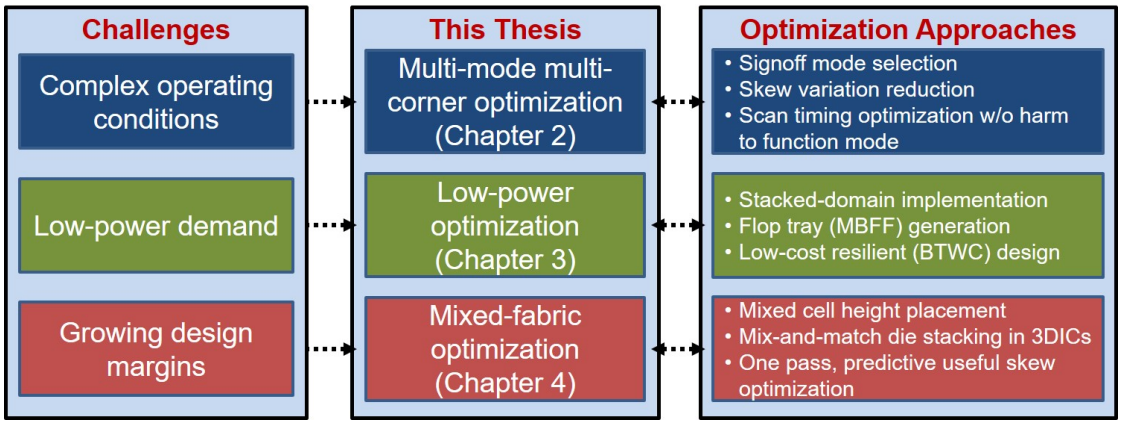



**Figure 1.5**: Scope and organization of this thesis.

To mitigate the impact of complex operating conditions and corner explosion, in the *multi-mode multi-corner optimization* thrust this thesis presents approaches to optimize signoff corner selection, reduce skew variation in the clock network, and perform scan timing optimization without causing any quality-of-results (QoR) degradation in functional mode.

To minimize design power and energy, in the *low-power optimization* thrust this thesis presents a stacked power domain scheme to align SoC power domain voltages with battery voltages for power delivery efficiency and battery lifetime improvements; a novel flop tray (i.e., multi-bit flip-flop) generation technique for clock power reduction; and a low-cost resilient design flow to enable better than worst-case design for energy saving.

To reduce design margins and improve design quality, in the *mixed-fabric optimization* thrust this thesis describes the concept of "mixed-fabric optimization" and presents several novel optimization techniques for improved design performance, power, area, reliability and turnaround time. First, we propose an optimization flow to implement design blocks with mixed non-integer multiple cell heights for a better tradeoff among performance, power and area. Second, we exploit the dual-$V_{th}$ libraries and propose a "no-loop" predictive useful skew optimization flow. Last, we integrate dies with different process conditions in a 3DIC and apply mix-and-match-aware design optimization to improve performance and reliability of 3DICs.

The remainder of this thesis is organized as follows.

- Chapter 2 presents three distinct optimization methodologies for circuit optimization in the multi-mode and multi-corner context. First, using our concept of "mode dominance" as a guideline, we propose a scalable, model-based adaptive search methodology to explore the design space for signoff mode (i.e., (operating frequency, voltage) pair) selection. Our

proposed methodology is duty cycle-aware in its minimization of lifetime energy. 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 to address the "ping-pong" effects (i.e., fixing violations at one corner can lead to violations at other corners). 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 propose two techniques for late-stage scan chain timing optimization with negligible timing, area and power impact in function mode. Specifically, we propose skew-aware scan ordering to minimize the number of hold buffers, and DVD-aware gating insertion to improve scan shift timing slacks.

- Chapter 3 presents three distinct techniques for low-power optimization, which address the low-power requirement in three aspects – system, clock and datapath. First, we present an optimization framework for stacked-domain designs. Based on an initial placement solution, we apply a flow-based partitioning that is aware of multiple operating scenarios, cell placement, and timing-critical paths to partition cells into two power domains with balanced cross-domain current and minimized number of inserted level shifters. We further propose heuristics to define regions for each power domain so as to minimize placement perturbation, as well as a dynamic programming-based method to minimize the area cost of power domain generation. Second, we propose an optimization flow to generate and place flop trays from a library of arbitrary given sizes and aspect ratios (ARs), to achieve clock network power reduction. Our optimization starts with an initial placement solution using only single-bit flops. It then performs capacitated K-means clustering to generate solutions with different flop tray sizes and ARs. Our optimization is aware of flop tray sizes and ARs, as well as timing-critical start-end pairs. Third, we use resilient design with minimized overheads to reduce power on datapaths. Our methodology uses two levers: selective-endpoint optimization (i.e., sensitivity-based margin insertion) and clock skew optimization. We integrate the two optimization techniques in an iterative optimization flow which comprehends toggle rate information and the tradeoff between cost of resilience and margin on combinational paths.

- Chapter 4 presents a new concept of "mixed-fabric optimization" for improved design quality, along with three examples. Here, *mixed fabric* indicates standard cells with dif-

ferent $V_{th}$ flavors or track heights, or tiers with different process corners in a 3DIC, etc. First, we propose a novel physical design optimization flow to implement design blocks with mixed non-integer multiple-height cells in a fine-grained manner. Our optimization resolves the "chicken-and-egg" loop between floorplan site definition and the optimized choices of cell heights after placement. The optimization also comprehends the constraints and costs of mixing cells of different heights (e.g., the "breaker cell" area overheads of row alignment between sub-blocks of 8T and 12T cell rows). Second, we propose NOLO, a simple, "no-loop" predictive useful skew flow with dual-$V_{th}$ libraries that applies useful skew at the post-synthesis stage within a one-pass chip implementation. Third, we study the "mix-and-match" of multiple stacked die, according to binning information, to improve overall product yield. We study die-stacking optimizations to improve 3DIC product reliability, as well as performance improvements in 3DIC implementation that leverage *a priori* knowledge of mix-and-match die stacking during manufacturing. Regarding the design-stage optimization for mix-and-match die stacking, 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.

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

# Chapter 2

# Multi-Mode Multi-Corner Optimization

Modern SoCs typically exploit complex operating scenarios to maximize performance as well as reduce lifetime energy. For example, a baseband processor SoC might be designed to meet performance and low-power criteria across turbo, nominal and supply voltage-scaled operating modes. In addition, delay (especially clock skew) variation across different PVT (process, voltage, temperature) corners and additional signoff criteria incur difficulties for modern SoC design. Further, challenges arise in the context of cost-efficient testability, notably, scan chain optimization without design quality (e.g., timing, power) degradation in function mode.

This chapter presents three distinct methodologies for circuit optimization in the multi-mode and multi-corner context. First, using our concept of "mode dominance" as a guideline, we propose a scalable, model-based adaptive search methodology to explore the design space for signoff mode (i.e., (operating frequency, voltage) pair) selection. Our proposed methodology is duty cycle-aware in its minimization of lifetime energy. Results in both $65nm$ and $28nm$ technologies show that our proposed methodology provides up to $> 8\%$ improvement in performance, for given $VDD$, area and power constraints, compared to the conventional "signoff and scale" method. Further, the signoff modes determined by our methods result in $< 4\%$ overhead in power compared with the optimal signoff modes. 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; this addresses the well-known challenge of "ping-pong" effects in timing closure and optimization (i.e., fixing violations at one corner can lead to violations at other corners). The global optimiza-

tion 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 propose two techniques for late-stage scan chain timing optimization with negligible timing, area and power impact in function mode. Specifically, we propose skew-aware scan ordering to minimize the number of inserted hold buffers, and DVD-aware gating insertion to improve scan shift timing slacks. Our optimizations at the post-CTS and post-routing stages reduce hold buffers by up to 82%, and DVD-induced timing degradation by up to 58%.

## 2.1 Optimization of Overdrive Signoff in High-Performance and Low-Power ICs

In the era of heterogeneous multi-core SOCs, the performance of single-threaded operations limits the overall speedup of applications. Designers use frequency overdrive at elevated voltages to obtain better performance in consumer electronic devices [57]. An *operating mode* (for simplicity, *mode*) is defined by an (operating frequency, voltage) pair. Devices typically operate at two or three modes, e.g., supply voltage-scaled (SVS), nominal and turbo (overdrive). The nominal and SVS modes correspond to a lower operating voltage and a lower frequency, whereas the overdrive mode corresponds to a higher operating voltage and a higher frequency. We define the average power ($P_{avg}$) for a circuit with both nominal and overdrive modes as

$$P_{avg} = r \times P_{OD} + (1 - r) \times P_{nom}, \quad 0 < r < 1 \tag{2.1}$$

where the duty cycle $r$ is the total overdrive time normalized to the total lifetime. $P_{OD}$ and $P_{nom}$ are the circuit power at overdrive and nominal modes, respectively.

We define the *signoff mode design space* (or *design space*) as the set of feasible signoff mode combinations. A *point* in this design space specifies $m$ (frequency, voltage) pairs for $m$-mode signoff, where $m \geq 1$. Signing off at different points in a design space results in circuits with different performance, power and area. Figure 2.1 shows that the average power of a given design can vary by up to 26% across 40 different definitions of the overdrive mode, with a fixed nominal mode. Even when the overdrive frequency is fixed, the average power can vary by up to 12% for different overdrive voltages. Circuit power varies with signoff voltage because when

signing off at a lower voltage, buffer insertion to meet timing constraints leads to higher power. On the other hand, although circuit area decreases with a higher signoff voltage, power increases with operating voltage. The optimal signoff voltage must comprehend this tension.

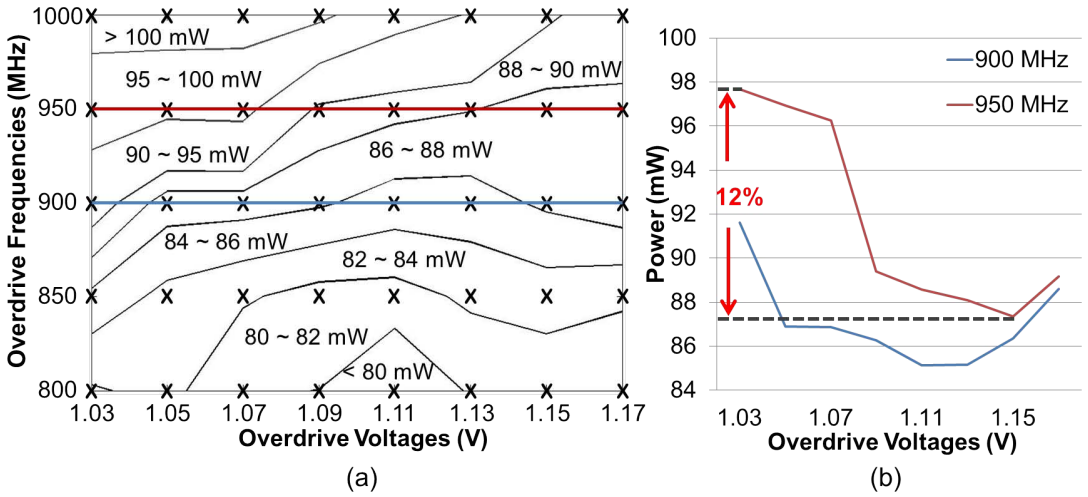



**Figure 2.1**: $P_{avg}$ of circuits signed off at the same nominal mode ($500MHz$, $0.9V$) but 40 different overdrive modes. Design: *AES* [230]. Technology: foundry $65nm$. Corner: FF/125°$C$. $r$ = 10%.

Figure 2.1 suggests that we can reduce design cost by carefully optimizing the signoff modes. Accordingly, in this work, we study the *signoff mode optimization problem*, which seeks the optimal nominal and overdrive modes with respect to optimization objectives and constraints. Similar multi-mode signoff optimization has been studied by [104]. However, our work achieves greater insight into the basic tradeoff between frequency and voltage at the circuit level. As an extension to the previous work [30], we propose a more efficient and effective methodology for multi-mode signoff optimization.

Our contributions are summarized as follows.

1. Based on the property of *equivalent dominance*, we propose a global optimization flow (using *model-based adaptive search*) to analyze and identify the dominant modes *before* circuit implementation.

2. Our proposed methodologies lead to $> 8\%$ and $6\%$ performance improvements compared to the traditional "signoff and scale" and previous work [30], respectively, while maintaining similar power and area.

3. The proposed methodologies can successfully determine signoff modes that reduce lifetime energy for a given duty cycle.

### 2.1.1 Dominance of Modes

To analyze the dominance of modes, we define the concept of *design cone* as follows.

**Definition:** The *design cone* of a given mode ***M*** is the union of (*maximum* frequency, voltage) operating modes for all feasible circuit implementations that are signed off at mode ***M***.

Figure 2.2 illustrates the design cone ***R*** of mode ***A***. Circuits signed off at mode ***A*** will have their own frequency vs. voltage tradeoffs.[1] At a given voltage, the boundary of the design cone is determined by the upper and lower bounds of the maximum frequency that is achievable by circuits signed off at mode ***A***.

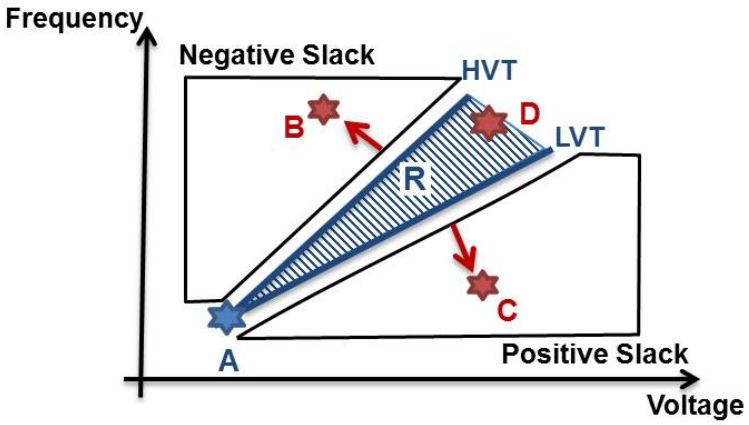



**Figure 2.2**: Illustration of the design cone of mode ***A*** (the shaded region).

Given the design cone of mode ***A***, a mode ***C*** $(f_C, V_C)$ has a *positive slack* (respectively, a *negative slack*) with respect to mode ***A*** if $f_C$ is below (respectively, above) the lower (respectively, upper) boundary of design cone at $V_C$. Since the positive slack can be exploited to reduce power [30], we say that the existence of positive slack indicates *overdesign*.

We further define the *dominance of modes* as follows.

**Definition:** Given two modes $M_1$ and $M_2$, if mode $M_2$ shows positive slacks with respect to mode $M_1$, we define mode $M_1$ as the *dominant mode*, and mode $M_2$ as the *dominated mode*.

**Definition:** Given two modes $M_1$ and $M_2$, if mode $M_1$ is in the design cone of mode $M_2$ and mode $M_2$ is in the design cone of mode $M_1$, we say that modes $M_1$ and $M_2$ exhibit *equivalent dominance*.

In Figure 2.2, mode ***A*** is dominant and mode ***C*** is dominated. The dominant mode has tighter constraints, thus determining the properties (e.g., area, gate count, total capacitance) of

[1] Boundaries of a design cone can be affected by threshold voltage, gate type and/or wire resistance. In this work, we determine the boundaries using frequency vs. voltage curves of high (HVT) and low threshold voltage (LVT) cells since other parameters have little impact at $65nm$ technology [30].

a design in a multi-mode signoff. Moreover, when *equivalent dominance* holds for multiple signoff modes, we expect resulting design properties that are similar to those when the design is signed off at each mode individually.

Based on the equivalent dominance concept, we state the following Lemmas.

**Lemma 1:** If two modes do not exhibit equivalent dominance, then each mode is outside of the design cone of the other.

**Proof (by contradiction):** Assume toward a contradiction that the claim is false, i.e., modes $M_1$ and $M_2$ do not exhibit equivalent dominance, but one mode ($M_1$) is located in the design cone of the other ($M_2$). According to the definition of design cone, any point in the design cone of $M_2$ lies on a frequency vs. voltage tradeoff curve corresponding to a circuit signed off at $M_2$. Therefore, there is at least one circuit with a frequency vs. voltage tradeoff curve that passes through both $M_1$ and $M_2$. This means that $M_2$ is also in the design cone of $M_1$. Hence, modes $M_1$ and $M_2$ exhibit equivalent dominance, contradicting our initial assumption. □

**Lemma 2:** Multi-mode signoff at modes which do not exhibit pairwise equivalent dominance leads to overdesign.

**Proof:** If a set of modes does not exhibit pairwise equivalent dominance, then there exist two modes for which equivalent dominance does not hold. According to *Lemma 1*, neither mode is located in the design cone of the other. Then, one of the modes must be dominant, and the other dominated. By definition of a dominated mode, the circuit being implemented at the dominated mode will have positive timing slack. Regardless of the duty cycle, positive timing slack indicates overdesign (cf. Figure 2.2). Therefore, at least one mode will be overdesigned if a set of modes does not exhibit pairwise equivalent dominance. □

**Lemma 3:** Mutual pairwise equivalent dominance holds among $m$ ($m \geq 3$) modes if and only if the modes are collinear in the $(v, f)$ space for signoff.

**Proof** (by induction on $m$):

**Base Case:** ($m$ = 3) Per the discussion in [30], the frequency vs. voltage tradeoff curve for a given circuit is taken to be a straight line. Further, any one circuit implementation corresponds to only one frequency vs. voltage tradeoff curve. Thus, signoff with any two out of the three modes will determine a frequency vs. voltage tradeoff curve (corresponding to the resultant circuit). Whenever the third signoff mode is below the frequency vs. voltage tradeoff curve of the other two modes, the supply voltage can be reduced to achieve lower power and still meet timing constraints; this corresponds to overdesign. And, whenever the third mode is above the frequency vs. voltage tradeoff curve of the other two modes, there must be a timing violation

at the third mode; this corresponds to a failed design. Therefore, the third signoff mode must be on the frequency vs. voltage tradeoff curve of the other two modes (i.e., the three modes are collinear) for equivalent dominance.

**Inductive Step:** By assuming that when any $k$ modes, $k \geq 3$ exhibit mutual pairwise equivalent dominance, they are collinear in the design space. We wish to prove that any ($k$+1) modes with mutual pairwise equivalent dominance must be collinear. Pick any subset $S_1$ of $k$ modes and let $A$ be the remaining $(k+1)^{st}$ mode. The modes in $S_1$ are collinear. Pick any subset $S_2$ of $k$ modes that includes $A$. The modes in $S_2$ are collinear. Since $|S_1 \cap S_2| \geq 2$ all $(k+1)$ modes are collinear. □

Figure 2.3 illustrates an example where four modes exhibit equivalent dominance. The desired design space (i.e., without incurring overdesign) for signoff is the line ***D-A-B-C***. We note that marketing or other product requirements may well lead to multiple modes that are not collinear in the design space. In such a situation, there must be overdesign with respect to at least one of the modes. A methodology to define signoff modes to minimize some global measure of overdesign is beyond our present scope, and we focus on scenarios involving just two modes in our work.

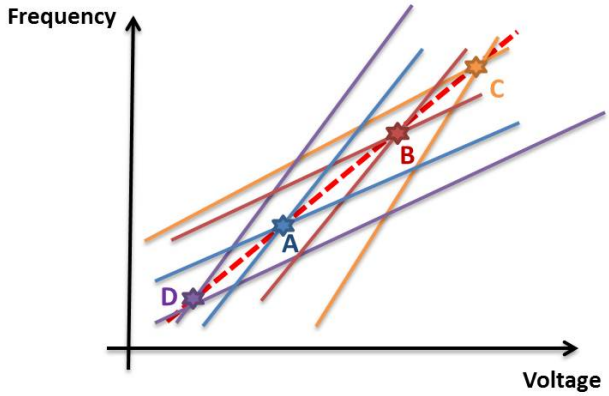



**Figure 2.3**: Four modes exhibit equivalent dominance. The desired design space is the line ***D-A-B-C***.

### 2.1.2 Problem Formulation

To sign off a circuit that operates at both nominal and overdrive modes, we need to select four parameters: nominal frequency ($f_{nom}$) and voltage ($V_{nom}$), and overdrive frequency ($f_{OD}$) and voltage ($V_{OD}$). In this work, we study the problems where two parameters are given and two parameters must be determined, as follows.

**The FIND_OD problem.** Given $f_{nom}$, $V_{nom}$ and $r$, and upper bounds on $V_{OD}$, $P_{avg}$ and $P_{OD}$, determine $f_{OD}$ and $V_{OD}$ such that $f_{OD}$ is maximized.

**The FIND_NOM problem.** Given $f_{OD}$, $V_{OD}$ and $r$, and upper bounds on $P_{avg}$ and $P_{OD}$, determine $f_{nom}$ and $V_{nom}$ such that $f_{nom}$ is maximized.

**The FIND_VOLT problem.** Given $f_{nom}$, $f_{OD}$ and $r$, and upper bounds on $V_{OD}$ and $P_{OD}$, determine $V_{nom}$ and $V_{OD}$ such that $P_{avg}$ is minimized.

**The FIND_FREQ problem.** Given $V_{nom}$, $V_{OD}$ and $r$, and upper bounds on $P_{avg}$ and $P_{OD}$, determine $f_{nom}$ and $f_{OD}$ such that $(1-r) \times f_{nom} + r \times f_{OD}$ is maximized.

### 2.1.3 Efficient Exploration of Design Space

The key challenge in signoff mode optimization is to efficiently search for the desired modes using a small number of implementation trials. To this end, we propose a *model-based adaptive search* to explore the design space for signoff mode selection. In the model-based adaptive search, new solutions are determined using models, which are updated or derived from implementations with previous solutions [91]. Figure 2.4 shows our adaptive search flow. We construct our power model based on initial samples. Using the power model, we *predict* the optimal signoff mode and sample (i.e., run SP&R) at the predicted mode. We iteratively sample and update the power model until the flow converges.

#### Power Model

Following industry standard models (Liberty format) and tools (e.g., [240]), we model circuit power as being comprised of three components – switching ($P_{sw}$), internal ($P_{int}$) and leakage ($P_{leak}$). Our power model uses the following *circuit properties*: load capacitance ($C_{load}$), which includes wire capacitance and the capacitance of input pins driven by nets [113]; total gate capacitance ($C_{gate}$); and percentage of cell instances with different $V_{th}$ flavors (i.e., $Pct_{\{LVT,HVT,NVT\}}$). As we observed in Figure 2.1(b), circuit power exhibits unimodal behavior with varying signoff voltage. This suggests that we model power as a second-order polynomial of the signoff voltage. We also observe below that power linearly depends on circuit properties. Therefore, we also model the circuit properties as second-order polynomials of the signoff voltage or frequency,[2] as

$$c_{load} = q_1 \times V^2 + q_2 \times V + q_3 \tag{2.2}$$

[2]Note that circuit properties may not always behave as second-order relations with the signoff voltage or frequency, which can lead to errors in power estimation. However, our experimental results show that the estimation error is less than 10%.

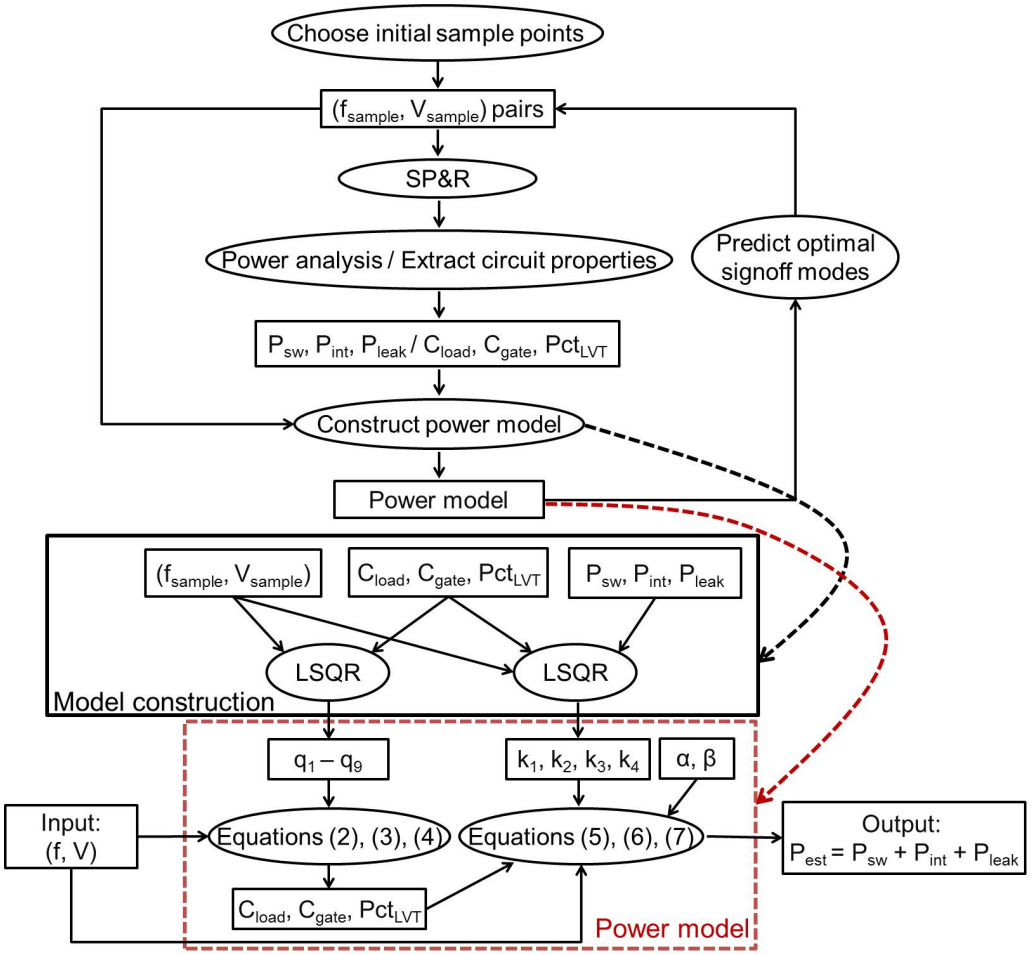



**Figure 2.4**: Our adaptive search flow (top) and power model (dotted box).

$$c_{gate} = q_4 \times V^2 + q_5 \times V + q_6 \tag{2.3}$$

$$Pct_{LVT} = q_7 \times V^2 + q_8 \times V + q_9 \tag{2.4}$$

where $q_1$–$q_9$ are fitting parameters. Equations (2.2)–(2.4) are used when $V$ is the variable in adaptive search; when $f$ is the variable, we use $f$ in place of $V$. We then use the estimated circuit properties to model power components.

**Net switching power.** We model net switching power as

$$P_{sw} = k_1 \times \alpha \times C_{load} \times f \times V^2 \tag{2.5}$$

where $\alpha$ is the switching activity factor; $f$ and $V$ are operating frequency and supply voltage, respectively; and $k_1$ is a fitting parameter used during adaptive search.

**Internal power.** Since the internal power mainly consists of the short circuit power, based

on [153][188], we model internal power as

$$P_{int} = k_2 \times \alpha \times C_{gate} \times f \times V^2 \tag{2.6}$$

where $k_2$ is a fitting parameter used during adaptive search.

**Leakage power.** We use gate capacitance as a parameter to fit leakage power [128]. Further, we use the functional form $e^{\beta \times V}$ ($\beta$ is a fitting parameter depending on technology and threshold voltages of transistors) to model the leakage current. Due to dominant impact of LVT cells on leakage power as compared to NVT and HVT cells, we also use percentage of LVT cell instances in our model. We model leakage power as

$$P_{leak} = V \times C_{gate} \times (k_3 \times Pct_{LVT} + k_4) \times e^{\beta \times V} \tag{2.7}$$

where $k_3$ and $k_4$ are fitting parameters for adaptive search.

We emphasize that Equations (2.5)–(2.7) are not for accurate power calculation. Rather, they are based on chosen parameters for power estimation within our adaptive search. In multi-mode signoff, since the circuit is mainly determined by the dominant mode, which has the tightest timing constraints, we use the dominant mode to model $C_{load}$, $C_{gate}$ and $Pct_{LVT}$. However, when two or more modes exhibit equivalent dominance, we choose the modes that are not yet fixed and among these modes we choose the mode with the largest duty cycle for power modeling as it has the greatest impact on $P_{avg}$.

### Adaptive Search

We now propose two generic adaptive search flows for signoff mode selection (shown in Algorithm 1). We then extend them to solve the problems described in Section 2.1.2.

Given a signoff frequency ($f$), we use the MIN_POWER flow to search for the signoff voltage ($V$) that minimizes circuit power ($P$). The inputs $V_{min}$ and $V_{max}$ are user-specified minimum and maximum signoff voltages, respectively. $V_{stop}$ is a stopping criterion for adaptive search. We first construct our power model based on three initial samples (Lines 1–3). Based on the obtained power model, we predict the optimal signoff voltage to minimize power (Line 6). We then run SP&R with the predicted signoff voltage and update the power model (Lines 7–9). If the change in the value of the estimated optimal signoff voltage is less than $V_{stop}$, the adaptive search terminates. Otherwise, more accurate estimation of the optimal signoff voltage is predicted from the improved power model.

Given a signoff voltage ($V$), we use the MAX_FREQ flow to search for the maximum

signoff frequency ($f$) under particular power constraints ($P_{max}$). The input $f_{min}$ is the predefined lower bound on performance and $f_{max}$ is the maximum achievable frequency with voltage $V$. $f_{min}$ and $f_{max}$ define the range of signoff frequency selection. $f_{stop}$ is a stopping criterion.

**Algorithm 1** Adaptive search flows.

**Procedure** MIN_POWER ($f$, $V_{min}$, $V_{max}$, $V_{stop}$)

Run SP&R with ($f$, $V_{min}$), ($f$, $V_{max}$), ($f$, $\frac{V_{min}+V_{max}}{2}$);
Extract circuit information (= $C_{load}$, $C_{gate}$, $Pct_{LVT}$, $P_{sw}$, $P_{int}$ and $P_{leak}$);
Build the power model based on extracted information;
$i \leftarrow 1$; $V_0 \leftarrow V_{min}$;
**while** $\Delta V \geq V_{stop}$ **do**
  $V_i \leftarrow$ select the optimal $V$ based on the power model;
  Run SP&R with ($f$, $V_i$);
  Extract circuit information;
  Update the power model using LSQR based on extracted information;
  $\Delta V \leftarrow V_i - V_{i-1}$; $i \leftarrow i + 1$;
**end while**
**return** $V_{i-1}$

**Procedure** MAX_FREQ ($V$, $P_{max}$, $f_{min}$, $f_{max}$, $f_{stop}$)

Run SP&R with ($f_{min}$, $V$), ($f_{max}$, $V$), ($\frac{f_{min}+f_{max}}{2}$, $V$);
Extract circuit information;
Build the power model based on extracted information;
$i \leftarrow 1$; $f_0 \leftarrow f_{min}$;
**while** $\Delta f \geq f_{stop}$ **do**
  $f_i \leftarrow$ select $f$ based on the power model such that $P = P_{max}$;
  Run SP&R with ($f_i$, $V$);
  Extract circuit information;
  Update the power model using LSQR based on extracted information;
  $\Delta f \leftarrow f_i - f_{i-1}$; $i \leftarrow i + 1$;
**end while**
**return** $f_{i-1}$

### 2.1.4 Methodology

**Design space reduction.** According to *Lemma 2*, we search only the design space in which the equivalent dominance property holds to reduce overdesign.

**Duty-cycle awareness.** Our power model estimates $P_{avg}$ based on $r$ and our optimizations aim at reducing $P_{avg}$ or are constrained by an upper bound on $P_{avg}$.

**Design cone approximation.** We estimate a design cone using LVT- and HVT-only inverter chains, as in [30].

### The FIND_OD Problem

We extend the MAX_FREQ flow to solve the FIND_OD problem (Algorithm 2). One key observation which reduces the number of MCMM (*Multi-Corner Multi-Mode*) implementations during the adaptive search is that a circuit implemented at a particular pair of nominal mode and overdrive mode can also run at other overdrive modes along its frequency vs. voltage tradeoff curve (Figure 2.5(a)). This implies that circuits implemented with a nominal mode and any overdrive mode along one frequency vs. voltage tradeoff curve will have similar circuit properties. Thus, we can extract circuit properties for solutions in the design cone by generating a few trial circuits with different frequency vs. voltage tradeoffs.

**Algorithm 2** Method for solving the FIND_OD problem.

1: Find the design cone of the nominal mode ($f_{nom}, V_{nom}$);
2: Find the intersections of the maximum supply voltage $V_{max}$ and boundaries of the design cone. Define the minimum and maximum frequencies of these intersections as $f_a$ and $f_b$, respectively;
3: Run MCMM SP&R with the given nominal mode and overdrive modes defined by $\{f_a, f_b, \frac{f_a+f_b}{2}\}$ and $V_{max}$;
4: Extract circuit information. Build or update the power model;
5: Estimate $P_{avg}$, based on the given $r$, corresponding to feasible overdrive modes within the design cone. Find the maximum $f_{OD}$ along with the corresponding $V_{OD}$ satisfying power constraints;
6: Run MCMM SP&R with the overdrive mode obtained in Step 5. Repeat Steps 4-6 until $\Delta f_{OD}$ is less than a stopping criterion $f_{stop}$.

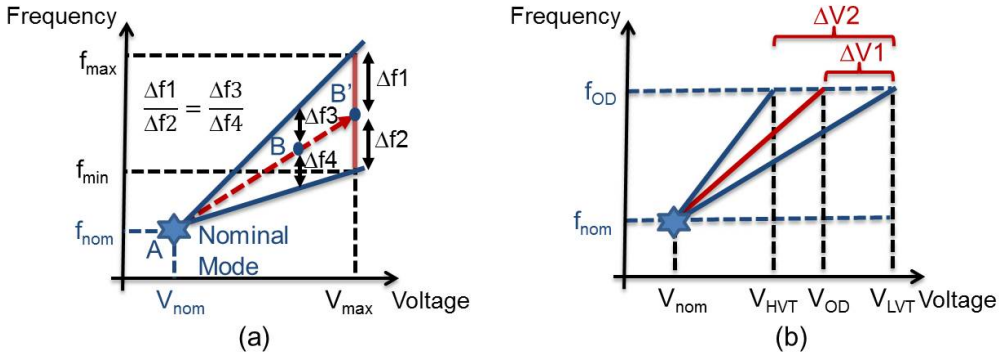



**Figure 2.5**: (a) Projection of mode ***B*** to mode ***B'*** for circuit property modeling. (b) $\lambda(V_{nom})$ calculation, where $\lambda(V_{nom}) = \Delta V1/\Delta V2$. $V_{HVT}$ and $V_{LVT}$ are defined by the intersections of $f_{OD}$ and the design cone.

### The FIND_NOM Problem

The FIND_NOM problem is similar to the FIND_OD problem. We solve the FIND_NOM problem using the same methodology as for the FIND_OD problem.

**The FIND_VOLT Problem**

Finding the optimal $V_{nom}$ and $V_{OD}$ pair using exhaustive search incurs large runtime because there are $O(n^2)$ feasible solutions ($n$ is the number of feasible signoff voltages). To reduce the runtime complexity, we propose an *approximate* optimization method: for each $V_{nom}$, we consider only one $V_{OD}$, in which we determine the $V_{OD}$ based on a parameter $\lambda(V_{nom})$ as illustrated in Figure 2.5(b).[3] $\lambda(V_{nom})$ indicates the *ratio of HVT cells to total cells* in the critical paths. When the signoff voltage increases, paths become faster and more HVT cells are used to reduce power. As a result, for a fixed $f_{nom}$, $\lambda(V_{nom})$ increases with $V_{nom}$. We heuristically approximate $\lambda(V_{nom})$ as a linear function of $V_{nom}$ in our method.

$$\lambda(V_{nom}) = \frac{\lambda(V_{max}) - \lambda(V_{min})}{V_{max} - V_{min}} \times V_{nom} + \lambda(V_{min}) \tag{2.8}$$

in which $V_{max}$ and $V_{min}$ are respectively the maximum $V_{nom}$ at the given technology node and the minimum supply voltage at $f_{nom}$, which we assume can be determined by designers. We calculate $\lambda(V_{max})$ and $\lambda(V_{min})$ based on the desired $V_{OD}$ that minimizes $P_{avg}$ when $V_{nom}$ equals to $V_{max}$ and $V_{min}$, respectively. Algorithm 3 shows the steps to solve the FIND_VOLT problem.

**The FIND_FREQ Problem**

For each $f_{nom}$, we consider only one $f_{OD}$. Further, we approximate $\lambda(f_{nom})$ as a linear function of $f_{nom}$. Since the methodology for the FIND_FREQ problem is similar to that for the FIND_VOLT problem (in that the frequency and voltage axes are swapped), we skip the detailed descriptions.

**Algorithm 3** Method for solving the FIND_VOLT problem.

1: Define two nominal modes ($f_{nom}$, $V_{min}$) and ($f_{nom}$, $V_{max}$); For each nominal mode, determine the $V_{OD}$ with the minimum $P_{avg}$ by using the MIN_POWER flow;
2: Calculate $\lambda(V_{min})$ and $\lambda(V_{max})$ with the resultant $V_{OD}$;
3: Run MCMM SP&R at $\{V_{min}, V_{max}, \frac{V_{min}+V_{max}}{2}\}$ (with $f_{nom}$) and the corresponding $V_{OD}$ (with $f_{OD}$) determined by $\lambda$ values;
4: Extract circuit information. Build or update the power model;
5: Find $V_{nom}$ and the corresponding $V_{OD}$ that achieve minimum $P_{avg}$ based on the power model;
6: Run MCMM SP&R with the $V_{nom}$ and $V_{OD}$ obtained in Step 6. Repeat Steps 4-6 until $\Delta P_{avg}$ is less than a stopping criterion $P_{stop}$.

[3] Experimental results in Section 2.1.5 show that our approximate optimization can achieve results similar to those of the exhaustive search.

### 2.1.5 Experimental Results

Our experiments use two RTL designs (*AES* and *JPEG*) from *OpenCores* website [230] and four blocks (*FPU*, *MUL*, *EXU* and *SPU*) from OpenSPARC T1 [231]. Designs are implemented with both foundry 65$nm$ triple-$V_{th}$ libraries and 28$nm$ dual-$V_{th}$ libraries. We synthesize designs at both nominal and overdrive modes using *Synopsys Design Compiler vG-2012.06* [237], and pick the mode with less power after routing.[4] We run MCMM P&R using *Cadence Encounter Digital Implementation System v10.1* [217]. To eliminate tool noise, we execute each P&R run three times, perturbing the timing constraints by a small amount (i.e., 0.5% of the clock period) [95]. We use *SensOpt* [235] for post-routing leakage optimization, and *Synopsys PrimeTime vC-2009.06-SP2* [240] for timing and power analyses. We run timing analysis at SS corner and power analysis at FF corner. Our basic experimental configuration assumes $r$ = 50%. All implemented designs have worst negative slacks (WNS) $\geq$ -10$ps$.[5] During adaptive search, we derive and refine our power model using *MATLAB vR2009B* [228].

**FIND_OD problem.** Table 2.1 shows the experimental setup, where $P_{avg\_max}$, $P_{OD\_max}$ and $V_{max}$ respectively constrain $P_{avg}$, $P_{OD}$ and $V_{OD}$. We assume the same overdrive mode for all four blocks from OpenSPARC T1 and combine them into a single instance which we denote as *OST1*. For each instance, we implement four methods to optimize the overdrive mode. The *Signoff&Scale* method applies the traditional "signoff and scale", where we first sign off circuits with the given nominal mode and then perform timing and power analyses with libraries characterized at higher voltages to search for the maximum $f_{OD}$ under power constraints. Note that we perform an additional MCMM P&R run to optimize power at both modes after the overdrive mode is selected. The *Proposed method* uses the proposed adaptive search. The *Exhaustive search* explores the entire feasible design space for given design parameters. We also compare to the *Method in [30]*.

Results in Table 2.2 show that the *Proposed method* achieves up to $>$ 8% and 6% overdrive performance improvements compared to the *Signoff&Scale* and the *Method in [30]*, respectively, while maintaining similar area and power. Further, the *Proposed method* is within 4% of that obtained from the *Exhaustive search*, while using less than 8% of the *Exhaustive search* runtime. We also note that our *Proposed method* is scalable due to its use of adaptive search, which is able to converge to a near-optimal solution after a small number of SP&R runs.

[4] Although this may be unnecessary when modes are equivalently dominant, we use the same implementation for all experiments for fair comparisons.

[5] The small WNS is due to the discrepancy between timing analysis in *Cadence Encounter Digital Implementation System v10.1* [217] and in *Synopsys PrimeTime vC-2009.06-SP2* [240].

**Table 2.1**: Experimental setup for the FIND_OD problem.

| **Case** | **Design** | **Technology** | $f_{nom}$ $(MHz)$ | $V_{nom}$ $(V)$ | $P_{avg\_max}$ $(mW)$ | $P_{OD\_max}$ $(mW)$ | $V_{max}$ $(V)$ |
|---|---|---|---|---|---|---|---|
| 1 | *AES* | $65nm$ | 500 | 0.9 | 40 | 55 | 1.2 |
| 2 | *JPEG* | $65nm$ | 400 | 0.9 | 80 | 100 | 1.2 |
| 3 | *OST1* | $65nm$ | 600 | 0.9 | 210 | 300 | 1.2 |
| 4 | *AES* | $28nm$ | 800 | 0.8 | 25 | 30 | 1.1 |
| 5 | *AES* | $28nm$ | 800 | 0.8 | 35 | 40 | 1.1 |
| 6 | *JPEG* | $28nm$ | 500 | 0.8 | 35 | 50 | 1.1 |

When we optimize each block in *OST1* individually (fine-grained optimization) the *Proposed method* achieves 4-8% $f_{OD}$ improvement compared to the *Signoff&Scale*. For Case 3 in Table 2.2 (coarse-grained optimization), the corresponding $f_{OD}$ improvement is 6.5%. These consistent $f_{OD}$ improvements suggest that the *Proposed method* is scalable.

**FIND_VOLT problem.** Table 2.3 and Table 2.4 respectively show our experimental setup and results. The *Proposed method* achieves less than 6% power overhead, with 6× runtime reduction, compared to *Exhaustive search*. The *Proposed method* also achieves up to 12% reduction of average power compared to the *Method in [30]*.

**FIND_FREQ problem.** Table 2.5 and Table 2.6 respectively show our experimental setup and results. The *Proposed method* achieves less than 3% performance overhead, with around 10× runtime reduction, compared to *Exhaustive search*.

**Duty cycle-awareness validation.** To show that our proposed methodology is duty cycle-aware, we optimize *AES* (in the context of the FIND_OD problem) with different duty cycles ($r_{opt}$) in both $65nm$ and $28nm$ technologies. In $65nm$ technology, we assume the nominal mode as ($500MHz$, $0.9V$) and constraints on $P_{avg}$ and $V_{max}$ as $30mW$ and $1.2V$, respectively. In $28nm$ technology, we assume the nominal mode as ($800MHz$, $0.8V$) and constraints on $P_{avg}$ and $V_{max}$ as $30mW$ and $1.1V$, respectively. We then evaluate the maximum $f_{OD}$ of outcomes with different duty cycles ($r_{eval}$) under the power constraints.

Results in Table 2.7 show that $f_{OD}$ and $V_{OD}$ decrease with a larger $r_{opt}$. That is, given fixed power constraints, optimization with a smaller $r_{opt}$ results in a faster design. Further, maximum $f_{OD}$ is achieved when $r_{eval} = r_{opt}$. These observations confirm the duty cycle-awareness of our *Proposed method*. The results also show the cost of inaccurate prediction for $r$. For example, in $65nm$, if $r$ = 0.1 ($f_{OD}$ = $845MHz$), but the optimization assumes $r$ = 0.9 ($f_{OD}$ = $805MHz$), there is a performance penalty of 5%.

**Table 2.2**: Metrics of circuits implemented for the FIND_OD problem.

| | | *Signoff &Scale* | *Proposed method* | *Exhaustive search* | *Method in [30]* |
|---|---|---|---|---|---|
| *AES* (Case 1) | $f_{OD}\ (MHz)$ | 760 | 822 | 840 | 810 |
| | $V_{OD}\ (V)$ | 1.20 | 1.18 | 1.16 | 1.18 |
| | **Area** $(um^2)$ | 30002 | 30594 | 31405 | 30832 |
| | $P_{avg}\ (mW)$ | 35.1 | 36.2 | 37.3 | 36.0 |
| | **#P&R runs** | 2 | 4 | 66 | 7 |
| *JPEG* (Case 2) | $f_{OD}\ (MHz)$ | 580 | 638 | 660 | 600 |
| | $V_{OD}\ (V)$ | 1.16 | 1.18 | 1.12 | 1.18 |
| | **Area** $(um^2)$ | 114679 | 122394 | 127361 | 117355 |
| | $P_{avg}\ (mW)$ | 67.6 | 70.5 | 69.3 | 69.7 |
| | **#P&R runs** | 2 | 4 | 66 | 7 |
| *OST1* (Case 3) | $f_{OD}\ (MHz)$ | 860 | 916 | 940 | 870 |
| | $V_{OD}\ (V)$ | 1.16 | 1.14 | 1.12 | 1.16 |
| | **Area** $(um^2)$ | 151149 | 154253 | 156363 | 150491 |
| | $P_{avg}\ (mW)$ | 163.2 | 162.0 | 162.0 | 162.4 |
| | **#P&R runs** | 2 | 5 | 66 | 7 |
| *AES* (Case 4) | $f_{OD}\ (MHz)$ | 1220 | 1270 | 1280 | 1260 |
| | $V_{OD}\ (V)$ | 0.98 | 0.96 | 0.98 | 1.02 |
| | **Area** $(um^2)$ | 11591 | 12229 | 12051 | 11474 |
| | $P_{avg}\ (mW)$ | 19.4 | 20.7 | 20.9 | 21.2 |
| | **#P&R runs** | 2 | 4 | 30 | 10 |
| *AES* (Case 5) | $f_{OD}\ (MHz)$ | 1400 | 1470 | 1480 | 1440 |
| | $V_{OD}\ (V)$ | 1.06 | 1.02 | 1.06 | 1.10 |
| | **Area** $(um^2)$ | 11561 | 12527 | 11991 | 11457 |
| | $P_{avg}\ (mW)$ | 24.5 | 25.5 | 26.6 | 26.5 |
| | **#P&R runs** | 2 | 4 | 30 | 8 |
| *JPEG* (Case 6) | $f_{OD}\ (MHz)$ | 800 | 845 | 880 | 820 |
| | $V_{OD}\ (V)$ | 1.00 | 0.98 | 0.98 | 1.02 |
| | **Area** $(um^2)$ | 55125 | 57225 | 54549 | 53207 |
| | $P_{avg}\ (mW)$ | 34.0 | 35.1 | 34.5 | 35.3 |
| | **#P&R runs** | 2 | 4 | 30 | 10 |

### 2.1.6 Conclusion

Based on the properties of *equivalent dominance*, we propose guidelines and efficient methodologies to search for the optimal modes for overdrive signoff. The proposed methodologies can successfully determine the signoff modes that reduce lifetime energy, and are shown to

achieve up to $> 8\%$ and $6\%$ performance improvements compared to the traditional "signoff and scale" and the previous work [30], respectively. The methodologies also result in less than $6\%$ power overhead as compared to the optimal solutions.

**Table 2.3**: Experimental setup for the FIND_VOLT problem.

| **Case** | **Design** | **Technology** | $f_{nom}\ (MHz)$ | $f_{OD}\ (MHz)$ | $P_{OD\_max}\ (mW)$ | $V_{max}\ (V)$ |
|---|---|---|---|---|---|---|
| 7 | *AES* | $65nm$ | 700 | 850 | 50 | 1.2 |
| 8 | *JPEG* | $65nm$ | 600 | 720 | 100 | 1.2 |
| 9 | *AES* | $28nm$ | 1000 | 1300 | 30 | 1.1 |
| 10 | *JPEG* | $28nm$ | 600 | 800 | 40 | 1.1 |

**Table 2.4**: Metrics of circuits implemented for the FIND_VOLT problem.

| | | *Proposed method* | *Exhaustive search* | *Method in [30]* |
|---|---|---|---|---|
| *AES* (Case 7) | $V_{nom}\ (V)$ | 0.92 | 0.92 | 0.90 |
| | $V_{OD}\ (V)$ | 1.02 | 1.02 | 1.04 |
| | **Area** $(um^2)$ | 35349 | 35349 | 34599 |
| | $P_{avg}\ (mW)$ | 41.8 | 41.8 | 44.1 |
| | **#P&R runs** | 7 | 44 | 10 |
| *JPEG* (Case 8) | $V_{nom}\ (V)$ | 0.94 | 0.90 | 0.86 |
| | $V_{OD}\ (V)$ | 1.04 | 0.94 | 0.96 |
| | **Area** $(um^2)$ | 136747 | 148360 | 145906 |
| | $P_{avg}\ (mW)$ | 85.4 | 80.9 | 91.9 |
| | **#P&R runs** | 6 | 46 | 9 |
| *AES* (Case 9) | $V_{nom}\ (V)$ | 0.88 | 0.92 | 0.96 |
| | $V_{OD}\ (V)$ | 0.98 | 1.06 | 1.08 |
| | **Area** $(um^2)$ | 12084 | 12439 | 10150 |
| | $P_{avg}\ (mW)$ | 24.5 | 23.9 | 24.7 |
| | **#P&R runs** | 7 | 42 | 11 |
| *JPEG* (Case 10) | $V_{nom}\ (V)$ | 0.82 | 0.82 | 0.82 |
| | $V_{OD}\ (V)$ | 0.92 | 0.92 | 0.92 |
| | **Area** $(um^2)$ | 55276 | 55276 | 55276 |
| | $P_{avg}\ (mW)$ | 32.4 | 32.4 | 32.4 |
| | **#P&R runs** | 7 | 42 | 15 |

**Table 2.5**: Experimental setup for the FIND_FREQ problem.

| **Case** | **Design** | **Technology** | $V_{nom}$ $(V)$ | $V_{OD}$ $(V)$ | $P_{avg\_max}$ $(mW)$ | $P_{OD\_max}$ $(mW)$ |
|---|---|---|---|---|---|---|
| 11 | *AES* | $65nm$ | 0.9 | 1.1 | 40 | 55 |
| 12 | *JPEG* | $65nm$ | 0.9 | 1.2 | 80 | 120 |

**Table 2.6**: Metrics of circuits implemented for the FIND_FREQ problem.

| | | *Proposed method* | *Exhaustive search* |
|---|---|---|---|
| *AES* (Case 11) | $f_{nom}$ $(MHz)$ | 618 | 610 |
| | $f_{OD}$ $(MHz)$ | 810 | 860 |
| | $f_{avg}$ | 714 | 735 |
| | **Area** $(um^2)$ | 31526 | 32740 |
| | $P_{avg}$ $(mW)$ | 40.3 | 39.6 |
| | **#P&R runs** | 6 | 70 |
| *JPEG* (Case 12) | $f_{nom}$ $(MHz)$ | 431 | 440 |
| | $f_{OD}$ $(MHz)$ | 623 | 630 |
| | $f_{avg}$ | 527 | 535 |
| | **Area** $(um^2)$ | 119777 | 120670 |
| | $P_{avg}$ $(mW)$ | 81.2 | 82.6 |
| | **#P&R runs** | 6 | 52 |

**Table 2.7**: Metrics of circuits implemented with different $r_{opt}$.

| **Technology** | $r_{opt}$ | $f_{OD}$ $(MHz)$ | $V_{OD}$ $(V)$ | $f_{max}$ $(MHz)$ with $r_{eval}$ = | | | | |
|---|---|---|---|---|---|---|---|---|
| | | | | 0.1 | 0.3 | 0.5 | 0.7 | 0.9 |
| $65nm$ | 0.1 | 844 | 1.20 | **845** | 830 | 725 | 670 | 640 |
| | 0.3 | 832 | 1.19 | 840 | **830** | 725 | 670 | 640 |
| | 0.5 | 726 | 1.10 | 815 | 815 | **730** | 670 | 635 |
| | 0.7 | 670 | 1.05 | 805 | 805 | 720 | **670** | 635 |
| | 0.9 | 638 | 1.02 | 805 | 805 | 720 | 670 | **640** |
| $28nm$ | 0.1 | 1720 | 1.10 | **1660** | 1410 | 1225 | 1110 | 1035 |
| | 0.3 | 1440 | 1.04 | 1615 | **1465** | 1245 | 1130 | 1070 |
| | 0.5 | 1220 | 0.96 | 1600 | 1445 | 1230 | 1125 | 1070 |
| | 0.7 | 1110 | 0.92 | 1600 | 1450 | 1235 | **1130** | 1075 |
| | 0.9 | 1050 | 0.90 | 1600 | 1445 | 1225 | 1125 | **1075** |

## 2.2 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 work, 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 [178][239], 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.

### 2.2.1 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. [27] 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. [37] 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. [146] 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. [147] 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. [167] 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 [176] use mesh structures for the top-level tree which consumes less wire resource and power as compared to [167]. 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. [163][164] propose a nontree construction method to insert crosslinks[6] in a clock tree by estimating subtree delays using the 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 [151] 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.

### 2.2.2 Problem Formulation

The notations we use in this section are given in Table 2.8.

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

$$v_{i,i'}^{c_k,c_{k'}} = |\alpha_k \cdot skew_{i,i'}^{c_k} - \alpha_{k'} \cdot skew_{i,i'}^{c_{k'}}| \tag{2.9}$$

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). $\alpha_k$ is the normalization factor at corner $c_k$ with respect to the nominal corner. Note that $\alpha_k$ is an input parameter and can be determined by technology information (e.g., ratio between buffer delays at

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

**Table 2.8**: Description of notations used in our work.

| Term | Meaning |
|---|---|
| $c_k$ | operating corner, ($0 \leq k \leq K$; $c_0$ is the nominal corner) |
| $\alpha_k$ | 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 $\alpha_k$ values for each sink pair. In our work, we define $\alpha_k$ 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

$$V_{i,i'} = \max_{\forall (c_k, c_{k'})} v_{i,i'}^{c_k,c_{k'}} \tag{2.10}$$

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.

$$\text{Minimize} \sum_{\forall (f_i, f_{i'})} V_{i,i'} \tag{2.11}$$

### 2.2.3 Optimization Framework

Figure 2.6 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.

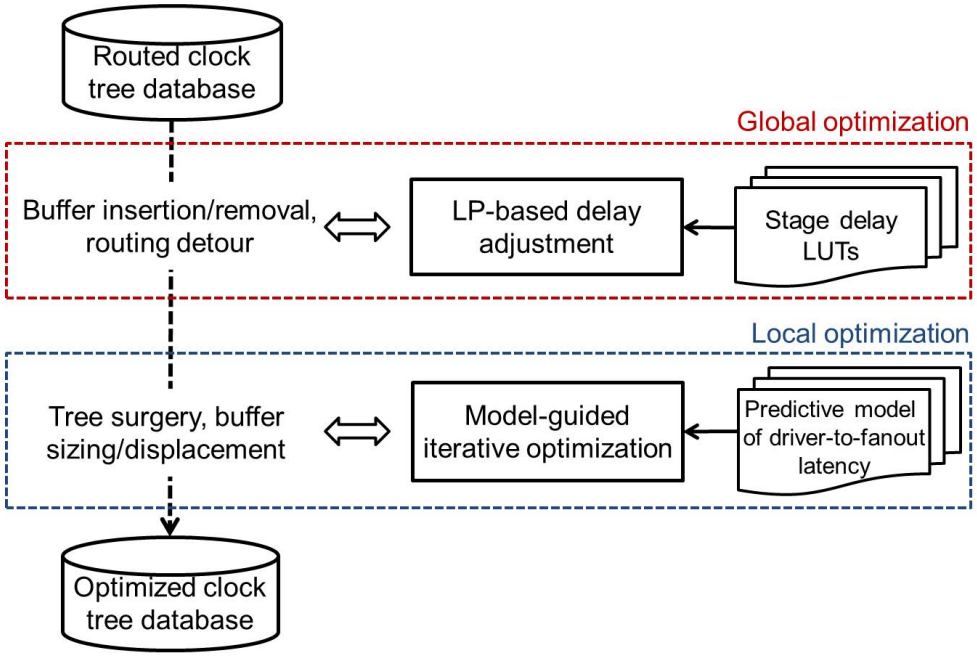



**Figure 2.6**: Overview of our optimization framework.

**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:

$$\text{Minimize} \sum_{1 \le j \le M,\ 0 \le k \le K} |\Delta_j^{c_k}| \tag{2.12}$$

where $\Delta_j^{c_k}$ is the latency change on arc $s_j$ at corner $c_k$.[7] The upper bound $U$ on the sum of skew variations is specified as

$$\sum_{(f_i,f_{i'})} V_{i,i'} \leq U \tag{2.13}$$

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.

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

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 $c_k$.

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

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

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

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

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

$$\sum_{s_j \in P_i} (D_j^{c_k} + \Delta_j^{c_k}) \leq D_{max} \tag{2.14}$$

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

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

$$D^{c_k}_{min\ j} \leq D^{c_k}_j + \Delta^{c_k}_j \leq \beta \cdot D^{c_k}_j \tag{2.15}$$

To increase the likelihood that the LP solution is practically implementable, we characterize lookup tables at each corner for stage delays of inverter pairs[8] 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 2.8). 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 2.7 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 ($W^{c_k,c_{k'}}_{max}$) and lower ($W^{c_k,c_{k'}}_{min}$) 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 (2.14)-(2.15). Thus, we use delay per unit distance of an arc in the original clock tree to estimate upper and lower bounds of delay ratios ($W^{c_k,c_{k'}}_{min,max}$), and apply these bounds in Constraint (2.16) to avoid LP solutions that are not practically implementable by ECOs.

$$W^{c_k,c_{k'}}_{min} \leq \frac{D^{c_k}_j + \Delta^{c_k}_j}{D^{c_{k'}}_j + \Delta^{c_{k'}}_j} \leq W^{c_k,c_{k'}}_{max} \tag{2.16}$$

**Complexity Analysis.** The LP formulation (Equations (4)-(11)) has $O(M \cdot K)$ variables to indicate delay change on each arc at each corner ($\Delta^{c_k}_j$); there are also $O(N^2)$ (i.e., 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 (2.14)); $(4 \cdot K)$ constraints to prevent local skew and skew variation degradations (Constraints (2.14)-(2.14)); $N$

[8] In this work, 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.

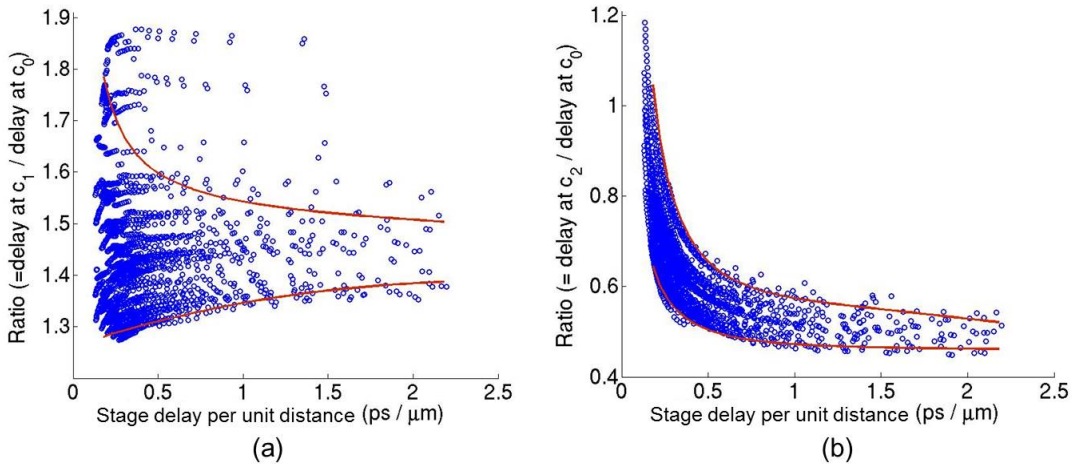



**Figure 2.7**: Delay ratios between ($c_1$, $c_0$) and ($c_2$, $c_0$), respectively. $c_0$ = (SS, 0.9$V$, -25°$C$, Cmax), $c_1$ = (SS, 0.75$V$, -25°$C$, Cmax) and $c_2$ = (FF, 1.1$V$, 125°$C$, Cmin).

constraints to specify the maximum latency (Constraint (2.14)); $(2 \cdot M)$ constraints to bound arc delay changes (Constraint (2.15)); and $C(K, 2)$ constraints to enhance ECO feasibility (Constraint (2.16)).

**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 work, 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 10$\mu m$ to 200$\mu m$ with a step size of 5$\mu m$ 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 2.8).

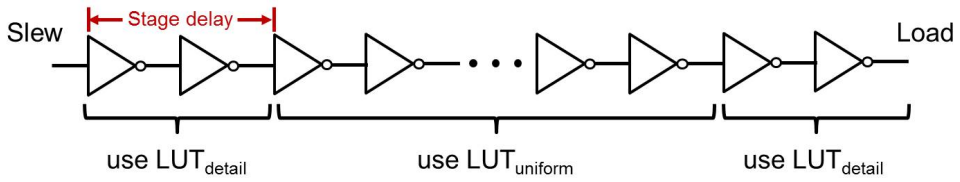



**Figure 2.8**: $LUT_{detail}$ is characterized with various input slews and fanout loads capacitance; $LUT_{uniform}$ contains average stage delay with particular gate size and routed wirelengths between consecutive inverters.

Algorithm 4 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, 21).

**Algorithm 4** LP-guided ECO flow.

1: **for all** $s_j$ to be optimized **do**
2: Remove current inverter pairs on $s_j$
3: $err_{min} \leftarrow +\infty$; $sol \leftarrow \emptyset$
4: **for** $p$ := 1 to $N_{size}$, $q$ := 1 to $N_{WL}$ **do**
5: $u_{est} \leftarrow$ round$(D_{LP}^{c_k}/d(LUT_{uniform})_{p,q}^{c_k})$
6: **for** $u := Max(u_{est}-2,0)$ to $u_{est}+2$ **do**
7: $err \leftarrow 0$
8: **for** $k$ := 0 to $K$ **do**
9: $err \leftarrow err + |D_{est}^{c_k} - D_{LP}^{c_k}|$
10: **end for**
11: **for all** corner pair $(c_k, c_{k'})$ **do**
12: $err \leftarrow err + |(D_{est}^{c_k} - D_{est}^{c_{k'}}) - (D_{LP}^{c_k} - D_{LP}^{c_{k'}})|$
13: **end for**
14: **if** $err < err_{min}$ **then**
15: $err_{min} \leftarrow err$; $sol \leftarrow (p, q, u)$
16: **end if**
17: **end for**
18: **end for**
19: Perform ECO inverter pair insertion based on $sol$
20: **end for**
21: Legalize all inserted inverters and perform ECO routing

**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 Figure 2.9(b)-(d) – (I) buffer sizing and/or buffer displacement, (II) displacement of a buffer and gate sizing on one of its child buffers, and (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.[9] 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 the impact of local moves is necessary. Previous work [84] 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 [42] tree and single-trunk Steiner tree. We approximate wire delays correspondingly using Elmore delay and D2M [7] 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 [118] and update gate delays one and two stages downstream based on Liberty tables.[10] However, as observed in [84], 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.

We therefore construct machine learning-based models to minimize such discrepancy. We use Artificial Neural Networks (ANN) [85], Support Vector Machines (SVM) with a Radial Basis Function (RBF) kernel [85], and Hybrid Surrogate Modeling (HSM) [112].[11] In addition to the estimated delays based on $\{$FLUTE tree, single-trunk Steiner tree$\}$ $\times$ $\{$Elmore delay,

[9] 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 2.5$GHz$ Intel Xeon server, is around 70 minutes (i.e., 30 minutes for ECO and parasitic extraction, and 40 minutes for timing analysis).

[10] 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.

[11] Further details of the applied machine-learning techniques that we use may be found in [85] and [112].

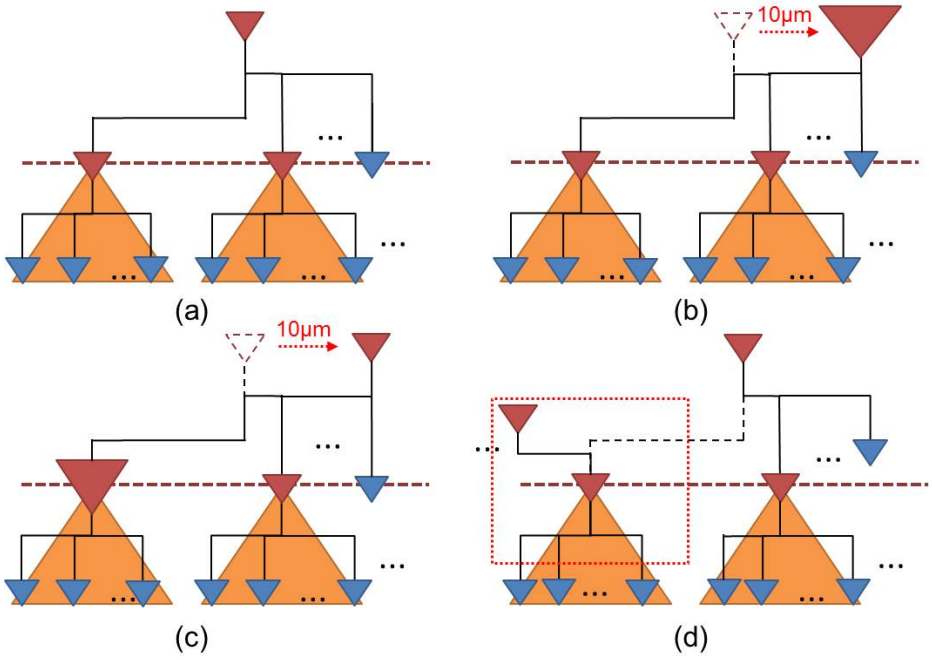



**Figure 2.9**: Local optimization moves used in our flow. (a) Initial subtree; (b) sizing and/or displacement; (c) displacement and sizing of child node; and (d) tree surgery, i.e., driver reassignment.

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 2.10(a) shows the predicted vs. actual latencies that we compute from the predicted delta latencies by our model at corner $c_3$ in Table 2.10. Figure 2.10(b) shows the corresponding histogram 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

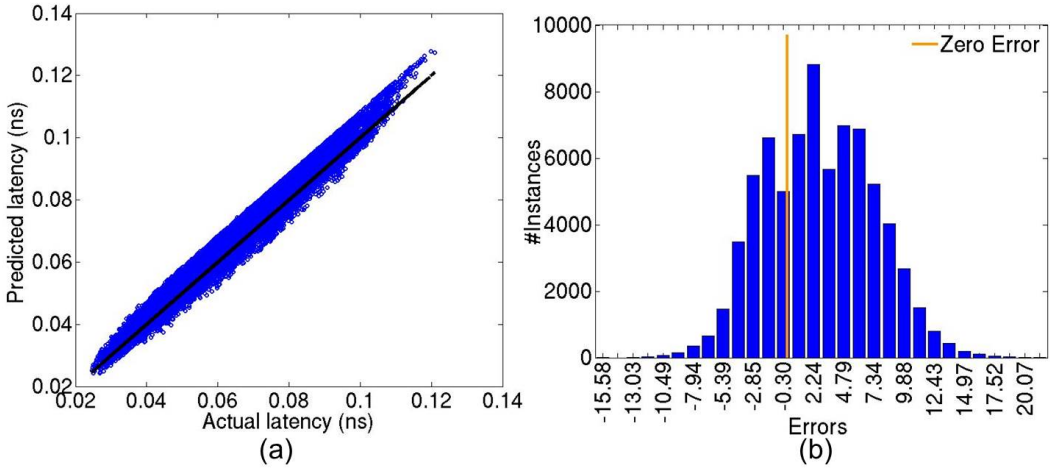



**Figure 2.10**: Examples of (a) predicted vs. actual latencies, and (b) percentage error histograms from our model for $c_3$ corner in Table 2.10.

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 6 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.

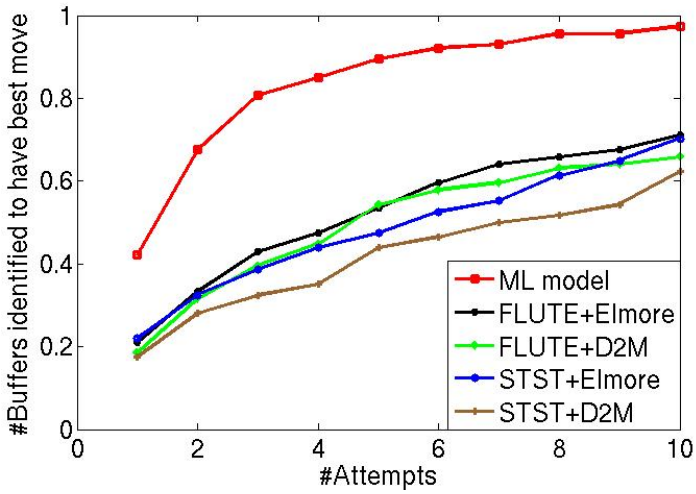



**Figure 2.11**: Accuracy comparison between our learning-based model and analytical models. An attempt is an ECO. There are 114 buffers, and each buffer has 45 candidate moves. In one attempt, the learning-based model (resp. analytical models) can identify best moves for 40% (resp. up to 20%) of the buffers.

**Iterative Optimization Flow.** Based on our model, we perform iterative local optimization flow illustrated in Algorithm 5. We first enumerate all candidate local moves and generate the input data to our model (Line 1). The moves we consider in this work are shown in Table 2.9. 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 2.5$GHz$ 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., $R$ = 5 in this work) 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 5** 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 2.9**: 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$\mu m$ |

### 2.2.4 Experimental Results

Our experiments are implemented in foundry 28$nm$ LP technology. We construct the original clock tree and perform ECO optimizations using *Synopsys IC Compiler vI-2013.12-*

*SP1* [239]. We use *Synopsys PrimeTime vH-2013.06-SP3* [240] and *Synopsys PrimeTime-PX vH-2013.06-SP3 PT-PX* [240] for timing and power analyses, respectively. We construct the machine learning-based model using *MATLAB vR2013a* [228]. The optimization flow is implemented using C++ and Tcl scripts.

**Testcase Description**

We have developed 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 2.10. We use the testcase generation methodology described in [28], and the top-level structures of the testcases T1 and T2 in [28]. 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.[12] Figure 2.12(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 2.11. Corners $c_0$ and $c_1$ are setup-critical, and $c_3$ is hold-critical. Table 2.11 summarizes various post-synthesis metrics of these testcases.

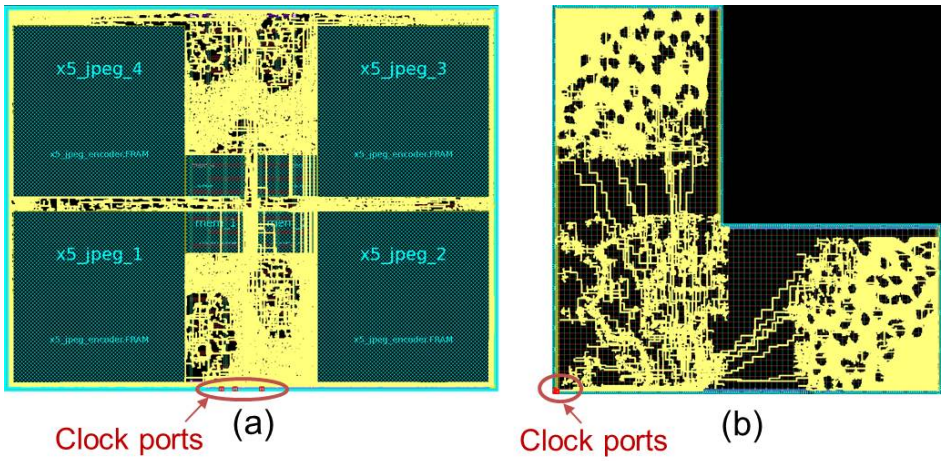



**Figure 2.12**: Floorplans of (a) *CLS1v1*, and (b) *CLS2v1*. In yellow are routed clock nets.

[12]We understand from our industry collaborators that best-practices flows for high-speed and memory controller blocks start with 50%–60% utilization before placement [169].

**Table 2.10**: Description of corners.

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

**Table 2.11**: Summary of testcases.

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

We also study a testcase *CLS2v1* of class memory controller, which is new as compared to [28]. Table 2.11 summarizes the post-synthesis metrics of this testcase, and Figure 2.12(b) shows its floorplan. We use the methodology described in [28] 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$1$mm$). 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 2.11, 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 [169]. We set the skew target as 0$ps$ in the CTS tools, as our studies (with skew targets ranging from 0$ps$ to 250$ps$, in steps of 50$ps$) indicate that a target skew of 0$ps$ 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 2.12 shows the experimental results,[13] 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,[14] 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 2.12**: Experimental results.

| **Testcase** | **Flow** | **Variation [norm]** ($ns$) | **Skew** ($ps$) | | | **#Cells** | **Power** ($mW$) | **Area** ($\mu m^2$) |
|---|---|---|---|---|---|---|---|---|
| | | | $c_0$ | $c_1$ | $c_{2,3}$ | | | |
| *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 |

Figure 2.13 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

[13] Our optimization does not create any maximum transition or maximum capacitance violations.

[14] The number of optimized sink pairs for *CLS1v1*, *CLS1v2* and *CLS2v2* are respectively 15012, 14671 and 15142.

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.

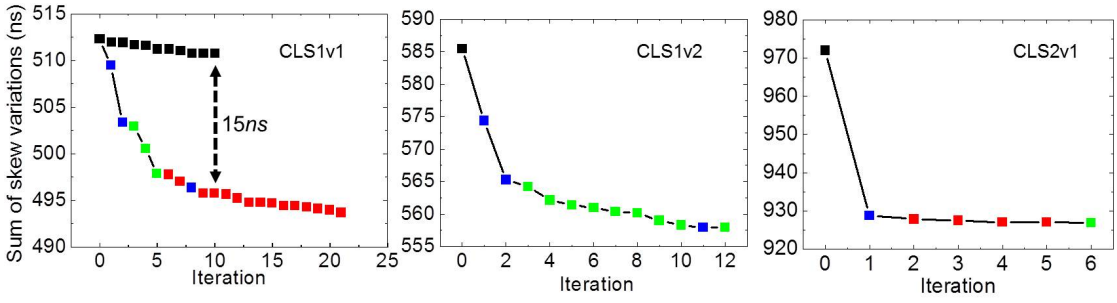



**Figure 2.13**: Sum of skew variations reduces during the local iterative optimization. In blue are type-I moves, in red are type-II moves, and in green are type-III moves.

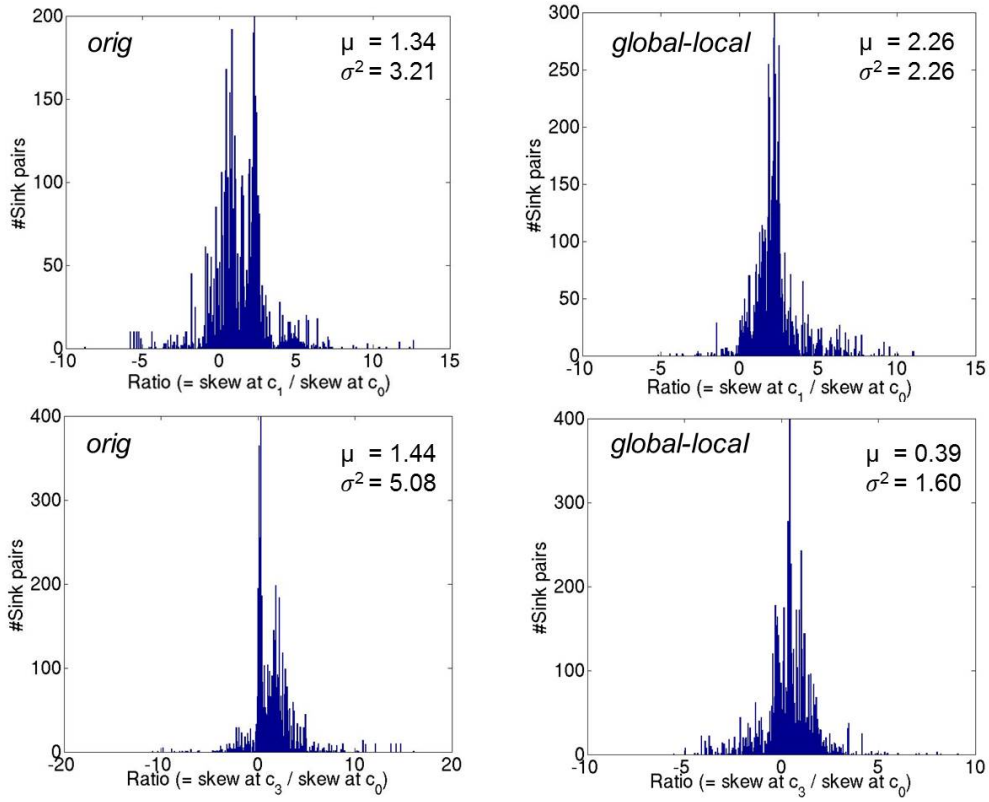



**Figure 2.14**: Distribution of skew ratios between ($c_1$, $c_0$) and ($c_3$, $c_0$) of (i) original clock tree, and (ii) optimized clock tree for *CLS1v1*.

Figure 2.14 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.

### 2.2.5 Conclusion

In this work, 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.

## 2.3 Comprehensive Optimization of Scan Chain Timing During Late-Stage IC Implementation

Scan chain timing is an important consideration in modern scan chain design. Setup timing of scan shift timing paths or scan paths directly affects test time and cost, and any improvement of scan shift timing or scan timing will not only reduce test time and cost, but potentially improve robustness of test as well.[15]

In modern designs, the volume of test pattern is very likely to increase significantly due to (i) the increased importance of delay testing (especially in FinFET technology), (ii) increased test coverage requirements, and (iii) increased design gate counts, which in turn increases test time. To compensate the large volume of test pattern and to reduce the test time, speedup of scan shift is needed. However, fast scan shift will result in worse dynamic voltage drop (DVD) due to higher switching activity and smaller scan timing slacks, which make scan timing more vulnerable to DVD.

Furthermore, due to the small number of logic instances along scan timing paths (i.e., between consecutive scan flip-flops in a given scan chain), scan timing paths are vulnerable to hold violations. Mitigation of hold timing violations along scan chains entails hold buffer insertions which induce power and area overheads.

In this work, we address two timing issues related to scan chain.

- First, we perform scan ordering that exploits knowledge of clock skew and scan cell locations, so as to reduce hold violations along the scan chain and enable the removal of hold buffers. Figure 2.15 shows a simple example where reordering scan cells leads to positive skews between consecutive scan cells in a scan chain, thus removing hold violations.

- Second, scan test at a high frequency (especially during scan shift) is highly likely to incur large dynamic voltage drop (DVD), which in turn degrades scan timing and causes "false failure" in silicon. To address this, we perform DVD-aware gating insertion to reduce dynamic voltage drop during scan shift and maximize scan timing slacks.

Although gating insertion and scan ordering optimizations have been proposed by many previous workers [21][60][65][73][86][89][136][166], these optimizations are performed during early-stage IC implementation (e.g., during synthesis or placement). However, due to subsequent physical implementation steps (notably, clock tree synthesis, signal routing, and buffering and

[15] Depending on the individual SoC design, structural test time can be up to 50% or even a larger proportion of the total test time. Scan time typically dominates structural test time.


**Figure 2.15**: Illustration of skew-aware scan ordering that removes hold violation. $L$ is clock latency.

gate sizing), design information such as clock skew, cell power and timing slacks can be very different at the late design stage (e.g., after routing). Figures 2.16 and 2.17 respectively show that hold-critical scan timing paths and DVD hotspots vary markedly between post-placement and post-routing states of the design. As a result, an early-stage optimization might be misleading and result in poor solution quality.

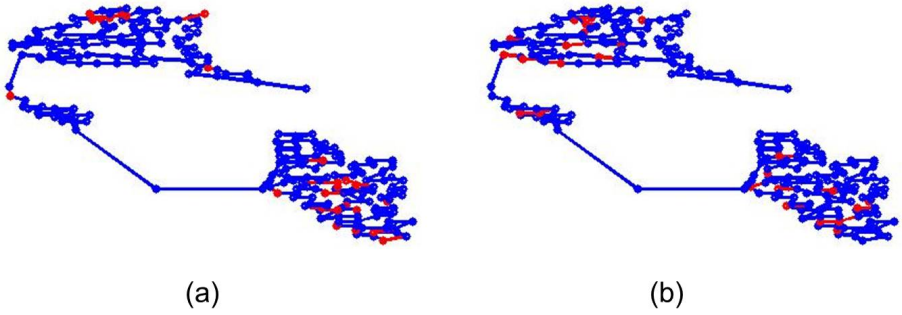



**Figure 2.16**: Hold-critical scan timing paths vary between (a) post-placement stage and (b) post-routing stage. In red are the top 10% of the hold-critical paths. In blue are the non-critical paths. Design: *LEON3MP*. Technology: 28LP.

On the other hand, previous post-routing stage scan chain optimizations mainly focus on test pattern optimization. For example, these optimizations use an ATPG engine to generate low-power test patterns so as to reduce DVD and improve scan timing. However, flip-flop toggling ratios may not necessarily be an accurate indicator of circuits' switching activity or devices' power consumption, and hence may not be able to effectively mitigate the real DVD hotspot issues. Furthermore, generating power-aware test patterns suffers significantly from test pattern inflation and test coverage degradation. For instance, based on our industrial colleague's design experience with modern CPU designs containing multi-millions of flip-flops, imposing the limit of 20% and 30% peak flip-flop switching activities in scan capture and scan shift, respectively, ends up with a 2 or 3 times higher test pattern count along with degradation in test coverage.

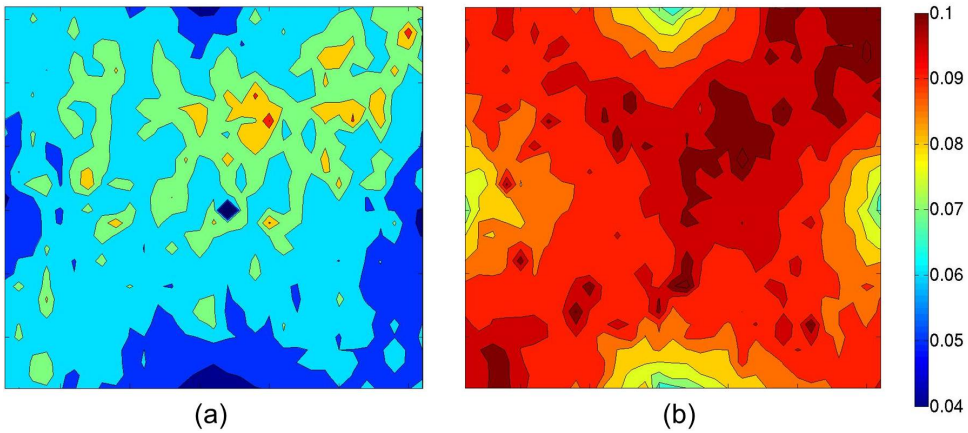



**Figure 2.17**: Dynamic voltage drop (DVD) varies between (a) post-placement stage and (b) post-routing stage. Design: *LEON3MP*. Technology: 28LP.

In this work, we are the first to propose late-stage scan timing optimizations for hold buffer removal and DVD-aware gating insertion. Of course, such late-stage ECO-based scan optimization risks large impacts on design quality – in particular, timing degradation and/or power and area overheads from additional gates being inserted in the functional paths as well as incremental routing due to scan reordering. Therefore, our optimizations necessarily comprehend timing impacts on datapaths at function mode, while keeping area and power overheads to negligible levels (e.g., $< 1\%$). We note that test pattern (compression) optimization is not the focus of this work. However, our optimizations can easily be combined with existing test pattern optimizations. Our contributions are as follows.

- We are the first to propose a comprehensive scan timing optimization at late-stage IC implementation.
- We propose a scan reordering optimization that is aware of clock skew and scan cell locations; this optimization removes up to 82% of hold buffers along scan chains in 28LP testcases.
- We propose an ECO-based gating insertion approach to improve DVD-aware scan timing during scan shift; this optimization reduces the DVD-induced slack degradation by up to 58% with negligible area and power overhead.
- We validate our approaches on a realistic implementation flow (including DFT and DVD-aware timing analysis) in a commercial 28LP technology; this implementation flow has been developed under the guidance of our industrial colleagues.

### 2.3.1 Related Work

We now review previous works on scan ordering optimization and gating insertion, as well as other methods for DVD reduction during scan shift.

**Scan ordering.** Scan chain ordering optimizations have typically been formulated as (symmetric or asymmetric) *Traveling Salesperson Problem* (TSP) optimizations, usually with a wirelength minimization (i.e., total tour cost) objective. An early TSP-based heuristic for scan ordering, which largely ignores physical information, is due to Feuer and Koo [65]. Works such as [21][86][89][136][166] consider physical information and routing to minimize the wirelength overhead of scan chains. Gupta et al. [80] reduce wirelength overhead of scan routing by considering the availability of connection points on entire fanout routing trees (not simply at scan-out pins), and use [233] as their TSP solver. The same authors later consider timing in addition to wirelength minimization [81]. Cui et al. [48] propose pre-ordering of clusters to improve power reduction. Seo et al. [171] combine gating, clustering and reordering to minimize scan power.

The above-mentioned academic works, along with commercial tools [158], focus on constructing scan chains from scratch for given new layout. Alternatively, Kahng et al. [109] address an ECO scan chain optimization context. Tudu et al. [186] propose a graph-based optimization to reorder scan cells based on a given test pattern, such that the transitions during scan shift and capture are minimized.

For one of our present foci, namely, scan reordering to minimize hold buffers, we are aware of only the previous work embodied in the 2003 patent of Teene [181]. In Section 2.3.2 below, we propose a scan ordering approach for minimization of hold buffers along scan chains. As compared to the approach in [181] which is aware of skew, our approach also comprehends cell locations, wire delay and setup timing constraints along datapaths.

**Gating approaches.** Due to excessive switching activity during at-speed test, test power is typically a major issue during circuit design and test pattern generation. *Gating insertion* is one of the known methods to reduce scan power. To suppress the activity of fanout combinational cells from scan flip-flops during scan shifting, Gerstendörfer et al. [73] propose to insert gating logic at the output (i.e., Q pin) of scan flip-flops. To minimize the area overhead and the delay impact, Elshoukry et al. [60] propose a critical path-aware partial gating approach, in which the gating points are selected based on the number of fanout cells and their fanouts. The authors of [60] also point out that such gating approaches can incur large peak power when gating logic instances change from the gating mode to the transparent mode. They propose to assign a separate control block for each scan chain and to enable/disable the gating logic instances for different

scan chains one at a time. In addition to gating at the Q pin of a scan flip-flop, Jayaraman et al. [94] propose to also gate the internal nodes inside the fanout cone of a scan flip-flop. The increased flexibility offers a better tradeoff between area overhead of gating logics and test power reduction. Although such gating approaches reduce shift power, the increased capacitance can increase the capture power. Zhao et al. [212] study the tradeoff between test power and capture power reductions.

Although the above gating optimization approaches all reduce test power, they perform optimization at the post-synthesis stage, when detailed (final) layout information is not available. Therefore, these approaches cannot accurately capture the locations of DVD hotspots, and may make suboptimal or guardbanded decisions. Lee et al. [130] address such issues when removing DVD hotspots by using a post-layout test pattern modification. However, the updated test pattern is not efficient and can increase the test time. In another work, since the voltage drop issue typically occurs at the beginning of scan shift (where the scan shift causes a sudden increase in switching activity), Schulze et al. [170] propose to start scan shift with a lower frequency (implicitly assuming that the scan shift uses an independent clock). Once the test power supply has responded to its initial di/dt event, they increase the shift frequency.

We observe that none of the above works consider timing impact of DVD and optimize DVD-aware scan timing slacks. To our knowledge, ours is the first work to propose gating insertion to minimize scan timing degradation due to DVD.

### 2.3.2 Methodology

We propose two basic optimization levers to improve scan timing at the post-routing stage while minimizing power and area overheads. Our optimizations comprehend both hold and setup constraints not only in scan mode, but in function mode(s) as well.

- **Post-routing scan reordering.** Since scan chains contain few combinational gates, they are typically hold-critical. As a result, hold buffers are inserted during the placement and routing stages, incurring power and area overheads. Such buffer insertion must be guardbanded since final routing, extraction and signoff timing analysis are not yet known. On the other hand, we propose scan chain reordering at the *post-routing* stage which can exploit knowledge of clock skews, exact scan cell locations, and post-routing timing slacks. This enables a more surgical improvement of hold timing and removal of hold buffers.

- **Fanout gating for dynamic voltage drop mitigation.** Dynamic voltage drop (DVD) during scan shift can degrade scan setup timing, leading to “false failures” in silicon.

We propose gating insertion to reduce the DVD and improve scan setup timing slack. Again, while gating insertion is a previously-known technique, our contribution lies in demonstrating practicality and benefits when applied at the post-routing stage.

Table 2.13 lists notations used in the following discussion.

**Table 2.13**: Notations used in our work.

| Term | Meaning |
|---|---|
| $q_k$ | $k^{th}$ scan chain, ($1 \leq k \leq K$) |
| $Q$ | set of scan chains in design |
| $v_k$ | number of hold violations along scan chain $q_k$ |
| $p_j$ | $j^{th}$ scan timing path ($1 \leq j \leq M$) |
| $S_j$ | setup timing slack of scan timing path $p_j$ ($1 \leq j \leq M$) |
| $c_i$ | $i^{th}$ cell ($1 \leq i \leq N$) |
| $g_i$ | switching power of cell $c_i$ ($1 \leq i \leq N$) |
| $s_i$ | setup timing slack of cell $c_i$ |
| $h_r$ | DVD hotspot ($1 \leq r \leq R$) |
| $d_{i,i'}$ | Manhattan distance between cells $c_i$ and $c_{i'}$ |

**Scan Reordering for Hold Buffer Removal**

Scan timing paths with negative clock skew values[16] and small distances between launch and capture flip-flops are prone to hold violations (as illustrated in Figure 2.18, (a) negative skew values correlate with increased likelihood of hold violations, and (b) start-end pairs of scan cells separated by small distances have small hold slacks), whereby hold buffers are inserted along the scan chains to meet hold constraints. However, hold buffer insertions cause area and power overheads, which at least indirectly compromise functional and/or test timing.[17] Thus, we propose a scan cell reordering optimization at the post-routing stage (i) to achieve a greater incidence of positive skew values, and (ii) to slightly increase distance between consecutive scan cells so as to increase wire delay and enable buffer removal. We define the scan reordering problem for hold buffer removal as follows.

**Post-Routing Scan Reordering.** **Given** a design (i.e., netlist (.v), placement, CTS, and routing solutions (DEF) with scan chain(s) inserted), timing constraints (SDC), hold buffer list, upper

[16] We use the standard definition of *skew* between two sequentially adjacent flip-flops as launch clock latency minus capture clock latency. Note that a *scan timing path* is a timing path between (launch and capture) flip-flops that are consecutive in a scan chain.

[17] Larger area spreads and slows timing paths, while larger power may be compensated by reduced clock frequencies. Moreover, hold buffer insertion can potentially create new setup timing criticalities.

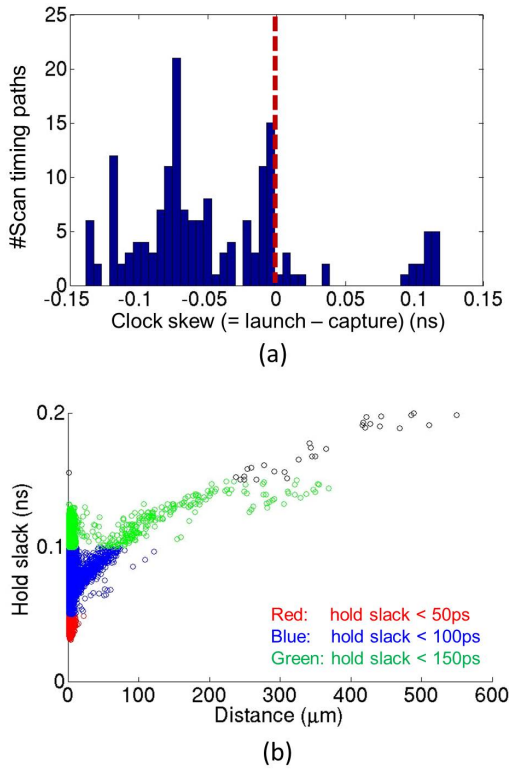



**Figure 2.18**: Causes of hold violations on scan timing paths. (a) Skew distribution of scan timing paths with hold buffers inserted. (b) Distances between consecutive scan cells versus hold timing slacks. Design: *LEON3MP*. Technology: 28LP.

bounds on wirelength penalty, Liberty timing and power models (.lib), and fixed subchain ordering constraints (SCANDEF), **perform scan reordering** to maximize the number of hold buffers removed.

Our reordering optimization uses the *2-opt* local search heuristic for the traveling salesperson problem [129], as explained in Algorithm 6. We sequentially optimize each scan chain in a design. For each scan chain $q_k$ and a given node $c_i$, we perform 2-opt swaps along $c_i$'s downstream nodes and select the ordering solution with the minimum number of hold violations (this heuristically maximizes the number of hold buffer removals) (Lines 4-13).

*Treatment of subchains.* To honor the fixed scan subchain ordering constraints specified by SCANDEF [238] in the input, each subchain with fixed ordering is merged into a single node in the input to our optimization. Further, we observe that hold buffers may be shared between scan timing paths and datapaths: scan timing optimization cannot remove these hold buffers if the removals will cause hold timing violations along datapaths. We therefore do not optimize the corresponding subchains, but instead also merge such subchains into single nodes in the input to our optimization.

*Avoidance of setup timing violations.* We observe that a given hold buffer can shield large wire capacitance and test input pin capacitance at the Q-pin of a timing-critical flip-flop. Removing the hold buffer can thus incur setup timing violations along datapaths. Therefore, timing constraints for both datapaths and scan timing paths must be comprehended when evaluating the swap (i.e., scan reordering) moves. In our optimization, we evaluate timing slack changes due to scan chain ordering based on Liberty timing models (Line 7). We estimate wire delay and capacitance based on Manhattan distance between consecutive scan cells.

*Additional constraints in the local search.* During the local search, we select solutions (i) that have a reduced number of hold violations, (ii) that have no timing degradation along either scan timing paths and datapaths, and (iii) that satisfy the prescribed upper bound on wirelength penalty (Lines 8). More specifically, we set upper bounds on wirelength increase for each pair of consecutive scan cells and each scan chain. Note that to avoid ECO impact (e.g., buffer insertion, placement legalization), we discard solutions that create additional hold violations for pairs of consecutive scan cells which were hold timing-feasible (i.e., have non-negative hold timing slacks) in the original solution. (Without such constraints, additional hold violations can be created for a hold timing-feasible scan cell pair when the total number of hold violations along the scan chain reduces).

To summarize: $feasible()$ in Line 8 indicates a solution that is timing feasible, maintains a bounded wirelength overhead, and incurs no additional hold violations. Finally, after all scan chains are optimized, we perform ECO routing with a commercial place-and-route tool.

**Algorithm 6** Scan reordering.

```
1:  for all q_k ∈ Q do
2:     q' ← q_k
3:     v' ← timingAnalysis(q')
4:     for i := 2 to N^k − 2 do
5:        for i' := i + 1 to N^k − 1 do
6:           q'' ← 2OptSwap(q', i, i')
7:           v'' ← timingAnalysis(q'')
8:           if v'' < v' && feasible(q'') then
9:              q' ← q''; v' ← v''
10:          end if
11:       end for
12:       q_k ← q'
13:    end for
14:    Reorder scan chain based on updated q_k
15: end for
16: ECO route
```

**DVD-Aware Gating Insertion**

As noted above, scan shift typically consumes high power and causes excessive dynamic voltage drop (DVD), which in turn degrades scan setup timing and leads to "false failures" in silicon. To address such DVD-aware timing degradation, we propose the post-routing application of gating insertion [60][73][94] to reduce switching activity of downstream cells and thereby reduce DVD impact on scan timing. We state the DVD-aware gating insertion problem as follows.

**Dynamic Voltage Drop Mitigation by Gating.** **Given** a design (i.e., netlist (.v), placement, CTS, and routing solutions (DEF) with scan chain(s) inserted), timing constraints (SDC), Liberty timing and power models (.lib), switching activities per cell instance, and upper bounds on area/power overheads, **perform gating insertion as ECOs** to maximize the minimum slack of DVD-aware scan timing.

Figure 2.19 shows our proposed three-step optimization flow. (1) We first determine DVD hotspots which have the largest impact on scan timing (i.e., the DVD hotspots containing timing-critical scan cells with the worst slack degradation due to DVD), by solving an integer linear program (ILP). (2) We then allocate gating locations (in the netlist) so that the switching activities of downstream non-scan cells within the selected DVD hotspots are minimized. The reduced switching activities lead to minimized power and thus DVD reduction within the selected hotspots. We determine the gating locations through a netlist traversal that is guided by sensitivity functions (i.e., functions to estimate the sensitivity of dynamic power reduction on downstream cells to a gating insertion). (Note that the potential gating insertion locations are not limited to Q pins of scan cells.) We use both AND and OR gates for gating. Figure 2.20 shows an example with OR gate. In addition, we honor datapath timing constraints in function mode. Further, we perform sizing and $V_{th}$ swapping on the inserted gating cells, such that timing impact due to gating insertion is minimized. (3) Last, we insert gating cells as an ECO step: specifically, we perform a matching-based optimization to determine the whitespace suited for gating insertions. We now give additional details of these steps.

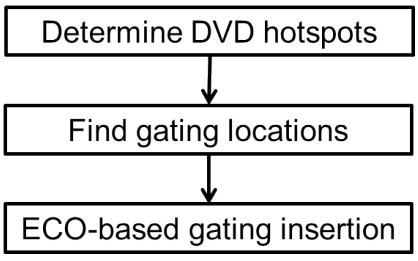



**Figure 2.19**: Optimization flow for gating insertion to optimize DVD-aware timing slacks.

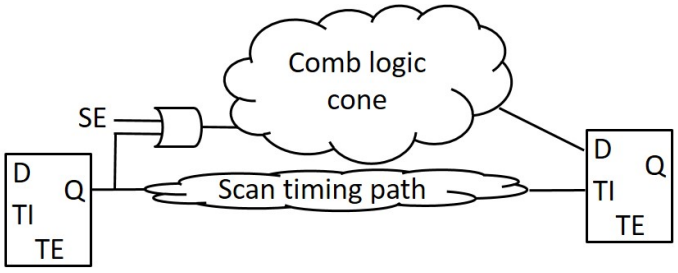



**Figure 2.20**: Illustration of gating insertion with an OR gate.

**(1) Determine DVD hotspots.** To minimize the ECO impact and corresponding area and power overheads of gating insertions while maximizing timing benefits from DVD reduction, it is important to determine the DVD hotspots that have the largest impact on the worst scan timing path. To achieve this, we divide the block area into grids, where each grid is a candidate DVD hotspot. We then set up and solve the following ILP, which selects the DVD hotspots to optimize from among candidates that have maximum impact on scan timing.

$$\begin{aligned}
&\text{Maximize} \quad S_{min} \\
&\text{Subject to} \quad S_j + \sum_{c_i \in p_j} \alpha_i \cdot \Delta_i \geq S_{min},\ \forall p_j && (2.17)\\
&\qquad\qquad \alpha_i \quad \leq L \cdot \beta_r,\ \ \forall c_i \in h_r && (2.18)\\
&\qquad\qquad \sum \beta_r \leq R && (2.19)\\
&\text{Binary} \quad \alpha_i,\ \beta_r && (2.20)
\end{aligned}$$

Here, $S_{min}$ is the minimum slack in the design; $S_j$ is the worst slack of scan timing path $p_j$; $\Delta_i$ is the cell slack improvement due to DVD reduction; $c_i$ is a cell instance; $\alpha_i$ is a binary indicator of whether the DVD on $c_i$ will be optimized (i.e., whether $c_i$ is in the selected DVD hotspot); $\beta_r$ is an indicator of whether hotspot $h_r$ will be optimized; $L$ is a large constant number; and $R$ defines the upper bound on the number of hotspots to be optimized. Given that the number of scan timing paths grows linearly with the number of flip-flops and the number of stages along a scan timing path is typically small, the runtime complexity of our ILP is not high. In our experiments, even for the largest design with 474K instances and 445 scan chains, the runtime is less than 1 second on a $2.5GHz$ Intel Xeon server. Three important considerations are as follows.

*Grid size.* We note that the sizes of grids (i.e., DVD hotspots) can have significant impact on solution quality. A large grid size can result in a large number of gating insertions, with

corresponding area and power overheads. On the other hand, a small grid size may contain only a small number of cells; gating these cells will not effectively reduce DVD.

*Grid aspect ratio.* We observe that the aspect ratios of grids also impact solution quality. Given that cells within the same row share the same power and ground rails, it is more effective to define row-based hotspots (i.e., hotspots with single-row height). Our experimental results in a commercial 28LP technology confirm this hypothesis: the DVD reduction of optimization with single row-height hotspots is 1.7× that of double row-height hotspots with the same area. Thus, in the experiments reported below, we empirically define the grid size as $80\mu m \times 1.2\mu m$.

*Need for iteration.* Last, we note that there is a "chicken-and-egg" loop between the assumed DVD reduction to estimate $\Delta_i$, versus the optimized DVD values. To address this, we perform iterative optimization such that we use the average DVD reduction value from simulation on the optimized design as the input DVD reduction assumption to the next-iteration optimization. Our experimental results show that such an iterative optimization converges (i.e., no improvement between two consecutive iterations) after the second iteration in most cases.

**(2) Find Gating Locations.** Based on the selected DVD hotspots, we traverse the netlist and determine the gating locations based on sensitivity functions. The objective of this optimization is to minimize the switching activities of non-scan cells within the selected DVD hotspots with minimized area and power overheads. In addition, such gating insertions must comprehend the datapath timing constraints at function mode and not create additional timing violations.

All gating cells need to connect to scan enable (SE) nets, which typically have a tree structure in a design block (shown in Figure 2.21(a)). Since the layout location of SE nets and the polarity of these SE net signals will affect wirelength penalty due to gating insertions as well as the types of gating cells used (e.g., AND gate versus OR gate), we extract the SE net information as input to our optimization. We divide the block area into $5\mu m \times 5\mu m$ grids, and for a grid containing SE nets, we assign the SE net to the grid so that a gating insertion within the grid will connect to this SE net. We break ties based on the levels of the SE nets, preferring to select a net closer to the bottom level so as to minimize the delay impact on fanout SE nets. For the grids without SE routing, we execute spiral search in their neighbor grids for the SE net with minimum distance (as shown in Figure 2.21 (b)).

Algorithm 7 describes our gating insertion flow. We perform power simulation and use $G_i$ to store the total switching power of cells within selected DVD hotspots $H$ at the $i^{th}$ iteration (Line 1). Note that here the cells are non-scan cells. Since we will not insert gating cells along the scan chain, we only visit non-scan cells when we traverse the netlist. In other words, we

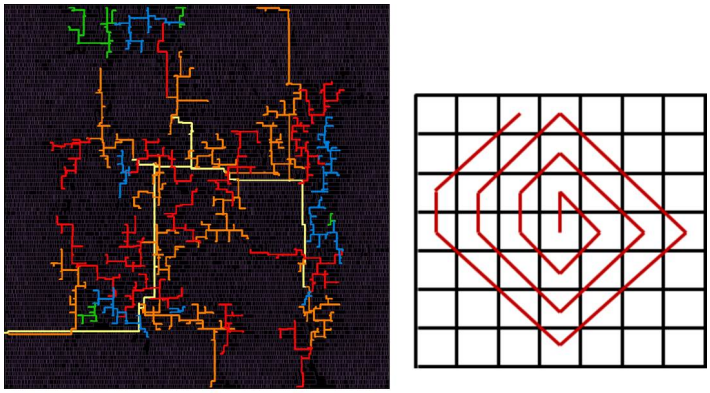



**Figure 2.21**: (a) Layout of scan enable (SE) nets. Different colors indicate different levels from the SE port. (b) Illustration of spiral search for SE nets in neighbor grids.

**Algorithm 7** Gating insertion.

1: $G_0 \leftarrow \sum_{c_i \in H} g_i$; $n \leftarrow 0$
2: **while** $n == 0$ —— $G_n < G_{n-1} \cdot (1-\theta)$ **do**
3: **for all** $i := 1$ to $N$ **do**
4: $f(c_i) \leftarrow 0$
5: **end for**
6: **for all** $c_i \in H$ **do**
7: $f'(c_i) = g_i$; $queue \leftarrow \emptyset$; $queue.push(c_i)$
8: **while** $queue \neq \emptyset$ **do**
9: $c_i \leftarrow queue.pop()$
10: **for all** $c_{i'} \in fanin(c_i)$ **do**
11: $f'(c_{i'}) \leftarrow SF(f'(c_i))$
12: **if** $c_{i'} \neq$ flip-flop **then**
13: $queue.push(c_{i'})$
14: **end if**
15: **end for**
16: **end while**
17: **for all** $i := 1$ to $N$ **do**
18: $f(c_i) \leftarrow f(c_i) + f'(c_i)$; $f'(c_i) \leftarrow 0$
19: **end for**
20: **end for**
21: $c \leftarrow$ cell with maximum $f(c_i)$ $\forall i \in [1,\ N]$
22: Insert gating at output pin of $c$; Update power
23: Size/$V_{th}$-swap $c$ to minimize power w.r.t. timing constraints
24: $n$++; $G_n \leftarrow \sum_{c_i \in H} g_i$
25: **end while**

only traverse along the nets which are candidates for gating insertion. At each iteration, we first initialize to zero the gain value $f(c_i)$ of each cell $c_i$ (Lines 3-5). We then start from each cell within the selected DVD hotspots and traverse backwards to calculate the gain values of their fanin cells, based on a sensitivity function $SF()$ (Lines 7-16). Due to large runtime of power

simulation, it is practically infeasible to perform exhaustive search within the fanin cone of cells in a given DVD hotspot to search for the gating insertion locations.[18] In our work, we search for gating locations based on gain values of cells, which are calculated based on sensitivity functions. We study different sensitivity functions based on netlist structure, logic function of cell instances, etc. Figure 2.22 shows the performance of various sensitivity functions (their sensitivity functions are also shown in the figure) and an example of sensitivity function-based gain value propagation. Each gating location of the optimal gating insertion solution is achieved with an exhaustive search. We observe that sensitivity function $SF = f \cdot duty\_cycle$ offers the best solution quality, where $duty\_cycle$ indicates the probability that a particular input signal is not masked by any other signal. The duty cycle can be estimated by logic function and static probability of other input signals of the gate. For example, having a '0' signal on one input of an AND gate will mask the other input signal. Therefore, $duty\_cycle$ of an input signal of an AND gate is the probability of '1' signal at the other input of the AND gate. We accumulate the gain values calculated based on cells within the selected DVD hotspots (Line 18). Last, we select the cell with maximum gain value to insert gating. Note that at this stage, we only insert gating cells in the netlist; ECO implementation steps (placement, routing) are executed only when all gating locations are determined. We perform sizing and $V_{th}$-swapping of the inserted gating cells to minimize power overhead while satisfying timing constraints (Line 23). We terminate the iterative optimization when the power reduction ratio is less than $\theta$ (Line 2).

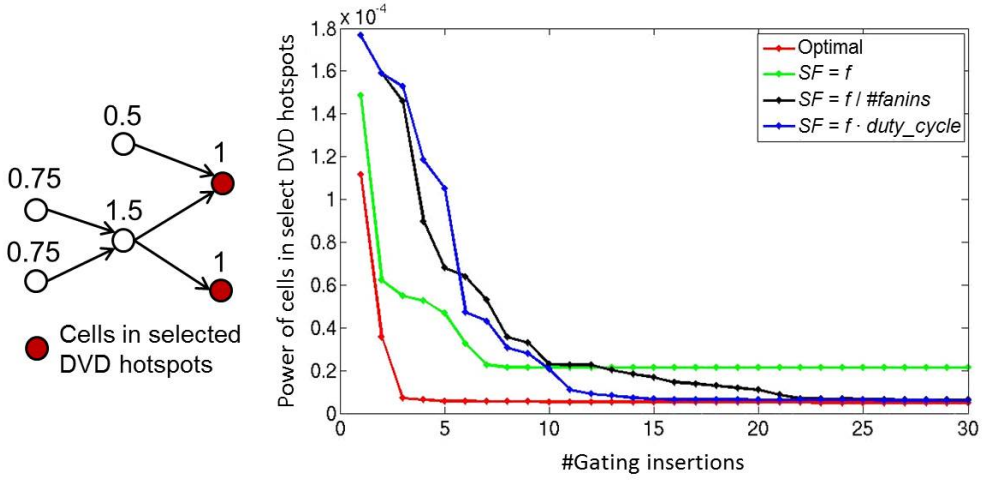



**Figure 2.22**: Performance of different sensitivity functions. Left figure shows an example of sensitivity function $SF = f/\#fanins$, where each cell within DVD hotspots has one unit of power.

[18] One iteration of exhaustive search in the fanin cone of 100 cells in a small design (e.g., 15K instances) can take more than four hours.

**(3) ECO Optimization.** Based on the identified gating insertion locations (in the netlist) and the design placement, we search for whitespaces near the gating cells and perform a matching optimization to determine the geometric locations for gating cell insertions.[19] We formulate the cost matrix using distances between the output pins and/or their fanout wire segments, and the nearby whitespaces. We then apply the Hungarian method to perform the matching optimization. We perform ECO placement legalization and routing after gating insertion.

### 2.3.3 Experimental Results

We perform experiments at $28nm$ LP foundry technology with dual-$V_{th}$ libraries. The supply voltage is $0.9V$. We use four designs – *DES*, *VGA*, *LEON3MP*, *NETCARD* – from ISPD-2012 contest[154] as our benchmarks. The benchmark information is shown in Table 2.14.[20] These designs are synthesized using *Synopsys Design Compiler vH-2013.03-SP3* [237] and then placed and routed using *Synopsys IC Compiler vI-2013.12-SP1* [239]. We use *Synopsys DFT Compiler vH-2013.03-SP3* to perform scan chain insertion. We set the maximum length of each chain to 250. We also perform scan compression in our implementation. We enable the DFT optimization options during placement and clock tree synthesis stages in *IC Compiler* to generate our initial scan chain solutions. We further use *Synopsys PrimeTime vH-2013.06-SP2* and *Synopsys PrimeTime-PX vH-2013.06-SP2 PT-PX* [240] for timing and power analysis, with wire parasitics (SPEF) obtained from *IC Compiler*.

We perform vectorless dynamic voltage drop (DVD) analysis using *ANSYS RedHawk* [232]. As inputs to DVD simulation, we report rise and fall arrival timing windows of all signal pins using *PrimeTime-SI* and report instance toggle rate and power information using *PT-PX*. Our DVD IR analysis is vectorless (with assumed 50% switching activity at test inputs), due to lack of open-source representative simulation vectors. To our understanding, this reflects common industry practice. We also note that our approach can be applied to scenarios with vector-based DVD analysis. We place power pads uniformly along the block periphery. Our scan ordering optimization is implemented in C++. Gating insertion flow is implemented in Tcl using *PT-PX* and *IC Compiler*. We conduct our experiments on a $2.5GHz$ Intel Xeon server.

[19] When there is no whitespace that satisfies the required minimum width, we either increase the region area to search for the available whitespaces or select whitespaces with smaller widths. However, these will incur larger timing impact due to longer wirelength and/or placement legalization.

[20] We use the same clock period for scan shift. We note that the much smaller clock period for scan shift as compared to those of industrial designs is due to the simple clock tree structure and single-block implementations.

**Table 2.14**: Benchmark information.

| **Design** | **Clock period** $(ns)$ | **#Instances** | **#Scan chains** |
|---|---|---|---|
| *DES* | 0.85 | 74035 | 45 |
| *VGA* | 1.10 | 80412 | 78 |
| *LEON3MP* | 2.00 | 474108 | 445 |
| *NETCARD* | 1.80 | 428974 | 358 |

### Scan Ordering

We perform scan reordering optimization at the post-routing stage to minimize the number of hold buffers. Table 2.15 shows our experimental results which include the number of hold buffers along scan chains, worst negative slack (WNS), total negative slack (TNS), total hold slack (THS) and total wirelength of the initial designs (orig) and of our optimized designs (opt). We observe that our optimization can remove up to 82% of hold buffers along scan chains (i.e., for the *LEON3MP* case). The optimized solution incurs negligible wirelength and timing penalties. We also observe wirelength reduction for large designs (e.g., for the *NETCARD* case).

**Table 2.15**: Scan ordering results.

| | | **#Hold buffers** | **WNS** $(ps)$ | **TNS** $(ns)$ | **THS** $(ns)$ | **Wirelength** $(mm)$ |
|---|---|---|---|---|---|---|
| *DES* | orig | **1296** | -21 | -0.089 | -0.101 | 765.9 |
| | opt | **487** | -21 | -0.081 | -0.121 | 766.3 |
| *VGA* | orig | **202** | -6 | -0.019 | -1.222 | 3087.9 |
| | opt | **89** | -6 | -0.018 | -1.223 | 3089.7 |
| *LEON3MP* | orig | **25581** | 30 | 0 | -0.734 | 11088 |
| | opt | **4538** | 30 | 0 | -0.705 | 11084 |
| *NETCARD* | orig | **30864** | -4 | -0.004 | -13.317 | 12729 |
| | opt | **26887** | -4 | 0.0 | -13.304 | 12720 |

### Gating Insertion

We perform gating insertion to minimize the timing slack degradation due to DVD. Table 2.16 shows our experimental results, where $\Delta$Slack indicates the scan timing slack degradation due to DVD; #Gating cells indicate the number of inserted gating cells; and DVD is the maximum DVD of the design. We observe that our optimization achieves up to 58% reduction of the slack degradation due to DVD (i.e., for the *DES* case). Our solution inserts only a

small number of gating cells, and has minimal area overhead (e.g., $< 1\%$). Since the number of inserted gates is small, the corresponding power and area overheads and impact on DVD in function mode is negligible. We further observe that the worst DVD value of a design is not necessarily correlated with slack degradation. In other words, it is the DVD on timing-critical scan cells, rather than the worst DVD in the design, that is more critical to optimize; this has not been captured by previous works.

**Table 2.16**: Gating insertion results.

| | | **ΔSlack** ($ps$) | **WNS** ($ps$) | **TNS** ($ns$) | **#Gating cells** | **DVD** ($mV$) | **Area** ($\mu m^2$) |
|---|---|---|---|---|---|---|---|
| *DES* | orig | **60** | -21 | -0.089 | - | 85 | 79662 |
| | opt | **25** | -21 | -0.079 | 36 | 84 | 79705 |
| *VGA* | orig | **159** | -6 | -0.019 | - | 93 | 120832 |
| | opt | **118** | -6 | -0.029 | 42 | 82 | 120873 |
| *LEON3MP* | orig | **471** | 30 | 0 | - | 129 | 699885 |
| | opt | **383** | 30 | 0 | 62 | 121 | 699969 |
| *NETCARD* | orig | **576** | -4 | -0.004 | - | 163 | 575869 |
| | opt | **496** | -8 | -0.008 | 111 | 147 | 576022 |

### 2.3.4 Conclusion

In this work, we propose a comprehensive scan timing optimization during late-stage IC implementation. We develop two optimization approaches: (i) scan reordering that is aware of clock skew and scan cell locations to remove hold buffers along scan chains, and (ii) gating insertion to minimize the DVD impact on scan timing slack. Our optimizations achieve up to 82% hold buffer reduction and 58% improvement of scan timing degradation due to DVD. Our future works include: (i) a more comprehensive scan ordering optimization to explore the tradeoff among wirelength, hold and setup timing of scan chains, (ii) a more comprehensive formulation of scan ordering cost (e.g., considering lockup latch insertion/removal when multiple scan clocks are driving cells along a scan subchain), (iii) co-optimization of gating insertion, scan ordering and test pattern generation to minimize the DVD impact on scan timing, (iv) DVD optimization during capture stage, and (v) a predictive model to determine DVD hotspots.

## 2.4 Acknowledgments

Chapter 2 contains reprints of Kun Young Chung, Andrew B. Kahng and Jiajia Li, "Comprehensive Optimization of Scan Chain Timing During Late-Stage IC Implementation", *Proc. ACM/IEEE Design Automation Conference*, 2016; Tuck-Boon Chan, Andrew B. Kahng, Jiajia Li, Siddhartha Nath and Bongil Park, "Optimization of Overdrive Signoff in High-Performance and Low-Power ICs", *IEEE Transactions on Very Large Scale Integration Systems* 23(8), 2015; 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/IEEE Design Automation Conference*, 2015; and Tuck-Boon Chan, Andrew B. Kahng, Jiajia Li and Siddhartha Nath, "Optimization of Overdrive Signoff", *Proc. Asia and South Pacific Design Automation Conference*, 2013. The dissertation author is the primary author of the papers.

I would like to thank my coauthors Tuck-Boon Chan, Kun Young Chung, Kwangsoo Han, Andrew B. Kahng, Jongpil Lee, Siddhartha Nath and Bongil Park, as well as the research support from Samsung Electronics.

# Chapter 3

# Low-Power Optimization

Energy and battery lifetime constraints are critical challenges for IC product designs. This chapter presents three distinct techniques for low-power optimization, which address the low-power requirement in three distinct contexts – system, clock and datapath. First, we present an optimization framework for stacked-domain designs. Based on an initial placement solution, we apply a flow-based partitioning to partition cells into two power domains with balanced cross-domain current and minimized number of inserted level shifters. The partitioning is aware of multiple operating scenarios, cell placement, and timing-critical paths. We further propose heuristics to define regions for each power domain so as to minimize placement perturbation. Our method achieves more than $\sim$10% and 2$\times$ battery lifetime improvements in function and sleep modes, respectively. Second, we propose an optimization flow to generate and place flop trays (multi-bit flip-flops) from a library of arbitrary given sizes and aspect ratios (ARs), to achieve clock network power reduction. Our optimization starts with an initial placement solution using only single-bit flops. It then performs capacitated K-means clustering to generate solutions with different flop tray sizes and ARs. Our optimization is aware of flop tray sizes and ARs, as well as timing-critical start-end pairs. Results in foundry 28FDSOI technology show up to 16% total block power reduction over designs with flop trays generated by logical clustering during synthesis. Third, we use resilient designs with minimized overheads to reduce power on datapaths. Our methodology uses two levers: selective-endpoint optimization (i.e., sensitivity-based margin insertion) and clock skew optimization. We integrate the two optimization techniques in an iterative optimization flow which comprehends toggle rate information and the tradeoff between cost of resilience and margin on combinational paths. Our proposed flow achieves energy reductions of up to 21% and 10% compared to a conventional (with only margin

used to attain robustness) design and a brute-force implementation (i.e., a typical resilient design, where resilient endpoints are (greedily) instantiated at timing-critical endpoints), respectively.

## 3.1 Floorplan and Placement Methodology for Improved Energy Reduction in Stacked Power-Domain Design

Energy and battery lifetime constraints induce new and critical challenges to IC designs, especially for mobile and IoT (Internet of Things) applications. To achieve power autonomy in the era of a slowing Moore's law, new low-power techniques must be exploited. While many low-power techniques [53] have concentrated on the circuit side of system design, power management techniques have received growing attention due to the importance of power efficiency. Notably, the misalignment of battery voltages compared to scaled core voltages causes inefficiencies that present significant opportunities for power saving. In order to better align SoC power domain voltages with battery voltages, *stacked power domain* (or *voltage stacking*) has been proposed [131][162][187].

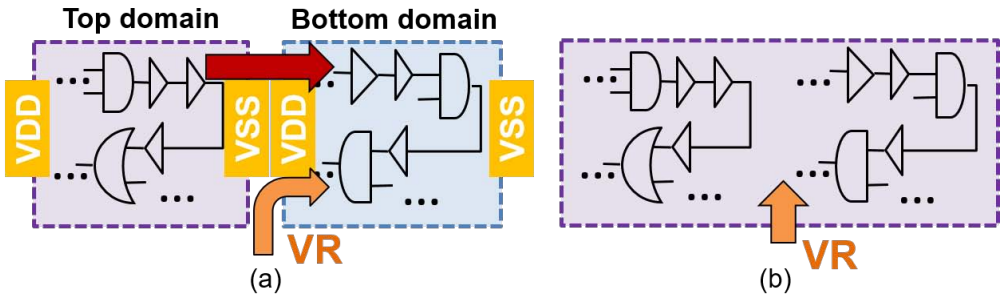



**Figure 3.1**: Comparison between (a) stacked power-domain design, versus (b) conventional design. VR indicates voltage regulator. The orange arrows indicate current from voltage regulators. The red arrow indicates stacked current.

Figure 3.1 illustrates the basic idea of stacked power domain. A stacked power-domain (or stacked-domain) design connects in series power domains that are connected in parallel in a conventional design.[21] Figure 3.1 shows that one power domain (i.e., the top domain) is placed over the other (i.e., the bottom domain) to double the voltage and (ideally) halve the current compared to that in a conventional design. More specifically, if the supply voltage of a conventional design is $V$ (i.e., $VDD = V$ and $VSS = 0$), then the $\{VDD, VSS\}$ of the top and bottom

[21] Our study focuses on optimization with two power domains (i.e., top and bottom domains) and conventional planar monolithic implementation (as opposed to three-dimensional integration). In other words, both stacked-domain and conventional designs in the following discussions are planar monolithic implementations. Stacked-domain optimization with more than two power domains and/or in 3DICs is left as a direction for future research.

domains in the corresponding stacked-domain design are $\{2V, V\}$ and $\{V, 0\}$, respectively. For bulk CMOS, note that the $\{2V, V\}$ top domain must be placed on a deep n-well so that the bulk potentials can be maintained at ($2V$, $V$). Moreover, in the ideal case the current is balanced across the two domains. The stacked-domain scheme provides implicit 2:1 downconversion of external supplies. In light of this, there is no need to employ a bulky supply to generate the supply voltage for the core (i.e., gate instances and memories). Instead, it suffices to employ a much smaller converter that acts only as a watchdog to the supply rail that connects the power domains. This results in increased power efficiency for the overall system [19].

**Table 3.1**: Description of notations used in our discussion.

| Term | Meaning |
|---|---|
| $P_{ext}$ | total input power from external supply (e.g., battery) |
| $P_{VR,in}$ | input power of voltage regulator from external supply |
| $P_{VR,out}$ | output power of voltage regulator to core |
| $\eta_{VR}$ | power conversion efficiency of voltage regulator |
| $P_{stk}$ | direct power of stacked power domain from external supply |
| $P_{core}$ | total power consumption of core |
| $I_{stk}$ | stacked current (current from top domain to bottom domain) |
| $I_{VR}$ | output current from voltage regulator |
| $T$ | battery lifetime |

Based on the power conversion modeling proposed in [20], we derive the battery lifetime improvement from stacked-domain optimization as follows. (Table 3.1 lists the notations used in our discussion.) Since battery lifetime ($T$) is inversely proportional to $P_{ext}$, we compare $P_{ext}$ of a stacked-domain design to that of a conventional design.[22] By definition (see Table 3.1), in a stacked-domain design we have

$$P_{ext} = P_{stk} + P_{VR,in} = P_{stk} + P_{VR,out}/\eta_{VR} \tag{3.1}$$

On the other hand, in a conventional design, power is only supplied through the voltage regulator. Thus, the total input power from the external supply of a conventional design ($P'_{ext}$) is calculated as

$$P'_{ext} = P_{core}/\eta_{VR} \tag{3.2}$$

[22] We use battery lifetime ($T$) as the metric to evaluate energy improvement achieved by our proposed methodology. We also report power values of core ($P_{core}$) and the entire system ($P_{ext}$) from our optimization in Table 3.3.

By assuming the same core power consumption in both stacked-domain and conventional designs, we have

$$P_{core} = P_{stk} + P_{VR,out} \tag{3.3}$$

Furthermore, based on the model described in [20] which assumes that the voltage regulator power efficiency is the same for both cases, we have

$$P_{VR,out}/P_{core} = I_{VR}/(2 \cdot I_{stk} + I_{VR}) \tag{3.4}$$

Finally, based on the above analyses, the battery lifetime ratio between the stacked-domain design ($T$) versus the conventional design ($T'$) is[23]

$$\frac{T}{T'} = \frac{P'_{ext}}{P_{ext}} = \frac{2 \cdot I_{stk} + I_{VR}}{2 \cdot \eta_{VR} \cdot I_{stk} + I_{VR}} \tag{3.5}$$

We observe that the battery lifetime benefit from a stacked-domain implementation increases with a smaller $I_{VR}$. As a motivating example, if the current is perfectly balanced between two domains (i.e., $I_{VR}$ = 0), assuming $\eta_{VR}$ = 80%, the stacked-domain implementation provides 25% battery lifetime improvement over the conventional implementation. Moreover, since the power efficiency of the voltage regulator decreases with small supply currents, the battery lifetime benefit from the stacked-domain implementation is expected to be higher for designs in low-power modes that use a voltage regulator optimized for high-power cases.

Although the stacked-domain implementation provides significant battery lifetime improvement, it also raises non-trivial implementation methodology challenges that must be solved. First, the communication between the power domains must be ensured by level shifters that can convert such extreme signal levels. Second, the power efficiency improvement is directly dependent on the current balancing between the two power domains. In other words, the design must be bipartitioned in terms of current consumption. We also note that such a partitioning optimization must comprehend multiple operating scenarios, area and power penalties as well as timing impact of level shifters, as well as additional placement constraints imposed by region definition of power domains. The first challenge has been thoroughly investigated, with and several different level shifter architectures having been proposed [162][187]. However, the optimization of partitioning and layout planning of the designs has remained an open challenge that prior works (which have mostly been ad hoc or design-specific) do not ultimately answer for general systems. In this work, we address this open challenge and provide a comprehensive optimization framework for partitioning and floorplanning of stacked-domain implementation that can be used for a wide range of systems.

[23] $\frac{P'_{ext}}{P_{ext}} = \frac{P_{core}/\eta_{VR}}{P_{stk}+P_{VR,out}/\eta_{VR}} = \frac{(2 \cdot I_{stk}+I_{VR})/\eta_{VR}}{2 \cdot I_{stk}+I_{VR}/\eta_{VR}}$. See also [20].

The contributions of this work are as follows.

- We propose a comprehensive optimization framework for stacked-domain implementation. Key elements include a flow-based partitioning with layout and timing-path awareness, heuristics for layout region generation of power domains, and a matching-based optimization for level shifter insertion.
- We are the first to propose a partitioning optimization at the sub-block level for stacked-domain implementation that can be used for a wide range of systems.
- We validate our optimization flow on industrial designs, in the context of an industrial implementation flow that includes placement, clock tree synthesis and routing.
- Our optimization accommodates current balancing constraints between stacked domains, and multiple partitioning scenarios with respect to movement of hard macros and logic across the stacked domains.
- Using the power delivery block described in [19], our optimized stacked-power domain designs achieve more than 10% and 2× battery lifetime improvement compared to the conventional designs in function and sleep modes, respectively.

### 3.1.1 Related Work

In this section, we review the previous literature on (i) stacked-domain implementation and (ii) netlist partitioning.[24]

#### Stacked-Domain Implementation

The circuit blocks needed for a stacked-domain implementation – such as level shifters and voltage regulators – are well-studied in the literature. However, to our best knowledge, no existing work is able to fully automate the implementation flow of a stacked-domain design. Various voltage regulators and level shifters have been studied in [162][187], but the designs used in their studies lack the complexity of a realistic application. A smart regulation scheme has been proposed in [131], and the studied design has relatively higher complexity, featuring processor

[24] Our stacked-domain optimization problem is different from the power-island generation problem [197][36][79], in that the power-island generation optimization assumes different supply voltages for power domains and minimizes the power overhead from voltage assignments, while our optimization exploits charge recycling by balancing current across domains. Moreover, many critical issues such as timing impact of level shifters, insertion of shifter rows, region definition of power domains, etc. are not addressed in power island-related works.

cores. At the same time, in the work of [131] there is no connection between different processor cores, which makes the partitioning problem much simpler. Similarly, [139] and [24] only focus on specific design blocks such as IO cells and memories. A recent work [20] applies stacked-domain optimization to a complete MCU system designed with a standard design flow. The partitioning approach presented in [20] is somewhat ad hoc, and is not applicable to a general design. By contrast, here we present a comprehensive optimization framework for stacked-domain implementation that is applicable to a wider range of designs.

**Netlist Partitioning**

As a classic problem in VLSI optimization, netlist partitioning has been thoroughly studied in previous literature. A comprehensive, still-relevant taxonomy of approaches is given in [8]. We highlight four basic partitioning approaches.

**Move-based approach.** To partition a given set of vertices into two partitions with balanced weights and minimized number of hyperedge cuts, Kernighan-Lin [119] and Fiduccia-Mattheyses [66] iteratively move or swap vertices guided by gain functions (within a pass-based structure) to achieve a local optimal solution. This greedy iterative improvement approach is efficient and leads to a relatively good solution quality. Important improvements and/or extensions have been proposed, such as multi-way partitioning [116], multi-level extension [25], timing path awareness [115], and "lookahead" gain functions (e.g., gain vectors, CLIP/CDIP and LIFO gain buckets, etc. [56][82][124]).

**Mathematical programming-based approach.** Other works formulate mathematical programs to optimize the netlist partitioning. Shih et al. [173] formulate the timing-aware partitioning problem as quadratic boolean programming. They minimize the total cost of cell-to-partition assignments as well as the number of cuts, with respect to capacity and timing constraints. Goemans et al. [76] use semidefinite programming for partitioning optimization. However, their objective is to maximize the number of cuts under capacity constraints.

**Flow-based approach.** In light of the max-flow min-cut theorem, Yang et al. [206] propose to use repeated max-flow computations and clustering operations to achieve a balanced bipartitioning solution. The work of [26] documents high efficiency and relatively good solution quality of flow-based partitioning with a min-cut objective.

**Clustering approach.** Netlist partitioning can also be achieved by bottom-up clustering. For example, Rajaraman et al. [165] propose a clustering approach to minimize the delay from PIs to POs. With a maximum-area constraint for each cluster, they iteratively cluster cells until all cells are clustered.

In this work, we apply the flow-based approach [206] to partition instances into two power domains. We propose several extensions to the existing flow-based partitioning including layout and timing-path awareness, multi-scenario weight (i.e., current) balancing, and a prior clustering step for runtime reduction.

### 3.1.2 Methodology

We now describe our optimization framework for stacked-domain logic design partitioning implementation. We first state our stacked-domain optimization problem as follows.

**Given:** A netlist, timing constraints, level shifters, voltage regulator efficiency, and switching information of instances in the netlist,

**Do: partition** the netlist instances into two domains, **define** the layout region of each domain, and **place** instances and level shifters, **such that** battery lifetime is maximized.

As implied by Equation (3.5), to maximize the battery lifetime, our basic objective is to balance the current between the two stacked power domains, while minimizing the power penalty due to level shifter insertion.

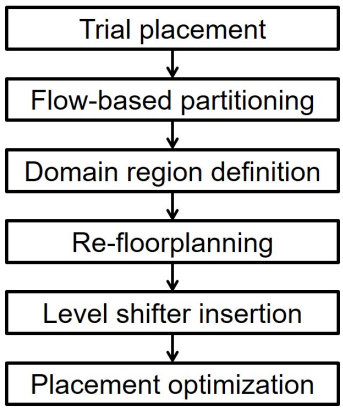



**Figure 3.2**: Overall optimization flow.

Figure 3.2 shows our overall optimization flow. A common practice in stacked-domain implementations is to partition the netlist (i.e., define the power domain of each instance or block) prior to the floorplanning stage [20]. However, performing a power domain assignment before placement can result in suboptimal floorplan and placement solutions. More importantly, the placement optimization inserts buffers and upsized cells, which can change the current profile of each power domain. As a result, currents that have been balanced during the partitioning stage are no longer balanced after the placement stage. To resolve this, we propose to perform a *trial placement*, based on which we perform a layout-aware partitioning (with minimized number of cuts as well as placement perturbations) to assign instances to power domains.

Figure 3.3(a) shows an example of our layout-aware partitioning solution on design *AES* [230] in 28LP technology. In blue are instances assigned to the bottom domain and in red are instances assigned to the top domain. Based on the partitioning solution, we define the layout region for each power domain such that each domain has a continuous region (Figure 3.3(b)). Note that since gaps must be inserted along the boundary between two power domains, we propose a dynamic programming optimization to minimize the boundary length between two domains. We then legalize instance placements within the (updated) region for each power domain using a commercial P&R (place-and-route) tool [215]. We then update the floorplan by shifting the power domains (as shown in Figure 3.3(c)) and inserting level shifters.[25] We perform a matching-based optimization to determine level shifter placement locations that minimize wirelength. In Figure 3.3, in yellow are level shifters. Last, we perform an incremental placement optimization to fix timing violations.

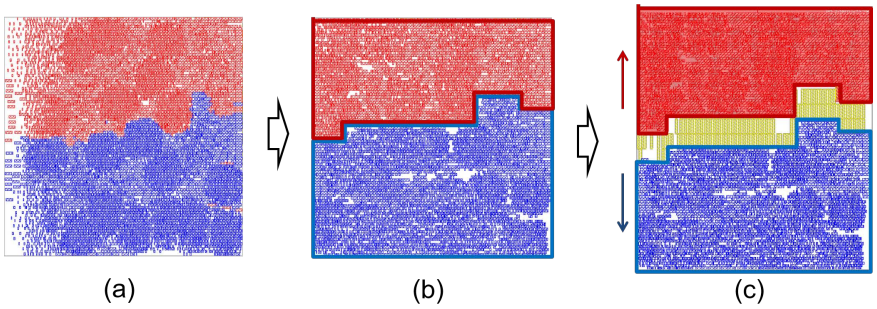



**Figure 3.3**: Example of optimization: (a) layout-aware partitioning, (b) region definition of power domains, and (c) level shifter insertion in the updated floorplan. Design: *AES* (∼11K instances). Technology: 28LP.

### Flow-Based Netlist Partitioning

The greedy iterative partitioning approach is not naturally amenable to timing path-aware partitioning, and has no mechanism to preserve solution structure of an initial (trial) placement. And, mathematical programming-based approaches typically have large runtimes. Thus, we apply the flow-based approach described in [206] to partition instances into two power domains. Our objectives include (i) to minimize the number of cuts, which reduces timing, area and power penalties from level shifter insertions, and (ii) to minimize the perturbation to the

[25] We understand that modification of the block size might not be consistent with certain implementation flows. At the same time, we believe that performing the initial trial placement (with appropriate instance bloating) in a block having Figure 3(c)'s shape will not diverge significantly from the initial trial placement in Figure 3(a), particularly with improved (smaller) level shifter designs. Ongoing work is aimed at a predictive (or, "one loop") methodology to determine the block size prior to trial placement.

initial placement solution. Figure 3.4 illustrates the basic idea of the flow-based partitioning. To construct the flow network, following the approach in [206], we insert a vertex for each cell or cluster of cells. For each net, we insert two nodes and a bridging edge of unit capacity between the two nodes. For each cell or cluster of cells incident to a net, we insert two edges of infinite capacity between the vertex corresponding to the cell or cluster and each of the two nodes corresponding to the net. Finally, the weight of a vertex corresponding to a cell or a cluster is estimated based on the current of the cell or cluster, while the weights of vertices corresponding to nets are set as zero. According to the max-flow min-cut theorem, the approach finds the partitioning solution with the minimum number of cuts for a given netlist via a max-flow optimization. However, this does not guarantee that the balancing constraint is met. To address this, after each max-flow optimization, the approach clusters the vertices belonging to the smaller partition together with one neighbor vertex (to avoid obtaining the same partitioning solution) into one super vertex. Based on the updated flow network, another max-flow optimization is performed. The approach iteratively performs (incremental) max-flow optimization and clustering until the balancing constraint is satisfied. In other words, the iterative max-flow optimization and clustering procedure keeps track of the aggregated current until the currents of two partitions are balanced.

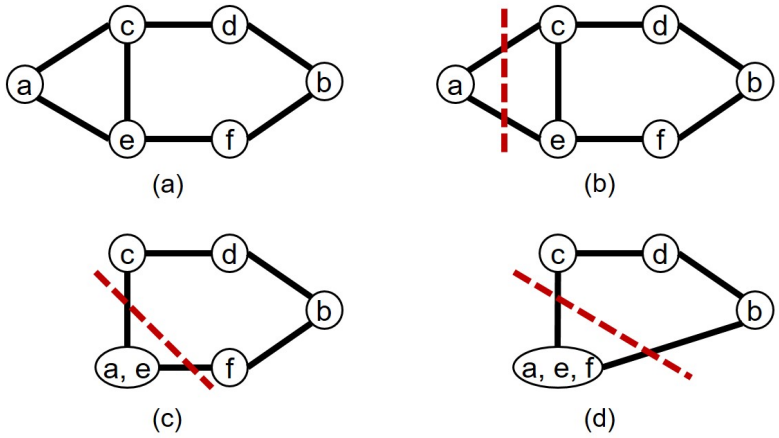



**Figure 3.4**: Flow-based partitioning. a and b are source and sink, respectively. All vertices have the same weight. Red dotted lines indicate cuts. (a) Initial flow network. (b) First max-flow min-cut computation. (c) Clustering operation. (d) Second max-flow min-cut computation.

We adopt the flow-based partitioning approach to our stacked-domain optimization with the following five extensions. Algorithm 8 describes our partitioning procedure.

**Extension 1. Source and sink selection.** The approach in [206] randomly picks two nodes (instances) in the flow network (netlist) as the source and sink nodes. However, there are cases in

**Algorithm 8** Flow-based partitioning.

```
1: Pre-cluster cells into clusters
2: Define source and sink vertices
3: ΔI ← +∞
4: while ΔI > Δ_max do
5:     Perform max-flow optimization to achieve the min-cut solution
6:     Remove outliers (layout awareness)
7:     Remove V-shaped vertices (timing-path awareness)
8:     Cluster the smaller partition and one neighbor
9:     Update ΔI
10: end while
```

which the flows between selected source and sink vertices cannot cover the entire flow network, resulting in unbalanced partitioning solutions. As an example, the selection of vertices a and b as the source and sink from the flow network shown in Figure 3.5(a) will not able to achieve a balanced partitioning solution. To address this, we add a supersource and a supersink and connect them to multiple vertices (e.g., PIs and POs in a netlist, or instances located at the core boundary) with edges of infinite capacity to minimize the number of uncovered vertices (instances) as shown in Figure 3.5(b).

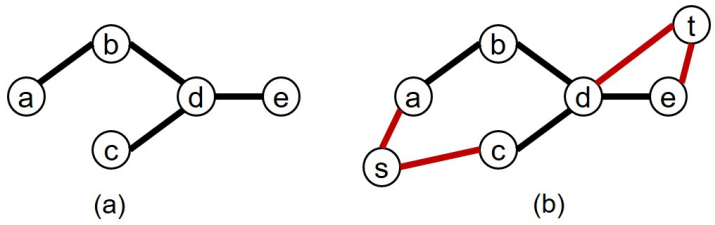



**Figure 3.5**: (a) Choosing a / b, or c / d, or d / c as source / sink cannot lead to a balanced solution. (b) Adding a supersource (s) and a supersink (t) resolves the issue. Edges in black have unit capacities. Edges in red have infinite capacities.

**Extension 2. Layout awareness.** To avoid excessive placement perturbation, which can result in current profile changes and thus power penalty, the partitioning optimization must be aware of trial placement locations of the instances – such that instances partitioned into the same power domain are placed close to each other in the original trial placement. We achieve this required layout awareness in two ways. (i) We only select the instances located close to each other to connect to the supersource (or supersink). As an example, we select instances located within a particular distance (e.g., ten cell rows) from the bottom (resp. top) core boundary to connect to the supersource (resp. supersink). (ii) After each max-flow optimization, we detect *outliers*, which are instances belonging to the larger-current partition that are located within a region with a majority of instances belonging to the smaller-current partition. We then cluster these outliers with the instances from the smaller-current partition (Line 6 in Algorithm 8).

**Extension 3. Critical-path awareness.** Ignoring signal flow direction and timing path structure during the partitioning optimization can easily result in multiple cuts along one timing path. We extend the partitioning flow in [206] to minimize the number of cuts along timing-critical paths. Similarly to the layout awareness extension discussed above, after each max-flow optimization we detect "V-shaped vertices" [115], which are a sequence of instances belonging to the larger-current partition along a timing-critical path, where the fanin and fanout instances of these instances along the timing-critical path are in the smaller-current partition. We then collapse (cluster) the instances corresponding to the V-shaped vertices into the smaller-current partition as long as this does not violate the balancing constraints (Line 7 in Algorithm 8).

**Extension 4. Pre-clustering.** Although the max-flow optimization can be achieved with an incremental flow computation and the entire optimization takes $O(N)$ iterations to converge where $N$ is the number of instances in a design, the runtime in practice can be substantial for a large design. To reduce the runtime, we perform a pre-clustering optimization based on the heavy-edge matching (HEM) strategy [117]. We enforce layout-awareness constraints (i.e., an upper bound on the distance between two vertices that can be clustered) during the HEM. Figure 3.6 shows an example of the HEM clustering up through 18 levels (clustering ratio = 0.76 at each level), showing how instances within the same cluster are spatially proximate. But since we are limited by 64 available colors, different clusters can have the same color. Also, clusters with small size might not be visible from the figure. Our experimental results show that we can reduce runtime by 75% (two HEM levels and overall clustering ratio of 0.5) with negligible degradation of solution quality (e.g., cut number).

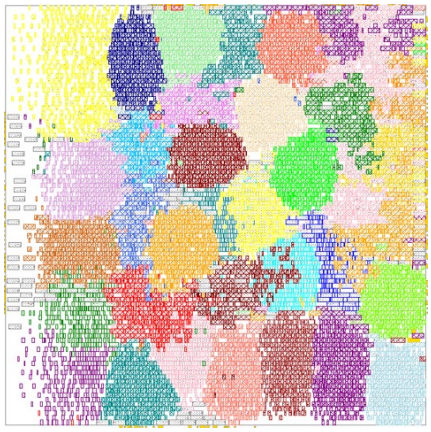


**Figure 3.6**: HEM clustering solution. Different clusters are indicated by different colors. #Clusters = 200. Levels of clustering = 18. Clustering ratio at each level = 0.76. Design: *AES*. Technology: 28LP.

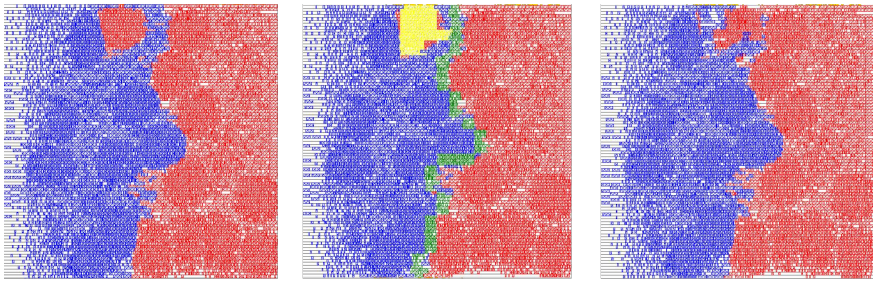



**Figure 3.7**: FM-based grid movement. (a) Initial placement solution. (b) In yellow are outliers of the top domain. In green are neighboring grids of the top domain. (c) Post-movement placement, where each domain has a continuous region. Design: *AES*. Technology: 28LP.

**Extension 5. Multiple operating scenarios.** To ensure high power efficiency across different operating scenarios, the partitioning optimization must balance currents between two domains across multiple scenarios. To achieve this, we use the weighted sum of normalized currents from different scenarios during our optimization (Line 9 in Algorithm 8). Specifically, the delta current is calculated as

$$\Delta I = \sum_i \left(w^i \cdot |I^i_{top} - I^i_{bot}| \;/\; (I^i_{top} + I^i_{bot})\right) \tag{3.6}$$

where $I^i_{top}$ and $I^i_{bot}$ are respectively the currents of top and bottom domains in the $i^{th}$ mode, and $w^i$ is the weighting factor of the $i^{th}$ mode, such that $\sum_i w^i = 1$. Our optimization ensures that $\Delta I$ does not exceed a predefined upper bound (i.e., $\Delta_{max}$ in Algorithm 8).[26]

### Domain Region Definition

In this subsection, we describe our methodologies to define the layout region (power island) for each power domain. The definition of the layout region for each power domain affects the design quality in two fundamental ways. (i) Gap area must be inserted along the boundary between different power domains. Therefore, a longer boundary length will lead to higher area penalty. (ii) The power domain definitions will have downstream impact on the PDN (power delivery network) design, which is not yet implemented at this point. Therefore, it is desirable to adjust the power domain definitions for minimized area, power and performance penalties.[27] If the partitioning and the trial placement results in discontinuous power domains, the length of the power domain boundaries is highly likely to be longer compared to the case when the regions of each power domain are merged. Moreover, the power routing will be more difficult, since

[26] A *mode* is an operating scenario, such as sleep mode or function mode.

[27] Since the power domain definitions change after our partitioning optimization, in our implementation flow we perform re-floorplanning with updated power grids.

different power rails will need to be routed to discontinuous power domain regions. Thus, we seek to have only two regions (i.e., power islands) corresponding to the two power domains.

Although our partitioning optimization is layout-aware, there can still be separated regions for each power domain. Figure 3.7(a) shows an example trial placement and partitioning solution where the top domain (shown in red) has two separated regions. To merge the regions while minimizing placement perturbation (e.g., wirelength increase), we perform an FM-based grid movement optimization (i.e., an iterative, swap-based greedy algorithm as described in Algorithm 9). We first divide the core area into grids. The power domain of each grid if defined as the power domain of majority instances within the grid. We then define the outliers (i.e., grids outside the largest continuous region of the corresponding domain) and neighboring grids (i.e., grids adjacent to the largest continuous region of the different domain) (Line 1). Figure 3.7(b) shows an example of outliers and neighboring grids. We calculate the cost to swap pairs of outliers and neighboring grids (Lines 2-8). We iteratively swap the pair of an outlier and a neighboring grid with the minimum movement cost, until all outliers (e.g., yellow grids in Figure 3.7(b)) are removed (Lines 9-16). In Figure 3.7(c)), the small number of remaining outliers are minority instances in their grids, and will be legalized during an incremental placement.

**Algorithm 9** FM-based grid movement.

1: $U \leftarrow$ find outliers; $H \leftarrow$ find neighboring grids
2: **for all** $u \in U$, $h \in H$ **do**
3:  **if** $u.domain \neq h.domain$ **then**
4:  $cost(u, h) \leftarrow$ half-perimeter wirelength increase by swapping $u$ and $h$
5:  **else**
6:  $cost(u, h) \leftarrow +\infty$
7:  **end if**
8: **end for**
9: **while** $U \neq \emptyset$ **do**
10:  $(u', h') \leftarrow Min_{cost(u,h)}\{(u, h) \mid u \in U,\ h \in H\}$
11:  swap $u'$ and $h'$
12:  **if** create new outliers **then**
13:  revert the swap; $cost(u', h') \leftarrow +\infty$
14:  **end if**
15:  update $U$, $H$ and costs
16: **end while**

In the last step of domain region definition, we apply dynamic programming to minimize the length of the boundary between two power domains while maintaining the area within each domain. As the base cases, we calculate the boundary length decrease of each boundary segment by simplifying the boundary shape (e.g., highlighted segment in Figure 3.8). We note that such simplification must meet an upper bound of moved area (i.e., total area with changed domain

assignment). Assuming that the (turning) points along the boundary are indexed from left to right or from bottom to top, the recurrence relation in our dynamic programming optimization is

$$Sol(j) = Min(Sol(i).length + seg(i,j).length),\ \forall 1 < i < j \tag{3.7}$$

where $Sol(j)$ is the optimized boundary solution from the first point to $j^{th}$ point, and $seg(i,j)$ is the simplified boundary segment between the $i^{th}$ and the $j^{th}$ points. The dynamic programming-based boundary simplification has $O(M^2)$ time complexity, where $M$ is the number of points or segments. The time complexity further decreases to $O(M)$ if we only search a limited range of existing sub-solutions (i.e., $i$ in Equation (3.7)).

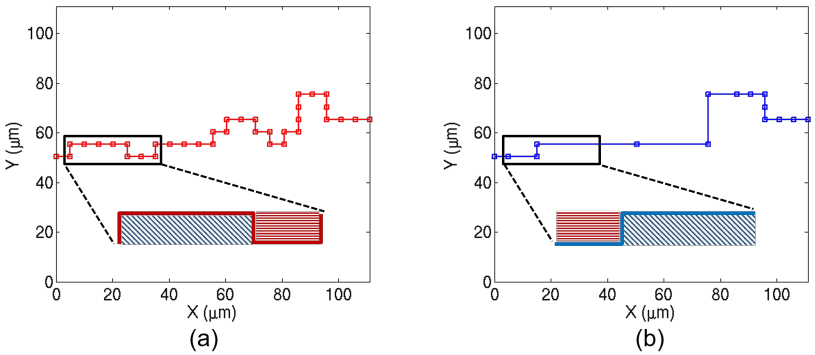



**Figure 3.8**: Boundary optimization. (a) Original boundary. (b) Optimized boundary with smaller length. An example of segment optimization is shown. Optimized segments have smaller total length while maintaining the same area in each power domain. Design: *AES*. Technology: 28LP.

### Level Shifter Insertion

In the last step of our optimization, we insert and place required level shifters between the top and bottom domains, and perform re-floorplanning if the total cell area exceeds the block floorplan area. Specifically, we define placement regions for level shifters, where each region must have an even number of level shifter rows due to deep n-well sharing. Furthermore, we assume that the layout of the level shifter has already included the boundary of the deep n-well of either or both power domains, and that the edges facing either of the power domains have a standard-cell row structure. As a result, we are able to seamlessly integrate the level shifters with only a small separation (i.e., 2.5$\mu$m) at left and right ends of each level shifter row from standard cells. The row height of our level shifter is 6X of the standard-cell row height.[28] Furthermore, additional space (i.e., $\sim$10% of area within each level shifter placement region) is required for

[28] Our level shifter model is from our industry collaborators.

tie and decap cell insertion. The objectives for our level shifter placement optimization are to minimize the area overhead and minimize the wirelength penalty due to level shifter region definition and level shifter placement, respectively.

**Algorithm 10** Level shifter placement.

1: Matching optimization between candidate locations and level shifters
2: Level shifter placement
3: Clustering of level shifters to define level shifter placement regions
4: Placement blockage insertion and candidate locations update
5: Matching optimization between updated candidate locations and level shifters
6: Level shifter placement
7: Clumping level shifters to create space for tie / decap cell insertion
8: Legalization of standard cells

Since the level shifter insertion approach in [18] does not comprehend the above layout constraints as well as the space for tie / decap cell insertion, it cannot be applied in a realistic implementation of a stacked-domain design. We therefore propose a new level shifter placement approach. Algorithm 10 shows our level shifter placement procedure. We first perform a matching optimization (using the Hungarian algorithm [222]) to map each inserted level shifter to a candidate placement location (i.e., a level shifter placement site near the boundaries of power domains) with minimized wirelength (Line 1). More specifically, we enumerate possible placement locations near the boundaries between two domains and calculate the potential cost of placing each level shifter onto each candidate placement location. We define the cost as the total HPWL (half-perimeter wirelength) of nets connected to the level shifter. Based on the cost matrix, we perform matching optimization to assign the placement location for each level shifter while minimizing the total cost. Note that such a level shifter placement solution does not honor the layout constraint where each region must contain an even number of level shifter rows. We therefore cluster level shifters that are separated by a distance that is smaller than a predefined value (e.g., 20$\mu$m) and create a region having an even number of rows for each cluster of level shifters (Line 3).

According to the clustering solution, we generate placement blockages for standard cells and update candidate locations for level shifter placement (Line 4). Importantly, we also comprehend spacing requirement at the ends of each level shifter row during placement blockage insertion. We then perform another iteration of matching optimization based on the updated candidate placement locations, and place level shifters accordingly (Lines 5, 6). Observe that in the above optimizations we use level shifters with bloated (e.g., by 10%) widths. We now recover the level shifters' cell widths and clump them to create space for tie / decap cell insertion

(Line 7). Finally, we perform placement legalization of standard cells to move them out from the created level shifter placement regions. Figure 3.9 shows an example of level shifter placement.

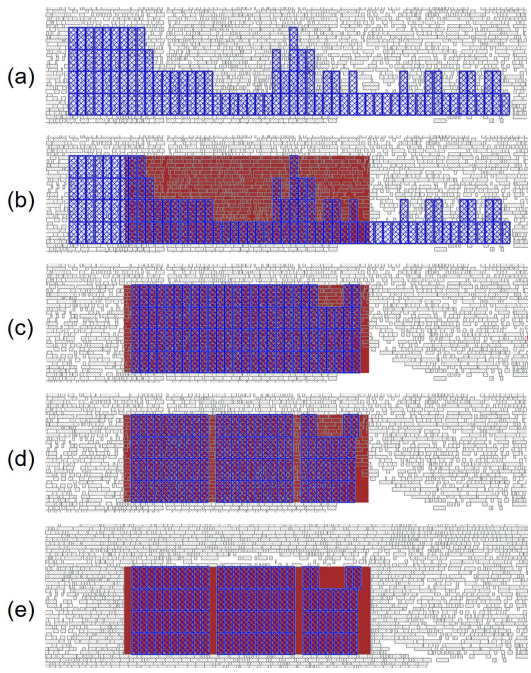



**Figure 3.9**: Example of level shifter insertion. (a) Level shifter (in blue) placement after first matching. (b) Placement blockage (in red) insertion. (c) Level shifter placement after second matching. (d) Clumping of level shifters. (e) Placement legalization applied to nearby standard cells.

### 3.1.3 Experimental Results

**Table 3.2**: Testcase parameters.

| Design | Technology | #Instances | #Flops | Clock period |
|---|---|---|---|---|
| *AES* | 28$nm$ LP | ∼11K | 530 | 1.2$ns$ |
| *DES* | 28$nm$ LP | ∼17K | 530 | 1.4$ns$ |
| *JPEG* | 28$nm$ LP | ∼42K | 4512 | 1.6$ns$ |
| *VGA* | 28$nm$ LP | ∼58K | 17053 | 2.0$ns$ |
| *TC1* | 40$nm$ | ∼106K | 15245 | 20$ns$ |
| *TC2* | 40$nm$ | 1.00 | 0.18 | 20$ns$ |

We perform experiments in a 28$nm$ LP foundry technology with dual-$V_{th}$ libraries. We use four design blocks (*AES*, *DES*, *JPEG*, *VGA*) from *OpenCores* [230] as our testcases. Parameters of these four testcases are shown in Table 3.2.[29] The worst-case timing and power

[29]The instance and flip-flop counts of design *TC2* are normalized with respect to the corresponding (instance and

analysis view for *AES*, *DES*, *JPEG* and *VGA* is (SS, 0.95$V$, 125°$C$). We synthesize designs using *Synopsys Design Compiler vI-2013.12-SP3* [237] and then place and route using *Cadence Innovus Implementation System v16.1* [215]. We set the placement density at the floorplan stage as 70%, and perform timing and power analyses using *Cadence Innovus Implementation System v16.1*. We also validate our optimization framework on two industrial designs, designated as *TC1* and *TC2*, in a 40$nm$ CMOS foundry technology with HVT-only cells. *TC1* contains a dual-core MCU and six memories. *TC2* contains 15 memories and a number of IP blocks. The worst-case timing and power analysis views for these two industrial designs are respectively (SS, 0.99$V$, -40°$C$) and (TT, 1.1$V$, 25°$C$). We implement these two industrial designs with Cadence tools. The shifter propagation delay for nominal PVT (TT, 1.1$V$, 25°$C$) is 400$ps$. For the non-industrial benchmarks, we generate level shifter models in the 28$nm$ LP technology according to the delay, area and power ratios between the level shifter and the minimum-size inverter in the 40$nm$ technology. Figure 3.10 shows the relation between output current versus the power efficiency of the used voltage regulator. Our optimization flow is implemented in C++. Functions used in P&R tools are implemented in Tcl. We conduct our experiments using a 2.5$GHz$ Intel Xeon server.

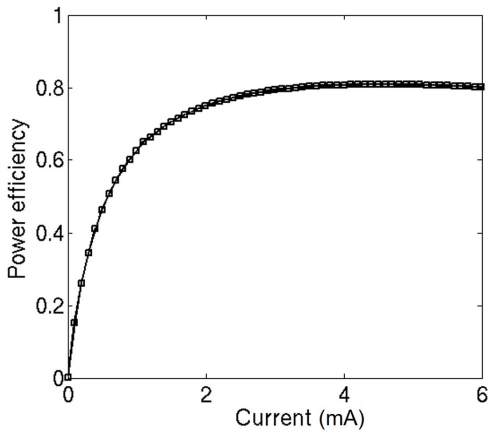



**Figure 3.10**: Power efficiency of switched-capacitor voltage regulator used in [19].

### Comparison to Conventional Designs

Table 3.3 shows the post-CTS comparison between our stacked-domain optimization (*opt*) versus the conventional implementation (*ref*) on four testcases in 28$nm$ LP and two in-

flip-flop) counts of the conventional implementation.

**Table 3.3**: Experimental results. (Power unit: $mW$. Current unit: $mA$. $\eta$ values are estimated based on [19].)

| Design | Flow | WNS | Total cell area | #LS | Func mode | | | | Sleep mode | | | | Runtime |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | | ($ps$) | ($\mu m^2$) | | $P_{core}$ | $I_{bot}/I_{top}$ | $P_{ext}$ | $T$ | $P_{core}$ | $I_{bot}/I_{top}$ | $P_{ext}$ | $T$ | ($min$) |
| *AES* | *ref* | -2 | 8853 | 0 | 9.36 | 9.85/0.00 | 11.68 | 1.00 | 0.08 | 0.09/0.00 | 0.55 | 1.00 | - |
| (28$nm$) | *opt* | -5 | 11035 | 169 | 10.23 | **5.37/5.40** | 10.40 | **1.12** | 0.10 | **0.06/0.05** | 0.16 | **3.34** | 8 |
| *DES* | *ref* | 10 | 16828 | 0 | 17.63 | 18.56/0.00 | 22.01 | 1.00 | 0.07 | 0.07/0.00 | 0.44 | 1.00 | - |
| (28$nm$) | *opt* | 16 | 18428 | 135 | 17.87 | **9.86/8.95** | 18.39 | **1.20** | 0.07 | **0.04/0.03** | 0.09 | **4.97** | 7 |
| *JPEG* | *ref* | 2 | 47507 | 0 | 42.63 | 44.87/0.00 | 53.22 | 1.00 | 0.29 | 0.31/0.00 | 0.72 | 1.00 | - |
| (28$nm$) | *opt* | 5 | 55376 | 687 | 43.36 | **21.32/24.33** | 44.10 | **1.21** | 0.31 | **0.16/0.16** | 0.33 | **2.21** | 14 |
| *VGA* | *ref* | -1 | 98004 | 0 | 64.82 | 68.24/0.00 | 80.92 | 1.00 | 0.36 | 0.38/0.00 | 0.88 | 1.00 | - |
| (28$nm$) | *opt* | 2 | 104087 | 521 | 65.07 | **35.22/33.27** | 65.69 | **1.23** | 0.36 | **0.20/0.18** | 0.50 | **1.75** | 21 |
| *TC1* | *ref* | 9 | 843961 | 0 | 13.76 | 12.51/0.00 | 17.18 | 1.00 | 0.168 | 0.152/0.000 | 0.641 | 1.00 | - |
| | *opt* | 2 | 861128 | 601 | 14.02 | **6.17/6.58** | 14.54 | **1.18** | 0.167 | **0.075/0.077** | 0.175 | **3.66** | 34 |
| *TC2* | *ref* | 102 | 1.00 | 0 | 1.00 | 1.00/0.00 | 1.00 | 1.00 | 1.00 | 1.00/0.00 | 1.00 | 1.00 | - |
| | *opt* | -36 | 1.10 | 914 | 1.05 | **0.63/0.43** | 0.87 | **1.15** | 1.17 | **0.65/0.49** | 0.32 | **3.17** | 113 |

dustrial designs in 40$nm$.[30] Our optimization comprehends both function mode and sleep mode (i.e., with only leakage power). For the industrial design *TC2* we only show normalized metrics – the instance count, area, power ($P_{core}$, $P_{ext}$) and battery lifetime ($T$) values of the optimized designs are normalized to those of the conventional design; the currents ($I_{bot}$, $I_{top}$) of the optimized designs are normalized to total current of the conventional design. We estimate $\eta$ values based on [19]. In 28$nm$ technology, our optimization achieves an average of 19% and 207% battery lifetime improvements in function and sleep modes, respectively. For the industrial designs in 40$nm$, we also achieve more than 10% and 2$\times$ battery lifetime improvements in function and sleep modes, respectively. In other words, we observe similar benefits from the stacked-domain optimization in both 28$nm$ and 40$nm$ technologies. Moreover, the power penalty due to our optimization (see $P_{core}$) is less than 10%, with well-balanced currents (i.e., with $< 10\%$ difference) between the top and bottom power domains (see $I_{bot}$ / $I_{top}$) for most cases. As a result, our optimization significantly reduces $P_{ext}$, and leads to an improved battery lifetime. We also observe that the battery lifetime increase is greater in the sleep mode. This is because most of the current (leakage) goes through the stacked domains, while the regulator needs to provide very little current to maintain the mid node voltage. Hence, there is a high power delivery efficiency despite the voltage regulator having lower efficiency with smaller current (as shown in Figure 3.10). Therefore, stacked-domain optimization is expected to provide more energy and battery lifetime

[30] We use command *ccopt_design* from Innovus to perform clock tree synthesis. We observe that the tool inserts additional clock buffers to compensate the delay of level shifters for a given skew target. Since our optimization minimizes the number of level shifter insertions, we observe from our experimental results that the power penalty from such additional clock buffer insertion is small (e.g., $< 1\%$ clock buffer area penalty on design *TC2*).

benefits if the voltage regulator efficiency is low. We note that all the implementation solutions have negligible timing violations (i.e., #timing violation paths $< 5$), and that the slightly improved worst negative slack (WNS) values of our optimization solutions might be due to P&R tools' noise [95]. Moreover, since logic gates are densely connected in blocks *AES*, *DES*, *JPEG* and *VGA*, and since the block sizes are small, the relative area overheads due to level shifter insertion are large. Runtimes shown in Table 3.3 indicate the extra runtime of our optimization that includes partitioning, re-floorplanning and incremental placement optimization.

### Sensitivity to Level Shifter Delay

We further study the impact of level shifter model on our stacked-domain optimization. We use a pessimistic (i.e., worst-case) model that has roughly 3-4$\times$ power, area and delay compared our current (i.e., nominal-case) model. Figure 3.11 shows that the pessimistic level shifter model leads to slightly larger total design power ($P_{core}$) due to larger level shifter power and timing impact. Since our partitioning optimization minimizes the number of level shifters, the corresponding power penalty due to the pessimistic level shifter model is not large. Results also show larger $\Delta I$ with the pessimistic level shifter model. The larger current difference comes from the level shifters' timing and area impact.

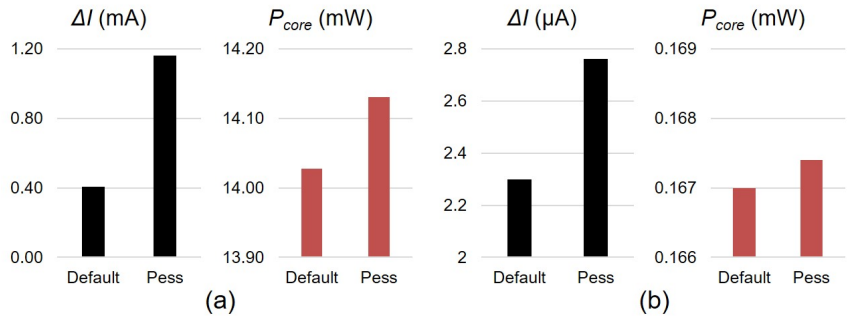



**Figure 3.11**: Impact of level shifter delay, area and power on design QoR in (a) function mode and (b) sleep mode. Design: *TC1*. Technology: $40nm$.

### Sensitivity to Voltage Regulator Power Efficiency

Last, we study impact of voltage regulator efficiency on battery lifetime improvement in stacked-domain designs. More specifically, we vary the $\eta$ value from 40% to 90% with a step size of 10%. For each $\eta$ value, we estimate the battery lifetime improvement from our stacked-domain optimization compared to the conventional design. Figure 3.12 and Figure 3.13 respectively show normalized battery lifetime with respect to that of the conventional design, evaluated

using different voltage regulator efficiencies. Results show that battery lifetime decreases with a higher voltage regulator efficiency. When the regulator efficiency is high, stacked-domain implementation – which has power penalty due to level shifter insertion – can even degrade the battery lifetime of the design (e.g., $\eta = 90\%$ in sleep mode).

We note that the voltage regulator efficiency and area cannot both be optimal in the same power converter [175]. In a typical CMOS process, while highly efficient converters exist, they have poor power density, and the opposite holds for high power density converters, where the efficiency is reduced. Since optimization of both power efficiency and power density cannot be performed beyond technology limitations, an alternate method for further improvement is voltage stacking. Here, the better the matching between the power domains, the less current the regulator has to provide, improving the maximum output current requirements that are related to the area, and also improving the external power supply and, therefore, the power delivery efficiency, in accordance with (3.5).

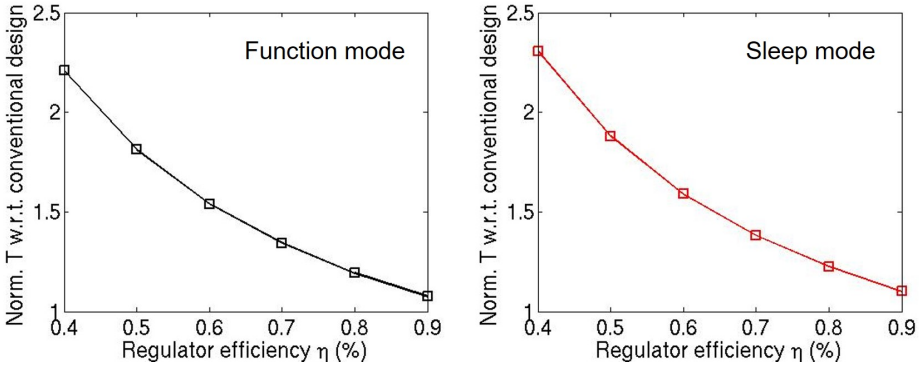



**Figure 3.12**: Impact of voltage regulator efficiency on battery lifetime improvement. Design: *TC1*. Technology: $40nm$.

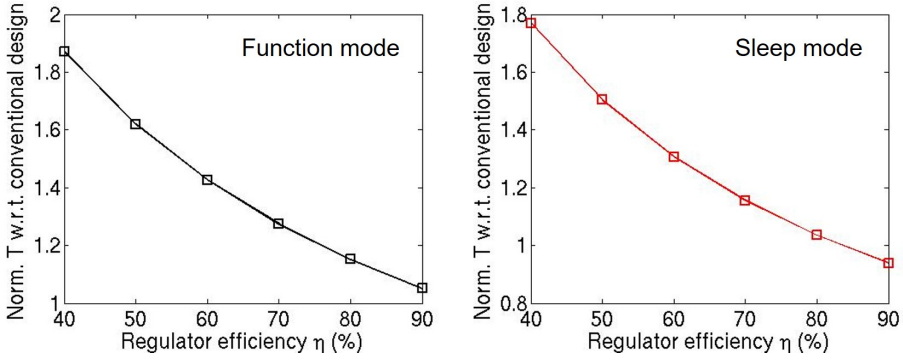



**Figure 3.13**: Impact of voltage regulator efficiency on battery lifetime improvement. Design: *TC2*. Technology: $40nm$.

**Tradeoff between Current Balancing versus Level Shifter Cost**

A relaxed current balancing constraint will lead to a smaller number of level shifters, and hence reduced area and power penalties. In this subsection, we study this tradeoff between current balancing versus the cost of level shifters (e.g., the number of level shifters and resultant $P_{core}$ increase) using the industrial designs *TC1* and *TC2*. Specifically, we vary the instances defined as source and sink as well as the maximum delta current constraint in our flow-based partitioning to achieve different partitioning solutions with different numbers of level shifters.

Table 3.4 shows results for designs *TC1* and *TC2* with different current balancing constraints, normalized with respect to those for the conventional design. In Table 3.4 each column corresponds to one implementation. From left to right, $\Delta I$ increases while the number of level shifter insertions reduces. We evaluate battery lifetime with three $\eta$ values (i.e., 50%, 80% and 95%). For each $\eta$ value the solution with the maximum battery lifetime is shown in bold font. Results show that when $\eta$ is small, solutions with more balanced current offer larger battery lifetime. On the other hand, when $\eta$ is large, solutions with relaxed current balancing constraint and smaller number of level shifters (e.g., *opt'* of *TC2*) provide larger battery lifetime. Results in Table 3.4 also show that since our optimization is able to achieve a balanced-current partitioning solution with a small number of level shifter insertions, for design *TC1* our solution (*opt*) achieves the maximum battery lifetime for various voltage regulator efficiencies.

**Table 3.4**: Results with different current balancing constraints. Designs: *TC1* and *TC2*. $\Delta I$, $P_{core}$ and $T$ are normalized to those of the conventional design.

| **design** | **metric** | *ref* | *opt'* | *opt"* | *opt* |
|---|---|---|---|---|---|
| *TC1* | $\Delta I$ | 1.00 | 0.91 | 0.88 | 0.03 |
| | $P_{core}$ | 1.00 | 1.00 | 1.01 | 1.02 |
| | **#LS** | 0 | 174 | 406 | 601 |
| | $T$ ($\eta$ = 50%) | 1.00 | 1.11 | 1.32 | **1.62** |
| | $T$ ($\eta$ = 80%) | 1.00 | 1.05 | 1.09 | **1.15** |
| | $T$ ($\eta$ = 95%) | 1.00 | 1.01 | 1.00 | **1.03** |
| *TC2* | $\Delta I$ | 1.00 | 0.82 | 0.50 | 0.20 |
| | $P_{core}$ | 1.00 | 1.00 | 1.02 | 1.05 |
| | **#LS** | 0 | 362 | 513 | 914 |
| | $T$ ($\eta$ = 50%) | 1.00 | 1.06 | 1.07 | **1.91** |
| | $T$ ($\eta$ = 80%) | 1.00 | 1.02 | 1.02 | **1.22** |
| | $T$ ($\eta$ = 95%) | 1.00 | **1.02** | 1.01 | 1.01 |

**Block-Aware Partitioning**

Design *TC2* has two usage scenarios – with and without logic block 2. To ensure that the currents between two domains are balanced in both scenarios, we perform *block-aware partitioning* on *TC2*. To achieve this, we first partition the logic block 1 and memories into two domains with balanced currents. We then fix the partitioning solution on logic block 1 and memories, and partition logic block 2 into two domains with the current heuristically balanced for the entire design, and with a minimized number of level shifter insertions. Figure 3.14(a) shows the relative locations of three blocks. Figure 3.14(b) shows the relative locations of top (in red) and bottom (in blue) domains within each block.[31]

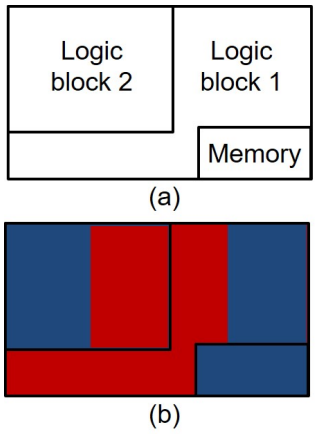



**Figure 3.14**: (a) Approximate layout of *TC2*. (b) Approximate partitioning solution of *TC2* (in red are top domains, in blue are bottom domains).

Figure 3.15 shows the optimized solution of *TC2*, evaluated in the two scenarios of with and without logic block 2 working. The voltage regulator efficiency is estimated based on the power delivery block described in [19]. The solution has negligible timing violations (i.e., the worst negative slack is -60$ps$, with $< 5$ path timing violations) and 1987 level shifters. Results show that by applying the block-aware partitioning, we achieve balanced currents in both working scenarios, and thus improved battery lifetime. The currents are not perfectly balanced because of the high complexity of the industrial design (otherwise, there will be power and timing penalties from the large number of level shifter insertions) as well as the multi-scenario (i.e., function mode and sleep mode) balancing constraints.

[31] We are unfortunately not able to provide additional floorplan and layout details for *TC1* and *TC2*.

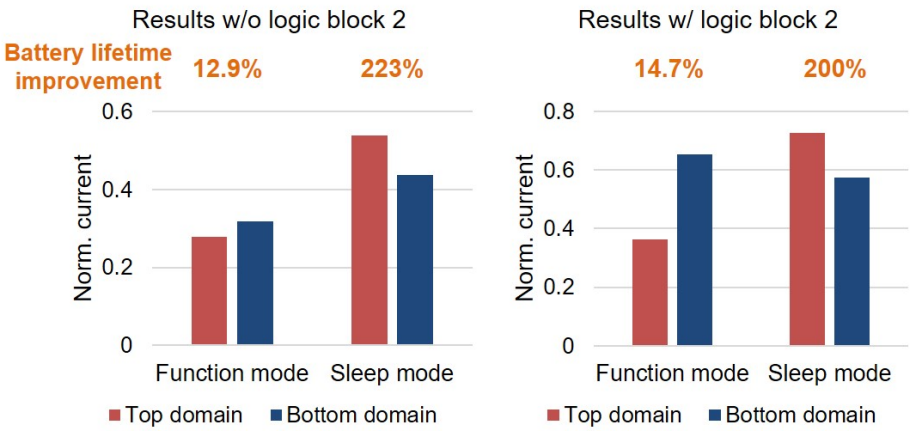



**Figure 3.15**: Block-aware partitioning solution, evaluated in both scenarios (with and without logic block 2). Current values are normalized to the total current of the conventional design including both logic block 1 and logic block 2. Battery lifetime improvements are with respect to the conventional design.

### 3.1.4 Conclusion

In this work, we propose the first comprehensive optimization framework for stacked power-domain implementation with maximized battery lifetime. We extend the existing flow-based partitioning methodology with layout- and timing-path-awareness, as well as multi-scenario balancing objective. We further propose an FM-based grid movement and a dynamic programming-based boundary optimization to define the layout region (power island) of each power domain. Last, we insert level shifter rows in an updated floorplan and place level shifters using a matching optimization. We validate our optimization flow in both $28nm$ LP and $40nm$ technologies, as well as on industrial designs. Our optimization achieves more than $10\%$ and $2\times$ battery lifetime improvements for function and sleep modes compared to the conventional design. Our future works include (i) a predictive methodology to determine the block size prior to trial placement, and (ii) stacked-domain optimization with $> 2$ power domains and/or in 3DICs.

## 3.2 Improved Flop Tray-Based Design Implementation for Power Reduction

Clock network optimization is critical in modern SoC designs due to the following reasons: (i) clock network typically has large power due to its high switching activity; (ii) clock skew and latency (with on-chip variation) have significant impact on design performance; and (iii) clock network routing consumes routing resources and can cause routing congestion. In this work, we study design optimization with *flop trays*[32] (i.e., macro cells of multi-bit flip-flops), where the application of flop trays can significantly reduce the number of sinks in (similar to [9]) and thus result in an improved clock network. Further, careful design of the internal routing within a flop tray prevents hold buffer insertion between flops within the tray, especially along scan chains. This reduces the number of hold buffers, DFT (Design for Test) overheads, and potential placement congestion.

**Flop tray potential benefits.** It is intuitively reasonable that more clock power reduction can be achieved by using larger sizes (i.e., greater number of bits) of flop trays. As a motivating "thought experiment", consider a clock tree with $N$ sinks and fanout of $f$ at each level: the total number of (internal) clock buffers between the clock root and the clock pins of sinks (i.e., flops, flop trays) is $\approx \frac{N-1}{f-1}$. If we could replace all single-bit flops with $K$-bit flop trays, the number of clock buffers would reduce to only $\approx \frac{N/K-1}{f-1}$ (e.g., using 64-bit flop trays to replace single-bit flops could reduce the number of clock buffers by up to 98.4% (= $\frac{N-N/64}{N-1} \approx \frac{63}{64}$)). Furthermore, Figure 3.16 illustrates how inverters for clock signals can be shared among flops in a flop tray, resulting in power and area reduction as compared to multiple single-bit flops. These power and area reductions would also increase with flop tray sizes.

**Current approaches and their limitations.** Flop tray-based implementation is very challenging due to the following reasons. (1) In advanced nodes, flops (including single-bit flops and flop trays) typically occupy a large portion of the entire block area due to their large sizes.[33] Moreover, flop trays can have high aspect ratios (e.g., a 64-bit flop tray may be implemented as a 4 $\times$ 16 array of flops, with much greater width than height); flop tray size and shape have been ignored by previous literature on multi-bit flop optimization [135][140][193] and flop clustering [51][157]. Flop trays with large area and high aspect ratio make placement optimization

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

[33] As an example, a minimum-size inverter occupies two placement sites; a single-bit flop occupies 18 sites; and a 64-bit flop tray can occupy 244 sites in width and four cell rows in height. Due to their large sizes, flops and flop trays can consume a substantial fraction of overall cell area (e.g., *VGA* from *OpenCores* website [230] has 30% of its instances as flops, which accounts for 51% of the total cell area).

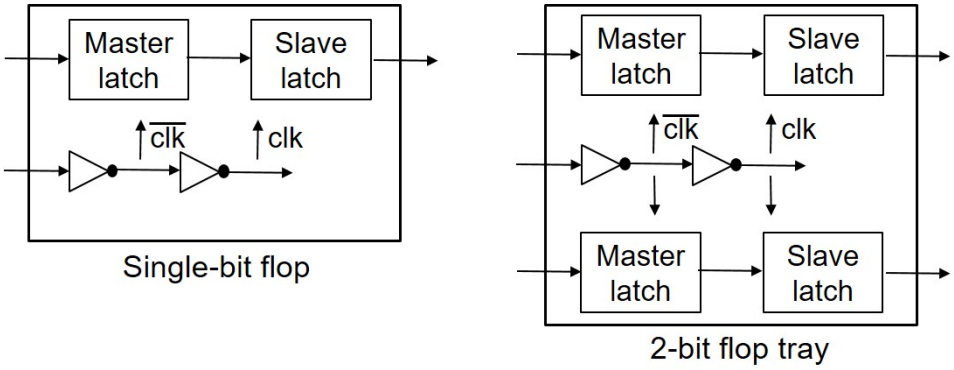



**Figure 3.16**: Two inverters for the clock signal are shared between the two flops in a 2-bit flop tray.

very difficult [54][152]. (2) Clustering of flops imposes additional placement constraints on their fanin and fanout logic cones, which is highly likely to degrade the placement solution quality [152]. (3) Usage of flop trays can easily cause routing congestion. (4) Clustering of single-bit flops into flop trays has large impact on timing and limits the application of useful skew optimization. Most previous works study small-size flop trays, and do not fully address the above challenges in their optimization approaches. Crucially, further achievable benefits of using large-size flop trays are not exploited by previous works. To maximize obtained benefits from flop tray deployment, our present work proposes a flop tray-based optimization that comprehends arbitrary flop tray sizes. (Below, we show results with flop tray size up to 64 bits.)

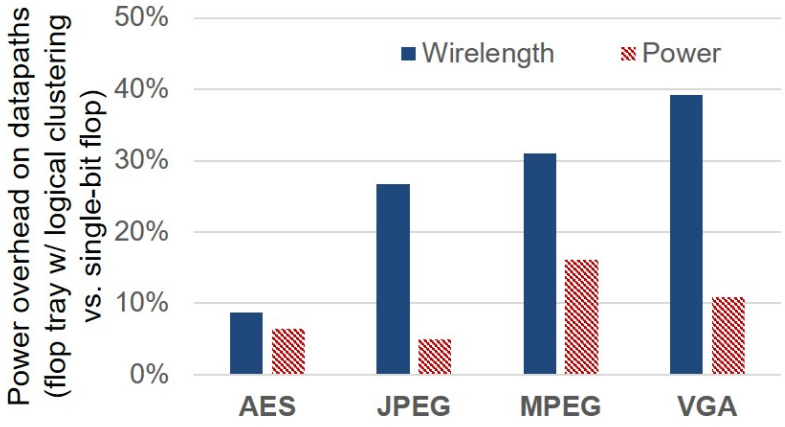



**Figure 3.17**: Wirelength and power overheads on datapaths due to flop tray-based implementations compared to implementations using only single-bit flops. Technology: 28FDSOI. Designs are from *OpenCores* website [230].

A common practice for flop tray-based implementation is to cluster flops during the synthesis stage based on logic functions of the design, along with clock domain and clock gating

information. We refer to this as *logical clustering* in the following discussion. However, flop tray generation without physical information can result in placement and routing congestion and degrade place-and-route (P&R) solution qualities. Figure 3.17 shows examples where flop tray-based implementations with logical clustering during synthesis stage can result in 8% – 39% wirelength overhead and 5% – 16% power overhead on datapaths after detailed routing even at a low conversion ratio from single-bit flops to flop trays. (In the example, numbers of flops and flop trays in flop tray-based implementations, as percentages of flop numbers in implementations with single-bit flops, are 43%, 37%, 41% and 45% for *AES*, *JPEG*, *MPEG* and *VGA*, respectively.) This degrades power benefits from flop tray deployment. Therefore, feedback loops and iterations are required between early-stage flop clustering and P&R optimization, which can significantly increase design time [54]. Furthermore, although splitting large flop trays into smaller trays or single-bit flops during placement and/or routing can mitigate the congestion and power penalty, benefits of applying flop trays then become limited. In addition, the capability of logical clustering to realize flop tray benefits can be limited according to attributes of the given design. Designs with few multi-bit signals may not derive substantial benefits from flop tray deployment. On the other hand, designs with many multi-bit signals might use flop trays aggressively, with large-size flop trays in particular causing placement and routing congestion.

**Our approach.** In this work, we focus on post-placement flop tray optimization.[34] We first place the design with all single-bit flops, where the placement solution is considered to give *ideal* locations of individual flops and combinational cells (given that there are no additional constraints induced by flop clustering). We then cluster flops based on the placement solution. In this way, we resolve the "chicken-and-egg" loop between early-stage flop tray generation and placement optimization of flop trays. However, post-placement flop tray generation such as ours must carefully comprehend different flop tray sizes and aspect ratios; it must also minimize perturbation on datapath placement and timing degradation (otherwise, the assumption of "ideal" combinational cell placement does not hold).

To maximize the benefits of applying flop trays while minimizing the perturbation on the initial placement solution, we propose a *capacitated K-means optimization* which iteratively executes min-cost flow to cluster single-bit flops into flop trays, and a linear programming-based optimization to place flop trays. Based on the proposed capacitated K-means optimization, we achieve a solution (including flop clustering and flop tray placement) for each given flop tray size and AR. We then formulate an integer linear program (ILP) to select the best combination

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

of flop tray solutions. In addition to minimization of displacement of flops (i.e., from the initial single-bit flop location to the flop location in a flop tray), our optimization is also aware of timing-critical start-end flop pairs. Specifically, we minimize the *relative location* displacement of timing-critical start-end pairs to minimize the timing impact from flop tray insertion.

The contributions of this work are as follows.

- We propose a capacitated K-means iterative optimization that applies (i) min-cost flow based clustering, and (ii) LP-based placement optimization) to generate flop trays with various sizes (e.g., 4-bit, 16-bit and 64-bit) at the post-placement stage.
- Our optimization is aware of flop tray aspect ratios and *relative location* displacement of timing-critical start-end pairs.
- We apply a new *Silhouette*-based metric in addition to displacement distance to evaluate flop clustering solutions.
- Our optimization is able to convert more single-bit flops into flop trays, but with smaller datapath power overhead, as compared to a logical clustering flow implemented with commercial tools.
- We achieve up to 32% and 90% reductions of total block power and clock power as compared to implementations using only single-bit flops; and up to 16% and 40% reductions of total block power and clock power as compared to a commercial tool-based flow with logical clustering. We also achieve 13% clock power reduction on average compared to the previous work in [97].
- We evaluate the benefit (i.e., leakage reduction) of useful skew optimization on flop tray-based design and propose a useful skew-aware clustering to maximize such benefit.

The remainder of this section is organized as follows. Section 3.2.1 reviews related works on flop tray optimization. Section 3.2.2 describes our capacitated K-means optimization flow. In Section 3.2.3, we describe our experimental setup and results. Section 3.2.4 concludes and gives directions for ongoing work.

### 3.2.1 Related Work

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

Several early works propose flop tray generation at early design stages. Kretchmer et al. [123] and Chen et al. [33] propose register banking during logic synthesis. They create Liberty models of flop trays, which can be used by logic synthesis tools. But, flop tray generation during synthesis has only logic topology as its main lever, and the lack of physical information can result in a sub-optimal clustering solution, degraded timing and larger power. To address this, Hou et al. [87] further propose register banking removal based on routing congestion and timing information. However, such a "(flop) clustering at early stage and (flop tray) removal at late stage" flow is not able to effectively exploit the benefits of flop tray usage. Thus, many other works propose flop tray generation during and/or after placement.

Yan et al. [204] generate flop trays at the post-placement stage. They first construct an intersection graph based on routing length and congestion constraints derived from an initial placement solution with single-bit flops. They then perform minimum-clique partitioning to reduce the number of flop trays. Lin et al. [134] use progressive window-based optimization to improve the methodology proposed in [204] considering given flop tray sizes. They solve the clustering problem by finding $K$-cliques and maximum independent sets in a merging graph constructed based on feasible-location regions of flops. Similarly, Wang et al. [193] use clique partitioning to identify a set of non-conflicting cliques. Jiang et al. [97] propose an efficient post-placement flop tray generation technique using interval graphs and a pair of linearized sequences. Liu et al. [140] also propose flop clustering based on an intersection graph. In addition to reducing the number of flop trays, they apply agglomerative clustering to minimize displacements of flops, wirelength and clock power. More recently, Lin et al. [135] develop a clock tree-aware in-placement flop tray generation technique. They build an intersection graph considering clock latency, wirelength and timing, then iteratively perform flop tray generation and timing-driven incremental placement. Xu et al. [203] propose an analytical clustering score for flop tray generation, permitting seamless integration with the traditional wirelength objective. Tsai et al. [183] propose to generate flop trays during placement. During analytical global placement, they guide placement of flops (to enable flop tray generation) with additional bonding force (resembling ionic bonds in chemistry). Other works optimize flop trays with awareness of crosstalk [88], clock gating [143], etc.

In addition to flop tray-based design, flop and/or latch clustering optimizations have been widely applied in previous works for clock tree and latch placement optimization. Mehta et al. [149] propose a clustering algorithm to obtain approximately load-balanced clusters and construct clock trees so as to minimize skew. Papa et al. [157] apply K-means clustering algo-

rithm to minimize latch displacement during a physical synthesis optimization. Deng et al. [51] propose a register clustering methodology in generating the leaf-level topology of the clock tree to reduce clock power consumption.

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

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

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

- Previous works use local search to cluster flops into flop trays. However, due to capacity constraints of flop trays, such local search can result in outliers with large displacement distances. By contrast, in this work we apply a more globally-aware optimization based on (i) a capacitated K-means formulation (with iterative min-cost flow-based clustering and LP-based placement optimization), and (ii) a practically scalable ILP-based matching and selection of flop tray solutions to globally optimize flop clustering with given capacity constraints (i.e., flop tray sizes).[35]

### 3.2.2 Methodology

We now describe our optimization methodology for flop tray generation and placement. Figure 3.18 illustrates our overall optimization flow, where we integrate our flop tray optimization (steps in blue boxes) into a conventional SP&R (synthesis, place, and route) flow. To address the "chicken-and-egg" loop between flop tray generation and placement optimization, we first

[35] Our ILP runtime (CPLEX 12.6) is less than one minute on the *VGA* testcase [230] (with 17K flops and 1000 timing-critical paths) with five candidate flop tray sizes studied in Section 3.2.2 and Section 3.2.3 below, using 20 threads on a $2.5GHz$ Intel Xeon server.

perform an initial placement with only single-bit flops, where the placement is considered to be "optimal" with no placement constraints induced by flop clustering. We note that since the initial placement is timing- and congestion-aware, minimizing subsequent perturbations can mitigate potential congestion due to flop trays, as well as minimize timing impacts. Further, to comprehend multiple flop tray sizes and ARs, we perform flop tray optimization for each flop tray choice (i.e., a {size, AR} combination). Last, we perform an integer linear programming (ILP)-based optimization to select the optimal combination of flop trays and their placement solutions.[36]

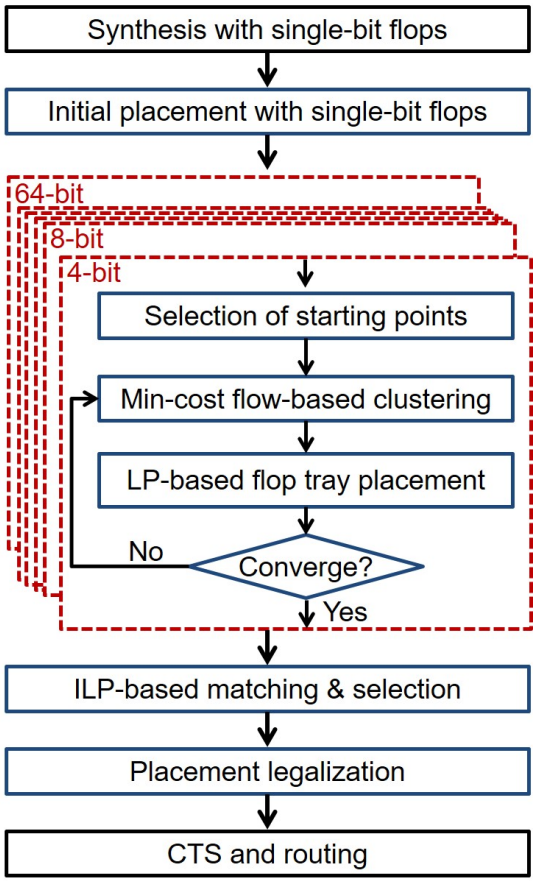



**Figure 3.18**: Overall optimization flow of flop tray generation.

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

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

The following subsections describe our capacitated K-means clustering and our ILP-based selection of flop tray solutions. Table 3.5 lists the notations used in our discussion.

**Table 3.5**: Description of notations used in our formulation.

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

### Capacitated K-Means Clustering

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

To address this problem, we propose a capacitated K-means algorithm [120]. (As noted above, K-means clustering algorithms have also been applied to flop (or latch) clustering in previous works [51][157].) There are two steps in a standard K-means algorithm: (i) clustering, and (ii) updating the center location of each cluster. We associate these two steps with: (i) matching of single-bit flops to flop slots in flop-trays, and (ii) updating the locations of flop trays. We propose a min-cost flow to address (i), and a linear programming (LP)-based optimization to address (ii). We iterate between these two steps until convergence (i.e., no further displacement reduction can be achieved, or a maximum number of iterations (= 35 in our experiments below) is reached).

In our capacitated K-means clustering, we use an algorithm that is similar to K-means++ [10] to select the starting points. Selection of $\lceil N/K \rceil$ starting points for clustering is described in Algorithm 11. In Algorithm 11 we calculate center-to-center distances between single-bit flops. To comprehend the aspect ratio of flop trays, we scale the horizontal distance by (1/AR) (= height/width) of the given flop tray.

**Algorithm 11** Selection of starting points.

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

These selected starting points serve as initial locations of flop trays. We then apply a min-cost flow to achieve *capacitated clustering* of flops. Our min-cost flow is illustrated in Figure 3.19. To construct the flow instance, we create a node for each single-bit flop $h_l$. For each flop tray $t_i$, we further create $K$ nodes for its $K$ slots, $f_{i1} \ldots f_{iK}$. For each edge between a pair of $h_l$ and $f_{ij}$, we set its capacity as 1 and its cost as the Manhattan distance between $h_l$ and $f_{ij}$. Here, we directly calculate the Manhattan distance between single-bit flops and flop slots without any scaling. Finally, we create one source and one sink, and assign edges connected to them with capacity as 1 and cost as 0, as illustrated in Figure 3.19. Notice that by considering the distances between the locations of single-bit flops and flop slots in flop trays, our min-cost flow optimization is explicitly aware of physical information (in particular, dimensions and ARs) of the given flop trays.

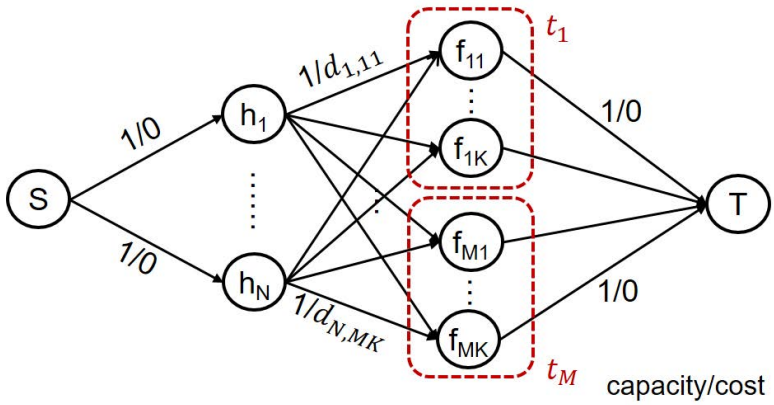



**Figure 3.19**: Example of min-cost flow with K-bit flop trays.

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

$$\text{Minimize} \quad D \tag{3.8}$$

$$\text{Subject to } |X_i + x'_{ij} - x_l| + |Y_i + y'_{ij} - y_l| = d_l \quad \forall h_l \tag{3.9}$$

$$\sum_l d_l = D \tag{3.10}$$

Constraint (3.9) calculates the displacement for each flop ($d_l$), and the objective seeks to minimize the total displacement over all flops.

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

To confirm benefits from awareness of flop tray ARs, we show in Figure 3.20 representative clustering solutions from (i) the classic K-means approach, which treats each flop tray as a point, and (ii) our min-cost flow-based clustering, which is aware of flop tray ARs. We observe that our clustering solution more closely matches the AR of given flop trays. Further, classic K-means without awareness of flop tray AR can result in 2× increase in average displacement from the "ideal" single-bit flop placement; this is likelier to incur datapath power and timing overheads.

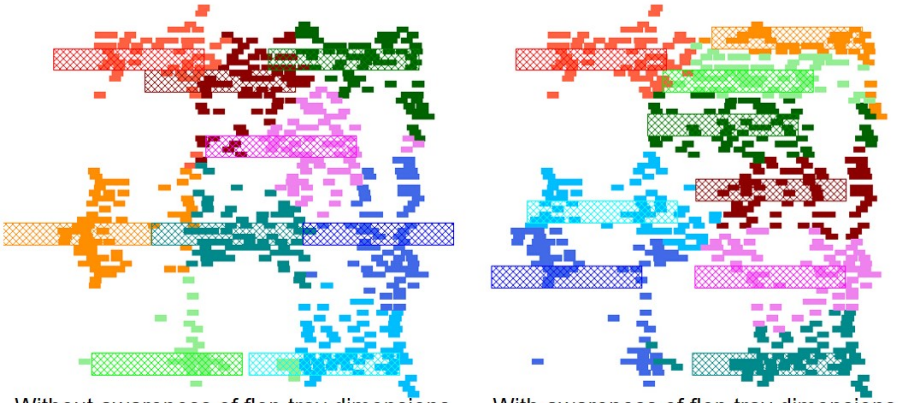



**Figure 3.20**: Clustering solutions into 64-bit flop trays (i) without awareness of flop tray aspect ratio and dimensions, and (ii) with awareness of flop tray aspect ratio and dimensions. Design: *AES* (530 single-bit flops). Technology: 28FDSOI.

In our capacitated K-means algorithm, as with K-means approaches in general, the selection of starting points has a strong impact on the final solution quality. We adapt the Silhou-

ette metric [168] and use Equation (3.11) to evaluate the solution quality of generated starting points.[37]

$$func(h_l) = \frac{min_{i' \neq i,j'}(d_{l,i'j'}) - d_{l,ij}}{max(d_{l,ij},\ min_{i' \neq i,j'}(d_{l,i'j}))} \tag{3.11}$$

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

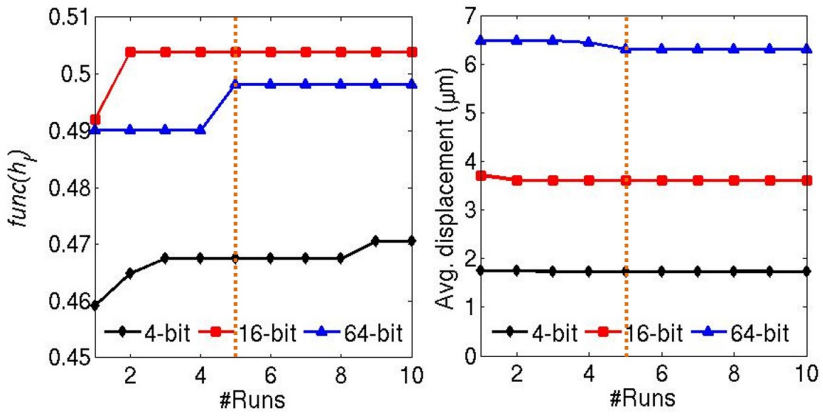



**Figure 3.21**: Best clustering solution (i.e., $func(h_l)$ (left) and displacement (right)) with multiple runs (numbers of runs are shown in the x-axis).

We apply a multistart strategy to improve the selection of starting points. Multiple runs (five in our experiments) of the procedure in Algorithm 11 are each followed by a small number (15 in our experiments) of iterations between the min-cost flow and LP-based placement optimization. We then select the solution with the highest average $func(h_l)$ value and proceed with capacitated K-means iterations until convergence. Figure 3.21 shows a typical improvement of the average value of $func(h_l)$ (left) and the average displacement (right) with increased number of runs. In our studies, the improvement of $func(h_l)$ and displacement typically saturates after five runs. Thus, the experiments reported below apply five multistarts to mitigate the impact of starting point selection.

[37] As presented in [168], the Silhouette value is a measure of how similar an object is to its own cluster, compared to other clusters. A general Silhouette value is defined as $s(i) = \frac{b(i)-a(i)}{max(a(i),b(i))}$, where $a(i)$ is the average dissimilarity (e.g., average distance) of $i$ with all other data within the same cluster, and $b(i)$ is the lowest average dissimilarity (e.g., minimum average distance) of $i$ to the data in any other cluster other than its own. By definition, $-1 \leq s(i) \leq 1$, and a larger Silhouette value indicates a better clustering solution. In this work, data are slots of flop trays, and dissimilarities are measured by distances.

**ILP-Based Matching Optimization**

The next step of our optimization approach addresses the following problem: **Given** candidate flop trays with various capacities, each with a fixed placement location, **select** the optimal subset of the candidate flop trays, and **determine** a mapping of single-bit flops into slots of selected candidate flop trays, **such that** (i) every single-bit flop is mapped to a slot of a selected flop tray (including flop trays with one bit, i.e., no clustering), and (ii) a weighted sum of the total displacement of flops, relative displacement of timing-critical start-end pairs, and total flop tray costs is minimized.

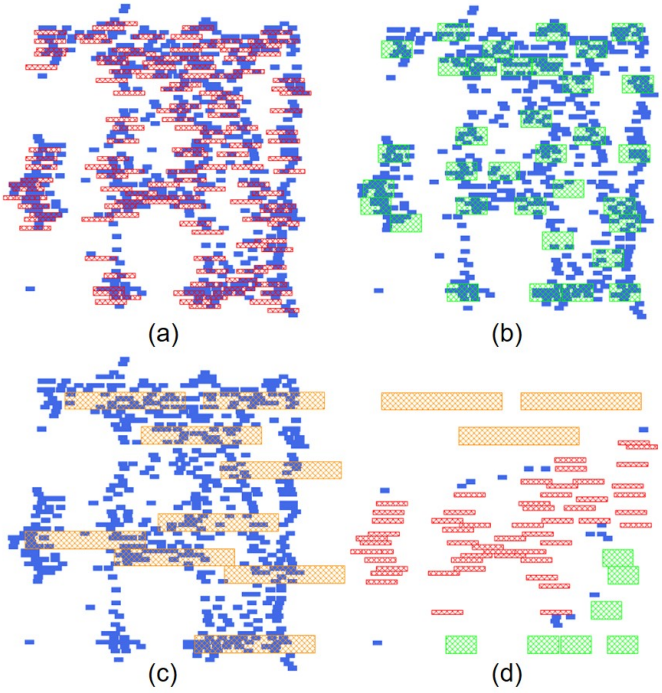



**Figure 3.22**: Example of our ILP-based optimization.

As discussed in Section 3.2.2, we run capacitated K-means clustering with different flop tray sizes and ARs, and use these flop trays together with their optimized placement locations as inputs ("candidates") for an ILP-based matching optimization. Our ILP-based optimization selects an optimal subset of candidate flop trays with various flop tray sizes as our final solution. As an example, Figures 3.22(a)-(c) show solutions of flop trays with fixed sizes and ARs on the *AES* testcase. Specifically, Figures 3.22(a)-(c) respectively show solutions with only 4-bit flop trays (flop trays are in red, #flop trays = 133, average displacement = $2\mu m$), only 16-bit flop trays (flop trays are in green, #flop trays = 34, average displacement = $3\mu m$), and only 64-bit flop trays (flop trays are in orange, #flop trays = 9, average displacement = $5\mu m$). Figure 3.22(d)

shows the final solution, i.e., solution with a combination of single-bit flops and 4-bit, 16-bit and 64-bit flop trays (#flops + #flop trays = 81, average displacement = $2\mu m$). Our objective is to minimize a weighted sum of total displacement of flops, relative displacement of timing-critical start-end flop pairs, and total flop tray cost. Relative displacement of a timing-critical start-end flop pair is illustrated in Figure 3.23. As an improvement to previous approaches, we comprehend timing impact of flop tray generation considering timing-critical paths (i.e., start-end pairs). Specifically, if the flop tray generation moves two flops towards each other, combinational cells in the logic cone between the flops are forced to be placed in a more compact region, which results in congestion and distortion of the placement and routing. Alternatively, if the flop tray generation moves two flops away from each other, timing paths between the two flops will tend to have longer wirelength, degrading timing. We therefore seek to minimize the *relative displacement* of flops that are timing-critical start-end pairs.

Our ILP to select the optimal combination of flop tray solutions with various sizes and ARs is given below.[38]

[38]Note that our ILP can be extended to be aware of clock gating, clock domain and useful skew optimization, etc. with additional constraints. Section 3.2.3 briefly describes a useful skew-aware extension and corresponding benefits.

$$\text{Minimize } \alpha \cdot W + D + \beta \cdot Z \tag{3.12}$$

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

$$+|\sum_{ij}(Y_i + y'_{ij} - y_l) \cdot b_{l,ij}| = d_l \quad \forall l \tag{3.13}$$

$$\sum_{l} d_l = D \tag{3.14}$$

$$d_l \leq d_{max} \quad \forall l \tag{3.15}$$

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

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

$$= z_{ll'} \qquad \forall (h_l,\ h_{l'}) \in \text{timing-critical paths} \tag{3.16}$$

$$\sum_{(h_l,h_{l'}) \in cri\_paths} z_{ll'} = Z \tag{3.17}$$

$$z_{ll'} \leq d_{max} \quad \forall (h_l,\ h_{l'}) \in \text{timing-critical paths} \tag{3.18}$$

$$b_{l,ij} \leq e_i \qquad \forall l,\ j \tag{3.19}$$

$$e_i \leq \sum_{lj} b_{l,ij} \quad \forall i \tag{3.20}$$

$$\sum_{i} w_i \cdot e_i = W \tag{3.21}$$

$$\sum_{l} b_{l,ij} \leq 1 \quad \forall j \tag{3.22}$$

$$\sum_{i,j} b_{l,ij} = 1 \quad \forall i \tag{3.23}$$

Here, $W$ is the total cost of selected flop trays, which is determined based on their power consumption and sizes (i.e., number of bits); $D$ is the total displacement over all flops; $Z$ is the total relative displacement over all timing-critical start-end flop pairs; and $\alpha$ and $\beta$ are weighting parameters. Constraints (3.13) and (3.14) calculate the total displacement of all flops. Constraint (3.15) bounds the maximum displacement of each flop. Constraints (3.16) and (3.17) calculate the total relative displacement of timing-critical start-end flop pairs (i.e., $(h_l,\ h_{l'})$). Constraint (3.18) bounds the maximum relative displacement of each timing-critical start-end flop pair. Constraints (3.19) and (3.20) force the binary indicator variable $e_i$ to be 1 if the corresponding flop tray is used, and 0 otherwise. Constraint (3.21) calculates the total cost of selected flop

trays. Constraints (3.22) and (3.23) ensure that each flop is matched to exactly one slot, and that each slot is matched to at most one flop. We note that additional mutual exclusion constraints can avoid placement overlaps between pairs of flop trays (e.g., $e_i + e_j \leq 1$ if there is overlap between the $i^{th}$ and $j^{th}$ flop trays). However, such mutual exclusion constraints might limit the solution space and thus degrade the solution quality. We therefore perform placement legalization in the commercial P&R tool to remove overlaps among flop trays.[39] We also note that although an ILP-based optimization typically has large runtime, in our formulation, the number of binary variables is only $O(N \cdot Q)$, where $N$ is the number of flops and $Q$ is the number of candidate flop tray choices (i.e., sizes and dimensions). In practice, our method exhibits practically reasonable runtimes (see Footnote 4 above).

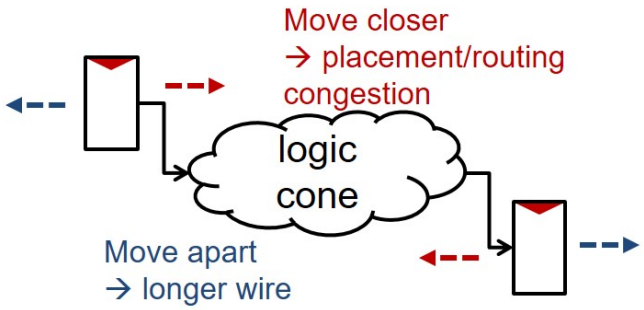



**Figure 3.23**: Illustration of the timing impact due to relative displacement between timing-critical start-end flop pairs.

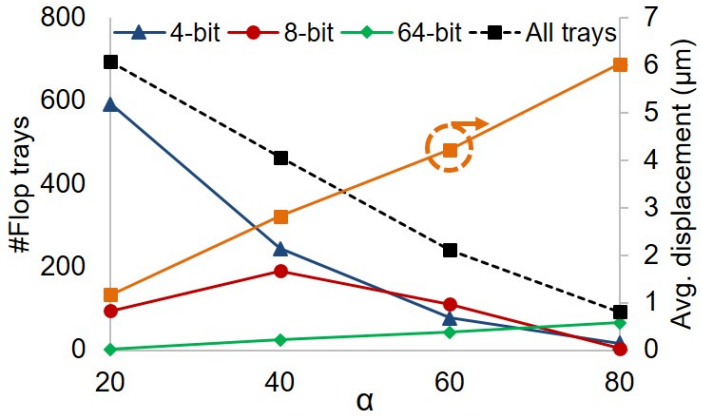



**Figure 3.24**: Number of flop trays and average displacement of flops change with different $\alpha$ values. Design: *JPEG*. Technology: 28FDSOI.

To give an understanding of how the weighting parameters $\alpha$ and $\beta$ affect solution quality, Figure 3.24 shows the number of flop trays and the average flop displacement resulting from

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

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

To evaluate the impact of $\beta$, we uniformly place flop trays within the block area and fix their locations. The number of flop trays is determined according to the number of flops, such that no flop tray can be empty, which eliminates the impact of $W$ in our objective function. We then perform an ILP-based matching optimization to cluster flops into flop trays. Figure 3.25 shows the total block power of the *AES* and *JPEG* testcases implemented with various $\beta$ values. We observe reduced block power with $\beta > 0$, where our optimization minimizes the relative displacement between timing-critical start-end flop pairs. This confirms the benefits of minimizing the relative displacement between timing-critical start-end flop pairs. We also observe increased block power with a large $\beta$ value. This is because with a large $\beta$ value, relative displacements between timing-critical start-end flop pairs dominate our objective function. The resultant large displacements of non-timing critical flops incur datapath power penalty. We empirically use $\beta$ = 1 in our experiments.

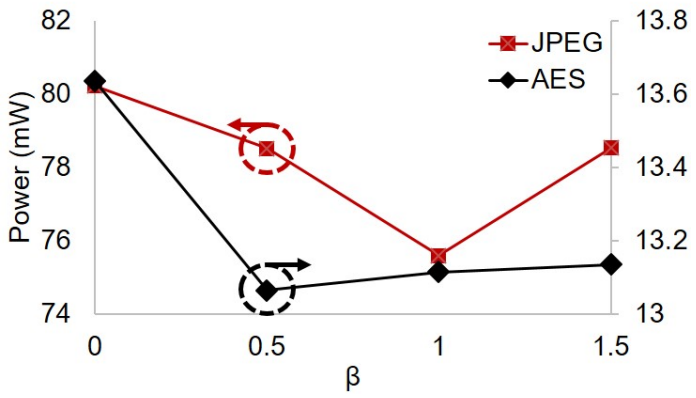



**Figure 3.25**: Power change with various $\beta$ values. Designs: *AES*, *JPEG*. Technology: 28FDSOI.

### 3.2.3 Experimental Results

We perform experiments in a $28nm$ FDSOI foundry technology with dual-$V_{th}$ libraries. We use four design blocks (*AES*, *JPEG*, *MPEG*, *VGA*) from *OpenCores* website [230] as our testcases. Parameters of these four testcases are shown in Table 3.6. We scale flop tray power and area based on the ratios shown in Table 3.7. Layout ARs of flop trays are also shown in Table 3.7. We synthesize designs using *Synopsys Design Compiler vI-2013.12-SP3* [237] and then place and route using *Cadence Innovus Implementation System v15.2* [215]. We set the placement density at the floorplan stage as 70%. We also perform timing and power analyses using *Cadence Innovus Implementation System v15.2*. We perform vectorless power simulation with a default switching activity of 10% at primary inputs. Our optimization flow is implemented in C++. We use *CPLEX v12.6* [223] as our ILP solver and LEMON [226] as our min-cost flow solver. Functions used in P&R tools are implemented in Tcl. We conduct our experiments on a 2.5$GHz$ Intel Xeon server.

**Table 3.6**: Testcase parameters.

| **Design** | **#Instances** | **#Flops** | **Clock period** |
|---|---|---|---|
| *AES* | ∼12K | 530 | 600$ps$ |
| *JPEG* | ∼47K | 4512 | 600$ps$ |
| *MPEG* | ∼13K | 3181 | 500$ps$ |
| *VGA* | ∼56K | 17053 | 700$ps$ |

**Table 3.7**: Normalized flop tray area and power, and layout AR.

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

#### Comparison to Logical Clustering

To evaluate the performance of our proposed methodology, we compare our solutions to three reference flows: (i) the conventional implementation flow with only single-bit flops (*ref_1b*), (ii) a flop tray-based implementation flow which generates flop trays during commercial synthesis based on logical clustering, followed by conventional commercial P&R optimization (*ref_mb1*), and (iii) a flop tray-based implementation flow which generates flop trays at the post-

placement stage using the method proposed in [97], followed by clock tree synthesis and routing (*ref_mb2*). No value judgment or "benchmarking" regarding any commercial tool is intended by, or should be inferred from, our present discussion.

Table 3.8 shows results evaluated at the post-routing stage. Figure 3.26 shows the layouts of placement solutions with single-bit flops and optimized flop trays. We observe that our proposed optimization (*opt_mb*) is able to significantly reduce the number of sinks with application of flop trays (e.g., we reduce the number of sinks by 98% on the *VGA* testcase compared to the implementation using only single-bit flops). The reduction in number of sinks results in smaller clock power: our optimization reduces clock power by up to 90% and 40% compared to implementations with single-bit flops and flop trays generated by logical clustering, respectively. Our flop tray generation also results in reduced power on flops. Moreover, we observe that although our optimization has large conversion ratio from single-bit flops to flop trays, the incurred datapath power and wirelength penalties are small as compared to the implementation with logical clustering. This strongly suggests that our approach of optimization with minimum perturbation from a "good" initial placement solution forestalls placement and routing congestion while also minimizing the datapath power penalty from application of flop trays. For the *MPEG* testcase, our optimization actually results in smaller datapath power as compared to the "ideal" implementation with single-bit flops; we believe this is likely due to reduced placement density (i.e., usage of flop trays reduces total area of flops).

Our optimization (*opt_mb*) also achieves up to 7% total block power reduction compared to the previous work [97] (*ref_mb2*). Since *ref_mb2* only uses up to 8-bit flop trays, we limit the flop tray options to 4-bit and 8-bit flop trays in *opt_mb'* for a fair comparison. Table 3.8 shows that with the same set of flop tray options our optimization achieves 13% clock power reduction on average compared to *opt_mb'*, along with smaller datapath power for most of the testcases (the exception is the *JPEG* testcase with $< 1\%$ power overhead).

**Optimization with Various Flop Tray Sizes**

We further perform flop tray optimization with various combinations of flop tray sizes. More specifically, we implement designs with (i) single-bit flops only, (ii) {4-bit} flop trays, (iii) {4-bit, 8-bit} flop trays, (iv) {4-bit, 8-bit, 16-bit} flop trays, and (v) {4-bit, 8-bit, 16-bit, 32-bit, 64-bit} flop trays with various $\alpha$ values (i.e., 20, 40, 60, 80). We note that setups (ii)-(v) can also use single-bit flops. For each setup, we select the minimum total block power solution with $< 5\%$ power penalty on datapaths as compared to the case with only single-bit flops. Figure 3.27

**Table 3.8**: Experimental results.

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

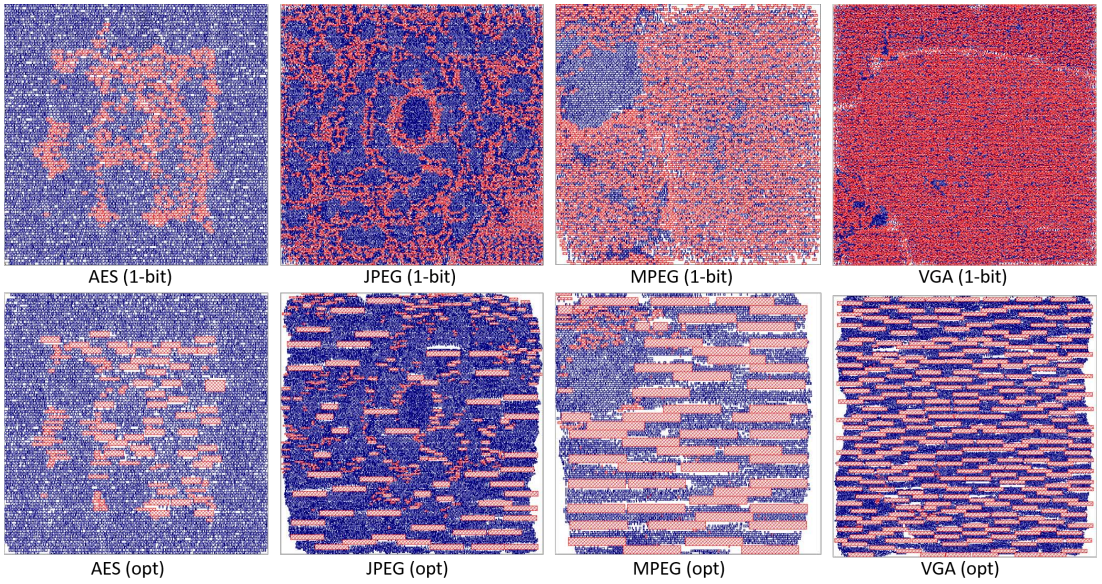



**Figure 3.26**: Layout comparison between implementations with only single-bit flops and with optimized flop trays. In the flop tray-based solutions, the candidate flop tray sizes are 4-bit, 8-bit, 16-bit, 32-bit and 64-bit.

shows flop power and clock power, normalized to implementations using only single-bit flops. We observe that with only 4-bit flop trays, our optimization achieves $> 7\%$ power reduction on flops and flop trays. However, including larger flop trays does not afford much further reduction

of flop power. (This may be due to our conservative assumptions regarding power-per-bit in larger flop trays, as shown in Table 3.7). On the other hand, application of large-size flop trays can effectively reduce clock power. For example, optimizations with {16-bit, 32-bit, 64-bit} flop trays achieve 11% more clock power reduction on average as compared to the cases with only {4-bit, 8-bit} flop trays.

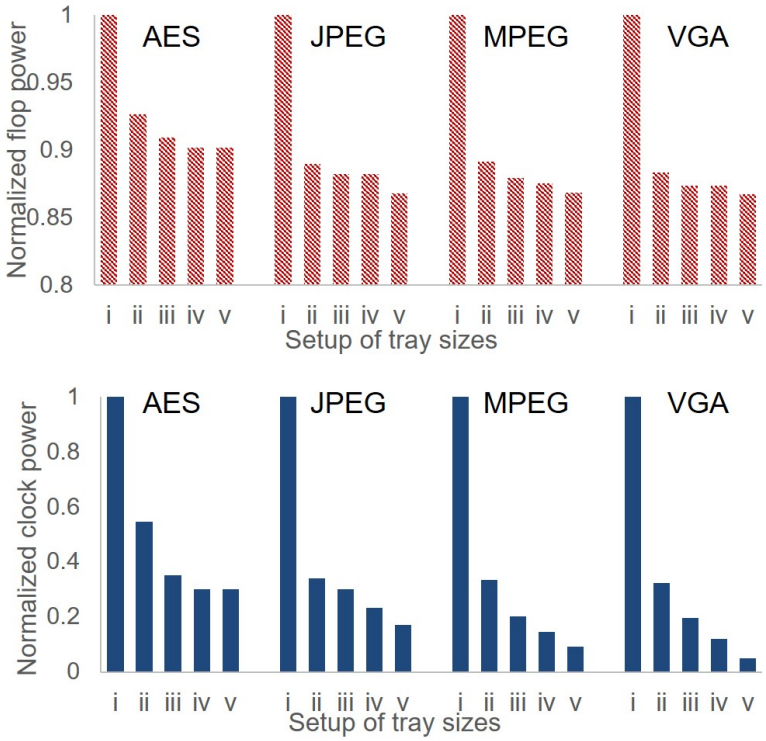


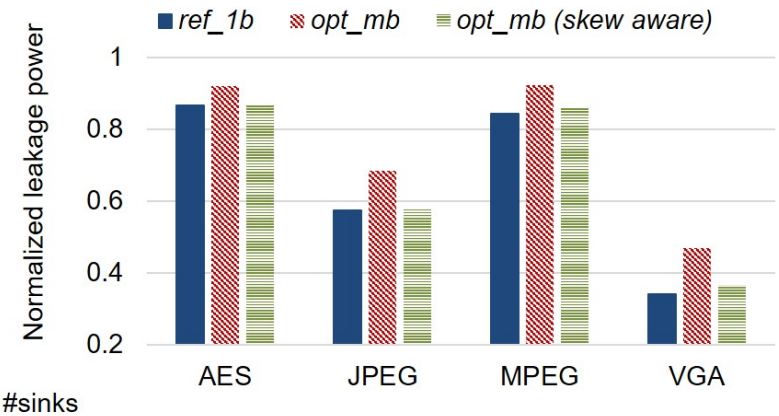

**Figure 3.27**: Flop (tray) power and clock power of designs with various flop tray sizes. Candidate tray sizes are 4-bit, 8-bit, 16-bit, 32-bit and 64-bit.

**Study of Useful Skew Optimization with Flop Trays**

Last, we evaluate the benefits of useful skew optimization in terms of leakage power reduction on (i) designs with only single-bit flops (*ref_1b*), and (ii) flop tray-based designs (*opt_mb* as shown in Figure 3.28.[40] Based on the approach proposed in [4], we formulate the useful skew optimization as a maximum mean weight cycle problem and apply iterative shortest path search to maximize the average endpoint slack. We then perform leakage power optimization using a commercial tool [215], i.e., we exploit the increased timing slacks for leakage power reduction. We observe from Figure 3.28 that due to clustering of endpoints, flop tray-based de-

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

signs have 9% less leakage power reduction on average across four designs as compared to cases with only single-bit flops. To reduce the impact of flop tray generation on benefits from useful skew optimization, we study skew-aware flop tray generation that only allows clustering of flops with desired skew less than $\theta$ (we use $\theta = 20ps$ in our experiments). Figure 3.28 shows that the skew-aware clustering (*opt_mb (skew aware)*) can achieve similar leakage power reduction as compared to the cases with only single-bit flops (green vs. blue bars), but at the cost of more sinks (i.e., an average of 21% less reduction in number of sinks).




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

**Figure 3.28**: Datapath leakage power results, normalized to implementations with only single-bit flops.

### 3.2.4 Conclusion

In this work, we present a novel flop tray-based optimization for improved design power reduction. We propose a capacitated K-means algorithm which iteratively applies a min-cost flow-based clustering and a LP-based flop tray placement. We also propose an ILP-based matching optimization to generate flop trays while minimizing the perturbation to the initial placement solution. Our work achieves several improvements as compared to previous works: (i) awareness of flop tray aspect ratio and (large) size; (ii) explicit minimization of relative displacement of timing-critical start-end flop pairs; and (iii) global optimization instead of local search. The proposed techniques allow us to achieve up to 32% total block power reduction as compared to designs with only single-bit flops, and up to 16% total block power reduction over designs with flop trays generated by logical clustering during synthesis. We also achieve 13% clock power

reduction on average compared to the previous work in [97]. We further study the impact of flop tray sizes on optimization solution quality. Finally, we study useful skew optimization in the context of our flop tray-based designs. Our future works include (i) scalable optimization considering all flop tray candidate sizes simultaneously; (ii) awareness of IR-drop in the flop tray clustering and placement; and (iii) floorplan blockage-aware and routing congestion-aware flop tray generation.

## 3.3 An Improved Methodology for Resilient Design Implementation

IC products in advanced technology nodes are susceptible to dynamic variations that manifest via supply voltage droop, temperature fluctuation, cross-coupling, aging, and other mechanisms. To ensure correct functionality and robustness, traditional IC implementation methodologies build guardband into clock frequencies and design signoffs – notably, timing signoff at slow corners and for hold-time correctness. However, it is well-recognized that designing for worst-case conditions incurs considerable power and performance overheads. *Better Than Worst-Case design* [12], where an error checker and corresponding recovery mechanism enable typical-case optimization, can significantly reduce overdesign compared to traditional methodologies. A similar idea for guardband reduction has been proposed by Bowman et al. in [23], where several techniques for dynamic variation tolerance (i.e., resilient designs) are presented.

Resilient designs, as discussed in this work, use variant register (i.e., timing endpoint) circuit designs to trade off design robustness against design quality (performance, power and area); ideally, they can ensure correctness against variation and improve signoff performance [40][50][61][74][77][177]. *Razor* [61] is a well-known technique to detect and correct timing errors. Razor detects timing violations by supplementing error-tolerant flip-flops with *shadow latches*. A shadow latch strobes the output of a logic stage at a fixed delay after the main flip-flop; if a timing violation occurs, the main flip-flop and shadow latch will have different values, signaling the need for correction. Correction involves recovery using the correct value(s) stored in the shadow latch(es), or via instruction rollback/replay. In the following discussion, we define the maximum timing violation that a resilient design can tolerate as the *safety margin* of the corresponding design.

By allowing timing errors, resilient designs are used to improve performance. An example is *timing speculation* [190], which increases the clock frequency and exploits error detection and recovery mechanisms to correct resulting errors. Timing improvement from resilient designs can lead to further power and area benefits over conventional designs. In other words, due to relaxed timing constraints, we can reduce the power and area of logic cells in the fanin cone of an endpoint at which an error-tolerant register has been instantiated.

A practical methodology for deployment of resilient designs must overcome a number of significant overheads of resilience. Notably, resilient designs require additional circuits or cycles to detect and correct timing errors. Figure 3.29 shows the structure of Razor, Razor-Lite [122] and TIMBER [40] flip-flops. All have additional circuits, and hence power and area overheads,

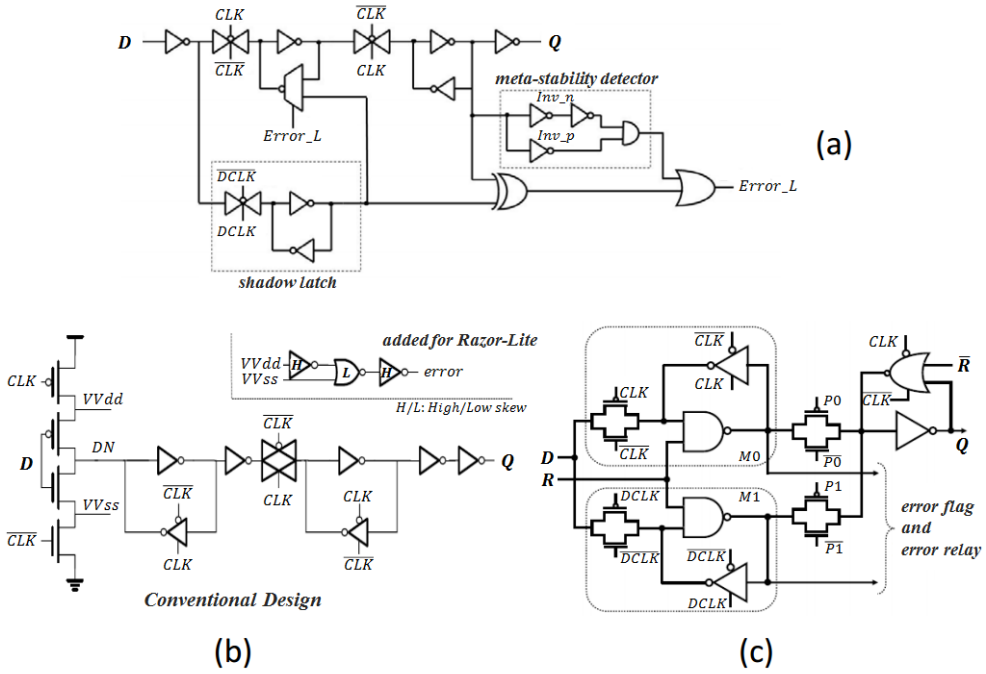



**Figure 3.29**: Structure of (a) Razor, (b) Razor-Lite, and (c) TIMBER flip-flops.

compared to a conventional flip-flop. For instance, Razor has its shadow latch and other error-tolerant circuits (comparator, multiplexer and OR-gate). When compared to a conventional flip-flop, the total power overhead of a Razor flip-flop is 30% [50]. Although the power overhead has been significantly reduced for each error-tolerant register in a recent work [122], the cost incurred by the error-detection network is still large.

We use the term *pure-resilient design* to indicate a design that uses only error-tolerant registers (instead of conventional ones). Our background studies indicate that in a pure-resilient design, the error-detection network alone can consume up to 9% of the total wirelength. Furthermore, the additional cycles needed to recover from errors can lead to performance (throughput) degradation. Moreover, error-tolerant circuits are vulnerable to hold violations. Designers must ensure that benefits (in terms of performance, and/or area and power reduction from the error resilience) outweigh the additional costs of error-tolerant circuits.

A crucial open question is, "What is the minimum achievable cost of (a given amount of) resilience?" As a step toward answering this question, in this work we perform in-depth studies of the tradeoff between the overhead of error-tolerant circuits and the cost of the traditional timing optimizations, with the goal of assessing the 'true' benefits of resilient design techniques. To our knowledge, no previous work has conducted such studies to explore the 'true' benefits of resilience comprehending all types of resilience cost and various types of resilient designs.

We propose two effective design optimization techniques – *selective-endpoint optimization*, and *clock skew (useful skew) optimization* – to minimize the costs of resilience, i.e., (i) power and area overheads of resilient circuits, and (ii) throughput degradation due to additional cycles for error recovery. Based on the optimization, we develop a complete implementation flow (i.e., from placement to signoff) for resilient designs. Our flow comprehends the impacts of signoff corners and process variation, ensuring timing correctness of conventional flip-flops at the slow corner while optimizing design energy at the typical corner. Since our selective-endpoint optimization reduces the number of error-tolerant flip-flops and the applied clock skew optimization is hold-timing aware, we do not specifically optimize on short-path padding, and small hold penalty is observed. However, our optimization flow can easily be combined with existing short-path padding optimizations (e.g., [205]). Our contributions include the following.

- We propose an optimization methodology to reduce the cost of resilience. Our methodology exploits both error-tolerant registers and clock skew scheduling.
- We study the benefits and cost of resilient design implementations, where we trade off among (i) power and area overheads of error-tolerant registers, (ii) optimization of logic cells in the fanin and fanout cones, and (iii) throughput degradation due to timing errors.
- We propose an implementation flow to construct the error-detection network, which is required in an actual implementation. We exploit geometric placement information of the error-tolerant registers to substantially mitigate wirelength overhead of the error-detection network.
- We perform typical-corner optimization of resilient designs to maximize energy reduction. At the same time, our implementations comprehend both multiple signoff corners and process variation.
- By reducing the number of error-tolerant registers, our optimization minimizes the cost of short-path padding.
- We assess the opportunities and costs of resilient implementations across different error-tolerant register designs as well as in the adaptive voltage scaling (AVS) context.

The rest of this section is organized as follows. Section 3.3.1 presents related works. Section 3.3.2 formulates the problem of minimizing the cost of resilience and describes our methodology for implementing low-cost resilience. Section 3.3.3 presents our experimental results and analyses, and Section 3.3.4 concludes the section.

### 3.3.1 Related Work

A number of resilient design techniques have been proposed that allow timing errors, in conjunction with different error detection and correction mechanisms. These previous works can be roughly classified into two categories. In the first category, designs use replica circuits for error masking. These designs typically incur large power and area overheads due to its additional circuits. In the second category, designs use error-tolerant registers to detect timing errors. Although circuit power and area overheads can be smaller, instruction rollback or replay is required to recover from timing errors. The additional cycles for error recovery lead to throughput degradation.

**Replica Circuits for Error Masking.** A well-known technique compares output values in each cycle using redundant hardware circuits. *DIVA* [11] applies a functional checker to verify the correctness of the core processor's computation, only permitting correct results to commit. *Paceline* [77] employs a *leader-checker* which checks timing errors due to overclocking. *CPipe* [177] enables reliable overclocking through core-replication. The outputs of the main combinational logic are compared with those of the duplicated logic in each cycle. Choudhury et al. [41] synthesize error-masking circuits and use 2-to-1 multiplexers to mask errors at the output of critical paths. Similarly, Yuan et al. [210] mask errors by adding redundant approximation logic which has higher speed than the original circuit. *TIMBER* flip-flops and latches [40] enable online timing error masking via time-borrowing from the succeeding pipeline stage. This kind of approach provides error resilience with high reliability, but also incurs significant power and area overheads due to the redundant logic circuits.

**Error-Tolerant Registers with Error Recovery.** *Razor* and related works [13][50][61][122] replace registers with specialized flip-flops which detect timing errors by capturing the correct value at shadow latches with a delayed clock. *Razor* [61] can correct timing errors within a specific safety margin of the error-tolerant register. *Razor II* [50] provides analysis of the Razor flip-flop – with respect to timing constraints, safety margin and clocking scheme – and reduces complexity and area of the Razor flip-flop. *Bubble Razor* [69], which uses two-phase latch timing, stalls the pipeline based on the propagation of error clock gating control signals ("bubbles") to mitigate timing errors. A Markov chain-based analysis for a ring of Bubble Razors is provided in [211]. A more recent work – *Razor-Lite* [122] – further reduces the area and power penalties of error-tolerant registers. *STEM* [13] improves the capability of error-detection with a second shadow latch. [22] introduces two error-tolerant circuits – transition detector with time-borrowing, and double sampling with time-borrowing.

**Resilient Design Optimization.** With the above error-tolerant registers, various design-level optimization techniques [41][78][105][106][142][141][190][210] have been proposed which identify and optimize critical paths that are frequently exercised during operation. These works apply resilient techniques to timing-critical and/or frequently-exercised paths, but typically fail to holistically consider the 'true' benefits and costs of the error-tolerant circuits during the optimization. In other words, the tradeoff between the cost of resilience and the costs of margin insertion for data paths (Figure 3.31) is ignored. Further, none of these works consider all types of costs in a resilient design (i.e., power and area of resilient circuits and data paths and throughput degradation) simultaneously. For instance, [41] and [210] optimize area and power of resilient circuits, but not the costs of data paths; the optimizations in [105][141][190] consider power of data paths and throughput degradation, but not the overhead of resilient registers; [142] minimizes the number of error-tolerant registers and the cost of short path padding, but without regard to the overhead of throughput degradation. In addition, optimizations in [41][142][141][190][210] occur at the synthesis stage. However, the timing-critical paths can vary after placement and routing (P&R), and this discrepancy can degrade the solution quality. Our present work is differentiated by performing optimization during P&R stage that comprehends the tradeoff between the cost of resilience and the costs of margin insertion, as well as by simultaneously considering (more comprehensively) the costs in a resilient design.

**Clock Skew Optimization.**

Of particular note are clock skew optimizations that have been proposed by previous works on enhancement of design robustness and timing speculation. An early work [67] formulates a linear program to maximize the minimum timing slack in a design via clock skew scheduling, which improves the tolerance of the design to variations. [198] adjusts clock latencies to minimize the probability of timing errors being latched by overlapping separate error-latching windows. Their optimization also prevents errors from propagating along pipeline stages. [34] and [207] propose online clock skew tuning methods to minimize timing errors during runtime using tunable delay buffers and clock tuning elements (i.e., circuits with multiple skew configurations), respectively. However, due to implementation complexity, fine-grained optimization is practically impossible. Further, clock tuning logic can introduce extra area and power costs. [208] proposes clock skew scheduling optimization to minimize the error rate in a resilient design. Based on error probability at each endpoint, they determine skew values using a gradient-descent method. The work uses Razor flip-flops for timing-critical endpoints and ignores the tradeoff between cost of resilience and data path optimization. The optimization also ignores

hold constraints, which are critical in a design with resilient registers, as well as the potential power implications (e.g., for data paths) of the skew scheduling. Our present work proposes clock skew optimization that maximizes both setup and hold slacks at all timing-violated paths with comprehension of toggle rate information. Further, in our work the improved timing slacks are exploited to enable removal of error-tolerant registers and power reduction on data paths.

### 3.3.2 Methodology

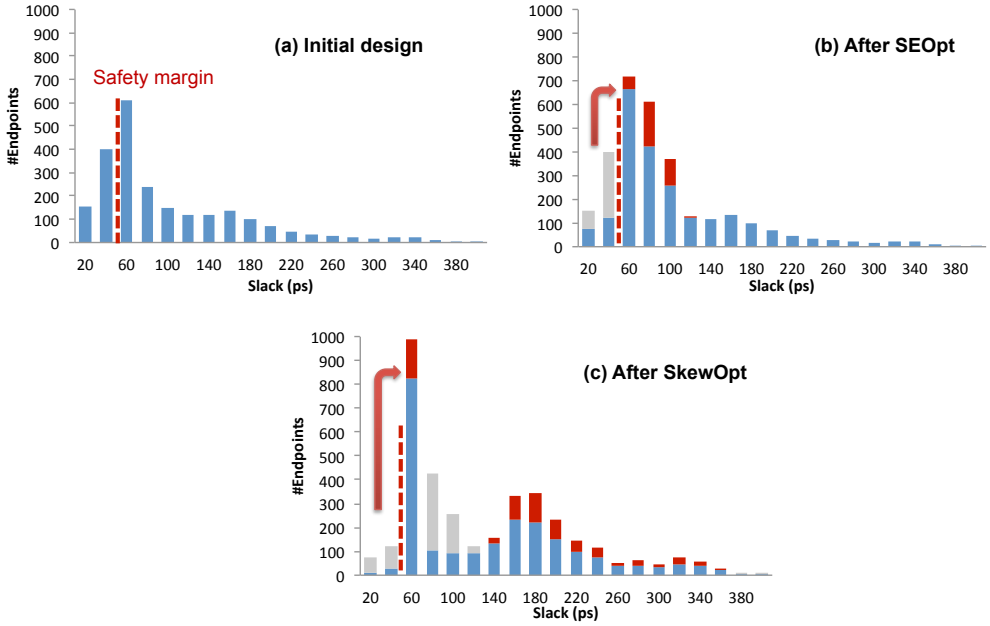



**Figure 3.30**: Slack distribution of endpoints in (a) original design; (b) design with only selective-endpoint optimization; and (c) design with combined selective-endpoint and useful skew optimization. Red dotted lines indicate required safety margin. Design: *FPU*. Technology: 28$nm$ FDSOI.

In this section, we define a resilience cost reduction problem and describe our optimization flow for low-cost resilient design implementation. Our flow uses two optimization techniques – *selective-endpoint optimization* (SEOpt) and *clock skew optimization* (SkewOpt) – to minimize resilience overheads of energy, area and throughput degradation. Figure 3.30 illustrates the basic idea of our optimization approach. In the initial resilient design (a), a large number of endpoints have timing violations at the target frequency (with respect to the safety margin), and error-tolerant registers or error-masking circuits are used for those endpoints. In our selective-endpoint optimization (b), we tightly optimize a set of selected endpoints to reduce the resilience overheads. During clock skew optimization (c), we increase timing slacks of endpoints having timing violations by optimizing the clock-arrival time at individual endpoints,

further reducing the resilience overheads. In our optimization flow, we iteratively perform SEOpt and SkewOpt to minimize the cost of resilient design. We will show in Section 3.3.3 that our proposed optimization achieves significant improvement in terms of area and energy as compared to previous works – (i) conventional resilient design implementation, and/or (ii) useful skew optimization on resilient designs.

**Resilience Cost Reduction Problem**

We solve the following **resilience cost reduction problem**. Given an RTL design along with (i) throughput requirements, (ii) power and area overheads as well as safety margin for each type of error-tolerant register, and (iii) number of cycles needed to recover from an error: implement the design to attain minimum energy, comprehending the energy penalties of additional circuits and the throughput degradation due to instruction rollback or replay.

We calculate design energy based on total power and throughput information, i.e.,

$$\text{Energy} = \frac{\text{Power}}{\text{Throughput}} \tag{3.24}$$

We further estimate the throughput based on error rate information as

$$\text{Throughput} = \frac{1 - \text{ErrorRate}}{T} + \frac{\text{ErrorRate}}{\theta \times T} \tag{3.25}$$

where $T$ is the clock period, and $\theta$ is the number of cycles needed to recover from an error. For an accurate design, the throughput is $1/T$.

We further estimate the error rate based on toggle information of flip-flops (including toggles of both negative-slack and positive-slack fanin paths) as

$$\text{ErrorRate} = \alpha \times \frac{\sum (h_{ff} \times \frac{\sum h_{p\_neg}}{\sum h_{p\_all}})}{\sum h_{ff}} \tag{3.26}$$

where $h_{ff}$ is the toggle rate of a flip-flop, $h_{p\_neg}$ and $h_{p\_all}$ are respectively the toggle rates of negative-slack fanin paths and all fanin paths to the flip-flop, and $\alpha$ is a parameter to compensate pessimism due to (i) the fact that multiple errors can occur in one cycle and (ii) the existence of false paths. We empirically use $\alpha = 0.35$ in our experiments.

**Selective-Endpoint Optimization**

We now describe *selective-endpoint optimization* (SEOpt) for reduction of resilience cost (primarily area, power and throughput degradation). SEOpt trades off between the costs of resilience and of data path optimization. We note that ours is the first work to consider such

tradeoff in resilient design optimization. As illustrated in Figure 3.31(a), increasing timing margin at an endpoint allows replacement of the error-tolerant register with a conventional one and removal of replica circuits. However, these margins incur area and power costs in combinational logic cones. Figure 3.31(b) shows the example of the OpenSPARC T1 *FPU*: as we reduce the number of Razor flip-flops from 300 to zero, resilience cost decreases while power of the non-resilient part increases. This results in an observed unimodal behavior of the design energy change. In this work, we seek the subset of endpoints for margin insertion that, when optimized (replaced with conventional registers) in this way, leads to minimum design energy. Two key questions are (i) '*which endpoints should be optimized?*', and (ii) '*how many endpoints should be optimized?*'.

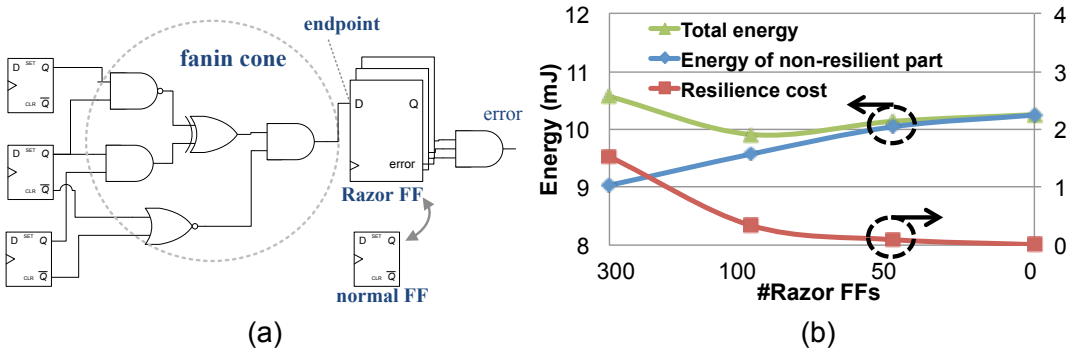



**Figure 3.31**: (a) Illustration of the tradeoff between cost of resilience and cost of data path optimization. (b) With reduced number of Razor flip-flops, resilience cost decreases but power of data paths increases. Design: *FPU* (OpenSPARC T1). Technology: $28nm$ FDSOI.

For Question (i), area and power of combinational cells in the fanin cone of an endpoint will increase when we add slack margin for the endpoint. Further, each endpoint will exhibit a different cost versus margin relationship. Therefore, to reduce the optimization cost, we should preferentially optimize endpoints which are less sensitive to slack margin insertion. In SEOpt, we evaluate sensitivity functions for endpoints to estimate the potential optimization cost and guide the selection of endpoints for optimization. That is, a given sensitivity function of an endpoint reflects the available performance versus power and/or area tradeoff of the corresponding fanin cone. We observe that the optimization cost increases significantly for an endpoint which (i) is timing-critical, (ii) has a large number of timing-critical fanin cells (i.e., negative-slack cells in the fanin cone of the endpoint), and (iii) has a fanin cone with large power (e.g., due to high toggle rate). We therefore use slack at endpoint $slack(p)$, number of timing-critical fanin cells $num_{cri}(p)$, and power of timing-critical fanin cells $power(c)$ to evaluate the sensitivity of each endpoint. We study sensitivity functions for a given timing endpoint $p$ with different

combinations of these parameters, and the following five empirically show good results:

$$SF1(p) = |slack(p)| \tag{3.27}$$

where we consider timing critically at the endpoint;

$$SF2(p) = |slack(p)| \times num_{cri}(p) \tag{3.28}$$

where we consider slack at the endpoint and the number of negative-slack cells in the fanin cone;

$$SF3(p) = |slack(p)| \times \frac{num_{cri}(p)}{num_{total}(p)} \tag{3.29}$$

where we consider slack at the endpoint and portion of negative-slack cells over all cells in the fanin cone;

$$SF4(p) = |slack(p)| \times \sum_{c \in fanin(p)} power(c) \tag{3.30}$$

where we consider slack at the endpoint and power of timing-critical fanin cells; and

$$SF5(p) = \sum_{c \in fanin(p)} (|slack(c)| \times power(c)) \tag{3.31}$$

where we consider the products of slack and power of timing-critical fanin cells.

To study the performance of each sensitivity function, we sort the endpoints in increasing order of their estimated sensitivities, based on the given sensitivity function. We then optimize the top $\eta\%$ endpoints of the sorted list, where we increase $\eta$ from 0 to 100 with a step size of 5. Figure 3.32 shows power and area resulting from selective-endpoint optimizations based on the five sensitivity functions on *FPU* in a foundry $28nm$ FDSOI technology. In this example, the safety margin is 10% of the clock period.[41] In all experiments, we assume a switching activity of 0.2 at primary inputs and propagate activities using a commercial P&R tool. We then dump out a *switching activity interchange format* (SAIF) file and use it for power analysis and error rate estimation. Note that our optimization framework can be extended to vector-based scenarios, where we generate SAIF file from the *value change dump* (VCD) file derived from gate-level simulation. We observe in Figure 3.32 that for a given number of endpoints to optimize, SEOpt based on SF2 and SF5 incurs smaller power and area penalties. We use SF5 in the experiments reported in Section 3.3.3.[42]

[41] In [40], safety margins of 10%, 20% and 30% of clock period are studied.

[42] Although the best sensitivity function might vary in a different technology/library or with a different implementation tool, one can apply the same evaluation method given any specific design enablement, i.e., to pick the most successful sensitivity function from among various options based on the selected parameters.

For Question (ii), optimizing more endpoints reduces the number of error-tolerant registers required. However, the cost of this optimization (i.e., area and power penalty on data paths) also increases. We iteratively increase the number of endpoints to be optimized and select the solution with minimum cost (e.g., a function of area and/or power).

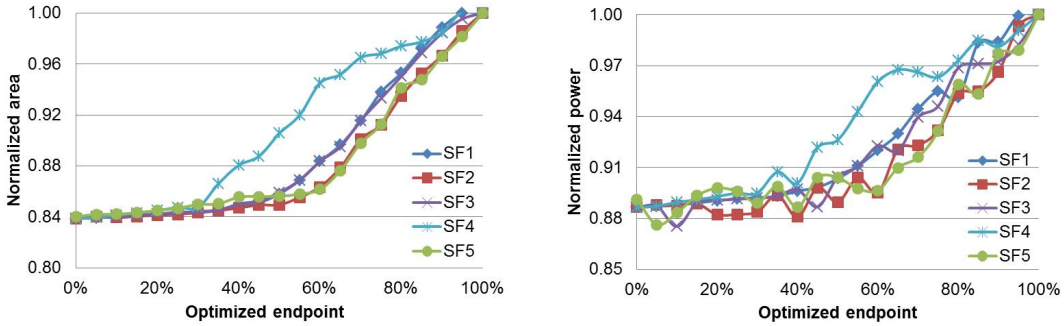



**Figure 3.32**: Cell area and total power resulting from selective-endpoint optimization with different sensitivity functions. Design: *FPU* (OpenSPARC T1). Technology: $28nm$ FDSOI.

### Clock Skew Optimization

To further reduce the number of error-tolerant registers and minimize timing errors, we apply *clock skew optimization* (SkewOpt), which maximizes slacks at timing-violated endpoints. In SkewOpt, we formulate the clock skew optimization problem as a *maximum mean weight cycle* problem [4]. This is because the maximum achievable timing slack of a given path is determined by the maximum average slack of a cycle (i.e., a loop formed by timing paths) which contains that path. We use the *parametric shortest path* algorithm [209] to determine the maximum mean weight cycle. The algorithm as we have implemented it is described in Algorithm 12. We first construct a graph $G$ where each endpoint corresponds to a vertex and each timing path corresponds to two edges (i.e., one for setup and one for hold ) (Line 1). The weights of edges indicate setup/hold slacks of timing paths in the corresponding FF-to-FF (flop-to-flop) logic cones.

We optimize setup timing slacks of endpoints with error-tolerant registers (with respect to hold constraints and setup constraints on other paths). We classify edges in the graph into two categories – (i) *parameterized edges* and (ii) *non-parameterized edges* – where timing corresponding to parameterized edges will be optimized, while non-parameterized edges will serve as constraints during the optimization. We define parameterized edges based on setup constraints on paths having timing violations with respect to the safety margin, and non-parameterized edges based on hold/setup constraints on the remaining paths. We formulate the constraints in SkewOpt as

$$l_q + \underbrace{(T - d_q - d_{p,q}^{max} - t_q^{setup} - t_{p,q}^{margin})}_{s_{p,q}} - \lambda \geq l_p \ (q \in R) \tag{3.32}$$

$$l_q + \underbrace{(T - d_q - d_{p,q}^{max} - t_q^{setup} - t_{p,q}^{margin})}_{s_{p,q}} \geq l_p \ (q \notin R) \tag{3.33}$$

$$l_p + \underbrace{(d_p - d_{p,q}^{min} - t_q^{hold} - d_q)}_{s_{p,q}} \geq l_q \ \ (\forall q) \tag{3.34}$$

where $T$ is the clock period; $l_p$ is the clock arrival time of endpoint $p$; $d_p$ is the clock-to-Q delay of $p$; $d_{p,q}^{max}$ and $d_{p,q}^{min}$ are, respectively, the maximum and minimum path delay from $p$ to $q$; $t_q^{setup}$ and $t_q^{hold}$ are the setup and hold times of $q$; and $t_{p,q}^{margin}$ is the required safety margin between $p$ and $q$. $R$ is the set of endpoints which use error-tolerant registers, and $\lambda$ is the *parameter* which will indicate the slack change. Constraint (3.32) corresponds to a parameterized edge in the constructed graph with an edge weight of $(s_{p,q} - \lambda)$. Constraints (3.33) and (3.34) are respectively induced by setup and hold constraints on a given non-parameterized edge in the constructed graph with an edge weight of $s_{p,q}$.

In the graph $G(V, E)$, we always maintain a tree $(V, E_T)$ to store edges corresponding to timing-critical paths. We initialize the tree by inserting a dummy vertex (i.e., root $r$) and dummy edges connecting $r$ and other vertices (Lines 3-4). In Line 5, $p\_w(p)$ is the total weight along the shortest path (i.e., path with the minimum total weight) from $r$ to $p$ in $G$. Then, we iteratively add edges corresponding to the most timing-critical paths to the tree (Lines 7-15) while removing any dummy edges that have the same head vertex as an added edge (Line 17). When adding an edge to the tree results in a cycle[43], we coalesce the cycle (including vertices and edges on the cycle) into one vertex (Lines 18-24). The edges on the cycle are added to the solution graph and the optimized slacks are stored. To each parameterized edge, a weight is assigned equal to the summation of weights (i.e., slacks) on the cycle divided by the number of parameterized edges on the cycle. We assign zero slack to the non-parameterized edges on the cycle. That is, timing paths with conventional registers as endpoints will have zero slack with respect to the safety margin if they are in a maximum mean weight cycle that contains critical paths with error-tolerant registers as endpoints. Note that assigning new weights indicates a change of clock arrival times. Therefore, we update the weights of edges incident to vertices on the cycle. We then optimize slacks on the updated graph. We iteratively determine and optimize the most critical maximum mean weight cycle until there is only one edge in the graph (i.e., no

[43] Since we always add the edge corresponding to the most timing-critical path to the tree, the resulting cycle is the most critical maximum mean weight (i.e., slack) cycle.

**Algorithm 12** Clock skew optimization (SkewOpt)

**Procedure** $SkewOpt(N)$

1: $G(V,\ E) \leftarrow$ construct graph corresponding to $N$ // $N$ is the input netlist
2: Initialize solution graph $G'(V,\ \emptyset)$
3: $V \leftarrow \{r\} \cup V$; $E \leftarrow \{(r,p)\} \cup E,\ \forall p \neq r$; $w(r,p) \leftarrow 0,\ \forall p \neq r$
4: $E_T \leftarrow \{(r,p)\},\ \forall p \neq r$
5: Update $p\_w(p),\ \forall p \in V$ // $p\_w(p) = \sum w(p_i,p_j),\ \forall (p_i,p_j) \in$ shortest path (SP) from $r$ to $p$
6: **while** $|E| > 1$ **do**
7: $\lambda_{min} \leftarrow +\infty$
8: **for all** $(p,q) \in E$ for which $(p,q) \notin E_T$ **do**
9: $\lambda_{p,q} \leftarrow$ Solve $p\_w(p) + w(r,q) = p\_w(q)$
10: **if** $\lambda_{p,q} < \lambda_{min}$ **then**
11: $\lambda_{min} \leftarrow \lambda_{p,q}$
12: $e_{min} \leftarrow (p,q)$
13: **end if**
14: **end for**
15: $E_T \leftarrow E_T \cup \{(p,q)\}$
16: $\lambda \leftarrow \lambda_{min}$
17: Remove all edges from $E_T$ that have the same head vertex as $e_{min}$
18: **if** there is a cycle in $E_T$ **then**
19: $slack(p,q) \leftarrow \lambda_{min},\ \forall (p,q) \in cycle$
20: Add all edges on cycle to $G'$
21: $E \leftarrow E \setminus \{(p,q) \mid (p,q) \in cycle\}$
22: Contract all vertices on cycle into $p_{new}$
23: Update $E$ and $E_T$
24: **end if**
25: **end while**
26: Traverse $G'$ to calculate $l_q$ based on $slack(p,q)$ and $l_p$
27: $N_{sol} \leftarrow$ apply $l_p,\ \forall p$ to $N$
28: **return** $N_{sol}$

more cycles can be found). Last, we traverse the solution graph and calculate the clock arrival times based on the optimized path slacks (Line 26).

To further enable error-rate awareness and reduce the cost of throughput degradation, we extract toggle rate information of each timing path and replace Constraint (3.32) by

$$l_q + \frac{s_{p,q}}{1 + \beta \times h(p,q)} - \lambda \geq l_p \ (q \in R) \tag{3.35}$$

where $h(p,q)$ indicates the toggle rate of the maximum-delay path between endpoints $p$ and $q$, and $\beta$ is a weighting factor (we use $\beta = 2$ in our experiments).

### Proposed Optimization Flow

As mentioned in Section 3.3.2, SEOpt reduces the cost of resilience via optimizations on data paths. However, such optimization incurs power and area overheads. By contrast, Ske-

wOpt migrates timing slacks from timing non-critical paths to timing-critical paths, which does not incur power and area penalty. But, SkewOpt cannot generate additional slacks, hence its performance highly depends on the topology of the sequential graph. For example, SkewOpt might not perform well when there are many cycles consisting of timing-critical paths. In this work, we combine the SEOpt and SkewOpt methods and execute them iteratively. The basic idea is that we use SEOpt to create timing slacks on data paths with low power penalty. We then apply SkewOpt for an improved distribution of timing slacks. In this way, we reduce the number of error-tolerant registers and minimize error rates of a resilient design without incurring large power and area penalties.

**Algorithm 13** Combined optimization (CombOpt)

**Procedure** $CombOpt(N)$
1: Run $STA$ to initialize slack values for the netlist $N$
2: $P \leftarrow \emptyset$
3: **for all** timing endpoints $p$ in the netlist $N$ **do**
4:  **if** $slack(p) <$ safety margin **then**
5:  Compute sensitivity value for endpoint $p$
6:  $P \leftarrow P \cup \{p\}$
7:  **end if**
8: **end for**
9: $m \leftarrow |P|/k$  // $m$ indicates the number of iterations
10: $C_{min} \leftarrow \infty$
11: **for** $i = 0 ; i < m ; i \leftarrow i + 1$ **do**
12:  Pick the top $k$ endpoints $P_i$ with minimum $sensitivity$ in $P$
13:  $N_i \leftarrow TimingOpt(N_{i-1}, P_i)$
14:  $N_i \leftarrow SkewOpt(N_{i-1})$
15:  Run $ISTA(N_i, P_i)$
16:  **for all** endpoint $p$ in $P$ **do**
17:  **if** $slack(p) \geq 0$ **then**
18:  Replace error-tolerant register by a conventional one at endpoint $p$
19:  **end if**
20:  **end for**
21:  $C_i \leftarrow COST(N_i)$
22:  **if** $C_i < C_{min}$ **then**
23:  $C_{min} \leftarrow C_i$
24:  $N_{min} \leftarrow N_i$
25:  **end if**
26:  $P \leftarrow P - P_i$
27:  Update sensitivity values for all endpoints in $P$
28: **end for**
29: **return** $N_{min}$

Algorithm 13 describes our combined optimization, which we call *CombOpt*, to reduce the error-resilience overhead. The procedure takes as input a netlist $N$ which has error-tolerant registers at endpoints with timing violations. The procedure runs static timing analysis ($STA$)

and computes a sensitivity value for each endpoint $p$ (Lines 1-8). The procedure finds all fanin cells by tracing backward from the endpoint register using depth-first search. During the fanin-cone tracing, we count only the timing-critical fanin cells since non-critical fanin cells have little effect on the cost of endpoint optimization. The procedure optimizes the top $k$ endpoints according to the sensitivity in each iteration (Lines 12-13). $TimingOpt(N_{i-1}, P_i)$ (Line 13) represents a timing optimization on the set of endpoints $P_i$ in netlist $N_{i-1}$. We perform SkewOpt after optimization on the fanin cones of the top $k$ endpoints (Line 14). $ISTA(N_i, P_i)$ in Line 15 indicates incremental static timing analysis ($STA$) after optimization. If the timing slack of endpoint $p$ becomes positive, the procedure replaces the error-tolerant register of $p$ with a conventional one (Line 18). Then, the cost of the netlist ($COST(N_i)$) is updated. After the iterations of endpoint optimization, the procedure finds a netlist ($N_{min}$) which has a heuristically minimized cost in terms of area and/or power consumption.

**Construction of Error-Detection Network**

To detect timing errors, resilient designs typically connect all error-detection signals of error-tolerant registers via the error-detection network (e.g., an "OR tree"). There are two basic types of error-detection network. In *centralized pipeline recovery*, one error-detection network connects to all error-tolerant registers. In *distributed pipeline recovery*, a separate error-detection network can be applied to each pipeline stage [49]. For a design with large number of pipeline stages, centralized pipeline recovery can incur large cost in terms of wirelength, area and power, as compared to the distributed strategy. To be conservative about the resilience cost, in this work we assume a centralized pipeline recovery scheme. We also note that in a resilient design with distributed pipeline recovery our OR tree insertion algorithm can be applied to each pipeline stage, since all registers in the same pipeline stage are connected together with an error-detection network.

For a design in which the usage of error-tolerant registers is defined before synthesis, the construction of the error-detection network can be accomplished by commercial SP&R tools. However, in our optimization flow, the usage of error-tolerant registers is defined during the placement stage, where an algorithm is required to guide the construction of error-detection network. Further, the size of the error-detection network increases with the number of error-tolerant registers in a resilient design, and this can increase cost. Our initial studies show that in a pure-resilient design, when we construct the error-detection network based on a random clustering method, the wirelengths of the error-detection nets can contribute up to 9% of the design's to-

**Algorithm 14** OR tree insertion

**Procedure** $ORTreeInsertion(N)$

$Q \leftarrow$ registers with timing violations w.r.t. required margin in $N$
$N_{sol} \leftarrow N$
**while** $|Q| > 1$ **do**
  Compute distance matrix $D$ where $D_{i,j} = dist(c_i,\ c_j)$ $(c_{\{i,j\}} \in Q)$
  $Sol \leftarrow$ apply Hungarian method on $D$
  **for all** cycle in $Sol$ **do**
    $Q_{local} \leftarrow$ cells on the cycle
    **while** $|Q_{local}| > 1$ **do**
      $(c_1, c_2) \leftarrow$ find the nearest pair of cells in $Q_{local}$
      $c_{new}.x = \frac{c_1.x + c_2.x}{2}$
      $c_{new}.y = \frac{c_1.y + c_2.y}{2}$
      Insert OR cell $c_{new}$ to $N_{sol}$
      Connect outputs of $c_1$ and $c_2$ to inputs of $c_{new}$
      $Q_{local} \leftarrow Q_{local} \cup \{c_{new}\} \setminus \{c_1, c_2\}$
    **end while**
    $Q \leftarrow Q \cup \{c_{new}\}$
  **end for**
**end while**
**return** $N_{sol}$

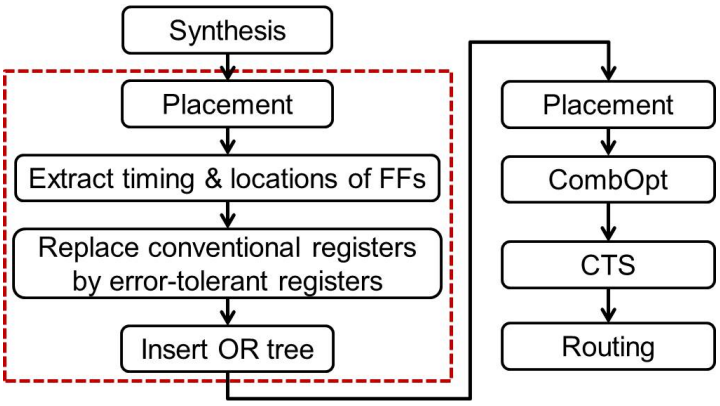



**Figure 3.33**: Implementation flow. OR tree insertion flow is indicated by the red dotted box.

tal wirelength. To minimize the overhead, we develop a matching-based clustering algorithm and use a commercial router to construct the OR tree. Figure 3.33 depicts our implementation flow (red dotted box). Based on the locations of error-tolerant flip-flops extracted from an initial placement, we construct the OR tree bottom-up. We heuristically cluster error-tolerant flip-flops and/or OR gates by iteratively applying (i) the Hungarian [222] method to achieve an assignment (in which cycles are considered to be clusters of flip-flops and/or OR gates) and (ii) a nearest-neighbor method to build an OR tree within a given cluster. Our OR tree insertion (i.e., clustering of error-tolerant flip-flops and insertion of OR gates) algorithm is described in Algorithm 14.

In the construction of the error-detection network, we start with a synthesized netlist which has only conventional flip-flops. Based on the timing information extracted from the initial placement, all flip-flops having negative slacks with respect to safety margin are set to be error-tolerant flip-flops (Line 1). We then calculate the Manhattan distances between each pair of error-tolerant flip-flops and construct a distance matrix accordingly. In the distance matrix, each row and each column corresponds to an error-tolerant flip-flop, such that the matrix entry $D_{i,j}$ is the distance $dist(c_i,\ c_j)$ between the $i^{th}$ and $j^{th}$ flip-flops (Line 4). To avoid a trivial assignment, we define $dist(c_i,\ c_i) = +\infty$ for all $i$. Since both rows and columns correspond to the same set of error-tolerant flip-flops, the distance matrix $D$ is a symmetric square matrix. We apply the Hungarian algorithm on the distance matrix to obtain a matching solution with minimum total distance (Line 5). The solution matrix (*Sol*) is a permutation matrix, i.e., there is exactly one '1' in each row and each column; this permutation can be decomposed into cycles that we consider as *clusters*.[44] Within each cycle, i.e., cluster, we use a nearest neighbor-based heuristic to construct an OR tree (Lines 8-14). Then, in the next iteration, we form a new distance matrix where each row and column correspond to one cluster from the previous iteration. The location of each cluster is defined by its center coordinates (i.e., the $x$ and $y$ coordinates of the cluster are respectively the averages of the $x$ and $y$ coordinates of all cells in the cluster). We continue the construction of the error-detection network until all error-tolerant flip-flops are connected.

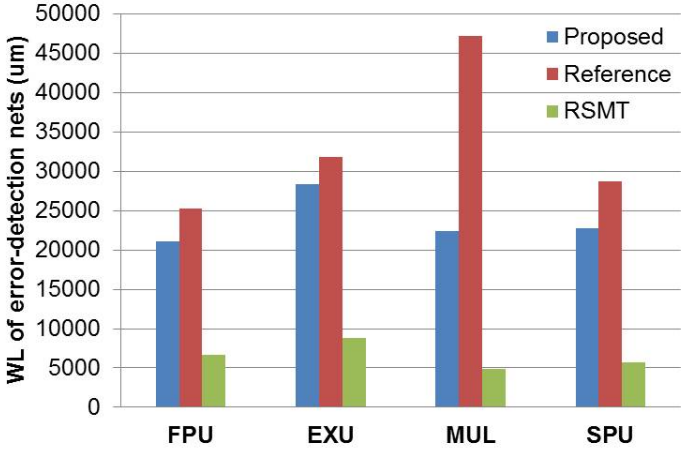



**Figure 3.34**: Our proposed OR tree insertion flow achieves an average of 29% wirelength reduction for the error-detection network, as compared to a reference flow. RSMT cost is a (loose) lower bound.

[44]For example, if the solution $Sol$ contains the matching edges $(c_{i_0},\ c_{i_1})$, $(c_{i_1},\ c_{i_2})$, ..., $(c_{i_{n-1}},\ c_{i_n})$ and $(c_{i_n},\ c_{i_0})$, this is a *cycle* in the permutation defined by $Sol$. We heuristically consider each cycle as a cluster of the $i_0^{th}$, $i_1^{th}$, ..., $i_n^{th}$ cells.

Figure 3.34 compares the wirelength of error-detection nets between our proposed OR tree insertion flow and a reference flow which performs ECO cell and net insertions to construct the error-detection network. The reference flow also uses the nearest-neighbor clustering method for each level of the OR tree. Both the proposed and the reference optimization methods construct an OR tree with only 2-input OR gates. A lower bound for the wirelength is given by the rectilinear Steiner minimum tree (RSMT) [214] over all error-tolerant flip-flops. However, this lower bound is far from achievable due to congestion induced by other nets and power/ground distribution. We compare wirelength values for four pure-resilient designs. The figure shows that our proposed flow achieves an average of 29% wirelength reduction compared to the reference flow.

Note that, when SEOpt replaces an error-tolerant flip-flop with a conventional flip-flop, we need to modify the OR tree as shown in Figure 3.35. The figure shows two cases of the flip-flop replacement. When the flip-flop (u2) is replaced with a conventional one, we remove the connected OR gate (u1) and modify the OR tree. The steps of the modification are as follows.

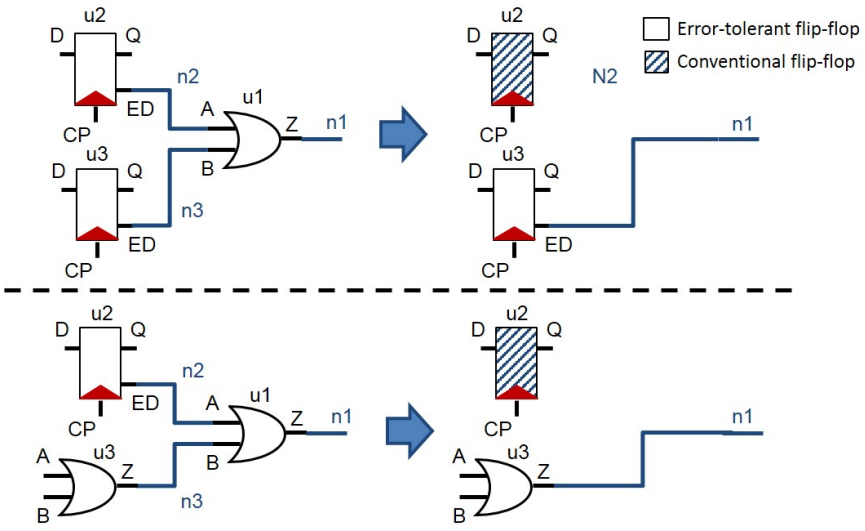



**Figure 3.35**: Replacement of an error-tolerant flip-flop with a conventional flip-flop for u2. Note that for readability, nets connected to D, Q and CP pins of flip-flops are not shown.

1. Detach nets (n1, n2 and n3) from the OR gate output (u1/Z) and flip-flop error detection pins (u2/ED and u3/ED);

2. Delete nets (n2 and n3) which are connected to the OR gate;

3. Delete the OR gate instance (u1);

4. Attach net n1 to another flip-flop (u3/ED) or OR gate (u3/Z);

5. Replace the error-tolerant flip-flop (u2) with a conventional flip-flop;

6. Refine placement and update timing.

### Typical-Corner Optimization

To accurately assess the benefits and overheads of resilience approaches, it is important to consider impacts of signoff corners and process variation during implementation. For paths with conventional registers as endpoints, we ensure that there is no timing violation at the slow corner. However, since error-tolerant registers can detect and correct timing errors, we can allow some amount of negative timing slack[45] for paths with error-tolerant registers as endpoints, and we should optimize these paths at the typical corner. Our process-variation-aware implementation is shown in Figure 3.36. In the figure, the *error-detection window* indicates the timing interval, or safety margin, during which a error-tolerant register can detect timing errors. The right hand side of the figure indicates larger timing slacks at endpoints. When an endpoint has no timing violation at the slow corner (SS, 125°$C$ in our experiments), we use the conventional flip-flop for that endpoint. Introduction of a guardband can enhance design robustness and enable adaptive voltage scaling. Error-tolerant registers can be used for other endpoints. Note that we evaluate error rate at the typical corner (TT, 25°$C$ in our experiments). Therefore, when an endpoint has negative slack with respect to the typical corner, this leads to throughput degradation.

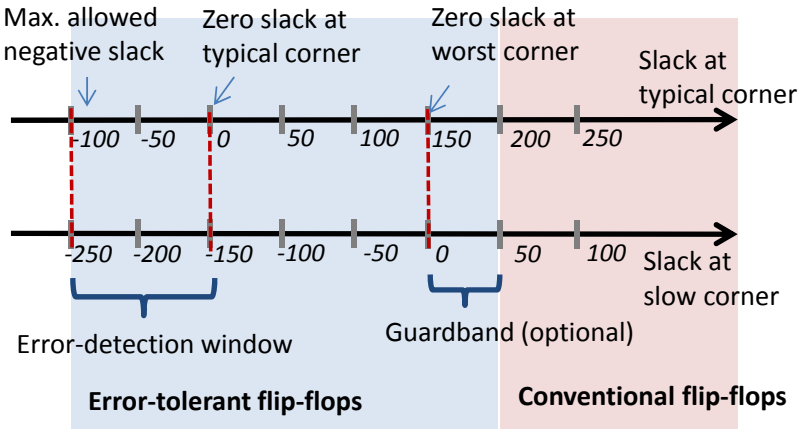



**Figure 3.36**: Illustration of how we consider process variation in our implementations. The slack values shown here are not representative of actual values in 28$nm$ FDSOI.

[45]The maximum allowable negative slack is determined by the design of the error-tolerant register.

We implement resilient designs at the typical corner, but with CombOpt, there will also be conventional flip-flops in the design. To ensure timing correctness of conventional flip-flops at the slow corner, we estimate slow-corner path delays and apply maximum-delay constraints correspondingly. Specifically, we perform timing analysis at both slow and typical corners, then calculate maximum-delay constraints based on the ratio between delay values at typical and slow corners, as shown in Equation (3.36). We perform the timing analysis and update the maximum-delay constraints before each iteration of the CombOpt and at the pre-placement, pre-CTS, pre-routing and post-routing (i.e., before signoff) stages.

$$max\_delay = (clock\_period - guardband) \times \frac{delay\_TT}{delay\_SS} \tag{3.36}$$

**Optimization with TIMBER Flip-Flops**

We now discuss optimization steps that are specific to TIMBER-based designs [40]. As described in Section 3.3.2, our optimization starts with a netlist in which all endpoints with timing violations use error-tolerant registers. To use TIMBER flip-flops, there are two additional constraints for selection of endpoints [32][63]. First, since TIMBER flip-flops borrow timing from their fanout timing paths to mask timing errors, we must avoid a loop of TIMBER flip-flops within which continuous time borrowing can cause timing failure. Second, additional timing slacks are required at endpoints with conventional flip-flops to compensate the accumulated time borrowings of TIMBER flip-flops in previous stages. Thus, a larger number of chained TIMBER flip-flops will lead to tighter timing constraints on the following timing paths, and corresponding increased power and area cost. Moreover, a larger number of chained TIMBER flip-flops requires more complex clock delay circuits, which also incurs area and power cost. In our study, we limit the chained TIMBER flip-flops to be less than three stages (i.e., assuming two error-detection intervals).

To address these additional constraints, we select endpoints based on sensitivity functions which indicate power changes due to the usage of TIMBER flip-flops. A higher sensitivity of an endpoint indicates that greater power reduction is expected from timing margin recovery with TIMBER. Since the maximum stage number of chained TIMBER flip-flops is two, we formulate the optimization as two *maximum-weight independent set (MWIS)* problems in sequence such that no TIMBER flip-flops are adjacent after the first-stage optimization, and no chained TIMBER flip-flops with more than two stages after the second-stage optimization.

We formulate the MWIS problem as a *Integer Linear Program* (ILP).

$$\begin{aligned} &\text{Maximize} && \textstyle\sum_{p_i \in P} SF(p_i) \times B_i \\ &\text{Subject to} && B_i + B_j \leq 1, \quad (p_i \text{ and } p_j \text{ are adjacent in } G) \\ & && B_i = 0 \text{ or } 1 \end{aligned} \tag{3.37}$$

in which $SF(p_i)$ is the sensitivity function of endpoint $p_i$, $P$ is the set of endpoints, and $B_i$ is a binary variable indicating whether $p_i$ uses a TIMBER flip-flop (i.e., $B_i = 1$) or not (i.e., $B_i = 0$). The constraints specify that no TIMBER flip-flops are adjacent in graph $G$. Note that in the first-stage optimization, $G$ is the sequential graph extracted from the netlist. In the second-stage optimization, we update graph $G$ such that we remove the selected vertices from the first-stage optimization. Further, for each removed vertex, we connect each of its incoming vertices to its outgoing vertices.

To describe the calculation of sensitivity functions, we first define function $f(c, t)$ as

$$f(c,t) = \begin{cases} |slack(c) - t| \times power(c), & \text{if } slack(c) < t \\ 0, & \text{otherwise} \end{cases} \tag{3.38}$$

in which $c$ is a cell in the netlist, $slack(c)$ and $power(c)$ are respectively the worst timing slack and power of $c$, and $t$ is a constant timing interval.

We assume that all endpoints in the initial netlist use conventional flip-flops. By replacing an endpoint with a TIMBER flip-flop, we can recover timing margins in the fanin timing paths by one error-detection window. But this also leads to additional margin insertion which is equal to one error-detection window in the fanout paths (as shown in Figure 3.37(a)). To estimate the power change from such timing margin migration, we define the sensitivity of endpoint $p$ as

$$SF_{TBF\_a}(p) = \sum_{c \in cone(c,p)} f(c, t_{ED}) - \sum_{c \in cone(p,c)} f(c, t_{ED}) \tag{3.39}$$

in which $cone(c, p)$ (respectively, $cone(p, c)$) indicates the combinational logic cone between conventional flip-flops and $p$ (respectively, $p$ and conventional flip-flops), and $t_{ED}$ indicates the duration of the error-detection window.

In the second optimization stage, some of the endpoints have been selected to use TIMBER flip-flops. Therefore, to calculate the sensitivity function values for the remaining endpoints, there are four scenarios as shown in Figure 3.37. Endpoints in (a) are not promising, otherwise they would have been selected in the first optimization stage. Selection of endpoints in (b) will violate the constraint that there are no chains of more than two TIMBER flip-flops.

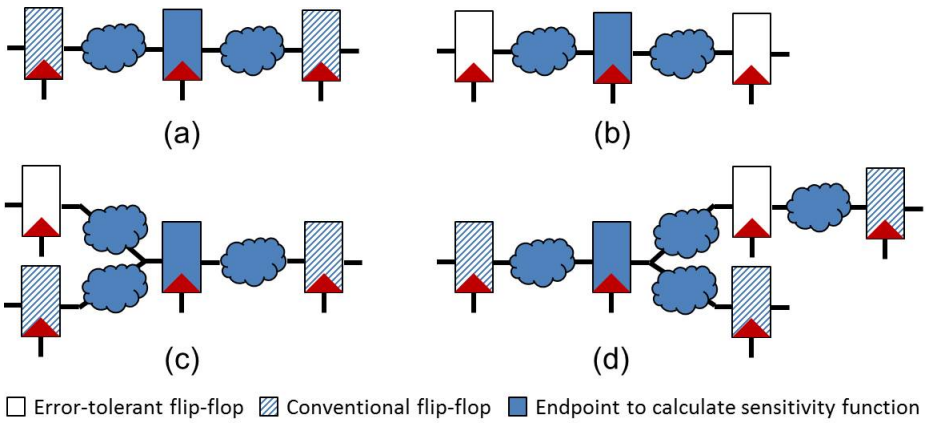



**Figure 3.37**: Scenarios of sensitivity-function calculation for selection of TIMBER flip-flops.

Based on analysis similar to that shown above, we calculate the sensitivity functions corresponding to scenarios (c) and (d) as follows.

$$SF_{TBF\_c}(p) = \sum_{c \in cone(t,p)} f(c, 2 \times t_{ED}) + \sum_{c \in cone(c,p)} f(c, t_{ED}) - \sum_{c \in cone(p,c)} f(c, 2 \times t_{ED}) \quad (3.40)$$

$$SF_{TBF\_d}(p) = \sum_{c \in cone(c,p)} f(c, t_{ED}) - \sum_{c \in cone(p,c)} f(c, t_{ED}) - \sum_{c \in cone(t,-)} f(c, t_{ED}) \quad (3.41)$$

Here, $cone(t, p)$ indicates the combinational logic cone between TIMBER flip-flops and $p$, and $cone(t, -)$ indicates the fanout cone of TIMBER flip-flops which are endpoints of the fanout timing paths of $p$.

### 3.3.3 Experimental Results

#### Experimental Setup

We implement experiments with four sub-modules (Table 3.9) from the *OpenSPARC T1* processor [231]. The modules are implemented in a foundry $28nm$ FDSOI technology; synthesis is performed with *Synopsys Design Compiler vH-2013.03-SP3* [237], placement and routing are performed with *Cadence Encounter Digital Implementation System v13.1* [217]. We use three error-tolerant flip-flops in our experiments, with overheads of power, area (estimated based on extra transistor count), and throughput as given in Table 3.10.

In our experiments, (i) we model power penalty by multiplying the total power of the error-tolerant flip-flops by the corresponding power overhead; (ii) we model area overhead by scaling the size of flip-flops in LEF; and (iii) we model the safety margin and additional hold margin of error-tolerant flip-flops by adding constant shifts to setup and hold constraint values in the Liberty model. To validate our estimation of error rate (Equation (3.26)) and determine $\alpha$,

we perform gate-level simulation using *Cadence NC-Verilog v8.2* [216]. Figure 3.38 compares the actual error rates and estimated error rates at different supply voltages based on input vectors with random values. Our estimated error rates roughly match the actual values. To find timing slack and power values at specific voltages, we prepare Synopsys Liberty (.lib) files containing 90 commonly used cells (40 combinational and five sequential, with dual-$V_{th}$ flavors) for each value of supply voltage from $1.20V$ to $0.50V$ in $20mV$ increments, using *Synopsys SiliconSmart v2013.06-SP1* [236].

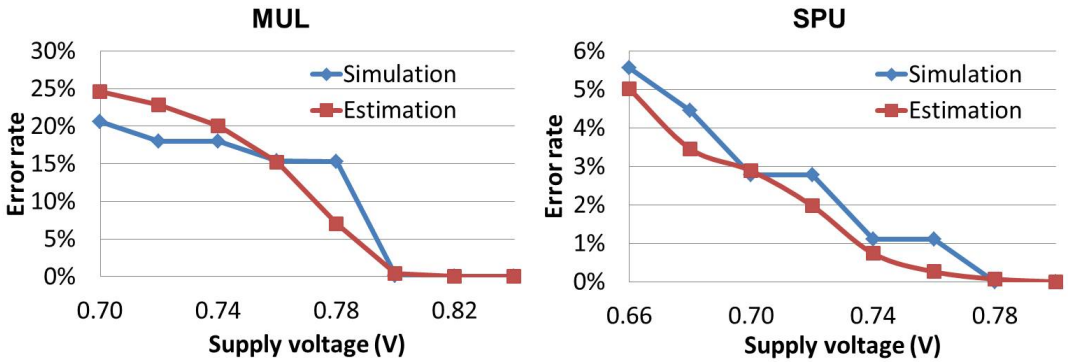



**Figure 3.38**: Actual error rates vs. estimated error rates at different voltages.

**Table 3.9**: Testcases from OpenSPARC T1.

| **Module** | **Description** | **#Instances** | **Area** ($\mu m^2$) |
|---|---|---|---|
| *FPU* | floating point adder | 12986 | 34633 |
| *EXU* | integer execution | 17614 | 58721 |
| *MUL* | integer multiplier | 13162 | 40693 |
| *SPU* | stream processing | 8066 | 28150 |

**Table 3.10**: Penalties of error-tolerant flip-flops.

| **Design** | Razor | Razor-Lite | TIMBER |
|---|---|---|---|
| **Power penalty** | 30% [50] | ~0% [122] | 100% [40] |
| **Area penalty** | 182% [122] | 33% [122] | 255% [35] |
| **#Recovery cycles** | 5 [190] | 11 [122] | 0 [40] |

**Methodology Comparison**

In our first experiment, we compare design energy and area resulting from CombOpt to (i) pure-margin designs[46] and (ii) a brute-force methodology, i.e., a typical resilient design implementation, where resilient endpoints are (greedily) instantiated at timing-critical endpoints.[47] We use Razor flip-flops for error resilience in this experiment. We compare our methodology at three different slow corners, where corresponding SS corners are at $1\sigma$, $2\sigma$ and $3\sigma$. The clock period for all implementations is $0.9ns$. We use the minimum voltage that satisfies timing constraints at the slow corner for pure-margin implementations; for brute-force and CombOpt implementations, we use a discretized exhaustive search to determine the signoff voltages (i.e., we search for the signoff voltage that achieves minimum energy within -$80mV$ of the signoff voltage of pure-margin, with a step size of $20mV$). The runtimes for *FPU*, *EXU*, *MUL* and *SPU* are respectively 30 $min$, 20 $min$, 60 $min$ and 7 $min$ per iteration of CombOpt on a $2.5GHz$ Intel Xeon server with four threads. We perform 10 iterations of optimization on each design in the experiments.

Figure 3.39 shows that the benefits of CombOpt increase with a larger process variation. In the figure, small, medium and large margins respectively indicate $1\sigma$, $2\sigma$ and $3\sigma$ for SS corner. We observe that CombOpt achieves up to 10% (8% on average) energy reduction compared to the brute-force method, and up to 21% (12% on average) energy reduction compared to the conventional pure-margin method. With larger process variation, brute-force has larger energy cost due to throughput degradation (e.g., *FPU* and *EXU*), while CombOpt is able to jointly minimize the number of error-tolerant flip-flops and error rate, thus achieving greater improvement over brute-force.

The additional circuits for error detection typically cause resilient designs to have larger area than conventional designs. We observe that the brute-force method leads to an average of 45% area overhead. Using CombOpt, we reduce the area overhead to 23%. In the regime of "dark silicon" [62], energy cost is "more expensive" than area cost, and our optimization thus focuses mainly on energy reduction.

Figure 3.39 also shows the comparison of error rates for designs resulting from CombOpt and brute-force. We observe that CombOpt achieves smaller error rates on most of the testcases. However, our optimization does not explicitly minimize the error rate of a design, but

[46] We define a *pure-margin design* as one wherein only timing margins are inserted to ensure correct operation under dynamic variation.

[47] We implement designs without considering the safety margin, then replace with error-tolerant registers any endpoints having timing violations with respect to the safety margin.

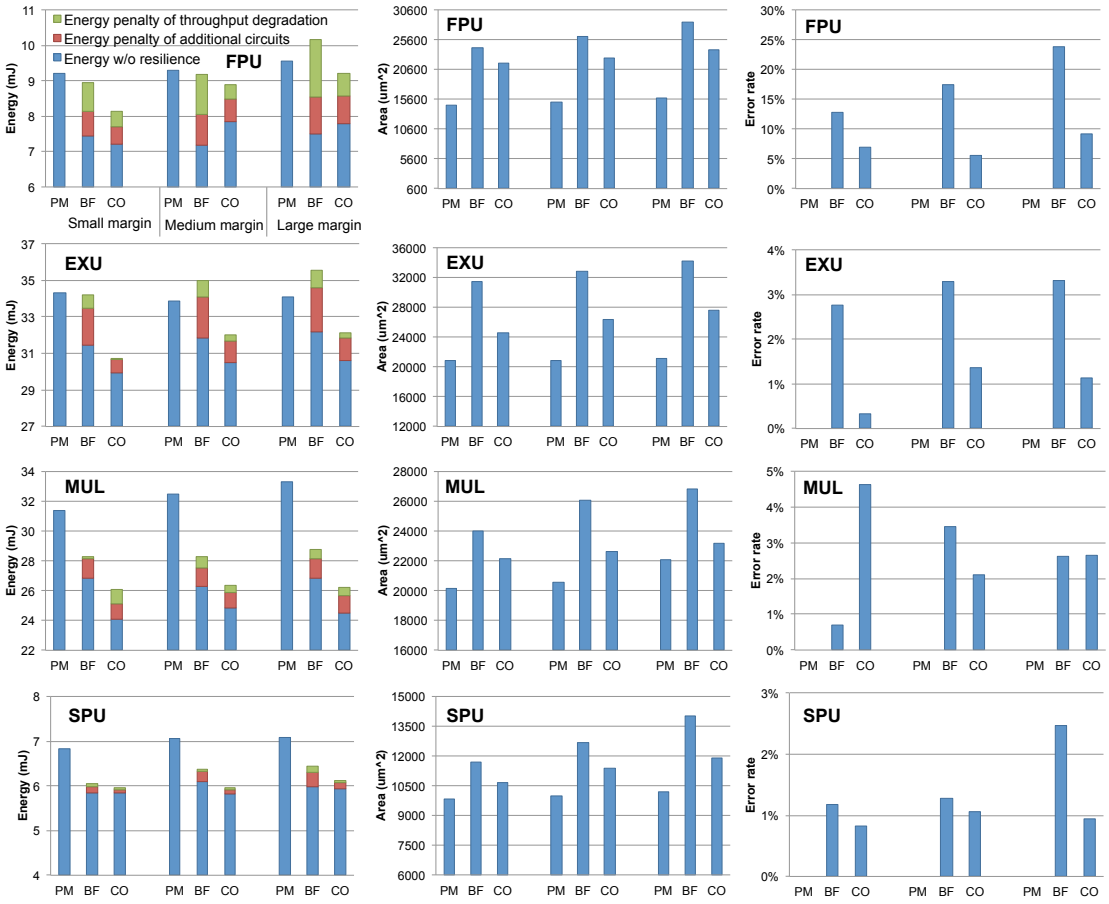



**Figure 3.39**: Energy and area results from different implementation methodologies – pure-margin (PM), brute-force (BF), and CombOpt (CO).

rather the design energy considering the tradeoff between cost of resilience and margin on combinational paths. For example, although for *MUL* with small margin CombOpt leads to a larger error rate than that resulting from brute-force, the power of combinational paths (i.e., indicated by *energy w/o resilience*) is significantly reduced in CombOpt, which leads to smaller design energy.

### Impact of Clock Skew Optimization

As discussed in Section 3.3.2, SkewOpt improves slacks of all timing-violating paths with respect to the safety margin to minimize the error rate. Moreover, the improved timing slacks are further exploited to enable removal of error-tolerant registers and power reduction on data paths. To assess the impact of clock skew optimization, we perform optimization without SkewOpt and compare the outcomes with those of CombOpt. As shown in Table 3.11, Com-

bOpt achieves reduced design energy (in terms of both throughput penalty and power of circuit), total cell area and number of error-tolerant flip-flops as compared to the optimization without SkewOpt. Further, since SkewOpt comprehends hold constraints, performing optimization with SkewOpt does not increase the number of hold buffers significantly. For the *MUL* testcase, applying SkewOpt even reduces the number of hold buffers due to the decreased number of error-tolerant flip-flops. Table 3.11 further compares our optimization solution (CombOpt) to that of a conventional resilient design implementation with application of SkewOpt at the post-placement stage. We observe from the fourth and seventh columns of Table 3.11 that although SkewOpt alone reduces the throughput penalty, ignoring the tradeoff between data path optimization and cost of resilience (which is captured by SEOpt) leads to more error-tolerant flip-flops and significant area and power overheads. Specifically, for *MUL*, CombOpt achieves 5% and 13% more reduction in energy and area, respectively, as compared to brute-force+SkewOpt.

**Table 3.11**: Impact of SkewOpt.

| **Design** | *MUL* | | | *SPU* | | |
|---|---|---|---|---|---|---|
| **Flow** | CombOpt | w/o SkewOpt | brute-force+SkewOpt | CombOpt | w/o SkewOpt | brute-force+SkewOpt |
| **Total energy** ($mJ$) | 26.19 | 27.07 | 27.54 | 6.12 | 6.18 | 6.28 |
| **Throughput penalty** ($mJ$) | 1.14 | 1.16 | 0.92 | 0.05 | 0.08 | 0.05 |
| **#Error-tolerant flip-flops** | 660 | 780 | 1003 | 225 | 269 | 465 |
| **Total cell area** ($\mu m^2$) | 23200 | 24315 | 26352 | 11922 | 12324 | 13931 |
| **#Hold buffers** | 214 | 321 | 445 | 345 | 107 | 220 |

### Impacts of Short Path Padding and Error-Detection Network

A common observation is that resilient designs require a large hold margin, which necessitates more buffers for short-path padding. We include this overhead by assuming that the required hold margin for a resilient design is equal to its safety margin, i.e., 15% of the clock period. Further, the error-detection network (i.e., OR tree) incurs additional energy and area penalties. We evaluate short-path padding and error-detection network overheads by removing the additional hold margin and/or error-detection network in the implementations, and then assessing energy and area differences.

Figure 3.40 shows the energy and area outcomes for *MUL*. All implementations are optimized with CombOpt. For the *without hold* case, we ignore the additional hold margin during the implementation; for the *without OR tree* case, we remove the error-detection network at the post-routing stage and perform an incremental optimization; and for the *without hold & OR tree* case, we ignore the additional hold margin during the implementation and remove the

error-detection network at the post-routing stage.

Since CombOpt significantly reduces the number of error-tolerant flip-flops (Razor in this example) and the size of the error-detection network, the energy and area cost of short-path padding and error-detection network is small. The short-path padding leads to 4% energy and 2% area cost; and the error-detection network leads to $< 1\%$ energy and 2% area cost. Another reason for the small energy cost of the error-detection network is its low activity. Figure 3.41 shows an example CombOpt implementation result, in which the area overhead of hold buffers and OR gates is only 2.7%.

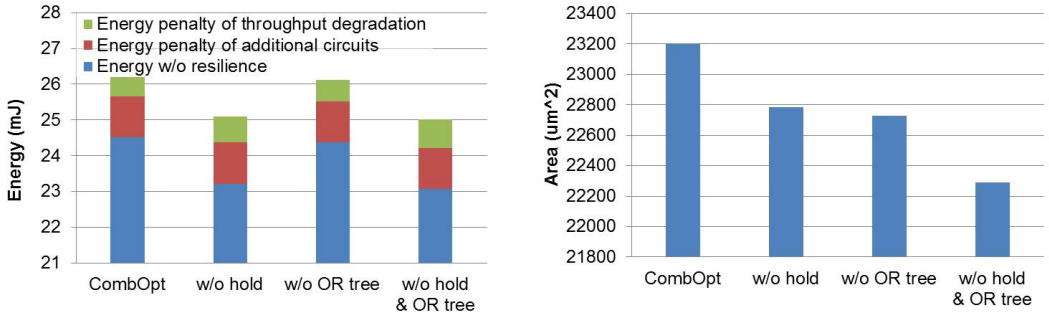



**Figure 3.40**: Impacts of hold margin and error-detection network. Design: *MUL* (OpenSPARC T1). Technology: $28nm$ FDSOI.

**Figure 3.41**: Layout of CombOpt result for the *SPU* testcase with $3\sigma$ corner. Razor flip-flops are in blue; conventional flip-flops are in purple; OR gates are in red; and hold buffers are in green.

### Typical-Corner Optimization

Because resilient designs can detect and recover from timing errors, it is not necessary to optimize them at the slow corner. Furthermore, power analysis (especially error rate estimation) of resilient designs at the slow corner (which is the case in [108]) can be pessimistic. We assess the pessimism of slow-corner optimization by performing error rate estimation and energy analysis (of designs shown in Figure 3.39) at the slow corner.

As shown in Table 3.12, energy can be overestimated by up to 21% when we perform an energy analysis at the slow corner for resilient designs. This is mainly caused by the overestimated error rates.

**Table 3.12**: Pessimism of slow-corner optimization.

| **Design** | *FPU* | *EXU* | *MUL* | *SPU* |
|---|---|---|---|---|
| **Typical-corner analysis** (TT, 25°$C$) | | | | |
| **Total energy** ($mJ$) | 9.21 | 32.12 | 26.19 | 6.12 |
| **Throughput penalty** ($mJ$) | 0.63 | 0.29 | 0.54 | 0.05 |
| **Slow-corner analysis** (SS w/ 3$\sigma$, 125°$C$) | | | | |
| **Total energy** ($mJ$) | 10.60 | 34.50 | 31.62 | 6.36 |
| **Throughput penalty** ($mJ$) | 1.46 | 1.77 | 5.36 | 0.14 |

### Validation with Different Switching Activities

To validate our proposed optimization and study the impact of switching activity on energy consumption, we analyze the energy of an implemented design (*MUL*) across a range of switching activity assumptions. Figure 3.42 shows that CombOpt can achieve the minimum energy when the activity factor is no lower than 5%. With a lower activity factor (e.g., 1%), error rate decreases and the benefits of CombOpt over the brute-force method become smaller. Compared to pure-margin, CombOpt achieves reduced energy of combinational cells through SEOpt. However, with a low activity factor, clock energy dominates and thus the benefits of CombOpt also decrease.

### Study of Different Error-Tolerant Flip-Flops

We also study the cost of different error-tolerant flip-flops. We compare designs implemented with Razor, Razor-Lite and TIMBER types of error-tolerant flip-flops. We implement the designs with the brute-force methodology mentioned above, and CombOpt.

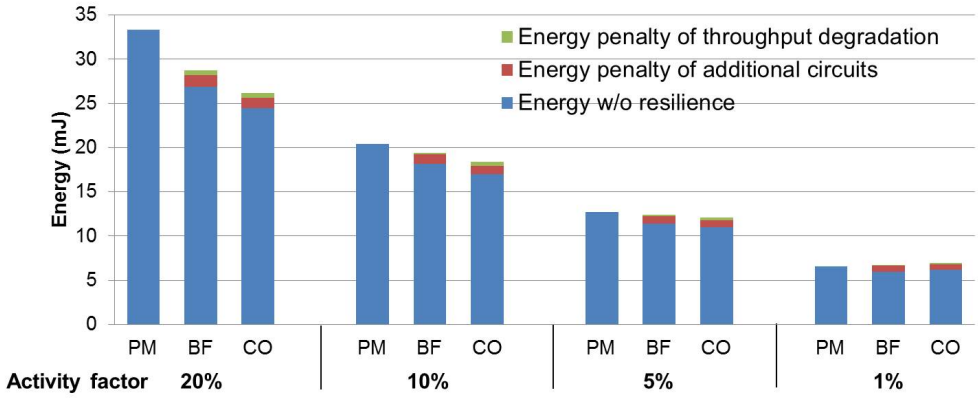



**Figure 3.42**: Energy consumption with different switching activity factors. Design: *MUL* (OpenSPARC T1). Technology: $28nm$ FDSOI.

**Table 3.13**: Comparison among error-tolerant flip-flops.

| **Design** | Razor-Lite | Razor | TIMBER |
|---|---|---|---|
| **Method** | brute-force | | |
| **Total energy** $(mJ)$ | 27.71 | 28.78 | 33.62 |
| **Energy w/o resilience** $(mJ)$ | 27.16 | 26.87 | 29.74 |
| **Energy w/ additional circuits** $(mJ)$ | 0.00 | 1.32 | 3.87 |
| **Energy w/ throughput penalty** $(mJ)$ | 0.55 | 0.59 | 0.00 |
| **#Error-tolerant flip-flops** | 931 | 924 | 575 |
| **Total cell area** $(\mu m^2)$ | 21064 | 26814 | 26700 |
| **Method** | CombOpt | | |
| **Total energy** $(mJ)$ | 26.14 | 26.19 | 28.20 |
| **Energy w/o resilience** $(mJ)$ | 25.55 | 24.51 | 27.43 |
| **Energy w/ additional circuits** $(mJ)$ | 0.00 | 1.14 | 0.77 |
| **Energy w/ throughput penalty** $(mJ)$ | 0.60 | 0.54 | 0.00 |
| **#Error-tolerant flip-flops** | 627 | 660 | 128 |
| **Total cell area** $(\mu m^2)$ | 19164 | 23200 | 20464 |

Table 3.13 shows results for *MUL*, where the slow corner is at SS with $3\sigma$ and all designs are implemented using CombOpt. We observe that although Razor-Lite has negligible energy and area overheads, it leads to 11% more energy penalties due to throughput degradation as compared to Razor. The small number of TIMBER flip-flops is because of additional constraints described in Section 3.3.2. Further, TIMBER flip-flops require additional timing margin on fanout timing paths to compensate timing errors, which leads to larger energy of combinational cells as compared to Razor and Razor-Lite. Note that we also consider the area and power

overheads of error relay logic[48] between two stages of TIMBER flip-flops [63]. We group the TIMBER flip-flops which share the same fanin TIMBER flip-flops and insert the error relay logic for each group. We also observe that CombOpt significantly reduces the number of error-tolerant flip-flops (by 33%, 29% and 77% for Razor-Lite, Razor and TIMBER[49] , respectively). Such reductions can be enabling to the energy- and area-feasibility of resilient designs.

**Energy Reduction from Adaptive Voltage Scaling**

In our last experiment, we study the energy reduction of resilient design in an adaptive voltage scaling context. We compare energy of designs implemented with the brute-force method and our CombOpt at different supply voltages. Note that to allow voltage downscaling, we build a margin of 25% of the clock period into the paths that have conventional flip-flops as endpoints. In addition, for each testcase we implement a pure-margin design that satisfies timing constraints at minimum voltage, as a reference. Figure 3.43 shows results for our four testcases. The designs implemented with CombOpt achieve significant energy reduction with voltage scaling. This is because our optimization comprehends the toggle information and trade-off between power consumption on combinational cells and error-tolerant registers; this results in less energy penalty from throughput degradation and additional circuits. Note that although throughput degradation increases at lower supply voltages, the total power also reduces. This leads to the observed decrease in energy cost of throughput degradation at lower supply voltages for most cases. However, further downscaling of the supply voltage is limited by the paths with conventional flip-flops as endpoints. We also observe that the benefits of resilience can be design-dependent: a design with larger error rate (e.g., *FPU*) derives less benefit from resilience because of large recovery overheads. From our proposed optimization (CombOpt), we achieve 8% and 17% energy reductions on average compared to the brute-force and conventional (pure-margin) methods, respectively.

### 3.3.4 Conclusion

By allowing timing errors, resilient design techniques can reduce design effort and obtain power and area benefits over conventional designs which always operate correctly. However,

[48] Based on the error-detection signal of a TIMBER flip-flop, an error relay logic determines the number of time intervals to mask timing errors of the TIMBER flip-flop in the successive stage.

[49] We observe similar improvement (i.e., 7% energy reduction) on an industrial processor (with 13K instances and 1642 flip-flops) at $40nm$ technology as compared to the brute-force method. In the example, the number of TIMBER flip-flops is reduced from 363 to 10, in which cells in the fanin cones of the selected 10 endpoints consumes 56% of total power in the corresponding conventional design.

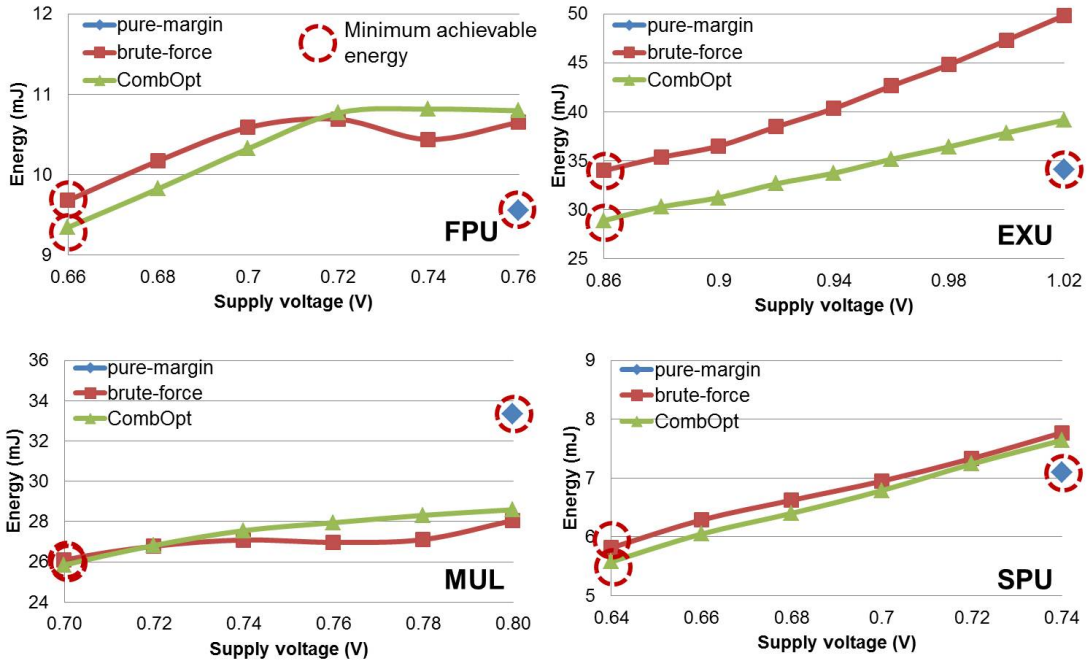



**Figure 3.43**: Energy consumption with voltage scaling, and minimum achievable energy for each method.

throughput and circuit power and/or area overheads can diminish the benefits of resilient design.

In this work, we provide a new design flow for mixing of resilient and non-resilient circuits within a given implementation, so as to minimize the overhead of error resilience. We propose a *selective-endpoint optimization*, which reduces timing-critical endpoints with small cost of timing optimization. We also propose a *clock skew optimization*, specifically targeted to a resilient design methodology, which improves robustness to process, voltage and temperature variations. In addition, we propose a matching-based algorithm to construct the error-detection network with substantially reduced wirelength cost. Our implementations further account for the impacts of signoff corners and process variation. Our proposed optimization techniques achieve significant energy reductions – up to 21% and 10% – compared to conventional (pure-margin) design and a brute-force resilience implementation, respectively. In an adaptive voltage scaling context, our method shows further benefits of error resilience.

A number of research directions remain open. In particular, our ongoing work seeks to (1) find an improved sensitivity metric to determine the minimum set of endpoints to optimize for minimum power and area, and (2) build a unified framework for simultaneous data- and clock-path optimization.

## 3.4 Acknowledgments

Chapter 3 contains reprints of Kristof Blutman, Hamed Fatemi, Andrew B. Kahng, Ajay Kapoor, Jiajia Li, and José Pineda de Gyvez, "Floorplan and Placement Methodology for Improved Energy Reduction in Stacked Power-Domain Design", *Proc. Asia and South Pacific Design Automation Conference*, 2017; Andrew B. Kahng, Jiajia Li and Lutong Wang, "Improved Flop Tray-Based Design Implementation for Power Reduction", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2016; Andrew B. Kahng, Seokhyeong Kang, Jiajia Li and José Pineda de Gyvez, "An Improved Methodology for Resilient Design Implementation", *ACM Transactions on Design Automation of Electronic Systems* 20(4), 2015; and Andrew B. Kahng, Seokhyeong Kang and Jiajia Li, "A New Methodology for Reduced Cost of Resilience", *Proc. Great Lakes Symposium on Very Large Scale Integration*, 2014. Chapter 3 also contains the draft submitted to *IEEE Transactions on Very Large Scale Integration Systems*, Kristof Blutman, Hamed Fatemi, Andrew B. Kahng, Ajay Kapoor, Jiajia Li and Jose Pineda de Gyvez, "Logic Design Partitioning for Stacked Power Domains", 2017. The dissertation author is the primary author of the papers and the submitted draft.

I would like to thank my coauthors Kristof Blutman, Hamed Fatemi, José Pineda de Gyvez, Andrew B. Kahng, Seokhyeong Kang, Ajay Kapoor and Lutong Wang, as well as the research support from NXP Semiconductors.

# Chapter 4

# Mixed-Fabric Optimization

Circuits are typically designed with "mixed fabrics", such as standard cells with different $V_{th}$ flavors and track heights, or tiers with different process corners in a 3DIC. In this chapter, we propose the concept of "mixed-fabric optimization" for improved design quality. We present three examples of mixed-fabric optimization. First, we propose a novel physical design optimization flow to implement design blocks with mixed, non-integer multiple-height cells in a fine-grained manner. Our optimization resolves the "chicken-and-egg" loop between floorplan site definition and the optimized choices of cell heights after placement. With full comprehension of the constraints and costs of mixing cells of different heights (e.g., the "breaker cell" area overheads of row alignment between sub-blocks of 8T and 12T cell rows), our optimization achieves up to 30+ percent area and power reductions versus 12T-only implementation while maintaining the same performance, and up to 10+ percent performance improvement along with power and area reductions versus 8T-only implementation. Second, we propose NOLO, a simple, "no-loop" predictive useful skew flow with dual-$V_{th}$ libraries that applies useful skew at the post-synthesis stage within a one-pass chip implementation. Experimental results in a $28nm$ FDSOI technology show that our predictive useful skew flow can reduce runtime by 66% and improve total negative slack by 5% compared to the previous useful skew back-annotation flow. Third, we study the "mix-and-match" of multiple stacked die, according to binning information, to improve overall product yield. We study die-stacking optimization to improve 3DIC product reliability, as well as performance improvements that are achievable in 3DIC implementation by leveraging – during the design flow – foreknowledge of mix-and-match die stacking during manufacturing. For the design-stage optimization for mix-and-match die stacking, 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. Experimental results show that our optimization flow achieves up to overall 16% timing improvement, relative to the existing 3DIC implementation flow, in the context of mix-and-match die stacking.

## 4.1 Design Implementation with Non-Integer Multiple-Height Cells for Improved Design Quality in Advanced Nodes

Standard cell-based implementation has been widely used for VLSI designs due to its relatively accurate abstraction and semi-regular layout. In advanced nodes, cells are designed with different heights (e.g., different numbers of fins in FinFET node). Larger cell heights have higher drive strengths at the cost of larger area and power consumption as well as pin capacitance. Smaller cell heights result in relatively smaller area, but have weaker drive strengths and are more likely to suffer from routing congestion and pin accessibility issues.[50] Figure 4.1 shows the delay and area tradeoff of buffers and inverters at foundry $28nm$ LP technology. In red are 12T cells, and in blue are 8T cells. As expected, 12T cells can achieve smaller delay at the cost of larger area.

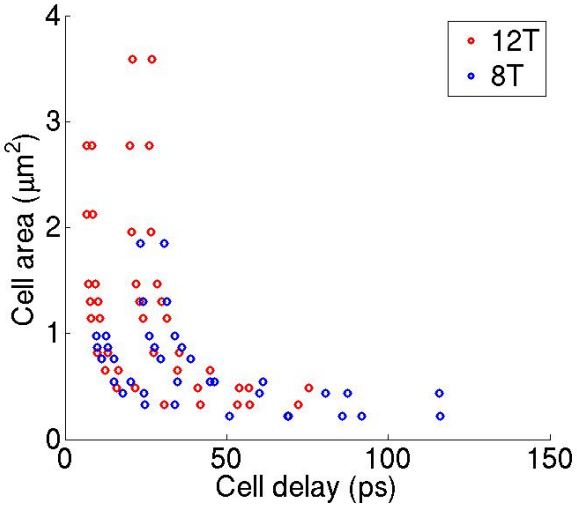



**Figure 4.1**: Delay-area tradeoff of 8T and 12T buffers/inverters in $28nm$ LP foundry libraries. Load cap = FO4 + $20\mu m$ M3 wire.

Given that cells of different heights exhibit different tradeoffs among performance, power and area, mixing cells of different heights in a design is able to provide a larger so-

[50] Although a cell with smaller height can be designed with large width to gain drive strength, the additional poly capacitance and metal capacitance can lead to area and power overheads as compared to a cell with the larger height.

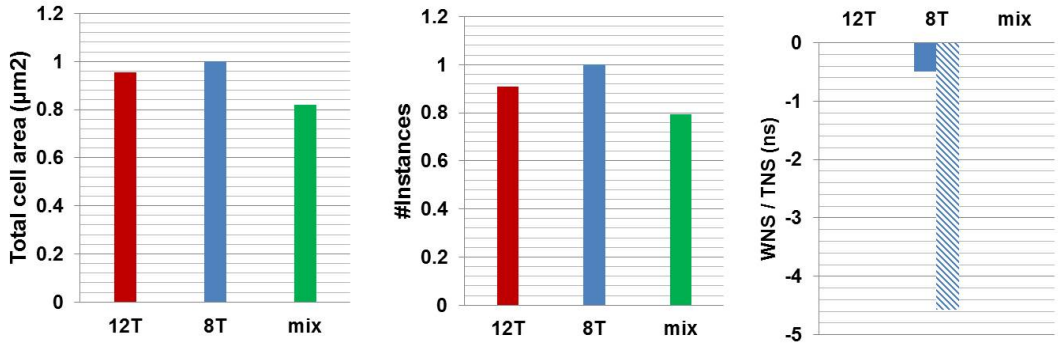



**Figure 4.2**: Post-synthesis netlist with mixed cell heights has significant area reduction compared to 12T-only and 8T-only netlists. Technology: $28nm$ LP. Design: *AES*. Frequency: $1.5GHz$. Corner: (SS, $0.95V$, $125°C$).

lution space and improved design quality. Figure 4.2 shows the post-synthesis area and timing comparison among implementations of an open-source design *AES* [230] with 12T-only, 8T-only and mixed cell heights, where total cell area and number of instances are normalized to those of the 8T-only case. In the right figure, the solid bar indicates WNS, and the shaded bar indicates TNS. Implementations with 12T-only and mixed cell heights have no timing violations. Due to generally larger areas of cell instances (particularly those with low drive strengths), 12T-only implementation tends to have larger design area. On the other hand, weak drive strengths of 8T cells result in a large number of buffer insertions to meet timing constraints, which also increases design area.[51] We observe from the example that mixing cell heights achieves 14% and 18% area reduction at the post-synthesis stage versus the 12T-only and 8T-only implementations, respectively.

Motivated by the above observations, in this work we propose to mix cell heights at the sub-block level (i.e., within a single P&R block) of physical implementation, to achieve improved design quality – specifically, tradeoffs of achievable performance, power and area.[52] However, optimizing a design by mixing non-integer multiple cell heights is highly nontrivial. The challenges include the following.

- Current design methodologies and tool flows can only mix cells of different heights at the block level, i.e., each block of a design uses cells with a particular height, with fine-grained mixing not available with today's EDA tools.

[51] The larger total cell area of the 8T-only netlist as compared to that of the 12T-only netlist is due to tight timing constraints. We demonstrate in Section 4.1.4 that with loose timing constraints, 12T-only implementations typically incur area overhead as compared to 8T-only ones.

[52] Since today's P&R tools already support placement of integer multiple-height standard cells (e.g., [217]), this work mainly focuses on implementation with non-integer multiple-height cells, which has not been addressed in the literature.

- There is a "chicken-and-egg" quandary: heights of cell rows are defined (in the placement site map) at the floorplan stage, but the optimized choices of cell heights are highly dependent on the placement outcome and timing constraints.
- There are costs associated with mixing cells of different heights. For instance, "breaker cells" must be inserted for row alignment between sub-blocks of cell rows with different heights.[53]

The contributions of this work are as follows.

- To our knowledge, we are the first in the literature to propose mixed cell-height implementation with non-integer multiple heights at the sub-block or sub-island level in advanced nodes.
- We develop methodologies which can easily be integrated within existing physical design flow, using standard commercial tools, for mixed cell-height implementation.
- We show that mixing 12T and 8T cells in a $28nm$ LP foundry technology achieves up to 30+ percent area and power reductions versus 12T-only implementation while maintaining the same performance, and up to 10+ percent performance improvement along with power and area reductions versus 8T-only implementation.

### 4.1.1 Related Work

To our knowledge, there is no previous work on mixed cell-height design methodology that addresses non-integer multiple heights within a block. A recent work [137] optimizes cell placement with heights being integer multiples of a particular cell height. And, commercial placers have been capable of handling multi-cell row height cells (i.e., with heights that are integer multiples of a single-row height) for over two decades. However, placement of single-height, double-height, triple-height etc. cells does not require site map change (i.e., the floorplan rows and placement sites are fixed during the optimization), whereas in our problem, different non-integer cell heights require different row and site definitions. Therefore, the placement problem involving cells with multiple non-integer heights cannot be solved by existing optimizations that handle the integer-multiples problem.

Notwithstanding the above, the mixed cell-height placement problem bears some similarity to the problem of voltage island placement, in that both problems try to assign a certain

[53] We define a *breaker cell* as the space (i.e., placement and/or routing blockages) that must be inserted between the boundaries of regions of different cell heights.

attribute with different values (e.g., cell height or supply voltage) to standard-cell instances, in order to improve performance or reduce power. Further, both problems must comprehend a type of incurred cost (e.g., area overhead of "breaker cells" or insertion of level shifters) when performing the assignment. We therefore review exemplary works from the literature on voltage island placement.

Wu et al. [197] and Ching et al. [36] propose partitioning methodologies to divide post-placement die area into regions which will be assigned to different supply voltages. The objectives are respectively to minimize the number of partitions and to handle non-rectangular partitions. Wu et al. [200] further consider timing constraints during the voltage assignment. However, the interaction with standard-cell placement is missing in all of these works. An improved optimization proposed in [199] performs incremental placement to move timing-critical cells out from the low-voltage regions by adjusting net weights and cell delays. Guo et al. [79] embed their voltage-island-aware placement optimization to a partitioning-based placement algorithm to minimize the number of level shifters.

Although the voltage island placement problem has been well-studied in previous literature, there is still no available solution for mixed cell-height design implementation due to the existence of "chicken-and-egg" loop between floorplan and cell height selection, additional layout constraints, and area impact of cell height choices.

### 4.1.2 Problem Formulation

Table 4.1 gives notations used in the following discussion.

**Table 4.1**: Notations used in our work.

| Term | Meaning |
|---|---|
| $h_i$ | available cell heights, ($0 \leq i \leq N$; $h_0$ is the minimum one) |
| $(X^l, Y^b, X^r, Y^t)$ | coordinates of block area (i.e., standard-cell placement region) |
| $P_j$ | partition in which cells are of the same height, ($0 \leq j \leq M$) |
| $(x_j^l, y_j^b, x_j^r, y_j^t)$ | coordinates of partition $P_j$, ($0 \leq j \leq M$) |
| $H_j$ | height of partition $P_j$, ($0 \leq j \leq M$) |
| $W_j$ | width of partition $P_j$, ($0 \leq j \leq M$) |
| $t_j$ | cell height corresponding to partition $P_j$, ($0 \leq j \leq M$) |
| $w_{site}$ | placement site pitch (width) |
| $d$ | shift of cell rows in vertical direction to avoid cell overlap |

We state the **non-integer multiple-height cell placement problem** as follows.[54]
**Given** a design (i.e., gate-level netlist), timing constraints, Liberty and technology models for cell libraries with multiple track heights, and P&R block area and aspect ratio, **place** the design such that each cell instance is legally placed in row sites with corresponding height. The **objective** of the placement is to achieve minimum design power or area while maintaining the (same) target performance.

Additional layout constraints for mixed cell-height implementation applied in our studies below are as follows.[55]

$C_1$: To ensure manufacturability, each region of a particular cell height must have at least two cell rows.

$C_2$: Due to N-well sharing, each region of a particular cell height must have an even number of rows.

$C_3$: Every region with a particular cell height must align with the block's overall metal and poly track definitions.

$C_4$: The horizontal distance between two regions of different cell heights must be no less than four placement sites.

$C_5$: The minimum vertical distance between two partitions of different cell heights must ensure that the power/ground (P/G) rail of one cell does not encroach beyond the P/G rail of another cell. (Figure 4.3 shows an example with M2 pitch = $64nm$, and P/G rail width = $48nm$ and $64nm$ for 8T and 12T cells, respectively. Although the P/G rail width difference between 8T and 12T cells is less than one M2 pitch, to align cells to routing tracks, the minimum $d$ in the example is $64nm$.)

$C_6$: "Breaker cells" must be inserted to ensure the minimum horizontal and vertical distances between two regions of different cell heights.

### 4.1.3 Methodology

We now describe our optimization methodology for mixed cell-height implementation. The overall optimization flow is shown in Figure 4.4. Given an input design (i.e., RTL netlist) and timing constraints, we first synthesize it with Liberty files of all available cell heights having

[54] Since we focus on placement with non-integer multiple-height cells, in the following discussions "mixed cell heights" or "mixed heights" refers to "mixed non-integer multiple cell heights".

[55] Our proposed approaches given below transparently handle other values of the parameters that define these constraints (e.g., minimum number of cell rows in a given-height region, or minimum separation between two different-height regions, etc.).

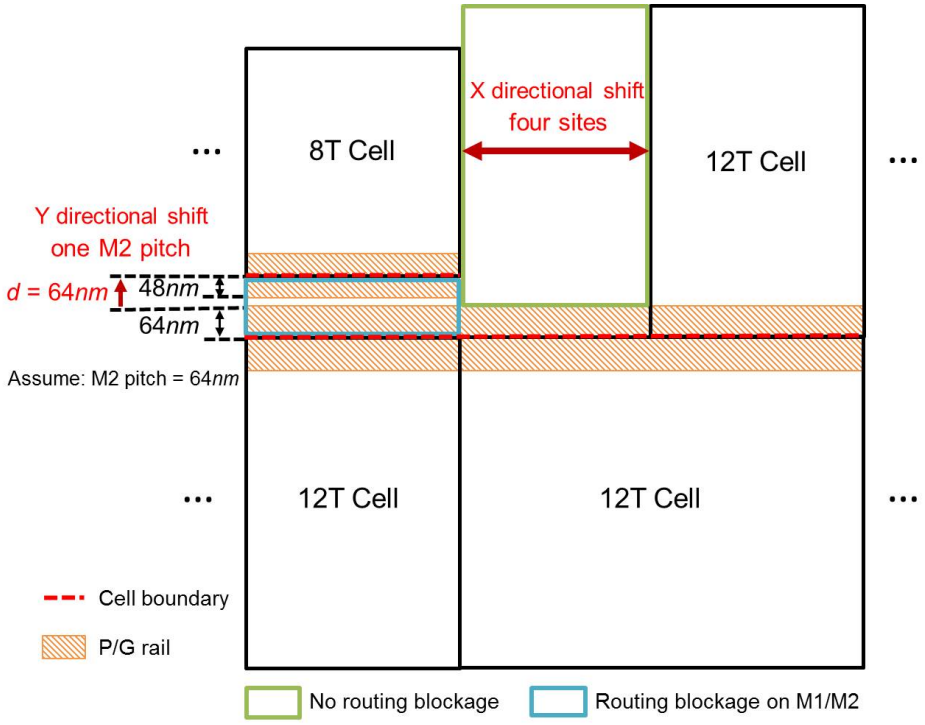



**Figure 4.3**: Area cost of "breaker cells".

been made available to the logic synthesis tool. To resolve the "chicken-and-egg" loop between floorplan and cell height selection, we modify standard-cell LEF files such that all cells have the same height (i.e., the minimum cell height among all the available heights), while maintaining the original area of each cell.[56] In the discussion below, we refer to such modified LEF files as **mLEF**. In this way, we break the "chicken-and-egg" loop and enable a commercial placement tool to "freely" place cells with timing-awareness. Since we use the original Liberty timing/power models and preserve the original area for each standard cell (although with different aspect ratio of cell layout), this placement optimization is able to comprehend the tradeoff between timing constraints, power and area overheads. As a result, timing-critical cells tend to have larger heights (i.e., larger width with **mLEF**), while non-critical cells are smaller. An example initial placement solution of design *AES* is shown in Figure 4.4(a), in which 12T cells (with **mLEF**) are in red, and 8T cells are in blue.

Based on the initial placement solution, we partition the block area into regions of particular cell heights with awareness of area cost due to "breaker cells".[57] We then legalize the

[56] In doing so, we round cell widths to the nearest whole site with no cell area reduction. E.g., given three libraries with heights 8T, 9T and 12T, (i) a 12T, 6-site cell would be represented by an 8T, 9-site cell; (ii) a 9T, 5-site cell would be represented by an 8T, 6-site cell; etc.

[57] Here cell height indicates the original cell height as opposed to the cell height in **mLEF**.

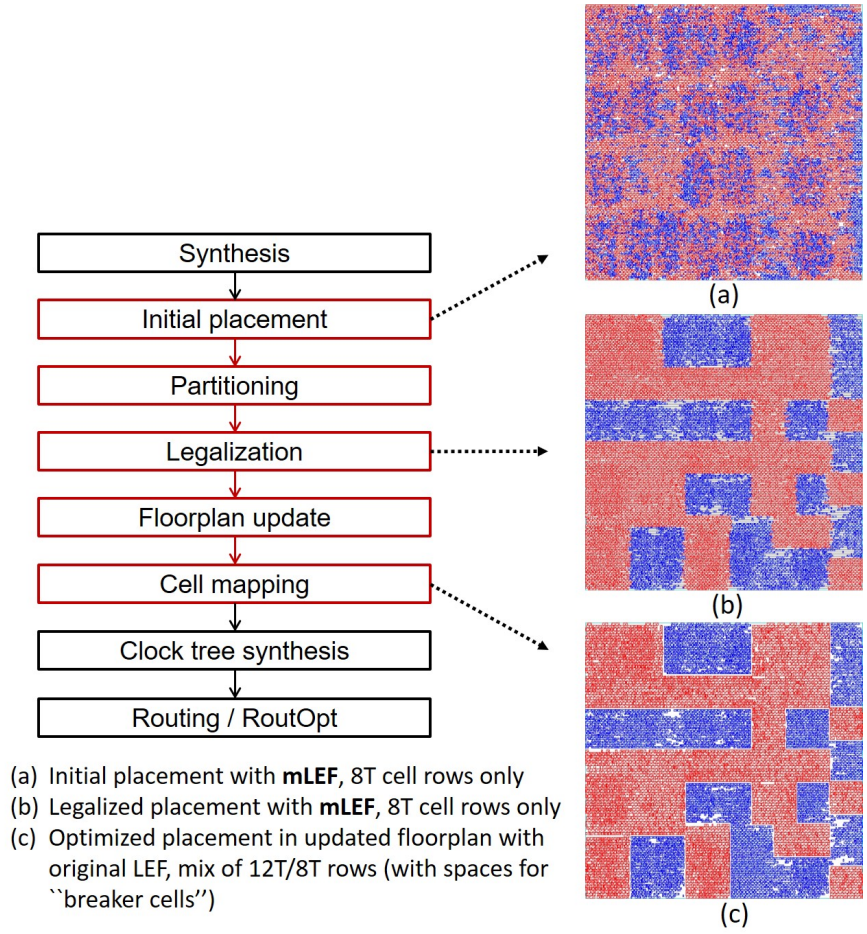



**Figure 4.4**: Overall flow of our optimization. In the example, the maximum cut number ($K$) = 30.

placement solution by (i) displacement of cells (i.e., placement perturbation) and (ii) swapping of cells across different heights (i.e., gate sizing). Here, we say that a placement solution is *legal* when each cell instance is placed in a region with the same height (e.g., as shown in Figure 4.4(b)). Once the placement solution is legal, we update the floorplan with space inserted to model the cost of breaker cells. We then map cells to the cell rows of the updated floorplan, using the original standard-cell LEF files. In the end, we perform clock tree synthesis and route the design (Figure 4.4(c)).

### Floorplan Partitioning and Region Definition

We perform slicing-based partitioning using dynamic programming to divide the block area into regions of particular cell heights. Algorithm 15 shows our partitioning procedure. We first evaluate the cost of each candidate partition, i.e., $cost(x^l, y^b, x^r, y^t, 0)$, in which the fifth

parameter indicates the number of cuts within the partition (Line 1). If the height of a candidate partition is $h_j$, the cost of the partition is calculated as

$$cost = \beta \cdot \sum_{h_i \neq h_j} area_i + \lambda \cdot \Delta power + \eta \cdot \Delta delay \tag{4.1}$$

where $area_i$ is the total area of cells with height $h_i$ located within the candidate partition; $\Delta power$ is the estimated total cell power increase by swapping cells with original cell height $h_i < h_j$ to height $h_j$; $\Delta delay$ is the estimated total cell delay increase by swapping timing-critical cells (i.e., cells with slack $< 20\%$ of the clock period) with original height $h_i > h_j$ to height $h_j$; and $\beta$, $\lambda$ and $\eta$ are weighting factors.

Reducing the value of component $\sum_{h_i \neq h_j} area_i$ (i.e., the sum of areas of cells whose heights differ from the height of the partition) will help to minimize the cost of placement legalization (i.e., displacement and cell-height swapping) as well as the perturbation to the initial placement solution. Furthermore, to reduce the potential power and timing penalties due to legalization, we minimize the estimated cell power and delay increase within each candidate partition. More specifically, we first find the best candidate library cell (with height $h_j$) for each cell instance (assume that the original height of the cell instance is $h_i$), such that the best candidate library cell has the minimum cell power without any delay increase (resp. the minimum cell delay without any power penalty) if $h_j > h_i$ (resp. $h_j < h_i$). We then estimate $\Delta power$ and $\Delta delay$ values for each candidate partition accordingly.

Furthermore, we set the cost of a candidate partition to infinity if it violates any of the constraints (e.g., Constraints $C_1$ and $C_2$) described in Section 4.1.2. More specifically, a partition $(x^l, y^b, x^r, y^t)$ with height $h_j$ must satisfy

$$y^t - y^b \geq 2 \cdot h_j \tag{4.2}$$

$$\lfloor \frac{y^t - y^b}{2 \cdot h_j} \rfloor \cdot 2 \cdot h_j \cdot (x^r - x^l) \geq (y^t - y^b) \cdot (x^r - x^l) \cdot U_j \tag{4.3}$$

Inequality (4.2) forces each partition to have at least two rows. Inequality (4.3) ensures that partitions in the updated floorplan, after rounding to an even number of rows per partition according to Constraint $C_2$, have enough sites to place cells; here, $U_j$ is the placement utilization within the partition. Finally, for each candidate partition, we set the height of the partition as the height which leads to the minimum cost function value.

Figure 4.5 shows contour maps of power and delay costs, as well as partitioning solutions with various weighting factors of design *AES*, where cell area, power and delay are respectively measured in units of $\mu m^2$, $\mu W$ and $ps$ for cost estimation. In red are 12T cells, and in

**Algorithm 15** DP-based partitioning.

calculate $cost(x^l, y^b, x^r, y^t, 0)$
$\forall X^l \leq x^l \leq x^r \leq X^r, Y^b \leq y^b \leq y^t \leq Y^t$
**for** $k := 1$ to $K$ **do**
  **for** $x^l := X^l$ to $X^r - \Delta x$ **do**
    **for** $y^b := Y^b$ to $Y^t - \Delta y$ **do**
      **for** $x^r := x^l + \Delta x$ to $X^r$ **do**
        **for** $y^t := y^b + \Delta y$ to $Y^t$ **do**
    $cost(x^l, y^b, x^r, y^t, k) = \min_{x^l \leq \times \leq x^u, y^b \leq y \leq y^t}$ (
$cost(x^l, y^b, x, y^t, k') + cost(x, y^b, x^r, y^t, k'') + 4 \cdot w_{site} \cdot (y^t - y^b)$,
$cost(x^l, y^b, x^r, y, k') + cost(x^l, y, x^r, y^t, k'') + d \cdot (x^r - x^l)$
) $\forall k',\ k''\ s.t.\ k' + k'' = k - 1$
        **end for**
      **end for**
    **end for**
  **end for**
  **if** $cost(X^l, Y^b, X^r, Y^t, k) \geq cost(X^l, Y^b, X^r, Y^t, k-1)$ **then**
    **return** $cost(X^l, Y^b, X^r, Y^t, k-1)$
  **end if**
**end for**
**return** $cost(X^l, Y^b, X^r, Y^t, K)$

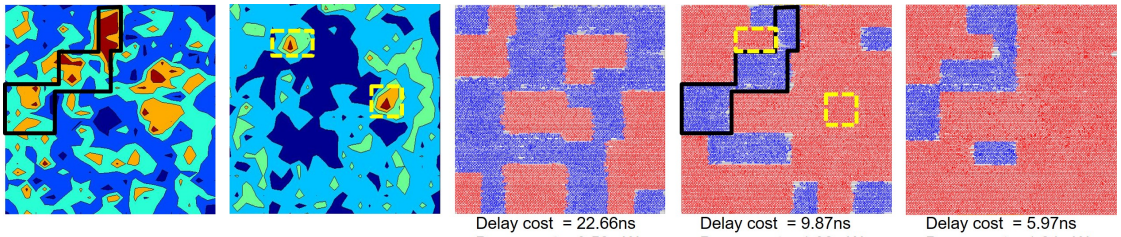



**Figure 4.5**: (a) Contour map of power cost function. (b) Contour map of delay cost function. (c) Partitioning solution with $\beta = 1$, $\lambda = 0.8$, $\eta = 0.2$. (d) Partitioning solution with $\beta = 1$, $\lambda = 0.7$, $\eta = 0.3$. (e) Partitioning solution with $\beta = 1$, $\lambda = 0.6$, $\eta = 0.4$. Design: *AES*. Technology: $28nm$ LP.

blue are 8T cells.[58] As the value of $\lambda$ decreases and the value of $\eta$ increases, the area of 12T regions and power both increase, while timing slack improves. In other words, by comprehending power and timing penalties, our partitioning optimization defines the height of regions with large power penalty as 8T (e.g., region defined by black boundaries) and the height of regions

[58] We measure area, power and delay in units of $\mu m^2$, $\mu W$ and $ps$ for cost function estimation. A small value of $\beta$ will result in large perturbation to the initial placement, such that the power, area and timing costs due to legalization can be high. On the other hand, a large value of $\beta$ will limit the timing- and power-awareness in the partitioning optimization. We empirically use $\beta = 1$ and vary the values of $\lambda$ and $\eta$ between 0 and 1 to explore the tradeoff between power and timing during partitioning optimization.

with large timing penalty as 12T (e.g., regions in yellow-dotted boxes). In our experiments, we empirically set ($\beta$, $\lambda$, $\eta$) as (1, 1, 1), (1, 0.1, 1), (1, 1, 0.1), (1, 0.1, 0.1) and select the outcome with minimum power that satisfies timing constraints.

The heart of the dynamic programming recurrence (i.e., in determining the partitioning solution with minimum cost) is given in Lines 2-16. We recursively search for the minimum-cost partitioning solution of a rectangular region with $k$ cuts, and increase the value of $k$ in each iteration up to a given maximum allowable number of cuts, $K$, which is a user-defined parameter.[59] Figure 4.6 and its caption tell us that the total cost within partitions reduces as the number of cuts increases. However, the area cost of breaker cells increases with the number of cuts. We therefore sweep the number of cuts during our partitioning optimization and select the solution with the minimum total cost.

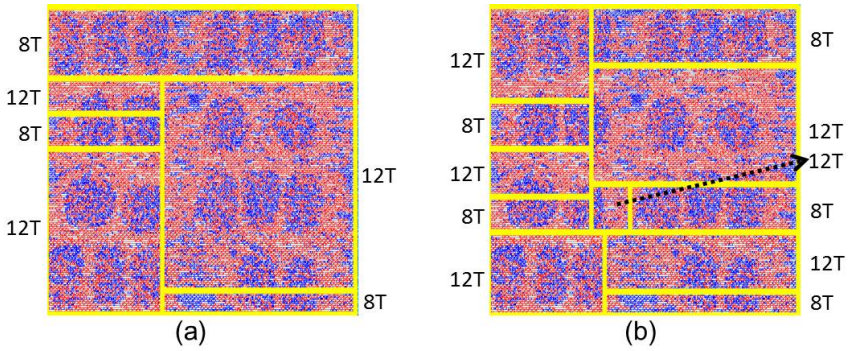



**Figure 4.6**: Examples of partitioning solutions for the *AES* testcase. In red are 12T cells (with **mLEF**); and in blue are 8T cells. Yellow lines are cuts. The cell height of a partition is marked on its side. $\beta$ = 1, $\lambda$ = 0 and $\eta$ = 0. (a) Cut number = 5, cost = 4818$\mu m^2$. (b) Cut number = 10, cost = 4584$\mu m^2$.

To find the best partitioning solution of a region $(x^l, y^b, x^r, y^t)$ using exactly $k$ cuts, we observe that such a solution can always be seen as a single "top-level" cut, along with the best solutions of the two sub-regions induced by that cut. Hence, to find the best $k$-cut solution, we enumerate all potential vertical and horizontal cuts of the region, and select the solution that minimizes the sum of the costs of the two separate parts (sub-regions) – with respective number of cuts $k'$ and $k''$ satisfying $k' + k'' = k - 1$ – plus the cost of the single vertical or horizontal "top-level" cut. Note that the proposed partitioning comprehends the area cost of breaker cells, for which width = $4 \cdot w_{site}$ for a vertical cut, and height = $d$ for a horizontal cut (Line 7). For the example shown in Figure 4.3, $d$ must be larger than 64$nm$. The procedure terminates when the cost does not decrease with an increased cut number (Line 13), or the maximum cut number $K$ is

[59] In our experiments, we set $K$ to a large value (e.g., 30 for a 100$\mu m$ × 100 $\mu m$ floorplan) in order to ensure good solution quality.

achieved (Line 16). To improve the scalability, we divide the block area into $M \times N$ grids (where $M$ and $N$ are also user-defined parameters), and perform the proposed partitioning method on these grids. The runtime complexity of the procedure is $O((M+N)(M \cdot N \cdot K)^2)$.[60]

**Timing-Aware Placement Legalization**

Based on the partitioning solution, we perform iterative optimization to achieve a legal placement. Note that we still use **mLEF** at this optimization stage, but boundaries and cell heights of regions have been defined. We apply two knobs in our iterative heuristic: *displacement of a cell (e.g., moving a 12T cell from an 8T region to a 12T region)*, and *cell-height swapping* (e.g., assign an 8T cell to a 12T cell master in a 12T region via gate sizing). Both of these knobs affect timing, and cell-height swapping also affects area. Thus, to ensure that the optimization does not lead to large design quality degradation, we evaluate the timing and area impacts of each potential move (one move is a cell displacement or a cell-height swap).

Because timing analysis with commercial P&R tools is typically slow, our optimization approach requires a relatively accurate and fast timing engine. We have developed an internal timing analysis engine (i.e., *internal timer*) to guide the optimization. Our internal timer estimates gate delay and slew at an output pin based on the Liberty lookup tables. It further uses D2M [7] and PERI [118] models that respectively estimate wire delay and slew propagation along the interconnect. Wirelength change due to cell displacement is measured by net HPWL (Half-Perimeter Wire Length), and wire capacitance and resistance are scaled correspondingly.

To comprehend wire congestion effects, we add a penalty in the form of wire resistance and capacitance scaling, based on routing demand vs. supply overflows within the bounding box of a given net.[61] More specifically, if the average horizontal (resp. vertical) routing congestion within the bounding box of a net is $X\%$, we penalize the horizontal (resp. vertical) portion of HPWL by a multiplicative factor of ($X\%$-$X_{th}\%$) whenever $X > X_{th}$. Here, $X_{th}\%$ is a threshold that we set to 95% based on separate studies. The value $X_{th}\% = 95\%$ is used in all experiments reported below.[62] To maintain the accuracy of our internal timer, we correlate timing slack, wire capacitance and overflow information during the optimization through a Tcl socket with *Cadence Innovus Implementation System v16.1* [215]. Figure 4.7 shows our optimization framework. We

[60] In our experiments, partitioning with number of grids no larger than 30 $\times$ 30, and maximum cut number no larger than 40, requires less than one minute of a single thread on a 2.5$GHz$ Intel Xeon server.

[61] We estimate overflow based on the trial routing solution from *Cadence Innovus Implementation System v16.1* [215].

[62] For example, if the average horizontal congestion is 98%, we multiply the $x$-component of HPWL by 1.03 = 0.98 / 0.95.

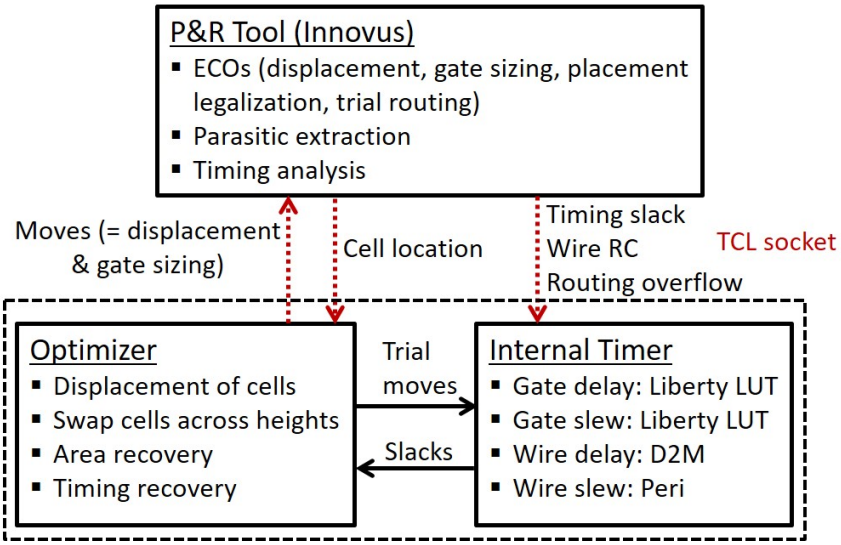



**Figure 4.7**: Framework of our optimization.

believe that our internal timer approach most closely resembles that of the previous work [107]; however, our internal timer better comprehends the impact of cell displacement on timing by considering both wirelength change and routing congestion information.

Algorithm 16 describes our heuristic to legalize the placement. We first evaluate the cost (in terms of area and timing) of each potential move (i.e., cell displacement or swapping) (Line 4). We consider cell displacement in eight directions (i.e., {N, S, E, W, NE, NW, SE, SW}) with the maximum movement distance of $D$ ($D$ = 15$\mu m$ in our experiments). The set of candidate cell displacements is similar to what is applied in the local optimization of [83]. For cell-height swapping, we consider candidate library cells whose heights match that of the partition. We use the cost function shown in Equation (4.4)

$$cost = \alpha \cdot \frac{\max(0, -\Delta\text{slack})}{\max(1ps, \text{slack}_{orig})} + (1-\alpha) \cdot \frac{\max(0, \Delta\text{area})}{\max(1\mu m^2, \text{whitespace}_{orig})}$$

where $\Delta$slack and $\Delta$area are respectively the timing slack and cell area changes due to displacement and/or swapping. slack$_{orig}$ and whitespace$_{orig}$ are the original timing slack of the cell and whitespace of the corresponding grid. We divide the block area into an $M \times N$ mesh of grids. For each grid, we estimate whitespace based on placement utilization. The parameter $\alpha$ is a weighting factor, which has an initial value of 0.5. We adaptively change the value of $\alpha$ for each cell during the iterative optimization, such that when an attempt leads to timing violation (resp. placement utilization violation), we increase (resp. decrease) $\alpha$ of the cell by 1.5×.

The cost function (4.4) only considers area and timing impacts due to each move. In

separate studies, we have also included input pin and wire capacitance, as well as leakage power, into our cost function for legalization moves. However, the resultant solutions show negligible improvement (e.g., $< 1\%$) in terms of power, area and timing. This might be due to the positive correlations among area, pin capacitance and leakage power. Furthermore, we adaptively change the value of $\alpha$ during our optimization. As a result, we observe from our experiments that moves with small area and power costs are typically selected during the early steps. As the value of $\alpha$ increases, moves with small delay penalties are selected during the late-stage legalization.

We sort all cells which have different height than their partition in decreasing order of cost, and apply moves to legalize the placement (Line 7). When a move results in timing failure or violation of placement density, we undo the move (Lines 12-13); here $\omega$ is the required whitespace according to the area of breaker cells and maximum placement density constraints.[63] To ensure the convergence of the flow (i.e., that optimization can lead to a legalized placement), we commit the move of a cell which has been visited $F$ times, regardless of its impact on timing and area. We use $F$ = 5 in our optimization.[64] In addition, we apply a form of Tabu search [75] during the optimization to increase the likelihood of finding feasible solutions for cells. Specifically, we record the latest three attempts and forbid these moves for the current move of optimization. During the optimization, we (re-)correlate our internal timer with *Innovus* in terms of timing slack/slew, cell location, wire parasitic, and routing overflow after every $\gamma\%$ of the total number of cells has been changed (Lines 17-19). We use $\gamma$ = 2 in our optimization. We also include area recovery (Lines 20-26) and timing recovery (Lines 27-32) in our optimization to maintain timing and area quality. The parameter $\theta$ is a threshold of slack violation that triggers timing recovery; we empirically set this to 0.15. Note that during the timing recovery, we perform backward (in which we downsize fanout cells) and forward (in which we upsize cells) maximum transition violation fixes, which enhance the timing recovery quality.

We observe from our experimental results that the ratio between the number of cells being swapped and the number of cells being displaced ranges from 1.2 to 5.4. This ratio seems highly dependent on the partitioning solution, timing constraints, netlist structure, etc. For instance, fewer partitions and/or tighter timing constraints can lead to more swaps relative to displacements.

We list all the user-defined parameters applied during partitioning and placement legalization in Table 4.2.

[63] In our experiments, we set the maximum placement density of the entire block as the placement density from the initial placement plus 5%.

[64] We observe in our experiments that the number of cells which have been visited six times without a feasible solution is quite small, e.g., less than 60 in a design with 15K cells.

**Algorithm 16** Heuristic to legalize placement.

```
1: while there exists a cell with a different height than its partition do
2:   list ← ∅
3:   for all cell g with a different height than its partition do
4:     calculate cost function of g
5:     add g to list
6:   end for
7:   sort list in order of decreasing cost
8:   swap_cnt ← 0
9:   for all g ∈ list do
10:      apply displacement/swapping based on cost function
11:    incremental timing analysis
12:    if slack of g < Min(0, original slack of g) || whitespace_of_grid < ω then
13:      undo change
14:    else
15:      ++swap_cnt
16:    end if
17:    if swap_cnt ≥ γ · total_gate_count then
18:      apply ECOs in Innovus
19:      correlate internal timer with Innovus
20:      for all cell g in the design do
21:        downsize g
22:        incremental timing analysis
23:        if slack of g < Min(0, original slack of g) then
24:          undo change
25:        end if
26:      end for
27:      if WNS ≤ −θ · clock_period then
28:        fix maximum transition violations
29:        timing recovery
30:        apply ECOs in Innovus
31:        correlate internal timer with Innovus
32:      end if
33:    end if
34:  end for
35: end while
```

### Mapping from mLEF to Original LEF in Assigned Regions

As discussed above (e.g., in the context of Figure 4.4), we use **mLEF** with adjusted aspect ratios for cell layouts during the initial placement, partitioning and legalization stages, where all cells have the same height (i.e., the minimum cell height $h_0$) but scaled cell widths. When the placement solution is legalized (i.e., each cell instance is placed in a region with the same height), we update the floorplan to insert cell rows according to the *actual* cell height of each partition. We also allocate space to model the area cost of breaker cells in the updated floorplan.

**Table 4.2**: User-defined parameters.

| Term | Meaning |
|---|---|
| $\beta$, $\lambda$, $\eta$ | weighting factors used in partitioning cost function |
| $K$ | maximum number of cuts |
| $M \times N$ | number of grids |
| $D$ | maximum displacement distance |
| $\gamma$ | determines correlation frequency |
| $\theta$ | slack violation tolerance threshold |
| $F$ | maximum number of visits of a cell before a move is applied |

Cells with the minimum height have their original layout aspect ratios in **mLEF**, and therefore no aspect ratio changes are needed to map these cells in the updated floorplan. In a partition with the minimum cell height, we only remove overlaps between cell instances and breaker cells. In other words, we perform *cross-row cell displacement* to ensure that the total cell width (i.e., total number of placement sites) in a row does not exceed the available number of placement sites with the existence of breaker cells (which are modeled as spaces in our experiments).

Algorithm 17 describes our cross-row cell displacement procedure. We perform four iterations of cross-row cell displacements. The first and the third iterations are top-to-bottom, traversing from the topmost row to the bottommost row, and optimizing one row at a time. The second and the fourth iterations optimize similarly, but in a bottom-to-top manner. Furthermore, in the first and second iterations, we move cells to adjacent rows only if there is enough space in the adjacent rows. In the third (resp. fourth) iteration, we force cells to move to the lower (resp. upper) adjacent row regardless of available space in the adjacent row. This strategy enables chained moves of cells across rows. In Algorithm 17, $k$ is the row index, which ranges from 1 (the bottommost row) to $R$ (the topmost row); $W_i$ is the total cell width in the $i^{th}$ row; $W_i^{max}$ is the maximum allowed cell width in the $i^{th}$ row; $w(g)$ is the width of cell $g$; $\Delta$ records the HPWL increase due to cross-row cell displacement; and function $move(g,\ dir)$ moves cell $g$ to the row that is adjacent in direction $dir$. After the cross-row cell displacements, we perform single-row incremental placement optimization using dynamic programming to further reduce wirelength.[65]

On the other hand, cells originally with large height (i.e., larger than the minimum cell height) become shorter and wider in **mLEF**. We must recover the original aspect ratio of cell

[65] Our dynamic programming formulation is the same as the "Minimum HPWL" formulation in [114]. Formulation details are given in [114].

**Algorithm 17** Cross-row cell displacement.

```
1: for k := 1 to 4 do
2:   for all i^th row in the partition do
3:     while i^th row has capacity/overlap violation do
4:       Δ_min ← +∞;  g_move ← ∅
5:       for all g ∈ i^th row do
6:         if i ≠ 1 && (W_{i-1} + w(g) ≤ W_{i-1}^max || k == 3) then
7:           Δ ← HPWL increase by moving g to (i − 1)^th row
8:           if Δ < Δ_min then
9:             Δ_min ← Δ;   g_move ← g;   dir = down
10:          end if
11:        end if
12:        if i ≠ R && (W_{i+1} + w(g) ≤ W_{i+1}^max || k == 4) then
13:          Δ ← HPWL increase by moving g to (i + 1)^th row
14:          if Δ < Δ_min then
15:            Δ_min ← Δ;  g_move ← g;   dir = up
16:          end if
17:        end if
18:      end for
19:      if g_move == ∅ then
20:        break
21:      else
22:        move(g_move, dir)
23:      end if
24:    end while
25:  end for
26: end for
```

layouts in the updated cell rows with actual cell heights. For example, assume that there are 20 10T cells uniformly placed in five 8T cell rows (i.e., as a $5 \times 4$ mesh). To update the floorplan, we maintain the same partition area and place cell rows according to the height of the partition. We therefore have four 10T cell rows. Given that the layout of these 10T cells (with the same cell area) are scaled back to their original height with a reduced cell width, five cells now can fit into one row in the updated floorplan. The mapped cell placement becomes a $4 \times 5$ mesh. As shown in the example, cell mapping in the updated floorplan can be viewed as embedding a graph to another graph with a different aspect ratio (e.g., embed a $5 \times 4$ mesh to a $4 \times 5$ mesh). We therefore revisit the graph embedding literature.

Ellis [59] shows that to embed a 2D mesh of size $w \times h$ (with unit distance between every two adjacent nodes in both horizontal and vertical directions) to another 2D mesh of size $w' \times h'$, where $w' < w$ and $h'$ is the smallest integer satisfying $w' \cdot h' \geq w \cdot h$, if $\frac{w}{w'}$ is no larger than 2, the maximum wirelength of a two-pin net (in Manhattan distance) in the embedded graph is no more than two units. An example with $\frac{w}{w'} = \frac{5}{4}$ is shown in Figure 4.8(a): the wirelength

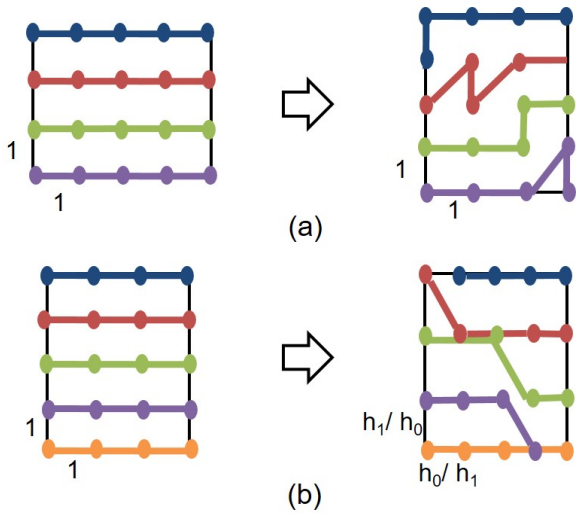



**Figure 4.8**: Illustration of graph embedding (a) from [59], and (b) for proposed cell mapping. Vertical connections are not shown.

of each connection in the original graph is one, and the maximum wirelength in the embedded graph (i.e., the diagonal connection) is two. Note that our optimization of cell mapping differs from [59], in that [59] varies the area of the graph (i.e., mesh) while our optimization assumes a fixed mesh area (i.e., area of a partition).

Following the discussions in [59], we can show that if we map a 2D-mesh placement with cell height $h_0$, in which all cells have the same cell area, to another 2D-mesh placement with cell height $h_1$, the maximum wirelength scaling of a mesh edge (i.e., two-pin net) according to the mapping is no more than $\frac{h_0}{h_1} + \frac{h_1}{h_0}$.[66] Figure 4.8(b) shows an example with 10T and 8T cells. Assuming unit wirelength for each two-pin connection between any horizontally or vertically adjacent cells in the original 2D-mesh placement, the maximum wirelength increase is 1.05.

Inspired by the graph-embedding theory, we propose an approach to map cells to cell rows with original cell heights for general cases, in which cells have different widths and are not necessarily placed in a 2D mesh. Algorithm 18 shows our procedure to map cells from an initial floorplan with $R$ rows of height $h_0$ to an updated floorplan with $R'$ rows of height $h_j$. We first estimate the average total cell width of each cell row in the updated floorplan (Line 1), in which $g$ is a cell in partition $P_j$; $w(g)$ is the actual width of cell $g$ corresponding to height $h_j$. We then store cells in the $i^{th}$ row from the initial floorplan into $list_1$ and sort them by increasing order of their X-coordinates (Lines 5-6). For each cell on the $(i+1)^{th}$ row of the initial floorplan, we estimate the wirelength (i.e., HPWL) difference between the case of assigning the cell to the $i'^{th}$ row of the updated floorplan versus the case of assigning the cell to the $(i'+1)^{th}$ row of

[66] Proof details are given in [59].

the updated floorplan (Lines 9-12). We then sort the cells on the $(i+1)^{th}$ row of the initial floorplan by the corresponding delta wirelength values (Line 13). We iteratively add these cells to $list_2$ until the total width of cells in $list_1$ and $list_2$ exceeds 1.05$\times$ of the average total cell width of each row ($W_{avg}$) or the maximum allowed total width in the $i'^{th}$ row ($W_{i'}^{max}$, where the width of breaker cells is considered) (Lines 14-18). If all cells from $list'$ are added to $list_1$ and $list_2$, we fill $list'$ with cells from the next row in the initial floorplan (Lines 21-22). Finally, we use dynamic programming to order and place cells from $list_1$ and $list_2$ onto the $i'^{th}$ row in the updated floorplan. In summary, Lines 1-23 of Algorithm 18 determine Y-coordinates of cells and optimize the vertical component of HPWL; while the $DPPlacer(i', list_1, list_2)$ further minimizes the horizontal component of HPWL. As an improvement to [92] which also uses dynamic programming to optimally order two rows of cells into a single row with minimized wirelength, our formulation also determines the optimal cell locations. The recurrence relation in our dynamic programming optimization is

$$sol(i,j,k) = Min \begin{cases} sol(i,j,k-1) \\ sol(i-1,j,k-w(g_i)) + cost(g_i,k) \\ sol(i,j-1,k-w(g_j)) + cost(g_j,k) \end{cases}$$

where $sol(i,j,k)$ is the wirelength corresponding to the optimal placement solution of the first $i$ cells in $list_1$ and the first $j$ cells in $list_2$ within the first $k$ placement sites in the updated cell row; $g_i$ is the $i^{th}$ cell in $list_1$; $g_j$ is the $j^{th}$ cell in $list_2$; $w(g)$ is cell width (i.e., in terms of the number of sites) of $g$; and $cost(g,k)$ is the HPWL increase by allocating $g$ (i.e., right edge of $g$) at the $k^{th}$ placement site.

Figure 4.9 shows the optimized wirelength (i.e., HPWL) comparison between our proposed method (using dynamic programming) versus a greedy method proposed in [55]. We observe that our proposed optimization achieves up to 16% wirelength reduction compared to the greedy method. Furthermore, it is obvious that the proposed algorithm can achieve the mapping solution or a solution with the same total wirelength shown in Figure 4.8(b). We note that the procedure described in Algorithm 18 only applies to the case where the ratio between $h_j$ and $h_0$ is no larger than two. We can easily extend our mapping procedure to address cases with height ratio greater than two by extending our dynamic programming formulation to optimize more than two lists of cells.

**Algorithm 18** Cell mapping.

1: $W_{avg} = (\sum_{g \in P_j} w(g)) / R'$
2: $i = 1$
3: $list' \leftarrow$ cells in $i^{th}$ row from the initial floorplan
4: **for** $i' := 1$ to $R'$ **do**
5: $list_1 \leftarrow list'$
6: sort $list_1$ in order of increasing cells' X-coordinate
7: $W \leftarrow$ total width of $list_1$
8: $++i$
9: **for all** cell $g$ in $i^{th}$ row from the initial floorplan **do**
10: $\Delta(g) \leftarrow HPWL\_diff(g, i', i'+1)$
11: $list'.push(g)$
12: **end for**
13: sort $list'$ in order of increasing $\Delta(g)$
14: **while** $list' \neq \emptyset$ && $W \leq Min(1.05 \cdot W_{avg},\ W_{i'}^{max})$ **do**
15: $g \leftarrow list'.pop()$
16: $list_2.push(g)$
17: $W \leftarrow W + w(g)$
18: **end while**
19: sort $list_2$ in order of increasing cells' X-coordinate
20: **if** $list' == \emptyset$ **then**
21: $++i$
22: $list' \leftarrow$ cells in $i^{th}$ row from the initial floorplan
23: **end if**
24: $DPPlace(i', list_1, list_2)$
25: **end for**

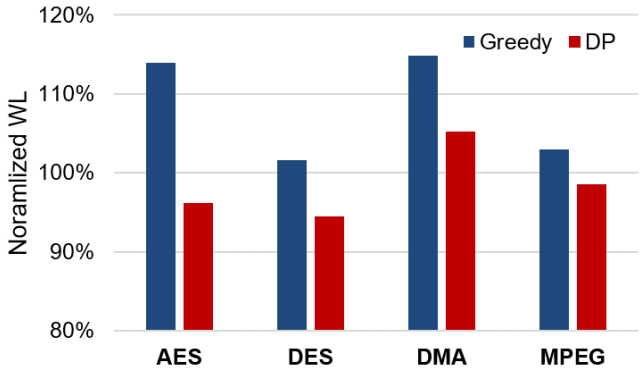



**Figure 4.9**: Wirelength comparison between our dynamic programming-based optimization versus a greedy optimization in [55]. Wirelength values are normalized to the wirelength before cell mapping.

### 4.1.4 Experimental Results

We perform experiments in a $28nm$ LP foundry technology with dual-$V_{th}$ libraries, $0.95V$ nominal supply voltage, and cell height choices 12T and 8T. To confirm that our optimization can perform a fine-grained mixed cell-height implementation, we select four design

blocks (*AES*, *DES*, *JPEG*, *MPEG*) from the *OpenCores* website [230]. Parameters of these four testcases are shown in Table 4.3. For each design, we determine a range of clock periods starting from a clock period with relative loose timing constraint, up to the clock period at which the 8T-only implementation shows setup timing violations. These designs are synthesized using *Synopsys Design Compiler vI-2013.12-SP3* [237] and then placed and routed using *Cadence Innovus Implementation System v16.1* [215]. We set the gate density at the floorplan stage as 60%. We respectively use *Cadence Innovus Implementation System* and *Synopsys PrimeTime-PX vH-2013.06-SP3 (PT-PX)* [240] for timing and power analysis at the post-routing stage (with ideal clocks) and wire parasitics (SPEF) obtained from Innovus. We use *Synopsys PrimeTime vH-2013.06-SP2* [240] to search for the minimum supply voltage that satisfies a given frequency target. Our optimization flow is implemented in C++. Functions used in P&R tools and the socket between our optimizer and the P&R tool are implemented in Tcl. We conduct our experiments on a 2.5$GHz$ Intel Xeon server.

**Table 4.3**: Benchmarks.

| Design | #Instances | #Flip-flops | Clock period range |
|---|---|---|---|
| *AES* | ∼16K | 530 | 650$ps$ – 800$ps$ |
| DES | ∼23K | 1984 | 600$ps$ – 750$ps$ |
| *JPEG* | ∼60K | 4512 | 750$ps$ – 900$ps$ |
| *MPEG* | ∼16K | 3193 | 550$ps$ – 700$ps$ |

**Modeling breaker cell costs.** The placement site pitch (width) and the M2 metal pitch in the 28$nm$ LP technology that we use are respectively 0.136$\mu m$ and 0.1$\mu m$. Based on the discussion in Section 4.1.2, the horizontal and vertical shifts between any 8T and 12T regions must be no less than 0.544$\mu m$ and 0.1$\mu m$, respectively. In addition, to preserve cell row alignment in the design, we shift cell rows by 0.8$\mu m$ in the vertical direction between any 12T and 8T regions.[67] We also insert placement and routing blockages correspondingly. Figure 4.10 shows one layout example.

### Performance-Area Tradeoff Comparison

We implement our benchmark designs using our proposed flow with mixed 8T/12T cells. We also perform conventional SP&R (synthesis, placement, clock tree synthesis and routing)

[67] We have separately observed in our experiments that slight decrease or increase of the breaker cell cost leads to negligible power and area impacts (i.e., $< 2\%$). However, increase the breaker cell cost by more than $2\times$ can result in severe routing and placement congestion where floorplan resizing is needed.

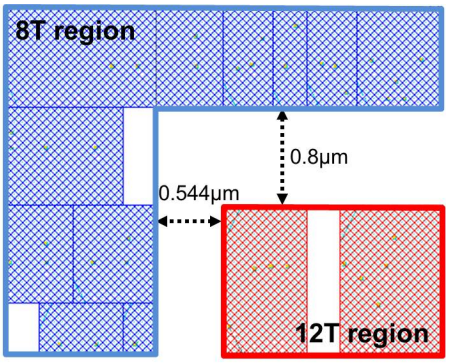



**Figure 4.10**: Inserted space on the boundaries between 12T and 8T regions to model the cost of breaker cells.

with 12T-only cells and 8T-only cells for comparison. The designs are implemented with clock periods shown in Table 4.3. We use the nominal voltage $0.95V$ at (SS, $125°C$) corner for design implementation and timing analysis. Table 4.4 shows our experimental results, in which the clock period is the clock period used for implementation and each benchmark design is implemented in four clock periods. Total power values are reported at the signoff frequency. We divide the block area of each design into grids of size around $6\mu m \times 6\mu m$ for partitioning and placement density evaluation. Figure 4.11 further shows the Pareto curves illustrating tradeoffs between performance and area of implemented designs at the post-routing stage, where the frequency given is the maximum achievable operating frequency.

Results show that by mixing 8T and 12T cells, our optimization achieves significant area reduction (e.g., 20+ percent on design *AES*) compared to designs with only single-height cells, especially for comparison to 12T-only designs (e.g., 30+ percent area reduction on design *AES*).[68] This is because mixed cell heights provide a wider range of tradeoff between performance and area such that 8T cells are applied to timing paths with large slacks for area reduction, and 12T cells are used in timing-critical paths to meet timing constraints. Furthermore, with loose timing constraints, the area benefit of mixed cell-height designs over 8T-only designs reduces. On the other hand, mixed cell heights also have similar or even higher maximum achievable performance compared to designs with only single-height cells. For instance, we observe up to 13% performance improvement from mixed cell heights over 8T-only designs (i.e., on design *AES*). This is because the maximum achievable performance of an 8T-only design is limited by the weak drive strengths of 8T cells. Moreover, mixed cell heights are able to have

[68] Note that our mixed-height optimization on design *AES* (clock period = $800ps$) results in a 8T-only design, but with 27% area reduction compared to that of the 8T-only implementation. This might be due to pessimism and a large number of inserted buffers during the synthesis stage of the 8T-only flow.

more compact area, which reduces wire capacitance, as well as smaller pin capacitance (i.e., by using 8T cells) on non-timing critical fanouts of a timing-critical driver (which is typically a 12T cell) compared to 12T-only designs.

Experimental results also show that our optimization with mixed cell heights offers comparable routed wirelength compared to 8T-only designs and smaller routed wirelength compared to 12T-only designs, which is mainly due to our wirelength-aware cell mapping and reduced total cell area (i.e., a more compact layout). Furthermore, in our experimental flow we use command *ccopt_design* from *Innovus* to perform clock tree synthesis. Results show that our optimized designs with mixed cell heights have similar clock tree metrics (i.e., total buffer area and clock tree wirelength) compared to designs with only single-height cells. This indicates that our optimization does not incur any power and area penalties with respect to the clock tree synthesis optimization. On the downside, we observe in our experiments that including more libraries (i.e., with mixed cell heights) typically increases the runtime of commercial SP&R tools. Furthermore, although we apply our internal timer during the placement legalization, the incremental timing analysis, placement legalization and trial routing used for correlation also incur runtime overhead. Therefore, our mixed-height implementations are achieved at the cost of larger runtimes compared to single-height cases.

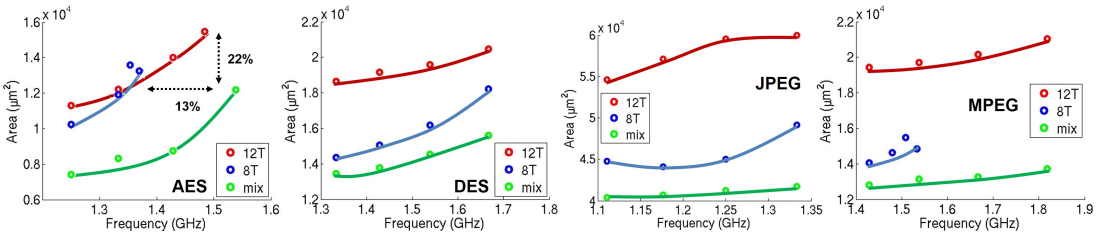



**Figure 4.11**: Pareto curves of performance-area tradeoff for implementations with 8T-only, 12T-only and mixed cells.

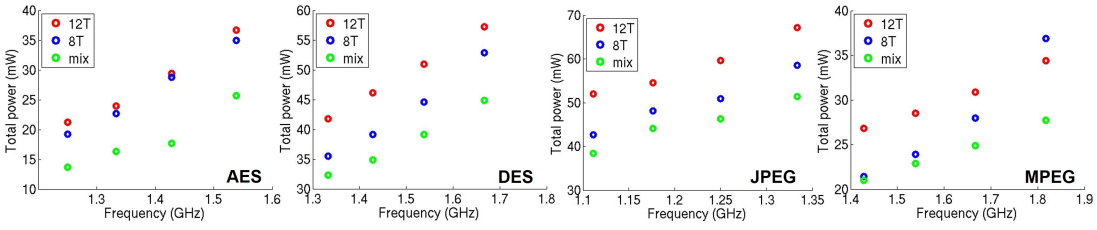



**Figure 4.12**: Iso-performance power comparison with voltage scaling among implementations with 8T-only, 12T-only and mixed cells.

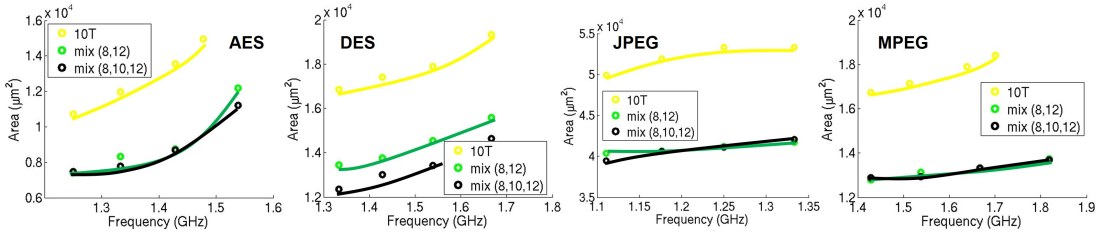



**Figure 4.13**: Pareto curves of performance-area tradeoff for implementations with 10T-only and mixed (8T and 12T) cells.

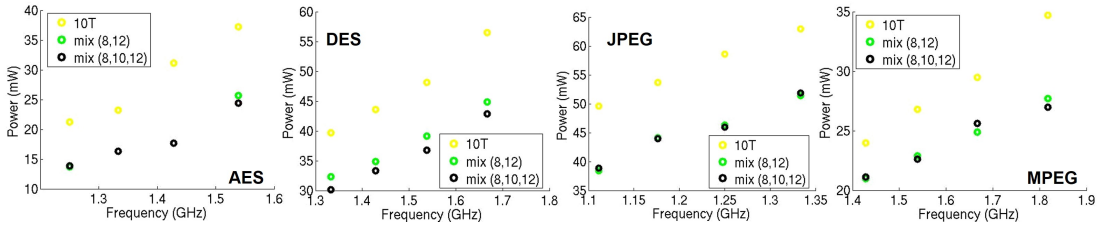



**Figure 4.14**: Iso-performance power comparison with voltage scaling among implementations with 10T-only and mixed (8T and 12T) cells.

### Iso-Performance Power Comparison

Given that certain designs have timing violations, to achieve a fair power comparison we perform voltage scaling on each design so that all designs meet the timing constraints. We then compare power at the scaled supply voltage. In our experiments, we define *scaling_lib_group* in the *PT-PX* tool to enable such comparisons. Note that to compensate the slack discrepancy between *Innovus* and *PrimeTime*, we apply a constant slack shift (i.e., the difference between the WNS from *PrimeTime* versus the WNS from *Innovus*) of the entire block to correlate the post-routing worst slack values, then perform voltage scaling. When the difference between the scaled voltage and the signoff voltage is larger than $30mV$, we perform SP&R with the scaled voltage and use the smaller power value between that of the initial implementation and that of the additional implementation in our comparison.

Figure 4.12 shows the iso-performance power comparison. We observe that 8T-only designs typically have smaller power compared to 12T-only ones. The exception of design *MPEG* with frequency = $1.8GHz$ might be due to a larger number of buffer insertion as well as voltage scaling in the 8T-only design to meet the performance constraints. Moreover, our optimized designs with mixed cell heights provide power reduction compared to 8T-only and 12T-only designs. Such power reduction mainly comes from reduced cell area and wirelength, as well as power-awareness in our partitioning optimization. Similarly to area benefit, power benefit from mixed heights over 8T-only designs also reduces at a loose timing constraint.

**Comparison to 10T-Only Designs**

We generate 10T cell libraries by performing interpolation on timing and power tables of 12T and 8T libraries. We also generate cell LEF by scaling area of cells proportional to cell drive strengths according to area and drive strength information of 8T and 12T cells. Therefore, 10T cells offer averaged performance-area and performance-power tradeoffs of 8T and 12T cells. Figure 4.13 and Figure 4.14 respectively show performance-area Pareto curves and iso-performance comparisons between 10T-only designs and mixed-height designs with 12T and 8T cells. We observe significant power and area reductions from the optimized designs with mixed heights over the 10T-only designs. This indicates that single-height designs with an optimized performance-power/area tradeoff are not able to provide the similar performance, power and area benefits compared to mixed cell-height designs which are able to explore a wider range of tradeoff among performance, power and area.[69] Moreover, we perform mixed-height optimization with 8T, 10T and 12T cells as an example to demonstrate the scalability of our methodology to more than two cell heights. The black curves and dots in Figure 4.13 and Figure 4.14 respectively show performance-area and performance-power tradeoffs of the optimized design with three cell heights. Results show similar design quality between the mixed-height solutions with 8T and 12T cells versus the solutions with 8T, 10T and 12T cells for most of the designs. On design *DES*, adding 10T cells reduces power and area. The slight power increase on design *MPEG* (frequency = $1.66MHz$) might come from the noise of SP&R tools as well as our optimization.

### 4.1.5 Conclusion

In this work, we have proposed a novel physical design optimization flow (which includes synthesis, placement, clock tree synthesis and routing) to mix cells with different, non-integer multiple heights in a fine-grained manner within a single place-and-route block. Our flow addresses the "chicken-and-egg" loop between floorplan site definition and the post-placement choice of cell heights, and correctly models (based on industry feedback from 20SOC and 16FF design experience) "breaker cell" overheads of the mixed-height placement. Our optimization, applied to production 12T and 8T libraries in a 28LP foundry technology, can achieve 30+ percent area and power reductions while maintaining performance, as compared to a 12T-only design flow. Moreover, our optimized mixed-height designs can achieve significant performance increase along with area and power reductions as compared to designs with 8T-only cells.

[69] For our benchmark designs and selected clock periods, 8T (resp. 12T) cells typically lead to timing violations (resp. power and area overheads), we therefore consider 10T cells to have an optimized performance-power/area tradeoff.

**Table 4.4**: Parameters and results of implemented designs.

| **Flow** | 12T | 8T | mix | 12T | 8T | mix | 12T | 8T | mix | 12T | 8T | mix |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Design** (**clk period**) | *AES* (650$ps$) | | | *DES* (600$ps$) | | | *JPEG* (750$ps$) | | | *MPEG* (550$ps$) | | |
| **#Instances** | 14466 | 16056 | 14967 | 19896 | 23699 | 24738 | 52887 | 60137 | 57756 | 14117 | 15709 | 14820 |
| **Setup WNS** ($ps$) | -54 | -119 | 1 | -0 | -18 | 1 | -0 | -20 | -0 | -10 | -143 | 2 |
| **Setup TNS** ($ns$) | -2.755 | -18.963 | 0.000 | -0.000 | -1.083 | 0.000 | -0.000 | -2.741 | -0.000 | -0.107 | -18.861 | 0.000 |
| **Area** ($\mu m^2$) | 15481 | 13582 | 12190 | 20470 | 18218 | 15598 | 59998 | 49156 | 41703 | 21055 | 15494 | 13730 |
| **WL** ($mm$) | 160 | 151 | 157 | 183 | 173 | 177 | 535 | 512 | 510 | 150 | 148 | 152 |
| **Leakage** ($mW$) | 0.489 | 0.366 | 0.158 | 0.346 | 0.385 | 0.203 | 1.017 | 0.959 | 0.511 | 0.216 | 0.206 | 0.090 |
| **Total power** ($mW$) | 35.0 | 30.0 | 25.7 | 57.2 | 52.9 | 44.9 | 67.1 | 58.6 | 51.4 | 34.4 | 30.2 | 27.7 |
| **Clock area** ($\mu m^2$) | 15 | 13 | 12 | 53 | 41 | 39 | 158 | 131 | 112 | 102 | 167 | 64 |
| **Clock WL** ($\mu m$) | 2017 | 1841 | 1463 | 8050 | 6834 | 6490 | 23977 | 19549 | 17912 | 11678 | 10620 | 10470 |
| **Runtime** ($min$) | 146 | 147 | 224 | 113 | 140 | 257 | 237 | 265 | 514 | 57 | 91 | 163 |
| **Design** (**clk period**) | *AES* (700$ps$) | | | *DES* (650$ps$) | | | *JPEG* (800$ps$) | | | *MPEG* (600$ps$) | | |
| **#Instances** | 14126 | 15688 | 14594 | 18504 | 22708 | 22321 | 54439 | 57310 | 56084 | 13776 | 15957 | 13655 |
| **Setup WNS** ($ps$) | -3 | -60 | 1 | -0 | -13 | 0 | -2 | -8 | -0 | 1 | -82 | -5 |
| **Setup TNS** ($ns$) | -0.009 | -7.452 | 0.000 | -0.000 | -0.367 | 0.000 | -0.005 | -0.182 | -0.000 | 0.000 | -8.482 | -0.048 |
| **Area** ($\mu m^2$) | 14021 | 13241 | 8755 | 19586 | 16186 | 14539 | 59529 | 44998 | 41240 | 20171 | 14849 | 13268 |
| **WL** ($mm$) | 166 | 141 | 135 | 180 | 159 | 158 | 518 | 477 | 468 | 147 | 139 | 136 |
| **Leakage** ($mW$) | 0.387 | 0.350 | 0.128 | 0.290 | 0.302 | 0.144 | 1.038 | 0.760 | 0.516 | 0.164 | 0.153 | 0.081 |
| **Total power** ($mW$) | 29.4 | 27.5 | 17.7 | 51.0 | 44.6 | 39.1 | 59.6 | 50.9 | 46.3 | 30.9 | 26.4 | 24.9 |
| **Clock area** ($\mu m^2$) | 15 | 13 | 11 | 49 | 35 | 36 | 109 | 100 | 93 | 81 | 102 | 65 |
| **Clock WL** ($\mu m$) | 2079 | 1915 | 1363 | 7823 | 6107 | 6019 | 21308 | 22895 | 16612 | 11631 | 11270 | 9724 |
| **Runtime** ($min$) | 136 | 158 | 128 | 91 | 151 | 176 | 239 | 310 | 495 | 37 | 65 | 107 |
| **Design** (**clk period**) | *AES* (750$ps$) | | | *DES* (700$ps$) | | | *JPEG* (850$ps$) | | | *MPEG* (650$ps$) | | |
| **#Instances** | 13805 | 14981 | 14122 | 18123 | 22206 | 21782 | 52401 | 58055 | 57304 | 13421 | 14490 | 13478 |
| **Setup WNS** ($ps$) | -3 | -24 | 3 | 0 | -8 | -0 | 0 | -11 | 0 | 1 | -56 | 2 |
| **Setup TNS** ($ns$) | -0.003 | -1.740 | 0.000 | 0.000 | -0.091 | -0.000 | 0.000 | -0.241 | 0.000 | 0.000 | -5.163 | 0.000 |
| **Area** ($\mu m^2$) | 12202 | 11901 | 8334 | 19158 | 15040 | 13767 | 57128 | 44073 | 40678 | 19717 | 14635 | 13127 |
| **WL** ($mm$) | 157 | 137 | 131 | 171 | 154 | 156 | 539 | 425 | 423 | 149 | 143 | 135 |
| **Leakage** ($mW$) | 0.275 | 0.290 | 0.096 | 0.253 | 0.243 | 0.104 | 0.915 | 0.679 | 0.423 | 0.141 | 0.147 | 0.063 |
| **Total power** ($mW$) | 24.0 | 22.7 | 16.3 | 46.1 | 39.1 | 34.9 | 54.5 | 48.1 | 44.1 | 28.5 | 23.3 | 22.9 |
| **Clock area** ($\mu m^2$) | 19 | 10 | 12 | 47 | 29 | 40 | 100 | 106 | 95 | 82 | 58 | 51 |
| **Clock WL** ($\mu m$) | 1960 | 1683 | 1959 | 7605 | 5933 | 6326 | 20133 | 18441 | 16055 | 11709 | 10027 | 9734 |
| **Runtime** ($min$) | 49 | 73 | 113 | 97 | 111 | 93 | 189 | 244 | 579 | 59 | 91 | 63 |
| **Design** (**clk period**) | *AES* (800$ps$) | | | *DES* (750$ps$) | | | *JPEG* (900$ps$) | | | *MPEG* (700$ps$) | | |
| **#Instances** | 12883 | 14480 | 14043 | 17636 | 20178 | 19788 | 51991 | 57388 | 56287 | 11977 | 14514 | 13445 |
| **Setup WNS** ($ps$) | 0 | -11 | 2 | -0 | -6 | 1 | 0 | -11 | 0 | 1 | -12 | 3 |
| **Setup TNS** ($ns$) | 0.000 | -0.280 | 0.000 | -0.000 | -0.024 | 0.000 | 0.000 | -0.375 | 0.000 | 0.000 | -0.455 | 0.000 |
| **Area** ($\mu m^2$) | 11294 | 10244 | 7402 | 18637 | 14359 | 13447 | 54647 | 44758 | 40400 | 19434 | 14038 | 12794 |
| **WL** ($mm$) | 155 | 133 | 128 | 165 | 150 | 153 | 591 | 483 | 479 | 159 | 138 | 140 |
| **Leakage** ($mW$) | 0.220 | 0.224 | 0.078 | 0.220 | 0.207 | 0.094 | 0.711 | 0.699 | 0.413 | 0.129 | 0.116 | 0.054 |
| **Total power** ($mW$) | 21.2 | 19.2 | 13.7 | 41.8 | 35.5 | 32.3 | 52.0 | 42.7 | 38.4 | 26.8 | 21.4 | 21.0 |
| **Clock area** ($\mu m^2$) | 13 | 11 | 11 | 47 | 30 | 32 | 111 | 85 | 93 | 101 | 50 | 46 |
| **Clock WL** ($\mu m$) | 2004 | 1566 | 1570 | 7397 | 6451 | 7270 | 21171 | 17433 | 17862 | 11823 | 9717 | 9818 |
| **Runtime** ($min$) | 74 | 107 | 76 | 81 | 120 | 76 | 139 | 285 | 424 | 61 | 83 | 58 |

## 4.2 NOLO : A No-Loop, Predictive Useful Skew Methodology for Improved Timing in IC Implementation

Zero-skew clock tree synthesis is commonly used in conventional chip implementation flows to minimize the maximum clock skew. Figure 4.15(a) shows a conventional chip implementation flow, in which we synthesize a design described in RTL to obtain a gate-level netlist. We then place the gate-level netlist, perform clock tree synthesis (CTS) based on the placement results, and route the connections in the design. We refer to this as a *zero-skew flow*.

By intentionally skewing clock latencies[70] of flip-flops (*flops*), we can increase the timing slacks on critical paths while still satisfying the timing constraints on non-timing critical paths [67][179]. This skew scheduling methodology for timing optimization is well-known as *useful skew*. Previous works that study useful skew mainly focus on two objectives – (i) to minimize the clock period and (ii) to maximize the timing margin (robustness). Fishburn [67] formulates a linear program (LP) to optimize clock latencies for performance improvement. The LP formulation considers both setup and hold constraints. Szymanski [179] further improves the efficiency of the LP by selectively generating constraints. Wang et al. [194] also propose an LP-based approach to evaluate potential slacks in circuits and optimize clock skew. The clock skew optimization problem can also be solved by graph-based methods as in [52].

More recent work of Albrecht et al. [3][4] formulates useful skew optimization as a *maximum mean weight cycle* (MMWC) problem, which optimizes not only the minimum slack in a circuit, but also the slacks on other paths. The MMWC approach achieves better timing improvement than the LP-based approach, and is currently the standard approach for useful skew optimization in commercial EDA tools. Runtimes are reduced using faster MMWC algorithms such as [192][195].

Figure 4.15(b) shows a *typical useful skew flow*, in which the clock latencies are optimized after synthesis, placement and CTS in the *Skew_opt* step. A crucial observation is that the typical useful skew flow suffers from a "chicken-and-egg" quandary: after the netlist has been synthesized and placed with zero skew, what useful skew can accomplish is limited.

To fully exploit the potential of useful skew, Albrecht et al. [6] interleave useful skew with RTL synthesis to optimize the performance and area of a design. Hurst et al. [93] propose a placement algorithm with a tight integration of useful skew to minimize maximum mean delay in any circuit loop. Although these methods can inject useful skew into synthesis or placement

[70] We define clock latency as the delay from the clock source to a flip-flop clock input pin.

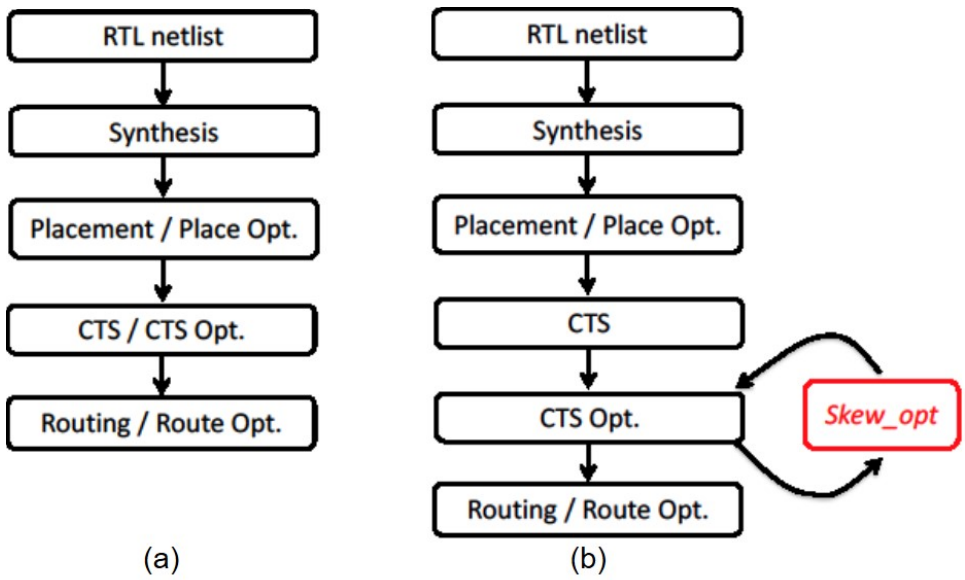



**Figure 4.15**: (a) A conventional zero-skew chip implementation flow (*zero-skew flow*). (b) A standard useful skew flow (*typical useful skew flow*).

stages of implementation, substantial changes would be required to implement them with existing commercial tools. Thus, the work of Wang et al. [191] is notable for its feasibility with modern back-end EDA tools: the authors propose to back-annotate post-placement clock latencies (obtained from useful skew optimization) to the pre-synthesis stage, and re-execute the flow. I.e., after feeding back the clock latencies, [191] re-performs synthesis and placement, followed by another useful skew optimization (see Figure 4.16)(a). This synthesis, placement and useful-skew loop continues until there are no further improvements; empirical results in [191] imply that only two iterations are required to realize the benefits of the proposed methodology.

**Our Work**

Although the back-annotation flow can account for interactions between synthesis, placement and useful skew optimizations, having such a loop in the flow has unacceptable turnaround time impacts. According to [148], it is practically infeasible to make multiple iterations through re-synthesis and physical implementation, as even the time for placement alone of a large hard macro block in a $28nm$ SOC can be five days (and, a single pass through placement + placeOpt + CTS can have over a week of runtime). This motivates us to seek a *predictive, one-pass* means of addressing the chicken-egg problem for useful skew.

To avoid turnaround time impact, we *predict* and enforce useful skews at the post-synthesis stage, within a one-pass implementation. As outlined in Figure 4.16(b), our new

NOLO ("no-loop") flow predicts useful skews based on timing analysis of the synthesized netlist using the default wireload model provided in timing libraries. Experimental results in Section 4.2.2 show that our simple prediction flow achieves good timing quality compared to a **Typical** useful skew flow without only a single implementation pass (i.e., no runtime penalty). We further improve circuit timing with a variant flow (the dotted box in Figure 4.16(b)) that predicts the useful skews based on *two* synthesized netlists. With the optional flow, we can improve total negative slack by 5% compared to the back-annotation flow of [191]. Note that the additional synthesis run has no turnaround impact as we can launch both synthesis runs in parallel.

To complete our study, we also implement a wide range of alternative back-annotation flows (e.g., post-routing information can be fed back to synthesis, to placement, or to clock tree synthesis stages) to experimentally assess their runtime and timing quality tradeoffs.

Our discussion below will use the following definitions.

- *Zero-skew flow :* the conventional chip implementation flow with zero-skew CTS.
- *Typical useful skew flow :* one-pass chip implementation flow with useful skew optimization using a commercial tool, e.g., *skew_opt* in *Synopsys IC Compiler vH-2013.03-ICC-SP3* [239].
- *Back-annotation flow :* a chip-implementation flow that feeds back circuit information to earlier stages for useful skew optimization. Variants of back-annotation flows are described in Section 4.2.2.
- *Prediction flow :* **our** new one-pass chip implementation flow, NOLO, with useful skew optimization at the post-synthesis stage.

We use *slack* to denote the endpoint setup slack on maximum-delay paths between sequentially adjacent flops or ports [71]. Furthermore, since MMWC is the de facto standard approach for useful skew optimization, we perform useful skew scheduling using the *maximum mean weight cycle* formulation of [4] and the algorithms given in [3]. Thus, (i) our useful skew optimization is same as that in the back-annotation flow of Wang et al. [191], and (ii) we assume that the "typical useful skew flow" also optimizes the skew schedule using the MMWC formulation.

**Scope and Organization**

Our work achieves the somewhat surprising result that an improved useful skew optimization at the post-synthesis stage can enable a single-pass flow to achieve similar or better

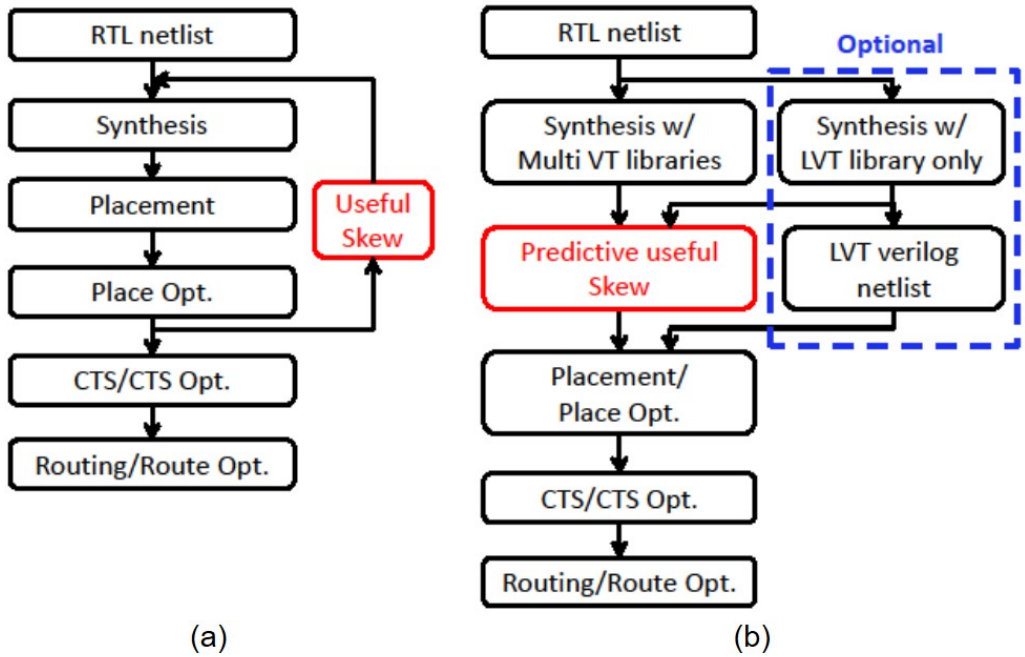



**Figure 4.16**: (a) A chip implementation flow with useful skew back-annotation (*back-annotation flow*). (b) Our predictive NOLO ("no-loop") useful skew flow (*prediction flow*).

timing improvements compared to back-annotation flows. We focus on optimization of useful skews rather than the downstream physical implementation (i.e., CTS, placement and routing with given useful skews). Our three main contributions are summarized as follows.

1. We show that applying useful skews at post-synthesis stage of circuit implementation improves the timing correlation between post-synthesis stage and post-routing stage.

2. We also show that with an additional synthesis run, our predictive useful skew flow can achieve better timing slacks compared to back-annotation flows.

3. We implement different useful skew flows to study the tradeoffs between runtime and timing slacks (with the same area and power).

We present our NOLO prediction flow in Section 4.2.1. Section 4.2.2 describes our experimental setup, implementation details of different useful skew flows and experimental results. Section 4.2.3 concludes our discussion and gives several directions for future work.

### 4.2.1 Methodology

Our predictive flow applies useful skew optimization to a post-synthesis netlist, such that the useful skew optimization is not affected by an initial placement, and allows for a one-pass chip implementation flow.

#### Analysis of the Impact of Placement and Timing Optimization

Intuitively, applying predicted useful skews at the post-synthesis stage is risky, in that timing information at this stage is incomplete. In other words, the circuit timing will be changed by subsequent placement, routing and optimization steps (e.g., cell resizing and/or swapping, buffer insertion, cloning, parasitics from wiring, etc.). To gain initial understanding of the impact of a predictive useful skew flow at the post-synthesis stage, we run two basic implementation flows as illustrated in Figure 4.17.

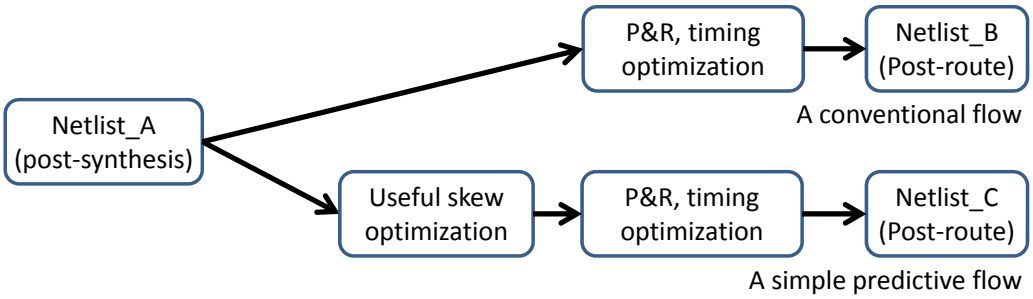



**Figure 4.17**: Overview of two basic implementation flows.

Given a post-synthesis netlist ($Netlist\_A$), we run placement and routing (P&R) to obtain a post-routing netlist without any useful skew optimization ($Netlist\_B$). Meanwhile, we extract timing information from $Netlist\_A$, and apply MMWC-based useful skew optimization. Based on the useful skew results, we annotate clock latencies in an *SDC* file and run the same P&R flow to obtain another post-routing netlist ($Netlist\_C$).

Figure 4.18 shows the timing slacks (for all sequentially adjacent flop pairs) at post-synthesis stage versus the timing slacks at the post-routing stage. In Figure 4.18(a), we can see that in a chip implementation flow without any useful skew optimization (i.e., the top flow in Figure 4.17), the timing slacks at post-synthesis stage have poor correlation with the timing slacks at post-routing stage. For example, critical paths at post-routing stage (timing slack = 0) correspond to the paths with $0ps$ to $250ps$ timing slacks at post-synthesis stage. On the other hand, Figure 4.18(b) shows that with useful skew optimization at post-synthesis stage, the timing slacks at post-synthesis and post-routing stages have much better correlation. More specifically,

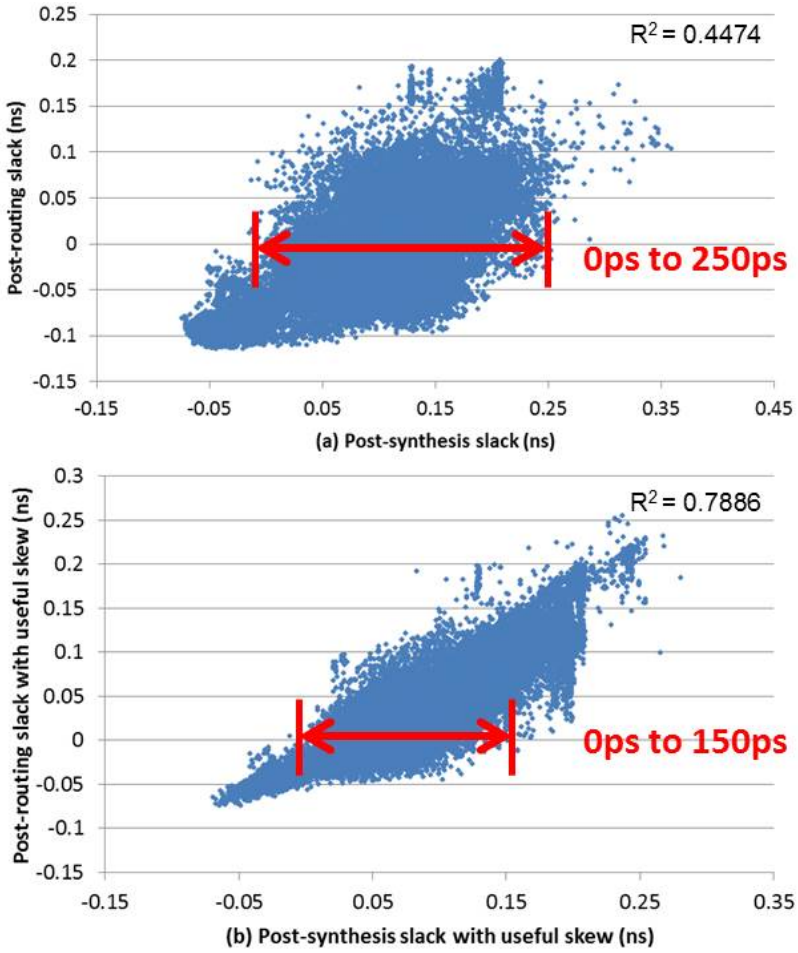



**Figure 4.18**: Timing slacks at post-synthesis versus timing slacks at post-routing stage: (a) without useful skew, and (b) with useful skew. Paths are extracted from the *MPEG* testcase with $0.4ns$ clock period (Table 4.5).

the critical paths at post-routing stage (timing slack = 0) correspond to the paths with $0ps$ to $150ps$ timing slacks at post-synthesis stage when useful skew is applied at post-synthesis stage. This is because the useful skew optimization at post-synthesis relaxes the timing constraints. As a result, the P&R stages do not need to significantly perturb the netlist to meet the timing constraints. Further, Figure 4.19 shows that the relative values of useful skew and timing slacks are similar for post-synthesis and post-routing stages. The post-routing slack is slightly smaller due to the impact of interconnect delay and power/area optimization during the P&R stage.

**A Key Observation.** Because of the good correlation between timing slacks at post-synthesis stage and post-routing stages, the clock latencies resulting from useful skew optimization are similar at these two stages. Therefore, we expect that applying useful skew optimization at post-synthesis stage will lead to similar timing improvements compared to applying useful skew

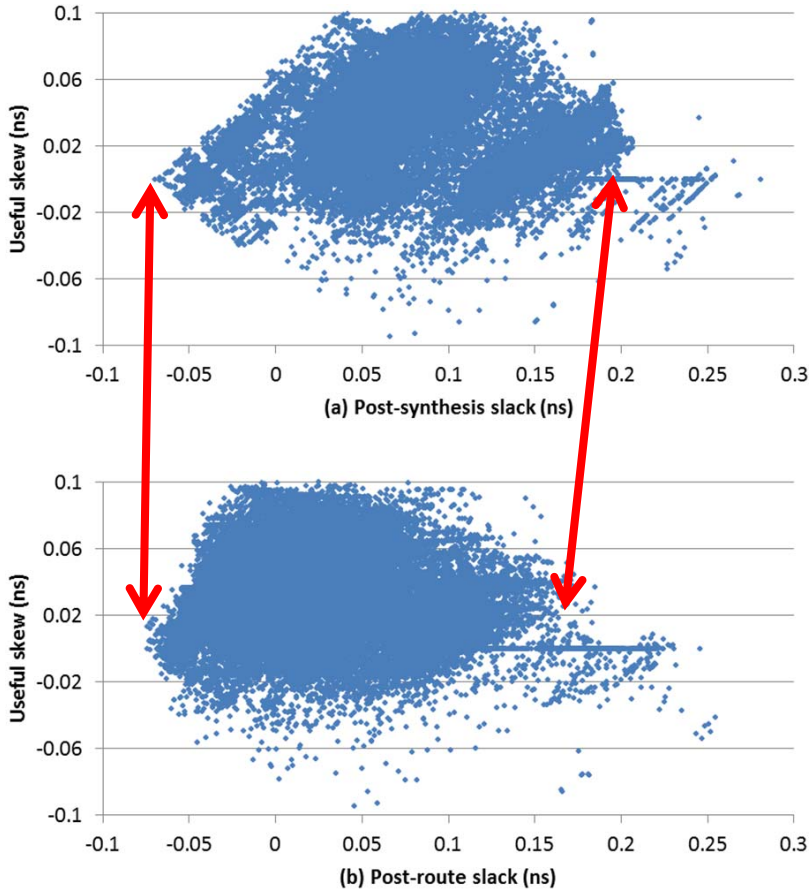



**Figure 4.19**: Useful skew versus timing slacks at (a) post-synthesis and (b) post-routing stages. Paths are extracted from the *MPEG* testcase with 0.4$ns$ clock period (Table 4.5).

optimization at later stages. We validate this hypothesis by generating the optimal useful skews at post-routing stage ($Netlist\_C$) and comparing with the predictive useful skews generated at post-synthesis stage ($Netlist\_A$). Each dot in Figure 4.20 represents the useful skew of a pair of sequentially adjacent flops, where paths are extracted from the *MPEG* testcase with 0.4$ns$ clock period (Table 4.5). The x-axis is the optimal useful skew at post-synthesis stage and the y-axis is the optimal useful skew at post-routing stage; the correlation coefficient for the useful skews is 0.83.

Since the predicted useful skews at the post-synthesis stage are very similar to the **optimal** useful skews at the post-routing stage, our predictive useful skew flow would seem likely to achieve near-optimal timing quality. In other words, results in Figure 4.18 suggest why simple prediction of useful skews at the post-synthesis stage is feasible. Note that the results in Figures 4.18 to 4.20 are representative for all other testcases in our study.

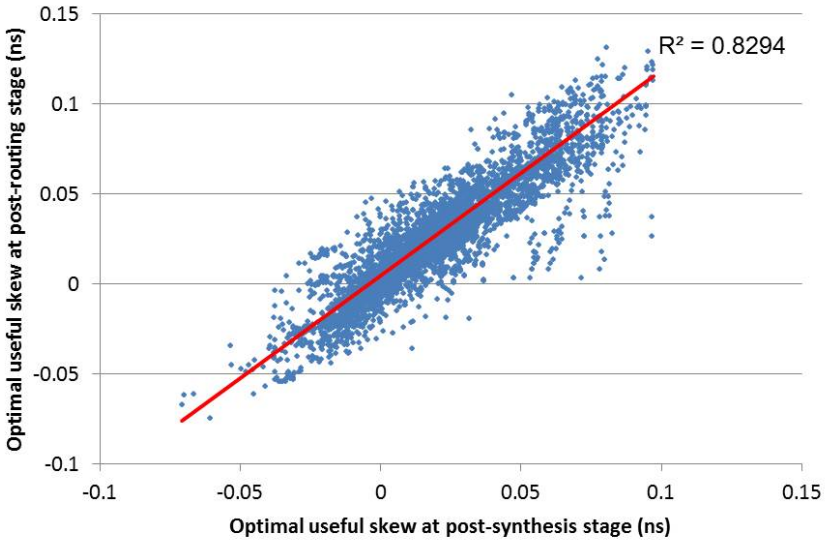



**Figure 4.20**: Optimal useful skews (obtained from MMWC) based on timing information at post-synthesis and post-routing stages have good correlation.

**Implementation of Predictive Useful Skew Flow**

It is well known that useful skew optimization migrates timing slack from a non-critical path to the sequentially adjacent critical paths. Thus, the maximum achievable timing slack is bounded by the mean timing slack of paths that form a cycle. Therefore, we follow standard practice and formulate the useful skew optimization as the *maximum mean weight cycle* (MMWC) problem [3][4]. Given a post-synthesis netlist with edge-triggered flops, we model the netlist using the directed graph $G(V, E)$, where each flop in the netlist is represented by a vertex[71] and there is an edge between two vertices whenever there is a purely combinational path between the corresponding flops. The setup and hold slacks on the path are modeled by the following equations

$$\begin{aligned} s_{i,j,setup} &= -x_i + x_j + T - d_i - d_{i,j}^{max} - t_j^{setup} \\ s_{i,j,hold} &= x_i - x_j + d_i - d_j + d_{i,j}^{min} - t_j^{hold} \end{aligned} \tag{4.4}$$

where $s_{i,j,setup}$ and $s_{i,j,hold}$ are respectively the setup and hold slack on the path from the $i^{th}$ flop ($f_i$) to the $j^{th}$ flop ($f_j$). $x_i$ denotes the clock latency of the $i^{th}$ flop. $T$ is the clock period, $d_i$ is the clock-to-Q delay of $f_i$, and $d_{i,j}^{max}$ and $d_{i,j}^{min}$ are respectively the maximum and minimum path delay from $f_i$ to $f_j$. Last, $t_j^{setup}$ and $t_j^{hold}$ are the setup and hold time of $f_j$, respectively.

[71] Following guidance from [2], all input (resp. output) ports are merged and treated as a single vertex in our MMWC useful skew optimization. This step enables every maximum-delay combinational path (flop-flop, PI-flop or flop-PO) to be included in at least one cycle.

We then formulate our useful skew optimization as

$$\begin{aligned} \text{Maximize} \quad & \sum_{i,j} s_{i,j,setup} \\ \text{Subject to} \quad & s_{i,j,hold} \geq 0, \forall i, j \end{aligned} \tag{4.5}$$

We optimize the sum of setup slacks (flop pairs) because a larger setup slack can potentially improve the achievable operating frequency, or be traded off for power and area recovery. We also consider hold time constraints to ensure correct circuit operation. In the MMWC optimization, we first calculate the weight of each edge (i.e., the worst setup slack corresponding to a pair of flops). We then find the minimum-weight edge in each iteration and label it as a critical path. For an efficient implementation, we determine the minimum-weight edges using the *parametric shortest path* algorithm (details of which are given in [3]). When the critical paths form a cycle, we set the weight (i.e., timing slack) of each edge on the cycle as the maximum mean weight of the cycle. Based on the timing slack, we then determine the clock latency for each vertex (flop). After assigning the clock latencies, we contract the cycle into one vertex and update the weights of incoming/outgoing edges of the contracted vertex. We iteratively search for the minimum-mean-weight cycle and contract the cycle until every vertex is assigned to a clock latency. To incorporate hold constraints in the MMWC, we add edges in parallel to the edges corresponding to setup slack (but with reversed direction). Similarly, each of these (hold) edges is given a weight that corresponds to the hold slack. The parametric shortest path algorithm will honor the constraints defined by hold edges when it searches for the minimum (setup) weight edge.

**An Improved Predictive Useful Skew**

The solution quality of useful skew optimization at the post-synthesis stage will be affected by various timing optimizations during place and route, such as $V_{th}$-swapping and sizing. To address this issue, we also predict useful skews based on a netlist synthesized with only the fastest available cells (e.g., low threshold voltage (LVT) library) (Algorithm 19). Prediction of useful skews based on the LVT-only netlist not only comprehends the impact of $V_{th}$-swapping in later-stage optimizations, but also estimates the achievable slack between each flop pair. However, hold time analysis on a netlist with only the fastest cells is too conservative. Thus, we also propose to synthesize the design with multiple libraries (e.g., multi-$V_{th}$ cell libraries) and formulate the hold constraints based on the multi-$V_{th}$ netlist (Line 4 in Algorithm 19). As shown in Algorithm 19, this prediction flow requires two synthesis runs, which can be executed in parallel

so that there is no turnaround time impact. Based on the synthesized LVT and multi-$V_{th}$ netlists, we optimize useful skews using the MMWC algorithm (Line 5). We then use the LVT netlist for placement and routing (P&R) (Line 6). Note that we use multi-$V_{th}$ libraries for P&R implementations, i.e., the P&R tools will optimize power by swapping LVT cells to other $V_{th}$ flavors on *non-critical timing paths*. Thus, the accuracy of our useful skew prediction based on LVT-only netlist is less affected by the $V_{th}$ swapping. In the following discussion, we use **SimPred** to refer to the simple prediction flow described in Section 4.2.1, and **ImpPred** for this improved predictive useful skew flow based on two synthesis runs.

**Algorithm 19** No-loop, predictive useful skew methodology

**Procedure** $ImpPred(RTL,\ SDC,\ Liberty_{LVT}, Liberty_{MVT})$
**Output:** $N_{out}$
1: $N_{LVT} \leftarrow$ Synthesis$(RTL,\ SDC,\ Liberty_{LVT})$;
2: $N_{MVT} \leftarrow$ Synthesis$(RTL,\ SDC,\ Liberty_{MVT})$;
3: $V \leftarrow$ flops, PIs, POs in $N_{LVT}$;
4: $E \leftarrow$ max-delay paths in $N_{LVT} \cup$ min-delay paths in $N_{LVT}$;
5: clock latencies $\leftarrow$ MMWC($V$, $E$);
6: $N_{out} \leftarrow$ P&R($N_{LVT}$, $SDC$, $Liberty_{LVT}$, clock latencies);

### 4.2.2 Experimental Results

Our experiments use a dual-$V_{th}$ $28nm$ FDSOI library and three RTL designs from the *OpenCores* website [230]. We show statistics of testcases (including clock period, total number of cells, number of flops, and number of maximum/minimum delay paths (i.e., number of edges in the sequential graph)) in Table 4.5. We use *Synopsys Design Compiler vH-2013.03-SP3* [237] to synthesize the RTL netlists.[72] We run P&R using *Synopsys IC Compiler vH-2013.03-ICC-SP3* [239]. We also use *Synopsys IC Compiler* for power analysis, and *Synopsys PrimeTime H-2013.06-SP2* [240] for timing analysis. The setups for timing analysis are given in Table 4.6, where in the absence of AOCV tables we use timing derates to model on-chip variation. All (dual-$V_{th}$) implementation experiments are run with two signoff corners at $\{125°C$, $0.9V$, SS$\}$ and $\{$-40$°C$, 1.05$V$, FF$\}$. To mitigate the effects of tool noise [95], each P&R implementation executes three separate runs with small perturbations of clock period (i.e., -1$ps$, 0$ps$, +1$ps$); we report the largest endpoint slack results obtained over all three final-routed netlists.

The back-annotation flow can have different variants. In addition to the back-annotation flow proposed in [191], we have implemented four variant back-annotation flows, designated as

[72] A physical synthesis flow is used: We first run the default synthesis flow, then implement a fast placement of the synthesized netlist, based on which another pass of synthesis is made with topographical ("topo") option.

**Table 4.5**: Benchmark designs.

| **Design** | **Clk period** ($ns$) | **#Cells** | **#Flip-flops** (**#Vertices**) | **#Paths** (**#Edges**) |
|---|---|---|---|---|
| *AES* | 0.6 | ∼23K | 530 | 16251 |
| *DES* | 0.5 | ∼11K | 1985 | 23153 |
| *JPEG* | 0.6 | ∼50K | 4712 | 137333 |
| *MPEG* | 0.4 | ∼11K | 3381 | 95490 |

**Table 4.6**: Experimental setups for timing analysis.

| **Parameter** | **Value** |
|---|---|
| Clock uncertainty (synthesis) | 0.15 × clock period |
| Clock uncertainty (placement, CTS) | 0.10 × clock period |
| Clock uncertainty (CTS opt, routing) | 0.05 × clock period |
| Maximum transition | 0.08 × 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 |

**BA-I**, **BA-II**, **BA-III** and **BA-IV**.

In **BA-I** (Figure 4.21(a)), we collect timing information at post-placement stage, optimize useful skew, and back-annotate the clock latencies to the post-synthesis stage. For **BA-II**, **BA-III** and **BA-IV**, we collect timing information at post-routing stage and optimize useful skew. The optimized clock latencies are then back-annotated to the synthesis, placement, and CTS stages, respectively, in **BA-II**, **BA-III** and **BA-IV**.

We perform chip implementations on designs (listed in Table 4.5) with eight chip implementation flows – (a) the standard useful skew flow (**Typical**) where we use the command *skew_opt* in *Synopsys IC Compiler vH-2013.03-SP3* [239] to generate desired clock latencies for incremental clock tree optimization; (b) the back-annotation flow by Wang et al. [191] (**BA-W**), which is depicted in Figure 4.16(a); (c) four variants of back-annotation flows described in Section 4.2.2; and (d) our two NOLO ("no-loop") predictive flows (**SimPred** and **ImpPred**), in which we apply predicted useful skews at post-synthesis stage and continue to use them throughout timing optimization in P&R.

Results in Table 4.7 show that different flows achieve similar power and area. Also, all designs are free of any hold time violation. Thus, we achieve clean comparisons of different flows based on the total negative slack.

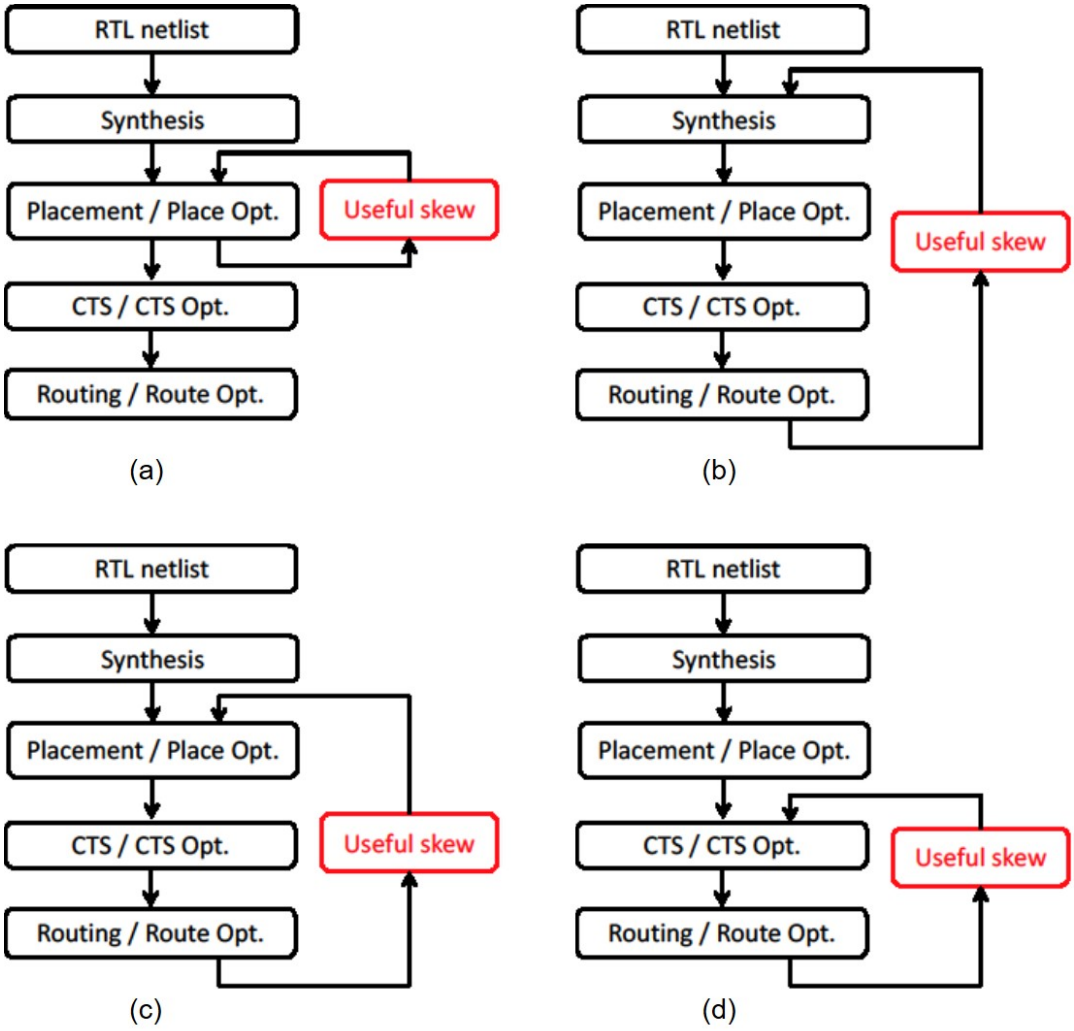



**Figure 4.21**: (a) BA-I flow. (b) BA-II flow. (c) BA-III flow. (d) BA-IV flow.

**Back-annotation vs. Typical.** Results in Table 4.7 show that the **BA-W** flow can achieve better total negative slack (TNS) compared to the **Typical** flow (average across all testcases in Table 4.5). This is mainly because the useful skew optimization in the **BA-W** flow can interact with the synthesis and placement stages through the feedback loop. As a result, the cells on critical paths can be re-sized, re-structured and/or re-allocated to improve timing quality. However, the runtime of the **BA-W** flow is 85% longer than the **Typical** flow.

**SimPred vs. Back-annotation.** Results in Table 4.7 show that although the **SimPred** flow can also achieve significant improvement compared to the **Typical** flow, the average TNS achieved by the **SimPred** flow is approximately 20% worse compared to **BA-W**. This is expected because the useful skew solution (at post-synthesis stage) may be suboptimal due to design changes in

the place and route stages. However, the **SimPred** reduces runtime by 66% compared to the **BA-W** flow.

**ImpPred vs. Back-annotation.** Our results also show that with the concurrent LVT-only synthesis run, the **ImpPred** flow achieves improved TNS, power and area (on average) compared to **BA-W**. This is because the benefit of useful skew optimization is limited by the zero-skew placement in **BA-W**. For example, buffers are inserted in the zero-skew netlist to fix timing violations, which increases area and power. Moreover, the critical paths will not fully exploit the potential benefits of useful skew. In contrast, our **ImpPred** flow relaxes timing constraints at the post-synthesis stage via an early-stage useful skew optimization (see Section 4.2.1). We believe that this enables the optimized netlist to meet timing constraints with less area and power penalty (e.g., less buffer insertions).

Among the four testcases, **BA-W** only does better for the *JPEG* testcase, by a small margin. Overall, our prediction of useful skew at post-synthesis stage is superior to the **BA-W** back-annotation flow. Moreover, our **ImpPred** is a **one-pass** implementation which reduces runtime by 66% compared to **BA-W**. Note that the runtime of the **ImpPred** flow is smaller than the runtime of the **SimPred** flow, even though **ImpPred** implements two synthesis runs. This is because we execute the synthesis runs simultaneously, and the improved timing quality leads to a faster convergence in the P&R stages.

**Design Dependencies.** We observe that the improvements from useful skew implementations are design-dependent. Timing improvements with useful skew are less for a design with fewer flops, because the number of paths that can be improved is smaller (e.g., *AES*). In this work, we have focused only on optimization of timing. Conventional wisdom would suggest that our improvements in timing can be traded for power and area improvements, and we plan to consider the tradeoffs between timing and power/area objectives in our future work.

**Comparison Among Variants of Useful Skew Flows.** We compare the runtime and resultant total negative slacks of various useful skew flows. In the back-annotation flows, we iteratively optimize until the improvement in the average setup slack is less than $50ps$. All the back-annotation flows converge within three iterations.

Figure 4.22 shows that the TNS values of the back-annotation flows vary depending on the testcase. This suggests that even with back-annotation, the useful skew optimization may be misled by the initial netlist and thus end up with suboptimal solutions. Since the back-annotation flows achieve different TNS values, we also plot the average TNS of all back-annotation flows (including **BA-W**) for comparison (i.e., the blue diamond symbol and dotted lines). The re-

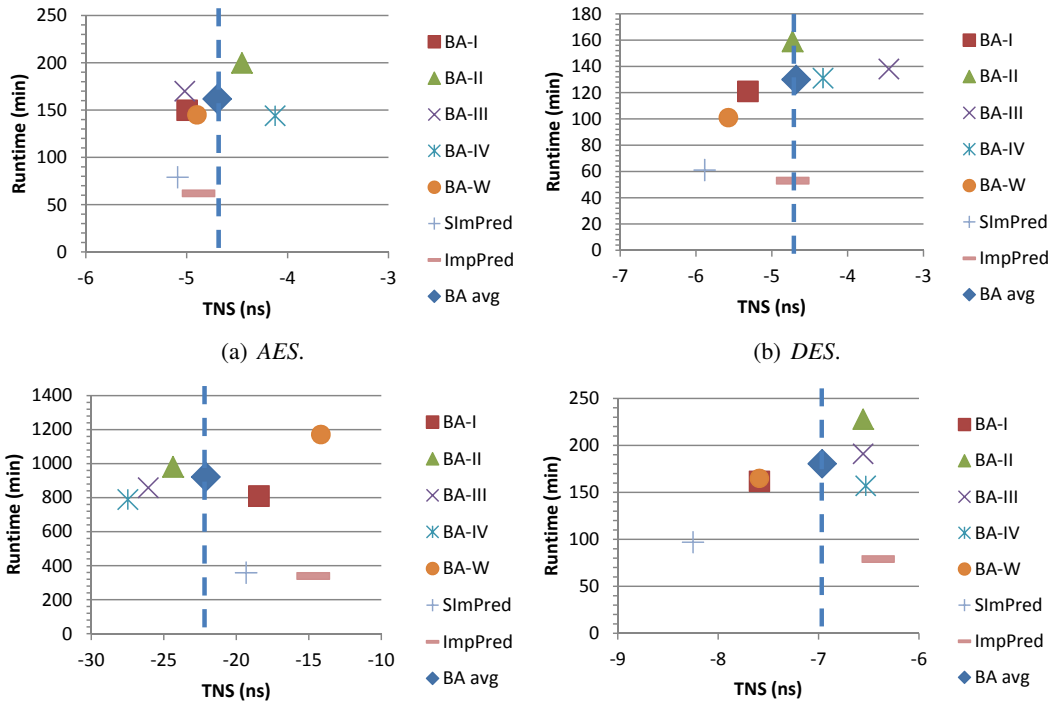
**Table 4.7**: Design metrics of routed design from different flows.

| Design | Flow | Power ($mW$) | Area ($\mu m$) | #Hold violations | TNS ($ns$) | WNS ($ns$) | Runtime ($min$) |
|---|---|---|---|---|---|---|---|
| *AES* | Typical | 16984 | 35.8 | 0 | -7.806 | -0.047 | 117 |
| | BA-W | 16860 | 35.0 | 0 | -4.898 | -0.042 | 145 |
| | SimPred | 16539 | 34.7 | 0 | -5.089 | -0.035 | 79 |
| | ImpPred | 16002 | 34.3 | 0 | -4.883 | -0.036 | 62 |
| *DES* | Typical | 21971 | 65.8 | 0 | -13.920 | -0.046 | 108 |
| | BA-W | 20445 | 61.2 | 0 | -5.574 | -0.032 | 101 |
| | SimPred | 20603 | 62.2 | 0 | -5.885 | -0.034 | 61 |
| | ImpPred | 19618 | 57.2 | 0 | -4.726 | -0.035 | 53 |
| *JPEG* | Typical | 72799 | 77.0 | 0 | -136.650 | -0.131 | 496 |
| | BA-W | 58874 | 64.6 | 0 | -14.166 | -0.043 | 1171 |
| | SimPred | 57878 | 63.4 | 0 | -19.317 | -0.043 | 358 |
| | ImpPred | 56970 | 61.7 | 0 | -14.695 | -0.045 | 339 |
| *MPEG* | Typical | 27655 | 52.6 | 0 | -137.855 | -0.168 | 134 |
| | BA-W | 25761 | 48.5 | 0 | -7.590 | -0.049 | 165 |
| | SimPred | 25415 | 48.3 | 0 | -8.251 | -0.054 | 97 |
| | ImpPred | 25250 | 48.4 | 0 | -6.408 | -0.046 | 79 |
| Average of 4 designs | Typical | 34852 | 57.8 | 0 | -74.058 | -0.098 | 213 |
| | BA-W | 30485 | 52.3 | 0 | -8.057 | -0.042 | 395 |
| | SimPred | 30108 | 52.2 | 0 | -9.636 | -0.042 | 148 |
| | ImpPred | 29460 | 50.4 | 0 | -7.678 | -0.041 | 133 |

sults show that **ImpPred** can achieve better results compared to the average TNS of the back-annotation flows (*BA avg*) for larger testcases (*JPEG* and *MPEG*). For smaller testcases (*AES* and *DES*), **ImpPred** achieves similar TNS compared to the average of back-annotation flows. Also, it is clear that our predictive flows have significantly less runtime than the back-annotation flows for all testcases.

### 4.2.3 Conclusion

We propose NOLO, a "no-loop" predictive useful skew optimization flow, based on timing information of a post-synthesis netlist. To account for the potential of timing changes during the place and route stages, we improve our estimate of potential slack in the netlist by running an additional logic synthesis step using fast library cells. Based on this technique, we show that




(a) *AES*. (b) *DES*.

(c) *JPEG*. (d) *MPEG*.

**Figure 4.22**: Comparison among useful skew flows. Our **ImpPred** flow achieves better or similar TNS but with 66% runtime reduction compared to back-annotation flows.

an improved predictive useful skew flow (**ImpPred**) can achieve similar or better total negative slack compared to back-annotation flows, with only one pass through chip implementation. The runtime of our predictive useful skew flows is similar to the runtime of the **Typical** flow, which is approximately 66% less than the runtime of the back-annotation flow in [191].

Our study of different back-annotation flows indicates that back-annotation (or optimization loops) cannot completely resolve the "chicken-and-egg" problem. We see that the timing quality varies depending on testcases. This is because even with back-annotation, the useful flows can be misled to a suboptimal local solution.

There are two major directions for our future work. First, we plan to analyze and apply our useful skew flows across multiple PVT corners. Second, we plan to study and develop models of the tradeoffs among area, power and timing with useful skew.

## 4.3 Reliability-Constrained Die Stacking Order in 3DICs Under Manufacturing Variability

*Stacked-die 3D integrated circuits* (3DICs) using through-silicon via (TSV) technology are an emerging architecture for heterogeneous integration and More-than-Moore scaling in late-CMOS technologies. A 3DIC die stack, or simply *stack*, offers increased transistor density in a given form factor, as well as potential cost and yield benefits (multiple smaller dies versus a single larger die). However, the stacking of multiple thinned die (also referred to as *tiers*) increases power density, creating temperature management and reliability challenges. Puttaswamy et al. [160] show that 3DICs with two tiers and four tiers increase peak temperature by $17°C$ and $33°C$, respectively, compared to planar implementations. Since current density and temperature have a significant impact on IC reliability, reliability issues are especially important in the 3DIC context [180][185].

With technology scaling, additional challenges arise from *process variability*, with variation sources spanning dopant fluctuation, mask data preparation and OPC, line-edge roughness, misalignment in double-patterning, and a variety of across-field and across-wafer variability mechanisms [126][127]. These process variations are present (and, uncorrelated) within a 3DIC stack; because of higher temperatures due to die stacking, the process variations can heavily affect performance as well as leakage power of the 3DIC product [64]. So that a given product can meet its performance requirements, process variations in each manufactured die are typically characterized at manufacturing time (e.g., for product binning, or to set one-time programmable tables for adaptive voltage scaling [125][213]).

In this section, we study reliability-variability interactions and optimizations in the context of 3DIC die stacking. Specifically, we focus on the stacking of multiple copies of logic dies (e.g., as envisioned for many-core processor die in high-performance computing architectures) [43][161]. We use the term *stacking style* to indicate both the selection of dies which exhibit particular process variations (e.g., fast, typical, or slow dies) as well as the ordering of dies within a given 3DIC product stack. Because of inter-die process variation, the choice of stacking styles will impact performance, power consumption and reliability of 3DICs. In our studies, the required performance for each die is predefined, and the *adaptive voltage scaling* (AVS) [58] is assumed.

There are three methods to bond dies in 3DIC fabrication: die-to-die, die-to-wafer and wafer-to-wafer. Applying different methods results in different flexibility, yield, and cost. Among these three methods, die-to-die bonding offers the highest flexibility and yield, but also

incurs high cost. On the other hand, although wafer-to-wafer bonding offers the highest throughput in production, bad dies cannot be scrapped before bonding, which results in the lowest yield. Die-to-wafer bonding, which is easy to implement while offering flexibility and yield that are similar to the die-to-die method, is promising for 3DIC fabrication. Our work applies primarily to the die-to-die and die-to-wafer bonding contexts.

We assume that all logic dies are identical (a similar assumption can be applied to memory-logic integration, where all memory dies are identical). Such a case may arise if applying the identical design to all the tiers in a stack reduces design efforts as well as manufacturing cost [70]. Based on such an assumption, dies can be used interchangeably in different tiers. Hence, we are able to change the stacking order during optimization.

Our discussion below will assume face-to-back stacking of the multiple logic-die tiers, with a heat sink (or other heat removal mechanism) adjacent to the top tier as illustrated in Figure 4.23. The figure shows a "STF" stacking order for a 3-tier stack, i.e., a slow-corner die on bottom, typical-corner die in the middle, and fast-corner die on top.

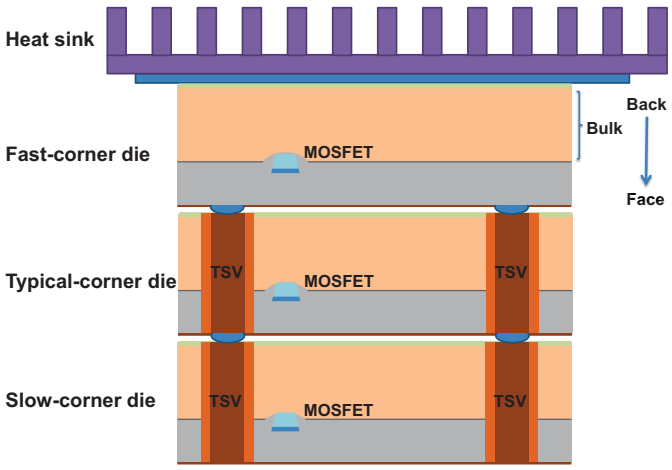



**Figure 4.23**: "STF" stack in which a slow-corner die is located on the bottom tier, a typical-corner die in the middle, and a fast-corner die on the top tier (adjacent to the heat sink).

To motivate our present work, Figure 4.24 shows the *mean time to failure* (MTTF) of 3-tier stacks with different stacking styles (orders). The maximum difference in MTTF resulting from different stacking styles can be up to 2 years (44%), where we assume that the same performance requirement and AVS are applied to all dies in a stack. The study is conducted by estimating temperature using *Hotspot* [221] and calculating MTTF based on Black's equation [15], with the assumption that the supply voltage of each tier in a stack is adjusted using *adaptive voltage scaling* (AVS) to meet a given fixed performance requirement. (Details of this experiment are given in Section 4.3.4 below.)

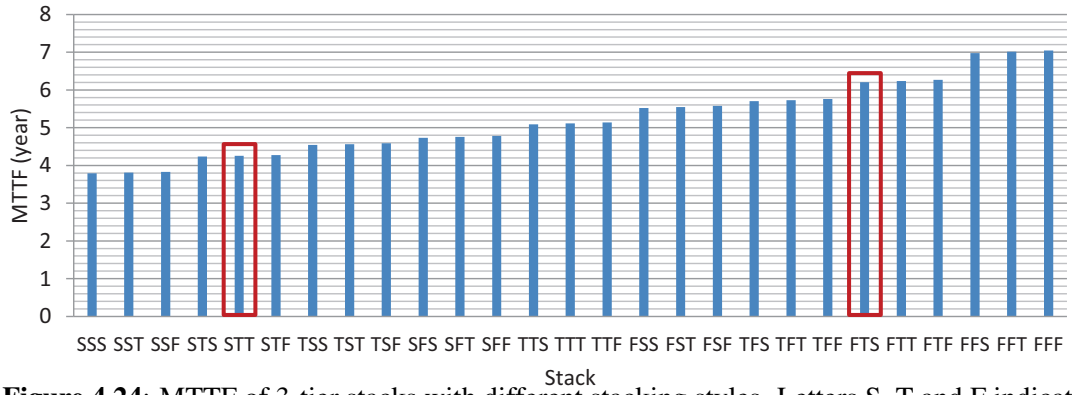



**Figure 4.24**: MTTF of 3-tier stacks with different stacking styles. Letters S, T and F indicate the (slow, typical, fast) process corners to which individual dies belong. Strings over {S, T, F} indicate stacking styles (left-to-right in the string corresponds to bottom-to-top in the stack).

**Related Work**

Relatively few previous works study the issue of stacking styles (and, stacking of multiple logic dies is not yet the focus of current 3DIC products). Ferri et al. [64] examine the impact of process variation on 3DICs, and propose optimization strategies for stacking to increase parametric yield (performance and leakage power) of 3DICs. However, their studies only focus on 3DICs with two tiers, integrating one memory die and one logic die. Ferri et al. use reduction from 3D matching to show that stacking dies to optimize parametric yield ("as measured by performance, leakage, or revenue") is NP-hard; such a problem is tractable only when the number of tiers is $\leq 2$. Cho et al. [38] propose efficient models to predict geo-spatial thermal characteristics within and across different dies without detailed cycle-level simulation. Based on these models, optimal stacking methods are given to improve temperature in 3DICs. However, the work of [38] does not consider the issues of process variation and reliability that are our motivation here.

In general, TSV-based 3DIC integration offers a variety of value propositions. Beyond the integration of heterogeneous technologies (memory, logic, RF, analog, microfluidic, etc. components - e.g., [64]), previous works mainly focus on logic-memory stacking [64][138][145] to increase performance and reduce memory bottlenecks. Logic-logic stacking shortens global wiring and thus decreases signaling latency between blocks, potentially yielding higher performance and smaller power consumption [16]. As noted above, our present study performs experiments with logic-logic stacking; however, we believe that insights from our studies can also be applied to memory-logic integration, particular in scenarios where multiple commodity memory dies are stacked with logic [144][159].

**Scope and Organization**

Based on modeling of power consumption and temperature gradients, and their impacts on chip-level power consumption and reliability, we study the variability and reliability implications of various alternative stacking styles for several distinct product objectives. Our main contributions are as follows.

1. We identify a simple rule-of-thumb (namely, that slower dies should be located closer to the heat sink in 3DICs to achieve better reliability and reduce temperature) for 3DIC stack ordering.

2. We propose an $O(n\ log\ n)$ heuristic method (based on the simple rule-of-thumb) and an integer linear programming (ILP) method to determine stacking styles for large populations of manufactured dies to optimize 3DIC product yield or reliability.

3. Experiments using 5-tier die stacks demonstrate that the methods we propose achieve $\sim$7%, $\sim$28% and $\sim$3% improvements in average MTTF, minimum MTTF and performance (under a reliability constraint), respectively, of the die stacks.

4. Interestingly, our results show that when high-quality stacking optimizations are applied, a limited amount of manufacturing variation can be *helpful* in improving 3DIC product reliability metrics.

The remainder of this section is organized as follows. Section 4.3.1 describes how we model reliability of 3DICs as well as process variation. The simple rule-of-thumb for stacking is also introduced in this section. Section 4.3.2 formulates several stacking optimization problems to improve reliability, yield and performance (under a reliability constraint) of 3DICs. Section 4.3.3 proposes heuristic and ILP-based methods for reliability-driven stacking optimization. Experimental results are described in Section 4.3.4. The section concludes in Section 4.3.5.

### 4.3.1 Modeling

**Reliability**

Narrower line widths and larger current densities make interconnect reliability of increasing concern for overall IC reliability. In particular, signal and power-delivery *electromigration* (EM) is now a dominant reliability constraint in current IC designs [180][202]. Especially given the exponential dependence of EM lifetime on temperature, we will focus our discussion on EM reliability; however, in principle our methodology can apply to any (power- and

temperature-dependent) IC reliability mechanism. We use the well-known empirical estimate given by Black's equation [15] to estimate the EM mean time to failure (MTTF) of each given die:

$$MTTF = \frac{A}{J^n} \cdot exp(\frac{E_a}{k \cdot T}) \tag{4.6}$$

where $A$ is a process parameter based on the cross-sectional area of the wire, $J$ is the current density, $n$ is a scaling factor, $E_a$ is the activation energy, $k$ is the Boltzmann constant, and $T$ is the temperature. Our work uses $E_a$ = 0.7$eV$, $n$ = 2 [15][133]. To evaluate the MTTF of 3DIC stacks, we must establish necessary definitions of *failure rate* and *reliability*, as follows.

**Definition:** The *failure rate* ($\lambda$) is defined as the number of units failing per unit time.

Figure 4.25 illustrates the familiar reliability "bathtub curve" that models the change of failure rate during the lifetime of an electronic device [196]. Such lifetime can be divided into three periods. The first, early-lifetime or "infant mortality" period is characterized by decreasing failure rate. Dominant reliability concerns during this period include oxide defects, masking defects and contamination. Techniques such as burn-in and power- and thermal-cycling are applied during this period to filter out bad devices. During the second period, random failures appear, and the failure rate is modeled as a constant. This period indicates the typical lifetime for usage (useful lifetime) of a device. Thus, our studies mainly focus on this period. The third period is the wear-out period, failure rate increases during this period till the end of a device's lifetime.

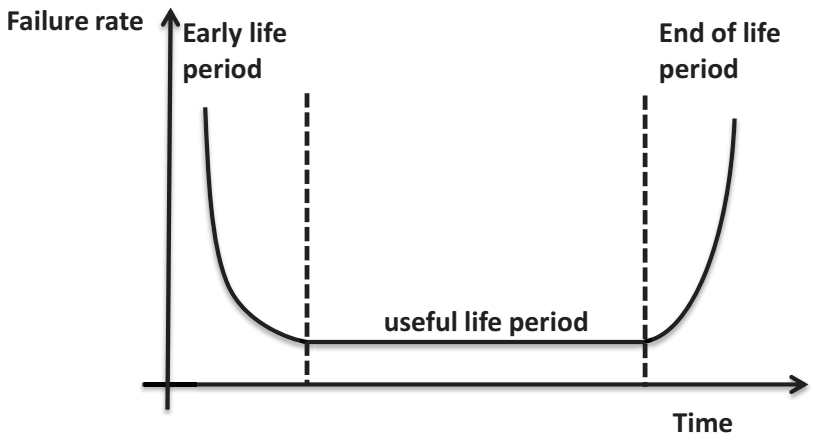



**Figure 4.25**: Reliability "bathtub curve".

**Definition:** The *reliability* ($R(t)$) is defined as the probability that a device (or a die) operating under specified conditions shall perform satisfactorily for a given period of time ($t$).

The reliability can be calculated as [220]

$$R(t) = e^{-\lambda \cdot t} \tag{4.7}$$

Based on (4.7) and a constant $\lambda$ during the useful lifetime, the MTTF of a die (i.e., expectation of the time to failure) can be calculated as

$$MTTF = \int_0^\infty R(x) \cdot dx = \int_0^\infty e^{-\lambda \cdot x} \cdot dx = \frac{1}{\lambda} \tag{4.8}$$

Note that according to (4.8), the value of $\lambda$ can be calculated using Black's equation (4.6).

Furthermore, since any failure of any die in a 3DIC can cause the 3DIC to fail, the failure rate of a 3DIC can be evaluated as

$$\lambda_{stack} = \prod_{i=1}^{L} \lambda_{die_i} \tag{4.9}$$

where $\lambda_{stack}$ is the failure rate of the 3DIC, and $\lambda_{die_i}$ $(i = 1, 2, \ldots, L)$ is the failure rate of the $i^{th}$ die in the stack. Based on (4.8) and (4.9), the MTTF of a 3DIC is

$$MTTF_{stack} = \frac{1}{\prod_{i=1}^{L} \frac{1}{MTTF_{die_i}}} \tag{4.10}$$

where $MTTF_{stack}$ is the MTTF of the 3DIC, and $MTTF_{die_i}$ $(i = 1, 2, \ldots, L)$ is the MTTF of the $i^{th}$ die in the stack.

In our MTTF calculations reported below, we use Black's equation (4.6) to estimate the MTTF for each die in a 3DIC based on temperature and current density information. We then apply (4.10) to calculate the MTTF of a given 3DIC.

**Process Variation**

Given an arbitrary number of dies, each exhibiting different process variation (e.g., characterized during manufacturing test [125][213]), the number of possible stacking styles in 3DICs composed of these dies can be quite large. For example, if there are 2000 distinct (in terms of process variation) manufactured dies, and the 3DIC to be produced has 5 tiers, then the number of distinct stacking styles is $P(2000, 5) = 2000 \cdot 1999 \cdot \ldots \cdot 1996$. Stacking the 2000 dies into 400 5-tier stacks would have an even more unmanageable solution space, wherein figuring out the optimal (set of) stacking styles is intractable. (As noted above, the previous work of [64] shows that the stacking optimization problem for certain objectives is NP-hard.)

In our work, we classify dies into a *constant* number of (i.e., $O(1)$) process *bins* according to the speed of dies. Dies are classified into the same bin if they have similar process

variations, and to make the stacking optimization tractable, we assume the same process variation characteristics for all dies that are classified into a given bin. This bin-based model assumption greatly reduces the number of distinct stacking styles as well as the solution space for stacking optimization. (E.g., for the same example of 2000 manufactured input dies and a 5-tier stack, if we classify the dies into 3 process bins, the number of feasible stacking styles is reduced to $3^5$.)

Taking advantage of such a bin-based model, we are able to explore the reduced solution space and determine optimal stacking styles when given a small number of bins. When instantiating each distinct 3DIC stack (e.g., each of the 400 5-tier stacks to be made out of 2000 manufactured dies), we randomly select dies from corresponding bins to make up the stack.[73] As discussed below, when the number of process bins is sufficiently large, results from stack optimization flows that apply the bin-based models can be near-optimal.

**A Rule-of-Thumb**

For EM reliability, peak temperature is the main determinant of a given die's reliability. Additionally, the die with the weakest reliability in a stack determines the reliability of the entire stack. Thus, to optimize reliability of a 3DIC, we seek to minimize the peak temperature among all stacked dies in a 3DIC. It is not difficult to realize that two factors have significant impacts on the temperature of dies in a 3DIC stack: process variation and stacking order.

As previewed in the discussion above, we assume that the same performance requirement is applied to all dies in a 3DIC, and that to compensate for interdie process variation, AVS is deployed.[74] In this context, individual dies will have different supply voltages, corresponding to process variation. Slow dies require higher supply voltages than fast dies in order to satisfy the performance requirements. Such high supply voltages can lead to high power consumption on slow dies, which increases temperature. Hence, as a consequence of process variation and deployment of AVS, slow dies will have higher temperature than fast dies.

The stacking order can also affect the temperature distribution of dies. We assume that a vertical temperature gradient always exists in the 3DIC stack, because only the top-layer die is directly contacted to the heat sink (a cartoon is shown in Figure 4.26). For dies in lower tiers, the thermal dissipation through the heat sink, which is the primary mechanism for thermal dissipation in 3DICs, is blocked by dies in the upper tiers. Hence, higher temperatures are

[73] For example, in a "FTTTS" 5-tier stack, we would successively pick one random die from the Fast bin, three random dies from the Typical bin, and one random die from the Slow bin, and stack them bottom-up in this order.

[74] Indeed, for nearly all low-power consumer SOCs in advanced nodes today, sensor-based AVS is the norm; it is the only available mechanism to recover power from a chip that has been overdesigned due to large model guardbanding.

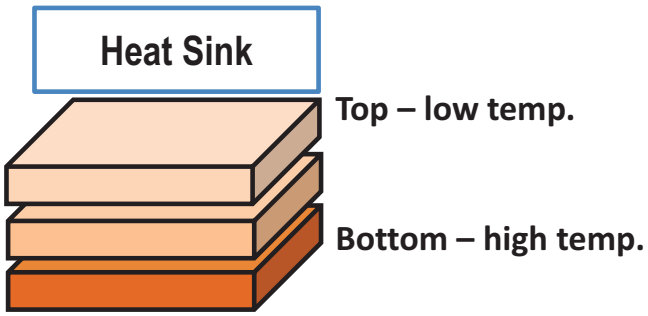



**Figure 4.26**: Temperature gradient. The top-tier die is in direct contact with the heat sink, and thus has the lowest temperature. Due to intervening dies that block thermal conduction to the heat sink, dies in bottom tiers have higher temperature.

observed in bottom-tier dies. Moreover, heat generated from dies in adjacent tiers exacerbates thermal issues for any individual die in the stack. Figure 4.27 shows a simulated 5-tier 3DIC, where all dies in the stack are assumed to exhibit the same process variation. In this example, the maximum temperature difference between the bottom-layer die and the top-layer die is $35°C$.[75]

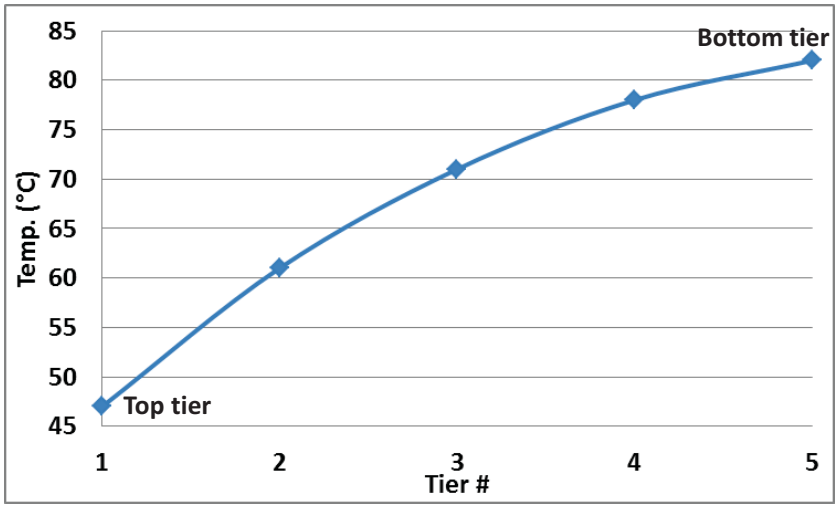



**Figure 4.27**: Example simulated temperature gradient in a 5-tier 3DIC stack. The difference between the peak temperatures in the bottom-tier die and the top-tier die can reach $35°C$.

Based on the above analysis, considering effects of process variation as well as stacking order on temperature distribution, we expect that if the same performance is required from dies which exhibit different process variations, the worst-case peak temperature among feasible stacking styles will occur when we locate slower dies in lower tiers (e.g., the slowest die is located in the bottom tier, the second-slowest die is located in the next-to-bottom tier, and so on). Furthermore, such worst-case peak temperature will likely correspond to the minimum MTTF

[75] In the simulation, we assume that the thermal resistivity for silicon is $100mK/W$, die thickness is $50\mu m$, ambient temperature is $45°C$, and that there is a heat sink on top of the stack.

among all stacking styles, that is, the worst-case of reliability of the 3DIC. On the other hand, if we locate slow dies on top, by taking advantage of thermal dissipation through the heat sink, high temperature caused by high supply voltages can be relieved. Hence, the peak temperature will decrease, and the MTTF of the stack will increase. Note that even when the thermal gradient is small, the vertical thermal distribution is still monotonic, so that placing slow dies on top still results in improved MTTF.

The experimental results shown in Figure 4.28 confirm our expectation. Figure 4.28 shows QoR metrics (MTTF and power) of 5-tier stacks implemented with different stacking orders. We observe in the experimental results that placing slow dies close to the heat sink helps improve the MTTF of the stack. We conclude this part of our discussion with the following *rule-of-thumb*.

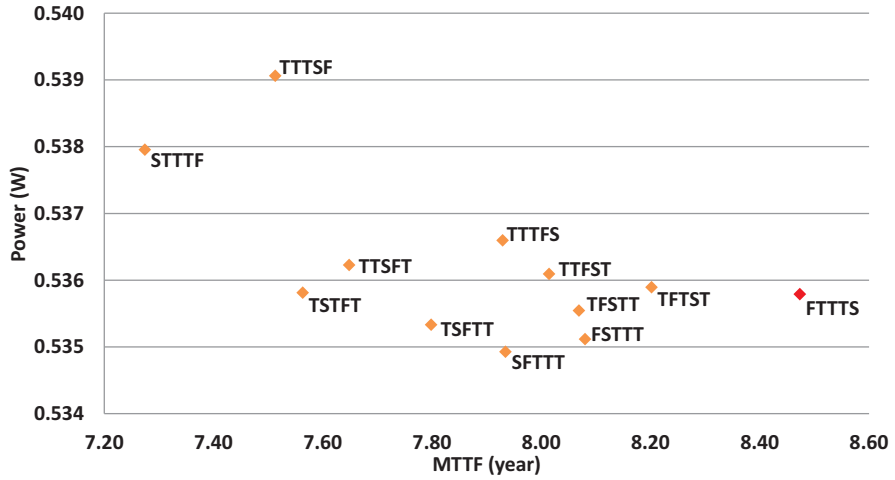



**Figure 4.28**: QoR metrics (MTTF, power) of stacks with different stacking orders. Placing slow dies close to the heat sink helps achieve large MTTF of stacks.

**Rule-of-thumb:** To optimize reliability of a 3DIC, the slowest dies should be located closest to the heat sink in the stack.

The rule-of-thumb can further reduce the complexity of the stacking optimization problem, since for a stack with fixed composition, the reliability-aware optimal stacking order can be fixed according to the rule-of-thumb. In other words, for a stack whose input dies are given, instead of enumerating all permutations (stacking orders), the optimal stacking style is defined by the rule-of-thumb. Therefore, in a case where input dies are classified into $\boldsymbol{K}$ bins and output stacks are assumed to have $\boldsymbol{L}$ tiers, the number of stacking styles that need to be considered for reliability-driven stacking optimization can be reduced from $K^L$ to $\binom{K+L-1}{L}$.

### 4.3.2 Problem formulation

Given ***N*** dies which are classified into ***K*** bins, we want to determine the optimal stacking style for each output stack that contains ***L*** tiers. Our experimental results show that power consumption mainly depends on composition of the stack. We observe that for a particular number of given input dies, the power consumption of output stacks exhibits only slight differences ($< 1\%$) across different stacking orders, while the difference in MTTF can be up to 16% for 5-die implementations. Therefore, we only focus on optimization for reliability in our studies. Three exemplary reliability-driven stacking optimization problems are formulated as follows.

**Formulation 1: *OPT_MTTF*.**

One objective of reliability-driven 3D stacking optimization is to maximize the sum of MTTFs of output stacks ($MTTF_{sum}$), where a required frequency ($f_{req}$) is predefined as a constraint for dies in a stack. AVS is applied to achieve the same performance across dies. In other words, in a 3DIC, due to interdie process variation, each die has a particular supply voltage corresponding to its process variation. The problem that searches for the optimal stacking style of each stack can be formulated as follows.

*OPT_MTTF:* **Given *N*** dies, each of which is classified into one of the ***K*** process bins
**Maximize** $MTTF_{sum}$
**such that** frequency of each die in a stack $= f_{req}$

**Formulation 2: *OPT_YIELD*.**

We may also optimize the minimum MTTF ($MTTF_{min}$) among all output stacks to improve the yield of 3DICs with respect to a particular reliability (MTTF) requirement. In this scenario, MTTF constraints are predefined for 3DICs, and when constraints are not satisfied, the failed 3DICs are scrapped. The objective for optimization is to maximize the number of good stacks.

*OPT_YIELD*: **Given *N*** dies, each of which is classified into one of the ***K*** process bins
**Maximize** Number of good stacks
**such that** frequency of each die in a stack $= f_{req}$
MTTF of each good stack $\geq MTTF_{req}$

Note that we can maximize the minimum MTTF over all stacks by performing binary search over $MTTF_{req}$, until the number of good stacks equals to the number of all stacks (i.e., $N/L$).

**Formulation 3: *OPT_PERFORMANCE*.**

We also formulate a reliability-driven stacking optimization problem to improve the performance ($f_{stack}$) of 3DICs where reliability constraints are applied, e.g., by setting a lower bound MTTF ($MTTF_{req}$) on 3DICs.

*OPT_PERFORMANCE*: **Given** ***N*** dies, each of which is classified into one of the ***K*** process bins
**Maximize** $f_{stack}$
**such that** MTTF of each stack $\geq$ $MTTF_{req}$.

### 4.3.3 Methodology

#### ILP-Based Method

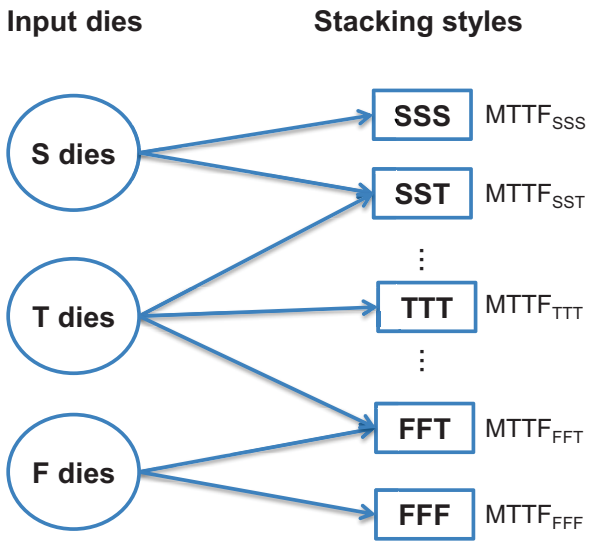



**Figure 4.29**: Allowed assignments in ILP-based stacking optimization method.

We propose an ILP-based method for reliability-driven stacking optimization. As mentioned in Section 4.3.2, inputs of such optimization are ***N*** dies that are classified into ***K*** bins, while outputs are stacks such that each stack has ***L*** tiers. We create matching relationships between input dies and feasible stacking styles of output stacks. This is conceptually shown in Figure 4.29. Vertices on the left part of the bipartite graph indicate the available pools/populations of input dies which are classified into process bins. Feasible stacking styles are enumerated on the right part of the bipartite graph. In the graph, all input dies classified into a particular process bin are connected to the stacking styles containing dies belonging to that bin. The relationships between input dies and stacking styles define the assignment constraints in the ILP formulation. Such constraints indicate that each die can be used exactly once. During the assignments, the

process bins should be consistent between the composition of stacking styles and the dies used (e.g., a die belonging to the Slow bin cannot be assigned to the stacking style "FFT"). Each stacking style corresponds to a MTTF estimated from simulation. In the ILP-based method, we optimally assign input dies to stacking styles to maximize the sum of MTTFs of output stacks. We give the notations and formulate the ILP as follows.

**Notations:**

1. $Die_i$ $(i = 1, 2, \ldots, N)$: input dies
2. $Style_j$ $(j = 1, 2, \ldots, M)$: feasible stacking styles (based on the rule-of-thumb, $M = \binom{K+L-1}{L}$)
3. $Bin_q$ $(q = 1, 2, \ldots, K)$: process bins
4. $X_q$ $(q = 1, 2, \ldots, K)$: number of input dies that are classified into $Bin_q$, such that $\sum_{1 \le q \le K} X_q = N$
5. $Y_{q,j}$ $(q = 1, 2, \ldots, K;\ j = 1, 2, \ldots, M)$: number of dies that are classified to $Bin_q$ contained in $Style_j$, such that $\forall j \sum_{1 \le q \le K} Y_{q,j} = L$
6. $MTTF_j$ $(j = 1, 2, \ldots, M)$: MTTF of the stack implemented with $Style_j$
7. $C_j$ $(j = 1, 2, \ldots, M)$: number of output stacks implemented with $Style_j$, where $L \cdot \sum_{1 \le j \le K} C_j = N$.

**ILP formulation ($OPT\_MTTF$, $OPT\_PERFORMANCE$):**

$$\text{Maximize} \sum_{1 \le j \le M} MTTF_j \cdot C_j \tag{4.11a}$$

$$\text{Subject to} \sum_{1 \le j \le M} C_j \cdot Y_{q,j} = X_q,\ \forall q \tag{4.11b}$$

$$C_j \ge 0,\ \forall j \tag{4.11c}$$

This formulation is used to solve the *OPT_MTTF* and *OPT_PERFORMANCE* problems. In the formulation, (4.11a) gives the objective of maximizing the sum of MTTFs of output stacks; (4.11b) are the assignment constraints, which indicate that each input die should be used exactly once in the stacking implementation, consistent with its process bin; and (4.11c) are the non-negativity constraints which indicate that the number of output stacks implemented with stacking style $Style_j$ cannot be negative. An additional loop, which searches for the maximum frequency, is applied to solve the *OPT_PERFORMANCE* problem.

To solve the *OPT_YIELD* problem, we set up the following ILP.

**ILP formulation ($OPT\_YIELD$):**

$$\text{Maximize} \sum_{1 \le j \le M} C_j \tag{4.12a}$$

$$\text{Subject to} \sum_{1 \le j \le M} C_j \cdot Y_{q,j} = X_q, \ \forall q \tag{4.12b}$$

$$C_j \ge 0, \ \forall j \tag{4.12c}$$

$$C_j \cdot MTTF_j \ge C_j \cdot MTTF_{min} \tag{4.12d}$$

where (4.12a) is the objective function which maximizes the number of good dies; (4.12b) are the assignment constraints; (4.12c) are the non-negativity constraints; and (4.12d) are the lower-bound constraints on MTTF, in which $MTTF_{min}$ indicates the lower bound. Note that the factor $C_j$ in (4.12d) eliminates the constraints on the MTTF for stacking styles which are not implemented (i.e., when $C_j = 0$, the MTTF of $Style_j$ does not affect $MTTF_{min}$). We determine the maximum value of $MTTF_{min}$ by doing binary search. The binary search terminates when the change in $MTTF_{min}$ is less than 0.01 year.

### Greedy Method

We also study a greedy method, based on process binning, for reliability-driven stacking optimization. We evaluate MTTFs of all stacking styles. Then, we select the stacking style with maximum MTTF for each stack, one at a time. A stacking style is valid only if the numbers of dies required by the stacking style are less than or equal to the remaining dies in process bins.

### "Zig-zag" Heuristic Method

The rule-of-thumb proposed in Section 4.3.1 suggests that slower dies should be located closer to the heat sink. Based on the rule-of-thumb, we propose a heuristic method which stacks dies in a "zig-zag" manner as shown in Figure 4.30.

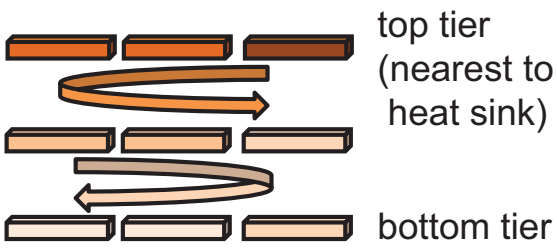



**Figure 4.30**: Zig-zag method: stack dies from slow to fast, from top tiers to bottom tiers.

Given the input dies, we sort them according to their performance (as measured at manufacturing test). Then, we assign the sorted dies (starting from the slowest die) one at a time, from top tiers to bottom tiers. For die assignment in each tier, we record the sequence of die assignment. The sequence is reversed when we start the assignment for the next tier. In this way, all output stacks satisfy the rule-of-thumb proposed in Section 4.3.1. The time complexity of this method is $O(n \cdot log\ n)$ ($n$ indicates the number of input dies), which is required for die sorting. As we will discuss in Section 4.3.4, the zig-zag stacking method offers similar or even better QoR compared to the ILP-based method.

### 4.3.4 Experimental Results

#### Implementation Tools

Our experiments use RTL design *JPEG* obtained from the *OpenCores* website [230] as the logic die. The design is implemented using 65$nm$ NVT, LVT and HVT libraries. The RTL is synthesized using *Synopsys Design Compiler vC-2009.06-SP2* [237] and then placed and routed using *Cadence Encounter Digital Implementation System v10.1* [217]. We characterize all libraries at different corners (SS, TT and FF), for a range of voltages (0.8$V$-1.2$V$) and temperatures (45°$C$-165°$C$) using *Cadence Library Characterizer v9.1* [217]. Timing analyses and power estimation are performed using *Synopsys PrimeTime C2009.6* [240]. We estimate the temperature of stacks using *Hotspot 5.02* [221] and solve ILPs using *lp_solve 5.5* [227].

#### Hotspot Configuration

In *Hotspot*, we set the chip thickness as 50$\mu m$, convection capacitance as 140.4$J/K$, convection resistance as 0.7$K/W$, ambient temperature as 60°$C$, the thickness of heat spreader and heat sink as 1$mm$ and 6.9$mm$ respectively [174][221]. Based on number of I/O pins ($\sim$100 per die), we set the spreader side and the heat sink side as 15$mm$ and 30$mm$, respectively, for 5-tier stacks. We model TSVs by changing the thermal resistivity of *thermal interface material* layers [150]. The thermal resistivity of such a layer is set as 0.2$mK/W$.

#### Estimation of Stacks' MTTF

We implement a flow deploying voltage-temperature feedback loops to estimate the MTTF of an output stack or a stacking style. A change in temperature will change performance. Thus, voltages are altered to retain the required performance. This in turn results in

a change in the temperature, and again affects frequencies. Taking such a "chicken-egg" chain into consideration, an accurate estimation of MTTF requires a feedback loop in the analysis flow, as illustrated in Figure 4.31. The inputs to the flow are stacking styles, the required frequency and an initial temperature (ambient temperature). Then, a voltage-temperature feedback loop is applied to each tier. To avoid large execution time resulting from running simulation for power and timing analysis in each loop, we build lookup tables and apply interpolation to estimate the supply voltage and power consumption.

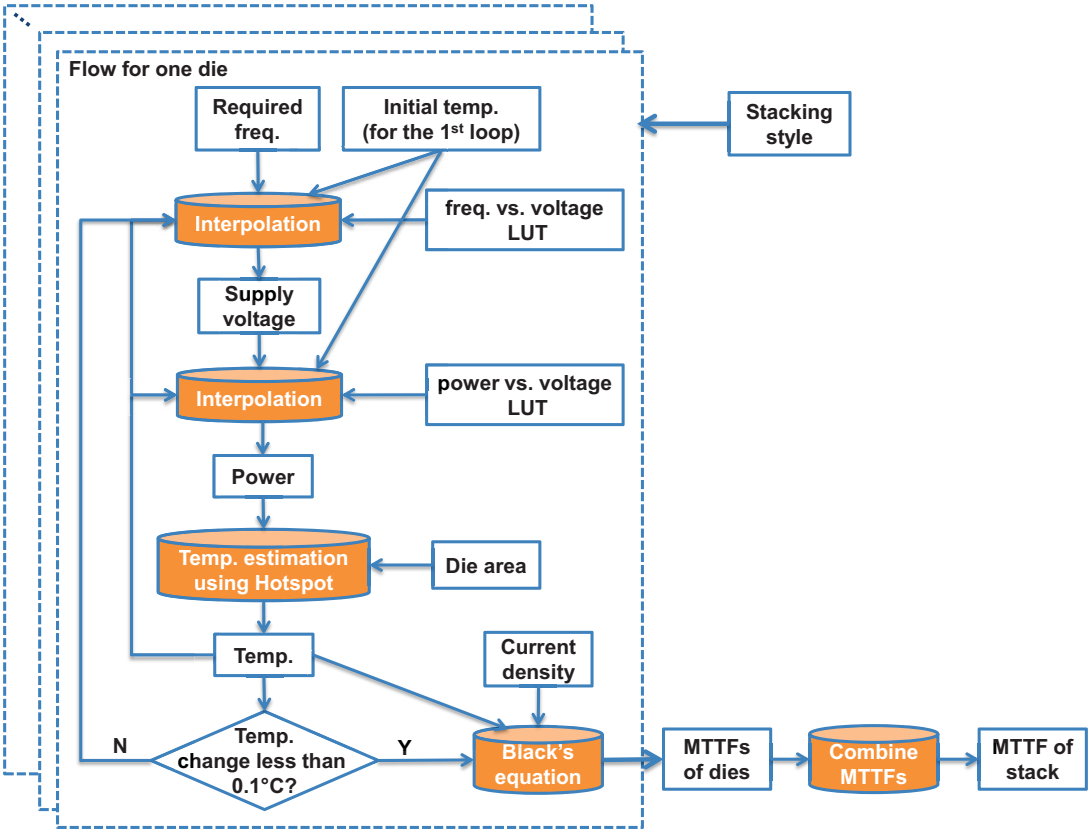



**Figure 4.31**: The flow of MTTF estimation.

First, based on the required frequency and temperature either from input (for the first loop) or from the *Hotspot* simulation, we estimate the required supply voltage.

Second, based on the supply voltage we estimate the power consumption of each die. According to the estimated power and area of each die, *Hotspot* is used to estimate each die's temperature. Then, we check the temperature change with respect to the previous loop. If the change in temperature is less than $0.1°C$, the loop converges. Otherwise, the impact of temperature change on frequencies is estimated, and another loop is applied. Based on the output temperature from the voltage-temperature feedback loop, together with the current density, we

use Black's equation (4.6) to evaluate the MTTF of each die. We calculate the MTTF of the entire stack using the model proposed in Section 4.3.1.

**Design of Experiments**

We implement experiments on the *JPEG* circuit [230] in TSMC $65nm$ technology. We assume that the process variation distributions of input dies are Gaussian, where the SS corner and FF corner of TSMC $65nm$ technology are at $\pm 3\sigma$. For the bin-based model of process variation, we implement 30 trials of picking dies randomly from process bins to stack 3DICs. We observe that the variation in results are small ($< 1\%$), so we only show the average results of the 30 trials in the following discussion.

In the experiments, we compare QoR of four different stacking methods including the ILP-based method (ILP), the zig-zag heuristic method (Zig-zag), the Greedy method (Greedy) and a reference case where no stacking optimization is applied (Random). Three problems formulated in Section 4.3.2 are studied.

*OPT_MTTF.* We implement four methods to optimize the sum of MTTFs of output stacks (*Cases 1-6* in Table 4.8). The average MTTF of stacks resulting from four methods are compared.

*OPT_YIELD.* We implement four methods to optimize the minimum MTTF of output stacks. We apply different MTTF limitations on output stacks and compare the number of good stacks resulting from different optimization methods. Such experiments are implemented on *Case 5* shown in Table 4.8.

*OPT_PERFORMANCE.* We implement stacking optimization to improve performance of stacks under the reliability constraints. For 3-tier cases (*Case 1* in Table 4.8), the lower bound on MTTF is set as 12 years; for 4-tier and 5-tier cases (*Cases 2-6* in Table 4.8), such constraints are set as 10 years.

**Table 4.8**: Experiment design for reliability-driven stacking optimization.

| Case | #Dies | #Tiers | $\sigma$ | $\mu$ | #Bins |
|---|---|---|---|---|---|
| 1 | 1200 | 3 | 1.0 | 0.0 | 9 |
| 2 | 1600 | 4 | 1.0 | 0.0 | 9 |
| 3 | 2000 | 5 | 0.2 | 0.0 | 9 |
| 4 | 2000 | 5 | 0.6 | 0.0 | 9 |
| 5 | 2000 | 5 | 1.0 | 0.0 | 9 |
| 6 | 2000 | 5 | 1.4 | 0.0 | 9 |

### Results for Optimization Problems

**Results for *OPT_MTTF*.** We study the impacts of bin-based modeling and the number of input dies on QoR of the ILP-based method to solve the *OPT_MTTF* problem.

Figure 4.32 shows the average MTTF of output stacks resulting from the ILP-based method modeled with different number of bins. We observe that as the number of bins increases, better MTTF is achieved. With certain number of bins (e.g., 13 in this case), the solution approaches optimality and noise occurs afterwards (e.g., number of bins $\geq$ 13 in this case).

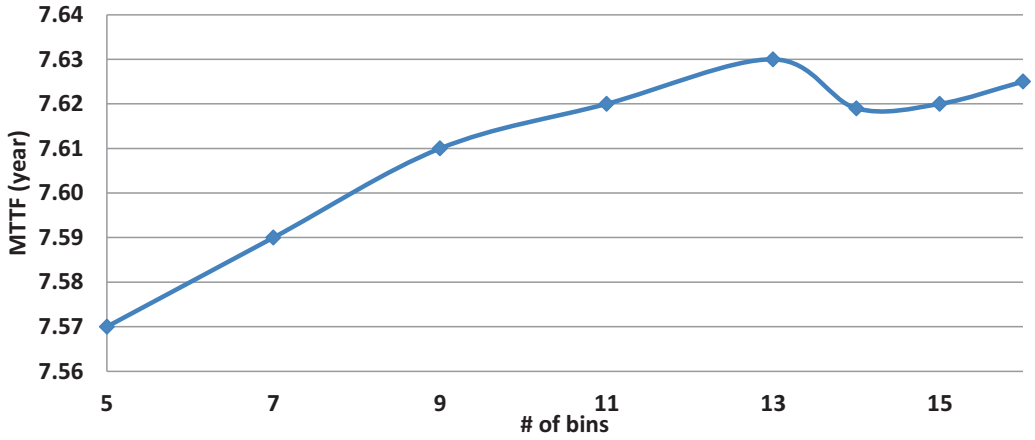



**Figure 4.32**: As the number of process bins increases, MTTF of stacks increases. The results approach optimality when the number of bins is equal to 13, noise appears after that.

Table 4.9 shows the QoR of the ILP-based method with different numbers of input dies and the number of process bins equal to 9. We observe that as the number of dies increases, the average MTTF of output stacks increases. This is because the degradation induced by discretization in bin-based modeling reduces as number of dies increases. We also use the zig-zag heuristic method as reference and observe that (for this experiment, where the number of bins is 9) the zig-zag heuristic method always performs better than the ILP-based method.

From the results in Table 4.10 we observe that the ILP-based and the zig-zag heuristic methods offer $\sim$7% improvement in the average MTTF ($MTTF_{avg}$) compared to the random method where no optimization is applied.

**Results for *OPT_YIELD*.** Table 4.10 shows that the ILP-based and the zig-zag heuristic methods achieve $\sim$28% improvement in the minimum MTTF ($MTTF_{min}$). In addition, the ILP-based and zig-zag heuristic methods also reduce the variation in MTTFs of stacks, which is illustrated in Figure 4.33. Among the four methods, the greedy method leads to large variation in MTTFs of output stacks.

**Table 4.9**: Impact of number of dies on QoR of the ILP-based method.

| #Dies | QoR | ILP | Zig-zag |
|---|---|---|---|
| 200 | MTTF ($year$) | 7.57 | 7.58 |
| | **Power** (mW) | 533.6 | 533.6 |
| | **Runtime** | 86 $min$ | <1 $sec$ |
| 500 | MTTF ($year$) | 7.58 | 7.59 |
| | **Power** ($mW$) | 533.4 | 533.4 |
| | **Runtime** | 86 $min$ | <1 $sec$ |
| 2000 | MTTF ($year$) | 7.61 | 7.63 |
| | **Power** ($mW$) | 530.9 | 531.0 |
| | **Runtime** | 86 $min$ | <1 $sec$ |
| 10000 | MTTF ($year$) | 7.63 | 7.65 |
| | **Power** ($mW$) | 530.6 | 530.6 |
| | **Runtime** | 86 $min$ | <1 $sec$ |
| 100000 | MTTF ($year$) | 7.63 | 7.65 |
| | **Power** ($mW$) | 530.8 | 530.8 |
| | **Runtime** | 86 $min$ | <1 $sec$ |

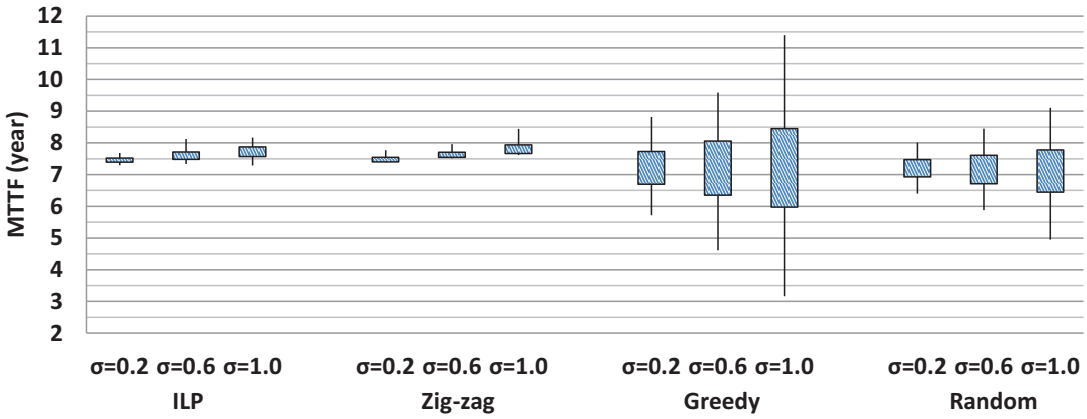



**Figure 4.33**: Stacking optimization using the ILP-based method and the zig-zag method helps increase the minimum MTTF of output stacks, while reducing the variation in MTTFs.

Figure 4.34 shows the yield of stacks constrained by different $MTTF_{req}$ using the four methods. We observe that the improvement in yield can be up to 300% (when MTTF limitation = 7.5 years) by using the zig-zag heuristic method, compared to the random case.

**Results for *OPT_PERFORMANCE*.** Table 4.10 shows that the ILP-based and the zig-zag heuristic methods offer $\sim$3% improvement in performance compared to the random case.

**Table 4.10**: QoR of output stacks from different methods.

| **Case** | | ILP | Zig-zag | Greedy | Random |
|---|---|---|---|---|---|
| 1 | $MTTF_{avg}$ $(year)$ | 11.20 | 11.20 | 10.37 | 10.31 |
| | $MTTF_{min}$ $(year)$ | 10.42 | 10.78 | 6.02 | 7.20 |
| | **Power** $(mW)$ | 319.4 | 319.4 | 318.8 | 318.8 |
| | $f_{max}$ $(MHz)$ | 975.0 | 975.4 | 943.4 | 943.4 |
| | **Runtime** | 11 $min$ | <1 $sec$ | 11 $min$ | – |
| 2 | $MTTF_{avg}$ $(year)$ | 9.85 | 9.88 | 9.29 | 9.23 |
| | $MTTF_{min}$ $(year)$ | 9.47 | 9.66 | 5.91 | 7.11 |
| | **Power** $(mW)$ | 424.8 | 424.9 | 424.2 | 424.2 |
| | $f_{max}$ $(MHz)$ | 993.7 | 995.4 | 966.4 | 966.5 |
| | **Runtime** | 33 $min$ | <1 $sec$ | 33 $min$ | – |
| 3 | $MTTF_{avg}$ $(year)$ | 7.30 | 7.30 | 7.22 | 7.22 |
| | $MTTF_{min}$ $(year)$ | 7.23 | 7.27 | 6.71 | 6.98 |
| | **Power** $(mW)$ | 527.7 | 527.7 | 527.4 | 527.4 |
| | $f_{max}$ $(MHz)$ | 860.3 | 862.3 | 857.1 | 856.0 |
| | **Runtime** | 86 $min$ | <1 $sec$ | 86 $min$ | – |
| 4 | $MTTF_{avg}$ $(year)$ | 7.47 | 7.47 | 7.22 | 7.20 |
| | $MTTF_{min}$ $(year)$ | 7.30 | 7.39 | 5.72 | 6.40 |
| | **Power** $(mW)$ | 528.9 | 528.9 | 528.1 | 528.2 |
| | $f_{max}$ $(MHz)$ | 867.3 | 869.7 | 854.7 | 853.1 |
| | **Runtime** | 86 $min$ | <1 $sec$ | 86 $min$ | – |
| 5 | $MTTF_{avg}$ $(year)$ | 7.61 | 7.63 | 7.21 | 7.16 |
| | $MTTF_{min}$ $(year)$ | 7.34 | 7.51 | 4.61 | 5.88 |
| | **Power** $(mW)$ | 530.9 | 531.0 | 530.1 | 530.2 |
| | $f_{max}$ | 875.1 | 876.7 | 851.8 | 849.2 |
| | **Runtime** | 86 $min$ | <1 $sec$ | 86 $min$ | – |
| 6 | $MTTF_{avg}$ $(year)$ | 7.78 | 7.80 | 7.21 | 7.12 |
| | $MTTF_{min}$ $(year)$ | 7.29 | 7.62 | 3.16 | 5.05 |
| | **Power** $(mW)$ | 533.3 | 533.4 | 532.5 | 532.7 |
| | $f_{max}$ $(MHz)$ | 884.0 | 886.0 | 849.1 | 844.3 |
| | **Runtime** | 86 $min$ | <1 $sec$ | 86 $min$ | – |

### Suboptimality of the Zig-zag Heuristic Method

Although experimental results show that the zig-zag heuristic method performs better than other methods, it is still suboptimal for an adversarial example. Given 6 input dies

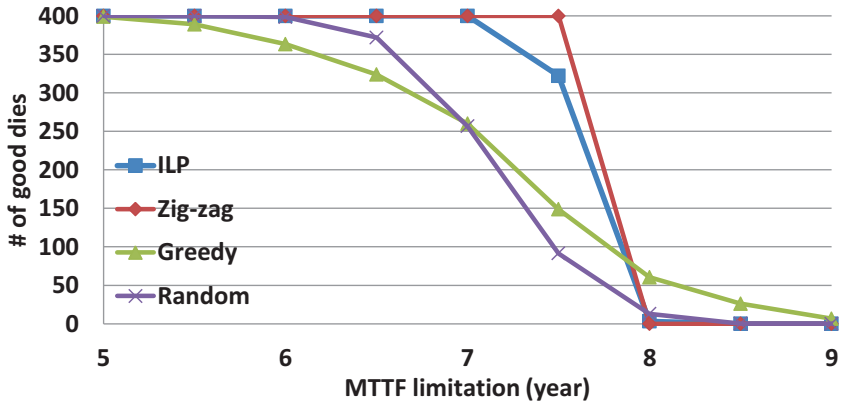



**Figure 4.34**: Yield decreases with MTTF limitation. The ILP-based and the zig-zag heuristic methods help increase the yield of 3DICs compared to the random case.

($die_{\{1...6\}}$), and each output stack has 3 tiers. Without loss of generality, we assume that $die_i$ is faster than $die_j$, when $i > j$. Further, we assume large performance difference between $die_5$ and $die_6$. In this example, the output stacks resulting from the zig-zag heuristic method are "$die_6$ $die_3$ $die_2$" and "$die_5$ $die_4$ $die_1$". Due to the large performance difference between $die_5$ and $die_6$, the bottom two tiers of stack "$die_5$ $die_4$ $die_1$" generate more heat than the bottom two tiers of stack "$die_6$ $die_3$ $die_2$". If we swap $die_1$ and $die_2$, the $MTTF_{min}$ of output stacks is higher. On the other hand, due to the nonlinear relationship between temperature and MTTF, the stacks "$die_5$ $die_3$ $die_1$" and "$die_6$ $die_4$ $die_2$" can achieve better $MTTF_{avg}$ compared to stacks resulting from the zig-zag heuristic method, at the cost of having larger MTTF variation for output stacks.

### Variability Helps

The experimental results show that when no stacking optimization is applied, the MTTF of output stacks decreases as process variation increases. However, when stacking optimization is applied, MTTF increases with process variation. This trend is illustrated in Figure 4.35, in which the solid lines indicate the average MTTF and the dotted lines indicate the minimum MTTF of output stacks with different process variation distributions. When the $\sigma$ of process variation distribution changes from 0.2 to 0.6, the improvement in the average MTTF changes from 1.1% to 9.6%, while the improvement in the minimum MTTF changes from 4.2% to 50.9%, where the zig-zag heuristic method is applied. A similar benefit from process variation is observed in [121], where process variation with a proposed matching solution helps to reduce clock skew in 3DICs. In other words, manufacturing variation helps improve the average MTTF and the minimum MTTF with the zig-zag stacking optimization.

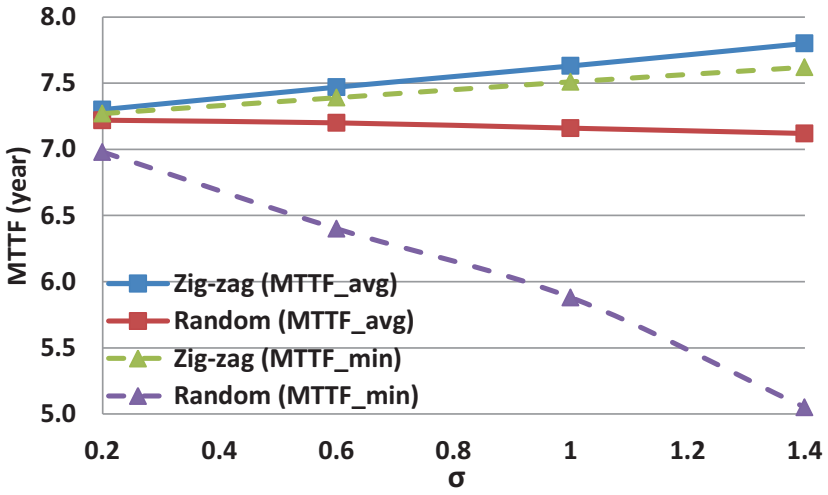



**Figure 4.35**: The solid lines and dotted lines indicate the average and the minimum MTTF of stacks, respectively.

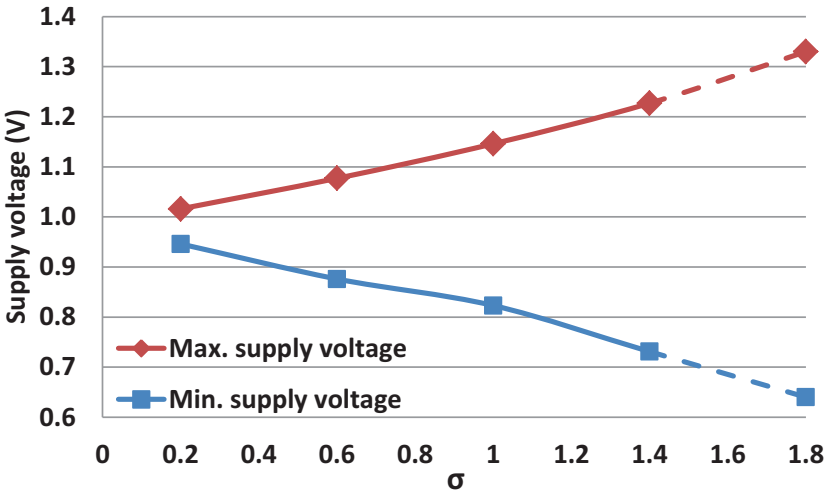



**Figure 4.36**: The maximum supply voltage of stacks increases with process variation, while the minimum voltage decreases. The solid line corresponds to our experimental results. The dashed line is an extrapolation of the trend.

The benefit from process variation disappears when the variation exceeds a certain amount. This is because supply voltages of slow dies can exceed the maximum voltage allowed by the package as the process variation keeps increasing. Figure 4.36 shows the increase (decrease) of the maximum (minimum) supply voltage with process variation. If the package can only tolerate up to $1.3V$ supply voltage, the help of variability in stacking optimization will stop when $\sigma$ is close to 1.7. Therefore, we conclude that a limited amount of manufacturing variation can "help" improve reliability of die stacks when stacking optimization is applied. In other words, the reliability benefit of stacking optimization depends on the magnitude of die-to-die process variation.

### 4.3.5 Conclusion

In this work, we study variability-reliability interactions and optimizations in 3DIC. We propose a "rule-of-thumb" guideline for stacking optimization to reduce the peak temperature and increase the MTTF of 3DICs. An ILP-based method and an $O(n\ log\ n)$ zig-zag heuristic method for reliability-driven stacking optimization achieve $\sim 7\%$, $\sim 28\%$ and $\sim 3\%$ improvement in average MTTF, minimum MTTF and performance (under reliability constraints) of 3DICs, respectively, compared to the case where no optimization is applied. Optimization of yield of 3DICs under reliability requirements is also implemented. Interestingly, our studies show that *a limited amount of variability can "help"* to improve reliability of stacks when stacking optimization is applied.

## 4.4 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 [29][64][72][98] have pointed out that "mix-and-match" of multiple stacked die, according to binning information, can improve overall product yield.[76] Without loss of generality, assuming that dies are classified into two process bins, SS and FF, the example in Figure 4.37 (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 75$ps$ timing improvement for a small 28$nm$ FDSOI block as compared to the conventional worst-case analysis.[77] 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.

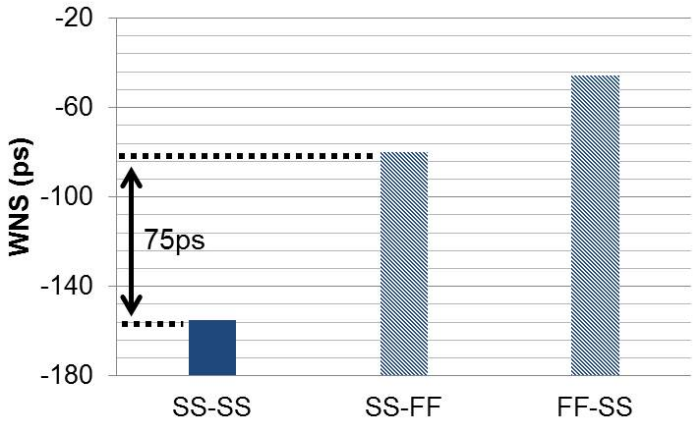



**Figure 4.37**: Worst negative slack (WNS) of design *AES* [230] at 28FDSOI technology. Clock period = 1.2$ns$. The *AES* implementation was simply bipartitioned for minimum net cut using MLPart [25][229].

Separately, many works [45][90][96][132][156][182] have suggested approaches for partitioning of logic into multiple die, e.g., to obtain the wirelength (hence, power and delay)

[76] 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.

[77] 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 }.[78] 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 4.38 (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.

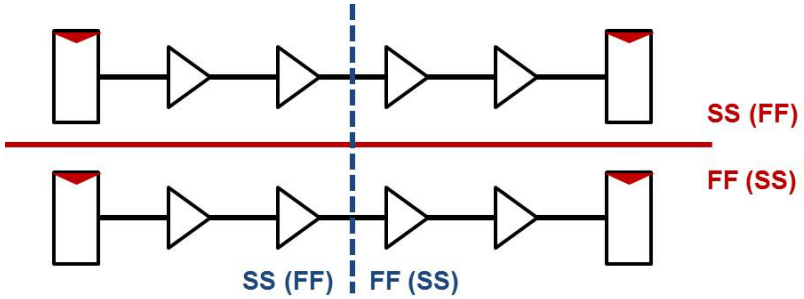



**Figure 4.38**: Partitioning solutions affect a design's performance in the regime of mix-and-match stacking.

In this work, 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

[78] 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., $3\sigma$ FF + $2\sigma$ SS) as discussed in [102] will also increase the difficulty of the partitioning optimization. Figure 4.39 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 4.39(d). In Figure 4.39, 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 work 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.

### 4.4.1 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. [64] 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. [72] formulate mathematical programs to improve the performance yield of 3DICs via mix-and-match die stacking. Chan et al. [29] propose an integer linear programming-based method as well as a heuristic method to optimize reliability of 3DICs (i.e., to improve the

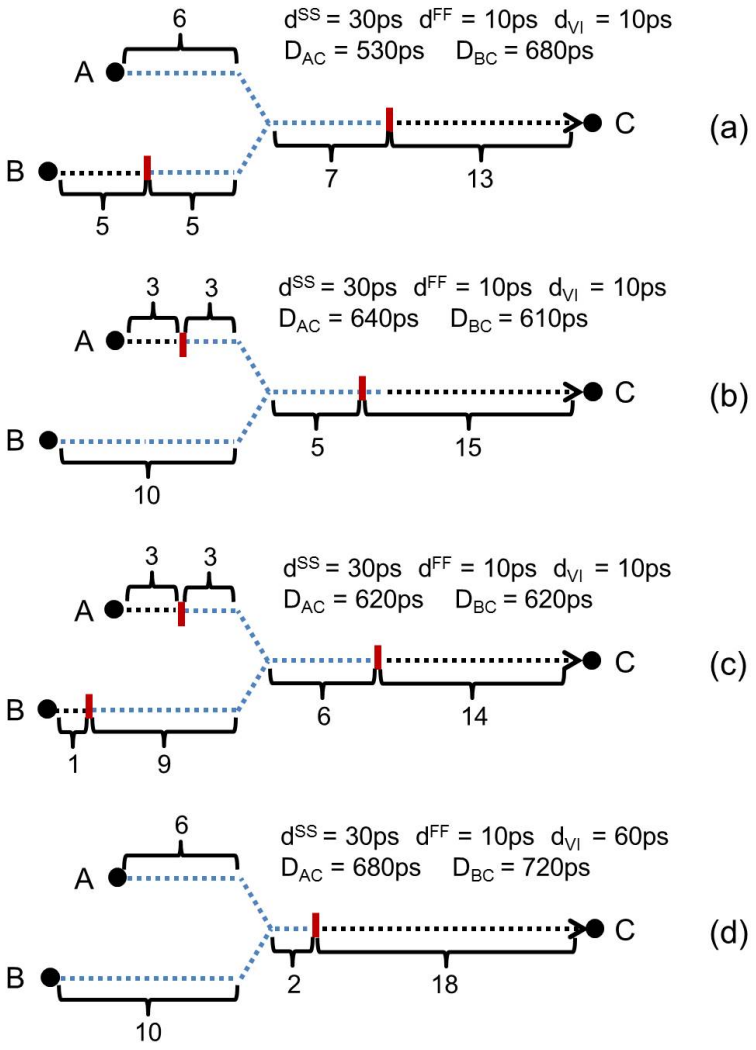



**Figure 4.39**: Area-balanced partitioning solutions on path A-C (26 stages) and path B-C (30 stages) which respectively minimize (a) delay of path A-C ($D_{AC}$), (b) delay of path B-C ($D_{BC}$), (c) worst-case delay over the two paths, and (d) worst-case delay over the two paths with large VI delay impact ($d_{VI}$).

mean time to failure). To avoid the large runtime of thermal simulation, Juan et al. [98] 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 contract, 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. [132] 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 hyper-

graph partitioning. Thorolfsson et al. [182] use hMetis [116] 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 [96], 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. [45] 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 x), Panth et al. [156] 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.

### 4.4.2 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.[79]

In the next section, we describe an ILP-based partitioning methodology which is able to achieve near-optimal solutions. Section 4.4.4 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.

### 4.4.3 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 4.11 summarizes our notations.

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

**Table 4.11**: Description of notations used in our work.

| Term | Meaning |
|---|---|
| $\alpha_j$ | 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 $\alpha_j$ |
| $D_k$ | 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 |
| $\theta$ | area balancing criterion |

Minimize $D_{max}$

Subject to

$$\beta_{i,i'} \geq y_{i'} - y_i \qquad \forall \text{ adjacent cells } c_i, c_{i'} \in C \tag{4.13}$$

$$\beta_{i',i} \geq y_i - y_{i'} \qquad \forall \text{ adjacent cells } c_i, c_{i'} \in C \tag{4.14}$$

$$\beta_{i,i'} + \beta_{i',i} \leq 1 \qquad \forall \text{ adjacent cells } c_i, c_{i'} \in C \tag{4.15}$$

$$\sum_{c_i \in p_k} (d_i^j \cdot (1 - y_i) + d_i^{j'} \cdot y_i) + \sum_{adjacent\ c_i, c_{i'} \in p_k} (\Delta_{i'}^{j,j'} \cdot \beta_{i,i'} + \Delta_{i'}^{j',j} \cdot \beta_{i',i})$$
$$+ \sum_{adjacent\ c_i, c_{i'} \in p_k} (\beta_{i,i'} + \beta_{i',i}) \cdot d_{VI} \leq D_k \quad \forall (\alpha_j, \alpha_{j'}),\ p_k \in P \tag{4.16}$$

$$D_k \leq D_{max} \qquad \forall p_k \in P \tag{4.17}$$

$$\sum_{c_i \in C} a_i \cdot y_i - \sum_{c_i \in C} a_i \cdot (1 - y_i) \leq \theta \cdot \sum_{c_i \in C} a_i \tag{4.18}$$

$$\sum_{c_i \in C} a_i \cdot (1 - y_i) - \sum_{c_i \in C} a_i \cdot y_i \leq \theta \cdot \sum_{c_i \in C} a_i \tag{4.19}$$

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 (4.13) and (4.14) 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 (4.16) 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 (4.16) is the sum of stage delays along path $p_k$. We extract stage delays at a particular corner $\alpha_j$ based on the timing analysis assuming all cells are at $\alpha_j$. 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 $\alpha_j$, and $c_{i'}$ is at $\alpha_{j'}$) versus the case where the $c_i$ and $c_{i'}$ are at the same process corner.[80] 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.[81] The third term on the left-hand side of Constraint (4) 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 (4.17), 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 $\theta$ in Constraints (4.18) and (4.19). We set $\theta$ as 5% in our experiments.

[80] 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.

[81] 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.

### 4.4.4 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.[82] 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 [116] or the number of paths passing across different partitions [115], 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, few if any previous works have applied a semidefinite program-based maximum cut optimization [76] 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

$$l'_k \cdot \frac{D_k^{SS}}{l_k} + (l_k - l'_k) \cdot \frac{D_k^{FF}}{l_k} \tag{4.20}$$

$$l'_k \cdot \frac{D_k^{FF}}{l_k} + (l_k - l'_k) \cdot \frac{D_k^{SS}}{l_k} \tag{4.21}$$

where (4.20) assumes the stacking of SS Tier 0 + FF Tier 1, and (4.21) assumes the stacking of FF Tier 0 + SS Tier 1. Maximizing the minimum value between (4.20) and (4.21) corresponds to

[82] 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.

having (4.20) = (4.21) 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.[83] We note that when 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 20. 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 [76] 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 [234] as our semidefinite programming solver.

[83] 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 $s_{gb}$.

**Algorithm 20** 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 $G_1$.

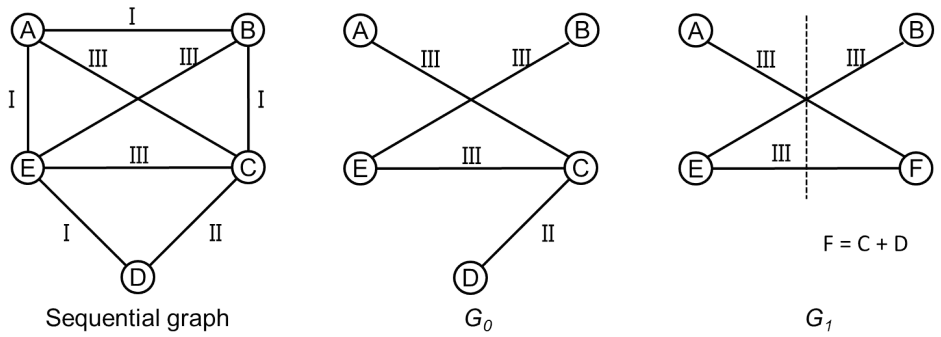



**Figure 4.40**: Example of maximum-cut partitioning of the sequential graph. Types of paths are shown in edge labels. The dotted line indicates the final maximum-cut solution. We assume the same weight for all edges.

Figure 4.40 illustrates Algorithm 20 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 [66] 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

$$gain(u) = \frac{\Delta slack(u)}{slack(u) - WNS} \tag{4.22}$$

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 4.41 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.[84] 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 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.

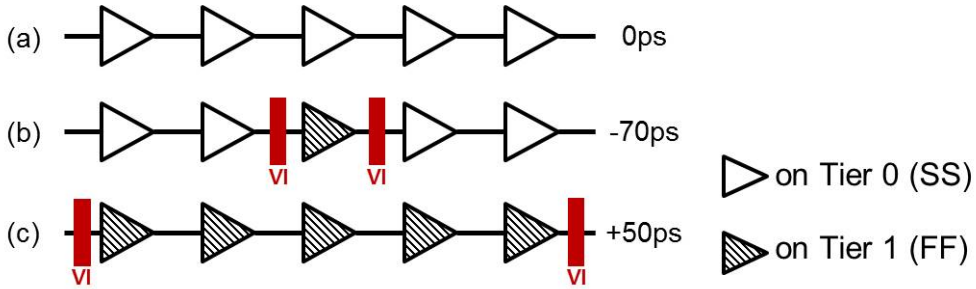



**Figure 4.41**: Example to optimize a cell with a negative gain value. (a) Initial path with zero slack. (b) Moving one cell to Tier 1 degrades the slack by $70ps$ due to VI insertions. (c) Further optimization on the shown segment improves the slack by $50ps$.

Algorithm 21 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

[84] We are aware of "lookahead" approaches, such as gain vectors, CLIP/CDIP and LIFO gain buckets, etc. [56][82][124]. However, these are cut-centric and not path-aware, hence inapplicable to our current problem.

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 4.42 shows one example). Assuming that the incremental timing analysis is performed in constant time,[85] the runtime complexity of our clustering algorithm is $O(|C|^3)$.

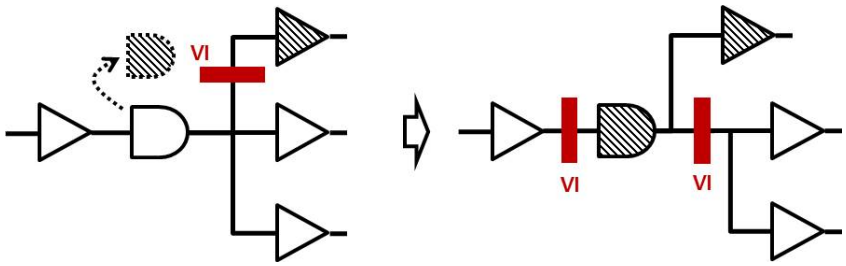



**Figure 4.42**: Example of VI insertion/removal due to cell movement across tiers. Shaded cells are on Tier 1 and the others are on Tier 0.

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 4.43, 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

[85] 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 21** Clustering.

```
1: cell_list ← sort all cells in increasing order of their slacks
2: for all c ∈ cell_list that is not clustered do
3:    queue.push_front(c); u ← ∅ // initialize cluster u
4:    while |u| < N_ub do
5:       s' ← −∞; c' ← ∅; queue' ← ∅
6:       while |queue| > 0 do
7:          c ← queue.pop_front()
8:          s_u ← slack(u); s_c ← slack(c)
9:          move c to a different tier; incremental timing analysis
10:         if |u| == 0 || slack(u) ≥ s_u && slack(c) ≥ s_c then
11:            u ← u ∪ {c}
12:            if |u| ≥ N_ub then
13:               break
14:            end if
15:            queue.push_back(neighbors of c that are not clustered)
16:         else
17:            if Min(slack(c), slack(u)) > s' then
18:               s' ← min(slack(c), slack(u)); c' ← c
19:            end if
20:            queue'.push_back(c)
21:            recover c to its original tier; incremental timing analysis
22:         end if
23:      end while
24:      if |u| ≥ N_ub || |queue'| == 0 then
25:         break
26:      end if
27:      move c' to a different tier; incremental timing analysis
28:      u ← u ∪ {c'}
29:      queue.push_back(neighbors of c' that is not clustered)
30:      queue.push_back(queue'); queue' ← ∅
31:   end while
32: end for
```

we terminate the optimization when the area balancing criterion is met. Figure 4.43 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 -200$ps$ to -14$ps$ in this example with ∼3000 moves.

### 4.4.5 Experimental Results

#### Experimental Setup

Our partitioning methodologies for mix-and-match stacking are implemented in C++. We use *CPLEX v12.5* [223] as our ILP solver and *SDPA* [234] as our semidefinite programming solver. Our SP&R (synthesis, placement and routing) flow uses *Synopsys Design Compiler vH-*

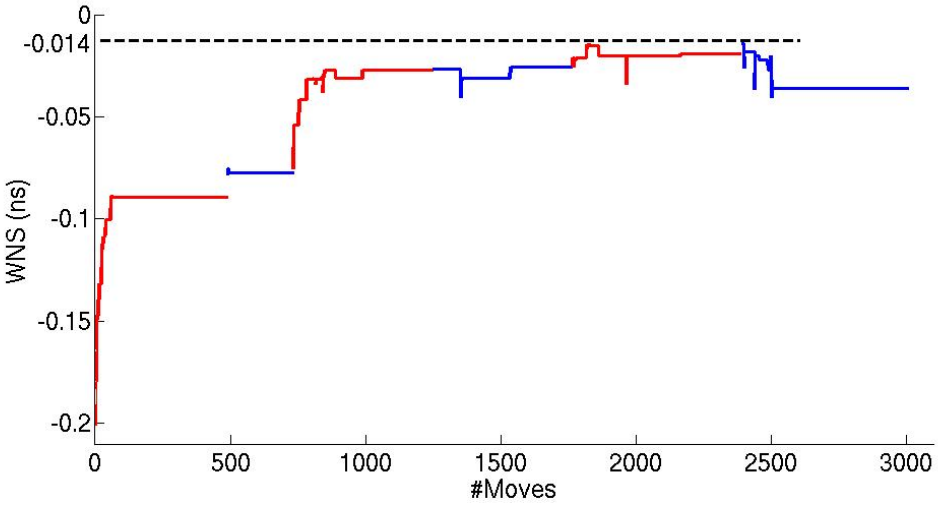



**Figure 4.43**: An example of our multi-phase FM optimization. Design: *AES*. Technology: 28FDSOI. WNS improves from -200$ps$ to -14$ps$. Runtime = 565 seconds on a 2.5$GHz$ Intel Xeon server.

*2013.03-SP3* [237], *Cadence Encounter Digital Implementation System v12.0* [217], *Synopsys PrimeTime vH-2013.06-SP2* [240] for logic synthesis, P&R, and timing and power analyses, respectively. Similarly to [155], 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) [230] and an *ARM CORTEX M0* in our experiments. These testcases are generated with foundry 28$nm$ FDSOI 12-track, dual-$V_{th}$ libraries. We use a BEOL stack of six metal layers for routing.

**Table 4.12**: Testcases used in the experiments.

| **Design** | **#Instances** | **Clock period** ($ns$) |
|---|---|---|
| *DMA* | ∼2K | 0.6 |
| *USB* | ∼4K | 0.8 |
| *ARM 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 [99]. (iii) We further assess the benefit from our partitioning method within a state-of-the-art 3DIC implementation flow (Shrunk2D) [155]. 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 22.[86] 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.

**Algorithm 22** GT2012 3DIC implementation flow.

1: Netlist partitioning (MLPart [229] 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;

We also use the 3DIC implementation flow in [155] to validate our partitioning method. The flow first performs 2D implementation with scaled (i.e., 0.7 x) cell sizes and floorplan. 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

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

are forced to be on Tier 0. Following [155], 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.[87]

### Experimental Results

**Table 4.13**: Validation of our partitioning methodology on GT2012 and Shrunk2D flows.

| Design | Flow | WNS ($ps$) | TNS ($ns$) | Area ($\mu m^2$) | Power ($mW$) | #Instances | Wirelength ($\mu m$) | #VIs | Utilization (**bottom / top**) |
|---|---|---|---|---|---|---|---|---|---|
| *ARM 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% |

**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., { $3\sigma$ SS + $3\sigma$ FF, $2\sigma$ SS + $3\sigma$ FF, $3\sigma$ SS + $2\sigma$ FF }).

[87] 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.

Comparison results in Figure 4.44 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.

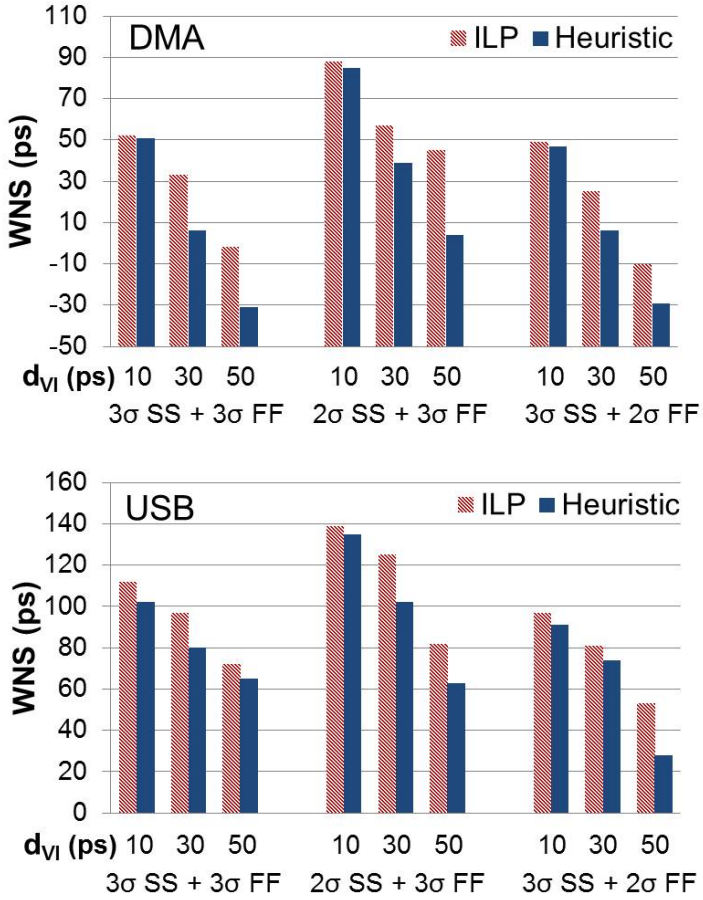



**Figure 4.44**: Comparison of solution qualities between the ILP-based method (which is near-optimal) and the heuristic method.

**Validation of Our Method on GT2012 Flow.** Table 4.13 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 [25][229], 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 4.13 shows design metrics of implementations using the original Shrunk2D flow [155] 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 *ARM CORTEX M0*) with similar area, power and wirelength. Note that to maintain the solution of the 2D implementation in the scaled floorplan, we include 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.

### 4.4.6 Conclusion

In this work, 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.

## 4.5 Acknowledgments

Chapter 4 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; Sorin Dobre, Andrew B. Kahng and Jiajia Li, "Mixed Cell-Height Implementation for Improved Design Quality in Advanced Nodes", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2015; Tuck-Boon Chan, Andrew B. Kahng and Jiajia Li, "NOLO: A No-Loop, Predictive Useful Skew Methodology for Improved Timing in IC Implementation", *Proc. International Symposium on Quality Electronic Design*, 2014; and Tuck-Boon Chan, Andrew B. Kahng and Jiajia Li, "Reliability-Constrained Die Stacking Order in 3DICs under Manufacturing Variability", *Proc. International Symposium on Quality Electronic Design*, 2013. Chapter 4 also contains the draft submitted to *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Sorin Dobre, Andrew B. Kahng and Jiajia Li, "Design Implementation with Non-Integer Multiple-Height Cells for Improved Design Quality in Advanced Nodes". The dissertation author is the primary author of the papers and the submitted draft.

I would like to thank my coauthors Tuck-Boon Chan, Sorin Dobre, Kwangsoo Han and Andrew B. Kahng.

# Chapter 5

# Conclusion and Future Directions

This thesis has presented innovative physical design optimization techniques and signoff methodologies to address pressing challenges faced by current and future SoC designs. The methods presented enable design teams and product companies to realize design-based equivalent scaling for improved PPAC tradeoffs.

Chapter 2 presents three distinct multi-mode multi-corner optimization methodologies that respectively optimize signoff corner selection, skew variation in a clock tree, and scan (shift) timing. Based on the properties of equivalent dominance, the chapter presents guidelines and efficient methodologies to search for the optimal modes for multi-mode signoff. The proposed methodologies can successfully determine the signoff modes that reduce lifetime energy, and are shown to achieve up to $> 8\%$ performance improvement compared to the traditional "signoff and scale" approach. The chapter also presents the first framework to minimize the sum of skew variations over all sequentially adjacent sink pairs, using both global and local optimizations. 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, a linear programming formulation comprehends ECO feasibility based on characterized lookup tables of stage delays. In the local optimization, the chapter demonstrates that machine learning-based predictors of latency changes can provide accurate estimation of local move impacts. Last, the chapter presents a comprehensive scan timing optimization that can be deployed during late-stage IC implementation without causing any function-mode solution quality degradation. Specifically, the chapter presents two optimization approaches: (i) scan reordering that removes hold buffers along scan chains with awareness of clock skew and scan cell locations, and (ii) gating insertion to minimize the DVD impact

on scan timing slack. These optimizations achieve up to 82% hold buffer reduction and 58% improvement of scan timing degradation due to DVD.

Chapter 3 addresses the low-power imperative in modern SoC designs through three optimization techniques. First, the chapter presents a comprehensive optimization framework for stacked power-domain implementation with maximized battery lifetime. The approach extends the existing flow-based partitioning methodology with layout- and timing-path-awareness, as well as a multi-scenario balancing objective. It further uses an iterative grid movement and a dynamic programming-based boundary optimization to define the layout region (power island) of each power domain. Validations are performed in both $28nm$ LP and $40nm$ technologies, as well as on industrial designs. The optimization achieves more than 10% and $2\times$ battery lifetime improvements for function and sleep modes compared to the conventional design. Second, the chapter presents a flop tray-based optimization for improved design power reduction. The optimization uses a capacitated K-means algorithm which iteratively applies a min-cost flow-based clustering and an LP-based flop tray placement. The optimization also includes an ILP-based matching optimization to generate flop trays while minimizing the perturbation to the initial placement solution. The proposed techniques achieve up to 32% total block power reduction relative to designs with only single-bit flops, and up to 16% total block power reduction relative to designs with flop trays generated by logical clustering during synthesis. Last, the chapter presents a new design flow for mixing of resilient and non-resilient circuits within a given implementation, so as to minimize the overhead of error resilience. The new design flow includes (i) a selective-endpoint optimization, which reduces timing-critical endpoints while maintaining small cost of timing optimization; and (ii) a clock skew optimization, specifically targeted to a resilient design methodology, which improves robustness to process, voltage and temperature variations. The proposed optimization techniques achieve significant energy reductions of up to 21% and 10% compared to conventional (pure-margin) design and a brute-force resilience implementation, respectively.

Chapter 4 presents the concept of "mixed-fabric optimization" for physical design and signoff. First, the chapter presents a mixed cell-height optimization flow, which mixes cells with different, non-integer multiple heights in a fine-grained manner within a single place-and-route block. The flow applies a dynamic programming-based partitioning to define regions for different cell heights, and performs iterative displacement and/or cell height swapping to achieve a legal placement solution. By comprehending the "breaker cell" overheads of the mixed-height placement, the optimization achieves 30+ percent area and power reductions while maintain-

ing performance, as compared to a 12T-only design flow in 28LP technology. Moreover, the optimized mixed-height designs can achieve significant performance increase along with area and power reductions as compared to designs with 8T-only cells. Second, the chapter presents NOLO, a "no-loop" predictive useful skew optimization flow, based on timing information (with dual-$V_{th}$ libraries) of a post-synthesis netlist. The predictive useful skew flow achieves similar or better total negative slack compared to back-annotation flows, with only one pass through chip implementation. The runtime of the predictive useful skew flows is similar to the runtime of the typical (i.e., without useful skew optimization) flow, which is approximately 66% less than the runtime of the back-annotation flow in [191]. Last, the chapter presents two studies of mix-and-match die stacking optimization in 3DICs. To help product companies better navigate future variability-reliability interactions and optimizations, the chapter proposes a "rule-of-thumb" guideline for stacking optimization that reduces the peak temperature and increases the MTTF of 3DICs. An ILP-based method and an $O(n\ log\ n)$ zig-zag heuristic method for reliability-driven stacking optimization achieve $\sim$7%, $\sim$28% and $\sim$3% improvement in average MTTF, minimum MTTF and performance (under reliability constraints) of 3DICs, respectively, compared to the case where no optimization is applied. Furthermore, the chapter develops a design-stage optimization for mix-and-match die stacking. An ILP-based partitioning methodology and a heuristic partitioning methodology perform maximum cut on the timing-critical sequential graph followed by an iterative multi-phase FM optimization. The optimization yields 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.

Looking beyond this thesis, future directions and ongoing works include the following.

- 3DIC with multiple tiers is a promising technology in the "More-than-Moore" era to integrate more functionality with greater bandwidth and less power. However, due to inter-tier process variation, delay and area penalties from vertical interconnects, and high thermal density, physical design and signoff for 3DICs are challenging. Moreover, there is no production-quality 3D design tool/flow available. Therefore, physical design (especially, multi-tier partitioning and clock network synthesis) and signoff methodologies must be considered as key future directions beyond this thesis.

- Many alternative low-power computing techniques (e.g., approximate computing and stochastic computing) are promising for specific applications such as neural networks, image processing and communication. Further, IC design for these alternative computing techniques typically introduces new constraints and requirements (e.g., [1]) that are unseen

in conventional designs; these require new physical design and signoff methodologies. Hence, physical design and signoff must go beyond traditional PPAC tradeoffs: it must comprehend and address the additional tradeoffs in these alternative computing designs such as the accuracy-power or accuracy-area tradeoff in approximate computing, and the tradeoffs among power (or area), accuracy and computation latency in stochastic computing.

- New interconnect and device technologies typically yield new objectives for traditional P&R optimizations. For instance, while wirelength minimization and routing congestion reduction are the main objectives in the traditional routing optimization, signal loss minimization, which is affected by waveguide crossings and bends, is a primary objective for on-chip optical interconnect routing [44]. Therefore, new physical design techniques are needed to enable the usage of new interconnect and device technologies in IC designs. As an example, a recent work [47] performs floorplan optimization for silicon photonic NoCs (Networks-on-Chip) in many-core systems.

- Last, there remain many rich opportunities to apply machine learning (ML) techniques to physical design and signoff optimizations. Armed with relatively accurate modeling and prediction of design flow outcomes in late-design stages, an ML-based predictive flow can break the "chicken-and-egg" loops that pervade today's physical design flows, thus reducing design turnaround time and improving design quality. Moreover, many steps in the physical design flow (such as floorplanning, and post-routing timing closure) cannot be optimally solved due to the large runtime complexity. Therefore, many heuristics and trial-and-error techniques are applied in current physical design and signoff flows. To systematically optimize the design and leverage a design organization's or an EDA tool's previous optimization results (experience), reinforcement learning might be a promising framework.

# Bibliography


[1] A. Alaghi, W.-T. J. Chan, J. P. Hayes, A. B. Kahng and J. Li, "Optimizing Stochastic Circuits for Accuracy-Energy Tradeoffs", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2015, pp. 178-185.

[2] C. Albrecht, *personal communication*, July 2013.

[3] C. Albrecht, "Efficient Incremental Clock Latency Scheduling for Large Circuits", *Proc. Design, Automation and Test in Europe*, 2006, pp. 6-10.

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

[5] C. Albrecht, B. Korte, J. Schietke and J. Vygen, "Cycle Time and Slack Optimization for VLSI-Chips", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 1999, pp. 232-237.

[6] C. Albrecht, P. Witte and A. Kuehlmann, "Performance and Area Optimization using Sequential Flexibility", *Proc. International Workshop on Logic and Synthesis*, 2004.

[7] 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.

[8] C. J. Alpert and A. B. Kahng, "Recent Directions in Netlist Partitioning: A Survey", *Integration, the VLSI Journal* 19(1-2) (1995), pp. 1-81.

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

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

[11] T. M. Austin, "DIVA: A Reliable Substrate for Deep Submicron Microarchitecture Design", *Proc. IEEE/ACM International Symposium on Microarchitecture*, 1999, pp. 196-207.

[12] T. Austin, V. Bertacco, D. Blaauw and T. Mudge, "Opportunities and Challenges for Better Than Worst-Case Design", *Proc. Asia and South Pacific Design Automation Conference*, 2005, pp. 2-7.

[13] N. D. P. Avirneni, V. Subramanian and A. K. Somani, "Low Overhead Soft Error Mitigation Techniques for High-Performance and Aggressive Systems", *Proc. International Conference on Dependable Systems & Networks*, 2009, pp. 185-194.

[14] J. Bhasker and R. Chadha, *Static Timing Analysis for Nanometer Designs: A Practical Approach*, Springer, 2009.

[15] J. R. Black, "Electromigration – A Brief Survey and Some Recent Results", *IEEE Transactions on Electron Devices* 16(4) (1969), pp. 338-347.

[16] B. Black, M. Annavaram, N. Brekelbaum, J. DeVale, L. Jiang, G. H. Loh, D. McCauley, P. Morrow, D. W. Nelson, D. Pantuso, P. Reed, J. Rupley, S. Shankar, J. Shen and C. Webb, "Die Stacking (3D) Microarchitecture", *Proc. IEEE/ACM International Symposium on Microarchitecture*, 2006, pp. 469-479.

[17] K. Blutman, NXP Semiconductors, *personal communication*, June 2016.

[18] K. Blutman, H. Fatemi, A. B. Kahng, A. Kapoor, J. Li and J. Pineda de Gyvez, "Floorplan and Placement Methodology for Improved Energy Reduction in Stacked Power-Domain Design", *Proc. Asia and South Pacific Design Automation Conference*, 2017, pp. 444-449.

[19] K. Blutman, A. Kapoor, A. Majumdar, J. G. Martinez, J. Echeverri, L. Sevat, A. van der Wel, H. Fatemi, J. Pineda de Gyvez and K. Makinwa, "A Microcontroller with 96% Power-Conversion Efficiency using Stacked Voltage Domains", *Proc. IEEE Symposium on VLSI Circuits*, 2016, pp. 1-2.

[20] K. Blutman, A. Kapoor, J. G. Martinez, H. Fatemi and J. Pineda de Gyvez, "Lower Power by Voltage Stacking: A Fine-grained System Design Approach", *Proc. ACM/EDAC/IEEE Design Automation Conference*, 2016, pp. 78:1-78:5.

[21] Y. Bonhomme, P. Girard, L. Guiller, C. Landrault, S. Pravossoudovitch and A. Virazel, "Design of Routing-Constrained Low Power Scan Chains", *Proc. Design, Automation and Test in Europe*, 2004, pp. 62-67.

[22] K. A. Bowman, J. W. Tschanz, N. S. Kim, J. C. Lee, C. B. Wilkerson, S. L. Lu, T. Karnik and V. K. De, "Energy-Efficient and Metastability-Immune Resilient Circuits for Dynamic Variation Tolerance", *IEEE Journal of Solid State Circuits* 44(1) (2009), pp. 49-63.

[23] K. Bowman, J. Tschanz, C. Wilkerson, S.-L. Lu, T. Karnik, V. De and S. Borkar, "Circuit Techniques for Dynamic Variation Tolerance", *Proc. ACM/IEEE Design Automation Conference*, 2009, pp. 4-7.

[24] A. C. Cabe, Z. Qi and M. R. Stan, "Stacking SRAM Banks for Ultra Low Power Standby Mode Operation", *Proc. ACM/IEEE Design Automation Conference*, 2010, pp. 699-704.

[25] 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.

[26] A. E. Caldwell, A. B. Kahng and I. L. Markov, "Optimal Partitioners and End-Case Placers for Standard-Cell Layout", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 19(11) (2000), pp. 1304-1313.

[27] A. Cao, S.-M. Chang and D.-C. Yuan, "Local Clock Skew Optimization", *U.S. Patent* No. US8635579B1, January 2014.

[28] 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.

[29] 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.

[30] T.-B. Chan, A. B. Kahng, J. Li and S. Nath, "Optimization of Overdrive Signoff", *Proc. Asia and South Pacific Design Automation Conference*, 2013, pp. 344-349.

[31] W.-T. J. Chan, A. B. Kahng, S. Nath and I. Yamamoto, "The ITRS MPU and SOC System Drivers: Calibration and Implications for Design-Based Equivalent Scaling in the Roadmap", *Proc. IEEE International Conference on Computer Design*, 2014, pp. 153-160.

[32] V. Chandra, ARM and M. Choudhury, IBM, *personal communication*, June 2014.

[33] L. Chen, A. Hung, H. M. Chen, E. Tsai, S. H. Chen, M. H. Ku and C. C. Chen, "Using Multi-bit Flip-Flop for Clock Power Saving by Design Compiler", *Proc. Synopsys Users Group Conference*, 2010.

[34] H. Chen, S. Roy and K. Chakraborty, "DARP: Dynamically Adaptable Resilient Pipeline Design in Microprocessors", *Proc. Design, Automation and Test in Europe*, 2014, pp. 1-6.

[35] C.-H. Chen, Y. Tao and Z. Zhang, "Efficient In Situ Error Detection Enabling Diverse Path Coverage", *Proc. IEEE International Symposium on Circuits and Systems*, 2013, pp. 773-776.

[36] R. L. S. Ching, E. F. Y. Young, K. C. K. Leung and C. Chu, "Post-Placement Voltage Island Generation", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2006, pp. 641-646.

[37] 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.

[38] C. B. Cho, W. Zhang and T. Li, "Thermal Design Space Exploration of 3D Die Stacked Multi-core Processors Using Geospatial-Based Predictive Models", *Proc. SPEC Benchmark Workshop on Computer Performance Evaluation and Benchmarking*, 2009, pp. 102-120.

[39] 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.

[40] M. Choudhury, V. Chandra, K. Mohanram and R. Aitken, "TIMBER: Time Borrowing and Error Relaying for Online Timing Error Resilience", *Proc. Design, Automation and Test in Europe*, 2010, pp. 1554-1559.

[41] M. R. Choudhury and K. Mohanram, "Masking Timing Errors on Speed-Paths in Logic Circuits", *Proc. Design, Automation and Test in Europe*, 2009, pp. 87-92.

[42] C. Chu, "FLUTE: Fast Lookup Table Based Wirelength Estimation Technique", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2004, pp. 696-701.

[43] R. L. Clay, "Exascale Computing Systems R&D at Sandia", talk at Texas A&M University, September 27, 2011, *http://www.cs.tamu.edu/tref/clay*

[44] C. Condrat, P. Kalla and S. Blair, "Channel routing for integrated optics", *Proc. ACM International Workshop on System-Level Interconnect Prediction*, 2013 pp. 1-8.

[45] 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.

[46] B. D. Cory, R. Kapur and B. Underwood, "Speed Binning with Path Delay Test in 150-nm Technology", *IEEE Design and Test of Computers* 20(5) (2003), pp. 41-45.

[47] A. Coskun, A. Gu, W. Jin, A. J. Joshi, A. B. Kahng, J. Klamkin, Y. Ma, J. Recchio, V. Srinivas and T. Zhang, "Cross-Layer Floorplan Optimization For Silicon Photonic NoCs In Many-Core Systems", *Proc. Design, Automation and Test in Europe*, 2016, pp. 1309-1314.

[48] A. Cui, T. Yu, G. Qu and M. Li, "An Improved Scan Design for Minimization of Test Power under Routing Constraint", *Proc. IEEE International Symposium on Circuits and Systems*, 2015, pp. 629-632.

[49] S. Das, D. Roberts, S. Lee, S. Pant, D. Blaauw, T. Austin, K. Flautner and T. Mudge, "A Self-Tuning DVS Processor Using Delay-Error Detection and Correction", *IEEE Journal of Solid State Circuits* 41(4) (2006), pp. 792-804.

[50] S. Das, C. Tokunaga, S. Pant, W.-H. Ma, S. Kalaiselvan, K. Lai, D. M. Bull and D. T. Blaauw, "Razor II: In Situ Error Detection and Correction for PVT and SER Tolerance", *Proc. International Solid State Circuits Conference*, 2008, pp. 400-622.

[51] 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.

[52] R. B. Deokar and S. S. Sapatnekar, "A Graph-theoretic Approach to Clock Skew Optimization", *Proc. IEEE International Symposium on Circuits and Systems*, 1994, pp. 407-410.

[53] S. Devadas and S. Malik, “A Survey of Optimization Techniques Targeting Low Power VLSI Circuits”, *Proc. ACM/IEEE Design Automation Conference*, 1995, pp. 242-247.

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

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

[56] 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.

[57] M. Elgebaly, K. Z. Malik, L. G. Chua-Eoan and S. Jung, “Adaptive Voltage Scaling for an Electronics Device”, *U.S. Patent* No. 7417482B2, August 2008.

[58] M. Elgebaly and M. Sachdev, “Variation-Aware Adaptive Voltage Scaling System”, *IEEE Transactions on Very Large Scale Integration Systems* 15(5) (2007), pp. 560-571.

[59] J. A. Ellis, “Embedding Rectangular Grids Into Square Grids”, *IEEE Transactions on Computers* 40(1) (1991), pp. 46-51.

[60] M. Elshoukry, M. Tehranipoor and C. P. Ravikumar, “A Critical-Path-Aware Partial Gating Approach for Test Power Reduction”, *ACM Transactions on Design Automation of Electronic Systems* 12(2) (2007), pp. 17:1-17:22.

[61] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner and T. Mudge, “Razor: A Low-Power Pipeline Based on Circuit-Level Timing Speculation”, *Proc. IEEE/ACM International Symposium on Microarchitecture*, 2003, pp. 7-18.

[62] H. Esmaeilzadeh, E. Blem, R. S. Amant, K. Sankaralingam and D. Burger, “Dark Silicon and the End of Multicore Scaling”, *Proc. International Symposium on Computer Architecture*, 2011, pp. 365-376.

[63] S. Fabrie, NXP Semiconductors, *personal communication*, May–July 2014.

[64] C. Ferri, S. Reda and R. I. Bahar, “Parameter Yield Management for 3DICs: Models and Strategies for Improvement”, *ACM Journal on Emerging Technologies in Computing Systems* 4(4) (2008), pp. 19:1-19:22.

[65] M. Feuer and C. C. Koo, “Method for Rechaining Shift Register Latches Which Contain More Than One Physical Book”, *IBM Technical Disclosure Bulletin* 25(9) (1983), pp. 4818-4820.

[66] C. M. Fiduccia and R. M. Mattheyses, “A Linear Time Heuristic for Improving Network Partitions”, *Proc. ACM/IEEE Design Automation Conference*, 1982, pp. 175-181.

[67] J. P. Fishburn, “Clock Skew Optimization”, *IEEE Transactions on Computers* 39(7) (1990), pp. 945-951.

[68] M. Fojtik, D. Fick, Y. Kim, N. Pinckney, D. Harris, D. Blaauw and D. Sylvester, "Bubble Razor: An Architecture-Independent Approach to Timing-Error Detection and Correction", *Proc. International Solid State Circuits Conference*, 2012, pp. 488-489.

[69] M. Fojtik, D. Fick, Y. Kim, N. Pinckney, D. M. Harris, D. Blaauw and D. Sylvester, "Bubble Razor: Eliminating Timing Margins in an ARM Cortex-M3 Processor in 45 nm CMOS Using Architecturally Independent Error Detection and Correction", *IEEE Journal of Solid State Circuits* 48(1) (2013), pp. 66-81.

[70] P. D. Franzon, W. R. Davis, M. B. Steer, H. Hao, S. Lipa, S. Luniya, C. Mineo, J. Oh, A. Sule and T. Thorolfsson, "Design for 3D Integration and Applications", *Proc. International Symposium on Signals, Systems and Electronics*, 2007, pp. 263-266.

[71] E. G. Friedman, *Clock Distribution Networks in VLSI Circuits and Systems*, New York, IEEE Press, 1995.

[72] 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.

[73] S. Gerstendorfer and H.-J. Wunderlich, "Minimized Power Consumption for Scan-Based BIST", *Journal of Electronic Testing* 16(3) (2000), pp. 203-212.

[74] S. Ghosh and K. Roy, "CRISTA: A New Paradigm for Low-Power and Robust Circuit Synthesis Under Parameter Variations Using Critical Path Isolation", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 26(11) (2007), pp. 1947-1956.

[75] F. Glover and M. Laguna, *Tabu Search*, Kluwer Academic Publishers, 1999.

[76] 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.

[77] B. Greskamp and J. Torrellas, "Paceline: Improving Single-Thread Performance in Nanoscale CMPs through Core Overclocking", *Proc. International Conference on Parallel Architectures and Compilation Techniques*, 2007, pp. 213-224.

[78] B. Greskamp, L. Wan, W. R. Karpuzcu, J. J. Cook, J. Torrellas, D. Chen and C. Zilles, "BlueShift: Designing Processors for Timing Speculation from the Ground Up", *Proc. IEEE Symposium on High Performance Computer Architecture*, 2009, pp. 213-224.

[79] L. Guo, Y. Cai, Q. Zhou and X. Hong, "Logic and Layout Aware Voltage Island Generation for Low Power Design", *Proc. Asia and South Pacific Design Automation Conference*, 2007, pp. 666-671.

[80] P. Gupta, A. B. Kahng and S. Mantik, "Routing-Aware Scan Chain Ordering", *Proc. Asia and South Pacific Design Automation Conference*, 2003, pp. 857-862.

[81] P. Gupta, A. B. Kahng and S. Mantik, "A Proposal for Routing-Based Timing-Driven Scan Chain Ordering", *Proc. International Symposium on Quality Electronic Design*, 2003, pp. 339-343.

[82] 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.

[83] K. Han, A. B. Kahng, J. Lee, J. Li and S. Nath, "A Global-Local Optimization Framework for Simultaneous Multi-Mode Multi-Corner Clock Skew Variation Reduction", *Proc. ACM/EDAC/IEEE Design Automation Conference*, 2015, pp. 26:1-26:6.

[84] 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.

[85] T. Hastie, R. Tibshirani and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer, 2009.

[86] M. Hirech, J. Beausang and X. Gu, "A New Approach to Scan Chain Reordering Using Physical Design Information", *Proc. International Test Conference*, 1998, pp. 1089-3539.

[87] W. Hou, D. Liu and P.-H. Ho, "Automatic Register Banking for Low-Power Clock Trees", *Proc. International Symposium on Quality Electronic Design*, 2009, pp. 647-652.

[88] C.-C. Hsu, Y.-T. Chang and M. P.-H. Lin, "Crosstalk-Aware Power Optimization with Multi-Bit Flip-Flops", *Proc. Asia and South Pacific Design Automation Conference*, 2012, pp. 431-436.

[89] L.-C. Hsu and H.-M. Chen, "On Optimizing Scan Testing Power and Routing Cost in Scan Chain Design", *Proc. International Symposium on Quality Electronic Design*, 2006, pp. 451-456.

[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, M. C. Fu and S. I. Marcus, "A Model Reference Adaptive Search Method for Global Optimization", *Operations Research* 55(3) (2005), pp. 549-568.

[92] 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.

[93] A. Hurst, P. Chong and A. Kuehlmann, "Physical Placement Driven by Sequential Timing Analysis", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2004, pp. 379-386.

[94] D. Jayaraman, R. Sethuram and S. Tragoudas, "Gating Internal Nodes to Reduce Power During Scan Shift", *Proc. Great Lakes Symposium on Very Large Scale Integration*, 2010, pp. 79-84.

[95] K. Jeong and A. B. Kahng, "Methodology From Chaos in IC Implementation", *Proc. International Symposium on Quality Electronic Design*, 2010, pp. 885-892.

[96] I. H.-R. Jiang, "Generic Integer Linear Programming Formulation for 3D IC Partitioning", *Proc. IEEE International SOC Conference*, 2009, pp. 321-324.

[97] I. H.-R. Jiang, C. L. Chang and Y. M. Yang, "INTEGRA: Fast Multibit Flip-Flop Clustering for Clock Power Saving", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31(2) (2012), pp. 192-204.

[98] 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.

[99] M. Jung, *personal communication*, 2013.

[100] A. B. Kahng, "Lithography-Induced Limits to Scaling of Design Quality", *Proc. SPIE*, 2014, pp. 905302-1-905302-14.

[101] A. B. Kahng, "Toward Holistic Modeling, Margining and Tolerance of IC Variability", *Proc. IEEE Computer Society Annual Symposium on VLSI*, 2014, pp. 284-289.

[102] A. B. Kahng, "New Game, New Goal Posts: A Recent History of Timing Closure", *Proc. ACM/EDAC/IEEE Design Automation Conference*, 2015, pp. 1-6.

[103] A. B. Kahng, "PPAC Scaling at 7nm and Below", Cadence Distinguished Speaker Series talk, San Jose, CA, April 7, 2016.

[104] A. B. Kahng, S. Kang, R. Kumar and J. Sartori, "Enhancing the Efficiency of Energy-Constrained DVFS Designs", *IEEE Transactions on Very Large Scale Integration Systems* 21(10) (2013), pp. 1769-1782.

[105] A. B. Kahng, S. Kang, R. Kumar and J. Sartori, "Recovery-driven Design: A Methodology for Power Minimization for Error Tolerant Processor Modules", *Proc. ACM/IEEE Design Automation Conference*, 2010, pp. 825-830.

[106] A. B. Kahng, S. Kang, R. Kumar and J. Sartori, "Slack Redistribution for Graceful Degradation Under Voltage Overscaling", *Proc. Asia and South Pacific Design Automation Conference*, 2010, pp. 825-831.

[107] A. B. Kahng, S. Kang, H. Lee, I. L. Markov and P. Thapar, "High-Performance Gate Sizing with a Signoff Timer", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2013, pp. 450-457.

[108] A. B. Kahng, S. Kang and J. Li, "A New Methodology for Reduced Cost of Resilience", *Proc. Great Lakes Symposium on Very Large Scale Integration*, 2014, pp. 157-162.

[109] A. B. Kahng, I. Kang and S. Nath, "Incremental Multiple-Scan Chain Ordering for ECO Flip-Flop Insertion", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2013, pp. 705-712.

[110] A. B. Kahng, H. Lee and J. Li, "Measuring Progress and Value of IC Implementation Technology", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2016, pp. 27:1-27:8.

[111] A. B. Kahng, J. Lienig, I. L. Markov and J. Hu, *VLSI Physical Design: From Graph Partitioning to Timing Closure*, Springer, 2011.

[112] 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.

[113] A. B. Kahng, B. Lin and S. Nath, "Explicit Modeling of Control and Data for Improved NoC Router Estimation", *Proc. ACM/EDAC/IEEE Design Automation Conference*, 2012, pp. 392-397.

[114] 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.

[115] 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.

[116] G. Karypis and V. Kumar, "Multilevel K-Way Hypergraph Partitioning", *Proc. ACM/IEEE Design Automation Conference*, 1999, pp. 343-348.

[117] G. Karypis and V. Kumar, "A Fast and High Quality Multilevel Scheme for Partitioning Irregular Graphs", *SIAM Journal on Scientific Computing* 20(1) (1998), pp. 359-392.

[118] 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.

[119] B. W. Kernighan and S. Lin, "An Efficient Heuristic Procedure for Partitioning Graphs", *Bell Labs Technical Journal* 49(2) (1970), pp. 291-307.

[120] S. Khuller and Y. J. Sussmann, "The Capacitated K-Center Problem", *SIAM Journal on Discrete Mathematics* 13(3) (2000), pp. 403-418.

[121] T. Y. Kim and T. Kim, "Post Silicon Management of On-Package Variation Induced 3D Clock Skew", *Journal of Semiconductor Technology and Science* 12(2) (2012), pp. 139-149.

[122] S. Kim, I. Kwon, D. Fick, M. Kim, Y.-P. Chen and D. Sylvester, "Razor-Lite: A Side-Channel Error-Detection Register for Timing-Margin Recovery in 45nm SOI CMOS", *Proc. International Solid State Circuits Conference*, 2013, pp. 264-265.

[123] Y. Kretchmer, "Using Multi-Bit Register Inference to Save Area and Power: The Good, The Bad, and The Ugly", *EE Times Asia*, 2001.

[124] B. Krishnamurthy, "An Improved Min-Cut Algorithm for Partitioning VLSI Networks", *IEEE Transactions on Computers* 33(5) (1984), pp. 438-446.

[125] M. W. Kuemerle, S. K. Lichtensteiger, D. W. Douglas and I. L. Wemple, "Integrated Circuit Design Closure Method for Selective Voltage Binning", *U.S. Patent* No. US7475366B2, January 2009.

[126] K. J. Kuhn, "Reducing Variation in Advanced Logic Technologies: Approaches to Process and Design for Manufacturability of Nanoscale CMOS", *Proc. IEEE International Electron Devices Meeting*, 2007, pp. 471-474.

[127] K. J. Kuhn, "CMOS Transistor Scaling Past 32nm and Implications on Variation", *Proc. Advanced Semiconductor Manufacturing Conference*, 2010, pp. 241-246.

[128] R. Kumar and C. P. Ravikumar, "Leakage Power Estimation for Deep Submicron Circuits in an ASIC Design Environment", *Proc. Asia and South Pacific Design Automation Conference*, 2002, pp. 45-50.

[129] E. L. Lawler, J. K. Lenstra, A. Rinnooy-Kan and D. Shmoys, *The Traveling Salesman Problem: A Guided Tour of Combinatorial Optimization*, Wiley, 1985.

[130] J. Lee, S. Narayan, M. Kapralos and M. Tehranipoor, "Layout-Aware, IR-drop Tolerant Transition Fault Pattern Generation", *Proc. Design, Automation and Test in Europe*, 2008, pp. 1172-1177.

[131] S. K. Lee, T. Tong, X. Zhang, D. Brooks and G.-Y. Wei, "A 16-Core Voltage-Stacked System with An Integrated Switched-Capacitor DC-DC Converter", *Proc. Symposium on VLSI Circuits*, 2015, pp. C318-C319.

[132] 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.

[133] J. Lienig, "Introduction to Electromigration-Aware Physical Design", *Proc. ACM International Symposium on Physical Design*, 2006, pp. 39-46.

[134] M. P.-H. Lin, C. C. Hsu and Y.-T. Chang, "Post-Placement Power Optimization with Multi-Bit Flip-Flops", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 30(12) (2011), pp. 1870-1882.

[135] M. P. H. Lin, C. C. Hsu and Y. C. Chen, "Clock-Tree Aware Multibit Flip-Flop Generation During Placement for Power Optimization", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34(2) (2015), pp. 280-292.

[136] L. Y.-Z. Lin, C. C.-H. Liao and C. H.-P. Wen, "Synthesizing Multiple Scan Chains by Cost-Driven Spectral Ordering", *Proc. Asia and South Pacific Design Automation Conference*, 2013, pp. 540-545.

[137] Y. Lin, B. Yu, X. Xu, J.-R. Gao, N. Viswanathan, W.-H. Liu, Z. Li, C. J. Alpert and D. Z. Pan, "MrDP: Multiple-row Detailed Placement of Heterogeneous-sized Cells for Advanced Nodes", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2016, pp. 7:1-7:8.

[138] L. Liu, I. Ganusov, M. Burtscher and S. Tiwari, "Bridging the Processor-Memory Performance Gap with 3D IC Technology", *IEEE Design and Test of Computers* 22(6) (2005), pp. 556-564.

[139] Y. Liu, P.-H. Hsieh, S. Kim, J. Seo, R. Montoye, L. Chang, J. Tierno and D. Friedman, "A 0.1pJ/b 5-to-10Gb/s Charge-Recycling Stacked Low-Power I/O for On-Chip Signaling in 45nm CMOS SOI", *Proc. International Solid State Circuits Conference*, 2013, pp. 400-401.

[140] S. S. Y. Liu, W. T. Lo, C. J. Lee and H. M. Chen, "Agglomerative-Based Flip-Flop Merging and Relocation for Signal Wirelength and Clock Tree Optimization", *ACM Transactions on Design Automation of Electronic Systems* 18(3) (2013), pp. 40:1-40:20.

[141] Y. Liu, R. Ye, F. Yuan, R. Kumar and Q. Xu, "On Logic Synthesis for Timing Speculation", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2012, pp. 591-596.

[142] Y. Liu, F. Yuan and Q. Xu, "Re-Synthesis for Cost-Efficient Circuit-Level Timing Speculation", *Proc. ACM/EDAC/IEEE Design Automation Conference*, 2011, pp. 158-163.

[143] S.-C. Lo, C.-C. Hsu and M. P.-H. Lin, "Power Optimization for Clock Network with Clock Gate Cloning and Flip-Flop Merging", *Proc. ACM International Symposium on Physical Design*, 2014, pp. 77-84.

[144] G. H. Loh, "3D-Stacked Memory Architecture for Multi-core Processors", *Proc. International Symposium on Computer Architecture*, 2008, pp. 453-464.

[145] G. Loi, B. Agarwal, N. Srivastava, S. Lin, T. Sherwood and K. Banerjee, "A Thermally-Aware Performance Analysis of Vertically Integrated (3-D) Processor-Memory Hierarchy", *Proc. ACM/IEEE Design Automation Conference*, 2006, pp. 991-996.

[146] 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.

[147] 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.

[148] N. MacDonald, Broadcom Corp., *personal communication*, June 2013.

[149] 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.

[150] J. Meng, K. Kawakami and A. K. Coskun, "Optimizing Energy Efficiency of 3-D Multicore Systems with Stacked DRAM under Power and Thermal Constraints", *Proc. ACM/EDAC/IEEE Design Automation Conference*, 2012, pp. 648-655.

[151] 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.

[152] G.-J. Nam, IBM, *personal communication*, March 2016.

[153] K. Nose and T. Sakurai, "Analysis and Future Trend of Short-Circuit Power", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 19(9) (2000), pp. 1023-1030.

[154] 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.

[155] 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 Electronics and Design*, 2014, pp. 171-176.

[156] 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.

[157] D. Papa, N. Viswanathan, C. Sze, Z. Li, G.-J. Nam, C. Alpert and I. L. Markov, "Physical Synthesis with Clock-Network Optimization for Large Systems on Chips", *IEEE Micro* 31(4) (2011), pp. 51-62.

[158] N. Parimi, "Leveraging Physically Aware Design-for-Test to Improve Area, Power, and Timing", https://www.cadence.com/rl/Resources/white_papers/PhysicallyAware_DFT_wp.pdf

[159] J. T. Pawlowski, "Hybrid Memory Cube: Breakthrough DRAM Performance with a Fundamentally Re-architected DRAM Subsystem", *HOT Chips: A Symposium on High Performance Chips* 23, 2011, http://www.hotchips.org/wp-content/uploads/hc_archives/hc23/HC23.18.3-memory-FPGA/HC23.18.320-HybridCube-Pawlowski-Micron.pdf

[160] K. Puttaswamy and G. H. Loh, "Thermal Analysis of a 3D Die-Stacked High-Performance Microprocessor", *Proc. Great Lakes Symposium on Very Large Scale Integration*, 2006, pp. 19-24.

[161] R. Radojcic, "Roadmap for Design and EDA Infrastructure for 3D Products", *Electronic Design Processes Workshop*, April 2012.

[162] S. Rajapandian, K. Shepard, P. Hazucha and T. Karnik, "High-Tension Power Delivery: Operating 0.18 $\mu$m CMOS Digital Logic at 5.4V", *Proc. International Solid State Circuits Conference*, 2005, pp. 298-299.

[163] A. Rajaram, J. Hu and R. Mahapatra, "Reducing Clock Skew Variability via Crosslinks", *Proc. ACM/IEEE Design Automation Conference*, 2004, pp. 18-23.

[164] 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.

[165] R. Rajaraman and D. F. Wong, "Optimum Clustering for Delay Minimization", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 14(12) (1995), pp. 1490-1495.

[166] J.-C. Rau, C.-H. Lin and J.-Y. Chang, "An Efficient Low-Overhead Policy for Constructing Multiple Scan-Chains", *Proc. IEEE Asian Test Symposium*, 2004, pp. 82-87.

[167] 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.

[168] P. J. Rousseeuw, "Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis", *Journal of Computational and Applied Mathematics* 20 (1987), pp. 53-65.

[169] Samsung Electronics Corporation (System LSI application processor principal engineer), *personal communication*, July 2014.

[170] J. Schulze and R. Tally, "Mitigating Voltage Droop During Scan with Variable Shift Frequency", *Proc. International Test Conference*, 2014, pp. 1-8.

[171] S. Seo, Y. Lee, J. Lee and S. Kang, "A Scan Shifting Method based on Clock Gating of Multiple Groups for Low Power Scan Testing", *Proc. International Symposium on Quality Electronic Design*, 2015, pp. 162-166.

[172] W. Shi and 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] M. Shih and E. S. Kuh, "Quadratic Boolean Programming for Performance-Driven System Partitioning", *Proc. ACM/IEEE Design Automation Conference*, 1993, pp. 761-765.

[174] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan and D. Tarjan, "Temperature-Aware Microarchitecture", *Proc. International Symposium on Computer Architecture*, 2003, pp. 2-13.

[175] M. Steyaert, N. Butzen, H. Meyvaert, A. Sarafianos, P. Callemeyn, T. Van Breussegem and M. Wens, "DCDC performance Survey".
http://homes.esat.kuleuven.be/ steyaert/DCDC_Survey/DCDC_PS.html

[176] H. Su and S. S. Sapatnekar, "Hybrid Structured Clock Network Construction", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2001, pp. 333-336.

[177] V. Subramanian and A. Somani, "Conjoined Pipeline: Enhancing Hardware Reliability and Performance through Organized Pipeline Redundancy", *Proc. IEEE Pacific Rim International Symposium on Dependable Computing*, 2008, pp. 9-16.

[178] S. Sunder and K. Scholtman, "Multi-Mode Multi-Corner Clocktree Synthesis", *U.S. Patent* No. US20090217225A1, August 2009.

[179] T. G. Szymanski, "Computing Optimal Clock Schedules", *Proc. ACM/IEEE Design Automation Conference*, 1992, pp. 399-404.

[180] C. M. Tan and F. He, "3D Circuit Model for 3DIC Reliability Study", *Proc. International Conference on Thermal, Mechanical and Multiphysics Simulation and Experiments in Micro-Electronics and Micro-Systems*, 2009, pp. 1-7.

[181] Andres R. Teene, "Clock Skew Insensitive Scan Chain Reordering", *U.S. Patent* No. US6539509B1, May 2003.

[182] 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.

[183] C. C. Tsai, Y. Shi, G. Luo and I. H.-R. Jiang, "FF-bond: Multi-Bit Flip-Flop Bonding at Placement", *Proc. ACM International Symposium on Physical Design*, 2013, pp. 147-153.

[184] 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.

[185] K. N. Tu, "Reliability Challenges in 3D IC Packaging Technology", *Microelectronics Reliability* 51(3) (2011), pp. 517-523.

[186] J. T. Tudu, E. Larsson, V. Singh and H. Fujiwara, "Graph Theoretic Approach for Scan Cell Reordering to Minimize Peak Shift Power", *Proc. Great Lakes Symposium on Very Large Scale Integration*, 2010, pp. 73-78.

[187] K. Ueda, F. Morishita, S. Okura, L. Okamura, T. Yoshihara and K. Arimoto, "Low-Power On-Chip Charge-Recycling DC-DC Conversion Circuit and System", *IEEE Journal of Solid State Circuits* 48(11) (2013), pp. 2608-2617.

[188] H. J. M. Veendrick, "Short-Circuit Dissipation of Static CMOS Circuitry and Its Impact on the Design of Buffer Circuits" *IEEE Journal of Solid State Circuits* 19(4) (1984), pp. 468-473.

[189] 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.

[190] L. Wan and D. Chen, "DynaTune: Circuit-Level Optimization for Timing Speculation Considering Dynamic Path Behavior", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2009, pp. 172-179.

[191] K. Wang, L. Duan and X. Cheng, "ExtensiveSlackBalance: An Approach to Make Front-end Tools Aware of Clock Skew Scheduling", *Proc. ACM/IEEE Design Automation Conference*, 2006, pp. 951-954.

[192] K. Wang, H. Fang, H. Xu and X. Cheng, "A Fast Incremental Clock Skew Scheduling Algorithm for Slack Optimization", *Proc. Asia and South Pacific Design Automation Conference*, 2008, pp. 492-497.

[193] S. H. Wang, Y. Y. Liang, T. Y. Kuo and W. K. Mak, "Power-Driven Flip-Flop Merging and Relocation", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 31(2) (2012), pp. 180-191.

[194] K. Wang and M. Marek-Sadowska, "Potential Slack Budgeting with Clock Skew Optimization", *Proc. IEEE International Conference on Computer Design*, 2004, pp. 265-271.

[195] X. Wei, Y. Cai and X. Hong, "Effective Acceleration of Iterative Slack Distribution Process", *Proc. IEEE International Symposium on Circuits and Systems*, 2007, pp. 1077-1080.

[196] M. H. Woods, "MOS VLSI Reliability and Yield Trends", *Proc. of the IEEE* 74(12) (1986), pp. 1715-1729.

[197] H. Wu, I.-M. Liu, M. D. F. Wong and Y. Wang, "Post-Placement Voltage Island Generation under Performance Requirement", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2005, pp. 309-316.

[198] K-.C. Wu and D. Marculescu, "Clock Skew Scheduling for Soft-Error-Tolerant Sequential Circuits", *Proc. Design, Automation and Test in Europe*, 2010, pp. 717-722.

[199] H. Wu and M. D. F. Wong, "Improving Voltage Assignment by Outlier Detection and Incremental Placement", *Proc. ACM/IEEE Design Automation Conference*, 2007, pp. 459-464.

[200] H. Wu, M. D. F. Wong and I.-M. Liu, "Timing-Constrained and Voltage-Island-Aware Voltage Assignment", *Proc. ACM/IEEE Design Automation Conference*, 2006, pp. 429-432.

[201] J. G. Xi and W. W.-M. Dai, "Jitter-Tolerant Clock Routing in Two-phase Synchronous Systems", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 1996, pp. 316-320.

[202] J. Xie, V. Narayanan and Y. Xie, "Mitigating Electromigration of Power Supply Networks Using Bidirectional Current Stress", *Proc. Great Lakes Symposium on Very Large Scale Integration*, 2012, pp. 299-302.

[203] C. Xu, P. Li, G. Luo, Y. Shi and I. H.-R. Jiang, "Analytical Clustering Score with Application to Post-Placement Multi-Bit Flip-Flop Merging", *Proc. ACM International Symposium on Physical Design*, 2015, pp. 93-100.

[204] J. T. Yan and Z. W. Chen, "Construction of Constrained Multi-Bit Flip-Flops for Clock Power Reduction", *Proc. International Conference on Green Circuits and Systems*, 2010, pp. 675-678.

[205] Y.-M. Yang, I. H.-R. Jiang and S.-T. Ho, "PushPull: Short Path Padding for Timing Error Resilient Circuits", *Proc. ACM International Symposium on Physical Design*, 2013, pp. 50-57.

[206] H. H. Yang and D. F. Wong, "Efficient Network Flow Based Min-Cut Balanced Partitioning", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 15(12) (1996), pp. 1533-1540.

[207] R. Ye, F. Yuan and Q. Xu, "Online Clock Skew Tuning for Timing Speculation", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2011, pp. 442-447.

[208] R. Ye, F. Yuan, H. Zhou and Q. Xu, "Clock Skew Scheduling for Timing Speculation", *Proc. Design, Automation and Test in Europe*, 2012, pp. 929-934.

[209] N. E. Young, R. E. Tarjan and J. B. Orlin, "Faster Parametric Shortest Path and Minimum Balance Algorithms", *Networks* 21(2) (1991), pp. 205-221.

[210] F. Yuan and Q. Xu, "InTimeFix: A Low-Cost and Scalable Technique for In-Situ Timing Error Masking in Logic Circuits", *Proc. ACM/EDAC/IEEE Design Automation Conference*, 2013, pp. 183:1-183:6.

[211] G. Zhang and P. A. Beerel, "Stochastic Analysis of Bubble Razor", *Proc. Design, Automation and Test in Europe*, 2014, pp. 109:1-109:6.

[212] W. Zhao, M. Tehranipoor and S. Chakravarty, "Power-Safe Test Application Using An Effective Gating Approach Considering Current Limits", *Proc. IEEE VLSI Test Symposium*, 2011, pp. 160-165.

[213] V. Zolotov, C. Visweswariah and J. Xiong, "Voltage Binning Under Process Variation", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2009, pp. 425-432.

[214] Bookshelf: Rectilinear Steiner Minimum Tree Slot.
http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/RSMT/

[215] Cadence Innovus Implementation System User Guide.

[216] Cadence NC-Verilog User Guide.

[217] Cadence SOC Encounter User Guide.

[218] CAD/CAM/CAE Wallchart.
http://www.garysmitheda.com/wp-content/uploads/2015/05/All_WC-15.pdf

[219] CRAFT Program Aims for Affordable Designer Circuits that Do More with Less Power.
http://www.darpa.mil/news-events/2015-08-17

[220] Guidelines to Understanding Reliability Prediction. http://www.epsma.org

[221] Hotspot User Guide. http://lava.cs.virginia.edu/HotSpot/index.htm

[222] Hungarian Algorithm. http://www.informatik.uni-freiburg.de/stachnis/index.html

[223] IBM ILOG CPLEX. www.ilog.com/products/cplex/

[224] ITRS 2011 Design Chapter. http://www.itrs2.net/2011-itrs.html

[225] ITRS 2013 System Drivers Chapter. http://www.itrs2.net/2013-itrs.html

[226] LEMON (Library for Efficient Modeling and Optimization in Networks). http://lemon.cs.elte.hu/trac/lemon

[227] LP_Solve Manuals. http://lpsolve.sourceforge.net/5.5/

[228] MATLAB. http://www.mathworks.com/products/matlab/

[229] MLPart. http://vlsicad.ucsd.edu/GSRC/bookshelf/Slots/Partitioning/MLPart

[230] OpenCores: Open Source IP-Cores. http://www.opencores.org

[231] Sun OpenSPARC Project. http://www.sun.com/processors/opensparc/

[232] RedHawk User Guide. https://www.apache-da.com/products/redhawk

[233] ScanOpt. http://vlsicad.ucsd.edu/GSRC/Bookshelf/Slots/ScanOpt/

[234] SDPA Official Page. http://sdpa.sourceforge.net/

[235] Sensitivity-Based Leakage Optimizer. http://vlsicad.ucsd.edu/SIZING/

[236] Synopsys SiliconSmart User Guide.

[237] Synopsys Design Compiler User Guide.

[238] Synopsys DFT Compiler User Guide.

[239] Synopsys IC Compiler User Guide.

[240] Synopsys PrimeTime User Guide.