UNIVERSITY OF CALIFORNIA, SAN DIEGO

New Applications of Learning-Based Modeling in Nanoscale Integrated-Circuit Design

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

in

Computer Science (Computer Engineering)

by

Siddhartha Nath

Committee in charge:

Professor Andrew B. Kahng, Chair
Professor Chung-Kuan Cheng
Professor Bill Lin
Professor Bhaskar Rao
Professor Lawrence Saul

2016

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

_____

_____

_____

_____

_____

Chair

University of California, San Diego

2016

DEDICATION

I dedicate this thesis to my parents, Dr. Swaraj K. Nath and Professor Gargi Nath. Without their endless support, love, encouragement and sacrifice this thesis would not have been possible.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACKNOWLEDGMENTS

Foremost, I would like to thank my parents (my father Dr. Swaraj K. Nath and my mother Professor Gargi Nath), my wife Dr. Mrittika Bhattacharya Nath, her parents (her father Gurudas Bhattacharya and her mother Debika Bhattacharya) and my sister Sanghamitra Nath for their endless love throughout the years. Their patience and support have been the most important contributions to my course.

I would like to thank my advisor Professor Andrew B. Kahng for his constant and invaluable advice and support. His attitude to achieve perfection and principles on research have provided an indelible lesson throughout my Ph.D. study. His immense knowledge and guidance helped me in conducting research and in the writing of this thesis.

Besides my advisor, I would like to thank my thesis committee members, Professor Chung-Kuan Cheng, Professor Bill Lin, Professor Bhaskar Rao, and Professor Lawrence Saul, for their time to review my research and for their valuable comments. Especially, I thank Professor Bill Lin for his advice and encouragement at the beginning of my Ph.D. studies.

I thank Dr. Kambiz Samadi for mentoring me on many of my research projects.

I am indebted to my music teachers, Pandit Nagaraj Rao Havaldar and Smt. Bombay Jayashri Ramnath for enabling me to balance my Ph.D. work with music. I am extremely thankful to my group of music lovers, Mr. Abhay Ramachandra, Mr. Amol Patwardhan, Ms. Kanika Khanna, Mr. Adithya Suresh and Ms. Revathi Subra.

Last, but not least, I would like to thank my fellow labmates in the UCSD VLSI CAD Laboratory (Vaishnav Srinivas, Wei-Ting Chan, Jiajia Li, Hyein Lee, Kwangsoo Han, Lutong Wang, Bangqi Xu) and former lab members (Dr. Kwangok Jeong, Professor Seokhyeong Kang, Dr. Jingwei Lu, Dr. Tuck-Boon Chan and Ilgweon Kang) for the stimulating discussions, the sleepless nights, and the many great times together. Immense thanks to Mr. Adithya Suresh, Mr. Rahul Hazra and Ms. Sunandha Srikanth for proofreading this thesis.

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

Chapter 3 is in part a reprint of W.-T. J. Chan, Y. Du, A. B. Kahng, S. Nath and K. Samadi, "BEOL Stack-Aware Routability Prediction from Placement Using Data Mining Techniques", *Proc. IEEE International Conference on Computer Design*, 2016, to appear; W.-T. J. Chan, K. Y. Chung, A. B. Kahng, N. MacDonald and S. Nath, "Learning-Based Prediction of Embedded Memory Timing Failures during Initial Floorplan Design", *Proc. Asia and South Pacific Design Automation Conference*, 2016; W.-T. J. Chan, Y. Du, A. B. Kahng, S. Nath and K. Samadi, "3D-IC Benefit Estimation and Implementation Guidance from 2D-IC Implemen-

tation", *Proc. IEEE/ACM/EDAC Design Automation Conference*, 2015; A. B. Kahng, B. Lin and S. Nath, "ORION3.0: A Comprehensive NoC Router Estimation Tool", *IEEE Embedded Systems Letters* 7(2) (2015); A. B. Kahng, B. Lin and S. Nath, "High-Dimensional Metamodeling for Prediction of Clock Tree Synthesis Outcomes", *Proc. ACM International Workshop on System-Level Interconnect Prediction*, 2013; and A. B. Kahng, B. Lin and S. Nath, "Explicit Modeling of Control and Data for Improved NoC Router Estimation", *Proc. IEEE/ACM/EDAC Design Automation Conference*, 2012.

Chapter 4 is in part a reprint of A. B. Kahng, M. Luo and S. Nath, "SI for Free: Machine Learning of Interconnect Coupling Delay and Transition Effects", *Proc. ACM International Workshop on System-Level Interconnect Prediction*, 2015; 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; K. Han, A. B. Kahng, J. Lee, J. Li and S. Nath, "A Global-Local Optimization Framework for Simultaneous Multi-Mode Multi-Corner Skew Variation Reduction", *Proc. IEEE/ACM/EDAC Design Automation Conference*, 2015; and 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.

Chapter 5 is in part a reprint of A. B. Kahng and S. Nath, "Optimal Reliability-Constrained Overdrive Frequency Selection in Multicore Systems", *Proc. International Symposium on Quality Electronic Design*, 2014; and the submitted drafts: W.-T. J. Chan, Y. Du, A. B. Kahng, S. Nath, K. Samadi and H. Yao, "Toward "True 3D": Assessment of a New Objective in Analytic 3DIC Placement", *Proc. Asia and South Pacific Design Automation Conference*, submitted draft; and P. Agrawal, M. Broxterman, B. Chatterjee, P. Cuevas, K. H. Hayashi, A. B. Kahng, P. K. Myana and S. Nath, "Optimal Scheduling and Allocation for IC Design Management and Cost Reduction", *ACM Transactions on Design Automation of Electronic Systems*, submitted draft.

My co-authors (Prabhav Agrawal, Mike Broxterman, Wei-Ting Jonas Chan, Biswadeep Chatterjee, Dr. Kun Young Chung, Patrick Cuevas, Dr. Yang Du, Kwangsoo Han, Professor Seung-Soo Han, Kathy H. Hayashi, Professor Andrew B. Kahng, Jongpil Lee, Jiajia Li, Professor Bill Lin, Mulong Luo, Nancy MacDonald, Pranay K. Myana, Dr. Kambiz Samadi, Ashok Vydyanathan, and Professor Hailong Yao, listed in alphabetical order) have all kindly approved the inclusion of the aforementioned publications in my thesis.

VITA

1980	Born, Kolkata, India
2003	B.E.(Hons), Electrical and Electronics Engineering, Birla Institute of Technology and Science, Pilani, India
2003	Software Engineer, Intel Technology India Private Limited, Bangalore, India
2013	M.Sc., Computer Science (Computer Engineering), University of California, San Diego
2014	C.Phil., Computer Science (Computer Engineering), University of California, San Diego
2016	Ph.D., Computer Science (Computer Engineering), 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, Yang Du, Andrew B. Kahng, **Siddhartha Nath**, Kambiz Samadi and Hailong Yao, "Toward "True 3D": Assessment of a New Objective in Analytic 3DIC Placement", *Proc. Asia and South Pacific Design Automation Conference*, 2017, submitted draft.

- Prabhav Agrawal, Mike Broxterman, Biswadeep Chatterjee, Patrick Cuevas, Kathy H. Hayashi, Andrew B. Kahng, Pranay K. Myana and **Siddhartha Nath**, "Optimal Scheduling and Allocation for IC Design Management and Cost Reduction", *ACM Transactions on Design Automation of Electronic Systems*, 2016, submitted draft.

- Wei-Ting J. Chan, Yang Du, Andrew B. Kahng, **Siddhartha Nath** and Kambiz Samadi, "BEOL Stack-Aware Routability Prediction from Placement Using Data Mining Techniques", *Proc. IEEE International Conference on Computer Design*, 2016, to appear.

- Wei-Ting J. Chan, Kun Young Chung, Andrew B. Kahng, Nancy MacDonald and **Siddhartha Nath**, "Learning-Based Prediction of Embedded Memory Timing Failures during Initial Floorplan Design", *Proc. Asia and South Pacific Design Automation Conference*, 2016, pp. 178-185.

- Andrew B. Kahng, Mulong Luo, Gi-Joon Nam, **Siddhartha Nath**, David Z. Pan and Gabriel Robins, "Toward Metrics of Design Automation Research Impact",
Proc. IEEE/ACM International Conference on Computer-Aided Design, 2015, pp. 263-270.

- Andrew B. Kahng, Mulong Luo and **Siddhartha Nath**, "SI for Free: Machine Learning of Interconnect Coupling Delay and Transition Effects", *Proc. ACM International Workshop on System-Level Interconnect Prediction*, 2015, pp. 1-8.
- 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. IEEE/ACM/EDAC Design Automation Conference*, 2015, pp. 1-6.
- Wei-Ting J. Chan, Yang Du, Andrew B. Kahng, **Siddhartha Nath** and Kambiz Samadi, "3D-IC Benefit Estimation and Implementation Guidance from 2D-IC Implementation", *Proc. IEEE/ACM/EDAC Design Automation Conference*, 2015, pp. 1-6.
- Andrew B. Kahng, Bill Lin and **Siddhartha Nath**, "ORION3.0: A Comprehensive NoC Router Estimation Tool", *IEEE Embedded Systems Letters* 7(2) (2015), pp. 41-45.
- Tuck-Boon Chan, Andrew B. Kahng, Jiajia Li, **Siddhartha Nath** and Bong-Il 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.
- Andrew B. Kahng and **Siddhartha Nath**, "Optimal Reliability-Constrained Overdrive Frequency Selection in Multicore Systems", *Proc. International Symposium on Quality Electronic Design*, 2014, pp. 300-308.
- Seung-Soo Han, Andrew B. Kahng, **Siddhartha Nath** and Ashok Vydyanathan, "A Deep Learning Methodology to Proliferate Golden Signoff Timing", *Proc. Design, Automation and Test in Europe*, 2014, pp. 1-6.
- Tuck-Boon Chan, Kwangsoo Han, Andrew B. Kahng, Jae-Gon Lee and **Siddhartha Nath**, "OCV-Aware Top-Level Clock Tree Optimization", *Proc. Great Lakes Symposium on Very Large Scale Integration*, 2014, pp. 33-38.
- Wei-Ting J. Chan, Andrew B. Kahng and **Siddhartha Nath**, "Methodology for Electromigration Signoff in the Presence of Adaptive Voltage Scaling", *Proc. ACM International Workshop on System-Level Interconnect Prediction*, 2014, pp. 1-7.
- Juan-Antonio Carballo, Wei-Ting J. Chan, Paolo A. Gargini, Andrew B. Kahng and **Siddhartha Nath**, "ITRS 2.0: Toward a Re-Framing of the Semiconductor Technology Roadmap", *Proc. IEEE International Conference on Computer Design*, 2014, pp. 139-146.

- Wei-Ting J. Chan, Andrew B. Kahng, **Siddhartha Nath** and Ichiro 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.
- 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.
- Andrew B. Kahng, **Siddhartha Nath** and Tajana S. Rosing, "On Potential Design Impacts of Electromigration Awareness", *Proc. Asia and South Pacific Design Automation Conference*, 2013, pp. 527-532.
- Andrew B. Kahng, Bill Lin and **Siddhartha Nath**, "Enhanced Metamodeling Techniques for High-Dimensional IC Design Estimation Problems", *Proc. Design, Automation and Test in Europe*, 2013, pp. 1861-1866.
- Andrew B. Kahng, Seokhyeong Kang, Hyein Lee, **Siddhartha Nath** and Jyoti Wadhwani, "Learning-Based Approximation of Interconnect Delay and Slew in Signoff Timing Tools", *Proc. ACM International Workshop on System-Level Interconnect Prediction*, 2013, pp. 1-8.
- Andrew B. Kahng, Bill Lin and **Siddhartha Nath**, "High-Dimensional Metamodeling for Prediction of Clock Tree Synthesis Outcomes", *Proc. ACM International Workshop on System-Level Interconnect Prediction*, 2013, pp. 1-7.
- Andrew B. Kahng, Bill Lin and **Siddhartha Nath**, "Explicit Modeling of Control and Data for Improved NoC Router Estimation", *Proc. IEEE/ACM/EDAC Design Automation Conference*, 2012, pp. 392-397.

ABSTRACT OF THE DISSERTATION

New Applications of Learning-Based Modeling in Nanoscale Integrated-Circuit Design

by

Siddhartha Nath

Doctor of Philosophy in Computer Science (Computer Engineering)

University of California, San Diego, 2016

Professor Andrew B. Kahng, Chair

In today's leading-edge semiconductor technologies, it is increasingly difficult for IC designers to achieve sufficient improvements of performance, power and area metrics in their next-generation products. One root cause of this difficulty is the increased margins that are used in the design process to guardband for (i) variability and aging, as well as (ii) analysis inaccuracies. Currently, these margins incur huge costs to design companies, because the benefits of deploying the next technology node are only approximately 20% in circuit performance, power and density. To reduce margins, fast and accurate pathfinding of architecture, technology and constraints choices are essential. A second root cause is the high cost (and, therefore, limited supply) of electronic design automation tool licenses, accompanied by the lack of any systematic methodology to optimize the use of available tools within long-duration, highly iterative design processes. This constrains designers to perform only limited design-space exploration,

so as to keep within limits on design infrastructure cost and design turnaround time. This thesis presents new techniques to reduce guardbands in optimization loops in the IC design process by using fast and accurate learning-based models. These techniques can be grouped into three main thrusts: (i) design productivity gains through improved design- and implementation-space exploration; (ii) improved accuracy of electrical modeling and enablement of basic physical design optimizations; and (iii) optimizations of design power, energy, project management, and cost.

The thrust on *design productivity gains through improved design- and implementation-space exploration* presents four applications of learning-based models for accurate prediction of area, power, timing and routability. To enable area and power estimation of Networks-on-Chip routers, such that architecture-level (RTL-level) design-space exploration can be efficiently performed, this thesis presents an open-source tool, *ORION3.0*.

The thrust on *improved accuracy of electrical modeling and enablement of basic physical design optimizations* presents new methodologies to perform high-dimensional learning-based modeling of delay, transition time and slack in timing paths. A methodology to develop accurate models of post-routing optimization of signal delays at multiple signoff corners, so as to enable a new optimization of clock skew variation across corners is also described.

The thrust on *optimizations of design power, energy, project management, and cost* presents three distinct works that directly benefit leading-edge SoC design companies. The first work describes a new analytic three-dimensional placement tool using a new objective function that achieves significant wirelength and power reduction relative to two-dimensional implementations. The second work provides two mixed integer-linear programs for optimal multi-project, multi-resource allocation with task precedence and resource co-constraints for IC design management and cost reduction. The third work presents a maximum-value, reliability-constrained overdrive frequencies problem that guarantees prescribed lower bounds on acceptable performance and acceptable throughput in multicore systems, without exceeding prescribed lifetime budget for any core.

Chapter 1

Introduction

The *International Technology Roadmap for Semiconductors* (ITRS) [304] has been a very successful and influential industry-wide roadmapping effort since 1997. In the ITRS, the *microprocessor* (MPU) and *system-on-chip* (SoC) product classes are two *system drivers* that drive requirements for design technologies and processes. Density (i.e., layout area per DRAM bit, SRAM bitcell, or logic gate) is a key metric of technology scaling. Lithography improvements, which reduce the *metal pitch* (i.e., a wire width plus a wire spacing) and poly pitch, have been a main driver of density scaling. In a two-dimensional layout, the pitch scales by $0.7\times$ in both horizontal and vertical dimensions at a new technology node, which scales the area by $0.49\times$. Thus, there is an "available" doubling of transistor density inherent in the classical understanding of "Moore's Law" scaling. Historically, the *geometric scaling* of $0.7\times$ in each successive technology node [304] has enabled doubling of transistor count in a constant die area.

However, Figure 1.1 shows that the "realized" density scaling in products has slowed down to $\sim 1.6\times$ per node (in contrast to the "available" $2\times$ per node density scaling) since 2007 [121] [126]. Kahng [126] refers to this difference as the "design capability gap". This slowdown in density scaling is also well-correlated with data from [334], and indicates that in advanced technology nodes, a significant gap exists between the "realized" and "available" benefits of technology scaling. Overall, there is a slowdown of *power-performance-area-cost* (PPAC) scaling as realized from underlying process and device scaling.

Figure 1.2, adapted from [118], illustrates a traditional IC design flow. The cuboids in orange color show the various stages in the flow. At the architecture design stage, decisions on power requirements, technology, number of IP blocks, communication protocol are made. Once the architecture is determined, functionalities of and connectivities between various blocks (e.g.,

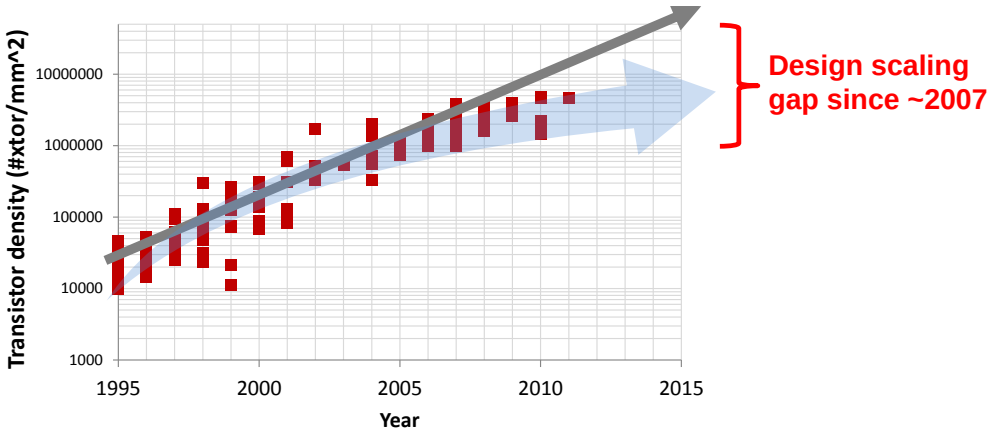


Figure 1.1: Gap between "available" density scaling (grey arrow) and "actual density" scaling (red squares) [126].

CPU, GPU and DSP cores) of the design are defined. In the high-level synthesis stage, the design is described using a hardware description language, e.g., Verilog, at the register-transfer level (RTL). In the logic synthesis stage, electronic design automation (EDA) tools are deployed to convert the HDL to gate-level circuit elements and output a gate-level netlist. The design flow then transitions to the physical design stages. In the floorplanning stage, locations of macro blocks (e.g., embedded memories) and primary inputs and outputs (PI/POs) are determined. In the placement stage, standard cells in the gate-level netlist are assigned spatial locations. In the clock tree synthesis (CTS) stage, the clock signal is routed to meet prescribed latency and skew requirements. In the routing stage, wiring connections are established between logically connected pins in the netlist. In this stage, wires are routed on specific tracks of metal layers, and various design rules are met, so that the design can be fabricated. Optimization is performed in each of placement, CTS and routing stages to minimize wirelength, power, resources (e.g., chip area occupied by buffers) and latency.

The slowdown in density scaling in advanced technology nodes (as discussed above) is largely attributed to power, performance and area resources being spent to guardband variability, reliability, complex design rules, and other design considerations that become prominent at the "nanometer scale". Guardbands are applied at all stages of the IC design flow, which is why designers obtain only a part of Moore's Law [349] scaling benefits. Figure 1.3 [126] illustrates this lost benefit in design quality due to signoff with increased guardbands. The brown line shows how design quality (e.g., frequency) scales with nominal or "available" process and device technology scaling. Due to application of guardbands, design quality reduces substantially in advanced technology nodes.

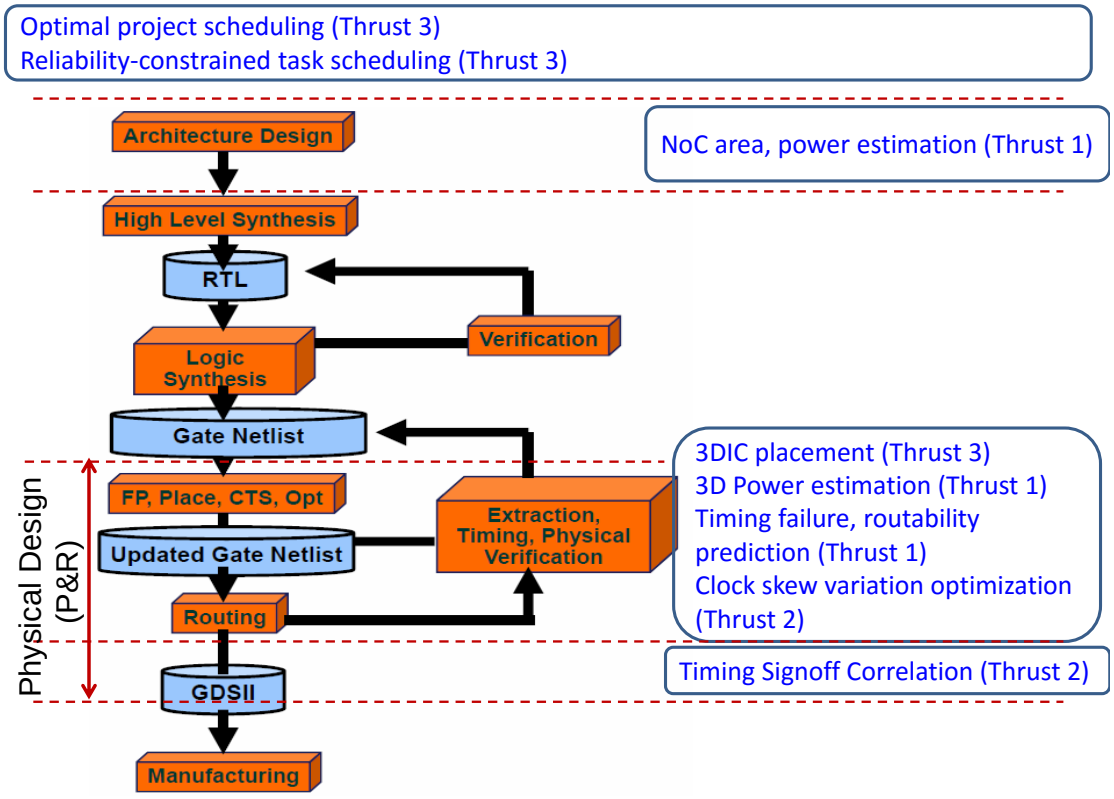


Figure 1.2: Traditional IC design flow [118] and the three thrusts of this thesis.

The work reported in this thesis develops a wide variety of fast, accurate machine learning-based models that seek to reduce these large design guardbands by enabling (i) early-stage *pathfinding*, and (ii) incremental physical design optimizations. Early-stage pathfinding is difficult because it involves long EDA tool runtimes and is a highly iterative process. The modeling works in this thesis improve design productivity by enabling identification of design- and implementation space parameters that help deliver the PPAC scaling required for a healthy semiconductor industry. Improved modeling leads to better incremental physical design optimizations. The new techniques in this thesis can be grouped into three main *thrusts*.

- Design productivity gains through improved design- and implementation-space exploration.
- Improved accuracy of electrical modeling and enablement of basic physical design optimizations.
- Optimizations of design power, energy, project management, and cost.

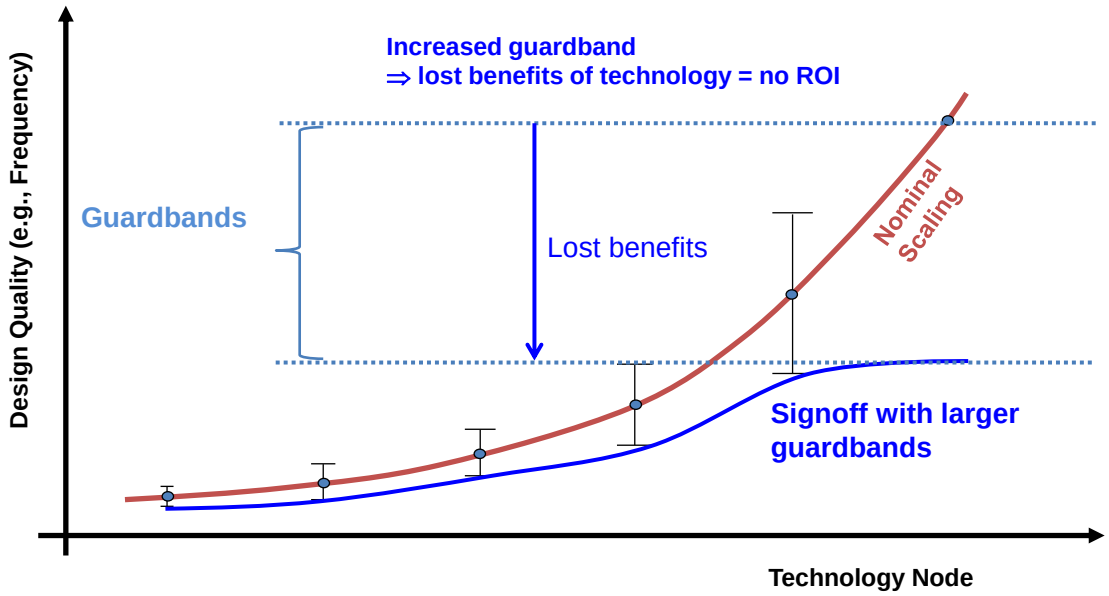


Figure 1.3: Illustration of lost benefit in design quality due to increased signoff guardbands [126].

The thrust on *design productivity gains through improved design- and implementation-space exploration* presents four applications of learning-based models for accurate prediction of area, power, timing and routability. To enable area and power estimation of network-on-chip (NoC) routers, such that architecture-level (RTL-level) design-space exploration can be efficiently performed, this thesis presents an open-source tool, *ORION3.0*.

The thrust on *improved accuracy of electrical modeling and enablement of basic physical design optimizations* presents new methodologies to perform high-dimensional learning-based modeling of delay, transition time and slack in timing paths. A methodology to develop accurate models of post-routing optimization of signal delays at multiple signoff corners, so as to enable a new optimization of clock skew variation across corners, is also described.

The thrust on *optimizations of design power, energy, project management, and cost* presents three distinct works that directly benefit leading-edge SoC design companies. The first work describes a new analytic three-dimensional placement tool using a new objective function that achieves significant wirelength and power reduction relative to two-dimensional implementations. The second work provides two mixed integer-linear programming formulations for optimal multi-project, multi-resource allocation with task precedence and resource co-constraints, with applications to IC design management and cost reduction. The third work presents a maximum-value, reliability-constrained overdrive frequencies optimization that guar-

antees prescribed lower bounds on acceptable performance and acceptable throughput in multi-core systems, without exceeding the prescribed lifetime budget of any core. Even though Thrust 3 is not related to learning-based modeling, optimization changes the envelope of what is being modeled or predicted. Thus, better optimization leads to better flows and ground truth, which leads to more accurate and realistic modeling; the first work exemplifies this. Furthermore, better modeling enables more accurate prediction of constraints or requirements, which leads to better optimization; the last two works exemplify this.

Figure 1.2 shows areas in the IC design flow where the three thrusts of this thesis apply. Categorization of individual works into IC design flow stages is according to the following reasoning.

- The two works listed above the "architecture design" box in the flow encompass the entire IC design flow. They optimize design cost by optimal scheduling of design projects and resources, and by optimal scheduling of tasks to guarantee acceptable performance and balance core wearout over the lifetime of a multi-core system. These works enable reduction of guardbands in resource allocation through optimal scheduling formulations and solvers.

- The NoC work enables efficient architecture-level (RTL-level) design-space exploration of parameters with comprehension of area and power metrics from the routing stage in physical design. This work reduces design guardband by guiding the appropriate choice of RTL-level parameters so as to meet area and power requirements after physical design optimizations have been performed.

- The work on three-dimensional IC (3DIC) placement develops a new analytic placement tool to achieve wirelength and power benefits over 2DIC implementations. Since this work is related to placement in the physical design flow, it appears with the P&R flow in the figure.

- The work on 3D power estimation enables model-guided implementation-space exploration that affects 3DIC floorplanning, placement and optimizations. Since this work enables selection of parameters that affect optimization, we place it after 3DIC placement.

- The works on timing failure and routability prediction appear at the same level as 3D power estimation since these works affect downstream optimizations.

- The work on clock skew variation minimization appears as the same level as the works

on placement, since it is applicable to clock trees that are typically co-optimized with placement.

- The works related to timing signoff appear at the end of the P&R flow since they apply to performance analysis and verification that occurs after the design has been routed.

The remainder of this thesis is organized as follows.

- Chapter 2 reviews related works pertaining to the thrusts of this thesis. We broadly categorize and review works related to machine learning in IC design and optimizations in IC design. We further taxonomize the first category into predictions of (i) design-space exploration of NoC routers, (ii) prediction of design metrics and quality of results (QoR), and (iii) timing analysis and correlation. In (ii), we describe prior works that predict clock network metrics, IC defects at the mask and post-silicon stages, routability and 3DIC metrics. In the second category, we review works related to 3DIC flows and placers, project scheduling under resource constraints and reliability-constrained task scheduling in multi-core systems.

- Chapter 3 presents four new applications of machine learning-based models for efficient design- and implementation-space exploration. The first work describes the development of a comprehensive suite of parametric and non-parametric area and power models for NoC routers, *ORION3.0*. We have released *ORION3.0* on the web. Over 760 downloads have been made from industry and academia since availability commenced in February 2013. The second work proposes a modeling methodology to perform early prediction of timing failures of designs with embedded memories. Our methodology uses only the netlist, timing constraints and floorplan context (wherein the embedded memories are placed). The third work describes a learning-based methodology to predict whether a back-end-of-line stack-aware placement solution is routable. Our modeling does not use any information from trial or early global routing. We also predict Pareto frontiers of utilization, number of metal layers and aspect ratio based on very few placements. The fourth work describes a methodology to estimate 3DIC power benefit based on corresponding golden 2DIC implementation parameters. We describe a novel "stress testing" method and use cases for model-guided implementations.

- Chapter 4 first presents two new approaches to the signoff timer correlation problem using "big-data" mindsets. In the first approach, we describe a machine learning-based tool,

GTX, that is developed using a deep learning methodology. GTX corrects divergence between timers in flip-flop setup time, cell arc delay, wire delay, stage delay and path slack at timing endpoints between timers. We demonstrate accuracy of GTX using multiple technology libraries, real designs from OpenCores [318] and multiple types of correlations between leading-edge commercial signoff timing and design implementation tools. In our second approach to the timer correlation problem, we extend GTX to develop predictors of timing in signal integrity (SI) mode based on timing reports from non-SI mode. We propose new parameters that span electrical and logic structures and report accurate predictions of incremental arc delay and slew. In the third work in this chapter, we describe a methodology to develop accurate models of post-routing optimization of signal delays at multiple signoff corners, so as to enable a new optimization of clock skew variation across corners. We demonstrate that our models are significantly more accurate than analytical models of signal delay and slew.

- Chapter 5 presents three distinct works related to design cost and QoR optimization. The first work is a new analytic 3DIC placement tool, *APlace3D* that applies a new "true 3D" wirelength objective function. In conjunction with a commercial EDA tool, APlace3D achieves signification wirelength and power reduction compared to 2DIC implementations of real designs. We demonstrate that our results generalize over a wide range of designs and multiple technology libraries. The second work provides two mixed integer-linear programs, along with associated solver implementation, for optimal multi-project, multi-resource allocation with task precedence and resource co-constraints. Our solver enables decision support to management in IC design companies via "what-if" analyses of cost and schedule tradeoffs. We describe real-world use cases wherein our solver achieves significantly better solutions (resource allocations) than the ones adopted by the design companies. The third work presents a maximum value, reliability-constrained overdrive frequencies (MVRCOF) problem for task scheduling in multi-core systems. Our solver guarantees prescribed lower bounds on acceptable performance and throughput, without exceeding prescribed lifetime budget for any core. We present both optimal and heuristic solutions that determine the execution times of each core in each combination of simultaneously active cores, such that the cores wear out in a balanced manner over chip lifetime. Our solutions are usable by a scheduler in a multi-core system.

Chapter 2

Review of Related Works

In this chapter, we review related works in three areas: (i) machine learning-based regression and classification models for pathfinding in IC design or EDA, (ii) modeling to perform timing analysis and correlation, and (iii) optimizations related to design power, cost and reliability. These three areas correspond to the three thrusts of this thesis.

Application of machine learning in IC design is challenging because data is "limited" [254]. Lack of data may hide the complexity of the underlying behavior that the learning algorithms seek to model. Since IC design cycles are time-sensitive, data generation, modeling and model refinement must all be completed in a short span of time relative to the design cycle, which is at most ∼80–100 weeks for complex systems-on-chip (SoCs). Certain stages in IC design naturally benefit from large volumes of data being available for modeling; hence, machine learning-based modeling has seen applications to analysis of lithography mask patterns [210] [254] [269], prediction of lithography-induced variation [109] and post-silicon testing [90]. In other stages of IC design (e.g., architecture design, physical design, etc.), data must be carefully generated (or, extracted) using appropriate design of experiments that comprehend tool flows, technology libraries and implemented designs. In this chapter, we review some of these aspects of applying machine learning to IC design.

2.1 Modeling for Pathfinding in IC Design

We broadly categorize modeling for pathfinding in EDA into architecture-level and implementation-level efforts. We further categorize architecture-level modeling into (i) template-based models, (ii) parametric models, and (iii) non-parametric models. And, we categorize mod-

eling at the implementation level as modeling that occurs during (i) synthesis, (ii) floorplanning, power planning and placement, (iii) clock network synthesis, (iv) routing, and (v) IC testing. We also briefly highlight works on hardware implementation of machine learning algorithms.

2.1.1 Pathfinding at the Architecture-Level

In this section, we primarily review works related to NoC architecture, since these provide the context for our NoC area and power estimation work in Section 3.1. Previous works on NoC architecture-level modeling can be broadly categorized into (i) template-based (e.g., logic template models for each router component block – crossbar, switch and VC arbiter, and input and output buffers) models, (ii) parametric models, and (iii) non-parametric models.

Template-Based Models

Patel et al. [198] propose a transistor count-based analytical model for networks-on-chip (NoCs) power. Large errors can result because router microarchitectural parameters are not considered. ORION [253] and ORION2.0 [115] use microarchitecture and technology parameters for the router component blocks.

Parametric Models

Prior works in this category are based on pre-layout (RTL or post-synthesis gate-level) [189] [33] [80] [150] or post-layout [14] [15] [200] [173] simulations. Chan et al. [33] develop cycle-accurate power models with reported average errors up to 20%. Meloni et al. [173], and Lee and Bagherzadeh [150] perform parametric regression analysis on post-layout and RTL simulation results, respectively. Their models cannot be used to explain sensitivity of power dissipation in each router block to change in load, microarchitecture or implementation parameters. Ye et al. [265] and Penolazzi et al. [200] estimate power dissipation using a bit-level model, and Penolazzi et al. [200] propose a static bit-based model to estimate Nostrum NoC power.

In Section 3.1 of this thesis, we present new parametric models to estimate area and power of each component of NoCs. These models are developed using post-synthesis data and hence are very accurate as compared to template-based models. We also present new models for flit-level power.

Non-Parametric Models

Surrogate or non-parametric models or metamodels are gaining popularity for early exploration and characterization of the solution space for various aspects of IC design. Multivariate Adaptive Regression Splines (MARS) and decision trees are examples of tree-based models, whereas Kriging (KG) and Radial Basis Function (RBF) are examples of Gaussian-process models [85]. Support Vector Machines (SVMs) and Artificial Neural Networks (ANNs) are other popular non-parametric modeling techniques used in EDA. Ipek et al. [96] use ANNs to predict performance of core and memory of chip-multiprocessors. Lee et al. [148] use KG to estimate processor power in multiprocessor systems. Dubois et al. [64] use KG to estimate area of network-on-chip (NoC) routers. Jeong et al. [98] and Kahng et al. [116] use MARS to model area and power of NoCs. The authors show that parametric regression has large errors and hence propose the use of non-parametric regression. The authors use MARS with linear splines, and report $\sim$60% worst-case and $\sim$6% average errors for both area and power. SVM regression has been used to estimate NoC latency in [202].

For analog and mixed signal design automation, Li et al. [153] develop a statistical regression (*STAR*) technique using non-parametric regression to predict moment values, and use moment matching to reduce number of variation-related parameters in designing low noise amplifiers (LNAs). Compared to traditional methods of reducing dimensionality, *STAR* is 12$\times$ faster and uses 12$\times$ fewer samples to achieve the same modeling accuracy. Crombecq et al. [59] use adaptive sampling to study accuracy, as well as impacts of high-dimensionality in various surrogate models for LNA gain estimation. They quantify the root mean square error in each experiment, and propose to try all models and choose the best one. Goel et al. [76] propose weighted surrogate modeling instead of using the best estimation model. Liu et al. [160] use boosting with linear regressors to predict gain, power, slew rate, unity gain frequency and phase margin of LNAs. The main conclusion of the authors is that fitting each model requires substantial tuning in the number of stages in boosting.

In the field of computer architecture and compilers, Stephenson et al. [234] use genetic programming, a non-parametric technique, to learn compiler heuristics such as data prefetching, register allocation and hyperblock selection, and report up to 73% speedup in compiler optimizations. Hamerly et al. [82] propose to use machine learning (k-means clustering) to identify repetitive patterns in a program's execution so as to reduce the runtime of cycle-accurate architectural simulators. Lee et al. [149] propose to use linear regression and ANN to perform efficient design-space exploration of microarchitectural parameters; their models predict the per-

formance of various parallel applications with at most 10.5% error. Ozisikyilmaz et al. [188] also propose to use linear regression and ANN to perform efficient design-space exploration of microarchitectural parameters to achieve speedups on SPEC 2006 CPU benchmarks, and report that their models have 96.5% accuracy in identifying the best set of parameters for each SPEC 2006 benchmark. Liao et al. [156] use multiple parametric and non-parametric modeling techniques to perform data prefetching in datacenters. The authors of [156] demonstrate that their model-predicted prefetches are highly accurate, with worst-case error of 1%.

In Section 3.1 of this thesis, we explore multiple non-parametric modeling techniques to develop area and power models for NoCs using post-P&R data. We demonstrate that each technique achieves a different modeling accuracy depending on the size of the training set, and are significantly more accurate than template-based NoC models across multiple technologies and RTL generators.

2.1.2 Pathfinding at the Implementation-Level

Previous works on implementation-level modeling can be broadly categorized into modeling during (i) logic synthesis, (ii) floorplanning, power planning and placement, (iii) clock network synthesis, (iv) routing, and (v) IC testing. All of the works that we review use non-parametric models.

Modeling during Logic Synthesis

Rokach et al. [211] propose the use of decision trees along with a greedy approach to learn functions as part of their *Circuit-Decomposition-Engine*. Given a library of components and their sizes, the decision trees learn the function of disjunctive normal form (DNF) boolean representations and synthesize circuit structures from the libraries. Hutton and Karchmer [94] propose efficient design-space exploration of metrics such as operating speed, power and area during synthesis of FPGAs. The authors propose to create models using regression on existing synthesized designs. Li and Jabri [154] use ANNs to design standard cell libraries. The authors develop models to predict the number of vias and models to estimate cell layout shape functions using layout and timing constraints.

Modeling during Floorplanning, Power Planning and Placement

Chang et al. [45] propose a model-guided design flow to create power delivery networks (PDNs) that satisfy both IR-drop and electromigration constraints. The authors use a learning-

based model to predict the increase in wirelength for a given PDN configuration. The authors have explored multiple learning techniques such as Gaussian regression, Bayesian regression, stepwise regression, ridge linear regression and SVM, and report average errors between 2.8% to 6.1%. Cheng et al. [47] use MARS to model the worst-case performance under power supply noise variation. Liu [159] uses KG to predict IR drop of cells using spatial correlation of IR drop maps and on-chip temperature. The author reports worst-case error of 76%.

Yu [271] explores the application of Hopfield neural nets for standard cell placement. The author makes a negative recommendation because the complexity of neural nets can be asymptotically large O(n^4), network parameters are difficult to control, and solutions for a small $4\times$ mesh can have 20% worse half-perimeter wirelength (HPWL) as compared to exhaustive pairwise swap. Ward et al. [257] develop a high-performance placement tool, *PADE*, using SVM and ANN models. The authors create models for datapaths in netlists by analyzing netlist structures and then use SVM and ANN to classify a path as either datapath or non-datapath. The authors evaluate modeling accuracy using ISPD-2005 benchmarks as well as six industrial designs, and report 90% accuracy in predicting datapaths and 99.9% accuracy in predicting non-datapaths, out of 100 datapaths and 10K non-datapaths in their dataset.

Congestion estimation has attracted much attention in the research community. To address routability issues prior to the routing stage and to minimize turnaround time, modern placers are equipped with congestion estimators to guide the placement to achieve router-friendly placement solutions. Caldwell et al. [29] accurately estimate routed wirelength by comprehending floorplan aspect ratios for routability-driven placement. Roy et al. [214] propose a congestion-driven whitespace allocation algorithm during placement and they apply their algorithm in their placement tool ROOSTER. Westra et al. propose a constructive approach in [261] for placement. Furthermore, Taghavi et al. [238] propose *MILOR* to avoid routing infeasibility due to local congestion at the placement stage. Although the above works do not use machine learning, they provide essential background on modeling for congestion estimation before the routing stage.

In the area of three-dimensional ICs (3DICs), previous works have mainly performed analytical modeling of 3D HPWL using 2D placements. Mak and Chu [168] present a loose theoretical upper bound on the potential wirelength improvement possible with a 3DIC implementation of any design as compared to its 2DIC implementation. They report that for realistic sizes of vertical interconnects (VIs), the benefits will always be negative (-2% on average). Kim et al. [136] [141] use Rent's rule to predict wirelength distributions in 3DICs with two or more

dies as well as by varying the number of VIs. The authors derive analytical models of parasitics for the VIs and for buffer insertion, then derive models to estimate stage delays across dies. The authors also derive analytical models to estimate 3D power when heights and widths of VIs, and the number of buffers inserted into the netlist, are varied. These models do not account for IC implementation details such as floorplan context, technology libraries, signoff corners and constraints. Toufexis et al. [244] propose an in-built statistical prediction engine to estimate area, performance and power, thereby enabling a fast implementation-space exploration flow for 3DICs. The authors represent the chip as a cuboidal mesh, implement a power-ground network and compute power based on current drawn by standard cells. An interpolation scheme is used to predict power, with reported maximum modeling error of $\sim$58%.

In Section 3.2 of this thesis, we propose a novel modeling methodology for floorplanning with embedded memories, so as to minimize the risk of timing failures during multiphysics signoff. In Section 3.3 of this thesis, we develop fast and accurate models to predict whether a 2D placement is routable by using only information from the placement stage, i.e., no information from trial or early global routing. We also predict Pareto frontiers of maximum utilization, aspect ratio and number of metal layers at iso-performance using our models. In Section 3.4 of this thesis, we propose a modeling methodology to estimate 3DIC power using 2DIC implementations. Our models presented in Section 3.4 guide designers to choose appropriate floorplan contexts (e.g., aspect ratio, utilization, placement of embedded memories) so as to maximize benefits from 3DIC implementations relative to 2DIC implementations.

Modeling during Clock Network Synthesis

Prediction of clock network metrics is a sparsely explored area in CAD. Kahng et al. [106] use their METRICS infrastructure to collect design flow information during clock network synthesis, and perform data mining using *CUBIST* to estimate clock skew and insertion delays. They use *CTGen*, a commercial CTS tool in 2001, to conduct their studies on industrial testcases and report correlation coefficients of around 0.82 in predicting maximum and minimum insertion delay, maximum skew, and routing violations. Samanta et al. [218] use SVM regression [248] to estimate clock skew with an accurate delay model to size buffers and wires in a non-tree clock network. Ward et al. [258] propose placement of latches and local clock buffers using decision trees. The authors create models that take as inputs types and sizes of latches and clock buffers, densities, ratios, skew and delay. The authors develop a new similarity measure of placement solutions that they use in their flow, and report modeling accuracy of 91.8%, and 30% total power

savings using their model-guided latch and clock buffer placement.

In Section 4.3 of this thesis, we propose new model-guided optimization methods to minimize clock skew variation across multiple corners. Our optimization is guided using accurate machine learning models of clock signal delay.

Modeling during Routing

Dong et al. [63] show that only five metal layers are sufficient to route designs with up to 5M gates for signal and clock routing only (i.e., no PDN). Thereafter, the number of layers scales linearly, e.g., 50M gates require eight layers and 100M gates require 10 layers. The study in [63] uses analytical models and gate areas from 65nm process. Andreev et al. [9] have patented a dynamic programming technique that assigns segments of signal nets from M1 through to the top metal layer, minimizing the amount of metal layer area consumed by vias. The patent assumes that all layers have the same pitch and guarantees that the design is routable. The patent of Lin [158] describes a method to choose widths of various metal layers in the stack to minimize IR drop and RC delay of routed nets. The patent also describes material choices for pads and dielectric to achieve minimum RC delay.

In the area of congestion prediction, Brenner et al. [27] and Jiang et al. [99] propose approaches to estimate congestion at the global routing stage for congestion-driven placement. Wang et al. [255] and Zhong et al. [277] propose approaches to cure hotspots at the global routing stage. Several works use wire density to avoid congestion [86] [91] [231] [247]. Westra et al. [260] extract routing patterns (L/Z-shapes) to predict congestion. Kahng and Xu [130] propose a statistical model for congestion that comprehends effects of blockages and routing bends. He et al. [87], Liu et al. [163] [162], Pan and Chu [191], and Kim et al. [137] use global routers to predict congestion. Shojaei et al. [223] [224] propose a congestion-estimation framework with integer linear programming (ILP) at the global routing stage. Wei et al. [259] propose *Glare* for local and global congestion estimation. Qi et al. [201] use MARS to develop predictors of routing congestion using pin density and congestion maps from global routing. The authors report 13% reduction in the number of design rule violations when their predictor is used, as compared to using analytical models of routing congestion. Zhou et al. [278] propose a learning-based congestion model for detailed routing. The authors use parameters from global routing and achieve accuracy of $\sim$80% using multivariate adaptive regression splines.

In the area of photomask preparation and lithography simulation, several works apply machine learning to detect and classify lithographic "hotspots" [62] [170] [171] [172] [268]

[270] [273]. Ding et al. [61] develop a hotspot predictor for lithographic layout patterns using SVM, and use it with a detailed router to estimate layout printability. Yu et al. [272] propose a topological classifier to determine hotspots in lithographic mask patterns. The authors use SVM with RBF kernel in their modeling, and obtain their training and test datasets from an open-source GDSII library. Across six benchmarks from the GDSII library, the authors report classification accuracies that range between 86%–98%. In a much earlier work, Kahng and Muddu [109] perform response surface modeling of variation of linewidth (i.e., critical dimension) caused during lithography by accurately identifying sources of variation. The authors report coefficient of determination of up to 0.94 and a tight distribution of errors, with $\sim$70% of the errors being less than 1nm of linewidth variation.

Modeling during IC Testing

Callegari et al. [30] use classification to develop a feature-based rule learning algorithm and apply it to analyze silicon test measurement data. They uncover mismatches between design and silicon even in the presence of random noise. Fagot et al. [69] use the nearest neighbors algorithm to generate efficient test patterns for use during BIST. Fountain et al. [71] use expectation maximization algorithm to predict yield. The authors use data from die-level parametric and functional tests, and report modeling accuracy of 96%. Gosavi [79] uses SVM to classify defects in ICs as either permanent or intermittent. The author uses data from IC testing to train and test the model. Huang et al. [90] use SVM classification to predict defects in analog chips at the post-silicon stage. They use LNAs as their test circuit and are able to predict defects within an error of 2.9%. Robles et al. [210] patent a density-based feature encoding method to detect hotspots in lithographic mask patterns. The authors propose two-level SVM-based classification to minimize false positives and false negatives in the prediction.

Hardware Implementation of Learning Algorithms

Afifi et al. [2] propose a fully functional circuit design for accelerating the SVM learning phase based on sequential minimization optimization algorithm. The authors demonstrate that fixed-point design on FPGA has similar performance as floating-point algorithm on *Matlab*. Kuan et al. [147] propose a flexible and parallel implementation of SVM. Ardakini et al. [10] implement deep neural network using five transfer functions in 65nm foundry technology. Chakrabartty and Cauwenberghs [32] implement an analog system-on-chip 24-class SVM classifier [85] [311] for biometric signature verification by analysis of voice samples. Zhang et

al. [275] implement the learning vector quantization algorithm, which is a variant of an ANN classifier, using 16-bit wide datapaths and 75MHz frequency in 180nm.

2.2 Timing Analysis and Correlation

In this section, we review literature pertaining to our works on timing correlation described in Chapter 4. Prior works that quantify miscorrelations between signoff STA tools or propose methodologies to minimize tool divergence are limited. Kahng et al. [125] develop an internal incremental STA tool by using least-squares regression to model wire delay. They then use offset-based correlation with a signoff timing tool to minimize divergence in path slack estimates of their incremental STA tool, relative to the signoff tool. Their models are developed using the ISPD-2013 [293] gate-sizing contest library, and do not include any models for stage or cell delays, or for flip-flop setup times.

To model effects of temporal and spatial manufacturing variations on path delay, Ganapathy et al. [74] use multivariate regression. They report estimation errors to be within 5% of SPICE simulations. Tetelbaum [240] uses root-sum-square (RSS) of variations in stage delay and a weighted function of the worst case sum of variations in stage delay to estimate total path delay; path delay estimation errors of less than 5% are reported. Sinha et al. [226] propose use of RSS for delay variation in their announcement of the TAU-2013 contest to speed up timing analysis by using multicores and parallel computing techniques.

To model crosstalk effects, Sapatnekar [219] propose an analytical model that captures crosstalk-induced delay. This model lumps coupling capacitance to ground with the value of Miller coupling factor being 0, 1 or 2 based on the timing window overlap and switching directions of the signals. The effect of crosstalk on net delay is estimated using an iterative algorithm with runtime that is polynomial in the number of nets. The results are not verified with results from other tools or models. Xiao et al. [263] derive an analytical two-pole model for RC interconnect noise waveform calculation with coupling capacitance. A Newton-Raphson iteration is used to obtain the timing information. Correlation with SPICE shows good matching. However, the Newton-Raphson iteration is computationally expensive and may not be practical for use with SoC designs. Ilumoka [95] uses RBF to estimate interconnect crosstalk.

In correlating STA tools, Mishra et al. [176] recalculate clock uncertainties based on miscorrelation between two tools and apply the updated uncertainty values to achieve better timing correlation between the tools. Rakheja et al. [207] demonstrate that timing reports from design implementation tools, such as Synopsys *IC Compiler* [340], and signoff STA tools, such

as Cadence *Encounter Timing System* [288], can differ. They propose a manual and iterative approach to fix paths for which the tools have large divergence in timing estimates. Motassadeq [67] quantifies differences in output slew between Synopsys *HSPICE* [339] and *PrimeTime* [342] for *Nonlinear Delay Model* (NLDM) and *Composite Current Source* (CCS) [289] delay models. Thiel et al. [241] leverage the ability of PrimeTime (PT) [342] to output a SPICE netlist, and use SPICE simulation to calibrate the PT timing report. This work does not take crosstalk effects into consideration. Motassadeq et al. [66] extend this analysis flow by using PrimeTime SI (PTSI) [342] instead of PT to include crosstalk effects. Mohamed et al. [178] correlate PTSI-reported delta delay with coupling capacitance and drive strengths of the aggressor and victim. Venugopal et al. [251] characterize delays calculated by PTSI and correlated with HSPICE [339].

In correlating post-P&R timing in early stages of design, Alpert et al. [6] propose the adoption of physical synthesis in design flows to have better correlation with post-layout metrics such as worst negative slack (WNS), at the post-synthesis stage. Alpert et al. [8] propose analytical buffered delay models in the presence of blockages and report errors within 1% for three-pin nets. The authors propose integration of their models within a floorplanner for fast and accurate estimation of timing. Clarke et al. [52] propose detection of congestion-induced timing issues during synthesis, and prevention of congestion by avoiding decomposition of complex cells such as *MUX* and *XOR* to *NAND* and *AOI* cells during logic synthesis so as to reduce pin counts. Jones et al. [101] derive wireload models to estimate wire delay due to parasitics at the placement stage. The authors propose to divide the block into equal-sized regions, perform Steiner tree routing and use Rent's rule for fanout distributions in each region. Kim et al. [138] characterize standard cells and parasitics at different temperatures and propose thermal-aware delay models at the floorplanning stage. Vujkovic [252] proposes the use of multiple wireload models during synthesis for better correlation of post-layout wire delay. Yaldiz et al. [264] develop a closed-form model of SRAM latency that comprehends inter- and intra-die process variations. The authors demonstrate accuracy of their models in 90nm and report errors $\leq$15%.

In Section 4.1 of this thesis, we describe a novel methodology to correlate golden timing signoff tools. We demonstrate that our methodology is highly accurate and can be used to correlate design implementation and signoff timing tools as well. In Section 4.2 of this thesis, we present a methodology to map timing reports from non-crosstalk analysis to timing reports from crosstalk analysis. Both these techniques use layered or deep learning methodologies. The effectiveness of the proposed techniques is demonstrated across a wide range of technologies and designs.

2.3 Optimizations for Design Power, Cost and Reliability

In this section, we review related works pertaining to our optimizations described in Chapter 5. These are (i) quality of results (QoR) improvements with 3DIC flow and placement, (ii) design cost reduction through optimal multi-project, multi-resource scheduling, and (iii) task scheduling in multicore systems subject to lifetime reliability constraints.

2.3.1 3DIC Flow and Placement

While there is no golden EDA flow for 3DIC implementation, a number of researchers have implemented 3DICs using 2D EDA tools and flows in conjunction with in-house 3D design tools. Thorolfsson et al. [242] propose a 3D design flow based on a 2D flow to implement a FFT processor. Kim et al. [140] implement a multi-core processor based on commercial 2D EDA tools and use in-house tools to place the VIs. The authors verify the result through fabrication in Tezzaron 3D technology at the 130nm node. Another 3DIC implementation flow addresses design requirements for sequential 3D [18] technology that permits cell-level 3D integration. Panth et al. [196] [197] propose a "shrunk2D" (S2D) design flow for sequential 3D based on commercial EDA and in-house tools, and validate the flow on OpenSPARC T2 and other IPs [318]. This flow is, we believe, currently the strongest and full-featured in the research literature. S2D invokes commercial 2D P&R with scaled LEF and then partitions the result onto two tiers.

In the area of 3DIC placement, we broadly classify 3DIC placers as (i) folding- and partition-driven, or (ii) analytical. We highlight placers that evaluate by routing versus by HPWL in the discussion below.

Folding- and Partition-Driven 3DIC Placers

Cong et al. [55] propose a technique to fold a partitioned 2D netlist into multiple stacked dies. Folding reduces the wirelengths of nets that span different partitions, whereas wirelengths of nets that are not partitioned remain unchanged. Therefore, overall reduction in wirelength depends on which nets have been partitioned. Panth et al. [197] propose a Fiduccia-Matheyeses (FM)-based partitioner to partition a routed 2D netlist across two tiers. Their placer intentionally creates overlaps in a 2D placement by doubling the floorplan site capacity. The overlapped cells are then partitioned into two tiers to minimize the routing overflow. The authors divide the 2D placement region into multiple grids, then apply their partitioner in each grid to achieve an area-balanced min-cut bipartitioning within each grid.

Analytical 3DIC Placers

Previous works such as [13] [56] [57] [92] propose analytical placers for 3D. They extend the cost function in 2D analytical placers to 3D by adding a z-direction cost. Cong et al. [56] use the z-direction cost to model the number of VIs between tiers and uses a weight to control this number. The log-sum-exponent (log-sum-exp) method is used to make the HPWL metric continuously differentiable, and a conjugate gradient solver minimizes the metric. The authors implement their placer using multilevel coarsening and evaluate it using IBM-PLACE benchmarks [298] on four tiers, reporting 7% reduction in HPWL relative to 2D implementations. In [57], the same authors propose a mixed-size 3D analytical placer that decompose large cells into multiple smaller cells. The authors report 27% HPWL reduction relative to 2D implementations on IBM-PLACE benchmarks.

Goplen and Sapatnekar [78] propose an iterative thermal-aware force-directed 3DIC placer. In each iteration, the authors use finite element analysis to compute temperature, from which they calculate thermal forces that move cells from regions of high temperature to regions of low temperature. Kim et al. [135] propose use of a force-directed placer after FM-based partitioning. The chicken-and-egg loop between partitioning and placement leaves an opportunity for QoR improvement. Balabanov et al. [13] and Hsu et al. [92] consider VI sizes and locations in their density function and propose a novel weighted-average wirelength method instead of the log-sum-exp method to smooth the wirelength metric. These innovations enable their placer to achieve 13% less HPWL compared to [56], along with routability using a commercial EDA tool. Kim et al. [139] study several 3D wirelength objectives based on HPWL to improve estimation for routed 3D wirelength during VI planning stage that may be used for 3D placement. Recently, Lu et al. [165] have proposed a new cost function that is analogous to electrical field equations. They improve the density function to balance the placement among multiple dies, and report 27.5% improvement in HPWL relative to 2D implementations of the IBM-PLACE benchmarks.

In Section 5.1 of this thesis, we describe a new analytic placement tool that implements a "true 3D" wirelength objective. Our placement solutions are routable, and in conjunction with a commercial router, we achieve significant wirelength and power reductions as compared to the S2D flow.

2.3.2 Project Scheduling under Resource Constraints

Resource-constrained project scheduling has been solved in many different settings, with varying constraints and/or objective functions. A common objective is to minimize the

makespan [17]. Objective functions studied typically minimize project cost given time-dependent and/or resource-dependent penalties [146] [239].

Project Scheduling

Several previous works solve the scheduling problem for a single project with multiple activities [22] [50] [134] [143] [180] [217]. The activities can be either preemptive or nonpreemptive [17] [143] [217] [239]. Kolisch et al. [143] [144] formulate the *resource-constrained project scheduling problem* (RCPSP) and propose methods to generate RCPSP instances. They present the PSPLIB and MPSPLIB benchmark suites, along with optimal as well as heuristic solutions. Further variations involve the scheduling of activities that can execute in multiple *modes* [22] [143] [146] [217] [239]. These works consider that the resource usage and the time taken by an activity can vary across available modes; they provide optimal scheduling solutions across combinations of modes of activities. Mohring et al. [180] and Christofides et al. [50] provide branch-and-bound algorithms to solve the resource-constrained multi-activity single project scheduling problem. The authors of [180] further try to identify special cases that are solvable in polynomial time. Generally, solution frameworks involve linear or integer linear programming, although stochastic [134] and nonlinear [25] formulations have also been studied. Cyclic scheduling has been addressed in [11] [25], where sets of activities are executed indefinitely over time in a periodic fashion. The work of [134] is noteworthy in that its formulation permits temporary resource expansion, albeit for a penalty which features in the objective function. The formulation provided in [134] does not include precedence constraints within activities.

Several commercial tools and services exist today [292] [299] [301] [313] [328] that serve design project management needs. Some of these tools are specific to IC design such as [292] [299] [313], whereas the other tools can serve project management needs for any industry. These tools, however, are not based on combinatorial optimization.

Human Resource Scheduling

To optimize human resources at an enterprise scale, Li et al. [152] minimize the makespan of a single project with multiple activities, subject to upper bounds of human resources. Mohanty and Nayak [179] propose a particle swarm optimization (PSO) algorithm to optimize the tradeoff between cost and profit when a given number of employees are assigned to an activity. Their formulation considers employees with different skill competencies for different activities. Qiong et al. [203] propose an ant colony optimization (ACO) algorithm to

minimize the makespan of a project with multiple activities and precedence constraints between the activities. The activities are assumed to be non-preemptive, and the algorithm is applicable to general parallel machine scheduling problems.

Datacenter Job Allocations

In the datacenter literature, several works propose algorithms to handle job scheduling within a datacenter, e.g., to minimize makespan as well as other penalty functions. With energy consumption a major concern in modern datacenters, recent formulations by Friese et al. [73] propose multi-objective optimization of makespan and energy consumption. Furthermore, formulations for datacenters are focused on providing job scheduling solutions either in real-time or "online" [155], often by using live data from various thermal, network and rack utilization sensors.

In Section 5.2 of this thesis, we describe two new formulations that minimize overall schedule makespan and optimally allocate multiple resources across multiple projects subject to activity precedence and resource co-constraints. Our optimizations are performed "offline", that is, we do not monitor status of project executions in real-time during the optimization of our objective functions. Applications to industrial testcase instances show that our solvers achieve significantly better solutions than those actually used by management within a large IC design company.

2.3.3 Task Scheduling in Multi-core Systems

We taxonomize prior works on task scheduling for multi-core systems as reliability-constrained (RC) or non-reliability-constrained (NRC). RC task scheduling policies can be further classified as those that make system "lifetime guarantees" (LG) and those that make "no lifetime guarantees" (NLG). Existing RC-LG policies apply (1) dynamic power management (DPM), (2) dynamic thermal management (DTM), or (3) dynamic reliability management (DRM). Such works are "performance-guaranteeing" (PG) if they guarantee lower bounds on "acceptable performance".

NRC and RC-NLG Policies

Reiss et al. [208] present analysis of NRC task scheduling policies in Google datacenters. Each core maximizes throughput by operating at its worst-case temperature. The "BubbleWrap" work of Karpuzcu et al. [132] proposes a *DVSAM-Perf* policy to maximize perfor-

mance in the presence of delay degradation due to bias temperature instability. DVSAM-Perf is an example of RC-NLG policy.

RC-LG Policies

Mihic et al. [174] and Rosing et al. [213] perform detailed system modeling to devise a DPM policy to minimize total system energy subject to all tasks in a task graph completing execution within the multi-core system's lifetime. Rong et al. [212] propose a DPM policy to minimize system energy subject to meeting all task deadlines within the multi-core system's lifetime. Both these policies adjust voltage/frequency settings of a core to ensure all tasks complete execution within the system's lifetime.

Coskun et al. [53] propose five DTM policies to minimize thermal hotspots subject to completion of all tasks in a task graph while the multi-core system meets its lifetime. In [54], Coskun et al. devise four DTM task scheduling and migration policies. They propose that tasks should be scheduled on cores towards the periphery of the chip. The proposed policies in [53] and [54] do not guarantee any minimum frequency of operation of a core.

Srinivasan et al. [233] develop *RAMP*, a reliability simulator, and propose a DRM policy to adjust voltage/frequency settings of cores to maximize the lifetime of the multi-core system. Karl et al. [131] use a proportion-integral-derivative (PID) control system to determine the maximum voltage/frequency setting of a core to complete a given task. These policies do not guarantee any minimum frequency of operation of a core.

In Section 5.3 of this thesis, we describe a formulation that maximizes overdrive frequencies in multi-core systems subject to lifetime reliability constraints and balanced wearout of cores. We provide counterexamples showing that existing policies are suboptimal, in that they cannot guarantee lower bounds on acceptable performance and acceptable throughput. Our solver guarantees acceptable performance and acceptable throughput, and is an improvement over previous RC-LG policies.

Chapter 3

Design Productivity Gains through Improved Design- and Implementation-Space Exploration

This chapter presents four new applications of machine learning-based models for fast, accurate design- and implementation-space exploration. The first work describes the development of parametric as well as non-parametric power and area models for Network-on-Chip routers. A comprehensive suite of these models has been implemented in *ORION3.0*, and has been released on the web for download. *ORION3.0* enables efficient architecture-level (RTL-level) design-space exploration. The second work proposes a learning-based methodology to perform early prediction of timing failure risk given only the netlist, timing constraints, and floorplan context (wherein embedded memories are placed). This work can be used to identify which memories are "at risk", guide floorplan changes to reduce predicted "risk", and help refine underlying system-on-chip (SoC) implementation methodologies. The third work describes a learning-based methodology to predict whether a back-end-of-line (BEOL) stack-aware placement solution is routable without conducting trial or early global routing. The fourth work describes a learning-based methodology to estimate three-dimensional IC (3DIC) power benefit based on corresponding golden two-dimensional IC (2DIC) implementation parameters. Our models recommend a most-promising set of implementation parameters and constraints, and provide *a priori* estimates of 3D power benefits from 2DIC implementations only.

3.1 *ORION3.0*: A Comprehensive NoC Router Estimation Tool

Networks-on-Chip (NoCs) have proven to be highly scalable, low-latency interconnection fabrics in the era of many-core architectures, as evidenced by commercial chips such as the Intel 80-core [302], IBM Blue Gene [297] and Tilera TILE-Gx [348] processors. Because of their growing importance, NoC implementations must be optimized for latency and power [253] [43] [199] [183]. To facilitate early design-space exploration, accurate NoC power and area estimators are required.

One widely adopted approach for NoC power and area estimation, as embodied in ORION2.0 [115], is to develop a logic structure template for each NoC router component, namely, the input and output buffers, crossbar, and switch and virtual channel (VC) arbiters. Despite significant enhancements to improve leakage and clock power modeling, ORION2.0 still produces large estimation errors versus actual implementation. This is because there is often a mismatch between the actual RTL versus the templates assumed. Also, typical design flows involve sophisticated design steps that have complex interactions among them, making their effects difficult to characterize. For example, the crossbar is assumed to have a multiplexer tree structure, and the switch and VC arbiter is assumed to be a set of *NOR* and *INV* gates. However, after logic synthesis and technology mapping, the actual structure can be quite different. For example, *AOI* instances may be used instead of *NOR* and *INV* gates for the arbiter, and tri-state buffers may be used to implement the crossbar. Logic synthesis may also add buffers for performance optimizations. These synthesis and other design flow transformations are hard to predict.

An alternative approach is to use parametric regression with high-level analytical models. Although regression analysis is performed using actual physical implementation data, previous evaluations of this approach [116] show that high-level parametric regression leads to large estimation errors because important architecture details are often missing in these analytical models.

To illustrate the degree of inaccuracy, we show in Figure 3.1(a) the power estimation errors in 65nm for both ORION2.0 and the parametric regression approach based on high-level analytical models, as a function of the number of virtual channels in the router. The maximum errors are 185% and 75%, respectively. Similarly, in Figure 3.1(b), we show the power estimation errors in 90nm when the flit-width is changed.

As reviewed in Section 2.1.1, there is no previous work that comprehensively studies NoC area and power estimation using parametric and non-parametric modeling techniques. In

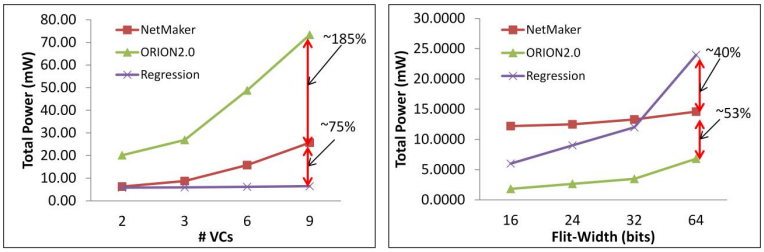


Figure 3.1: Poor estimations by ORION2.0 [115] and the previous parametric regression approach [116]. *Netmaker* vs. ORION2.0 vs. regression at (a) 65nm and (b) 90nm.

this work, we propose comprehensive parametric and non-parametric modeling methodologies that fundamentally differ from approaches such as ORION2.0, in that the estimation models are derived from actual post place-and-route (P&R) data that correspond to the actual RTL generator and target cell library. Following this paradigm, we describe two approaches.

The first approach is based on parametric modeling. Our work is a substantial departure from the ORION2.0 approach because no logic template is assumed for any router component block. Instead, for each component block in the router RTL, appropriate parametric models are derived from the post-synthesis netlists by observing how instance counts change with microarchitectural, implementation, and operational parameters. We call these models ORION_NEW. We perform least-squares regression (LSQR) with actual post-P&R power and area data to refine these ORION_NEW models. The resulting parametric models achieve worst-case errors significantly better than those of ORION2.0 as well as previous non-parametric regression approaches [116]. We describe modeling extensions that enable more detailed flit-level power estimation when integrated with simulation tools such as *GARNET* [3]. Our parametric modeling methodology does not require the architect or developer to understand how the architectural components are implemented. Rather, the methodology relies on a one-time characterization of post-synthesis data to derive parametric models of component blocks, and automatic fitting of these models to post-P&R data using parametric regression.

The second approach is based on non-parametric modeling, which was first described in [116]. In this approach, estimation models are also derived from actual post P&R layout power and area data that correspond to the actual RTL generator and target cell library. As described in [116], the non-parametric modeling approach is powerful in that it can automatically derive accurate estimation models based on a sample set of post P&R results. In [116], Multivariate Adaptive Regression Splines (MARS) [85] linear splines were used to derive the estimation models. In this work, we extend the ideas of [116] by incorporating four other metamodeling

techniques for automatic model generation: Radial Basis Functions (RBF), Kriging (KG), Multivariate Adaptive Regression Splines with cubic splines, and Support Vector Machines (SVM) regression [85]. This non-parametric modeling approach does not require the architect or developer to understand how the architectural components are implemented.

Our main contributions are as follows:

1. We describe a new parametric modeling methodology that derives accurate parametric models from actual post-synthesis netlists generated from actual router RTLs by observing how instance counts change with microarchitectural, implementation, and operational parameters. The post-synthesis netlists accurately capture contributions from both the control and data paths. The derived models are highly accurate and robust across multiple router RTLs, and across microarchitecture, implementation, and operational parameters.

2. We demonstrate that parametric regression with accurate models can significantly reduce the worst-case error compared to previous template-based approaches as well as non-parametric regression approaches for NoC routers.

3. We demonstrate that popular non-parametric metamodeling techniques, namely, Radial Basis Functions, Kriging, Multivariate Adaptive Regression Splines, and Support Vector Machine regression, can be highly accurate (worst-case errors less than 20%) in NoC power and area estimations.

4. We are the first to propose a detailed, efficient, and fine-grained flit-level power estimation model that seamlessly integrates with full-system NoC simulators.

3.1.1 *ORION3.0* Parametric Modeling Methodology

As shown in Figures 3.1(a) and (b), ORION2.0 [115] and the previous parametric regression techniques [116] have large errors compared to implementation. This is because NoC architecture template-based models are incomplete. They do not consider the impact of frequency scaling, and do not consider more optimized implementations of blocks such as the crossbar.

We now describe the *ORION_NEW* modeling of each component in a modern on-chip network router. We have developed these models by analyzing post-synthesis and post-P&R netlists of two RTL generators, *Netmaker* [314] from Cambridge and the *Stanford NoC* router [335]. Figure 3.2 shows the component blocks of a router, i.e., crossbar, switch and VC arbiter, and input and output buffers [199]. We model instances (or gates) in each component block

because our studies show that accurate estimations of area and power are possible only if the instance modeling is accurate. The microarchitecture parameters used are #Ports (P), #VCs (V), #Buffers (B) and Flit-width (F).

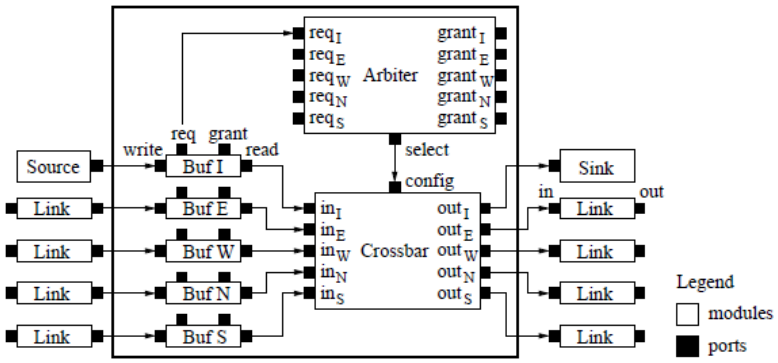


Figure 3.2: Router architecture [253].

The new model explicitly accounts for control and data resources in the router. The new modeling elements are

1. tri-state crossbar model;
2. control resources such as FIFO select and decode logic signals in the input and output buffers;
3. additional input buffer resources for delay-optimized arbiters [199];
4. output buffer model to store only head flits;
5. clock and control logic resources; and
6. clock frequency-based derating of total number of instances in the router.

Crossbar (XBAR) model. The crossbar (XBAR) is responsible for connecting input ports to output ports so that all flit bits are transferred to output ports [60]. ORION2.0 crossbar models consider two implementations: matrix [199] and multiplexer tree [115]. The multiplexer tree is the smaller of these in terms of instance counts and area, and is modeled as $P \times P \times F$ multiplexers *at each level* of the tree.

Modern router RTLs such as *Netmaker* and *Stanford NoC* use a simpler and smaller crossbar implementation where each flit bit is controlled using a tri-state buffer, which can be modeled as a $2:1$ MUX. Hence, the *total* number of such MUXes required is: $P \times P \times F$. The

new model reduces the instance count by a factor of $(P-1)$ when compared to the multiplexer tree implementation.

Switch and VC arbiter (SWVC) model. The switch and VC arbiter (SWVC) generates control signals for the crossbar such that a connection is established between input buffers and output ports [60]. ORION2.0 adds an overhead of 30% to the arbiter by default. Our analysis indicates that this overhead is not needed with frequencies in the range 400MHz–900MHz for process nodes 45nm to 130nm. Beyond this range of frequency, a derating factor must be applied. The ORION_NEW model for switch and VC arbiter is: $9(P^2V^2+P^2+PV-P)$. The constant factor 9 arises because six 2-input *NOR* gates, two *INV* gates, and one *D flip-flop* are used to generate one grant signal on each path. The new model removes the overhead factor of 30%.

Input buffer (InBUF) model. The input buffer (InBUF) holds the entire incoming payload of flits at the input stage of the router for decode [60]. ORION2.0 models only the buffer instances and does not take into account control signals which are needed at this stage for decode; these include FIFO select, buffer enable control signals, and logic for housekeeping such as the number of free buffers available per VC, VC identification tag per buffer, etc. As a result, ORION2.0 underestimates the instances at the input stage of the router.

In ORION_NEW, we model control signals and housekeeping logic in addition to the actual FIFO buffers. Modern routers implement the same stage VC and SW allocation to optimize delay [199], leading to doubling of input buffer resources. Hence, in the new model the number of FIFO buffers is $2\times P\times V\times B\times F$. The control signals for decoding the housekeeping logic are modeled as: $180\times P\times V+2\times P^2\times V\times B+3\times P\times V\times B+5\times P^2\times B+P^2+F\times P+15\times P$ (as analyzed from the post-synthesis and post-P&R netlists). Each constant factor in the model denotes the number of instances per path. For example, the 180 factor accounts for instances to generate FIFO select signals and flags for each buffer in the $P\times V$ path. The smaller constant factors 2, 3 and 5 account for instances that realize local flags in the decode logic. The factor 15 corresponds to the number of buffers in each FIFO select path of an input port.

Output buffer (OutBUF) model. The output buffer (OutBUF) holds the head flits between the switch and the channel for a switch with output speedup [60]. ORION2.0 models output buffers in exactly the same way as it models input buffers; this is inaccurate for modern routers that use hybrid output buffers, and leads to an overestimate of the instance count. The output buffers

need to store enough flits to match speeds between the switch and the channel. At the output, these buffers are used to stage the flits between the switch and channel when channel and switch speeds mismatch. Instead of using $P \times V \times B \times F$ in ORION2.0, output buffers are proportional to $P \times V$. There are several control signals per port and VC associated with each buffer, which makes the overall instance counts grow in the new model as $P \times (80 \times V + 25)$. The constant factor 80 accounts for the instances used to generate flow control credit signals for each VC, while the constant factor 25 accounts for buffers and flags.

Clock and control logic (CLKCTRL) model. The clock and control logic (CLKCTRL) models clock buffers and control logic routing resources as clock frequency scales. ORION2.0 does not model impact on these resources because of frequency scaling. ORION_NEW models these resources as 2% of the sum of instances in the SWVC, InBUF and OutBUF component blocks.

Frequency derating model. As frequency changes, timing constraints change. To meet setup time at higher frequencies, buffers are inserted which leads to an overall increase in instance counts in the design. ORION2.0 scaling is agnostic to implementation parameters such as clock frequency. This causes large errors in area and instance counts at higher frequencies for component blocks such as SWVC, InBUF and OutBUF. We find that the number of instances in the crossbar does not vary significantly with frequency because there are no critical paths; we thus ignore the effects of frequency on the crossbar.

To derate for frequency, we find the frequency below which instance counts change by less than 1%. In 65nm technology, this is 400MHz for both *Netmaker* and *Stanford NoC* routers. We derate instance counts by a multiplier $\Delta Instance$ that is based on this frequency as: $\Delta Instance = \Delta Frequency \times ConstantFactor$. The constant factor depends on the amount of control logic versus FIFO for each component block. To account for setup buffers, a fitted constant factor of 1 is used in SWVC and InBUF, and a fitted constant factor of 0.03 is used in OutBUF.

Table 3.1 summarizes the ORION_NEW modeling of instance counts of each component block. Key elements of our modeling methodology (see details in Table 3.2) include the following.

- Multiple parametrized NoC RTL generators: *Netmaker* [314] from Cambridge and the *Stanford NoC* [335].

- Range of values of microarchitecture parameters (#Ports (P), #VCs (V), #Buffers (B) and Flit-width (F)) and implementation parameters (clock frequency and technology node).
- Operational parameters for power calculation: toggle rate[1] (TR) and static probability of 1's in the input (SP).
- Multiple commercial tools: Synopsys *Design Compiler vC-2009.09-SP2* (DC) [337] and Cadence *RTL Compiler v6.1* (RC) [285], with options to preserve module hierarchy after synthesis because we analyze each router component block. We compare instance counts, area and power reported by each tool to ensure that for a given RTL these results do not vary by more than 10% across the two synthesis tools.
- Cadence *SOC Encounter v10.12* (SOCE) [287] with die utilization of 75% and die aspect ratio of 1.0 to place and route the synthesized router netlist.
- Synopsys *PrimeTime-PX vF-2009.06-SP2* (PT-PX) [342] to run power analysis based on the post-P&R netlist, SPEF [333] and SDC [21].
- *MATLAB vR2012b* [311] function *lsqnonneg* for regression analysis.

Table 3.1: ORION_NEW model for Instances.

Component	Equation
XBAR	P^2F
SWVC	$9(P^2V^2+P^2+PV-P)$
InBUF	$180PV+2PVBF+2P^2VB+3PVB+5P^2B+P^2+PF+15P$
OutBUF	$25P+80PV$
CLKCTRL	$0.02\times(SWVC+InBUF+OutBUF)$

Figure 3.3 shows the flow we use to develop ORION_NEW models for each component block of the router. There are two ways to estimate NoC area and power using the ORION_NEW models, manual and LSQR, as shown in Figure 3.4. The benefits of each are described below.

- Both the approaches achieve minimum estimation error when the router RTLs are modular, so that the instance count and area numbers per component block can be calculated.

[1]Toggle rate is defined as the average number of times a signal transitions from high to low (or low to high) per clock cycle. We assume the toggle rate of the clock signal to be 1.

Table 3.2: ORION_NEW Methodology: Tools and Parameters.

Stage	Tool	Options
RTL	*Netmaker* *Stanford NoC*	ISLAY config default
μarch	Ports; VCs; BUFs; Flit-Width	$P = \{5, 6, 8, 10\}$; $V = \{2, 3, 6, 9\}$ $B = \{8, 10, 15, 22\}$; $F = \{16, 24, 32, 64\}$
Impl	Clock Freq Tech Nodes	*Freq* = {400, 700, 1200, 2000}MHz 45nm = OpenPDK45 from NCSU/OSU {45nm, 65nm} = TSMC GS, GP resp. {90nm, 130nm} = TSMC G, GHT resp.
Op	Toggle Rate Static Prob of 1's	$TR = \{0.2, 0.4, 0.6, 0.8\}$ $SP = \{0, 0.25, 0.5, 0.75, 1.0\}$
Syn	Synopsys *DC* (*vC-2009.06-SP2*) Cadence *RC* (*v6.1*)	*compile_ultra -exact_map* *-no_autoungroup -no_boundary_optimization* *report_area -hierarchy; report_power -hierarchy* default synthesis flow
P&R	Cadence *SOCE* (*v10.12*)	*set_default_switching_activity -input_activity TR* *propagate_activity* remaining flow is default
Power	Synopsys *PT-PX* (*v2009.06-SP2*)	*set power_enable_analysis true* *set power_analysis_mode averaged* *set_switching_activity -toggle_count TR* *-static_probability SP -type inputs* *read_sdc router.sdc; read_parasitics router.spef*
Regression	*MATLAB v2012b*	*lsqnonneg*

- The manual approach requires knowledge of process node and finer implementation details such as (G, LP) $\times$ (HVT, NVT, LVT) $\times$ (bc, wc) to correctly select a technology library file. The regression analysis approach, on the other hand, is agnostic of implementation details and only depends on the set of training data.

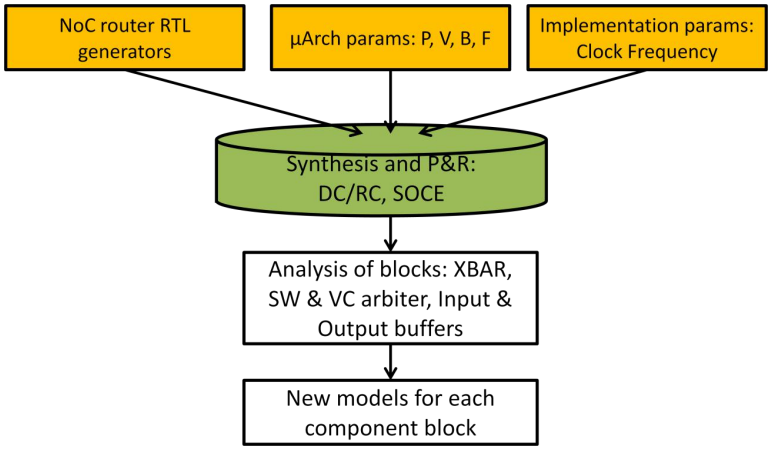


Figure 3.3: High-level flow used to develop the ORION_NEW models.

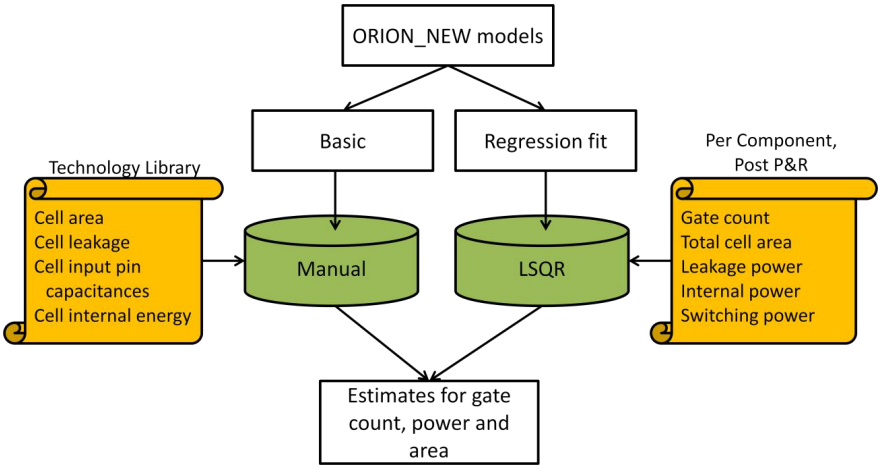


Figure 3.4: High-level view of power and area estimation methodology using manual and regression analysis (LSQR) approaches.

- The manual approach leads to faster estimation since it only involves technology library lookups and plugging-in of library values into the ORION_NEW models. In contrast, the regression analysis approach requires synthesis and P&R to be performed on the router RTL to generate data points for the training set. On an Intel Core i3 2.4GHz processor, the runtime of the manual approach when used with ORION2.0 code is less than 10ms, whereas the regression analysis approach takes 64 SP&R runs to generate the data points for training.[2]

[2]In our simulations we use 64 data points for training and 192 data points to test the model.

- It is extremely difficult to capture fine-grained implementation details in ORION_NEW models, e.g., area and power contribution of wires after routing, and change in coupling capacitance and power after metal fill. These missing details cause large estimation errors versus actual implementation when the manual approach is used. In order to reduce errors with respect to implementation, the regression analysis (LSQR) approach with post-P&R area and power is preferred.

Manual Approach to Estimate NoC Power and Area

This approach uses ORION_NEW models along with the technology library file of the process in which the router is to be fabricated. The key ingredients of this approach are

- microarchitecture parameters $\{P, V, B$ and $F\}$ and implementation parameters (clock frequency and technology node);
- cell areas, leakage, internal energy and load capacitance; and
- toggle rate.

ORION_NEW simplifies the design of a NoC, using only a few standard cells. The instance count for each component block for a given set of router microarchitecture parameters is calculated from Table 3.1. Cell area is obtained from technology files. The area calculation, along with TSMC standard-cell names in parentheses, is shown in Table 3.3.

Table 3.3: Area models using instance count.

Component	Logic (TSMC Cell Name)	Area
XBAR	*MUX2* (MUX2D0)	$Area_{MUX} \times XBAR_{insts}$
SWVC	6 *NOR2*, 2 *INV*, 1 *DFF* (NR2D1, INVD1, DFQD1)	$\left(\frac{6Area_{NOR}+2Area_{INV}+Area_{DFF}}{9}\right) \times SWVC_{insts}$
InBUF + OutBUF	1 *AOI*, 1 *DFF* (AOI22D1, DFQD1)	$\left(\frac{Area_{AOI}+Area_{DFF}}{2}\right) \times (In+Out)BUF_{insts}$
CLKCTRL	1 *INV*, 1 *AOI* (INVD1, AOI22D1)	$\left(\frac{Area_{AOI}+Area_{INV}}{2}\right) \times (CLKCTRL)_{insts}$

Power has three components: leakage, internal and switching. Leakage power is static power when the cell is not transitioning between logic states. It is dependent on current state

of the input pins of the cell as well as process corner, voltage, and temperature. Internal and switching power together constitute dynamic power, which varies with operating voltage, capacitive load and frequency of operation. Internal power is the power dissipated inside a cell and consists of short-circuit power and switching power of internal nodes; switching power is the power consumed when a load capacitance on a net is charged and discharged.

In ORION_NEW, toggle rate (*TR*) is equal to the average toggle rate of all input signals in the crossbar, switch and VC arbiter, and control logic in input and output buffers. We assume that buffer cells toggle at 25% of the input toggle rate, since multiple VCs do not require buffer contents to change in every cycle.

Leakage power calculation. For leakage power, the model uses the weighted average of the state-dependent leakage of the cells. Equations (3.1)-(3.4) are used to calculate the leakage power of each component block.

$$P_{leak\_XBAR} = MUX_{leak} \times XBAR_{insts} \tag{3.1}$$

$$P_{leak\_SWVC} = \left(\frac{6NOR_{leak} + 2INV_{leak} + DFF_{leak}}{9} \right) \times SWVC_{insts} \tag{3.2}$$

$$P_{leak\_BUF} = \left(\frac{AOI_{leak} + DFF_{leak}}{2} \right) \times (In + Out) BUF_{insts} \tag{3.3}$$

$$P_{leak\_CLKCTRL} = \left(\frac{AOI_{leak} + INV_{leak}}{2} \right) \times (CLKCTRL)_{insts} \tag{3.4}$$

Internal power calculation. For internal power, table lookups in technology library files return the internal energy of a standard cell given its load capacitance[3] and input slew value of $\approx 5 \times FO4\ delay$.[4] Internal energy is the minimum of the rise and fall energies. Equations (3.5)-(3.8) are used to calculate internal power of each component block.

$$P_{int\_XBAR} = MUX_{int} \times TR \times XBAR_{insts} \tag{3.5}$$

[3]Load capacitance of a cell depends on its fanout and the cell(s) it drives. We use a fanout of one. The cells driven depend on the component of the router as shown in Table 3.3. For example, one DFF drives another DFF and one AOI in the input and output buffers. So, the load capacitance of the DFF is the sum of input pin capacitances of one DFF and one AOI.

[4]The $FO4$ delay is the delay of a minimum-sized INV driving four identical INV instances and is a standard proxy for switching speed in a given process technology. The resulting slew time values are 80–100ps for 45nm and 65nm technologies.

$$P_{int\_SWVC} = (6NOR_{int} + 2INV_{int} + DFF_{int}) \times TR \times SWVC_{insts} \tag{3.6}$$

$$P_{int\_BUF} = (AOI_{int} + 0.25DFF_{int}) \times TR \times (In + Out)BUF_{insts} \tag{3.7}$$

$$P_{int\_CLKCTRL} = (AOI_{int} + INV_{int}) \times TR \times (CLKCTRL)_{insts} \tag{3.8}$$

Switching power calculation. For switching power, the load capacitance is calculated as the sum of the input capacitances of pins that are driven by a net and the wire capacitance of the net. The wire capacitance is approximately calculated as a constant factor times the total pin capacitances. This constant factor is set to 1.4 in 65nm and is assumed to decrease by 14% with each successive process node shrink. Equations (3.9)-(3.12) are used to calculate switching power of each component block.

$$P_{sw\_XBAR} = XBAR_{load} \times TR \times XBAR_{insts} \tag{3.9}$$

$$P_{sw\_SWVC} = SWVC_{load} \times TR \times SWVC_{insts} \tag{3.10}$$

$$P_{sw\_BUF} = (In + Out)BUF_{load} \times TR \times (In + Out)BUF_{insts} \tag{3.11}$$

$$P_{sw\_CLKCTRL} = (CLKCTRL)_{load} \times TR \times (CLKCTRL)_{insts} \tag{3.12}$$

The following steps below describe how total area and power are estimated using the ORION_NEW models and equations above.

1. Choose microarchitecture parameters (*P, V, B, F*), clock frequency, and average toggle rate at inputs.

2. Use models in Table 3.1 to calculate the instance count of each component block of the router.

3. Use models in Table 3.3 to calculate the area of each router component block. Total area is calculated as the sum of areas of all blocks.

4. Obtain state-dependent leakage of cells from technology library files. Use Equations (3.1)–(3.4) to calculate leakage power of each component block. Total router leakage power is calculated as the sum of leakage power of all component blocks.

5. Obtain internal energy of cells from technology library files. Use Equations (3.5)–(3.8) to calculate internal power of each component block. Total internal power is calculated as the sum of internal power of all component blocks.

6. Obtain input pin capacitances of cells from technology library files. Use Equations (3.9)–(3.12) to calculate switching power of each component block. Total switching power is calculated as the sum of switching power of all component blocks.

7. The total power dissipated by the router is calculated as the sum of total leakage power, total internal power and total switching power.

Regression Analysis Approach to Estimate NoC Power and Area

As another approach to estimation of router area and power, we use parametric regression to fit parameters for cell area, leakage, internal energy, and load capacitance into ORION_NEW models. This approach requires instance counts, area, and total leakage, internal and switching power of each component block of the router from post-P&R results. Options are set in synthesis tools to preserve module hierarchy and names. Constrained least-squares regression (LSQR) is used to enforce non-negativity of coefficients (cell area, leakage, internal energy, load capacitance). We use the *MATLAB* [311] function *lsqnonneg* for this purpose, and tool options as given in Table 3.2.

LSQR is applied to fit a model of post-P&R instance counts for each router component block. Our training set has 64 data points. Parametric LSQR is set up as

$$a_1 \cdot Insts_{mod\ <component>} + a_0 = Insts_{tool\ <component>} \tag{3.13}$$

where $Insts^R_{mod\ <component>}$ is the refined instance count of each component block after LSQR. The refined instance count is used to fit models of post-P&R area and power as

$$b_1 \cdot Insts^R_{mod\ <component>} + b_0 = Area_{tool\ <component>}. \tag{3.14}$$

In Equation (3.14), b_1 is the fitting coefficient for cell area, and the coefficient b_0 accounts for the routing overhead.

We model leakage, internal and switching power as

$$\begin{aligned}
&\{c_5,\, d_5,\, e_5\} \cdot Insts^{R}_{mod\ XBAR} + \{c_4,\, d_4,\, e_4\} \cdot Insts^{R}_{mod\ SWVC} + \\
&\{c_3,\, d_3,\, e_3\} \cdot Insts^{R}_{mod\ InBUF} + \{c_2,\, d_2,\, e_2\} \cdot Insts^{R}_{mod\ OutBUF} + \\
&\{c_1,\, d_1,\, e_1\} \cdot Insts_{mod\ CLKCTRL} = \{P_{leak},\, P_{int},\, P_{sw}\}_{tool}
\end{aligned} \tag{3.15}$$

where coefficients $\{c_5, \cdots, c_1\}$ are used to fit cell leakage power, and similarly $\{d_5, \cdots, d_1\}$ and $\{e_5, \cdots, e_1\}$ are respectively used to fit internal energy and load capacitance.

It is possible to skip the instance count refinement step (Equation (3.14)) and directly perform LSQR for area and leakage, internal, and switching power using the above equations. We observe that average error can change by 3% in either direction by omitting the instance count refinement step. Note that it is necessary to perform per-component LSQR: if LSQR is performed for the entire router's area or power, large errors result because multiple components have the same parametric combination of (*P, V, B, F*). Failing to separate these contributors to area or power can result in large errors, e.g., in 65nm we have experimentally observed worst-case errors of 296% for power and 557% for area. Thus, it is important to preserve module hierarchy during synthesis in the model development flow.[5]

Extension to Flit-Level Power Modeling

The dynamic power models used in ORION2.0 and ORION_NEW do not consider bit encodings in a flit, which can lead to significant errors in dynamic power estimation. As an example, consider two encodings with two consecutive 8-bit flits, where every flit has exactly four bits as 1. In the first encoding, the two consecutive flits are 8b'11110000 and 8b'11110000. In the second encoding, the two consecutive flits are 8b'11110000 and 8b'00001111. In the first encoding, there are no toggles per consecutive flits, whereas in the second encoding there are eight toggles per consecutive flits. Clearly, the second encoding will lead to higher dynamic power than the first one. To model this effect, we use the flow shown in Figure 3.5. Before using a testbench, the netlists must pass an equivalence check using tools such as Synopsys *Formality* [338]. We inject different bit encodings in the input during simulation over 10000 cycles using *GARNET* [3], and the resultant value change dump (VCD) is validated using a wave-

[5]Use of hierarchical synthesis in general leads to lower instance counts, standard-cell area, and total power as compared with flat synthesis results. This comes at the cost of frequency (timing slack), since flat optimization across module boundaries can sometimes achieve better timing results. For our selection of microarchitecture and implementation parameters, hierarchical synthesis on average has 35% fewer instances, 48.8% less standard-cell area and 49.4% less total power compared with flat synthesis. The runtimes for hierarchical and flat synthesis are within 5% of each other.

form analyzer such as Synopsys *DVE* [346]. A satisfactory VCD is used as input to Synopsys *PrimeTime-PX* [342] to obtain power values. Regression analysis is performed using the tool-reported power values with the ORION_NEW estimates to obtain an enhanced ORION_NEW model for flit-level power estimation. These models may be invoked by NoC full-system simulators such as *GARNET* to obtain very accurate estimates.

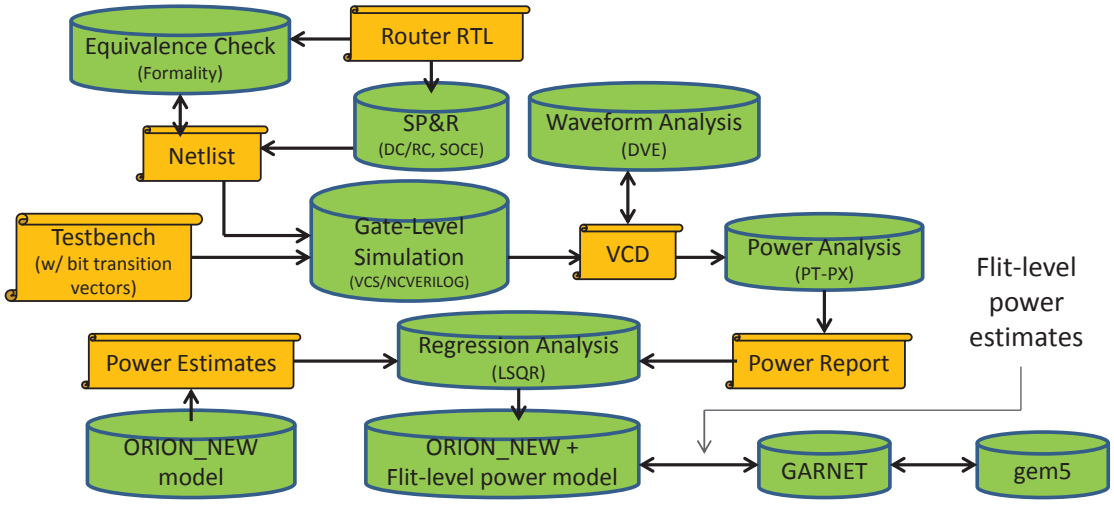


Figure 3.5: Methodology to enhance ORION_NEW dynamic power models with flit-level power estimation.

3.1.2 *ORION3.0* Non-Parametric Modeling Methodology

Non-parametric regression techniques provide another approach to estimate NoC power and area [116]. Such techniques can considerably reduce modeling efforts because they require only the microarchitectural, implementation, and operational parameters as input variables. The models determine the interactions between these input variables and how they affect the output (or response). This alleviates the effort needed to model a NoC, as details of architecture-level implementations are avoided. At the same time, non-parametric regression approaches are scalable across multiple router RTLs.

We now give a brief background on four widely used non-parametric regression or metamodeling techniques. We have reviewed previous works in Section 2.1.1 that apply these techniques for modeling at the architecture-level. Metamodeling techniques can be broadly classified [85] into linear regression-based methods, interpolation-based methods, neural network and kernel-based smoothing methods, and additive tree-based methods. Popular techniques for estimation purposes are Artificial Neural Networks (ANN), k-Nearest Neighbors (kNN), b-Splines,

Radial Basis Functions (RBF), Kriging (KG), Polynomial Regression (PR), Support Vector Machine Regression (SVM) and Multivariate Adaptive Regression Splines (MARS) [100] [266]. Recent [267] as well as previous studies [100] show that RBF, KG, and MARS models have higher estimation accuracy than others. Jin et al. [100] show that RBF excels in accuracy and robustness for *sparse* as well as *dense* training datasets whereas MARS and KG have high accuracy for dense training datasets.[6]

We use the metamodeling techniques to estimate NoC area and power by developing models using training data. We first perform synthesis using Synopsys *Design Compiler vC-2009.06-SP2* [337], followed by place and route using Cadence *SOC Encounter v10.12* [287], of the *Netmaker* [314] router RTL. Next, we generate area and power reports for these designs to use as training and test data points. Figure 3.6 shows our synthesis and P&R flow; Table 3.2 lists the architecture, implementation, and operational parameters; and Table 3.4 lists the tool options used in our experiments. We generate 256 data points for our experiments. We use two sampling methodologies to generate the training sets – a modified Latin Hypercube Sampling (LHS) [100], and a restricted sampling methodology which samples only values from the lower ranges of the parameters (*B, V, P, F*). Our LHS methodology (for 64 data points of four variables) is as follows.

1. Generate 64 normalized LHS samples over four parameters using the *MATLAB vR2012b* command *lhsdesign(64, 4)*.

2. Maximize the minimum distance between samples by using the *MATLAB vR2012b* command *bsxfun* [311].

3. Map the samples generated in the previous step to our ranges of *B, V, P* and *F* parameter values by selecting the value for each parameter which is closest to the value occurring in the sample.

4. Adjust the frequency of the values to make the samples uniformly distributed across our ranges of values for *B, V, P* and *F* so that each of them occurs the same number of times in the training set.

The restricted sampling methodology does not include higher values of the microarchitectural parameters in the training set. More precisely, the resulting training sets omit all values of $\{B = 7\}$, or of $\{P = 9\}$, or of $\{V = 7\}$, or of $\{F = 64\}$.

[6] We define a training set as *sparse* if it contains at most 20% of the total data points. We define a training set as *dense* if it contains at least 30% of the total data points.

Unlike previous approaches using MARS [116], we model leakage, internal, and switching power separately. This results in a more accurate fit of the training data because each of these components of power does not change in the same fashion with microarchitectural, implementation, and operational parameters. Figure 3.7 shows the flow of our methodology.

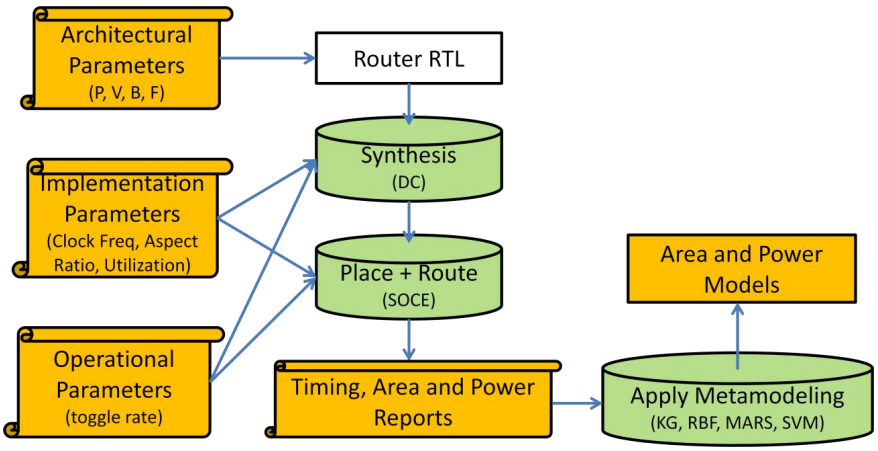


Figure 3.6: Implementation flow to generate training and test data points.

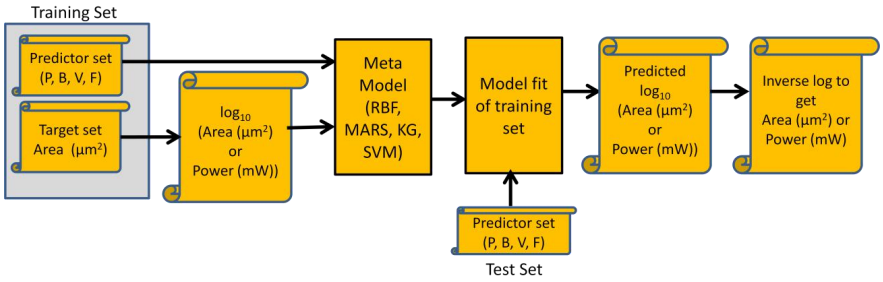


Figure 3.7: Area and power modeling and prediction flow.

We assess the goodness of fit of RBF, KG, MARS and SVM models using three metrics.

- *Magnitude of Mean Error* (MME): This is the mean of the magnitude of errors at each predicted output using the test data points.
- *Root Mean Square Error* (RMSE): This is the square root of the mean of the sum of squared error for the predicted outputs using the test data points.
- *Maximum Error* (MAXE): This is the maximum of the magnitude of errors at each predicted output using the test data points. We include this metric to give a sense of the worst-case error, which is of practical concern for hardware designers and computer architects.

Table 3.4: Metamodeling Methodology: Tools and Parameters.

Stage	Tool	Values
LHS	*MATLAB*	{lhsdesign, bsxfun}
KG	*DACE*	Reg model = {Order 1 and 2 Poly} Corr model = {EXP, GAUSS}, θ = {40}
RBF	*RBF2*	Type = {Ridge Regression} Func. Type = {GAUSS, MULTIQUADRIC} $2 \leq r \leq 0.5$
MARS	*ARESLAB*	Max Basis Funcs = {50} Max Interactions = {3} Spline Type = {linear, cubic}
SVM	*LIBSVM* v3.12	Type = {nu-SVR} Kernel = {RBF}

3.1.3 Experimental Setup and Results

We set up experiments as described in Table 3.2. We use parameters and tools for our experiments as listed in Table 3.2. To account for process variation, we perform multi-mode multi-corner place-and-route by defining two scenarios for setup and hold analyses. The *nominal* scenario uses {ss, 0.85V, 125C}[7] and {ff, 1.30V, 125C} for setup and hold corners, respectively. The *overdrive* scenario uses {ss, 1.10V, 125C} and {ff, 1.30V, 125C} for setup and hold corners, respectively. We discuss the results in two parts: (i) ORION2.0 versus ORION_NEW estimation of area and power, and (ii) impact of our regression analysis approach versus the approach used in prior work of [116]. Comparisons are made with respect to post-P&R instance counts, power and area outcomes, and both router RTL generators, *Netmaker* [314] and *Stanford NoC* [335].

ORION2.0 versus ORION_NEW Comparisons

Since the instance counts per component are at the core of the ORION_NEW model, we compare ORION2.0 estimates of instance (or gate) counts, as well as the ORION_NEW model estimates, with implementation (post-P&R) for each component block. Figures 3.8(a), 3.9(a),

[7]We represent process, voltage, temperature (PVT) corners as 3-tuples – (process, voltage, temperature). Our corners consist of two process conditions {ss, ff}, four voltages {0.85V, 1.05V, 1.10V, 1.30V} and one temperature {125C}.

and 3.10(a) show the large errors in ORION2.0 for crossbar, output buffer and input buffer respectively, and Figures 3.8(b), 3.9(b), and 3.10(b) show the significant reduction in estimation errors for these components with ORION_NEW models. ORION2.0 and ORION_NEW are plotted in different graphs because of the large errors in instance counts for ORION2.0.

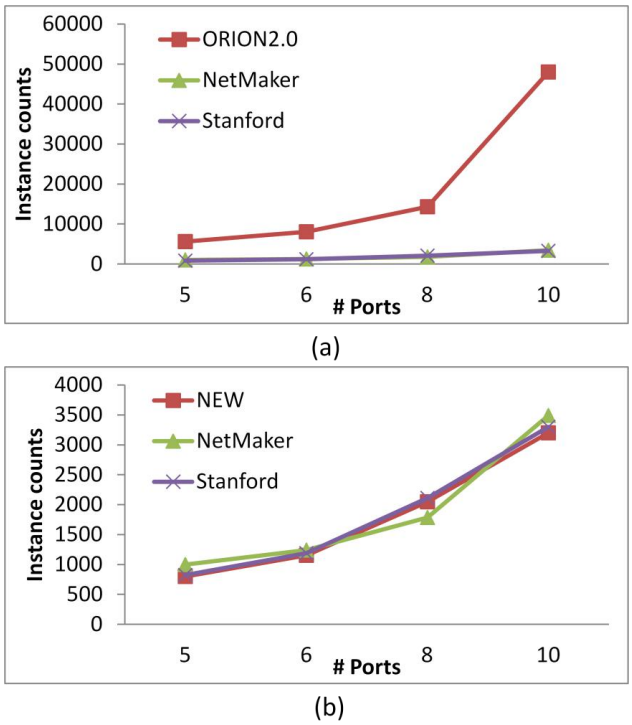


Figure 3.8: (a) XBAR with #ports: ORION2.0 vs. implementation. (b) XBAR with #ports: ORION_NEW vs. implementation.

ORION2.0 modeling of instance counts for a component does not consider implementation parameters such as clock frequency. As a result, the instance counts do not scale when frequency is changed, even though at higher frequencies buffers are typically inserted to meet tight setup time constraints. Our ORION_NEW models apply a frequency derating factor to the instance models for component blocks, and hence achieve higher accuracy. Figures 3.11(a) and (b) respectively show results for output and input buffer component blocks; the incorrect estimates by ORION2.0 contrast sharply with the estimates from ORION_NEW, which are very close to actual implementation.

Table 3.5 summarizes ORION2.0 and ORION_NEW estimation errors with respect to *Netmaker* and *Stanford NoC* post-P&R area. Higher error values are highlighted in red. Figures 3.12(a) and (b) plot the estimation errors for power and area respectively in 45nm and 65nm

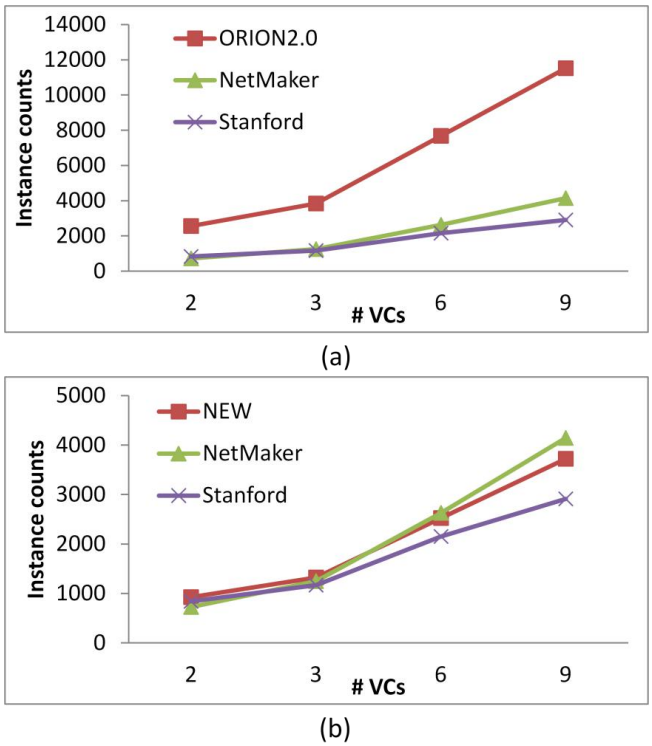


Figure 3.9: (a) Output buffer with #VCs: ORION2.0 vs. implementation. (b) Output buffer with #VCs: ORION_NEW vs. implementation.

Table 3.5: Instance counts and area error comparison of ORION2.0 vs. ORION_NEW.

Component	Avg Error: #Instances		Max Error: #Instances		Avg Error: Total Area		Max Error: Total Area	
	2.0	NEW	2.0	NEW	2.0	NEW	2.0	NEW
XBAR	86.10%	2.10%	93.10%	3.00%	86.20%	0.90%	93.20%	1.80%
SWVC	12.30%	12.30%	35.40%	35.40%	15.90%	20.80%	39.10%	66.80%
InBUF	270.70%	8.00%	417.30%	19.30%	134.40%	6.50%	199.40%	20.20%
OutBUF	69.00%	13.60%	80.60%	27.80%	74.70%	24.80%	86.40%	60.10%
Overall	109.50%	8.80%	156.60%	21.40%	77.80%	13.30%	104.50%	37.20%

technology nodes after applying the regression fitting approach described in Section 3.1.1. We see that ORION_NEW estimates are very close to actual implementation (average error of 9.3% in estimating *Netmaker* power in 45nm) and are robust across multiple microarchitecture, implementation parameters, and router RTLs.

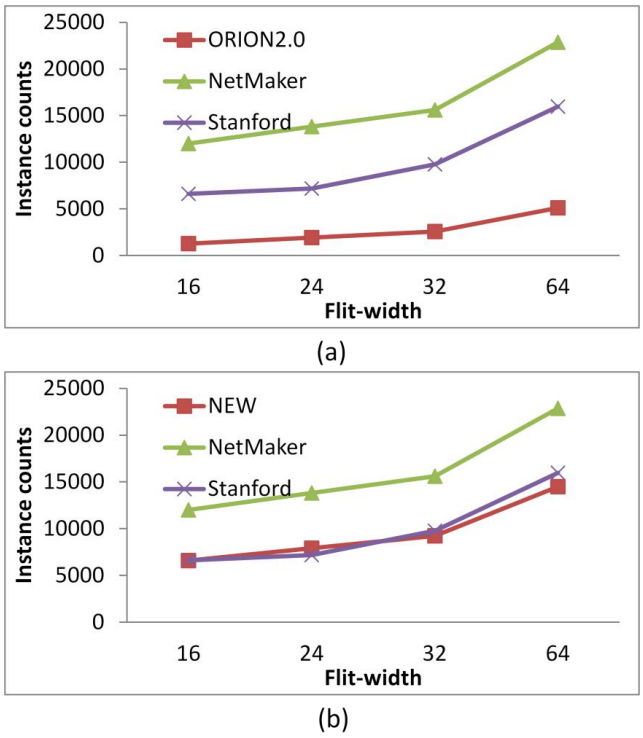


Figure 3.10: (a) Input buffer with flit-width: ORION2.0 vs. implementation. (b) Input buffer with flit-width: ORION_NEW vs. implementation.

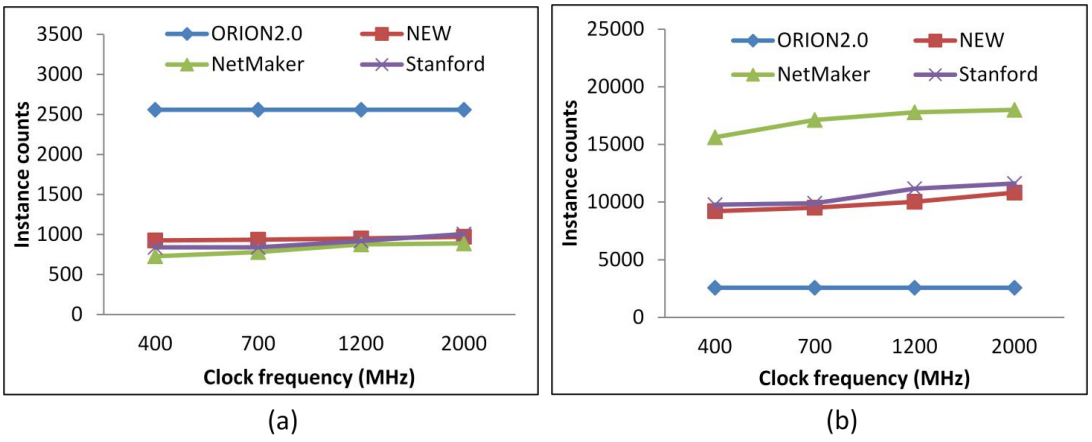


Figure 3.11: (a) Output buffer with clock frequency: ORION2.0 vs. ORION_NEW. (b) Input buffer with clock frequency: ORION2.0 vs. ORION_NEW.

Next, we analyze the impact of flit-level power modeling as described in Section 3.1.1. To capture the effect of running simulations with input vectors having different bit encodings (shown in Figure 3.5), we use options in Synopsys *PrimeTime-PX* [342] to vary toggle rates

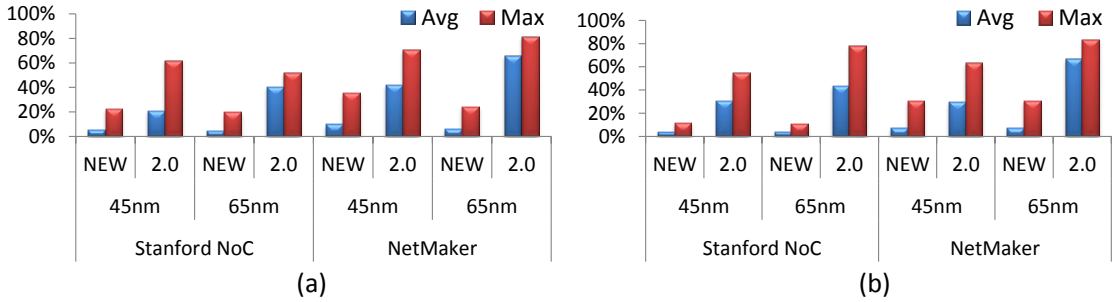


Figure 3.12: ORION_NEW with regression fit vs. ORION2.0: (a) area estimation error and (b) power estimation error.

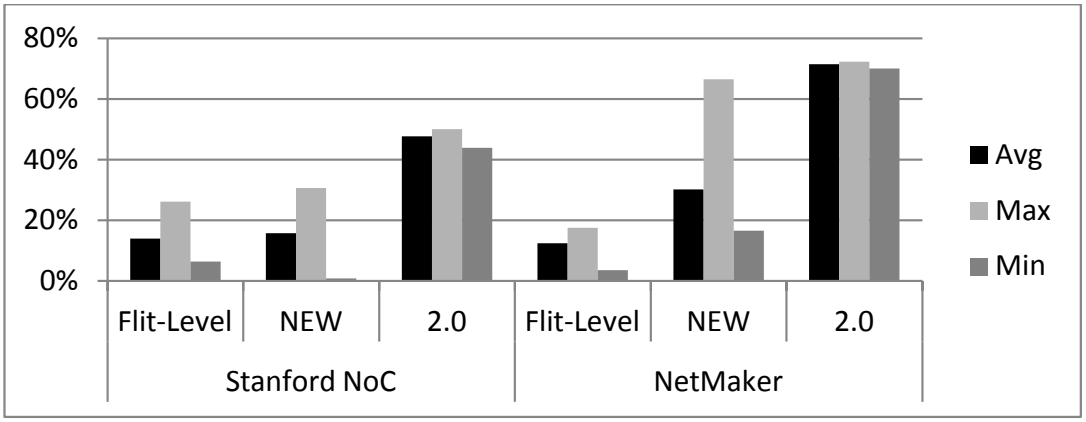


Figure 3.13: Comparison of dynamic power estimation error using (1) ORION2.0, (2) ORION_NEW, and (3) ORION_NEW with flit-level power models.

and bit encodings in the input. We run simulations using four different toggle rates (0.2, 0.4, 0.6, and 0.8) and four different encodings of 1's in 32-bit input flits. We observe that leakage power is not dependent on bit encodings (changes by less than 2%). However, dynamic power varies by up to 30% (on average) depending on bit encodings in each flit. ORION2.0 models are incomplete because they consider only the flit arrival rates in the dynamic power estimation models. Figure 3.13 compares dynamic power estimation error of ORION2.0, ORION_NEW, and ORION_NEW with flit-level power models. We observe that by using flit-level power models, average dynamic power estimation error can be within 12%.

Impact of Our Regression Analysis Approach

In Section 3.1.1, we describe our parametric regression analysis approach using the ORION_NEW models. As seen from the above results, the ORION_NEW models are accurate

across microarchitecture, implementation, and operational parameters. This demonstrates that regression analysis can minimize errors and generate accurate fitting coefficients. The previous parametric regression approach [116] reports large errors because underlying ORION2.0 models do not model control path elements. The non-parametric regression approach of [116] using MARS achieves reduced average power modeling errors of 5.82% in 65nm and 5.65% in 90nm, and reduced average area errors of 5.41% in 65nm and 5.01% in 90nm. In our work, we use parametric regression analysis but with accurate ORION_NEW models. Our average errors are slightly higher compared to [116]; however, our maximum error for power (resp. area) is reduced from 59.41% to 24.42% (resp. from 61.84% to 30.30%) in 65nm. In 90nm the reduction of maximum power (resp. area) error is from 60.15% to 28.04% (resp. from 60.07% to 19.36%). The reduction of maximum estimation error is significant because NoC designers and architects care about worst-case accuracy.

Results of Metamodeling Techniques

We set up experiments for each of the metamodeling techniques using the parameters, tools, and methodology described in Table 3.4. We generate 256 data points of post-P&R power and area values using 45nm and 65nm technology libraries. The input variables to all the models are *P, V, B* and *F* and the responses are post-P&R power and area. We use LHS to generate training sets of three sizes.

- 50 data points – sparse and restricted,
- 64 data points – sparse, and
- 102 data points – dense.

We use the sparse and restricted training set to test the accuracy of models in estimating area and power for input parameters which are beyond the range of values used for training. In each experiment, model generation takes around 3s and response estimation takes around 1.88s. We repeat all experiments 10 times for each training set size, and report the averages of all the error values across the 10 trials.

We present the results of metamodeling techniques as (i) comparisons among the techniques used, (ii) comparisons against MARS with linear splines [116], and (iii) comparisons against parametric regression techniques [119].

(i) Comparisons with metamodeling techniques used. Figures 3.14(a) and (b) show the percentage errors observed in estimating standard-cell area in 65nm and 45nm. With a dense training set, RBF, KG and MARS have similar maximum estimation errors of around 20%. SVM, on the other hand, has maximum estimation errors of 37.8% in 45nm and 25% in 65nm. The average estimation errors of all these models are less than 10.7%, with SVM having higher average estimation error than RBF, KG and MARS. With a sparse training set (64 data points), RBF and KG have higher accuracies than MARS and SVM in 45nm. RBF always performs better than KG, MARS, and SVM with a sparse and restricted training set (50 data points). Figure 3.14(b) shows that with a sparse and restricted training set the maximum estimation error is less than 12.8% for RBF, whereas the maximum estimation errors are more than 32% for KG, MARS, and SVM. The accuracies of these models in estimating power are similar to their accuracies in estimating area, as shown in Figures 3.15(a) and (b). With a sparse and restricted training set, RBF can be three times more accurate than KG, SVM and MARS. RBF and KG have similar errors for sparse as well as dense training sets. Across all training set sizes used in our experiments, we observe that area and power estimation errors are the smallest for RBF, and are the highest for SVM.

(ii) Comparison of MARS linear and cubic splines. Prior work in [116] uses MARS with linear splines to model NoC area and power, and reports maximum estimation errors of around 60% in 65nm. We use MARS with cubic splines in our experiments. Figures 3.16(a) and (b) compare area and power in 65nm with our metamodeling techniques. In general, our maximum estimation errors are smaller than those of [116] across all the techniques. In particular, our estimation errors (maximum, and average) for MARS are smaller than in [116]; this is because cubic splines are better than linear splines in minimizing estimation errors. Figures 3.17(a) and (b) show that cubic splines perform better than linear splines across different technologies and training set sizes. With a sparse training set in 65nm, the maximum area (resp. power) estimation error is 24.8% (resp. 19.7%) with cubic splines, whereas it is 33.6% (resp. 28.3%) with linear splines.

(iii) Comparisons with parametric regression. We use a sparse training set of 64 data points to estimate area and power using the parametric LSQR technique used to fit parametric models of router component blocks in [119]. Figures 3.18(a) and (b) compare the maximum and average estimation errors of the metamodeling techniques with estimation errors of parametric models

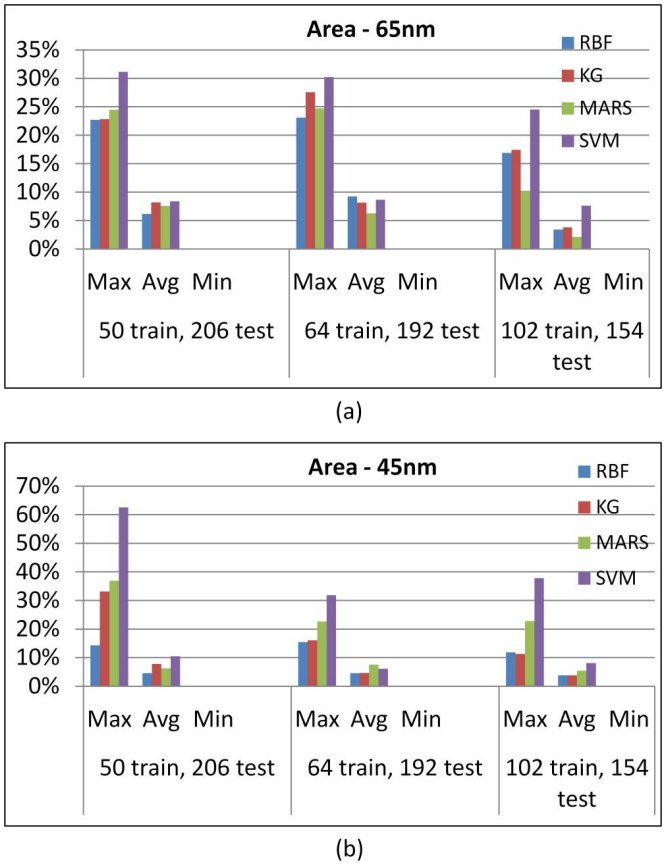


Figure 3.14: Area estimation accuracy of metamodeling techniques in (a) 65nm and (b) 45nm.

fitted using LSQR (NEW in the figures). We observe that the maximum area estimation error for NEW in 45nm is 22.8%, whereas the maximum area estimation error for RBF is 15.3%. The average area estimation errors for NEW are similar to the average area estimation errors for SVM and KG in both 65nm and 45nm, but can be up to 2.6% higher than the average area estimation errors for RBF in 45nm. In 45nm, the maximum power estimation error for NEW is smaller than the maximum power estimation errors of MARS and SVM. In 65nm, the maximum power estimation error for NEW is 2.5% higher than the maximum power estimation errors for RBF and KG. In general, we observe that the estimation errors for NEW and RBF are smaller than those for SVM. The average estimation error for NEW is less than 8%, and its maximum estimation error is less than 30%, in both 65nm and 45nm. RBF is more accurate than all other techniques in estimating NoC area and power in both 45nm and 65nm.

Figures 3.19(a) and (b) compare the maximum and average flit-level power estimation errors of metamodeling techniques with ORION_NEW flit-level power models fitted with post-

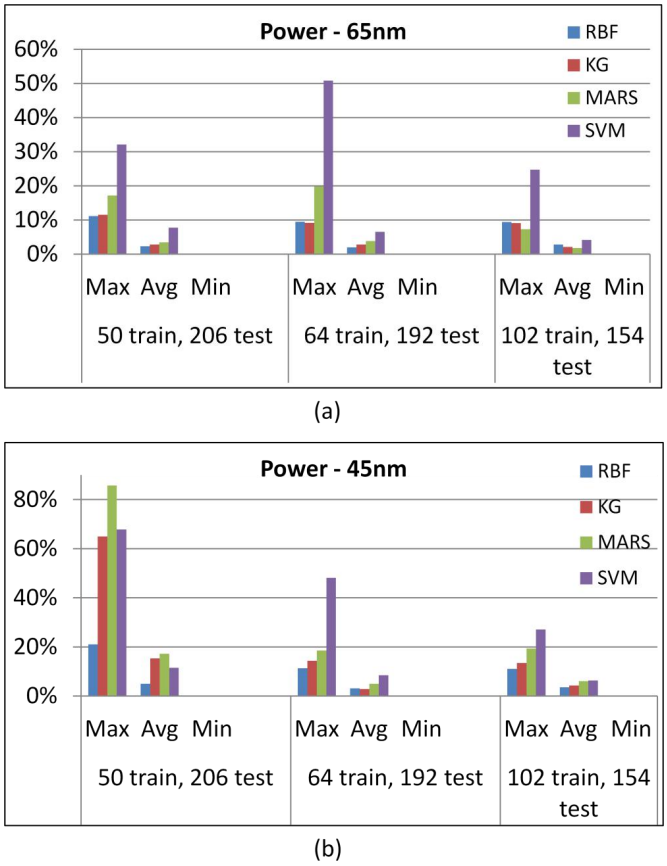


Figure 3.15: Power estimation accuracy of metamodeling techniques in (a) 65nm and (b) 45nm.

P&R dynamic power using parametric regression (NEW in the figures). We use a sparse and restricted training set of 50 data points for all the modeling techniques in this experiment. We observe that NEW and RBF are more accurate than the other metamodeling techniques. NEW is accurate because its fine-grained modeling of the component blocks in a router enables accurate estimation of dynamic power dissipation in each component block for different bit encodings in flits. KG, MARS and SVM cannot estimate flit-wise power dissipation of each component block because they fit the training data points by treating the router as a black box. Therefore, estimation errors are large when fitting data points with different flit-level bit encodings. We observe that both maximum and average estimation errors for SVM, KG, and MARS are significantly higher (>60%) than those for NEW (21.6%) and RBF (20.1%). RBF and NEW also achieve smaller average estimation errors in both 65nm and 45nm as compared to the average estimation errors of KG, MARS and SVM.

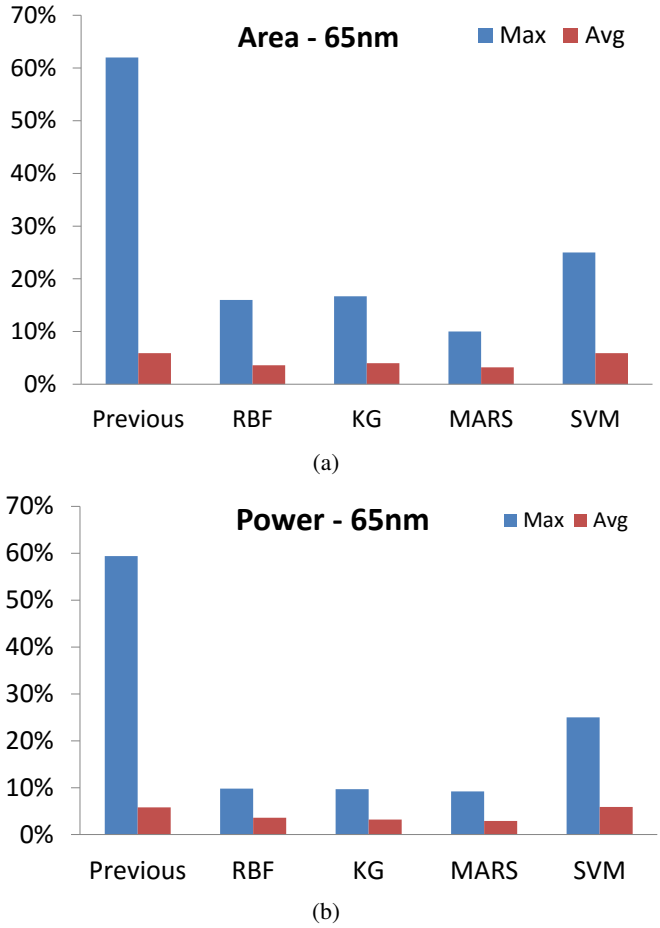


Figure 3.16: Comparison with estimation errors of "Previous" [116] in 65nm: (a) area and (b) power.

3.1.4 Conclusions

Accurate modeling for NoC area and power estimation is critical to successful early design-space exploration in the era of many-core computing. ORION2.0, while very popular, has large errors versus actual implementation. This is because there is often a mismatch between the actual router RTL and the assumed templates. Also, typical design flows involve sophisticated optimizations that are difficult to characterize. In this work, we propose comprehensive parametric and non-parametric modeling techniques to accurately estimate NoC power and area. Our parametric models, ORION_NEW, explicitly account for control and data path resources. We propose a new methodology that we used to develop the ORION_NEW parametric models from post-synthesis netlists. We further refine these parametric models by performing least-squares regression analysis (LSQR) on post-P&R data. We demonstrate that accurate parametric

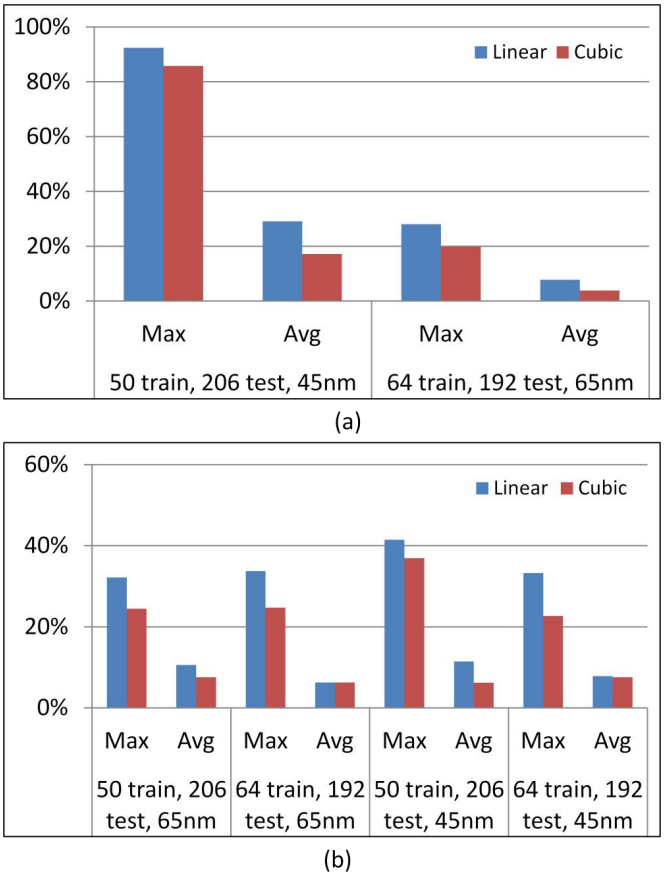


Figure 3.17: MARS linear vs. cubic splines: (a) power and (b) area.

models and LSQR can reduce the worst-case estimation errors by more than 50% as compared to previous non-parametric regression models for NoC routers [116]. We are also the first to propose detailed flit-level power estimation models that can seamlessly integrate with full-system NoC simulators such as *GARNET*.

For non-parametric regression (or metamodeling), we use four popular techniques – RBF, KG, MARS, and SVM. Our results show that these techniques can be low-overhead and highly accurate in estimating NoC power and area. We describe two methodologies to generate training sets to test the accuracy and robustness of these techniques. Among these four techniques, RBF proved to be the most accurate and robust across technologies and training set sizes. However, these techniques are not accurate for detailed flit-level power modeling because they cannot model flit-level power dissipation in each component block of the router. The ORION_NEW model fitted with post-P&R dynamic power using parametric regression provides more accurate estimates of flit-level dynamic power compared to KG, MARS and SVM.

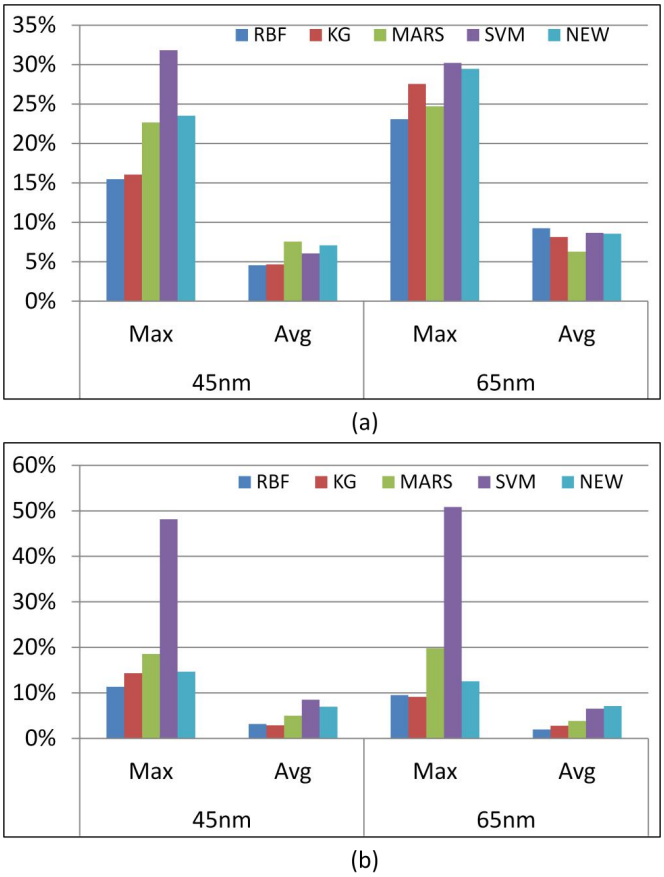


Figure 3.18: Comparison of estimation errors of non-parametric vs. parametric regression techniques: (a) area and (b) power.

We validate robustness of our modeling methodologies across multiple router RTLs, and across microarchitecture, implementation, and operational parameters. We conclude that our modeling methodologies are highly accurate with average errors $\leq$9.3%. Implementations of our modeling methodologies are being made available for download in an *ORION3.0* distribution [315]. To date, there have been over 760 downloads of the *ORION3.0* distribution.

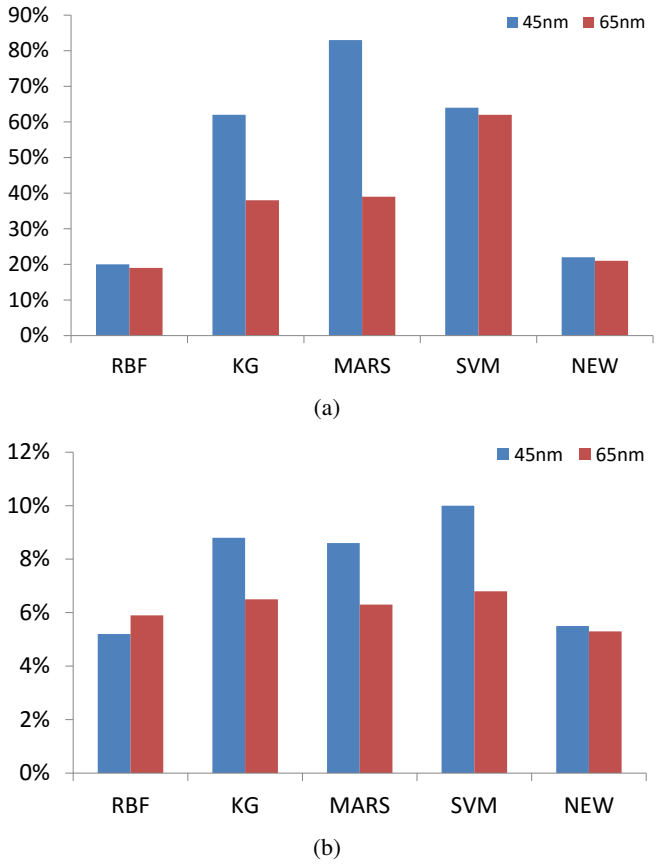


Figure 3.19: Comparison of estimation errors in flit-level power modeling in 65nm and 45nm: (a) maximum and (b) average.

3.2 Learning-Based Prediction of Embedded Memory Timing Failures during Initial Floorplan Design

Timing closure in modern SoCs is complex and time-consuming, due to multiple iterations between various analyses and design fixes. Early, accurate prediction of post-layout slack can potentially deliver dramatic design turnaround time and design cost reductions. However, to the best of our knowledge, no existing tool can predict slack at an early design stage (in particular, the post-synthesis, physical floorplanning stage).[8] Predicting post-layout slack without physical synthesis or trial placement information is challenging because wire delay must be estimated without spatial embedding information. The prediction problem becomes even more difficult because of (i) embedded memories, and (ii) "multiphysics" analysis.

Embedded memories (SRAMs) complicate SoC physical implementation on several fronts [81] [65]. They occupy significant die area [117] and are typically placed in arrays, which not only makes floorplanning difficult, but also creates placement and routing blockages. Timing analysis and closure are costly, e.g., with respect to cross-corners in low-power, split-rail designs. Hence, despite long tool runs and complex design subflows, SoCs with multiple SRAMs can have unpredictable timing at the post-P&R stage, not to mention in silicon.

Verification of timing correctness in advanced nodes increasingly demands analyses that close the loop across crosstalk, IR and temperature [185] [276], i.e., more than one "physics". We use *multiphysics analysis* to mean performing multiple analyses such as IR, thermal, reliability, crosstalk, etc., and then performing static timing analysis (STA) using reports from these analyses. Timing assessments can vary widely with the specific analyses performed, e.g., turning SI mode on can worsen slack by 100ps due to crosstalk [128]. Figure 3.20(a) shows slack values of five memories in a small block[9], according to four different analyses that combine IR analysis and STA: (i) STA with no IR analysis; (ii) STA with static IR analysis; (iii) STA with dynamic voltage drop (DVD) IR analysis; and (iv) four iterations of STA with DVD IR analysis, i.e., STA is performed with back-annotated instance-specific DVD IR drop, going around this loop four times. Figure 3.20(b) shows that across different implementations of the same netlist (i.e., when clock period and maximum transition time constraints are varied), the slack difference between (i) STA with no IR, and (ii) two iterations of STA with DVD IR, can vary by $\sim$15ps depending

[8] A long history of RTL signoff and RTL planning tools is best exemplified today by Synopsys SpyGlass [345], which performs early analysis of designs but uses its own simplified placement, clock tree synthesis and routing engines that do not necessarily match production back-end tool outcomes.

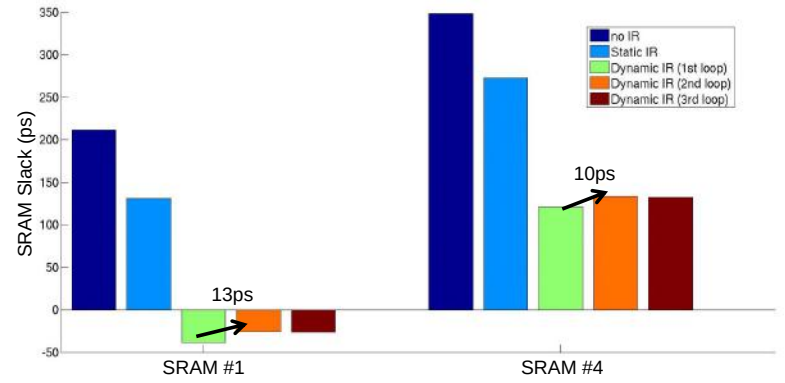
[9] We place only one power/ground pad pair at the south edge for this testcase (OpenCore *THEIA*) to emulate a severe voltage-drop situation. The signoff clock period for this example is 3.5ns.

on the implementations. By closing multiphysics analysis loops, design teams achieve more accurate timing results, but the results of such analyses are non-trivial to predict in early stages of implementation.

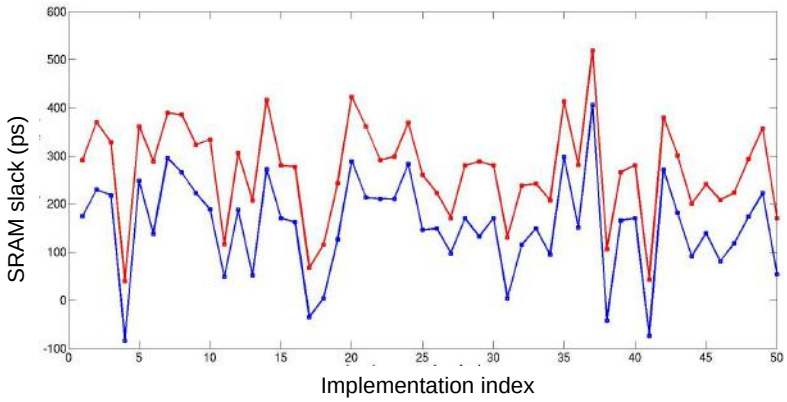
We show two examples to illustrate the challenges of predicting post-layout slack. **(1) Sensitivity of slack to spacing between memories.** Figure 3.21(a) shows a floorplan with five embedded SRAMs, blockages, and a rectilinear standard-cell placement region. Figure 3.21(b) shows variation of worst timing slack (at any timing endpoint in a given SRAM) across these five SRAMs when the spacing (i.e., channel width) between memories is varied in steps of 10μm. Due to congestion, buffer placement, etc., the difference in slack can be larger than 300ps at a spacing of 10μm, and slacks vary in a highly non-obvious and/or noisy manner as the spacing is changed. **(2) Sensitivity of IR drop map to power pad locations.** Figures 3.22(a)–(c) show three IR drop maps when the locations of power pads are varied. When power pads are placed uniformly on all edges of the die as in Figure 3.22(a), the IR map has very few hotspots. When the power pads are placed only on the left and right edges of the die as in Figure 3.22(b), or on the bottom and top edges as in Figure 3.22(c), the IR drop map has multiple hotspots. The IR drop map, and timing slacks, have similarly challenging sensitivities to SRAM placement relative to power distribution network stripes (PDN stripes), the availability of buffer placement locations within or near memory channels, etc.

In this work, we apply machine learning to achieve accurate predictive modeling of slacks at embedded SRAM timing endpoints. Given *only* a post-synthesis netlist, constraints and a floorplan, we predict (i) post-P&R slack, and (ii) slack with multiphysics analysis, of SRAMs. As reviewed in Section 2.1.2, previous works do not address slack prediction of SRAMs at the post-routing stage, nor at the multiphysics signoff stage.

Figure 3.23 shows the stages of physical implementation that we must comprehend with our modeling, as well as the stages from which we can extract available modeling parameters. We estimate the combined effects of placement, clock network synthesis, routing, extraction and timing using our modeling function f as shown in the figure. Our work envisages two basic use scenarios. (1) For products in the early planning stage, our predicted slacks enable floorplans and constraints – as well as physical implementation methodology – to be adjusted to prevent post-layout timing failures on SRAMs. (2) For products in the production stage, our model enables designers to filter out floorplans and constraints that have high risk of post-layout timing failures under voltage and frequency scaling, or process variation. Our model can prevent costly iterations of floorplan and constraint adjustments.



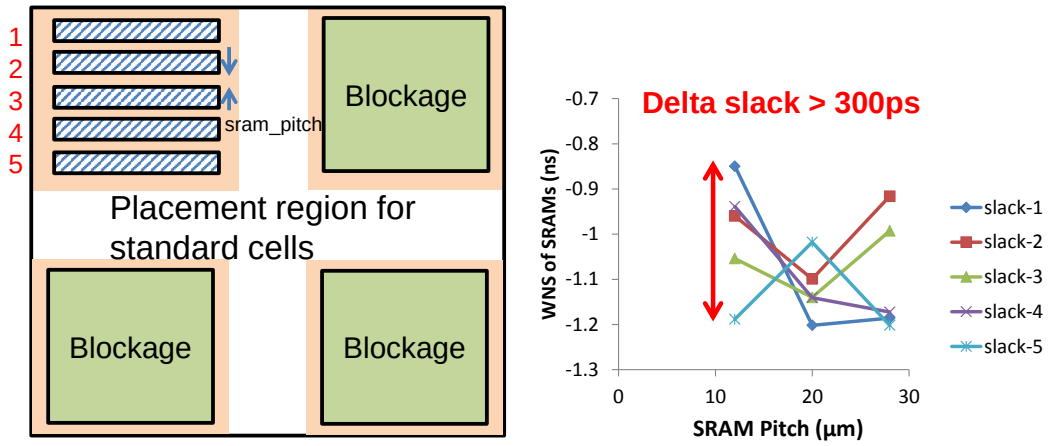
(a)



(b)

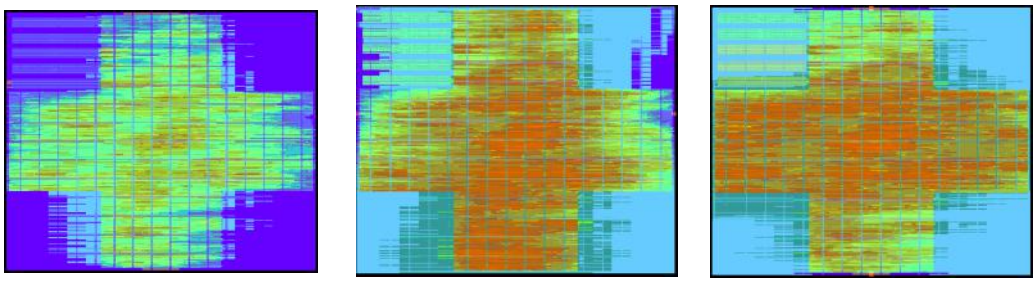
Figure 3.20: Multiphysics analysis. (a) SRAM slack with (i) no IR, (ii) static IR, (iii) dynamic voltage drop (DVD) IR, and (iv) four iterations of DVD IR and STA. (b) Difference in slack between (i) no IR and (ii) STA + DVD IR with two loops. The indices in the x-axis of (b) refer to different implementations when clock period and maximum transition time constraints are varied.

In this work, we also advance application of machine learning for predictive IC design with a new implementation of the Boosting technique. Previous works such as Kotisantis et al. [145] create an ensemble of regressors by using Bagging [85], Boosting with SVM without a kernel, and Random Forests [85]. The authors of [145] then combine outcomes of each regressor using weights that are proportional to the inverse of the error of the outcomes from each regressor. In their method the weights are calculated based on error in the test set, rather than on error in the training set. The work of Ogutu et al. [186] compares Random Forests, Boosting with a linear regressor, and SVM to predict genomic breeding values. The authors of [186] conclude that SVM is more accurate than the other two techniques. Our implementation of Boosting builds on [72] [145] by using SVM with a nonlinear kernel.



(a) (b)

Figure 3.21: Sensitivity of slack to spacing between SRAMs: (a) floorplan and (b) slack variation.



(a) (b) (c)

Figure 3.22: Sensitivity of IR drop map to power pad placement: (a) on all four edges, (b) left and right edges only, and (c) bottom and top edges of the layout.

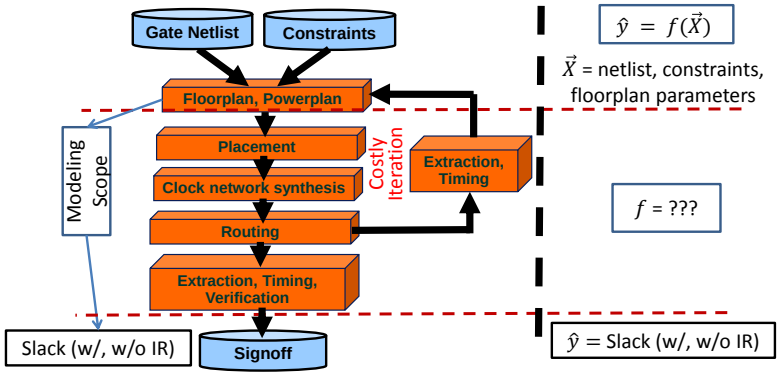


Figure 3.23: Traditional IC design flow from [118], with dotted horizontal lines bounding the scope of our model. We comprehend multiple stages of the physical design flow in our model, which is represented by the function f in the figure.

The main contributions of this work are summarized as follows.

- To the best of our knowledge, we are the first to propose a modeling methodology that can effectively predict post-P&R slack values at endpoints on embedded SRAMs, using information available at the floorplanning stage. Our model applies machine learning techniques to predict post-P&R slack within a worst-case error of 224ps and average error of 4.0ps across all designs and floorplans tested in a 28nm foundry FDSOI technology.

- We confirm the robustness of our prediction methodology by predicting slack values *after* multiphysics analysis – a very difficult prediction problem – to within a worst-case error of 253ps and average error of 9.0ps. A model that uses information from the post-synthesis netlist results in multiphysics slack worst-case prediction error of 358ps.[10]

- We automate using commercial EDA tools the extraction of relevant model parameters, and prediction of timing failure risks, from given netlist, constraints and floorplan context. By predicting multiphysics slack for every embedded memory endpoint, our model enables early filtering and improvement of floorplans that would otherwise eventually lead to timing failures at the post-layout and signoff stages.

- We advance application of machine learning for predictive IC design with a new implementation of the Boosting technique that uses Support Vector Machines (SVMs) as *weak learners*. We also propose a weighting strategy for negative-slack outcomes during our model construction, to accurately focus our model on avoidance of critical timing failures. SVM in Boosting reduces worst-case prediction errors by 30ps relative to use of SVM only.

3.2.1 Methodology

We now describe the key elements of our work: multiphysics analysis flow, model parameter selection, and machine learning-based modeling methodology. We also note how our analyses and modeling flows would be reproduced in a new environment.

[10] We use parameters $N1$ through $N6$ from Table 3.6 and use three different modeling techniques – LASSO, linear SVM and SVM with a Radial Basis Function (RBF) as kernel – to predict multiphysics slack. The worst-case (resp. average) prediction errors are 565ps (resp. 87ps) for LASSO, 412ps (resp. 55ps) for linear SVM method, and 358ps (resp. 42ps) for SVM with RBF kernel. The SVM model with RBF kernel has smaller prediction errors as compared to those of linear SVM and LASSO models. Therefore, we compare our results with those from the SVM model with RBF kernel.

Multiphysics STA

Figure 3.24 shows our multiphysics analysis flow. Due to the very large number of testcase implementations, we focus on an IR-STA multiphysics analysis loop.[11] We perform STA using Synopsys *PrimeTime-SI* (PTSI) [342]. The inputs to the tool are Liberty timing libraries characterized at multiple voltage corners, Verilog netlist of the design, SPEF parasitics [333] with coupling capacitances, and Synopsys Design Constraints (SDC) [21] with timing constraints as well as back-annotated rail voltages of all instances based on the IR drop map. Note that STA is always performed with SI enabled in our flows.

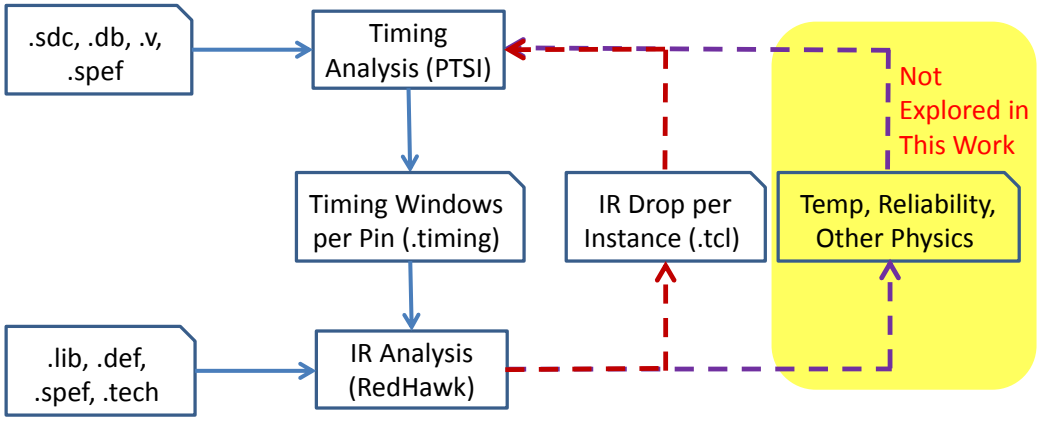


Figure 3.24: Our multiphysics analysis flow.

IR (voltage drop) analysis is the first dimension of multiphysics analysis that must be joined with timing analysis. To assess the vulnerability of various floorplans to timing failures from IR drops, we parameterize the stripe width and pitch of on-chip PDNs, the width of power rings, the metal layers that are used for the PDN stripes, and the placement of memories relative to the nearest power pad. To supply power to memories, we generate secondary meshes on a metal layer that is different from the ones used for PDN stripes. For standard cells, we use M2 metal layers to connect PDN stripes. We connect power meshes and stripes in the lower metal layers to the upper metal layers (M9 and M10) through via stacks, as in normal SoC methodology. We perform vectorless dynamic voltage drop (DVD) analysis using ANSYS *RedHawk* [325]; inputs consist of the post-layout design database, technology Layout Exchange Format (LEF) [307] files, Liberty timing libraries, and the minimum and maximum, rise and fall arrival timing windows of all signal pins as reported from *PrimeTime-SI* [342]. Our DVD IR analysis

[11] This can be extended to more complete multiphysics analyses that include temperature and reliability - e.g., using ANSYS Sentinel-TI and RedHawk-SEM. While we have prototyped such analyses, they are cumbersome with available tools, and we do not report any studies here.

is vectorless due to lack of representative simulation vectors; to our understanding, this reflects common industry practice. We place power pads uniformly along the block periphery so that the IR drop tends to be worst at the block center.

Once we have obtained an IR drop map, we back-annotate individual cell instances with rail voltage in *PrimeTime-SI*, and perform STA using timing libraries that have been characterized at multiple voltage and temperature corners using Synopsys *SiliconSmart* [344]. Again, we view this STA as "multiphysics" in its integration of the IR drop map on a per-instance basis. The standard practice in industry is to perform the above-described multiphysics analysis once. But, recall from Figure 3.20(a) above that applying more than one iteration can help remove pessimism in timing analysis by up to 15ps.

Model Parameter Selection

We use model parameters that span netlist structure, floorplan context and layout constraints. The modeling problem is high-dimensional when we consider multiple knobs in commercial tools, as well as multiple netlist and layout context parameters. To make our modeling methodology practically applicable, we focus on only those parameters that we have so far found to affect modeling accuracy. We assess the sensitivity of slack to each parameter independently by varying values of one parameter at a time and keeping the remaining parameters the same. We also assess the combined impact of various parameters on the slack of memories using *variance inflation factor* (VIF) [5]. We choose parameters whose VIF values are less than 0.5 [122] and let the modeling techniques (described in Section 3.2.1) combine relevant parameters. Some of our parameters are for the entire layout, whereas the remaining parameters are for each memory instance so that the modeling can capture variable number of memories in the netlist, variations of floorplans, and the placement of memories within these floorplans. Table 3.6 lists our modeling parameters. The first column gives the parameter identifier; the second column describes the parameter; the third column shows whether the parameter is of type netlist, floorplan or constraint; and the last column indicates that the parameter is obtained per memory instance when it is "Yes". Some of our modeling parameters are based on guidance provided in [6], [52] and [122].

Modeling Techniques

Recall from Figure 3.23 that we seek to model (i) multiple stages of the physical design flow such as placement, clock network synthesis, routing, extraction, etc., (ii) inherent noise in

Table 3.6: Parameters used in our modeling.

Parameter	Description	Type	Per-memory?
N1	Max delay across all timing paths at the post-synthesis stage	Netlist	Yes
N2	Area of cells in the intersection of startpoint fanout and endpoint fanin cones of max-delay incident path	Netlist	Yes
N3	Number of stages in the max-delay incident path	Netlist	Yes
N4, N5, N6	Max, min and average product of #transitive fanin and #transitive fanout endpoints	Netlist	Yes
N7	Width and height of memory	Netlist	Yes
FP1	Aspect ratio of floorplan	Floorplan	No
FP2	Standard cell utilization	Floorplan	No
FP3, FP4	PDN stripe width and pitch	Floorplan	No
FP5	Size of buffer screen around memories	Floorplan	No
FP6	Area of blockage (%) relative to floorplan area	Floorplan	No
FP7, FP8	Lower-left placement coordinates of memories	Floorplan	Yes
FP9, FP10	Width, height of channels for memories	Floorplan	Yes
FP11	#memory pins per channel	Floorplan	Yes
C1	Sum of width and spacing of top-three routing layers after applying non-default rules (NDRs)	Constraint	No
C2	% cells that are LVT	Constraint	No
C3, C4	Max fanout of any instance in data and clock paths	Constraint	No
C5, C6	Max transition time of any instance in data and clock paths	Constraint	No
C7	Delay of the largest buffer expressed as $FO4$ delay	Constraint	No
C8	Clock period used for P&R expressed as $FO4$ delay	Constraint	No
C9	Ratio of clock periods used during synthesis and P&R	Constraint	No

commercial tools, and (iii) a very high-dimensional space of parameters that span across netlists, floorplan contexts and timing constraints. Interactions between parameters are complex, e.g., an increase in PDN stripe density can cause a large congestion on upper metal layers and thereby increase coupling capacitances which will ultimately result in timing failures even when the IR drop is small. The type of timing analysis can contribute to large difference in slack, e.g., turning SI mode on can worsen slack by 100ps or more [128].

We use both linear as well as nonlinear machine learning techniques. We use LASSO regression with L1 regularization [192] as a linear technique, and Support Vector Machine (SVM) regression [85] with a Radial Basis Function (RBF) kernel [85], Artificial Neural Networks (ANN) [85], and Boosting [72] with a weak SVM learner as the nonlinear techniques. The Boosting learning technique combines predictions of multiple weak learning techniques to create an accurate learning model. Learning techniques such as linear classification and regression trees are used commonly as weak learners. For a comprehensive discussion on LASSO, SVM, ANN and Boosting, see [85]. For each technique, we use training and validation data sets that are 50% and 10% of the total data points, respectively, and we search for values of hyperparameters using grid search such that the training and validation mean-square errors (MSEs) are comparable. For SVM with RBF kernel, the hyperparameters are ε along with the cost C that

control the margin errors of the support vectors, and the width parameter γ of the RBF kernel. For ANN, we define the architecture as one input and one output layer and two hidden layers. The hyperparameters are the number of epochs for back propagation and the number of neurons per hidden layer. For LASSO, the hyperparameter is the regularization coefficient λ. For Boosting with SVM, the hyperparameters are ε, C, γ, and the number of iterations.

For each machine learning technique, we perform five-fold cross-validation so as to make the models generalizable. We normalize the parameters to within $[0,1]$ before we proceed with modeling.[12] The nonlinear techniques (SVM, ANN and Boosting) help to capture complex interactions between parameters. Our preliminary studies indicate that SVM with a RBF kernel method achieves higher accuracy (less than 300ps worst-case error) than linear SVM without a kernel (more than 335ps worst-case error). Therefore, we use SVM with a kernel method. The LASSO technique has large modeling error (greater than or equal to 300ps) when the number of parameters is larger than five. Therefore, we use the linear LASSO technique to make predictions with a simplified model and use the outcomes of this linear technique as the bias in the final step in which we combine outcomes of all the techniques using Hybrid Surrogate Modeling (HSM) [122] to obtain the final predicted slack of each memory instance. Even though the procedure to combine predictions from various linear and nonlinear techniques is not obvious, the HSM technique enables us to combine the predictions using appropriate weights and improve overall prediction accuracy. Figure 4.25 shows our modeling flow. We implement our modeling in *MATLAB vR2013a* [311] using default toolboxes for ANN and LASSO, the open-source *libsvm* [42] toolbox for SVM, and our own implementation of Boosting.

Figure 3.26 shows our high-level implementation of Boosting.[13] We implement Boosting with weak SVM learners as follows.

- Initially, we set the weight W_0 of all training data points to be uniform, i.e., $W_0 = 1/N_{tr}$, where N_{tr} is the number of training data points.
- We use SVM as a weak learner by restricting the grid search to only three different values of each hyperparameter.
- We calculate the error e_i for the i^{th} stage using the validation set and W_i values of data points are set for the subsequent $(i+1)^{st}$ stage as $exp(0.5\log(\frac{1-e_i}{e_i}))$ when the error in

[12] We have tried normalization using z-scores to within $[-1,1]$ as well. The predicted values change by less than 0.5%. Therefore, we use normalization to within $[0,1]$ in our experiments.

[13] Of possibly independent interest is that to our knowledge, Boosting with SVM as a weak learner (i.e., regressor) has not been tried before in the machine learning and VLSI CAD literatures. Our scripts are available at http://vlsicad.ucsd.edu/Riskmap/.

slack prediction is greater than or equal to 50% of the clock period, and are set to 1 otherwise.

- To make our predictions pessimistic on data points for which the actual slack is negative, we increase W_i by a factor of five when the predicted slack value for such a data point is positive.
- We terminate when worst-case error in the validation set is less than 20% of the clock period or when the number of iterations reaches $k = 40$.[14]
- We combine outcomes of each iteration using coefficients β_i (determined by using least-squares regression), where $i = 1, ..., k$, to determine the final outcome of Boosting.

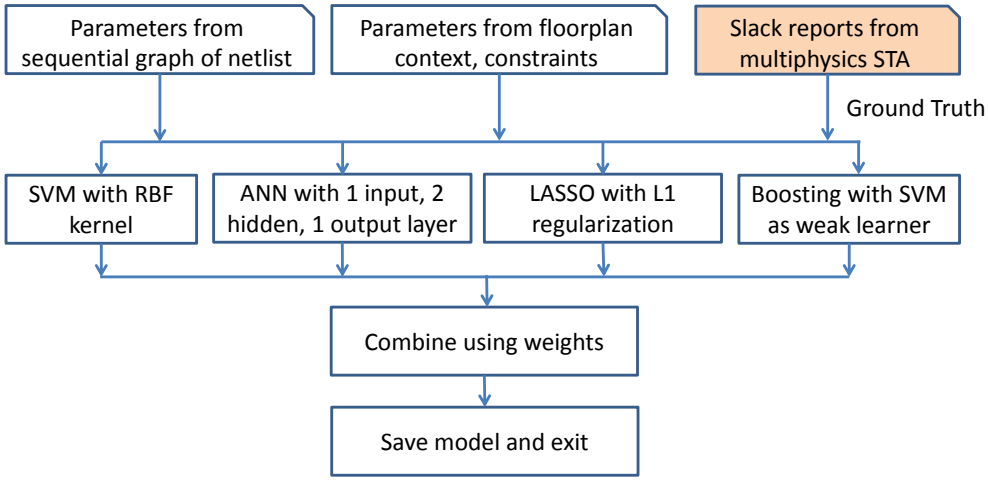


Figure 3.25: Modeling flow with linear and nonlinear regression techniques.

From the above discussion, the reader will note that the proposed methodology will in practice use post-P&R databases of various projects taped-out in a given (technology node, library, tool flow) as the basis of netlist structural analyses, floorplan structural analyses, and multiphysics performance analyses. Designers of new projects in the same technology node would use our model to filter out floorplans and constraints that can cause timing failures. In a new technology or design environment, the one-time initial model fitting effort that we describe above must be performed.

[14] With values of $k > 40$ we do not observe significant improvement in error.

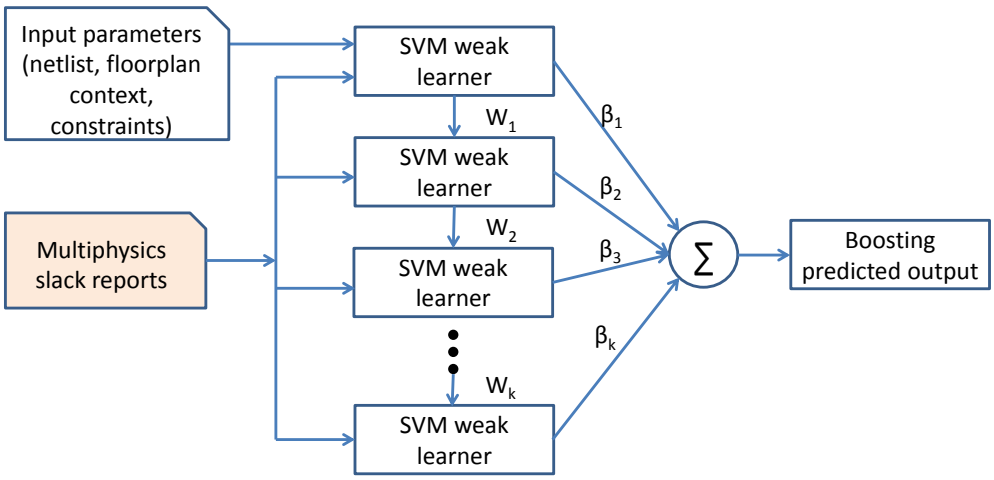


Figure 3.26: Flow with Boosting [72] with weak SVM learners.

3.2.2 Experimental Setup and Results

In this section, we describe our testcases and design of experiments, and present our modeling results.

Testcases

We have developed a generator to create testcases to vary (i) the number of SRAMs in the netlist,[15] (ii) the floorplan context such as aspect ratio, utilization, buffer screens, PDN structure, etc., and (iii) the placement of SRAMs in the floorplan. Figure 3.53 illustrates various parameterizations of floorplans in our testcase generator, using the floorplans shown in Figures 3.29(a) and 3.31(a). We can independently change the width and height of buffer screens around SRAMs or blockages, the dimensions of each blockage, and the area for standard cell placement.

Our netlists contain both logic and SRAMs. For logic, we use open-source designs such as *THEIA*[16] and *nova* from OpenCores [318], our own artificial testcases with an embedded processor, and blocks from OpenCores such as *aes_cipher_top* and *reed_solomon_codec*. We perform synthesis using 28nm foundry FDSOI libraries and Synopsys *Design Compiler vI-2013.12-SP3*. Table 3.7 summarizes our netlists with post-synthesis metrics.

Figure 3.28 illustrates a PDN structure used in our testcases. We use metal layers M9 and M10 for the power ring around the core; we also use M9 and M10 for the top-level power

[15] We use single-port SRAMs of two different sizes from the 28nm FDSOI foundry libraries.

[16] We have used the original OpenCores *THEIA* design as well as modified versions of the design. In the modified designs (*THEIA_v1*, ..., *THEIA_v4*), we vary the number of SRAMs. The unmodified design is *THEIA_v0*.

Table 3.7: Description of our netlists.

Netlist	Clock Period (ns)	#Std Cells	#SRAMs	Logic Area (μm^2)	SRAM Area (μm^2)
THEIA_v0	3.0	147274	40	157416	347252
THEIA_v1	2.7	146505	5	157068	40027
THEIA_v2	3.0	146914	6	157012	48032
THEIA_v3	3.0	146243	8	156212	64043
THEIA_v4	3.0	146606	10	155991	80054
nova	2.0	66031	5	68970	25117
artificial	2.0	201015	6	213075	14925

mesh. We use M6 to generate secondary meshes to supply power to SRAMs, and M2 to connect standard cells to the VDD and ground rails. From the post-P&R databases we generate the routed Design Exchange Format (DEF) [307] file using Synopsys *IC Compiler vH-2013.03-SP3* and provide it as an input to ANSYS/Apache *RedHawk v10.1.7* along with technology LEF and Liberty timing libraries. We use Synopsys *PrimeTime-SI vH-2013.06-SP2* to obtain timing windows of all signal pins.

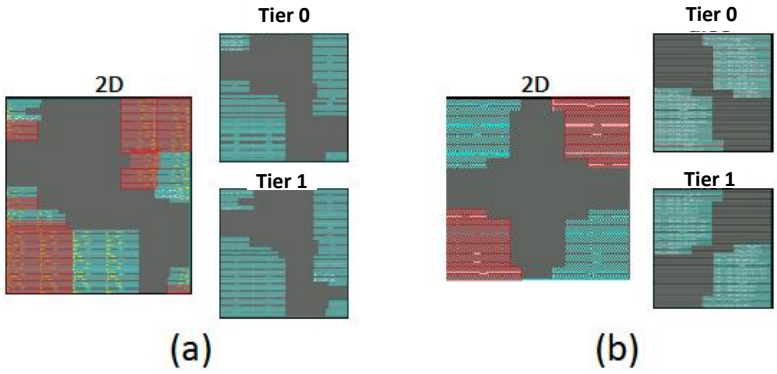


Figure 3.27: Parameterized floorplan used to generate testcase instances.

Design of Experiments

Using our generator described in Section 3.2.2 and netlists in Table 3.7, we create various testcases in which we vary floorplans, PDN structures and constraints. Table 3.8 lists the parameters we vary in our design of experiments. We vary the standard-cell placement region to have cross-, L- and T-shapes as shown in Figures 3.29(a)–(c). Each of these region shapes changes P&R tool outcomes since it changes the degree of nonconvexity (i.e., the number of non-

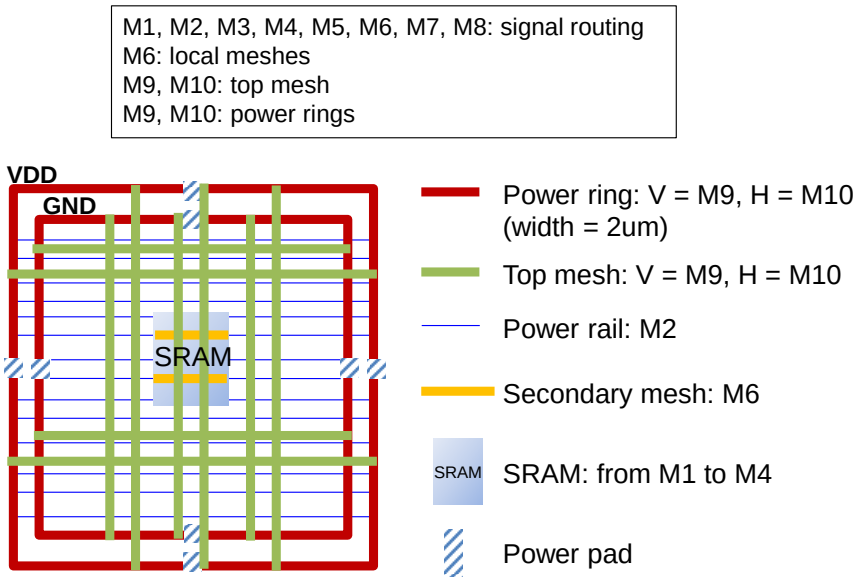
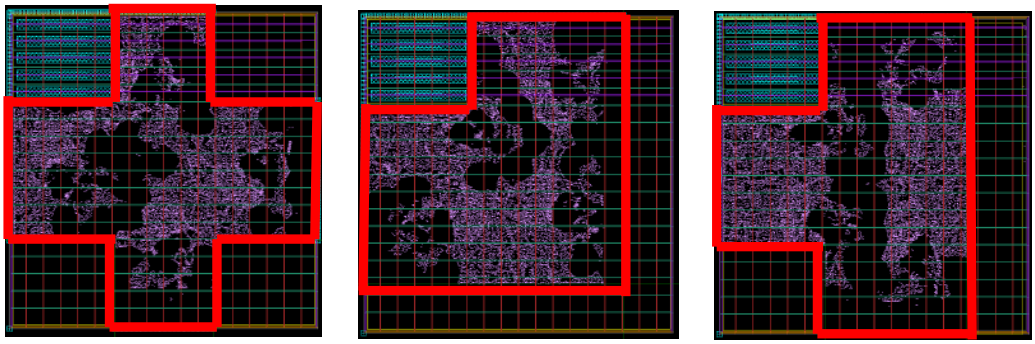


Figure 3.28: Example of PDN structure in our testcase with SRAMs.

convex corners) in the placement region, as well as the placement of IO pins. For example, the cross-shaped floorplan has more nonconvex corners and is expected to have higher congestion near these corners as compared to the L-shaped floorplan.

To emulate real designs, we frame our experiments in the context of a general, "tic-tac-toe" floorplan. We divide the block with two shiftable gridlines in each axis; each of the nine resulting gridcells can be fully or partially occupied by essential components of a floorplan, that is, hard macros, standard cells or blockages. The tic-tac-toe implementation (i) enables generality and parameterizability, (ii) enables the ability to explore a discrete design space systematically, and (iii) captures how designers tend to floorplan their blocks. Figure 3.30 shows an example instance of a tic-tac-toe floorplan. (Note that the tic-tac-toe framework allows us to explore floorplans either at the die-level or block-level, but not in between.)

We create multiple variations of floorplans for netlists with five, six, eight, 10 and 40 memories. Figures 3.31(a)–(f) show examples of six variations that we generate. All of these floorplans can be created with the tic-tac-toe implementation. Specifically, we create the floorplans with eight, 10 and 40 memories using this implementation. For the tic-tac-toe implementation, we focus on testcases with more than eight memories. Note that our modeling parameters listed in Table 3.6 can handle variations of floorplans shown in Figures 3.29 and 3.31 because we include parameters that are floorplan-specific as well as memory instance-specific. We allow only buffer instances to be placed within the buffer screens around SRAMs.



(a) (b) (c)

Figure 3.29: Variations in floorplans in our testcases: (a) cross-shape, (b) L-shape, and (c) T-shape. The red lines are used to highlight these shapes.

A real industry flow will run *Design Compiler* in topographical mode with floorplan constraints and generate a DEF with placement of standard cells. However, this requires a tool license which is not available to us. We denoise each P&R run by varying the parameters by $\pm 0.5\%$ from its value. We generate a total of 2515 data points for modeling, out of which we use 1248 (50%) data points for training, 226 (9%) data points for validation, and the remaining 1041 (41%) data points for testing. We challenge our modeling by testing on all data points of the design *nova* and values of aspect ratio, utilization, and PDN width and height, which are not used for training.

We use the multiphysics analysis flow described in Section 3.2.1, and the modeling methodology described in Section 3.2.1 to derive our model. Each P&R and analysis run requires approximately 10 hours using a single core, and the training time is around three hours for 2515 data points on an Intel Xeon E5-2640 2.5GHz when using four cores. The testing time for 1041 data points is less than two minutes.

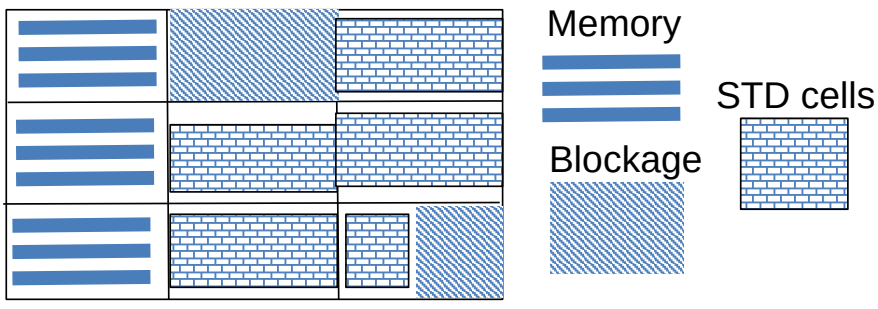


Figure 3.30: Example of a floorplan enumerated with tic-tac-toe implementation.

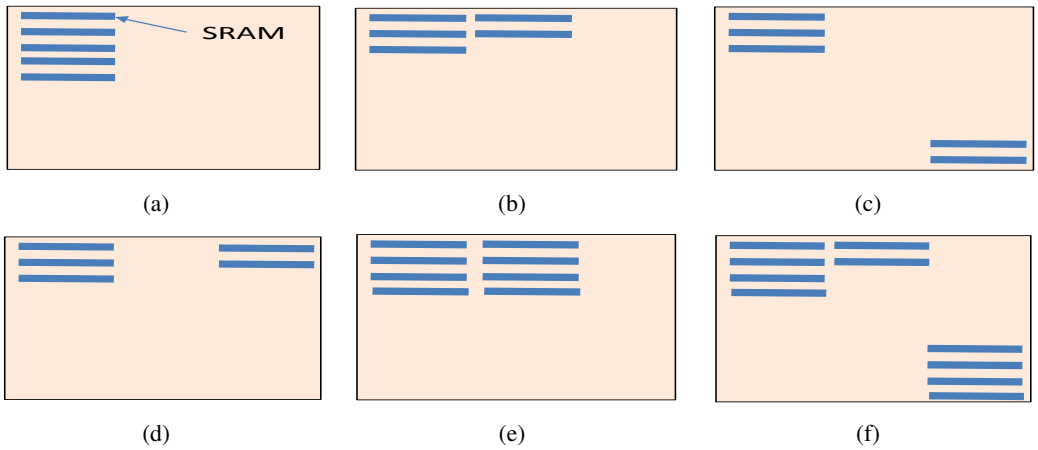


Figure 3.31: Examples of memory placements in our testcases: (a) 5×1 vertical stacking, (b) 3×1, 2×1 side-by-side arrays at upper-left, (c) 3×1, 2×1 arrays at upper-left and lower-right, (d) 3×1, 2×1 arrays at upper-left and upper-right, (e) 4×1, 4×1 side-by-side arrays at upper-left, and (f) 4×1, 2×1 side-by-side arrays at upper-left and 4×1 at lower-right.

Table 3.8: Our design of experiments.

Parameter	**Value(s)** (* is default)
Aspect ratio	{1.2, 1.1, 1.0*, 0.8}
Utilization (std cells)	{40%, 50%*, 60%, 70%}
PDN stripe width	{0.5, 0.75, 1.0*, 1.5, 2.0, 2.5, 3.5}μm
PDN stripe pitch	{7, 15, 20, 30*, 40}μm
SRAM spacing (channel width)	{6, 8, 12, 16, 20*, 24}μm
Buffer screen width	{10, 12, 14*, 16}μm
Routing metal layers	{7, 8*}
Memory placement	{Face-to-face*, face-to-back}
Clock period	*THEIA_*{*v0, v1, v3, v4*} = {3.0, 3.5*, 4.0}ns *THEIA_v2* = {3.0*}ns *nova* = {3.2*, 3.7, 4.2}ns *artificial* = {2.0*}ns
Max transition	{200*, 240, 280}ps
Max fanout	{8*, 10}
Threshold voltage mixes	{{LVT}, {LVT, RVT}*, {RVT}}
Clock buffer sizes	{{X32}*, {X32, X24}, {X32, X24, X16}}
NDRs on clock nets	{1W1S*, 2W2S, 3W3S, 3W2S, 2W3S}

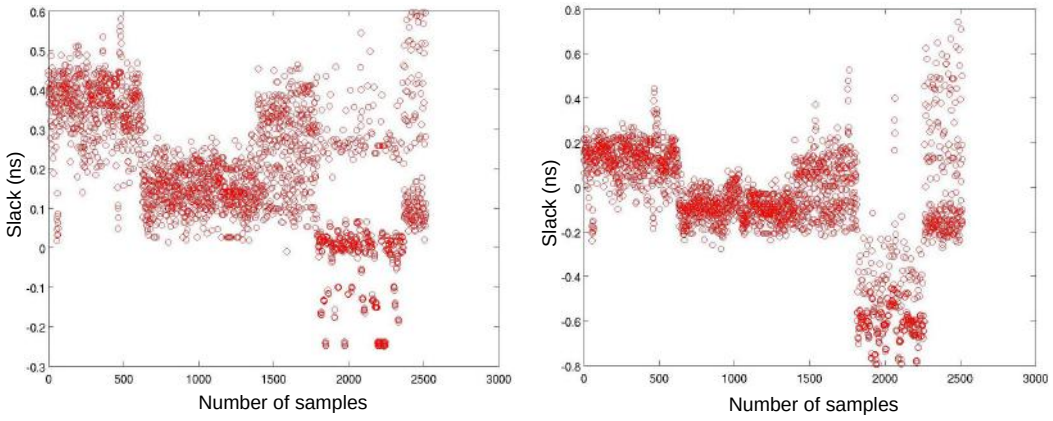
We conduct three experiments to validate our model. In all of our experiments we use data points from designs *THEIA_v0* and *nova* exclusively for testing, i.e., no data points from these two designs are used to train the model.

- **Experiment 1** tests accuracy of our model in predicting post-P&R slack values of SRAMs. With the training data generated by the design of experiments described above in Table 3.8, we apply our modeling flow described in Figure 4.25 to predict the post-P&R memory timing slack values. We also compare the accuracy of various modeling techniques and present our results in Table 3.9.

- **Experiment 2** tests accuracy of our model in predicting slack values with multiphysics analysis. We use the same design of experiments and modeling flow as in Experiment 1 to predict SRAM timing slack values after annotating IR drop from the *RedHawk* reports to cell instances.

- **Experiment 3** tests fidelity of our model in providing floorplan guidance to reduce timing failures at signoff with multiphysics analysis. We report the confusion matrix of timing pass or fail predictions, again using the same design of experiments and modeling flow.

For all of our experiments, we separately report modeling errors (i.e., predicted slack − actual slack) for both training and test datasets. Figures 3.32(a) and (b) show the ground truth that we predict. Figure 3.33 compares post-synthesis slack of SRAMs with post-P&R slack. Due to changes in constraints and implementations, post-P&R slack has no apparent correlation with post-synthesis slack.



(a) (b)

Figure 3.32: Ground truth data. (a) Slack at post-P&R stage without multiphysics analysis and (b) slack with multiphysics analysis.

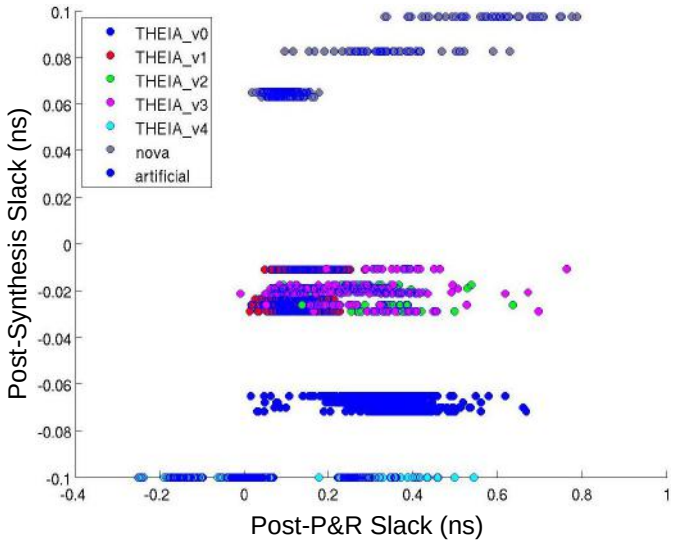


Figure 3.33: Slack at post-synthesis stage vs. post-P&R stage across six of our testcases. There is no correlation due to constraints and implementations.

Results of Experiment 1

Table 3.9 shows error metrics of our modeling techniques on the test dataset for the various machine learning techniques. Rows 4 and 5 in the table show that the worst-case error in slack prediction reduces by 30ps with our implementation of Boosting with SVM regressors, compared to the SVM-only technique. Figure 3.34(a) shows predicted versus actual slack values of memories at the post-P&R stage. Our model has worst-case error of 224ps (48%)[17] and average error of 4.0ps (7.2%). Note that most of the predicted slack values (when the actual slack values are negative) are below the solid black line (i.e., line of perfect correlation) as a result of negative-slack weighting strategy. Figure 3.34(b) shows a histogram of error in slack prediction in the test dataset, and Figure 3.34(c) shows the outcome of our negative slack weighting strategy during our model construction, i.e., greater magnitudes of the negative slack values have pessimistic predictions.

Results of Experiment 2

Figure 3.35(a) shows predicted versus actual slack values of memories with multiphysics analysis. Our model has worst-case error of 253ps (44%) and average error of 9.0ps (5.9%).

[17]The worst-case error in Experiment 1 occurs when the actual slack is 466ps, whereas the predicted slack is 242ps. The error is 242ps−466ps=−224ps; we calculate the magnitude of relative error for this data point (relative to the actual slack) as $\frac{224ps}{466ps} = 48\%$. We measure average error as the mean of all absolute errors, and average percentage error as the mean of magnitudes of relative errors (relative to actual slack values) expressed as a percentage.

Table 3.9: Error metrics of modeling techniques used in our experiments.

Technique	Min Error (ps)	Max Error (ps)	Mean Error (ps)	Standard Deviation (ps)	Mean-Square Error
LASSO	-380.5	281.6	-86.7	64.1	11.6
ANN	-250.6	272.9	-8.5	60.1	3.7
SVM	-243.7	252.9	-9.0	55.2	3.1
Boosting	-253.7	200.8	-5.1	55.7	3.1
HSM	-223.1	223.7	-4.0	58.9	3.5

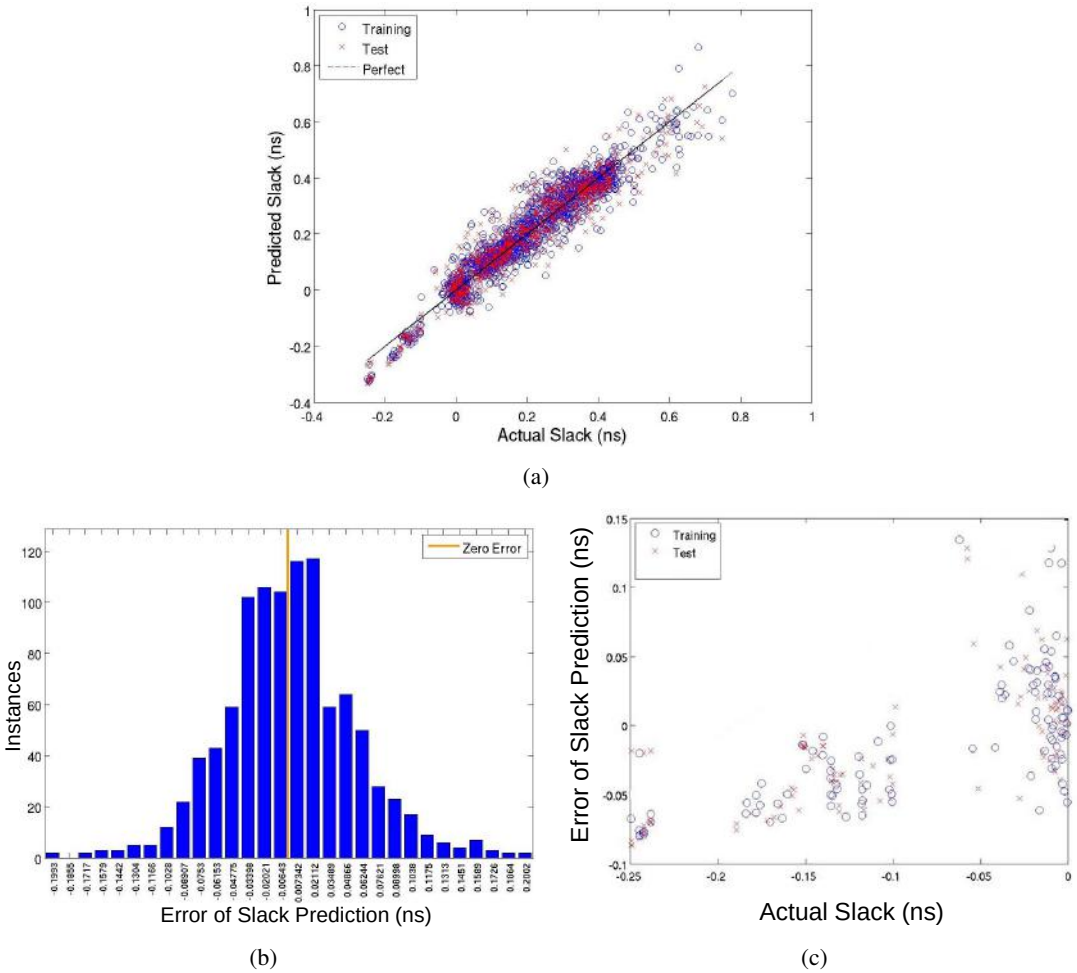
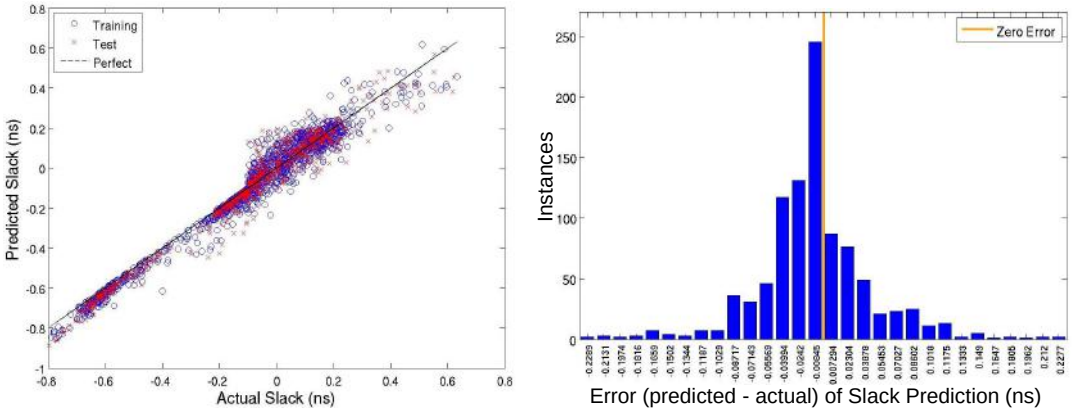


Figure 3.34: Accuracy of our model in predicting post-P&R SRAM slack values with HSM. (a) Scatter plot of actual and predicted data points in training and testing, (b) error distribution in the test dataset, and (c) effect of weighting strategy for negative slack values. Note in (c) that when actual slack values are less than -0.15ns, the error values are negative, i.e., the predicted slack is always pessimistic as compared to the actual slack.

Figure 3.35(b) shows a histogram of error in slack prediction in the test dataset. Even though our worst-case error is large, only a small number of predictions have error greater than 100ps. Some of these predictions are more pessimistic due to our negative-slack weighting strategy. Since prediction of slack with multiphysics analysis is more difficult than predicting post-P&R slack, the errors are larger than those in Experiment 1.



(a) (b)

Figure 3.35: Accuracy of predicted multiphysics SRAM slack values with HSM. (a) Scatter plot of data points in training and testing and (b) error distribution for the test dataset.

Results of Experiment 3

Figure 3.36 shows the confusion matrix of our predictions on the test set. Our predictive model of SRAM slack values with multiphysics analysis has few ($\sim$3%) false negatives (fn), that is, pessimistic predictions in which we provide guidance to change a floorplan that is actually not required. Our model also has few false positives (fp), that is, cases for which our model incorrectly deems a floorplan to be good. Such wrong predictions have a $\sim$4% incidence. The number of true positives (tp), that is, both the predicted and actual slack values are positive, is 584. The number of true negatives (tn), that is, both the predicted and actual slack values are negative, is 384.

In our model, the *precision* [85] (i.e., the ratio of tp to the sum of tp and fp) is 93.3%, and the *recall* (i.e., the ratio of tp to the sum of tp and fn) is 95.0%. Similarly, the precision for negative slack data points (referred to as *negative predictive value* in the machine learning literature) is 92.5% and the recall for negative slack data points (referred to as *specificity* in the machine learning literature) is 90.1%. Based on these large values of precision and recall metrics, we believe that our model can provide guidance to designers on the risk of SRAM timing failures with high fidelity.

		Actual	
		Pass	Fail
Predicted	Pass	584	42
	Fail	31	384

Figure 3.36: Confusion matrix of our predictions with HSM. False positives (42) are optimistic predictions, while false negatives (31) are pessimistic predictions.

3.2.3 Conclusions

Early prediction of post-layout timing failures is important to reduce design cost and turnaround time. However, this prediction problem is very difficult as it must comprehend tool flows, noise, and the physics used during timing analysis. We propose a machine learning-based methodology to predict post-P&R slack of SRAMs at the floorplanning stage, given only a netlist, constraints and floorplan context. We demonstrate that our methodology can be extended to predict slack with multiphysics (STA and DVD) analysis. We develop a new implementation of Boosting with SVM in which we use a negative-slack bias strategy. This strategy guides model predictions to be less optimistic when the actual slack values are negative. We report worst-case modeling error of 253ps in predicting slack with multiphysics analysis, and average error of 9.0ps. The number of predictions with error greater than 100ps are few ($\sim$15) in our test dataset. Fidelity of our predictions is high as measured by the precision and recall metrics. We believe that SoC designers can use our methodology to avoid floorplans and constraints that may cause timing failures at signoff.

3.3 BEOL Stack-Aware Routability Prediction from Placement

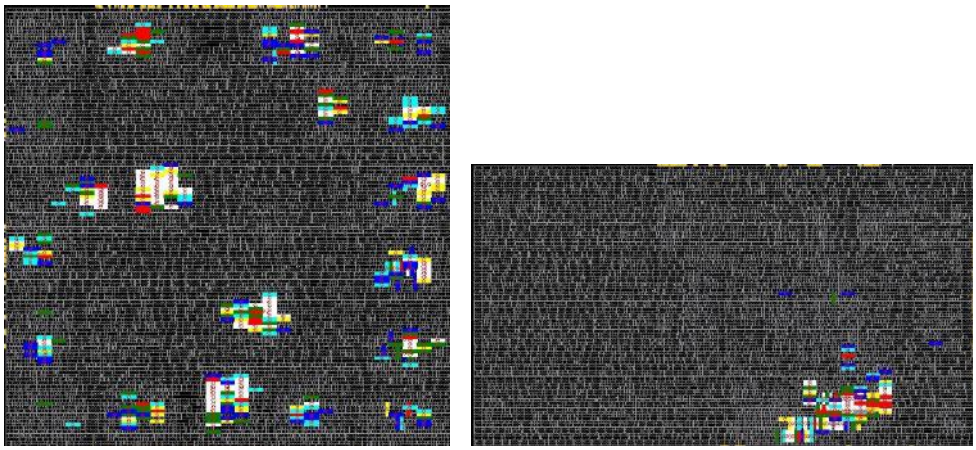
Physical design of digital integrated circuits in advanced technology nodes is very complex due to multiple design rules that must be satisfied before tapeout. Design rule violations are reported by commercial place-and-route (P&R) tools after the routing stage. However, discovering many design rule violations post-routing is costly: at best, it consumes engineer resources to fix all the violations and increases design turnaround time. Sometimes, the number of design rule violations is so large as to be unfixable; this scenario leads to disruptive changes to the placement, layout contexts and constraints. Early prediction of routability in the physical design flow is therefore critical to reduce design turnaround time and cost. However, to our best knowledge, no routability models exist today that enable IC physical design engineers to perform fast and accurate design-space exploration of timing constraints, utilization, aspect ratio and back-end-of-line (BEOL) stack options. Today, designers use congestion maps from P&R tools at the placement stage to predict routability. Congestion maps alone may not be sufficient (and, can be highly misleading) for the prediction of routability as measured by the number of design rule check (DRC) violations.[18] This is because congestion maps do not comprehend design rules and factors such as pin density or timing criticality, that affect local routability.

Physical design engineers typically use congestion maps from trial routing at the placement stage to determine whether a given placement is likely to be routable. However, this is still largely an "art", akin to reading tea leaves, as congestion maps are not straightforward indicators of design rule violations in detailed routing.

As a motivating illustration, we show two implementations of *aes_cipher_top* [318] and ARM *Cortex M0* designs in 28nm FDSOI with eight-track cells and BEOL stack with five metal layers, obtained using a commercial P&R tool. We implement *aes_cipher_top* with aspect ratio set to 1.0 and *Cortex M0* with aspect ratio set to 2.0. Figures 3.37(a) and (b) show layouts and congestion maps of *aes_cipher_top* and *Cortex M0*, respectively. Regions in red indicate congestion overflow (i.e., the difference in supply and demand of routing resources) that are < -5 and ≥ -7 and regions in white color indicate overflow that are < -7, i.e., white color indicates worse congestion than red color. By looking at these two maps, an experienced physical design engineer may conclude that the placement for *aes_cipher_top* is unroutable due to many regions of high congestion that can potentially lead to a large number of DRCs, and may conclude that the placement for *Cortex M0* is routable (or, routable with a manually-fixable number of DRCs) due to few hotspots in the right-bottom region. However, Figure 3.38(a) shows that

[18] In the following, we use #DRCs to denote the number of design rule violations (after the routing tool has run).

post-routing DRC violations are ∼6 for *aes_cipher_top*, and that these do not occur in the highly congested regions. Figure 3.38(b) shows that sufficient post-routing DRC violations occur for *Cortex_M0* in the congested region as well as in other non-congested regions, making the placement unroutable. The reasons for these DRC violations are not intuitive from examination of the congestion maps. We demonstrate below that by using relevant placement-driven parameters beyond congestion maps information, we can create models that accurately predict routability of a given BEOL stack-specific placement.[19]



(a) (b)

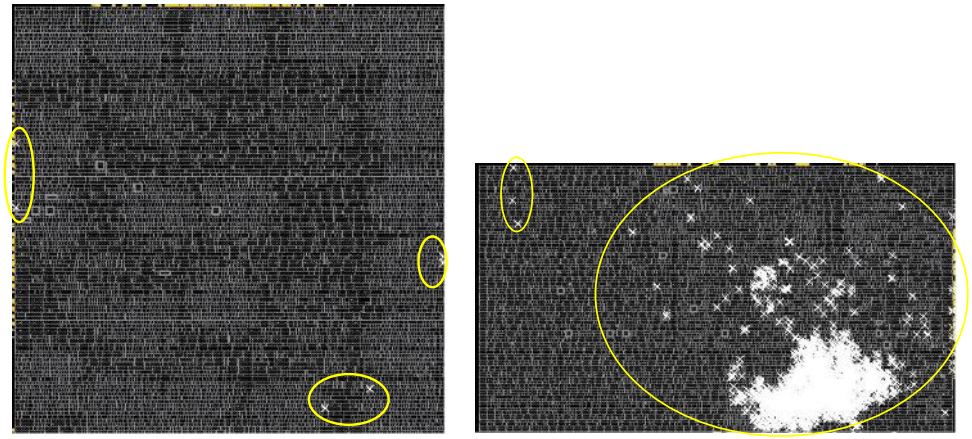
Figure 3.37: Congestion maps at placement stage in 28nm FDSOI foundry technology, with 8T cells, of (a) *aes_cipher_top* implementation at 77% utilization, aspect ratio 1.0 and BEOL stack with five metal layers and (b) ARM *Cortex M0* implementation at 77% utilization, aspect ratio 2.0 and BEOL stack with five metal layers. Red and white regions indicate large congestion with overflow < -5.

We define a placement to be *routable* when the #DRCs is $<$ *threshold* after the routing stage; conversely, a placement is *unroutable* when the #DRCs is $\geq$ *threshold* after the routing stage.[20] In this work, we develop models using data mining or machine learning techniques to accurately predict routability of a BEOL stack-specific placement. We study and propose placement-driven parameters that enable us to achieve high prediction accuracy. Applications of our models include the following use cases.

- Given a netlist, clock period, utilization, aspect ratio and BEOL stack-specific placement, our models predict whether the placement will be routable.

[19] We use the term "BEOL stack-specific placement" to acknowledge that the placement tool will output different placement solutions for the same netlist and site map, according to the specific metal layer stack.

[20] In this work, we set the *threshold* to 50, i.e., we assume (rather conservatively) that 50 DRC violations remaining after detailed routing can be manually fixed.



(a) (b)

Figure 3.38: DRC violations after routing in 28nm FDSOI foundry technology, with 8T cells, of (a) *aes_cipher_top* implementation at 77% utilization, aspect ratio 1.0 and BEOL stack with five metal layers and (b) ARM *Cortex M0* implementation at 77% utilization, aspect ratio 2.0 and BEOL stack with five metal layers. The white crosses show the DRC violations and the yellow oval shapes highlights these.

- Our models predict iso-performance Pareto frontiers of utilization, number of metal layers and aspect ratio based on very few (≤ 20) placements (and, no routing or trial routing) of a design. Using these Pareto frontiers, a designer can determine the minimum number of metal layers[21] or the maximum achievable utilization of a block.

To the best of our knowledge, the above use cases are not fully and accurately served in current design methodologies and flows as reviewed in Section 2.1.2. As described above, physical design engineers estimate routability based on congestion maps from commercial P&R tools, but these estimates can be quite misleading. Currently, to our knowledge, no tool exists that predicts the Pareto frontiers of utilization, number of metal layers and aspect ratio based on very few placements. The key contributions of our work are as follows.

1. We demonstrate that congestion maps from commercial tools are likely insufficient to predict routability. In fact, sometimes congestion maps can mislead designers to believe that a placement is routable, when it is actually not. We quantify the classification error to be 38% in 28nm FDSOI technology by using only congestion maps to predict routability.

2. We describe a methodology based on machine learning to predict whether a placement is routable, given a netlist, clock period, utilization, aspect ratio and BEOL stack. Our methodology is applicable to both 2D and 3D ICs.

[21] In advanced nodes, due to complexity of lithography (e.g., double-patterning, triple-patterning, etc.), each metal layer is a sizeable percentage of wafer cost.

3. We describe new parameters that we identify – related to congestion distribution, critical timing path distribution, and available routing resources [104] [108] – that guide our learning-based models to accurately predict routability, given a netlist, clock period, utilization, aspect ratio and BEOL stack. Note that we do not use any information from trial routing or early global routing from P&R tools. The worst-case classification error in our models is 14.1% in 45nm GS foundry technology. In 28nm FDSOI technology, classification error of our model is 13%.

4. Our models also enable accurate prediction of iso-performance Pareto frontiers of utilization, number of metal layers and aspect ratio based on very few placements.

3.3.1 Methodology

We now describe our modeling parameters, how we have identified them, and our modeling methodology. As noted above, the goal of our modeling is to predict whether a given BEOL stack-specific placement is routable. In our experimental results, we show applications of our models to predict Pareto frontiers of utilization, number of metal layers and aspect ratio based on very few placements. Note that we do not use any trial routing or early global routing information in making our predictions.

List of Parameters

We divide the placement region into grids whose height and width are multiples of the P&R tool's gcell (i.e., global routing cell) height and width [287].[22] We extract modeling parameters from these grids that intuitively affect local routing of net segments on various layers of metal. We obtain the following parameters from a placement for each grid:

- pin density;
- minimum proximity of any pair of pins;
- number of complex cells, i.e., *AOI*, *OAI*, three-input *XOR* and *XNOR*, and *MUX* cells;
- sum of incoming and outgoing hyperedges (signal nets with pins both inside and outside the grid);
- number of buried nets, that is, the number of nets that have all of their pins within the grid;

[22] Typical grid size in a commercial P&R tool is 15 tracks $\times$ 15 tracks [286], where a track is equal to the metal M2 pitch value.

- arithmetic and geometric mean values of placement-based Rent parameter;
- the worst signal transition time of all pins at the worst corner;
- the smallest values of the worst negative slack (WNS) of setup time of any pin within the grid;
- the percentage of routing resources consumed by the power delivery network (PDN); and
- the density of vertical interconnects (VIs) in each tier of a 3DIC.

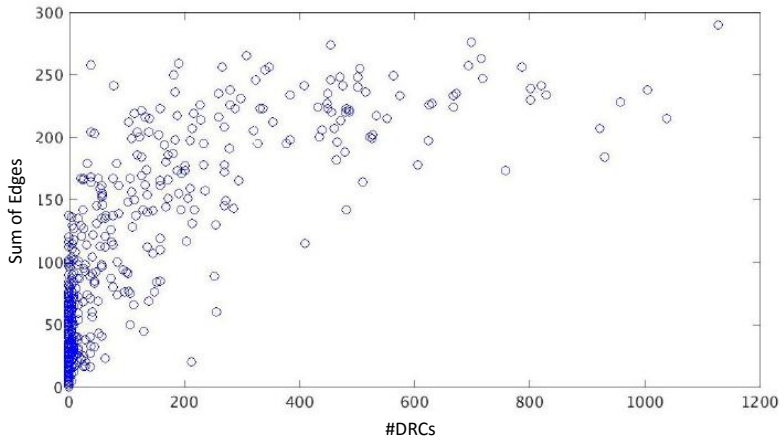
Figures 3.39(a) and (b) show correlation of #DRCs (our routability metric) to the sum of incoming and outgoing hyperedges[23] and minimum proximity of pins, respectively. When a grid has small values of minimum proximity of pins, it indicates that pins of adjacent cells within a grid are placed very closely and can lead to spacing-related DRC violations at the routing stage. When a grid has pins with large transition times or small WNS, it indicates that cells can be sized up and buffers can be inserted that can worsen local routability and increase the number of DRC violations.

From the above parameters, we compute the coefficient of variation (i.e., the ratio of standard deviation to mean) of pin density, minimum proximity, number of complex cells, sum of incoming and outgoing hyperedges, number of buried nets, arithmetic and geometric mean values of placement-based Rent parameter, worst transition time and worst WNS. We use the following as our modeling parameters.[24]

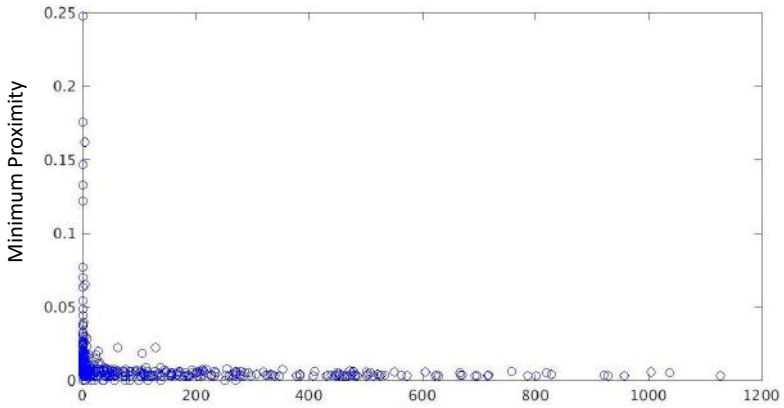
1. Coefficient of variation of pin density, minimum proximity, number of complex cells, sum of incoming and outgoing edges, number of buried nets, arithmetic and geometric mean values of Rent parameter, worst transition time and worst WNS.
2. Maximum values (across all grids) of pin density, number of complex cells, sum of incoming and outgoing edges, number of buried nets, arithmetic and geometric mean values of Rent parameter and worst transition time.
3. Minimum values (across all grids) of minimum proximity and worst WNS.
4. Utilization of standard cells.

[23] We use the term edges below to denote hyperedges.

[24] We greedily select parameters by incrementally adding each parameter one by one, creating models using our training dataset and checking accuracy of models using the test dataset. The next parameter to be selected is the one which improves model accuracy the most. We list the parameters that achieve the highest accuracy in our experiments.



(a)



(b)

Figure 3.39: Correlations of #DRCs with (a) sum of incoming and outgoing hyperedges and (b) minimum proximity of pins within a grid.

5. Clock period of design used for P&R.
6. Aspect ratio of the floorplan.
7. Numbers of horizontal and vertical tracks, which we calculate from the height and width of the placement region and pitches of all horizontal and vertical layers of the BEOL stack. For example, if there are three horizontal layers, each with pitch p_h, and the core height is H, then the number of horizontal tracks is $3 \cdot H / p_h$.
8. The percentage of routing resources consumed by the power delivery network (PDN).
9. The density of vertical interconnects (VIs) in each tier of a 3DIC.

The parameters listed in 1–3 above are an indication of the quality of a BEOL stack-specific placement and how it spreads across multiple grids. If the spread (indicated by coefficient of variation) is large, it suggests that grids may have local congestion. The timing parameters capture how critical paths and critical pins are distributed and the extent of violation. Large violations at the placement stage indicate that buffers can be inserted during routing, which can increase local congestion and violate design rules. The parameters listed in 4–6 above describe the layout context and timing constraints, and the parameters listed in 7 above capture details of the BEOL stack and the amount of routing resources available for the design. The parameters listed in 8 and 9 above are used for 3DICs and capturing the amount of routing resources available for signal and clock routing and the density of VIs that connect signals crossing tiers.

Parameter Value Interpolation and Extrapolation

Given a few (≤ 20) placement solutions of a design that span some values of clock period, utilization, aspect ratio and BEOL stack, we need to generate additional values of parameter to train our models. We propose the following methodology, partially adapted from [190], to interpolate and extrapolate values of parameters (e.g., {maximum, coefficient of variation, ...} $\times$ {pin density, sum of edges, ...}) from Section 3.3.1 across multiple values of clock period, utilization, aspect ratio and BEOL stack. We train models for each parameter as a function of clock period, utilization, aspect ratio, number of horizontal (#H) and vertical (#V) tracks and known values of the parameter extracted from the given placements. We train models for each parameter that achieves a given error bound UB_{error} as follows.

In procedure *genParamModel* of Algorithm 1, we assume that we are given a set $\mathbb{P}$ of placements of a design and an error upper bound UB_{error}. We expect a minimum of four and a maximum of 20 placements, that is $3 < |\mathbb{P}| \leq 20$.[25] In Lines 1–4, we choose a subset of three placements $\mathbb{P}_{tr}$ from $\mathbb{P}$ for training and one placement p_f for model fitting. In Line 5, we extract values of the parameter (e.g., max pin density, etc.) as well as other inputs used to fit a model. These inputs are the ratio of clock periods, utilization, aspect ratio, #H and #V tracks of p_f to those in $\mathbb{P}_{tr}$, as well as the values of the same parameter from the placements in $\mathbb{P}_{tr}$. In Line 6, we obtain the value of the parameter from p_f that we use to fit, and in Line 7 we train a model f_m. For example, to train a model for max pin density, we use the max pin density values of placements in $\mathbb{P}_{tr}$ and the ratio of clock periods, utilization, aspect ratio, #H and #V tracks of p_f

[25] Given placement solutions of a new design, we do not know how each parameter varies with the inputs. We use at least three data points to capture parameters that can be polynomial (with polynomial degree ≥ 2) with respect to the input parameters.

to those in $\mathbb{P}_{tr}$. In Line 8, we initialize the error e to $2\times$ UB$_{error}$. In Lines 9–22, we refine f_m by retraining to achieve $e \leq$ UB$_{error}$. In Lines 9–13, we choose the remaining placements in $\mathbb{P}$ for testing, extract the fitting parameters and the actual value of the parameter. We then use f_m to test the model. In Lines 14–20, we check if the error $e >$ UB$_{error}$ in the placements used for testing, we add these placements to our set of placements used for training $\mathbb{P}_{tr}$, and remove these from $\mathbb{P}$. The while loop exits when either all the remaining placements in $\mathbb{P}$ have been added to $\mathbb{P}_{tr}$ or when f_m achieves $e \leq$ UB$_{error}$.[26] Using f_m, we can now interpolate or extrapolate values of the parameter. That is, for any new value of clock period, utilization, aspect ratio and BEOL stack, we calculate the ratios of clock periods, utilization, aspect ratio, #H and #V tracks, etc. and use f_m to estimate the value of the parameter. We train one model f_m for each parameter described in Section 3.3.1.

Algorithm 1 Interpolation and extrapolation of parameter values.

Procedure *genParamModel*
Input : $\mathbb{P}$ ($3 < |\mathbb{P}| \leq 20$), UB$_{error}$
Output: Model f_m for parameter $m \in$ {max pin density, max # edges, etc.}.

1: $\mathbb{P}_{tr} \leftarrow \{p_i, p_j, p_k\} \in \mathbb{P}$, $i, j, k \leq |\mathbb{P}|$
2: $\mathbb{P} \leftarrow \mathbb{P} \setminus \{p_i, p_j, p_k\}$ // remove p_i, p_j, p_k from $\mathbb{P}$
3: $p_f \leftarrow \{p_l\} \in \mathbb{P}$, $l \leq |\mathbb{P}|$
4: $\mathbb{P} \leftarrow \mathbb{P} \setminus \{p_l\}$
5: $\mathbf{X}_{tr} \leftarrow \mathbb{P}_{tr} \cup \{p_f\}$ // extract inputs to f_m, e.g., ratio of util, clk period, ...
6: $y_{tr} \leftarrow p_f$ // extract value of parameter m in p_f
7: $\hat{y}_{tr} \leftarrow f_m(\mathbf{X}_{tr})$ // f_m is trained using MARS, SVM, etc.
8: $e \leftarrow 2\times$ UB$_{error}$
9: **while** $|\mathbb{P}| > 0$ && $|e| >$ UB$_{error}$ **do**
10: **for all** $p \in \mathbb{P}$ **do**
11: $y \leftarrow p$
12: $\mathbf{X}_{test} \leftarrow \mathbb{P}_{tr} \cup \{p\}$
13: $e \leftarrow f(\mathbf{X}_{test}) - y$
14: **if** $|e| >$ UB$_{error}$ **then**
15: $\mathbb{P}_{tr} \leftarrow \mathbb{P}_{tr} \cup \{p\}$
16: $\mathbb{P} \leftarrow \mathbb{P} \setminus \{p\}$
17: $\mathbf{X}_{tr} \leftarrow \mathbb{P}_{tr} \cup \{p_f\}$
18: $y_{tr} \leftarrow p_f$
19: $\hat{y}_{tr} \leftarrow f_m(\mathbf{X}_{tr})$ // retrain model
20: **end if**
21: **end for**
22: **end while**

To train each f_m, we use multivariate adaptive regression splines (MARS) [85] and Support Vector Machine (SVM) [85] using a Radial Basis Function (RBF) kernel [85] and combine their responses using weights determined by least-squares regression, and train a model for each parameter. Once we have obtained a set of estimated values of parameters using the above methodology, we will use these as our modeling parameters to predict routability, and predict Pareto frontiers as described in Section 3.3.2.

[26] Note that it is possible that we use all the placement solutions to train a model for a parameter, but error is $>$ UB$_{error}$.

The goal of modeling is to predict whether a BEOL stack-specific placement is routable (for a given router). We classify an implementation to be unroutable when the number of design rule violations at the post-route stage (or, the number of DRCs) obtained from commercial place-and-route (P&R) tools is ≥ 50. When the number of violations is < 50, these violations are typically fixed by designers manually. Our model for each technology is a binary classifier developed using the SVM algorithm using a RBF kernel. We use the label "+1" when a design is routable and the label "-1" when an implementation is unroutable. We use five-fold cross-validation to generalize our models.

3.3.2 Experimental Setup and Results

We now describe our experimental setup and design of experiments (DoE), and present our results. We describe our DoE for three foundry technologies and designs that we use to train our models. To test our models, we use new designs that the training data has not seen. We describe the DoE of our test dataset and applications our models in the description that follows. We use the A3D flow (described in Section 5.1) to predict the routability and Pareto frontiers for 3DICs.

Design of Experiments

We conduct our experiments on multiple designs: *aes_cipher_top* from OpenCores [318], *aes_x2* and *aes_x3* created by instantiating and stitching two and three *aes_cipher_top* designs respectively, ARM *Cortex M0* core, *leon3mp* core and *jpeg_x5* created by instantiating and stitching five *jpeg_encoder* designs [318]. We synthesize these designs using Synopsys *Design Compiler vI-2013.12-SP3* [337]. Table 3.10 shows the DoE used to obtain ground truth for modeling of designs with 28nm FDSOI eight-track (8T), 28nm LP 12-track (12T) and 45nm GS nine-track (9T) technology libraries. Table 3.11 shows the DoE used to obtain ground truth for modeling of designs with 28nm FDSOI 8T and 28nm LP 12T technology libraries. We use these data points to train our models (one model for each technology). We create custom LEF files with eight, seven, six, five and four metal layers, all having $1\times$ pitch as the Mx layer. We run P&R using Cadence *Innovus v15.2* [287]. We perform denoising [97] [107] by executing P&R for each point in our DoE six times, i.e., by perturbing each P&R clock period by {-5, +0, +5}ps at a fixed utilization value, and by perturbing each utilization value by {-0.05, +0.00, +0.05}%

at a fixed clock period.[27] We classify a placement as unroutable when all the six runs indicate that the #DRCs is ≥ 50. We then use custom scripts in *Tcl* to extract the parameters described in Section 3.3.1 as follows.

- We use grids to divide the entire layout. Note that the number of grids varies with designs and utilizations.
- To obtain pin density per grid, we count the number of pins in each grid and divide it by the grid area.
- To obtain the minimum proximity of pins, we calculate half-perimeter wirelength (HPWL) of all pairs of pins within a grid. We then use the minimum HPWL of these values as minimum proximity.
- To obtain the number of complex cells, we obtain all cells within the bounding box of a grid and names of cell masters. We then count the cells who master names are either *AOI*, *OAI*, three-input *XOR* and *XNOR*, or *MUX*.
- To obtain the number of buried nets, we count nets that have all of their pins within a grid.
- To obtain the number of edges, we count the number of incoming incident edges to pins within a grid and the number of outgoing edges from pins within a grid. We then add the number of incoming and outgoing edges.
- To obtain the placement-based Rent parameter, we use the *RentCon* tool [326] with 15 tracks $\times$ 15 tracks grid size, and shifting of evaluation windows by $\frac{1}{4}\times$ the size of the grid. Thus, 16 grids over the layout region are used for evaluation of the placement-based Rent parameter.
- To obtain the worst signal transition time of all pins within a grid, we obtain all pins over all critical paths that are within a grid in the worst corner. We then take the worst signal transition time of all these pins using the *get_property* command.

[27] On a 2.6GHz Intel Xeon E5-2690 processor, placement takes around 1.5 hours (on average) for *aes_cipher_top*, 2.4 hours (on average) for *aes_x2* and 3.3 hours (on average) for *aes_x3* with two cores. On average, routing executes in 2.5, 3.1 and 5 hours, respectively for these designs when the placement is routable, and takes around 3.5, 5.1 and 6.3 hours, respectively when the placement is unroutable. We choose clock periods so that the implementations meet timing at the post-route stage. Note that out of six runs, one of the runs that uses the utilization and clock period values from the DoE is duplicated.

- To obtain the worst setup WNS within a grid, we check all pins within a grid and obtain WNS of the paths to which these pins belong. We then take the worst (i.e., minimum value) WNS as our parameter.

- To obtain the number of VIs and their locations, we parse the post-routing DEF files of each tier.

- We obtain statistical information, i.e., max and min values by considering the maximum and minimum values of our parameters across all grids. We calculate coefficient of variation by dividing the standard deviation by the mean value of parameters (across all grids).

We now explain our choice of grid sizes. We use grid sizes of 45 tracks $\times$ 45 tracks because these are multiples of gcell sizes (15 tracks $\times$ 15 tracks) used by our P&R tool. We have tried grid sizes of 15 tracks $\times$ 15 tracks, 30 tracks $\times$ 30 tracks and 90 tracks $\times$ 90 tracks as well. Small grid sizes hide the correlation between #DRCs and our parameters because multiple neighboring gcells can together cause DRC violations. Large grid sizes blur the differences between various utilizations, i.e., do not capture local hotspots. We use 45 tracks $\times$ 45 tracks because these grids show correlation with #DRCs and at the same time does not blur differences across utilizations. Figures 3.40(a) and (b) show that by using grid sizes of 15 tracks $\times$ 15 tracks the correlation between pin density and #DRCs is not apparent in 28nm FDSOI. By using grid sizes of 45 tracks $\times$ 45 tracks, the correlations between #DRCs and pin density become more apparent. We compare the coefficient of determination R^2 in both figures and observe that Figure 3.40(b) has larger R^2 value than Figure 3.40(a). Similarly, Figures 3.41(a) and (b) show that by using grid sizes of 15 tracks $\times$ 15 tracks the correlation between #DRCs and the number of complex cells is not apparent in 28nm FDSOI. By using grid sizes of 45 tracks $\times$ 45 tracks, the correlation between #DRCs and the number of complex cells becomes more apparent. We compare the coefficient of determination R^2 in both figures and observe that Figure 3.41(b) has larger R^2 value than Figure 3.41(a).[28]

For 2D modeling, we use 1377 data points for training in 28nm FDSOI, out of which 906 data points are from routable implementations and the remaining 471 are from unroutable implementations. In 28nm LP, we use a total of 918 data points for training, out of which 861 are from routable and 25 are from unroutable implementations. In 45nm GS, we use a total of 918 data points for training, out of which 618 are from routable and 290 are from unroutable

[28]Commercial P&R tools use pitch of the M2 layer as track size. In our technology libraries, 28nm FDSOI and LP technologies have M2 pitch of 0.1μm and 45nm GS technology has M2 pitch of 0.14μm. Therefore, our grid sizes are 4.5μm $\times$ 4.5μm for 28nm FDSOI and LP, and 6.3μm $\times$ 6.3μm for 45nm GS.

Table 3.10: Design of experiments used to obtain ground truth for training our 2DIC model.

		aes_cipher_top	*aes_x2*	*aes_x3*
Syn Clk Period (ns)	28FDSOI, 8T	0.6	0.6	0.6
	28LP, 12T	0.6	0.6	–
	45GS, 9T	1.0	1.0	–
#Instances	28FDSOI, 8T	11461	22770	34834
	28LP, 12T	11783	24744	–
	45GS, 9T	13601	28562	–
#FFs	28FDSOI, 8T	530	1060	1590
	28LP, 12T	530	1060	–
	45GS, 9T	530	1060	–
P&R Clk Period (ns)	28FDSOI, 8T	{0.6, 0.9, 1.1}	{0.6, 0.9, 1.1}	{0.6, 0.9, 1.1}
	28LP, 12T	{0.6, 0.9, 1.1}	{0.6, 0.9, 1.1}	–
	45GS, 9T	{1.0, 1.2, 1.8}	{1.0, 1.2, 1.8}	–
Util (%)	28FDSOI, 8T	{70, …, 86}	{70, …, 86}	{70, …, 86}
	28LP, 12T	{70, …, 86}	{70, …, 86}	–
	45GS, 9T	{70, …, 86}	{70, …, 86}	–
Aspect Ratio	28FDSOI, 8T	{1.0, 1.3, 1.8}	{1.0, 1.3, 1.8}	{1.0, 1.3, 1.8}
	28LP, 12T	{1.0, 1.3, 1.8}	{1.0, 1.3, 1.8}	–
	45GS, 9T	{1.0, 1.3, 1.8}	{1.0, 1.3, 1.8}	–
#Metal Layers	28FDSOI, 8T	{6, 5, 4}	{6, 5, 4}	{6, 5, 4}
	28LP, 12T	{6, 5, 4}	{6, 5, 4}	–
	45GS, 9T	{6, 5, 4}	{6, 5, 4}	–
Grid Size (#tracks)	28FDSOI, 8T	45 × 45	45 × 45	45 × 45
	28LP, 12T	45 × 45	45 × 45	–
	45GS, 9T	45 × 45	45 × 45	–
#Routable	28FDSOI, 8T	306	306	294
	28LP, 12T	437	424	–
	45GS, 9T	317	311	–
#Unroutable	28FDSOI, 8T	153	153	165
	28LP, 12T	22	35	–
	45GS, 9T	142	148	–

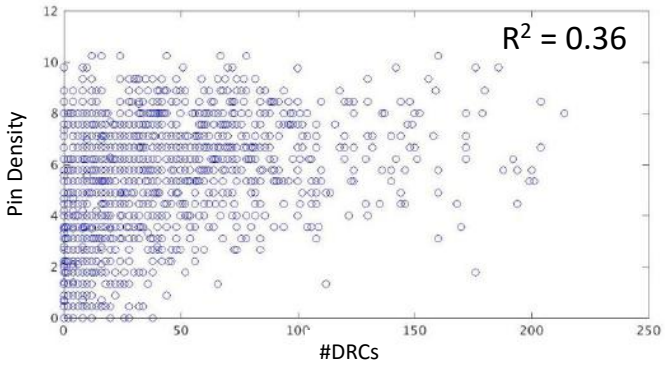
Table 3.11: Design of experiments used to obtain ground truth for training our 3DIC model.

		aes_cipher_top	*aes_x2*	*aes_x3*	*jpeg_encoder*
Syn Clk Period (ns)	28FDSOI, 8T	0.6	0.6	0.6	0.8
	28LP, 12T	0.8	0.8	0.8	1.0
#Instances	28FDSOI, 8T	11463	22778	34839	111356
	28LP, 12T	11778	24767	35363	111502
#FFs	28FDSOI, 8T	530	1060	1590	23560
	28LP, 12T	530	1060	1590	4712
P&R Clk Period (ns)	28FDSOI, 8T	{0.6, 0.8, 1.1}	{0.6, 0.8, 1.1}	{0.6, 0.8, 1.1}	{0.8, 1.1, 1.3}
	28LP, 12T	{0.8, 0.9, 1.1}	{0.8, 0.9, 1.1}	{0.8, 0.9, 1.1}	{1.0, 1.2, 1.4}
Util (%)	28FDSOI, 8T	{75, …, 86}	{75, …, 86}	{75, …, 86}	{75, …, 86}
	28LP, 12T	{75, …, 86}	{75, …, 86}	{75, …, 86}	{75, …, 86}
Aspect Ratio	28FDSOI, 8T	{1.0, 1.2, 1.9}	{1.0, 1.2, 1.9}	{1.0, 1.2, 1.9}	{1.0, 1.2, 1.9}
	28LP, 12T	{1.0, 1.2, 1.9}	{1.0, 1.2, 1.9}	{1.0, 1.2, 1.9}	{1.0, 1.2, 1.9}
#Metal Layers (Tier 0, Tier 1)	28FDSOI, 8T	{(6, 6), (5, 5),	{(6, 6), (5, 5),	{(6, 6), (5, 5),	{(6, 6), (5, 5),
	28LP, 12T	(4, 4), (4, 5), (5, 4)}	(4, 4), (4, 5), (5, 4)}	(4, 4), (4, 5), (5, 4)}	(4, 4), (4, 5), (5, 4)}
Grid Size (#tracks)	28FDSOI, 8T	45 × 45	45 × 45	45 × 45	45 × 45
	28LP, 12T	45 × 45	45 × 45	45 × 45	45 × 45
#Routable	28FDSOI, 8T	432	432	430	468
	28LP, 12T	512	512	511	520
#Unroutable	28FDSOI, 8T	108	108	110	72
	28LP, 12T	28	28	29	20

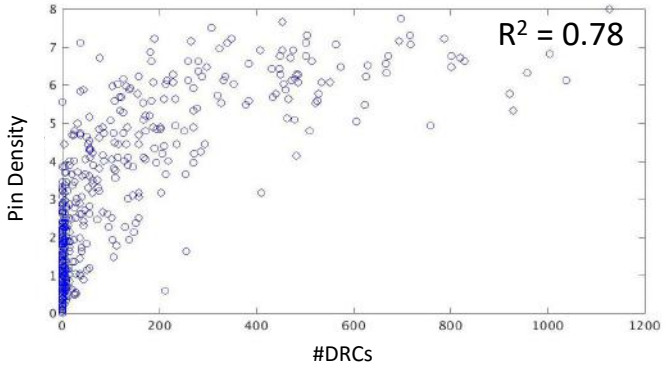
implementations. For 3D modeling, we use 2160 data points for training in 28nm FDSOI, out of which 1762 data points are from routable implementations and the remaining 398 are from unroutable implementations. In 28nm LP, we use a total of 2160 data points for training, out of which 2055 are from routable and 105 are from unroutable implementations. Owing to the lack of a PDN flow for 3DICs, we implement a 2DIC PDN on each tier as shown in Figures 3.42(a) and (b). We create a top-level global mesh on metal layers M5 and M6, and a local mesh on metal layer M3. We use around 20% of the routing resources on M3 for PDN. We create our models using *MATLAB vR2013a* scripts. We conduct two experiments to demonstrate application of our models, as follows.

- **Experiment 1.** To determine whether a placement is routable using our models on unseen data points from new designs across various technologies.

- **Experiment 2.** To predict Pareto frontiers of utilization, aspect ratio and number of metal layers at iso-performance.

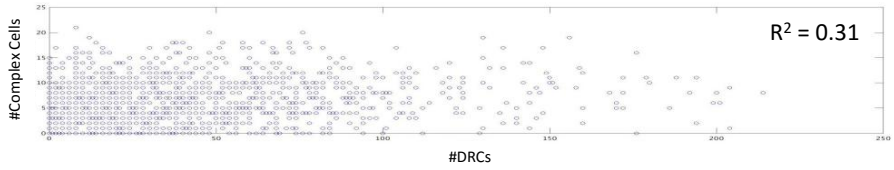


(a)

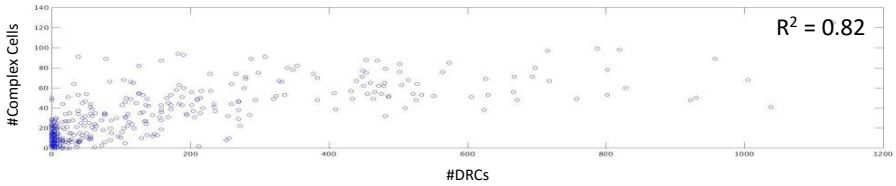


(b)

Figure 3.40: Correlations of #DRCs with pin density in 28nm FDSOI. The size of grids is set to (a) 15 tracks × 15 tracks and (b) 45 tracks × 45 tracks.



(a)



(b)

Figure 3.41: Correlations of #DRCs with the number of complex cells in 28nm FDSOI. The size of grids is set to (a) 15 tracks × 15 tracks and (b) 45 tracks × 45 tracks.

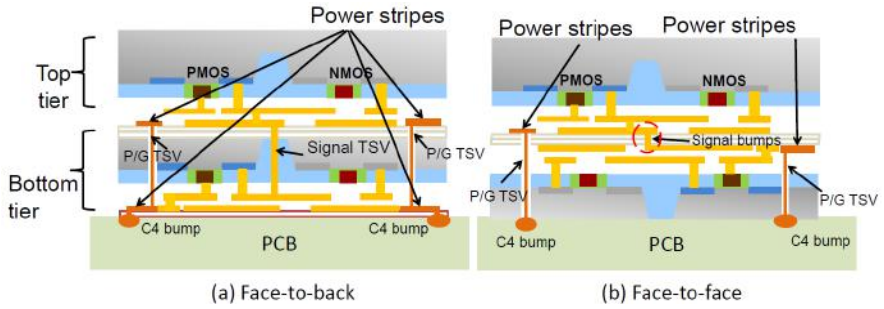


Figure 3.42: Illustration of 3DIC with two tiers with power delivery network: (a) face-to-back and (b) face-to-face. The power through-silicon vias (TSVs) connect Vdd and ground signals across tiers.

Results of Experiment 1

Experiment 1 uses our models to predict whether a placement is routable for a fixed utilization, aspect ratio, clock period and BEOL stack. We test our models on completely unseen data points from new designs, for both 2D and 3D ICs. Table 3.12 shows the DoE used to obtain ground truth for testing our 2DIC models in 28nm FDSOI, 28nm LP and 45nm GS technologies. We use new designs such as *Cortex M0*, *jpeg_x5* and *leon3mp*, as well as eight- and seven-layer BEOL stacks that we did not use for training. These make the classification problem more difficult and help assess whether our models are generalizable and scalable. We eliminate tool noise in the same manner as described for the training dataset, i.e., we execute six P&R runs for each point in the DoE by perturbing the clock period (by keeping utilization fixed) and utilization (by keeping clock period fixed) values. Table 3.13 shows the DoE used to obtain ground truth for testing our 3DIC models in 28nm FDSOI and 28nm LP technologies.

We use standard classification metrics to assess our training and test classifications such as accuracy, precision, recall and negative predictive value (NPV). We use confusion matrices to illustrate classification performed by our models on training and test data points. Accuracy is defined as the ratio of sum of true positives (TPs) and true negatives (TNs) to the sum of all data points used for classification (either for training or testing). Precision is defined as the ratio of TPs to the sum of TPs and false positives (FPs); recall is defined as the ratio of TPs to the sum of TPs and false negatives (FNs); and NPV is defined as the ratio of TNs to the sum of TNs and FNs.

Table 3.14 shows the confusion matrices of our predictions for 2DICs in 28nm FDSOI, 28nm LP and 45nm GS by using parameters listed in Section 3.3.1. For each technology, we use the data points from Table 3.10 for training, and the data points from Table 3.12 for testing.

Table 3.12: Design of experiments used to obtain ground truth for testing our 2DIC model.

		Cortex M0	*jpeg_x5*	*leon3mp*
Syn Clk Period (ns)	28FDSOI, 8T	0.8	1.0	1.0
	28LP, 12T	0.8	1.0	–
	45GS, 9T	1.2	1.5	–
#Instances	28FDSOI, 8T	9282	111342	442734
	28LP, 12T	9380	111463	–
	45GS, 9T	13601	168310	–
#FFs	28FDSOI, 8T	840	23560	108817
	28LP, 12T	840	23560	–
	45GS, 9T	840	23560	–
P&R Clk Period (ns)	28FDSOI, 8T	{0.8, 1.0, 1.5, 2.0}	{1.3, 1.5}	{1.5, 2.0}
	28LP, 12T	{0.8, 1.0, 1.5, 2.0}	{1.3, 1.5}	–
	45GS, 9T	{1.5, 2.0, 2.2}	{1.5, 2.0}	–
Util (%)	28FDSOI, 8T	76, ..., 90	76, ..., 90	76, ..., 90
	28LP, 12T	76, ..., 90	76, ..., 90	–
	45GS, 9T	76, ..., 90	76, ..., 90	–
Aspect Ratio	28FDSOI, 8T	{1.0, 1.8, 2.0, 2.2}	{1.0, 1.2, 1.5, 2.1}	{1.0, 1.2}
	28LP, 12T	{1.0, 1.5, 1.7}	{1.0, 1.1, 1.6}	–
	45GS, 9T	{1.0, 1.5, 1.7}	{1.0, 1.3, 2.0}	–
#Metal Layers	28FDSOI, 8T	{8, ..., 4}	{8, ..., 4}	{8, ..., 4}
	28LP, 12T	{6, 5, 4}	{6, 5, 4}	–
	45GS, 9T	{6, 5, 4}	{6, 5, 4}	–
Grid Size (#tracks)	28FDSOI, 8T	45×45	45×45	45×45
	28LP, 12T	45×45	45×45	–
	45GS, 9T	45×45	45×45	–
#Routable	28FDSOI, 8T	900	502	195
	28LP, 12T	508	246	–
	45GS, 9T	277	195	–
#Unroutable	28FDSOI, 8T	300	98	105
	28LP, 12T	32	24	–
	45GS, 9T	128	75	–

Table 3.13: Design of experiments used to obtain ground truth for testing our 3DIC model.

		Cortex M0	*leon3mp*
Syn Clk Period (ns)	28FDSOI, 8T	0.8	3.0
	28LP, 12T	0.9	3.2
#Instances	28FDSOI, 8T	9280	442736
	28LP, 12T	9387	442887
#FFs	28FDSOI, 8T	840	108817
	28LP, 12T	840	108817
P&R Clk Period (ns)	28FDSOI, 8T	{0.8, 1.0, 1.3}	{3.0, 3.3, 3.8}
	28LP, 12T	{0.9, 0.95, 1.15}	{3.2, 3.8, 4.0}
Util (%)	28FDSOI, 8T	80, …, 90	80, …, 90
	28LP, 12T	80, …, 90	80, …, 90
Aspect Ratio	28FDSOI, 8T	{1.0, 1.5, 2.0}	{1.0, 1.5, 2.0}
	28LP, 12T	{1.0, 1.5, 2.0}	{1.0, 1.5, 2.0}
#Metal Layers (Tier 0, Tier 1)	28FDSOI, 8T	{(6, 6), (5, 5), (4, 4)	{(6, 6), (5, 5), (4, 4)
	28LP, 12T	(4, 5), (5, 4)}	(4, 5), (5, 4)}
Grid Size (#tracks)	28FDSOI, 8T	45 × 45	45 × 45
	28LP, 12T	45 × 45	45 × 45
#Routable	28FDSOI, 8T	387	450
	28LP, 12T	463	438
#Unroutable	28FDSOI, 8T	108	45
	28LP, 12T	32	12

"True" refers to a placement being routable, that is, having a label "+1", and "False" refers to a placement being unroutable, that is, having a label "-1".

Table 3.15 shows error metrics (i.e., accuracy, precision, recall and NPV) for training and test datasets in 28nm FDSOI, 28nm LP and 45nm GS. We observe that the accuracy values are $\geq 85.9\%$ in the test dataset, that is, our models are able to classify placements as routable accurately and generalize to unknown and unseen data points. The accuracy is 90% for 28nm LP because the prediction problem is less difficult than the other two technologies as the number of unroutable placements are few. Across all technologies and designs, our precision is $\geq 90\%$ and recall is $\geq 86\%$ which indicates that our models accurately identify the unroutable placements. That is, there are few false positives and few false negatives in the classification results of the test dataset. The modeling problem in 28nm LP is relatively easy because only 7% of the data points are unroutable, that is, the training data are biased towards the routable label of "+1". However, our 28nm LP models do not overfit the routable data points and are able to identify five out of 57 unroutable placements correctly in spite of the bias in the training data. Even though the size of our test dataset is $1.5\times$ the size of our training dataset, our classification accuracy only degrades from 97.0% in the training dataset to 85.9% in the test dataset in 45nm GS.[29]

Tables 3.16 and 3.17 show the confusion matrices and error metrics by using only congestion maps from placements, that are typically used by physical design engineers to predict routability. NPV is a measure of how accurately a model can predict unroutable placements. In other words, NPV measures the ratio of placements that are truly unroutable to the placements that are predicted to be unroutable.[30] The overhead of incorrectly classifying a routable placement as unroutable is high, as design turnaround time increases and quality of results worsen. In Section 3.3, we illustrated the poor correlation of #DRCs with placement congestion maps; now we quantify the error across technologies and designs. We observe that accuracy for 28nm LP placements is 73.5% with NPV of 13.7% in the test dataset, whereas by using our new parameters the accuracy is 90.4% and NPV is 41.4% for the same test dataset. This shows that our new parameters enable accurate modeling.

Table 3.18 shows the confusion matrices of our predictions for 3DICs in 28nm FDSOI and 28nm LP. Table 3.19 shows error metrics for training and test datasets. We observe that

[29]To test robustness of our conclusions, we performed modeling by interchanging the training and test datasets. We trained our models using DoE data points from Table 3.12 and tested the models on data points from Table 3.10. In the training dataset, the worst-case differences in accuracy is 2.4%, precision is 1.1%, recall is 1.6% and NPV is 4.4%, across all technologies. In the test dataset, the worst-case differences in accuracy is 3.3%, precision is 1.6%, recall is 2.4% and NPV is 9.7%.

[30]For example, if every placement is always predicted to be routable, then NPV will be zero.

Table 3.14: Confusion matrices for 2DIC routability prediction for training and test datasets.

		Training				Testing		
			Actual				Actual	
			True	False			True	False
28nm FDSOI	Pred	True	869	11		True	1436	112
		False	37	460		False	161	391
28nm LP	Pred	True	829	1		True	682	5
		False	32	56		False	72	51
45nm GS	Pred	True	612	11		True	406	29
		False	16	279		False	66	174

Table 3.15: Classification error metrics for 2DIC training and test datasets.

	Dataset	Accuracy (%)	Precision (%)	Recall (%)	NPV (%)
28nm FDSOI	Training	96.5	98.8	95.9	92.5
	Testing	87.0	92.7	89.9	70.8
28nm LP	Training	96.4	99.8	96.3	63.6
	Testing	90.4	99.2	90.4	41.4
45nm GS	Training	97.0	98.2	97.4	94.6
	Testing	85.9	93.3	86.0	72.5

Table 3.16: Confusion matrices for 2DIC routability prediction by using congestion map only for training and test datasets.

		Training				Testing		
			Actual				Actual	
			True	False			True	False
28nm FDSOI	Pred	True	833	66		True	1076	283
		False	73	405		False	521	220
28nm LP	Pred	True	790	11		True	565	26
		False	71	46		False	189	30
45nm GS	Pred	True	586	47		True	334	89
		False	42	243		False	138	114

Table 3.17: Classification error metrics by using congestion map only for 2DIC training and test datasets.

	Dataset	Accuracy (%)	Precision (%)	Recall (%)	NPV (%)
28nm FDSOI	Training	89.9	92.6	91.9	84.7
	Testing	61.7	79.2	67.4	29.7
28nm LP	Training	91.1	98.6	91.7	39.3
	Testing	73.5	95.6	74.9	13.7
45nm GS	Training	90.3	92.6	93.3	85.3
	Testing	66.3	78.9	70.8	45.2

Table 3.18: Confusion matrices for 3DIC routability prediction for training and test datasets.

		Training				Testing		
			Actual				Actual	
			True	False			True	False
28nm FDSOI	Pred	True	1702	17		True	677	51
		False	60	381		False	90	102
28nm LP	Pred	True	1976	16		True	774	18
		False	79	89		False	127	26

Table 3.19: Classification error metrics for 3DIC training and test datasets.

	Dataset	Accuracy (%)	Precision (%)	Recall (%)	NPV (%)
28nm FDSOI	Training	96.4	96.6	99.0	95.7
	Testing	84.7	88.3	93.0	66.7
28nm LP	Training	95.6	96.2	99.2	84.8
	Testing	84.7	85.9	97.7	59.1

the accuracy values are $\geq 84.7\%$ in the test dataset, that is, our models are able to accurately classify placements as routable and generalize to unknown and unseen data points. Across both technologies and designs, our precision is $\geq 86\%$ and recall is $\geq 93\%$ which confirms that our models accurately identify the unroutable placements. That is, there are few false positives and few false negatives in the classification results of the test dataset.

Results of Experiment 2

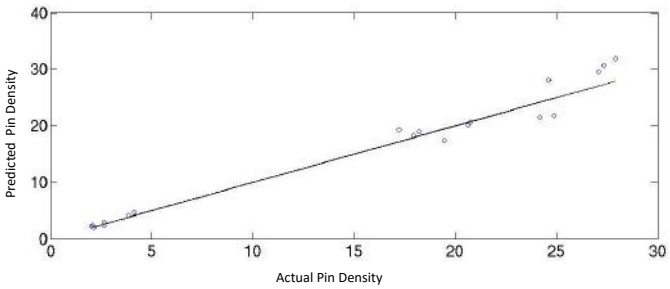
In this experiment, our goal is to determine the iso-performance Pareto frontiers of utilization, aspect ratio and number of metal layers for various designs using our models. We predict Pareto frontiers for both 2D and 3D ICs. Our models can predict "iso-performance" because we comprehend clock period and timing-related parameters in our modeling, and our results are actually different from "performance-oblivious" models. We are given only few placements, so we must interpolate and extrapolate our modeling parameters from these placements to predict the Pareto frontiers. This is very challenging because the metrics do not scale in a known manner (e.g., unimodal, linear, etc.) when utilization, aspect ratio and the BEOL stack are changed. Sometimes the P&R tools stop fixing timing or congestion violations at the placement stage when the utilization is very tight or the BEOL stack has insufficient number of metal layers. To overcome these challenges, we devise an interpolation and extrapolation method as described in Section 3.3.1 using machine learning. In the following, we describe Pareto frontier predictions first for 2DICs and then for 3DICs.

For Pareto frontier prediction of 2DICs, our testing dataset in each technology contains around 100–300 implementations of *Cortex M0* and *jpeg_x5* designs that span different utilizations, aspect ratio values and BEOL stack. We choose 20 of these placements per design that are implemented with the smallest clock period for these designs from Table 3.12.[31] We then execute our method in Section 3.3.1. Obtaining 20 placements is inexpensive – especially, relative to the cost of a failed routing job or wasting area or wafer cost – from both CPU and wall time standpoints. Note that we use designs that are not used for training to create the Pareto frontiers, and we use only information from the placement, that is, no information from routing.[32]

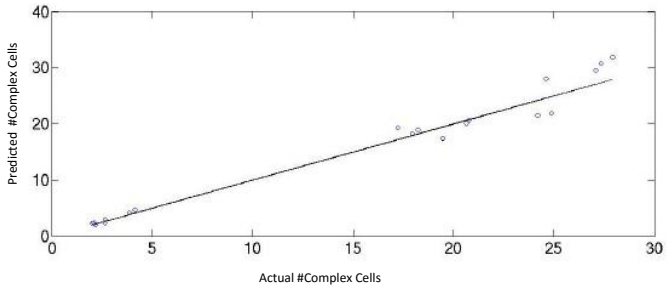
We set the error upper bound UB_{error} for each metric to be 20%. We use cubic splines for the MARS technique and use grid search to determine the best values of hyperparameters (e.g., the SVM regularization hyperparameter C, error margin ξ and RBF weight for each radius γ [85]) for SVM with RBF kernel. We create one model for each parameter, e.g., {max, average} $\times$ {pin density, #complex cells, sum of incoming and outgoing edges}, etc. as described in Section 3.3.1. Figures 3.43(a) and (b) compare average pin density and average #complex cells, respectively for the 20 placements of *jpeg_x5* in 28nm FDSOI when using our models to interpolate or extrapolate parameter values and actual values obtained from the placements. We

[31] For example, in 28nm FDSOI we choose utilizations {80, 82, 83, 84, 85}% for both designs; the corresponding numbers of metal layers for these utilizations are {4, 4, 5, 5, 6}. For *Cortex M0*, we use aspect ratios {1.0, 1.8, 2.0, 2.2}, and for *jpeg_x5* we use aspect ratios {1.0, 1.2, 1.5, 2.1}.

[32] Only the ground truth of the Pareto frontiers is obtained from actual routing information of these placements.



(a)



(b)

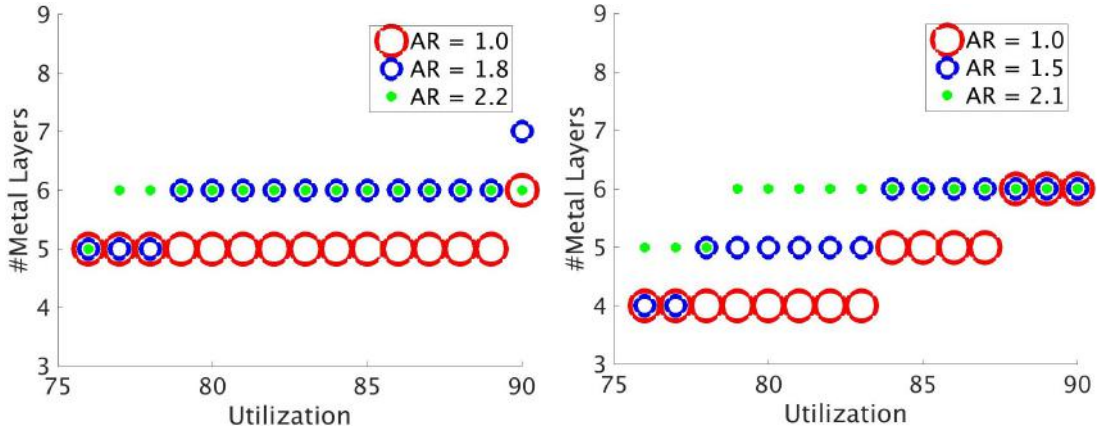
Figure 3.43: Prediction accuracy of our interpolation and extrapolation method in 28nm FDSOI for *jpeg_x5* for average (a) pin density and (b) #complex cells.

then create a test dataset using estimates from the models of each parameter, and use our models (developed using the DoE for training dataset in Table 3.10) to predict routability.

Figures 3.44(a) and (b) show the predicted Pareto frontiers of #metal layers, utilizations and aspect ratios for *Cortex M0* and *jpeg_x5*, respectively in 28nm FDSOI. Figures 3.45(a) and (b) show the predicted Pareto frontiers of #metal layers, utilizations and aspect ratios for *Cortex M0* and *jpeg_x5*, respectively in 45nm GS. Figures 3.46(a) and (b) show the ground truth Pareto frontiers of #metal layers, utilizations and aspect ratios, respectively for *Cortex M0* in 28nm FDSOI and *jpeg_x5* in 45nm GS.[33] From Figures 3.44(a) and 3.46(a), we observe that in 28nm FDSOI *Cortex M0* is routable with five metal layers when aspect ratio is 1.8 and utilization is 79%, but our model predicts that the maximum utilization in 78% (i.e., 79% requires six metal layers). Similarly, from Figures 3.45(b) and 3.46(b), in 45nm GS *jpeg_x5* is routable with four metal layers when aspect ratio is 1.5 and utilization is 77%, but our model predicts that no placement of *jpeg_x5* is routable with four metal layers at aspect ratio 1.5. Across three foundry technologies and two designs (that were not used for training), our predictions of maximum achievable utilization are within 2% of the maximum achievable utilization value in the ground

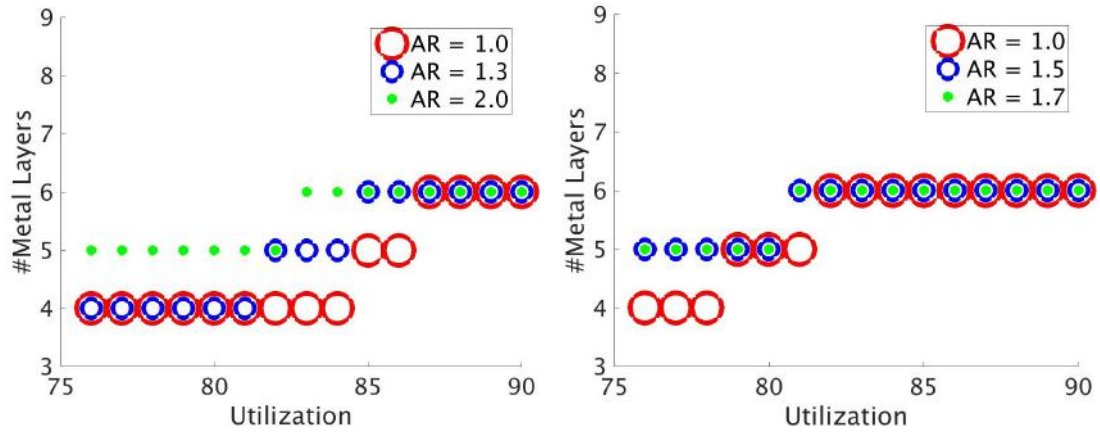
[33]The two designs show limited value from the M5 layer because beyond 82% utilization, both horizontal and vertical routing tracks are required for routability. Adding only M5 does not cure the routability issues.

truth.



(a) (b)

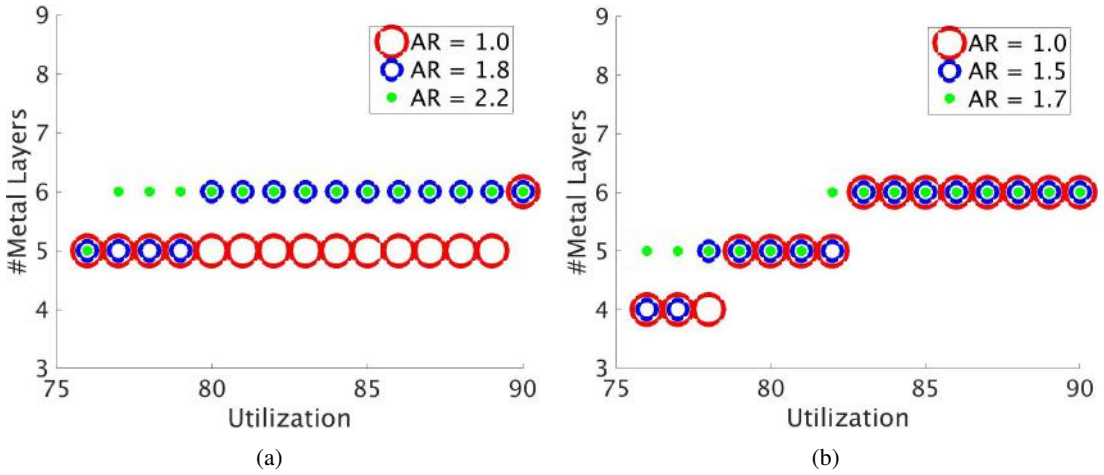
Figure 3.44: Pareto frontiers of #Metal Layers (y-axis) versus utilization (x-axis) for multiple aspect ratios using our models in 28nm FDSOI: (a) *Cortex M0* and (b) *jpeg_x5*.



(a) (b)

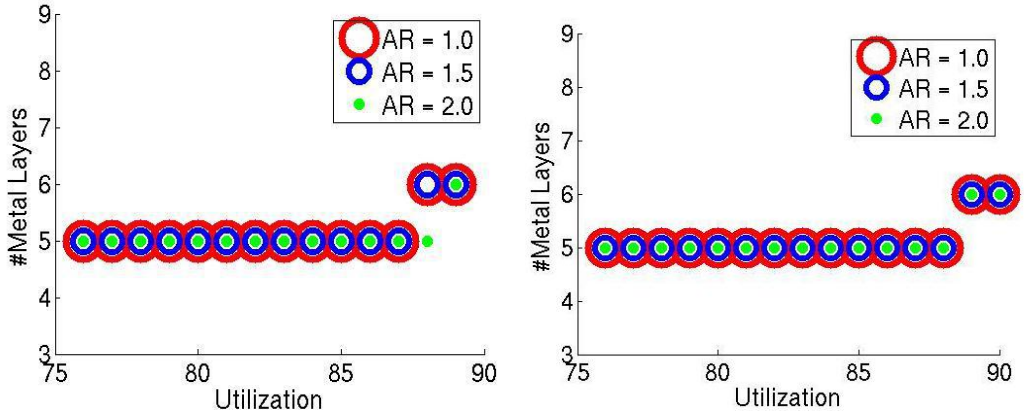
Figure 3.45: Pareto frontiers of #Metal Layers (y-axis) versus utilization (x-axis) for multiple aspect ratios in 45nm GS: (a) *Cortex M0* and (b) *jpeg_x5*.

For Pareto frontier prediction of 3DICs, we use the A3D flow (described in Section 5.1) as our 3D flow. Figures 3.47(a) and (b) show the ground truth and predicted Pareto frontiers, respectively of #metal layers, utilizations and aspect ratios for *Cortex M0* in 28nm FDSOI. With aspect ratio of 2.0 and five metal layers, the ground truth indicates that the maximum achievable utilization is 89%, while, the model prediction is 88%. Figures 3.48(a) and (b) show the ground truth and predicted Pareto frontiers, respectively of #metal layers, utilizations and aspect ratios for Tier 0 of *leon3mp* in 28nm FDSOI. With aspect ratio of 1.0 and six metal layers, the ground



(a) (b)

Figure 3.46: Ground truth Pareto frontiers of #Metal Layers (y-axis) versus utilization (x-axis) for multiple aspect ratios of (a) *Cortex M0* in 28nm FDSOI and (b) *jpeg_x5* in 45nm GS.



(a) (b)

Figure 3.47: Pareto frontiers of #Metal Layers (y-axis) versus utilization (x-axis) for multiple aspect ratios of Tier 0 of *Cortex M0* in 28nm FDSOI: (a) ground truth and (b) model predictions.

truth indicates that the maximum achievable utilization is 90%, while, the model prediction is 88%. These results show that our model predictions are pessimistic by 2% of the maximum achievable utilization in the ground truth. Our results for Tier 1 are similar to those of Tier 0, i.e., our model predictions are pessimistic by 1% of the maximum achievable utilization in the ground truth.

We have also conducted the same 3DIC experiments as above using the "shrunk2D" (S2D) flow of [196]. Figures 3.49(a) and (b) show the ground truth and predicted Pareto frontiers, respectively of #metal layers, utilizations and aspect ratios for *aes_cipher_top* in 28nm FDSOI. With aspect ratio of 1.0 and five metal layers, the ground truth indicates that the maximum

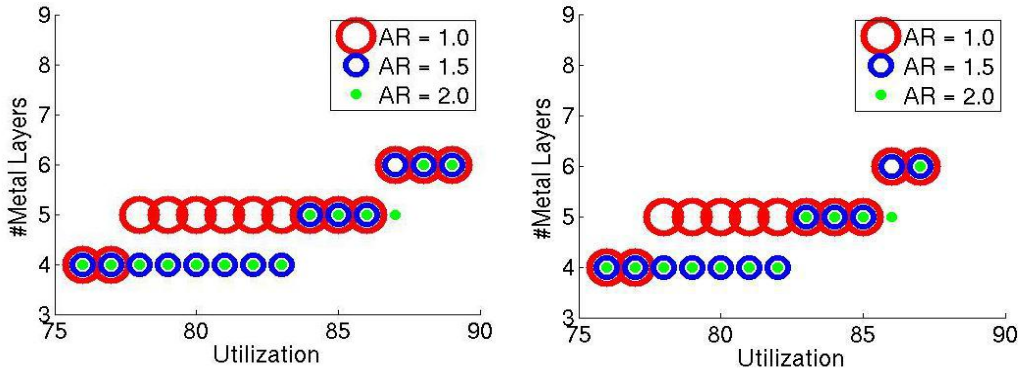


Figure 3.48: Pareto frontiers of #Metal Layers (y-axis) versus utilization (x-axis) for multiple aspect ratios of Tier 0 of *leon3mp* in 28nm FDSOI: (a) ground truth and (b) model predictions.

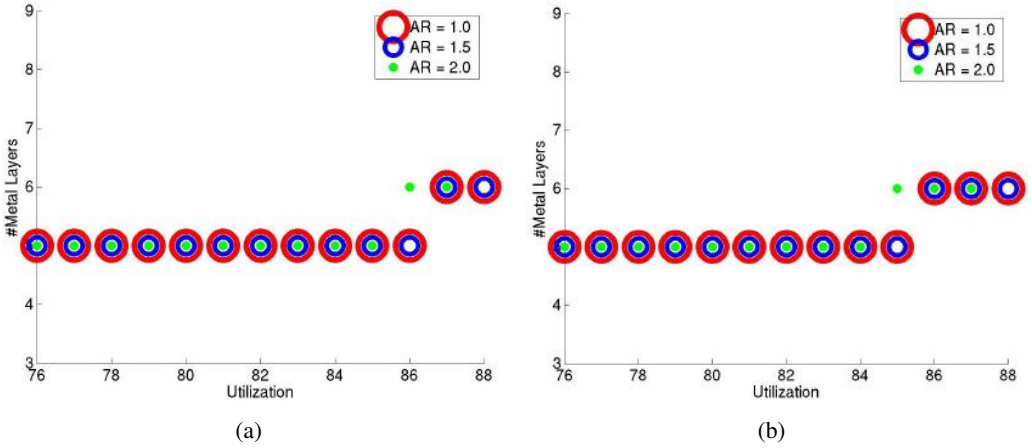


Figure 3.49: Pareto frontiers of #Metal Layers (y-axis) versus utilization (x-axis) for multiple aspect ratios of Tier 0 of *aes_cipher_top* in 28nm FDSOI with S2D flow of [196]: (a) ground truth and (b) model predictions.

achievable utilization is 86%, while, the model prediction is 85%. Figures 3.50(a) and (b) show the ground truth and predicted Pareto frontiers, respectively of #metal layers, utilizations and aspect ratios for Tier 0 of *leon3mp* in 28nm FDSOI. With aspect ratio of 1.5 and five metal layers, the ground truth indicates that the maximum achievable utilization is 85%, while, the model prediction is 83%. Again, these results show that our model predictions are pessimistic by 2% of the maximum achievable utilization in the ground truth. Our results for Tier 1 are similar to those of Tier 0, i.e., our model predictions are pessimistic by 2% of the maximum achievable utilization in the ground truth.

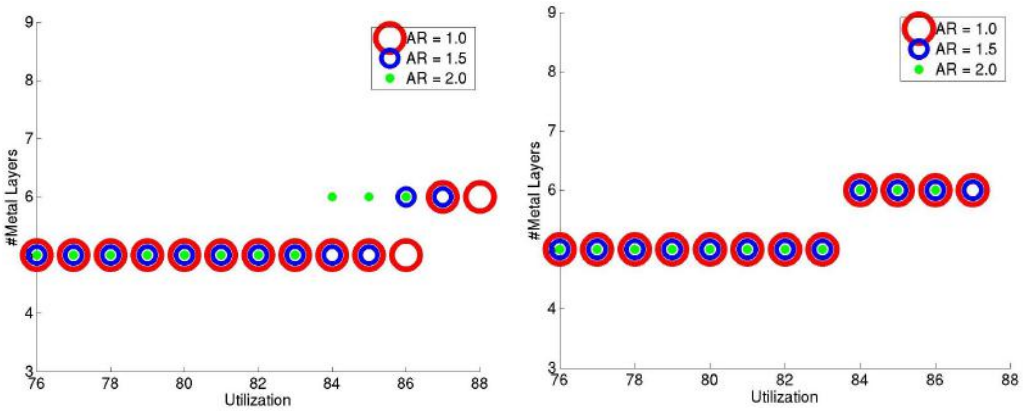


Figure 3.50: Pareto frontiers of #Metal Layers (y-axis) versus utilization (x-axis) for multiple aspect ratios of Tier 0 of *leon3mp* in 28nm FDSOI: (a) ground truth and (b) model predictions.

3.3.3 Conclusions

Efficient exploration of the space of utilization, aspect ratio and BEOL stack at iso-performance is very important for design turnaround time and to achieve good quality of results. Currently, physical design engineers use congestion maps of P&R tools from the placement stage to predict routability as measured by the #DRCs. However, our experimental results indicate that these maps can sometimes be misleading and inaccurate in predicting routability. We present new modeling parameters that enable us to analyze local hotspots in a placement and achieve accurate predictions of routability. We also present a new method of using only a few placements to predict (using our models) the Pareto frontiers of utilizations, aspect ratios and BEOL stacks at iso-performance. We demonstrate that our modeling methodology is applicable to both 2D and 3D ICs. Our experimental results indicate that our predictions are pessimistic by 2% of the maximum achievable utilization across 2D and 3D ICs, three different foundry technologies and two designs (that were not used for training). Overall, our classification accuracies for 2DICs are 87.0% in 28nm FDSOI, 90.4% in 28nm LP and 85.9% in 45nm GS. We achieve significant improvements as compared to using only congestion maps; classification accuracies for 2DICs by using only congestion maps are 61.7% in 28nm FDSOI, 73.5% in 28nm LP and 66.3% in 45nm GS. Future work might include (i) prediction of timing and routability using heterogeneous pitches of metal layers in the BEOL stack, and (ii) confidence intervals of our routability predictions.

3.4 3DIC Benefit Estimation and Implementation Guidance From 2DIC Implementation

As the semiconductor industry nears the end of the CMOS roadmap, product-level benefits from successive technology nodes have decreased due to reliability, variability, power and thermal constraints. It has been noted above that three-dimensional integrated circuits (3DICs) have emerged as a promising solution to extend both the use of today's device and process technologies, as well as the historical Moore's-Law trajectory of value scaling. Eventual cost benefits of 3DIC have yet to be quantified in a mature supply chain and high-volume production context. However, one consensus value proposition for 3DIC has emerged across both industry and academia, namely, *power reduction* benefits (with implied reliability, cost, and user experience benefits) due to shorter connections that are simply unachievable with 2D integration.

Current 3DICs are based on through-silicon vias (TSVs), but integration density is limited by the pitch of TSVs, with mass production focusing on memory-on-logic designs with relatively few vertical connections [193]. Two emerging alternatives to TSV-based 3D integration are (i) sequential face-to-back (F2B) and (ii) fine-grain face-to-face (F2F) integration technologies. They enable orders of magnitude higher integration density compared to that of TSV-based technology, due to the extremely small size of inter-tier (vertical) vias. For example, CEA-LETI [18] has pursued a sequential 3D integration using a low-temperature bonding process. Recent 3DIC design literature [196] [103] has explored implications of fine-grain F2F integration process. Figure 3.51 illustrates sequential F2B and fine-grain F2F 3DIC integration.

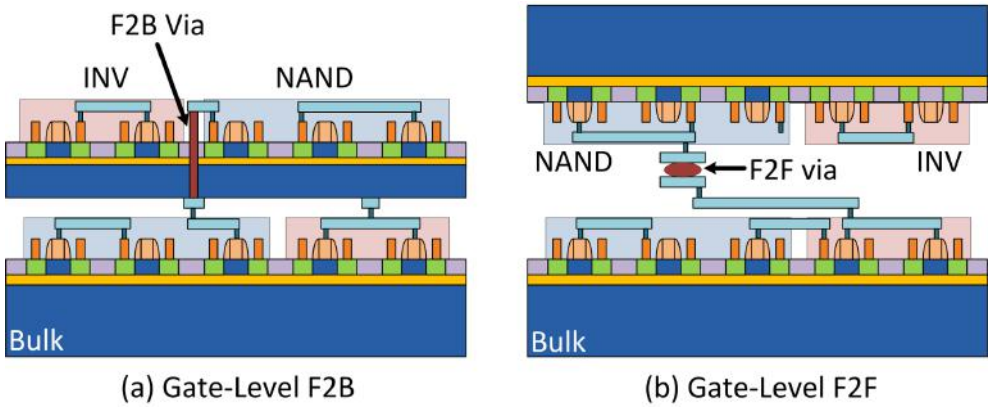


Figure 3.51: 3D integration: gate-level (a) F2B and (b) F2F [196].

To perform fast and accurate *implementation-space exploration* (ISE), sometimes referred to as *pathfinding* [175], chip architects and designers require accurate 3D power estima-

tion tools. This is especially critical with power-centric 3DIC value propositions. Unfortunately, power estimation of 3D implementations is challenging because (i) 3D benefit varies with netlist topologies, constraints and implementation styles, and (ii) there are no "golden" 3D implementation flows. To our knowledge, no tool today can accurately predict the power benefit of 3D implementation based on netlist, constraints, and whatever information might be available from 2D implementation. The lack of such an estimation tool results in a large number of iterations (often, not much better than "throwing darts") to identify the best set of implementation parameters and/or constraints for 3D implementation. Only after making many attempts in this manner can the designer gain an inkling of potential 3D implementation benefits for a given block. As reviewed in Section 2.1.2, no tool currently exists that allows fast and accurate ISE for 3DICs.

In this work, we overcome the above challenges by developing an efficient 3D power estimation methodology, along with an accurate *3D Power Estimation* (3DPE) prediction tool. 3DPE predicts *benefit*, i.e., the "delta" in power (= reduction from 2D implementation) that will be achieved by a given 3D flow. We experimentally confirm that 3DPE can estimate 3DIC power *reduction* with error of $\leq$10% across a set of testcases implemented in foundry 28nm FDSOI technology.

Our 3DPE model development includes a novel exploitation of sensitivities of post-synthesis and post-place-and-route (SP&R) power to wireload model (WLM) and capacitance scaling; this yields new parameters that increase modeling accuracy.[34] We also perform a novel *stress test* of 3DPE by verifying that the model cannot produce unreasonable values of estimated 3D power benefit. While practitioners have struggled with a gap between theoretical limits of 3DIC benefit and observed benefits, our model stress test provides some encouragement in the form of model parameter combinations that suggest potential large 3DIC power benefits. Additional experimental studies confirm the usability of 3DPE in *model-guided implementation* (MGI), e.g., for a given design and set of constraints, 3DPE can identify wireload model scaling, floorplan aspect ratio, target utilization, etc. settings in commercial SP&R flows to obtain minimum power in the final 3D implementation. We believe that the resulting modeling capability can be used for both ISE and MGI across architectural and physical implementation levels of design. We summarize our main contributions as follows.

- To our knowledge, we are the first to develop an estimation tool that focuses on the 3DIC value proposition of 3D power benefit. Our 3DPE model is achieved with *bounded error*

[34] Our models are based on the sensitivities of power to constraints and design parameters (e.g., mix of threshold voltage types and wirelength) between 2D and 3D implementations of the same designs, and are specific to the flow from [196]. The models must be rederived if the tool flow or technology changes.

machine learning techniques; it predicts delta power benefit of 3DIC with average (resp. maximum) error of $\leq$0.1% ($\leq$10%) based on netlist, design constraints and 2D implementation metrics.

- We develop novel estimation model parameters based on the sensitivity of synthesis and P&R outcomes to wireload model and capacitance model scaling. This is a heretofore unexplored approach to assessing how RTL and gate-level netlists will react to 3D vs. 2D implementation contexts.

- We propose novel validations of our 3DPE model, including (i) a "stress test" approach to verify that no unreasonable values of predicted power benefit can occur, and (ii) application of 3DPE in *model-guided implementation*.

3.4.1 Baseline 2D, Shrunk2D and 3D Flows

As mentioned above, the 3DIC implementation flows of [196] [197] are currently published in the research literature. To develop our 3D power estimation model, we use the "shrunk2D" (S2D) and 3D flows from [196] as proxies for golden 3DIC implementation. Through extensive interactions with the flow developer [194], we have transplanted the entire flow enablement (including EDA tool versions and PDK versions) and successfully replicated published results. We have subsequently made several automation-centered flow enhancements: automated floorplan adjustment to handle multiple block aspect ratios (ARs); AR- and perimeter-aware pitch selection and placement for pins; instantiation of memories specifically generated from foundry 28nm FDSOI technology enablement, with relative placements that scale with block AR; and unified flow and configuration files to enable automation across multiple small and large testcases. Furthermore, we automate parameter sweeps: clock period, capacitance scaling factor, Vt types, transparent use of F2F/F2B configuration, aspect ratio, target utilizations, and design rule constraints (maximum cap load, maximum transition time, etc.)

As described in Section 5.1, we have developed an analytic placement tool, *APlace3D*, based on the source code of APlace2D [113] [114], which implements a new "true 3D" objective. We show below that an *APlace3D* (*A3D*) flow can also be used to develop a 3D power estimation model.

Table 3.20: Testcases and their post-synthesis results.

Testcase Type	Design Name	Number of Instances	Clock Period (ns)	% #Buffers/ #cells	% #FFs/ #cells
GPU	THEIA	212K	1.6	20	8
CPU	*OST2 spc* (1-core)	347K	1.6	16	22
Modem	*viterbi*	98K	1.0	26	27
Multimedia	*dct*	12K	1.0	33	6
Peripheral Engine (PE)	*aes* (crypto)	10K	0.9	22	5

3.4.2 Experimental Setup and Results

The open-source designs used in our experiments are described in Section 3.4.2. We describe our approaches to identify parameters with greatest influence on 3D power benefit in Section 3.4.2.

Floorplan and Implementation of Testcases

We use a wide range of IPs as our testcases that include building blocks for a modern mobile SoC. The building blocks could be classified into CPU, GPU, modem, multimedia, and peripheral engines. For each class among , we use IPs from OpenCores [318] in which the number of instances in these designs range from 10K to 347K.[35] Table 3.20 summarizes the synthesis results for these testcases. The percentage of buffers from all the cells ranges from 15% to 33% and the percentage of flip-flops ranges from 5% to 27%.

As CPUs and GPUs are two key components in mobile SoCs, we use CPU- and GPU-like designs with various memory sizes, shapes, we implement the *OpenSPARC T2* (OST2) core [317] *spc* and *THEIA* GPU [318] testcases in foundry 28nm FDSOI technology. We overcome the lack of customized memory sizes and number of read/write ports by choosing memories with closest word sizes and word numbers, from foundry 28nm memory libraries that cover the required word sizes and word numbers. To emulate the effects of lower capacitance and wirelength in 3D, we "engineer" wireload models (WLMs) along with other design parameters to assess the sensitivity of 3DPE models to WLMs and these parameters.

To assess the sensitivity of 3DPE models to floorplan aspect ratios (AR), we implement testcases with AR ranging from 0.8 to 1.2. Given a fixed die area, our memory placement methodology is able to automatically place memory blocks with any floorplan AR in this range as

[35]The P&R runtime of each 2D or 3D run is 16 hours for *spc*, five hours for *THEIA* and *dec_viterbi* (*viterbi*), and two hours for *aes_cipher_top* (*aes*) and *dct* when using two threads on a Xeon E5-2640 server with 128GB memory.

described in Algorithm 2. Initially, we generate a floorplan with AR = 1.0 and cluster memories into four groups.[36] We then place these groups at four corners of the die area (Line 1). When the AR changes, we calculate the coordinates of the four corners of the modified die area, and adjust the placement of each memory group accordingly (Lines 6–10). When there are overlaps between groups due to AR being too large (or too small), we re-cluster the memories so as to remove the overlaps (Lines 11–13).

The clustering honors certain basic constraints, e.g., memories are placed face-to-face with respect to each other and each pair has routing channels in between them. We insert at least 5μm routing channels in between memories, and apply placement halos to these channels to avoid routing congestion. In 3D, we use the flow in [196] to place memories based on the corresponding 2D floorplan. Figures 3.53(a) and (b) respectively show the floorplans of *spc* and *THEIA* in both 2D and 3D.

Algorithm 2 Floorplan scaling with memories

Procedure $genFloorPlan$
Inputs : AR_list, $area_{sram}$, $area_{postsyn}$, $util$
Outputs : Coordinates of memories for different placement AR

1: Place memories with $AR = 1.0$, such that four memory clusters ($Cluster_{BL}$, $Cluster_{BR}$, $Cluster_{TL}$, $Cluster_{TR}$) are at four corners (BL, BR, TL, TR) of the die
2: $area = (area_{postsyn}/util + area_{sram})$
3: $x_{orig} = \sqrt{area}$
4: $y_{orig} = \sqrt{area}$
5: **for each** $AR \in AR\_list$ **do**
6: $x = \sqrt{area \times AR}$
7: $y = \sqrt{area/AR)}$
8: Move $Cluster_{BR}$ by $(x - x_{orig}, 0)$
9: Move $Cluster_{TL}$ by $(0, y - y_{orig})$
10: Move $Cluster_{TR}$ by $(x - x_{orig}, y - y_{orig})$
11: **if** There are overlapped memories **then**
12: Re-cluster memories to remove overlaps
13: **end if**
14: **end for**

Parameter Identification

We use the S2D [196] and A3D flows to sweep parameter values shown in Table 4.6 and generate the training and testing datasets. Since S2D is a proxy for 3D, the difference in power between S2D and 3D is shown to be <5% in [196]. We confirm that this observation is still true after our modifications described in Section 3.4.1. Figure 3.54, which shows eight implementations of the *dec_viterbi* decoder across four categories I–IV in Table 3.22, confirms

[36] The memory clusters $Cluster_{BL}$, $Cluster_{BR}$, $Cluster_{TL}$ and $Cluster_{TR}$ are respectively {memories in *IFU*, *FGU*}, {memories in *MMU*}, {non-array part of memories in *LSU*}, and {array part of memories in *LSU*, memories in *EXU0*, *EXU1* and *TLU*} for *OST2*; and are respectively {memories in *CORE0*}, {memories in *CORE3*}, {memories in *CORE1*}, and {memories in *CORE2*} for *THEIA*.

that we can use S2D as a proxy for 3D implementations in our studies. We execute our design of experiments (DoE) for each testcase using the parameter values shown in Table 4.6. We run 2D, S2D and A3D implementations using these parameter values for each testcase and extract outcomes of various metrics such as the number of buffers, power, wirelength, cell area, etc. to generate training data points.

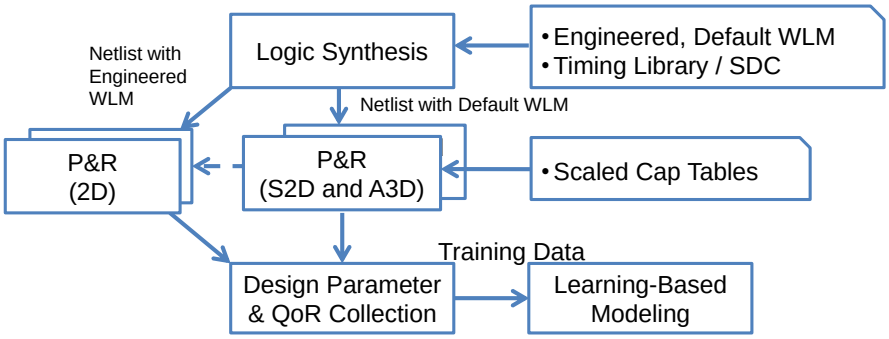


Figure 3.52: Our overall synthesis and implementation flow is based on the A3D and S2D flows of [41] and [196], respectively. We generate multiple "engineered" WLMs by scaling capacitance. Our learning-based models can identify 3D benefits by comprehending the change in WL with the change in capacitance between 2D and 3D implementations.

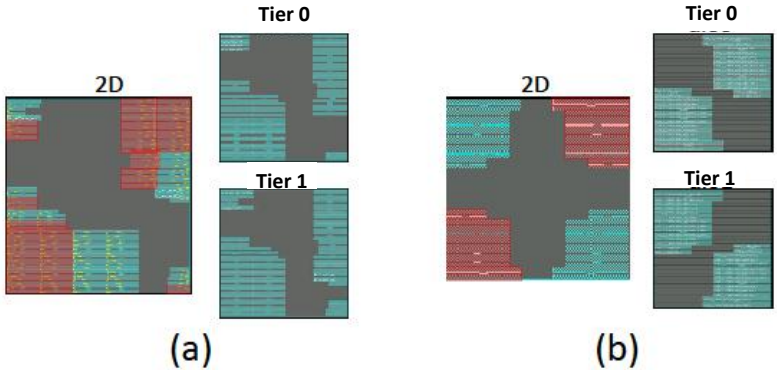


Figure 3.53: Floorplans of (a) *spc* and (b) *THEIA* (GPU) with AR = 1.0. The red-shadowed memories are partitioned to Tier 0.

Table 4.6 shows the key parameters that influence 3D implementation and can provide guidance in estimating the percentage delta power from a corresponding 2D implementation. Figure 3.55 illustrates the impact on 2D and 3D power for three of these parameters – utilization (UTIL), AR and max fanout (maxFO) – expressed in the figure in units of pF. Certain parameters such as maxCap do not have significant impact on 3D power, so we do not use these parameters in our modeling. To limit the number of dimensions in our models, we identify the top-10

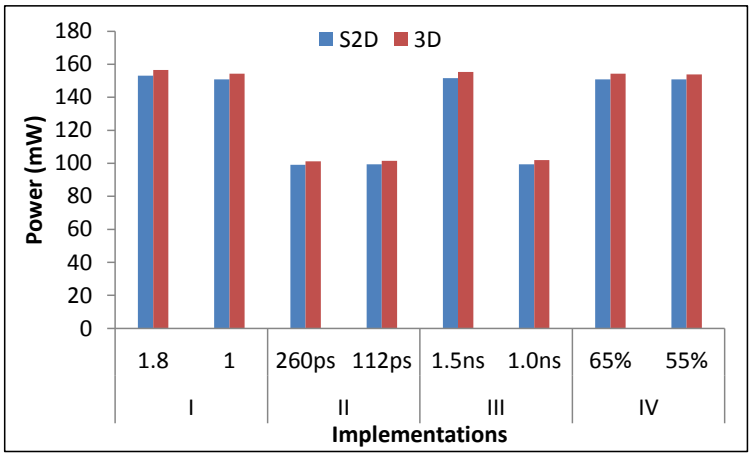


Figure 3.54: Comparison of power between S2D and 3D across eight implementations.

parameters based on how significantly these parameters affect percentage delta power. We summarize the percentage change in power with the minimum and maximum parameter values in Table 3.23. We explore sensitivities of power to capacitance scaling in our DoE by (i) running a 2D implementation with 0.7× capacitance scaling for all interconnects, and (ii) varying post-synthesis netlists by changing capacitance using wireload models. We correlate the change in metrics with and without capacitance scaling from both 2D P&R and logic synthesis as part of our model derivations.

The modeling problem of predicting percentage delta power in 3D by only observing metrics from a 2D implementation is nontrivial. Figure 3.55 as well as our experimental results indicate that 3DIC power is nonunimodal as well as nonmonotonic with different parameters. For example, a large change (i.e., $\sim$10×) in the number of buffers in 2D between scaled and non-scaled capacitances can lead to a relatively small (i.e., $<$10%) change in 3D power. Based on our experimental results, the parameters that affect delta power in 3D include WLM scaling, clock period, mix of Vt types, AR, UTIL, maxTran, maxFO, maxCKskew, maxCKlat and maxCKtran.

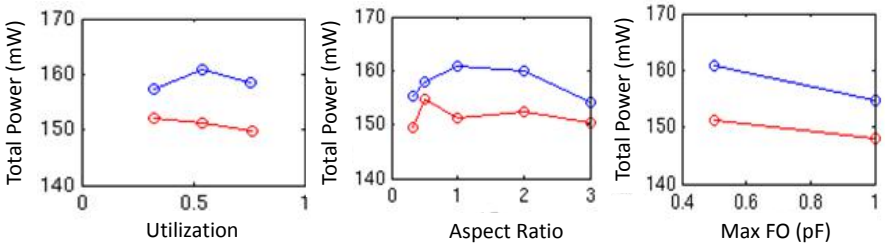


Figure 3.55: Illustration of implementation parameter impact on 2D (blue lines) and 3D power (red lines).

Table 3.21: Key parameters used by 3DPE models.

Design metric	Symbol	Description
Vt types	Vt	Mix of threshold voltage libraries = {RVT, RVT & LVT}
Aspect Ratio	AR	{1.0, 1.2, 1.5}
Utilization (%)	UTIL	{50%, 60%}
Max transition	maxTran	{50%, 70%} of max tran in library
Max capacitance	maxCap	{50%, 100%} of max cap in library
Max fanout	maxFO	{5, 10}
Max clock skew	maxCKskew	{0ps, 50ps}
Max clock latency	maxCKlat	{500ps, 2500ps}
Max clock transition	maxCKtran	{20%, 30%} of clock period
Corners	CORNER	PVT = {TT, 0.92V, 25^{o}C} Analysis = {setup}
WLM scaling	WLMSC	Capacitance scaling = {1.0, 0.7, 0.33}

Table 3.22: Implementations used in S2D vs. 3D comparisons.

Category	Clock Period	Util	AR	Max Cap	Max Tran	Max FO
I	1.0ns	65%	{1.8, 1.0}	450pF	260ps	10
II	1.5ns	65%	1.0	450pF	{260, 112}ps	10
III	{1.5, 1.0}ns	65%	1.0	450pF	112ps	10
IV	1.0ns	{65%, 55%}	1.0	450pF	260ps	10

Table 3.23: The percentage difference in power with the extreme points.

	UTIL	AR	maxCap	maxTran	maxFO
2D	2.2%	4.4%	3.0%	0.4%	4.0%
S2D	1.5%	3.4%	0.0%	0.6%	2.3%

Machine Learning Methodology

We develop a model to predict percentage delta power between 2D and 3D implementations of a given testcase. From the post-synthesis and post-P&R results of the testcase, we obtain the following parameters. (i) **Post-synthesis** – number of standard cells, number of buffers and inverters, area of standard cells, internal and leakage power of buffers[37] and non-buffer cells, and net switching power with capacitance in wireload models set to multiple values, and (ii) **Post-P&R** – number of standard cells, number of buffers and inverters, area of standard cells,

[37] In the following, we generically refer to buffers and inverters as "buffers".

internal and leakage power of buffers and non-buffer cells, net switching power with capacitance factor set to 1.0× and 0.7×, and wirelength.

The total power in 2D (resp. 3D (S2D and A3D)) implementations is the sum of internal, leakage and net switching power values. For each testcase, we calculate the total power in mW. The S2D implementations are used as a proxy for 3D implementation in our modeling. Table 3.24 shows the "ground truth" data for deltas in wirelength, number of buffers, and power for S2D versus 2D implementation according to our DoE and testcases (Sections 3.4.2 and 3.4.2). Table 4.6 shows examples of the range of values of our metrics. We create a total of three datasets for modeling – training, validation and testing. Out of all the data points we generate using 2D, S2D and A3D P&R flows, we use ∼40% for training, ∼20% for validation, and the remaining ∼40% for testing and reporting errors.

We use Artificial Neural Networks (ANN) as our modeling technique, via the in-built *MATLAB vR2013a* toolbox. We use nonlinear modeling because the percentage delta power is non-monotone with respect to the parameters. The complex interactions between parameters are determined automatically by the ANN technique using hidden layers and weights. The hyperparameters [85] we tune are the number of epochs and the number of neurons per hidden layer. We use two hidden layers – one for input and one for output. We vary the number of epochs from 1000 to 5000 in steps of 500 and the number of neurons per hidden layer from one to twice the number of modeling parameters k. We increase accuracy of our models by choosing appropriate hyperparameters such that the range of errors is within a bound. To achieve bounded errors, we search for the number of epochs and the number of neurons that satisfy the following two criteria: (i) the ratio of mean square errors (MSEs) in the training and the validation sets is $\leq$5, and (ii) add a large multiplicative penalty to suppress outliers (we use 1000×) whenever the range of errors is greater than the bound (we use ∼10% as our error bound and call these data points as outliers). We also perform five-fold cross-validation when training the model. Applying these criteria enables us to develop models that are not overfitted and can generalize to parameter values that are not present in the training dataset. Figure 3.56 illustrates our modeling flow, which is executed five times due to five-fold cross-validation.[38]

We now present our experimental studies. We discuss (i) accuracy results of our 3DPE tool , (ii) robustness and scalability of the 3DPE models, and (iii) model-guided implementation results.

[38]The runtime to train our models is four hours on an Intel Xeon E5-2640 2.5GHz server, using eight threads. This is a one-time cost.

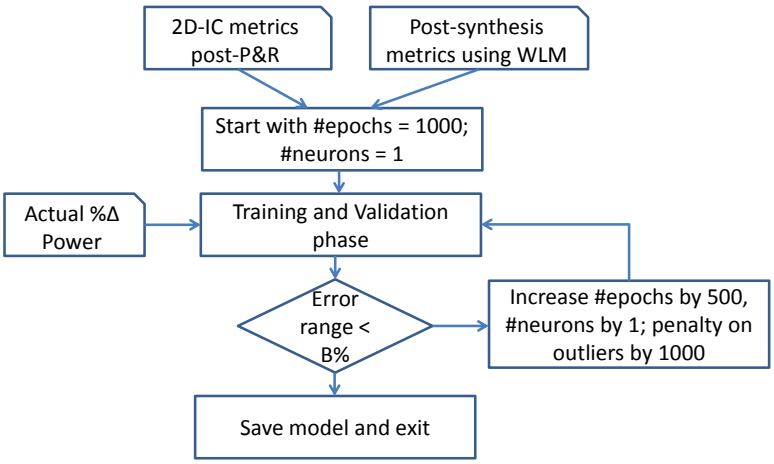


Figure 3.56: Illustration of our modeling flow.

Table 3.24: Experimental results of delta outcomes between 2D and S2D.

	2D - S2D			(2D - S2D) / 2D × 100%		
	Min	Max	Mean	Min	Max	Mean
ΔWL (m)	0.03	5.65	1.37	1.95%	35.99%	29.71%
Δ(#buf + #inv)	0.01K	25K	3.6K	0.40%	10.41%	5.05%
ΔTotal power (mW)	-1.40	9.10	2.29	-1.39%	12.72%	3.71%

Bounded-Error Models

We create three separate models to predict percentage delta (3D (S2D and A3D), relative to 2D) for each power component – internal, switching and leakage. We use the predicted values from these models to predict total power in 3DPE. To predict percentage delta internal power, we use seven parameters: (i) internal power from 2D; (ii) ratio of the number of buffers to the total cell count; (iii) delta internal power in 2D with and without capacitance scaling of 0.7× in the technology capacitance tables; (iv) delta internal power in the post-synthesis netlist with and without capacitance scaling by using wireload models (WLMs); (v) the ratio of utilization to cell area; (vi) ratio of memory area to total cell area; and (vii) AR. To predict percentage delta switching power, we use six parameters: (i) switching power from 2D; (ii) the ratio of total wirelength (WL) in 2D to utilization; (iii) delta WL; (iv) delta switching power in 2D with and without capacitance scaling of 0.7× in the technology capacitance tables; (v) delta switching power in the post-synthesis netlist with and without capacitance scaling by using wireload models (WLMs); and (vi) AR. To predict percentage delta leakage power, we use three parameters: (i) leakage power from 2D; (ii) the ratio of low-Vt cell area to total cell area; and (iii) the ratio of memory area to total cell area.

Figure 3.57 shows our prediction for percentage delta power across our testcases using the S2D flow. The solid black line in the middle indicates the line when there is perfect correlation between actual percentage delta power and predicted percentage delta power. The upper and the lower solid lines define the maximum and minimum errors, respectively. Across our testcases, the worst-case error of our estimations is ∼4.8%. Figure 3.58 shows our prediction of percentage delta power across our testcases using the A3D flow. The upper and lower solid lines respectively indicate the maximum and minimum errors. Across our testcases, the worst-case error of our estimations is ∼5.04%. The histogram of error distribution in Figure 3.59 shows that only a few outliers are responsible for the maximum and minimum errors. The average error from our total power model is ∼-0.1%.[39]

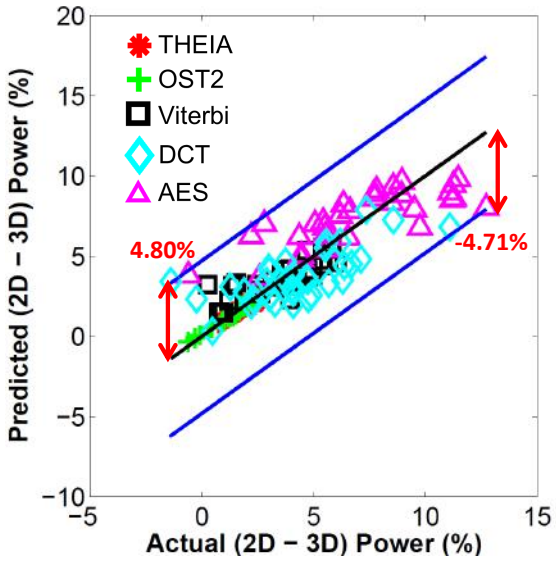


Figure 3.57: Predicted % delta power vs. actual % delta power between 2D and S2D.

Testing of Model Implications

As the time to generate each data point using 2D and 3D flows can be very large (e.g., up to 16 hours for one data point of *spc*), it is practically impossible to train models with a large range of parameter values. Our models should be scalable and generalizable due to use of cross-validation [85] in our methodology. However, we go a step further with a novel "stress test of our 3DPE models. That is, we explicitly test whether the models are capable of returning an

[39]The average (resp. maximum) of absolute errors in our internal, switching and leakage power models are respectively -0.42% (resp. 10.9%), -0.07% (resp. 5.6%) and -0.61% (resp. 2.9%). The testing time is ∼one second per every 500 data points.

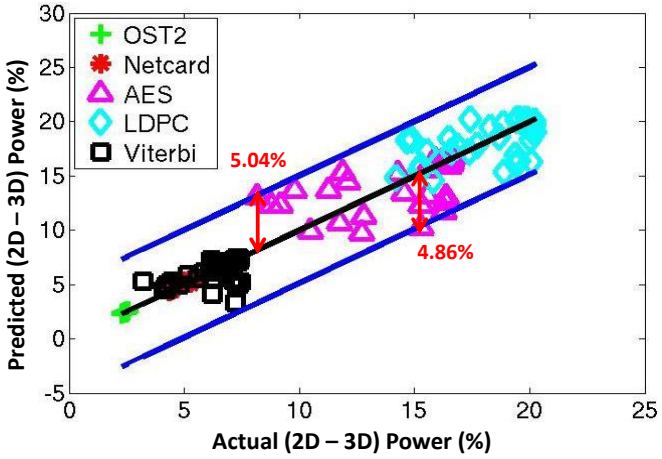


Figure 3.58: Predicted % delta power vs. actual % delta power between 2D and A3D.

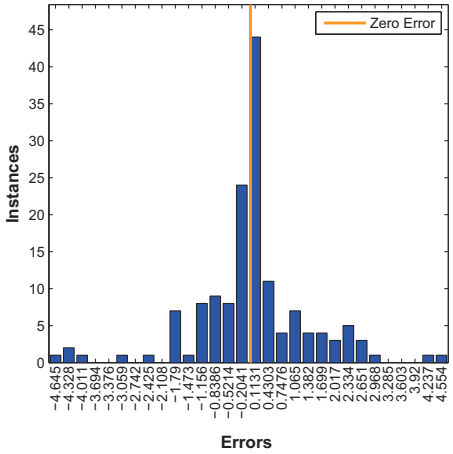


Figure 3.59: Model error distribution of % delta power.

unlikely or unreasonable prediction. (E.g., if it is possible for our models to predict 90% power benefit from 3DIC, this would cast doubt on the models.)[40] We perform stress testing on our total power model using the following methodology. We vary the following 10 parameters in our models: (i) internal, switching and leakage power values in 2D ($K = 1$, $K = 2$, and $K = 3$); (ii) total WL in 2D ($K = 4$); (iii) utilization ($K = 5$); (iv) number of cells ($K = 6$); (v) total cell area ($K = 7$); (vi) number of buffers ($K = 8$); (vii) ratio of memory area to cell area ($K = 9$); and (viii) maximum transition ($K = 10$). Table 3.25 shows the distribution of these parameters extracted from our training dataset. We execute the following steps.

[40]Our sanity-check approach can be a useful addition to the metamodeling works that have become very popular in the recent IC CAD literature.

- For each parameter i, where $i = 1, 2, ..., 10$, we obtain the mean (μ_i) and standard deviation (σ_i) from the training dataset.
- We construct a test dataset by assuming that each parameter follows a Gaussian distribution with mean and standard deviation respectively indexed into values from the sets μ_i' and σ_i', where
  $\mu_i' = \{\mu_i, 2\mu_i, ..., 10\mu_i\}$, and $\sigma_i' = \{\sigma_i, 2\sigma_i, ..., 10\sigma_i\}$.
- We index each of these values from sets μ_i' and σ_i' as $s = \{1, 2, ..., |\mu_i'|\}$, and generate a value for each parameter $x_i = \mu_i'(s) + j \times \sigma_i'(s)$, with j varying from -3 to +3 in steps of 0.2.

We generate a total of 434 test data points for the 10 tuples of parameters. Figure 3.60(a) shows a histogram of percentage predicted delta power. The minimum value is 0.08% (the corresponding bin is 2.17%) and the maximum value is ∼125% (the corresponding bin is 123.3%). The weighted mean of the predicted percentage delta power values is 9.5%. For 14 test data points, our models predict over 100% percentage delta power. These data points have the following attributes which are not practically realizable: (i) the ratio of cell area to the number of cells is larger than the area of the largest cell in the technology library,[41] that is, the number of cells, utilization and the cell area are mismatched; (ii) the ratio of wirelength to the number of cells is less than 50μm, that is, the wirelength and the number of cells are mismatched; and (iii) the maximum transition and/or the maximum fanouts are more than 10× the maximum value in the technology libraries, that is, constraints are mismatched. Figure 3.60(b) shows a histogram of percentage delta power benefits for data points that are realizable for practical netlists and do not violate constraints with respect to the technology libraries. The maximum possible percentage delta power for these data points is ∼39%, which indicates that our model predictions are close to the values of 3DIC power improvements reported in [136]. Recently published studies by Chan et al. [38] report the maximum possible percentage 3D power benefit to be 36% using an "infinite" dimension methodology. This is another possible confirmation that 3DPE predictions are not unreasonable.

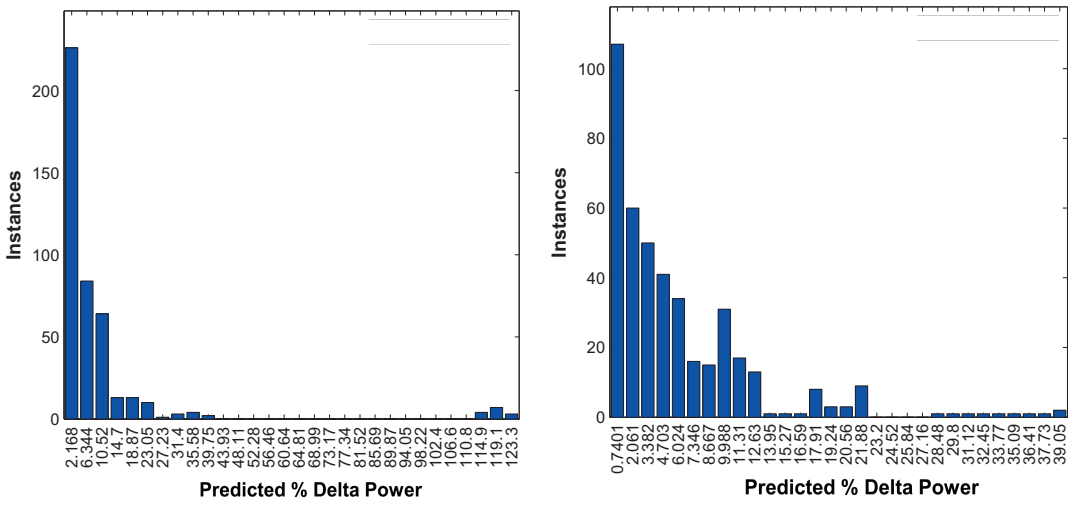
Model-Guided Implementation

An "ultimate" goal of our 3DPE modeling work is to enable fast and accurate design- and implementation-space exploration without actually having to implement a testcase either in

[41] In our 28nm FDSOI libraries, the size of the largest cell is 4.4μm^2. The inter-buffer distance is ∼120–150μm [35]. The max transition is 375ps and the max fanout is 20.

Table 3.25: Distributions of parameters for stress testing.

	$K=1$ (mW)	$K=2$ (mW)	$K=3$ (mW)	$K=4$ (m)	$K=5$ (%)	$K=6$ ($\times 10^3$)	$K=7$ (mm^2)	$K=8$ ($\times 10^3$)	$K=9$	$K=10$ (ps)
μ	71.45	34.97	0.33	4.49	0.56	134.51	0.18	186.48	0.16	200
σ	67.90	30.89	0.45	5.88	0.13	141.77	0.20	286.65	0.22	56



(a) (b)

Figure 3.60: Percentage predicted delta power distributions (a) when practically unrealizable data points are included and (b) when only practically realizable data points are included.

2D or 3D. Toward this end, we explore whether our models can provide guidance to designers as to which classes of testcases are amenable to what kinds of 3D benefits. We conduct two experiments to demonstrate how 3DPE (developed using the S2D flow) can provide guidance to designers.[42]

- **MGI-I** – To predict the wireload model (WLM) scaling *at synthesis* that can lead to the smallest post-P&R power in 3D for a testcase.
- **MGI-II** – To use a low-utilization (small tool runtime) trial 2D implementation to predict the % delta power of a high-utilization (large tool runtime) 3D implementation.

Experiment MGI-I. The goal of this experiment is to create a "properly 3D-aware" netlist by scaling WLM capacitances that can deliver the smallest power in 3D. We create eight WLMs with capacitances $\{1.0, 0.85, 0.70, 0.60, 0.50, 0.45, 0.40, 0.33, 0.3\}$pF. We use the *aes* testcase, set clock periods to $\{0.8, 0.9, 1.0\}$ns, run synthesis and S2D flow with netlists synthesized with

[42]Note that although we use one testcase to demonstrate MGI-I and MGI-II, the conclusions drawn are not limited to a specific testcase.

the scaled WLMs, and then obtain 27 data training data points. As part of our model training described in Section 3.4.2, we comprehend WLM capacitance as a parameter. We now create a test dataset in which WLM capacitances are varied in steps of 0.05pF and choose the WLM $W_{best,model}$ that achieves the largest delta power from our models. We run synthesis and S2D flow with WLM $W_{best,model}$ and quantify the cost of misprediction with the WLM $W_{best,actual}$ that delivers the minimum 3D power after implementation using the S2D flow. Figure 3.61 shows how 3D power changes (albeit not too significantly) with WLM capacitance for the *aes* testcase. S2D always uses 1.0pF, but the minimum power is achieved with WLM capacitance of 0.45pF, and the model predicts 0.75pF. The difference in S2D vs. our models is ∼1mW or ∼5%. We see that WLM scaling can achieve smaller 3D power, and that 3DPE models can guide the implementation to achieve within ∼1.62% of the minimum power.

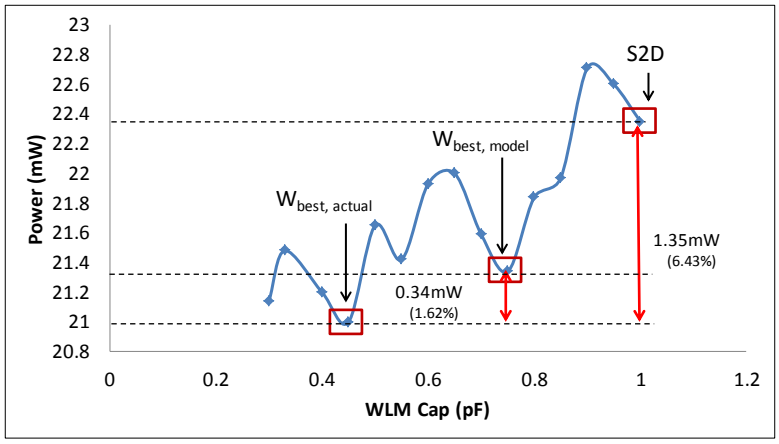


Figure 3.61: Percentage delta power benefits between actual, model and S2D implementations.

Experiment MGI-II. The goal of this experiment is to predict 3DIC power benefits (relative to 2DIC implementation) when the standard cell utilization is higher than the utilizations used in training the models. High utilizations in large designs such as *THEIA* incur large runtimes of around eight hours per data point. On the other hand, low-utilization runs can be fast but have smaller 3D benefit. Table 3.26 shows 3DPE modeling accuracy for different combinations of aspect ratios, clock periods and utilizations. The actual % delta power ranges from 1.82% to 2.88%. We implement the testcase with these parameters to quantify the modeling error. 3DPE can very accurately guide high-utilization design-space exploration because it is trained with small testcases (e.g., *aes* and *dct*) at high utilization and is able to generalize to larger testcases.

Table 3.26: Predicted vs. Actual 3D power with high utilization.

Clock Period (ns)	**AR**	**UTIL** (%)	Predicted % Delta Power (mW)	Actual % Delta Power (mW)
1.60	1.2	40	2.07	1.82
1.60	1.0	42	1.97	2.14
1.80	1.2	45	2.15	2.46
1.80	1.0	48	2.18	2.88

3.4.3 Conclusions

It is difficult to quantify the benefits of 3DIC over a corresponding 2DIC implementation for arbitrary designs because no golden 3DIC flow currently exists. Yet, estimating 3DIC benefits, particularly for the power reduction value proposition, is a critical open issue. We develop a new prediction tool, 3DPE, to predict percentage delta power benefits of 3DIC relative to 2DIC implementation. Such a tool is useful for designers because it filters out design blocks that can achieve large power benefits in 3D and performs fast design-space exploration to determine various 3DIC implementation parameters. 3DPE consists of internal, switching, leakage and total power models. We have developed each of these models using two separate academic 3D flows – the S2D flow of [196] and the A3D flow (described in Section 5.1), applying a novel modeling technique that includes WLM scaling, influential parameter identification and bounded errors. The generated models are accurate, with worst-case errors within 5%. We furthermore present novel applications/validations that include "stress test" and "model-guided implementation" (MGI). We demonstrate how 3DPE can be used in MGI to predict power benefits of blocks that have high utilization and long runtimes, in a fast and accurate manner.

3.5 Acknowledgments

Chapter 3 is in part a reprint of W.-T. J. Chan, Y. Du, A. B. Kahng, S. Nath and K. Samadi, "BEOL Stack-Aware Routability Prediction from Placement Using Data Mining Techniques", *Proc. IEEE International Conference on Computer Design*, 2016, to appear; W.-T. J. Chan, K. Y. Chung, A. B. Kahng, N. MacDonald and S. Nath, "Learning-Based Prediction of Embedded Memory Timing Failures during Initial Floorplan Design", *Proc. Asia and South Pacific Design Automation Conference*, 2016; W.-T. J. Chan, Y. Du, A. B. Kahng, S. Nath and K. Samadi, "3D-IC Benefit Estimation and Implementation Guidance from 2D-IC Implemen-

tation", *Proc. IEEE/ACM/EDAC Design Automation Conference*, 2015; A. B. Kahng, B. Lin and S. Nath, "ORION3.0: A Comprehensive NoC Router Estimation Tool", *IEEE Embedded Systems Letters* 7(2) (2015); A. B. Kahng, B. Lin and S. Nath, "High-Dimensional Metamodeling for Prediction of Clock Tree Synthesis Outcomes", *Proc. ACM International Workshop on System-Level Interconnect Prediction*, 2013; and A. B. Kahng, B. Lin and S. Nath, "Explicit Modeling of Control and Data for Improved NoC Router Estimation", *Proc. IEEE/ACM/EDAC Design Automation Conference*, 2012.

I would like to thank my co-authors Wei-Ting Jonas Chan, Dr. Kun Young Chung, Dr. Yang Du, Professor Bill Lin, Professor Andrew B. Kahng, Nancy MacDonald, and Dr. Kambiz Samadi.

Chapter 4

Improved Accuracy of Electrical Modeling and Enablement of Basic Physical Design Optimizations

This chapter first presents two new "big data" approaches to the timer correlation problem. In the first approach, we develop a machine learning-based tool, **G***olden* **T***imer e***X***tension* (GTX), to correct divergence between timers in flip-flop setup time, cell arc delay, wire delay, stage delay, and path slack at timing endpoints. We propose a methodology that allows us to apply GTX to two commercial timers. We then evaluate scalability of GTX across multiple designs and foundry technologies / libraries, both with and without signal integrity analysis. In our second approach to the timer correlation problem, we extend GTX to develop predictors of timing in signal integrity (SI) mode based on timing reports from non-SI mode. Timing analysis in non-SI mode is faster and the license costs can be several times less than those of SI mode. We propose electrical and logic structure parameters that affect the incremental arc delay/slew and path delay (i.e., the difference in arrival times at the clock pin of the launch flip-flop and the D pin of the capture flip-flop) in SI mode, and develop models that can predict these SI-aware delays. The third work in this chapter describes a methodology to develop accurate models of post-routing optimization of signal delays at multiple signoff corners, so as to enable a new optimization of clock skew variation across corners.

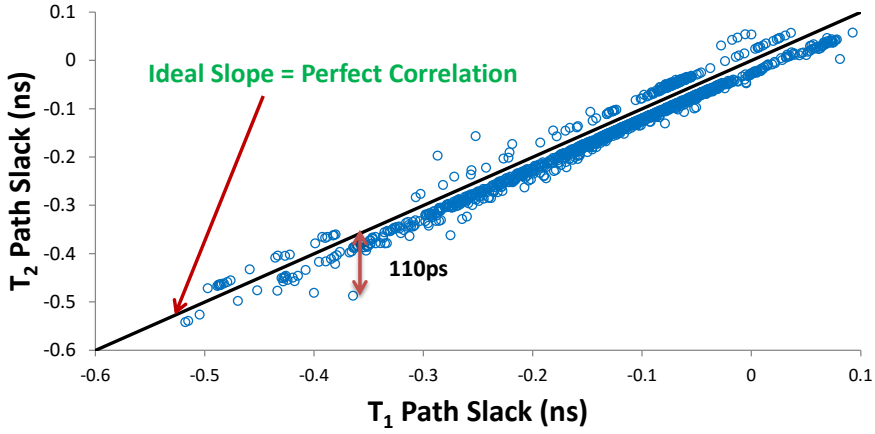
4.1 A Deep Learning Methodology to Proliferate Golden Signoff Timing

Accurate timing closure is a critical step in signoff flows of all semiconductor companies [176] and can consume up to 60% of design time [77]. Multiple static timing analysis (STA) tools exist today and different companies adopt different tools as "golden" or the best-in-class STA tool depending on their requirements and product quality standards. According to the analyst firm Gary Smith EDA [228], EDA vendors such as Synopsys [342], Cadence [288], CLK Design Automation [290], Incentia Design Systems [300] and Mentor Graphics [312] provide STA and signal integrity analysis tools for use in IC design. These tools typically have high license fees and long runtimes, and they invariably diverge in their timing reports – even though each is well-calibrated to the latest commercial circuit simulators and "qualified" for signoff at leading foundries. Owing to cost and budget constraints, design teams may have limited or no access to a particular "golden" timing tool, but may be interested in comparing the divergence in timing reports between the timing tool they use and that golden tool. The ability to correlate with another (golden) timing tool helps design teams understand if they have overdesign or underdesign, i.e., when their timing tool's reports are respectively pessimistic or optimistic compared to the golden tool's reports. Another use model may be to estimate, based on the timing reports of design implementation tools, how far the implementation is from signoff after each optimization loop (timing-driven placement, congestion-aware routing, leakage reduction, etc.).

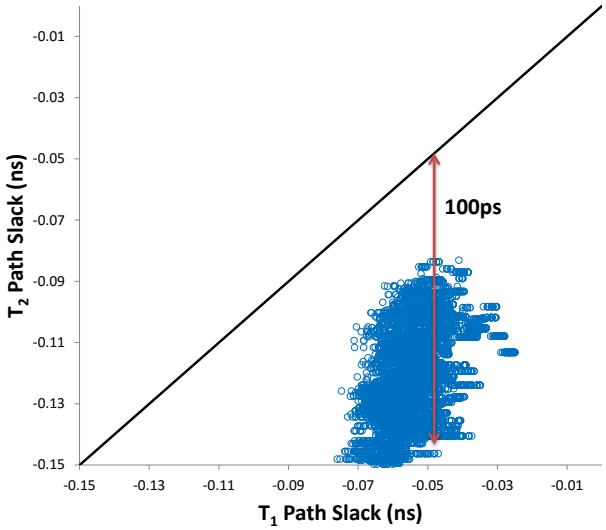
We use "gt1-gt2" (that is, "golden tool 1 to golden tool 2") to refer to the problem of correlating two signoff timing tools. We estimate the timing reports of one tool based on the reports of another tool. The correlation problem is extremely complex because:

- tools can suffer from the complexity of millions of lines of black-box code;
- tools can diverge from published user documentation [125], and maintain implementation "errors" for legacy reasons;
- discrepancies between tools change with releases [222] (typically 2$\times$ per year for mature tools from major EDA providers);
- tool licenses explicitly prohibit benchmarking and reverse-engineering of internal algorithms; and
- the correlation problem is seemingly "unbounded", as the space of possible timing paths,

slew times, multiple-input switching events, coupling effects on delay, etc. is essentially infinite.



(a) T_1–T_2.



(b) T_1–D_1.

Figure 4.1: Path slack discrepancies.

As reviewed in Section 2.2, no existing tool or methodologies correlate two signoff timing tools. The cost of leaving the gt1-gt2 problem unsolved grows as embedded processor cores reach 3GHz frequencies in 20nm and 16/14nm designs: miscorrelations of >100ps in timing slack correspond to discrepancies of multiple (3 – 4) logic stages at these advanced technology nodes and can strongly impact power and/or area tradeoffs [16] [34] [77]. Figures 4.1(a) and 4.1(b) respectively show examples of 110ps and 100ps timing miscorrelations between two

leading commercial signoff timing tools T_1 and T_2, as well as between T_1 and a commercial design implementation tool D_1. According to industry experts, reasons for miscorrelation include the use of multiple engines within tools for optimal accuracy and runtime as well as the effects of net length and long waveform tail [68] [181] [232]. Our premise is that the gt1-gt2 problem, while extremely complex, is still treatable as a finite problem that is amenable to big-data mindsets as has been recently seen in highly challenging applications such as natural language processing [295] [331]. Specifically, we identify appropriate modeling parameters and develop a tool, GTX (Golden Timer eXtension), using well-known machine learning techniques[43] to correct[44] setup time, cell delay, wire delay, stage delay, and path slack divergence between tools. Our methodology is properly considered to be deep learning-based because the models in GTX are hierarchical, e.g., the output of the cell and wire delay models are input to the stage delay model [216]. Our modeling goals for each model are to (1) minimize the sum of squared errors, and (2) minimize the maximum range of errors. We achieve:

- Correlation of path slack at timing endpoints[45] between two tools within a range of $<$30ps for designs implemented in 28nm FDSOI and 45nm GS foundry libraries using NLDM delay tables;
- Strong correlation results independent of whether signal integrity (SI) and on-chip variation (OCV) are enabled or disabled (non-SI, non-OCV); and
- Scalability and portability of GTX to design projects in new foundry libraries.

Our main contributions in this section are summarized as follows.

- We develop GTX by identifying appropriate modeling parameters, and by exploiting big-data mindsets and machine learning techniques to correct timing divergence between tools. To the best of our knowledge, our work is the first to attempt timing correlation with a big-data approach.
- Our models to correlate path slack between timing tools are accurate across multiple technology nodes and designs. In non-SI mode, our models reduce the range of divergence in path slack between tools from 32.5ps to 5.9ps (i.e., 5.5$\times$ reduction) in 28nm FDSOI. In SI mode, our models reduce the range from 139.3ps to 21.1ps (i.e., 6.6$\times$ reduction) in 45nm

[43] Detailed descriptions of the machine learning techniques used in this work can be found in [85].

[44] GTX uses the timing reports of T_1 to generate timing values that reduce divergence from T_2. Of course, GTX can also perform the reverse, i.e., use timing reports of T_2 to reduce divergence from T_1.

[45] We refer to path slacks at timing endpoints as, simply, path slacks.

GS. We demonstrate that our method applies to small as well as relatively large (*leon3mp*) designs.

- We demonstrate that GTX can reduce the number of outliers (from 407 to 26, i.e., 16× reduction, in the example we study) by incrementally modifying models when new designs are added.

- GTX can be applied to multiple designs, implementation flows, and technology nodes. We demonstrate the generality of GTX with two use cases – correlating two signoff tools, and correlating one signoff tool with a design implementation tool.

4.1.1 Methodology

We now describe our methodology to develop flip-flop setup time, cell, wire, and stage delay and path slack models for GTX. We describe parameters used in the models, and then the machine learning methodology used to develop these models.

Parameter Selection

Signoff timing tools typically differ in path slack due to discrepancies in cell, wire and stage delays. Further, tools differ in their calculations of rise/fall delays across each input-to-output pin arc of cells. Figures 4.2 – 4.5 illustrate these discrepancies between two leading commercial signoff timing tools T_1 and T_2. Figure 4.5 in particular highlights the discrepancies between tools across a single MUX21 cell.

Path slack is calculated from the required setup time at the capture flip-flop of the path and from stage delays; these in turn are calculated from cell and wire delays in each stage. Figures 4.2 and 4.3 show that one tool (T_1) can be optimistic in cell delay reports and pessimistic in wire delay reports as compared to the other tool (T_2). There is a "canceling" effect for stage delays [125]. However, the "canceling" effect does not eliminate stage delay discrepancies between tools, as illustrated in Figure 4.4.

Table 4.1 lists all parameters used in our models. Note that cell and wire delays include incremental values for SI mode analysis.

Modeling Flow for GTX Models

To minimize divergence and achieve close correlation between signoff timing tools, we use a big-data approach and machine learning models. We do not reverse-engineer tools as li-

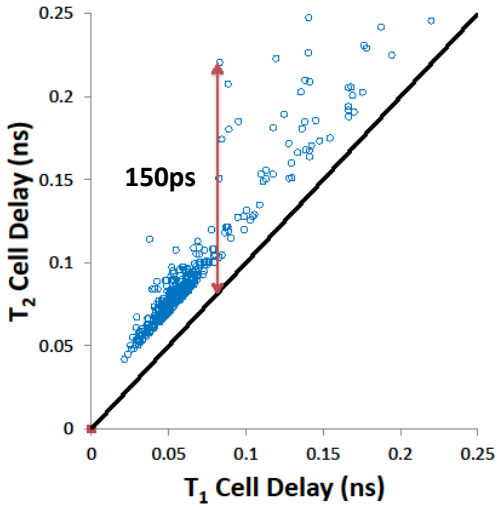


Figure 4.2: Cell delay discrepancy.

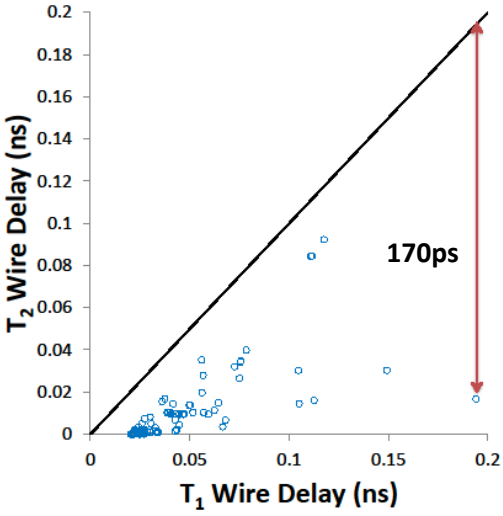


Figure 4.3: Wire delay discrepancy.

censes prohibit us from doing so; reverse-engineering can also become intractable because each tool implements millions of lines of legacy and black-box code. Instead, we develop machine learning-based models for GTX to correct the divergence in setup time, cell, wire, and stage delays and apply these models to fit path slack between two STA tools. In the following, we use the latest versions of two widely used commercial signoff tools, and show how reports from a tool T_2 can be used to develop models that estimate a tool T_1's reports. Our methodology is applicable to any pair of signoff or design implementation tools that can perform STA.

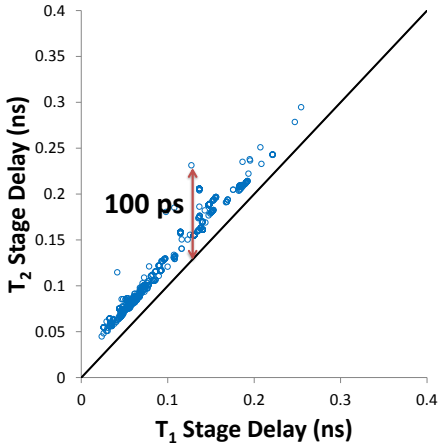


Figure 4.4: Stage delay discrepancy.

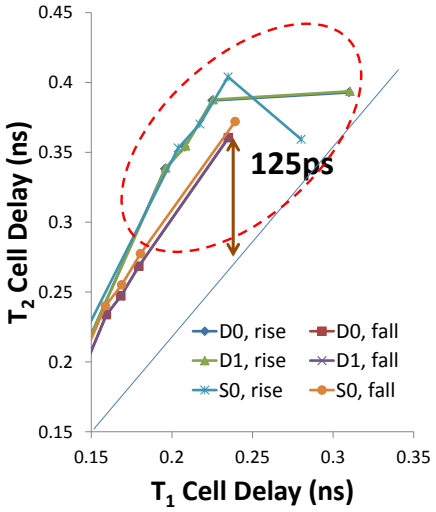


Figure 4.5: Pin arc, rise/fall discrepancy.

In non-SI mode, divergence between tools is typically smaller for wire delays than for cell delays, so we develop only a cell delay model.[46] In SI mode, however, wire delay divergence between tools can be significant due to differences in handling of crosstalk effects, so we model both cell and wire delays. Therefore, in both non-SI and SI modes we develop three (path slack, setup, and cell delay) models; additionally, in SI mode, we develop wire and stage delay models.

[46]Our experimental results indicate that by introducing wire delay models in non-SI mode, GTX results do not change significantly.

Table 4.1: Parameters reported by each tool in both SI and non-SI modes.

Parameter	Meaning	Mode
C_{eff}	Effective load capacitance	SI, non-SI
C_{coup}	Total coupling capacitances	SI
C_w	Wire ground capacitance	SI, non-SI
R_w	Wire resistance	SI, non-SI
$d_{tr,c,i}$	Cell input slew	SI, non-SI
$d_{tr,c,o}$	Cell output slew	SI, non-SI
d_c	Cell delay	SI, non-SI
d_w	Wire delay	SI, non-SI
d_{stg}	Total stage delay	SI, non-SI
$d_{su,\text{ff}}$	Flip-flop setup time	SI, non-SI
$d_{slk,p}$	Path slack	SI, non-SI

Figure 4.6 shows the hierarchy of the five models in GTX and why we refer to our methodology as "deep". We use hierarchical rather than flat modeling for improved correlation and decreased range of divergence. Combining individual models of cell delay, wire delay, and setup time in an additive manner to estimate path slack can result in errors being added up as well. For example, Kahng et al. [125] use additive wire delay models that result in large divergence in path slack. Therefore, they invoke the golden timer at regular intervals to correct the path slack. Hierarchical modeling prevents errors being added linearly by applying an additional layer of modeling that provides a better fit to timing estimates. In the following discussion, $T_1(\cdot)$ and $T_2(\cdot)$ refer to values of parameter $(\cdot)$ respectively reported by T_1 and T_2.

Setup time. Our experiments in 28nm FDSOI indicate that flip-flop setup time reports between timing tools can diverge by up to 17.5ps. To reduce the divergence between tools, we model setup time as

$$\widehat{T_1}(d_{su,\text{ff}}) = f\left(T_2(d_{su,\text{ff}}, d_{tr,c,i})\right) \tag{4.1}$$

where $\widehat{T_1}(d_{su,\text{ff}})$ is the predicted T_1 setup time, $T_2(d_{tr,c,i})$ refer to the T_2-reported input slews at the D and clock pins of the capture flip-flop, and $f(\cdot)$ is the modeling function. These parameters correspond to those used to index the NLDM setup time tables in foundry libraries.

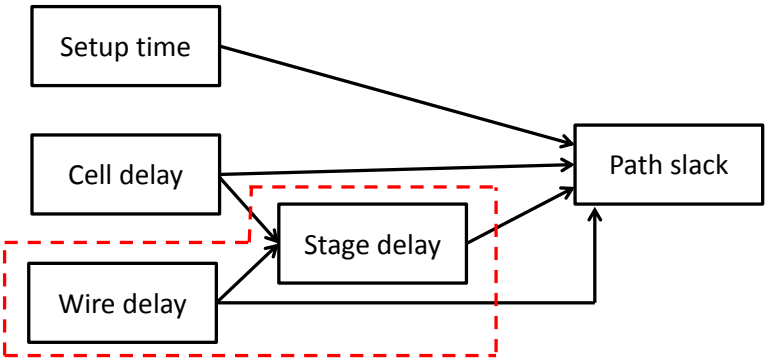


Figure 4.6: Hierarchical GTX models. The models within the dotted lines are used only in SI mode.

Cell delay. Our 28nm FDSOI studies also indicate that tools can differ in reported cell delays by >300ps (under extreme load and slew conditions).[47] Furthermore, the delay divergence between tools can vary across different input-to-output pin arcs, especially in complex cells such as *AOI* and *MUX*. In addition, tool reporting for rise and fall delays can diverge significantly. Figure 4.5 illustrates these divergences for rise and fall delays of *D0, D1* and *S0* pins of a 2:1 *MUX*. With these considerations, we develop rise and fall delay models of each input-to-output pin arc of each cell in the design as

$$\widehat{T_1}(d_c) = f\left(T_2(d_c), \mathrm{LUT}(d_c)\right) \tag{4.2}$$

where $\widehat{T_1}(d_c)$ is the predicted T_1 cell delay, and $\mathrm{LUT}(d_c)$ is the cell delay determined using linear interpolation of NLDM delay lookup tables (LUTs) of a given cell [125]. The inputs for LUT interpolation are $T_2(C_{eff})$ and $T_2(d_{tr,c,i}) + \Delta Slew$, where $\Delta Slew$ is the upstream slew correction between the tools. We use $d_{tr,c,i}$ and C_{eff} because these are the indices of the NLDM delay and slew lookup tables in the foundry timing libraries. We use $\Delta Slew$ to correct upstream slew differences between the tools because our experiments indicate that certain tools always propagate the worst slew in path-based analysis mode. We model $\Delta Slew$ as

$$\Delta Slew = (\alpha(LUT(d_{tr,c,o}) + \beta) - T_2(d_{tr,c,o}) \tag{4.3}$$

where $LUT(d_{tr,c,o})$ is the output slew of the upstream cell calculated using linear interpolation

[47] Simulations with HSPICE [339] indicate that T_1 is accurate to within 0.02ps of HSPICE results, whereas T_2 diverges more substantially from HSPICE.

between the library LUTs based on the T_2-reported $d_{tr,c,i}$ and C_{eff}. α and β are regression coefficients determined by fitting $T_2(d_{tr,c,o})$ of the upstream cell to $T_1(d_{tr,c,o})$.

Wire delay. We model wire delay, using a similar set of parameters as in [125], as

$$\widehat{T_1}(d_w) = f\left(T_2(d_w, d_{tr,c,o}), R_w \cdot \{C_w, C_{eff}, C_{coup}\}\right) \tag{4.4}$$

where $\widehat{T_1}(d_w)$ is the predicted T_1 wire delay and the parameters $R_w \cdot \{C_w, C_{eff}, C_{coup}\}$ represent delay due to different capacitances.

Stage delay. We model stage delay, using a similar set of parameters as in [226] and [240], as

$$\widehat{T_1}(d_{stg}) = f\left(T_2(d_{stg}), \widehat{T_1}(d_w, d_c)\right) \tag{4.5}$$

where $\widehat{T_1}(d_{stg})$ is the predicted T_1 stage delay.

Path slack. We develop two path slack models for non-SI and SI modes. The models are different because in SI mode, wire and stage delay models are required to correct large discrepancies in path slack as described above. Our path slack model in non-SI mode is

$$\widehat{T_1}(d_{slk,p})_{\tilde{SI}} = f\left(T_2(d_{slk,p}, \frac{\sigma}{\mu}(d_w)), \frac{\sigma}{\mu}(\widehat{T_1}(d_c, d_{su,\text{ff}}))\right) \tag{4.6}$$

where $\widehat{T_1}(d_{slk,p})_{\tilde{SI}}$ is the predicted T_1 path slack in non-SI mode and $\frac{\sigma}{\mu}(\cdot)$ is the *coefficient of variation* of the parameter $(\cdot)$. Our path slack model in SI mode is

$$\widehat{T_1}(d_{slk,p})_{SI} = f\left(T_2(d_{slk,p}), \frac{\sigma}{\mu}(\widehat{T_1}(d_w, d_c, d_{stg}, d_{su,\text{ff}}))\right) \tag{4.7}$$

where $\widehat{T_1}(d_{slk,p})_{SI}$ is the predicted T_1 path slack in SI mode.

Besides coefficient of variation, we also try two other normalization techniques, *standard score* [85] and *variance-to-mean* ratio [85]. We experimentally observe that coefficient of variation and standard score give similar results because they determine the contribution of each wire, cell, or stage delay to the overall delay of all wires, cells, or stages in a path. Variance-to-mean ratio, on the other hand, cannot determine the contribution of an individual (wire, cell, or stage) delay to the corresponding total delay in a given path; hence, it is less accurate.

Incremental modeling. Large product organizations often tape out multiple designs in the same technology. A new design can, conceivably, use cells and/or wiring configurations that are "out of scope" for the current fitted models. Such "new" cells/wires can introduce divergence in timing reports.[48] To mitigate these divergences, we propose an *incremental modeling* flow as follows.

- Step 1. Add any observations that result in divergence in timing of more than a threshold value (e.g., 10ps) to the existing training sets of each of the GTX models.
- Step 2. Re-train GTX models with the training sets from Step 1.
- Step 3. Test the updated models on all data points from the new design.

4.1.2 Experimental Setup and Results

We now present validation of GTX and results of our experiments. First, we describe our design of experiments, including descriptions of designs used and our flow to collect training, validation and testing data for modeling. Second, we conduct four experiments to assess and measure performance of GTX. We use two leading (foundry-qualified) signoff timing tools T_1 and T_2, and a leading design implementation tool D_1, in our experiments. All tool versions are 2013 releases.

- **Experiment 1.** Correlate tools T_1 and T_2 in non-SI mode.
- **Experiment 2.** Correlate tools T_1 and T_2 in SI mode.
- **Experiment 3.** Correlate tools T_1 and D_1 in SI mode.
- **Experiment 4.** Validate the incremental modeling flow on a new design with many outliers.

Design of Experiments

We use real-world designs as well as artificial circuits in our experiments. Real-world designs include the *leon3mp* multi-core processor from Aeroflex Gaisler AB [306], and *aes_cipher_top*, *wb_dma_top* and *jpeg_encoder* from OpenCores [318]. We generate artificial training circuits to finely control various aspects of a timing path to verify robustness of our

[48] If new cells are not introduced in a design, incremental modeling is not required for GTX.

methodology.[49] We synthesize all designs with 45nm bulk triple-Vt and 28nm FDSOI dual-Vt foundry libraries. We perform hierarchical synthesis in 45nm GS and flat synthesis in 28nm FDSOI to demonstrate the scalability of GTX across different flows and foundry technologies. We generate Verilog netlists, Synopsys Design Constraints (SDC) [21], and Standard Parasitic Exchange Format (SPEF) [333] files as inputs to timing tools.

Real-world designs. Table 4.2 shows the post-layout number of standard-cell instances for each design implemented in 45nm GS and 28nm FDSOI foundry libraries. In 45nm GS, we use less strict constraints on timing, maximum fanouts, and transition, and we restrict tools from using cell sizes $X0$, $X1$, and $\geq X20$.[50] However, in 28nm FDSOI we allow the tools to use all cells from the library, and apply tight timing constraints but relaxed maximum fanout and transition constraints.[51]

Table 4.2: Number of instances in real-world design.

	# Instances (clock period in ns)	
Testcase	45nm GS	28nm FDSOI
aes_cipher_top	18818 (1.0)	16688 (0.8)
wb_dma_top	3641 (0.5)	2349 (0.5)
jpeg_encoder	46702 (1.25)	53641 (0.67)
leon3mp	–	750854 (1.2)

Artificial training circuits. We develop generators using custom *Tcl* scripts to finely control various aspects of a timing path as listed below.

- Path – #stages and #fanouts.
- Cell – input slews, types, sizes, and Vt flavors.
- Wire – parasitics (R_w, C_w, C_c), #segments, and aggressors.

[49] We observe that synthesis and implementation tools tend to construct designs that occupy the middle region of delay tables. We create artificial training circuits to define the extreme ranges of timing discrepancies so as to create robust and scalable models.

[50] We observe that these cell sizes are known for being problematic in designs; some designers commonly use similar restrictions.

[51] In 45nm GS, the maximum fanout constraint is set to 20 and the maximum transition time is set to one-sixth of the clock period. In 28nm FDSOI, these values are respectively 40 and one-eighth of the clock period.

CPU time needed to generate the Verilog netlist, SDC, and SPEF files is ∼6s (independent of the number of fanouts and stages) on an Intel Xeon E5-2640 2.5GHz server. The size of each of these files is ∼4KB for a circuit with one stage and a fanout of one. The size of SPEF files can potentially be large (e.g., 232KB for a circuit with 60 stages and four fanouts in each stage) because we do not implement name mapping.

Each training circuit consists of a chain of *driver* and *driven* cells and flip-flops at the beginning (launch) and the end (capture) to create a *constrained path*. Optionally, cells can be added to achieve multiple fanouts from each driver. Pins that are not on the constrained path are connected to dummy flip-flops and/or ports to ensure that there are no floating pins. An example of a circuit with two *stages* without SI aggressors is shown in Figure 4.7. The *constrained path* is from *f1/Q* to *f2/D*, through instances *u1* and *u2*. To generate a training circuit with multiple stages, the "repeated unit" in Figure 4.7 is replicated between the launch and the capture flip-flops[52]. Figure 4.8 illustrates a circuit with one SI aggressor and coupling capacitances.

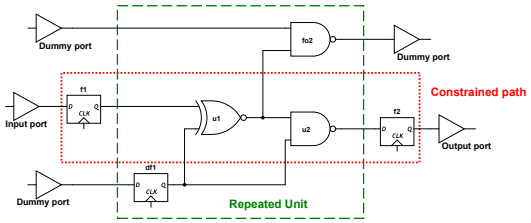


Figure 4.7: Example of a non-SI training circuit.

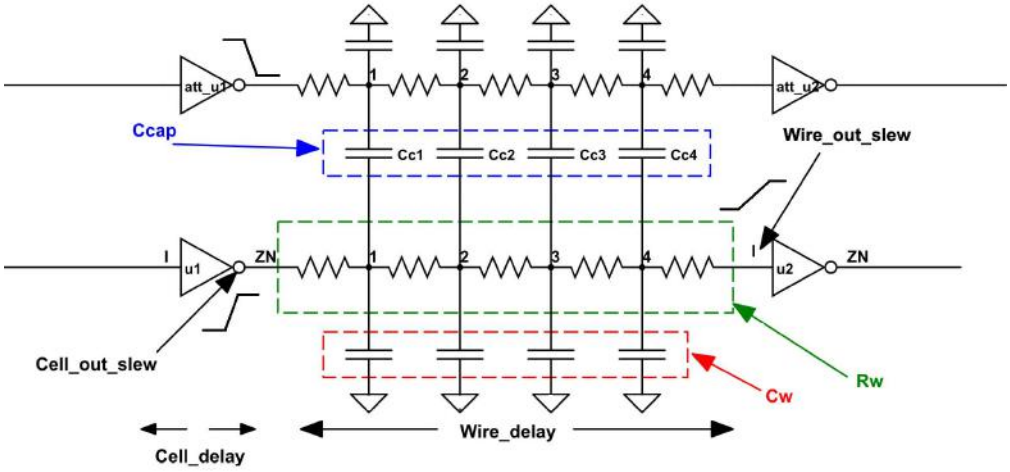


Figure 4.8: Example of an SI training circuit.

[52]The "repeated unit" contains a dummy flip-flop which is inserted to ensure valid operation of gates, and is not part of the *constrained path*.

Data Collection for Modeling

We generate training, validation, and test datasets in the following way. First, we obtain Verilog netlists, SDC, and SPEF files with coupling capacitances for our designs. Second, we use 2013-released versions of two commercial signoff timing tools, commonly adopted as golden tools by design teams, to perform path-based timing analysis of the top 10K worst paths in both SI and non-SI modes.[53] For Experiment 3, we use a commercial design implementation tool D_1. Last, to compare tools in a fair manner, we ensure that options and global flags for both tools are set to produce similar reports as follows:

- Timing reports. Each tool reports all parameters from Table 4.1.
- Path timing calculation. Each tool performs path-based analysis, i.e., slews are propagated only along "paths-of-interest".
- SI and OCV analyses. SI- and OCV-aware analysis modes are enabled, and glitch analysis is disabled.[54]
- Parasitic information. In SI mode, each tool uses coupling parasitic information for timing analysis.

Detailed cell characterization for the cell delay model. We perform a one-time detailed characterization of each input-to-output pin arc of each cell in a design because our experiments indicate that cell delay requires very detailed modeling to minimize the range of errors.[55] We create a single-stage artificial training circuit for the cell, annotate multiple input slews and capacitances that span values seen in the entire set of NLDM delay tables in the foundry libraries used by the design, and obtain rise and fall delays for each combination of slews and capacitances as well as all rise and fall input transitions. Similar characterization is performed for flip-flops as well. Table 4.3 shows sample resource utilization for cell characterization for a design implemented with 28nm foundry libraries. File size refers to the file with training, validation and test datasets for each cell.

[53] We use custom *Tcl* scripts to ensure that the same 10K paths are analyzed by respective tools as we generate our training sets.

[54] Our experiments indicate that in both OCV and non-OCV modes the divergence in clock-to-Q delay and setup times vary by less than 5ps, and delays for other cells vary by less than 1ps. Therefore, in the following we report results in OCV mode only.

[55] When signoff involves multiple corners, cell delays need to be characterized for each corner, and the corner-specific timing model must be used.

Table 4.3: Resource utilization for cell characterization in 28nm FDSOI.

Cell	#arcs	#data points	Time (min)	File size (KB)
INV	1	140	20	20
NAND2	2	280	55	36
MUX21	3	560	95	68
AOI13	4	560	95	68

We characterize a total of 397 cells in 28nm FDSOI and 305 cells in 45nm GS libraries; these contain a total of 1870 input-to-output pin arcs[56]. The characterization time for these cells is 116 hours per core (a one-time overhead of just under 5 days) on an eight-core Intel Xeon E5-1410 2.8GHz server. Table 4.3 shows resource utilization for cell characterization in 28nm FDSOI. *MUX21* and *AOI13* cells have the same runtime and number of training data points because NLDM table sizes vary between these cells, and we use more values of input slews and capacitances from the NLDM tables of *MUX21* than of *AOI13*.

Modeling Techniques

To develop models, we use training data points from artificial circuits and validation data points from real-world designs. To test the models, we use a separate set of data points from our real-world designs. Table 4.4 shows the sizes of the training, validation and test sets for each experiment. Extremely large sizes of our training and test sets reflect our "big-data" approach whereby models are derived using $\geq$200K data points for cell, wire, and stage delays. Thereafter, we may apply our incremental modeling flow for new designs in the same technology/library.

We apply both linear and nonlinear machine learning techniques (least-square regression (LSQR), Artificial Neural Networks (ANN) [85], Support Vector Machines regression (SVM) [42] with radial basis function kernel, and Random Forests (RF) [85]) to all GTX models. For each model, we choose the technique that best minimizes both mean squared error (MSE) and the range of errors, i.e., the difference between maximum and minimum errors. We observe that LSQR and ANN are not as effective as RF and SVM in minimizing the range of errors. ANN is effective in modeling setup time and cell delays, SVM is effective in modeling wire and stage delays, and RF is effective in modeling path slack. We use the built-in *MATLAB vR2013a* [311] toolbox for ANN, *LIBSVM* implementation of SVM in *MATLAB* [42], and an open-source

[56]We characterize only those cells used in our designs. If necessary, an entire library can be characterized.

MATLAB implementation of RF [323].[57] Once models are developed, the time to test a model depends on the size of the test dataset. In our experiments, runtime is ∼3.23s for 30K path slack data points in the test set. Figure 4.9 shows our complete modeling flow for GTX. Note that, by default, model development is a one-time effort. New designs may require incremental modeling to reduce the number of outliers.

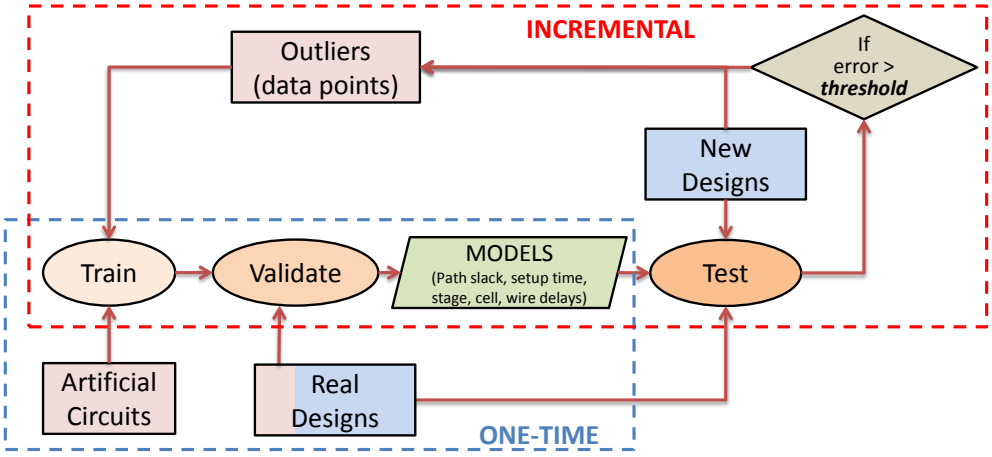


Figure 4.9: Our modeling flow.

We validate GTX with the four experiments described in Section 4.1.2.[58] All experiments are performed on an Intel Xeon E5-2640 2.5GHz server and all reported runtimes are for this platform.

Results of Experiment 1

We correlate timing between T_1 and T_2 in non-SI mode. Figures 4.10(a) and 4.10(b) show the timing divergence between tools before and after fitting. The total runtime is 38 minutes.[59] For ANN, we use up to five hidden layers to model cell delay and two hidden layers to model setup time. We constrain RF to 200 trees and 5000 observations per leaf node. Our

[57] ANN uses hidden layers as a modeling parameter. We sweep the number of hidden layers from one to 10 and choose the value that achieves minimum MSE and range of errors. RF uses multiple classification trees and applies different models to a set of observations at a leaf node of each tree [85]. We sweep the number of trees from 50 to 500 in steps of 50, and the number of observations per leaf node ranging from 1000 to 10000 in steps of 1000. For each experiment, we report the number of trees and the number of observations per leaf node that minimizes MSE and the range of errors.

[58] We ensure that identical input files (Liberty, netlist, SDC and SPEF) are provided to both tools, such that slack miscorrelation is due to delay and timing calculation only. Thus, in Experiment 3 we do not use, e.g., Cadence *Ostrich* [319] to perform parasitic correlation with golden SPEF from Synopsys *StarRC* [343], in which case the design implementation tool's (D_1) parasitic estimation may be another source of miscorrelation.

[59] The reported runtimes for experiments do not include cell characterization time, which is separately discussed in Section 4.1.2.

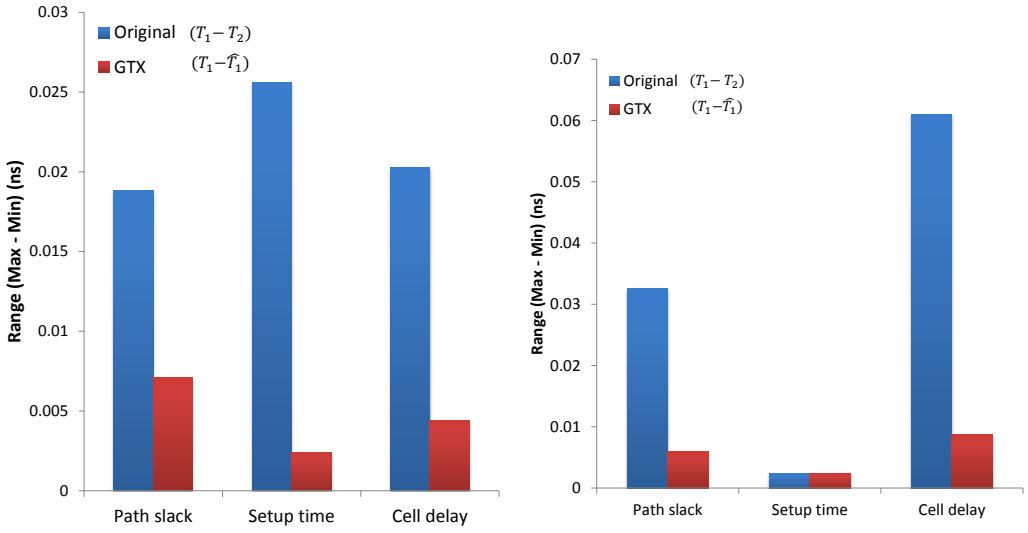
Table 4.4: Training, validation and test dataset sizes.

Experiment #	Module	Training	Validation	Testing
1	Path slack	22680	6480	33240
	Setup time	21798	6228	33114
	Cell delay	354320	15520	326760
2	Path slack	17270	7664	34066
	Setup time	28830	8236	34120
	Cell delay	323804	9875	315776
	Wire delay	304108	39788	143941
	Stage delay	323880	39872	184560
3	Path slack	21770	1440	35790
	Setup time	21540	1120	35340
	Cell delay	320118	11346	332613
	Wire delay	215506	9980	156774
	Stage delay	211736	10553	139327
4	Path slack	17554	5166	32616
	Setup time	28840	8237	34989
	Cell delay	341042	29972	100387
	Wire delay	344086	29900	100520
	Stage delay	341708	29926	98895

models reduce the range of divergence in path slack from 32.5ps to 5.9ps (i.e., 5.5× reduction) in 28nm FDSOI, and from 18.8ps to 7.1ps (i.e., 2.6× reduction) in 45nm GS.

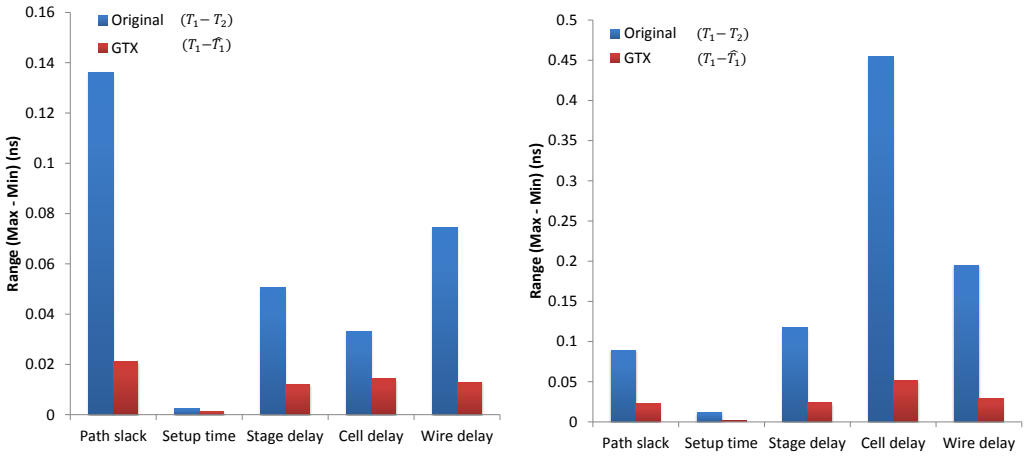
Results of Experiment 2

We correlate timing between T_1 and T_2 in SI mode. Figures 4.11(a) and 4.11(b) show the divergence between tools before and after fitting. The total runtime is 116 minutes. For ANN, we use up to seven hidden layers to model cell delays and two hidden layers for setup time. We constrain RF to 400 trees and 2000 observations per leaf node as we observe that this selection minimizes the range of errors. Our models reduce the range of divergence in path slack from 89.2ps to 22.3ps (i.e., 4× reduction) in 28nm FDSOI and from 139.3ps to 89.2ps (i.e., 6.6×



(a) (b)

Figure 4.10: Experiment 1 results in (a) 45nm GS and (b) 28nm FDSOI.



(a) (b)

Figure 4.11: Experiment 2 results in (a) 45nm GS and (b) 28nm FDSOI.

reduction) in 45nm GS. The stage delay model in GTX improves accuracy even when path slack diverges by >130ps.

To confirm the robustness of our approach, we also conduct the inverse experiment, i.e., where we use timing reports of T_1 to estimate timing reports of T_2. The error metrics are comparable to those shown in Figures 4.11(a) and 4.11(b). Figures 4.12(a) and 4.12(b) depict

five stages from a 28-stage path (Path #2197) from *jpeg_encoder*, with cell delays, wire delays and path slack reported by T_1 and T_2, and their respective fitted values $\widehat{T_1}$ and $\widehat{T_2}$ from GTX. The fitted values are within 8ps of the tool-reported values. Furthermore, we have conducted analogous experiments in a 7nm foundry technology. Specifically, we have trained GTX models using data points from *aes_cipher_top* and ARM *Cortex M0* designs and tested these models using data points from the *leon3mp* design. Figure 4.13(a) shows the path slack values from timing reports of T_1 and T_2, and the divergence is up to 72ps. Figure 4.13(b) shows the predicted path slack values of T_2 using our GTX models based on the timing reports of T_1. We observe that GTX reduces divergence from 72ps to 17ps.

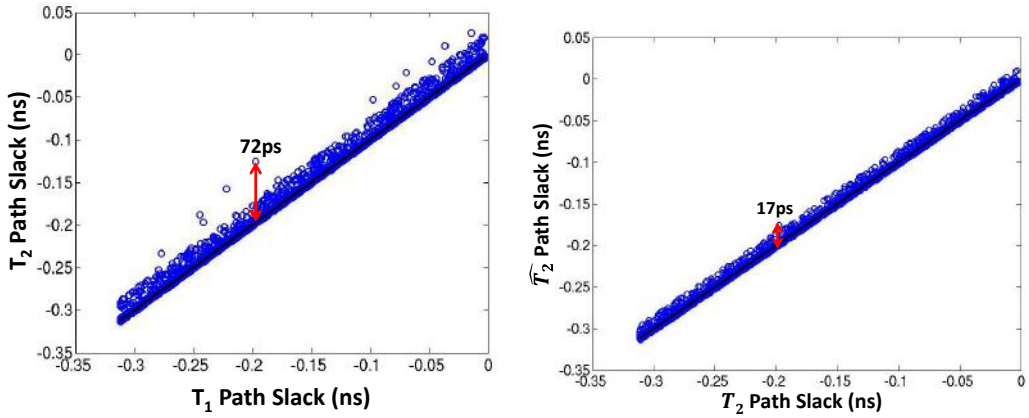
Instance	(Cell)	Dir	Delay(T_1)	Delay(T_2)	Delay($\widehat{T_1}$)	
FE_CN_C274/A1	(NAND2X7)	IN	0.0447	0.0062	0.0452	r
FE_CN_C274/ZN	(NAND2X7)	OUT	0.0565	0.1076	0.0545	f
FE_CN_C277/A	(BUFFX8)	IN	0.0110	0.0044	0.0082	r
FE_CN_C277/Z	(BUFFX8)	OUT	0.0272	0.0664	0.0266	r
FE_CN_C281/A	(INVX8)	IN	0.0057	0.0023	0.0051	f
FE_CN_C281/ZN	(INVX8)	OUT	0.0215	0.0264	0.0213	r
FE_CN_C286/A2	(XOR2X4)	IN	0.0070	0.0066	0.0072	f
FE_CN_C286/Z	(XOR2X4)	OUT	0.0332	0.0581	0.0352	f
FE_CN_C294/A1	(OAI22X4)	IN	0.0825	0.0225	0.0837	r
FE_CN_C294/ZN	(OAI22X4)	OUT	0.0677	0.0781	0.0598	f
slack (VIOLATED)			-0.339	-0.588	-0.342	

(a) T_2 fitted to T_1

Instance	(Cell)	Dir	Delay(T_1)	Delay(T_2)	Delay($\widehat{T_2}$)	
FE_CN_C274/A1	(NAND2X7)	IN	0.0447	0.0062	0.0065	r
FE_CN_C274/ZN	(NAND2X7)	OUT	0.0565	0.1076	0.1063	f
FE_CN_C277/A	(BUFFX8)	IN	0.0110	0.0044	0.0050	r
FE_CN_C277/Z	(BUFFX8)	OUT	0.0272	0.0664	0.0631	r
FE_CN_C281/A	(INVX8)	IN	0.0057	0.0023	0.0026	f
FE_CN_C281/ZN	(INVX8)	OUT	0.0215	0.0264	0.0260	r
FE_CN_C286/A2	(XOR2X4)	IN	0.0070	0.0066	0.0057	f
FE_CN_C286/Z	(XOR2X4)	OUT	0.0332	0.0581	0.0588	f
FE_CN_C294/A1	(OAI22X4)	IN	0.0825	0.0225	0.0231	r
FE_CN_C294/ZN	(OAI22X4)	OUT	0.0677	0.0781	0.0794	f
slack (VIOLATED)			-0.339	-0.588	-0.582	

(b) T_1 fitted to T_2

Figure 4.12: Five sample stages from a 28-stage path in *jpeg_encoder* in 28nm showing cell delay (OUT), wire delay (IN) and path slack reported by T_1 and T_2. The respective fitted values after using GTX are (a) Delay($\widehat{T_1}$) and (b) Delay($\widehat{T_2}$) when T_1 or T_2 is the respective fitted tool. All values are in ns.



(a) T_2 vs. T_1 (b) $\widehat{T_2}$ vs. T_2

Figure 4.13: Path slack values in 7nm technology: (a) T_2 vs. T_1 and (b) $\widehat{T_2}$ (using timing reports of T_1) vs. T_2. GTX reduces slack divergence from 72ps to 17ps.

Results of Experiment 3

We correlate timing between T_1 and a leading design implementation tool D_1 in SI mode in 28nm FDSOI. Figure 4.14 shows the divergence between tools before and after fitting. The total runtime is 104 minutes. For ANN, we use up to seven hidden layers to model cell delay and five hidden layers for setup time. We constrain RF to 450 trees and 4000 observations per leaf node. Our models reduce the range of divergence in path slack from 162.8ps to 23.1ps (i.e., 7× reduction).

Results of Experiment 4

We incrementally refine our models for a new design with many outliers while correlating timing parameters. A new design, 5× *jpeg_encoder*, is derived from the original *jpeg_encoder* design [318]. We create a new top module that instantiates the original *jpeg_encoder* module five times to obtain 5× *jpeg_encoder*. The new design is implemented in 28nm and has ∼300K cell instances in the post-layout netlist. We use a tighter timing constraint for this design than with *jpeg_encoder*, which results in different cells and timing paths being used. Change in top-level routing across each *jpeg_encoder* block also changes wire delay due to crosstalk effects. Therefore, 5× *jpeg_encoder* requires modification of the models derived for *jpeg_encoder*. Figure 4.15 shows the divergence between tools before and after incremental fitting for path slack, cell, wire and stage delays. The total runtime is 87 minutes. For ANN, we use up to seven hidden layers to model cell and stage delays and two hidden layers for setup time. We constrain RF to 400 trees and 2000 observations per leaf node. We do not report setup time because the divergence

is <3ps. The total runtime is 177 minutes. In the context of a new chip design project, this overhead of several hours is negligible. Our models reduce the range of divergence in path slack from 89.2ps to 36ps (2.5×), and the number of outliers from 407 to 26 (i.e., 16× reduction).

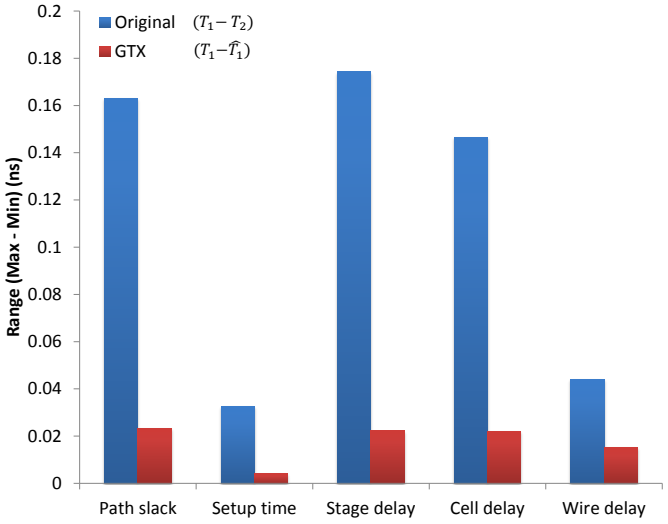


Figure 4.14: Experiment 3 results in 28nm FDSOI.

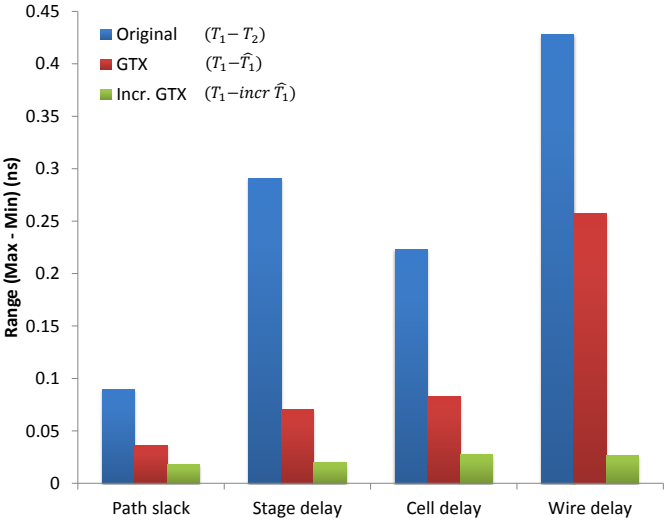


Figure 4.15: Experiment 4 results in 28nm FDSOI.

4.1.3 Conclusions

Improvements to timing signoff methodologies can significantly reduce the number of iterations in the IC design flow. Design teams often want to correlate one signoff tool's timing reports with those of another tool to reduce pessimism and/or optimism. We describe a new tool, GTX, that embodies a big-data approach for the correlation problem using a hierarchy of models. We apply machine learning to develop models for path slack, setup time, stage, cell, and wire delays and can "correct" endpoint path slack divergence between two signoff timers from 89.2ps to 22.3ps (i.e., 4× reduction) in 28nm FDSOI, and from 139.3ps to 21.1ps (i.e., 6.6× reduction) in 45nm GS with SI and OCV analysis enabled. In 7nm, GTX reduces path slack divergence from 72ps to 17ps (i.e., 4.2× reduction) with SI and OCV analysis enabled. GTX can also be applied to improve timing correlation between an implementation and a signoff tool; our experiments show 7× reduction of path slack divergence from 162.8ps to 23.1ps in 28nm FDSOI. We show that GTX scales to multiple foundry nodes and libraries, and that incremental modeling in GTX provides the capability to adapt to new designs in a given technology.

4.2 SI for Free: Machine Learning of Interconnect Coupling Delay and Transition Effects

Accurate signoff timing analysis must be conducted using signal integrity (SI) mode in signoff timing tools. According to recent reports of the analyst firm Gary Smith EDA [229], EDA vendors such as Cadence [288], CLK Design Automation [290], Incentia Design Systems [300], Mentor Graphics [312] and Synopsys [342], provide STA (Static Timing Analysis) and SI analysis tools for use in IC design. The cost of one license of a timing tool with SI mode analysis enabled is typically several times the cost of a default (with no SI analysis capability) license. In addition, the runtimes of SI-aware timing analysis are significantly larger than those of non-SI analysis. Our own studies indicate the runtime for SI-aware timing analysis on the top-10K paths can be up to 3× longer than the runtime of non-SI analysis on designs with ∼110K instances and ∼110K nets.

As would be expected, commercial signoff timing tools show significant differences between SI and non-SI modes when estimating arc delay of a stage as well as the accumulated arc delays in a path. We have studied SI and non-SI analyses with the same commercial timer, netlists, 28nm FDSOI libraries, and SPEF. For the non-SI analysis, we add twice the coupling capacitance to the ground capacitance to model worst-case Miller coupling [219]. Figure 4.16

shows that the path slack can differ by up to 81ps between SI and non-SI analyses. As reviewed in Section 2.2, no existing tool or methodology correlates non-SI timing reports to SI timing reports.

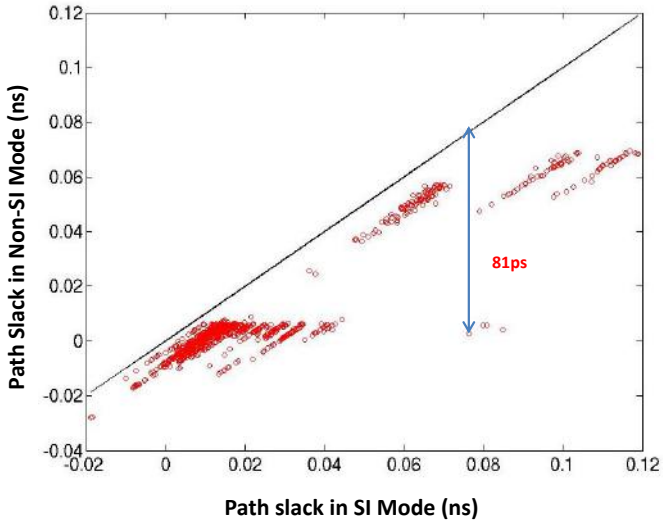


Figure 4.16: Path slack divergence in SI and non-SI analyses with clock period 1.0ns, as reported by a commercial timer in 28nm FDSOI technology.

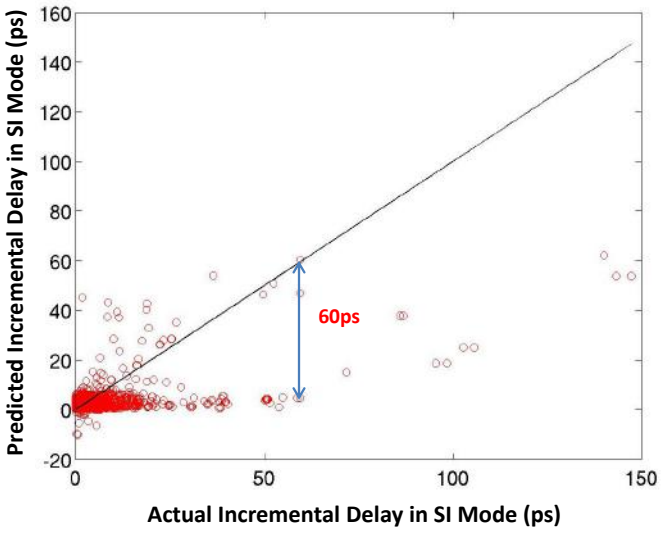


Figure 4.17: Actual incremental delay in SI mode versus predictions with clock period of 1.0ns, using models of [84].

In this work, we use machine learning techniques to estimate the incremental transition time, incremental delay due to SI, and SI-aware path delay from reports of a signoff timer that performs only non-SI analysis. Table 4.5 introduces the terminologies and notations we use in

our work. The GTX modeling work [84] described in Section 4.1 provides methodologies to calibrate non-SI to non-SI, or SI to SI, but does not attempt our present mapping of non-SI to SI. This is the gap that the present work seeks to fill. Figure 4.17 shows that the prediction of incremental delay in SI mode can be inaccurate by up to 60ps when using the wire delay model in [84] (Section 4.1) and timing reports from non-SI analysis.

Table 4.5: Terminologies and notations used in modeling to map non-SI to SI.

Term	Definition
SI mode	Timing analysis performed by enabling signal integrity
Non-SI mode	Timing analysis performed by disabling signal integrity
C_c	Coupling capacitance of an arc
C_g	Ground capacitance of an arc
C_{tot}	Total capacitance of an arc
$r_{C_c,C_{tot}}$	Ratio of coupling to total capacitance of an arc
R_w	Resistance of an arc
ΔT_{si}	Delta transition (*DTran*) time of an arc due to coupling reported in STA in SI mode
$T_{si'}$	Transition time of an arc without coupling reported in STA in non-SI mode
ΔD_{si}	Incremental SI delay (*SI Incr Delay*) of an arc due to coupling reported in STA in SI mode
$\Delta D_{si'}$	Incremental non-SI delay (*Non-SI Incr Delay*) of an arc without coupling reported in STA in non-SI mode
Path delay	Difference in arrival times at the clock pin of the launch flip-flop and D pin of the capture flip-flop
P_{si}	SI path delay across all timing arcs reported in STA in SI mode
$P_{si'}$	Non-SI path delay across all timing arcs reported in STA in non-SI mode
ΔP_{si}	Difference between $P_{si'}$ and P_{si}
$f_{C_c,red}$	Miller coupling factor in non-SI mode, i.e., $C_c \times f_{C_c,red}$ is added to C_g
f_{C_c}	Coupling capacitance factor in SI mode, i.e., C_c is changed to $C_c \times f_{C_c}$
f_{C_g}	Ground capacitance factor in SI or non-SI mode, i.e., C_g is changed to $C_g \times f_{C_g}$
f_{R_w}	Resistance factor in SI or non-SI mode, i.e., R_w is changed to $R_w \times f_{R_w}$
S	Stage in which the arc appears
N_{stg}	Number of stages in the path in which arc appears
$r_{S,N_{stg}}$	Ratio of arc-stage to total #stages in path
$clkp$	Clock period
N_{aggr}	Number of aggressors for a victim net
A_r	Toggle rate of a net
$arr_{(min,max),(r,f),(a,v)}$	Minimum (resp. maximum) rise (resp. fall) arrival time of an aggressor (resp. a victim)
LE	Logical effort of the driver of a net

Multiple parameters ranging from electrical to logic structure such as coupling capacitance, the ratio of ground and coupling capacitance of an arc, clock period, the fanin cone stage of the arc, etc. all affect the divergence of transition times and delays between SI and non-SI analyses. Complex interactions among these parameters, along with black-box code in commercial signoff timers, only make the modeling problem more difficult. For example, change in the

clock period changes toggle rates of aggressor and victim nets by different amounts that can lead to change in aggressor and victim timing window alignment. Two phenomena are particularly challenging for analytical SI delay models.

Challenge 1. Path slack variation with clock period. Figure 4.18 shows the maximum delta of slack in a path with 32 stages between SI and non-SI analyses for an OpenCores [318] design *dec_viterbi* that is signed off at 1.0ns. The delta is 81ps when the clock period varies between 0.87ns and 1.3ns. However, when the clock period decreases below 0.87ns, the maximum delta in path slack increases non-monotonically and becomes 143ps at a clock period of 0.8ns. Figure 4.19 shows timing parameters related to SI and non-SI analyses for several nets and cells. As defined in Table 4.5, "DTran" is the delta transition due to coupling, "SI Incr Delay" is the incremental delay due to coupling, "Non-SI Incr Delay" is the incremental delay without coupling, "SI Path Delay" is the accumulated path delay with coupling and "Non-SI Path Delay" is the accumulated path delay without coupling. The nets in green color do not contribute to "DTran" and "SI Incr Delay", whereas the nets in brown color cause non-zero "DTran" and "SI Incr Delay". The values in green are for the same path but analyzed at a clock period of 1.0ns The nets *n33458* and *n33452* shown in brown are responsible for large delta transition times and incremental delays in SI mode. We highlight these deltas and the impact to path slack using the blue box. The same path has a delta slack of 49ps when the clock period is 1.0ns, as shown in Figure 4.20. The path that has the maximum delta slack of 81ps at a clock period of 1.0ns continues to have the same value of delta slack at a clock period of 0.8ns, as shown in Figure 4.21.

Challenge 2. Arc delay and incremental transition time variation with ground and coupling capacitances. We illustrate non-intuitive impacts of varying ground and coupling capacitances of the victim net *n33452* on arc delay and incremental transition time respectively in Figures 4.22(a) and (b). When the ground capacitance is changed from 0.006pF to 0.0132pF, the incremental delay in non-SI mode increases from 4ps to 6ps, whereas the incremental delay due to coupling changes from 115ps to 100ps while delta transition time changes from 133ps to 147ps. The incremental delay and delta transition time in SI mode are affected in non-intuitive ways by changing the ratio of ground-to-coupling capacitance.

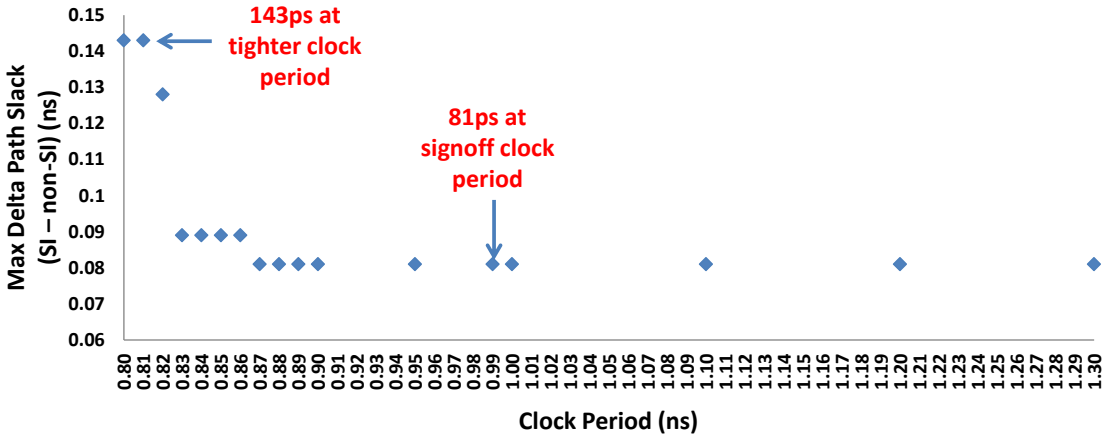


Figure 4.18: Maximum path slack delta between SI and non-SI modes over the top-1000 setup-critical paths in a design signed off at 1.0ns. The delta increases from 81ps to 143ps as the clock period is reduced below 0.87ns.

Cell / net name	DTran (ns)	SI Incr Delay (ns)	Non-SI Incr Delay (ns)	SI Path Delay (ns)	Non-SI Path Delay (ns)
inst_ram_ctrl_write_ram_fsm_reg_0_/Q	0.000	0.000	0.069	0.269	0.269
inst_ram_ctrl_write_ram_fsm_0_ (net)					
....					
FE_OCP_RBC23542_n28670/Z	0.000	0.000	0.027	0.428	0.428
FE_OCP_RBN23542_n28670 (net)					
FE_OCP_RBC23543_n28670/A	0.004	0.004	0.013	0.445	0.441
....					
U143152/Z	0.000	0.000	0.034	0.809	0.800
n33458 (net)					
U92231/C	0.003	0.002	0.000	0.811	0.801
...					
U99631/Z	0.000	0.000	0.065	0.769	0.762
n33477 (net)					
U145471/C	0.035	0.022	0.002	0.793	0.764
...					
U121581/Z	0.000	0.000	0.104	0.967	0.935
n33452 (net)					
U121579/B	0.133 (0.024)	0.115 (0.021)	0.004	1.082 (0.988)	0.939
U121579/Z	0.000	0.000	0.057	1.139 (1.045)	0.996
n79492 (net)					
inst_ram_ctrl_inst_generic_sp_ram_0_q_reg_21_/D	0.000	0.000	0.000	1.139 (1.045)	0.996

Figure 4.19: Timing divergence in a path with the maximum delta slack of 143ps at a clock period of 0.8ns.

Our contributions in this section are summarized as follows.

1. We analyze multiple sources that cause timing divergence between SI and non-SI modes and provide new insights on electrical and logic structure parameters that affect incremental transition time, incremental delay and path delay in SI mode. Unlike [84], we demonstrate that several new parameters affect *SI Incr Delay* ΔD_{si} (as defined in Table 4.5) of an arc in a timing path.

2. We develop new machine learning-based models for incremental transition time and delay due to SI, and compose these models to derive a new model for path delay that is different from [84].

Cell / net name	DTran (ns)	SI Incr Delay (ns)	Non-SI Incr Delay (ns)	SI Path Delay (ns)	Non-SI Path Delay (ns)
inst_ram_ctrl_write_ram_fsm_reg_0_/Q	0.000	0.000	0.069	0.269	0.269
inst_ram_ctrl_write_ram_fsm_0_ (net)					
....					
FE_OCP_RBC23542_n28670/Z	0.000	0.000	0.027	0.428	0.428
FE_OCP_RBN23542_n28670 (net)					
FE_OCP_RBC23543_n28670/A	0.004	0.004	0.013	0.445	0.441
....					
U143152/Z	0.000	0.000	0.034	0.809	0.800
n33458 (net)					
U92231/C	0.003	0.002	0.000	0.811	0.801
...					
U99631/Z	0.000	0.000	0.065	0.769	0.762
n33477 (net)					
U145471/C	0.035	0.022	0.002	0.793	0.764
...					
U121581/Z	0.000	0.000	0.104	0.963	0.935
n33452 (net)					
U121579/B	0.024	0.021	0.004	0.988	0.939
U121579/Z	0.000	0.000	0.057	1.045	0.996
n79492 (net)					
inst_ram_ctrl_inst_generic_sp_ram_0_q_reg_21_/D	0.000	0.000	0.000	1.045	0.996

Figure 4.20: The path with delta slack of 143ps at clock period of 0.8ns has delta slack of 49ps at clock period of 1.0ns.

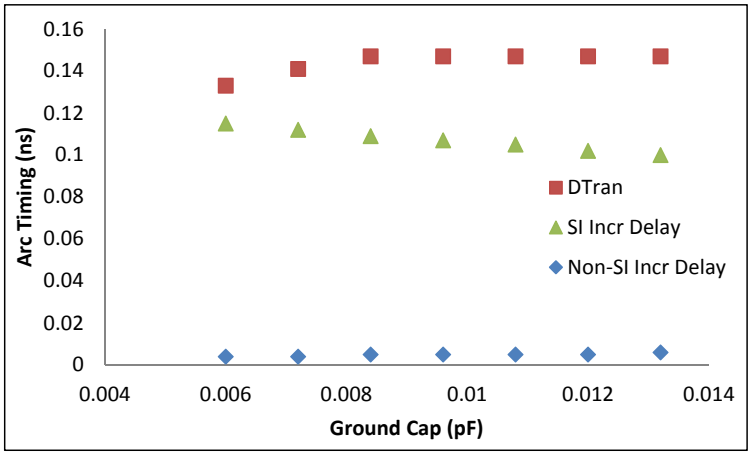
Cell / net name	DTran (ns)	SI Incr Delay (ns)	Non-SI Incr Delay (ns)	SI Path Delay (ns)	Non-SI Path Delay (ns)
inst_ram_ctrl_write_ram_ptr_reg_0_/Q	0.000	0.000	0.087	0.285	0.285
inst_ram_ctrl_write_ram_ptr_0_ (net)					
....					
FE_RC_3395_0/Z	0.000	0.000	0.015	0.424	0.424
FE_OCP_RBN22308_n20174(net)					
FE_OCP_RBN22308_n20174/A	0.014	0.009	0.019	0.452	0.443
....					
U98160/Z	0.000	0.000	0.029	0.541	0.532
n22678 (net)					
FE_OFC16-76_n22678/C	0.003	0.002	0.000	0.543	0.532
...					
U99420/Z	0.000	0.000	0.053	0.742	0.731
n25563 (net)					
U145193/C	0.016	0.012	0.000	0.754	0.731
U145193/Z	0.000	0.000	0.114	0.868	0.845
n25556 (net)					
U89670/B	0.089	0.058	0.006	0.932	0.851
...					
U121246/Z	0.000	0.000	0.021	1.063	0.982
n70246 (net)					
inst_ram_ctrl_inst_generic_sp_ram_1_q_reg_18_/D	0.000	0.000	0.000	1.063	0.982

Figure 4.21: Timing divergence in a path with delta slack of 81ps at clock periods of both 1.0ns and 0.8ns.

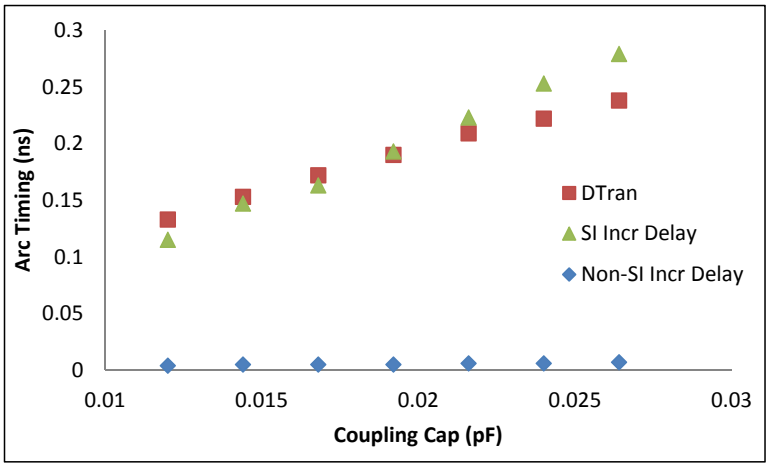
3. The worst-case absolute errors in our modeling predictions of incremental transition time, incremental delay due to SI and SI-aware path delay are 7.0ps, 5.2ps and 8.2ps, respectively. We have developed and tested our models using timing reports of block implementations in 28nm FDSOI foundry libraries. Compared to the recent work of [84], we reduce worst-case error in prediction of incremental delay due to SI changes from 60ps to 5.2ps.

4.2.1 Methodology for Timing Correlation in SI Mode

Our modeling methodology includes (i) selection of parameters that affect incremental delay in SI mode, and (ii) application of nonlinear modeling techniques to capture the complex interactions of parameters so as to accurately predict the incremental delay in SI mode.



(a)



(b)

Figure 4.22: Timing of the victim net that has the maximum divergence at a clock period of 0.8ns when only (a) ground capacitance and (b) coupling capacitance of the victim net is varied. The figure shows delta transition due to coupling as "DTran" in brown rectangles, arc delay due to coupling as "SI Incr Delay" in green triangles and arc delay without coupling as "Non-SI Incr Delay" in blue diamonds.

Selection of Parameters

We have studied multiple electrical and circuit parameters that can affect incremental delay in SI mode and have drawn from the list of parameters used to model wire delay in SI mode in [84]. Our analyses indicate that the transition time at the output pin of a net's driver, the product of wire resistance and capacitances, are not sufficient to predict the incremental delay in SI mode. Figures 4.23(a) and (b) show that the incremental delay in SI mode vary in the same way for two of the parameters used in [84]. In addition, signoff timing tools use complex

algorithms to determine timing windows for less pessimistic delay analyses in SI mode. This is difficult to model because timing windows change with operating conditions. We introduce new electrical parameters to approximate the effect of timing windows for the aggressor with the largest coupling capacitance. Figures 4.24(a)–(d) show two new electrical and two new structural parameters that affect the incremental delay in SI mode.

We use the following 12 parameters in our modeling: (i) incremental delay in non-SI mode; (ii) transition time in non-SI mode; (iii) clock period; (iv) resistance; (v) coupling capacitance; (vi) ratio of coupling-to-total capacitance; (vii) toggle rate; (viii) number of aggressors; (ix) ratio of the stage in which the arc of the victim net appears to the total number of stages in the path; (x) logical effort of the net's driver; and (xi), (xii) the differences in max (respectively, min) arrival times[60] of the signal at the driver's output pin for the victim and its strongest aggressor.[61] We choose our parameters based on sensitivity of the parameter to incremental transition time or incremental delay due to SI, or SI-aware path delay. Our experimental results indicate that dropping any of the parameters can reduce the modeling accuracy by at least 5%. Therefore, we use all the parameters indicated in Equations (4.8), (4.9) and (4.10) to develop our models. We do not use any layout parameters since layout is reflected in parameters such as coupling capacitance, total capacitance and wire resistance.

We model the incremental transition time due to SI as

$$\Delta T_{si} \quad = f(T_{si'}, R_w, C_c, r_{C_c,C_{tot}}, clkp, LE). \tag{4.8}$$

We further model the incremental delay due to SI as

$$\begin{aligned} \Delta D_{si} \quad = f(&\Delta D_{si'}, \Delta T_{si}, R_w, C_c, r_{C_c,C_{tot}}, r_{S,N_{stg}}, clkp, \\ &\Delta arr_{min,(r,f)}, \Delta arr_{max,(r,f)}, A_r, LE) \end{aligned} \tag{4.9}$$

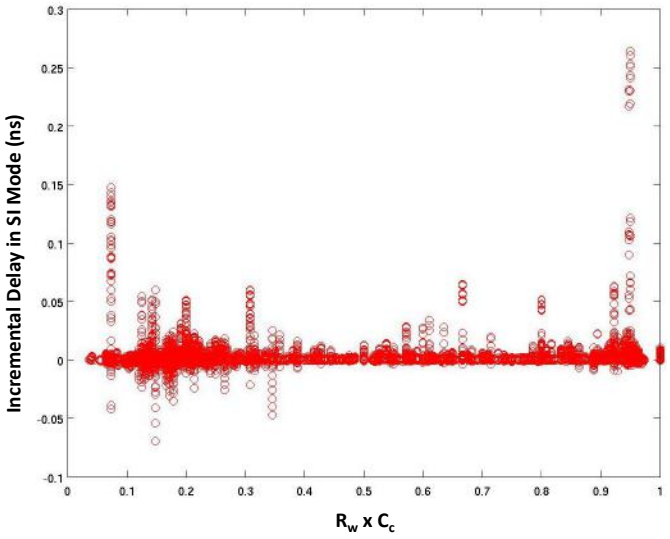
and the SI-aware path delay as

$$\Delta P_{si} \quad = f(P_{si'}, \textstyle\sum_{i=1}^{N_{stg}} \Delta D_{si}, N_{stg}) \tag{4.10}$$

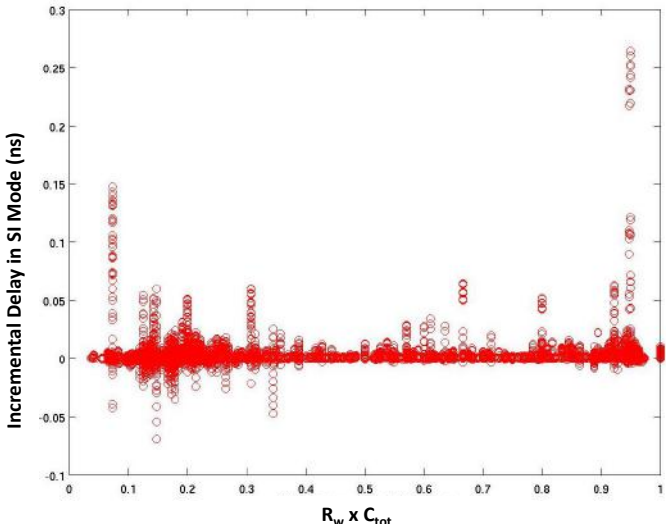
where ΔD_{si} is the predicted incremental delay due to SI per arc, obtained from the model developed using Equation (4.9).

[60] We use rise and fall arrival times based on the signal's transition at the output pin of the net's driver, from timing reports in non-SI mode.

[61] We consider the net with largest coupling capacitance to the victim as the strongest aggressor.



(a)



(b)

Figure 4.23: Incremental delay due to SI varies in the same way as (a) $R_w \times C_c$ and (b) $R_w \times C_{tot}$.

Nonlinear Modeling Technique

If the coupling capacitance is zero, we set the incremental delay due to SI as zero; otherwise, we proceed with modeling. We use nonlinear modeling techniques to model the incremental transition time, incremental delay due to SI, and SI-aware path delay, given the complex interactions between modeling parameters described above. For example, reducing the clock period can increase the toggle rates of both the victim and aggressor nets, and can change

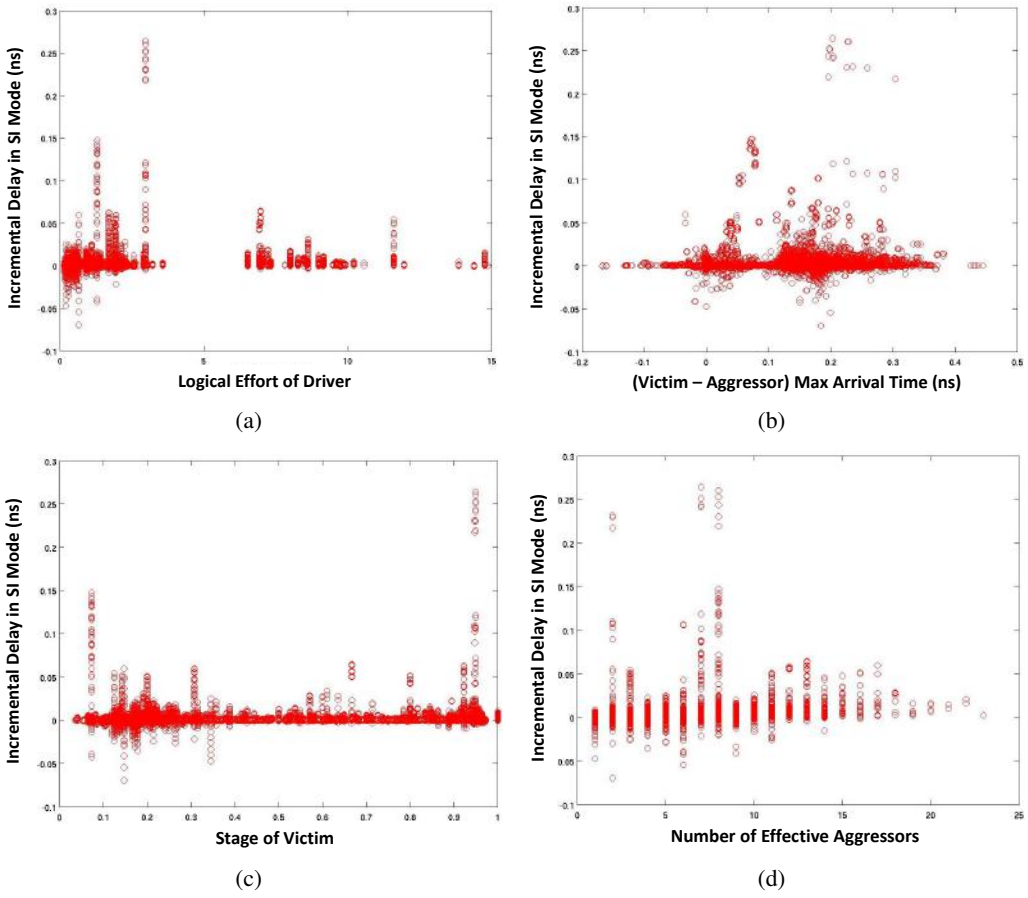


Figure 4.24: Incremental delay due to SI varies with (a) the logical effort of the net's driver, (b) the difference in max arrival times of victim and the strongest aggressor, (c) the stage in which the arc appears, and (d) the number of effective aggressors of the victim net.

the timing windows. As a result, the number of aggressors on the victim can increase. These interactions are non-obvious and cannot be captured by linear modeling techniques. We therefore use Artificial Neural Networks (ANN) and Support Vector Machines regression (SVM) [85] for our modeling.

We use Hybrid Surrogate Modeling (HSM) [122] to combine the predicted values from the ANN and SVM models. HSM is a variant of [76], which obtains improved estimates by finding weighted combinations of estimates from individual surrogate models. In this technique the response is estimated by adding weights to the estimated response from each of the surrogate models. We express this formally as

$$\hat{y}(\vec{x}) = w_1 \cdot \hat{y}(\vec{x})_{ANN} + w_2 \cdot \hat{y}(\vec{x})_{SVM} \tag{4.11}$$

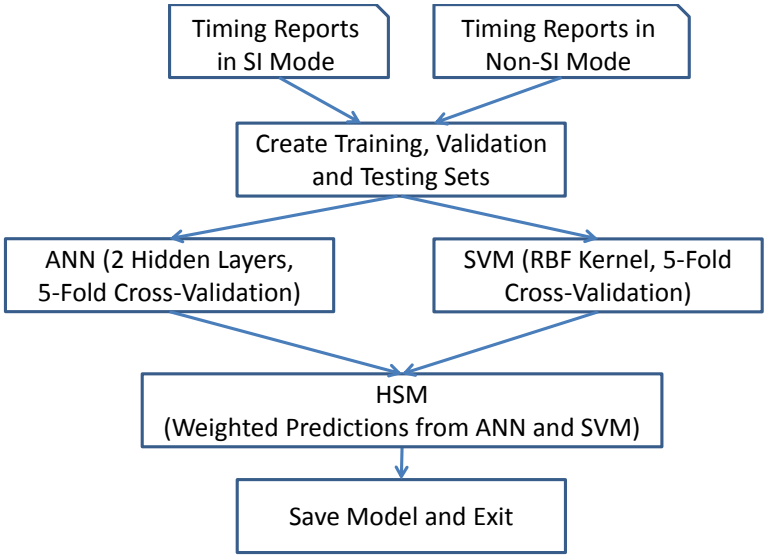


Figure 4.25: Modeling flow to map non-SI to SI using nonlinear modeling techniques.

where $y(\vec{x})_{ANN}$ is the estimated response of ANN, $\hat{y}(\vec{x})_{SVM}$ is the estimated response of SVM, and w_1 and w_2 are the weights of estimated responses of ANN and SVM respectively. We use least-squares regression to fit the hybrid model to 75% of the training data points which are randomly selected [85]. We perform cross-validation on the remaining 25% of the training data points to estimate the cross-validation error. The fitting is repeated 10 times. The weights that give the minimum cross-validation error out of these 10 tries are used to generate the hybrid surrogate model.

In ANN, we use one input, one output and two hidden layers. In each hidden layer, we use up to twice the number of neurons as the number of input parameters. We search for the minimum number of neurons per hidden layer that can achieve the smallest mean-squared error on the training set. In SVM, we use the Radial Basis Function (RBF) kernel with a gamma value of the inverse of the number of the parameters. To generalize our models and avoid overfitting, we use five-fold cross validation and use a separate validation set to reduce overfitting while training our models. For each technique (ANN, SVM and HSM), we create one model for $N_{stg} \leq 20$ and another model for $N_{stg} > 20$, as our separate studies indicate that modeling accuracy improves with this approach.

4.2.2 Experimental Setup and Results

We now describe our design of experiments, i.e., our testcases, methodology to generate "ground truth", and tool settings. We then describe our modeling results.

Design of Experiments

In our experiments, we use six real designs (*aes_cipher_top*, *dec_viterbi*, *jpeg_encoder* and *THEIA* from OpenCores [318]; *FIFO* from Synopsys *Designware* [337]; and single core of OpenSPARC T2 *spc* [317]) as well as artificial testcases developed in-house based on [84]. An illustration of our artificial testcase is shown in Figure 4.26. We use 28nm FDSOI foundry technology libraries for all our experiments. We vary parasitics, i.e., R_w, C_c, C_g, size of the driver, type of the driver cell, the number of fanouts, clock period, etc. We use default values of 1Ω for R_w, 1fF for C_c and C_g and use scaling factors f_{R_w}, f_{C_c} and f_{C_g} to respectively scale R_w, C_c and C_g in both real designs and artificial testcases.

We use one implementation of the *aes_cipher_top* design signed off at 1.0ns (∼13K standard cells at post-synthesis), one implementation of the *dec_viterbi* design signed off at 1.0ns (∼97K standard cells at post-synthesis), one implementation of the *jpeg_encoder* design signed off at 0.8ns (∼62K standard cells at post-synthesis), one implementation of the *FIFO* design signed off at 0.75ns (∼6.5K standard cells at post-synthesis), one implementation of the *THEIA* design signed off at 2.0ns (∼125K standard cells at post-synthesis) and one implementation of the *spc* design signed off at 2.2ns (∼350K standard cells at post-synthesis). Table 4.6 lists the ranges of various parameters that we sweep in our experiments.

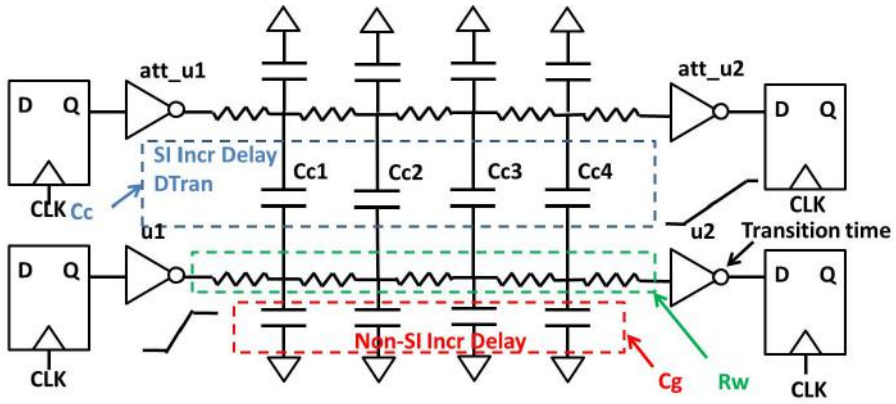


Figure 4.26: Illustration of an artificial testcase instance.

To generate "ground truth" data, we perform path-based setup timing analyses in both SI

Table 4.6: Key parameters swept in our experiments.

Parameter	Range	Design/Testcase
clkp	1.0ns + {-0.2, 0.2}ns	*aes_cipher_top*
	1.0ns + {-0.2, -0.1, 0.0, 0.1, 0.2}ns	*dec_viterbi*, artificial
	0.8ns + {-0.2, -0.1, 0.0, 0.1}ns	*jpeg_encoder*
	0.75ns + {-0.15, 0.15}ns	*FIFO*
	2.0ns + {-0.2, 0.2}ns	*THEIA*
	2.2ns + {-0.2, 0.2}ns	*spc*
N_{stg}	{15, 20, 25, 30}	artificial
$f_{C_c,red}$	{0.0, 1.0, 2.0}	all
f_{R_w}	{0.5, 1.0, 2.0}	all
f_{C_c}	{0.5, 1.0, 1.5, 2.0}	all
f_{C_g}	{0.5, 1.0, 2.0}	all
Driver size	{X6, X16, X24, X32}	artificial

and non-SI modes and report the top-1000 critical paths. In non-SI mode, we use $f_{C_c,red}$ values of 0.0, 1.0 and 2.0 to capture the following effects of victim and aggressor nets switching: (i) a value of 0.0 when the victim and aggressor switch in the same direction; (ii) a value of 1.0 when the victim does not switch but the aggressor switches; and (iii) a value of 2.0 when the victim and aggressor switch in the opposite directions. In SI mode, we use recommended tool settings for the most accurate (least pessimistic) analysis, which include (i) disabling of critical path reselection so that all aggressors are selected for analysis at all times for all victims; (ii) enabling the clock network for analysis so as to include coupling effects of clock nets on victim signal nets; and (iii) performing analysis in edge-alignment mode so as to consider all possible edge arrivals from the upstream logic, using minimum-delay (respectively, maximum-delay) edges for the minimum (respectively, maximum) incremental delay calculations.

Following are steps used for timing analysis in SI and non-SI mode. We specifically highlight the differences in SI versus non-SI mode, if any, in each of the steps.

- *Step 1.* Read databases of timing libraries.
- *Step 2.* Read and link the design; the post-layout netlist is in *.v* format.
- *Step 3.* Read the constraints specified in the SDC.
- *Step 4.* Read the parasitics specified in the SPEF. In SI mode, read the coupling capacitances, whereas in non-SI mode convert coupling capacitances to ground capacitances by using Miller coupling factor $f_{C_c,red}$.

- *Step 5.* (SI mode only) Set flag to reselect critical paths for SI analysis to false.
- *Step 6.* (SI mode only) Set flag to reselect clock nets for SI analysis to true.
- *Step 7.* (SI mode only) Set flag for delay analysis mode to be edge-aligned.
- *Step 8.* Perform path-based timing analysis of specified top-1000 paths of the signed off design.
- *Step 9.* Report capacitance, incremental delay, transition time, accumulated stage delay of all cells and nets in the top-1000 paths. In SI mode, report incremental delay and transition time due to coupling.

We generate a total of 188K data points of nets that have non-zero value of incremental SI delay, out of which we use 60% for training, 10% for validation and the remaining 30% for testing. The training time of our models is 10.6 hours for ANN, 23.9 hours for SVM and 12 minutes for HSM on an Intel Xeon E5-2640 2.5GHz server with eight threads. This is a one-time overhead. After the models are trained, the time to test is $\sim$10 minutes for every 10K data points.

We conduct three experiments to demonstrate accuracy and robustness of our models.

- **Experiment 1. (Accuracy)** Predict incremental transition time, incremental delay and path delay due to SI using a model derived from non-SI timing reports of a signoff timing tool.
- **Experiment 2. (Robustness)** Predict incremental delay due to SI on "unseen" data points from a new implementation of *jpeg_encoder*. The new implementation of *jpeg_encoder* uses different signoff and layout constraints as compared to the implementation (cf. Table 4.6) used to train the models.
- **Experiment 3. (Accuracy)** Compare the predictions of incremental delay and path delay due to SI of our models versus those of [84].

In our results, we compare path delay instead of path slack because the delta in slack arises due to differences in path delay. The required arrival times calculated in both SI and non-SI modes are the same because elements such as clock uncertainty, clock skew, and setup time of the capture flip-flop do not vary with coupling. Only the arrival times vary due to incremental delay in SI and non-SI modes. Therefore, the errors in correlating path slack will be the same as the errors observed in correlating path delay. We report predicted values of transition time

and incremental delay due to SI and SI-aware path delay only on the test dataset, that is, we do not include the training and validation datasets in reporting results in Experiments 1 and 2. We calculate percentage error in predicting incremental delay and transition time due to SI in an arc and SI-aware path delay as follows.

$$\textbf{Error}_{arc} = \frac{\textbf{(Predicted – Actual)}\ \Delta T_{si}\ \textbf{or}\ \Delta D_{si}}{\textbf{Actual}\ \Delta T_{si}\ \textbf{or}\ \Delta D_{si}} \tag{4.12}$$

$$\textbf{Error}_{path} = \frac{\textbf{(Predicted – Actual)}\ \Delta P_{si}}{\textbf{Actual}\ \Delta P_{si}} \tag{4.13}$$

Results of Experiment 1

The goal of this experiment is to validate our modeling accuracy in predicting incremental transition time, incremental delay due to SI and SI-aware path delay. Our models are developed by using timing reports in non-SI mode. We test the accuracy of our models by using ∼17K data points for incremental transition time and incremental delay and ∼320 paths for SI-aware path delay, across real designs and artificial testcases.

Figure 4.27 shows actual versus predicted incremental transition times due to SI. Our modeling predictions have a worst-case absolute error of 7.0ps (8.8%)[62] and have a range of errors of 11.3ps. Our average absolute error in predicting incremental transition time is 0.7ps (0.6%). Figure 4.28 shows actual versus predicted incremental delays due to SI. Our modeling predictions have a worst-case absolute error of 5.2ps (15.7%) and have a range of errors of 9.8ps. Our average absolute error in predicting incremental delay is 1.2ps (1.1%).

Figure 4.29 shows actual versus predicted SI-aware path delays. Our modeling predictions have a worst-case absolute error of 8.2ps (6.9%),[63] i.e., our worst-case absolute error in predicting path slack is also 8.2ps. The average absolute error in predicting path delay is 1.7ps (1.4%). Figure 4.30 shows the actual and predicted values of incremental delay and path delay in SI mode of the same path as shown in Figure 4.19. The path slack divergence between SI and non-SI modes of 143ps is reduced to 5ps by our models.

[62] In non-SI and SI modes the transition times are 34.6ps and 114.6ps, respectively. The actual incremental transition time due to SI is $114.6 - 34.6 = 80$ps, whereas our model for incremental transition time predicts 73ps. The difference is 7.0ps. Therefore, per Equation (4.12), the percentage error is $7.0/80 = 8.8\%$.

[63] In non-SI and SI modes the path delays are 1055.2ps and 935.5ps, respectively. The actual difference in SI-aware path delay is $1055.2 - 935.5 = 119.7$ps, whereas our model for SI-aware path delay predicts 109.6ps. The difference is 8.2ps. Therefore, per Equation (4.13), the percentage error is $8.2/119.7 = 6.9\%$.

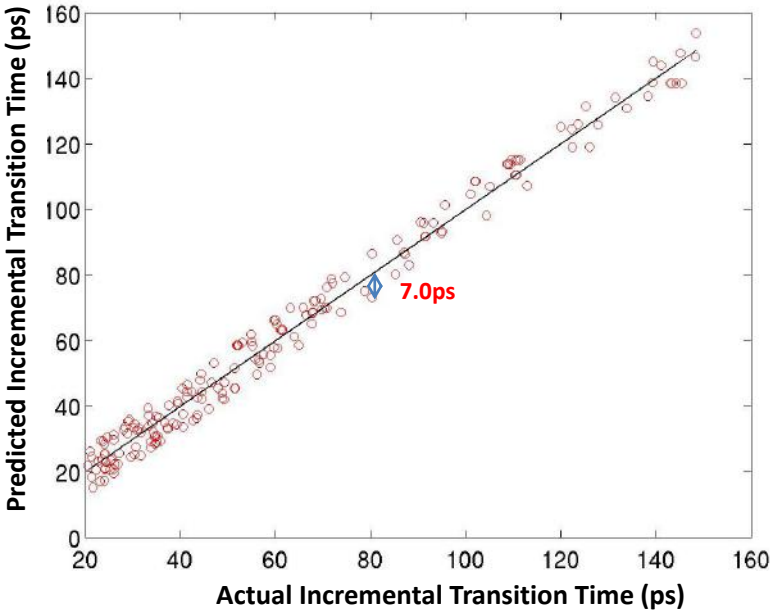


Figure 4.27: Actual versus predicted incremental transition times due to SI.

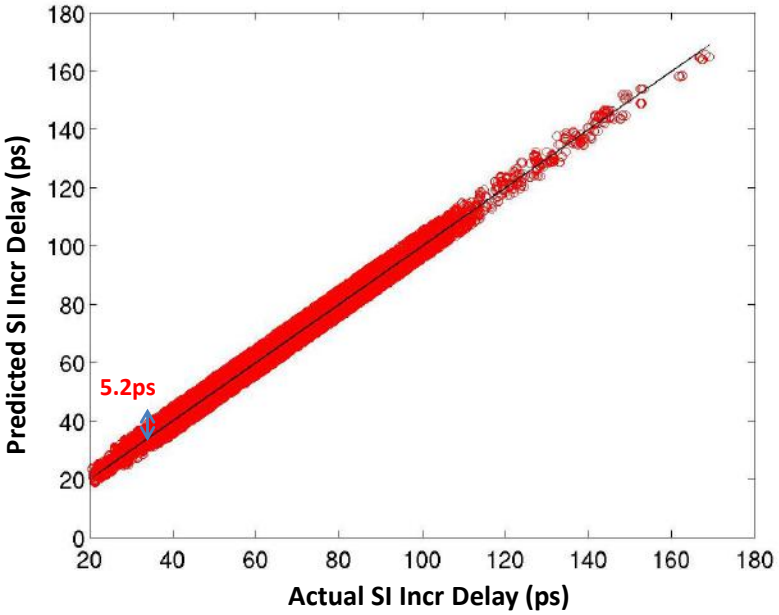


Figure 4.28: Actual versus predicted incremental delays due to SI.

Results of Experiment 2

The goal of this experiment is to validate the robustness of our models and stress-test our models on "unseen" data points. We train our models using data points from our design

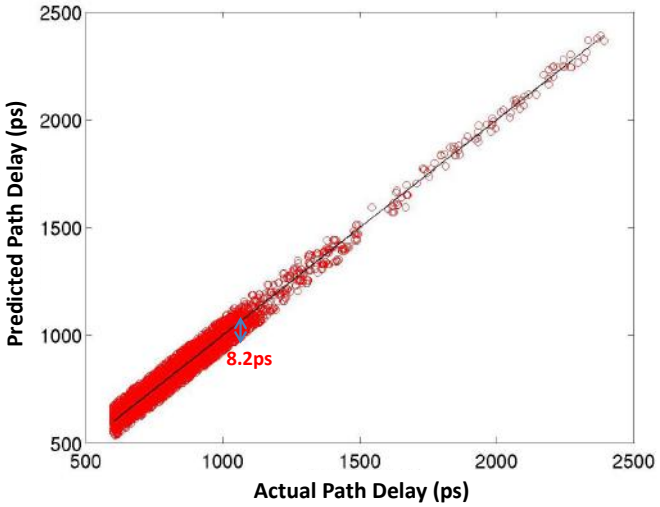


Figure 4.29: Actual versus predicted SI-aware path delays.

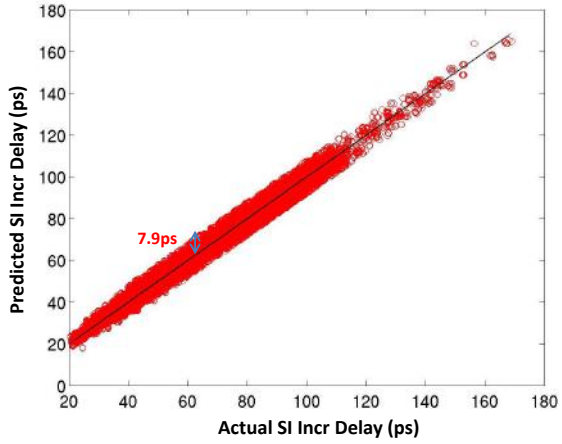
Cell / net name	(Actual) SI Incr Delay (ns)	(Model) SI Incr Delay (ns)	(Actual) SI Path Delay (ns)	(Model) SI Path Delay (ns)
inst_ram_ctrl_write_ram_fsm_reg_0_/Q	0.000	0.000	0.269	0.269
inst_ram_ctrl_write_ram_fsm_0_ (net)				
....				
FE_OCP_RBC23542_n28670/Z	0.000	0.000	0.428	0.428
FE_OCP_RBN23542_n28670 (net)				
FE_OCP_RBC23543_n28670/A	0.004	0.004	0.445	0.445
....				
U143152/Z	0.000	0.000	0.809	0.809
n33458 (net)				
U92231/C	0.002	0.002	0.811	0.811
...				
U99631/Z	0.000	0.000	0.769	0.769
n33477 (net)				
U145471/C	0.022	0.023	0.793	0.794
...				
U121581/Z	0.000	0.000	0.967	0.968
n33452 (net)				
U121579/B	0.115	0.118	1.082	1.086
U121579/Z	0.000	0.000	1.139	1.140
n79492 (net)				
inst_ram_ctrl_inst_generic_sp_ram_0_q_reg_21_/D	0.000	0.000	1.139	1.144

Figure 4.30: Actual and predicted values of "SI Incr Delay" and "SI Path Delay" (defined in Table 4.5) of the same path shown in Figure 4.19. Our models reduce the path delay (as well as path slack) divergence from 143ps to 5ps. The predicted values that differ from the actual values are highlighted in red.

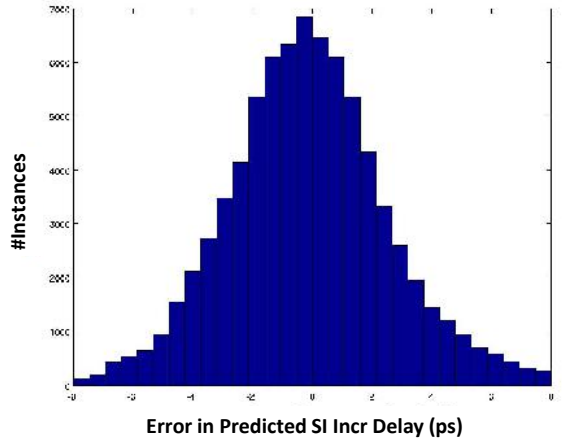
of experiments described above, and test the models using unseen data points from a new implementation of *jpeg_encoder* signed off with clock period 1.0ns, tighter maximum transition constraint of 150ps and utilization of 55%. The implementation used for testing is signed off at different clock period, has different mixes of cell types, number of stages per path, net parasitics, etc. as compared to the implementation (cf. Table 4.6) used to train our models. However, as we include important parameters that affect incremental transition time, incremental delay due to

SI, and SI-aware path delay, we expect that our models can be generalized to unseen data points in the same 28nm FDSOI foundry technology. Figure 4.31(a) shows actual and predicted values of incremental delay in SI mode for 2.5K unseen data points. The worst-case absolute error in prediction is 7.9ps (12.3%), however, the average absolute error is 1.6ps (2.6%). Figure 4.31(b) shows the distribution of errors across all test data points.

We have conducted additional experiments in a 7nm foundry technology to stress-test our models on unseen data points. We train our models on data points from *aes_cipher_top* and ARM *Cortex M0* designs and test on data points from the *leon3mp* design. Figure 4.32(a) shows that the divergence in path slack values between non-SI and SI is up to 97ps. Figure 4.32(b) shows that by using only non-SI timing reports and applying our models, we are able to reduce the divergence in path slack from 97ps to 12ps.

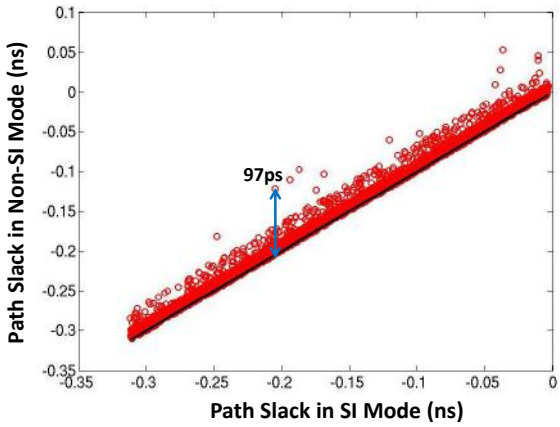


(a)

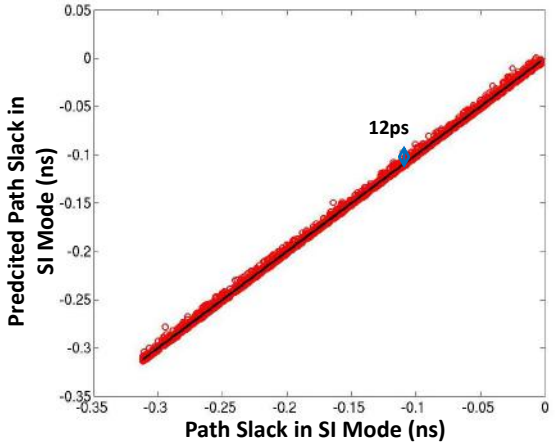


(b)

Figure 4.31: Robustness of our models in predicting incremental delays due to SI. (a) Actual versus predicted and (b) distribution of modeling errors.



(a)



(b)

Figure 4.32: Robustness of our models in predicting path slack values in SI mode in a 7nm technology. (a) SI mode versus non-SI mode, and (b) predicted versus actual path slack values in SI mode.

Results of Experiment 3

In this experiment we compare the accuracy of our models, versus that of the wire and path delay models in [84] that predict SI-aware path delay. We develop these models for wire and path delay using timing reports in non-SI mode. Recall that Figure 4.17 in Section 4.2 shows that the worst-case error in predicting incremental arc delay due to SI using the model in [84] can be as large as 60ps. Figure 4.33 shows that the worst-case error in path delay can be 87.3ps using the model in [84]. From results of Experiment 1 above, our models have worst-case errors of 5.2ps and 8.2ps in predicting incremental delay due to SI, and SI-aware path delay, respectively.

The models of [84] have large prediction errors in spite of using a layered modeling

approach. We attribute this to underfitting, with the parameters used in [84] being insufficient to capture fully the variations in incremental delay due to SI, and SI-aware path delay.

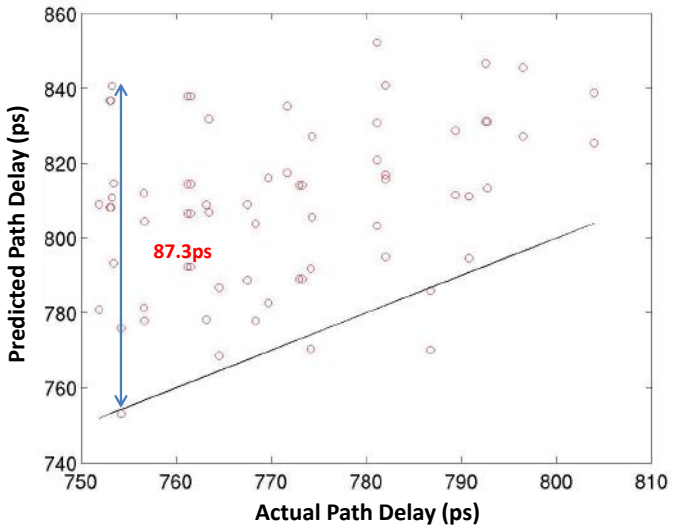


Figure 4.33: Actual SI-aware path delays versus predicted path delays using models of [84].

4.2.3 Conclusions

In this work, we analyze electrical and logic structure parameters that cause timing in non-SI mode to diverge from that in SI mode. We provide a machine learning-based methodology that can accurately model incremental delay due to SI, and SI-aware path delay. Our models for a 28nm FDSOI production technology and cell library have worst-case errors of 7.0ps, 5.2ps and 8.2ps, respectively in predicting incremental transition time, incremental delay due to SI, and SI-aware path delay. Our models for a 7nm foundry technology and cell library have a worst-case error of 12ps in predicting path slack values. We demonstrate that our models are more accurate than previous work [84]. Future works may include (i) predicting timing reports in path-based analysis using reports from graph-based analysis, and (ii) integrating our models with an academic timer [351].

4.3 An Optimization Framework for MMMC Clock Skew Variation Reduction

Modern systems-on-chip (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, Vt-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).

Several previous works optimize skew at one or more (process, voltage, temperature) (PVT) corners, but do not address skew variation across corners. Cao et al. [31] 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. [48] 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. [166] perform 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 in 130nm, 90nm and 65nm, and conclude that the correlation across corners is linear. However, such an assumption might not be valid in 28nm and below. Lung et al. [167] 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. [209] propose a two-dimensional nontree structure. They divide the nontree structure 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 [236] use mesh structures for the top-level tree which consumes less wire resource and power as compared to [209]. 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. [205] [206] propose a nontree construction method to insert crosslinks[64] 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. Hu et al. [93] propose to insert crosslinks in a tree, and achieve up to 9% clock skew and 25% clock skew variation reductions. Mittal and Koh [177] propose a greedy method to insert crosslinks to reduce skew variation.

Although many commercial EDA tools are capable of multi-mode multi-corner clock network synthesis [237] [340], 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.

The main contributions of this section 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.

[64] A crosslink is an additional wire between any two nodes of a given clock tree. When inserted into a clock tree, it creates a loop and hence a nontree topology.

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.

4.3.1 Problem Formulation and Optimization Framework

The notations we use in this work are given in Table 4.7.

Table 4.7: Description of notations used in our work on clock skew variation minimization.

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

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

$$v_{i,i'}^{c_k,c_{k'}} = |\alpha_k \cdot skew_{i,i'}^{c_k} - \alpha_{k'} \cdot skew_{i,i'}^{c_{k'}}| \tag{4.14}$$

where skew ($skew_{i,i'}^{c_k}$) is defined as the latency difference between capture and launch clock paths at c_k. We emphasize that our optimization is local skew-aware, so that we only optimize skews between launch-capture sink pairs that have valid datapaths in between them (i.e., we avoid the pessimism that would result from use of global skew in the formulation). α_k is the normalization factor at corner c_k with respect to the nominal corner. Note that α_k is an input parameter and can be determined by technology information (e.g., ratio between buffer delays at c_k and c_0), clock tree properties (e.g., Vt and sizes of buffers in the tree), etc. Further, one can define specific α_k values for each sink pair. In our work, we define α_k as the average skew ratio between c_0 and c_k over all sink pairs.

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

$$V_{i,i'} = \max_{\forall(c_k,c_{k'})} v_{i,i'}^{c_k,c_{k'}}. \tag{4.15}$$

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.

$$\text{Minimize} \sum_{\forall(f_i,f_{i'})} V_{i,i'} \tag{4.16}$$

Figure 4.34 illustrates our optimization framework. We perform global and local optimizations to reduce skew variations. *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 and is the focus of this discussion. 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 another stopping condition is reached.

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

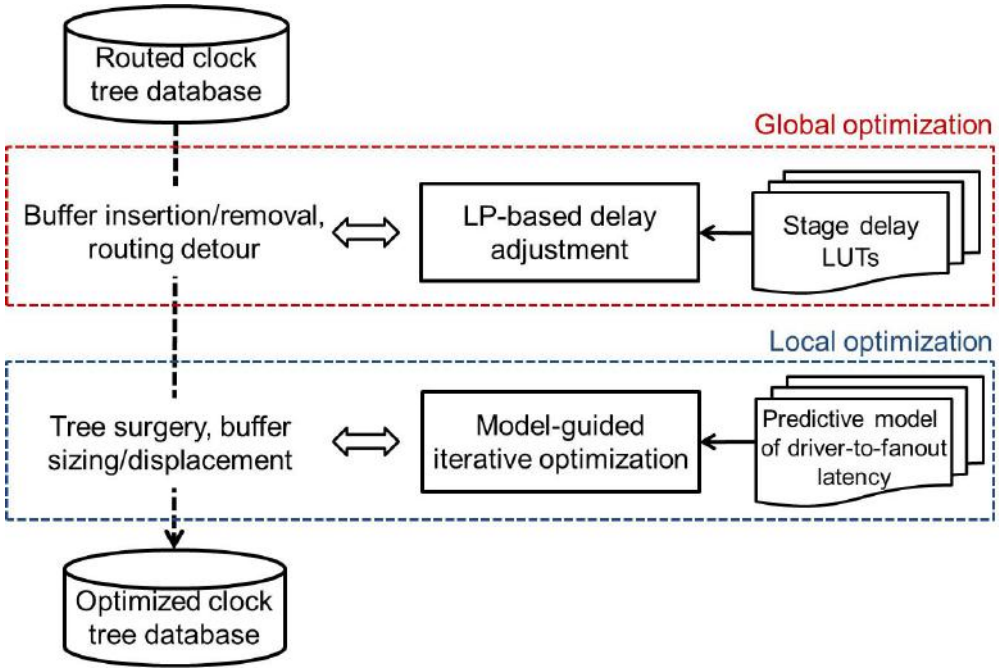


Figure 4.34: Overview of our clock skew variation optimization framework.

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:

$$\text{Minimize} \sum_{1 \le j \le M,\ 0 \le k \le K} |\Delta_j^{c_k}| \tag{4.17}$$

where $\Delta_j^{c_k}$ is the latency change on arc s_j at corner c_k.[65] The upper bound U on the sum of skew variations is specified as

$$\sum_{(f_i,f_{i'})} V_{i,i'} \leq U \tag{4.18}$$

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.

$$\begin{aligned}
V_{i,i'} \geq & \alpha_k \cdot \Big(\sum_{s_{j'} \in P_{i'}} (D_{j'}^{c_k} + \Delta_{j'}^{c_k}) - \sum_{s_j \in P_i} (D_j^{c_k} + \Delta_j^{c_k}) \\
& - \alpha_{k'} \cdot \Big(\sum_{s_{j'} \in P_{i'}} (D_{j'}^{c_{k'}} + \Delta_{j'}^{c_{k'}}) - \sum_{s_j \in P_i} (D_j^{c_{k'}} + \Delta_j^{c_{k'}}) \Big) \\
V_{i,i'} \geq & \alpha_{k'} \cdot \Big(\sum_{s_{j'} \in P_{i'}} (D_{j'}^{c_{k'}} + \Delta_{j'}^{c_{k'}}) - \sum_{s_j \in P_i} (D_j^{c_{k'}} + \Delta_j^{c_{k'}}) \Big) \\
& - \alpha_k \cdot \Big(\sum_{s_{j'} \in P_{i'}} (D_{j'}^{c_k} + \Delta_{j'}^{c_k}) - \sum_{s_j \in P_i} (D_j^{c_k} + \Delta_j^{c_k}) \Big)
\end{aligned} \tag{4.19}$$

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

$$\begin{aligned}
\sum_{s_{j'} \in P_{i'}} (D_{j'}^{c_k} + \Delta_{j'}^{c_k}) - \sum_{s_j \in P_i} (D_j^{c_k} + \Delta_j^{c_k}) \leq & \Big| \sum_{s_{j'} \in P_{i'}} D_{j'}^{c_k} - \sum_{s_j \in P_i} D_j^{c_k} \Big| \\
\sum_{s_j \in P_i} (D_j^{c_k} + \Delta_j^{c_k}) - \sum_{s_{j'} \in P_{i'}} (D_{j'}^{c_k} + \Delta_{j'}^{c_k}) \leq & \Big| \sum_{s_{j'} \in P_{i'}} D_{j'}^{c_k} - \sum_{s_j \in P_i} D_j^{c_k} \Big|
\end{aligned} \tag{4.20}$$

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

$$
\begin{aligned}
&\alpha_k \cdot (\sum_{s_{j'} \in P_{i'}} (D_{j'}^{c_k} + \Delta_{j'}^{c_k}) - \sum_{s_j \in P_i} (D_j^{c_k} + \Delta_j^{c_k})) \\
&- \sum_{s_{j'} \in P_{i'}} (D_{j'}^{c_0} + \Delta_{j'}^{c_0}) - \sum_{s_j \in P_i} (D_j^{c_0} + \Delta_j^{c_0}) \\
&\leq |\alpha_k \cdot (\sum_{s_{j'} \in P_{i'}} D_{j'}^{c_k} - \sum_{s_j \in P_i} D_j^{c_k}) - (\sum_{s_{j'} \in P_{i'}} D_{j'}^{c_0} - \sum_{s_j \in P_i} D_j^{c_0})| \\
&\sum_{s_{j'} \in P_{i'}} (D_{j'}^{c_0} + \Delta_{j'}^{c_0}) - \sum_{s_j \in P_i} (D_j^{c_0} + \Delta_j^{c_0}) \\
&- \alpha_k \cdot (\sum_{s_{j'} \in P_{i'}} (D_{j'}^{c_k} + \Delta_{j'}^{c_k}) - \sum_{s_j \in P_i} (D_j^{c_k} + \Delta_j^{c_k})) \\
&\leq |\alpha_k \cdot (\sum_{s_{j'} \in P_{i'}} D_{j'}^{c_k} - \sum_{s_j \in P_i} D_j^{c_k}) - (\sum_{s_{j'} \in P_{i'}} D_{j'}^{c_0} - \sum_{s_j \in P_i} D_j^{c_0})|
\end{aligned} \tag{4.21}
$$

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

$$
\sum_{s_j \in P_i} (D_j^{c_k} + \Delta_j^{c_k}) \leq D_{max} \tag{4.22}
$$

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

$$
D_{min\ j}^{c_k} \leq D_j^{c_k} + \Delta_j^{c_k} \leq \beta \cdot D_j^{c_k} \tag{4.23}
$$

Complexity analysis. The LP formulation has $O(M \cdot K)$ variables to indicate delay change on each arc at each corner ($\Delta_j^{c_k}$); there are also $O(N^2)$ (i.e., 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 (4.19)); $(4 \cdot K)$ constraints to prevent local skew and skew variation degradations (Constraints (4.20)-(4.21)); N constraints to specify the maximum latency (Constraint (4.22)); $(2 \cdot M)$ constraints to bound arc delay changes (Constraint (4.23)); and $C(K,2)$ constraints to enhance ECO feasibility (Constraint (4.24)).

$$
W_{min}^{c_k,c_{k'}} \leq \frac{D_j^{c_k} + \Delta_j^{c_k}}{D_j^{c_{k'}} + \Delta_j^{c_{k'}}} \leq W_{max}^{c_k,c_{k'}} \tag{4.24}
$$

Further details of the lookup table characterization, and of the LP-guided ECO flow that

implements solutions from the global optimization in a commercial CTS tool, are beyond the scope of this thesis. Interested readers may find these details in [83].

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 4.35(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.[66] 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 [51] 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 [133] and update gate delays one and two stages downstream based on Liberty tables.[67] 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 its wire delay can have discrepancies with respect to the commercial router's actual ECO solution. We therefore construct machine learning-based models to minimize such discrepancies.

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

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

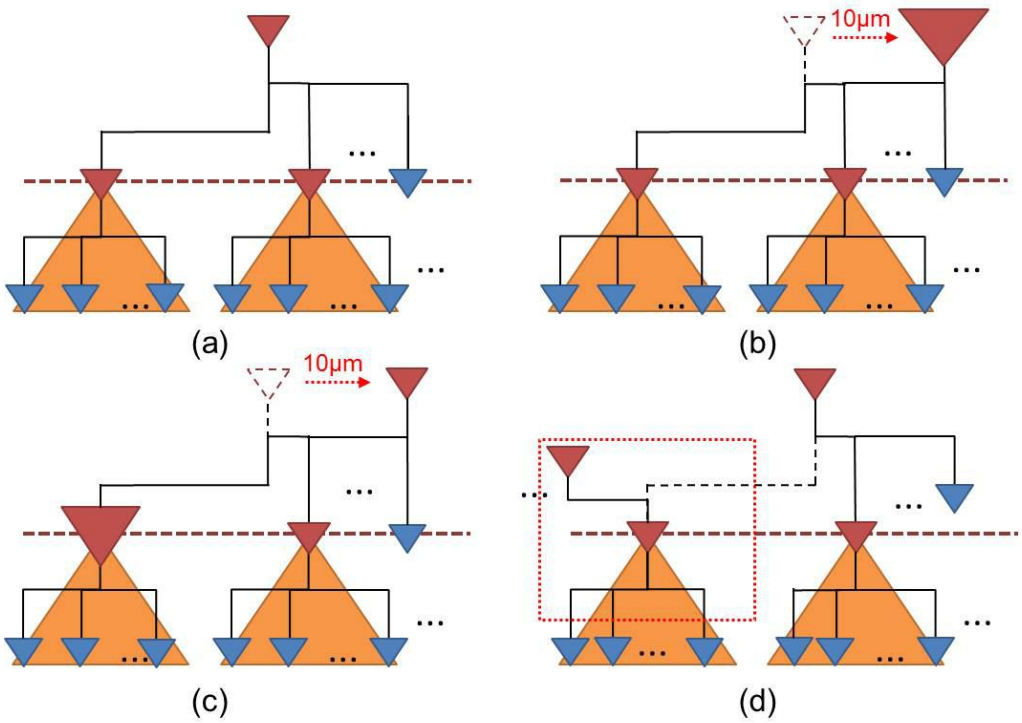


Figure 4.35: Local optimization moves used in our flow. (a) Initial subtree, (b) sizing and/or displacement, (c) displacement and sizing of child node, and (d) tree surgery, i.e., driver reassignment.

Our machine learning models use Artificial Neural Networks (ANN) [85], Support Vector Machines (SVM) with a Radial Basis Function (RBF) kernel [85], and Hybrid Surrogate Modeling (HSM) [122].[68] In addition to the estimated delays based on {FLUTE tree, single-trunk Steiner tree} × {Elmore delay, D2M}, the input parameters to the machine learning-based model also include the number of fanout cells, as well as the area and aspect ratio of the bounding box which contains driving pin and fanout cells. To generate training data, we construct artificial testcases (i.e., clock trees) that resemble real designs with fanout ranging from 1-5 (20-40 for last-stage buffers) and bounding box area and aspect ratio of the driven pins respectively ranging from $1000\mu m^2$ to $8000\mu m^2$ and from 0.5 to 1. We then place fanout cells or sinks randomly within the bounding box. We generate 150 artificial testcases and perform 450 moves on average with 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 4.36(a) shows the predicted vs. actual latencies that we compute from the predicted delta latencies by our model at corner c_3 in Table 4.9. Figure 4.36(b) shows the corresponding histogram of

[68] Further details of the applied machine-learning techniques that we use may be found in [85] and [122].

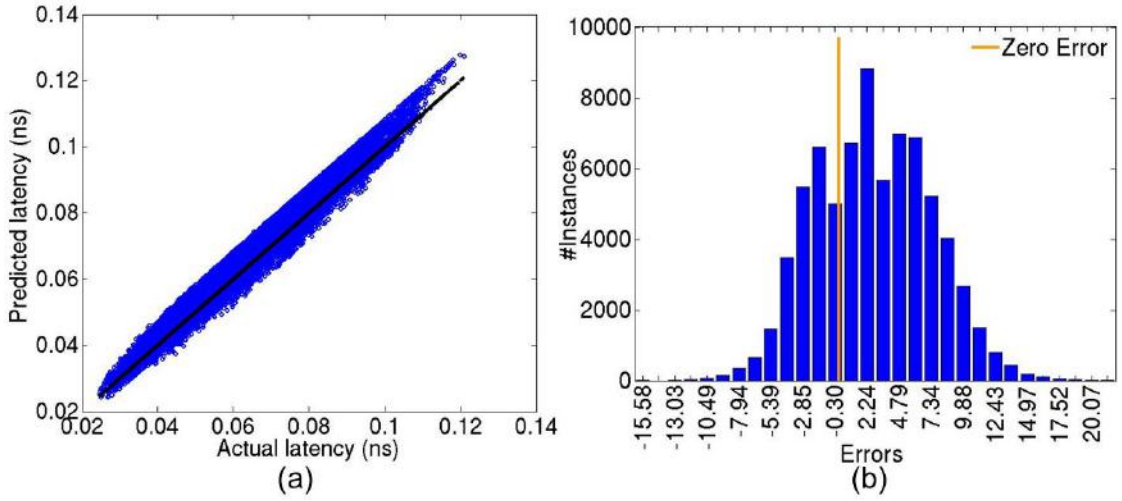


Figure 4.36: Examples of (a) predicted vs. actual latencies and (b) percentage error histograms from our model for c_3 corner in Table 4.9.

percentage errors. Across all the corners, our modeling error is 2.8% on average. The absolute maximum and minimum errors are 21.98% and 16.21% respectively. The modeling for each corner using the artificial testcases is a one-time effort. On a 2.5GHz Intel Xeon server, the time to train a model for each corner is around 5 hours with four threads. Models for each corner can be trained in parallel, e.g., on a server with 24 threads, we can train six models in 5 hours. Our models generalize to different testcases because (i) our training dataset generated from the artificial testcases span ranges of parameters that are typically seen in clock trees in SoC application processors and memory controllers, and (ii) we prevent overfitting by performing cross-validation. Our experimental results indicate that our models are generalizable and accurate when applied to "unseen" testcases during the model training phase. Figure 4.3.1 shows the accuracy comparison between our learning-based model and analytical models. We observe that with fewer attempts, our learning-based model is able to identify the best move for more buffers.

Iterative optimization flow. Based on our model, we perform iterative local optimization flow illustrated in Algorithm 3. 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 4.8. We predict the delta-latency resulting from each move based on our model (Line 2). We then estimate the skew variation reductions based on the predicted latency changes. Our experimental results show that we are able to evaluate the impacts of more than 160K moves at three corners in 17 minutes on a 2.5GHz Intel Xeon server with 15 threads. We sort the candidate moves in

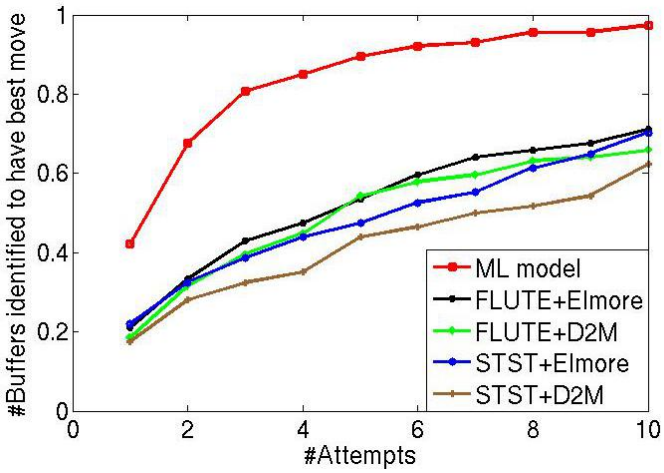


Figure 4.37: Accuracy comparison between our learning-based model and analytical models. An attempt is an ECO. There are 114 buffers, and each buffer has 45 candidate moves. In one attempt, the learning-based model (resp. analytical models) can identify best moves for 40% (resp. up to 20%) of the buffers.

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 3 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**

4.3.2 Experimental Setup and Results

Our experiments are implemented in foundry 28nm LP technology. We construct the original clock tree and perform ECO optimizations using *Synopsys IC Compiler* [340]. We use *Synopsys PrimeTime* [342] and *Synopsys PT-PX* for timing and power analyses, respectively.

Table 4.8: Candidate moves in our optimization.

Type	Candidate moves
I	displace {N, S, E, W, NE, NW, SE, SW} by 10μm $\times$ one-step up/down sizing
II	displace {N, S, E, W, NE, NW, SE, SW} by 10μm $\times$ 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 50μm $\times$ 50μm

We construct the machine learning-based model using *MATLAB* [311]. 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 in 28nm LP technology. The corners used in our experiments are shown in Table 4.9. We use the testcase generation methodology described in [35], and the top-level structures of the testcases T1 and T2 in [35]. 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 650μm $\times$ 650μm 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.[69] Figure 4.38(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 4.10. Corners c_0 and c_1 are setup-critical, and c_3 is hold-critical. Table 4.10 summarizes various post-synthesis metrics of these testcases.

We also study a testcase *CLS2v1* of class memory controller, which is new as compared to [35]. Table 4.10 summarizes the post-synthesis metrics of this testcase, and Figure 4.38(b) shows its floorplan. We use the methodology described in [35] to generate random logic and connect this logic to FFs; this includes datapaths across different clock groups. The memory

[69] We understand from our industry collaborators that best-practices flows for high-speed and memory controller blocks start with 50%–60% utilization before placement [330].

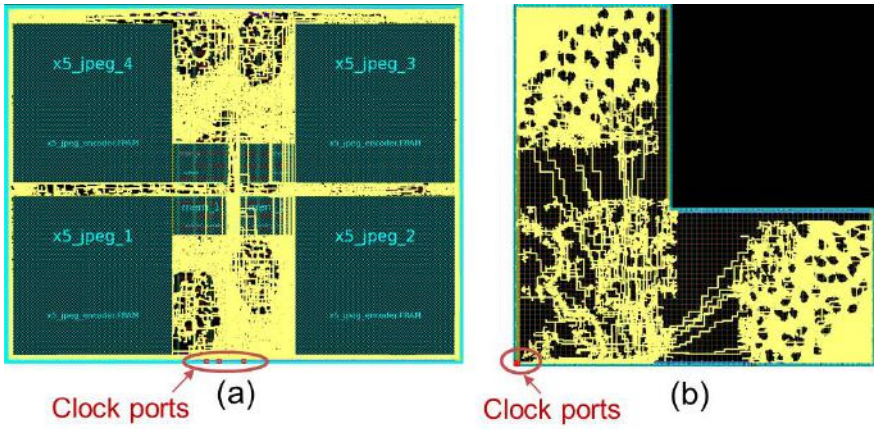


Figure 4.38: Floorplans of (a) *CLS1v1* and (b) *CLS2v1*. In yellow are routed clock nets.

Table 4.9: Description of corners.

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

Table 4.10: Summary of testcases.

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

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 FFs in the interface logic and controller are separated by large distances (e.g., $\sim$1mm). The large distance between sequentially adjacent sinks leads to large clock skew, which the commercial tool tries to balance by inserting buffers. However, these clock buffers lead to

skew variations across corners. We implement the *CLS2v1* testcase at corners c_0, c_1 and c_2 as shown in Table 4.10, 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 [330]. We set the skew target as 0ps in the CTS tools, as our studies (with skew targets ranging from 0ps to 250ps, in steps of 50ps) indicate that a target skew of 0ps steers the tool to deliver the smallest skew at each corner. We perform clock tree optimizations with both multi-mode multi-corner (MMMC) 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 4.11 shows the experimental results,[70] 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,[71] 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. 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). By combining the two optimizations, we reduce the sum of skew variations by up to 22% with negligible area and power overhead. The results also show no degradation of local skews.

Figure 4.39 shows the skew variation reduction during the local iterative optimization. We observe that tree surgeries (type-I moves) are more effective than sizing and displacement moves (type-II and type-III moves), and are applied by our model in the early iterations. For *CLS1v1*, we also show the results with 10 random moves (dots in black), where the gap between random movement and our optimization is 15ns. This supports a conclusion that benefits are due to our delta-latency model. The runtimes per iteration (with 15 threads) are 60 minutes, 80 minutes and 200 minutes for testcases *CLS1v1*, *CLS1v2* and *CLS2v1*, respectively.

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

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

Table 4.11: Experimental results of MMMC clock skew variation minimization.

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

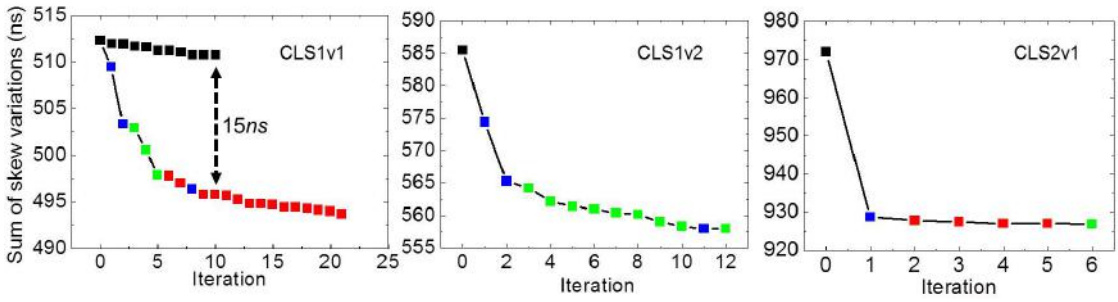


Figure 4.39: Sum of skew variations reduces during the local iterative optimization. In blue are type-I moves, in red are type-II moves, and in green are type-III moves.

Figure 4.40 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.

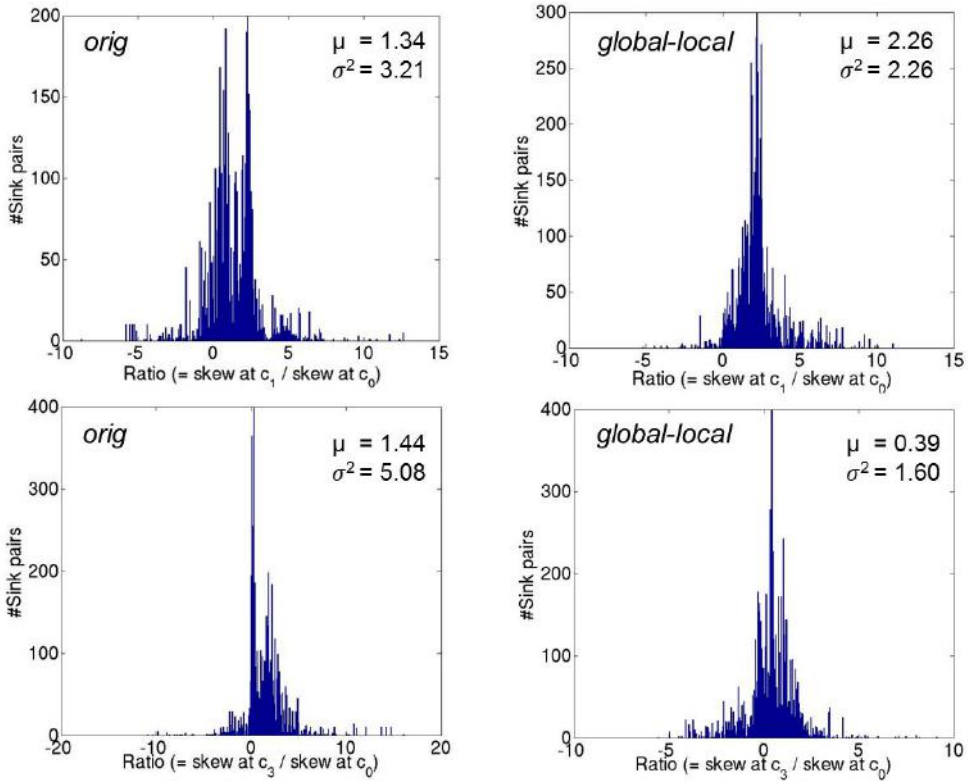


Figure 4.40: Distribution of skew ratios between (c_1, c_0) and (c_3, c_0) of (i) original clock tree and (ii) optimized clock tree for *CLS1v1*.

4.3.3 Conclusions

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 LP 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. Future works may include: (i) development of models to predict a buffer location for minimum skew over a continuous range of possible buffer locations; (ii) 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 (iii) 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.

4.4 Acknowledgments

Chapter 4 is in part a reprint of A. B. Kahng, M. Luo and S. Nath, “SI for Free: Machine Learning of Interconnect Coupling Delay and Transition Effects”, *Proc. ACM International Workshop on System-Level Interconnect Prediction*, 2015; 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; K. Han, A. B. Kahng, J. Lee, J. Li and S. Nath, “A Global-Local Optimization Framework for Simultaneous Multi-Mode Multi-Corner Skew Variation Reduction”, *Proc. IEEE/ACM/EDAC Design Automation Conference*, 2015,; and 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.

I would like to thank my co-authors Kwangsoo Han, Professor Seung-Soo Han, Jong-pil Lee, Jiajia Li, Professor Bill Lin, Mulong Luo, Professor Andrew B. Kahng, and Ashok Vydyanathan.

I also thank Tom Spyrou, Dr. Cho Moon, Roger Embree, and Dr. Puneet Sharma for their early feedback on our project. I thank Gary Smith of Gary Smith EDA for his listing of timing analysis tool providers. I also thank CMP and STMicroelectronics for access to the 28nm FDSOI design kit.

Chapter 5

Optimizations of Design Power, Energy, Project Management, and Cost

This chapter presents three distinct works that directly benefit leading-edge SoC design companies. While the works in this chapter are not based on learning-based modeling, their respective optimizations change the envelope of what is being modeled. The first work describes a new analytic three-dimensional IC (3DIC) placement tool, *APlace3D* (*A3D*) that applies a new "true 3D" wirelength objective function. We describe a flow that uses A3D with commercial EDA tools, and demonstrate that our placement solutions achieve significant wirelength and power reduction relative to two-dimensional IC (2DIC) implementations. This work enables generation of a stronger 3DIC implementation "ground truth" and thus improves the modeling of 3D power benefit and routability prediction. The second work provides two mixed integer-linear programs, along with an associated solver implementation, for optimal multi-project, multi-resource allocation with task precedence and resource co-constraints. This solver enables decision support to management in IC design companies via "what if" analyses of cost and schedule tradeoffs. Accurate modeling would interact with this capability by providing improved constraints and requirements, leading to better optimization of schedule and resource allocations. The third work presents a maximum-value, reliability-constrained overdrive frequencies (MVRCOF) problem that guarantees prescribed lower bounds on acceptable performance and acceptable throughput in multi-core systems, without exceeding prescribed lifetime budget for any core. We develop a solver for the MVRCOF problem and present optimal and heuristic solutions that determine the execution times of each core in each combination of simultaneously active cores, such that cores wear out in a balanced manner over the chip lifetime. Accurate modeling

would affect evaluation of constraints such as temperature, and thus lead to better optimization of the objective function.

5.1 Analytic 3DIC Placement using a New "True 3D" Objective

Three-dimensional integrated circuits (3DICs) are well-understood to offer substantial scaling possibilities for the semiconductor industry [215]. In the past, 3DIC has been used to refer to packaging-driven techniques, e.g., flip-chip, package-on-package, and more recently, through-silicon via (TSV)-based 3D. Recently 3D VLSI (3DV) has received attention as an alternative to packaging-driven 3D integration technologies. 3DV is a foundry-driven, wafer-level 3D integration process that can further be categorized into sequential face-to-back (F2B) and parallel F2B/face-to-face (F2F) integration technologies. The abundance of vertical interconnects (VIs) in 3DV enables designers to rethink existing block and SoC implementations, and enables significant power and performance improvements through shorter interconnects and fewer interconnect buffers. Notably, 3DV calls for new, "true 3D" physical implementation flows that properly comprehend the potential of this implementation technology.

As reviewed in Section 2.3.1, existing 3DIC physical implementation flows have primarily focused on packaging-driven 3DICs; such works typically address core- or block-level 3D implementations where all the IPs are 2D, and are only floorplanned in a 3D space. The recent "shrunk2D" (S2D) flow of Panth et al. [196] performs block-level 3D implementation using 2D EDA tools and is, to our understanding, the strongest available 3D implementation flow.

In this section, we describe a new analytic placer *APlace3D* (*A3D*) that, in conjunction with commercial P&R, achieves superior routed wirelength and power outcomes. The main contributions of this section are as follows.

- We implement a new analytical placer, *A3D*, for 3DICs that is built on APlace2D source code [114]. We propose a novel wirelength objective – a weighted sum of half-perimeter wirelengths (HPWLs) of subnets in each tier and the HPWL of a given net across tiers. We further apply the Gordian-L net model [113] [225] in 3D, and we propose a new net weighting method based on the ratio of a given net's Steiner minimum tree cost to its HPWL. We also study an objective that near-exactly evaluates "true" 3D wirelength. We demonstrate that A3D produces placements that are fully routable and that can be signed off for timing and power using commercial EDA tools. As reviewed in Section 2.3.1,

previous works on analytical 3DIC placers do not optimize a "true 3D" wirelength, and do not guarantee that the placement solutions are routable in a commercial EDA tool.

- Compared to S2D solutions, our solutions reduce routed wirelength by up to 24% and total power by up to 12% in 28nm FDSOI. We study available benefit in 3DICs using a methodology described in [38], and show that our improvements over S2D comprise a significant advancement relative to an upper bound on available 3D benefit. This may imply that, e.g., wirelength benefits of future 3DIC placement works will not substantially improve on those achieved by A3D.

- We demonstrate that our 3D implementation improvements relative to 2D implementations are general across two foundry technologies – 28nm FDSOI and 28nm LP – and two sets of available channel lengths. We achieve up to 31% reduction in routed WL and up to 20.2% reduction in total power at iso-performance on real designs relative to 2D implementations.

5.1.1 APlace3D Implementation and Flow Description

We start with an implementation of APlace2D (obtained from authors of [112] [113] [114]) that performs global placement by posing it as a constrained nonlinear optimization problem. Given modules to be placed, APlace2D divides the placement area into uniformly-sized grids and minimizes HPWL subject to the constraint that total module area in each grid must be balanced. Formally, the problem can be expressed as [113] [114]

$$\begin{aligned} &\text{minimize: } \textit{HPWL(x,y)} \\ &\text{subject to: } D_g(x,y) = D, \text{ for each grid } g \end{aligned} \tag{5.1}$$

where (x,y) denotes the vector of coordinates of the centers of placeable modules, *HPWL(x,y)* denotes the total HPWL of the placement solution, $D_g(x,y)$ denotes the density function to balance the total module area for each grid g, and D is a constant which is the average area of modules over all grids. To apply nonlinear optimization techniques, both HPWL and the density function must be continuously differentiable so as to minimize them. Therefore, these functions must be made "smooth".

Smoothed wirelength function in APlace2D. APlace2D uses a log-sum-exp method to transform the linear HPWL so that it can be continuously differentiated and effectively minimized [184]. For a hyperedge (net) e with pin coordinates $\{(x_1,y_1), (x_2,y_2), ... (x_n,y_n)\}$, the wirelength using log-sum-exp is expressed as [113] [114] [184]:

$$\begin{aligned} WL(e) \quad &= \alpha \cdot \left(\log \left(\sum_{i=1}^{n} e^{x_i/\alpha} \right) + \log \left(\sum_{i=1}^{n} e^{-x_i/\alpha} \right) \right) + \\ &\quad \alpha \cdot \left(\log \left(\sum_{i=1}^{n} e^{y_i/\alpha} \right) + \log \left(\sum_{i=1}^{n} e^{-y_i/\alpha} \right) \right) \end{aligned} \tag{5.2}$$

where α is a smoothing parameter [184] and *WL(e)* is differentiable. Naylor et al. [184] propose α so that *WL(e)* converges to *HPWL(e)* as α converges to 0. α guides the placer to choose nets whose length minimization must be prioritized over other nets. For example, given a two-pin net e' with pin coordinates $\{(x_1,y_1), (x_2,y_2)\}$, the partial derivative of the log-sum-exp wirelength method for x_1 is

$$\frac{\partial WL(e')}{\partial x_1} = \frac{1}{1+e^{(x_1-x_2)/\alpha}} + \frac{1}{1+e^{(x_2-x_1)/\alpha}} \tag{5.3}$$

Nets whose lengths are long relative to α will be preferentially chosen for minimization over nets whose lengths are small relative to α (because the exponentiated term will become 0 and $e^0 = 1$). APlace2D uses this characteristic of α during multi-level clustering. APlace2D adaptively adjusts α based on the grid size. When the value of α is made comparable to the grid size, only long nets are preferentially chosen for minimization over short nets.

APlace2D also implements another smoothening method, first proposed with Gordian-L [225], to minimize a linear wirelength objective using iterated quadratic minimizations. The x-direction wirelength of a two-pin net e with pin coordinates $\{(x_1,y_1), (x_2,y_2)\}$ is expressed as [114] [225]

$$WL(e) = \sum_{x_1,x_2} (x_1-x_2)^2/\gamma_{x_1,x_2} \tag{5.4}$$

where $\gamma_{x_1,x_2} = \max\{r_0, |x_1-x_2|\}$, and is updated as x_1 and x_2 change; r_0 sets a minimum value for γ_{x_i,x_j} so as to prevent overflow. Kahng et al. [114] propose to set r_0 to $\sim$10% of the width of grid g to obtain the best wirelength reduction, based on empirical observations.

Smoothed density function in APlace2D. The density function D_g in Equation (5.1) is smoothed to make it continuously differentiable via [113] [114]

$$D_g(x,y) = \sum_v P_x(g,v) \cdot P_y(g,v) \tag{5.5}$$

where v denotes a module (e.g., a standard cell), and $P_x(g,v)$ and $P_y(g,v)$ are functions to represent the overlap between grid g and module v along x- and y-directions, respectively. Naylor et al. [184] smoothe these functions using corresponding "bell-shaped" functions $p_x(g,v)$ and $p_y(g,v)$. Kahng et al. [113] improve the handling of large block-sized modules (instead of modules being limited to standard cells only), via the density function

$$D_g(x,y) = \sum_v C_v \cdot p_x(g,v) \cdot p_y(g,v) \tag{5.6}$$

where C_v is a normalization factor chosen such that $\sum_g C_v \cdot p_x(g,v) \cdot p_y(g,v)$ is equal to the total module area.

Congestion-Directed Placement in APlace2D. APlace2D uses the congestion estimation method proposed by Kahng and Xu in [130]. The method accounts for the impact of the number of bends in a routing path, and the impact of neighboring nets on a path's occurence probability. When a grid is estimated to be congested, its density function D_g is decreased. When a grid is estimated to be uncongested, its D_g is increased such that the sum of D_g over all grids remains constant, and is equal to the total area of standard cells. Formally, D_g is modified as follows [113]:

$$D_g \propto 1 + \gamma \cdot \left(1 - 2\frac{Congestion(g)}{max_g Congestion(g)}\right) \tag{5.7}$$

where γ is a factor used to adjust congestion ($\gamma = 0.6$ is used in our reported experiments), and *Congestion(g)* is the congestion in grid g estimated using the method in [130].

Penalty Function in APlace2D. Using the smoothed wirelength and density functions, APlace2D solves the optimization problem in Equation (5.1) by solving a sequence of unconstrained minimization problems with a quadratic penalty:

$$\text{minimize: } WL(x,y) + \frac{1}{2\mu}\sum_{g}(D_g(x,y) - D)^2 \tag{5.8}$$

for a sequence of values of μ. The authors of APlace2D [112] [113] [114] decide μ_0, the initial value of μ, using values of wirelength and density (i.e., Equation (5.6)) gradients as follows:

$$\mu_0 = \frac{1}{2} \cdot \frac{\sum_{x_i,y_j}\sum_{g}|D_g - D| \cdot \left(\left|\frac{\partial D_g}{\partial x_i}\right| + \left|\frac{\partial D_g}{\partial y_j}\right|\right)}{\sum_{x_i,y_j}\left(\left|\frac{\partial WL}{\partial x_i}\right| + \left|\frac{\partial WL}{\partial y_j}\right|\right)}. \tag{5.9}$$

We extend APlace2D to A3D using a modified density function and three wirelength calculation methods.

Density Function in A3D. We change the original APlace2D density function to comprehend overlap in the z-direction. Thus, we add changes to compute density gradient in the z-direction. With 3D support, the density function from Equation (5.6) becomes

$$D_g(x,y,z) = \sum_{v} C_v \cdot p_x(g,v) \cdot p_y(g,v) \cdot p_z(g,v) \tag{5.10}$$

where $p_z(g,v)$ is the function to represent overlap of module v along the z-direction, and is calculated in the same manner as functions $p_x(g,v)$ and $p_y(g,v)$ [113]. Figure 5.1(a) illustrates the rectangle-shaped 0/1 density function $p_z(g,v)$, where w_g denotes the grid width, z_v denotes the location of module v along the z-direction, and z_g denotes the grid center. When the z-direction between grid g and module v is $|z_v - z_g| < w_g/2$, $p_y(g,v)$ is 1 and otherwise is 0. Figure 5.1(b) illustrates the smoothened density function based on techniques described in [113] [184].

Wirelength in A3D. We have implemented three new wirelength calculation methods in A3D. In Section 5.1.2 we demonstrate results with each of the three different methods.

(i) Gordian-L in 3D (A3D-GL3D). The first method is a straightforward extension of log-sum-exp and Gordian-L methods to 3D by adding a cost in the z-direction. We also compute

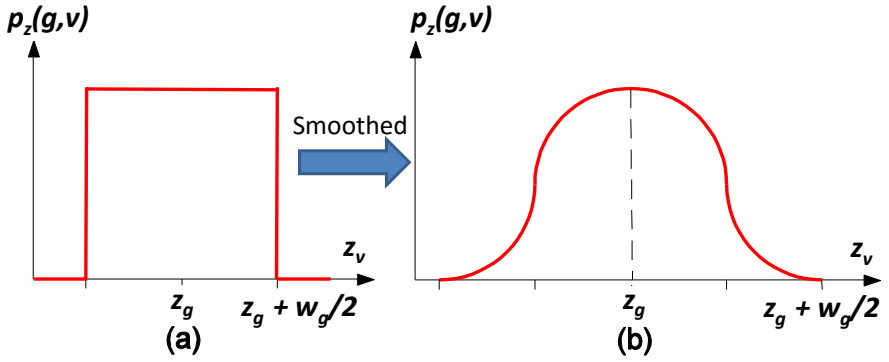


Figure 5.1: Illustration of density function $p_z(g,v)$: (a) rectangle-shaped 0/1 density function and (b) bell-shaped smoothened density function [113] [184].

the wirelength gradient in the z-direction. With 3D support, the log-sum-exp wirelength from Equation (5.2) for a net e with pin coordinates $\{(x_1,y_1,z_1), (x_2,y_2,z_2), \ldots (x_n,y_n,z_n)\}$ is

$$\begin{aligned} WL(e) \quad &= \alpha \cdot \left(\log\left(\sum_{i=1}^{n} e^{x_i/\alpha} \right) + \log\left(\sum_{i=1}^{n} e^{-x_i/\alpha} \right) \right) + \\ &\alpha \cdot \left(\log\left(\sum_{i=1}^{n} e^{y_i/\alpha} \right) + \log\left(\sum_{i=1}^{n} e^{-y_i/\alpha} \right) \right) + \\ &\alpha \cdot \left(\log\left(\sum_{i=1}^{n} e^{z_i/\alpha} \right) + \log\left(\sum_{i=1}^{n} e^{-z_i/\alpha} \right) \right) \end{aligned} \tag{5.11}$$

The Gordian-L wirelength [225] in the z-direction of a two-pin net e with pin coordinates $\{(x_1,y_1,z_1), (x_2,y_2,z_2)\}$ is expressed as:[72]

$$WL(e) = \sum_{z_1,z_2} (z_1 - z_2)^2 / \gamma_{z_1,z_2} \tag{5.12}$$

where $\gamma_{z_1,z_2} = \max\{r_0, |z_1 - z_2|\}$, and is updated as z_1 and z_2 change; r_0 is the minimum value of γ_{z_i,z_j} used to prevent overflow.

(ii) Weighted-wirelength (A3D-WWL). The second method is a new net weighting method that uses the ratio of Steiner wirelength and the net's HPWL. For each net, we calculate the Steiner wirelength using FLUTE [51] and the HPWL using the union of pin locations over all tiers. When the Steiner wirelength of a net is larger than the HPWL, the weight is large and guides the optimization to minimize wirelength for this net.

[72] APlace2D uses the log-sum-exp net model only for unclustered hypergraphs, and the Gordian-L net model for clustered hypergraphs. In A3D-GL3D we retain this same code flow used in APlace2D.

(iii) True 3D wirelength (A3D-T3D). The third method is a new wirelength metric that we refer to as *true 3D* (T3D). Given pin placement on two tiers, the T3D wirelength metric is calculated as follows.

$$T3D = W_1 \cdot HPWL_0 + W_2 \cdot HPWL_1 + W_3 \cdot HPWL_{ov} \tag{5.13}$$

where $W_{1,2,3}$ are user-specified weights and $HPWL_{ov}$ denotes the HPWL across two tiers, i.e., when the two tiers are vertically overlapped. The assumption here is that no signal net crosses between tiers more than once, i.e., a net uses at most one vertical interconnect (VI). Minimizing the *T3D* objective function enables the optimization to minimize HPWL of subnets on each tier as well as the HPWL of nets across tiers, for each net. Figure 5.2 illustrates objective *T3D* for a 4-pin net with pins *A* and *B* placed on Tier 0, and pins *C* and *D* placed on Tier 1. We calculate $HPWL_0$ and $HPWL_1$ from the respective bounding boxes in green color in Tiers 0 and 1. The figure shows the projections of pins *C* and *D* onto Tier 0 and the resulting bounding box of all the four pins in brown color. We calculate $HPWL_{ov}$ from this (brown color) bounding box. *T3D* wirelength is equal to the 3D Steiner wirelength for 2-pin nets. However, for multi-pin nets, *T3D* overestimates the 3D Steiner wirelength because HPWL_{ov} may double-count the HPWL on each tier. Assuming one VI per signal net that crosses tiers, Figures 5.3(a) and (b) respectively show examples where *T3D* overestimates wirelength for a 4-pin net and a 3-pin net. In Figure 5.3(a), the wirelength is overestimated by either HPWL_0 or HPWL_1. In Figure 5.3(b), the wirelength is overestimated by HPWL_0, i.e., *T3D* computes the wirelength as 40 (HPWL_0 is 20, HWPL_{ov} is 20, so *T3D* is 40), instead of 20. Figure 5.3(b) shows that in the worst case, *T3D* wirelength is $2\times$ the 3D wirelength. To implement a "truer true 3D" Steiner wirelength, HPWL_{ov} in *T3D* can be modified to be the minimum Manhattan distance between any pin on Tier 0 and any pin on Tier 1, when the subnet bounding boxes on each tier do not overlap, and zero otherwise (i.e., when the subnet bounding boxes on each tier overlap). We use *T3D'* to denote *T3D* with HPWL_{ov} modified in this way. Our experimental results in Section 5.1.2 indicate that the differences in routed wirelengths between *T3D* and *T3D'* are not significant ($\sim$0.02% in wirelength and 0.1% in power).

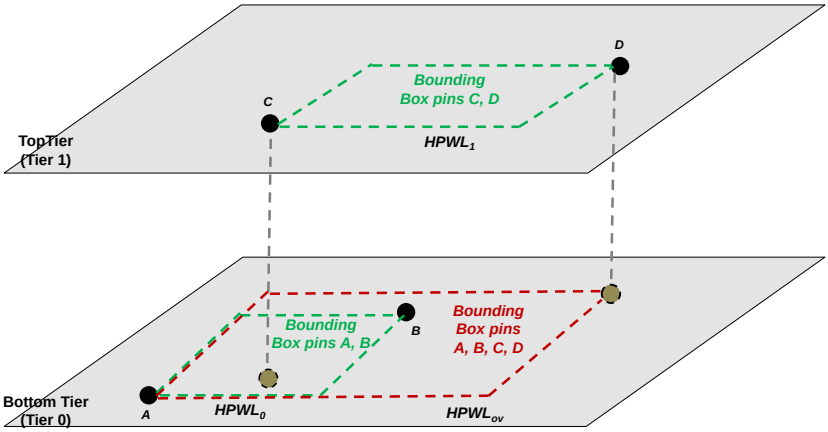


Figure 5.2: Illustration of wirelength objective *T3D* using a 4-pin net, whose pins A and B are placed on Tier 0 and pins C and D are placed on Tier 1. The bounding box in brown color is the overlapped bounding box of the union of pins of net over both tiers. We calculate $HPWL_{ov}$ from this bounding box.

Using 3D-aware WL[73] and density function (i.e., Equation (5.10)), the sequence of unconstrained minimization problems with a quadratic penalty changes as follows.

$$\text{minimize: } WL(x,y,z) + \frac{1}{2\mu}\sum_{g}(D_g(x,y,z) - D)^2 \tag{5.14}$$

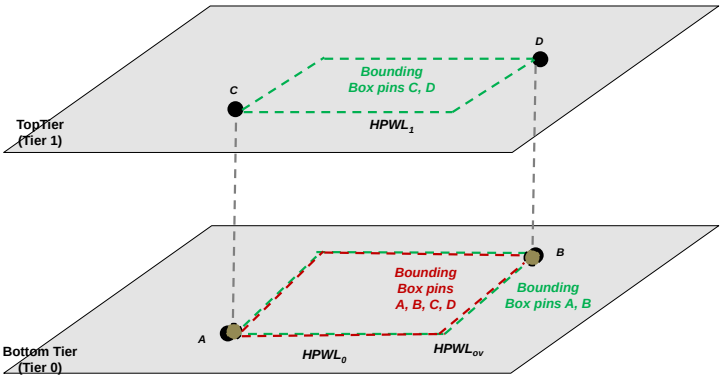
where the initial value of μ, μ_0 is calculated as follows.

$$\mu_0 = \frac{1}{2} \cdot \frac{\sum_{x_i,y_j,z_k}\sum_{g}|D_g - D| \cdot \left(\left|\frac{\partial D_g}{\partial x_i}\right| + \left|\frac{\partial D_g}{\partial y_j}\right| + \left|\frac{\partial D_g}{\partial z_k}\right|\right)}{\sum_{x_i,y_j}\left(\left|\frac{\partial WL}{\partial x_i}\right| + \left|\frac{\partial WL}{\partial y_j}\right| + \left|\frac{\partial WL}{\partial z_k}\right|\right)} \tag{5.15}$$

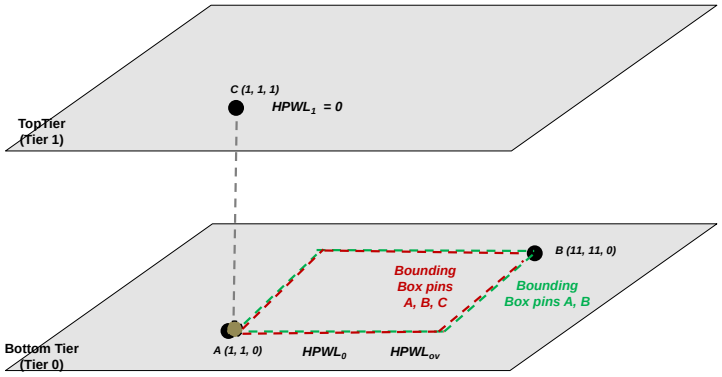
A3D flow description. Figure 5.4 shows our flow using A3D. Given a post-synthesis netlist, we perform placement of PI/POs and flip-flops. To perform clock tree synthesis (CTS), we evaluate two choices: (i) split at source (i.e., at the clock roots), and (ii) split at sinks. Our experimental results shown in Table 5.1, with two open-source designs, indicate that splitting at sinks achieves ∼18%–20% less clock power, #buffers and skew than splitting at the source. These results are consistent with those presented in [197].[74] We therefore perform CTS using the *splitting at sinks*

[73]Note that we implement each of A3D-GL3D, A3D-WWL and A3D-T3D separately. We have also applied A3D-WWL using HPWL of each tier and combined with A3D-T3D. The results are similar to (i.e., within 0.12% for all testcases studied) the A3D-T3D method.

[74]Note that for each design reported in Table 5.1, we have performed CTS using both split at source and split at sink methodologies using identical values of maximum latency, target skew and maximum #levels.



(a)



(b)

Figure 5.3: Examples where *T3D* overestimates wirelength in 3D. Overestimated by value of: (a) $HPWL_0$ or $HPWL_1$, and (b) $HPWL_0$. The assumption is that only one VI is used per signal net that crosses tiers.

methodology with low-Vt BFX-type cells from the technology library. In this methodology, all clock buffers are placed on Tier 0, but flip-flops can be placed either on Tier 0 or Tier 1. We invoke A3D and obtain a global placement of all standard cells across each tier. (Note that PI/POs from the original design and clock buffers are fixed on Tier 0.) We then use Cadence *Innovus v15.2* [286] to legalize and refine the A3D placement. After that, we insert VIs as dummy cells, and invoke A3D again to obtain a placement of VIs (standard cell placements are fixed in the input to A3D). After obtaining locations of VIs, we change these VI dummy cells to PI/POs on each tier, and use Cadence *Innovus* to perform tier-by-tier legalization, routing and optimization. To perform signoff timing and power analysis, we use Synopsys *PrimeTime (PT)-SI* and *PTPX* [342].

Table 5.1: Comparison of CTS methodologies for 3D in 28nm FDSOI.

Split at	*aes_cipher_top*			*jpeg_encoder*		
	#Clk Bufs	Skew (ps)	Clk Power (mW)	#Clk Bufs	Skew (ps)	Clk Power (mW)
source	15	22	1.1	134	52	9.81
sink	12	17	0.9	111	42	8.25

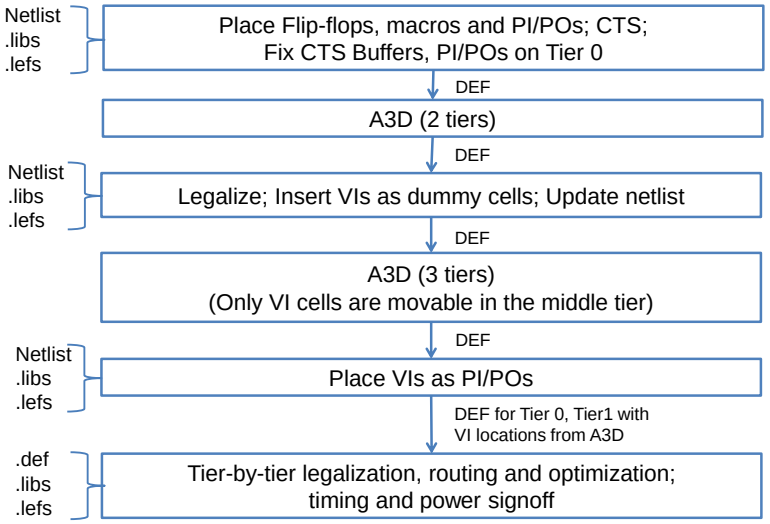


Figure 5.4: 3D P&R flow using two tiers with A3D.

5.1.2 Experimental Setup and Results

We now describe our design of experiments, and present results using our A3D placer.

Design of Experiments

We have implemented all three wirelength (WL) methods (A3D-GL3D, A3D-WWL and A3D-T3D) in A3D in C++, compiled with *g++ v4.4.7*, and verified on Intel Xeon E5-2690 2.6GHz server with *CentOS v6.8*. In our experiments we use open-source designs *aes_cipher_top*, *jpeg_encoder*, *ldpc* decoder and *dec_viterbi* from OpenCores [318], *netcard* and *leon3mp* from the ISPD-12 benchmark suite [187], and OST2 *spc* core [317]. Table 5.2 shows details of the post-synthesis netlists in 28nm FDSOI. We perform synthesis using Synopsys *Design Compiler vI-2013.12-SP3* [337], P&R using Cadence *Innovus v15.2* [286] and signoff timing and power analyses using Synopsys *PrimeTime-SI* and *PrimeTime-PX vJ-2014.12* [342], respectively. We use two 12-track foundry technology libraries, 28nm FDSOI and 28nm LP; in 28nm FDSOI we

conduct experiments using four channel lengths (24nm, 27.6nm, 33nm and 38.4nm) as well as one channel length (24nm). In all of our experiments we use six metal layers for routing (M1–M6 pitches are 0.136μm, 0.1μm, 0.1μm, 0.1μm, 0.1μm, and 0.1μm) and perform 3D P&R using two tiers.

We perform the following three experiments.

- **Experiment 1.** The goal of this experiment is to quantify WL and power benefits of using the three WL calculation methods in A3D and the A3D flow over a *best 2D* implementation [38] of our designs at iso-performance and iso-area. We compare QoR using 28nm FDSOI and 28nm LP technologies. For 28nm FDSOI, we use two different sets of channel lengths for our comparisons.

- **Experiment 2.** The goal of this experiment is to calibrate our results with respect to an "upper bound" on power benefit in 3D. We apply an "infinite dimension" analysis methodology described in [38].

- **Experiment 3.** The goal of this experiment is to compare the A3D flow QoR using the three variants of WL calculation methods with the S2D flow QoR at iso-performance and iso-area. We study both face-to-face (F2F) as well as face-to-back (F2B) [193] configurations.

Table 5.2: Netlist details at the post-synthesis stage in 28nm FDSOI.

Design	#Instances	#Flip-flops	#Memories	Cell Area (μm^2)
aes_cipher_top (*aes*)	7933	530	0	8386
jpeg_encoder (*jpeg*)	18362	4307	0	32310
ldpc	43515	2048	0	48713
dec_viterbi (*viterbi*)	54048	26081	0	116736
netcard	344605	87197	0	398228
leon3mp	435019	108817	0	659413
spc	234881	44623	135	1527247

Results of Experiment 1

In this experiment we quantify WL and power benefits of using the three WL calculation methods in A3D and the A3D flow over a best 2D implementation [38] of our designs at iso-performance and iso-area. For the best 2D flow, we determine a P&R clock period for each

design such that the worst-negative slack (WNS) at signoff is between -50ps and -70ps. Using this clock period, we sweep the synthesis clock period by factors of {0.83, 0.91, 1.0, 1.11} times the P&R clock period and choose the netlist that delivers minimum power and zero total negative slack (TNS) at post-synthesis. We then determine the maximum utilization with six metal layers for each design by sweeping the target utilization from 75% to 90% in steps of 1%. We denoise each run by varying the utilization by {-0.05, 0, 0.05}%, and choose the maximum utilization across the three denoised runs that has the number of design rule violations (DRCs) ≤ 50 in at least one of the denoised implementations. The tuples <design, clock period, maximum utilization> are <*aes*, 0.65ns, 86%>, <*jpeg*, 0.85ns, 85%>, <*ldpc*, 0.85ns, 75%>, <*viterbi*, 1.15ns, 80%>, <*netcard*, 1.4ns, 73%>, <*leon3mp*, 3.1ns, 73%>, <*spc*, 4.0ns, 72%>.[75] Our setup and hold corners are {ss, 0.9V, 25C} and {ff, 0.9V, 25C}, respectively.

Tables 5.3–5.9 compare 2D (using results from the best 2D flow) QoR versus A3D-GL3D, A3D-WWL and A3D-T3D QoR for multiple metrics for seven designs, across 28nm FDSOI and LP foundry technologies. For 28nm FDSOI, we also present results with four channel lengths and one channel length. All our A3D implementations are routable with #DRCs ≤ 50, and we can perform signoff timing and power analyses. For WL, we achieve 31% reduction using A3D-WWL and A3D-T3D for both *aes* (see Table 5.3) and *ldpc* (see Table 5.5) relative to their corresponding 2D implementations. Our achieved reduction in WL is comparable to those reported in [1] for *ldpc*, albeit at a different technology. For power, we achieve 20.28% reduction for *ldpc* (see Table 5.5) and 16.72% reduction for *aes* (see Table 5.3) relative to their 2D implementations. Across all designs, we achieve at least 18% reduction in WL and 2.5% reduction in power. Table 5.11 compares QoR of A3D-T3D and A3D-T3D′ flows for *aes* and *ldpc* in 28nm FDSOI technology with four channel lengths. The differences in routed wirelength and power between A3D-T3D and A3D-T3D′ flows are $\leq 0.1\%$, i.e., insignificant. Hence, in what follows we use the A3D-T3D flow for comparisons with other flows.

We have conducted additional experiments with A3D-GL3D to observe how #VIs change as we vary cost in the z-direction. Figure 5.5 shows for the *jpeg* design that when z-cost is zero, almost all nets in the design are cut across tiers. When the z-cost is 1, the #VIs is the value (i.e., 7766) reported in Table 5.4 for A3D-GL3D and *jpeg*. When we set the z-cost to a large value (100), the #VIs reduces to 5582. Reduction of #VIs from from 100 is not significant because to maintain area balance across tiers, a subset of nets must be cut.

[75] We conducted experiments for *spc* only in 28nm FDSOI because we did not have the required memory macros in 28nm LP.

Table 5.3: Comparison of *aes_cipher_top* metrics from 2D vs. A3D flows using F2F integration in 28nm FDSOI and LP technologies.

	Four Channel Lengths / One Channel Length / LP			
	2D	A3D-GL3D	A3D-WWL	A3D-T3D
#Cells	8552 / 8558 / 11545	8393 / 8352 / 10916	8362 / 8620 / 11039	8362 / 8500 / 11036
#Buffers	1077 / 1097 / 1465	856 / 917 / 1137	859 / 905 / 1144	859 / 976 / 1146
#Clk Buffers	14 / 14 / 16	13 / 13 / 15	13 / 13 / 15	13 / 13 / 15
Cell Area (μm^2)	10954 / 10899 / 6572	10323 / 10290 / 5937	10053 / 10214 / 5911	10048 / 10279 / 5779
#VIs	–	4789 / 4782 / 4312	4258 / 4260 / 4240	3266 / 3269 / 3310
WL (μm)	116.7K / 116.3K / 98.20K	89.36K / 89.19K / 87.41K	80.49K / 80.48K / 80.13K	80.67K / 80.29K / 80.13K
%ΔWL wrt 2D	–	**-23.4 / -23.3 / -11.0**	**-31.0 / -30.8 / -18.4**	**-30.9 / -31.0 / -18.4**
WNS (ps)	-34 / -9 / -31	0 / 0 / -12	0 / 0 / -4	0 / 0 / -4
TNS (ns)	-0.8 / -0.07 / -12.3	0 / 0 / -8.7	0 / 0 / -3.2	0 / 0 / -3.2
Power (mW)	12.26 / 12.39 / 4.66	10.83 / 10.97 / 4.49	10.21 / 10.72 / 4.45	10.23 / 10.66 / 4.45
%ΔPower wrt 2D	–	**-11.7 / -11.5 / -3.7**	**-16.7 / -13.5 / -4.5**	**-16.6 / -14.0 / -4.5**

Table 5.4: Comparison of *jpeg_encoder* metrics from 2D vs. A3D flows using F2F integration in 28nm FDSOI and LP technologies.

	Four Channel Lengths / One Channel Length / LP			
	2D	A3D-GL3D	A3D-WWL	A3D-T3D
#Cells	19744 / 19712 / 25770	19596 / 19540 / 25163	20976 / 20442 / 26961	20957 / 20743 / 27130
#Buffers	2028 / 2045 / 2636	1711 / 1740 / 2139	2218 / 1897 / 2751	2184 / 2184 / 2790
#Clk Buffers	111 / 111 / 117	99 / 99 / 103	99 / 99 / 103	99 / 99 / 103
Cell Area (μm^2)	36099 / 36032 / 21298	33436 / 34561 / 20061	34252 / 34922 / 20060	34266 / 34890 / 20131
#VIs	–	7766 / 7724 / 7623	7654 / 7648 / 7582	6397 / 6388 / 6480
WL (μm)	259.1K / 259.1K / 248.9K	202.7K / 202.8K / 205.1K	195.8K / 195.8K / 200.2K	194.3K / 194.3K / 199.5K
%ΔWL wrt 2D	–	**-21.2 / -21.2 / -17.6**	**-24.4 / -24.4 / -19.6**	**-25.0 / -25.0 / -19.8**
WNS (ps)	-209 / -138 / -77	-12 / -9 / 0	0 / -2 / 0	-1 / -4 / 0
TNS (ns)	-76.3 / -112.2 / -124.9	-1.2 / -1.0 / 0	0 / -1.0 / 0	-0.04 / -0.9 / 0
Power (mW)	45.65 / 46.25 / 20.67	42.35 / 42.89 / 19.55	42.24 / 42.67 / 19.45	42.23 / 42.65 / 19.47
%ΔPower wrt 2D	–	**-7.2 / -7.3 / -5.4**	**-7.5 / -7.7 / -5.9**	**-7.5 / -7.8 / -5.8**

Results of Experiment 2

In this experiment we calibrate our results with respect to an "upper bound" on power benefit in 3D. We apply an "infinite dimension" analysis methodology described in [38]. This analysis uses synthesis with zero wireload model as a surrogate for implementation in "infinite" dimensions. According to [38], the power benefit of "infinite" dimensions is in some sense an upper bound on power benefit of 3D. Figure 5.6 compares power using the infinite-dimension, best 2D and the A3D flows using the three variants of WL calculation methods. We observe that A3D-WWL and A3D-T3D are able to capture a significant portion of the available benefit

Table 5.5: Comparison of *ldpc* metrics from 2D vs. A3D flows using F2F integration in 28nm FDSOI and LP technologies.

	Four Channel Lengths / One Channel Length / LP			
	2D	A3D-GL3D	A3D-WWL	A3D-T3D
#Cells	47610 / 47622 / 62845	43094 / 43109 / 55573	42982 / 42980 / 55570	42990 / 43002 / 55544
#Buffers	7989 / 7916 / 10226	4670 / 4732 / 5944	4674 / 4701 / 5943	4580 / 4587 / 5974
#Clk Buffers	61 / 64 / 71	55 / 56 / 58	55 / 56 / 58	55 / 56 / 58
Cell Area (μm^2)	62141 / 62156 / 42256	49172 / 49169 / 35976	49312 / 49333 / 35938	49300 / 49312 / 36051
#VIs	–	12866 / 12872 / 12846	14967 / 14984 / 15014	11866 / 11835 / 11826
WL (μm)	1473K / 1473K / 1208K	1021K / 1021K / 862.0K	1014K / 1014K / 859.4K	1014K / 1014K / 859.4K
%ΔWL wrt 2D	–	**-30.7 / -30.7 / -28.6**	**-31.2 / -31.2 / -28.8**	**-31.2 / -31.2 / -28.8**
WNS (ps)	-28 / -7 / -35	0 / 0 / -11	0 / 0 / -9	0 / -2 / -3
TNS (ns)	-7.9 / -0.9 / -20.2	0 / 0 / -56.2	0 / 0 / -20.1	0 / -0.04 / -0.2
Power (mW)	168.6 / 171.2 / 55.2	134.5 / 136.9 / 45.8	134.4 / 136.7 / 45.7	134.4 / 136.7 / 45.7
%ΔPower wrt 2D	–	**-20.2 / -20.1 / -17.1**	**-20.3 / -20.2 / -17.2**	**-20.3 / -20.2 / -17.2**

Table 5.6: Comparison of *dec_viterbi* metrics from 2D vs. A3D flows using F2F integration in 28nm FDSOI and LP technologies.

	Four Channel Lengths / One Channel Length / LP			
	2D	A3D-GL3D	A3D-WWL	A3D-T3D
#Cells	58709 / 58712 / 78083	58211 / 58214 / 75068	58183 / 58296 / 76219	58184 / 58182 / 75079
#Buffers	8029 / 8033 / 10759	6512 / 6513 / 8393	7415 / 7481 / 9416	7406 / 7384 / 9493
#Clk Buffers	617 / 618 / 688	481 / 483 / 556	481 / 483 / 556	481 / 483 / 556
Cell Area (μm^2)	122833 / 122841 / 73700	121078 / 121076 / 72882	120375 / 121151 / 72707	120268 / 121474 / 72797
#VIs	–	33407 / 33406 / 32856	33412 / 33421 / 32906	28097 / 28097 / 28004
WL (μm)	1276K / 1276K / 1148K	1047K / 1047K / 980K	1046K / 1046K / 975K	1046K / 1046K / 974K
%ΔWL wrt 2D	–	**-18.0 / -18.0 / -14.7**	**-18.0 / -18.0 / -15.1**	**-18.0 / -18.0 / -15.2**
WNS (ps)	-20 / 0 / -67	0 / 0 / -13	-2 / -1 / -15	0 / 0 / -12
TNS (ns)	-0.7 / 0 / -123.5	0 / 0 / -56.2	-0.1 / -0.01 / -34.6	0 / 0 / -45.2
Power (mW)	123.3 / 125.1 / 49.2	114.1 / 115.8 / 45.8	114.1 / 115.8 / 45.7	114.1 / 115.8 / 45.7
%ΔPower wrt 2D	–	**-7.5 / -7.4 / -6.7**	**-7.5 / -7.4 / -6.8**	**-7.5 / -7.5 / -6.8**

from 3D. For *aes* and *ldpc*, out of the available 20.3% and 8.9% benefit, respectively, A3D-T3D captures 20% and 8.1% benefit, respectively. For designs that contain $\geq$ 15% of the cells as flip-flops or memory macros such as *viterbi*, *netcard*, *leon3mp* and *spc*, A3D captures 8.1%, 5.6%, 10.6% and 2.6% of the available 16.1%, 16.7%, 19.7% and 6.0% benefit, respectively. Even though absolute power benefit from A3D may look small in Tables 5.3–5.9, it is quite significant compared to the "upper bound" on available benefit from 3D. For example, we believe it is unlikely that future 3DIC P&R flows achieve more than 10% power reduction from our present results.

Table 5.7: Comparison of *netcard* metrics from 2D vs. A3D flows using F2F integration in 28nm FDSOI and LP technologies.

	Four Channel Lengths / One Channel Length / LP			
	2D	A3D-GL3D	A3D-WWL	A3D-T3D
#Cells	375138 / 375142 / 487679	366803 / 366776 / 478620	367506 / 367513 / 479608	367625 / 367649 / 480018
#Buffers	39698 / 39723 / 49226	32575 / 32532 / 42680	33421 / 33357 / 42948	33466 / 33476 / 42770
#Clk Buffers	2227 / 2226 / 2242	1982 / 1982 / 1988	1982 / 1982 / 1988	1982 / 1982 / 1988
Cell Area (μm^2)	438288 / 438292 / 276121	423352 / 423370 / 257055	423350 / 423377 / 256408	422908 / 422875 / 256325
#VIs	–	200134 / 199978 / 200046	200224 / 200230 / 198034	197654 / 197680 / 197558
WL (μm)	9473K / 9473K / 8052K	7202K / 7202K / 6263K	7190K / 7190K / 6261K	7144K / 7144K / 6259K
%ΔWL wrt 2D	–	**-24.0 / -24.0 / -22.2**	**-24.1 / -24.1 / -22.2**	**-24.6 / -24.6 / -22.3**
WNS (ps)	-39 / -12 / -79	-77 / -15 / -18	-64 / -12 / -8	-45 / -13 / 0
TNS (ns)	-6.1 / -1.2 / -321.5	-356.7 / -102.3 / -73.7	-231.4 / -56.1 / -44.6	-209.2 / -20.3 / 0
Power (mW)	403.8 / 413.4 / 121.9	382.5 / 393.3 / 116.9	382.5 / 393.1 / 116.9	382.3 / 393.2 / 116.9
%ΔPower wrt 2D	–	**-5.3 / -4.9 / -4.1**	**-5.3 / -4.9 / -4.1**	**-5.3 / -4.9 / -4.1**

Table 5.8: Comparison of *leon3mp* metrics from 2D vs. A3D flows using F2F integration in 28nm FDSOI and LP technologies.

	Four Channel Lengths / One Channel Length / LP			
	2D	A3D-GL3D	A3D-WWL	A3D-T3D
#Cells	449462 / 449453 / 575311	431509 / 431514 / 556693	432607 / 432641 / 562388	434123 / 434124 / 562162
#Buffers	47208 / 47222 / 62786	43500 / 43516 / 55061	43838 / 43843 / 54854	44700 / 44690 / 55015
#Clk Buffers	2794 / 2796 / 2812	1966 / 1967 / 1972	1966 / 1967 / 1972	1966 / 1967 / 1972
Cell Area (μm^2)	643324 / 643308 / 392428	607332 / 607343 / 367384	605547 / 605552 / 367323	604689 / 605545 / 366990
#VIs	–	130684 / 130692 / 129886	130792 / 130712 / 130016	128824 / 128816 / 128898
WL (μm)	7641K / 7641K / 6266K	5958K / 5958K / 5009K	5931K / 5931K / 5007K	5930K / 5930K / 5007K
%ΔWL wrt 2D	–	**-22.0 / -22.0 / -20.1**	**-22.4 / -22.4 / -20.1**	**-22.4 / -22.4 / -20.1**
WNS (ps)	-27 / 0 / -45	-57 / 0 / -20	0 / 0 / -20	0 / 0 / -16
TNS (ns)	-1.6 / 0 / -324.7	-133.0 / 0 / -180.6	0 / 0 / -111.2	0 / 0 / -92.3
Power (mW)	222.2 / 230.4 / 73.24	200.8 / 208.9 / 69.83	200.8 / 208.9 / 69.82	200.8 / 280.9 / 69.82
%ΔPower wrt 2D	–	**-9.6 / -9.3 / -4.7**	**-9.6 / -9.3 / -4.7**	**-9.6 / -9.3 / -4.7**

Results of Experiment 3

In this experiment we assess QoR of the A3D flow using the three variants of WL calculation methods, versus the current S2D flow, at iso-performance and iso-area. We use the post-synthesis netlists, clock periods and maximum utilization of six designs obtained using the best 2D flow in Section 5.1.2 and execute the S2D flow in 28nm FDSOI.[76] Table 5.10 shows metrics from the S2D flow. Figures 5.7(a) and (b) compare WL between S2D and A3D variants,

[76]We did not conduct Experiment 3 using 28nm LP owing to non-availability of shrunk LEFs and LEFs required for VI insertion. We omit *spc* as we did not have shrunk LEFs for memory macros.

Table 5.9: Comparison of *spc* metrics from 2D vs. A3D flows using F2F integration in 28nm FDSOI and LP technologies.

	Four Channel Lengths / One Channel Length / LP			
	2D	A3D-GL3D	A3D-WWL	A3D-T3D
#Cells	242012 / 242113 / –	237777 / 237791 / –	237662 / 237679 / –	238025 / 238019 / –
#Buffers	9780 / 9811 / –	8674 / 8683 / –	8617 / 8609 / –	8559 / 8582 / –
#Clk Buffers	1864 / 1866 / –	1871 / 1875 / –	1871 / 1875 / –	1871 / 1875 / –
Cell Area (μm^2)	306684 / 306697 / –	303485 / 303487 / –	303504 / 303520 / –	303471 / 303447 / –
#VIs	–	136779 / 137007 / –	138234 / 138212 / –	134125 / 134132 / –
WL (μm)	8211K / 8211K / –	6323K / 6323K / –	6302K / 6302K / –	6302K / 6302K / –
%ΔWL wrt 2D	–	**-23.0 / -23.0** / –	**-23.3 / -23.3** / –	**-23.3 / -23.3** / –
WNS (ps)	-7 / 0 / –	-8 / 0 / –	-9 / -2 / –	-9 / -2 / –
TNS (ns)	-23.2 / 0 / –	-16.7 / 0 / –	-12.9 / -11.0 / –	-12.7 / -7.7 / –
Power (mW)	307.03 / 302.2 / –	299.4 / 294.6 / –	299.2 / 294.3 / –	299.2 / 294.4 / –
%ΔPower wrt 2D	–	**-2.5 / -2.5** / –	**-2.6 / -2.6** / –	**-2.6 / -2.6** / –

Table 5.10: Metrics from the S2D flow using F2F integration in 28nm FDSOI technology.

	Designs (Four Channel Lengths / One Channel Length)					
Metrics	***aes***	***jpeg***	***ldpc***	***viterbi***	***netcard***	***leon3mp***
#Cells	8478 / 8399	20210 / 20164	45308 / 45305	58497 / 58495	368041 / 368040	448447 / 448433
#Buffers	1029 / 1107	2205 / 2233	6213 / 6210	7846 / 7870	33083 / 33094	46020 / 46031
#Clk Buffers	14 / 14	110 / 111	60 / 62	610 / 614	2222 / 2220	2792 / 2795
Cell Area (μm^2)	10770 / 10692	35569 / 35546	58445 / 58446	121593 / 121594	425874 / 425820	639254 / 639225
#VIs	3374 / 3373	6598 / 6602	15434 / 15446	29474 / 29487	198481 / 198469	129739 / 129782
WL (μm)	105.9K / 105.7K	239.4K / 239.3K	1180.6K / 1080.6K	1064.9K / 1064.9K	7316.9K / 7316.9K	6512.6K / 6512.6K
WNS (ps)	-94 / -9	-113 / -102	-99 / -35	-108 / -65	-149 / -34	-143 / -45
TNS (ns)	-7.7 / -0.07	-13.4 / -34.2	-135.8 / -61.2	-83.6 / -9.2	-544.8 / -44.4	-106.9 / -5.4
Power (mW)	11.68 / 11.78	44.99 / 45.57	150.5 / 152.8	119.6 / 121.3	385.3 / 395.9	215.1 / 223.2

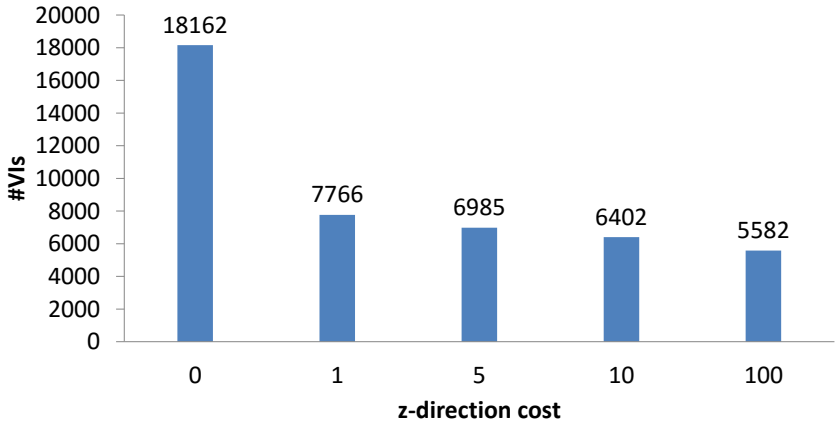


Figure 5.5: Impact on #VIs by only varying the z-direction cost for *jpeg*.

Table 5.11: Comparison of metrics from A3D-T3D vs. A3D-T3D′ flows using F2F integration in 28nm FDSOI technology with four channel lengths.

	aes_cipher_top		*ldpc*	
	A3D-T3D	A3D-T3D′	A3D-T3D	A3D-T3D′
#Cells	8362	8359	42990	42986
#Buffers	859	857	4580	4578
#Clk Buffers	13	12	55	54
Cell Area (μm^2)	10048	10047	49300	49298
#VIs	3266	3264	11866	11865
WL (μm)	80.67K	80.65K	1014K	1014K
%ΔWL wrt A3D-T3D	–	**-0.02**	–	**-0.005**
WNS (ps)	0	0	0	0
TNS (ns)	0	0	0	0
Power (mW)	10.23	10.22	134.4	134.37
%ΔPower wrt A3D-T3D	–	**-0.10**	–	**-0.02**

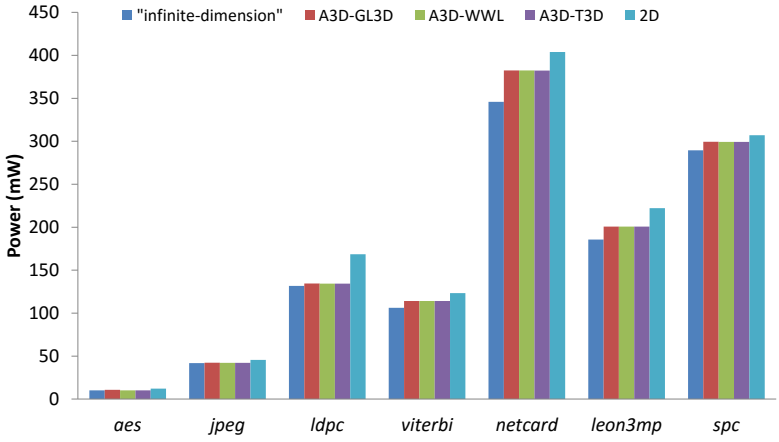


Figure 5.6: Comparison of power using "infinite-dimension" [38], 2D and A3D variants.

and show that A3D-WWL and A3D-T3D achieve up to 24% reduction in WL.[77] Figures 5.8(a) and (b) compare power between S2D and A3D variants, and show that A3D-WWL and A3D-T3D achieve up to 12% reduction in power. Figures 5.9(a) and (b) compare WL and power, respectively between S2D and A3D variants using F2B integration. We observe similar reductions in WL and power by A3D relative to S2D. Across all designs and integration styles, A3D achieves 1.7%–19% reduction in WL, and 0.7%–10.6% reduction in power, relative to S2D in 28nm FDSOI.

[77] We have obtained confirmation from the authors of the S2D flow [44] [157] [195] to report metrics from the S2D flow in this work.

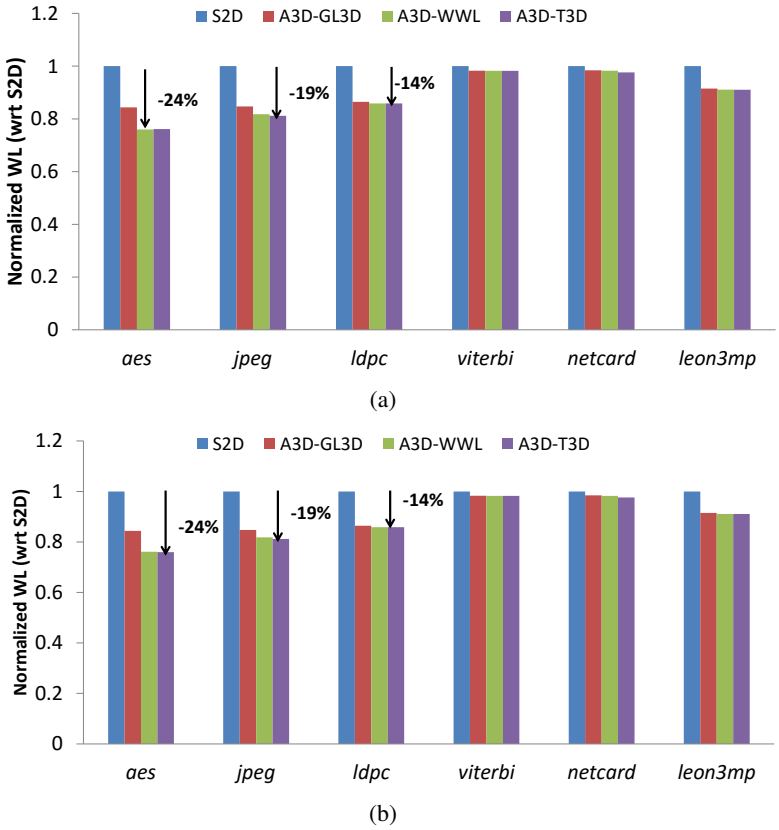
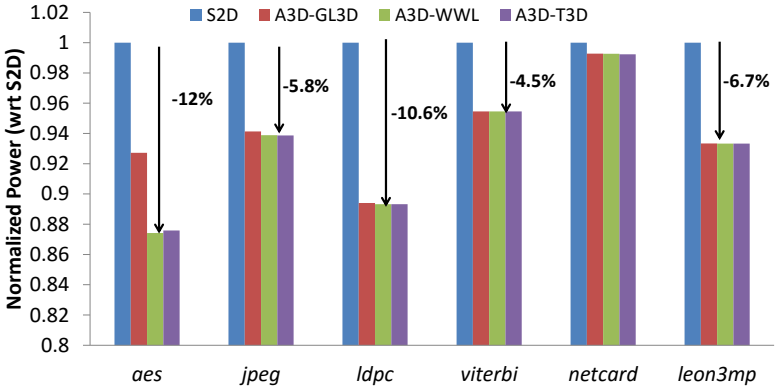


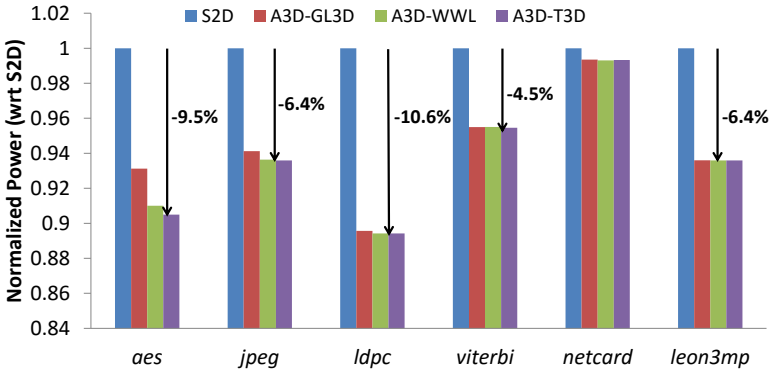
Figure 5.7: Δ(S2D - A3D)% WL using F2F configuration: (a) four channel lengths and (b) one channel length.

5.1.3 Conclusions

3DV calls for "true 3D" implementation flows to comprehend the potential of this implementation technology. We propose a 3D analytic placer, *A3D*, that implements a new "true 3D" wirelength objective and achieves significant routed wirelength and power benefits relative to 2D implementations of real designs. In addition, we apply the Gordian-L net model in 3D, and we propose a new net weighting method based on the ratio of Steiner minimum tree cost and the HPWL of a given net. We implement our three wirelength calculation methods in A3D and propose a 3DIC implementation flow using commercial EDA tools. We demonstrate that our placement solutions are routable in a commercial EDA tool. Compared to 2D implementations, we achieve up to 31% reduction in routed wirelength and 20.2% reduction in power. Compared to the S2D flow, we achieve up to 24% reduction in routed wirelength and 12% reduction in power. We apply a recent "infinite dimension" methodology proposed in [38] to demonstrate that A3D improvements over 2D and S2D are significant relative to the available benefit from



(a)



(b)

Figure 5.8: Δ(S2D - A3D)% Power using F2F configuration: (a) four channel lengths and (b) one channel length.

3D. Future works can include (i) using multi-bit flip-flops and physical synthesis to further close the gap with available benefit from 3D, and (ii) flow improvements by adding power delivery networks and enabling signoff IR drop analysis.

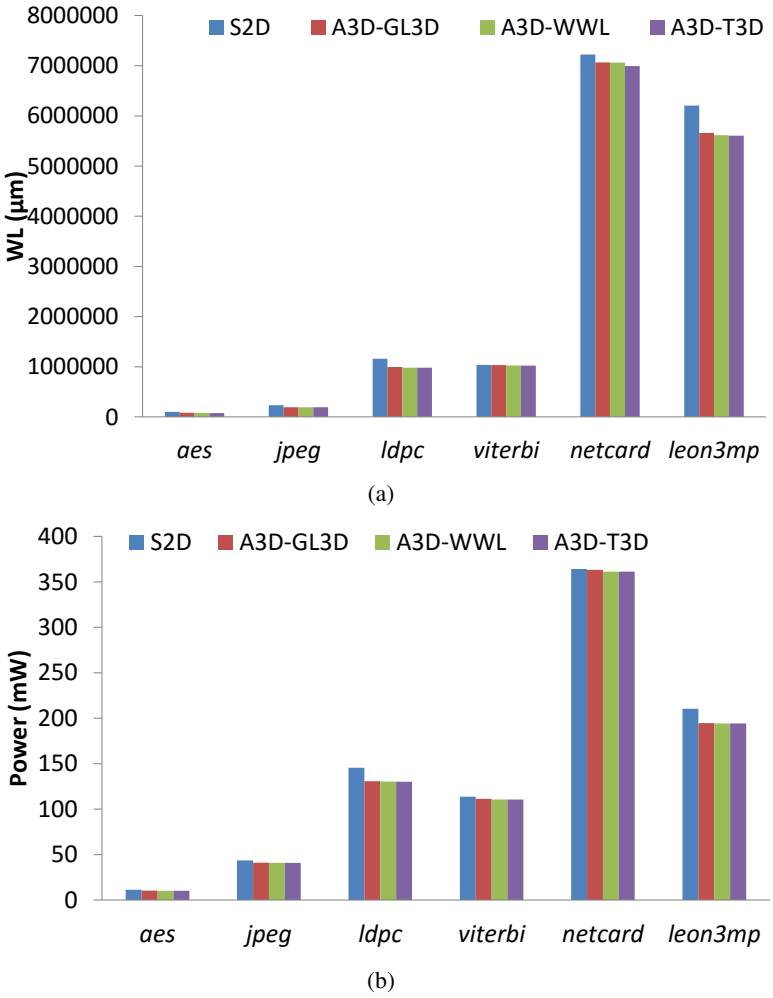


Figure 5.9: Δ(S2D - A3D)% using F2B configuration: (a) WL and (b) power.

5.2 Optimal Scheduling and Allocation for IC Design Management and Cost Reduction

Since 2001 the *International Technology Roadmap for Semiconductors* [304] chapter on Design Technology has presented a Design Cost model calibrated to mobile system-on-chip (SOC) products (e.g., Qualcomm Snapdragon [322], Samsung Exynos [329], etc. that are the main processing cores of tablets and smartphones) and their associated development costs [36] [110] [230]. For well over a decade, the Design Cost model has documented design costs of tens of millions of dollars for a single SOC product. Major contributors to design cost include engineering headcount, compute infrastructure (servers, filers, datacenters), and electronic design automation (EDA) tool licenses. The large investment requirement for new product development stifles semiconductor startup activity and innovation, and has arguably contributed to consolidation and a slowdown of growth for the industry.

Today, a large semiconductor product company will spend hundreds of millions of dollars annually on design infrastructure (datacenters, EDA tools, design teams, etc.) to meet tapeout schedules for multiple concurrent projects. Resources (servers, licenses, engineers, etc.) are limited and must be shared across projects. Not only are schedule slips extremely costly but, as highlighted in recent years (e.g., the "How Green is My Silicon Valley" plenary panel at the 2009 Design Automation Conference (DAC) [294]), there is now tremendous concern to reduce the energy footprint of semiconductor integrated circuit (IC) design. In contrast to traditional scheduling optimizations seen in the operations research and industrial engineering literature, IC design flows often exhibit *co-constraints* between resource types (e.g., one license needed per every two cores used in a multi-threaded tool run[78]). Common design center practices, such as the setting up of dedicated vs. shareable resource pools as permitted by LSF-type gridware [320], also make scheduling and allocation hard. Further, design managers, while increasingly able to track and diagnose design activity [70] [327], have no decision support tools to help determine the resource investments (e.g., "Is it better to add 500 more servers or 50 more timing analysis tool licenses?") that enable schedule requirements to be met with minimum cost. Thus, a company may leave millions of dollars and gigawatt-hours per year – as well as weeks of schedule time – on the table. In a competitive and cost-driven industry, there is an urgent need to recover such wasted resources.

[78] Maintaining design schedules with constant engineering headcount, even as SOC complexities continue to scale, increasingly relies on multi-threading (e.g., detailed routing, static timing analysis, physical verification) and/or massively distributed tool runs (e.g., to perform functional verification).

In the field of operations research, Kolisch et al. [142] [143] [144] give an integer-linear programming (ILP) formulation to solve the *resource-constrained project scheduling problem* (RCPSP). The formulation optimally allocates renewable, non-renewable and doubly-constrained resources across multiple activities (with precedence constraints) in a project. The objective of the formulation is to minimize makespan of a project with multiple activities. We extend this formulation in the context of IC design cost optimization in various ways. Specifically, we describe two mixed integer-linear programming (MILP) formulations that efficiently and optimally perform multi-project multi-resource allocation with complex task precedence and resource co-constraints. The first is the *Schedule Cost Minimization* (SCM) formulation, and the second is the *Resource Cost Minimization* (RCM) formulation. We solve these two general resource-constrained project scheduling problems that arise in a multi-tenanted, heterogeneous, *high-throughput computing* (HTC) environment. A problem instance consists of projects that can be scheduled in parallel, each involving multiple activities, where each activity must consume prescribed amounts of resources to reach completion. The goal is to schedule the projects either with minimum total loss according to given penalty functions, or with minimum number of resources consumed per time unit.[79]

As reviewed in Section 2.3.2, several previous works besides Kolisch et al. [142] [143] [144] address project scheduling problem formulations. Table 5.12 places our work in the context of the works reviewed in Section 2.3.2. While a number of previous works address optimizations related to resource-constrained project scheduling, they cannot address important use cases that arise for large SoC product companies. Our formulations address real-world use cases that incorporate: (i) resource co-constraints, (ii) tethering to forecast resource allocations, and (iii) simultaneous allocation of three different categories of resources (*Fully-shared*, *Segregated*, and *Conditionally-shared*). Our formulations also handle *stability* constraints so that allocation of resources (in particular, engineers) are shuffled as infrequently as possible across projects; this induces a tradeoff between schedule cost and frequency of task switching. Overall, we enable management to identify the minimum cost (in terms of any penalty functions deemed appropriate for the situation) of project completion within a set period of time, capturing many constraint types that arise in the industry. Our solver can also help analyze how varying resource allocation affects cost and schedule of product tapeout. We demonstrate a use case of handling late-breaking bugs in one project without major disruptions in allocations of other projects.

[79] A typical real-world HTC environment has multiple concurrent projects – each working on a specific schedule that is largely non-negotiable, and each having different workload characteristics in terms of infrastructure requirements.

Table 5.12: Representative previous works on project scheduling.

Reference	Formulation	Objective	Modes	Preemptive	Resource Co-constraints	Conditionally-Shared Resources
[11]	ILP	throughput	✗	✗	✗	✗
[17]	LP	makespan	✗	✗	✗	✗
[22]	LP	cost	✓	✓	✗	✗
[25]	CP	throughput	✗	✓	✗	✗
[50]	ILP, LP	cost	✗	✓	✗	✗
[134]	Stochastic ILP	cost	✗	✗	✗	✗
[144], [143]	ILP	makespan	✓	✗	✗	✗
[146]	MILP, LP	cost	✓	✓	✗	✗
[152]	MILP	makespan	✗	✓	✗	✗
[179]	PSO	cost	✗	✓	✗	✗
[203]	ACO	makespan	✗	✗	✗	✗
[217]	ILP	cost	✓	✗	✗	✗
OUR	MILP	cost, makespan	✗	✓	✓	✓

The challenge in practice for a large semiconductor design organization is to provide 'just-in-time' resources for each project, such that (i) project execution is not delayed by resource starvation, and (ii) utilization of each resource type satisfies resource limits or usage policies. Current industry dynamics lead to strict boundary conditions (e.g., time-to-market, tapeout deadline), and constrained capital spending pushes business units to seek increased productivity through maximum utilization of existing resources. Today, resource planning and allocation, especially involving allocation of multiple disparate resource types, have largely been dictated by heuristics and historical experience. Decision support is urgently needed for "course corrections" and understanding of the impact of resource allocation decisions. With this as background, our main contributions in this section are as follows.

1. We model two resource-constrained optimal project scheduling formulations, SCM and RCM, as MILPs. Our formulations handle multiple projects, multiple activities with precedence constraints, and multiple types of resources.

2. We handle *co-constraints* between resource types and allocation of resources from multiple (fully-shared, conditionally-shared, segregated) resource pools. Each pool may have a different penalty function, capturing real-world scenarios in a large SOC design company. To our knowledge, we are the first to consider co-constraints between resource types.

3. We optionally enforce *stability* constraints that upper-bound the change in a project's allocated resources between successive timesteps.

4. Application of SCM to a three-project scheduling problem extracted from a leading-edge design center of Company X[80] shows substantial compute and license cost savings compared to the actual allocation/scheduling solution used by the product company. Our solution reduces the schedule makespan of all projects by 1.4 work-weeks,[81] i.e., $\sim$2.7% of annual design infrastructure cost. (Per "Moore's Law", the semiconductor industry advances at $\sim$1% in a calendar week [349]. Therefore, during this time the semiconductor industry advances by more than 1%.) We also demonstrate the scheduling of two dozen chip development projects at the design center level, subject to resource and datacenter capacity limits as well as per-project penalty functions for schedule slips. The design center was unable to solve this problem and ended up purchasing 600 additional servers to avoid schedule slips. Our solution shows that the schedule requirements could have been met without purchasing any additional servers.

5. Application of RCM to a four-project scheduling problem extracted from a leading-edge design center of Company X shows substantial human resource costs left on the table by the actual allocation/scheduling solution used by the company. For a particular activity related to chip design, our solution reduces head count by 37%, which translates to $\sim$\$5M savings at that particular (non-U.S.) design center. Our solver can also provide decision support via "what-if" analyses of cost and schedule tradeoffs.

6. Of separate interest is the description of our testcase generator that we use to perform scalability and sensitivity studies. We propose to make our generator and solvers open-source as prototyped at [350].

5.2.1 Problem Formulations

We now present (i) notations used in our discussion, (ii) resource categories that arise in multi-tapeout project scheduling, and (iii) our MILP formulations. We have spent considerable time working with technical management at one of a world top-5 semiconductor company's design centers, to arrive at the optimization formulations described below. Table 5.13 gives notations used in our work. "I" represents an input to the MILP and "O" represents an optimization variable. We also indicate which notations are used in each of the SCM and RCM formulations.

[80] Owing to confidentiality reasons we cannot reveal the name of the company, so we refer to it as Company X henceforth.

[81] In the semiconductor industry, we typically refer to one "work-week" as five working days in a week.

Table 5.13: Notations used in SCM and RCM formulations.

Parameter	Description	I/O	Formulation
N	Total number of projects	I	SCM, RCM
T	Maximum duration over all projects; $t = 1,2,...,T$	I	SCM, RCM
P_i	Projects indexed by $i = 1,2,...,N$	–	SCM, RCM
$J(i)$	Total number of activities for P_i	I	SCM, RCM
$a_{i,j}$	P_i's activities, where $j = 1,2,...,J(i)$	–	SCM, RCM
$H^a(i,j)$	Set of predecessor activities of $a_{i,j}$ that must complete before $a_{i,j}$ can start	I	SCM, RCM
K	Available resource types	I	SCM, RCM
$R_{k,t}$	Upper bound (UB) on # resources of type k at time t	I	SCM
$H^r(i,j,k)$	Set of predecessor resources for resource type k for $a_{i,j}$	I	SCM
$g(i,j,h,k,t)$	Function that sets an UB on # resource type k at any time t, for each predecessor $h \in H^r(i,j,k)$	I	SCM
$L_{i,j,k}$	# resources of type k required to complete $a_{i,j}$	I	SCM, RCM
$U_{k,t}$	UB on # fully-shared resources of type k at time t	I	SCM
$\tilde{U}_k$	UB on # fully-shared resources of type k at any time t	O	RCM
$V_{i,k,t}$	UB on # segregated resources of type k for P_i at time t	I	SCM
$M_{i,k,t}$	UB on # conditionally-shared resources of type k for P_i at time t	I	SCM
$G_{i,k,t}$	UB on total # resources of type k used by P_i at time t	I	SCM, RCM
$B_{i,k,t}$	UB on change in resources consumed by P_i from $t-1$ to t	I	SCM, RCM
$d_{i,j}^{nom}$ (e_i^{nom})	Nominal duration of $a_{i,j}$ (P_i)	I	SCM, RCM
$C_{i,j}^a(t)$ ($C_i^p(t)$)	Penalty function for $a_{i,j}$ (P_i) at time t	I	SCM, RCM
C_k	Weight for resource type k	I	RCM
C	Cost of switching activities/projects	I	SCM+
$w_{i,j,k,t}$	# resources of type k consumed by $a_{i,j}$ at time t, given by forecast resource allocation	I	SCM
δ	% of variation allowed in $w_{i,j,k,t}$	I	SCM
$s_{i,j}^{nom}$ ($f_{i,j}^{nom}$)	Nominal start (finish) time of $a_{i,j}$ for tethering constraints	I	SCM, RCM
$r_{i,j,k,t}$	# fully-shared resources of type k consumed by $a_{i,j}$ at time t	O	SCM, RCM
$q_{i,j,k,t}$	# segregated resources of type k consumed by $a_{i,j}$ at time t	O	SCM
$y_{i,j,k,t}$	# conditionally-shared resources of type k consumed by $a_{i,j}$ at time t	O	SCM
$z_{i,j,k,t}$	# unused conditionally-shared resources of type k consumed by $a_{i,j}$ at time t	O	SCM
$s_{i,j}$ ($f_{i,j}$)	Start (finish) time of $a_{i,j}$	O	SCM, RCM
$S_{i,j,t}$ ($F_{i,j,t}$)	0-1 variable, set to 1 if $t \geq s_{i,j}$ ($t \geq f_{i,j}$); 0 otherwise	O	SCM, RCM

Resource Pool Types

Chip design companies typically have three pools for each resource type. (Resource types include compute nodes, memory, storage, people, etc. [204].)

Fully-shared resources are shared across all projects. We use $r_{i,j,k,t}$ to denote the number of fully-shared resources of type k used by activity $a_{i,j}$ of project P_i at time t. For example, if there are two projects P_1 and P_2 with one activity each, and 20 fully-shared resources of type k are available, then P_1 and P_2 can share these 20 resources among themselves such that $r_{1,1,k,t} + r_{2,1,k,t} \leq 20$.

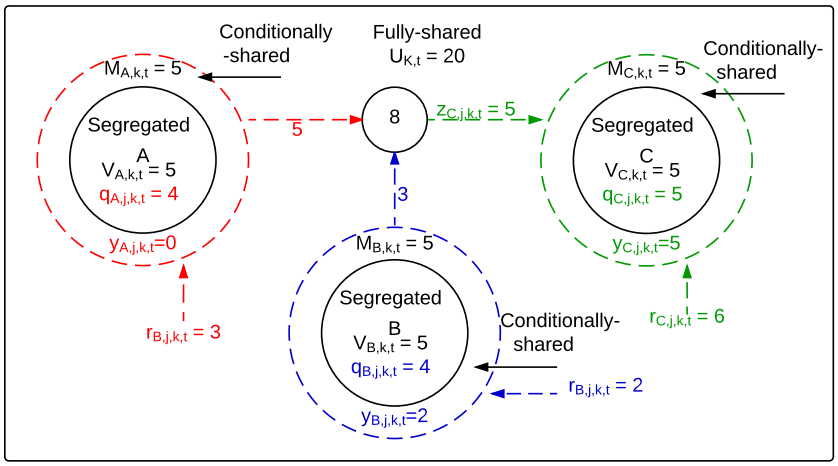
Segregated/dedicated resources are allocated exclusively to a specific project. These resources are not available for use by any other projects at any time. We use $q_{i,j,k,t}$ to denote the number of segregated resources of type k used by activity $a_{i,j}$ of project P_i, at time t. For example, if there

are two projects P_1 and P_2 with one activity each, and they are respectively allocated 10 and 20 segregated resources of type k, then $q_{1,1,k,t} \leq 10$, and $q_{2,1,k,t} \leq 20$.

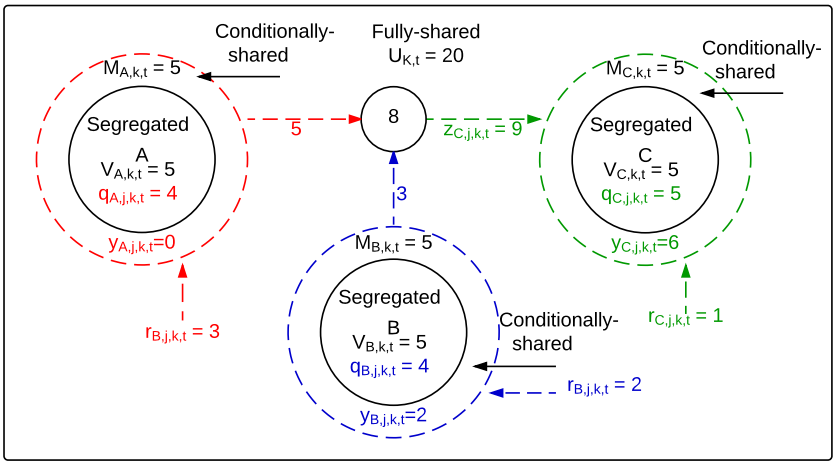
Conditionally-shared resources are allocated to each project, but any resource unused by a project may be used by other projects. We use $y_{i,j,k,t}$ to denote the number of conditionally-shared resources of type k used by activity $a_{i,j}$ of project P_i, at time t. For example, if there are two projects P_1 and P_2 with one activity each, and they are respectively allocated 10 and 20 conditionally-shared resources of type k, then $y_{1,1,k,t} \leq 10$, and $y_{2,1,k,t} \leq 20$. We use the notation $z_{i,j,k,t}$ to denote the number of resources of type k that is used by activity $a_{i,j}$ of project P_i at time t, from the pool of unused conditionally-shared resources of other projects. In the preceding example, we have $z_{1,1,k,t} \leq 20 - y_{2,1,k,t}$, and $z_{2,1,k,t} \leq 10 - y_{1,1,k,t}$.

Figure 5.10 illustrates two scenarios with three projects A, B and C. Each project has one activity and consumes resource type k at time t. Each project may use resources from any of the three pools with the following constraints: (i) segregated resources $q_{i,j,k,t}$ consumed by a project cannot exceed the upper bound $V_{i,k,t}$ as shown in Figure 5.10, and (ii) conditionally-shared resources $y_{i,j,k,t}$ consumed by a project cannot exceed $M_{i,k,t}$. Figure 5.10(a) shows a feasible allocation of resources from each pool. Projects A and B have a total of eight units of unused resources in their conditionally-shared pools after allocation of resources from each pool.[82] Project C uses five out of these eight units, i.e., $z_{C,j,k,t} = 5$. The total number of fully-shared resources consumed by all three projects, i.e., $3+2+6 = 11$, cannot exceed $U_{k,t} = 20$. Figure 5.10(b) shows an allocation of resources that is infeasible because $y_{C,j,k,t} = 6 > M_{C,k,t} = 5$. Furthermore, project C uses more resources from the unused conditionally-shared resource pool lable, i.e., $z_{C,j,k,t} > 8$.

[82] Projects A, B and C use 4, 4 and 5 resources from their respective segregated pools, which are within the upper bounds $V_{A,k,t} = V_{B,k,t} = V_{C,k,t} = 5$.



(a)



(b)

Figure 5.10: Examples showing (a) feasible and (b) infeasible allocations of resources among three projects *A*, *B* and *C*.

MILP Description of the Schedule Cost Minimization (SCM) Formulation

Given the inputs listed in Table 2 for the SCM formulation, we seek to minimize the total cost (i.e., sum of schedule penalties) of all projects:

$$\text{minimize} \quad \sum_{i=1}^{N}\sum_{t=1}^{T} C_i^p(t) + \sum_{i=1}^{N}\sum_{j=1}^{J(i)}\sum_{t=1}^{T} C_{i,j}^a(t) \tag{5.16}$$

This optimization is subject to the following constraints.[83]

Constraints on start and finish times. Constraint (5.17) indicates that all $S_{i,j,t}$ and $F_{i,j,t}$ are binary variables. Constraints (5.18) and (5.19) respectively establish the relation between $s_{i,j}$ and $S_{i,j,t}$, and between $f_{i,j}$ and $F_{i,j,t}$. Constraint (5.20) sets all $S_{i,j,t}$ and $F_{i,j,t}$ to zero before the start time of the first activity of the project (if $s_{i,1}^{nom}$ is not given, we assume the project can start at $t = 1$, i.e., $s_{i,1}^{nom} = 1$) [25] [143] [146]. Constraint (5.21) (resp. Constraint (5.22)) prevents each start $S_{i,j,t}$ (resp. finish $F_{i,j,t}$) indicator variable from having a value of zero once an activity has started (resp. finished) execution [25] [143] [146]. Constraint (5.23) ensures that an activity's start time precedes its finish time [11] [25] [50] [239].

$$\forall i,\ \forall j,\ \forall t,\ \ S_{i,j,t}, F_{i,j,t} \in \{0,1\} \tag{5.17}$$

$$\forall i,\ \forall j,\ \ s_{i,j} = T - \left(\sum_{t=1}^{T} S_{i,j,t}\right) + 1 \tag{5.18}$$

$$\forall i,\ \forall j,\ \ f_{i,j} = T - \left(\sum_{t=1}^{T} F_{i,j,t}\right) \tag{5.19}$$

$$\forall i,\ \forall j,\ \forall t < s_{i,1}^{nom},\ \ S_{i,j,t} = 0,\ F_{i,j,t} = 0 \tag{5.20}$$

$$\forall i,\ \forall j,\ \forall t,\ \ S_{i,j,t} \geq S_{i,j,t-1} \tag{5.21}$$

$$\forall i,\ \forall j,\ \forall t,\ \ F_{i,j,t} \geq F_{i,j,t-1} \tag{5.22}$$

$$\forall i,\ \forall j,\ \ \sum_{t=1}^{T} S_{i,j,t} \geq \sum_{t=1}^{T} F_{i,j,t} \tag{5.23}$$

Constraint on activity precedence. Constraint (5.24) assures precedence requirements: all predecessors of an activity $a_{i,j}$ must complete before its start time $s_{i,j}$ [11] [17] [134] [143].

$$\forall i,\ \ s_{i,j} > f_{i,h},\ \forall h \in H^a(i,j) \tag{5.24}$$

[83] In our description, we point to example references that adopt similar formulations.

Constraint: Upper bounds on resource consumptions. Constraints (5.25) and (5.26) upper-bound the total number of resources of each type that are used at time t, summed over all activities of all projects (each project). Recall that we use $y_{i,j,k,t}$ to denote the number of conditionally-shared resources of type k that are used by activity $a_{i,j}$ of project P_i at time t. We use $z_{i,j,k,t}$ to denote the number of conditionally-shared resources of type k that are used by activity $a_{i,j}$ of project P_i at time t from the pool of unused conditionally-shared resources of other projects. That is, $z_{i,j,k,t}$ denotes the number of resources borrowed from other projects. Constraints (5.27) – (5.34) ensure that an activity does not use any resources before it starts or after it ends [22] [50] [239]. For example, Constraint (5.27) ensures that no resources are used before the activity starts ($S_{i,j,t} = 0$, $\forall t < s_{i,j}$ which forces $r_{i,j,k,t} = 0$, $\forall t < s_{i,j}$) and Constraint (5.28) ensures that no resources are used after the activity finishes ($F_{i,j,t} = 1$, $\forall t > f_{i,j}$ which forces $r_{i,j,k,t} = 0$, $\forall t > f_{i,j}$). Constraint (5.29) also sets an upper bound on the number of segregated resources of type k used by $a_{i,j}$. Constraint (5.35) sets an upper bound on the total number of fully-shared resources of type k used by all activities of all projects. Constraint (5.36) sets an upper bound on the total number of conditionally-shared resources of type k used by all activities of P_i. Constraint (5.37) ensures that the total number of resources used by all the projects from the unused conditionally-shared resource pool is not greater than the number of resources available in the pool. Constraints (5.38) and (5.37) together ensure that a project does not receive resources from its own contribution to the unused conditionally-shared resource pool.

$$\forall k,\ \forall t,\ \ \sum_{i=1}^{N}\sum_{j=1}^{J(i)}(r_{i,j,k,t} + q_{i,j,k,t} + y_{i,j,k,t} + z_{i,j,k,t}) \leq R_{k,t} \tag{5.25}$$

$$\forall i,\ \forall k,\ \forall t, \sum_{j=1}^{J(i)}(r_{i,j,k,t} + q_{i,j,k,t} + y_{i,j,k,t} + z_{i,j,k,t}) \leq G_{i,k,t} \tag{5.26}$$

$$\forall i,\ \forall k,\ \forall j,\ \forall t,\ \ r_{i,j,k,t} \leq U_{k,t} \times S_{i,j,t} \tag{5.27}$$

$$\forall i,\ \forall k,\ \forall j,\ \forall t,\ \ r_{i,j,k,t} \leq U_{k,t} \times (1 - F_{i,j,t}) \tag{5.28}$$

$$\forall i,\ \forall k,\ \forall j,\ \forall t,\ \ q_{i,j,k,t} \leq V_{i,k,t} \times S_{i,j,t} \tag{5.29}$$

$$\forall i,\ \forall k,\ \forall j,\ \forall t,\ \ q_{i,j,k,t} \leq V_{i,k,t} \times (1 - F_{i,j,t}) \tag{5.30}$$

$$\forall i,\ \forall k,\ \forall j,\ \forall t,\ \ y_{i,j,k,t} \leq M_{i,k,t}(S_{i,j,t}) \tag{5.31}$$

$$\forall i,\ \forall k,\ \forall j,\ \forall t,\ \ y_{i,j,k,t} \leq M_{i,k,t}(1 - F_{i,j,t}) \tag{5.32}$$

$$\forall i,,\ \forall j,\ \forall k,\ \forall t,\ \ z_{i,j,k,t} \leq \sum_{p=1}^{N} M_{p,k,t} S_{i,j,t} \tag{5.33}$$

$$\forall i,,\ \forall j,\ \forall k,\ \forall t,\ \ z_{i,j,k,t} \leq \sum_{p=1}^{N} M_{p,k,t}(1 - F_{i,j,t}) \tag{5.34}$$

$$\forall k,\ \forall t,\ \ \sum_{i=1}^{N} \sum_{j=1}^{J(i)} r_{i,j,k,t} \leq U_{k,t} \tag{5.35}$$

$$\forall i,\ \forall k,\ \forall t,\ \ \sum_{j=1}^{J(i)} y_{i,j,k,t} \leq M_{i,k,t} \tag{5.36}$$

$$\forall k,\ \forall t,\ \ \sum_{i=1}^{N} \sum_{j=1}^{J(i)} z_{i,j,k,t} \leq \sum_{i=1}^{N} (M_{i,k,t} - \sum_{j=1}^{J(i)} y_{i,j,k,t}) \tag{5.37}$$

$$\forall i,\ \forall k,\ \forall t,\ \ \sum_{j=1}^{J(i)} z_{i,j,k,t} \leq \sum_{p \neq i} (M_{p,k,t} - \sum_{j=1}^{J(p)} y_{p,j,k,t}) \tag{5.38}$$

Constraint: Resource requirements of activities. Constraint (5.39) ensures the completion of an activity [217].

$$\forall i,\ \forall k,\ \forall j, \sum_{t=1}^{T} (r_{i,j,k,t} + q_{i,j,k,t} + y_{i,j,k,t} + z_{i,j,k,t}) = L_{i,j,k} \tag{5.39}$$

Constraint: Resource co-constraints. Constraint (5.40) ensures that the number of resources of type k used by activity $a_{i,j}$ satisfies the upper bound constraints implied by the co-constraints between its predecessor resources (cf. Table 5.13). For instance, let the number of used resources of type $k = 1$ (e.g., compute nodes) be upper-bounded by $2\times$ the number of used resources of type $k = 2$ (e.g., static timing analysis (STA) licenses) at all times for $a_{1,1}$, i.e., at most two compute nodes can be used for every STA license. Therefore, $H^r(1,1,1) = \{2\}$ and $g(1,1,2,1,t) = 2$ at all times. The constraint will set $(r_{1,1,1,t} + q_{1,1,1,t} + y_{1,1,1,t} + z_{1,1,1,t}) \leq 2 \times (r_{1,1,2,t} + q_{1,1,2,t} + y_{1,1,2,t} + z_{1,1,2,t}),\ \forall t$. Note that this constraint is specific to each activity of a project, and not for the entire project. To the best of our knowledge, previous works do not handle such co-constraints, as reviewed in Section 2.3.2.

$$\begin{aligned} &\forall i, \forall j,\ \forall k,\ \forall t,\ \forall h \in H^r(i,j,k), \\ &(r_{i,j,k,t} + q_{i,j,k,t} + y_{i,j,k,t} + z_{i,j,k,t}) \leq g(i,j,h,k,t) \times (r_{i,j,h,t} + q_{i,j,h,t} + y_{i,j,h,t} + z_{i,j,h,t}) \end{aligned} \tag{5.40}$$

Constraint: Stability in resource allocation. Constraints (5.41) and (5.42) ensure stability in the consumption of resources, for each project. That is, we upper-bound the change in the

quantity of each resource used by any given project between successive timesteps t and $t-1$. In the real world, resources such as engineers may work on activities related to multiple projects in a day. However, major changes to allocations do not, as a practical matter, occur within short time windows. For example, if 100 engineers work on an activity of Project A for a week, reassigning 80 of them to work only on Project B in the following week would be undesirable to see in management's plans.

$$\begin{aligned}
&\forall i,\ \forall k,\ \forall t,\\
&\sum_{j=1}^{J(i)}(r_{i,j,k,t}+q_{i,j,k,t}+y_{i,j,k,t}+z_{i,j,k,t})-\\
&\sum_{j=1}^{J(i)}(r_{i,j,k,t-1}+q_{i,j,k,t-1}+y_{i,j,k,t-1}+z_{i,j,k,t-1}) \le B_{i,k,t} \qquad (5.41)\\
&\sum_{j=1}^{J(i)}(r_{i,j,k,t-1}+q_{i,j,k,t-1}+y_{i,j,k,t-1}+z_{i,j,k,t-1})-\\
&\sum_{j=1}^{J(i)}(r_{i,j,k,t}+q_{i,j,k,t}+y_{i,j,k,t}+z_{i,j,k,t}) \le B_{i,k,t} \qquad (5.42)
\end{aligned}$$

Constraint: Tethering forecast resource allocations. Constraints (5.43) and (5.44) ensure that a project's forecast resource allocation is not modified by more than a certain degree (indicated by δ). Specifically, no forecast value in the active period ($s^{nom}_{i,j} \le t \le f^{nom}_{i,j}$) of the activity[84] can be perturbed by more than $\delta\%$ in the MILP solution. Constraint (5.45) ensures that activity $a_{i,j}$ consumes exactly the amount of resources needed, according to the forecast resource allocation, for its completion.

$$\forall i,\ \forall j,\ \forall k,\ \forall\, s^{nom}_{i,j} \le t \le f^{nom}_{i,j}, (r_{i,j,k,t}+q_{i,j,k,t}+y_{i,j,k,t}+z_{i,j,k,t}) \ge w_{i,j,k,t}(1-\delta/100) \qquad (5.43)$$

$$\forall i,\ \forall j,\ \forall k,\ \forall\, s^{nom}_{i,j} \le t \le f^{nom}_{i,j}, (r_{i,j,k,t}+q_{i,j,k,t}+y_{i,j,k,t}+z_{i,j,k,t}) \le w_{i,j,k,t}(1+\delta/100) \qquad (5.44)$$

$$\forall i,\ \forall j,\ \forall k,\ \sum_{t=1}^{T}(r_{i,j,k,t}+q_{i,j,k,t}+y_{i,j,k,t}+z_{i,j,k,t}) = \sum_{t=1}^{T} w_{i,j,k,t} \qquad (5.45)$$

Intuition behind the variables included in the model. We choose input parameters and optimization variables based on typical usages in IC design companies. We use $H^a(i,j)$ to enforce precedence relations among the activities of a project (e.g., parasitic extraction cannot start until the design has completed routing; STA cannot start until the design has been synthesized; or STA

[84] If a schedule cannot be pulled in, then the lower bound on t should be 1 (instead of $s^{nom}_{i,j}$) in Constraints (5.43) and (5.44).

with signal integrity cannot start until the design has been placed). We use $R_{k,t}$ because companies typically budget for a certain number of resources during the planning phase. However, they may increase the number of resources of a particular type during the project's execution when they realize that its deadline cannot be met without these additional resources. The time-dependent variable allows us to handle such changes in our formulation. We introduce $H^r(i,j,k)$ and $g(i,j,h,k,t)$ to handle co-constraints between resource types. For instance, at most two compute nodes can be used for each STA license used. Similar to $R_{k,t}$, we use $U_{k,t}$ as the upper bound on the number of fully-shared resources, which can change over time. For example, when a project's deadline becomes risky to meet, units of resources may be removed from the shared pool and allocated to the dedicated pool of the project four work-weeks before tapeout (TO). We use $B_{i,k,t}$ to achieve a stable allocation, since resources should not be drastically shuffled ("whipsawed") across projects in consecutive units of time. For instance, we may not want to allocate 100 engineers to a project on Day 1, but only five engineers on Day 2.

Penalty functions in the objective. The objective function can be any function that is linear in the optimization variables presented in Table 5.13. We use an objective function that minimizes the sum of two schedule-related penalties over all projects [180]. The first penalty is for the overall duration of each project relative to the nominal duration of the project. The second penalty is for the duration of each activity in each project relative to the nominal duration of the activity. Commonly used penalty functions are: **Ramp**: penalty due to each successive day of schedule slip increases linearly as we move further past the deadline (therefore, the total penalty is quadratic in number of days in the slip); **Step**: penalty due to each successive day of slip is constant (and the total penalty is linear in the magnitude of schedule slip); **Delta**: total penalty for slip is constant (does not depend on the extent of the slip). We use nominal duration of the activities (and projects) to penalize the schedule. The nominal finish time of a project is calculated using the nominal start time of the first activity of the project $s_{i,1}^{nom}$ and the nominal duration of the project e_i^{nom}. The nominal finish time of $a_{i,j}$ can be calculated using the nominal start time of the activity and the nominal duration of the activity, i.e., $f_{i,j}^{nom} = s_{i,j}^{nom} + d_{i,j}^{nom}$, where $s_{i,j}^{nom} = max\{1 + f_{i,h}^{nom}\}$, over all $h \in H^a(i,j)$, or $s_{i,j}^{nom} = s_{i,1}$ if $H^a(i,j) = \emptyset$.

Complexity of the MILP. Even in a large SoC product company, number of projects $N \leq 30$, number of activities per project $J(i) \leq 20$, number of resource types $K \leq 10$, and $T \leq 300$ when the unit of time is days. There are $(2 \times N \times J(i) + 2 \times N \times J(i) \times T + 4 \times N \times K \times J(i) \times T)$

variables ($= 2 \times 30 \times 20 + 2 \times 30 \times 20 \times 365 + 4 \times 30 \times 10 \times 20 \times 365 \approx 9M$, for 365 days). We note that actual values of N, K, T, etc. will likely be smaller than these bounds. If necessary, to reduce the number of variables, we can change the unit of time from days to weeks or months. In our experiments, we use *IBM ILOG CPLEX v12.6* [296] as our solver and the runtime of our MILP is around 45 seconds for a total of $\sim$10K variables, and 9 minutes for a total of $\sim$100K variables, and 52 minutes for a total of $\sim$500K variables (see also Figure 5.17 below).

Notice that there are two types of input scenarios that can lead to infeasible solutions.

- If the value of T (maximum duration over all projects) is not large enough for all projects to finish within that duration, *CPLEX* will report that the MILP is infeasible.
- Infeasibility can also arise due to inconsistent resource constraints. For example, if 20 units of resource A and 10 units of resource B are required for the completion of an activity of a project, but the co-constraint is such that to use one unit of A, one unit of B must be used, infeasibility arises because we will never be able to use more than 10 units of A.

Example of SCM. We now describe the SCM problem formulation, with the help of a small example. Table 5.14 shows the values of input variables and their meaning.

Optimal solution. We seek to minimize the schedule makespan of both projects for this example. Table 5.15 shows one of the possible optimal solutions for the example problem. Both of the projects can be completed by $t = 4$. (Resource utilization for each activity is shown only for the first resource. The utilization for the second resource is identical.) We note that $a_{1,2}$ utilizes five units of the first resource at $t = 4$ from the unused conditionally-shared pool of P_2. The formulation is able to capture the notion that if a project is not using any of its conditionally-shared resources, then those resources can be used by other active projects.

Table 5.14: Input variables and their meanings for the SCM example.

Variable and value	**Meaning**
$N = 2$	There are two projects P_1 and P_2.
$T = 10$	The maximum duration over both projects is 10.
$J(1) = 2$	There are two activities $a_{1,1}$ and $a_{1,2}$ in project P_1.
$J(2) = 1$	There is one activity $a_{2,1}$ in project P_2.
$H^a(1,2) = 1$,	In P_1, activity a_1 must complete before activity a_2.
$K = 2$	There are two types of resources.
$R_{1,t} = 40$, $R_{2,t} = 40$	At most 40 units of either resource can be used at any point in time.
$H^r(1,1,2) = 1$, $H^r(1,2,2) = 1$, $H^r(2,1,2) = 1$	The first resource is a predecessor resource for the second resource for all activities and projects.
$g(1,1,1,2,t) = 1\ \forall t$, $g(1,2,1,2,t) = 1\ \forall t$, $g(2,1,1,2,t) = 1\ \forall t$	One unit of the first resource must be used before using one unit of the second resource by any activity at any point in time.
$L_{1,1,1} = 60$, $L_{1,2,1} = 65$, $L_{1,1,2} = 60$, $L_{1,2,2} = 65$, $L_{2,1,1} = 30$, $L_{2,1,2} = 30$	60 units of each resource are required to complete activity $a_{1,1}$; 65 units of each resource are required to complete activity $a_{1,2}$; and 30 units of each resource are required to complete activity $a_{2,1}$.
$U_{1,t} = 20\ \forall t$, $U_{2,t} = 20\ \forall t$	At most 20 units of each resource are fully-shared between both projects at any point in time.
$V_{1,1,t} = 5$, $V_{1,2,t} = 5\ \forall t$, $V_{2,1,t} = 5$, $V_{1,2,t} = 5\ \forall t$	Each project has five units of each resource that are segregated, i.e., they can be used only by activities of that project at any point in time. These resources are not shared with other projects.
$M_{1,1,t} = 5$, $M_{1,2,t} = 5\ \forall t$, $M_{2,1,t} = 5$, $M_{1,2,t} = 5\ \forall t$	Each project has five units of each resource that are conditionally-shared at any point in time, i.e., they can be used by activities of other projects if they are unused by the project.
$G_{1,1,t} = 35$, $G_{1,2,t} = 35\ \forall t$, $G_{2,1,t} = 35$, $G_{1,2,t} = 35\ \forall t$	At most 35 units of either resource can be used by either project at any point in time.

Table 5.15: Consumption of the first resource for both the projects in an optimal solution.

	Activities											
	$a_{1,1}$				$a_{1,2}$				$a_{2,1}$			
Time t	$r_{1,1,1,t}$	$q_{1,1,1,t}$	$y_{1,1,1,t}$	$z_{1,1,1,t}$	$r_{1,2,1,t}$	$q_{1,2,1,t}$	$y_{1,2,1,t}$	$z_{1,2,1,t}$	$r_{2,1,1,t}$	$q_{2,1,1,t}$	$y_{2,1,1,t}$	$z_{2,1,1,t}$
1	20	5	5	0	0	0	0	0	0	5	5	0
2	20	5	5	0	0	0	0	0	0	5	5	0
3	0	0	0	0	20	5	5	0	0	5	5	0
4	0	0	0	0	20	5	5	5	0	0	0	0
Total (for each activity)	60				65				30			

MILP Description of the Resource Cost Minimization (RCM) Formulation

Given the inputs listed in Table 5.13 for the RCM formulation, we seek to minimize the total number of resources required and the total cost (i.e., sum of schedule penalties) of all projects:

$$\text{minimize} \quad \sum_{i=1}^{K} C_k \tilde{U}_k + \sum_{i=1}^{N} \sum_{t=1}^{T} C_i^p(t) + \sum_{i=1}^{N} \sum_{j=1}^{J(i)} \sum_{t=1}^{T} C_{i,j}^a(t) \tag{5.46}$$

This optimization is subject to the following constraints.

Constraints on start and finish times. We use Constraints (5.17) – (5.23) as in the SCM Formulation (Section 5.2.1).

Constraint on activity precedence. We use Constraint (5.24) as in the SCM Formulation (Section 5.2.1).

Constraint: Upper bounds on resource consumptions. Constraints (5.47) and (5.48) upper-bound the number of resources of each type used at time t across all activities of all projects (each project). Constraints (5.49) and (5.50) ensure that an activity does not use any resources before it starts or after it ends.

$$\forall i,\ \forall k,\ \forall t, \sum_{j=1}^{J(i)} r_{i,j,k,t} \leq G_{i,k,t} \tag{5.47}$$

$$\forall k,\ \forall t,\ \sum_{i=1}^{N} \sum_{j=1}^{J(i)} r_{i,j,k,t} \leq \tilde{U}_k \tag{5.48}$$

$$\forall i,\ \forall k,\ \forall j,\ \forall t,\ \ r_{i,j,k,t} \leq G_{i,k,t} \times S_{i,j,t} \tag{5.49}$$

$$\forall i,\ \forall k,\ \forall j,\ \forall t,\ \ r_{i,j,k,t} \leq G_{i,k,t} \times (1 - F_{i,j,t}) \tag{5.50}$$

Constraint: Resource requirements of activities. Constraint (5.51) ensures the completion of an activity.

$$\forall i,\ \forall k,\ \forall j, \sum_{t=1}^{T} r_{i,j,k,t} = L_{i,j,k} \tag{5.51}$$

Constraint: Stability in resource allocation. We modify Constraints (5.41) and (5.42) from the SCM Formulation as follows, to ensure stability in the consumption of resources for each project.

$$\forall i,\ \forall k,\ \forall t,\quad \sum_{j=1}^{J(i)}(r_{i,j,k,t}) - \sum_{j=1}^{J(i)}(r_{i,j,k,t-1}) \leq B_{i,k,t} \tag{5.52}$$

$$\forall i,\ \forall k,\ \forall t,\quad \sum_{j=1}^{J(i)}(r_{i,j,k,t-1}) - \sum_{j=1}^{J(i)}(r_{i,j,k,t}) \leq B_{i,k,t} \tag{5.53}$$

5.2.2 Experimental Setup and Results

In this section, we describe computational studies using three multi-project scheduling problem instances taken from a large design center (tens of market-leading system-on-chip product tapeouts per year) of a world top-5 semiconductor company, referred to henceforth as Company X. The results show a potential for significant resource savings (data center provisioning, EDA tool licenses, people, and schedule) from our MILP formulations, when compared to the scheduling solutions actually used by Company X's design center. We also show the scaling of solver runtime with instance parameters.

Table 5.16: Activity requirements (per block) for each project.

Activity	#core	#mem	L1	L2	L3	Hrs.
1. A1	1	1	1			12
2. A2	4	2	2			24
3. A3	4	2	2			72
4. A4	4	2		1		8
5. A5 (per corner)	8	8			1	4
6. A6	4	4			1	12
7. A7	4	2		1		8
8. A8 (per corner)	8	8			1	4
9. A9	8	8		1	1	24
10. A10	4	2		1		8
11. A11 (per corner)	8	8			1	4

Schedule Modification Use Case

The first industry problem instance has $N = 3$ projects, each in the final pass of implementation, within an overall timeline of $T = 90$ days. The three projects P_1, P_2 and P_3 respectively contain 15, 10 and 10 "hard macro" blocks. As listed in Table 5.16, there are 11 activities associated with each project ($a_{i,1}$ = A1, ..., $a_{i,11}$ = A11).[85] The table shows that each activity, *per block*, uses some amount of each of five resource types: compute cores, units of memory (e.g., a unit might be 16GB RAM), and tool licenses of types L1, L2 and L3.[86] Further, activities A5, A8 and A11 per block are performed at 75 corners and two modes (functional and test).[87] The projects have access to the following total amounts of these resources: (i) compute cores = 4800, (ii) units of memory = 4800, (iii) L1 licenses = 50, (iv) L2 licenses = 30, and (v) L3 licenses = 400.[88]

Additional constraints governing the projects and the scheduling solution are: (i) for each project, the activities must follow a given precedence order, as shown in Figure 5.11 – for example, in Project P_2, activities $a_{2,1}, a_{2,2}, a_{2,3}$ and $a_{2,4}$ must all be completed before activity $a_{2,5}$ can commence, but there are no ordering constraints among $a_{2,1} - a_{2,4}$; (ii) at any point in time, the number of compute cores consumed cannot exceed 10 times the number of tool licenses consumed, and also cannot exceed twice the number of units of memory consumed; (iii) each project is given a 30% allocation of the 4800 total compute cores (i.e., as segregated resources), with the remaining 10% of the compute cores being fully-shared resources; and (iv) no project can use more than 350 L3 licenses, 40 L1 licenses, or 60% of the total supply of any other type of resource (compute cores, memory units, L2 licenses) at any time.

MILP solution using SCM. Our SCM MILP formulation straightforwardly allows capture of the above-described multi-tapeout project scheduling problem. All projects can be completed within the 90-day limit. In one optimal solution, projects P_1, P_2 and P_3 are completed in 59, 39 and 34 days, respectively. Figures 5.12(a)–(c) show the resource consumption profiles of the three projects, where no stability constraints, i.e., Constraints (5.41) and (5.42), are imposed.

[85] Please see Table 5.20 in Section 5.2.4 for a mapping of activities and resources to actual chip design flow terminologies.

[86] According to [229], leading exemplars of these resources include EDA tools such as Cadence's *Innovus* [286], *Assura QRC* [284] and *Tempus Timing Signoff* [288], and Synopsys's *IC Compiler* [340], *Star-RCXT* [343] and *PrimeTime-SI* [342]. The EDA tools used in the production design flows studied here cannot be specifically revealed here, but are from this set.

[87] Thus, for example, performing A5, A8 or A11 activity for a 10-block chip will require 10 (blocks) $\times$ 75 (corners) $\times$ 2 (modes) $\times$ 8 (cores) $\times$ 4 (hours) = 48000 core-hours of compute resource.

[88] In this instance, there are $\sim$65K variables and $\sim$980K constraints. The runtime of the solver is around 9 minutes.

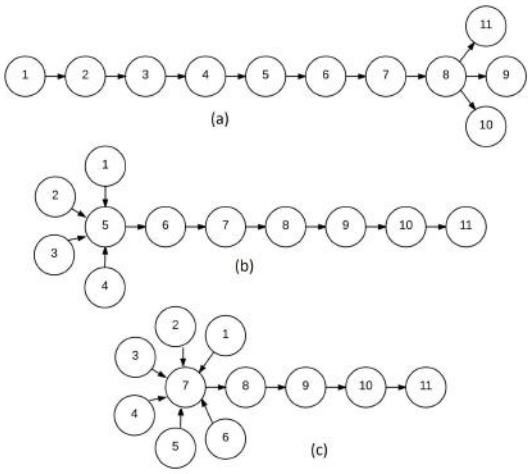
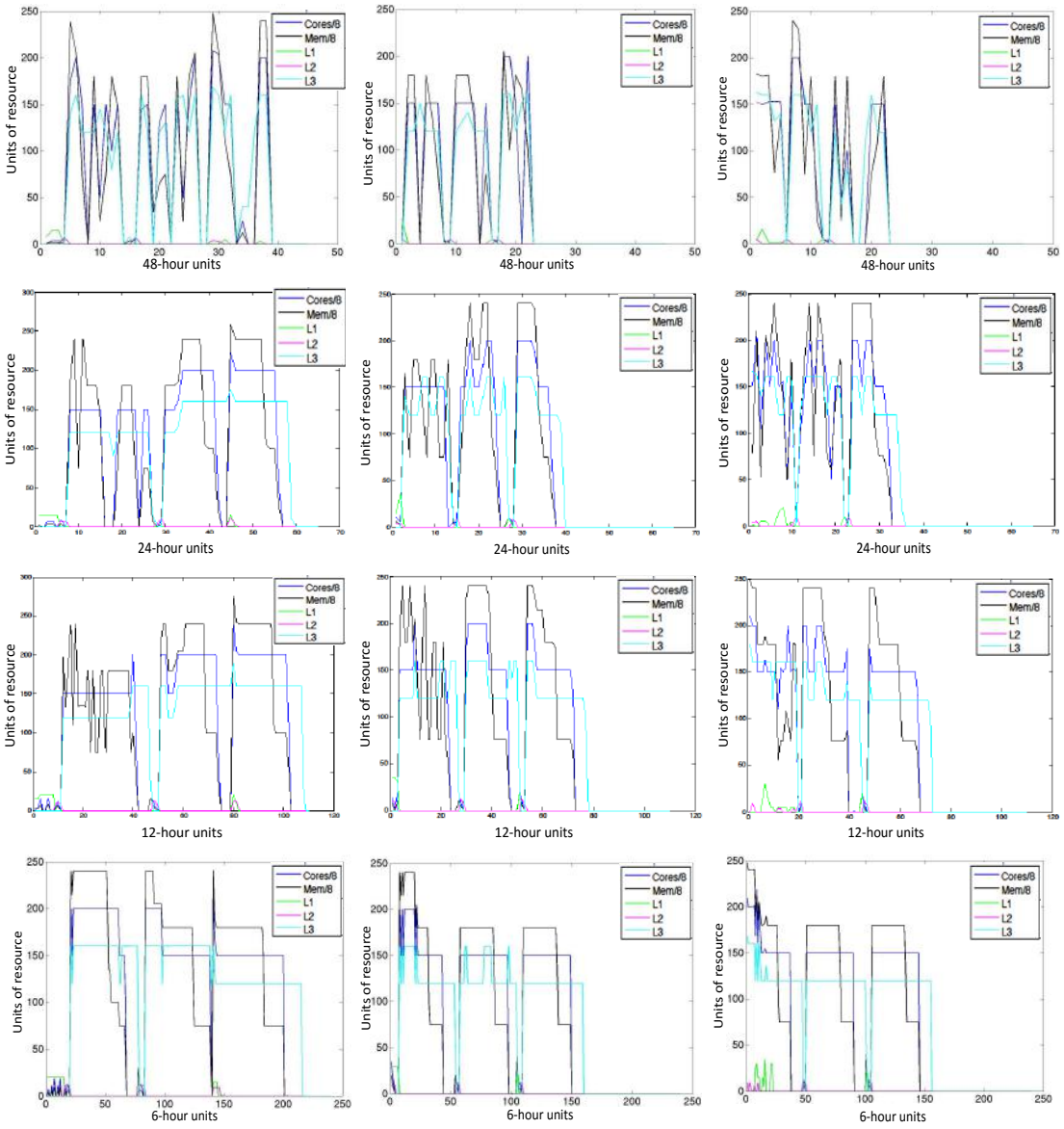


Figure 5.11: Precedence order of activities in projects (a) P_1, (b) P_2, and (c) P_3.

From top to bottom the schedules for each project are shown at 48-hour, 24-hour, 12-hour and six-hour granularities, respectively. When timestep granularity is coarsest (48-hour), MILP runtime is around 3.4 minutes and resource consumptions switch rapidly across activities, whereas when timestep granularity is finest (six-hour), MILP runtime is around 2.2 hours and makespan tightens as compared to the makespan of 48-hour granularity solutions.

Schedule modification. The salient problem that we address here is one of late-breaking schedule changes. After projects are initially scheduled, there can arise a need to modify some of the instance parameters during schedule execution (e.g., due to a design bug and resulting Engineering Change Order (ECO)), then re-solve for the project schedules from that point on. Here, a late-breaking bug (i.e., a bug that is found and fixed very late in the schedule) in the behavioral description of the design for project P_2 caused large-scale changes. In the actual project, this led to a push-out of activity $a_{2,8}$ (A8), which in turn pushed out all downstream activities for the project. As a result, there is a need to determine optimal scheduling of the remaining activities, i.e., where P_1 resumes from $a_{1,5}$ on, P_2 resumes from $a_{2,8}$ on, and P_3 has only its last activity $a_{3,11}$ remaining to be scheduled. An optimal MILP solution for the "from the moment of the ECO onward" scheduling problem is shown in Figures 5.13(a), (c) and (e) with projects P_1, P_2, P_3. The solution actually used in the company design center is shown in Figures 5.13(b), (d) and (f). In the MILP solution, all three projects are completed by 34 extra days from the point of the late breaking bug, while the industry solution takes 41 extra days for completion. Our MILP solution could thus have saved 1.4 work-weeks in the schedule makespan of the three projects.[89]

[89] According to the actual industry solution, each of the projects P_2 and P_3 should be given 50% of the resources until P_3 is completed. This entails that P_1 will be given 50% of the resources while P_2 and P_3 (for cleanup) get 20%



(a) Project P_1 (b) Project P_2 (c) Project P_3

Figure 5.12: MILP solutions for projects (a) P_1 (b) P_2, and (c) P_3 at 48-hour, 24-hour, 12-hour and six-hour granularities from top to bottom, respectively. For readability we have scaled down the values of cores and storage memory as Cores/8 and Mem/8.

of the resources each, and the number of fully-shared resources is restored to 10%. Since we do not consider cleanup activity (of P_3) to get the optimized solution, we re-allocate P_3's resources to P_1 (10%, as no project can consume more than 60% of the resources) and P_2 (the remaining 10%) for fair comparison of the solutions.

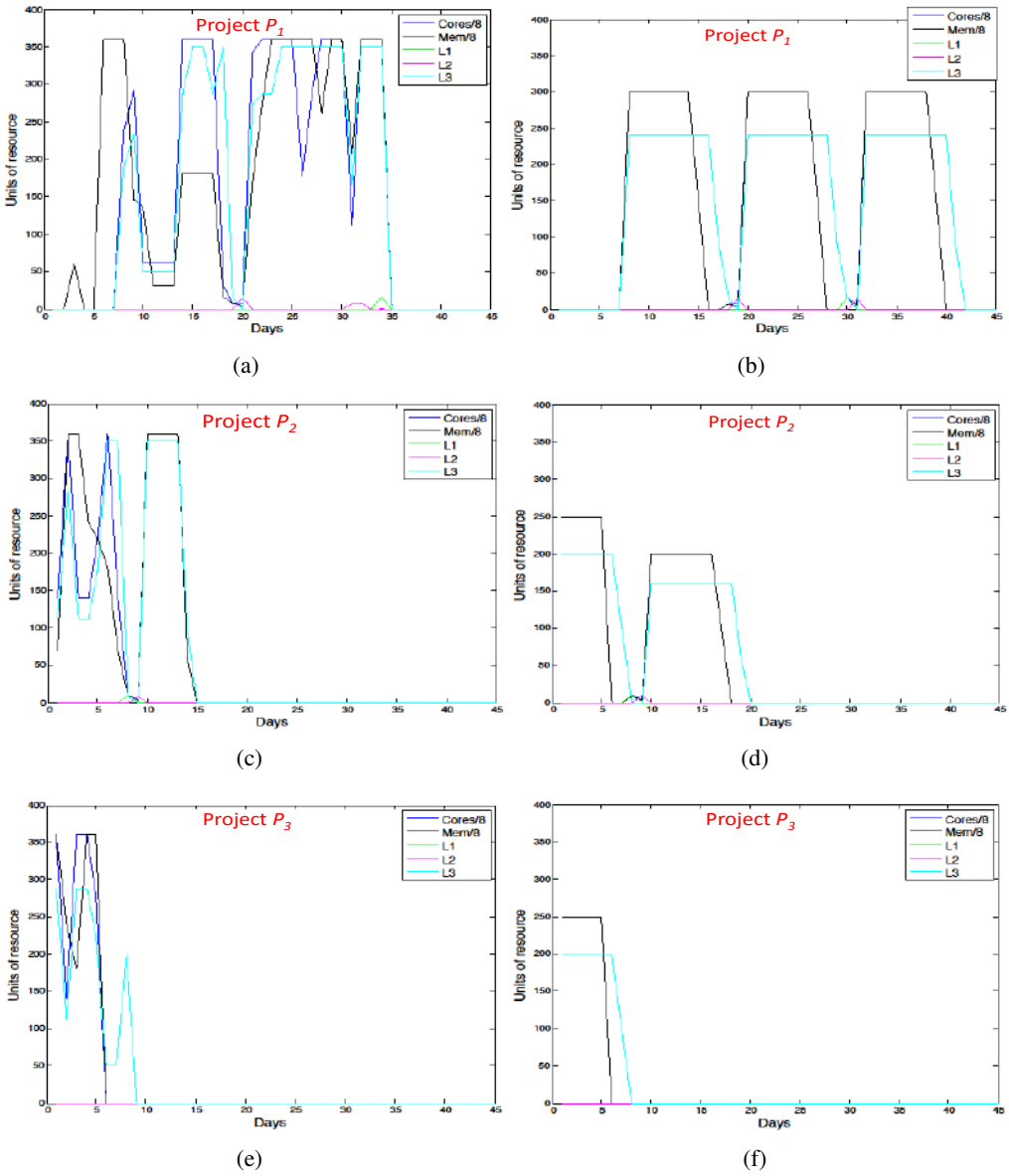


Figure 5.13: Solutions of (a) MILP for P_1, (b) industry for P_1, (c) MILP for P_2, (d) industry for P_2, (e) MILP for P_3, and (f) industry for P_3 with a late-breaking RTL bug in project P_2. Cores and Mem are exactly overlapping in the industry solution in all three figures. For readability we have scaled down the values of cores and storage memory as Cores/8 and Mem/8.

Scheduling Tethered to Forecasts

The SCM MILP formulation can be extended, with a few additional constraints (and corresponding inputs), to address a *forecast-tethered* resource allocation problem. The use case is that we are given (typically, bottoms-up from project owners) a forecast resource allocation for activity $a_{i,j,k}$ and its consumption $w_{i,j,k,t}$ of resource type k. The optimal solution must satisfy

the Constraints (5.43)–(5.45). Figure 5.14(a) illustrates the allocation for one project, and Figure 5.14(b) illustrates a consumption forecast over time for three identical such projects. At times, forecast consumption is greater than the upper bounds of resources (e.g., servers/datacenter capacity), hence the allocation is infeasible. Figure 5.14(c) shows a feasible scheduling that is obtained by modifying the forecast resource allocation within upper bounds, thereby constraining the consumption peaks to be within bounds.

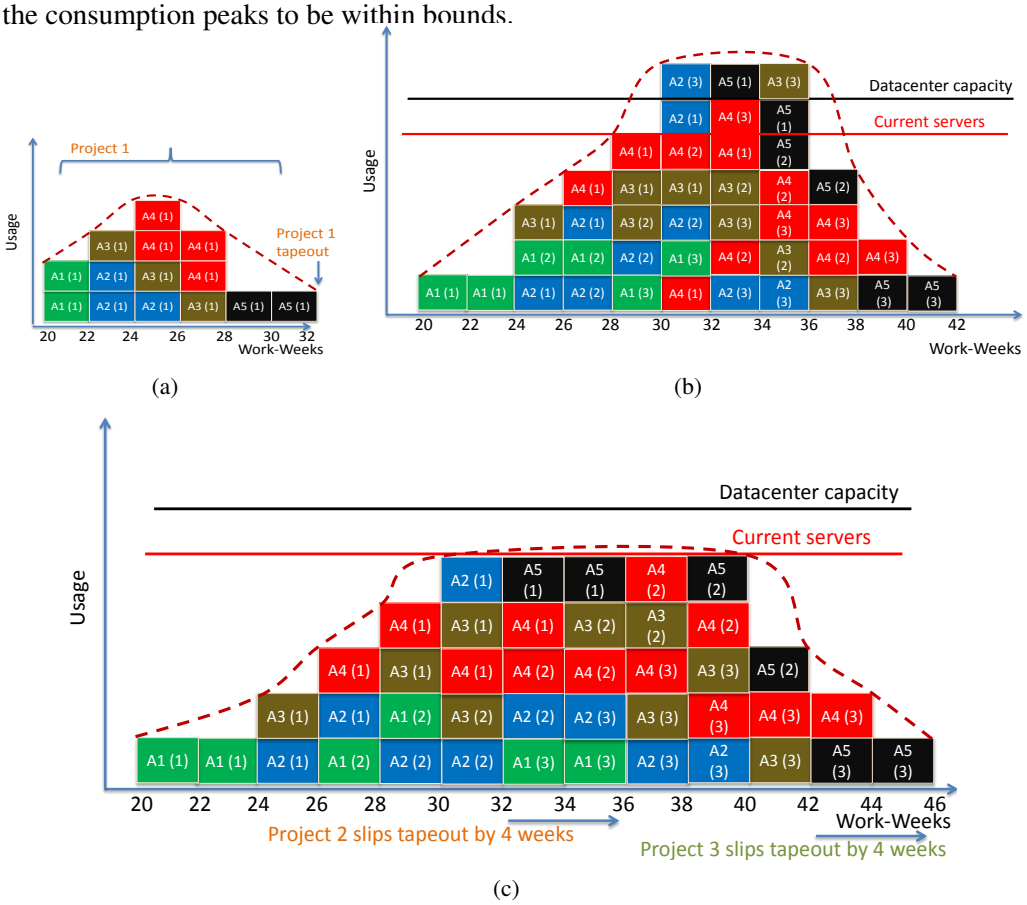


Figure 5.14: Forecast allocation for (a) one project, (b) three such projects – infeasible, and (c) a feasible MILP-derived allocation.

We find an optimal schedule by tethering an instance of an industrial forecast resource allocation from the design center of Company X. The instance consists of 24 projects along with the forecast resource consumption of each project from November 2014 to September 2015. The total forecast resource consumption, over all the projects, is greater than the current servers (and datacenter capacity) during certain months. Therefore, we optimize the allocation so as to bound the consumption within $R_{k,t}$ (i.e., the current servers or datacenter capacity). We consider two variants: (i) pull-in of the project schedule, and (ii) reduction of the amounts of shared allocations from the upper bound $R_{k,t}$. Table 5.17 summarizes the experiments we conduct for this instance.

$CS = 1560$ denotes the number of current servers, and $DC = 2100$ denotes the datacenter capacity. *fs-in* denotes whether the fully-shared resources (210 units of *CS* or *DC*) are included in $R_{k,t}$; *pi-en* denotes whether pull-in is enabled; *pi* denotes the number of months by which the schedule is pulled in; and *po* denotes the number of months by which the schedule is pushed out. Our penalty functions for schedule changes (pull-in or push-out) per month are as follows: no penalty when the change is <5% of the forecast duration of the project; penalty function pen_1 for changes between 5% and 30% of the duration; and penalty function pen_2 for changes beyond 30% of the duration. Usually, pen_2 is significantly higher than pen_1. Furthermore, there are two types of projects – *committed* and *proposed*. Committed projects are penalized more than proposed projects when not adhering to the forecast schedule. Values in parentheses show the total number of months that the committed projects are either pushed out or pulled in.[90]

Table 5.17: Resource allocations tethered to forecasts.

$R_{k,t}$	δ (%)	***fs-in***	***pi-en***	pen_1	pen_2	**# *po***	**# *pi***
CS	30	✓	✗	ramp	step	6 (3)	-
CS	30	✗	✗	infeasible			
CS	40	✗	✗	ramp	step	14 (8)	-
CS	40	✓	✗	ramp	step	3 (0)	-
CS	30	✓	✗	step	delta	15 (8)	-
DC	30	✓	✗	ramp	step	**0 (0)**	-
DC	30	✗	✗	ramp	step	**0 (0)**	-
CS	30	✓	✓	ramp	step	5 (2)	5 (0)

Note that for the allocation to be bounded by $R_{k,t}$, δ must be sufficiently large such that tethering can bring the total consumption, for each of the months, to be within $R_{k,t}$. For example, if the total forecast consumption for a month is 100 units and $R_{k,t}$ is 70 units, then $\delta \geq 30\%$ to obtain a feasible solution. The maximum CPU time needed to solve any of the instances in Table 5.17 is only a few seconds, since the unit of T is months and $T = 11$. The design center management of Company X could not solve this problem. Their solution was to purchase the additional 600 servers required to meet committed project forecast demands during the months of peak execution. However, our solver can provide an allocation that does not require the purchase of additional servers while still meeting the schedule.

[90] In this instance, there are ∼1K variables and ∼4.5K constraints. The runtime of the solver is around 3.2 seconds.

(Human) Resource Allocation Use Case

Our third industry instance has $N = 4$ projects, each with a makespan of 16 work-weeks (or, 80 days). Each of the four projects P_1, P_2, P_3 and P_4 has eight activities with assigned "billable man-weeks", i.e., total amount of human resources needed to complete each activity. Four types of human resources (HR1, ..., HR4) are available for each project. Table 5.18 shows the resource requirement for each project across multiple activities.[91] Project P_4 begins first; the start dates of projects P_3, P_2 and P_1 are respectively offset by five, nine and five work-weeks (there are five days in a work-week) relative to the start date of project P_4. The precedence graph for activities for each project is shown in Figure 5.15. In addition, all resources are fully-shared across the four projects.[92]

From a scheduling standpoint, design management would typically like to assess time-to-market versus resource costs. Our solver allows "what-if" analyses to understand these trade-offs. To understand resource costs when time-to-market is critical, we set the makespan of each project to 16 work-weeks. Figure 5.16(a) compares our RCM MILP solution with the industry solution. The RCM MILP solution reduces the maximum amount of resources required in any work-week, $\tilde{U}_1$ (for HR1) by 13.5% (185 units to 160 units), $\tilde{U}_2$ (for HR2) by 37.5% (240 units to 150 units), $\tilde{U}_3$ (for HR3) by 25.5% (200 units to 149 units), and $\tilde{U}_2$ (for HR4) by 30% (130 units to 91 units). Such reduction in the number of human resources required can result in highly significant cost savings for a company. For example, one unit of a HR resource costs \$56 per hour in a non-U.S. location [204]. The resource works 8 hours per day for 5 days per week. The overall makespan of all four projects is 26 work-weeks. Therefore, reducing $\tilde{U}_2$ by 90 units for HR2 saves $56 \times 8 \times 5 \times 26 \times 90 \sim$ \$5.2M for the company. To understand resource costs when time-to-market can be relaxed, we have evaluated a solution in which we set the makespan of each project to 20 work-weeks. Figure 5.16(b) compares MILP solutions for 20 work-weeks with those for 16 work-weeks. The relaxed project makespans enable further reductions in the maximum amount of resources required in any work-week: $\tilde{U}_3$ (for HR3) by 10.7% (149 units to 133 units), $\tilde{U}_4$ (for HR4) by 14.3% (91 units to 78 units), and $\tilde{U}_2$ (for HR2) by 16.7% (150 units to 125 units), relative to our solutions for the 16 work-week project makespan. The additional reduction of $\tilde{U}_2$ by 25 units for HR2 alone can result in further savings of $(56 \times 8 \times 5 \times 26 \times 150) - (56 \times 8 \times 5 \times 29 \times 125) \sim$ \$0.62M for the company.

[91] Table 5.20 in Section 5.2.4 provides a mapping of activities and resources to chip design flow terminologies.

[92] In this instance, there are ~6K variables and ~31K constraints. The runtime of the solver is around 18 seconds.

Table 5.18: Billable engineer work-weeks for each activity for each project.

		Project			
Activity	**Resource type used**	**Project P_1**	**Project P_2**	**Project P_3**	**Project P_4**
1. A12	HR1	140	145	45	160
2. A13	HR1	420	425	45	500
3. A14	HR2	115	100	45	200
4. A15	HR2	345	300	145	580
5. A16	HR3	870	990	140	640
6. A17	HR3	80	260	30	50
7. A18	HR2	220	300	90	390
8. A19	HR4	480	550	180	540

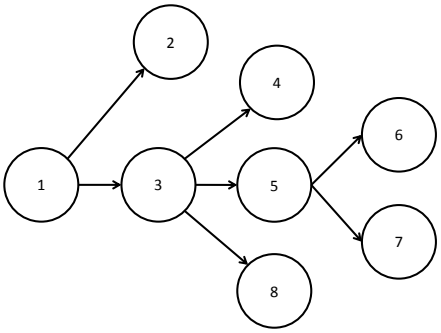
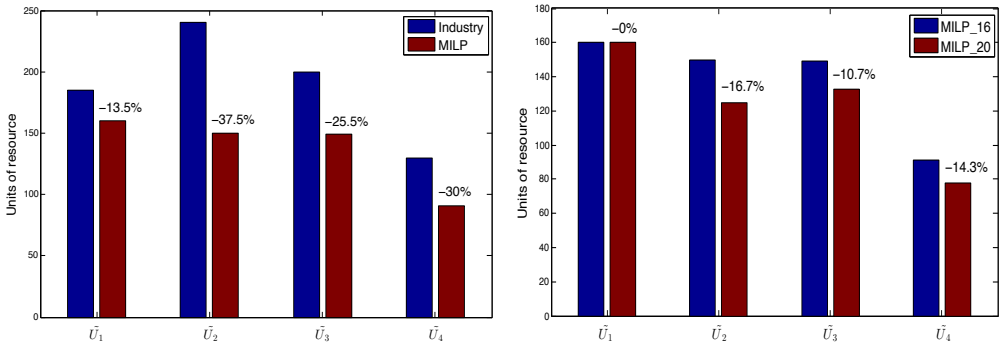


Figure 5.15: Activity precedence for all projects.



(a) (b)

Figure 5.16: Comparison of allocations: (a) Industry vs. MILP with makespan of all projects set to 16 work-weeks and (b) MILP solutions when the makespan of all projects are 16 vs. 20 work-weeks. $\tilde{U}_{i=1,2,3,4}$ is the maximum over total amount of HR$\{1,2,3,4\}$ units consumed in any work-week.

Artificial Testcase Generator

We have separately developed a generator of random multi-tapeout project scheduling instances, where parameters such as N, T, $J(i)$, $R_{k,t}$, $U_{k,t}$, $V_{i,k,t}$, $M_{i,k,t}$, $G_{i,k,t}$, and d_{nom} (cf. Table

5.13) are all Gaussian random variables, and various pairs of resources may be co-constrained. Further, the randomly generated instances can have different topologies of precedence constraints (cf. Figure 5.11). Our generator is implemented in *Python*; it takes in values of N, T, $J(i)$, $R_{k,t}$, $U_{k,t}$, $V_{i,k,t}$, $M_{i,k,t}$, $G_{i,k,t}$, and d_{nom} as command-line arguments, and generates a problem instance input as illustrated in the example in Section 5.2.1. IC design companies typically deal with $\sim$30 projects with known priorities (set by marketing teams and management). There is usually one ordering of these projects and not $N!$ permutations, so we do not study all possible orderings of projects. In particular, we do not exhaustively enumerate instances as in [143]. As demonstrated above, our MILP can handle ≤ 30 simultaneous projects whose priorities have already been decided.[93]

Scalability Studies

Furthermore, we have also studied the scalability of our optimal solution approach with respect to *CPLEX v12.6* solver runtimes. We use artificial testcases from our generator described in Section 5.2.2 for our scalability studies. Figure 5.17 shows the sensitivity of *CPLEX* runtime to changes in various instance parameters, relative to a base instance configuration of $N = 6,\ J = 8,\ T = 200,\ K = 6$ (the red point shown in each of the plots in the figure). Each plot sweeps one of the instance parameters as: (i) $N = \{2, 4, 6, 8\}$, (ii) $J = \{2, 4, 6, 8, 10\}$ ($J(i) = J\ \forall i = \{1,\ldots,N\}$), (iii) $T = \{150, 200, 300\}$ days, and (iv) $K = \{2, 4, 6, 8\}$. All other parameters are fixed at the base configuration values. The parameter being varied is shown on the x-axis; runtime (seconds) is shown on the y-axis. Here, all projects are identical, i.e., they all have the same number of activities, and corresponding activities have the same resource requirements.

Schedule granularity. Our problem formulation discretizes time, with the unit t potentially representing hours, days, months, etc. A more granular time unit permits more accurate modeling at the cost of runtime. Recall that Figures 5.12(a)–(c) showed solutions to the industrial schedule modification instance by varying timesteps at 48-hour, 24-hour, 12-hour and six-hour

[93] We have run our solver on 480 testcases for the *j30* benchmark from PSPLIB [321], for which optimal solutions for each testcase are available. Other benchmarks such as *j60*, *j90* and *j120* do not have optimal solutions posted in PSPLIB [321]. This benchmark has one project with 30 activities, and each testcase varies the following: (i) precedence constraints between activities, and (ii) upper bounds of resources for each activity over various timesteps. For each testcase, renewable resources map to $R_{k,t}$, and non-renewable and doubly-constrained resources map to $G_{i,k,t}$ in our MILP. For non-renewable resources we set the same upper bound for all t. We have confirmed that MILP solutions are the same as optimal solutions posted in [321] for *j30*. The runtimes of both PSPLIB and MILP solutions differ by $\pm 3\%$ when the respective solver implementations (in *CPLEX v12.6*) are run on an Intel Xeon E5-1410 server at 2.80GHz.

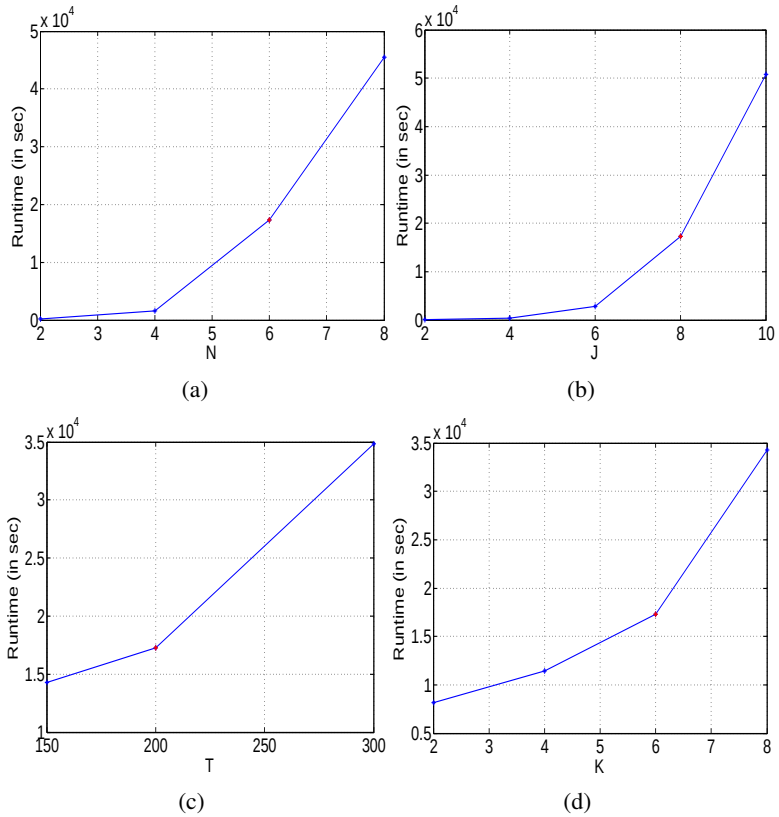


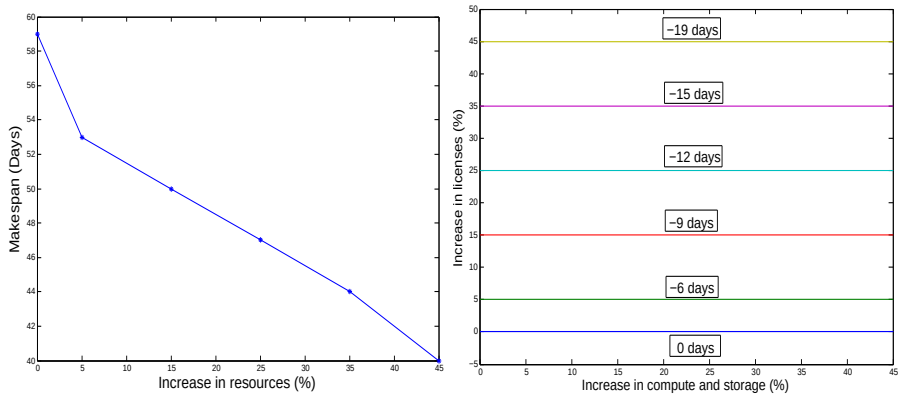
Figure 5.17: Runtime variation with parameters: (a) N, (b) $J(i)$, (c) T, and (d) K.

granularities from top to bottom, respectively. As expected, we see slightly tighter makespans for the projects at the finest granularity (six-hour) than at the coarsest granularity (48-hour). The solver runtime increases from 3.4 seconds (with 48-hour solution granularity) to 2.2 hours (with six-hour solution granularity).

Sensitivity Studies

We analyze the effect of the upper bounds (on the resources) on the optimal schedule. In the first industry instance described in Section 5.2.2, we increase the upper bounds of the resources, that is, we proportionately increase $R_{k,t}$, $U_{k,t}$, $V_{i,k,t}$ and $G_{i,k,t}$. Figure 5.18(a) shows that the makespan decreases when the upper bounds of *all* the resources increase.

We also create another instance in which we increase only upper bounds of compute server and storage resources, and do not change the upper bounds of other resources. From the Pareto curves in Figure 5.18(b), we can see that in this instance at Company X, additional compute servers and storage would not help *at all*. In this particular instance, the number of licenses is the bottleneck, and makespan can be improved only if more licenses are made available. (This also shows effects of the resource co-constraints, i.e., additional compute and storage resources



(a) (b)

Figure 5.18: (a) Impact on makespan when the upper bounds on resources are increased. (b) Pareto curves for changes to resource upper bounds.

cannot be maximally utilized due to shortage of licenses.) From our interactions with senior management at the Company X design center, we understand that these types of sensitivity analyses can be very useful for resource planning and procurement.

Engineer Allocation Studies

Last, we study *stable allocation* of certain resource types such as engineers, who cannot be rapidly re-allocated to different projects or activities. We modify the objective of our SCM problem as follows.

$$\begin{aligned} \text{minimize} \quad & C \sum_{k=1}^{K} \sum_{i=1}^{N} \sum_{j=1}^{J(i)} \sum_{t=2}^{T} |r_{i,j,k,t} - r_{i,j,k,t-1}| + \\ & \sum_{i=1}^{N} \sum_{t=1}^{T} C_i^p(t) + \sum_{i=1}^{N} \sum_{j=1}^{J(i)} \sum_{t=1}^{T} C_{i,j}^a(t) \end{aligned} \tag{5.54}$$

This optimization is subject to start and finish times, resource upper bounds and activity precedence constraints as described in Section 5.2.1. We consider each engineer as a resource type, so $R_{k,t} = 1,\ \forall t$. In consecutive timesteps $t-1$ and t, the engineer is either working on an activity $a_{i,j}$ of Project P_i, or working on another activity either of the same project or a different project. When the engineer is working on the same activity, there is no switching between consecutive timesteps and the absolute difference of $|r_{i,j,k,t} - r_{i,j,k,t-1}|$ is zero. However, when the engineer works on a different activity or project, then the absolute difference is one, and is multiplied by the fixed cost C of switching activities or projects. The total number of switches made by the engineer is multiplied by C in the objective function to obtain the total cost of switching. Our objective is to minimize the cost of switching across all engineers.

To verify the above formulation, we use our generator described in Section 5.2.2 and create four input instances $E_{1,2,3,4}$ with five, 10, 30 and 30 projects, respectively. Each instance has one activity per project (i.e., $J(i) = 1, \forall i = 1, \ldots, N$), and the number of engineers varying between 100 and 250 (i.e., $K = 100$, $K = 150$, or $K = 250$), $T = 90$ weeks, $R_{k,t} = U_{k,t} = 1, \forall t$, and $s_{i,1} = 1$. In instance E_1, we assess solutions with C set to 0, 0.1, 1, 10 and 100. $C = 0$ corresponds to zero cost of switching between projects, $C = 1$ corresponds to a small cost of switching, and $C = 100$ corresponds to a large cost of switching. Table 5.19 summarizes the number of switches made by each engineer over all projects. In instance E_1, when the cost of switching is zero, there are a total of 4108 switches without any impact to the overall schedule makespan. When the cost of switching is large ($C = 100$), there are zero switches but the overall schedule makespan increases by four weeks. When the cost of switching is small ($C = 1$), the total number of switches reduces from 4108 to 2180, but the makespan increases by one week and the total cost increases from 0 to 15050. Thus, we observe sensible behavior of the tradeoff between total number of switches (stable assignment) and overall schedule makespan. Instances E_2 and E_3 further show the tradeoff in total number of switches and overall schedule makespan, using different values of C for the same number of projects and engineers. Instance E_4 demonstrates scalability as the number of projects is increased to 30, and the number of engineers is increased to 250. In all instances, we reduce the total number of switches made by engineers by increasing C; however, the total cost increases as schedules of projects are pushed out by one to seven weeks.[94]

We also solve a variant of the engineer allocation problem by adding constraints that upper-bound the number of switches of each engineer during the overall schedule makespan. We use the same objective function as the SCM problem with additional constraints. We validate our solver for this variant using instance E_4 from above, and by varying the upper bound on the number of switches allowed per engineer as $\{+\infty, 30, 20, 10, 5, 1\}$. When the upper bound is $+\infty$, the total cost is zero and matches the total cost when $C = 0$ for instance E_4 in Table 5.19. When the upper bound is 30, the total cost is 3830; when the upper bound is 20, the total cost is 33150; when the upper bound is 10, the total cost is 97150; when the upper bound is 5, the total cost is 116650; and when the upper bound is 1, the total cost is 226250.

[94] Instance E_1 has $\sim$46K variables, $\sim$350K constraints and a runtime of around three minutes. Instance E_2 has $\sim$150K variables, $\sim$1.1M constraints and a runtime of around 28 minutes. Instance E_3 has $\sim$450K variables, $\sim$3.2M constraints and a runtime of around 50 minutes. Instance E_4 has $\sim$680K variables, $\sim$5.4M constraints and a runtime of around 2.3 hours.

Table 5.19: Total cost, number of switches, μ and σ of switches over all engineers, and overall schedule makespan impact.

Testcase	N	#Engineers	Switching Cost C	Total #Switches	μ (#Switches)	σ (#Switches)	Total Cost	Δ Overall Makespan
E_1	5	100	0	4108	39.8	7.33	0	0 weeks
			0.1	3804	38.8	7.20	410.4	+1 weeks
			1	2180	21.5	6.36	1920	+1 weeks
			10	1280	12.4	4.28	12840	+1 weeks
			100	0	0	0	15050	+4 weeks
E_2	10	150	0	5690	37.5	5.25	0	0 weeks
			0.1	5520	36.6	5.20	572	+1 weeks
			1	2777	18.5	5.06	3207	+2 weeks
			10	1550	10.6	3.4	16360	+2 weeks
			100	0	0	0	22680	+4 weeks
E_3	30	150	0	5910	39.0	5.36	0	0 weeks
			0.1	5760	38.2	5.33	606	+1 weeks
			1	1965	12.9	4.22	3475	+2 weeks
			10	1342	8.8	2.75	17940	+3 weeks
			100	0	0	0	28500	+4 weeks
E_4	30	250	0	9870	39.2	6.05	0	0 weeks
			0.1	8920	35.6	5.20	1112	+1 weeks
			1	5680	22.8	5.59	30520	+3 weeks
			10	2870	11.2	3.65	72950	+4 weeks
			100	0	0	0	251280	+7 weeks

5.2.3 Conclusions

The lack of tools for management of semiconductor design resources (servers, tool licenses, engineering headcount) can impact a company's bottom line by many millions of dollars per year. In this work, we capture multi-project, multi-resource constrained project scheduling as SCM and RCM MILP formulations that are readily solvable using commercial engines such as *CPLEX*. The MILP solutions provide *optimal* scheduling and allocation solutions for complex multi-tapeout management scenarios in a large design center. Aspects of our formulation that are unique to semiconductor design, and that take our work beyond the earlier RCPSP formulation of Kolisch et al. [142] [144] [143], include multiple resource pools and co-constraints between resources of different types. We demonstrate the flexibility and value of our optimization in three scenarios taken from Company X's design center's recent history. (1) We find an optimal schedule for three concurrent tapeouts when a late-breaking RTL change hits one of the projects, and save 1.4 work-weeks of schedule compared to the solution deployed by the company. This level of saving corresponds to 2.7% of annual labor and infrastructure costs, and enhances market competitiveness. (2) We find an optimal schedule for 20+ projects, subject to datacenter

capacity limits and a tethering constraint with respect to original forecast resource allocations. Our solution shows that a slight relaxation of the tethering constraint would allow committed projects to proceed within resource limits. Our solution meets the schedule with no additional servers. By contrast, in the absence of decision support tools, the company's solution entailed the purchase of hundreds of additional servers. (3) We find an optimal allocation of human resources for four projects and save up to 37% of a particular resource type relative to the solution adopted by the company. In a non-U.S. location, this single resource type reduction would imply a $\sim$\$5M savings for the company within a half-year project scheduling makespan. We also provide "what-if" analyses capabilities with our solver, and demonstrate sensitivity analyses (schedule benefits of incremental resources) and scalability of our solution approach. Since we introduce new concepts such as conditionally-shared, segregated and fully-shared resource types along with and resource co-constraints, we are unable to compare against any previously existing MILP formulations of such problems.

Our work is applicable across multiple stages of project and capacity planning processes. For example, it can be used (i) as part of a fiscal planning process to comprehend the overall resource requirements of a site or a computing cluster; (ii) by program management as a what-if tool so that infrastructure can join engineering headcount as a factor in scheduling decisions; and (iii) by "Engineering Compute" operations teams to understand the impact of product roadmap or schedule changes on datacenter and EDA licensing infrastructure, so that corrective actions may be taken in a proactive and principled manner. By providing a foundation for improved engineering resource allocation to maintain high overall resource utilizations and low schedule latencies, we enable design organizations to improve design throughput and efficiency with given resources. Ultimately, this helps to continue the scaling of design cost efficiencies that are so vital to the IC industry.

Our formulations can be improved in a number of directions which are the subject of ongoing investigation. For instance, it will be helpful to be able to automatically determine threshold values of inputs (e.g., the schedule length T or the tethering constraint δ) at which feasible solutions exist. Scheduling decisions should also comprehend distributions of job sizes or job complexities (which can vary per block and according to the state of a project), and automatic change of timestamps when schedule changes occur. Solutions can be further stabilized by adopting iterative optimization approaches. A further open direction is to optimize robustness of scheduling solutions in the face of stochasticity in resources and personnel.

5.2.4 Glossary of Chip Design Flow Terminologies

Table 5.20 maps activities and resources from problem instances in Section 5.2.2 and Section 5.2.2 to chip design flow terminologies.

Table 5.20: Glossary for the schedule modification use case (Section 5.2.2) and (human) resource allocation use case (Section 5.2.2).

Schedule modification use case (Section 5.2.2)		**Human resource allocation use case (Section 5.2.2)**	
Activity / Resource	Chip Design Flow Mapping	Activity / Resource	Chip Design Flow Mapping
A1	Placement	A12	Block-Level Design (BLD)
A2	Routing	A13	Full-Chip Design (FCD)
A3	Search and Repair	A14	Block-Level Verification (BLV)
A4	Extraction	A15	Full-Chip Verification (FCV)
A5	Static Timing Analysis (STA)	A16	Block-Level Physical Design (BLPD)
A6	Functional ECO	A17	Full-Chip Physical Design (FCPD)
A7	Extraction	A18	Gate-Level Simulation (GLS)
A8	STA (per corner)	A19	Emulation (EMU)
A9	Timing ECO	HR1	Design Resources
A10	Extraction	HR2	Design Verification Resources
A11	STA (per corner)	HR3	PD Resources
L1	P&R License	HR4	EMU Resources
L2	RCX License	–	–
L3	STA License	–	–

5.3 Optimal Reliability-Constrained Overdrive Frequency Selection in Multi-core Systems

Modern systems with multi-core processors typically operate at multiple operating modes, e.g., nominal and overdrive (or, "turbo") [303]. Applications running on multi-core systems have different requirements for the number of cores used at any given moment, as well as for corresponding operating modes. For example, in a system with eight cores, applications A and B may have very different usage of these cores over time. Figure 5.19(a) conceptually illustrates the time-varying usage of cores by applications A and B. Each application uses at most four cores simultaneously over the course of its execution. When memory and I/O resources are not a bottleneck, *the operating system scheduler packs tasks using some or all of the available processing cores in the multi-core system.* Figure 5.19(b) illustrates how the scheduler might pack executions of A and B across eight cores (cf. Packed A, B shown in green color in Figure 5.19(b)). A

key observation is that when all of the cores are not simultaneously active, task scheduling on a subset of the available cores can be adjusted so that the cores wear out in a balanced manner and meet lifetime as well as performance requirements.

Mean time to failure (MTTF) is a measure of the lifetime of a core. MTTF of a core degrades due to reliability mechanisms such as electromigration, time-dependent dielectric breakdown, stress migration, thermal cycling, bias temperature instability, hot carrier injection, etc. [213] [233]. Below, we use the following terminology.

- *power-on-hours* (POH) denotes the *effective* number of lifetime hours consumed. POH is a measure of a given core's lifetime degradation, and differs from the total number of hours for which the core operates, due to operating conditions, e.g., frequency and temperature;
- *nominal temperature* is the temperature at which the MTTF degradation of a core is the same as the number of hours it is active;
- *nominal frequency* is the frequency at which the temperature of a core attains its nominal value;
- *overdrive frequency* is the frequency due to overclocking the cores;[95]
- *acceleration factor* (AF) denotes the increased MTTF degradation due to operating at higher temperatures [305] [347]. AF is the ratio of original MTTF (at nominal temperature) to actual MTTF due to operating at a higher than nominal temperature.[96]

To meet performance and throughput requirements, cores operate at overdrive frequencies, and hence at higher than nominal temperatures. Thus, overdrive modes can result in cores' actual MTTFs becoming significantly smaller than original MTTFs. MTTF degradation can lead to two challenges. (1) All the cores in a multi-core system can fail even before all assigned tasks are completed. A common strategy is to dynamically adapt overdrive frequencies so that all tasks are completed within the system's lifetime [131] [213]. (2) To reduce MTTF degradation, overdrive frequencies must be reduced and can violate a minimum "acceptable performance" requirement for tasks. We use Black's Equation [24] to calculate the MTTF due to electromigration. We consider a core to be reliable as long as the POH of the core is $\leq$ MTTF.

As reviewed in Section 2.3.3, prior works on task scheduling for multi-core systems are taxonomized as reliability-constrained (RC) or non-reliability-constrained (NRC). RC task

[95] At overdrive frequencies, the core's temperature is higher than nominal.

[96] $AF = 1$ when a core is active at its nominal temperature, and $AF > 1$ at higher than nominal temperatures. This captures the well-known acceleration of wearout with higher operating temperature [305].

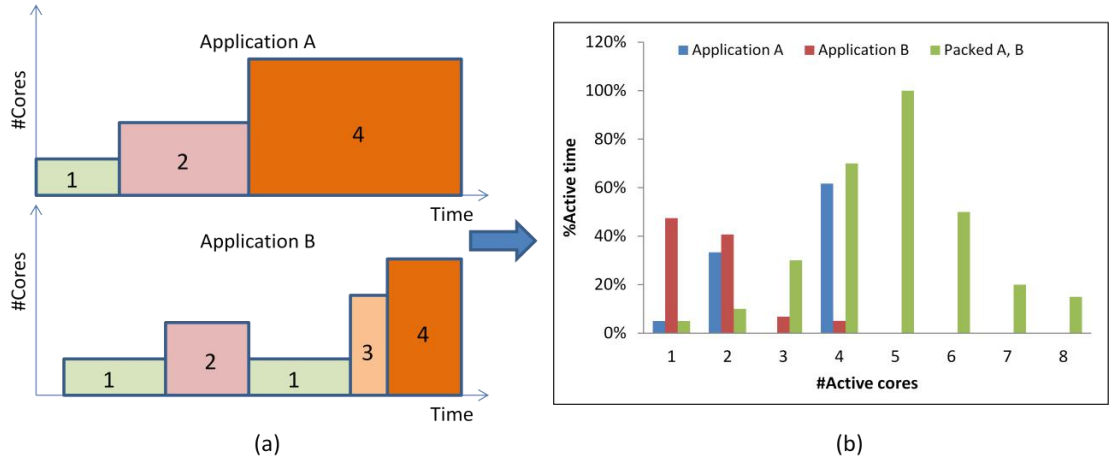


Figure 5.19: Core usage profiles of (a) individual applications A and B, and (b) after the scheduler packs execution using eight cores. Note that B starts after A. The numbers in the colored boxes refer to the number of cores active.

scheduling policies can be further classified as those that make system "lifetime guarantees" (LG) and those that make "no lifetime guarantees" (NLG). Existing RC-LG policies apply (1) dynamic power management (DPM), (2) dynamic thermal management (DTM), or (3) dynamic reliability management (DRM). Such works are "performance-guaranteeing" (PG) if they guarantee lower bounds on "acceptable performance". Table 5.21 classifies existing works and our work according to the foregoing taxonomy.

Previous NRC, RC-NLG and RC-LG policies can be suboptimal. We demonstrate the suboptimality of these existing policies using a simple counterexample. We consider a system with four cores and MTTF of seven years (= 61320h) for each core. The nominal frequency and temperature are respectively 1.5GHz and 358K, the maximum frequency is 3.0GHz, and the maximum temperature is 398K. We assume that the scheduler assigns tasks for each (m (= the number of active cores), nominal execution time, overdrive execution time) 3-tuple as follows: (1, 1000h, 3000h), (2, 2000h, 5000h), (3, 3000h, 8000h), and (4, 2000h, 5000h). Last, we assume that overdrive-mode tasks require a *minimum overdrive frequency* or "acceptable performance" of 1.8GHz.

Using *HotSpot v5.01* [227] as detailed in Section 5.3.2 below, for the above instance optimal (discretized) overdrive frequencies can be found using exhaustive search for each m respectively as 2.85GHz, 2.3GHz, 1.8GHz and 1.8GHz.[97] However, NRC and RC-NLG policies will operate always at 3.0GHz and 398K, inducing an acceleration factor AF = 9.77 relative to

[97]We describe in Section 5.3.1 the use of *HotSpot v5.01* [227] to simulate temperatures (from which wearout acceleration factors (AFs) are derived) for different frequencies and combinations of active cores.

Table 5.21: Classification of existing works on multi-core task scheduling and our work.

Work	NRC	RC				
		NLG	LG			
			DPM	DTM	DRM	PG
Reiss et al. [208]	✓					
Karpuzcu et al. [132]		✓				
Mihic et al. [174]			✓			
Rosing et al. [213]			✓			
Rong et al. [212]			✓			
Coskun et al. [53] and [54]				✓		
Srinivasan et al. [233]					✓	
Karl et al. [131]					✓	
Our work					✓	✓

the nominal operating temperature of 358K. Assuming execution of tasks are balanced across the four cores, Figures 5.20 and 5.21 respectively illustrate suboptimalities of NRC and RC-LG policies for a system with four cores, where each core has an initial lifetime of seven years (61320h) and the tasks listed in the previous paragraph are as assigned by the scheduler. The figures show how time progresses along with execution of nominal and overdrive tasks, starting with $m = 1$ and followed by $m = 2,3$ and $m = 4$ for Figure 5.21. For example, for $m = 1$ in both figures, the duration of nominal tasks $E_{nom} = 1000$h, each core executes 250h at a nominal frequency $f_{nom} = 1.5$GHz, and the corresponding AF is 1.0, hence the value of power-on-hours, POH, is 250h$\times$1.0 = 250h for each core. All POH values are shown with a negative sign to indicate effective lifetime consumed. Further, for $m = 1$, the duration of overdrive tasks $E_{OD} =$ 3000h, each core executes 750h at an overdrive frequency $f_{OD} = 3.0$GHz, and the corresponding AF is 9.77, hence the POH is 750h$\times$9.77 = 7372.5h.

With NRC policies, Figure 5.20 shows that the effective lifetime of each core at the end of $m = 3$ nominal tasks is 24947.5h ("Lifetime remaining" in the figure). This is because each core consumes 1220h + 24425h after executing 1000h in nominal and 2500h in overdrive modes for $m = 2$, and 3150h after executing 2250h in nominal mode for $m = 3$. For balanced execution of $m = 3$ overdrive tasks, each core requires 6000h of execution; this requires the lifetime of each

core to be 6000h$\times$9.77 = 58620h, which exceeds the effective lifetime of 24947.5h. (Moreover, this is without resourcing any tasks for $m = 4$.) This simple example shows that *existing NRC policies are not optimal, in that they cannot guarantee that cores will complete all tasks before failing*. The conclusion from our counterexample: Operating at the maximum frequency for all overdrive tasks is not the optimal strategy, as it ignores lifetimes of cores.

With RC-LG policies, Figure 5.21 shows that the POH of each core is 250h + 7327.5h + 1220h + 22100h after executing 250h in nominal mode and 750h in overdrive mode for $m = 1$, and 1000h in nominal mode and 2500h in overdrive mode for $m = 2$. Cores can be run at the maximum value of 3.0GHz for $m = 1$, but cannot maintain this frequency for $m = 2,3,4$ if the required tasks are to be completed while maintaining the system's lifetime ("Lifetime remaining" is 1422.5h ≥ 0 in the figure). Executing the frequency assignment approach of RC-LG policies, we experimentally determine that overdrive frequency for $m = 3,4$ can be at most 1.6GHz, which is less than the minimum required overdrive frequency of 1.8GHz as illustrated in Figure 5.21. The positive value of "Lifetime remaining" cannot be utilized to increase any of the overdrive frequencies further, as this violates lifetime requirements. Thus, our simple example shows that *existing RC-LG policies are not optimal, in that they cannot guarantee lower bounds on "acceptable performance"*. The conclusion from our counterexample: executing overdrive tasks at the maximum frequency and decreasing the frequency to meet lifetime is not the optimal strategy, as it does not consider duration of all overdrive tasks across all combinations of active cores.

		Core 1	Core 2	Core 3	Core 4	
Initial Lifetime		61320	61320	61320	61320	
m = 1	f_{nom} = 1.5GHz	-250	-250	-250	-250	E_{nom} = 1000; AF = 1.00
	f_{OD} = 3.0GHz	-7327.5	-7327.5	-7327.5	-7327.5	E_{OD} = 3000; AF = 9.77
m = 2	f_{nom} = 1.5GHz	-1220	-1220	-1220	-1220	E_{nom} = 2000; AF = 1.22
	f_{OD} = 3.0GHz	-24425	-24425	-24425	-24425	E_{OD} = 5000; AF = 9.77
m = 3	f_{nom} = 1.5GHz	-3150	-3150	-3150	-3150	E_{nom} = 3000; AF = 1.40
Lifetime remaining		24947.5	24947.5	24947.5	24947.5	
m = 3	f_{OD} = 3.0GHz	-58620	-58620	-58620	-58620	E_{OD} = 8000; AF = 9.77

time

Figure 5.20: Suboptimality of NRC policies. E_{nom} and E_{OD} respectively indicate the nominal and overdrive execution times. Lifetime of all cores are completely used up by the end of nominal mode execution in $m = 3$. Thus, tasks requiring $m = 3$ overdrive mode execution, and all tasks requiring $m = 4$, cannot be completed. All times are in hours.

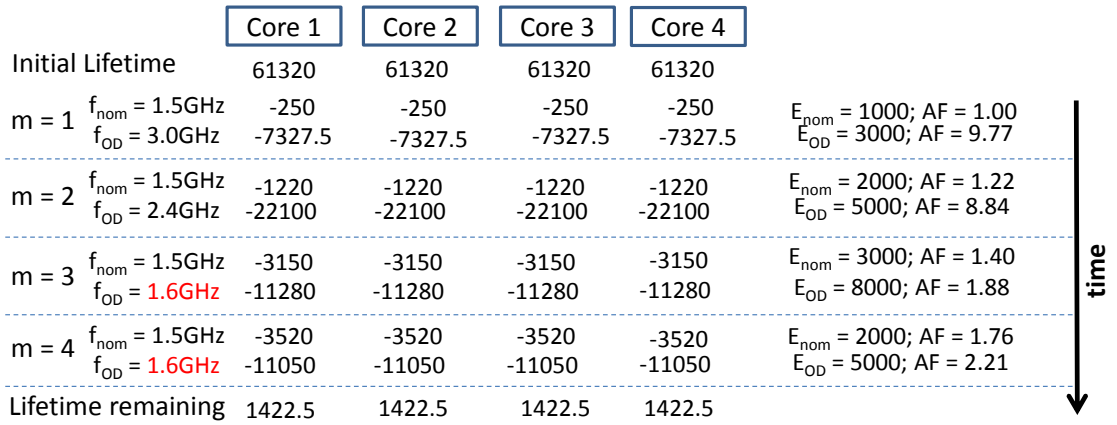


Figure 5.21: Suboptimality of RC-LG policies. E_{nom} and E_{OD} respectively indicate the nominal and overdrive execution times. Tasks requiring $m = 3$ and $m = 4$ overdrive mode execution must operate at an overdrive frequency of 1.6GHz instead of the required 1.8GHz. All times are in hours.

In this section, we formulate and solve a new *maximum-value, reliability-constrained overdrive frequencies* (MVRCOF) optimization problem that, unlike existing works, *guarantees* prescribed levels of "acceptable performance" and "acceptable throughput". To the best of our knowledge, ours is the first approach that guarantees minimum performance and throughput requirements under reliability constraints. Our MVRCOF formulation maximizes the value (or, the advantage) of operating active cores at overdrive frequencies. The optimization is performed *offline* and is subject to four types of constraints: a lower bound on lifetime of each core, completion of all tasks within the system's lifetime, and upper bounds on instantaneous power and temperature. We develop a solver for the MVRCOF problem that determines the duration each combination should remain active so that all cores have balanced wearout. If no feasible solution exists, the scheduler may be notified to modify the task profile. To find the optimal solution, we perform exhaustive search over all overdrive frequencies within upper and lower bounds and all combinations of simultaneously active cores.[98] To make our flow scalable and efficient, we perform a *one-time characterization* of temperature and wearout for each combination of active cores, at each overdrive frequency. To our understanding, the MVRCOF solution is suitable for task migration in datacenters and other multi-core systems [169].

[98] We implicitly consider that all possible overdrive frequencies are feasible for a given block implementation. The relationship between feasible range of overdrive frequencies and the area, mix of different-Vt cells in implementation, etc. of a block is an open direction for future work.

Our contributions are the following.

- We propose a new MVRCOF formulation to maximize the value of operating multiple cores in overdrive frequencies under reliability constraints and given weights (relative importance) for overdrive and nominal frequencies in different modes.
- Our formulation guarantees satisfaction of prescribed lower bound constraints on both "acceptable performance" and "acceptable throughput" across all combinations ($C(N,m)$) of active cores. (m is the number of active cores, $m = 1,2,...,N$). To the best of our knowledge, we are the first to make minimum performance and throughput guarantees under electromigration reliability constraints.
- We determine optimal overdrive frequencies for each m out of N available cores. These frequencies can be maintained throughout the lifetime of the multi-core system.[99]
- We develop both exhaustive (discretized) search for the optimal solution, as well as an approximate heuristic. In practice, our heuristic is within 3.3% of optimal across multiple testcases and converges up to $\sim 10\times$ faster than the exhaustive search.[100]
- Our approaches determine the execution times of each combination of active cores; this can be used by OS schedulers to assign tasks to cores while maintaining required MTTF for all cores. (As we discuss below, this implies balanced wearout of cores.) Although we validate our solutions using four, six and eight cores, our method can be applied to systems with more than eight cores.
- We empirically demonstrate that our optimal solutions improve the objective function value by between 2.2% and 17.4% when compared to the existing reliability-aware task scheduling approaches of [53] and [213].

5.3.1 Problem Formulation

We now formulate our MVRCOF optimization problem to maximize the value[101] of a set of overdrive frequencies, such that no core wears out before its prescribed lifetime requirement. We assume that the scheduler packs tasks from multiple applications and provides a final

[99] This implicitly assumes that each core in the system can adapt (e.g., by supply voltage scaling) to aging in order to maintain a given target (nominal or overdrive) operating frequency.

[100] Our method is based on offline simulations. Comparison of our results with measured data remains a direction for future work.

[101] By "value", we refer to the advantage of operating at overdrive frequencies. We maximize the advantage by maximizing the overdrive frequencies.

operating schedule as conceptually illustrated in Figure 5.22. Further, we do not consider the cost of task migration.

There are two kinds of inputs: (1) system description, and (2) task description. Table 5.22 lists the parameters used in our formulation and solution. Figure 5.23 shows the inputs and outputs of the MVRCOF problem graphically. We assume that the values for the system description parameters are given. The operating system scheduler provides values for the task description parameters depending on application demands for performance and the number of cores used. For example, applications that benefit from frequency overdrive are accounted for in $E_{OD,m}$ and $w_{OD,m}$ parameters, whereas applications that do not benefit from frequency overdrive are accounted for in $E_{nom,m}$, $f_{nom,m}$ and $w_{nom,m}$ parameters.

We define "acceptable performance" as $1.3 \times f_{nom,m}$, based on [303]; this does not compromise the generality of our conclusions. Further, we define "acceptable throughput" as the ability to complete all tasks within the multi-core system's given lifetime.

Given the above-described inputs, the problem to maximize the value of a set of overdrive frequencies is formally expressed as

$$\text{maximize} \sum_{m=1}^{N} (w_{OD,m} \cdot f_{OD,m} \cdot E_{OD,m} + w_{nom,m} \cdot f_{nom,m} \cdot E_{nom,m})$$

subject to

$$\forall m, 1.3 \times f_{nom,m} \leq f_{OD,m} \leq f_{max} \tag{5.55}$$

$$\forall i, MTTF_i \leq MTTF \tag{5.56}$$

$$\forall i, P_i \leq P_{max} \tag{5.57}$$

$$\forall m, T_m \leq T_{max} \tag{5.58}$$

where

- Constraint (5.55) ensures lower bounds on "acceptable overdrive performance" that are at least 30% greater than $f_{nom,m}$, and also restricts $f_{OD,m}$ values to be less than the maximum frequency of the system.

Table 5.22: Parameters used in MVRCOF problem formulation and solution.

Parameter	Description	Type
N	#symmetric cores indexed by $i = 1, 2, \cdots, N$	System
P_{max}	maximum power consumption of any core	System
f_{max}	maximum frequency of any core	System
T_{nom}	core temperature at nominal frequency	System
T_{max}	maximum die temperature	System
$MTTF$	lifetime of each core	System
E_a	metal activation energy (0.7eV [124])	Physical
k	Boltzmann's constant (8.62×10^{-5}eV/K)	Physical
m	#simultaneously running, or *active*, cores, $1 \leq m \leq N$ (at the same nominal or overdrive frequency)	Task
l	operating mode of any core ($\{$nom, OD$\}$)	Task
$f_{nom,m}$	nominal frequency of m cores	Task
$w_{l,m}$	weights in objective function of achieved frequencies, $w_{l,m} \geq 0, \forall l, m$	Task
$E_{l,m}$	execution time in operating mode l with m cores	Task
P_i	power of the i^{th} core at any time	Variable
T_m	temperature of the die with m active cores at any time	Variable
$t_{j,m,l}$	execution times for the j^{th} combination of m cores	Variable
$b_{i,j,m}$	binary variable (1 when core i is present in the j^{th} combination of m cores, 0 otherwise)	Variable
$MTTF_i$	effective lifetime of the i^{th} core after it has been active	Variable
$POH_{l,i}$	POH of the i^{th} core in operating mode l	Variable
$AF_{i,j,m,l}$	AF of the i^{th} core in the j^{th} combination of m cores in operating mode l	Variable
$T_{i,j,m,l}$	temperature of the i^{th} core in the j^{th} combination of m cores in operating mode l	Variable
Δ_{fOD}	discrete amount by which overdrive frequency is increased	Variable
$f_{OD,m}$	overdrive frequency for m cores	Output
$v_{j,m,l}$	% total execution time for the j^{th} combination of m cores in operating mode l	Output
$u_{i,l}$	% of lifetime during which the i^{th} core is active in operating mode l	Output

- Constraint (5.56) ensures "acceptable throughput", i.e., tasks are completed within the system's lifetime.

- Constraints (5.57) and (5.58) ensure that the power of each core, P_i, and the temperature of the die with m active cores, T_m, are within maximum power and temperature upper bounds.[102] $MTTF_i$ is calculated using Equations (5.59) – (5.61) [305].

We use the execution times in the objective function to determine the duration over which cores in mode m execute at $f_{OD,m}$.

$$AF_{i,j,m,l} = exp\left(\frac{E_a}{k} \cdot \left[\frac{1}{T_{nom}} - \frac{1}{T_{i,j,m,l}}\right]\right) \tag{5.59}$$

$$POH_{l,i} = \sum_{m=1}^{N} \sum_{j=1}^{C(N,m)} t_{j,m,l} \cdot b_{i,j,m} \cdot AF_{i,j,m,OD} \tag{5.60}$$

$$MTTF_i = \sum_{l=\{nom,OD\}} POH_{l,i} \tag{5.61}$$

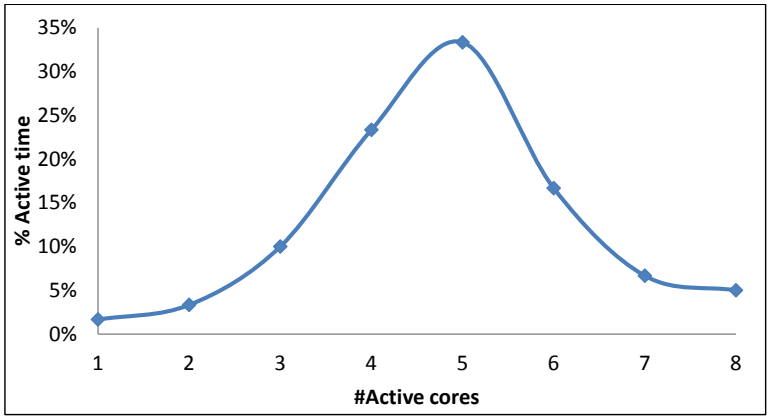


Figure 5.22: Core usage profile from scheduler after packing tasks from multiple applications. The task profile represents typical datacenter workload from [23].

Optimal (Discretized) Solution Flow

We now present our flow to solve the MVRCOF optimization problem. We also present our heuristic flow and a baseline flow to compare reliability-constrained with lifetime guarantee (RC-LG) policies. To work around potential nonlinear constraints, we perform exhaustive search of $f_{OD,m}$ across all m by increasing values of $f_{OD,m}$ from $1.3 \times f_{nom,m}$ to f_{max} by Δ_{fOD}. We perform one-time characterization (~9.5h on an Intel Xeon E5-2640 2.5GHz system) of temper-

[102] We obtain P_i from post-layout power simulation of a core at $f_{nom,m}$ or $f_{OD,m}$. T_m is the die temperature obtained from *HotSpot v5.01* [227] simulations.

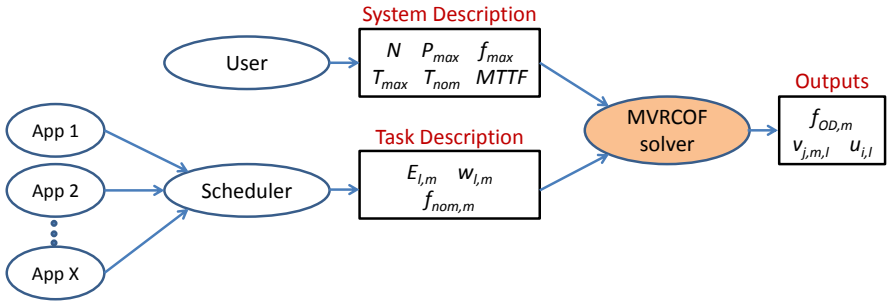


Figure 5.23: Graphical representation of inputs and outputs of the MVRCOF problem.

ature and AF for all discretized values of $f_{OD,m}$ for each combination of m out of N cores and generate a lookup table (LUT)[103] as follows.

1. Perform post-layout gate-level power simulation of a single core. We set the switching activity factor at primary inputs in the nominal mode as 0.11 [304] and in the overdrive mode as 0.3 [221]; We also set the clock period as the inverse of the overdrive frequency in the SDC file [21].

2. Perform temperature simulation with power using *HotSpot* [227].[104]

3. Create a LUT entry for the i^{th} core if it is present in the j^{th} combination of m cores as ($f_{OD,m}$, temperature, AF). For each core and for each $f_{OD,m}$, there are at most $N \times C(N,m)$ entries. Figure 5.24 illustrates an example of a subset of a LUT for a system with three cores.

4. Increase $f_{OD,m}$ by Δ_{fOD} as long as $f_{OD,m} + \Delta_{fOD} \leq f_{max}$. Go to Step (1).

Algorithm 4 shows details of our flow to determine the optimal values of $f_{OD,m}$. $f(.)$ in the algorithm indicates that the output value is a function of the input parameters $(.)$. Lines 1–18 initialize MTTF of all cores, overdrive frequencies across all m, the variable $b_{i,j,m}$ for each core, and the variables that store the optimal value of the objective function and the overdrive frequencies. Lines 19–46 contain the loops for the exhaustive search. Lines 22–34 comprise the innermost loop which, given an overdrive frequency, determines if all cores in all combinations

[103]The reader will notice that we are assuming that temperature is restored to its nominal value before task execution begins on each successive combination of active cores. We believe this assumption is reasonable as long as task assignments have relatively long durations, and given that, in a day, cores are utilized ∼60% of the time (i.e., system is idle for ∼9h) [208]. Our experiments indicate that the time for temperature to drop from 125^{o}C to 85^{o}C is ∼10s, and this is consistent with the results of [279]. Achieving a solution that is "history-aware" remains a direction for future work.

[104]Thermal parameters for *HotSpot* [227] simulations are calibrated with Intel Xeon processors [54].

Core	(m, j)	$f_{OD,m}$ (GHz)	Temp (K)	AF
1	(1,1)	3.0	398	9.77
		2.95	392.5	7.34
		2.90	396.9	5.34
		• • •	• • •	• •
		1.80	362.0	1.29
•	• • •	• • • •	• • •	• •
2	(2,1)	3.0	398	9.77
		2.95	393.1	7.58
		2.90	388.4	5.90
		• • •	• • •	• •
		1.80	367.6	1.82
•	• • •	• • • •	• • •	• •
3	(3,1)	3.0	398	9.77
		2.95	397.5	9.53
		2.90	396.9	9.24
		• • •	• • •	• •
		1.80	377.4	3.42

Figure 5.24: Example of a subset of a lookup table (LUT) for a three-core system.

can complete execution within the prescribed lifetime. If a core is present in a combination, Line 25 obtains its AF from the LUT, and Line 26 executes the following linear program (LP), which calculates $t_{j,m,nom}$ and $t_{j,m,OD}$ to balance wearout across all cores. Formally, the LP is expressed as

maximize c

subject to

$$\forall i, c \leq \sum_{l=\{nom,OD\}} POH_{l,i} \tag{5.62}$$

$$\forall i, \sum_{l=\{nom,OD\}} POH_{l,i} \leq MTTF \tag{5.63}$$

$$\sum_{j=1,\ l\in\{nom,OD\}}^{C(N,m)} t_{j,m,l} = E_{l,m} \tag{5.64}$$

where

- Constraint (5.62) ensures c is the minimum POH across all cores.
- Constraint (5.63) ensures that all tasks are completed within the multi-core system's lifetime.
- Constraint (5.64) ensures that the sum of nominal and overdrive execution times across all combinations of m active cores meet the respective required execution times. $POH_{l,i}$ is obtained from Equation (5.60).

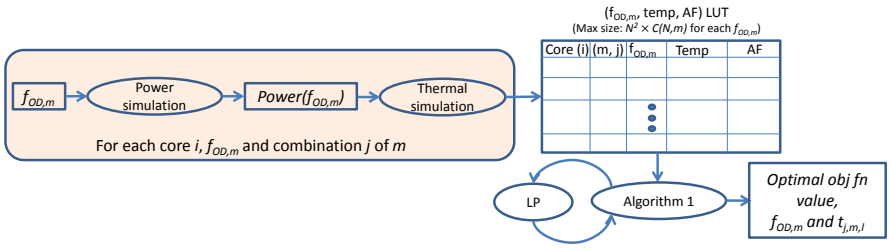


Figure 5.25: Flow to obtain optimum solution. Algorithm 4 uses the lookup table (LUT) and repeatedly invokes the LP solver.

If an optimal solution exists for the LP, Line 27 calculates POH for the core using Equation (5.61) and Line 28 updates the core's effective MTTF, $MTTF_i$, based on $t_{j,m,l}$ and $E_{l,m}$ values. If all cores can complete execution within their lifetimes, Line 36 calculates the value of the objective function. Lines 38–41 store the overdrive frequencies and largest value of the objective function obtained up through the iteration $iter$. (We compare $iter$ across our testcases in Section 5.3.2.) Lines 42–43 increment $f_{OD,m}$ by Δ_{fOD}. When all values of frequencies across all m have been tried, Lines 47–51 output the optimal solution or report infeasibility if no feasible solution exists.

Figure 5.25 shows our complete flow from generating the LUT to using Algorithm 4 and the LP to achieving optimal values of the objective function, $f_{OD,m}$ and $t_{j,m,l}$. We can now determine the optimal values of $v_{j,m,l}$ and $u_{i,l}$ from the values of $t_{j,m,l}$. Exhaustive search across all discretized values of $f_{OD,m}$ across all cores present in the j^{th} combinations of m achieves a discretized optimal solution. The time complexity of Algorithm 4 is $O(N^2 \cdot C(N,m) \cdot f_{steps})$, where $f_{steps} = (f_{max} - 1.3 \times f_{nom,m}) / \Delta_{fOD}$. By construction, our optimal flow guarantees "acceptable performance" because we search for optimal values of $f_{OD,m}$ that are at least 30% above the nominal frequency. The optimal flow also guarantees "acceptable throughput" because our LP always guarantees balanced wearout and Lines 29–31 of Algorithm 4 do not allow cores to operate at frequencies that cannot guarantee task completion within a core's effective lifetime.

Heuristic flow. We also develop a heuristic flow to solve the optimization by maximizing overdrive frequencies for each m in the order of decreasing $w_{OD,m} \times E_{OD,m}$. This is because maximizing $f_{OD,m}$ for the largest $w_{OD,m} \times E_{OD,m}$ causes large improvement in the objective function value. The flow is similar to the exhaustive search; however, rather than explore all modes in their numerical order, we start with the mode that has the largest product of $w_{OD,m} \times E_{OD,m}$ and obtain the maximum $f_{OD,m}$. The flow uses Lines 20–45 of Algorithm 4 to determine the maxi-

Algorithm 4 Determination of optimal $f_{OD,m}$

Input: N, f_{max}, $f_{nom,m}$, $E_{nom,m}$, $E_{OD,m}$, $w_{nom,m}$, $w_{OD,m}$, ($f_{OD,m}$, temp) LUT, $MTTF$

1: **for all** $i = 1,2,...,N$ **do**
2: $MTTF_i \leftarrow MTTF$
3: **end for**
4: **for all** $m = 1,2,...,N$ **do**
5: $f_{OD,m} \leftarrow 1.3 \times f_{nom,m}$
6: **end for**
7: **for all** $m = 1,2,...,N$ **do**
8: **for all** $j = 1,2,...,C(N,m)$ **do**
9: **for all** $i = 1,2,...,N$ **do**
10: **if** core $i \in$ combination j **then**
11: $b_{i,j,m} \leftarrow 1$
12: **else**
13: $b_{i,j,m} \leftarrow 0$
14: **end if**
15: **end for**
16: **end for**
17: **end for**
18: $iter \leftarrow 1$; $Bestval \leftarrow 0$; $Bestfod[] \leftarrow 0$
19: **for all** $m = 1,2,...,N$ **do**
20: **while** $f_{OD,m} \leq f_{max}$ **do**
21: $Val_{iter} \leftarrow 0$; $term \leftarrow 0$
22: **for all** $j = 1,2,...,C(N,m)$ **do**
23: **for all** $i = 1,2,...,N$ **do**
24: **if** $b_{i,j,m} > 0$ **then**
25: Get AF: $AF_{i,j,m,OD} \leftarrow f(f_{OD,m}, LUT, i, j, m)$
26: Solve LP: $t_{j,m,l} \leftarrow f(AF_{i,j,m,l}, MTTF)$
27: Calculate POH: $POH_i \leftarrow f(t_{j,m,l}, E_{nom,m}, E_{OD,m})$
28: Update MTTF: $MTTF_i \leftarrow MTTF_i - POH_i$
29: **if** $MTTF_i \leq 0$ **then**
30: $term \leftarrow 1$
31: **end if**
32: **end if**
33: **end for**
34: **end for**
35: **if** $term > 0$ **then**
36: Calculate objective function value:
37: $Val_{iter} \leftarrow w_{nom,m} \cdot f_{nom,m} \cdot E_{nom,m} + w_{OD,m} \cdot f_{OD,m} \cdot E_{OD,m}$
38: **if** $Val_{iter} > Bestval$ **then**
39: $Bestval \leftarrow Val_{iter}$
40: $Bestfod[m] \leftarrow f_{OD,m}$
41: **end if**
42: $f_{OD,m} \leftarrow f_{OD,m} + \Delta_{fOD}$
43: $iter \leftarrow iter + 1$
44: **end if**
45: **end while**
46: **end for**
47: **if** $Bestval > 0$ **then**
48: Output $Bestval$ and $Bestfod[1,2,...,N]$
49: **else**
50: Output no feasible solution exists
51: **end if**

mum $f_{OD,m}$ for each m in decreasing order of $w_{OD,m} \times E_{OD,m}$. We compare the $iter$ value (i.e., the number of iterations) from Line 43 between the optimal and heuristic solutions in Section 5.3.2.

Baseline reliability-constrained with lifetime guarantee (RC-LG) flow. To compare the solutions of our optimal and existing RC-LG policies, we describe a baseline flow in which we use frequency assignment approach in RC-LG policies. RC-LG policies assign the maximum frequency of a combination m as long as power, thermal upper bound and lifetime constraints are met [132] [208] [213]. The frequency is dynamically changed to meet lifetime requirements. The baseline flow is as follows.

- Choose core(s) with the maximum MTTF for execution and whose $MTTF_i \geq$ the required (nominal or overdrive) execution times.
- Find the maximum $f_{OD,m}$ subject to power, thermal upper bound constraints and $MTTF_i$ (when multiple cores are active, use the minimum $MTTF_i, \forall i$).[105]

5.3.2 Experimental Setup and Results

We now describe our testcase generator, and present our results.

Testcase Generation

To construct different testcases, we modify (1) $w_{nom,m}$ and $w_{OD,m}$, (2) $E_{nom,m}$ and $E_{OD,m}$, and (3) $f_{nom,m}$ values. While we make certain choices of parameter values in our experiments, these choices do not compromise the generality of our method and conclusions. Let the ratio $r_m = \frac{w_{nom,m}}{w_{OD,m}}$ be a random variable chosen uniformly in the interval [0.1, 10]. Then, $w_{OD,m} = \frac{1}{1+r_m}$ and $w_{nom,m} = \frac{r_m}{1+r_m}$.[106] For each m, we generate a value of r_m to obtain $w_{nom,m}$ and $w_{OD,m}$.

Figure 5.22 illustrates an example of the scheduler-determined total execution times in an eight-core system when $m = 1, 2, ..., 8$ cores, respectively, are active. To model a similar skewed Gaussian distribution of $E_{tot,m} = E_{nom,m} + E_{OD,m}$ for the random variable m with mean μ, standard deviation σ and probability density function $f(m|\mu,\sigma)$, we assume the following.

- All cores begin execution with the same MTTF $= \{7, 10\}$ years to represent typical server cores [243].
- $\mu = \lfloor \frac{N}{2} \rfloor + 1, \forall N$; μ is an integer as it equals m with the largest $E_{tot,m}$.

[105] We perform exhaustive search for the maximum $f_{OD,m}$, starting from f_{max}. If a frequency can complete current and future tasks within the lifetime of all cores, then we set this as the maximum $f_{OD,m}$. Otherwise, we decrease in 50MHz steps until $f_{OD,m} = f_{nom,m}$.

[106] $0.1 \leq r_m \leq 10$ allows us to obtain extreme values such as $w_{nom,m} = 0.1$ and $w_{OD,m} = 0.9$, and $w_{OD,m} = 0.1$ and $w_{nom,m} = 0.9$.

- $3\sigma = \max\left(N - \left(\lfloor \frac{N}{2} \rfloor + 1\right), \lfloor \frac{N}{2} \rfloor\right) = \lfloor \frac{N}{2} \rfloor$ because the execution times with fewer or more active cores are less than the ones which use approximately half of N [208]. σ does not need to be an integer because we only want to make sure that $|m - \mu| \leq 3\sigma$. Therefore, $\sigma = \frac{1}{3} \cdot \lfloor \frac{N}{2} \rfloor, \forall N$.

- U is a uniformly random number in [0.35, 0.5], denoting the peak core utilization of Amazon EC2 datacenters, following core utilization data in [161]. Then, we have that $E_{nom,\mu} + E_{OD,\mu} = MTTF \times U$, and $E_{tot,m} = E_{tot,\mu} \times f(m|\mu,\sigma)$ when $m \neq \mu$.

- $r_{e,m}$ is a uniformly random number in [0.1, 0.5] following task priority information in [161], which denotes the ratio of $E_{nom,m}$ to $E_{nom,m} + E_{OD,m}$. Then, we have that $E_{nom,m} = r_{e,m} \times E_{tot,m}$, and $E_{OD,m} = E_{tot,m} - E_{nom,m}$.

- $f_{nom,m}$ takes on values between [1.5, 2.0]GHz in steps of 50MHz so that the maximum frequency is $\leq$ 3.0GHz [282].

Decomposition of Task Trace

We decompose packed tasks from Figure 5.22 to resemble realistic datacenter traces with the following assumptions. (1) For a given m, all cores execute either nominal or overdrive tasks. (2) In a day, nominal tasks run for $\sim$20% (5h) and overdrive tasks run for $\sim$40% (10h) [23] [151] [208]. The cores are idle for the remaining 9h. (3) Tasks are nonpreemptive. (4) Overdrive tasks are scheduled before nominal tasks.

We obtain $E_{nom,m}$ and $E_{OD,m}$ from above and calculate the per-day nominal and overdrive tasks for each m so that they are in the ratio of their total execution times.[107] Therefore, in a day, across all m, the sum of nominal tasks in a day adds up to 5h and the sum of overdrive tasks adds up to 10h. We now generate a trace by sequencing tasks as {overdrive, nominal} for each day such that execution times of all nominal and overdrive tasks respectively add to $E_{nom,m}$ and $E_{OD,m}$. The sequencing gives the notion that overdrive tasks must be completed before nominal tasks. We use these traces to validate our solutions, as we describe next.

Results

To simulate a design to fill 25mm^2, 38mm^2 and 51mm^2 die at $\sim$70% utilization respectively with four, six, and eight vector processor-like cores, we create a floorplan file for *Hotspot*

[107] For example, if $E_{nom,1}$ is 10% of total execution time of nominal tasks across all m, then in a day the nominal task for $m = 1$ is 10% of 5h.

[227] that instantiates each processor core as 72× *jpeg_encoder* [318] cores. To obtain area and power for 72× *jpeg_encoder*, we perform synthesis, place-and-route, and power analysis of a single *jpeg_encoder* core and scale the area and power values by 72. We use 45nm foundry libraries, synthesize the *jpeg_encoder* design using Synopsys *Design Compiler vG-2012.06* [337] and perform place-and-route using Cadence *SOC Encounter vEDI10.1* [287] with clock period set to 0.5ns. The post-layout netlist for a single *jpeg_encoder* core contains ∼38K instances and the area of standard-cells is 0.06mm^2. We then perform power analysis by varying supply voltage from 0.8V to 1.2V in steps of 10mV, varying the frequency (transcribed to clock period in the SDC [21] file) from 1.5GHz to 3.0GHz in steps of 50MHz.[108] We develop a solver using custom *Tcl* scripts to implement Algorithm 4 and the heuristic flow, and solve all our generated LP instances using *lp_solve* [309]. For all m, we set $f_{nom,m} = 1.5$GHz and the minimum $f_{OD,m}$ to be 1.8GHz (i.e., 20% greater than the nominal frequency, following [303]). We set P_{max} to 30W [303] and T_{max} to 398K [305].

We report experimental setup and results with four[109], six and eight cores, and MTTF of each core set to seven years. Table 5.23 shows the $E_{nom,m}$, $E_{OD,m}$, $f_{nom,m}$, $w_{nom,m}$, and $w_{OD,m}$ values for each testcase. We name our testcases as N-testcase# where N indicates the number of cores. Testcase 6-II uses twice the $E_{OD,m}$ values from Testcase 6-I. Testcases 4-I and 4-II use different weights for $m = 1,4$. Our results enable (1) validation of our discretized optimal solutions, and (2) comparison of solutions from optimal, heuristic, and baseline (RC-LG) flows.

To validate our solutions, we confirm the sensibility of how varying Δ_{fOD}, $w_{nom,m}$, and $w_{OD,m}$ affect the value of the objective function. Then, for Testcase 8-I, we present the optimal values of $v_{j,m,l}$ and $u_{i,l}$ which can potentially be used by the operating system scheduler for task migration.[110]

Impact of Δ_{fOD}. Table 5.24 shows the optimal values of $f_{OD,m}$ and the value of our objective function for different values of Δ_{fOD} from our optimal flow. The objective function value varies by less than 0.3% when Δ_{fOD} varies from 200MHz to 50MHz. However, when $\Delta_{fOD} = 50$MHz, the solution requires ∼13× the number of iterations to converge.[111] Smaller values of Δ_{fOD} achieves higher values of the objective function, as expected. Because the MVRCOF solver runs

[108] We assume that for a given block implementation all the possible f_{OD} values are feasible by construction, i.e., the implementation ensures no timing violations when the frequency is set to f_{max}.

[109] The area of four cores with 70% utilization is $(4 \times 72 \times 0.06)/0.7 \approx 25$mm^2.

[110] Testcase 6-II is infeasible because no value of $f_{OD,m}$ with lower bound ("acceptable performance") set to 1.8GHz can complete the tasks within the system's lifetime.

[111] On a Intel Xeon E5-2640 2.5GHz system, each iteration executes roughly in 0.7s.

Table 5.23: Details of testcases. Each row shows execution times and weights for two values of m.

Testcase	m	$E_{nom,m}$ (Kh)	$E_{OD,m}$ (Kh)	$w_{nom,m}$	$w_{OD,m}$	$f_{nom,m}$ (GHz)
4-I	1, 2 3, 4	1, 2 3, 2	3, 5 8, 5	0.5, 0.3 0.2, 0.4	0.5, 0.7 0.8, 0.6	1.5
4-II	1, 2 3, 4	1, 2 3, 2	3, 5 8, 5	0.2, 0.3 0.2, 0.5	0.8, 0.7 0.8, 0.5	1.5
4-III	1, 2 3, 4	1, 2 3, 2	3, 5 8, 5	0.1, 0.2 0.9, 0.8	0.9, 0.8 0.1, 0.2	1.5
4-IV	1, 2 3, 4	1, 2 3, 2	3, 5 8, 5	0.1, 0.2 0.1, 0.2	0.9, 0.8 0.9, 0.8	1.5
4-V	1, 2 3, 4	1, 2 3, 2	3, 5 8, 5	0.9, 0.8 0.1, 0.2	0.1, 0.2 0.9, 0.8	1.5
6-I	1, 2 3, 4 5, 6	0.5, 0.7 1.2, 1.6 1.3, 0.9	1, 2 3.5, 5 4, 2.5	0.5, 0.45 0.3, 0.2 0.4, 0.4	0.5, 0.55 0.7, 0.8 0.6, 0.6	1.5
6-II	= 6-I	= 6-I	= 2×6-I	= 6-I	= 6-I	= 6-I
8-I	1, 2 3, 4 5, 6 7, 8	0.55, 0.45 0.65, 0.8 0.77, 0.73 0.74, 0.59	0.6, 0.8 0.9, 1.1 1, 0.95 0.82, 0.73	0.5, 0.45 0.3, 0.2 0.3, 0.35 0.4, 0.4	0.5, 0.55 0.7, 0.8 0.7, 0.65 0.6, 0.6	1.5

offline and runtime is not significant concern for our testcases, we use $\Delta_{fOD} = 50$MHz to achieve higher value of the objective function.

Impact of $w_{\mathbf{nom,m}}$ and $w_{\mathbf{OD,m}}$. Table 5.25 compares solutions of all testcases with four cores, 4-I – 4-V, from our optimal flow. The value of the objective function depends on the values of $E_{nom,m} \times w_{nom,m}$ and $E_{OD,m} \times w_{OD,m}$. In our experiments, we set $f_{nom,m} = 1.5$GHz for all m, so the value depends on $E_{OD,m} \times w_{OD,m}$. At smaller values of m, $f_{OD,m}$ is larger as compared to larger values of m because as m increases, overdrive frequency is limited by the maximum die temperature T_{max}. Specifically, because $E_{OD,m} \times w_{OD,m}$ is higher in 4-I – 4-IV for $m = 1,2,3$ than in 4-V, testcases 4-I – 4-IV yield a higher value of the objective function than 4-V. Testcase 4-IV

has the highest value of $E_{OD,3} \times w_{OD,3}$, so it yields the highest value of the objective function even though $f_{OD,3}$ for 4-IV is the same as 4-III. Therefore, larger values of $f_{OD,m}$ are achieved for smaller values of m and larger value of the objective function is achieved when $E_{OD,m} \times w_{OD,m}$ is large for these corresponding values of m.

Table 5.24: Objective function value vs. Δ_{fOD}. Larger value is better.

Testcase	m	Δ_{fOD}	$f_{OD,m}$ (GHz)	Objective function value	$iter$
4-I	1, 2 3, 4	200MHz	3.0, 2.2 1.8, 1.8	32870	21
4-I	1, 2 3, 4	50MHz	2.85, 2.3 1.8, 1.8	32995	271
6-I	1, 2, 3 4, 5, 6	100MHz	2.8, 2.9, 2.6 1.8, 1.8, 1.8	28367.5	1500
6-I	1, 2, 3 4, 5, 6	50MHz	2.85, 2.9, 2.6 1.8, 1.8, 1.8	28392.5	10572
8-I	1, 2, 3 4, 5, 6 7, 8	100MHz, 50MHz	2.9, 2.9, 2.8 1.8, 1.8, 1.8 1.8, 1.8	12316.0, 12316.0	1845, 11254

Table 5.25: Objective function value vs. weights at $\Delta_{fOD} = 50$MHz.

Testcase	m	$f_{OD,m}$ (GHz)	Objective function value	$iter$
4-I	1, 2, 3, 4	2.85, 2.3, 1.8, 1.8	32995	271
4-II	1, 2, 3, 4	2.95, 2.25, 2.05, 1.8	36175	408
4-III	1, 2, 3, 4	2.95, 2.35, 2.05, 1.8	28005	424
4-IV	1, 2, 3, 4	2.95, 2.35, 2.05, 1.8	41125	424
4-V	1, 2, 3, 4	3.0, 2.3, 2.0, 1.8	29600	410

Task migration policy based on optimal values of $v_{j,m,l}$ and $u_{i,l}$ for Testcase 8-I. Based on notations and terminologies described in Section 5.2.1, Table 5.26 shows values of $v_{j,m,l}$ for each active combination j of $1 \leq m \leq 8$ active cores in both nominal and overdrive execution modes. For example, $v_{11,2,OD} = v_{17,2,OD} = 25.8$ for $C(N = 8, m = 2)$. Out of total lifetime of seven years, each core is active as follows: core 1 for 12.91%, core 2 for 10.29%, core 3 for

Table 5.26: Comparison of % execution time in each core and in each active combination with 8-I, Δ_{fOD} = 50MHz. Execution times are shown as nominal/overdrive.

	$C(8,2) == 28$ combinations ($j = 1,2,...,28$)								$C(8,6) == 28$ combinations ($j = 1,2,...,28$)							
	Cores								Cores							
j	1	2	3	4	5	6	7	8	1	2	3	4	5	6	7	8
3	50/0	-	-	50/0	-	-	-	-	1.3/0	1.3/0	1.3/0	1.3/0	1.3/0	-	-	1.3/0
4	0/24.2	-	-	-	0/24.2	-	-	-	0/0.3	0/0.3	0/0.3	0/0.3	-	0/0.3	0/0.3	-
5	0/0	-	-	-	-	0/0	-	-	0/0	0/0	0/0	0/0	-	0/0	-	0/0
9	-	0/0	-	0/0	-	-	-	-	0/31	0/31	0/31	-	0/31	-	0/31	0/31
11	-	0/25.8	-	-	-	0/25.8	-	-	0/0	0/0	-	0/0	0/0	0/0	0/0	-
12	-	0/0	-	-	-	-	0/0	-	98.7/7.8	98.7/7.8	-	98.7/7.8	98.7/7.8	98.7/7.8	-	98.7/7.8
13	-	0/0	-	-	-	-	-	0/0	0/0.3	0/0.3	-	0/0.3	0/0.3	-	0/0.3	0/0.3
16	-	-	0/0	-	-	0/0	-	-	0/2	-	0/2	0/2	0/2	0/2	0/2	-
17	-	-	0/25.8	-	-	-	0/25.8	-	0/0	-	0/0	0/0	0/0	0/0	-	0/0
19	-	-	-	0/0	0/0	-	-	-	0/29.7	-	0/29.7	0/29.7	-	0/29.7	0/29.7	0/29.7
22	-	-	-	0/24.2	-	-	-	0/24.2	-	0/28.7	0/28.7	0/28.7	0/28.7	0/28.7	0/28.7	-
23	-	-	-	-	0/0	0/0	-	-	-	0/0	0/0	0/0	0/0	0/0	-	0/0
25	-	-	-	-	50/0	-	-	50/0	-	0/0	0/0	0/0	-	0/0	0/0	0/0
Sum	50/24.2	0/25.8	0/25.8	50/24.2	50/24.2	0/25.8	0/25.8	50/24.2	100/71.1	100/68.1	1.3/91.7	100/68.8	100/69.8	98.7/68.5	0/92	100/68.8

11.46%, core 4 for 11.58%, core 5 for 12.44%, core 6 for 10.05%, core 7 for 9.66% and core 8 for 13.44%. As expected, as they enjoy better heat removal, the cores towards the periphery of the die (e.g., cores 1, 4, 5, 8) are active for longer duration (higher percentages of chip lifetime) than other cores [53].[112] *The operating system scheduler can use the optimal values of $v_{j,m,l}$ and $u_{i,l}$ to determine how long each core should execute at nominal and overdrive frequencies for balanced wearout.* Using the linear programming approach to determine the execution time of each combination leads to the times being very imbalanced across combinations. As part of our future work, we will explore techniques such as geometric programming which may lead to more balanced solutions.

Comparisons of optimal, heuristic, and RC-LG solutions. We compare our optimal solutions with (1) our heuristic solutions and (2) baseline (RC-LG) solutions [53] and [213]. Figures 5.26–5.28 show the results of these comparisons. Figure 5.26 shows that for our testcases, the heuristic solutions are at most 3.3% worse than the optimal solutions, but can converge in up to 10× fewer iterations as shown in Figure 5.27 when compared to the number of iterations in Table 5.24. Moreover, our solutions guarantee that all tasks in each combination of m execute at $f_{OD,m} \geq$ 1.8GHz, i.e., the solutions meet "acceptable overdrive performance" requirements. In addition, we guarantee "acceptable throughput" because all tasks complete within the multi-core system's lifetime. Figure 5.26 shows that the baseline method's solutions can be up to

[112]To confirm the optimality of these solutions, we separately solve the dual program for each m, minimize $\sum_{i=N}^{2N} \lambda_i MTTF_i + \mu_1 E_{nom,m} + \mu_2 E_{OD,m}$ and verify that we obtain identical objective function values.

17.4% below optimal, even as they execute around 27% of the overall overdrive execution time below the minimum required frequency of 1.8GHz as shown in Figure 5.28. In other words, the baseline solutions do not meet "acceptable performance" requirements. For Testcase 6-II, the baseline flow achieves a solution within lifetime constraints, but unfortunately executes 75.5% of the overall overdrive execution time at frequencies below 1.8GHz. Runtime versus accuracy tradeoffs would determine a user's choice between optimal and heuristic methods.

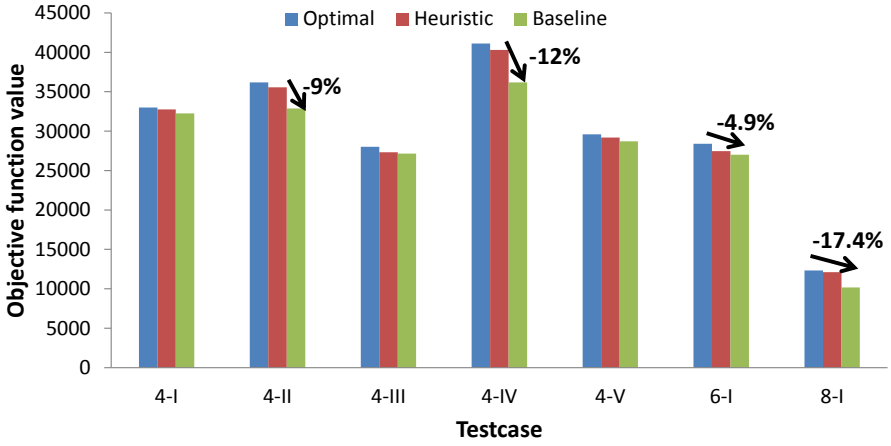


Figure 5.26: Comparison of objective function values from optimal, heuristic, and baseline solutions for seven testcases that yield an optimal solution.

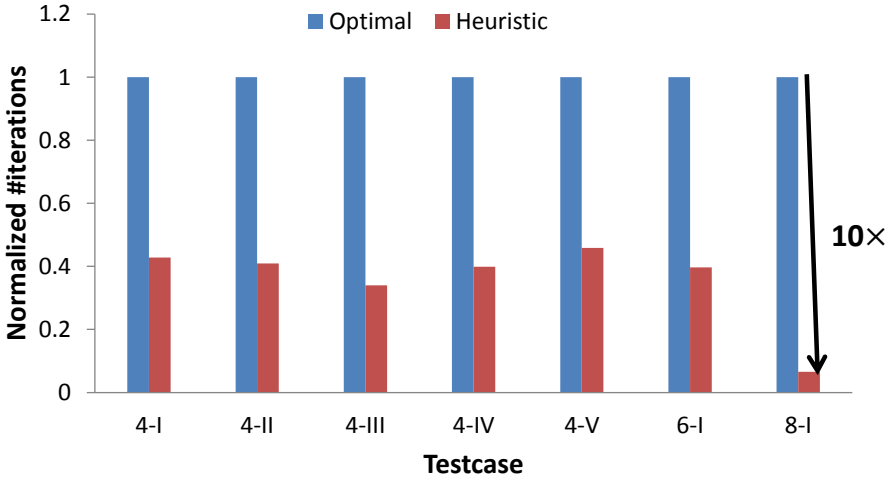


Figure 5.27: Comparison of the normalized #iterations (normalized with respect to the #iterations of the optimal solution) between optimal and heuristic solutions for seven testcases that yield an optimal solution.

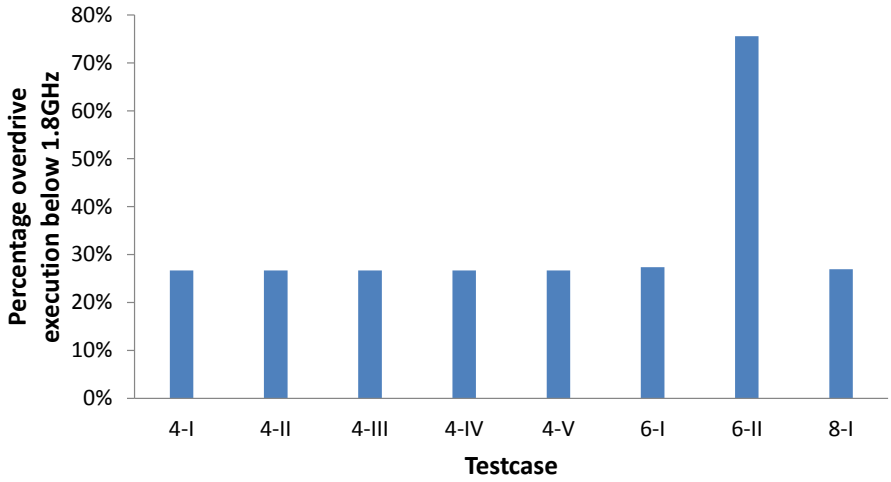


Figure 5.28: Percentage of overdrive execution time below "acceptable performance" of 1.8GHz of baseline solutions. Across all testcases, the baseline solutions will execute 27% of the overall overdrive execution time below the minimum required frequency of 1.8GHz. Our optimal and heuristic solutions always *guarantee* execution at overdrive frequencies $\geq$1.8GHz.

5.3.3 Conclusions

When scheduling tasks in multi-core systems implemented in leading-edge IC technologies, reliability awareness is critical to achieving guaranteed lower bounds on performance and throughput. With this in mind, we have formulated and solved a new *maximum-value, reliability-constrained overdrive frequencies* (MVRCOF) problem. We show how an optimal (discretized) solution may be found using exhaustive search, and we propose a heuristic that maximizes the overdrive frequency in the order of user-specified weights for each operating mode. We develop a solver that serves as the foundation of both the optimal and heuristic flows. Our methods are the first to *guarantee* both acceptable performance and throughput, in that all tasks are executable for their entire duration at the optimal overdrive frequencies, without exceeding the total lifetime requirement of any core. Further, we determine optimal execution times of each core in each mode; these can be utilized by schedulers for balanced wearout of cores. Experimentally, across eight testcases on between 4 and 8 cores, our optimal overdrive frequencies achieve between 2.2% and 17.4% greater value than existing RC-LG policies [53] and [213] (the largest improvement is for the 8-core testcase).

5.4 Acknowledgments

Chapter 5 is in part a reprint of A. B. Kahng and S. Nath, "Optimal Reliability-Constrained Overdrive Frequency Selection in Multicore Systems", *Proc. International Symposium on Quality Electronic Design*, 2014; and the submitted drafts: W.-T. J. Chan, Y. Du, A. B. Kahng, S. Nath, K. Samadi and H. Yao, "Toward "True 3D": Assessment of a New Objective in Analytic 3DIC Placement", *Proc. Asia and South Pacific Design Automation Conference*, submitted draft; and P. Agrawal, M. Broxterman, B. Chatterjee, P. Cuevas, K. H. Hayashi, A. B. Kahng, P. K. Myana and S. Nath, "Optimal Scheduling and Allocation for IC Design Management and Cost Reduction", *ACM Transactions on Design Automation of Electronic Systems*, submitted draft.

I would like to thank my co-authors Prabhav Agrawal, Mike Broxterman, Wei-Ting J. Chan, Biswadeep Chatterjee, Patrick Cuevas, Dr. Yang Du, Kathy H. Hayashi, Professor Andrew B. Kahng, Pranay K. Myana, Dr. Kambiz Samadi and Professor Hailong Yao.

I also thank Prof. Sungkyu Lim, Dr. Shreepad Panth and Kyungwook Chang of Georgia Tech for their generosity with time and bandwidth to confirm correctness of our replication of, and comparisons with, the shrunk2D flow. I thank the authors of [112] [113] [114] for providing APlace source code.

Bibliography

[1] K. Acharya, K. Chang, B. W. Ku, S. Panth, S. Sinha, B. Cline, G. Yeric and S. K. Lim, "Monolithic 3D IC Design: Power, Performance, and Area Impact at 7nm", *Proc. International Symposium on Quality Electronic Design*, 2016, pp. 41-48.

[2] S. M. Afifi, H. G. Hosseini and R. Sinha, "Hardware Implementations of SVM on FPGA: A State-of-the-Art Review of Current Practice", *International Journal of Innovative Science, Engineering & Technology* 2(11) (2015), pp. 733-752.

[3] N. Agarwal, T. Krishna, L.-S. Peh and N. K. Jha, "GARNET: A Detailed On-Chip Network Model Inside a Full-System Simulator", *Proc. IEEE International Symposium on Performance Analysis of Systems and Software*, 2009, pp. 33-42.

[4] P. Agrawal, M. Broxterman, B. Chatterjee, P. Cuevas, K. H. Hayashi, A. B. Kahng, P. K. Myana and S. Nath, "Optimal Scheduling and Allocation for IC Design Management and Cost Reduction", *ACM Transactions on Design Automation of Electronic Systems*, 2016, submitted draft.

[5] P. D. Allison, *Multiple Regression: A Primer*, Thousand Oaks, Pine Forge Press, 1999.

[6] C. J. Alpert, C. Chu and P. G. Villarubia, "The Coming of Age of Physical Synthesis", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2007, pp. 246-249.

[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, J. Hu, S. S. Sapatnekar and C. N. Sze, "Accurate Estimation of Global Buffer Delay within a Floorplan", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 25(6) (2006), pp. 1140-1146.

[9] A. Andreev, I. Pavisic and P. Raspopovic, "Multi-Layer Assignment", *U.S. Patent* No. 6,182,272, 2001.

[10] A. Ardakani, F. Leduc-Primeau, N. Onizawa, T. Hanyu and W. J. Gross, "VLSI Implementation of Deep Neural Network Using Integral Stochastic Computing" *arXiv:1509.08972v1* preprint, 2015.

[11] M. Ayala and C. Artigues, "On Integer Linear Programming Formulations for the Resource-Constrained Modulo Scheduling Problem", *Technical Report 10393, Rapport LAAS*, 2010.

[12] M. Becer, V. Zolotov, R. Panda, A. Grinshpon, I. Algor, R. Levy and C. Oh, "Pessimism Reduction in Crosstalk Noise Aware STA", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2005, pp. 954-961.

[13] V. Balabanov, M.-K. Hsu and Y.-W. Chang, "Method of Analytical Placement with Weighted-Average Wirelength Model", *U.S. Patent* No. 8,689,164 B2, 2014.

[14] A. Banerjee, R. Mullins and S. Moore, "A Power and Energy Exploration of Network-on-Chip Architecture", *Proc. IEEE/ACM International Symposium on Networks-on-Chip*, 2007, pp. 163-172.

[15] N. Banerjee, P. Vellanki and K. S. Chatha, "A Power and Performance Model for Network-on-Chip Architectures", *Proc. Design, Automation and Test in Europe*, 2004, pp. 1250-1255.

[16] S. Bansal and R. Goering, "Making 20nm Design Challenges Manageable", http://www.chipdesignmag.com/pdfs/chip_design_special_DAC_issue_2012.pdf (accessed on July 21, 2016).

[17] P. Baptiste and S. Demassey, "Tight LP Bounds for Resource Constrained Project Scheduling", *Operations Research Spectrum* 26 (2004), pp. 251-262.

[18] P. Batude, M. Vinet, A. Pouydebasque, C. Le Royer, B. Previtali, C. Tabone, J.-M. Hartmann, L. Sanchez, L. Baud, V. Carron, A. Toffoli, F. Allain, V. Mazzochhi, D. Lafond, O. Thomas, O. Cueto, N. Bouzaida, D. Fluery, A. Amara, S. Deleonibus and O. Faynot, "Advances in 3D CMOS Sequential Integration", *Proc. IEEE International Electron Devices Meeting*, 2009, pp. 1-4.

[19] D. A. Belsley, "Multicollinearity: Diagnosing Its Presence and Assessing The Potential Damage It Causes Least-Squares Estimation", *National Bureau of Economic Research Working Paper No. 154*, 1976.

[20] D. A. Belsley, *Regression Diagnostics: Identifying Influential Data and Sources of Collinearity*, Wiley, 1980.

[21] J. Bhasker and R. Chadha, *Static Timing Analysis for Nanometer Designs: A Practical Approach*, Springer, 2009.

[22] D. Bienstock and M. Zuckerberg, "A New LP Algorithm for Precedence Constrained Production Scheduling", *Optimization Online* (2009), pp. 1-33.

[23] O. Bilgir, M. Martonosi and Q. Wu, "Exploring the Potential of CMP Core Count Management on Data Center Energy Savings", *Proc. Workshop on Energy Efficient Design*, 2011.

[24] J. R. Black, "Electromigration Failure Modes in Aluminium Metallization for Semiconductor Devices", *IEEE Letters* 57(9) (1969), pp. 1578-1594.

[25] A. Bonfietti, M. Lombardi, L. Benini and M. Milano, "Cross Cyclic Resource-Constrained Scheduling Solver", *Artificial Intelligence* 206 (2014), pp. 25-52.

[26] G. E. P. Box and M. E. Muller, "A Note on the Generation of Random Normal Deviates", *Annals of Mathematical Statistics* 29(2) (1958), pp. 610-611.

[27] U. Brenner and A. Rohe, "An Effective Congestion Driven Placement Framework", *Proc. ACM International Symposium on Physical Design*, 2002, pp. 6-11.

[28] M. D. Buhmann, S. Dinew and E. Larsson, "A Note on Radial Basis Function Interpolation Limits", *IMA Journal of Numerical Analysis* 30 (2010), pp. 543-554.

[29] A. E. Caldwell, A. B. Kahng, S. Mantik, I. L. Markov and A. Zelikovsky, "On Wirelength Estimations for Row-Based Placement", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 18(9) (1999), pp. 1265-1278.

[30] N. Callegari, D. Dramanac, L.-C. Wang and M. S. Abadir, "Classification Rule Learning Using Subgroup Discovery of Cross-Domain Attributes Responsible for Design-Silicon Mismatch", *Proc. IEEE/ACM/EDAC Design Automation Conference*, 2010, pp. 374-379.

[31] A. Cao, S.-M. Chang and D.-C. Yuan, "Local Clock Skew Optimization", *U.S. Patent* No. 8,635,579, 2014.

[32] S. Chakrabartty and G. Cauwenberghs, "Sub-Microwatt Analog VLSI Trainable Pattern Classifier", *Journal of Solid State Circuits* 42(5) (2007), pp. 1169-1179.

[33] J. Chan and S. Parameswaran, "NoCEE: Energy Macro-Model Extraction Methodology for Network-on-Chip Routers", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2005, pp. 254-259.

[34] 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.

[35] 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.

[36] 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.

[37] W.-T. J. Chan, Y. Du, A. B. Kahng, S. Nath and K. Samadi, "3D-IC Benefit Estimation and Implementation Guidance from 2D-IC Implementation", *Proc. IEEE/ACM/EDAC Design Automation Conference*, 2015, pp. 1-6.

[38] W.-T. J. Chan, A. B. Kahng and J. Li, "Revisiting 3DIC Benefit with Multiple Tiers", *Proc. ACM International Workshop on System-Level Interconnect Prediction*, 2016, pp. 6:1-6:8.

[39] W.-T. J. Chan, K. Y. Chung, A. B. Kahng, N. MacDonald and S. Nath, "Learning-Based Prediction of Embedded Memory Timing Failures during Initial Floorplan Design", *Proc. Asia and South Pacific Design Automation Conference*, 2016, pp. 178-185.

[40] W.-T. J. Chan, Y. Du, A. B. Kahng, S. Nath and K. Samadi, "BEOL Stack-Aware Routability Prediction from Placement Using Data Mining Techniques", *Proc. IEEE International Conference on Computer Design*, 2016, to appear.

[41] W.-T. J. Chan, Y. Du, A. B. Kahng, S. Nath, K. Samadi and H. Yao, "Toward "True 3D": Assessment of a New Objective in Analytic 3DIC Placement", *Proc. Asia and South Pacific Design Automation Conference*, 2017, submitted draft.

[42] C.-C. Chang and C.-J. Lin, "LIBSVM: A Library for Support Vector Machines", *ACM Transactions on Intelligent Systems and Technology* 3(2) (2011), pp. 27:1-27:27.

[43] K. Chang, J. Shen and T. Chen, "A Low-Power Crossroad Switch Architecture and Its Core Placement for Network-on-Chip", *Proc. Design, Automation and Test in Europe*, 2005, pp. 375-380.

[44] K. Chang, Georgia Institute of Technology, *personal communications*, 2016.

[45] W.-H. Chang, L.-D. Chen, C.-H. Lin, S.-P. Mu, M. C.-T. Chao, C.-H. Tsai and Y.-C. Chiu, "Generating Routing-Driven Power Distribution Networks with Machine-Learning Technique", *Proc. ACM International Symposium on Physical Design*, 2016, pp. 145-152.

[46] X. Chen and L.-S. Peh, "Leakage Power Modeling and Optimization in Interconnection Networks", *Proc. International Symposium on Low Power Electronic Design*, 2003, pp. 90-95.

[47] C.-K. Cheng, A. B. Kahng, K. Samadi and A. Shayan, "Worst-Case Performance Prediction under Supply Voltage and Temperature Variation", *Proc. ACM International Workshop on System-Level Interconnect Prediction*, 2010, pp. 91-96.

[48] 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.

[49] 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.

[50] N. Christofides, R. Alvarez-Valdes and J. M. Tamarit, "Project Scheduling with Resource Constraints: A Branch and Bound Approach", *European Journal of Operational Research* 29 (1987), pp. 262-273.

[51] C. Chu and Y.-C. Wong, "FLUTE: Fast Lookup Table Based Rectilinear Steiner Minimal Tree Algorithm for VLSI Design" *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 27(1) (2008), pp. 70-83.

[52] M. Clarke, D. Hammerschlag, M. Rardon and A. Sood, "Eliminating Routing Congestion Issues with Logic Synthesis", *Whitepaper*, Cadence Design Systems, 2011. http://www.cadence.com/rl/resources/white_papers/routing_congestion_wp.pdf (accessed on July 21, 2016).

[53] A. K. Coskun, T. S. Rosing, K. A. Whisnant and K. C. Gross, "Static and Dynamic Temperature-Aware Scheduling for Multiprocessor SoCs", *IEEE Transactions on Very Large Scale Integration Systems* 16(9) (2008), pp. 1127-1140.

[54] A. K. Coskun, R. Strong, D. M. Tullsen and T. S. Rosing "Evaluating the Impact of Job Scheduling and Power Management on Processor Lifetime for Chip", *Proc. ACM SIGMETRICS*, 2009, pp. 169-180.

[55] 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.

[56] J. Cong and G. Luo, "A Multilevel Analytical Placement for 3D ICs", *Proc. Asia and South Pacific Design Automation Conference*, 2009, pp. 361-366.

[57] J. Cong and G. Luo, "An Analytical Placer for Mixed-Size 3D Placement", *Proc. ACM International Symposium on Physical Design*, 2010, pp. 61-66.

[58] N. Cressie, "Geostatistics", *The American Statistician* 43(4) (1989), pp. 197-202.

[59] K. Crombecq, L. De Tommasi, D. Gorissen and T. Dhaene, "A Novel Sequential Design Strategy for Global Surrogate Modeling", *Proc. Winter Simulation Conference*, 2009, pp. 731-742.

[60] W. J. Dally and B. Towles, *Principles and Practices of Interconnection Networks*, Morgan Kaufmann, 2004.

[61] D. Ding, J.-R. Gao, K. Yuan and D. Z. Pan, "AENEID: A Generic Lithography-Friendly Detailed Router Based on Post-RET Data Learning and Hotspot Detection", *Proc. IEEE/ACM/EDAC Design Automation Conference*, 2011, pp. 795-800.

[62] D. Ding, B. Yu, J. Ghosh and D. Z. Pan, "EPIC: Efficient Prediction of IC Manufacturing Hotspots with a Unified Meta-Classification Formulation", *Proc. Asia and South Pacific Design Automation Conference*, 2012, pp. 263-270.

[63] X. Dong, J. Zhao and Y. Xie, "Fabrication Cost Analysis and Cost-Aware Design Space Exploration for 3D-ICs", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 29(12) (2010), pp. 1959-1972.

[64] F. Dubois, V. Catalano, M. Coppola and F. Petrot, "Accurate On-Chip Router Area Modeling with Kriging Methodology", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2012, pp. 450-457.

[65] J. Dyck and K. Singhal, "De-Risking Variation-Aware Custom IC Design with Solido Variation Designer and Synopsys HSPICE", *Whitepaper*, Synopsys Inc., 2010. https://www.synopsys.com/Tools/Implementation/CustomImplementation/CapsuleModule/Solido-HSPICE-wp.pdf (accessed on July 21, 2016).

[66] T. El Motassadeq, V. Sarathi, S. Thameem and M. Nijam, “SPICE versus STA Tools: Challenges and Tips for Better Correlation”, *Proc. IEEE International System-on-Chip Conference*, 2009, pp. 325-328.

[67] T. El Motassadeq, “CCS vs NLDM Comparison Based on a Complete Automated Correlation Flow between PrimeTime and HSPICE”, *Proc. Saudi International Electronics, Communications and Photonics Conference*, 2011, pp. 1-5.

[68] R. Embree, *personal communication*, July 2013.

[69] C. Fagot, P. Girard and C. Landrault, “On Using Machine Learning for Logic BIST”, *Proc. IEEE International Test Conference*, 1997, pp. 338-346.

[70] S. Fenstermaker, D. George, A. B. Kahng, S. Mantik and B. Thielges, “METRICS: A System Architecture for Design Process Optimization”, *Proc. IEEE/ACM/EDAC Design Automation Conference*, 2000, pp. 705-710.

[71] T. Fountain, T. Dietterich and B. Sudyka, “Mining IC Test Data to Optimize VLSI Testing“, *Proc. ACM SIGKDD International Conference*, 2000, pp. 18-25.

[72] Y. Freund and R. E. Schapire, “A Short Introduction to Boosting”, *Journal of Japanese Society for Artificial Intelligence* 14(5) (1999), pp. 771-780.

[73] R. Friese, T. Brinks, C. Oliver, H. J. Siegel and A. A. Maciejewski, “Analyzing the Trade-Offs between Minimizing Makespan and Minimizing Energy Consumption in a Heterogeneous Resource Allocation Problem”, *Proc. International Conference on Advanced Communications and Computation*, 2012, pp. 81-89.

[74] S. Ganapathy, R. Canal, A. Gonzalez and A. Rubio, “Circuit Propagation Delay Estimation through Multivariate Regression-Based Modeling under Spatio-Temporal Variability”, *Proc. Design, Automation and Test in Europe*, 2010, pp. 417-422.

[75] R. Gandikota, K. Chopra, D. Blaauw and D. Sylvester, “Victim Alignment in Crosstalk-Aware Timing Analysis”, *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 29(2) (2010), pp. 261-274.

[76] T. Goel, R. T. Haftka, W. Shyy and N. V. Queipo “Ensemble of Surrogates”, *Structural and Multidisciplinary Optimization* 33(3) (2007), pp. 199-216.

[77] R. Goering, “What’s Needed to “Fix” Timing Signoff?”, *DAC Panel*, 2013.

[78] B. Goplen and S. Sapatnekar, “Efficient Thermal Placement of Standard Cells in 3D ICs using a Force Directed Approach”, *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2003, pp. 86-89.

[79] S. S. Gosavi, “Machine Learning Methods for Fault Classification”, *Master’s Thesis*, Universität Stuttgart, 2013.

[80] G. Guindani, C. Reinbrecht, T. Raupp, N. Calazans and F. G. Moraes, “NoC Power Estimation at the RTL Abstraction Level”, *Proc. IEEE Computer Society Annual Symposium on VLSI*, 2008, pp. 475-478.

[81] S. Hamdioui, "Testing Embedded Memories: A Survey", *Mathematical and Engineering Methods in Computer Science* 7721 (2013), pp. 32-42.

[82] G. Hamerly, E. Perelman, J. Lau, B. Calder and T. Sherwood, "Using Machine Learning to Guide Architecture Simulation", *Journal of Machine Learning* 7 (2006), pp. 343-378.

[83] K. Han, A. B. Kahng, J. Lee, J. Li and S. Nath, "A Global-Local Optimization Framework for Simultaneous Multi-Mode Multi-Corner Skew Variation Reduction", *Proc. IEEE/ACM/EDAC Design Automation Conference*, 2015, pp. 1-6.

[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. J. H. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*, Springer, 2009.

[86] X. He, T. Huang, L. Xiao, H. Tian, G. Cui and E. F. Young, "Ripple: An Effective Routability-Driven Placer by Iterative Cell Movement", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2011, pp. 74-79.

[87] X. He, T. Huang, W.-K. Chow, J. Kuang, K.-C. Lam, W. Cai and E. F. Y. Young "Ripple 2.0: High Quality Routability-Driven Placement via Global Router Integration", *Proc. IEEE/ACM/EDAC Design Automation Conference*, 2013, pp. 1-6.

[88] F. J. Hickernell, "A Generalized Discrepancy and Quadrature Error Bound", *Mathematics of Computation* 67(221) (1998), pp. 299-322.

[89] L. Huang, F. Yuan and Q. Xu, "Lifetime Reliability-Aware Task Allocation and Scheduling for MPSoC Platforms", *Proc. Design, Automation and Test in Europe*, 2009, pp. 51-56.

[90] K. Huang, H.-G. Stratigopoulos and S. Mir, "Fault Diagnosis of Analog Circuits Based on Machine Learning", *Proc. Design, Automation and Test in Europe*, 2010, pp. 1761-1766.

[91] M.-K. Hsu, S. Chou, T.-H. Lin and Y.-W. Chang, "Routability-Driven Analytical Placement for Mixed-Size Circuit Designs", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2011, pp. 80-84.

[92] M.-K. Hsu, V. Balabanov and Y.-W. Chang, "TSV-Aware Analytical Placement for 3-D IC Designs Based on a Novel Weighted-Average Wirelength Model", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 32(4) (2013), pp. 497-509.

[93] J. Hu, A. B. Kahng, B. Liu, G. Venkataraman and X. Xu, "A Global Minimum Clock Distribution Network Augmentation Algorithm for Guaranteed Clock Skew Yield", *Proc. Asia and South Pacific Design Automation Conference*, 2007, pp. 24-31.

[94] M. D. Hutton and D. Karchmer, "Early Timing Estimation of Timing Statistical Properties of Placement", *U.S. Patent* No. 8,112,728, 2012.

[95] A. A. Ilumoka, "Efficient Prediction of Crosstalk in VLSI Interconnects using Neural Networks", *Proc. Electrical Performance of Electronic Packages and Systems*, 2000, pp. 87-90.

[96] E. Ipek, S. A. McKee, B. R. de Supinski, M. Schulz and R. Caruana, "Efficiently Exploring Architectural Design Spaces via Predictive Modeling", *Proc. ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2006, pp. 195-206.

[97] K. Jeong and A. B. Kahng, "Methodology from Chaos in IC Implementation", *Proc. International Symposium on Quality Electronic Design*, 2010, pp. 885-892.

[98] K. Jeong, A. B. Kahng, B. Lin and K. Samadi, "Accurate Machine Learning-Based On-Chip Router Modeling", *IEEE Embedded Systems Letters* 2(3) (2010), pp. 62-66.

[99] Z.-W. Jiang, B.-Y. Su and Y.-W. Chang, "Routability-Driven Analytical Placement by Net Overlapping Removal for Large-Scale Mixed-Size Designs", *Proc. IEEE/ACM/EDAC Design Automation Conference*, 2008, pp. 167-172.

[100] R. Jin, W. Chen and T. W. Simpson, "Comparative Studies of Metamodeling Techniques under Multiple Modeling Criteria", *Structural and Multidisciplinary Optimization* 23(1) (2001), pp. 1-13.

[101] T. R. Jones, S. L. Crain and J. J. Burkis, "Method for Determining Timing Delays Associated with Placement and Routing of an Integrated Circuit", *U.S. Patent* No. 5,629,860, 1994.

[102] M. Jung, T. Song, Y. Wan, Y.-J. Lee, D. Mohapatra, H. Wang, G. Taylor, D. Jariwala, V. Pitchumani, P. Morrow, C. Webb, P. Fischer and S. K. Lim, "How to Reduce Power in 3D IC Designs: A Case Study with OpenSPARC T2 Core", *Proc. IEEE Custom Integrated Circuits Conference*, 2013, pp. 1-4.

[103] M. Jung, T. Song, Y. Wan, Y. Peng and S. K. Lim, "On Enhancing Power Benefits in 3D ICs: Block Folding and Bonding Styles Perspective", *Proc. IEEE/ACM/EDAC Design Automation Conference*, 2014, pp. 1-6.

[104] A. B. Kahng, S. Mantik and D. Stroobandt, "Requirements for Models of Achievable Routing", *Proc. ACM International Symposium on Physical Design*, 2000, pp. 4-11.

[105] A. B. Kahng and D. Stroobandt, "Wiring Layer Assignments with Consistent Stage Delays", *Proc. ACM International Workshop on System-Level Interconnect Prediction*, 2000, pp. 115-122.

[106] A. B. Kahng and S. Mantik, "A System for Automatic Recording and Prediction of Design Quality Metrics", *Proc. International Symposium on Quality Electronic Design*, 2001, pp. 81-86.

[107] A. B. Kahng and S. Mantik, "Measurement of Inherent Noise in EDA Tools", *Proc. International Symposium on Quality Electronic Design*, 2002, pp. 206-211.

[108] A. B. Kahng, S. Mantik and D. Stroobandt, "Toward Accurate Models of Achievable Routing", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 20(5) (2001), pp. 648-659.

[109] A. B. Kahng and S. Muddu, "Predictive Modeling of Lithography-Induced Linewidth Variation", *SPIE Photomask and Next-Generation Lithography Mask Technology*, 2008, pp. 70280M-1-70280-14.

[110] A. B. Kahng and G. Smith, "A New Design Cost Model for the 2001 ITRS", *Proc. International Symposium on Quality Electronic Design*, 2002, pp. 190-193.

[111] A. B. Kahng and X. Xu, "Accurate Pseudo-Constructive Wirelength and Congestion Estimation", *Proc. ACM International Workshop on System-Level Interconnect Prediction*, 2003, pp. 61-68.

[112] A. B. Kahng and Q. Wang, "Implementation and Extensibility of an Analytic Placer", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 24(5) (2005), pp. 734-747.

[113] A. B. Kahng, S. Reda and Q. Wang, "Architecture and Details of a High Quality, Large-Scale Analytical Placer", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2005, pp. 890-897.

[114] A. B. Kahng, S. Reda and Q. Wang, "APlace: A General Analytic Placement Framework", *Proc. ACM International Symposium on Physical Design*, 2005, pp. 233-235.

[115] A. B. Kahng, B. Li, L.-S. Peh and K. Samadi, "ORION 2.0: A Fast and Accurate NoC Power and Area Model for Early-Stage Design Space Exploration", *Proc. Design, Automation and Test in Europe*, 2009, pp. 423-428.

[116] A. B. Kahng, B. Lin and K. Samadi, "Improved On-Chip Router Analytical Power and Area Modeling" *Proc. Asia and South Pacific Design Automation Conference*, 2010, pp. 241-246.

[117] A. B. Kahng, "The Road Ahead: Product Futures", *IEEE Design and Test of Computers* 28(6) (2011), pp. 88-89.

[118] A. B. Kahng, J. Lienig, I. L. Markov and J. Hu, *VLSI Physical Design: From Graph Partitioning to Timing Closure*, New York, Springer, 2011.

[119] A. B. Kahng, B. Lin and S. Nath, "Explicit Modeling of Control and Data for Improved NoC Router Estimation", *Proc. IEEE/ACM/EDAC Design Automation Conference*, 2012, pp. 392-397.

[120] A. B. Kahng, B. Lin and S. Nath, "Comprehensive Modeling Methodologies for NoC Router Estimation", *Technical Report CS2012-0989*, University of California, San Diego Computer Science and Engineering Department, 2012.

[121] A. B. Kahng, "The ITRS Design Technology and System Drivers Roadmap: Process and Status", *Proc. IEEE/ACM/EDAC Design Automation Conference*, 2013, pp. 1-6.

[122] 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.

[123] A. B. Kahng, B. Lin and S. Nath, "High-Dimensional Metamodeling for Prediction of Clock Tree Synthesis Outcomes", *Proc. ACM International Workshop on System-Level Interconnect Prediction*, 2013, pp. 1-7.

[124] A. B. Kahng, S. Nath and T. S. Rosing, "On Potential Design Impacts of Electromigration Awareness", *Proc. Asia and South Pacific Design Automation Conference*, 2013, pp. 527-532.

[125] A. B. Kahng, S. Kang, H. Lee, S. Nath and J. Wadhwani, "Learning-Based Approximation of Interconnect Delay and Slew in Signoff Timing Tools", *Proc. ACM International Workshop on System-Level Interconnect Prediction*, 2013, pp. 1-8.

[126] A. B. Kahng, "Lithography-Induced Limits to Scaling of Design Quality", *Proc. SPIE Conference on Design-Process-Technology Co-Optimization for Manufacturability*, 2014, pp. 905302-1-905302-14.

[127] A. B. Kahng and S. Nath, "Optimal Reliability-Constrained Overdrive Frequency Selection in Multicore Systems", *Proc. International Symposium on Quality Electronic Design*, 2014, pp. 300-308.

[128] A. B. Kahng, M. Luo and S. Nath, "SI for Free: Machine Learning of Interconnect Coupling Delay and Transition Effects", *Proc. ACM International Workshop on System-Level Interconnect Prediction*, 2015, pp. 1-8.

[129] A. B. Kahng, B. Lin and S. Nath, "ORION3.0: A Comprehensive NoC Router Estimation Tool", *IEEE Embedded Systems Letters* 7(2) (2015), pp. 41-45.

[130] A. B. Kahng and X. Xu, "Accurate Pseudo-Constructive Wirelength and Congestion Estimation", *Proc. ACM International Workshop on System-Level Interconnect Prediction*, 2003, pp. 61-68.

[131] E. Karl, D. Blaauw, D. Sylvester and T. Mudge, "Multi-Mechanism Reliability Modeling and Management in Dynamic Systems", *IEEE Transactions on Very Large Scale Integration Systems* 16(4) (2008), pp. 476-487.

[132] U. R. Karpuzcu, B. Greskamp and J. Torrellas, "The BubbleWrap Many-Core: Popping Cores for Sequential Acceleration", *IEEE International Symposium on Microarchitecture*, 2009, pp. 447-458.

[133] 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.

[134] B. Keller and G. Bayraksan, "Scheduling Jobs Sharing Multiple Resources under Uncertainty: A Stochastic Programming Approach", *IIE Transactions* 42(1) (2009), pp. 16-30.

[135] D. H. Kim, K. Athikulwongse and S. K. Lim, "A Study of Through-Silicon-Via Impact on the 3D Stacked IC Layout", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2009, pp. 674-680.

[136] D. H. Kim and S. K. Lim, "Through-Silicon-Via-Aware Delay and Power Prediction Model for Buffered Interconnects in 3D ICs", *Proc. ACM International Workshop on System-Level Interconnect Prediction*, 2010, pp. 25-32.

[137] M.-C. Kim, J. Hu, D.-J. Lee and I. L. Markov, "A SimPLR Method for Routability-Driven Placement", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2011, pp. 67-73.

[138] M. Kim, B.-G. Ahn, J. Kim, B. Lee and J. Chong, "Thermal Aware Timing Budget for Buffer Insertion in Early Stage of Physical Design", *Proc. IEEE International Symposium on Circuits and Systems*, 2012, pp. 357-360.

[139] D. H. Kim, R. O. Topaloglu and S. K. Lim, "Block-Level 3D IC Design with Through-Silicon-Via Planning", *Proc. Asia and South Pacific Design Automation Conference*, 2012, pp. 335-340.

[140] D. H. Kim, K. Athikulwongse, M. Healy, M. Hossain, M. Jung, I. Khorosh, G. Kumar, Y.-J. Lee, D. Lewis, T.-W. Lin, C. Liu, S. Panth, M. Pathak, M. Ren, G. Shen, T. Song, D. H. Woo, X. Zhao, J. Kim, H. Choi, G. Loh, H.-H. Lee and S. K. Lim, "3D-MAPS: 3D Massively Parallel Processor with Stacked Memory", *Proc. International Solid State Circuits Conference*, 2012, pp. 188-189.

[141] D. H. Kim, S. Mukhopadhyay and S. K. Lim, "TSV-Aware Interconnect Distribution Models for Prediction of Delay and Power Consumption of 3-D Stacked ICs", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 33(9) (2014), pp. 1384-1395.

[142] R. Kolisch and S. Hartmann, "Heuristic Algorithms for the Resource-Constrained Project Scheduling Problem: Classification and Computational Analysis", *International Series in Operations Research & Management Science* 14 (1999), pp. 147-178.

[143] R. Kolisch and A. Sprecher, "PSPLIB – A Project Scheduling Problem Library", *European Journal of Operational Research* 96 (1996), pp. 205-216.

[144] R. Kolisch, A. Sprecher and A. Drexl, "Characterization and Generation of a General Class of Resource-Constrained Project Scheduling Problems", *Management Science* 41(10) (1992), pp. 1693-1703.

[145] S. Kotisantis and D. Kanellopoulos, "Combining Bagging, Boosting and Random Subspace Ensembles for Regression Problems", *International Journal of Innovative Computing, Information and Control* 8(6) (2012), pp. 3953-3961.

[146] B. A. Kramer and C. L. Hwang, "Resource Constrained Project Scheduling: Modeling with Multiple Alternatives", *Mathematical and Computational Modeling* 15(8) (1991), pp. 49-63.

[147] T.-W. Kuan, J.-F. Wang, J.-C. Wang, P.-C. Lin and G.-H. Gu, "VLSI Design of an SVM Learning Core on Sequential Minimal Optimization Algorithm", *IEEE Transactions on Very Large Scale Integration Systems* 20(4) (2012), pp. 673-683.

[148] B. C. Lee and D. M. Brooks, "Accurate and Efficient Regression Modeling for Microarchitectural Performance and Power Prediction", *Proc. ACM International Conference on Architectural Support for Programming Languages and Operating Systems*, 2006, pp. 185-194.

[149] B. C. Lee, D. M. Brooks, B. R. de Supinski, M. Schulz, K. Singh and S. A. McKee, "Methods of Inference and Learning for Performance Modeling of Parallel Applications", *Proc. ACM Symposium on Principles and Practice of Parallel Programming*, 2007, pp. 249-258.

[150] S. E. Lee and N. Bagherzadeh, "A High Level Power Model for Network-on-Chip (NoC) Router", *Integration, the VLSI Journal* 35(6) (2009), pp. 1-7.

[151] J. Leverich, M. Monchiero, V. Talwar, P. Ranganathan and C. Kozyrakis, "Power Management of Datacenter Workloads Using Per-Core Power Gating", *IEEE Computer Architecture Letters* 8(2) (2009), pp. 48-51.

[152] M. Li, Y. Zhang, W. Jiang and J. Xie, "A Particle Swarm Optimization Algorithm with Crossover for Resource Constrained Project Scheduling Problem", *Proc. International Conference on Services Science, Management and Engineering*, 2009, pp. 69-72.

[153] X. Li and H. Liu, "Statistical Regression for Efficient High-Dimensional Modeling of Analog and Mixed-Signal Performance Variations", *Proc. IEEE/ACM/EDAC Design Automation Conference*, 2008, pp. 38-43.

[154] X. Q. Li and M. A. Jabri, "Machine Learning-Based VLSI Cells Shape Function Estimation", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 17(7) (2002), pp. 613-623.

[155] Y. Li, J. Han and W. Zhou, "Cress: Dynamic Scheduling for Resource Constrained Jobs", *Proc. International Conference on Computational Science and Engineering*, 2014, pp. 1945-1952.

[156] S.-W. Liao, T.-H. Hung, D. Nguyen, C. Chou, C. Tu and H. Zhou, "Machine Learning-Based Prefetch Optimization for Data Center Applications", *Proc. ACM Conference on High Performance Computing Networking, Storage and Analysis*, 2009, pp. 1-10.

[157] S. K. Lim, Georgia Institute of Technology, *personal communications*, 2016.

[158] M.-S. Lin, "Top Layers of Metal for High Performance ICs", *U.S. Patent* No. 6,383,916, 2002.

[159] F. Liu, "A General Framework for Spatial Correlation Modeling in VLSI Design", *Proc. IEEE/ACM/EDAC Design Automation Conference*, 2007, pp. 817-822.

[160] H. Liu, A. Singhee, R. A. Rutenbar and L. R. Carley, "Remembrance of Circuits Past: Macromodeling by Data Mining in Large Analog Design Spaces", *Proc. IEEE/ACM/EDAC Design Automation Conference*, 2002, pp. 437-442.

[161] H. Liu, "A Measurement Study of Server Utilization in Public Clouds", *Proc. Dependable Autonomic and Secure Computing*, 2011, pp. 435-442.

[162] W.-H. Liu, Y.-L. Li and C.-K. Koh, "A Fast Maze-Free Routing Congestion Estimator with Hybrid Unilateral Monotonic Routing", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2012, pp. 713-719.

[163] W.-H. Liu, T.-K. Chien and T.-C. Wang, "Region-Based and Panel-Based Algorithms for Unroutable Placement Recognition", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34(4) (2015), pp. 502-514.

[164] S. N. Lophaven, H. B. Nielsen and J. Sondergaard, "Aspects of the MATLAB Toolbox DACE", *Technical Report IMM-REP-2002-13*, Technical University of Denmark, 2002.

[165] J. Lu, H. Zhuang, I. Kang, P. Chen and C.-K. Cheng, "ePlace-3D: Electrostatics Based Placement for 3D-ICs", *Proc. ACM International Symposium on Physical Design*, 2016, pp. 11-18.

[166] C.-L. Lung, H.-C. Hsiao, Z.-Y. Zeng and S.-Y. Chang, "LP-Based Multi-Mode Multi-Corner Clock Skew Optimization", *Proc. IEEE International Symposium on VLSI Design Automation and Test*, 2010, pp. 335-338.

[167] 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.

[168] W.-K. Mak and C. Chu, "Rethinking the Wirelength Benefit of 3-D Integration", *IEEE Transactions on Very Large Scale Integration Systems* 20(12) (2012), pp. 2346-2351.

[169] J. Mars and L. Tang, UC San Diego, *personal communication*, May 2013.

[170] T. Matsunawa, J.-R. Gao, B. Yu and D. Z. Pan, "A New Lithography Hotspot Detection Framework Based on AdaBoost Classifier and Simplified Feature Extraction", *Proc. SPIE Advanced Lithography*, 2015, pp. 9427S-1-9427S-10.

[171] T. Matsunawa, B. Yu and D. Z. Pan, "Laplacian Eigenmaps and Bayesian Clustering Based Layout Pattern Sampling and Its Applications to Hotspot Detection and OPC", *Proc. Asia and South Pacific Design Automation Conference*, 2016, pp. 679-684.

[172] T. Matsunawa, B. Yu and D. Z. Pan, "Optical Proximity Correction with Hierarchical Bayes Model", *SPIE Journal of Micro/Nanolithography, MEMS, and MOEMS* 15(2) (2016), pp. 021009-1-021009-8.

[173] P. Meloni, I. Loi, F. Angiolini, S. Carta, M. Barbaro, L. Raffo and L. Benini, "Area and Power Modeling for Network-on-Chip with Layout Awareness", *Proc. IEEE/ACM International Conference on VLSI Design*, 2007, pp. 1-12.

[174] K. Mihic, T. Simunic and G. de Micheli, "Reliability and Power Management of Integrated Systems", *Proc. EUROMICRO Conference on Digital System Design*, 2004, pp. 5-11.

[175] D. Milojevic, T. E. Carlson, K. Croes, R. Radojcic, D. F. Ragett, D. Seynhaeve, F. Angiolini, G. Van der Plas and P. Marchal, "Automated PathFinding Tool Chain for 3D-Stacked Integrated Circuits: Practical Case Study", *Proc. IEEE 3D System Integration Conference*, 2009, pp. 1-6.

[176] A. Mishra, J. Kumar and U. Singhal, "Resolving Timing Miscorrelation Using Timing Uncertainties",
http://www.edn.com/design/integrated-circuit-design/4390721/
Resolving-timing-miscorrelation-using-timing-
uncertainties (accessed on July 21, 2016).

[177] 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.

[178] S. A. Mohamed, A. A. Manaf and C. C. Teh, "A Noise and Signal Integrity Verification Flow for Hierarchical Design", *Proc. IEEE International Conference on Computer Application and Industrial Electronics*, 2011, pp. 250-255.

[179] S. Mohanty and M. K. Nayak, "Optimization Model in Human Resource Management for Job Allocation in ICT Project", *International Journal of the Computer, the Internet and Management* 19(3) (2011), pp. 21-27.

[180] R. H. Mohring, A. S. Schulz, F. Stork and M. Uetz, "On Project Scheduling with Irregular Starting Time Costs", *Operations Research Letters* 28 (2001), pp. 149-154.

[181] C. Moon, Synopsys Inc., *personal communication*, July 2013.

[182] D. F. Morrison, *Multivariate Statistical Methods*, 3rd edition, McGraw-Hill Publishing Company, 1990.

[183] R. Mullins, A. West and S. Moore, "The Design and Implementation of a Low-Latency On-Chip Network", *Proc. Asia and South Pacific Design Automation Conference*, 2006, pp. 164-169.

[184] W. Naylor, "Non-Linear Optimization System and Method for Wire Length and Delay Optimization for an Automatic Electric Circuit Placer", *U.S. Patent* No. 6,301,693, 2001.

[185] S. K. Nithin, S. Gowrysankar and S. Chandrasekar, "Dynamic Voltage (IR) Drop Analysis and Design Closure: Issues and Challenges", *Proc. International Symposium on Quality Electronic Design*, 2010, pp. 611-617.

[186] J. O. Ogutu, H.-P. Piepho and T. Schulz-Streeck, "A Comparison of Random Forests, Boosting and Support Vector Machines for Genomic Selection", *BioMedical Central Proceedings* 5(3) (2011), pp. 1-5.

[187] 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.

[188] B. Ozisikyilmaz, G. Memik and A. Choudhary, "Machine Learning Models to Predict Performance of Computer System Design Alternatives", *Proc. IEEE International Conference on Parallel Processing*, 2008, pp. 495-502.

[189] G. Palermo and C. Silvano, "PIRATE: A Framework for Power/Performance Exploration of Network-on-Chip Architectures", *Proc. IEEE International Workshop on Power and Timing Modeling, Optimization and Simulation*, 2004, pp. 521-531.

[190] G. Palermo, C. Silvano and V. Zaccaria, "ReSPIR: A Response Surface-Based Pareto Iterative Refinement for Application-Specific Design Space Exploration", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 28(12) (2009), pp. 1816-1829.

[191] M. Pan and C. Chu, "IPR: An Integrated Placement and Routing Algorithm", *Proc. IEEE/ACM/EDAC Design Automation Conference*, 2007, pp. 59-62.

[192] M. Y. Park and T. Hastie, "L_1 Regularization Path Algorithm for Generalized Linear Models", *Journal of the Royal Statistical Society* 69(4) (2007), pp. 659-677.

[193] S. Panth, K. Samadi, Y. Du and S. K. Lim, "High-Density Integration of Functional Modules Using Monolithic 3D-IC Technology", *Proc. Asia and South Pacific Design Automation Conference*, 2013, pp. 681-686.

[194] S. Panth, Georgia Institute of Technology, *personal communications*, 2014.

[195] S. Panth, Altera, *personal communications*, 2016.

[196] S. Panth, K. Samadi, Y. Du and S. K. Lim, "Design and CAD Methodologies for Low Power Gate-Level Monolithic 3D ICs", *Proc. International Symposium on Low Power Electronic Design*, 2014, pp. 171-176.

[197] 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.

[198] C. S. Patel, S. M. Chai, S. Yalamanchili and D. E. Schimmel, "Power Constrained Design of Multiprocessor Interconnection Networks", *Proc. IEEE International Conference on Computer Design*, 1997, pp. 408-416.

[199] L.-S. Peh, "Flow Control and Micro-Architectural Mechanisms for Extending the Performance of Interconnection Networks" *Ph.D. Thesis*, Stanford University, 2001.

[200] S. Penolazzi and A. Jantsch, "A High Level Power Model for the Nostrum NoC", *Proc. EUROMICRO Conference on Digital System Design*, 2006, pp. 673-676.

[201] Z. Qi, Y. Cai and Q. Zhou, "Accurate Prediction of Detailed Routing Congestion using Supervised Data Learning", *Proc. IEEE International Conference on Computer Design*, 2014, pp. 97-103.

[202] Z. Qian, D.-C. Juan, P. Bogdan, C.-Y. Tsui, D. Marculescu and R. Marculescu, "SVR-NoC: A Performance Analysis Tool for Network-on-Chips using Learning-Based Support Vector Regression Model", *Proc. Design, Automation and Test in Europe*, 2013, pp. 354-357.

[203] Z. Qiong, G. Yichao, Z. Ging, Z. Jie and C. Xuefang, "An Ant Colony Optimization Model for Parallel Machine Scheduling with Human Resource Constraints", *Proc. International Conference on Digital Enterprise Technology Advances in Intelligent and Soft Computing*, 2010, pp. 917–926.

[204] Qualcomm Inc. (IT project manager), *personal communication*, August 2014.

[205] A. Rajaram, J. Hu and R. Mahapatra, "Reducing Clock Skew Variability via Crosslinks", *Proc. IEEE/ACM/EDAC Design Automation Conference*, 2004, pp. 18-23.

[206] 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.

[207] S. Rakheja and N. S. Krishna, "Establishing Timing Correlation between Tools", http://www.edn.com/design/integrated-circuit-design/4313674/Establishing-timing-correlation-between-tools (accessed on July 21, 2016).

[208] C. Reiss, A. Tumanov, G. R. Ganger, R. H. Katz and M. A. Kozuch, "Heterogeneity and Dynamicity of Clouds at Scale: Google Trace Analysis", *Proc. ACM Symposium on Cloud Computing*, 2012.

[209] 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.

[210] J. A. T. Robles, S. M. Fahmy, K. Madkour and J.-Y. Wuu, "Hotspot Detection Based on Machine Learning", *U.S. Patent* No. 8,402,397 B2, 2013.

[211] L. Rokach, A. Feldman, M. Kalech and G. Provan, "Machine-Learning-Based Circuit Synthesis", *Proc. IEEE Convention of Electrical and Electronics Engineers in Israel*, 2012, pp. 1-5.

[212] P. Rong and M. Pedram, "Power-Aware Scheduling and Dynamic Voltage Setting for Tasks Running on a Hard Real-Time System", *Proc. Asia and South Pacific Design Automation Conference*, 2006, pp. 473-478.

[213] T. S. Rosing, K. Mihic and G. de Micheli, "Power and Reliability Management of SoCs", *IEEE Transactions on Very Large Scale Integration Systems* 15(4) (2007), pp. 391-403.

[214] J. A. Roy and I. L. Markov, "Seeing the Forest and the Trees: Steiner Wirelength Optimization in Placement", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 23(4) (2007), pp. 632-644.

[215] M. M. Sabry Aly, M. Gao, G. Hills, C.-S. Lee, G. Pitner, M. M. Shulaker, T. F. Wu and M. Asheghi, "Energy-Efficient Abundant-Data Computing: The N3XT 1,000x", *IEEE Computer* 48(12) (2015), pp. 24-33.

[216] R. Salakhutdinov, J. B. Tenenbaum and A. Torralba, "Learning with Hierarchical-Deep Models", *IEEE Transactions on Pattern Analysis and Machine Intelligence* 35(8) (2013), pp. 1958-1971.

[217] F. Salewski, A. Schirmer and A. Drexl, "Project Scheduling under Resource and Mode Identity Constraints: Model, Complexity, Methods, and Application", *European Journal of Operational Research* 102(1) (1997), pp. 88-110.

[218] R. Samanta, J. Hu and P. Li, "Discrete Buffer and Wire Sizing for Link-Based Non-Tree Clock Networks", *IEEE Transactions on Very Large Scale Integration Systems* 18(7) (2010), pp. 1025-1035.

[219] S. S. Sapatnekar, "Capturing the Effect of Crosstalk on Delay", *Proc. IEEE/ACM International Conference on VLSI Design*, 2000, pp. 364-369.

[220] M. Sarevska, B. Milovanovic and Z. Stankovic, "Reliability of Radial Basis Function – Neural Network Smart Antenna", *Proc. WSEAS International Conference on Communications*, 2005, pp. 1-7.

[221] R. Sasanka, S. V. Adve, Y.-K. Chen and E. Debes, "The Energy Efficiency of CMP vs. SMT for Multimedia Workloads", *Proc. ACM International Conference on Supercomputing*, 2004, pp. 196-206.

[222] P. Sharma, Freescale Inc., *personal communication*, July 2013.

[223] H. Shojaei, A. Davoodi and J. T. Linderoth, "Congestion Analysis for Global Routing via Integer Programming", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2011, pp. 256-162.

[224] H. Shojaei, A. Davoodi and J. T. Linderoth, "Planning for Local Net Congestion in Global Routing", *Proc. ACM International Symposium on Physical Design*, 2013, pp. 85-92.

[225] G. Sigl, K. Doll and F. M. Johannes, "Analytical Placement: A Linear or a Quadratic Objective Function?", *Proc. IEEE/ACM/EDAC Design Automation Conference*, 1991, pp. 427-432.

[226] D. Sinha, L. Guerra e Silva, J. Wang, S. Raghunathan, D. Netrabile and A. Shebaita, "TAU 2013 Variation Aware Timing Analysis Contest", *Proc. ACM International Symposium on Physical Design*, 2013, pp. 171-178.

[227] K. Skadron, M. R. Stan, W. Huang, S. Velusamy, K. Sankaranarayanan and D. Tarjan, "Temperature-Aware Microarchitecture", *Proc. ACM/IEEE International Symposium on Computer Architecture*, 2003, pp. 2-13.

[228] G. Smith, Gary Smith EDA, *personal communication*, September 2013.

[229] G. Smith, Gary Smith EDA, *personal communication*, June 2014.

[230] G. Smith, "Updates of the ITRS Design Cost and Power Models", *Proc. IEEE International Conference on Computer Design*, 2014, pp. 161-165.

[231] P. Spindler and F. M. Johannes, "Fast and Accurate Routing Demand Estimation for Efficient Routability-Driven Placement", *Proc. Design, Automation and Test in Europe*, 2007, pp. 1226-1231.

[232] T. Spyrou, Altera Corporation, *personal communication*, July 2013.

[233] J. Srinivasan, S. V. Adve, P. Bose and J. A. Rivers, "The Case for Lifetime Reliability-Aware Microprocessors", *Proc. ACM/IEEE International Symposium on Computer Architecture*, 2004, pp. 276-287.

[234] M. Stephenson, S. Amarasinghe, M. Martin and U. O'Reilly, "Meta Optimization: Improving Compiler Heuristics with Machine Learning", *Proc. ACM Conference on Programming Language Design and Implementation*, 2003, pp. 77-90.

[235] D. Stroobandt, "Recent Advances in System-Level Interconnect Prediction", *IEEE Circuits and Systems Newsletter* 11 (2000), pp. 4-20.

[236] H. Su and S. S. Sapatnekar, "Hybrid Structured Clock Network Construction", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2001, pp. 333-336.

[237] S. Sunder and K. Scholtman, "Multi-Mode Multi-Corner Clocktree Synthesis", *U.S. Patent* No. US20090217225 A1, 2009.

[238] T. Taghavi, C. J. Alpert, A. Huber, Z. Li, G.-J. Nam and S. Ramji, "New Placement Prediction and Mitigation Techniques for Local Routing Congestion", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2010, pp. 621-624.

[239] F. B. Talbot, "Resource-Constrained Project Scheduling with Time-Resource Tradeoffs: The Nonpreemptive Case", *Management Science* 28(10) (1982), pp. 1197-1210.

[240] A. Tetelbaum, "Method of Estimating a Total Path Delay in an Integrated Circuit Design with Stochastically Weighted Conservatism", *U.S. Patent* No. 7,213,223, 2007.

[241] T. Thiel, "Have I Really Met Timing-Validating PrimeTime Timing Reports with Spice", *Proc. Design, Automation and Test in Europe*, 2004, pp. 114-119.

[242] T. Thorolfsson, K. Gonsalves and P. D. Franzon, "Design Automation for a 3DIC FFT Processor for Synthetic Aperture Radar: A Case Study", *Proc. IEEE/ACM/EDAC Design Automation Conference*, 2009, pp. 51-56.

[243] W. Torell and V. Avelar, "Performing Effective MTBF Comparisons for Data Center Infrastructure", *APC Whitepaper 112*, 2005.

[244] F. Toufexis, A. Papanikolaou, D. Soudris, G. Stamoulis and S. Bantas, "Power, Performance and Area Prediction of 3D ICs during Early Stage Design Exploration in 45nm", *Proc. IEEE International Conference on Electronics Circuits and Systems*, 2011, pp. 715-718.

[245] R.-S. Tsay, "Exact Zero Skew", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 1991, pp. 336-339.

[246] K. Tseng and M. Horowitz, "False Coupling Exploration in Timing Analysis", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 24(11) (2005), pp. 1795-1805.

[247] K. Tsota, C.-K. Koh and V. Balakrishnan, "Guiding Global Placement with Wire Density", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2008, pp. 212-217.

[248] V. Vapnik, *The Nature of Statistical Learning Theory*, Springer-Verlag, 1995.

[249] V. Veetil, K. Chopra, D. Blaauw and D. Sylvester, "Fast Statistical Static Timing Analysis Using Smart Monte Carlo Techniques", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 30(6) (2011), pp. 852-856.

[250] R. Venkatesan, J. A. Davis, K. A. Bowman and J. D. Meindl, "Optimal n-Tier Multilevel Interconnect Architectures for Gigascale Integration (GSI)", *IEEE Transactions on Very Large Scale Integration Systems* 9(6) (2001), pp. 899-912.

[251] C. R. Venugopal, P. Soraiyur and J. Rao, "Evaluation of the PTSI Crosstalk Noise Analysis Tool and Development of an Automated Spice Correlation Suite to Enable Accuracy Validation", *Proc. International Symposium on Quality Electronic Design*, 2008, pp. 334-337.

[252] M. Vujkovic, D. Wadkins, B. Swartz and C. Sechen, "Efficient Timing Closure without Timing Driven Placement and Routing", *Proc. IEEE/ACM/EDAC Design Automation Conference*, 2004, pp. 268-273.

[253] H.-S. Wang, L.-S. Peh and S. Malik, "Orion: A Power-Performance Simulator for Interconnection Networks", *IEEE International Symposium on Microarchitecture*, 2002, pp. 294-305.

[254] L.-C. Wang and M. S. Abadir, "Data Mining in EDA – Basic Principles, Promises and Constraints", *Proc. IEEE/ACM/EDAC Design Automation Conference*, 2014, pp. 1-6.

[255] M. Wang, X. Yang, K. Eguro and M. Sarrafzadeh, "Multicenter Congestion Estimation and Minimization during Placement", *Proc. ACM International Symposium on Physical Design*, 2000, pp. 147-152.

[256] S. Wang and J.-J. Chen, "Thermal-Aware Lifetime Reliability in Multicore Systems", *Proc. International Symposium on Quality Electronic Design*, 2010, pp. 399-405.

[257] S. Ward, D. Ding and D. Z. Pan, "PADE: A High-Performance Placer with Automatic Datapath Extraction and Evaluation through High-Dimensional Data Learning", *Proc. IEEE/ACM/EDAC Design Automation Conference*, 2012, pp. 756-761.

[258] S. I. Ward, N. Viswanathan, N. Y. Zhou, C. C. N. Sze, Z. Li, C. J. Alpert and D. Z. Pan, "Clock Power Minimization using Structured Latch Templates and Decision Tree Induction", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2013, pp. 599-606.

[259] Y. Wei, C. Sze, N. Viswanathan, Z. Li, C. J. Alpert, L. Reddy, A. D. Huber, G. E. Tellez, D. Keller and S. S. Sapatnekar, "GLARE: Global and Local Wiring Aware Routability Evaluation", *Proc. IEEE/ACM/EDAC Design Automation Conference*, 2012, pp. 768-773.

[260] J. Westra, C. Bartels and P. Groeneveld, "Probabilistic Congestion Prediction", *Proc. ACM International Symposium on Physical Design*, 2004, pp. 204-209.

[261] J. Westra and P. Groeneveld, "Is Probabilistic Congestion Estimation Worthwhile?", *Proc. ACM International Workshop on System-Level Interconnect Prediction*, 2005, pp. 99-106.

[262] T. Xiao and M. Marek-Sadowska, "Worst Delay Estimation in Crosstalk Aware Static Timing Analysis", *Proc. IEEE International Conference on Computer Design*, 2000, pp. 115-120.

[263] T. Xiao and M. Marek-Sadowska, "Efficient Delay Calculation in Presence of Crosstalk", *Proc. International Symposium on Quality Electronic Design*, 2000, pp. 491-497.

[264] S. Yaldiz, U. Arslan, X. Li and L. Pileggi, "Efficient Statistical Analysis of Read Timing Failures in SRAM Circuits", *Proc. International Symposium on Quality Electronic Design*, 2009, pp. 617-621.

[265] T. T. Ye, G. de Micheli and L. Benini, "Analysis of Power Consumption on Switch Fabrics in Network Routers", *Proc. IEEE/ACM/EDAC Design Automation Conference*, 2002, pp. 524-529.

[266] M. B. Yelten, T. Zhu, S. Koziel, P. D. Franzon and M. B. Steer, "Demystifying Surrogate Modeling for Circuits and Systems", *IEEE Circuits and Systems Magazine* 12(1) (2012), pp. 45-63.

[267] N. Yosboonruang, A. Na-udom and J. Rungrattanaubol, "A Comparison of Prediction Accuracy of Statistical Models for Computer Simulated Experiments", *Proc. International Conference on Statistics and Applied Statistics*, 2010, pp. 1-15.

[268] B. Yu, J.-R. Gao, D. Ding, X. Zeng and D. Z. Pan, "Accurate Lithography Hotspot Detection Based on PCA-SVM Classifier with Hierarchical Data Clustering", *SPIE Journal of Micro/Nanolithography, MEMS, and MOEMS* 14(1) (2015), pp. 011003-1-011003-12.

[269] B. Yu, D. Z. Pan, T. Matsunawa and X. Zeng, "Machine Learning and Pattern Matching in Physical Design", *Proc. Asia and South Pacific Design Automation Conference*, 2015, pp. 286-293.

[270] B. Yu, X. Xu, S. Roy, Y. Lin, J. Ou and D. Z. Pan, "Design for Manufacturability and Reliability in Extreme-Scaling VLSI" *Science China Information Sciences* 59 (2016), pp. 1-23.

[271] M. L. Yu, "A Study of the Applicability of Hopfield Decision Neural Nets to VLSI CAD", *Proc. IEEE/ACM/EDAC Design Automation Conference*, 1989, pp. 412-417.

[272] Y.-T. Yu, G.-H. Lin, I. H.-R. Jiang and C. Chiang, "Machine-Learning-Based Hotspot Detection using Topological Classification and Critical Feature Extraction", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 34(3) (2015), pp. 460-470.

[273] H. Zhang, B. Yu and E. F. Y. Young, "Enabling Online Learning in Lithography Hotspot Detection with Information-Theoretic Feature Optimization", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2016, to appear.

[274] S. Zhang and K. S. Chatha, "Approximation Algorithm for the Temperature-Aware Scheduling Problem", *Proc. IEEE/ACM International Conference on Computer-Aided Design*, 2007, pp. 281-288.

[275] X. Zhang, F. An, L. Chen and H. J. Mattausch, "Reconfigurable VLSI Implementation for Learning Vector Quantization with On-Chip Learning Circuit", *Japanese Journal of Applied Physics* 55(4S) (2016), pp. 1-6.

[276] J. Zhao and R. Molena, "Techniques to Accelerate Power and Timing Signoff of Advanced-Node SoCs", *Whitepaper*, Cadence Design Systems, 2014. http://www.cadence.com/rl/Resources/white_papers/power_timing_signoff_wp.pdf (accessed on July 21, 2016).

[277] K. Zhong and S. Dutt, "Algorithms for Simultaneous Satisfaction of Multiple Constraints and Objective Optimization in a Placement Flow with Application to Congestion Control", *Proc. IEEE/ACM/EDAC Design Automation Conference*, 2002, pp. 854-859.

[278] Q. Zhou, X. Wang, Z. Qi, Z. Chen, Q. Zhou and Y. Cai, "An Accurate Detailed Routing Routability Prediction Model in Placement", *Proc. Asia Symposium on Quality Electronic Design*, 2015, pp. 119-122.

[279] X. Zhou, J. Yang, Y. Xu, Y. Zhang and J. Zhao, "Thermal-Aware Task Scheduling for 3D Multicore Processors", *IEEE Transactionas on Parallel and Distributed Systems* 21(1) (2010), pp. 60-71.

[280] T. Zhu and P. D. Franzon, "Application of Surrogate Modeling to Generate Compact and PVT-Sensitive IBIS Models", *Proc. Electrical Performance of Electronic Packages and Systems*, 2009, pp. 77-80.

[281] "3D ICs with TSVs – Design Challenges and Requirements." http://www.cadence.com/rl/resources/white_papers/3dic_wp.pdf (accessed on July 21, 2016).

[282] *AMD FX*, http://www.engadget.com/2012/10/23/amd-fx-processor-refresh/ (accessed on July 21, 2016).

[283] *ARM Cortex-M0 processor*, http://www.arm.com/products/processors/cortex-m/cortex-m0.php (accessed on July 21, 2016).

[284] *Cadence Assura QRC*, https://www.cadence.com/content/cadence-www/global/en_US/home/tools/digital-design-and-signoff.html (accessed on July 5, 2016).

[285] *Cadence Encounter RTL Compiler User Guide*, http://www.cadence.com/products/ld/rtl_compiler/pages/default.aspx (accessed on July 21, 2016).

[286] *Cadence Innovus Implementation System*, https://www.cadence.com/content/cadence-www/global/en_US/home/tools/digital-design-and-signoff/hierarchical-design-and-floorplanning/innovus-implementation-system.html (accessed on July 21, 2016).

[287] *Cadence SoC Encounter User Guide*, https://www.cadence.com/content/cadence-www/global/en_US/home/tools/digital-design-and-signoff/block-implementation/first-encounter-design-exploration-and-prototyping.html (accessed on July 21, 2016).

[288] *Cadence Tempus Timing Signoff*, https://www.cadence.com/content/cadence-www/global/en_US/home/tools/digital-design-and-signoff.html (accessed on July 21, 2016).

[289] *CCS*, http://www.opensourceliberty.org/ccspaper/ccs_bgr.pdf (accessed on July 21, 2016).

[290] *CLK Design Automation Inc.*, http://www.clkda.com (accessed on July 21, 2016).

[291] *Clock Routing Rules*, www.cadence.com/Community/blogs/di/archive/2011/05/10/five-minute-tutorial-setting-up-clock-routing-rules.aspx (accessed on July 21, 2016).

[292] *Dassault Systems Enovia Synchronicity*, http://www.3ds.com/products-services/enovia/products/v6/synchronicity-designsync/ (accessed on July 21, 2016).

[293] *Discrete Gate Sizing Contest*, http://www.ispd.cc/contests/13/ispd2013_contest.html (accessed on July 21, 2016).

[294] "*How Green is my Silicon Valley?*" http://dac.com/sites/default/files/DACArchive/pubs/46DAC_Final_Prgm.pdf (accessed on July 21, 2016).

[295] *Google Translate*, http://translate.google.com (accessed on July 21, 2016).

[296] *IBM ILOG CPLEX*, http://www-03.ibm.com/software/products/en/ibmilogcpleoptistud/ (accessed on July 5, 2016).

[297] *IBM Blue Gene processor*, http://www.research.ibm.com/journal/rd49-23.html (accessed on July 21, 2016).

[298] *IBM-PLACE 2.0 Benchmark Suite*, http://vlsicad.eecs.umich.edu/BK/Slots/cache/er.cs.ucla.edu/benchmarks/ibm-place2/ (accessed on July 21, 2016).

[299] *IC Manage*, https://www.icmanage.com/ic-design-management-best-practices/ (accessed on July 21, 2016).

[300] *Incentia Design Systems Inc.*, http://www.incentia.com (accessed on July 21, 2016).

[301] *inMotion Creative Project Management*, http://explore.inmotionnow.com/capterra-project-management (accessed on July 21, 2016).

[302] *Intel 80-core Report*, http://techresearch.intel.com/ProjectDetails.aspx?Id=151 (accessed on July 21, 2016).

[303] *Intel Turbo Boost technology – On-Demand Processor Performance*, http://www.intel.com/content/www/us/en/architecture-and-technology/turbo-boost/turbo-boost-technology.html (accessed on July 21, 2016).

[304] *International Technology Roadmap for Semiconductors*, http://www.itrs2.net/ (accessed on July 21, 2016).

[305] *Failure Mechanisms and Models for Semiconductor Devices*, JEDEC JEP122G, 2011.

[306] *Leon3 Multicore Processor*, http://www.gaisler.com/index.php/products/processors/leon3 (accessed on July 21, 2016).

[307] *LEF DEF Reference Manual*, http://www.si2.org/openeda.si2.org/projects/lefdef (accessed on July 21, 2016).

[308] *Liberty Technical Advisory Board*, http://www.si2.org/Liberty-TAB (accessed on July 21, 2016).

[309] *lp_solve reference guide*, http://lpsolve.sourceforge.net/5.5 (accessed on July 21, 2016).

[310] *ARESLab*, http://www.cs.rtu.lv/jekabsons/regression.html (accessed on July 21, 2016).

[311] *MATLAB*, http://www.mathworks.com/products/matlab (accessed on July 21, 2016).

[312] *Mentor Graphics Inc.*, http://www.mentor.com (accessed on July 21, 2016).

[313] *Nefelus Design Tools*, http://www.nefelus.com/design-tools/ (accessed on July 5, 2016).

[314] *Netmaker*, http://www-dyn.cl.cam.ac.uk/~rdm34/wiki (accessed on July 21, 2016).

[315] *ORION3.0*, http://vlsicad.ucsd.edu/ORION3/ (accessed on July 21, 2016).

[316] *Openaccess API*, http://www.si2.org (accessed on July 21, 2016).

[317] *OpenSPARC T2*, http://www.oracle.com/technetwork/systems/opensparc/index.html (accessed on July 21, 2016).

[318] *OpenCores*, http://opencores.org (accessed on July 21, 2016).

[319] *Ostrich*, http://www.cadence.com/community/blogs/di/archive/2008/10/15/an-interview-with-global-timing-debug-architect-thad-mccraken.aspx (accessed on July 21, 2016).

[320] *Platform Load Sharing Facility*, http://www-03.ibm.com/systems/services/platformcomputing/lsf.html (accessed on July 5, 2016).

[321] *PSLIB Data Sets*, http://www.om-db.wi.tum.de/psplib/download.html (accessed on July 5, 2016).

[322] *Qualcomm Snapdragon*, https://www.qualcomm.com/products/snapdragon (accessed on July 5, 2016).

[323] *Random Forest*, https://code.google.com/randomforest-matlab (accessed on July 21, 2016).

[324] *RBF2 Manual*, http://www.anc.ed.ac.uk/~mjo/rbf.html (accessed on July 21, 2016).

[325] *RedHawk User Guide*, https://www.apache-da.com/products/redhawk (accessed on July 21, 2016).

[326] *Rent Parameter Evaluation Using Different Methods v2.2-2008*, http://vlsicad.ucsd.edu/WLD/RentCon.pdf (accessed on July 21, 2016).

[327] *Runtime Design Automation*, http://www.rtda.com/ (accessed on July 21, 2016).

[328] *Salesforce Project Management*, https://www.salesforce.com/ (accessed on July 5, 2016).

[329] *Samsung Exynos*, http://www.samsung.com/semiconductor/products/exynos-solution/application-processor/ (accessed on July 5, 2016).

[330] Samsung Electronics Co., Ltd. (System LSI application processor principal engineer), *personal communication*, July 2014.

[331] *Apple Siri*, http://www.apple.com/ios/siri (accessed on July 21, 2016).

[332] *SPICE*, http://bwrcs.eecs.berkeley.edu/Classes/IcBook/SPICE/ (accessed on July 21, 2016).

[333] *Standard Parasitic Exchange Format*, https://en.wikipedia.org/wiki/Standard_Parasitic_Exchange_Format (accessed on July 21, 2016).

[334] *Stanford CPUDB*, http://cpudb.stanford.edu (accessed on July 21, 2016).

[335] *Stanford NoC*, https://nocs.stanford.edu/cgi-bin/trac.cgi (accessed on July 21, 2016).

[336] *Synopsys 32/28nm Generic Library for Teaching IC Design*, http://www.synopsys.com/COMMUNITY/UNIVERSITYPROGRAM/Pages/32-28nm-generic-library.aspx (accessed on July 21, 2016).

[337] *Synopsys Design Compiler User Guide*, http://www.synopsys.com (accessed on July 21, 2016).

[338] *Synopsys Formality User Guide*, http://www.synopsys.com/tools/verification/formalequivalence/ pages/formality.aspx (accessed on July 21, 2016).

[339] *Synopsys HSPICE User Guide*, http://www.synopsys.com (accessed on July 21, 2016).

[340] *Synopsys IC Compiler*, http://www.synopsys.com (accessed on July 21, 2016).

[341] *Synopsys Interconnect Technology Format*, http://www.synopsys.com/community/interoperability/pages/tapinitf.aspx (accessed on July 21, 2016).

[342] *Synopsys PrimeTime*, http://www.synopsys.com/Tools/Implementation/SignOff/PrimeTime/Pages/default.aspx (accessed on July 21, 2016).

[343] *Synopsys Star-RCXT*, http://www.synopsys.com/Tools/Implementation/SignOff/Pages/StarRC-ds.aspx (accessed on July 21, 2016).

[344] *Synopsys SiliconSmart*, http://www.synopsys.com (accessed on July 21, 2016).

[345] *Synopsys SpyGlass*, http://www.synopsys.com/Tools/Verification/SpyGlass/default.aspx (accessed on July 21, 2016).

[346] *Synopsys VCS and DVE User Guide*, http://www.synopsys.com/tools/verification/functionalverification/pages/vcs.aspx (accessed on July 21, 2016).

[347] *Thermal Reliability*, http://wdc.com/wdproducts/library/other/2579-001134.pdf (accessed on July 21, 2016).

[348] *Tilera TILE-Gx Processor*, http://www.tilera.com/products (accessed on July 21, 2016).

[349] "*When The Chips are Down*", http://qz.com/387490/as-moores-law-turns-50-computer-chips-continue-to-get-cheaper-and-more-powerful/ (accessed on July 21, 2016).

[350] *UCSD Design Cost Optimization Solver for Multi-Tapeout Project Scheduling*, http://vlsicad.ucsd.edu/MILP/ (accessed on July 21, 2016).

[351] *UCSD Gate Sizer*, http://vlsicad.ucsd.edu/SIZING (accessed on July 21, 2016).